technische universität
dortmund

fakultät für informatik

Master-Thesis

Convolutional Neural Networks auf Graphrepräsentationen von Bildern

Matthias Fey

12. Juli 2017

Gutachter:

Prof. Dr. Heinrich Müller

M.Sc. Jan Eric Lenssen

Lehrstuhl Informatik VII
Graphische Systeme
TU Dortmund

Inhaltsverzeichnis

1 Einleitung

Insbesondere hier schon auf räumlich und spektral eingehen, da es woanders keinen sinn macht

In the context of the generalization of Convolutional Neural Networks (CNNs) to irregular domains modelled by graphs, the problem is the classification or regression of graph signals, i.e. signals who take a value at each vertex of a weighted graph. There is two approaches to convolve a graph signal with a (learned) filter: Spatially sliding a filter on the vertices, as you would slide a filter on a 2D image or a 1D audio signal (the pixels form a grid graph, the time forms a line graph), i.e. the straightforward application of the definition of a convolution. This approach however presents two challenges: (1) the definition of a receptive field / neighbourhood, because sampling on arbitrary graphs is not necessarily uniform and (2), the ordering of nodes, because problem-specific ordering, e.g. spatially ordered pixels or time ordered samples, is missing. These recent works, who present the same ideas differently, spatially define the convolution operator on graphs: The spectral formulation builds on Spectral Graph Theory and Computational Harmonic Analysis. It decomposes the graph Laplacian (via an eigendecomposition) to form a Fourier basis, which properties are analog to the classical Fourier basis on n-dimensional Euclidean spaces. A convolution in the graph domain is then equivalent to a multiplication in the spectral domain. See [1211.0053] for an overview of the Graph Signal Processing field. Note that the spectral approach was also proposed in [1312.5851] (using FFTs) to speed up CNNs on regular 2D Euclidean spaces (i.e. for images). The limitations of this approach is (for now, we have plans to address those): (1) filters are rotation invariant, and (2) filters are not directly transferrable to a different graph. These recent works, who build on one another, spectrally define the convolution operator on graphs:

Generell Lernen auf Graphen, die in der Ebene oder im Raum eingebettet sind, z.B. Straßennetze, Karten.

1.1 Problemstellung

1.2 Aufbau der Arbeit

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 [24] 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 $\mathcal{A}$ heißt *geordnet*, wenn auf dessen Elementen eine reflexive, transitive und antisymmetrische Ordnung $\mathcal{A} \times \mathcal{A}$ definiert ist [15]. Eine geordnete Menge $\mathcal{A}$ mit N Elementen wird über $\mathcal{A} \coloneqq (a_1, \ldots, a_N)$ gekennzeichnet, wobei die Reihenfolge der Elemente $(a_1, \ldots, a_N)$ dessen Ordnung beschreibt.
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 $\langle \mathbf{v}, \mathbf{w} \rangle \coloneqq \sum_{n=1}^{N} \mathbf{v}_n \mathbf{w}_n$ [15]. $\mathbf{v}$ und $\mathbf{w}$ stehen *orthogonal* zueinander, d.h. $\mathbf{v} \perp \mathbf{w}$, genau dann, wenn $\langle \mathbf{v}, \mathbf{w} \rangle = 0$ [15]. 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. $\mathbf{M}^{\top}_{mn} = \mathbf{M}_{nm}$ [15]. 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 [7].

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 [31]. 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. [31]). 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 [15]. 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 [28]). 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 $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ [3]. 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 [3]. 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 [32].
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 $w(v_i, v_j) = 0$ [3]. 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 $w(v_i, v_j) = w(v_j, v_i)$ [3]. 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 [3]. 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 [7]. 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 [7]. 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. $\mathbf{D}_{ii} = d(v_i)$ [7]. Ein Knoten $v \in \mathcal{V}$ heißt *isoliert*, falls dieser keinen Nachbarsknoten besitzt, d.h. $\deg(v) = 0$ [7].
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 $\{v_n\}_{n=1}^N$ [3]. 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 [12]. v_j ist von v_i aus *erreichbar*, wenn $s(v_i, v_j) \in \mathbb{N}$ [3]. 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 $\mathcal{G}$ [3]. 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 $v_j \in \mathcal{V}$ [12]. 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 [12].
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 [25].
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 [32]. 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 [24]. Dabei lernt bzw. approximiert das Netz durch eine Anpassung ihrer Parameter über einer Menge an Trainingsbeispielen eine stetige 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 und dessen Parameter unbekannte Eingaben nicht generalisieren können, wird als ein *überangepasstes* (engl. *overfitted*) Netz bezeichnet [24].
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

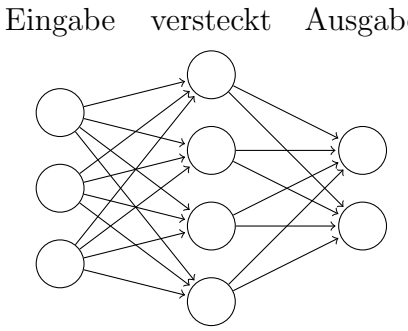


Abbildung 2.1: Beispiel eines Feedforward-Netzes mit drei vollverbundenen Schichten von einer Eingabe mit drei Neuronen zu einer Ausgabe mit zwei Neuronen und einer dazwischenliegenden versteckten Schicht.

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 [24].

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* $\mathrm{sig}(z) \coloneqq 1/(1+\exp(-z))$ oder die *Rectified Linear Unit (ReLU)*-Funktion $\mathrm{ReLU}(z) \coloneqq \max(z, 0)$ zum Einsatz [24]. Die Menge der Gewichte $\mathcal{W} \coloneqq \left\{\mathbf{W}^{(t)}\right\}_{t=2}^{T}$ sowie die Menge der Biaswerte $\mathcal{B} \coloneqq \left\{\mathbf{b}^{(t)}\right\}_{t=2}^{T}$ für $T \in \mathbb{N}$ viele 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 stetige *Kostenfunktion* minimiert wird. Die Größe der Menge $\mathcal{X}$ wird dabei als *Batch-Size* betitelt [24]. Ein Beispiel einer Kostenfunktion ist die *quadratische Kostenfunktion*, die über

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

definiert ist, wobei $\mathbf{y}$ die Ausgabe des Netzes und $\mathbf{y}'$ die erwartete Ausgabe bezüglich $\mathbf{x}$ beschreibt [24]. 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 $\mathbf{y}$ und $\mathbf{y}'$ 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 [24]. 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 [24]. 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*) [24]. 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, in dem der Fehler zwischen der Ausgabe des Netzes und der erwarteten Ausgabe über die Ausgabe- zur Eingabeschicht zurück propagiert wird [29]. Aus den Fehlern der einzelnen Neuronen kann damit schließlich der Gradient ∇C berechnet werden (vgl. [24]). 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 [24].

Eine Spezialform eines neuronalen Netz ist das *Convolutional Neural Network (CNN)*
conv2d

3 Spektrales Lernen auf Graphen

Das *spektrale Lernen auf Graphen* bzw. die Formulierung eines spektralen Faltungsoperator 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.

3.1 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. [32]). 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. [4]).

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.

3.1.1 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 [15]. 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. $\mathbf{M} = \mathbf{M}^\top$ [15]. 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 [15]. 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. $\mathbf{\Lambda}_{ii} = \lambda_i$ [7]. Dann gilt $\mathbf{MU} = \mathbf{U\Lambda}$ und insbesondere ist $\mathbf{M}$ diagonalisierbar über [15]

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

Weiterhin gilt für die k-te Potenz von $\mathbf{M}$, $k \in \mathbb{N}$, [18]

$$\mathbf{M}^k = \left(\mathbf{U\Lambda U}^\top\right)^k = \mathbf{U\Lambda}^k\mathbf{U}^\top. \tag{3.1}$$

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.

$$\sum_{\substack{j=1 \\ j \neq i}}^{N} |\mathbf{M}_{ij}| \leq |\mathbf{M}|_{ii}, \tag{3.2}$$

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 $\mathbf{x} \in \mathbb{R}^N$ [15]. Eigenwerte symmetrischer positiv semidefiniter Matrizen $\lambda_i \in \mathbb{R}_+$ sind positiv reell und es lässt sich folglich auf diesen eine Ordnung definieren mit $0 \leq \lambda_1 \leq \dots \leq \lambda_N \coloneqq \lambda_{\max}$ [15].

