Karlsruher Institut für Technologie

Geometric deep learning: Ein Überblick

Bachelorarbeit von

David Culley

an der Fakultät für Informatik
Institut für Visualisierung und Datenanalyse
Geometrieverarbeitung (CAGD)

Erstgutachter:	Prof. Dr. rer. nat. Hartmut Prautzsch
Zweitgutachter:	Prof. Dr.-Ing. Carsten Dachsbacher
Betreuender Mitarbeiter:	Dipl.-Inform. Maximilian Eifried

Bearbeitungszeit: 06. April 2021 – 06. August 2021

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 06. August 2021

. . . . . .

(David Culley)

Zusammenfassung

In dem sich in rapidem Tempo entwickelnden Gebiet des maschinellen Lernens mittels künstlicher neuronaler Netzwerke werden Jahr für Jahr bahnbrechende Erfolge erzielt. Dies gilt insbesondere im Fall euklidischer Daten wie Bildsignale oder sequentielle Daten, beispielsweise Text in natürlicher Sprache, Audiosignale, Zeitreihendaten oder Genomsequenzen. Die Verarbeitung nichteuklidischer geometrischer Datenstrukturen wie Graphen, Mannigfaltigkeiten, Polygonnetze oder Punktwolken hingegen ist weniger erforscht. Da eine Vielzahl von Problemstellungen sich mittels Graphen beschreiben lässt, eröffnen die Methoden des aufsteigenden Teilgebiets namens *geometric deep learning* nicht nur für die Computergrafik und das maschinelle Sehen ungeahnte Möglichkeiten. Aufgabe dieser Arbeit ist es, einen umfassenden Überblick über diese junge Teildisziplin und ihre Techniken zu verschaffen. Ziel dabei ist es, auch unerfahrenen Lesern einen zugänglichen Einstieg in diesen Themenbereich zu präsentieren. Zu diesem Zweck werden unterschiedliche Ansätze zur Verallgemeinerung der klassischen Faltung auf nichteuklidische Gebiete behandelt. Darüber hinaus werden am Beispiel verschiedener Anwendungsfälle konkrete neuronale Netzwerkmodelle vorgestellt.

Abstract

In the rapidly developing field of deep learning, breakthroughs are achieved every year. That is particularly true in the case of Euclidean data such as image signals or sequential data like natural language text, audio signals, time series, or genome sequences. The processing of non-Euclidean geometric data structures such as graphs, manifolds, polygon meshes, or point clouds is, in comparison, less studied. Since a wealth of problems may be described using graphs, the methods of the emerging subfield named geometric deep learning open up unimagined possibilities not only regarding computer graphics and vision applications. The objective of this work is to provide a comprehensive overview of this young subdiscipline and its techniques. Its goal is to present even an inexperienced reader with an accessible entry point to this subject area. For that purpose, different approaches to generalize the classic convolution to non-Euclidean domains are covered. Moreover, concrete neural network models are introduced using the example of various use cases.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Geometric deep learning ist ein Sammelbegriff für derzeit neu entstehende Techniken zur Verallgemeinerung verschiedener Modelle künstlicher neuronaler Netzwerke mit dem Ziel, neuronale Netzwerkmodelle auch auf solche Daten anwendbar zu machen, deren zugrundeliegende Struktur kein euklidischer Raum $\mathbb{R}^n$ ist, womit typischerweise Graphen oder Mannigfaltigkeiten gemeint sind.

Unter der Bezeichnung *deep learning* versteht man eine spezielle Art des maschinellen Lernens (*machine learning*) mittels künstlicher neuronaler Netzwerke. Maschinelles Lernen ist nach Tom Mitchell [Mit97] wie folgt definiert:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Maschinelles Lernen kann nicht nur dazu genutzt werden, eine Abbildung von einer gewissen Repräsentation der Eingabedaten auf die Ausgabe zu lernen, sondern auch um die Repräsentation an sich zu lernen. Diesen Ansatz des maschinellen Lernens nennt man *representation learning* oder auch *feature learning*. Mit *deep learning* sind nun auf künstlichen neuronalen Netzwerken basierende Methoden des maschinellen Lernens gemeint, die diesen Ansatz des *representation learning* verfolgen.

Goodfellow, Bengio und Courville [GBC16] verstehen unter *deep learning* entgegen der landläufigen Meinung nicht etwa neuronale Netzwerkarchitekturen mit besonders vielen versteckten Schichten, sondern eine Form des *representation learning*, bei der die gesuchte Repräsentation aus einfacheren anderen Repräsentationen oder auch *features* zusammengesetzt wird, was es ermöglicht, komplexe Konzepte aus simpleren Konzepten aufzubauen. Beispielsweise setzt sich ein Bild einer Person aus einfacheren Konzepten wie etwa Ecken und Umrissen zusammen, welche selbst wiederum aus noch einfacheren Konzepten wie Kanten aufgebaut sind. Diesem hierarchischen Ansatz folgen auch die sogenannten Faltungsnetzwerke (*convolutional neural networks*), welche sich am biologischen visuellen Cortex [HW62] orientieren. Einer der frühesten Vertreter der Faltungsnetzwerke ist *LeNet-5* [LeC+89], welches im Jahr 1989 von LeCun u. a. vorgestellt wurde. Über die *features* werden beim *deep learning* die dem Anwendungsfall entsprechenden Regelmäßigkeiten oder auch Gesetzmäßigkeiten (*regularities*) erfasst. Der Lernprozess eines solchen künstlichen neuronalen Netzwerks – auch als dessen Training bezeichnet – ist durch die Anpassung der Parameter des Netzwerks realisiert, wobei die Genauigkeit der geschätzten Ergebnisse iterativ verbessert wird, indem mittels einer Form des Gradientenverfahrens (*gradient descent*) eine Kostenfunktion minimiert wird. Der Lernprozess gilt als abgeschlossen, sobald die Betrachtung weiterer Muster die Fehlerrate des Modells nicht länger in sinnvoller Weise reduziert.

Bei der Teildisziplin des *geometric deep learning* werden diese Gesetzmäßigkeiten durch geometrische Leitsätze freigelegt. Für einen zugänglichen Einstieg in dieses Themenfeld soll zunächst die Hintergrundgeschichte des Begriffs *geometric deep learning* geklärt werden. Über 2000 Jahre lang kannte man einzig die aus der Schulmathematik vertraute euklidische Geometrie, in welcher für die zeichnerische Darstellung einer Figur nur Zirkel und Lineal verwendet werden dürfen und der Umfang eines Kreises mit Radius r immer gleich $2\pi r$ ist. Im 19. Jahrhundert begannen Mathematiker jedoch mit der Konstruktion verschiedener neuer Geometrien, unter anderem auch nichteuklidischer Geometrien, nachdem es ihnen jahrhundertelang nicht gelungen war das Parallelenpostulat der euklidischen Geometrie zu beweisen und ihnen der Gedanke kam, dass es sich bei der Geometrie des realen Raumes, mit der die Wirklichkeit wiedergegeben werden kann, möglicherweise nicht wie bislang angenommen um die euklidische Geometrie handelt. Ab dem Jahr 1826 stellten mehrere Mathematiker ihre jeweilige Konstruktion einer nichteuklidischen Geometrie vor, etwa Riemann (elliptische Geometrie), Lobachevsky, Bolyai und Gauß, wobei die letzteren drei unabhängig voneinander dieselbe nichteuklidische Geometrie (hyperbolische Geometrie) entdeckten. Mit der dazu parallelen Entwicklung der projektiven Geometrie, einer Erweiterung der euklidischen Geometrie, sowie einigen weiteren Geometrien, entstand im Laufe des 19. Jahrhunderts im übertragenen Sinne ein ausgemachter Zoo von Geometrien. Die zeitgenössischen Mathematiker stritten sich darüber, welche Geometrie die einzig wahre den realen Raum beschreibende Geometrie sei und was eine Geometrie überhaupt definiert. Im Jahr 1872 legte Felix Klein, einer der bedeutendsten Mathematiker des 19. Jahrhunderts, anlässlich seiner Berufung zur Professur eine Programmschrift vor, die später als *Erlanger Programm* bekannt wurde, und in der er einen Weg vorschlug wie die unterschiedlichen Geometrien in Beziehung zueinander gesetzt und in einer Hierarchie angeordnet werden können. Die Ideen von Klein verschafften endlich Klarheit darüber was eine Geometrie ausmacht, da Klein zeigte, dass eine Geometrie durch eine geeignete Wahl von Symmetrietransformationen definiert werden kann. Unter einer Symmetrietransformation versteht man in der Physik eine Transformation, die den Zustand eines Systems nicht verändert. Eine Symmetrie ist die Eigenschaft des Systems, nach einer bestimmten Transformation unverändert zu bleiben, also invariant unter dieser Transformation zu sein. Beispiele für Symmetrien sind die Rotationsinvarianz, Translationsinvarianz oder Spiegelinvarianz. Formalisiert werden Symmetrietransformationen mittels der Gruppentheorie. Klein schlug vor, dass man das Studium von Geometrien auffassen sollte als das Studium von Symmetrien oder, in anderen Worten, Invarianzen unter gewissen Transformationen.

Die Arbeit von Klein war enorm profund und hatte auch auf die Physik große Auswirkungen. Beispielsweise spielt die hyperbolische Geometrie, eine der nichteuklidischen Geometrien, eine wichtige Rolle in der speziellen Relativitätstheorie. Dank den Beiträgen von Klein konnte Emmy Noether im Jahr 1918 beweisen, dass jede kontinuierliche Symmetrie der Wirkung einen Erhaltungssatz zur Folge hat und umgekehrt, dass zu jedem Erhaltungssatz eine kontinuierliche Symmetrie der Wirkung gehört. Anders ausgedrückt konnte Noether dank Klein unter anderem beweisen, dass aus der Invarianz des Beginns der Zeit unter Translation folgt, dass Energie eine Invariante ist und dass Energie somit weder aus dem Nichts entstehen kann, noch dass sie verloren gehen kann, sondern lediglich in eine andere Form von Energie umgewandelt wird. Dank dem enorm bedeutenden Noether-Theorem konnten, von der Gravitation abgesehen, die fundamentalen Wechsel-

wirkungen vereinheitlicht werden, was im Jahr 1975 schließlich zum Standardmodell der Teilchenphysik führte, welches unser heutiges Verständnis der Physik zusammenfasst.

Der Begriff *geometric deep learning* wird häufig synonym für den Einsatz von *graph neural networks* verwendet. *Geometric deep learning* ist jedoch weit mehr als das. Der aktuelle Stand der Wissenschaft im Bereich *deep learning* weist starke Parallelen zur Situation im 19. Jahrhundert auf, als die zahlreichen neu aufgekommenen Geometrien noch nicht vereinheitlicht waren. Derzeit gibt es einen Zoo unterschiedlicher Netzwerkarchitekturen wie etwa Faltungsnetzwerke (*convolutional neural networks*), rekurrente neuronale Netzwerke (*recurrent neural networks*), *transformers*, *graph neural networks* oder *deep sets* für verschiedene Typen von Daten, aber kaum vereinheitlichende Leitsätze. Mangels vereinheitlichender Prinzipien ist es schwierig die Zusammenhänge zwischen den Methoden zu verstehen, weshalb unnötigerweise dieselben Konzepte unter anderem Namen neu entwickelt werden. Das selbst gesteckte Ziel des federführenden Autors der wegweisenden Arbeit von Bronstein u. a. [Bro+17b], durch die der in [Eur17] geprägte Begriff *geometric deep learning* Verbreitung fand, ist eine Vereinheitlichung der Netzwerkarchitekturen mit den Mitteln der Geometrie nach dem Vorbild des *Erlanger Programms* von Felix Klein. Eine solche Vereinheitlichung dient zwei Zwecken: einerseits soll ein gemeinsames mathematischen Rahmenwerk bereitgestellt werden, aus dem sich alle heutigen Netzwerkarchitekturen ableiten lassen und andererseits soll ein Prozedere entwickelt werden, mittels dem zukünftige Architekturen auf von Grundsätzen geleitete Weise entworfen werden können.

> „Die Kenntnis bestimmter Prinzipien kann leicht die Kenntnis bestimmter Tatsachen ersetzen."
>
> — *Claude Adrien Helvétius (1715–1771), französischer Philosoph*

Eine der großen Schwierigkeiten bei solch einer geometrischen Vereinheitlichung der verschiedenen Netzwerkarchitekturen ist die Verallgemeinerung derjenigen Operationen, aus denen die Modelle neuronaler Netzwerke sich zusammensetzen, etwa die Faltung (*convolution*) oder das *pooling*. Zweck der Verallgemeinerung dieser fundamentalen Komponenten eines neuronalen Netzwerkmodells ist, das Modell nicht nur auf Daten mit euklidischer Struktur, sondern auch auf Daten nichteuklidischer Natur wie Graphen oder Mannigfaltigkeiten anwenden zu können. Einem Bildsignal liegt ein zweidimensionales reguläres Gitter zugrunde, das Pixelgitter, weshalb Operationen wie die klassische Faltung sich ohne Schwierigkeiten auf solche Daten anwenden lassen. Geometrie hat jedoch üblicherweise keine reguläre Gitterstruktur oder überhaupt eine Gitterstruktur, was es erforderlich macht, die als selbstverständlich gegeben erachteten Grundbausteine künstlicher neuronaler Netzwerke in einer allgemeineren Form neu zu definieren.

Im Rahmen dieser Arbeit soll eine leicht zugängliche Einführung in diese Problematik dargelegt werden. Dazu soll ein umfassender Einblick in das noch junge Feld des *geometric deep learning* geboten, ein Überblick über die verschiedenen Ansätze zur Verallgemeinerung der Faltung verschafft sowie ausgewählte Verfahren näher vorgestellt werden. Die Architekturen aktueller *graph neural networks* fallen in eine von drei Kategorien, je nachdem auf welche Weise in einer Schicht des *graph neural network* die *feature vectors* der einzelnen zu einer Nachbarschaft gehörigen Knoten eines Graphen aggregiert werden: basierend auf Faltung [DBV16; KW16; Wu+19], basierend auf *attention* [Mon+17; Vel+18]

oder basierend auf *message passing* [Gil+17; Bat+18; Wan+19]. Der Entwurf solcher Ansätze zur Aggregation der *feature vectors* einer Nachbarschaft ist ein aktueller Gegenstand der Forschung. Die Techniken basierend auf *attention* beheben einige der Defizite der früheren, auf der Faltung basierenden Methoden. Die Verfahren basierend auf *message passing* sind von den drei Kategorien die allgemeinsten Verfahren. Um ein solides Grundwissen zu vermitteln, liegt der hauptsächliche Fokus dieser Arbeit in den Grundlagenkapiteln auf den im Vergleich frühen Methoden aus der Kategorie der faltungsbasierten Modelle, wohingegen in den späteren Kapiteln dieser Arbeit auch Modelle der anderen beiden Kategorien vorgestellt werden. [Bro+21a; Bro21a]

1.1. Geschichte des *geometric deep learning*

Einer der ersten Ansätze, künstliche neuronale Netzwerke auf Graphen zu verallgemeinern, wird Gori, Monfardini und Scarselli [GMS05] zugeschrieben und wurde im Jahr 2005 vorgeschlagen, fand jedoch kaum Beachtung und geriet in Vergessenheit. Scarselli u. a. [Sca+09] veröffentlichten im Jahr 2009 eine weitere Publikation, die zunächst wenig beachtet wurde, sich später aber noch als bedeutend erweisen sollte, als das Interesse an *deep learning* ab dem Jahr 2012, ausgelöst durch die beeindruckenden Ergebnisse von *AlexNet* [KSH12], zu steigen begann. Die ersten Modellierungen von *graph neural networks* setzten noch auf rekurrente neuronale Netzwerke (*recurrent neural networks*) und wurden, nachdem die Veröffentlichungen [GMS05; Sca+09] wiederentdeckt wurden, von Li u. a. [Li+16] durch *gated recurrent units* verbessert. Die Arbeit [Li+16] diente wiederum als Ausgangspunkt für [Gil+17], worin Gilmer u. a. im Jahr 2017 das Rahmenwerk *Message Passing Neural Networks* vorstellen.

Nachdem die ersten *graph neural networks* noch auf rekurrenten neuronalen Netzwerken aufbauten, folgte relativ bald ein Wechsel zu faltungsbasierten Formulierungen von *graph neural networks*. Der erste Vertreter der die Faltung verallgemeinernden *graph neural networks* wurde im Jahr 2014 in [Bru+14] präsentiert. Dieses frühe Verfahren von Bruna u. a. gehört zu den Spektralmethoden, welche in Abschnitt 3.2 näher vorgestellt werden. So bedeutend die Spektralmethoden auch waren für die Entwicklung von *geometric deep learning*, so hatten sie dennoch erhebliche Nachteile wie unter anderem eine beträchtliche Rechenkomplexität. Einige Nachteile der Spektralmethoden konnten mit den darauf folgenden Spektrum-freien Methoden behoben werden, welche in Abschnitt 3.3 ausführlich behandelt werden. Zu den Spektrum-freien Methoden gehören unter anderem [DBV16; KW16; MBB17; MOB18], welche alle auf polynomielle spektrale Filter setzen sowie [Lev+19], welche stattdessen auf rationale spektrale Filter setzt.

Parallel zu den Spektrum-freien Methoden entwickelten sich die kartografierenden Methoden (*charting-based methods*) zur Verallgemeinerung der Faltung, welche in Abschnitt 3.4 näher vorgestellt werden. Diese Verfahren haben ihren Ursprung in den Bereichen der Computergrafik und des maschinellen Sehens, wo die verallgemeinerten Faltungsnetzwerke nicht auf Graphen, sondern auf Netzflächen (*meshed surfaces*) angewendet werden sollen. Ein Vorreiter dieser Art von Verfahren ist [Mas+15a] von Masci u. a. aus dem Jahr 2015, auf den [Bos+16b] und später Gaußsche Mischmodelle (*Gaussian mixture models*) [Mon+17; Vel+18] folgten. [Bro+17b; Wan+19]

1.2. Anwendungsfälle von *geometric deep learning*

Viele Problembeschreibungen lassen sich als Graph formulieren, beispielsweise die Beziehungen zwischen Kunden und Produkten eines Anbieters oder die chemischen Bindungen zwischen den Atomen eines Moleküls. *Graph neural networks* eignen sich daher hervorragend für unter anderem Empfehlungsdienste (*recommender systems*) wie sie etwa von Unternehmen wie Netflix, Facebook, Amazon, oder Uber Eats eingesetzt werden. Sie eignen sich auch vorzüglich zur Erforschung neuer Medikamente, etwa beim Wirkstoffdesign mittels *virtual screening* oder bei der Prüfung auf Toxizität zum Nachweis der Unbedenklichkeit neuer Wirkstoffe. Weiterhin können mittels *graph neural networks* jegliche Formen von Netzwerken analysiert werden, seien es Zitationsnetzwerke, biologische neuronale Netzwerke im Gehirn oder soziale Netzwerke. Dadurch können nicht nur Forschungsarbeiten klassifiziert werden, sondern auch in den Neurowissenschaften durch Erkennung von Funktionsstörungen oder abnormaler Muster in der Konnektivität des Gehirns Hinweise auf Autismus wahrgenommen werden und das zunehmende Problem der absichtlichen manipulativen Verbreitung von Falschinformation (sogenannte *fake news*) bekämpft werden. Insbesondere in den Gebieten der Computergrafik und des maschinellen Sehens (*computer vision*) gibt es zudem auch zahlreiche Anwendungszwecke für das Lernen auf Punktwolken und Mannigfaltigkeiten, wobei es sich um geometrische Datenstrukturen handelt, welche als Graph interpretiert werden können.

Matrizen können ebenfalls als spezielle Ausprägung eines Graphen aufgefasst werden, der allerdings eine zweidimensionale Gitterstruktur hat. Beispielsweise können die durch einen Graphen gegebenen Informationen zu einem gewissen Teil als ein Bild enkodiert werden. Indem ein Graph zunächst in eine Matrix überführt wird, können somit auch klassische euklidische Methoden zur Lösung von durch Graphen beschriebener Problemstellungen eingesetzt werden. Jedoch geht bei der Überführung von nichteuklidischen Daten wie einem Graph in euklidische Daten wie eine Matrix zwangsläufig Information verloren, was nicht förderlich für die Qualität der Vorhersageergebnisse ist. Die direkte Verarbeitung von nichteuklidischen Gebieten wie Graphen oder Mannigfaltigkeiten ermöglicht weitaus bessere Ergebnisse, ist daher wünschenswert und hat das Potential dazu, nicht nur jedes der genannten Anwendungsgebiete, sondern noch viele mehr zu revolutionieren. [Bro+21a; Bro21a]

1.3. Aufbau dieser Arbeit

Die vorliegende Arbeit ist in sieben Kapitel unterteilt. Bei Kapitel 1 handelt es sich um diese Einleitung. In Kapitel 2 werden zum besseren Verständnis dieser Arbeit die Grundlagen der Verallgemeinerung der Faltung auf nichteuklidische Gebiete aufbereitet. In Kapitel 3 werden die vier verschiedenen Ansätze zur besagten Verallgemeinerung der Faltung erläutert. In den Kapiteln 4, 5 und 6 wird jeweils ein konkreter Anwendungsfall von *geometric deep learning* vorgestellt und zu diesem Zwecke jeweils ein ausgewähltes aktuelles Verfahren behandelt. Dabei wird im Detail darauf eingegangen wie dieses Verfahren die Faltung verallgemeinert und auf diese Weise der Bezug zu Kapitel 3 hergestellt. In Kapitel 7 werden abschließend die Punkte dieser Arbeit zusammengefasst und ein Fazit gezogen.

2. Grundlagen

In diesem Kapitel werden die für das Verständnis der in den späteren Kapiteln beschriebenen Ansätze und Methoden nötigen Grundlagen erklärt. Zunächst wird, nach der Definition des topologischen Raums, der zentrale Begriff des *nichteuklidischen Gebiets* geklärt und die klassische Faltung präsentiert. Anschließend werden typischerweise betrachtete Ausprägungen nichteuklidischer Gebiete vorgestellt, namentlich Mannigfaltigkeiten und Graphen. Auch werden weitere fundamentale Konzepte und Begriffe vorgestellt, unter anderem Symmetrien im Sinne der Physik wie beispielsweise die Translationsinvarianz, Funktionenräume und Hilberträume, und weiterhin auch die Fourier-Analysis und das Faltungstheorem sowie die für die Spektralanalyse wichtigen Eigenfunktionen des Laplace-Beltrami-Operators aus der Operatortheorie.

2.1. Topologie und Topologischer Raum

Vorbereitend wird in diesem Abschnitt der Begriff des topologischen Raums definiert. Einen topologischen Raum erhält man, indem man eine Grundmenge mit einer topologischen Struktur versieht, was bedeuten soll, dass man eine Abstraktion von Nähe und andere intuitive Lagebeziehungen wie „Streben gegen“ einführt. Die topologische Struktur einer Menge wird auch als die Geometrie der Lage bezeichnet und gibt an wie die Elemente der Grundmenge räumlich zueinander in Beziehung stehen. Sie ermöglicht Aussagen darüber zu treffen, mit welchen Punkten der Grundmenge ein Punkt benachbart ist, muss allerdings nicht notwendigerweise auch Aussagen darüber ermöglichen, wie groß der Abstand zwischen diesen Punkten ist.

Definiert wird die Struktur von einer Topologie, wohingegen sie von der Menge getragen wird. Formal ist ein topologischer Raum ein Paar (X, T) bestehend aus einer Grundmenge X und einer Topologie T. Eine Topologie T auf einer Menge X ist ein nichtleeres System von Teilmengen von X, für das beliebige Vereinigungen und endliche Durchschnitte von solchen Mengen wieder zu dem System gehören. Die Mengen in T nennt man offen in X.

Ein topologischer Raum ist die allgemeinste Form eines Raums, der die Definition von Grenzwerten, Stetigkeit und Zusammenhang zulässt. Ermöglicht werden diese Konzepte von der Topologie. Zusätzlich zur Topologie kann eine Grundmenge X weitere Strukturen, Eigenschaften oder Einschränkungen haben. Ein Beispiel für solch einen spezielleren Raum ist der bekannte euklidische Raum $\mathbb{R}^n$. Ein anderes Beispiel für spezielle topologische Räume sind die in Abschnitt 2.9 vorgestellten Graphen oder die in Abschnitt 2.8 vorgestellten Mannigfaltigkeiten.

2.2. Nichteuklidische Gebiete

Ein zentrales Ziel im noch jungen Feld des *geometric deep learning* ist die Verallgemeinerung der für euklidische Gebiete (*Euclidean domains*) entworfenen klassischen Faltungsnetzwerke (*convolutional neural networks*) sowie ihrer Techniken und Konstrukte auf nichteuklidische Gebiete (*non-Euclidean domains*). Als Gebiet Ω (*domain*) bezeichnet man in den Teildisziplinen Topologie und Analysis der Mathematik eine offene, nichtleere und zusammenhängende Teilmenge eines topologischen Raumes.

Mit dem Begriff *nichteuklidisches Gebiet* ist in dieser Arbeit typischerweise ein Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ oder eine Mannigfaltigkeit $\mathcal{X}$ gemeint, welche in späteren Abschnitten noch näher vorgestellt werden. Den ebenfalls in dieser Arbeit betrachteten Punktwolken fehlt die topologische Struktur – eine Punktwolke ist einfach nur eine Grundmenge von Punkten ohne weitere Struktur – was wie in Kapitel 5 noch zu lesen sein wird, Schwierigkeiten verursacht. Obwohl Graphen und Mannigfaltigkeiten in sehr unterschiedlichen Teilbereichen der Mathematik auftreten – der Graphentheorie beziehungsweise Differentialgeometrie – haben sie doch mehrere Eigenheiten gemeinsam, was sich darin äußert, dass beispielsweise der später vorgestellte *graph Laplacian* oder das innere Produkt für Graphen die diskreten Analoga des *Laplacian* beziehungsweise inneren Produkts für Mannigfaltigkeiten sind.

Ein Gebiet Ω dient als Definitionsbereich eines Funktionenraums, wie in Abbildung 2.1 auf der nächsten Seite veranschaulicht. Ein bekannter Funktionenraum ist der Raum $C(\mathbb{R})$ bestehend aus den stetigen Funktionen $f\colon \mathbb{R} \to \mathbb{R}$. Ähnlich wie die Elemente eines Vektorraums Vektoren sind, sind die Elemente eines Funktionenraums Funktionen. Auch hat ein Funktionenraum, wie ein Vektorraum, eine Basis. Im euklidischen Fall ist das etwa die Fourier-Basis, welche in dieser Arbeit eine wichtige Rolle spielt und noch vorgestellt wird. Im nichteuklidischen Fall sind ein Analogon zu den Fourier-Basisfunktionen die in Abschnitt 2.14 vorgestellten Eigenfunktionen des *Laplacian* beziehungsweise *graph Laplacian*.

Beispielsweise ist im Fall von Bildern für den Definitionsbereich des Pixelgitters, die Ebene $\mathbb{R}^2$ beziehungsweise $\mathbb{Z}^2$, ein Funktionenraum $C(\mathbb{R}^2)$ definiert, dessen Elemente die stetigen Funktionen sind – die möglichen Bildsignale. Ein solches Bildsignal $f\colon \mathbb{R}^2 \to \mathbb{R}$ ordnet einem Pixel mit den Koordinaten (x, y) eine Pixelintensität zu. Diese Funktion f, also das Bildsignal, kann als Linearkombination der Fourier-Basis – verschiedene sinus- und kosinusförmige Funktionen – aufgefasst werden oder umgekehrt, über die Fourierreihe in die Fourier-Basis des Funktionenraums zerlegt werden, was in Unterabschnitt 2.14.2 noch näher beschrieben wird.

Die Funktionenräume $L^2(\Omega)$ bestehend aus den in Unterabschnitt 2.10.2 definierten quadratintegrierbaren Funktionen, die in dieser Arbeit thematisiert werden, werden in Abschnitt 2.10 noch näher vorgestellt.

Klassische künstliche neuronale Netzwerke wie beispielsweise die Faltungsnetzwerke arbeiten ausschließlich mit Funktionen, die auf euklidischen Gebieten $\Omega = \mathbb{R}^n$ definiert wurden, etwa mit Bildern, also auf kartesischen Gittern diskretisierte Bildsignale, oder mit Vektoren. Denn klassische Faltungsnetzwerke setzen voraus und nutzen auch aus, dass für die verarbeitenden Daten die euklidische Geometrie gilt. Dies bedeutet unter anderem, dass es möglich sein muss, Vektoren voneinander zu subtrahieren, Vektoren mit Matrizen zu multiplizieren oder das Skalarprodukt zweier Vektoren zu berechnen. Es muss also eine

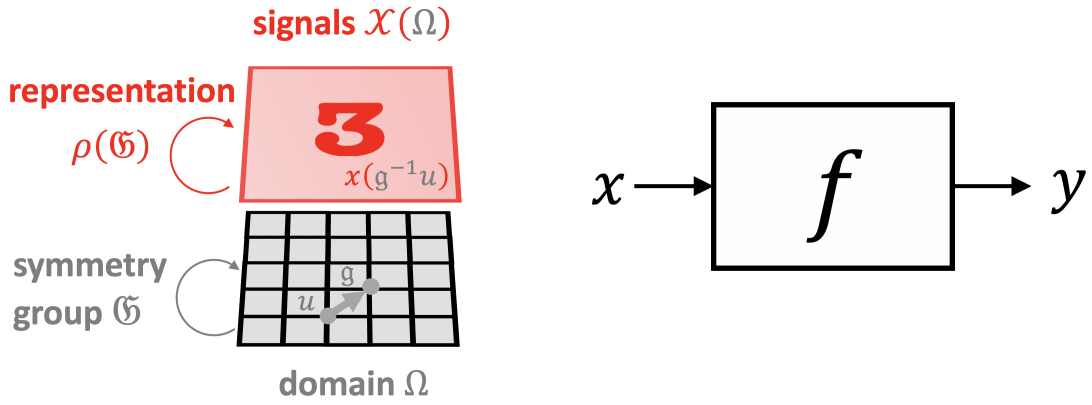


Abbildung 2.1.: **Veranschaulichung eines auf einem Gebiet Ω definierten Signals $x \in \mathcal{X}(\Omega)$.** Auf einem Gebiet Ω – am Beispiel von Bildern ein euklidischer Raum $\mathbb{R}^2$ beziehungsweise ein kartesisches Gitter – sind Signale definiert. In diesem Beispiel sind das die auf dem kartesischen Gitter diskretisierten Bildsignale. Umgekehrt liegt dem Raum der Signale, hier mit $\mathcal{X}(\Omega)$ bezeichnet, ein Gebiet Ω zugrunde. Auf einem konkreten Signal $x \in \mathcal{X}(\Omega)$, in diesem Beispiel ein Bild, operiert nun eine Funktion f, etwa ein Bildklassifikator. Vorweggreifend lässt sich sagen, dass eine Symmetriegruppe, etwa die Gruppe $\mathfrak{G}$ bestehend aus der Menge $\mathfrak{G}$ aller Translationen $\mathfrak{g} \in \mathfrak{G}$, über ihre Gruppenrepräsentation, hier der *shift operator* $\rho(\mathfrak{g})$, im Raum der Signale operiert. Indem man Annahmen darüber trifft, wie beispielsweise ein Bildklassifikator f mit der Symmetriegruppe interagiert, lässt sich die Menge der möglichen Bildklassifikatoren (*hypothesis class*) einschränken. Dies wird im Laufe dieser Arbeit noch klarer. [Bro+21a]

Vektorraumstruktur vorhanden sein, was wie in Abschnitt 2.11 auf Seite 26 angeführt bei nichteuklidischen Gebieten nicht der Fall ist.

Beim *geometric deep learning* werden künstliche neuronale Netzwerkarchitekturen entworfen, die auch mit nichteuklidischen Gebieten $\Omega \neq \mathbb{R}^n$ umgehen können. Dazu müssen elementare und im euklidischen Fall als selbstverständlich gegeben erachtete Operationen auf nichteuklidischen Gebieten neu definiert werden, um die von euklidischen Gebieten bekannte Konzepte wie etwa die Faltung (*convolution*) oder *pooling* auf nichteuklidische Gebiete übertragen zu können. Beispielsweise werden für Graphen und Mannigfaltigkeiten verallgemeinerte, auch als inneres Produkt bezeichnete Skalarprodukte definiert, die Infinitesimalrechnung auf den nichteuklidischen Gebieten ermöglichen, was schließlich die Definition eines *Laplacian* genannten Differentialoperators (siehe Unterabschnitt 2.12.2) im Fall von Mannigfaltigkeiten beziehungsweise dessen diskretes Analogon *graph Laplacian* im Fall von Graphen (siehe Unterabschnitt 2.12.3) erlaubt, welcher von zentraler Wichtigkeit für die verallgemeinerte Definition der auf nichteuklidischen Gebieten nicht existenten Fourier-Basis und einer verallgemeinerten Faltung ist.

Euklidische und nichteuklidische Geometrie

Die von Euklid im 3. Jahrhundert v. Chr. vorgeschlagene Geometrie war bis zur Vorstellung der ersten nichteuklidischen Geometrie im Jahr 1826 durch Lobachevsky die einzig bekannte Geometrie. Euklid legte in seinem Werk *Elemente* neben Definitionen und Axiomen auch fünf Postulate fest. Axiome sind nach Aristoteles allgemeine und unbezweifelbare Grundsätze, Postulate sind nach Aristoteles Grundsätze, die akzeptiert oder abgelehnt werden können. Nicht nur Euklid war davon überzeugt, dass seine Axiome und Postulate die Wirklichkeit wiedergeben, sondern auch die Mathematiker bis ins 19. Jahrhundert. Da diese Axiome und Postulate, vom Parallelenpostulat abgesehen, so selbstverständlich erschienen und als offensichtlich wahr angenommen wurden, kam man über 2000 Jahre lang gar nicht auf die Idee, dass möglicherweise die Wirklichkeit nicht mit der euklidischen Geometrie beschrieben werden kann und dass dazu eine andere Geometrie nötig sein könnte.

Dies begann sich erst zu ändern, nachdem man jahrhundertelang am Beweis des Parallelenpostulats gescheitert war. Einer der Beweisversuche war, das Parallelenpostulat in der Absicht einen Widerspruch herbeizuführen wegzulassen, was nie erfolgreich war. In anderen Versuchen ließ man das Parallelenpostulat weg, führte dafür aber ersatzweise ein neues Postulat ein, und erwartete logische Brüche. Entgegen der Erwartung führte das Ersetzen des umstrittenen Parallelenpostulats durch andere Postulate zu logisch einwandfreien geometrischen Systemen, eben den verschiedenen heute bekannten nichteuklidischen Geometrien. Nichteuklidische Geometrien unterscheiden sich von der euklidischen Geometrie somit dadurch, dass in ihnen das Parallelenpostulat nicht gilt.

Die Vorstellung verschiedener nichteuklidischer Geometrien hatte eine tiefe Krise in den Wissenschaften zur Folge, da nach zwei Jahrtausenden der euklidischen Geometrie nun im Abstand von nur wenigen Jahren mehrere konkurrierende und einander widersprechende Visionen unantastbarer wissenschaftlicher Wahrheit folgten. Die Frage, ob man den realen Raum mit einer nichteuklidischen Geometrie beschreiben kann, wird heutzutage unterschiedlich beantwortet. In jedem Fall spielen nichteuklidische Geometrien

eine zentrale Rolle bei der Beschreibung der Realität des Weltalls in der theoretischen Physik. Aus diesem Grund ist auch die Arbeit von Felix Klein, die er im Jahr 1872 im Alter von nur 23 Jahren in seiner *Erlanger Programm* genannten Programmschrift darlegte, von so weitreichender Bedeutung.

2.3. Gruppentheorie

Der Begriff der Gruppe ist wichtig für die in Unterabschnitt 2.6.1 beschriebene Translationsinvarianz (*shift invariance*). Eine Gruppe $(G, *)$ ist ein Paar bestehend aus einer Menge G, beispielsweise der Menge der Drehungen oder die Menge der Translationen, und einer Verknüpfung $*$, die zwei Elementen $g, h \in G$ ein drittes in der Menge G liegendes Element zuordnet. Diese Zuordnung muss die drei Gruppenaxiome erfüllen, namentlich das Assoziativgesetz, die Existenz eines neutralen Elements und die Existenz von inversen Elementen. Eine der bekanntesten Gruppen ist die Menge $\mathbb{Z}$ der ganzen Zahlen mit der Addition + als Verknüpfung.

Eine weitere wichtige Gruppe ist die euklidische Gruppe, die auch die Bewegungsgruppe genannt wird und deren Menge sämtliche Drehungen, Translationen und Spiegelungen des euklidischen Punktraums $\mathbb{E}^n$ enthält. Sie ist die Gruppe der Isometrien, also der abstandserhaltenden Abbildungen eines euklidischen Punktraums auf sich selbst. Die euklidische Gruppe wird mit $\mathbb{E}(n)$ oder auch $\mathrm{Iso}(n)$ bezeichnet. Diese die Metrik erhaltenden Abbildungen (daher der Name *Isometrie*) werden auch euklidische Abbildungen genannt.

2.4. Symmetrie und Invarianz

In der Physik versteht man unter einer Symmetrie die Eigenschaft eines Systems, bei einer bestimmten Transformation unverändert zu bleiben, also invariant unter der Transformation zu sein. Beispiele für Symmetrien sind die Rotationsinvarianz (*rotational symmetry*), Spiegelinvarianz (*reflection symmetry*), Translationsinvarianz (*translational symmetry*), Zeitinvarianz (*time translation symmetry*) oder Eichtransformationsinvarianz (*gauge symmetry*). Unter einer Symmetrietransformation versteht man in der Physik somit eine Transformation, die den Zustand eines Systems nicht verändert. Das System nennt man invariant unter der Wirkung der entsprechenden Transformationsgruppe, also beispielsweise der Gruppe $(G, *)$, deren Menge G aus den Translationen besteht. Die Wirkung einer Gruppe $(G, *)$ nennt man auch Gruppenaktion (*group action*) oder Gruppenoperation. Mathematisch formalisiert werden Symmetrien mittels der Gruppentheorie. Nach Ansicht von Philip W. Anderson, Nobelpreisträger für Physik, ist Physik nichts anderes als das Studium von Symmetrien.

2.5. *Erlanger Programm* von Felix Klein

Der Arbeit von Felix Klein und seinem *Erlanger Programm* ist zu verdanken, dass die Geometrien heutzutage in Beziehung zueinander gesetzt und hierarchisch angeordnet werden können. Klein zeigte, dass eine Geometrie definiert werden kann durch eine geeignete Wahl

von Symmetrietransformationen und schlug vor, dass das Studium von Geometrien aufgefasst werden sollte als das Studium von Symmetrien oder, in anderen Worten, Invarianzen unter gewissen Transformationen. Vor dem Hintergrund dieses Zusammenhangs zwischen Physik und Geometrie ist es auch nicht verwunderlich, dass Albert Einstein zur Beschreibung seiner allgemeinen Relativitätstheorie die Methoden der Differentialgeometrie nutzte, deren zentraler Untersuchungsgegenstand die Mannigfaltigkeiten sind.

Beispielsweise steht in der Hierarchie der Geometrien die affine Geometrie über der euklidischen Geometrie und die projektive Geometrie über der affinen Geometrie. Denn nach Klein ist die euklidische Geometrie definiert durch die Gruppenwirkung der in Abschnitt 2.3 beschriebenen euklidischen Gruppe $\mathrm{Iso}(n)$. Und nach Klein ist die affine Geometrie definiert durch die unter der Wirkung der affinen Gruppe invarianten geometrischen Eigenschaften. Die affine Gruppe besteht aus den bijektiven affinen Abbildungen. In der affinen Geometrie gilt wie in der euklidischen Geometrie das Parallelenpostulat, jedoch gibt es in der affinen Geometrie anders als in der euklidischen Geometrie keine metrische Struktur.

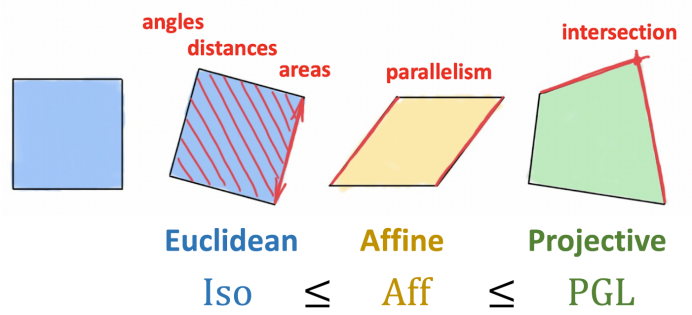


Abbildung 2.2.: **Illustration der Hierarchie der Geometrien nach Felix Klein.** Von den dargestellten Geometrien ist die projektive Geometrie die allgemeinste, die euklidische Geometrie die speziellste und die affine Geometrie dazwischen angesiedelt. [Bro+21b]

Mit der Begründung, dass die nur von der Dimension n des euklidischen Punktraums $\mathbb{E}^n$ abhängige euklidische Gruppe $\mathrm{Iso}(n)$ eine Untergruppe der affinen Gruppe $\mathrm{Aff}(V)$ ist, wobei V einen Vektorraum bezeichnet, welche wiederum eine Untergruppe der projektiven linearen Gruppe $\mathrm{PGL}(V)$ ist, lässt sich der Gruppenhierarchie der durch ihre Gruppenwirkung die Geometrien definierenden Gruppen entsprechend die bereits erwähnte und in Abbildung 2.2 gezeigte Hierarchie $\mathrm{Iso} \leq \mathrm{Aff} \leq \mathrm{PGL}$ der Geometrien aufstellen.

2.6. Operatortheorie

Operatoren spielen beim *geometric deep learning* eine fundamentale Rolle. Operatoren sind Abbildungen zwischen Funktionenräumen. Um eine Abbildung zwischen Funktionenräumen begrifflich von den Abbildungen zwischen Vektorräumen abgrenzen zu können, nennt man Abbildungen zwischen Funktionenräumen auch Operatoren. In der Funktionalanalysis untersucht man lineare Operatoren, ähnlich wie in der linearen Algebra lineare

Abbildungen untersucht werden. Beispiele für Operatoren sind der Faltungsoperator $*$ aus der klassischen Signalverarbeitung, die Fourier-Transformation $\mathcal{F}$ oder der in Abschnitt 2.12 beschriebene *Laplacian*, auch Laplace-Beltrami-Operator genannt, welcher ein Differentialoperator und eine Verallgemeinerung der aus der Schulmathematik bekannten zweiten Ableitung ist.

2.6.1. Invarianz und Äquivarianz eines Operators

Die Invarianz eines Systems oder Operators unter einer bestimmten Transformation wurde in Abschnitt 2.4 bereits vorgestellt. Neben der Invarianz spielt auch die Äquivarianz eines Operators wie beispielsweise dem Faltungsoperator $*$ eine wichtige Rolle beim maschinellen Lernen.

Definition 2.6.1 (Invarianz eines Operators) *Ein Operator f heißt invariant unter einer Gruppenaktion, also unter der Wirkung einer Gruppe $(G, *)$, falls $f \circ g = f$ für alle Elemente $g \in G$.*

Definition 2.6.2 (Äquivarianz eines Operators) *Ein Operator f heißt äquivariant unter einer Gruppenaktion, also unter der Wirkung einer Gruppe $(G, *)$, falls $f \circ g = g \circ f$ für alle Elemente $g \in G$.*

Besonders wichtig für Faltungsnetzwerke ist die Invarianz oder Äquivarianz – je nach Anwendungsfall – unter Translation. Dies kann man sich am Anwendungsfall der Objektklassifikation verdeutlichen. Wenn ein neuronales Netzwerk ein Bild einer Katze als ‚Katze' klassifiziert, soll das neuronale Netzwerk das Bild auch dann noch korrekt als ‚Katze' erkennen, wenn die Katze an eine andere Position im Bild verschoben wurde. Wenn für ein Gebiet Ω mit $x, v \in \Omega$ durch

$$\mathcal{T}_v f(x) = f(x - v)$$

ein Translationsoperator $\mathcal{T}_v$ definiert ist, der auf Funktionen $f \in L^2(\Omega)$ operiert, dann soll für die Invarianz einer Funktion y unter Translation (*translation invariance* oder häufig auch synonym *shift invariance*, in der Physik auch *translational symmetry*) für jede Funktion $f \in L^2(\Omega)$, beispielsweise Bildsignale, die Gleichung

$$y(\mathcal{T}_v f) = y(f)$$

erfüllt sein. Ein Operator y heißt somit invariant unter der Wirkung einer Transformationsgruppe $(G, *)$, wenn die entsprechende Transformation des Eingangssignals, in diesem Beispiel die Translation, vor Anwendung des Operators y keinen Einfluss auf das Ausgangssignal hat.

Bei anderen Anwendungsfällen, beispielsweise der Objektlokalisierung oder der semantischen Segmentierung, ist nicht die Invarianz, sondern die Äquivarianz unter Translation (*translation equivariance* oder auch *shift equivariance*) erwünscht. Wenn beispielsweise eine Katze in einem Bild an eine andere Position verschoben wird, dann soll auch das die Katze umgebende Begrenzungsrechteck (*bounding box*) mit verschoben werden. Für

die Äquivarianz spielt die Kommutativität eine entscheidende Rolle. Eine Funktion y ist äquivariant unter dem Translationsoperator $\mathcal{T}_v$, wenn für jede Funktion $f \in L^2(\Omega)$ die Gleichung

$$y(\mathcal{T}_v f) = \mathcal{T}_v y(f)$$

erfüllt ist.

Obwohl die Faltung eine translations*äquivariante* Operation ist, kann sie dennoch für die Objektklassifikation – einem Anwendungsfall, der Translations*invarianz* erfordert – eingesetzt werden, da in einer Faltungsnetzwerkarchitektur nach der Faltung typischerweise globales *pooling* ausgeführt wird und *pooling* translations*invariant* ist. Dies wird in Abbildung 2.3 auf der nächsten Seite am Beispiel eines Graphen verdeutlicht. Dass die vor der Einführung von Faltungsnetzwerken oder *pooling* gebräuchlichen *multi-layer perceptrons* nicht translationsinvariant sind, ist auch einer der Gründe, warum es in den 1970er Jahren fehlschlug, *multi-layer perceptrons* zur Objektklassifikation einzusetzen. Aus der Motivation heraus, das Problem der translationsinvarianten Objektklassifikation lösen zu können, entstanden die heute bekannten euklidischen Faltungsnetzwerkarchitekturen mit Einsatz von *pooling*.

Das Verständnis geometrischer Prinzipien wie Symmetrie führt zu einer universellen Blaupause für künftige Netzwerkarchitekturen, die sich für verschiedene Typen geometrischer Strukturen eignet und mittels derer sich stabile Repräsentationen hochdimensionaler Daten lernen lassen, die von unter gewissen Transformationen invarianten Funktionen unter Ausnutzung unterschiedlicher Symmetriegruppen erzeugt werden. Neben der Verallgemeinerung der von Faltungsnetzwerken bekannten Konstrukte ist ein solcher Bauplan für ein vereinheitlichtes Rahmenwerk für Netzwerkarchitekturen das angestrebte Ziel des *geometric deep learning*. [Bro+21a]

2.6.2. Hinweis auf mathematisch korrekte Begriffsverwendung

Der Faltungsoperator $*$ kommutiert mit dem Translationsoperator, denn es gilt

$$f(x - x_0) * g(x) = (f * g)(x - x_0),$$

womit die Faltung nach Unterabschnitt 2.6.1 translationsäquivariant ist. In der Literatur wird fälschlicherweise auch dann von Translations*invarianz* (*translation invariance* oder synonym häufig auch *shift invariance*) gesprochen, wenn eigentlich Translations*äquivarianz* gemeint ist. Diese fälschliche Begriffsverwendung ist unglücklicherweise so weit verbreitet, dass sich kaum noch etwas dagegen unternehmen lässt. Um keine Verwirrung hervorzurufen bei denjenigen Lesern, die aus der Literatur die inkorrekt gebrauchten Begriffe gewohnt sind, verwenden die die Bezeichnung *geometric deep learning* prägenden Autoren Bronstein u. a. [Bro+17b] daher bewusst die falsche Terminologie und sprechen von Translations*invarianz* statt von Translations*äquivarianz*. Da in der Literatur bedauerlicherweise nicht die korrekten mathematischen Begrifflichkeiten verwendet werden und Bronstein u. a. deshalb erst gar nicht versuchen, etwas daran zu ändern, wird auch in dieser Arbeit nicht der Versuch unternommen, ausnahmslos die mathematisch korrekte Terminologie zu verwenden und deshalb im Anschluss an diesen Abschnitt unter Umständen auch in solchen Fällen von Invarianz die Rede sein, in denen es nach der Definition

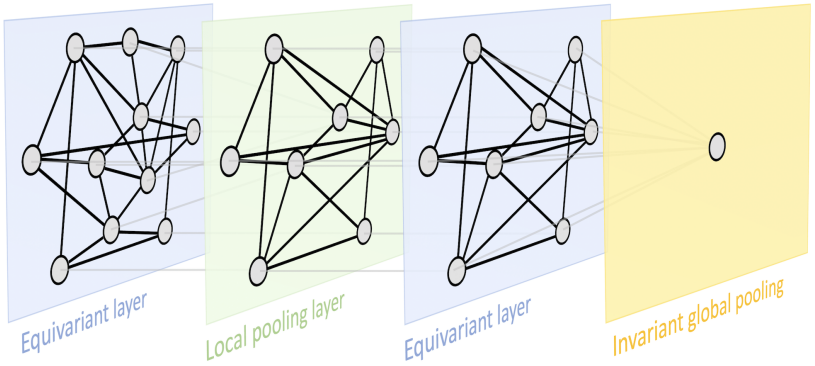


Abbildung 2.3.: **Illustration der Blaupause des *geometric deep learning*.** Die Implementierung der geometrischen Leitsätze bei Einhaltung dieses sehr allgemeinen Bauplans liefert die heutigen Netzwerkarchitekturen. Implementieren die äquivarianten Schichten die Translationsinvarianz, so erhält man ein Faltungsnetzwerk. *Graph neural networks*, *deep sets* [Zah+17] und *transformers* [Vas+17] implementieren Permutationsinvarianz. Implementieren die äquivarianten Schichten die Zeitnormierung (*time warping*) [TO18], so erhält man *gated recurrent neural networks* wie etwa *LSTM (long short-term memory) networks*. Implementieren die Schichten die Eichsymmetrie (*gauge symmetry*), so erhält man die in der Computergrafik gebräuchlichen *intrinsic mesh convolutional neural networks* [Mas+15a; Coh+19]. [Bro+21a]

in Unterabschnitt 2.6.1 korrekterweise Äquivarianz heißen müsste. Der Leser ist hiermit jedoch auf die unterschiedliche Bedeutung der Terminologie hingewiesen und angehalten im Zweifelsfall selbst zu bestimmen welcher der beiden Begriffe deren Definitionen nach der mathematisch korrekte ist. [Bro+17b]

2.7. Klassische Faltung

In diesem Abschnitt wird die klassische, auf euklidischen Gebieten definierte Faltung (*convolution*) vorgestellt, die beim *geometric deep learning* und in Kapitel 3 auf nichteuklidische Gebiete verallgemeinert wird. Die klassische Faltung und verschiedene Verallgemeinerungen davon sind die zentralen Operationen in dieser Arbeit.

2.7.1. Definition der klassischen Faltung

Die Faltung $f * g$ zweier Funktionen f und g ist eine Operation, die genutzt wird, um eine Funktion f mit einer Funktion g zu filtern. Diese im Englischen als *convolution* bezeichnete Operation ist von zentraler Bedeutung für die nach ihr benannten Faltungsnetzwerke (*convolutional neural networks*). Die klassische Faltung zur Anwendung auf einem euklidischen Gebiet $\Omega = \mathbb{R}^n$ ist definiert durch

$$(f * g)(x) = \int_{\mathbb{R}^n} f(x - \tau) g(\tau) \mathrm{d}\tau. \tag{2.1}$$

Aufgrund der durch $f * g = g * f$ gegebenen Kommutativität der Faltung existiert auch die gleichwertige alternative Definition

$$(f * g)(x) = \int_{\mathbb{R}^n} f(\tau) g(x - \tau) \mathrm{d}\tau.$$

Die Faltung ist somit definiert durch das Integral des Produkts einer Funktion f mit gespiegelten und verschobenen Versionen einer Funktion g, weshalb auch die in Unterabschnitt 2.6.1 vorgestellte Translationsäquivarianz so ausgemacht bedeutend für die klassische Faltung auf euklidischen Gebieten ist und weshalb die klassische Faltung sich nicht für nichteuklidische Gebiete eignet, da sie dort aufgrund nicht gegebener Translationsäquivarianz nicht wohldefiniert ist. Die klassische Faltung geht überhaupt erst aus der Translationsäquivarianz hervor, da die Faltung die einzig mögliche lineare Operation ist, welche die Forderung der Translationsäquivarianz erfüllt.

Die klassische Faltung gewichtet für sämtliche $\tau \in \mathbb{R}^n$ deren Funktionswert $f(\tau)$ mit dem Funktionswert $g(x-\tau)$ und bildet durch Integration den Mittelwert $(f*g)(x)$ all dieser Produkte. Indem jeder Wert von f durch das gewichtete Mittel der ihn umgebenden Werte ersetzt wird, wird f sozusagen „verschmiert“. Dieses Prinzip wird auch zur Realisierung des hierarchischen Aufbaus von Faltungsnetzwerken (*convolutional neural networks*) genutzt.

2.7.2. Diskrete Faltung

Für Funktionen mit der Menge $\mathbb{Z}$ der ganzen Zahlen als Definitionsbereich kann auch eine diskrete Faltung definiert werden. Dies ist beispielsweise bei Bildern der Fall, denn dort weist ein Bildsignal f einem Pixel mit ganzzahligen Koordinaten (x, y) pro Farbkanal eine Intensität $f(x, y)$ aus dem Intervall $[0, 255]$ zu. Die diskrete klassische Faltung, wie sie in euklidischen Faltungsnetzwerken zum Einsatz kommt, wird in Abbildung 2.4 veranschaulicht.

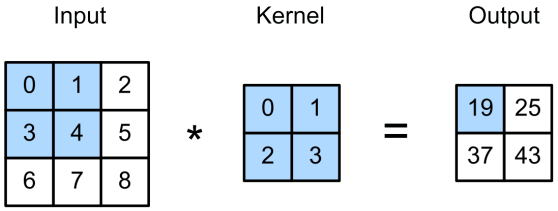


Abbildung 2.4.: **Veranschaulichung der diskreten klassischen Faltung im Fall eines euklidischen Gebiets.** Der Ausgabewert 19 berechnet sich durch $0 \cdot 0 + 1 \cdot 1 + 3 \cdot 2 + 4 \cdot 3$. [Dee21]

Aufgrund des Gleichnisses

$$f(x - x_0) * g(x) = (f * g)(x - x_0), \tag{2.2}$$

kommutiert die Faltung mit Translation, weshalb die Faltung nach Unterabschnitt 2.6.1 äquivariant unter Translation ist. Was die Translationsäquivarianz und Gleichung 2.2 für die Faltung bedeuten, soll am Beispiel einer in Abbildung 2.4 illustrierten diskreten

Faltung veranschaulicht werden. Im Fall eines euklidischen Gebiets $\Omega = \mathbb{R}^n$ als Definitionsbereich (und nur dann) ist dank der Äquivarianz unter Translation die Funktionsweise der Faltung für jeden vom Faltungskern (*convolution kernel*) abgedeckten 2×2-Bereich (*patch*, in der Eingabe blau eingefärbt) der Eingabe gleich. Im Fall nichteuklidischer Gebiete $\Omega \neq \mathbb{R}^n$ ist erschwerender Weise keine Translationsäquivarianz gegeben, weshalb die in Gleichung 2.1 definierte klassische Faltung auf nichteuklidischen Gebieten Ω nicht wohldefiniert ist. Aufgrund der bei nichteuklidischen Gebieten nicht existenten Translationsäquivarianz kann man sich eine Verallgemeinerung der Faltung, sofern sie nicht dem in Abschnitt 3.4 vorgestellten Ansatz der kartografierenden Methoden folgt, somit nicht wie im euklidischen Fall vorstellen als ein gleitendes Fenster (*sliding window*), welches über das Signal geschoben wird.

