Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs

Martin Simonovsky, Nikos Komodakis

Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs

Martin Simonovsky
Université Paris Est, École des Ponts ParisTech
martin.simonovsky@enpc.fr

Nikos Komodakis
Université Paris Est, École des Ponts ParisTech
nikos.komodakis@enpc.fr

Abstract

A number of problems can be formulated as prediction on graph-structured data. In this work, we generalize the convolution operator from regular grids to arbitrary graphs while avoiding the spectral domain, which allows us to handle graphs of varying size and connectivity. To move beyond a simple diffusion, filter weights are conditioned on the specific edge labels in the neighborhood of a vertex. Together with the proper choice of graph coarsening, we explore constructing deep neural networks for graph classification. In particular, we demonstrate the generality of our formulation in point cloud classification, where we set the new state of the art, and on a graph classification dataset, where we outperform other deep learning approaches. The source code is available at https://github.com/mys007/ecc.

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) [22].

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 current formulations of graph convolution do not exploit edge labels, 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 so far, their voxelization being the only trend to the best of our knowledge [26, 19]. 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 [9].

Our contributions are as follows:

- We formulate a convolution-like operation on graph signals performed in the spatial domain where filter weights are conditioned on edge labels (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 classification. 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 [39], outperforming other approaches based on deep learning there.

2. Related Work

The first formulation of a convolutional network analogy for irregular domains modeled with graphs has been introduced by Bruna *et al.* [6], who looked into both the spatial and the spectral domain of representation for performing localized filtering.

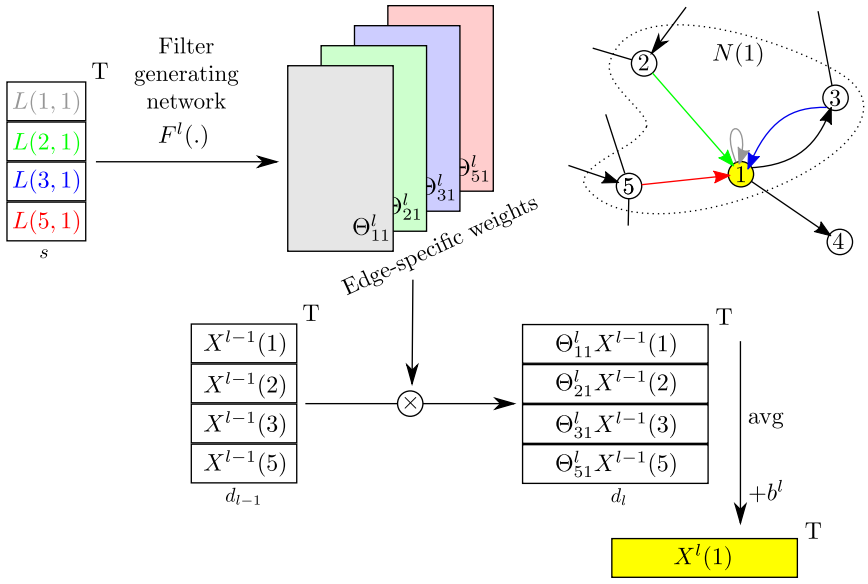

Figure 1. Illustration of edge-conditioned convolution on a directed subgraph. The feature $X^l(1)$ on vertex 1 in the l-th network layer is computed as a weighted sum of features $X^{l-1}(.)$ on the set of its predecessor vertices, assuming self-loops. The particular weight matrices are dynamically generated by filter-generating network F^l based on the corresponding edge labels $L(.)$, visualized as colors.

Spectral Methods. A mathematically sound definition of convolution operator makes use of the spectral analysis theory, where it corresponds to multiplication of the signal on vertices transformed into the spectral domain by graph Fourier transform. The spatial locality of filters is then given by smoothness of the spectral filters, in case of [6] modeled as B-splines. The transform involves very expensive multiplications with the eigenvector matrix. However, by a parameterization of filters as Chebyshev polynomials of eigenvalues and their approximate evaluation, computationally efficient and localized filtering has been recently achieved by Defferrard *et al.* [11]. Nevertheless, the filters are still learned in the context of 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 vertices may differ. This precludes applications on problems where the graph structure varies in the dataset, such as meshes, point clouds, or diverse biochemical datasets.

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 [6], as the number of vertices adjacent to a particular vertex varies and their ordering is often not well definable.

Spatial Methods. Bruna *et al.* [6] assumed fixed graph structure and did not share any weights among neighborhoods. Several works have independently dealt with this problem. Duvenaud *et al.* [14] sum the signal over neighboring vertices followed by a weight matrix multiplication, effectively sharing the same weights among all edges. Atwood and Towsley [2] share weights based on the number of hops between two vertices. Kipf and Welling [21] further approximate the spectral method of [11] 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 the standard convolution on grids. In contrast, our method can make use of possible edge labels and is shown to generalize regular convolution (Section 3.2).