3.1.2 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 [4]. 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 [4]. Die Laplace-Matrix ist damit ein geeignetes Mittel zur Betrachtung und Analyse eines Graphen.
Für einen schleifenlosen, ungerichteteten, gewichtet oder ungewichteten Graphen $\mathcal{G}$ und dessen Adjazenzmatrix $\mathbf{A}$ mit Gradmatrix $\mathbf{D}$ ist die *kombinatorische Laplace-Matrix* $\mathbf{L}$ definiert als $\mathbf{L} \coloneqq \mathbf{D} - \mathbf{A}$ [4]. 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. $\mathbf{D}_{ii} = 0$ [4]. Daraus ergibt sich die elementweise Definition

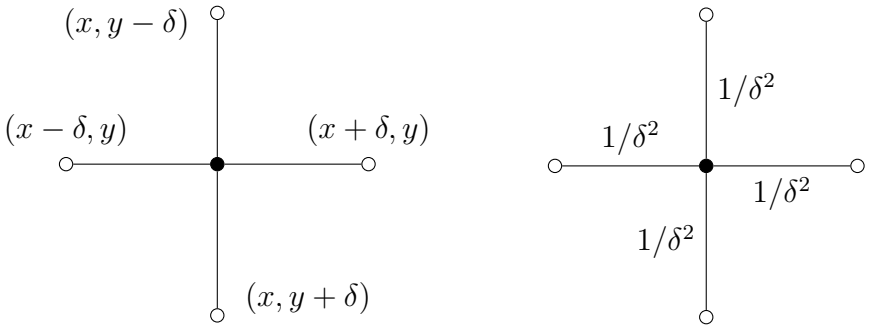
$$\tilde{\mathbf{L}}_{ij} \coloneqq \begin{cases} 1, & \text{wenn } i = j, \\ -\frac{w(v_i,v_j)}{\sqrt{d(v_i)d(v_j)}}, & \text{wenn } v_j \in \mathcal{N}(v_i), \\ 0, & \text{sonst.} \end{cases}$$

Für zusammenhängende Graphen kann $\tilde{\mathbf{L}}$ vereinfacht werden zu [4]

$$\tilde{\mathbf{L}} \coloneqq \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}. \tag{3.3}$$

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 [27]. $\mathbf{L}$ 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 [12]. 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 3.1: 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 [12]. 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 $\{(x, y), (x+\delta, y), (x-\delta, y), (x, y+\delta), (x, y-\delta)\}$ [12] (vgl. Abbildung 3.1). 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 kontinuierlichem Laplace-Beltrami Operator ∇^2 konvergiert [12]. 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, in dem 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 [4]. Folglich besitzt $\mathcal{L}$ nach Kapitel 3.1.1 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 (3.2) 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 $\mathbf{L}\mathbf{u}_1 = \mathbf{0}$ [32]. $\tilde{\mathbf{L}}$ hingegen ist nicht zwingend schwach diagonaldominant. Es lässt sich jedoch zeigen, dass auch für $\tilde{\mathbf{L}}$ gilt, dass $\lambda_1 = 0$ [4].
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 [4]. 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 [4].
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 $s(v_i, v_j) > k$ [12]. Damit beschreibt $\mathcal{L}_i^k$ bildlich gesprochen die Menge an Knoten, die maximal k Kanten von v_i entfernt liegen.

3.2 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 [32]. 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. [32]). 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.

3.2.1 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 Eingabeignal $\mathbf{f}$ 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 [32]

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

Die komplexen Exponentiale $e^{2\pi i\omega t}$ beschreiben dabei die Eigenfunktionen des eindimensionalen Laplace-Beltrami Operators [32]:

$$-\nabla^2 e^{2\pi i\omega t} = -\frac{\partial^2}{\partial t^2} e^{2\pi i\omega t} = (2\pi\omega)^2 e^{2\pi i\omega t}. \tag{3.4}$$

$\hat{f}$ kann damit als die Ausdehnung von f in Bezug auf die Eigenfunktionen des Laplace-Beltrami Operators ∇^2 verstanden werden [12].

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 [32]:

$$\hat{f}(\lambda_i) \coloneqq \langle \mathbf{f}, \mathbf{u}_i \rangle \qquad \text{bzw.} \qquad \hat{\mathbf{f}} \coloneqq \mathbf{U}^\top \mathbf{f}. \tag{3.5}$$

Die inverse Graph-Fourier-Transformation ergibt sich dann als [32]

$$f(v_i) = \sum_{n=1}^{N} \hat{f}(\lambda_n)(\mathbf{u}_n)_i \qquad \text{bzw.} \qquad \mathbf{f} = \mathbf{U}\hat{\mathbf{f}}. \tag{3.6}$$

In der klassischen Fourier-Analyse sind für die Eigenwerte $\left\{(2\pi\omega)^2\right\}_{\omega\in\mathbb{R}}$ in (3.4) 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. [32]).

Die Graph-Fourier-Transformation (3.5) und ihre Inverse (3.6) 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.

3.2.2 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

$$\hat{f}_{\text{out}}(\omega) \coloneqq \hat{f}_{\text{in}}(\omega)\hat{g}(\omega) \tag{3.7}$$

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.

$$(f_{\text{in}} \star g)(t) \coloneqq \int_{\mathbb{R}} f_{\text{in}}(\tau)g(t-\tau)\,\mathrm{d}\tau = f_{\text{out}}(t). \tag{3.8}$$

Wir können die Filterung der Frequenzen in der Fourier-Domäne analog zu (3.7) 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 [32]. $\hat{\mathbf{g}} \in \mathbb{R}^N$ ist damit ein *nicht-parametrischer* Filter, d.h. ein Filter, dessen Werte für alle Frequenzen $\{\lambda_n\}_{n=1}^N$ frei wählbar sind [7]. Daraus ergibt sich analog zu (3.8) der *spektrale Faltungsoperator* auf Graphen in der Knotendomäne mit Hilfe der Graph-Fourier-Transformation (3.5) und ihrer Inversen (3.6) als [7, 32]

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \coloneqq \mathbf{U}\Big(\Big(\mathbf{U}^\top \mathbf{f}_{\text{in}}\Big) \odot \hat{\mathbf{g}}\Big) = \mathbf{f}_{\text{out}}. \tag{3.9}$$

3.2.3 Polynomielle Approximation

Es zeigt sich, dass die Benutzung des spektralen Faltungsoperators in (3.9) 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 ist, denn die Multiplikation mit der dichtbesetzten Eigenvektormatrix $\mathbf{U}$ liegt in $\mathcal{O}(N^2)$ [7]. Zudem muss $\mathbf{U}$ zuerst bestimmt werden — ein kostspieliger Aufwand für Graphen mit möglicherweise weit mehr als hundert Knoten [18]. 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 [7]. 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

$$\hat{g}(\lambda_i) \approx \sum_{k=0}^{K} c_k \lambda_i^k \tag{3.10}$$

vom Grad K mit Koeffizienten $\mathbf{c} \coloneqq [c_0, \ldots, c_K]^\top \in \mathbb{R}^{K+1}$ approximiert werden [7, 12]. Die Filtergröße sinkt somit auf einen konstanten Faktor K mit Lernaufwand $\mathcal{O}(K)$, dem gleichen Aufwand klassischer zweidimensionaler CNNs [7]. $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ ergibt dann nach (3.1), (3.9) und (3.10) approximiert durch [7]

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx \sum_{k=0}^{K} c_k \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^\top \mathbf{f}_{\text{in}} = \sum_{k=0}^{K} c_k \mathcal{L}^k \mathbf{f}_{\text{in}}. \tag{3.11}$$

Insbesondere ist die spektrale Faltung damit nicht mehr abhängig von der Berechnung der Eigenwerte bzw. Eigenvektoren von $\mathcal{L}$. Mittels Kapitel 3.1.2 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 [12].

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 [32].

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. [12]). 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 $T_1(x) = x$ [12]. Ein Tschebyschow-Polynom T_k ist ein Polynom k-ten Grades und liegt im Intervall $[-1, 1]$ für $x \in [-1, 1]$ [12]. 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 [7]. Die spektrale Faltung mittels Tschebyschow-Polynomen ergibt sich dann nach (3.11) als

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx \sum_{k=0}^{K} c_k \mathbf{U} T_k\Big(\tilde{\mathbf{\Lambda}}\Big) \mathbf{U}^\top \mathbf{f}_{\text{in}}, \tag{3.12}$$

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 [7]. $\mathbf{U} T_k\Big(\tilde{\mathbf{\Lambda}}\Big) \mathbf{U}^\top$ kann aufgrund der polynomiellen Form von T_k und (3.1) 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 [7]. Damit kann (3.12) weiter vereinfacht werden zu

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx \sum_{k=0}^{K} c_k T_k\Big(\tilde{\mathcal{L}}\Big) \mathbf{f}_{\text{in}}. \tag{3.13}$$

und ist nun ein rekursiv formulierter Faltungsoperator, der weiterhin ohne die explizite Berechnung von $\mathbf{U}$ auskommt [7]. 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 uns 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. [12]). $\mathbf{f}_{\text{out}}$ lässt sich damit über $K + 1$ Multiplikationen einer dünnbesetzen Matrix mit einem Vektor und einer abschließenden Vektormultplikation beschreiben. Mit $N \leq |\mathcal{E}|$ ergibt dies eine finale Laufzeit der spektralen Faltung von $\mathcal{O}(K|\mathcal{E}|)$ [7].

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, in dem für jede Ausgabekarte über jede Eingabekarte gefaltet und dessen Ergebnisse sukzessive aufsummiert werden (vgl. Kapitel 2.3). Modellieren wir diesen Fall für den spektralen Faltungsoperator, dann erhalten wir rekursiv für eine Merkmalsmatrix $\mathbf{F}_{\text{in}} \in \mathbb{R}^{N \times M_{\text{in}}}$ gefaltet auf eine Merkmalsmatrix $\mathbf{F}_{\text{out}} \in \mathbb{R}^{N \times M_{\text{out}}}$ über den N Knoten eines Graphen $\mathcal{G}$ mittels des Filtertensors $\mathbf{W} \in \mathbb{R}^{(K+1) \times M_{\text{in}} \times M_{\text{out}}}$

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

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 $\bar{\mathbf{F}}_1 = \tilde{\mathcal{L}}\,\mathbf{F}_{\text{in}}\mathbf{W}_1$.

3.3 Graph Convolutional Networks

Kipf und Welling motivieren einen weiteren Ansatz zur Faltung auf Graphen, genannt *Graph Convolutional Network (GCN)*, der auf der Methodik des spekralen Faltungsoperators aus Kapitel 3.2 aufbaut und dabei wie eine „differenzierbare und parametrisierte Generalisierung des eindimensionalen Weisfeiler-Lehman-Algorithmus auf Graphen“ fungiert [18].

