technische universität
dortmund

fakultät für informatik

Master-Thesis

Convolutional Neural Networks auf Graphrepräsentationen von Bildern

Matthias Fey

26. Juli 2017

Gutachter:

Prof. Dr. Heinrich Müller

M.Sc. Jan Eric Lenssen

Lehrstuhl Informatik VII
Graphische Systeme
TU Dortmund

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Problemstellung 2

1.2 Aufbau der Arbeit 3

2 Grundlagen **5**

2.1 Mathematische Notationen 5

2.2 Graphentheorie 7

2.3 Convolutional Neural Networks 8

3 Graphrepräsentationen von Bildern **15**

3.1 Gitter 17

3.2 Superpixel 18

3.2.1 Verfahren 21

3.2.2 Merkmalsextraktion 25

4 Räumliches Lernen auf Graphen **29**

4.1 Grundlagen 29

4.2 Räumliche Faltung 31

4.3 Erweiterung auf Graphen im zweidimensionalen Raum 35

4.4 Netzarchitektur 38

5 Spektrales Lernen auf Graphen **41**

5.1 Spektrale Graphentheorie 41

5.1.1 Eigenwerte und Eigenvektoren reell symmetrischer Matrizen . 42

5.1.2 Laplace-Matrix 43

5.2 Spektraler Faltungsoperator 45

5.2.1 Graph-Fourier-Transformation 46

5.2.2 Spektrale Filterung 47

5.2.3 Polynomielle Approximation 48

5.3 Graph Convolutional Networks 50

5.4 Erweiterung auf Graphen im zweidimensionalen Raum 53
5.4.1 Partitionierung 55
5.4.2 Polynomielle Approximation über B-Spline-Kurven 58
5.5 Pooling auf Graphen 61
5.5.1 Graphvergröberung 62
5.5.2 Effizientes Pooling mittels binärer Bäume 63
5.5.3 Erweiterung auf Graphen im zweidimensionalen Raum 65
5.6 Netzarchitektur 66

6 Evaluation **69**
6.1 Merkmalsselektion 69
6.2 Versuchsaufbau 71
6.2.1 Datensätze 71
6.2.2 Metriken 78
6.2.3 Parameter 79
6.2.4 Augmentierung 80
6.2.5 Implementierung 81
6.3 Ergebnisse 83
6.4 Laufzeitanalyse 88
6.5 Diskussion 92

7 Ausblick **95**

Glossar **99**

Abbildungsverzeichnis **105**

Tabellenverzeichnis **107**

Algorithmenverzeichnis **109**

Literaturverzeichnis **111**

1 Einleitung

Convolutional Neural Networks, eine Spezialform der neuronalen Netze, gelten Dank ihrer effizienten Architektur und ihrer Fähigkeit, lokal stationäre Strukturen auf multiskalare, hierarchische Merkmale zu übertragen, als Allzweckmittel zur Lösung von Problemen in Bereichen der Bild-, Video- und Spracherkennung [8]. Sie sind jedoch nur im Kontext regulärer Datendomänen wohldefiniert. So lassen sich aber insbesondere zahlreiche Probleme als maschinelles Lernen auf irregulär strukturierten Daten verstehen. Dabei liefern Graphen eine geeignete generische Datenstruktur, die es ermöglicht, eine Vielzahl dieser Probleme wie etwa in Bereichen der sozialen Netzwerke, der Biologie, Chemie oder Telekommunikation zu modellieren [44]. Es ist daher nicht verwunderlich, dass in der Vergangenheit zahlreiche Anstrengungen unternommen wurden, den Faltungsoperator der Convolutional Neural Networks für die Anwendung auf irregulären Graphstrukturen zu generalisieren [8, 26, 36]. Diese Verfahren lassen sich dabei in zwei unterschiedliche Herangehensweisen gliedern:

1. Das *räumliche Lernen auf Graphen* widmet sich der räumlichen Verschiebung eines Filters auf den Knoten eines Graphen ähnlich zu der Verschiebung eines Filters auf einem zweidimensionalen Bild oder einem eindimensionalen Audiosignal [36]. Dafür wird zu einem Knoten eine feste sowie eindeutige Nachbarschaft definiert und ein gemeinsamer Filter auf allen Nachbarschaften einer Knotenauswahl angewendet.

2. Das *spektrale Lernen auf Graphen* basiert auf der spektralen Graphentheorie und formuliert einen Faltungsoperator auf Graphen über die Zerlegung seiner Knotenmerkmale in das Spektrum des Graphen ähnlich zu der Transformation eines Signals in dessen Frequenzraum mittels der klassischen Fourier-Transformation. Eine Multiplikation der Knotenmerkmale in der spektralen Domäne entspricht damit einer Faltung in der Domäne des Graphen.

Beide Herangehensweisen zeichnen sich im Gegensatz zur klassischen Faltung der Convolutional Neural Networks auf regulären Strukturen durch ihre (notwendigerweise) rotationsinvariante Faltung aus. In dieser Arbeit präsentieren wir daher zwei

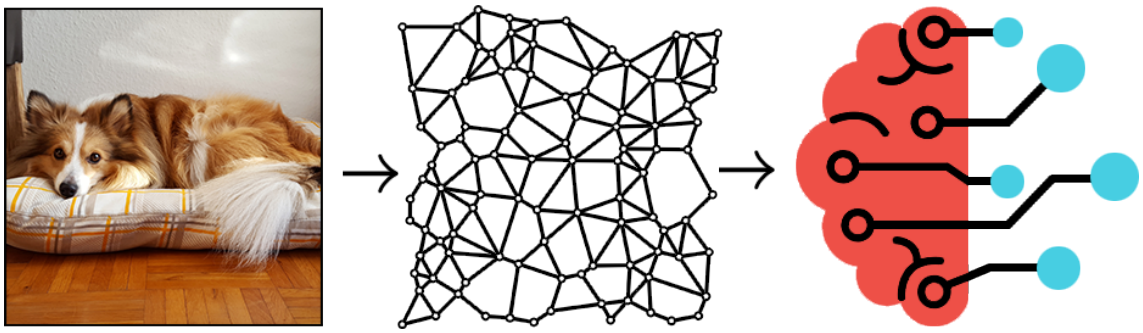


Abbildung 1.1: Illustration der verfolgten Problemstellung in dieser Arbeit. Bilder werden für ihre Eingabe in ein neuronales Netz zuvor in eine korrespondierende Graphrepräsentationen konvertiert.

Ansätze bezüglich einer rotationsvarianten Faltung auf Graphen, bei denen Knoten eindeutige Positionen im zweidimensionalen euklidischen Raum zugeordnet sind.

1.1 Problemstellung

Die vorliegende Arbeit beschäftigt sich mit dem maschinellen Lernen mittels neuronaler Netze bzw. Convolutional Neural Networks auf Graphrepräsentation von Bildern. Als eine *Graphrepräsentation eines Bildes* verstehen wir dabei die Abbildung eines Bildes in einen Graphen, welcher das korrespondierende Bild mittels einer Menge von Knoten und Verbindungen von Knotenpaaren beschreibt, sodass signifikante Merkmale des Bildes erhalten bleiben. Knoten des Graphen müssen dabei nicht für alle Pixel eines Bildes existieren, sondern können auch ganze Bildbereiche abdecken. Das bietet sich insbesondere dann an, wenn das vorliegende Bild zu großen Teilen aus Hintergrundbereichen besteht, welche für das Training eines neuronalen Netzes lediglich mehr Aufwand, aber keinen Mehrwert bedeuten. Damit kann sich folglich für Bilder eine nicht unerhebliche Datenreduktion ergeben, die unter anderem das Potential besitzt, schnellere Ausführungszeiten eines neuronalen Netzes zu garantieren. Dabei wird das Bild folglich von einer regulären Gitterstruktur in eine irreguläre Graphstruktur übertragen (vgl. Abbildung 1.1). Das Lernen auf solch einer Graphstruktur ist eine sowohl theoretisch als auch implementierungtechnisch herausfordernde Aufgabe, denn schließlich kann der lediglich auf regulären Strukturen definierte klassische Faltungsoperator der Convolutional Neural Networks nicht genutzt werden. Vorangegangene Arbeiten zeigen, dass eine Generalisierung des Faltungsoperators auf irregulären Datendomänen zwar möglich ist [8, 26, 36], sich diese aber letztendlich aufgrund ihrer recht strengen Generalisierung auf willkürliche Graphen ohne Positionsberücksichtigung der Knoten und ihrer in Folge dessen innewoh-

nenden Rotationsinvarianz für die beschriebende Problemstellung als unzureichend herausstellt. In dieser Arbeit werden daher für die Ansätze des räumlichen sowie des spektralen Lernens auf Graphen weiterführende Methodiken entwickelt, die es ermöglichen, eine Faltung auf Graphen zu realisieren, sodass deren Knotenpositionen und Kantenausrichtungen (wie oben, unten, links oder rechts) analog zu einer Faltung auf Bildern berücksichtigt werden. Wir zeigen, dass wir mit Hilfe der entwickelten Methodiken im Kontext von Graphen, deren Kanten eine Orientierung zu Grunde liegt, die bisherigen Faltungsoperatoren auf Graphen signifikant verbessern können. Die vorgestellten Ansätze sind dabei für Graphen, die aus einem Gitter gewonnen wurden, äquivalent zu der klassischen Faltung auf Bildern und besitzen dabei weiterhin den Vorteil, auch auf irreguläre Datendomänen im euklidischen Raum angewendet werden zu können. Obwohl das Lernen auf den Graphrepräsentationen der Bilder bezüglich ihrer Genauigkeit und Laufzeit (noch) nicht ganz mit den klassischen Verfahren auf den Rohdaten der Bilder konkurrieren kann, so zeigen sich die in dieser Arbeit vorgestellten Ansätze dennoch aufgrund ihrer Datenreduktion und Ausweitung der Anwendungsgebiete als lohnenswert für zukünftige Arbeiten.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich, neben den in Kapitel 2 vorgestellten Grundlagen, die für das weitere Verständnis der Arbeit von Nöten sind, in vier Bereiche.
Das Kapitel 3 widmet sich der Gewinnung einer Graphrepräsentation aus einem Bild. Dabei werden zwei Verfahren zur Generierung eines Graphen aus einem Bild vorgestellt: die herkömmliche Gitterrepräsentation eines Bildes (Kapitel 3.1) sowie die Gewinnung eines Graphen aus einer vorberechneten Superpixelrepräsentation (Kapitel 3.2).
Kapitel 4 und 5 erläutern die beiden unterschiedlichen Ansätze des graphbasierten Lernens in neuronalen Netzen, das räumliche und das spektrale Lernen, getrennt voneinander. Beide Verfahren umfassen den aktuellen Stand der wissenschaftlichen Forschung auf diesem Gebiet. Zu jedem Ansatz finden sich ebenso die entwickelten Erweiterungen bezüglich der beschriebenen Problemstellung.
Kapitel 6 evaluiert die vorgestellten sowie entwickelten Ansätze anhand der gegebenen Problemstellung im Vergleich zueinander sowie im Vergleich zu klassischen Lösungen auf diesem Gebiet. Dazu wird zunächst in Unterkapitel 6.2 der Versuchsaufbau erläutert und dessen Ergebnisse in den Unterkapiteln 6.3 und 6.4 präsentiert. In Kapitel 7 folgt ein abschließender Ausblick zu möglichen weiterführenden Arbeiten.

2 Grundlagen

Das folgende Kapitel erläutert die Grundlagen und Notationen, die zum weiteren Verständnis der Arbeit benötigt werden. Zunächst werden in Unterkapitel 2.1 die grundlegenden Notationen zu Mengen, Vektoren, Matrizen und Tensoren definiert, die im weiteren Verlauf verwendet werden. Daraufhin folgt in Unterkapitel 2.2 eine kurze Einführung in das Gebiet der Graphentheorie. Abschließend führt das Unterkapitel 2.3 in das Gebiet der neuronalen Netze und insbesondere der Convolutional Neural Networks ein, auf denen diese Arbeit aufbaut. Für einen umfassenderen Blick in das Gebiet des Deep-Learnings, den diese Arbeit nicht leisten kann, sei auf Nielsen [35] verwiesen.

2.1 Mathematische Notationen

Eine *ungeordnete Menge* $\mathcal{A} \coloneqq \{a_1, \ldots, a_N\} = \{a\}_{n=1}^{N}$, $N \in \mathbb{N}$, beschreibt eine Gruppierung von N Elementen a_n mit $1 \leq n \leq N$, wobei $|\mathcal{A}| = N$ die *Kardinalität* von $\mathcal{A}$ genannt wird. Eine Menge $\mathcal{A} \coloneqq (a_1, \ldots, a_N) = (a_n)_{n=1}^{N}$ heißt *geordnet*, wenn auf ihren Elementen eine reflexive, transitive und antisymmetrische Ordnung $\mathcal{A} \times \mathcal{A}$ definiert ist, wobei die Reihenfolge der Elemente in $(a_1, \ldots, a_N)$ die Ordnung beschreibt [21].
Sei $\mathbf{v} \in \mathbb{R}^N$, $N \in \mathbb{N}$, mit $\mathbf{v} = [v_1, \ldots, v_n]^\top$ ein *Vektor* mit N reellen Elementen $\{v_n\}_{n=1}^{N}$, $v_n \in \mathbb{R}$, wobei $\mathbf{v}_n = v_n$ das n-te Element von $\mathbf{v}$ referenziert. Zu zwei Vektoren $\mathbf{v}, \mathbf{w} \in \mathbb{R}^N$ ist das *Skalarprodukt* $\langle \mathbf{v}, \mathbf{w} \rangle$ definiert als $\langle \mathbf{v}, \mathbf{w} \rangle \coloneqq \sum_{n=1}^{N} \mathbf{v}_n \mathbf{w}_n$ [21]. $\mathbf{v}$ und $\mathbf{w}$ stehen *orthogonal* zueinander, d.h. $\mathbf{v} \perp \mathbf{w}$, genau dann, wenn $\langle \mathbf{v}, \mathbf{w} \rangle = 0$ [21]. Die *Länge* eines Vektors $\mathbf{v} \in \mathbb{R}^N$ ist definiert über die *euklidische Norm*

$$\|\mathbf{v}\|_2 \coloneqq \sqrt{\sum_{n=1}^{N} \mathbf{v}_n^2}.$$

Eine *Matrix* $\mathbf{M} \in \mathbb{R}^{N \times M}$, $N, M \in \mathbb{N}$, erweitert einen Vektor um M Spalten. Der Wert $\mathbf{M}_{nm} \in \mathbb{R}$ beschreibt damit das Element in der n-ten Zeile und m-ten Spalte von $\mathbf{M}$. Die *transponierte Matrix* $\mathbf{M}^\top \in \mathbb{R}^{M \times N}$ ist eine Matrix, die aus $\mathbf{M} \in \mathbb{R}^{N \times N}$

durch Vertauschen der Zeilen und Spalten entsteht, d.h. $\mathbf{M}^{\top}_{mn} = \mathbf{M}_{nm}$ [21]. Zu einem Vektor $\mathbf{v} \in \mathbb{R}^N$ lässt sich weiterhin dessen korrespondiere quadratische *Diagonalmatrix* $\operatorname{diag}(\mathbf{v}) \in \mathbb{R}^{N \times N}$ über

$$\operatorname{diag}(\mathbf{v}) \coloneqq \begin{bmatrix} \mathbf{v}_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{v}_N \end{bmatrix}$$

bzw. $\operatorname{diag}(\mathbf{v})_{nn} \coloneqq \mathbf{v}_n$ definieren [8].

Dünnbesetzte Matrizen. Eine *dünnbesetzte* oder *schwachbesetzte Matrix* ist eine Matrix, bei der so viele Einträge aus Nullen bestehen, dass sich statt der üblichen Speicherung einer Matrix als zweidimensionales Feld speichereffizientiere Datenstrukturen ergeben. In der Regel gilt eine Matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$ als dünnbesetzt, wenn diese nicht aus mehr als N oder $N \log N$ Einträgen ungleich Null besteht. Neben dem Speichergewinn lassen sich viele Operationen auf Matrizen ebenso berechnungseffizienter implementieren [43]. So muss zum Beispiel zur Bestimmung des größten Elements einer Matrix nicht die komplette Matrix, sondern lediglich deren explizit eingetragene Werte betrachtet werden. Es gibt jedoch auch Operationen, die nicht auf dünnbesetzten Matritzen definiert sind, sodass diese vorher in eine *dichte* Matrix überführt werden müssen (vgl. [43]). Im Laufe dieser Arbeit haben wir es oft mit dünnbesetzten Matrizen zu tun. So wird zum Beispiel eine Diagonalmatrix *immer* als eine dünnbesetzte Matrix implementiert.

Tensoren. Ein *Tensor* $\mathbf{T} \in \mathbb{R}^{N_1 \times \cdots \times N_R}$, $N_1, \ldots, N_R \in \mathbb{N}$, ist ein mathematisches Objekt, welches das Konzept der Vektoren und Matrizen auf beliebig viele Dimensionen erweitert. Die Anzahl der Dimensionen R eines Tensors wird auch *Rang* genannt [21]. Vektoren können damit insbesondere als Tensor mit Rang 1 sowie Matrizen als Tensor mit Rang 2 verstanden werden. Ein Tensor mit Rang 0 beschreibt ein Skalar. Aus einem Tensor $\mathbf{T} \in \mathbb{R}^{N_1 \times \cdots \times N_R}$ können über die Indexnotation Tensoren mit geringerer Dimension gefiltert werden. So beschreibt $\mathbf{T}_{n_1 \cdots n_R}$ einen Tensor mit Rang 0 bzw. das Element des Tensors an Position $(n_1, \ldots, n_R)$. Analog beschreibt zum Beispiel $\mathbf{T}_{n_1} \in \mathbb{R}^{N_2 \times \cdots \times N_R}$ einen Tensor mit Rang $R - 1$, bei dem die erste Dimension über n_1 festgehalten wird.

Bilder. Ein *Bild* kann folglich durch einen dreidimensionalen Tensor $\mathbf{B} \in \mathbb{R}^{H \times W \times C}$ repräsentiert werden, wobei $H, W \in \mathbb{N}$ die Höhe bzw. Breite des Bildes angeben

und $C \in \{1, 3\}$ die Anzahl der Farbkanäle des Bildes beschreibt, d.h. ein Graubild mit nur einem Kanal oder ein Farbbild mit drei Kanälen (zum Beispiel über das RGB- oder Lab-Farbmodell [40]). Ein *Pixel* eines Bildes $\mathbf{B}$ an der Position (x, y) mit $1 \leq x \leq W, 1 \leq y \leq H$, wird durch den Wert $\mathbf{B}_{yx} \in \mathbb{R}^C$ beschrieben.

2.2 Graphentheorie

Ein *Graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ bezeichnet eine endliche Menge $\mathcal{V} = \{v_n\}_{n=1}^N$ von $N \in \mathbb{N}$ Knoten, $|\mathcal{V}| = N < \infty$, zusammen mit einer Menge geordneter Knotenpaare bzw. Kanten $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ [4]. Seien $v_i, v_j \in \mathcal{V}$ im Folgenden zwei beliebige Knoten. Falls $(v_i, v_j) \in \mathcal{E}$, dann ist v_j *adjazent* zu v_i [4]. Zu einem Graphen $\mathcal{G}$ definiert die *Nachbarschaftsfunktion* $\mathcal{N}(v_i) \subseteq \mathcal{V}$ über $\mathcal{N}(v_i) \coloneqq \{v_j \mid (v_i, v_j) \in \mathcal{E}\}$ die Nachbarschaftsmenge von v_i [44].

Ein *gewichteter Graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ ist ein Graph, der zusätzlich eine *Gewichtsfunktion* $w \colon \mathcal{V} \times \mathcal{V} \to \mathbb{R}_+$ auf den Kanten des Graphen definiert, sodass $(v_i, v_j) \notin \mathcal{E}$ genau dann, wenn $w(v_i, v_j) = 0$ [4]. Im Falle eines ungewichteten Graphen ist die Gewichtsfunktion w implizit durch $\mathcal{E}$ über $w \colon \mathcal{V} \times \mathcal{V} \to \{0, 1\}$ gegeben.

Ein Graph heißt *ungerichtet*, falls $w(v_i, v_j) = w(v_j, v_i)$ [4]. Als *Schleife* wird eine Kante bezeichnet, die einen Knoten mit sich selbst verbindet, d.h. $w(v, v) > 0$ für einen Knoten $v \in \mathcal{V}$. Ein Graph ohne Schleifen wird *schleifenloser Graph* genannt [4]. Für den weiteren Verlauf dieser Arbeit fordern wir, solange nicht explizit anders angegeben, gewichtete, ungerichtete sowie schleifenlose Graphen.

Ein Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ kann weiterhin eindeutig über dessen (in der Regel dünnbesetzte) *Adjazenzmatrix* $\mathbf{A} \in \mathbb{R}_+^{N \times N}$ mit $\mathbf{A}_{ij} \coloneqq w(v_i, v_j)$ definiert werden [8]. Als *Grad* eines Knotens $v \in \mathcal{V}$ wird die Anzahl der Knoten bezeichnet, die adjazent zu ihm sind, d.h. $\deg(v) \coloneqq |\mathcal{N}(v)|$. Im Falle von gewichteten Graphen wird der Grad eines Knotens von $v_i \in \mathcal{V}$ auch oft über $d(v_i) \coloneqq \sum_{j=1}^N \mathbf{A}_{ij}$ definiert [8]. Die unterschiedliche Notation macht deutlich, welcher Grad eines Knotens gemeint ist. Die *Gradmatrix* $\mathbf{D} \in \mathbb{R}_+^{N \times N}$ eines Graphen $\mathcal{G}$ ist dann definiert als Diagonalmatrix $\mathbf{D} \coloneqq \operatorname{diag}\big([d(v_1), \ldots, d(v_N)]^\top\big)$ bzw. $\mathbf{D}_{ii} = d(v_i)$ [8]. Ein Knoten $v \in \mathcal{V}$ heißt *isoliert*, falls dieser keinen Nachbarknoten besitzt, d.h. $\deg(v) = 0$ [8].

Ein *Weg* der Länge K auf $\mathcal{G}$ ist eine Folge von Knoten $\big(v_{x(1)}, v_{x(2)}, \ldots, v_{x(K)}\big)$, sodass $\big(v_{x(k)}, v_{x(k+1)}\big) \in \mathcal{E}$ für alle $1 \leq k < K$, wobei $x \colon \{1, \ldots, K\} \to \{1, \ldots, N\}$ eine Abbildung auf den Indizes der Knoten $\{v_n\}_{n=1}^N$ [4]. Ein *Pfad* ist ein Weg mit der Bedingung, dass $v_{x(k)} \neq v_{x(k+1)}$. Im Kontext von schleifenlosen Graphen sind die Begriffe Weg und Pfad äquivalent. Wir schreiben $s(v_i, v_j)$ mit Hilfe der *Abstands-*

funktion $s\colon \mathcal{V} \times \mathcal{V} \to \mathbb{N} \cup \{\infty\}$ für die Länge des kürzesten Pfades von v_i nach v_j, d.h. die minimale Anzahl an Kanten, die zwischen v_i und v_j liegen [17]. v_j ist von v_i aus *erreichbar*, wenn $s(v_i, v_j) \in \mathbb{N}$ [4]. Falls v_j nicht von v_i aus erreichbar ist, so gilt $glss(v_i, v_j) = \infty$. Die Relation der Erreichbarkeit in der Knotenmenge $\mathcal{V}$ eines Graphen ist eine Äquivalenzrelation. Die Äquivalenzklassen der Erreichbarkeitsrelation heißen die *Zusammenhangskomponenten* des Graphen $\mathcal{G}$ [4]. Wir nennen $\mathcal{G}$ *zusammenhängend*, wenn $\mathcal{G}$ genau eine Zusammenhangskomponente besitzt, d.h. zu jedem Knoten v_i existiert mindestens ein Weg zu jedem anderen Knoten $v_j \in \mathcal{V}$ [17]. Es lässt sich weiter die Nachbarschaftsfunktion $\mathcal{N}$ zu einer *K-lokalisierten Nachbarschaftsfunktion* $\mathcal{N}_K \subseteq \mathcal{V}$ mit $\mathcal{N}_K(v_i) \coloneqq \{v_j | s(v_i, v_j) \leq K\}$ generalisieren [17].
Aus einem Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ kann weiterhin ein *Teilgraph* $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ mit verkleinerter Knotenmenge $\mathcal{V}' \subseteq \mathcal{V}$ gewonnen werden, indem dessen Kantenrelation $\mathcal{E}' = \{(v_i, v_j) \in \mathcal{E} \mid v_i, v_j \in \mathcal{V}'\}$ nur die Kanten in $\mathcal{E}$ derjenigen Knoten enthält, die in $\mathcal{V}'$ liegen [36].
Eine Funktion $f\colon \mathcal{V} \to \mathbb{R}^M$ auf den Knoten eines Graphen $\mathcal{G}$, die auf einen M-dimensionalen Vektor abbildet, heißt *Merkmalsfunktion* der Knotenmenge. Eine Merkmalsfunktion f kann ebenso als *Merkmalsmatrix* $\mathbf{F} \in \mathbb{R}^{N \times M}$ mit $\mathbf{F}_{im} \coloneqq f(v_i)_m$ interpretiert werden [44]. Bildet f lediglich auf ein einziges Element ab, d.h. $M = 1$ und folglich $f\colon \mathcal{V} \to \mathbb{R}$, ergibt sich daraus analog ein *Merkmalsvektor* $\mathbf{f} \in \mathbb{R}^N$ mit $\mathbf{f}_i \coloneqq f(v_i)$.

2.3 Convolutional Neural Networks

Neuronale Netze bzw. Deep-Learning gehören zu den derzeit besten und beliebtesten Lösungen zu Problemen der Bild- oder Spracherkennung [35]. Dabei lernt bzw. approximiert das Netz durch eine Anpassung ihrer Parameter über einer Menge an Trainingsbeispielen eine Funktion, sodass die Trainingsbeispiele auf ihre gewünschte Ausgabe abbilden und auch für unbekannte Eingaben zuverlässige Vorhersagen getroffen werden können. Neuronale Netze sind daher größtenteils in dem Bereich des *überwachten maschinellen Lernens* anzuordnen. Ein Netz, welches lediglich die Trainingsmenge lernt, aber über der Trainingsmenge hinausgehende Eingaben nicht gut beschreiben kann, wird als ein *überangepasstes* (engl. *overfitted*) Netz bezeichnet [35].
Ein *neuronales Netz* besteht aus einer beliebigen Anzahl miteinander verbundener *Neuronen*. Neuronen sind üblicherweise mit anderen Neuronen in sequentiellen *Schichten* bzw. *Ebenen* angeordnet. Die erste Schicht eines neuronalen Netzes wird

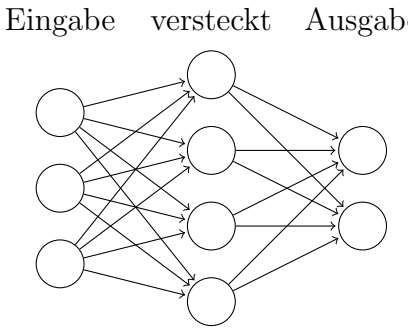


Abbildung 2.1: Beispiel eines Feedforward-Netzes mit drei vollverbundenen Schichten von einer Eingabe mit drei Neuronen zu einer Ausgabe mit zwei Neuronen und einer dazwischenliegenden versteckten Schicht.

als *Eingabe-* und die letzte Schicht als *Ausgabeschicht* bezeichnet. Schichten zwischen Ein- und Ausgabe heißen *versteckt* (engl. *hidden*). Als *Deep-Learning* wird ein Netz mit mindestens zwei versteckten Schichten verstanden. Die einfachste Form eines neuronalen Netzes ist das *Feedforward*-Netz, bei der jedes Neuron einer Schicht mit allen Neuronen der darauffolgenden Schicht verbunden ist. Die Schichten eines Feedforward-Netzes werden deshalb auch als *vollverbunden* (engl. *fully-connected*) betitelt. Abbildung 2.1 zeigt ein Beispiel eines solchen Netzes mit drei Schichten. Andere Netzvarianten erlauben zum Beispiel Schleifen, Rückwärtskanten oder das Überspringen einer Schicht [35].

Ein Neuron besitzt genau einen reellen Wert, der sich aus den Neuronen der vorherigen Schicht erschließt. Die t-te Neuronenschicht lässt sich folglich als ein Vektor $\mathbf{x}^{(t)} \in \mathbb{R}^{N^{(t)}}$ auffassen, wobei $N^{(t)} \in \mathbb{N}$ die Anzahl der Neuronen in der t-ten Schicht beschreibt. Zu jeder Kante existiert zusätzlich ein Gewicht, welches den Anteil des Neurons zu dessen verbundenen Neuron angibt. Damit lassen sich die Neuronenwerte der $(t+1)$-ten Schicht über

$$\mathbf{x}^{(t+1)} := \mathbf{W}^{(t+1)}\mathbf{x}^{(t)}$$

definieren, wobei $\mathbf{W}^{(t+1)} \in \mathbb{R}^{N^{(t+1)} \times N^{(t)}}$ eine *Gewichtsmatrix* der Kanten beschreibt, sodass $\mathbf{W}_{ji}^{(t+1)} \in \mathbb{R}$ das Gewicht der Kante des i-ten Neurons in der t-ten Schicht zu dem j-ten Neuron der $(t+1)$-ten Schicht angibt. Zusätzlich zu den Gewichten existert zu jedem Neuron in der t-ten Schicht außer der Eingabeschicht ein *Bias* $\mathbf{b}^{(t)} \in \mathbb{R}^{N^{(t)}}$. Mit einer elementweisen Anwendung einer nicht-linearen *Aktivierungsfunktion* $\sigma\colon \mathbb{R} \to \mathbb{R}$ ergibt sich damit die finale Version der Neuronenwerte der

$(t+1)$-ten Schicht als

$$\mathbf{x}^{(t+1)} \coloneqq \sigma\Big(\mathbf{W}^{(t+1)}\mathbf{x}^{(t)} + \mathbf{b}^{(t+1)}\Big).$$

Als Aktivierungsfunktion kommt dabei beispielsweise die nicht-lineare *Sigmoidfunktion* $\operatorname{sig}(z) \coloneqq 1/(1+\exp(-z))$ oder die *Rectified Linear Unit (ReLU)*-Funktion $\operatorname{ReLU}(z) \coloneqq \max(z, 0)$ zum Einsatz [35]. Die Menge der Gewichte $\mathcal{W} \coloneqq \left\{\mathbf{W}^{(t)}\right\}_{t=2}^{T}$ sowie die Menge der Biaswerte $\mathcal{B} \coloneqq \left\{\mathbf{b}^{(t)}\right\}_{t=2}^{T}$ der $T \in \mathbb{N}$ vielen Schichten werden die *Parameter* des Netzes genannt, über dessen Anpassung das Netz trainiert wird. Diese Werte werden dabei sequentiell über einer kleinen Eingabemenge $\mathcal{X}$ mit den Eingaben $\mathbf{x} \in \mathcal{X}$ so angepasst, dass eine *Kostenfunktion* minimiert wird. Die Größe der Menge $\mathcal{X}$ wird dabei als *Batch-Size* betitelt [35]. Ein Beispiel einer Kostenfunktion ist die *quadratische Kostenfunktion*, die über

$$C(\mathcal{X}, \mathcal{W}, \mathcal{B}) \coloneqq \frac{1}{2|\mathcal{X}|} \sum_{\mathbf{x}\in\mathcal{X}} \|y(\mathbf{x}) - \hat{y}(\mathbf{x})\|_2^2 \tag{2.1}$$

definiert ist, wobei $y\colon \mathbb{R}^{N^{(1)}} \to \mathbb{R}^{N^{(T)}}$ bzw. $\hat{y}\colon \mathbb{R}^{N^{(1)}} \to \mathbb{R}^{N^{(T)}}$ die Ausgabe bzw. die erwartete Ausgabe des Netzes bezüglich $\mathbf{x}$ beschreibt [35]. Die implizit durch das Netz gegebenen Werte $\mathcal{W}$ und $\mathcal{B}$ werden dabei oft weggelassen, sodass wir lediglich $C(\mathcal{X})$ schreiben. C zeichnet sich dabei als Kostenfunktion aus, denn sie ist für alle Eingaben stets positiv und wird umso kleiner, je ähnlicher sich $y(\mathbf{x})$ und $\hat{y}(\mathbf{x})$ werden. Die Anpassung der Parameter des Netzes erfolgt über die sukzessive Eingabe einer Menge $\mathcal{X}$ in das Netz und der Berechnung eines negativen Gradienten in Richtung des steilsten Abstieges von C. Formal betrachtet entspricht der Gradient der Kostenfunktion C dem Vektor der partiellen Ableitungen

$$\nabla C \coloneqq \left[\frac{\partial C}{\partial w_i}, \ldots, \frac{\partial C}{\partial b_j}, \ldots\right]^\top$$

aller Gewichte w_i und aller Biaswerte b_j des Netzes [35]. Durch die Anpassung der Gewichte und Biaswerte des Netzes über

$$w_i \to w_i' = w_i - \gamma\frac{\partial C}{\partial w_i} \qquad \text{und} \qquad b_j \to b_j' = b_j - \gamma\frac{\partial C}{\partial b_j}$$

wird die Kostenfunktion C minimiert, wobei $\gamma \in \mathbb{R}_+$ der *Lernrate*, d.h. der gewählten Schrittweite, entlang des negativen Gradienten entspricht [35]. Üblicherweise wird dabei γ so gewählt, dass das Netz nicht zu langsam trainiert, aber dennoch eine gute

Approximation erreicht wird und erfordert in der Regel eine manuelle Anpassung. Die Berechnung des Gradienten ∇C wird dabei aus Berechnungsgründen nicht für alle, sondern lediglich für Eingaben einer zufällig ausgewählten Untermenge von $\mathcal{X}$ durchgeführt und daraus dessen Durchschnitt gebildet, welcher stochastisch gesehen in etwa dem realen Gradienten über allen Eingaben entspricht (*stochastischer Gradientenabstieg*) [35]. Zur Berechnung des Gradienten der Kostenfunktion C kommt dabei der effiziente *Backpropagation*-Algorithmus zum Einsatz, der es ermöglicht die optimierten Werte der Parameter nach einem Trainingsschritt zu ermitteln, indem der Fehler zwischen der Ausgabe des Netzes und der erwarteten Ausgabe über die Ausgabe- zur Eingabeschicht zurück propagiert wird [41]. Aus den Fehlern der einzelnen Neuronen kann damit schließlich der Gradient ∇C berechnet werden (vgl. [35]).

Der Prozess des Trainings eines neuronalen Netzes, d.h. die Anpassung seiner Parameter zur Minimierung der Kostenfunktion, wird dabei solange wiederholt, bis ∇C ein lokales Minimum erreicht. Ein einmaliges Durchlaufen aller Eingaben der Eingabemenge wird dabei als *Epoche* bezeichnet [35].

Eine Spezialform eines neuronalen Netz ist das *Convolutional Neural Network (CNN)*, welches hauptsächlich in der Bild- und Spracherkennung zum Einsatz kommt und seinen Namen aufgrund der *Faltung* seiner Eingabe mit einem *Filter* erhält. [29]. Entgegen eines Feedforward-Netzes, welches stets einen Vektor als Eingabe erwartet, arbeitet ein CNN auf den originalen Bilddaten, d.h. einer Menge zweidimensionaler Matrizen, und ermöglicht so, dass dessen räumliche Struktur beim Lernen berücksichtigt wird [35]. Dabei wird ein Neuron nicht mehr mit allen Neuronen der vorherigen Schicht, sondern nur noch mit einer Teilmenge dieser innerhalb eines Fensters um das Neuron verbunden. Dieses Fenster wird auch oft, angelehnt an die Rezeptoren eines Auges, *rezeptives Feld* (engl. *Receptive-Field*) genannt [29]. Die Größe des Receptive-Fields, zum Beispiel 5×5, wird dabei als *Filtergröße* bezeichnet. Abbildung 2.2 veranschaulicht die Vernetzung der Neuronen durch ein Receptive-Field. Eine Besonderheit des CNNs ist weiterhin, dass sich die Receptive-Fields ihre Gewichte und Biaswerte teilen. Damit erlernt ein Receptive-Field ein Merkmal, zum Beispiel eine Kante oder eine Textur, und findet dieses Merkmal auch in anderen Receptive-Fields, nur an unterschiedlichen Positionen [35]. Das ermöglicht insbesondere das Erlernen translationsinvarianter Merkmale, die für eine Bilderkennung fundamental sind. Aus diesem Grund spricht man auch häufig von einer *Merkmalskarte* (engl. *Feature-Map*), die durch eine Faltung über den Receptive-Fields entsteht. Um mehrere Merkmale eines Receptive-Fields zu generieren, wird dabei üblicherweise

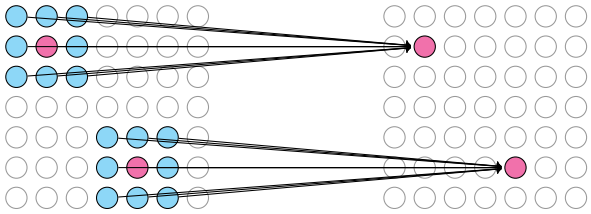


Abbildung 2.2: Illustration der Neuronenverbindungen in einem CNN bei einer Filtergröße von 3×3, hier am Beispiel von zwei Neuronen (rot). Der Wert eines Neurons berechnet sich aus allen Neuronen innerhalb eines Receptive-Fields um das Neuron der vorherigen Schicht (blau).

eine Reihe von zusammengehörigen *Eingabekarten* auf eine neue Menge von *Ausgabekarten* projiziert. Ein Bild $\mathbf{B} \in \mathbb{R}^{H \times W \times 3}$ mit drei Farbkanälen beschreibt zum Beispiel drei Merkmalskarten, die als Eingabe in das CNN dienen.

Formal lässt sich damit die *zweidimensionale Faltungsoperation*

$$\text{conv2d} \colon \mathbb{R}^{H \times W \times M_\text{in}} \to \mathbb{R}^{H \times W \times M_\text{out}},$$

im Folgenden auch oft als *klassische Faltungsoperation* betitelt, als

$$\text{conv2d}(\mathbf{B})_{yxm} \coloneqq \sum_{\Delta y=1}^{F_y} \sum_{\Delta x=1}^{F_x} \sum_{m_\text{in}=1}^{M_\text{in}} \mathbf{B}_{y+\lceil F_y/2 \rceil - \Delta y, x+\lceil F_x/2 \rceil - \Delta x, m_\text{in}} \mathbf{W}_{\Delta y, \Delta x, m_\text{in}, m}$$

mit $1 \leq y \leq H$, $1 \leq x \leq W$ und $1 \leq m \leq M_\text{out}$ definieren, wobei $\mathbf{B} \in \mathbb{R}^{H \times W \times M_\text{in}}$ die Eingabe und $\mathbf{W} \in \mathbb{R}^{F_y \times F_x \times M_\text{in} \times M_\text{out}}$ den Filter bzw. Gewichtstensor der Faltungsoperation mit Filtergröße $F_y \times F_x$ von $M_\text{in} \in \mathbb{N}$ Eingabekarten auf $M_\text{out} \in \mathbb{N}$ Ausgabekarten beschreibt [1]. Auf die Aufsummierung eines Bias $\mathbf{b} \in \mathbb{R}^{M_\text{out}}$ sowie die Anwendung der Aktivierungsfunktion σ auf conv2d wurde hier zur Vereinfachung verzichtet. Der Wert eines Indexzugriffs auf die Eingabekarte $\mathbf{B}$, der über die Grenzen von $\mathbf{B}$ hinausgeht, wird üblicherweise auf Null gesetzt (*Zero-Padding*), sodass dieser keinen Einfluss auf die Faltung nimmt und die Ausgabekarte damit die gleiche Höhe und Breite der Eingabekarte besitzt [1]. Eine Reduktion der Auflösung innerhalb der Faltungsschicht kann erreicht werden, indem nicht über jedes Neuron, sondern nur über eine Untermenge dieser in fester Schrittweite $s_y \times s_x$ zueinander gefaltet wird. Eine Alternative dazu ist die Verwendung einer *Poolingschicht*, die üblicherweise direkt nach einer Faltungsschicht zum Einsatz kommt und versucht, die Ausgabe der Faltungsschicht zu reduzieren bzw. auf das Wesentliche zu vereinfachen. Eine Poolingoperation besitzt analog zu conv2d eine Fenstergröße sowie eine Schrittweite. Die beliebteste Poolingoperation ist das *Max-Pooling*, welche sich je-

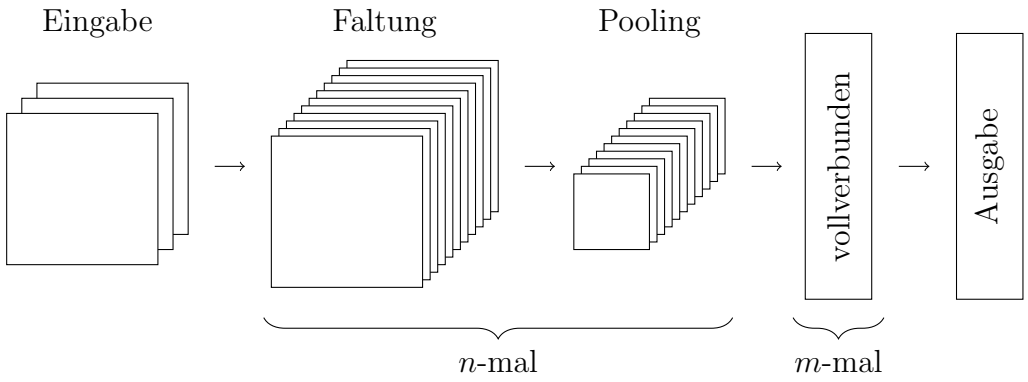


Abbildung 2.3: Typische Netzarchitektur eines CNNs bestehend aus beliebig vielen Faltungs- gefolgt von Poolingschichten. Die Umsortierung der Merkmalskarten zu einem Vektor erlaubt die Verwendung beliebig vieler vollverbundener Schichten hin zur Ausgabe.

weils für das Maximum der Aktivierungen innerhalb jedes Fensters entscheidet [35]. Eine Poolingoperation mit Fenstergröße 2×2 sowie Schrittweite 2×2 reduziert damit die Daten der Merkmalskarte um genau die Hälfte.

