Deep Learning on Attributed Graphs

Martin Simonovsky

UNIVERSITÉ PARIS-EST

Deep Learning on Attributed Graphs

A Journey from Graphs to Their Embeddings and Back

Martin SIMONOVSKY

A doctoral thesis in the domain of automated signal and image processing supervised by **Nikos KOMODAKIS**

Submitted to **École Doctorale Paris-Est**
Mathématiques et Sciences et Technologies de l'Information et de la Communication.

Presented on 14 December 2018 to a committee consisting of:

Nikos Komodakis	École des Ponts ParisTech	Supervisor
Matthew B. Blaschko	KU Leuven	Reviewer
Stephen Gould	Australian National University	Reviewer
Renaud Marlet	École des Ponts ParisTech	Examiner
Josiane Zerubia	INRIA Sophia Antipolis	Committee chair

École des Ponts ParisTech
LIGM-IMAGINE
6, Av Blaise Pascal - Cité Descartes
Champs-sur-Marne
77455 Marne-la-Vallée cedex 2
France

Université Paris-Est Marne-la-Vallée
École Doctorale Paris-Est MSTIC
Département Études Doctorales
6, Av Blaise Pascal - Cité Descartes
Champs-sur-Marne
77454 Marne-la-Vallée cedex 2
France

Abstract

A graph is a powerful concept for representation of relations between pairs of entities. Data with underlying graph structure can be found across many disciplines, describing chemical compounds, surfaces of three-dimensional models, social interactions, or knowledge bases, to name only a few. There is a natural desire for understanding such data better. Deep learning (DL) has achieved significant breakthroughs in a variety of machine learning tasks in recent years, especially where data is structured on a grid, such as in text, speech, or image understanding. However, surprisingly little has been done to explore the applicability of DL on arbitrary graph-structured data directly.

The goal of this thesis is to investigate architectures for DL on graphs and study how to transfer, adapt or generalize concepts that work well on sequential and image data to this domain. We concentrate on two important primitives: embedding graphs or their nodes into a continuous vector space representation (encoding) and, conversely, generating graphs from such vectors back (decoding). To that end, we make the following contributions.

First, we introduce Edge-Conditioned Convolutions (ECC), a convolution-like operation on graphs performed in the spatial domain where filters are dynamically generated based on edge attributes. The method is used to encode graphs with arbitrary and varying structure.

Second, we propose SuperPoint Graph, an intermediate point cloud representation with rich edge attributes encoding the contextual relationship between object parts. Based on this representation, ECC is employed to segment large-scale point clouds without major sacrifice in fine details.

Third, we present GraphVAE, a graph generator allowing us to decode graphs with variable but upper-bounded number of nodes making use of approximate graph matching for aligning the predictions of an autoencoder with its inputs. The method is applied to the task of molecule generation.

Keywords: deep learning, graph convolutions, graph embedding, graph generation, point cloud segmentation

Résumé

Le graphe est un concept puissant pour la représentation des relations entre des paires d'entités. Les données ayant une structure de graphes sous-jacente peuvent être trouvées dans de nombreuses disciplines, décrivant des composés chimiques, des surfaces des modèles tridimensionnels, des interactions sociales ou des bases de connaissance, pour n'en nommer que quelques-unes. L'apprentissage profond (DL) a accompli des avancées significatives dans une variété de tâches d'apprentissage automatique au cours des dernières années, particulièrement lorsque les données sont structurées sur une grille, comme dans la compréhension du texte, de la parole ou des images. Cependant, étonnamment peu de choses ont été faites pour explorer l'applicabilité de DL directement sur des données structurées sous forme des graphes.

L'objectif de cette thèse est d'étudier des architectures de DL sur des graphes et de rechercher comment transférer, adapter ou généraliser à ce domaine des concepts qui fonctionnent bien sur des données séquentielles et des images. Nous nous concentrons sur deux primitives importantes : le plongement de graphes ou leurs nœuds dans une représentation de l'espace vectorielle continue (codage) et, inversement, la génération des graphes à partir de ces vecteurs (décodage). Nous faisons les contributions suivantes.

Tout d'abord, nous introduisons Edge-Conditioned Convolutions (ECC), une opération de type convolution sur les graphes réalisés dans le domaine spatial où les filtres sont générés dynamiquement en fonction des attributs des arêtes. La méthode est utilisée pour coder des graphes avec une structure arbitraire et variable.

Deuxièmement, nous proposons SuperPoint Graph, une représentation intermédiaire de nuages de points avec de riches attributs des arêtes codant la relation contextuelle entre des parties des objets. Sur la base de cette représentation, l'ECC est utilisé pour segmenter les nuages de points à grande échelle sans sacrifier les détails les plus fins.

Troisièmement, nous présentons GraphVAE, un générateur de graphes permettant de décoder des graphes avec un nombre de nœuds variable mais limité en haut, en utilisant la correspondance approximative des graphes pour aligner les prédictions d'un auto-encodeur avec ses entrées. La méthode est appliquée à génération de molécules.

Mots clés: apprentissage profond, convolution sur de graphes, plongement de graphes, génération de graphes, segmentation des nuage de points

Résumé substantiel

Le graphe est un concept puissant pour la représentation des relations entre les paires d'entités. Sa polyvalence et sa solide compréhension théorique permettent une pléthore de cas d'utilisation dans une variété de disciplines scientifiques et de problèmes techniques. Par exemple, les graphes peuvent naturellement être utilisés pour décrire la structure des composés chimiques, les interactions entre les régions du cerveau, les interactions sociales entre les personnes, la topologie des modèles tridimensionnels et des plans de transport, les dépendances des pages Web liées, les flux des programmes ou les bases de connaissance. Le développement de la théorie et des algorithmes pour manipuler les graphes a donc été d'un intérêt majeur.

La théorie classique des graphes, que remonte au XVIIIe siècle, se concentre sur la compréhension et l'analyse de la structure des graphes et sur les problèmes combinatoires tels que la recherche de applications, chemins, flots, ou colorations. D'autre part, il y a eu beaucoup de recherches sur les données structurées par graphes au cours des dernières décennies, particulièrement dans les disciplines du traitement du signal et de l'apprentissage automatique. Dans le premier cas, la recherche a notamment porté la transformée de Fourier à des domaines irréguliers, donnant naissance à la théorie spectrale des graphes et adaptant des opérations fondamentales de traitement du signal comme le filtrage, la translation, la dilatation ou la sous-échantillonnage (Shuman et al., 2013). Habituellement, un graphe fixe est considéré avec des données changeantes sur ses nœuds. Dans le domaine de l'apprentissage automatique, les chercheurs ont explicitement utilisé des structures de graphes dans les données dans des domaines tels que le partitionnement des graphes pour le clustering, la propagation des étiquettes pour l'apprentissage semi-supervisé, des méthodes sur variétés pour réduire la dimensionnalité, des noyaux pour décrire la similarité entre différents graphes ou des modèles graphiques pour enregistrer une interprétation probabiliste des données.

Au cours des dernières années, l'apprentissage profond (DL) a permis de réaliser des avancées significatives en termes de performance quantitative par rapport aux approches traditionnelles d'apprentissage automatique dans un éventail de tâches et le domaine est devenu une technologie utile pour les applications commerciales. En particulier, une classe d'architecture spécifique, réseau neuronal convolutif (CNN), a gagné en popularité dans les tâches où la représentation des données sous-jacentes a une structure de grille, comme dans le traitement de la parole et la compréhension du langage naturel (1D, convolutions temporelles), dans la classification et la segmentation des images (2D, convolutions

spatiales) et dans la compréhension des vidéos et images médicales (3D, convolutions volumétriques) (LeCun et al., 2015). D'autre part, le cas des données sur des domaines irréguliers ou généralement non euclidiens avait reçu comparativement beaucoup moins d'attention dans la communauté mais est devenu un sujet brûlant lors de la préparation de cette thèse (Bronstein et al., 2017).

Encouragés par le succès de DL sur les grilles, nous étudions dans cette thèse les architectures pour les données structurées en graphes. Plus précisément, nous nous concentrons sur deux primitives importantes: le plongement de graphes ou leurs nœuds dans une représentation spatiale vectorielle continue (codage) et, inversement, la génération de graphes à partir de ces vecteurs en retour (décodage). Nous présentons les deux problèmes, leurs applications et leurs défis dans les deux sous-sections suivantes. Suivant la philosophie de DL qui consiste à abandonner les fonctions faites à la main pour leurs équivalens apprises. La principale question de notre recherche est: *Comment pouvons-nous transférer, adapter ou généraliser les concepts DL qui fonctionnent bien sur les données séquentielles et les données d'image aux graphes?* Néanmoins, comme le domaine est plutôt jeune, nous pensons que de nombreuses approches traditionnelles seront adaptées avec succès au DL en les rendant différentiables.

Plongement de graphes et leurs nœuds. L'une des tâches centrales abordées par DL est celle de l'apprentissage de la représentation (Bengio et al., 2013), c'est-à-dire trouver une correspondance entre les entrées bruts et un espace vectoriel continu à dimensionnalité fixe. L'objectif est d'optimiser cet application afin que les propriétés d'objets d'entrée pertinentes aux tâches soient préservées et reflétées dans les relations géométriques dans l'espace de plongement appris. Par exemple, dans le cas de la classification des chiffres écrites, l'application devrait enregistrer les propriétés concernant le type de chiffre, être invariante par rapport à celles concernant le style d'écriture et s'efforcer de rendre les plongements des images appartenant aux différentes classes séparable par hyperplan.

Dans le cas des graphes, nous nous intéressons principalement à la représentation des nœuds individuels ainsi qu'à celle des graphes entiers. Habituellement, l'information provenant du voisinage local d'un nœud est prise en compte dans son plongement, tandis que leplongement des graphes est ensuite calculée comme une forme d'agrégation (pooling) des plongements de nœuds. L'analogie dans l'apprentissage profond sur les images est, respectivement, le comptage des caractéristiques en pixels (par exemple pour la segmentation sémantique des images) et des caractéristiques en images (par exemple pour la reconnaissance des images). Dans le cas des graphes, deux types d'informations

sont disponibles. La première est la structure du graphe définie par des nœuds et des arêtes. Le deuxième sont les données associées au graphe, c'est-à-dire les attributs des noeuds et des arêtes. On peut soutenir qu'un puissant réseau de plongement devrait être capable d'exploiter autant d'informations que possible.

La possibilité de calculer les plongements de nœuds a donné lieu à de nombreuses applications, telles que la prédiction de liens (Schlichtkrull et al., 2017), la classification (semi-supervisée) dans les réseaux de citation (Kipf and Welling, 2016a), la réalisation de tâches de raisonnement logique (Li et al., 2016b), la recherche des correspondances entre meshes (Monti et al., 2017) ou de segmentation des points de nuage (chapitre 4). Le plongement de graphes a également été utile dans de nombreuses tâches, par exemple pour mesurer la similarité des réseaux cérébraux par apprentissage métrique (Ktena et al., 2017), suggérer des mouvements heuristiques pour approximer les problèmes NP-durs (Dai et al., 2017), prévoir la satisfaction des propriétés formelles des programmes (Li et al., 2016b), classifier les effets chimiques des molécules (chapitre 3) et la régression leurs propriétés physiques (Gilmer et al., 2017), ou classification des nuages de points (chapitre 3).

La pierre angulaire des architectures de réseau populaires pour le traitement des images naturelles, de la vidéo ou de la parole est la couche convolutionnelle (LeCun et al., 1998). L'invariance à translation, l'utilisation de filtres avec un support compact et l'application sur plusieurs résolutions s'adaptent très bien aux propriétés statistiques de ces données: la stationnarité, la localisation et la compositionnalité.

En plus du fait que l'architecture impose un prieur particulièrement adapté aux images naturelles (Ulyanov et al., 2017), un autre avantage majeur est la forte liaison des paramètres induite (partagement des poids) sur la grille, qui réduit considérablement le nombre de paramètres libres dans le réseau par rapport à une couche entièrement connectée (perceptron) sans aucun sacrifice en capacité expressive et permet de traiter des entrées à dimensions variables (Long et al., 2015).

Supposant que les principes de stationnarité, de localité et éventuellement de composition de la représentation s'appliquent également aux données de graphes, il est utile de considérer une architecture hiérarchique de type CNN pour leur traitement afin de bénéficier des avantages décrits ci-dessus.

Génération de graphes. Les modèles génératifs basés sur l'apprentissage profond ont gagné en popularité massive au cours des dernières années, en particulier dans le cas des images et des textes. L'idée principale est de collecter une grande quantité de données dans un domaine donné, puis de former un modèle pour générer des données

similaires, c'est-à-dire échantillonner d'une distribution apprise qui se rapproche de la distribution réelle des données. Comme le modèle a beaucoup moins de paramètres que la taille de l'ensemble de données, il est forcé de découvrir "l'essence" des données plutôt que de les mémoriser. Habituellement, le processus de génération (aussi appelé décodage) est conditionné par un vecteur aléatoire et/ou un point d'un espace vectoriel défini, tel que l'espace de plongement d'un codeur.

Dans le cas des graphes, nous sommes intéressés à générer à la fois la structure et les attributs associés. Il existe de nombreuses applications d'un tel générateur de graphes, par exemple créer des graphes similaires (Bojchevski et al., 2018) ou maintenir des représentations intermédiaires dans les tâches de raisonnement (Johnson, 2017). Dans une configuration encodeur-décodeur, tout type d'entrée peut être encodé dans un espace de plongement latent puis traduit en graphe. Parmi les applications possibles, on peut citer la découverte de médicaments par optimisation continue de certaines propriétés chimiques dans l'espace latent (Gómez-Bombarelli et al., 2016), l'échantillonnage de molécules ayant certaines propriétés (chapitre 5) ou correspondant à des mesures de laboratoire (soit par conditionnement soit par traduction), ou simplement la pré-apprentissage de codeurs dans le cas de données annotées sont chères.

Résumé des contributions. Après avoir tracé les deux grandes orientations de cette thèse, nous présentons un résumé de nos contributions.

- Nous proposons Edge-Conditioned Convolution (ECC) dans le chapitre 3, une nouvelle opération de type convolution sur des graphes exécuté dans le domaine spatial où les filtres sont conditionnés par des attributs des arêtes (discret ou continu) et générés dynamiquement pour chaque graphe spécifique en entrée. Cela permet à l'algorithme d'exploiter suffisamment d'informations structurelles dans les voisinages locaux. Notre formulation permet de généraliser la convolution discrète sur les grilles. En raison de son application dans le domaine spatial, la méthode peut travailler sur des graphes avec une structure arbitraire et variable. Dans le chapitre 4, nous intégrons l'ECC dans un réseau récurrent et réduisons ses besoins en mémoire et en computation, en le reformulant comme ECC-VV.

- Nous présentons SuperPoint Graph (SPG) dans chapitre 4, une nouvelle représentation de nuages de points avec des fonctions des arêtes riches codant la relation contextuelle entre les parties d'objet. Sur la base de cette représentation, nous sommes en mesure d'appliquer l'apprentissage profond sous la forme d'ECC/ECC-VV sur des nuages de points à grande échelle sans sacrifice majeur dans les détails fins.

- Nous démontrons les multiples applications de l'ECC/ECC-VV. Le chapitre 3 examine les applications de classification des graphes, en particulier pour les graphes représentant des composés chimiques et pour les graphes de voisinage sur nuages de points. De plus, nous évaluons la performance sur la classification des nœuds dans le contexte de la segmentation sémantique des scènes à grande échelle au chapitre 4. ECC est également utilisé comme codeur dans l'autoencodeur moléculaire présenté au chapitre 5. En plus d'avoir obtenu l'état de l'art en performance sur plusieurs ensembles de données parmi les méthodes DL aux moments respectifs de publication (NCI1 (Wale et al., 2008), Sydney Urban Objects (De Deuge et al., 2013), Semantic3D (Hackel et al., 2017) et S3DIS (Armeni et al., 2016)), nous étions les premiers à appliquer des convolutions sur graphes pour nuages de points, avec la motivation de préservation de creux et la précision dans le traitement des détails.

- Nous proposons un décodeur de graphes formulé dans le cadre des auto-encodeurs variationnels (Kingma and Welling, 2013) au chapitre 5. Le décodeur produit directement un graphe probabiliste entièrement connecté d'une taille maximale prédéfinie. C'évite quelque peu les problèmes de discrétisation, et utilise la correspondance des graphes pendant l'apprentisage afin de tenter de relever le défi du classement indéfini de nœuds. Nous évaluons sur la tâche difficile de la génération de molécules.

Publication List

Simonovsky, M. and Komodakis, N. (2016). OnionNet: Sharing Features in Cascaded Deep Classifiers. In *Proceedings of the British Machine Vision Conference (BMVC)*.

Simonovsky, M., Gutiérrez-Becker, B., Mateus, D., Navab, N., and Komodakis, N. (2016). A Deep Metric for Multimodal Registration. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*.

Simonovsky, M. and Komodakis, N. (2017). Dynamic Edge-conditioned Filters in Convolutional Neural Networks on Graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Landrieu*, L. and Simonovsky*, M. (2018). Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Simonovsky, M. and Komodakis, N. (2018). Towards Variational Generation of Small Graphs. In *Sixth International Conference on Learning Representations (ICLR), workshop track*.

Simonovsky, M. and Komodakis, N. (2018). GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. In *27th International Conference on Artificial Neural Networks (ICANN)*.

* Joint first authorship

Acknowledgements

Foremost, I would like to express gratitude to my supervisor Nikos Komodakis for accepting me as his student and securing funding for the whole period, for useful discourses and insights, and especially for giving me the trust and freedom to pursue the research directions I personally found the most interesting and promising. I'm also grateful to the rest of my committee for devoting their precious time to reviewing this thesis and for coming up with many great, thought-provoking questions.

I have experienced very friendly and relaxed atmosphere in my research group, Imagine. In particular, I'm obliged to Renaud Marlet for his warm attitude, negotiating difficult issues and, with the strong support of Brigitte Mondou, for shielding the lab from the outside world. I thank Guillaume Obozinski for his research attitude and didactics, as well as the organization of reading groups, together with Mathieu Aubry. I was also fortunate enough to collaborate on interesting problems with external researchers who taught me a lot along the way. Thank you, Loïc Landrieu, Benjamín Gutiérrez-Becker, Martin Čadík, Jan Brejcha, and Shinjae Yoo.

Uprooted and replanted to France, I was blessed with amazing labmates, many of whom have become good friends of mine. Thank you for all the fun, chats, help, ideas, as well as putting up with GPU exploitation, Benjamin Dubois, Francisco Massa, Laura Fernández Julià, Marina Vinyes, Martin De La Gorce, Mateusz Koziński, Pierre-Alain Langlois, Praveer Singh, Raghudeep Gadde, Sergey Zagoruyko, Shell Hu, Spyros Gidaris, Thibault Groueix, Xuchong Qiu, and Yohann Salaun, as well as Maria Vakalopoulou and Norbert Bus. The past three years would have been lonely and boring without you! Professionally, I'm especially grateful to Sergey, who bootstrapped me into the engineering side of deep learning and taught me lots of good practice, to Francisco for stimulating discussions on everything, and to Shell for keeping me in touch with the theory.

Furthermore, I'm indebted to the community behind Torch, PyTorch, and the whole Python universe. Without all of you, I would still be implementing my first paper now.

A very special thanks goes to my dear parents, Dana and Vašek, for their strong belief in education and for their unconditional support through my whole studies; to you I dedicate this thesis. Last but certainly not least, I would like to thank my partner Beata for her patience and tolerance, for enduring our long long-distance relationship, as well as for all the love, moral support, and fun.

Table of contents

Chapter 1

Introduction

1.1 Motivation

A graph is a powerful representation of relations between pairs of entities. Its versatility and strong theoretical understanding has resulted in a plethora of use cases across a variety of science disciplines and engineering problems. For example, graphs can naturally be used to describe the structure of chemical compounds, the interactions between regions in the brain, social interactions between people, the topology of three-dimensional models and transportation plans, dependencies of linked web pages, program flows, or knowledge bases. The development of the theory and algorithms to handle graphs has therefore been of major interest.

Formally, a (directed) graph is an ordered pair $G = (\mathcal{V}, \mathcal{E})$ such that $\mathcal{V}$ is a non-empty set of vertices (also called nodes) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. Additional information can be attached to both vertices and edges in the form of attributes. A vertex-attributed graph assumes function $\mathcal{V} \to \mathbb{R}^{d_v}$ assigning attributes to each vertex. An edge-attributed graph assumes function $\mathcal{E} \to \mathbb{R}^{d_e}$ assigning attributes to each edge.

The classical graph theory, foundations of which date back to the 18th century, focuses on understanding and analyzing the graph structure and addressing combinatorial problems such as finding mappings, paths, flows or colorings. On the other hand, there has been a lot of research on graph-structured data in the past decades, especially in the signal processing and machine learning communities. In the former, the research has notably brought Fourier transform to irregular domains, giving rise to spectral graph theory and adapting fundamental signal processing operations such as filtering, translation, dilation, or downsampling (Shuman et al., 2013). Usually, a fixed graph is considered with changing data on its nodes. In machine learning, researchers have explicitly made use of graph structures in the data in areas such as graph partitioning

for clustering, label propagation for semi-supervised learning, manifold methods for dimensionality reduction, graph kernels to describe similarity between different graphs, or graphical models for capturing probabilistic interpretation of data.

In recent years, deep learning (DL) has achieved significant breakthroughs in quantitative performance over traditional machine learning approaches in a broad variety of tasks and the field has matured into technologies useful for commercial applications. Prominently, a specific class of architecture, called Convolutional Neural Networks (CNNs), has gained massive popularity in tasks where the underlying data representation has a grid structure, such as in speech processing and natural language understanding (1D, temporal convolutions), in image classification and segmentation (2D, spatial convolutions), and in video and medical image understanding (3D, volumetric convolutions) (LeCun et al., 2015). On the other hand, the case of data lying on irregular or generally non-Euclidean domains has received comparatively much less attention in the community but has become a hot topic during the preparation of this thesis (Bronstein et al., 2017).

Encouraged by the success of DL on grids, in this thesis we study DL architectures for graph-structured data. Specifically, we concentrate on two important primitives: embedding graphs or their nodes into a continuous vector space representation (encoding) and, conversely, generating graphs from such vectors back (decoding). We introduce both problems, their applications and challenges in the following two subsections. Following the DL philosophy of abandoning hand-crafted features for their learned counterparts, our contributions pragmatically build more on accomplished DL methods rather than on past approaches from the machine learning and signal processing community amendable to differentiable re-formulation. The main research question we ask is: *How can we transfer, adapt or generalize DL concepts working well on sequential and image data to graphs?*

1.1.1 Embedding Graphs and Their Nodes

One of the central tasks addressed by DL is that of representation learning (Bengio et al., 2013), *i.e.* finding a mapping from raw input objects to a continuous vector space of fixed dimensionality $\mathbb{R}^d$. Both the mapping and the vector space are also interchangeably called embedding or encoding in the literature. The goal is to optimize this mapping so that task-relevant object properties are preserved and reflected in the geometric relationships within the learned embedding space. As an example, in the task of hand-written digit classification, the mapping should capture properties regarding the type of digit, be invariant to those concerning writing style, and strive to make embeddings of images belonging to different classes separable by hyperplanes.

In the case of graphs, we are primarily interested in finding representation of individual nodes as well as of the entire graphs. Usually, information from the local neighborhood of a node is considered in its embedding, while graph embeddings are further computed as some form of aggregation (pooling) of node embeddings. The analogy in deep learning on images is computing pixel-wise features (*e.g.* for semantic image segmentation) and image-wise features (*e.g.* for image recognition), respectively. In the case of graphs, there are two kinds of information available. The first is the graph structure as defined by the nodes and edges. The second is the data associated to the graph, *i.e.* its node and edge attributes. Arguably, a powerful embedding network should be able to exploit as much information as possible.

The ability to compute node embeddings has given rise to many application, such as link prediction (Schlichtkrull et al., 2017), (semi-supervised) node classification in citation networks (Kipf and Welling, 2016a), performing logical reasoning tasks (Li et al., 2016b), finding correspondences across meshes (Monti et al., 2017) or point cloud segmentation (Chapter 4). Graph embeddings has also been useful in many tasks, for example measuring similarity of brain networks by metric learning (Ktena et al., 2017), suggesting heuristic moves for approximating NP-hard problems (Dai et al., 2017), predicting satisfaction of formal properties in computer programs (Li et al., 2016b), classifying chemical effects of molecules (Chapter 3) and regressing their physical properties (Gilmer et al., 2017), or point cloud classification (Chapter 3).

Challenges

The cornerstone of popular network architectures for processing natural images, video or speech is the convolutional layer (LeCun et al., 1998). The translation invariance of the operation, the use of filters with compact support and the application over multiple resolutions fit very well to the statistical properties of such data, namely the stationarity, locality and compositionality.

Besides the fact that the architecture imposes a prior especially suitable for natural images (Ulyanov et al., 2017), another major benefit is the induced strong tying of parameters (weight sharing) across the grid, which greatly reduces the number of free parameters in the network compared to a fully-connected layer (perceptron) while still being able to capture the statistics of real data and enables processing of variable-sized inputs in so-called fully-convolutional style (Long et al., 2015).

Assuming that the stationarity, locality and possibly compositionality principles of representation hold to at least some level in graph data as well, it is meaningful to consider a hierarchical CNN-like architecture for processing it in order to benefit from

advantages described above. However, the graph domain poses several challenges which make extensions of CNNs to graphs not straightforward. We detail the main challenges below.

Irregular Neighborhoods Unlike on grids, where each node has the same amount of neighbors except for boundaries, nodes in general graphs can have arbitrary number of adjacent nodes. This means that there is no trivial analogy of translation invariance and thus the research objective is to come up with strategies to share parameters of the convolution operator among its applications to different nodes.

Unordered Neighborhoods There is no specific ordering of neighbors of a node, as both nodes and edges are defined as sets. In order to do better that isotropic smoothing, the ability to assign different convolutional weights to different edges is desired. This requires making assumptions or exploiting additional information such as the global structure, degrees of nodes, or edge attributes. On contrary, grids imply a natural ordering of neighborhoods.

Structural Variability In practical scenarios, the graph structure may vary throughout the dataset or may remain fixed, with only the data on nodes changing. The latter is the usual case for applications stemming from the signal processing community, centered around spectral filtering methods (Shuman et al., 2013), and data-mining community, centered around matrix factorization methods (Hamilton et al., 2017a). However, as such methods cannot naturally handle datasets with varying graph structure (for instance meshes, molecules, or frequently updated large-scale graphs), spatial filtering methods have gained on popularity recently. The research question here is to devise explicit local propagation rules and build links to established past work.

Computational Complexity The current DL architectures on grids benefit from heavy parallelism on Graphics Processing Units (GPUs) enabled by well-engineering implementations in popular frameworks such as PyTorch (Paszke et al., 2017) or TensorFlow (Abadi et al., 2015). Unfortunately, the irregular structure of graphs is less amendable to parallelism. The research investigates implementation details such as the use of sparse matrices or other data structures. In addition, for methods based on spectral processing, the naive use of Graph Fourier transform has quadratic complexity in the number of nodes.

1.1.2 Generating Graphs

Deep learning-based generative models have gained massive popularity during several past years, especially in the case of images and text. The principal idea is to collect a large amount of (unannotated) data in some domain and then train a model to generate similar data, *i.e.* draw samples from a learned distribution closely approximating the true data distribution. As the model has much less parameters than the size of the dataset, it is forced to discover the "essence" of the data rather than to memorize it. Usually, the generative process (also called decoding) is conditioned on a random vector and/or a point from a defined vector space, such as the embedding space of an encoder.

In the case of graphs, we are interested in generating both the structure and associated attributes. There are many applications of such a graph generator, for example creating similar graphs (Bojchevski et al., 2018) or maintaining intermediate representations in reasoning tasks (Johnson, 2017). In an encoder-decoder setup, any kind of input can be encoded to a latent embedding space and then translated to a graph. Possible applications include drug discovery by continuous optimization of certain chemical properties in the latent space (Gómez-Bombarelli et al., 2016), sampling molecules with certain properties (Chapter 5) or corresponding to given laboratory measurements (either by conditioning or translation), or simply encoder pre-training in cases where labeled data is expensive.

Challenges

In addition to the challenges usually demonstrated in generative models on grids, such as tricky training (Bowman et al., 2016; Radford et al., 2015), problematic capturing of all distribution modes (Radford et al., 2015) or difficult quantitative evaluation (Theis et al., 2015), generative models for graph data face several more challenges, detailed below.

Unordered Nodes Data on regular grids provide an implicit ordering of nodes (that is of pixels, characters or words) that is amendable both to step by step generation with autoregressive processes (Bowman et al., 2016; van den Oord et al., 2016) and generation of entire content in a feed-forward network (Radford et al., 2015). However, nodes in graphs are defined as a set and so there is often no clear way how to linearize the construction in a sequence of steps for training autoregressive generators or order nodes in a graph adjacency matrix for generation by a feed-forward network. While there are practical algorithms for approximate graph canonization, *i.e.* finding consistent node ordering, available (McKay and Piperno, 2014), the empirical result of Vinyals et al.

(2015) that the linearization order matters when learning on sets suggest that it is a priori unclear that enforcing a specific canonical ordering would lead to the best results.

Discrete Nature Similar to text and unlike images, graphs are discrete structures. Incremental construction with autoregressive methods involves discrete decisions, which are not differentiable and thus problematic for gradient-based optimization methods common in DL. In sequence generation, this is typically circumvented by a maximum likelihood objective with so-called teacher forcing (Williams and Zipser, 1989), which allows to decompose the generation into a sequence of conditional decisions and maximize the likelihood of each of them. However, the choice of such a decomposition for graphs is not clear, as we argue above, and teacher forcing is prone the inability to fix its mistakes, resulting in possibly poor prediction performance if the conditioning context diverges from sequences seen during training (Bengio et al., 2015). On the other hand, generation of graphs in a probabilistic formulation can only postpone dealing with non-differentiablity issues to the final step, when the discretized graph has to be output.

Large Graphs The level to which the locality and compositionality principles apply to a particular graph can strongly vary due to possibly arbitrary connectivity - for example, a generator should be able to output both a path and a complete graph. This seems problematic for scaling up to large graphs, as one cannot trivially enforce some form of hierarchical decomposition or coarse-to-fine construction, conceptually similar to using up-sampling operators in images (Radford et al., 2015). This may bring about the necessity to use more parameters while having less regularization implied by the architecture design in the case a feed-forward network. For autoregressive methods, large graphs may make training difficult due to very long chains of construction steps, especially for denser graphs.

1.2 Summary of Contributions

Having charted the two main directions of this thesis and their challenges, we summarize our contributions in this section. Most of these contributions were published as "Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs" at CVPR (Simonovsky and Komodakis, 2017), as "Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs" at CVPR (Landrieu and Simonovsky, 2018), as "Towards Variational Generation of Small Graphs" at ICLR workshop track (Simonovsky and Komodakis, 2018b), and as "GraphVAE: Towards Generation of Small Graphs Using

Variational Autoencoders" at ICANN (Simonovsky and Komodakis, 2018a). Martin Simonovsky is the leading author in all publications with the exception of Landrieu and Simonovsky (2018), where the contribution is equally shared with Loïc Landrieu.

- We propose Edge-Conditioned Convolution (ECC) in Chapter 3, a novel convolution-like operation on graphs performed in the spatial domain where filters are conditioned on edge attributes (discrete or continuous) and dynamically generated for each specific input graph. This allows the algorithm to exploit enough structural information in local neighborhoods so that our formulation can be shown to generalize discrete convolution on grids. Due the application in spatial domain, the method can work on graphs with arbitrary and varying structure. In Chapter 4, we integrate ECC into a recurrent network and reduce its memory and computational requirements, reformulating it as ECC-VV.

- We introduce SuperPoint Graph (SPG) in Chapter 4, a novel point cloud representation with rich edge attributes encoding the contextual relationship between object parts. Based on this representation, we are able to apply deep learning in the form of ECC/ECC-VV on large-scale point clouds without major sacrifice in fine details.

- We demonstrate multiple applications of ECC/ECC-VV. Chapter 3 investigates graph classification applications, in particular for graphs representing chemical compounds and for neighborhood graphs of point clouds. Further, we evaluate the performance on node classification in the context of semantic segmentation of large-scale scenes in Chapter 4. ECC is also used as encoder in the molecular autoencoder presented in Chapter 5. Besides having obtained the state of the art performance on several datasets among DL methods at the respective times of publication (NCI1 (Wale et al., 2008), Sydney Urban Objects (De Deuge et al., 2013), Semantic3D (Hackel et al., 2017) and S3DIS (Armeni et al., 2016)), we were the first to apply graph convolutions to point cloud processing, with the motivation of preserving sparsity and presumably fine details.

- We propose a graph decoder formulated in the framework of variational autoencoders (Kingma and Welling, 2013) in Chapter 5. The decoder outputs a probabilistic fully-connected graph of a predefined maximum size directly at once, which somewhat sidesteps the issues of discretization, and uses graph matching during training in order to attempt to overcome the challenge of undefined node ordering. We evaluate on the difficult task of molecule generation.