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 (3.13) 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 [18]. Mit $K = 1$ betrachtet der spektrale Faltungsoperator nur noch die lokale Nachbarschaft eines jeden Knotens (vgl. 3.2.3). 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. [14, 18, 33]). Unter dieser Restriktion vereinfacht sich $\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}}$ zu

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx c_0\, \mathbf{f}_{\text{in}} + c_1\Big(\frac{2}{\lambda_{\max}}\mathcal{L} - \mathbf{I}\Big)\mathbf{f}_{\text{in}} \tag{3.14}$$

mit zwei freien Parametern c_0 und c_1 [18]. Für $\tilde{\mathbf{L}}$ auf einem zusammenhängenden Graphen $\mathcal{G}$ gilt dann nach (3.3) und (3.14) weiter

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx c_0\, \mathbf{f}_{\text{in}} + c_1\big(\tilde{\mathbf{L}} - \mathbf{I}\big)\mathbf{f}_{\text{in}} = c_0\, \mathbf{f}_{\text{in}} - c_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{f}_{\text{in}}, \tag{3.15}$$

wobei $\lambda_{\max} \coloneqq 2$ auf dessen obere Schranke gesetzt wird [18]. Um die Gefahr des Overfittings und die Anzahl an Berechnungen pro Schicht weiter zu beschränken, reduziert sich (3.15) mit einem einzigen Parameter $c \coloneqq c_0$ mit $c = -c_1$ zu [18]

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

Die skalierten Eigenwerte von $\tilde{\mathbf{\Lambda}}$ liegen aufgrund der Addition mit $\mathbf{I}$ nun im Intervall $[0, 2]$ (vgl. [18]). Demnach können wiederholte Anwendungen des Faltungsoperators zu „numerischen Instabilitäten und folglich zu explodierenden oder verschwindenen Gradienten" führen [18]. 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

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx c\,\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{f}_{\text{in}} \tag{3.16}$$

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 3.2.3 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 3.1: Eindimensionaler Weisfeiler-Lehman-Algorithmus auf einer initialen Knotenfärbung bzw. Merkmalsfunktion $\mathbf{f}^{(0)} \in \mathbb{R}^N$ eines Graphen $\mathcal{G}$ mit $v_i \in \mathcal{N}(v_i)$ [37]. Der Prozess der Verfärbung eines jeden Knotens v_i auf Basis der Farben seiner lokalen Nachbarsknoten 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

$$\mathbf{F}_{\text{out}} \coloneqq \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{F}_{\text{in}} \mathbf{W} \tag{3.17}$$

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 [18].

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 [9]. 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 3.1).

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

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

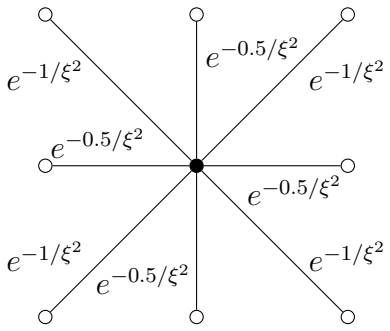
wobei $w(v_i, v_j)/\sqrt{d_i d_j} \in \mathbb{R}$ die Normalisierungskonstante für die Kante $(v_i, v_j) \in \mathcal{E}$ entsprechend der Normalisierung $\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ aus (3.16) und $\mathbf{W}^{(t)} \in \mathbb{R}^{N \times N}$ die Filtermatrix der t-en Faltungsschicht beschreibt [18]. Folglich kann die Faltung des GCNs als „differenzierbare und parametrisierte Generalisierung des eindimensionalen Weisfeiler-Lehman-Algorithmus auf Graphen“ verstanden werden [18].

3.4 Erweiterung auf Graphen im zweidimensionalen Raum

Die Ansätze von Defferrard u. a. aus Kapitel 3.2 und von Kipf und Welling aus Kapitel 3.3 zeigen konkurrenzfähige Resultate auf einer Reihe von Datensätzen auf Graphen (vgl. [7, 18]). 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 $K = 3$ [18].
Die spektrale Faltung auf Graphen entspricht einer Generalisierung der Faltung klassischer CNNs auf zweidimensionalen Bildern [16]. 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 [7].
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 3.2). 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 3.2: 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 (**??**) 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. [16]). 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

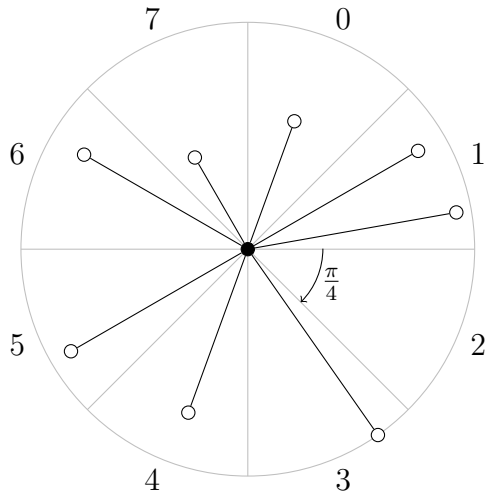


Abbildung 3.3: Partitionierung eines Graphknotens im Uhrzeigersinn in $P = 8$ Bereiche mit gleichmäßigen Innenwinkeln der Größe $\pi/4$.

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.

3.4.1 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 **??**). 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 3.3 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 3.3 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 (3.16) 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

$$\mathbf{f}_{\text{in}} \star \hat{\mathbf{g}} \approx \sum_{p=0}^{P} c_p\tilde{\mathbf{A}}_p\mathbf{f}_{\text{in}} \tag{3.18}$$

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 (3.18) 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 $i \in \{0, \dots, N\}$. 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 (3.18) mit den Parametern $[c_0, \dots, c_8]^\top \in \mathbb{R}^9$ folgt damit sofort die Äquivalenz zu conv2d($\mathbf{f}_{\text{in}}$) auf regulären Gittern.

Implementierung. Analog zu (3.17) 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 (3.17) 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.

3.4.2 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 $\mathbf{d}_0, \ldots, \mathbf{d}_M \in \mathbb{R}^d$ [6]. 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

$$b(t) \coloneqq \sum_{m=0}^{M} \mathbf{d}_m \mathrm{N}_m^K(t), \tag{3.19}$$

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 $(k-1 \to k)$

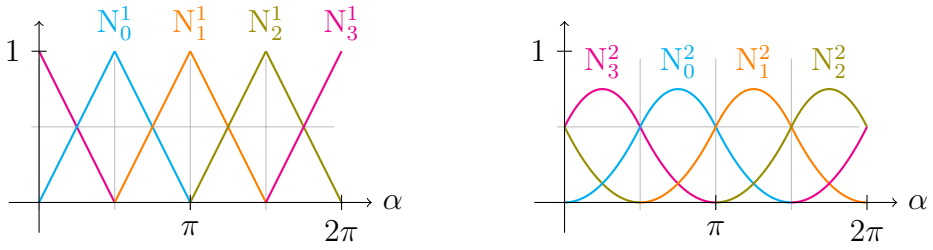
$$\mathrm{N}_m^k(t) \coloneqq \frac{t - t_m}{t_{m+k} - t_m} \mathrm{N}_m^{k-1}(t) + \frac{t_{m+k+1} - t}{t_{m+k+1} - t_{m+1}} \mathrm{N}_{m+1}^{k-1}(t)$$

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 $t \in (t_0, t_{M+1}]$ [6].

Wir können die Definition der B-Spline-Kurve $b(t)$ aus (3.19) nutzen, um sie aufbauend auf Kapitel 3.4.1 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 (3.18) nun die Koeffizienten der



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

Abbildung 3.4: 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).

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 $(v_i, v_j) \notin \mathcal{E}$. Wir können uns $b(\alpha)$ damit weiterhin als eine P-fache Partitionierung von $\mathcal{G}$ auf Basis seiner Kantenausrichtungen vorstellen, mit dem Unterschied, dass $b(\alpha)$ nun eine lokale Kontrollierbarkeit inne hält. 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 3.4 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 (3.19).

Faltungsoperator. Es kann folglich analog zu (3.18) 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 aggreriert 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

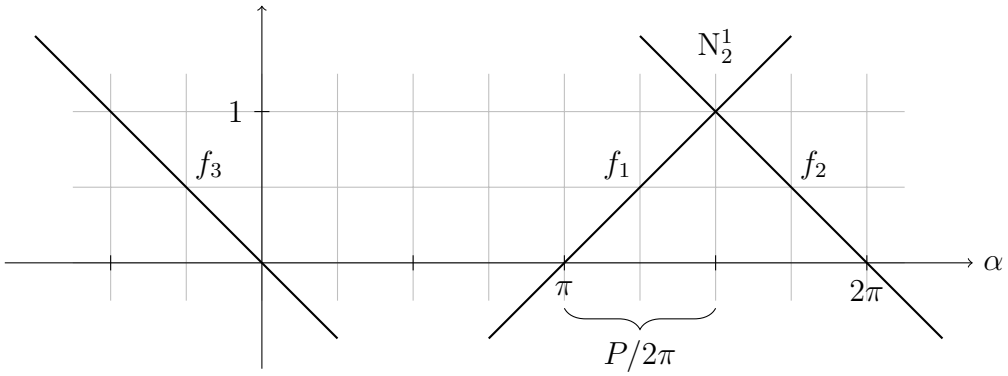


Abbildung 3.5: Beweisidee zur „kreisförmigen“ Dreiecksfunktion für B-Spline-Funktionen mit $K = 1$ und $P = 4$ als Darstellung über eine Minimum-Maximumfunktion auf drei Geraden, hier illustriert am Beispiel N_2^1.

aus (3.18) 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 (3.4.1) 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, in dem 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 3.5). Den Geraden kann die Steigung $P/2\pi$ bzw. $-P/2\pi$ zugeordnet werden und sie können damit folglich über

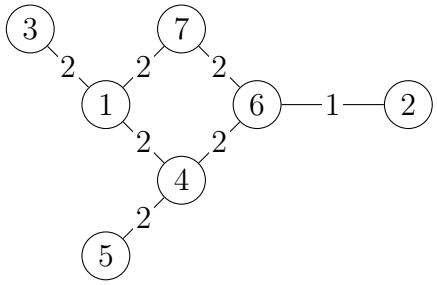
$$
\begin{aligned}
f_1(\alpha) &:= \frac{P}{2\pi}\left(\alpha - 2\pi\frac{p}{P}\right) = \frac{P}{2\pi}\alpha - p \\
f_2(\alpha) &:= -\frac{P}{2\pi}\left(\alpha - 2\pi\frac{p+2}{P}\right) = -\frac{P}{2\pi}\alpha + p + 2 \\
f_3(\alpha) &:= -\frac{P}{2\pi}\left(\alpha - 2\pi\frac{p+2}{P} + 2\pi\right)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

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 $p \in \{0, \ldots, P-1\}$.

