



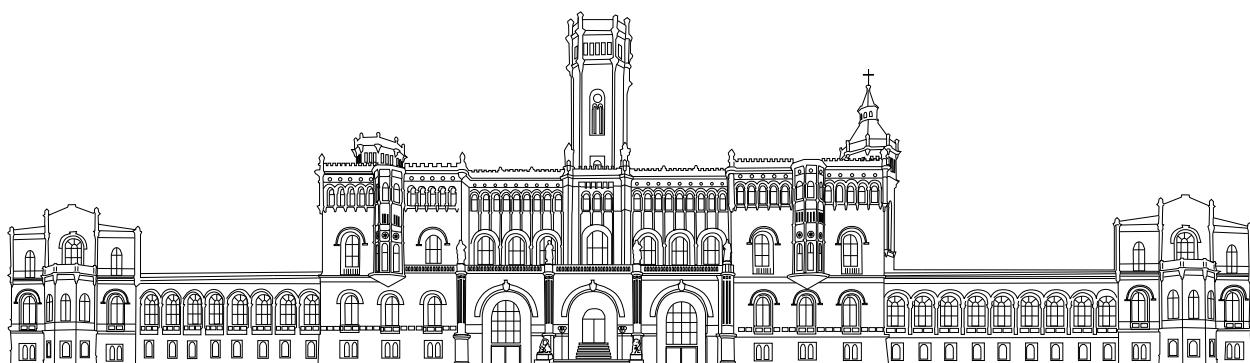
Fakultät für Elektrotechnik und Informatik

Computational Health Informatics

Entwicklung und Vergleich verschiedener Deep Learning Algorithmen zur semantischen Segmentierung eines medizinischen Datensatzes

Masterarbeit im Studiengang Informatik (M.Sc.),
eingereicht von LENNARD NÖHREN am 18.04.2019

Erstprüferin: Prof. Dr.-Ing. Gabriele von Voigt
Zweitprüfer: Dr. Daniel Lückehe
Betreuer: Fabian Pflug
Matrikelnummer: 3068460



Erklärung der Selbständigkeit

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind, und die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat.

Hannover, den 18.04.2019

Lennard Nöhren

Erklärung zur Plagiatsprüfung

Mit der Übermittlung meiner Arbeit auch an externe Dienste zur Plagiatsprüfung durch Plagiatssoftware erkläre ich mich einverstanden.

Hannover, den 18.04.2019

Lennard Nöhren

Abstract

In this thesis, a short overview of semantic segmentation and the basics of neural networks is given. Five different network architectures for semantic segmentation are presented. Two medical data sets were provided on which experiments with the different architectures were carried out. The first data set consists of endoscope images of vocal folds [27], the other of electron microscope images of mitochondria [32]. The data sets have different complexities, which made it possible to investigate the influence of the type of data on the choice of network architectures. Furthermore, the results were further improved by pre-training, augmentation and ensemble structures. On the vocal cord data set, an ensemble from the ERFNet [36], DeepLabV3+ [8] and U-Net [37] networks achieved the best result. On the mitochondrial data set, U-Net and DeepLabV3+ achieved the best results.

Zusammenfassung

In dieser Arbeit wird zunächst ein kurzer Überblick über semantische Segmentierung und die Grundlagen von neuronalen Netzen gegeben. Zudem werden fünf verschiedene Netzwerkarchitekturen für semantische Segmentierung vorgestellt. Für die Arbeit wurden zwei medizinische Datensätze zu Verfügung gestellt, auf denen Experimente mit den verschiedenen Architekturen durchgeführt wurden. Der erste Datensatz besteht aus Endoskopbildern von Stimmbändern [27], der andere aus elektronenmikroskopischen Aufnahmen von Mitochondrien [32]. Die Datensätze haben eine unterschiedliche Komplexität, wodurch untersucht werden konnte, welchen Einfluss die Art der Daten auf die Wahl der Netzwerkarchitekturen hat. Des Weiteren wurde versucht, die Ergebnisse durch Pre-Training, Augmentierungen und Ensemble Strukturen weiter zu verbessern. Auf dem Stimmband-Datensatz wurde durch ein Ensemble aus dem Netzwerk ERFNet [36], DeepLabV3+ [8] und U-Net [37] das beste Ergebnis erreicht. Auf dem Mitochondrien-Datensatz konnten mit U-Net und DeepLabV3+ die besten Ergebnisse erzielt werden.

Inhaltsverzeichnis

Zusammenfassung	iii
1 Einleitung	1
1.1 Motivation	1
1.2 Aufbau dieser Arbeit	2
2 Grundlagen	5
2.1 Semantische Segmentierung	5
2.1.1 Intersection over Union	6
2.2 Neuronale Netze	7
2.2.1 Training	9
2.2.2 Aktivierungsfunktionen	10
2.2.3 Categorical Cross-Entropy	11
2.2.4 Adam Optimizer	12
2.3 Layer Typen	13
2.3.1 Convolutional Layer	13
2.3.2 Pooling Layer	17
2.3.3 Dropout Layer	18
2.3.4 Batch Normalization	19
2.3.5 Initialisierung	20
2.4 Netzwerk Strukturen	20
2.4.1 Encoder-Decoder Struktur	21
2.4.2 Residual Blocks	22
3 Architekturen	25
3.1 U-Net	25
3.2 SegNet	27
3.3 E-Net	28
3.4 ERFNet	31
3.5 DeepLab	32
3.5.1 DeepLabV1	33
3.5.2 DeepLabV2	34
3.5.3 DeepLabV3	34
3.5.4 DeepLabV3+	35

4 Datensätze	39
4.1 Stimmband-Datensatz	40
4.1.1 Bisherige Experimente	42
4.2 Mitochondrien-Datensatz	42
4.3 Cityscapes-Datensatz	43
4.4 ImageNet-Datensatz	45
5 Fortgeschrittene Methoden	47
5.1 Pre-Training	47
5.2 Augmentierung	48
5.3 Ensemble Netzwerke	50
6 Experimente	51
6.1 Verschiedene Architekturen	52
6.1.1 Stimmband-Datensatz	53
6.1.2 Mitochondrien-Datensatz	55
6.1.3 Analyse	56
6.2 Pre-Training	58
6.2.1 Stimmband-Datensatz	58
6.2.2 Mitochondrien-Datensatz	59
6.2.3 Analyse	59
6.3 Augmentierungen	60
6.3.1 Stimmband-Datensatz	60
6.3.2 Mitochondrien-Datensatz	61
6.3.3 Analyse	61
6.4 Ensemble Netzwerke	63
6.4.1 Stimmband-Datensatz	63
6.4.2 Mitochondrien-Datensatz	64
6.4.3 Analyse	64
7 Fazit und Ausblick	67
7.1 Fazit	67
7.2 Ausblick	69

Abbildungsverzeichnis

2.1	Visualisierung der Ergebnisse einer semantischen Segmentierung des Cityscapes-Datensatzes [10].	6
2.2	Schematische Darstellung eines künstlichen Neurons.	8
2.3	Schematische Darstellung eines künstlichen neuronalen Netzes.	9
2.4	Vergleich verschiedener Optimierungsalgorithmen [24].	12
2.5	Schematische Darstellung einer 2D Convolution mit einem 3x3 Kernel und Padding [12].	13
2.6	Schematische Darstellung einer 2D Convolution mit einer Schrittweite von zwei [12].	14
2.7	Schematische Darstellung einer Dilated 2D Convolution mit einem 3x3 Kernel und dilation rate zwei [12].	15
2.8	Schematische Darstellung einer transposed 2D Convolution mit einem 3x3 Kernel und Schrittweite zwei. Dies ist äquivalent zu einer Convolution bei den Nullen zwischen den Werten der Eingabe eingefügt wurden [12].	16
2.9	Ein künstliches neuronales Netz (KNN) mit und ohne Dropout [43].	18
2.10	Eine typische Convolutional Neural Network (CNN)-Architektur. Die Blöcke sind Convolutional Layer. Zwischen den Blöcken findet Downsampling statt, wodurch die Größe reduziert wird. Das Netzwerk gibt eine Wahrscheinlichkeitsverteilung der Klassen aus [31].	21
2.11	Die Architektur aus 2.10 um einen Decoder für semantische Segmentierung erweitert [31].	21
2.12	Zwei Varianten des Residual Blocks. Links die einfache Variante bestehend aus zwei Convolutions und einer alternativen Verbindung. Rechts die tiefere bottleneck Variante mit drei Convolutions [18].	22
3.1	Die U-Net Netzwerkstruktur. Der Encoder und Decoder sind symmetrisch und durch skip connections an mehreren Stellen verbunden [37].	26
3.2	Die SegNet Netzwerkstruktur. Auch hier sind der Encoder und Decoder symmetrisch. Zum Upsampling wird Max-Unpooling eingesetzt [3].	27
3.3	Die Blöcke aus denen E-Net aufgebaut wird. (a) ist der Initiale Block, der für Downsampling eingesetzt wird. (b) ist der Residual Block aus dem das restliche Netzwerk aufgebaut ist. Wenn der Block Downsampling durchführen soll, wird Max-Pooling und Padding eingesetzt [4].	29
3.4	Die ERFNet Netzwerkstruktur [36].	31

3.5 Verschiedene Varianten von Residual Blöcken. (a) und (b) sind die beiden in [18] vorgestellten Varianten. (c) ist das Non-bottleneck-1D (Non-bt-1D) Design, dass in ERFNet zum Einsatz kommt [36].	31
3.6 Der Ablauf von DeepLabV1. Zunächst werden grobe Ergebnisse mit einer leicht veränderten Version von VGG16 erzeugt. Diese werden danach interpoliert und durch ein Conditional Random Field (CRF) verfeinert [5].	33
3.7 Atrous Spatial Pyramid Pooling (ASPP) aus DeepLabV2. Die Eingabe wird mit parallelen dilated Convolutions mit verschiedenen dilation rates verarbeitet [6].	34
3.8 Eine dilated Convolution mit dilation rate 3 auf einer 3x3 Eingabe. Der Kernel ist zu groß, wodurch sich die Convolution wie ein 1x1 Convolution verhält, da nur der zentrale Punkt des Kernels auf dem Bild liegt.	35
3.9 Der Aufbau von DeepLabV3. Die Ausgabe ist um einen Faktor von 16 kleiner als die Eingabe [7].	35
3.10 Der Aufbau von DeepLabV3+. Der Encoder entspricht dem von DeepLabV3, der in 3.9 dargestellt ist. Ein simpler Decoder wurde angefügt, um präzisere Ergebnisse zu erzeugen [8].	36
 4.1 Visualisierung der Ground Truth des Stimmband-Datensatzes und das dazugehörige Originalbild. Die Klassen wurden mit folgenden Farben versehen: <i>void</i> : grau, <i>vocal folds</i> : rot, <i>other tissue</i> : blau, <i>glottal space</i> : grün, <i>pathology</i> : lila, <i>surgical tool</i> : orange und <i>intubation</i> : gelb [27].	39
4.2 Bilder aus den verschiedenen Sequenzen des Stimmband-Datensatzes [27].	41
4.3 Bilder aus dem Trainings-, Validierungs- und Testdatensatz mit großer Ähnlichkeit [27].	41
4.4 Anzahl der Pixel der einzelnen Klassen in dem Stimmband-Datensatz [27].	42
4.5 Beispielbild und die zugehörige Ground Truth des Mitochondrien-Datensatzes. Die weißen Bereiche in der Ground Truth sind die Mitochondrien [32].	43
4.6 Ein präzise und ein grob annotiertes Bild aus dem Cityscapes-Datensatz [10].	43
4.7 Die verschiedenen Klassen und die Anzahl ihrer Pixel im Cityscapes-Datensatz [10].	44
4.8 Bilder aus drei verschiedenen Klassen des ImageNet-Datensatzes [40].	45
 5.1 Die verschiedenen Augmentierungen, die in dieser Arbeit genutzt wurden. (a) ist das originale Bild aus dem Stimmband-Datensatz. (b) wurde horizontal gespiegelt, (c) rotiert, in (d) wurde ein Ausschnitt vergrößert, (e) zeigt eine leichte elastische Deformierung und (f) eine sehr starke elastische Deformierung [27].	49
5.2 Drei Netzwerke und ihr Ensemble, die darauf trainiert wurden, Punkte im ersten und dritten Quadranten eines Bildes zu erkennen [17].	50
 6.1 Validierungs-Intersection over Union (IoU) der verschiedenen Architekturen während des Trainings auf dem Stimmband-Datensatz.	54

6.2	Ergebnisbilder nach dem Training des Stimmband-Datensatzes [27]. Die Ausgabe der Netzwerke wurde in Farbe über das Eingabebild gelegt. Grau markierte Bereiche wurden von den Netzwerken falsch klassifiziert.	55
6.3	Ergebnisbilder nach dem Training des Mitochondrien-Datensatzes [32]. Weiße Bereiche sind korrekt erkannte Mitochondrien, grau markierte Bereiche wurden von den Netzwerken falsch klassifiziert.	56
6.4	Validierungs-IoU von ERFNet und DeepLabV3+ Non-bt-1D während des Trainings auf dem Stimmband-Datensatz mit und ohne Pre-Training.	59
6.5	Ergebnisbilder aus dem Training ohne und mit der Zoom Augmentierung auf dem Stimmband-Datensatz [27].	62
6.6	Ergebnisbilder aus dem Training der einzelnen Netzwerke und ihrem Ensemble auf dem Stimmband-Datensatz [27].	65
7.1	Ergebnisbilder der besten Architektur aus dem Stimmband-Datensatz [27].	67
7.2	Ergebnisbilder der besten Architektur aus dem Mitochondrien-Datensatz [32].	68

Tabellenverzeichnis

3.1	In dieser Arbeit genutzte U-Net Implementierung. C steht für die Anzahl der Klassen.	27
3.2	In dieser Arbeit genutzte SegNet Implementierung. C steht für die Anzahl der Klassen.	28
3.3	In dieser Arbeit genutzte E-Net Implementierung. C steht für die Anzahl der Klassen.	30
3.4	In dieser Arbeit genutzte ERFNet Implementierung. C steht für die Anzahl der Klassen.	32
3.5	In dieser Arbeit genutzte DeepLabV3+ Implementierung mit dem veränderten ResNet Encoder, basierend auf den Non-bt-1D Blöcken von ERFNet. C steht für die Anzahl der Klassen.	37
6.1	Anzahl der Parameter, Trainingsgeschwindigkeit und Inferenzgeschwindigkeit in Iterationen pro Sekunde der verschiedenen Architekturen.	53
6.2	Test-IoUs der Architekturen nach dem Training auf dem Stimmband-Datensatz.	54
6.3	Test-IoUs der Architekturen nach dem Training auf dem Mitochondrien-Datensatz.	55
6.4	Test-IoUs der von ERFNet und DeepLabV3+ Non-bottleneck-1D (DL-Nbt) nach dem Training auf dem Stimmband-Datensatz und Pre-Training auf Cityscapes (Cs) und ImageNet (IN).	58
6.5	Test-IoUs der von U-Net und DL-Nbt nach dem Training auf dem Mitochondrien-Datensatz und Pre-Training auf Cityscapes (Cs) und ImageNet (IN).	59
6.6	Test-IoUs der von ERFNet und DL-Nbt nach dem Training auf dem Stimmband-Datensatz mit verschiedenen Augmentierungen.	61
6.7	Test-IoUs der von U-Net und DL-Nbt nach dem Training auf dem Augmentierten Mitochondrien-Datensatz.	61
6.8	Test-IoUs der verschiedenen Ensemble Architekturen nach dem Training auf dem Stimmband-Datensatz.	64
6.9	Test-IoUs der verschiedenen Ensemble Architekturen nach dem Training auf dem Mitochondrien-Datensatz.	64

Akronyme

ASPP Atrous Spatial Pyramid Pooling

BN Batch Normalization

CNN Convolutional Neural Network

CRF Conditional Random Field

DL-Xc DeepLabV3+ Xception

DL-RN DeepLabV3+ ResNet101

DL-Nbt DeepLabV3+ Non-bottleneck-1D

FCN Fully Convolutional Network

GPU Graphics Processing Unit

iIoU instance-level Intersection over Union

ILSVRC ImageNet Large-Scale Visual Recognition Challenge

IoU Intersection over Union

it/s Iterationen pro Sekunde

KNN künstliches neuronales Netz

Non-bt-1D Non-bottleneck-1D

PReLU Parametric Rectified Linear Unit

ReLU Rectified Linear Unit

SGD Stochastic Gradient Descent

synset synonym set

1

KAPITEL

Einleitung

1.1 Motivation	1
1.2 Aufbau dieser Arbeit	2

1.1 Motivation

Der Bereich der digitalen Bildanalyse wird aktuell von künstlichen neuronalen Netzen (KNNs) dominiert. Es werden täglich neue wissenschaftliche Arbeiten dazu veröffentlicht und die besten Ergebnisse werden regelmäßig weiter verbessert [1]. Einer der wichtigsten Bereiche der Bildanalyse ist die semantische Segmentierung. In diesem Themenbereich wurden die besten Ergebnisse ebenso durch neuronale Netze erzeugt [30].

Auch in der medizinischen Bildanalyse haben sich neuronale Netze als sehr wertvolles Werkzeug erwiesen. Allerdings hat man dort mit der, im Vergleich zur allgemeinen Bildanalyse, geringen Datenmenge zu kämpfen [30]. Denn um ein KNN effektiv zu trainieren, sind sehr große Datensätze notwendig [10]. Dadurch ist es schwerer sehr gute Ergebnisse zu erzielen und es sind besondere Verfahren notwendig, um die geringe Datenmenge auszugleichen.

Für diese Arbeit wurde ein medizinischer Datensatz mit Bildern von Stimmbändern für eine semantische Segmentierung zur Verfügung gestellt. Auf Grundlage von Laves et al. [27] sollten verschiedene Verfahren implementiert und getestet werden, um diese Aufgabe so gut wie möglich zu erfüllen. Ziel war es, die Ergebnisse von Laves et al. [27] zu reproduzieren und zu verbessern. Dafür wurden fünf verschiedene Architekturen von neuronalen Netzen implementiert und verglichen. Außerdem wurden einige Methoden, die die Ergebnisse weiter verbessern sollen, getestet.

Zusätzlich dazu wurden die Experimente auf einem weiterem medizinischen Datensatz, der aus Daten eines Elektronenmikroskops besteht, durchgeführt [32]. Dieser enthält weniger Klassen und ist allgemein simpler. Anhand des Vergleichs der Ergebnisse der

beiden Datensätze konnte untersucht werden, welchen Einfluss die Komplexität der zugrundeliegenden Daten auf die Auswahl der optimalen Verfahren hat.

Die hier genutzten Architekturen liefern bei aktuellen Wettbewerben für semantische Segmentierung sehr gute Ergebnisse [13]. Insbesondere das DeepLabV3+ Netzwerk ist dort aktuell an der Spitze der Rangliste [8]. Allerdings wurde dieses nicht für medizinische Daten entwickelt. In dieser Arbeit wurde deshalb auch untersucht, wie gut sich das DeepLab Netzwerk für medizinische Datensätze im Vergleich mit Netzwerken, die im Bereich der medizinischen Bildanalyse häufiger verwendet werden, wie zum Beispiel das U-Net [37], schlägt.

Die Ergebnisse von modernen Netzwerken können häufig durch Augmentierungen weiter verbessert werden [25, 37]. Dies gilt besonders für kleine Datensätze. Aus diesem Grund wurden in dieser Arbeit verschiedene Augmentierungsverfahren getestet und verglichen.

1.2 Aufbau dieser Arbeit

Zunächst werden alle notwendigen Grundlagen in Kapitel 2 erläutert. Dies umfasst semantische Segmentierung, neuronale Netzwerke, verschiedene Arten von Layern, die in KNNs genutzt werden und größere Netzwerkstrukturen, die man häufig wiederfindet. Dabei ist es nicht das Ziel einen gesamten Überblick über den Bereich der neuronalen Netze zu geben. Vielmehr geht es darum die Mechanismen, die in dieser Arbeit genutzt wurden, zu erklären. Daher beschränkt sich das Kapitel hauptsächlich auf Convolutional Neural Networks (CNNs), die hier genutzten Layer-Typen und typische Strukturen für Netzwerke, die für semantische Segmentierung eingesetzt werden.

In Kapitel 3 werden die fünf Netzwerkarchitekturen U-Net, SegNet, E-Net, ERFNet und DeepLab, die im Rahmen dieser Arbeit implementiert und verwendet wurden, vorgestellt. Es wird dabei ein genauer Einblick in den Aufbau der Netzwerke geliefert.

Kapitel 4 befasst sich mit den verschiedenen Datensätzen, die zum Einsatz kamen. Die Experimente wurden sowohl mit dem Stimmband- als auch mit dem Mitochondrien-Datensatz durchgeführt. So konnten die Ergebnisse für Daten mit unterschiedlichen Komplexitätsgraden verglichen werden. Des Weiteren werden in Kapitel 4 noch zwei Datensätze, die in der Forschung sehr häufig genutzt werden, beschrieben: der Cityscapes-Datensatz und der ImageNet-Datensatz. Diese wurden in dieser Arbeit lediglich für die Methode des Pre-Trainings eingesetzt.

Das 5 Kapitel beschäftigt sich mit den weiterführenden Methoden, die genutzt wurden, um die Qualität der Ergebnisse der KNNs noch weiter zu verbessern. Bei den Methoden handelt es sich um Pre-Training, Augmentierungen und Ensemble Netzwerke.

Schließlich werden in Kapitel 6 die Experimente, die im Rahmen dieser Arbeit mit den vorher vorgestellten Themen durchgeführt wurden, erläutert. Es wurde folgende Vorgehensweise realisiert: zunächst wurden die fünf Architekturen ohne Verbesserungen auf den zwei medizinischen Datensätzen trainiert. Die Ergebnisse wurden hinsichtlich mehrerer architektonischer Aspekte analysiert. Anschließend wurde die erste Erweiterung, das Pre-Training, untersucht. Danach wurden zusätzlich Augmentierungen eingesetzt,

um die Resultate weiter zu verfeinern. Dabei wurde insbesondere untersucht, welches Augmentierungsverfahren die größten Einflüsse hat. Zum Schluss wurden Ensemble-Netzwerke aufgebaut und verglichen.

Zum Abschluss der Arbeit wird in Kapitel 7 ein Fazit aus den Experimenten gezogen und ein Ausblick auf mögliche Erweiterungen gegeben.

Anmerkung: In dieser Arbeit werden häufig englische Fachbegriffe auftreten. Diese wurden absichtlich nicht ins Deutsche übersetzt, da fast die gesamte Fachliteratur in Englisch verfasst wird. Deshalb könnten Übersetzungen von häufig genutzten, englischen Begriffen zu Verwirrungen oder Verwechslungen führen.

KAPITEL 1. EINLEITUNG

2

KAPITEL

Grundlagen

2.1	Semantische Segmentierung	5
2.1.1	Intersection over Union	6
2.2	Neuronale Netze	7
2.2.1	Training	9
2.2.2	Aktivierungsfunktionen	10
2.2.3	Categorical Cross-Entropy	11
2.2.4	Adam Optimizer	12
2.3	Layer Typen	13
2.3.1	Convolutional Layer	13
2.3.2	Pooling Layer	17
2.3.3	Dropout Layer	18
2.3.4	Batch Normalization	19
2.3.5	Initialisierung	20
2.4	Netzwerk Strukturen	20
2.4.1	Encoder-Decoder Struktur	21
2.4.2	Residual Blocks	22

In diesem Kapitel werden die grundlegenden Techniken, die zum Verständnis der Arbeit nötig sind, erläutert. Zunächst wird das Verfahren semantische Segmentierung definiert. Danach folgen einige Abschnitte zu dem Thema neuronale Netze. Dabei wird zunächst allgemein erklärt wie diese funktionieren. Anschließend werden die Bausteine der Netzwerke, die so genannten Layer, vorgestellt. Zum Schluss des Kapitels werden größere Strukturen, die in solchen Netzwerken genutzt werden können, erläutert.

2.1 Semantische Segmentierung

Segmentierung bezeichnet die Unterteilung eines Bildes in separate Bereiche. Im simplen Fall wird kein Wert darauf gelegt diesen Bereichen eine Bedeutung zu geben be-



Abbildung 2.1: Visualisierung der Ergebnisse einer semantischen Segmentierung des Cityscapes-Datensatzes [10].

ziehungsweise es wird nicht versucht zu erkennen, durch welche Objekte diese Unterteilungen gebildet werden. Deshalb reicht es für eine simple Segmentierung häufig aus, Kanten- oder Flächenerkennungsalgorithmen zu nutzen [23, 46].

Bei der semantischen Segmentierung wird versucht, den einzelnen Abschnitten semantische Informationen zuzuordnen. Üblicherweise gibt es mehrere Klassen, welche den erkannten Objekten zugeordnet werden [46]. Im Gegensatz zu Objekt Detektion, wo man als Ausgabe Bounding Boxen erhält, stehen bei den Ergebnissen einer semantischen Segmentierung die genauen Konturen zu Verfügung. Für viele Verfahren sind solche präzisen Daten notwendig. Ein möglicher Nachteil ist allerdings, dass bei einem segmentierten Bild im Allgemeinen nicht die einzelnen Instanzen der Objekte voneinander getrennt werden können [46].

In der medizinischen Bildverarbeitung findet Segmentierung viele Anwendungsfälle. Dazu gehören zum Beispiel die Analyse von Gehirngewebe, computergestützte Analyse von Organen oder Segmentierung von Läsionen [30]. Ein möglicher Anwendungsfall in der Zukunft könnten auch automatisierte Roboterassistenzsysteme für Operationen sein [27].

Neuronale Netze, welche zur semantischen Segmentierung eingesetzt werden, arbeiten auf Pixel-Ebene. Das heißt, sie versuchen jedem Pixel eine Klasse zuzuordnen. Also wäre die präzisere Bezeichnung hier pixelweise Klassifizierung. Da das Gesamtergebnis jedoch das Gleiche ist wie bei einer semantischen Segmentierung, wird dieser Begriff üblicherweise auch im Kontext neuronaler Netze genutzt [3, 4, 8, 31, 36].

Abbildung 2.1 zeigt ein Ergebnis einer semantischen Segmentierung. Zur Visualisierung wurden die verschiedenen Klassen in unterschiedlichen Farben dargestellt und über das Eingabebild gelegt. Die Klasse *Auto* wird in blau dargestellt, *Personen* in rot, *Vegetation* in grün und so weiter. In dem Bild wurden die Objekte zwar schon gut erkannt, die Konturen sind zum Teil allerdings noch sehr ungenau. Das Bild stammt aus dem Cityscapes-Datensatz, auf den später noch weiter eingegangen wird [10].

2.1.1 Intersection over Union

Eine der am meisten verwendeten Metriken für die Genauigkeit der Ergebnisse einer semantischen Segmentierung ist die so genannte Intersection over Union (IoU), auch Jaccard Index genannt. Diese wird für jede Klasse berechnet und ist durch folgende

Formel definiert [10]:

$$IoU = \frac{TP}{TP + FP + FN} \quad (2.1)$$

Dabei steht TP für true positive, also die Anzahl der Pixel, welche zu der Klasse gehören und korrekt zugeordnet wurden. False positive (FP) bezeichnet die Pixel, welche der Klasse zugeordnet wurden, obwohl sie zu einer anderen gehören und false negative (FN) ist die Anzahl der Pixel, welche einer anderen Klasse zugeordnet wurden, obwohl sie zu der betrachteten Klasse gehören [10].