Ein CNN besteht typischerweise aus mehreren Faltungs- und Poolingschichten, sodass die Menge der Merkmalskarten dabei stetig erhöht und die Auflösung stetig verringert wird [35]. Damit entstehen im späteren Verlauf des Netzes Merkmalskarten, die immer größere Receptive-Fields des Eingabebildes basierend auf den ermittelten Merkmalen vorangegangener Schichten beschreiben. Nach einer gewissen Anzahl an Faltungen werden die Merkmalskarten zu einem Vektor umsortiert, sodass vollverbundene Schichten hin zur Ausgabe genutzt werden können [35]. Abbildung 2.3 illustriert den üblichen Aufbau eines CNNs.

3 Graphrepräsentationen von Bildern

Als eine *Graphrepräsentation eines Bildes* $\mathbf{B} \in \mathbb{R}^{H \times W \times C}$ wird eine Darstellung von $\mathbf{B}$ als ein gewichteter, ungerichteter sowie schleifenloser Graph $\mathcal{G}$ verstanden, deren Knoten Informationen zu ausgewählten Bereichen von $\mathbf{B}$ über eine Merkmalsmatrix $\mathbf{F} \in \mathbb{R}^{N \times M}$ speichern und deren Kanten eine Aussage über die örtlichen Nachbarschaften eines jeden Bildbereichs innewohnt. Formal lässt sich eine Graphrepräsentation eines Bildes damit als ein *Graph im zweidimensionalen euklidischen Raum* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, p)$ verstehen, dem zusätzlich zu seinen Knoten- und Kantenmengen anstatt einer Gewichtsfunktion $w\colon \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ eine Positionsfunktion $p\colon \mathcal{V} \to \mathbb{R}^2$ auf seinen Knoten in den zweidimensionalen euklidischen Raum $\mathbb{R}^2$ zugeordnet ist. Das Gewicht $w\colon \mathcal{E} \to [0, 1]$ einer Kante ergibt sich dann implizit als „Abstandsfunktion“ mit Hilfe der euklidischen Norm $\|\cdot\|_2$ auf den Positionen der Knoten und der Gaußfunktion als

$$w(v_i, v_j) \coloneqq \begin{cases} \exp\left(-\frac{\|p(v_i)-p(v_j)\|_2^2}{2\xi^2}\right), & \text{wenn } (v_i, v_j) \in \mathcal{E}, \\ 0, & \text{sonst} \end{cases} \tag{3.1}$$

mit der Standardabweichung $\xi \in \mathbb{R}$ [44]. Abbildung 3.1 veranschaulicht die Gewichtsfunktion anhand unterschiedlich gewählter ξ. Aufgrund der Symmetrie von $\|\cdot\|_2$ folgt insbesondere sofort die Ungerichtheit des Graphen $\mathcal{G}$, d.h. $w(v_i, v_j) = w(v_j, v_i)$. Die Gaußfunktion „invertiert“ dabei den Abstand zweier Knoten zueinander, sodass Knoten die weiter voneinander entfernt liegen ein geringeres Gewicht besitzen. Das korrespondiert mit dem üblichen Verständnis eines Kantengewichts in Graphen. Knoten, die näher am Ursprungsknoten liegen, gelten als „wichtiger“ und bekommen deshalb ein größeres Gewicht. Insbesondere stimmt dies mit der Konvention überein, dass $w(v_i, v_j) = 0$ für $(v_i, v_j) \notin \mathcal{E}$. Damit lässt sich weiterhin die korrespondierende Adjazenzmatrix $\mathbf{A}_{\text{dist}} \in [0, 1] \in \mathbb{R}^{N \times N}$ wie bekannt mit $(\mathbf{A}_{\text{dist}})_{ij} \coloneqq w(v_i, v_j)$ definieren. Zusätzlich zu dem Abstand der Knoten bzw. der Länge der Kanten eines Graphen im zweidimensionalen euklidischen Raum $\mathcal{G} = (\mathcal{V}, \mathcal{E}, p)$ lassen sich die Ausrichtungen der Kanten über die Positionen der Knoten von $\mathcal{G}$ festhalten. Aus $p\colon \mathcal{V} \to \mathbb{R}^2$ wird

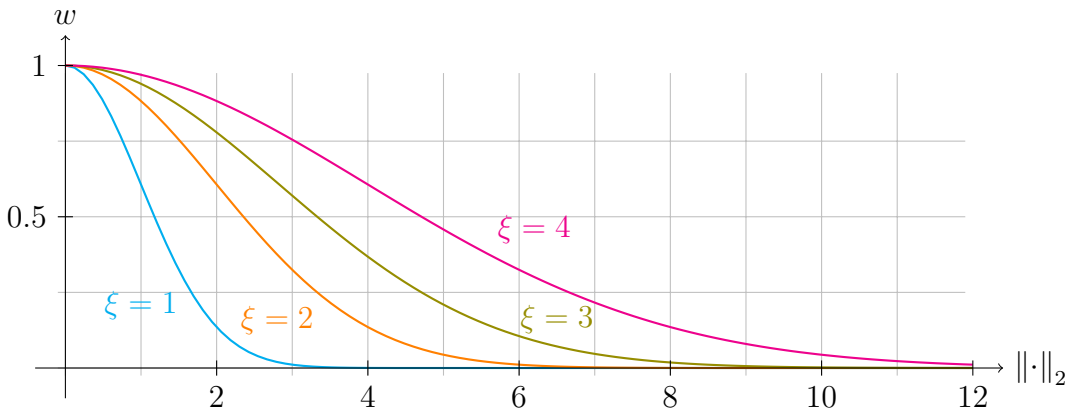


Abbildung 3.1: Illustration der „invertierten“ Kantengewichte $w \in [0,1]$ im Verhältnis zu deren Länge gegeben durch die euklidische Norm $\|\cdot\|_2$. Je größer ξ gewählt wird, umso „stumpfer“ zeichnet sich die Gaußfunktion und umso längere Kanten erhalten ein Gewicht $\gg 0$. Die Wahl von ξ ist damit abhängig von der Ausdehnung der Distanzen eines Graphen.

folglich über

$$\varphi(v_i, v_j) \coloneqq \begin{cases} \operatorname{atan2}\big(p(v_j)_1 - p(v_i)_1, p(v_j)_2 - p(v_i)_2\big) + \pi, & \text{wenn } (v_i, v_j) \in \mathcal{E}, \\ 0, & \text{sonst} \end{cases} \tag{3.2}$$

eine *Winkelfunktion* $\varphi\colon \mathcal{V} \times \mathcal{V} \to [0, 2\pi]$ auf den Kanten von $\mathcal{G}$ im Uhrzeigersinn definiert, wobei $\operatorname{atan2}(x, y) \in [-\pi, \pi]$ den Arkustangens von x/y berechnet, aber im Gegensatz zu $\operatorname{atan}(\cdot)$ die Vorzeichen beider Parameter beachtet und so den Quadranten des Ergebnisses bestimmen kann (vgl. [37]). Der Winkel Null wird dabei explizit als 2π definiert, sodass weiterhin $\varphi(v_i, v_j) = 0$ gilt, falls $(v_i, v_j) \notin \mathcal{E}$. Analog zu $\mathbf{A}_{\text{dist}}$ lässt sich damit die Adjazenzmatrix $\mathbf{A}_{\text{rad}} \in [0, 2\pi]^{N \times N}$ mit $(\mathbf{A}_{\text{rad}})_{ij} \coloneqq \varphi(v_i, v_j)$ definieren. Es ist anzumerken, dass die Winkelfunktion φ im Gegensatz zur Gewichtsfunktion w nicht symmetrisch bzw. ungerichtet ist, d.h. $\varphi(v_i, v_i) \neq \varphi(v_j, v_i)$. Insbesondere definiert $\mathbf{A}_{\text{rad}}$ damit einen gerichteten Graphen.

Die beiden Adjazenzmatrizen $\mathbf{A}_{\text{dist}}$ und $\mathbf{A}_{\text{rad}}$ beschreiben den Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E}, p)$ bis auf Translation eindeutig. Eine Veränderung einer Position $p(v)$ eines Knotens $v \in \mathcal{V}$ führt sowohl zu einer Veränderung in $\mathbf{A}_{\text{dist}}$ als auch in $\mathbf{A}_{\text{rad}}$. Eine Translation aller Knoten um $\delta \in \mathbb{R}^2$, d.h. $p(v) \to p(v) + \delta$, generiert hingegen vollkommen äquivalente Adjazenzmatrizen $\mathbf{A}_{\text{dist}}$ sowie $\mathbf{A}_{\text{rad}}$. In der Regel ist dies kein Nachteil, denn nur in den seltensten Fällen interessieren uns die absoluten Positionen der Knoten in $\mathcal{G}$, wohingegen der Abstand und die Ausrichtungen der Knoten zueinander eine größere Bedeutung genießen.

Im Folgenden werden basierend auf der Definition eines Graphen im zweidimensionalen euklidischen Raum zwei Ansätze für eine Graphrepräsentation von Bildern vorgestellt — der Repräsentation über ein reguläres Gitter sowie der Repräsentation über die Bestimmung von Superpixeln eines Bildes.

3.1 Gitter

Die einfachste Form einer Graphrepräsentation $\mathcal{G}$ eines Bildes $\mathbf{B} \in \mathbb{R}^{H \times W \times C}$ ist die Repräsentation des Bildes über einen regulären Gittergraphen, denn dafür müssen keine Berechnungen am Bild vorgenommen werden. Ein *regulärer Gittergraph im zweidimensionalen euklidischen Raum* ist ein Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, p)$, der aus einem regulären Gitter gewonnen wurde und demnach genau $N \coloneqq H \times W$ Knoten enthält, d.h. einen Knoten für jedes Pixel in $\mathbf{B}$ [8]. Die Positionsfunktion der Knoten $p \colon \mathcal{V} \to \mathbb{R}^2$ entspricht damit genau der Koordinate des korrespondierenden Pixels im Urpsprungsbild. Sei dafür $v \in \mathcal{V}$ der Knoten zu dem Bildpunkt an Position (x, y). Folglich gilt für die Position des Knotens $p(v) \coloneqq [x, y]^\top$. Die örtlich um den Knoten $v \in \mathcal{V}$ liegenden Knoten gelten dann auch im Gittergraph als benachbart und werden über eine Kante in $\mathcal{E}$ verbunden. Dabei unterscheidet man zwischen zwei frei wählbaren *Konnektivitäten* des Graphen [8]. Eine Konnektivität von 4 bedeutet, dass ein Knoten (ohne Berücksichtigung der Randknoten) genau vier Nachbarn besitzt, die horizontal und vertikal zu ihm stehen. Bei einer Konnektivität von 8 gelten zusätzlich die vier diagonal zu ihm stehenden Knoten als Nachbarn und die Nachbarschaft wird folglich als ein 3×3 Fenster um den Knoten v verstanden.

Analog zu den Daten der Farbkanäle an den einzelnen Pixeln des Bildes besitzt auch der Graph über einer Merkmalsfunktion $f \colon \mathcal{V} \to \mathbb{R}^C$ bzw. einer Merkmalsmatrix $\mathbf{F} \in \mathbb{R}^{N \times C}$ diese Information mit $f(v) \coloneqq \mathbf{B}_{p(v)_2, p(v)_1}$ an seinen Knoten.

Effiziente Adjazenzbestimmung. Entgegen der Pixelanordnung in einem Bild ist die Anordnung der Knoten in einem Gittergraphen völlig irrelevant und kann willkürlich gewählt werden. Ein Aufbau der Kantenmenge $\mathcal{E}$ bzw. der korrespondierenden, ungewichteten Adjazenzmatrix $\mathbf{A}$ kann jedoch besonders effizient gestaltet werden, wenn die Knoten zeilenweise entsprechend ihrer Bildkoordinate angeordnet werden. Dafür wird das Bild $\mathbf{B}$ zunächst an dessen Rändern um eine zusätzliche Spalte bzw. Zeile erweitert, d.h. $\mathbf{B} \in \mathbb{R}^{(H+2) \times (W+2) \times C}$. Daraus ergeben sich $N \coloneqq (H+2)(W+2)$ viele Knoten eines Graphen, die die jeweiligen Gitterpunkte repräsentieren. Ein Kno-

ten $v_i \in \mathcal{V}$, $W+3 < i < N-W-3$, ist dann adjazent zu den Knoten $v_j \in \mathcal{V}$ mit

$$j \in \{i-W-2, i-1, i+1, i+W+2\}$$

bei einer Konnektivität von 4 bzw. mit

$$j \in \{i-W-3, i-W-2, i-W-1, i-1, i+1, i+W+1, i+W+2, i+W+3\}$$

bei einer Konnektivität von 8. Anschließend müssen die ungültigen Knoten, d.h. die an den Rändern des Gitters hinzugefügten Knoten, aus $\mathcal{V}$ gelöscht werden. Dazu zählen die ersten und letzten $W+3$ Knoten aus $\mathcal{V}$ sowie die vertikal am Rand des Gitters liegenden Knoten, die über eine Schrittweite von $W+2$ über der Knotenmenge eliminiert werden können.

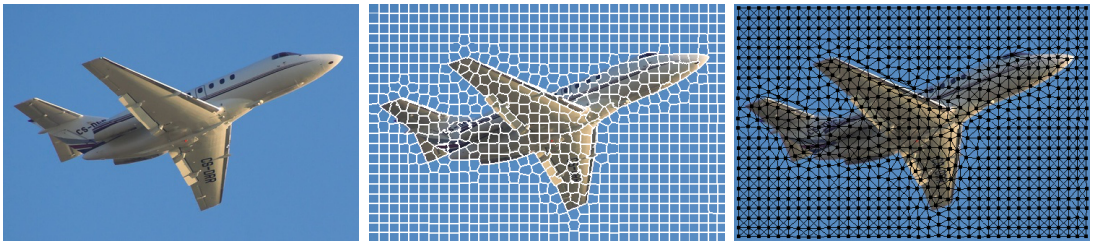
3.2 Superpixel

Superpixel erfreuen sich in allen Anwendungen der Bildverarbeitung, insbesondere als Vorverarbeitungsschritt, immer größerer Beliebtheit, da sie die Eingabegröße eines Problems mit oftmals zu vernachlässigem Fehler reduzieren [15]. Superpixel fassen dabei ein Bild in logische, räumliche Gruppen anhand ausgewählter Metriken zusammen, sodass sich Pixel innerhalb einer Gruppe besonders ähnlich sind.
Formal besteht eine *Superpixelrepräsentation* eines Bildes $\mathbf{B} \in \mathbb{R}^{H \times W \times C}$ aus einer Menge von $N \in \mathbb{N}$ *Superpixeln* bzw. *Regionen* $\mathcal{S} \coloneqq \{\mathcal{S}_n\}_{n=1}^{N}$ mit $\mathcal{S}_n \subseteq W \times H$, sodass $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ und $\bigcup_{n=1}^{N} \mathcal{S}_n = W \times H$ [50]. Für eine gültige Superpixelsegmentierung fordern wir desweiteren, dass $\mathcal{S}_n$ zusammenhängend ist [50]. Aus der Menge an Superpixeln $\mathcal{S}$ lässt sich damit eine *Segmentierungsmaske* $\mathbf{S} \in \{1, \dots, N\}^{H \times W}$ über

$$\mathbf{S}_{yx} = n \text{ genau dann, wenn } (x, y) \in \mathcal{S}_n$$

gewinnen, die jedem Pixel in $\mathbf{B}$ eine eindeutige Superpixelzugehörigkeit n zuweist.
Aus einer Superpixelrepräsentation $\mathcal{S}$ bzw. $\mathbf{S}$ eines Bildes $\mathbf{B}$ kann ein Graph im zweidimensionalen euklidischen Raum $\mathcal{G} = (\mathcal{V}, \mathcal{E}, p)$ wie folgt definiert werden. Seien dafür die einzelnen Superpixel $\mathcal{S} = \{\mathcal{S}_n\}_{n=1}^{N}$ die Knoten $\mathcal{V} = \{v_n\}_{n=1}^{N}$ des Graphen $\mathcal{G}$, d.h. $v_n = \mathcal{S}_n$. Dann ordnet die Positionsfunktion $p\colon \mathcal{V} \to \mathbb{R}^2$ den einzelnen Superpixeln über

$$p(v_n) \coloneqq [\bar{x}, \bar{y}]^\top = \frac{1}{|\mathcal{S}_n|} \sum_{(x,y) \in \mathcal{S}_n} [x, y]^\top$$



(a) Bild (b) Superpixelsegmentierung (c) Graphrepräsentation

Abbildung 3.2: Illustration des Prozesses zur Graphgenerierung anhand einer Superpixelrepräsentation eines Bildes. Ein Bild (a) wird dafür zuerst in eine Menge von Superpixeln segmentiert (b). Die absoluten Zentren der Superpixel bilden die Knotenpunkte des Graphen. Benachbarte Superpixel werden über eine Kante miteinander verbunden (c).

eine eindeutige Position in der Ebene über deren absoluten Schwerpunkt zu. Die Kantenrelation $\mathcal{E}$ des Graphen $\mathcal{G}$ wird desweiteren über die örtlichen Nachbarregionen der Superpixel basierend auf einem 3×3 Fenster um die Pixel in $\mathbf{S}$ definiert. Falls sich beispielsweise die Werte in $\mathbf{S}_{y+1,x}$ und $\mathbf{S}_{y,x}$ unterscheiden, d.h. $\mathbf{S}_{y,x} \neq \mathbf{S}_{y+1,x}$ mit $\mathbf{S}_{y,x} = i$ und $\mathbf{S}_{y+1,x} = j$, dann wird den Knoten $v_i, v_j \in \mathcal{V}$ über $(v_i, v_j) \in \mathcal{E}$ eine Kante in $\mathcal{G}$ zugeordnet. Formal lassen sich dabei erneut die zwei Konnektivitäten 4 und 8 unterscheiden, für die in der Praxis lediglich ein Unterschied in Bereichen erkennbar ist, deren Superpixelregionen rechteckig als Gitter aneinander liegen. Abbildung 3.2 illustriert den beschriebenen Prozess der Graphgenerierung anhand einer Superpixelrepräsentation eines Bildes.
Zusätzlich zu der Positionsfunktion p auf den Superpixelknoten kann der Graph $\mathcal{G}$ um eine *Massefunktion* $m\colon \mathcal{V} \to \mathbb{N}$ mit $m(v_n) := |S_n|$ zu $\mathcal{G} = (\mathcal{V}, \mathcal{E}, p, m)$ erweitert werden, die die Masse bzw. den Flächeninhalt der einzelnen Superpixelregionen beschreibt.

Effiziente Adjazenzbestimmung. Die Bestimmung einer Kantenrelation $\mathcal{E}$ auf einer Superpixelrepräsentation ist aufgrund der pixelweisen Iteration auf der Segmentierungsmaske und dem sukzessiven Vergleich mit den jeweiligen Nachbarpixeln über ein 3×3 Fenster sehr berechnungsaufwändig. Eine effizientere und zugleich elegantere Bestimmung einer Kantenrelation als ungewichtete Adjazenzmatrix kann über eine versetzte Überlagerung der Segmentierungsmasken gewonnen werden. Sei dafür $\mathbf{S} \in \{1, \dots N\}^{H\times W}$ eine Segmentierungsmaske. Dann wird für die vertikale Adjazenzbestimmung die oberste und unterste Zeile aus $\mathbf{S}$ abgeschnitten. Die so entstehenden Matrizen werden in $\mathbf{S}_\uparrow$ bzw. $\mathbf{S}_\downarrow \in \{1, \dots N\}^{(H-1)\times W}$ gespeichert. $\mathbf{S}_\uparrow$ und $\mathbf{S}_\downarrow$ werden anschließend paarweise auf Ungleichheit getestet und dessen Ergebnis in der

$$\mathbf{S} = \begin{bmatrix} 1 & \underline{1} & 2 & 3 \\ 1 & \underline{2} & 2 & 3 \\ 1 & 1 & 4 & 4 \end{bmatrix}$$

(a)

$$\mathbf{S}_\uparrow = \begin{bmatrix} 1 & 2 & 2 & 3 \\ 1 & 1 & 4 & 4 \end{bmatrix}$$

$$\mathbf{S}_\downarrow = \begin{bmatrix} 1 & 1 & 2 & 3 \\ 1 & 2 & 2 & 3 \end{bmatrix}$$

$$(\mathbf{S}_\uparrow \neq \mathbf{S}_\downarrow) = \begin{bmatrix} \times & \checkmark & \times & \times \\ \times & \checkmark & \checkmark & \checkmark \end{bmatrix}$$

(b)

$$(2,1) \in \mathcal{E},\ (1,2) \in \mathcal{E}$$
$$(4,2) \in \mathcal{E},\ (4,3) \in \mathcal{E}$$

$$\left(\mathbf{A} \vee \mathbf{A}^\top\right) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

(c)

Abbildung 3.3: Adjazenzbestimmung einer Segmentierungsmaske $\mathbf{S} \in \{1,2,3,4\}^{3\times4}$ mit eingezeichneten vertikalen Adjazenzen (a). Daraus lassen sich $\mathbf{S}_\uparrow$, $\mathbf{S}_\downarrow$ sowie die boolsche Matrix $(\mathbf{S}_\uparrow \neq \mathbf{S}_\downarrow)$ bestimmen (b). Die Einträge in $\mathbf{S}_\uparrow$ und $\mathbf{S}_\downarrow$ an den Indizes der wahren Werte in $\mathbf{S}_\uparrow \neq \mathbf{S}_\downarrow$ bilden jeweils eine Kante im Graphen (c).

boolschen Matrix $(\mathbf{S}_\uparrow \neq \mathbf{S}_\downarrow) \in \{0,1\}^{(H-1)\times W}$ gespeichert. Liegen dabei wahre, d.h. ungleiche Einträge vor, so ist eine vertikale Adjazenz zwischen zwei unterschiedlichen Regionen vorhanden. Die Einträge $i, j \in \{1, \dots N\}$ in $\mathbf{S}_\uparrow$ und $\mathbf{S}_\downarrow$ an den Koordinaten der wahren Einträge in der Binärmaske bilden folglich die Knotenindizes einer Kante $(v_i, v_j) \in \mathcal{E}$ benachbarter Regionen und werden in eine Adjazenzmatrix mit $\mathbf{A}_{ij} = 1$ übertragen. Analog wird für die horizontalen bzw. für die diagonalen Adjazenzen vorgegangen. $\mathbf{A} \vee \mathbf{A}^\top$ liefert dann die resultierende ungerichtete Adjazenzmatrix (vgl. [48]). Abbildung 3.3 illustriert das Verfahren anhand eines einfachen Beispiels.

Globale und lokale Normierung. Eine Menge generierter Graphen aus den Superpixelrepräsentationen einer Bildermenge können sich zu Teilen extrem unterscheiden, beispielsweise durch variierende Auflösungen der Bilder. So besitzt ein Graph möglicherweise nur sehr „kurze“ Kanten, wohingegen ein anderer im Vergleich dazu ausschließlich aus „längeren“ Kanten besteht. Das kann bei der Transformation solcher Graphen in die Adjazenzmatrixrepräsentation $\mathbf{A}_{\text{dist}} \in [0,1]^{N\times N}$ aufgrund der Wahl des Parameters $\xi \in \mathbb{R}$ zu Problemen führen. Eine statische Wahl des Parameters ξ führt dann gegebenenfalls zu Einträgen in $\mathbf{A}_{\text{dist}}$, die stets sehr nahe bei Eins bzw. Null liegen. Eine Normierung der Kantenlängen erscheint daher für die Transformation nach $\mathbf{A}_{\text{dist}}$ sinnvoll. Eine *Normierung* eines Graphen $\mathcal{G}$ sieht dafür bei der Transformation nach $\mathbf{A}_{\text{dist}}$ eine Skalierung von $\|p(v_i) - p(v_j)\|_2^2 \in \mathbb{R}$ in das Intervall $[0,1]$ vor. Eine *globale Normierung* von $\mathcal{G}$ skaliert dabei $\|p(v_i) - p(v_j)\|_2^2$ über die maximale Kantenlänge $\|\cdot\|_2^2$ aller Kanten des Graphen, wohingegen die *lokale Normierung* lediglich eine Skalierung über die maximale Kantenlänge der ausgehenden Kanten von $v_i \in \mathcal{V}$ vorsieht. Die lokale Normierung generiert dabei üblicherweise

einen gerichteten Graphen bzw. eine nicht-symmetrische Adjazenzmatrix $\mathbf{A}_{\text{dist}}$. Ein üblicher Trick für die Symmetrieerhaltung ist die anschließende Anpassung von $\mathbf{A}_{\text{dist}}$ über $\mathbf{A}_{\text{dist}} \rightarrow \frac{1}{2}\left(\mathbf{A}_{\text{dist}} + \mathbf{A}_{\text{dist}}^{\top}\right)$ [39].
Es ist anzumerken, dass ein normierter Graph, repräsentiert als $\mathbf{A}_{\text{dist}}$, neben der bereits vorhandenen Translationsinvarianz insbesondere skalierungsinvariant ist. Im weiteren Verlauf dieser Arbeit wird der Prozess der Normierung eines Graphen implizit angenommen.

3.2.1 Verfahren

In der Literatur finden sich zahlreiche Verfahren zur Bestimmung einer Superpixelrepräsentation aus einem Bild mit jeweils unterschiedlichen Stärken und Schwächen [3, 50]. Zwei beliebte Verfahren, die aufgrund ihrer im Allgemeinen guten Resultate bei geringer Berechnungskomplexität immer wieder zu finden sind, sind der SLIC- sowie der Quickshift-Algorithmus [3, 13, 15, 20, 50]. Beide Verfahren werden im Folgenden näher vorgestellt.

SLIC. Der Superpixelalgorithmus *Simple Linear Iterative Clustering (SLIC)* ist ein recht einfach gehaltener lokaler *K-Means*-Clustering-Ansatz, welcher sich dennoch durch seine Geschwindigkeit, seine Speichereffizienz und insbesondere durch seine erfolgreiche Segmentierung hinsichtlich der Farbabgrenzen seiner Superpixel im Vergleich zu anderen „State-of-the-Art"-Algorithmen auszeichnet [3, 15].
Ein Bild $\mathbf{B} \in \mathbb{R}^{H \times C \times 3}$ im Lab-Farbraum wird dazu initial in $K \in \mathbb{N}$ viele Clusterzentren $\mathbf{c}_k \coloneqq [l_k, a_k, b_k, x_k, y_k]^{\top}$ mit jeweils gleichmäßigem Abstand $S \coloneqq \sqrt{WH/K}$ zerlegt [3]. Daraufhin werden in einem iterativen Prozess die Zugehörigkeiten der Pixel zu ihren jeweiligen Clustern über eine Distanzmetrik D bestimmt und die Clusterzentren entsprechend ihrer neuen Gruppierungen angepasst. Entgegen der konventionellen k-Means-Formulierung werden dabei aber nicht die Distanzen jedes Pixels zu jedem Clusterzentrum berechnet, sondern lediglich für die Pixel, die sich in der Region der Größe $2S \times 2S$ um $\mathbf{c}_k$ befinden [3]. Dies führt folglich zu einer drastischen Reduzierung der Anzahl an Distanzberechnungen und zu einem effizienteren Algorithmus. Wurde zu jedem Pixel dessen ähnlichstes Clusterzentrum bestimmt, werden die Zentren über die Durchschnittsbildung all seiner zugehörigen Pixel neu justiert. Nach einer festgelegten Anzahl an Iterationsschritten (üblicherweise 10) bricht der Algorithmus schließlich ab [3]. Das gesamte Verfahren ist in Algorithmus 3.1 zusammengefasst.
Die Distanzmetrik D basiert auf der euklidischen Norm $\|\cdot\|_2$ im fünfdimensionalen

Eingabe: Bild $\mathbf{B} \in \mathbb{R}^{H \times W \times 3}$ im Lab-Farbraum, Anzahl der Cluster K
Ausgabe: Segmentierungsmaske $\mathbf{S} \in \mathbb{N}^{H \times W}$
  Initialisiere K Clusterzentren $\{\mathbf{c}_k\}_{k=1}^{K}$ mit gleichmäßigem Abstand S.
  $\mathbf{S}_{yx} \leftarrow -1$ für jedes Pixel $\mathbf{B}_{yx}$ an (x, y).
  $\mathbf{D}_{yx} \leftarrow \infty$ für jedes Pixel $\mathbf{B}_{yx}$ an (x, y).
  repeat
    for $\mathbf{c}_k \in \{\mathbf{c}_k\}_{k=1}^{K}$ **do**
      for Pixel $\mathbf{B}_{yx}$ an (x, y) in $2S \times 2S$ Region um $\mathbf{c}_k$ **do**
        Berechne Distanz D zwischen $\mathbf{c}_k$ und $\mathbf{B}_{yx}$.
        if $D < \mathbf{D}_{yx}$ **then**
          $\mathbf{D}_{yx} \leftarrow D$
          $\mathbf{S}_{yx} \leftarrow k$
        end if
      end for
    end for
    Justiere Clusterzentren $\{\mathbf{c}_k\}_{k=1}^{K}$.
  until Endbedingung

Algorithmus 3.1: SLIC-Algorithmus, der eine Segmentierungsmaske $\mathbf{S} \in \mathbb{N}^{H \times W}$ in den Ausmaßen des Eingabebildes $\mathbf{B} \in \mathbb{R}^{H \times W \times 3}$ über ein lokales K-Means-Clustering bei K gleichmäßig verteilten initialen Clusterzentren generiert.

Raum $[l, a, b, x, y]^\top$ auf den Farben und den Positionen der Pixel bzw. der Clusterzentren. Dabei ergeben sich jedoch Inkonsistenten für unterschiedliche Superpixelgrößen. Bei großen Superpixeln führt dies zu einer stärkeren Gewichtung der räumlichen Positionen der Superpixel, wohingegen bei kleinen Superpixeln genau das Gegenteil der Fall ist [3]. Es ist daher notwendig, die Distanzen der Positionen D_S und Farben D_F getrennt voneinander zu betrachten und mittels dem Abstand S der initialen Clusterzentren und einer Normalisierungskonstante $F \in \mathbb{R}$ bezüglich der Gewichtung der Farben zu normieren. Damit ergibt sich D als [3]

$$
\begin{aligned}
D &\coloneqq \sqrt{\left(\frac{D_S}{S}\right)^2 + \left(\frac{D_F}{F}\right)^2} \\
D_S &\coloneqq \left\| [x, y]^\top - [x_k, y_k]^\top \right\|_2 \\
D_F &\coloneqq \left\| [l, a, b]^\top - [l_k, a_k, b_k]^\top \right\|_2.
\end{aligned}
$$

Die Normalisierungskonstante F kann damit auch als Gewichtung zwischen der Form und den Farbabgrenzungen der Superpixel verstanden werden. Falls F sehr klein gewählt wird, respektieren die Superpixel Farbabgrenzungen besser, aber besitzen im Allgemeinen auch eine eher unreguläre Form [3].

(a) SLIC (b) Quickshift

Abbildung 3.4: Ein Bus segmentiert über SLIC mit jeweils 400, 800 und 1600 Superpixeln (a) sowie über Quickshift mit 600 Superpixeln (b). Dabei werden die unterschiedlichen Verfahren zur Generierung von Superpixeln deutlich. Wohingegen SLIC möglichst quadratische, gleichgroße Superpixel erzeugt, generiert Quickshift sowohl sehr große wie auch sehr kleine Superpixel in allen möglichen Formvariationen.

Damit ist SLIC ein $\mathcal{O}(WH)$ effizienter Algorithmus mit nur zwei frei wählbaren Parametern K und F [3]. Viele Anwendungen im Bereich der Bildverarbeitung machen sich daher SLIC zunutze (vgl. [15, 20, 50]). Abbildung 3.4 (a) zeigt ein Beispielresultat des SLIC-Algorithmus bei unterschiedlich gewählten Anzahlen an Clusterzentren.

Quickshift. Ein weiteres Verfahren zur Bildsegmentierung ist der gradientenbasierte Algorithmus *Quickshift*, der unter anderem bereits zur Objektlokalisierung auf Basis von Superpixeln benutzt wurde [13, 53].
Quickshift startet dabei mit der Berechnung der *Parzen-Dichteschätzung* h eines Bildes $\mathbf{B} \in \mathbb{R}^{H \times W \times 3}$ für jedes Pixel $\mathbf{B}_{yx} \in \mathbb{R}^3$ im fünfdimensionalen Raum über die Gaußfunktion, d.h.

$$h(x, y, \mathbf{B}) = \sum_{\substack{1 \leq i \leq W \\ 1 \leq j \leq H}} \frac{1}{(2\pi\xi)^5} \exp\left(-\frac{1}{2\xi^2} \left\| \begin{pmatrix} x - x_i \\ y - y_i \\ \alpha(\mathbf{B}_{yx} - \mathbf{B}_{y_i x_i}) \end{pmatrix} \right\|_2^2 \right),$$

mit der Standardabweichung $\xi \in \mathbb{R}$ und der Gewichtung des Farbeinflusses über $\alpha \in \mathbb{R}$ [53]. Je größer ξ gewählt wird, umso größer wird der Anteil weitentfernter Pixel zu der Dichte des jeweils betrachteten Pixels. In der Praxis kann, analog wie bei SLIC, eine Fenstergröße $S \times S$ zur Einschränkung der Betrachtung der Pixel um

einen Pixel definiert werden [50]. Obwohl Quickshift standardmäßig S auf ∞ setzt, lohnt es sich aus Effizienzgründen eine Größe in Abhängigkeit von ξ zu wählen [53]. Basierend auf den Dichten des Bildes generiert Quickshift einen Baum, der jedem Pixel $\mathbf{B}_{y_i x_i}$ einen Nachbarpixel $\mathbf{B}_{y_j x_j}$ zuordnet, welcher einen höheren Dichtewert besitzt, d.h.

$$p(x_j, y_j, \mathbf{B}) > p(x_i, y_i, \mathbf{B}).$$

Falls $\mathbf{B}_{y_i x_i}$ kein solches Pixel zugeordnet werden kann, wird es mit keinem Pixel verbunden. Die verbundene Menge an Knoten repräsentiert dann schließlich einen Superpixel des Bildes.
Quickshift segmentiert damit ein Bild auf Basis von drei Parametern: ξ für die Standardabweichung der Gaußfunktion, α zur Gewichtung des Farbterms sowie S zur Einschränkung der Berechnung über ein Fenster der Größe $S \times S$.
Die Superpixel, die Quickshift produziert, besitzen im Gegensatz zu den Superpixeln in SLIC keine Einschränkung in ihrer Anzahl, Form oder Größe. Ein segmentiertes Bild mit feinen Strukturen enthält damit weitaus mehr Superpixel als ein Bild mit großen gleichfarbigen Flächen, welches nur sehr wenige, großflächige Superpixel generiert. Ebenso gibt es im Gegensatz zu SLIC keinen Parameter, der die Form der Superpixel steuern kann, sodass Quickshift Superpixel aller erdenklichen Formen produzieren kann. Abbildung 3.4 (b) illustriert ein Resultat einer Quickshift-Segmentierung und insbesondere den Vergleich zu dem entsprechenden SLIC-Äquivalent.

Weitere Verfahren. In den letzten Jahren wurden ebenso zahlreiche eigenvektorbasierte Verfahren zur Bildsegmentierung entwickelt [12, 50]. Obwohl diese vielversprechende Resultate aufweisen, sind sie jedoch entsprechend langsam um von praktischem Nutzen für die meisten Anwendungen zu sein [12]. Andere Algorithmen wiederum zeigen sich als besonders berechnungseffizient, liefern aber dementsprechend unzufriedenstellende Ergebnisse [12]. Namenshaft zu erwähnen sei noch die *effiziente graphbasierte Bildsegmentierung* von Felzenszwalb und Huttenlocher, die in der Regel unter dem Namen *Felzenszwalb-Segmentierung* bekannt ist [12]. Der Algorithmus von Felzenszwalb und Huttenlocher zeichnet sich dabei durch seine geringe Berechnungskomplexität aus und versucht gleichzeitig, globale Eigenschaften des Bildes zu erhalten. Dafür wird das Bild als regulärer Gittergraph initialisiert, bei dem die Kantengewichte eine Aussage über den Unterschied in Helligkeit und Farbe der benachbarten Pixel treffen. Der Algorithmus versucht daraufhin, Knoten insofern zu verschmelzen, dass der Unterschied zwischen den Kantengewichtungen

innerhalb einer Region möglichst gering bleibt und zwischen benachbarten Regionen möglichst groß wird (vgl. [12]). Die Felzenszwalb-Segmentierung zeigte sich aber in Tests auf einer Reihe von Bildern weniger geeignet, da dessen Parameter für jedes Bild eine spezielle Anpassung benötigt und folglich eine statische Wahl dieser Parameter zu unbrauchbaren Ergebnissen führt.

3.2.2 Merkmalsextraktion

Die Darstellung eines Bildes über einen Graphen $\mathcal{G}$, der aus einer Superpixelrepräsentation $\mathcal{S}$ gewonnen wurde, besitzt in der Regel weitaus weniger Knoten im Gegensatz zu der reinen Darstellung des Bildes über eine Gitterrepräsentation. Die Superpixel bzw. die Regionen der Segmentierungsmaske können dabei jedoch die willkürlichsten Formen annehmen und besitzen lediglich die Einschränkung, dass diese stets zusammenhängend sind. Die Form eines Superpixels muss demnach bestmöglichst eingefangen bzw. beschrieben werden können — ein Prozess, der in der Bildverarbeitung als *Merkmalsextraktion* bekannt ist [49]. Ein geeignetes Mittel zur Beschreibung einzelner Objekte in einem segmentierten Bild sind die *Momente*, welche in nicht-zentrierte, translationsinvariante, skalierungsinvariante und rotationsinvariante Momente unterschieden werden [49].

Nicht-zentrierte Momente. Zu der binären Segmentierungsmaske $\mathbf{S} \in \{0,1\}^{H \times W}$ sind die *nicht-zentrierten Momente* vom Grad $(i+j)$, $i, j \in \mathbb{N}$, definiert als [49]

$$\mathbf{M}_{ij} \coloneqq \sum_{x}^{W} \sum_{y}^{H} x^i y^j \mathbf{S}_{yx}.$$

Obwohl der Grad eines Moments beliebig hoch gewählt werden kann, so reichen in der Praxis meist wenige Momente niedrigen Grades aus (≤ 3), um eine Region hinreichend genau zu charakterisieren [49]. Bildeigenschaften, die durch nicht-zentrierte Momente beschrieben werden können, sind unter anderem dessen Fläche über $\mathbf{M}_{00}$ sowie dessen absoluter Schwerpunkt $\{\bar{x}, \bar{y}\} = \{\mathbf{M}_{10}/\mathbf{M}_{00}, \mathbf{M}_{01}/\mathbf{M}_{00}\}$ [49].

Translationsinvariante (zentrale) Momente. Nicht-zentrierte Momente sind aufgrund ihrer Berücksichtigung der Position einer Region im Bild meist unerwünscht, sie helfen aber für die weitere Definition von translationsinvarianten Momenten. Mit Hilfe der absoluten Schwerpunktskoordinaten $\{\bar{x}, \bar{y}\}$ können die *translationsinvari-*

anten Momente über

$$\mu_{ij} \coloneqq \sum_{x}^{W} \sum_{y}^{H} (x - \bar{x})^i (y - \bar{y})^j \mathbf{S}_{yx}$$

definiert werden [49]. Sie lassen sich weiterhin direkt aus $\mathbf{M}_{ij}$ ermitteln. So gilt zum Beispiel, dass $\mu_{00} = \mathbf{M}_{00}$ oder $\mu_{11} = \mathbf{M}_{11} - \bar{x}\mathbf{M}_{01} = \mathbf{M}_{11} - \bar{y}\mathbf{M}_{10}$ (vgl. [49]).