Besides the central topic of deep learning on graphs, we have conducted research along several other lines within the duration of the thesis. These activities have resulted in two publications, which are not part of this manuscript and which we therefore briefly summarize below:

- In the domain of medical image registration, we proposed a patch similarity metric based on CNNs and applied it to registering volumetric images of different modalities. The key insight was to exploit the differentiability of CNNs and directly backpropagate through the trained network to update transformation parameters within a continuous optimization framework. The training was formulated as a classification task, where the goal was to discriminate between aligned and misaligned patches. The metric was validated on intersubject deformable registration on a dataset different from the one used for training, demonstrating good generalization. In this task, we outperformed mutual information by a significant margin. The work was done in collaboration with Benjamín Gutiérrez-Becker from TU Munich as a second author and was published as "A Deep Metric for Multimodal Registration" at MICCAI 2016 (Simonovsky et al., 2016).

- For retrieval scenarios in computer vision, we proposed a way of speeding up evaluation of deep neural networks by replacing a monolithic network with a cascade of feature-sharing classifiers. The key feature was to allow subsequent stages to add both new layers as well as new feature channels to the previous ones. Intermediate feature maps were thus shared among classifiers, preventing them from the necessity of being recomputed. As a result demonstrated in three applications (patch matching, object detection, and image retrieval), the cascade could operate significantly faster than both monolithic networks and traditional cascades without sharing at the cost of marginal decrease in precision. The work was published as "OnionNet: Sharing Features in Cascaded Deep Classifiers" at BMVC 2016 (Simonovsky and Komodakis, 2016).

1.3 Outline

In this section we summarize the chapters of the thesis and relate them to each other.

Chapter 2: Background and Related Work This chapter provides the background and overviews related and prior work on graph-structured data analysis. Both classic and deep learning-based methods are covered. Further discussions of related and contemporary work are also provided within each core chapter.

Chapter 3: Edge-Conditioned Convolutions This chapter introduces the main workhorse of this thesis, graph convolutions able to process continuous edge attributes. The method is applied to small-scale point cloud classification and biological graph datasets.

Chapter 4: Large-scale Point Cloud Segmentation This chapter applies ECC to the the problem of segmentation of large areas, such as streets or office blocks, by working on homogeneous segments instead of individual points.

Chapter 5: Generation of Small Graphs This chapter introduces a graph autoencoder based on ECC and graph matching with an application to drug discovery, where novel molecules need to be sampled.

Chapter 6: Conclusion This chapter reflects on the contributions of the thesis and hypothesizes on directions of future work.

Chapter 2

Background and Related Work

This chapter provides background and overview of related and prior work on (sub)graph embedding techniques. After introducing our notation in Section 2.1, we dive into classical (non-deep) signal processing on graphs in Section 2.2, which is mostly formulated in the spectral domain. This knowledge is then useful for the overview of spectral deep learning-based convolution methods, introduced in Section 2.3. We mention their shortcoming and move on to review the progress in spatial convolution methods in Section 2.4, also in relation to the contributions of this thesis. To make the picture complete, we briefly mention non-convolutional approaches for embedding, namely direct embedding and graph kernels, in Section 2.5.

While this thesis focuses exclusively on graphs, it is worth to mention the progress made in (deep) learning on manifolds. Whereas the fields of differential geometry and graph theory are relatively distant, both graphs and manifolds are examples of non-Euclidean domains and the approaches for (spectral) signal processing and for convolutions in particular are closely related, especially when considering discretized manifolds - meshes. We refer the interested reader to the nice survey paper of Bronstein et al. (2017) for concrete details.

2.1 Attributed Graphs

Let us repeat and extend graph-related definitions from the introduction. A directed graph is an ordered pair $G = (\mathcal{V}, \mathcal{E})$ such that $\mathcal{V}$ is a non-empty set of vertices (also called nodes) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges (also called links). Undirected graph can be defined by constraining the relation $\mathcal{E}$ to be symmetric.

It is frequently very useful to be able to attach additional information to elements of graphs in the form of attributes. For example, in graph representation of molecules,

vertices may be assigned atomic numbers while edges may be associated with information about the chemical bond between two atoms. Concretely, a vertex-attributed graph assumes function $F : \mathcal{V} \to \mathbb{R}^{d_v}$ assigning attributes to each vertex. An edge-attributed graph assumes function $E : \mathcal{E} \to \mathbb{R}^{d_e}$ assigning attributes to each edge. In the special case of non-negative one dimensional attributes $\mathcal{E} \to \mathbb{R}^+$, we call the graph weighted. Discrete attributes $\mathcal{E} \to \mathbb{N}$ can be also called types.

In addition to attributes, which are deemed being a fixed part of the input, we introduce mapping $H : \mathcal{V} \to \mathbb{R}^d$ to denote vertex representation due to signal processing or graph embedding algorithms operating on the graph. This mapping is called signal (especially if $d = 1$) or simply data. Often, vertex attributes F may be used to initialize H. The methods for computation of H will constitute a major part of discussion in this thesis.

Finally, assuming a particular node ordering, the graph can be conveniently described in matrix, resp. tensor form using its (weighed) adjacency matrix A, signal matrix H, node attribute matrix F and edge attribute tensor E of rank 3.

2.2 Signal Processing on Graphs

Here we briefly introduce several concepts regarding multiscale signal filtering on graphs, which are the basis of spectral graph convolution methods presented in Section 2.3 and also used for pooling in Chapter 3.

Mathematically, both a one-dimensional discrete-time signal with N samples as well as a signal on a graph with N nodes can be regarded as vectors $\mathbf{f} \in \mathbb{R}^N$. Nevertheless, applying classical signal processing techniques on the graph domain would yield suboptimal results due to disregarding (in)dependencies arising from the irregularity present in the domain.

However, the efforts to generalize basic signal processing primitives such as filtering or downsampling faces two major challenges, related to those mentioned in Section 1.1.1:

Translation While translating time signal $f(t)$ to the right by a unit can be expressed as $f(t-1)$, there is no analogy to shift-invariant translation in graphs, making the classical definition of convolution operation $(f * w)(t) := \int_\tau g(\tau)w(t-\tau)d\tau$ not directly applicable.

Coarsening While time signal decimation may amount to simply removing every other point, it is not immediately clear which nodes in a graph to remove and how to aggregate signal on the remaining ones.

In the following subsections, we describe three core concepts developed by the signal processing community: graph Fourier transform, graph filtering operations and graph coarsening operations. We refer to the excellent overview papers of Shuman et al. (2016, 2013) for a broader perspective as well as details.

2.2.1 Graph Fourier Transform

Assuming undirected finite weighted graph G on N nodes, let A be its weighted adjacency matrix and D its diagonal degree matrix, *i.e.* $D_{i,i} = \sum_j A_{i,j}$. The central concept in spectral graph analysis is the family of Laplacians, in particular the unnormalized graph Laplacian $L := D - A$ and the normalized graph Laplacian $L := D^{-1/2}(D - A)D^{-1/2} = I - D^{-1/2}AD^{-1/2}$. Intuitively, Laplacians capture the difference between the local average of a function around a node and the value of the function at the node itself. We denote $\{\mathbf{u}_l\}$ the set of real-valued orthonormal eigenvectors of a Laplacian and $\{\lambda_l\}$ the set of corresponding eigenvalues. Interestingly, Laplacian eigenvalues provide a notion of frequency in the sense that eigenvectors associated with small eigenvalues are smooth and vary slowly across the graph (*i.e.* , the values of an eigenvector at two nodes connected by an edge with large weight are likely to be similar) and those associated with larger eigenvalues oscillate more rapidly.

The analogy between graph Laplacian spectra and the set of complex exponentials, which are eigenfunctions of the classical Laplacian operator on Euclidean space and provide the basis for the classical Fourier transform, is the motivation for introducing an analogy to Fourier transform on graphs. The *graph Fourier transform* $\hat{\mathbf{f}}$ of signal $\mathbf{f} \in \mathbb{R}^N$ on the nodes of G is defined as the expansion in terms of Laplacian eigenvectors $\hat{f}(\lambda_l) :=< \mathbf{f}, \mathbf{u}_l >= \sum_{i=0}^{N-1} f(i)u_l(i)$ and the *inverse graph Fourier transform* is then computed as $f(i) = \sum_{l=0}^{N-1} \hat{f}(\lambda_l)u_l(i)$.

We remark that extensions to directed graphs are complicated (Sardellitti et al., 2017). Also, handling of discrete or multidimensional edge attributes is not supported by the framework, to the best of our knowledge, although one way of circumventing the latter might be to simultaneously consider multiple graphs with the same connectivity but different edge weights, each graph corresponding to a single real, non-negative attribute dimension.

2.2.2 Signal Filtering

Using graph Fourier transform, a signal can be equivalently represented in the spatial (node) domain and in the graph spectral domain. Thus, as in classical signal processing, the signal can be filtered in either of the domains.

Graph spatial filtering amounts to expressing the filtered signal g at node i as a linear combination of the input signal $\mathbf{f}$ at nodes within its K-hop local neighborhood $\mathcal{N}_K$: $g(i) = B_{i,i}f(i) + \sum_{j \in \mathcal{N}_K(i)} B_{i,j}f(j)$ with filter coefficients matrix B. While this is a (spatially) localized operation by design, it requires specifying $\mathcal{O}(N^2)$ coefficients unless some form of regularity is assumed.

Graph spectral filtering follows the classical case, where filtering corresponds to multiplication in the spectral domain, *i.e.* the amplification or attenuation of the set of basis functions. It is defined as $\hat{g}(\lambda_l) = \hat{f}(\lambda_l)\hat{w}(\lambda_l)$. This suggests a way of generalizing the convolution operation on graphs by the means of spectral filtering as

$$g(i) = (f *_G w)(i) := \sum_{l=0}^{N-1} \sum_{j=0}^{N-1} f(j)\hat{w}(\lambda_l)u_l(i)u_l(j) \tag{2.1}$$

It is important to remark that spectral filter coefficients $\hat{\mathbf{w}}$ are specific to the given graph, resp. the particular choice of its Laplacian eigenvectors.

Using the full range of N coefficients makes spectral filtering perfectly localized in the Fourier domain but not localized in the spatial domain due to uncertainty principle, its extension to graphs first developed by Agaskar and Lu (2013) but still considered an open problem. However, this property is usually undesired in practice. The popular remedy is to make the filter spectrum "smooth" by ordering the eigenvectors according to their eigenvalues and expressing coefficients as continuous function of only a few free parameters. To make the link between spectral and spatial filtering, it is interesting to parameterize the spectral filter as order K polynomial $\hat{w}(\lambda_l) = \sum_{k=0}^{K-1} a_k \lambda_l^k$ with coefficients $\mathbf{a}$. It can be shown that this construction corresponds to a spatial filter on K-hop neighborhood for a particular choice of $B_{i,j} := \sum_{k=0}^{K-1} a_k L_{i,j}^k$ (Shuman et al., 2013). Thus, polynomial spectral filters are exactly localized within a certain spatial neighborhood and have only a limited number of free parameters, independent of N, which is beneficial for any learning task, as shown by Defferrard et al. (2016).

Nevertheless, even in the localized case the cost of filtering a graph signal in the spectral domain is an $\mathcal{O}(N^2)$ operation due to the multiplications with Fourier basis in Equation 2.1. Fortunately, approximation methods have been developed for fast filtering in the case of sparse graphs, where the filter is formulated directly in the spatial domain as a polynomial in L that can be evaluated recursively, leading to computational

complexity linear with the number of edges. A popular choice for the polynomial is truncated Chebyshev expansion (Shuman et al., 2011). Note that this does not require an explicit computation of the Laplacian eigenbasis.

Building banks of filters localized both in space and frequency has been intensively studied in the context of graph wavelets and we refer to the review in (Shuman et al., 2013) for more details. Wavelet coefficients can serve as descriptors for classification or matching tasks in non-deep machine learning Masoumi and Hamza (2017).

2.2.3 Graph Coarsening

In the Euclidean world, multiresolution representation and processing of signals and images with techniques such as pyramids, wavelets or CNNs has been very successful. Finding a similar concept for graphs is therefore of interest. The task of *graph coarsening* is to form a series of successively coarser graphs $G^{(s)} = (\mathcal{V}^{(s)}, \mathcal{E}^{(s)})$ from the original graph $G = G^{(0)}$. Coarsening typically consists of three steps: subsampling or merging nodes, creating the new edge structure $E^{(s)}$ and weights (so-called *reduction*), and mapping the nodes in the original graph to those in the coarsened one with $C^{(s)} : \mathcal{V}^{(s-1)} \to \mathcal{V}^{(s)}$. In many practical cases, a desirable property of coarsening is to halve the number nodes at each level while clustering nodes connected with large weights together. For unweighted bipartite graphs, there is a natural way of coarsening by sampling "every other node". But the task is more complex for other cases, which has led to creation of various algorithms.

One line of work is based on partitioning the set of nodes into clusters and representing these with a single node in the coarsened graph. For example, Graclus (Dhillon et al., 2007) greedily merges pairs of nodes based on clustering heuristics such the normalized cut $A_{i,j}(1/D_{i,i}+1/D_{j,j})$, which is efficient but not optimal. The union of the neighborhoods of the two original nodes becomes the neighborhood of the merged node, leading to decreased sparsity of the coarsened graphs. This algorithm is used for pooling in Defferrard et al. (2016).

Another line of work keeps a strict subset of original nodes during the coarsening steps. For example, Shuman et al. (2016) select the nodes to keep based on the sign of entries in the eigenvector associated with the largest Laplacian eigenvalue (frequency), which can be computed efficiently by the power method. The sign is shown to behave intuitively on bipartite graphs, such as grids or trees. Kron reduction (Dörfler and Bullo, 2013) is suggested as a method of choice to compute the coarsened Laplacian, defned as the Schur complement of the original Laplacian with respect to the removed node indices. It offers many nice properties but leads to a decrease in sparsity, which can be amended by postprocessing with spectral sparsification (Spielman and Srivastava, 2011),

among others. This randomized algorithm samples edges according to the probability of the edge appearing in a random spanning tree of the graph, thus mostly removing those structurally unimportant edges. We use this pipeline for pooling operation later in Chapter 3.

Finally, let us remark that graph resolutions generated in this fashion are completely independent of any signals residing on the graph nodes.

2.3 Spectral Graph Convolutions

In this section, we review how the deep learning community have applied spectral signal processing methods, introduced above, to learning tasks. The characteristic feature of these methods is the explicit use of graph Laplacian in the convolution operation.

In their pioneering work, Bruna et al. (2013) learned filters using the formulation of spectral convolution in Equation 2.1. This can be conveniently expressed in matrix form as $\mathbf{h}^{t+1} = U\Theta U^T\mathbf{h}^t$, where U is the column-wise concatenation of Laplacian eigenvectors $\{\mathbf{u}_l\}$, Θ is a diagonal matrix of filter coefficients and $\mathbf{h}^t$ and $\mathbf{h}^{t+1}$ is the input and output signal vector, respectively. Each spectral filter Θ is parameterized as B-spline with the number of free parameters independent of graph size N to reduce the risk of overfitting and to make filters smooth and therefore localized in space. In addition, high frequency eigenvectors may be discarded for better computational efficiency of Fourier transform, which nevertheless remains quadratic in the number of eigenvectors. By using multiple filters per layer, stacking multiple convolutional layers and considering coarsened graphs created by agglomerative clustering, Bruna *et al.*were the first to present a graph analogue of classic CNNs.

Defferrard et al. (2016) later proposed an efficient spectrum-free method by formulating spectral filters as Chebyshev polynomials, as briefly introduced in Section 2.2.2 above. The filtering is evaluated directly in the spatial domain on a K-hop neighborhood and can be written in matrix notation as $\mathbf{h}^{t+1} = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})\mathbf{h}^t$ where θ is a vector of K parameters and T_k is Chebyshev polynomial of order k evaluated at the scaled Laplacian $\tilde{L} := 2L/\max\lambda - I$. Coarsening for pooling purposes is done by Graclus (Dhillon et al., 2007), which does not require computation of the spectrum either.

This approach was further simplified by Kipf and Welling (2016a), who proposed its first-order approximation in spatial domain as $\mathbf{h}^{t+1} = \theta(I - \hat{L})\mathbf{h}^t$ where θ is a scalar parameter and $\hat{L}$ is the Laplacian of the adjacency matrix with added self-connections $\hat{A} = A + I$. In effect, this model corresponds to simply taking average of neighboring nodes' signal weighted proportionally to the Laplacian.

Note that despite the spatial evaluation of spectral methods, model parameters are still tied to the particular Laplacian and the trained networks may not generalize to other graph structures; this has been empirically demonstrated by Monti et al. (2017) for the model of Defferrard et al. (2016). Yi et al. (2017) embarked on addressing this for the case of 3D shapes represented as meshes by mapping the spectral representation of signal on an input graph to a predefined canonical graph, in the context of which filters are learned. The mapping is a linear transformation predicted by a special network from the voxelized eigenbasis of the input mesh. The authors leverage duality in functional maps between spatial and spectral correspondences and use ground truth correspondences between shapes to initialize the training, which they described as extremely challenging.

Finally, let us recap that spectral methods have been shown to work on undirected graphs only, as the adaptation of Laplacian to directed graphs is not straightforward (Sardellitti et al., 2017). An interesting way of circumventing this was proposed by Monti et al. (2018), who exploit the concept of motifs (small subgraphs) to construct a symmetric motif-induced adjacency matrix based on a non-symmetric adjacency matrix by counting the number of times an edge takes part in a set of canonical motifs.

2.4 Spatial Graph Convolutions

We have seen above that while the spectral construction is theoretically well motivated, there are obstacles in going beyond a single undirected graph structure. The spatial construction, on the other hand, may not have these limitations, though its designs are perhaps less principled. This section overviews the development in spatial construction methods for graph convolution and is closely related to the contributions presented in Chapters 3 and 4.

Let $H^t \in \mathbb{R}^{N \times d^t}$ be d^t-dimensional signal matrix at N nodes. Spatial graph convolution amounts to expressing the output signal $H_i^{t+1} \in \mathbb{R}^{d^{t+1}}$ at node i as a function of the input signal $H_j^t \in \mathbb{R}^{d^t}$ at nodes j within its K-hop local neighborhood. Scarselli et al. (2009) proposed one of the first propagation models M, which (somewhat ironically) is arguably also the most general one: it may depend in an arbitrary way on the signals H, as well as edge and node attributes E and F within an arbitrary neighborhood $\mathcal{N}(i)$, formally written as

$$H_i^{t+1} = m(\{H_a^t | a \in \mathcal{N}(i) \cup \{i\}\}, \{F_a | a \in \mathcal{N}(i) \cup \{i\}\}, \{E_{a,b} | a, b \in \mathcal{N}(i) \cup \{i\}\}). \quad (2.2)$$

To account for the fact that neighboring nodes are typically unordered, the general form is then restricted to pairwise potentials with an aggregation over direct neighbors as

$$H_i^{t+1} = u(F_i, \sum_{j \in \mathcal{N}(i)} m(H_j^t, F_i, F_j, E_{j,i}). \tag{2.3}$$

The functions m and u are designed to be contractive operators, the whole network is treated as a dynamical system and solved for fixed point solution for H from any initial state in its forward pass[1]. No concept of graph-level output or graph coarsening was mentioned.

Recently, many graph convolution variants have appeared in the community. To make their comparison easier, Gilmer et al. (2017) suggested Message Passing Neural Network (MPNN) family fitting the vast majority of models of that time. It is a modified version of Scarselli's Equation 2.3, where a) node labels are used for initializing node state H^0 rather than for propagation and b) states of both the source and the destination node can be used for propagation. Concretely, Gilmer et al. (2017) use the message function m and the node update function u in the following way:

$$\begin{aligned} M_i^{t+1} &= \sum_{j \in \mathcal{N}(i)} m_t(H_i^t, H_j^t, E_{j,i}) \\ H_i^{t+1} &= u_t(H_i^t, M_i^{t+1}) \end{aligned} \tag{2.4}$$

These function define a recurrent neural network (RNN) where the state is processed over several iterations (time steps) $t = 1 \dots T$. At each iteration t, the update function takes its hidden state H_i^t and updates it with M_i^{t+1}, which is the aggregation of messages m_t incoming from each neighbor. Finally, a readout function is defined to aggregate final node states into a graph-level state. We will use this framework (without the readout function, for simplicity) to chronologically review interesting related works in the following. Note that the list is far from being exhaustive, as there has been an explosion of many very closely related or sometimes even identical models in the community recently.

Bruna et al. (2013) suggested a spatial method, in addition to his spectral model described in Section 2.3. Here, $m_t : W^{t,j,i} H_j^t$ and $u_t : \mathrm{ReLU}(M_i^{t+1})$ with an edge-specific matrices of parameters W. The disadvantage of no weight sharing is a large number of parameters and no generalization over different graphs.

[1] It would be a very interesting exercise to implement this approach in a current deep learning framework and benchmark it against contemporary RNN-based graph convolution methods. Li et al. (2016b) provide a theoretical insight that contraction is expected to attenuate the effect of long-range dependencies, though.

Duvenaud et al. (2015) proposed message passing function $m_t : H_j^t$ and update function $u_t : \tanh(W^{t,\deg(i)} M_i^{t+1})$ with node-degree specific matrices of parameters W. The signal is passed over edges indiscriminately (nevertheless, there are weak constraints for possible edge types and node degree in their specific context of molecular fingerprints).

Mou et al. (2016) proposed convolution operation on ordered trees, where the weights linearly depend on the position of a node within a triangular window containing the parent node and children, $m_t : (\eta_j^1 W^1 + \eta_j^2 W^2 + \eta_j^3 W^3) H_j^t + \mathbf{b}$ and $u_t : \tanh(M_i^{t+1})$, where η are barycentric coordinates and W, $\mathbf{b}$ are learnable parameters.

Li et al. (2016b) introduced support for discrete edge types and successful RNN layers, such as the Gated Recurrent Unit (Cho et al., 2014a), by having $m_t : W^{E_{j,i}} H_j^t$ and $u_t : \mathrm{GRU}(H_i^t, M_i^{t+1})$, where W are edge-type and edge-direction specific weight matrices.

Kearnes et al. (2016) developed a model where both nodes and edges hold hidden representations updated in an alternating fashion. Formally $m_t : E_{j,i}^t$, $u_t : \mathrm{ReLU}(W^2[\mathrm{ReLU}(W^1 H_i^t), M_i^{t+1}])$ and $E_{j,i}^{t+1} = \mathrm{ReLU}(W^5[\mathrm{ReLU}(W^3 E_{j,i}^t), \mathrm{ReLU}(W^4[H_i^t, H_j^t])])$ where $[\cdot]$ denotes concatenation and W weight matrices.

Battaglia et al. (2016) implemented the general version of MPNN in Equation 2.4 with m_t and u_t as multi-layer perceptrons. This stands out from most other approaches by not explicitly decomposing message passing into a form of multiplication of H_j^t with some weights. While the formulation has been proposed before Gilmer et al. (2017), it has been demonstrated only as a single convolutional layer on fully connected graphs with discrete edge attributes.

Schütt et al. (2017) replaced the usual matrix-vector multiplication with element-wise multiplication in lower-dimensional space, leading to message function $m_t : \tanh(W^1((W^2 H_j^t + \mathbf{b}^1) \odot (W^3 E_{j,i} + \mathbf{b}^2))$ and $u_t : H_i^t + M_i^{t+1}$ where W and $\mathbf{b}$ are learnable parameters. Later in (Schütt et al., 2017), the authors allowed for handling continuous edge attributes, leading to basically the same message function as in Gilmer et al. (2017).

Monti et al. (2017) , building on their previous work on learning on manifolds and meshes, proposed a Gaussian mixture model for message passing conditioned on continuous edge attributes. The model with K mixture components can be written as $m_t : w(E_{j,i}) H_j^t$ with edge weighting function w,

$w(\mathbf{x}) = \sum_{k=0}^{K} \mathbf{a}_k \exp(-0.5(\mathbf{x} - \boldsymbol{\mu}^k)^T \text{diag}(\boldsymbol{\sigma}^k)^{-1}(\mathbf{x} - \boldsymbol{\mu}^k))$ where $\mathbf{a}$, $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ are learnable parameters and $\text{diag}(\cdot)$ builds a diagonal matrix from a vector.

Gilmer et al. (2017) , the authors of MPNN, proposed message passing conditioned on continuous edge attributes using a neural network w outputting a edge-specific weight matrix, which is then used in $m_t : w(E_{j,i})H_j^t$. A GRU is used for maintaining node state, $u_t : \text{GRU}(H_i^t, M_i^{t+1})$.

Hamilton et al. (2017b) applied graph convolutions to unsupervised node embedding using a loss encouraging close nodes to have a similar embedding and distant ones otherwise. The concrete functions are $m_t : \text{ReLU}(W^1 H_j^t + \mathbf{b})$ and $u_t : \text{ReLU}(W^{t,2}[H_i^t, M_i^{t+1}])$ where W and $\mathbf{b}$ are learnable parameters. In addition, max is used instead of sum in Equation 2.4 The neighborhoods are subsampled to support large node degrees and edges are untyped.

Veličković et al. (2018) used self-attention mechanism to weigh contributions of neighbors using coefficients $\alpha_i = \text{softmax}_j(\mathbf{a}[WH_i^t, WH_j^t])$ in functions $m_t : \alpha_{ij} W H_j^t$ and $u_t : \text{ReLU}(M_i^{t+1})$, where W and $\mathbf{a}$ are learnable parameters and the neighborhood $\mathcal{N}(i)$ includes the central node i itself. Independently, Wang et al. (2018a) proposed further formulations of α_{ij} in the context of non-local image processing.

Wang et al. (2018b) proposed a message passing function formulated as $m_t : w(H_i^t, H_j^t - H_i^t)$ with a neural network w. While being less general than Battaglia et al. (2016), it forces the network to perceive H_j^t relatively to H_i^t, which is arguably advantageous if the embeddings also encode global information, such as position. Max aggregation is used instead of sum in Equation 2.4 and the update function is simply $u_t : M_i^{t+1}$.

Verma et al. (2018) used a variant of message passing $m_t : w(H_i^t, H_j^t)H_j^t$ with weights defined as a mixture of K matrices, $w(\mathbf{x}, \mathbf{y}) = \sum_{k=0}^{K} W^k \text{softmax}_j(\mathbf{x}^T \mathbf{a}^j + \mathbf{y}^T \mathbf{b}^j + \mathbf{c}^j)_k$, where W and $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are learnable parameters.

Our work also fits into MPNN framework. Specifically, Edge-Conditioned Convolutions (ECC), presented in Chapter 3, can be defined as $m_t : w^t(E_{j,i})H_j^t + \mathbf{b}^t$ and $u_t : \text{ReLU}(M_i^{t+1})$, where w^t is a neural network[2] and $\mathbf{b}^t$ a learnable bias. In addition, mean is used instead of sum in Equation 2.4 and the neighborhood $\mathcal{N}(i)$ includes the central node i itself. Several models listed above are put into relation to our formulation in more detail in Subsection 3.3.2. In Chapter 4, we further extend ECC with

[2]The message function is equivalent to that in Gilmer et al. (2017), which was a concurrent work.

element-wise multiplication and GRU input gating, defined as $m_t : w(E_{j,i}) \odot H_j^t$ and $u_t : \text{GRU}(H_i^t, (WH_i^t + \mathbf{b}) \odot M_i^{t+1})$, where W and $\mathbf{a}$ are learnable parameters.

Interestingly, the spatial representation of spectral methods can also be interpreted within MPNN framework, see Appendix in Gilmer et al. (2017) for details. For example, the equivalent formulation of Kipf and Welling (2016a) is $m_t : \hat{A}_{i,j}(\deg(i)\deg(j))^{-1/2}H_j^t$ and $u_t : \text{ReLU}(W^t M_i^{t+1})$.

Finally, let us note that there are also several works which do not easily fit the MPNN framework. Diffusion-convolutional NN (Atwood and Towsley, 2016) produces features by applying diffusion of different length, *i.e.* computes the average of features at all nodes weighted proportionally to transition probabilities given by powers of graph stochastic matrix $P = D^{-1}A$. Patchy-SAN (Niepert et al., 2016) linearizes selected graph neighborhoods by choosing a lexicographically maximal adjacency matrix so that a conventional 1D CNN can be used.

2.5 Non-convolutional Embedding Methods

Besides convolutional methods, there have been other approaches of learning the embeddings of nodes or parts of graphs. While these methods are not directly related to the contributions of this thesis, we find it interesting to provide a brief overview for completeness. In particular, we review direct node embedding and graph kernels in the following.

2.5.1 Direct Node Embedding

Direct methods define the i-th node embedding $\mathbf{z}_i$ to be a function of only the node's own attributes F_i or even its unique identity if no attributes are given. Note that this is different from the convolutional approach, where the embedding of a node is defined as the function of its surroundings as well. The main idea of the majority of direct embedding approaches is to use unsupervised objective functions encouraging the embedding $\mathbf{z}$ of "close" nodes to be similar, usually measured as their inner-product or Euclidean distance.

Early methods, *e.g.* Laplacian eigenmaps (Belkin and Niyogi, 2001), often approached the task as a dimensionality reduction problem. Given proximity matrix S describing the "closeness" between nodes, such as the (weighted) adjacency matrix, the optimization problem has the form of matrix factorization $||Z^T Z - S||$, where Z is the sought matrix of embeddings. In effect, the obtained embedding vectors just approximates some prescribed measure.

The advent of deep learning in natural language processing opened new ways for representation learning of discrete objects, such as words. Multiple successful techniques for unsupervised learning of word embeddings were based on co-occurence statistics, *i.e.* the assumption that similar words tend to appear in similar word neighborhoods (context). In particular, Skip-gram (Mikolov et al., 2013) aims to predict a word's embedding based on the embedding of its contextual words and vice versa.

This idea has been adapted to graphs in various ways of defining the context of a node. A popular way of establishing context is by using random walks on graph, pioneered by DeepWalk (Perozzi et al., 2014) and later improved on by node2vec with biased walks (Grover and Leskovec, 2016), allowing for smooth transition between depth-first and breath-first exploration. The similarity of embeddings approximates the stochastic transition matrix P by $e^{\mathbf{z}_i^T \mathbf{z}_j} / \sum_{v_k \in V} e^{\mathbf{z}_i^T \mathbf{z}_k} \approx P_{i,j}$, $P_{i,j}$ being the probability of visiting node j from i. The unsupervised loss function can combined with a supervised task objective, such as node classification.

Nevertheless, the vast majority of direct encoding approaches learn a unique embedding vector for each node individually, which leads to two major drawbacks: the number of parameters grows linearly with the size of the graph and it is not possible to generate embeddings for nodes not seen during training or generalize across graphs. In semi-supervised learning, this is called transductive learning, as opposed to inductive learning, which can deal with novel test nodes. One way of addressing this deficiency is to learn a network encoding node attributes instead of identities, as in Bojchevski and Günnemann (2017). The other way is to use spatial graph convolutions introduced in Section 2.4 above (Hamilton et al. (2017b) in particular), which may not be suitable for link prediction of yet unconnected new nodes, though.

2.5.2 Graph Kernels

Kernel methods (Schölkopf and Smola, 2002) are a family of established approaches based on measuring the similarity between two objects using an explicit kernel function $\mathcal{K}$ corresponding to the inner product in reproducing kernel Hilbert space $\mathcal{H}$. Such kernel is then used in place of inner products in various kernelized algorithms, notably Support Vector Machines.

A kernel between two graphs G and G' is given by $\mathcal{K}(G, G') := < \Phi(G), \Phi(G') >_{\mathcal{H}}$, where $\Phi(G)$ denotes a feature vector containing counts of a certain class of substructures in graph G and $\mathcal{H}$ is typically a Euclidean space. The challenge is to find substructure decompositions that captures the semantics of the graph while being computationally tractable to enumerate and evaluate, such as decompositions into graphlets (subgraphs

of limited size), subtrees, or random walks. The Weisfeiler-Lehman framework increased the efficiency of previous kernels by using a relabeling procedure and scaled to graphs with thousands of nodes (Shervashidze et al., 2011). Recently, Lei et al. (2017) cast the computation process of Weisfeiler-Lehman kernel as a (differentible) recurrent network.

Similarly to direct node embedding introduced above, there has been works exploiting Skip-gram for substructure embedding. Such embeddings can be used to improve graph kernels; Yanardag and Vishwanathan (2015) proposed not treating substructures as independent and introduced bilinear form $\Phi(G)^T M \Phi(G')$ called Deep graph kernels, where learnable matrix M captures the relationship between substructures and is computed from inner products of their individual embeddings. Subgraph2vec (Narayanan et al., 2016) later improved the way how the context of subgraphs in a graph should be modeled.

Chapter 3

Edge-Conditioned Convolutions

3.1 Introduction

Convolutional Neural Networks (CNNs) have gained massive popularity in tasks where the underlying data representation has a grid structure, such as in speech processing and natural language understanding (1D, temporal convolutions), in image classification and segmentation (2D, spatial convolutions), or in video parsing (3D, volumetric convolutions) (LeCun et al., 2015).

On the other hand, in many other tasks the data naturally lie on irregular or generally non-Euclidean domains, which can be structured as graphs in many cases. These include problems in 3D modeling, computational chemistry and biology, geospatial analysis, social networks, or natural language semantics and knowledge bases, to name a few. Assuming that the locality, stationarity, and composionality principles of representation hold to at least some level in the data, it is meaningful to consider a hierarchical CNN-like architecture for processing it.

However, a generalization of CNNs from grids to general graphs is not straightforward and has recently become a topic of increased interest. We identify that the formulations of graph convolution predating this work do not exploit continuous edge attributes, which results in an overly homogeneous view of local graph neighborhoods, with an effect similar to enforcing rotational invariance of filters in regular convolutions on images. Hence, in this work we propose a convolution operation which can make use of this information channel and show that it leads to an improved graph classification performance.