The approach of Niepert *et al.* [28] 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. Finally, Graph neural networks [34, 24] propagate features across a graph until (near) convergence and exploit edge labels as one of the sources of information as we do. However, their system is quite different from the current multilayer feed-forward architectures, making the reuse of today's common building blocks not straightforward.

CNNs on Point Clouds and Meshes. There has been little work on deep learning on point clouds or meshes. Masci *et al.* [25] define convolution over patch descriptors around every vertex of a 3D mesh using geodesic distances, formulated in a deep learning architecture. 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 [26] or segmentation [19] purposes. Instead, we regard point cloud as graphs in Euclidean space in this work.

3. Method

We propose a method for performing convolutions over local graph neighborhoods exploiting edge labels (Section 3.1) and show it to generalize regular convolutions (Section 3.2). Afterwards, we present deep networks with our convolution operator (Section 3.3) in the case of point clouds (Section 3.4) and general graphs (Section 3.5).

3.1. Edge-Conditioned Convolution

Let us consider a directed or undirected graph $G = (V, E)$, where V is a finite set of vertices with $|V| = n$ and $E \subseteq V \times V$ is a set of edges with $|E| = m$. Let $l \in \{0, .., l_{\max}\}$ be the layer index in a feed-forward neural network. We assume the graph is both vertex- and edge-labeled, *i.e.* there exists function $X^l : V \mapsto \mathbb{R}^{d_l}$ assigning labels (also called signals or features) to each vertex and $L : E \mapsto \mathbb{R}^s$ assigning labels (also called attributes) to each edge. These functions can be regarded as matrices $X^l \in \mathbb{R}^{n \times d_l}$ and $L \in \mathbb{R}^{m \times s}$, X^0 then being the input signal. A neighborhood $N(i) = \{j; (j, i) \in E\} \cup \{i\}$ of vertex i is defined to contain all adjacent vertices (predecessors in directed graphs) including i itself (self-loop).

Our approach computes the filtered signal $X^l(i) \in \mathbb{R}^{d_l}$ at vertex i as a weighted sum of signals $X^{l-1}(j) \in \mathbb{R}^{d_{l-1}}$ in its neighborhood, $j \in N(i)$. While such a commutative aggregation solves the problem of undefined vertex 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 label. To this end, we borrow the idea from Dynamic filter networks [5] and define a filter-generating network $F^l : \mathbb{R}^s \mapsto \mathbb{R}^{d_l \times d_{l-1}}$ which given edge label $L(j, i)$ outputs edge-specific weight matrix $\Theta^l_{ji} \in \mathbb{R}^{d_l \times d_{l-1}}$, see Figure 1.

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

$$\begin{aligned} X^l(i) &= \frac{1}{|N(i)|} \sum_{j \in N(i)} F^l(L(j, i); w^l) X^{l-1}(j) + b^l \\ &= \frac{1}{|N(i)|} \sum_{j \in N(i)} \Theta^l_{ji} X^{l-1}(j) + b^l \end{aligned} \tag{1}$$

where $b^l \in \mathbb{R}^{d_l}$ is a learnable bias and F^l is parameterized by learnable network weights w^l. For clarity, w^l and b^l are model parameters updated only during training and Θ^l_{ji} are dynamically generated parameters for an edge label in a particular input graph. The filter-generating network F^l can be implemented with any differentiable architecture; we use multi-layer perceptrons in our applications.

Complexity. Computing X^l for all vertices requires at most[1] m evaluations of F^l and $m + n$ or $2m + n$ matrix-vector multiplications for directed, resp. undirected graphs. Both operations can be carried out efficiently on the GPU in batch-mode.

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.

The weights in ECC are tied by edge label, which is in contrast to tying them by hop distance from a vertex [2], according to a neighborhood linearization heuristic [28], by being the central vertex or not [21], indiscriminately [14], or not at all [6].

In fact, our definition reduces to that of Duvenaud *et al.* [14] (up to scaling) in the case of uninformative edge labels: $\sum_{j \in N(i)} \Theta^l_{ji} X^{l-1}(j) = \Theta^l \sum_{j \in N(i)} X^{l-1}(j)$ if $\Theta^l_{ji} = \Theta^l \ \ \forall (j, i) \in E$.