Skalierungsinvariante Momente. Für $i + j \geq 2$ können desweiteren die *skalierungsinvarianten Momente* η_{ij} konstruiert werden, die invariant bezüglich Skalierung und Translation sind. Dafür wird das entsprechende translationsinvariante Moment μ_{ij} durch die entsprechende Fläche $\mathbf{M}_{00}$ bzw. μ_{00} des Segments geteilt, d.h. [49]

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{(1+(i+j)/2)}}.$$

(Rotationsinvariante) Hu-Momente. Hu verwendet eine nichtlineare Kombination der skalierungsinvarianten Momente bis zum Grad 3, um aus ihnen zusätzlich eine Rotationsinvarianz zu gewinnen [22, 49]. Daraus ergeben sich die sieben *rotationsinvarianten Momente* bzw. die *Hu-Momente* [49]:

$$\begin{aligned}
\mathbf{h}_1 &= \eta_{20} + \eta_{02} \\
\mathbf{h}_2 &= (\eta_{20} - \eta_{02})^2 + (2\eta_{11})^2 \\
\mathbf{h}_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
\mathbf{h}_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
\mathbf{h}_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\Big((\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\Big) \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\Big(3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\Big) \\
\mathbf{h}_6 &= (\eta_{20} - \eta_{02})\Big((\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\Big) + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
\mathbf{h}_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\Big((\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\Big) \\
&\quad + (\eta_{03} - 3\eta_{12})(\eta_{21} + \eta_{03})\Big(3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\Big)
\end{aligned}$$

Weitere Merkmale. Aus den translationsinvarianten Momenten lassen sich weitere Merkmale gewinnen, denen insbesondere eine anschauliche Interpretation zu Grunde liegt. Sei dafür die *Kovarianzmatrix* der binären Segmentierungsmaske definiert über

$$\begin{bmatrix} \mu'_{20} & \mu'_{11} \\ \mu'_{11} & \mu'_{02} \end{bmatrix},$$

wobei $\mu'_{20} \coloneqq \mu_{20}/\mu_{00}$, $\mu'_{02} \coloneqq \mu_{02}/\mu_{00}$ und $\mu'_{11} \coloneqq \mu_{11}/\mu_{00}$ [49]. Die beiden Eigenvektoren dieser Matrix

$$\lambda_i = \frac{\mu'_{20} + \mu'_{02}}{2} \pm \frac{\sqrt{4\mu'_{11} + (\mu'_{20} - \mu'_{02})^2}}{2}$$

entsprechen über $\text{axis}_i \coloneqq 4\sqrt{\lambda_i}$ der Länge der großen bzw. kleinen Halbachse einer Ellipse, die die Region minimal umschließt [49, 52]. Damit kann die *Orientierung* der Region aus dem Winkel des Eigenvektors des größten Eigenwerts mit Hilfe des Arkustangens über $\text{ori} \coloneqq \text{atan}(2\mu'_{11}/(\mu'_{20} - \mu'_{02}))/2$ berechnet werden [52]. Ähnlich dazu lässt sich die *Exzentrizität* (engl. *Eccentricity*) $\text{ecc} \coloneqq \sqrt{1 - \lambda_1/\lambda_2}$ als das Verhältnis der beiden Hauptachen zueinander definieren [45, 52].
Neben den Merkmalen, die sich aus den Momenten ergeben, können noch zahlreiche weitere Merkmale aus den Formen einer Region gewonnen werden (vgl. [45]). Beispiele dafür sind unter anderem die Merkmale des *minimalen Hüllkörpers* (engl. *Bounding-Box*) einer Region, d.h. dem kleinsten waagerechten Rechteck, welches die Region umschließt. Die Breite, Höhe und Fläche dieses Hüllkörpers können aus einer binären Segmentierungsmaske $\mathbf{S} \in \{0,1\}^{H \times W}$ über

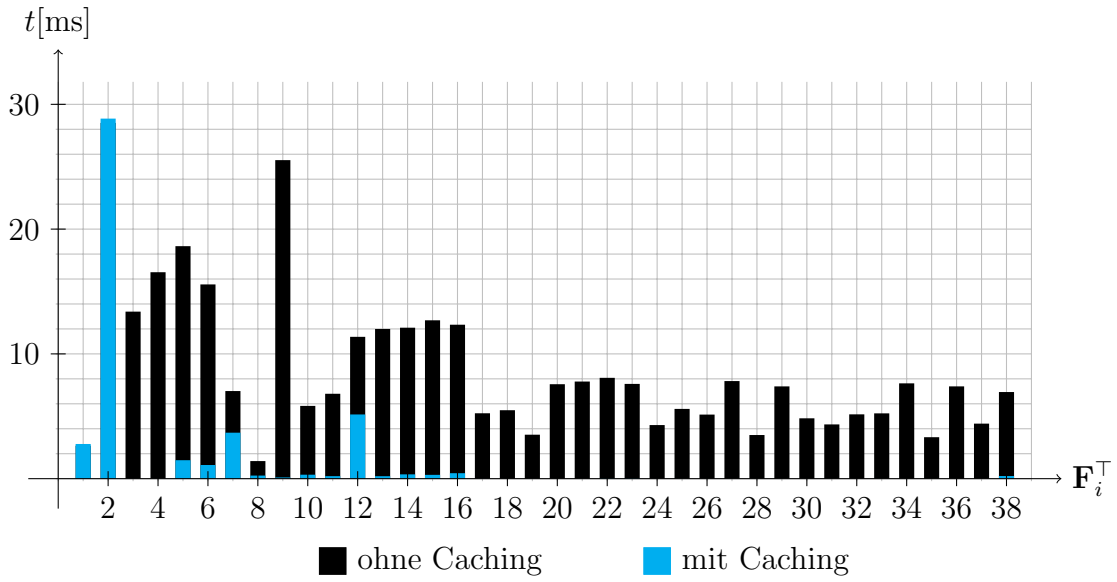
$$\begin{aligned}
\text{box}_x &\coloneqq \max(\{x \mid \mathbf{S}_{yx} = 1\}) - \min(\{x \mid \mathbf{S}_{yx} = 1\}) \\
\text{box}_y &\coloneqq \max(\{y \mid \mathbf{S}_{yx} = 1\}) - \min(\{y \mid \mathbf{S}_{yx} = 1\}) \\
\text{box} &\coloneqq \text{box}_x \text{box}_y
\end{aligned}$$

gewonnen werden. Das *Ausmaß* einer Region (engl. *Extent*) ist weiterhin definiert als $\text{ext} \coloneqq \mathbf{M}_{00}/\text{box}$ [52]. Der *gleichwertige Durchmesser* (engl. *Equivalent-Diameter*) zu der Fläche einer Region lässt sich über $\text{dia} \coloneqq \sqrt{4\mathbf{M}_{00}/\pi}$ beschreiben [45, 52]. Abschließend lassen sich die absoluten Schwerpunktskoordianten $\{\bar{x}, \bar{y}\}$ in relative bzw. translationsinvariante Schwerpunktskoordinaten $\{\hat{x}, \hat{y}\}$ mittels

$$\hat{x} \coloneqq \bar{x} - \min(\{x \mid \mathbf{S}_{yx} = 1\}) \qquad \hat{y} \coloneqq \bar{y} - \min(\{y \mid \mathbf{S}_{yx} = 1\})$$

überführen. Insgesamt ergeben sich daraus 38 Merkmale, die die Form eines einzelnen Superpixels beschreiben (vgl. Abbildung 3.5). Zusätzlich zu diesen beschreiben drei weitere Merkmale die Durchschnittsfarbe der drei Farbkanäle eines Superpixels.

Caching. Viele der 38 ermittelten Merkmale werden über bereits definierte Merkmale beschrieben. So bauen insbesondere die Momente sukzessive über die zusätzlichen Stufen der Invarianz aufeinander auf. Weiterhin erfordern Merkmale wie die



1. $\mathbf{M}_{00}$	2. box	3. box_x	4. box_y	5. $\hat{x}$	6. $\hat{y}$	7. ecc	8. dia
9. ext	10. $\mathbf{h}_1$	11. $\mathbf{h}_2$	12. $\mathbf{h}_3$	13. $\mathbf{h}_4$	14. $\mathbf{h}_5$	15. $\mathbf{h}_6$	16. $\mathbf{h}_7$
17. η'_{02}	18. η'_{11}	19. η'_{20}	20. λ_1	21. λ_2	22. axis_1	23. axis_2	24. μ_{02}
25. μ_{03}	26. μ_{11}	27. μ_{12}	28. μ_{20}	29. μ_{21}	30. μ_{30}	31. η_{02}	32. η_{03}
33. η_{11}	34. η_{12}	35. η_{20}	36. η_{21}	37. η_{30}	38. ori		

Abbildung 3.5: Laufzeitverteilung einer Extraktion aller 38 Form-Merkmale einer SLIC-Segmentierung mit 800 Regionen. Die schwarzen Balken kennzeichen die einzelnen Berechnungskomplexitäten der Merkmale. Die blauen Balken hingegen demonstrieren den Laufzeitgewinn mittels Caching bei der Berechnung aller Merkmale. Es zeigt sich, dass bereits nach der Berechnung des zweiten Merkmals ein deutlicher Geschwindigkeitsgewinn zu vernehmen ist.

Orientierung oder die Exzentrizität gleichermaßen die Berechnung der translationsinvarianten Momente sowie deren Kovarianzmatrix. Auch bei Merkmalen, die nicht über die Momente beschrieben sind, wie zum Beispiel die Berechnung des minimalen Hüllkörpers und der relativen Schwerpunktskoordinaten, lassen sich Gemeinsamkeiten in der Berechnung wiederfinden, zum Beispiel über die Eckpunktkoordinaten des Hüllkörpers. Obwohl eine Berechnung aller Merkmale aufgrund der stetigen Wiederverwertung der Daten redundant erscheint und die Menge der zu benutzenden Merkmale für ein spezifisches Problem auf eine Untermenge reduziert werden sollte (vgl. Kapitel 6.1), zeigt es sich dennoch als lohnenswert, alle bereits errechneten Daten für die weiteren Berechnungen zu cachen. Abbildung 3.5 illustriert dabei den enormen Laufzeitgewinn gegenüber einer Berechnung ohne Verwendung eines Caches. Es ist jedoch anzumerken, dass der Laufzeitgewinn mit den Kosten eines erhöhten Speicheraufwands verbunden ist.

4 Räumliches Lernen auf Graphen

Das *räumliche Lernen auf Graphen* basiert auf der direkten Überführung der Definition einer Faltung auf zweidimensionalen Bildern bzw. regulären Gittern über ein räumlich verschiebbares Fenster. Ähnlich wie die Faltung in CNNs wird dafür über eine Anordnung der Knoten mit festgelegter Schrittweite iteriert und für diese jeweils eine feste Menge an Nachbarknoten bestimmt, die das Fenster bzw. das Receptive-Field eines jeden Knotens beschreiben [36]. Aufgrund der gleichmäßigen räumlichen Anordnung der Pixel bzw. Knoten auf einem regulären Gitter ist die Definition einer Abtastfolge der Knoten und einer jeweiligen Nachbarschaftsbestimmung mit fester Größe und Ordnung recht trivial, wohingegen solche Definitionen im Kontext von willkürlichen Graphen aufgrund ihrer gegebenenfalls fehlenden räumlichen (oder zeitlichen) Anordnung nicht gleich ersichtlich erscheinen.
Niepert u. a. definieren in ihrer Arbeit einen solchen räumlichen Faltungsoperator auf Graphen, der im Folgenden vorgestellt und anschließend in den Kontext von Graphen im zweidimensionalen euklidischen Raum überführt wird [36].

4.1 Grundlagen

Für die Definition eines räumlichen Faltungsoperators auf Graphen werden spezielle Konstrukte der Graphentheorie benötigt, die im Folgenden vorgestellt werden.

Färbung von Knoten. Eine *Knotenfärbung* ℓ ist eine Funktion $\ell\colon \mathcal{V} \to \mathcal{C}$ auf den Knoten eines Graphen $\mathcal{G}$, die jedem Knoten in $\mathcal{V}$ eine *Farbe* einer endlich abzählbaren Menge $\mathcal{C} \subseteq \mathbb{R}$ zuordnet [36]. Mit Hilfe der Knotenfärbung lässt sich folglich eine Ordungsrelation $>_\ell$ auf der Knotenmenge von $\mathcal{G}$ definieren, wobei $v_i >_\ell v_j$ genau dann, wenn $\ell(v_i) < \ell(v_j)$ [36]. Falls ℓ weiterhin injektiv ist, so spricht man von einer *totalen Ordnung* und die Knoten $v \in \mathcal{V}$ können insbesondere so permutiert werden, dass sie die Ordnung von $>_\ell$ respektieren [36].
Beispiele für eine Knotenfärbung sind Metriken, die die Wichtigkeit der einzelnen Knoten beschreiben. Eine naive Metrik dafür ist zum Beispiel der Knotengrad

deg bzw. d, der die *Zentralität* der Knoten beschreibt [36]. Komplexere Metriken für die Zentralität der Knoten sind unter anderem die Nähe, die Betweenness-Zentralität sowie die Eigenvektorzentralität [14, 36]. Letztere ist eng mit dem *PageRank*-Algorithmus von Google verwandt [14]. Die *Nähe* (engl. *Closeness*)

$$c(v) \coloneqq \frac{1}{\sum_{v_i \in \mathcal{V}, v \neq v_i} s(v, v_i)}$$

ist ein Maß für die durchschnittliche Länge zwischen einem Knoten und allen weiteren Knoten in $\mathcal{V}$ [14]. Je *näher* ein Knoten sich an der Knotenmenge befindet, als desto zentraler gilt er. Die *Betweenness-Zentralität* eines Knotens $v \in \mathcal{V}$ ist über

$$c(v) \coloneqq \sum_{v_i \neq v \neq v_j} \frac{\kappa_{ij}(v)}{\kappa_{ij}}$$

definiert, wobei $\kappa_{ij} \in \mathbb{N}$ die Anzahl an kürzesten Pfaden von v_i nach v_j angibt und $\kappa_{ij}(v) \in \mathbb{N}$ die Anzahl dieser Pfade beschreibt, die durch v führen [14]. Nach der Methode der *Eigenvektorzentralität* gilt ein Knoten als umso wichtiger, je wichtiger seine Nachbarknoten sind [14]. Sie ist definiert über

$$c(v) = \frac{1}{\lambda} \sum_{v_i \in \mathcal{N}(v)} c(v_i),$$

wobei $\lambda \in \mathbb{R}$. Mit der Darstellung der Eigenvektorzentralität $c \colon \mathcal{V} \to \mathbb{R}_+$ als Vektor $\mathbf{c} \in \mathbb{R}_+^N$ und $\mathcal{G}$ als ungewichtete Adjazenzmatrix $\mathbf{A} \in \{0,1\}^{N \times N}$ kann die Bestimmung von λ bzw. $\mathbf{c}$ als Eigenwertproblem $\mathbf{Ac} = \lambda \mathbf{c}$ aufgefasst werden. Dann beschreibt der größte Eigenwert λ die Eigenvektorzentralität der Knoten über dessen Eigenvektor $\mathbf{c}$ [14].

Isomorphie und kanonische Ordnung. Seien $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ und $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ zwei Graphen mit der gleichen Anzahl an Knoten, d.h. $|\mathcal{V}_1| = |\mathcal{V}_2|$. Dann heißt eine bijektive Abbildung $p \colon \mathcal{V}_1 \to \mathcal{V}_2$ *Isomorphismus* zwichen $\mathcal{G}_1$ und $\mathcal{G}_2$, falls $(v_i, v_j) \in \mathcal{E}_1$ genau dann, wenn $(p(v_i), p(v_j)) \in \mathcal{E}_2$ [33]. Zwei Graphen sind genau dann *isomorph* zueinander, wenn ein Isomorphismus zwischen ihnen existiert. Die Komplexität zur Bestimmung der Isomorphie zweier Graphen liegt in NP, wobei nicht bekannt ist, ob sie in P enthalten oder NP-vollständig ist [36].

Die Komposition mehrerer Isomorphismen ist ebenfalls ein Isomorphismus [33]. Die Menge aller Isomorphismen (zuzüglich der identischen Abbildung, die die Knoten auf sich selber abbildet) heißt die *Isomorphismenklasse* des Graphen [33].

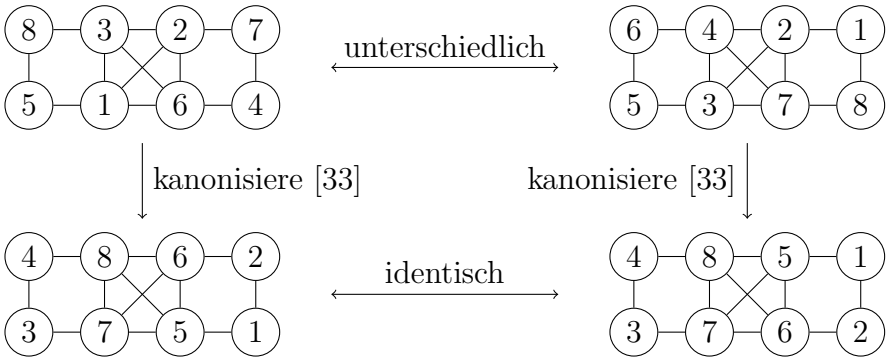


Abbildung 4.1: Illustration zweier isomorpher Graphen (oben), die jedoch nicht identisch sind (zum Beispiel sind die Knoten 1 und 5 im linken Graphen adjazent, aber nicht im rechten). Der Prozess zur Bestimmung einer kanonischen Ordnung sorgt dafür, dass zwei Graphen eine identische Knotenabbildung erhalten, falls sie isomorph zueinander sind (unten). Die Kanten der Graphen verweisen jeweils auf die gleichen Indizes, auch wenn sich die Darstellung bzw. Position der Knoten unterscheidet.

Eine *kanonische Ordnung* eines Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ist ein Graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ mit einer eindeutigen Ordnung seiner Knoten $\mathcal{V}'$, der isomorph zu $\mathcal{G}$ ist und seine gesamte Isomorphismenklasse repräsentiert [36]. Zwei Graphen sind insbesondere genau dann zueinander isomorph, wenn ihre kanonischen Ordnungen übereinstimmen [33]. Abbildung 4.1 illustriert das Prinzip der kanonischen Ordnung anhand zwei einfach gewählter zueinander isomorpher Graphen.
Ein Isomorphismus bzw. eine kanonische Ordnung kann ebenfalls eine Knotenfärbung ℓ berücksichtigen, indem Knoten durch einen Isomorphismus nur auf Knoten der gleichen Farbe abgebildet werden dürfen [33]. Das reduziert die Menge der verfügbaren Isomorphismenklassen und insbesondere die Komplexität ihrer Berechnung. Zur Berechnung der Isomorphismenklassen und der kanonischen Ordnung, auch unter Berücksichtigung einer Knotenfärbung, zeichnet sich das Programm `nauty` aufgrund dessen bemerkenswerter Berechnungslaufzeit aus (vgl. [33]).

4.2 Räumliche Faltung

Der räumliche Faltungsoperator von Niepert u. a. hat das Ziel, den Graphen in einen dreidimensionalen Tensor zu konvertieren, sodass die klassische Faltungsoperation conv2d auf diesen angewendet werden kann. Der Prozess zur *Vektorisierung* eines Graphen besteht dabei aus drei Schritten: (1) Bestimme eine Auswahl $\mathcal{V}_{\text{out}} \subseteq \mathcal{V}$, $|\mathcal{V}_{\text{out}}| = L$, der Knoten des Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ mit einer zuvor definierten Länge $L \in \mathbb{N}$, (2) finde für jeden ausgewählten Knoten $v \in \mathcal{V}_{\text{out}}$ eine Nachbarschaftsgrup-

Eingabe: Knotenmenge $\mathcal{V}$, Färbung $\ell\colon \mathcal{V} \to \mathcal{C} \subseteq \mathbb{R}$, Größe $L \in \mathbb{N}$, Schrittweite $S \in \mathbb{N}$
Ausgabe: geordnete Knotenmenge $\mathcal{V}_{\text{out}} \subseteq \mathcal{V}$ mit $|\mathcal{V}_{\text{out}}| \leq l$

```
V_out ← [ ]
V_sort ← Sortiere V absteigend bezüglich ℓ.
i, j ← 1
while i < L do
  if j ≤ |V_sort| then
    V_out[i] ← V_sort[j]
  end if
  i ← i + 1
  j ← j + S
end while
return V_out
```

Algorithmus 4.1: Berechnung der Knotenauswahl $\mathcal{V}_{\text{out}}$ anhand einer Knotenfärbung $\ell\colon \mathcal{V} \to \mathcal{C} \subseteq \mathbb{R}$, einer maximalen Größe $L \in \mathbb{N}$ von $\mathcal{V}_{\text{out}}$ sowie einer Schrittweite $S \in \mathbb{N}$.

pierung $\mathcal{N}_v \subseteq \mathcal{V}$, (3) normalisiere die jeweiligen Nachbarschaften $\mathcal{N}_v$ auf eine festgelegte Größe $K \in \mathbb{N}$ zu $\mathcal{N}_{\text{out}}$, $|\mathcal{N}_{\text{out}}| = K$, und bestimme auf diesen eine Ordnung, sodass Graphen mit ähnlichen Nachbarschaftsstrukturen eine äquivalente Ordnung erzeugen [36]. Im Folgenden werden die drei Schritte näher erläutert.

Knotenauswahl. Die Knotenauswahl beschreibt den Prozess zur Auswahl einer geordneten Teilmenge $\mathcal{V}_{\text{out}} \subseteq \mathcal{V}$ mit nicht mehr als $L \in \mathbb{N}$ vielen Knoten, für die im späteren Verlauf eine Nachbarschaft des Graphen bzw. ein Receptive-Field des Netzes erzeugt wird [36]. Algorithmus 4.1 veranschaulicht die grobe Vorgehensweise dieses Verfahrens. Dafür werden anhand einer gegebenen Knotenfärbung $\ell\colon \mathcal{V} \to \mathcal{C} \subseteq \mathbb{R}$ die Knoten in $\mathcal{V}$ absteigend sortiert, sodass Knoten, die aufgrund ihrer Zentralität als wichtiger gelten, am Anfang der geordneten Knotenmenge $\mathcal{V}_{\text{sort}}$ stehen. Folglich kann über die geordnete Knotenmenge $\mathcal{V}_{\text{sort}}$ iteriert werden und in Abständen bzw. Schrittweiten $S \in \mathbb{N}$ solange Knoten aus $\mathcal{V}_{\text{sort}}$ entnommen werden, bis L viele gefunden wurden oder die Menge zu Ende iteriert wurde. S gibt damit den Abstand zweier benachbarter Knoten in $\mathcal{V}_{\text{out}}$ an, für die ein Receptive-Field generiert wird. Falls die Liste eine kleinere Anzahl an Knoten als $S(L-1)+1$ besitzt, dann besitzt $\mathcal{V}_{\text{out}}$ folglich eine Kardinalität kleiner als L. Demnach ist es möglich, dass $\mathcal{V}_{\text{out}}$ weniger, aber niemals mehr Knoten als L enthält (vgl. [36]).

Nachbarschaftsgruppierung. Für jeden in $\mathcal{V}_{\text{out}} \subseteq \mathcal{V}$ enthaltenen Knoten muss nun ein Receptive-Field konstruiert werden. Dafür generiert Algorithmus 4.2 zuvor eine

Eingabe: Knoten $v \in \mathcal{V}_{\text{out}}$, Receptive-Field-Größe $K \in \mathbb{N}$
Ausgabe: Nachbarschaftsmenge $\mathcal{N}_v \subseteq \mathcal{V}$
$\mathcal{N}_v, \mathcal{T} \leftarrow \{v\}$
while $|\mathcal{N}_v| < K$ und $|\mathcal{T}| > 0$ **do**
$\mathcal{T} \leftarrow \bigcup_{v \in \mathcal{T}} \mathcal{N}(v, 1)$
$\mathcal{N}_v \leftarrow \mathcal{N}_v \cup \mathcal{T}$
end while
return $\mathcal{N}_v$

Algorithmus 4.2: Berechnung einer Vorauswahl der maximal in dem Receptive-Field des Knotens $v \in \mathcal{V}_{\text{out}}$ mit Größe $K \in \mathbb{N}$ enthaltenen Knoten über eine von v ausgehende Breitensuche.

Nachbarschaftsgruppierung $\mathcal{N}_v \subseteq \mathcal{V}$ eines jeden Knoten $v \in \mathcal{V}_{\text{out}}$, die eine Vorauswahl der maximal in dessen Receptive-Field enthalteden Knoten beschreibt. Für einen Knoten $v \in \mathcal{V}_{\text{out}}$ und eine Receptive-Field-Größe $K \in \mathbb{N}$ vollzieht der Algorithmus eine von v ausgehende Breitensuche durch den Graphen, bei der die Knoten in Abhängigkeit zu ihrem Abstand zu v durchlaufen werden. Dabei werden zuerst die lokalen Nachbarknoten $v_i \in \mathcal{V}$ mit $(v, v_i) \in \mathcal{E}$ zu der Menge $\mathcal{N}_v \in \mathcal{V}$ hinzugefügt. Falls $|\mathcal{N}_v| < K$ werden die Nachbarschaften der zuletzt zu $\mathcal{N}_v$ hinzugefügten Knoten durchlaufen und zu $\mathcal{N}_v$ hinzugefügt. Der Prozess wiederholt sich solange, bis mindestens K Knoten gefunden wurden oder keine Knoten mehr zur Verfügung stehen, für die ein Weg nach v existiert und noch nicht in der Nachbarschaftsgruppierung enthalten sind. $\mathcal{N}_{\text{v}}$ erzeugt dabei in den allermeisten Fällen eine Menge, die sich in ihrer Anzahl stark von K unterscheidet (vgl. [36]).

Normalisierung. Die Nachbarschaftsgruppierung $\mathcal{N}_v$ eines Knoten $v \in \mathcal{V}_{\text{out}}$ wird im Anschluss darauf zu einem Receptive-Field $\mathcal{N}_{\text{out}}$ mit einer festen Größe $|\mathcal{N}_{\text{out}}| = K$ und einer totalen Ordnung auf dessen Elementen *normalisiert*, sodass Knoten in zwei verschiedenen Graphen genau dann an eine ähnliche Position im Receptive-Field gelangen, wenn sich ihre strukturellen Rollen in den Graphen ähneln [36]. Die Normalisierung ist damit ein Prozess, der eine ungeordnete Knotenmenge in einen Vektorraum mit linearer Ordnung transformiert [36].

Algorithmus 4.3 veranschaulicht den von Niepert u. a. entwickelten Normalisierungsprozess bezüglich eines Knotens $v \in \mathcal{V}_{\text{out}}$ basierend auf dessen zuvor ermittelter Nachbarschaftsmenge $\mathcal{N}_v$. Dafür werden die Knoten der Nachbarschaftsmenge $v_i \in \mathcal{N}_v$ zuerst aufsteigend bezüglich ihrer Abstände $s(v, v_i)$ zu v und absteigend bezüglich ihrer jeweiligen Knotenfarbe bei gleichen Abständen sortiert. Falls v_i näher an v als v_j liegt, d.h. $s(v, v_i) < s(v, v_j)$, befindet sich v_i immer vor v_j in der

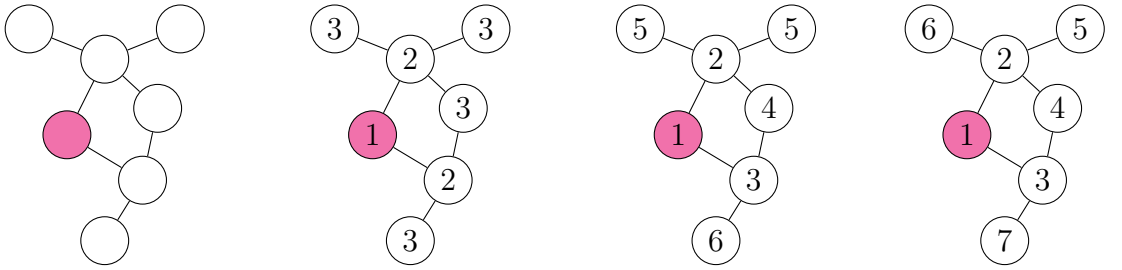
Eingabe: Graph $\mathcal{G}$, Knoten $v \in \mathcal{V}_{\text{out}}$, Nachbarschaftsmenge $\mathcal{N}_v \subseteq \mathcal{V}$, Receptive-Field-Größe $K \in \mathbb{N}$, Färbung $\ell \colon \mathcal{V} \to \mathcal{C} \subseteq \mathbb{R}$
Ausgabe: geordnete Nachbarschaftsmenge $\mathcal{N}_{\text{out}}$ mit $|\mathcal{N}_{\text{out}}| = K$
berechne Ordnung $>_v$, sodass $v_i >_v v_j$ genau dann, wenn $s(v, v_i) > s(v, v_j)$ und $\ell(v_i) < \ell(v_j)$ falls $s(v, v_i) = s(v, v_j)$.
$\mathcal{N}_{\text{sort}} \leftarrow$ Sortiere $\mathcal{N}_v$ aufsteigend bezüglich $>_v$.
if $|\mathcal{N}_v| > K$ **then**
  $\mathcal{N}_{\text{sort}} \leftarrow \mathcal{N}_{\text{sort}}[0 : K]$
end if
$\mathcal{G}' = (\mathcal{V}', \mathcal{E}') \leftarrow$ Generiere Teilgraph aus $\mathcal{G}$ mit Knotenmenge $\mathcal{V}' \coloneqq \mathcal{N}_{\text{sort}} \subseteq \mathcal{V}$.
$\mathcal{N}_{\text{out}} \leftarrow$ Kanonisiere $\mathcal{G}'$ bezüglich $>_v$ (hebt Gleichheiten in $>_v$ auf).
return $\mathcal{N}_{\text{out}} \cup (\max(K - |\mathcal{N}_v|, 0)$ Fakeknoten)

Algorithmus 4.3: Berechnung der Vektorrepräsentation einer Nachbarschaftsmenge $\mathcal{N}_v$ eines Knotens $v \in \mathcal{V}_{\text{out}}$ mittels einer vordefinierten Größe $K \in \mathbb{N}$, einer Färbung $\ell \colon \mathcal{V} \to \mathcal{C} \subseteq \mathbb{R}$ sowie der Abstandsfunktion $s \colon \mathcal{V} \times \mathcal{V} \to \mathbb{N}$ des Graphen $\mathcal{G}$.

resultierenden normalisierten Nachbarschaftsmenge $\mathcal{N}_{\text{out}}$. Damit befindet sich der Wurzelknoten v stets an erster Position in $\mathcal{N}_{\text{out}}$ und je weiter ein Knoten von v entfernt liegt, umso höher ist sein Index in der Vektorrepräsentation. Anschließend kann die ermittelte Sortierung auf die Größe K reduziert werden, indem die Elemente ab der Position $K + 1$ verworfen werden.

Da die meisten Knotenfärbungsalgorithmen ℓ nicht injektiv sind und daher möglicherweise auf die gleiche Farbe innerhalb einer „Abstandsgruppe“ abbilden, ist die generierte Sortierung $\mathcal{N}_{\text{sort}}$ insbesondere nicht total. Es ist daher notwendig, die Äquivalenzen dieser Knotenfarbe aufzulösen. Für diesen Zweck wird ein Teilgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ aus $\mathcal{G}$ generiert, der $\mathcal{G}$ auf die Nachbarschaftsmenge $\mathcal{V}' \coloneqq \mathcal{N}_{\text{sort}}$ beschränkt. Eine kanonische Ordnung auf diesem Graphen mit Berücksichtigung der Knotenfärbung ℓ liefert damit eine totale Ordnung der Knoten in $\mathcal{N}_{\text{sort}}$ mit gleicher Knotenfarbe. Eine kanonische Ordnung auf Graphen mit fester Größe K ist dabei in linearer Zeit möglich (vgl. [36]). Da die Kanonisierung des Graphen nur für Knoten mit gleicher Knotenfarbe angewendet werden muss, zeigt sich der zusätzliche Aufwand dieser Operation als zu vernachlässigen.

Abschließend muss weiterhin der Fall betrachtet werden, dass $\mathcal{N}_v$ weniger als K Knoten enthält. Dafür werden $\max(K - |\mathcal{N}_v|, 0)$ *Fakeknoten* an das Ende von $\mathcal{N}_{\text{out}}$ gehängt, denen keine Bedeutung innewohnt und später so belegt werden, dass diese keinen Einfluss auf das Resultat der Faltung nehmen. Abbildung 4.2 illustriert den Normalisierungsprozess an einem einfachen Beispiel.



(a) $\mathcal{G}' = (\mathcal{N}_v, \mathcal{E}')$ (b) Distanz (c) Zentralität (d) Kanonisierung

Abbildung 4.2: Illustration der Normalisierung einer Nachbarschaft eines Knotens (rot) mit 7 Knoten (a) zur Bestimmung einer eindeutigen Anordnung dieser. Dazu werden die Nachbarschaftsknoten zuerst auf Basis ihrer Distanz zum Wurzelknoten gruppiert (b). Innerhalb einer Gruppierung werden die Knoten auf Basis einer gegebenen Zentralitätsmetrik sortiert (c). Im Anschluss werden Äquivalenzen in der Zentralität innerhalb einer Gruppe über dessen kanonische Ordnung aufgelöst (d). Gegebenenfalls muss die gefundene Ordnung auf die gewünschte Größe des Receptive-Fields zugeschnitten oder um Fakeknoten erweitert werden.

Für jeden Graphen $\mathcal{G}$ kann damit ein dreidimensionaler Tensor $\mathbf{T} \in \mathbb{R}^{L \times K \times M}$ über die drei beschriebenen Schritte mit Hilfe einer Merkmalsmatrix $\mathbf{F} \in \mathbb{R}^{N \times M}$ auf den Knoten des Graphen generiert werden. Dafür werden L Knoten des Graphen ausgewählt und für diese die entsprechenden K großen Nachbarschaften bzw. Receptive-Fields ermittelt. Jeder Eintrag in $\mathcal{N}_{\text{out}}$ wird dann durch seine entsprechenden Merkmale aus $\mathbf{F}$ ersetzt. Das erlaubt die Verwendung der klassischen conv2d-Operation im Kontext von Graphen, die auf dreidimensionalen Tensoren operiert. Damit entspricht die Transformation eines Graphen in den Vektorraum und einer anschließenden Faltung auf dieser Repräsentation der ersten Faltungsschicht eines klassischen CNNs. Für eine Knotenauswahl, die weniger als L Knoten ermittelt, sowie für Receptive-Fields, die Fakeknoten enthalten, werden die entsprechenden Einträge in $\mathbf{T}$ über den Wert Null repräsentiert, welche somit keinen Einfluss auf die Faltung mittels conv2d nehmen.

4.3 Erweiterung auf Graphen im zweidimensionalen Raum

Niepert u. a. haben einen Algorithmus zur Generierung von Receptive-Fields über den Nachbarschaften eines Graphen vorgestellt, der effizient implementiert werden kann und über einer Reihe von Testdatensätzen konkurrenzfähige Resultate im Ver-

gleich zu Methoden, die auf *Graphkernen* basieren, liefert (vgl. [36]). Dabei hängt die Ordnung der Knotenauswahl und der Receptive-Fields stark von einer gewählten Zentralitätsmetrik ab, die auf einem allgemeinen Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ eine fehlende räumliche oder zeitliche Ordnung ersetzt. Im Kontext von Graphen im zweidimensionalen euklidischen Raum $\mathcal{G} = (\mathcal{V}, \mathcal{E}, p)$, bei denen den Knoten jeweils eine eindeutige Position in der Ebene zugeordnet werden kann, kann die Knotenauswahl $\mathcal{V}_{\text{out}}$ (Algorithmus 4.1) sowie die Normalisierung der Nachbarschaftsmenge $\mathbb{N}_v$ eines Knotens $v \in \mathcal{V}_{\text{out}}$ (Algorithmus 4.3) diese jedoch folglich berücksichtigen. Dies erscheint insbesondere dann sinnvoll, wenn sich ein Graph durch viele gleichwertig zentrale Knoten auszeichnet. So bildet ein Graph, der durch eine SLIC-Segmentierung generiert wurde, (in etwa) auf ein unstrukturiertes Gitter ab. Eine Zentralitätsmetrik auf diesem Graphen besitzt so gut wie keine Aussagekraft. Der Graph zeichnet sich dabei eher durch seine eindeutige Knotenlage in der Ebene aus.
Die Knotenauswahl wird folglich über die Anordnung der Knoten bezüglich ihrer *Scanline*-Ordnung beschrieben, d.h. von oben nach unten und von links nach rechts. Formal lässt sich dafür die Knotenfärbung $\ell_{\text{scanline}} \colon \mathcal{V} \to \mathbb{R}$ über

$$\ell_{\text{scanline}}(v)^{-1} \coloneqq w_{\max}\, \hat{p}(v)_1 + \hat{p}(v)_2$$

mit dem maximalen Breitenabstand $w_{\max} \coloneqq \max_{v_i, v_j \in \mathcal{V}} \big(p(v_i)_2 - p(v_j)_2\big)$ und der normalisierten Position

$$\hat{p}(v) \coloneqq \left[\left\lfloor \frac{p(v)_1 - y_{\min}}{\delta} \right\rfloor, p(v)_2 - x_{\min} \right]^\top$$

mit $y_{\min} \coloneqq \min(p(v_1)_1, \ldots, p(v_N)_1)$ bzw. $x_{\min} \coloneqq \min(p(v_1)_2, \ldots, p(v_N)_2)$ und $\delta \in \mathbb{R}_+$ definieren, sodass Knoten einen höheren Farbwert erhalten, umso geringer ihre y-Koordinate ist und bei annähernd gleicher y-Koordinate ihre x-Koordinate die Farben weiter differenziert. Dafür werden die jeweiligen Positionen $p(v)$ mittels $\hat{p}(v)$ in den positiven reellen Raum transformiert. Die y-Koordinate von $\hat{p}$ wird weiter auf eine über δ steuerbare natürliche Zahl gerundet. Dadurch werden kleine Fluktuationen in den y-Koordinaten ausgeglichen, die ansonsten eine vollkommen willkürliche Ordnung erzeugen würden. Die y-Koordinate von $\hat{p}(v)$ wird in ℓ_{scanline} daraufhin insofern gewichtet, dass für zwei unterschiedliche y-Koordinaten die Ordnung, die die Knotenfärbung generiert, nicht von der x-Koordinate beeinflusst wird. Damit entspricht der Algorithmus 4.1 mit der Knotenfärbung ℓ_{scanline} in etwa der Pixelauswahl einer klassischen Faltung auf regulären Gittern, mit dem Unterschied, dass die Knotenauswahl letztendlich eine eindimensionale geordnete Menge definiert. Das ist

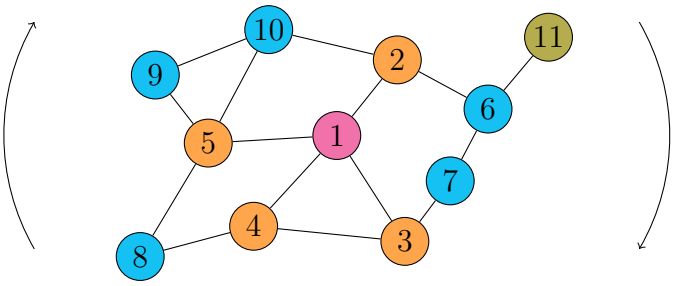


Abbildung 4.3: Normalisierung eines Receptive-Fields der Größe 11 eines Graphen im zweidimensionalen euklidischen Raum. Knoten werden basierend bezüglich ihrer Pfadlänge zum Wurzelknoten (rot) gruppiert, hier dargestellt über die unterschiedlichen Farbtöne der Knoten. In diesen werden die Knoten entsprechend ihrer Winkel zum Wurzelknoten im Uhrzeigersinn geordnet.

aufgrund der möglichen Irregularitäten in den jeweiligen Knotenpositionen allerdings nicht zu vermeiden. Folglich sind benachbarte Knoten in $\mathcal{V}_{\text{out}}$ nicht zwangsläufig auch in der Ebene benachbart. ℓ_{scanline} generiert jedoch eine Knotenfärbung, die für zwei strukturell gleiche Graphen eine ähnliche Ordnung generiert.

Analog zu einem klassischen CNN lässt sich auch die Ordnung des Receptive-Fields bzw. die Normalisierung der Nachbarschaftsmenge $\mathcal{N}_v$ eines Knotens $v \in \mathcal{V}_{\text{out}}$ gestalten. Dafür wird eine Knotenfärbung $\ell_v \colon \mathcal{V} \to \mathbb{R}$ bestimmt, die die Knoten entsprechend ihrer Winkel zum Wurzelknoten $v \in \mathcal{V}$ ordnet. Sei dafür $\varphi_v \colon \mathcal{V} \to (0, 2\pi]$ analog zu (3.2) gegeben als

$$\varphi_v(v_i) \coloneqq \operatorname{atan2}(p(v_i)_1 - p(v)_1, p(v_i)_2 - p(v)_2) + \pi.$$

Dann kann ℓ_v bezüglich eines Wurzelknotens $v \in \mathcal{V}$ über

$$\ell_v(v_i)^{-1} \coloneqq \frac{d_{\max}}{\varphi_{\min}} \varphi_v(v_i) + \|p(v_i) - p(v)\|_2$$

definiert werden, wobei $d_{\max} \coloneqq \max_{v_i \in \mathcal{V}} \|p(v_i) - p(v)\|_2$ den maximalen Abstand eines Knotens zum Wurzelknoten und $\varphi_{\min} \coloneqq \min_{v_i, v_j \in \mathcal{V},\, \varphi_v(v_i) \neq \varphi_v(v_j)} |\varphi_v(v_i) - \varphi_v(v_j)|$ den minimalen, echt positiven Winkelabstand zweier Knoten beschreibt. Damit werden Knoten entsprechend ihrer Winkel zum Wurzelknoten v sortiert. Besitzen zwei Knoten den exakt gleichen Winkel, wird über den Abstand ihrer Positionen zum Wurzelknoten der Farbwert weiter differenziert. Die Gewichtung des Winkels über $d_{\max}/\varphi_{\min}$ sorgt dafür, dass für zwei unterschiedliche Winkel $\varphi_v(v_i) \neq \varphi_v(v_j)$ die Distanzen der Knoten zum Wurzelknoten keinen Einfluss nehmen.

Damit ist ℓ_v für alle Graphen injektiv, für die die Positionsfunktion p injektiv ist.

Folglich verfällt bei einer injektiven Positionsabbildung insbesondere die Notwendigkeit der Berechnung einer kanonischen Ordnung des Teilgraphs (vgl. Algorithmus 4.3). Mit der Einschränkung, dass für $\mathcal{G}(\mathcal{V}, \mathcal{E}, p)$ keine zwei Knoten auf eine gleiche Position abgebildet werden, können damit niemals Äquivalenzen in der Knotenfärbung ℓ_v existieren. Abbildung 4.3 verdeutlicht die entstehende Ordnung, die nach der Anwendung des Normalisierungsalgorithmus 4.3 bei Benutzung von ℓ_v entsteht.
Mit ℓ_{scanline} und ℓ_v stehen uns damit für die Knotenauswahl bzw. der Normalisierung zwei Knotenfärbungen zur Verfügung, die uns erlauben, den räumlichen Faltungsoperator auf Graphen im zweidimensionalen euklidischen Raum unter Berücksichtigung ihrer Knotenpositionen zu benutzen.