3.5 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 [24].

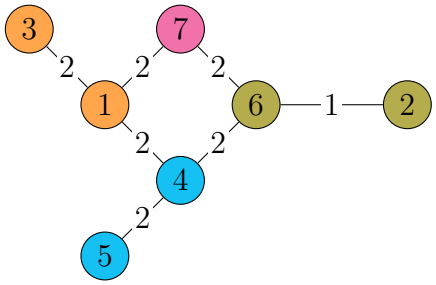
Eine Pooling-Operation auf Graphen erfordert dafür analog zum Pooling auf einem regulären Gitter eine logische Zusammenfassung von Knoten zu Clustern [7]. Der zu verwendende Clusteralgorithmus hat dabei jedoch einige Anforderungen, um als Poolingoperation in einem Netz genutzt werden zu könnnen. So muss dieser vorallem 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 [7]. 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 vorallem Clustertechniken sinnvoll, die die Größe eines Graphen um den Faktor zwei reduzieren [7]. Damit können größere Poolinggrößen über die mehrfache Anwendung des Clusteralgorithmus realisiert werden.



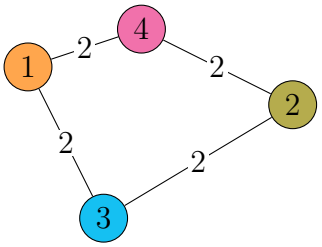
(a) Graph mit zufälliger Ordnung

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 3.6: 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).

3.5.1 Graphvergröberung

Es existieren eine Reihe von Clustertechniken auf Graphen [7, 8, 22]. Defferrard u. a. benutzen für die Poolingschicht eines Netzes auf Graphen die *Vergröberungsphase* des mehrstufigen Clusteralgorithmus *Graclus* [8]. Graclus zeigt sich dabei als besonders erfolgreich über einer Vielzahl von Graphen bei minimaler Berechnungskomplexität [7]. Abbildung 3.6 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 [8]. 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 [8]. Die Kantengewichte $\mathcal{E}_{l+1}$ werden dann über die Summe der Kantengewichte der je-

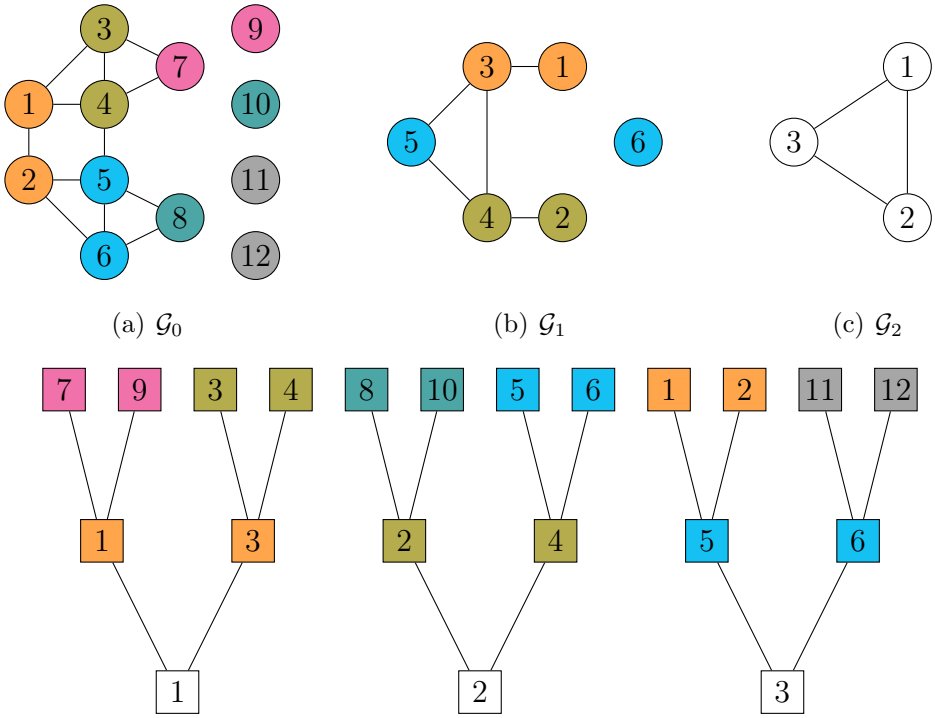
weiligen Knotenuntermengen ohne Schleifen gebildet [8].
Die Auswahl der Untermengen erfolgt gierig. Die Knoten des Graphen $\mathcal{G}_l$ werden als unmarkiert initialisert und zufällig durchlaufen. Für jeden Knoten $v_i \in \mathcal{V}_l$, der noch unmarkiert ist, wird ein lokaler, ebenfalls noch unmarkierter, Nachbarschaftsknoten $v_j \in \mathcal{N}(v_i)$ nach einer zuvor definierten Strategie bestimmt und v_i sowie v_j zu einem Superknoten $v^* \coloneqq \{v_i, v_j\} \in \mathcal{V}_{l+1}$ verschmelzt. Anschließend werden v_i sowie v_j markiert. Falls v_i keinen unmarkierten Nachbarsknoten 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 [8]. Bei weiteren Anwendungen des Clusteralgorithmus werden die Knoten im Folgenden anhand ihrer aufsteigenden Knotengrade durchlaufen [7].
Strategien für die Nachbarschaftsauswahl basieren üblicherweise auf der Maximierung von $w(v_i, v_j)$ [8]. 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 [7, 8]. 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 [7]. 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.

3.5.2 Effizientes Pooling mittels binärer Bäume

Ein tiefes neuronales Netz besitzt für gewöhnlich viele Poolingschichten. Eine Pooling-Operation auf den Merkmalen der Graphknoten muss folglich vorallem 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 3.7: Illustration einer Poolingoperationen 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, so dass 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 [7]. 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 [7].

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 [7]. 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 [7]. Abbildung 3.7 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 [7]. Ebenso kann auf diesen Fakeknoten weiterhin ohne Einschränkungen gefaltet werden, da sie keinerlei Nachbarsknoten 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 [7]. Diese Anordnung der Knoten macht den Prozess des Poolings extrem effizient und erlaubt ebenso eine parallele Verarbeitungsarchikektur auf GPUs [7].

Größere Poolinggrößen müssen ein Vielfaches von zwei sein und können dann ebenso leicht über eine tieferstufigere Vergröberung implementiert werden (vgl. Abbildung 3.7).

3.5.3 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 **??**). Dann lässt sich der mehrstufige Clusteralgorithmus Graclus aus Kapitel 3.5.1 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 **??** 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 **??** 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.

3.6 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 [24]. 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, so dass diese alle die gleiche Bildgröße besitzen (zum Beispiel 224×224) [13]. 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 hinzufügen, 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 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 [13]. Die vollverbundenen Schichten jedoch brauchen rein nach ihrer Definition eine feste Eingabegröße [13]. Dadurch ergibt sich die Bedingung einer festen Eingabegröße lediglich durch den Übergang von einer Faltungs- zu einer vollverbundenen Schicht [13]. 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 und Merkmale auf den Knoten folglich bereits den globalen Graphen abdecken sollten.
Abbildung 3.8 veranschaulicht die beschriebene typische spektrale Netzarchitektur. Alternative Architekturen bzw. alternative Ansätze im Hinblick auf die Durchschnittsbildung zwischen der Faltungs- und der vollverbundenen Schicht werden in Kapitel 5 diskutiert.

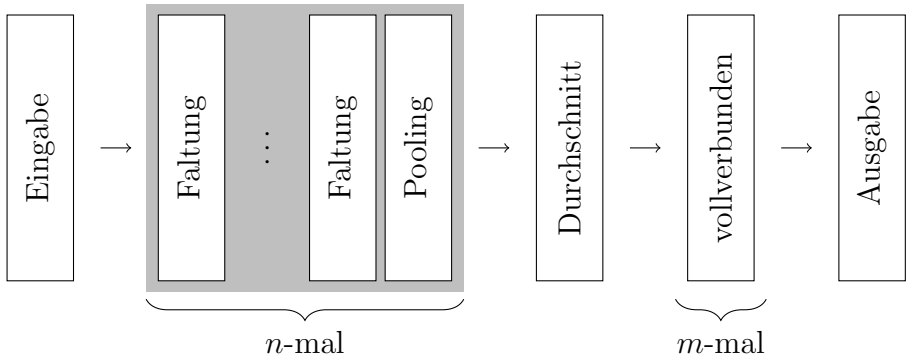


Abbildung 3.8: Typische spektrale Netzarchitektur auf Graphen bestehend aus beliebig vielen verketteten Faltungsschichten gefolgt von jeweils einer Poolingschicht. Im Anschluss sorgt die Benutzung einer Durchschnittsbildung auf den Knoten jedes Merkmals für die Verwendung von vollverbundenen Schichten hin zur Ausgabe.

4 Evaluation

Das folgende Kapitel evaluiert die beschriebenen Ansätze der Graphrepräsentationen von Bildern aus Kapitel **??** sowie die räumlichen und spektralen Ansätze zum Lernen auf diesen (Kapitel **??** und 3) am Beispiel der Bildklassifizierung über mehrerer Datensätze. 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.