2.7.3. Herleitung der klassischen Faltung aus Grundprinzipien

Wenngleich die Definition der klassischen Faltung in Unterabschnitt 2.7.1 bereits gegeben wurde, wird in diesem Abschnitt die klassische Faltung aus Grundprinzipien (*first principles*) hergeleitet um einerseits zu zeigen, wo die Definition in Gleichung 2.1 herstammt und andererseits um den Grundstein zu legen für die in Abschnitt 3.2 beschriebene Verallgemeinerung der Faltung im Spektralbereich, die direkte Interpretation der Intuition hinter der in diesem Abschnitt behandelten klassischen Faltung.

Zyklische Matrizen sind besondere Matrizen, denn anders als Matrizen im Allgemeinen kommutieren sie. Für je zwei beliebige zyklische Matrizen $\mathbf{A}$ und $\mathbf{B}$ gilt $\mathbf{AB} = \mathbf{BA}$. Eine zyklische Matrix ist eine quadratische $n \times n$-Matrix, bei der jeder Zeilenvektor relativ zum darüberliegenden Zeilenvektor um einen Eintrag nach rechts verschoben ist. Eine zyklische Matrix hat immer auch eine Toeplitz-Struktur, da ihre Haupt- und Nebendiagonalen konstant sind. Für einen Vektor $\mathbf{a} = (a_0, \ldots, a_{n-1})^\mathsf{T}$, der den ersten Spaltenvektor bildet, hat eine zyklische Matrix $\mathbf{C_a}$ somit folgende Gestalt:

$$\mathbf{C_a} = \begin{pmatrix} a_0 & a_{n-1} & a_{n-2} & \cdots & a_1 \\ a_1 & a_0 & a_{n-1} & & a_2 \\ a_2 & a_1 & a_0 & & a_3 \\ \vdots & & \ddots & \ddots & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_0 \end{pmatrix}$$

Aus der Menge der zyklischen Matrizen hebt sich eine ganz bestimmte zyklische Matrix noch einmal hervor, nämlich $\mathbf{C_a}$ mit $\mathbf{a} = (0, 1, 0, \ldots, 0)^\mathsf{T}$. Matrizen beschreiben lineare Abbildungen und eine Abbildung zwischen Funktionenräumen anstelle von Vektorräumen bezeichnet man als Operator. Die Matrix $\mathbf{C_a}$ mit $\mathbf{a} = (0, 1, 0, \ldots, 0)^\mathsf{T}$ bezeichnet man als *shift operator* $\mathbf{S}$, wobei „*shift*“ synonym für Translation steht. Nach Lemma 3.1 in [Bam20] ist eine Matrix genau dann zyklisch, wenn sie mit Translation kommutiert, also mit der Matrix $\mathbf{S}$. Diese Erkenntnis ist enorm profund.

Einerseits ergibt sich aus diesem Lemma, dass jeder translationsäquivariante Operator – was den Faltungsoperator einschließt – eine zyklische Struktur hat. Bei der Suche nach einem Faltungsoperator kann man sich somit auf diejenigen linearen Operatoren mit zyklischer Struktur beschränken. Aus dieser Überlegung heraus ergibt sich im Fall

euklidischer Gebiete die Definition in Gleichung 2.1. Gleichung 2.1 ergibt sich also aus der Translationsäquivarianz und nicht etwa umgekehrt die Translationsäquivarianz aus Gleichung 2.1. Vielmehr ist die klassische Faltung eben gerade diejenige Operation, die, wie in Abbildung 2.3 auf Seite 15 verdeutlicht, die Translationsäquivarianz implementiert. Aus Grundprinzipien wie Symmetrie lassen sich somit Grundbausteine künstlicher neuronaler Netzwerke herleiten, womit an dieser Stelle noch einmal das Ziel von *geometric deep learning* verdeutlicht werden soll.

Andererseits, und dies wird in Kapitel 3 das zentrale Thema sein, ist auf euklidischen Gebieten jeder lineare Operator, der sich mit einer zyklischen Matrix beschreiben lässt und als Folge davon von der Fourier-Basis diagonalisiert wird, translationsäquivariant und somit ein Faltungsoperator. Die diskrete Faltung $\mathbf{x} * \mathbf{a}$ zweier beliebiger n-dimensionaler Vektoren $\mathbf{x}$ und $\mathbf{a}$ lässt sich formulieren als $\mathbf{C_a x}$. Dies wird am Beispiel des *shift operator* $\mathbf{S}$ in Abbildung 2.5 verdeutlicht. Mittels geometrischen Grundprinzipien und dank der Arbeit von Felix Klein lässt sich somit wie in Abbildung 2.3 auf Seite 15 gezeigt eine universelle Blaupause zur Vereinheitlichung aller Netzwerkarchitekturen aufstellen. [Bro20a; Bro+21a]

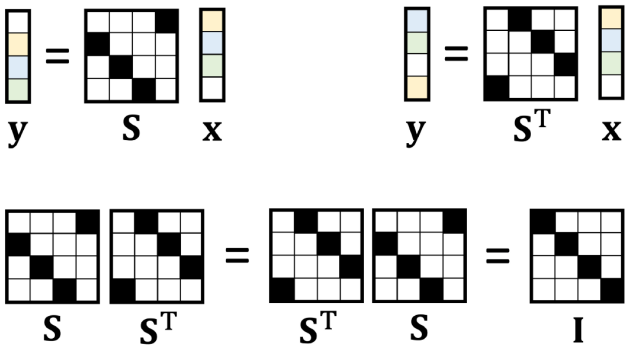


Abbildung 2.5.: **Illustration des *(right) shift operator* S.** Das Matrix-Vektor-Produkt $\mathbf{Sx}$ entspricht der diskreten Faltung $\mathbf{x} * (0, 1, 0, \ldots, 0)^\mathsf{T} = \mathbf{y}$, durch welche die Komponenten des Vektors $\mathbf{x}$ zyklisch um eine Position verschoben werden. Die Multiplikation eines Vektors mit dem *left shift operator* $\mathbf{S}^\mathsf{T}$ verschiebt die Vektorkomponenten zyklisch um eine Position in die Gegenrichtung, weshalb die Hintereinanderausführung $\mathbf{SS}^\mathsf{T} = \mathbf{S}^\mathsf{T}\mathbf{S}$ der Identitätsabbildung $\mathbf{I}$ entspricht. Der *shift operator* $\mathbf{S}$ ist somit eine orthogonale Abbildung, weshalb seine Eigenvektoren orthogonal (mangels der Symmetrie von $\mathbf{S}$ aber komplex) sind, was zum in Unterabschnitt 2.13.3 vorgestellten Faltungstheorem führt. [Bro20a]

2.8. Mannigfaltigkeiten

In diesem Abschnitt wird die Mannigfaltigkeit vorgestellt. Mannigfaltigkeiten $\mathcal{X}$ sind neben Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ die typischerweise betrachteten und die für diese Arbeit wichtigen nichteuklidischen Gebiete Ω.

Differenzierbare Mannigfaltigkeiten sind ein Oberbegriff für Kurven, Flächen und andere geometrische Objekte, auf denen die Berechnung von Ableitungen und verwandten

Konzepten möglich ist. Differenzierbare Mannigfaltigkeiten sind Untersuchungsschwerpunkt der Differentialgeometrie, spielen aber auch eine zentrale Rolle in der theoretischen Physik bei der Beschreibung der Raumzeit in der allgemeinen Relativitätstheorie von Albert Einstein. Differenzierbare Mannigfaltigkeiten können mit einem als *Riemannsche Metrik* bezeichneten inneren Produkt versehen werden, mit der sich die charakteristischen geometrischen Eigenschaften der Mannigfaltigkeit beschreiben lassen. Das innere Produkt wird in Unterabschnitt 2.10.3 vorgestellt. Eine solche Mannigfaltigkeit kann weiterhin mit einer Textur oder anderen Attributen ausgestattet werden und dient in der Computergrafik und im Feld des maschinellen Sehens (*computer vision*) dazu, dreidimensionale geometrische Objekte zu repräsentieren.

2.8.1. Definition einer Mannigfaltigkeit

Im Kontext dieser Arbeit kann man sich Mannigfaltigkeiten vorstellen als die Oberflächen dreidimensionaler Objekte. Eine Mannigfaltigkeit $\mathcal{X}$ mit Dimension d ist vereinfacht ausgedrückt ein Raum, bei dem die unmittelbare Nachbarschaft jedes Punkts $x \in \mathcal{X}$ einem euklidischen Raum $\mathbb{R}^d$ gleicht. Am einfachsten verstehen lassen sich Mannigfaltigkeiten an einem Beispiel wie dem in Abbildung 2.6 gezeigten. Die Oberfläche einer Kugel ist zwar kugelförmig, wählt man die Umgebung eines Oberflächenpunktes jedoch klein genug, so scheint es sich bei der Umgebung um eine Ebene zu handeln. Der für das Verständnis der Mannigfaltigkeiten hilfreichste Gedanke ist derselbe Gedanke, der die Menschheit über Generationen hinweg glauben ließ, die Erde wäre eine Scheibe.

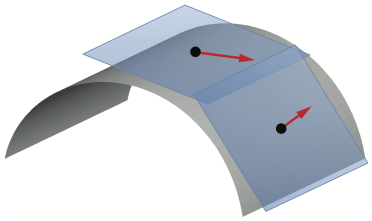


Abbildung 2.6.: **Beispiel einer zweidimensionalen Mannigfaltigkeit.** Grau gefärbt die zweidimensionale, in einen dreidimensionalen Raum eingebettete Mannigfaltigkeit $\mathcal{X}$. Blau gefärbt der jeweils zu einem schwarz dargestellten Punkt $x \in \mathcal{X}$ der Mannigfaltigkeit gehörige Tangentialraum $\mathbb{R}^2$, bei dem es sich um einen euklidischen Raum derselben Dimension wie die Mannigfaltigkeit $\mathcal{X}$ handelt, der zudem topologisch äquivalent mit einer Nachbarschaft des Punkts x ist. [Bro+17b]

Definiert ist eine d-dimensionale Mannigfaltigkeit als ein topologischer Raum, bei dem jeder Punkt eine Nachbarschaft hat, die topologisch äquivalent zu einem euklidischen Raum $\mathbb{R}^d$ derselben Dimension wie die Mannigfaltigkeit ist. Dieser Raum $\mathbb{R}^d$ wird als Tangentialraum (*tangent space*) bezeichnet, unabhängig seiner Dimension. Die Grundmenge der Punkte muss also über eine topologische Struktur verfügen, wodurch eine Abstraktion von Nähe gegeben ist und somit die Vorstellung von Nachbarschaften möglich wird. Zudem muss ein Homöomorphismus existieren, der für jeden Punkt $x \in \mathcal{X}$ eine

solche Nachbarschaft in einen euklidischen Raum derselben Dimension d wie die Mannigfaltigkeit überführen kann und umgekehrt den d-dimensionalen euklidischen Raum in die Nachbarschaft des Punktes überführen kann. Anschaulich bedeutet dies für die in der Computergrafik und in dieser Arbeit typischerweise zweidimensionalen Mannigfaltigkeiten, dass ähnlich zu den von Funktionsgraphen reeller Funktionen bekannten Tangenten an jeden Punkt x der Mannigfaltigkeit $\mathcal{X}$ eine Tangentialebene $\mathbb{R}^2$ angelegt werden kann und dass für jeden Punkt x eine Nachbarschaft existieren muss, die topologisch dieser angelegten Tangentialebene gleicht.

Zweidimensionale Mannigfaltigkeiten werden in der Computergrafik häufig als Begrenzungsoberfläche (*boundary surface*) genutzt um dreidimensionale Formen zu modellieren. Auch wenn zweidimensionale Mannigfaltigkeiten zur Modellierung dreidimensionaler Formen genutzt werden, handelt es sich bei ihnen dennoch nicht um dreidimensionale Räume. Es werden lediglich zweidimensionale Mannigfaltigkeiten in einen dreidimensionalen Raum eingebettet. Als Beispiel aus dem Alltag kann man sich vorstellen, dass man ein Stück Aluminiumfolie zwar um ein dreidimensionales Objekt herumwickeln kann, die Aluminiumfolie selbst ist aber nur zweidimensional. Für eine sich in solch einer zweidimensionalen Welt bewegende Ameise gibt es nur die Bewegungsrichtungen nach vorne, hinten oder zur Seite, auch wenn sie dabei einer gekrümmten Fläche folgt, während ein menschlicher Beobachter von seinem extrinsischen Blickpunkt aus eine Bewegung im dreidimensionalen Raum wahrnimmt.

2.8.2. Tangentialraum und Riemannsche Metrik

Ein zu einem Punkt $x \in \mathcal{X}$ gehöriger Tangentialraum $T_x\mathcal{X}$ (*tangent space*) ist ein euklidischer Vektorraum $\mathbb{R}^d$, der eine differenzierbare d-dimensionale Mannigfaltigkeit $\mathcal{X}$ am Punkt x linear approximiert. Der Tangentialraum $T_x\mathcal{X}$ wird von den Tangentialvektoren in x aufgespannt. Nur die Punkte einer differenzierbaren Mannigfaltigkeit haben einen Tangentialraum. Einer topologischen Mannigfaltigkeit fehlt die dafür nötige Struktur.

Für einen Tangentialraum kann man ein inneres Produkt $\langle \cdot, \cdot \rangle_{T_x\mathcal{X}} \colon T_x\mathcal{X} \times T_x\mathcal{X} \to \mathbb{R}$ definieren, welches Riemannsche Metrik genannt wird. Für die Funktionsräume der auf der Mannigfaltigkeit definierten Funktionen – Skalarfelder und Vektorfelder – kann man ohne Riemannsche Metrik kein inneres Produkt definieren, weil der Mannigfaltigkeit die dafür nötige Vektorraumstruktur fehlt. Die Riemannsche Metrik induziert eine Volumenform $\mathrm{d}x$, welche die in Unterunterabschnitt 2.10.4.1 gezeigte Definition von inneren Produkten für die beiden Funktionsräume $L^2(\mathcal{X})$ und $L^2(T\mathcal{X})$ ermöglicht. Die Riemannsche Metrik ist nicht eindeutig bestimmt. Man kann das innere Produkt für den Tangentialraum auf verschiedene Weisen definieren. Zwei unterschiedliche Realisierungen der Riemannschen Metrik nennt man Isometrien. Eine Riemannsche Metrik ist jedoch keine Metrik im eigentlichen Sinne, sondern lediglich ein inneres Produkt. Verfügt eine Mannigfaltigkeit über eine Riemannsche Metrik, so heißt sie Riemannsche Mannigfaltigkeit. Eigenschaften der Mannigfaltigkeit, die sich rein unter Verwendung der Metrik ohne Rückgriff auf den umgebenden Raum beschreiben lassen, nennt man intrinsisch. Die Wichtigkeit, Operatoren wie beispielsweise den Faltungsoperator oder die verschiedenen in Unterabschnitt 2.12.2 vorgestellten Differentialoperatoren *intrinsisch* zu definieren, zeigt sich in Abbildung 3.2 auf Seite 59.

2.8.3. Tangentialbündel

Das Tangentialbündel $T\mathcal{X}$ (*tangent bundle*) einer Mannigfaltigkeit $\mathcal{X}$ ist die disjunkte Vereinigung aller Tangentialräume $T_x\mathcal{X}$.

2.8.4. Vektorfeld

Ein Vektorfeld ist eine Funktion, die jedem Punkt eines Raumes einen Vektor zuordnet. In Abbildung 2.7 wird dies illustriert. Im Kontext dieser Arbeit ist ein Tangentialvektorfeld $F\colon \mathcal{X} \to T\mathcal{X}$ (*tangent vector field*) die Abbildung, die jedem Punkt x einer Mannigfaltigkeit $\mathcal{X}$ einen Vektor aus dem zu x gehörigen Tangentialraum $T_x\mathcal{X}$ – also einen Tangentialvektor – zuweist. Für zweidimensionale Mannigfaltigkeiten ist $T_x\mathcal{X}$ eine euklidische Ebene, also $\mathbb{R}^2$.

Tangentialvektorfelder werden genutzt um das für die Differentialrechnung nötige Konzept der infinitesimalen Verschiebung, das bei Mannigfaltigkeiten aufgrund der fehlenden Vektorraumstruktur jedoch nicht erlaubt ist, dennoch für Mannigfaltigkeiten formalisieren zu können. Dadurch kann eine Verallgemeinerung des Konzepts der Richtungsableitung umgesetzt werden.

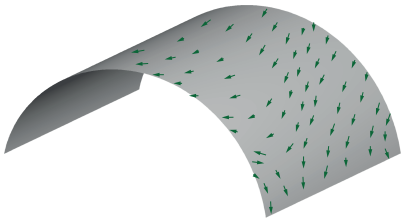


Abbildung 2.7.: **Veranschaulichung eines auf einer Mannigfaltigkeit definierten Vektorfelds.** Jedem Punkt x der in grau dargestellten Mannigfaltigkeit $\mathcal{X}$ wird ein Tangentialvektor zugeordnet. [Bro+21a]

2.8.5. Skalarfeld

Ein Skalarfeld (*scalar field*) oder auch skalares Feld ist eine Funktion $f\colon \mathcal{X} \to \mathbb{R}$, die jedem Punkt eines Raumes $\mathcal{X}$, hier eine Mannigfaltigkeit, eine reelle Zahl zuordnet. Beispielsweise wird in der Physik den Punkten des Raums ein Wert für die Temperatur, den Luftdruck oder eine Dichte zugewiesen. Die Intensität dieses Werts kann wie auf der nächsten Seite in Abbildung 2.8 dargestellt durch verschiedene Farben repräsentiert werden.

2.9. Graphen

In diesem Abschnitt werden Graphen thematisiert. Graphen sind neben Mannigfaltigkeiten die typischerweise betrachteten und die für diese Arbeit wichtigen nichteuklidische Gebiete.

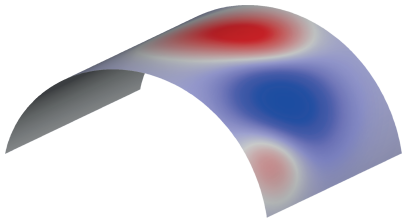


Abbildung 2.8.: **Veranschaulichung eines auf einer Mannigfaltigkeit definierten Skalarfelds.** Jedem Punkt x der in grau dargestellten Mannigfaltigkeit $\mathcal{X}$ wird ein Skalar zugeordnet. [Bro+21a]

2.9.1. Definition eines Graphen

Ein Graph $\mathcal{G}$ ist definiert als ein Paar $(\mathcal{V}, \mathcal{E})$ bestehend aus einer Knotenmenge $\mathcal{V}$ und einer Kantenmenge $\mathcal{E}$, wobei $|\mathcal{V}| = n$ die Anzahl der Knoten in $\mathcal{V}$ angibt und $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ die Paarungen (i, j) von Knoten $i, j \in \mathcal{V}$ enthält, die über eine Kante miteinander verbunden sind.

2.9.2. Zusammenhang zwischen Graphen und Mannigfaltigkeiten

In diesem Abschnitt soll der Zusammenhang von Graphen und Mannigfaltigkeiten verdeutlicht werden. Graphen können als diskretes Analogon zu Mannigfaltigkeiten aufgefasst werden. Die Überlegungen für Mannigfaltigkeiten lassen sich analog auf den diskreten Fall für Graphen übertragen, wie man beispielsweise auf Seite 25 in Unterabschnitt 2.10.4 bei der Definition der inneren Produkte oder auf Seite 28 in Abschnitt 2.12 bei der Definition des Laplace-Beltrami-Operators sehen kann. Gerade dies ist ja auch die Absicht des *geometric deep learning* – eine universelle Blaupause zu erstellen, die alle möglichen Netzwerkarchitekturen abdeckt und zur Anwendung auf allen möglichen Typen von Daten geeignet ist um so mit den Mitteln der Geometrie eine Vereinheitlichung nach dem Vorbild von Felix Klein und seinem *Erlanger Programm* zu erreichen.

Tatsächlich wird eine Mannigfaltigkeit beziehungsweise deren Diskretisierung, ein Polygonnetz (*mesh*), häufig als Graph interpretiert um diesen Graphen mit einem *graph neural network* zu verarbeiten, wie in Kapitel 4 oder Kapitel 6 dargelegt. Auch können aus Punktwolken Graphen konstruiert werden und anschließend *graph neural networks* auf diesen Graphen angewandt werden, wie in Kapitel 5 beschrieben.

2.10. Funktionenräume

In diesem Abschnitt werden die in dieser Arbeit betrachteten Funktionenräume näher vorgestellt. Bei einem Funktionenraum handelt es sich um einen Raum, dessen Elemente Funktionen sind, ganz ähnlich zu Vektorräumen, deren Elemente Vektoren sind. Wie für Vektorräume auch lassen sich für Funktionenräume Skalarprodukte definieren, was unter anderem ein Konzept von Orthogonalität ermöglicht. Während man sich für Vektoren noch bildhaft vorstellen kann, was es bedeutet, wenn zwei Vektoren senkrecht aufeinander

stehen, erscheint die Vorstellung zweier senkrecht aufeinander stehender Funktionen weniger sinnergebend. Anstatt sich auf sein Vorstellungsvermögen zu verlassen, überträgt man die von Vektoren bekannten mathematischen Definitionen auf Funktionen und nennt zwei Funktionen orthogonal zueinander, wenn deren Skalarprodukt gleich Null ist.

2.10.1. Hilbertraum

Bei den in dieser Arbeit für ein Gebiet Ω (*domain*) definierten, aus quadratintegrierbaren Funktionen f bestehenden Funktionenräumen $L^2(\Omega)$ handelt es sich um Hilberträume. Ein Funktionenraum kann mit einer Vektoraddition und Skalarmultiplikation versehen werden und bildet dann einen Vektorraum. Einen solchen Funktionenraum bezeichnet man als linearen Funktionenraum. Ein Hilbertraum ist ein Vektorraum über dem Körper der reellen oder komplexen Zahlen, der folgende zwei Bedingungen erfüllen muss: Der Vektorraum muss mit einem Skalarprodukt versehen sein und der Vektorraum muss vollständig bezüglich der (hier durch das Skalarprodukt induzierten) Metrik sein. Hilberträume sind für gewöhnlich unendlich-dimensional.

Das Skalarprodukt induziert eine Norm (Längenbegriff) und die Norm induziert wiederum eine Metrik (Abstandsbegriff). Indem für einen Vektorraum ein Skalarprodukt definiert wird, wird er zu einem metrischen Raum. Die zweite Bedingung für Hilberträume besagt, dass dieser metrische Raum vollständig sein muss, dass in ihm also jede Cauchy-Folge konvergieren muss. Es muss also genug Grenzwerte geben, um Differential- und Integralrechnung betreiben zu können. Um eine Norm oder Metrik zu haben, braucht ein Vektorraum nicht notwendigerweise ein Skalarprodukt. Es existieren auch Normen, die nicht von einem Skalarprodukt induziert wurden und Metriken, die nicht von einer Norm induziert wurden.

Ist nur die erste Bedingung erfüllt, spricht man anstatt eines Hilbertraums von einem Prähilbertraum. Ist nur die zweite Bedingung erfüllt, spricht man von einem vollständigen Raum oder auch Cauchy-Raum. Wurde die Metrik von einer Norm induziert, spricht man auch von einem Banachraum, unabhängig davon, ob der Raum vollständig ist oder nicht.

Hilberträume tragen durch ihr Skalarprodukt eine topologische Struktur, ermöglichen also eine Vorstellung von Nähe, was zur Bildung von Nachbarschaften der Elemente wichtig ist und beispielsweise bei Punktwolken große Probleme bereitet. Dies wird in Kapitel 5 wichtig werden, da Punktwolken einfach nur Mengen von Punkten sind und ohne Skalarprodukt oder topologische Struktur auch die Bildung von Nachbarschaften erschweren.

Von den Funktionenräumen L^p ist der Raum L^2 der quadratintegrierbaren Funktionen der einzige, der sich mit einem inneren Produkt versehen und somit zu einem Hilbertraum machen lässt. Das innere Produkt erlaubt es Konzepte wie Winkel oder Orthogonalität zu definieren. Das innere Produkt ist eine Verallgemeinerung des bekannten Skalarprodukts. Das Konzept von Orthogonalität ist notwendig um eine orthogonale beziehungsweise orthonormale Basis zu ermöglichen, etwa die Fourier-Basis oder Eigenbasis des *Laplacian*. Die Inverse einer Orthogonalmatrix, etwa die aus den orthonormalen Basisfunktionen einer der genannten Basen bestehende Matrix, entspricht ihrer Transponierten. Dies spielt in Abschnitt 3.2 bei der Verallgemeinerung der Faltung im Spektralbereich eine Rolle.

Stark vereinfacht gesagt ist ein Hilbertraum eine Menge von Funktionen, die alle denselben Definitionsbereich haben – eben das Gebiet Ω – beispielsweise eine Mannigfaltigkeit $\mathcal{X}$,

die Knotenmenge $\mathcal{V}$ oder Kantenmenge $\mathcal{E}$ eines Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ oder einen euklidischen Raum $\mathbb{R}^n$. Bei diesen Funktionen kann es sich beispielsweise um die Skalarfelder oder Vektorfelder einer Mannigfaltigkeit handeln, die einem Punkt einer Mannigfaltigkeit eben einen Skalar oder Vektor zuordnen.

Die Verallgemeinerung der euklidischen Geometrie auf unendlichdimensionale Räume führt zu den Hilberträumen. Beschränkt man sich umgekehrt auf endlichdimensionale Räume und die reellen Zahlen $\mathbb{R}$, so erhält man den euklidischen Raum. Der euklidische Vektorraum ist ein bekanntes Beispiel für einen Hilbertraum, da für ihn ein Skalarprodukt definiert ist und die davon abgeleitete Metrik vollständig ist.

2.10.2. Quadratintegrierbare Funktionen

Bei den in dieser Arbeit betrachteten Funktionen $f \in L^p(\Omega)$ mit $p = 2$ handelt es sich um die p-fach integrierbaren beziehungsweise quadratintegrierbaren Funktionen. Eine Funktion f heißt genau dann quadratintegrierbar oder auch quadratisch integrierbar, wenn die Integration der Funktion f, auf welche noch die Betragsfunktion und Quadratsfunktion angewendet wird, einen endlichen Wert liefert, also wenn gilt:

$$f\colon \mathbb{R} \to \mathbb{C} \text{ quadratintegrierbar} \quad \Longleftrightarrow \quad \int_{-\infty}^{\infty} |f(x)|^2 \,\mathrm{d}x < \infty \tag{2.3}$$

Die quadratintegrierbaren Funktionen L^2 bilden einen Hilbertraum, da für zwei quadratintegrierbaren Funktionen f und g durch

$$\langle f, g \rangle = \int_A \overline{f(x)} g(x) \,\mathrm{d}x \tag{2.4}$$

ein Skalarprodukt definiert ist und L^2 vollständig bezüglicher der durch dieses Skalarprodukt über die Norm induzierten Metrik ist. Dabei bezeichnet $\overline{f(x)}$ die komplexe Konjugation des Funktionswerts $f(x)$ und der Integrationsbereich entspricht dem Gebiet Ω, etwa eine Mannigfaltigkeit $\mathcal{X}$.

2.10.3. Inneres Produkt

Ein inneres Produkt ist weniger speziell als ein Skalarprodukt. Ein Skalarprodukt ist eine

- Bilinearform, die
- positiv-definit und
- symmetrisch ist, also $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$ erfüllt,

wohingegen ein inneres Produkt lediglich eine

- Linearform, also nur in einem Argument linear zu sein braucht, die
- positiv-definit und
- konjugiert symmetrisch oder auch hermitesch ist, also $\langle \mathbf{x}, \mathbf{y} \rangle = \overline{\langle \mathbf{y}, \mathbf{x} \rangle}$ erfüllt.

Ein Skalarprodukt ist somit auch ein inneres Produkt, aber nicht jedes innere Produkt ist auch ein Skalarprodukt in diesem Sinne.

Das bekannteste Skalarprodukt ist das auch als euklidisches Skalarprodukt bezeichnete Standardskalarprodukt zweier Vektoren $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, definiert durch

$$\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n = \sum_{i=1}^{n} x_i y_i.$$

Das bekannteste Beispiel für ein inneres Produkt ist das komplexe Standardskalarprodukt zweier Vektoren $\mathbf{x}, \mathbf{y} \in \mathbb{C}^n$, definiert durch

$$\langle \mathbf{x}, \mathbf{y} \rangle := \bar{x}_1 y_1 + \bar{x}_2 y_2 + \cdots + \bar{x}_n y_n = \sum_{i=1}^{n} \bar{x}_i y_i = \overline{\langle \mathbf{y}, \mathbf{x} \rangle}.$$

2.10.4. Definition der inneren Produkte

In diesem Abschnitt werden die inneren Produkte für Mannigfaltigkeiten $\mathcal{X}$ als auch für Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ definiert.

2.10.4.1. Mannigfaltigkeiten

Der Raum $L^2(\mathcal{X})$ der für den Definitionsbereich $\mathcal{X}$ definierten Skalarfelder wird durch Definition des Skalarprodukts

$$\langle f, g \rangle_{L^2(\mathcal{X})} = \int_{\mathcal{X}} f(x) g(x) \mathrm{d}x \tag{2.5}$$

zum Hilbertraum. Der Raum $L^2(T\mathcal{X})$ der für das Tangentialbündel $T\mathcal{X}$ der Mannigfaltigkeit $\mathcal{X}$ definierten Tangentialvektoren wird durch Definition des Skalarprodukts

$$\langle F, G \rangle_{L^2(T\mathcal{X})} = \int_{\mathcal{X}} \langle F(x), G(x) \rangle_{T_x\mathcal{X}} \mathrm{d}x \tag{2.6}$$

zum Hilbertraum. Dabei steht $T_x\mathcal{X}$ für den zu einem Punkt x der Mannigfaltigkeit $\mathcal{X}$ gehörigen Tangentialraum (*tangent space*). Das innere Produkt $\langle \cdot, \cdot \rangle_{T_x\mathcal{X}}$ in Gleichung 2.6 ist die in Unterabschnitt 2.8.2 vorgestellte und in Unterabschnitt 2.12.2 angesprochene Riemannsche Metrik der Mannigfaltigkeit. Als Riemannsche Metrik bezeichnet man das für die Tangentialräume einer differenzierbaren Mannigfaltigkeit definierte innere Produkt. Eine Mannigfaltigkeit, die über eine Riemannsche Metrik verfügt, bezeichnet man als Riemannsche Mannigfaltigkeit. Erst die Riemannsche Metrik induziert die Volumenform $\mathrm{d}x$ in Gleichung 2.5 und Gleichung 2.6, weshalb die Definition dieser beiden inneren Produkte nur für Riemannsche Mannigfaltigkeiten möglich ist.

2.10.4.2. Graphen

In der Differentialgeometrie arbeitet man mit stetigen, also integrierbaren Skalarfeldern und Vektorfeldern. Bei Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ hingegen arbeitet man mit auf den Knoten $\mathcal{V}$

definierten Funktionen $f\colon \mathcal{V} \to \mathbb{R}$ und auf den Kanten $\mathcal{E}$ definierten Funktionen $F\colon \mathcal{E} \to \mathbb{R}$. Somit sind für Graphen diskrete Analoga zu den für Mannigfaltigkeiten definierten innere Produkte nötig.

Das innere Produkt zweier auf Knoten operierenden Funktionen f und g sei

$$\langle f, g \rangle_{L^2(\mathcal{V})} = \sum_{i \in \mathcal{V}} a_i f_i g_i,$$

wobei $a_i > 0$ das einem Knoten $i \in \mathcal{V}$ zugeordnete Gewicht ist. Das innere Produkt zweier auf Kanten operierenden Funktionen F und G sei

$$\langle F, G \rangle_{L^2(\mathcal{E})} = \sum_{i \in \mathcal{E}} w_{ij} F_{ij} G_{ij},$$

wobei $w_{ij} \geq 0$ das einer Kante $(i, j) \in \mathcal{E}$ zugeordnete Gewicht ist. Durch Definition dieser inneren Produkte spezifiziert man, dass die Funktionen $f \in L^2(\mathcal{V})$ und $F \in L^2(\mathcal{E})$ jeweils Hilberträume bilden.

2.11. Hauptprobleme nichteuklidischer Gebiete

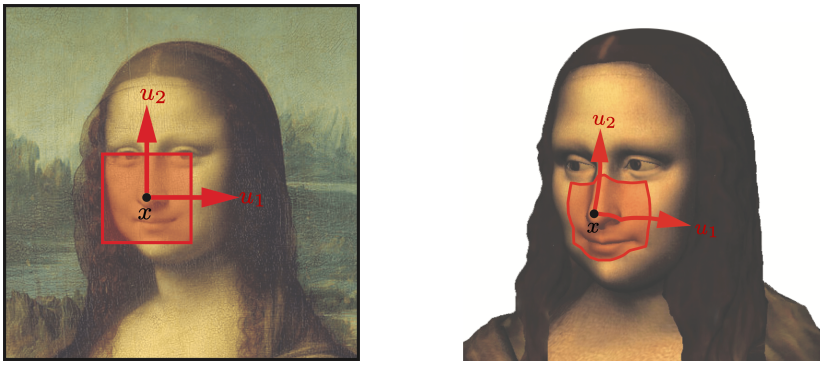
Beim maschinellen Lernen auf nichteuklidischen Gebieten mittels künstlichen neuronalen Netzwerken treten einige Probleme auf, die es bei euklidischen Daten nicht gibt. Diese Probleme werden in diesem Abschnitt genauer behandelt. Zwei der größten Probleme bei nichteuklidischen Gebieten sind die nicht vorhandene Translationsäquivarianz (von Bronstein u. a. [Bro+17b] bewusst als *shift invariance* bezeichnet, siehe Unterabschnitt 2.6.2) sowie die nicht vorhandene globale Parametrisierung. Aber auch die fehlende Vektorraumstruktur stellt eine Hürde dar, wie sich beispielsweise in Unterabschnitt 2.12.2 zeigt.

2.11.1. Fehlende Translationsäquivarianz

Dieses Problem bedeutet im Wesentlichen, dass man sich bei nichteuklidischen Gebieten von der vertrauten Vorstellung der Faltung, nach welcher eine Vorlage in der Manier eines gleitenden Fensters (*sliding window*) über ein Signal geschoben wird, trennen muss.

Die klassische Faltung ist wie in Abschnitt 2.7 ausgeführt und in Abbildung 2.3 auf Seite 15 dargestellt diejenige Operation, welche die Translationsäquivarianz implementiert.

Nach Definition 2.6.2 auf Seite 13 heißt ein Operator f äquivariant unter der Wirkung einer Gruppe $(G, *)$, falls $f \circ g = g \circ f$ für alle Elemente $g \in G$. Für die Existenz von Translationsäquivarianz muss als Voraussetzung zunächst eine Translationsgruppe existieren, welche auf das Gebiet Ω wirkt. Für nichteuklidische Gebiete ist es im Allgemeinen jedoch nicht offensichtlich wie eine solche Translationsgruppe definiert werden soll, wenn es denn überhaupt möglich ist. Ohne die Existenz einer Translationsgruppe existiert somit auch keine Translationsäquivarianz, weshalb es auf solchen Gebieten auch keine Faltung als diejenige Operation, welche die Translationsäquivarianz implementiert, geben kann und die klassische Faltung auf nichteuklidischen Gebieten nicht wohldefiniert ist. Dies liegt im Fall von Mannigfaltigkeiten an deren Krümmung und wird in Abbildung 2.9 auf der nächsten Seite verdeutlicht.



(a) Klassische Faltung auf einem Bild

(b) Verallgemeinerte Faltung auf einer gekrümmten Fläche

Abbildung 2.9.: **Veranschaulichung der Bedeutung fehlender Translationsäquivarianz.** Abbildung 2.9a zeigt die klassische Faltung auf einem euklidischen Gebiet, hier einem RGB-Bild. Aufgrund der Translationsäquivarianz verläuft die Faltung an jeder Position x identisch. Abbildung 2.9b zeigt die verallgemeinerte Faltung auf einem nichteuklidischen Gebiet, hier eine diskretisierte Mannigfaltigkeit. Die lokale Struktur einer Fläche unterscheidet sich aufgrund ihrer Krümmung von Position zu Position. Mangels Translationsäquivarianz ist die Faltung positionsabhängig. Die klassische Faltung ist auf nichteuklidischen Gebieten nicht wohldefiniert. [Bro+17a]

Es kann lediglich nach einem linearen Operator gesucht werden, welcher analog zu der klassischen Faltung arbeitet. Die Suche nach einem solchen ähnlich der klassischen Faltung arbeitenden Operator ist das zentrale Thema in Kapitel 3 und auch dieser Arbeit. Aus der Funktionalanalysis ist bekannt, dass im euklidischen Fall jeder lineare Operator, der mit einer Toeplitz-Matrix kommutiert, ein Faltungsoperator ist. Ein Operator mit Toeplitz-Struktur ist beispielsweise der Laplace-Operator. Für eine Verallgemeinerung der Faltung auf nichteuklidische Gebiete kann man diese Forderung auflockern und nur noch Kommutativität mit dem in Abschnitt 2.12 vorgestellten *Laplacian* beziehungsweise *graph Laplacian*, einer Verallgemeinerung des Laplace-Operators auf nichteuklidische Gebiete, fordern. Diese Überlegungen führen auf intuitivem Weg zu einer ersten Idee, wie die Faltung auf nichteuklidische Gebiete verallgemeinert werden könnte, die in Abschnitt 3.2 beschrieben wird.

Die bei euklidischen Gebieten vorhandene Translationsäquivarianz bestimmt viele wichtige Eigenschaften der klassischen Faltung. Denn wenn ein Operator mit Translation kommutiert, ist er nicht positionsabhängig. Aus der Translationsäquivarianz folgt, dass die in Abbildung 2.4 auf Seite 16 veranschaulichte Extraktion eines Bereichs (*patch*) aus einem Bildsignal an jeder Position identisch abläuft. Nur aus diesem Grund kann man sich die klassische Faltung vorstellen als eine Vorlage g, auch Faltungskern genannt, die in der Manier eines gleitenden Fensters (*sliding window*) über ein Signal f geschoben wird, wobei die Filterkoeffizienten in g dem abgedeckten Bereich des Signals f zugeordnet werden. Für die klassische Faltung auf euklidischen Gebieten wurde auf Seite 16 in Gleichung 2.2 die Äquivarianz unter Translation gezeigt. [Bro+17b; Bro+21a; Bro19]

2.11.2. Fehlende globale Parametrisierung

Unter globaler Parametrisierung versteht man die Abbildung von einem Gebiet Ω auf einen euklidischen Raum $\mathbb{R}^n$, üblicherweise die Ebene $\mathbb{R}^2$. Über eine solche Abbildung kann man für die Punkte eines Gebiets Ω auf ein globales System von Koordinaten zurückgreifen.

Für die in dieser Arbeit untersuchten Mannigfaltigkeiten und Graphen ist jedoch keine bedeutungsvolle globale Parametrisierung möglich. Nach dem *Theorema egregium* („herausragender Satz“) von Carl Friedrich Gauß, der sich auch mit der Geodäsie beschäftigte, ist die Gaußsche Krümmung einer Fläche eine intrinsische Invariante einer Fläche, also invariant unter lokaler Isometrie. Anders ausgedrückt bedeutet dies, dass zwei isometrisch parametrisierte Flächenstücke dieselbe Gaußsche Krümmung haben. Somit ist keine isometrische, also die Längen von Kurven erhaltende Einbettung einer gekrümmten Fläche $S \subset \mathbb{R}^3$ in die Ebene möglich, da eine gekrümmte Fläche eine positive Krümmung $\kappa > 0$ (Kugelfläche) beziehungsweise negative Krümmung $\kappa < 0$ (Sattelfläche) hat, wohingegen eine Ebene Krümmung $\kappa = 0$ hat. Eine gekrümmte Fläche und eine Ebene können daher nicht isometrisch sein. Dies ist auch ein großes Problem in der Kartografie.

Um auf einem nichteuklidischen Gebiet auf ein System von Koordinaten zurückgreifen zu können, ist eine Idee, für jeden Bereich des Gebiets ein lokales (intrinsisches) Koordinatensystem einzuführen. Diese Idee verfolgen die in Abschnitt 3.4 vorgestellten kartografierenden Methoden (*charting-based methods*). Jedoch ist die Konstruktion eines solchen lokalen Systems von Koordinaten positionsabhängig und funktioniert je nach Bereich (*patch*) des Gebiets unterschiedlich.

Fehlende globale Parametrisierung ist effektiv somit gleichbedeutend mit dem Fehlen eines globalen Systems von Koordinaten. Ohne Koordinatensystem können die Punkte der Grundmenge eines topologischen Raums auch nicht in eine Reihenfolge geordnet werden, was beim Einlesen der nichteuklidischen Daten problematisch ist. Ohne eine kanonische Einlesereihenfolge (in Ermangelung eines globalen Systems von Koordinaten) muss ein künstliches neuronales Netzwerk für zufriedenstellende Ergebnisse auf nichteuklidischen Gebieten somit für sämtliche Permutationen der Grundmenge korrekt arbeiten. Diese Invarianz unter Permutation oder auch Permutationsinvarianz spielt insbesondere in Kapitel 5 eine wichtige Rolle. [Har17]

2.12. *Laplacian* und *graph Laplacian*

In diesem Abschnitt wird der Laplace-Beltrami-Operator vorgestellt und hergeleitet, welcher für die Definition einer verallgemeinerten Faltung von zentraler Wichtigkeit ist. Da durch die Eigenfunktionen des Laplace-Beltrami-Operators die Verallgemeinerung der Fourier-Basis auf nichteuklidische Gebiete definiert ist, ermöglicht der Laplace-Beltrami-Operator die Spektralanalyse auf Mannigfaltigkeiten und Graphen. Eine weit verbreitete alternative Bezeichnung für den Laplace-Beltrami-Operator lautet *Laplacian* oder manchmal auch *verallgemeinerter Laplace-Operator*, da der Laplace-Beltrami-Operator den Laplace-Operator des euklidischen Raums auf gekrümmte Flächen und (pseudo-)riemannsche Mannigfaltigkeiten verallgemeinert. Im Rahmen dieser Arbeit wird der Begriff *Laplacian*

verwendet, wenn der Operator im Kontext von Mannigfaltigkeiten gebraucht wird. Im Zusammenhang mit Graphen wird hingegen der Begriff *graph Laplacian* verwendet.

2.12.1. Allgemeines zum Laplace-Beltrami-Operator

In der mehrdimensionalen Analysis wird der Laplace-Operator Δ verwendet, welcher nicht mit dem in dieser Arbeit thematisierten Laplace-Beltrami-Operator zu verwechseln ist. Er wird auch als euklidischer *Laplacian* bezeichnet, da er auf Funktionen f mit einem euklidischen Raum als Definitionsbereich angewendet wird. Er lässt sich unabhängig vom gewählten Koordinatensystem definieren durch $\Delta f = \operatorname{div}(\operatorname{grad} f)$, also die Divergenz des Gradienten einer Funktion, oder mit dem Nabla-Operator notieren als $\Delta f = \nabla^2 f$. Bei Wahl des kartesischen Koordinatensystems ergibt sich die Darstellung

$$\Delta f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2},$$

die sich in einer Dimension auf die zweite Ableitung $\Delta f = f''$ reduziert. Eine Verallgemeinerung dieses Laplace-Operators für auf Riemannschen Mannigfaltigkeiten definierte Funktionen ist der Laplace-Beltrami-Operator, der Gegenstand dieser Arbeit ist und mit der Bezeichnung *Laplacian* gemeint ist.

Der Laplace-Beltrami-Operator ist ein linearer Differentialoperator. Er ist weiterhin ein selbstadjungierter positiv semi-definiter Operator, was bedeutet, dass man ihn – ein kompaktes Gebiet vorausgesetzt, was in der Praxis immer der Fall ist – in eine diskrete Menge orthonormaler Eigenfunktionen $\phi_1, \phi_2, \ldots$ zerlegen kann, welche in Abschnitt 2.14 vorgestellt werden. Diese Eigenfunktionen des Laplace-Beltrami-Operators spielen im Verlauf dieser Arbeit immer wieder eine tragende Rolle, da sie als eine Verallgemeinerung der klassischen Fourier-Basis auf nichteuklidische Gebiete interpretiert werden können und somit die Spektralanalyse auf Mannigfaltigkeiten beziehungsweise im Fall des *graph Laplacian* auf Graphen ermöglichen. Dies wird unter anderem wichtig bei den in Abschnitt 3.2 vorgestellten Spektralmethoden, die eine verallgemeinerte Faltung im Spektrum des Laplace-Beltrami-Operators definieren, weshalb der Laplace-Beltrami-Operator von fundamentaler Bedeutung ist.

Der *Laplacian* ist ein Differentialoperator, der ein zweimal differenzierbaren Skalarfeld f auf ein anderes abbildet, indem er dem Skalarfeld f die negative Divergenz seines Gradienten zuordnet. Die Begriffe *Divergenz* und *Gradient* werden noch definiert, für Mannigfaltigkeiten in Unterabschnitt 2.12.2 und für Graphen in Unterabschnitt 2.12.3. Das Funktionsprinzip ist für Mannigfaltigkeiten und Graphen identisch, da es sich bei den auf den Knoten und Kanten eines Graphen definierten Funktionen um die diskreten Analoga zu den stetigen Skalarfeldern und Vektorfeldern einer Mannigfaltigkeit handelt.

2.12.2. Herleitung des *Laplacian* für Mannigfaltigkeiten

Um mit den auf einer Mannigfaltigkeit $\mathcal{X}$ definierten Funktionen Differentialrechnung betreiben zu können und den für die Verallgemeinerung der Faltung wichtigen *Laplacian* definieren zu können und schlussendlich natürlich auch auf der Differentialrechnung

basierende künstliche neuronale Netzwerke anwenden zu können, müssen noch einige Definitionen eingeführt werden. Denn eines der großen Probleme nichteuklidischer Gebiete ist, dass sie wie in Abschnitt 2.11 erwähnt über keine Vektorraumstruktur verfügen, womit nicht einmal die Addition oder Subtraktion zweier Vektoren definiert ist. Somit ist auch ein Ausdruck wie $f(x + \mathrm{d}x)$ nicht erlaubt, wobei $\mathrm{d}x$ die in Unterunterabschnitt 2.10.4.1 erwähnte Volumenform ist, die durch die Riemannsche Metrik induziert wird. In der Differentialrechnung beschreibt die Ableitung, wie sich der Wert einer Funktion f ändert, wenn dessen Argument einer infinitesimalen Änderung unterzogen wird. Ohne Vektorraumstruktur kann die Differentialrechnung auf diesem klassischen Wege nicht für Mannigfaltigkeiten ermöglicht werden. Dieses Problem soll nun angegangen werden.

Auf der Mannigfaltigkeit selbst kann man Konzepte wie Differentiale nicht definieren, wohl aber lokal in den einzelnen auf Seite 19 in Abbildung 2.6 gezeigten Tangentialräumen der Punkte x der Mannigfaltigkeit. Allerdings müssen die Tangentialräume, um in sie ausweichen zu können, auch existieren. Damit die Tangentialräume existieren muss die Mannigfaltigkeit eine differenzierbare Mannigfaltigkeit sein, da nur deren Punkte über Tangentialräume verfügen. Die Struktur einer topologischen Mannigfaltigkeit reicht hierfür nicht aus. Die Tangentialräume sind wie in Abschnitt 2.8 erwähnt Vektorräume derselben Dimension d wie die Mannigfaltigkeit, verfügen im Gegensatz zur Mannigfaltigkeit $\mathcal{X}$ also über eine Vektorraumstruktur. Für jeden Punkts $x \in \mathcal{X}$ wird nun für dessen Tangentialraum $T_x\mathcal{X}$ ein inneres Produkt $\langle \cdot, \cdot \rangle_{T_x\mathcal{X}} \colon T_x\mathcal{X} \times T_x\mathcal{X} \to \mathbb{R}$ definiert. Dieses innere Produkt nennt man Riemannsche Metrik. Die Riemannsche Metrik ermöglicht es für die auf der Mannigfaltigkeit $\mathcal{X}$ definierten Funktionen – sowohl die Skalarfelder als auch die Vektorfelder – nun ebenfalls innere Produkte zu definieren. Durch Definition des auf Seite 25 in Gleichung 2.5 formulierten inneren Produkts $\langle f, g \rangle_{L^2(\mathcal{X})}$ wird der Funktionenraum bestehend aus den auf der Mannigfaltigkeit definierten Skalarfelder zum Hilbertraum $L^2(\mathcal{X})$. Durch Definition des auf Seite 25 in Gleichung 2.6 formulierten inneren Produkts $\langle F, G \rangle_{L^2(T\mathcal{X})}$ wird der Funktionenraum bestehend aus den auf der Mannigfaltigkeit definierten Vektorfelder zum Hilbertraum $L^2(T\mathcal{X})$.

Nun ist es möglich einen zum aus der Analysis bekannten Gradienten ähnlichen Operator zu definieren, der jedoch auf intrinsisch Weise arbeitet, weshalb man den neu definierten Operator $\nabla \colon L^2(\mathcal{X}) \to L^2(T\mathcal{X})$ auch als intrinsischen Gradienten bezeichnet. Er beschreibt wie der bekannte Gradient die Richtung der größten Änderung, bei der es sich hier um einen Tangentialvektor handelt. Das Konzept der Richtungsableitung wird über die in Unterabschnitt 2.8.4 vorgestellten Tangentialvektorfelder umgesetzt. Den zum Gradienten adjungierten Operator nennt man die Divergenz $\operatorname{div} \colon L^2(T\mathcal{X}) \to L^2(\mathcal{X})$, die definiert ist durch

$$\langle F, \nabla f \rangle_{L^2(T\mathcal{X})} = \langle \nabla^* F, f \rangle_{L^2(\mathcal{X})} = \langle -\operatorname{div} F, f \rangle_{L^2(\mathcal{X})}$$

und ebenfalls auf intrinsische Weise arbeitet. Die negative Divergenz des Gradienten eines Skalarfelds f, also

$$\Delta f = -\operatorname{div}(\nabla f),$$

ist nun der gesuchte *Laplacian* oder auch Laplace-Beltrami-Operator, der eine zentrale Rolle bei der Verallgemeinerung der Faltung spielt und in Abbildung 2.10 auf der nächsten Seite veranschaulicht wird. Der *Laplacian* ist einer der wichtigsten Operatoren in der mathematischen Physik und wird unter anderem in der Quantenmechanik sowie bei der Beschreibung

der Ausbreitungsgeschwindigkeit von Wellen oder der Wärmeleitung verwendet. In der Literatur wird der *Laplacian* häufig als die positive Divergenz des Gradienten definiert, was ebenfalls zulässig ist. Die Inklusion des negativen Vorzeichens in die Definition des *Laplacian* vereinfacht allerdings spätere Berechnungen wie beispielsweise in Unterabschnitt 2.14.2 nachvollzogen werden kann, weshalb sie hier präferiert wird. Aufgrund seiner intrinsischen Konstruktion ist auch der *Laplacian* intrinsisch definiert. [Bro+17b]

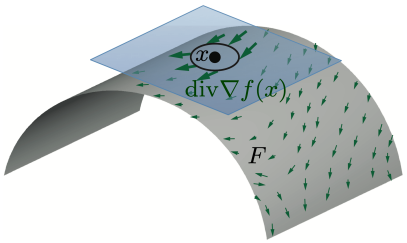


Abbildung 2.10.: **Veranschaulichung des *Laplacian***, dem Differentialoperator Δ, der auch als Laplace-Beltrami-Operator bekannt ist. Den *Laplacian* kann man interpretieren als die Differenz zwischen dem Durchschnitt einer Funktion auf einer infinitesimalen Kugel um einen Punkt x herum und dem Funktionswert des Punkts x selbst. [Bro+17a]

Verdeutlichung der Zusammenhänge

Der Gradient angewandt auf ein Skalarfeld ergibt ein Vektorfeld. Die Divergenz ist ein Operator, der auf Vektorfelder wirkt. Die Divergenz eines Vektorfelds (und somit auch der *Laplacian* angewandt auf ein Skalarfeld) gibt an wie sehr die Vektoren in der Umgebung eines Punktes auseinanderstreben. Die Divergenz eines Vektorfelds (und somit auch der *Laplacian* angewandt auf ein Skalarfeld) ergibt wieder ein Skalarfeld. Ein Skalarfeld ist eine Funktion.

2.12.3. Herleitung des *graph Laplacian* für Graphen

Der Einfachheit wegen werden ungerichtete, aber gewichtete Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ betrachtet. Es gilt also $(i, j) \in \mathcal{E} \iff (j, i) \in \mathcal{E}$ für die Kantenmenge $\mathcal{E}$. Die Knotenmenge $\mathcal{V} = \{1, \dots, n\}$ besteht aus n Knoten. Jedem Knoten $i \in \mathcal{V}$ ist ein Gewicht $a_i > 0$ zugeordnet und jeder Kante $(i, j) \in \mathcal{E}$ ein Gewicht $w_{ij} \geq 0$. Die Funktionenräume $L^2(\mathcal{V})$ und $L^2(\mathcal{E})$ sind mit den in Unterunterabschnitt 2.10.4.2 definierten inneren Produkte versehen und somit Hilberträume. Die Definition der Differentialoperatoren ∇, div und Δ für Graphen erfolgt analog zu denen für Mannigfaltigkeiten (siehe Unterabschnitt 2.12.2),

Der *graph gradient* $\nabla\colon L^2(\mathcal{V}) \to L^2(\mathcal{E})$ bildet über die Berechnungsvorschrift

$$(\nabla f)_{ij} = f_i - f_j \tag{2.7}$$

Funktionen, die auf Knoten definiert sind, ab auf Funktionen, die auf Kanten definiert sind. Da die Kanten des Graphen ungerichtet sind, es für jede Kante (i, j) also auch eine

Kante (j, i) gibt, wobei mit $F_{ij} = -F_{ji}$ die auf den Kanten definierten Funktionen $F\colon \mathcal{E} \to \mathbb{R}$ antisymmetrisch sind, muss der *graph gradient* $(\nabla f)_{ij} = -(\nabla f)_{ji}$ erfüllen, was für Gleichung 2.7 der Fall ist. Die *graph divergence* ist wie bei den Mannigfaltigkeiten ein zum Gradienten adjungierter Operator $\operatorname{div}\colon L^2(\mathcal{E}) \to L^2(\mathcal{V})$. Definiert ist die *graph divergence* durch

$$(\operatorname{div} F)_i = \frac{1}{a_i} \sum_{(i,j)\in\mathcal{E}} w_{ij} F_{ij}.$$

Der *graph Laplacian* $\Delta\colon L^2(\mathcal{V}) \to L^2(\mathcal{V})$ ist wie schon bei den Mannigfaltigkeiten definiert durch die negative Divergenz des Gradienten der Funktion, also durch

$$(\Delta f)_i = -\frac{1}{a_i} \sum_{(i,j)\in\mathcal{E}} w_{ij} \left(f_i - f_j\right) = \frac{1}{a_i} \sum_{(i,j)\in\mathcal{E}} w_{ij} \left(f_j - f_i\right)$$

gegeben. Dass auch bei den Graphen der Gradient und die Divergenz adjungiert zueinander bezüglich der in Unterunterabschnitt 2.10.4.2 definierten inneren Produkte sind, ist durch

$$\langle F, \nabla f\rangle_{L^2(\mathcal{E})} = \langle \nabla^* F, f\rangle_{L^2(\mathcal{V})} = \langle -\operatorname{div} F, f\rangle_{L^2(\mathcal{V})}$$

gezeigt. Der *graph Laplacian* kann alternativ auch in einer Matrix-Vektor-Form

$$\Delta\mathbf{f} = \mathbf{A}^{-1}(\mathbf{D} - \mathbf{W})\mathbf{f}$$

geschrieben werden, wobei $\mathbf{A} = \operatorname{diag}(a_1, \ldots, a_n)$ die $n \times n$ Diagonalmatrix mit den Knotengewichten a_i auf der Hauptdiagonalen ist und $\mathbf{D} = \operatorname{diag}(\sum_{j:j\neq i} w_{ij})$ die als Gradmatrix (*degree matrix*) bezeichnete Diagonalmatrix ist, die auf ihrer Hauptdiagonalen für jeden Knoten dessen Knotengrad (*degree*) trägt. Die $n \times n$-Matrix $\mathbf{W}$ setzt sich zusammen aus den Kantengewichten w_{ij}, wobei für ein Knotenpaar (i, j), das nicht durch eine Kante verbunden ist, der Wert $w_{ij} = 0$ eingetragen wird. [Bro+17b]

Betrachtet man statt gewichteten Graphen ungewichtete Graphen, so vereinfacht sich $\mathbf{A}^{-1}(\mathbf{D} - \mathbf{W})$ zu $\mathbf{D} - \mathbf{W}$, wobei $\mathbf{W}$ in diesem Fall der Adjazenzmatrix des Graphen entspricht. Die in diesem Fall gegebene Matrix $\mathbf{L} := \mathbf{D} - \mathbf{W}$ bezeichnet man als die Laplace-Matrix (*Laplacian matrix*), die für einen schlichten Graphen dessen *graph Laplacian* darstellt.