4.4 Netzarchitektur

Der räumliche Faltungsoperator bildet die erste Faltungsschicht eines neuronalen Netzes auf Graphen [36]. Dabei kann auf dem berechneten dreidimensionale Tensor der Größe $|\mathcal{V}_{\text{out}}| \times |\mathcal{N}_{\text{out}}| \times M$ des Graphen, der zu jeder Knotenauswahl dessen Nachbarschaftsmenge auf dessen M Mermale der Merkmalsmatrix $\mathbf{F} \in \mathbb{R}^{N \times M}$ abbildet, mit Hilfe der klassischen Faltungsoperation conv2d gefaltet werden (vgl. Abbildung 4.4). Es ist jedoch insbesondere darauf zu achten, nicht über den Grenzen des Receptive-Fields hinaus zu falten. Insbesondere sollte eine Faltung entlang der Knotenauswahl $\mathcal{V}_{\text{out}}$ vermieden werden, da dessen Knotenanordnung im Kontext einer Faltung keine Bedeutung besitzt, denn zwei „übereinanderliegende“ Receptive-Fields müssen nicht zwangsläufig im Graphen benachbart sein. Die conv2d-Operation operiert daher mit der Filtergröße und Schrittweite $1 \times |\mathcal{N}_{\text{out}}|$ [36]. Die restliche Struktur des Netzes kann nach Belieben gewählt werden [36]. Üblicherweise wird der Ausgabetensor der Faltungsschicht dafür zu einem Vektor umsortiert, sodass vollverbundenen Schichten hin zur Ausgabe an das Ergebnis der Faltungsschicht gestapelt werden können. Abbildung 4.4 veranschaulicht die typische räumliche Netzarchitektur auf Graphen.
Es ist weiterhin vorstellbar, auch mehrmals in einem Receptive-Field auf analoge Weise zu dem *Fire Module* des *SqueezeNet* zu falten (vgl. [24]). So können zum Beispiel über eine Faltung mit Filtergröße und Schrittweite 1×1 zuvor neue Merkmale aus den Eingabedaten generiert werden und im Anschluss die beschriebene Faltung über das gesamte Receptive-Field vollzogen werden.
Die räumliche Netzarchitektur erlaubt aufgrund ihrer Architektur nur eine Faltungs-

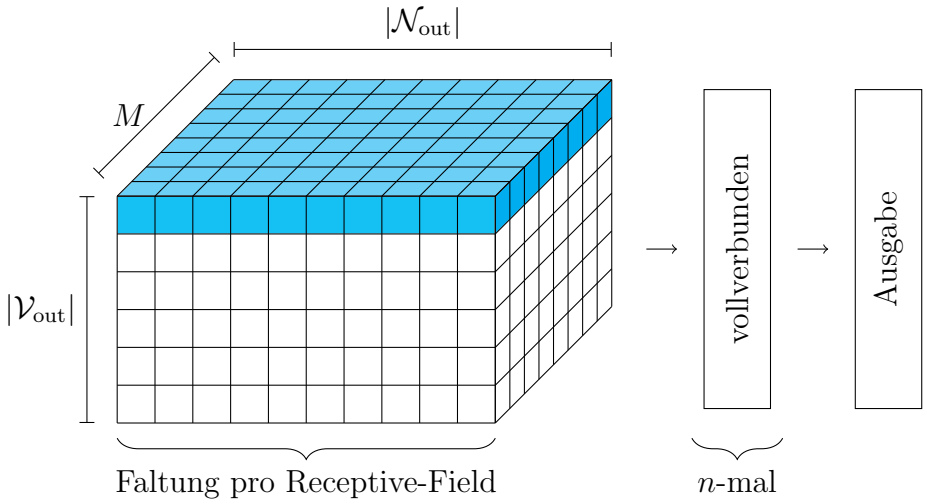


Abbildung 4.4: Typische räumliche Netzarchitektur auf Graphen. Der „Quader“, der durch die Anordnung der Knotenauswahl $\mathcal{V}_{\text{out}}$, ihrer Nachbarschaften $\mathcal{N}_{\text{out}}$ und entsprechenden Merkmale in $\mathbf{F} \in \mathbb{R}^{N \times M}$ entsteht, wird entlang der jeweiligen Nachbarschaften bzw. ihrer Merkmale gefaltet (blauer Quader). Eine Faltung entlang der Knotenauswahl $\mathcal{V}_{\text{out}}$ besitzt allerdings keine Bedeutung und ist deshalb zu vermeiden. Im Anschluss können vollverbundene Schichten hin zur Ausgabe an den umsortierten Vektor des Quaders gestapelt werden.

schicht in einem neuronalen Netz. Damit sind insbesondere die für das Deep-Learning typischen gestapelten Faltungen mit Poolingoperationen nicht möglich. Der Ansatz des spektralen Lernens auf Graphen, welcher in Kapitel 5 vorgestellt wird, überwindet dieses Problem.

5 Spektrales Lernen auf Graphen

Das *spektrale Lernen auf Graphen* bzw. die Formulierung eines spektralen Faltungsoperators auf Graphen basiert auf der spektralen Graphentheorie, d.h. der Betrachtung des Spektrums eines Graphen definiert über dessen Eigenwerte. Merkmale auf den Knoten eines Graphen können über das Spektrum analog zur Fourier-Transformation in dessen Frequenzraum zerlegt und wieder retransformiert werden. Diese Transformation erlaubt damit die fundamentale Formulierung eines Faltungsoperators in der spektralen Domäne des Graphen. Da der so definierte spektrale Faltungsoperator insbesondere rotationsinvariant ist, wird dieser im Verlauf des Kapitels für den Kontext von Graphen im euklidischen Raum modifiziert.

Durch die spektrale Formulierung kann weiterhin ein effizientes Pooling auf Graphen formuliert werden, welches uns erlaubt, Netzarchitekturen auf Graphen völlig analog zu klassischen CNNs auf zweidimensionalen Bildern zu generieren.

5.1 Spektrale Graphentheorie

Die spektrale Graphentheorie beschäftigt sich mit der Konstruktion, Analyse und Manipulation von Graphen. Sie beweist sich dabei als besonders nützlich in Anwendungsgebieten wie der Charakterisierung von Expandergraphen, dem Graphenzeichnen oder dem spektralen Clustering (vgl. [44]). Weiterhin hat die spektrale Graphentheorie zum Beispiel auch Anwendungsgebiete in der Chemie, bei der die Eigenwerte des Spektrums des Graphen mit der Stabilität von Molekülen assoziert werden (vgl. [5]).

Es zeigt sich, dass die Eigenwerte des Spektrums eines Graphen eng mit den Eigenschaften eines Graphen verwandt sind. Als spektrale Graphentheorie versteht man damit insbesondere die Studie über die gemeinsamen Beziehungen dieser beiden Bereiche. Dieses Kapitel gibt eine Einführung in die wichtigsten Definitionen und Intuitionen der spektralen Graphentheorie, die es uns schlussendlich erlauben, die spektrale Faltung auf Graphen zu formulieren.

5.1.1 Eigenwerte und Eigenvektoren reell symmetrischer Matrizen

Das *Eigenwertproblem* einer Matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$ ist definiert als $\mathbf{Mu} = \lambda \mathbf{u}$, wobei $\mathbf{u} \in \mathbb{R}^N$, $\mathbf{u} \neq \mathbf{0}$ *Eigenvektor* und $\lambda \in \mathbb{R}$ der entsprechende *Eigenwert* zu $\mathbf{u}$ genannt werden [21]. Ein Eigenvektor $\mathbf{u}$ beschreibt damit einen Vektor, dessen Richtung durch die Abbildung $\mathbf{Mu}$ nicht verändert, sondern lediglich um den Faktor λ skaliert wird. Zu einem Eigenwert λ gibt es unendlich viele (skalierte) Eigenvektoren $\mathbf{u}$. Wir definieren den Eigenvektor $\mathbf{u}$ eines Eigenwertes λ daher eindeutig über die Bedingung $\|\mathbf{u}\|_2 = 1$.

Sei $\mathbf{M}$ weiterhin symmetrisch, d.h. $\mathbf{M} = \mathbf{M}^\top$ [21]. Dann gilt für zwei unterschiedliche Eigenvektoren $\mathbf{u}_1$ und $\mathbf{u}_2$, dass diese orthogonal zueinander stehen, d.h. $\mathbf{u}_1 \perp \mathbf{u}_2$, und $\mathbf{M}$ genau N reelle Eigenwerte mit $\{\lambda_n\}_{n=1}^N$ hat [21]. Wir definieren demnach zu $\mathbf{M}$ die orthogonale *Eigenvektormatrix* $\mathbf{U} \coloneqq [\mathbf{u}_1, \dots, \mathbf{u}_n] \in \mathbb{R}^{N \times N}$, d.h. $\mathbf{U}\mathbf{U}^\top = \mathbf{U}^\top\mathbf{U} = \mathbf{I}$, und dessen Eigenwertdiagonalmatrix $\mathbf{\Lambda} \coloneqq \operatorname{diag}\big([\lambda_1, \dots, \lambda_N]^\top\big)$, d.h. $\mathbf{\Lambda}_{ii} = \lambda_i$ [8]. Dann gilt $\mathbf{MU} = \mathbf{U\Lambda}$ und insbesondere ist $\mathbf{M}$ diagonalisierbar über [21]

$$\mathbf{M} = (\mathbf{MU})\mathbf{U}^\top = (\mathbf{U\Lambda})\mathbf{U}^\top.$$

Weiterhin gilt für die k-te Potenz von $\mathbf{M}$, $k \in \mathbb{N}$, [26]

$$\mathbf{M}^k = \left(\mathbf{U\Lambda U}^\top\right)^k = \mathbf{U\Lambda}^k\mathbf{U}^\top \tag{5.1}$$

aufgrund der Induktion $(k-1 \to k)$

$$\left(\mathbf{U\Lambda U}^\top\right)^k = \left(\mathbf{U\Lambda U}^\top\right)^{k-1}\mathbf{U\Lambda U}^\top = \mathbf{U\Lambda}^{k-1}\mathbf{U}^\top\mathbf{U\Lambda U} = \mathbf{U\Lambda}^k\mathbf{U}^\top.$$

Falls $\mathbf{M}$ weiterhin *schwach diagonaldominant* ist, d.h.

$$\sum_{\substack{j=1 \\ j \neq i}}^{N} |\mathbf{M}_{ij}| \leq |\mathbf{M}|_{ii}, \tag{5.2}$$

und weiterhin $\mathbf{M}_{ii} \geq 0$ für alle $i \in \{1, \dots, N\}$, dann ist $\mathbf{M}$ *positiv semidefinit*, d.h. $\mathbf{x}^\top\mathbf{Mx} \geq 0$ für alle $\mathbf{x} \in \mathbb{R}^N$ [21]. Eigenwerte symmetrischer positiv semidefiniter Matrizen $\lambda_i \in \mathbb{R}_+$ sind positiv reell und es lässt sich folglich auf diesen eine Ordnung definieren mit $0 \leq \lambda_1 \leq \dots \leq \lambda_N \coloneqq \lambda_{\max}$ [21].

5.1.2 Laplace-Matrix

Die Laplace-Matrix ist in der spektralen Graphentheorie eine Matrix, die die Beziehungen der Knoten und Kanten eines beliebigen Graphen $\mathcal{G}$ in einer generalisierten und normalisierten Form beschreibt. Viele der Eigenschaften von $\mathcal{G}$ können durch die Eigenwerte ihrer Laplace-Matrix beschrieben werden, wohingegen dies beispielsweise für die Eigenwerte der Adjazenzmatrix $\mathbf{A}$ von $\mathcal{G}$ nur bedingt zutrifft und insbesondere nicht verallgemeinbar für beliebge Graphen ist [5]. Dies ist vor allem dem Fakt geschuldet, dass die Eigenwerte der Laplace-Matrix konsistent sind mit den Eigenwerten des Laplace-Beltrami Operators ∇^2 in der spektralen Geometrie [5]. Die Laplace-Matrix ist damit ein geeignetes Mittel zur Betrachtung und Analyse eines Graphen.
Für einen schleifenlosen, ungerichteteten, gewichteten oder ungewichteten Graphen $\mathcal{G}$ und dessen Adjazenzmatrix $\mathbf{A}$ mit Gradmatrix $\mathbf{D}$ ist die *kombinatorische Laplace-Matrix* $\mathbf{L}$ definiert als $\mathbf{L} \coloneqq \mathbf{D} - \mathbf{A}$ [5]. Die *normalisierte Laplace-Matrix* $\tilde{\mathbf{L}}$ ist definiert als $\tilde{\mathbf{L}} \coloneqq \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$ mit der Konvention, dass $\mathbf{D}_{ii}^{-1/2} = 0$ für isolierte Knoten $v_i \in \mathcal{V}$ in $\mathcal{G}$, d.h. $\mathbf{D}_{ii} = 0$ [5]. Daraus ergibt sich die elementweise Definition

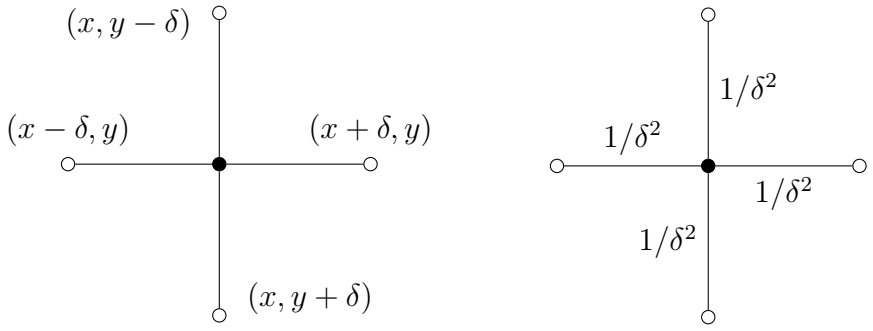
$$\tilde{\mathbf{L}}_{ij} \coloneqq \begin{cases} 1, & \text{wenn } i = j, \\ -\frac{w(v_i, v_j)}{\sqrt{d(v_i)d(v_j)}}, & \text{wenn } v_j \in \mathcal{N}(v_i), \\ 0, & \text{sonst.} \end{cases}$$

Für zusammenhängende Graphen kann $\tilde{\mathbf{L}}$ vereinfacht werden zu [5]

$$\tilde{\mathbf{L}} \coloneqq \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}. \tag{5.3}$$

Jeder Eintrag auf der Diagonalen der normalisierten Laplace-Matrix ist folglich Eins. $\tilde{\mathbf{L}}$ ist damit normalisiert auf den (gewichteten) Grad zweier adjazenter Knoten v_i und v_j. Es ist anzumerken, dass $\mathbf{L}$ und insbesondere $\tilde{\mathbf{L}}$ symmetrisch sind, wohingegen eine Normalisierung der Form $\mathbf{D}^{-1}\mathbf{L}$ dies in der Regel nicht wäre [39]. $\mathbf{L}$ und $\tilde{\mathbf{L}}$ sind desweiteren keine ähnlichen Matrizen, insbesondere sind ihre Eigenvektoren verschieden. Die Nutzung von $\mathbf{L}$ oder $\tilde{\mathbf{L}}$ ist damit abhängig von dem Problem, welches man betrachtet [17]. Wir schreiben $\mathcal{L}$ wenn die Wahl der Laplace-Matrix, ob $\mathbf{L}$ oder $\tilde{\mathbf{L}}$, für die weitere Berechnung irrelevant ist.

Interpretation. Sei $f\colon \mathcal{V} \to \mathbb{R}$ bzw. $\mathbf{f} \in \mathbb{R}^N$ mit $f(v_i) = \mathbf{f}_i$ eine Funktion bzw. ein Signal auf den Knoten eines Graphen $\mathcal{G}$. Dann kann für die kombinatorische



(a) Gitterrepräsentation (b) Graphrepräsentation

Abbildung 5.1: Illustration des 5-Punkte-Sterns in zwei Dimensionen mit gleicher Approximation des ∇^2 Operators (bei umgekehrtem Vorzeichen), einmal mit der 5-Punkte-Stern Approximation auf regulären Gittern (a) und einmal mit der kombinatorischen Laplace-Matrix $\mathbf{L}$ auf Graphen (b).

Laplace-Matrix $\mathbf{L}$ verifiziert werden, dass sie die Gleichung

$$(\mathbf{Lf})_i = \sum_{v_j \in \mathcal{N}(v_i)} w(v_i, v_j)(\mathbf{f}_i - \mathbf{f}_j)$$

erfüllt [17]. Sei $\mathcal{G}$ nun ein Graph, der aus einem (unendlichen) zweidimensionalen regulärem Gitter entstanden ist, d.h. jeder Knoten v_i besitzt genau vier achsenparallele rechtwinklige Nachbarn mit gleichen Kantengewichten $1/\delta^2$, wobei $\delta \in \mathbb{R}$ den Abstand der Knoten zueinander beschreibt. Zur einfacheren Veranschaulichung benutzen wir dabei für die Signalstärke $\mathbf{f}_i$ eines Knoten v_i an Position (x, y) die Indexnotation $\mathbf{f}_{x,y}$. Dann beschreibt

$$(\mathbf{Lf})_{x,y} = \frac{4\mathbf{f}_{x,y} - \mathbf{f}_{x+1,y} - \mathbf{f}_{x-1,y} - \mathbf{f}_{x,y+1} - \mathbf{f}_{x,y-1}}{h^2}$$

die *5-Punkte-Stern* Approximation $\nabla^2 f$ (bei umgekehrtem Vorzeichen) definiert auf den Punkten $\{(x, y), (x+\delta, y), (x-\delta, y), (x, y+\delta), (x, y-\delta)\}$ [17] (vgl. Abbildung 5.1). Ähnlich zu einem regulären Gitter lässt sich ein Graph $\mathcal{G}$ auch über beliebig viele Abtastpunkte einer differenzierbaren Mannigfaltigkeit konstruieren. Es zeigt sich, dass mit steigender Abtastdichte und geeigneter Wahl der Kantengewichte die normalisierte Laplace-Matrix $\tilde{\mathbf{L}}$ zu dem kontinuierlichen Laplace-Beltrami Operator ∇^2 konvergiert [17]. Damit kann $\tilde{\mathbf{L}}$ als die diskrete Analogie des ∇^2 Operators auf Graphen verstanden werden. Der Laplace-Beltrami Operator $\nabla^2 f(p)$ misst dabei, in wie weit sich eine Funktion f an einem Punkt p von dem Durchschnitt aller Funktionspunkte um einen kleinen Bereich um p unterscheidet. Die Laplace-Matrix operiert dabei völlig analog, indem sie misst, wie sehr sich eine (diskrete) Funktion

um einen Knoten im Vergleich zu seinen Nachbarknoten unterscheidet.
Die Eigenwerte und Eigenvektoren von $\mathcal{L}$ helfen uns beim Verständnis der linearen Transformation einer Funktion $\mathbf{f}$ (mehrfach) angewendet auf $\mathcal{L}$. Wir können dafür $\mathbf{f}$ als Linearkombination der Eigenbasis $\mathbf{f} = \sum_{n=1}^{N} a_n \mathbf{u}_n$, $a_n \in \mathbb{R}$ schreiben und erhalten

$$\mathcal{L}^k \mathbf{f} = \sum_{n=1}^{N} a_n \mathcal{L}^k \mathbf{u}_n = \sum_{n=1}^{N} a_n \lambda_n^k \mathbf{u}_n.$$

Somit können Eigenschaften von $\mathcal{L}$ und damit des Graphen selber durch dessen Eigenwerte und Eigenvektoren beschrieben werden.

Eigenschaften. $\mathcal{L} \in \mathbb{R}^{N \times N}$ ist eine reell symmetrische, positiv semidefinite Matrix [5]. Folglich besitzt $\mathcal{L}$ nach Kapitel 5.1.1 genau N positiv reelle Eigenwerte $\{\lambda_n\}_{n=1}^{N}$ mit Ordnung $0 \leq \lambda_1 \leq \cdots \leq \lambda_N$ und N korrespondierenden orthogonalen Eigenvektoren $\{\mathbf{u}_n\}_{n=1}^{N}$.
Die kombinatorische Laplace-Matrix $\mathbf{L}$ ist nach (5.2) weiterhin schwach diagonaldominant. Insbesondere summiert sich jede Zeilen- und Spaltensumme von $\mathbf{L}$ zu Null auf, d.h. $\sum_{j=1}^{N} \mathbf{L}_{ij} = \sum_{j=1}^{N} \mathbf{L}_{ji} = 0$. Daraus folgt unmittelbar, dass $\lambda_1 = 0$, da $\mathbf{u}_1 = 1/\sqrt{N}[1, \ldots, 1]^\top \in \mathbb{R}^N$ Eigenvektor von $\mathbf{L}$ mit $\mathbf{L}\mathbf{u}_1 = \mathbf{0}$ [44]. $\tilde{\mathbf{L}}$ hingegen ist nicht zwingend schwach diagonaldominant. Es lässt sich jedoch zeigen, dass $\lambda_1 = 0$ auch für $\tilde{\mathbf{L}}$ gilt [5].
Eine der interessantesten Eigenschaften eines Graphen ist dessen Konnektivität. Die Laplace-Matrix $\mathcal{L}$ bzw. deren Eigenwerte stellen ein geeignetes Mittel zur Untersuchung dieser Eigenschaft dar. So gilt beispielsweise für einen zusammenhängenden Graphen $\mathcal{G}$, dass $\lambda_2 > 0$. Falls $\lambda_i = 0$ und $\lambda_{i+1} \neq 0$, dann besitzt $\mathcal{G}$ genau i zusammenhängende Komponenten [5]. Damit ist die Anzahl der Null-Eigenwerte äquivalent zu der Anzahl an Komponenten, die ein Graph besitzt. Für $\tilde{\mathbf{L}}$ lässt sich weiterhin zeigen, dass $\lambda_{\max} \leq 2$ eine obere Schranke ihrer Eigenwerte ist [5].
Aus der Laplace-Matrix können ebenso Rückschlüsse über die kürzeste Pfaddistanz zweier Knoten gewonnen werden. So gilt für $\mathcal{L}^k$ mit $k \in \mathbb{N}$, dass $\mathcal{L}_{ij}^k = 0$ genau dann, wenn $s(v_i, v_j) > k$ [17]. Damit beschreibt $\mathcal{L}_i^k$ bildlich gesprochen die Menge an Knoten, die maximal k Kanten von v_i entfernt liegen.

5.2 Spektraler Faltungsoperator

Sei $\mathbf{f} \in \mathbb{R}^N$ ein Signal auf den Knoten eines Graphen $\mathcal{G}$, welches abhängig von der Struktur des Graphen weiter verarbeitet werden soll. Es ist jedoch nicht selbstver-

ständlich, wie recht einfache, dennoch fundamentale Signalverarbeitungsprozesse wie Translation oder Filterung und die daraus entstehende Faltung in der Domäne des Graphen definiert werden können [44]. So kann ein analoges Signal $f(t)$ beispielsweise mittels $f(t-3)$ um 3 nach rechts verschoben werden. Es ist hingegen völlig unklar, was es bedeutet, ein Graphsignal auf den Knoten um 3 nach rechts zu bewegen (vgl. [44]). Die spektrale Graphentheorie bietet uns dafür einen geeigneten Weg, indem Eingabesignale in das Spektrum des Graphen zerlegt bzw. abgebildet, modifiziert und wieder retransformiert werden können.

5.2.1 Graph-Fourier-Transformation

Das Spektrum eines Graphen $\mathcal{G}$ bilden die Eigenwerte $\{\lambda_n\}_{n=1}^N$ der Laplace-Matrix $\mathcal{L}$ von $\mathcal{G}$. Diese werden deshalb auch oft als die *Frequenzen* von $\mathcal{G}$ betitelt. In der spektralen Domäne können wir ein Eingabesignal $\mathbf{f}$ über $\mathcal{G}$ dann analog wie ein zeitdiskretes Abtastsignal in der Fourier-Domäne behandeln.

Klassische Fourier-Transformation. Die Fourier-Transformation $\hat{f}$ einer Funktion $f(t)$ ist definiert als [44]

$$\hat{f}(\omega) \coloneqq \left\langle f, e^{2\pi i\omega t} \right\rangle = \int_{\mathbb{R}} f(t) e^{-2\pi i\omega t} \, \mathrm{d}t.$$

Die komplexen Exponentiale $e^{2\pi i\omega t}$ beschreiben dabei die Eigenfunktionen des eindimensionalen Laplace-Beltrami Operators [44]:

$$-\nabla^2 e^{2\pi i\omega t} = -\frac{\partial^2}{\partial t^2} e^{2\pi i\omega t} = (2\pi\omega)^2 e^{2\pi i\omega t} \tag{5.4}$$

$\hat{f}$ kann damit als die Ausdehnung von f in Bezug auf die Eigenfunktionen des Laplace-Beltrami Operators ∇^2 verstanden werden [17].

Analog lässt sich die *Graph-Fourier-Transformation* einer Funktion $f\colon \mathcal{V} \to \mathbb{R}$ bzw. $\mathbf{f} \in \mathbb{R}^N$ auf den Knoten eines Graphen $\mathcal{G}$ als Ausdehnung von f in Bezug auf die Eigenvektoren $\{\mathbf{u}_n\}_{n=1}^N$ der Laplace-Matrix $\mathcal{L}$ definieren [44]:

$$\hat{f}(\lambda_i) \coloneqq \langle \mathbf{f}, \mathbf{u}_i \rangle \qquad \text{bzw.} \qquad \hat{\mathbf{f}} \coloneqq \mathbf{U}^\top \mathbf{f} \tag{5.5}$$

Die inverse Graph-Fourier-Transformation ergibt sich dann als [44]

$$f(v_i) = \sum_{n=1}^{N} \hat{f}(\lambda_n)(\mathbf{u}_n)_i \qquad \text{bzw.} \qquad \mathbf{f} = \mathbf{U}\hat{\mathbf{f}}. \tag{5.6}$$

In der klassischen Fourier-Analyse sind für die Eigenwerte $\left\{(2\pi\omega)^2\right\}_{\omega\in\mathbb{R}}$ aus (5.4) nahe bei Null die korrespondieren Eigenfunktionen kleine, weich schwingende Funktionen, wohingegen für größere Eigenwerte bzw. Frequenzen die Eigenfunktionen sehr schnell und zügig anfangen zu oszillieren. Bei der Graph-Fourier-Transformation ist dies ähnlich. So ist für $\mathbf{L}$ der erste Eigenvektor $\mathbf{u}_1 = 1/\sqrt{N}[1,\dots,1]^\top$ zum Eigenwert $\lambda_1 = 0$ konstant und an jedem Knoten gleich. Generell zeigt sich, dass die Eigenvektoren geringer Frequenzen nur geringfügig im Graph variieren, wohingegen Eigenvektoren größerer Eigenwerte immer unähnlicher werden (vgl. [44]).

Die Graph-Fourier-Transformation (5.5) und ihre Inverse (5.6) bieten uns eine Möglichkeit, ein Signal in zwei unterschiedlichen Domänen zu repräsentieren, nämlich der Knotendomäne, d.h. das unveränderte Signal auf der Knotenmenge $f(v_i)$, und der spektralen Domäne, d.h. das transformierte Signal in das Spektrum des Graphen $\hat{f}(\lambda_i)$. Diese Transformation erlaubt uns die Formulierung fundamentaler Signalverarbeitungsoperationen.

5.2.2 Spektrale Filterung

In der Signalverarbeitung versteht man unter der Frequenzfilterung die Transformation eines Eingabesignals in die Fourier-Domäne und der verstärkenden oder dämpfenden Veränderung der Amplituden der Frequenzkomponenten. Formal betrachtet ergibt dies

$$\hat{f}_{\text{out}}(\omega) \coloneqq \hat{f}_{\text{in}}(\omega)\hat{g}(\omega) \tag{5.7}$$

mit dem Filter $\hat{g}\colon \mathbb{R} \to \mathbb{R}$. Shuman u. a. zeigen, dass die Filterung in der Fourier-Domäne äquivalent zu einer Faltung in der Zeitdomäne ist, d.h.

$$(f_{\text{in}} \star g)(t) \coloneqq \int_{\mathbb{R}} f_{\text{in}}(\tau)g(t-\tau)\,\mathrm{d}\tau = f_{\text{out}}(t). \tag{5.8}$$

Wir können die Filterung der Frequenzen in der Fourier-Domäne analog zu (5.7) für die spektrale Domäne auf Graphen über

$$\hat{f}_{\text{out}}(\lambda_i) \coloneqq \hat{f}_{\text{in}}(\lambda_i)\hat{g}(\lambda_i) \qquad \text{bzw.} \qquad \hat{\mathbf{f}}_{\text{out}} \coloneqq \hat{\mathbf{f}}_{\text{in}} \odot \hat{\mathbf{g}}$$

beschreiben, wobei $\odot$ das elementweise Hadamard-Produkt ist [44]. $\hat{\mathbf{g}} \in \mathbb{R}^N$ ist damit ein *nicht-parametrischer* Filter, d.h. ein Filter, dessen Werte für alle Frequenzen $\{\lambda_n\}_{n=1}^N$ frei wählbar sind [8]. Daraus ergibt sich analog zu (5.8) der *spektrale Faltungsoperator* auf Graphen in der Knotendomäne mit Hilfe der Graph-Fourier-Transformation (5.5) und ihrer Inversen (5.6) als [8, 44]

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \coloneqq \mathbf{U}\Big(\Big(\mathbf{U}^\top \mathbf{f}_{\text{in}}\Big) \odot \hat{\mathbf{g}}\Big) = \mathbf{f}_{\text{out}}. \tag{5.9}$$

5.2.3 Polynomielle Approximation

Es zeigt sich, dass die Benutzung des spektralen Faltungsoperators in (5.9) im Kontext eines CNNs auf Graphen mehrere Schwächen aufweist. So ist zum Beispiel die Auswertung von $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ extrem berechnungsintensiv, denn die Multiplikation mit der dichtbesetzten Eigenvektormatrix $\mathbf{U}$ liegt in $\mathcal{O}(N^2)$ [8]. Zudem muss $\mathbf{U}$ zuerst bestimmt werden, ein kostspieliger Aufwand für Graphen mit möglicherweise weit mehr als hundert Knoten [26]. Desweiteren führt ein Filter $\hat{\mathbf{g}} \in \mathbb{R}^N$ der Größe N zu einem Lernaufwand in $\mathcal{O}(N)$, d.h. der Dimensionalität der Eingabedaten [8]. Ebenso kann $\hat{\mathbf{g}}$ so nicht für das Lernen auf Graphen mit variierendem N verwendet werden. Um die oben genannten Schwächen zu umgehen kann $\hat{g}(\lambda_i)$ über ein Polynom

$$\hat{g}(\lambda_i) \approx \sum_{k=0}^{K} c_k \lambda_i^k \tag{5.10}$$

vom Grad K mit Koeffizienten $\mathbf{c} \coloneqq [c_0, \ldots, c_K]^\top \in \mathbb{R}^{K+1}$ approximiert werden [8, 17]. Die Filtergröße sinkt somit auf einen konstanten Faktor K mit Lernaufwand $\mathcal{O}(K)$, dem gleichen Aufwand klassischer zweidimensionaler CNNs [8]. $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ ergibt dann nach (5.1), (5.9) und (5.10) approximiert durch [8]

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx \sum_{k=0}^{K} c_k \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^\top \mathbf{f}_{\text{in}} = \sum_{k=0}^{K} c_k \mathcal{L}^k \mathbf{f}_{\text{in}}. \tag{5.11}$$

Insbesondere ist die spektrale Faltung damit nicht mehr abhängig von der Berechnung der Eigenwerte bzw. Eigenvektoren von $\mathcal{L}$. Mittels Kapitel 5.1.2 kann $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ in der Knotendomäne nun als eine *lokaliserte lineare Transformation* interpretiert werden. So sammelt ein Summand $\mathcal{L}^k \mathbf{f}_{\text{in}}$ des spektralen Filters an einem Knoten v genau die Signale von Knoten auf, die maximal k Kanten von v entfernt liegen [17].

Eine Faltung über $f_{\text{in}}(v_i)$ wird damit als Linearkombination

$$f_{\text{out}}(v_i) \approx b_{ii} f_{\text{in}}(v_i) + \sum_{v_j \in \mathcal{N}_K(v_i)} b_{ij} f_{\text{in}}(v_j)$$

über dessen k-lokalisierte Nachbarschaft $\mathcal{N}_K(v_i)$ mit $b_{ij} \coloneqq \sum_{k=0}^{K} c_k \mathcal{L}_{ij}^k$ beschrieben [44].

Tschebyschow-Polynome. Obwohl der Aufwand zur Bestimmung von $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ durch die polynomielle Approximation und insbesondere durch den Wegfall der Berechnung der Eigenvektormatrix $\mathbf{U}$ deutlich reduziert wurde, ist diese immer noch recht teuer aufgrund der Berechnung der K-ten Potenz von $\mathcal{L}$. So ist $\mathcal{L}$ zwar eine dünnbesetzte Matrix mit $|\mathcal{E}| + N \ll N^2$, $N \leq |\mathcal{E}|$, Einträgen, $\mathcal{L}^K$ ist dies jedoch zwangsläufig nicht. Eine Lösung zu diesem Problem ist die Benutzung eines Polynoms mit einer rekursiven Formulierung. Ein rekursives Polynom, dass dafür üblicherweise genutzt wird, ist das *Tschebyschow-Polynom*, da sich dieses zusätzlich durch einen sehr günstigen Fehlerverlauf auszeichnet (vgl. [17]). Tschebyschow-Polynome bezeichnen eine Menge von Polynomen $T_k(x) \colon \mathbb{R} \to \mathbb{R}$ mit dem rekursiven Zusammenhang

$$T_k(x) = 2x\, T_{k-1}(x) - T_{k-2}(x)$$

mit $T_0(x) = 1$ und $T_1(x) = x$ [17]. Ein Tschebyschow-Polynom T_k ist ein Polynom k-ten Grades und liegt im Intervall $[-1, 1]$ für $x \in [-1, 1]$ [17]. Wir können diese Tatsache nutzen und anstatt T_k auf $\mathbf{\Lambda} \in [0, \lambda_{\max}]^{N \times N}$ auf die skalierten und verschobenen Eigenwerte $\tilde{\mathbf{\Lambda}} \coloneqq 2\mathbf{\Lambda}/\lambda_{\max} - \mathbf{I} \in [-1, 1]^{N \times N}$ anwenden [8]. Die spektrale Faltung mittels Tschebyschow-Polynomen ergibt sich dann nach (5.11) als

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx \sum_{k=0}^{K} c_k \mathbf{U} T_k\Big(\tilde{\mathbf{\Lambda}}\Big) \mathbf{U}^\top \mathbf{f}_{\text{in}}, \tag{5.12}$$

wobei die Werte $\mathbf{c} \coloneqq [c_0, \ldots, c_K]^\top \in \mathbb{R}^{K+1}$ nun die Koeffizienten der Tschebyschow-Polynome $T_k\Big(\tilde{\mathbf{\Lambda}}\Big) \in \mathbb{R}^{N \times N}$ bilden [8]. $\mathbf{U} T_k\Big(\tilde{\mathbf{\Lambda}}\Big) \mathbf{U}^\top$ kann aufgrund der polynomiellen Form von T_k und (5.1) ebenso auf $T_k\Big(\tilde{\mathcal{L}}\Big)$ mit $\tilde{\mathcal{L}} \coloneqq \mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{U}^\top = \frac{2}{\lambda_{\max}}\mathcal{L} - \mathbf{I}$ angewendet werden [8]. Damit kann (5.12) weiter vereinfacht werden zu

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx \sum_{k=0}^{K} c_k T_k\Big(\tilde{\mathcal{L}}\Big) \mathbf{f}_{\text{in}} \tag{5.13}$$

und ist nun ein rekursiv formulierter Faltungsoperator, der weiterhin ohne die explizite Berechnung von $\mathbf{U}$ auskommt [8]. Für $\mathcal{L}$ kann $\lambda_{\max} \leq 2$ auf dessen obere Schranke, d.h. $\lambda_{\max} \coloneqq 2$, gesetzt werden, sodass die Berechnung von $\lambda_{\max}$ vermieden werden kann ohne die Schranken von $\tilde{\mathbf{\Lambda}} \in [-1, 1]^{N \times N}$ zu verletzen.
Der rekursive Zusammenhang von T_k hilft dabei, $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ effizient zu bestimmen. Dafür muss $\bar{f}(k) \coloneqq T_k\big(\tilde{\mathcal{L}}\big)\mathbf{f}_{\text{in}} \in \mathbb{R}^N$ für alle $k \in \{0, \ldots, K\}$ rekursiv mit

$$\bar{f}(0) = \mathbf{f}_{\text{in}} \qquad \bar{f}(1) = \tilde{\mathcal{L}}\,\mathbf{f}_{\text{in}} \qquad \bar{f}(k) = 2\tilde{\mathcal{L}}\,\bar{f}(k-1) - \bar{f}(k-2)$$

berechnet werden. Dann ergibt sich $\mathbf{f}_{\text{out}} = \Big[\bar{f}(0), \bar{f}(1), \ldots, \bar{f}(K)\Big]\mathbf{c} \in \mathbb{R}^N$ (vgl. [17]). $\mathbf{f}_{\text{out}}$ lässt sich damit über $K + 1$ Multiplikationen einer dünnbesetzen Matrix mit einem Vektor und einer abschließenden Vektormultplikation beschreiben. Aufgrund $N \leq |\mathcal{E}|$ ergibt dies eine finale Laufzeit der spektralen Faltung von $\mathcal{O}(K|\mathcal{E}|)$ [8].

Implementierung. Für gewöhnlich besteht eine CNN-Schicht nicht nur aus einem, sondern aus M_{in} vielen Signalen bzw. Merkmalen pro Knoten mit jeweils unterschiedlichen Filtern bzw. Gewichten pro Ein- und Ausgabekarte. In klassichen zweidimensionalen CNNs werden diese Merkmale auf M_{out} viele Merkmale abgebildet, indem für jede Ausgabekarte über jede Eingabekarte gefaltet und ihre Ergebnisse sukzessive aufsummiert werden (vgl. Kapitel 2.3). Die spektrale Faltung auf einer Merkmalsmatrix $\mathbf{F}_{\text{in}} \in \mathbb{R}^{N \times M_{\text{in}}}$ in eine Merkmalsmatrix $\mathbf{F}_{\text{out}} \in \mathbb{R}^{N \times M_{\text{out}}}$ ergibt sich dann mittels eines Filtertensors $\mathbf{W} \in \mathbb{R}^{(K+1) \times M_{\text{in}} \times M_{\text{out}}}$ völlig analog zu

$$\mathbf{F}_{\text{out}} \coloneqq \sum_{k=0}^{K} \bar{\mathbf{F}}_k,$$

wobei $\bar{\mathbf{F}}_k = \Big(2\tilde{\mathcal{L}}\,\bar{\mathbf{F}}_{k-1} - \bar{\mathbf{F}}_{k-2}\Big)\mathbf{W}_k \in \mathbb{R}^{N \times M_{\text{out}}}$ mit $\bar{\mathbf{F}}_0 = \mathbf{F}_{\text{in}}\mathbf{W}_0$ und $\bar{\mathbf{F}}_1 = \tilde{\mathcal{L}}\,\mathbf{F}_{\text{in}}\mathbf{W}_1$.

5.3 Graph Convolutional Networks

Kipf und Welling präsentieren einen weiteren Ansatz zur Faltung auf Graphen, genannt *Graph Convolutional Network (GCN)*, der auf der Methodik des spekralen Faltungsoperators aus Kapitel 5.2 aufbaut und dabei wie eine „differenzierbare und parametrisierte Generalisierung des eindimensionalen Weisfeiler-Lehman-Algorithmus auf Graphen“ fungiert [26].

Faltungsoperator. Sei $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx \sum_{k=0}^{K} c_k T_k\big(\tilde{\mathcal{L}}\big)\mathbf{f}_{\text{in}}$ der in (5.13) definierte spektrale Faltungsoperator mit $K = 1$. Dann ist $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ eine lineare Funktion bezüglich $\mathcal{L}$ und damit eine lineare Funktion auf dem Spektrum des Graphen [26]. Mit $K = 1$ betrachtet der spektrale Faltungsoperator nur noch die lokale Nachbarschaft eines jeden Knotens (vgl. 5.2.3). Es ist anzumerken, dass dies in der Regel keinen Nachteil darstellt. So hat es sich bei gegenwärtigen „State-of-the-Art"-CNNs auf Bildern ebenfalls bewährt, nur noch über minimale 3×3 Filtergrößen zu falten und stattdessen Merkmale weit entfernterer Knoten über die mehrfache Aneinanderreihung der Faltungsschichten mittels tieferer Netze zu gewinnen (vgl. [19, 26, 46]). Unter dieser Restriktion vereinfacht sich $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ zu

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx c_0\, \mathbf{f}_{\text{in}} + c_1\Big(\frac{2}{\lambda_{\max}}\mathcal{L} - \mathbf{I}\Big)\mathbf{f}_{\text{in}} \tag{5.14}$$

mit zwei freien Parametern c_0 und c_1 [26]. Für $\tilde{\mathbf{L}}$ auf einem zusammenhängenden Graphen $\mathcal{G}$ gilt dann nach (5.3) und (5.14) weiter

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx c_0\, \mathbf{f}_{\text{in}} + c_1\big(\tilde{\mathbf{L}} - \mathbf{I}\big)\mathbf{f}_{\text{in}} = c_0\, \mathbf{f}_{\text{in}} - c_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{f}_{\text{in}}, \tag{5.15}$$

wobei $\lambda_{\max} \coloneqq 2$ auf dessen obere Schranke gesetzt wird [26]. Um die Gefahr des Overfittings und die Anzahl an Berechnungen pro Schicht weiter zu beschränken, reduziert sich (5.15) mit einem einzigen Parameter $c \coloneqq c_0$ mit $c = -c_1$ zu [26]

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx c\big(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\big)\mathbf{f}_{\text{in}}.$$

Die skalierten Eigenwerte von $\tilde{\mathbf{\Lambda}}$ liegen aufgrund der Addition mit $\mathbf{I}$ nun im Intervall $[0, 2]$ (vgl. [26]). Demnach können wiederholte Anwendungen des Faltungsoperators zu „numerischen Instabilitäten und folglich zu explodierenden oder verschwindenen Gradienten" führen [26]. Kipf und Welling führen zur Behebung dieses Problems die folgende Renormalisierung durch: $\mathbf{I}+\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} \to \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ mit $\tilde{\mathbf{A}} \coloneqq \mathbf{A}+\mathbf{I}$ und $\tilde{\mathbf{D}}_{ii} \coloneqq \sum_{j=1}^{N} \tilde{\mathbf{A}}_{ij}$ bzw. $\tilde{\mathbf{D}}_{ii} = \mathbf{D} + \mathbf{I}$. Der entgültige Faltungsoperator des GCNs ergibt sich dann als

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx c\,\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{f}_{\text{in}} \tag{5.16}$$

auf einem einzigen freien Parameter $c \in \mathbb{R}$.

Implementierung. Die Faltung des GCNs auf Merkmalsmatrizen lässt sich analog zur Tensorimplementierung des spektralen Faltungsoperators in Kapitel 5.2.3 beschreiben, mit dem Unterschied, dass wir aufgrund der Festlegung von $K = 1$

Eingabe: Initiale Knotenfärbung $\mathbf{f}^{(0)} \in \mathbb{R}^N$
Ausgabe: Finale Knotenfärbung $\mathbf{f}^{(T)} \in \mathbb{R}^N$ nach T Durchläufen
$t \leftarrow 0$
repeat
  for $v_i \in \mathcal{V}$ **do**
    $\mathbf{f}_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{v_j \in \mathcal{N}(v_i)} \mathbf{f}_j^{(t)}\right)$
  end for
  $t \leftarrow t + 1$
until Konvergenz
return $\mathbf{f}^{(T)}$

Algorithmus 5.1: Eindimensionaler Weisfeiler-Lehman-Algorithmus auf einer initialen Knotenfärbung bzw. Merkmalsfunktion $\mathbf{f}^{(0)} \in \mathbb{R}^N$ eines Graphen $\mathcal{G}$ mit $v_i \in \mathcal{N}(v_i)$ [54]. Der Prozess der Verfärbung eines jeden Knotens v_i auf Basis der Farben seiner lokalen Nachbarknoten wird solange wiederholt, bis diese konvergieren.

keinen Filtertensor, sondern lediglich eine Filtermatrix $\mathbf{W} \in \mathbb{R}^{M_{\text{in}} \times M_{\text{out}}}$ nutzen. Die Faltung einer Eingabemerkmalsmatrix $\mathbf{F}_{\text{in}} \in \mathbb{R}^{N \times M_{\text{in}}}$ auf eine Ausgabemerkmalsmatrix $\mathbf{F}_{\text{out}} \in \mathbb{R}^{N \times M_{\text{out}}}$ ergibt sich dann als

$$\mathbf{F}_{\text{out}} \coloneqq \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{F}_{\text{in}} \mathbf{W} \tag{5.17}$$

mit Faltungsaufwand $\mathcal{O}(M_{\text{in}} M_{\text{out}} |\mathcal{E}|)$, weil $\tilde{\mathbf{A}}\mathbf{F}_{\text{in}}$ effizient mit der Multiplikation einer dünnbesetzten mit einer dichtbesetzten Matrix implementiert werden kann [26].