4.1 Merkmalsselektion

Die in Kapitel **??** ermittelten Formmerkmale auf den Knoten eines Graphen, der aus einer Superpixelrepräsentation generiert wurde, besitzen mit 38 Dimensionen 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 uns 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 4.1 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-

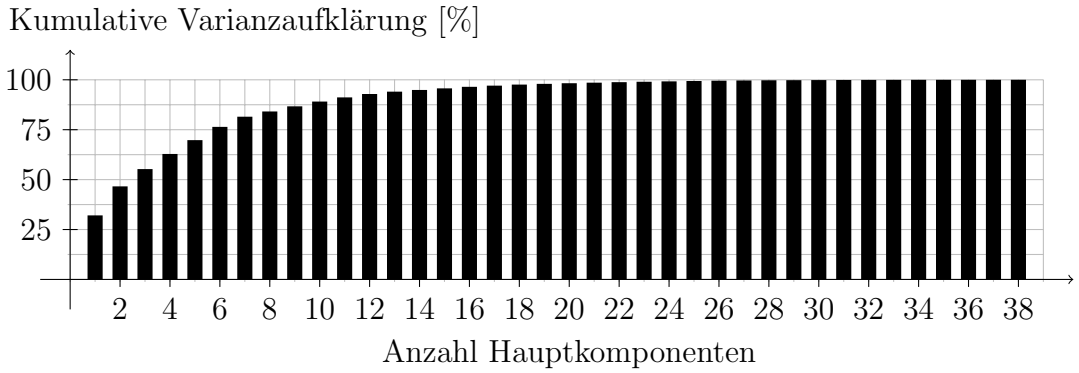


Abbildung 4.1: Kumulative Varianzabdeckung der Hauptkomponenten einer PCA in Abhängigkeit ihrer Anzahl. Die Merkmale wurden dabei aus den Knotenmerkmalen von 1000 segmentierten Bildern gewonnen. Bereits nach wenigen Hauptkomponenten ist der Größteil der erklärten Varianz abgedeckt.

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 [26].

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. [26]). 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 basierend ihrer Gewichte aus, in dem rekursiv eine immer kleinere Menge an Merkmalen betrachtet wird [26]. 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 [26].

4.2 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, die eine beliebige Eingabe, d.h. ein Bild (oder in diesem Fall ein Graph), auf eine Y-dimensionale Ausgabe $\mathbf{y} \in \mathbb{R}^Y$ überführt, wobei $\mathbf{y}_i$ die Evidenz des Auftretens der Klasse y_i in der Eingabe beschreibt. Der Index des höchsten Werts des Ausgabevektors $\mathbf{y}$ 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 weiterforschtes 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.

4.2.1 Datensätze

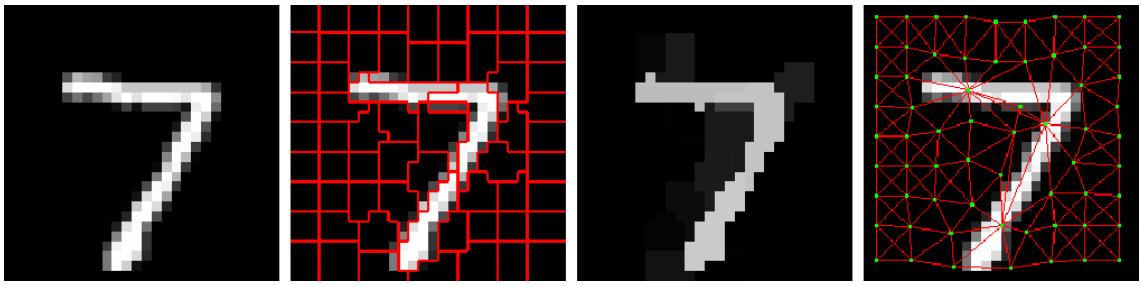
Die vorgestellten Faltungsmethoden aus Kapitel **??** und 3 bezüglich des Lernens auf Graphen im zweidimensionalen euklidischen Raum werden über einer Reihe von Datensätzen verifiziert, die im Folgenden vorgestellt werden. Dafür werden die Bildermengen in eine Superpixelrepräsentation (Simple Linear Iterative Clustering (SLIC) und Quickshift) konvertiert und darauf basierend in eine Graphrepräsentation transformiert (vgl. Kapitel **??**). 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 4.1 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 [21]. Er besteht aus 55000 Trainingsbildern, 5000 Validierungsbildern sowie 10000 Testbildern. Die Bilder des Datensatzes sind einheitlich auf die

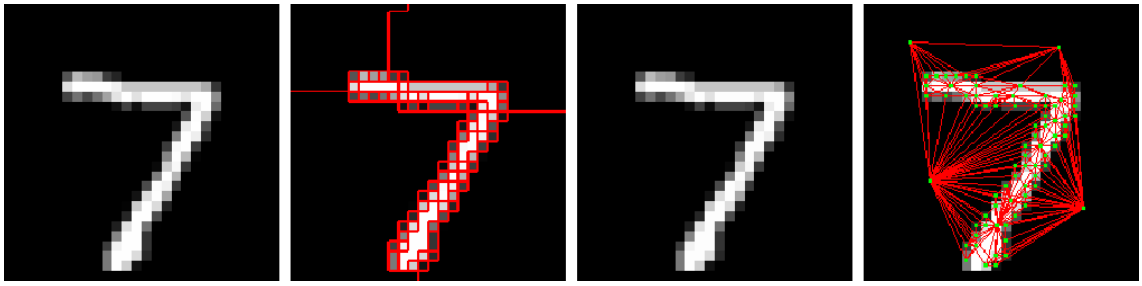
SLIC						Quickshift					
K	100	F	5			ξ	2	α	1	S	2
$\overline{N}$	64.6	$N_{\min}$	50	$N_{\max}$	80	$\overline{N}$	82.1	$N_{\min}$	5	$N_{\max}$	154
$\overline{\deg}$	5.7	$\deg_{\min}$	1	$\deg_{\max}$	19	$\overline{\deg}$	6.8	$\deg_{\min}$	1	$\deg_{\max}$	101

Tabelle 4.1: 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 [21]. Aufgrund seiner kleinen Datengröße und leichten Handhabung gilt er als die ideale Einführung in Prinzipien des maschinellen Lernens und zeichnet sich damit als ideal 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 4.1 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 **??** 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 4.2 veranschaulicht die beiden Superpixel- bzw. Graphrepräsentationen anhand eines Bildes aus dem MNIST Datensatz.
Die in Kapitel 4.1 beschriebene Merkmalsselektion reduziert für MNIST die Menge an Formmerkmalen (vgl. Kapitel **??**) im ersten statistisch basierten Test auf 12



(a) SLIC



(b) Quickshift

Abbildung 4.2: 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 4.2: Merkmalsselektion des MNIST Datensatzes zu 9 Formmerkmalen.

Merkmale, welche im zweiten Schritt rekursiv auf 9 Merkmale weiter beschränkt wird. Die ermittelten (unterschiedlichen) Formmerkmale für SLIC und Quickshift bezüglich MNIST sind in Tabelle 4.2 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 [19]. 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,

SLIC						Quickshift					
K	200	F	5			ξ	1	α	1	S	5
$\overline{N}$	232.1	$N_{\min}$	186	$N_{\max}$	263	$\overline{N}$	182.0	$N_{\min}$	18	$N_{\max}$	624
$\overline{\deg}$	6.3	$\deg_{\min}$	1	$\deg_{\max}$	21	$\overline{\deg}$	7.4	$\deg_{\min}$	1	$\deg_{\max}$	67

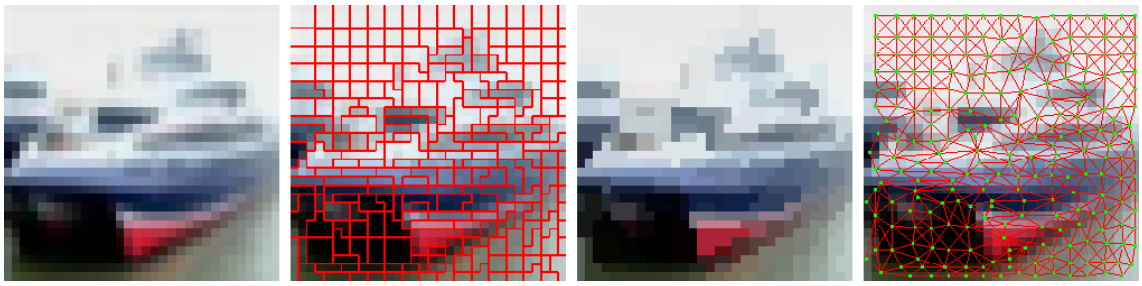
Tabelle 4.3: Wahl der Superpixelparameter des CIFAR-10 Datensatzes.

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 4.4: Merkmalsselektion des CIFAR-10 Datensatzes zu 9 Formmerkmalen.

Hund, Frosch, Pferd, Schiff und Lastwagen. Die Bilder der Klassen sind dabei *einander ausschließend*. So enthält die Klasse „Auto“ nur kleinere Personenwagen, wohingegen die Klasse „Lastwagen“ auch nur als solche zu klassifizierenden Fahrzeuge enthält [19]. 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 [19].
Tabelle 4.3 zeigt die Wahl der Parameter der beiden Superpixelalgorithmen. Im Vergleich zu dem MNIST Datensatz würden 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 Knotengraden. 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, d.h. im Durchschnitt werden nur zwei Pixel einem Superpixel zugeordnet werden, und kann als extremer Ausreißer gewertet werden. Abbildung 4.3 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 4.4 zeigt die berechnete Wahl der Merkmale der Selektion. Es ist auffällig, dass sich für die beiden Superpixelrepräsentation



(a) SLIC

(b) Quickshift

Abbildung 4.3: Bild aus dem CIFAR-10 Datensatz, jeweils dargestellt als (1) Originalbild, (2) Superpixelrepräsentation, (3) Durchschnittsfarbe der Superpixel und (4) Graphrepräsentation.

dabei die Selektion der Merkmale bei nur drei von neun Merkmalen unterscheidet. Inklusive den Durchschnittsfarbwerten eines Superpixels über den drei Farbkanälen 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 4.3).