2.13. Fourier-Analysis

Für die in Kapitel 3 vorgestellten Ansätze zur Verallgemeinerung der Faltung auf nichteuklidische Gebiete ist ein grundlegendes Verständnis der Fourier-Analysis nötig. In diesem Abschnitt werden die wichtigsten Konzepte und Begrifflichkeiten der Fourier-Analysis kurz zusammengefasst. Die Fourier-Analysis wird auch als klassische harmonische Analyse bezeichnet. In der abstrakten harmonischen Analyse werden Verallgemeinerungen der Fourierreihen untersucht, die bestimmte Eigenschaften in Bezug zu Symmetrien aufweisen. Die Wirkung des Translationsoperators auf Funktionen spielt eine wichtige Rolle in der harmonischen Analyse. Der Laplace-Beltrami-Operator ermöglicht die harmonische Analyse von auf nichteuklidischen Gebieten definierten Funktionen. Der für die in Abschnitt 3.2

behandelte direkte Interpretation der Intuition hinter der klassischen Faltung wichtige Spektralbereich (*spectral domain*) ist das Analogon zum klassischen Frequenzbereich der Fourier-Analyse, die in diesem Abschnitt in ihren Grundzügen dargelegt wird.

2.13.1. Grundlegendes zur Fourier-Analysis

Die Fourier-Analysis wird vor allem verwendet, um Signale in deren Frequenzanteile zu zerlegen. Diese Frequenzanteile nennt man das Spektrum des Signals beziehungsweise der Funktion. Durch die Zerlegung eines Signals in seine Frequenzanteile erhält man somit ein Frequenzspektrum. Bei dem zu zerlegenden Signal kann es sich um ein von der Zeit abhängiges Signal handeln, beispielsweise eine Schallwelle, oder auch um ein ortsabhängiges Signal wie es in der Bildverarbeitung der Fall ist. Beispielsweise kann es sich um ein Bildsignal handeln, welches in Abhängigkeit von den Koordinaten eines auf einem Gitter angeordneten Pixels einen gewissen Intensitätswert aufweist.

In der Fourier-Analysis werden verschiedene Transformationen betrachtet mittels derer die Beschreibung des Signalverlaufs in eine alternative Beschreibung überführt werden kann. Hierbei wird auch eine spezielle Nomenklatur verwendet, deren Begriffe sich je nach Anwendungsfall unterscheiden, sinngemäß aber für dasselbe stehen: den Urbildbereich und den Bildbereich der Transformation. Bei zeitabhängigen Signalen transformiert man zwischen dem Zeitbereich (*time domain*) und dem Frequenzbereich (*frequency domain*). Bei ortsabhängigen Signalen transformiert man zwischen dem Ortsbereich (*spatial domain*) und dem Frequenzbereich (*frequency domain*).

Die Fourier-Basis besteht aus verschiedenen sinus- und kosinusförmigen Funktionen (*sinusoids*). Auf nichteuklidischen Gebieten ist die Fourier-Basis jedoch nicht definiert. Um die Theorie der Fourier-Analysis auf nichteuklidische Gebiete zu übertragen, werden die Eigenfunktionen linearer Operatoren untersucht, welche in Abschnitt 2.14 vorgestellt werden. Die Eigenwerte eines linearen Operators sind eng mit dem Konzept von Frequenz verwandt und werden als das Spektrum des Operators bezeichnet. Im Kontext nichteuklidischer Gebiete spricht man vom Spektralbereich (*spectral domain*) statt vom Frequenzbereich (*frequency domain*). Die Spektraltheorie für die in der Funktionalanalysis untersuchten linearen Operatoren ist eine Verallgemeinerung der Eigenwerttheorie aus der linearen Algebra und wird in Abschnitt 2.14 auf Seite 37 näher beschrieben.

Von zentraler Bedeutung für nichteuklidische Gebiete sind die Eigenfunktionen des in Abschnitt 2.12 vorgestellten *Laplacian* beziehungsweise *graph Laplacian*. Der Laplace-Beltrami-Operator ermöglicht die harmonische Analyse auf nichteuklidischen Gebieten. Die Eigenfunktionen des Laplace-Beltrami-Operators können als Verallgemeinerung der klassischen Fourier-Basis auf nichteuklidische Gebiete interpretiert werden. Im Fall eines euklidischen Raums sind die Fourier-Basisfunktionen gerade die Eigenfunktionen des (eindimensionalen) Laplace-Operators wie in Abschnitt 2.14 noch gezeigt wird.

Die hauptsächliche Motivation für Joseph Fourier um die Fourier-Analysis zu entwickeln war der Wunsch nach der Lösung von partiellen Differentialgleichungen wie etwa der Wärmeleitungsgleichung (*heat equation*), die beispielsweise in Unterunterabschnitt 3.4.5.2 bei dem *Anisotropic CNN* eine zentrale Rolle spielt.

2.13.2. Fourier-Transformation

Die (kontinuierliche) Fourier-Transformation ist eine lineare Abbildung $\mathcal{F}$, die von einem Funktionenraum in einen zweiten Funktionenraum abbildet. Abbildungen zwischen Funktionenräumen nennt man zur begrifflichen Abgrenzung von Abbildungen zwischen Vektorräumen auch Operatoren. Die Fourier-Transformierte $\mathcal{F}\{f\}$ einer Funktion f mit $\mathbf{x}, \boldsymbol{\xi} \in \mathbb{R}^n$ kann definiert werden durch

$$\mathcal{F}\{f\}(\boldsymbol{\xi}) = \frac{1}{\sqrt{2\pi}^n} \int_{\mathbb{R}^n} f(\mathbf{x})\, \mathrm{e}^{-\mathrm{i}\mathbf{x}\cdot\boldsymbol{\xi}}\, \mathrm{d}\mathbf{x} = \int_{\mathbb{R}^n} f(\mathbf{x})\, \mathrm{e}^{-2\pi\mathrm{i}\mathbf{x}\cdot\boldsymbol{\xi}}\, \mathrm{d}\mathbf{x}, \tag{2.8}$$

wobei $\boldsymbol{\xi}$ die Frequenz repräsentiert und $\mathbf{x} \cdot \boldsymbol{\xi}$ das Skalarprodukt der zwei Vektoren ist. Die verschiedenen Exponentialfunktionen $\mathrm{e}^{-\mathrm{i}\mathbf{x}\cdot\boldsymbol{\xi}}$ werden als Fourier-Basisfunktionen bezeichnet. Die Fourier-Transformation $\mathcal{F}$ wird auch als Fourier-Analyse bezeichnet. Die inverse Fourier-Transformation $\mathcal{F}^{-1}$ wird auch als Fourier-Synthese bezeichnet und ist bei Beachtung der in Gleichung 2.8 eingehaltenen Konvention definiert durch

$$f(\mathbf{x}) = \frac{1}{\sqrt{2\pi}^n} \int_{\mathbb{R}^n} \mathcal{F}\{f\}(\boldsymbol{\xi})\, \mathrm{e}^{\mathrm{i}\mathbf{x}\cdot\boldsymbol{\xi}}\, \mathrm{d}\boldsymbol{\xi} = \int_{\mathbb{R}^n} \mathcal{F}\{f\}(\boldsymbol{\xi})\, \mathrm{e}^{2\pi\mathrm{i}\mathbf{x}\cdot\boldsymbol{\xi}}\, \mathrm{d}\boldsymbol{\xi}.$$

Es existieren auch andere Definitionen des Paars bestehend aus $\mathcal{F}$ und $\mathcal{F}^{-1}$, die alternativen Konventionen folgen. In der Signalverarbeitung gebräuchlich ist unter Verwendung der Kreisfrequenz $\boldsymbol{\omega} = 2\pi\boldsymbol{\xi}$ auch die Definition der Fourier-Transformation $\mathcal{F}$ durch

$$\mathcal{F}\{f\}(\boldsymbol{\omega}) = \int_{\mathbb{R}^n} f(\mathbf{x})\, \mathrm{e}^{-\mathrm{i}\boldsymbol{\omega}\cdot\mathbf{x}}\, \mathrm{d}\mathbf{x}. \tag{2.9}$$

Die zur Definition in Gleichung 2.9 gehörige inverse Fourier-Transformation ist aufgrund von $\boldsymbol{\xi} = \frac{\boldsymbol{\omega}}{2\pi}$ definiert durch

$$f(\mathbf{x}) = \frac{1}{(2\pi)^n} \int_{\mathbb{R}^n} \mathcal{F}\{f\}(\boldsymbol{\omega})\, \mathrm{e}^{\mathrm{i}\boldsymbol{\omega}\cdot\mathbf{x}}\, \mathrm{d}\boldsymbol{\omega}.$$

In dieser Form ist die Fourier-Transformation jedoch keine unitäre Abbildung mehr auf $L^1(\mathbb{R}^n) \cap L^2(\mathbb{R}^n)$, weshalb im Fall dieser Definition der Satz von Plancherel nicht mehr anwendbar ist. Der Satz von Plancherel besagt, dass die Fourier-Transformation angewendet auf eine quadratintegrierbare Funktion eine Isometrie ist, also dass eine quadratintegrierbare Funktion und ihre Fourier-Transformierte die gleiche Norm haben. Aus diesem Grund ist auch die Definition

$$\mathcal{F}\{f\}(\boldsymbol{\omega}) = \frac{1}{\sqrt{2\pi}^n} \int_{\mathbb{R}^n} f(\mathbf{x})\, \mathrm{e}^{-\mathrm{i}\boldsymbol{\omega}\cdot\mathbf{x}}\, \mathrm{d}\mathbf{x}$$

gepaart mit

$$f(\mathbf{x}) = \frac{1}{\sqrt{2\pi}^n} \int_{\mathbb{R}^n} \mathcal{F}\{f\}(\boldsymbol{\omega})\, \mathrm{e}^{\mathrm{i}\boldsymbol{\omega}\cdot\mathbf{x}}\, \mathrm{d}\boldsymbol{\omega}$$

üblich. Allgemein werden in den Ingenieurwissenschaften unter Umständen andere Symbole als in der Mathematik für dieselben Konzepte verwendet, beispielsweise der Buchstabe j

anstelle von i für die imaginäre Zahl um Verwechslungen mit dem Symbol I für den elektrischen Strom zu vermeiden.

Der Operator $\mathcal{F}$ ist ein linearer Operator, genau wie der Faltungsoperator $*$ oder der in Abschnitt 2.12 vorgestellte Laplace-Beltrami-Operator, also der Differentialoperator Δ.

Die Fourier-Transformation erlaubt es, ein Signal nicht nur in Form dessen Intensität an jedem Punkt zu untersuchen, sondern auch unter dem Aspekt wie stark jede der in dem Signal enthaltenen Frequenzen vertreten ist. Die Fourier-Transformation ist somit ein Weg um zu messen wie eng ein gegebenes Signal mit einer bestimmten Frequenz zusammenhängt. Allerdings liefert diese alternative Darstellung keine Information darüber, an welchem Punkt in der Zeit beziehungsweise an welchem Punkt im Raum die jeweiligen Frequenzen auftreten. Dies ist der in Unterabschnitt 2.13.4 vorgestellten Unschärferelation geschuldet.

Die Fourier-Transformation ist nur eine von vielen Transformationen, die im Bereich der Fourier-Analysis eingesetzt werden. Sie wird zur Zerlegung aperiodischer Signale in ein kontinuierliches Spektrum genutzt und ist eine Verallgemeinerung der Fourierreihe, mit welcher sich lediglich periodische Funktionen und deren Spektrum beschreiben lassen. Aus der Fourier-Transformation heraus haben sich verschiedene Varianten und Optimierungen entwickelt. Auf eine davon, die Kurzzeit-Fourier-Transformation (*short-time Fourier transform*), wird in Unterabschnitt 3.5.1 näher eingegangen.

2.13.3. Faltungstheorem

Ein Korrolar des Faltungstheorems ist, dass die Faltung einer Funktion f mit einer Funktion g gleichbedeutend ist mit der inversen Fourier-Transformation des punktweisen Produkts der beiden Fourier-Transformierten $\mathcal{F}\{f\}$ und $\mathcal{F}\{g\}$. Es gilt also

$$\{f * g\}(x) = \mathcal{F}^{-1}\big\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\big\}, \tag{2.10}$$

wobei $*$ der Faltungsoperator ist, $x \in \mathbb{R}$ und $\cdot$ das punktweise Produkt ist.

Das Faltungstheorem selbst besagt, dass

$$\mathcal{F}\{f * g\}(x) = \mathcal{F}\{f\}(x)\,\mathcal{F}\{g\}(x). \tag{2.11}$$

In Verbindung mit der Definition der Fourier-Transformation $\mathcal{F}$ in Gleichung 2.9 besagt das Faltungstheorem somit, dass

$$\widehat{(f * g)}(\omega) := \mathcal{F}\{f * g\}(\omega) = \int_{\mathbb{R}^n} f(x)\,\mathrm{e}^{-\mathrm{i}\omega x}\,\mathrm{d}x \int_{\mathbb{R}^n} g(x)\,\mathrm{e}^{-\mathrm{i}\omega x}\,\mathrm{d}x,$$

woraus sich gemäß Gleichung 2.10 durch Anwendung der inversen Fourier-Transformation $\mathcal{F}^{-1}$ das Ergebnis der Faltung $f * g$ gewinnen lässt. Der Grund für diese Eigenschaft ist, dass die Fourier-Transformation $\mathcal{F}$ den Faltungsoperator $*$ diagonalisiert. Die Erklärung dafür ist in Unterabschnitt 2.7.3 beschrieben: zwei kommutierende Matrizen sind simultan diagonalisierbar und zyklische Matrizen kommutieren. Bei Wahl der in Unterabschnitt 2.7.3 definierten orthogonalen Matrix $\mathbf{S}$, dem *shift operator*, ergibt sich nach Abschnitt 4.1 in [Bam20], dass der *shift operator* von der Fourier-Transformation diagonalisiert wird.

Dies bedeutet, dass $\overline{\mathbf{\Phi}^{\mathsf{T}}}\mathbf{S}\mathbf{\Phi}$ eine Diagonalmatrix ist, wobei $\overline{\mathbf{\Phi}^{\mathsf{T}}}$ die Matrix-Form der Fourier-Transformation ist, die aufgrund der komplexen Eigenvektoren von $\mathbf{S}$ – mangels Symmetrie von $\mathbf{S}$ – komplex konjugiert werden muss, und $\mathbf{\Phi}$ die inverse Fourier-Transformation ist. Da alle zyklischen Matrizen simultan diagonalisierbar sind und die Faltung sich mit einer zyklischen Matrix beschreiben lässt, wird neben dem *shift operator* auch die Faltung von der Fourier-Transformation diagonalisiert, was eben das Faltungstheorem liefert.

Die Faltung ist eine vergleichsweise rechenintensive Operation, während für die Multiplikation von Matrizen optimierte Algorithmen existieren, die von heutigen Rechnern ohne großen Rechenaufwand ausgeführt werden können. Das Faltungstheorem ermöglicht deshalb eine effiziente Implementierung der Faltung und somit auch von Faltungsnetzwerken. Im Fall euklidischer Gebiete ist ein solcher Algorithmus etwa die schnelle Fourier-Transformation (*fast Fourier transform*, FFT), mittels derer die Berechnung der Fourier-Transformation $\mathcal{F}$ und ihrer inversen Transformation $\mathcal{F}^{-1}$ sich enorm beschleunigen lässt. In der Signalverarbeitung auf Graphen und Mannigfaltigkeiten existieren zur schnellen Fourier-Transformation analoge Algorithmen jedoch schlichtweg nicht, was insbesondere bei den in Abschnitt 3.2 vorgestellten Spektralmethoden große Probleme zur Folge hat und die Motivation für die in Abschnitt 3.3 vorgestellten Spektrum-freien Methoden ist, welche dieses Problem des nicht existenten Analogons der schnellen Fourier-Transformation für Graphen und Mannigfaltigkeiten umgehen. [Bro20a]

2.13.4. Unschärferelation

Unter einer Unschärferelation versteht man eine Beziehung zwischen zwei physikalischen Größen, die sich darin auswirkt, dass sich nicht beide Größen gleichzeitig beliebig genau bestimmen lassen. Eine bekannte Unschärferelation ist die Heisenbergsche Unschärferelation (*uncertainty principle*) aus der Quantenphysik, welche besagt, dass es unmöglich ist, zwei komplementäre Eigenschaften eines Elementarteilchens – beispielsweise der Ort und der Impuls des Teilchens – gleichzeitig beliebig genau zu bestimmen. Die Heisenbergsche Unschärferelation ist nur ein spezifisches Beispiel eines wesentlich allgemeineren Kompromisses, der im Zusammenhang mit Wellenfunktionen eingegangen werden muss. Die Heisenbergsche Unschärferelation ergibt sich aus der Tatsache, dass Quantenobjekte durch Materiewellen beschrieben werden.

Für die Fourier-Transformation bedeutet dies, dass die gleichzeitige Kenntnis zweier komplementärer Eigenschaften von Fourier-Paaren limitiert ist. Die beiden komplementären Eigenschaften sind in diesem Fall etwa die Dauer und die Frequenz einer Wellenfunktion. Die Unschärferelation verhindert eine beliebig genaue Auflösung in beiden Bereichen zur selben Zeit. In anderen Worten ist es unmöglich, eine messbare Größe gleichzeitig beliebig genau im Ortsbereich als auch im Frequenzbereich der Fourier-Transformation zu bestimmen. Dies wird sich auch nie ändern, es sei denn es stellt sich heraus, dass das Universum von anderen Gesetzen der Physik beherrscht wird als bislang angenommen.

Im Kontext der Fourier-Transformation hat man die Wahl zwischen folgenden zwei Optionen:

- Perfekte Auflösung im Zeit-/Ortsbereich, aber hohe Unschärfe im Frequenzbereich
- Perfekte Auflösung im Frequenzbereich, aber hohe Unschärfe im Zeit-/Ortsbereich

Wenngleich sich eine messbare Größe niemals gleichzeitig beliebig genau im Urbildbereich als auch im Bildbereich der Fourier-Transformation bestimmen lässt, so kann man doch versuchen einen besseren Kompromiss zwischen Auflösung im Urbildbereich und Auflösung im Bildbereich zu finden. Solche Verfahren, die Kurzzeit-Fourier-Transformation und die Wavelet-Transformation, werden in Abschnitt 3.5 vorgestellt.

2.14. Eigenfunktionen linearer Operatoren

In diesem Abschnitt werden die Eigenfunktionen eines linearen Operators vorgestellt. Insbesondere sind die Eigenfunktionen des in Abschnitt 2.12 vorgestellten *Laplacian* beziehungsweise *graph Laplacian* von zentraler Bedeutung beim *geometric deep learning*.

2.14.1. Eigenfunktion

Eigenfunktionen sind in der Funktionalanalysis das Analogon zu den Eigenvektoren in der linearen Algebra. Tatsächlich kann man sich Eigenfunktionen als eine spezielle Art von Eigenvektor vorstellen, da in der Funktionalanalysis der Vektorbegriff auch auf Funktionen angewendet wird. Von Eigenfunktionen statt Eigenvektoren spricht man, wenn ein linearer Operator T auf einem Funktionenraum definiert ist. So wie zu einer linearen Abbildung beziehungsweise Matrix die Eigenvektoren ermittelt werden können, so können zu einem linearen Operator, also einer linearen Abbildung zwischen beispielsweise Funktionenräumen statt Vektorräumen, dessen Eigenfunktionen bestimmt werden.

In der linearen Algebra lässt sich im Fall eines endlichdimensionalen Vektorraums V über einem Körper K, etwa dem Körper $\mathbb{R}$ der reellen Zahlen, jede lineare Abbildung von V in sich selbst beschreiben durch eine quadratische $n \times n$-Matrix $\mathbf{A}$ mit $\dim(V) = n \in \mathbb{N}$. Jede Lösung $\mathbf{x} \neq 0$ des Eigenwertproblems

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{2.12}$$

heißt Eigenvektor der Matrix $\mathbf{A}$ zum Eigenwert λ. Ein Eigenvektor einer Abbildung ist somit ein vom Nullvektor verschiedener Vektor, der durch die Abbildung lediglich skaliert wird. Den Skalierungsfaktor λ nennt man den zum Eigenvektor $\mathbf{x}$ gehörigen Eigenwert der Abbildung beziehungsweise Matrix, hier $\mathbf{A}$.

Eine zu Gleichung 2.12 äquivalente Formulierung ist

$$(\mathbf{A} - \lambda\mathbf{I}) \cdot \mathbf{x} = \mathbf{0},$$

wobei $\mathbf{I}$ die Einheitsmatrix bezeichnet. Aufgrund der Voraussetzung $\mathbf{x} \neq 0$ besteht das Spektrum einer Matrix $\mathbf{A}$, also die Menge ihrer Eigenwerte, somit aus den Lösungen der Gleichung

$$\det(\mathbf{A} - \lambda\mathbf{I}) = \mathbf{0}$$

über dem Körper K. Anders ausgedrückt ist das Spektrum von $\mathbf{A}$ die Menge aller $\lambda \in K$, für die die Matrix $\mathbf{A} - \lambda\mathbf{I}$ nicht invertierbar ist. Eine $n \times n$-Matrix hat höchstens n Eigenwerte.

In der Funktionalanalysis ist die Verallgemeinerung dieses Eigenwertproblems ins Unendlichdimensionale ein Untersuchungsgegenstand. Das Spektrum eines Operators T

besteht aus den Zahlen λ, für die der Operator $(T - \lambda \,\mathrm{id})$, wobei id die Identität bezeichnet, entweder nicht invertierbar ist oder dessen Inverse, falls sie existiert, kein beschränkter Operator ist. Die Elemente des Spektrums eines Operators nennt man Spektralwerte, welche als verallgemeinerte Eigenwerte aufgefasst werden können. Da die untersuchten Vektorräume unendlichdimensional sind, haben lineare Operatoren im Allgemeinen unendlich viele Elemente im Spektrum. Für selbstadjungierte Operatoren wie beispielsweise dem *Laplacian* beziehungsweise *graph Laplacian* liegen die Spektralwerte in $\mathbb{R}$.

Weiterhin muss man im Gegensatz zur linearen Algebra unterscheiden, ob der Operator $(T - \lambda \,\mathrm{id})$, wenn er nicht invertierbar ist, nicht bijektiv, lediglich nicht injektiv oder lediglich nicht surjektiv ist. Im Fall, dass $(T - \lambda \,\mathrm{id})$ nicht injektiv ist, nennt man λ noch einen Eigenwert. In der linearen Algebra ist diese Unterscheidung nicht nötig, da dort eine nicht invertierbare Matrix $\mathbf{A} - \lambda\mathbf{I}$ immer auch nicht bijektiv und somit auch nicht injektiv und nicht surjektiv ist.

Auch wenn diese unendlichdimensional sein können, so können lineare Operatoren doch als Matrizen und Eigenfunktionen als Spaltenvektoren dargestellt werden. Die Konzepte der Eigenwerttheorie aus der linearen Algebra lassen sich somit auf Eigenfunktionen übertragen.

2.14.2. Eigenfunktionen des Laplace-Beltrami-Operators

Ein zentrales Thema dieser Arbeit sind die Eigenfunktionen des *Laplacian* beziehungsweise *graph Laplacian*. Ein Grund dafür ist, dass die Eigenfunktionen des in Abschnitt 2.12 vorgestellten Laplace-Beltrami-Operators als Verallgemeinerung der klassischen Fourier-Basis auf nichteuklidische Gebiete interpretiert werden können.

Im euklidischen eindimensionalen Fall, also $\Omega = \mathbb{R}$, entspricht die Fourier-Basis den Eigenfunktionen des Laplace-Operators, welcher auch als der euklidische *Laplacian* bezeichnet wird. Indem man den Laplace-Operator $\Delta f = -\frac{\partial^2}{\partial x^2} f$ anwendet auf eine Fourier-Basisfunktion $\mathrm{e}^{\mathrm{i}k\cdot x}$ der Fourierreihe erkennt man, dass das Ergebnis $\Delta \mathrm{e}^{\mathrm{i}k\cdot x} = -\mathrm{i}^2 k^2 \mathrm{e}^{\mathrm{i}kx} = k^2 \mathrm{e}^{\mathrm{i}kx}$ wieder die Fourier-Basisfunktion ergibt, allerdings mit einem Vorfaktor k^2. Aus der Eigenwertgleichung $\Delta\phi_i = \lambda_i \phi_i$ mit $i = 0, 1, \ldots$ folgt für den euklidischen Fall, dass es sich bei den Fourier-Basisfunktionen $\mathrm{e}^{\mathrm{i}k\cdot x}$ der Fourierreihe um die Eigenfunktionen des eindimensionalen Laplace-Operators handelt und die erhaltenen Vorfaktoren k^2 die den Eigenfunktionen zugehörigen Eigenwerte sind. Dies bedeutet, dass im eindimensionalen Fall zur Filterung einer Funktion mit einem Filter im Frequenzbereich lediglich die Basisfunktionen skaliert werden müssen. Vor diesem Hintergrund legt man im nichteuklidischen Fall fest, dass die auf nichteuklidischen Gebieten nicht definierten Fourier-Basisfunktionen ebenfalls den Eigenfunktionen des (verallgemeinerten) Laplace-Operators, also denen des Laplace-Beltrami-Operators, entsprechen sollen.

Der Laplace-Beltrami-Operator ist ein selbstadjungierter positiv-semidefiniter Operator und lässt sich wie in Unterabschnitt 2.12.1 beschrieben auf einem kompakten Gebiet somit in eine diskrete Menge bestehend aus orthonormalen Eigenfunktionen zerlegen. Da der Laplace-Beltrami-Operator auf intrinsische Weise definiert wurde, ist auch die von den Eigenfunktionen des Laplace-Beltrami-Operators geformte Basis intrinsisch. Die Menge

der ersten k nach der Dirichlet-Energie (*Dirichlet energy*)

$$\mathcal{E}_{\text{Dir}}(f) = \int_{\mathcal{X}} \|\nabla f(x)\|^2_{T_x\mathcal{X}} \mathrm{d}x = \int_{\mathcal{X}} f(x)\Delta f(x)\mathrm{d}x \tag{2.13}$$

sortierten Eigenfunktionen $\phi_0, \ldots, \phi_{k-1}$, wobei f eine auf einem Gebiet, hier eine Mannigfaltigkeit $\mathcal{X}$, definierte Funktion bezeichnet, bildet eine orthonormale Basis der auf der Mannigfaltigkeit $\mathcal{X}$ definierten Funktionenräume. Aufgrund der positiven Semidefinitheit des *Laplacian* sind dessen Spektralwerte $0 = \lambda_0 \leq \lambda_1 \leq \ldots$ zudem nichtnegativ und können auf nichteuklidischen Gebieten als Analogon zu den Frequenzen in der Fourier-Analysis des euklidischen Falls interpretiert werden. Die klassische Fourier-Basis auf euklidischen Gebieten ist ebenfalls eine orthonormale Basis. Auf nichteuklidischen Gebieten ist die Fourier-Basis jedoch nicht definiert. Wie für die Faltung muss auch eine Verallgemeinerung der Fourier-Basis zunächst definiert werden. Die Eigenbasis des Laplace-Beltrami-Operators lässt sich als eine Verallgemeinerung der Fourier-Basis auf nichteuklidische Gebiete interpretieren und bietet sich daher an. Analog dazu wie auf euklidischen Gebieten eine Funktion als Linearkombination der verschiedenen sinus- und kosinusförmigen Funktionen der Fourier-Basis dargestellt werden kann, werden im nichteuklidischen Fall Funktionen auf die Eigenbasis des Laplace-Beltrami-Operators projiziert.

Im euklidischen Fall entsteht bei der Fourierreihenzerlegung durch Projektion einer periodischen Funktion $f\colon [-\pi, \pi] \to \mathbb{R}$ auf die Fourier-Basisfunktionen $\mathrm{e}^{\mathrm{i}kx}$ mit $k = 0, 1, \ldots$ eine diskrete Menge von Fourier-Koeffizienten $\hat{f}_k := \langle f, \mathrm{e}^{\mathrm{i}kx}\rangle_{L^2([-\pi,\pi])}$. Eine auf einer Mannigfaltigkeit $\mathcal{X}$ definierte quadratintegrierbare Funktion f kann analog dazu auf die Eigenbasis des *Laplacian* projiziert werden:

$$f(x) = \sum_{i\geq 0} \underbrace{\langle f, \phi_i\rangle_{L^2(\mathcal{X})}}_{\hat{f}_i} \phi_i(x) \tag{2.14}$$

Dabei sind die Koeffizienten $\hat{f}_i$ die „Fourier-Koeffizienten“. Die verallgemeinerten Fourier-Koeffizienten $\hat{f}_i$ erhält man durch Berechnung des inneren Produkts der Funktion f und der Eigenfunktion ϕ_i. Das Skalarprodukt $\langle\cdot,\cdot\rangle_{L^2(\mathcal{X})}$ in Gleichung 2.14 ist definiert in Gleichung 2.5 auf Seite 25.

Die „Fourierreihenzerlegung“ einer auf einem nichteuklidischen Gebiet definierten Funktion mittels der Eigenbasis des *Laplacian* beziehungsweise *graph Laplacian*, worüber man die „Fourier-Koeffizienten“ erhält, führt zu einem relativ intuitiven Ansatz zur Definition einer im Spektrum des Laplace-Beltrami-Operators verallgemeinerten Faltung, was die Herangehensweise der in Abschnitt 3.2 beschriebenen Spektralmethoden ist. [Shu+13; Bro+17b; Bro+17a]

3. Ansätze zur Verallgemeinerung der Faltung auf nichteuklidische Gebiete

Dies ist das zentrale Kapitel dieser Arbeit und soll einen umfassenden Einblick in die Techniken des Themengebiets *geometric deep learning* bieten. Es dient weiterhin der Vermittlung eines besseren Verständnisses der in den Kapiteln 4 bis 6 im Detail vorgestellten Verfahren zur Lösung konkreter Anwendungsfälle des *geometric deep learning* und soll den Zusammenhang dieser späteren Kapitel erkennen lassen.

In Abschnitt 1.1 wurde bereits auf die geschichtliche Entwicklung von *geometric deep learning* sowie des Lernens auf nichteuklidischen Gebieten (*non-Euclidean domains*) eingegangen und aufgeführt wie faltungsbasierte Modelle sich zeitlich in diesen Kontext einordnen lassen. In diesem Kapitel werden die vier verschiedenen Herangehensweisen zur Formulierung künstlicher neuronaler Netzwerke erläutert, welche auf ein verallgemeinertes Konzept der klassischen Faltung setzen. Zu diesem Zweck werden in diesem Kapitel jeweils auch ausgewählte relevante Verfahren kurz vorgestellt, die dem entsprechenden Ansatz folgen. Dabei wird auch auf die Stärken und Schwächen der jeweiligen Ansätze eingegangen sowie welche Hürden alternativer Vorgehensweisen mit dem jeweiligen Ansatz überwunden werden können.

3.1. Übersicht

Der entscheidende Unterschied zwischen den vier in diesem Kapitel präsentierten Herangehensweisen ist die Weise, auf die eine der klassischen Faltung ähnliche Operation formuliert wird um das Konzept der von euklidischen Gebieten bekannten Faltung auf nichteuklidische Gebiete wie Graphen und Mannigfaltigkeiten zu übertragen.

Die ersten Verfahren zur Formulierung der Faltung auf nichteuklidische Gebiete definieren eine solche Verallgemeinerung der klassischen in Abschnitt 2.7 vorgestellten Faltung im Spektralbereich (*spectral domain*), einem Begriff aus der spektralen Graphentheorie, welcher das Analogon zum in der Fourier-Analyse gebräuchlichen klassischen Frequenzbereich ist. Einen guten Einstieg in die Signalverarbeitung auf Graphen bieten beispielsweise Shuman u. a. [Shu+13]. Dieser in Abschnitt 3.2 beschriebene Ansatz der Spektralmethoden ist eine direkte Interpretation der Intuition hinter der aus der klassischen Signalverarbeitung bekannten Faltung. Denn nach dem Faltungstheorem ist die klassische Faltung ein linearer Operator, der von der Fourier-Basis des betrachteten Funktionenraums diagonalisiert wird und umgekehrt ist auch jeder linearer Operator, der von der Fourier-Basis diagonalisiert wird, eine Faltung. Mit diesem Wissen braucht lediglich nach einem linearen Operator gesucht werden, der sich ausdrücken lässt durch die Hintereinanderausführung $\mathbf{\Phi}\,\mathrm{diag}(\hat{\mathbf{g}})\mathbf{\Phi}^\mathsf{T}$ der Fourier-Transformation $\mathbf{\Phi}^\mathsf{T}$, der Multiplikation mit

der Diagonalmatrix bestehend aus den Fourier-Koeffizienten des Filters $\mathbf{g}$ und abschließend der inversen Fourier-Transformation $\mathbf{\Phi}$.

Auch wenn diese frühe Herangehensweise relativ intuitiv ist, so hat sie dennoch entscheidende schwerwiegende Nachteile, was zu ihrer Ablösung durch die Spektrum-freien Methoden führte, welche in Abschnitt 3.3 behandelt werden. An dieser Stelle sei erwähnt, dass der Filter sowohl eine Repräsentation im Spektralbereich (*spectral domain*) als auch eine Repräsentation im Ortsbereich (*spatial domain*) hat. Die Arbeit mit der Repräsentation im Spektralbereich, also den Fourier-Koeffizienten des Filters, was das Prinzip der Spektralmethoden ist, ist äußerst ineffizient. Die Spektralmethoden haben die Laufzeitkomplexität $O(n^2)$. Dahingegen ist die Arbeit mit der Repräsentation des Filters im Ortsbereich deutlich effizienter. Zudem löst diese räumliche Repräsentation noch einige weitere Probleme der Spektralmethoden. Bei den Spektrum-freien Methoden wird die verallgemeinerte Faltung wie schon zuvor bei den Spektralmethoden über das Faltungstheorem im Spektralbereich definiert, jedoch wird für die Arbeit mit dem Filter zur Filterung anderer Signale nicht länger dessen Repräsentation im Spektralbereich verwendet, sondern dessen räumliche Repräsentation. Ihren Ursprung haben die Spektrum-freien Methoden vor den Spektralmethoden, da sie bereits im Jahr 2005 von Gori, Monfardini und Scarselli [GMS05] thematisiert wurden. Allerdings fand dieser Spektrum-freie Ansatz zunächst kaum Beachtung und trat erst nach dem Aufkommen den Spektralmethoden wieder in Erscheinung, um die Probleme der Spektralmethoden umgehen zu können. Chronologisch sind die in Abschnitt 3.3 vorgestellten Spektrum-freien Verfahren daher nach den in Abschnitt 3.2 beschriebenen Spektralmethoden einzuordnen.

Einen gänzlich anderen Ansatz, der sich parallel zu den in Abschnitt 3.3 vorgestellten, aus der Signalverarbeitung auf Graphen stammenden Spektrum-freien Methoden aus anderen Anforderungen heraus in den Bereichen der Computergrafik sowie des maschinellen Sehens (*computer vision*) entwickelte, verfolgen die sogenannten kartografierenden Methoden (*charting-based methods*), welche in Abschnitt 3.4 behandelt werden und derzeit die gebräuchlichste Herangehensweise zur Formulierung faltungsbasierter neuronaler Netzwerkmodelle für die Verwendung auf nichteuklidischen Gebieten darstellen. Bei dieser Verfahrensweise wird, anders als bei den Spektralmethoden und Spektrum-freien Methoden, die Faltung nicht im Spektralbereich definiert, sondern im Ortsbereich und somit ausschließlich im Ortsbereich gearbeitet, nicht bloß mit der räumlichen Repräsentation einer im Spektralbereich definierten Operation. Die kartografierenden Verfahren entsprechen in ihrer Funktionsweise von allen vier in diesem Kapitel vorgestellten Ansätzen am ehesten der von der klassischen Faltung bekannten Vorstellung, einen Faltungskern (*convolution kernel*) in der Manier eines gleitenden Fensters (*sliding window*) über ein Signal zu schieben und die Filterkoeffizienten dem abgedeckten Bereich (*patch*) des Signals zuzuordnen.

Wenngleich die Spektrum-freien Methoden mit der räumlichen Repräsentation eines Filters arbeiten, ist die verallgemeinerte Faltung dennoch im Spektralbereich definiert und diese Art von Verfahren arbeiten ausschließlich mit Informationen aus dem Spektralbereich. Die kartografierenden Methoden hingegen greifen ausschließlich auf Informationen aus dem Ortsbereich zurück. Der Grund, warum diese beiden Ansätze sich auf Informationen aus nur einem der beiden Bereiche beschränken, hat seinen Ursprung in einer Unschärferelation zwischen diesen beiden Bereichen. Ein vierter Ansatz zur Verallgemeinerung

der Faltung verfolgt die Idee, dieser Unschärferelation (*uncertainty principle*) entgegenzuwirken um Informationen aus beiden Bereichen verwenden zu können. Dieses Prinzip wird in der Signalverarbeitung auch als Zeit-Frequenz-Analyse bezeichnet. Da in dieser Arbeit keine von der Zeit abhängigen Funktionen (beispielsweise Schallwellen), sondern örtliche Signale – beispielsweise ist ein Bildsignal abhängig von den Pixelkoordinaten – untersucht werden, spricht man in diesem Kontext auch von *spatio-frequency analysis*. Dieser Ansatz wird in Abschnitt 3.5 vorgestellt.

Als Quelle für die in diesem Kapitel erläuterten Ansätze dient vor allem die wegweisende und enorm einflussreiche Arbeit [Bro+17b] von Bronstein u. a. aus dem Jahr 2017, durch welche der in [Eur17] von Bronstein u. a. geprägte Begriff *geometric deep learning* Verbreitung fand und welche dem interessierten Leser nachdrücklich zur weiteren Lektüre empfohlen wird. Auch [Bro19] von Bronstein wird zur Vertiefung empfohlen.

3.2. Spektralmethoden

Die Spektralmethoden folgen dem frühen Ansatz, die Faltung im Spektrum des *Laplacian* beziehungsweise *graph Laplacian* zu definieren. Ihre Verfahrensweise ist eine direkte Interpretation der Intuition hinter der aus der klassischen Signalverarbeitung bekannten Faltung. Die Orientierung an bekannten Konzepten aus der klassischen Signalverarbeitung wie etwa dem Faltungstheorem ist naheliegend für die ersten Versuche, die Faltung auf nichteuklidische Gebiete zu verallgemeinern. Jedoch bringt dieser frühe Ansatz einige schwerwiegende Probleme mit sich, weshalb die Spektralmethoden sich nicht für den Einsatz in der Praxis eignen. Für ein abgerundetes Gesamtbild und die in Abschnitt 3.3 vorgestellten Spektrum-freien Methoden ist es dennoch lohnenswert, das Funktionsprinzip der Spektralmethoden, etwa [Bru+14; HBL15], zu verinnerlichen.

Als nichteuklidisches Gebiet Ω dient in diesem Abschnitt eine Mannigfaltigkeit $\Omega = \mathcal{X}$. Folglich werden in diesem Abschnitt Funktionen $f \in L^2(\mathcal{X})$ untersucht, die dem Funktionenraum L^2 bestehend aus den quadratintegrierbaren Funktionen mit Definitionsbereich $\mathcal{X}$ entstammen.

3.2.1. Grundlegendes Prinzip

Von zentraler Wichtigkeit für ein gutes Verständnis der Spektralmethoden ist das Verständnis der in Abschnitt 2.13 vorgestellten Fourier-Analysis. Insbesondere das in Unterabschnitt 2.13.3 auf Seite 35 beschriebene Faltungstheorem sollte verstanden sein. Denn die in Abschnitt 2.7 auf Seite 15 definierte klassische Faltung ist auf nichteuklidischen Gebieten nicht anwendbar, da sie mangels Translationsäquivarianz nicht wohldefiniert ist. Die Definition der klassischen Faltung in Gleichung 2.1 setzt voraus, dass man von einem Punkt x des Gebiets andere Punkte τ des Gebiets subtrahieren kann. Auf einem Graph oder einer Mannigfaltigkeit ohne Vektorraumstruktur ist jedoch nicht einmal die simple Operation $x - \tau$ definiert, was ein weiterer Grund ist, weshalb die Definition in Gleichung 2.1 nicht anwendbar ist. Hier kommt das Faltungstheorem ins Spiel.

Das Faltungstheorem erlaubt es, die Faltung $\mathbf{f} * \mathbf{g}$ einer Funktion $\mathbf{f}$ mit einem Faltungskern $\mathbf{g}$ über das punktweise Produkt der Fourier-Transformierten $\mathcal{F}\{f\}$ und $\mathcal{F}\{g\}$ zu

realisieren. Wichtiger für diesen Abschnitt ist aber, dass nach dem Faltungstheorem die klassische Faltung ein linearer Operator ist, welcher von der Fourier-Basis des für das Gebiet Ω definierten Funktionenraums diagonalisiert wird und umgekehrt sind auch diejenigen linearen Operatoren, die von der Fourier-Basis diagonalisiert werden, Faltungsoperatoren. Dadurch, dass die Fourier-Transformation $\mathcal{F}$ den Faltungsoperator $*$ diagonalisiert, lässt die Faltung $\mathbf{f} * \mathbf{g}$ sich alternativ als ein Matrix-Vektor-Produkt $\mathbf{\Phi}\,\mathrm{diag}(\hat{g}_1, \ldots, \hat{g}_n)\mathbf{\Phi}^\mathsf{T}\mathbf{f}$ im Frequenzbereich schreiben. Dies kann genutzt werden, um ein Analogon zur Faltung für den Fall eines nichteuklidischen Definitionsbereichs beziehungsweise Gebiets zu entwickeln.

3.2.2. Formulierung der Faltung als Operation im Frequenzbereich

In diesem Abschnitt werden Vorüberlegungen zur Verallgemeinerung der Faltung im Spektralbereich des *Laplacian* beschrieben. Der Begriff *Operator* dient lediglich der begrifflichen Abgrenzung der Abbildungen zwischen Funktionenräumen von Abbildungen zwischen Vektorräumen. Ein linearer Operator ist somit nichts anderes als eine lineare Abbildung und lineare Abbildungen lassen sich im endlichdimensionalen Fall durch Matrizen beschreiben. Da die klassische Faltung $\mathbf{f} * \mathbf{g}$ zweier Vektoren $\mathbf{f} = (f_1, \ldots, f_n)^\mathsf{T}$ und $\mathbf{g} = (g_1, \ldots, g_n)^\mathsf{T}$ ein *linearer* Operator ist, kann man sich die klassische Faltung vorstellen als ein Matrix-Vektor-Produkt $\mathbf{G}\,\mathbf{f}$, wobei $\mathbf{G}$ die folgende Matrix ist:

$$\mathbf{G} = \begin{pmatrix} g_1 & g_2 & \cdots & \cdots & g_n \\ g_n & g_1 & g_2 & \cdots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \cdots & g_1 & g_2 \\ g_2 & g_3 & \cdots & \cdots & g_1 \end{pmatrix}$$

Bei dieser Matrix $\mathbf{G}$ sind die Haupt- und Nebendiagonalen konstant, weshalb es sich um eine Toeplitz-Matrix handelt. Da die Matrix zudem quadratisch ist und jeder Zeilenvektor relativ zum darüberliegenden Zeilenvektor um einen Eintrag nach rechts verschoben ist, handelt es sich sogar um eine zyklische Matrix. In der in Unterabschnitt 2.7.3 verwendeten Notation entspricht diese Matrix $\mathbf{G}$ der zyklischen Matrix $\mathbf{C_a}$ mit $\mathbf{a} = (g_1, g_n, \ldots, g_3, g_2)^\mathsf{T}$. Die Matrix $\mathbf{G}$ hat diese zyklische Gestalt aufgrund der in Unterabschnitt 2.6.1 beschriebenen Translationsäquivarianz, da ein Operator beziehungsweise eine Matrix wie in Unterabschnitt 2.7.3 beschrieben genau dann eine zyklische Struktur hat, wenn er beziehungsweise sie kommutiert mit Translation, also translationsäquivariant ist. Der Nutzen dieser alternativen Schreibweise $\mathbf{f} * \mathbf{g} = \mathbf{G}\,\mathbf{f}$ für die klassische Faltung im euklidischen Fall ist, dass zyklische Matrizen die besondere Eigenschaft haben, von der Fourier-Basis diagonalisiert zu werden, woraus sich das Faltungstheorem herleitet. Es existiert also eine zur Matrix $\mathbf{G}$ ähnliche Diagonalmatrix $\mathrm{diag}(\hat{\mathbf{g}})$ mit den Fourier-Koeffizienten des auf die Fourier-Basis projizierten Vektors $\mathbf{g}$ auf der Hauptdiagonalen.

Auf diskreten Gebieten hat die Fourier-Transformation eine Matrix-Form, die als Basiswechselmatrix fungiert. Die Fourier-Basisvektoren lassen sich als Matrix $\mathbf{\Phi}$ schreiben und da die Fourier-Basis eine Orthonormalbasis ist, ist $\mathbf{\Phi}$ eine Orthogonalmatrix. Für Orthogonalmatrizen entspricht deren Inverse ihrer Transponierten. In Abschnitt 2.14 ist die Projektion einer Funktion auf die Fourier-Basisfunktionen beschrieben, wobei eine

diskrete Menge von Fourier-Koeffizienten $\hat{f}_k$ mit $k = 1, \ldots, n$ entsteht. Somit folgt aus der Matrix-Vektor-Notation $\mathbf{f} * \mathbf{g} = \mathbf{G}\,\mathbf{f}$ der klassischen Faltung die folgende Gleichung:

$$\mathbf{f} * \mathbf{g} = \mathbf{G}\,\mathbf{f} = \boldsymbol{\Phi} \begin{pmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{pmatrix} \boldsymbol{\Phi}^{\mathsf{T}} \mathbf{f} = \boldsymbol{\Phi} \begin{pmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{pmatrix} \begin{pmatrix} \hat{f}_1 \\ \vdots \\ \hat{f}_n \end{pmatrix} = \boldsymbol{\Phi} \begin{pmatrix} \hat{f}_1 \cdot \hat{g}_1 \\ \vdots \\ \hat{f}_n \cdot \hat{g}_n \end{pmatrix} \tag{3.1}$$

Die Multiplikation mit der Matrix $\boldsymbol{\Phi}^{\mathsf{T}}$ entspricht der Fourier-Transformation $\mathcal{F}$, die Multiplikation mit der Matrix $\boldsymbol{\Phi}$ entspricht der inversen Fourier-Transformation $\mathcal{F}^{-1}$. Auf diese Weise lässt sich die klassische Faltung umformulieren als eine von der Fourier-Basis $\boldsymbol{\Phi}$ abhängige Operation im Frequenzbereich. Auf dieser Vorüberlegung aufbauend wird im nächsten Abschnitt ganz analog die auf nichteuklidische Gebiete verallgemeinerte Faltung im Spektrum des Laplace-Beltrami-Operators $\boldsymbol{\Delta}$ formuliert. [Bro+17b; Bro+17a]

3.2.3. Definition einer verallgemeinerten Faltung als Operation im Spektrum

Zentral für das Verständnis der Verallgemeinerung der Faltung im Spektralbereich ist, dass eine quadratintegrierbare Funktion $f \in L^2(\mathcal{X})$ aus dem Raum L^2 der für eine Mannigfaltigkeit $\mathcal{X}$ definierten Funktionen sich mit

$$f(x) = \sum_{i=1}^{n} \underbrace{\langle f, \phi_i \rangle_{L^2(\mathcal{X})}}_{\hat{f}_i} \phi_i(x) \tag{3.2}$$

in eine „Fourierreihe“ zerlegen lässt, wobei $\phi_i(x)$ die in Abschnitt 2.14 auf Seite 37 beschriebenen Eigenfunktionen des *Laplacian* bezeichnet und $\langle \cdot, \cdot \rangle_{L^2\mathcal{X}}$ das in Gleichung 2.5 auf Seite 25 definierte innere Produkt ist, mit welchem der Funktionenraum L^2 versehen wurde. Diese Zerlegung einer auf einem nichteuklidischen Gebiet Ω definierten Funktion $f \in \Omega$ in eine verallgemeinerte Fourierreihe ist möglich, weil im nichteuklidischen Fall die Eigenbasis des *Laplacian* als Verallgemeinerung der Fourier-Basis festgelegt ist. In Verbindung mit den Überlegungen in Abschnitt 2.14 lässt sich auch auf nichteuklidischen Gebieten die in Gleichung 3.2 definierte „Fourierreihenzerlegung“ herleiten. Durch Projektion der Funktion f auf die Basisfunktionen, im Fall nichteuklidischer Gebiete eben die Eigenbasis $\boldsymbol{\Phi} = (\phi_1, \ldots, \phi_n)$ (hier in Form einer Basiswechselmatrix mit den Eigenfunktionen als Spalten) des *Laplacian* bestehend aus den Eigenfunktionen $\phi_i(x)$, entsteht eine diskrete Menge von „Fourier-Koeffizienten“ $\hat{f}_1, \ldots, \hat{f}_n$. Diese Projektion auf die Basisfunktionen verallgemeinert die Fourier-Transformation $\mathcal{F}$, auch Fourier-Analyse genannt, aus der klassischen Signalverarbeitung. Die auf diese Weise gewonnenen „Fourier-Koeffizienten“ $\hat{f}_i$ einer Funktion f sind von großem Interesse, da man mit ihnen über die von ihnen repräsentierten, in Gleichung 3.2 gezeigten inneren Produkte ganz analog zu Gleichung 3.1

folgende Gleichung aufstellen kann:

$$\begin{aligned}\mathbf{f} * \mathbf{g} = \mathbf{\Phi} \begin{pmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{pmatrix} \mathbf{\Phi}^\mathsf{T} \mathbf{f} &= \mathbf{\Phi} \begin{pmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{pmatrix} \begin{pmatrix} \langle f_1, \phi_1 \rangle_{L^2(\mathcal{X})} \\ \vdots \\ \langle f_n, \phi_n \rangle_{L^2(\mathcal{X})} \end{pmatrix} \\ &= \mathbf{\Phi} \begin{pmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{pmatrix} \begin{pmatrix} \hat{f}_1 \\ \vdots \\ \hat{f}_n \end{pmatrix} = \mathbf{\Phi} \begin{pmatrix} \hat{f}_1 \cdot \hat{g}_1 \\ \vdots \\ \hat{f}_n \cdot \hat{g}_n \end{pmatrix}\end{aligned} \tag{3.3}$$

Da der *Laplacian* ein selbstadjungierter positiv-semidefiniter Operator ist, sind dessen Eigenfunktionen $\phi_1, \ldots, \phi_n$ orthonormal und $\mathbf{\Phi}$ ist somit eine Orthogonalmatrix. Die Inverse von $\mathbf{\Phi}$ entspricht also auch in Gleichung 3.3 ihrer Transponierten. Da auf nichteuklidischen Gebieten keine Translationsäquivarianz gegeben ist, ist die Matrix $\mathbf{G} := \mathbf{\Phi}\,\mathrm{diag}(\hat{\mathbf{g}})\mathbf{\Phi}^\mathsf{T}$ in Gleichung 3.3 jedoch anders als in Gleichung 3.1 keine zyklische Matrix.

Durch Substituieren der „Fourier-Koeffizienten“ $\hat{f}_i$ und $\hat{g}_i$ durch das jeweilige von ihnen repräsentierte innere Produkt, welches in Gleichung 3.2 gezeigt ist, kommt man schließlich auf die auf nichteuklidische Gebiete, im Spektrum des *Laplacian* definierte, verallgemeinerte Faltung:

$$(\mathbf{f} * \mathbf{g})(x) = \sum_{i \geq 1} \underbrace{\langle \mathbf{f}, \phi_i \rangle_{L^2(\mathcal{X})}}_{\hat{f}_i} \underbrace{\langle \mathbf{g}, \phi_i \rangle_{L^2(\mathcal{X})}}_{\hat{g}_i} \phi_i(x) \tag{3.4}$$

Das Summieren der Produkte $\hat{f}_i\,\hat{g}_i\,\phi_i(x)$ verallgemeinert die inverse Fourier-Transformation $\mathcal{F}^{-1}$, auch Fourier-Synthese genannt, aus der klassischen Signalverarbeitung. Mit Gleichung 3.4 existiert ein relativ intuitiver Weg, die klassische Faltung auf nichteuklidische Gebiete zu verallgemeinern. Zum Verständnis der in Unterabschnitt 3.2.5 beschriebenen Nachteile der Spektralmethoden als auch dem fundamentalen Nachteil der in Abschnitt 3.3 vorgestellten Spektrum-freien Methoden ist es wichtig sich vor Augen zu führen, dass diese Formulierung der Faltung im Spektrum des *Laplacian* von der durch die Matrix $\mathbf{\Phi}$ repräsentierten Eigenbasis des *Laplacian* abhängig ist. [Bro+17b; Bro+17a]

3.2.4. *Spectral CNN (SCNN)*

Nachdem mit Gleichung 3.4 eine Verallgemeinerung der Faltung für Mannigfaltigkeiten hergeleitet wurde, wird in diesem Abschnitt nun am Beispiel der Arbeit von Bruna u. a. [Bru+14] aus dem Jahr 2014, der ersten Spektralmethode, für einen ungerichteten Graphen präsentiert, wie eine spektrale Faltungsschicht zur Verwendung in einem neuronalen Netzwerk, hier einem *graph neural network*, definiert werden kann.

Gleichung 3.4 arbeitet mit dem *Laplacian* von Mannigfaltigkeiten. Da $\mathbf{G} := \mathbf{\Phi}\,\mathrm{diag}(\hat{\mathbf{g}})\mathbf{\Phi}^\mathsf{T}$ in Gleichung 3.4 wegen $\mathbf{G}\,\mathbf{\Delta}\mathbf{f} = \mathbf{\Delta}\mathbf{G}\,\mathbf{f}$ mit dem *Laplacian* kommutiert, kann für eine Verallgemeinerung der Faltung auf Graphen also ein linearer Operator gesucht werden, der mit dem *graph Laplacian* kommutiert. Dies führt über die Eigenvektoren des *graph Laplacian* zu einer Verallgemeinerung der Faltung analog zu Gleichung 3.4. Allerdings sind im Fall von

Graphen in der Praxis nur die ersten k der nach Glattheit im Sinne der in der Potentialtheorie bekannten und in Gleichung 2.13 auf Seite 39 für Mannigfaltigkeiten $\mathcal{X}$ definierten Dirichlet-Energie (*Dirichlet energy*) sortierten n Eigenvektoren des *graph Laplacian* von Nutzen, wobei $n = |\mathcal{V}|$ die Anzahl der Knoten im Graphen ist. Deshalb werden diejenigen Eigenvektoren mit einem Index größer als einem wählbaren Index k verworfen, was dem Festlegen einer *cutoff* Frequenz gleichkommt. Glattheit bedeutet, dass eine Funktion ϕ für ein nur geringfügig abweichendes Argument x' keinen allzu stark von $\phi(x)$ abweichenden Funktionswert $\phi(x')$ liefern soll. Typischerweise ist k sehr viel kleiner als n, also $k \ll n$, da schon relativ früh der Wert $\phi_i(x')$ mit $i = 1, \ldots, n$ zu sehr von $\phi_i(x)$ abweicht. Die Arbeit von Shuman u. a. [Shu+13] verschafft einen guten Überblick über die *graph Laplacian quadratic form* genannte, von Spielman [Spi12] vorgestellte diskretisierte *Dirichlet form*, welche die totale Variation einer Funktion misst und ein Maß für das lokale Schwingungsverhalten der Funktion ist. Bei den Fourier-Basisfunktionen, $\mathrm{e}^{\mathrm{i}kx}$ mit $k = 0, 1, \ldots$ im Fall der Fourierreihe, handelt es sich um Funktionen, die mit einer gewissen Frequenz schwingen, welche sich von Fourier-Basisfunktion zu Fourier-Basisfunktion erhöht. Im Fall eines Graphen sind die „Fourier-Basisfunktionen“ definiert durch die Eigenvektoren des *graph Laplacian*. Die mittels der *graph Laplacian quadratic form* gemessene Dirichlet-Energie (*Dirichlet energy*) lässt sich zur Verallgemeinerung des Konzepts von Frequenz auf Graphen nutzen.