This novel formulation also opens up a broader range of applications; we concentrate here on point clouds specifically. Point clouds have been mostly ignored by deep learning before publication of the work presented in this chapter, their voxelization being the only trend (Huang and You, 2016; Maturana and Scherer, 2015; Qi et al., 2016). To

offer a competitive alternative with a different set of advantages and disadvantages, we construct graphs in Euclidean space from point clouds in this work and demonstrate state of the art performance on Sydney dataset of LiDAR scans (De Deuge et al., 2013).

This chapter is largely based on our CVPR 2017 publication (Simonovsky and Komodakis, 2017). Its contributions to the field at the time of submission are as follows:

- We formulate a convolution-like operation on graph signals 'performed in the spatial domain where filter weights are conditioned on edge attributes (discrete or continuous) and dynamically generated for each specific input sample. Our networks work on graphs with arbitrary varying structure throughout a dataset.
- We are the first to apply graph convolutions to point cloud processing. Our method outperforms volumetric approaches and attains the new state of the art performance on Sydney dataset, with the benefit of preserving sparsity and presumably fine details.
- We reach a competitive level of performance on graph classification benchmark NCI1 (Wale et al., 2008), outperforming other approaches based on deep learning there.

3.2 Related Work

3.2.1 Graph Convolutions

Here we put the contributions of this chapter in the context of related work and refer the reader to Chapter 2 for the review of the topic in general. Spectral methods (Bruna et al., 2013; Defferrard et al., 2016) offer a mathematically sound definition of convolution operator and learn filters in relation to the spectrum of graph Laplacian, which therefore has to be the same for all graphs in a dataset. This means that the graph structure is fixed and only the signal defined on the nodes may differ, precluding applications on problems where the graph structure varies in the dataset, such as meshes, point clouds, or diverse biochemical data.

To cover these important cases, we formulate our filtering approach in the spatial domain, where the limited complexity of evaluation and the localization property is provided by construction. The main challenge here is dealing with weight sharing among local neighborhoods, as the number of nodes adjacent to a particular vertex varies and their ordering is often not well definable.

Bruna et al. (2013) assumed fixed graph structure and did not share any weights among neighborhoods. Several works have independently dealt with this problem. Duvenaud et al. (2015) sum the signal over neighboring nodes followed by a weight matrix multiplication given by the degree of central node. Atwood and Towsley (2016) share weights based on the number of hops between two nodes. Kipf and Welling (2016a) further approximate the spectral method of Defferrard et al. (2016) and weaken the dependency on the Laplacian, but ultimately arrive at center-surround weighting of neighborhoods. None of these methods captures finer structure of the neighborhood and thus does not generalize regular convolution on grids. In contrast, here we show that methods using additional information in the form of edge attributes do offer such a convenient property. These includes the work of Li et al. (2016b) and Monti et al. (2017). Section 3.3.2 demonstrates that our method generalizes the above mentioned models. In a concurrent work, Gilmer et al. (2017) presented an equivalent formulation to our method. Finally, the approach of Niepert et al. (2016) introduces a heuristic for linearizing selected graph neighborhoods so that a conventional 1D CNN can be used. We share their goal of capturing structure in neighborhoods but approach it in a different way.

3.2.2 Deep Learning on Sparse 3D Data

Three-dimensional data can be captured and processed in various forms. Here we restrict ourselves to point clouds and polygonal meshes, which can be treated as volumes, sets or graphs for their processing.

Volumetric representation. Volumetric representation, which can be processed by regular convolutions on grid, was the first one to be used in deep learning-based 3D understanding methods and has enjoyed popularity since then. Medical images (Shen et al., 2017), distance fields (Li et al., 2016a) or uncertainties in output space (Wu et al., 2015b) are just a few examples of dense spatial data. Nevertheless, for simplicity and efficient processing in hardware, also sparse data may be voxelized into a dense volumes. In fact, prior to the submission of our work (2016), this was the only way of processing point clouds using deep learning, be it for classification (Maturana and Scherer, 2015; Qi et al., 2016) or segmentation (Huang and You, 2016) purposes. However, voxel grid representation of sparse data tends to be much more expensive in terms of memory and brings about discretization artifacts due to limited resolution or the need to resort to a sliding window approach.

To address this, Graham (2015) proposed a way of computing sparse convolution on lattices where the input is gathered into a dense matrix to allow for standard matrix multiplication. Unfortunately, the bookkeeping is performed on the CPU in their implementation and the evaluation has been fairly limited. In a work concurrent to ours, OctNet (Riegler et al., 2017b) and O-CNN (Wang et al., 2017) have expanded on the topic and replaced uniform grids by non-uniform octrees of a predefined depth. An octree is a spatial subdivision structure with adaptive cell size that represents empty space efficiently while allowing to store data in a linear array. However, unlike in Graham (2015) where the sparsity is decreased at each layer due to convolution dilating the number of active (non-zero) locations, octree structures are kept unmodified. While preserving sparsity, such a formulation of convolution has different behavior from the one on dense grids in how it propagates information across empty space [1]. Nevertheless, this does not seem to be a problem in practice and has not prevented octree-based methods from reaching state of the art results. This formulation of spatial convolution has also been named submanifold sparse convolutions by Graham et al. (2018). There, the authors revisit the algorithm of Graham (2015) and rely on strided convolutions to allow information to flow between disconnected components in the input, claiming better efficiency than OctNet. The necessity to know the grid subdivision structure or the submanifold ahead is inconvenient in reconstruction and shape generation applications. However, in later works (Riegler et al., 2017a; Tatarchenko et al., 2017), octrees were successfully used also in such applications by refining spatial subdivisions in a multi-resolution pipeline. Independently, Klokov and Lempitsky (2017) proposed hierarchical processing of point clouds according to the partitioning given by balanced kd-trees, where cells are merged by multilayer perceptrons. However, the distance metric induced by axis-aligned kd-trees is often rather different to the Euclidean one, demonstrated in the method's sensitivity to global rotations.

Set representation. A set is an intuitive representation for point clouds. Concurrently to our work, Qi et al. (2017a) was the first to propose a network operating directly on sets of points. Their key idea is to embed each point individually into high-dimensional latent space and then apply a permutation-invariant aggregation function (channel-wise maximum) to arrive at global descriptors, which can be again concatenated with point embeddings for point-wise prediction tasks. As the learned mapping is sensitive to global

[1] Interestingly, this is also true for all methods operating directly on point clouds and meshes mentioned further below: their convolutions add no new points or nodes to the domain. In fact, the computation of convolution on octree cells can be seen as a particular instance of a graph, which could be processed by our graph convolution method presented in this Chapter to give rise to the same output.

transformations, a spatial transformer network (Jaderberg et al., 2015) in the form of a smaller PointNet is used to roughly normalize the global coordinate system of input point clouds. While the architecture is remarkably simple, efficient and achieves great performance on a variety of tasks, Qi et al. (2017b) observed its limitations in fine-grained segmentation tasks due to its obliviousness to any concept of spatial closeness. Their follow-up work PointNet++ (Qi et al., 2017b) applies PointNet hierarchically on a nested partitioning of the input set, in reminiscence of strided convolution. While improving over PointNet, points in local point sets are still treated independently. Engelmann et al. (2017) later investigated incorporation of spatial context into PointNet by several ways of fusing embeddings of neighboring point cloud parts.

Graph representation. Unlike sets, graphs can model the context of spatial data explicitly. While neighborhood graphs constructed from point clouds express only geometric information, polygonal meshes additionally capture topological constraints (faces) and represent discretized manifolds (surfaces). We regard point cloud as graphs in Euclidean space in this work. Masci et al. (2015) are motivated by connections between meshes and manifolds and define convolution on spatially binned neighborhoods around every node in a mesh using geodesic distances. Their architecture contains only a single convolutional layer operating on combination of intrinsic spectral shape features. Follow-up work proposed a different local binning technique using anisotropic diffusion (Boscaini et al., 2016) and, concurrently to our work, using Gaussian mixture model (Monti et al., 2017). The latter, already mentioned above, can be regarded as a general graph convolution approach, though subsumed by our proposed model. Later, Wang et al. (2018b) introduced an architecture where edge attributes are defined as the difference of current node embeddings and the neighborhood graph is repeatedly dynamically rebuilt.

3.3 Method

We propose a method for performing convolutions over local graph neighborhoods exploiting edge attributes (Section 3.3.1) and show it to generalize regular convolutions (Section 3.3.2). Afterwards, we present deep networks with our convolution operator (Section 3.3.3) in the case of point clouds (Section 3.3.4) and general graphs (Section 3.3.5).

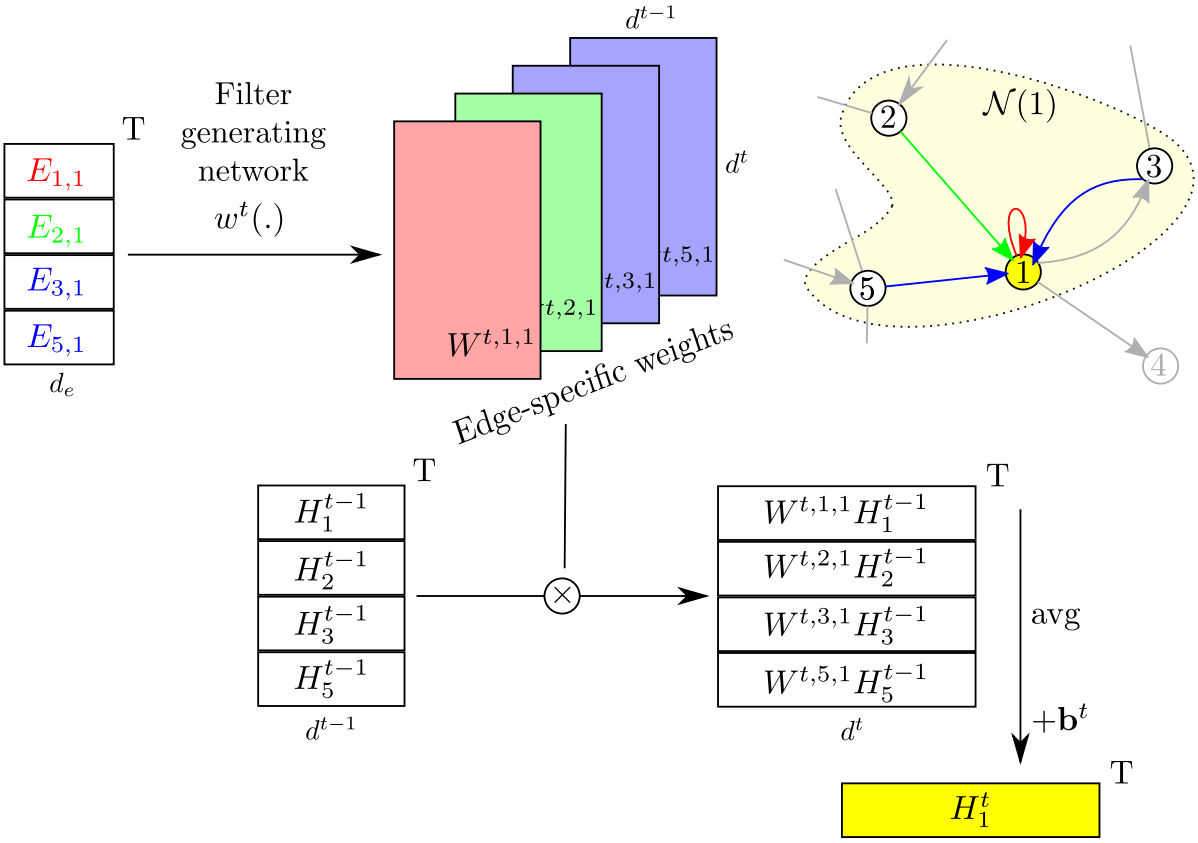


Fig. 3.1 Illustration of edge-conditioned convolution on a directed subgraph. Signal H_1^t on vertex 1 in the t-th network layer is computed as a weighted sum of signals H^{t-1} on the set of its predecessor nodes, assuming self-loops. The particular weight matrices W are dynamically generated by filter-generating network w^t based on the corresponding edge attributes E, visualized as colors.

3.3.1 Edge-Conditioned Convolution

Let us consider a directed graph $G = (\mathcal{V}, \mathcal{E})$ with edge attributes $E : \mathcal{E} \to \mathbb{R}^{d_e}$. Building on top of graph notation introduced in Section 2.1, let $H^t : \mathcal{V} \to \mathbb{R}^{d^t}$ denote the node representation (signal) computed at convolutional layer $t \in \{1, \ldots, T\}$ in a neural network. We make the input signal equal to node attributes $H^0 = F$ if these are available and set $H^0 = 0$ otherwise. The neighborhood $\mathcal{N}(i) = \{j; (j, i) \in \mathcal{E}\}$ of node i is defined to contain all its direct predecessors.

Our approach computes the filtered signal $H_i^t \in \mathbb{R}^{d^t}$ at node i as a weighted sum of its own signal $H_i^{t-1} \in \mathbb{R}^{d^{t-1}}$ and signals $H_j^{t-1} \in \mathbb{R}^{d^{t-1}}$ in its neighborhood, $j \in \mathcal{N}(i)$. While such a commutative aggregation solves the problem of undefined node ordering and varying neighborhood sizes, it also smooths out any structural information. To retain it,

we propose to condition each filtering weight on the respective edge attribute. To this end, we borrow the idea from Dynamic Filter Networks (Brabandere et al., 2016) and define filter-generating network $w^t : \mathbb{R}^{d_e} \to \mathbb{R}^{d^t \times d^{t-1}}$ with parameters θ^t which outputs an edge-specific weight matrix $W^{t,j,i} \in \mathbb{R}^{d^t \times d^{t-1}}$ given edge attributes $E_{j,i}$. See Figure 3.1 for an illustration.

The convolution operation, coined Edge-Conditioned Convolution (ECC), is formalized as follows:

$$\begin{aligned} H_i^{t+1} &= \text{ReLU}\left(\frac{1}{|\mathcal{N}(i) \cup \{i\}|} \sum_{j \in \mathcal{N}(i) \cup \{i\}} w^t(E_{j,i}; \theta^t) H_j^t + \mathbf{b}^t\right) \\ &= \text{ReLU}\left(\frac{1}{|\mathcal{N}(i) \cup \{i\}|} \sum_{j \in \mathcal{N}(i) \cup \{i\}} W^{t,j,i} H_j^t + \mathbf{b}^t\right) \end{aligned} \tag{3.1}$$

Equivalently, in the framework of Message Passing Neural Networks (see Section 2.4), ECC can be defined with message function $m_t : w^t(E_{j,i}; \theta^t) H_j^t + \mathbf{b}^t$, update function $u_t : \text{ReLU}(M_i^{t+1})$ and modified Equation 2.4 where mean is used instead of sum and the neighborhood includes the central node i itself.

It is important to understand that we do not learn convolution weights directly but rather a network which predicts them. In fact, it would be intractable to learn convolution weights for individual values of continuous attributes. Therefore, θ^t and $\mathbf{b}^t$ are learned model parameters updated with gradient descent during training and $W^{t,j,i}$ are dynamically predicted parameters for attributes of a particular edge in a particular input graph.

Parameters θ^t can be untied in each network layer t (such as in this chapter and Chapter 5) or shared across layers as in Chapter 4, where a more involved update function u_t is used to compensate for that.

The filter-generating networks w can be implemented using any differentiable architecture. As attributes are simple vectors in all our applications, we use multi-layer perceptrons (MLPs). In the case of structured attributes such as images or text (*e.g.* a free-form description of relations in a knowledge graph), CNNs or RNNs would be a more appropriate choice for w.

Note that ECC is defined on directed graphs. However, directed edges (i, j) without corresponding opposite edges (j, i) cause asymmetries in message passing. In the extreme case, start nodes in a directed acyclic graph would obtain no information about any other node. Therefore, we suggest making sure that both edge directions are present in input graphs and that their attributes differ so that message passing depends on the direction of that edge. For instance, it is reasonable to set $E_{i,j} = -E_{j,i}$ if attributes represent

offsets between points in spaces. Having $E_{i,j} = E_{j,i}$ is sufficient in the case of undirected graphs, which are supported by replacing each undirected edge $\{i, j\}$ with two directed ones (i, j) and (j, i).

Identity connections (ECC-id). Finally, we discuss the treatment of central node i. The formulation of ECC in Equation 3.1 effectively adds self-loops to the graph and does not treat i in any special way (other than consistent weights $W^{t,i,i}$ for all i if self-loops have a distinct attribute $E_{i,i}$), which is consistent with the definition of convolution on grids. However, the success of Residual Networks (He et al., 2016) is a strong motivation to consider adding identity skip-connections to the model and encourage ECC in learning residuals. In addition, moving the central node out of the aggregation makes its influence independent of the size of the neighborhood. We formulate ECC-id as follows:

$$H_i^{t+1} = \text{ReLU}\left(\text{id}(H_i^t) + \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} w(E_{j,i};\theta^t)H_j^t + \mathbf{b}^t\right) \tag{3.2}$$

where id() is an identity mapping if $d_t = d_{t-1}$ and a linear mapping otherwise. The equivalent update function in Message Passing framework is $u_t : \text{ReLU}(\text{id}(H_i^t) + M_i^{t+1})$.

Complexity. Computing H^t for all nodes requires at most[2] $|\mathcal{E}|$ evaluations of w and $|\mathcal{E}| + |\mathcal{V}|$ or $2|\mathcal{E}| + |\mathcal{V}|$ matrix-vector multiplications for directed, resp. undirected graphs. Both operations can be carried out efficiently on the GPU in batch-mode. Aggregation over neighborhood can be implemented as sparse-dense matrix multiplication, available in both PyTorch and TensorFlow frameworks. Therefore, the method can scale well to large sparse graphs both theoretically and in practice.

3.3.2 Relationship to Existing Formulations

Our formulation of convolution on graph neighborhoods retains the key properties of the standard convolution on regular grids that are useful in the context of CNNs: weight sharing and locality.

Importantly, the standard discrete convolution on grids is a special case of ECC, which we demonstrate in 1D for clarity. Consider an ordered set of nodes $\mathcal{V}$ forming a path graph (chain). To obtain convolution with a centered kernel of size d_e, we form $\mathcal{E}$ so that each node is connected to its d_e spatially nearest neighbors including self

[2]If edge attributes are represented by d_e discrete values in a particular graph and $d_e < |\mathcal{E}|$, w can be evaluated only d_e-times.

by a directed edge labeled with one-hot coding of the neighbor's discrete offset δ, see Figure 3.2. Taking w^t as a single linear layer without bias, we have $w(E_{j,i};\theta^t) = \theta^t_\delta$, where θ^t_δ denotes the respective reshaped column of the parameter matrix $\theta^t \in \mathbb{R}^{(d_t \times d_{t-1}) \times d_e}$. With a slight abuse of notation, we arrive at the equivalence to the standard convolution: $H^{t+1}_i = \mathrm{ReLU}(\sum_{j \in \mathcal{N}(i)} W^{t,j,i} H^t_j) = \mathrm{ReLU}(\sum_\delta \theta^t_\delta H^t_{i-\delta})$, ignoring the normalization factor $1/|\mathcal{N}(i) \cup \{i\}|$ playing a role only at grid boundaries.

This shows that ECC can retain the same number of parameters and computational complexity of the regular convolution in the case of grids. Note that such reduction is not possible with any of related work oblivious to edge attributes due to their way of weight tying. The weights in ECC are tied by edge attribute, which is in contrast to tying them by hop distance from a node (Atwood and Towsley, 2016), according to a neighborhood linearization heuristic (Niepert et al., 2016), by being the central node or not (Kipf and Welling, 2016a), by node degree (Duvenaud et al., 2015), or not at all (Bruna et al., 2013).

In fact, our definition of message passing function can be shown to generalize a number of prior graph convolution methods introduced in Section 2.4 by using a single linear layer without bias as w^t and defining attributes appropriately. The model of Bruna et al. (2013) can be recovered by assigning a unique one-hot code to every edge. Duvenaud et al. (2015) requires a one-hot code indicating the degree of target node $\deg(i)$ to every edge (j, i), while the model of (Kipf and Welling, 2016a) is obtained by setting $E_{i,j} = (\deg(i)\deg(j))^{-1/2}$. The support for discrete attributes in Li et al. (2016b); Schlichtkrull et al. (2017) can be obtained by using them directly in one-hot coding. The Gaussian mixture model of Monti et al. (2017) (concurrent work) is subsumed by our general neural network w^t by the universal approximation theorem (Hornik, 1991). Finally, the message passing function of Gilmer et al. (2017) (also concurrent work) is equivalent to our method.

Last, let us remark that the idea of exchanging messages among nodes is also the algorithmic basis of many inference techniques within the context of graphical models, such as belief propagation (Pearl, 1988). We postpone further discussion to Section 4.3.4 in the following Chapter, where we contrast message passing networks and conditional random fields.

3.3.3 Deep Networks with ECC

While ECC is in principle applicable to both node classification and graph classification tasks, in this chapter we restrict ourselves only to the latter one, *i.e.* predicting a class for the whole input graph. Hence, we follow the common architectural pattern for

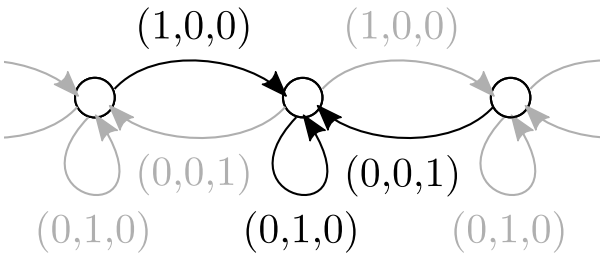


Fig. 3.2 Construction of a directed graph with one-hot edge attributes where the proposed edge-conditioned convolution is equivalent to the regular 1D convolution with a centered filter of size $d_e = 3$.

feed-forward networks of interlaced convolutions and poolings topped by global pooling and fully-connected layers, see Figure 3.3 for an illustration. This way, information from local neighborhoods gets combined over successive layers to gain context due to enlarged receptive field. While edge attributes are fixed for a particular graph, their (learned) interpretation by the means of filter generating networks w^t may change from layer to layer if the weights are untied. Therefore, the restriction of ECC to 1-hop neighborhoods $\mathcal{N}(i)$ is not a constraint, akin to using small 3×3 filters in exchange for deeper networks in CNNs on grids, which is known to be beneficial (He and Sun, 2015).

We use batch normalization (Ioffe and Szegedy, 2015) after each convolution, which was necessary for the learning to converge. Interestingly, we had no success with other feature normalization techniques such as data-dependent initialization (Mishkin and Matas, 2016) or layer normalization (Ba et al., 2016). On the other hand, explicitly assuring the convolution operation being a non-expansive operator (Scarselli et al., 2009) by applying activation function $\nu\tanh(w(\cdot))$ with $0 < \nu < 1$ can also make the network converge; the experimental performance was worse than with batch normalization, though.

Pooling. While (non-strided) convolutional layers and all point-wise layers do not change the underlying graph and only evolve the signal on nodes, pooling layers are defined to output aggregated signal on the nodes of a new, coarsened graph. Therefore, a pyramid of S progressively coarser graphs has to be constructed for each input graph. Let us extend here our notation with an additional scale superscript $s \in \{0, \dots, S\}$ to distinguish among different graphs $G^{(s)} = (\mathcal{V}^{(s)}, \mathcal{E}^{(s)})$ in the pyramid when necessary. Each $G^{(s)}$ has also its associated attributes $E^{(s)}$ and signal $H^{t,(s)}$.

As introduced in Section 2.2.3, a coarsening of $G^{(s-1)}$ typically consists of three steps: subsampling or merging nodes, creating the new edge structure $\mathcal{E}^{(s)}$ and labeling $E^{(s)}$ (so-called reduction), and mapping the nodes in the original graph to those in the

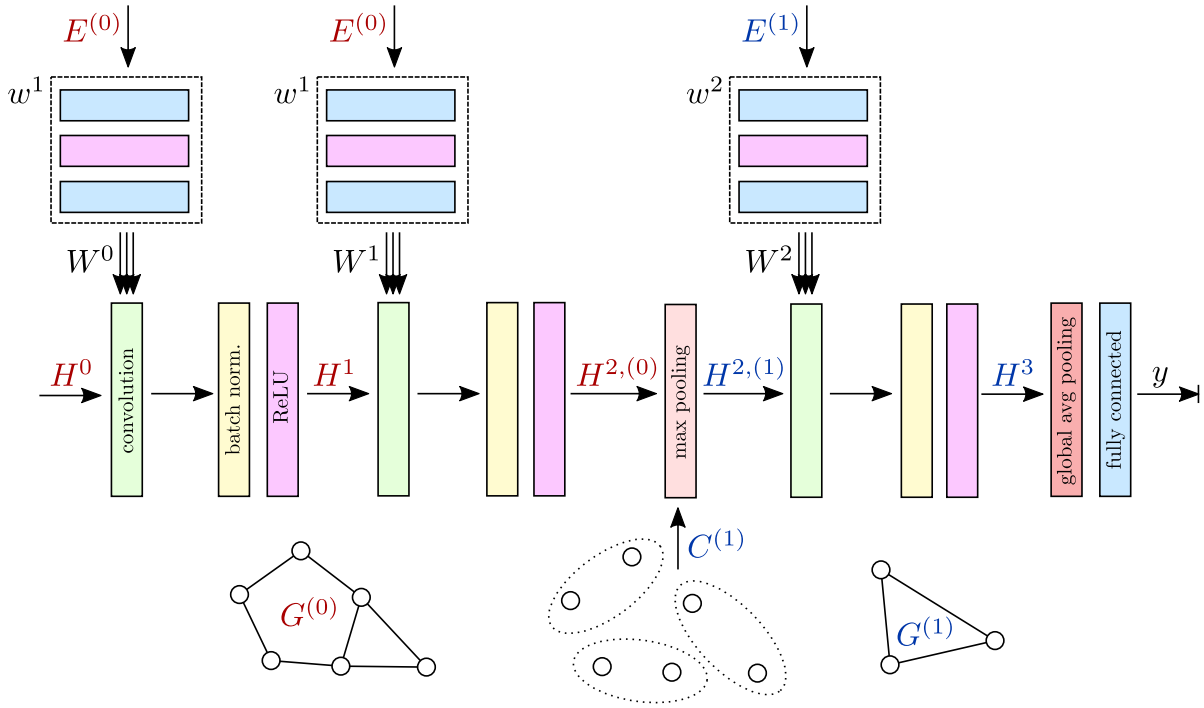


Fig. 3.3 Illustration of a deep network with three edge-conditioned convolutions and one pooling. The last convolution is executed on a structurally different graph $G^{(1)}$, which is related to the input graph $G^{(0)}$ by coarsening and signal aggregation in the max pooling step according to mapping $C^{(1)}$. See Section 3.3.3 for more details.

coarsened one with $C^{(s)} : \mathcal{V}^{(s-1)} \to \mathcal{V}^{(s)}$. We use a different algorithm depending on whether we work with general graphs or graphs in Euclidean space, therefore we postpone discussing the details to the applications. Finally, the s-th pooling layer aggregates $H^{t,(s-1)}$ into a lower dimensional $H^{t,(s)}$ based on $C^{(s)}$. See Figure 3.3 for an example of using the introduced notation.

During coarsening, a small graph may be reduced to several disconnected nodes in its lower resolutions without problems as our formulation of graph convolution always assumes a virtual self-edge. Since the architecture is designed to process graphs with variable number of edges and nodes, we deal with varying node count $|\mathcal{V}^{(S)}|$ in the lowest graph resolution by global average/max pooling.

3.3.4 Application in Point Clouds

Point clouds are an important 3D data modality arising from many acquisition techniques, such as laser scanning (LiDAR) or multi-view reconstruction. However, their natural

irregularity and sparsity present a challenge to the massive parallelism in GPU-based processing. In fact, at the submission time of our paper (Simonovsky and Komodakis, 2017), the only way of processing point clouds using deep learning has been to first voxelize them before feeding them to a 3D CNN, be it for classification (Maturana and Scherer, 2015) or segmentation (Huang and You, 2016) purposes. Such a dense representation is very hardware friendly and simple to handle with the current deep learning frameworks.

On the other hand, there are several disadvantages too. First, voxel representation tends to be much more expensive in terms of memory than usually sparse point clouds. Second, the necessity to fit them into a fixed size 3D grid brings about discretization artifacts and the loss of details and possibly metric scale. With the work presented in this chapter, we attempted at offering a competitive alternative to the mainstream by performing deep learning on point clouds directly. While we concentrate on point clouds in this thesis, we believe this work can also directly apply to meshes, as the graph structure is given and mesh downsampling is a well studied problem.

Graph Construction. Given a point cloud $\mathcal{P}$ with point positions P and features F (such as laser return intensity or color) we build a directed graph $G = (\mathcal{P}, \mathcal{E})$ by connecting each node i to all nodes j in its spatial neighborhood by a directed edge (j, i). In our experiments with neighborhoods, fixed metric radius ρ worked better than a fixed number of neighbors, likely due to more symmetric message passing and more homogeneous speed of information propagation. The node signal is set as $H^0 = F$ (or 0 if there are no features F). The offset $\delta = P_j - P_i$ between the points corresponding to nodes j, i is represented in Cartesian and spherical coordinates as 6D edge attribute vector $E_{j,i} = (\delta_x, \delta_y, \delta_z, ||\delta||, \arccos \delta_z / ||\delta||, \arctan \delta_y / \delta_x)$.

Graph Coarsening. For a single input point cloud $\mathcal{P}$, a pyramid of downsampled point clouds $\mathcal{P}^{(s)}$ is obtained by the VoxelGrid algorithm (Rusu and Cousins, 2011), which overlays a grid of resolution $r^{(s)}$ over the point cloud and replaces all points within a voxel with their centroid (and thus maintains subvoxel accuracy). Each of the resulting point clouds $\mathcal{P}^{(s)}$ is then independently converted into a graph $G^{(s)}$ and attributes $E^{(s)}$ with neighborhood radius $\rho^{(s)}$ as described above. The pooling map $C^{(s)}$ is defined so that each point in $\mathcal{P}^{(s-1)}$ is assigned to its spatially nearest point in the subsampled point cloud $\mathcal{P}^{(s)}$.

Data Augmentation. In order to reduce overfitting on small datasets, we perform online data augmentation. In particular, we randomly rotate point clouds about their up-axis, jitter their scale, perform mirroring, or delete random points.

3.3.5 Application in General Graphs

Many problems can be modeled directly as graphs. In such cases the graph dataset is already given and only the appropriate graph coarsening scheme needs to be chosen. Without any concept of spatial localization of nodes in general graphs, this is by no means trivial and there exists a large body of literature on this problem, briefly reviewed in Section 2.2.3.

Here, we resort to established graph coarsening algorithms and utilize the multiresolution framework of Shuman *et al.* (Perraudin et al., 2014; Shuman et al., 2016), which works by repeated downsampling and graph reduction of the input graph. The downsampling step is based on splitting the graph into two components by the sign of the largest eigenvector of the Laplacian. This is followed by Kron reduction (Dörfler and Bullo, 2013), which also defines new scalar edge attributes, enhanced with spectral sparsification of edges (Spielman and Srivastava, 2011). Note that the algorithm regards graphs as unweighted for the purpose of coarsening.

This method is attractive for us because of two reasons. Each downsampling step removes approximately half of the nodes, guaranteeing a certain level of pooling strength, and the sparsification step is randomized. The latter property is exploited as a useful data augmentation technique since several different graph pyramids can be generated from a single input graph. This is in spirit similar to the effect of fractional max-pooling (Graham, 2014). We do not perform any other data augmentation, as this would need to be domain specific.

3.4 Experiments

The proposed method is evaluated in point cloud classification (real-world data in Section 3.4.1 and synthetic in 3.4.2) and on a standard graph classification benchmark (Section 3.4.3). In addition, we validate our method and study its properties on MNIST (Section 3.4.4). Finally, we perform several ablation studies in Section 3.4.5.

3.4.1 Sydney Urban Objects

This point cloud dataset (De Deuge et al., 2013) consists of 588 objects in 14 categories (vehicles, pedestrians, signs, and trees) manually extracted from 360° LiDAR scans, see Figure 3.4. It demonstrates non-ideal sensing conditions with occlusions (holes) and a large variability in viewpoint (single viewpoint). This makes object classification a challenging task.

Following the protocol employed by the dataset authors, we report the mean F1 score weighted by class frequency, as the dataset is imbalanced. This score is further aggregated over four standard training/testing splits.

Network Configuration. Our ECC-network has 7 parametric layers and 4 levels of graph resolution. Its configuration can be described as C(16)-C(32)-MP(0.25,0.5)-C(32)-C(32)-MP(0.75,1.5)-C(64)-MP(1.5,1.5)-GAP-FC(64)-D(0.2)-FC(14), where C(c) denotes ECC with c output channels followed by affine batch normalization and ReLU activation, MP(r,ρ) stands for max-pooling down to grid resolution of r meters and neighborhood radius of ρ meters, GAP is global average pooling, FC(c) is fully-connected layer with c output channels, and D(p) is dropout with probability p. The filter-generating networks w^t have configuration FC(16)-FC(32)-FC($d_t d_{t-1}$) with orthogonal weight initialization (Saxe et al., 2014) and ReLUs in between. Input graphs are created with $r^0 = 0.1$ and $\rho^0 = 0.2$ meters to break overly dense point clusters. Networks are trained with SGD and cross-entropy loss for 250 epochs with batch size 32 and learning rate 0.1 step-wise decreasing after 200 and 245 epochs. Node signal H^0 is scalar laser return intensity (0-255), representing depth.