More 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 vertices V forming a path graph (chain). To obtain convolution with a centered kernel of size s, we form E so that each vertex is connected to its s spatially nearest neighbors including self by a directed edge labeled with one-hot encoding of the neighbor's discrete offset δ, see Figure 2. Taking F^l as a single-layer perceptron without bias, we have $F^l(L(j, i); w^l) = w^l(\delta)$, where $w^l(\delta)$ denotes the respective reshaped column of the parameter matrix $w^l \in \mathbb{R}^{(d_l \times d_{l-1}) \times s}$. With a slight abuse of notation, we arrive at the equivalence to the standard convolution: $X^l(i) = \sum_{j \in N(i)} \Theta^l_{ji} X^{l-1}(j) = \sum_\delta w^l(\delta) X^{l-1}(i - \delta)$, ignoring the normalization factor of $1/|N(i)|$ playing a role only at grid boundaries.

This shows that ECC can retain the same number of parameteres and computational complexity of the regular convolution in the case of grids. Note that such equivalence is not possible with none of [2, 21, 14] due to their way of weight tying.

3.3. Deep Networks with ECC

While ECC is in principle applicable to both vertex classification and graph classification tasks, in this paper 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 feed-forward networks of in-

[1] If edge labels are represented by s discrete values in a particular graph and $s < m$, F^l can be evaluated only s-times.

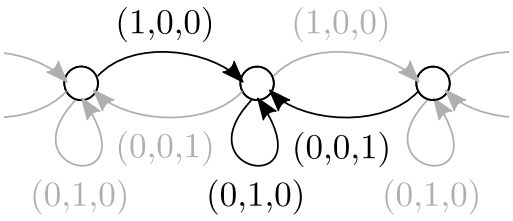

Figure 2. Construction of a directed graph with one-hot edge labeling where the proposed edge-conditioned convolution is equivalent to the regular 1D convolution with a centered filter of size $s = 3$.

terlaced convolutions and poolings topped by global pooling and fully-connected layers, see Figure 3 for an illustration. This way, information from the local neighborhoods gets combined over successive layers to gain context (enlarge receptive field). While edge labels are fixed for a particular graph, their (learned) interpretation by the means of filter generating networks may change from layer to layer (weights of F^l are not shared among layers). Therefore, the restriction of ECC to 1-hop neighborhoods $N(i)$ is not a constraint, akin to using small 3×3 filters in normal CNNs in exchange for deeper networks, which is known to be beneficial [17].

We use batch normalization [20] 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 [27] or layer normalization [3].

Pooling. While (non-strided) convolutional layers and all point-wise layers do not change the underlying graph and only evolve the signal on vertices, pooling layers are defined to output aggregated signal on the vertices of a new, coarsened graph. Therefore, a pyramid of $h_{\max}$ progressively coarser graphs has to be constructed for each input graph. Let us extend here our notation with an additional superscript $h \in \{0, .., h_{\max}\}$ to distinguish among different graphs $G^{(h)} = (V^{(h)}, E^{(h)})$ in the pyramid when necessary. Each $G^{(h)}$ has also its associated labels $L^{(h)}$ and signal $X^{(h),l}$. A coarsening typically consists of three steps: subsampling or merging vertices, creating the new edge structure $E^{(h)}$ and labeling $L^{(h)}$ (so-called reduction), and mapping the vertices in the original graph to those in the coarsened one with $M^{(h)} : V^{(h-1)} \mapsto V^{(h)}$. 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 pooling layer with index l_h aggregates $X^{(h-1),l_h-1}$ into a lower dimensional $X^{(h),l_h}$ based on $M^{(h)}$. See Figure 3 for an example of using the introduced notation.

During coarsening, a small graph may be reduced to several disconnected vertices in its lower resolutions without problems as self-edges are always present. Since the architecture is designed to process graphs with variable n, m, we deal with varying vertex count $n^{(h_{max})}$ in the lowest graph resolution by global average/max pooling.

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. Due to their natural irregularity and sparsity, so far 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 [26] or segmentation [19] 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 (we are not aware of any GPU implementation of convolutions on sparse tensors). Second, the necessity to fit them into a fixed size 3D grid brings about discretization artifacts and the loss of metric scale and possibly of details. With this work, we would like to offer a competitive alternative to the mainstream by performing deep learning on point clouds directly. As far as we know, we are the first to demonstrate such a result.

Graph Construction. Given a point cloud P with its point features X_P (such as laser return intensity or color) we build a directed graph $G = (V, E)$ and set up its labels X^0 and L as follows. First, we create vertex $i \in V$ for every point $p \in P$ and assign the respective signal to it by $X^0(i) = X_P(p)$ (or 0 if there are no features $X_P(p)$). Then we connect each vertex i to all vertices 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. The offset $\delta = p_j - p_i$ between the points corresponding to vertices j, i is represented in Cartesian and spherical coordinates as 6D edge label vector $L(j, i) = (\delta_x, \delta_y, \delta_z, ||\delta||, \arccos \delta_z/||\delta||, \arctan \delta_y/\delta_x)$.