Um die Gesamtmetrik zu erhalten, wird die IoU für jede Klasse berechnet und der Mittelwert gebildet. Dieser Wert wird auch mean-IoU genannt.

Diese Metrik hat allerdings einen Bias zu größeren Klassen [10]. In vielen Fällen ist das nicht erwünscht, weil gerade die Klassen, die weniger Fläche einnehmen, häufig wichtig für die Anwendungen sind. Ein Beispiel sind Straßenszenen. Hier nehmen Fußgänger nur einen verhältnismäßig kleinen Bereich ein, wodurch sie von der IoU nicht so stark einbezogen werden, wie beispielsweise die Klasse für die Straße. Dieses Problem kann auch bei medizinischen Bildern auftreten, da zum Beispiel Tumore in der Regel deutlich weniger Fläche des Bildes ausmachen als das umliegende Gewebe, aber es darum geht, diese zu erkennen.

Eine mögliche Lösung für dieses Problem ist eine leichte Abwandlung namens instance-level Intersection over Union (iIoU), die folgendermaßen definiert ist:

$$iIoU = \frac{iTP}{iTP + FP + iFN} \quad (2.2)$$

Dabei stehen iTP und iFN für gewichtete Varianten von true positive und false negative. False positive wird nicht gewichtet. Die Gewichtung wird anhand der Größe der Instanzen der Klasse vorgenommen, wodurch kleinere Klassen eine größere Auswirkung auf die Metrik haben [10].

2.2 Neuronale Netze

Machine Learning kann in zwei Hauptkategorien unterteilt werden: Überwachtes Lernen und unüberwachtes Lernen. Ersteres benötigt immer einen beschrifteten Datensatz, der genutzt wird, um die Parameter des Algorithmus zu wählen [30]. KNNs sind aktuell eines der am häufigsten genutzten Verfahren aus der Kategorie überwachtes Lernen [28, 33]. Weitere beliebte Algorithmen aus dieser Kategorie sind zum Beispiel Random Forests [21] oder naive Bayes classifier [35].

KNNs sind komplexe Programmstrukturen aus künstlichen Neuronen, die auch Perzeptrons genannt werden [30]. Die ursprüngliche Idee war, die Struktur und Funktionsweise von biologischen Gehirnen zu simulieren. Deshalb ist der Aufbau der Neuronen und der Netzwerke grob an der Struktur von echten Neuronen angelehnt [46]. Die ersten mathematischen Grundlagen für Perzeptrons wurden bereits 1958 entwickelt [38].

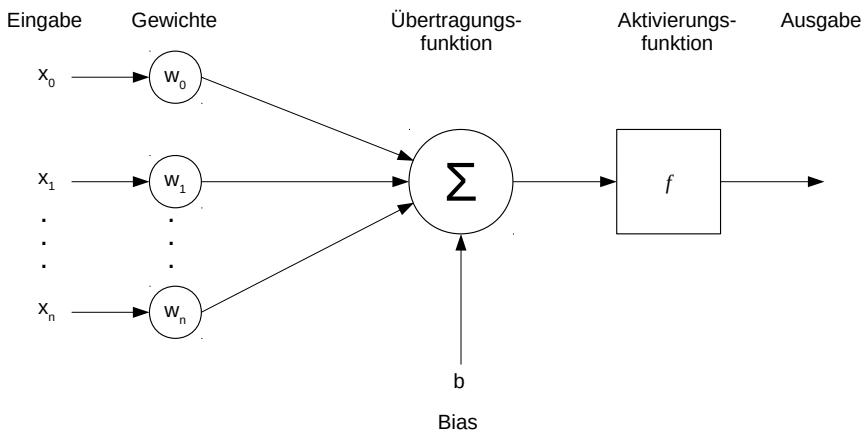


Abbildung 2.2: Schematische Darstellung eines künstlichen Neurons.

Allerdings konnten mit KNNs für lange Zeit keine guten Ergebnisse erzielt werden, was hauptsächlich daran lag, dass keine Computer mit ausreichender Rechenleistung und keine Datensätze, die groß genug für ein ausführliches Training waren, existierten [28]. In dem letzten Jahrzehnt besitzen Rechner, insbesondere durch moderne Grafikkarten, genug Leistung und Speicher, um die Nutzung von komplexen KNNs zu ermöglichen [28]. Außerdem wurden viele große Datensätze öffentlich zugänglich gemacht, wie zum Beispiel der ImageNet-Datensatz mit über 1 Million Bildern [11].

2012 wurde durch AlexNet [25] jeder andere Algorithmus bei der ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [40] deutlich geschlagen. Dadurch gewannen KNNs wieder mehr Aufmerksamkeit in der Forschung. In den folgenden Jahren wurden viele neue Architekturen und Verbesserungen entwickelt, die auf AlexNet basieren. Dadurch haben sich KNNs besonders im Bereich Computer Vision als absoluter State of the Art durchgesetzt [28].

In Abbildung 2.2 sind die Bestandteile eines klassischen künstlichen Neurons zu sehen. Es gibt mehrere Eingänge, welche jeweils ein Gewicht haben. Diese gewichteten Eingaben werden in eine Übertragungsfunktion, üblicherweise eine Summe, gegeben. Häufig wird auch ein Bias-Wert auf das Ergebnis der Funktion addiert. Der resultierende Wert wird in eine Aktivierungsfunktion gegeben [46]. Häufig genutzte Aktivierungsfunktionen sind Sigmoid Funktionen oder die so genannte Rectified Linear Unit (ReLU) Funktion [15]. Man kann ein Neuron somit durch folgende Formel darstellen:

$$o(x) = f\left(\sum_i (w_i \cdot x_i) + b\right) \quad (2.3)$$

Mit Aktivierungsfunktion f , Gewichten W , Eingabe X und Bias b .

Ein gesamtes KNN ist aus mehreren Layern von Neuronen aufgebaut. Jedes Layer besteht dabei aus mehreren parallel geschalteten Neuronen. In Abbildung 2.3 ist diese Struktur dargestellt. Üblicherweise ist jedes Neuron mit jedem Neuron aus dem vor-

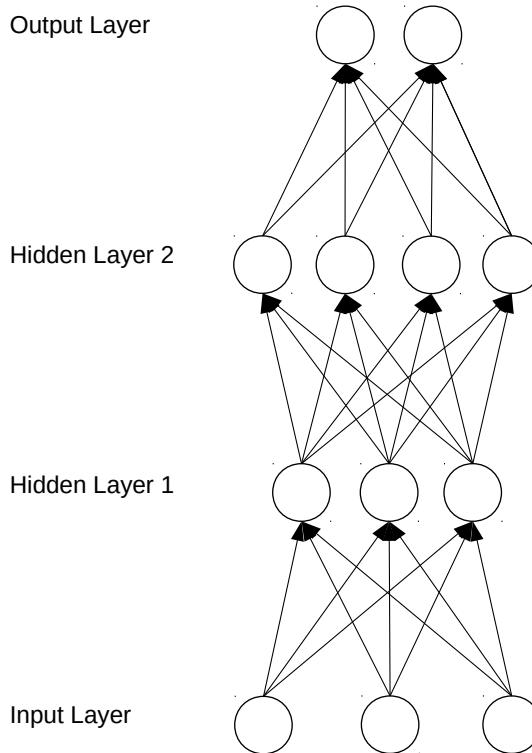


Abbildung 2.3: Schematische Darstellung eines künstlichen neuronalen Netzes.

gen Layer verbunden, weshalb man die einfachste Layer-Art auch Fully-connected Layer nennt [28].

Das erste Layer eines Netzwerkes wird häufig Input Layer genannt. Das letzte ist das Output Layer und die dazwischen liegenden sind die Hidden Layer. Die Anzahl der Hidden Layer kann je nach Architektur stark variieren. Ein KNN mit mehreren Hidden Layers wird auch Deep Neural Network genannt, woher auch der häufig verwendete Begriff Deep Learning stammt [30].

Bei dem Output Layer des Netzwerkes wird eine besondere Aktivierungsfunktion genutzt, um das gewünschte Ausgabeformat der Daten zu erhalten. Sehr beliebt ist zum Beispiel die Softmax Funktion. Dadurch wird eine Wahrscheinlichkeitsverteilung erzeugt. Daran kann zum einen erkannt werden, welcher Klasse das Netzwerk die Eingabe zuordnet. Zusätzlich wird deutlich, wie sicher sich das Netzwerk dabei ist [30].

2.2.1 Training

Neuronale Netze haben zwei Betriebsmodi. Die Inferenz und das Training. Bei der Inferenz werden dem Netzwerk unbekannte Daten übergeben und das Netzwerk soll die richtigen Ergebnisse ausgeben. Dabei werden die Parameter des KNN nicht mehr verändert. Damit korrekte Ergebnisse geliefert werden können, müssen die Parameter zunächst korrekt gewählt werden. Bei einem der Neuronen, wie sie in dem vorigen Abschnitt beschrieben wurden, sind das der Gewichtsvektor W und der Bias b . Um diese zu konfigurieren, muss das Netzwerk trainiert werden [28]. Dazu werden beschriftete Trai-

ningsdaten benötigt. Für ein Netzwerk, das eine Bildanalyse ausführen soll, wird also ein Datensatz aus Bildern mit der korrekten Klassifizierung benötigt. Dieser wird dann üblicherweise in drei Abschnitte aufgeteilt: Training, Validierung und Test [10].

Während des Trainings werden die Trainingsdaten in das Netzwerk gegeben. Anhand der Ausgabe des KNNs kann mithilfe einer Fehlerfunktion ein Fehlerwert berechnet werden. Auf Basis dieses Wertes wird eine so genannte Backpropagation durchgeführt. Dabei wird das Netzwerk rückwärts durchlaufen und für jedes Modul im Netzwerk wird ein Gradient berechnet. Mit diesen Gradienten können die Parameter der Module aktualisiert werden [28].

Dieses Verfahren wird automatisch von so genannten Optimierern durchgeführt. Heutzutage sind die am häufigsten genutzten Optimierer Stochastic Gradient Descent (SGD) und der Adam-Optimierer [24].

Üblicherweise wird, nachdem der gesamte Trainingsdatensatz einmal durch das Netzwerk gegeben wurde, ein Validierungsschritt durchgeführt. Dafür wird der Validierungsdatensatz an das Netzwerk übergeben und der Fehlerwert sowie weitere Metriken, wie zum Beispiel die IoU, berechnet. Es ist wichtig, dass sich die Validierungsdaten von den Trainingsdaten unterscheiden, um zu prüfen, ob das Netzwerk allgemeine Merkmale gelernt hat und nicht nur die genauen Strukturen der Trainingsdaten erkennen kann [27].

Ein Durchlauf der Trainingsdaten und der Validierungsdaten bilden eine Trainingsepoch. Diese Epochen werden so lange wiederholt, bis ein Abbruchkriterium erreicht wurde. Dies kann zum Beispiel eine bestimmte Anzahl an abgeschlossenen Epochen oder ein bestimmter Wert bei einer Validierungsmetrik sein. Danach wird noch einmal mit dem Testdatensatz die Qualität der Ergebnisse überprüft. Häufig werden auch hier Metriken berechnet oder visuelle Ergebnisse erzeugt, die von Menschen qualitativ untersucht werden können [27].

2.2.2 Aktivierungsfunktionen

ReLU Funktion

Die ReLU Funktion ist die am häufigsten genutzte Aktivierungsfunktionen für Neuronen in KNNs [28]. Es ist eine simple Funktion, die alle Werte kleiner Null auf Null setzt und größere Werte beibehält [15]. Sie kann folgendermaßen beschrieben werden:

$$f(x) = \begin{cases} x & : x > 0 \\ 0 & : x \leq 0 \end{cases} \quad (2.4)$$

Oder auch einfach:

$$f(x) = \max(x, 0) \quad (2.5)$$

Diese Art der Aktivierung hat einige Vorteile gegenüber anderen Aktivierungsfunktionen, die sonst genutzt werden, wie zum Beispiel die Sigmoid Funktion. Durch die ReLU Funktion können einzelne Neuronen im Netzwerk komplett abgeschaltet werden, da sie immer

eine Null ausgeben, wenn das Ergebnis der Übergangsfunktion negativ ist. Dadurch erhält man ein spärliches (sparse) Netzwerk, was laut Glorot et al. [15] vorteilhaft sein kann. Die restlichen Berechnungen in so einem Netzwerk sind linear, anders als bei der Nutzung von Aktivierungsfunktionen wie der Sigmoid Funktion, welche Nichtlinearitäten enthält. Dadurch wird die Berechnung der Gradienten vereinfacht. Außerdem ist diese Art von KNN auch näher an der biologischen Vorlage als ein voll besetztes Netzwerk [15]. Der größte Vorteil der ReLU Funktion ist jedoch, dass es damit fast immer möglich ist ein Netzwerk relativ schnell ohne unüberwachtes Pre-Training zu trainieren. Als noch andere Aktivierungsfunktionen standardmäßig genutzt wurden, war es fast immer notwendig, ein Netzwerk auf einem sehr großen Datensatz vor zu trainieren, um gute Initialisierungen der Parameter zu erhalten. Aufgrund der beschriebenen Eigenschaften der ReLU Funktion, ist dies in modernen Netzwerken meistens nicht mehr notwendig [15, 28]. Dadurch kann das Training eines Netzwerkes stark beschleunigt werden, was wiederum die Nutzung von größeren Netzwerkarchitekturen ermöglicht.

ReLU wird auch in den Netzwerken, die im Rahmen dieser Arbeit implementiert wurden, standardmäßig genutzt.

Softmax Funktion

Die Softmax Funktion wird sehr häufig als Aktivierungsfunktion für das letzte Layer von KNNs genutzt. Die Funktion erzeugt eine Wahrscheinlichkeitsverteilung. Das heißt, sie kann sehr gut genutzt werden, um bei Klassifizierungsproblemen den einzelnen Klassen Wahrscheinlichkeiten zuzuweisen. Im Gegensatz dazu, nur diskret auszugeben, welcher Klasse die Eingabe zugeordnet wurde, bietet diese Methode den Vorteil, dass die Unsicherheit des Netzwerkes erkennbar wird. Daher kann mit diesen Ergebnissen die Qualität des Netzwerkes noch besser bewertet werden [30].

Die Formel, um den Softmax Wert für Klasse i zu berechnen, lautet folgendermaßen [30]:

$$S(x) = \frac{e^{x_i}}{\sum_{j \in |x|} e^{x_j}} \quad (2.6)$$

2.2.3 Categorical Cross-Entropy

Categorical Cross-Entropy ist eine Fehlerfunktion, die häufig in Zusammenhang mit der Softmax Aktivierungsfunktionen bei der Backpropagation genutzt wird. Sie berechnet einen Fehlerwert für prozentuale Ergebnisse. Die Funktion ist folgendermaßen definiert:

$$f(p, y) = - \sum_{c \in C} y_c \log p_c \quad (2.7)$$

C ist die Menge aller Klassen. y_c ist ein binärer Wert, der angibt, ob die Klasse c korrekt ist. p_c ist die Wahrscheinlichkeit, mit der das Modell die Eingabe der Klasse c zugeordnet hat [27].

Eine häufig genutzte Erweiterung der Cross-Entropy ist eine Gewichtung. Wenn die Klassen, welche für die Anwendung besonders wichtig sind höher gewichtet werden, wird das

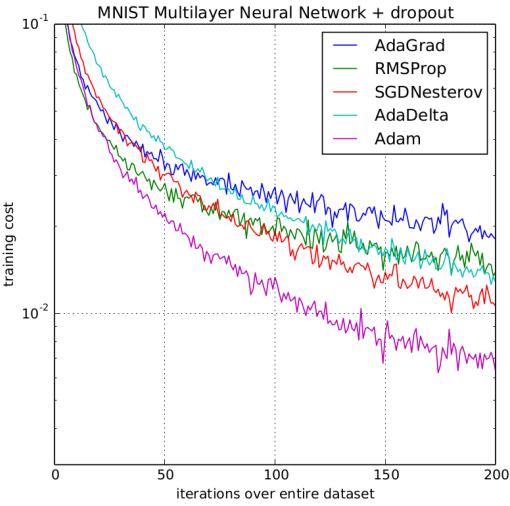


Abbildung 2.4: Vergleich verschiedener Optimierungsalgorithmen [24].

Netzwerk dazu gebracht diese Klassen stärker zu optimieren als andere. Das ist besonders sinnvoll, wenn die wichtigeren Klassen deutlich kleiner sind als andere, weil sie ohne eine Gewichtung sonst nur eine schwache Auswirkung auf die Fehlerfunktion hätten. Dadurch könnte es passieren, dass diese Klassen bei der Optimierung vernachlässigt werden [27].

2.2.4 Adam Optimizer

Adam ist ein sehr beliebter Optimierer für die Backpropagation von KNNs und wird aktuell von einigen Experten als bester Kandidat für die Wahl des Optimierers empfohlen [39]. Adam steht für “adaptive moment estimation” und ist eine Erweiterung der Algorithmen AdaGrad und RMSProp [24, 39].

Der einfachste Algorithmus, der früher meist für die Optimierung von KNNs genutzt wurde, ist SGD. Der größte Unterschied zwischen den beiden Algorithmen ist, dass es bei SGD nur eine feste Lernrate gibt, die für jeden Parameter des Netzwerkes gilt und sich im Laufe der Optimierung nicht verändert. Bei Adam werden separate Lernraten für jeden Parameter genutzt, die während der Optimierung auch verbessert werden [24]. Diese werden mithilfe des ersten und zweiten Moments der Gradienten des Netzwerks gewählt. Dafür werden gleitende Durchschnitte der Gradienten mithilfe von zwei Parametern β_1 und β_2 berechnet [24].

Adam hat noch zwei weitere Parameter: α und ϵ . α ist die initiale Lernrate, welche im Laufe der Optimierung angepasst wird. ϵ wird genutzt, um Teilung durch 0 zu vermeiden. In der originalen Veröffentlichung wurden folgende Werte für die Parameter empfohlen [24]:

$$\alpha = 1e^{-3}, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e^{-8} \quad (2.8)$$

Der größte Vorteil von Adam ist, dass die Konvergenz des Netzwerks während des Trai-

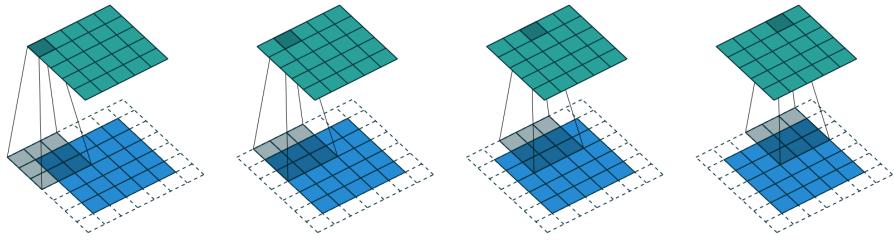


Abbildung 2.5: Schematische Darstellung einer 2D Convolution mit einem 3x3 Kernel und Padding [12].

nings deutlich schneller ist als mit anderen Optimierern [24, 39].

In Abbildung 2.4 wurde ein Netzwerk auf dem MNIST-Datensatz [29] mit verschiedenen Optimierungsalgorithmen trainiert. Dabei wurden unter anderem auch AdaGrad und RMSProp genutzt, welche sehr ähnlich zu Adam sind [24]. Es ist zu sehen, dass Adam deutlich weniger Iterationen benötigt, um bessere Ergebnisse zu erzielen als alle anderen getesteten Algorithmen.

Aufgrund dieser Ergebnisse wurde der Adam Optimierer auch für alle Trainings, die im Rahmen dieser Arbeit durchgeführt wurden, genutzt.

2.3 Layer Typen

Die in Abschnitt 2.2 vorgestellten Neuronen bilden die grundlegende Struktur für Layer eines KNNs. Es gibt allerdings sehr viele verschiedene Typen von Neuronen, aus denen andere Arten von Layern aufgebaut werden können. Einige von den am häufigsten verwendeten und für diese Arbeit relevanten Typen von Layern werden in den folgenden Abschnitten vorgestellt.

2.3.1 Convolutional Layer

Eine der wichtigsten Layerarten für die Bildanalyse ist das so genannte Convolutional Layer. Netzwerke, die aus diesen Layern bestehen, werden CNN genannt. Im Bereich der semantischen Segmentierung und Computer Vision allgemein werden fast ausschließlich diese Art von Netzwerken genutzt [28].

In Convolutional Layern ist die gewichtete Summe durch eine Convolution mit einer Kernel Matrix ersetzt. Eine diskrete 2D Convolution zwischen einem Kernel K mit der Größe $(2w + 1 \times 2h + 1)$ und der Eingabe I ist folgendermaßen definiert [44]:

$$(K * I)(x, y) = \sum_{a=-w}^w \sum_{b=-h}^h K(a, b) \cdot I(x - a, y - b) \quad (2.9)$$

Auf diese Weise wird für jeden Punkt (x, y) aus I der neue Wert berechnet. In Abbildung 2.5 ist eine schematische Darstellung einer solchen Convolution zu sehen. Die weißen

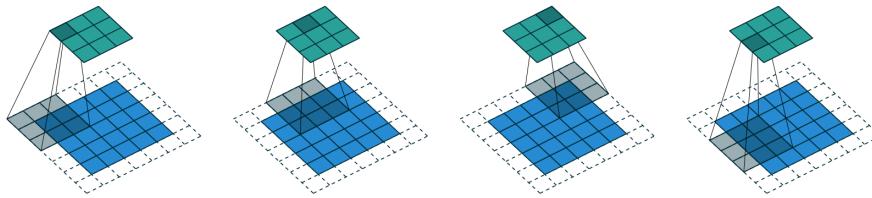


Abbildung 2.6: Schematische Darstellung einer 2D Convolution mit einer Schrittweite von zwei [12].

Punkte der Eingabe wurden durch ein Padding hinzugefügt. Das ist nötig, wenn die Ausgabe die gleiche Größe haben soll wie die Eingabe und eine Kernel Matrix verwendet wird, die größer als 1×1 ist [12].

Convolutions können auch genutzt werden, um die Größe der Eingabe absichtlich zu reduzieren. Dazu kann die Schrittweite, mit der der Kernel über die Eingabe verschoben wird, erhöht werden. Wird zum Beispiel eine Schrittweite von zwei genutzt, wird die Größe in x und y Richtung halbiert, da nur für jeden zweiten Pixel ein neuer Wert berechnet wird [31]. Das ist in Abbildung 2.6 dargestellt.

Ein Convolutional Neuron kann also folgendermaßen beschrieben werden [30]:

$$o(x) = f(w * x + b) \quad (2.10)$$

Dabei ist f die Aktivierungsfunktion des Neurons, w der Kernel, b der Bias und $*$ eine diskrete Convolution.

Mit diesem Verfahren wird nicht mehr für jeden Punkt der Eingabe ein Parameter benötigt, wie es bei den Layern, die in 2.2 beschrieben wurden, der Fall war. Stattdessen sind die Werte in der Kernel Matrix die trainierbaren Parameter des Layers. Das hat zwei Vorteile. Zum einen hängt die Anzahl der Parameter eines CNN nicht mehr von der Größe der Eingabe ab, was für hochauflösende Bilder eine extreme Senkung im Vergleich zu einem klassischen KNN bedeutet [30]. Zum anderen kann das Netzwerk ein gelerntes Feature an einer beliebigen Position im Bild erkennen, da die Kernel Matrix bei der Convolution über das gesamte Bild wandert [28, 30].

In Bilddaten sind außerdem häufig lokal nah beieinander liegende Punkte sehr eng miteinander verbunden. Da bei Convolutions immer die direkte Umgebung jedes Punktes mit betrachtet wird, kann diese Eigenschaft sehr gut genutzt werden, um Features zu erkennen [28].

Auch Convolutional Layer bestehen aus mehreren Neuronen. Dabei kann jedes Neuron ein anderes Feature erkennen, da sie unterschiedliche Kernels haben. Deshalb spricht man auch von mehreren Feature Maps in jedem Layer [28].

Dilated Convolutions

Im Zusammenhang von CNNs spricht man häufig von einem Rezeptiven Feld [5, 6, 30, 31, 50]. Dieses gibt an, wie groß der Bereich ist, der von dem Netzwerk auf ein Mal

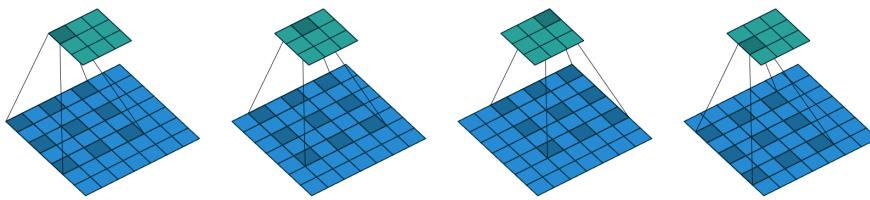


Abbildung 2.7: Schematische Darstellung einer Dilated 2D Convolution mit einem 3x3 Kernel und dilation rate zwei [12].

betrachtet werden kann. Das bedeutet, ein CNN mit einem größeren Rezeptiven Feld kann auch größere zusammenhängende Features in den Bildern erkennen. Deshalb ist ein wichtiges Kriterium beim Entwurf von Netzwerken die Größe des Rezeptiven Feldes [30]. Je nach Anwendungsbereich kann es sinnvoller sein ein sehr großes Rezeptives Feld zu nutzen, um große Features besser erkennen zu können oder die Größe klein zu halten, um sich besser auf kleinere Features konzentrieren zu können.

Um das Rezeptive Feld zu vergrößern, gibt es drei übliche Methoden. Der simpelste Ansatz ist, die Kernel Größe von Convolutional Layern zu erhöhen. In älteren Architekturen kommen deshalb häufig einige Layer mit einer Kernel Größe von 5x5 oder größer vor [5, 25, 31, 42]. Allerdings sind diese großen Convolutions extrem rechenaufwändig, wodurch die CNNs mit dieser Architektur deutlich langsamer werden [5]. In moderneren Architekturen werden daher hauptsächlich 3x3 oder 1x1 Convolutions genutzt [6].

Eine weitere sehr häufig genutzte Methode um das Rezeptive Feld eines CNNs zu vergrößern sind Downampling Layer. Dabei wird die Größe der Feature Maps der Layer reduziert. Üblicherweise werden dafür entweder Convolution Layer mit einer Schrittweite von zwei oder Pooling Layer benutzt. Letztere werden in Abschnitt 2.3.2 im Detail vorgestellt.

Eine neuere Methode sind die dilated (auch atrous) Convolutions. Dabei wird in die Formel für die Convolution eine so genannte dilation rate r eingefügt [6, 50].

$$(K *_r I)(x, y) = \sum_{a=-w}^w \sum_{b=-h}^h K(a, b) \cdot I(x - r \cdot a, y - r \cdot b) \quad (2.11)$$

Eine normale Convolution kann durch den Sonderfall $r = 1$ erreicht werden [6, 50].

In Abbildung 2.7 ist schematisch dargestellt wie diese Form der Convolution funktioniert. Dadurch, dass die einbezogenen Punkte über einen größeren Bereich verteilt sind, kann mithilfe der dilated Convolutions das Rezeptive Feld des CNNs vergrößert werden [50]. Dabei gibt es, im Gegensatz zu einer normalen Convolution mit der gleichen Kernel Größe, keinen größeren Rechen- oder Speicheraufwand [6]. Auch hier kann ein Padding an den Rändern der Eingabe hinzugefügt werden, damit die Ausgabe die gleiche Größe behält wie die Eingabe.

