technische universität
dortmund

Semantische Segmentierung mit Deep Convolutional Neural Networks

Masterarbeit

Kai Brandenbusch
12. November 2018

Betreuer:
Eugen Rusakov, M.Sc.
Prof. Dr.-Ing. Gernot A. Fink

Fakultät für Informatik
Technische Universität Dortmund
http://www.cs.uni-dortmund.de

INHALTSVERZEICHNIS

1 EINLEITUNG

Eine zentrale Problemstellung des maschinellen Sehens ist das Verstehen von Szenen, welche auf Bildern oder Videos erfasst wurden [GGOEO+17]. Die semantische Segmentierung stellt einen Schritt zu einem solchen Verständnis von Bildinhalten dar und hat das Ziel, ein Bild in sinnvolle Regionen zu unterteilen und diesen die Klasse des jeweils enthaltenen Objektes zuzuordnen [SK14]. Eine detaillierte Segmentierung kann beispielsweise in Bereichen wie dem autonomen Fahren verwendet werden, um bei der Erkennung der Fahrbahn, Fahrbahnmarkierungen, Schilder und Verkehrszeichen zu helfen. Zudem ist es möglich, Hindernisse wie Fußgänger oder andere Autos zu lokalisieren und Informationen zu deren Größe zu erhalten. Eine weitere Anwendungsmöglichkeit ist die Verwendung zur Erkennung der Umgebung im Kontext von Augmented-Reality-Anwendungen. Zur Segmentierung der Regionen wird bei der semantischen Segmentierung eine Vorhersage von Klassen für jedes Pixel eines gegebenen Bildes berechnet. Dies entspricht somit einer Erweiterung der Klassifikation, deren Ziel eine Klassenvorhersage für das gesamte Bild ist [GGOEO+17]. Durch die Verwendung von tiefen neuronalen Faltungsnetzen (engl. deep convolutional neural networks) konnten bei Problemen des maschinellen Sehens große Fortschritte erreicht werden [HKH17]. Beispielsweise konnte bei der Klassifikation von Bildern mit tiefen Faltungsnetzen zum Teil die menschliche Fehlerrate bei diesem Problem unterschritten werden [HZRS15]. Tiefe Faltungsnetze, welche zur Klassifikation entwickelt wurden, können als Basis für Architekturen zur semantischen Segmentierung verwendet werden. Dazu sind jedoch Veränderungen dieser Architekturen erforderlich, die es ermöglichen, eine pixelweise Klassenvorhersage zu berechnen.

Ein Problem der Verwendung tiefer neuronaler Faltungsnetze zur semantischen Segmentierung ist die große Menge erforderlicher annotierter Trainingsdaten [HKH17]. Die Erstellung qualitativ guter Annotationen erfordert eine manuelle, möglichst pixelgenaue Segmentierung des Bildinhalts. In Folge dessen ist der Aufwand zur Erstellung vollständiger Annotationen sehr hoch, was dazu führt, dass die Anzahl und der Umfang der verfügbaren Datensätze eingeschränkt ist. Insbesondere bei Anwendungsgebieten, welche domänenspezifisches Expertenwissen erfordern, ist der hohe Annotationsaufwand ein großes Problem. Neben vollständigen Annotationen existieren auch schwächere Formen der Annotation. Bild-Level-Annotationen, welche zum Training eines Klassifikators verwendet werden, Punktannotationen, Bounding-

Box-Annotationen oder Krakelannotationen können mit einem wesentlich geringeren Aufwand erstellt werden. Im Vergleich zu vollständigen Annotationen fehlen jedoch, abhängig von der Annotationsform, Informationen zu Position und Ausdehnung der Objekte auf dem Bild. Um einen Teil dieser Informationen zu erhalten, kann Vorwissen zur visuellen Erscheinung, Position oder Ausdehnung von Objekten der zu segmentierenden Klassen genutzt werden. Die Verwendung schwacher Annotationen und die Berücksichtigung weiterer Informationen erfordert eine Anpassung des Trainings, um dieses auch mit dem deutlich verringerten Informationsgehalt der Annotation durchführen zu können.

Im Rahmen dieser Arbeit werden zwei Faltungsnetze zur semantischen Segmentierung untersucht, welche mit unterschiedlichen Maßnahmen zur pixelweisen Klassenvorhersage angepasst wurden. Die Modelle werden zuerst mit vollständig annotierten und anschließend mit schwach annotierten Daten trainiert. Als schwache Annotationen werden Bild-Level-Annotationen, Punktannotationen und die Kombination von Punktannotationen mit Objectness als zusätzliche Information verwendet und das Training an die Annotationsformen angepasst. Der Einfluss der Annotationsform auf die Qualität der berechneten Segmentierung der Modelle wird untersucht und in Relation zum Annotationsaufwand gestellt. Zudem wird der Einfluss der Lernrate und der Augmentierung der Trainingsdaten bei der Verwendung schwacher Annotationen betrachtet.

Der verbleibende Teil der Arbeit ist wie folgt strukturiert. In Kapitel 2 werden zunächst das Problem der Klassifikation und der semantischen Segmentierung formal beschrieben. Anschließend werden der Aufbau neuronaler Netze und die Methode zum Training dieser Modelle erläutert und häufig verwendete Modelle zur Klassifikation beschrieben. Als spezielle Form der neuronalen Netze werden Faltungsnetze und dort verwendete Operationen diskutiert. In Kapitel 3 werden Arbeiten diskutiert, welche die Grundlage dieser Arbeit bilden. Zu diesem Zweck werden verschiedene Architekturen für die semantische Segmentierung beschrieben. Des Weiteren werden verschiedene Formen der Annotation, der Aufwand zu deren Erstellung und mögliche Verwendungen beim Training neuronaler Netze erläutert. Die Auswahl der in dieser Arbeit verwendeten Modelle wird in Kapitel 4 diskutiert und es wird erklärt, wie diese im Rahmen der Arbeit mit schwachen Annotationen trainiert werden. Der Aufbau und die Ergebnisse der Experimente zur Evaluation der Modelle werden in Kapitel 5 präsentiert. Abschließend werden in Kapitel 6 die Schlussfolgerungen aus den Experimenten zusammengefasst.

2 GRUNDLAGEN

In diesem Kapitel werden die Grundlagen für diese Arbeit erläutert. Zunächst wird in Abschnitt 2.1 eine Einführung in das Problem der Klassifikation und der semantischen Segmentierung gegeben. Anschließend wird in Abschnitt 2.2 beschrieben, wie künstliche neuronale Netze als Klassifikatoren verwendet werden können. In Abschnitt 2.3 wird das Verfahren zum Training neuronaler Netze und mögliche Schwierigkeiten bei dem Training dargestellt. Als eine spezielle Form der neuronalen Netze werden die Faltungsnetze in Abschnitt 2.4 erläutert. In Abschnitt 2.5 werden Beispiele für neuronale Netze zur Klassifikation beschrieben.

2.1 GRUNDBEGRIFFE DER MUSTERERKENNUNG

Die *Klassifikation* ist ein Problem aus dem Gebiet der *Mustererkennung* [Nie83]. Diese befasst sich mit den mathematisch-technischen Aspekten der (vor allem menschlichen) Perzeption und der automatischen Verarbeitung und Auswertung von *Mustern*. Muster sind die Elemente eines *Problemkreises*, welcher nur Objekte (Funktionen) eines bestimmten und begrenzten Anwendungsgebiets enthält. Beispiele für Muster sind Audiosignale, wie Sprache, oder auch Bilder, welche im Fokus dieser Arbeit stehen. Ein Problemkreis ist vollständig und disjunkt in *Klassen* unterteilt. Das Ziel einer Klassifikation besteht darin, einem gegebenen Muster unter holistischer Betrachtung einer Klasse zuzuordnen. Informationen über den Problemkreis sind als *repräsentative Stichprobe* gegeben. Die Stichprobe ist eine Menge, bestehend aus Paaren von einem Muster und der Zuordnung des Musters zu einer Klasse, welche als Annotation bezeichnet wird. Diese Paare werden im Folgenden auch als *Trainingspaare*/*Trainingsbeispiele* und die Stichprobe als *Trainingsdaten* bezeichnet. Das Training eines Systems zur Klassifikation erfolgt meist durch *Optimierung* einer geeigneten Zielfunktion bzw. Gütefunktion basierend auf den Trainingsdaten.

Muster wie Bilder bestehen meistens aus *Regionen* [Sun17]. Eine Region hat nach definierten Maßen Ähnlichkeiten, wie z. B. ähnliche Farben. Die disjunkte Unterteilung eines Bildes in Regionen wird als *Segmentierung* bezeichnet. Die grundlegenden Ansätze zur Berechnung einer Segmentierung ist das Finden der Grenzen zwischen Regionen oder die Ermittlung des Inneren der Regionen. Eine Region wird dazu über

ein *Homogenitätskriterium* definiert, welches für alle Pixel einer Region zutreffen muss und für die Pixel benachbarter Regionen nicht zutreffen darf.

Ziel der semantischen Segmentierung ist hingegen eine Unterteilung der Bildes in sinnvolle Regionen, welche Objekte enthalten, und die Zuweisung einer Klasse zu jeder Region [SK14]. Die semantische Segmentierung stellt somit eine Erweiterung der Klassifikation dar. Anstelle von globalen Vorhersagen für Bilder bei holistischer Betrachtung, ist das Ziel der semantischen Segmentierung eine fein aufgelöste Vorhersage von Klassen [GGOEO+17]. Wie auch zur Klassifikation ist eine Menge von n Klassenlabels $\mathcal{K} = \{k_1, k_2, \ldots, k_n\}$ gegeben. Die Eingabe liegt i. A. in Form einer Menge von Zufallsvariablen $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$ vor, deren Elemente im Fall eines Bildes in einem zweidimensionalen Pixelgitter der Größe $H \times B = N$ angeordnet sind. Ziel der Klassifikation ist es, dem Muster $\mathcal{X}$ als Ganzes betrachtet, eine Klasse in Form eines Klassenlabels $k \in \mathcal{K}$ zuzuweisen. Ziel der semantischen Segmentierung ist es, jedem Element des Musters $\mathcal{X}$, im Fall des Bildes also jedem Pixel x_i, eine Klasse k_i zuzuweisen, und somit eine *dichte Klassifikationskarte* zu erhalten. Architekturen von z. B. Long et al. [LSD15] oder Yu et al. [YK15] geben zur Berechnung dieser Karte eine Feature-Map mit Klassenwahrscheinlichkeiten mit einer Größe von $H \times B \times n$ aus, wobei $H \times B$ der Größe des Bildes und n der Anzahl der Klassenwahrscheinlichkeiten für jedes Pixel entsprechen. Analog zur Stichprobe bei der Klassifikation wird zum Training neuronaler Netze zur semantischen Segmentierung eine Stichprobe mit Annotationen für jedes Pixel benötigt. Abbildung 2.1.1b zeigt ein Beispiel für eine dichte Klassifikationskarte, welche als Annotation des Bildes 2.1.1a dient.

(a) Eingabebild (b) Klassifikationskarte

Abbildung 2.1.1: Beispiel für eine dichte Klassifikationskarte als Annotation zur semantischen Segmentierung [EVGW+12]. Das Eingabebild (a) zeigt Fahrradfahrer, welche in Menschen und Fahrräder zu segmentieren sind. Auf der dichten Klassifikationskarte (b) ist jedem Pixel eine Klasse zugeordnet, welche farblich gekennzeichnet ist.

2.2 KÜNSTLICHE NEURONALE NETZE

Eine beliebte Methode des maschinellen Lernens sind künstliche neuronale Netze [Agg18]. Diese können beispielsweise zur Klassifikation verwendet werden und erreichen bei diesem Problem teilweise geringere Fehlerraten als Menschen [HZRS15]. Die berechnenden Einheiten in neuronalen Netzen sind die *Neuronen*. Diese sind in *Schichten* organisiert. In diesem Abschnitt wird zunächst der Aufbau eines Perzeptrons als einfachste Form eines neuronalen Netzes mit nur einer Schicht erklärt. Anschließend werden komplexere neuronale Netze mit mehreren Schichten erläutert.

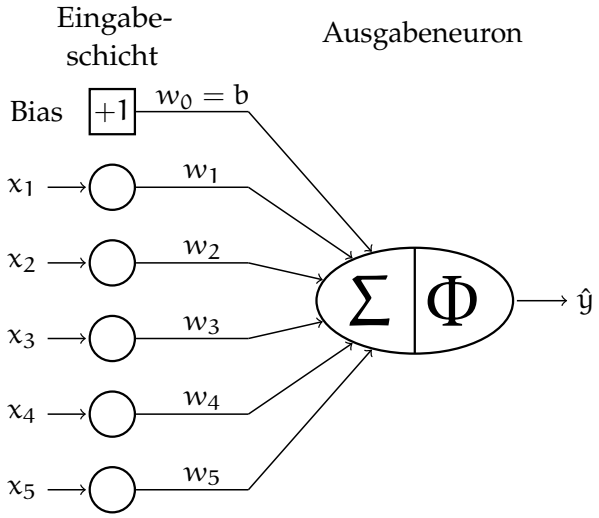


Abbildung 2.2.1: Beispiel eines Perzeptrons mit Bias [Agg18].

Die Weiterentwicklung erster Modelle, welche logische Funktionen mit binären Ausgaben berechneten [MP43], durch Frank Rosenblatt [Ros58] führte zu dem Perzeptron als Modell eines neuronalen Netzes. Das Perzeptron ist die einfachste Form eines neuronalen Netzes [Agg18]. Abbildung 2.2.1 zeigt ein Perzeptron mit fünf Eingabeneuronen und einem Ausgabeneuron. Die Eingabeneuronen dienen ausschließlich der Repräsentation einer d-dimensionalen Eingabe $\mathbf{x} = [x_1, \ldots, x_d]^T$ und führen keine Berechnungen durch. Aus diesem Grund wird die Eingabeschicht bei der Anzahl der Schichten in einem neuronalen Netz nicht berücksichtigt. Das Ausgabeneuron berechnet zuerst eine gewichtete Summe über den Eingaben, auf die anschließend eine

Aktivierungsfunktion $\Phi(\cdot)$ angewendet wird. Daraus folgt die Berechnungsvorschrift der Ausgabe:

$$\hat{y} = \Phi\left(\sum_{i=1}^{d} w_i x_i\right). \tag{2.2.1}$$

Das Ausgabeneuron hat zur Berechnung der gewichteten Summe ein Gewicht w_i pro Eingabekomponente x_i, welches im Training (vgl. Abschnitt 2.3) angelernt wird.

Um von den Eingaben unabhängige Anteile in der durch das Perzeptron berechneten Funktion modellieren zu können, wird die gewichtete Summe um die Addition eines *Bias* b erweitert:

$$\hat{y} = \Phi\left(\sum_{i=1}^{d} w_i x_i + b\right) \tag{2.2.2}$$

Wie in Abbildung 2.2.1 dargestellt, wird der Bias als Eingabeneuron mit konstanter Eingabe 1 und Multiplikation mit einem zusätzlichen Gewicht w_0 des Ausgabeneurons zur Vereinfachung der Berechnungsvorschrift zu

$$\hat{y} = \Phi\left(\sum_{i=0}^{d} w_i x_i\right) \tag{2.2.3}$$

mit $\mathbf{x} = [1, x_1, \ldots, x_d]^T$ und $w_0 = b$ modelliert. Der Unterschied zu den Formeln 2.2.1 und 2.2.2 liegt in dem Beginn der gewichteten Summe beim Index $i = 0$, um den Bias in der Form $w_0 x_0$ mit $x_0 = 1$ zu berechnen.

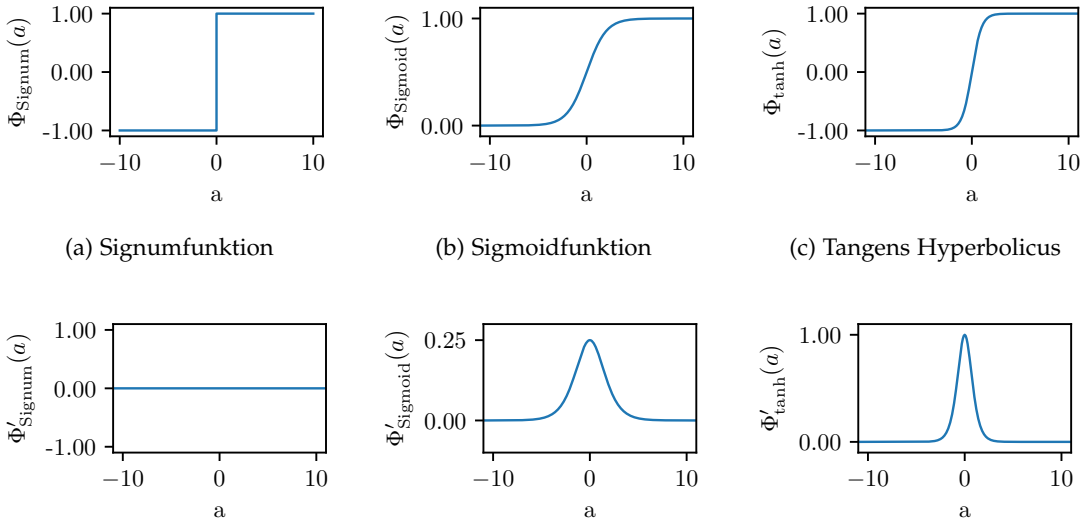
Als Aktivierungsfunktion wurden ehemals häufig die *Signumfunktion*, die *Sigmoidfunktion* und der *Tangens Hyperbolicus* verwendet:

$$\Phi_{\text{Signum}}(a) = \text{sign}(a) \tag{2.2.4}$$

$$\Phi_{\text{Sigmoid}}(a) = \frac{1}{1 + e^{-a}} \tag{2.2.5}$$

$$\Phi_{\tanh}(a) = \frac{e^{2a} - 1}{e^{2a} + 1} \tag{2.2.6}$$

Die Signumfunktion (vgl. Formel 2.2.4 und Abbildung 2.2.2a) kann verwendet werden, um binäre Ausgaben zu erzeugen. Da die Funktion an $a = 0$ nicht differenzierbar ist und der Gradient auf dem Definitionsbereich konstant Null ist (vgl. Abbildung 2.2.2d), ist ein Training durch den Gradientenabstieg, wie in Abschnitt 2.3 beschrieben, nicht möglich. Die Sigmoidfunktion (vgl. Formel 2.2.5 und Abbildung 2.2.2b) ist differenzierbar und bildet kontinuierlich auf $(0, 1) \subset \mathbb{R}$ ab. Die Ausgabe kann als Wahrscheinlichkeit interpretiert werden. Eine Funktion mit ähnlichem Verlauf ist der Tangens



(a) Signumfunktion (b) Sigmoidfunktion (c) Tangens Hyperbolicus

(d) Gradient der Signumfunktion (e) Gradient der Sigmoidfunktion (f) Gradient des Tangens Hyperbolicus

Abbildung 2.2.2: Verlauf der Aktivierungsfunktionen (a)–(c) und deren Gradienten (d)–(f) auf dem Intervall $[-10, 10]$ [Agg18].

Hyperbolicus (vgl. Formel 2.2.6), dessen Wertebereich auf $(-1, 1) \subset \mathbb{R}$ skaliert ist. Die Sigmoidfunktion und der Tangens Hyperbolicus sind differenzierbare Funktionen, bei denen es jedoch bei großen positiven und negativen Werten zu einer *Sättigung der Funktionswerte* und somit zu einem verschwindenden Gradienten kommt. Der Verlauf der Gradienten ist für die Sigmoidfunktion in Abbildung 2.2.2e und für den Tangens Hyperbolicus in Abbildung 2.2.2f dargestellt. Die Abbildungen zeigen, dass der Gradient der Sigmoidfunktion einen Wert von $\Phi'_{\text{Sigmoid}}(\cdot) \leqslant 0{,}25$ und der des Tangens Hyperbolicus einen Wert von $\Phi'_{\tanh}(\cdot) \leqslant 1$ annimmt. Für die Anpassung der Gewichte innerhalb des Netzes werden die Gradienten entlang der Pfade bis zur Eingabe multipliziert. Bei Aktivierungsfunktionen mit einer Ableitung größer als 1, führt dies zu einem exponentiellen Anstieg des Gradienten mit zunehmender Tiefe des Netzes und einer entsprechend großen Änderung der Gewichte in den betroffenen Schichten. Dieses Problem wird als *exploding gradient problem* bezeichnet. Der entgegengesetzte Effekt tritt bei der Verwendung von Aktivierungsfunktionen mit einer Ableitung kleiner als 1 ein. Bei der Multiplikation entlang der Pfade wird der zurück propagierte Gradient jede Schicht geringer. Dieses Problem wird als das

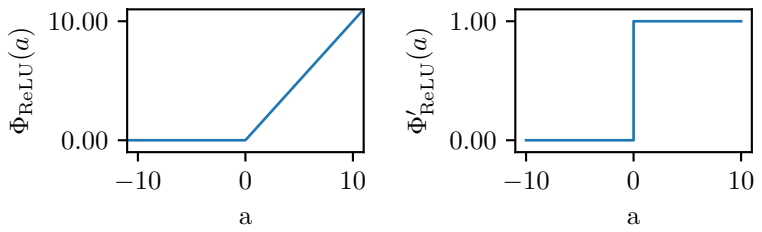
vanishing gradient problem bezeichnet. Beide Probleme können durch Verwendung von Aktivierungsfunktionen wie z. B. der *Rectified Linear Unit* (ReLU, vgl. Abbildung 2.2.3a) vermieden werden

$$\Phi_{\text{ReLU}}(a) = \max\{a, 0\} \tag{2.2.7}$$

Der Gradient der ReLU-Aktivierungsfunktion ist definiert als:

$$\Phi'_{\text{ReLU}}(a) = \begin{cases} 1, \text{ wenn } x > 0 \\ 0, \text{ sonst.} \end{cases} \tag{2.2.8}$$

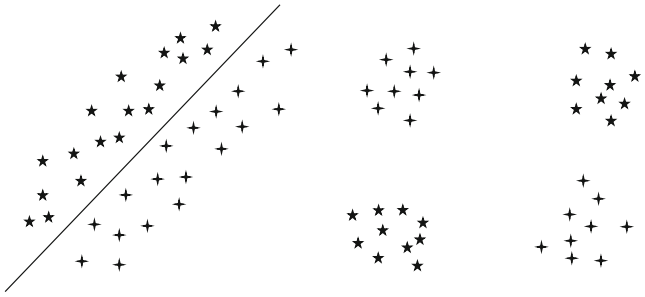
Für negative Funktionsargumente ist der Gradient gleich Null. Da die Ableitung für positive Argumente gleich Eins ist, bleiben die Gradienten bei mehrfacher Multiplikation mit dem Gradienten der ReLU erhalten.



(a) ReLU (b) Gradient der ReLU

Abbildung 2.2.3: Verlauf der ReLU (a) und der Gradienten der ReLU (b) auf dem Intervall $[-10, 10]$ [Agg18].

Die gewichtete Summe des Perzeptrons berechnet eine *lineare Trennfunktion*, indem es eine Hyperebene $\sum_{i=0}^{d} w_i x_i = 0$ definiert. In Folge dessen ist es zur Trennung *linear separierbarer* Daten geeignet, wie in Abbildung 2.2.4a dargestellt ist. Zur Trennung nicht linear separierbarer Daten (z. B. Abbildung 2.2.4b) ist eine komplexere Architektur notwendig. In komplexeren neuronalen Netzen sind mehrere Neuronen hintereinander in Schichten angeordnet. Diese mehrschichtigen Modelle werden als *Feedforward Neural Networks* oder *Multilayer Perzeptrons* (MLPs) bezeichnet (s. z. B. [Agg18, GBC16]). Bei dem Perzeptron folgt auf die Eingabeschicht eine einzige Schicht, welche Berechnungen durchführt und deren Ausgabe beobachtbar ist. Bei einem MLP folgen auf die Eingabeschicht mehrere Schichten, die Berechnungen durchführen, wobei nur die Berechnungen der letzten Schicht (Ausgabeschicht) beobachtbar sind. Aus



(a) (b)

Abbildung 2.2.4: Beispiel für linear separierbare (a) und nicht linear separierbare (b) Daten. Abbildungen entnommen aus [Agg18].

diesem Grund werden die Schichten zwischen Ein- und Ausgabeschicht als *versteckte Schichten* bezeichnet. Abbildung 2.2.5 zeigt ein MLP mit fünf Eingabeneuronen, gefolgt von zwei versteckten Schichten mit jeweils drei Neuronen und einem Neuron in der Ausgabeschicht. Da, wie bei dem Perzeptron, die Eingabeschicht nicht in der Anzahl der Schichten berücksichtigt wird, hat das abgebildete MLP insgesamt $k = 2$ versteckte Schichten. Die Anzahl der Schichten wird auch als *Tiefe* des neuronalen Netzes bezeichnet. Jede Schicht hat eine beliebige, feste Anzahl $M^{(l)}$ von Neuronen, welche nur mit Neuronen in der vorherigen und nachfolgenden Schicht verbunden sind. Bei einem MLP erfolgt die Berechnung schichtweise von der Eingabeschicht zur Ausgabeschicht (*feedforward*), wobei die $M^{(l)}$ Ausgaben der Schicht l, Eingaben für alle $M^{(l+1)}$ Neuronen der folgenden Schicht $l+1$ darstellen. Jedes Neuron i in Schicht l ist folglich mit jedem Neuron j in Schicht $l+1$ über ein Gewicht $w_{ij}^{(l+1)}$ verbunden. Schichten mit dieser Form der Verbindung werden als *vollständig verbundene Schichten* bezeichnet. Die Gewichtsvektoren der $M^{(l)}$ Neuronen einer Schicht werden zu einer $M^{(l-1)} \times M^{(l)}$ Gewichtsmatrix $\mathbf{W}^{(l)}$ zusammengefasst. Die Ausgaben der einzelnen Schichten in Abbildung 2.2.5 werden bei Eingabe $\mathbf{x} = [1, x_1, \ldots, x_d]^T$ berechnet durch

$$\mathbf{h}^{(1)} = \Phi\left(\mathbf{W}^{(1)T}\mathbf{x}\right) \tag{2.2.9}$$

$$\mathbf{h}^{(l+1)} = \Phi\left(\mathbf{W}^{(l+1)T}\mathbf{h}^{(l)}\right) \qquad \forall l \in 1 \ldots k-1 \tag{2.2.10}$$

$$\mathbf{y} = \Phi\left(\mathbf{W}^{(k+1)T}\mathbf{h}^{(k)}\right). \tag{2.2.11}$$

Die Ausgaben der Schichten (inkl. der Ausgabeschicht) sind Vektoren, deren Dimensionalität der Anzahl $M^{(l)}$ der Neuronen in Schicht l entspricht. Aktivierungsfunktionen wie die Sigmoidfunktion oder die ReLU-Funktion (vgl. Formeln 2.2.5

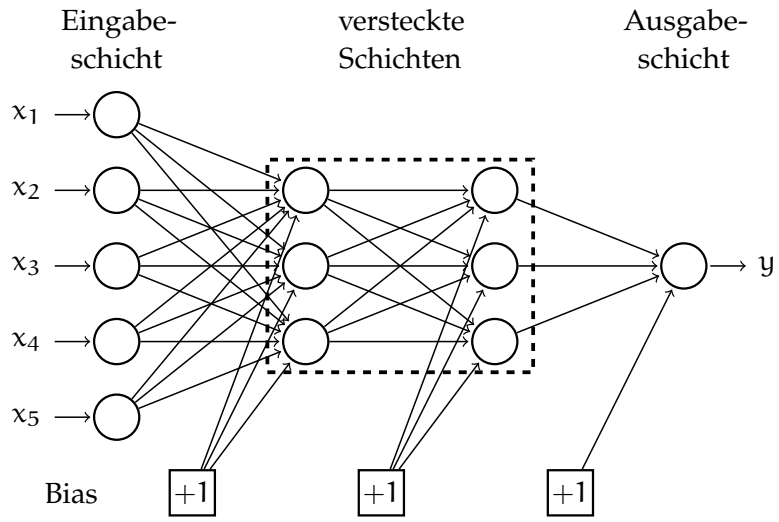


Abbildung 2.2.5: Beispiel für ein Multilayer Perzeptron mit zwei versteckten Schichten [Agg18]. Die Neuronen des Netzes sind als Kreise dargestellt.

und 2.2.7) werden elementweise auf die berechneten Vektoren angewendet. Die Auswahl der Aktivierungsfunktion der Ausgabeschicht und die Anzahl der Neuronen in der Ausgabeschicht sind problemabhängig. Bei der Anwendung eines neuronalen Netzes zur Klassifikation wird häufig ein Ausgabeneuron pro Klasse verwendet. Durch Anwendung einer Softmax-Aktivierungsfunktion (vgl. Formel 2.2.12), welche im Gegensatz zu den in Formeln 2.2.4–2.2.7 definierten Aktivierungsfunktionen eine vektorielle Eingabe verwendet, werden Wahrscheinlichkeiten für die einzelnen Klassen berechnet. Die $M^{(k)}$ Ausgaben der letzten versteckten Schicht k werden als Eingaben $\mathbf{a} = [a_1, \ldots, a_{M^{(k)}}]^T$ verwendet um $M^{(k)} = M^{(k+1)}$ Wahrscheinlichkeiten für die Klassen zu erhalten. Die Wahrscheinlichkeit der Klasse i wird berechnet durch:

$$\Phi(\mathbf{a})_i = \frac{\exp(a_i)}{\sum_{j=1}^{M^{(k)}} \exp(a_j)} \qquad \forall i \in 1 \ldots M^{(k)}. \tag{2.2.12}$$

2.3 TRAINING NEURONALER NETZE

Um eine problemspezifische Funktion (z. B. Trennfunktion zur Klassifikation) zu erlernen, werden in einem Trainingsprozess, welcher in diesem Abschnitt beschrieben wird, die Gewichte des Netzes angepasst. Die Datengrundlage für das Training bildet eine Stichprobe (Trainingsdaten) [Nie83]. Die Trainingsdaten sind als *Trainingspaa-*

re/Trainingsbeispiele $(\mathbf{x}, \mathbf{y})$ gegeben, bestehend aus der Eingabe $\mathbf{x}$ und der erwarteten Ausgabe $\mathbf{y}$ der zu approximierenden Funktion [Agg18]. Die erwartete Ausgabe wird auch als *Label* oder *Annotation* bezeichnet. Aus den gegebenen Eingabedaten berechnet das neuronale Netz eine Vorhersage $\hat{\mathbf{y}}$, deren Vergleich mit der gegebenen Annotation zu einem *Fehler* ϵ führt. Diese Form des Trainings wird *überwachtes Lernen* genannt [Nie83].

In Abschnitt 2.3.1 werden mögliche Fehlerfunktionen, im Folgenden auch *Loss-Funktionen* genannt, beschrieben. Anschließend wird in Abschnitt 2.3.2 erläutert, wie der berechnete Fehler zur Anpassung der Gewichte verwendet wird. Mögliche Schwierigkeiten, die beim Training neuronaler Netze auftreten können, werden abschließend in Abschnitt 2.3.3 beschrieben.

2.3.1 *Loss-Funktion*

Nach Berechnung einer Vorhersage $\hat{\mathbf{y}}$ für eine Eingabe $\mathbf{x}$ wird zum Training des neuronalen Netzes der Fehler, im Folgenden auch als *Loss* bezeichnet, zwischen der Vorhersage und dem tatsächlichen Label $\mathbf{y}$ mittels einer Loss-Funktion mit $\epsilon = L(\mathbf{y}, \hat{\mathbf{y}})$ ermittelt. Dieser wird zur Berechnung angepasster Gewichte im neuronalen Netz verwendet [Agg18]. Eine mögliche Loss-Funktion ist der quadratische Fehler zwischen Label und Vorhersage:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = (\mathbf{y} - \hat{\mathbf{y}})^2 \tag{2.3.1}$$

Zur Klassifikation wird als Aktivierungsfunktion häufig eine Softmax-Funktion (vgl. Formel 2.2.12) auf die Ausgabe angewendet, um Wahrscheinlichkeiten für die einzelnen Klassen zu erhalten. Zum Training neuronaler Netze mit probabilistischen Ausgaben kann als Loss-Funktion die Kreuzentropie verwendet werden. Bei den vorhergesagten Wahrscheinlichkeiten $\hat{\mathbf{y}} = [\hat{y}_1, \ldots, \hat{y}_n]^T$ und einer gegebenen Annotation $\mathbf{y} = [y_1, \ldots, y_n]^T$ mit $y_i = 1$ für die korrekte Klasse (sonst $y_i = 0$), wobei n der Anzahl der Klassen entspricht, wird der Fehler wie folgt berechnet [Nie83]:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{n} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \tag{2.3.2}$$

Häufig wird jedoch nur die ausgegebene Wahrscheinlichkeit $\hat{y}_r$ für die korrekte Klasse r in der Berechnung der Fehler berücksichtigt (z. B. in der Bibliothek PyTorch[1]) [Agg18]:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\log(\hat{y}_r). \tag{2.3.3}$$

1 https://pytorch.org/docs/0.4.0/nn.html#conv2d

Die Wahl der Aktivierungsfunktion der Ausgabeschicht und der Loss-Funktion sind problemabhängig. Für diskrete Ausgaben, wie z. B. zur Klassifikation, wird in der Regel eine Softmax-Aktivierungsfunktion und ein Kreuzentropie-Loss verwendet, während für kontinuierliche, reelle Ausgaben häufig eine lineare Aktivierungsfunktion (z. B. Identität) mit dem quadratischen Fehler zum Training verwendet wird.