PASCAL VOC. Der *Pascal Visual Object Classes (PASCAL VOC)* Datensatz besteht aus 17126 farbigen Bildern beliebiger Größen, der aufgrund seiner ausführlichen Bildannotierungen nicht nur für eine Bildklassifizierung, sondern ebenfalls für Objektdetektionen oder eine Bildsegmentierung genutzt werden kann [10]. Zu jedem Bild stehen insbesondere Informationen zu den enhaltenden Objekten und deren Hüllkörpern zur Verfügung [10]. Ein Bild besitzt dabei minimal ein Objekt der 20 annotierten Objektklassen. Gleiche oder unterschiedliche Objektklassen kön-

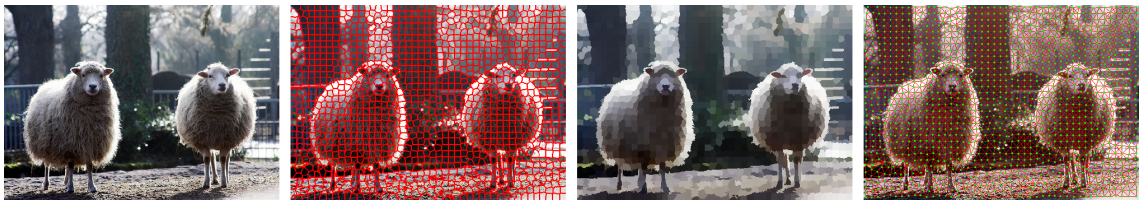
SLIC						Quickshift					
K	1600	F	30			ξ	2	α	0.75	S	8
$\overline{N}$	1540.9	$N_{\min}$	1082	$N_{\max}$	1839	$\overline{N}$	2131.6	$N_{\min}$	234	$N_{\max}$	29010
$\overline{\deg}$	6.5	$\deg_{\min}$	1	$\deg_{\max}$	50	$\overline{\deg}$	7.7	$\deg_{\min}$	1	$\deg_{\max}$	256

Tabelle 4.5: Wahl der Superpixelparameter des PASCAL VOC Datensatzes.

nen mehrfach in einem Bild existieren. Die enthaltenden Objekte der Bilder sind im Einzelnen [10]:

- 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 4.5. 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 [11]. Abbildung 4.4 veranschaulicht die Wahl der Parameter der beiden Superpixelalgorithmen.
Auch für PASCAL VOC bestimmt die Merkmalsselektion 9 Merkmale aus den 38 Formmerkmalen (vgl. Tabelle 4.6). Dabei wird sich erneut nicht für die eigentlichen



(a) SLIC

(b) Quickshift

Abbildung 4.4: Bild aus dem PASCAL VOC Datensatz, jeweils dargestellt als (1) Originalbild, (2) Superpixelrepräsentation, (3) Durchschnittsfarbe der Superpixel und (4) 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 4.6: Merkmalsselektion des PASCAL VOC Datensatzes zu 9 Formmerkmalen.

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 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 existieren eine Reihe weitere 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 [30]. 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 [30]. 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 [10]. Namenhaft zu erwähnen sei weiterhin der von CIFAR-10 inspirierte *STL-10* Datensatz von Stanford mit einer einheitlichen Bildergröße von 96×96 Pixeln [5]. Er enthält aber nur sehr wenige annotierte Bilder und findet seine Anwendung daher vorallem 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 [23]. Er enthält damit weitaus mehr Bilder als der zu ihm ähnliche MNIST Datensatz und versucht dabei ein weitaus schwereres Problem zu lösen — der Objekterkennung von Zahlen in natürlichen Szenen. Er kann jedoch insbesondere aufgrund der beliebigen Anzahl an Ziffern im Bild nicht für eine Bildklassifizierung genutzt werden.

4.2.2 Metriken

Eine typische Aktivierungsfunktion für die Ausgabe $\mathbf{y} \in \mathbb{R}^Y$ 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, 24]. Damit gilt insbesondere $\operatorname{softmax}(\mathbf{y}) \in [0, 1]^Y$ sowie $\sum_{i=1}^{Y} = \operatorname{softmax}(\mathbf{y})_i = 1$ [24]. 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 fällen

(vgl. [24]). Die *Kreuzentropie* ist dabei über einer Eingabemenge $\mathcal{X}$ definiert als

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

wobei $\mathbf{y}' \in \{0,1\}^Y$ die *Ground-Truth* der Eingabe $\mathbf{x} \in \mathcal{X}$ als *One-Hot*-Kodierung beschreibt [1, 24]. $H(\mathcal{X})$ ist aufgrund der Intervalleinschränkung der jeweiligen Ausgaben $\mathbf{y}$ von $\mathbf{x} \in \mathcal{X}$ mit Hilfe der Softmax Regression stets positiv und wird umso kleiner, je ähnlicher sich $\mathbf{y}'$ und dessen Ground-Truth $\mathbf{y}$ werden [24].
Zur Evaluation der Ergebnisse einer Klassifizierung wird neben der Kostenfunktion die *Genauigkeit* (engl. *Accuracy*) über einer Eingabemenge $\mathcal{X}$ als

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

definiert, wobei $\text{argmax}(\mathbf{y}) = i$ genau dann, wenn $\mathbf{y}_i \geq \mathbf{y}_j$ für alle $1 \leq j \leq Y$. 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 [24].

4.2.3 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 (**??**) 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 4.3 manuell aufgelistet sind, weitere einstellbare Größen. Alle Netze wurden mit einer Batch-Size von 64 trainiert. Die Gewichte des Netzes wurden normal verteilt mit 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 verwen-

det. 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 wurden 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. [20, 34]).

4.2.4 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 gespiegelt werden, d.h.

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

sodass ein Bild erzeugt wird, welches das Ursprungsbild „von der anderen Seite“ zeigt [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 [1]. Alle Augmentierungsschritte werden 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.

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

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.

4.2.5 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*, d.h. Anteil getesteter Quelltextzeilen, von 88% und wurde auf allen bedeutenden Pythonversionen (2.7 und > 3.5) 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 Kosten der Ausgabe eines Berechnungsgraphen über das Backpropagationverfahren minimiert werden können. Dies erlaubt insbesondere den Bau der implementierten

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

Faltungsoperatoren mit den in `tensorflow` zur Verfügung stehenden Operationen.

Zur Generierung der Superpixelrepräsentationen der Bildermengen wurde die Bibliothek `scikit-image` (`skimage`) benutzt [36]. 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 4.4).

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 [17, 35]. 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 [26].

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 wurden 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 Daten-

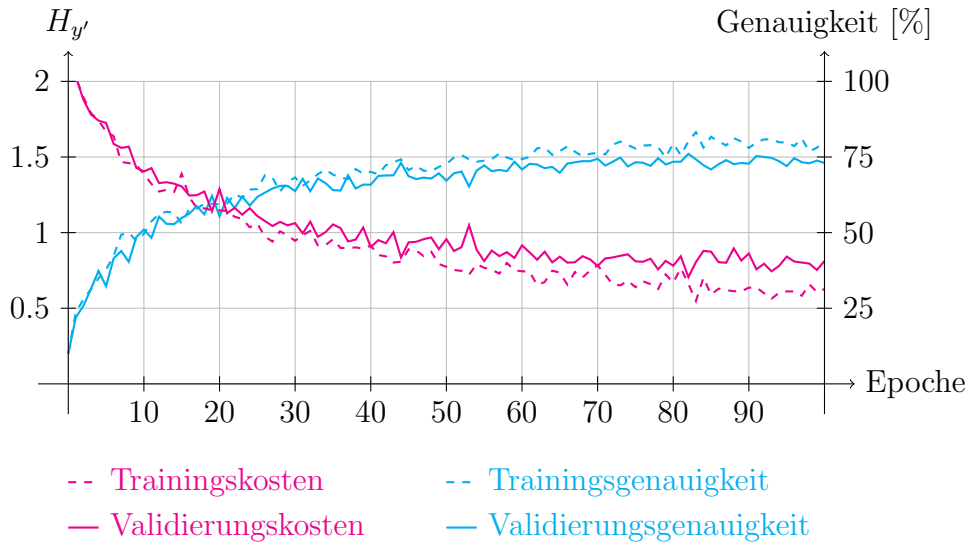


Abbildung 4.5: Kosten- und Genauigkeitsverlauf eines Trainings auf dem CIFAR-10 Datensatz mit dem spektralen Faltungsoperator auf Graphen im zweidimensionalen euklidischen Raum, welche über Quickshift generiert wurden. Die gestrichelten Linien zeigen die Kosten und Genauigkeiten des Trainingsdatensatzes in Abhängigkeit der Epochen, wohingegen die durchgezogenen Linien sich auf den Validierungsdatensatz beziehen. Nach der 60ten Epoche stellt sich eine deutliche Verlangsamung der Lerngeschwindigkeit ein.

sätze 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 4.4. Ein entscheidener Vorteil der Vorverarbeitung zur Laufzeit ist jedoch die in Kapitel 4.2.4 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.

4.3 Ergebnisse

4.4 Laufzeitanalyse

Vergleich bezüglich vorhandener Implementierungen.