Häufig werden mehrere dilated Convolutions mit ansteigender dilation rate hintereinander geschaltet [7, 50].

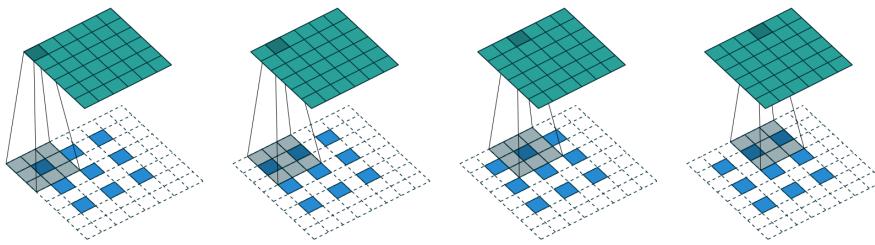


Abbildung 2.8: Schematische Darstellung einer transposed 2D Convolution mit einem 3x3 Kernel und Schrittweite zwei. Dies ist äquivalent zu einer Convolution bei der Nullen zwischen den Werten der Eingabe eingefügt wurden [12].

Transposed Convolutions

In den meisten CNNs ist es üblich, dass es mehrere Downsampling Schritte gibt, um die Menge der Daten, die von dem Netzwerk verarbeitet werden müssen zu reduzieren und damit die Laufzeit und Speicher Kosten zu verbessern [28]. Außerdem kann dadurch das Rezeptive Feld des Netzwerkes vergrößert werden. Dafür werden üblicherweise entweder Pooling Layer oder Convolution Layer mit einer Schrittweite von zwei genutzt [28, 36]. Bei der semantischen Segmentierung muss die Ausgabe des Netzwerkes allerdings wieder so groß sein wie die ursprüngliche Eingabe. Dafür wurden in der Vergangenheit verschiedene Ansätze ausprobiert, wie zum Beispiel das Zusammensetzen von Ergebnissen mit verschobenen Eingaben (shift-and-stitch) [31]. Allerdings sind diese Verfahren sehr rechenaufwändig und liefern in der Regel keine hochauflösenden Ergebnisse. In den meisten aktuellen Netzwerkarchitekturen ist die übliche Methode, dass hinter das Netzwerk ein Decoder gehängt wird, der die Daten wieder auf die originale Größe erweitert [3, 4, 8, 36, 37].

Für das Upsampling, das in einem solchen Decoder durchgeführt werden muss, gibt es auch verschiedene Methoden. Dazu gehören simples Bilineares Upsampling [8], eine inverse Operation zum Pooling [3] oder auch die so genannten transposed Convolutions. Andere Namen dafür sind Deconvolution, backwards Convolution oder fractionally strided Convolutions [12, 31, 37].

Eine transposed Convolution kann als eine Convolution, die rückwärts durchgeführt wird, betrachtet werden [12]. Aus mathematischer Sicht können Convolutions auch durch eine Matrixmultiplikation mit einer spärlich besetzten Matrix C dargestellt werden. Eine transposed Convolution kann einfach durch die Matrixmultiplikation mit C^T beschrieben werden, woher auch der Name stammt [12].

Manchmal wird diese Operation auch als Convolution mit einer fraktionierten Schrittweise beschrieben [31]. Nutzt man also eine Schrittweite von zwei in einer transposed Convolution entspricht das einer normalen Convolution mit einer Schrittweite von $\frac{1}{2}$ [12]. Das heißt, dass die Größe der Eingabe dieser transposed Convolution in x und y Richtung verdoppelt wird.

Das entspricht einer Convolution, bei der Null-Punkte zwischen jedem Punkt der Eingabe eingefügt werden. Dies ist in Abbildung 2.8 zu sehen [12].

2.3.2 Pooling Layer

Wie bereits im vorigen Abschnitt erwähnt, werden in CNNs üblicherweise einige Down-sampling Layer eingefügt, um die Speicher- und Rechenkosten des Netzwerkes zu reduzieren. Außerdem ist es dadurch möglich, größere zusammenhängende Features zu erkennen, ohne größere Kernels in den Convolutional Layers zu nutzen. Dadurch wird also insgesamt das Rezeptive Feld der Netzwerke vergrößert. Ein weiterer Vorteil ist, dass die Netzwerke eine größere Invarianz gegenüber Translationen erhalten [28].

Eine der am häufigsten verwendeten Methoden zum Downsampling sind Pooling Layer. Bei dieser Art der Layer wird, ähnlich wie bei Convolutional Layern, ein Fenster über die Eingabe verschoben und die neuen Ergebnisse werden anhand der Werte unter dem Fenster berechnet. Üblicherweise wird entweder das Maximum (Max-Pooling) oder der Durchschnitt (Average-Pooling) der Werte in dem betrachteten Bereich gebildet [12].

Meistens wird eine Schrittweite von zwei genutzt, wodurch die Größe der Eingabe in x und y Richtung halbiert wird. Eine Visualisierung hiervon würde äquivalent zu Abbildung 2.6 aussehen.

Pooling Layer haben keine Parameter, die trainiert werden müssen und können relativ schnell berechnet werden. Deshalb sind sie sehr beliebt und werden in sehr vielen CNN-Architekturen verwendet [3, 4, 8, 25, 31, 37]. Eine Alternative, welche auch in einigen Netzwerken genutzt wird, sind Convolutional Layer mit einer Schrittweite von zwei [4, 9, 18, 36]. Allerdings haben diese strided Convolutions einen deutlich höheren Rechenaufwand als Pooling Layer und benötigen zusätzliche Parameter.

Max-Unpooling

Ähnlich wie die transposed Convolutions wurde auch für Pooling Layer eine inverse Variante entwickelt, die genutzt werden kann, um ein Upsampling durchzuführen. Das so genannte Max-Unpooling wurde zum ersten Mal im SegNet verwendet [3].

Um ein Max-Unpooling zum Upsampling durchzuführen, werden die Indizes der Werte, die beim Downsampling durch ein Max-Pooling beibehalten wurden, benötigt. Also die Indizes von den Maxima der jeweils betrachteten Bereiche. Mithilfe dieser Indizes werden beim Upsampling die Eingabewerte zurück an die gleichen Positionen wie vor dem Downsampling gesetzt. Die restlichen Punkte werden mit Nullen aufgefüllt. Dadurch entsteht eine spärlich besetzte Ausgabe. Danach wird üblicherweise ein Convolution Layer eingefügt, damit die Null-Punkte mit validen Punkten gefüllt werden können [3].

Durch die Downsampling Schritte zu Beginn eines Netzwerkes gehen viele Informationen verloren. Dadurch ist es schwieriger die exakten Konturen von Objekten zu rekonstruieren [31]. Eine häufig genutzte Idee ist es, so genannte skip connections in ein Netzwerk einzubauen, die Informationen von einem höheren Level des Netzwerkes weitergeben [31, 37].

Auf dieser Idee basiert auch das Max-Unpooling, allerdings ist hier ein zusätzliches Ziel, den Speicheraufwand der skip connections zu minimieren. Dadurch, dass nur die Indizes der Pooling Layer gespeichert werden müssen, ist die Datenmenge, die durch das Netzwerk gegeben werden muss, sehr viel geringer, als wenn die gesamten Feature Maps

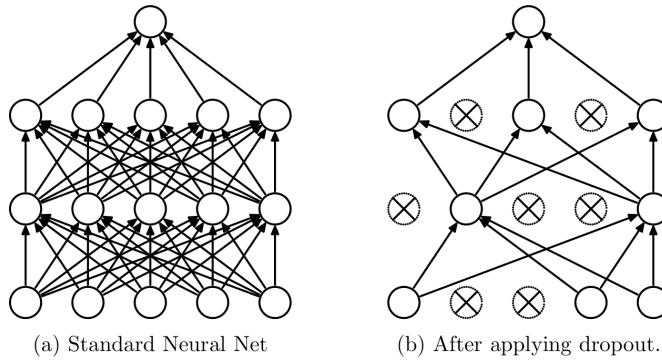


Abbildung 2.9: Ein KNN mit und ohne Dropout [43].

verwendet werden würden [3]. Trotzdem erhält der Decoder so einige Informationen aus den höheren Layern, wodurch präzisere Ergebnisse ermöglicht werden sollen.

2.3.3 Dropout Layer

Ein sehr häufig auftretendes Problem beim Training von KNNs ist Overfitting. Dabei steigt die Trainingsgenauigkeit des Netzwerkes immer weiter an, aber die Genauigkeit bei der Validierung stagniert oder sinkt sogar. Die Ursache dafür ist, dass das Netzwerk zu spezifische Informationen der Trainingsdaten lernt, anstatt allgemeine Features zu erkennen [20, 43].

Um dieses Problem zu umgehen, wurde lange Zeit ein Ensemble von Netzwerken als beste Lösung angesehen [43]. Die Netzwerke werden unabhängig voneinander trainiert und haben somit unterschiedliche Parameter. Dadurch ist es sehr unwahrscheinlich, dass sich alle Netzwerke zu stark auf die gleichen Informationen aus den Trainingsdaten spezialisiert haben, wodurch eine Kombination der Netzwerke besser generalisieren kann. Allerdings ist das Training von mehreren Netzwerken extrem zeitaufwändig und auch die Laufzeit bei der Inferenz ist deutlich langsamer als von einem einzelnen Netz.

Ein Ansatz, der seit seinem ersten Einsatz im AlexNet extrem beliebt ist, heißt Dropout [25]. Dabei wird bei den Layern ein prozentualer, zufälliger Anteil der Ausgaben der Neuronen auf Null gesetzt. Nach dem Training wird der Dropout deaktiviert und die Ausgabe des Layers wird mit der dropout rate multipliziert, damit die Ergebnisse des Netzwerkes dadurch nicht negativ beeinträchtigt werden [43].

Durch dieses Verfahren werden in jedem Trainingsschritt unterschiedliche Teile des Netzwerkes deaktiviert, was letztendlich eine ähnliche Wirkung hat, als würden mehrere dünnere Varianten des Netzwerkes gleichzeitig trainiert werden [20, 43]. Das ist gut in Abbildung 2.9 zu sehen. In (b) wurde zu dem Netzwerk Dropout hinzugefügt, wodurch sich die Struktur deutlich verändert hat.

Außerdem müssen die Neuronen eines Layers durch Dropout immer mit einem unterschiedlichen Satz an anderen Neuronen zusammenarbeiten. Dadurch müssen sie robuster sein und allgemeinere Features erkennen [20].

Eine Erweiterung von Dropout, die insbesondere für CNNs entwickelt wurde, ist der Spatial Dropout [47]. Dabei werden nicht nur zufällige Ausgaben von einzelnen Neuronen

auf Null gesetzt, sondern gesamte Feature Maps eines Layers. Der Hintergrundgedanke dabei ist, dass räumlich nah beieinander liegende Punkte in Bildern sehr eng zusammenhängen. Deshalb reicht es nicht aus einzelne Punkte zu deaktivieren, sondern es ist nötig größere Bereiche auszuschalten, um einen signifikanten Unterschied zu erreichen [47].

2.3.4 Batch Normalization

Bei den üblichen Verfahren, die für das Training von KNNs genutzt werden, wie zum Beispiel SGD oder Adam, werden häufig so genannte Mini-Batches genutzt [22, 24]. Das heißt, dass in jedem Schritt nicht nur eine Eingabe an das Netzwerk übergeben wird, sondern ein Batch, bestehend aus mehreren Eingaben. Das hat zwei Hauptgründe. Zum einen kann der Gradient eines Batches als Repräsentation des Gradienten des gesamten Trainingssatzes angesehen werden. Je größer die Batchgröße gewählt wird, desto präziser ist diese Repräsentation. Das bedeutet, in jedem Trainingsschritt kann die Optimierung der Parameter besser an den gesamten Datensatz angepasst werden, anstatt nur an ein spezifisches Beispiel. Außerdem ermöglicht die Nutzung von Batches einen höheren Grad an Parallelisierung, was zu einer Beschleunigung im Vergleich zum Training mit einzelnen Eingaben führen kann [22]. Allerdings ermöglicht die Nutzung von Batches zusätzlich auch noch den Einsatz eines neuen Layer Typs. Dem so genannten Batch Normalization (BN) Layer.

BN wurde entwickelt, um einem Problem namens Internal Covariate Shift entgegen zu wirken. Jedes interne Layer L des Netzwerkes erhält die Ausgabe des vorigen Layers L_{-1} als Eingabe. Wenn sich während des Trainings die Parameter des Layers L_{-1} ändern, ändern sich damit die Eigenschaften der Eingabe des Layers L . Dadurch muss sich das Layer L zunächst wieder an die neuen Eigenschaften anpassen, was das Training verlangsamt. Je tiefer ein Netzwerk ist, desto stärker werden die späteren Layer von diesem Phänomen beeinträchtigt [22].

BN soll dieses Problem reduzieren, indem die Eingabe von jedem Layer normalisiert wird, wodurch Änderungen am vorigen Layer nicht so starke Auswirkungen haben. Dies wird folgendermaßen erreicht [22] :

$$BN_{\gamma, \beta}(x_i) = \gamma \cdot \left(\frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \beta \quad (2.12)$$

μ_B ist der Mittelwert des Batches und σ_B^2 ist die Varianz des Batches. γ und β sind trainierbare Parameter des BN Layers. Diese Parameter sind notwendig, damit das Netzwerk durch die BN Layer nicht negativ beeinträchtigt werden kann. Denn wird $y = \sqrt{\sigma_B^2 + \epsilon}$ und $\beta = \mu_B$ gewählt, wird das BN Layer zur Identität. Wenn das die optimale Vorgehensweise wäre, könnte das Layer also sogar im Training deaktiviert werden [22].

Das Reduzieren des Covariate Shifts ermöglicht BN ein deutlich schnelleres Training. Außerdem ist es laut Ioffe et al. [22] möglich, größere Lernraten zu verwenden. Des Weiteren reduziert BN Overfitting, wodurch man weniger Dropout in den Netzwerken

benötigt, was das Training noch weiter beschleunigen kann [22].

2.3.5 Initialisierung

Jeder trainierbare Parameter eines Layers muss zu Beginn des Trainings initialisiert werden. Die Art der Initialisierung kann große Auswirkungen auf die Geschwindigkeit des Trainings und sogar auf die Qualität der endgültigen Präzision des Netzwerkes haben. Deshalb gibt es viele verschiedene Initialisierungsmethoden.

Ein Problem, das durch schlechte Initialisierung auftreten kann, ist das Vanishing Gradient Problem. Dabei werden die Gradienten einzelner Layer sehr klein und unterscheiden sich nur noch durch so geringe Werte, dass die Optimierung sehr schwer wird [14, 28].

Einer der am häufigsten verwendeten Initialisierer ist der Glorot Initialisierer. Dieser wählt die Werte für die Initialisierung der Parameter aus einer Gleichverteilung mit folgenden Grenzen aus:

$$[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}] \quad (2.13)$$

Dabei ist n_j die Anzahl der eingehenden Verbindungen in das Layer und n_{j+1} die Anzahl der ausgehenden Verbindungen [14].

Bei besonders tiefen Netzwerken kann es allerdings sein, dass auch mit dem Glorot Initialisierer die Optimierung nicht gut möglich ist. Deshalb ist eine weitere Methode zur Initialisierung der Parameter, ein skalierbarer Initialisierer. Diese Methode wurde in einigen Netzwerken dieser Arbeit genutzt. Dabei werden die Werte aus einer Normalverteilung mit einer folgendermaßen definierten Standardabweichung gezogen:

$$\sigma = \sqrt{\frac{s}{n_j}} \quad (2.14)$$

Der Wert für s wird dabei im Laufe des Netzwerkes reduziert. In dem ersten Layer beträgt er 1.0 und für fortschreitende Layer wird er jeweils um einen Faktor von 0.75 reduziert. Diese Methode der Initialisierung ist besonders gut für so genannte Residual Networks, welche im weiteren Verlauf der Arbeit noch vorgestellt werden, geeignet [16].

2.4 Netzwerk Strukturen

Es gibt zwar sehr viele verschiedene Netzwerkarchitekturen, aber einige typische Strukturen treten immer wieder auf. Dazu gehört zum Beispiel der typische Aufbau eines CNNs. Dabei werden üblicherweise abwechselnd Convolution Blöcke und downsampling Layer geschaltet [28]. Für das Downsampling wird am häufigsten Max-Pooling verwendet, aber strided Convolutions sind auch beliebt. Meistens wird die Eingabe auf diese Weise insgesamt 16- oder 32-Fach verkleinert, bevor das Softmax Layer eine Wahrscheinlichkeitsverteilung für die Klassen erzeugt [7].

In Abbildung 2.10 ist eine solche Architektur dargestellt. Die Größe der Feature Maps

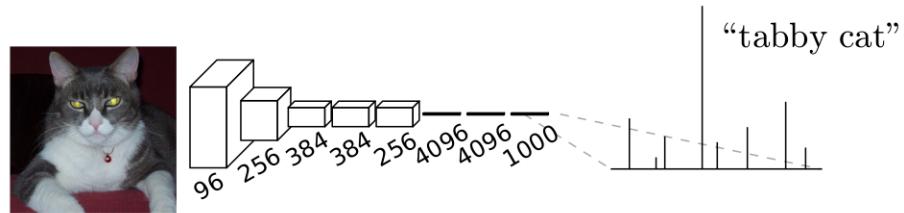


Abbildung 2.10: Eine typische CNN-Architektur. Die Blöcke sind Convolutional Layer. Zwischen den Blöcken findet Downsampling statt, wodurch die Größe reduziert wird. Das Netzwerk gibt eine Wahrscheinlichkeitsverteilung der Klassen aus [31].

der Convolutional Layer wird im Laufe des Netzwerks reduziert, aber ihre Anzahl wird gleichzeitig erhöht [31]. Auf diese Weise können die Laufzeit- und Speicherkosten eines Netzwerkes gering genug gehalten werden, um ein effektives Training mit großen Bilddaten zu ermöglichen [30].

Außerdem ermöglicht diese Struktur einen typischen Ablauf in dem Netzwerk: die ersten Layer erkennen sehr kleine Strukturen, wie Kanten oder Punkte. Die tieferen Layer des Netzwerkes können Features, die über größere Bereiche verteilt sind, aus den vorher erkannten Strukturen zusammensetzen [28].

2.4.1 Encoder-Decoder Struktur

Bei der semantischen Segmentierung gibt es etwas andere Anforderungen an die Netzwerke. Die Ausgabe des CNN soll die gleiche Größe haben wie die Eingabe. Deshalb ist aktuell eine Encoder-Decoder Struktur am beliebtesten. Zum ersten Mal wurde diese Art des Netzwerkes bei dem Fully Convolutional Network (FCN) eingesetzt [31]. Seitdem haben viele andere Netzwerkarchitekturen darauf aufgebaut [3, 8, 36, 37].

Die Idee ist es ein normales Klassifizierungs-CNN als Encoder (auch Feature Extractor genannt) zu nutzen. Das Softmax Layer, welches die Zuweisung zu den Klassen durchgeführt hat, wird entfernt und stattdessen wird einen Decoder, der die Features wieder auf die originale Größe hoch skaliert, angefügt. In Abbildung 2.11 ist zu sehen, wie die Ar-

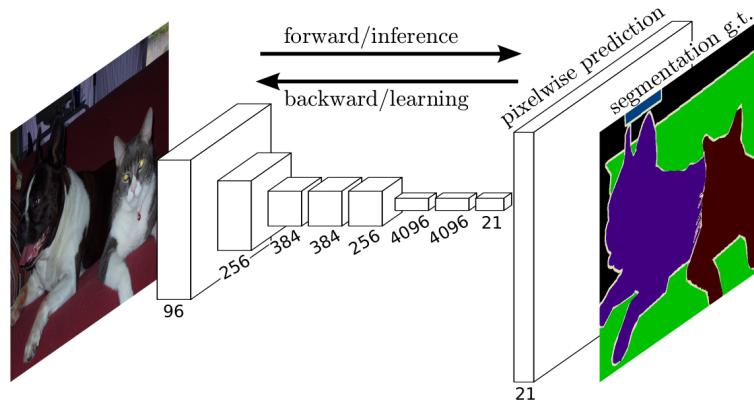


Abbildung 2.11: Die Architektur aus 2.10 um einen Decoder für semantische Segmentierung erweitert [31].

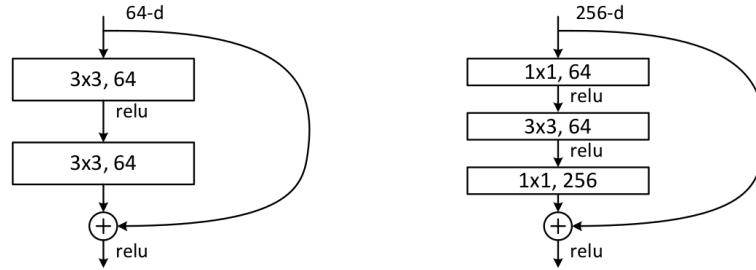


Abbildung 2.12: Zwei Varianten des Residual Blocks. Links die einfache Variante bestehend aus zwei Convolutions und einer alternativen Verbindung. Rechts die tiefere bottleneck Variante mit drei Convolutions [18].

chitektur aus 2.10 erweitert werden könnte, um sie für eine semantische Segmentierung zu nutzen [31].

Für den Decoder gibt es viele verschiedene Ansätze. Diese reichen von simplen bilinearem Upsampling in einem Schritt auf die Originalgröße [7], über komplexere Kettung von Upsampling Layern [8, 31, 36], bis zu Decodern, die den gesamten Encoder spiegeln und für jedes Layer darin ein entsprechendes Layer enthalten [3, 37]. Einige verschiedene Architekturen und deren Decoder werden in Kapitel 3 genauer vorgestellt.

2.4.2 Residual Blocks

Es hat sich gezeigt, dass CNNs mit mehr Layern häufig bessere Ergebnisse liefern als kleinere Netzwerke. Allerdings gilt das bei klassischen Netzwerken nur bis zu einer gewissen Tiefe. Ab einer bestimmten Größe bricht die Qualität der Ergebnisse des Netzwerkes ein. Dieses Problem nennt sich Degradation Problem [18].

Das Degradation Problem ist sehr kontraintuitiv, da es immer möglich sein sollte ein kleineres Netzwerk durch ein größeres zu simulieren, indem die zusätzlichen Layer als Identität gewählt werden. Nach dieser Logik sollte ein größeres Netzwerk immer mindestens genauso gut sein wie ein kleineres. Das Degradation Problem tritt vermutlich auf, da es mit aktuellen Methoden nicht möglich ist ein sehr großes Netzwerk immer optimal zu trainieren [18].

Um dieses Problem zu umgehen, wurden Residual Blöcke entwickelt. Die Idee dahinter ist, dem Netzwerk eine leichtere Möglichkeit zu geben Layer zur Identität umzuwandeln, um die Optimierung von sehr tiefen Netzwerken zu vereinfachen. Das wird erreicht, indem eine alternative Verbindung in das Netzwerk eingebaut wird, mit der die Layer des Residual Blocks übersprungen werden können. So kann das Netzwerk diese Layer sehr leicht ausschalten, falls das im Training die optimale Vorgehensweise ist [18].

In Abbildung 2.12 sind zwei Varianten eines Residual Blocks dargestellt. Links ist die einfachste Variante, bestehend aus zwei 3x3 Convolutions und einer alternativen Verbindung. Die rechte Variante ist dafür da, um noch tiefere Netzwerke zu ermöglichen. Sie besteht aus einer 3x3 Convolution, die von zwei 1x1 Convolutions umgeben ist. Die erste Convolution hat dabei die Aufgabe die Anzahl der Filter zu reduzieren und die dritte Convolution erhöht die Anzahl der Filter wieder auf den ursprünglichen Wert. Der Sinn

von diesem Aufbau ist, dass die 3x3 Convolution weniger Filter hat, wodurch die Laufzeit dieses Blocks um ein Vielfaches verbessert wird. Damit ist es möglich sehr viel tiefere Architekturen zu nutzen, ohne die Laufzeit- und Speicherkomplexität zu stark zu erhöhen und ohne auf das Degradation Problem zu stoßen. Dieser Block wird auch bottleneck Block genannt und hat in dem ResNet101 Netzwerk sehr gute Ergebnisse geliefert [18]. Seitdem werden diese Art der Blöcke auch in anderen Architekturen häufig genutzt [6, 36].

3

KAPITEL

Architekturen

3.1	U-Net	25
3.2	SegNet	27
3.3	E-Net	28
3.4	ERFNet	31
3.5	DeepLab	32
3.5.1	DeepLabV1	33
3.5.2	DeepLabV2	34
3.5.3	DeepLabV3	34
3.5.4	DeepLabV3+	35

In den folgenden Abschnitten werden die verschiedenen Netzwerkarchitekturen, die in dieser Arbeit genutzt wurden, vorgestellt.

3.1 U-Net

U-Net wurde für die Segmentierung von medizinischen Bildern entwickelt. Es baut auf der FCN-Architektur [31] auf und nutzt eine Encoder-Decoder Struktur. Ein Ziel war es ein Netzwerk zu entwickeln, das besonders gut mit kleineren Datensätzen arbeiten und trotzdem präzise Resultate erzeugen kann, da im Bereich der medizinischen Bildverarbeitung häufig nur sehr geringe Datenmengen zur Verfügung stehen. Das Besondere hier ist, dass der Decoder genau symmetrisch zu dem Encoder aufgebaut ist. In Abbildung 3.1 ist die Architektur dargestellt [37].

Encoder und Decoder bestehen jeweils aus vier Blöcken. Jeder Block enthält drei 3x3 Convolutional Layer. Außerdem gibt es einen weiteren zentralen Block mit zwei Convolutions zwischen dem Encoder und Decoder. Im Encoder wird 2x2 Max-Pooling zum Downampling benutzt. Zum Upsampling im Decoder werden 2x2 transposed Convolutions genutzt. Die Anzahl der Feature Maps wird beim Downsampling verdoppelt und

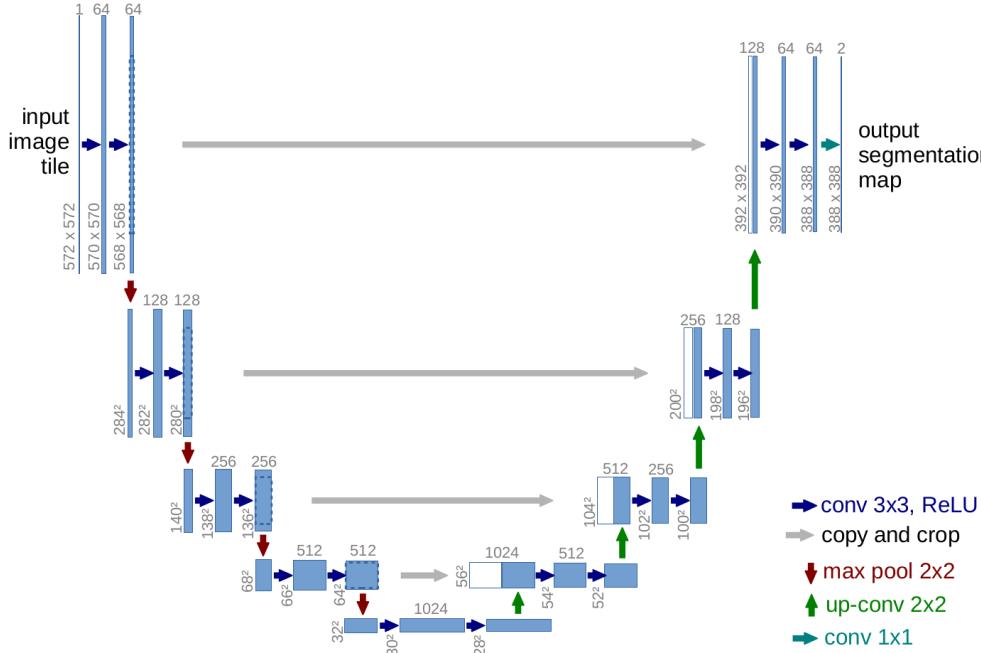


Abbildung 3.1: Die U-Net Netzwerkstruktur. Der Encoder und Decoder sind symmetrisch und durch skip connections an mehreren Stellen verbunden [37].

beim Upsampling wieder halbiert. In der originalen Architektur wurde kein Padding bei den Convolutionen genutzt, wodurch die Ausgabe des Netzwerkes ein wenig kleiner ist als die Eingabe [37].