Graph Coarsening. For a single input point cloud P, a pyramid of downsampled point clouds $P^{(h)}$ is obtained by the VoxelGrid algorithm [31], which overlays a grid of resolution $r^{(h)}$ 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 $P^{(h)}$ is then independently converted into a graph $G^{(h)}$ and labeling $L^{(h)}$ with neighborhood radius $\rho^{(h)}$ as described above. The pooling map $M^{(h)}$ is defined so that each point in $P^{(h-1)}$ is assigned to its spatially nearest point in the subsampled point cloud $P^{(h)}$.

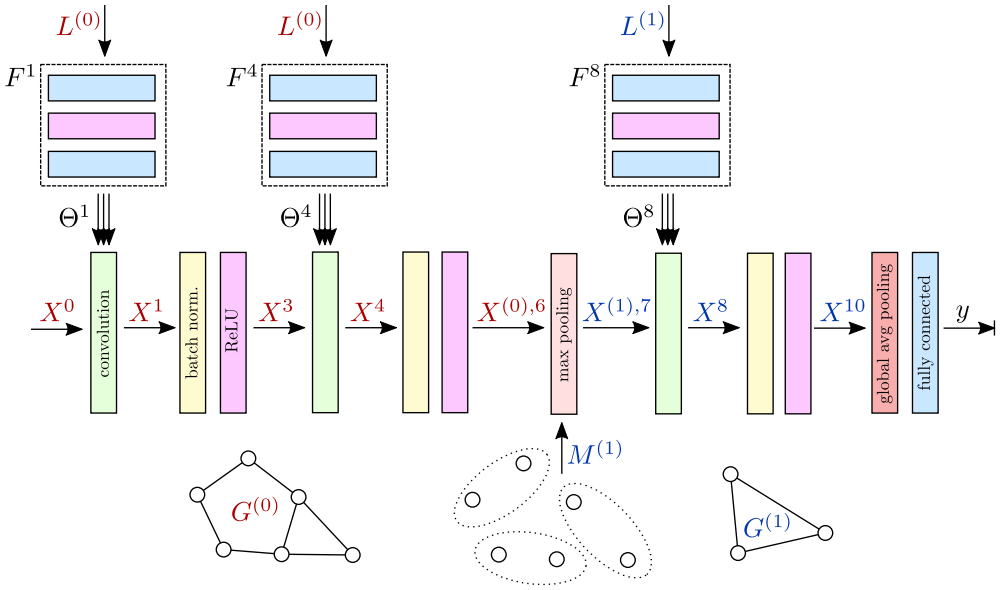

Figure 3. Illustration of a deep network with three edge-conditioned convolutions (first, fourth, and eight network layer) and one pooling (seventh layer). The last convolution is executed on a structurally different graph $G^{(1)}$, which is related to the input graph $G^{(0)}$ by coarsening and signal aggregation in the max pooling step according to mapping $M^{(1)}$. See Section 3.3 for more details.

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.

3.5. 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. This is by no means trivial and there exists a large body of literature on this problem [32]. Without any concept of spatial localization of vertices, we resort to established graph coarsening algorithms and utilize the multiresolution framework of Shuman *et al.* [36, 29], 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 [13], which also defines the new edge labeling, enhanced with spectral sparsification of edges [37]. 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 vertices, 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 [16]. We do not perform any other data augmentation.

4. Experiments

The proposed method is evaluated in point cloud classification (real-world data in Section 4.1 and synthetic in 4.2) and on a standard graph classification benchmark (Section 4.3). In addition, we validate our method and study its properties on MNIST (Section 4.4).

4.1. Sydney Urban Objects

This point cloud dataset [9] consists of 588 objects in 14 categories (vehicles, pedestrians, signs, and trees) manually extracted from 360° LiDAR scans, see Figure 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

Model	Mean F1
Triangle+SVM [9]	67.1
GFH+SVM [7]	71.0
VoxNet [26]	73.0
ORION [1]	77.8
ECC 2ρ	74.4
ECC 1.5ρ	76.9
ECC	78.4

Table 1. Mean F1 score weighted by class frequency on Sydney Urban Objects dataset [9]. Only the best-performing models of each baseline are listed.

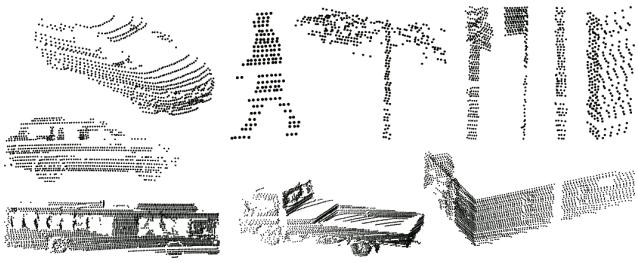

Figure 4. Illustrative samples of the majority of classes in Sydney Urban Objects dataset, reproduced from [9].