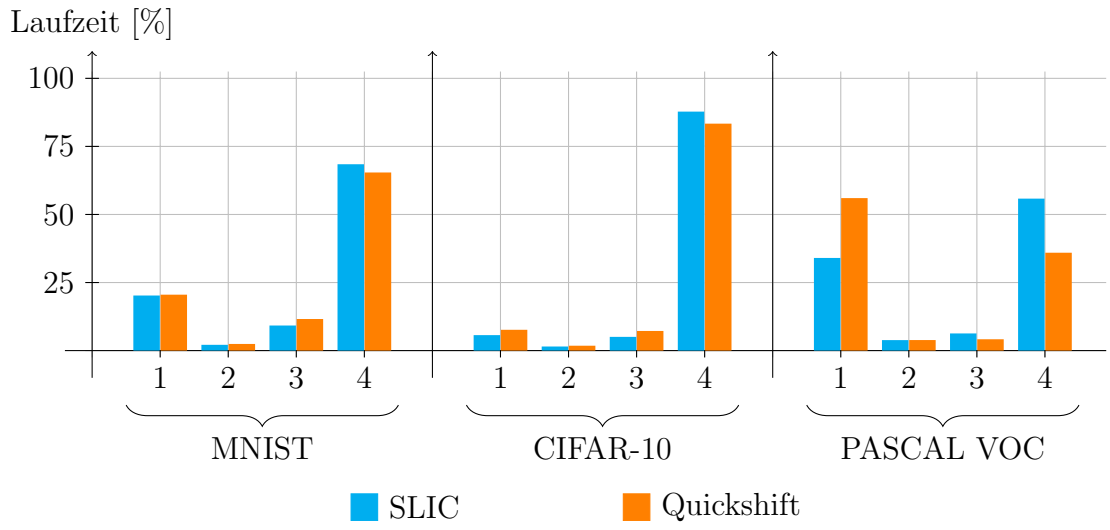


Abbildung 4.6: Laufzeitverteilung der einzelnen räumlichen Vorverarbeitungsschritte für alle Datensätze und alle Superpixelalgorithmen. Die Berechnung der Receptive-Fields umfasst je 25 Knoten bei einer Knotenauswahl mit Schrittweite 2. Sie zeigt sich fast in allen Datensätzen als die dominante Berechnungsaufgabe.

	Räumlich [ms]		Spektral [ms]	
	SLIC	Quickshift	SLIC	Quickshift
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 4.7: Laufzeiten der räumlichen sowie spektralen Vorverarbeitung für alle Datensätze und Superpixelalgorithmen basierend auf den Laufzeiten aus Abbildung 4.6 und 4.7. Die räumliche Vorverarbeitung ist aufgrund der einzelnen Receptive-Field-Berechnungen deutlich ineffizienter.

4.5 Diskussion

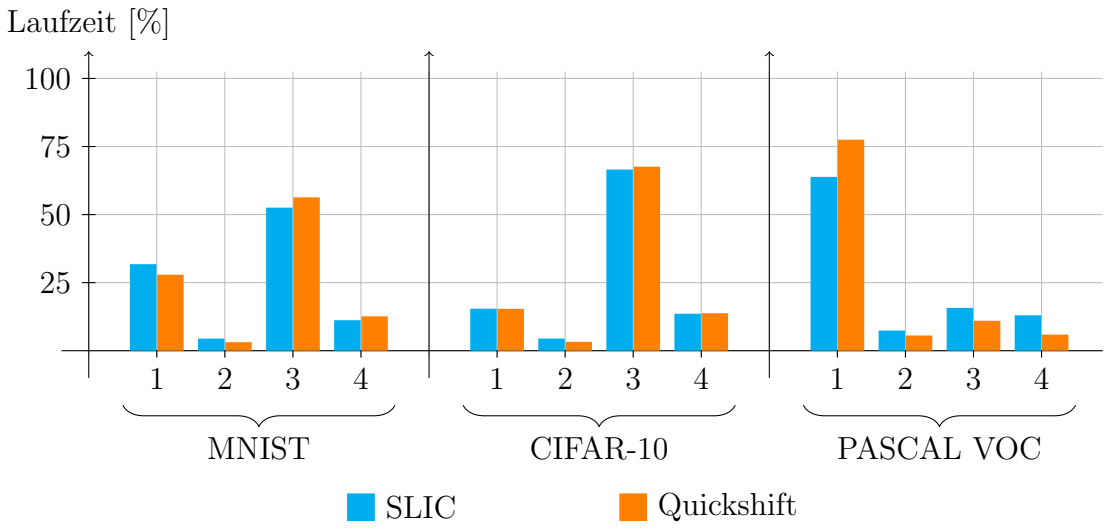
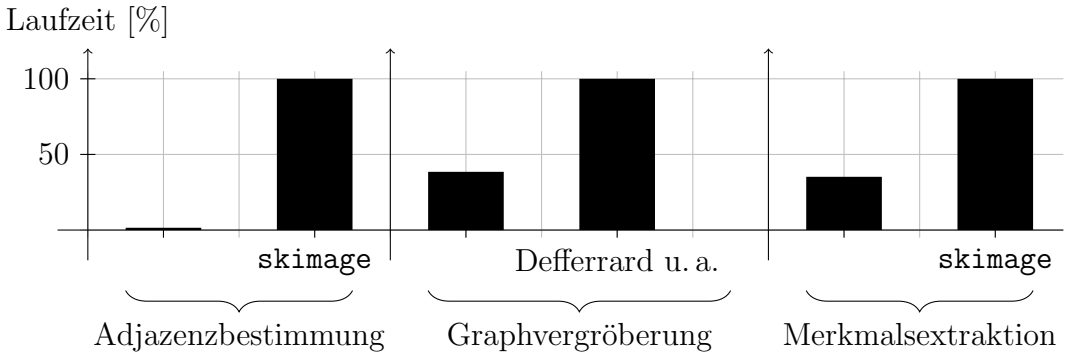


Abbildung 4.7: Laufzeitverteilung der einzelnen spektralen Vorverarbeitungsschritte für alle Datensätze und Superpixelalgorithmen. Wohingegen für MNIST und CIFAR-10 die Graphvergröberung die längste Berechnungsdauer in Anspruch nimmt, zeigt sich dagegegen für größere Datensätze wie PASCAL VOC die Segmentierung als Flaschenhals der Vorverarbeitung.



Vorverarbeitungsschritt	vorhanden [ms]	optimiert [ms]	Gewinn [%]
Adjazenzbestimmung	866	13	6535
Graphvergröberung (4 Ebenen)	87	33	160
Merkmalsextraktion	144	50	184

Abbildung 4.8: Vergleich der Laufzeiten verschiedener Vorverarbeitungsschritte bezüglich des Lernens auf Graphen zwischen bereits vorhandenen, quelloffenen Implementierungen (rechts) und ihrer jeweiligen optimierten Version (links) am Beispiel eines Bildes aus PASCAL VOC: (a) Adjazenzbestimmung einer Segmentierungsmaske, (b) Graphvergröberungsschritt inklusive der entsprechenden Anordung der Knoten zu binären Bäumen und (c) Merkmalsextraktion. In allen Bereichen konnten deutliche Laufzeitgewinne erreicht werden.

5 Ausblick

Weitere Anwendungsgebiete.

Entfernung irrelevanter Knoten. nicht nur bei MNIST auch bei PASCAL VOC slic? (zum Beispiel Regelmäßigkeiten erkennen)
Problem kanten bzw knoten mit sehr vielen kanten enthalten wenig information und sind meist hintergrundsknoten, die eine grosse gleichfarvige fläche bilden (das ist natürlich nicht immer der fall) Wir können diese knoten über einen treshhold entfernen (zb 20) Laufen jedoch gefahr isolierte knoten zu schaffen bzw einen nicht verbundenen graphen (zeige beispiel) Eine weitere idee ist für jeden knoten einen threshold an anzahl an kanten zusetzen und die längsten kanten zu entfernen. Dann bleibt aber der unnütze hintergrundknoten erhalten.
Im falle von mnist kann mn hintergrundsknoten leicht über farbe ermitteln. In anderen interessanteren datensätzen ist dies nicht so einfach.
Gibt es weitere ideen? Kann man isolierte graphen leicht ermitteln? Was tun in so einem fall?
Testen über Eigenvektoren ob Graph zusammenhängend ist Wenn nicht, dann Knoten mit nächsten verbinden.

Spatial-Pyramid-Pooling.

Attention-Algorithmus.

Glossar

T Tschebyschow-Polynom. 41, 42

ℓ Knotenfärbung $\ell\colon \mathcal{V} \to \mathcal{C} \subseteq \mathbb{R}$ eines Graphen $\mathcal{G}$. 21, 23, 24, 26, 28–30

γ Lernrate eines neuronalen Netzes. 71

$\lambda_{\max}$ Größter Eigenwert eines Eigenwertproblems $\mathbf{Mu} = \lambda\mathbf{u}$. 34, 37, 41–43

λ Eigenwert eines Eigenwertproblems $\mathbf{Mu} = \lambda\mathbf{u}$. 19, 20, 22, 34, 37–40, 65, 66, 69

$\mathbf{A}_{\text{dist}}$ Distanzadjazenzmatrix $\mathbf{A}_{\text{dist}} \in [0,1]^{N\times N}$ eines Graphen $\mathcal{G}$ im zweidimensional euklidischen Raum. 7, 8, 12, 13, 47, 48, 50, 57, 59

$\mathbf{A}_{\text{rad}}$ Winkeladjazenzmatrix $\mathbf{A}_{\text{rad}} \in [0,2\pi]^{N\times N}$ eines Graphen $\mathcal{G}$ im zweidimensional euklidischen Raum. 8, 47, 50, 51, 57, 59, 72

$\tilde{\mathbf{A}}_{\text{dist}}$ transformierte Distanzadjazenzmatrix $\tilde{\mathbf{A}}_{\text{dist}} \coloneqq \mathbf{A}_{\text{dist}} + \mathbf{I}$. 47, 48, 51