Results. Table 3.1 compares our result (ECC, 78.4) against two methods based on volumetric CNNs evaluated on voxelized occupancy grids of size 32^3 (VoxNet (Maturana and Scherer, 2015) 73.0 and ORION (Alvar et al., 2016) 77.8), which we outperform by a small margin and set the new state of the art result on this dataset.

3.4.2 ModelNet

ModelNet (Wu et al., 2015a) is a large scale collection of object meshes. We evaluate classification performance on its subsets ModelNet10 (3991/908 train/test examples in 10 categories) and ModelNet40 (9843/2468 train/test examples in 40 categories). Synthetic point clouds are created from meshes by uniformly sampling 1000 points on mesh faces

Model	Representation	Mean F1
Triangle+SVM (De Deuge et al., 2013)	image	67.1
GFH+SVM (Chen et al., 2014)	histogram	71.0
VoxNet (Maturana and Scherer, 2015)	volumetric	73.0
ORION (Alvar et al., 2016)	volumetric	77.8
ECC	graph	78.4

Table 3.1 Mean F1 score weighted by class frequency on Sydney Urban Objects dataset De Deuge et al. (2013). Only the best-performing models of each baseline are listed.

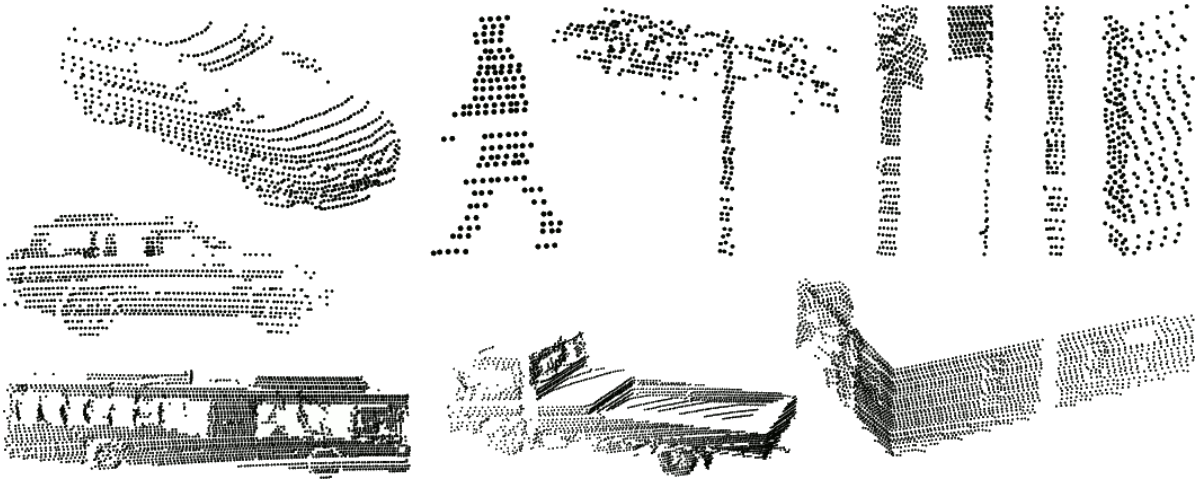


Fig. 3.4 Illustrative samples of the majority of classes in Sydney Urban Objects dataset, reproduced from De Deuge et al. (2013).

according to face area (a simulation of acquisition from multiple viewpoints) and rescaled into a unit sphere.

Network Configuration. Our ECC-network for ModelNet10 has 7 parametric layers and 3 levels of graph resolution with configuration C(16)-C(32)-MP(2.5/32,7.5/32)-C(32)-C(32)-MP(7.5/32,22.5/32)-C(64)-GMP-FC(64)-D(0.2)-FC(10), GMP being global max pooling. Other definitions and filter-generating networks w^t are as in Section 3.4.1. Input graphs are created with $r^0 = 1/32$ and $\rho^0 = 2/32$ units, mimicking the typical grid resolution of 32^3 in voxel-based methods. The network is trained with SGD and cross-entropy loss for 175 epochs with batch size 64 and learning rate 0.1 step-wise decreasing after every 50 epochs. There is no node signal, *i.e.* H^0 are zero. For ModelNet40, the

Model	Representation	ModelNet10		ModelNet40	
3DShapeNets (Wu et al., 2015a)	volumetric	83.5		77.3	
MVCNN (Su et al., 2015)	images			90.1	
VoxNet (Maturana and Scherer, 2015)	volumetric	92		83	
ORION (Alvar et al., 2016)	volumetric	93.8			
SubvolumeSup (Qi et al., 2016)	volumetric			86.0	(89.2)
* O-CNN (Wang et al., 2017)	sparse vol.				(90.6)
* KdNet (Klokov and Lempitsky, 2017)	sparse vol.		(94.0)		(91.8)
* PointNet (Qi et al., 2017a)	point set			86.2	(89.2)
* PointNet++ (Qi et al., 2017b)	point set				(90.7)
* DynamicGraph (Wang et al., 2018b)	graph			90.2	(92.2)
ECC	graph	89.3	(90.0)	82.4	(87.0)
ECC (12 votes)	graph	90.0	(90.8)	83.2	(87.4)

Table 3.2 Mean class accuracy (resp. mean instance accuracy) on ModelNets (Wu et al., 2015a). Only the best models of each baseline are listed. Concurrent baselines are denoted with a star.

network is wider (C(24), C(48), C(48), C(48), C(96), FC(64), FC(40)) and is trained for 100 epochs with learning rate decreasing after each 30 epochs.

Results. Table 3.2 compares our result to several previous works, based either on volumetric (Alvar et al., 2016; Maturana and Scherer, 2015; Qi et al., 2016; Wu et al., 2015a) or rendered image representation (Su et al., 2015), as well as to contemporary methods. Test sets were expanded to include 12 orientations (ECC). We also evaluate voting over orientations (ECC 12 votes), which slightly improves the results likely due to the rotational variance of VoxelGrid algorithm. While not having fully reached the state of the art at the time of submission, our method remained very competitive (90.8%, resp. 87.4% mean instance accuracy).

3.4.3 Graph Classification

We evaluate on a graph classification benchmark frequently used in the community, consisting of five datasets: NCI1, NCI109, MUTAG, ENZYMES, and D&D. Their properties can be found in Table 3.3, indicating the variability in dataset sizes, in graph sizes, and in the availability of attributes. Following Shervashidze et al. (2011), we perform 10-fold cross-validation with 9 folds for training and 1 for testing and report the average prediction accuracy.

NCI1 and NCI109 (Wale et al., 2008) consist of graph representations of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, respectively. MUTAG (Debnath et al., 1991) is a dataset of nitro compounds labeled according to whether or not they have a mutagenic effect on a bacterium. ENZYMES (Borgwardt and Kriegel, 2005) contains representations of tertiary structure of 6 classes of enzymes. D&D (Dobson and Doig, 2003) is a database of protein structures (nodes are amino acids, edges indicate spatial closeness) classified as enzymes and non-enzymes.

Network Configuration. Our ECC-network for NCI1 and NCI109 has 8 parametric layers and 3 levels of graph resolution. Its configuration can be described as C(48)-C(48)-C(48)-MP-C(48)-C(64)-MP-C(64)-GAP-FC(64)-D(0.1)-FC(2), where C(c) denotes ECC with c output channels followed by affine batch normalization, ReLU activation and dropout (probability 0.05), MP stands for max-pooling onto a coarser graph, GAP is global average pooling, FC(c) is fully-connected layer with c output channels, and D(p) is dropout with probability p. The filter-generating networks w^t have configuration FC(64)-FC($d_t d_{t-1}$) with orthogonal weight initialization Saxe et al. (2014) and ReLU in between. attributes are encoded as one-hot vectors ($d_0 = 37$ and $s = 4$ due to an extra attribute for self-connections). Networks are trained with SGD and cross-entropy loss for 50 epochs with batch size 64 and learning rate 0.1 step-wise decreasing after 25, 35, and 45 epochs. The dataset is expanded five times by randomized sparsification (Section 3.3.5).

We made small deviations from this description for the other three datasets as follows. As MUTAG is a tiny dataset of small graphs, we trained a downsized ECC-network to combat overfitting with configuration C(16)-C(32)-C(48)-MP-C(64)-MP-GAP-FC(64)-D(0.2)-FC(2). Due to higher complexity of ENZYMES we use a wider ECC-network configured as C(64)-C(64)-C(96)-MP-C(96)-C(128)-MP-C(128)-C(160)-MP-C(160)-GAP-FC(192)-D(0.2)-FC(6). Finally, due to large graphs in D&D dataset we designed a ECC-network with more pooling configured as C(48)-C(48)-C(48)-MP-C(48)-MP-C(64)-MP-C(64)-MP-C(64)-MP-C(64)-MP-GAP-FC(64)-D(0.2)-FC(2).

Baselines. We compare our method (ECC) to the state of the art Weisfeiler-Lehman graph kernel (Shervashidze et al., 2011) and to four approaches using deep learning as at least one of their components (Atwood and Towsley, 2016; Dai et al., 2016; Narayanan et al., 2016; Niepert et al., 2016; Yanardag and Vishwanathan, 2015). Randomized sparsification used during training time can also be exploited at test time, when the

network prediction scores (ECC-5-scores) or votes (ECC-5-votes) are averaged over 5 runs. To judge the influence of edge attributes, we run our method with uniform attributes and w^t being a single layer FC($d_t d_{t-1}$) without bias[3] (ECC no edge attributes).

Results. Table 3.4 conveys that while there is no clear winning algorithm, our method performs at the level of state of the art for edge-labeled datasets (NCI1, NCI109, MUTAG). The results demonstrate the importance of exploiting edge attributes for convolution-based methods, as the performance of DCNN (Atwood and Towsley, 2016) and ECC without edge attributes is distinctly worse, justifying the motivation behind this chapter. Averaging over random sparsifications at test time improves accuracy by a small amount. Our results on datasets without edge attributes (ENZYMES, D&D) are somewhat below the state of the art but still at a reasonable level, though improvement in this case was not the aim of this work. This indicates that further research is needed into the adaptation of CNNs to general graphs. In the following, we discuss the results for each dataset in more detail.

NCI1. ECC (83.80%) performs distinctly better than convolution methods that are not able to use edge attributes (DCNN (Atwood and Towsley, 2016) 62.61%, PSCN (Niepert et al., 2016) 78.59%). Methods not approaching the problem as convolutions on graphs but rather combining deep learning with other techniques are stronger (subgraph2vec (Narayanan et al., 2016) 78.05%, Deep WL (Yanardag and Vishwanathan, 2015) 80.31%, structure2vec (Dai et al., 2016) 83.72%) but are still outperformed by ECC. While the Weisfeiler-Lehman graph kernel remains the strongest method (WL (Shervashidze et al., 2011) 84.55%), it is fair to conclude that ECC, structure2vec, and WL perform at the same level.

NCI109. ECC (82.14%) performs distinctly better than DCNN (Atwood and Towsley, 2016) (62.86%), which is not able to use edge attributes, and is on par with non-convolutional approaches (subgraph2vec (Narayanan et al., 2016) 78.39%, Deep WL (Yanardag and Vishwanathan, 2015) 80.32%, structure2vec (Dai et al., 2016) 82.16%, WL (Shervashidze et al., 2011) 84.49%).

MUTAG. While by numbers ECC (89.44%) outperforms all other approaches except of PSCN (Niepert et al., 2016) (92.63%), we note that all four leading methods (subgraph2vec (Narayanan et al., 2016) 87.17%, Deep WL (Yanardag and Vishwanathan, 2015) 87.44%, structure2vec (Dai et al., 2016) 88.28%, ECC, PSCN) can be seen to perform equally

[3] Also possible for unlabeled ENZYMES and D&D, since our method uses attributes from Kron reduction for all coarsened graphs by default.

	NCI1	NCI109	MUTAG	ENZYMES	D&D
# graphs	4110	4127	188	600	1178
mean $\|V\|$	29.87	29.68	17.93	32.63	284.32
mean $\|E\|$	32.3	32.13	19.79	62.14	715.66
# classes	2	2	2	6	2
# node attributes	37	38	7	3	82
# edge attributes	3	3	11	—	—

Table 3.3 Characteristics of the graph benchmark datasets, extended from (Dai et al., 2016). Both edge and node attributes are categorical.

well due to fluctuations caused by the dataset size. We account the tiny decrease in performance with test-time randomization (88.33%) to the same reason.

ENZYMES. As this dataset is not edge-labeled, we do not expect to obtain the best performance. Indeed, our method (53.50%) performs at the level of Deep WL (Yanardag and Vishwanathan, 2015) (53.43%) and is overperformed by WL (Shervashidze et al., 2011) (59.05%) and structure2vec (Dai et al., 2016) (61.10%). Note that the gap to the other convolution-based method DCNN (Atwood and Towsley, 2016) (18.10%) is huge and there is an improvement of more than 4 percentage points due to edge attributes in coarser graph resolutions from Kron reduction.

D&D. As this dataset is also not edge-labeled, we do not expect to obtain the best performance. Our method (74.10%) is overperformed by the others who evaluated on this dataset (PSCN (Niepert et al., 2016) 77.12%, WL (Shervashidze et al., 2011) 79.78%, structure2vec (Dai et al., 2016) 82.22%), though the margin is not very large.

3.4.4 MNIST

To further validate our method, we applied it to the MNIST classification problem (LeCun et al., 1998), a dataset of 70k greyscale images of handwritten digits represented on a 2D grid of size 28×28. We regard each image I as point cloud $\mathcal{P}$ with points $p = (x, y, 0)$ and signal $F_P(p) = I(x, y)$ representing each pixel, $x, y \in \{0, .., 27\}$. Edge labeling and graph coarsening is performed as explained in Section 3.3.4. We are mainly interested in two questions: Is ECC able to reach the standard performance on this classic baseline? What kind of representation does it learn?

Network Configuration. Our ECC-network has 5 parametric layers with configuration C(16)-MP(2,3.4)-C(32)-MP(4,6.8)-C(64)-MP(8,30)-C(128)-D(0.5)-FC(10); the

Model	NCI1	NCI109	MUTAG	ENZYMES	D&D
DCNN (Atwood and Towsley, 2016)	62.61	62.86	66.98	18.10	—
subgraph2vec (Narayanan et al., 2016)	78.05	78.39	87.17	—	—
PSCN (Niepert et al., 2016)	78.59	—	92.63	—	77.12
Deep WL (Yanardag and Vishwanathan, 2015)	80.31	80.32	87.44	53.43	—
structure2vec (Dai et al., 2016)	83.72	82.16	88.28	61.10	82.22
WL (Shervashidze et al., 2011)	84.55	84.49	83.78	59.05	79.78
* CCN (Kondor et al., 2018)	76.27	75.54	91.64	—	—
* DGCNN (Zhang et al., 2018)	74.44	—	85.83	—	79.37
ECC (no edge attributes)	76.82	75.03	76.11	45.67	72.54
ECC	83.80	81.87	89.44	50.00	73.65
ECC (5 votes)	83.63	82.04	88.33	53.50	73.68
ECC (5 scores)	83.80	82.14	88.33	52.67	74.10

Table 3.4 Mean accuracy (10 folds) on graph classification datasets. Only the best-performing models of each baseline are listed. Baselines newer than our work are denoted with a star.

notation and filter-generating network being as in Section 3.4.1. The last convolution has a stride of 30 and thus maps all 4×4 points to only a single point. Input graphs are created with $r^0 = 1$ and $\rho^0 = 2.9$. This model exactly corresponds to a regular CNN with three convolutions with filters of size 5×5, 3×3, and 3×3 interlaced with max-poolings of size 2×2, finished with two fully connected layers. Networks are trained with SGD and cross-entropy loss for 20 epochs with batch size 64 and learning rate 0.01 step-wise decreasing after 10 and 15 epochs.

Results. Table 3.5 proves that our ECC network can achieve the level of quality comparable to the good standard in the community (99.14). This is exactly the same accuracy as reported by Defferrard et al. (2016) and better than what is offered by other spectral-based approaches (98.2 (Bruna et al., 2013), 94.96 (Edwards and Xie, 2016)). Note that we are not aiming at becoming the state of the art on MNIST by this work.

Next, we investigate the effect of regular grid and irregular mesh. To this end, we discard all black points $i : F_i = 0$) from the point clouds, corresponding to 80.9% of data, and retrain the network (ECC sparse input). Exactly the same test performance is obtained (99.14), indicating that our method is very stable with respect to graph structure changing from sample to sample.

Furthermore, we check the quality of the learned filter generating networks w^t. We compare with ECC configured to mimic regular convolution using single-layer filter

Model	Train accuracy	Test accuracy
ECC	99.12	99.14
ECC (sparse input)	99.36	99.14
ECC (one-hot)	99.53	99.37

Table 3.5 Accuracy on MNIST dataset (LeCun et al., 1998).

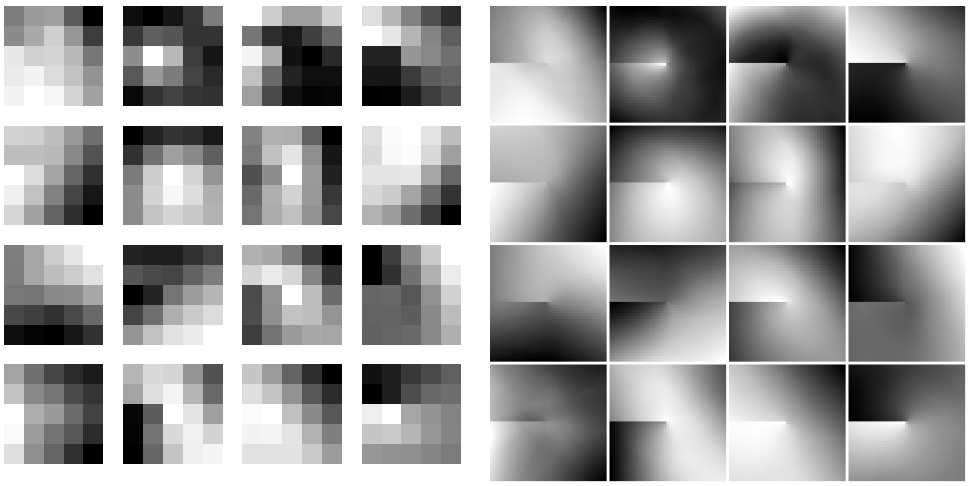


Fig. 3.5 Convolutional filters learned on MNIST in the first layer for sparse input ECC, sampled in two different resolutions. See Section 3.4.4 for details.

networks and one-hot encoding of offsets (ECC one-hot), as described in Section 3.3.2. This configuration reaches 99.37 accuracy, or 0.23 more than ECC, implying that w^t are not perfect but still perform very well in learning the proper partitioning of edge attributes.

Last, we explore the generated filters visually for the case of the sparse input ECC. As filters $W^0 \in \mathbb{R}^{16\times 1}$ are a continuous function of an edge attribute, we can visualize the change of values in each dimension in 16 images by sampling attributes over grids of two resolutions. The coarser one in Figure 3.5 has integer steps corresponding to the offsets $\delta_x, \delta_y \in \{-2, .., 2\}$. It shows filters exhibiting the structured patterns typically found in the first layer of CNNs. The finer resolution in Figure 3.5 (sub-pixel steps of 0.1) reveals that the filters are in fact smooth and do not contain any discontinuities apart from the angular artifact due to the 2π periodicity of azimuth. Interestingly, the artifact is not distinct in all filters, suggesting the network may learn to overcome it if necessary.

Model	Mean F1
ECC 2ρ	74.4
ECC 1.5ρ	76.9
ECC	78.4
ECC-id 2ρ	77.4
ECC-id 1.5ρ	79.5
ECC-id	77.0

Table 3.6 Influence of varied neighborhood radius on Sydney Urban Objects dataset De Deuge et al. (2013).

3.4.5 Detailed Analyses and Ablations

This section provides analysis of several design choices and investigates robustness of point cloud classification to noise. In the second part, we explore two extensions of our ECC formulation, specifically with degree attributes and with a learned normalization factor.

Neighborhood Radius. In Table 3.6 we study the dependence on convolution radii ρ: increasing them $1.5\times$ or $2\times$ in all convolutional layers leads to a drop in performance in Sydney dataset, which would correspond to a preference of using smaller filters in regular CNNs. The average neighborhood size is roughly 10 nodes for our best-performing network. We hypothesized that larger radii smooth out the information in the central node. To investigate this, we increased the importance of the self-loop by adding an identity skip-connection (ECC-id) and retrained the networks. Indeed, stronger identity connection allowed for successful integration of a larger context, up to some limit, which suggests that information should be aggregated neither too much nor too little.

Identity Connections We compare the performance of ECC (quation 3.1) and ECC-id (Equation 3.1) in Table 3.7. With two exceptions (NCI109 and ENZYMES), ECC does not benefit from identity connections in the specific network configurations. The trend may be different for other configurations, *e.g.* ECC 1.5ρ improved from 76.9 to 79.5 mean F1 score on Sydney due to identity connections as mentioned above.

Edge Attributes for Point Clouds In Section 3.3.4 we defined edge attributes $E_{j,i}$ as the offset $\delta = P_j - P_i$ expressed in Cartesian and spherical coordinates. Here, we explore the importance of individual elements in the proposed edge labeling and further evaluate attributes invariant to rotation about objects' vertical axis z (IRz). Table 3.8

	NCI1	NCI109	MUTAG	ENZYMES	D&D	Sydney	ModelNet10
ECC-id	83.24	*81.97*	85.56	*51.83*	70.48	77.0	88.5 (89.3)
ECC	83.80	81.87	89.44	50.00	73.65	78.4	89.3 (90.0)

Table 3.7 The effect of adding identity connections (improvements in italics). Performance metrics vary and are specific to each dataset, as introduced in the respective sections.

attribute $E_{j,i}$	Mean F1
$(\delta_x, \delta_y, \delta_z, \|\|\delta\|\|, \arccos \delta_z/\|\|\delta\|\|, \arctan \delta_y/\delta_x)$	78.4
$(\delta_x, \delta_y, \delta_z)$	76.1
$(\|\|\delta\|\|, \arccos \delta_z/\|\|\delta\|\|, \arctan \delta_y/\delta_x)$	77.3
$(\|\|\delta_{xy}\|\|, \delta_z, \|\|\delta\|\|, \arccos \delta_z/\|\|\delta\|\|)$	75.8
$(\|\|\delta_{xy}\|\|, \delta_z)$	78.2
$(\|\|\delta\|\|, \arccos \delta_z/\|\|\delta\|\|)$	78.7
$(\|\|\delta\|\|)$	60.7
(0)	38.9

Table 3.8 ECC on Sydney dataset with varied edge attribute definition.

conveys that models with isotropic (60.7) or no attributes (38.9) perform poorly as expected, while either of the coordinate systems is important. IRz labeling performs comparably or even slightly better than our proposed one. However, we believe this is a property of the specific dataset and may not necessarily generalize, an example being MNIST, where IRz is equivalent to full isotropy and decreases accuracy to 89.9%.

Robustness to Noise Real-world point clouds contain several kinds of artifacts, such as holes due to occlusions and Gaussian noise due to measurement uncertainty. Figure 3.6 shows that ECC is highly robust to point removal and can be made robust to additive Gaussian noise by a proper training data augmentation.

Node Degrees in Edge Attributes In the task of graph classification, we used categorical attributes (if present) encoded as one-hot vectors for edges in the input graph and scalars computed by Kron reduction for edges in all coarsened graphs. Here we investigate making the edge attributes more informative by including the degrees of the pair of nodes forming an edge. The degree information is implicitly used by spectral convolution methods, as the degree information is contained in the graph Laplacian,

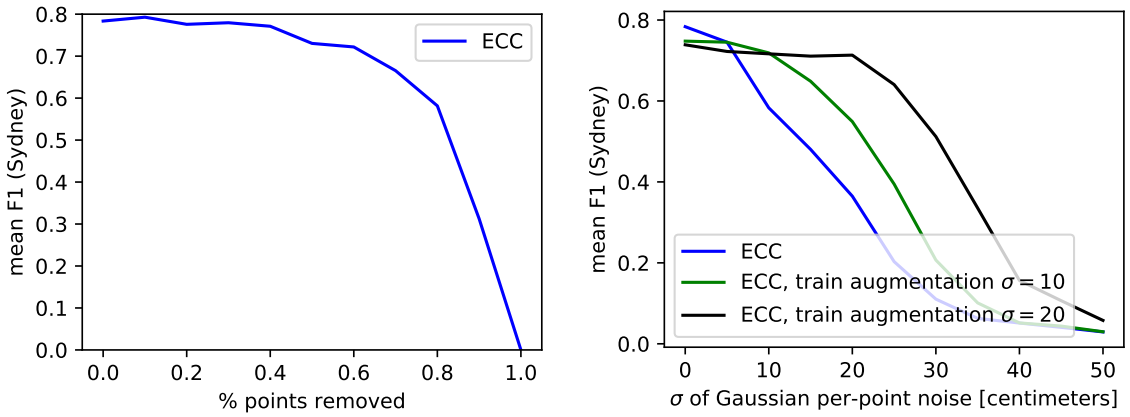


Fig. 3.6 Robustness to point removal and Gaussian noise on Sydney dataset.

	NCI1	NCI109	MUTAG	ENZYMES	D&D
$L_{deg}(i) = 1/\sqrt{\deg(i)}$	82.99	81.94	87.78	*53.67*	73.65
$L_{deg}(i) = 1/\deg(i)$	83.60	*82.40*	88.89	*52.67*	71.77
$L_{deg}(i) = \sqrt{\deg(i)}$	83.58	*82.28*	86.67	*55.00*	*75.79*
$L_{deg}(i) = \deg(i)$	83.16	*83.03*	86.67	*52.83*	*73.74*
ECC without $L_{deg}(i)$	83.80	81.87	89.44	50.00	73.65

Table 3.9 The effect in mean classification accuracy of adding node degrees to edge attributes (improvements in italics).

and also appears in the explicit propagation rules (Atwood and Towsley, 2016; Kipf and Welling, 2016a).

Our model can be easily extended to make use of this information by simply appending it to the existing edge attribute vectors. We consider four variants of providing additional degree attributes $L_{deg}(j)$ and $L_{deg}(i)$ about a directed edge (j, i): $L_{deg}(i) = 1/\sqrt{\deg(i)}$, $L_{deg}(i) = 1/\deg(i)$, $L_{deg}(i) = \sqrt{\deg(i)}$, and $L_{deg}(i) = \deg(i)$, where $\deg(i) = |\mathcal{N}(i)|$ is the degree of node $i \in \mathcal{V}$. We use these additional attributes in all graph resolutions.

Table 3.9 reveals that degree information can improve the results considerably, especially for datasets without given edge attributes (by up to 5 percentage points for ENZYMES and up to 2.14 percentage points for D&D). However, no variant of $L_{deg}(i)$ can guarantee consistent improvement over all datasets.

Node Degrees in Normalization The formulation of ECC in Equation 3.1 performs normalization by the neighborhood size. Here we explore learning an additional multi-

	NCI1	NCI109	MUTAG	ENZYMES	D&D	Sydney	ModelNet10
ECC-f	83.48	*82.57*	86.67	*52.50*	72.03	75.5	*89.9 (90.6)*
ECC	83.80	81.87	89.44	50.00	73.65	78.4	89.3 (90.0)

Table 3.10 The effect of adding a learned normalization factor (improvements in italics). Performance metrics vary and are specific to each dataset, as introduced in the respective sections.

plicative factor, conditioned on the neighborhood size $1/|\mathcal{N}(i)|$. The motivation is to investigate whether modeling a relation between neighborhood size and feature magnitude (such as a smooth transition from a sum to an average) could improve predictive performance. To this end, we again make use of Dynamic filter networks (Brabandere et al., 2016) and design a factor-generating network $f : \mathbb{R} \to \mathbb{R}$ which given node degree $\deg(i) = |\mathcal{N}(i)|$ outputs a node-specific normalization factor. We formulate ECC-f as follows:

$$H_i^{t+1} = \text{ReLU}\left(\frac{f^t(|\mathcal{N}(i)|;\theta^t)}{|\mathcal{N}(i)\cup\{i\}|}\sum_{j\in\mathcal{N}(i)\cup\{i\}} w^t(E_{j,i};\theta^t)H_j^t + \mathbf{b}^t\right) \tag{3.3}$$

In our experiments, the factor-generating networks f^t have configuration FC(32)-FC(1) with orthogonal weight initialization (Saxe et al., 2014) and ReLUs in between.

The results in Table 3.10 show that while being helpful on some datasets (NCI109, ENZYMES, ModelNet10), ECC-f harms the performance on the other ones. Embedding node information in attributes instead seems to achieve higher performance, see above.

3.5 Discussion

In this section, we take a critical viewpoint and discus several limitations of our work as well.

Memory requirements. While the possibility to work with continuous attributes in ECC is very convenient in many problems, the model may become quite GPU memory demanding, especially during training when all intermediate activations need to be stored (*e.g.* around 9 GB for our ModelNet10 network). This is particularly valid for point clouds, where nearly all edges have their unique edge attributes in practice and thus also their unique filter weights W.

There are at least two options for addressing this problem. From a modeling perspective, one can decrease the number of parameters in generated weights W. One way is to restrict W to low rank q by generating $W_a \in \mathbb{R}^{q \times d^{t-1}}$ and $W_b \in \mathbb{R}^{d^t \times q}$ and computing $WH_j^t = W_b(W_a H_j^t)$, which is more efficient as long as $q(d^t + d^{t-1}) < d^t d^{t-1}$. Unfortunately, we have seen considerably worse performance in practice. It turned out that a better way in terms of both performance and memory is to make W diagonal, effectively replacing matrix-vector multiplication with element-wise multiplication; we discuss this extension in more detail farther in Chapter 4.

From an engineering perspective, we experimented with clustering of edge attributes during training with K-means. This can be done independently for each training sample offline or even online in each iteration. Unlike simple predefined quantization, this approach can maintain the original empirical distribution of attributes over the dataset and also doubles as data augmentation. In spite of that, we observed a small decrease in performance.

In general, the overhead in terms of memory and computation together with no distinct benefit in predictive performance is likely the main reason why concurrently introduced set-based approaches such as PointNet (Qi et al., 2017a) have become more popular in the community for simple, small-scale point clouds. In such cases, modeling nearest neighbor structures using graphs may indeed be a bit of an overkill. However, in Chapter 4 we demonstrate a major advantage of using explicit spatial relationships for modeling large-scale point clouds.

Feed-forward update function. In this Chapter, we were using simple update functions $u_t : \mathrm{ReLU}(M_i^{t+1})$ and $u_t : \mathrm{ReLU}(\mathrm{id}(H_i^t) + M_i^{t+1})$. While being consistent with many popular feed-forward architectures for images, the community has drawn inspiration from recurrent networks and often adopted gated update functions (Gilmer et al., 2017; Li et al., 2016b; Schütt et al., 2017), which should provide increased protection against oversmoothing information within neighborhoods. In Chapter 4, we also adopt this view.

Rotation variance. Our definitions of edge attributes for point clouds bind together the degree of (in)variance to local and global transformations, in particular to rotations. Instead, very often, the ideal would be to remain equivariant for reasoning about local spatial relationships and obtain invariance only at the global scale. Popular remedies include data augmentation, as e.g. in this thesis[4], or spatial transformer

[4]We also performed preliminary experiment finding a local coordinate system in each neighborhood with principal component analysis of point coordinates but were not able to achieve reasonable results.

networks (Jaderberg et al., 2015), as e.g. in PointNet (Qi et al., 2017a). Solving the problem of equivariance efficiently and in a principled way is an active research topic both in sparse and dense representations and in 2D and 3D - see Kondor (2018); Thomas et al. (2018) for very recent methods targeting the point cloud domain.

3.6 Conclusion

We have introduced edge-conditioned convolution (ECC), an operation on graph node signal performed in the spatial domain where filter weights are conditioned on edge attributes and dynamically generated for each specific input sample. We have shown that our formulation generalizes the standard convolution on graphs if edge attributes are chosen properly and experimentally validated this assertion on MNIST. We applied our approach to point cloud classification in a novel way, setting a new state of the art performance on Sydney dataset. Furthermore, we have outperformed other deep learning-based approaches on graph classification dataset NCI1. The source code has been published at https://github.com/mys007/ecc.

In the next chapter, we integrate ECC into a recurrent network, investigate a reduction of its memory and computational requirements, and apply it to node-wise prediction task rather than graph classification tasks as in this chapter.

Chapter 4

Large-scale Point Cloud Segmentation

4.1 Introduction

Visual understanding of 3D environment is a fundamental requirement for agents acting in the real world, allowing for applications in autonomous driving and robotics, perception assistance tools or augmented reality. Point cloud representation frequently arises in such systems, usually originating in dedicated sensors, such as LiDAR and other active remote sensing devices, or coming from multi-view / structure-from-motion reconstruction methods. However, analysis of large 3D point clouds presents numerous challenges, the most obvious one being the scale of the data. Another hurdle is the lack of clear structure akin to the regular grid arrangement in images, especially in cases when the data has been fused from multiple sensors or the position of the sensor is uncertain.

Previous attempts at using deep learning for large 3D data were trying to replicate successful CNN architectures used for image segmentation. For example, SnapNet (Boulch et al., 2017) converts a 3D point cloud into a set of virtual 2D color and depth (RGB-D) snapshots, the semantic segmentation of which can then be projected on the original data. SegCloud (Tchapmi et al., 2017) uses 3D convolutions on a regular voxel grid. However, we argue that such methods do not capture the inherent structure of 3D point clouds, which results in limited discrimination performance. Indeed, converting point clouds to 2D format comes with loss of information and requires to perform surface reconstruction, a problem arguably as hard as semantic segmentation. Volumetric representation of point clouds is inefficient and tends to discard small details, as discussed in Section 3.2 before.