U-Net nutzt außerdem so genannte skip connections zwischen den Blöcken des Encoders und Decoders. Diese übergeben die High-Level Features aus dem Encoder an den Decoder. Dort werden sie mit den Features, die im Decoder hoch skaliert wurden, konkateniert. Das Ziel von dieser Methode ist es, die Features präziser wiederherstellen zu können. Durch das Downsampling gehen im Laufe des Netzwerkes immer mehr Informationen verloren, wodurch es schwer ist die präzisen Konturen der Objekte zu rekonstruieren. In den High-Level Features aus dem Encoder sind diese Konturen noch besser erhalten, wodurch der Decoder diese Informationen für eine präzisere Klassifizierung nutzen kann [37].

In der hier genutzten Implementierung wurden zwei Spatial Dropout Layer mit dropout rate 0.5 in dem Block zwischen dem Encoder und Decoder eingefügt, um Overfitting zu reduzieren. Außerdem wurden alle Convolutional Layer durch Padding erweitert, um die originale Größe der Eingabebilder beizubehalten. In Tabelle 3.1 wird die hier genutzte Netzwerkarchitektur im Detail aufgelistet.

U-Net hat über 34 Millionen Parameter und ist damit eines der größten Netzwerke, die in dieser Arbeit genutzt wurden.

Block Name	Layer Typ	Ausgabegröße
Input	RGB-Bild	$512 \times 512 \times 3$
Encoder 1	2 x 3x3 Convolution, ReLU 2x2 Max-Pooling	$512 \times 512 \times 64$ $256 \times 256 \times 64$
Encoder 2	Wie Encoder 1	$128 \times 128 \times 128$
Encoder 3	Wie Encoder 1	$64 \times 64 \times 256$
Encoder 4	Wie Encoder 1	$32 \times 32 \times 512$
Dropout 1	Spatial Dropout, Rate 0.5	$32 \times 32 \times 512$
Zentrum	2 x 3x3 Convolution, ReLU	$32 \times 32 \times 1024$
Dropout 2	Spatial Dropout, Rate 0.5	$32 \times 32 \times 1024$
Decoder 1	2x2 transposed Convolution, ReLU Merge Skip Connection 2 x 3x3 Convolution, ReLU	$64 \times 64 \times 512$ $64 \times 64 \times 1024$ $64 \times 64 \times 512$
Decoder 2	Wie Decoder 1	$128 \times 128 \times 256$
Decoder 3	Wie Decoder 1	$256 \times 256 \times 128$
Decoder 4	Wie Decoder 1	$512 \times 512 \times 64$
Klassen	1x1 Convolution, Softmax	$512 \times 512 \times C$

Tabelle 3.1: In dieser Arbeit genutzte U-Net Implementierung. C steht für die Anzahl der Klassen.

3.2 SegNet

SegNet hat einen ähnlichen Aufbau zu U-Net. Auch hier ist der Decoder exakt symmetrisch zu dem Encoder und auch hier werden skip connections genutzt, um die Konturen besser rekonstruieren zu können. In Abbildung 3.2 ist der schematische Aufbau zu sehen [3].

Aber es gibt auch einige Unterschiede im Aufbau. Bei SegNet wird hinter jeder Convolution Batch Normalization (BN) hinzugefügt. Außerdem hat der Encoder den gleichen Aufbau wie die ersten 13 Layer des viel genutzten VGG16-Netzwerkes [42]. Dieses Netzwerk ist ein sehr beliebtes und sehr tiefes Klassifizierungsnetzwerk. Dadurch, dass in SegNet der gleiche Aufbau genutzt wird, können Parameter, die in dem VGG16-Netzwerk auf großen Klassifizierungsdatensätzen, wie zum Beispiel ImageNet, trainiert wurden, genutzt werden [3].

Ein weiterer großer Unterschied zum U-Net ist, dass anstatt der transposed Convolutions

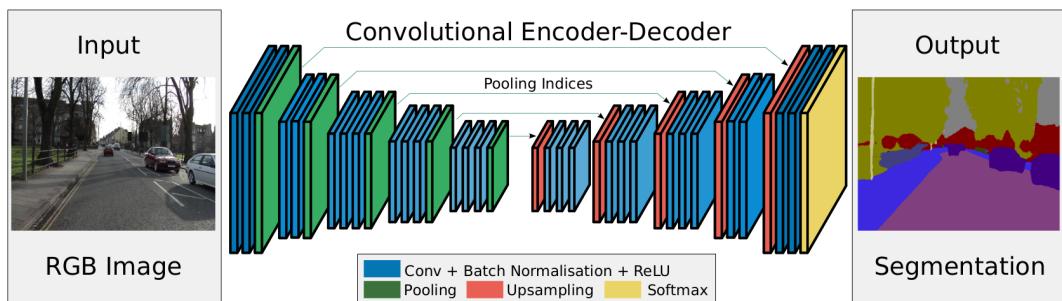


Abbildung 3.2: Die SegNet Netzwerkstruktur. Auch hier sind der Encoder und Decoder symmetrisch. Zum Upsampling wird Max-Unpooling eingesetzt [3].

Block Name	Layer Typ	Ausgabegröße
Input	RGB-Bild	$512 \times 512 \times 3$
Encoder 1	2 x 3x3 Convolution, BN, ReLU 2x2 Max-Pooling	$512 \times 512 \times 64$ $256 \times 256 \times 64$
Encoder 2	Wie Encoder 1	$128 \times 128 \times 128$
Encoder 3	3 x 3x3 Convolution, BN, ReLU 2x2 Max-Pooling	$128 \times 128 \times 256$ $64 \times 64 \times 256$
Dropout 1	Spatial Dropout, Rate 0.5	$64 \times 64 \times 256$
Encoder 4	Wie Encoder 3	$32 \times 32 \times 512$
Dropout 2	Spatial Dropout, Rate 0.5	$32 \times 32 \times 512$
Encoder 5	Wie Encoder 3	$16 \times 16 \times 512$
Dropout 3	Spatial Dropout, Rate 0.5	$16 \times 16 \times 512$
Decoder 1	2x2 Max-Unpooling 3 x 3x3 Convolution, BN, ReLU	$32 \times 32 \times 512$ $32 \times 32 \times 512$
Dropout 4	Spatial Dropout, Rate 0.5	$32 \times 32 \times 512$
Decoder 2	Wie Decoder 1	$64 \times 64 \times 256$
Dropout 5	Spatial Dropout, Rate 0.5	$64 \times 64 \times 256$
Decoder 3	Wie Decoder 1	$128 \times 128 \times 128$
Decoder 4	2x2 Max-Unpooling 2 x 3x3 Convolution, BN, ReLU	$256 \times 256 \times 128$ $256 \times 256 \times 64$
Decoder 5	Wie Decoder 4	$512 \times 512 \times 32$
Klassen	1x1 Convolution, Softmax	$512 \times 512 \times C$

Tabelle 3.2: In dieser Arbeit genutzte SegNet Implementierung. C steht für die Anzahl der Klassen.

das in 2.3.2 erwähnte Max-Unpooling zum Upsampling im Decoder genutzt wird. Das bedeutet auch, dass bei den skip connections im Gegensatz zum U-Net nicht die gesamten Feature Maps weiter gegeben werden, sondern nur die Max-Pooling-Indizes. Mit diesen Indizes sollen beim Max-Unpooling die Konturen der Objekte besser rekonstruiert werden können. Der große Vorteil von diesem Verfahren ist, dass der Speicheraufwand sehr viel geringer ist [3].

Ähnlich wie beim U-Net wurde im Rahmen dieser Arbeit wieder Spatial Dropout in das Netzwerk eingefügt. Der hier genutzte Aufbau ist in Tabelle 3.2 beschrieben. SegNet hat ungefähr 25 Millionen Parameter.

3.3 E-Net

Wenn semantische Segmentierung in mobilen Anwendungen, wie zum Beispiel autonomen Fahrzeugen, eingesetzt werden soll, steht nicht unbegrenzt viel Rechenleistung zur Verfügung. Trotzdem müssen die Algorithmen häufig in Echtzeit arbeiten. Um das zu ermöglichen, wurde die E-Net Architektur entwickelt. Darin wurden einige Maßnahmen vorgenommen, um die Anzahl der Parameter zu reduzieren und die Laufzeit während der Inferenz zu beschleunigen [4].

Das Netzwerk ist aus Residual Blöcken aufgebaut, wobei verschiedene Konfigurationen dieser Blöcke genutzt werden. In Abbildung 3.3 (b) ist der hier genutzte Residual Block

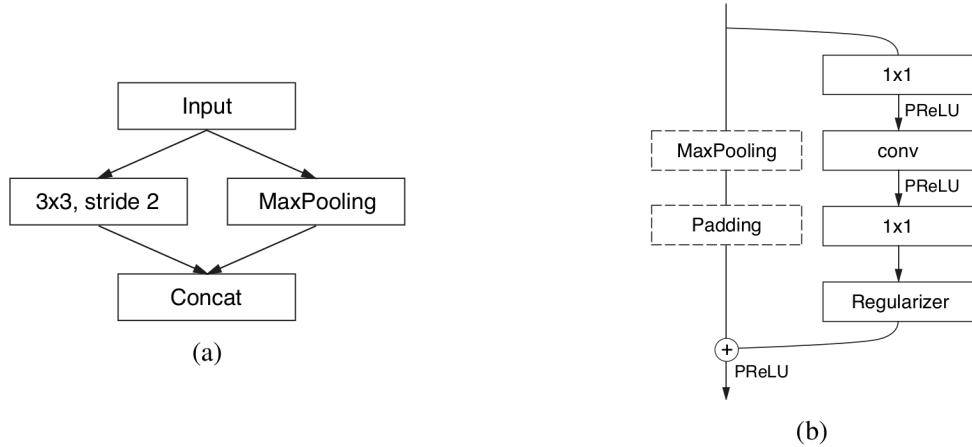


Abbildung 3.3: Die Blöcke aus denen E-Net aufgebaut wird. (a) ist der Initiale Block, der für Downsampling eingesetzt wird. (b) ist der Residual Block aus dem das restliche Netzwerk aufgebaut ist. Wenn der Block Downsampling durchführen soll, wird Max-Pooling und Padding eingesetzt [4].

abgebildet. Das grundsätzliche Design entspricht dem bottleneck-Block aus ResNet [18]. Es gibt eine 1x1 Convolution, die die Anzahl der Filter reduziert, eine 3x3 Convolution und noch eine 1x1 Convolution, die die Anzahl der Filter wieder erhöht. Außerdem wird Spatial Dropout zum Reduzieren von Overfitting eingesetzt. Bis zum Block Encoder 2 wird eine dropout rate von 0.01 verwendet, danach 0.1.

Wenn der Block ein Downsampling durchführen soll, wird auf der alternativen Verbindung ein Max-Pooling und Padding hinzugefügt. Außerdem wird die erste 1x1 Convolution durch eine 2x2 Convolution mit einer Schrittweite von 2 ersetzt.

Die zentrale Convolution wird manchmal durch eine dilated Convolution ersetzt, um ein größeres Rezeptives Feld zu erhalten. Manchmal wird sie auch durch zwei asymmetrische Convolutions mit Kernel Größen 5x1 und 1x5 ersetzt. Diese zwei Convolutions enthalten weniger Redundanz als eine 5x5 Convolution, aber haben eine Laufzeit ähnlich zu einer 3x3 Convolution [4].

Wenn der Block zum Upsampling eingesetzt werden soll, wird anstatt dem Max-Pooling ein Max-Unpooling Layer eingesetzt. Außerdem wird die erste Convolution durch eine transposed Convolution mit Schrittweite 2 ersetzt. Für das Max-Unpooling nutzt man skip connections, um die Pooling Indizes aus dem Encoder weiterzugeben.

In Abbildung 3.3 (a) ist der Initiale Block des Netzwerkes abgebildet. Dieser Block soll ein starkes Downsampling mit wenigen Feature Maps durchführen. Dafür läuft ein Max-Pooling und eine strided Convolution parallel ab. Die Idee dahinter ist, dass Bilder sehr viel redundante Informationen enthalten, weshalb ein starkes Downsampling zu Beginn des Netzwerkes keinen starken negativen Einfluss auf die Qualität der Ergebnisse hat. Dagegen werden die Laufzeit- und Speicherkosten dadurch stark reduziert, weil alle folgenden Feature Maps kleiner sind [4].

Der Decoder ist bei E-Net im Gegensatz zu U-Net und Segnet nicht symmetrisch zum Encoder, sondern sehr viel kleiner. Bei E-Net soll der Decoder nur dafür verantwortlich sein, die Ergebnisse des Encoders wieder auf die originale Größe zu bringen. Deshalb

Block Name	Layer Typ	Ausgabegröße
Input	RGB-Bild	$512 \times 512 \times 3$
Initialer Block	E-Net Downsamplingblock	$256 \times 256 \times 16$
Encoder 1	bottleneck, Downsampling 4 x bottleneck	$128 \times 128 \times 64$ $128 \times 128 \times 64$
Encoder 2	bottleneck, Downsampling bottleneck bottleneck, dilated 2 bottleneck, asymmetric bottleneck, dilated 4 bottleneck bottleneck, dilated 8 bottleneck, asymmetric bottleneck, dilated 16	$64 \times 64 \times 128$ $64 \times 64 \times 128$
Encoder 3	Wie Encoder 2, ohne Downsampling	$64 \times 64 \times 128$
Decoder 1	bottleneck, Upsampling 2 x bottleneck	$128 \times 128 \times 64$ $128 \times 128 \times 64$
Decoder 2	bottleneck, Upsampling bottleneck	$256 \times 256 \times 16$ $256 \times 256 \times 16$
Klassen	3x3 transposed Convolution, Softmax	$512 \times 512 \times C$

Tabelle 3.3: In dieser Arbeit genutzte E-Net Implementierung. C steht für die Anzahl der Klassen.

wird er so klein wie möglich gehalten, um die optimale Laufzeit zu liefern. Der gesamte Aufbau des Netzwerkes ist in Tabelle 3.3 zu sehen.

In den Convolutions im E-Net wird kein Bias verwendet, um die Speicherkosten weiter zu reduzieren. Außerdem wird nach den Convolutions BN genutzt [4].

In dem Netzwerk wird auch kein normales ReLU genutzt, sondern eine Erweiterung davon namens Parametric Rectified Linear Unit (PReLU) [19]. Diese Aktivierungsfunktion ist folgendermaßen definiert:

$$f(x) = \begin{cases} x : x > 0 \\ \alpha \cdot x : x \leq 0 \end{cases} \quad (3.1)$$

α ist ein trainierbarer Parameter. Wird $\alpha = 0$ gewählt entspricht es einer ReLU Funktion. Bei $\alpha = 1$ wäre es eine lineare Funktion [19]. In der Entwicklung von E-Net hat sich gezeigt, dass es manchmal von Vorteil war ReLU zu entfernen. Deswegen wurde ReLU durch PReLU ersetzt, weil das Netzwerk auf diese Weise die Möglichkeit hat die ReLU Funktionen selber zu deaktivieren [4].

Durch all diese Designentscheidungen hat das Netzwerk nur noch 400.000 Parameter, wodurch es deutlich kompakter ist als U-Net und SegNet.

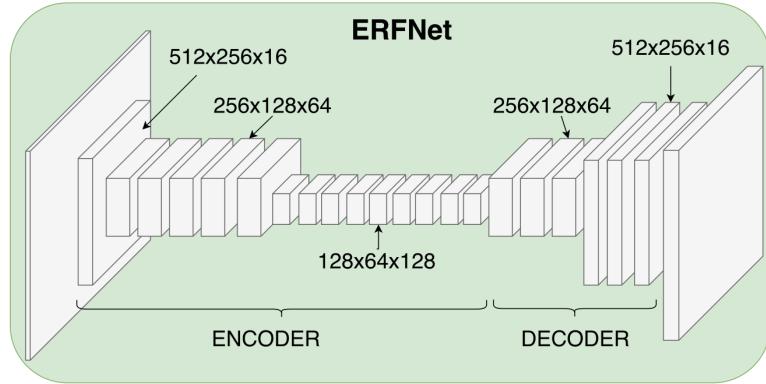


Abbildung 3.4: Die ERFNet Netzwerkstruktur [36].

3.4 ERFNet

Mit E-Net wurde eine Netzwerkarchitektur entwickelt, bei der das eindeutige Ziel war, die Laufzeit- und Speicherkosten so gering wie möglich zu halten [4]. Bei vielen anderen Architekturen ist das Ziel, möglichst gute Ergebnisse zu erreichen [3, 8, 31, 37]. ERFNet versucht einen guten Ausgleich zwischen den beiden Aspekten zu finden. Ergebnisse, die mit dem State of the Art mithalten können, aber gleichzeitig eine Laufzeit, die die Nutzung in Echtzeitanwendungen ermöglicht [36].

Um das zu erreichen, wurde eine neue Variante der Residual Blöcke vom ResNet entwickelt. In Abbildung 3.5 (c) ist dieses Design abgebildet. Es ist eine Verbesserung des originalen Non-bottleneck Designs. Dabei werden die beiden 3×3 Convolutionen jeweils durch eine 3×1 und eine 1×3 Convolution ersetzt. Durch diese Dekomposition wird die Anzahl der Parameter pro Convolution um 33% reduziert [36].

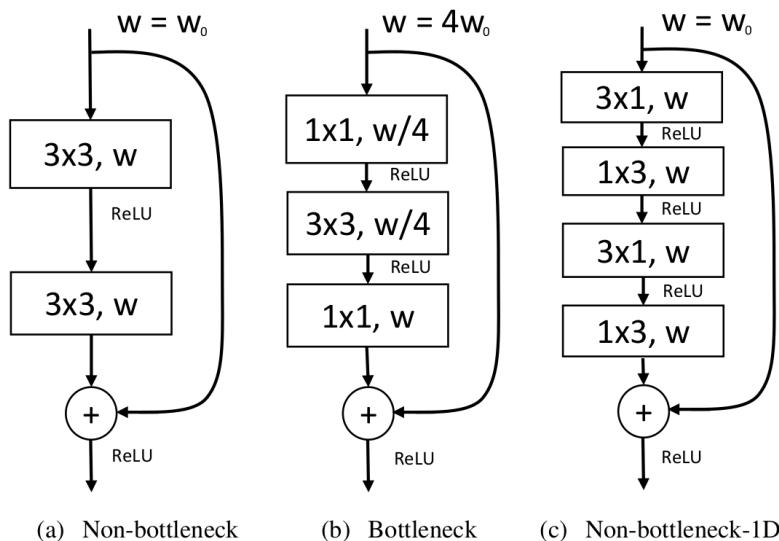


Abbildung 3.5: Verschiedene Varianten von Residual Blöcken. (a) und (b) sind die beiden in [18] vorgestellten Varianten. (c) ist das Non-bottleneck-1D (Non-bt-1D) Design, dass in ERFNet zum Einsatz kommt [36].

Block Name	Layer Typ	Ausgabegröße
Input	RGB-Bild	$512 \times 512 \times 3$
Initialer Block	E-Net Downsamplingblock	$256 \times 256 \times 16$
Encoder 1	E-Net Downsamplingblock 5 x Non-bt-1D	$128 \times 128 \times 64$ $128 \times 128 \times 64$
Encoder 2	E-Net Downsamplingblock Non-bt-1D, dilated 2 Non-bt-1D, dilated 4 Non-bt-1D, dilated 8 Non-bt-1D, dilated 16 Non-bt-1D, dilated 2 Non-bt-1D, dilated 4 Non-bt-1D, dilated 8 Non-bt-1D, dilated 16	$64 \times 64 \times 128$ $64 \times 64 \times 128$
Decoder 1	3x3 transposed Convolution, ReLU 2 x Non-bt-1D	$128 \times 128 \times 64$ $128 \times 128 \times 64$
Decoder 2	3x3 transposed Convolution, ReLU 2 x Non-bt-1D	$256 \times 256 \times 16$ $256 \times 256 \times 16$
Klassen	3x3 transposed Convolution, Softmax	$512 \times 512 \times C$

Tabelle 3.4: In dieser Arbeit genutzte ERFNet Implementierung. C steht für die Anzahl der Klassen.

Aus dieser Non-bt-1d Variante des Residual Blocks wird das ERFNet aufgebaut. Dieser Aufbau ist in Abbildung 3.4 schematisch dargestellt und in Tabelle 3.4 im Detail aufgelistet.

Ähnlich zu E-Net wird eine asymmetrische Encoder-Decoder Struktur genutzt, wobei der Encoder deutlich größer ist als der Decoder. Im Gegensatz zu allen anderen bisher vorgestellten Architekturen nutzt ERFNet keine skip connections zwischen dem Encoder und dem Decoder. Zum Upsampling werden transposed Convolutions genutzt.

Für das Downsampling wird der in Abbildung 3.3 (a) dargestellte Downampling Block verwendet. Im Gegensatz zu E-Net wird er in ERFNet nicht nur am Anfang des Netzwerkes sondern immer, wenn ein Downsampling durchgeführt wird, genutzt [36].

Wie bei E-Net werden auch hier einige dilated Convolutions in den Encoder eingefügt, um das Rezeptive Feld zu erhöhen, ohne einen höheren Rechenaufwand zu erzeugen. Außerdem wird in jedem Non-bt-1D Block ein Spatial Dropout Layer mit dropout rate 0.3 eingesetzt.

ERFNet hat insgesamt ungefähr 2 Millionen Parameter, womit es immer noch um ein Vielfaches kleiner ist als SegNet und U-Net.

3.5 DeepLab

DeepLab ist eine von Google entwickelte Netzwerkarchitektur für semantische Segmentation, die in den letzten Jahren immer weiter entwickelt wurde [5, 6, 7, 8].

In diesen Netzwerken wurden einige besondere Methoden eingesetzt, die die Ergebnisse verbessern sollten. Die grundsätzliche Struktur basiert auf erfolgreichen Klassifizierungs-

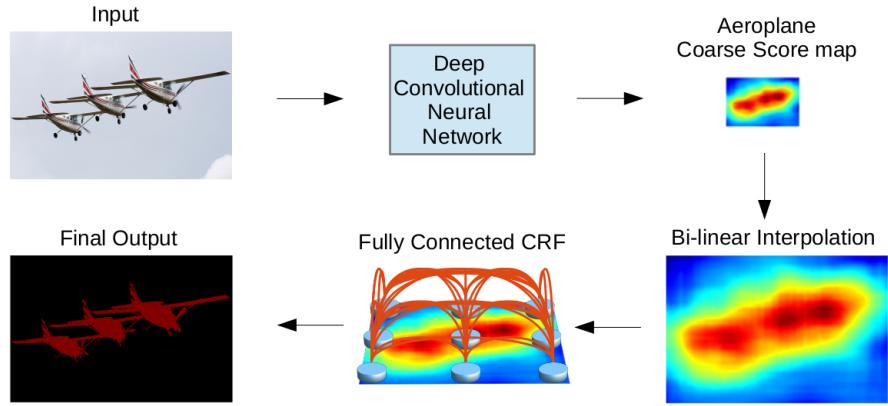


Abbildung 3.6: Der Ablauf von DeepLabV1. Zunächst werden grobe Ergebnisse mit einer leicht veränderten Version von VGG16 erzeugt. Diese werden danach interpoliert und durch ein CRF verfeinert [5].

netzwerken, wie VGG16 [42], ResNet101 [18] oder Xception [9].

3.5.1 DeepLabV1

Die erste Version von DeepLab basierte auf VGG16 [42]. Eines der Hauptziele war es das Netzwerk für semantische Segmentierung anzupassen, ohne dabei eine Encoder-Decoder Struktur mit transposed Convolutions wie bei dem FCN [31] zu nutzen. Der Grund dafür ist der zusätzliche Rechenaufwand, der durch einen solchen Decoder entsteht [5].

Um das zu erreichen, wurden in VGG16 einige der Downampling Layer entfernt und durch dilated Convolutions ersetzt. Damit soll das Rezeptive Feld des Netzwerks eine vergleichbare Größe behalten, aber das Downampling reduziert werden. Das Netzwerk gibt durch diesen Aufbau Ergebnisse aus, die um einen Faktor 8 kleiner sind als die Eingaben. Um die originale Größe zu erhalten, wird hier kein Decoder verwendet, sondern die Ausgabe des Netzwerkes wird durch bilineare Interpolation vergrößert [5].

Eine weitere Methode, die in DeepLab zur Verbesserung der Ergebnisse eingesetzt wurde, sind Conditional Random Fields (CRFs) [26]. Diese sollen dafür genutzt werden, die Präzision der Ergebnisse zu erhöhen. Damit bilden CRFs eine Alternative zu skip connections, die in anderen Architekturen für den gleichen Zweck, den Informationsverlust des Downsamplings auszugleichen, genutzt werden [3, 31, 37].

Die Fully-connected-CRF Variante von Krähenbühl et al. [26] hat eine relativ gute Laufzeit und eignet sich gut dafür, grobe Ergebnisse von einer semantischen Segmentierung zu verfeinern. Insbesondere bei Farbunterschieden zwischen den Klassen können CRFs sehr präzise Ergebnisse liefern [5]. Damit sind sie gut geeignet, um Fehler an den Übergängen zwischen den Klassen zu verbessern.

In Abbildung 3.6 ist der gesamte Ablauf von DeepLabV1 dargestellt.

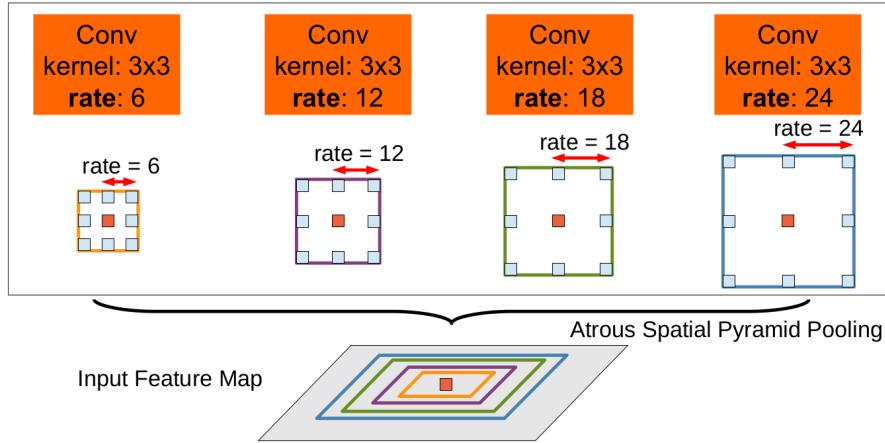


Abbildung 3.7: ASPP aus DeepLabV2. Die Eingabe wird mit parallelen dilated Convolutions mit verschiedenen dilation rates verarbeitet [6].