2.3.2 *Backpropagation*

Nach Berechnung des Fehlers erfolgt eine Anpassung der Gewichte in Richtung des negativen Gradienten der Loss-Funktion, um den Fehler zu minimieren [Agg18, Nie83]. Dieser Prozess wird als *Gradientenabstieg* bezeichnet. Die Gewichte werden dabei wie folgt korrigiert:

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \alpha \frac{\partial \epsilon}{\partial w_{ij}^{(l)}}. \tag{2.3.4}$$

Die Schrittweite der Korrektur wird durch die Lernrate α festgelegt. Im *Vorwärtsdurchlauf* wird zunächst eine Vorhersage $\hat{\mathbf{y}}$ aus einem Trainingsbeispiel $\mathbf{x} = [1, x_1, \ldots, x_d]^T$ wie in Abschnitt 2.2 beschrieben berechnet und der Fehler $\epsilon = L(\mathbf{y}, \hat{\mathbf{y}})$ bestimmt. Auf den Vorwärtsdurchlauf folgt der *Rückwärtsdurchlauf*, in dem der Gradient der Loss-Funktion bezüglich der einzelnen Gewichte mittels der *Kettenregel* berechnet und zur Korrektur dieser Gewichte verwendet wird. Das neuronale Netz wird zu diesem Zweck als *Computational Graph* dargestellt. Ein Gewicht hat im Vorwärtsdurchlauf Einfluss auf die Ergebnisse entlang verschiedener Pfade durch die folgenden Schichten. Analog dazu ergibt sich der Gradient der Loss-Funktion bezüglich eines bestimmten Gewichtes durch Aggregation der Gradienten entlang der Pfade vom Gewicht zur Ausgabe.

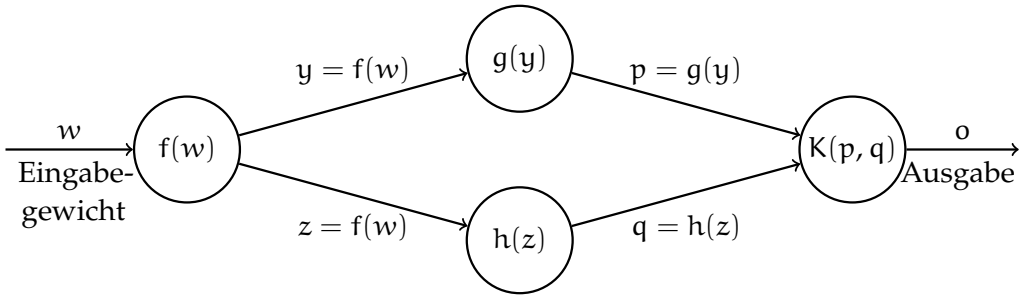


Abbildung 2.3.1: Veranschaulichung der Kettenregel zur Berechnung des Gradienten bezüglich eines Gewichtes am Beispiel eines einfachen Computational Graphs [Agg18].

Abbildung 2.3.1 zeigt ein Beispiel eines Computational Graphs. Das Gewicht w hat Einfluss auf die Zwischenergebnisse p und q, welche in einem MLP den Ausgaben zweier Neuronen in der folgenden Schicht entsprechen. Die Ausgabe des gesamten Graphen $o = K(p, q) = K(g(f(w)), h(f(w)))$ ist von den Zwischenergebnissen p und q abhängig. Bei Anwendung der Kettenregel ergibt sich der Gradient $\frac{\partial o}{\partial w}$ durch Addition der Gradienten entlang der Pfade über $g(\cdot)$ und $h(\cdot)$ vom Gewicht w zur Ausgabe o:

$$\begin{aligned}\frac{\partial o}{\partial w} &= \frac{\partial o}{\partial p}\frac{\partial p}{\partial w} + \frac{\partial o}{\partial q}\frac{\partial q}{\partial w} \\ &= \frac{\partial o}{\partial p}\frac{\partial p}{\partial y}\frac{\partial y}{\partial w} + \frac{\partial o}{\partial q}\frac{\partial q}{\partial z}\frac{\partial z}{\partial w}\end{aligned} \tag{2.3.5}$$

Analog zum Beispiel wird die Kettenregel zur Berechnung der Gradienten in einem neuronalen Netz angewendet. Entlang eines einzelnen Pfades $[h^{(1)}, h^{(2)}, \ldots, h^{(k)}]$ beliebiger Länge wird der Gradient berechnet durch

$$\frac{\partial \epsilon}{\partial w^{(l)}} = \frac{\partial \epsilon}{\partial y} \cdot \left[\frac{\partial y}{\partial h^{(k)}} \prod_{i=l}^{k-1} \frac{\partial h^{(i+1)}}{\partial h^{(i)}}\right] \frac{\partial h^{(l)}}{\partial w^{(l)}} \qquad \forall l \in 1 \ldots k. \tag{2.3.6}$$

In Formel 2.3.6 ist $h^{(l)}$ die skalare Ausgabe des Neurons in Schicht l, welches auf dem betrachteten Pfad liegt. Zur besseren Lesbarkeit wurde auf doppelte Indizierung des Gewichts w zur Angabe der Verbindung von Neuron $h^{(l)}$ zu Neuron $h^{(l+1)}$ verzichtet. In der Regel führt mehr als ein Pfad von Neuron $h^{(l)}$ zur Ausgabe. Für eine Menge $\mathcal{P}$ von Pfaden von $h^{(l)}$ zur Ausgabe y wird der Gradient allgemein berechnet durch

$$\begin{aligned}\frac{\partial \epsilon}{\partial w^{(l)}} &= \frac{\partial \epsilon}{\partial y}\left[\sum_{[h^{(l)}, h^{(l+1)}, \ldots, h^{(k)}, y] \in \mathcal{P}} \frac{\partial y}{\partial h^{(k)}} \prod_{i=l}^{k-1} \frac{\partial h^{(i+1)}}{\partial h^{(i)}}\right] \frac{\partial h^{(l)}}{\partial w^{(l)}} \\ &= \frac{\partial \epsilon}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial w^{(l)}}.\end{aligned} \tag{2.3.7}$$

Formel 2.3.7 zeigt, dass die Berechnung des Gradienten dem Produkt der Terme $\frac{\partial \epsilon}{\partial h^{(l)}}$ und $\frac{\partial h^{(l)}}{\partial w^{(l)}}$ entspricht. Der Term $\frac{\partial \epsilon}{\partial h^{(l)}}$ aggregiert die Gradienten entlang der Pfade von $h^{(l)}$ zum Fehler der Ausgabe y. Der Term $\frac{\partial h^{(l)}}{\partial w^{(l)}}$ berechnet den Gradienten des Gewichts $w^{(l)}$ bezüglich der Ausgabe des Neurons $h^{(l)}$ und wird berechnet durch

$$\frac{\partial h^{(l)}}{\partial w^{(l)}} = h^{(l-1)} \cdot \Phi'(a^{(l)}), \tag{2.3.8}$$

wobei $\Phi'(\cdot)$ die Ableitung der Aktivierungsfunktion ist. Der erste Term $\frac{\partial \epsilon}{\partial h^{(l)}}$ von Formel 2.3.7 lässt sich rekursiv für frühere Schichten aus den Gradienten späterer Schichten mittels Kettenregel berechnen:

$$\frac{\partial \epsilon}{\partial h^{(l)}} = \sum_{h : h^{(l)} \Rightarrow h} \frac{\partial \epsilon}{\partial h} \frac{\partial h}{\partial h^{(l)}}. \tag{2.3.9}$$

Ist ein Neuron $h^{(l)}$ in der Schicht vor Neuron h und die Verbindung der Neuronen gewichtet mit $w_{(h^{(l)},h)}$, wird der Gradient an Neuron $h^{(l)}$ berechnet durch

$$\frac{\partial h}{\partial h^{(l)}} = \frac{\partial h}{\partial a^{(h)}} \frac{\partial a^{(h)}}{\partial h^{(l)}} = \Phi'(a^{(h)}) \cdot w_{(h^{(l)},h)}. \tag{2.3.10}$$

Die Ausgabe des Neurons h ergibt sich dabei aus der Anwendung der Aktivierungsfunktion $\Phi(\cdot)$ auf die Linearkombination $a^{(h)}$ der Eingaben des Neurons h. Formel 2.3.9 zeigt, dass zur Berechnung des Gradienten eines Neurons in Schicht l die Gradienten aller Neuronen h in späteren Schichten $l+1, \ldots, k$ auf den Pfaden zur Ausgabe zuvor berechnet werden müssen. Folglich wird der Gradient beginnend bei der Ableitung des Fehlers nach der Ausgabe $\frac{\partial \epsilon}{\partial y}$ schichtweise rückwärts durch das neuronale Netz propagiert. Der Algorithmus dieser rekursiven Berechnung wird daher als *Backpropagation-Algorithmus* bezeichnet.

Der Fehler ϵ kann über unterschiedlich viele Trainingsbeispiele berechnet werden. Für den Gradientenabstieg kann der Fehler über alle n Paare des Trainingsdatensatzes berechnet werden, sodass

$$\epsilon = \sum_{i=1}^{n} \epsilon_i \tag{2.3.11}$$

gilt. Die Aktualisierung der Gewichte erfolgt mit dem über den Datensatz akkumulierten Fehler:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \alpha \frac{\partial \epsilon}{\partial \mathbf{W}^{(l)}} \tag{2.3.12}$$

Dieser Ansatz ist jedoch für große Datensätze oder große Netzarchitekturen aufgrund eines zu hohen Speicheraufwands in der Praxis nicht realisierbar. Eine Alternative stellt der *stochastische Gradientenabstieg* (engl. *stochastic gradient descent*, SGD) dar. Dieser Ansatz wird als stochastisch bezeichnet, da der Gradientenabstieg anhand einzelner Trainingsbeispiele in zufälliger Reihenfolge durchgeführt wird. Die Aktualisierung der Gewichte wird für den Fehler eines einzelnen Datenpaares berechnet:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \alpha \frac{\partial \epsilon_i}{\partial \mathbf{W}^{(l)}}. \tag{2.3.13}$$

Die Berechnung der Gradienten des Fehlers eines einzelnen Punktes stellt eine Approximation des Gradienten des Fehlers aller Beispiele des Trainingsdatensatzes dar und kann schnell berechnet werden. Die Sortierung der Trainingspaare kann einen starken Einfluss auf das Ergebnis der Optimierung durch den Gradientenabstieg nehmen. Folglich kann eine ungeeignete Sortierung der Trainingsdaten dazu führen, dass durch das Training keine Verbesserung oder eine Verschlechterung der Ergebnisse erreicht wird. Der *stochastische Mini-Batch Gradientenabstieg* stellt einen Kompromiss bezüglich Aufwand und Stabilität der Berechnung des Gradientenabstiegs dar. Bei diesem Ansatz wird der Fehler für eine kleine Menge $\mathcal{B}$, welche im Folgenden als *Batch* oder *Mini-Batch* bezeichnet wird, akkumuliert:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \alpha \sum_{i \in \mathcal{B}} \frac{\partial \epsilon_i}{\partial \mathbf{W}^{(l)}}. \tag{2.3.14}$$

Vorwärtsdurchlauf, Rückwärtsdurchlauf und Korrektur der Gewichte mittels aller Trainingsbeispiele in einer Mini-Batch werden zusammen als eine *Iteration* bezeichnet. Wurden alle Beispiele im gesamten Trainingsdatensatz exakt einmal zur Aktualisierung der Gewichte verwendet, wird dies als *Epoche* bezeichnet. Das Training eines neuronalen Netzes kann mehrere tausend Epochen dauern [Agg18]. Die Anzahl der Iterationen pro Epoche ergibt sich folglich durch $\#\text{Iterationen} = n/|\mathcal{B}|$, wobei n der Anzahl der Beispiele im Trainingsdatensatz und $|\mathcal{B}|$ der Größe der Mini-Batches entspricht.

Bei aufeinanderfolgenden Iterationen kann es dazu kommen, dass die berechneten Gradienten in unterschiedliche Richtungen zeigen. Dies verlangsamt die Minimierung des Fehlers durch den Gradientenabstieg. Die Anpassung der Gewichte (vgl. Formel 2.3.14) erfolgt durch Addition eines Korrekturterms und lässt sich zur Verdeutlichung der additiven Anpassung wie folgt darstellen:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} + \mathbf{V}^{(l)} \qquad \text{, mit } \mathbf{V}^{(l)} = -\alpha \sum_{i \in \mathcal{B}} \frac{\partial \epsilon_i}{\partial \mathbf{W}^{(l)}}. \tag{2.3.15}$$

Die Gradienten aufeinanderfolgender Iterationen können durch die Erweiterung um ein *Momentum* $\beta \in (0, 1)$ geglättet werden:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} + \mathbf{V}^{(l)} \qquad \text{, mit } \mathbf{V}^{(l)} = \beta \mathbf{V}^{(l)} - \alpha \sum_{i \in \mathcal{B}} \frac{\partial \epsilon_i}{\partial \mathbf{W}^{(l)}}. \tag{2.3.16}$$

Für ein Momentum von $\beta = 0$ entspricht die Berechnung in Formel 2.3.16 dem stochastischen (Mini-Batch) Gradientenabstieg (vgl. Formeln 2.3.14 und 2.3.15). Für ein Momentum $\beta > 0$ werden Richtung und Betrag des Gradienten der vorherigen Iteration zusammen mit dem Gradienten der aktuellen Iteration gewichtet zur Korrektur

der Gewichte verwendet. In Folge dessen bleibt die Richtung des Gradienten und somit die Richtung der Korrektur der Gewichte über mehrere aufeinanderfolgende Iterationen ähnlich.

2.3.3 *Probleme beim Training neuronaler Netze*

Ein Problem, welches beim Training neuronaler Netze auftreten kann, ist das *Overfitting*. Beim Training wird der Fehler auf dem Trainingsdatensatz minimiert, was jedoch nicht garantiert, dass der Fehler auf unbekannten Daten ebenfalls gering ist [Agg18]. Eine gute Vorhersage auf ungesehenen Daten, nach dem Training auf einem Datensatz mit einer festen Anzahl von Trainingsbeispielen, wird als *Generalisierungsleistung* eines Netzes bezeichnet. Einen großen Einfluss auf die Generalisierungsleistung haben die Anzahl der trainierbaren Parameter des Modells (Gewichte des Netzes) und die Anzahl der Beispiele im Trainingsdatensatz. Durch eine Vergrößerung des Trainingsdatensatzes bei gleich bleibender Anzahl von Gewichten im Netz kann die Generalisierungsleistung erhöht werden. Zu viele Parameter bei zu wenigen Trainingsbeispielen können zum Overfitting und somit einer schlechten Generalisierungsleistung führen. Eine zu geringe Anzahl trainierbarer Parameter kann jedoch dazu führen, dass komplexe Zusammenhänge in den Trainingsdaten nicht gelernt werden können.

Eine Möglichkeit, dem Overfitting entgegen zu wirken, ist die *Regularisierung*, welche z. B. von Aggarwal [Agg18] beschrieben wird. Ziel der Regularisierung ist es, betraglich kleinere und weniger Parameter ungleich Null im Training zu erlernen. Zu diesem Zweck wird der Fehler um einen, vom Betrag der Gewichte abhängigen, Term $\lambda \cdot ||\mathbf{W}||^p$ erweitert:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \alpha \left(\sum_{i \in \mathcal{B}} \frac{\partial \epsilon_i}{\partial \mathbf{W}^{(l)}} + \frac{\partial (\lambda \cdot ||\mathbf{W}^{(l)}||^p)}{\partial \mathbf{W}^{(l)}} \right). \tag{2.3.17}$$

Der Exponent wird häufig als $p = 2$ gewählt. In diesem Fall wird die Regularisierung als *Tikhonov-* oder *L2-Regularisierung* bezeichnet. Der *Weight-Decay* λ gewichtet den Einfluss der Regularisierung auf den Gradienten und somit den Einfluss des Betrags der Gewichte auf deren Anpassung durch den Gradientenabstieg.

Eine andere Methode der Regularisierung ist das Verwenden von *Dropout* [Agg18]. Neuronen des Netzes werden mit einer gewählten Wahrscheinlichkeit p_{drop} unabhängig voneinander für die Auswertung eines Trainingsbeispiels oder einer Mini-Batch entfernt. Bei dem Gradientenabstieg werden anschließend nur die Gewichte der nicht entfernten Neuronen aktualisiert. Das zufällige Entfernen von Neuronen entspricht dem Nullsetzen ihrer Ausgaben und wird für jedes Trainingsbeispiel bzw. jede Mini-

Batch wiederholt. Dadurch wird ein Rauschen in den von den Schichten berechneten Merkmalen erzeugt, welches beim Training eine Redundanz zwischen Merkmalen erzwingt und zu einer robusteren Auswertung des Modells führt.

2.4 FALTUNGSNETZE

Faltungsnetze sind eine spezielle Form der neuronalen Netze [GBC16]. Die Eingabe liegt dabei in einer bekannten Gitterstruktur einer bestimmten Dimensionalität vor. Das bekannteste Beispiel einer solchen Eingabe ist ein Bild mit einem festgelegten Pixelgitter (2D) [Agg18]. Wird in mindestens einer Schicht eines neuronalen Netzes anstelle der Matrixmultiplikation mit paarweisen Verbindungen aller Neuronen der vorherigen zur nächsten Schicht (vgl. Abschnitt 2.2) eine Faltungsoperation berechnet, wird das Netz als neuronales Faltungsnetz (engl. convolutional neural network) bezeichnet. Mit zunehmender Tiefe der Faltungsnetze werden diese auch als tiefe neuronale Faltungsnetze (engl. deep convolutional neural networks) bezeichnet. In Abschnitt 2.4.1 wird die Faltung als allgemeine Operation definiert und beschrieben, wie diese in einem neuronalen Netz auf Eingaben, wie z. B. Bilder, angewendet wird. Anschließend wird in Abschnitt 2.4.2 die in Faltungsnetzen häufig verwendete Poolingoperation erläutert. In den Abschnitten 2.4.3 und 2.4.4 werden die Transpose und die Dilated Convolution als weitere mögliche Operationen in einem Faltungsnetz beschrieben.

2.4.1 *Faltungsschicht*

Faltungsschichten berechnen das Skalarprodukt zwischen einer Gitterstruktur von Gewichten (Tensor) und Bereichen der Eingabe mit einer gleichen Gitterstruktur an verschiedenen räumlichen Positionen der Eingabe [Agg18]. Im Vergleich zu vollständig verbundenen Schichten sind die Neuronen der Faltungsschichten nur mit einem Teil der Neuronen der vorherigen Schicht verbunden [GBC16]. Dies führt zu einer Reduktion der Anzahl der Parameter der Schicht und verringertem Speicherverbrauch des Modells. Zudem werden dieselben Gewichte eines Neurons in einer Faltungsschicht nicht nur mit einer Eingabe, sondern mit Eingabewerten an verschiedenen räumlichen Positionen der Eingabe verwendet. Dieses *Parameter Sharing* führt zu einer weiteren Verringerung der Anzahl der Parameter des Modells. Durch das Parameter Sharing wird zudem eine *Äquivarianz* der Berechnungen der Faltungsschicht erreicht. Somit führen Veränderungen der Eingabe zu denselben Veränderungen der Ausgabe. Am Beispiel eines Bildes als zweidimensionale Eingabe bedeutet dies, dass Ergebnisse der

Faltungsoperation bei Verschiebung des Eingabebildes entsprechend in der Ausgabe verschoben werden. Faltungsnetze eignen sich durch diese Eigenschaften besonders gut, um auf strukturierte Eingaben angewendet zu werden, da räumliche Verhältnisse erhalten bleiben.

Jede Faltungsschicht für zweidimensionale Eingaben hat eine dreidimensionale Gitterstruktur mit einer Höhe $H^{(l)}$, einer Breite $B^{(l)}$ und einer Tiefe $D^{(l)}$. Breite und Höhe entsprechen den Dimensionen der Eingabe und unterteilen diese in Pixel [Agg18]. Die Tiefe der Faltungsschicht ermöglicht es, mehrere Eigenschaften pro Pixel zu repräsentieren. Bei einem RBG-Bild als Eingabe werden so Rot-, Grün- und Blauanteile jeweils in einem $B^{(l)} \times H^{(l)} \times 1$ Gitter repräsentiert, welches als *Kanal* bezeichnet wird, was zu einer Tiefe von $D^{(l)} = 3$ führt. Für $l > 1$ werden die Kanäle als *Feature-Maps* oder *Activation-Maps* und die enthaltenen Werte als Merkmale bezeichnet. Die Gewichte der Faltungsschichten sind dreidimensionale Gitter der Größe $F^{(l)} \times F^{(l)} \times D^{(l)}$ und werden als *Filter* oder *Kernel* bezeichnet.

Im Allgemeinen ist die Faltung als Operation auf zwei reellwertigen Funktionen $x(\cdot)$ und $w(\cdot)$ wie folgt definiert [GBC16]:

$$s(t) = (x \star w)(t) = \int x(a)w(t-a)\,da. \tag{2.4.1}$$

Die Eingaben der neuronalen Netze (z. B. Bilder) sind i. A. nicht kontinuierlich, sondern enthalten diskrete Abtastwerte einer kontinuierlichen Funktion. Zudem kann eine mehrdimensionale Eingabe vorliegen. Für eine diskrete, zweidimensionale Eingabe X_{ij} mit einem Kanal und diskretem Kernel K_{ij} ist die Faltung definiert als

$$S(i,j) = (I \star K)_{ij} = \sum_m \sum_n I_{m,n} K_{i-m,j-n} = \sum_m \sum_n I_{i-m,j-n} K_{m,n}. \tag{2.4.2}$$

Formel 2.4.2 zeigt zudem die *Kommutativität der Faltungsoperation*. In vielen Bibliotheken zur Implementierung neuronaler Netze (z. B. PyTorch[2]) wird anstelle der zweidimensionalen Faltung die *Kreuzkorrelation* berechnet:

$$S(i,j) = (I \star K)(i,j) = \sum_m \sum_n I_{i+m,j+n} K_{m,n}. \tag{2.4.3}$$

Im Vergleich zur Faltung wird der Kernel nicht gespiegelt. Da bei Anwendung der Faltung in neuronalen Netzen die Gewichte gelernt werden, wird bei Verwendung der Kreuzkorrelation anstelle der Faltung der Kernel gespiegelt gelernt. Aus diesem Grund wird im Folgenden nicht explizit zwischen Faltung und Kreuzkorrelation unterschieden. Abbildung 2.4.1 zeigt ein Beispiel für die Berechnung der zweidimensionalen

2 https://pytorch.org/docs/0.4.0/nn.html#conv2d

Faltung ohne den Kernel zu spiegeln (Kreuzkorrelation) mit einem 2×2 Kernel und einer 4×3 Pixel großen Eingabe.

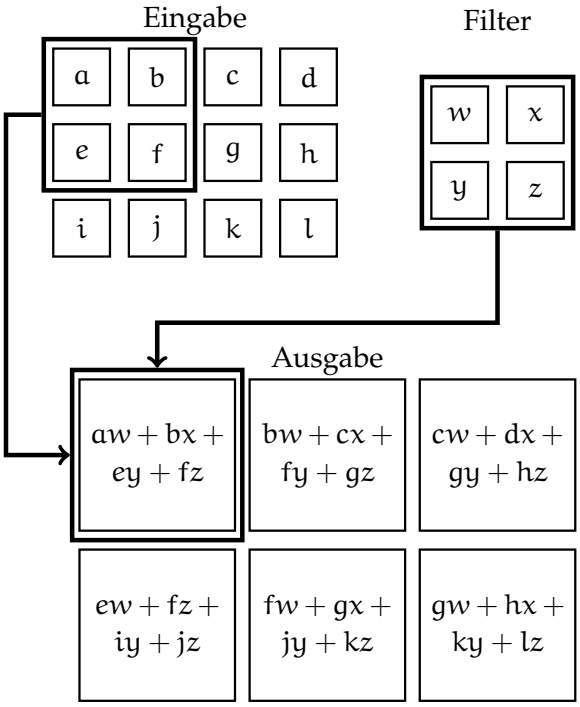


Abbildung 2.4.1: Beispiel zur Visualisierung der Berechnung der zweidimensionalen Faltung ohne Spiegelung des Filters [GBC16]. Die Rahmen und Pfeile markieren den Teil der Eingabe mit dem das Skalarprodukt mit dem Filter zur Berechnung des oberen linken Pixels der Ausgabe verwendet wird.

In dieser Arbeit werden Faltungsnetze auf RGB-Bildern (zweidimensionale Eingaben mit drei Kanälen) angewendet. Die Feature-Maps der Faltungsschichten sind daher dreidimensional mit einer Größe von $B^{(l)} \times H^{(l)} \times D^{(l)}$ und werden als Tensor $\mathbf{H}^{(l)} = \left[h_{ijk}^{(l)}\right]$ repräsentiert. Die Gewichte des Filters p in Schicht l sind ebenfalls als dreidimensionaler Tensor $\boldsymbol{W}^{(p,l)} = \left[w_{ijk}^{(p,l)}\right]$ der Größe $F^{(l)} \times F^{(l)} \times D^{(l)}$ gegeben.

Die Faltungsoperation von Schicht l zu Schicht $l+1$ wird in diesem Fall wie folgt berechnet [Agg18]:

$$h_{ijp}^{(l+1)} = \sum_{r=1}^{F^{(l)}} \sum_{s=1}^{F^{(l)}} \sum_{k=1}^{D^{(l)}} w_{rsk}^{(p,l)} h_{i+r-1,j+s-1,k}^{(l)} \quad \begin{array}{l} \forall i \in \{1,\ldots,B^{(l)} - F^{(l)} + 1\} \\ \forall j \in \{1,\ldots,H^{(l)} - F^{(l)} + 1\} \\ \forall p \in \{1,\ldots,D^{(l+1)}\}. \end{array} \tag{2.4.4}$$

Die Indizes (i,j) geben in Formel 2.4.4 die Position in der Höhe und der Breite der Feature-Map an, der Index p den Filter (Tiefe). Wie bei der Berechnung der Neuronen in vollständig verbundenen Schichten, können auch die Neuronen der Faltungsschichten um die Addition eines Bias erweitert werden. Jeder Filter p in jeder Faltungsschicht l hat einen eigenen Bias-Parameter $b^{(p,l)}$. Im Vergleich zu vollständig verbundenen Schichten werden nicht alle Ausgaben der vorherigen Schicht zur Eingabe jedes Neurons in der darauffolgenden Schicht. Ein Teil der Merkmale in Schicht l hat Einfluss auf die Berechnung eines Merkmals als Ausgabe eines Neurons in Schicht $l+1$. Folglich hat mit zunehmender Anzahl von aufeinanderfolgenden Faltungschichten ein größerer Bereich des Eingabebildes Einfluss auf ein Merkmal. Dieser Bereich wird als *rezeptives Feld* bezeichnet. Beispielsweise haben Neuronen bei Verwendung eines 3×3 Filters in der ersten versteckten Schicht ein rezeptives Feld von 3×3 Pixeln des Eingabebildes. Dieses Feld wird durch eine darauffolgende Faltungsschicht mit einem 3×3 Filter auf 5×5 Pixel vergrößert.

Die Größe der Ausgabe einer Faltungsschicht hängt von Breite und Höhe der Eingabe und Größe und Anzahl der Filter, nicht aber von der Tiefe der Eingabe ab. Das Skalarprodukt von Filter und Pixeln der Eingabe kann an $B^{(l)} - F^{(l)} + 1$ Positionen entlang der Breite und $H^{(l)} - F^{(l)} + 1$ Positionen entlang der Höhe berechnet werden, woraus die Breite und Höhe der Schicht $l+1$ resultieren. Die Tiefe der folgenden Schicht $l+1$ entspricht der Anzahl der verwendeten Filter. Die Verringerung der Größe der Feature-Maps ist häufig nicht erwünscht. *Padding* wirkt der Verkleinerung entgegen, indem die Eingabe der Schicht vor der Faltung um einen Rand von $(F^{(l)} - 1)/2$ Pixel vergrößert wird. Beim *Zero-Padding* wird der Wert der Pixel bzw. Merkmale unabhängig von den gegebenen Werten auf Null gesetzt, um bei der Faltung das Ergebnis der Skalarprodukte nicht zu verändern. Eine andere Form des Paddings ist das *Reflection-Padding*, wie es z. B. von Yu et al. [YK15] verwendet wird. Beim Reflection-Padding wird der Rand anstelle von Nullen mit den Pixelwerten des, an dessen Rand reflektierten, Bildes aufgefüllt.

Die Faltungsoperation in Formel 2.4.4 berechnet das Skalarprodukt zwischen einem Filter und einem Eingabegitter derselben Größe an jeder möglichen Position. Die

Anzahl der Positionen kann verringert werden, indem die Faltung anstelle der Positionen $1, 2, 3, \ldots$ (in Höhe und Breite) nur an den Positionen $1, S^{(l)} + 1, 2S^{(l)} + 1, \ldots$ berechnet wird. Der Faktor $S^{(l)}$ wird in diesem Zusammenhang als *Stride* bezeichnet. Formel 2.4.4 beschreibt eine Faltungsoperation mit Stride $S^{(l)} = 1$. Durch die Verwendung eines größeren Strides wird die Größe der Ausgabe auf $(B^{(l)} - F^{(l)})/S^{(l)} + 1$ bzw. $(H^{(l)} - F^{(l)})/S^{(l)} + 1$ verringert. Gleichzeitig führt die Verwendung eines größeren Strides zu einer schnelleren Vergrößerung des rezeptiven Feldes und ermöglicht somit die Berechnung komplexerer Merkmale über einem größeren Bereich des Eingabebildes. Eine Alternative Operation zur Verkleinerung der Feature-Maps bei Vergrößerung des rezeptiven Feldes ist das im Folgenden beschriebene *Pooling*.

2.4.2 *Pooling*

Eine weitere Operation, welche häufig in tiefen neuronalen Faltungsnetzen verwendet wird, ist das *Pooling*. Wie auch die Faltungsoperation, arbeitet das Pooling auf einem kleinen Bereich des Eingabegitters der Größe $P^{(l)} \times P^{(l)}$ [Agg18]. Die am häufigsten verwendete Variante ist das *Max-Pooling*, bei dem das Maximum aus dem betrachteten Gitterbereich zurückgegeben wird. Eine seltener verwendete Form ist das *Average-Pooling*, bei dem der Mittelwert der Merkmale im Gitterbereich berechnet wird. Im Gegensatz zur Faltung wird das Pooling auf jeder Feature-Map unabhängig von den anderen Feature-Maps durchgeführt. In Folge dessen bleibt die Tiefe der Schicht durch Verwendung von Pooling unverändert. Die Ausgabe des Poolings mit einem Stride $S^{(l)}$ und einem Pooling-Bereich der Größe $P^{(l)} \times P^{(l)}$ hat eine Größe von $(H^{(l)} - P^{(l)})/S^{(l)} + 1 \times (B^{(l)} - P^{(l)})/S^{(l)} + 1 \times D^{(l)}$. Das Pooling wird häufig

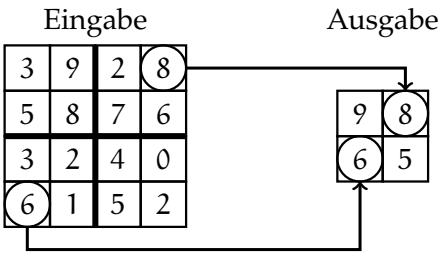


Abbildung 2.4.2: Beispiel zur Visualisierung des Max-Poolings auf einer 4×4 Eingabe mit einem 2×2 Pooling-Bereich und einem Stride von $S = 2$. Die dickeren Linien markieren die Bereiche, aus denen jeweils der maximale Wert zurückgegeben wird. In zwei Bereichen wird als Beispiel das Maximum mit einem Kreis markiert. Die Pfeile verdeutlichen die Position in der 2×2 Ausgabe. Jedes Merkmal der Ausgabe hat ein rezeptives Feld der Größe 2×2.

mit einem 2×2 Gitter als Pooling-Bereich und einem Stride $S^{(l)} = 2$ durchgeführt. Abbildung 2.4.2 zeigt ein Beispiel für die Anwendung des Max-Poolings auf eine 4×4 Eingabe. Die Verwendung eines Strides von $S^{(l)} = 2$ führt, wie im Beispiel, zu einer Halbierung der Größe der Feature-Maps. Gleichzeitig führt Pooling zu einer Vergrößerung des rezeptiven Feldes. Ein weiterer Effekt der durch Verwendung von Pooling erreicht werden soll ist, dass leichte Verschiebungen des Eingabebildes zu keiner großen Veränderung der Feature-Maps führen. Diese Eigenschaft wird als *Translationsinvarianz* bezeichnet.