Deep learning architectures specifically designed for 3D point clouds (Engelmann et al., 2017; Qi et al., 2017a,b; Riegler et al., 2017b), including our work presented in

Chapter 3, display good results but are limited by the size of inputs they can handle at once.

In this chapter we propose a representation of large 3D point clouds as a collection of interconnected simple shapes, coined superpoints, in spirit similar to superpixel methods for image segmentation (Achanta et al., 2012). As illustrated in Figure 4.1, this structure can be captured by an attributed directed graph called the superpoint graph (SPG). Its nodes represent simple shapes while edges describe their adjacency relationship characterized by rich edge attributes.

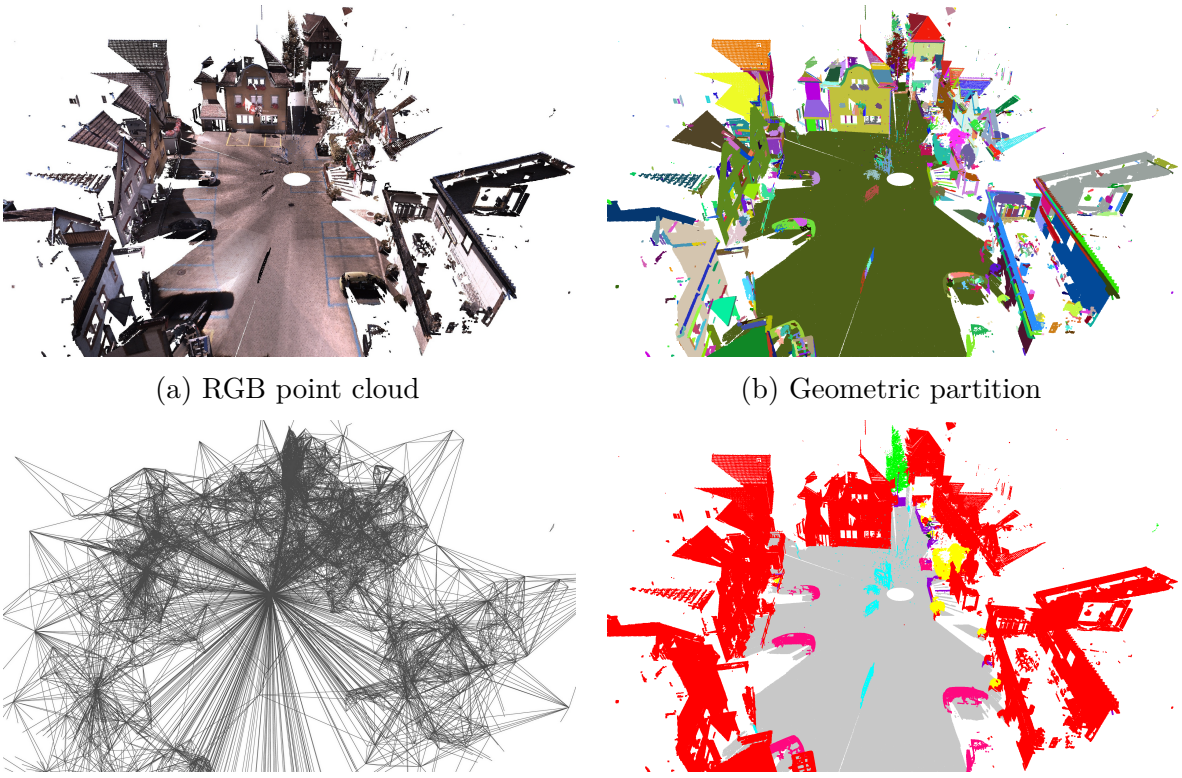
The SPG representation has several compelling advantages. First, instead of classifying individual points or voxels, it considers entire object parts as whole, which are easier to identify. Second, it is able to describe in detail the relationship between adjacent objects, which is crucial for contextual classification: cars are generally above roads, ceilings are surrounded by walls, etc. Third, the size of the SPG is defined by the number of simple structures in a scene rather than the total number of points, which is typically several order of magnitude smaller. This allows us to model long-range interaction which would be intractable otherwise without strong assumptions on the nature of the pairwise connections.

This chapter is largely based on our CVPR 2018 publication (Landrieu and Simonovsky, 2018), where the contribution was equally shared with Loïc Landrieu. Its contributions to the field at the time of publication are as follows:

- We introduce superpoint graphs, a novel point cloud representation with rich edge attributes encoding the contextual relationship between object parts in 3D point clouds.

- Based on this representation, we are able to apply deep learning on large-scale point clouds without major sacrifice in fine details. Our architecture consists of PointNets (Qi et al., 2017a) for superpoint embedding and graph convolutions for contextual segmentation. For the latter, we introduce a novel, more efficient version of Edge-Conditioned Convolutions from previous chapter as well as a new form of input gating in Gated Recurrent Units (Cho et al., 2014a).

- We set a new state of the art on two publicly available datasets: Semantic3D (Hackel et al., 2017) and S3DIS (Armeni et al., 2016). In particular, we improve mean per-class intersection over union (mIoU) by 11.9 points for the Semantic3D reduced test set, by 8.8 points for the Semantic3D full test set, and by up to 12.4 points for the S3DIS dataset.



(a) RGB point cloud (b) Geometric partition

(c) Superpoint graph (d) Semantic segmentation

Fig. 4.1 Visualization of individual steps in our pipeline. An input point cloud (a) is partitioned into geometrically simple shapes, called superpoints (b). Based on this preprocessing, a superpoint graph (SPG) is constructed by linking nearby superpoints by superedges with rich attributes (c). Finally, superpoints are transformed into compact embeddings, processed with graph convolutions to make use of contextual information, and classified into semantic labels.

4.2 Related Work

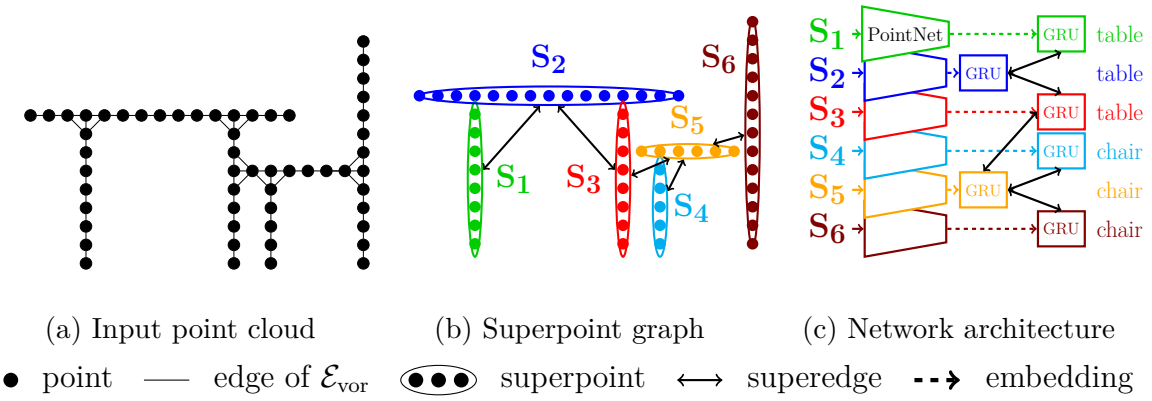
The classic approach to large-scale point cloud segmentation is to classify each point or voxel independently using handcrafted features derived from their local neighborhood (Weinmann et al., 2015). The solution is then spatially regularized using graphical models (Anand et al., 2013; Kim et al., 2013; Koppula et al., 2011; Lu and Rasmussen, 2012; Martinovic et al., 2015; Munoz et al., 2009; Niemeyer et al., 2014; Shapovalov et al., 2013; Wolf et al., 2015) or structured optimization (Landrieu et al., 2017). Clustering as preprocessing (Guinard and Landrieu, 2017; Hu et al., 2013) or postprocessing (Weinmann et al., 2017) have been used by several frameworks to improve the accuracy of the classification.

Deep Learning on Point Clouds. Several different approaches going beyond naive volumetric processing of point clouds have been proposed recently and briefly reviewed in Section 3.2 before. However, very few methods with deep learning components have been demonstrated to be able to segment large-scale point clouds. PointNet (Qi et al., 2017a) can segment large clouds with a sliding window approach, therefore constraining contextual information within a small area only. Engelmann et al. (2017) improves on this by increasing the context scope with multi-scale windows or by considering directly neighboring window positions on a voxel grid. SEGCloud (Tchapmi et al., 2017) handles large clouds by voxelizing followed by interpolation back to the original resolution and post-processing with a conditional random field (CRF). None of these approaches is able to consider fine details and long-range contextual information simultaneously. In contrast, our pipeline partitions point clouds in an adaptive way according to their geometric complexity and allows deep learning architecture to use both fine detail and interactions over long distance.

Graph Convolutions. A key step of our approach is using graph convolutions to spread contextual information. Formulations that are able to deal with graphs of variable sizes can be seen as a form of message passing over graph edges (Gilmer et al., 2017). Of particular interest are models supporting continuous edge attributes, which we use to represent interactions. In image segmentation, convolutions on graphs built over superpixels have been used for post-processing: Liang et al. (2017, 2016) traverses such graphs in a sequential node order based on unary confidences to improve the final labels. We update graph nodes in parallel and exploit edge attributes for informative context modeling. Xu et al. (2017) convolves information over graphs of object detections to infer their contextual relationships. Our work infers relationships implicitly to improve segmentation results. Qi et al. (2017c) also relies on graph convolutions on 3D point clouds. However, we process large point clouds instead of small RGB-D images with nodes embedded in 3D instead of 2D in a novel, rich-attributed graph. Finally, we note that graph convolutions also bear functional similarity to deep learning formulations of CRFs (Zheng et al., 2015), which we discuss more in Section 4.3.4.

4.3 Method

The main obstacle that our framework tries to overcome is the size of typical LiDAR scans. Indeed, they can reach hundreds of millions of points, making direct deep learning approaches intractable. The proposed superpoint graph (SPG) representation allows us to split the semantic segmentation problem into three distinct problems of different



(a) Input point cloud (b) Superpoint graph (c) Network architecture

Fig. 4.2 Illustration of our framework on a toy scan of a table and a chair. We perform geometric partitioning on the point cloud (a), which allows us to build the superpoint graph (b). Each superpoint is embedded by a PointNet network. The embeddings are then refined in GRUs by message passing along superedges to produce the final labeling (c).

scales, shown in Figure 4.2, which can in turn be solved by methods of corresponding complexity:

1 **Geometrically homogeneous partition:** The first step of our algorithm is to partition the point cloud into geometrically simple yet meaningful shapes, called superpoints. This unsupervised step takes the whole point cloud as input, and therefore must be computationally very efficient. The SPG can be easily computed from this partition.

2 **Superpoint embedding:** Each node of the SPG corresponds to a small part of the point cloud corresponding to a geometrically simple primitive, which we assume to be semantically homogeneous. Such primitives can be reliably represented by downsampling small point clouds to at most hundreds of points. This small size allows us to utilize recent point cloud embedding methods such as PointNet (Qi et al., 2017a).

3 **Contextual segmentation:** The graph of superpoints is by orders of magnitude smaller than any graph built on the original point cloud. Deep learning algorithms based on graph convolutions can then be used to classify its nodes using rich edge attributes facilitating long-range interactions.

The SPG representation allows us to perform end-to-end learning of the trainable two last steps. We will describe each step of our pipeline in the following subsections.

4.3.1 Geometric Partition with a Global Energy

In this subsection, we describe our method for partitioning the input point cloud into parts of simple shape. Our objective is not to retrieve individual objects such as cars or chairs, but rather to break down the objects into simple parts, as seen in Figure 4.4. However, the clusters being geometrically simple, one can expect them to be semantically homogeneous as well, *i.e.* not to cover objects of different classes. Note that this step of the pipeline is purely unsupervised and makes no use of class labels beyond validation.

We follow the global energy model described by Guinard and Landrieu (2017) for its computational efficiency. Another advantage is that the segmentation is adaptive to the local geometric complexity. In other words, the segments obtained can be large simple shapes such as roads or walls, as well as much smaller components such as parts of a car or a chair.

Let us consider the input point cloud $\mathcal{P}$ as a set of n 3D points. Each point $i \in \mathcal{P}$ is defined by its 3D position P_i, and, if available, other observations O_i such as color or intensity. For each point, we compute a set of d_g geometric features $F_i \in \mathbb{R}^{d_g}$ characterizing the shape of its local neighborhood. In this paper, we use three dimensionality values proposed by Demantké et al. (2011): linearity, planarity and scattering, as well as the verticality feature introduced by Guinard and Landrieu (2017). We also compute the elevation of each point, defined as the z coordinate of P_i normalized over the whole input cloud.

The global energy proposed by Guinard and Landrieu (2017) is defined with respect to the 10-nearest neighbor adjacency graph $G_{\text{nn}} = (\mathcal{P}, \mathcal{E}_{\text{nn}})$ of the point cloud (note that this is not the SPG). The geometrically homogeneous partition is defined as the constant connected components of the solution of the following optimization problem:

$$\arg\min_{G \in \mathbb{R}^{d_g \times n}} \sum_{i \in \mathcal{P}} \|G_i - F_i\|^2 + \mu \sum_{(i,j) \in \mathcal{E}_{\text{nn}}} \boldsymbol{\alpha}_{(i,j)} \left[G_i \neq G_j\right], \tag{4.1}$$

where $[\cdot]$ is the Iverson bracket. The edge weight $\boldsymbol{\alpha} \in \mathbb{R}_+^{|E|}$ is chosen to be inversely proportional to the edge length. The factor μ is the regularization strength and determines the coarseness of the resulting partition.

The problem defined in Equation 4.1 is known as generalized minimal partition problem, and can be seen as a continuous-space version of the Potts energy model, or an ℓ_0 variant of the graph total variation. The minimized functional being nonconvex and noncontinuous implies that the problem cannot realistically be solved exactly for large point clouds. However, the ℓ_0-cut pursuit algorithm introduced by Landrieu and

Obozinski (2017) is able to quickly find an approximate solution with a few graph-cut iterations. In contrast to other optimization methods such as α-expansion (Boykov et al., 2001), the ℓ_0-cut pursuit algorithm does not require selecting the size of the partition in advance. The constant connected components $\mathcal{S} = \{S_1, \cdots, S_k\}$ of the solution of Equation 4.1 define our geometrically simple elements, and are referred as *superpoints* (*i.e.* set of points) in the rest of this chapter.

4.3.2 Superpoint Graph Construction

In this subsection, we describe how we compute the SPG as well as its key features. The SPG is a structured representation of the point cloud, defined as an oriented attributed graph $\mathcal{G} = (\mathcal{S}, \mathcal{E}, E)$ whose nodes are the set of superpoints $\mathcal{S}$ and edges $\mathcal{E}$ (referred to as *superedges*) represent the adjacency between superpoints. The superedges are annotated by a set of d_f attributes: $E \in \mathbb{R}^{|\mathcal{E}| \times d_f}$ characterizing the adjacency relationship between superpoints.

We define $G_{\text{vor}} = (\mathcal{P}, \mathcal{E}_{\text{vor}})$ as the symmetric Voronoi adjacency graph of the complete input point cloud as defined by Jaromczyk and Toussaint (1992). Two superpoints S and R are adjacent if there is at least one edge in $\mathcal{E}_{\text{vor}}$ with one end in S and one end in R:

$$\mathcal{E} = \left\{ (S, R) \in \mathcal{S}^2 \mid \exists\, (i, j) \in \mathcal{E}_{\text{vor}} \cap (S \times R) \right\}. \tag{4.2}$$

Important spatial attributes associated with a superedge (S, R) are obtained from the set of offsets $\delta(S, R)$ for edges in $\mathcal{E}_{\text{vor}}$ linking both superpoints:

$$\delta\,(S, R) = \{ (p_i - p_j) \mid (i, j) \in \mathcal{E}_{\text{vor}} \cap (S \times R) \}\,. \tag{4.3}$$

Superedge attributes can also be derived by comparing the shape and size of the adjacent superpoints. To this end, we compute $|S|$ as the number of points comprised in a superpoint S, as well as shape attribute $\text{length}\,(S) = \lambda_1$, $\text{surface}\,(S) = \lambda_1\lambda_2$, $\text{volume}\,(S) = \lambda_1\lambda_2\lambda_3$ derived from the eigenvalues $\lambda_1, \lambda_2, \lambda_3$ of the covariance of the positions of the points comprised in each superpoint, sorted by decreasing value. In Table 4.1, we describe a list of the different superedge attributes used in this paper. Note that the break of symmetry in the edge attributes makes the SPG a directed graph.

4.3.3 Superpoint Embedding

The goal of this stage is to compute a descriptor for every superpoint S_i by embedding it into a vector Z_i of fixed-size dimensionality d_z. Note that each superpoint is embedded

Attribute name	Size	Description
mean offset	3	$\text{mean}_{m \in \delta(S,R)}\ \delta_m$
offset deviation	3	$\text{std}_{m \in \delta(S,R)}\ \delta_m$
centroid offset	3	$\text{mean}_{i \in S}\ P_i - \text{mean}_{j \in R}\ P_j$
length ratio	1	$\log \text{length}\,(S)\,/\text{length}\,(R)$
surface ratio	1	$\log \text{surface}\,(S)\,/\text{surface}\,(R)$
volume ratio	1	$\log \text{volume}\,(S)\,/\text{volume}\,(R)$
point count ratio	1	$\log \lvert S\rvert / \lvert R\rvert$

Table 4.1 List of $d_f = 13$ superedge attributes characterizing the adjacency between two superpoints S and R.

in isolation; contextual information required for its reliable classification is provided only in the following stage by the means of graph convolutions.

Several deep learning-based methods have been proposed for this purpose recently. We choose PointNet (Qi et al., 2017a) for its remarkable simplicity, efficiency, and robustness. In PointNet, input points are first aligned by a Spatial Transformer Network (Jaderberg et al., 2015), independently processed by multi-layer perceptrons (MLPs), and finally max-pooled to summarize the shape.

In our case, input shapes are geometrically simple objects, which can be reliably represented by a small amount of points and embedded by a rather compact PointNet. This is important to limit the memory needed when evaluating many superpoints on current GPUs. In particular, we subsample superpoints on-the-fly down to $n_p = 128$ points to maintain efficient computation in batches and facilitate data augmentation. Superpoints of less than n_p points are sampled with replacement, which in principle does not affect the evaluation of PointNet due to its max-pooling. However, we observed that including very small superpoints of less than $n_{\text{minp}} = 40$ points in training harms the overall performance. Thus, embedding of such superpoints is set to zero so that their classification relies solely on contextual information.

In order for PointNet to learn spatial distribution of different shapes, each superpoint is rescaled to unit sphere before embedding. Points are represented by their normalized position P'_i, observations O_i, and geometric features F_i (since these are already available precomputed from the partitioning step). Furthermore, the original metric diameter of the superpoint is concatenated as an additional feature after PointNet max-pooling in order to stay covariant with shape sizes, see Section 4.3.5 for further details.

4.3.4 Contextual Segmentation

The final stage of the pipeline is to classify each superpoint S_i based on its embedding Z_i and its local surroundings within the SPG. Graph convolutions are naturally suited to this task. In this section, we explain the propagation model of our system.

Our approach builds on the ideas from Gated Graph Neural Networks (Li et al., 2016b) and Edge-Conditioned Convolutions (ECC) introduced in the previous chapter. The general idea is that superpoints refine their embedding according to pieces of information passed along superedges. Concretely, each superpoint S_i maintains its state hidden in a Gated Recurrent Unit (GRU) (Cho et al., 2014a). The hidden state is initialized with embedding Z_i and is then processed over several iterations (time steps) $t = 1 \dots T$. At each iteration t, a GRU takes its hidden state H_i^t and an incoming message M_i^t as input, and computes its new hidden state H_i^{t+1}. The incoming message M_i^t to superpoint i is computed as a weighted sum of hidden states H_j^t of neighboring superpoints j. The actual weighting for a superedge (j, i) depends on its attributes $E_{j,i}$, listed in Table 4.1. In particular, it is computed from the attributes by a multi-layer perceptron w, so-called Filter Generating Network. Formally:

$$\begin{aligned} H_i^{t+1} &= (1 - U_i^t) \odot Q_i^t + U_i^t \odot H_i^t \\ Q_i^t &= \tanh(X_i^{t,1} + R_i^t \odot H_i^{t,1}) \\ U_i^t &= \sigma(X_i^{t,2} + H_i^{t,2}) \\ R_i^t &= \sigma(X_i^{t,3} + H_i^{t,3}) \end{aligned} \tag{4.4}$$

$$\begin{aligned} (H_i^{t,1}, H_i^{t,2}, H_i^{t,3})^T &= \rho(W_h H_i^t + \mathbf{b}_h) \\ (X_i^{t,1}, X_i^{t,2}, X_i^{t,3})^T &= \rho(W_x X_i^t + \mathbf{b}_x) \end{aligned} \tag{4.5}$$

$$X_i^t = \sigma(W_g H_i^t + \mathbf{b}_g) \odot M_i^t \tag{4.6}$$

$$M_i^t = \text{mean}_{j|(j,i)\in\mathcal{E}}\ w(E_{j,i,\cdot}; W_e) \odot H_j^t \tag{4.7}$$

$$H_i^1 = Z_i \tag{4.8}$$

$$Y_i = W_o(H_i^1, \dots, H_i^{T+1})^T, \tag{4.9}$$

where $\odot$ is element-wise multiplication, $\sigma(\cdot)$ sigmoid function, and $W_\cdot$ and $\mathbf{b}_\cdot$ are trainable parameters shared among all GRUs. Equation 4.4 lists the standard GRU rules (Cho et al., 2014a) with its update gate U_i^t and reset gate R_i^t. To improve stability during training, in Equation 4.5 we apply Layer Normalization (Ba et al., 2016) defined as

$\rho(\mathbf{a}) := (\mathbf{a} - \text{mean}(\mathbf{a}))/(\text{std}(\mathbf{a}) + \epsilon)$ separately to linearly transformed input X_i^t and transformed hidden state H_i^t, with ϵ being a small constant. Finally, the model includes three interesting extensions in Equations 4.6–4.9, which we detail below.

Input Gating. We argue that GRU should possess the ability to down-weight (parts of) an input vector based on its hidden state. For example, GRU might learn to ignore its context if its class state is highly certain or to direct its attention to only specific feature channels. Equation 4.6 achieves this by gating message M_i^t by the hidden state before using it as input X_i^t.

Edge-Conditioned Convolution. ECC plays a crucial role in our model as it can dynamically generate filtering weights for any value of continuous attributes $E_{j,i}$ by processing them with a multi-layer perceptron w. In the original formulation, w regresses a weight matrix to perform matrix-vector multiplication $w(E_{j,i}; W_e)H_j^t$ for each edge. In this chapter, we propose a lightweight variant with lower memory requirements and fewer parameters, which is beneficial for datasets with few but large point clouds. Specifically, we regress only an edge-specific weight vector and perform element-wise multiplication as in Equation 4.7 (ECC-VV). Channel mixing, albeit in an edge-unspecific fashion, is postponed to Equation 4.5. Finally, let us remark that w is shared over time iterations and that self-loops are not necessary due to the existence of hidden states in GRUs.

State Concatenation. Inspired by DenseNet (Huang et al., 2017), we concatenate hidden states over all time steps and linearly transform them to produce segmentation logits Y_i in Equation 4.9. This allows to exploit the dynamics of hidden states due to increasing receptive field for the final classification.

4.3.5 Implementation Details

Training. While the geometric partitioning step is unsupervised, superpoint embedding and contextual segmentation are trained jointly in a supervised way with cross entropy loss. Superpoints are assumed to be semantically homogeneous and, consequently, assigned a hard ground truth label corresponding to the majority label among their contained points. We also considered using soft labels corresponding to normalized histograms of point labels and training with Kullback-Leibler (Kullback and Leibler, 1951) divergence loss. It performed slightly worse in our initial experiments, though.

Naive training on large SPGs may approach memory limits of current GPUs. We circumvent this issue by randomly subsampling the sets of superpoints at each iteration

and training on induced subgraphs, *i.e.* graphs composed of subsets of nodes and the original edges connecting them. Specifically, graph neighborhoods of order 3 are sampled to select at most 512 superpoints per SPG with more than n_{minp} points (smaller superpoints are not embedded). Note that as the induced graph is a union of small neighborhoods, relationships over many hops may still be formed and learned. This strategy also doubles as data augmentation and a strong regularization, together with randomized sampling of point clouds described in Section 4.3.3. Additional data augmentation is performed by randomly rotating superpoints around the vertical axis and jittering point features by Gaussian noise $N(0, 0.01)$ truncated to $[-0.05, 0.05]$.

We train using Adam (Kingma and Ba, 2015) with initial learning rate 0.01 and batch size 2, *i.e.* effectively up to 1024 superpoints per batch. For Semantic3D, we train for 500 epochs with stepwise learning rate decay of 0.7 at epochs 350, 400, and 450. For S3DIS, we train for 250 epochs with steps at 200 and 230. We clip gradients within $[-1, 1]$.

Testing. In modern deep learning frameworks, testing can be made very memory-efficient by discarding layer activations as soon as the follow-up layers have been computed. In practice, we were able to label full SPGs at once. To compensate for randomness due to subsampling of point clouds in PointNets, we average logits obtained over 10 runs with different seeds.

Voxelization. We pre-process input point clouds with voxelization subsampling by computing per-voxel mean positions and observations over a regular 3D grid (5 cm bins for Semantic3D and 3 cm bins for S3DIS dataset). The resulting semantic segmentation is interpolated back to the original point cloud in a nearest neighbor fashion. Voxelization helps decreasing the computation time and memory requirement, and improves the accuracy of the semantic segmentation by acting as a form of geometric and radiometric denoising as well. The quality of further steps is practically not affected, as superpoints are usually strongly subsampled for embedding during learning and inference anyway (Section 4.3.3).

Geometric Partition. We set regularization strength $\mu = 0.8$ for Semantic3D and $\mu = 0.03$ for S3DIS, which strikes a balance between semantic homogeneity of superpoints and the potential for their successful discrimination (S3DIS is composed of smaller semantic parts than Semantic3D). In addition to five geometric features f (linearity, planarity, scattering, verticality, elevation), we use color information O for clustering in S3DIS due to some classes being geometrically indistinguishable, such as boards or doors.

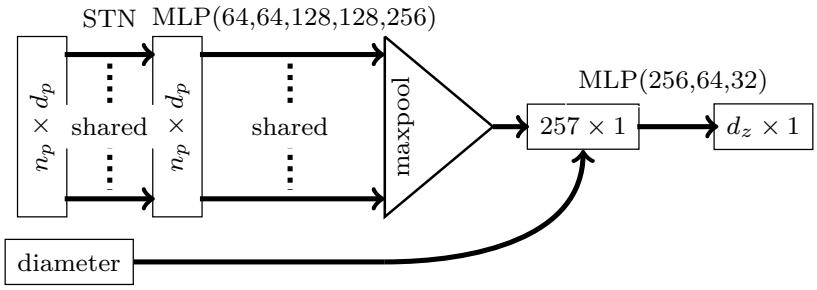


Fig. 4.3 The PointNet embedding n_p d_p-dimensional samples of a superpoint to a d_z-dimensional vector.

PointNet. We use a simplified shallow and narrow PointNet architecture with just a single Spatial Transformer Network (STN), see Figure 4.3. Input points are processed by a sequence of MLPs (widths 64, 64, 128, 128, 256) and max pooled to a single vector of 256 features. The scalar metric diameter is appended and the result further processed by a sequence of MLPs (widths 256, 64, d_z=32). A residual matrix $\Phi \in \mathbb{R}^{2\times 2}$ is regressed by STN and $(I + \Phi)$ is used to transform XY coordinates of input points as the first step. The architecture of STN is a "small PointNet" with 3 MLPs (widths 64, 64, 128) before max pooling and 3 MLPs after (widths 128, 64, 4). Batch Normalization (Ioffe and Szegedy, 2015) and ReLUs are used everywhere. Input points have d_p=11 dimensional features for Semantic3D (position P, color O, geometric features F), with 3 additional ones for S3DIS (room-normalized spatial coordinates, as in past work (Qi et al., 2017a)).

Segmentation Network. We use embedding dimensionality $d_z = 32$ and $T = 10$ iterations. ECC-VV is used for Semantic3D (there are only 15 point clouds even though the amount of points is large), while full ECC is used for S3DIS (large number of point clouds). Filter-generating network w is a MLP with 4 layers (widths 32, 128, 64, and 32 or 32^2 for ECC-VV or ECC) with ReLUs. Batch Normalization is used only after the third parametric layer. No bias is used in the last layer. Superedges have $d_f = 13$ dimensional attributes, normalized by mean subtraction and scaling to unit variance based on the whole training set.

4.4 Experiments

We evaluate our pipeline on two large point cloud segmentation benchmarks, Semantic3D (Hackel et al., 2017) and Stanford Large-Scale 3D Indoor Spaces (S3DIS) (Armeni et al.,

2016), on both of which we set the new state of the art. Furthermore, we perform a thorough ablation study of our pipeline in Section 4.4.3 and Section 4.4.4.

Even though the two data sets are quite different in nature (large outdoor scenes for Semantic3D, smaller indoor scanning for S3DIS), we use nearly the same model for both, described above. The deep model is rather compact and 6 GB of GPU memory is enough for both testing and training.

Performance is evaluated using three metrics: per-class intersection over union (IoU), per-class accuracy (Acc), and overall accuracy (OA), defined as the proportion of correctly classified points. We stress that the metrics are computed on the original point clouds, not on superpoints.

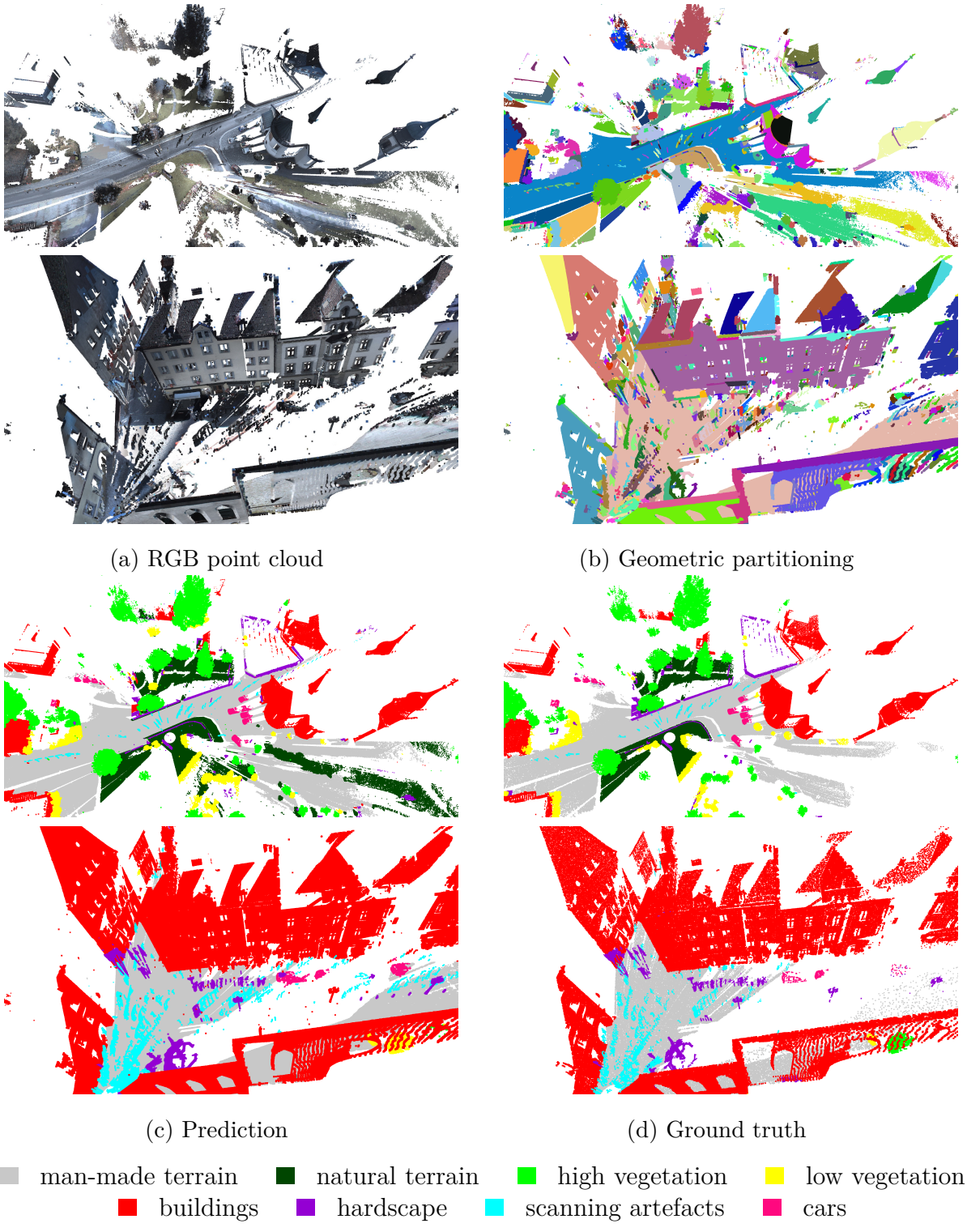
4.4.1 Semantic3D

Semantic3D (Hackel et al., 2017) is currently the largest available LiDAR dataset with over 3 billion points from a variety of urban and rural scenes. Each point has RGB and intensity values (the latter of which we do not use). The dataset consists of 15 training scans and 15 test scans with withheld labels. We also evaluate on the reduced set of 4 subsampled scans, as common in past work.