3.5.2 DeepLabV2

DeepLabV2 nutzt zwei Verbesserungen im Gegensatz zur ersten Version. Zum einen wurde analysiert wie sich das CNN, basierend auf ResNet101 [18], im Vergleich zu VGG16 [42] verhält. ResNet101 hat bessere Ergebnisse geliefert und nutzt weniger Parameter, wodurch auch die Rechenzeit und der Specheraufwand geringer sind [6]. Deshalb nutzt DeepLabV2 ResNet101 anstatt VGG16 als Grundlage.

Die zweite Verbesserung ist ein Mechanismus namens Atrous Spatial Pyramid Pooling (ASPP). Wie in Abbildung 3.7 dargestellt, werden dafür mehrere dilated Convolutions parallel geschaltet. Die Convolutions haben verschiedene dilation rates, wodurch sie Objekte in unterschiedlichen Größenordnungen erkennen können. Damit soll das Netzwerk insgesamt eine größere Invarianz gegenüber der Skalierung von Objekten erhalten [6]. Durch diese Änderungen an dem Netzwerk konnte DeepLabV2 deutlich bessere Ergebnisse liefern als DeepLabV1. Bei der PASCAL VOC 2012 Challenge [13] wurde die mean-*IoU* von 71,6%, welche DeepLabV1 erreicht hat, auf 79,7% erhöht [5, 6].

3.5.3 DeepLabV3

Im Rahmen der Entwicklung von DeepLabV3 wurde der Einsatz von dilated Convolutions noch detaillierter untersucht. Insbesondere wurde die Effektivität von vielen in Reihe geschalteten dilated Convolutions mit steigenden dilation rates untersucht. Letztendlich hat sich gezeigt, dass ASPP, wie es in DeepLabV2 zum Einsatz kam, effektiver ist. Deshalb wurde für DeepLabV3 eine leicht verbesserte Version von ASPP entwickelt und eingesetzt [7].

In DeepLabV2 wurden, wie in Abbildung 3.7 zu sehen, vier dilated Convolutions mit den Raten 6, 12, 18 und 24 parallel geschaltet. Allerdings ist aufgefallen, dass es zu Problemen kommen kann, wenn zu große dilation rates verwendet werden. Ist der Kernel der Convolution größer als die Eingabe, kann es passieren, dass die dilated Convolution sich wie eine 1x1 Convolution verhält, weil nur einer der Punkte vom Kernel gleichzeitig in dem eigentlichen Bild liegen kann [7]. In Abbildung 3.8 ist ein Beispiel dafür dargestellt.

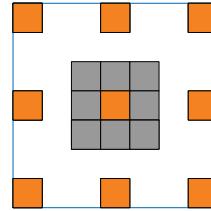


Abbildung 3.8: Eine dilated Convolution mit dilation rate 3 auf einer 3x3 Eingabe. Der Kernel ist zu groß, wodurch sich die Convolution wie ein 1x1 Convolution verhält, da nur der zentrale Punkt des Kernels auf dem Bild liegt.

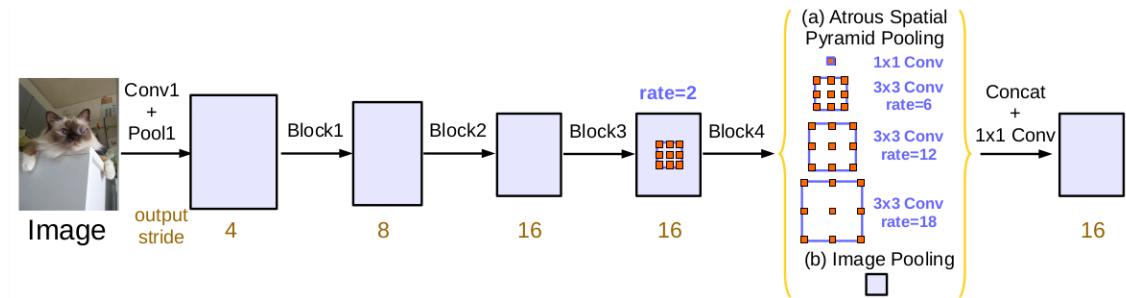


Abbildung 3.9: Der Aufbau von DeepLabV3. Die Ausgabe ist um einen Faktor von 16 kleiner als die Eingabe [7].

Der graue Bereich ist das Eingabebild und die orangenen Punkte sind der Kernel. Durch die dilation rate ist der Kernel zu groß, um auf die Eingabe zu passen.

Deshalb wurde ASPP wie in Abbildung 3.9 verändert. Die Convolution mit Rate 24 wurde entfernt und stattdessen eine 1x1 Convolution hinzugefügt. Außerdem wird ein Global Average-Pooling, also ein Average-Pooling über das gesamte Bild hinzugefügt, um Image-Level Features zu extrahieren. Anschließend werden alle 5 parallelen Pfade konkateniert und in eine weitere 1x1 Convolution gegeben.

In DeepLabV3 wird im Gegensatz zu den älteren Versionen auch nur ein Downampling Schritt durch dilated Convolutions ersetzt, wodurch die Ausgabe um einen Faktor von 16 kleiner ist als die Eingabe. Trotzdem wird nur eine bilineare Interpolation verwendet, um ein Upsampling durchzuführen [7].

Ein weiterer Unterschied von DeepLabV3 zu den Vorgängerversionen ist, dass BN in das Netzwerk eingefügt wurde.

Die größte Abweichung ist aber, dass das CRF am Ende des Netzwerkes entfernt wurde. Die Begründung dafür waren die hohen Laufzeitkosten [7].

Insgesamt erreicht DeepLabV3 deutlich bessere Ergebnisse als DeepLabV2, obwohl keine CRFs mehr genutzt werden. In der PASCAL VOC 2012 Challenge wurde 86,9% mean-IoU erreicht [7, 13].

3.5.4 DeepLabV3+

Die neueste Version der DeepLab Architektur erweitert DeepLabV3 um einen Decoder. Damit ist es die erste Version von DeepLab, die eine echte Encoder-Decoder Struktur

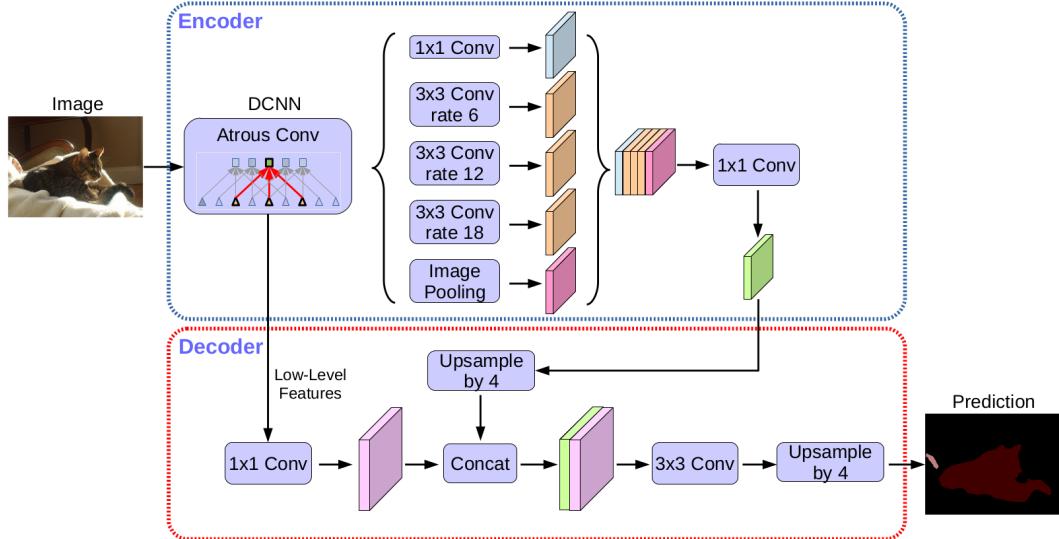


Abbildung 3.10: Der Aufbau von DeepLabV3+. Der Encoder entspricht dem von DeepLabV3, der in 3.9 dargestellt ist. Ein simpler Decoder wurde angefügt, um präzisere Ergebnisse zu erzeugen [8].

verwendet [8].

Der Aufbau ist in Abbildung 3.10 dargestellt und ist folgendermaßen:

Die Ergebnisse des ASPP werden zunächst mit bilinearem Upsampling um einen Faktor 4 vergrößert und dann mit Low-Level Features konkateniert. Diese werden mithilfe einer skip connection aus dem Encoder entnommen. Bei diesen Features wird die Anzahl der Filter mithilfe einer 1x1 Convolution reduziert, damit sie die Features, die von dem ASPP kommen, nicht zu stark ausstechen. Damit soll es dem Decoder, wie in anderen Encoder-Decoder-Architekturen, besser möglich sein, die präzisen Konturen der Objekte zu rekonstruieren. Danach werden zwei 3x3 Convolutions durchgeführt. Zum Schluss werden die Ergebnisse durch eine weiter bilineare Interpolation um einen Faktor 4 auf die gewünschte Ausgabegröße gebracht [8].

Im Rahmen von DeepLabV3+ wurde auch noch eine neue Variante des Encoders untersucht. Dieser basiert auf dem Xception Modell [9]. Xception setzt so genannte separable Convolutions ein, welche die Komplexität der Berechnungen stark reduzieren sollen [8]. Mit diesem Encoder konnten in dem DeepLab Paper auch noch bessere Ergebnisse erzielt werden. Insgesamt erreicht DeepLabV3+ eine mean-IoU von 89% bei der PASCAL VOC 2012 Challenge. Damit ist es zum Stand dieser Arbeit (April 2019) immer noch auf Platz eins der Rangliste dieser Challenge [13].

Außerdem wurde im Rahmen dieser Arbeit eine leicht veränderte Variante des ResNet Encoders getestet. Die bottleneck Module, aus denen es sonst aufgebaut wurde, wurden durch die verbesserten Non-bt-1D Module aus ERFNet ersetzt. Der daraus resultierende Aufbau ist in Tabelle 3.5 beschrieben.

Mit diesem Aufbau hat das Netzwerk ungefähr 34 Millionen Parameter und hat damit eine vergleichbare Größe wie U-Net.

Block Name	Layer Typ	Ausgabegröße
Input	RGB-Bild	$512 \times 512 \times 3$
Encoder 1	3x3 Convolution, BN, ReLU, strided 2	$256 \times 256 \times 64$
	2 x 3x3 Convolution, BN, ReLU	$256 \times 256 \times 64$
	Max-Pooling	$128 \times 128 \times 64$
Encoder 2	3 x Non-bt-1D	$64 \times 64 \times 64$
Encoder 3	bottleneck, Downsampling	$64 \times 64 \times 128$
	3 x Non-bt-1d	$64 \times 64 \times 128$
Encoder 4	bottleneck, Downsampling	$32 \times 32 \times 256$
	22 x Non-bt-1D	$32 \times 32 \times 256$
Encoder 5	Non-bt-1D, dilated 2	$32 \times 32 \times 512$
	Non-bt-1D, dilated 4	$32 \times 32 \times 512$
	Non-bt-1D, dilated 8	$32 \times 32 \times 512$
ASPP	ASPP, wie in 3.10 dargestellt	$32 \times 32 \times 256$
Decoder 1	Bi-Lineare Interpolation	$128 \times 128 \times 256$
	Merge skip connection	$128 \times 128 \times 512$
	2 x 3x3 Convolution, ReLU	$128 \times 128 \times 256$
Decoder 2	Bi-Lineare Interpolation	$512 \times 512 \times 256$
Klassen	1x1 Convolution, Softmax	$512 \times 512 \times C$

Tabelle 3.5: In dieser Arbeit genutzte DeepLabV3+ Implementierung mit dem veränderten ResNet Encoder, basierend auf den Non-bt-1D Blöcken von ERFNet. C steht für die Anzahl der Klassen.

4

KAPITEL

Datensätze

4.1	Stimmband-Datensatz	40
4.1.1	Bisherige Experimente	42
4.2	Mitochondrien-Datensatz	42
4.3	Cityscapes-Datensatz	43
4.4	ImageNet-Datensatz	45

In dieser Arbeit wurden vier verschiedene Datensätze verwendet. Zwei medizinische Datensätze, die allerdings aus sehr unterschiedlichen Arten von Daten bestehen und deshalb auch verschiedene Herausforderungen beim Training bringen sollten. Die anderen beiden Datensätze sind sehr große, öffentliche Datensätze, die in der Forschung sehr häufig genutzt werden. In dieser Arbeit wurden sie für Pre-Training eingesetzt.

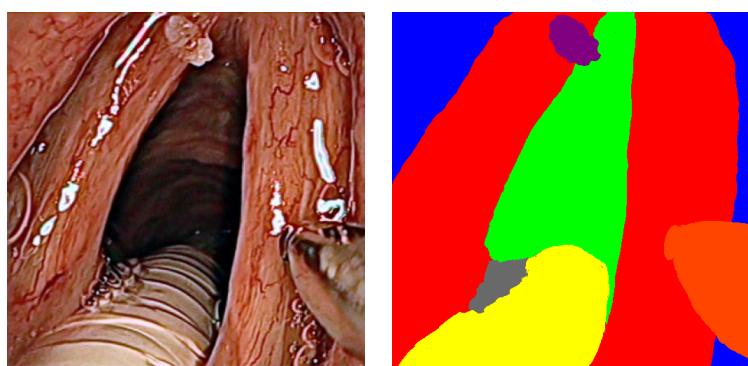


Abbildung 4.1: Visualisierung der Ground Truth des Stimmband-Datensatzes und das dazugehörige Originalbild. Die Klassen wurden mit folgenden Farben versehen: *void*: grau, *vocal folds*: rot, *other tissue*: blau, *glottal space*: grün, *pathology*: lila, *surgical tool*: orange und *intubation*: gelb [27].

4.1 Stimmband-Datensatz

Der wichtigste Datensatz für diese Arbeit ist ein medizinischer Datensatz mit Bildern von Stimmbändern. Er besteht aus 400 RGB-Bildern, die während zwei Operationen mit einem Stereoendoskop aufgenommen wurden [27].

Die Bilder wurden per Hand segmentiert und die Annotationen wurden mit zur Verfügung gestellt. Der Datensatz enthält sieben Klassen:

- **void:** Nicht genau bestimmte Objekte
- **vocal folds:** Stimmbandgewebe
- **other tissue:** Anderes Hautgewebe
- **glottal space:** Rachenraum
- **pathology:** Tumorgewebe
- **surgical tool:** Operationswerkzeuge
- **intubation:** Intubationsröhre

In Abbildung 4.1 ist eine Visualisierung der Ground Truth des Datensatzes zu sehen. Dabei wurden die verschiedenen Klassen mit unterschiedlichen Farben dargestellt.

Es wurden acht Videosequenzen zur Erstellung des Datensatzes aufgenommen. Die Bilder wurden aus fünf dieser Sequenzen entnommen. Die übrigen Sequenzen wurden aufgrund von zu hoher Ähnlichkeit zu den anderen Sequenzen nicht genutzt. Dabei wurde jeweils ein Mal pro Sekunde ein Bild entnommen, um die Ähnlichkeit zwischen den Bildern zu reduzieren. Die Videosequenzen wurden in unterschiedlichen Stadien der Operationen aufgenommen. Die Sequenzen haben folgende Eigenschaften [27]:

- **Sequenz 1:** Patient 1, vor der Operation, mit sichtbarem Tumor, keine Intubation, keine Operationswerkzeuge, 32 Bilder
- **Sequenz 2:** Patient 1, vor der Operation, mit sichtbarem Tumor, mit Intubation, mit Operationswerkzeugen, 34 Bilder
- **Sequenz 3:** Patient 1, nach der Operation, ohne sichtbaren Tumor, mit Intubation, keine Operationswerkzeuge, 26 Bilder
- **Sequenz 7:** Patient 2, vor der Operation, ohne sichtbaren Tumor, mit Intubation, mit Operationswerkzeugen, 122 Bilder
- **Sequenz 8:** Patient 2, nach der Operation, ohne sichtbaren Tumor, mit Intubation, mit Operationswerkzeugen, 186 Bilder

In Abbildung 4.2 sind Bilder aus den verschiedenen Sequenzen des Datensatzes dargestellt. Es ist leicht zu erkennen, dass sich die Bilder von Patient 1 und Patient 2 sehr stark unterscheiden, aber untereinander große Ähnlichkeiten aufweisen.

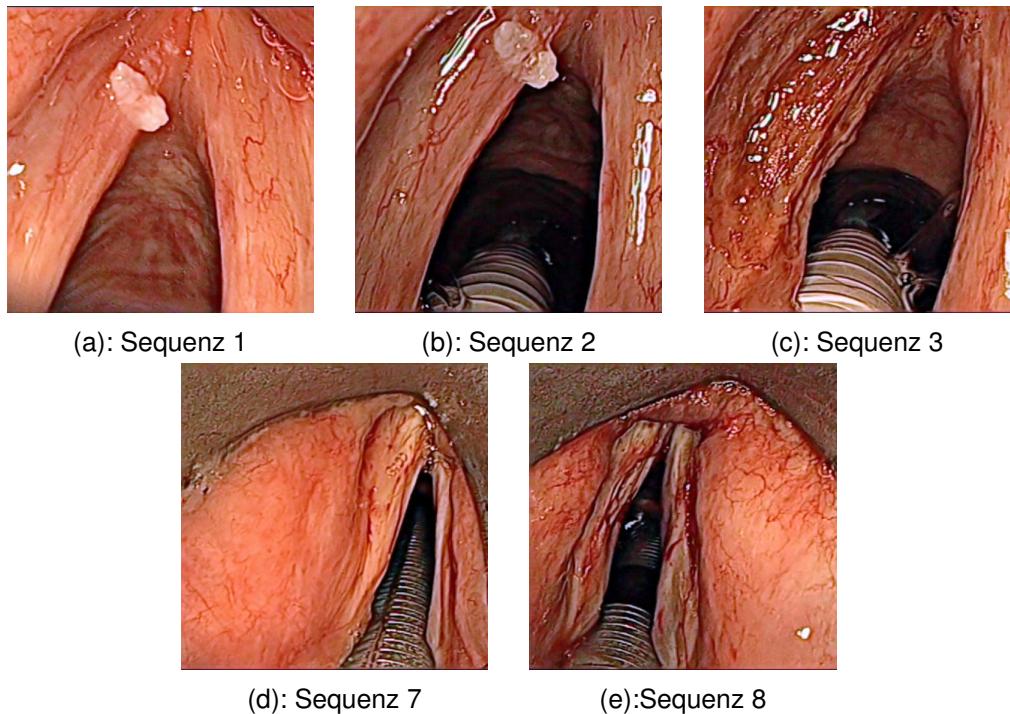


Abbildung 4.2: Bilder aus den verschiedenen Sequenzen des Stimmband-Datensatzes [27].

Der Datensatz wurde folgendermaßen für Training, Validierung und Test aufgeteilt. Die ersten beiden Viertel der Bilder aus allen Sequenzen sind die Trainingsdaten. Für die Validierung wurde das dritte Viertel und für die Tests das vierte Viertel jeder Sequenz verwendet. Auf diese Weise soll jede Eigenschaft des Datensatzes in den Trainings-, Validierungs- und Testabschnitt gleich häufig vorkommen. Außerdem ist dadurch, dass die Reihenfolge der Aufnahme der Bilder beibehalten wird, die Ähnlichkeit zwischen den drei Abschnitten des Datensatzes geringer, als wären die Bilder in zufälliger Reihenfolge zugewiesen worden.

Trotz dieser Aufteilung gibt es in den drei Unterdatensätzen einige Bilder, die sich sehr ähnlich sehen, wie in Abbildung 4.3 zu sehen ist. Durch so hohe Ähnlichkeiten in den Trainings-, Validierungs- und Testdatensätzen können die Ergebnisse verfälscht werden,

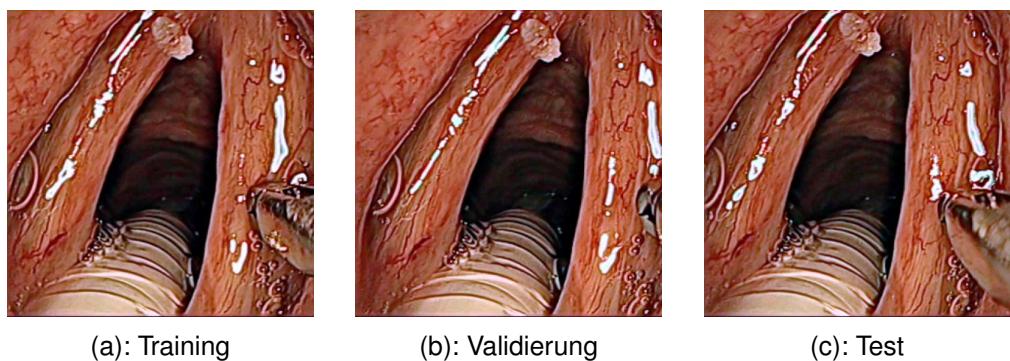


Abbildung 4.3: Bilder aus dem Trainings-, Validierungs- und Testdatensatz mit großer Ähnlichkeit [27].

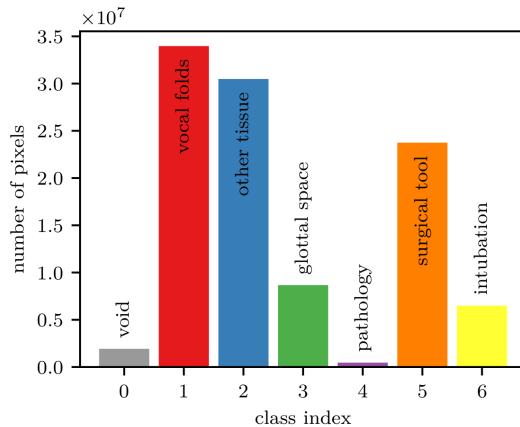


Abbildung 4.4: Anzahl der Pixel der einzelnen Klassen in dem Stimmband-Datensatz [27].

da hohes Overfitting der Trainingsdaten, besser erscheinende Ergebnisse bei der Validierung und bei den Tests bewirken könnte.

Ein weiteres Problem in diesem Datensatz ist, dass das Vorkommen der unterschiedlichen Klassen sehr unausgeglichen ist. Insbesondere die *pathology* Klasse kommt nur sehr selten in dem Datensatz vor. In Abbildung 4.4 sind die Anzahlen der Pixel der verschiedenen Klassen im Datensatz dargestellt.

Außerdem ist nur bei Bildern von Patient 1 die *pathology* Klasse vertreten und das immer durch den gleichen Tumor. Dadurch sieht jedes Objekt der *pathology* Klasse in dem Datensatz sehr ähnlich aus. Dies kann die Ergebnisse beeinflussen, da das Netzwerk diese Klasse sehr leicht durch starkes Overfitting der Objekte lernen kann. Sollte in der Inferenz jedoch ein Tumor mit einer etwas anderen Gestalt auftreten, kann das Netzwerk diesen vermutlich nicht gut erkennen.

4.1.1 Bisherige Experimente

Auf diesem Datensatz wurden bereits einige Experimente für semantische Segmentierung durchgeführt [27]. Diese wurden als Grundlage für diese Arbeit genutzt. Die Ergebnisse werden durch die in Kapitel 6 durchgeföhrten Experimenten erweitert.

Laves et al. [27] hat mit vier verschiedenen Netzwerkarchitekturen (U-Net, SegNet, E-Net, ERFNet) Experimente durchgeführt. Außerdem wurden Augmentierungen, Pre-Training und Ensemble-Strukturen genutzt, um die Ergebnisse zu verbessern.

Die besten Ergebnisse konnten mit einer Ensemble-Struktur aus U-Net und ERFNet, beziehungsweise aus allen vier Netzwerken mit einer mean-IoU von 84,7% erreicht werden [27].

4.2 Mitochondrien-Datensatz

Ein weiterer medizinischer Datensatz, der hier genutzt wurde, besteht aus Elektronenmikroskopaufnahmen von Mitochondrien [32]. Die Daten liegen in zwei 1024x768x165 großen Volumen vor. Da die hier genutzten Netzwerke auf 2D Bildern arbeiten, wurden

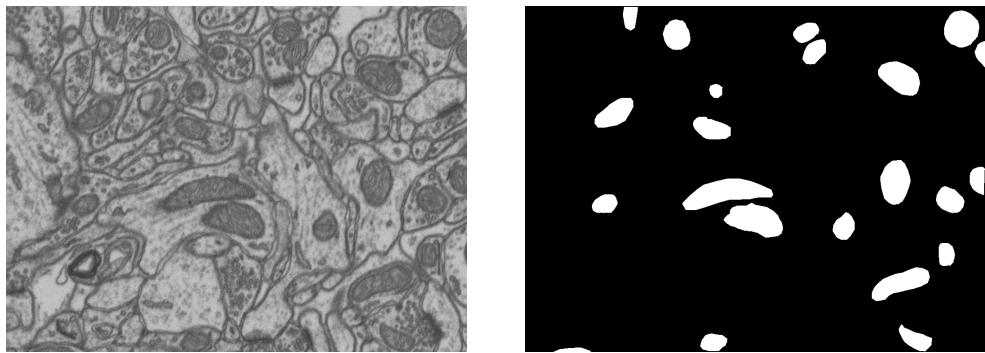


Abbildung 4.5: Beispielbild und die zugehörige Ground Truth des Mitochondrien-Datensatzes. Die weißen Bereiche in der Ground Truth sind die Mitochondrien [32].

diese in jeweils 165 1024x768 Grauwert-Bilder aufgespalten. Die Bilder aus dem ersten Volumen werden für das Training verwendet, die anderen für die Evaluierung.

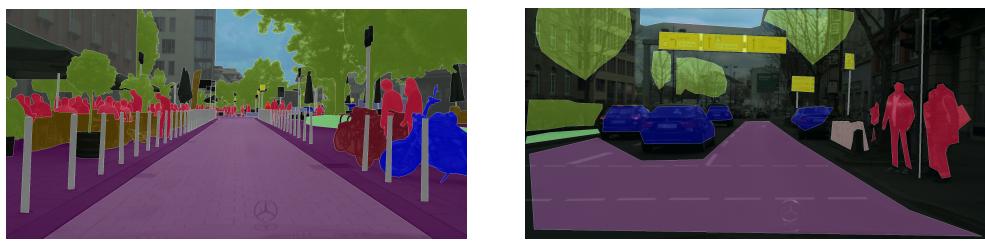
Die zu dem Datensatz vorliegenden Annotationen umfassen die Klassen Hintergrund und Mitochondrien. In Abbildung 4.5 ist ein Bild aus dem Datensatz und die zugehörige Annotation zu sehen.

Die Komplexität dieses Datensatzes ist durch die geringe Anzahl der Klassen und der Ähnlichkeit aller Bilder sehr viel geringer als die des Stimmband-Datensatzes. Deshalb soll dieser Datensatz hauptsächlich dazu genutzt werden, um zu untersuchen, ob die Methoden, die bei dem Stimmband-Datensatz die besten Ergebnisse liefern, auch bei anderen Arten von Daten optimal sind oder ob für simplere Daten andere Netzwerkarchitekturen besser geeignet sind.

4.3 Cityscapes-Datensatz

Der Cityscapes-Datensatz ist ein öffentlich zugänglicher Datensatz für semantische Segmentierung. Er wurde mit dem Ziel, die Entwicklung in der semantischen Szenenanalyse voranzubringen, entwickelt, da es laut den Autoren zur Zeit der Veröffentlichung keinen anderen Datensatz mit einer vergleichbaren Größe und Qualität dafür gab [10].