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 F^l have configuration FC(16)-FC(32)-FC($d_l d_{l-1}$) with orthogonal weight initialization [33] 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. Vertex signal X^0 is scalar laser return intensity (0-255), representing depth.

Results. Table 1 compares our result (ECC, 78.4) against two methods based on volumetric CNNs evaluated on voxelized occupancy grids of size 32x32x32 (VoxNet [26] 73.0 and ORION [1] 77.8), which we outperform by a small margin and set the new state of the art result on this dataset.

In the same table, we also 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, which would correspond to a preference of using smaller filters in regular CNNs. The average neighborhood size is roughly 10 vertices for our best-performing network. We hypothesize that larger radii smooth out the information in the central vertex. To investigate this, we increased the importance of the self-loop by adding an identity skip-connection (see Appendix E) and retrained the networks. We achieved 77.0, 79.5 (the new state of the art), and 77.4 mean F1 for ECC, ECC 1.5ρ, and ECC 2ρ, respectively. Stronger identity connection allowed for successful integration of a larger context, up to some limit, which indeed suggests that information should be aggregated neither too much nor too little.

4.2. ModelNet

ModelNet [40] 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 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 F^l are as in Section 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 vertex signal, *i.e.* X^0 are zero. For ModelNet40, the 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 2 compares our result to several recent works, based either on volumetric [40, 26, 1, 30] or rendered image representation [38]. 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 fully reaching the state of the art, we believe our method remains very competitive (90.8%, resp. 87.4% mean instance accuracy). For a fairer comparison, a leading volumetric method should be retrained on voxelized synthetic point clouds.

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, indicating the variability in dataset sizes, in graph sizes, and in the availability of labels. Following [35], we perform 10-fold cross-validation

Model	ModelNet10	ModelNet40
3DShapeNets [40]	83.5	77.3
MVCNN [38]	—	90.1
VoxNet [26]	92	83
ORION [1]	93.8	—
SubvolumeSup [30]	—	86.0 (89.2)
ECC	89.3 (90.0)	82.4 (87.0)
ECC (12 votes)	90.0 (90.8)	83.2 (87.4)

Table 2. Mean class accuracy (resp. mean instance accuracy) on ModelNets [40]. Only the best models of each baseline are listed.

with 9 folds for training and 1 for testing and report the average prediction accuracy.

NCI1 and NCI109 [39] consist of graph representations of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, respectively. MUTAG [10] is a dataset of nitro compounds labeled according to whether or not they have a mutagenic effect on a bacterium. ENZYMES [4] contains representations of tertiary structure of 6 classes of enzymes. D&D [12] is a database of protein structures (vertices are amino acids, edges indicate spatial closeness) classified as enzymes and non-enzymes.

Network Configuration. Our ECC-network for NCI1 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 F^l have configuration FC(64)-FC($d_l d_{l-1}$) with orthogonal weight initialization [33] and ReLU in between. Labels are encoded as one-hot vectors ($d_0 = 37$ and $s = 4$ due to an extra label 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 3.5). Small deviations from this description for the other four datasets are mentioned in the supplementary.

Baselines. We compare our method (ECC) to the state of the art Weisfeiler-Lehman graph kernel *et al.* [35] and to four approaches using deep learning as at least one of their components [2, 28, 41, 8]. 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 labels, we run our method with uniform labels and F^l being a single layer FC($d_l d_{l-1}$) without bias[2] (ECC no edge labels).

	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
# vertex labels	37	38	7	3	82
# edge labels	3	3	11	—	—

Table 3. Characteristics of the graph benchmark datasets, extended from [8]. Both edge and vertex labels are categorical.

Model	NCI1	NCI109	MUTAG	ENZYMES	D&D
DCNN [2]	62.61	62.86	66.98	18.10	—
PSCN [28]	78.59	—	92.63	—	77.12
Deep WL [41]	80.31	80.32	87.44	53.43	—
structure2vec [8]	83.72	82.16	88.28	61.10	82.22
WL [35]	84.55	84.49	83.78	59.05	79.78
ECC (no edge labels)	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 4. Mean accuracy (10 folds) on graph classification datasets. Only the best-performing models of each baseline are listed.

Results. Table 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 labels for convolution-based methods, as the performance of DCNN [2] and ECC without edge labels is distinctly worse, justifying the motivation behind this paper. Averaging over random sparsifications at test time improves accuracy by a small amount. Our results on datasets without edge labels (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. A more detailed discussion for each dataset is available in the supplementary.