Beziehung zum Weisfeiler-Lehman-Algorithmus. Der *eindimensionale Weisfeiler-Lehman Algorithmus* beschreibt eine weit untersuchte Methode zur Knotenklassifizierung eines Graphen basierend auf einer initialen Färbung bzw. Merkmalsfunktion auf den Knoten eines Graphen $\mathcal{G}$, die unter anderem zur Bestimmung von Graphisomorphismen genutzt wird [10]. Basierend auf einer initialen kontinuierlichen Knotenfärbung $\mathbf{f}^{(0)} \in \mathbb{R}^N$ wird die Farbe eines jeden Knotens $v_i \in \mathcal{V}$ sukzessive mit Hilfe einer Hashfunktion hash$(\cdot)$ so angepasst, dass sie die vorangegangene Farbe des Knotens zusammen mit den Farben seiner lokalen Nachbarschaft repräsentiert. Dieser Prozess wiederholt sich solange, bis eine stabile Knotenfärbung gefunden wurde, d.h. die gefundene Färbung des Graphen konvergiert (vgl. Algorithmus 5.1).

Sei die Hashfunktion nun gegeben als eine differenzierbare, nicht-lineare Aktivierungsfunktion $\sigma \colon \mathbb{R} \to \mathbb{R}$, beispielsweise $\text{ReLU}(\cdot) \coloneqq \max(\cdot, 0)$, eines neuronalen Netzes. Dann ergibt sich eine Faltungsschicht des GCNs durch die Verkettung des

Faltungsoperators mit anschließender Aktivierung als

$$\mathbf{f}_i^{(t+1)} = \sigma\left(\sum_{v_j \in \mathcal{N}(v_i)} \frac{w(v_i, v_j)}{\sqrt{d_i d_j}} \mathbf{f}_j^{(t)} \mathbf{W}^{(t)}\right),$$

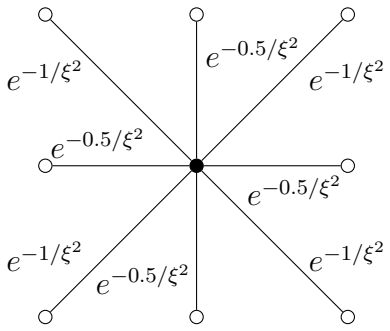
wobei $w(v_i, v_j)/\sqrt{d_i d_j} \in \mathbb{R}$ die Normalisierungskonstante für die Kante $(v_i, v_j) \in \mathcal{E}$ entsprechend der Normalisierung $\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ aus (5.16) und $\mathbf{W}^{(t)} \in \mathbb{R}^{N \times N}$ die Filtermatrix der t-en Faltungsschicht beschreibt [26]. Folglich kann die Faltung des GCNs als „differenzierbare und parametrisierte Generalisierung des eindimensionalen Weisfeiler-Lehman-Algorithmus auf Graphen“ verstanden werden [26].

5.4 Erweiterung auf Graphen im zweidimensionalen Raum

Die Ansätze von Defferrard u. a. aus Kapitel 5.2 und von Kipf und Welling aus Kapitel 5.3 zeigen konkurrenzfähige Resultate auf einer Reihe von Datensätzen auf Graphen (vgl. [8, 26]). So erreicht das GCN zum Beispiel in der teilweise-überwachten Knotenklassifizierung von *Referenzgraphen*, d.h. einer Menge von Knoten, die Dokumente über eine Reihe von Bag-of-Words-Merkmalen repräsentieren und (ungerichtet) über dessen Referenzierungen miteinander verbunden sind, beachtliche Ergebnisse und schneidet in diesen sogar knapp besser ab als über die Tschebyschow-Approximation mit $K = 2$ und $K = 3$ [26].
Die spektrale Faltung auf Graphen entspricht einer Generalisierung der Faltung klassischer CNNs auf zweidimensionalen Bildern [23]. Es ist jedoch anzumerken, dass die spektrale Faltung im Gegensatz zur klassischen Faltung auf einem regulären Gitter insbesondere rotationsinvariant ist. Das ist in der Regel für generelle Graphen keine Schwäche, schließlich kann den Knoten bzw. Kanten eines Graphen, kodiert als Adjazenzmatrix, keine Örtlichkeit bzw. Richtung (wie links, rechts, oben oder unten) zugeordnet werden. Die Rotationsinvarianz kann folglich sowohl als Einschränkung als auch als Vorteil interpretiert werden, abhängig von dem Problem, welches man betrachtet [8].
Im Kontext dieser Arbeit, dem Lernen auf Graphen im zweidimensionalen euklidischen Raum, bei denen Graphknoten eine eindeutige Position besitzen, ist die Rotationsinvarianz weitestgehend unerwünscht. Das kann leicht verifiziert werden, indem wir den Filter des GCNs auf einer Graphrepräsentation eines Gitters mit Abstand $\|1\|_2$ visualisieren (vgl. Abbildung 5.2). Diese Repräsentation entspricht damit

$(-1,-1)$	$(0,-1)$	$(1,-1)$
$(-1,0)$	$(0,0)$	$(1,0)$
$(-1,1)$	$(0,1)$	$(1,1)$

(a) Reguläres Gitter



(b) Graphrepräsentation

Abbildung 5.2: Illustration (a) eines 3×3 großen regulären Gitters zentriert um den Punkt $(0,0)$ und (b) dessen lokale Nachbarschaft der entsprechenden Graphrepräsentation mit einer Konnektivität von 8 bei horizontalen bzw. vertikalen Kantengewichten mit $\exp(-0.5/\xi^2) \in \mathbb{R}$ bzw. mit $\exp(-1/\xi^2) \in \mathbb{R}$ bei den Diagonalen.

genau dem Problem der zweidimensionalen Faltung auf Bildern mit einer Filtergröße von 3×3. Hier zeigt sich jedoch besonders deutlich die Limitierung des Netzes durch die Rotationsinvarianz. Wohingegen wir bei klassischen CNNs 3×3 unterschiedliche Parameter mit eindeutiger Örtlichkeit auf den benachbarten Bildpixeln trainieren, reduziert sich der Filter des GCNs (vereinfacht ohne Normalisierung mit $\tilde{\mathbf{D}}^{-1/2}$) effektiv zu einer Filtermatrix der Form

$$\begin{bmatrix} ce^{-1/\xi^2} & ce^{-0.5/\xi^2} & ce^{-1/\xi^2} \\ ce^{-0.5/\xi^2} & c & ce^{-0.5/\xi^2} \\ ce^{-1/\xi^2} & ce^{-0.5/\xi^2} & ce^{-1/\xi^2} \end{bmatrix} = c \begin{bmatrix} e^{-1/\xi^2} & e^{-0.5/\xi^2} & e^{-1/\xi^2} \\ e^{-0.5/\xi^2} & 1 & e^{-0.5/\xi^2} \\ e^{-1/\xi^2} & e^{-0.5/\xi^2} & e^{-1/\xi^2} \end{bmatrix}$$

mit einem einzigen trainierbaren Parameter $c \in \mathbb{R}$ bei horizontalen bzw. vertikalen Kantengewichten nach (3.1) mit $\exp(-0.5/\xi^2) \in \mathbb{R}$ bzw. mit $\exp(-1/\xi^2) \in \mathbb{R}$ bei den Diagonalen. Damit reduziert sich das Training einer Faltungsschicht eines solchen GCNs letztendlich auf eine Skalarmultiplikation. Es scheint schwer vorstellbar, mit diesem Ansatz komplexe Probleme wie zum Beispiel das Segmentieren eines Bildes zu lösen (vgl. [23]). Ein Vergleich zwischen der spektralen Faltung auf regulären Gittergraphen und der klassischen zweidimensionalen Faltung auf Bildern wird der spektralen Faltung aber nicht gerecht, schließlich wurden die klassischen CNNs speziell für die Anwendung auf Gittern entwickelt. So ist es zu erwarten, dass durch

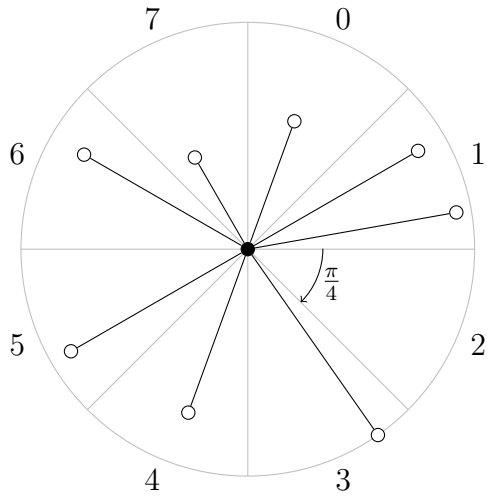


Abbildung 5.3: Partitionierung eines Graphknotens im Uhrzeigersinn in $P = 8$ Bereiche mit gleichmäßigen Innenwinkeln der Größe $\pi/4$.

die Formulierung einer Faltung für generelle Graphen gewisse Einschränkungen in Kauf genommen werden müssen. Im Folgenden lässt sich der Faltungsoperator der GCNs aber insofern modifizieren, dass sich dieser für beliebige Graphen in einem zweidimensionalen euklidischen Raum äquivalent zu der klassischen Formulierung auf regulären Gittern verhält.

5.4.1 Partitionierung

Sei $\mathcal{G}$ ein Graph im zweidimensionalen euklidischen Raum, definiert über dessen Adjazenzmatrizen $\mathbf{A}_{\text{dist}} \in [0,1)^{N\times N}$ und $\mathbf{A}_{\text{rad}} \in [0,2\pi]^{N\times N}$ (vgl. Kapitel 3). Dann lässt sich $\mathcal{G}$ in $P \in \mathbb{N}$ Bereiche $\{\mathbf{A}_p\}_{p=0}^{P-1}$ *partitionieren*, sodass

$$(\mathbf{A}_p)_{ij} \coloneqq \begin{cases} (\mathbf{A}_{\text{dist}})_{ij}, & \text{wenn } (\mathbf{A}_{\text{rad}})_{ij} \in (2\pi p/P, 2\pi(p+1)/P], \\ 0, & \text{sonst.} \end{cases}$$

Damit beschreiben die Matrizen $\{\mathbf{A}_p\}_{p=0}^{P-1}$ disjunkte Partitionen der Kanten des Graphen $\mathcal{G}$ abhängig von ihren Ausrichtungen im Raum mit $\mathbf{A}_{\text{dist}} = \sum_{p=0}^{P-1} \mathbf{A}_p$. $\mathbf{A}_p$ ist insbesondere nicht symmetrisch, da $(\mathbf{A}_{\text{rad}})_{ij} \neq (\mathbf{A}_{\text{rad}})_{ji}$ für alle $v_i, v_j \in \mathcal{V}$ mit $v_j \in \mathcal{N}(v_i)$. Abbildung 5.3 veranschaulicht den Prozess der Partitionierung. Mit $P = 8$ erhalten die jeweiligen Bereiche zum Beispiel einen gleichmäßigen Innenwinkel der Größe $\pi/4$.

Faltungsoperator. Es lässt sich analog zu Kapitel 5.3 ein Faltungsoperator definieren, bei dem nun jeder Partition ein eigener frei trainierbarer Parameter zugeordnet wird. Der Parameter einer Partition hat damit folglich eine eindeutige Örtlichkeitszuweisung über das Intervall bzw. den Bereich der Richtungen seiner Kanten. Analog zu (5.16) muss dafür zuerst die (Re-)normalisierung der Form

$$\tilde{\mathbf{D}}_{\text{dist}}^{-\frac{1}{2}}\tilde{\mathbf{A}}_{\text{dist}}\tilde{\mathbf{D}}_{\text{dist}}^{-\frac{1}{2}} = \tilde{\mathbf{D}}_{\text{dist}}^{-1} + \sum_{p=0}^{P-1}\tilde{\mathbf{D}}_{\text{dist}}^{-\frac{1}{2}}\mathbf{A}_p\tilde{\mathbf{D}}_{\text{dist}}^{-\frac{1}{2}}$$

mit $\tilde{\mathbf{A}}_{\text{dist}} \coloneqq \mathbf{A}_{\text{dist}} + \mathbf{I}$ und $\left(\tilde{\mathbf{D}}_{\text{dist}}\right)_{ii} \coloneqq \sum_{j=1}^{N}\left(\tilde{\mathbf{A}}_{\text{dist}}\right)_{ij}$ durchgeführt werden. Dies lässt sich mit Hilfe von $\mathbf{A}_{\text{dist}} = \sum_{p=0}^{P-1}\mathbf{A}_p$ und $\tilde{\mathbf{D}}_{\text{dist}}^{-1/2}\tilde{\mathbf{D}}_{\text{dist}}^{-1/2} = \tilde{\mathbf{D}}_{\text{dist}}^{-1}$ verifizieren. Im Folgenden sei $\tilde{\mathbf{A}}_p \coloneqq \tilde{\mathbf{D}}_{\text{dist}}^{-1/2}\mathbf{A}_p\tilde{\mathbf{D}}_{\text{dist}}^{-1/2}$ für $p \in \{0,\dots,P-1\}$ und $\tilde{\mathbf{A}}_P \coloneqq \tilde{\mathbf{D}}_{\text{dist}}^{-1}$. Dann folgt für den Faltungsoperatur $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ auf Graphen im zweidimensionalen euklidischen Raum, dass dieser über

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx \sum_{p=0}^{P} c_p\tilde{\mathbf{A}}_p\mathbf{f}_{\text{in}} \tag{5.18}$$

mit den freien Parametern $[c_0,\dots,c_P]^\top \in \mathbb{R}^{P+1}$ beschrieben werden kann.

Es stellt sich heraus, dass die Faltung in (5.18) auf den Partitionen eines regulären Gittergraphen mit $P = 8$ äquivalent zu der klassischen Faltung auf einem regulären Gitter mit Filtergröße 3×3 ist. Sei dafür $\mathcal{G}$ ein (unendlicher) regulärer Gittergraph bei einer Konnektivität von 8 und $\mathbf{f}_{\text{in}}$ Merkmalsvektor auf dem Graphen mit Koordinatenindexnotation $(\mathbf{f}_{\text{in}})_{x,y}$. Die klassische Faltung conv2d an einem Gitterpunkt (x, y) ist damit gegeben als

$$\text{conv2d}(\mathbf{f}_{\text{in}})_{x,y} = \sum_{i,j\in\{1,2,3\}}(\mathbf{f}_{\text{in}})_{x+i-2,y+j-2}\mathbf{W}_{i,j},$$

wobei $\mathbf{W} \in \mathbb{R}^{3\times3}$ eine Filtermatrix der Größe 3×3 ist. Da $\tilde{\mathbf{A}}_{\text{dist}}$ ein reguläres Gitter beschreibt sind die Einträge ihrer Matrixreihen äquivalent unter unterschiedlicher Permutation und folglich sind die Einträge auf der Diagonalen von $\tilde{\mathbf{D}}_{\text{dist}}$ identisch (o.B.d.A. entfällt hier die Randknotenbetrachtung). Aufgrund der Partitionierung von $\mathbf{A}_{\text{dist}}$ in 8 disjunkte Bereiche $\left\{\tilde{\mathbf{A}}_p\right\}_{p=0}^{7}$ enthält $\tilde{\mathbf{A}}_p$ genau einen Eintrag pro Matrixreihe korrespondierend zu einer Kante des regulären Gitters. Für $p \in \{1,3,5,7\}$ beschreibt $\tilde{\mathbf{A}}_p$ die horizontalen und vertikalen Kanten des Graphen mit jeweils gleichen Einträgen $\theta_0 \in \mathbb{R}$. Analog verweist $\tilde{\mathbf{A}}_p$ für $p \in \{0,2,4,6\}$ auf die diagonalen Kanten des Graphen mit den festen Einträgen $\theta_1 \in \mathbb{R}$. Sei weiterhin o.B.d.A.

$\theta_2 \coloneqq \left(\tilde{\mathbf{D}}_{\text{dist}}^{-1}\right)_{ii}$ für beliebiges $i \in \{0, \ldots, N\}$. Mit der Zuordnung

$$\mathbf{W} = \begin{bmatrix} c_6\theta_1 & c_7\theta_0 & c_0\theta_1 \\ c_5\theta_0 & c_8\theta_2 & c_1\theta_0 \\ c_4\theta_1 & c_3\theta_0 & c_2\theta_1 \end{bmatrix}$$

für den Faltungsoperator $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ aus (5.18) mit den Parametern $[c_0, \ldots, c_8]^\top \in \mathbb{R}^9$ folgt damit sofort die Äquivalenz zu conv2d($\mathbf{f}_{\text{in}}$) auf regulären Gittern.

Implementierung. Analog zu (5.17) lässt sich die Faltung für Merkmalsmatrizen $\mathbf{F}_{\text{in}} \in \mathbb{R}^{N \times M_{\text{in}}}$ und $\mathbf{F}_{\text{out}} \in \mathbb{R}^{N \times M_{\text{out}}}$ über einem Filtertensor $\mathbf{W} \in \mathbb{R}^{(P+1) \times M_{\text{in}} \times M_{\text{out}}}$ als

$$\mathbf{F}_{\text{out}} \coloneqq \sum_{p=0}^{P} \tilde{\mathbf{A}}_p \mathbf{F}_{\text{in}} \mathbf{W}_{p+1}$$

beschreiben. Es ist anzumerken, dass die Multiplikation mit den dünnbesetzten partitionierten Adjazenzmatrizen $\left\{\tilde{\mathbf{A}}_p\right\}_{p=0}^{P}$ extrem effizient ist, da $|\mathcal{E}_p| \ll |\mathcal{E}|$ und insbesondere $\sum_{p=0}^{P} |\mathcal{E}_p| = |\mathcal{E}| + N$ gilt, wobei $\mathcal{E}_p \in \mathcal{V} \times \mathcal{V}$ die Kantenmenge der Adjazenzmatrix $\tilde{\mathbf{A}}_p$ beschreibt. Die Laufzeit erhöht sich jedoch im Vergleich zu (5.17) durch die P-fache Multiplikation mit der Filtermatrix $\mathbf{W}_p$ zu $\mathcal{O}(PM_{\text{in}}M_{\text{out}}|\mathcal{E}|)$.

Obwohl die Faltung auf den Partitionen eines Graphen insbesondere durch die Äquivalenz zur klassischen Faltung auf regulären Gittern vielversprechend erscheint, gehen dabei dennoch Informationen über die Ausrichtung der Kanten im Raum verloren. Kanten mit verschiedenen Richtungen im Intervall $(2\pi p/P, 2\pi(p+1)/P]$ landen jeweils in der gleichen Partition p, auch wenn sich diese eventuell extrem unterscheiden. Eine Lösung zu diesem Problem ist sicherlich, P insoweit zu erhöhen, dass die Innenwinkel der einzelnen Partitionen entsprechend klein werden. Dies erscheint jedoch für beliebige, unbekannte Graphstrukturen nicht zwangsläufig sinnvoll. Neben dem erhöhten Aufwand der Faltung erhalten wir im schlimmsten Fall viele Partitionsmatrizen $\mathbf{A}_p$, die eine Nullmatrix $\mathbf{0}$ darstellen oder nur extrem wenige Kanten beinhalten. Kleinste Veränderungen in den Ausrichtungen der Kanten sorgen dann schließlich dafür, dass eine komplett andere Filtermatrix angesprochen wird. Ein dazu alternativ entwickelter Ansatz ist die Approximation des Filters über stückweise stetiger Polynome zwischen den Partitionsgrenzen des Graphen mit Hilfe von B-Spline-Kurven.

5.4.2 Polynomielle Approximation über B-Spline-Kurven

Eine B-Spline-Kurve beschreibt eine stückweise polynomielle Approximation einer Kurve vom Grad $K \in \mathbb{N}$ über $M+1$ Kontrollpunkten $\mathbf{d}_0, \ldots, \mathbf{d}_M \in \mathbb{R}^d$ [7]. Eine B-Spline Kurve kann dabei offen, d.h. mit beliebigen Endpunkten, sowie geschlossen, d.h. mit gleichem Anfangs- und Endpunkt, beschrieben werden [2]. Die *geschlossene B-Spline-Kurve* $b(t)$ ist auf dem Intervall $t \in (t_0, t_{M+1}]$ definiert als

$$b(t) \coloneqq \sum_{m=0}^{M} \mathbf{d}_m \mathrm{N}_m^K(t), \tag{5.19}$$

mit den *B-Spline-Funktionen* $\left\{\mathrm{N}_m^K\right\}_{m=0}^{M}$ zu einem *Knotenvektor* $\tau \in \mathbb{R}^{M+K+2}$ der Form $\tau \coloneqq [t_0, \ldots, t_{M+K+1}]^\top$ mit der Bedingung, dass $t_{M+k+2} \coloneqq t_{M+k+1} + (t_{k+1} - t_k)$ für $k \in \{0, \ldots, K-1\}$ [2]. Die B-Spline-Funktion $\mathrm{N}_m^K \colon (t_0, t_{m+1}] \to [0,1]$ ist rekursiv über $k \in \{0, \ldots, K\}$ definiert mit der Initialisierung $(k=0)$

$$\mathrm{N}_m^0(t) \coloneqq \begin{cases} 1, & \text{falls } t \in (t_m, t_{m+1}], \\ 0, & \text{sonst} \end{cases}$$

und dem Rekursionsschritt $(k-1 \to k)$

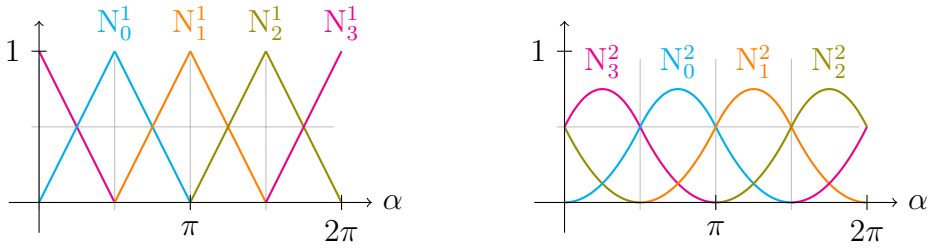
$$\mathrm{N}_m^k(t) \coloneqq \frac{t - t_m}{t_{m+k} - t_m} \mathrm{N}_m^{k-1}(t) + \frac{t_{m+k+1} - t}{t_{m+k+1} - t_{m+1}} \mathrm{N}_{m+1}^{k-1}(t)$$

für $t \in (t_m, t_{M+1}]$ sowie $\mathrm{N}_m^k(t) \coloneqq \mathrm{N}_m^k(t - t_0 + t_{M+1})$ für $t \in (t_0, t_m]$ [2]. Ein Kontrollpunkt $\mathbf{d}_m$ beeinflusst die Kurve damit lediglich im Intervall $t_m < t \leq t_{m+K+1}$. Die Größe von K wird deshalb auch oft *lokale Kontrollierbarkeit* genannt. Weiterhin gilt für die Aufsummierung der B-Spline-Funktionen $\left\{\mathrm{N}_m^K\right\}_{m=0}^{M}$, dass $\sum_{m=0}^{M} \mathrm{N}_m^K(t) = 1$ für beliebige $K \in \mathbb{N}$ und $t \in (t_0, t_{M+1}]$ [7].

Wir können die Definition der B-Spline-Kurve $b(t)$ aus (5.19) nutzen, um sie aufbauend auf Kapitel 5.4.1 in das Anwendungsgebiet eines stückweisen polynomiellen Filters $b(\alpha)$ auf den Richtungen bzw. Winkeln eines Graphen $\mathcal{G}$, beschrieben durch $\mathbf{A}_{\text{dist}}$ und $\mathbf{A}_{\text{rad}}$, zu übertragen. Sei dafür $b \colon [0, 2\pi] \to \mathbb{R}$ im Folgenden eine B-Spline-Kurve auf den Winkeln der Graphkanten mit

$$b(\alpha) \coloneqq \sum_{p=0}^{P-1} c_p N_p^K(\alpha),$$

wobei die freien Parameter $[c_0, \ldots, c_{P-1}]^\top \in \mathbb{R}^P$ aus (5.18) nun die Koeffizienten



(a) $K = 1$ (b) $K = 2$

Abbildung 5.4: Illustration geschlossener B-Spline-Funktionen $\mathrm{N}_p^K \colon (0, 2\pi] \to [0, 1]$ für $P = 4$ gleichmäßig verteilter Kontrollpunkte, einmal mit lokaler Kontrollierbarkeit $K = 1$ (a) und einmal mit $K = 2$ (b).

der B-Spline-Funktionen $\left\{N_p^K\right\}_{p=0}^{P-1}$ bilden. Insbesondere definieren wir $b(0) \coloneqq 0$, da der Winkel 0 in der Adjazenzmatrix $\mathbf{A}_{\text{rad}}$ weiterhin angeben soll, dass $(v_i, v_j) \notin \mathcal{E}$. Damit lässt sich $b(\alpha)$ weiterhin als eine P-fache Partitionierung von $\mathcal{G}$ auf Basis seiner Kantenausrichtungen vorstellen, mit dem Unterschied, dass $b(\alpha)$ nun eine lokale Kontrollierbarkeit innehält. Kanten, die vorher zwar in einer gleichen Partition lagen, sich jedoch eventuell stark in ihrer Ausrichtung unterschieden, sind nun „eindeutig“ differenzierbar über ihre abweichenden Auswertungen von $\left\{\mathrm{N}_p^K\right\}_{p=0}^{P-1}$ bzw. ihrer Anteile von $[c_0, \ldots, c_{P-1}]^\top$ an $b(\alpha)$ entsprechend ihrer unterschiedlichen Winkel. Abbildung 5.4 soll dabei das Konzept geschlossener B-Spline-Funktionen auf Winkeln veranschaulichen. Mit der expliziten Forderung gleichmäßig verteiler Knoten im Intervallbereich ergibt sich dann der Knotenvektor $\tau \coloneqq [\alpha_0, \ldots, \alpha_{P+K}]^\top \in \mathbb{R}_+^{P+K+1}$ mit $\alpha_p \coloneqq 2\pi p / P$ und erfüllt damit insbesondere die Bedingungen aus (5.19).

Faltungsoperator. Es kann folglich analog zu (5.18) der Faltungsoperator $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ auf Basis eines stückweisen polynomiellen Filters beschrieben werden. Mit $\tilde{\mathbf{A}}_{\text{dist}} \in \mathbb{R}^{N \times N}$ und $\tilde{\mathbf{D}}_{\text{dist}} \in \mathbb{R}^{N \times N}$ ergibt sich $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ dann als

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx c_P \tilde{\mathbf{D}}_{\text{dist}}^{-1} \mathbf{f}_{\text{in}} + \sum_{p=0}^{P-1} \left(\left(c_p \mathrm{N}_p^K(\mathbf{A}_{\text{rad}}) \right) \odot \left(\tilde{\mathbf{D}}_{\text{dist}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{\text{dist}} \tilde{\mathbf{D}}_{\text{dist}}^{-\frac{1}{2}} \right) \right) \mathbf{f}_{\text{in}}$$

mit den freien Parametern $[c_0, \ldots, c_P]^\top \in \mathbb{R}^{P+1}$, wobei N_p^K elementweise auf die Matrix $\mathbf{A}_{\text{rad}}$ angewendet wird. $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ sammelt dabei die aktuellen Merkmale an den einzelnen Knoten über $c_P \tilde{\mathbf{D}}_{\text{dist}}^{-1} \mathbf{f}_{\text{in}}$ und aggregiert diese mit den Merkmalen der lokalen Nachbarschaft über den polynomiellen Filter $\left(c_p \mathrm{N}_p^K(\mathbf{A}_{\text{rad}}) \right) \odot \left(\tilde{\mathbf{D}}_{\text{dist}}^{-1/2} \tilde{\mathbf{A}}_{\text{dist}} \tilde{\mathbf{D}}_{\text{dist}}^{-1/2} \right)$, der so die Länge und Richtungen der Kanten berücksichtigt. Für $K = 0$ ist die Faltung über den B-Splines äquivalent zu der Faltung über den Partitionen des Graphen

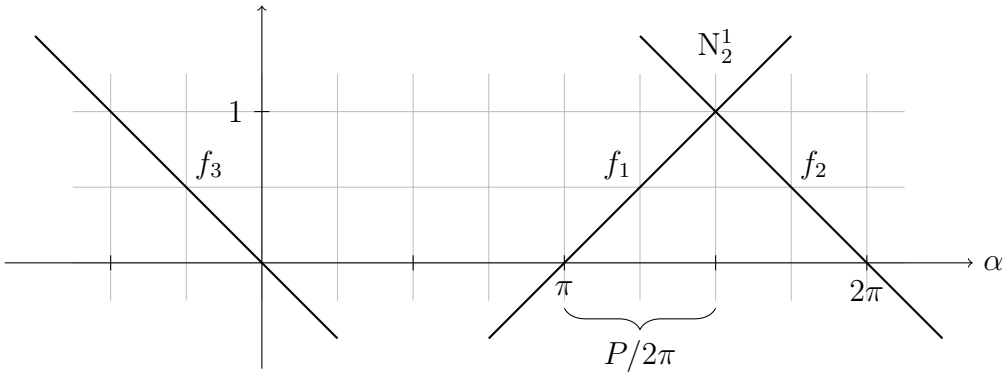


Abbildung 5.5: Beweisidee zur „kreisförmigen“ Dreiecksfunktion für B-Spline-Funktionen mit $K = 1$ und $P = 4$ als Darstellung über eine Minimum-Maximumfunktion auf drei Geraden, hier illustriert am Beispiel N_2^1.

aus (5.18) aufgrund der Rechteckfunktionen $\left\{\mathrm{N}_p^0\right\}_{p=0}^{P-1}$.

Implementierung. Für Merkmalsmatrizen $\mathbf{F}_{\text{in}} \in \mathbb{R}^{N \times M_{\text{in}}}$ und $\mathbf{F}_{\text{out}} \in \mathbb{R}^{N \times M_{\text{out}}}$ kann die Faltung über einem Filtertensor $\mathbf{W} \in \mathbb{R}^{(P+1) \times M_{\text{in}} \times M_{\text{out}}}$ damit als

$$\mathbf{F}_{\text{out}} \coloneqq \tilde{\mathbf{D}}_{\text{dist}}^{-1}\mathbf{F}_{\text{in}}\mathbf{W}_{P+1} + \sum_{p=0}^{P-1}\left(\mathrm{N}_p^K(\mathbf{A}_{\text{rad}}) \odot \left(\tilde{\mathbf{D}}_{\text{dist}}^{-\frac{1}{2}}\tilde{\mathbf{A}}_{\text{dist}}\tilde{\mathbf{D}}_{\text{dist}}^{-\frac{1}{2}}\right)\right)\mathbf{F}_{\text{in}}\mathbf{W}_{p+1}$$

beschrieben werden. Im Vergleich zu Kapitel (5.4.1) erhöht sich der Aufwand der Faltung zu $\mathcal{O}(KPM_{\text{in}}M_{\text{out}})$, da die Partitionen der Adjazenzmatrizen nicht mehr disjunkt, sondern $(K+1)$-mal betrachtet werden.
Für $K = 1$ lässt sich weiterhin die „kreisförmige“ Dreiecksfunktion $\mathrm{N}_p^1(\alpha)$ aus einer Reihe von Minimum- und Maximumfunktionen effizient implementieren. So gilt, dass $\mathrm{N}_p^1(\alpha)$ zu

$$\mathrm{N}_p^1(\alpha) = \max\Bigg(\min\Big(\max\Big(\frac{P}{2\pi}\alpha - p, 0\Big), \max\Big(-\frac{P}{2\pi}\alpha + p + 2, 0\Big)\Big), \max\Big(-\frac{P}{2\pi}\alpha + p + 2 - P, 0\Big)\Bigg)$$

vereinfacht werden kann. Dies lässt sich verifizieren, indem die kreisförmige Dreiecksfunktion im Intervall $(0, 2\pi]$ als Verknüpfung dreier Geraden verstanden wird: zwei Geraden, die das Dreieck im Intervall $(2\pi p/P, 2\pi(p+2)/P]$ aufspannen, und einer absteigenden Geraden, die um 2π nach links verschoben wurde und dafür sorgt, die B-Spline-Funktion für $p = P - 1$ kreisförmig abzuschließen und in allen anderen Fällen keinen Anteil im Gültigkeitsbereich der Funktion $\mathrm{N}_p^1\colon (0, 2\pi] \to [0, 1]$ besitzt

(vgl. Abbildung 5.5). Den Geraden kann die Steigung $P/2\pi$ bzw. $-P/2\pi$ zugeordnet werden und sie können damit folglich über

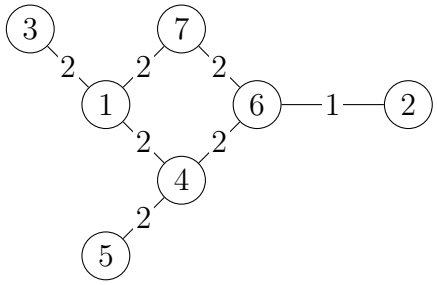
$$
\begin{aligned}
f_1(\alpha) &:= \frac{P}{2\pi}\left(\alpha - 2\pi\frac{p}{P}\right) = \frac{P}{2\pi}\alpha - p \\
f_2(\alpha) &:= -\frac{P}{2\pi}\left(\alpha - 2\pi\frac{p+2}{P}\right) = -\frac{P}{2\pi}\alpha + p + 2 \\
f_3(\alpha) &:= -\frac{P}{2\pi}\left(\alpha - 2\pi\frac{p+2}{P} + 2\pi\right) = -\frac{P}{2\pi}\alpha + p + 2 - P
\end{aligned}
$$

beschrieben werden. Mittels $\max(f_i(\alpha), 0) \in \mathbb{R}_+$ können diese auf den positiven reellen Raum eingegrenzt werden. $f_\triangle := \min(\max(f_1, 0), \max(f_2, 0))$ beschreibt damit die Dreiecksfunktion, indem sie sich stets für das Minimum von $\max(f_1, 0)$ bzw. $\max(f_2, 0)$ entscheidet. Folglich beschreibt $\max(\max(f_3, 0), f_\triangle)$ die B-Spline-Funktion N_p^1 im Intervall $(0, 2\pi]$ für alle $p \in \{0, \ldots, P-1\}$.

5.5 Pooling auf Graphen

Neben den Faltungsschichten gehören die sogenannten Poolingschichten zu den fundamentalen Schichten eines CNNs. Poolingschichten eines Netzes, üblicherweise über Max-Pooling realisiert, werden dabei für gewöhnlich direkt nach einer Faltungsschicht benutzt und sorgen dafür, die Ausgabe der Faltung zu filtern bzw. zu reduzieren [35].

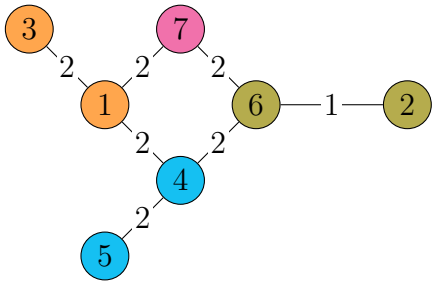
Eine Pooling-Operation auf Graphen erfordert dafür analog zum Pooling auf einem regulären Gitter eine logische Zusammenfassung von Knoten zu Clustern [8]. Der zu verwendende Clusteralgorithmus hat dabei jedoch einige Anforderungen, um als Poolingoperation in einem Netz genutzt werden zu könnnen. So muss dieser vor allem als ein mehrstufiges Clustering fungieren, bei dem jede Stufe eines Graphen den Blick auf den Graphen bei einer unterschiedlichen „Auflösung“ zeigt und insbesondere die zugrunde liegende Geometrie des Graphen erhält [8]. Das Clustering von Knoten wird daher in dieser Arbeit auch oft als *Vergröberung* eines Graphen betitelt. Die mehrfache Anwendung eines Clusteralgorithmus erlaubt damit die mehrfache Benutzung von Poolingschichten im Netz. Es ist weiterhin notwendig, dass die Poolingoperation benutzerspezifische Poolinggrößen ermöglicht. Hier erscheinen vor allem Clustertechniken sinnvoll, die die Größe eines Graphen um den Faktor zwei reduzieren [8]. Damit können größere Poolinggrößen über die mehrfache Anwendung des Clusteralgorithmus realisiert werden.



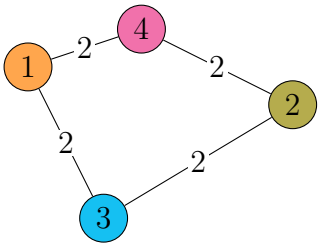
(a) Graph mit zufälliger Ordnung

Kante	Gewicht	Normalized-Cut
$(1,3)$	2	$\mathbf{1.\bar{3}}$
$(1,4)$	2	$0.\bar{3}$
$(1,7)$	2	$0.8\bar{3}$
$(2,6)$	1	$\mathbf{1.2}$
$(4,5)$	2	$\mathbf{1.\bar{3}}$
$(4,6)$	2	$0.7\bar{3}$
$(6,7)$	2	$\mathbf{0.9}$

(b) Bestimmung des Normalized-Cuts



(c) Clustering der Knoten



(d) vergröberter Graph

Abbildung 5.6: Vergröberung eines Graphen $\mathcal{G}$ mittels Graclus und Normalized-Cut bei zufälliger Knotenordung, hier angegeben über die Knotenindizes von $\mathcal{G}$ (a). Die Maxima des Normalized-Cuts (b) zwischen den Knoten und deren lokalen Nachbarschaften bestimmen in aufsteigender Reihenfolge das (höchstens) paarweise Clustering von $\mathcal{V}$, insbesondere sorgt der Normalized-Cut für die präferierte Verschmelzung mit den Blättern des Graphen (c). Der vergröberte Graph ergibt sich dann als clusterweise Aufsummierung der Kanten ohne Schleifen (d).

5.5.1 Graphvergröberung

Es existiert eine Reihe von Clustertechniken auf Graphen [8, 9, 32]. Defferrard u. a. benutzen für die Poolingschicht eines Netzes auf Graphen die *Vergröberungsphase* des mehrstufigen Clusteralgorithmus *Graclus* [9]. Graclus zeigt sich dabei als besonders erfolgreich über einer Vielzahl von Graphen bei minimaler Berechnungskomplexität [8]. Abbildung 5.6 illustriert den Ablauf des Algorithmus. Dabei wird ein initialer Graph $\mathcal{G}_0$ sukzessive in kleinere Graphen $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_L$ mit $|\mathcal{V}_0| > |\mathcal{V}_1| > \cdots > |\mathcal{V}_L|$, $L \in \mathbb{N}$, transformiert [9]. Für die Transformation von einem Graphen $\mathcal{G}_l$ zu einem Graphen $\mathcal{G}_{l+1}$ mit kleinerer Knotenanzahl $|\mathcal{V}_{l+1}| < |\mathcal{V}_l|$ werden aus disjunkten Knotenuntermengen von $\mathcal{V}_l$ *Superknoten* für $\mathcal{V}_{l+1}$ gebildet [9]. Die Kantengewichte $\mathcal{E}_{l+1}$ werden dann über die Summe der Kantengewichte der je-

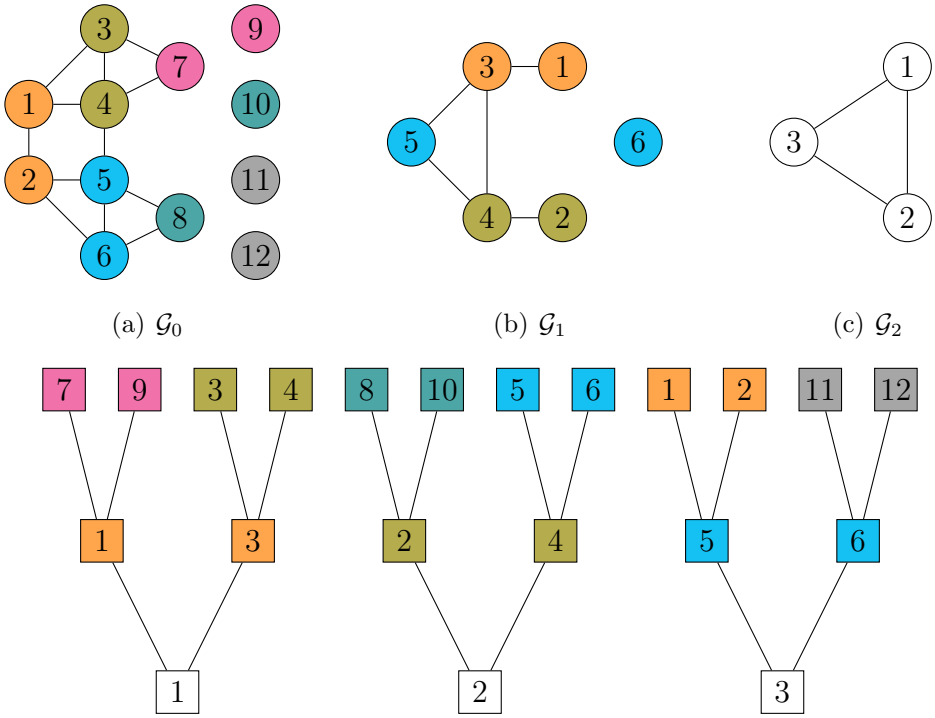
weiligen Knotenuntermengen ohne Schleifen gebildet [9].
Die Auswahl der Untermengen erfolgt gierig. Die Knoten des Graphen $\mathcal{G}_l$ werden als unmarkiert initialisert und zufällig durchlaufen. Für jeden Knoten $v_i \in \mathcal{V}_l$, der noch unmarkiert ist, wird ein lokaler, ebenfalls noch unmarkierter, Nachbarschaftsknoten $v_j \in \mathcal{N}(v_i)$ nach einer zuvor definierten Strategie bestimmt und v_i sowie v_j zu einem Superknoten $v^* \coloneqq \{v_i, v_j\} \in \mathcal{V}_{l+1}$ verschmolzen. Anschließend werden v_i sowie v_j markiert. Falls v_i keinen unmarkierten Nachbarknoten besitzt, wird v_i alleine als Superknoten $v^* \coloneqq \{v_i\} \in \mathcal{V}_{l+1}$ deklariert und markiert. Der Algorithmus ist abgeschlossen, sobald alle Knoten erfolgreich markiert wurden [9]. Bei weiteren Anwendungen des Clusteralgorithmus werden die Knoten im Folgenden anhand ihrer aufsteigenden Knotengrade durchlaufen [8].
Strategien für die Nachbarschaftsauswahl basieren üblicherweise auf der Maximierung von $w(v_i, v_j)$ [9]. Eine darauf aufbauende Strategie ist der *Normalized-Cut*

$$\mathrm{NCut}(v_i, v_j) \coloneqq w(v_i, v_j)\left(\frac{1}{d(v_i)} + \frac{1}{d(v_j)}\right),$$

der die Kantengewichte relativ zu ihrem Knotengrad gewichtet [8, 9]. Der Normalized-Cut sorgt dafür, dass Kanten als wichtiger gezählt werden, wenn ihre entsprechenden Knoten einen geringen Grad besitzen und damit folglich als unwahrscheinlicher gelten, mit einem anderen Knoten zu verschmelzen.
Graclus reduziert die Knotenanzahl eines beliebigen Graphen näherungsweise um die Hälfte, d.h. $2|\mathcal{V}_{l+1}| \approx |\mathcal{V}_l|$. Es können jedoch vereinzelt Superknoten entstehen, die nur aus einem einzigen Knoten der vorangegangenen Schicht gebildet wurden. In der Praxis zeigt sich jedoch, dass Graclus nur sehr wenige solcher Superknoten generiert [8]. Es ist weiterhin wichtig anzumerken, dass der Algorithmus bei mehrmaliger Anwendung auf dem gleichen Graphen aufgrund seiner zufälligen Iteration auf den Knoten zwei verschiedene, vergröberte Graphen erzeugen kann. Damit kann der Prozess des Clusterings bereits als ein Augmentierungsschritt der Eingabedaten verstanden werden.