2.4.3 *Transpose Convolution*

Transpose Convolution wird häufig auch *Deconvolution* oder *fractionally strided Convolution* genannt [Agg18]. Die Faltungsoperation kann auch als Matrixmultiplikation dargestellt werden. Zur Vereinfachung wird im Folgenden eine Faltungsoperation mit einem Stride von 1 betrachtet. Bei Faltung einer Eingabe der Größe $H^{(l)} \times B^{(l)} \times 1$ mit einem Filter der Größe $F^{(l)} \times F^{(l)}$ wird eine Ausgabe mit der Größe $(H^{(l)} - F^{(l)} + 1) \times (B^{(l)} - F^{(l)} + 1) \times 1$ berechnet. Die Fläche der Eingabe bzw. die Anzahl ihrer Elemente beträgt $H^{(l)} \times B^{(l)} \times 1 = A^{(l)}$ und die der Ausgabe $(H^{(l)} - F^{(l)} + 1) \times (B^{(l)} - F^{(l)} + 1) \times 1 = A^{(l+1)}$. Indem die Zeilen hintereinander konkateniert werden, können Ein- und Ausgabe als Spaltenvektor $\mathbf{h}^{(l)}$ und $\mathbf{h}^{(l+1)}$ repräsentiert werden. Die Faltungsoperation kann dann als Multiplikation mit einer dünn besetzten $A^{(l)} \times A^{(l+1)}$ Matrix $\mathbf{C}^{(l)}$ dargestellt werden:

$$\mathbf{h}^{(l+1)} = \mathbf{C}^{(l)} \cdot \mathbf{h}^{(l)}. \tag{2.4.5}$$

Jede Reihe der Matrix $\mathbf{C}$ wird zur Berechnung eines Ausgabewertes verwendet und entspricht somit jeweils einer Position des Filters auf der Eingabe. Analog zur Umwandlung der Matrix in einen Spaltenvektor vor der Berechnung, wird der Ausgabevektor $\mathbf{h}^{(l+1)}$ nach der Berechnung wieder als $(H^{(l)} - F^{(l)} + 1) \times (B^{(l)} - F^{(l)} + 1)$ Matrix interpretiert. Für die Berechnung mit mehr als einer Feature-Map als Eingabe wird die zuvor beschriebene Umwandlung und Multiplikation für jede Feature-Map p durchgeführt und anschließend aufsummiert:

$$\mathbf{h}^{(l+1)} = \sum_{p} \mathbf{C}^{(p,l)} \cdot \mathbf{h}^{(p,l)}. \tag{2.4.6}$$

Bei Verwendung mehrerer Filter wird für jeden Filter k zur Berechnung der Feature-Map k die Multiplikation jeweils mit der Matrix $\mathbf{C}^{(p,k)}$ durchgeführt:

$$\mathbf{h}^{(l+1,k)} = \sum_{p} \mathbf{C}^{(p,l,k)} \cdot \mathbf{h}^{(p,l)} \tag{2.4.7}$$

Abbildung 2.4.3 zeigt ein Beispiel für die Faltung einer $3 \times 3 \times 1$ Eingabe mit einem 2×2 Filter als Matrixmultiplikation. Die Eingabe wird zu einem Spaltenvektor mit neun Komponenten. Als Ausgabe wird ein Spaltenvektor mit vier Komponenten berechnet, welcher als eine 2×2 Matrix interpretiert wird. Für die Transpose Convolution wird die

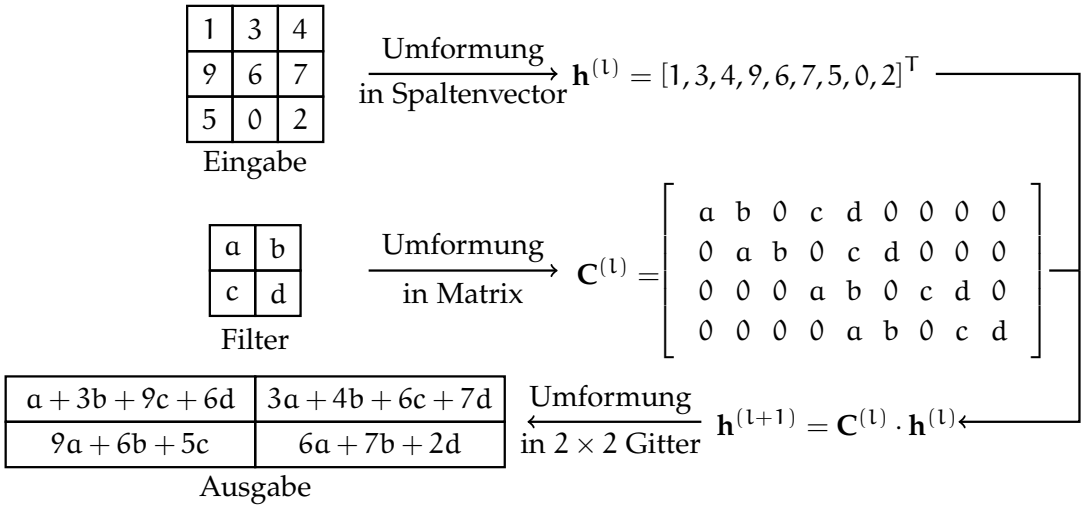


Abbildung 2.4.3: Visualisierung der Faltung als Matrixmultiplikation [Agg18].

Matrix $\mathbf{C}$ transponiert. Wird diese Berechnung auf den Gradienten $\mathbf{g}^{(l+1)}$ der Schicht $l+1$ angewendet, nachdem dieser in einen Spaltenvektor umgeformt wurde, kann der Gradient, wie in Abschnitt 2.3.2 beschrieben, durch das Netz zurück propagiert werden. Bei einer Ausgabe mit einer Tiefe $D^{(l+1)} > 1$ wird der Gradient zur Schicht l wie folgt zurück propagiert:

$$\mathbf{g}^{(p,l)} = \sum_{k=1}^{d} \mathbf{C}^{(l,p,k)T} \mathbf{g}^{(l+1,k)} \tag{2.4.8}$$

Der Filter und die Feature-Map der Ausgabe, deren Gradient zurück propagiert wird, wird mit k indiziert. Der Index p bestimmt die Feature-Map der Eingabe, für die der Gradient berechnet wird. Dieselbe Berechnung kann zusätzlich auch zur Berechnung eines *Upsamplings* verwendet werden. Die Größenveränderung von Ein- zu Ausgabe der Schicht ist dabei genau invers zur Faltung als Matrixmultiplikation mit der nicht transponierten Matrix. Im Beispiel aus Abbildung 2.4.3 wird durch die Faltung mit einem 2×2 Filter die Größe der Eingabe von 3×3 auf 2×2 verkleinert. Bei einer Transpose Convolution mit einem 2×2 Filter und einer Eingabe der Größe 2×2 hätte

die Ausgabe eine Größe von 3×3. Bei Verwendung der Transpose Convolution zum Upsampling ist es möglich, die Filter ebenfalls zu trainieren [LSD15].

2.4.4 *Dilated Convolution*

Eine spezielle Form der Faltung ist die *Dilated Convolution*, welche z. B. von Chen et al. [CPK+16] und Yu et al. [YK15] verwendet wird. Bei der Berechnung der Summe einer normalen Faltung werden benachbarte Gewichte im Filter mit dem benachbarten Pixeln bzw. Merkmalen der Eingabe multipliziert. Bei der Dilated Convolution werden die Gewichte des Filters mit weiter entfernten Merkmalen multipliziert. Der Filter hat somit Lücken bzw. Null-Gewichte [CPK+16]. Die Größe dieser Lücken wird durch den *Dilation-Factor* r als zusätzlichen Hyperparameter der Faltung festgelegt. Zwischen den Gewichten werden $r-1$ Nullen ergänzt. Folglich ist die normale Faltung auch eine Dilated Convolution mit dem Dilation-Factor $r = 1$. Ein Filter der Größe $F \times F$ wird vergrößert zu einem Filter der Größe $F_{\text{dilated}} \times F_{\text{dilated}}$ mit $F_{\text{dilated}} = F + (F-1)(r-1)$. Die Anzahl der Filtergewichte ungleich Null bleibt unabhängig vom Dilation-Factor erhalten. Für die Berechnung der Dilated Convolution wird jedoch nicht der Filter vergrößert und mit Null-Gewichten aufgefüllt, sondern die Faltungsoperation wie folgt verändert angewendet, um die Vergrößerung des Filters zu simulieren:

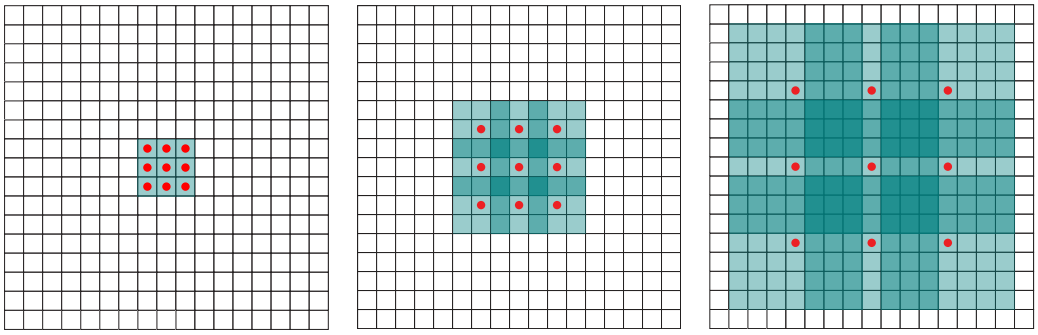
$$S(i,j) = (I \star K)(i,j) = \sum_m \sum_n I_{i+m,j+n} K_{m,n} \tag{2.4.9}$$

$$S(i,j) = (I \star_r K)(i,j) = \sum_m \sum_n I_{i+r\cdot m,j+r\cdot n} K_{m,n}. \tag{2.4.10}$$

Formel 2.4.9 zeigt die Berechnungsvorschrift der Faltung ohne Dilation-Factor mit einer zweidimensionalen Eingabe I und einem Filter K (nicht gespiegelt), wie diese in Abschnitt 2.4.1 definiert ist. In Formel 2.4.10 wurde die Definition um den Dilation-Factor r erweitert. Durch Veränderung der Indizierung des Pixel des Bildes kann eine Dilated Convolution berechnet werden, ohne den Filter zu verändern [YK15].

Die Verwendung von Dilated Convolutions ermöglicht eine exponentielle Vergrößerung des rezeptiven Feldes bei linearem Anstieg der Anzahl von Parametern ohne die Auflösung zu verringern (ausgenommen der Rand, falls kein Padding verwendet wird) [YK15]. Zudem können alle Pixel innerhalb des rezeptiven Feldes Einfluss auf die Berechnung haben. Um dies zu erzielen, können beispielsweise mehrere aufeinanderfolgende Faltungen mit 3×3 Filtern mit exponentiell steigendem Dilation-Factor $r = 2^i$ mit $i = 0, 1, 2, \ldots$ verwendet werden, wobei $i = 0$ der Eingabe entspricht. Das rezeptive Feld eines Merkmals nach i Faltungsschichten hat eine Größe von $(2^{i+1}-1) \times (2^{i+1}-1)$. Abbildung 2.4.4 zeigt ein Beispiel für die Anwendung von

3×3 Filtern mit verschiedenen Dilation-Factors und die dadurch erreichte exponentielle Vergrößerung des rezeptiven Feldes.



(a) (b) (c)

Abbildung 2.4.4: Beispiel für die exponentielle Vergrößerung des rezptiven Feldes durch Dilated Convolutions. Abbildung (a) zeigt das rezeptive Feld in Form von farblich hinterlegten Zellen bei Anwendung eines 3×3 Filters mit Dilation-Factor $r = 1$ an genau einer Position. Die Merkmale der Eingabe, mit denen die Gewichte des Filters multipliziert werden, sind durch Zellen mit roten Punkten markiert. Das rezeptive Feld hat nach dieser Faltung eine Größe von 3×3 Pixeln. Nach Anwendung einer weiteren Faltung mit einem 3×3 Filter mit einem Dilation-Factor von $r = 2$ beträgt die Größe des rezeptiven Feldes 7×7 Pixel, wie in Abbildung (b) dargestellt ist. Die Anwendung einer dritten Faltung mit einem 3×3 Filter und einem Dilation-Factor von $r = 4$ führt zur Vergrößerung des rezeptiven Feldes auf 15×15 Pixel, welches in Abbildung (c) hervorgehoben ist. Abbildungen entnommen aus [YK15].

2.5 NEURONALE NETZE ZUR KLASSIFIKATION

Semantische Segmentierung ist kein vollständig neues Problem des maschinellen Lernens, sondern kann als eine komplexere Form der Klassifikation betrachtet werden. Anstelle der Zuweisung eines oder mehrerer Klassenlabels für das gesamte Bild, wird jedem Pixel des Bildes eine Klasse zugewiesen [GGOEO+17]. In Folge dessen werden für viele Architekturen (z. B. [LSD15], [BKC17], [YK15], [CPK+16]) zur semantischen Segmentierung Modelle zur Klassifikation als Basis verwendet und modifiziert, um eine dichte pixelweise Ausgabe zu berechnen. Beispiele für Modelle sind das *AlexNet* [KSH12], welches 2012 die ImageNet Large Scale Visual Recognition Competition (ILSVRC) [RDS+15] mit einer Top-5 Testgenauigkeit von 84,6 % gewann und das

VGG-16 [SZ14], eine von mehreren von der *Visual Geometry Group* (VGG) vorgestellten Architekturen, welche bei der ILSVRC-2013 eine Top-5 Testgenauigkeit von 92,7 % erreichte [GGOEO+17] und im Folgenden detaillierter beschrieben wird. Weitere Architekturen sind das *GoogLeNet* [SLJ+15], welches die beste Top-5 Testgenauigkeit mit 93,3 % bei der ILSVRC-2014 erreichte, und das *ResNet* [HZRS16], welches die ILSVRC-2016 mit einer Top-5 Testgenauigkeit von 96,4 % gewann.

Das VGG-16 ist ein Modell, welches von der VGG zusammen mit weiteren ähnlichen Architekturen vorgestellt wurde [SZ14]. Andere Architekturen, welche bei der ILSVRC-2012 oder 2013 die besten Ergebnisse erzielt haben, verwenden in den Faltungsschichten zum Teil Filter mit einer Größe von 11×11 Pixeln (z. B. [KSH12]). Im Gegensatz dazu werden bei den Achitekturen der VGG nur 3×3 Filter verwendet [SZ14]. Die Neuronen haben durch diese Verringerung der Filtergröße ein kleineres rezeptives Feld, welches jedoch durch mehrfach aufeinanderfolgende Faltungen erhöht werden kann. Beispielsweise führen zwei aufeinanderfolgende Faltungen mit einer Filtergröße von 3×3 zu einem rezeptiven Feld der Größe 5×5, wie eine einzelne Faltung mit einem 5×5 Filter. Durch mehrfache Faltung mit einem kleinen 3×3 Filter kann die Anzahl der Parameter im Vergleich zu einer einzelnen Faltung mit demselben rezeptiven Feld verringert werden. Zudem werden nach jeder Faltungsschicht ReLU-Aktivierungsfunktionen als Nichtlinearität verwendet. Die somit häufigere Verwendung von Nichtlinearitäten führt zu einem besseren diskriminativem Verhalten. Aus diesen Gründen wurden Experimente mit unterschiedlich tiefen Architekturen durchgeführt.

Alle Architekturkonfigurationen nehmen ein 224×224 RGB-Bild als Eingabe. In den Faltungsschichten werden ausschließlich 3×3 Filter und ein Stride von 1 verwendet. Um die Größe der Feature-Maps nach der Faltung zu erhalten, wird vor jeder Faltung mittels Zero-Padding die Eingabefeature-Map um einen ein Pixel breiten Rand vergrößert. Die Anzahl der Feature-Maps variiert abhängig von der Architekturkonfiguration. In jeder Konfiguration werden fünf Pooling-Schichten verwendet, welche ein Max-Pooling auf einem 2×2 Gitter mit einem Stride von 2 durchführen. Nach den Faltungs- und Pooling-Schichten folgen drei vollständig verbundene Schichten mit 4096, 4096 und 1000 Neuronen, wobei die Anzahl der Neuronen der letzten Schicht der Anzahl der Klassen im ILSVRC Datensatz [DDS+09] entspricht. Simonyan und Zisserman [SZ14] stellen Modelle mit 11, 13, 16 und 19 Faltungsschichten vor, wobei auf jede Faltungsschicht eine ReLU-Aktivierungsfunktion folgt.

In der ILSVRC-2014 [RDS+15] erreichten die VGG-Modelle (als Ensemble) mit einem Top-5 Testfehler von 7,32 % eine deutlich bessere Leistung als das beste Modell (*Clarifai*) des Vorjahres mit einem Top-5 Fehler von 11,74 %. Das GoogLeNet [SLJ+15] erreichte 2014 mit 6,66 % einen besseren Top-5 Fehler als die VGG-Netze. Das Modell,

welches als Basis der in dieser Arbeit betrachteten Architekturen dient, ist das VGG-16 (zweit tiefstes Modell der VGG). Abbildung 2.5.1 zeigt den Aufbau dieses Netzes.

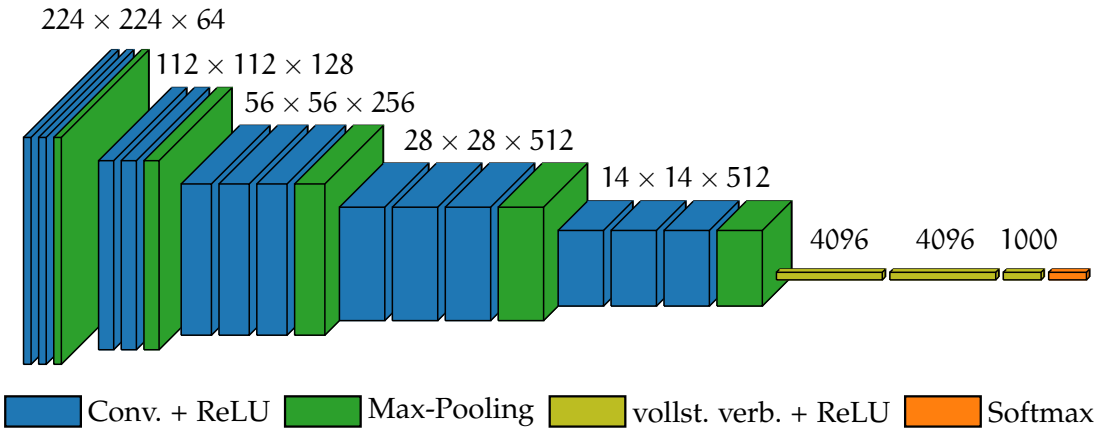


Abbildung 2.5.1: Visualisierung der Architektur des VGG-16 Netzes [SZ14]. Bei den Faltungsschichten sind die Größen der Feature-Maps angegeben als $H \times B \times D$, wobei $H \times B$ den zwei Dimensionen der Eingabe entsprechen und D die Anzahl der Kanäle bzw. die Anzahl der Feature-Maps angibt. Bei den vollständig verbundenen Schichten ist die Anzahl der Neuronen angegeben.

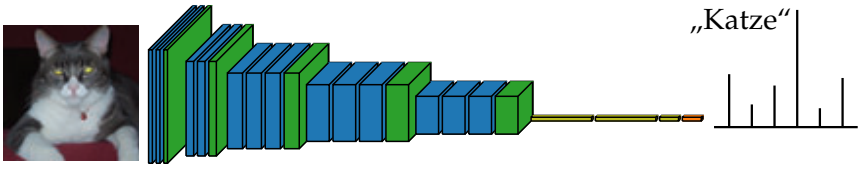
3 VERWANDTE ARBEITEN

In diesem Kapitel werden Modelle und Methoden vorgestellt, welche zum Teil Einfluss auf die Methodik dieser Arbeit haben. In Abschnitt 2.5 wurden Architekturen zur Klassifikation beschrieben, welche als Basis vieler Architekturen zur semantischen Segmentierung dienen. In Abschnitt 3.1 wird erläutert, mit welchen Veränderungen diese Modelle zur semantischen Segmentierung verwendet werden können und es werden Beispiele für Architekturen beschrieben. Anschließend werden in Abschnitt 3.2 die unterschiedlichen Level von Annotationen und der Annotationsaufwand diskutiert. Zudem werden zwei verschiedene Ansätze zum Umgang mit schwachen Annotationen für das Training beschrieben.

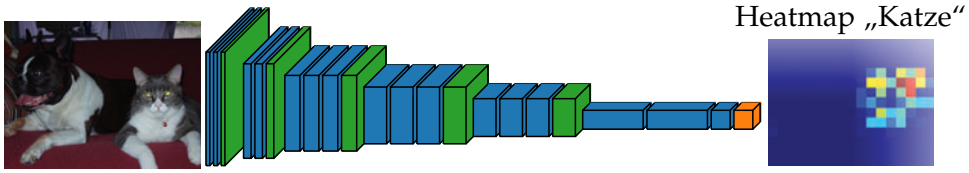
3.1 TIEFE NETZE ZUR SEMANTISCHEN SEGMENTIERUNG

Die Verwendung von tiefen neuronalen Netzen hat in verschiedenen Bereichen des maschinellen Sehens zu großen Fortschritten geführt [HKH17], so auch bei dem Problem der semantischen Segmentierung. Erfolgreiche Architekturen zur Klassifikation, wie z. B. das AlexNet [KSH12] oder das VGG-16 [SZ14], werden als Basis für viele Architekturen zur semantischen Segmentierung verwendet. Dies erfordert jedoch Anpassungen der Architektur, um eine räumliche Ausgabe in Form einer (dichten) Klassifikationskarte zu erzeugen, da Architekturen zur Klassifikation nur Klassenwahrscheinlichkeiten für das gesamte Bild ausgeben [LSD15]. Ein weiteres Problem bei vielen Architekturen liegt in der Beschränkung auf eine Eingabe fester Größe. Beide Probleme entstehen durch die Verwendung vollständig verbundener Schichten. Die Berechnung einer vollständig verbundenen Schicht kann jedoch auch als Faltung interpretiert werden, bei der die Filtergröße der Größe der Eingabe der vollständig verbundenen Schicht entspricht. Beispielsweise haben die Feature-Maps nach dem fünften Max-Pooling im VGG-16 eine Größe von 7×7 bei einer festgelegten Größe der Eingabe von 224×224 Pixeln [SZ14]. Die Berechnung der darauffolgenden vollständig verbundenen Schicht ist analog zu einer Faltung mit einem 7×7 Filter pro Neuron. Da die Schicht 4096 Neuronen enthält, werden nach der Umwandlung in eine Faltungsschicht mit einem $7 \times 7 \times 4096$ Filter 4096 Feature-Maps berechnet. Analoges gilt für die beiden weiteren vollständig verbundenen Schichten. Durch die

ausschließliche Verwendung von Faltungsschichten können Eingaben beliebiger Größe verarbeitet werden [LSD15]. Als Ausgabe berechnet das Netz nach der Umwandlung eine Klassifikationskarte bzw. eine *Heatmap* mit Wahrscheinlichkeiten für jede Klasse. Bei Berechnung der Wahrscheinlichkeiten mittels einer Softmax-Aktivierungsfunktion wird diese für jedes Pixel unabhängig von den anderen Pixeln berechnet [BKC17]. Die Heatmap hat durch die Verwendung von Subsampling, wie z. B. Max-Pooling mit einem Stride $S > 1$, eine geringere Auflösung als das Eingabebild. Jeder Wert der Ausgabe entspricht dabei der Ausgabe des ursprünglichen Netzes auf einem bestimmten Teil des Eingabebildes [LSD15]. Diese Umwandlung von vollständig verbundenen Schichten in Faltungsschichten wird im Folgenden als *Convolutionalization* bezeichnet und kann die Berechnung auf dem gesamten Bild im Vergleich zur Berechnung auf jedem möglichen Teil der Eingabe mit dem ursprünglichen Netz stark beschleunigen [LSD15]. Abbildung 3.1.1 zeigt ein Beispiel für die Convolutionalization und den Effekt auf die Ausgabe bei einem größeren Eingabebild.



(a) Netz zur Klassifikation mit vollständig verbundenen Schichten.



(b) Netz, bei dem die vollständig verbundenen Schichten in Faltungsschichten umgewandelt wurden.

Abbildung 3.1.1: Umwandlung der vollständig verbundenen Schichten eines Netzes zur Klassifikation in Faltungsschichten. Vor der Umwandlung (a) kann nur ein Teil des Bildes einer festgelegten Größe als Eingabe verwendet werden. Die Ausgabe ist ein Klassenlabel bzw. Wahrscheinlichkeiten für die einzelnen Klassen. Nach der Umwandlung (b) können größere Eingabebilder verarbeitet werden, wobei die Ausgabe einer grob aufgelösten Heatmap für jede Klasse entspricht. Teile der Abbildungen sind entnommen aus [LSD15].

Die Verwendung tiefer Faltungsnetze zur Klassifikation als Basis für Modelle zur semantischen Segmentierung, erfordert Veränderungen der Architektur der Modelle,

bietet jedoch auch die Möglichkeit des *Transfer Learnings* [GGOEO+17]. Eine Variante des Transfer Learnings, welche zu diesem Zweck häufig eingesetzt wird, ist die Initialisierung der Gewichte eines Modells mit Gewichten, welche auf einem anderen Datensatz vortrainiert wurden. Ein möglicher Grund für diese Initialisierung, der insbesondere beim Training tiefer Faltungsnetze zur semantischen Segmentierung auftritt, ist die benötigte Menge von Trainingsdaten. Die Verfügbarkeit und der Umfang vollständig annotierter Datensätze ist deutlich geringer als die der Datensätze für die Klassifikation (z. B. ImageNet [DDS+09, RDS+15]). Transfer Learning ist auch zwischen verschiedenen Aufgaben (z. B. Klassifikation und Segmentierung) möglich [YCBL14], jedoch ist auf die Wahl der Hyperparameter zu achten [GGOEO+17]. Zudem müssen die Architekturen des vortrainierten Netzes und des weiter trainierten Netzes soweit übereinstimmen, dass die Gewichte übernommen werden können. Bei vielen Ansätzen ist diese Initialisierung mit vortrainierten Gewichten möglich (z. B. [LSD15], [BKC17], [YK15], [CPK+16]) und wurde von den Autoren durchgeführt, da nur teilweise veränderte Modelle zur Klassifikation (z. B. [KSH12], [SZ14], [SLJ+15], [HZRS16]) verwendet wurden.

Im Folgenden werden Architekturen zur semantischen Segmentierung erläutert, welche alle unter auf dem VGG-16 basieren. Die gemeinsamen Vorgänger dieser Architekturen sind die Fully Convolutional Networks [LSD15], welche im folgenden Abschnitt erläutert werden. Unterschiedliche Faltungsnetze zur Klassifikation werden mittels Convolutionalization zur Vorhersage von Klassifikationskarten angepasst, deren Auflösung anschließend mittels Transpose Convolution erhöht wird. Als Architekturen, welche das Subsampling verringern, andere Methoden zum Upsampling oder eine zusätzliche Nachbearbeitung verwenden, um die Qualität der Ausgabe zu verbessern, werden zudem das Dilated Network [YK15], das SegNet [BKC17] und das DeepLab [CPK+16] beschrieben.

3.1.1 *Fully Convolutional Networks*

Die Fully Convolutional Networks (FCN) [LSD15] basieren auf Faltungsnetzen zur Klassifikation, welche für die ImageNet Large Scale Visual Recognition Competition (ILSVRC) [RDS+15] trainiert wurden. Die Anpassung ist für verschiedene Netze möglich und wurde von Long et al. [LSD15] für das AlexNet [KSH12], das GoogLeNet [SLJ+15] und das VGG-16 [SZ14] durchgeführt. Die letzte Schicht aller Netze, welche zur Klassifikation dient, wurde entfernt. Beim GoogLeNet wurde das letzte globale Average-Pooling verworfen. Die vollständig verbundenen Schichten wurden, wie im vorherigen Abschnitt beschrieben, in Faltungsschichten umgewandelt. Zur

Abbildung der berechneten Merkmale auf Vorhersagen für die einzelnen Klassen wurde eine Faltungsschicht mit einem 1×1 Kernel pro Klasse ergänzt. Die Auflösung der daraus resultierenden Ausgabe ist durch Subsampling wie z. B. Pooling mit einem Stride $S > 1$ jedoch geringer als die der Eingabe. Bei einem VGG-16, welches wie beschrieben in ein FCN umgewandelt wurde, beträgt beispielsweise die Auflösung der Ausgabe nur $1/32$ der Eingabegröße. Um die Auflösung der Ausgabe zu erhöhen, sodass diese der Auflösung des Eingabebildes entspricht, wurde von Long et al. *Transpose Convolution* (vgl. Abschnitt 2.4.3) verwendet. Im Vergleich der angepassten Versionen des AlexNet, GoogLeNet und des VGG-16 wurden die besten Ergebnisse mit dem VGG-16 erreicht.