Der Datensatz besteht aus 5000 präzise und 20000 grob annotierten Bildern. Diese wurden aus einem Fahrzeug heraus in 50 verschiedenen Städten in Deutschland und der Umgebung aufgenommen. Dabei wurde großer Wert darauf gelegt, dass der Datensatz



(a): Präzise annotiert

(b): Grob annotiert

Abbildung 4.6: Ein präzise und ein grob annotiertes Bild aus dem Cityscapes-Datensatz [10].

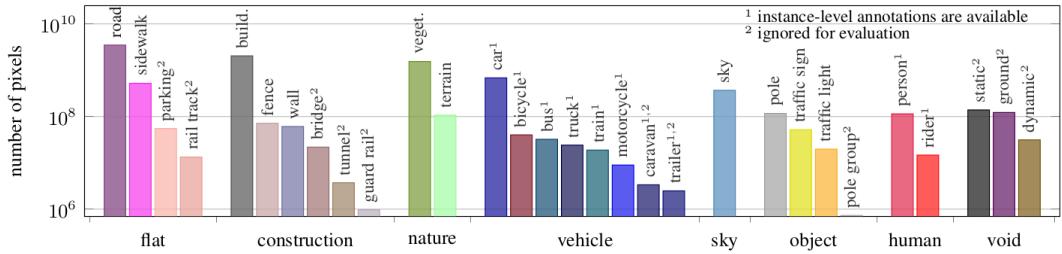


Abbildung 4.7: Die verschiedenen Klassen und die Anzahl ihrer Pixel im Cityscapes-Datensatz [10].

alle Jahreszeiten und verschiedene Wetterbedingungen abdeckt. Extreme Wetterbedingungen wie Schneestürme oder Starkregen sind absichtlich nicht enthalten, weil es sinnvoller ist für solche Sonderfälle speziell angepasste Datensätze zu nutzen [10].

In Abbildung 4.6 ist ein Bild aus dem präzise annotiertem Teil und ein Bild aus dem groben Teil des Datensatzes dargestellt. Die Ground Truth wurde zur Visualisierung über das Originalbild gelegt.

Der Datensatz enthält 30 verschiedene Klassen. Darunter befinden sich zum Beispiel Klassen für unterschiedliche Fahrzeugtypen wie Auto, Bus, LKW aber auch Klassen wie Gebäude oder Straße. Selten vorkommende Klassen werden für die Evaluierung nicht berücksichtigt, wodurch in dem Validierungs- und Testdatensatz noch 19 Klassen übrig bleiben. In Abbildung 4.7 sind die Klassen und ihre Häufigkeit im Datensatz dargestellt. Die mit ² gekennzeichneten Klassen werden in der Evaluierung ignoriert.

Für die präzise annotierten Daten wurde auch eine Aufteilung in Trainings-, Validierungs- und Testdatensatz vorgenommen. So enthält der Datensatz 2975 Trainings-, 500 Validierungs- und 1525 Testbilder. Die 20000 grob annotierten Bilder können genutzt werden, um die Datenmenge im Training zu erhöhen. Dies kann trotz der schlechteren Qualität der Ground Truth die Ergebnisse der Netzwerke deutlich verbessern [10].

Bei der Aufteilung der Bilder in die drei Unterdatensätze wurde großer Wert auf eine homogene Verteilung aller Eigenschaften des Datensatzes gelegt. Deshalb enthält jeder der drei Teile folgende vier Eigenschaften [10]:

- Bilder stammen aus großen, mittelgroßen und kleinen Städten
- Die Städte liegen im geografischen Westen, Mitte und Osten
- Die Städte liegen im geografischen Norden, Mitte und Süden
- Die Bilder wurden am Anfang, in der Mitte und am Ende des Jahres aufgenommen.

Die Bilder haben eine Größe von 2048 x 1024 Pixeln und liegen sowohl in 16 Bit HDR Format, als auch in 8 Bit LDR Format vor. Außerdem stehen Stereo-Bilder, die zur Nutzung von Tiefen-Informationen eingesetzt werden können, zur Verfügung



Abbildung 4.8: Bilder aus drei verschiedenen Klassen des ImageNet-Datensatzes [40].

4.4 ImageNet-Datensatz

ImageNet ist ein sehr umfangreicher, öffentlich zugänglicher Bilddatensatz für Klassifizierung und Objekterkennung. Die Klassenstruktur basiert auf der Struktur von der WordNet Datenbank [34]. Darin ist eine hierarchische Struktur aus so genannten synonym sets (synsets) aufgebaut. Ein synset beschreibt dabei eine Klasse. WordNet enthält 80000 dieser synsets. In ImageNet sollen für jedes synset letztendlich 500 bis 1000 Bilder zur Verfügung gestellt werden [11].

Die Bilder des Datensatzes können sehr unterschiedliche Qualitäten und Formate haben. In Abbildung 4.8 sind einige Bilder aus dem Datensatz zu sehen.

Seit 2010 gibt es einen jährlichen Wettbewerb im Zusammenhang mit ImageNet namens ILSVRC. Dabei wird ein Teildatensatz von ImageNet zu Verfügung gestellt, auf dem die Teilnehmer Experimente durchführen sollen. Dieser besteht aus ungefähr 1,4 Millionen Bildern aus 1000 verschiedenen synsets. Die Testdaten werden nicht veröffentlicht. Die Evaluierung findet auf einem dafür ausgelegten Server statt und die Ergebnisse werden am Ende des Wettbewerbes veröffentlicht [40].

ImageNet ist einer der Standarddatensätze, um die Qualität von Klassifizierungsnetzwerken zu messen und zu vergleichen [9, 18, 25, 42]. Außerdem wird er aufgrund seiner Größe und der hohen Diversität der Bilder und Klassen sehr häufig für Pre-Training genutzt, selbst für Architekturen, die nicht für Klassifizierung eingesetzt werden [3, 8].

5

KAPITEL

Fortgeschrittene Methoden

5.1 Pre-Training	47
5.2 Augmentierung	48
5.3 Ensemble Netzwerke	50

Abgesehen von dem Entwickeln besserer Architekturen gibt es noch einige weitere Methoden, um die Ergebnisse von neuronalen Netzwerken zu verbessern. Im Folgenden werden drei dieser Methoden vorgestellt.

5.1 *Pre-Training*

Eine sehr häufig genutzte Technik bei der Entwicklung von KNNs ist das so genannte Pre-Training. Dabei wird das Netzwerk vor dem Training auf dem eigentlichen Datensatz auf einem anderen, üblicherweise deutlich größeren Datensatz, trainiert. Die Parameter werden danach beibehalten und durch ein weiteres Training auf dem richtigen Datensatz verfeinert.

Dieses Verfahren bringt einige Vorteile im Gegensatz zu einem Training von zufällig initialisierten Parametern mit sich. Zum einen ist die Konvergenz während des Trainings deutlich schneller, da die Parameter bereits sinnvollere Werte enthalten. Außerdem können Netzwerke, die auf größeren Datensätzen vortrainiert wurden, häufig bessere Ergebnisse erzielen als Netzwerke, die nur auf kleinen Datensätzen trainiert wurden. Je kleiner der eigentliche Datensatz ist, desto größer ist die Verbesserung, die durch Pre-Training erzielt werden kann. [45].

Sehr häufig wird der ImageNet-Datensatz aufgrund seiner Größe und Vielfältigkeit für das Pre-Training genutzt. Für ein Netzwerk, das semantische Segmentierung durchführt, kann es auch sinnvoll sein einen darauf ausgelegten Datensatz, wie zum Beispiel Cityscapes, für das Pre-Training zu benutzen.

In medizinischen Anwendungsbereichen gibt es nur selten große und frei zugängliche Datensätze. Es gibt also keine speziellen Datensätze für das Pre-Training von medizinischen Anwendungen. Tajbakhsh et al. [45] hat allerdings gezeigt, dass auch Netzwerke für medizinische Anwendungen von Pre-Training auf Datensätzen wie ImageNet profitieren können.

5.2 Augmentierung

Daten Augmentierung ist eine sehr beliebte Methode, um die Qualität der Ergebnisse von KNNs zu verbessern. Dabei wird der Datensatz vergrößert, indem die Trainingsdaten auf irgendeine Weise verändert werden.

Eine große Verbesserung, die damit erreicht werden kann, ist, dass die Netzwerke größere Invarianzen gegen gewisse Eigenschaften der Daten erhalten können [37]. Wird zum Beispiel ein Datensatz genutzt, in dem alle Objekte in eine Richtung ausgerichtet sind, kann das darauf trainierte Netzwerk Schwierigkeiten damit haben Objekte mit einer anderen Ausrichtung zu erkennen. Wird der Datensatz allerdings durch Spiegelungen augmentiert, kann das Netzwerk durch diese erweiterten Daten auch in der Inferenz Objekte mit anderen Ausrichtungen besser erkennen [49].

Ein Training auf einem größeren Datensatz kann im Allgemeinen auch bessere Ergebnisse liefern als ein Training auf einem sehr kleinen Datensatz [10]. Deshalb ist es sinnvoll Augmentierungen zu nutzen, um den Datensatz zu vergrößern, wenn er sehr klein ist. Besonders bei medizinischen Anwendungen ist es sehr schwer an große Datenmengen zu gelangen, da es sehr strenge Datenschutzrichtlinien gibt, die berücksichtigt werden müssen [48].

Durch Augmentierung kann man außerdem Overfitting reduzieren, da die Daten eine höhere Datenmenge und Varianz erhalten [49].

Grundsätzlich gibt es grenzenlose Möglichkeiten, um Bilddaten zu augmentieren. Allerdings ist es wichtig, dass die veränderten Daten immer noch realistisch sind [27]. Es macht keinen Sinn in einem Fahrzeugdatensatz vertikale Spiegelungen zu nutzen, da in den meisten Anwendungsfällen keine Fahrzeuge vorkommen, die auf ihrem Dach liegen. Außerdem müssen Augmentierungen die Daten auch stark genug verändern, um einen Vorteil zu bringen. Einzelne Pixel in einem hochauflösenden Bild zu verändern, wird im Training keinen wertvollen Mehrwert liefern.

In dieser Arbeit wurden die folgenden Methoden genutzt, um Daten zu augmentieren:

Horizontale Spiegelung

Die Bilder werden entlang der X-Achse gespiegelt. In Abbildung 5.1 (b) wurde diese Operation an einem Bild aus dem Stimmband-Datensatz durchgeführt.

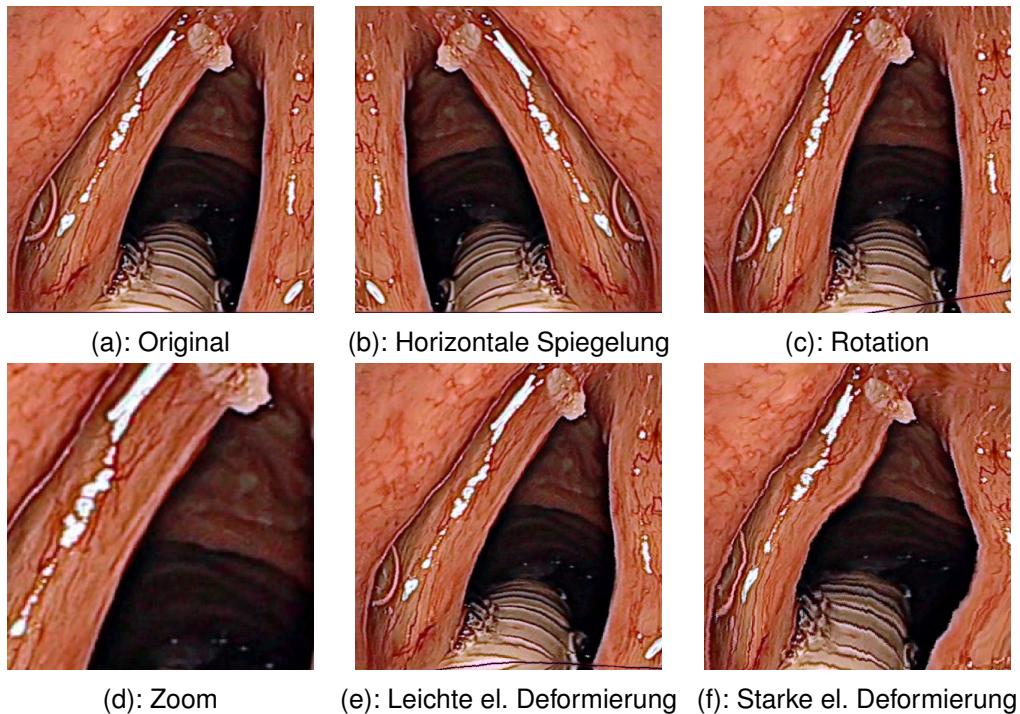


Abbildung 5.1: Die verschiedenen Augmentierungen, die in dieser Arbeit genutzt wurden. (a) ist das originale Bild aus dem Stimmband-Datensatz. (b) wurde horizontal gespiegelt, (c) rotiert, in (d) wurde ein Ausschnitt vergrößert, (e) zeigt eine leichte elastische Deformierung und (f) eine sehr starke elastische Deformierung [27].

Rotation

Hierbei wird eine Rotation um den Mittelpunkt des Bildes durchgeführt. Als Padding wird eine Spiegelung der Daten genutzt. Abbildung 5.1 (c) zeigt ein Beispiel für eine leichte Rotation.

Zoom

Bei dieser Operation wird das Bild vergrößert und auf die ursprüngliche Größe zugeschnitten. Dadurch bleibt nur ein vergrößerter Ausschnitt des Ursprungsbildes übrig.

Elastische Deformierung

Elastische Deformierungen sollen Teile des Bildes leicht verzerrten. Bei medizinischen Bildern kann so eine Augmentierung sehr realistisch sein, wenn in den Bildern weiches Gewebe zu sehen ist. Dafür werden mit einer Gauß-Verteilung zufällige Verzerrungen für die x und y Richtung erzeugt. Diese werden auf ein Gitter angewendet, mit dem sie auf das eigentliche Bild übertragen werden [41].

Eine solche Deformierung hat zwei Parameter. α wird für eine Skalierung genutzt, um die Intensität der Deformierung festzulegen. σ ist die Standardabweichung der Gauß-Verteilungen. Das heißt durch ein größeres α können Deformierungen erzeugt werden, die über einen größeren Bereich gehen und durch ein kleineres σ kann die Stärke der Deformierung erhöht werden [41].

Dies ist in Abbildung 5.1 sichtbar. In (e) wurde $\alpha = 5000$ und $\sigma = 100$ gewählt. In (f) wurde dagegen $\alpha = 10000$ und $\sigma = 80$ genutzt. Es ist gut zu erkennen, dass in (f) deutlich extremere Deformierungen entstanden sind. Allerdings sind mit diesen Parametern auch schon einige Kanten entstanden, die nicht mehr einer realistischen Deformierung von weichem Gewebe entsprechen.

5.3 Ensemble Netzwerke

Die letzte Methode zur Verbesserung der Qualität der Ergebnisse, die hier behandelt wird, sind Ensemble Netzwerke. Dabei wird eine Kombination, also ein Ensemble, aus mehreren einzeln trainierten KNNs gebildet.

Die Idee dahinter ist, dass die verschiedenen Netzwerke unterschiedliche Stärken und Schwächen haben können. Durch eine Kombination der Netzwerke können die unterschiedlichen Stärken genutzt und die Schwächen ausgeglichen werden [17]. In Abbildung 5.2 ist ein gutes Beispiel dafür zu sehen. Die ersten drei Netzwerke wurden trainiert, um Punkte im ersten und dritten Quadranten des Bildes zu erkennen. Dies liefert keine guten Ergebnisse. Das Ensemble der Netzwerke erzeugt sehr gute Ergebnisse.

Es gibt verschiedene Methoden, um Ensembles zu erstellen. Entweder kann mehrfach die gleiche Architektur verwendet aber das Training leicht verändert werden, um sicher zu stellen, dass die Netzwerke unterschiedliche Ergebnisse liefern. Dafür kann zum Beispiel der Datensatz für die verschiedenen Netzwerke oder die Hyperparameter, wie die Lernrate, verändert werden. Außerdem könnten unterschiedliche Methoden zur Initialisierung der Parameter genutzt werden [51].

Eine andere Option, welche in dieser Arbeit angewendet wurde, ist, das Ensemble aus unterschiedlichen Netzwerkarchitekturen aufzubauen [27].

Es gibt auch unterschiedliche Möglichkeiten, um die Ergebnisse der einzelnen Netzwerke zu kombinieren. Die beliebtesten Methoden sind entweder ein majority voting oder eine gewichtete Summe. Bei dem majority voting wird das Ergebnis gewählt, dass von der Mehrzahl der Netzwerke erzeugt wurde [17]. Für eine gewichtete Summe wird ein Parameter für die Gewichtung eingefügt und trainiert, während die restlichen Parameter der Netzwerke eingefroren wurden [27].

Bei dem Aufbau von Ensemble-Netzwerken ist zu beachten, dass es nicht immer sinnvoll ist so viele Netzwerke wie möglich zu nutzen. Wenn zwei KNNs sehr ähnliche Ergebnisse liefern, bringt es keinen Vorteil beide in einem Ensemble einzusetzen [51].



Abbildung 5.2: Drei Netzwerke und ihr Ensemble, die darauf trainiert wurden, Punkte im ersten und dritten Quadranten eines Bildes zu erkennen [17].

6

KAPITEL

Experimente

6.1	Verschiedene Architekturen	52
6.1.1	Stimmband-Datensatz	53
6.1.2	Mitochondrien-Datensatz	55
6.1.3	Analyse	56
6.2	Pre-Training	58
6.2.1	Stimmband-Datensatz	58
6.2.2	Mitochondrien-Datensatz	59
6.2.3	Analyse	59
6.3	Augmentierungen	60
6.3.1	Stimmband-Datensatz	60
6.3.2	Mitochondrien-Datensatz	61
6.3.3	Analyse	61
6.4	Ensemble Netzwerke	63
6.4.1	Stimmband-Datensatz	63
6.4.2	Mitochondrien-Datensatz	64
6.4.3	Analyse	64

In den folgenden Abschnitten werden die Experimente, die mit den vorgestellten Methoden und Architekturen durchgeführt wurden, beschrieben.

Die Durchführung entsprach folgendem Ablauf: Zunächst wurden die verschiedenen, in Kapitel 3 vorgestellten, Architekturen getestet und verglichen. Auf die beiden besten Architekturen wurden danach die drei Methoden aus Kapitel 5 angewendet, um die Ergebnisse zu verbessern. Alle diese Experimente wurden ausführlich auf dem Stimmband-Datensatz und danach auf dem Mitochondrien-Datensatz durchgeführt.

Die Algorithmen wurden in Python mit dem Tensorflow Framework [2] für neuronale Netzwerke implementiert. Dieses enthält bereits Implementierungen für viele der notwendigen Mechanismen, wie zum Beispiel verschiedene Layer Typen, Optimizer oder auch Mechanismen, um die Parameter von Netzwerken zu speichern und laden. Außerdem ist es bereits für den Einsatz auf Graphics Processing Units (GPUs) optimiert.

Als Fehlerfunktion wurde Categorical Cross-Entropy genutzt. Außerdem wurde der Adam Optimierer zum Training eingesetzt.

Aufgrund der hohen Anzahl an Experimenten konnten einige Hyperparameter aus Zeitgründen nicht für jedes Experiment optimiert werden. Deshalb wurden sie für alle Experimente folgendermaßen festgelegt:

- **Lernrate:** $1e^{-4}$ zu Beginn des Trainings. Senkung nach 5 Epochen ohne Verbesserung der Validierungsmetrik.
- **Adam Optimierer Parameter:** $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e^{-8}$
- **Batchgröße:** 1

Außerdem wurde bei dem Stimmband-Datensatz für die Berechnung der Categorical Cross-Entropy eine Gewichtung, wie in [27] beschrieben, vorgenommen.

Mit diesen Parametern wurden die Netzwerke in allen Experimenten bis zur Konvergenz trainiert. Dies wurde 10 Mal wiederholt, um zufällige Abweichungen, die zum Beispiel durch die Initialisierung der Parameter entstehen können, zu reduzieren.

Alle Experimente wurden auf einer Nvidia Tesla V100 PCIe GPU mit 16 GB Speicher durchgeführt.

6.1 Verschiedene Architekturen

Im ersten Experiment wurden die fünf Architekturen, U-Net, SegNet, E-Net, ERFNet und DeepLabV3+, die in Kapitel 3 vorgestellt wurden, verglichen. Dafür wurden sie von einer zufälligen Initialisierung aus auf dem Stimmband-Datensatz trainiert. Bei DeepLabV3+ wurden außerdem die verschiedenen backbone Modelle verglichen. Das bedeutet, es wurden separate Tests für DeepLabV3+ ResNet101 (DL-RN) und DeepLabV3+ Xception (DL-Xc) durchgeführt. Außerdem wurde die in dieser Arbeit entwickelte Version DeepLabV3+ Non-bottleneck-1D (DL-Nbt), welche die Non-bt-1D Blöcke aus ERFNet verwendet, getestet. Der genaue Aufbau dieser Architektur wurde in Tabelle 3.5 beschrieben.

Bei U-Net, SegNet, E-Net, ERFNet und DeepLabV3+ Xception wurde ein Glorot Initialisierer genutzt, um die Parameter zu Beginn des Trainings festzulegen. Bei den anderen beiden DeepLabV3+ Varianten hat das Training mit diesem Initialisierer nur sehr selten funktioniert. Deshalb wurde für sie ein skalierbarer Initialisierer, wie in Abschnitt 2.3.5 beschrieben, verwendet. Aber auch mit diesem Initialisierer ließ sich die Variante mit ResNet101 als backbone nur ungefähr in jedem zweiten Durchlauf trainieren. Die neue Non-bt-1D Variante konnte mit diesem Initialisierer dagegen immer gut optimiert werden. In Tabelle 6.1 sind einige Eigenschaften der Architekturen aufgelistet. Zunächst fällt auf, dass E-Net und ERFNet mit deutlichem Abstand die kleinsten Architekturen sind. ERFNet ist um mehr als einen Faktor von 10 kleiner als alle anderen Architekturen und E-Net ist noch einmal um einen Faktor von 4 kleiner. Die größte Architektur ist DL-Xc mit 63 Millionen Parametern. Auch die Version mit ResNet101 ist mit fast 58 Millionen Parametern

Netzwerk	Anzahl Parameter	Geschwindigkeit	Geschwindigkeit
		Training	Inferenz
U-Net	34 513 735	10 it/s	21 it/s
SegNet	24 897 255	9 it/s	17 it/s
E-Net	405 115	12 it/s	30 it/s
ERFNet	2 057 379	22 it/s	40 it/s
DL-Nbt	34 107 958	7 it/s	22 it/s
DL-RN	57 709 494	10 it/s	24 it/s
DL-Xc	63 004 110	5 it/s	20 it/s

Tabelle 6.1: Anzahl der Parameter, Trainingsgeschwindigkeit und Inferenzgeschwindigkeit in Iterationen pro Sekunde der verschiedenen Architekturen.

sehr groß. Die hier entwickelte Variante mit den Non-bt-1D Blöcken ist mit 34 Millionen Parametern deutlich kleiner und hat damit eine vergleichbare Größe wie U-Net.

Aus Sicht der Laufzeit ist in den hier durchgeführten Experimenten allerdings nicht E-Net, sondern ERFNet am besten. Auf dem Stimmband-Datensatz erreichte es im Training eine Geschwindigkeit von 22 Iterationen pro Sekunde (it/s) und 40 it/s bei der Inferenz. E-Net ist mit 12 it/s im Training und 30 it/s in der Inferenz das zweitschnellste Netzwerk. Bei den verschiedenen DeepLabV3+ Varianten gibt es auch Unterschiede in der Laufzeit. DL-RN ist mit 10 it/s im Training das schnellste der DeepLabV3+ Netze. Die Xception Variante ist mit nur 5 it/s im Training das langsamste hier getestete Netzwerk. In der Inferenz ist allerdings SegNet mit 17 it/s am langsamsten. DL-Xc kann dabei 20 it/s erreichen.

6.1.1 Stimmband-Datensatz

In Abbildung 6.1 ist die Validierungs-mean-IoU der Architekturen während des Trainings auf dem Stimmband-Datensatz abgebildet. Daran lässt sich ablesen, wie viele Epochen ein Netzwerk benötigt, um zu konvergieren. Hierdurch kann die Dauer des Trainings abgeschätzt werden. Es fällt auf, dass die drei DeepLabV3+ Varianten zu Beginn des Trainings deutlich schneller optimiert werden als alle anderen hier getesteten Architekturen. Sie benötigen nur ungefähr 50 Epochen bis sie bereits Werte erreichen, die sehr nah an ihren endgültigen Ergebnissen liegen. Die Xception Version konvergiert allerdings bei einem deutlich geringerem Wert als die anderen beiden Versionen. SegNet wird am Anfang des Trainings am langsamsten optimiert und erreicht auch mit deutlichem Abstand den schlechtesten finalen Wert. Die anderen drei Architekturen haben ein sehr ähnliches Verhalten. Sie haben aber einen sehr viel langsameren Anstieg der mean-IoU als die DeepLabV3+ Netzwerke. Sie benötigen über 150 Epochen, um endgültig zu konvergieren, können dann aber Werte mit ähnlich guter Qualität wie DeepLabV3+ erreichen.

In Tabelle 6.2 sind die Test-IoUs der verschiedenen Klassen und die Test-mean-IoU nach dem Training auf dem Stimmband-Datensatz aufgelistet. DL-Nbt hat mit 76.8 % mean-IoU das beste Ergebnis erreicht. Die ResNet101 Variante von DeepLab und ERFNet liegen mit 75.7 % und 75.4 % knapp dahinter. Die schlechteste Architektur war in diesem Experiment SegNet mit einer mean-IoU von 59.6 %.

Die *pathology* Klasse ist besonders interessant, da es die Klasse ist, die am wenigsten

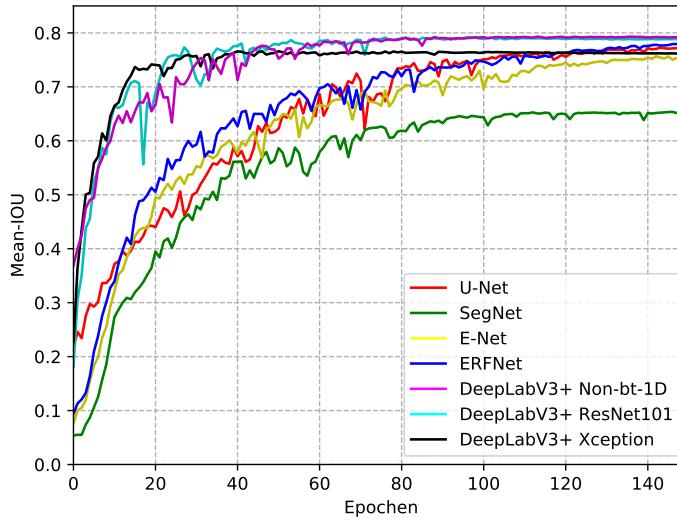


Abbildung 6.1: Validierungs-IoU der verschiedenen Architekturen während des Trainings auf dem Stimmband-Datensatz.

im Datensatz vertreten ist, aber gleichzeitig für viele Anwendungsfälle am wichtigsten ist. Dabei hat auch DL-Nbt mit 73.7 % IoU das beste Ergebnis erzielt. Am zweitbesten war auch dort DL-RN, allerdings nur mit 66.7 %, also bereits einem deutlichen Unterschied.