4.4. MNIST

To further validate our method, we applied it to the MNIST classification problem [23], 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

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

P with points $p_i = (x, y, 0)$ and signal $X^0(i) = I(x, y)$ representing each pixel, $x, y \in \{0, .., 27\}$. Edge labeling and graph coarsening is performed as explained in Section 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 notation and filter-generating network being as in Section 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 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.* [11] and better than what is offered by other spectral-based approaches (98.2 [6], 94.96 [15]). 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 ($X^0(i) = 0$) 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 F^l. We compare with ECC configured to mimic regular convolution using single-layer filter networks and one-hot encoding of offsets (ECC one-hot), as described in Section 3.2. This configuration reaches 99.37 accuracy, or 0.23 more than ECC, implying that F^l are not perfect but still perform very well in learning the proper partitioning of edge labels.

Last, we explore the generated filters visually for the case of the sparse input ECC. As filters $\Theta^1 \in \mathbb{R}^{16\times1}$ are a continuous function of an edge label, we can visualize the change of values in each dimension in 16 images by sampling labels over grids of two resolutions. The coarser one in Figure 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 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	Train accuracy	Test accuracy
ECC	99.12	99.14
ECC (sparse input)	99.36	99.14
ECC (one-hot)	99.53	99.37

Table 5. Accuracy on MNIST dataset [23].

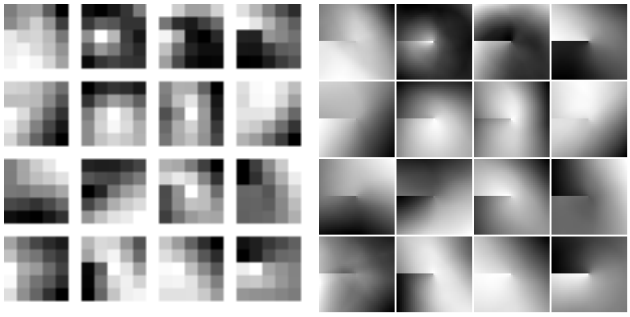

Figure 5. Convolutional filters learned on MNIST in the first layer for sparse input ECC, sampled in two different resolutions. See Section 4.4 for details.

5. Conclusion

We have introduced edge-conditioned convolution (ECC), an operation on graph signal performed in the spatial domain where filter weights are conditioned on edge labels and dynamically generated for each specific input sample. We have shown that our formulation generalizes the standard convolution on graphs if edge labels 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 is available at https://github.com/mys007/ecc.

In feature work we would like to treat meshes as graphs rather than point clouds. Moreover, we plan to address the currently higher level of GPU memory consumption in case of large graphs with continuous edge labels, for example by randomized clustering, which could also serve as additional regularization through data augmentation.

Acknowledgments.

We gratefully acknowledge NVIDIA Corporation for the donated GPU used in this research. We are thankful to anonymous reviewers for their feedback.

References

[1] N. S. Alvar, M. Zolfaghari, and T. Brox. Orientation-boosted voxel nets for 3d object recognition. *CoRR*, abs/1604.03351, 2016. 6, 7

[2] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In *NIPS*, 2016. 2, 3, 7, 10, 11

[3] L. J. Ba, R. Kiros, and G. E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. 4

[4] K. M. Borgwardt and H. Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*, pages 74–81, 2005. 7

[5] B. D. Brabandere, X. Jia, T. Tuytelaars, and L. V. Gool. Dynamic filter networks. In *NIPS*, 2016. 3, 12

[6] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2013. 1, 2, 3, 8

[7] T. Chen, B. Dai, D. Liu, and J. Song. Performance of global descriptors for velodyne-based urban object recognition. In *2014 IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, MI, USA, June 8-11, 2014*, pages 667–673, 2014. 6

[8] H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, 2016. 7, 10

[9] M. De Deuge, A. Quadros, C. Hung, and B. Douillard. Unsupervised feature learning for classification of outdoor 3d scans. In *Australasian Conference on Robitics and Automation*, volume 2, 2013. 1, 5, 6

[10] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. 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, 1991. 7

[11] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016. 2, 8

[12] P. D. Dobson and A. J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003. 7

[13] F. Dörfler and F. Bullo. Kron reduction of graphs with applications to electrical networks. *IEEE Trans. on Circuits and Systems*, 60-I(1):150–163, 2013. 5

[14] D. K. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, 2015. 2, 3

[15] M. Edwards and X. Xie. Graph based convolutional neural network. In *BMVC*, 2016. 8

[16] B. Graham. Fractional max-pooling. *CoRR*, abs/1412.6071, 2014. 5

[17] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *CVPR*, 2015. 4

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 11

[19] J. Huang and S. You. Point cloud labeling using 3d convolutional neural network. In *ICPR*, 2016. 1, 3, 4