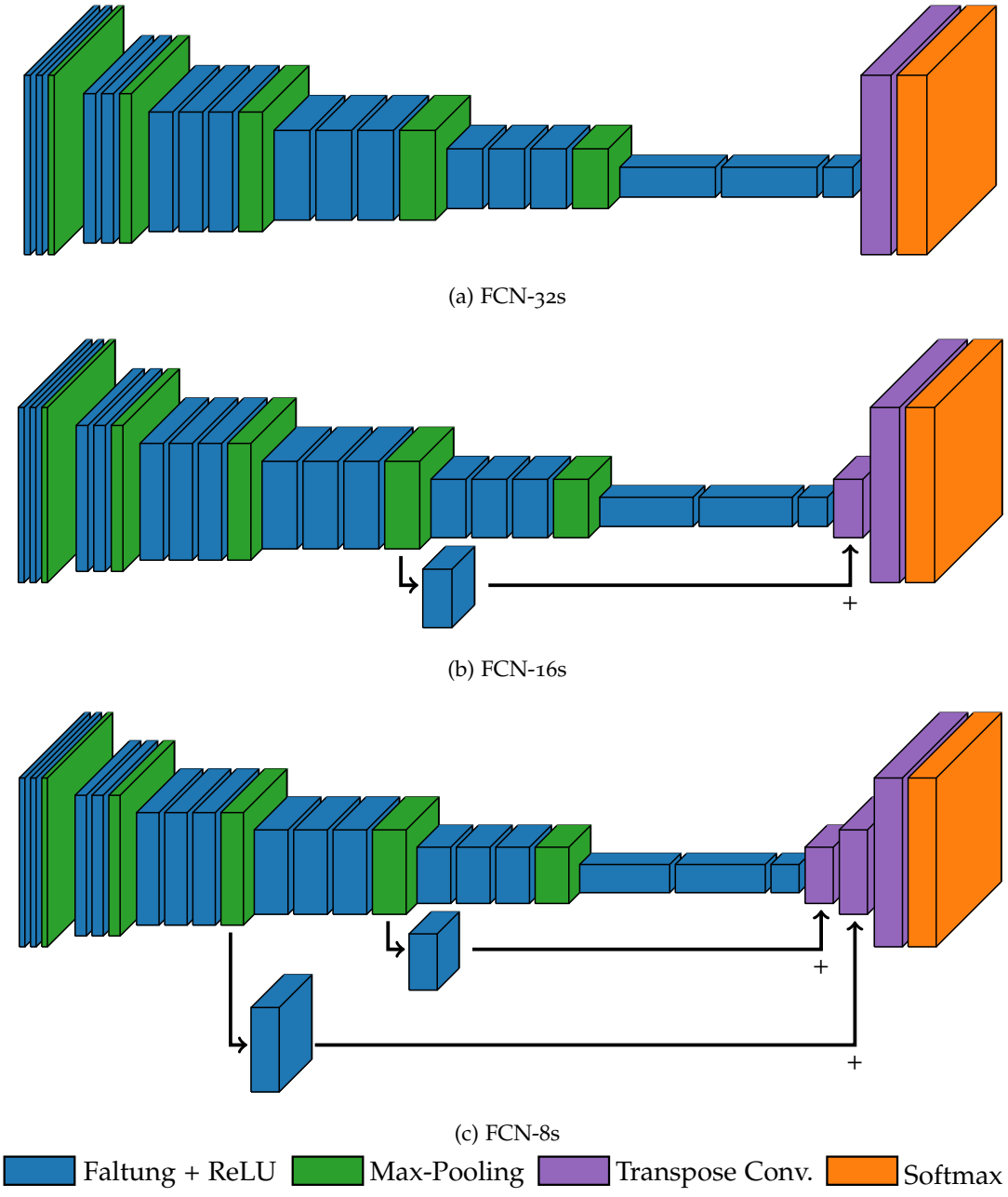
Das VGG-16 hat 16 Faltungsschichten, jeweils gefolgt von einer elementweisen ReLU-Aktivierungsfunktion [SZ14]. In den Faltungsschichten werden ausschließlich 3×3 Filter und ein Stride von 1 verwendet. Um die Größe der Feature-Maps durch die Faltungsschichten nicht zu verändern, wird Zero-Padding auf einem ein Pixel breiten Rand vor jeder Faltung verwendet. Insgesamt wird fünf mal ein Max-Pooling mit einem 2×2 Kernel und einem Stride von 2 durchgeführt. Die Verwendung eines Strides von 2 führt bei jedem Max-Pooling jeweils zu einer Halbierung der Größe der Feature-Maps. Die Anzahl der verwendeten Filter in jeder Schicht ist in Tabelle 3.1.1 aufgeführt.

Zur Anpassung für die Ausgabe einer dichten Klassifkationskarte werden die vollständig verbundenen Schichten des VGG-16 in Faltungsschichten umgewandelt. Die letzte vollständig verbundene Schicht wird durch eine Faltungsschicht mit 1×1 Filtern ersetzt, wobei die Anzahl der Filter der Anzahl der Klassen im Datensatz (21 Klassen für den Pascal VOC Datensatz [EVGW+10]) entspricht. Das rezeptive Feld der Neuronen dieser Schicht hat durch das fünffache Max-Pooling einen Stride von 32 Pixeln auf dem Eingabebild und die Größe der berechneten Feature-Maps entspricht nur $1/32$ der Auflösung des Eingabebildes. Die veränderte Version des VGG-16 wird aus diesem Grund im Folgenden als FCN-32s bezeichnet. Um die Auflösung um den Faktor 32 zu erhöhen, wird eine Transpose Convolution mit einem 64×64 Filter und einem Stride von 32 angefügt. Die Gewichte dieser Schicht werden dabei mit Gewichten für eine bilineare Interpolation initialisiert. Zur Berechnung von Klassenwahrscheinlichkeiten wird nach dem Upsampling eine Softmax-Aktivierungsfunktion angewendet. Die Architektur des FCN-32s ist in Abbildung 3.1.2a dargestellt.

Die Erhöhung der Auflösung um den Faktor 32 beschränkt den Detailgrad, der durch das Upsampling erreicht werden kann. Schichten, welche nahe der Ausgabe liegen und auf viele Faltungsschichten folgen, werden im Folgenden anhand der Berechnungsreihenfolge im Netz als *späte Schicht* bezeichnet. Die Feature-Maps dieser Schichten weisen eine niedrige Auflösung und Merkmale mit einem großen rezeptiven

Tabelle 3.1.1: Aufbau des VGG-16 [SZ14] und des Fully Convolutional Networks basierend auf dem VGG16. Die Hyperparameter zu den Faltungsschichten werden angegeben als conv⟨Filtergröße⟩-⟨Anzahl der Feature-Maps⟩. Nach jeder versteckten Schicht wird eine ReLU-Aktivierungsfunktion elementweise angewendet.

VGG-16	Fully Conv. VGG-16
RGB Bild (224×224)	RGB Bild beliebiger Größe
conv3-64	conv3-64
conv3-64	conv3-64
Max-Pooling (112×112)	Max-Pooling (1/2 Größe)
conv3-128	conv3-128
conv3-128	conv3-128
Max-Pooling (56×56)	Max-Pooling (1/4 Größe)
conv3-256	conv3-256
conv3-256	conv3-256
conv3-256	conv3-256
Max-Pooling (28×28)	Max-Pooling (1/8 Größe)
conv3-512	conv3-512
conv3-512	conv3-512
conv3-512	conv3-512
Max-Pooling (14×14)	Max-Pooling (1/16 Größe)
conv3-512	conv3-512
conv3-512	conv3-512
conv3-512	conv3-512
Max-Pooling (7×7)	Max-Pooling (1/32 Größe)
vollst. verbunden 4096	conv7-4096 (Padd.= 0)
vollst. verbunden 4096	conv1-4096 (Padd.= 0)
vollst. verbunden 1000	conv1-21 (Padd.= 0)
	transp. conv64-21 (urspr. Größe)
Softmax	Softmax



(a) FCN-32s

(b) FCN-16s

(c) FCN-8s

Abbildung 3.1.2: Visualisierung der Architektur der Fully Convolutional Networks [LSD15], basierend auf dem VGG-16.

Feld auf. Folglich enthalten die berechneten Merkmale globale Informationen, welche jedoch nur wenig genau lokalisiert sind. Schichten nahe der Eingabe, welche auf wenige/keine Faltungsschichten folgen, werden im Folgenden als *frühe Schichten* bezeichnet. Die berechneten Feature-Maps haben eine höhere Auflösung, jedoch Merkmale mit einem kleineren und somit einem lokalen rezeptiven Feld. Um feinere Details mit einzubeziehen, werden wie von Long et al. [LSD15] vorgestellt, zusätzliche Verbindungen mit höher aufgelösten Ausgaben früherer Schichten eingefügt. Somit sollen die lokalen Informationen früher Schichten mit den globalen Informationen der späteren Schichten kombiniert werden. Die Abbildungen 3.1.2b und 3.1.2c zeigen die Erweiterung des FCN-32s um diese Verbindungen. Beim FCN-16s wird dazu die Ausgabe des vierten Max-Poolings verwendet, deren Merkmale ein rezeptives Feld mit einem Stride von 16 Pixeln im Eingabebild aufweisen. Um Scores für die einzelnen Klassen zu erhalten, wird eine Faltung mit einem 1×1 Filter pro Klasse im Datensatz auf die ausgegebenen Feature-Maps angewendet. Die Transpose Convolution vor dem Softmax wird dahingehend verändert, dass die Auflösung verdoppelt wird. Vor einem weiteren Upsampling mittels Transpose Convolution auf die Auflösung der Eingabe, werden die berechneten Scores nach dem vierten Max-Pooling auf die Scores nach der ersten Transpose Convolution addiert. Eine analoge Modifikation des FCN-16s wird durchgeführt, um die Merkmale nach dem dritten Max-Pooling ebenfalls über eine zusätzliche Verbindung am Ende des Netzes zu verwenden. Die daraus resultierende Architektur wird als FCN-8s bezeichnet und ist in Abbildung 3.1.2c dargestellt.

Die drei Architekturen wurden von Long et al. [LSD15] auf dem Pascal VOC 2011 Datensatz [EVGW+11] mit zusätzlichen Annotationen [HAB+11] trainiert und auf einem Teil des Pascal VOC 2011 Validierungsdatensatzes, welcher keine der zusätzlich annotierten Bilder enthält, evaluiert. Mit dem FCN-32s wurde eine Mean Intersection over Union (MIoU) von 59,4 % erreicht. Durch das Einfügen der zusätzlichen Verbindung zu früheren Schichten konnte die MIoU auf 62,4 % beim FCN-16s und 62,7 % beim FCN-8s verbessert werden.

3.1.2 *Dilated Network*

Das Dilated Network (DNet) wurde von Yu et al. [YK15] vorgestellt. Die Benennung der Architektur von Yu et al. als Frontend wird in dieser Arbeit nicht verwendet. Als Basis dient das VGG-16 zur Klassifikation. Das Vorgehen zur Anpassung für die semantische Segmentierung wird wie im vorherigen Abschnitt beschrieben durchgeführt. Bei dem daraus resultierenden FCN-32s ist der Detailgrad der Ausgabe durch das fünffache Subsampling durch Max-Pooling und nur einer Transpose Convolution zum

Upsampling beschränkt. Die häufige Verwendung des Max-Poolings wurde bei dem Entwurf eines neuronalen Netzes zu Klassifikation vorgestellt. Für die semantische Segmentierung, welche die Ausgabe einer hoch aufgelösten Klassifikationskarte erfordert, ist eine derartig häufige Verwendung ungeeignet [YK15]. Aus diesem Grund werden beim DNet das vierte und fünfte Max-Pooling entfernt. Um das rezeptive Feld zu erhalten, wird stattdessen in den darauffolgenden Faltungsschichten für jedes ersetzte Max-Pooling der Dilation-Factor um den Faktor 2 erhöht. Der Dilation-Factor der drei Faltungsschichten zwischen der entfernten vierten und fünften Max-Pooling-Schicht wird auf 2 und der der Faltungsschicht nach dem fünften Max-Pooling auf 4 gesetzt. Da die beiden darauffolgenden Faltungsschichten Filter der Größe 1×1 verwenden, ist die Berechnung dieser Schichten unabhängig vom gewählten Dilation-Factor. Aus diesem Grund ist keine Anpassung des Dilation-Factors dieser Schichten erforderlich. Zudem wird auf Padding bei allen Faltungsschichten des Netzes verzichtet. Stattdessen wird in einem Vorverarbeitungsschritt das Eingabebild mittels Reflection-Padding vergrößert. Der Aufbau des DNets ist in Abbildung 3.1.3 dargestellt. Die Abbildung zeigt, dass nach dem dritten Max-Pooling die Auflösung der Feature-Maps nicht mehr halbiert wird. Durch das fehlende Padding innerhalb des Netzes wird die Auflösung der Feature-Maps durch jede Faltungsschicht verkleinert, was jedoch zur Vereinfachung der Grafik nicht dargestellt ist. Nach dem Training des DNets mit den erweiterten Trainingsdaten des Pascal VOC 2012 Datensatzes [EVGW+12, HAB+11] wurde mit dem Modell auf dem Pascal VOC 2012 Testdatensatz eine MIoU von 67,6 % erreicht [YK15].

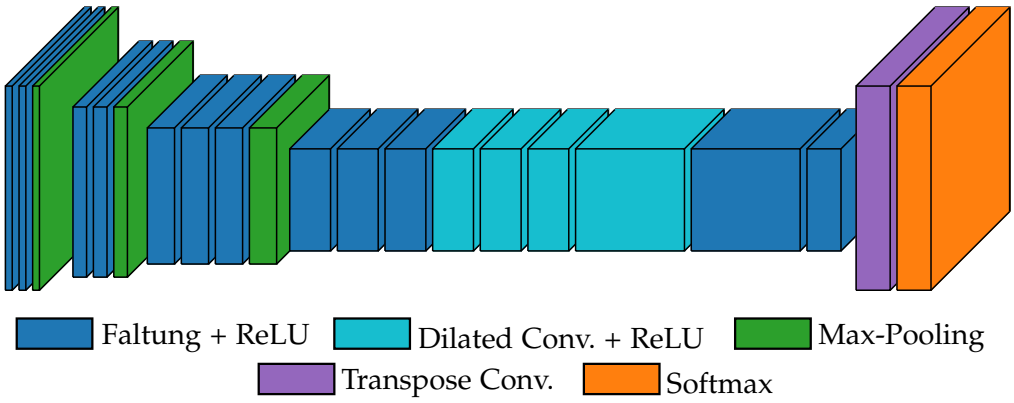


Abbildung 3.1.3: Visualisierung der Architektur des DNets [YK15].

3.1.3 *SegNet*

Eine weitere Architektur, welche auf dem VGG-16 basiert, ist das SegNet [BKC17]. Im Gegensatz zum FCN-32s und dem DNet werden die vollständig verbundenen Schichten des VGG-16 jedoch nicht in Faltungsschichten umgewandelt, sondern aus dem Netz entfernt. Das SegNet ist in einen *Encoder*, welcher auf den Faltungsschichten des VGG-16 basiert und zur Berechnung von Merkmalen dient, und einen *Decoder* aufgeteilt, der zum Upsampling der Feature-Maps dient, die vom Encoder berechnet werden. Dieser Aufbau ist in Abbildung 3.1.4 dargestellt. Nach jeder Faltungsschicht wurde sowohl im Encoder als auch im Decoder eine Batch-Normalisierung [IS15, BMC15] durchgeführt. Durch das fünffache Max-Pooling mit einem Stride von 2 im Encoder wird das rezeptive Feld der Merkmale erhöht und die Verschiebungsinvarianz der Merkmale verbessert. Das Max-Pooling führt jedoch zum Verlust von Informationen zur Lokalisierung der Merkmale. Um diese zu erhalten, wird bei jedem Max-Pooling der Index des maximalen Werts im Pooling-Bereich gespeichert. Der Decoder ist symmetrisch zum Encoder aufgebaut. Zur Erhöhung der Auflösung hat jede Max-Pooling-Schicht im Encoder eine korrespondierende Max-Unpooling Schicht im Decoder. Bei dem Max-Unpooling werden die gespeicherten Indizes des korrespondierenden Max-Poolings verwendet, um die Merkmale an die entsprechenden Positionen in Feature-Maps mit erhöhter Auflösung zu positionieren. Die daraus resultierenden Feature-Maps sind dünn besetzt. Um vollständig besetze Feature-Maps zu erhalten, folgen nach jedem Max-Unpooling mehrere Faltungsschichten. Zur Berechnung von Klassenwahrscheinlichkeiten für die einzelnen Pixel wird eine Softmax-Aktivierungsfunktion auf die Ausgabe des Decoders angewendet.

Das SegNet wurde von Badrinarayanan et al. [BKC17] auf Bildern von Straßenszenen und von Wohnräumen evaluiert. Für das Training zur Segmentierung von Bildern von Straßenszenen wurde ein Datensatz mit 3.433 Bildern aus mehreren Datensätzen ([BFC09], [GLU12], [GFK09], [RTMF08]) zusammengestellt. Bei dem Test des trainierten Modells auf dem CamVid [BFC09] Testdatensatz wurde eine MIoU von 60,1 % erreicht. Für das Training auf Bildern von Wohnräumen wurde der SUNRGB-D [SLX15] Datensatz verwendet, wobei die Tiefeninformationen verworfen wurden. Auf dem SUNRGB-D Testdatensatz wurde mit dem SegNet eine MIoU von 31,84 % erreicht.

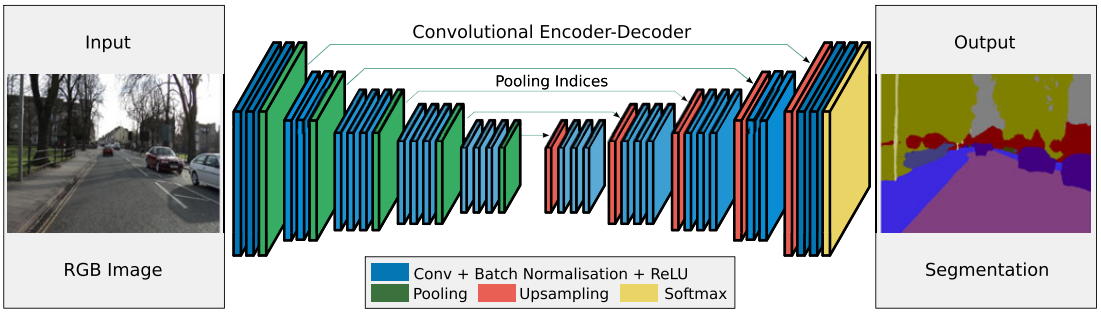


Abbildung 3.1.4: Aufbau des SegNet. Die tiefen der jeweils berechneten Feature-Maps sind aus Gründen der Übersichtlichkeit nicht dargestellt. Abbildung entnommen aus [BKC17].

3.1.4 *DeepLab*

Ähnlich wie Yu et al. [YK15] beim DNet, verwenden Chen et al. [CPK+16] ebenfalls Dilated Convolutions in den von ihnen vorgestellten Varianten des DeepLab. Als Basis dienen das VGG-16 [SZ14] für das im Folgenden betrachtete DeepLab und das neuere ResNet-101 [HZRS16] für eine verbesserte Version des DeepLab. Um eine Halbierung der Auflösung durch Faltungsschichten oder Max-Pooling-Schichten mit einem Stride von 2 zu verhindern, kann bei diesen Schichten der Stride auf 1 gesetzt werden. Um das rezeptive Feld zu erhalten, muss der Dilation-Factor der folgenden Schichten um den Faktor 2 erhöht werden. Mit diesem Vorgehen ist es möglich, die Auflösung des Eingabebilds bei jeder Feature-Map und der Ausgabe zu erhalten. Aus Gründen der Effizienz wurde dies wie beim DNet bis zu einem Dilation-Factor von $r = 4$ durchgeführt. In Folge dessen ist die Auflösung der Ausgabe um den Faktor 8 (statt 32 ohne Dilated Convolution) kleiner als die der Eingabe und wird mittels bilinearer Interpolation auf die Auflösung der Eingabe erhöht. Um weitere Details zu erhalten wenden Chen et al. [CPK+16] das Modell eines *fully connected conditional random fields* (FC-CRF) von Krähenbühl und Koltun [KK11] auf die Ausgabe des Netzes an. Das FC-CRF wurde getrennt vom neuronalen Netz trainiert. Beim DeepLab, welches auf dem VGG-16 basiert, wurden zudem Experimente mit der Filtergröße und dem Dilation-Factor der ehemaligen ersten vollständig verbundenen Schicht „fc6“ durchgeführt. Im VGG-16 hat diese Schicht eine 7×7 Pixel große Eingabe. Die Faltungsschicht nach der Convolutionalization hat folglich eine Filtergröße von 7×7. Chen et al. [CPK+16] untersuchten die Veränderung der MIoU und der Zeit, welche für die Auswertung eines Bildes benötigt wurde, für unterschiedliche Filtergrößen und Dilation-Factors. Das DeepLab-LargeFOV verwendet in dieser Schicht 3×3 Filter

mit einem Dilation-Factor von $r = 12$ anstelle von $r = 4$, sodass das rezeptive Feld der Merkmale erhalten bleibt. Zudem wurde die Anzahl der Filter und somit die Anzahl der Feature-Maps in der „fc6"-Schicht und der folgenden Schicht von 4096 auf 1024 verringert. In Folge dessen wird die Anzahl der Parameter stark reduziert und die Geschwindigkeit der Auswertung mehr als verdreifacht.

Die Architekturvarianten wurden auf dem Pascal VOC 2012 Datensatz [EVGW+12] mit zusätzlichen Annotationen [HAB+11] für weitere Bilder des Datensatzes trainiert. Bei der Evaluierung auf dem Pascal VOC 2012 Validierungsdatensatz wurde mit dem DeepLab, basierend auf dem VGG-16 mit einem 7×7 Filter und einem Dilation-Factor von 4 in der „fc6"-Schicht, eine MIoU von 64,38 % vor und 67,64 % nach Anwendung des FC-CRFs erreicht. Mit dem DeepLab-LargeFOV wurde vor Anwendung des FC-CRFs eine MIoU von 62,25 % und bei zusätzlicher Verwendung des FC-CRFs (DeepLab-CRF-LargeFOV) ebenfalls eine MIoU von 67,64 % erzielt.

3.2 SCHWACH ÜBERWACHTE SEMANTISCHE SEGMENTIERUNG

Eine große Schwierigkeit beim Training tiefer neuronaler Faltungsnetze ist die Menge annotierter Trainingsdaten. Insbesondere die hohe Anzahl von Parametern solcher Modelle birgt die Notwendigkeit großer Datensätze, um Probleme wie z. B. Overfitting (vgl. Abschnitt 2.3.3) zu vermeiden [HKH17]. Zum Training von Faltungsnetzen zur semantischen Segmentierung werden Annotationen für jedes Pixel in jedem Bild des Datensatzes benötigt. Diese *vollständigen Annotationen* werden auch als *Pixel-Level* oder *pixelweise* Annotation bezeichnet. Abbildung 3.2.1b zeigt ein Beispiel für die pixelweise Annotation des Bildes 3.2.1a. Die manuelle Erstellung solcher Annotationen ist sehr aufwändig. Insbesondere liegt die Schwierigkeit im Erhalt einer gleichbleibend guten Qualität der Annotationen. Des Weiteren erfordert die Annotation für bestimmte Anwendungen, wie z. B. die Analyse medizinischer Aufnahmen, domänenspezifisches Expertenwissen.

3.2.1 *Level von Annotationen*

Neben Pixel-Level Annotationen gibt es noch weitere Formen der Annotation, welche als *schwache Annotationen* bezeichnet werden. Im Vergleich zu vollständigen Annotationen ist der Annotationsaufwand geringer, jedoch enthalten diese weniger Informationen [HKH17]. Die einfachste Form der schwachen Annotationen sind *Bild-Level-Annotationen* (s. Abbildung 3.2.1c). Diese geben an, welche Klassen in einem Bild vorhanden bzw. nicht vorhanden sind. Folglich fehlen Informationen zu Ausdeh-

(a) RGB-Bild (b) Pixel-Level (c) Bild-Level (d) Punkt-Level (e) Bounding Boxes (f) Krakel

Abbildung 3.2.1: Beispiele für unterschiedliche Level der Annotationen. Abbildungen entnommen aus [HKH17].

nung und Position der vorhandenen Objekte. *Punktannotationen* (s. Abbildung 3.2.1d) enthalten, zusätzlich zu den Informationen zu An- bzw. Abwesenheit von Klassen, Wissen über die räumliche Position der vorhandenen Klassen. Diese können z. B. als ein annotiertes Pixel pro vorhandener Klasse oder auch als ein annotiertes Pixel pro vorhandener Instanz einer Klasse gegeben sein [BRFFF16]. Als Analogon zum Zeigen des Menschen auf ein Objekt, sind Punktannotationen eine natürliche Form der Annotation. Zusätzliche Informationen zur Ausdehnung vorhandener Objekte ist bei *Bounding-Box-Annotationen* (s. Abbildung 3.2.1e) gegeben [HKH17]. Dazu werden vorhandene Objekte jeweils mit einem rechteckigen Rahmen markiert, sodass das jeweilige Objekt vollständig innerhalb des Rechtecks liegt. Eine Mischung aus Punkt- und Bounding-Box-Annotationen bilden die *Krakelannotationen* (engl. scribbles). Bei dieser Form der Annotation wird mittels einer frei gezeichneten Linie (Krakel) das Objekt und dessen Ausdehnung markiert (s. Abbildung 3.2.1f). Mit mehreren annotierten Pixeln pro Objekt enthalten diese Annotationen mehr Informationen als die Punktannotationen. Im Vergleich zu Bounding-Box-Annotationen sind die Informationen zur Ausdehnung des Objektes ungenauer. Durch die freie Wahl des Verlaufs der Linie ist es jedoch besser möglich, Objekte beliebiger Form zu markieren als mit rechteckigen Rahmen.

3.2.2 *Vergleich der Annotationsaufwände*

Die Aufwände zur Erstellung der verschiedenen zuvor erwähnten Annotation unterscheiden sich zum Teil stark und werden im Folgenden miteinander am Beispiel des *Pascal Visual Object Classes (VOC) 2012* Datensatzes [EVGW+12, EVGW+10] verglichen. Neben dem Level der zu erstellenden Annotation ist der Aufwand ebenfalls vom

zu annotierenden Datensatz abhängig. Kriterien für den Aufwand zur Annotation einzelner Bilder des Datensatzes sind die Anzahl der Klassen des Datensatzes, die mittlere Anzahl von Klassen pro Bild und die mittlere Anzahl von Objektinstanzen pro Bild [BRFFF16]. Der Pascal VOC 2012 Datensatz hat Bilder mit 20 Objektklassen, wobei jedes Bild im Durchschnitt 1,5 Klassen und 2,8 Objektinstanzen enthält [EVGW+10]. Alle Pixel, welche keiner dieser 20 Klassen zuzuordnen sind, sind einer generischen Hintergrundklasse zugeordnet. Die aufwändigste Form der Annotation ist die pixelweise Annotation. Bei der Annotation des COCO-Datensatzes wurden 22 Stunden für 1000 Segmentierungen benötigt [LMB+14]. Für die Annotation einer einzelnen Objektsegmentierung ergeben sich somit 79 Sekunden. Zusätzlich wird pro nicht vorhandener Klasse (durchschnittlich 18,5 Klassen pro Bild) eine Sekunde benötigt [PCKF14]. Für den Pascal VOC Datensatz bedeutet dies einen Aufwand von $2{,}8 \cdot 79\,\mathrm{s} + 18{,}5\,\mathrm{s} = 239{,}7\,\mathrm{s}$ pro Bild [BRFFF16]. Der Aufwand zur Erstellung schwacher Annotationen ist deutlich geringer. Zur Annotation mit Bild-Level-Annotationen wird pro vorhandener und nicht vorhandener Klasse jeweils eine Sekunde benötigt [PCKF14]. Folglich dauert die Erstellung von Bild-Level-Annotationen für den Pascal VOC Datensatz 20 Sekunden pro Bild. Bei Erstellung von Punktannotationen ist zwischen verschiedenen Möglichkeiten zu unterscheiden. Das Annotieren des ersten Punktes einer Klasse dauerte im Median 2,4 Sekunden [BRFFF16]. Die Erstellung von Punktannotationen mit genau einem annotierten Pixel pro Klasse dauert folglich $18{,}5 \cdot 1\,\mathrm{s} + 1{,}5 \cdot 2{,}4\,\mathrm{s} = 22{,}1\,\mathrm{s}$ pro Bild. Für die Annotation weiterer Pixel derselben Klasse wurden im Median 0,9 Sekunden benötigt. Bei Punktannotationen mit einem annotierten Pixel pro vorhandener Objektinstanz ergibt sich ein Annotationsaufwand von $18{,}5 \cdot 1\,\mathrm{s} + 1{,}5 \cdot 2{,}4\,\mathrm{s} + (2{,}8 - 1{,}5) \cdot 0{,}9\,\mathrm{s} = 23{,}3\,\mathrm{s}$ pro Bild. In Rahmen der Arbeit von Bearman et al. [BRFFF16] wurden zudem Krakelannotationen erstellt. Das zeichnen eines Krakels dauerte 10,9 Sekunden. Die Annotation eines Bildes mit jeweils einem Krakel pro vorhandener Klasse dauerte im Durchschnitt $18{,}5 \cdot 1\,\mathrm{s} + 1{,}5 \cdot 10{,}9\,\mathrm{s} = 34{,}9\,\mathrm{s}$ pro Bild. Bei Bounding-Box-Annotationen muss um jede Objektinstanz ein Rechteck gezeichnet werden. In [RLFF15] lag der durchschnittliche Aufwand pro Bounding-Box bei 10,2 Sekunden. Eine zur Punktannotation mit einem Punkt pro Objektinstanz analoge Berechnung des Aufwands führt zu einem erwarteten Aufwand von $18{,}5 \cdot 1\,\mathrm{s} + 2{,}8 \cdot 10{,}2\,\mathrm{s} = 47{,}06\,\mathrm{s}$ pro Bild des Pascal VOC Datensatzes. Die Annotation von nur einer Bounding-Box pro vorhandener Klasse würde den Aufwand auf $18{,}5 \cdot 1\,\mathrm{s} + 1{,}5 \cdot 10{,}2\,\mathrm{s} = 33{,}8\,\mathrm{s}$ pro Bild verringern. Ein zusammenfassender Vergleich der Annotationsaufwände der verschieden starken Formen der Annotationen ist in Tabelle 3.2.1 gegeben.

Tabelle 3.2.1: Vergleich des Annotationsaufwands bei verschiedenen starken Formen der Annotation.

Level	Annotationsaufwand
vollst. Annotation	239,7 s
Bild-Level-Annot.	20,0 s
Punktannotation (1 Punkt/Klasse)	22,1 s
Punktannotation (1 Punkt/Instanz)	23,3 s
Krakel (1 Krakel/Klasse)	34,9 s
Bounding-Box (1 Box/Klasse)	33,8 s
Bounding-Box (1 Box/Instanz)	47,1 s

3.2.3 *Verwendung von zusätzlichen Informationen*

Um die fehlenden Informationen bei Verwendung schwacher Annotationen im Training von Modellen zur semantischen Segmentierung auszugleichen, werden bei manchen Ansätzen zusätzliche Informationen bzw. *Vorwissen* in den Trainingsprozess einbezogen [HKH17]. Insbesondere bei Bild-Level- und Punktannotationen fehlen Informationen bezüglich der Ausdehnung der abgebildeten Objekte. Beispielsweise kann zusätzlich Vorwissen über die Größe von Objekten verwendet werden (z. B. [PKD15]). Eine andere Form zusätzlicher Information ist die *Objectness*, auch *Saliency* genannt [HKH17]. Die Objectness ist ein reeller Wert, der einem Pixel oder einem Ausschnitt eines Bildes zugeordnet ist, und angibt, ob dieser Bereich zu einem Objekt gehört. Diese Angabe ist *unabhängig von der Klasse* des Objektes. In Folge dessen werden größere Bereiche der Objekte berücksichtigt, als bei klassenabhängigen Informationen, welche häufig nur diskriminative Teile der Objekte berücksichtigen.

Eine Möglichkeit die Objectness in einem Ausschnitt eines Bildes zu berechnen wurde von Alexe et al. [ADF12] beschrieben. Die Objectness eines Ausschnitts wird anhand mehrerer Kriterien bewertet. *Multiscale Saliency* [HZ07] bevorzugt Bereiche im Bild, deren Erscheinung sich vom Rest des Bildes unterscheidet. Ein weiteres Kriterium ist der *Color Contrast* [ADF12]. Dieses Kriterium vergleicht das Fenster anhand dessen LAB-Farbhistogramms mit einem Umfeld, welches durch Vergrößerung des Fensters entsteht. Somit soll der Unterschied in der Farbverteilung des Objektes zur Farbverteilung des Hintergrunds berücksichtigt werden. Die *Edge Density* [ADF12] bewertet die Dichte von Kanten nahe dem Rand des Fensters. Am Rand eines Objektes sollte die Dichte von Kanten hoch sein, innerhalb des Objektes niedrig. Ein Kriterium, welches

ebenfalls den geschlossenen Rand eines Objektes bewertet, ist das *Superpixel Straddeling* [ADF12]. Superpixel unterteilen ein Bild in Bereiche homogener Erscheinung bezüglich Farbe oder Textur, wobei Grenzen zwischen Objekten erhalten bleiben. Das Kriterium bewertet, wie viele Superpixel zum Teil innerhalb und zum Teil außerhalb des Fensters liegen. Da Superpixel die Grenzen von Objekten berücksichtigen und ein einzelnes Superpixel folglich innerhalb oder außerhalb eines Objektes liegt, sollte der Rand eines Fensters durch möglichst wenige Superpixel verlaufen. Ein fünftes Kriterium [ADF12] berücksichtigt die Wahrscheinlichkeit von Position und Größe eines Fensters unabhängig vom Inhalt des Bildes. Die Parameter zur Berechnung der fünf Kriterien wurden auf 1183 Bildern aus den Trainings- und Validierungsdaten des Pascal VOC 2007 Datensatzes [EVGW+07] trainiert. Es wurden nur Bilder mit Objekten der Klassen Vogel, Auto, Katze, Kuh, Hund und Schaf verwendet. Die Evaluation wurde auf 2941 Bildern aus dem Pascal VOC 2007 Datensatz, welche nicht im Training verwendet wurden, unter Berücksichtigung von Objektinstanzen aller 20 Klassen durchgeführt. Bewertet wurde das Verhältnis von Detektionsrate zur Anzahl der ausgegebenen Vorschläge für Bildausschnitte. Die fünf Kriterien wurden zunächst einzeln evaluiert, wobei das Superpixel Straddeling die besten Ergebnisse lieferte. Zudem wurden jeweils zwei oder drei Kriterien in einem Bayes'schen Framework miteinander kombiniert. Die besten Ergebnisse wurden bei der Kombination der Kriterien Superpixel Straddeling, Multiscale Saliency und Color Contrast erreicht.