In Table 4.2, we provide the results of our algorithm compared to other recent works and in Figure 4.4, we provide qualitative results of our framework. Our framework improves significantly on the state of the art of semantic segmentation for this data set, *i.e.* by nearly 12 mIoU points on the reduced set and by nearly 9 mIoU points on the full set. In particular, we observe a steep gain on the "artefact" class. This can be explained by the ability of the partitioning algorithm to detect artifacts due to their singular shape, while they are hard to capture using snapshots, as suggested by (Boulch et al., 2017). Furthermore, these small object are often merged with the road when performing spatial regularization.

4.4.2 Stanford Large-Scale 3D Indoor Spaces

The S3DIS dataset (Armeni et al., 2016) consists of 3D RGB point clouds of six floors from three different buildings split into individual rooms. We evaluate our framework following two dominant strategies found in previous works. As advocated by Qi et al. (2017a), we perform 6-fold cross validation with micro-averaging, *i.e.* computing metrics once over the merged predictions of all test folds. Following Tchapmi et al. (2017), we also report the performance on the fifth fold only (Area 5), corresponding to a building not present in the other folds. Since some classes in this data set cannot be partitioned



(a) RGB point cloud

(b) Geometric partitioning

(c) Prediction

(d) Ground truth

Fig. 4.4 Example visualizations on Semantic3D. The colors are chosen randomly for each element of the partition.

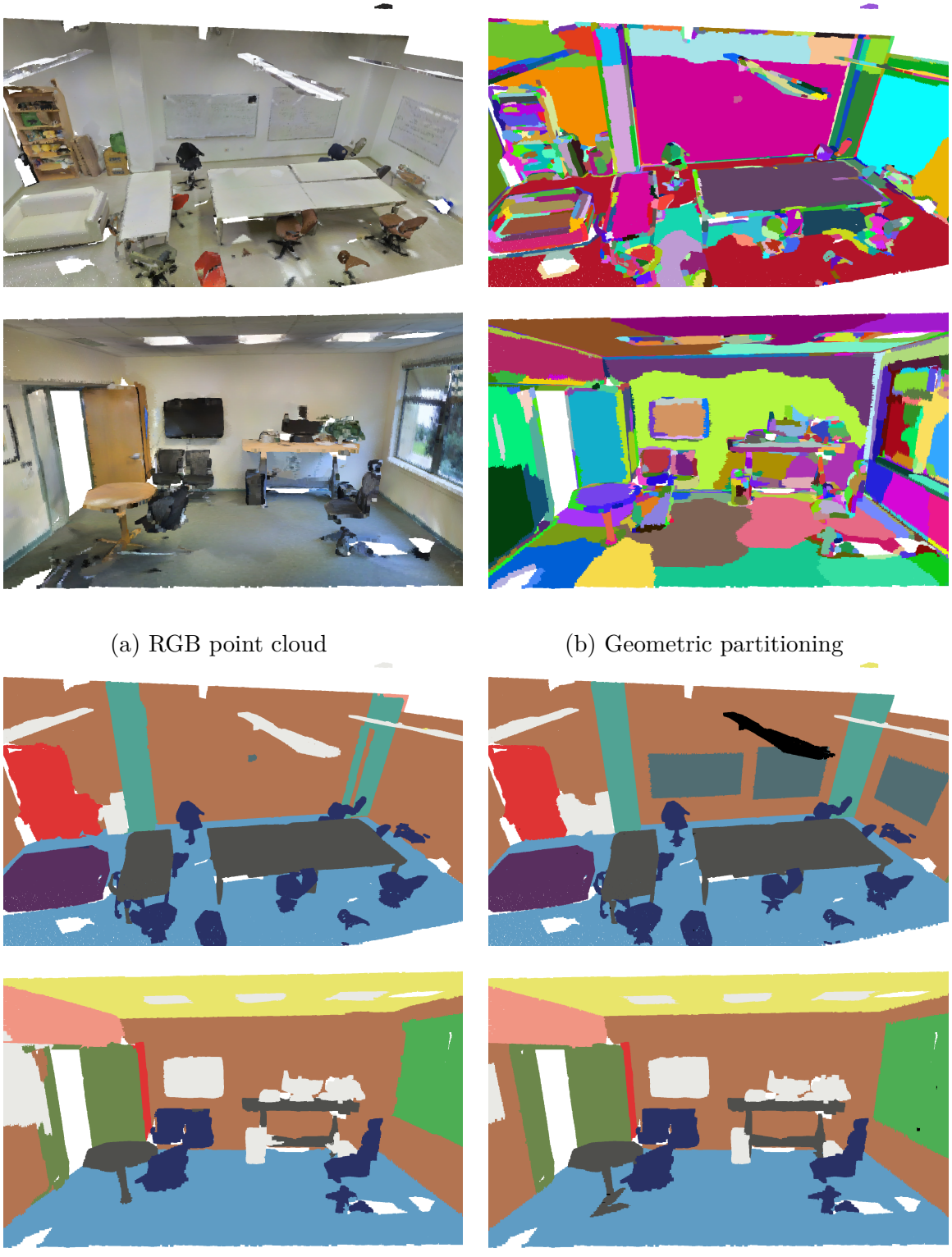


Fig. 4.5 Example visualizations on S3DIS. The colors are chosen randomly for each element of the partition.

	Method	OA	mIoU	MaTe	NaTe	HiVe	LoVe	Bu	Ha	ScAr	Ca
reduced set	Hackel et al. (2016)	86.2	54.2	89.8	74.5	53.7	26.8	88.8	18.9	36.4	44.7
	Lawin et al. (2017)	88.9	58.5	85.6	83.2	74.2	32.4	89.7	18.5	25.1	59.2
	Boulch et al. (2017)	88.6	59.1	82.0	77.3	79.7	22.9	91.1	18.4	37.3	64.4
	Tchapmi et al. (2017)	88.1	61.3	83.9	66.0	86.0	40.5	91.1	30.9	27.5	64.3
	SPG (Ours)	**94.0**	**73.2**	**97.4**	**92.6**	**87.9**	**44.0**	**93.2**	**31.0**	**63.5**	**76.2**
full set	Hackel et al. (2016)	85.0	49.4	91.1	69.5	32.8	21.6	87.6	25.9	11.3	55.3
	Boulch et al. (2017)	91.0	67.4	89.6	**79.5**	74.8	56.1	90.9	36.5	34.3	77.2
	SPG (Ours)	**92.9**	**76.2**	**91.5**	75.6	**78.3**	**71.7**	**94.4**	**56.8**	**52.9**	**88.4**

Table 4.2 Intersection over union metric for the different classes of the Semantic3D dataset: man-made terrain (MaTe), natural terrain (NaTe), high vegetation (HiVe), low vegetation , (LoVe), buildings (Bu), hardscape (Ha), scanning artefact (ScAr), cars (Ca). OA is the global accuracy, while mIoU refers to the unweighted average of IoU of each class. Full test set has 2 091 952 018 points, reduced test set 78 699 329 points.

purely using geometric features (such as boards or paintings on walls), we concatenate the color information O to the geometric features F for the partitioning step.

The quantitative results are displayed in Table 4.3, with qualitative results in Figure 4.5. S3DIS is a difficult dataset with hard to retrieve classes such as white boards on white walls and columns within walls. From the quantitative results we can see that our framework performs better than other methods on average. Notably, doors are able to be correctly classified at a higher rate than other approaches, as long as they are open. Indeed, doors are geometrically similar to walls, but their position with respect to the door frame allows our network to retrieve them correctly. On the other hand, the partition merges white boards with walls, depriving the network from the opportunity to even learn to classify them: the IoU of boards for theoretical perfect classification of superpoints (Section 4.4.3) is only 51.3.

Computation Time. In Table 4.4, we report computation time over the different steps of our pipeline for the inference on Area 5 measured on a 4 GHz CPU and GTX 1080 Ti GPU. While the bulk of time is spent on the CPU for partitioning and SPG computation, we show that voxelization as pre-processing leads to a significant speed-up as well as improved accuracy.

4.4.3 Segmentation Baselines

To demonstrate the advantage of our overall pipeline design, we compare it to several baselines. Due to the lack of public ground truth for test sets of Semantic3D, we evaluate

	Method	OA	mAcc	mIoU
Area 5	Qi et al. (2017a)	–	48.98	41.09
	Tchapmi et al. (2017)	–	57.35	48.92
	** Tatarchenko et al. (2018)	82.5	62.2	52.8
	** Huang et al. (2018)	–	59.42	51.93
	** Li et al. (2018a)	85.91	63.86	57.26
	SPG (Ours)	**86.38**	**66.50**	**58.04**
6-fold	Qi et al. (2017a) #	78.5	66.2	47.6
	Engelmann et al. (2017)	81.1	66.4	49.7
	** Huang et al. (2018)	–	66.45	56.47
	** Li et al. (2018a)	**88.14**	**75.61**	**65.39**
	SPG (Ours)	85.5	73.0	62.1

	Method	ceiling	floor	wall	beam	column	window	door	chair	table	bookcase	sofa	board	clutter
Area 5	Qi et al. (2017a)	88.80	97.33	69.80	**0.05**	3.92	46.26	10.76	52.61	58.93	40.28	5.85	26.38	33.22
	Tchapmi et al. (2017)	90.06	96.05	69.86	0.00	18.37	38.35	23.12	75.89	70.40	58.42	40.88	12.96	41.60
	** Huang et al. (2018)	**93.34**	**98.36**	**79.18**	0.00	15.75	45.37	50.10	65.52	67.87	22.45	52.45	**41.02**	43.64
	SPG (Ours)	89.35	96.87	78.12	0.0	**42.81**	**48.93**	**61.58**	**84.66**	**75.41**	**69.84**	**52.60**	2.10	**52.22**
6-fold	Qi et al. (2017a) #	88.0	88.7	69.3	42.4	23.1	47.5	51.6	42.0	54.1	38.2	9.6	29.4	35.2
	Engelmann et al. (2017)	90.3	92.1	67.9	44.7	24.2	52.3	51.2	47.4	58.1	39.0	6.9	30.0	41.9
	** Huang et al. (2018)	**92.48**	92.83	**78.56**	32.75	34.37	51.62	68.11	59.72	60.13	16.42	**50.22**	**44.85**	52.03
	SPG (Ours)	89.9	**95.1**	76.4	**62.8**	**47.1**	**55.3**	**68.4**	**73.5**	**69.2**	**63.2**	45.9	8.7	**52.9**

Table 4.3 Results on the S3DIS dataset on fold "Area 5" and micro-averaged over all 6 folds. Intersection over union is shown split per class. Concurrent baselines are denoted with a star. Hash denotes results obtained by Engelmann et al. (2017).

Step	Full cloud	2 cm	3 cm	4 cm
Voxelization	0	40	24	16
Feature computation	439	194	88	43
Geometric partition	3428	1013	447	238
SPG computation	3800	958	436	252
Inference	10×24	10×11	10×6	10×5
Total	7907	2315	1055	599
mIoU 6-fold	54.1	60.2	62.1	57.1

Table 4.4 Computation time in seconds for the inference on S3DIS Area 5 (68 rooms, 78 649 682 points) for different voxel sizes.

Model	mAcc	mIoU
Best	73.0	62.1
Perfect	92.7	88.2
Unary	50.8	40.0
iCRF	51.5	40.7
CRF − ECC	65.6	55.3
GRU13	69.1	58.5

Table 4.5 Comparison to various segmentation baselines on S3DIS (6-fold cross validation).

on S3DIS with 6-fold cross validation and show comparison of different models to our Best model in Table 4.5.

Performance Limits. The contribution of contextual segmentation can be bounded both from below and above. The lower bound (Unary) is estimated by training PointNet with $d_z = 13$ but otherwise the same architecture, denoted as PointNet13, to directly predict class logits, without SPG and GRUs. The upper bound (Perfect) corresponds to assigning each superpoint its ground truth label, and thus sets the limit of performance due to the geometric partition. We can see that contextual segmentation is able to win roughly 22 mIoU points over unaries, confirming its importance. Nevertheless, the learned model still has room of up to 26 mIoU points for improvement, while about 12 mIoU points are forfeited to the semantic inhomogeneity of superpoints.

CRFs. We compare the effect of our GRU+ECC-based network to CRF-based regularization. As a baseline (iCRF), we post-process Unary outputs by CRF inference over SPG connectivity with scalar transition matrix, as described by (Guinard and Landrieu, 2017). Next (CRF − ECC), we adapt CRF-RNN framework of Zheng et al. (2015) to general graphs with edge-conditioned convolutions (see 4.5 for details) and train it with PointNet13 end-to-end. Finally (GRU13), we modify Best to use PointNet13. We observe that iCRF barely improves accuracy (+1 mIoU), which is to be expected, since the partitioning step already encourages spatial regularity. CRF − ECC does better (+15 mIoU) due to end-to-end learning and use of edge attributes, though it is still below GRU13 (+18 mIoU), which performs more complex operations and does not enforce normalization of the embedding. Nevertheless, the 32 channels used in Best instead of the 13 used in GRU13 provide even more freedom for feature representation (+22 mIoU).

4.4.4 Ablation Studies

To better understand the influence of made design decisions and chosen hyperparameters, we perform ablation studies and present their results in Table 4.6. Concretely, we explore

the advantages of design choices by individually removing them from Best in order to compare the framework's performance with and without them.

a) Spatial Transformer Network. While STN makes superpoint embedding orientation invariant, the relationship with surrounding objects are still captured by superedges, which are orientation variant. In practice, STN helps by 4 mIoU points.

b) Geometric Features. Geometric features f_i are computed in the geometric partition step and can therefore be used in the following learning step for free. While PointNets could be expected to learn similar features from the data, this is hampered by superpoint subsampling, and therefore their explicit use helps (+4 mIoU).

c) State Concatenation. The advantage of state concatenation is compared to considering only the last hidden state in GRU for output ($Y_i = W_o H_i^{T+1}$). This accounts for about 5 mIoU points.

d) ECC Variant. ECC − VV decreases the performance on the S3DIS dataset by 3 mIoU points. Nevertheless, it has improved the performance on Semantic3D by 2 mIoU.

e) Sampling Superpoints. The main effect of subsampling SPG is regularization by data augmentation. Too small a sample size leads to disregarding contextual information (-4 mIoU) while too large a size leads to overfitting (-2 mIoU). Lower memory requirements at training is an extra benefit. There is no subsampling at test time.

f) Long-range Context. We observe that limiting the range of context information in SPG harms the performance. Specifically, capping distances in G_{vor} to 1 m (as used in PointNet (Qi et al., 2017a)) or 5 m (as used in SegCloud[1] (Tchapmi et al., 2017)) worsens the performance of our method (even more on our Semantic 3D validation set).

g) Input Gate. We evaluate the effect of input gating (IG) for GRUs as well as LSTM units. While a LSTM unit achieves higher score than a GRU (-3 mIoU), the proposed IG reverses this situation in favor of GRU (+1 mIoU). Unlike the standard input gate of LSTM, which controls the information flow from the hidden state and input to the cell, our IG controls the input even before it is used to compute all other gates.

h) Regularization Strength μ. We investigate the balance between superpoints' discriminative potential and their homogeneity controlled by parameter μ . We observe that the system is able to perform reasonably over a range of SPG sizes.

Superedge Attributes. Finally, in Table 4.7 we evaluate empirical importance of individual superedge attributes by removing them from Best. Although no single attribute is crucial, the most being offset deviation (+3 mIoU), without any superedge attributes[2] the network performs distinctly worse (-22 mIoU), falling back even below

[1]Furthermore, SegCloud divides the inference into cubes without overlap, possibly causing inconsistencies across boundaries.

[2]We perform homogeneous regularization by setting all superedge attributes to scalar 1.

a) Spatial transf.	no	**yes**		
mIoU	58.1	62.1		
b) Geometric features	no	**yes**		
mIoU	58.4	62.1		
c) State concatenation	no	**yes**		
mIoU	57.7	62.1		
d) ECC variant	ECC-VV	**ECC**		
mIoU	59.4	62.1		
e) Max superpoints	256	**512**	1024	
mIoU	57.9	62.1	60.4	
f) Superedge limit	1 m	5 m	∞	
mIoU	61.0	61.3	62.1	
g) Input gate	LSTM	LSTM+IG	GRU	**GRU+IG**
mIoU	61.0	61.0	57.5	62.1
h) Regularization μ	0.01	0.02	**0.03**	0.04
# superpoints	785 010	385 091	251 266	186 108
perfect mIoU	90.6	88.2	86.6	85.2
mIoU	59.1	59.2	62.1	58.8

Table 4.6 Ablation study of design decisions on S3DIS (6-fold cross validation). Our choices in bold.

iCRF to the level of Unary, which validates their design and the overall motivation for SPG.

4.5 Discussion

Relation to CRFs. In image segmentation, post-processing of convolutional outputs using Conditional Random Fields (CRFs) is widely popular. Several inference algorithms can be formulated as (recurrent) network layers amendable to end-to-end

Attribute set	mAcc	mIoU
Best	73.0	62.1
no superedge attributes	50.1	39.9
no mean offset	72.5	61.8
no offset deviation	71.7	59.3
no centroid offset	74.5	61.2
no len/surf/vol ratios	71.2	60.7
no point count ratio	72.7	61.7

Table 4.7 Ablation study of superedge attributes on S3DIS (6-fold cross validation).

learning (Schwing and Urtasun, 2015; Zheng et al., 2015), possibly with general pairwise potentials (Chandra and Kokkinos, 2016; Larsson et al., 2017; Lin et al., 2016). While our method of information propagation shares both these characteristics, our GRUs operate on d_z-dimensional intermediate feature space, which is richer and less constrained than low-dimensional vectors representing beliefs over classes, as also discussed in Gadde et al. (2016). Such enhanced access to information is motivated by the desire to learn a powerful representation of context, which goes beyond belief compatibilities, as well as the desire to be able to discriminate our often relatively weak unaries (superpixel embeddings).

We have empirically evaluated these claims in our experiments above. To provide a fair comparison, we have adapted CRF-RNN mean field inference introduced by Zheng et al. (2015) to use the same pairwise information as our model and denoted it CRF-ECC. Here we describe this adaptation in more detail. The original work proposed a dense CRF with unary potentials U_i $(= Z_i)$ and pairwise potentials Ψ defined to be a mixture of m Gaussian kernels as $\Psi_{ij} = \gamma \sum_m \mathbf{w}_m K_m(E_{i,j})$, where γ is label compatibility matrix, $\mathbf{w}$ are parameters, and K are fixed Gaussian kernels applied on edge features.

We replaced this definition of the pairwise term with a Filter generating network w parameterized with weights W_e, which generalizes the message passing and compatibility transform steps of Zheng et al. . Furthermore, we use superedge connectivity $\mathcal{E}$ instead of assuming a complete graph. The pseudo-code is listed in Algorithm 1. Its output are marginal probability distributions Q. In practice we run the inference for $T = 10$ iterations.

Algorithm 1 CRF-ECC

$Q_i \leftarrow \text{softmax}(U_i)$
while not converged **do**
  $\hat{Q}_i \leftarrow \sum_{j|(j,i)\in\mathcal{E}} w(E_{j,i}; W_e) Q_j$
  $\breve{Q}_i \leftarrow U_i - \hat{Q}_i$
  $Q_i \leftarrow \text{softmax}(\breve{Q}_i)$
end while

Adjacency Graphs. In this paper, we use two different adjacency graphs on points of the input clouds: G_{nn} in Section 4.3.1 and G_{vor} in Section 4.3.2. Indeed, different definitions of adjacency have different advantages. Voronoi adjacency is more suited to capture long-range relationships between points, which is beneficial for the SPG. Nearest neighbors adjacency tends not to connect objects separated by a small gap. This is desirable for the global energy but tends to produce a SPG with many small connected

components, decreasing embedding quality. Fixed radius adjacency should be avoided in general as it handles the variable density of LiDAR scans poorly.

Limitations. Despite the strong experimental results, it is worth discussing limitations of our method as well. Foremost, the assumed semantic homogeneity of the obtained partitions does not necessarily hold up once applied to real world data. The framework has been evaluated on high-end scanner acquisitions and may not immediately work on point clouds coming from structure-from-motion reconstructions and especially cheap depth cameras, such as Kinect.

While the regularization strength μ provides a powerful and an intuitive handle for the user to adapt the method for a particular dataset, we believe a trainable partitioning step or at least partitioning of learned features might go even further and allow the model to learn *e.g.* SLAM-specific features and provide a better partition. Very recently, several attempts at learning features for clustering have been presented: Wolf et al. (2017) predict altitudes for watershed algorithm, Tu et al. (2018) predict affinities for computing superpixels, and Verelst et al. (2018) looks into backpropagation through SLIC, a popular superpixel algorithm. This makes us speculate that features for global energy term in Equation 4.1 could be learned in a similar spirit. Nevertheless, end-to-end training may remain challenging from the engineering point of view on the current hardware simply due to the scale of the full problem.

4.6 Conclusion

We presented a deep learning framework for performing semantic segmentation of large point clouds based on a partition into simple shapes. We showed that SPGs allow us to use effective deep learning tools, which would not be able to handle the data volume otherwise. Our method significantly improves on the state of the art on two publicly available datasets. Our experimental analysis suggested that future improvements can be made in both partitioning and learning deep contextual classifiers. The source code has been published at https://github.com/loicland/superpoint_graph.

Chapter 5

Generation of Small Graphs

5.1 Introduction

Generative models capture data distribution in an unsupervised way and may allow us to draw samples from it. In the recent years, deep generative models have gone through fast-paced advances leading to massive rise in the realism of generated samples. They have found use in a myriad of problems and applications, including image super-resolution (Ledig et al., 2017), domain or modality transfer (Isola et al., 2017; Reed et al., 2016), anomaly detection (Schlegl et al., 2017), or proposing new molecules in drug discovery as in this chapter. The most popular model families include generative adversarial networks (GANs) (Goodfellow et al., 2014), posing the training process as a game between a generator and a discriminator, variational autoencoders (VAEs) (Kingma and Welling, 2013), approximating data likelihood by its variational lower bound, and auto-regressive networks (Sutskever et al., 2011), learning the conditional distribution of each next sampling step.

Progress in deep learning on graphs, on the other hand, has been nearly exclusively concentrated on learning graph embedding tasks before the submission of our work, *i.e.* encoding an input graph into a vector representation. Hence, it is an intriguing question how one can transfer the achievements in generative models for images and text to the domain of graphs, *i.e.* their decoding from a vector representation.

However, learning to generate graphs is a difficult problem for methods based on gradient optimization, as graphs are discrete structures. Unlike sequence (text) generation, graphs can have arbitrary connectivity and it is not trivial to choose how to linearize their construction in a sequence of steps. On the other hand, learning the order for incremental construction involves discrete decisions, which are not differentiable.

In this work, we propose to sidestep these hurdles by having the decoder output a probabilistic fully-connected graph of a predefined maximum size directly at once. In a probabilistic graph, the existence of nodes and edges, as well as their attributes, are modeled as independent random variables. The method is formulated in the framework of variational autoencoders by Kingma and Welling (2013).

We demonstrate our method, coined GraphVAE, in cheminformatics on the task of molecule generation. Molecular datasets are a challenging but convenient testbed for our generative model, as they easily allow for both qualitative and quantitative tests of decoded samples. Past generative models have operated on textual input data only. An advantage of a graph-based decoder compared to text-based decoder is the possibility to predict detailed attributes of atoms and bonds in addition to the base structure.

This chapter is largely based on our ICLR workshop paper (Simonovsky and Komodakis, 2018b) and follow-up ICANN 2018 publication (Simonovsky and Komodakis, 2018a). While our method is applicable for generating smaller graphs only and its practical performance leaves space for improvement, we believe our work at the time of its submission was an important initial step towards powerful and efficient graph decoders with the following contributions:

- We present one of the first methods for graph generation using deep learning. Our method can generate graphs of variable though limited sizes without any step-wise supervision.
- We evaluate on two molecular datasets, offering a better selection of valid but at the same time diverse samples on QM9 dataset (Ramakrishnan et al., 2014) than previous text generation-based methods. We also suggest a conditional setting of our model as an alternative way of controlling the molecule generation process.

5.2 Related work

De Novo Molecular Design. In the pharmaceutical industry, generative models may become promising for computational design of molecules fulfilling certain criteria (such as solubility) and optimizing certain desirable properties (such as binding strength to a protein). The hypothesis is that optimization in the continuous space of models or embeddings is easier than in the original discrete chemical space. In the context of deep generative models, the field has explored mainly two strategies for navigating the chemical space, either by fine-tuning a generative model, *e.g.* with simple hill-climbing (Segler et al., 2018) or with reinforcement learning (Olivecrona et al., 2017), or by optimization

in the continuous embedding space (Gómez-Bombarelli et al., 2016). Here, we suggest to directly drive certain properties by introducing a conditional version of the decoder, also concurrently proposed by Li et al. (2018b).

While molecules have an intuitive and richer representation as graphs, the field has had to resort to textual representations with fixed syntax, *e.g.* so-called SMILES strings (Weininger, 1988), to exploit recent progress made in text generation with RNNs (Gómez-Bombarelli et al., 2016; Olivecrona et al., 2017; Segler et al., 2018). As their syntax is brittle, many invalid strings tend to be generated, which has been recently addressed by Kusner et al. (2017) by incorporating context-free grammar rules into decoding to mask out invalid steps, effectively generating a parse tree. While encouraging, their approach does not guarantee semantic (chemical) validity, similarly to our method. In a concurrent work, Dai et al. (2018) propose to structure the string decoding space even more tightly by using attribute grammars, which allow to capture semantic rules as well and thus generate higher proportions of valid outputs.

Discrete Data Decoders. Text is the most common discrete representation. Generative models there are usually trained in a maximum likelihood fashion by teacher forcing (Williams and Zipser, 1989), which avoids the need to backpropagate through output discretization by feeding the ground truth instead of the past sample at each step. Bengio et al. (2015) argued this may lead to expose bias, *i.e.* possibly reduced ability to recover from own mistakes. Recently, efforts have been made to overcome this problem. Notably, computing a differentiable approximation using Gumbel distribution (Kusner and Hernández-Lobato, 2016) or bypassing the problem by learning a stochastic policy in reinforcement learning (Yu et al., 2017). Our work deals with the non-differentiability problem by formulating the loss on a probabilistic graph.

Graph Decoders. Related work pre-dating deep learning includes random graphs (Barabási and Albert, 1999; Erdos and Rényi, 1960), stochastic blockmodels (Snijders and Nowicki, 1997), or state transition matrix learning (Gong and Xiang, 2003).

Graph generation was largely unexplored in deep learning before 2018. The closest work to ours is by Johnson (2017), who incrementally constructs a probabilistic (multi)graph as a world representation according to a sequence of input sentences to answer a query. While our model also outputs a probabilistic graph, we do not assume having a prescribed order of construction transformations available and we formulate the learning problem as an autoencoder.

Xu et al. (2017) learns to produce a scene graph from an input image. They construct a graph from a set of object proposals, provide initial embeddings to each node and edge, and use message passing to obtain a consistent prediction. In contrast, our method is a generative model which produces a probabilistic graph from a single vector, without specifying the number of nodes or the structure explicitly.

Multiple concurrent and follow up methods have then appeared, which we briefly review here. These works do not sample edges in an independent way and outperform the method presented here.

Li et al. (2018b) relax several assumptions made by Johnson (2017) and present the first self-contained auto-regressive graph decoder. The graphs are incrementally constructed by a sequence of decisions on adding nodes and edges, predicted by RNN running on embeddings continuously updated throughout the construction. The network is trained with random or fixed node orderings and the authors report difficulties due to extensive length of sequences. You et al. (2018b) improve on this by proposing a more concise graph building strategy with graph-level and edge-level RNNs and introduce a breadth-first-search node-ordering scheme, which limits the number of possible permutations and steps, to significantly improve scalability allowing to handle hundreds of nodes and edges. Li et al. (2018d) propose even more lightweight construction strategy (graph-level RNN or even casting it as Markov process) and use depth-first ordering with random noise. You et al. (2018a) then casted the construction as a Markov process in a reinforcement learning framework with a GAN loss.

Samanta et al. (2018) also train to generate probabilistic graphs in a VAE framework. However, they model graph embedding as a set of node embeddings rather than a single opaque vector, which allows them to avoid graph matching for loss computation but also means that node embeddings must be sampled independently during inference and the number of them is not part of the latent code. Edge probabilities are modeled as functions of node embeddings, similarly to Kipf and Welling (2016b), and invalid steps are masked out during edge decoding to improve the ratio of semantically valid graphs, similarly to Kusner et al. (2017). Liu et al. (2018) further builds up on Samanta et al. (2018) and, among other minor improvements, adds conditioning on the embedding of the current state of the graph to the edge sampling function, making it a Markov process unlike in Li et al. (2018b); You et al. (2018b), where decisions depend on the generation history. Cao and Kipf (2018) generate graphs at once as in our work but use GAN framework for training, which alleviates the need to use graph matching but makes the model more susceptible to mode collapse.

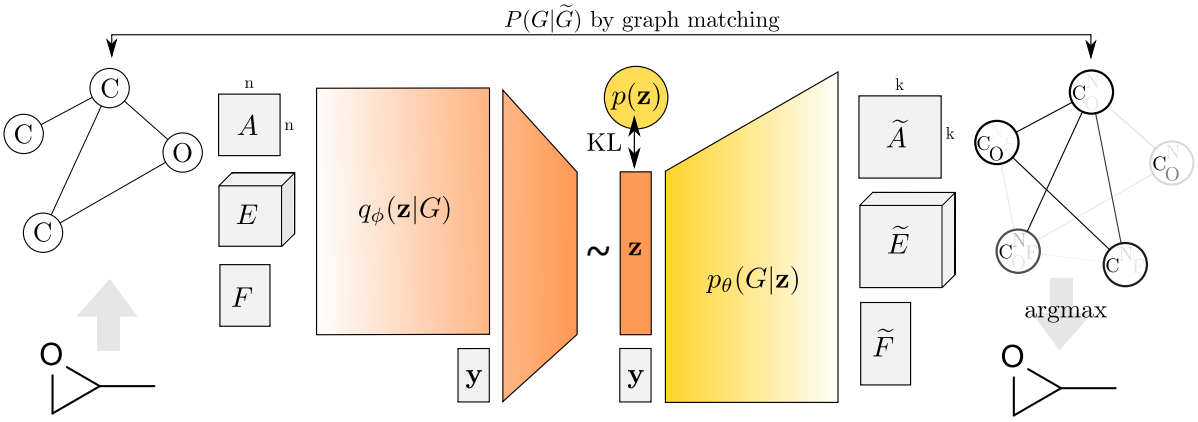


Fig. 5.1 Illustration of the proposed variational graph autoencoder. Starting from a discrete attributed graph $G = (A, E, F)$ on n nodes (*e.g.* a representation of propylene oxide), stochastic graph encoder $q_\phi(\mathbf{z}|G)$ embeds the graph into continuous representation $\mathbf{z}$. Given a point in the latent space, our novel graph decoder $p_\theta(G|\mathbf{z})$ outputs a probabilistic fully-connected graph $\widetilde{G} = (\widetilde{A}, \widetilde{E}, \widetilde{F})$ on predefined $k \geq n$ nodes, from which discrete samples may be drawn. The process can be conditioned on label $\mathbf{y}$ for controlled sampling at test time. Reconstruction ability of the autoencoder is facilitated by approximate graph matching for aligning G with $\widetilde{G}$.

For the specific task of molecular graph generation, Jin et al. (2018) builds on the insight that molecules can be decomposed into trees of fragments. From a combined tree and graph embedding, a molecule is decoded by reconstructing the tree, followed by a series of greedy choices of compatible fragments with a scoring function involving the latent embedding as a context. The masking of invalid combinations prevents creation of invalid molecules.

5.3 Method

We approach the task of graph generation by devising a neural network able to translate vectors in a continuous code space to graphs. Our main idea is to output a probabilistic fully-connected graph and use a standard graph matching algorithm to align it to the ground truth. The proposed method is formulated in the framework of variational autoencoders (VAE) by Kingma and Welling (2013), although other forms of regularized autoencoders would be equally suitable (Li et al., 2015; Makhzani et al., 2015). We briefly recapitulate VAE below and continue with introducing our novel graph decoder together with an appropriate objective.

5.3.1 Variational Autoencoder

Let $G = (A, E, F)$ be a graph specified with its adjacency matrix A, edge attribute tensor E, and node attribute matrix F. We wish to learn an encoder and a decoder to map between the space of graphs G and their continuous embedding $\mathbf{z} \in \mathbb{R}^c$, see Figure 5.1. In the probabilistic setting of a VAE, the encoder is defined by a variational posterior $q_\phi(\mathbf{z}|G)$ and the decoder by a generative distribution $p_\theta(G|\mathbf{z})$, where ϕ and θ are learned parameters. Furthermore, there is a prior distribution $p(\mathbf{z})$ imposed on the latent code representation as a regularization; we use a simplistic isotropic Gaussian prior $p(\mathbf{z}) = N(0, I)$. The whole model is trained by minimizing the upper bound on negative log-likelihood $-\log p_\theta(G)$ (Kingma and Welling, 2013):

$$\mathcal{L}(\phi, \theta; G) = \mathbb{E}_{q_\phi(\mathbf{z}|G)}[-\log p_\theta(G|\mathbf{z})] + \mathrm{KL}[q_\phi(\mathbf{z}|G)||p(\mathbf{z})] \tag{5.1}$$

The first term of $\mathcal{L}$, the reconstruction loss, enforces high similarity of sampled generated graphs to the input graph G. The second term, KL-divergence, regularizes the code space to allow for sampling of $\mathbf{z}$ directly from $p(\mathbf{z})$ instead from $q_\phi(\mathbf{z}|G)$ later. The dimensionality of $\mathbf{z}$ is usually fairly small so that the autoencoder is encouraged to learn a high-level compression of the input instead of learning to simply copy any given input. While the regularization is independent on the input space, the reconstruction loss must be specifically designed for each input modality. In the following, we introduce our graph decoder together with an appropriate reconstruction loss.