[20] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 4

[21] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. 2, 3, 11

[22] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. 1

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 7, 8

[24] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel. Gated graph sequence neural networks. In *ICLR*, 2016. 2

[25] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. pages 37–45, 2015. 2

[26] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS*, 2015. 1, 3, 4, 6, 7

[27] D. Mishkin and J. Matas. All you need is a good init. In *ICLR*, 2016. 4

[28] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, 2016. 2, 3, 7, 10

[29] N. Perraudin, J. Paratte, D. I. Shuman, V. Kalofolias, P. Vandergheynst, and D. K. Hammond. GSPBOX: A toolbox for signal processing on graphs. *CoRR*, abs/1408.5781, 2014. 5

[30] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *CVPR*, 2016. 6, 7

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

[32] I. Safro, P. Sanders, and C. Schulz. Advanced coarsening schemes for graph partitioning. *ACM Journal of Experimental Algorithmics*, 19(1), 2014. 5

[33] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*, 2014. 6, 7, 12

[34] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009. 2

[35] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011. 6, 7, 10

[36] D. I. Shuman, M. J. Faraji, and P. Vandergheynst. A multiscale pyramid transform for graph signals. *IEEE Trans. Signal Processing*, 64(8):2119–2134, 2016. 5

[37] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. 5

[38] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*, 2015. 6, 7

[39] N. Wale, I. A. Watson, and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008. 1, 7

[40] Z. Wu, S. Song, A. Khosla, X. Tang, and J. Xiao. 3d shapenets for 2.5d object recognition and next-best-view prediction. In *CVPR*, 2015. 6, 7

[41] P. Yanardag and S. V. N. Vishwanathan. Deep graph kernels. In *SIGKDD*, 2015. 7, 10

Appendix

A. Overview

In the first part, the appendix provides further discussion of the graph classification results (Section B) and investigates robustness of point cloud classification to noise (Section C). In the second part, we explore several extensions of our ECC formulation, specifically with different edge labeling for point clouds (Section D), with identity connections (Section E), with degree labels (Section F), and with a learned normalization factor (Section G).

B. Details on Graph Classification Benchmark

In this section we describe the differences in our network architecture to the one introduced for NCI1 in the main paper and discuss evaluation results for each dataset in detail.

NCI1. ECC (83.80%) performs distinctly better than convolution methods that are not able to use edge labels (DCNN [2] 62.61%, PSCN [28] 78.59%). Methods not approaching the problem as convolutions on graphs but rather combining deep learning with other techniques are stronger (Deep WL [41] 80.31%, structure2vec [8] 83.72%) but are still outperformed by ECC. While the Weisfeiler-Lehman graph kernel remains the strongest method (WL [35] 84.55%), it is fair to conclude that ECC, structure2vec, and WL perform at the same level.

NCI109. We use the same ECC-network configuration and training details as described in Section 4.3 for NCI1, since both datasets are similar. ECC (82.14%) performs distinctly better than DCNN [2] (62.86%), which is not able to use edge labels, and is on par with non-convolutional approaches (Deep WL [41] 80.32%, structure2vec [8] 82.16%, WL [35] 84.49%).

MUTAG. As MUTAG is a tiny dataset of small graphs, we trained a downsized ECC-network to combat overfitting. Using the notation from Section 4.3, its configuration is C(16)-C(32)-C(48)-MP-C(64)-MP-GAP-FC(64)-D(0.2)-FC(2), all other details are as with NCI1. While by numbers ECC (89.44%) outperforms all other approaches except of PSCN [28] (92.63%), we note that all four leading methods (Deep WL [41] 87.44%, structure2vec [8] 88.28%, ECC, PSCN) can be seen to perform equally 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. Due to higher complexity of this task 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) using the notation and other details in Section 4.3. 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 [41] (53.43%) and is overperformed by WL [35] (59.05%) and structure2vec [8] (61.10%). Note that the gap to the other convolution-based method DCNN [2] (18.10%) is huge and there is an improvement of more than 4 percentage points due to edge labels in coarser graph resolutions from Kron reduction.

D&D. Due to large graphs in this 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) using the notation and other details in Section 4.3. As this dataset is 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 [28] 77.12%, WL [35] 79.78%, structure2vec [8] 82.22%), though the margin is not very large.