3.2.4 *Training mit schwachen Annotationen*

Für das Training von Faltungsnetzen zur semantischen Segmentierung werden, wie in Abschnitt 2.1 beschrieben, Klassenlabels für jedes Pixel benötigt. Pathak et al. [PSLD15] stellen einen Ansatz zum Training tiefer Netze zur semantischen Segmentierung mit Bild-Level-Annotationen vor. Die Schwierigkeit liegt darin, ein pixelweises Gütekriterium zu berechnen, unter ausschließlicher Verwendung von Informationen zu An- und Abwesenheit von Klassen für das gesamte Bild. Pathak et al. verwenden als Ansatz *multi-class multiple instance learning* (multi-class MIL). Anstelle eines gegebenen Labels pro Pixel, werden die Pixel eines Bildes zu einer Pixelmenge zusammengefasst, für die Klassenlabels gegeben sind. Als Architektur wird das VGG-16 [SZ14] (vgl. Abschnitt 2.5) verwendet und mittels Convolutionalization (vgl. Abschnitt 3.1) zu einem Fully Convolutional Network [LSD15] umgewandelt, um für strukturierte räumliche Vorhersagen verwendet werden zu können. Der Loss wird an den Pixeln berechnet, an denen die Wahrscheinlichkeit für die im Bild vorhandenen Klassen maximal ist, und von diesen Pixeln der Ausgabe ausgehend mittels des Backpropagation-Algorithmus

zur Korrektur der Gewichte verwendet. Der MIL-Loss für ein Bild als Pixelmenge $\mathcal{I}$ und der Menge der im Bild vorhandenen Klassen $\mathcal{L}_\mathcal{I}$ wird wie folgt berechnet

$$L_{MIL}(\hat{\mathbf{y}}, \mathcal{L}_\mathcal{I}) = \frac{-1}{|\mathcal{L}_\mathcal{I}|} \sum_{l \in \mathcal{L}_\mathcal{I}} \log \hat{\mathbf{y}}_{t_l,l} \quad \text{mit } t_l = \underset{i \in \mathcal{I}}{\operatorname{argmax}}\, \hat{\mathbf{y}}_{il}, \tag{3.2.1}$$

wobei die Ausgabe der Softmax-Aktivierungsfunktion $\hat{\mathbf{y}}_{ic}$ die Wahrscheinlichkeit der Klasse c an Pixel i angibt. Das Netz wurde mit vortrainierten Gewichten des VGG-16 auf dem ILSVRC Datensatz [DDS+09] initialisiert. Die Gewichte der letzten Schicht wurden von den Klassen aus dem ILSVRC Datensatz übernommen, die mit denen des Pascal VOC 2011 Datensatzes [EVGW+11] übereinstimmen. Nach Umwandlung zu einem Fully Convolutional Network wurde das Modell mit dem MIL-Loss, einer Lernrate von $\alpha = 0{,}0001$, einem Momentum von $\beta = 0{,}9$ und einem Weight-Decay von $\lambda = 0{,}0005$ auf den Trainings- und Validierungsdaten des Pascal VOC 2011 Datensatzes [EVGW+11] mit zusätzlichen Augmentierungen von Hariharan et al. [HAB+11] für die semantische Segmentierung trainiert und validiert. Mit einer MIoU von 25,05 % im Vergleich zu 13,11 % auf dem Pascal VOC Validierungsdatensatz von 2011 und 25,66 % im Vergleich zu 13,09 % auf den Pascal VOC Testdaten von 2012 wurde durch das Nachtrainieren mit dem MIL-Loss eine relative Verbesserung von 96 % im Vergleich zum vortrainierten VGG-16 erreicht.

Eine andere Vorgehensweise zum Training mit schwachen Annotationen wurde von Oh et al. vorgestellt [OBK+17]. Mit Hilfe eines trainierten Moduls, dem *Guide Labeller* (vgl. Abbildung 3.2.2), wird aus einem Bild mit gegebenen Bild-Level-Annotationen eine vollständige Annotation als eine Abschätzung der tatsächlichen pixelweisen Annotation berechnet. Im Gegensatz zum zuvor beschriebenen Ansatz von Pathak et al. [PSLD15] kann das Training wie bei einer gegebenen vollständigen Annotation durchgeführt werden, ohne Änderungen am Modell oder am Trainingsprozess vorzunehmen. Der Guide-Labeller besteht aus drei Komponenten. Der *Seeder* wird mit dem Ziel trainiert, die Ortsvorschläge für die im Bild vorhandenen Klassen zu berechnen. Das *Saliency-Modul* soll eine klassenunabhängige Salienzkarte ausgeben, um eine Abschätzung der Ausdehnung der vorhandenen Objekte zu erhalten. Ein drittes Modul führt die Ausgaben des Seeders und des Saliency-Moduls zusammen. Den Bereichen der Salienzkarte, deren Wert größer als ein festgelegter Schwellenwert ist, werden entsprechend der Klassen, welche nach Ausgabe des Seeders im jeweiligen Bereich liegen, mit verschiedenen Strategien die im Bild vorhandenen Vordergrundklassen zugeordnet. Das Resultat dieser Verschmelzung ist eine pixelweise Zuordnung der Klassen als Abschätzung der tatsächlichen vollständigen Annotation. Für das Training werden Bild-Level-Annotationen des augmentierten Pascal VOC

2012 Datensatzes [EVGW+12, HAB+11] für das Seeder-Modul und ein Teil der klassenunabhängigen Bounding-Box Annotationen des MSRA Datensatzes [LYS+11] für das Saliency-Modul verwendet. Als Netz zur semantischen Segmentierung wurde das DeepLab-LargeFOV [CPK+16] mit den durch den Guide-Labeller berechneten Annotationen trainiert. In der besten Kombination möglicher Variationen der drei Module wurde eine MIoU von 55,7 % auf dem Pascal VOC 2012 Validierungsdatensatz und von 56,7 % auf den Pascal VOC 2012 Testdaten erzielt, was 80,6 % der MIoU entspricht, welche das DeepLab-LargeFOV nach einem Training mit vollständigen Annotationen erreicht.

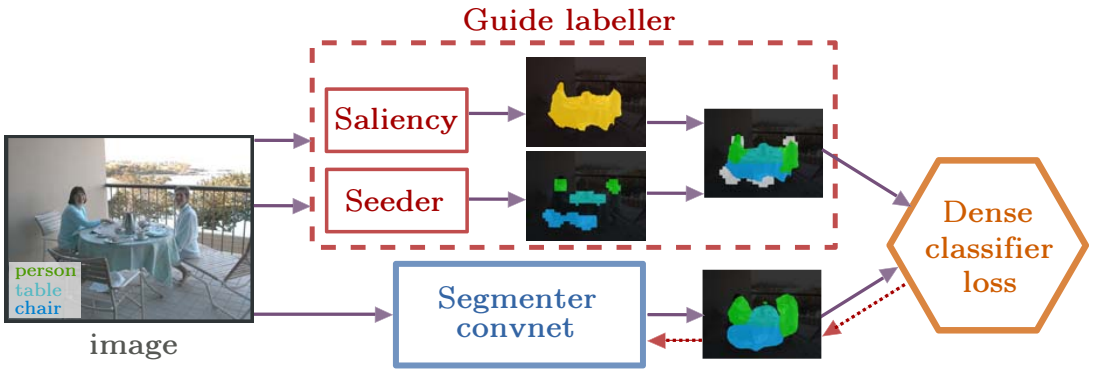


Abbildung 3.2.2: Aufbau des Guide-Labellers, welcher Abschätzungen der tatsächlichen pixelweisen Annotation berechnet, welche zum Training von Modellen zur semantischen Segmentierung verwendet werden können. Abbildung entnommen aus [OBK+17].

4 METHODIK

In den vorherigen Kapiteln wurden die Grundlagen zu tiefen Faltungsnetzen erläutert, Beispiele für deren Verwendung zur Klassifikation beschrieben und mögliche Anpassungen für die semantische Segmentierung diskutiert. In diesem Kapitel werden in Abschnitt 4.1 Vor- und Nachteile der im vorherigen Kapitel beschriebenen Architekturen für das Training mit schwachen Annotationen diskutiert. Anschließend wird in Abschnitt 4.2 erläutert, mit welcher Methode diese Architekturen mit schwachen Annotationen trainiert werden.

4.1 ARCHITEKTUREN

Für die semantische Segmentierung werden häufig tiefe Faltungsnetze zur Klassifikation mit angepasster Architektur verwendet [GGOEO+17], welche in Abschnitt 3.1 beschrieben wurden. Die Fully Convolutional Networks [LSD15] basieren auf dem VGG-16 [SZ14]. Das FCN-32s wurde nur leicht verändert, um dichte Klassifikationskarten berechnen zu können, und ist im Detailgrad der berechneten Segmentierung durch ein einmaliges Upsampling um den Faktor 32 beschränkt. Das FCN-16s und das FCN-8s haben zusätzliche Verbindungen, um die grob aufgelöste Ausgabe mit feiner aufgelösten Feature-Maps früherer Schichten zu kombinieren. In ihren Experimenten erreichten Long et al. [LSD15] eine Steigerung der Mean Intersection over Union (MIoU) von 59,4 % MIoU beim FCN-32s zu 62,4 % MIoU beim FCN-16s und 62,7 % MIoU beim FCN-8s. Für das Training des FCN-16s wird zuerst das FCN-32s trainiert und dessen Gewichte zur Initialisierung des FCN-16s verwendet. Nach dem Training des FCN-16s werden dessen Gewichte wiederum zur Initialisierung der Gewichte des FCN-8s verwendet. Da in dieser Arbeit der Einfluss von schwachen Annotationen untersucht werden soll, kann dieses Vorgehen jedoch zu Problemen führen. Das Ziel der Initialisierung der Gewichte ist, Vorwissen in Form vortrainierter Gewichte zu verwenden, um z. B. Problemen durch fehlende Informationen bei zu kleinen Datensätzen entgegenzuwirken. Durch das Training mit schwachen Annotationen können die Gewichte des FCN-32s aufgrund fehlender Informationen in den Annotationen nicht optimal trainiert werden und sind somit ggf. für eine Initialisierung komplexerer Modelle ungeeignet. Die Absicht des weiteren Trainings ist eine Verbesserung der De-

taillierung der berechneten Segmentierung. Jedoch sind beim Training mit schwachen Annotationen keine oder nur wenig detaillierte Informationen vorhanden. Aus diesen Gründen werden das FCN-16s und das FCN-8s im Folgenden nicht betrachtet. Zudem werden durch den Verzicht auf ein mehrstufiges Training eine mögliche Varianz und zusätzliche Fehlerquellen bei den verfeinernden Trainingsschritten vermieden.

Badrinarayanan et al. [BKC17] führten zum Vergleich des SegNets mit anderen Architekturen Experimente zur Segmentierung von Straßenszenen und Bildern von Wohnräumen durch. Das SegNet erreichte bei beiden Vergleichen eine deutliche bessere MIoU als die Fully Convolutional Networks. Bei der Segmentierung der Straßenszenen erreicht das SegNet zudem eine deutlich bessere MIoU als das DeepLab-LargeFOV und das DeepLab-CRF-LargeFOV. Nur bei der Segmentierung der Bilder von Innenräumen war die MIoU des DeepLab-LargeFOV besser als die des SegNets.

Das FCN-32s kann mit Gewichten eines VGG-16, welches auf dem ILSVRC Datensatz [DDS+09] trainiert wurde, initialisiert werden [LSD15, BRFFF16, PSLD15]. Für das Training mit vollständigen Annotationen werden von Long et al. [LSD15] die Gewichte der letzten Schicht, welche der Berechnung der Scores für die Klassen dient, mit Nullen initialisiert. Beim Training mit schwachen Annotationen ist es besser, ebenfalls die vortrainierten Gewichte der Neuronen zu übernehmen, welche die Scores von Klassen berechnen, die in beiden Datensätzen vorkommen [PSLD15]. Diese Initialisierung soll verhindern, dass die Modelle beim Training zur ausschließlichen Vorhersage der Hintergrundklasse konvergieren. Aufgrund des tiefen Encoder-Decoder Aufbaus des SegNet ist jedoch nur eine Initialisierung der Gewichte des Encoders möglich. Da für das Training mit schwachen Annotationen eine Initialisierung der Gewichte aller Schichten mit vortrainierten Gewichten sinnvoll ist, wird das SegNet trotz der guten Segmentierungsleistung in den Experimenten von Badrinarayanan et al. [BKC17] in dieser Arbeit nicht verwendet.

Das DeepLab-LargeFOV und das DeepLab-CRF-LargeFOV erreichen in den Experimenten von Badrinarayanan et al. [BKC17] ebenfalls bessere Ergebnisse als die Fully Convolutional Networks. Durch die Veränderung der Filtergröße im Vergleich zum VGG-16, dessen vollständig verbundene Schichten in Faltungsschichten umgewandelt wurden, ist auch bei dem DeepLab-LargeFOV nur eine teilweise Initialisierung der Gewichte mit vortrainierten Gewichten möglich. Aus diesem Grund werden auch die verschiedenen Varianten des DeepLabs nicht in dieser Arbeit verwendet.

Bei dem Dilated Network werden von Yu et al. [YK15] keine Filtergrößen verändert. Dies ermöglicht eine Initialisierung der Gewichte aller Schichten, wie sie von Pathak et al. [PSLD15] und Bearman et al. [BRFFF16] für das FCN-32s durchgeführt wurde. Durch die Verwendung von Dilated Convolutions ist der Aufbau ähnlich zum DeepLab-LargeFOV und die Auflösung der Feature-Maps der späteren Schichten ist höher als

beim FCN-32s. Bei der Evaluation auf den Pascal VOC 2012 Testdaten erreicht das DNet mit 67,6 % MIoU ein besseres Ergebnis als das FCN-8s mit 62,2 % MIoU und das DeepLab mit dem fully connected conditional random field mit 66,4 % MIoU, trotz des einfacheren Aufbaus des DNets [YK15]. Somit ist zu erwarten, dass das Modell nach dem Training mit vollständigen Annotationen auch besser als das FCN-32s ist.

Aus diesen Gründen werden das DNet und das FCN-32s in dieser Arbeit neben dem Training mit vollständigen Annotationen auch mit schwachen Annotationen trainiert und evaluiert. Die Qualität der berechneten Segmentierungen des FCN-32s und des DNets werden untersucht und mit den Ergebnissen von Bearman et al. [BRFFF16] verglichen.

4.2 TRAINING MIT SCHWACHEN ANNOTATIONEN

Die in Abschnitt 3.1 vorgestellten Modelle benötigen für ein Training mit einer pixelweisen Berechnung der Loss-Funktion vollständige Annotationen. Die Erstellung dieser Annotationen ist sehr aufwändig (vgl. Abschnitt 3.2.2), weshalb in dieser Arbeit Methoden zum Training dieser Netze mit schwachen Annotationen untersucht werden. Das Vorgehen dazu basiert auf der Arbeit von Bearman et al. [BRFFF16]. Häufig verwendete schwache Annotationen sind Bild-Level-Annotationen, welche für Datensätze zur Klassifikation (z. B. ImageNet [DDS+09]) in großem Umfang verfügbar sind. Neben dem Training mit Bild-Level-Annotationen werden in dieser Arbeit zudem Punktannotationen verwendet. Zusätzlich zu der Information über das Vorhandensein eines Objektes, geben Punktannotationen räumliche Informationen über die Position des Objektes auf einem Bild. Punktannotation sind zudem eine natürliche Form der Annotation, da sie das Zeigen eines Menschen auf ein Objekt repräsentieren [BRFFF16]. Als zusätzliche Information wird die sogenannte Objectness verwendet. Die Objectness ist ein heuristischer Ansatz zur Berechnung einer Wahrscheinlichkeit einer Vordergrundklasse, gegeben ein Pixel. Im Gegensatz zur Arbeit von Oh et al. [OBK+17] wird in dieser Arbeit kein zusätzliches System zur Berechnung einer Approximation der pixelweisen Annotation erstellt, sondern die Loss-Funktion für das Training der Modelle für die Verwendung schwacher Annotationen angepasst. Neben der Anpassung der Loss-Funktionen werden keine Änderungen an den Modellen oder der Vorverarbeitung der Eingabebilder vorgenommen.

Bei Verwendung vollständiger Annotationen wird in Arbeiten wie z. B. von Long et al. [LSD15] als Loss-Funktion die Summe über die pixelweise Kreuzentropie berechnet. Zur Vereinfachung wird im Folgenden das Bild als Pixelmenge $\mathcal{I}$ repräsentiert und die einzelnen Pixel mit i indiziert. Die Ausgabe der Softmax-Aktivierungsfunktion $\hat{\mathbf{y}}_{ic}$ ist

die Wahrscheinlichkeit der Klasse c an Pixel i. Die Annotation $\mathbf{y}$ enthält für jedes Pixel i den Index der korrekten Klasse. Die Summe über die pixelweise Kreuzentropie wird wie folgt berechnet:

$$L_{pix}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i \in \mathcal{I}} \log\left(\hat{\mathbf{y}}_{i y_i}\right). \tag{4.2.1}$$

Die Loss-Funktion, welche zum Training mit Bild-Level-Annotationen verwendet wird, basiert auf dem MIL-Loss von Pathak et al. [PSLD15] und wurde um einen Term erweitert, der die Abwesenheit von Klassen berücksichtigt [PKD15]. Bei Bild-Level-Annotationen sind ausschließlich Informationen zu An- bzw. Abwesenheit von Klassen für jedes Bild gegeben. Die Menge $\mathcal{L}$ enthält die Indizes der im Bild $\mathcal{I}$ vorhandenen Klassen, die Menge $\mathcal{L}'$ die Indizes der im Bild nicht vorhandenen Klassen. Die Loss-Funktion wird wie folgt berechnet [BRFFF16]:

$$L_{img}(\mathcal{L}, \mathcal{L}', \hat{\mathbf{y}}) = -\frac{1}{|\mathcal{L}|}\sum_{c \in \mathcal{L}} \log\left(\hat{\mathbf{y}}_{t_c c}\right) - \frac{1}{|\mathcal{L}'|}\sum_{c \in \mathcal{L}'} \log\left(1 - \hat{\mathbf{y}}_{t_c c}\right) \quad \text{mit } t_c = \underset{i \in \mathcal{I}}{\operatorname{argmax}}\, \hat{\mathbf{y}}_{ic}. \tag{4.2.2}$$

Die Beträge $|\mathcal{L}|$ und $|\mathcal{L}'|$ geben die Anzahl der Elemente der jeweiligen Menge an. Der erste Teil der Loss-Funktion entspricht dem MIL-Loss [PSLD15] und berücksichtigt die im Bild vorhandenen Klassen. Für jede dieser Klassen soll mindestens ein Pixel eine möglichst hohe Wahrscheinlichkeit haben. Der zweite Term der Loss-Funktion basiert auf einer Nebenbedingung von Pathak et al. [PKD15]. Für Klassen, welche nicht im Bild enthalten sind, soll mit diesem Term bewirkt werden, dass kein Pixel eine hohe Wahrscheinlichkeit hat.

Für die Verwendung von Punktannotationen wird ebenfalls eine Loss-Funktion definiert. Zusätzlich zu den Informationen zu An- und Abwesenheit der Klassen enthalten Punktannotationen Informationen über die Lokalisierung von Objektinstanzen der vorhandenen Klassen auf dem Bild. Diese Information entspricht annotierten Pixeln mit dem Unterschied zu einer vollständigen Annotation, dass nur eine Teilmenge $\mathcal{I}_s$ aller Pixel des Bildes annotiert ist. Alle nicht in $\mathcal{I}_s$ enthaltenen Pixel gehören zu einer Klasse in $\mathcal{L}$. Die Loss-Funktion für die Verwendung von Punktannotationen [BRFFF16] wird berechnet durch:

$$L_{point}(\mathcal{L}, \mathcal{L}', \mathbf{y}, \hat{\mathbf{y}}) = L_{img}(\mathcal{L}, \mathcal{L}', \hat{\mathbf{y}}) - \sum_{i \in \mathcal{I}_s} \alpha_i \log\left(\hat{\mathbf{y}}_{i y_i}\right) \tag{4.2.3}$$

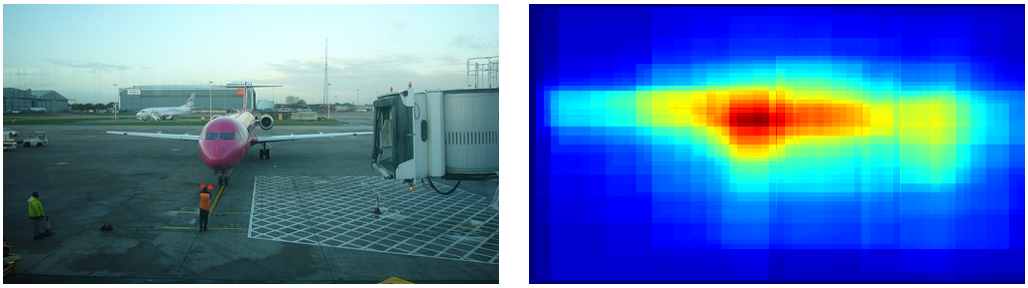
Die Informationen zu An- bzw. Abwesenheit von Klassen wird mittels der Loss-Funktion in Formel 4.2.2 berücksichtigt. Dazu wird die Summe über die pixelweise

Kreuzentropie der annotierten Pixel subtrahiert. Der Gewichtungsfaktor α_i in der Summe kann unterschiedlich gewählt werden, abhängig von den gegebenen Annotationen [BRFFF16]. Bei einem annotierten Pixel pro vorhandener Klasse wird dasselbe α_i für jedes Pixel verwendet. Werden die Annotationen durch mehrere Personen für dasselbe Bild erstellt, kann über den Faktor α_i die Unsicherheit der Annotation gewichtet werden. Ein anderer Fall, der zu mehreren annotierten Pixeln pro Klasse führen kann, ist die Annotation jeder Objektinstanz der entsprechenden Klasse. In diesem Fall kann durch α_i die Reihenfolge der Annotation der einzelnen Pixel (falls bekannt) für die Gewichtung berücksichtigt werden.

Als zusätzliche Information kann die Objectness ebenfalls bei der Berechnung der Loss-Funktion Einfluss auf das Training nehmen. Die Objectness $\mathbf{p}_i$ ist gegeben als Wahrscheinlichkeit des Pixels i zu einer Vordergrundklasse zu gehören. Somit enthält die Objectness klassenunabhängige Informationen zur Ausdehnung von Vordergrundobjekten. Der Pascal VOC 2012 Datensatz [EVGW+12] hat beispielsweise 20 Vordergrundklassen und eine Hintergrundklasse. Der Loss wird anhand der Objectness $\mathbf{p}$ und der Menge der Vordergrundklassen $\mathcal{O}$ wie folgt berechnet [BRFFF16]:

$$L_{obj}(\mathbf{p}, \hat{\mathbf{y}}) = -\frac{1}{|\mathcal{I}|}\left[\sum_{i\in\mathcal{I}} \mathbf{p}_i \log\left(\sum_{c\in\mathcal{O}} \hat{\mathbf{y}}_{ic}\right) + (1-\mathbf{p}_i)\log\left(1-\sum_{c\in\mathcal{O}} \hat{\mathbf{y}}_{ic}\right)\right] \tag{4.2.4}$$

Abbildung 4.2.1 zeigt ein Beispiel für eine Objectness-Karte, dargestellt als Heatmap. Die Pixel des Flugzeugs haben hohe Objectnesswerte (rot markiert). Der Wert der Loss-Funktion wird kleiner, je höher die vorhergesagte Wahrscheinlichkeit des Netzes für eine Vordergrundklasse ist. Für Pixel mit niedriger Objectness (blau markiert) sollen die vorhergesagten Wahrscheinlichkeiten für Vordergrundobjekte möglichst gering sein. Zur Berechnung der Loss-Funktion wird ausschließlich die Ausgabe des Netzes und die Objectness für alle Pixel des Bildes benötigt. In Folge dessen kann die Loss-Funktion mit jeder anderen Loss-Funktion unabhängig von der gegebenen Annotation durch Addition der Loss-Werte kombiniert werden.



(a) Eingabebild (b) Objectness-Heatmap

Abbildung 4.2.1: Beispiel für die Objectness eines Eingabebilds [EVGW+12], berechnet mit dem vortrainierten Modell von Alexe et al. [ADF12]. Die Wahrscheinlichkeiten sind als Heatmap dargestellt. Hohe Objectnesswerte, und somit eine hohe Wahrscheinlichkeit des Pixels zu einem Vordergrundobjekt zu gehören, sind rot dargestellt, niedrige Werte blau.

5 EXPERIMENTE

Im Rahmen dieser Arbeit werden das FCN-32s und das Dilated Network (DNet) auf verschieden starken Annotationen trainiert. Die Implementierung der Architekturen und des Trainingsprozesses wurden zu diesem Zweck mit Hilfe des Frameworks *PyTorch*[1] in Version 0.4.0 durchgeführt. Zum Training und zur Evaluierung wird ein Teil des Pascal VOC Datensatzes von 2012 [EVGW+12] verwendet und mit zusätzlichen Annotationen [HAB+11] erweitert. Details zum Datensatz und der Aufteilung der Daten für Training und Validierung werden in Abschnitt 5.1 beschrieben. Zudem werden die verwendeten Punktannotationen [BRFFF16] und die Berechnung der verwendeten Objectness-Karten [ADF12] erläutert. Der Aufbau des Trainings wird in Abschnitt 5.2 zuerst für die Verwendung vollständiger und schwacher Annotationen beschrieben und die dazu durchgeführte Vorverarbeitung der Trainingsdaten diskutiert. Das Vorgehen bei der Bewertung und die verwendete Metrik wird in Abschnitt 5.3 diskutiert. Die Ergebnisse werden in Abschnitt 5.4 präsentiert. Die Segmentierungsleistung des FCN-32s und des DNet nach dem Training mit vollständigen und mit schwachen Annotationen bewertet und verglichen.

5.1 DATENSATZ

Für die Experimente in dieser Arbeit wird der Pascal Visual Object Classes (VOC) [EVGW+10, EVGW+12] Datensatz verwendet. Die erste Version des Datensatzes von 2005 enthielt 1.578 Bilder (Training, Validierung und Test) zur Klassifikation und Detektion von vier Klassen. Die Bilder des Datensatzes wurden zuerst 2006 und nochmals 2007 ersetzt. Seit 2007 enthält der Datensatz Objekte 20 verschiedener Vordergrundklassen. Abbildung 5.1.1 zeigt die 20 Objektklassen als Unterteilung der Kategorien *Fahrzeuge*, *Haushalt*, *Tiere* und *Person*. Im Vergleich zu den Klassen der vorherigen Datensätze, wurden diese semantisch genauer unterteilt (z. B. *Fahrzeug* in *Auto* und *Motorrad*). Schwierigkeiten in der Unterscheidung der Klassen der Kategorie Tiere sollen durch ähnlich aussehende Klassen wie z. B. *Hund* und *Katze* erzeugt werden. Weitere Schwierigkeiten entstehen durch Klassen wie z. B. *Stuhl*, welche eher durch ihre Funktionalität als durch ihre Erscheinung definiert sind. Alle nicht zu den 20

1 https://pytorch.org/

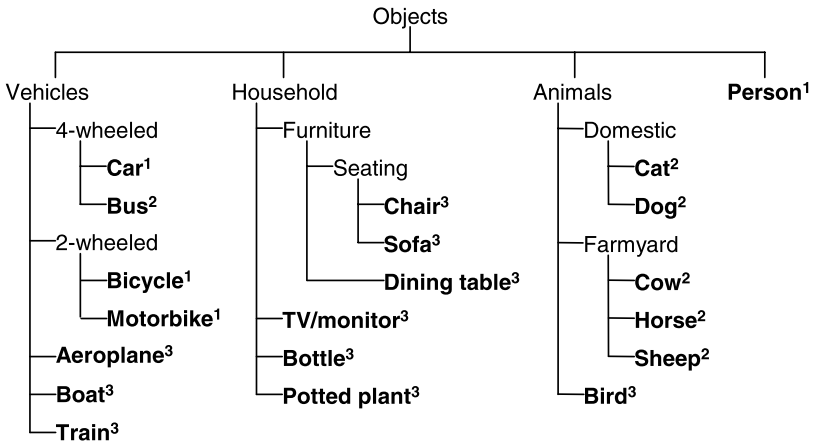


Abbildung 5.1.1: Klassen des Pascal VOC Datensatzes seit 2007. Klassen, welche in älteren Versionen des Pascal VOC Datensatzes verwendet wurden, sind durch Hochstellungen (2005^1, 2006^2, 2007^3) gekennzeichnet. Die Klassen sind als Unterteilung von vier Kategorien dargestellt. Neben der Verfeinerung der Kategorien wurden im Laufe der Jahre weitere Oberkategorien hinzugefügt um die Domäne des Datensatzes zu erweitern. Abbildung entnommen aus [EVGW+10].

Objektklassen gehörigen Pixel wurden als *Hintergrund* annotiert. 2007 wurde der Umfang, zusätzlich zu Datensätzen zur Klassifikation und Detektion, um Datensätze zu Personen Layouts[2] und zur Segmentierung erweitert. Die Bilder wurden auf der Plattform *flickr*[3] anhand von Stichworten automatisiert gesucht und heruntergeladen. Das Ziel der automatisierten Sammlung von Bildern für den Datensatz war, dass kein Bias zu bestimmten Aufnahmebedingungen und Motiven der Bilder durch manuelle Auswahl entsteht. Die Bilder auf flickr wurden aus unterschiedlichen Blickwinkeln bei verschiedenen Lichtverhältnissen aufgenommen. Zudem werden Objekte vor oder in Szenen gezeigt, statt diese z. B. einzeln vor sehr homogenem Hintergrund zu fotografieren. Mehrfach vorhandene Bilder, welche sich nur in Kompression, Verdeckungen oder Verzerrungen unterschieden, wurden aussortiert. Bilder, welche keine Instanzen der 20 Klassen enthielten oder deren Annotationen nicht mit ausreichender Sicherheit erstellt werden konnten, wurden ebenfalls verworfen. Die Annotation wurde anhand einer Richtlinie [WE07] durchgeführt. Bei der pixelweisen Annotation wurde um Objekte ein fünf Pixel breiter Rand zugelassen, welcher als *void* markiert werden durfte, und in der Berechnung der Loss-Funktion und der Bewertung der Ergebnisse nicht berück-

2 Ziel ist die Erkennung von Körperteilen. Diese werden mit Bounding-Boxes markiert und den Klassen Kopf, Hand und Fuß zugewiesen. [EVGW+12].

3 https://www.flickr.com/

sichtigt wird. Das void-Label wurde zudem für schwierig zu annotierende Objekte verwendet. Für 2.913 Bilder wurden pixelweise Annotationen erstellt. Die Menge der annotierten Bilder wurde in einen Trainingsdatensatz mit 1.464 Bildern und einen Validierungsdatensatz mit 1.449 Bildern unterteilt und veröffentlicht [EVGW+12]. Der Testdatensatz wurde nicht veröffentlicht. Die Annotationen wurden mit dem *Semantic Boundaries Dataset* (SBD) von Hariharan et al. [HAB+11] erweitert, wodurch für 12.031 Bilder vollständige Annotationen gegeben sind.