5.3.2 Probabilistic Graph Decoder

Graphs are discrete objects, ultimately. While this does not pose a challenge for encoding, demonstrated by the recent developments in graph convolution networks, graph generation was an open problem until early 2018. In a related task of text sequence generation, the currently dominant approach is character-wise or word-wise prediction (Bowman et al., 2016). However, graphs can have arbitrary connectivity and there is no clear way how to linearize their construction in a sequence of steps[1]. On the other hand, iterative construction of discrete structures during training without step-wise supervision involves discrete decisions, which are not differentiable and therefore problematic for back-propagation.

[1] While algorithms for canonical graph orderings are available (McKay and Piperno, 2014), Vinyals et al. (2015) empirically found out that the linearization order matters when learning on sets. However, see also the evolution of node ordering strategies used by follow up works discussed in Section 5.2

Fortunately, the task can become much simpler if we restrict the domain to the set of all graphs on maximum k nodes, where k is fairly small (in practice up to the order of tens). Under this assumption, handling dense graph representations is still computationally tractable. We propose to make the decoder output a probabilistic fully-connected graph $\widetilde{G} = (\widetilde{A}, \widetilde{E}, \widetilde{F})$ on k nodes at once. This effectively sidesteps both problems mentioned above.

In probabilistic graphs, the existence of nodes and edges is modeled as Bernoulli variables, whereas node and edge attributes are multinomial variables. While not discussed in this work, continuous attributes could be easily modeled as Gaussian variables represented by their mean and variance. We assume all variables to be independent.

Each tensor of the representation of $\widetilde{G}$ has thus a probabilistic interpretation. Specifically, the predicted adjacency matrix $\widetilde{A} \in [0, 1]^{k \times k}$ contains both node probabilities $\widetilde{A}_{a,a}$ and edge probabilities $\widetilde{A}_{a,b}$ for nodes $a \neq b$. The edge attribute tensor $\widetilde{E} \in \mathbb{R}^{k \times k \times d_e}$ indicates class probabilities for edges and, similarly, the node attribute matrix $\widetilde{F} \in \mathbb{R}^{k \times d_n}$ contains class probabilities for nodes.

The decoder itself is deterministic. Its architecture is a simple multi-layer perceptron (MLP) with three outputs in its last layer. Sigmoid activation function is used to compute $\widetilde{A}$, whereas edge- and node-wise softmax is applied to obtain $\widetilde{E}$ and $\widetilde{F}$, respectively. At test time, we are often interested in a (discrete) point estimate of $\widetilde{G}$, which can be obtained by taking edge- and node-wise argmax in $\widetilde{A}, \widetilde{E}$, and $\widetilde{F}$. Note that this can result in a discrete graph on less than k nodes.

5.3.3 Reconstruction Loss

Given a particular instance of a discrete input graph G on $n \leq k$ nodes and its probabilistic reconstruction $\widetilde{G}$ on k nodes, evaluation of Equation 5.1 requires computation of likelihood $p_\theta(G|\mathbf{z}) = P(G|\widetilde{G})$.

Since no particular ordering of nodes is imposed in either $\widetilde{G}$ or G and matrix representation of graphs is not invariant to permutations of nodes, comparison of two graphs is hard. However, approximate graph matching described further in Subsection 5.3.4 can obtain a binary assignment matrix $X \in \{0, 1\}^{k \times n}$, where $X_{a,i} = 1$ only if node $a \in \widetilde{G}$ is assigned to $i \in G$ and $X_{a,i} = 0$ otherwise.

Knowledge of X allows to map information between both graphs. Specifically, input adjacency matrix is mapped to the predicted graph as $A' = XAX^T$, whereas the predicted node attribute matrix and slices of edge attribute matrix are transferred to the input graph as $\widetilde{F}' = X^T\widetilde{F}$ and $\widetilde{E}'_{\cdot,\cdot,l} = X^T\widetilde{E}_{\cdot,\cdot,l}X$. The maximum likelihood estimates,

i.e. cross-entropy, of respective variables are as follows:

$$\begin{aligned}
\log p(A'|\mathbf{z}) &= 1/k \sum_{a} A'_{a,a} \log \widetilde{A}_{a,a} + (1 - A'_{a,a}) \log(1 - \widetilde{A}_{a,a}) + \\
&\quad + 1/k(k-1) \sum_{a \neq b} A'_{a,b} \log \widetilde{A}_{a,b} + (1 - A'_{a,b}) \log(1 - \widetilde{A}_{a,b}) \\
\log p(F|\mathbf{z}) &= 1/n \sum_{i} \log F^T_{i,\cdot} \widetilde{F}'_{i,\cdot} \\
\log p(E|\mathbf{z}) &= 1/(||A||_1 - n) \sum_{i \neq j} \log E^T_{i,j,\cdot} \widetilde{E}'_{i,j,\cdot}
\end{aligned} \tag{5.2}$$

where we assumed that F and E are encoded in one-hot notation. The formulation considers existence of both matched and unmatched nodes and edges but attributes of only the matched ones. Furthermore, averaging over nodes and edges separately has shown beneficial in training as otherwise the edges dominate the likelihood. The overall reconstruction loss is a weighed sum of the previous terms:

$$-\log p(G|\mathbf{z}) = -\lambda_A \log p(A'|\mathbf{z}) - \lambda_F \log p(F|\mathbf{z}) - \lambda_E \log p(E|\mathbf{z}) \tag{5.3}$$

5.3.4 Graph Matching

The goal of (second-order) graph matching is to find correspondences $X \in \{0,1\}^{k \times n}$ between nodes of graphs G and $\widetilde{G}$ based on the similarities of their node pairs $S : (i,j) \times (a,b) \to \mathbb{R}^+$ for $i,j \in G$ and $a,b \in \widetilde{G}$. It can be expressed as integer quadratic programming problem of similarity maximization over X and is typically approximated by relaxation of X into continuous domain: $X^* \in [0,1]^{k \times n}$ (Cho et al., 2014b). For our use case, the similarity function is defined as follows:

$$\begin{aligned}
S((i,j),(a,b)) &= (E^T_{i,j,\cdot} \widetilde{E}_{a,b,\cdot}) A_{i,j} \widetilde{A}_{a,b} \widetilde{A}_{a,a} \widetilde{A}_{b,b} [i \neq j \wedge a \neq b] + \\
&\quad + (F^T_{i,\cdot} \widetilde{F}_{a,\cdot}) \widetilde{A}_{a,a} [i = j \wedge a = b]
\end{aligned} \tag{5.4}$$

The first term evaluates similarity between edge pairs and the second term between node pairs, $[\cdot]$ being the Iverson bracket. Note that the scores consider both feature compatibility ($\widetilde{F}$ and $\widetilde{E}$) and existential compatibility ($\widetilde{A}$), which has empirically led to more stable assignments during training. To summarize the motivation behind both Equations 5.3 and 5.4, our method aims to find the best graph matching and then further improve on it by gradient descent on the loss. Given the stochastic way of training deep networks, we argue that solving the matching step only approximately is sufficient. This

is conceptually similar to the approach for learning to output unordered sets by (Vinyals et al., 2015), where the closest ordering of the training data is sought.

In practice, we are looking for a graph matching algorithm robust to noisy correspondences which can be easily implemented on GPU in batch mode. Max-pooling matching (MPM) by Cho et al. (2014b) is a simple but effective algorithm following the iterative scheme of power methods. It can be used in batch mode if similarity tensors are zero-padded, *i.e.* $S((i,j),(a,b)) = 0$ for $n < i,j \leq k$, and the amount of iterations is fixed.

Max-pooling matching outputs continuous assignment matrix X^*. Unfortunately, attempts to directly use X^* instead of X in Equation 5.3 performed badly, as did experiments with direct maximization of X^* or soft discretization with softmax or straight-through Gumbel softmax (Jang et al., 2016) or using Sinkhorn iterations to arrive at doubly-stochastic matrices (Sinkhorn and Knopp, 1967). We therefore discretize X^* to X using Hungarian algorithm to obtain a strict one-on-one mapping[2]. While this operation is non-differentiable, gradient can still flow to the decoder directly through the loss function and training convergence proceeds without problems. Note that this approach is often taken in works on object detection, *e.g.* Stewart et al. (2016), and more recently point cloud generation Fan et al. (2017), where a set of detections needs to be matched to a set of ground truth entities and treated as fixed before computing a differentiable loss.

5.3.5 Further Details

Encoder. A feed forward network with edge-conditioned graph convolutions (ECC) as presented in Chapter 3 is used as encoder, although any other graph embedding method is applicable. As our edge attributes are categorical, a single linear layer for the filter generating network in ECC is sufficient. Due to smaller graph sizes no pooling is used in encoder except for the global one, for which we employ gated pooling by Li et al. (2016b). As usual in VAE, we formulate the encoder as probabilistic and enforce Gaussian distribution of $q_\phi(\mathbf{z}|G)$ by having the last encoder layer outputs $2c$ features interpreted as mean and variance, allowing to sample $\mathbf{z}_l \sim N(\mu_l(G), \sigma_l(G))$ for $l \in 1,..,c$ using the re-parameterization trick (Kingma and Welling, 2013).

Disentangled Embedding. In practice, rather than random drawing of graphs, one often desires more control over the properties of generated graphs. In such case, we follow

[2]Some predicted nodes are not assigned for $n < k$. Our current implementation performs this step on CPU although a GPU version has been published (Date and Nagi, 2016).

Sohn et al. (2015) and condition both encoder and decoder on label vector $\mathbf{y}$ associated with each input graph G. Decoder $p_\theta(G|\mathbf{z}, \mathbf{y})$ is fed a concatenation of $\mathbf{z}$ and $\mathbf{y}$, while in encoder $q_\phi(\mathbf{z}|G, \mathbf{y})$, $\mathbf{y}$ is concatenated to every node's features just before the graph pooling layer. If the size of latent space c is small, the decoder is encouraged to exploit information in the label. If the latent space is large, however, one might better explicitly minimize mutual information between the condition and the embedding, an important detail which we nevertheless leave for future work.

Max-Pooling Matching We briefly review max-pooling matching algorithm of Cho et al. (2014b) for clarity. In its relaxed form, a continuous correspondence matrix $X^* \in [0, 1]^{k \times n}$ between nodes of graphs G and $\widetilde{G}$ is determined based on similarities of node pairs $i, j \in G$ and $a, b \in \widetilde{G}$ represented as matrix elements $S_{ia;jb} \in \mathbb{R}^+$.

Let $\mathbf{x}^*$ denote the column-wise replica of X^* (with overloaded indexing notation $\mathbf{x}^*_{ia} = X^*_{ia}$). The relaxed graph matching problem is expressed as quadratic programming task $\mathbf{x}^* = \arg\max_{\mathbf{x}} \mathbf{x}^T S \mathbf{x}$ such that $\sum_{i=1}^{n} \mathbf{x}_{ia} \leq 1$, $\sum_{a=1}^{k} \mathbf{x}_{ia} \leq 1$, and $\mathbf{x} \in [0, 1]^{kn}$. The optimization strategy of choice is derived to be equivalent to the power method with iterative update rule $\mathbf{x}^{t+1} = S\mathbf{x}^{(t)}/||S\mathbf{x}^t||_2$. The starting correspondences $\mathbf{x}^0$ are initialized as uniform and the rule is iterated until convergence; in our use case we run for a fixed amount of iterations.

In the context of graph matching, the matrix-vector product $S\mathbf{x}$ can be interpreted as sum-pooling over match candidates: $\mathbf{x}_{ia} \leftarrow \mathbf{x}_{ia} S_{ia;ia} + \sum_{j \in N_i} \sum_{b \in N_a} \mathbf{x}_{jb} S_{ia;jb}$, where N_i and N_a denote the set of neighbors of node i and a. The authors argue that this formulation is strongly influenced by uninformative or irrelevant elements and propose a more robust max-pooling version, which considers only the best pairwise similarity from each neighbor: $\mathbf{x}_{ia} \leftarrow \mathbf{x}_{ia} S_{ia;ia} + \sum_{j \in N_i} \max_{b \in N_a} \mathbf{x}_{jb} S_{ia;jb}$.

5.4 Evaluation

We demonstrate our method for the task of molecule generation by evaluating on two large public datasets of organic molecules, QM9 and ZINC.

5.4.1 Application in Cheminformatics

Quantitative evaluation of generative models of images and texts has been troublesome (Theis et al., 2015), as it very difficult to measure realness of generated samples in an automated and objective way. Thus, researchers frequently resort there to qualitative

evaluation and embedding plots. However, qualitative evaluation of graphs can be very unintuitive for humans to judge unless the graphs are planar and fairly simple.

Fortunately, we found graph representation of molecules, as undirected graphs with atoms as nodes and bonds as edges, to be a convenient testbed for generative models. On one hand, generated graphs can be easily visualized in standardized structural diagrams. On the other hand, chemical validity of graphs, as well as many further properties a molecule can fulfill, can be checked using software packages (`SanitizeMol` in RDKit) or simulations. This makes both qualitative and quantitative tests possible.

Chemical constraints on compatible types of bonds and atom valences make the space of valid graphs complicated and molecule generation challenging. In fact, a single addition or removal of edge or change in atom or bond type can make a molecule chemically invalid. Comparably, flipping a single pixel in MNIST-like number generation problem is of no issue.

To help the network in this application, we introduce three remedies. First, we make the decoder output symmetric $\widetilde{A}$ and $\widetilde{E}$ by predicting their (upper) triangular parts only, as undirected graphs are sufficient representation for molecules. Second, we use prior knowledge that molecules are connected and, at test time only, construct maximum spanning tree on the set of probable nodes $\{a : \widetilde{A}_{a,a} \geq 0.5\}$ in order to include its edges (a, b) in the discrete pointwise estimate of the graph even if $\widetilde{A}_{a,b} < 0.5$ originally. Third, we do not generate Hydrogen explicitly and let it be added as "padding" during chemical validity check [3].

5.4.2 QM9 Dataset

QM9 dataset (Ramakrishnan et al., 2014) contains about 134k organic molecules of up to 9 heavy (non Hydrogen) atoms with 4 distinct atomic numbers and 4 bond types, we set $k = 9$, $d_e = 4$ and $d_n = 4$. We set aside 10k samples for testing and 10k for validation (model selection).

We compare our unconditional model to the character-based generator of Gómez-Bombarelli et al. (2016) (CVAE) and the grammar-based generator of Kusner et al. (2017) (GVAE). We used the code and architecture in Kusner et al. (2017) for both baselines, adapting the maximum input length to the smallest possible. In addition, we

[3] The implicit Hydrogen count is calculated as the difference between the valence of a particular atom and the sum of its bonds to other heavy (non-Hydrogen) atoms. Typically, the number of heavy atoms is approximately half the total number of atoms in organic molecules. Thus, implicit Hydrogen makes molecule representations considerably smaller, saving memory and running time. In addition, such a padding is forgiving to decoders generating less bonds than required by the respective valences. Note that this does not affect the reconstruction error, though.

demonstrate a conditional generative model for an artificial task of generating molecules given a histogram of heavy atoms as 4-dimensional label $\mathbf{y}$, the success of which can be easily validated.

Setup. The encoder has two graph convolutional layers (32 and 64 channels) with identity connection, batchnorm, and ReLU; followed by the graph-level output formulation in Equation 7 of Li et al. (2016b) with auxiliary networks being a single fully connected layer (FCL) with 128 output channels; finalized by a FCL outputting (μ, σ). The decoder has 3 FCLs (128, 256, and 512 channels) with batchnorm and ReLU; followed by parallel triplet of FCLs to output graph tensors. We set $c = 40$, $\lambda_A = \lambda_F = \lambda_E = 1$, batch size 32, 75 MPM iterations and train for 25 epochs with Adam with learning rate 1e-3 and β_1=0.5.

Embedding Visualization. To visually judge the quality and smoothness of the learned embedding $\mathbf{z}$ of our model, we may traverse it in two ways: along a slice and along a line. For the former, we randomly choose two c-dimensional orthonormal vectors and sample $\mathbf{z}$ in regular grid pattern over the induced 2D plane. For the latter, we randomly choose two molecules $G^{(1)}, G^{(2)}$ of the same label from test set and interpolate between their embeddings $\mu(G^{(1)}), \mu(G^{(2)})$. This also evaluates the encoder, and therefore benefits from low reconstruction error.

We plot two planes in Figure 5.2, for a frequent label (left) and a less frequent label in QM9 (right). Both images show a varied and fairly smooth mix of molecules. The left image has many valid samples broadly distributed across the plane, as presumably the autoencoder had to fit a large portion of database into this space. The right exhibits stronger effect of regularization, as valid molecules tend to be only around center.

An example of several interpolations is shown in Figure 5.3. We can find both meaningful (1st, 2nd and 4th row) and less meaningful transitions, though many samples on the lines do not form chemically valid compounds.

Decoder Quality Metrics. The quality of a conditional decoder can be evaluated by the validity and variety of generated graphs. For a given label $\mathbf{y}^{(l)}$, we draw $n_s = 10^4$ samples $\mathbf{z}^{(l,s)} \sim p(\mathbf{z})$ and compute the discrete point estimate of their decodings $\hat{G}^{(l,s)} = \arg\max p_\theta(G|\mathbf{z}^{(l,s)}, \mathbf{y}^{(l)})$.

Let $V^{(l)}$ be the list of chemically valid molecules from $\hat{G}^{(l,s)}$ and $C^{(l)}$ be the list of chemically valid molecules with atom histograms equal to $\mathbf{y}^{(l)}$. We are interested in ratios $\text{Valid}^{(l)} = |V^{(l)}|/n_s$ and $\text{Accurate}^{(l)} = |C^{(l)}|/n_s$. Furthermore, let $\text{Unique}^{(l)} =$

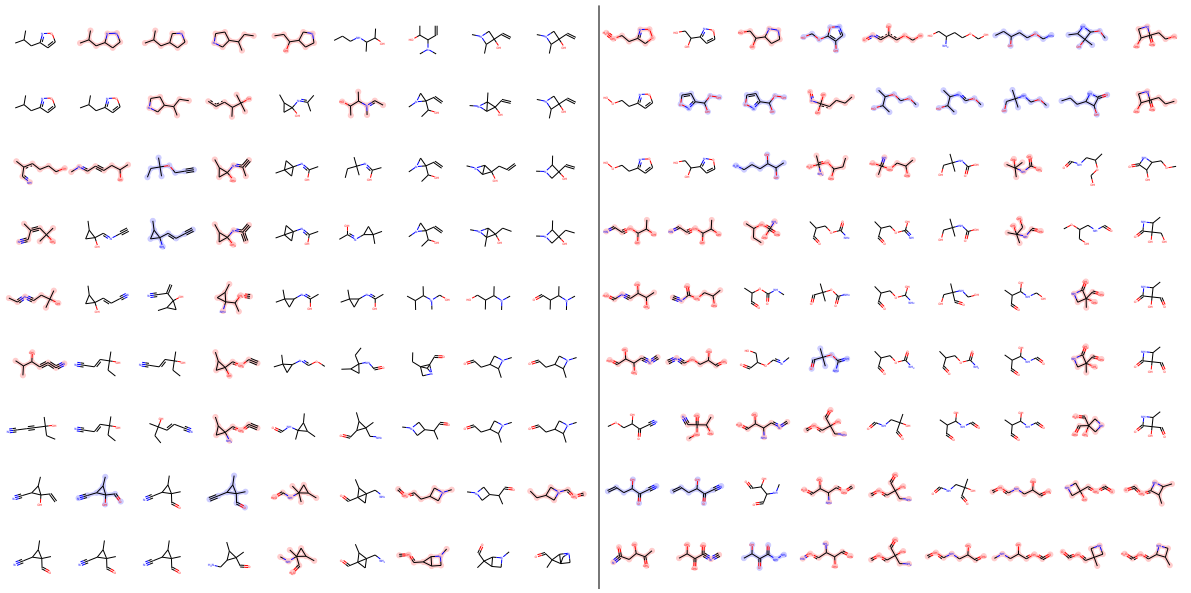


Fig. 5.2 Decodings of latent space points of a conditional model sampled over a random 2D plane in $\mathbf{z}$-space of $c = 40$ (within 5 units from center of coordinates). Left: Samples conditioned on 7x Carbon, 1x Nitrogen, 1x Oxygen (12% QM9). Right: Samples conditioned on 5x Carbon, 1x Nitrogen, 3x Oxygen (2.6% QM9). Color legend as in Figure 5.3.

$|\text{set}(C^{(l)})|/|C^{(l)}|$ be the fraction of unique correct graphs and $\text{Novel}^{(l)} = 1 - |\text{set}(C^{(l)}) \cap \text{QM9}|/|\text{set}(C^{(l)})|$ the fraction of novel out-of-dataset graphs; we define $\text{Unique}^{(l)} = 0$ and $\text{Novel}^{(l)} = 0$ if $|C^{(l)}| = 0$. Finally, the introduced metrics are aggregated by frequencies of labels in QM9, *e.g.* $\text{Valid} = \sum_l \text{Valid}^{(l)} \text{freq}(\mathbf{y}^{(l)})$. Unconditional decoders are evaluated by assuming there is just a single label, therefore Valid = Accurate.

In Table 5.1, we can see that on average 50% of generated molecules are chemically valid and, in the case of conditional models, about 40% have the correct label which the decoder was conditioned on. Larger embedding sizes c are less regularized, demonstrated by a higher number of Unique samples and by lower accuracy of the conditional model, as the decoder is forced less to rely on actual labels. The ratio of Valid samples shows less clear behavior, likely because the discrete performance is not directly optimized for. For all models, it is remarkable that about 60% of generated molecules are out of the dataset, *i.e.* the network has never seen them during training.

Looking at the baselines, CVAE can output only very few valid samples as expected, while GVAE generates the highest number of valid samples (60%) but of very low variance (less than 10%). Additionally, we investigate the importance of graph matching by using identity assignment X instead and thus learning to reproduce particular node

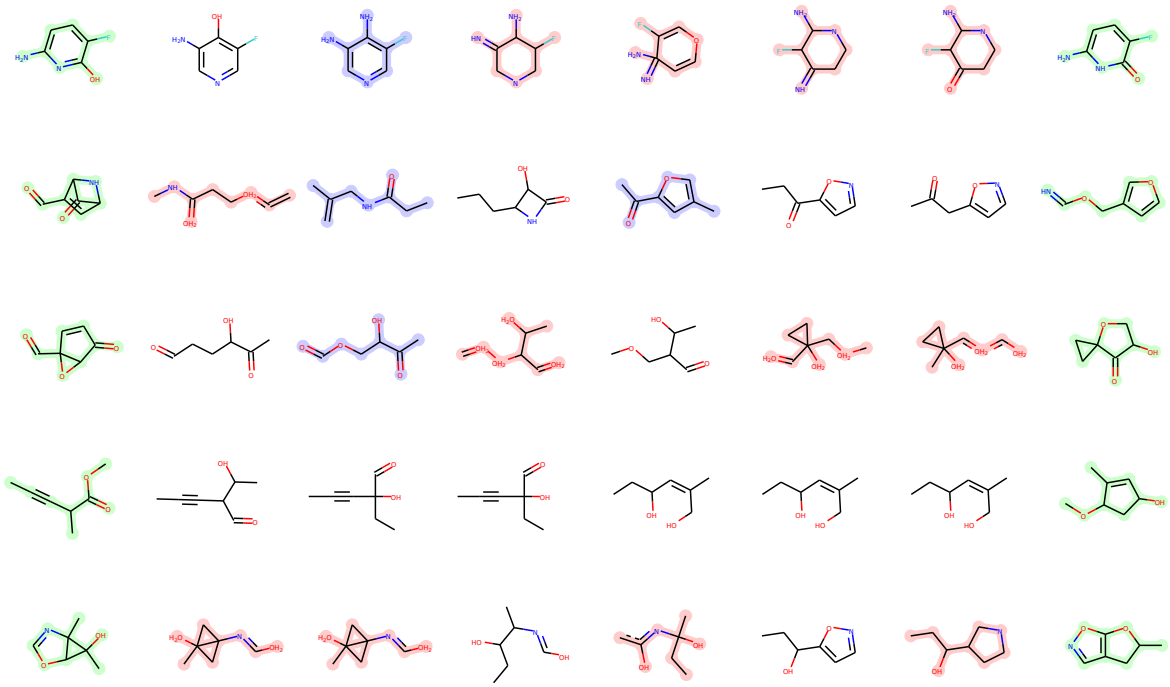


Fig. 5.3 Linear interpolation between row-wise pairs of randomly chosen molecules in $\mathbf{z}$-space of $c = 40$ in a conditional model. Color legend: encoder inputs (green), chemically invalid graphs (red), valid graphs with wrong label (blue), valid and correct (white).

permutations in the training set, which correspond to the canonical ordering of SMILES strings from RDKit. This ablated model (denoted as NoGM in Table 5.1) produces many valid samples of lower variety and, surprisingly, outperforms GVAE in this regard. In comparison, our model can achieve good performance in both metrics at the same time.

Likelihood. Besides the application-specific metric introduced above, we also report evidence lower bound (ELBO) commonly used in VAE literature, which corresponds to $-\mathcal{L}(\phi, \theta; G)$ in our notation. In Table 5.1, we state mean bounds over test set, using a single $\mathbf{z}$ sample per graph. We observe both reconstruction loss and KL-divergence decrease due to larger c providing more freedom. However, there seems to be no strong correlation between ELBO and Valid, which makes model selection somewhat difficult.

Implicit Node Probabilities. Our decoder assumes independence of node and edge probabilities, which allows for isolated nodes or edges. Making further use of the fact that molecules are connected graphs, here we investigate the effect of making node

		$\log p_\theta(G\|\mathbf{z})$	ELBO	Valid	Accurate	Unique	Novel
Cond.	Ours $c = 20$	-0.578	-0.722	0.565	0.467	0.314	0.598
	Ours $c = 40$	-0.504	-0.617	0.511	0.416	0.484	0.635
	Ours $c = 60$	-0.492	-0.585	0.520	0.406	0.583	0.613
	Ours $c = 80$	-0.475	-0.557	0.458	0.353	0.666	0.661
Unconditional	Ours $c = 20$	-0.660	-0.916	0.485	0.485	0.457	0.575
	Ours $c = 40$	-0.537	-0.744	0.542	0.542	0.618	0.617
	Ours $c = 60$	-0.486	-0.656	0.517	0.517	0.695	0.570
	Ours $c = 80$	-0.482	-0.628	0.557	0.557	0.760	0.616
	NoGM $c = 80$	-2.388	-2.553	0.810	0.810	0.241	0.610
	CVAE $c = 60$	–	–	0.103	0.103	0.675	0.900
	GVAE $c = 20$	–	–	0.602	0.602	0.093	0.809

Table 5.1 Performance on conditional and unconditional QM9 models evaluated by mean test-time reconstruction log-likelihood ($\log p_\theta(G|\mathbf{z})$), mean test-time evidence lower bound (ELBO), and decoding quality metrics (Section 5.4.2). Baselines CVAE (Gómez-Bombarelli et al., 2016) and GVAE (Kusner et al., 2017) are listed only for the embedding size with the highest Valid.

probabilities a function of edge probabilities. Specifically, we consider the probability for node a as that of its most probable edge: $\widetilde{A}_{a,a} = \max_b \widetilde{A}_{a,b}$.

The evaluation on QM9 in Table 5.2 shows a clear improvement in Valid, Accurate, and Novel metrics in both the conditional and unconditional setting. However, this is paid for by lower variability and higher reconstruction loss. This indicates that while the new constraint is useful, the model cannot fully cope with it.

Unregularized Autoencoder. The regularization in VAE works against achieving perfect reconstruction of training data, especially for small embedding sizes. To understand the reconstruction ability of our architecture, we train it as unregularized in this section, *i.e.* with a deterministic encoder and without KL-divergence term in Equation 5.1.

Unconditional models for QM9 achieve mean test log-likelihood $\log p_\theta(G|\mathbf{z})$ of roughly -0.37 (about -0.50 for the implicit node probability model) for all $c \in \{20, 40, 60, 80\}$. While these log-likelihoods are significantly higher than in Tables 5.1 and 5.2, our architecture cannot achieve perfect reconstruction of inputs. We were successful to increase training negative log-likelihood to zero only on fixed small training sets of hundreds of examples, where the network could overfit. This indicates that the network has problems finding generally valid rules for assembly of output tensors.

		$\log p_\theta(G\|\mathbf{z})$	ELBO	Valid	Accurate	Unique	Novel
Cond.	Ours/imp $c = 20$	-0.784	-0.919	*0.572*	*0.482*	0.238	*0.718*
	Ours/imp $c = 40$	-0.671	-0.776	*0.611*	*0.518*	0.307	*0.665*
	Ours/imp $c = 60$	-0.618	-0.714	*0.566*	*0.448*	0.416	*0.710*
	Ours/imp $c = 80$	-0.627	-0.713	*0.583*	*0.451*	0.475	*0.681*
Uncond.	Ours/imp $c = 20$	-0.857	-1.091	*0.533*	*0.533*	0.228	*0.610*
	Ours/imp $c = 40$	-0.737	-0.932	*0.562*	*0.562*	0.420	*0.758*
	Ours/imp $c = 60$	-0.634	-0.797	*0.587*	*0.587*	0.459	*0.730*
	Ours/imp $c = 80$	-0.642	-0.777	*0.571*	*0.571*	0.520	*0.719*

Table 5.2 Performance on conditional and unconditional QM9 models with implicit node probabilities. Improvement with respect to Table 5.1 is emphasized in italics.

5.4.3 ZINC Dataset

ZINC dataset (Irwin et al., 2012) contains about 250k drug-like organic molecules of up to 38 heavy atoms with 9 distinct atomic numbers and 4 bond types, we set $k = 38$, $d_e = 4$ and $d_n = 9$ and use the same split strategy as with QM9. We investigate the degree of scalability of an unconditional generative model.

Setup. The setup is equivalent as for QM9 but with a wider encoder (64, 128, 256 channels).

Decoder Quality Metrics. Our best model with $c = 40$ has archived Valid $= 0.135$, which is clearly worse than for QM9. Using implicit node probabilities brought no improvement. For comparison, CVAE failed to generated any valid sample, while GVAE achieved Valid $= 0.357$ (models provided by Kusner et al. (2017), $c = 56$).

We attribute such a low performance to a generally much higher chance of producing a chemically-relevant inconsistency (number of possible edges growing quadratically). To confirm the relationship between performance and graph size k, we kept only graphs not larger than $k = 20$ nodes, corresponding to 21% of ZINC, and obtained Valid $= 0.341$ (and Valid $= 0.185$ for $k = 30$ nodes, 92% of ZINC). To verify that the problem is likely not caused by our proposed graph matching loss, we synthetically evaluate it in the following.

Matching Robustness. Robust behavior of graph matching using our similarity function S is important for good performance of GraphVAE. Here we study graph matching in isolation to investigate its scalability. To that end, we add Gaussian noise

Noise	$k = 15$	$k = 20$	$k = 25$	$k = 30$	$k = 35$	$k = 40$
$\epsilon_{A,E,F} = 0$	99.55	99.52	99.45	99.4	99.47	99.46
$\epsilon_A = 0.4$	90.95	89.55	86.64	87.25	87.07	86.78
$\epsilon_A = 0.8$	82.14	81.01	79.62	79.67	79.07	78.69
$\epsilon_E = 0.4$	97.11	96.42	95.65	95.90	95.69	95.69
$\epsilon_E = 0.8$	92.03	90.76	89.76	89.70	88.34	89.40
$\epsilon_F = 0.4$	98.32	98.23	97.64	98.28	98.24	97.90
$\epsilon_F = 0.8$	97.26	97.00	96.60	96.91	96.56	97.17

Table 5.3 Mean accuracy of matching ZINC graphs to their noisy counterparts in a synthetic benchmark as a function of maximum graph size k.

$N(0, \epsilon_A), N(0, \epsilon_E), N(0, \epsilon_F)$ to each tensor of input graph G, truncating and renormalizing to keep their probabilistic interpretation, to create its noisy version G_N. We are interested in the quality of matching between self, $P[G, G]$, using noisy assignment matrix X between G and G_N. The advantage to naive checking X for identity is the invariance to permutation of equivalent nodes.

In Table 5.3 we vary k and ϵ for each tensor separately and report mean accuracies (computed in the same fashion as losses in Equation 5.3) over 100 random samples from ZINC with size up to k nodes. While we observe an expected fall of accuracy with stronger noise, the behavior is fairly robust with respect to increasing k at a fixed noise level, the most sensitive being the adjacency matrix. Note that accuracies are not comparable across tensors due to different dimensionalities of random variables. We may conclude that the quality of the matching process is not a major hurdle to scalability.

5.5 Discussion

Having presented one of the first works on graph generation, we are aware of range of limitations and possible extensions, many of them already addressed by the follow-up works reviewed in Section 5.2.