Bruna u. a. beschränken sich durch Wahl der im Sinne der Dirichlet-Energie glattesten k Eigenvektoren auf diejenigen k Vektoren mit der geringsten Frequenz. Denn je mehr Eigenvektoren hinzugenommen werden, umso größer werden im Spektrum auch die Frequenzen, die hinzugenommen werden. Insbesondere für Eigenvektoren mit hoher Frequenz kann schon eine kleine Verformung des Gebiets eine sehr große Veränderung der Eigenvektoren hervorrufen. Dies äußert sich in unerwünschter Instabilität des Filters unter Verformung des Gebiets. In Abschnitt 3.3 werden deshalb Methoden vorgestellt, die nicht ausdrücklich mit dem Spektrum arbeiten, sondern stattdessen eine räumliche Repräsentation als Alternative zu der hier verwendeten spektralen Repräsentation des Filters verwenden.

Bruna u. a. [Bru+14] definieren ihre Faltungsschicht ganz analog zur klassischen Faltungsschicht, die durch

$$g_l(x) = \xi\left(\sum_{l'=1}^{p} \left(f_{l'} * \gamma_{l,l'}\right)(x)\right) \tag{3.5}$$

definiert ist, wobei ξ eine Nichtlinearität wie die Aktivierungsfunktion *ReLU (Rectified Linear Unit)* ist. Der in einer klassischen Faltungsschicht zu verwendende Faltungskern $\gamma_{l,l'}$ wird über die Indizes $l = 1, \ldots, q$ und $l' = 1, \ldots, p$ bestimmt. Das p-dimensionale Eingangssignal auf den Knoten des Graphen wird repräsentiert durch die $n \times p$-Matrix $\mathbf{F} = (\mathbf{f}_1, \ldots, \mathbf{f}_p)$, das q-dimensionale Ausgangssignal durch die $n \times q$-Matrix $\mathbf{G} = (\mathbf{g}_1, \ldots, \mathbf{g}_q)$. Ganz ähnlich zu Gleichung 3.5 definieren Bruna u. a. ihre *spectral convolutional layer*, indem sie die klassische Faltung $f_{l'} * \gamma_{l,l'}$ durch die verallgemeinerte Faltung, ausgedrückt in der inzwischen bekannten Matrix-Vektor-Notation, ersetzen, wobei die $k \times k$-Matrix $\mathbf{\Gamma}_{l,l'}$ der sogenannte *spectral multiplier operator* ist, der einen Filter im Spektrum des *graph Laplacian* repräsentiert und $\mathbf{\Phi} = (\phi_1, \ldots, \phi_n)$ die Eigenbasis bestehend aus den n Eigenvektoren des *graph*

Laplacian bezeichnet:

$$\mathbf{g}_l = \xi\left(\sum_{l'=1}^{p} \mathbf{\Phi}_k \mathbf{\Gamma}_{l,l'} \mathbf{\Phi}_k^{\mathsf{T}} \mathbf{f}_{l'}\right) \tag{3.6}$$

Auf diesem Wege werden von euklidischen Faltungsnetzwerken bekannte Konstrukte auf nichteuklidische Gebiete anwendbar gemacht, was der Zweck des *geometric deep learning* ist. Weitere Optimierungen dieses frühen Verfahrens, wie sie in [Bru+14] oder auch in [Bro+17b] angeführt werden, sind in Unterabschnitt 3.2.6 beschrieben.

3.2.5. Nachteile von Spektralmethoden

Wie eingangs erwähnt haben Spektralmethoden, also Verfahren, die nicht nur die Faltung im Spektrum des *Laplacian* definieren, sondern auch noch mit der spektralen statt mit einer räumlichen Repräsentation des Filters arbeiten, mehrere schwerwiegende Nachteile. Diese Nachteile der Spektralmethoden sind so fundamental, dass die Spekralmethoden sich nicht für den Einsatz in der Praxis eignen, auch wenn sie mit ihren Konzepten einen wichtigen Beitrag zur Entwicklung des Themenfelds *geometric deep learning* geleistet haben.

Die Faltung in ihrer Matrix-Vektor-Notation $\mathbf{f} * \mathbf{g} = \mathbf{G}\,\mathbf{f} = \mathbf{\Phi}\,\mathrm{diag}(\hat{g}_1, \ldots, \hat{g}_n)\mathbf{\Phi}^{\mathsf{T}}\mathbf{f}$ hat die ineffiziente quadratische Laufzeitkomplexität $O(n^2)$. Im euklidischen Fall lässt sich dieser hohe Rechenaufwand dank effizienter Algorithmen wie der schnellen Fourier-Transformation reduzieren zu $O(n \log n)$. Für nichteuklidische Gebiete existieren solche effizienten Algorithmen jedoch nicht, weshalb die Laufzeitkomplexität sich auf nichteuklidischen Gebieten nicht reduzieren lässt und es bei einer wenig zufriedenstellenden Komplexität $O(n^2)$ bleibt.

Die Spektralmethoden haben neben dem beträchtlichen Rechenaufwand weitere entscheidende Nachteile, wie etwa, dass die Anzahl der in den Faltungsschichten jeweils erforderlichen Parameter von der Größe der Eingabe abhängt, was die Parameteranzahl des neuronalen Netzwerks und damit auch den Rechenaufwand in die Höhe treibt, vor allem aber auch die Gefahr des *overfitting* erhöht. *Overfitting* ist das Problem, dass ein Modell zu komplex für die zur Verfügung stehenden Trainingsdaten ist und somit nicht in der Lage ist auf nie zuvor gesehene Daten zu generalisieren.

Weiterhin ist bei Spektralmethoden die räumliche Repräsentation des Filters nicht lokalisiert, was gerade eine der großen Stärken der euklidischen Faltungsnetzwerke ist. Die Neuronen klassischer Faltungsnetze haben ein kleines rezeptives Feld (*receptive field*) und sind hierarchisch in Schichten angeordnet, was dem visuellen Cortex [HW62] nachempfunden ist. Auf diese Weise lassen sich Kanten zu Umrissen zusammensetzen und diese wiederum zu Nasen, Augen, et cetera, und diese wiederum zu Gesichtern, und so weiter. Bei Spektralmethoden ist diese Lokalisierung der Filter im Ortsbereich nicht gegeben.

Zudem akzeptieren Spektralmethoden nur ungerichtete Graphen, da gerichtete Graphen keine symmetrische Laplace-Matrix (*graph Laplacian*, siehe Unterabschnitt 2.12.3) haben und somit keine Zerlegung in orthogonale Eigenvektoren möglich ist, womit für gerichtete Graphen keine intuitive Konstruktion der Faltung im Spektralbereich möglich ist.

Das vielleicht schwerwiegendste Problem der Definition der Faltung im Spektralbereich ist, dass die Filterkoeffizienten abhängig von der gewählten Basis sind. Eine Spektralme-

thode, die ihre spektralen Filterkoeffizienten auf einem Gebiet $\mathcal{X}$ gelernt hat, funktioniert nur auf $\mathcal{X}$ wie beabsichtigt. Auf einem anderen Gebiet $\mathcal{Y}$ arbeitet die Spektralmethode aufgrund einer anderen Basis Ψ nicht mehr korrekt, die Anwendung der auf $\mathcal{X}$ gelernten spektralen Filterkoeffizienten führt auf $\mathcal{Y}$ zu anderen Ergebnissen als auf $\mathcal{X}$. Dies macht die Spektralmethoden ungeeignet für viele Anwendungen der Computergrafik, bei denen beispielsweise Übereinstimmungen zwischen verschiedenen Formen gefunden werden sollen (*shape correspondence*). In diesem Abschnitt und auch in Abschnitt 3.3 wird daher von einem unveränderlichen Gebiet ausgegangen. Auch bedeutet die Arbeit mit der Repräsentation des Filters im Spektralbereich, dass eine Verformung des Gebiets Instabilitäten zur Folge hat. Dies wird in Abbildung 3.1 veranschaulicht.

Abbildung 3.1.: **Illustration der Instabilität im Spektralbereich definierter Filter unter Deformation.** Links: eine auf einer Mannigfaltigkeit $\mathcal{X}$ definierte Funktion $\mathbf{f}$, deren Funktionswerte durch Farbe repräsentiert werden und sich als rot dargestellte Bereiche äußern. Mitte: dieselbe Funktion $\mathbf{f}$ auf derselben Mannigfaltigkeit $\mathcal{X}$ mit einem im Spektrum des *Laplacian* definierten Kantenerkennungsfilter $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T}$ gefiltert, wobei $\hat{\mathbf{W}}$ die Diagonalmatrix ist, die die auf der Mannigfaltigkeit $\mathcal{X}$ mit Basis $\mathbf{\Phi}$ gelernten Filterkoeffizienten enthält (nota bene die blau dargestellten Bereiche). Rechts: obwohl die Mannigfaltigkeit $\mathcal{X}$ mit Basis $\mathbf{\Phi}$ nur geringfügig verformt wurde, handelt es sich im rechten Bild nach der Verformung um eine andere Mannigfaltigkeit $\mathcal{Y}$ mit anderer Basis $\mathbf{\Psi}$; der auf $\mathcal{X}$ wie gewünscht arbeitende Kantenerkennungsfilter produziert nach Verformung der Mannigfaltigkeit $\mathcal{X}$ aufgrund der anderen, nicht länger zu den gelernten Filterkoeffizienten in $\hat{\mathbf{W}}$ passenden Basis $\mathbf{\Psi}$ von den erwünschten Ergebnissen verschiedene Resultate auf der neuen Mannigfaltigkeit $\mathcal{Y}$ bei der Filterung von $\mathbf{f}$ mit $\mathbf{\Psi}\hat{\mathbf{W}}\mathbf{\Psi}^\mathsf{T}$; der im Spektrum des *Laplacian* definierte Filter ist instabil unter Deformation. [Bro+21a]

In Unterabschnitt 3.2.6 und Abschnitt 3.3 werden Ideen vorgestellt, welche einige der Nachteile der Spektralmethoden – allerdings nicht alle – lösen oder wenigstens umgehen können. Die in Abbildung 3.1 illustrierte Abhängigkeit von der Basis können auch die Spektrum-freien Methoden nicht beseitigen, weshalb die Spektrum-freien Methoden ebenfalls nicht über mehrere Gebiete hinweg korrekt arbeiten. [Bro+17b]

3.2.6. Ansatz für im Ortsbereich lokalisierte Filter

Dieser Abschnitt behandelt eine Optimierung des Grundprinzips der Spektralmethoden, die einige der in Unterabschnitt 3.2.5 beschriebenen Probleme angeht und als Vorbereitung auf die in Abschnitt 3.3 vorgestellten Spektrum-freien Methoden von Wichtigkeit ist. Mittels dieser Verbesserung wird die Gefahr des *overfitting* bei Spektralmethoden reduziert und eine bessere Lokalisierung der Filter im Ortsbereich erzielt. Das fundamentale Problem der Spektralmethoden, die rechenaufwändigen Multiplikationen mit $\mathbf{\Phi}^\mathsf{T}$ beziehungsweise $\mathbf{\Phi}$, bleibt jedoch weiterhin bestehen und kann auch mit der in diesem Abschnitt erläuterten Optimierung nicht gelöst werden. Die Spektrum-freien Verfahren hingegen können dieses Problem umgehen, wenn auch nicht lösen. Dazu bauen sie stark auf die in diesem Abschnitt dargelegte Idee auf.

In Unterabschnitt 3.2.4 wurde bereits die Wichtigkeit von Glattheit angesprochen. Glattheit im Frequenz- beziehungsweise Spektralbereich hängt über den in Unterabschnitt 2.13.2 erwähnten Satz von Plancherel mit einer Lokalisierung des Filters im Ortsbereich zusammen. Örtliche Lokalisierung ist gewünscht und bedeutet, dass zur vollständigen Definition der Wirkung eines Filters nur wenige Parameter nötig sind. Beispielsweise sind bei klassischen Faltungsnetzwerken, bei denen die Filter eine hohe Lokalisierung im Ortsbereich vorweisen, im Fall eines 3×3 Faltungskerns lediglich 9 Parameter (multipliziert mit der Anzahl der *feature maps*) zur Filterung des Signals notwendig. Bei Bildern ist die Größe des Filters, in diesem Beispiel 3×3, unabhängig von der Größe n der Eingabe und genau diese Unabhängigkeit wünscht man sich auch für nichteuklidische Gebiete. Je lokalisierter ein Filter ist, desto stabiler ist er auch, was bedeutet, dass die sich aus der Filterung ergebende Funktion bei ähnlichen Argumenten auch ähnliche Funktionswerte annimmt. Ohne weitere Optimierung bezüglich der Glattheit im Spektralbereich liefert die Multiplikation $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T}$ in dieser Form jedoch keinen im Ortsbereich lokalisierten Filter. Um die Anzahl der benötigten Parameter zu senken und lokalisierte Filter zu erhalten, parametrisiert man die Einträge von $\hat{\mathbf{W}}$ (*spectral multiplier operator*) so, das sie glatt sind.

Auf euklidischen Gebieten ergibt sich Glattheit aus der Frequenz. Beispielsweise kann für einen zweidimensionalen Frequenzvektor $\boldsymbol{\omega}$ und dem zweidimensionalen Vektor $\mathbf{x}$, der die Koordinaten in einer Ebene bezeichnet, in der Ebene die Ähnlichkeit zweier Fourier-Atome $e^{i\boldsymbol{\omega}^\mathsf{T}\mathbf{x}}$ und $e^{i\boldsymbol{\omega}'^\mathsf{T}\mathbf{x}}$ mit dem Abstand $\|\boldsymbol{\omega} - \boldsymbol{\omega}'\|_2$ gemessen werden.

Auf nichteuklidischen Gebieten existiert das Konzept von Glattheit jedoch nicht, weshalb in Unterabschnitt 3.2.4 die Dirichlet-Energie zur Bestimmung der Glattheit verwendet wurde. Dies kam effektiv dem Verwerfen der Frequenzen über einem gewissen Grenzwert gleich. Die Einträge von $\hat{\mathbf{W}}$ waren dadurch aber noch nicht notwendigerweise glatt und die Filter somit nicht im Ortsbereich lokalisiert.

Im Fall eines planaren Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ könnte ein zu $\mathcal{G}$ dualer Graph $\mathcal{G}^*$ konstruiert werden. Dies ist ein Graph $\mathcal{G}^*$, der für jede von Kanten begrenzte Region im primalen Graphen $\mathcal{G}$, in der keine Knoten liegen, einen Knoten enthält. Für jedes Paar dieser als Facetten bezeichneten Regionen in $\mathcal{G}$, die durch eine Kante getrennt sind, sind in $\mathcal{G}^*$ die entsprechenden Knoten durch eine Kante miteinander verbunden. Die Gewichte w^*_{ij} der Kanten des zu $\mathcal{G}$ dualen Graphen spiegeln die Ähnlichkeit zweier Eigenfunktionen ϕ_i und ϕ_j wider. Das Konzept von Glattheit auf dem dualen Graph äußert sich dann in Lokalisierung auf dem primalen Graphen. Wie diese Gewichte w^*_{ij} genau gewählt werden

sollen um eine optimale Glattheit zu erreichen ist derzeit nicht wahrhaftig verstanden und ein aktueller Gegenstand der Forschung. In der Praxis hat sich aber die Wahl $w_{ij}^* = |\lambda_i - \lambda_j|$ bewährt, also der Betrag der Differenz der Eigenwerte λ_i und λ_j des *graph Laplacian*. Die Eigenwerte des *graph Laplacian* sind gerade dessen Spektrum.

Zur Parametrisierung der Filter nutzen die auf die in diesem Abschnitt erläuterte Weise optimierten Spektralmethoden als auch die an diese Idee anknüpfenden Spektrum-freien Methoden Transferfunktionen, welche von den Eigenwerten $\lambda_1, \ldots, \lambda_n$ des *graph Laplacian* abhängig sind. Für die gewünschte Lokalisierung im Ortsbereich wählt man Basisfunktionen $\beta_1, \ldots, \beta_q$, die gemäß dem eben beschriebenen Konzept von Glattheit im Spektrum glatt sind. Die gewählten Basisfunktionen dienen als Transferfunktionen mittels denen durch Linearkombination die Filter im Ortsbereich synthetisiert werden. Bei diesen Transferfunktionen, also den Basisfunktionen, handelt es sich üblicherweise um Polynome, beispielsweise um kubische Splines oder andere Polynome geringen Grades. Eine übliche Wahl für die Basisfunktionen sind auch die Tschebyschow-Polynome, was der Ansatz des von Defferrard, Bresson und Vandergheynst in 2016 vorgestellten *ChebNet* [DBV16] ist, einer Spektrum-freien Methode, die auf die Ideen in diesem Abschnitt anknüpft. Mit diesem Konzept von Glattheit und den darüber ausgewählten Basisfunktionen erhält man einen Vektor $\boldsymbol{\beta}(\lambda) = \big(\beta_1(\lambda), \ldots, \beta_q(\lambda)\big)$ bestehend aus q glatten, von den Eigenwerten des *graph Laplacian* abhängigen Funktionen. Die Anzahl q der Basisfunktionen ist als von der Größe n der Eingabe unabhängig gewählt, wobei n die Anzahl der Eigenwerte beziehungsweise Eigenfunktionen des *graph Laplacian* bezeichnet.

Mittels den glatten Basisfunktionen kann die Fourier-Transformierte $\hat{w}_k$ der Impulsantwort an Position k approximiert werden, wobei $k = 1, \ldots, n$. Dazu braucht lediglich der k-te Eigenwert in die Gleichung

$$\hat{w}_k = \sum_{i=1}^{q} \alpha_i \beta_i(\lambda_k)$$

eingesetzt zu werden. Die Koeffizienten $\alpha_1, \ldots, \alpha_q$ der q Basisfunktionen mit $q \ll n$ sind die Parameter einer individuellen Schicht, die von einem neuronalen Netzwerk während des Trainingsprozesses gelernt werden müssen. Durch

$$\underbrace{\begin{pmatrix} \hat{w}_1 \\ \vdots \\ \hat{w}_n \end{pmatrix}}_{\hat{\mathbf{w}}} = \underbrace{\begin{pmatrix} \beta_1(\lambda_1) & \cdots & \beta_q(\lambda_1) \\ \vdots & \ddots & \vdots \\ \beta_1(\lambda_n) & \cdots & \beta_q(\lambda_n) \end{pmatrix}}_{\mathbf{B}} \underbrace{\begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_q \end{pmatrix}}_{\boldsymbol{\alpha}}$$

ist der Vektor $\hat{\mathbf{w}}$ gegeben, dessen Komponenten die Einträge der Diagonalmatrix $\hat{\mathbf{W}}$ sind. Beachtenswert ist, dass je Schicht einer solchen Netzwerkarchitektur nur q Parameter α_i gelernt werden müssen. Die übrigen Parameter erhält man unter Verwendung der $n \times q$-Matrix $\mathbf{B}$ mittels Kernel-Interpolation. Anders als bei den Spektralmethoden in ihrer ursprünglichen Form ist mit dieser Approximation der Fourier-Transformierten die Anzahl der Parameter, hier q, also unabhängig von der Größe n der Eingabe, womit sich die Gefahr des *overfitting* reduziert. Mit der Kurzschreibweise $\hat{\mathbf{W}} = \operatorname{diag}(\hat{\mathbf{w}}) = \operatorname{diag}(\mathbf{B}\boldsymbol{\alpha})$

und Gleichung 3.6 kann für eine Spektralmethode demnach über

$$\mathbf{W} = \mathbf{\Phi}_k \hat{\mathbf{W}} \mathbf{\Phi}_k^\mathsf{T} = \mathbf{\Phi}_k \operatorname{diag}\left(\sum_{i=1}^{q} \alpha_i \beta_i(\lambda_1), \ldots, \sum_{i=1}^{q} \alpha_i \beta_i(\lambda_n) \right) \mathbf{\Phi}_k^\mathsf{T}$$

ein im Ortsbereich lokalisierter Filter erzielt werden, wobei $q \ll n$ sowie $k \ll n$ und $\beta_i(\lambda) = 0$ für die sortierten Eigenwerte mit $\lambda > \lambda_k$ gilt. Die Eigenwerte können als Frequenzen interpretiert werden, nach denen sortiert wird. Durch $k \ll n$ wird bewirkt, dass sich auf diejenigen k Eigenvektoren mit der geringsten Frequenz beziehungsweise die glattesten Funktionen im Sinne der Dirichlet-Energie beschränkt wird, wohingegen $q \ll n$ bezweckt, dass die Anzahl der Parameter unabhängig von der Größe der Eingabe ist und somit die Gefahr des *overfitting* reduziert wird.

Der Filter $\hat{\mathbf{W}}$ im Spektralbereich kann auf diese Weise im Ortsbereich synthetisiert werden und als $n \times n$-Matrix $\mathbf{W}$ geschrieben werden, wobei $\mathbf{W} = \mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T}$. Auch wenn diese Optimierung schlussendlich zu einer räumlichen Repräsentation $\mathbf{W}$ des Filters führt, so ist auf dem Weg dorthin dennoch weiterhin die rechenintensive Multiplikation $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T}$ nötig. Von den großen Problemen der Spektralmethoden konnten also zwar die fehlende Lokalisierung und die Abhängigkeit der Parameterzahl von der Größe der Eingabe gelöst werden, nicht aber die wenig zufriedenstellende quadratische Laufzeitkomplexität und die Abhängigkeit des gelernten Filters von der Basis $\mathbf{\Phi}$. Von diesen beiden noch offenen Problemen kann zumindest das Problem der hohen Laufzeitkomplexität von den Spekrumfreien Methoden durch die Fortführung der in diesem Abschnitt beschriebenen Idee behoben beziehungsweise umgangen werden. [Bro+17b; Bro19]

3.3. Spektrum-freie Methoden

In diesem Abschnitt wird die Spektrum-freie Definition der im Spektralbereich verallgemeinerten Faltung thematisiert. Auch wenn die Ideen hinter den Spektrum-freien Methoden wie in Abschnitt 1.1 beschrieben mindestens in das Jahr 2005 zurückreichen und ihr Ursprung chronologisch somit vor den Spektralmethoden einzuordnen ist, wurde die in diesem Abschnitt dargelegte Herangehensweise nach den Spektralmethoden vorgestellt.

Die Spektrum-freien Methoden sind als eine Fortführung des in Abschnitt 3.2 vorgestellten Ansatzes der Spektralmethoden und insbesondere deren in Unterabschnitt 3.2.6 vorgestellten Verbesserung zu verstehen. Nachdem durch diese erste Optimierung der spektralen Definition der verallgemeinerten Faltung bereits einige der in Unterabschnitt 3.2.5 angeführten Nachteile der spektralen Verallgemeinerung der Faltung gelöst werden konnten, beheben die in diesem Abschnitt vorgestellten Spektrum-freien Verfahren eines der verbleibenden fundamentalen Probleme der Spektralmethoden: ihre inakzeptable quadratische Laufzeitkomplexität.

3.3.1. Motivation

Im Fall euklidischer Gebiete kann man sich die Faltung vorstellen als die Operation $\mathbf{y} = \mathcal{W}\mathbf{x}$, wobei $\mathcal{W}\colon L^2(\mathbb{Z}^d) \to L^2(\mathbb{Z}^d)$ eine Toeplitz-Matrix ist – daher auch *Toeplitz operator*

genannt wird – und somit äquivariant unter Translation ist. Der Faltungsoperator $*$ ist derjenige lineare Operator, der die Translationsäquivarianz implementiert. Es gilt $\mathcal{W}f = f * w$. Nach dem Faltungstheorem wird der lineare Operator $*$ von der Fourier-Basis diagonalisiert, was bedeutet, dass die klassische Faltung sich alternativ im Frequenzbereich der Fourier-Transformation berechnen lässt. Es gilt also

$$\mathcal{W}f = f * w = \mathcal{F}^{-1}\left\{\hat{f} \cdot \hat{w}\right\} = \mathcal{F}^{-1}\left\{\mathcal{F}\{f\} \cdot \hat{w}\right\}, \tag{3.7}$$

wobei $\hat{w}$ die Fourier-Transformierte des Filters w beziehungsweise der Impulsantwort ist.

Das Prinzip der bereits vorgestellten Spektralmethoden ist deshalb, dass sie über die Zerlegung des *Laplacian* beziehungsweise *graph Laplacian* in dessen Eigenfunktionen ϕ_k die auf nichteuklidischen Gebieten nicht existente Fourier-Transformation auf nichteuklidische Gebiete verallgemeinern und dadurch ein Analogon zu Gleichung 3.7 konstruieren. Das Matrix-Vektor-Produkt $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T}\mathbf{f}$ definiert die verallgemeinerte „Fourier-Transformation“ $\mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \hat{w}\}$. Streng genommen darf man bei einem solchen Analogon zu Gleichung 3.7 nicht mehr von Faltung (*convolution*) sprechen, da die Operation $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T}\mathbf{f}$ nicht translationsäquivariant ist. Bei $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T}$ handelt es sich lediglich um einen linearen Operator, der zur Filterung eines als Vektor $\mathbf{f}$ dargestellten Signals f eingesetzt werden kann.

Das Problem im Fall nichteuklidischer Gebiete ist, dass mangels Translationsäquivarianz der Faltungsoperator $*$ nicht definiert ist und somit ein wichtiger Zwischenschritt in Gleichung 3.7 nicht möglich ist. In Ermangelung der Definition der Operation $f * w$ legen die Spektralmethoden deshalb $\mathcal{W}f = \mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T}\mathbf{f}$ fest, wobei aufgrund der nicht existenten Translationsäquivarianz auf nichteuklidischen Gebieten jedoch die Toeplitz-Struktur von $\mathcal{W}$ verloren geht und die Impulsantwort anders als bei der klassischen Faltung positionsabhängig ist.

Für die Spektralmethoden gilt am Beispiel eines Graphen mit Knotenmenge $\mathcal{V}$ und der darauf definierten Signale f sowie den n Eigenfunktionen $\phi_0, \ldots, \phi_{n-1}$ also

$$\begin{aligned}\mathcal{W}f &= \mathcal{F}^{-1}\left\{\mathcal{F}\{f\} \cdot \hat{w}\right\} \\ &= \sum_{k=0}^{n-1} \hat{f}_k \hat{w}_k \phi_k = \sum_{k=0}^{n-1} \langle f, \phi_k \rangle_{L^2(\mathcal{V})} \hat{w}_k \phi_k = \mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T}\mathbf{f}.\end{aligned} \tag{3.8}$$

Der Schritt $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T}\mathbf{f}$ hat wie in Unterabschnitt 3.2.5 erläutert eine inakzeptable Laufzeitkomplexität $\mathcal{O}(n^2)$, da für nichteuklidische Gebiete keine effizienten Algorithmen zur Beschleunigung der Fourier-Transformation wie etwa die schnelle Fourier-Transformation existieren.

Zusammenfassend hat man also mit den Spektralmethoden eine Verallgemeinerung der Fourier-Transformation eingeführt, da kein Weg bekannt war, der die Arbeit im Ortsbereich ermöglichte. Bei den Spektrum-freien Methoden sucht man nun einen Weg, diese „Fourier-Transformation“ aufgrund ihrer ineffizienten Performanz eben gerade nicht verwenden zu müssen. Dazu führt man mithilfe der bei den Spektralmethoden gewonnenen Erkenntnisse das in Gleichung 3.8 fehlende Glied $f * w$ für nichteuklidische Gebiete ein, wobei w die räumliche Repräsentation des im Spektrum des *Laplacian* beziehungsweise *graph Laplacian* definierten linearen Filters $\hat{w}$ ist. Dieser Weg führt wie in Abschnitt 1.1 angedeutet üblicherweise zu polynomiellen Filtern.

3.3.2. Grundlegendes Prinzip

Der Kernpunkt der Spektrum-freien Methoden ist den rechenaufwändigen Weg über die „Fourier-Transformation“ zu vermeiden. Zu diesem Zweck wird der *graph Laplacian* direkt auf eine zu filternde Funktion angewendet, statt ihn zunächst in seine Eigenfunktionen zu zerlegen, um auf diesem Weg an die Eigenbasis $\mathbf{\Phi}$ zu gelangen und mit dieser eine Verallgemeinerung der Fourier-Transformation zu definieren. Vielmehr werden verschiedene Potenzen $\mathbf{\Delta}^j$ des *graph Laplacian* $\mathbf{\Delta}$ auf die Funktion angewandt, wobei $j = 0, \dots, r-1$. Die r Potenzen des *graph Laplacian* bilden ein Polynom P von Grad $r-1$.

Durch die in Unterabschnitt 3.2.6 beschriebene Verwendung von Basisfunktionen, beispielsweise Polynome geringen Grades, die nach ihrer Glattheit ausgewählt werden, lässt sich ein Filter über seine spektrale Repräsentation $\hat{\mathbf{W}}$ im Ortsbereich synthetisieren, wodurch man eine Matrix $\mathbf{W}$ erhält. Die Spektralmethoden synthetisieren den Filter jedoch über den Rechenweg

$$\mathbf{\Phi}\,\mathrm{diag}\left(\sum_{j=0}^{r-1}\alpha_j\beta_j(\lambda_1),\dots,\sum_{j=0}^{r-1}\alpha_j\beta_j(\lambda_n)\right)\mathbf{\Phi}^\mathsf{T} = \mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T} = \mathbf{W}$$

im Ortsbereich, also unter Verwendung der ineffizienten verallgemeinerten Fourier-Transformation. Dahingegen synthetisieren die Spektrum-freien Methoden den Filter durch die direkte Anwendung des *graph Laplacian* $\mathbf{\Delta}$, beispielsweise unter der Verwendung eines Polynoms P, über

$$\mathbf{\Phi}\,\mathrm{diag}\left(\sum_{j=0}^{r-1}\alpha_j\beta_j(\lambda_1),\dots,\sum_{j=0}^{r-1}\alpha_j\beta_j(\lambda_n)\right)\mathbf{\Phi}^\mathsf{T} = P_{\boldsymbol{\alpha}}(\mathbf{\Delta}) = \mathbf{W}.$$

Durch die Erkenntnis $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T} = P_{\boldsymbol{\alpha}}(\mathbf{\Delta})$ können die Spektrum-freien Methoden auf die rechenintensive „Fourier-Transformation“ verzichten. Stattdessen brauchen lediglich die glatten Basisfunktionen direkt auf den *graph Laplacian* angewandt zu werden. Damit wird die aufwändige Berechnung der Eigenfunktionen des *graph Laplacian*, welche die Basis $\mathbf{\Phi}$ formen, sowie die ineffiziente Multiplikation mit $\mathbf{\Phi}^\mathsf{T}$ und $\mathbf{\Phi}$ umgangen werden. Auf diese Weise wird trotz der spektralen Definition des Filters vermieden, im Spektralbereich zu operieren.

Trotz ihrer inakzeptablen Laufzeitkomplexität $O(n^2)$ sind die Spektralmethoden also von großer konzeptueller Bedeutung für die Entwicklung des *geometric deep learning*. Insbesondere die von Bruna u. a. [Bru+14] vorgeschlagene und in Unterabschnitt 3.2.6 beschriebene Optimierung war wegweisend für die Entwicklung der Spektrum-freien Methoden.

3.3.3. Polynomielle spektrale Filter

Wie die Transferfunktionen $\beta_0, \dots, \beta_{r-1}$ definiert sein sollen, kann frei gewählt werden. Bewährt haben sich Polynome, insbesondere Tschebyschow-Polynome. Es bieten sich aber auch andere Polynome geringen Grades an, beispielsweise kubische Splines. In diesem Abschnitt wird das Funktionsprinzip polynomieller spektraler Filter anhand von Beispielen verdeutlicht.

Beispiel Monome

Ein Monom ist ein Polynom, welches aus nur einem Glied besteht. Im Fall des Polynomrings $R[X]$ sind sowohl 1 als auch X und X^4 Beispiele für Monome. Die Monome können eine Basis bilden. In diesem Abschnitt besteht die Monombasis aus den Basisfunktionen $\beta_j(\lambda) = \lambda^j$ mit $j = 0, \ldots, r-1$.

Da das Gebiet Ω kompakt und der *graph Laplacian* $\mathbf{\Delta}$ ein selbstadjungierter positivsemidefiniter Operator ist, kann $\mathbf{\Delta}$ wie in Unterabschnitt 2.12.1 erwähnt geschrieben werden als $\mathbf{\Phi\Lambda\Phi}^\mathsf{T}$, wobei $\mathbf{\Lambda} = (\lambda_1, \ldots, \lambda_n)^\mathsf{T}$ die Diagonalmatrix bestehend aus den Eigenwerten des *graph Laplacian* $\mathbf{\Delta}$ ist. Mit der Verwendung der Bezeichnung P für ein Polynom ist die räumliche Repräsentation $\mathbf{W}$ des Filters somit definiert durch

$$\begin{aligned}
\mathbf{W} = \mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T} &= \mathbf{\Phi}\,\mathrm{diag}\left(\sum_{j=0}^{r-1}\alpha_j\beta_j(\lambda_1), \ldots, \sum_{j=0}^{r-1}\alpha_j\beta_j(\lambda_n)\right)\mathbf{\Phi}^\mathsf{T} \\
&= \mathbf{\Phi}\underbrace{\begin{pmatrix}\sum_{j=0}^{r-1}\alpha_j\lambda_1^j & & \\ & \ddots & \\ & & \sum_{j=0}^{r-1}\alpha_j\lambda_n^j\end{pmatrix}}_{P_{\boldsymbol{\alpha}}(\mathbf{\Lambda})}\mathbf{\Phi}^\mathsf{T} = \sum_{j=0}^{r-1}\alpha_j\mathbf{\Phi}\begin{pmatrix}\lambda_1^j & & \\ & \ddots & \\ & & \lambda_n^j\end{pmatrix}\mathbf{\Phi}^\mathsf{T} \\
&= \sum_{j=0}^{r-1}\alpha_j\mathbf{\Phi}\mathbf{\Lambda}^j\mathbf{\Phi}^\mathsf{T} = \sum_{j=0}^{r-1}\alpha_j\Big(\underbrace{\mathbf{\Phi\Lambda\Phi}^\mathsf{T}}_{\mathbf{\Delta}}\Big)^j = \sum_{j=0}^{r-1}\alpha_j\mathbf{\Delta}^j = P_{\boldsymbol{\alpha}}(\mathbf{\Delta}).
\end{aligned}$$

Beispiel Tschebyschow-Polynome (*ChebNet*)

Besonders effizient ist es, den Filter $P_{\boldsymbol{\alpha}}(\mathbf{\Delta})$ als rekursiv berechenbares Polynom zu parametrisieren. Die übliche Wahl hierzu sind die Tschebyschow-Polynome. Das Tschebyschow-Polynom $T_j(x)$ der Ordnung j ist über die rekursive Bildungsvorschrift

$$T_j(x) = 2xT_{j-1}(x) - T_{j-2}(x)$$

berechenbar, wobei $T_0(x) = 1$ und $T_1(x) = x$.

Die Tschebyschow-Polynome bilden jedoch eine orthonormale Basis in $[-1, 1]$, weshalb der *graph Laplacian* skaliert werden muss, um dessen aufgrund seiner positiven Semidefinitheit nichtnegative Eigenwerte in $[0, \lambda_{\max}]$ auf das Intervall $[-1, 1]$ abzubilden. Zu diesem Zweck wird $\tilde{\mathbf{\Delta}} = 2\lambda_n^{-1}\mathbf{\Delta} - \mathbf{I}$ sowie $\tilde{\mathbf{\Lambda}} = 2\lambda_n^{-1}\mathbf{\Lambda} - \mathbf{I}$ definiert. Damit ist der Filter parametrisiert durch

$$P_{\boldsymbol{\alpha}}(\tilde{\mathbf{\Delta}}) = \sum_{j=0}^{r-1}\alpha_j\mathbf{\Phi}T_j(\tilde{\mathbf{\Lambda}})\mathbf{\Phi}^\mathsf{T} = \sum_{j=0}^{r-1}\alpha_jT_j(\tilde{\mathbf{\Delta}}). \tag{3.9}$$

Das Beispiel dieser Verfahrensweise beschreibt das von Defferrard, Bresson und Vandergheynst [DBV16] vorgestellte *ChebNet*. Diese Spektrum-freie Methode ist nicht auf die Berechnung der Eigenfunktionen angewiesen und hat im Vergleich zur quadratischen Laufzeitkomplexität der Spektralmethoden die Laufzeitkomplexität $O(rn)$. Sie wurde von Kipf

und Welling [KW16] weiter vereinfacht. Sowohl das *ChebNet*-Modell [DBV16] als auch dessen Vereinfachung von Kipf und Welling [KW16] können als Spezialfall des allgemeineren, zunächst wenig beachteten *graph neural network*-Rahmenwerks von Scarselli u. a. [Sca+09] aus dem Jahr 2009 verstanden werden.

Das Training solcher Spektrum-freien *graph convolutional networks* kann sich allerdings wie beispielsweise von Chiang u. a. [Chi+19] beschrieben insbesondere bei vielen Netzwerkschichten schwierig gestalten. Mit *Cluster-GCN* [Chi+19] aus dem Jahr 2019 existiert ein rechen- und speichereffizienter Algorithmus zum Training von *graph convolutional networks* wie etwa denen von Defferrard, Bresson und Vandergheynst [DBV16] und Kipf und Welling [KW16].

3.3.4. Vorteile der Spektrum-freien Verfahren gegenüber Spektralmethoden

Die Spektrum-freien Methoden umgehen oder beheben unter anderem die folgenden drei großen Nachteile des initialen Ansatzes der Spektralmethoden, also der direkten Interpretation der Intuition hinter der klassischen Faltung, bei welcher noch nicht wie in Unterabschnitt 3.2.6 beschrieben auf glatte *spectral multipliers* geachtet wird:

1. ineffiziente Matrix-Vektor-Multiplikation $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^{\mathsf{T}}\mathbf{f}$
2. von der Größe der Eingabe abhängige Parameteranzahl (erhöhter Rechenaufwand sowie Gefahr des *overfitting*)
3. fehlende Lokalisierung im Ortsbereich

Alle drei Probleme können durch gute Filterparametrisierung gelöst werden. Durch die in Unterabschnitt 3.2.6 erläuterte Optimierung können das zweite und dritte Problem dieser Aufzählung auch mit Spektralmethoden gelöst werden. Allerdings können Spektralmethoden nicht das erstgenannte Problem der wenig zufriedenstellenden quadratischen Laufzeitkomplexität $O(n^2)$ lösen, da sie selbst in der verbesserten Variation weiterhin auf die Verallgemeinerung der Fourier-Transformation setzen und das in dessen Laufzeitkomplexität nicht reduzierbare Produkt $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^{\mathsf{T}}$ berechnen, wobei $\hat{\mathbf{W}} = \operatorname{diag}(\hat{w}_1, \ldots, \hat{w}_n)$ die spektrale Repräsentation des Filters ist und $\mathbf{\Phi} = (\phi_1, \ldots, \phi_n)$ die Matrix mit den Eigenvektoren des Laplace-Beltrami-Operators $\mathbf{\Delta}$ als Spaltenvektoren bezeichnet.

Mit den Spektrum-freien Methoden können hingegen alle drei der genannten Probleme statt nur die letzten beiden behoben werden, da sie die aufwändige Berechnung von $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^{\mathsf{T}}$ umgehen, indem sie die gewählten, im Spektrum glatten Transferfunktionen auf den Laplace-Beltrami-Operator $\mathbf{\Delta}$ anwenden. Das Potenzieren des Laplace-Beltrami-Operators, welcher eine $n \times n$-Matrix ist, hat die Laufzeitkomplexität $O(n)$. Bei r Potenzen in einem Polynom vom Grad $r-1$ liegt der Gesamtrechenaufwand in $O(rn)$ statt in $O(n^2)$.

Die Entscheidung für einen Polynomfilter löst konstruktionsbedingt auch das zweite und dritte Problem, da die Anzahl der Parameter im Fall eines Polynoms nicht länger von der Größe n der Eingabe abhängt, sondern vom Grad $r-1$ des Polynomfilters, der niedrig gewählt wird. Zudem ist der Laplace-Beltrami-Operator ein lokaler Operator, da er nur die direkten Nachbarn (1-*hop neigbors*) eines Knotens einbezieht. Die Wirkung der j-ten Potenz des Laplace-Beltrami-Operators ist somit auf die j-*hop neigbors* begrenzt. Da

der Polynomfilter lediglich eine Linearkombination verschiedener Potenzen des Laplace-Beltrami-Operators ist, ist durch die Wahl eines geringen Grades $r-1$ automatisch der Filter im Ortsbereich lokalisiert.

Jedoch können selbst mit den Spektrum-freien Verfahren nicht *alle* Probleme der im Spektrum verallgemeinerten Faltung lösen. Die Abhängigkeit des Filters von der Basis des Gebiets bleibt bestehen, da die verallgemeinerte Faltung trotz des Wechsels zu der räumlichen Repräsentation des Filters im Spektrum des Laplace-Beltrami-Operators definiert ist. Der Grund dafür ist, dass die Eigenbasis des *Laplacian* beziehungsweise *graph Laplacian*, welche die Verallgemeinerung der Fourier-Basis auf nichteuklidische Gebiete ist, vom Gebiet abhängig ist. Deshalb eignen die Spektrum-freien Methoden sich genauso wenig wie die Spektralmethoden für Anwendungsfälle, in denen mit verformbaren Formen oder in anderer Weise über mehrere Gebiete hinweg gearbeitet werden soll, wie es in der Computergrafik und im Bereich des maschinellen Sehens (*computer vision*) häufig der Fall ist. Beispielsweise hat man es in der Computergrafik oft mit unterschiedlichen (diskretisierten) Mannigfaltigkeiten zu tun, die etwa verschiedene Posen einer dreidimensionalen Form enkodieren. Bei dem *shape correspondence* genannten (und in Abschnitt 4.8 noch näher vorgestellten) Problem ist dann eine Zuordnung zwischen den Punkten der einen Mannigfaltigkeit und einer anderen Mannigfaltigkeit (beziehungsweise zwischen zwei Polygonnetzen) gesucht. Soll ein auf einem Gebiet $\mathcal{X}$ gelernter Filter auch auf einem anderen Gebiet $\mathcal{Y}$ angewendet werden können, so eignen sich die in Abschnitt 3.4 vorgestellten kartografierenden Verfahren (*charting-based methods*), welche sich aus den anderen Anforderungen der Computergrafik und des maschinellen Sehens (*computer vision*) entwickelten, für diesen Anwendungszweck besser als die Spektralmethoden und auch die Spektrum-freien Methoden.

3.4. Kartografierende Methoden

In diesem Abschnitt wird eine Herangehensweise vorgestellt, die ortsabhängige Signale rein im Ortsbereich (*spatial domain*) der Fourier-Analysis verarbeitet, nachdem in Abschnitt 3.2 und Abschnitt 3.3 zwei Ansätze präsentiert wurden, die mit einer im Spektrum des *Laplacian* beziehungsweise *graph Laplacian* definierten faltungsähnlichen Operation arbeiten. Diese Verfahrensweise mit Ursprung in der Computergrafik und dem maschinellen Sehen (*computer vision*) kommt von allen in dieser Arbeit vorgestellten Ansätzen der vertrauten Vorstellung der klassischen Faltung am nächsten.

Zunächst wird der Begriff *kartografierend* (*charting-based*) geklärt und das grundlegende Prinzip dieser Art von Methoden dargelegt. Im Anschluss wird die Bedeutung der Zuordnung zwischen den Knoten einer Nachbarschaft und den verwendeten Gewichtsmatrizen erörtert. Abschließend werden drei frühe Vertreter dieser Vorgehensweise vorgestellt.

3.4.1. Begriffsklärung

Im Kontext von Mannigfaltigkeiten ist ein fundamentales Konzept das der Karte. Unter einer Karte versteht man einen Homöomorphismus auf einer offenen Teilmenge einer n-dimensionalen Mannigfaltigkeit $\mathcal{X}$ zu einer offenen Menge von $\mathbb{R}^n$. Im Rahmen dieser

Arbeit gilt $n = 2$. Zum besseren Verständnis kann man sich beispielsweise vorstellen, dass man Deutschland mit einer Karte auf eine Ebene $\mathbb{R}^2$ abbilden kann. Möchte man ein an Deutschland angrenzendes Land oder eine andere Region der Erde betrachten, so muss man zu einer anderen Karte wechseln. Durch einen vollständigen Satz von Karten können die Regionen der Erde vollständig beschrieben werden. Die Menge von Karten, deren Urbilder die Erde überdecken, nennt man Atlas. Analog nennt man die Menge von Karten, deren Urbilder eine Mannigfaltigkeit $\mathcal{X}$ überdecken, einen Atlas von $\mathcal{X}$.

Die kartografierenden Methoden nutzen dasselbe Prinzip. Auf nichteuklidischen Gebieten gibt es wie in Unterabschnitt 2.11.2 beschrieben keine globale Parametrisierung und damit auch kein gemeinsames globales System von Koordinaten. Eine mögliche Herangehensweise zur Lösung dieses Problems ist, für die Punkte eines nichteuklidischen Gebiets ein System von lokalen Koordinaten zu konstruieren, wozu *patch operators* definiert werden. Die grundlegende Funktionsweise dieses Ansatzes wird in Unterabschnitt 3.4.3 ausgeführt. Eine Karte (*chart*) wird im Englischen auch als *(coordinate) patch*, *coordinate map*, oder *local frame* bezeichnet. Ein Kartenwechsel des Atlas entspricht einem Koordinatenwechsel.

3.4.2. Motivation

Im Bereich des maschinellen Sehens (*computer vision*) führte die wachsende Verbreitung von bezahlbaren Tiefensensoren wie etwa Microsoft Kinect zu einem erhöhten Interesse an der Arbeit mit dreidimensionalen geometrischen Daten. Die geometrischen Daten wurden zu diesem Zweck traditionell in eine euklidische Repräsentation überführt – beispielsweise Voxelwolken, RGB-Bilder oder Tiefenkarten (*depth maps*) – um bekannte euklidische Techniken des maschinellen Lernens darauf anwenden zu können. Bei einer solchen Konvertierung geht jedoch nicht nur Information und Detailreichtum verloren. Auch ist eine euklidische Repräsentation eines geometrischen Objekts nicht intrinsisch, womit gemeint ist, dass für ihre Definition auf den umgebenden Raum zurückgegriffen wird, wodurch sie nicht invariant unter isometrischen, also abstandserhaltenden oder auch euklidischen Verformungen wie etwa einer Änderung der Pose ist. Intrinsisch nennt man eine Eigenschaft, wenn sie rein mittels der Metrik des topologischen Raums beschrieben werden kann. Die Umwandlung einer nichteuklidischen in eine euklidische Repräsentation hat also unerwünschte Konsequenzen.

Am Beispiel von Abbildung 3.2 auf der nächsten Seite und der in Abschnitt 2.8 auf Seite 18 vorgestellten Mannigfaltigkeiten lässt sich zeigen, wieso die Krümmung einer Mannigfaltigkeit eine intrinsische Ausführung der Faltung so wichtig macht. Denn durch eine intrinsische Definition ist die Faltung automatisch invariant unter abstandserhaltenden Verformungen. Aus dem Wunsch nach *intrinsischen* künstlichen neuronalen Netzwerken heraus entwickelten sich parallel zu den in Abschnitt 3.3 dargelegten Spektrum-freien Verfahren die kartografierenden Methoden.

3.4.3. Grundlegendes Prinzip

Bei der klassischen diskreten Faltung wird wie in Abbildung 2.4 auf Seite 16 veranschaulicht eine Stelle oder auch Bereich (*patch*) aus einem Gebiet extrahiert, auf dem eine Funktion

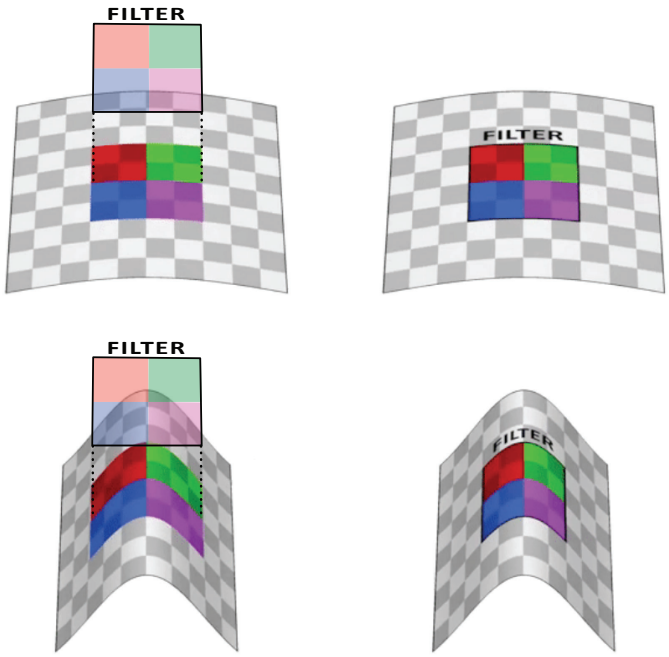


Abbildung 3.2.: **Gegenüberstellung der extrinsischen und intrinsischen Anwendung des Faltungsoperators.** Linke Spalte: eine Konvertierung von nichteuklidischen Daten in euklidische Daten – beispielsweise die Überführung eines Graphen in eine Matrix – zum Zwecke der Anwendung der klassischen Faltung auf die überführten Daten hat das Problem, dass die euklidische Faltung nicht invariant unter isometrischen Transformationen ist. Ein euklidischer 4×4 Faltungskern deckt aufgrund der extrinsischen Anwendung nach der Verformung des Gebiets hier fälschlicherweise einen 4×6 Bereich ab. Für einen Operator f und eine Verformung g gilt also nicht die Gleichung $f \circ g = f$ in Definition 2.6.1 auf Seite 13. Rechte Spalte: die intrinsische Anwendung eines geometrischen Faltungskerns ist invariant unter isometrischen Transformationen und arbeitet auch bei Verformungen wie vorgesehen. [Bro+17b]

definiert ist. Dies dient dem Zweck, die Summe der mit den Parametern des Faltungskerns gewichteten Pixelwerte in einem Bereich zu berechnen. Bei einer kartografierenden Methode wird ein Analogon zu solch einer Extraktion eines Bereichs gesucht, um dieses Prinzip auf nichteuklidische Gebiete zu übertragen. Hierzu definieren die kartografierenden Verfahren sogenannte *patch operators*, mittels derer ein Bereich (*coordinate patch*) um einen Punkt x der Mannigfaltigkeit $\mathcal{X}$ beziehungsweise Knoten des Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ extrahiert werden kann. Bei einem *patch operator* handelt es sich um ein gewichtetes Integral, das mittels einer Gewichtungsfunktion v spezifiziert wird und in diesem Abschnitt noch näher betrachtet werden wird. Dieses hiermit in seinen Grundzügen angerissene Funktionsprinzip wird in diesem Abschnitt genauer vorgestellt.

Streng genommen wird mit dem Begriff *Karte* oder im Englischen auch *(coordinate) patch* der Homöomorphismus ϕ bezeichnet, der eine Nachbarschaft $\mathcal{N}$ auf ihr Bild $\phi(\mathcal{N})$ abbildet. Häufig wird aber auch der Definitionsbereich oder die Zielmenge von ϕ – also auch die Nachbarschaft $\mathcal{N}$ und die Ebene, auf die $\mathcal{N}$ abgebildet wird – als Karte beziehungsweise *patch* bezeichnet.

Auf euklidischen Gebieten ist eine solche Abbildung ϕ beziehungsweise Extraktion eines *patch* möglich, da für euklidische Gebiete eine globale Parametrisierung existiert und es somit auch ein globales System von Koordinaten gibt. Bildern beispielsweise liegt ein reguläres Gitter zugrunde, und über die Pixelkoordinaten lassen sich die Pixel in eine Reihenfolge bringen. Für die Pixel existiert eine kanonische Reihenfolge. Somit kann, wie auf Seite 86 in Abbildung 4.3a illustriert, für die Pixel einer Nachbarschaft $\mathcal{N}$ auch eine Eins-zu-eins-Zuordnung zwischen einem Nachbarpixel und einer von $|\mathcal{N}|$ Gewichtsmatrizen umgesetzt werden. Hierauf wird in Unterabschnitt 3.4.4 noch detaillierter eingegangen, da es für ein gutfundiertes Verständnis der kartografierenden Methoden wichtig ist, diese Zuordnung zu verinnerlichen.

Weiterhin haben Pixel im Allgemeinen immer dieselbe Anzahl von Nachbarn. Beispielsweise hat ein Pixel im Fall eines 3×3-Faltungskerns, sich selbst nicht mitgezählt, ungeachtet seiner Position grundsätzlich immer acht Nachbarpixel, die unabhängig von der Position des Mittelpunkts im Pixelgitter jeweils auch immer gleich weit vom zentralen Pixel entfernt sind.

Bei nichteuklidischen Gebieten wie Mannigfaltigkeiten oder Graphen hingegen ist beides nicht der Fall. In einem Graphen oder einer (diskretisierten) Mannigfaltigkeit hat nicht jeder Knoten gleich viele Nachbarknoten. Auch haben nichteuklidische Gebiete kein globales System von Koordinaten, weshalb für die Knoten keine kanonische Reihenfolge existiert. Aus diesem Grund können im Fall von Daten nichteuklidischer Natur Bereiche nicht unmittelbar extrahiert werden zum Zwecke der Zuordnung des vom Filter abgedeckten Bereichs zu den Filterkoeffizienten. Auch kann keine Eins-zu-eins-Zuordnung zwischen den Pixeln einer Nachbarschaft $\mathcal{N}$ und den $|\mathcal{N}|$ Gewichtsmatrizen umgesetzt werden.

In Unterabschnitt 2.11.1 wurde erläutert, warum man sich auf nichteuklidischen Gebieten, anders als bei euklidischen Gebieten, die Faltung nicht vorstellen kann als eine Vorlage, die in der Manier eines gleitenden Fensters (*sliding window*) über das Signal geschoben wird, wobei die Korrelation zwischen Vorlage und abgedecktem Bereich festgehalten wird. Der Grund ist die nicht existente Translationsäquivarianz (*shift equivariance*), welche in Unterabschnitt 2.6.1 vorgestellt wurde. Um sich die der klassischen Faltung ähnliche Verallgemeinerung dennoch wie einen Schablonenabgleich (*template matching*)

vorstellen zu können, ist ein Weg erforderlich um im Ortsbereich einen Bereich (*patch*) der Mannigfaltigkeit beziehungsweise des Graphen zu extrahieren. Eine solche Extraktion ist zwar positionsabhängig, im Gegensatz zu der mangels Translationsäquivarianz auf nichteuklidischen Gebieten nicht wohldefinierten klassischen Faltung jedoch umsetzbar.

In Unterabschnitt 2.11.2 wurde zudem die Motivation dafür geschildert, den zu extrahierenden Bereich (*patch*) des Gebiets in einem lokalen intrinsischen Koordinatensystem zu repräsentieren. Das grundlegende Prinzip der kartografierenden Methoden ist, für jeden Punkt x des Gebiets Ω eine Abbildung (Karte) zwischen der Nachbarschaft $\mathcal{N}$ von x und einem System von lokalen intrinsischen Koordinaten zu schaffen. Anders ausgedrückt soll als Ersatz für die fehlende globale Parametrisierung eine lokale intrinsische Parametrisierung bestimmt werden. Dies kann erreicht werden durch Definition einer Menge von Gewichtungsfunktionen $v_1(x,\cdot),\ldots,v_J(x,\cdot)$, die die Nachbarschaftsbeziehung zweier Punkte x und x' eines nichteuklidischen Gebiets beschreiben und lokalisiert sind, also für Positionen nahe dem Punkt x ähnliche Funktionswerte liefern. Mittels einer solchen Gewichtungsfunktion v_j wird ein *patch operator* D_j spezifiziert, wobei $j = 1,\ldots,|\mathcal{N}|$.

Einen Bereich (*patch*) um einen Punkt x zu extrahieren kommt dann damit gleich, für $j = 1,\ldots,J$ das durch

$$D_j(x)f = \int_{\mathcal{X}} v_j(x,x')f(x')\mathrm{d}x', \quad j = 1,\ldots,J \tag{3.10}$$

definierte gewichtete Integral zu berechnen. Ein *patch operator* D_j wird somit spezifiziert, indem man eine Gewichtungsfunktion $v_j(x,\cdot)$ definiert, wobei $j = 1,\ldots,J$. Wie genau solche Gewichtungsfunktionen $v_1,\ldots,v_J$ definiert werden sollen, ist gerade die Frage, welche die kartografierenden Verfahren zu beantworten suchen. Die Definition dieser Gewichtungsfunktionen ist auch der Kernpunkt, in dem die verschiedenen konkreten kartografierenden Methoden sich voneinander unterscheiden. Die Aufgabe der in Unterabschnitt 3.4.5 und auch in den Kapiteln 4, 5 und 6 behandelten kartografierenden neuronalen Netzwerkmodelle besteht im Wesentlichen somit daraus, eine geeignete Wahl für die Gewichtungsfunktionen $v_1(x,\cdot),\ldots,v_J(x,\cdot)$ zu treffen. Der Unterschied zwischen kartografierenden Modellen ist also die Weise, auf welche sie eine zu der in Abbildung 4.3a auf Seite 86 gezeigten Eins-zu-eins-Zuordnung zwischen Filtern und Nachbarknoten analoge Abbildung realisieren.