Für die Bewertung der Experimente in dieser Arbeit wird der Validierungsdatensatz des Pascal VOC Datensatzes mit 1.449 Bildern verwendet. Das Training wird mit 10.582 Bildern durchgeführt, zusammengesetzt aus den 1.464 Trainingsbildern des Pascal VOC Datensatzes und 9.118 Bildern des SBD, welche nicht im Pascal VOC Validierungsdatensatz enthalten sind. Neben den vollständigen Annotationen werden in dieser Arbeit Bild-Level-Annotationen, Punktannotationen und Objectness verwendet. Die Punktannotationen wurden mittels *Amazon Mechanical Turk* im Rahmen der Arbeit von Bearman et al. [BRFFF16] für den zuvor beschriebenen Teil der Datensätze [EVGW+12, HAB+11] erstellt. Die annotierenden Personen sollten zum einen die Instanz einer Klasse, welche sie als erste wahrnehmen, und jede Instanz der Klasse auf dem Bild mit einem Punkt markieren. Die Rate nicht annotierter, vorhandener Klassen liegt bei 1,0 % [BRFFF16]. Bei der Markierung der ersten Instanz wurden 7,2 % der Punkte auf ein Pixel einer anderen Klasse und 0,8 % der Punkte auf ein Pixel mit void-Label gesetzt. Bei der Markierung aller Instanzen wurden 7,9 % der Instanzen nicht markiert, 14,8 % der Punkte auf Pixel anderer Klassen und 1,6 % der Punkte auf Pixel mit void-Label gesetzt. In den Experimenten in dieser Arbeit werden nur die Punkte verwendet, welche die erste Instanz der Klasse markieren. Als Bild-Level-Annotationen wurden alle Klassen verwendet, die in der so erhaltenen Punktannotation auf einem Bild markiert wurden. 0,76 % der Pixel des zusammengesetzten Trainingsdatensatzes und 5,49 % der Pixel des Validierungsdatensatzes sind mit dem void-Label annotiert. Die prozentualen Anteile der Klassen an den berücksichtigten Pixeln (nicht void) sind in Tabelle 5.1.1 aufgeführt. Fast drei Viertel der Pixel des Datensatzes sind als *Background* annotiert.

Zur Berechnung der Objectness wird das trainierte Modell von Alexe et al. [ADF12] (vgl. Abschnitt 3.2.3) verwendet. Als Kriterien für die Objectness wird die in [ADF12] beste Kombination von drei Kriterien verwendet. Die Kriterien sind die Multiscale Saliency, der Color Contrast und das Superpixel Straddeling. Das Training wurde von Alexe et al. [ADF12] anhand von 291 Objektinstanzen aus 50 Bildern verschiedener Datensätze ohne Überschneidungen mit dem Pascal VOC Datensatz durchgeführt [BRFFF16]. Zu diesem Zweck wurden 291 Bounding-Box-Annotationen benötigt. Der Aufwand zur Erstellung der Annotationen wird mit 10,2 Sekunden pro Bounding-

Tabelle 5.1.1: Prozentuale Anteile der Klassen des Pascal VOC 2012 Datensatzes an den erweiterten Trainingsdaten [EVGW+12, HAB+11] (Train.) und den Validierungsdaten [EVGW+12] (Val.). Berücksichtigt werden nur Pixel, welche nicht als void markiert sind.

Klasse	Train.	Val.
Background	69,39 %	73,32 %
Aeroplane	0,92 %	0,79 %
Bicycle	0,83 %	0,33 %
Bird	0,89 %	0,88 %
Boat	0,67 %	0,55 %
Bottle	0,52 %	0,78 %
Bus	1,23 %	1,76 %
Car	2,00 %	1,47 %
Cat	3,27 %	2,34 %
Chair	1,32 %	0,86 %
Cow	0,56 %	1,21 %

Klasse	Train.	Val.
Diningtable	1,06 %	1,17 %
Dog	2,91 %	2,06 %
Horse	0,91 %	1,08 %
Motorbike	1,18 %	1,06 %
Person	7,78 %	5,31 %
Pottetplant	0,63 %	0,50 %
Sheep	0,61 %	0,79 %
Sofa	1,21 %	1,42 %
Train	1,33 %	1,60 %
TVMonitor	0,78 %	0,72 %

Box angenommen [RLFF15], was verteilt über die 10.582 Bilder des Datensatzes einen zusätzlichen Aufwand von 0,28 Sekunden zur Folge hat [BRFFF16]. Für jedes der 10.582 Trainingsbilder wird mit dem trainierten Modell die Objectness für 1.000 zufällig gewählte Fenster berechnet und jedem Pixel der Mittelwert der Objectness aller Fenster, welche das Pixel enthalten, zugewiesen.

5.2 AUFBAU DES TRAININGS

In dieser Arbeit werden das FCN-32s und das DNet für die semantische Segmentierung trainiert und evaluiert. Zuerst wird ein Training mit vollständigen Annotationen durchgeführt. Anschließend werden die Modelle mit verschiedenen schwachen Annotationen trainiert und der Einfluss auf die Segmentierungsleistung untersucht.

5.2.1 *Training mit vollständigen Annotationen*

Zunächst werden in dieser Arbeit das FCN-32s und das DNet mit vollständigen Annotationen trainiert. Das Training folgt dem Vorgehen von Bearman et al. [BRFFF16] bzw.

Yu et al. [YK15]. Das FCN-32s wird bis auf die letzte Schicht vor dem Upsampling mit vortrainierten Gewichten des VGG-16 [SZ14] initialisiert, welches zur Klassifikation auf dem ImageNet Datensatz [DDS+09] trainiert wurde. Die letzte Schicht vor dem Upsampling wird durch eine Faltungsschicht mit einem $1 \times 1 \times 21$ Filter ersetzt, um für jede der 21 Klassen des Pascal VOC Datensatzes eine Feature-Map mit Scores zu berechnen. Die Gewichte dieser Schicht werden mit Nullen initialisiert. Für die Berechnung des Upsamplings werden die Gewichte der Transpose Convolution mit Gewichten für ein bilineares Upsampling initialisiert, welche nicht trainiert werden. Als Loss-Funktion wird die Kreuzentropie berechnet. Das Training wird mittels stochastischem Gradientenabstieg mit einer Mini-Batch-Größe von 20 Bildern, einer festen Lernrate von $\alpha = 10^{-5}$, einem Momentum von $\beta = 0{,}9$ und einem Weight-Decay von $\lambda = 0{,}0005$ durchgeführt. Für die Bias-Gewichte wird die Lernrate verdoppelt und der Weight-Decay auf 0 gesetzt. Im VGG-16 wurde nach der ersten und zweiten vollständig verbundenen Schicht nach dem fünften Max-Pooling, genannt „fc6“ und „fc7“, eine Dropout-Schicht eingefügt [SZ14]. Analog zu dieser Position wird im FCN-32s jeweils eine *Spatial Dropout*-Schicht [TGJ+15] auf die Ausgaben der beiden Faltungschichten nach dem fünften Max-Pooling angewendet, welche den in Faltungschichten umgewandelten vollständig verbundenen Schichten „fc6“ und „fc7“ entsprechen. Auf die Ausgabe der letzten Faltungsschicht, welche die Scores der einzelnen Klassen berechnet, wird kein Dropout angewendet. In den Spatial-Dropout-Schichten werden mit einer Wahrscheinlichkeit von 50 % einzelne Feature-Maps der Ausgabe der vorherigen Schicht auf Null gesetzt. Alle Gewichte des FCN-32s, außer die der Transpose Convolution, werden 60.000 Iterationen lang mit den 10.582 Bildern der erweiterten [HAB+11] Trainingsdaten des Pascal VOC 2012 Datensatzes [EVGW+12] trainiert. Die Bilder werden in einem Vorverarbeitungsschritt auf eine einheitliche Größe von 500×500 Pixeln skaliert und vom Mittelwert befreit.

Die Gewichte des DNets werden, wie für das FCN-32s beschrieben, mit vortrainierten Gewichten des VGG-16 initialisiert und die letzte Faltungsschicht vor dem Upsampling durch eine Faltungsschicht mit einem $1 \times 1 \times 21$ Filter ersetzt. Der Loss wird mittels der Kreuzentropie berechnet und Training mittels stochastischem Gradientenabstieg mit einer Mini-Batch-Größe von 14 Bildern, einer festen Lernrate von $\alpha = 10^{-3}$, einem Momentum von $\beta = 0{,}9$ und einem Weight-Decay von $\lambda = 0{,}0005$ durchgeführt. Wie beim FCN-32s wird für die Bias-Gewichte die Lernrate verdoppelt und der Weight-Decay auf 0 gesetzt. Spatial Dropout wird beim Training ebenfalls mit einer Wahrscheinlichkeit von 50 % auf die Ausgaben derselben Schichten wie im FCN-32s verwendet. Das Training erfolgt auf denselben Trainingsdaten für ebenfalls 60.000 Iterationen. Die Vorverarbeitung ist beim Training des DNet jedoch umfangreicher. Die Bilder werden, wie von Yu et al. [YK15] vorgeschlagen, mittels Reflection-Padding

um einen Rand von 186 Pixeln an jeder Seite erweitert. Dieser Rand kompensiert das im Netz fehlende Zero-Padding. Liegen Höhe oder Breite des Bildes unter 500 Pixeln, wird die entsprechende Bilddimension mittels Reflection-Padding gleichmäßig an beiden gegenüberliegenden Rändern bis auf 500 Pixel erweitert. Anschließend wird ein 500×500 Pixel großer zentraler Ausschnitt aus dem Bild vom Mittelwert befreit und als Eingabe für das DNet verwendet. Die Annotation wird analog zum Eingabebild verarbeitet, wird aber anstelle des Reflection-Paddings mittels konstantem Padding mit dem void-Label erweitert. Da im Netz kein Zero-Padding verwendet wird, ist die Ausgabe durch die Schichten der Architektur im Vergleich zum Eingabebild um einen Rand von 186 Pixeln an allen vier Seiten verkleinert. Die Auflösung vor dem Upsampling ist zudem durch das dreifache Max-Pooling mit einem Stride von 2 um den Faktor $2^3 = 8$ verringert. Beim Training des DNet wird anstelle des Upsamplings mittels Transpose Convolution im Netz ein Downsampling der Annotation um den Faktor 8 zum Training durchgeführt. Von Yu et al. [YK15] wurden beim Training des DNets zudem als Augmentierung die Trainingsbilder nach dem Reflection-Padding zufällig mit einer Wahrscheinlichkeit von 50 % horizontal gespiegelt. Des Weiteren wurde anstelle des zentralen Ausschnitts ein zufälliger 500×500 Pixel großer Ausschnitt zum Training verwendet. Das Netz wird zum einen zur Vergleichbarkeit der Ergebnisse mit denen von Yu et al. [YK15] das DNet in dieser Arbeit mit den zuvor beschriebenen Augmentierungen trainiert. Um eine Vergleichbarkeit zum FCN-32s zu erreichen, wird zudem ein Training ohne die zuvor genannten Maßnahmen zur Augmentierung durchgeführt.

5.2.2 *Training mit schwachen Annotationen*

Neben dem Training mit vollständigen Annotationen werden beide Architekturen mit schwachen Annotationen trainiert und die Segmentierungsleistungen für die verschiedenen Annotationen verglichen. Das Training wird mit Bild-Level-Annotationen, Punktannotationen und der Kombination von Punktannotationen mit Objectness als zusätzliche Information durchgeführt. Anstelle der Kreuzentropie wird die Loss-Funktion aus Formel 4.2.2 für Bild-Level-Annotationen und die Loss-Funktion 4.2.3 für Punktannotationen verwendet. Für die Kombination von Punktannotationen und Objectness wird der Loss als die Summe der Loss-Funktion für Punktannotationen 4.2.3 und der Loss-Funktion 4.2.4 für die Objectness berechnet. Die Initialisierung des FCN-32s und des DNets wird ebenfalls mit vortrainierten Gewichten des VGG-16 durchgeführt. Im Gegensatz zu den Modellen, welche mit vollständigen Annotationen trainiert werden, werden beim Training mit schwachen Annotationen auch die Gewich-

te der Faltungsschicht, welche die Scores der Klassen berechnet, mit den vortrainierten Gewichten initialisiert. Zu diesem Zweck werden die Gewichte der Neuronen zur Berechnung der Scores der Klassen des ImageNet Datensatzes verwendet, welche den 20 Vordergrundklassen des Pascal VOC 2012 Datensatzes entsprechen. Die Gewichte für den Filter, welcher die Scores der Hintergrundklasse berechnet, werden mit Nullen initialisiert.

Das Training des FCN-32s mit schwachen Annotationen wird mit denselben Hyperparametern wie das Training mit vollständigen Annotationen ebenfalls über 60.000 Iterationen durchgeführt. Spatial Dropout wird wie beim Training des FCN-32s mit vollständigen Annotationen verwendet. Die Stauchung oder Streckung entlang einer oder beider Bilddimensionen wird bei der Vorverarbeitung der Punktannotationen und der Objectness der skalierten Eingabebilder des FCN-32s beachtet.

Da für das Training des DNets mit vollständigen Annotationen eine vergleichsweise große feste Lernrate von 10^{-3} verwendet wird, werden zudem kleinere feste Lernraten von 10^{-4} und 10^{-5} verwendet. Momentum, Weight-Decay und Mini-Batch-Größe werden wie beim Training mit vollständigen Annotationen gewählt. Spatial Dropout wird ebenfalls wie beim Training des DNets mit vollständigen Annotationen angewendet. Das Training wird mit und ohne die im vorherigen Abschnitt genannten Augmentierungen über 60.000 Iterationen durchgeführt. Die Vorverarbeitung der Trainingsbeispiele für das DNet wurde analog zum Training mit vollständigen Annotationen durchgeführt. Bei der Verarbeitung von Bild-Level-Annotationen wurden jedoch alle, auf dem gesamten Bild vorhandene, Klassen als Annotation verwendet. Da Bild-Level-Annotationen keine Positionsinformationen enthalten, kann nicht bestimmt werden, welche Klassen durch die Wahl des 500×500 Pixel Ausschnitts entfallen. Bei Punktannotationen wurden ebenfalls für den Teil der Loss-Funktion, der die Bild-Level-Annotationen berücksichtigt, alle vorhandenen Klassen des Bildes verwendet. Bei gegebenen Punktannotationen ist eine Lokalisierung der vorhandenen Klassen möglich. Die Beschränkung auf die Klassen, welche in dem gewählten Ausschnitt enthalten sind, hat jedoch zu einer Verschlechterung der Qualität der Segmentierung geführt. Ohne das Wählen des 500×500 Pixel großen zentralen Ausschnitts enthalten die 10.582 Bilder des erweiterten Trainingsdatensatzes im Durchschnitt 1,53 Punkte pro Bild. Bei der Einschränkung auf einen 500×500 Pixel großen zentralen Ausschnitt und Verkleinerung der Annotation um einen Rand von 186 Pixeln sinkt die durchschnittliche Anzahl der Punkte pro Bild auf 0,61. Folglich werden bei der Berechnung der Loss-Funktion für Bild-Level-Annotationen im Durchschnitt deutlich mehr Klassen als vorhanden angenommen, als im Ausschnitt der Annotation enthalten sind. Im zweiten Term der Loss-Funktion werden die Punkte als einzelne annotierten Pixel berücksichtigt. Diese wurden, wie beim Training mit vollständigen Annotationen, auf

die Pixel im gewählten Ausschnitt beschränkt. Bei Augmentierung durch horizontale Spiegelung des Bildes und zufällige Wahl des Bildausschnitts sinkt die Anzahl der durchschnittlichen Punkte pro Bild auf 0,27 (gemittelt über 10.000 Iterationen). Bei Verwendung der Objectness beim Training des DNets wird ebenfalls nur ein skalierter Ausschnitt der ursprünglichen Objectness-Karte verwendet. Analog zur pixelweisen Annotation wird auch die Objectness-Karte vor der Wahl des Ausschnitts mittels Padding vergrößert. Als Padding werden in den Experimenten Zero-Padding und Reflection-Padding untersucht.

5.3 BEWERTUNG DER TRAINIERTEN NETZE

Nach dem Training der Netze wird die Genauigkeit der Segmentierung auf dem Validierungsdaten des Pascal VOC 2012 Datensatzes [EVGW+12] bewertet. Der Validierungsdatensatz umfasst 1449 Bilder, welche nicht im Trainingsdatensatz enthalten sind. Die Standardmetrik zur Bewertung der Genauigkeit der Segmentierung ist die *Mean Intersection over Union* (MIoU) [GGOEO+17], welche auch in anderen Arbeiten z. B. [BRFFF16], [YK15] und [LSD15] verwendet wurde. Die MIoU ist der Mittelwert der *Intersection over Union* (IoU) der einzelnen Klassen. Berechnet wird die IoU zwischen der Ausgabe des Modells und der gegebenen pixelweisen Annotation als das Verhältnis von Pixeln, welche derselben Klasse wie in der Annotation zugewiesen wurden (Intersection), zur Anzahl aller Pixel, welche in der Annotation oder in der Ausgabe des Modells der Klasse zugewiesen wurden (Union). Die Anzahl der Pixel, welche zu Klasse i zugehören und als Klasse j klassifiziert werden, wird mit p_{ij} angegeben. Für jede Klasse i wird das Verhältnis von korrekt klassifizierten Pixeln p_{ii} (true positives) zur Summe von korrekt klassifizierten Pixeln p_{ii} und Pixeln, welche entweder nur in der Annotation (false negatives, p_{ij} mit $i \neq j$) oder nur in der Ausgabe des Modells (false positives, p_{ji} mit $i \neq j$) der Klasse i zugewiesen wurden, berechnet. Die Anzahl der jeweiligen Paare werden für alle 21 Klassen (20 Objektklassen und die Hintergrundklasse) berechnet, über alle Bilder des Validierungsdatensatzes akkumuliert und anschließend zur Berechnung der MIoU verwendet. Bei k Vordergrundklassen ($k = 20$ beim Pascal VOC 2012 Datensatz [EVGW+12]) und einer Hintergrundklasse mit Index 0 wird die MIoU wie folgt berechnet [GGOEO+17]:

$$\text{MIoU} = \frac{1}{k+1} \sum_{i=0}^{k} \frac{p_{ii}}{\sum_{j=0}^{k} p_{ij} + \sum_{j=0}^{k} p_{ji} - p_{ii}}. \tag{5.3.1}$$

Pixel, welche mit einem void-Label annotiert sind, werden in der Berechnung nicht berücksichtigt. Eine weitere Metrik zur Bewertung einer Klassifikationskarte ist die

Pixel Accuracy. Diese gibt das Verhältnis der Anzahl von korrekt klassifizierten Pixeln zur Gesamtanzahl der Pixel an und wird wie folgt berechnet [GGOEO+17]:

$$\mathrm{PA} = \frac{\sum_{i=0}^{k} p_{ii}}{\sum_{i=0}^{k} \sum_{j=0}^{k} p_{ij}}. \tag{5.3.2}$$

Aufgrund des hohen Anteils von Hintergrundpixeln (s. Tabelle 5.1.1) wird die Pixel Accuracy für den Pascal VOC Datensatz durch die Genauigkeit der Klassifikation der Hintergrundklasse dominiert. Bei der MIoU hingegen wird die IoU Klassenweise berechnet und über die 21 Klassen gemittelt. Somit ist der Einfluss jeder Klasse auf die MIoU unabhängig von der Klassenverteilung der Pixel. Aus diesem Grund wird in dieser Arbeit nur die MIoU zur Bewertung der berechneten Segmentierungen verwendet.

5.4 ERGEBNISSE

Nach dem Training über 60.000 Iterationen auf den Trainingsbildern des erweiterten Pascal VOC 2012 Datensatzes werden die 1.449 Bilder des Pascal VOC 2012 Validierungsdatensatzes mit den trainierten Modellen segmentiert und anschließend die Segmentierung anhand der Mean Intersection over Union bewertet. Diese Bewertung wurde alle 10.000 Iterationen während des Trainings durchgeführt, um den Trainingsfortschritt bewerten zu können. Im Folgenden werden zunächst in Abschnitt 5.4.1 die Modelle, welche auf vollständig annotierten Daten trainiert wurden, evaluiert und mit den Referenzergebnissen von Bearman et al. [BRFFF16] und Yu et al. [YK15] verglichen. Die im Training mit vollständigen Annotationen erreichten Werte der MIoU dienen als Referenzwert für den Vergleich der Modelle, welche mit schwach annotierten Bildern trainiert werden. Die Ergebnisse des Trainings mit schwachen Annotationen werden in Abschnitt 5.4.2 diskutiert.

5.4.1 *Training mit vollständigen Annotationen*

Die Ergebnisse nach 60.000 Iterationen Training auf den Pascal VOC 2012 Trainingsdaten sind in Tabelle 5.4.1 dargestellt. Mit dem von Bearman et al. [BRFFF16] trainierten FCN-32s wurde nach 50.000 Iterationen eine MIoU von 58,30 % erreicht. Die Implementierung des FCN-32s, die im Rahmen dieser Arbeit erstellt wurde, ist mit einer MIoU von 49,26 % nach 50.000 Iterationen und 50,13 % nach 60.000 Iterationen deutlich schlechter als die Referenzergebnisse. Mögliche Gründe für diese Abweichung sind Unterschiede im Training und in der Vorverarbeitung, wie z. B. die Normierung der

Tabelle 5.4.1: Ergebnisse (in % MIoU) auf dem Pascal VOC 2012 Validierungsdatensatz. Angegeben sind die Referenzergebnisse des FCN-32s, trainiert von Bearman et al. [BRFFF16], nach 50.000 Iterationen und des DNets, welches von Yu et al. [YK15] trainiert und nach 60.000 Iterationen auf dem Pascal VOC 2012 Testdatensatz evaluiert wurde. Für die in dieser Arbeit trainierten Modelle wurde alle 10.000 Iterationen die Segmentierungsleistung bewertet.

Iteration	10.000	20.000	30.000	40.000	50.000	60.000
FCN-32s [BRFFF16]					58,30 %	
FCN-32s	39,30 %	44,39 %	46,71 %	48,11 %	49,26 %	50,13 %
DNet (aug.) [YK15]						67,60 %
DNet (aug.)	58,23 %	60,09 %	60,89 %	60,73 %	60,71 %	59,61 %
DNet	55,14 %	56,59 %	56,15 %	56,22 %	55,70 %	55,05 %

Eingabe. Zudem konvergieren die Werte der MIoU im Vergleich zum DNet noch nicht nach der festgelegten Anzahl von Trainingsiterationen. Da die Referenzergebnisse bei 50.000 Iterationen gegeben sind, das DNet beim Training mit vollständigen Annotationen und beide Architekturen bei Training mit schwachen Annotationen bereits nach wenigen Zehntausend Iterationen konvergieren, wird auf ein Training des FCN-32s über mehr Iterationen verzichtet. Obwohl nach 60.000 Iterationen keine Verschlechterung der Segmentierungsleistung wie beim DNet festzustellen ist, wird mit dem FCN-32s eine Verbesserung von weniger als einem Prozentpunkt in den letzten 10.000 Iterationen des Trainings erreicht.

Für das DNet ist von Yu et. al [YK15] die erreichte MIoU nach 60.000 Iterationen des Trainings bei Verwendung von zufällig ausgewählten Ausschnitten, zufällig horizontal gespiegelter Eingabebilder[4] gegeben. Mit dem Modell wurde auf dem Testdatensatz, dessen Annotationen nicht öffentlich verfügbar sind, eine MIoU von 67,60 % erreicht. Auch mit der in dieser Arbeit erstellten Implementierung des DNets wird nach dem Training mit den gleichen Augmentierungen mit einer MIoU von 59,61 % eine deutlich schlechtere Segmentierungsleistung erreicht. Der Verlauf der MIoU über die Trainingsiterationen zeigt, dass das beste Ergebnis mit einer MIoU von 60,89 % nach 30.000 Iterationen erreicht wird. Die MIoU liegt über der des FCN-32s, welches jedoch ohne Augmentierung der Trainingsdaten trainiert wurde. Zur besseren Vergleichbarkeit zu den Ergebnissen des FCN-32s wurde das DNet ebenfalls ohne

4 Entnommen aus der öffentlich verfügbaren Referenzimplementierung: https://github.com/fyu/dilation.

Augmentierung trainiert. Die besten Ergebnisse werden bei diesem Training bereits nach 20.000 Iterationen mit einer MIoU von 56,59 % erreicht. Über den Trainingsverlauf ist das Modell, welches ohne Augmentierungen trainiert wurde, nach derselben Anzahl von Iterationen etwa vier Prozentpunkte schlechter als das Modell, welches mit augmentierten Eingabebildern trainiert wurde. Dies zeigt, dass bei Verwendung von vollständigen Annotationen die von Yu et al. [YK15] gewählten Augmentierungen zu einer deutlichen Verbesserung der Segmentierungsleistung führen. Mit einer MIoU von 56,59 %, als bestes Ergebnis während des Trainings, ist die Segmentierungsleistung schlechter als das FCN-32s von Bearman et al., jedoch etwa sechs Prozentpunkte besser als die Implementierung des FCN-32s in dieser Arbeit. Dies unterstützt die Aussage von Long et al. [LSD15], dass der Detailgrad der Ausgabe nach dem Upsampling mittels Transpose Convolution aufgrund des starken Subsamplings durch Pooling mit einem Stride von 2 beschränkt ist. Durch das Wegfallen der letzten beiden Max-Pooling-Schichten ist die Auflösung der Feature-Map vor der Transpose Convolution beim DNet höher als beim FCN-32s.

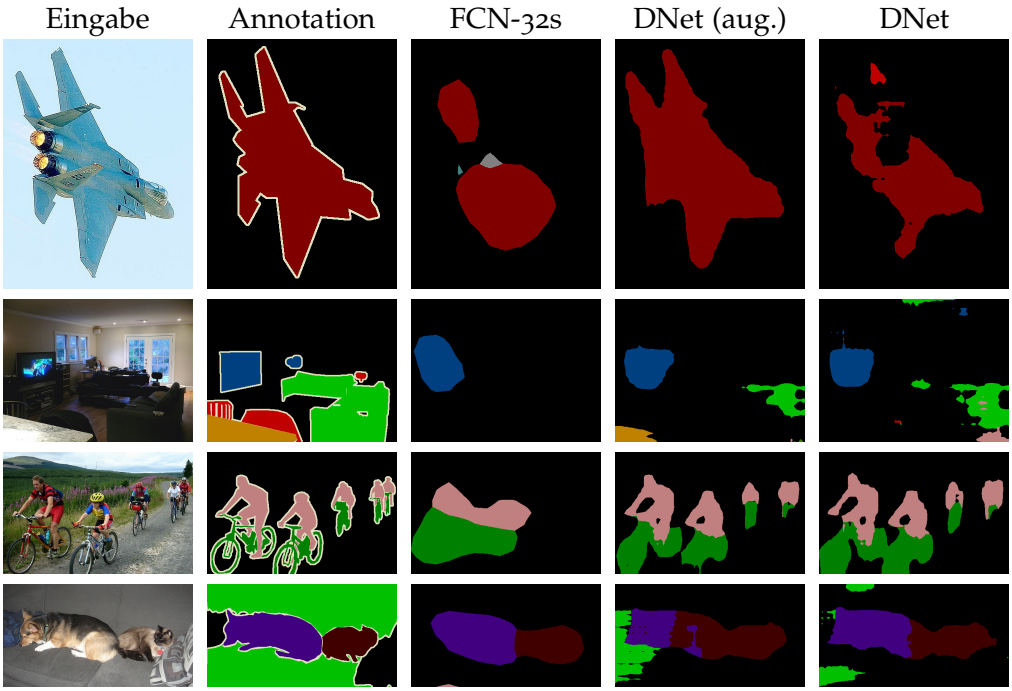


Abbildung 5.4.1: Qualitative Ergebnisse des FCN-32s und des DNet am Beispiel von vier Bildern aus dem Pascal VOC 2012 Validierungsdaten [EVGW+12]. Die abgebildeten Segmentierungen wurden jeweils mit dem besten Modell berechnet.

Abbildung 5.4.1 zeigt qualitative Ergebnisse der semantischen Segmentierung mit dem FCN-32s und dem DNet (mit und ohne Augmentierung trainiert) für vier verschiedene Bilder aus den Pascal VOC 2012 Validierungsdaten. Das erste Bild zeigt ein zentrales großes Objekt (Kampfjet, Klasse: *Aeroplane*) vor homogenem Hintergrund. Mit dem DNet, welches mit augmentierten Daten trainiert wurde, wird das, bei (subjektiver) visueller Beurteilung, beste Ergebnis erzielt. Das DNet, welches ohne Augmentierung trainiert wurde, zeigt etwas schlechtere Ergebnisse. Bei beiden Ausgaben lässt sich jedoch ein großer Teil der Kontur des Kampfjets erkennen. Im Gegensatz dazu steht die Ausgabe des FCN-32s. Diese zeigt keine Details und ist ein Beispiel für die Aussage von Long et al. [LSD15], dass der Detailgrad der Ausgabe dieses Modells beschränkt ist. Das zweite Bild stellt eine Szene in einem Wohnraum dar. Im Gegensatz zum ersten Bild sind mehrere kleine Objektinstanzen verschiedener Klassen vor einem inhomogenen Hintergrund abgebildet. In diesem Beispielbild werden von allen Modellen viele Objektinstanzen nicht korrekt segmentiert. Im Vergleich zum FCN-32s klassifiziert das DNet Pixel auch als *Diningtable*, *Sofa* und an wenigen Pixeln *Chair*, welches ebenfalls im Bild vorhandene Klassen sind. Das dritte Beispielbild zeigt erneut ein Beispiel für den fehlenden Detailgrad des FCN-32s. Die Segmentierung ist deutlich weniger detailliert als die, welche mit dem DNet berechnet wurde. Bei beiden Modellen des DNets sind auch Details wie die Reifen der Fahrräder oder die Arme der Fahrradfahrer erkennbar. Das vierte Bild zeigt einen Hund und eine Katze auf einem Sofa. Die Konturen im Bild sind weniger detailliert als z. B. die Konturen im dritten Bild. Die mit den verschiedenen Modellen berechneten Segmentierungen sind visuell beurteilt ähnlich gut. Das FCN-32s trennt Hund und Katze visuell am besten. In den Ausgaben aller Modelle ist die Schwierigkeit der Klassifikation der restlichen Pixel erkennbar. Das Bild zeigt ein Beispiel, bei dem die Klassifikation des weitgehend homogenen Hintergrundes als *Sofa* nur teilweise erfolgt. Die liegenden Tiere, die Stofftextur und die Kissen sind Anhaltspunkte für die Wahrnehmung des Menschen, dass ein Sofa abgebildet ist. Da das Sofa nicht vollständig innerhalb des Bildes dargestellt ist, sind jedoch keine Konturen des Objektes vorhanden. Ein weiteres Problem ist die Verwendung einer generischen Hintergrundklasse, zu der auch der visuell ähnliche Radweg auf dem dritten Bild gehört.

5.4.2 *Training mit schwachen Annotationen*

Neben dem Training mit vollständigen Annotationen, werden das FCN-32s und das DNet zudem mit schwachen Annotationen trainiert. Im Folgenden werden die Ergeb-

nisse der Modelle nach dem Training mit Bild-Level-Annotationen, Punktannotationen und der Kombination von Punktannotationen und Objectness präsentiert.

FCN-32s

Die mit dem FCN-32s erreichte MIoU beim Training mit verschieden starken Annotationen ist in Tabelle 5.4.2 dargestellt. Bei Verwendung von Bild-Level-Annotationen zum Training erreichten Bearman et al. [BRFFF16] mit dem FCN-32s eine MIoU von 29,80 % nach 50.000 Iterationen. Mit Implementierung des FCN-32s, welche im Rahmen dieser Arbeit erstellt wurde, wird bei einem Training mit ausschließlich Bild-Level-Annotationen eine bessere MIoU von 29,99 % nach 50.000 Iterationen und 30,25 % nach 60.000 Iterationen. Mögliche Gründe für die geringe Verbesserung gegenüber dem Ergebnis von Bearman et al. [BRFFF16] ist, dass im Rahmen dieser Arbeit sowohl die Modelle als auch der Trainingsprozess vollständig neu in einem anderen Framework (PyTorch anstelle von Caffe[5]) implementiert wurden. Zudem sind Unterschiede in der Vorverarbeitung der Eingabebilder möglich. Im Vergleich zu der MIoU von 11,56 % des mit vortrainierten Gewichten initialisierten Modells, ist zu erkennen, dass die stärkste Verbesserung der Segmentierungsleistung während der ersten 10.000 Iterationen erreicht wird. In den folgenden Iterationen werden nur kleine Verbesserungen von weniger als einem Prozentpunkt alle 10.000 Iterationen erzielt. Auch Pathak et al. [PSLD15], deren Arbeit Vorbild für den ersten Term der Bild-Level-Loss-Funktion ist, verwenden das FCN als Architektur und berichten, dass mit dem MIL-Loss (vgl. Abschnitt 3.2.4) bereits nach weniger als 10.000 Iterationen eine Konvergenz im Training erreicht wurde.