In Abbildung 6.2 ist ein Bild aus dem Testdatensatz mit den Ergebnissen der Netzwerke nach dem Training zu sehen. Ähnlich wie in Abbildung 4.1 beschrieben, wurden die Klassen zur Visualisierung eingefärbt. Anders als dort, stehen hier die grauen Bereiche allerdings nicht für die *void* Klasse, sondern für Punkte, die von dem Netzwerk einer falschen Klasse zugeordnet wurden. In (d) sieht man, dass ERFNet hauptsächlich leichte Fehler an den Stellen macht, wo eine Klasse zu einer anderen übergeht. Die restlichen Flächen wurden in diesem Beispiel sonst sehr gut erkannt. (b) zeigt, dass SegNet deutlich größere Fehler macht, die nicht nur direkt an den Übergängen zwischen Klassen liegen. Es fällt auch auf, dass es einige Bereiche in dem Bild gibt, wo jede Architektur sehr ähnliche Fehler macht.

Netzwerk	vocal folds	other tissue	glottal space	pathology	surgical tool	intubation	mean-IoU
U-Net	77.7 %	72.5 %	70.6 %	59.9 %	81.9 %	77.1 %	73.3 %
SegNet	66.2 %	63.9 %	53.3 %	31.8 %	72.4 %	69.9 %	59.6 %
E-Net	78.9 %	75.8 %	68.8 %	44.5 %	81.9 %	77.9 %	71.3 %
ERFNet	79.9 %	76.9 %	72.9 %	60.0 %	83.3 %	79.2 %	75.4 %
DL-Nbt	78.3 %	76.4 %	67.5 %	73.7 %	84.4 %	80.7 %	76.8 %
DL-RN	79.6 %	77.1 %	67.8 %	66.7 %	82.3 %	80.7 %	75.7 %
DL-Xc	75.4 %	74.3 %	65.3 %	51.2 %	81.1 %	77.9 %	70.9 %

Tabelle 6.2: Test-IoUs der Architekturen nach dem Training auf dem Stimmband-Datensatz.

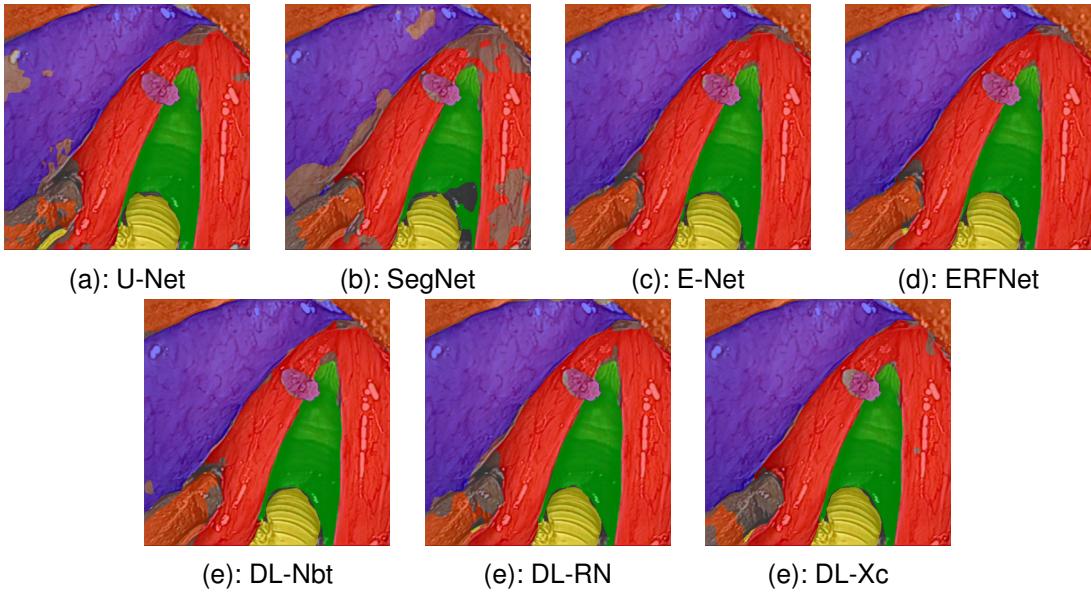


Abbildung 6.2: Ergebnisbilder nach dem Training des Stimmband-Datensatzes [27]. Die Ausgabe der Netzwerke wurde in Farbe über das Eingabebild gelegt. Grau markierte Bereiche wurden von den Netzwerken falsch klassifiziert.

6.1.2 Mitochondrien-Datensatz

Mit dem Mitochondrien-Datensatz wurden die gleichen Experimente wie mit dem Stimmband-Datensatz durchgeführt. Dabei gab es allerdings Probleme bei den DeepLabV3+ ResNet101 und Non-bt-1D Varianten. DL-RN konnte mit dem skalierbaren Initialisierer nicht auf dem Mitochondrien-Datensatz trainiert werden. Bei DL-Nbt gab es auch Probleme, allerdings konnte es in einigen Fällen erfolgreich optimiert werden.

In 6.3 sind die Ergebnisse der Experimente auf dem Mitochondrien-Datensatz aufgelistet. Es fällt auf, dass bei diesem Datensatz das U-Net, mit 80.2 % mean-Test-IoU, die besten Ergebnisse erzeugt. DL-Nbt erreicht im Schnitt 78.1 % mean-IoU, wenn das Training erfolgreich ist. Das schlechteste Netzwerk bei diesem Experiment ist DL-Xc mit nur 52.5 % mean-IoU.

In Abbildung 6.3 sind die Ausgaben der Netzwerke nach dem Training auf dem Mitochondrien-Datensatz dargestellt. Es ist zu sehen, dass U-Net fast jedes Mitochondrium in dem Beispiel sehr gut erkennen konnte. Die Xception Variante von DeepLabV3+ hat im Gegen-

Netzwerk	background	mitochondria	mean-IoU
U-Net	97.7 %	62.7 %	80.2 %
SegNet	95.1 %	23.5 %	59.3 %
E-Net	96.9 %	52.7 %	74.8 %
ERFNet	97.2 %	57.3 %	77.3 %
DL-Nbt	97.2 %	59.1 %	78.1 %
DL-RN	-	-	-
DL-Xc	94.4 %	10.3 %	52.5 %

Tabelle 6.3: Test-IoUs der Architekturen nach dem Training auf dem Mitochondrien-Datensatz.

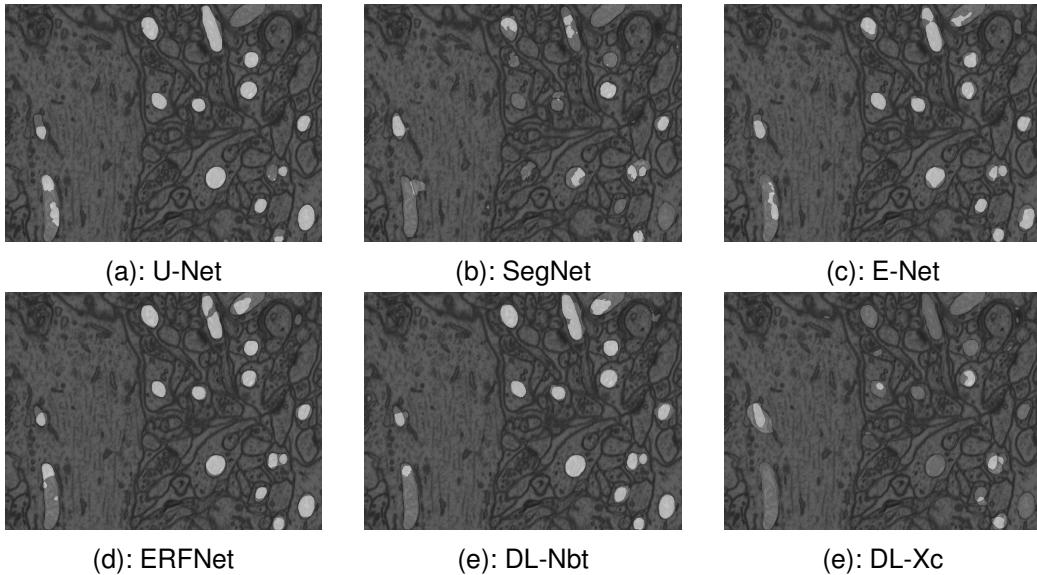


Abbildung 6.3: Ergebnisbilder nach dem Training des Mitochondrien-Datensatzes [32]. Weiße Bereiche sind korrekt erkannte Mitochondrien, grau markierte Bereiche wurden von den Netzwerken falsch klassifiziert.

satz dazu fast kein Mitochondrium mit guter Präzision erkannt. Es fällt auf, dass ERFNet zwar jedes Mitochondrium erkennt, aber die Kanten nicht so gut rekonstruieren kann wie U-Net.

6.1.3 Analyse

Es gibt verschiedene Aspekte, die bei der Wahl der optimalen Netzwerkarchitektur betrachtet werden können. Diese sind zum Beispiel Speicher Kosten, Trainingszeit, Inferenzzeit und Qualität der Ergebnisse. Für unterschiedliche Anwendungsfälle müssen sie unterschiedlich priorisiert werden.

In dem Fall, dass nur eine Hardware mit sehr begrenztem Speicher zur Verfügung steht, ist die E-Net Architektur am sinnvollsten. Durch die extrem geringe Anzahl an Parametern benötigt das Netzwerk mit Abstand am wenigsten Speicher. Als weiteren Vorteil ermöglicht das auch die Nutzung von größeren Batchgrößen. Dies kann einige Vorteile haben, wie in Abschnitt 2.3.4 beschrieben wurde [22].

Obwohl E-Net extrem kompakt ist, hat ERFNet in den hier durchgeführten Experimenten sowohl im Training, als auch in der Inferenz eine bessere Laufzeit erreicht. Da ERFNet mit einer mean- IoU von 75,4 % auch deutlich bessere Ergebnisse liefert als E-Net, ist es in Anwendungsfällen, die sehr schnelle Laufzeiten benötigen, zu bevorzugen. Sollte ein besonders hoher Wert auf der gesamten Trainingsdauer liegen, ist ERFNet allerdings nicht eindeutig das schnellste Netzwerk. Die DeepLabv3+ Architekturen können im Training sehr viel schneller konvergieren. Sie benötigen zwar etwas mehr Zeit pro Iteration, dafür erreichen sie schon nach ungefähr 50 Epochen sehr gute Ergebnisse. ERFNet benötigt dafür über 150 Epochen. Da alle drei DeepLabv3+ Varianten sehr schnell konvergieren, muss dieses Verhalten durch den besonderen Decoder oder das ASPP verursacht werden.

Bei genauerer Untersuchung der drei Versionen von DeepLabV3+ fällt als erstes auf, dass DL-Xc deutlich schlechtere Ergebnisse liefert als die anderen beiden Varianten. Das ist besonders überraschend, da diese Version in der originalen Veröffentlichung [8] besser war als DL-RN. Mit der hier eingeführten Non-bt-1D Version des Netzwerkes konnten sowohl die Speicher Kosten reduziert, als auch die Qualität der Ergebnisse verbessert werden. Dafür muss allerdings eine etwas schlechtere Laufzeit in Kauf genommen werden. Des Weiteren lässt sich DL-Nbt mit der hier genutzten Initialisierung der Parameter verlässlicher optimieren als DL-RN. Dies könnte daran liegen, dass die Non-bt-1D Blöcke weniger anfällig für das Degradation Problem sind als die klassischen Residual Blöcke [36].

Die anderen beiden Architekturen, U-Net und SegNet, haben bei dem Training auf dem Stimmband-Datensatz in jedem Aspekt schlechter abgeschnitten als andere Netzwerke. Besonders SegNet hat sehr ernüchternde Ergebnisse geliefert. Da sich SegNet und U-Net in ihrem Aufbau sonst sehr ähneln, kann davon ausgegangen werden, dass die schlechteren Ergebnisse von Segnet durch die Nutzung des Max-Unpoolings verursacht wurden. Es scheint als würden die Max-Pooling-Indizes zum Wiederherstellen der verlorenen Informationen deutlich schlechter geeignet sein, als die skip connections, die in U-Net zum Einsatz kommen.

Für den Stimmband-Datensatz kann zusammenfassend gesagt werden, dass DL-Nbt am besten ist, wenn das Hauptkriterium die Qualität der Ergebnisse ist. Falls eine höhere Inferenzgeschwindigkeit und geringerer Speicheraufwand nötig sind, ist ERFNet besser geeignet und liefert immer noch sehr gute Ergebnisse. E-Net kann genutzt werden, wenn noch härtere Speicherrestriktionen vorliegen. Allerdings muss dann bereits eine schlechtere Qualität der Ergebnisse in Kauf genommen werden.

Auf dem Mitochondrien-Datensatz sind die Ergebnisse der Experimente anders ausgefallen. Dort konnte U-Net die besten Ergebnisse erzeugen. Der Mitochondrien-Datensatz besteht aus deutlich simpleren Daten als der Stimmband-Datensatz. Die Bilder liegen nur in Graustufen vor, es gibt nur zwei Klassen und die gesamten Bilder sehen sich relativ ähnlich. Daher scheint die größte Herausforderung bei diesem Datensatz darin zu liegen, die genaue Form der Mitochondrien wiederherzustellen. U-Net ist dafür besonders gut geeignet, da es viele skip connections einsetzt. DeepLabv3+ nutzt nur eine skip connection zwischen dem Encoder und Decoder und ERFNet gar keine.

Bei genauerer Untersuchung der Ausgaben der Netzwerke fällt auf, dass U-Net an den Übergängen zwischen den Klassen zum Teil präziser ist als ERFNet, aber an anderen Bereichen größere Fehler macht. In [36] wurde behauptet, dass das Nutzen von skip connections in ERFNet keine Verbesserung der Genauigkeit bringt. Die hier erzeugten Ergebnisse scheinen aber doch die Wichtigkeit von skip connections zur präziseren Wiederherstellung der Konturen zu zeigen.

6.2 Pre-Training

In den folgenden Experimenten wurden nur noch die zwei besten Architekturen verwendet. Für den Stimmband-Datensatz also ERFNet und DL-Nbt und für den Mitochondrien-Datensatz U-Net und DL-Nbt.

Mit jeweils beiden Architekturen wurde ein Pre-Training auf dem Cityscapes-Datensatz und anschließend ein weiteres Training auf dem Stimmband- beziehungsweise dem Mitochondrien-Datensatz, durchgeführt. Dabei wurden die Parameter, die beim Cityscapes Training erzeugt wurden, als Initialisierung für das folgende Training genutzt. Nur das letzte Layer musste neu initialisiert werden, da die Datensätze unterschiedliche Anzahlen an Klassen haben, wodurch sich die Struktur des letzten Layers verändert.

Für DL-Nbt wurde außerdem ein Pre-Training mit dem ImageNet-Datensatz durchgeführt. Da dieser Datensatz nur eine Klassifizierung und keine semantische Segmentierung durchführt, wurde nur der Encoder, also das ResNet mit Non-bt-1D Blöcken, auf ImageNet trainiert. In dem Encoder wurden die dabei erzeugten Parameter dann als Initialisierung für das andere Training genutzt. Die restlichen Parameter wurden zufällig initialisiert.

Da das Training auf dem ImageNet-Datensatz aufgrund seiner Größe sehr lange dauert, wurde es aus Zeitgründen nur für die DL-Nbt Architektur durchgeführt.

6.2.1 Stimmband-Datensatz

In Tabelle 6.4 sind die Ergebnisse des Trainings auf dem Stimmband-Datensatz nach dem Pre-Training aufgelistet. Bei einem Vergleich der Werte mit den aus Tabelle 6.2 fällt auf, dass es keine starke Verbesserung der Ergebnisse durch das Pre-Training gegeben hat.

In Abbildung 6.4 ist die Validierungs-mean-IoU während des Trainings auf dem Stimmband-Datensatz mit und ohne Pre-Training dargestellt. Dort ist zu sehen, dass sowohl ERFNet als auch DeepLab deutlich schneller konvergieren, wenn vorher ein Pre-Training durchgeführt wurde. DL-Nbt kann nach dem Pre-Training auf ImageNet sogar noch schneller optimiert werden als mit Cityscapes. Bereits nach 30 Epochen kann es in dieser Konfiguration sehr gute Ergebnisse erzeugen.

Netzwerk	vocal folds	other tissue	glottal space	pathology	surgical tool	intubation	mean-IoU
ERFNet Cs	81.0 %	77.2 %	72.4 %	60.7 %	81.6 %	79.9 %	75.5 %
DL-Nbt Cs	76.5 %	74.9 %	67.6 %	73.1 %	84.1 %	81.9 %	76.3 %
DL-RN IN	81.5 %	77.0 %	71.1 %	59.8 %	80.9 %	82.5 %	75.4 %

Tabelle 6.4: Test-IoUs der von ERFNet und DL-Nbt nach dem Training auf dem Stimmband-Datensatz und Pre-Training auf Cityscapes (Cs) und ImageNet (IN).

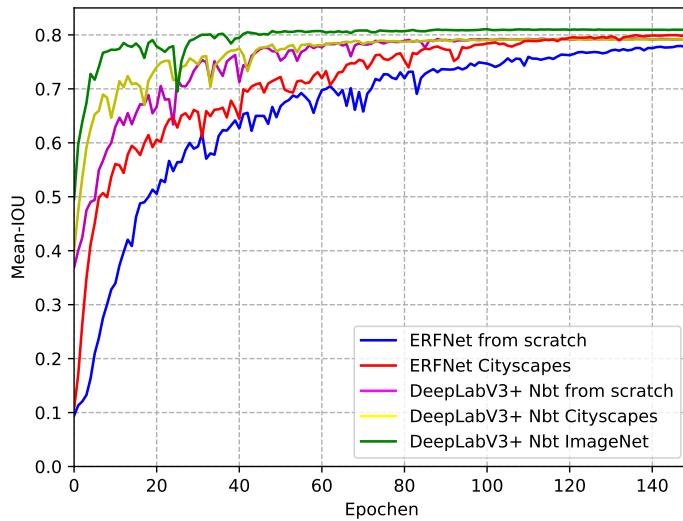


Abbildung 6.4: Validierungs-IoU von ERFNet und DeepLabV3+ Non-bt-1D während des Trainings auf dem Stimmband-Datensatz mit und ohne Pre-Training.

6.2.2 *Mitochondrien-Datensatz*

Bei dem Mitochondrien-Datensatz konnten die Ergebnisse durch das Pre-Training zum Teil verbessert werden, wie aus Tabelle 6.5 abzulesen ist. Die mean-IoU von U-Net wurde von 80.2 % auf 81.5 % erhöht. DL-Nbt mit ImageNet Pre-Training hat eine mean-IoU von 81.2 % erreicht, womit es fast so gut ist wie U-Net. Allerdings wurde die Qualität der Ergebnisse von DL-Nbt durch das Pre-Training auf Cityscapes reduziert.

Dadurch, dass nach dem Pre-Training keine zufällige Initialisierung mehr durchgeführt werden muss, konnte DeepLabV3+, anders als ohne Pre-Training, jedes mal ohne Probleme optimiert werden.

6.2.3 *Analyse*

Anhand der hier durchgeführten Experimente werden die größten Vorteile von Pre-Training sichtbar. Abbildung 6.4 zeigt deutlich wie stark das Training von neuronalen Netzen durch ein Pre-Training beschleunigt werden kann. Besonders wenn ein Netzwerk mehrfach trainiert werden soll, zum Beispiel um Hyperparameter zu testen, kann dadurch viel Zeit gespart werden, da das Pre-Training nur einmal durchgeführt werden muss und dadurch jedes folgende Training weniger Zeit kostet.

Ein weiterer großer Vorteil ist, dass es nach einem Pre-Training keine Probleme mehr mit

Netzwerk	background	mitochondria	mean-IoU
U-Net Cs	97.8 %	64.9 %	81.5 %
DL-Nbt Cs	96.6 %	46.9 %	73.0 %
DL-RN IN	97.7 %	64.9 %	81.2 %

Tabelle 6.5: Test-IoUs der von U-Net und DL-Nbt nach dem Training auf dem Mitochondrien-Datensatz und Pre-Training auf Cityscapes (Cs) und ImageNet (IN).

der Initialisierung der Parameter geben kann. Dies hat sich besonders bei dem Training von DL-Nbt auf dem Mitochondrien-Datensatz gezeigt. Ohne Pre-Training musste das Training sehr häufig neu gestartet werden, da es für den Optimierer mit einer schlechten Initialisierung nicht möglich war das Netzwerk zur Konvergenz zu bringen. Nach dem Pre-Training konnte das Netzwerk jedes mal ohne Probleme optimiert werden.

Der dritte Vorteil hat sich auch beim Training auf dem Mitochondrien-Datensatz gezeigt. Die Netzwerke konnten nach dem Pre-Training zum Teil bessere Endergebnisse erzielen, als bei einem Training nach zufälliger Initialisierung. Allerdings hat DL-Nbt nach dem Pre-Training auf Cityscapes schlechtere Ergebnisse erzielt als ohne. Das ist ein sehr überraschendes Ergebnis und sollte in zukünftigen Experimenten genauer untersucht werden.

6.3 Augmentierungen

Für die Augmentierungen wurden die vier, in Kapitel 5.2 vorgestellten, Verfahren genutzt. Bei dem Stimmband-Datensatz wurde zunächst jedes Augmentierungsverfahren einzeln angewendet und getestet. So sollte untersucht werden, welches Verfahren die größten Einflüsse hat. Danach wurden alle vier Methoden zusammen auf den Datensatz angewendet. Bei dem Mitochondrien-Datensatz wurde nur das Experiment mit allen Augmentierungen durchgeführt.

Wenn alle Augmentierungen genutzt wurden, wurden die Datensätze auf folgende Weise angepasst: alle originalen Bilder blieben im Datensatz erhalten. Jedes Bild wurde außerdem horizontal gespiegelt hinzugefügt. Danach wurden solange Bilder ergänzt bis der Datensatz insgesamt 4000 Bilder (3000 beim Mitochondrien-Datensatz) enthielt. Jedes der dabei hinzugefügten Bilder wurde mit 70 prozentiger Wahrscheinlichkeit zwischen 0 und ± 10 Grad (90 Grad beim Mitochondrien-Datensatz) rotiert, mit 70 prozentiger Wahrscheinlichkeit elastisch deformiert ($\alpha = 5000, \sigma = 100$) und mit 50 prozentiger Wahrscheinlichkeit heran gezoomt. Sollte bei einem Bild keines dieser Verfahren angewendet worden sein, was mit einer Wahrscheinlichkeit von 4.5 % zutraf, wurde es mit $\alpha = 5000$ und $\sigma = 80$ elastisch deformiert, wodurch eine starke Deformierung erzeugt wurde.

6.3.1 Stimmband-Datensatz

Die Ergebnisse der verschiedenen Augmentierungsmethoden sind in Tabelle 6.6 aufgelistet. Zunächst fällt auf, dass ERFNet in jedem Experiment bessere Ergebnisse erzielen konnte als DL-Nbt mit dem gleichen Verfahren. Allerdings ist das Ergebnis mit jedem Verfahren bei beiden Netzwerken besser als ohne Augmentierungen.

Von den einzelnen Methoden bringt die horizontale Spiegelung die geringste Verbesserung. ERFNet erreicht dabei 79.3 % mean-IoU und DL-Nbt 76.7 %. Das effektivste Verfahren sind Zooms, wobei 81 % bei ERFNet und 80.5 % mean-IoU bei DL-Nbt erreicht werden.

Werden alle vier Augmentierungen gemeinsam genutzt, können deutlich bessere Ergebnisse erreicht werden, als mit den einzelnen Verfahren. ERFNet erreicht 85.1 %, was eine

Aug. Methode	vocal folds	other tissue	glottal space	pathology	surgical tool	intubation	mean- IoU
ERFNet:							
Spiegelung	82.1 %	77.9 %	73.7 %	73.1 %	86.1 %	82.8 %	79.3 %
Rotation	83.7 %	80.0 %	76.0 %	74.1 %	88.2 %	81.3 %	80.6 %
Zoom	83.2 %	80.4 %	75.7 %	75.7 %	88.1 %	82.5 %	81.0 %
El. Def.	83.1 %	79.6 %	75.9 %	71.6 %	89.7 %	82.6 %	80.4 %
Alle	85.8 %	83.9 %	77.7 %	86.6 %	92.5 %	84.1 %	85.1 %
DL-Nbt:							
Spiegelung	78.5 %	75.8 %	68.6 %	68.3 %	85.7 %	83.4 %	76.7 %
Rotation	80.5 %	78.5 %	72.8 %	73.6 %	87.1 %	83.8 %	79.4 %
Zoom	81.2 %	79.5 %	74.2 %	75.3 %	87.6 %	85.5 %	80.5 %
El. Def.	80.8 %	78.9 %	72.8 %	74.5 %	86.7 %	82.8 %	79.4 %
Alle	83.4 %	80.9 %	75.5 %	85.3 %	91.7 %	85.7 %	83.8 %

Tabelle 6.6: Test-
IoUs der von ERFNet und DL-Nbt nach dem Training auf dem Stimmband-Datensatz mit verschiedenen Augmentierungen.

Steigerung von fast 10 Prozentpunkten im Gegensatz zum Training ohne Augmentierungen ist. DL-Nbt erreicht 83.8 % mean-
IoU

6.3.2 Mitochondrien-Datensatz

Auch bei dem Mitochondrien-Datensatz konnten die Ergebnisse beider Netzwerke durch das Nutzen der Augmentierungen deutlich verbessert werden. Wie in Tabelle 6.7 zu sehen ist, konnte U-Net einen Wert von 88.9 % mean-
IoU erreichen. Dies ist eine Verbesserung von sieben Prozentpunkten im Vergleich zum Training ohne Augmentierungen. DL-Nbt erreicht ein Ergebnis von 85.9 % mean-
IoU.

6.3.3 Analyse

Bei beiden Datensätzen haben die Augmentierungen eine sehr starke Verbesserung der Ergebnisse bewirkt. Dies zeigt wie wichtig das Nutzen von Augmentierungen ist, aber insbesondere auch, wie wichtig die Qualität der Datensätze für die Ergebnisse ist. Durch Augmentierungen ist es möglich, Datensätze, die zu klein sind oder deren Bilder sich zu stark ähneln, besser für neuronale Netze nutzbar zu machen.

Bei dem Stimmband-Datensatz konnten die Ergebnisse durch das Nutzen des Zooms am stärksten verbessert werden. In Abbildung 6.5 ist ein Beispielbild für die beiden Architekturen mit und ohne Zoom Augmentierung im Training dargestellt. Dieses Bild unterscheidet sich deutlich von einem großen Teil der restlichen Bilder im Datensatz. Die Features

Netzwerk	background	mitochondria	mean- IoU
U-Net	98.7 %	80.0 %	88.9 %
DL-Nbt	98.1 %	73.9 %	85.9 %

Tabelle 6.7: Test-
IoUs der von U-Net und DL-Nbt nach dem Training auf dem Augmentierten Mitochondrien-Datensatz.