Mittels eines auf diese Weise definierten *patch operator* D kann schließlich durch

$$(f * g)(x) = \sum_j g_j D_j(x)f \tag{3.11}$$

ein intrinsisches Analogon zu der klassischen Faltung im Ortsbereich definiert werden, wobei g_j den Filterkoeffizienten des Filters g bezeichnet, mit dem der über $D_j(x)f$ erhaltene Funktionswert eines Nachbarn von x gewichtet wird. Anschaulich entspricht dies der in Abbildung 2.4 auf Seite 16 dargestellten klassischen Faltung.

Der große Vorteil der kartografierenden Methoden gegenüber den Spektralmethoden als auch den Spektrum-freien Methoden ist, dass eine gemäß Gleichung 3.11 verallgemeinerte Faltung auch über mehrere Gebiete hinweg wie gewünscht arbeitet. Dies ist insbesondere in der Computergrafik und im Bereich des maschinellen Sehens (*computer*

vision) von Bedeutung, da dort ein Einsatzgebiet für die verallgemeinerte Faltung die in Abschnitt 4.8 thematisierte Ermittlung einer Entsprechung der Punkte verschiedener, durch Mannigfaltigkeiten modellierter Formen (*shape correspondence*) ist.

Im Gegensatz zu den Spektralmethoden ist die Anzahl der Parameter bei kartografierenden Methoden auch nicht abhängig von der Größe n der Eingabe, weshalb auch die Gefahr des als *overfitting* bezeichneten Problems kleiner ist. Die Anzahl der Parameter ist hier gleich der Anzahl J der Gewichtungsfunktionen, wobei J nicht von n abhängt, also in der Komplexitätsklasse $O(1)$ liegt. Auf die hier beschriebene Weise konstruierte Filter sind lokalisiert, was wie in Unterabschnitt 3.2.6 erläutert bedeutet, dass – wie bei klassischen Faltungsnetzwerken – nur relativ wenige Parameter zur Filterung eines Signals nötig sind.

Da Gleichung 3.11 und Gleichung 3.9 auf Seite 55 sich sehr ähneln, kann man sich unter diesem Aspekt betrachtet die in Abschnitt 3.3.3 präsentierte, *ChebNet* genannte Spektrum-freie Methode sowie deren Vereinfachung von Kipf und Welling [KW16] auch vorstellen im Sinne einer Anwendung von lokalen Gewichtungsfunktionen.

Der Nachteil dieses Ansatzes der lokalen Filterung, den die kartografierenden Methoden verfolgen, ist, dass sie zur Definition der Filter auf hartkodierte lokale Pseudo-Koordinaten setzen, die möglicherweise suboptimal bestimmt sind.

3.4.4. Zuordnung zwischen Nachbarknoten und Gewichtsmatrizen

Im vorigen Abschnitt wurde die in Abbildung 4.3 auf Seite 86 gezeigte Zuordnung zwischen den $|\mathcal{N}|$ Knoten einer Nachbarschaft $\mathcal{N}$ und den $|\mathcal{N}|$ Gewichtsmatrizen erwähnt. Die Realisierung dieser Zuordnung ist gerade die Weise, in welcher die verschiedenen kartografierenden Methoden sich voneinander unterscheiden, weshalb an dieser Stelle genauer darauf eingegangen wird.

In klassischen Faltungsnetzwerken werden für gewöhnlich alle Nachbarn in einer Nachbarschaft mit derselben Gewichtsmatrix assoziiert. Dies beruht auf einer als *weight sharing* bezeichneten Technik, die dazu dient, die Anzahl der für ein neuronales Netzwerk erforderlichen Parameter zu reduzieren und von den Entdeckungen der beiden Neurowissenschaftlern und Nobelpreisträgern Hubel und Wiesel [HW62] inspiriert ist, welche den visuellen Cortex untersuchten. Eigentlich würde jeder der $|\mathcal{N}|$ Pixel in einer Nachbarschaft $\mathcal{N}$ mit einer eigenen Gewichtsmatrix multipliziert werden. Zu einem beispielhaften Neuron mit einem rezeptiven Feld (*receptive field*) der Größe 3×3 gäbe es somit neun verschiedene Gewichtsmatrizen und jedem der neun Pixel in der Nachbarschaft wäre eine andere der neun Gewichtsmatrizen zugeordnet. Jedoch würde diese Verfahrensweise die Anzahl der Parameter einer Faltungsschicht drastisch erhöhen. Sie entspricht auch nicht der Funktionsweise des visuellen Cortex. Gering halten lässt sich die Parameteranzahl durch eine an die Untersuchungsergebnisse von Hubel und Wiesel angelehnte Annahme. Wenn es wichtig ist, dass an einem Pixel eine Kante erkannt wird, dann ist es aufgrund der bei Bildern vorhandenen Translationsäquivarianz auch an dessen benachbartem Pixel wichtig, die Kante zu erkennen. Ein und dieselbe Kante braucht jedoch nicht für jeden der neun Pixel in der 3×3-Nachbarschaft oder jede der 3×3-Nachbarschaften im Bild neu gelernt zu werden. Stattdessen kann ein Neuron für alle neun Pixel der Nachbarschaft $\mathcal{N}$ dieselbe Gewichtsmatrix verwenden. In klassischen Faltungsnetzwerken reicht also eine Gewichtsmatrix statt $|\mathcal{N}|$ Gewichtsmatrizen aus. Auch teilen die Neuronen derselben

Schicht sich diese Gewichtsmatrix. Es wird somit auch für jede Nachbarschaft die gleiche Gewichtsmatrix verwendet. Ohne diese *weight sharing* genannte Technik würde beispielsweise ein Bild mit einer Katze im rechten Bildbereich nicht als ‚Katze' klassifiziert werden, wenn der Trainingsdatensatz nur Bilder mit Katzen im linken Bildbereich enthielt.

Ohne *weight sharing*, wie es bei nichteuklidischen Gebieten der Fall ist, müssten aber $|\mathcal{N}|$ Gewichtsmatrizen verwendet werden und jedem der $|\mathcal{N}|$ Nachbarn eine der $|\mathcal{N}|$ Gewichtsmatrizen zugeordnet werden. Im Fall eines euklidischen Gebiets ist dies ohne Weiteres möglich, da hier aufgrund der globalen Parametrisierung beziehungsweise aufgrund des globalen Koordinatensystems eine kanonische Reihenfolge existiert und somit eine Eins-zu-eins-Zuordnung realisiert werden kann, die auch als *hard assignment* bezeichnet wird. Dies wird in Abbildung 4.3a auf Seite 86 veranschaulicht. Die Pixelreihenfolge ergibt sich für ein Bild, indem man die Pixel Zeile für Zeile von links nach rechts abwandert. Im Fall eines nichteuklidischen Gebiets existiert jedoch mangels globaler Parametrisierung beziehungsweise globalen Koordinatensystems keine kanonische Reihenfolge, weshalb kein *hard assignment* umgesetzt werden kann und jeder der $|\mathcal{N}|$ Nachbarn mit jeder der $|\mathcal{N}|$ Gewichtsmatrizen gewichtet werden muss. Eine solche Zuordnung nennt man *soft assignment* und ist in Abbildung 4.3b auf Seite 86 veranschaulicht.

3.4.5. Ausgewählte frühe Verfahren

In diesem Abschnitt werden Vorreiter der kartografierenden Methoden (*charting-based methods*) kurz vorgestellt.

Da die in den Kapiteln 4, 5 und 6 dieser Arbeit ausführlich erläuterten aktuellen Verfahren allesamt kartografierende Methoden sind, werden die in diesem Abschnitt thematisierten frühen neuronalen Netzwerkmodelle nicht im Detail behandelt. Aufgrund ihrer Relevanz sollen sie jedoch nicht unerwähnt bleiben.

3.4.5.1. *Geodesic CNN* oder auch *ShapeNet*

Das *Geodesic CNN* genannte Netzwerkmodell ist das erste neuronale Modell für *deep learning* auf Mannigfaltigkeiten beziehungsweise Polygonnetzen, welche diskretisierte Mannigfaltigkeiten sind. *Geodesic CNN* wurde im Jahr 2015 von Masci u. a. [Mas+15a] vorgestellt und trug zunächst den Namen *ShapeNet* [Mas+15b] bevor es umbenannt wurde.

Masci u. a. nutzen die Tatsache aus, dass zu jedem Punkt x einer differenzierbaren Mannigfaltigkeit $\mathcal{X}$ ein in Unterabschnitt 2.8.2 beschriebener Tangentialraum $T_x\mathcal{X}$ gehört, in dessen lokalem Koordinatensystem gearbeitet werden kann. Auf einer zweidimensionalen Mannigfaltigkeit mit zweidimensionalen Tangentialräumen $\mathbb{R}^2$ kann um einen Punkt x ein System von lokalen Polarkoordinaten oder auch Kreiskoordinaten konstruiert werden. Der Abstand eines Punktes x' vom Pol (hier der Punkt x), also dessen Radialkoordinate ρ, wird hierzu definiert durch einen intrinsischen Abstand $\rho\,(x') = d\,(x, x')$. Diese Distanz $d\,(x, x')$ entspricht dem geodätischen Weg, also der lokal kürzesten Verbindungskurve der zwei Punkte x und x'. Die Winkelkoordinate θ des lokalen Polarkoordinatensystems wird konstruiert, indem man von dem Punkt x in gleichem Winkel voneinander entfernte geodätische Linien in Richtung θ ausstrahlen lässt.

Weiterhin kann man sich eine bijektive Funktion h vorstellen, welche die Punkte in der Nachbarschaft eines Punkts $x \in \mathcal{X}$ in dieses System von lokalen Polarkoordinaten um x abbildet. Zur Extraktion eines Bereichs (*patch*) um x braucht dann nur deren Umkehrfunktion h^{-1} auf die lokalen geodätischen Polarkoordinaten (ρ, θ) um x angewendet zu werden. Durch $(D(x)f)(\rho,\theta) = \left(f \circ h^{-1}(x)\right)(\rho,\theta)$ kann eine Funktion f somit in den lokalen Polarkoordinaten interpoliert werden. Dazu werden die beiden Gewichtungsfunktionen $v_\rho(x,x')$ für die Radialkoordinate ρ beziehungsweise $v_\theta(x,x')$ für die Winkelkoordinate θ verwendet. Die beiden Gewichte v_ρ und v_θ sowie Beispiele verschiedener lokaler geodätischer Bereiche (*patches*) sind in Abbildung 3.3 veranschaulicht.

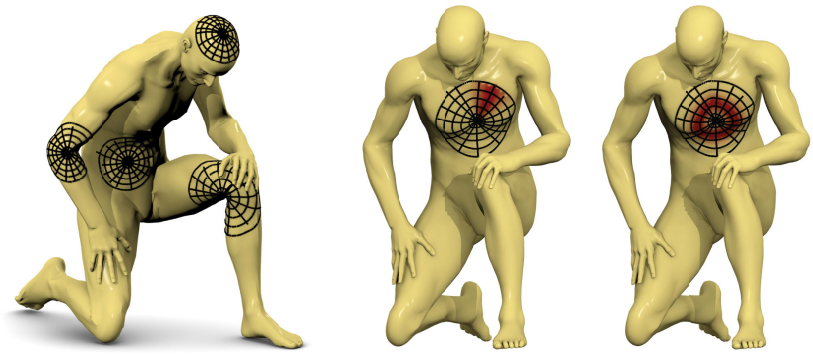
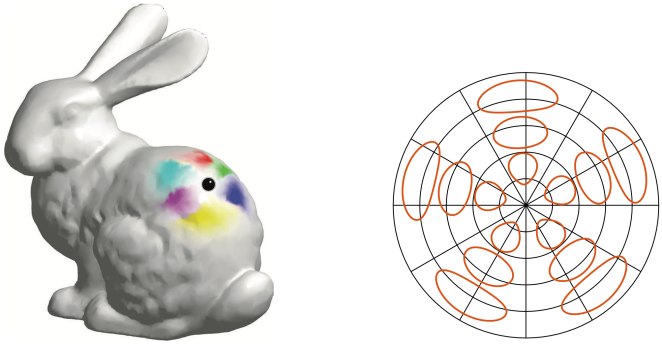


Abbildung 3.3.: **Illustration der Konstruktion eines lokalen geodätischen Polarkoordinatensystems auf einer Mannigfaltigkeit.** Links: Beispiele für lokale geodätische Bereiche (*patches*). Mitte: Beispiel für Gewichtung v_θ der Winkelkoordinaten (in Rot dargestellt die großen Gewichte). Rechts: Beispiel für Gewichtung v_ρ der Radialkoordinaten. [Mas+15a]

Die Kombination der beiden Gauß-Funktionen $v_\rho(x,x')$ und $v_\theta(x,x')$ (*Gaussians*), also Wahrscheinlichkeitsdichtefunktionen, ergibt die durch

$$v_{ij}(x,x') = \mathrm{e}^{-(\rho(x')-\rho_i)^2/2\sigma_\rho^2}\,\mathrm{e}^{-\left(\theta(x')-\theta_j\right)^2/2\sigma_\theta^2}$$

gegebenen Produkte von Gauß-Funktionen. Diese Produkte $v_{ij}(x,x')$ dienen in Gleichung 3.10 auf Seite 61 als Gewichtungsfunktionen im gewichteten Integral, durch welches der *patch operator* D definiert ist. Mittels D werden die Bereiche (*patches*) der Mannigfaltigkeit extrahiert. Dabei bezeichnet $i = 1,\ldots,J$ die Indizes der in Abbildung 3.4b auf der nächsten Seite dargestellten „Behälter“ (*bins*), in die die Radialkoordinaten eingeteilt werden und $j = 1,\ldots,J'$ die Indizes der „Behälter“, in die die Winkelkoordinaten eingeteilt werden. Die resultierenden JJ' Gewichtungsfunktionen sind „Behälter“, die in Polarkoordinaten gemessen die Breite $\sigma_p \times \sigma_\theta$ haben. Über den mittels dieser Gewichtungsfunktionen v_{ij} spezifizierten *patch operator* D wird schließlich die der klassischen Faltung ähnliche, *geodesic convolution* genannte Operation im Ortsbereich definiert. [Mas+15a; Bro+17b]



(a) Illustration der JJ' Gewichtungsfunktionen

(b) Illustration der JJ' „Behälter“ (*bins*)

Abbildung 3.4.: **Veranschaulichung der Gewichtungsfunktionen v_{ij} des *Geodesic CNN*.** Abbildung 3.4a zeigt die zur Konstruktion des *patch operator* D am schwarz markierten Punkt x verwendeten Gewichtungsfunktionen $v_{ij}(x, x')$. Abbildung 3.4b zeigt die Repräsentation der JJ' Gewichtungsfunktionen im System der lokalen Polarkoordinaten. [Bro+17b]

3.4.5.2. *Anisotropic CNN*

Boscaini u. a. [Bos+16b] knüpfen mit dem im Jahr 2016 veröffentlichten *Anisotropic CNN* an das *Geodesic CNN* von Masci u. a. [Mas+15a] an. *Geodesic CNN* faltet die extrahierten Bereiche mit in Polarkoordinaten dargestellten Filtern. Die Orientierung der Filter ist dabei jedoch nicht unbedingt eindeutig, was *Geodesic CNN* dadurch löst, dass es sämtliche Orientierungen eines Filters anwendet, jedoch nur die maximale Filterantwort behält (*angular max-pooling*). Das *Anisotropic CNN* erweitert das *Geodesic CNN*, indem es zur Extraktion der Bereiche auf anisotrope Diffusion [PM90] setzt, wobei die extrahierten Bereiche durch Rotation an der jeweiligen Hauptkrümmungsrichtung mit dem Maximalwert aller Krümmungen ausgerichtet werden. Anisotropie ist die Eigenschaft, die in verschiedenen Richtungen unterschiedliche Änderungen erlaubt, wohingegen bei der Isotropie die Änderungen in allen Richtungen gleich verlaufen.

Wie bei dem *Geodesic CNN* werden auch bei dem *Anisotropic CNN* Gewichtungsfunktionen $v_1, \ldots, v_J$ definiert, über die die *patch operators* $D_1, \ldots, D_J$ aus Gleichung 3.10 auf Seite 61 konstruiert werden. Die zentrale Idee des *Anisotropic CNN* ist, hierbei anisotrope Wärmeleitungskerne (*heat kernels*) als die lokalen Gewichtungsfunktionen $v_1, \ldots, v_J$ zu interpretieren. Für das Verständnis dieser Methode ist ein Verständnis der Wärmeleitungsgleichung (*heat equation*) hilfreich, welche daher im Folgenden zunächst kurz vorgestellt wird, bevor das eigentliche Verfahren präsentiert wird. Erwähnenswert ist, dass der *patch operator* – obwohl die Bereiche (*patches*) im Ortsbereich definiert sind – im Spektralbereich konstruiert werden kann, was beim *Anisotropic CNN* der Fall ist. Er kann, wie es etwa beim *Geodesic CNN* der Fall ist, aber auch im Ortsbereich konstruiert werden.

Wärmeleitungsgleichung Mit der erstmals von Joseph Fourier im Jahr 1822 gelösten Wärmeleitungsgleichung (*heat equation*) lässt sich der Transport von thermischer Energie modellieren und somit die Wärmediffusion beschreiben. Bei der Wärmeleitungsgleichung handelt sich um eine partielle Differentialgleichung, deren Fundamentallösung als Wär-

meleitungskern $h_t(x, \cdot)$ (*heat kernel*) bezeichnet wird. Im einfachsten Fall – in homogenen Medien mit isotropem, also in allen Richtungen gleich schnellem Wärmefluss – ist die Wärmeleitungsgleichung durch

$$\begin{cases} f_t(x,t) = -c\Delta f(x,t) \\ f(x,0) = f_0(x) \quad \text{(Anfangsbedingung)} \end{cases} \tag{3.12}$$

definiert. In dieser Gleichung repräsentiert $f(x,t)$ die Temperatur an einem Punkt x zu einem Zeitpunkt t. Der konstante Proportionalitätsfaktor c (*thermal diffusivity constant*) wird als Temperaturleitfähigkeit oder Temperaturleitzahl des Mediums bezeichnet.

Die Lösung partieller Differentialgleichungen wie der Wärmeleitungsgleichung war die hauptsächliche Motivation für Joseph Fourier um die in Unterabschnitt 2.13.1 vorgestellte Fourier-Analysis zu entwickeln. Die Fundamentallösung der Wärmeleitungsgleichung – also der Wärmeleitungskern $h_t(x, \cdot)$ – lässt sich im Spektralbereich (*spectral domain*) ausdrücken oder, in anderen Worten, mittels des Spektrums des *Laplacian* beziehungsweise *graph Laplacian*. Das Spektrum des Laplace-Beltrami-Operators besteht aus dessen Eigenwerten beziehungsweise Spektralwerten. Die den Eigenwerten zugeordneten Eigenfunktionen des *Laplacian* beziehungsweise *graph Laplacian* sind wie in Unterabschnitt 2.13.1 beschrieben eine Verallgemeinerung der Fourier-Basis auf nichteuklidische Gebiete.

Durch Projektion auf die Eigenbasis des Laplace-Beltrami-Operators Δ kann eine quadratintegrierbare Funktion wie hier $f_0 \in L^2(\mathcal{X})$ wie in Gleichung 2.14 auf Seite 39 gezeigt in eine „Fourierreihe“ zerlegt werden. Das dadurch auftretende innere Produkt kann gemäß der Definition in Gleichung 2.5 auf Seite 25 durch ein Integral ersetzt werden. Die Fundamentallösung von Gleichung 3.12 erhält man, indem man den *heat operator* $H^t = \mathrm{e}^{-t\Delta}$ auf die Anfangsbedingung anwendet. Die Temperaturleitzahl c soll hier aus Gründen der Einfachheit den Wert 1 haben. Auf diese Weise erhält man die folgende Gleichung:

$$\begin{aligned} f(x,t) &= \mathrm{e}^{-t\Delta} f_0(x) = \mathrm{e}^{-t\Delta} \sum_{i\geq 0} \langle f_0, \phi_i \rangle_{L^2(\mathcal{X})}\, \phi_i(x) = \sum_{i\geq 0} \langle f_0, \phi_i \rangle_{L^2(\mathcal{X})}\, \mathrm{e}^{-t\Delta} \phi_i(x) \\ &= \sum_{i\geq 0} \langle f_0, \phi_i \rangle_{L^2(\mathcal{X})} \underbrace{\left(\sum_{n\geq 0} \frac{(-t\Delta)^n}{n!} \right)}_{\exp(-t\Delta)} \phi_i(x) = \sum_{i\geq 0} \langle f_0, \phi_i \rangle_{L^2(\mathcal{X})} \left(\sum_{n\geq 0} \frac{(-t)^n \lambda_i^n \phi_i(x)}{n!} \right) \\ &= \sum_{i\geq 0} \langle f_0, \phi_i \rangle_{L^2(\mathcal{X})} \left(\sum_{n\geq 0} \frac{(-t\lambda_i)^n}{n!} \right) \phi_i(x) = \sum_{i\geq 0} \langle f_0, \phi_i \rangle_{L^2(\mathcal{X})}\, \mathrm{e}^{-t\lambda_i} \phi_i(x) \\ &= \sum_{i\geq 0} \left(\int_{\mathcal{X}} f_0(x')\, \phi_i(x')\, \mathrm{d}x' \right) \mathrm{e}^{-t\lambda_i} \phi_i(x) = \int_{\mathcal{X}} f_0(x') \underbrace{\sum_{i\geq 0} \mathrm{e}^{-t\lambda_i} \phi_i(x) \phi_i(x')}_{h_t(x,x')}\, \mathrm{d}x' \end{aligned} \tag{3.13}$$

Zunächst wird die quadratintegrierbare Funktion gemäß Gleichung 2.14 in eine „Fourierreihe“ zerlegt. Nachdem der *heat operator* $\mathrm{e}^{-t\Delta}$ in die Summe hineingezogen wurde, kann die Exponentialfunktion gemäß ihrer Definition als Potenzreihe umgeschrieben werden.

Aufgrund der Eigenwertgleichung beziehungsweise $(\Delta)^n\phi = (\lambda)^n\phi$ kann $\Delta\phi_i$ ersetzt werden durch $\lambda_i\phi_i$ beziehungsweise $\Delta^n\phi_i$ durch $\lambda_i^n\phi_i$. Daraufhin wird die Exponentialfunktion von ihrer Potenzreihendarstellung in ihre ursprüngliche Repräsentation umgeschrieben. Die Fundamentallösung der Wärmeleitungsgleichung ist der Wärmeleitungskern $h_t(x, x')$, der sich aus Gleichung 3.13 ablesen lässt.

Der Wärmeleitungskern $h_t(x, \cdot)$ der Wärmeleitungsgleichung produziert auf nichteuklidischen Gebieten tropfenförmige Gewichte um den Punkt x, die wie bei dem *Geodesic CNN* als lokale Gewichtungsfunktionen $v_1, \ldots, v_J$ interpretiert werden können. Der Nachteil an der bisher beschriebenen Verfahrensweise ist, dass die dadurch erhaltenen Wärmeleitungskerne isotrop sind, was bedeutet, dass die Wärme sich gleich schnell in alle Richtungen ausbreitet. Die Ausbreitung kann dabei über den Parameter t kontrolliert werden.

Anisotrope Wärmeleitung In Gleichung 3.12 wurde der einfachste Fall von Wärmefluss beschrieben. Wie im vorigen Abschnitt ausgeführt ist deren Fundamentallösung, also der gesuchte Wärmeleitungskern $h_t(x, \cdot)$, isotrop. Dies bedeutet, dass mit Gleichung 3.12 nur solcher Wärmefluss modelliert werden kann, der in alle Richtungen gleich schnell abläuft. Eine allgemeinere anisotrope Wärmediffusion (*anisotropic heat diffusion*) auf einer Mannigfaltigkeit kann durch

$$f_t(x, t) = -\operatorname{div}\big(\mathbf{A}(x)\nabla f(x, t)\big) \tag{3.14}$$

formalisiert werden. Dabei ist $\mathbf{A}(x)$ eine Matrix, mit der die Wärmeleitfähigkeit (*thermal conductivity*) bezeichnet wird. Die Wärmeleitfähigkeit ist im allgemeinen anisotropen Fall ein Tensor zweiter Stufe und wird somit eben durch eine Matrix beschrieben. Der Tensor der Wärmeleitfähigkeit erlaubt wie von Andreux u. a. [And+14] ausgeführt die positions- und richtungsabhängige Modellierung des Wärmeflusses.

Gleichung 3.14 ähnelt der Definition des *Laplacian* in Unterabschnitt 2.12.2 auf Seite 30. Der anisotrope *Laplacian* ist durch

$$\Delta_{\alpha\theta} f(x) = -\operatorname{div}\big(\mathbf{A}_{\alpha\theta}(x)\nabla f(x)\big) \tag{3.15}$$

definiert. Die gesuchte Fundamentallösung der partiellen Differentialgleichung in Gleichung 3.14, also den Wärmeleitungskern (*heat kernel*) dieser anisotropen Diffusionsgleichung, erhält man über die durch

$$h_{\alpha\theta t}(x, x') = \sum_{i\geq 0} \mathrm{e}^{-t\lambda_{\alpha\theta i}} \phi_{\alpha\theta i}(x)\phi_{\alpha\theta i}(x') \tag{3.16}$$

gegebene spektrale Konstruktion, wobei mit $\phi_{\alpha\theta 0}(x), \phi_{\alpha\theta 1}(x), \ldots$ die Eigenfunktionen des in Gleichung 3.15 definierten anisotropen *Laplacian* bezeichnet sind und $\lambda_{\alpha\theta 0}(x), \lambda_{\alpha\theta 1}(x), \ldots$ die zugehörigen Eigenwerte sind.

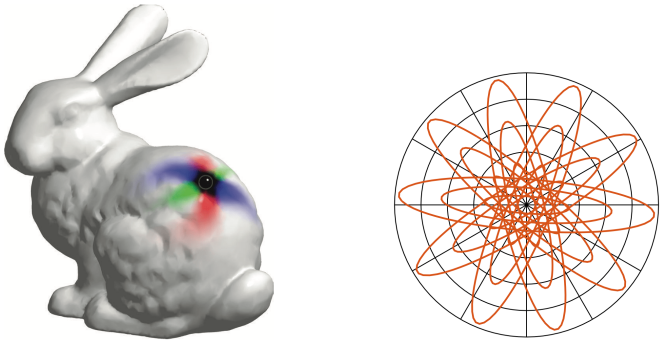
Der Tensor $\mathbf{A}(x)$ beziehungsweise $\mathbf{A}_{\alpha\theta}(x)$ der Wärmeleitfähigkeit (*thermal conductivity tensor*) ist im Fall von zweidimensionalen Mannigfaltigkeiten eine 2×2-Matrix. Boscaini u. a. [Bos+16a] schlagen für $\mathbf{A}_{\alpha\theta}(x)$ folgende, durch

$$\mathbf{A}_{\alpha\theta}(x) = \mathbf{R}_\theta(x) \begin{pmatrix} \alpha & \\ & 1 \end{pmatrix} \mathbf{R}_\theta^\mathsf{T}(x)$$

gegebene 2×2-Matrix vor, wobei die 2×2-Rotationsmatrix $\mathbf{R}_\theta(x)$ eine Drehung, beispielsweise um die Hauptkrümmungsrichtung, um den Winkel θ durchführt. Der Parameter $\alpha > 0$ steuert den Grad der Anisotropie, wobei $\alpha = 1$ dem isotropen Fall entspricht.

Die anisotropen Wärmeleitungskerne $h_{\alpha\theta t}(x, \cdot)$ (*heat kernels*) sehen wie langgestreckte rotierte Tropfen aus und sind in Abbildung 3.5 illustriert. Die Längung wird mittels dem Parameter α gesteuert, die Orientierung mittels θ und die Größenordnung mittels t.

Indem man die Wärmeleitungskerne $h_{\alpha\theta t}(x, \cdot)$ als die bei der Spezifikation der *patch operators* in Gleichung 3.10 auf Seite 61 gebrauchten Gewichtungsfunktionen $v_1, \ldots, v_J$ interpretiert, kann man die Bereiche (*patches*) einer Mannigfaltigkeit ähnlich dem in Unterunterabschnitt 3.4.5.1 vorgestellten *Geodesic CNN* zum Zwecke der verallgemeinerten Faltung extrahieren. Zum Vergleich, der bei dem *Anisotropic CNN* eingesetzte Parameter θ entspricht grob der Winkelkoordinate θ des lokalen Polarkoordinatensystems des *Geodesic CNN* und der Parameter t der Radialkoordinate ρ. [Bos+16b; Bro+17b]



(a) Illustration der Gewichtungsfunktionen

(b) Illustration der Gewichtungsfunktionen in Polarkoordinaten)

Abbildung 3.5.: **Veranschaulichung der Wärmeleitungskerne $h_{\alpha\theta t}$ des *Anisotropic CNN*.** Abbildung 3.5a zeigt die zur Konstruktion des *patch operator* D am schwarz markierten Punkt x verwendeten Wärmeleitungskerne $h_{\alpha\theta t}$ beziehungsweise Gewichtungsfunktionen. Abbildung 3.5b zeigt die Repräsentation der Wärmeleitungskerne im System der lokalen Polarkoordinaten. [Bro+17b]

3.4.5.3. *Mixture model network (MoNet)*

Das von Monti u. a. im Jahr 2017 vorgestellte *Mixture model network (MoNet)* [Mon+17] ist eine Verallgemeinerung der in den beiden vorangegangenen Abschnitten vorgestellten Verfahren *Geodesic CNN* beziehungsweise *Anisotropic CNN* und somit auch von klassischen euklidischen Faltungsnetzwerken. *Mixture model network (MoNet)* ist von allen in Abschnitt 3.4 präsentierten Verfahren die am allgemeinsten formulierte Möglichkeit zur Extraktion von Bereichen (*patches*).

Monti u. a. parametrisieren mit *Mixture model network (MoNet)* die Bereiche (*patches*), indem sie um jeden Knoten x eines Graphen ein lokales System von d-dimensionalen

Pseudo-Koordinaten $\mathbf{u}(x, x')$ definieren. Auf diese Pseudo-Koordinaten wenden Monti u. a. eine Menge von parametrischen Gaußkernen $v_1(\mathbf{u}), \ldots, v_J(\mathbf{u})$ an, welche durch

$$v_j(\mathbf{u}) = \exp\left(-\frac{1}{2}(\mathbf{u} - \boldsymbol{\mu}_j)^\mathsf{T}\Sigma_j^{-1}(\mathbf{u} - \boldsymbol{\mu}_j)\right) \tag{3.17}$$

definiert sind, um durch Kerndichteschätzung die in Gleichung 3.10 auf Seite 61 gezeigten Gewichtungsfunktionen v für den *patch operator* D zu erhalten. Die Parameter dieser in Gleichung 3.17 definierten Gaußkerne werden dabei gelernt. Bei den zu lernenden Parametern handelt es sich um die $d \times 1$-Erwartungswertvektoren $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_J$ sowie die $d \times d$-Kovarianzmatrizen $\Sigma_1, \ldots, \Sigma_J$.

Die Formen der Filter lernt *Mixture model network (MoNet)* ebenfalls, und zwar indem es die Erwartungswerte und Varianzen der Gauß-Funktionen (*Gaussians*), welche die Filtergewichte mit den lokalen Pseudo-Koordinaten assoziieren, lernt. Mit dieser Vorgehensweise kann die in Gleichung 3.11 auf Seite 61 definierte Verallgemeinerung der Faltung als Gaußsches Mischmodell (*Gaussian mixture model*) interpretiert werden, was den Namen von *Mixture model network (MoNet)* erklärt.

Mixture model network (MoNet) lernt somit nicht nur die Filter, sondern auch die in Gleichung 3.11 gezeigten *patch operators*. [Mon+17; Bro+17b]

3.5. Ortsbereich und Spektralbereich kombinierende Methoden

In diesem Abschnitt werden Weiterentwicklungen der Fourier-Transformation wie die Kurzzeit-Fourier-Transformation und die Wavelet-Transformation sowie deren Verallgemeinerungen auf nichteuklidische Gebiete vorgestellt. Im Anschluss wird eine neuronale Netzwerkarchitektur vorgestellt, welche die Kurzzeit-Fourier-Transformation zur Konstruktion eines in Abschnitt 3.4 präsentierten *patch operator* nutzt.

Aufgrund der in Unterabschnitt 2.13.4 beschriebenen Unschärferelation ist es unmöglich, eine messbare Größe gleichzeitig beliebig genau im Ortsbereich als auch im Frequenzbereich der Fourier-Transformation zu bestimmen. Man kann jedoch versuchen einen besseren Kompromiss zwischen Auflösung im Ortsbereich und Auflösung im Frequenzbereich zu finden als es bei der Fourier-Transformation der Fall ist, was es erlaubt, ein Signal in beiden Bereichen gleichzeitig (bis zu einem gewissen Grad) zu lokalisieren.

3.5.1. Kurzzeit-Fourier-Transformation

Bei der Kurzzeit-Fourier-Transformation (*short-time Fourier transform*), im Englischen auch als *windowed Fourier transform* bezeichnet, wird anders als bei der Fourier-Transformation nicht das Signal als Ganzes transformiert. Stattdessen wird das Signal mittels einer Fensterfunktion in kürzere Abschnitte gleicher Länge unterteilt, die individuell transformiert werden. Durch die gesonderte Überführung der Abschnitte in den Frequenzbereich ergibt sich für jeden Abschnitt dessen jeweiliges einzelnes Spektrum.

Dazu wird im Fall einer kontinuierlichen Funktion das zu transformierende Signal f mit einer Fensterfunktion g multipliziert, deren Werte $g(\tau)$ lediglich für einen bestimmten

Abschnitt von Null verschieden sind. Gebräuchliche Fensterfunktionen sind etwa die Rechteckfunktion, das Von-Hann-Fenster oder das Gauß-Fenster. Während das Fenster g im Fall einer zeitabhängigen Funktion entlang der Zeitachse geschoben wird, wird das dabei entstehende Signal mittels der Fourier-Transformation transformiert. Dabei bezeichnet τ den für die Verschiebung nötigen Translationsparameter.

3.5.1.1. Definition der klassischen Kurzzeit-Fourier-Transformation

Die klassische Kurzzeit-Fourier-Transformation **STFT** ist in der Signalverarbeitung für eine integrierbare zeitabhängige Funktion $f\colon \mathbb{R} \to \mathbb{C}$ definiert durch

$$\mathbf{STFT}\{f(t)\}(\tau,\omega) = \int_{-\infty}^{\infty} f(t)\, \underbrace{g(t-\tau)\mathrm{e}^{-\mathrm{i}\omega t}}_{\overline{g_{\tau,\omega}}(t)}\, \mathrm{d}t = \left\langle f, g_{\tau,\omega}\right\rangle_{L^2(\mathbb{R})}. \tag{3.18}$$

Da die Fensterfunktion g außerhalb des Fensters den Wert 0 liefert, verschwindet das Produkt $f(\cdot)g(\cdot)$ dort. Die in Unterabschnitt 2.13.2 auf Seite 34 vorgestellte Fourier-Transformation $\mathcal{F}$ zum Vergleich ist für den Definitionsbereich $\mathbb{R}$ definiert durch

$$\mathcal{F}\{f(t)\}(\omega) = \int_{-\infty}^{\infty} f(t)\, \mathrm{e}^{-\mathrm{i}\omega t}\, \mathrm{d}t.$$

Im Unterschied zur Fourier-Transformation $\mathcal{F}$ hängt die Kurzzeit-Fourier-Transformation **STFT** neben der Frequenz ω (Kreisfrequenz) noch von der zeitlichen Lokalisierung τ des Fensters ab.

Je nach Wahl der Fensterfunktion g wird der Kompromiss zwischen Lokalisierung im Zeit- beziehungsweise Ortsbereich und Lokalisierung im Frequenzbereich unterschiedlich bestimmt. Ein breites Fenster führt zu einer höheren Auflösung im Frequenzbereich, aber einer schlechteren Auflösung im anderen Bereich, wohingegen es bei einem schmalen Fenster umgekehrt ist.

Bei guter Auflösung im Frequenzbereich können nahe beisammenliegende Frequenzanteile getrennt werden. Bei guter Auflösung im Zeit- beziehungsweise Ortsbereich erkennt man den genauen Zeitpunkt beziehungsweise Ort, an dem eine Frequenz sich ändert. Dass aus einer hohen Auflösung im einen Bereich eine unscharfe Auflösung im jeweils anderen Bereich folgt, ergibt sich aus der Küpfmüllerschen Unbestimmtheitsrelation, nach der das Produkt aus Zeit und Frequenz ein konstanter Wert ist.

In den einzelnen Abschnitten liegt bei hoher Auflösung im Frequenzbereich zwar eine hohe Unschärfe im Zeitbereich vor, allerdings erlaubt die Unterteilung des Signals über das Fenster und den Translationsparameter τ insgesamt eine bessere zeitliche beziehungsweise räumliche Lokalisierung als es bei der Fourier-Transformation der Fall ist.

3.5.1.2. Verallgemeinerung auf nichteuklidische Gebiete

Um die Kurzzeit-Fourier-Transformation auf nichteuklidische Gebiete zu verallgemeinern, definieren Shuman, Ricaud und Vandergheynst [SRV16] neben der spektralen Faltung der in Abschnitt 3.2 vorgestellten Spektralmethoden einen verallgemeinerten Translationsoperator als auch einen verallgemeinerten Modulationsoperator. Für die Modulation

muss lediglich mit einer Eigenfunktion des *Laplacian* beziehungsweise *graph Laplacian* multipliziert werden. Die Definition eines Translationsoperators ist aufgrund der nicht vorhandenen Translationsinvarianz schwieriger, kann aber realisiert werden durch Faltung mit einer Delta-Funktion δ unter Verwendung der spektralen Definition der Faltung in Gleichung 3.4 auf Seite 46. Die Translation ist – hier im Fall einer Mannigfaltigkeit $\mathcal{X}$ – somit definiert durch

$$\begin{aligned}(g * \delta_{x'})(x) &= \sum_{i \geq 0} \langle g, \phi_i \rangle_{L^2(\mathcal{X})} \langle \delta_{x'}, \phi_i \rangle_{L^2(\mathcal{X})} \phi_i(x) \\ &= \sum_{i \geq 0} \hat{g}_i \phi_i(x') \phi_i(x).\end{aligned}$$

Die verschobenen und modulierten Atome können dann in Form von

$$g_{x',j}(x) = \phi_j(x') \sum_{i \geq 0} \hat{g}_i \phi_i(x) \phi_i(x')$$

dargestellt werden. Im nichteuklidischen Fall wird das Fenster somit im Spektralbereich über die Fourier-Koeffizienten $\hat{g}_i$ spezifiziert. Mit ϕ_i beziehungsweise ϕ_j sind Eigenfunktionen des *Laplacian* bezeichnet. Einsetzen dieser verschobenen und modulierten Atome in ein Analogon zum in Gleichung 3.18 gezeigten Skalarprodukt liefert die auf nichteuklidische Gebiete verallgemeinerte Kurzzeit-Fourier-Transformation, die durch

$$\mathbf{STFT}\{f(x)\}(x', j) = \left\langle f, g_{x',j} \right\rangle_{L^2(\mathcal{X})} = \sum_{i \geq 0} \hat{g}_i \phi_i(x') \left\langle f, \phi_i \phi_j \right\rangle_{L^2(\mathcal{X})}$$

definiert ist. Diese Definition ist intrinsisch, da alle an ihrer Konstruktion beteiligten Konstrukte intrinsisch sind.

3.5.1.3. Nachteile der klassischen Kurzzeit-Fourier-Transformation

Die Kurzzeit-Fourier-Transformation geht den fundamentalen Nachteil der Fourier-Transformation an, nämlich die bei perfekter Auflösung im Frequenzbereich fehlende Lokalisierung im Zeit- beziehungsweise Ortsbereich. Allerdings ist auch die Kurzzeit-Fourier-Transformation nicht frei von Schwächen.

Eines ihrer großen Probleme ist, dass bei der Kurzzeit-Fourier-Transformation die Auflösung festgesetzt ist, da die Breite des Fensters unveränderlich festgelegt ist. Dabei beeinflusst die Breite des Fensters, ob die Auflösung gut im Zeit- beziehungsweise Ortsbereich wird oder ob sie im Frequenzbereich gut wird. Mit einer veränderlichen Breite könnte der Kompromiss noch weiter verbessert werden. Über die Multiskalenanalyse (*multiresolution analysis*) kann dieser Schwachpunkt der Kurzzeit-Fourier-Transformation behoben werden, was zu der in Unterabschnitt 3.5.2 vorgestellten Wavelet-Transformation führt.

3.5.2. Wavelet-Transformation

Die Wavelet-Transformation kann man sich vorstellen als die Weiterentwicklung der in Unterabschnitt 3.5.1 präsentierten Kurzzeit-Fourier-Transformation. Sie setzt an dem

in Unterunterabschnitt 3.5.1.3 angeführten Nachteil der Kurzzeit-Fourier-Transformation an um den Kompromiss zwischen hoher Auflösung im Zeit- beziehungsweise Ortsbereich und hoher Auflösung im Frequenzbereich weiter zu verbessern, welcher der in Unterabschnitt 2.13.4 beschriebenen Unschärferelation geschuldet ist.

Ein Problem der Kurzzeit-Fourier-Transformation ist, dass nur ein Fenster, welches eine feststehende Breite hat, über das Signal geschoben wird. Aufgrund der festgesetzten Fensterbreite ist jedoch die Auflösung im Zeit- beziehungsweise Ortsbereich als auch die Auflösung im Frequenzbereich über das gesamte Signal hinweg unverändert. Allerdings erfordern niedrige Frequenzen eine hohe Auflösung im Frequenzbereich, wohingegen hohe Frequenzen eine hohe Auflösung im Zeit- beziehungsweise Ortsbereich erfordern. Deshalb kann die Kurzzeit-Fourier-Transformation nicht zur gleichen Zeit sowohl mit den niedrigen Frequenzen als auch mit den hohen Frequenzen im Signal gut umgehen.

Die Wavelet-Transformation ermöglicht unter Einsatz eines Skalierungsfaktors Fenster unterschiedlicher Breite, wodurch den genannten Anforderungen besser entsprochen werden kann (*multiresolution analysis*). Dank eines breiten Fensters kann bei niedrigen Frequenzen eine hohe Auflösung im Frequenzbereich ermöglicht werden und bei hohen Frequenzen, dank eines schmalen Fensters, eine hohe Auflösung im Zeit-/Ortsbereich.

3.5.2.1. Definition der Wavelet-Transformation

Wie schon bei der Kurzzeit-Fourier-Transformation wird das zu transformierende Signal mit einer Fensterfunktion multipliziert. Anstatt jedoch das Fenster zu verschieben und zu modulieren wie bei der Kurzzeit-Fourier-Transformation, wird das Fenster verschoben und *skaliert*. Dazu wird bei der Wavelet-Transformation das Konzept von Frequenz ersetzt durch ein Konzept von Größenordnung (*scale*).

Analog zur Kurzzeit-Fourier-Transformation ist τ der Translationsparameter. Im Unterschied zur Kurzzeit-Fourier-Transformation wird jedoch anstatt der Frequenz ω ein Skalierungsparameter $s := \frac{1}{\omega}$ verwendet. Definiert ist die kontinuierliche Wavelet-Transformation für ein zeitabhängiges Signal f durch

$$\mathcal{W}_\psi(\tau, s) = \frac{1}{\sqrt{|s|}} \int_{-\infty}^{\infty} f(t)\psi^* \left(\frac{t-\tau}{s}\right) \mathrm{d}t,$$

wobei f das zu transformierende Signal bezeichnet und ψ^* ein komplex konjugiertes Wavelet ψ, welches als Fensterfunktion dient. Wavelets werden hier als Basisfunktionen verwendet, die bei der Wavelet-Transformation alternativ zu den bei den diversen Varianten der Fourier-Transformation gebrauchten Fourier-Basisfunktionen $\mathrm{e}^{-\mathrm{i}\omega t}$ eingesetzt werden.

Unter einem Wavelet versteht man eine wellenförmige Funktion, die über den Skalierungsfaktor s gestaucht, also verschmälert, oder aber auch expandiert, also verbreitert werden kann. Über die beiden Parameter τ und s werden die jeweiligen verschobenen und skalierten Wavelets aus einem sogenannten *mother wavelet* $\psi(t)$ erzeugt. Aus einem *mother wavelet* $\psi(t)$ lassen sich verschiedene Wavelet-Familien $\psi_{s\tau} = \frac{1}{s}\psi\left(\frac{t-\tau}{s}\right)$ ableiten. Die einfachste der Wavelet-Familien ist die Familie der Haar-Wavelets. Ein Haar-Wavelet kann aus der Kombination zweier Rechteckfunktionen gebildet werden.

3.5.2.2. Multiskalenanalyse auf Graphen und Mannigfaltigkeiten

Die Wavelet-Transformation ist im Allgemeinen der Fourier-Transformation überlegen, nicht nur in Bezug auf den besseren, der Unschärferelation geschuldeten Kompromiss zwischen Auflösung im Frequenzbereich und Auflösung im Zeit- beziehungsweise Ortsbereich. Für die auf Graphen verallgemeinerte Wavelet-Transformation existiert mit [HVG11] zudem ein schneller Algorithmus zur Berechnung der verallgemeinerten Wavelet-Transformation und ihrer inversen Transformation. Dieser Algorithmus setzt auf die Tschebyschow-Polynome und hat die Laufzeitkomplexität $O(m|\mathcal{E}|)$, wobei m der höchste Grad der verwendeten Tschebyschow-Polynome und $|\mathcal{E}|$ die Anzahl der Kanten im Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ist. Damit ist die verallgemeinerte Wavelet-Transformation wesentlich effizienter als die verallgemeinerte Fourier-Transformation, die eine ineffiziente quadratische Laufzeitkomplexität hat und für die kein Analogon zur schnellen Fourier-Transformation existiert.

In [Xu+19] wird ein auf der verallgemeinerten Wavelet-Transformation basierendes neuronales Netzwerkmodell vorgestellt, das in der *node classification* genannten Aufgabe sowohl die in Abschnitt 3.2 beschriebenen, auf der verallgemeinerten Fourier-Transformation basierenden Spektralmethoden schlägt als auch bessere Ergebnisse erzielt als die in Abschnitt 3.3 präsentierten Spektrum-freien Verfahren, etwa *ChebNet*, oder das in Unterunterabschnitt 3.4.5.3 vorgestellte *Mixture model network (MoNet)*, welches eine kartografierende Methode ist.

Mit [WD21] existiert ein aktuelles, auf der verallgemeinerten Wavelet-Transformation basierendes Verfahren, das sich in der Aufgabe *node classification* mit den Modellen auf dem derzeitigen Stand der Technik misst.

Für einen Einstieg in die Wavelet-Theorie auf nichteuklidischen Gebieten verweisen Bronstein u. a. [Bro+17b] auf Coifman und Maggioni [CM06] und, daran anknüpfend, auf die Veröffentlichungen [Szl+05; GNC10]. In [RG13] stellen die Autoren ein neuronales Netzwerkmodell vor, das für Graphen die Konstruktion einer Basis bestehend aus Wavelets lernt, mittels der eine gewünschte Klasse von Signalen auf Graphen dargestellt werden kann. In [CCM15] wird ein neuronales Netzwerkmodell präsentiert, das unter anderem für Graphen Repräsentation der Graphen erzeugt, die invariant unter Translation und Rotation sowie auch stabil unter Deformation sind (*wavelet scattering*).

3.5.3. *Localized Spectral CNN (LSCNN)*

In [Bos+15] aus dem Jahr 2015 stellen Boscaini u. a. ein neuronales Netzwerkmodell vor, welches die in Unterabschnitt 3.5.1 vorgestellte Kurzzeit-Fourier-Transformation als *patch operator* für die in Abschnitt 3.4 beschriebenen kartografierenden Methoden (*charting-based methods*) gebraucht.

Mittels der Kurzzeit-Fourier-Transformation kann eine Funktion f um einen Punkt x herum in Form eines aus Gleichung 3.10 auf Seite 61 bekannten *patch operator* $D_j(x)f = \mathbf{STFT}\{f\}(x, j)$ im Spektralbereich dargestellt werden um auf Mannigfaltigkeiten und Punktwolken eine verallgemeinerte Faltung wie in Gleichung 3.11 intrinsisch zu definieren.

Ein auf diese Weise extrahierter *patch* kann ähnlich zu der von Abschnitt 3.2 oder Abschnitt 3.3 bekannten Matrix $\hat{\mathbf{W}}$ in der Multiplikation $\mathbf{\Phi}\hat{\mathbf{W}}\mathbf{\Phi}^\mathsf{T}$ als *spectral multiplier operator* interpretiert werden, wobei ein solcher den *patch* repräsentierenden *spectral multiplier ope-*

rator hier mit den lernbaren Filterkoeffizienten g_j multipliziert wird. Im Unterschied zu den Spektralmethoden und Spektrum-freien Methoden arbeitet das in [Bos+15] präsentierte Modell dem Anschein nach auch über mehrere Gebiete hinweg wie gewünscht.

Zusätzlich kann auch die Definition der in der Kurzzeit-Fourier-Transformation verwendeten Fensterfunktion gelernt werden. [Bro+17b]

4. Vervollständigung verformbarer Formen

In diesem Kapitel wird das im Englischen als *shape completion* bezeichnete Problem behandelt, bei dem anhand einer unvollständigen Beobachtung die Oberfläche einer dreidimensionalen Form (*shape*) rekonstruiert werden soll. Im Speziellen wird der ungleich schwierigere Fall der Oberflächengenerierung eines unvollständigen *verformbaren* Motivs thematisiert. Synonym dazu wird auch von einer deformierbaren Form gesprochen.

Zunächst wird das Problem *deformable shape completion* beschrieben und als konkreter Anwendungszweck für *geometric deep learning* in den Zusammenhang dieser Arbeit eingeordnet. Eine vielversprechende Herangehensweise an die Problematik ist der Einsatz von generativen Methoden wie etwa *variational autoencoders*. Diesen Ansatz verfolgen Litany u. a. [Lit+18], deren Arbeit in diesem Kapitel untersucht wird.

Nach einer vorbereitenden Abhandlung von *autoencoders* und der bayesschen Statistik werden *variational autoencoders* vorgestellt, bevor im Anschluss eine die klassische Faltung verallgemeinernde *graph convolutional operation* präsentiert wird, welche es dem *variational autoencoder* erlaubt, direkt auf als Graphen interpretierten Polygonnetzen zu arbeiten. Nachdem auf das der *shape completion* zugrundeliegende Problem *shape correspondence* eingegangen wurde, wird das Kapitel mit der Schilderung der Vorgehensweise zur Ermittlung einer plausiblen Vervollständigung zu einer gegebenen partiellen Form abgeschlossen.

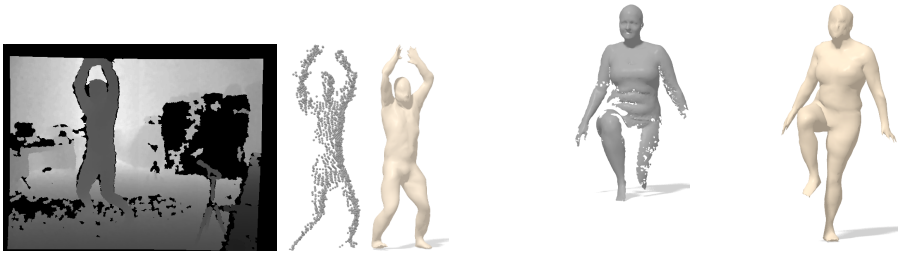
4.1. Problembeschreibung

Dank der zunehmenden Verfügbarkeit von bezahlbaren portablen Tiefensensoren wie etwa Microsoft Kinect, Intel RealSense oder aktuelle, mit der Apple-Technologie TrueDepth ausgerüstete iPhone-Modelle ist es heutzutage so einfach wie nie zuvor, dreidimensionale Scans von Objekten, Menschen oder kompletten Schauplätzen anzufertigen. Ein Problem, das beim Erfassen eines Motivs beziehungsweise der Umgebung mittels eines Tiefensensors auftreten kann, sind fehlende Bereiche im erzeugten Scan, die von kleinen Lücken bis hin zu ganzen Szenen reichen können und sich beispielsweise auf die Verdeckung des betroffenen Motivs durch ein anderes zurückführen lassen können.

Die Rekonstruktion des gescannten Motivs durch die Komplettierung dessen Oberfläche ist für ein breites Spektrum von Anwendungsgebieten von Interesse, unter anderem für die Gebiete des autonomen Fahrens, der virtuellen Realität (*virtual reality* als auch *augmented reality*) oder der Robotik. Aufgrund des bahnbrechenden Erfolgs der in rasantem Tempo voranschreitenden Entwicklung der Methoden des *deep learning* und der dank der Verbreitung von kostengünstigen Tiefensensoren vorhandenen Trainingsdatensätze hat das Problem *shape completion* einen Aufschwung der Erforschung von Methoden erlebt, mittels derer sich aus einem oder mehreren unvollständigen Scans das erfasste

dreidimensionale Objekt rekonstruieren lässt. Die Problemstellung wird in Abbildung 4.1 veranschaulicht. Die zu lösende Aufgabe ist das dreidimensionale Analogon zu der in der Bildbearbeitung gebräuchlichen Methode des *inpainting*, bei der versucht wird, zerstörte oder verlorene Teile eines Bildes wiederherzustellen.

Eine der Schwierigkeiten bei dem Problem *shape completion* ist dessen mehrdeutige Lösbarkeit, was unter Umständen eine Vielzahl verschiedener zulässiger Rekonstruktionen ermöglicht.



(a) Vervollständigung einer Punktwolke (b) Vervollständigung eines Polygonnetzes

Abbildung 4.1.: **Beispiele für Vervollständigungen unvollständiger Formen.** Abbildung 4.1a zeigt die Komplettierung einer Punktwolke, die aus einer mittels Microsoft Kinect angefertigen Tiefenkarte (*depth map*) extrahiert wurde. Abbildung 4.1b zeigt die Komplettierung eines unvollständigen Polygonnetzes aus dem Datensatz *DFAUST* [Bog+17]. [Lit+18]

Deformable shape completion

Von der uneindeutigen Lösbarkeit abgesehen ergeben sich zusätzliche Schwierigkeiten bei der Oberflächenvervollständigung, wenn es sich um verformbare statt um starre Objekte handelt. Beispiele für deformierbare Formen sind Menschen, welche offensichtlich verschiedene Posen einnehmen können, oder Gesichter, welche unterschiedliche Gesichtsausdrücke annehmen können. Solche auch als nichteuklidische Deformationen (*non-rigid deformations*) bezeichneten Transformationen erhalten nicht unbedingt den Abstand, weshalb es sich nach jeder Verformung effektiv um ein neues geometrisches Objekt handelt. Anders als bei euklidischen, also abstandserhaltenden Transformationen (*rigid transformations*), beispielsweise dem Verschieben oder dem Drehen des gesamten Objekts, kann man bei nicht den Abstand erhaltenden Verformungen nicht von Selbstähnlichkeit unter der Verformung ausgehen. Auf genau diese Annahme setzen jedoch Methoden, die auf die klassische euklidische Faltung bauen. Um das Problem *deformable shape completion* dennoch mittels klassischen euklidischen Faltungsnetzwerken angehen zu können, würde das Netzwerkmodell aufgrund der Vielzahl neuer Formen deutlich mehr Parameter erfordern und der nötige Rechenaufwand würde sich dadurch drastisch erhöhen.

Allgemein wären, um auch größere fehlende Bereiche vervollständigen zu können, zur Lösung von *deformable shape completion* komplexere A-priori-Verteilungen – welche im Verlauf dieses Kapitels noch thematisiert werden – über die geometrischen Merkmale der Formen notwendig als es bei *shape completion* und euklidischen Transformationen der

Fall ist. Der Grund dafür ist, dass bei deformierbaren Eingabeformen auch Verformungen zulässig wären, die während des Trainingsprozesses nicht gesehen wurden. Umgekehrt kann ein Trainingsdatensatz realistisch betrachtet nicht alle denkbaren plausiblen Verformungen einer Form enthalten. Dieser Ansatz mittels komplexeren A-priori-Verteilungen wird beispielsweise von Anguelov u. a. [Ang+05] verfolgt, die ihre Arbeit im Jahr 2005 – also vor dem Wiederaufleben von *deep learning* im Jahr 2012 – veröffentlichten.