Verglichen mit dem FCN-32s, welches mit vollständigen Annotationen trainiert wurde, berechnet das FCN-32s nach Training mit Bild-Level-Annotationen eine deutlich schlechtere Segmentierung. Während der Annotationsaufwand zur Erstellung von Bild-Level-Annotationen mit durchschnittlich 20 Sekunden pro Bild weniger als einem Zehntel des Aufwands zur Erstellung einer vollständigen Annotation entspricht (vgl. Abschnitt 3.2.2), wird die Segmentierungsleistung mit einer Verringerung um 19,88 Prozentpunkte um weniger als die Hälfte geringer. Abbildung 5.4.2 zeigt qualitative Ergebnisse am Beispiel von vier Bildern aus dem Pascal VOC 2012 Validierungsdatensatz [EVGW+12] nach dem Training mit unterschiedlichen Formen schwacher Annotationen. Die vierte Spalte zeigt die Ausgabe des FCN-32s, welches über 60.000 Iterationen mit Bild-Level-Annotationen trainiert wurde. Mit Ausnahme des Bildes in der ersten Zeile ist zu erkennen, dass in der Ausgabe weitgehend die korrekten Klassen vorhandenen sind. Die Verhältnisse der Positionen der Klassen stimmen weitgehend

5 https://caffe.berkeleyvision.org/

Tabelle 5.4.2: Ergebnisse (in % MIoU) auf dem Pascal VOC 2012 Validierungsdatensatz nach dem Training mit unterschiedlich starken Annotationen. Angegeben sind die Referenzergebnisse des FCN-32s, trainiert von Bearman et al. [BRFFF16], nach 50.000 Iterationen. Für die in dieser Arbeit trainierten Modelle wurde die Segmentierungsleistung alle 10.000 Iterationen bewertet. Das FCN-32s mit einer Initialisierung mit vortrainierten Gewichten, wie in Abschnitt 5.2.2 beschrieben, erreicht bereits vor dem Training eine MIoU von 10,56 %.

Iteration	10.000	20.000	30.000	40.000	50.000	60.000
Vollst. [BRFFF16]					58,30 %	
Vollst.	39,30 %	44,39 %	46,71 %	48,11 %	49,26 %	50,13 %
Bild-Level [BRFFF16]					29,80 %	
Bild-Level	27,41 %	28,81 %	29,48 %	29,71 %	29,99 %	30,25 %
Punkt [BRFFF16]					35,10 %	
Punkt	27,14 %	30,32 %	32,95 %	34,11 %	35,31 %	36,19 %
Pkt. & Obj. [BRFFF16]					42,70 %	
Pkt. & Obj.	34,81 %	37,01 %	38,33 %	39,02 %	39,54 %	39,82 %

mit denen der vollständigen Annotation überein. Es werden jedoch zu große Bereiche des Bildes den jeweiligen Klassen zugewiesen. Wie auch beim FCN-32s, welches mit vollständigen Annotationen trainiert wurde, ist die Detaillierung der Konturen in den vier Beispielen sehr gering.

Bei der Verwendung von Punktannotationen im Training stehen, zusätzlich zu Informationen zu An- und Abwesenheit einer Klasse, Informationen zur Position vorhandener Klassen im Bild zur Verfügung. Da im Training genau ein Punkt pro vorhandener Klasse berücksichtigt wurde, entspricht der Annotationsaufwand im Durchschnitt 22,1 Sekunden pro Bild. Trotz nur durchschnittlich 2,1 Sekunden zusätzlichem Annotationsaufwand pro Bild, wird die Qualität der Segmentierung erhöht. Bearman et al. [BRFFF16] erreichen eine Verbesserung von 5,3 Prozentpunkten im Vergleich zur Verwendung von Bild-Level-Annotationen. Das im Rahmen dieser Arbeit mit Punktannotationen trainierte Modell ist mit einer MIoU von 36,19 % nach 60.000 Iterationen 5,94 Prozentpunkte besser als das mit Bild-Level-Annotationen trainierte Modell. Dies zeigt die Bedeutung der Verwendung von Informationen zur Lokalisierung der Klassen im Bild während des Trainings, wenn eine strukturierte Ausgabe in Form einer Klassifikationskarte berechnet werden soll. Auch beim Training mit Punktannotationen konnten in dieser Arbeit bessere Ergebnisse als in der

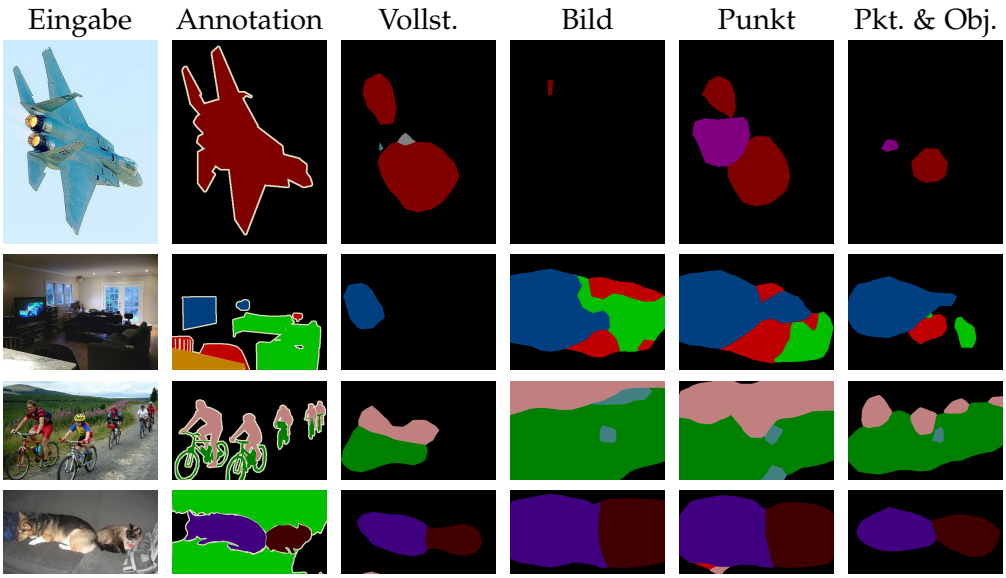


Abbildung 5.4.2: Qualitative Ergebnisse des FCN-32s bei Training mit vollständigen Annotationen (Vollst.), Bild-Level-Annotationen (Bild), Punktannotationen (Punkt) und Punktannotationen zusammen mit Objectness (Pkt. & Obj.) am Beispiel von vier Bildern aus dem Pascal VOC 2012 Validierungsdaten [EVGW+12]. Die abgebildeten Segmentierungen wurden jeweils mit dem besten Modell berechnet.

Arbeit von Bearman et al. [BRFFF16] erreicht werden. Beim Training mit Punktannotationen zeigt sich nach den ersten 10.000 Iterationen etwa dieselbe Verbesserung der MIoU wie beim Training mit Bild-Level-Annotationen. Jedoch werden auch in den folgenden Trainingsiterationen noch deutliche Verbesserungen der MIoU erzielt. Die qualitativen Ergebnisse der Segmentierung durch das FCN-32s, welches mit Punktannotationen trainiert wurde, sind in der fünften Spalte von Abbildung 5.4.2 abgebildet. Die Ergebnisse sind ähnlich zu den Ergebnissen des FCN-32s, welches mit Bild-Level-Annotationen trainiert wurde. Die Segmentierung ist ebenfalls wenig genau, enthält jedoch weitgehend die korrekten Klassen. Außer bei dem ersten Beispielbild sind die den vorhandenen Klassen zugewiesenen Flächen etwas kleiner, wodurch die Segmentierung präziser ist.

Eine Erweiterung der Informationen zum Training wird durch die Berücksichtigung der Objectness, als die Wahrscheinlichkeit eines Pixels zu einer Vordergrundklasse zu gehören, erreicht. Bearman et al. [BRFFF16] konnten durch Verwendung der Objectness als zusätzliches Wissen zu den Punktannotationen eine Verbesserung von

7,6 Prozentpunkten zu einer MIoU von 42,70 % nach 50.000 Iterationen erreichen. Im Gegensatz zu den Ergebnissen nach dem Training mit Bild-Level-Annotationen und Punktannotationen konnten in dieser Arbeit bei Training mit Punktannotationen und Objectness keine besseren Ergebnisse erreicht werden als in der Arbeit von Bearman et al. [BRFFF16]. In Folge dessen ist die Verbesserung durch Berücksichtigung der Objectness geringer und nach 60.000 Iterationen wird eine MIoU von 39,82 % erreicht. Zur Kombination der Loss-Funktion 4.2.4, welche die Objectness bewertet, und der Loss-Funktion 4.2.3 wurden die Funktionen in dieser Arbeit summiert. Eine eventuell andere Gewichtung der Summanden durch Bearman et al. [BRFFF16] ist ein möglicher Grund für den Unterschied der Ergebnisse. An den Ergebnissen ist dennoch erkennbar, dass durch die Erweiterung der Punktannotationen durch die Objectness eine Verbesserung der Segmentierungsleistung erzielt werden kann. Die pixelweise Wahrscheinlichkeit von Vordergrundobjekten erweitert die Informationen zur Position der Klassen um Informationen zur Ausdehnung der Vordergrundobjekte. Den Einfluss dieser Information auf die Ausgabe des Netzes nach dem Training ist in den Beispielen in der sechsten Spalte von Abbildung 5.4.2 erkennbar. Die Positionierung der Klassen ist ähnlich zu den anderen Ausgaben, die Ausdehnung der Bereiche der jeweiligen Klassen ist jedoch deutlich geringer. Insbesondere im dritten Bild ist die Fläche der Klasse *Person* in der Ausgabe unterteilt und stellt eine deutlich genauere Approximation der korrekte Segmentierung dar als die Ausgaben der Modelle, die ohne Berücksichtigung der Objectness trainiert wurden. Auch im vierten Bild ist diese Verkleinerung der Fläche, welche einer Vordergrundklasse zugewiesen wurde, besonders deutlich erkennbar.

Des Weiteren zeigt das vierte Bild ein Problem bei der Berechnung und Verwendung der Objectness bei Bildern, welche fast vollständig in Vordergrundklassen segmentiert sind. Die in Abschnitt 3.2.3 beschriebenen Kriterien bewerten die Objectness von Bildausschnitten im Vergleich zur Umgebung dieser Ausschnitte. Objekte, wie der Hund oder die Katze, haben im Vergleich zum Hintergrund ein anderes Aussehen. Der Hintergrund, das Sofa, hat jedoch keine Kontur, welche auf dem Bild enthalten ist, und eine homogene Farbverteilung. Somit ist als Vorhersage des Objectness-Modells für die Pixel des Sofas eine geringe Objectness zu erwarten. Der Objectness-Loss in Formel 4.2.4 steht mit dieser Zuweisung im Konflikt, da an Pixeln mit hoher Objectness Vordergrund- und an Pixeln mit niedriger Objectness Hintergrundklassen klassifiziert werden sollen, um den Loss zu minimieren.

DNet

Als weitere Architektur wurde das DNet neben vollständigen Annotationen ebenfalls mit Bild-Level-Annotationen, Punktannotationen und der Kombination von Punktannotationen und Objectness trainiert. Die Ergebnisse des Trainings mit schwachen Annotationen werden in diesem Abschnitt mit den Ergebnissen des DNets, welches mit vollständigen Annotationen trainiert wurde, und mit den Ergebnissen des FCN-32s nach Training mit denselben schwachen Annotationen verglichen. Abbildung 5.4.3 zeigt qualitative Ergebnisse der besten Modelle mit den unterschiedlichen Annotationen.

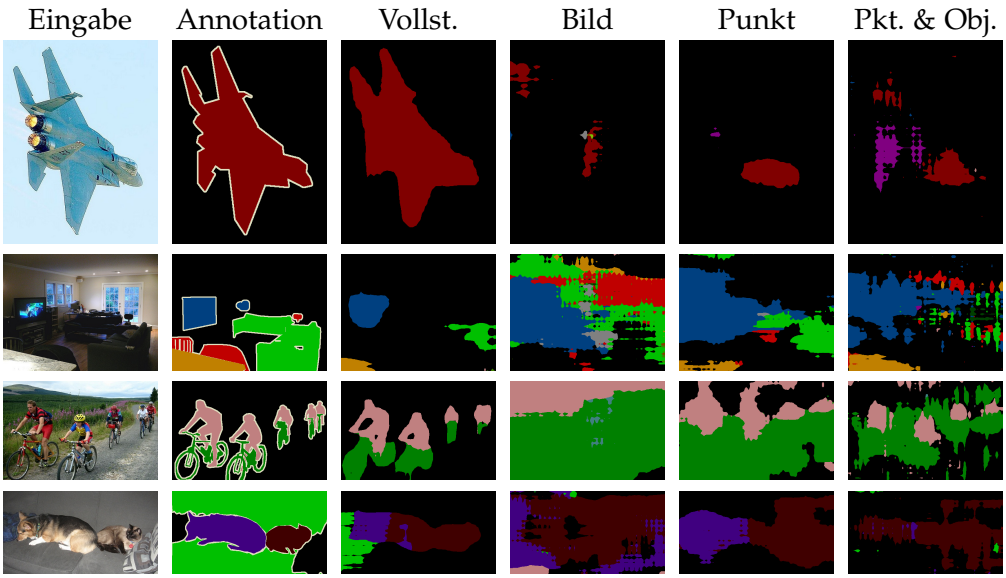


Abbildung 5.4.3: Qualitative Ergebnisse des DNets bei Training mit vollständigen Annotationen (Vollst.), Bild-Level-Annotationen (Bild), Punktannotationen (Punkt) und Punktannotationen zusammen mit Objectness (Pkt. & Obj.) am Beispiel von vier Bildern aus dem Pascal VOC 2012 Validierungsdaten [EVGW+12]. Die abgebildeten Segmentierungen wurden jeweils mit dem besten Modell berechnet.

Bei einem Training mit vollständigen Annotationen berechnet das DNet bessere Segmentierungen als das FCN-32s. Beim Training mit Bild-Level-Annotationen ist die Qualität der Segmentierung jedoch deutlich geringer als die der Ausgaben des FCN-32s. Wird beim Training des DNets mit Bild-Level-Annotationen dieselbe Lernrate ($\alpha = 10^{-3}$) wie beim Training mit vollständigen Annotation verwendet, wird mit dem

Modell eine MIoU von 15,62 % (ohne Augmentierungen) auf dem Pascal VOC 2012 Validierungsdatensatz erzielt, während mit dem FCN-32s eine MIoU von 31,69 % erreicht wird. Im Vergleich zum DNet, welches mit vollständigen Annotationen trainiert wurde, ist folglich die Verschlechterung von 56,59 % auf 15,62 % MIoU bei Verwendung von Bild-Level-Annotationen stärker, als beim FCN-32s mit einer Verschlechterung der MIoU von 50,13 % auf 30,25 %. Bei Bewertung des mit vortrainierten Gewichten initialisierten Modells wird vor dem Training eine MIoU von 9,77 % erreicht. Die beim Training erreichte Verbesserung ist im Vergleich zu der beim FCN-32s erreichten Verbesserung deutlich geringer.

Ein Problem beim Training des DNets ist die von Yu et al. [YK15] verwendete Vorverarbeitung. Als Eingabebild wird nur ein Ausschnitt des Bildes und als Annotation ein nochmals verkleinerter Ausschnitt davon verwendet. In Folge dessen können alle Instanzen einer Klasse auf dem Eingabebild durch die Wahl des Ausschnitts wegfallen. Bei Verwendung vollständiger Annotationen wird dies durch die Wahl des Ausschnitts der pixelweisen Annotation berücksichtigt. Bei Bild-Level-Annotationen ist dies nicht möglich, da keine Informationen zur Position der Instanzen der vorhandenen Klassen bekannt sind. Folglich kann im Training der Fall eintreten, dass Klassenlabels für die Berechnung der Loss-Funktion als vorhanden angenommen werden, welche durch die Wahl der Bildausschnitts weggefallen sind. Durch die Verwendung zufälliger Bildausschnitte als Augmentierung kann dieses Problem nicht vermieden werden. Im Gegensatz zur Wahl fester Ausschnitte wird jedoch durch die Wahl verschiedener Bildausschnitte bewirkt, dass nicht in jeder Epoche des Trainings dieselben Objektinstanzen auf einem Bild wegfallen.

Da beim Training des FCN-32s für jedes Annotationslevel die von Bearman et al. [BRFFF16] verwendete, kleinere Lernrate von $\alpha = 10^{-5}$ gewählt wurde, wurde das DNet mit verschiedenen Lernraten $\alpha \in \{10^{-3}, 10^{-4}, 10^{-5}\}$ trainiert. Zudem wurde mit jeder Lernrate ein Training mit und ein Training ohne Augmentierung durch zufällige horizontale Spiegelung und die Auswahl eines zufälligen Bildausschnitts durchgeführt. Abbildung 5.4.4 zeigt den Verlauf der MIoU über die 60.000 Iterationen des Trainings. Die Modelle wurden alle 10.000 Iterationen bewertet. Die Werte der MIoU sind in Tabelle A.0.2a im Anhang gegeben. Wie beim Training mit vollständigen Annotationen wird auch beim Training mit Bild-Level-Annotationen bereits sehr früh die beste MIoU erreicht. Mit dem DNet, welches mit der Lernrate $\alpha = 10^{-3}$ und ohne Augmentierung trainiert wurde, und mit den Modellen, welche mit der Lernrate $\alpha = 10^{-5}$ trainiert wurden, wird bereits nach 10.000 Iterationen das beste Ergebnis erreicht. Abbildung 5.4.4 zeigt, dass die besten Ergebnisse mit einer MIoU von 18,32 % und 21,96 % (aug.) mit der kleinsten Lernrate von $\alpha = 10^{-5}$ erreicht werden. Eine deutliche Verbesserung durch Verwendung von Augmentierung wie beim Training mit

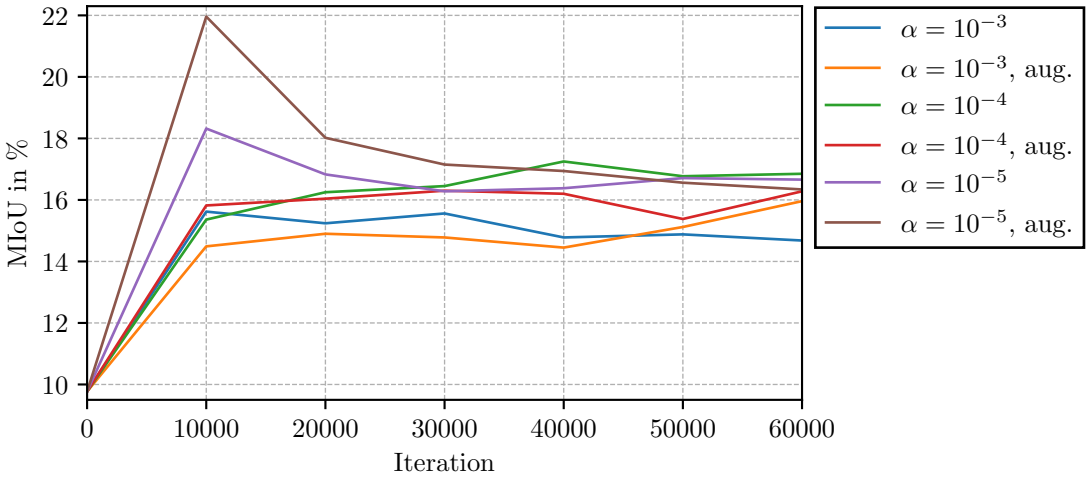


Abbildung 5.4.4: Verlauf der MIoU während des Trainings des DNets mit Bild-Level-Annotationen. Das Training wurde mit unterschiedlichen Lernraten $\alpha \in \{10^{-3}, 10^{-4}, 10^{-5}\}$ und mit den in Abschnitt 5.2 beschriebenen Augmentierungen (gekennzeichnet mit aug.) durchgeführt.

vollständigen Annotationen ist nur bei einer Lernrate von $\alpha = 10^{-5}$ zu beobachten. Bei einer Lernrate von $\alpha = 10^{-3}$ wird nur eine minimale Verbesserung von 15,62 % MIoU zu 15,96 % MIoU erreicht. Der Trainingsfortschritt wird im Gegenzug verlangsamt, sodass das minimal bessere Ergebnis erst nach 60.000 Iterationen erreicht werden. Bei Verwendung der Lernrate $\alpha = 10^{-4}$ ist die MIoU des Modells, das mit Augmentierung trainiert wurde, mit 16,30 % (bestes Ergebnis nach 30.000 Iterationen) schlechter als die 17,25 % MIoU (bestes Ergebnis nach 40.000 Iterationen) des Modells, das ohne Augmentierung trainiert wurde.

Die vierte Spalte von Abbildung 5.4.3 zeigt qualitative Ergebnisse am Beispiel von vier Bildern aus dem Pascal VOC 2012 Validierungsdatensatz [EVGW+12]. Die Segmentierung wurde mit dem Modell berechnet, welches die beste MIoU nach dem Training mit Bild-Level-Annotationen erreichte ($\alpha = 10^{-5}$, 10.000 Iterationen mit augmentierten Trainingsdaten). Ähnlich wie bei der Segmentierung des FCN-32s, welches mit Bild-Level-Annotationen trainiert wurde, werden, mit Ausnahme des ersten Bildes, große Bereiche mit den vorhandenen Vordergrundklassen segmentiert. Die Konturen der Regionen verlaufen jedoch visuell sehr ungenau.

Beim Training des DNets mit Punktannotationen konnten die Ergebnisse im Vergleich zum Training mit Bild-Level-Annotationen deutlich verbessert werden. Auch bei Punktannotationen wurde das DNet zuerst mit einer Lernrate von $\alpha = 10^{-3}$ und ohne

Augmentierung trainiert. Mit 33,4 % nach 40.000 Iterationen ist die MIoU um 17,78 Prozentpunkte höher als nach dem Training des DNets mit Bild-Level-Annotationen und denselben Hyperparametern für das Training. Zudem ist das Ergebnis nur wenig schlechter als das des FCN-32s von Bearman et al. [BRFFF16], mit dem eine MIoU von 35,1 % erreicht wurde.

Auch bei Verwendung der Punktannotationen kann die Vorverarbeitung Probleme verursachen. Im Vergleich zum Training mit Bild-Level-Annotationen können die vorhandenen Klassen aus den Punktannotationen ermittelt werden. In Folge dessen sind Positionen zu den vorhandenen Klassen bekannt. In einem weiteren Experiment wurden nur die Klassen der im gewählten Ausschnitt der Annotation enthaltenen Punkte und die Hintergrundklasse als vorhandene Klassen betrachtet. Mit diesem Vorgehen zur Ermittlung der auf dem Bild vorhandenen Klassen wurde während des Trainings höchstens eine MIoU von 26,82 % erreicht. Das Problem bei dieser Ermittlung der Bild-Level-Annotationen liegt darin, dass nur eine Objektinstanz auf dem Bild mit einem Punkt markiert ist. Liegt die markierte Objektinstanz der Klasse außerhalb des Ausschnitts der Annotation, kann dennoch eine andere Instanz der Klasse im Ausschnitt des Eingabebildes enthalten sein, wird jedoch bei der Berechnung der Loss-Funktion als nicht vorhanden angenommen. Liegt die markierte Objektinstanz nur teilweise innerhalb des Ausschnitts der Annotation, kann dasselbe Problem auftreten, abhängig von der Lage des Punktes auf dem Objekt. Aus diesem Grund wurden dieselben Bild-Level-Annotationen wie für das Training mit ausschließlich Bild-Level-Annotationen verwendet. Bei der Berechnung des zweiten Teils des Punkt-Loss (vgl. Formel 4.2.3) wurden nur die Punkte berücksichtigt, welche im gewählten Ausschnitt der Annotation liegen. Da bei diesen Punkten nur die Anwesenheit, nicht aber die Abwesenheit der Punkte aller anderen Klassen Einfluss auf den Punkt-Loss hat, sollte diese Auswahl einen geringeren Einfluss auf das Training haben als die Annahme zu vieler vorhandenen Klassen.

Auch bei dem Training mit Punktannotationen wurden kleinere Lernraten $\alpha \in \{10^{-3}, 10^{-4}, 10^{-5}\}$ und Augmentierungen verwendet. Der Verlauf der MIoU über 60.000 Iterationen ist in Abbildung 5.4.5 dargestellt. Die Werte der MIoU alle 10.000 Iterationen sind in Tabelle A.0.2b im Anhang gegeben. Abbildung 5.4.5 zeigt, dass die verwendete Augmentierung beim Training des DNets mit Punktannotationen bei allen Lernraten zu einer Verschlechterung der MIoU führt. Mit dem besten Modell ($\alpha = 10^{-5}$), welches mit augmentierten Eingaben trainiert wurde, wird eine MIoU von 29,83 % erreicht, während mit dem besten Modell ($\alpha = 10^{-4}$) ohne Augmentierung der Trainingsdaten eine MIoU von 34,79 % erzielt wird. Vermutlich liegt der Grund in der starken Reduktion der berücksichtigten Punkte. Im Vergleich zu durchschnittlich 0,61 Punkten pro Bild ohne Augmentierung, werden bei augmentierten Trainingsdaten

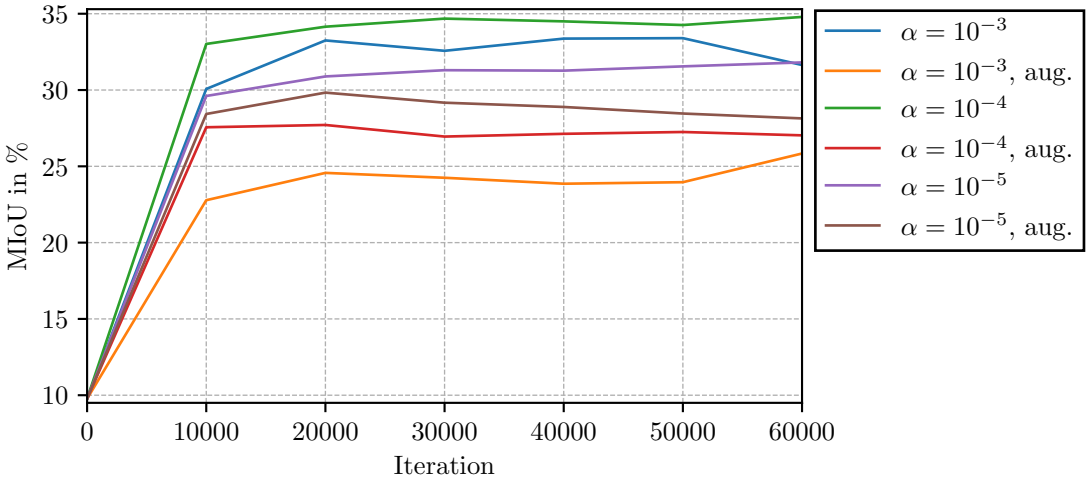


Abbildung 5.4.5: Verlauf der MIoU während des Trainings des DNets mit Punktannotationen. Das Training wurde mit unterschiedlichen Lernraten $\alpha \in \{10^{-3}, 10^{-4}, 10^{-5}\}$ und mit den in Abschnitt 5.2 beschriebenen Augmentierungen (gekennzeichnet mit aug.) durchgeführt.

durchschnittlich nur 0,27 Punkte pro Bild bei der Berechnung der Loss-Funktion berücksichtigt.

Eine Verringerung der Lernrate führt auch beim Training mit Punktannotationen zu einem besseren Ergebnis. Das Modell, mit dem die beste Segmentierung berechnet wird, wurde mit eine Lernrate von $\alpha = 10^{-4}$ trainiert. Mit 34,79 % MIoU ist die Qualität der Segmentierung nur wenig geringer als die des FCN-32s von Bearman et al. [BRFFF16] mit 35,1 % MIoU. Unterschiede in der Wahl der Lernrate und der Verwendung von augmentierten Trainingsdaten sind bereits nach 10.000 Iterationen erkennbar. Im Gegensatz zum Training mit Bild-Level-Annotationen bleiben diese Unterschiede während des weiteren Trainings erhalten. Zudem sind bei allen Varianten nach 10.000 Iterationen weitere Verbesserungen der MIoU zu beobachten.

Die Verbesserung durch die zusätzlich genutzte Information der Positionen der vorhandenen Klassen führt beim DNet zu visuellen Verbesserungen der Segmentierung der vier Beispielbilder (s. fünfte Spalte von Abbildung 5.4.3). In der Segmentierung der Bilder in den letzten beiden Zeilen sind die Konturen visuell deutlich besser positioniert und detaillierter als bei der Segmentierung des mit Bild-Level-Annotationen trainierten DNets. Insbesondere die Radfahrer wurden in getrennte Instanzen segmentiert.

Das Verwenden der Objectness als zusätzliche Information zu den Punktannotationen führt nicht bei jeder Kombination von Lernrate und der Verwendung von Augmentierung zu einer Verbesserung gegenüber den Modellen, die ausschließlich mit Punktannotationen trainiert wurden. Tabelle 5.4.3 zeigt die jeweils besten Ergebnisse, welche mit dem DNet für unterschiedliche Hyperparameter durch Training mit Punktannotationen und der Kombination von Punktannotationen und Objectness erreicht werden. Zudem wird nach dem Padding unterschieden, welches bei der Vorverarbeitung der Objectness-Karten verwendet wurde. Ein mögliches Problem bei der Verwendung von Zero-Padding zur Vergrößerung der Objectness-Karten ist, dass bei der Berechnung der Loss-Funktion der aufgefüllte Bereich die niedrigste mögliche Wahrscheinlichkeit ($\mathbf{p}_i = 0$) für eine Vordergrundklasse hat. Das Eingabebild wird hingegen mittels Reflection-Paddings vergrößert, sodass durch die Reflexion auch Vordergrundobjekte in dem aufgefüllten Bereich vorhanden sein können. Aus diesem Grund wurde zum Vergleich zudem das DNet mit Objectness-Karten trainiert, welche mittels Reflection-Paddings statt Zero-Paddings vergrößert wurden. Tabelle 5.4.3 zeigt, dass die Verwendung der verschiedenen Arten des Paddings nur zu sehr kleinen Abweichungen bei der Qualität der Segmentierung führt. Die durchschnittliche Differenz der MIoU liegt bei 0,27 Prozentpunkten. Zudem ist zu erkennen, dass ohne Augmentierung nur minimale Unterschiede zum Training mit ausschließlich Punktannotationen bestehen. Die besten Ergebnisse ohne Augmentierung der Trainingsdaten werden beim Training mit einer Lernrate von $\alpha = 10^{-3}$ erreicht. Die MIoU liegt bei Verwendung von Zero-Padding bei der Vorverarbeitung der Objectness-Karten bei 33,67 % nach 10.000 Iterationen. Damit erreicht das Modell dieselbe MIoU wie das DNet, welches ohne Objectness trainiert wurde. Mit dem Modell, bei dessen

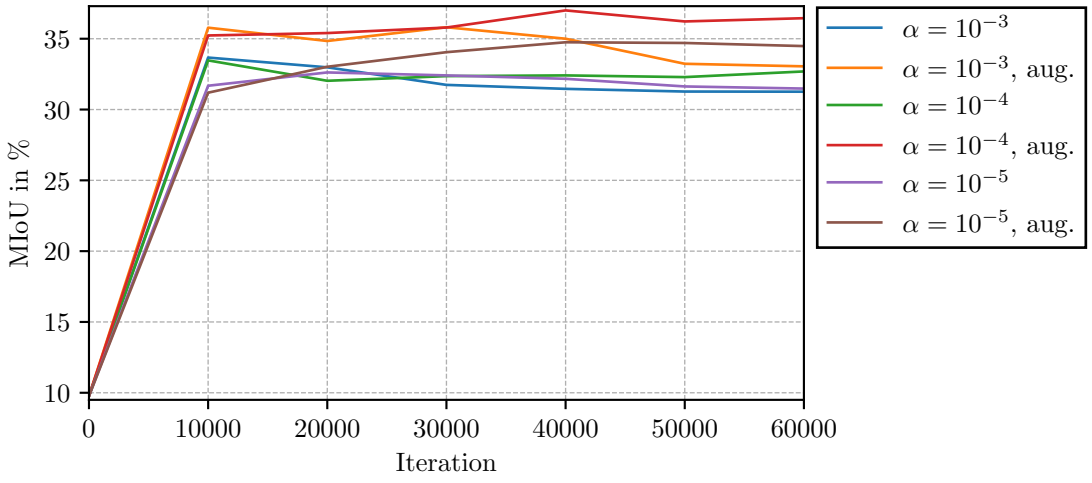
Tabelle 5.4.3: Vergleich der MIoU der Modelle des DNets, bei deren Training die Objectness-Karten bei der Vorverarbeitung mit Zero-Padding und mit Reflection-Padding vergrößert wurden. Angegeben ist die MIoU für das jeweils beste Modell für verschiedene Lernraten α jeweils mit (aug.) und ohne Augmentierung der Trainingsdaten (n. aug.). Die erste Zeile dient zum Vergleich mit Modellen, welche ausschließlich mit Punktannotationen trainiert wurden.