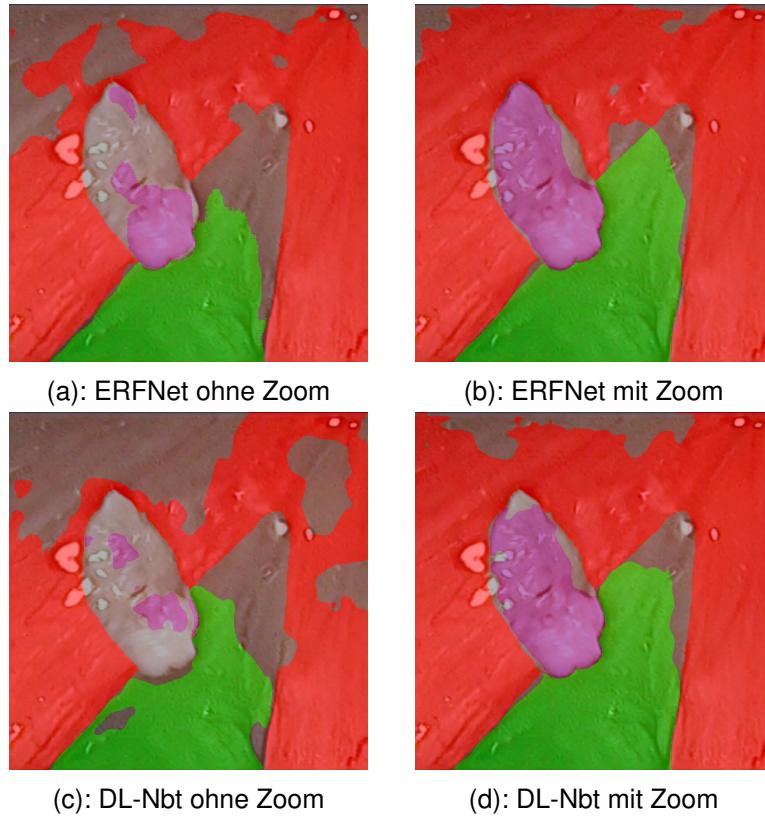


Abbildung 6.5: Ergebnisbilder aus dem Training ohne und mit der Zoom Augmentierung auf dem Stimmband-Datensatz [27].

sind in diesem Bild sehr viel größer, da das Endoskop bei der Aufnahme näher an der Halswand war, als bei den meisten anderen Bildern. Dadurch, dass es wenige solcher Bilder im Datensatz gibt, konnten sie nur sehr schlecht klassifiziert werden. Durch das Nutzen der Zooms zum Augmentieren des Datensatzes wurden einige weitere Bilder, in denen die Features ähnlich skaliert waren, hinzugefügt. Dadurch sind die Ergebnisse für diese Bilder deutlich besser ausgefallen, als ohne die Augmentierung.

Das zeigt, wie wichtig es ist Augmentierungen zu wählen, die zu dem Datensatz passen. Es ist also sehr sinnvoll die Datensätze genau zu analysieren und zu versuchen herauszufinden welche Art oder Konstellationen der Features zu wenig repräsentiert werden.

Noch sinnvoller als die Datensätze nur durch Augmentierungen zu erweitern, wäre es die Größe durch das Hinzufügen weiterer Bilder zu erhöhen. Durch das Nutzen von zusätzlichen Bildern, am besten von anderen Patienten, kann die Varianz der Daten deutlich besser erhöht werden als durch das Nutzen von Augmentierungen. Für den Fall, dass es sehr schwer ist an weitere Daten zu kommen, so wie in den meisten Bereichen der Medizin, können Augmentierungen zumindest eine Hilfe sein, um die Daten künstlich zu erweitern.

Insbesondere für die *pathology* Klasse sollten noch weitere Beispiele zu dem Stimmband-Datensatz hinzugefügt werden. Durch die Augmentierungen kann das Aussehen dieser Klasse nur sehr begrenzt verändert werden. Die Ergebnisse wirken für diese Klasse mit 86.6 % IoU zwar schon relativ gut, aber dabei muss bedacht werden, dass dafür vermut-

lich Overfitting verantwortlich ist, da jedes Vorkommen der Klasse in dem Datensatz sehr ähnlich aussieht.

6.4 Ensemble Netzwerke

In dieser Arbeit werden die Ensemble Netzwerke aus mehreren verschiedenen Netzwerken zusammengesetzt. Wie in [51] beschrieben, ist es nicht immer am sinnvollsten so viele Netzwerke wie möglich in ein Ensemble einzubinden. Deshalb wurde auch in den folgenden Experimenten nicht jede Architektur genutzt.

SegNet hat in den Experimenten von Abschnitt 6.1 bei jedem betrachteten Kriterium schlechter abgeschnitten als die anderen Architekturen, weshalb es für die Ensembles nicht verwendet wurde.

Der einzige Vorteil den E-Net gegenüber den anderen Netzwerken hat, ist die geringere Parameteranzahl. Ein Ensemble Netzwerk besteht aber in jedem Fall aus sehr vielen Parametern, weshalb es nicht viel Sinn macht dieses Kriterium zu berücksichtigen. Da E-Net keinen anderen Vorteil gegenüber den anderen Architekturen hat, wurde es auch nicht in den Ensembles genutzt.

Da sich die einzelnen Varianten von DeepLabV3+ nur relativ gering unterscheiden, würde es keinen großen Vorteil bringen mehrere Versionen davon in dem Ensemble zu nutzen. Von den drei Versionen hat die Non-bt-1D Variante am besten abgeschnitten. Daher ist DL-Nbt eine der Architekturen, die für die Ensemble Netzwerke genutzt wurden.

ERFNet hat bei dem Stimmband-Datensatz sehr gute Ergebnisse geliefert. Mit Augmentierungen war es sogar besser als DL-Nbt, weshalb es auch in den Ensembles eingesetzt wurde.

Des Weiteren wurde U-Net genutzt, da es die Besonderheit hat sehr viele skip connections zu nutzen, wodurch es ein wenig besser darin ist Konturen wiederherzustellen als zum Beispiel ERFNet. Bei dem Mitochondrien-Datensatz konnte U-Net dadurch sogar die besten Ergebnisse erzeugen.

Im Folgenden wurde jede Kombination dieser drei Architekturen als Ensemble Netzwerk getestet.

6.4.1 Stimmband-Datensatz

In Tabelle 6.8 sind die Ergebnisse der Experimente auf dem Stimmband-Datensatz aufgelistet. Alle drei Architekturen zusammen im Ensemble zu verwenden, hat mit 86.7 % mean- IoU die besten Resultate gebracht. ERFNet + DL-Nbt liegt mit 86.3 % knapp dahinter. U-Net nur gemeinsam mit einem der anderen Netzwerke zu nutzen, hat nicht so gute Ergebnisse gebracht. Die Kombination zwischen U-Net und ERFNet hat mit 84.5 % sogar schlechtere Ergebnisse gebracht als ERFNet alleine, was 85.1 % mean- IoU erreichen konnte.

Netzwerke	vocal folds	other tissue	glottal space	pathology	surgical tool	intubation	mean-IoU
ERFNet + DL-Nbt	86.2 %	84.5 %	79.3 %	89.1 %	92.9 %	85.8 %	86.3 %
U-Net + DL-Nbt	86.7 %	81.2 %	78.6 %	78.7 %	92.0 %	85.5 %	83.8 %
ERFNet + U-Net	87.1 %	81.8 %	78.6 %	82.3 %	92.4 %	84.8 %	84.5 %
Alle	87.3 %	85.3 %	79.9 %	88.2 %	93.9 %	85.8 %	86.7 %

Tabelle 6.8: Test-IoUs der verschiedenen Ensemble Architekturen nach dem Training auf dem Stimmband-Datensatz.

6.4.2 Mitochondrien-Datensatz

Die Ergebnisse der Ensembles für den Mitochondrien-Datensatz sind in Tabelle 6.9 aufgelistet. Die besten Ergebnisse konnten die Ensembles U-Net + DL-Nbt und ERFNet + U-Net mit 88.9 % mean-IoU erreichen. Alle drei Architekturen zu verwenden, hat mit 88.9 % ein etwas schlechteres Ergebnis erzielt. Das Ensemble, das U-Net nicht verwendet hat, lieferte die schlechtesten Ergebnisse.

6.4.3 Analyse

Das Nutzen der Ensembles konnte bei beiden Datensätzen die Ergebnisse noch einmal ein wenig verbessern. Bei dem Stimmband-Datensatz hat das Ensemble mit allen drei Netzwerken die besten Ergebnisse geliefert. Das U-Net hat dabei allerdings nur eine sehr kleine Verbesserung gebracht, denn das Ensemble aus ERFNet und DL-Nbt liegt nur mit einem Unterschied von 0.4 Prozentpunkten mean-IoU dahinter. Es scheint dort wichtiger zu sein, dass beide der besten Architekturen verwendet werden.

Bei dem Mitochondrien-Datensatz hat im Gegensatz dazu das Ensemble mit allen Netzwerken nicht am besten abgeschnitten. Beide Netzwerke, die U-Net und eines der anderen Netzwerke genutzt haben, konnten die besten Ergebnisse erzielen. Das Ensemble in dem U-Net nicht enthalten war, hat mit 87.6 % deutlich schlechter abgeschnitten. Insgesamt wurde bei diesem Datensatz durch die Ensembles auch nur eine Verbesserung von 0.8 Prozentpunkten im Vergleich zum U-Net alleine erreicht. Bei dem Stimmband-Datensatz haben die Ensembles eine Verbesserung von 1.6 Prozentpunkten mean-IoU gebracht. Das verdeutlicht, dass die U-Net Architektur für den Mitochondrien-Datensatz bereits sehr gut geeignet ist und durch die anderen Architekturen nicht stark verbessert

Netzwerke	background	mitochondria	mean-IoU
ERFNet + DL-Nbt	98.4 %	77.1 %	87.6 %
U-Net + DL-Nbt	98.8 %	81.0 %	89.7 %
ERFNet + U-Net	98.8 %	80.9 %	89.7 %
Alle	98.6 %	79.4 %	88.9 %

Tabelle 6.9: Test-IoUs der verschiedenen Ensemble Architekturen nach dem Training auf dem Mitochondrien-Datensatz.

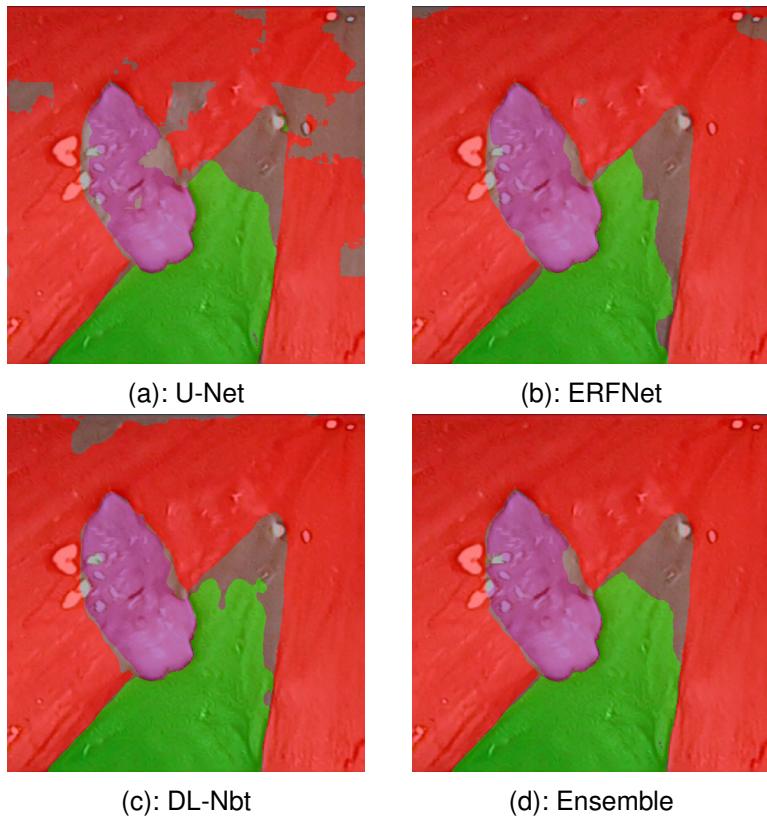


Abbildung 6.6: Ergebnisbilder aus dem Training der einzelnen Netzwerke und ihrem Ensemble auf dem Stimmband-Datensatz [27].

werden kann.

In Abbildung 6.6 ist ein Beispielbild, das mit den verschiedenen Netzwerken und ihrem Ensemble klassifiziert wurde, dargestellt. Es ist relativ gut sichtbar, dass das Ensemble eine Kombination der Ergebnisse nutzt. Bei genauerer Betrachtung des oberen Bereichs fällt auf, dass die rechte Ecke von U-Net und ERFNet nicht korrekt klassifiziert wurde. DeepLab erzeugt dagegen in dem Bereich korrekte Ergebnisse. Dafür hat DeepLab in der Mitte des oberen Bereichs Fehler gemacht, die von U-Net und ERFNet nicht gemacht wurden. In dem Ensemble ist der gesamte obere Bereich korrekt klassifiziert. Allerdings fällt auch auf, dass Bereiche, die in allen drei Netzwerken falsch waren, auch im Ensemble falsch sind, da es, abgesehen von der Gewichtung der einzelnen Unternetzwerke, nicht mehr trainiert wird. Deshalb ist es wichtig für Ensembles Netzwerke zu wählen, die unterschiedliche Stärken haben.

7

KAPITEL

Fazit und Ausblick

7.1 Fazit	67
7.2 Ausblick	69

7.1 Fazit

In dieser Arbeit wurden fünf Netzwerke zur semantischen Segmentierung implementiert und anhand von zwei medizinischen Datensätzen getestet. Die Netzwerke waren U-Net, SegNet, E-Net, ERFNet und DeepLabV3+. Der erste Datensatz bestand aus Endoskopbildern von Stimmbändern und wurde bereits durch ähnliche Experimente untersucht [27]. Diese Experimente wurden rekonstruiert und durch das Nutzen der DeepLabV3+ Architektur und umfangreicherer Augmentierungen erweitert. Dadurch konnten die Ergebnisse von 84.7 % auf 86.7 % mean-IoU verbessert werden. Dieses Ergebnis wurde durch ein Ensemble von U-Net, ERFNet und DeepLabV3+ erzielt, welches zunächst auf dem Cityscapes-Datensatz vorgenommen und danach auf einer augmentierten Version des Stimmband-Datensatzes verfeinert wurde. In 7.1 sind einige Ergebnisse, die von dieser

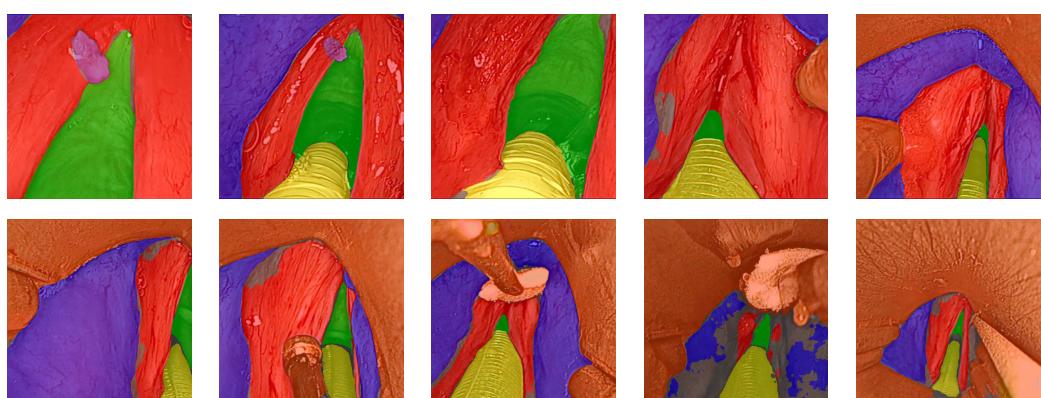


Abbildung 7.1: Ergebnisbilder der besten Architektur aus dem Stimmband-Datensatz [27].

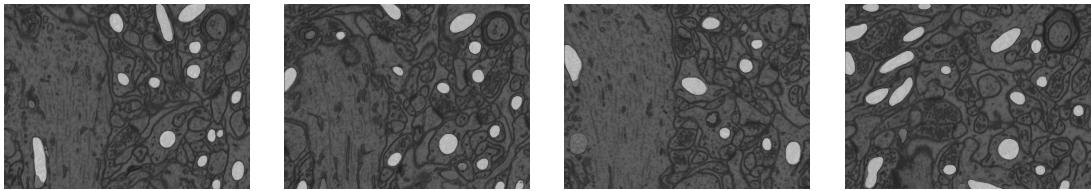


Abbildung 7.2: Ergebnisbilder der besten Architektur aus dem Mitochondrien-Datensatz [32].

Architektur erzeugt wurden, abgebildet. Das beste einzelne Netzwerk war auf diesem Datensatz ERFNet. Dieses konnte nach Pre-Training auf Cityscapes und Training mit Augmentierungen 85.1 % mean-IoU erreichen.

Des Weiteren wurden jegliche Experimente auf einem zweiten Datensatz durchgeführt, der aus Elektronenmikroskop-Aufnahmen von Mitochondrien besteht [32]. Dieser Datensatz hat eine geringere Komplexität, da er weniger Klassen enthält und sich die Bilder geringer unterscheiden. Es hat sich gezeigt, dass dadurch andere Anforderungen an die Netzwerke gestellt werden. Da die allgemeine Klassifizierung auf diesem Datensatz leichter ist als auf dem Stimmband-Datensatz, ist die Hauptschwierigkeit die Rekonstruktion der genauen Konturen der Objekte. Netzwerke, die den Informationsverlust in ihrer Architektur besser reduzieren, sind hierfür besser geeignet. Dadurch konnte das U-Net auf diesem Datensatz durch die Nutzung von skip connections bessere Ergebnisse liefern, als alle anderen Netzwerke alleine. Das beste Ergebnis auf diesem Datensatz wurde durch ein Ensemble von U-Net und ERFNet beziehungsweise U-Net und DL-Nbt erzielt. Auch hierfür wurden Augmentierungen und Pre-Training eingesetzt. Einige Beispieldaten dieser Netzwerke sind in 7.2 abgebildet.

Bei DeepLabV3+ wurden verschiedene Varianten der Architektur verglichen. Zwei Varianten wurden in der originalen Veröffentlichung [8] beschrieben und eine dritte, die Blöcke aus der ERFNet Architektur nutzt, wurde in dieser Arbeit entwickelt. Diese Non-bt-1D Variante hat in den Experimenten von den drei Versionen die besten Ergebnisse geliefert. Trotzdem war DeepLabV3+ bei beiden Datensätzen jeweils nur die zweitbeste Architektur. Das ist überraschend, da es bei anderen Datensätzen deutlich bessere Ergebnisse liefern kann, als alle anderen hier verwendeten Architekturen [10, 13]. Dies lässt vermuten, dass DeepLabV3+ nicht so gut für medizinische Daten geeignet ist wie Architekturen, die spezifisch dafür konzipiert wurden, wie zum Beispiel das U-Net.

In den Experimenten konnte zudem die Wirkung von den drei, in Kapitel 5 vorgestellten, Methoden gezeigt werden. Pre-Training hat die Konvergenz der Netzwerke während des Trainings stark beschleunigt und konnte bei dem Mitochondrien-Datensatz sogar die Qualität der Netzwerke verbessern. Außerdem ist es hilfreich, um Probleme bei der Initialisierung zu lösen, wie sich bei DeepLabV3+ gezeigt hat.

Die Augmentierung der Datensätze hat die größten Verbesserungen gebracht. Bei ERFNet konnten die Ergebnisse des Trainings auf dem Stimmband-Datensatz dadurch zum Beispiel von 75.5 % auf 85.1 % mean-IoU erhöht werden. Es hat sich auch gezeigt, dass es sinnvoll ist den Datensatz genau zu untersuchen, um die sinnvollsten Augmentierungsverfahren zu finden. Bei dem Stimmband-Datensatz war das zum Beispiel ein Zoom, da

es einige Bilder in dem Datensatz gab, die bereits stark vergrößert im Vergleich zu anderen Bildern waren. Diese Bilder konnten durch die Nutzung des Zooms deutlich besser klassifiziert werden.

Das letzte getestete Verfahren waren die Ensemble Strukturen. Indem die Stärken der verschiedenen Architekturen gemeinsam genutzt wurden, konnten die finalen Ergebnisse noch weiter verfeinert werden. Die Ensemble Netzwerke konnten die mean-IoU um weitere 1.6 Prozentpunkte bei dem Stimmband-Datensatz und 0.8 Prozentpunkte bei dem Mitochondrien-Datensatz erhöhen.

7.2 Ausblick

In dieser Arbeit konnten bereits sehr gute Ergebnisse für die semantische Segmentierung der beiden Datensätze erzeugt werden. Vorherige Ergebnisse beider Datensätze [27, 32] konnten durch die hier genutzten Methoden und Architekturen übertroffen werden. Trotzdem gibt es noch viele weitere Verfahren, wodurch die Ergebnisse eventuell weiter verbessert werden könnten. Dazu gehören zum Beispiel CRFs [26], wie sie in den ersten Versionen von DeepLab genutzt wurden. Diese haben zwar größere Laufzeitkosten, aber können unter Umständen die Ergebnisse eines neuronalen Netzes noch weiter verfeinern.

Des Weiteren könnte es sich lohnen, die Architekturen noch weiter zu verbessern. In dieser Arbeit wurde bereits versucht DeepLabV3+ und ERFNet zu kombinieren, was zu der DL-Nbt Architektur geführt hat, die bessere Ergebnisse liefern konnte als die anderen DeepLabV3+ Varianten. Durch weitere Kombinationen der Architekturen ist es vielleicht möglich ihre unterschiedlichen Stärken besser gemeinsam zu nutzen. Da ERFNet auf dem Stimmband-Datensatz mit Augmentierungen die besten Ergebnisse geliefert hat, könnte diese Architektur als Grundlage dienen. DeepLabV3+ konnte im Training deutlich schneller konvergieren. Es wäre interessant zu prüfen, ob eine Variante von ERFNet, die das ASPP von DeepLabV3+ nutzt, auch schneller konvergieren könnte. Außerdem hat sich gezeigt, dass U-Net durch die skip connections besser in der Lage war, Konturen von Objekten zu rekonstruieren. ERFNet könnte auch um skip connections erweitert werden, um die Präzision weiter zu verbessern. Durch solche Erweiterungen würden natürlich die Laufzeit- und Speicherkosten der Architektur ansteigen.

Einige Ergebnisse der hier durchgeführten Experimente waren auch sehr überraschend und könnten in zukünftigen Experimenten noch ausführlicher untersucht werden. Zum einen hat die Xception Version von DeepLabV3+ in dieser Arbeit immer relativ schlechte Ergebnisse geliefert, obwohl sie in der originalen Veröffentlichung am besten war [8]. Außerdem konnte DeepLabV3+ nach dem Pre-Training auf dem Cityscapes-Datensatz nur deutlich schlechtere Ergebnisse für den Stimmband-Datensatz erzeugen als ohne Pre-Training. Theoretisch sollte durch Pre-Training die Qualität der Ergebnisse niemals schlechter werden, da jeder unpassende Parameter beim Verfeinern auf dem eigentlichen Datensatz genauso verbessert werden kann wie ohne Pre-Training. Deshalb wäre es sinnvoll zu untersuchen, was die Ursache für dieses Verhalten war.

KAPITEL 7. FAZIT UND AUSBLICK

In Kapitel 4 wurde auch bereits angedeutet, dass der Stimmband-Datensatz einige Probleme hat, wodurch die Ergebnisse verfälscht wurden sein könnten. Zum einen gibt es einige Bilder in den Evaluierungsabschnitten des Datensatzes, die sehr ähnlich zu Bildern aus dem Trainingssatz sind. Durch solche Bilder kann die Qualität der Ergebnisse besser wirken als sie eigentlich ist, da die Bilder durch Overfitting sehr gut klassifiziert werden können. Zum anderen gibt es in dem ganzen Datensatz nur ein sichtbares Objekt der *pathology* Klasse, welches in vielen Bildern vorkommt. Dadurch könnten die Ergebnisse dieser Klasse besonders stark verfälscht sein. Um die Einflüsse dieser Probleme zu untersuchen, müsste der Datensatz erweitert werden. Durch eine Erweiterung könnten die Ergebnisse vermutlich auch noch stark verbessert werden, weil der Datensatz relativ klein ist und größere Datensätze besser für das Training von neuronalen Netzen geeignet sind [10]. Am besten wäre es, wenn Bilder von sehr vielen verschiedenen Patienten vorliegen würden, da sich die Bilder eines Patienten untereinander sehr ähnlich sehen. Eine höhere Varianz der Bilder wäre sehr hilfreich, damit die trainierten Netzwerke besser verallgemeinerte Features lernen könnten.

Literaturverzeichnis

- [1] Machine learning: Authors and titles for recent submissions. <https://arxiv.org/list/cs.LG/pastweek?show=25>. Zugriffen: 12.04.2019.
- [2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. 2016.
- [4] Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. 2016.
- [5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. 2016.
- [6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. 2017.
- [7] Liang-Chieh Chen, George Papandreou and Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. 2017.
- [8] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. 2018.

- [9] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. 2017.
- [10] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. 2016.
- [11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR09*, 2009.
- [12] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. 2018.
- [13] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. <http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html>. Zugegriffen: 12.4.2019.
- [14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feed-forward neural networks. 2010.
- [15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. 2011.
- [16] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. 2018.
- [17] Lars Kai Hansen and Peter Salamon. Neural network ensembles. 1990.
- [18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2015.
- [19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015.
- [20] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. 2012.
- [21] Tin Kam Ho. Random decision forests. 1995.
- [22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.
- [23] Dilpreet Kaur and Yadwinder Kaur. Various image segmentation techniques: A review. 2014.
- [24] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. 2015.
- [25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. 2012.

- [26] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. 2012.
- [27] Max-Heinrich Laves, Jens Bicker, Lüder A. Kahrs, and Tobias Ortmaier. A dataset of laryngeal endoscopic images with comparative study on convolution neural network based semantic segmentation. 2018.
- [28] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.
- [29] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist dataset of handwritten digits. 1999.
- [30] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, and Clara I. Sanchez. A survey on deep learning in medical image analysis. 2017.
- [31] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. 2015.
- [32] Arélien Lucchi, Yungpeng Li, and Pascal Fua. Learning for structured prediction using approximate subgradient descent with working sets. 2013.
- [33] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman and Hall/CRC, 2009.
- [34] Georgia A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38, 1995.
- [35] I. Rish. An empirical study of the naive bayes classifier. 2001.
- [36] Eduardo Romera, José M. Álvarez, Luis M. Bergasa, and Roberto Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. 2017.
- [37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. 2015.
- [38] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. 1958.
- [39] Sebastian Ruder. An overview of gradient descent optimization algorithms. 2016.
- [40] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [41] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best practices for convolutional neural networks applied to visual document analysis. 2003.

LITERATURVERZEICHNIS

- [42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.
- [43] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. 2014.
- [44] George Stockman and Linda G. Shapiro. *Computer Vision*. Prentice Hall, 2001.
- [45] Nima Tajbakhsh, Jae Y. Shin, Suryakanth R. Gurudu, R. Todd Hurst, Christopher B. Kendall, Michael B. Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? 2016.
- [46] Martin Thoma. A survey of semantic segmentation. 2016.
- [47] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. 2015.
- [48] Jason Wang and Luis Perez. The effectiveness of data augmentation in image classification using deep learning. 2017.
- [49] Sebastien C. Wong, Adam Gatt, Victor Stamatescu, and Mark D. McDonnell. Understanding data augmentation for classification: when to warp? 2016.
- [50] Fischer Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. 2016.
- [51] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: Many could be better than all. 2002.