Einordnung

Wenngleich für starre Formen das Problem *shape completion* in Arbeiten wie [Wei+15; Son+16; Qi+17a] bereits mit klassischen euklidischen Verfahren gelöst werden konnte und auch die Oberflächenvervollständigung verformbarer Formen in Arbeiten wie der von Anguelov u. a. [Ang+05] bereits mittels euklidischer Methoden angegangen wurde, lässt sich erst mit den Techniken des *geometric deep learning* auch das Problem *deformable shape completion* praktikabel lösen.

Der Grund weshalb für *deformable shape completion* die Methoden des *geometric deep learning* so viel besser geeignet sind ist, dass die auf Mannigfaltigkeiten beziehungsweise Graphen verallgemeinerte Faltung – im Gegensatz zu euklidischen Techniken – *intrinsisch* definiert ist und somit automatisch invariant unter euklidischen, also isometrischen oder auch den Abstand erhaltenden Deformationen ist. Dies bedeutet, dass die Funktionsweise des verallgemeinerten Faltungsoperators sich durch die Deformation nicht ändert, weshalb ein geometrisches Netzwerkmodell signifikant weniger Parameter als ein euklidisches Verfahren erfordert, woraus auch ein geringerer Rechenaufwand folgt.

4.2. Grundlegendes Prinzip

Das in diesem Kapitel betrachtete Verfahren von Litany u. a. [Lit+18] ist in der Lage, direkt auf der dreidimensionalen Struktur der Formen – hier Polygonnetze – zu arbeiten. Dadurch kann das Netzwerkmodell die Repräsentationen der Formen unmittelbar von den Daten, die es während des Trainingsprozesses beobachtet, lernen und muss sich nicht auf *handcrafted features* verlassen. Es ist also nicht wie zu den Zeiten vor dem Wiederaufkommen des *deep learning* erforderlich, dass ein menschlicher Fachexperte mit tief reichendem und sehr spezifischem Domänenwissen sich mühsam zu überlegen braucht wie die Merkmale der Daten extrahiert werden könnten. Stattdessen brauchen dem neuronalen Netzwerkmodell während dessen Trainingsprozess lediglich genügend Beispieldaten gezeigt zu werden, sodass dieses die Muster in den Daten auch ohne die Mitwirkung eines solchen Fachexperten zu erkennen lernt. Ermöglicht wird die direkte Verarbeitung der Polygonnetze durch die Definition einer *graph convolutional operation* mittels der die *graph convolutional layers* der Netzwerkarchitektur – also die Netzwerkschichten, in denen diese der klassischen Faltung ähnliche Operation ausgeführt wird – realisiert werden.

Das Arbeitsprinzip des von Litany u. a. [Lit+18] präsentierten Modells ist in zwei Stufen erklärt. Während des Trainingsprozesses lernt ein *variational autoencoder* die Generierung vollständiger Formen. *Variational autoencoder* werden in Abschnitt 4.6 näher vorgestellt. Im Wesentlichen handelt es sich dabei um ein Paar aus Encoder und Decoder, wobei der

Encoder die Eingabeformen in eine andere Repräsentation überführt und der Decoder wiederum bei Eingabe einer solchen internen Repräsentation daraus eine vollständige Form generiert. Während der als Inferenz bezeichneten Phase, in der das trainierte Netzwerkmodell ausgewertet wird – in anderen Worten, bei der eigentlichen Vervollständigung einer partiellen Eingabe – spielt der Encoder keine Rolle mehr. Dieser dient nur dem Lernen einer effizienteren Repräsentation der Eingabedaten. Nachdem diese erlernt ist und der Decoder gelernt hat daraus Ausgaben zu generieren, wird der Encoder nicht länger benötigt. Mittels des Decoders wird während der Inferenz im Zuge eines Optimierungsverfahrens aus dem erlernten *latent space* (‚verborgener Raum') der Codierungen vollständiger Formen aus allen Codierungen diejenige Codierung ausgewählt, für welche die daraus erzeugbare vollständige Form am besten zu der unvollständigen Eingabe passt.

4.3. Wissenschaftlicher Beitrag

Die in Abschnitt 4.2 beschriebene Entkopplung der eigentlichen Komplettierung der Formen vom Trainingsprozess des generativen Modells ist der hauptsächliche wissenschaftliche Beitrag von Litany u. a. [Lit+18].

Frühere Netzwerkmodelle lernen die *deformable shape completion* von Ende zu Ende und sind dadurch voreingenommen in Richtung der während des Trainings gesehenen Art von fehlenden Daten. Dies soll bedeuten, dass die Vervollständigung an sich, neben dem Erlernen der internen alternativen Repräsentation der Daten, ebenfalls im Rahmen des Trainingsprozesses anhand der beobachteten Beispieldaten gelernt wird, statt wie von Litany u. a. als eigenständiges Optimierungsproblem behandelt zu werden. Ende zu Ende trainierte Netzwerkmodelle können somit nur zur Komplettierung solcher Bereiche eingesetzt werden, für die der Trainingsdatensatz genügend Beispiele enthielt, denen genau dieser Bereich fehlte. Neuronale Netzwerke vor dem *graph convolutional variational autoencoder* von Litany u. a. [Lit+18] können also beispielsweise nur dazu eingesetzt werden, um menschliche Arme zu vervollständigen, nicht aber zur Komplettierung menschlicher Unterkörper, wenn während des Trainingsprozesses keine fehlenden Unterkörper gesehen wurden, da diese Modelle möglicherweise nicht gut auf Arten fehlender Daten verallgemeinern, die nicht Teil des Trainingsdatensatzes waren.

Bei dem Modell von Litany u. a. hingegen kann die Ergänzung partieller Formen mittels ausschließlich vollständiger Formen erlernt werden, infolgedessen es auch nicht zu einer solchen Voreingenommenheit kommt. Während des Trainingsprozesses müssen keine unvollständigen Formen gesehen werden. Da bei Litany u. a. [Lit+18] die eigentliche Ergänzung nicht Teil des Trainingsprozesses ist, kann deren Modell, um das obige Beispiel aufzugreifen, nach dessen Training auf (vollständigen) menschlichen Körpern sowohl Arme als auch Beine oder ganze Unterkörper sowie beliebige andere fehlende Bereiche des menschlichen Körpers komplettieren.

Neben der Vervollständigung von menschlichen Körpern, komplettieren Litany u. a. in ihrer Arbeit [Lit+18] auch partielle Gesichter um zu zeigen, dass ihr Modell – im Gegensatz zu früheren Verfahren – auch auf verschiedene Klassen von Formen verallgemeinert. Um andere Klassen von Formen komplettieren zu lernen, muss entsprechend natürlich ein anderer Datensatz für das Training verwendet werden. Allerdings müssen bei dem *graph*

convolutional variational autoencoder von Litany u. a. im Unterschied zu früheren Verfahren nicht auch noch Änderungen am Modell vorgenommen werden.

4.4. *Autoencoders*

Bei einem *autoencoder* handelt es sich um ein künstliches neuronales Netzwerk, welches lernt Ausgaben zu erzeugen, die den während des Trainingsprozesses gesehenen Eingaben ähnlich sind. Ein *autoencoder* darf hierzu allerdings nicht einfach Kopien bereits gesehener Eingaben erstellen. Indem dem *autoencoder* gewisse Einschränkungen auferlegt werden, ist er gezwungen nach Mustern in den Eingabedaten zu suchen und diese Muster auszunutzen um die Eingabedaten auf effizientere Weise zu repräsentieren. Dabei muss ein *autoencoder* jedoch in der Lage dazu bleiben, aus einer internen alternativen Repräsentation eines Eingabedatums – die Codierung, welche in ihrer Gesamtheit den *latent space* (‚verborgener Raum') bilden – die ursprüngliche Eingabe wiederherzustellen.

Aufgrund des sich aus den Einschränkungen ergebenden Flaschenhalses in der Netzwerkarchitektur lernt ein *autoencoder* die wichtigen Merkmale (*features*) einer Eingabe zu erkennen und die weniger wichtigen Merkmale zu verwerfen. *Autoencoders* werden daher traditionell gerne als *feature detector* oder, ähnlich der Hauptkomponentenanalyse, zur Reduzierung der Dimension (*dimensionality reduction*) des Parameterraums eingesetzt. Eine der frühesten Anwendungen von *autoencoders* wird von Goodfellow, Bengio und Courville [GBC16] auf LeCun und in das Jahr 1987 zurückgeführt. Mittlerweile werden *autoencoders* in verschiedenen Ausführungen vor allem aber als generierendes Modell (*generative model*) zur Erzeugung neuer Daten verwendet.

Ein *autoencoder* besteht aus zwei Teilen, die in Abbildung 4.2 auf der nächsten Seite dargestellt sind: einem Encoder (*recognition network*), welcher die Eingaben in eine interne Repräsentation (Codierung oder auch *latent space representation*) konvertiert, und einem Decoder (*generative network*), welcher die interne Repräsentation in die den Eingaben ähnelnden Ausgaben überführt. Die Reduktion der Dimension ergibt sich aus der Tatsache, dass die mittleren Schichten der *autoencoder*-Architektur signifikant kleiner als die Eingabeschicht sind.

Mathematisch ausgedrückt besteht das Ziel eines *autoencoder* darin, für eine Eingabe x den durch $L_r = \|\text{dec} \circ \text{enc}(x) - x\|_2$ definierten *reconstruction loss* zu minimieren.

4.5. Satz von Bayes

Der Satz von Bayes ist eine der wichtigsten Formeln, wenn nicht gar die wichtigste, in der Wahrscheinlichkeitsrechnung. Auf ihm und dem bayesschen Wahrscheinlichkeitsbegriff basiert die bayessche Statistik, ein Zweig der schließenden Statistik, bei dem ein Vorwissen (A-priori-Verteilung) über die zu untersuchende Variable kombiniert wird mit durch Beobachtung von Daten neu gewonnenen Erkenntnissen (Likelihood-Funktion). Die gesuchte Wahrscheinlichkeit (A-posteriori-Verteilung) der aufgestellten Hypothese kann aktualisiert werden, wenn mehr Daten verfügbar werden.

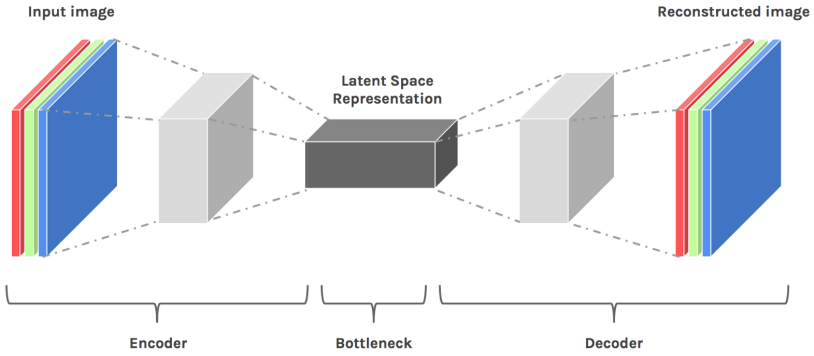


Abbildung 4.2.: **Schematische Darstellung der *autoencoder*-Architektur.** Ein *autoencoder* besteht aus mindestens drei Schichten: eine Eingabeschicht; mindestens eine signifikant kleinere Schicht, die die Kodierung formt; eine Ausgabeschicht. Bei der Ein- und Ausgabe des *autoencoder* in dieser Abbildung handelt es sich um Bilder. [Lar+18]

Der Satz von Bayes im Kontext dieses Kapitels ist gegeben durch

$$p(\mathbf{z}|\mathbf{X}) = \frac{p(\mathbf{X}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{X})}, \tag{4.1}$$

wobei die bedingte Wahrscheinlichkeit $p(\mathbf{z}|\mathbf{X})$ die gesuchte A-posteriori-Verteilung über die *latent variables* $\mathbf{z}$ ist, $\mathbf{X}$ die beobachteten Daten repräsentiert, die bedingte Verteilung $p(\mathbf{X}|\mathbf{z})$ die Likelihood-Funktion ist und $p(\mathbf{z})$ die im Vorfeld bekannte A-priori-Verteilung über die *latent variables* $\mathbf{z}$ ist. Auf die nähere Bedeutung der hier mit $\mathbf{z}$ bezeichneten Hypothese und der mit $\mathbf{X}$ bezeichneten Evidenz wird in Abschnitt 4.9 eingegangen.

Auf der bayesschen Inferenz (*Bayesian inference*) bauen verschiedene bayessche Verfahren auf, beispielsweise die in Abschnitt 4.6 vorgestellten *variational autoencoders*.

4.6. *Variational autoencoders*

Bei einem *variational autoencoder* handelt es sich um einen probabilistischen *autoencoder*, da dessen Ausgabe in gewissem Maß zufällig bestimmt wird. Der Vorteil eines *variational autoencoder* gegenüber einem gewöhnlichen *autoencoder* ist, dass ein *variational autoencoder* aus derselben Eingabe unterschiedlich aussehende Ausgaben generieren kann, die allerdings ähnlich zueinander sind.

Ein gewöhnlicher *autoencoder* lernt jedes der Merkmale (*features*) der Trainingsdaten mit einem Skalar zu beschreiben. Ein Merkmal kann beispielsweise sein, ob eine Person männlich oder weiblich ist, ob sie eine Brille trägt oder nicht, ob sie Bart trägt oder nicht, et cetera. Diese Skalare bilden gemeinsam den *latent vector* $\mathbf{z}$, aus dem der Decoder die Ausgabe generiert. Für dieselbe Eingabe generiert ein gewöhnlicher *autoencoder* somit immer dieselbe Ausgabe.

Ein *variational autoencoder* hingegen lernt die Merkmale mittels Wahrscheinlichkeitsverteilungen statt mit Skalaren zu beschreiben. Bei einer zufälligen Abtastung der für

jedes Merkmal aufgestellten Wahrscheinlichkeitsverteilung liefert jede Auswertung des neuronalen Netzwerks andere skalare Werte für die Merkmale, die somit auch bei jeder Auswertung gemeinsam eine unterschiedliche Codierung $\mathbf{z}$ (*latent vector*) bilden, aus welcher schließlich der Decoder unterschiedliche Instanzen generiert.

Zu diesem Zweck wird die Kostenfunktion, die bei einem gewöhnlichen *autoencoder* aus nur einem einzigen Term L_r (dem *reconstruction loss*) besteht, um einen zweiten Term L_{KL} erweitert. Dieser zusätzliche Term L_{KL} sorgt dafür, dass die einzelnen Vektorkomponenten der vom Encoder erzeugten Codierungen zu einem gewissen Grad so aussehen, als würden sie einer Wahrscheinlichkeitsverteilung (üblicherweise eine Normalverteilung) folgen.

Ein gewöhnlicher *autoencoder* hat zudem den Nachteil, dass mit dem *reconstruction loss* allein die Effektivität des Decoders – der generierenden Komponente des Modells – nicht objektiv beurteilt werden kann, und somit nicht erkennbar ist, wann der Trainingsprozess des Modells beendet werden sollte. Im Gegensatz dazu wird beim Training eines *variational autoencoder* gelernt, die Wahrscheinlichkeitsverteilung für die einzelnen Merkmale jeweils einem Vorwissen (A-priori-Verteilung) in Form einer Normalverteilung anzunähern, womit bei guter Wahl der A-priori-Verteilung die Effektivität des Decoders sichergestellt ist und dieses Problem des *autoencoder* sich hier nicht ergibt.

Der Knackpunkt des *variational autoencoder* ist die Idee, das Problem des Erlernens einer alternativen Repräsentation $\mathbf{z}$ als ein *variational Bayesian inference* Problem zu behandeln. Dabei steht der Begriff *Bayesian inference* für die bayessche Inferenz und der Term *variational* rührt von der Variationsrechnung von Euler und Lagrange her. Was dies bedeutet wird nach den folgenden Sätzen ersichtlich.

Ein angestrebtes Teilziel des in Unterabschnitt 4.9.1 beschriebenen Trainingsprozesses ist die Ermittlung der durch Gleichung 4.1 auf der vorherigen Seite bestimmten A-posteriori-Verteilung $p(\mathbf{z}|\mathbf{X})$. Jedoch handelt es sich bei

$$p(\mathbf{X}) = \int p(\mathbf{X}|\mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z}$$

im Nenner von Gleichung 4.1 um ein nicht in Polynomialzeit lösbares Integral, was die Berechnung der gesuchten A-posteriori-Verteilung $p(\mathbf{z}|\mathbf{X})$ praktisch unmöglich macht. Bei der *variational Bayesian inference* wird eine solche nicht berechenbare Verteilung wie hier $p(\mathbf{z}|\mathbf{X})$ stattdessen angenähert durch eine als *variational distribution* bezeichnete Verteilung $q(\mathbf{z}|\mathbf{X})$, die deutlich einfacher zu berechnen ist. Die Bezeichnung $q_\theta(\mathbf{z}|\mathbf{X})$ repräsentiert den Encoder und $p_\phi(\mathbf{X}|\mathbf{z})$ den Decoder, wobei θ die zu lernenden Parameter (Gewichte und Bias) des Encoders sind und ϕ die zu lernenden Parameter des Decoders.

Der neu eingeführte zweite Term der Kostenfunktion wird typischerweise mittels der in Abschnitt 4.7 vorgestellten Kullback-Leibler-Divergenz realisiert. Mit dieser wird die Unähnlichkeit zwischen der gelernten, als *variational distribution* oder auch *latent space distribution* bezeichneten Verteilung und einer A-priori-Verteilung (für gewöhnlich eine zentrierte Normalverteilung) gemessen und schlussendlich minimiert.

Die zu optimierende Kostenfunktion eines *variational autoencoder* ist damit beschrieben durch

$$L = L_r(\mathbf{X}, \mathbf{X}') + L_{\mathrm{KL}}\big(q(\mathbf{z}|\mathbf{X}), p(\mathbf{z})\big), \tag{4.2}$$

wobei $\mathbf{X}$ für die Eingabedaten steht und $\mathbf{X}'$ für die vom Decoder rekonstruierte Ausgabe.

Der *variational autoencoder* wurde im Jahr 2014 von Kingma und Welling [KW14] vorgestellt und ist auch heute noch, neben dem *generative adversarial network (GAN)* von Goodfellow u. a. [Goo+14] aus dem Jahr 2014, das wichtigste generierende Modell. Die Autoren Kingma und Welling liefern mit ihrem Nachfolgewerk [KW19] eine umfassende Einführung in das Thema *variational autoencoder*. Auch Blei, Kucukelbir und McAuliffe [BKM17] bieten dem interessierten Leser einen guten Einstieg für ein tieferes Verständnis.

4.7. Kullback-Leibler-Divergenz

Die Kullback-Leibler-Divergenz ist ein Maß für die Unähnlichkeit zweier Wahrscheinlichkeitsverteilungen. Für den Anwendungsfall, dass eine Wahrscheinlichkeitsverteilung P durch eine andere Wahrscheinlichkeitsverteilung Q approximiert werden soll, lässt sich mittels der Kullback-Leibler-Divergenz messen wie sehr die sich annähernde Verteilung Q von der Referenz P abweicht. Eine solche Annäherung einer schwierig zu berechnenden Wahrscheinlichkeitsverteilung durch eine einfacher zu berechnende Verteilung ist die Grundidee der *variational Bayesian inference* und der *variational autoencoders*.

Die Kullback-Leibler-Divergenz ist daher Bestandteil der während des Trainingsprozesses zu minimierenden Kostenfunktion (Gleichung 4.2 auf der vorherigen Seite) eines *variational autoencoder* und des in diesem Kapitel vorgestellten Modells. Sie ist definiert durch

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x,$$

wobei p und q die aus den Wahrscheinlichkeitsverteilungen P und Q abgeleiteten Wahrscheinlichkeitsdichtefunktionen sind.

4.8. *Shape correspondence*

Bei dem als *shape correspondence* bezeichneten Problem geht es darum herauszufinden, welcher Punkt y einer Form $\mathcal{Y}$ einem gegebenen Punkt x einer anderen Form $\mathcal{X}$ entspricht. Weiß man beispielsweise für jeden Punkt $y \in \mathcal{Y}$ einer Referenzform $\mathcal{Y}$, die einen Menschen repräsentiert, um welchen Körperteil es sich bei dem Punkt y handelt, so wüsste man gerne auch für die durch Verformung von $\mathcal{Y}$ entstandene Form $\mathcal{Y}'$ oder für einen, durch Form $\mathcal{X}$ modellierten, anderen Menschen, für den diese Information über die Zuordnung zu einem Körperteil nicht vorhanden ist, um welches Körperteil es sich bei einem Punkt $y \in \mathcal{Y}'$ beziehungsweise $x \in \mathcal{X}$ handelt.

Dieses *shape correspondence* genannte Problem liegt vielen weiteren Problemen der Computergrafik sowie des maschinellen Sehens (*computer vision*) zugrunde, beispielsweise dem Transfer von Texturen oder Posen zwischen verschiedenen Oberflächen, oder der in diesem Kapitel behandelten Vervollständigung unvollständiger Formen. Das Finden einer intrinsischen *shape correspondence* zwischen verschiedenen Formen – also eine *correspondence*, die von einer Verformung unberührt bleibt – ermöglicht das Lösen von Problemen höherer Ebene.

Die Verfahren auf dem aktuellen Stand der Technik interpretieren *shape correspondence* als Klassifikationsproblem und weisen jedem Punkt beziehungsweise – im Kontext eines Graphen – Knoten einer Eingabeform $\mathcal{X}$ einen Punkt beziehungsweise Knoten einer Referenzform $\mathcal{Y}$ zu. Das in Unterunterabschnitt 3.4.5.3 vorgestellte *Mixture model network (MoNet)* [Mon+17], welches in Unterabschnitt 4.9.2 noch eine wichtige Rolle spielen wird, gehört zu diesen Verfahren.

Für eine detaillierte Übersicht über die Literatur verweisen Litany u. a. [Lit+18] auf die Veröffentlichungen [KHC11; Bia+16]. Auch Bronstein u. a. [Bro+17b] bieten tiefere Einblicke in das Problem *shape correspondence*.

4.9. Funktionsweise im Detail

Wie bereits in Abschnitt 4.2 angedeutet, ist das Arbeitsprinzip der in diesem Kapitel vorgestellten Methode zur entkoppelten Vervollständigung verformbarer Formen in zwei Phasen unterteilt, wovon die eine während des Trainings des neuronalen Netzwerkes erfolgt und der andere während der Auswertung des neuronalen Netzwerkes. Diese beiden Phasen werden in den folgenden Abschnitten näher beschrieben.

4.9.1. Training des neuronalen Netzwerkes

Während des Trainingsprozesses lernt ein *variational autoencoder* vollständige dreidimensionale Formen zu generieren. Im Zentrum steht dabei die von Verma, Boyer und Verbeek [VBV17] neu definierte *graph convolutional operation*. Ebenfalls eine bedeutende Rolle spielt die in Abschnitt 4.7 vorgestellte Kullback-Leibler-Divergenz, welche Teil der zu minimierenden Kostenfunktion ist. Im Folgenden werden diese Bestandteile des Modells der Reihe nach vorgestellt.

Allgemeiner Ablauf

Bei den zu vervollständigenden Formen in den Untersuchungen von Litany u. a. [Lit+18] handelt es sich um Oberflächenmodelle und Polygonnetze. Wie in Unterabschnitt 2.9.2 auf Seite 22 beschrieben, lässt sich eine Mannigfaltigkeit durch Abtastung in eine Punktwolke überführen, welche sich als Graph interpretieren lässt, welcher sich wiederum in ein Polygonnetz (*mesh*) überführen lässt. Umgekehrt lässt sich auch ein Polygonnetz durch Weglassen dessen Facetten in einen Graphen überführen.

Für einen Graphen lässt sich jedem Knoten des Graphen eine dreidimensionale Vektorrepräsentation zuordnen, die *vertex embedding* genannt wird. Bei N Knoten wird eine als Graph interpretierte Form aufgefasst als das dreidimensionale *vertex embedding* $\mathbf{X} \in \mathbb{R}^{3\times N}$.

Der Encoder des *variational autoencoder* encodiert ein solches dreidimensionales *embedding* $\mathbf{X}$ eines Eingabepolygonnetzes als *latent vector* $\mathbf{z} = \mathrm{enc}(\mathbf{X})$. Die Vektorkomponenten von $\mathbf{z}$ sind dabei jeweils wie in Abschnitt 4.6 ausgeführt durch eine Wahrscheinlichkeitsverteilung beschrieben. Der Decoder erhält einen solchen *latent vector* $\mathbf{z}$ als Eingabe und weist ihm ein dreidimensionales *embedding* $\mathbf{X'} = \mathrm{dec}(\mathbf{z})$ eines Polygonnetzes zu.

Auf diese Weise wird der Trainingsprozess des *variational autoencoder* mittels *vollständiger* Formen durchgeführt, um einerseits eine für Unterabschnitt 4.9.2 relevante

Referenzform sowie andererseits einen *latent space* bestehend aus Codierungen zu lernen, welcher die Einbettung der Knoten dieser Referenzform in den dreidimensionalen Raum parametrisiert.

Die Interna des *variational autoencoder* und auch die des Encoders $\text{enc}(\mathbf{X})$ beziehungsweise $q_\theta(\mathbf{z}|\mathbf{X})$ und des Decoders $\text{dec}(\mathbf{z})$ beziehungsweise $p_\phi(\mathbf{X}|\mathbf{z})$ hängen stark von der gewählten Repräsentation der Formen ab.

Einführung einer *graph convolutional operation*

Um auf Polygonnetzen eine der klassischen Faltung ähnliche Operation durchzuführen, setzen Litany u. a. auf die Vorarbeit des von Verma, Boyer und Verbeek [VBV17] vorgestellten *FeaStNet* und übernehmen die von Verma, Boyer und Verbeek auf irreguläre Graphen verallgemeinerte *convolutional layer*. Bei dieser Verallgemeinerung der Faltung handelt es sich um eine kartografierende Methode, die in Abschnitt 3.4 für Mannigfaltigkeiten vorgestellt wurden.

Bevor im Folgenden die von Verma, Boyer und Verbeek [VBV17] verallgemeinerte *convolutional layer* vorgestellt wird, sei an dieser Stelle an die bekannte Faltungsschicht klassischer Faltungsnetze erinnert, welche definiert ist durch

$$\mathbf{y}_i = \mathbf{b} + \sum_{m=1}^{M} \mathbf{W}_m \mathbf{x}_{n(m,i)}, \tag{4.3}$$

wobei mittels Gewichtsmatrizen $\mathbf{W}_m \in \mathbb{R}^{E\times D}$ die Eingabe-*feature*-Vektoren $\mathbf{x} \in \mathbb{R}^D$ auf die Ausgabe-*feature*-Vektoren $\mathbf{y} \in \mathbb{R}^E$ abgebildet werden. Mit D wird die Anzahl der *feature maps* der Eingabe bezeichnet, mit E die Anzahl der *feature maps* der Ausgabe. Weiterhin ist $\mathbf{b} \in \mathbb{R}^E$ der Biasvektor. Bei Filtern der Größe $h \times w$ ist $M = h \cdot w$ die Anzahl der Gewichtsmatrizen. Die Funktion n liefert zu einem Pixel i für die Positionen $m = 1, \ldots, M$ der umliegenden Pixel den jeweiligen Index seiner M Nachbarpixel. Auf diese Weise wird jeder der M Gewichtsmatrizen genau einer der M Nachbarn eines Pixels zugewiesen. Somit wird jeder der M Vektoren $\mathbf{x}$, die jeweils zu einem der M Nachbarn eines zentralen Pixels i gehören, mit einer anderen der M Gewichtsmatrizen $\mathbf{W}$ multipliziert. Das Verfahren ist in Abbildung 4.3a auf Seite 86 illustriert.

Die Hauptschwierigkeit im Fall irregulärer Graphen ist – mangels der in Unterabschnitt 2.11.2 beschriebenen fehlenden globalen Parametrisierung und der sich daraus ergebenden nicht existenten kanonischen Reihenfolge der Knoten, was eine *hard-assignment* genannte Eins-zu-eins-Abbildung praktisch unmöglich macht – die Frage nach der Definition einer analogen Zuweisung der Nachbarn zu Gewichtsmatrizen. Verma, Boyer und Verbeek lösen diese Frage mit der Einführung einer Gewichtungsfunktion q, die der von den kartografierenden Methoden aus Abschnitt 3.4 bekannten Gewichtungsfunktion v entspricht und in der Schicht vor der verallgemeinerten Faltungsschicht berechnet wird. Mit dem Gewicht $q_m(\mathbf{x}_i, \mathbf{x}_j)$ wird der zu einem Nachbarn j eines Knotens i gehörige Vektor $\mathbf{x}_j$ der m-ten Gewichtsmatrix zugewiesen, wobei die M Gewichte normiert sind, sodass sie sich zu 1 summieren. Es gilt also $\sum_{m=1}^{M} q_m(\mathbf{x}_i, \mathbf{x}_j) = 1$.

Mittels dieser als *soft-assignment* bezeichneten Zuweisung von benachbarten Knoten zu Gewichtsmatrizen lässt sich die verallgemeinerte Faltung ähnlich zu Gleichung 4.3

definieren als

$$\mathbf{y}_i = \mathbf{b} + \sum_{m=1}^{M} \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} q_m \left(\mathbf{x}_i, \mathbf{x}_j\right) \mathbf{W}_m \mathbf{x}_j, \tag{4.4}$$

wobei $\mathcal{N}_i$ die Menge der benachbarten Knoten eines Knotens i einschließlich i selbst ist und $|\mathcal{N}_i|$ die Anzahl der Elemente in dieser Menge ist. In anderen Worten, $\mathcal{N}_i$ bezeichnet den in Abschnitt 3.4 vorgestellten *patch* um einen Knoten i.

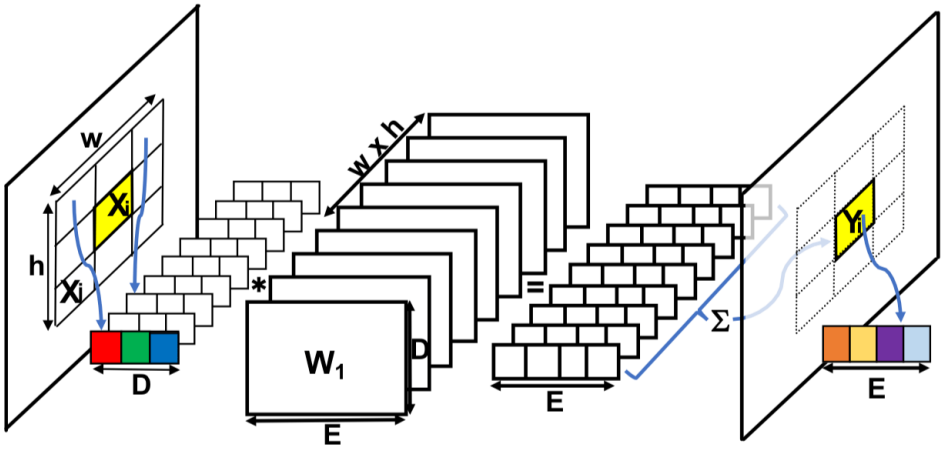
Die Eingabe einer mittels dieser Operation definierten *dynamic filtering convolutional layer* ist ein Vektorfeld auf der Knotenmenge des als Graph interpretierten Polygonnetzes, das jedem Knoten i des Graphen einen *feature vector* $\mathbf{x}_i$ zuweist. Auch die Ausgabe der Schicht ist ein Vektorfeld, welches einem Knoten den mittels Gleichung 4.4 berechneten *feature vector* $\mathbf{y}_i$ zuweist, welcher sich in der Dimension von $\mathbf{x}_i$ unterscheiden kann. Diese *graph convolutional operation* wird in Abbildung 4.3b veranschaulicht.

Die Gewichtungsfunktionen $q_m\left(\mathbf{x}_i, \mathbf{x}_j\right)$ sind die positiven Gewichte der Kanten in dem *patch*, wobei $q_m\left(\mathbf{x}_i, \mathbf{x}_j\right) \propto \exp\left(\mathbf{u}_m^{\mathsf{T}}\left(\mathbf{x}_i - \mathbf{x}_j\right) + c_m\right)$. Die Gewichtungsfunktionen q sind also direkt proportional zur genannten Exponentialfunktion. Die lernbaren Parameter der Faltungsschicht sind die Gewichtsmatrix $\mathbf{W}_m$, der Vektor $\mathbf{u}_m$ und der Skalar c_m der Gewichtungsfunktion sowie der Biasvektor $\mathbf{b}$.

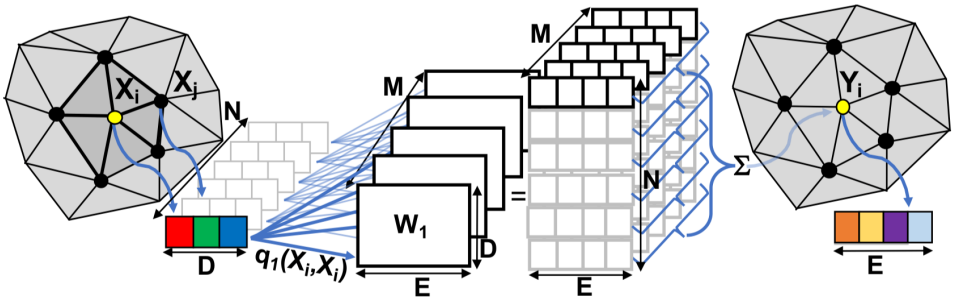
Die Anzahl M der Gewichtsmatrizen ist ein im Vorfeld, beim Entwurf des Modells festgelegter Parameter. Im Gegensatz zur klassischen Faltung in herkömmlichen euklidischen Faltungsnetzwerken ist bei dieser *graph convolutional operation* die Anzahl M der Gewichtsmatrizen und somit auch die Anzahl der Parameter des Modells unabhängig von der Größe $|\mathcal{N}_i|$ der Nachbarschaft. Daher erhöhen *graph convolutional* Filter mit größerem *spatial support* – also Filter, die einen größeren Bereich abdecken, bei klassischen Faltungsnetzwerken etwa 5×5-Faltungskerne anstelle von 3×3-Faltungskernen – anders als euklidische Faltungsnetzwerke auch nicht die Anzahl der Parameter. Da bei dieser *graph convolutional operation* eine Abbildung zwischen Gewichten und Nachbarn gelernt wird, ist es somit nicht notwendig, dass die Gewichtsmatrizen sich die Gewichte teilen (*weight sharing*). Selbst bei größeren Filtern ist dies nicht erforderlich. *Weight sharing* ist der in Unterabschnitt 3.4.4 beschriebene Ansatz klassischer Faltungsnetzwerke zur Lösung des *curse of dimensionality* genannten Problems. Weiterhin sind keine anderen Techniken wie die Dilatation der Filter nötig.

Definition der Kostenfunktion

An dieser Stelle wird die während des Trainingsprozesses zu minimierende Kostenfunktion näher beschrieben, welche sich wie für *variational autoencoders* üblich aus zwei Bestandteilen zusammensetzt: dem *shape reconstruction loss* und dem *regularization prior loss*. Mittels dem *shape reconstruction loss* soll erreicht werden, dass die Ausgabe möglichst wieder der Eingabe entspricht. Der *prior loss* dient zur Regularisierung, was bedeutet, dass durch ihn das Problem des *overfitting* verhindert werden soll. Von *overfitting* spricht man im Kontext neuronaler Netzwerke, wenn das Modell zu sehr an die während des Trainingsprozesses beobachteten Daten angepasst wird und dadurch die Fähigkeit, auf zuvor ungesehene Daten zu verallgemeinern, verliert. Bayessche Methoden wie etwa *variational autoencoders*



(a) Veranschaulichung von Gleichung 4.3



(b) Veranschaulichung von Gleichung 4.4

Abbildung 4.3.: **Illustrationen von Faltungsschichten.** Abbildung 4.3a veranschaulicht die Funktionsweise einer Faltungsschicht eines klassischen Faltungsnetzwerks ohne den Einsatz von *weight sharing*. Abbildung 4.3b veranschaulicht die Funktionsweise der *graph convolutional layer* von Verma, Boyer und Verbeek. [VBV17]

setzen Regularisierung durch, indem sie bestrafen, wenn die Codierungen des *latent space* von den mittels einer A-priori-Verteilung modellierten Grundannahmen abweichen.

Der *shape reconstruction loss* L_r ist definiert durch $L_r = \|\text{dec} \circ \text{enc}(\mathbf{X}) - \mathbf{X}\|_2$ und soll dafür sorgen, dass nach Encodierung einer Eingabe $\mathbf{X}$ und der anschließenden Dekodierung möglichst wieder die ursprüngliche Eingabe $\mathbf{X}$ zurückgeliefert wird. Die alleinige Verwendung des *shape reconstruction loss* würde den *variational autoencoder* zu einem gewöhnlichen *autoencoder* reduzieren. Die Hinzunahme des *regularization prior loss* macht aus dem *autoencoder* ein probabilistisches Modell, hier einen *variational autoencoder*.

Die Aufgabe des *variational autoencoder* ist es, eine *variational distribution* $q(\mathbf{z}|\mathbf{X})$ zu lernen, welche eine nur in Exponentialzeit bestimmbare A-posteriori-Verteilung $p(\mathbf{z}|\mathbf{X})$ approximieren soll. Dazu wird eine unveränderliche A-priori-Verteilung $p(\mathbf{z})$ herangezogen, welche das Vorwissen über die Merkmale der Formen mathematisch modelliert. Der *regularization prior loss* kann als Strafterm aufgefasst werden, welcher die Abweichung der zu lernenden Verteilung $q(\mathbf{z}|\mathbf{X})$ von der A-priori-Verteilung $p(\mathbf{z})$ bestraft. Der *prior loss* L_p dient somit dazu, eine bekannte, selbst gewählte A-priori-Verteilung $p(\mathbf{z})$ über die *latent variables* des *latent vector* $\mathbf{z}$ durchzusetzen. Gemessen wird L_p mittels der in Abschnitt 4.7 vorgestellten Kullback-Leibler-Divergenz. Definiert ist L_p durch $L_p = D_{\text{KL}}\left(q(\mathbf{z}|\mathbf{X}) \parallel p(\mathbf{z})\right)$.

Bei $p(\mathbf{z})$ handelt es sich hier um die mehrdimensionale Normalverteilung $\mathcal{N}(\mathbf{0}, \mathbf{I})$ mit Erwartungswertvektor $\boldsymbol{\mu} = \mathbf{0}$ und der Einheitsmatrix $\mathbf{I}$ als Kovarianzmatrix, was die übliche Wahl für eine A-priori-Verteilung ist. Es gilt also $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$. Die Kovarianzmatrix übernimmt bei mehrdimensionalen Normalverteilungen die Rolle der skalaren Varianz σ^2.

Insgesamt setzt sich die von Gleichung 4.2 bekannte Kostenfuntion des *variational autoencoder* zusammen als $L = L_r + \lambda L_p$, wobei hier noch ein passend zu wählender Parameter $\lambda \geq 0$ steuert wie ähnlich die zu lernende Verteilung $q(\mathbf{z}|\mathbf{X})$ der A-priori-Verteilung $p(\mathbf{z})$ werden soll.

Dieser Parameter λ wird verwendet, da L_r und L_p gegensätzliche Ziele anstreben und daher in Konflikt miteinander stehen. Wohingegen die Minimierung von L_r bedeutet, dass derjenige Vektor $\mathbf{z}$ ermittelt wird, für den der Rekonstruktionsfehler minimal ist, und dass dazu nötigenfalls auch von dem modellierten, möglicherweise nicht optimalen oder realitätsnahen Vorwissen abgewichen werden kann, hat die Minimierung von L_p zur Folge, dass die zu lernenden Verteilungen sich jeweils möglichst genau der modellierten Normalverteilung annähern. Das Sampling der gelernten Normalverteilungen liefert jeweils eine Komponente des gesuchten Vektors $\mathbf{z}$. Einerseits sollen die Vektorkomponenten von $\mathbf{z}$ genau bestimmt sein, andererseits sollen sie normalverteilt sein, also eine gewisse Varianz aufweisen. Einerseits sollen sie erforderlichenfalls von der modellierten Verteilung abweichen dürfen, andererseits sollen sie der modellierten Verteilung folgen. Mit λ kann durchgesetzt werden, dass nicht einfach eine extrem enge Normalverteilung gelernt wird sowie gesteuert werden, wie ähnlich die gelernten Verteilungen den modellierten Verteilungen sehen müssen. Dieser Konflikt in der Kostenfunktion ist auch der Grund, warum von *variational autoencoders* generierte Bilder dazu tendieren verschwommen zu sein, während von *generative adversarial networks* generierte Bilder dieses Problem entwurfsbedingt nicht haben.

Bei der Wahl von λ gilt es einen guten Kompromiss zwischen realistischem Aussehen der generierten Formen und der Vielfalt der Formen, die das Modell erzeugen kann, zu finden. Größere Werte für λ gewichten den Realismus höher, kleinere Werte gewichten

die Vielfalt höher. Litany u. a. [Lit+18] sprechen der Vielfalt eine höhere Bedeutung zu, weshalb Litany u. a. mit $\lambda = 10^{-8}$ einen eher kleineren Wert wählen.

Neuere Arbeiten wie die von Dai und Wipf [DW19] haben Wege untersucht wie λ gelernt werden kann, wodurch die Performanz eines *variational autoencoder* im Vergleich zu selbst ermittelten λ-Werten erwartungsgemäß gesteigert werden kann.

4.9.2. Auswertung des neuronalen Netzwerkes

In diesem Abschnitt wird thematisiert, wie das trainierte neuronale Netzwerk aus einer unvollständigen Eingabe eine vollständige Ausgabe erzeugt. Dazu ist nur noch der Decoder des *variational autoencoder* von Interesse. Der Encoder wird nach abgeschlossenem Training des Modells nicht länger benötigt.

Wie in Unterabschnitt 4.9.1 beschrieben erhält der *variational autoencoder* während des Trainingsprozesses als Eingaben verschiedene *embeddings* $\mathbf{X}$ *vollständiger* Formen. Ein solches *embedding* $\mathbf{X}$ einer Form encodiert der Encoder als *latent vector* $\mathbf{z}$ und der Decoder lernt aus einem Vektor $\mathbf{z}$ das *embedding* $\mathbf{X}$ und somit die Form wiederherzustellen. Der Decoder weist dazu dem *latent vector* $\mathbf{z}$, den er als Eingabe erhält, ein dreidimensionales *embedding* $\mathbf{X}' \in \mathbb{R}^{3\times N}$ zu, durch das eine Form repräsentiert wird. Es gilt also $\mathbf{z} = \text{enc}(\mathbf{X})$ und $\mathbf{X}' = \text{dec}(\mathbf{z})$. Durch diesen Trainingsprozess wird eine Referenzform gelernt, die für das Ermitteln der in Abschnitt 4.8 vorgestellten *shape correspondence* wichtig ist.

Zum Zeitpunkt der Inferenz erhält der *variational autoencoder* als Eingabe ein *embedding* $\mathbf{Y}$ einer *unvollständigen* Form. Um für eine partielle Eingabe $\mathbf{Y}$ eine plausible komplette Form bestimmen zu können, muss zunächst ermittelt werden, welchen Knoten der während des Trainingsprozesses gelernten Referenzform die Knoten der unvollständigen Eingabe $\mathbf{Y}$ entsprechen. Zum Lösen dieses Problems wird das in Unterunterabschnitt 3.4.5.3 vorgestellte *Mixture model network (MoNet)* eingesetzt. Die mittels eines *Mixture model network (MoNet)* berechnete *dense partial intrinsic correspondence* zwischen der inkompletten Eingabe $\mathbf{Y}$ und der gelernten Referenzform – daher die Bezeichnung der teilweisen Entsprechung, die sich unter Verformungen nicht ändert (*intrinsic*) – wird durch eine Permutationsmatrix $\mathbf{\Pi}$ repräsentiert. Die Multiplikation $\mathbf{X}'\mathbf{\Pi}$ dieser Permutationsmatrix $\mathbf{\Pi}$ mit einem vom Decoder generierten *embedding* $\mathbf{X}' \in \mathbb{R}^{3\times N}$ produziert eine Teilmenge der dreidimensionalen *vertex embeddings* der N Knoten des Polygonnetzes, die derart umsortiert wurde, dass die Reihenfolge der Knoten der generierten vollständigen Form $\mathbf{X}'$ mit der Reihenfolge ihrer entsprechenden Gegenstücke in der unvollständigen Eingabe $\mathbf{Y}$ übereinstimmt.

Vor diesem Hintergrund kann eine extrinsische Unähnlichkeit (*dissimilarity*) zwischen einer Eingabeform $\mathbf{Y}$ und einer Form $\mathbf{X}$ definiert werden. Diese Unähnlichkeit ist beschrieben durch $D(X, Y) = \|X - Y\|_2$ und kann gegebenenfalls noch mit der Konfidenz der *shape correspondence* an jedem Knoten der Form gewichtet werden. Da ein vom Decoder generiertes *embedding* $\mathbf{X}'$ einer vollständigen Form aus einem *latent vector* $\mathbf{z}$ erzeugt wird, besteht das Ziel während der Inferenz darin, denjenigen *latent vector* $\mathbf{z}^*$ zu finden, für den die Unähnlichkeit zwischen der Eingabe $\mathbf{Y}$ und dem zwecks Kompatibilität angepassten *embedding* $\mathbf{X}'\mathbf{\Pi}$ mit $\mathbf{X}' = \text{dec}(\mathbf{z})$ minimal ist.

Zunächst wird jedoch die Eingabe $\mathbf{Y}$ ausgerichtet an $\mathbf{X}'\mathbf{\Pi}$ mittels einer Kombination von Drehungen und Verschiebungen. Eine solche Kombination von Rotationen und Trans-

lationen heißt Bewegung im Sinne eines Elements der in Abschnitt 2.3 vorgestellten Bewegungsgruppe $\mathbb{E}(n)$. Betrachtet man von $\mathbb{E}(n)$ nur die Kombinationen von Rotationen und Translationen und lässt die Spiegelungen außen vor, so erhält man die *special Euclidean group* genannte und mit $\mathrm{SE}(n)$ bezeichnete Untergruppe der Bewegungsgruppe. Eine Ausrichtung $\mathbf{T}$ an eine Form oder aber auch eine Punktwolke mittels einer solchen (abstandserhaltenden) Kombination von Rotationen und Translationen wird im Englischen *registration* genannt. Dabei ist $\mathbf{T} \in \mathrm{SE}(3)$ die intrinsische Repräsentation einer solchen Bewegung im dreidimensionalen Fall. Wenn $\mathbf{X}' = \mathrm{dec}(\mathbf{z})$ gilt und $\mathbf{T}$ eine euklidische oder auch abstandserhaltende Transformation bezeichnet, dann ist zum Zwecke der Vervollständigung einer unvollständigen Eingabe somit die Lösung $\mathbf{z}^*$ des Minimierungsproblems

$$\min_{\mathbf{z},\mathbf{T}\in\mathrm{SE}(3)} \mathrm{D}(\mathrm{dec}(\mathbf{z})\mathbf{\Pi}, \mathbf{TY}) \tag{4.5}$$

gesucht.

Zur Lösung dieses Optimierungsproblems geht man nach zufälliger Initialisierung alternierend vor. In abwechselnden Schritten werden $\mathbf{z}$ beziehungsweise $\mathbf{T}$ variiert bis das Minimum der Zielfunktion D beliebig genau angenähert ist. Das *embedding* $\mathbf{X}' = \mathrm{dec}(\mathbf{z}^*)$ repräsentiert dann die gesuchte vervollständigte Form. Da Gleichung 4.5 insbesondere für größere fehlende Bereiche nicht unbedingt eine eindeutige Lösung hat, können durch verschiedene Initialisierungen auch unterschiedliche zulässige Ausgaben erzeugt werden. Abbildung A.1 auf Seite 125 zeigt wie verschiedene mögliche Ausgaben für dieselbe Eingabe aussehen können.

4.10. Modellarchitektur

Da es sich bei dem Modell von Litany u. a. um einen *(variational) autoencoder* handelt, besteht dieses wie schon in Abschnitt 4.4 angeführt aus zwei groben Teilen: dem Encoder und dem Decoder. Die Modellarchitektur ist in Abbildung 4.4 auf der nächsten Seite näher erläutert, worin auch die in Abschnitt 4.9 beschriebene Funktionsweise illustriert wird.

4.11. Erkundung des *latent space*

Zum Abschluss dieses Kapitels ist noch der Hinweis erwähnenswert, welche interessante Struktur der gelernte *latent space* aufweist.

Wenn durch zwei *latent vectors* $\mathbf{z}$ und $\mathbf{z}'$ beispielsweise die Start- beziehungsweise Endpose einer Bewegung encodiert ist, so können sämtliche Zwischenposen der Bewegung interpoliert werden, indem der Decoder auf konvexe Kombinationen von $\mathbf{z}$ und $\mathbf{z}'$ angewendet wird. Auch Arithmetik ist im *latent space* möglich. Wenn beispielsweise $\mathbf{z}'$ einen Menschen mit stark angehobenem linken Bein repräsentiert und $\mathbf{z}$ eine Pose mit weniger stark angehobenem linken Bein repräsentiert, so kann der Unterschied $\mathbf{z} - \mathbf{z}'$ dieser zwei *latent vectors* auf einen dritten *latent vector* $\mathbf{z}''$, welcher einen Menschen mit stark angehobenem rechten Bein repräsentiert, addiert werden, um dadurch eine Pose mit weniger stark angehobenem rechten Bein zu generieren. Die Interpolation und Arithmetik im *latent space* werden in Abbildung A.2 auf Seite 126 veranschaulicht.

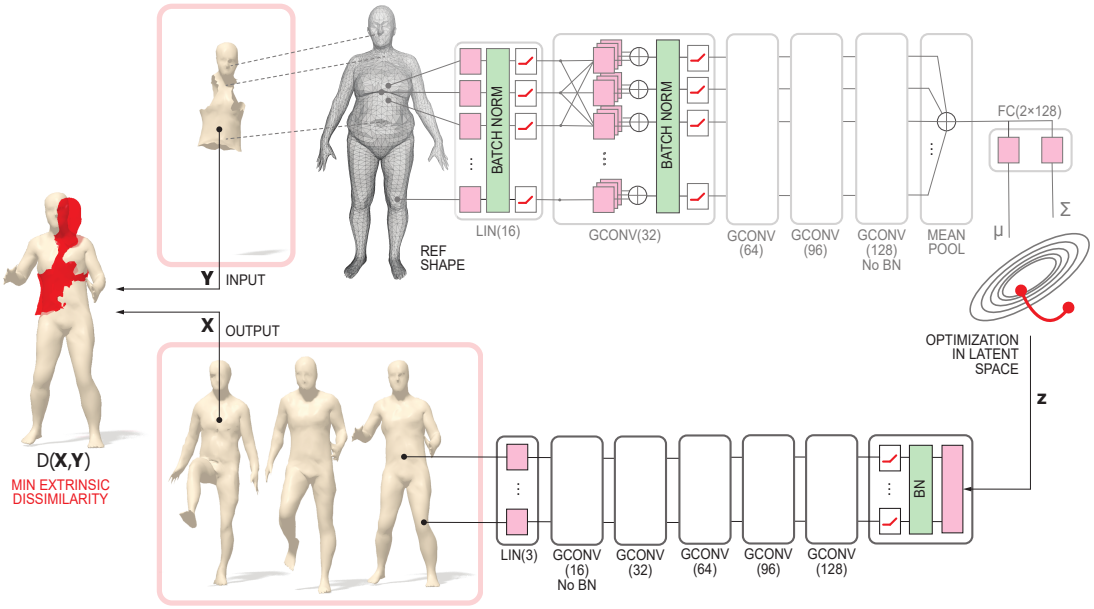


Abbildung 4.4.: **Schematische Darstellung der Modellarchitektur des *graph convolutional variational autoencoder*.** Illustriert ist der Ablauf während der Inferenz, nicht der Trainingsprozess, was daran erkennbar ist, dass eine unvollständige Form als Eingabe dient. Im oberen Zweig ist der Encoder schematisch skizziert, im unteren Zweig der Decoder. Der Encoder wird nach dem Trainingsprozess, also während der Inferenz, nicht mehr benötigt und ist daher ausgegraut dargestellt. *LIN* bezeichnet eine lineare Schicht. *BATCH NORM* beziehungsweise *BN* steht für *batch normalization* und wird von der Aktivierungsfunktion *ReLU (Rectified Linear Unit)* gefolgt. *GCONV* bezeichnet eine *graph convolutional layer*. Auf *mean pooling* folgt eine mit *FC* bezeichnete *fully connected layer*, welche die Erwartungswerte μ_i und Varianzen σ_i^2 mit $i = 1, \ldots, n$ zur Beschreibung der Merkmale der Eingabe mittels Wahrscheinlichkeitsverteilungen ausgibt. Zu Beginn der Inferenz wird mittels des in Unterunterabschnitt 3.4.5.3 vorgestellten *Mixture model network (MoNet)* eine Entsprechung (*correspondence*) zwischen der unvollständigen Eingabeform $\mathbf{Y}$ und der während des Trainingsprozesses gelernten Referenzform ermittelt, was die Permutationsmatrix $\mathbf{\Pi}$ produziert. Im Zuge eines Optimierungsproblems wird derjenige *latent vector* $\mathbf{z}$ ermittelt, für den die vom Decoder generierte vollständige Form $\mathrm{dec}(\mathbf{z})$ am besten zu der partiellen Eingabe $\mathbf{Y}$ passt. [Lit+18]

5. Klassifikation und Segmentierung von Punktwolken

In diesem Kapitel wird vorrangig die Klassifikation und Segmentierung von Punktwolken behandelt sowie auf welche Weise *geometric deep learning* sich zu diesem Zweck einsetzen lässt. Hierfür wird ein Verfahren von Wang u. a. [Wan+19] namens *Dynamic Graph CNN* vorgestellt, das im Gegensatz zu den Vorreitern [Qi+17a; Qi+17b] der direkten Verarbeitung von Punktwolken in der Lage dazu ist, neben der globalen Information auch die lokalen geometrischen Strukturen der Punktwolke zu erfassen. Zunächst werden Gründe für die Schwierigkeit des Lernens auf Punktwolken beschrieben und Besonderheiten aufgeführt, die beim Lernen auf Punktwolken beachtet werden müssen. Im Anschluss wird eine von Wang u. a. [Wan+19] neu eingeführte Operation zur Verallgemeinerung der Faltung vorgestellt. Abschließend wird auf den namensgebenden dynamisch aktualisierten Graphen eingegangen, auf den diese Operation angewendet wird.

5.1. Problembeschreibung

Punktwolken sind neben den in Kapitel 4 betrachteten Polygonnetzen eine weitere wichtige Art geometrischer Daten, da die meisten Geräte zur Akquise dreidimensionaler Daten (*3D sensing*) die gemessenen Rohdaten in Form einer Punktwolke ausgeben. Zu diesen Geräten gehören etwa die bereits in Kapitel 4 erwähnten Tiefensensoren wie Microsoft Kinect oder auch LiDAR-Systeme (Light Detection and Ranging), die unter anderem im Bereich des autonomen Fahrens große Anwendung finden. Punktwolken sind zudem die simpelste denkbare geometrische Repräsentation einer Form.

Bei einer Punktwolke handelt es sich um eine Menge von ungeordneten Punkten, die mittels Elementen eines euklidischen Raums $\mathbb{R}^n$ beschrieben werden. Die Punkte sind im einfachsten Fall dreidimensional und repräsentieren ihre Position im Raum. Die Dimension der Punkte kann allerdings auch höher sein, um neben der Position weitere Informationen zu beinhalten, etwa über Farbe oder Normalen. Trotz oder gerade wegen ihrer Einfachheit sind Punktmengen jedoch eine schwierig handzuhabende Datenstruktur, mit der klassische euklidische neuronale Netzwerke nicht gut umgehen können. Wie in Abschnitt 2.2 bereits angedeutet handelt es sich bei einer Punktwolke lediglich um eine Grundmenge, die keine topologische Struktur trägt, weshalb bei Punktmengen keine Abstraktion des Konzepts von Nähe existiert und man sich zu einem gegebenen Punkt keine Nachbarschaft vorstellen kann. Dies erschwert den Entwurf einer der klassischen Faltung ähnlichen Operation, die sich direkt auf Punktwolken anwenden lässt.