C. 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 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.

D. Edge Labels for Point Clouds

In Section 3.4 we defined edge labels $L(j,i)$ as the offset $\delta = p_j - p_i$ in Cartesian and spherical coordinates, $L(j,i) = (\delta_x, \delta_y, \delta_z, ||\delta||, \arccos \delta_z/||\delta||, \arctan \delta_y/\delta_x)$. Here, we explore the importance of individual elements in the proposed edge labeling and further evaluate labels invariant to rotation about objects' vertical axis z (IRz). Table 6 conveys that models with isotropic (60.7) or no labels (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,

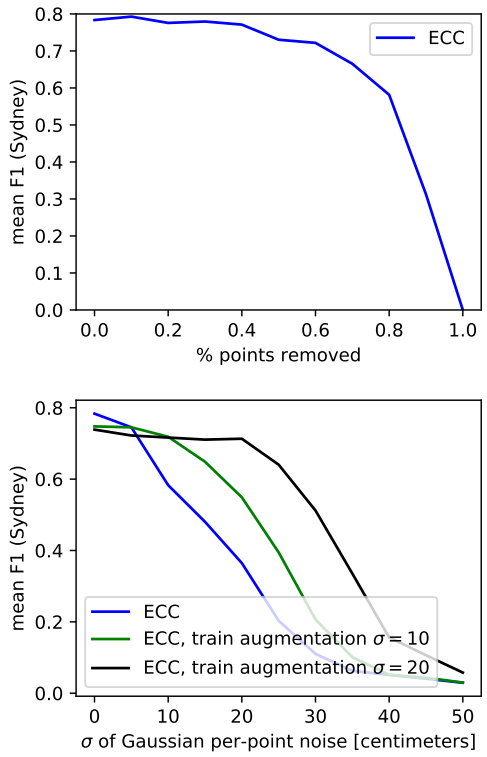

Figure 6. Robustness to point removal and Gaussian noise.

Label $L(j,i)$	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 6. ECC on Sydney with varied edge label definition.

where IRz is equivalent to full isotropy and decreases accuracy to 89.9%.

E. Identity Connections

The formulation of ECC in Equation 1 does not treat self-loop edges in a special way. However, the success of residual networks [18] is a strong motivation to consider adding identity skip-connections to our model and encouraging ECC in residual learning. We thus formulate ECC-resnet as follows:

	NCI1	NCI109	MUTAG	ENZYMES	D&D	Sydney	ModelNet10
ECC-resnet	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 7. The effect of adding identity connections (improvements in italics). Performance metrics vary and are specific to each dataset, as introduced in the main paper.

$$X^l(i) = \frac{1}{|N(i)|} \sum_{j \in N(i)} \Theta^l_{ji} X^{l-1}(j) + b^l + \mathrm{id}(X^{l-1}(i)) \tag{2}$$

where $\mathrm{id}()$ is an identity mapping if $d_l = d_{l-1}$ and a linear mapping otherwise.

The results listed in Table 7 show that 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 in Section 4.1.

F. Vertex Degrees in Edge Labels

In the task of graph classification, we used categorical labels (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 labels more informative by including the degrees of the pair of vertices forming an edge. The degree information is implicitly used by spectral convolution methods, as the degree information is contained in the graph Laplacian, and also appears in the explicit propagation rules [21, 2].

Our model can be easily extended to make use of this information by simply appending it to the existing edge label vectors. We consider four variants of providing additional degree labels $L_{deg}(j)$ 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 $\deg(i) = |N(i)|$ is the degree of vertex $i \in V$. We use these additional labels in all graph resolutions.

Table 8 reveals that degree information can improve the results considerably, especially for datasets without given edge labels (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.

G. Vertex Degrees in Normalization

The formulation of ECC in Equation 1 performs normalization by the neighborhood size. Here we explore learning an additional multiplicative factor, conditioned on the

	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 8. The effect in mean classification accuracy of adding vertex degrees to edge labels (improvements in italics).

	NCI1	NCI109	MUTAG	ENZYMES	D&D	Sydney	ModelNet10
ECC-Z	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 9. The effect of adding a learned normalization factor (improvements in italics). Performance metrics vary and are specific to each dataset, as introduced in the main paper.

neighborhood size $1/|N(i)|$. To this end, we again make use of Dynamic filter networks [5] and design a factor-generating network $Z^l : \mathbb{R} \mapsto \mathbb{R}$ which given vertex degree $\deg(i) = |N(i)|$ outputs a vertex-specific normalization factor. We formulate ECC-Z as follows:

$$X^l(i) = \frac{Z^l(|N(i)|; w^l)}{|N(i)|} \sum_{j \in N(i)} \Theta^l_{ji} X^{l-1}(j) + b^l \qquad (3)$$

In our experiments, the factor-generating networks Z^l have configuration FC(32)-FC(1) with orthogonal weight initialization [33] and ReLUs in between.

The results in Table 9 show that while being helpful on some datasets (NCI109, ENZYMES, ModelNet10), ECC-Z harms the performance on the other ones. Embedding vertex information in labels instead seems to achieve higher performance (Section F).