5.5.2 Effizientes Pooling mittels binärer Bäume

Ein tiefes neuronales Netz besitzt für gewöhnlich viele Poolingschichten. Eine Pooling-Operation auf den Merkmalen der Graphknoten muss folglich vor allem effizient sein. Ein naiver Ansatz dafür ist eine *Lookup-Tabelle*, in der die Verbindungen der Knoten zu ihren Superknoten gespeichert sind. Eine Poolingoperation muss demnach für jedes Cluster ihre Knoten in der Tabelle referenzieren und ihre Operation auf



(a) $\mathcal{G}_0$ (b) $\mathcal{G}_1$ (c) $\mathcal{G}_2$

(d) Anorderung der Knoten zu binären Bäumen

Abbildung 5.7: Illustration einer Poolingoperation der Größe 4 (bzw. zweier Poolingoperationen der Größe 2) auf einem Graphen $\mathcal{G}$ der Größe $|\mathcal{V}| = 8$. Das Clustering von $\mathcal{G}$ liefert uns im ersten Schritt $N_1 = |\mathcal{V}_1| = 5$ Knoten und im darauf folgenden $N_2 = |\mathcal{V}_2| = 3$ Knoten. Die Größen der Graphen werden daraufhin zu $N_2 = 3$, $N_1 = 6$ und $N_0 = 12$ angepasst, sodass jeder Knoten auf genau 2 Vorgängerknoten verweist, indem Fakeknoten zu $\mathcal{G}_1$ (1 Knoten) und $\mathcal{G}_0$ (4 Knoten) hinzugefügt werden. Mit der Anordnung der Knoten zu binären Bäumen (d) kann die Poolingoperation $\max(\cdot)$ eines Signals $\mathbf{f} \in \mathbb{R}^{12}$ auf $\mathcal{G}_0$ dann effizient als $\mathbf{f}_{\text{pool}} := [\max(\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3, \mathbf{f}_4), \max(\mathbf{f}_5, \mathbf{f}_6, \mathbf{f}_7, \mathbf{f}_8), \max(\mathbf{f}_9, \mathbf{f}_{10}, \mathbf{f}_{11}, \mathbf{f}_{12})]^\top \in \mathbb{R}^3$ implementiert werden, wobei die Werte der Fakeknoten $\mathbf{f}_2, \mathbf{f}_6, \mathbf{f}_{11}, \mathbf{f}_{12}$ auf den neutralen Wert Null gesetzt werden.

diesen vollführen. Das resultiert in einer extrem speicherineffizienten, langsamen und schwer zu parallelisierenden Implementierung [8]. Es ist jedoch möglich, die Knoten des Graphen so anzuordnen, dass eine Poolingoperation auf diesen in linearer Zeit, d.h. $\mathcal{O}(N)$, realisiert werden kann [8].

Nach jedem Vergröberungsschritt besitzt ein Knoten entweder genau einen oder zwei Kinder, je nachdem ob es zum Zeitpunkt der Betrachtung noch einen unmarkierten Nachbarn gibt. Falls ein Knoten v im Graphen $\mathcal{G}_{l+1}$ nur ein Kind besitzt, so wird ein *Fakeknoten* ohne Kanten in den Graphen $\mathcal{G}_l$ hinzugefügt [8]. Damit besitzt jeder Knoten folglich nun immer zwei Kinder. Foglich kann über die Eltern-Kind-

Beziehung von der L-ten bis zur 0-ten Stufe ein balancierter, binärer Baum aufgebaut werden. Reguläre Knoten aus dem Graphen besitzen damit entweder (a) genau zwei reguläre Knoten, (b) einen regulären Knoten und einen Fakeknoten als Kinder und (c) Fakeknoten besitzen immer genau zwei Fakeknoten als Kinder [8]. Abbildung 5.7 illustriert die Konstruktion eines solchen Baumes. Die Merkmale an den Fakeknoten eines Graphen werden mit einem neutralen Wert initialisiert, d.h. zum Beispiel mit Null bei einem Netz mit ReLU-Aktivierungsfunktion und der Verwendung von Max-Pooling als Poolingoperation [8]. Ebenso kann auf diesen Fakeknoten weiterhin ohne Einschränkungen gefaltet werden, da sie keinerlei Nachbarknoten besitzen. Die Anordnung der Knoten zu einem balancierten, binären Baum liefert uns eine Ordnung auf den Knoten von $\mathcal{G}_0$ bis $\mathcal{G}_{L-1}$, auf der wir eine Poolingoperation völlig analog zu einem eindimensionalen Gitter mit einer Größe und Schrittweise von zwei definieren können [8]. Diese Anordnung der Knoten macht den Prozess des Poolings extrem effizient und erlaubt ebenso eine parallele Verarbeitungsarchikektur auf GPUs [8].

Größere Poolinggrößen müssen ein Vielfaches von zwei sein und können dann ebenso leicht über eine tieferstufigere Vergröberung implementiert werden (vgl. Abbildung 5.7).

5.5.3 Erweiterung auf Graphen im zweidimensionalen Raum

Sei $\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l, p_l)$ ein Graph im zweidimensionalen euklidischen Raum, wobei die Position der Knoten des Graphen eindeutig über die Funktion $p_l \colon \mathcal{V}_l \to \mathbb{R}^2$ bestimmt ist (vgl. Kapitel 3). Dann lässt sich der mehrstufige Clusteralgorithmus Graclus aus Kapitel 5.5.1 insofern anpassen, dass dieser die Vergröberung eines Graphen auch in Bezug auf die Position seiner Superknoten in der Ebene wiederspiegelt. Sei dafür weiterhin $v^* \coloneqq \{v_i, v_j\} \in \mathcal{V}_{l+1}$, ein Superknoten der Knoten $v_i, v_j \in \mathcal{V}_l$. Dann ergibt sich die neue Position von v^* als

$$p_{l+1}(v^*) \coloneqq \frac{p_l(v_i) + p_l(v_j)}{2}.$$

Für Superknoten $v^* \coloneqq \{v_i\} \in \mathcal{V}_{l+1}$ mit einem einzigen Knoten $v_i \in \mathcal{V}_l$ bleibt die Position unverändert mit $p_{l+1}(v^*) \coloneqq p_l(v_i)$. Die Adjazenzmatrizen $\mathbf{A}_{\text{dist}} \in [0,1)^{N \times N}$ und $\mathbf{A}_{\text{rad}} \in [0, 2\pi]^{N \times N}$ zu $\mathcal{G}_{l+1}$ können dann analog zu Kapitel 3 aus $\mathcal{V}_{l+1}$, $\mathcal{E}_{l+1}$ und p_{l+1} gewonnen werden.

Wohnt den Knoten in $\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l, p_l, m_l)$ wie in Kapitel 3.2 zusätzlich eine Masse $m_l \colon \mathcal{V}_l \to \mathbb{R}$ bei, so gewichtet sich die Position des Superknotens $v^* \in \mathcal{V}_{l+1}$ abhängig

der Massen $m(v_i)$ und $m(v_j)$ als

$$p_{l+1}(v^*) := \frac{m_l(v_i)p_l(v_i) + m_l(v_j)p_l(v_j)}{m_l(v_i) + m_l(v_j)}.$$

Die neue Masse des Superknotens ergibt sich dann als Vereinigung der Massen $m_{l+1}(v^*) := m_l(v_i) + m_l(v_j)$. Damit erhält der vergröberte Graph $\mathcal{G}_{l+1}$ die Positionen und Massen seiner Vorgängerknoten des Graphen $\mathcal{G}_l$ im zweidimensionalen euklidischen Raum.

5.6 Netzarchitektur

Die Architektur eines neuronalen Netzes auf Graphen mit spektralen Faltungen verhält sich durch die ähnliche Formulierung einer Faltungs- und einer Poolingschicht analog zu der Netzarchitektur klassischer CNNs. Dabei werden wie gewohnt mehrere Faltungsschichten mit stetig erhöhter Merkmalsausbreitung aneinander gereiht und an einigen Stellen über Poolingschichten getrennt, die die Anzahl der zu betrachtenden Knoten sukzessive reduzieren. Im Anschluss darauf finden sich im Allgemeinen zwei bis drei vollverbundene Schichten, die die Merkmalsgröße dann schlussendlich auf die gewünschte Ausgabegröße reduzieren [35]. Die mehrmalige Verkettung von Faltungsschichten sorgt dafür, dass auch bei einer relativ kleinen Faltung über die lokale Nachbarschaft eines jeden Knoten Merkmale weit entfernterer Knoten gewonnen werden können.

Ein CNN auf Bildern erfordert dabei in einer analogen Netzarchitektur eine feste Eingabegröße. Dafür werden die Bildermengen in der Regel skaliert und zugeschnitten, sodass diese alle die gleiche Bildgröße besitzen (zum Beispiel 224×224) [18]. Es erscheint jedoch schwierig, eine Menge von Graphen insofern anzupassen, dass diese alle die gleiche Anzahl an Knoten aufweisen. So ist es zwar vorstellbar, zusätzliche Fakeknoten zu jedem Graphen hinzuzufügen, damit diese alle eine feste Anzahl an Knoten aufweisen. Neben dem erhöhtem Speicheraufwand ist dieser Ansatz jedoch insbesondere nicht geeignet für unbekannte Graphen, die in das Netz eingespeist werden. So können diese eventuell eine größere Anzahl als die zuvor festgelegte Größe aufweisen. Ebenso liefert uns der Prozess der Graphvergröberung eine stets unterschiedliche Repräsentation eines Graphen, die wohlmöglich bereits eine größere Menge an Fakeknoten hinzufügt und dabei die festgelegte Knotengröße der Graphen überschreitet. Es ist weiterhin schwierig, einen Graphen auf eine feste Größe zuzuschneiden. So ist es insbesondere nicht ersichtlich, welche Knoten aus einem

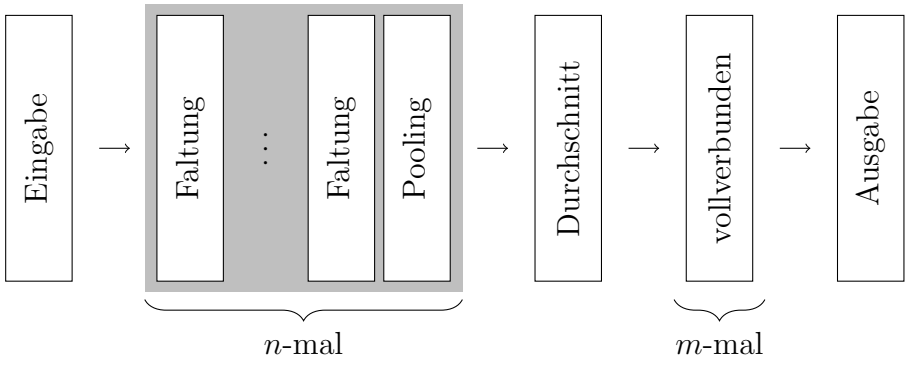


Abbildung 5.8: Typische spektrale Netzarchitektur auf Graphen bestehend aus beliebig vielen verketteten Faltungs- sowie Poolingschichten. Die Durchschnittsbildung erlaubt die Verwendung vollverbundener Schichten zur Ausgabe.

Graphen entfernt oder zusammengefasst werden können.
Die Architektur eines neuronalen Netzes auf Graphen erfordert folglich eine Struktur, die auf dynamischen Eingabegrößen operiert. Dazu muss jedoch zuerst geklärt werden, warum ein klassisches CNN auf Bildern eine feste Eingabegröße erfordert, denn die Faltungsschichten eines CNNs können Merkmalskarten beliebiger Größe generieren [18]. Die vollverbundenen Schichten jedoch brauchen rein nach ihrer Definition eine feste Eingabegröße [18]. Dadurch ergibt sich die Bedingung einer festen Eingabegröße lediglich durch den Übergang von einer Faltungs- zu einer vollverbundenen Schicht [18]. Eine einfache und elegante Lösung zur Umgehung dieses Problems bietet uns eine weitere Schicht, die zwischen der Faltungs- und vollverbundenen Schicht des Netzes hängt und dafür sorgt, die Ausgabekarten der letzten Faltungsschicht auf eine feste Größe zu projizieren. Die Operation, die dafür üblicherweise genutzt wird, ist die Durchschnittsbildung eines Merkmals über all seinen Knoten. Da die Anzahl der betrachteten Merkmale pro Knoten in jeder Schicht fest ist, liefert uns die Durchschnittsbildung zwischen der Faltungs- und der vollverbundenen Schicht eines Netzes stets eine feste Anzahl an zu betrachtenden Merkmalen. Es ist jedoch anzumerken, dass diese Operation auf Bildern insbesondere translationsinvariant ist und folglich die Position der Merkmale im Bild verloren gehen. Graphen, kodiert als Adjazenzmatrizen $\mathbf{A}_{\text{dist}}$ und $\mathbf{A}_{\text{rad}}$, sind jedoch bereits translationsinvariant und folglich gehen dabei keine Informationen verloren. Es ist jedoch darauf zu achten, dass die Anzahl der Knoten in der letzten Faltungsschicht relativ klein ist und Merkmale auf den Knoten folglich bereits den gesamten Graphen abdecken sollten.
Abbildung 5.8 veranschaulicht die beschriebene typische spektrale Netzarchitektur. Alternative Architekturen werden in Kapitel 7 diskutiert.

6 Evaluation

Das folgende Kapitel evaluiert die beschriebenen Ansätze der Graphrepräsentationen von Bildern aus Kapitel 3 sowie die räumlichen und spektralen Ansätze zum Lernen auf diesen (Kapitel 4 und 5) am Beispiel der Bildklassifizierung über mehreren Datensätzen. Dafür wird zuerst auf den Prozess der Merkmalsselektion eingegangen sowie der Versuchsaufbau inklusive der verwendeteten Metriken erläutert. Dabei werden insbesondere drei ausgewählte zu evaluierende Datensätze vorgestellt und auf die diesbezüglich ermittelten Superpixelparameter eingegangen. Im Anschluss finden sich die jeweiligen erreichten Ergebnisse der neuronalen Netze für die verschiedenen Ansätze und Datensätze sowie eine Laufzeitanalyse. Abschließend werden die Verfahren zum Lernen auf Graphen auf Basis der Ergebnisse in einer Diskussion rekapituliert und zusammengefasst.

6.1 Merkmalsselektion

Die in Kapitel 3.2.2 ermittelten 38 Formmerkmale auf den Knoten eines Graphen, der aus einer Superpixelrepräsentation generiert wurde, besitzen eine hohe Dimensionalität. Viele der Formmerkmale bauen dabei aufeinander auf und es ist daher fraglich, inwieweit die gesamte Menge an Merkmalen gebraucht wird oder ob eine Untermenge dieser als ausreichend gilt. Eine *Hauptkomponentenanalyse (PCA)* auf den Knotenmerkmalen kann dabei helfen, die reale Dimensionalität der Daten abzuschätzen. Dafür werden die Merkmale durch eine Linearkombination ihrer Hauptkomponenten beschrieben und können so durch weitaus weniger Dimensionen dargestellt werden. Abbildung 6.1 zeigt dabei die *kumulative Varianzaufklärung* der Hauptkomponenten einer PCA, d.h. die Varianzabdeckung der Merkmale durch die Hauptkomponenten, in Abhängigkeit ihrer Anzahl. Dabei zeigt sich, dass nach bereits recht wenigen Komponenten (≥ 9) der Großteil des Merkmalsraums abgedeckt werden kann.
Im Kontext von neuronalen Netzen ist die Verwendung einer PCA untypisch, denn schließlich müssen dafür weiterhin alle 38 Formmerkmale berechnet werden, um dar-

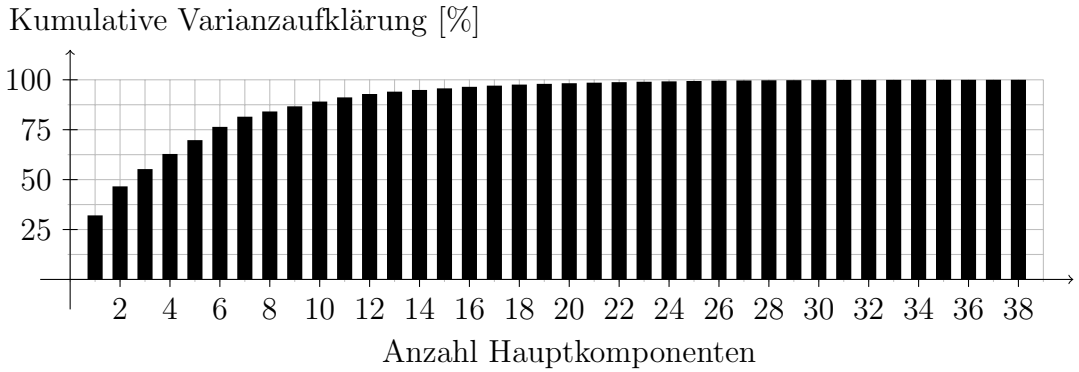


Abbildung 6.1: Kumulative Varianzabdeckung der Hauptkomponenten einer PCA in Abhängigkeit ihrer Anzahl. Die Merkmale wurden dabei aus den Knotenmerkmalen von 1000 segmentierten Bildern gewonnen. Bereits nach wenigen Hauptkomponenten ist der Größteil der erklärten Varianz abgedeckt.

aus die neuen Merkmale anhand der Hauptkomponenten zu ermitteln. Ein effizienteres Verfahren ist dagegen die Bestimmung einer Auswahl an Merkmalen, genannt *Merkmalsselektion*, die die Merkmalsmenge möglichst gut beschreibt. In dieser Arbeit kommen daher zwei Merkmalsselektionsalgorithmen zum Einsatz, die jeweils sequentiell auf der Menge der Merkmale dessen bedeutenste auswählen: die univariate Merkmalsselektion sowie die rekursive Merkmalseliminierung [38].

Die *univariate Merkmalsselektion* untersucht jedes Merkmal individuell auf Basis statistischer Tests und ermittelt daraus eine Bewertung der Wichtigkeit dieses Merkmals. Als statistischer Test kommt dafür die *Varianzanalyse* durch einen *F-Test* zum Einsatz (vgl. [38]). Dafür werden die Merkmale des Bildes je nach ihrer zugeordneten Klassifizierung in Gruppen eingeteilt. Die Varianzanalyse überprüft daraufhin, ob die Varianz zwischen den Gruppen größer als die Varianz innerhalb der Gruppen ist und kann folglich Schlussfolgerungen darüber ziehen, ob das Merkmal die Gruppe signifikant repräsentiert.

Die *rekursive Merkmalseliminierung* wählt Merkmale anhand ihrer Gewichte aus, indem rekursiv eine immer kleinere Menge an Merkmalen betrachtet wird [38]. Sei dafür eine *Schätzmethode* gegeben, die den Merkmalen jeweils ein Gewicht zuordnet. Zu Beginn errechnet die Schätzmethode die Gewichte aller Merkmale und die Merkmale mit den geringsten Gewichten werden aus der Merkmalsmenge eliminiert. Diese Prozedur wiederholt sich solange, bis die gewünschte Anzahl an Merkmalen erreicht wurde. Als Schätzer kommt dabei ein *Support Vector Machine (SVM)*-Klassifizierer mit der euklidischen Norm zum Einsatz [38].

6.2 Versuchsaufbau

Die Aufgabe einer *Klassifizerung* ist die Zuordnung einer Eingabe $\mathbf{x}$ zu genau einem Element einer endlichen Menge fest definierter Klassen $\mathcal{Y} = \{y_i\}_{i=1}^{Y}$ mit $|\mathcal{Y}| = Y$. Das neuronale Netz approximiert bzw. lernt dabei eine Funktion y, die eine beliebige Eingabe, d.h. ein Bild (oder in diesem Fall ein Graph), auf eine Y-dimensionale Ausgabe $y(\mathbf{x}) \in \mathbb{R}^Y$ überführt, wobei $y(\mathbf{x})_i$ die Konfidenz des Auftretens der Klasse y_i in der Eingabe beschreibt. Der Index des höchsten Werts des Ausgabevektors $y(\mathbf{x})$ gilt damit folglich als die Klasse der Eingabe $\mathbf{x}$, die diese am ehesten beschreibt.
Die Bildklassifizerung ist, gerade im Kontext neuronaler Netze bzw. CNNs, ein bereits weit erforschtes Gebiet. Es gibt eine Vielzahl an Datensätzen und Methodiken, die sich ausschließlich diesem Problem widmen. Gerade auch aufgrund ihrer vielen Vergleichsresultate zeichnet sie sich damit ideal für die Verifizierung der vorgestellten Faltungsoperatoren aus. Andere Probleme, wie etwa die Objekterkennung oder eine Segmentierung über eine Knotenklassifizierung, sind vorstellbar, werden aber im Rahmen dieser Arbeit nicht verfolgt.

6.2.1 Datensätze

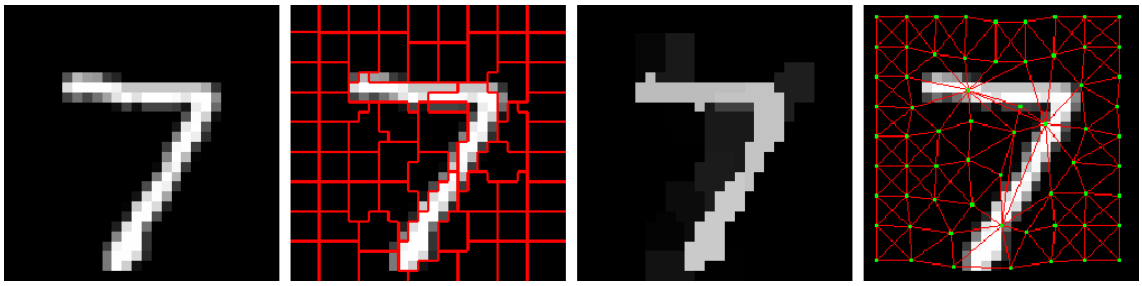
Die vorgestellten Faltungsmethoden aus Kapitel 4 und 5 bezüglich des Lernens auf Graphen im zweidimensionalen euklidischen Raum werden über einer Reihe von Datensätzen verifiziert, die im Folgenden vorgestellt werden. Dafür werden die Bildermengen in eine Superpixelrepräsentation (SLIC und Quickshift) konvertiert und darauf basierend in eine Graphrepräsentation transformiert (vgl. Kapitel 3). Zusätzlich zu der Präsentation der Datensätze enthält dieses Unterkapitel damit insbesondere die Parameterwahl der jeweiligen Superpixelalgorithmen, welche jeweils händisch über einer Untermenge der Bilder eines jeden Datensatzes ermittelt wurden. Weiterhin wird in diesem Unterkapitel abhängig von dem gewählten Datensatz und der Superpixelrepräsentation auf die entsprechenden Merkmalsselektionen der 38 Formmerkmale eingegangen, die nach dem beschriebenen Prinzip aus Kapitel 6.1 errechnet wurden.

MNIST. Der *Modified National Institute of Standards and Technology (MNIST)* Datensatz enthält eine große Menge eindeutig klassifizierter handgeschriebener Zahlen von 0 bis 9, welcher daher zum Lernen einer Schrift- bzw. Zahlenerkennung genutzt werden kann [30]. Er besteht aus 55000 Trainingsbildern, 5000 Validierungsbildern sowie 10000 Testbildern. Die Bilder des Datensatzes sind einheitlich auf die

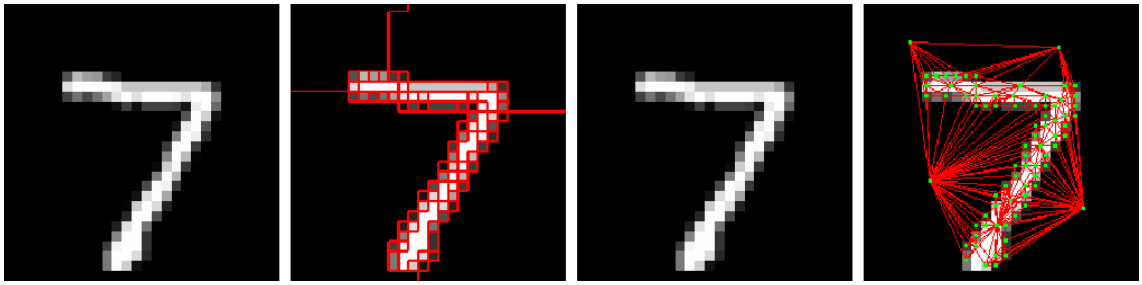
	Parameter	$\overline{N}$	$N_{\min}$	$N_{\max}$	$\overline{\deg}$	$\deg_{\min}$	$\deg_{\max}$
SLIC	$(K = 100, F = 5)$	64.6	50	80	5.7	1	19
Quickshift	$(\xi = 2, \alpha = 1, S = 2)$	82.1	5	154	6.8	1	101

Tabelle 6.1: Wahl der Superpixelparameter des MNIST Datensatzes.

Größe 28×28 skaliert und besitzen lediglich einen Farbkanal mit Grauwerten, welcher angibt, ob ein Pixel des Bildes zu einer Zahl (weiß), zu deren Rand oder zum Hintergrund (schwarz) gehört [30]. Aufgrund seiner kleinen Datengröße und leichten Handhabung gilt er als die ideale Einführung in Prinzipien des maschinellen Lernens und zeichnet sich damit insbesondere für die Verifizierung eines neuen Ansatzes bezüglich neuronaler Netze aus. Insbesondere kann der Datensatz während des gesamten Trainings im Hauptspeicher gehalten werden, was den Aufwand bezüglich der Verarbeitung und Eingabe der Daten auf ein Minimum reduziert.
Die ermittelten Parameter bezüglich der beiden benutzten Superpixelalgorithmen sind in Tabelle 6.1 gegeben. Für SLIC sind das die Parameter $K \in \mathbb{N}$, d.h. die Anzahl der gewünschten Segmente, sowie $F \in \mathbb{R}$ für die Gewichtung zwischen der Form und den Farbabgrenzungen der Superpixel. Für Quickshift ergeben sich dagegen drei wählbare Parameter: $\xi \in \mathbb{R}$ für die Wahl der Standardabweichung der Gaußfunktion, $\alpha \in \mathbb{R}$ für die Gewichtung des Farbterms sowie $S \in \mathbb{N}$ zur Einschränkung der Berechnung über ein Fenster der Größe $S \times S$. Für eine detaillierte Beschreibung der Parameter sei auf Kapitel 3.2.1 verwiesen.
Aus der Wahl der Superpixelparameter ergeben sich die ebenfalls in der Tabelle datierten Werte der durchschnittlichen, minimalen und maximalen Anzahl an Knoten $\overline{N}$, $N_{\min}$ bzw. $N_{\max}$ sowie dem durchschnittlichen, minimalen und maximalen Knotengrad $\overline{\deg}$, $\deg_{\min}$ bzw. $\deg_{\max}$ über der Menge aller aus den Bildern generierten Graphen bei einer Konnektivität von 8. Wohingegen SLIC über alle Bilder relativ gleich große Knotenmengen mit ähnlichem Knotengrad erzeugt, kann dies bei Quickshift je nach Bild stark variieren. So erzeugt Quickshift in dem MNIST Datensatz beispielsweise große schwarze Bereiche für den Hintergrund, die dementsprechend auch einen sehr hohen Knotengrad besitzen. Bei SLIC werden stattdessen auch die gleichfarbigen, schwarzen Flächen in einheitliche Intervalle unterteilt. Abbildung 6.2 veranschaulicht die beiden Superpixel- bzw. Graphrepräsentationen anhand eines Bildes aus dem MNIST Datensatz.
Die in Kapitel 6.1 beschriebene Merkmalsselektion reduziert für MNIST die Menge an Formmerkmalen (vgl. Kapitel 3.2.2) im ersten statistisch basierten Test auf 12 Merkmale, welche im zweiten Schritt rekursiv auf 9 Merkmale weiter beschränkt wer-



(a) SLIC



(b) Quickshift

Abbildung 6.2: Bild aus dem MNIST Datensatz, jeweils dargestellt als (1) Originalbild, (2) Superpixelrepräsentation, (3) Durchschnittsfarbe der Superpixel und (4) Graphrepräsentation.

SLIC	$\hat{x}$	$\hat{y}$	ecc	dia	ext	μ'_{20}	λ_2	axis_1	axis_2
Quickshift	$\hat{x}$	ecc	dia	ext	μ_{03}	μ_{21}	μ_{30}	η_{03}	ori

Tabelle 6.2: Merkmalsselektion des MNIST Datensatzes zu 9 Formmerkmalen.

den. Die ermittelten (unterschiedlichen) Formmerkmale aus Kapitel 3.2.2 für SLIC und Quickshift bezüglich MNIST sind in Tabelle 6.2 aufgezeigt. Wohingegen sich die Merkmalsselektion bei SLIC eher für Formmerkmale entscheidet, die aus den Momenten gewonnen werden können, genießen bei Quickshift die reinen translationsinvarianten Momente μ ein größeres Interesse. Daraus ergeben sich 10 Merkmale eines Knotens inklusive der Durchschnittsfarbe eines Superpixels.

CIFAR-10. Der *Canadian Institute for Advanced Research (CIFAR)* Datensatz, auch CIFAR-10 genannt, besteht aus 60000 farbigen Bildern, die jeweils genau einer von 10 Klassen zugeordnet sind [27]. 45000 Bilder werden dabei als Trainingsbilder, 5000 als Validierungsbilder und 10000 als Testbilder genutzt. Zu jeder Klasse existieren genau 6000 Bilder, welche gleichmäßig auf die Bilduntermengen aufgeteilt sind. Die Klassen der Bilder sind im Folgenden: Flugzeug, Auto, Vogel, Katze, Reh, Hund, Frosch, Pferd, Schiff und Lastwagen. Die Bilder der Klassen sind dabei *ein-*

	Parameter	$\overline{N}$	$N_{\min}$	$N_{\max}$	$\overline{\deg}$	$\deg_{\min}$	$\deg_{\max}$
SLIC	$(K = 200, F = 5)$	232.1	186	263	6.3	1	21
Quickshift	$(\xi = 1, \alpha = 1, S = 5)$	182.0	18	624	7.4	1	67

Tabelle 6.3: Wahl der Superpixelparameter des CIFAR-10 Datensatzes.

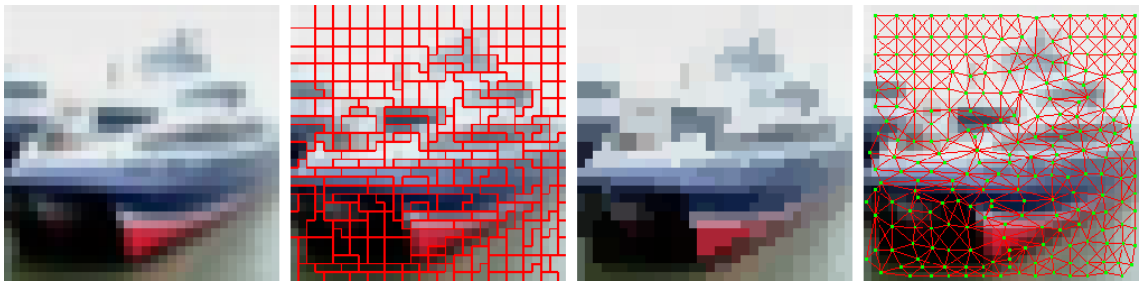
SLIC	$\mathbf{M}_{00}$	box_y	$\hat{x}$	dia	ext	$\mathbf{h}_1$	λ_1	axis_1	axis_2
Quickshift	box_y	box_x	$\hat{x}$	$\hat{y}$	ecc	ext	λ_1	axis_1	axis_2

Tabelle 6.4: Merkmalsselektion des CIFAR-10 Datensatzes zu 9 Formmerkmalen.

ander ausschließend. So enthält die Klasse „Auto“ nur kleinere Personenwagen, wohingegen die Klasse „Lastwagen“ auch nur als solche zu klassifizierenden Fahrzeuge enthält [27]. Die Bilder haben eine einheitliche Größe von 32×32 Pixeln und besitzen drei Farbkanäle. Sie passen aufgrund ihrer Größe ähnlich zu MNIST komplett in den Hauptspeicher. Aufgrund dessen ist CIFAR-10 für eine Bildklassifizierung sehr beliebt, da so schnelle Trainingszeiten garantiert sind und dabei trotzdem alle Techniken des Deep-Learnings ausgeschöpft werden müssen, um qualitativ hochwertige Resultate zu erzielen. Der CIFAR Datensatz liegt ebenfalls in einer zweiten Version vor, genannt *CIFAR-100*, der 60000 Bilder in 100 Klassen unterteilt, welcher aber in dieser Arbeit keine Verwendung findet [27].

Tabelle 6.3 zeigt die Wahl der Parameter der beiden Superpixelalgorithmen. Im Vergleich zu dem MNIST Datensatz wird dafür insbesondere für SLIC die approximierte Anzahl an Superpixeln von 100 auf 200 und für Quickshift die Größe des Fensters S von 2 auf 5 erhöht. Weiterhin zeigt die Tabelle erneut Informationen zu den generierten Graphen über die Anzahl der Knoten und ihrer Knotengrade. Hier lassen sich ebenfalls wieder die unterschiedlichen Vorgehensweisen der beiden Superpixelalgorithmen erkennen. Die maximale Anzahl an Knoten eines Graphen aus der Quickshift-Segmentierung liegt dabei mit 624 Knoten sehr hoch (im Durchschnitt werden nur zwei Pixel einem Superpixel zugeordnet) und kann als extremer Ausreißer gewertet werden. Abbildung 6.3 zeigt ein Bild aus dem CIFAR-10 Datensatz mit dessen entsprechenden Superpixel- bzw. Graphrepräsentationen.

Analog zu MNIST wurden für den CIFAR-10 Datensatz erneut 9 Formmerkmale nach dem gleichen Prinzip ermittelt. Tabelle 6.4 zeigt die berechnete Wahl der Merkmale der Selektion. Es ist auffällig, dass sich für die beiden Superpixelrepräsentation dabei die Selektion der Merkmale bei nur drei von neun Merkmalen unterscheidet. Inklusive den Durchschnittsfarbwerten eines Superpixels über den drei Farbkanälen



(a) SLIC

(b) Quickshift

Abbildung 6.3: Bild aus dem CIFAR-10 Datensatz, jeweils dargestellt als (1) Originalbild, (2) Superpixelrepräsentation, (3) Durchschnittsfarbe der Superpixel und (4) Graphrepräsentation.

ergeben sich daraus jeweils 12 Knotenmerkmale. Bei einer durchschnittlichen Anzahl an Knoten von 232.1 bei SLIC bzw. von 182.0 bei Quickshift führt dies zu einer Berechnung von 2785 bzw. 2184 Merkmalen eines Graphen. Im Vergleich zu der Anzahl an Merkmalen des Originalbildes ($32 \times 32 \times 3 = 3072$) ergibt sich folglich eine Datenreduktion auf 90.66% bzw. 71.09% der Eingangsdaten. Das ist aufgrund der ursprünglichen Bildgrößen des CIFAR-10 Datensatzes eine nicht zu unterschätzende Datenreduktion, bei der kaum entscheidende Informationen des Bildes verloren gehen (vgl. Abbildung 6.3).

PASCAL VOC. Der *Pascal Visual Object Classes (PASCAL VOC)* Datensatz besteht aus 17126 farbigen Bildern beliebiger Größe, der aufgrund seiner ausführlichen Bildannotierungen nicht nur für eine Bildklassifizierung, sondern ebenfalls für Objektdetektionen oder eine Bildsegmentierung genutzt werden kann [11]. Zu jedem Bild stehen insbesondere Informationen zu den enhaltenden Objekten und deren Hüllkörpern zur Verfügung [11]. Ein Bild besitzt dabei minimal ein Objekt der 20 annotierten Objektklassen. Gleiche oder unterschiedliche Objektklassen können mehrfach in einem Bild existieren. Die enthaltenen Objekte der Bilder sind im Einzelnen [11]:

	Parameter	$\overline{N}$	$N_{\min}$	$N_{\max}$	$\overline{\deg}$	$\deg_{\min}$	$\deg_{\max}$
SLIC	$(K = 1600, F = 30)$	1540.9	1082	1839	6.5	1	50
Quickshift	$(\xi = 2, \alpha = 0.75, S = 8)$	2131.6	234	29010	7.7	1	256

Tabelle 6.5: Wahl der Superpixelparameter des PASCAL VOC Datensatzes.

- Person
- Vogel, Katze, Kuh, Hund, Pferd, Schaf
- Flugzeug, Fahrrad, Boot, Bus, Auto, Motorrad, Zug
- Flasche, Stuhl, Esstisch, Topfblume, Sofa, Bildschirm

Für die eindeutige Zuweisung eines Bildes zu genau einer Klasse dient das Objekt mit dem größtem Hüllkörper. Die Höhen und Breiten der Bilder reichen von 142 bis 500 Pixeln. Die durchschnittliche Bildgröße des Datensatzes beträgt 389×467 Pixel und ist damit um einiges größer als der zuvor betrachtete CIFAR-10 Datensatz bei gleichzeitig weitaus weniger Beispielbildern. PASCAL VOC besitzt weiterhin keine eindeutige Zuordnung der Bilder zu Trainings-, Validierungs und Testbildern. Es existiert zwar ein seperater Testdatensatz, jedoch stehen dessen Annotierungen nicht zur freien Verfügung. Folglich wird aus der Bildermenge eine zufällige Validierungs- bzw. Testmenge von 1500 Bildern generiert. Damit stehen noch 15626 Bilder für das Training eines neuronalen Netzes zur Verfügung.
Die ermittelten Superpixelparameter finden sich in Tabelle 6.5. SLIC generiert damit um die $K = 1600$ Superpixel pro Bild. Im Vergleich zu der Wahl der Parameter für kleinere Bilder erhöht sich ebenfalls die Wahl der Normalisierungskonstante $F = 30$ und gibt damit der Form der Superpixel in großen Bildern eine bessere Gewichtung. Für Quickshift wird auf ähnliche Weise die Gewichtung des Farbterms $\alpha = 0.75$ abgeschwächt, sodass Superpixel in ihrer möglichen Größe etwas beschränkter sind. Weiterhin vergrößert sich die Wahl der Größe des Fensters $S \times S$ auf $S = 8$. Die Wahl der gefundenen Parameter deckt sich damit größtenteils mit den üblichen Werten dieser in der Literatur [13]. Abbildung 6.4 veranschaulicht die Wahl der Parameter der beiden Superpixelalgorithmen.
Auch für PASCAL VOC bestimmt die Merkmalsselektion 9 Merkmale aus den 38 Formmerkmalen (vgl. Tabelle 6.6). Dabei wird sich erneut nicht für die eigentlichen Momente entschieden, sondern mehr für die Merkmale, die aus diesen gewonnen werden können. Auffällig ist für SLIC die Wahl des Merkmals „box“, da dieses bereits

(a) SLIC

(b) Quickshift

Abbildung 6.4: Bild aus dem PASCAL VOC Datensatz, jeweils dargestellt als (1) Superpixelrepräsentation, (2) Durchschnittsfarbe der Superpixel und (3) Graphrepräsentation.

SLIC	box	box_y	box_x	$\hat{x}$	$\hat{y}$	ecc	ext	$\mathbf{h}_1$	axis_1
Quickshift	$\mathbf{M}_{00}$	box_y	box_x	$\hat{x}$	dia	λ_1	λ_2	axis_1	axis_2

Tabelle 6.6: Merkmalsselektion des PASCAL VOC Datensatzes zu 9 Formmerkmalen.

durch box_y und box_x implizit gegeben ist. Ein Knoten eines Graphen besitzt damit inklusive dessen Durchschnittsfarbe 12 Merkmale.
Bei den (relativ) großen Bildern in PASCAL VOC ergibt sich aufgrund der Vorsegmentierung des Bildes eine erhebliche Datenreduktion. Anstatt der durchschnittlichen Anzahl an Merkmalen des Originalbildes ($389 \times 467 \times 3 = 544989$) erhalten wir im Durchschnitt bei der Verwendung von SLIC 18491 bzw. von Quickshift 25579 Merkmale, was einer Datenreduktion auf lediglich 3.39% bzw. 4.69% der Eingangsdaten entspricht. Dabei werden insbesondere bei Quickshift auch kleine, auffällige Flächen des Bildes erhalten, sodass sich größtenteils nur eine Datenreduktion in Bereichen einstellt, die gleichfarbene, uninteressante Flächen des Bildes beschreiben.