Aus diesem Grund werden Punktwolken traditionell in eine andere Repräsentation überführt, die eine reguläre Struktur aufweist und sich somit besser als Eingabe eines

euklidischen Faltungsnetzwerks eignet. Einer Punktwolke liegt anders als etwa einem Bildsignal kein Gitter – geschweige denn ein reguläres – zugrunde. Wie die bereits aus Kapitel 4 bekannten Polygonnetze sind Punktwolken irregulär, da die Positionen der Punkte kontinuierlich im Raum verteilt sind, was bedeuten soll, dass die Punkte sich an einer Stelle häufen können, wohingegen an anderen Stellen des Raums möglicherweise kaum oder keine Punkte vorhanden sind. Klassische Faltungsnetze erzielen jedoch entwurfsbedingt die besten Ergebnisse dann, wenn sie zur Verarbeitung von Daten mit Gitterstruktur, beispielsweise Bildsignale oder dreidimensionale Voxeldaten, eingesetzt werden. Nur für Eingaben mit einer regulären Struktur können euklidische Faltungsnetzwerke Techniken zur Kerneloptimierung wie das bereits in Abschnitt 4.9.1 erwähnte *weight sharing* umsetzen. Als mögliche alternative Repräsentation der Daten wird häufig eine von zwei Optionen gewählt: eine volumetrische Repräsentation in Form von dreidimensionalen Voxelgittern oder Sammlungen von Bildern (*multi-view based representation*), die man durch Projektion der Punktwolke auf die zweidimensionale Ebene erhält. [Wan+19]

Diese Überführung in eine andere Repräsentation erfolgt durch Abtastung der Punktwolke, ist jedoch nicht unproblematisch. Während die Dünnbesetztheit von Punktwolken zwar zu deren irregulären Struktur beiträgt und somit für einige der Schwierigkeiten bei der Verarbeitung dieser geometrischen Datenstruktur verantwortlich ist, ist die Dünnbesetztheit auch einer der großen Vorzüge von Punktwolken, da sie sich in geringem Speicherbedarf äußert. Die Überführung einer dünnbesetzten Punktwolke in eine andere Repräsentation resultiert dagegen oftmals in unnötig voluminösen Daten. Zudem werden bei der Abtastung der Punktwolke häufig Quantisierungsartifakte eingeführt, welche die natürlichen Invarianzen der Daten – also Eigenschaften der Daten, die bei gewissen Transformationen unverändert bleiben – verschleiern können. Ziel ist daher die direkte Verarbeitung von Punktwolken um auf den Umweg über eine andere Repräsentation verzichten zu können. Die direkte Verarbeitung von Punktwolken ist jedoch keineswegs einfach, da diese neben ihrer Irregularität einige weitere verkomplizierende Eigenschaften haben, die es beim Entwurf eines neuronalen Netzwerkmodells zu beachten gilt. Diese Eigenschaften werden in Abschnitt 5.2 beschrieben.

Die im Rahmen dieses Kapitels besprochenen Anwendungsfälle sind die Klassifikation und die Segmentierung von Punktwolken – zwei Probleme, für deren Lösung Faltungsnetzwerke sich, gesetzt den Fall euklidischer Daten, als hervorragend geeignet erwiesen haben. Für einen Lösungsansatz dieser beiden Aufgaben im Fall von Punktwolken sollen die mit euklidischen Faltungsnetzwerken gewonnenen Erkenntnisse auf Punktmengen übertragen werden. Bei der Segmentierung lässt sich unterscheiden zwischen dem Kennzeichnen der einzelnen Bestandteile eines Objekts (*part segmentation*) und dem von der Bedeutung eines Objekts in einer Szene abhängigen Markieren dieses Objekts (*semantic segmentation*). In beiden Ausprägungen ist die Segmentierung eine Erweiterung der Klassifikation, denn jedem Punkt der Punktwolke wird zunächst eine Klasse zugewiesen, woraufhin die Punkte abhängig von ihrer Klasse semantisch gruppiert werden. Abbildung 5.1 auf der nächsten Seite zeigt die drei Aufgabenstellungen.

Da es sich bei einer Punktwolke lediglich um eine Menge von Punkten handelt für welche, im Unterschied zu einem Graphen oder Polygonnetz, keine Nachbarschaftsbeziehungen explizit vorgegeben sind, ist eine sinnvolle Herangehensweise, die Topologie der durch

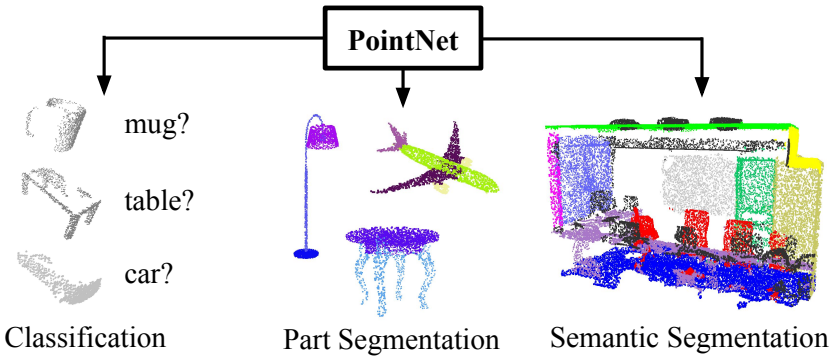


Abbildung 5.1.: **Darstellung der Klassifikation und Segmentierung von Punktwolken am Beispiel *PointNet* [Qi+17a].** Bei der Klassifikation einer Punktwolke wird der Punktwolke als Ganzes eine Klasse zugewiesen. Bei der *part segmentation* wird den ein Objekt darstellenden Punkten die Klasse des Bestandteils des Objekts zugewiesen, zu dem die Punkte gehören. Bei der *semantic segmentation* wird allen Punkten einer Szene, die ein Objekt desselben Typs repräsentieren, dieselbe Klasse zugewiesen. [Qi+17a]

die Punktwolke repräsentierten Form wiederherzustellen. Dies ist gerade der Ansatz der von Wang u. a. [Wan+19] eingeführten *EdgeConv*-Operation.

Einordnung

Für moderne Einsatzzwecke wie beispielsweise solche aus den Bereichen des autonomen Fahrens, der Robotik oder der Navigation in Innenräumen ist es hilfreich, in den Punktwolken nach semantischen Hinweisen zu suchen. Anders als geometrische Merkmale wie Kanten und Ecken lassen semantische Merkmale sich jedoch nicht gut mittels der Differentialgeometrie oder der algorithmischen Geometrie beschreiben. Daher ist man bei solchen Anforderungen auf die selbstständig von Daten lernenden Methoden des *deep learning* angewiesen, welche die gesuchten semantischen Hinweise ableiten von der statistischen Analyse des Datensatzes, der während des Trainingsprozesses des Modells herangezogen wurde. Wie bereits erwähnt können die irregulären Punktwolken in eine reguläre Repräsentation überführt werden, wodurch sie sich indirekt mittels klassischen euklidischen neuronalen Netzwerken verarbeiten lassen. Aufgrund der geschilderten, sich dabei ergebenden Probleme ist eine bessere, alternative Herangehensweise, zunächst anhand der Punktmenge einen Graphen zu konstruieren, mit dem sich Nachbarschaftsbeziehungen zwischen den Punkten modellieren lassen und daraufhin einen der in Kapitel 3 vorgestellten Ansätze zu verfolgen, um unter Verwendung eines auf nichteuklidische Gebiete verallgemeinerten Faltungsoperators den Graphen statt der Punktwolke zu verarbeiten.

5.2. Beachtenswerte Eigenschaften von Punktmengen

Eine Punktwolke ist eine Teilmenge von Punkten aus einem euklidischen Raum $\mathbb{R}^n$. Die drei Haupteigenschaften einer solchen Punktmenge sind folgende:

1. Die Punkte einer Punktwolke haben keine Ordnung. Man kann sie zwar sortieren, es existiert jedoch keine kanonische Reihenfolge. Das Einlesen der Punkte in einer alternativen Reihenfolge wäre ebenso richtig. Da eine Permutation der Punkte nicht deren räumliche Verteilung ändert, ist eine Punktwolke invariant unter Permutation.

2. Eine Punktwolke ist invariant unter gewissen Transformationen wie den euklidischen, also abstandserhaltenden oder auch isometrischen Transformationen, zu denen unter anderem Translationen und Rotationen gehören, da beispielsweise eine Translation der Punkte nicht deren räumliche Verteilung ändert. Nach einer solchen Transformation handelt es sich noch um dieselbe Punktwolke wie vor der Transformation, nur eben an eine andere Position verschoben.

3. Die Punkte stammen aus einem Raum mit einer Abstandsmetrik, die über die Norm durch das Skalarprodukt induziert ist. Hinsichtlich des Abstands existieren somit Beziehungen zwischen den Punkten. Beispielsweise bildet ein Punkt mit anderen Punkten, die einen geringen Abstand zu ihm haben, eine Nachbarschaft. Eine Nachbarschaft ist eine Teilmenge der Punktmenge, die eine gewisse Bedeutung trägt.

Damit ein neuronales Netzwerk korrekt mit Punktwolken umgehen kann, muss es diese drei eben genannten Eigenschaften von Punktwolken beachten:

1. Das neuronale Netzwerkmodell sollte die Permutationsinvarianz implementieren, damit es für die verschiedenen zulässigen Reihenfolgen, in der die Punkte der Punktwolke eingelesen werden können, in jedem Fall die korrekte Ausgabe für das Klassifikations- beziehungsweise Segmentierungsproblem liefert. Dies wird mit der in Abbildung 2.3 auf Seite 15 gezeigten Blaupause des *geometric deep learning* illustriert. Drei Strategien um ein neuronales Netzwerk permutationsinvariant zu gestalten werden in Abschnitt 5.3 angeführt.

2. Das neuronale Netzwerk sollte invariant unter euklidischen Transformationen wie etwa Translation sein. Die englischen Begriffe *translation invariance* und *shift invariance* sind Synonyme und wurden in Unterabschnitt 2.6.1 vorgestellt.
   - *PointNet* [Qi+17a], der Pionier der direkten Verarbeitung von Punktwolken, richtet dazu die Punktwolke an einem kanonischen Raum aus. Hierzu schätzt ein Netzwerk im Netzwerk namens *T-Net* eine affine Transformationsmatrix, mittels welcher die Eingabepunktwolke transformiert wird. Das *T-Net* in *PointNet* ist von dem *spatial transformer network* von Jaderberg u. a. [Jad+15] inspiriert.
   - *Dynamic Graph CNN* von Wang u. a. [Wan+19] ist zu einem gewissen Grad translationsinvariant, wie in Unterabschnitt 5.7.4 auf Seite 105 ausgeführt wird. Derjenige Teil $\boldsymbol{\phi}_m \cdot \mathbf{x}_i$ des in Unterabschnitt 5.7.2 näher vorgestellten *edge feature*, der abhängig ist von der Translation, kann optional deaktiviert werden, wodurch *Dynamic Graph CNN* komplett invariant unter Translation wird. Allerdings wird *Dynamic Graph CNN* dadurch dazu reduziert, ein Objekt nur noch anhand einer ungeordneten Menge von Bereichen (*patches*) erkennen zu können, wobei es die Positionen und Orientierungen der Bereiche komplett ignoriert. *Dynamic Graph CNN* muss somit einen Kompromiss zwischen Translationsinvarianz und

der Fähigkeit eingehen, zusätzlich zu der lokalen Geometrie der Bereiche (*patches*) auch die globalen Informationen über das Objekt lernen zu können. Im Gegensatz zu *PointNet* können dafür die lokalen geometrischen Strukturen erfasst werden, welches wie im Aufzählungspunkt 3 beschrieben dazu nicht in der Lage ist.

3. Das neuronale Netzwerkmodell sollte die Abstandsmetrik des zugrundeliegenden Raums berücksichtigen. Punkte der Punktwolke sollten somit nicht als isoliert voneinander behandelt werden. *PointNet* verarbeitet (als ersten Ansatz zur Gewährleistung von Permutationsinvarianz) die Punkte jedoch isoliert voneinander, weshalb *PointNet* wie auch dessen Weiterentwicklung *PointNet++* [Qi+17b] nicht fähig dazu ist, lokale geometrische Strukturen einzufangen. Dem in diesem Kapitel vorgestellten *Dynamic Graph CNN* hingegen gelingt dies.

5.3. Strategien zur Erreichung von Permutationsinvarianz

Aufgrund der nicht eindeutig bestimmten Einlesereihenfolge bei Punktwolken muss ein neuronales Netzwerkmodell Permutationsinvarianz implementieren, damit es für verschiedene Permutationen der Punktwolke korrekte Ausgaben liefern kann. Es sind drei Strategien bekannt um ein unter Permutation invariantes künstliches neuronales Netzwerkmodell zu entwerfen:

1. die Punkte der Punktwolke durch Sortieren in eine kanonische Reihenfolge bringen

2. die Punkte als sequentielles Signal interpretieren und mit zufällig permutierten Sequenzen ein *recurrent neural network* trainieren in der Hoffnung, dass das *recurrent neural network* invariant gegenüber der Eingabereihenfolge wird

3. eine symmetrische Funktion zur Aggregation der von jedem Punkt beigesteuerten Information verwenden, beispielsweise die Maximumsfunktion max

Diese drei Strategien werden von Qi u. a. [Qi+17a] näher erläutert. Qi u. a. erläutern auch, weshalb von diesen drei Strategien einzig die dritte – das Verwenden einer symmetrischen Aggregationsfunktion – erfolgversprechend ist.

5.4. Grundlegendes Prinzip

Eine der beiden Hauptideen von Wang u. a. [Wan+19] ist die Verwendung einer neu eingeführten, der klassischen Faltung ähnlichen Operation, die *edge convolution* oder kurz *EdgeConv* genannt wird. Diese Operation wird von Wang u. a. verwendet, um eine neue Netzwerkarchitektur namens *Dynamic Graph CNN* zu entwerfen.

Die als *EdgeConv* bezeichnete Operation erlaubt es im Gegensatz zu früheren Verfahren wie *PointNet* [Qi+17a] oder *PointNet++* [Qi+17b] neben den Informationen über die globale Geometrie der Punktwolke auch die lokalen geometrischen Strukturen der Punktwolke einzufangen. Das Einfangen der lokalen Geometrie erreichen Wang u. a., indem sie für

die Punktwolke zunächst einen Graphen konstruieren, welcher die Nachbarschaftsbeziehungen zwischen den einzelnen Punkten repräsentiert. Den Kanten, die in diesem Graph einen Punkt $\mathbf{x}_i$ mit seinen benachbarten Punkten verbinden, weist die *EdgeConv*-Operation sogenannte *edge features* zu – dabei handelt es sich um Skalare – um schließlich komponentenweise die *edge features* aller ausgehenden Kanten des Punktes $\mathbf{x}_i$ zu aggregieren und als Ausgabe der *EdgeConv*-Schicht einen neuen Vektor $\mathbf{x}'_i$ zu produzieren.

Zum Vergleich: Der Pionier für die direkte Verarbeitung von Punktwolken, *PointNet*, hatte die Punkte noch individuell und als isoliert voneinander betrachtet, um die Permutationsinvarianz zu gewährleisten, dadurch aber die Nachbarschaftsbeziehungen zwischen den Punkten gänzlich vernachlässigt. Eine Erweiterung von *PointNet* namens *PointNet++* ist zwar in der Lage, die Nachbarschaft eines Punktes einzubeziehen, indem es die Punktwolke anhand der Abstandsmetrik des zugrundeliegenden euklidischen Raums $\mathbb{R}^n$ partitioniert und auf die ineinander geschachtelten Partitionen der Punktwolke rekursiv *PointNet* anwendet. Auf lokaler Ebene behandelt *PointNet++* die Punkte jedoch weiterhin, wie schon *PointNet* zuvor, als isoliert voneinander, da *PointNet++* nicht wie *Dynamic Graph CNN* mit einem Nachbarschaftsgraphen arbeitet.

Indem *EdgeConv* auf jeden Punkt $\mathbf{x}$ der Punktwolke angewendet wird, wird für jeden Punkt $\mathbf{x}$ einer Punktwolke $\mathbf{X} \subseteq \mathbb{R}^F$ ein neuer Punkt $\mathbf{x}'$ berechnet, der nicht notwendigerweise dieselbe Dimension F wie $\mathbf{x}$ haben muss. In ihrer Gesamtheit bilden die Punkte $\mathbf{x}'$ eine neue Punktwolke $\mathbf{X}'$ mit Dimension F'. Somit wird mit jeder *EdgeConv*-Schicht eine neue Punktwolke produziert und in der folgenden *EdgeConv*-Schicht zu Beginn immer auch ein neuer Graph berechnet, der die neuen, sich im *feature space* geänderten Nachbarschaften repräsentiert. Anders ausgedrückt, die Nachbarn $\mathbf{x}_j$ eines Punktes $\mathbf{x}_i$ ändern sich von Schicht zu Schicht. Auf diese Weise vergrößert sich für zwei semantisch unterschiedliche Punkte deren Distanz im *feature space* immer weiter, wohingegen semantisch ähnliche Punkte eine im *feature space* geringe Entfernung zueinander erreichen, selbst wenn diese Punkte in der Eingabepunktwolke physisch weit voneinander entfernt sind. Dieses Prinzip, eine starke semantische Ähnlichkeit zweier Punkte über einen geringen Abstand voneinander im *feature space* auszudrücken, wird in Abbildung 5.2 auf der nächsten Seite veranschaulicht.

Nähe im *feature space* ist daher nicht gleichbedeutend mit Nähe in der Eingabe, was so interpretiert werden kann, als dass Information sich nichtlokal durch die Punktwolke hindurch ausbreiten kann. Dies ist beispielsweise für die Segmentierung einer Punktwolke in ihre semantisch gleichen Teilmengen von Nutzen. Die Punkte der Punktwolke werden also semantisch gruppiert, indem das Modell *Dynamic Graph CNN* zu Beginn jeder *EdgeConv*-Schicht den Nachbarschaftsgraphen neu konstruiert, anstatt wie frühere Verfahren vorzugehen und vor der Evaluation des Netzwerks einen Graphen zu konstruieren, der sich während der gesamten Auswertung nicht ändert. Neben der *EdgeConv* genannten Operation, welche die klassische Faltung auf Punktwolken verallgemeinert (und auch die von *PointNet* eingesetzte Operation verallgemeinert), ist eben dieses Aktualisieren des Nachbarschaftsgraphen von *EdgeConv*-Schicht zu *EdgeConv*-Schicht ein integraler Bestandteil der Arbeit von Wang u. a. [Wan+19].

Wang u. a. ließen sich für *Dynamic Graph CNN* einerseits von *PointNet* inspirieren, um mittels einer symmetrischen *aggregation operation* ihre *EdgeConv*-Operation für Punktwolken unter Wahrung der Permutationsinvarianz aufzustellen, andererseits aber auch

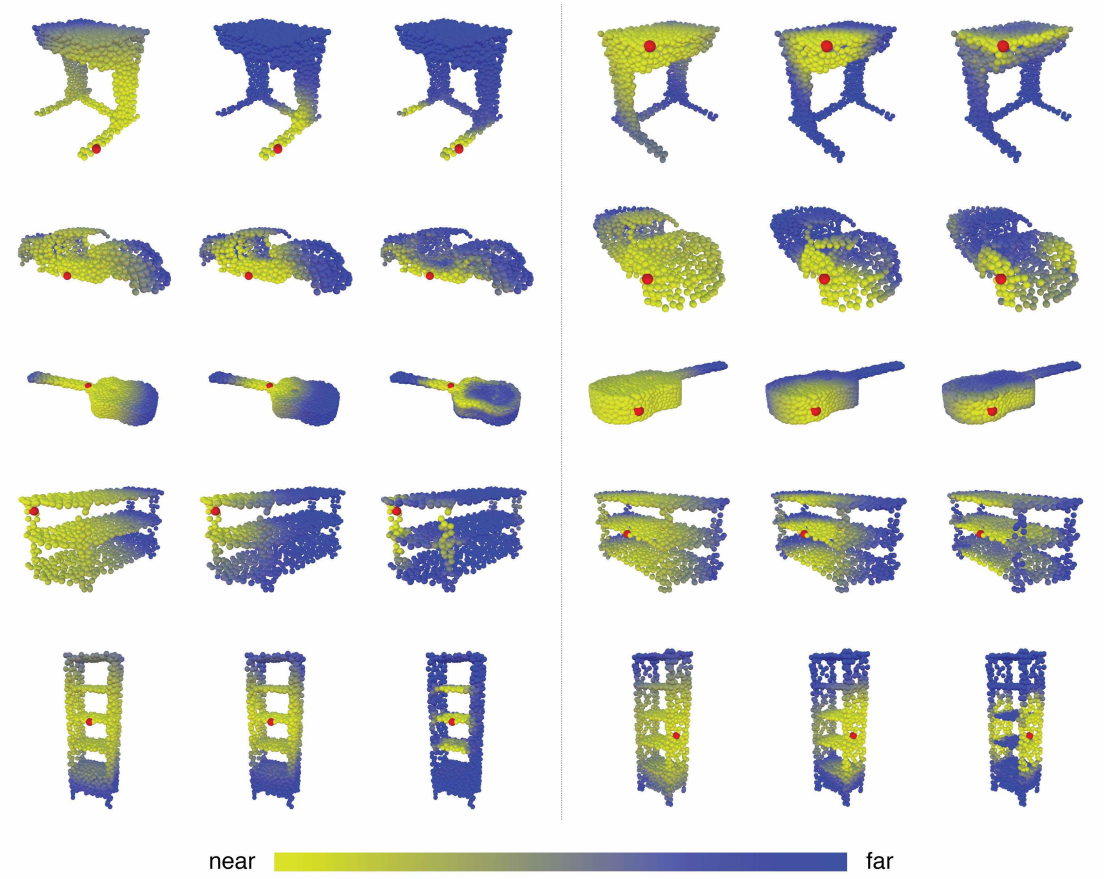


Abbildung 5.2.: **Darstellung der Struktur des *feature space*.** Dargestellt sind verschiedene beispielhafte Objekte, repräsentiert durch eine Punktwolke in jeweils drei verschiedenen Stadien. Das jeweils linkeste Bild veranschaulicht den euklidischen Abstand der Punkte einer Eingabepunktwolke zu dem roten Punkt. Das jeweils mittlere Bild veranschaulicht die Abstände nach der initialen Transformation der Punktwolke (siehe Abbildung 5.4 auf Seite 100). Das jeweils rechteste Bild veranschaulicht die Abstände der Punkte zu dem roten Punkt im *feature space* der letzten *EdgeConv*-Schicht. Im jeweils rechtesten Bild gelb gefärbt sind die Punkte mit semantisch ähnlicher Struktur statt wie zu Beginn räumlich nahe Punkte. [Wan+19]

von der Klasse der *graph convolutional neural networks* wie beispielsweise dem in Unterunterabschnitt 3.4.5.3 vorgestellten und auch schon in Unterabschnitt 4.9.2 zum Einsatz gekommenen *Mixture model network (MoNet)*, um mittels Nachbarschaftsgraphen die Nachbarschaftsbeziehungen zwischen den Punkten der Punktwolke zu berücksichtigen, wozu *PointNet* nicht in der Lage ist. Im Gegensatz zu früheren *graph convolutional neural networks* wie *Mixture model network (MoNet)*, die mit einem vor der Evaluation des Netzwerks konstruierten und sich während der gesamten Auswertung nicht ändernden Graphen arbeiten, aktualisiert *Dynamic Graph CNN* den Nachbarschaftsgraphen dynamisch, nachdem in der vorhergehenden *EdgeConv*-Schicht eine neue Punktwolke produziert wurde, was dieser Architektur ihren Namen verleiht. Abbildung A.3 auf Seite 127 verdeutlicht die Segmentierungsqualität von *PointNet* und *Dynamic Graph CNN* im Vergleich.

Zusammenfassend lässt sich sagen, dass mittels der neu eingeführten *EdgeConv*-Operation zu einer gegebenen Punktwolke die topologische Struktur (und damit die Vorstellung von Nähe) wiederhergestellt werden kann. Die *EdgeConv*-Operation kann als *neural network module* auch in bestehende Architekturen wie etwa die von *PointNet* eingebaut werden, um die Ergebnisqualität dieser Modelle zu verbessern.

5.5. k-nearest neighbor graph

Ein *nearest neighbor graph* für eine Punktwolke $\mathbf{X} \subseteq \mathbb{R}^n$ ist ein gerichteter Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in dem jeder Knoten $v_i \in \mathcal{V}$ nur mit denjenigen Knoten $v_j \in \mathcal{V}$ über eine gerichtete Kante verbunden ist, für die der Abstand zwischen v_i und v_j maximal so groß ist wie der Abstand zwischen v_i und einem beliebigen anderen Knoten in $\mathcal{V}$. In einem *nearest neighbor graph* hat jeder Knoten v_i somit nur eine ausgehende Kante, es sei denn es gibt neben v_j noch weitere Knoten mit gleichem Abstand zu v_i. Abbildung 5.3 auf der nächsten Seite zeigt ein Beispiel für einen *nearest neighbor graph*. Die Punktmenge $\mathbf{X}$ entspricht der Knotenmenge $\mathcal{V}$ des Graphen. Beachtenswert ist, dass die *nearest neighbor*-Relation nicht symmetrisch ist, denn auch wenn v_j der nächste Nachbar von v_i ist, kann v_j einen Nachbar haben, der näher an v_j liegt als v_i.

Bei einem *k-nearest neighbor graph* führt nicht nur dann eine gerichtete Kante von einem Knoten v_i zu einem anderen Knoten v_j, wenn für keinen weiteren Knoten dessen Distanz zu v_i kürzer ist als die zwischen v_i und v_j. In einem *k-nearest neighbor graph* ist ein Knoten v_i dann mit einem anderen Knoten v_j über eine gerichtete Kante verbunden, wenn der Abstand zwischen v_i und v_j zu den k kürzesten Distanzen zwischen v_i und einem beliebigen anderen Knoten $v \in \mathcal{V} \setminus \{v_i\}$ gehört.

5.6. Modellarchitektur

Die Architektur von *Dynamic Graph CNN (DGCNN)* unterscheidet sich je nach Anwendungsfall, also je nachdem ob eine Punktwolke klassifiziert oder segmentiert werden soll. Abbildung 5.4 auf Seite 100 zeigt eine schematische Darstellung der Architektur von *Dynamic Graph CNN*.

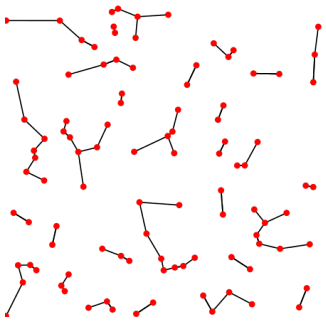


Abbildung 5.3.: **Beispiel für einen *nearest neighbor graph*.** Die Abbildung zeigt einen *nearest neighbor graph* für eine zweidimensionale Punktwolke $\mathbf{X} \subseteq \mathbb{R}^2$ bestehend aus 100 Punkten. [Epp11]

Die Eingabe der *EdgeConv*-Schicht ist ein $n \times f$-Tensor, der die n Punkte der Dimension f repräsentiert. Für diese Form der Punktwolke wird zunächst ein *k-nearest neighbor graph* konstruiert, wobei standardmäßig $k = 20$. Die *EdgeConv*-Operation ist über ein *multi-layer perceptron* der Form $\text{mlp}(a_1, a_2, \ldots, a_n)$ realisiert, wobei die Anzahl der a_i die Anzahl der *hidden layers* des *multi-layer perceptron* widerspiegelt. Für ein *multi-layer perceptron* der Form $\text{mlp}(64)$ und $k = 20$ würde ein $n \times 20 \times 64$-Tensor produziert, der mittels *pooling* zu einem $n \times 64$-Tensor umgewandelt wird.

Die Architektur für den Anwendungsfall der Klassifikation beinhaltet vier *EdgeConv*-Schichten, wovon die ersten drei jeweils über ein *multi-layer perceptron* der Form $\text{mlp}(64)$ und die vierte über $\text{mlp}(128)$ implementiert sind. Der Aufbau der Architektur ist in Abbildung 5.4 beschrieben. Für die Klassifikation wird zum Training des neuronalen Netzwerkmodells der Datensatz *ModelNet40* [Wu+15] verwendet, welcher $c = 40$ Klassen wie *Regal*, *Lampe*, *Zaun* oder *Baum* umfasst und aus 12.311 CAD-Modellen besteht.

Die Architektur für den Anwendungsfall der Segmentierung beinhaltet drei *EdgeConv*-Schichten, von denen die ersten beiden jeweils über ein *multi-layer perceptron* der Form $\text{mlp}(64, 64)$ und das dritte über $\text{mlp}(64)$ implementiert sind. Im Gegensatz zur Klassifikation erfolgt eingangs vor der ersten *EdgeConv*-Schicht zunächst eine Transformation der Punktwolke – die in Abbildung 5.2 auf Seite 97 erwähnte Transformation. Dabei wird die Eingabepunktwolke an einem kanonischen Raum ausgerichtet, indem sie mit einer geschätzten 3×3-Matrix multipliziert wird. Die Schätzung dieser 3×3-Matrix erfolgt mittels eines Tensors, der die Koordinaten eines jeden der n Punkte der Punktwolke konkateniert mit den Koordinatendifferenzen zwischen seiner k Nachbarn im *k-nearest neighbor graph*. Für die *part segmentation* wird der Datensatz *ShapeNetPart* [Yi+16] verwendet, welcher $p = 16$ Objektklassen wie *Gitarre*, *Auto*, *Stuhl* oder *Tasse* umfasst. Für die *semantic segmentation* kommt der Datensatz *Stanford 3D Indoor Scene Dataset (S3DIS)* [Arm+16] zum Einsatz, welcher $p = 13$ Objektklassen wie *Wand*, *Tür*, *Tisch* oder *Stuhl* überspannt.

5.7. Funktionsweise im Detail

In den folgenden Abschnitten wird der Ansatz von Wang u. a. [Wan+19] ausführlicher behandelt. Zunächst wird die Berechnung eines Nachbarschaftsgraphen beschrieben.

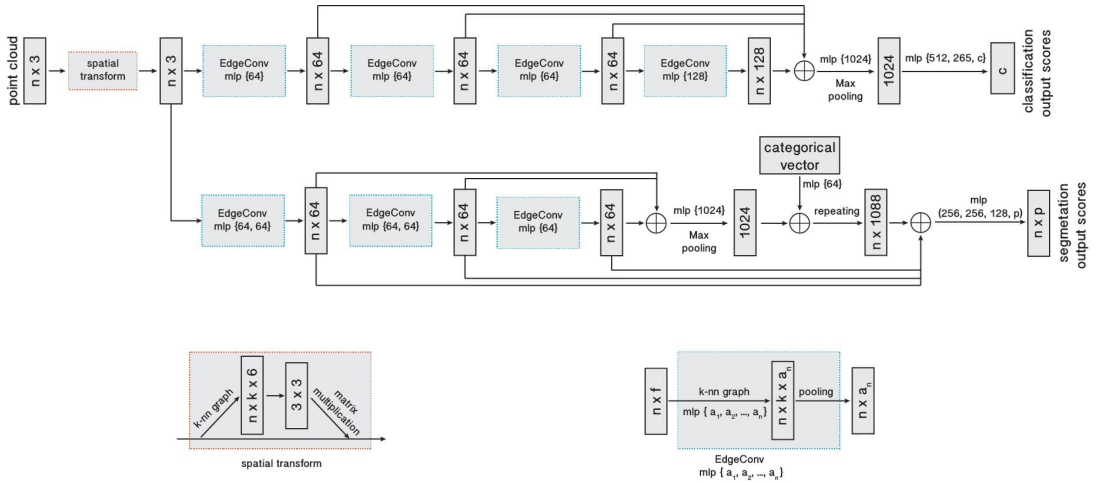


Abbildung 5.4.: **Schematische Darstellung der Architektur von *Dynamic Graph CNN*.** Für den Anwendungsfall der **Klassifikation** wird die (hier dreidimensionale) Punktwolke bestehend aus n Punkten direkt der ersten *EdgeConv*-Schicht übergeben und dem oberen Zweig der Darstellung gefolgt. Die Architektur besteht in diesem Fall aus vier *EdgeConv*-Schichten, die über *shortcut connections* verfügen, dank derer die Ausgaben der vier *EdgeConv*-Schichten von dem $\bigoplus$-Operator konkateniert werden können. Anschließend wird mittels eines *multi-layer perceptron* sowie *max pooling* ein 1024-dimensionaler *global feature*-Vektor produziert, der wiederum einem *multi-layer perceptron* der Form $\mathrm{mlp}(512, 256, c)$ übergeben wird. Die Ausgabe ist ein c-dimensionaler Vektor, der *classification scores* beschreibt, wobei die Zahl c der Anzahl der Klassen des Trainingsdatensatzes entspricht. Für den Anwendungsfall der **Segmentierung** wird die Punktwolke zunächst transformiert bevor sie der ersten *EdgeConv*-Schicht im unteren Zweig der Darstellung übergeben wird. Die Architektur besteht in diesem Fall aus nur drei *EdgeConv*-Schichten. Dafür wird für jeden der n Punkte mittels eines *multi-layer perceptron* aus einem *categorical vector* ein 64-dimensionaler Vektor produziert und mit dem 1024-dimensionalen *global feature*-Vektor konkateniert. Der entstandene $n \times 1088$-Tensor wird dank der *shortcut connections* mit den Ausgaben der drei *EdgeConv*-Schichten konkateniert und einem *multi-layer perceptron* der Form $\mathrm{mlp}(256, 256, 128, p)$ übergeben. Die Ausgabe ist ein $n \times p$-Tensor, der für jeden der n Punkte der Punktwolke die *per-point classification scores* für die p semantischen *labels* beschreibt. [Wan+19]

Daraufhin wird die *EdgeConv*-Operation eingeführt. Anschließend wird das dynamische Aktualisieren des Graphen erläutert.

Einleitend kann vorgegriffen werden, dass die *EdgeConv* genannte faltungsähnliche Operation nicht etwa Merkmale (*features*) der Punkte aus deren *embeddings* generiert. Stattdessen wird ein Graph erzeugt, dessen Kanten jeweils einen Punkt $\mathbf{x}_i$ mit einem anderen Punkt $\mathbf{x}_j$ verbinden. Für eine Kante (i, j) dieses Graphen wird ein *edge feature*-Vektor $\mathbf{e}_{ij}$ berechnet, welcher die Beziehung zwischen einem Punkt $\mathbf{x}_i$ und dem mit ihm verbundenen anderen Punkt $\mathbf{x}_j$ beschreibt. Die *edge feature*-Vektoren aller zu einem Punkt $\mathbf{x}_i$ gehörigen Kanten werden schließlich komponentenweise aggregiert, um für den Punkt $\mathbf{x}_i$ ein neues *embedding* $\mathbf{x}'_i$ zu produzieren.

5.7.1. Konstruktion des Nachbarschaftsgraphen

Gegeben sei eine Punktwolke $\mathbf{X}$ der Dimension F bestehend aus n Punkten, also $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^F$. Zunächst wird ein gerichteter Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ berechnet, mittels dessen die lokale Struktur der Punktwolke $\mathbf{X}$ repräsentiert wird. Dabei ist $\mathcal{V} = \{1, \ldots, n\}$ die Knotenmenge des Graphen und $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ dessen Kantenmenge. Im einfachsten Fall ist dieser Graph $\mathcal{G}$ ein *k-nearest neighbor graph* der Punktwolke $\mathbf{X}$, wie er in Abschnitt 5.5 vorgestellt wurde. Jedoch führt von jedem Knoten $v \in \mathcal{V}$ auch eine gerichtete Kante zu sich selbst (*self-loop*). Wang u. a. berechnen den *k-nearest neighbor graph* mittels einer Distanzmatrix, welche die paarweisen Abstände der Punkte im *feature space* zueinander repräsentiert und aus der sie für jeden Punkt die Indizes derjenigen $k = 20$ Punkte mit der geringsten Entfernung zu ihm ablesen (`https://github.com/WangYueFt/dgcnn`).

5.7.2. Einführung einer *edge convolution* Operation

Die *edge convolution* Operation, kurz *EdgeConv*, ist der hauptsächliche wissenschaftliche Beitrag von Wang u. a. [Wan+19]. Dank ihr können im Gegensatz zu früheren Verfahren wie etwa *PointNet* [Qi+17a] auch die lokalen geometrischen Strukturen erfasst werden. In diesem Abschnitt wird im Detail auf die *EdgeConv*-Operation eingegangen.

5.7.2.1. Vorüberlegungen

Zu einer Punktwolke $\mathbf{X} \in \mathbb{R}^F$ sei ein Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ gegeben wie er in Unterabschnitt 5.7.1 beschrieben ist. Sei außerdem $h_{\mathbf{\Theta}} : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}^M$ eine nichtlineare Funktion (*edge function*), zu der eine $F \times M$-Matrix $\mathbf{\Theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M)$ gehört, deren F-dimensionale Spaltenvektoren jeweils die lernbaren Parameter eines der M verschiedenen Filter repräsentieren.

Die *edge function* $h_{\mathbf{\Theta}}$ bildet ein Paar von F-dimensionalen Vektoren $\mathbf{x}_i, \mathbf{x}_j$ ab auf einen dritten, M-dimensionalen Vektor $\mathbf{e}_{ij}$, der nicht notwendigerweise dieselbe Dimension wie die beiden Eingabevektoren haben muss. Mittels dieser *edge function* $h_{\mathbf{\Theta}}$ werden für einen Knoten $\mathbf{x}_i$ des Graphen und einen durch eine gerichtete Kante mit $\mathbf{x}_i$ verbundenen Knoten $\mathbf{x}_j$ für die M verschiedenen Filter die von Wang u. a. neu eingeführten *edge features* $e_{ij1}, \ldots, e_{ijM}$ beziehungsweise der aus diesen *edge features* zusammengesetzte *edge feature*-Vektor $\mathbf{e}_{ij} = h_{\mathbf{\Theta}}(\mathbf{x}_i, \mathbf{x}_j)$ realisiert.

Nach Berechnung der mit den Kanten assoziierten *edge feature*-Vektoren werden für die zu einem Knoten $\mathbf{x}_i$ gehörigen Kanten (i, j) die *edge feature*-Vektoren $\mathbf{e}_{ij}$ komponentenweise aggregiert, um den F-dimensionalen Knoten $\mathbf{x}_i$ zu aktualisieren, wobei der M-dimensionale Knoten $\mathbf{x}'_i$ erzeugt wird. Dies wird in Abbildung 5.5 veranschaulicht. Dazu wird für alle Kanten, die von einem Knoten $\mathbf{x}_i$ ausgehen, auf die zu den Kanten gehörigen M-dimensionalen *edge feature*-Vektoren $\mathbf{e}_{ij}$ komponentenweise eine symmetrische *aggregation operation* $\square$ angewendet. Die *EdgeConv*-Operation kann man sich somit wie folgt vorstellen:

$$\mathbf{x}'_i = \square_{(i,j)\in\mathcal{E}} h_\Theta\left(\mathbf{x}_i, \mathbf{x}_j\right) = \square_{(i,j)\in\mathcal{E}} \mathbf{e}_{ij} = \left(\square_{(i,j)\in\mathcal{E}} e_{ij1}, \ldots, \square_{(i,j)\in\mathcal{E}} e_{ijM}\right)^\mathsf{T} \tag{5.1}$$

Die m-te Komponente des Vektors $\mathbf{e}_{ij}$ repräsentiert dabei das Ergebnis e_{ijm} der Funktion h_Θ für den m-ten der M Filter $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M$ (mehr dazu in Unterunterabschnitt 5.7.2.2). Diese Komponenten $e_{ij1}, \ldots, e_{ijM}$ sind die M *edge features* einer Kante (i, j).

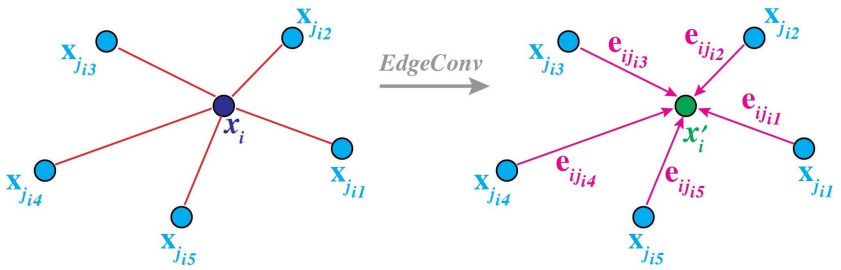


Abbildung 5.5.: **Illustration der *EdgeConv*-Operation.** Für die unterschiedlichen Kanten (i, j), die einen Knoten $\mathbf{x}_i$ mit seinen verschiedenen Nachbarknoten $\mathbf{x}_j$ verbinden, werden zunächst *edge feature*-Vektoren $\mathbf{e}_{ij}$ produziert, die schließlich komponentenweise aggregiert werden, um den F-dimensionalen Knoten $\mathbf{x}_i$ zu dem M-dimensionalen Knoten $\mathbf{x}'_i$ abzuändern. [Wan+19]

Bei der Operation $\square$ kann es sich beispielsweise um die Summation $\sum$ oder die Maximumsfunktion max handeln. Es muss sich jedoch um eine symmetrische Funktion handeln, um wie in Abschnitt 5.3 beschrieben die Permutationsinvarianz des Netzwerkmodells zu gewährleisten und die korrekte Arbeitsweise des Verfahrens auch für Permutationen der Punktwolke sicherzustellen.

Die *EdgeConv*-Operation überführt auf diese Weise eine F-dimensionale Punktwolke in eine M-dimensionale Punktwolke, wobei die beiden Punktwolken aus derselben Anzahl n von Punkten bestehen, aber nicht notwendigerweise $F = M$ gelten muss. Die Wahl von h_Θ und $\square$ haben offensichtlich entscheidenden Einfluss auf die Eigenschaften der als *edge convolution* bezeichneten Operation. Im folgenden Abschnitt wird untersucht wie die nichtlineare *edge function* h_Θ und die symmetrische *aggregation operation* $\square$ definiert sind.

5.7.2.2. Definition der *edge convolution* Operation

Wang u. a. [Wan+19] wählen für die symmetrische *aggregation operation* $\square$ die Maximumsfunktion max. Dabei orientieren sie sich an *PointNet* und den von Qi u. a. [Qi+17a]

durchgeführten Experimenten, welche gezeigt haben, dass das Modell *PointNet* bei einer Auswahl von *max pooling*, *average pooling* und einer *attention*-basierten gewichteten Summe die besten Ergebnisse unter Verwendung von *max pooling* erzielt.

Für die *EdgeConv*-Operation in *Dynamic Graph CNN* werden statt M Filter wie in Unterunterabschnitt 5.7.2.1 nun $2M$ verschiedene Filter eingesetzt. Dabei handelt es sich um M Filterpaare, welche jeweils aus einem Filter $\boldsymbol{\theta}_m$ für die lokalen Strukturen und einem Filter $\boldsymbol{\phi}_m$ für die globalen Strukturen bestehen. Die Vektoren $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M$ enkodieren somit die lokalen Parameter, die Vektoren $\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_M$ die globalen Parameter. Zusammengefasst enkodiert die $F \times 2M$-Matrix $\boldsymbol{\Theta} = \left(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M, \boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_M\right)$ die Gewichte der $2M$ Filter. Da diese $2M$ Spaltenvektoren alle Dimension F haben, erfordert eine *EdgeConv*-Schicht insgesamt $2MF$ Parameter.

Bei M aus zwei Vektoren $\boldsymbol{\theta}_m, \boldsymbol{\phi}_m$ zusammengesetzten Paaren ergeben sich je Kante $(i, j) \in \mathcal{E}$ also M *edge features* e_{ijm}, wobei $m = 1, \ldots, M$. Diese Skalare e_{ijm} bilden die M Komponenten des M-dimensionalen Vektors $\mathbf{e}_{ij}$. Die *edge function* $h_{\boldsymbol{\Theta}}(\mathbf{x}_i, \mathbf{x}_j)$, über die ein *edge feature*-Vektor $\mathbf{e}_{ij}$ gegeben ist, definieren Wang u. a. über

$$e_{ijm} = \mathrm{ReLU}\left(\boldsymbol{\theta}_m \cdot \left(\mathbf{x}_j - \mathbf{x}_i\right) + \boldsymbol{\phi}_m \cdot \mathbf{x}_i\right). \tag{5.2}$$

Mit e_{ijm} wird das *edge feature* der Kante (i, j) bei Anwendung des m-ten Filterpaars bezeichnet. Wird nur der Knoten $\mathbf{x}_i$ betrachtet, liefert das eine globale Information. Wird hingegen die Differenz zwischen $\mathbf{x}_i$ und einem Nachbarknoten $\mathbf{x}_j$ berücksichtigt, so liefert das eine lokale Information. Die als *Rectified Linear Unit (ReLU)* bezeichnete Nichtlinearität $f(x) = \max(0, x)$, die in künstlichen neuronalen Netzwerken als Aktivierungsfunktion zum Einsatz kommt, dient dazu, die *edge function* $h_{\boldsymbol{\Theta}}$ nichtlinear zu gestalten.

Die nichtlineare *edge function* $h_{\boldsymbol{\Theta}}$ zur Ermittlung der *edge feature*-Vektoren $\mathbf{e}_{ij}$ kann mittels eines *shared multi-layer perceptron* implementiert werden. Der Zusammenhang zwischen den Neuronen eines *multi-layer perceptron* und den eingesetzten M Filterpaaren ist in Abbildung 5.4 auf Seite 100 ersichtlich.

In Gleichung 5.2 fängt der Knoten $\mathbf{x}_i$ im Zentrum einer Nachbarschaft (*patch*) die globale Struktur der Form ein, wohingegen die Differenzen $\mathbf{x}_j - \mathbf{x}_i$ die lokale Nachbarschaftsinformation enkodieren. Über die Differenzen $\mathbf{x}_j - \mathbf{x}_i$ wird die geometrische Repräsentation der Form wie eine Ansammlung vieler *patches* behandelt, deren Positionen und Orientierungen – die globalen Informationen – durch die einzelnen Knoten $\mathbf{x}_i$ erfasst werden.

In Kombination mit der Maximumsfunktion als symmetrische *aggregation operation* ist die Ausgabe der *EdgeConv*-Operation für einen Knoten $\mathbf{x}_i$ somit

$$x'_{im} = \max_{(i,j)\in\mathcal{E}} e_{ijm}$$

beziehungsweise

$$\mathbf{x}'_i = \max_{(i,j)\in\mathcal{E}} \mathbf{e}_{ij}$$

mit $\mathbf{x}'_i = \left(x'_{i1}, \ldots, x'_{iM}\right)^\mathsf{T}$ und $\mathbf{e}_{ij} = \left(e_{ij1}, \ldots, e_{ijM}\right)^\mathsf{T}$. Die Dimension M der Ausgabe entspricht somit der Anzahl M der verwendeten Filterpaare.

Die *EdgeConv*-Operation ist differenzierbar, kann also mittels gradientenbasierten Optimierungsverfahren trainiert werden. Sie lässt sich auch in existierende Architekturen einbauen.

5.7.2.3. Vergleich mit der klassischen Faltung

Gleichung 5.1 ähnelt der Definition der klassischen diskreten Faltung. Tatsächlich ist die *edge convolution* eine Verallgemeinerung der klassischen Faltung. Dies ist nicht überraschend, denn die Verallgemeinerung der klassischen Faltung ist gerade eines der Ziele des *geometric deep learning*. Auf die Ähnlichkeit der beiden Definitionen wird in diesem Abschnitt näher eingegangen.

Den Knoten $\mathbf{x}_i$ kann man sich als zentralen Pixel einer Nachbarschaft in einem Bild vorstellen, wobei die Menge $\{\mathbf{x}_j : (i,j) \in \mathcal{E}\}$ der Knoten $\mathbf{x}_j$, die durch eine Kante mit $\mathbf{x}_i$ verbunden sind, die Nachbarschaft von $\mathbf{x}_i$ darstellen. Der Knoten $\mathbf{x}_i$ ist wie eingangs erwähnt durch eine gerichtete Kante (i,i) mit sich selbst verbunden. Die klassische Faltung erhält man aus der *EdgeConv*-Operation, wenn man als *aggregation operation* $\square$ die Summation wählt und als *edge function* h_Θ das Standardskalarprodukt $\boldsymbol{\theta}_m \cdot \mathbf{x}_j$ aus den Nachbarn von $\mathbf{x}_i$ und einem der M Filtervektoren $\boldsymbol{\theta}_m$, der die globalen Parameter enkodiert. Für den m-ten der M Filter liefert die klassische Faltung

$$x'_{im} = \sum_{(i,j)\in\mathcal{E}} \boldsymbol{\theta}_m \cdot \mathbf{x}_j$$

somit die m-te Komponente des aktualisierten *embedding* $\mathbf{x}'$, wobei $m = 1, \ldots, M$. Bei $\boldsymbol{\theta}_m$ und $\mathbf{x}_j$ handelt es sich um zwei Vektoren derselben Dimension F, wobei F die Dimension der Eingabepunktwolke der aktuellen Schicht ist.

5.7.2.4. Vergleich mit *PointNet*

Auch *PointNet* kann insofern als Spezialfall der *EdgeConv* aufgefasst werden, als dass die *edge function* h_Θ lediglich die Nachbarn eines Knotens $\mathbf{x}_i$ nicht zu berücksichtigen braucht. Für eine zu wählende *edge function* $h_\Theta(\mathbf{x}_i, \mathbf{x}_j)$ mit irrelevantem Argument $\mathbf{x}_j$ werden die Punkte $\mathbf{x}_i$ individuell und als isoliert von ihren Nachbarn $\mathbf{x}_j$ betrachtet wie es bei *PointNet* der Fall ist. Dies wird durch

$$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = h_\Theta(\mathbf{x}_i)$$

beschrieben. Bei einer solchen Vernachlässigung der Abstandsmetrik des euklidischen Raums $\mathbb{R}^n$ werden jedoch keine lokalen geometrischen Strukturen eingefangen.

PointNet kann weiterhin insofern als Spezialfall von *Dynamic Graph CNN* aufgefasst werden, als dass $k = 1$ bei der Berechnung des *k-nearest neighbor graph* gesetzt wird. *PointNet* ist somit ein *Dynamic Graph CNN*, für das ein Nachbarschaftsgraph mit leerer Kantenmenge $\mathcal{E}$ berechnet wird.

5.7.3. Dynamische Aktualisierung des Nachbarschaftsgraphen

Der in der ersten Schicht des Netzwerks berechnete Graph $\mathcal{G}$ aus Unterabschnitt 5.7.1 wird nicht über alle *EdgeConv*-Schichten des Modells hinweg weitergenutzt. Stattdessen wird er in jeder *EdgeConv*-Schicht der in Abbildung 5.4 auf Seite 100 dargestellten Netzwerkarchitektur zunächst aktualisiert bevor er für die weiteren Berechnungen genutzt wird. Gleichbedeutend damit ist, dass die Menge der *k-nearest neighbors* eines Punktes der Punktwolke sich von *EdgeConv*-Schicht zu *EdgeConv*-Schicht ändert. Dies ist ein großer

Unterschied zu früheren *graph convolutional neural networks*, die mit unveränderlichen Graphen arbeiten.

Auf diese Weise können Punkte sowohl im euklidischen Raum als auch im semantischen *feature space* gruppiert werden, was der Segmentierung dienlich ist. Auf diese Weise können auch von zwei Punkten, die sich in der Eingabepunktwolke räumlich weit voneinander entfernt befinden, die semantischen Charakteristika der Punktwolke aufgegriffen werden, und zwar durch Nähe dieser zwei Punkte im *feature space*. Was dies bedeutet wird in Abbildung 5.2 auf Seite 97 dargestellt.

Auch kann sich der Wert k von Schicht zu Schicht unterscheiden und wird daher in der l-ten Schicht mit $k^{(l)}$ angesprochen. Die Notation $\mathcal{G}^{(l)} = \left(\mathcal{V}^{(l)}, \mathcal{E}^{(l)}\right)$ bezeichnet den Graphen der l-ten Schicht. Die Kanten $(i, j_1), \ldots, (i, j_{k^{(l)}})$ im Graph der l-ten Schicht werden so erzeugt und der Kantenmenge $\mathcal{E}^{(l)}$ hinzugefügt, dass die Punkte $\mathbf{x}_{j_1}^{(l)}, \ldots, \mathbf{x}_{j_{k^{(l)}}}^{(l)}$ die $k^{(l)}$ nächsten Punkte zum Punkt $\mathbf{x}_i^{(l)}$ sind. Damit ist gemeint, dass wie in Abbildung 5.5 dargestellt ein Knoten $\mathbf{x}_i$ mit seinen k nächsten Nachbarn über Kanten verbunden wird. Wie dieser Graph $\mathcal{G}^{(l)}$ jeweils zu berechnen ist, wird von dem Netzwerkmodell während des Trainingsprozesses gelernt. Die Menge der k nächsten Nachbarn wird dabei aus der Sequenz der *embeddings* der Punkte berechnet.

Durch dieses dynamische Aktualisieren des Nachbarschaftsgraphen wird bei den späteren Berechnungen somit nicht mehr der Eingabegraph herangezogen. Anders ausgedrückt ist *Dynamic Graph CNN* das erste Netzwerkmodell, das den *computational graph* vom Eingabegraphen (*input graph*) entkoppelt. Diese Form des Lernens mit einem gesonderten *computational graph* wird als *latent graph learning* bezeichnet.

5.7.4. Translationsinvarianz der *edge convolution*

In diesem Abschnitt wird die bereits in Abschnitt 5.2 thematisierte Translationsinvarianz der *EdgeConv*-Operation näher betrachtet. Dazu wird angenommen, dass die gesamte Punktwolke und somit auch zwei Punkte $\mathbf{x}_i$ und $\mathbf{x}_j$ um $\mathbf{T}$ verschoben werden. Für die m-te Komponente des *edge feature*-Vektors $\mathbf{e}_{ij}$ gilt dann nach der Translation

$$\begin{aligned} e_{ijm} &= \mathrm{ReLU}\Big(\boldsymbol{\theta}_m \cdot \big((\mathbf{x}_j + \mathbf{T}) - (\mathbf{x}_i + \mathbf{T})\big) + \boldsymbol{\phi}_m \cdot (\mathbf{x}_i + \mathbf{T})\Big) \\ &= \mathrm{ReLU}\big(\boldsymbol{\theta}_m \cdot \big(\mathbf{x}_j - \mathbf{x}_i\big) + \boldsymbol{\phi}_m \cdot (\mathbf{x}_i + \mathbf{T})\big). \end{aligned}$$

Da der Abstand zwischen zwei Punkten, sofern beide um dasselbe $\mathbf{T}$ verschoben werden, unberührt bleibt, beeinflusst eine Translation lediglich die globalen Strukturen. Damit die *EdgeConv*-Operation invariant unter Translation gemäß Definition 2.6.1 auf Seite 13 wird, müsste $\boldsymbol{\phi}_m \cdot \mathbf{x}_i = \boldsymbol{\phi}_m \cdot (\mathbf{x}_i + \mathbf{T})$ gelten. Um wahrhaft translationsinvariant zu sein, müsste die *EdgeConv*-Operation somit durch Setzen von $\boldsymbol{\phi}_m = \mathbf{0}$ die globalen Strukturen ignorieren, was optional gemacht werden kann. Es muss also eine Entscheidung zwischen Translationsinvarianz und dem Lernen der globalen Struktur getroffen werden.

6. Rekonstruktion von dreidimensionalen Handmodellen aus RGB-Bildern

In diesem Abschnitt wird als möglicher Einsatzzweck für *geometric deep learning* die Rekonstruktion von dreidimensionalen Handmodellen aus RGB-Bildern behandelt. Dazu wird eine von Kulon u. a. [Kul+20] eingesetzte Verallgemeinerung der klassischen Faltung vorgestellt, welche auf Polygonnetzen mit fest vorgegebener Topologie arbeitet. Das Netzwerkmodell von Kulon u. a. ist interessant, weil es sich bei Polygonnetzen mit fest vorgegebener Topologie um einen Spezialfall handelt, der die Definition einer dem kartografierenden Ansatz folgenden verallgemeinerten Faltung erheblich vereinfacht. Während die Implementierung der Permutationsinvarianz insbesondere in Kapitel 5 noch von zentraler Wichtigkeit war, ist die Invarianz unter Permutation in diesem Spezialfall gänzlich irrelevant.