	$\alpha = 10^{-3}$		$\alpha = 10^{-4}$		$\alpha = 10^{-5}$	
	n. aug.	aug.	n. aug.	aug.	n. aug.	aug.
Punktannotationen	33,40 %	25,84 %	34,79 %	27,71 %	31,81 %	29,83 %
Zero-Padding	33,67 %	35,81 %	33,47 %	37,00 %	32,62 %	34,75 %
Reflection-Padding	33,12 %	36,49 %	33,27 %	36,84 %	32,62 %	34,78 %

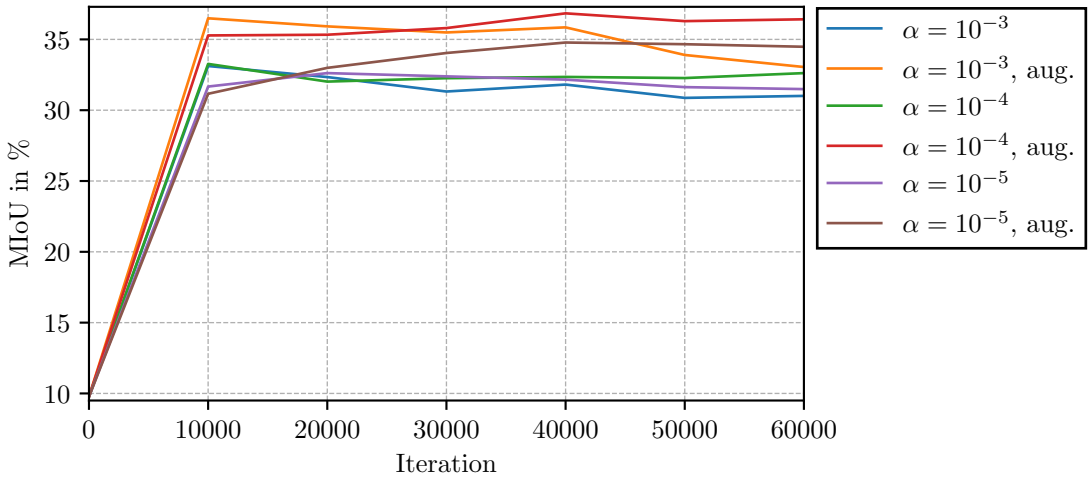
Training die Objectness-Karten mittels Reflection-Paddings vergrößert wurden, wird mit einer MIoU von 33,27 % nach 10.000 Iterationen Training mit einer Lernrate von $\alpha = 10^{-4}$ ein schlechteres Ergebnis erreicht als mit dem DNet, welches ohne die Objectness trainiert wurde. Im Durchschnitt werden mit den Modelle, welche ohne Augmentierung mit Punktannotationen und Objectness trainiert wurden, eine um 0,08 Prozentpunkte schlechtere MIoU erreicht als mit den Modellen, welche mit denselben Hyperparametern ohne Objectness trainiert wurden.

Im Gegensatz zum Training mit ausschließlich Punktannotationen, werden beim Training mit Objectness als zusätzliches Wissen die Ergebnisse durch die Augmentierung der Trainingsdaten verbessert. Die Probleme der Augmentierung bei Bild-Level-Annotationen und insbesondere bei Punktannotationen, welche zuvor diskutiert wurden, bleiben beim Training mit Punktannotationen und Objectness bestehen, da der Loss als Summe des Punkt-Loss (vgl. Formel 4.2.3) und des Objectness-Loss (vgl. Formel 4.2.4) berechnet wird. Die Objectness ist jedoch im Gegensatz zur Punktannotation für jedes Pixel des Eingabebildes gegeben. Der Effekt von Spiegelung und der Wahl zufälliger Ausschnitte entspricht somit dem Effekt der Augmentierung bei pixelweise annotierten Trainingsdaten. Eine Verschiebung des Ausschnitts des Eingabebildes führt zu einer entsprechenden Verschiebung des Ausschnitts der Objectness-Karte. Nur bei Verwendung augmentierter Daten zum Training wird eine Verbesserung von mehr als einem Prozentpunkt im Vergleich zu den Modellen, welche ohne Objectness trainiert wurden, erreicht. Das beste Ergebnis wird nach 40.000 Iterationen Training mit augmentierten Daten und der Lernrate $\alpha = 10^{-4}$ mit MIoU von 37 % erreicht. Die gemessene Qualität der Segmentierung ist damit nur 0,81 Prozentpunkte besser, als die des FCN-32s, welches mit Punktannotationen trainiert wurde.

Der Verlauf der mit den Modellen erreichten MIoU ist in Abbildung 5.4.6 dargestellt. Die Werte der MIoU alle 10.000 Iterationen sind für die Verwendung von Zero-Padding bei der Vorverarbeitung der Objectness-Karten in Tabelle A.0.1c und bei Verwendung von Reflection-Padding in Tabelle A.0.1d im Anhang gegeben. Mit Ausnahme des Modells, welches mit einer Lernrate von $\alpha = 10^{-3}$ und augmentierten Daten trainiert wurde, erreichen die Modelle bei gleichen Hyperparametern, jedoch unabhängig vom verwendeten Padding, nach derselben Anzahl von Iterationen die höchste MIoU. Wie bei Bild-Level-Annotationen und Punktannotationen zuvor, wird größte Verbesserung der MIoU innerhalb der ersten 10.000 Iterationen erreicht. Ohne Augmentierung der Trainingsdaten wird mit einer Lernrate von $\alpha \in \{10^{-3}, 10^{-4}\}$ nach den ersten 10.000 Iterationen bereits die beste MIoU erreicht. Bei Training ohne Augmentierung mit der Lernrate $\alpha = 10^{-5}$ wird nach 20.000 Iterationen das beste Ergebnis erzielt. Die Verwendung augmentierter Trainingsdaten führt nach den ersten 10.000 Iterationen zu einer maximalen Verbesserung von 3,62 Prozentpunkten MIoU in den folgenden



(a) Punktannotationen & Objectness mit Zero-Padding



(b) Punktannotationen & Objectness mit Reflection-Padding

Abbildung 5.4.6: Verlauf der MIoU während des Trainings des DNets mit Punktannotationen und Objectness als zusätzliche Information. Die Objectness-Karte wurde bei der Vorverarbeitung, wenn nötig, mittels Zero-Padding (a) und mittels Reflection-Padding (b) vergrößert. Das Training wurde mit unterschiedlichen Lernraten $\alpha \in \{10^{-3}, 10^{-4}, 10^{-5}\}$ und mit den in Abschnitt 5.2 beschriebenen Augmentierungen (gekennzeichnet mit aug.) durchgeführt.

Iterationen. Nach spätestens 40.000 Iterationen kommt es bei keinem der Experimente mit Punktannotationen und Objectness zu einer weiteren Verbesserung der MIoU.

Bei den Segmentierungen der vier Beispielbilder (s. Abbildung 5.4.3) sind auch visuell keine Verbesserungen durch die zusätzliche Berücksichtigung der Objectness erkennbar. Ähnlich wie in den Ausgaben des FCN-32s, welches mit Punktannotationen und Objectness trainiert wurde, sind in den Ausgaben des DNets größere Bereiche als Hintergrund klassifiziert als beim Training ohne Berücksichtigung der Objectness.

5.4.3 *Vergleich der Architekturen und Annotationen*

In den Experimenten, die im Rahmen dieser Arbeit durchgeführt wurden, wurden das FCN-32s und das DNet mit unterschiedlich starken Annotationen trainiert. Tabelle 5.4.4 führt für jedes der verwendeten Annotationslevel den Annotationsaufwand und die MIoU der jeweils besten Modelle auf. Die Verwendung vollständiger Annotationen zum Training der Modelle führt mit 50,13 % MIoU beim FCN-32s, 56,59 % beim DNet und 60,89 % beim DNet, welches mit augmentierten Daten trainiert wurde, zu den besten Ergebnissen. Die Erstellung der vollständigen Annotationen ist jedoch mit einem Aufwand von durchschnittlich 239,7 Sekunden pro Bild sehr hoch. Der Aufwand zur Erstellung von Bild-Level-Annotationen entspricht mit durchschnittlich 20 Sekunden pro Bild etwa einem Zwölftel des Annotationsaufwands bei vollständigen Annotationen. Die mit dem FCN-32s erreichte MIoU ist mit 31,69 % deutlich geringer als bei der Verwendung vollständiger Annotationen. Trotz der schlechteren Segmentierungsleistung beim Training mit Bild-Level-Annotationen ist das Verhältnis zwischen Annotationsaufwand und MIoU besser als bei vollständigen Annotationen. Durch Berücksichtigung der nicht vorhandenen Klassen in der Loss-Funktion konnte im Vergleich zur Arbeit von Pathak et al. [PSLD15] die erreichte MIoU von 25,1 % auf 30,25 % gesteigert werden. Bearman at al. [BRFFF16] konnten zudem durch die Berücksichtigung von Informationen zur Ausdehnung der Objekte mittels berechneter Objectness-Karten die mit dem FCN-32s erreichte MIoU auf 32,2 % erhöhen, bei nur

Tabelle 5.4.4: Vergleich der verschiedenen Level der Annotationen anhand des Annotationsaufwands [BRFFF16] und der durch die jeweils besten Modelle erzielten MIoU auf dem Pascal VOC 2012 Validierungsdaten [EVGW+12].

Level	Aufwand	FCN-32s	DNet	DNet (aug.)
vollst. Annotation	239,7 s	50,13 %	56,59 %	60,89 %
Bild-Level-Annot.	20,0 s	30,25 %	18,32 %	21,96 %
Punktannotation	22,1 s	36,19 %	34,79 %	29,83 %
Pkt. & Objectness	22,4 s	39,82 %	33,67 %	37,00 %

minimaler Steigerung des Annotationsaufwands um 0,3 Sekunden pro Bild. Während bei der Verwendung vollständiger Annotationen zum Training des DNets bessere Ergebnisse erreicht werden als mit dem FCN-32s, sind die Ergebnisse, welche beim Training des DNets mit schwachen Annotationen erreicht werden, zum Teil deutlich schlechter. Das Training des DNets wurde auf Bildausschnitten durchgeführt, für welche die vorhandenen Klassen nicht eingeschränkt werden konnten, da bei Bild-Level-Annotationen keine Informationen zur Position der Klassen vorliegen. Dies ist eine mögliche Ursache für die schlechtere MIoU des DNets beim Training mit Bild-Level-Annotationen. Dennoch erreichen die Modelle etwa ein Drittel der MIoU der Modelle, welche mit vollständigen Annotationen trainiert wurden. Die Erstellung von Punktannotation mit einem Punkt pro Klasse ist mit 22,1 Sekunden pro Bild nur wenig aufwändiger als die Erstellung von Bild-Level-Annotationen. Die dadurch gegebene Information zu der Position der Klassen führt hingegen zu einer deutlichen Steigerung der MIoU bei allen Modellen und zeigt die Bedeutung der Verwendung von Ortsinformationen beim Training von Modellen zur semantischen Segmentierung. Insbesondere wird die Segmentierungsleistung des DNets erhöht. Fehler, welche durch die fälschliche Annahme im Bildausschnitt vorhandener Klassen entstehen, haben einen geringeren Einfluss auf den Loss, da zusätzlich korrekte Informationen zu den im Ausschnitt vorhandenen Klassen in Form annotierter Pixel gegeben sind. Auch bei der Verwendung von Punktannotationen konnte mit einem geringen zusätzlichen Aufwand die Objectness berücksichtigt werden. Die somit gegebenen Informationen zur Ausdehnung der Objekte führen zu einer weiteren Verbesserung der MIoU beim FCN-32s. Beim DNet wird bei der in dieser Arbeit verwendeten Vorverarbeitung nur bei Augmentierung der Trainingsdaten eine Verbesserung der MIoU im Vergleich zum Training mit ausschließlich Punktannotationen erreicht.

6 FAZIT

Tiefe neuronale Faltungsnetze haben zu Fortschritten bei vielen Problemen des maschinellen Sehens geführt [HKH17] und konnten bei der Klassifikation bereits menschliche Fehlerraten bei diesem Problem unterbieten [HZRS15]. Auch bei dem Problem der semantischen Segmentierung sind tiefe Faltungsnetze der aktuelle Stand der Technik. Ein Problem bei der Verwendung tiefer Faltungsnetze sind die großen Mengen von annotierten Daten, welche für das Training der Modelle benötigt werden. Der Aufwand zur Erstellung vollständiger Annotationen ist sehr hoch und entspricht für den Pascal VOC Datensatz [EVGW+10] etwa dem zwölffachen Annotationsaufwand von Bild-Level-Annotationen. Um die Notwendigkeit vollständiger Annotationen für das Training zu vermeiden, sind Methoden erforderlich, die ein Training mit schwachen Annotationen wie z. B. Bild-Level-Annotationen oder Punktannotationen ermöglichen.

Im Rahmen dieser Arbeit wurden das FCN-32s [LSD15] und das DNet [YK15] als Modelle zur semantischen Segmentierung untersucht. Beide basieren auf der Architektur des VGG-16 [SZ14] und wurden zur Vorhersage dichter Klassifikationskarten angepasst. Die vollständig verbundenen Schichten des VGG-16 wurden in Faltungsschichten umgewandelt und die Auflösung der Ausgabe mittels Transpose Convolution erhöht. Das DNet verwendet zudem anstelle der letzten beiden Max-Pooling-Schichten Faltungsschichten mit einem Dilation-Factor $r > 1$ (Dilated Convolutions), um das rezeptive Feld des Merkmale zu erhöhen und dabei die Auflösung der Feature-Maps zu erhalten. In dieser Arbeit wurden beide Modelle mit vollständigen und schwachen Annotationen trainiert. Als schwache Annotationen wurden Bild-Level-Annotationen, Punktannotationen und Objectness als zusätzliche Information verwendet.

Die Verbesserung der Segmentierungsleistung durch die teilweise Verwendung von Dilated Convolution anstelle von Pooling zeigt, dass durch die weniger verkleinerten Feature-Maps eine detailliertere und somit bessere Segmentierung berechnet werden konnte. Eine weitere Verbesserung konnte bei Augmentierung der Trainingsdaten durch horizontale Spiegelung und die zufällige Auswahl von Bildausschnitten erreicht werden. Die ausschließliche Verwendung von Informationen zu An- und Abwesenheit von Klassen auf den Trainingsbildern führt zu einer deutlichen Verschlechterung der MIoU. Im Verhältnis zu dieser Verschlechterung ist die Einsparung des Annotationsaufwands jedoch größer. Die Ergebnisse zeigen, dass die Verschlechterung abhängig von der gewählten Architektur und der Vorverarbeitung der Trainingsdaten ist. Wäh-

rend beim Training mit vollständigen Annotationen mit dem DNet bessere Ergebnisse erreicht werden, ist die Qualität der berechneten Segmentierungen nach dem Training mit schwachen Annotationen beim FCN-32s besser. Beim Training mit Ausschnitten anstelle der vollständigen Bilder können die Annotationen aufgrund fehlender Positionsinformationen nicht angepasst werden. Folglich ist es möglich, dass Klassen als vorhanden angenommen werden, welche nicht im Bildausschnitt enthalten sind. Da das DNet auf Bildausschnitten und das FCN-32s auf den vollständigen Bildern trainiert wurde, ist dies die vermutliche Ursache für die stärkere Verschlechterung beim DNet. Die zusätzliche Berücksichtigung eines Punktes pro vorhandener Klasse führt zu einer geringen Erhöhung des Annotationsaufwands gegenüber Bild-Level-Annotationen, doch zu einer deutlichen Verbesserung der berechneten Segmentierungen und somit einem besseren Kompromiss von Annotationsaufwand und Segmentierungsleistung. Dies zeigt die Bedeutung von Positionsinformationen für das Training von Modellen zur semantischen Segmentierung. Die Objectness bietet zusätzliche klassenunabhängige Informationen zur wahrscheinlichen Ausdehnung der Vordergrundobjekte. Beim FCN-32s und beim DNet, welches mit augmentierten Daten trainiert wurde, konnten durch die Berücksichtigung der Objectness die Ergebnisse verbessert werden. Beim DNet, welches ohne Augmentierungen trainiert wurde, führte diese jedoch zu einer geringen Verschlechterung der MIoU. Durch Verwendung eines separat trainierten Modells zur Berechnung der Objectness kann diese mit konstantem Annotationsaufwand für Datensätze verschiedenen Umfangs berechnet werden. Die verwendeten Kriterien zur Berechnung der Objectness berücksichtigen die Einzigartigkeit von Bereichen im Vergleich zu einem lokalen Umfeld oder zum gesamten Bild. Bei z. B. Vordergrundobjekten mit homogener Textur, deren Umfeld auf dem Bild nicht erfasst wurde, oder Bildern, welche fast vollständig in Vordergrundobjekte zu segmentieren sind, sind diese Kriterien jedoch nicht zutreffend und somit ggf. für andere Datensätze ungeeignet. Des Weiteren zeigen die Ergebnisse, dass bei der Wahl der Lernrate sowohl die Architektur als auch die verwendete Annotationsform berücksichtigt werden muss. Während das DNet beim Training mit vollständigen Annotationen mit einer vergleichsweise großen Lernrate die besten Ergebnisse erzielte, wurden bei der Verwendung schwacher Annotationen mit kleineren Lernraten die besten Ergebnisse erzielt.

Neben der Wahl der Hyperparameter beim Training ist für zukünftige Untersuchungen eine weitere Modifikation der Loss-Funktion von Interesse. Bei den Loss-Funktionen werden sowohl bei Bild-Level-Annotationen als auch bei Punktannotationen nur ein Pixel der Ausgabe pro Klasse bzw. Punkt berücksichtigt. Bei Fotos von Objekten ist jedoch anzunehmen, dass die Pixel in einem nahen Umfeld ebenfalls zu dem Objekt gehören. Aus diesem Grund wäre die zusätzliche Berücksichtigung des Umfelds eine zu untersuchende Veränderung der Loss-Funktionen.

LITERATURVERZEICHNIS

[ADF12] Alexe, B. ; Deselaers, T. ; Ferrari, V.: Measuring the Objectness of Image Windows. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34 (2012), Nr. 11, S. 2189–2202

[Agg18] Aggarwal, C. C.: *Neural Networks and Deep Learning*. Springer International Publishing, 2018

[BFC09] Brostow, G. J. ; Fauqueur, J. ; Cipolla, R.: Semantic object classes in video: A high-definition ground truth database. In: *Pattern Recognition Letters* 30 (2009), Nr. 2, S. 88–97

[BKC17] Badrinarayanan, V. ; Kendall, A. ; Cipolla, R.: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2017), Nr. 12, S. 2481–2495

[BMC15] Badrinarayanan, V. ; Mishra, B. ; Cipolla, R.: Understanding symmetries in deep networks. In: *arXiv preprint arXiv:1511.01029* (2015)

[BRFFF16] Bearman, A. ; Russakovsky, O. ; Ferrari, V. ; Fei-Fei, L.: What's the Point: Semantic Segmentation with Point Supervision. In: *Proc. European Conf. Computer Vision*, 2016, S. 549–565

[CPK+16] Chen, L.-C. ; Papandreou, G. ; Kokkinos, I. ; Murphy, K. ; Yuille, A. L.: DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. In: *arXiv preprint arXiv:1606.00915* (2016)

[DDS+09] Deng, J. ; Dong, W. ; Socher, R. ; Li, L. ; Li, K. ; Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In: *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, S. 248–255

[EVGW+07] Everingham, M. ; Van Gool, L. ; Williams, C. K. I. ; Winn, J. ; Zisserman, A.: *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*. 2007

[EVGW+10] EVERINGHAM, M. ; VAN GOOL, L. ; WILLIAMS, C. K. I. ; WINN, J. ; ZISSERMAN, A.: The Pascal Visual Object Classes (VOC) Challenge. In: *International Journal of Computer Vision* 88 (2010), Nr. 2, S. 303–338

[EVGW+11] EVERINGHAM, M. ; VAN GOOL, L. ; WILLIAMS, C. K. I. ; WINN, J. ; ZISSERMAN, A.: *The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results*. 2011

[EVGW+12] EVERINGHAM, M. ; VAN GOOL, L. ; WILLIAMS, C. K. I. ; WINN, J. ; ZISSERMAN, A.: *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. 2012

[GBC16] GOODFELLOW, I. ; BENGIO, Y. ; COURVILLE, A.: *Deep Learning*. MIT Press, 2016

[GFK09] GOULD, S. ; FULTON, R. ; KOLLER, D.: Decomposing a scene into geometric and semantically consistent regions. In: *IEEE 12th International Conference on Computer Vision*, 2009, S. 1–8

[GGOEO+17] GARCIA-GARCIA, A. ; ORTS-ESCOLANO, S. ; OPREA, S. ; VILLENA-MARTINEZ, V. ; GARCIA-RODRIGUEZ, J.: A Review on Deep Learning Techniques Applied to Semantic Segmentation. In: *arXiv preprint arXiv:1704.06857* (2017)

[GLU12] GEIGER, A. ; LENZ, P. ; URTASUN, R.: Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In: *IEEE Conference on Computer Vision and Pattern Recognition*, 2012, S. 3354–3361

[HAB+11] HARIHARAN, B. ; ARBELÁEZ, P. ; BOURDEV, L. ; MAJI, S. ; MALIK, J.: Semantic contours from inverse detectors. In: *International Conference on Computer Vision*, 2011, S. 991–998

[HKH17] HONG, S. ; KWAK, S. ; HAN, B.: Weakly Supervised Learning with Deep Convolutional Neural Networks for Semantic Segmentation: Understanding Semantic Layout of Images with Minimum Human Supervision. In: *IEEE Signal Processing Magazine* 34 (2017), Nr. 6, S. 39–49

[HZ07] HOU, X. ; ZHANG, L.: Saliency Detection: A Spectral Residual Approach. In: *IEEE Conference on Computer Vision and Pattern Recognition*, 2007, S. 1–8

[HZRS15] He, K. ; Zhang, X. ; Ren, S. ; Sun, J.: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In: *IEEE International Conference on Computer Vision (ICCV)*, 2015, S. 1026–1034

[HZRS16] He, K. ; Zhang, X. ; Ren, S. ; Sun, J.: Deep Residual Learning for Image Recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, S. 770–778

[IS15] Ioffe, Sergey ; Szegedy, Christian: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: *arXiv preprint arXiv:1502.03167* (2015)

[KK11] Krähenbühl, P. ; Koltun, V.: Efficient inference in fully connected crfs with gaussian edge potentials. In: *Advances in neural information processing systems*, 2011, S. 109–117

[KSH12] Krizhevsky, A. ; Sutskever, I. ; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. In: *Advances in Neural Information Processing Systems*. 2012, S. 1097–1105

[LMB+14] Lin, T.-Y. ; Maire, M. ; Belongie, S. ; Hays, J. ; Perona, P. ; Ramanan, D. ; Dollár, P. ; Zitnick, C. L.: Microsoft COCO: Common Objects in Context. In: *European Conference on Computer Vision (ECCV)*. Springer International Publishing, 2014, S. 740–755

[LSD15] Long, J. ; Shelhamer, E. ; Darrell, T.: Fully Convolutional Networks for Semantic Segmentation. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, S. 3431–3440

[LYS+11] Liu, T. ; Yuan, Z. ; Sun, J. ; Wang, J. ; Zheng, N. ; Tang, X. ; Shum, H.: Learning to Detect a Salient Object. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33 (2011), Nr. 2, S. 353–367

[MP43] McCulloch, W. S. ; Pitts, W.: A logical calculus of the ideas immanent in nervous activity. In: *The bulletin of mathematical biophysics* 5 (1943), Nr. 4, S. 115–133

[Nie83] Niemann, H.: *Klassifikation von Mustern*. Springer Berlin Heidelberg, 1983

[OBK+17] Oh, S. J. ; Benenson, R. ; Khoreva, A. ; Akata, Z. ; Fritz, M. ; Schiele, B.: Exploiting Saliency for Object Segmentation from Image Level Labels. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, S. 5038–5047

[PCKF14] Papadopoulos, D. P. ; Clarke, A. D. F. ; Keller, F. ; Ferrari, V.: Training Object Class Detectors from Eye Tracking Data. In: *European Conference on Computer Vision (ECCV)*. Springer International Publishing, 2014, S. 361–376

[PKD15] Pathak, D. ; Krähenbühl, P. ; Darrell, T.: Constrained Convolutional Neural Networks for Weakly Supervised Segmentation. In: *The IEEE International Conference on Computer Vision (ICCV)*, 2015, S. 1796–1804

[PSLD15] Pathak, D. ; Shelhamer, E. ; Long, J. ; Darrell, T.: Fully Convolutional Multi-Class Multiple Instance Learning. In: *International Conference on Learning Representations (ICLR)* (2015)

[RDS+15] Russakovsky, O. ; Deng, J. ; Su, H. ; Krause, J. ; Satheesh, S. ; Ma, S. ; Huang, Z. ; Karpathy, A. ; Khosla, A. ; Bernstein, M. ; Berg, A. C. ; Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. In: *International Journal of Computer Vision* 115 (2015), Nr. 3, S. 211–252

[RLFF15] Russakovsky, O. ; Li, L. ; Fei-Fei, L.: Best of both worlds: Human-machine collaboration for object annotation. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, S. 2121–2131

[Ros58] Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. In: *Psychological Review* 65 (1958), Nr. 6, S. 386–408

[RTMF08] Russell, B. C. ; Torralba, A. ; Murphy, K. P. ; Freeman, W. T.: LabelMe: A Database and Web-Based Tool for Image Annotation. In: *International Journal of Computer Vision* 77 (2008), Nr. 1, S. 157–173

[SK14] Shotton, J. ; Kohli, P.: Semantic Image Segmentation. In: *Computer Vision: A Reference Guide*. Springer US, 2014, S. 713–716

[SLJ+15] Szegedy, C. ; Liu, W. ; Jia, Y. ; Sermanet, P. ; Reed, S. ; Anguelov, D. ; Erhan, D. ; Vanhoucke, V. ; Rabinovich, A.: Going Deeper with Convolutions. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, S. 1–9

[SLX15] SONG, S. ; LICHTENBERG, S. P. ; XIAO, J.: SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, S. 567–576

[Sun17] SUNDARARAJAN, D.: Segmentation. In: *Digital Image Processing: A Signal Processing and Algorithmic Approach*. Springer Singapore, 2017, S. 281–308

[SZ14] SIMONYAN, K. ; ZISSERMAN, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *arXiv preprint arXiv:1409.1556* (2014)

[TGJ+15] TOMPSON, J. ; GOROSHIN, R. ; JAIN, A. ; LECUN, Y. ; BREGLER, C.: Efficient Object Localization Using Convolutional Networks. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, S. 648–656

[WE07] WINN, J. ; EVERINGHAM, M.: *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) annotation guidelines*. 2007

[YCBL14] YOSINSKI, Jason ; CLUNE, Jeff ; BENGIO, Yoshua ; LIPSON, Hod: How transferable are features in deep neural networks? In: *Advances in Neural Information Processing Systems*. 2014, S. 3320–3328

[YK15] YU, F. ; KOLTUN, V.: Multi-Scale Context Aggregation by Dilated Convolutions. In: *arXiv preprint arXiv:1511.07122* (2015)

A

ERGEBNISSE DES DILATED NETWORKS BEI TRAINING MIT SCHWACHEN ANNOTATIONEN

Tabelle A.0.1: Verlauf der MIoU während des Trainings des Dilated Networks mit Bild-Level Annotationen (a) und Punktannnotationen (b). Bei der Verwendung von Punktannnotationen mit Objectness wurde die Objectness-Karte bei der Vorverarbeitung, wenn nötig, mittels Zero-Paddings (c) und mittels Reflection-Paddings (d) vergrößert. Das Training wurde mit unterschiedlichen Lernraten $\alpha \in \{10^{-3}, 10^{-4}, 10^{-5}\}$ und mit den in Abschnitt 5.2 beschriebenen Augmentierungen (gekennzeichnet mit aug.) durchgeführt. Die MIoU nach der Initialisierung mit vortrainierten Gewichten liegt bei 9,77 %.

Iteration	10.000	20.000	30.000	40.000	50.000	60.000
$\alpha = 10^{-3}$	15,62 %	15,24 %	15,56 %	14,78 %	14,88 %	14,68 %
$\alpha = 10^{-3}$, (aug.)	14,49 %	14,90 %	14,78 %	14,45 %	15,12 %	15,96 %
$\alpha = 10^{-4}$	15,36 %	16,25 %	16,45 %	17,25 %	16,77 %	16,85 %
$\alpha = 10^{-4}$, (aug.)	15,82 %	16,04 %	16,30 %	16,20 %	15,38 %	16,28 %
$\alpha = 10^{-5}$	18,32 %	16,83 %	16,28 %	16,38 %	16,71 %	16,66 %
$\alpha = 10^{-5}$, (aug.)	21,96 %	18,02 %	17,15 %	16,94 %	16,56 %	16,34 %

(a) Bild-Level-Annotationen

Iteration	10.000	20.000	30.000	40.000	50.000	60.000
$\alpha = 10^{-3}$	30,07 %	33,25 %	32,57 %	33,37 %	33,40 %	31,64 %
$\alpha = 10^{-3}$, (aug.)	22,78 %	24,57 %	24,25 %	23,86 %	23,96 %	25,84 %
$\alpha = 10^{-4}$	33,02 %	34,15 %	34,68 %	34,50 %	34,26 %	34,79 %
$\alpha = 10^{-4}$, (aug.)	27,56 %	27,71 %	26,95 %	27,13 %	27,25 %	27,03 %
$\alpha = 10^{-5}$	29,61 %	30,89 %	31,30 %	31,27 %	31,55 %	31,81 %
$\alpha = 10^{-5}$, (aug.)	28,43 %	29,83 %	29,17 %	28,89 %	28,46 %	28,14 %

(b) Punktannnotationen

Iteration	10.000	20.000	30.000	40.000	50.000	60.000
$\alpha = 10^{-3}$	33,67 %	32,98 %	31,74 %	31,46 %	31,27 %	31,26 %
$\alpha = 10^{-3}$, (aug.)	35,78 %	34,84 %	35,81 %	35,00 %	33,23 %	33,05 %
$\alpha = 10^{-4}$	33,47 %	32,03 %	32,36 %	32,41 %	32,29 %	32,69 %
$\alpha = 10^{-4}$, (aug.)	35,23 %	35,40 %	35,79 %	37,00 %	36,22 %	36,45 %
$\alpha = 10^{-5}$	31,68 %	32,62 %	32,41 %	32,17 %	31,63 %	31,48 %
$\alpha = 10^{-5}$, (aug.)	31,19 %	33,02 %	34,05 %	34,75 %	34,70 %	34,48 %

(c) Punktannnotationen & Objectness (Zero-Padding)

Iteration	10.000	20.000	30.000	40.000	50.000	60.000
$\alpha = 10^{-3}$	33,12 %	32,34 %	31,32 %	31,81 %	31,87 %	31,01 %
$\alpha = 10^{-3}$, (aug.)	36,49 %	35,92 %	35,49 %	35,85 %	33,90 %	33,05 %
$\alpha = 10^{-4}$	33,27 %	32,02 %	32,26 %	32,35 %	32,27 %	32,62 %
$\alpha = 10^{-4}$, (aug.)	35,28 %	35,33 %	35,80 %	36,84 %	36,29 %	36,42 %
$\alpha = 10^{-5}$	31,67 %	32,62 %	32,39 %	32,16 %	31,63 %	31,49 %
$\alpha = 10^{-5}$, (aug.)	31,16 %	32,99 %	34,04 %	34,78 %	34,66 %	34,48 %

(d) Punktannnotationen & Objectness (Reflection Padding)