Size Restriction. As each edge is predicted individually, the proposed model is expected to be useful only for generating small graphs. This is due to growth of GPU memory requirements and number of parameters in $O(k^2)$ as well as matching complexity in $O(k^4)$, with small decrease in quality for high values of k. In Section 5.4

we demonstrated results for up to $k = 38$. Nevertheless, in many applications, such as drug design, even generation of small graphs is still very useful.

Independence Assumption. We model the existence and attribute of each node and edge with an independent random variable, which allows us to easily take a point-wise estimate of the decoded probabilistic graph at the end. However, this assumption has turned out as too brittle in the case of imperfect predictions, as we have not observed strong correlation between ELBO and the ratio of valid samples in the experimental section, which suggests a mismatch between the training objective (probabilistic graph) and the desired objective (discrete graph). We have presented and evaluated two amendments to partially mitigate this problem: post-processing with a maximum spanning tree and making node probabilities implicit. Subsequent work has adopted sequential sampling approaches, which allows to explicitly control for validity at each step (Jin et al., 2018; Liu et al., 2018; Samanta et al., 2018).

Simplistic Prior. We use isotropic Gaussian prior $p(\mathbf{z}) = N(0, I)$ for regularization. However, there is a large body of work suggesting more complex distributions, such as using normalization flows on prior (Dinh et al., 2016) or posterior (Kingma et al., 2016), or better formulations of VAEs, notably Wasserstein autoencoders (Tolstikhin et al., 2018). While we expect that building on more powerful frameworks would lead to better results, it would not solve the limitations discussed above.

5.6 Conclusion

In this chapter we addressed the problem of generating graphs from a continuous embedding in the context of variational autoencoders. We evaluated our method on two molecular datasets of different maximum graph size. While we achieved to learn embedding of reasonable quality on small molecules, our decoder had a hard time capturing complex chemical interactions for larger molecules. Nevertheless, we believe our method has been an important initial step towards more powerful decoders and has sparked interest in the community.

Chapter 6

Conclusion

The goal of this thesis was to study and mutually relate deep learning-based architectures for processing graph-structured data and, primarily, to propose several novel contributions to this nascent but fast evolving field. Throughout the past pages, we have seen that many core concepts (*e.g.* convolutions, superpixels, or autoencoders) can be taken over from the domain of regular grids and repurposed for irregular data. We have touched upon both discriminative and generative models. The field being so young and challenging, it has offered us many unexplored paths to tread; many of them having a dead end but others leading us to good experimental results and novel approaches to tasks such as point cloud processing, graph classification or graph generation. Nevertheless, the proposed methods are hardly perfect and we tried to remain critical in stating their limitations as well.

Deep learning is a booming topic and with the justified popularity of preprint repositories, there are often multiple lines of work proposed independently. Specifically in deep learning on graphs, these papers often come from several, originally rather separate communities working in signal processing and vision, in biochemistry, in natural language processing, or in information retrieval. We have thus attempted at providing a somewhat fused but not exhaustive picture in our literature reviews and at helping the reader to better understand the context of our work.

In the next paragraphs, we will recapitulate our contributions and conclude with an outlook on possible future directions of work.

The first main contribution was the introduction of a crucial primitive for processing data structured on variable attributed graphs, called Edge-Conditioned Convolutions (ECC). The major insight was learning how to generate filters for every edge attribute rather than learning such filters directly. We have shown that the standard discrete convolution on grids as well as many formulations proposed previously can be seen as a special case of ECC. Our other contributions rely on having such an operation available.

In addition, this was the first time graph convolutions were applied to processing of (neighborhood graphs constructed on) point clouds.

The second main contribution was the formulation of a superpixel-inspired intermediate point cloud representation, called SuperPoint Graph (SPG). The major insight was that considering point clouds as a collection of interconnected simple shapes, rather than individual points, allows ECC to start off from more informative embeddings and scale up to massive point clouds consisting of millions of points without major sacrifice in fine details. Important ingredients were also the very efficient partitioning algorithm and the definition of rich contextual features captured in SPG. Having combined the parts together and performed an extensive ablation study, we were able to significantly improve on the state of the art methods on the two largest publicly available datasets covering outdoor and indoor scenes.

Finally, the third main contribution was the presentation of one the first works on deep-learning based graph generation. The major insight was to use approximate graph matching for aligning predictions of an autoencoder with its inputs, which allowed us to decode graphs with variable but upper-bounded number of nodes. Our motivation was to avoid several challenges associated with step-wise construction. While the method could outperform past text generation-based autoencoders on a small-sized dataset, the practical performance left space for improvement, quickly addressed by consecutive works. Nevertheless, we believe our work has helped to spark interest for graph generation in the community and has looked into the direction of generation of graph tensors at once.

In summary, we have ventured a journey from graphs to their embeddings and back. We hope that the research developed in the context of this thesis has provided the community with new insights about some of the challenges in the emergent field of deep learning on graphs and brings us one more step closer to solving real-world problems arising in the field's manifold applications, such as drug discovery, remote sensing, social network understanding, or autonomous systems, for the benefit of all of us.

6.1 Future Directions

Deep learning on graphs is an exciting field with large applicability, offering many future direction to follow. Here we expand just on some of them.

Dynamic Graphs. In practice, it is often the case that graph data is not static but rather evolves in time and, in fact, the precise timing of events can be important for the task at hand. This can include changes in node signal, edge attributes, or in the

number of nodes and their connectivity. For example, users communicate on social networks, a moving car updates information about its surroundings, new facts are included in knowledge bases, or - as we have seen in concurrent work in Chapter 5 - graphs are sequentially constructed. How to update the representation efficiently and how incorporate timing information, on what level of granularity, and whether to maintain history or treat the system as Markovian are just a few questions, the answering of which is likely application-driven. Extending graph embedding techniques in this direction could open up a wide range of exciting application domains.

With a few exceptions (Yuan et al., 2017), past DL research has dealt only with time series on graphs with fixed structure, such as for traffic prediction (Li et al., 2018c) or for modeling the dynamics of physical systems represented as complete graphs (Battaglia et al., 2016; Kipf et al., 2018). Extending the latter works to larger-scale systems is an open question; one way could be to adapt the graph structure according to object distances or attention scores, which is an application of graph editing discussed below.

Graph Editing. Graph generation is a very recent topic, which has just started to get explored in DL. While the primary real-world application has been in molecule generation so far, we believe that many more are awaiting. Past work has already looked into generating graphs as an intermediate representation, see Johnson (2017) for reasoning tasks and Brockschmidt et al. (2018) for code generation. We are particularly excited about the applications of auto-regressive methods to graph editing, where one starts from an existing graph rather than from scratch. Note that only the case of adding edges (link prediction) has been well researched. Graph editing could find use in simulating evolution of phenomena in time, generating similar molecules and translating measurements into plausible molecules, predicting chemical reactions, or locally optimizing certain combinatorial properties of graphs. In drug discovery, finding the right ligands for a specific binding site on a protein (analogous to finding a key for a keyhole) could been seen as graph inpainting. Also, graph generative models could be used as samplers for various reinforcement learning-based approaches for automating the architecture design process of deep networks, as *e.g.* in Baker et al. (2017). Ultimately, the variety of tasks that can be achieved with generative models in images can serve as an inspiration.

Node-Conditioned Convolutions. In ECC, we conditioned message function m on edge attributes, which allowed specializing the communication between each pair of nodes. However, all nodes shared the same learned weight matrices. Thus, it could be

interesting to condition update function u (in the form of RNN) on node attributes as well. We suspect this might be useful in cases where there are distinct node types present in a graph, such as in biological networks (with nodes for proteins, diseases, chemical compounds or symptoms) or perhaps multi-agent systems.

Packages and Benchmarks. Open source implementations and public datasets are one of the driving forces of the momentum behind deep learning. While many graph convolution methods have been open sourced, including ours, their programming interfaces are fairly diverse and written in different DL frameworks, which makes benchmarking and selection of the right method for an application laborious. It would be thus beneficial to initiate a single, up-to-date package containing a range of published methods, implemented with reusable modules and accessible under a unified interface. This should also make exploring novel model variants easier.

A related point is the desire for better benchmarks for evaluating graph representation learning approaches. Many popular datasets currently used in the community are tiny and prone to overfitting or not providing enough variety. For example, not all graphs may exhibit the degree of locality and stationarity assumed by convolutional methods[1]. Evaluating models under different conditions may often lead to (empirical) research insights. For example, a recently introduced benchmark for molecular machine learning (Wu et al., 2018) has shown that traditional methods are not yet outperformed by a selection of graph convolution models on several task.

Theoretical Understanding. The lack of theoretical understanding is a popular complaint against deep learning, and spatial graph convolution models are no exception. By leaving the graph spectral theory (Defferrard et al., 2016) and dynamic system theory (Scarselli et al., 2009) behind, we could learn feed-forward models working on arbitrary graphs but also forfeited theoretical insights, proposing innovations based mostly on intuition and experiments. The future work should look more into the properties and guarantees offered by various message passing and aggregation methods. Specifically, the convergence at training time and the stability and the way of spreading information at inference time should be of interest. A source of inspiration could be studies performed for RNNs as in *e.g.* Chen et al. (2018); Tallec and Ollivier (2018).

[1]The fitness of different architectures to different graph datasets could be potentially measured in a way similar to the work of Ulyanov et al. (2017), who have shown that the structure of networks is sufficient to capture a great deal of low-level image statistics.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. (2012). SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282.

Agaskar, A. and Lu, Y. M. (2013). A spectral graph uncertainty principle. *IEEE Trans. Information Theory*, 59(7):4338–4356.

Alvar, N. S., Zolfaghari, M., and Brox, T. (2016). Orientation-boosted voxel nets for 3D object recognition. *CoRR*, abs/1604.03351.

Anand, A., Koppula, H. S., Joachims, T., and Saxena, A. (2013). Contextually guided semantic labeling and search for three-dimensional point clouds. *The International Journal of Robotics Research*, 32(1):19–34.

Armeni, I., Sener, O., Zamir, A. R., Jiang, H., Brilakis, I., Fischer, M., and Savarese, S. (2016). 3D semantic parsing of large-scale indoor spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Atwood, J. and Towsley, D. (2016). Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*.

Ba, L. J., Kiros, R., and Hinton, G. E. (2016). Layer normalization. *CoRR*, abs/1607.06450.

Baker, B., Gupta, O., Naik, N., and Raskar, R. (2017). Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D. J., and Kavukcuoglu, K. (2016). Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4502–4510.

Belkin, M. and Niyogi, P. (2001). Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems (NIPS)*, pages 585–591.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1171–1179.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

Bojchevski, A. and Günnemann, S. (2017). Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. *CoRR*, abs/1707.03815.

Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. (2018). NetGAN: Generating graphs via random walks. *CoRR*, abs/1803.00816.

Borgwardt, K. M. and Kriegel, H. (2005). Shortest-path kernels on graphs. In *IEEE International Conference on Data Mining (ICDM)*, pages 74–81.

Boscaini, D., Masci, J., Rodolà, E., and Bronstein, M. M. (2016). Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3189–3197.

Boulch, A., Saux, B. L., and Audebert, N. (2017). Unstructured point cloud semantic labeling using deep segmentation networks. In *Eurographics Workshop on 3D Object Retrieval*, volume 2.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Józefowicz, R., and Bengio, S. (2016). Generating sentences from a continuous space. In *CoNLL*, pages 10–21.

Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239.

Brabandere, B. D., Jia, X., Tuytelaars, T., and Gool, L. V. (2016). Dynamic filter networks. In *Advances in Neural Information Processing Systems (NIPS)*.

Brockschmidt, M., Allamanis, M., Gaunt, A. L., and Polozov, O. (2018). Generative code modeling with graphs. *CoRR*, abs/1805.08490.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203.

Cao, N. D. and Kipf, T. (2018). Molgan: An implicit generative model for small molecular graphs. *CoRR*, abs/1805.11973.

Chandra, S. and Kokkinos, I. (2016). Fast, exact and multi-scale inference for semantic image segmentation with deep Gaussian CRFs. In *IEEE European Conference on Computer Vision (ECCV)*.

Chen, M., Pennington, J., and Schoenholz, S. S. (2018). Dynamical isometry and a mean field theory of RNNs: Gating enables signal propagation in recurrent neural networks. *arXiv preprint arXiv:1806.05394*.

Chen, T., Dai, B., Liu, D., and Song, J. (2014). Performance of global descriptors for velodyne-based urban object recognition. In *IEEE Intelligent Vehicles Symposium Proceedings*, pages 667–673.

Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014a). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Cho, M., Sun, J., Duchenne, O., and Ponce, J. (2014b). Finding matches in a haystack: A max-pooling strategy for graph matching in the presence of outliers. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2091–2098.

Dai, H., Dai, B., and Song, L. (2016). Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning (ICML)*.

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems (NIPS)*.

Dai, H., Tian, Y., Dai, B., Skiena, S., and Song, L. (2018). Syntax-directed variational autoencoder for structured data. In *International Conference on Learning Representations (ICLR)*.

Date, K. and Nagi, R. (2016). GPU-accelerated Hungarian algorithms for the linear assignment problem. *Parallel Computing*, 57:52–72.

De Deuge, M., Quadros, A., Hung, C., and Douillard, B. (2013). Unsupervised feature learning for classification of outdoor 3D scans. In *Australasian Conference on Robotics and Automation*, volume 2.

Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NIPS)*.

Demantké, J., Mallet, C., David, N., and Vallet, B. (2011). Dimensionality based scale selection in 3D lidar point clouds. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVIII-5/W12:97–102.

Dhillon, I. S., Guan, Y., and Kulis, B. (2007). Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11).

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real NVP. *CoRR*, abs/1605.08803.

Dobson, P. D. and Doig, A. J. (2003). Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783.

Dörfler, F. and Bullo, F. (2013). Kron reduction of graphs with applications to electrical networks. *IEEE Trans. on Circuits and Systems*, 60-I(1):150–163.

Duvenaud, D. K., Maclaurin, D., Aguilera-Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems (NIPS)*.

Edwards, M. and Xie, X. (2016). Graph based convolutional neural network. In *British Machine Vision Conference (BMVC)*.

Engelmann, F., Kontogianni, T., Hermans, A., and Leibe, B. (2017). Exploring spatial context for 3d semantic segmentation of point clouds. In *IEEE International Conference on Computer Vision (ICCV), 3DRMS Workshop*.

Erdos, P. and Rényi, A. (1960). On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60.

Fan, H., Su, H., and Guibas, L. J. (2017). A point set generation network for 3D object reconstruction from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2463–2471.

Gadde, R., Jampani, V., Kiefel, M., Kappler, D., and Gehler, P. (2016). Superpixel convolutional networks using bilateral inceptions. In *IEEE European Conference on Computer Vision (ECCV)*.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, pages 1263–1272.

Gómez-Bombarelli, R., Duvenaud, D. K., Hernández-Lobato, J. M., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2016). Automatic chemical design using a data-driven continuous representation of molecules. *CoRR*, abs/1610.02415.

Gong, S. and Xiang, T. (2003). Recognition of group activities using dynamic probabilistic networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 742–749.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. (2014). Generative adversarial networks. *CoRR*, abs/1406.2661.

Graham, B. (2014). Fractional max-pooling. *CoRR*, abs/1412.6071.

Graham, B. (2015). Sparse 3d convolutional neural networks. In *British Machine Vision Conference (BMVC)*, pages 150.1–150.9.

Graham, B., Engelcke, M., and van der Maaten, L. (2018). 3d semantic segmentation with submanifold sparse convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864.

Guinard, S. and Landrieu, L. (2017). Weakly supervised segmentation-aided classification of urban scenes from 3D LiDAR point clouds. In *ISPRS 2017*.

Hackel, T., Savinov, N., Ladicky, L., Wegner, J. D., Schindler, K., and Pollefeys, M. (2017). Semantic3d. net: A new large-scale point cloud classification benchmark. *arXiv preprint arXiv:1704.03847*.

Hackel, T., Wegner, J. D., and Schindler, K. (2016). Fast semantic segmentation of 3D point clouds with strongly varying density. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 3(3).

Hamilton, W. L., Ying, R., and Leskovec, J. (2017a). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

Hamilton, W. L., Ying, Z., and Leskovec, J. (2017b). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1025–1035.

He, K. and Sun, J. (2015). Convolutional neural networks at constrained time cost. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257.

Hu, H., Munoz, D., Bagnell, J. A., and Hebert, M. (2013). Efficient 3-d scene analysis from streaming data. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Huang, G., Liu, Z., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Huang, J. and You, S. (2016). Point cloud labeling using 3D convolutional neural network. In *ICPR*.

Huang, Q., Wang, W., and Neumann, U. (2018). Recurrent slice networks for 3D segmentation on point clouds. *arXiv preprint arXiv:1802.04402*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*.

Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. (2012). ZINC: A free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768.

Isola, P., Zhu, J., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976.

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. (2015). Spatial transformer networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2017–2025.

Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *CoRR*, abs/1611.01144.

Jaromczyk, J. W. and Toussaint, G. T. (1992). Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517.

Jin, W., Barzilay, R., and Jaakkola, T. S. (2018). Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning (ICML)*, pages 2328–2337.

Johnson, D. D. (2017). Learning graphical state transitions. In *International Conference on Learning Representations (ICLR)*.

Kearnes, S. M., McCloskey, K., Berndl, M., Pande, V. S., and Riley, P. (2016). Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, 30(8):595–608.

Kim, B.-S., Kohli, P., and Savarese, S. (2013). 3D scene understanding by voxel-CRF. In *IEEE International Conference on Computer Vision (ICCV)*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Kingma, D. P., Salimans, T., and Welling, M. (2016). Improving variational inference with inverse autoregressive flow. *CoRR*, abs/1606.04934.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *CoRR*, abs/1312.6114.

Kipf, T. N., Fetaya, E., Wang, K., Welling, M., and Zemel, R. S. (2018). Neural relational inference for interacting systems. In *International Conference on Machine Learning (ICML)*, pages 2693–2702.

Kipf, T. N. and Welling, M. (2016a). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

Kipf, T. N. and Welling, M. (2016b). Variational graph auto-encoders. *Advances in Neural Information Processing Systems (NIPS), Workshop on Bayesian Deep Learning*.

Klokov, R. and Lempitsky, V. S. (2017). Escape from cells: Deep Kd-networks for the recognition of 3D point cloud models. *CoRR*, abs/1704.01222.

Kondor, R. (2018). N-body networks: a covariant hierarchical neural network architecture for learning atomic potentials. *CoRR*, abs/1803.01588.

Kondor, R., Son, H. T., Pan, H., Anderson, B., and Trivedi, S. (2018). Covariant compositional networks for learning graphs. *CoRR*, abs/1801.02144.

Koppula, H. S., Anand, A., Joachims, T., and Saxena, A. (2011). Semantic labeling of 3D point clouds for indoor scenes. In *Advances in Neural Information Processing Systems (NIPS)*, pages 244–252.

Ktena, S. I., Parisot, S., Ferrante, E., Rajchl, M., Lee, M., Glocker, B., and Rueckert, D. (2017). Distance metric learning using graph convolutional networks: Application to functional brain networks. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22 (1):79–86.

Kusner, M. J. and Hernández-Lobato, J. M. (2016). GANS for sequences of discrete elements with the gumbel-softmax distribution. *CoRR*, abs/1611.04051.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. (2017). Grammar variational autoencoder. In *International Conference on Machine Learning (ICML)*, pages 1945–1954.

Landrieu, L. and Obozinski, G. (2017). Cut pursuit: Fast algorithms to learn piecewise constant functions on general weighted graphs. *SIAM Journal on Imaging Sciences*, 10(4):1724–1766.

Landrieu, L., Raguet, H., Vallet, B., Mallet, C., and Weinmann, M. (2017). A structured regularization framework for spatially smoothing semantic labelings of 3D point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 132:102 – 118.

Landrieu, L. and Simonovsky, M. (2018). Large-scale point cloud semantic segmentation with superpoint graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Landrum, G. (2011). RDKit: Open-source cheminformatics.

Larsson, M., Kahl, F., Zheng, S., Arnab, A., Torr, P. H. S., and Hartley, R. I. (2017). Learning arbitrary potentials in CRFs with gradient descent. *CoRR*, abs/1701.06805.

Lawin, F. J., Danelljan, M., Tosteberg, P., Bhat, G., Khan, F. S., and Felsberg, M. (2017). Deep projective 3D semantic segmentation. *arXiv preprint arXiv:1705.03428*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114.

Lei, T., Jin, W., Barzilay, R., and Jaakkola, T. (2017). Deriving neural architectures from sequence and graph kernels. *arXiv preprint arXiv:1705.09037*.

Li, Y., Bu, R., Sun, M., and Chen, B. (2018a). PointCNN. *CoRR*, abs/1801.07791.

Li, Y., Pirk, S., Su, H., Qi, C. R., and Guibas, L. J. (2016a). FPNN: field probing neural networks for 3d data. In *Advances in Neural Information Processing Systems (NIPS)*, pages 307–315.

Li, Y., Swersky, K., and Zemel, R. S. (2015). Generative moment matching networks. In *International Conference on Machine Learning (ICML)*, pages 1718–1727.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. S. (2016b). Gated graph sequence neural networks. In *International Conference on Learning Representations (ICLR)*.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. (2018b). Learning deep generative models of graphs. In *International Conference on Machine Learning (ICML)*.

Li, Y., Yu, R., Shahabi, C., and Liu, Y. (2018c). Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations (ICLR)*.

Li, Y., Zhang, L., and Liu, Z. (2018d). Multi-objective de novo drug design with conditional graph generative model. *J. Cheminformatics*, 10(1):33:1–33:24.

Liang, X., Lin, L., Shen, X., Feng, J., Yan, S., and Xing, E. P. (2017). Interpretable structure-evolving LSTM. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2175–2184.

Liang, X., Shen, X., Feng, J., Lin, L., and Yan, S. (2016). Semantic object parsing with graph LSTM. In *IEEE European Conference on Computer Vision (ECCV)*, pages 125–143.

Lin, G., Shen, C., van den Hengel, A., and Reid, I. D. (2016). Efficient piecewise training of deep structured models for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. L. (2018). Constrained graph variational autoencoders for molecule design. *CoRR*, abs/1805.09076.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440.

Lu, Y. and Rasmussen, C. (2012). Simplified Markov random fields for efficient semantic labeling of 3d point clouds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2690–2697.

Makhzani, A., Shlens, J., Jaitly, N., and Goodfellow, I. J. (2015). Adversarial autoencoders. *CoRR*, abs/1511.05644.

Martinovic, A., Knopp, J., Riemenschneider, H., and Van Gool, L. (2015). 3D all the way: Semantic segmentation of urban scenes from start to end in 3D. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Masci, J., Boscaini, D., Bronstein, M. M., and Vandergheynst, P. (2015). Geodesic convolutional neural networks on Riemannian manifolds. In *IEEE International Conference on Computer Vision (ICCV), Workshop*, pages 37–45.

Masoumi, M. and Hamza, A. B. (2017). Shape classification using spectral graph wavelets. *Appl. Intell.*, 47(4):1256–1269.

Maturana, D. and Scherer, S. (2015). Voxnet: A 3D convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

McKay, B. D. and Piperno, A. (2014). Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60(0):94 – 112.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mishkin, D. and Matas, J. (2016). All you need is a good init. In *International Conference on Learning Representations (ICLR)*.

Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., and Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model CNNs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434.

Monti, F., Otness, K., and Bronstein, M. M. (2018). Motifnet: a motif-based graph convolutional network for directed graphs. *CoRR*, abs/1802.01572.

Mou, L., Li, G., Zhang, L., Wang, T., and Jin, Z. (2016). Convolutional neural networks over tree structures for programming language processing. In *AAAI Conference on Artificial Intelligence*, pages 1287–1293.

Munoz, D., Bagnell, J. A., Vandapel, N., and Hebert, M. (2009). Contextual classification with functional max-margin Markov networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Narayanan, A., Chandramohan, M., Chen, L., Liu, Y., and Saminathan, S. (2016). subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928*.

Niemeyer, J., Rottensteiner, F., and Soergel, U. (2014). Contextual classification of lidar data and building object detection in urban areas. *ISPRS Journal of Photogrammetry and Remote Sensing*, 87:152–165.

Niepert, M., Ahmed, M., and Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In *International Conference on Machine Learning (ICML)*.

Olivecrona, M., Blaschke, T., Engkvist, O., and Chen, H. (2017). Molecular de novo design through deep reinforcement learning. *CoRR*, abs/1704.07555.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. *Advances in Neural Information Processing Systems (NIPS), Autodiff Workshop*.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: online learning of social representations. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710.

Perraudin, N., Paratte, J., Shuman, D. I., Kalofolias, V., Vandergheynst, P., and Hammond, D. K. (2014). GSPBOX: A toolbox for signal processing on graphs. *CoRR*, abs/1408.5781.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). PointNet: Deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Qi, C. R., Su, H., Nießner, M., Dai, A., Yan, M., and Guibas, L. J. (2016). Volumetric and multi-view CNNs for object classification on 3D data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems (NIPS)*.

Qi, X., Liao, R., Jia, J., Fidler, S., and Urtasun, R. (2017c). 3D graph neural networks for RGBD semantic segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5209–5218.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Ramakrishnan, R., Dral, P. O., Rupp, M., and von Lilienfeld, O. A. (2014). Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1.

Reed, S. E., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. (2016). Generative adversarial text to image synthesis. In *International Conference on Machine Learning (ICML)*, pages 1060–1069.

Riegler, G., Ulusoy, A. O., Bischof, H., and Geiger, A. (2017a). OctNetFusion: Learning depth fusion from data. In *Proceedings of the International Conference on 3D Vision*.

Riegler, G., Ulusoy, A. O., and Geiger, A. (2017b). OctNet: Learning deep 3D representations at high resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point cloud library (pcl). In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–4. IEEE.

Samanta, B., De, A., Ganguly, N., and Gomez-Rodriguez, M. (2018). Designing random graph models using variational autoencoders with applications to chemical design. *CoRR*, abs/1802.05283.

Sardellitti, S., Barbarossa, S., and Lorenzo, P. D. (2017). On the graph fourier transform for directed graphs. *J. Sel. Topics Signal Processing*, 11(6):796–811.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2014). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations (ICLR)*.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80.

Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., and Langs, G. (2017). Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *Information Processing in Medical Imaging (IPMI)*, pages 146–157.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. v. d., Titov, I., and Welling, M. (2017). Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*.

Schölkopf, B. and Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

Schütt, K. T., Arbabzadah, F., Chmiela, S., Müller, K. R., and Tkatchenko, A. (2017). Quantum-chemical insights from deep tensor neural networks. In *Nature communications*.

Schütt, K. T., Kindermans, P., Felix, H. E. S., Chmiela, S., Tkatchenko, A., and Müller, K. (2017). Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 992–1002.

Schwing, A. G. and Urtasun, R. (2015). Fully connected deep structured networks. *CoRR*, abs/1503.02351.

Segler, M. H. S., Kogej, T., Tyrchan, C., and Waller, M. P. (2018). Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS Central Science*, 4(1):120–131.

Shapovalov, R., Vetrov, D., and Kohli, P. (2013). Spatial inference machines. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Shen, D., Wu, G., and Suk, H.-I. (2017). Deep learning in medical image analysis. *Annual review of biomedical engineering*, 19:221–248.

Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561.

Shuman, D. I., Faraji, M. J., and Vandergheynst, P. (2016). A multiscale pyramid transform for graph signals. *IEEE Trans. Signal Processing*, 64(8):2119–2134.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98.

Shuman, D. I., Vandergheynst, P., and Frossard, P. (2011). Chebyshev polynomial approximation for distributed signal processing. In *Distributed Computing in Sensor Systems (DCOSS)*, pages 1–8.

Simonovsky, M., Gutiérrez-Becker, B., Mateus, D., Navab, N., and Komodakis, N. (2016). A deep metric for multimodal registration. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 10–18.

Simonovsky, M. and Komodakis, N. (2016). OnionNet: Sharing features in cascaded deep classifiers. In *British Machine Vision Conference (BMVC)*.

Simonovsky, M. and Komodakis, N. (2017). Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Simonovsky, M. and Komodakis, N. (2018a). GraphVAE: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks (ICANN)*.

Simonovsky, M. and Komodakis, N. (2018b). Towards variational generation of small graphs. In *International Conference on Learning Representations (ICLR), Workshop track*.

Sinkhorn, R. and Knopp, P. (1967). Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348.

Snijders, T. A. and Nowicki, K. (1997). Estimation and prediction for stochastic blockmodels for graphs with latent block structure. *Journal of Classification*, 14(1):75–100.

Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3483–3491.

Spielman, D. A. and Srivastava, N. (2011). Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926.

Stewart, R., Andriluka, M., and Ng, A. Y. (2016). End-to-end people detection in crowded scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2325–2333.

Su, H., Maji, S., Kalogerakis, E., and Learned-Miller, E. G. (2015). Multi-view convolutional neural networks for 3D shape recognition. In *IEEE International Conference on Computer Vision (ICCV)*.

Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *International Conference on Machine Learning (ICML)*, pages 1017–1024.

Tallec, C. and Ollivier, Y. (2018). Can recurrent neural networks warp time? In *International Conference on Learning Representations (ICLR)*.

Tatarchenko, M., Dosovitskiy, A., and Brox, T. (2017). Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *IEEE International Conference on Computer Vision (ICCV)*.

Tatarchenko, M., Park, J., Koltun, V., and Zhou., Q.-Y. (2018). Tangent convolutions for dense prediction in 3D. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Tchapmi, L. P., Choy, C. B., Armeni, I., Gwak, J., and Savarese, S. (2017). SEGCloud: Semantic segmentation of 3D point clouds. *arXiv preprint arXiv:1710.07563*.

Theis, L., van den Oord, A., and Bethge, M. (2015). A note on the evaluation of generative models. *CoRR*, abs/1511.01844.

Thomas, N., Smidt, T., Kearnes, S. M., Yang, L., Li, L., Kohlhoff, K., and Riley, P. (2018). Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *CoRR*, abs/1802.08219.

Tolstikhin, I. O., Bousquet, O., Gelly, S., and Schölkopf, B. (2018). Wasserstein auto-encoders. In *International Conference on Learning Representations (ICLR)*.

Tu, W.-C., Liu, M.-Y., Jampani, V., Sun, D., Chien, S.-Y., Yang, M.-H., and Kautz, J. (2018). Learning superpixels with segmentation-aware affinity loss. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2017). Deep image prior. *arXiv:1711.10925*.

van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. In *International Conference on Machine Learning (ICML)*, pages 1747–1756.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph Attention Networks. *International Conference on Learning Representations (ICLR)*.

Verelst, T., Berman, M., and Blaschko, M. B. (2018). Generating superpixels with deep representations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Deep Vision Workshop*.

Verma, N., Boyer, E., and Verbeek, J. (2018). FeaStNet: Feature-steered graph convolutions for 3D shape analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Vinyals, O., Bengio, S., and Kudlur, M. (2015). Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*.

Wale, N., Watson, I. A., and Karypis, G. (2008). Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375.

Wang, P.-S., Liu, Y., Guo, Y.-X., Sun, C.-Y., and Tong, X. (2017). O-CNN: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (SIGGRAPH)*, 36(4).

Wang, X., Girshick, R., Gupta, A., and He, K. (2018a). Non-local neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2018b). Dynamic graph CNN for learning on point clouds. *arXiv preprint arXiv:1801.07829*.

Weininger, D. (1988). SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36.

Weinmann, M., Hinz, S., and Weinmann, M. (2017). A hybrid semantic point cloud classification-segmentation framework based on geometric features and semantic rules. *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 85(3):183–194.

Weinmann, M., Schmidt, A., Mallet, C., Hinz, S., Rottensteiner, F., and Jutzi, B. (2015). Contextual classification of point cloud data by exploiting individual 3D neighborhoods. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3/W4:271–278.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Wolf, D., Prankl, J., and Vincze, M. (2015). Fast semantic segmentation of 3D point clouds using a dense CRF with learned parameters. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Wolf, S., Schott, L., Köthe, U., and Hamprecht, F. A. (2017). Learned watershed: End-to-end learning of seeded segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2030–2038.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. (2018). Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530.

Wu, Z., Song, S., Khosla, A., Tang, X., and Xiao, J. (2015a). 3D ShapeNets for 2.5D object recognition and next-best-view prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015b). 3D ShapeNets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920.

Xu, D., Zhu, Y., Choy, C. B., and Fei-Fei, L. (2017). Scene graph generation by iterative message passing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3097–3106.

Yanardag, P. and Vishwanathan, S. V. N. (2015). Deep graph kernels. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Yi, L., Su, H., Guo, X., and Guibas, L. J. (2017). SyncSpecCNN: Synchronized spectral CNN for 3D shape segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6584–6592.

You, J., Liu, B., Ying, R., Pande, V. S., and Leskovec, J. (2018a). Graph convolutional policy network for goal-directed molecular graph generation. *CoRR*, abs/1806.02473.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. (2018b). GraphRNN: A deep generative model for graphs. In *International Conference on Machine Learning (ICML)*.

Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017). SeqGAN: Sequence generative adversarial nets with policy gradient. In *AAAI Conference on Artificial Intelligence*.

Yuan, Y., Liang, X., Wang, X., Yeung, D., and Gupta, A. (2017). Temporal dynamic graph LSTM for action-driven video object detection. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1819–1828.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence*.

Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., and Torr, P. H. S. (2015). Conditional random fields as recurrent neural networks. In *IEEE International Conference on Computer Vision (ICCV)*.