Weitere Datensätze. Es existiert eine Reihe weiterer Bilddatensätze, die zwar in dieser Arbeit nicht benutzt werden, aber dennoch hier eine kurze Erwähnung finden sollen. Der weitaus größte und beliebteste Datensatz zur Bildklassifizierung ist der *ImageNet* Datensatz, der aus mehr als 1.2 Millionen Bildern besteht, denen jeweils

eine von 1000 Klassen zugeordnet ist [42]. Er wird aufgrund seiner Größe und seiner Komplexität für die reine Verifizierung der Faltungsansätze auf Graphrepräsentationen von Bildern in dieser Arbeit nicht verwendet. Er bildet jedoch insbesondere die Basis weiterer Datensätze. So beinhaltet beispielsweise der *Tiny ImageNet* Datensatz eine Untermenge der Bilder von ImageNet (200 Klassen mit jeweils 600 Bildern), bei denen die Bilder auf eine einheitliche Größe von 64×64 Pixeln skaliert wurden [42]. PASCAL VOC bedient sich ebenso einer Untermenge der Bilder des ImageNet Datensatzes, die um entsprechende Annotationen, wie zum Beispiel den Hüllkörpern der Objekte, erweitert wurden [11]. Namenhaft zu erwähnen sei weiterhin der von CIFAR-10 inspirierte *STL-10* Datensatz von Stanford mit einer einheitlichen Bildergröße von 96×96 Pixeln [6]. Er enthält aber nur sehr wenige annotierte Bilder und findet seine Anwendung daher vor allem im unüberwachten Lernen: ein Ansatz, der in dieser Arbeit nicht verfolgt wird. Der *Street View House Numbers (SVHN)* Datensatz beinhaltet 600000 annotierte Bilder (32×32) von Hausnummern, die aus *Google Street View* extrahiert wurden [34]. Er enthält damit weitaus mehr Bilder als der zu ihm ähnliche MNIST Datensatz und versucht dabei ein weitaus schwereres Problem zu lösen: die Objekterkennung von Zahlen in natürlichen Szenen. Er kann jedoch insbesondere aufgrund der beliebigen Anzahl an Ziffern im Bild nicht für eine Bildklassifizierung genutzt werden.

6.2.2 Metriken

Eine typische Aktivierungsfunktion für die Ausgabe $\mathbf{y} \coloneqq y(\mathbf{x}) \in \mathbb{R}^Y$ eines neuronalen Netzes für ein multidimensionales Klassifizierungsproblem ist die *Softmax Regression*

$$\operatorname{softmax}(\mathbf{y}) \coloneqq \frac{\exp(\mathbf{y})}{\sum_{i=1}^{Y} \exp(\mathbf{y}_i)},$$

die einer Wahrscheinlichkeitsverteilung der Klassen $\mathcal{Y}$ für das Auftreten dieser in der Eingabe $\mathbf{x}$ entspricht [1, 35]. Damit gilt insbesondere $\operatorname{softmax}(\mathbf{y}) \in [0,1]^Y$ sowie $\sum_{i=1}^{Y} \operatorname{softmax}(\mathbf{y})_i = 1$ [35]. Aufgrund der Verwendung der Exponentialfunktion werden dabei die höchsten Werte in $\mathbf{y}$ hervorgehoben, wohingegen Werte, die signifikant unter dem Maximum liegen, abgeschwächt werden.

Neben der in Kapitel 2.3 vorgestellten quadratischen Kostenfunktion aus (2.1) findet sich in neuronalen Netzen häufiger die Verwendung der *Kreuzentropie* als Kostenfunktion, welche dem Netz ermöglicht, schneller bessere Entscheidungen zu treffen

(vgl. [35]). Die Kreuzentropie ist dabei über einer Eingabemenge $\mathcal{X}$ definiert als

$$H(\mathcal{X}) \coloneqq -\frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} \langle \hat{y}(\mathbf{x}), \log(y(\mathbf{x})) \rangle,$$

wobei $\hat{y}(\mathbf{x}) \in \{0,1\}^Y$ die *Ground-Truth* der Eingabe $\mathbf{x} \in \mathcal{X}$ als *One-Hot*-Kodierung beschreibt [1, 35]. $H(\mathcal{X})$ ist aufgrund der Intervalleinschränkung der jeweiligen Ausgaben $y(\mathbf{x})$ von $\mathbf{x} \in \mathcal{X}$ mit Hilfe der Softmax Regression stets positiv und wird umso kleiner, je ähnlicher sich $y(\mathbf{x})$ und dessen Ground-Truth $\hat{y}(\mathbf{x})$ werden [35].
Zur Evaluation der Ergebnisse einer Klassifizierung wird neben der Kostenfunktion die *Genauigkeit* (engl. *Accuracy*) über einer Eingabemenge $\mathcal{X}$ als

$$\text{accuracy}(\mathcal{X}) \coloneqq \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} \max(-|\text{argmax}(y(\mathbf{x})) - \text{argmax}(\hat{y}(\mathbf{x}))| + 1, 0)$$

definiert, wobei $\text{argmax}(\mathbf{y}) = n$ genau dann, wenn $\mathbf{y}_n \geq \mathbf{y}_m$ für alle $1 \leq m \leq Y$. Sie beschreibt den Anteil korrekt klassifizierter Eingabedaten aus der jeweiligen Gesamtmenge der Trainings-, Validierungs- oder Testdaten $\mathcal{X}$ und ist daher ein geeignetes Mittel, die Qualität eines neuronalen Klassifizierungsnetzes zu bestimmen [35].

6.2.3 Parameter

Neben den Superpixelparametern und der Anzahl bzw. Auswahl der zur Verfügung stehenden Formmerkmale besitzt die Vorverarbeitung der Daten sowie das neuronale Netz an sich zahlreiche weitere optimierbare Parameter.
So kann sich zum Beispiel bei der Graphgenerierung zusätzlich zwischen einer Konnektiviät von 4 bzw. 8, einer globalen oder lokalen Normierung sowie einer frei wählbaren Standardabweichung ξ für die Gewichtsbestimmung der Kanten nach (3.1) entschieden werden. Aufgrund der unmöglichen Abdeckung aller möglichen Parameterkombinationen wurden speziell diese für die Evaluation festgehalten. Alle generierten Graphen wurden folglich über einer Konnektivität von 8, einer globalen Normierung sowie einer Standardabweichung ξ von Eins generiert.
Für das neuronale Netz ergeben sich neben der Anzahl an Faltungs-, Pooling- und vollverbundenen Schichten sowie deren Größen und Anordnungen, die in den Ergebnissen in Kapitel 6.3 manuell aufgelistet sind, weitere einstellbare Größen. Alle Netze wurden mit einer Batch-Size von 64 trainiert. Die Gewichte des Netzes wurden zusammen verteilt mit einem Durchschnitt von Null und einer Standardabweichung von 0.1 initialisiert, wohingegen alle Biaswerte zu Beginn des Trainings einen konstanten Wert von 0.1 besitzen. ReLU wird als Aktivierungsfunktion für alle bis auf

die letzte neuronale Schicht verwendet. Für jede Poolingschicht kommt das Max-Pooling zum Einsatz. Die gewählte Lernrate γ des Netzes unterscheidet sich je nach getestetem Modell zwischen 0.001 und 0.0001. Sie wird dabei zusätzlich stufenweise exponentiell nach einer definierten Anzahl an Epochen reduziert.

Es werden ebenso Methodiken angewendet, die das Overfitting eines Netzes reduzieren. So finden sich speziell in den vollverbundenen Schichten Anwendungen der *Dropout*-Technik sowie der *L2-Regularisierung* (vgl. [28, 47]).

6.2.4 Augmentierung

Üblicherweise findet während des Trainings eines neuronalen Netzes eine *Augmentierung* der Eingabedaten statt, die es ermöglicht, die Anzahl der Trainingsbilder virtuell zu erhöhen und die Gefahr des Overfittings zu reduzieren [1]. So kann zum Beispiel ein Bild $\mathbf{B} \in \mathbb{R}^{H \times W \times C}$ vertikal an dessen Bildmitte über

$$\operatorname{flip}(\mathbf{B})_{yx} \coloneqq \mathbf{B}_{y,W-x}$$

gespiegelt werden [1]. Ebenso kann die Helligkeit eines Bildes, welches im RGB-Farbmodell in Gleitkommarepräsentation als $\mathbf{B} \in [0,1]^{H \times W \times C}$ vorliegt, über

$$\operatorname{brightness}(\mathbf{B}, \delta)_{yxc} \coloneqq \min(\max(\mathbf{B}_{yxc} + \delta, 0), 1)$$

um den Faktor $\delta \in [-1, 1]$ justiert werden [1]. Weiterhin ist die Kontrastanpassung eines RGB-Bildes $\mathbf{B} \in [0,1]^{H \times W \times C}$ um den Faktor $\delta \in [-1, 1]$ definiert als

$$\operatorname{contrast}(\mathbf{B}, \delta)_{yxc} \coloneqq \min\Big(\max\Big(\delta\Big(\mathbf{B}_{yxc} - \overline{\mathbf{B}}_c\Big) + \overline{\mathbf{B}}_c, 0\Big), 1\Big),$$

wobei $\overline{\mathbf{B}}_c \coloneqq 1/(WH) \sum_{x=1}^{W} \sum_{y=1}^{H} \mathbf{B}_{yxc}$ die Durchschnittsfarbe des Bildes bezüglich des Farbkanals $c \in \{1, \ldots, C\}$ beschreibt [1]. Es sind eine ganze Reihe weiterer Augmentierungsschritte denkbar, wie zum Beispiel das Zuschneiden des Bildes an dessen Rändern auf eine kleinere Auflösung (engl. *Cropping*) [1]. Alle Augmentierungsschritte werden dabei für ein Eingabebild zufällig ausgeführt. Insbesondere erhält δ dabei einen zufälligen Wert aus einem vordefinierten Intervall $\delta \in [-\delta_{\max}, \delta_{\max}]$.

Die vorgestellten Augmentierungsschritte sind sowohl für Graphen im zweidimensionalen euklidischen Raum bzw. dessen Knotenmerkmale möglich. Die vertikale Spiegelung des Graphen kann, falls diese im Graphkontext eine Bedeutung besitzt, analog zu der Operation auf Bildern über eine Anpassung der Winkel in $\mathbf{A}_{\text{rad}} \in [0, 2\pi]^{N \times N}$

realisiert werden, d.h.

$$\text{flip}(\mathbf{A}_{\text{rad}})_{ij} \coloneqq \begin{cases} 2\pi - (\mathbf{A}_{\text{rad}})_{ij}, & \text{wenn } (\mathbf{A}_{\text{rad}})_{ij} > 0, \\ 0, & \text{sonst.} \end{cases}$$

Für das Zuschneiden eines Graphen können analog zu einem Bild diejenigen Knoten eliminiert werden, dessen Positionen sich nicht in der resultierenden Auflösung befinden. Weiterhin können die Helligkeits- und Kontrastanpassungen der Farben auf den Farbmerkmalen der Merkmalsmatrix $\mathbf{F} \in \mathbb{R}^{N \times M}$ angewendet werden. Es ist jedoch anzumerken, dass eine Justierung der Farbwerte auf den Knoten eines Graphen, der durch eine Superpixelrepräsentation gewonnen wurde, nur bedingt sinnvoll ist, denn die Nachbarschaften des Graphen sowie dessen Formmerkmale bleiben bei dieser Art der Augmentierung unverändert. Ein Superpixelalgorithmus, welcher Superpixel größtenteils aufgrund der Farbwerte eines Bildes generiert, bildet letztendlich bei einer Augmentierung der Farbwerte eines Bildes auf eine komplett andere Superpixelrepräsentation ab. Eine realistischere Augmentierung der Eingabedaten ist folglich nur dann gegeben, wenn die Superpixelrepräsentation bzw. dessen Graph erst nach der Augmentierung der Farben des Bildes generiert wird.

6.2.5 Implementierung

Alle vorgestellten Methodiken wurden in der Programmiersprache *Python* implementiert und stehen unter der *MIT*-Lizenz zur freien Verfügung[1]. Es liegen ausführbare Dateien zum Trainieren der neuronalen Netze auf allen erwähnten Datensätzen bereit, die sich bei Ausführung automatisch herunterladen. Die Implementation erreicht dabei eine *Testabdeckung* (Anteil getesteter Quelltextzeilen) von 100% und wurde auf den Pythonversionen 2.7 und $\geq$ 3.4 getestet. Es liegen zusätzliche Installationsanweisungen bei und der Quelltext ist so dokumentiert, dass er mit Hilfe dieser Arbeit verstanden werden kann.

Benutzte Bibliotheken. Für die Modellierung und den Betrieb neuronaler Netze in Python wurde die Bibliothek `tensorflow` von Google verwendet [1]. Sie erlaubt die Modellierung eines Berechnungsgraphen, bei dem Knoten mathematische Operationen repräsentieren und die Kanten zwischen ihnen den Datenfluss multidimensionaler Tensoren steuern. So gut wie alle implementierten mathematischen Operationen in `tensorflow` besitzen eine Gradientenimplementierung, sodass die

[1] `https://github.com/rusty1s/embedded_gcnn`

Kosten der Ausgabe eines Berechnungsgraphen über das Backpropagation-Verfahren minimiert werden können. Dies erlaubt insbesondere den Bau der implementierten Faltungsoperatoren mit den in `tensorflow` zur Verfügung stehenden Operationen.

Zur Generierung der Superpixelrepräsentationen der Bildermengen wurde die Bibliothek `scikit-image` (`skimage`) benutzt [52]. Sie enthält die Implementierungen zu den Superpixelalgorithmen SLIC und Quickshift, die in dieser Arbeit benutzt werden. Sie enthält ebenso eine Implementierung zur Graphgenerierung einer Superpixelrepräsentation, die sich aber in Tests als ineffizient herausstellte und daher in dieser Arbeit nicht verwendet wird (vgl. Kapitel 6.4).

Für die Merkmalsextraktion, die Graphgenerierung, die Implementierung der Graphvergröberungen für das spektrale Lernen sowie die Generierung der Receptive-Fields für das räumliche Lernen wurden die Bibliotheken `numpy` und `scipy` verwendet [25, 51]. Beide Bibliotheken gehören aufgrund ihrer effizienten Implementierung in C zu der Standardwahl wissenschaflicher Berechnungen in Python. Alle dünnbesetzten Matrizen, d.h. Adjazenzmatrizen sowie Diagonalmatrizen, sind insbesondere auch als solche implementiert. Für die Merkmalsselektion kommt die Bibliothek `scikit-learn` zum Einsatz [38].

Eingabe und Vorverarbeitung der Daten. Für die Eingabe der Knotenmerkmale in ein neuronales Netz werden diese pro Graph in dessen *Standardnormalverteilung* überführt, d.h. [1]

$$\hat{\mathbf{f}}_i \coloneqq \frac{\mathbf{f}_i - \overline{\mathbf{f}}}{\xi},$$

wobei $\overline{\mathbf{f}} \coloneqq (1/N) \sum_{n=1}^{N} \mathbf{f}_n$ den Durchschitt der Merkmale und ξ die Standardabweichung von $\mathbf{f} \in \mathbb{R}^N$ beschreibt. Die Verteilung eines Merkmals eines Graphen liegt damit durchschnittlich bei Null und besitzt eine Standardabweichung von Eins. Die Überführung der Eingabedaten in die Standardnormalverteilung ist ein übliches Verfahren im Kontext von neuronalen Netzen.

Für die Vorverarbeitung der Daten werden zwei unterschiedliche Ansätze verfolgt, die sowohl Vor- wie auch Nachteile aufweisen und daher je nach Datensatz Verwendung finden. Dem Benutzer steht es dabei frei, die vorverarbeiteten Daten in einem eigenen Datensatz vor Trainingsbeginn zu speichern oder alle notwendigen Vorverarbeitungsschritte der Bilder zur Laufzeit während des Trainings für das aktuell zu lernende Bild zu vollziehen. Die Vorverarbeitung während des Trainings wurde dabei mit Hilfe mehrerer Threads implementiert, sodass die Laufzeit der Lernprozedur möglichst nicht gestört wird. Es zeigt sich, dass die Vorverarbeitung der Daten

während des Trainings für kleine Datensätze wie MNIST oder CIFAR-10 so gut wie keinen Einfluss auf die Lerngeschwindigkeit mit sich zieht. Für größere Datensätze wie PASCAL VOC ist dies aber nicht zu empfehlen, da der Aufwand der Vorverarbeitung die Lerndauer durch die größeren Bilder, gerade aufgrund der steigenden Laufzeiten der Superpixelalgorithmen, teilweise extrem beeinflussen kann. Eine gezielte Laufzeitanalyse der beiden Vorverarbeitungsmethoden findet sich für die einzelnen Datensätze in Kapitel 6.4. Ein entscheidener Vorteil der Vorverarbeitung zur Laufzeit ist jedoch die in Kapitel 6.2.4 angesprochene Augmentierung der Eingabedaten. Wohingegen für vorverarbeitete Daten nur eine Augmentierung auf Graphen möglich ist, kann das volle Ausmaß einer Augmentierung erst über die Vorverarbeitung zur Laufzeit gewonnen werden.

6.3 Ergebnisse

Dieses Kapitel stellt die Ergebnisse einer ausgewählten Untermenge der in dieser Arbeit vorgestellten Graphrepräsentationen von Bildern, den Ansätzen zum Lernen auf diesen sowie den beschriebenen Datensätzen und Superpixelalgorithmen vor. Zur Kenntlichmachung der einzelnen Ansätze erhält das räumliche Lernen auf Graphen dabei die Abkürzung „RCNN“ und das spektrale Lernen auf Graphen über den Tschebyschow-Polynomen bzw. dessen Einschränkung auf eine Filtergröße von Eins die Abkürzungen „SGCNN“ bzw. „GCN“. Der erweiterte Ansatz zum spektralen Lernen ist über „EGCNN“ gekennzeichnet. Alle Faltungen des EGCNNs wurden dabei mit einer Partitionsgröße von 8 bei einem Grad von Eins implementiert, um ein CNN mit einem 3×3 Filter zu simulieren.
Netzarchitekturen werden im Folgenden beispielsweise über ein Muster der Form „C32-P2-FC128“ beschrieben und signalisieren eine Faltungsschicht, die auf 32 Merkmalskarten abbildet, gefolgt von einer Poolingschicht, welche zwei Knoten vereint, und danach mit einer vollverbundenen Schicht zu 128 Neuronen verknüpft wird. Die für das spektrale Lernen bei dynamischen Eingabegrößen notwendige Durchschnittsbildung zwischen Faltungs- und vollverbundener Schicht wird als „Avg“ gekennzeichnet (vgl. Kapitel 5.6). Auf die explizite Erwähnung der Eingabe- und Ausgabeschichten der Netze wird übersichtshalber verzichtet.

MNIST. Für die Validierung der Faltungsansätze starten wir mit dem MNIST-Datensatz und der in Kapitel 3.1 beschriebenen Graphrepräsentation der Bilder über ein reguläres Gitter. Damit kann jedes Bild über den selben Graphen dargestellt wer-

Ansatz	**W**	Architektur	Genauigkeit [%]
RCNN	25	C64-FC1024	98.774
SGCNN	25	C32-P4-C64-P4-FC1024	98.888
GCN	1	C32-C32-P4-C64-C64-P4-FC1024	96.675
EGCNN	9	C32-C32-P4-C64-C64-P4-FC1024	99.145
klassisch	5×5	C32-P4-C64-P4-FC1024	99.189
klassisch	3×3	C32-C32-P4-C64-C64-P4-FC1024	99.139

Tabelle 6.7: Testgenauigkeiten eines Trainings auf einer MNIST-Gitterrepräsentation nach 10 Epochen für die vorgestellten Ansätze inklusive äquivalenter Netzarchitekturen bei Benutzung der klassischen Faltungsoperation. Die Spalte **W** kennzeichnet die benutzten Filtergrößen der einzelnen Faltungsschichten.

den. Insbesondere verfällt damit die Notwendigkeit der Durchschnittsbildung, da die Anzahl an Neuronen zu jedem Zeitpunkt klar definiert ist. Für das räumliche Lernen wurden 196 Knoten mit Schrittweite 4 bei $\delta = 1$ und einer Nachbarschaftsgröße von 25 besimmt (vgl. Kapitel 4). Die Graphvergröberung des spektralen Lernens wiederum generiert für die erste Schicht 976 Knoten ($28^2 = 784$ Pixelknoten und 192 Fakeknoten). Die Anzahl der generierten Fakeknoten kann jedoch aufgrund der zufälligen Permutation variieren. Tabelle 6.7 fasst die Netzarchitekturen und dessen erreichte Genauigkeiten der einzelnen Ansätze zusammen. Die Netzarchitekturen sind dabei an der von `tensorflow` vorgeschlagenen Architektur orientiert [1]. Alle Netze wurden einheitlich mit einer Lernrate von 0.001 trainiert. Es wurde auf die für MNIST eher untypische Augmentierung der Eingabedaten, die L2-Regularisierung sowie auf die Verminderung der Lernrate in Abhängigkeit zur Trainingsdauer verzichtet [1]. Es findet sich jedoch die Anwendung eines Dropouts von 0.5 vor der Ausgabeschicht. Defferrard u. a. benutzen zur Validierung des SGCNNs eine ähnliche Netzarchitektur auf MNIST und schlagen dazu eine Filtergröße von 25 vor, sodass die Anzahl der Parameter mit der Anzahl der klassischen Faltung über ein 5×5 Fenster korrespondiert [8]. Diese Faltung lässt sich im Kontext des spektralen Faltungsoperators auf einem Gitter der Größe 28×28 aber kaum noch als lokale Faltung verstehen, denn sie lässt Knoten in die Berechnung mit einfließen, die 25 Kanten weit vom Ursprungsknoten entfernt liegen, und betrachtet folglich außer bei Randknoten das globale Bild (vgl. Kapitel 5.2). Eine lokalere Faltung lässt sich mit dem RCNN, GCN und EGCNN gewinnen. Für das GCN zeigt sich dabei jedoch eine eindeutige Limitierung durch die geringe Filtergröße. Die entwickelten Faltungsansätze RCNN und EGCNN beweisen sich aufgrund der Berücksichtigung ihrer Kantenausrichtungen als die bessere Alternative. Insbesondere lässt sich die Äquivalenz des EGCNNs

Ansatz	**W**	Architektur	Genauigkeit [%]	
			SLIC	QS
RCNN	25	C64-FC1024	82.873	85.226
SGCNN	9	C64-P4-C128-P4-Avg	91.884	92.718
GCN	1	C64-C64-P4-C128-C128-P4-Avg	78.155	86.358
EGCNN	9	C64-C64-P4-C128-C128-P4-Avg	97.405	98.025

Tabelle 6.8: Testgenauigkeiten eines Trainings auf dem MNIST Datensatz nach 17 Epochen, dessen Bilder zuvor durch die Superpixelalgorithmen SLIC und Quickshift in irreguläre Graphrepräsentationen gebracht werden.

bezüglich regulärer Gitter verifizieren. Das räumliche Lernen zeigt sich aufgrund der Beschränkung auf eine Faltungsschicht, der eindimensionalen Knotenauswahl sowie der uneinheitlichen Nachbarschaftsanordnung an Randknoten als leicht schwächer.
Die in Tabelle 6.8 vorgestellten Netzarchitekturen zeigen die Genauigkeiten des Trainings auf irregulären Dateneingaben, welche über die Superpixelalgorithmen SLIC und Quickshift (QS) anhand der in Kapitel 6.2.1 vorgestellten Parameterwerte gewonnen wurden. Für SGCNN wurde dabei aufgrund der reduzierten Datenmenge die Filtergröße im Vergleich zum Vortest gedrosselt und besitzt im Vergleich zu den GCN- und EGCNN-Netzarchitekturen nicht mehrere, aufeinanderfolgende Faltungsschichten.
Es zeigt sich, dass alle Netzansätze auf irregulären Daten garnicht oder nur knapp an die Ergebnisse auf regulären Gittereingaben heranreichen. Insbesondere offenbart der räumliche Faltungsansatz hier seine Schwächen, denn obwohl die Trainingsmenge erfolgreich gelernt wird, so lassen sich unbekannte Eingaben aufgrund der Anordnung zu eindimensionalen Knoten- sowie Nachbarschaftsmengen für irreguläre Graphen nur schwer bis garnicht generalisieren. Für alle Ansätze zeigt sich desweiteren die Quickshift-Segmentierung aufgrund ihrer überlegenden Segmentierung auf den Bildern des MNIST Datensatz als die bessere Wahl (vgl. Abbildung 6.2). Insbesondere erreicht der entwickelte spektrale Faltungsoperator über B-Spline-Kurven (EGCNN) auch hier die eindeutig besten Resultate.

CIFAR-10. Die Klassifikation auf dem CIFAR-10 Datensatz ist im Vergleich zu MNIST eine anspruchsvollere Aufgabe, die insbesondere eine längere Trainingsdauer über weitaus mehr Epochen in Anspruch nimmt. Wichtige Maßnahmen für qualitativ hochwertige Ergebnisse sind dabei insbesondere die in Kapitel 6.2.4 vorgestellten Augmentierungschritte der Eingabedaten [1]. Es hat sich auch in der Evaluierung bewahrheitet, dass eine Augmentierung auf den Graphen schwächere Resultate als

Ansatz	**W**	Architektur	Genauigkeit [%]	
			SLIC	QS
SGCNN	9	C64-P2-C128-P2-C256-P2-C512-Avg-FC256-FC128	66.814	67.748
GCN	1	C64-C64-P2-C128-P2-C256-P2-C512-Avg-FC256-FC128	54.497	53.606
EGCNN	9	C64-C64-P2-C128-P2-C256-P2-C512-Avg-FC256-FC128	74.218	75.230
klassisch	3×3	C64-C64-P4-C128-P4-C256-P4-FC256-FC128	82.061	

Tabelle 6.9: Testgenauigkeiten eines Trainings auf den durch SLIC und Quickshift zur Laufzeit generierten Graphrepräsentationen des CIFAR-10 Datensatz nach 80 Epochen im Vergleich zu einer ähnlichen klassischen Netzarchitektur auf den Bildrohdaten.

eine Augmentierung auf den Rohdaten des Bildes liefert und ein Genauigkeitsverlust von bis zu 5% nach sich zieht. Für alle Lernansätze wurden deshalb die Eingaben des CIFAR-10 Datensatz zur Laufzeit augmentiert und erst anschließend in eine Graphrepräsentation transformiert. Die Lernrate wurde dabei mit 0.001 initialisiert und alle 6 Epochen exponentiell um 0.96 verringert. In den vollverbundenen Schichten außer der Ausgabeschicht besitzen die Gewichte bezüglich der L2-Regularisierung eine Dämpfungskonstante von 0.004. Vor der Ausgabeschicht sorgt die zusätzliche Anwendung eines Dropouts von 0.5 ein Overfitting der Netze weiter zu vermeiden. Tabelle 6.9 fasst die Netzarchitekturen der einzelnen Ansätze und ihre erreichten Genauigkeiten nach rund 80 Epochen zusammen. Alle spektralen Graphansätze auf den aus SLIC und Quickshift generierten Graphrepräsentationen können nicht mit einer ähnlichen klassischen Netzarchitektur auf den Bildrohdaten konkurrieren. Wohingegen jedoch die vorhandenen Ansätze SGCNN und GCN eine maximale Genauigkeit von rund 67% erreichen, zeigt sich auch hier eine signfikante Verbesserung durch den neu entwickelten spektralen Faltungsoperator EGCNN. Er erreicht mit 75% mittels der Quickshift-Segmentierung das beste Resultat. Hochoptimierte Ansätze, bei denen noch weitaus mehr Augmentierungsschritte zum Einsatz kommen, erreichen auf CIFAR-10 jedoch bereits mehr als 90% und so ist nicht auszuschließen, dass auch für EGCNN weiterhin Potential nach oben besteht [16].

Abbildung 6.5 illustriert den Genauigkeitsverlauf des CIFAR-10 Trainings- und Validierungsdatensatzes in Abhängigkeit der vergangenen Epochen. Wohingegen der klassische Ansatz recht schnell nach ungefähr 14 Epochen seine Maximalgenauigkeit erreicht und den Rest des Trainings damit verbringt, seine Trainingsdaten „überanzupassen“, ist zu beobachten, dass die Graphansätze weitaus langsamer und kontinuierlicher lernen.

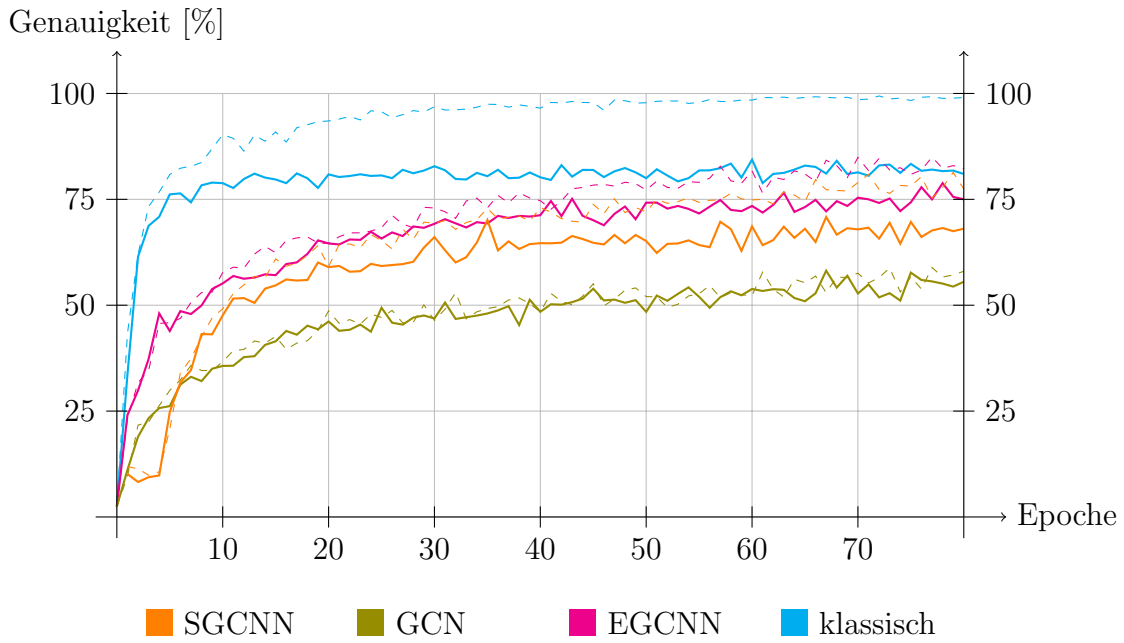


Abbildung 6.5: Genauigkeitsverlauf eines Trainings auf dem CIFAR-10 Datensatz der spektralen Ansätze auf Graphen, welche über Quickshift generiert wurden, sowie dem klassischen Ansatz auf den Bildrohdaten. Die gestrichelten Linien zeigen die Genauigkeiten des Trainingsdatensatzes in Abhängigkeit der Epochen, wohingegen die durchgezogenen Linien sich auf den Validierungsdatensatz beziehen.

PASCAL VOC. Für die Evaluierung der Ergebnisse auf dem PASCAL VOC Datensatz wurde zum Vergleich die SqueezeNet-Architektur in Version 1.1 auf den Bildrohdaten zu Rate gezogen (vgl. [24]). Die Architektur des SqueezeNets gilt aufgrund bewährter Ergebnisse auf dem ImageNet Datensatz bei einer gleichzeitig extrem geringen Anzahl zu trainierender Parameter als „State-of-the-Art“ in der Klassifizerung von Bildern mit recht hoher Auflösung [24]. Zur Verwendung wurden die Bilder dafür einheitlich auf die Größe 224×224 skaliert und mit schwarzen Bereichen an den horizontalen oder vertikalen Rändern je nach Ursprungsauflösung befüllt. Für das Lernen auf Graphen ist diese Sklarierung der Bildeingabedaten nicht von Nöten. Für die Netzarchitektur wurde dabei insbesondere im Vergleich zu vorherigen Testläufen auf die Verwendung mehrmaliger hintereinanderfolgender Faltungsschichten verzichtet. Die Superpixel, die aus den Bildern von PASCAL VOC generiert werden, sind bereits ausreichend groß und eine einzige Faltung auf diesen deckt bereits einen großen Bereich des Bildes ab. Dabei findet das Training ausschließlich aus Laufzeit technischen Gründen auf vorverarbeiten Daten statt und erlaubt damit lediglich eine Augmentierung der Graphen zur Laufzeit (vgl. Kapitel 6.4). Es finden sich weiterhin Anwendungen der Dropout-Technik und der L2-Regularisierung in den vollverbun-

Ansatz	**W**	Architektur	Genauigkeit [%]	
			SLIC	QS
EGCNN	9	C64-P2-C128-P2-C256-P2-C512-Avg-FC256-FC128	54.473	54.516
klassisch	—	SqueezeNet 1.1	54.731	

Tabelle 6.10: Testgenauigkeiten eines Trainings auf den durch SLIC und Quickshift vorab generierten Graphrepräsentationen des PASCAL VOC Datensatz nach 70 Epochen im Vergleich zu der Verwendung des SqueezeNets in Version 1.1 auf den Bildrohdaten [24].

denen Schichten bei einer Lernrate von 0.0001 für das SqueezeNet und 0.001 bei den Graphansätzen.

Tabelle 6.10 stellt die Ergebnisse des SqueezeNets mit den Ergebnissen auf irregulären Dateneingaben bei Benutzung des EGCNNs gegenüber. Es zeigt sich, dass auf PASCAL VOC die Graphansätze durchaus mit klassischen Lösungen auf diesem Gebiet mithalten können. Sie beweisen sich als lediglich marginal schwächer gegenüber der Benutzung einer ausgereiften Architektur wie dem SqueezeNet. Die allgemein relativ schlechten Ergebnisse im Vergleich zu MNIST und CIFAR-10 sind damit größtenteils dem Datensatz geschuldet, da dieser eine weitaus geringere Trainingsmenge sowie keine einander ausschließenden Bildklassen besitzt und seine Verwendung eher in der Lösung zu Problemen wie der Objekterkennung oder Bildsegmentierung sieht.

6.4 Laufzeitanalyse

Die Vorverarbeitung der Eingabedaten sowie das Training der neuronalen Netze erfolgte auf einem 3.5 GHz 6-Core Intel Xeon E5, 16GB DDR3 Arbeitsspeicher und einer Solid-State-Drive. Die in dieser Arbeit präsentierten Laufzeiten wurden ausschließlich über Berechnungen auf dem Prozessor gewonnen. Bei der Verwendung einer Grafikkarte stellen sich dabei Laufzeitgewinne in Bereichen, die auf die diese ausgelagert werden können, von $300 - 400\%$ ein.

Vorverarbeitung. Die Vorverarbeitung der Eingabedaten in eine geeignete Graphrepräsentation, das Berechnen der Merkmalsmatrix sowie zusätzliche Aufwände wie die Vergröberung des Graphen oder die Bestimmung der Receptive-Fields seiner Knoten, welche für die Eingabe in ein neuronales Netz notwendig sind, trägt einen entscheidenen Anteil an der Ausführungsgeschwindigkeit des Trainings und der Auswertung eines neuronalen Netzes bei. Graphen sollen dabei bestmöglichst

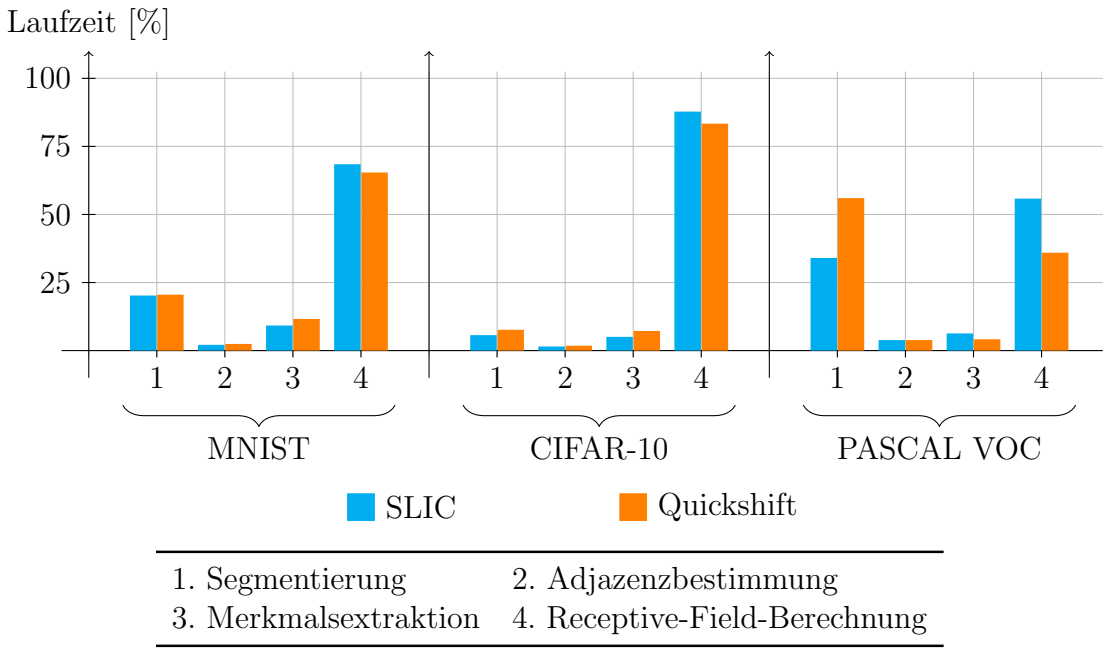


Abbildung 6.6: Laufzeitverteilung der einzelnen räumlichen Vorverarbeitungsschritte für alle Datensätze und alle Superpixelalgorithmen. Die Berechnung der Receptive-Fields umfasst je 25 Knoten bei einer Knotenauswahl mit Schrittweite 2. Sie zeigt sich fast in allen Datensätzen als die dominante Berechnungsaufgabe.

zur Laufzeit eines Trainings generiert werden können und möglichst die Ausführung des eigentlichen Trainings garnicht bis kaum beeinflussen.

Abbildung 6.6 illustriert die relative Laufzeitverteilung der einzelnen räumlichen Vorverarbeitungsschritte bezüglich aller Datensätze und Superpixelalgorithmen. Dabei stellt sich die Berechnung der eindeutigen Receptive-Fields einer groß gewählten Kotenauswahl als Flaschenhals in (fast) allen Datensätzen heraus, die die meiste Zeit in Anspruch nimmt. Lediglich auf großen Bildern, wie sie in PASCAL VOC enthalten sind, zeigt sich der Aufwand der Segmentierung als ebenso kostspielig.

Abbildung 6.7 veranschaulicht auf ähnliche Weise die Laufzeitverteilung der spektralen Vorverarbeitungsschritte. Anstelle der Receptive-Field-Berechnung entsteht hier ein Mehraufwand durch die Berechnung der Graphvergröberung. Dieser Aufwand zeigt sich jedoch auf Bildern aus PASCAL VOC als zu vernachlässigen, da die Laufzeit bei großen Bildern (fast) rein von der Laufzeit der Superpixelalgorithmen abhängig ist.

Tabelle 6.11 stellt die absoluten Aufwände zur räumlichen sowie spektralen Vorverarbeitung eines Bildes aus allen Datensätze und Superpixelalgorithmen gegenüber. Wohingegen sich für Bilder mit kleinen Auflösungen aus den Datensätzen MNIST

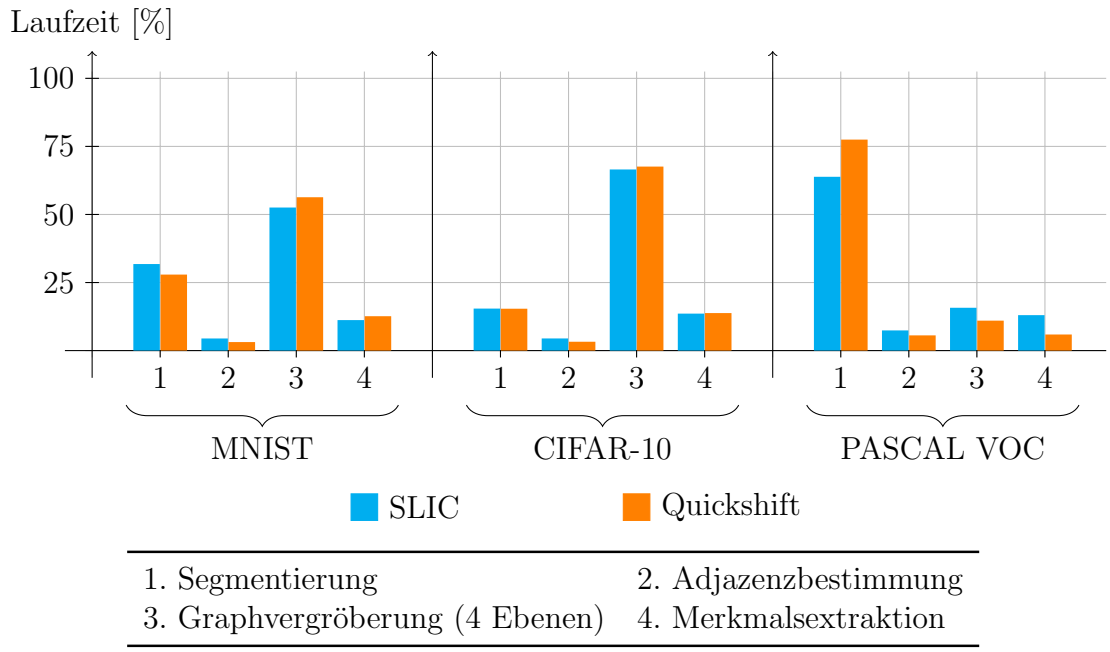


Abbildung 6.7: Laufzeitverteilung der einzelnen spektralen Vorverarbeitungsschritte für alle Datensätze und Superpixelalgorithmen. Wohingegen für MNIST und CIFAR-10 die Graphvergröberung die längste Berechnungsdauer in Anspruch nimmt, zeigt sich dagegegen für größere Datensätze wie PASCAL VOC die Segmentierung als Flaschenhals der Vorverarbeitung.

	Räumlich [ms]		Spektral [ms]	
	SLIC	QS	SLIC	QS
MNIST	30.59	24.00	20.32	19.40
CIFAR-10	72.74	43.85	25.34	23.18
PASCAL VOC	713.40	1080.78	372.79	782.56

Tabelle 6.11: Laufzeiten der räumlichen sowie spektralen Vorverarbeitung für ein Bild der jeweiligen Datensätze und Superpixelalgorithmen basierend auf den Laufzeiten aus Abbildung 6.6 und 6.7. Die räumliche Vorverarbeitung ist aufgrund der einzelnen Receptive-Field-Berechnungen deutlich ineffizienter.

und CIFAR-10 der Aufwand zur Vorverarbeitung in Grenzen hält, zeigen sich für ein Bild aus PASCAL VOC die Kosten zur Vorverarbeitung als nicht zu unterschätzen. Eine Vorverarbeitung zur Laufzeit ist auf diesem Datensatz daher (zurzeit) nicht zu empfehlen. Weiterhin sei anzumerken, dass die spektralen Vorverarbeitungsschritte in jedem Versuchsaufbau entschieden schneller als die räumlichen berechnet werden können.