6.1. Problembeschreibung

Die Rekonstruktion dreidimensionaler Modelle von Händen aus RGB-Bildern und somit auch aus Videos findet beispielsweise im Bereich der virtuellen Realität (*virtual reality* oder auch *augmented reality*) und der Mensch-Computer-Interaktion Anwendung, etwa um in Videomaterial eine Person oder wenigstens ihre Hände durch einen virtuellen Avatar ersetzen zu können, ähnlich dazu wie etwa Apple es mit der firmeneigenen TrueDepth-Technologie ermöglicht, das eigene Gesicht durch ein sogenanntes Memoji zu ersetzen. Die Rekonstruktion dreidimensionaler Handmodelle ist aber auch in der Erkennung von Gebärdensprache von großer Bedeutung, was insbesondere nicht hörenden und schwerhörigen Menschen den Alltag vereinfachen soll.

Die Gestik und Artikulation mit den Händen gehört ebenso zur menschlichen Kommunikation wie die Körpersprache. Bislang werden Handmodelle wie *MANO* [RTB17] jedoch getrennt vom restlichen Körper, etwa *SMPL* [Lop+15], untersucht. Zudem existieren für die Aufgaben der Schätzung der Pose einer abgebildeten Hand oder deren dreidimensionale Rekonstruktion anhand eines Bildes weitestgehend nur solche Datensätze, die in einer streng kontrollierten Laborumgebung in bestimmten Kamerawinkeln vor einem *green screen* aufgenommen wurden. Dementsprechend können auf solchen Datensätzen trainierte neuronale Netzwerke noch nicht gut mit real vorkommenden Situationen umgehen.

Zudem ist die von Menschen durchgeführte manuelle Annotation der aufgenommenen Bilder mit dreidimensionalen Polygonnetzen zum Zwecke der Erstellung eines Trainingsdatensatzes für künstliche neuronale Netzwerke mühselig und zeit- als auch kostenintensiv. Ohne geeignete Trainingsdatensätze, die aus mit Polygonnetzen annotierten, nicht in einer Laborumgebung aufgenommenen Bildern von Händen bestehen, kann jedoch nicht die Ent-

wicklung neuronaler Netzwerkmodelle zur Rekonstruktion von Händen aus RGB-Bildern vorangetrieben werden.

Kulon u. a. [Kul+20] stellen daher ein System vor, welches automatisiert einen für das Training künstlicher neuronaler Netzwerke geeigneten, präzise annotierten und wirklichkeitsnahen Datensatz erstellen kann. Zudem präsentieren Kulon u. a. ein auf diesem Datensatz trainiertes Modell, welches die Ergebnisse der vorherigen Verfahren auf dem aktuellen Stand der Technik übertrifft.

Einordnung

Für Polygonnetze ergeben sich neben den allgemeinen Schwierigkeiten nichteuklidischer Verfahren noch weitere Hürden. Ein Polygonnetz (*mesh*) ist eine Diskretisierung einer Mannigfaltigkeit und kann durch einen als *meshing* bezeichneten Vorgang aus der kontinuierlichen Oberfläche einer Mannigfaltigkeit konstruiert werden.

Die verallgemeinerte Faltung sollte idealerweise unempfindlich gegenüber der Weise sein, in welcher die Mannigfaltigkeit diskretisiert wurde. Der definierte *patch operator* sollte also unabhängig von der einem Polygonnetz zugrundeliegenden Graphtopologie sein. Dazu wird im Allgemeinen ein lokales Koordinatensystem konstruiert. Sollte die Topologie des Polygonnetzes jedoch fest vorgegeben sein, so sind diese Schwierigkeiten irrelevant. In solch einem speziellen Fall ist es nicht nötig, ein lokales Koordinatensystem zu konstruieren. Stattdessen braucht, da die Topologie sich nicht ändert, lediglich eine lokale Reihenfolge für die Knoten einer Nachbarschaft bestimmt zu werden.

Obwohl es grundsätzlich aufgrund der nicht existenten kanonischen Reihenfolge und damit mehrdeutigen Reihenfolge problematisch ist, sich auf eine konkrete Reihenfolge festzulegen und demnach sämtliche $n!$ möglichen Reihenfolgen bei n Punkten beachtet werden sollten (Permutationsinvarianz), spielt die Permutationsinvarianz im Fall einer fest vorgegebenen Topologie keine Rolle.

6.2. Wissenschaftlicher Beitrag

Kulon u. a. [Kul+20] leisten zwei hauptsächliche wissenschaftliche Beiträge. Einerseits stellen sie ein neues künstliches neuronales Netzwerkmodell zur Rekonstruktion von dreidimensionalen Handmodellen aus RGB-Bildern vor, welches die Ergebnisse vergleichbarer bestehender Modelle auf dem aktuellen Stand der Technik übertrifft.

Andererseits stellen sie einen neuen Datensatz vor oder vielmehr ein System zur automatisierten Erzeugung und algorithmischen Annotation eines Datensatzes, der anders als die meisten existierenden Datensätze nicht aus Bildern, die in einer streng kontrollierten Laborumgebung aufgenommen wurden, besteht, sondern aus Bildern, die aus realitätsnahen Videos von in Gebärdensprache kommunizierenden Menschen extrahiert wurden.

Auf diesem neu vorgestellten Datensatz können sowohl das Netzwerkmodell von Kulon u. a. als auch andere Modelle trainiert werden. Das Lernen eines künstlichen neuronalen Netzwerks auf einem algorithmisch annotierten Datensatz nennt man *weakly-supervised learning*, da hierbei das Training nicht unter der „Aufsicht“ von durch Menschen angelegten Annotationen der Trainingsdaten erfolgt.

Jeder dieser beiden Beiträge verbessert für sich genommen bereits die Ergebnisse vergleichbarer Verfahren auf dem aktuellen Stand der Technik. Das eigene Modell trainiert auf dem eigenen Datensatz übertrifft die Ergebnisse existierender Modelle, die auf einem bestehenden Datensatz trainiert wurden. Das Training eines fremden Netzwerkmodells auf dem eigenen Datensatz verbessert dessen Leistung im Vergleich zu dessen Training auf einem bestehenden fremden Datensatz. Die Kombination dieser beiden Beiträge schlägt auch die fremden Modelle, die auf dem eigenen neuen Datensatz trainiert wurden.

Das im Rahmen dieses Kapitels behandelte Verfahren von Kulon u. a. [Kul+20] ist eine Verbesserung und Vereinfachung der Netzwerkarchitektur der früheren Arbeit [Kul+19] des federführenden Autors.

In diesem Kapitel wird lediglich das für die Verallgemeinerung der Faltung im Zusammenhang mit *geometric deep learning* relevante Netzwerkmodell behandelt, nicht das System zur Erzeugung und algorithmischen Annotation eines Datensatzes.

6.3. Grundlegendes Prinzip

Mangels einer globalen Parametrisierung existiert im Fall von nichteuklidischen Gebieten kein System von globalen Koordinaten, weshalb auch keine Ordnung der Knoten und somit auch keine kanonische Reihenfolge existiert. Der Ansatz der kartografierenden Methoden zur Lösung dieses Problem ist im Allgemeinen, ein System von lokalen Koordinaten $\mathbf{u}(x, y)$ um einen Punkt x zu konstruieren. Die Anwendung lokaler Gewichtungsfunktionen w_ℓ auf die in der Nachbarschaft $\mathcal{N}(x)$ von x liegenden Knoten y, wobei $\ell = 1, \dots, L$, führt zu einer durch

$$(f * g)_x = \sum_{\ell=1}^{L} g_\ell \sum_{y \in \mathcal{N}(x)} w_\ell\big(\mathbf{u}(x, y)\big) f(y)$$

definierten verallgemeinerten Faltung. Aufgrund der fehlenden kanonischen Reihenfolge kann wie in Unterabschnitt 3.4.4 ausgeführt keine Eins-zu-eins-Zuordnung zwischen den Knoten einer Nachbarschaft und den Gewichtsmatrizen umgesetzt werden, weshalb die Zuordnung mittels *soft-attention*-Gewichten $w_\ell(\mathbf{u}(x, y))$ gelernt wird. Mit

$$\sum_{y \in \mathcal{N}(x)} w_\ell(\mathbf{u}(x, y)) f(y) \tag{6.1}$$

werden im allgemeinen Fall L verschiedene sogenannte *soft pixels* produziert, welche mit dem entsprechenden Filtergewicht g_ℓ multipliziert werden.

Für den Fall, dass die Topologie des Polygonnetzes jedoch fest vorgegeben sein sollte, sich also nicht ändert, vereinfacht diese Definition sich deutlich. Lim u. a. [Lim+19] und Bouritsas u. a. [Bou+19] haben bemerkt, dass im Fall einer sich nicht ändernden Topologie die fehlende kanonische Reihenfolge irrelevant ist, weshalb es in diesem Spezialfall auch nicht nötig ist, ein System von lokalen Pseudo-Koordinaten zu entwerfen, die möglicherweise suboptimal bestimmt sind. Da es sich bei Polygonnetzen um diskretisierte Mannigfaltigkeiten handelt, sollte ein *patch operator* grundsätzlich auch unempfindlich gegenüber der Diskretisierungsweise sein, also von der dem Polygonnetze zugrundeliegenden Graphtopologie unabhängig sein. Man spricht im Englischen auch von der Forderung

nach *insensitivity to meshing*. Auch diese Anforderung ist im Fall eines Polygonnetzes mit fest vorgegebener Topologie irrelevant.

Der kartografierende Ansatz vereinfacht sich in diesem Spezialfall somit dahingehend, dass lediglich die Knoten einer Nachbarschaft in eine lokale Reihenfolge geordnet werden müssen, was Lim u. a. [Lim+19] auf die Idee für die in Abbildung 6.1 illustrierte Vorgehensweise ihres *spiral patch operator S* brachte.

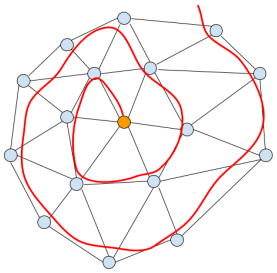


Abbildung 6.1.: **Illustration der spiralförmigen reihenweisen Anordnung der Knoten einer Nachbarschaft.** Lim u. a. [Lim+19] stellen für Polygonnetze mit fest vorgegebener Topologie eine intuitive Verfahrensweise zur Encodierung der Eingabe vor, die kein Resampling der Eingabemerkmale (*input features*) erfordert. Den *spiral operator* von Lim u. a. nutzen Bouritsas u. a. [Bou+19] als *patch operator* zur Definition einer *spiral convolution*, auf welche Kulon u. a. [Kul+20] zurückgreifen. [Bou+19]

Wenn die lokale Reihenfolge der Knoten feststeht, kann eine Funktion f unmittelbar auf die Nachbarknoten x_ℓ einer Nachbarschaft angewendet werden, womit Gleichung 6.1 sich vereinfacht zu

$$(f * g)_x = \sum_{\ell=1}^{L} g_\ell \, f(x_\ell).$$

Die Anwendung der Funktion f liefert analog zur Pixelintensität im Fall von Bildern den Funktionswert des jeweiligen Nachbarknotens x_ℓ, welcher mit dem Filterkoeffizienten g_ℓ gewichtet wird, um analog zur klassischen diskreten Faltung die Produkte zu summieren. Die L Nachbarknoten x_ℓ zu einem Knoten x mit $\ell = 1, \ldots, L$ erhält man durch Anwendung des entsprechenden *patch operator* S_ℓ auf den Knoten x. Dabei bezeichnet L die Anzahl der Knoten in der Nachbarschaft und auch die Anzahl der Filtergewichte g_ℓ, wobei $\ell = 1, \ldots, L$. Jeder der L verschiedenen *patch operators* S_ℓ liefert jeweils einen Knoten x_ℓ der Nachbarschaft. Es gilt also $x_\ell = S_\ell(x)$.

Im Fall eines Polygonnetzes mit fest vorgegebener Topologie ist die verallgemeinerte Faltung somit definiert durch

$$(f * g)_x = \sum_{\ell=1}^{L} g_\ell \, f\big(S_\ell(x)\big). \tag{6.2}$$

Zur Definition der durch Gleichung 6.2 gegebenen verallgemeinerten Faltung ist lediglich die Definition eines *patch operator S* nötig. Diese wird im folgenden Abschnitt präsentiert.

Zusammenfassend lässt sich sagen, dass die im allgemeinen Fall wichtige Permutationsinvarianz, welche *graph neural networks* wie in Abbildung 2.3 auf Seite 15 gezeigt grundsätzlich implementieren sollten, in Spezialfällen wie dem eines Polygonnetzes mit fest vorgegebener Topologie unberücksichtigt bleiben kann.

6.4. Funktionsweise im Detail

Lim u. a. [Lim+19] präsentierten im Jahr 2019 einen simplen und effizienten *patch operator*, der ein *resampling* eines Polygonnetzes unnötig macht und die Nachbarschaftsbeziehungen im Polygonnetz auf direkte Weise enkodiert. Lim u. a. bestimmen für einen Knoten v eine Nachbarschaft $\mathcal{N}$, indem sie von dem zentralen Knoten v ausgehend den Graphen wie in Abbildung 6.1 veranschaulicht spiralförmig abwandern und auf diese Weise die Elemente der k-disk des Knotens v in eine lokale Reihenfolge bringen. Definiert ist die k-disk eines Knotens v durch

$$\begin{aligned} 0\text{-ring}(v) &= \{v\}, \\ (k{+}1)\text{-ring}(v) &= N\big(k\text{-ring}(v)\big) \setminus k\text{-disk}(v), \\ k\text{-disk}(v) &= \bigcup_{i=0}^{k} i\text{-ring}(v). \end{aligned}$$

Der Knoten v selbst bildet dessen 0-ring. All diejenigen Knoten, die mit einem beliebigen Knoten im k-ring des Knotens v benachbart sind, bilden die Menge $N\big(k\text{-ring}(v)\big)$. Der 1-ring von v besteht somit aus den direkten Nachbarn von v, v selbst nicht eingeschlossen. Allgemeiner ist der $(k{+}1)$-ring eines Knotens v die Menge aller Knoten, die mit einem beliebigen Knoten im k-ring von v benachbart sind, wobei die in der k-disk liegenden Knoten ausgeschlossen werden. Die k-disk eines Knotens v ist die Vereinigung aller bisherigen i-rings, wobei $i = 0, \ldots, k$.

Nachdem auf diese Weise wie in Abbildung 6.1 veranschaulicht eine lokale Reihenfolge der Knoten einer Nachbarschaft etabliert wurde, wird durch

$$S(v) = \big(v, 1\text{-ring}(v), \ldots, k\text{-ring}(v)\big)$$

ein *spiral patch operator* S definiert. Insgesamt liefert der *patch operator* S somit die Sequenz $(v_1, \ldots, v_L)$ bestehend aus den reihenweise angeordneten Nachbarknoten von v. Lim u. a., welche diesen *spiral patch operator* vorgestellt haben, übergeben die damit erhaltene Knotensequenz einem rekurrenten neuronalen Netzwerk beziehungsweise einem *LSTM (long short-term memory) network*. Bouritsas u. a. [Bou+19] jedoch nutzen den *spiral patch operator* zur Definition einer *spiral convolution*, die durch Gleichung 6.2 gegeben ist. Innerhalb einer so realisierten Faltungsschicht eines neuronalen Netzwerks wird für jeden Knoten v dasselbe L verwendet.

Definitionsbedingt lässt dieser *spiral patch operator* kaum Freiheiten zu. Man hat jedoch zwei Wahlmöglichkeiten. Der erste auf den zentralen Knoten v folgende Knoten, also ein Knoten im 1-ring von v, kann selbst bestimmt werden. Zudem kann gewählt werden, ob der Graph im Uhrzeigersinn oder gegen den Uhrzeigersinn spiralförmig abgewandert

werden soll. Kulon u. a. [Kul+20] wählen den in der Sequenz ersten Knoten nach v zufällig aus und wandern den Graphen im Uhrzeigersinn ab.

Der mittels des *spiral patch operator S* gemäß Gleichung 6.2 definierte *spiral convolution*-Operator kann neben Polygonnetzen auch auf andere geometrische Datenstrukturen angewendet werden, für deren Elemente sich eine lokale Reihenfolge etablieren lässt, beispielsweise Punktwolken.

6.5. Modellarchitektur

Bei dem von Kulon u. a. [Kul+20] vorgestellten neuronalen Netzwerkmodell handelt es sich um einen aus Kapitel 4 bekannten *autoencoder*. Das Encoder-Netzwerk ist ein standardmäßiges, wie von He u. a. [He+15] vorgestelltes *ResNet-50*-Modell. Die Architektur des Decoder-Netzwerks von Kulon u. a. ist in Abbildung 6.3 dargestellt. Das Modell verwendet neben der verallgemeinerten Faltung, genannt *spiral convolution*, auch eine *upsampling*-Methode um, wie in Abbildung 6.2 zu sehen ist, eine Hierarchie von Topologien aufzustellen oder anders ausgedrückt, von immer feiner aufgelösten Handmodellen.

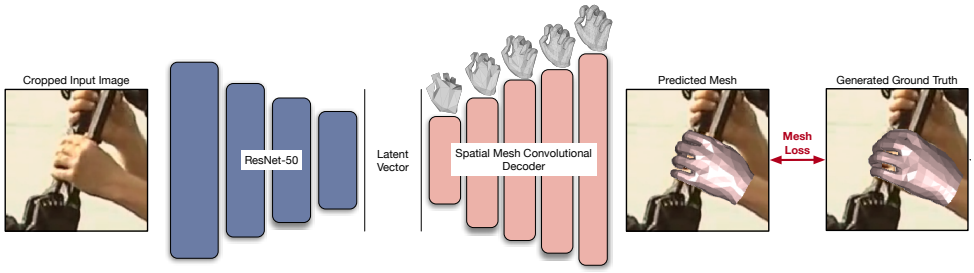


Abbildung 6.2.: **Modellarchitektur des *autoencoder* von Kulon u. a. [Kul+20].** Der Encoder ist ein *ResNet-50* [He+15]. Der Decoder ist in Abbildung 6.3 schematisch skizziert. [Kul+20]

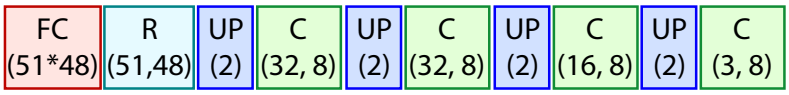


Abbildung 6.3.: **Architektur des Decoders.** Der Decoder besteht aus einer mit *FC* bezeichneten *fully connected layer*, die gefolgt wird von einer mit *R* für *Reshaping* bezeichneten Schicht, in welcher der Ausgabevektor der *fully connected layer* in eine 51×48-Matrix umgeformt wird. Es folgen abwechselnd Schichten, in denen *upsampling* beziehungsweise die *spiral convolution* ausgeführt wird. [Kul+20]

7. Zusammenfassung

In diesem Schlusskapitel werden der Hauptgedanke und die Kernpunkte der vorliegenden Arbeit kurz zusammengefasst. Im Anschluss wird ein Fazit gezogen und ein Ausblick auf die mögliche zukünftige Entwicklung von *geometric deep learning* gegeben.

7.1. Zusammenfassung

In dieser Ausarbeitung wurde ein umfassender Überblick über das aufsteigende und noch sehr junge Themengebiet *geometric deep learning* gegeben, dessen Begriff in [Eur17] von Bronstein u. a. geprägt wurde und durch deren wegweisende Veröffentlichung [Bro+17b] Verbreitung fand. Es wurde ein (für einen in den Bereichen des maschinellen Lernens und der Signalverarbeitung unerfahrenen Leser) zugänglicher Einstieg in das Teilgebiet *geometric deep learning* präsentiert.

In Kapitel 2 wurden nach dem einleitenden Kapitel 1 zunächst die nötigen Grundlagen aus den Gebieten der Topologie, Geometrie, mathematischen Physik, Gruppentheorie, Funktionalanalysis, Differentialgeometrie, Graphentheorie, Signalverarbeitung auf Graphen, harmonischen Analyse, Eigenwerttheorie sowie Spektraltheorie aufbereitet.

In Kapitel 3 wurden unter Verwendung der dargelegten Grundlagen vier verschiedene Herangehensweisen zur Verallgemeinerung der klassischen Faltung auf nichteuklidische Gebiete untersucht. Dieses Kapitel stellt den Kern dieser Arbeit dar.

Der erste Ansatz sind die Spektralmethoden, welche die direkte Interpretation der Intuition hinter der klassischen Faltung sind und eine der klassischen Faltung ähnliche Operation mittels einer verallgemeinerten Fourier-Basis im Spektrum des *Laplacian* im Fall von Mannigfaltigkeiten beziehungsweise *graph Laplacian* im Fall von Graphen definieren.

Der zweite Ansatz – die Spektrum-freien Methoden – knüpft an die Idee der Spektralmethoden an, umgeht oder behebt allerdings die fundamentalen Nachteile der Definition der faltungsähnlichen Operation im Spektralbereich etwa durch Parametrisierung des Filters mittels Polynomen oder rationaler Funktionen.

Der dritte Ansatz ist der Einsatz von kartografierenden Methoden (*charting-based methods*), welche das Problem der fehlenden globalen Parametrisierung lösen, indem sie für die Bereiche des Gebiets ein lokales intrinsisches System von Koordinaten konstruieren, was sie durch Definition von *patch operators* bewerkstelligen.

Der vierte Ansatz setzt auf Verfahren, welche die Unschärferelation zwischen Fourier-Paaren auflockern, indem sie anstelle der limitierenden Fourier-Transformation auf Weiterentwicklungen wie die Kurzzeit-Fourier-Transformation oder die Wavelet-Transformation zurückgreifen um eine gute Lokalisierung in beiden Bereichen der Fourier-Analysis erreichen zu können.

In Kapitel 4 wurde am Beispiel des Anwendungsfalls der Vervollständigung unvollständiger verformbarer Formen ein Netzwerkmodell basierend auf einer *graph convolution*-Operation vorgestellt, welches Polygonnetze direkt verarbeiten kann. Bei dem Modell handelt es sich um einen *variational autoencoder*.

In Kapitel 5 wurde am Beispiel des Anwendungsfalls der Klassifikation oder Segmentierung von Punktwolken ein Netzwerkmodell basierend auf einer *edge convolution*-Operation vorgestellt, welches Punktwolken direkt verarbeiten kann. Dazu wird zu einer Punktwolke ein dynamisch aktualisierbarer *k-nearest neighbor graph* konstruiert und zur Gewährleistung der Permutationsinvarianz eine symmetrische *aggregation operation* verwendet.

In Kapitel 6 wurde am Beispiel des Anwendungsfalls der Rekonstruktion von dreidimensionalen Handmodellen aus RGB-Bildern ein Netzwerkmodell basierend auf einer *spiral convolution*-Operation vorgestellt, welches Polygonnetze direkt verarbeiten kann. Für den Spezialfall eines Polygonnetzes mit unveränderlicher Topologie ist eine Konstruktion eines lokalen Systems von Koordinaten nicht nötig um die Knoten einer Nachbarschaft in eine Reihenfolge bringen zu können.

7.2. Fazit

Der letzte Abschnitt soll für ein Fazit dieser Arbeit und als Orientierung bezüglich möglicher zukünftiger Entwicklungen im Teilgebiet *geometric deep learning* genutzt werden.

Geometric deep learning ist ein Versuch, die zahlreichen bereits existierenden und noch nicht erfundenen Architekturen künstlicher neuronaler Netzwerke mit den Mitteln der Geometrie unter den Aspekten von Symmetrien und Invarianzen zu vereinheitlichen. Hierfür werden geometrische Prinzipien miteinander kombiniert, um gemeinsam eine universelle Blaupause für das Lernen stabiler Repräsentationen hochdimensionaler Daten zu schaffen. Zu diesem Zweck müssen die elementaren Bausteine künstlicher neuronaler Netzwerke wie etwa die Faltung oder das *pooling* neu definiert werden, um mit aus diesen Bausteinen zusammengesetzten Netzwerkmodellen auch auf nichteuklidischen Gebieten definierte Funktionen approximieren zu können.

Ähnlich wie klassische künstliche neuronale Netzwerkarchitekturen zunächst mit wenigen Schichten begannen und über die Jahre immer „tiefer“ wurden – vom Perzeptron zum *multi-layer perceptron* und in Verbindung mit der Entwicklung von Techniken wie *pooling* zum Faltungsnetzwerk (*convolutional neural network*) – so bestehen auch die Architekturen heutiger *graph neural networks* aus relativ wenigen Schichten. Ein aktueller Gegenstand der Forschung ist die Frage, ob im Fall eines *graph neural network* die Erhöhung der Zahl der Schichten die Leistungsfähigkeit der Modelle steigern kann. Im Gegensatz zu klassischen euklidischen neuronalen Netzwerken sieht es derzeit danach aus, dass dies nicht der Fall ist und dass mehr Schichten die Leistung sogar verschlechtern. [Zho+20]

Neben den Problemen, die auch klassische *deep neural networks* plagen wie etwa verschwindende Gradienten bei dem als *backpropagation* bezeichneten Verfahren zum Training von *feedforward*-Netzen oder die sich aus einer großen Anzahl von Parametern ergebende Gefahr des *overfitting*, haben Graphen auch einige spezifische Schwierigkeiten. Eine dieser Schwierigkeiten im Kontext des maschinellen Sehens (*computer vision*) ist, dass das Zusammensetzen von komplexen Strukturen aus einfachen geometrischen Primitiven

– wie aus vielen Schichten bestehende klassische Faltungsnetzwerke (*convolutional neural networks*) es machen – im Fall von Graphen nach derzeitigem Wissensstand unmöglich ist. Beispielsweise lassen sich aus Kanten keine Dreiecke zusammensetzen, ganz gleich wie viele Schichten das neuronale Netzwerk hat. [Arv+18; Che+20]

Aktuell fehlt das Verständnis diesbezüglich, welche Eigenschaften von Graphen sich mit *graph neural networks* mit wenigen Schichten berechnen lassen, welche Eigenschaften nur von *graph neural networks* mit vielen Schichten berechnet werden können und welche Eigenschaften sich in keinem Fall berechnen lassen.

Auf jeden Fall sollten aber *graph neural networks*, die auf *message passing* basieren, weiter verfolgt werden. *Message passing* erlaubt den Informationsaustausch zwischen den Knoten eines Graphen und kann somit als eine Form von Diffusion verstanden werden. Diffusion wird mittels Differentialgleichungen beschrieben, weshalb *graph neural networks* eng verwandt mit Differentialgleichungen sind. Eine neue Klasse von Netzwerkarchitekturen kann herbeigeführt werden, wenn man sich *graph neural networks* als diskrete partielle Differentialgleichungen vorstellt.

Tatsächlich haben vor dem Wiederaufleben von *deep learning* ab dem Jahr 2012 Methoden, die auf partiellen Differentialgleichungen und der Variationsrechnung basieren, die Bildverarbeitung dominiert. Zwei einflussreiche Personen waren – und sind es auch heute noch – Pietro Perona und Jitendra Malik, die im Jahr 1990 die in Unterunterabschnitt 3.4.5.2 thematisierte anisotrope Diffusion [PM90] formuliert haben, welche auch Perona-Malik-Diffusion genannt wird, und damit ein ganzes Gebiet von auf partiellen Differentialgleichungen basierten Methoden eröffnet haben. Mit Veröffentlichungen wie [Vel+18; Cha+21] folgt die aktuelle Forschung diesem auf partiellen Differentialgleichungen basierten Ansatz.

Zur weiteren Lektüre wird dem interessierten Leser die wegweisende und überaus einflussreiche Arbeit [Bro+17b] von Bronstein u. a. aus dem Jahr 2017 nachdrücklich empfohlen, durch welche die Bezeichnung *geometric deep learning* Verbreitung fand. Für den Einstieg in die Signalverarbeitung auf Graphen eignet sich [Shu+13] von Shuman u. a. Für ein noch tiefer greifendes Verständnis von *geometric deep learning*, das weit über [Bro+17b] und die in Kapitel 3 vorgestellten Ansätze zur Verallgemeinerung der Faltung hinaus geht und in seiner Gänze unmöglich im Rahmen dieser Bachelorarbeit abgedeckt werden kann, sei dem geneigten Leser noch die Arbeit [Bro+21a] von Bronstein u. a. aus dem Jahr 2021 empfohlen. [Bro+21a; Bro20b; Bro21b]

Literatur

[And+14] Mathieu Andreux u. a. „Anisotropic Laplace-Beltrami Operators for Shape Analysis“. In: Sep. 2014. ISBN: 978-3-319-16219-5. DOI: 10.1007/978-3-319-16220-1_21.

[Ang+05] Dragomir Anguelov u. a. „SCAPE: Shape Completion and Animation of People“. In: *ACM Trans. Graph.* 24.3 (Juli 2005), S. 408–416. ISSN: 0730-0301. DOI: 10.1145/1073204.1073207. URL: https://doi.org/10.1145/1073204.1073207.

[Arm+16] Iro Armeni u. a. „3D Semantic Parsing of Large-Scale Indoor Spaces“. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, S. 1534–1543. DOI: 10.1109/CVPR.2016.170.

[Arv+18] Vikraman Arvind u. a. „On Weisfeiler-Leman Invariance: Subgraph Counts and Related Graph Properties“. In: *CoRR* abs/1811.04801 (2018). arXiv: 1811.04801. URL: http://arxiv.org/abs/1811.04801.

[Bam20] Bassam Bamieh. *Discovering Transforms: A Tutorial on Circulant Matrices, Circular Convolution, and the Discrete Fourier Transform*. 2020. arXiv: 1805.05533 [eess.SP].

[Bat+18] Peter W. Battaglia u. a. „Relational inductive biases, deep learning, and graph networks“. In: *CoRR* abs/1806.01261 (2018). arXiv: 1806.01261. URL: http://arxiv.org/abs/1806.01261.

[Bia+16] Silvia Biasotti u. a. „Recent Trends, Applications, and Perspectives in 3D Shape Similarity Assessment“. In: *Comput. Graph. Forum* 35.6 (Sep. 2016), S. 87–119. ISSN: 0167-7055. DOI: 10.1111/cgf.12734. URL: https://doi.org/10.1111/cgf.12734.

[BKM17] David M. Blei, Alp Kucukelbir und Jon D. McAuliffe. „Variational Inference: A Review for Statisticians“. In: *Journal of the American Statistical Association* 112.518 (Apr. 2017), S. 859–877. ISSN: 1537-274X. DOI: 10.1080/01621459.2017.1285773. URL: http://dx.doi.org/10.1080/01621459.2017.1285773.

[Bog+17] Federica Bogo u. a. „Dynamic FAUST: Registering Human Bodies in Motion“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juli 2017.

[Bos+15] Davide Boscaini u. a. „Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks“. In: *Computer Graphics Forum* 34.5 (2015), S. 13–23. DOI: https://doi.org/10.1111/cgf.12693. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12693. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12693.

[Bos+16a] Davide Boscaini u. a. „Anisotropic Diffusion Descriptors“. In: *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics*. EG ’16. Lisbon, Portugal: Eurographics Association, 2016, S. 431–441.

[Bos+16b] Davide Boscaini u. a. „Learning Shape Correspondence with Anisotropic Convolutional Neural Networks“. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS’16. Barcelona, Spain: Curran Associates Inc., 2016, S. 3197–3205. ISBN: 9781510838819.

[Bou+19] Giorgos Bouritsas u. a. „Neural 3D Morphable Models: Spiral Convolutional Networks for 3D Shape Representation Learning and Generation“. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, S. 7212–7221. DOI: 10.1109/ICCV.2019.00731.

[Bro+17a] Michael M. Bronstein u. a. *Geometric Deep Learning on Graphs and Manifolds*. 2017. URL: https://nips.cc/Conferences/2017/Schedule?showEvent=8735 (besucht am 18. 05. 2021).

[Bro+17b] Michael M. Bronstein u. a. „Geometric Deep Learning: Going beyond Euclidean data“. In: *IEEE Signal Processing Magazine* 34.4 (2017), S. 18–42. DOI: 10.1109/MSP.2017.2693418.

[Bro+21a] Michael M. Bronstein u. a. „Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges“. In: *CoRR* abs/2104.13478 (2021). arXiv: 2104.13478. URL: https://arxiv.org/abs/2104.13478.

[Bro+21b] Michael M. Bronstein u. a. *Geometric foundations of Deep Learning*. 2021. URL: https://towardsdatascience.com/geometric-foundations-of-deep-learning-94cdd45b451d (besucht am 01. 08. 2021).

[Bro19] Alexander M. Bronstein. *Lecture 11: Learning on Non-Euclidean Domains*. 2019. URL: https://vistalab-technion.github.io/cs236781/lecture_notes/lecture_11/ (besucht am 01. 08. 2021).

[Bro20a] Michael M. Bronstein. *Deriving convolution from first principles*. 2020. URL: https://towardsdatascience.com/deriving-convolution-from-first-principles-4ff124888028 (besucht am 01. 08. 2021).

[Bro20b] Michael M. Bronstein. *Do we need deep graph neural networks?* 2020. URL: https://towardsdatascience.com/do-we-need-deep-graph-neural-networks-be62d3ec5c59 (besucht am 01. 08. 2021).

[Bro21a] Michael M. Bronstein. *Geometric Deep Learning: the Erlangen Programme of ML*. 2021. URL: https://iclr.cc/virtual/2021/invited-talk/3717 (besucht am 07. 07. 2021).

[Bro21b] Michael M. Bronstein. *Graph Neural Networks as Neural Diffusion PDEs*. 2021. URL: https://towardsdatascience.com/graph-neural-networks-as-neural-diffusion-pdes-8571b8c0c774 (besucht am 01. 08. 2021).

[Bru+14] Joan Bruna u. a. „Spectral Networks and Locally Connected Networks on Graphs“. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Hrsg. von Yoshua Bengio und Yann LeCun. 2014. URL: http://arxiv.org/abs/1312.6203.

[CCM15] Xiuyuan Cheng, Xu Chen und Stéphane Mallat. „Deep Haar Scattering Networks“. In: *CoRR* abs/1509.09187 (2015). arXiv: 1509.09187. URL: http://arxiv.org/abs/1509.09187.

[Cha+21] Benjamin Paul Chamberlain u. a. „GRAND: Graph Neural Diffusion“. In: *CoRR* abs/2106.10934 (2021). arXiv: 2106.10934. URL: https://arxiv.org/abs/2106.10934.

[Che+20] Zhengdao Chen u. a. „Can graph neural networks count substructures?“ In: *CoRR* abs/2002.04025 (2020). arXiv: 2002.04025. URL: https://arxiv.org/abs/2002.04025.

[Chi+19] Wei-Lin Chiang u. a. „Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks“. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, S. 257–266. ISBN: 9781450362016. DOI: 10.1145/3292500.3330925. URL: https://doi.org/10.1145/3292500.3330925.

[CM06] Ronald R. Coifman und Mauro Maggioni. „Diffusion wavelets“. In: *Applied and Computational Harmonic Analysis* 21.1 (2006). Special Issue: Diffusion Maps and Wavelets, S. 53–94. ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2006.04.004. URL: https://www.sciencedirect.com/science/article/pii/S106352030600056X.

[Coh+19] Taco S. Cohen u. a. „Gauge Equivariant Convolutional Networks and the Icosahedral CNN“. In: *CoRR* abs/1902.04615 (2019). arXiv: 1902.04615. URL: http://arxiv.org/abs/1902.04615.

[DBV16] Michaël Defferrard, Xavier Bresson und Pierre Vandergheynst. „Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering“. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, S. 3844–3852. ISBN: 9781510838819.

[Dee21] Dive into Deep Learning. *Convolutions for Images*. 2021. URL: https://d2l.ai/chapter_convolutional-neural-networks/conv-layer.html (besucht am 18. 05. 2021).

[DW19] Bin Dai und David P. Wipf. „Diagnosing and Enhancing VAE Models“. In: *CoRR* abs/1903.05789 (2019). arXiv: 1903.05789. URL: http://arxiv.org/abs/1903.05789.

[Epp11] David Eppstein. *Nearest neighbor graph*. 2011. URL: https://commons.wikimedia.org/wiki/File:Nearest_neighbor_graph.svg (besucht am 17. 06. 2021).

[Eur17] European Commision. *Deep LEarning on MANifolds and graphs | LEMAN Project | H2020*. 2017. URL: https://cordis.europa.eu/project/id/724228 (besucht am 01. 08. 2021).

[GBC16] Ian J. Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. https://www.deeplearningbook.org. MIT Press, 2016.

[Gil+17] Justin Gilmer u. a. „Neural Message Passing for Quantum Chemistry". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, S. 1263–1272.

[GMS05] Marco Gori, Gabriele Monfardini und Franco Scarselli. „A new model for learning in graph domains". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Bd. 2. 2005, 729–734 vol. 2. DOI: 10.1109/IJCNN.2005.1555942.

[GNC10] Matan Gavish, Boaz Nadler und Ronald R. Coifman. „Multiscale Wavelets on Trees, Graphs and High Dimensional Data: Theory and Applications to Semi Supervised Learning". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, S. 367–374. ISBN: 9781605589077.

[Goo+14] Ian J. Goodfellow u. a. „Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Hrsg. von Z. Ghahramani u. a. Bd. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[Har17] Harvard University. *Geometric Deep Learning | Michael Bronstein || Radcliffe Institute*. 2017. URL: https://www.youtube.com/watch?v=ptcBmEHDWds (besucht am 18. 05. 2021).

[HBL15] Mikael Henaff, Joan Bruna und Yann LeCun. „Deep Convolutional Networks on Graph-Structured Data". In: *CoRR* abs/1506.05163 (2015). arXiv: 1506.05163. URL: http://arxiv.org/abs/1506.05163.

[He+15] Kaiming He u. a. „Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[HVG11] David K. Hammond, Pierre Vandergheynst und Rémi Gribonval. „Wavelets on graphs via spectral graph theory". In: *Applied and Computational Harmonic Analysis* 30.2 (2011), S. 129–150. ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2010.04.005. URL: https://www.sciencedirect.com/science/article/pii/S1063520310000552.

[HW62] David H. Hubel und Torsten N. Wiesel. „Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of Physiology* 160.1 (1962), S. 106–154. DOI: https://doi.org/10.1113/jphysiol.1962.sp006837. eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1962.sp006837. URL: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1962.sp006837.

[Jad+15] Max Jaderberg u. a. „Spatial Transformer Networks“. In: *CoRR* abs/1506.02025 (2015). arXiv: 1506.02025. URL: http://arxiv.org/abs/1506.02025.

[KHC11] Oliver van Kaick, Ghassan Hamarneh und Daniel Cohen-Or. „A Survey on Shape Correspondence“. In: *Comput. Graph. Forum* 30 (Sep. 2011), S. 1681–1707. DOI: 10.1111/j.1467-8659.2011.01884.x.

[KSH12] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton. „ImageNet Classification with Deep Convolutional Neural Networks“. In: *Advances in Neural Information Processing Systems*. Hrsg. von F. Pereira u. a. Bd. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[Kul+19] Dominik Kulon u. a. „Single Image 3D Hand Reconstruction with Mesh Convolutions“. In: *CoRR* abs/1905.01326 (2019). arXiv: 1905.01326. URL: http://arxiv.org/abs/1905.01326.

[Kul+20] Dominik Kulon u. a. „Weakly-Supervised Mesh-Convolutional Hand Reconstruction in the Wild“. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, S. 4989–4999. DOI: 10.1109/CVPR42600.2020.00504.

[KW14] Diederik P. Kingma und Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[KW16] Thomas N. Kipf und Max Welling. „Semi-Supervised Classification with Graph Convolutional Networks“. In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

[KW19] Diederik P. Kingma und Max Welling. „An Introduction to Variational Autoencoders“. In: *Foundations and Trends® in Machine Learning* 12.4 (2019), S. 307–392. ISSN: 1935-8245. DOI: 10.1561/2200000056. URL: http://dx.doi.org/10.1561/2200000056.

[Lar+18] Tara Larrue u. a. *Denoising Videos with Convolutional Autoencoders*. 2018. URL: https://www.cs.umd.edu/sites/default/files/scholarly_papers/Larrue%2C%20Tara_1801.pdf.

[LeC+89] Yann LeCun u. a. „Backpropagation Applied to Handwritten Zip Code Recognition“. In: *Neural Computation* 1.4 (1989), S. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[Lev+19] Ron Levie u. a. „CayleyNets: Graph Convolutional Neural Networks With Complex Rational Spectral Filters“. In: *IEEE Transactions on Signal Processing* 67.1 (2019), S. 97–109. DOI: 10.1109/TSP.2018.2879624.

[Li+16] Yujia Li u. a. „Gated Graph Sequence Neural Networks“. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Hrsg. von Yoshua Bengio und Yann LeCun. 2016. URL: http://arxiv.org/abs/1511.05493.

[Lim+19] Isaak Lim u. a. „A Simple Approach to Intrinsic Correspondence Learning on Unstructured 3D Meshes“. In: *Computer Vision – ECCV 2018 Workshops*. Hrsg. von Laura Leal-Taixé und Stefan Roth. Cham: Springer International Publishing, 2019, S. 349–362. ISBN: 978-3-030-11015-4.

[Lit+18] Or Litany u. a. „Deformable Shape Completion with Graph Convolutional Autoencoders“. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, S. 1886–1895. DOI: 10.1109/CVPR.2018.00202.

[Lop+15] Matthew Loper u. a. „SMPL: A Skinned Multi-Person Linear Model“. In: *ACM Trans. Graph.* 34.6 (Okt. 2015). ISSN: 0730-0301. DOI: 10.1145/2816795.2818013. URL: https://doi.org/10.1145/2816795.2818013.

[Mas+15a] Jonathan Masci u. a. „Geodesic Convolutional Neural Networks on Riemannian Manifolds“. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*. ICCVW '15. USA: IEEE Computer Society, 2015, S. 832–840. ISBN: 9781467397117. DOI: 10.1109/ICCVW.2015.112. URL: https://doi.org/10.1109/ICCVW.2015.112.

[Mas+15b] Jonathan Masci u. a. „ShapeNet: Convolutional Neural Networks on Non-Euclidean Manifolds“. In: *CoRR* abs/1501.06297v1 (2015). arXiv: 1501.06297v1. URL: http://arxiv.org/abs/1501.06297v1.

[MBB17] Federico Monti, Michael M. Bronstein und Xavier Bresson. „Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks“. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, S. 3700–3710. ISBN: 9781510860964.

[Mit97] Thomas M. Mitchell. *Machine Learning*. 1. Aufl. USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077.

[MOB18] Federico Monti, Karl Otness und Michael M. Bronstein. „MotifNet: a motif-based Graph Convolutional Network for directed graphs“. In: *2018 IEEE Data Science Workshop (DSW)*. 2018, S. 225–228. DOI: 10.1109/DSW.2018.8439897.

[Mon+17] Federico Monti u. a. „Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs“. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, S. 5425–5434. DOI: 10.1109/CVPR.2017.576.

[PM90] Pietro Perona und Jitendra Malik. „Scale-space and edge detection using anisotropic diffusion“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.7 (1990), S. 629–639. DOI: 10.1109/34.56205.

[Qi+17a] Charles R. Qi u. a. „PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation“. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juli 2017, S. 77–85. DOI: 10.1109/CVPR.2017.16.

[Qi+17b] Charles R. Qi u. a. „PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space“. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, S. 5105–5114. ISBN: 9781510860964.

[RG13] Raif Rustamov und Leonidas J. Guibas. „Wavelets on Graphs via Deep Learning“. In: *Advances in Neural Information Processing Systems*. Hrsg. von C. J. C. Burges u. a. Bd. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper/2013/file/33e8075e9970de0cfea955afd4644bb2-Paper.pdf.

[RTB17] Javier Romero, Dimitrios Tzionas und Michael J. Black. „Embodied Hands: Modeling and Capturing Hands and Bodies Together“. In: *ACM Trans. Graph.* 36.6 (Nov. 2017). ISSN: 0730-0301. DOI: 10.1145/3130800.3130883. URL: https://doi.org/10.1145/3130800.3130883.

[Sca+09] Franco Scarselli u. a. „The Graph Neural Network Model“. In: *Trans. Neur. Netw.* 20.1 (Jan. 2009), S. 61–80. ISSN: 1045-9227. DOI: 10.1109/TNN.2008.2005605. URL: https://doi.org/10.1109/TNN.2008.2005605.

[Shu+13] David I Shuman u. a. „The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains“. In: *IEEE Signal Processing Magazine* 30.3 (2013), S. 83–98. DOI: 10.1109/MSP.2012.2235192.

[Son+16] Shuran Song u. a. „Semantic Scene Completion from a Single Depth Image“. In: *CoRR* abs/1611.08974 (2016). arXiv: 1611.08974. URL: http://arxiv.org/abs/1611.08974.

[Spi12] Daniel Spielman. „Spectral Graph Theory“. In: *Combinatorial Scientific Computing*. Jan. 2012, S. 495–524. ISBN: 978-1-4398-2735-2. DOI: 10.1201/b11644-22.

[SRV16] David I Shuman, Benjamin Ricaud und Pierre Vandergheynst. „Vertex-frequency analysis on graphs“. In: *Applied and Computational Harmonic Analysis* 40.2 (2016), S. 260–291. ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2015.02.005. URL: https://www.sciencedirect.com/science/article/pii/S1063520315000214.

[Szl+05] Arthur D. Szlam u. a. „Diffusion-driven multiscale analysis on manifolds and graphs: top-down and bottom-up constructions“. In: *Wavelets XI*. Hrsg. von Manos Papadakis, Andrew F. Laine und Michael A. Unser. Bd. 5914. International Society for Optics und Photonics. SPIE, 2005, S. 445–455. URL: https://doi.org/10.1117/12.616931.

[TO18] Corentin Tallec und Yann Ollivier. „Can recurrent neural networks warp time?“ In: *CoRR* abs/1804.11188 (2018). arXiv: 1804.11188. URL: http://arxiv.org/abs/1804.11188.

[Vas+17] Ashish Vaswani u. a. „Attention is All you Need“. In: *Advances in Neural Information Processing Systems*. Hrsg. von I. Guyon u. a. Bd. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[VBV17] Nitika Verma, Edmond Boyer und Jakob Verbeek. „Dynamic Filters in Graph Convolutional Networks“. In: *CoRR* abs/1706.05206 (2017). arXiv: 1706.05206. URL: http://arxiv.org/abs/1706.05206.

[Vel+18] Petar Veličković u. a. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].

[Wan+19] Yue Wang u. a. „Dynamic Graph CNN for Learning on Point Clouds“. In: *ACM Trans. Graph.* 38.5 (Okt. 2019). ISSN: 0730-0301. DOI: 10.1145/3326362. URL: https://doi.org/10.1145/3326362.

[WD21] Jingyi Wang und Zhidong Deng. „A Deep Graph Wavelet Convolutional Neural Network for Semi-supervised Node Classification“. In: *CoRR* abs/2102.09780 (2021). arXiv: 2102.09780. URL: https://arxiv.org/abs/2102.09780.

[Wei+15] Lingyu Wei u. a. „Dense Human Body Correspondences Using Convolutional Networks“. In: *CoRR* abs/1511.05904 (2015). arXiv: 1511.05904. URL: http://arxiv.org/abs/1511.05904.

[Wu+15] Zhirong Wu u. a. „3D ShapeNets: A deep representation for volumetric shapes“. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, S. 1912–1920. DOI: 10.1109/CVPR.2015.7298801.

[Wu+19] Felix Wu u. a. *Simplifying Graph Convolutional Networks*. 2019. arXiv: 1902.07153 [cs.LG].

[Xu+19] Bingbing Xu u. a. „Graph Wavelet Neural Network“. In: *CoRR* abs/1904.07785 (2019). arXiv: 1904.07785. URL: http://arxiv.org/abs/1904.07785.

[Yi+16] Li Yi u. a. „A Scalable Active Framework for Region Annotation in 3D Shape Collections“. In: *ACM Trans. Graph.* 35.6 (Nov. 2016). ISSN: 0730-0301. DOI: 10.1145/2980179.2980238. URL: https://doi.org/10.1145/2980179.2980238.

[Zah+17] Manzil Zaheer u. a. „Deep Sets“. In: *Advances in Neural Information Processing Systems*. Hrsg. von I. Guyon u. a. Bd. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf.

[Zho+20] Kuangqi Zhou u. a. „Effective Training Strategies for Deep Graph Neural Networks“. In: *CoRR* abs/2006.07107 (2020). arXiv: 2006.07107. URL: https://arxiv.org/abs/2006.07107.

A. Anhang

A.1. Vervollständigung verformbarer Formen

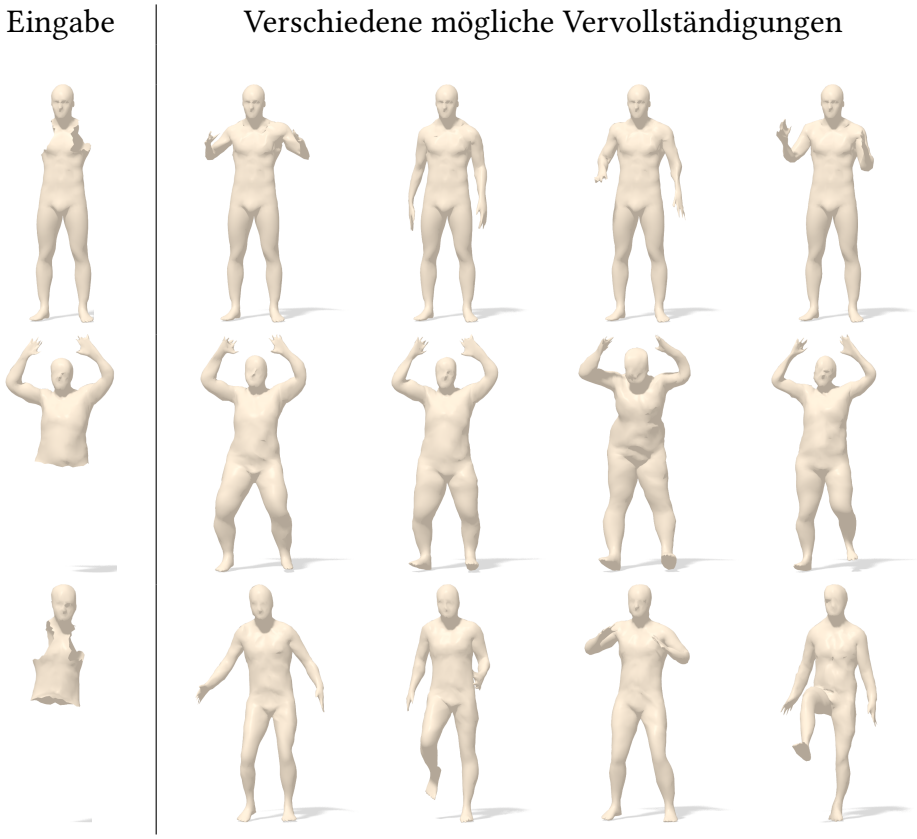


Abbildung A.1.: **Beispiele für verschiedene mögliche Vervollständigungen unvollständiger Formen.** Die einzelnen Reihen zeigen jeweils eine unvollständige Form als Eingabe (links), sowie vier unterschiedliche plausible Vervollständigungen (rechts), die allesamt zulässige Lösungen der nicht eindeutigen Problemstellung darstellen. [Lit+18]

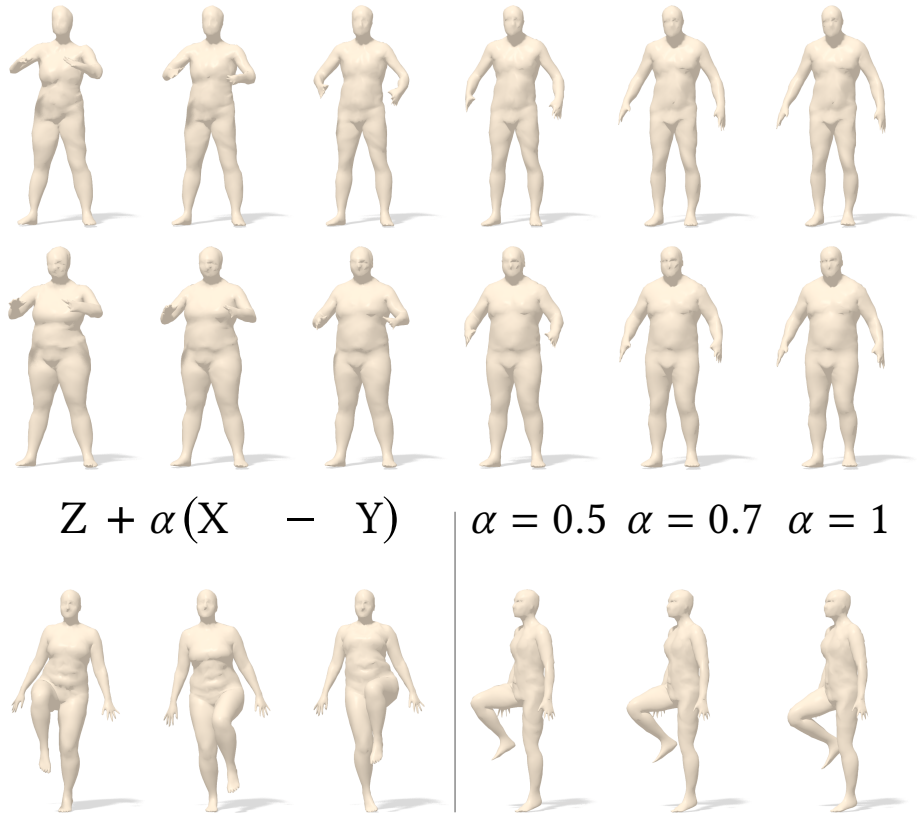
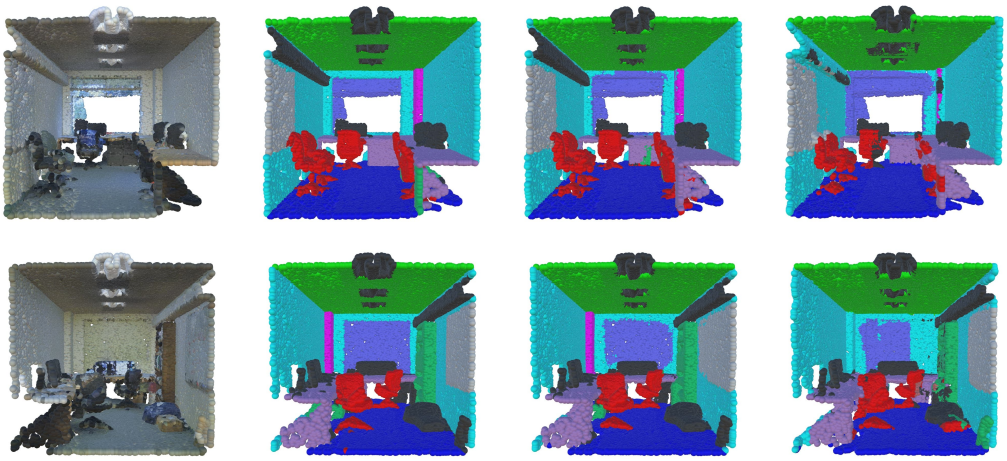


Abbildung A.2.: **Illustration der Interpolation und Arithmetik im *latent space*.** Die oberen beiden Reihen zeigen jeweils ein Beispiel für die Interpolation zwischen der linkesten Pose $\mathbf{X} = \text{dec}(\mathbf{z})$ und der rechtesten Pose $\mathbf{Y} = \text{dec}(\mathbf{z}')$. Die mittleren Posen erhält man durch Decodierung konvexer Kombinationen von $\mathbf{z}$ und $\mathbf{z}'$. Die untere Reihe veranschaulicht die Arithmetik im *latent space*. Für drei Posen $\mathbf{X} = \text{dec}(\mathbf{z})$, $\mathbf{Y} = \text{dec}(\mathbf{z}')$ und $\mathbf{Z} = \text{dec}(\mathbf{z}'')$ kann im *latent space* die Differenz $\mathbf{z} - \mathbf{z}'$ auf $\mathbf{z}''$ addiert werden, um das Bein der Pose $\mathbf{Z}$ abzusenken was mit α noch kontrolliert werden kann. [Lit+18]

A.2. Klassifikation und Segmentierung von Punktwolken



Original *Ground truth* *DGCNN* *PointNet*

Abbildung A.3.: **Beispiele verschiedener semantischer Segmentierungen von Punktwolken.** Von links nach rechts: Ursprüngliche Punktwolke mit Farbinformation, *ground truth*, Segmentierung mit *Dynamic Graph CNN (DGCNN)*, Segmentierung mit *PointNet*. *Dynamic Graph CNN* liefert bessere Segmentierungsergebnisse als *PointNet*, was beispielsweise an der Säule (magenta) oder den Füßen der Stühle (rot) ersichtlich ist. [Wan+19]