$\tilde{\mathbf{D}}_{\text{dist}}$ transformierte Gradmatrix $\tilde{\mathbf{D}}_{\text{dist}} \coloneqq \tilde{\mathbf{D}}_{\text{dist}} + \mathbf{I}$ eines Graphen $\mathcal{G}$ im zweidimensionalen euklidischen Raum. 47, 48, 51

$\mathbb{N}$ Menge der natürlichen Zahlen $\{0,1,2,\dots,\}$. 3–6, 10, 11, 13, 14, 17, 22–26, 28, 34, 37, 47, 49, 50, 53, 64

$\mathbb{R}_+$ Menge der positiven reellen Zahlen. 5, 22, 28, 34, 51, 52

$\mathbb{R}$ Menge der reellen Zahlen. 3–10, 12–16, 21, 22, 24, 26–31, 34–46, 48–51, 56, 57, 63, 64, 70, 72–74

$\mathcal{C}$ Menge der Farben $\mathcal{C} \subseteq \mathbb{R}$ einer Knotenfärbung ℓ eines Graphen $\mathcal{G}$. 21, 24, 26

$\mathcal{E}$ Kantenmenge $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ eines Graphen $\mathcal{G}$. 5–12, 22, 23, 25–28, 30, 41, 42, 44, 45, 49, 50, 54, 57

$\mathcal{G}$ Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ mit Knotenmenge $\mathcal{V}$ und Kantenmenge $\mathcal{E}$. 5–13, 17, 21–23, 26–28, 30, 35–38, 42–44, 47, 48, 50, 53–57

$\mathcal{N}$ Nachbarschaftsfunktion $\mathcal{N}_K(v_i) \coloneqq \{v_j \mid s(v_i, v_j) \leq K\}$. 5, 6, 22, 24–27, 29–31, 35, 36, 41, 44, 47, 54

$\mathcal{O}$ O-Notation. 15, 40, 42, 44, 49, 51, 55

$\mathcal{S}$ Menge von N Superpixeln $\mathcal{S} \coloneqq \{\mathcal{S}_n\}_{n=1}^{N}$ mit $\mathcal{S}_n \subset W \times H$. 10, 11, 17

$\mathcal{V}$ Knotenmenge $\mathcal{V} \coloneqq \{v_n\}_{n=1}^{N}$ eines Graphen $\mathcal{G}$. 5–11, 13, 21–26, 28–31, 35, 38, 44, 47, 49, 53–57, 81

NCut Normalized-Cut. 55

N Basisfunktionen einer B-Spline-Kurve. 49–53

ReLU Aktivierungsfunktion $\mathrm{ReLU}(\cdot) \coloneqq \max(\cdot, 0)$ eines neuronalen Netzes. 44, 56, 71

conv2d klassische Faltungsoperation auf einem regulären Gitter. 6, 23, 27, 30, 48

deg Gradfunktion $\deg(v) \coloneqq |\mathcal{N}(v)|$ auf den Knoten eines Graphen $\mathcal{G}$. 5, 22, 64, 66, 68

diag Diagonalfunktion $\mathrm{diag} \to \mathbb{R}^N \to \mathbb{R}^{N \times N}$ mit $\mathrm{diag}(\mathbf{v})_{ii} = \mathbf{v}_i$. 4, 5, 34

$\odot$ elementweises Hadamard-Produkt. 39, 40, 51

$\perp$ Orthogonalität zweier Vektoren, d.h. $\langle \mathbf{v}, \mathbf{w} \rangle = 0$. 3, 34

σ Aktivierungsfunktion eines neuronalen Netzes. 44

φ Winkelfunktion$\varphi\colon \mathcal{V} \times \mathcal{V} \to [0, 2\pi]$ auf den Kanten eines Graphen $\mathcal{G}$. 8, 29

ξ Standardabweichung der Gaußfunktion. 7, 8, 12, 15, 16, 46, 64, 66, 68, 71, 74

b B-Spline-Kurve. 49, 50

d gewichtete Gradfunktion $d(v_i) \coloneqq \sum_{j=1}^{N} \mathbf{A}_{ij}$ auf den Knoten eines Graphen $\mathcal{G}$. 5, 22, 35, 55

m Massefunktion $m\colon \mathcal{V} \to \mathbb{R}$ auf den Knoten des Graphen $\mathcal{G}$. 11, 57

p Positionsfunktion $p\colon \mathcal{V} \to \mathbb{R}^2$ auf den Knoten des Graphen $\mathcal{G}$. 7–13, 28–30, 57

s kürzeste Pfaddistanz $s\colon \mathcal{V} \times \mathcal{V} \to \mathbb{N} \cup \{\infty\}$ zweiter Knoten eines Graphen $\mathcal{G}$. 5, 6, 22, 25, 26, 37

v Knoten $v \in \mathcal{V}$ eines Graphen $\mathcal{G}$. 5–13, 21–30, 35–37, 39–41, 44, 45, 47, 50, 54, 55, 57

w Gewichtsfunktion $w\colon \mathcal{V} \times \mathcal{V} \to \mathbb{R}_+$ auf den Kanten eines Graphen $\mathcal{G}$. 5, 7, 8, 35, 36, 44, 45, 55

$\mathbf{A}$ Adjazenzmatrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ eines Graphen $\mathcal{G}$. 5, 9, 12, 22, 35, 43, 47–49

$\mathbf{B}$ Bild $\mathbf{B} \in \mathbb{R}^{H \times W \times C}$. 4, 5, 7, 9, 10, 13–16, 72

$\mathbf{D}$ Gradmatrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ eines Graphen $\mathcal{G}$ mit $\mathbf{D}_{ii} = d(v_i)$. 5, 35, 43

$\mathbf{F}$ Merkmalsmatrix $\mathbf{F} \in \mathbb{R}^{N \times M}$ auf den Knoten eines Graphen $\mathcal{G}$. 6, 7, 9, 20, 27, 30, 31, 42–44, 49, 51, 73

$\mathbf{I}$ Einheitsmatrix $\mathbf{I} \in \mathbb{R}^{N \times N}$ mit $\mathbf{I}_{ii} = 1$. 34, 35, 41, 43, 48

$\mathbf{L}$ kombinatorische Laplace-Matrix. 35–37, 39

$\mathbf{M}$ Nicht-zentrierte Momente. 17–20, 66, 69

$\mathbf{S}$ Segmentierungsmaske $\mathbf{S} \in \{1, \dots, N\}^{H \times W}$ bezüglich einer Menge von N Superpixeln. 10–12, 14, 17–19

$\mathbf{U}$ Eigenvektormatrix $\mathbf{U} \coloneqq [\mathbf{u}_1, \dots, \mathbf{u}_N] \in \mathbb{R}^{N \times N}$ einer reell symmetrischen Matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$. 34, 38–42

$\mathbf{W}$ Gewichtsmatrix oder Gewichtstensor einer neuronalen Schicht. 42–45, 48, 49, 51

$\mathbf{\Lambda}$ Diagonalmatrix der Eigenwerte $\{\lambda_n\}_{n=1}^{N}$ einer symmetrischen Matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$. 34, 40, 41

η Skaleriungsinvariante Momente. 18, 20, 65

$\mathcal{L}$ kombinatorische oder normalisierte Laplace-Matrix. 35, 37, 38, 40–43

μ Translationsinvariante Momente. 18–20, 65

$\tilde{\mathbf{A}}$ transformierte Adjazenzmatrix $\tilde{\mathbf{A}} \coloneqq \mathbf{A} + \mathbf{I}$. 43–45, 48, 49

$\tilde{\mathbf{D}}$ transformierte Diagonalmatrix $\tilde{\mathbf{D}} \coloneqq \mathbf{D} + \mathbf{I}$. 43–46

$\tilde{\mathbf{L}}$ normalisierte Laplace-Matrix $\tilde{\mathbf{L}} \coloneqq \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$. 35–37, 43

$\tilde{\mathbf{\Lambda}}$ skalierte und transformierte Diagonalmatrix der Eigenwerte $\tilde{\mathbf{\Lambda}} \coloneqq 2\Lambda/\lambda_{\max} - \mathbf{I}$. 41–43

$\tilde{\mathcal{L}}$ skalierte und transformierte Laplace-Matrix $\tilde{\mathcal{L}} \coloneqq 2\mathcal{L}/\lambda_{\max} - \mathbf{I}$. 41, 42

$\mathbf{f}$ Merkmalsvektor $\mathbf{f} \in \mathbb{R}^N$ auf den Knoten eines Graphen $\mathcal{G}$. 6, 35–44, 48, 51, 56, 74

$\mathbf{h}$ (Rotationsinvariante) Hu-Momente. 18, 20, 66, 69

$\mathbf{u}$ Eigenvektor zu einem Eigenwert λ eines Eigenwertproblems $\mathbf{Mu} = \lambda\mathbf{u}$. 34, 37–39

CIFAR Canadian Institute for Advanced Research. 65–68, 70, 74–77, 85, 87

CNN Convolutional Neural Network. 6, 21, 27, 29, 33, 40, 42, 43, 45, 46, 53, 58, 63

GCN Graph Convolutional Network. 42–46

MNIST Modified National Institute of Standards and Technology. 63–66, 70, 74, 76, 77, 79, 85, 87

PASCAL VOC Pascal Visual Object Classes. 67–70, 74, 76–79, 85, 87

PCA Hauptkomponentenanalyse. 61, 62, 85

SLIC Simple Linear Iterative Clustering. 13–16, 20, 28, 63–69, 73, 76, 77, 85, 89

SVHN Street View House Numbers. 70

SVM Support-Vector-Machine. 62

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

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

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

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

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

[7] 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

[8] 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

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

[10] 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

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

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

[13] 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

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

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

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

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

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

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

[20] 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

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

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

[23] 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)

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

[25] 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

[26] 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

[27] 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

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

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

[30] 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

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

[32] 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

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

[34] 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

[35] 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

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

[37] 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