Datensatz	Superpixel	Verfahren	Laufzeit [s]			
			vor Beginn	zur Laufzeit		
				Vorverarbeitung	Training	$\sum$
MNIST	SLIC	RCNN	0.04	—		
MNIST	SLIC	SGCNN		—		
MNIST	SLIC	GCN	0.20	—		
MNIST	SLIC	EGCNN	0.61	—		
MNIST	QS	RCNN	0.04	—		
MNIST	QS	SGCNN	0.37	—		
MNIST	QS	GCN	0.21	—		
MNIST	QS	EGCNN	0.63	—		
CIFAR-10	SLIC	SGCNN	—			
CIFAR-10	SLIC	GCN	—	1.24	0.39	1.63
CIFAR-10	SLIC	EGCNN	—	0.44	2.00	2.44
CIFAR-10	QS	SGCNN	—	0.43	1.95	2.38
CIFAR-10	QS	GCN	—	1.09	0.36	1.45
CIFAR-10	QS	EGCNN	—			
PASCAL VOC	SLIC	EGCNN		23.05	4.17	27.22
PASCAL VOC	QS	EGCNN	5.28	51.32	5.46	56.78
MNIST	—	klassisch		—		
CIFAR-10	—	klassisch	0.49	—		
PASCAL VOC	—	klassisch		—		

Tabelle 6.12: Vergleich der Laufzeiten des Trainings eines Batches der Größe 64 für alle vorgestellten Netzarchitekturen.

Training. Tabelle 6.12 zeigt die im Durschnitt ermittelten Laufzeiten des Trainings eines Batches der Größe 64 für alle vorgestellten Netzarchitekturen. Die Eingabedaten wurden für MNIST bzw. PASCAL VOC aufgrund nicht benötigter Augmentierung bzw. der aufwändigen Berechnungen vorab gespeichert, wohingegen die Grapheingabedaten für CIFAR-10 zur Laufzeit generiert wurden. Es zeigt sich, dass das Training auf Graphen im Allgemeinen deutlich langsamer als im Vergleich zu ausgereiften Bildimplementierungen ist. Lediglich der Ansatz des räumlichen Lernens kann aufgrund seiner Verwendung der klassischen Faltungsoperation und bei einer Vorverarbeitung vor Beginn des Trainings schnelle Laufzeiten garantieren.

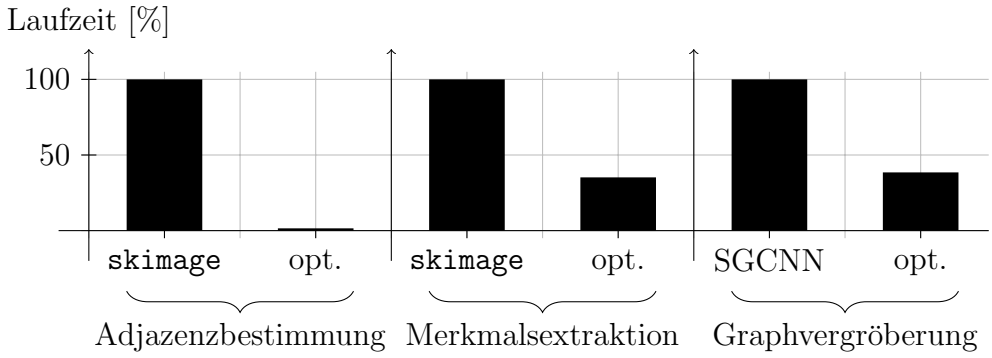
Zwischen den spektralen Faltungsansätzen zeigt sich das GCN aufgrund seiner geringen Berechnungskomplexität als sehr effizient, leidet jedoch wie in Kapitel 6.3 beschrieben an mangelnder Genauigkeit. Das EGCNN weist unter allen Ansätzen die höchste Berechnungskomplexität auf. Dies ist im Vergleich mit dem SGCNN insoweit zu erklären, dass dessen Netzarchitektur in der Regel tiefer ist und mehrere Faltungsschichten hintereinander verlangt, um an Informationen nicht direkt

benachbarter Knoten zu gelangen. Das SGCNN erfordert dies nicht, da dessen Filtergröße in direktem Zusammenhang zur Größe der Faltung steht.
Es ist weiter bei einer Vorverarbeitung während des Trainings zu beobachten, dass die Aufwände für die Vorverarbeitung umso höher werden, umso schneller die eigentliche Lernprozedur vonstatten geht. So zeigt sich beispielsweise bei CIFAR-10, dass das EGCNN und SGCNN in der Regel lediglich ≈ 0.4s auf die Vorverarbeitung warten müssen, wohingegen bei der effizienteren Faltung des GCNs das Training pro Batch ≈ 1.1s pausiert. Damit wird trotz Multithreading die Vorverarbeitung zur Laufzeit zum Flaschenhals des Trainings.

Vergleich bezüglich vorhandener Implementierungen. Obwohl die Vorverarbeitung das Training eines neuronalen Netzes zur Laufzeit behindern kann, so konnten bereits viele Vorverarbeitungsschritte im Laufe dieser Arbeit effizienter als ihre vorhandenen, quelloffenenen Implementierungen gestaltet werden. So besitzt beispielsweise `skimage` eine Implementierung zur Adjazenzbestimmung einer Segmentierungsmaske und zur Extraktion der meisten in dieser Arbeit vorgestellten Merkmale. Beide Verfahren konnten mit deutlichem Laufzeitgewinnn optimiert werden. Der frei zugängliche Quelltext des spektralen Faltungsoperators (SGCNN) besitzt ebenso eine Implementierung zur Vergröberung eines Graphen. Dieser konnte, trotz der Berücksichtigung der Knotenpositionen und Generierung zweier Adjazenzmatrizen $\mathbf{A}_{\text{dist}}$ und $\mathbf{A}_{\text{rad}}$, fast dreimal so schnell implementiert werden. Abbildung 6.8 fasst die beschriebenen Laufzeitgewinne separat zusammen.

6.5 Diskussion

Das räumliche Lernen hat sich in Tests auf irregulären Grapheingaben als unzureichend herausgestellt. Das Verfahren scheint zwar aus Laufzeit technischen Gründen aufgrund der Benutzung der klassischen Faltungsoperation als interessant, jedoch lassen sich Grapheingaben nicht ohne Weiteres in eine reguläre Eingabe überführen. Ebenso erweist sich die Vorabberechnung der Receptive-Fields als unzufriedenstellend. Wohingegen diese auf Bildrohdaten von der Faltungsoperation bestimmt und berechnet werden, müssen sie beim räumlichen Lernen bereits für die Eingabe in ein Netz gegeben sein. Damit erlaubt das räumliche Lernen lediglich eine Faltungsschicht in einem neuronalen Netz. Die Anwendung der klassischen Faltungsoperation wird zudem etwas missentfremdet, denn die Faltung entspricht aufgrund der Form des verschiebbaren Fensters einer eindimensionalen Faltung auf vorab berechneten



Vorverarbeitungsschritt	vorhanden [ms]	optimiert [ms]	Gewinn [%]
Adjazenzbestimmung	866	13	6615
Merkmalsextraktion	144	50	288
Graphvergröberung (4 Ebenen)	87	33	264

Abbildung 6.8: Vergleich der Laufzeiten verschiedener Vorverarbeitungsschritte bezüglich des Lernens auf Graphen zwischen bereits vorhandenen, quelloffenen Implementierungen (links) und ihren jeweiligen entwickelten optimierten Versionen (rechts) am Beispiel eines Bildes aus PASCAL VOC: (1) Adjazenzbestimmung einer Segmentierungsmaske, (2) Merkmalsextraktion und (3) Graphvergröberung inklusive der entsprechenden Anordung der Knoten zu binären Bäumen. In allen Bereichen konnten deutliche Laufzeitgewinne erreicht werden.

zusammenhängenden Daten mit festen Grenzen.

Der Ansatz des spektralen Lernens erweist sich sowohl in der Theorie als auch in der Praxis als das zukunftsorientiertere Verfahren. Dabei wird nicht versucht, die Grapheingaben in eine reguläre Form zu transformieren, sondern den ursprünglich nur auf regulären Gittern definierten Faltungsoperator für weiterführende Eingaben zu generalisieren. Anhand der Evaluierung lässt sich dieses Verfahren im Vergleich als den klaren Gewinner auszeichnen. Bisherige Unternehmungen auf diesem Gebiet zeigen ebenso für andere Anwendungsgebiete beachtliche Ergebnisse (vgl. [8, 26]). Probleme ergeben sich jedoch dennoch aufgrund der in der Praxis nicht tolerierbaren hohen Laufzeiten spektraler neuronaler Netze. Kapitel 7 versucht diesbezüglich Lösungsansätze für weiterführende Arbeiten aufzuzeigen.

In dieser Arbeit wurde weiterhin versucht, den spektralen Faltungsoperator auf Graphen im zweidimensional euklidischen Raum zu erweitern, sodass den Kantenausrichtungen der Graphen in der Faltung eine Bedeutung zugeschrieben werden kann. In allen Testdurchläufen erwies sich dieser Ansatz als geglückt und es konnten signifikante Verbesserungen im Vergleich zu den bisherigen Ansätzen auf diesem Gebiet gewonnen werden. In der Evaluierung wurde aus implementierungstechnischen

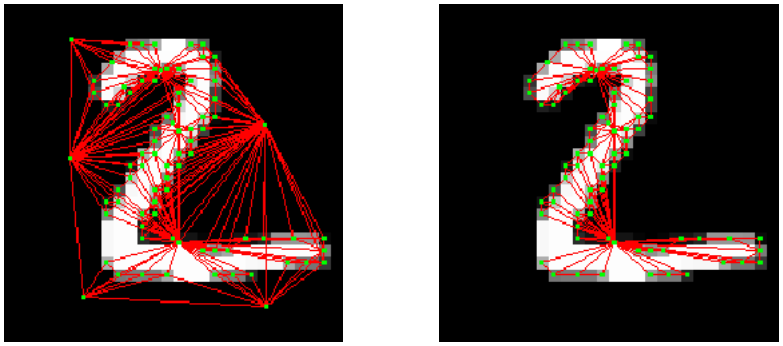
Gründen dabei ein maximaler Grad der B-Spline-Funktionen von Eins verwendet. Es erscheint jedoch vorstellbar die Filtergröße bei größerer lokaler Kontrollierbarkeit zu reduzieren und die Gefahr des Overfittings damit aufgrund der kleineren Anzahl an Trainingsparametern weiter einzuschränken.
Die Problemstellung in dieser Arbeit, die Klassifizierung aus Bildern gewonnener Graphrepräsentationen, erfordert aufgrund allgemein schwächerer Ergebnisse im Vergleich zu den klassischen Lösungen auf diesem Gebiet weitere Nachforschungen bzw. Optimierungen. Es wurde jedoch ein spektraler Ansatz entwickelt, der die regulären Grenzen des klassischen Faltungsoperators sprengt und ein Tor zu einer Vielzahl weiterer Anwendungsgebiete öffnet.

7 Ausblick

Die vorgestellten und entwickelten Ansätze zum Lernen auf Graphen im zweidimensional euklidischen Raum wurden in dieser Arbeit über einer Klassizierung auf Graphrepräsentationen von Bildern evaluiert. Sie lassen sich jedoch auch mühelos auf andere Probleme wie zum Beispiel der Objekterkennung oder der Segmentierung anwenden. Dabei sind ebenso völlig andere als die in dieser Arbeit präsentierten Anwendungsgebiete denkbar, denn die einzige Einschränkung der Graphen für die Benutzung der entwickelten Faltungsoperatoren ist die eindeutige oder zumindest relative Positionierung seiner Knoten zueinander im Raum. So lassen sich solche Graphen beispielsweise über Karten oder Straßennetze gewinnen, deren Eingabe in die vorgestellten Netzarchitekturen vielversprechende Möglichkeiten offenbaren.
Zum Abschluss dieser Arbeit wird ein Ausblick zu denkbaren Erweiterungen vorgestellt, für deren Erforschung an dieser Stelle der Platz fehlt.

Erweiterung auf den dreidimensionalen Raum. Alle Faltungsansätze in dieser Arbeit wurden lediglich auf Graphen im zweidimensionalen Raum betrachtet. Dabei ist es aber ebenso vorstellbar, diese Ansätze auf den dreidimensionalen Raum auszuweiten. In diesem Fall besitzt ein Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, p)$ eine Positionsfunktion $p\colon \mathcal{V} \to \mathbb{R}^3$, die den Knoten jeweils drei Koordinaten zuordnet. Die implizit gegebene Winkelfunktion seiner Kanten $\varphi\colon \mathcal{V} \times \mathcal{V} \to [0, 2\pi]^2$ bildet damit folglich auf zwei Winkel ab. Insbesondere der spektrale Faltungsoperator, der in Kapitel 5.4.2 mit Hilfe einer polynomiellen Approximation über B-Spline-Kurven erweitert wurde, erlernt nun eine Repräsentation einer geschlossenen Fläche anstatt einer Kurve, die mittels der *Tensorproduktkonstruktion* über zwei Basisfunktionen konstruiert werden kann [2]. Eine Implementierung bzw. Erweiterung dieser Ansätze auf den dreidimensionalen Raum erlaubt damit insbesondere das Lernen auf Polygonnetzen und folglich auch auf dreidimensionalen Objekten.

Entfernung irrelevanter Knoten. In den Graphrepräsentationen von Bildern können bei der Benutzung der Quickshift-Segmentierung Knoten vereinzelnd einen extrem hohen Knotengrad aufweisen (vgl. zum Beispiel Tabelle 6.1). Das ist insbeson-



(a) Ursprungsgraph (b) Knotenentfernung

Abbildung 7.1: Illustration der Entfernung irrelevanter Knoten eines Graphen, der über Quickshift eines Bildes aus dem MNIST Datensatz gewonnen wurde. (a) zeigt den Ursprungsgraphen der Segmentierung, wohingegen der Graph in (b) irrelevante Knoten, die den Hintergrund beschreiben, entfernt.

dere dann der Fall, wenn Knoten eine große einheitliche Fläche des Bildes abdecken (zum Beispiel einen einfarbigen Hintergrund) und folglich an dessen Segmenträndern viele benachbarte Pixel anliegen. Eine Faltung auf diesen Knoten besitzt so gut wie keine lokale Aussage, sodass dessen Nutzen für ein neuronales Netzes fraglich oder gar hinderlich ist. Es erscheint sinnvoll, diese Knoten in einem weiteren Vorverarbeitungsschritt über einer zuvor definierten Strategie zu entfernen. Abbildung 7.1 illustriert eine mögliche irrelevante Knotenentfernung anhand eines Bildes aus dem MNIST-Datensatz. Dabei lassen sich zwischen datensatzspezifischen und -unspezifischen Strategien unterscheiden. Auf dem MNIST Datensatz können zum Beispiel alle Knoten entfernt werden, die einen schwarzen Farbwert oder eine (relativ) große Fläche aufweisen. Eine allgemeinere Vorgehensweise ist die Knotenentfernung basierend auf einem (relativen) Grenzwert seiner Knotengrade. Bei der bloßen Entfernung von Knoten aus einem Graphen läuft man jedoch Gefahr, dass dieser nun möglicherweise isolierte Knoten bzw. mehrere Zusammenhangskomponenten besitzt. Insbesondere die Poolingoperation sowie die Faltung basierend auf den GCNs erfordern jedoch zusammenhängende Graphen. Ein Test mit Hilfe des zweiten Eigenwerts der Laplace-Matrix zeigt, dass die Knotenentfernung basierend auf ihren Farbwerten für Graphen, die über Quickshift aus dem MNIST-Datensatz generiert wurden, in 99.9% der Fällen einen zusammenhängenden Graphen erhält, wohingegen für eine Knotenentfernung auf Basis ihrer Knotengrade dies nur noch für 45.4% der Graphen der Fall ist. Es lassen sich jedoch die beiden nächsten zueinander liegenden Knoten unterschiedlicher Zusammenhangskomponenten über eine Kante verbinden, bis der Graph wieder zusammenhängend ist.

Spatial Pyramid Pooling. Eine Alternative zu der verwendeten Durchschnittsbildung zwischen Faltungs- und vollverbundener Schicht, die für die Implementierung der spektralen Netzarchitektur auf Graphen mit dynamischer Knotengröße benutzt wurde (vgl. Kapitel 5.6), ist das *Spatial Pyramid Pooling (SPP)* [18]. Die SPP-Schicht erhält entgegen der Durchschnittsbildung räumliche Informationen, indem über einer Menge räumlicher Gruppierungen unterschiedlicher Auflösungen *gepoolt* wird, sodass sich als Endresultat ein Vektor fester Größe ergibt (vgl. [18]). Ein modifizierter Ansatz dieses Algorithmus auf Graphen erfordert demnach die Vergröberung der Graphen auf deren gemeinsame kleinste Größe (Vielfaches von zwei) mit möglicherweise unterschiedlich vielen Ebenen. Damit ist die Verwendung einer SPP-Schicht mit weitaus mehr Berechnungsaufwand verbunden, dessen Qualität aufgrund der Translationsinvarianz von Adjazenzmatrizen erst zu evaluieren ist.
Eine weitere Alternative ist die Verwendung eines *Attentation*-Mechanismus, der beispielsweise in *Recurrent Neural Networks (RNNs)* zum Einsatz kommt [31].

Effiziente GPU-Implementierung. Die Laufzeiten der vorgestellten Faltungsoperatoren können derzeit, obgleich einer kleineren Eingabegröße, nicht mit den Laufzeiten der klassichen Faltung auf Bildern konkurrieren. Der klassische Faltungsoperator besitzt eine für die GPU höchst optimierte batchweise Implementierung, wohingegen die Graphimplementierung aufgrund der Limitierung dünnbesetzter Tensoren auf genau zwei Dimensionen dies nicht leisten kann. Für die Verarbeitung eines Batches muss folglich über jeden Graphen einzeln iteriert werden, sodass die Vorteile der Nutzung einer GPU bezüglich ihrer Parallelisierungsmechanismen verloren gehen. Eine Möglichkeit, diese Schwäche zu umgehen, ist die batchweise Darstellung der Adjazenzmatrizen $\mathcal{A} \coloneqq \{\mathbf{A}_m\}_{m=1}^{M}$ über genau eine dünnbesetzte Matrix, sodass die Adjazenzmatrizen diagonalbasiert angeordnet sind, d.h.

$$\mathcal{A} \coloneqq \begin{bmatrix} \mathbf{A}_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{A}_M \end{bmatrix},$$

und folglich einheitlich verarbeitet werden können. Die Optimierung dieses Ansatzes bleibt ein interessantes Forschungsgebiet für weiterführende Arbeiten.

Glossar

T Tschebyschow-Polynom. 49–51

ℓ Knotenfärbung $\ell\colon \mathcal{V} \to \mathcal{C} \subseteq \mathbb{R}$ eines Graphen $\mathcal{G}$. 29, 31, 32, 34, 36–38

γ Lernrate eines neuronalen Netzes. 10, 80

$\hat{y}$ erwartete Ausgabefunktion eines neuronalen Netzes. 10, 79

$\lambda_{\max}$ Größter Eigenwert eines Eigenwertproblems $\mathbf{Mu} = \lambda\mathbf{u}$. 42, 45, 49–51

λ Eigenwert eines Eigenwertproblems $\mathbf{Mu} = \lambda\mathbf{u}$. 27, 28, 30, 42, 45–48, 73, 74, 77

$\mathbf{A}_{\text{dist}}$ Distanzadjazenzmatrix $\mathbf{A}_{\text{dist}} \in [0,1]^{N\times N}$ eines Graphen $\mathcal{G}$ im zweidimensional euklidischen Raum. 15, 16, 20, 21, 55, 56, 58, 65, 67, 92

$\mathbf{A}_{\text{rad}}$ Winkeladjazenzmatrix $\mathbf{A}_{\text{rad}} \in [0,2\pi]^{N\times N}$ eines Graphen $\mathcal{G}$ im zweidimensional euklidischen Raum. 16, 55, 58–60, 65, 67, 80, 81, 92

$\tilde{\mathbf{A}}_{\text{dist}}$ transformierte Distanzadjazenzmatrix $\tilde{\mathbf{A}}_{\text{dist}} \coloneqq \mathbf{A}_{\text{dist}} + \mathbf{I}$. 56, 59, 60

$\tilde{\mathbf{D}}_{\text{dist}}$ transformierte Gradmatrix $\tilde{\mathbf{D}}_{\text{dist}} \coloneqq \tilde{\mathbf{D}}_{\text{dist}} + \mathbf{I}$ eines Graphen $\mathcal{G}$ im zweidimensionalen euklidischen Raum. 56, 57, 59, 60

$\mathbb{N}$ Menge der natürlichen Zahlen $\{0,1,2,\dots,\}$. 5–10, 12, 18, 19, 21, 22, 25, 30–34, 36, 42, 45, 55, 58, 62, 72

$\mathbb{R}_+$ Menge der positiven reellen Zahlen. 7, 10, 30, 36, 42, 59, 61

$\mathbb{R}$ Menge der reellen Zahlen. 5–10, 12, 15–18, 20–23, 29, 30, 32, 34–39, 42–54, 56–60, 64, 65, 71, 72, 78, 80–82, 95

$\mathcal{C}$ Menge der Farben $\mathcal{C} \subseteq \mathbb{R}$ einer Knotenfärbung ℓ eines Graphen $\mathcal{G}$. 29, 32, 34

$\mathcal{E}$ Kantenmenge $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ eines Graphen $\mathcal{G}$. 7, 8, 15–20, 30, 31, 33–36, 38, 49, 50, 52, 53, 57, 59, 62, 65, 95

$\mathcal{G}$ Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ mit Knotenmenge $\mathcal{V}$ und Kantenmenge $\mathcal{E}$. 7, 8, 15–20, 25, 29–31, 34–36, 38, 43–46, 51, 52, 55, 56, 58, 59, 62–66, 95

$\mathcal{N}$ Nachbarschaftsfunktion $\mathcal{N}_K(v_i) \coloneqq \{v_j \mid s(v_i, v_j) \leq K\}$. 7, 8, 30, 32–35, 37–39, 43, 44, 49, 52, 53, 55, 63

$\mathcal{O}$ O-Notation. 23, 48, 50, 52, 57, 60, 64

$\mathcal{S}$ Menge von N Superpixeln $\mathcal{S} \coloneqq \{\mathcal{S}_n\}_{n=1}^{N}$ mit $\mathcal{S}_n \subset W \times H$. 18, 25

$\mathcal{V}$ Knotenmenge $\mathcal{V} \coloneqq \{v_n\}_{n=1}^{N}$ eines Graphen $\mathcal{G}$. 7, 8, 15–20, 29–34, 36–39, 43, 46, 52, 55, 57, 62–65, 95, 99

NCut Normalized-Cut. 63

N Basisfunktionen einer B-Spline-Kurve. 58–61

ReLU Aktivierungsfunktion $\mathrm{ReLU}(\cdot) \coloneqq \max(\cdot, 0)$ eines neuronalen Netzes. 10, 52, 65, 79

conv2d klassische Faltungsoperation auf einem regulären Gitter. 12, 31, 35, 38, 56, 57

deg Gradfunktion $\deg(v) \coloneqq |\mathcal{N}(v)|$ auf den Knoten eines Graphen $\mathcal{G}$. 7, 30, 72, 74, 76

diag Diagonalfunktion $\mathrm{diag} \to \mathbb{R}^N \to \mathbb{R}^{N \times N}$ mit $\mathrm{diag}(\mathbf{v})_{ii} = \mathbf{v}_i$. 6, 7, 42

$\odot$ elementweises Hadamard-Produkt. 47, 48, 59, 60

$\perp$ Orthogonalität zweier Vektoren, d.h. $\langle \mathbf{v}, \mathbf{w} \rangle = 0$. 5, 42

σ Aktivierungsfunktion eines neuronalen Netzes. 9, 10, 12, 52, 53

φ Winkelfunktion$\varphi\colon \mathcal{V} \times \mathcal{V} \to [0, 2\pi]$ auf den Kanten eines Graphen $\mathcal{G}$. 16, 37, 95

ξ Standardabweichung der Gaußfunktion. 15, 16, 20, 23, 24, 54, 72, 74, 76, 79, 82

b B-Spline-Kurve. 58, 59

d gewichtete Gradfunktion $d(v_i) \coloneqq \sum_{j=1}^{N} \mathbf{A}_{ij}$ auf den Knoten eines Graphen $\mathcal{G}$. 7, 30, 43, 63

m Massefunktion $m\colon \mathcal{V} \to \mathbb{R}$ auf den Knoten des Graphen $\mathcal{G}$. 19, 65, 66

p Positionsfunktion $p\colon \mathcal{V} \to \mathbb{R}^2$ auf den Knoten des Graphen $\mathcal{G}$. 15–20, 36–38, 65, 66, 95

s kürzeste Pfaddistanz $s\colon \mathcal{V} \times \mathcal{V} \to \mathbb{N} \cup \{\infty\}$ zweier Knoten eines Graphen $\mathcal{G}$. 7, 8, 30, 33, 34, 45

v Knoten $v \in \mathcal{V}$ eines Graphen $\mathcal{G}$. 7, 8, 15–20, 29–38, 43–45, 47–49, 52, 53, 55, 59, 63–66

w Gewichtsfunktion $w\colon \mathcal{V} \times \mathcal{V} \to \mathbb{R}_+$ auf den Kanten eines Graphen $\mathcal{G}$. 7, 15, 16, 43, 44, 53, 63

y Ausgabefunktion eines neuronalen Netzes. 10, 71, 78, 79

$\mathbf{A}$ Adjazenzmatrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ eines Graphen $\mathcal{G}$. 7, 17, 20, 30, 43, 51, 55–57, 97

$\mathbf{B}$ Bild $\mathbf{B} \in \mathbb{R}^{H \times W \times C}$. 6, 7, 12, 15, 17, 18, 21–24, 80

$\mathbf{D}$ Gradmatrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ eines Graphen $\mathcal{G}$ mit $\mathbf{D}_{ii} = d(v_i)$. 7, 43, 51

$\mathbf{F}$ Merkmalsmatrix $\mathbf{F} \in \mathbb{R}^{N \times M}$ auf den Knoten eines Graphen $\mathcal{G}$. 8, 15, 17, 28, 35, 38, 39, 50, 52, 57, 60, 81

$\mathbf{I}$ Einheitsmatrix $\mathbf{I} \in \mathbb{R}^{N \times N}$ mit $\mathbf{I}_{ii} = 1$. 42, 43, 49, 51, 56

$\mathbf{L}$ kombinatorische Laplace-Matrix. 43–45, 47

$\mathbf{M}$ nicht-zentrierte Momente. 25–28, 74, 77

$\mathbf{S}$ Segmentierungsmaske $\mathbf{S} \in \{1, \dots, N\}^{H \times W}$ bezüglich einer Menge von N Superpixeln. 18–20, 22, 25–27

$\mathbf{U}$ Eigenvektormatrix $\mathbf{U} \coloneqq [\mathbf{u}_1, \dots, \mathbf{u}_N] \in \mathbb{R}^{N \times N}$ einer reell symmetrischen Matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$. 42, 46–50

$\mathbf{W}$ Gewichtsmatrix oder Gewichtstensor einer neuronalen Schicht. 9, 10, 12, 50, 52, 53, 56, 57, 60

$\mathbf{\Lambda}$ Diagonalmatrix der Eigenwerte $\{\lambda_n\}_{n=1}^{N}$ einer symmetrischen Matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$. 42, 48, 49

η skaleriungsinvariante Momente. 26, 28, 73

$\mathcal{L}$ kombinatorische oder normalisierte Laplace-Matrix. 43, 45, 46, 48–51

μ translationsinvariante Momente. 26–28, 73

$\tilde{\mathbf{A}}$ transformierte Adjazenzmatrix $\tilde{\mathbf{A}} \coloneqq \mathbf{A} + \mathbf{I}$. 51–53, 56, 57

$\tilde{\mathbf{D}}$ transformierte Diagonalmatrix $\tilde{\mathbf{D}} \coloneqq \mathbf{D} + \mathbf{I}$. 51–54

$\tilde{\mathbf{L}}$ normalisierte Laplace-Matrix $\tilde{\mathbf{L}} \coloneqq \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$. 43–45, 51

$\tilde{\mathbf{\Lambda}}$ skalierte und transformierte Diagonalmatrix der Eigenwerte $\tilde{\mathbf{\Lambda}} \coloneqq 2\Lambda/\lambda_{\max} - \mathbf{I}$. 49–51

$\tilde{\mathcal{L}}$ skalierte und transformierte Laplace-Matrix $\tilde{\mathcal{L}} \coloneqq 2\mathcal{L}/\lambda_{\max} - \mathbf{I}$. 49–51

$\mathbf{b}$ Biasvektor einer neuronalen Schicht. 9, 10, 12

$\mathbf{f}$ Merkmalsvektor $\mathbf{f} \in \mathbb{R}^N$ auf den Knoten eines Graphen $\mathcal{G}$. 8, 43–53, 56, 57, 59, 64, 82

$\mathbf{h}$ (rotationsinvariante) Hu-Momente. 26, 28, 74, 77

$\mathbf{u}$ Eigenvektor zu einem Eigenwert λ eines Eigenwertproblems $\mathbf{M}\mathbf{u} = \lambda\mathbf{u}$. 42, 45–47

CIFAR Canadian Institute for Advanced Research. 73–76, 78, 83, 85–92, 105, 107

CNN Convolutional Neural Network. 11–13, 29, 35, 37, 41, 48, 50, 51, 53, 54, 61, 66, 67, 71

EGCNN spektraler Faltungsoperator auf Graphen im zweidimensionalen euklidischen Raum. 83–88, 91, 92

GCN Graph Convolutional Network. 50–55, 83–87, 91, 92, 96

MNIST Modified National Institute of Standards and Technology. 71–74, 78, 83–85, 88–91, 96, 105, 107

PASCAL VOC Pascal Visual Object Classes. 75–78, 83, 86–91, 93, 105, 107

PCA Hauptkomponentenanalyse. 69, 70, 105

QS Quickshift. 85, 86, 88, 90, 91

RCNN räumlicher Faltungsoperator. 83–85, 91

RNN Recurrent Neural Network. 97

SGCNN spektraler Faltungsoperator über Tschebyschow-Polynome. 83–87, 91–93

SLIC Simple Linear Iterative Clustering. 21–24, 28, 36, 71–77, 82, 85, 86, 88–91

SPP Spatial Pyramid Pooling. 97

SVHN Street View House Numbers. 78

SVM Support Vector Machine. 70

Abbildungsverzeichnis

1.1 Problemstellung 2

2.1 Feedforward-Netz 9
2.2 Faltung innerhalb eines CNNs 12
2.3 Netzarchitektur eines CNNs 13

3.1 Kantengewichtsberechnung über Gaußfunktion 16
3.2 Graphgenerierung aus einer Superpixelrepräsentation 19
3.3 Effiziente Adjazenzbestimmung einer Segmentierungsmaske 20
3.4 SLIC und Quickshift Beispielresultat 23
3.5 Laufzeitverteilung einer Merkmalsextraktion 28

4.1 Kanonische Ordnung 31
4.2 Normalisierung 35
4.3 Normalisierung für Graphen im zweidimensionalen Raum 37
4.4 Räumliche Netzarchitektur auf Graphen 39

5.1 5-Punkte-Stern 44
5.2 Graphrepräsentation eines regulären Gitters 54
5.3 Partitionierung eines Graphknotens 55
5.4 Geschlossene B-Spline-Funktionen 59
5.5 B-Spline-Funktion mit $K = 1$ über Minimum-Maximumfunktion . . . 60
5.6 Graphvergröberung mittels Graclus und Normalized-Cut 62
5.7 Pooling auf Graphen 64
5.8 Spektrale Netzarchitektur auf Graphen 67

6.1 Kumulative Varianzabdeckung einer PCA 70
6.2 MNIST 73
6.3 CIFAR-10 75
6.4 PASCAL VOC 77
6.5 CIFAR-10 Genauigkeitsverlauf über Quickshift 87
6.6 Laufzeitverteilung der räumlichen Vorverarbeitungsschritte 89

6.7 Laufzeitverteilung der spektralen Vorverarbeitungsschritte 90
6.8 Laufzeitvergleich bezüglich anderer Implementierungen 93

7.1 Entfernung irrelevanter Knoten 96

Tabellenverzeichnis

6.1 MNIST Superpixelparameter 72
6.2 MNIST Merkmalsselektion 73
6.3 CIFAR-10 Superpixelparameter 74
6.4 CIFAR-10 Merkmalsselektion 74
6.5 PASCAL VOC Superpixelparameter 76
6.6 PASCAL VOC Merkmalsselektion 77
6.7 Testgenauigkeiten der MNIST-Gitterrepräsentation 84
6.8 Testgenauigkeiten der MNIST Superpixelrepräsentationen 85
6.9 Testgenauigkeiten der CIFAR-10 Superpixelrepräsentationen 86
6.10 Testgenauigkeiten der PASCAL VOC Superpixelrepräsentationen . . 88
6.11 Laufzeiten der räumlichen und spektralen Vorverarbeitung 90
6.12 Vergleich der Trainingslaufzeiten 91

Algorithmenverzeichnis

3.1 SLIC 22

4.1 Knotenauswahl 32
4.2 Nachbarschaftsgruppierung 33
4.3 Normalisierung 34

5.1 Weisfeiler-Lehman 52

Literaturverzeichnis

[1] Abadi, Martin; Agarwal, Ashish; Barham, Paul; Brevdo, Eugene: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. `http://www.tensorflow.org`. 2015

[2] Abramowski, Stephan; Müller, Heinrich: *Geometrisches Modellieren*. B.I. Wissenschaftsverlag, 1991 (Reihe Informatik)

[3] Achanta, Radhakrishna; Shaji, Appu; Smith, Kevin; Lucchi, Aurelien; Fua, Pascal; Susstrunk, Sabine: SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2012), S. 2274–2282

[4] Biggs, Norman L.; ; Lloyd, E. Keith; Wilson, Robin J.: *Graph Theory*. Oxford University Press, 1999

[5] Chung, Fan .R.K.: *Spectral Graph Theory*. American Mathematical Society, 1997

[6] Coates, Adam; Lee, Honglak; Ng, Andrew Y.: An Analysis of Single Layer Networks in Unsupervised Feature Learning. In: *AISTATS* (2011)

[7] de Boor, Carl: *A Practical Guide to Splines*. Springer Verlag, 1978

[8] Defferrard, Michaël; Bresson, Xavier; Vandergheynst, Pierre: Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In: *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016, S. 3844–3852

[9] Dhillon, Inderjit S.; Guan, Yuqiang; Kulis, Brian: Weighted Graph Cuts Without Eigenvectors: A Multilvel Approach. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2007), S. 1944–1957

[10] Douglas, Brendan L.: The Weisfeiler-Lehman Method and Graph Isomorphism Testing. In: *arXiv preprint arXiv:1101.5211* (2011)

[11] Everingham, Mark; Eslami, S.M. Ali; Van Gool, Luc; Williams, Christoper K. I.; Winn, John; Zisserman, Andrew: The Pascal Visual Object Classes Challenge: A Retrospective. In: *International Journal of Computer Vision* (2015), S. 98–136

[12] Felzenszwalb, Pedro F.; Huttenlocher, Daniel P.: Efficient Graph-Based Image Segmentation. In: *International Journal of Computer Vision* (2004), S. 167–181

[13] Fulkerson, Brian; Vedaldi, Andrea; Soatto, Stefano: Class Segmentation and Object Localization with Superpixel Neighborhoods. In: *International Conference on Computer Vision* (2009), S. 670–677

[14] Füllsack, Manfred: *Networking Networks. Origins, Applications, Experiments. Proceedings of the Multi-Disciplinary Network for the Simulation of Complex Systems - Research in the Von-Neumann-Galaxy.* Turia + Kant, 2013

[15] Gadde, Raghedeep; Jampani, Varun; Kiefel, Martin; Kappler, Daniel; Gehler, Peter V.: Superpixel Convolutional Networks using Bilateral Inceptions. In: *European Conference on Computer Vision* (2016)

[16] Graham, Benjamin: Fractional Max-Pooling. In: *Computing Research Repository* (2014)

[17] Hammond, David K.; Vandergheynst, Pierre; Gribonval, Réne: Wavelets on Graphs via Spectral Graph Theory. In: *Applied and Computational Harmonic Analysis* (2011), S. 129–150

[18] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2014), S. 346–361

[19] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian: Deep Residual Learning for Image Recognition. In: *Computer Vision and Pattern Recognition* (2016), S. 83–98

[20] He, Shengfeng; Lau, Rynson W. H.; Liu, Wenxi; Huang, Zhe; Yang, Qingxiong: SuperCNN: A Superpixelwise Convolutional Neural Network for Salient Object Detection. In: *International Journal of Computer Vision* (2015), S. 330–344

[21] Hoffman, Kenneth; Kunze, Ray Alden: *Linear Algebra*. Prentice-Hall, 1971

[22] Hu, Ming K.: Visual Pattern Recognition by Moment Invariants. In: *IRE Transactions on Information Theory* (1962), S. 179–187

[23] Huszár, Ferenc: *How Powerful are Graph Convolutions?* `http://www.inference.vc/how-powerful-are-graph-convolutions-review-of-kipf-welling-2016-2/`. 2016

[24] Iandola, Forrest N.; Han, Song; Moskewicz, Matthew W.; Ashraf, Khalid; William J. Dally; Kreutzer, Kurt: SqueezeNet: AlexNet-Level Accuracy with 50x fewer Parameters and <1MB Model Size. In: *Computing Research Repository* (2016)

[25] Jones, Eric; Oliphant, Travis; Peterson, Pearu: *SciPy: Open Source Scientific Tools for Python*. 2001

[26] Kipf, Thomas N.; Welling, Max: Semi-Supervised Classification with Graph Convolutional Networks. In: *Computing Research Repository* (2016)

[27] Krizhevsky, Alex: *Learning Multiple Layers of Features from Tiny Images*, Department of Computer Science, University of Toronto, Diplomarbeit, 2009

[28] Krogh, Anders; Hertz, John A.: A Simple Weight Decay Can Improve Generalization. In: *Advances in Neural Information Processing Systems 4*, San Francisco, CA: Margon Kaufmann, 1992, S. 950–957

[29] LeCun, Yann; Bengio, Yoshua: Convolutional Networks for Images, Speech, and Time Series. In: *The Handbook of Brain Theory and Neural Networks*. 1998, S. 255–258

[30] LeCun, Yann; Cortes, Corinna; Burges, Christopher J.C.: The MNIST Database of Handwritten Digits. (2010)

[31] Li, Yujia; Zemel, Richard; Brockschmidt, Marc; Tarlow, Daniel: Gated Graph Sequence Neural Networks. In: *Computing Research Repository* (2015)

[32] Luxburg, Ulrike: A Tutorial on Spectral Clustering. In: *Statistics and Computing* (2007), S. 395–416

[33] McKay, Brendan D.; Piperno, Adolfo: Practial Graph Isomorphism, II. In: *Journal of Symbolic Computation* (2014), S. 94–112

[34] Netzer, Yuval; Wang, Tao; Coates, Adam; Bissacco, Alessandro; Wu, Bo; Ng, Andrew Y.: Reading Digits in Natural Images with Unsupervised Feature Learning. In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning* (2011)

[35] Nielsen, Michael .A.: *Neural Networks and Deep Learning*. Determination Press, 2015

[36] Niepert, Mathias; Ahmed, Mohamed; Kutzkov, Konstantin: Learning Convolutional Neural Networks for Graphs. In: *Proceedings of the 33rd International Conference on Machine Learning*, 2016, S. 2014–2023

[37] Organick, Elliott I.: *A Fortran IV Primer*. Addison-Wesley, 1966

[38] Pedregosa, Fabian; Varoquaux, Gaël; Gramfort, Alexandre; Michel, Vincent; Thirion, Bertrand: Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* (2011), S. 2825–2830

[39] Reuter, Martin; Biasotti, Silvia; Giorgi, Daniela; Patanè, Guiseppe; Spagnuolo, Michela: Discrete Laplace-Beltrami Operators for Shape Analysis and Segmentation. In: *Computers & Graphics* (2009), S. 381–390

[40] Richter, Manfred: *Einführung in die Farbmetrik*. Walter de Gruyter, 1981

[41] Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J.: Learning Representations by Back-propagating Errors. In: *Neurocomputing: Foundations of Research*. 1988, S. 696–699

[42] Russakovsky, Olga; Deng, Jia; Su, Hao; Krause, Jonathan; Satheesh, Sanjeev; Ma, Sean; Huang, Zhiheng; Karpathy, Andrej; Khosla, Aditya; Bernstein, Michael; Berg, Alexander C.; Fei-Fei, Li: ImageNet Large Scale Visual Recognition Challenge. In: *International Journal of Computer Vision (IJCV)* (2015), S. 211–252

[43] Saad, Yousef: *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003

[44] Shuman, David I.; Narang, Sunil. K.; Frossard, Pascal; Ortega, Antonio; Vandergheynst, Pierre: The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains. In: *IEEE Signal Processing Magazine* (2013), S. 83–98

[45] Siedhoff, Dominic: *A Parameter-Optimizing Model-Based Approach to the Analysis of Low-SNR Image Sequences for Biological Virus Detection*, Technische Universität Dortmund, Dissertation, 2016

[46] Simonyan, Karen; Zisserman, Andrew: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *Computing Research Repository* (2014)

[47] Srivastava, Nitish; Hinton, Geoffrey; Krizhevsky, Alex; Sutskever, Ilja; Salakhutdinov, Ruslan: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In: *Journal of Machine Learning Research* (2014), S. 1929–1958

[48] Stackoverflow: *Determine adjacent regions in numpy array.* `https://stackoverflow.com/questions/38073433/determine-adjacent-regions-in-numpy-array`. 2016

[49] Stiene, Stefan: *Konturbasierte Objekterkennung aus Tiefenbildern eines 3D-Laserscanners*, Universität Osnabrück, Diplomarbeit, 2015

[50] Stutz, David: *Superpixel Segmentation using Depth Information.* http://davidstutz.de/. 2014

[51] van der Walt, Stéfan; Colvert, S. Chris; Varoquaux, Gaël: The NumPy Array: A Structure for Efficient Numerical Compution. In: *Computing in Science & Engineering* (2011), S. 22–30

[52] van der Walt, Stefan; Schönberger, Johannes L.; Nunez-Iglesias, Juan: scikit-image: Image Processing in Python. In: *PeerJ* (2014)

[53] Vedaldi, Andrea; Soatto, Stefano: Quick Shift and Kernel Methods for Mode Seeking. In: *European Conference on Computer Vision*, 2008, S. 705–718

[54] Weisfeiler, Boris; Lehman, A. A.: A Reduction of a Graph to a Canonical Form and an Algebra Arising During this Reduction. In: *Nauchno-Technicheskaya Informatsia* (1968), S. 12–16

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem Titel

_____

_____

_____

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift