FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis

Nitika Verma, Edmond Boyer, Jakob Verbeek

Dynamic Filters in Graph Convolutional Networks

Nitika Verma, Edmond Boyer, Jakob Verbeek
Inria, LJK, Université Grenoble Alpes

Abstract

Convolutional neural networks (CNNs) have massively impacted visual recognition in 2D images, and are now ubiquitous in state-of-the-art approaches. While CNNs naturally extend to other domains, such as audio and video, where data is also organized in rectangular grids, they do not easily generalize to other types of data such as 3D shape meshes, social network graphs or molecular graphs. To handle such data, we propose a novel graph-convolutional network architecture that builds on a generic formulation that relaxes the 1-to-1 correspondence between filter weights and data elements around the center of the convolution. The main novelty of our architecture is that the shape of the filter is a function of the features in the previous network layer, which is learned as an integral part of the neural network. Experimental evaluations on digit recognition, semi-supervised document classification, and 3D shape correspondence yield state-of-the-art results, significantly improving over previous work for shape correspondence.

1 Introduction

In recent years, deep learning has dramatically improved the state of the art in several research domains including computer vision, speech recognition, and natural language processing [13]. In particular, convolutional neural networks (CNNs) have now become ubiquitous in computational solutions to visual recognition problems such as image classification [7], semantic segmentation [27], object detection [19], and image captioning [25]. CNNs also extend beyond 2D visual information, and easily generalize to other data that come in the form of regular rectangular grids. This has been demonstrated with for instance 1D convolution for audio signal [17] and 3D convolution over space and time for video signal [22].

Of particular interest beyond 2D image understanding are 3D shape models for which two main categories of representations can be considered. Extrinsic or Eulerian representations are based on parametrizations external to the shape, the most common being voxel grids. Such representations enable standard CNNs to be applied over 3D grids, but lack invariance to even the most basic transformations of the shape. A simple rigid transformation of the shape can lead to significant changes in the 3D grid representation. Moreover, discretizing space, instead of shapes, tends to be inefficient, in particular with moving and deforming objects for which a significant part of the space grid can be empty, hence resulting in representations with often poor shape resolutions, *e.g.* using $20 \times 20 \times 20$ voxel grids [6]. On the other hand, intrinsic or Lagrangian representations, for example 3D meshes, are robust to many shape transformations and describe 3D entities more efficiently with discretizations that are attached to shapes and not the surrounding spaces. However, CNNs do not readily apply to such representations with non-regular structures.

In order to transfer the success achieved by CNNs in 2D visual scene understanding to 3D modeling with meshes or point clouds, a number of CCN-like deep learning methods for such data have been recently proposed [2, 5, 12, 15, 16, 18, 21]. The interest of these techniques does not only concern 3D shape modeling, but extends more generally to domains where data can be organized into graph structures, including for instance social networks or molecular graphs [4, 11]. Previous works on deep networks over graph-structured data can be divided into spectral and local approaches. Spectral

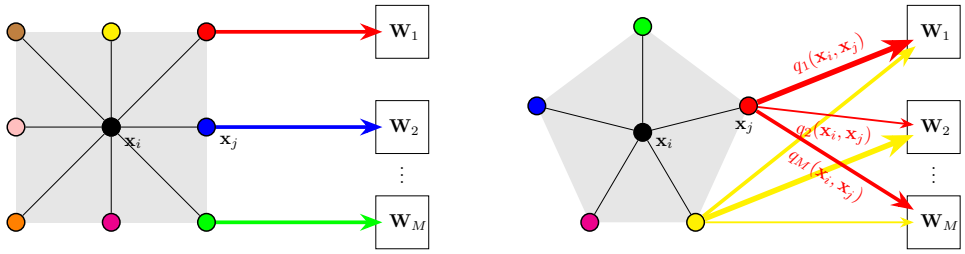


Figure 1: In CNNs for pixel grids (left), there is a 1-to-1 mapping between neighbors $\mathbf{x}_j$ of the center pixel $\mathbf{x}_i$ and filter weights. In our graph-convolutional approach (right), neighbors are soft-assigned across all weights, the assignments $q_m(\mathbf{x}_i, \mathbf{x}_j)$ are computed using features from the preceding layer.

filtering approaches [4, 5, 8, 11] rely on the eigen-decomposition of the graph Laplacian that enables convolutions over graphs. While useful for problems with a fixed graph representation, the non-local eigen-decomposition is unstable across different graphs, which makes the generalization across, *e.g.*, different shapes difficult [16]. In contrast, local filtering approaches [2, 15, 16] make use of mappings, between local graph neighborhoods and filter weights, to perform convolutions. However, it remains challenging to define such mappings in a consistent and principled way.

In this paper we present a novel graph convolutional neural network based on local filtering. In our approach, the mapping between the filter weights and the nodes in a neighborhood of the graph is learned as an integral part of the network. Moreover, the mapping is a function of the features in the preceding layer of the network, rather than based on manually defined local coordinates on the graph as in previous works. See Figure 1 for a schematic illustration. Experiments on digit recognition, document classification, and 3D shape correspondence validate our model. They demonstrate that, for 3D shape correspondence, our approach significantly outperforms recent state-of-the-art approaches. Another important result is that the best shape correspondences were obtained with 3D shape coordinates as input in our approach, where previous work is based on 3D shape descriptors.

2 Related work

In this section we briefly review related work on graph-convolutional networks, other deep learning approaches to process 3D shape data, and related data-adaptive CNNs.

Graph-convolutional networks. Existing approaches to generalize convolutional networks to non-regular graph-structured data can be divided into two broad categories: spectral filtering methods and local filtering methods. Spectral methods build on a mathematically elegant approach to develop convolution-like operators over graphs based on the spectral eigen-decomposition of the graph Laplacian [4, 5, 8, 11]. Any function defined over the graph nodes, *e.g.* features, can be mapped, by projection on the eigenvectors of the Laplacian, to the spectral domain where filtering consists of scaling the signals in the eigenbasis. While successful with noise-free data such as synthetic 3D shape models, spectral techniques have more difficulties with real observed data for which global decompositions may be unstable across, for instance, various shapes in various poses.

In an effort to better generalize over different graphs, *e.g.* different shape meshes, a number of techniques follow a different strategy based on local graph filtering [2, 15, 16]. These methods differ then in how they establish a correspondence between filter weights and nodes in local graph neighborhoods. The geodesic CNN model of Masci *et al.* [15] extract local patches on meshes which are convolved with filters expressed in polar coordinates. The orientation ambiguity of filters is dealt with by means of angular max-pooling, *i.e.* filters are applied in all possible orientations, and the maximum responses are retained. Boscani *et al.* [2] proposed the anisotropic CNN model which further extends the geodesic CNN model by using an anisotropic patch-extraction method, exploiting the maximum curvature directions to orient patches. Monti *et al.* [16] also parameterize local patches of the graph using hand-designed local pseudo-coordinates around each node. However, in contrast to the previously mentioned works, their method learns filter shapes by estimating the means and variances of Gaussians that associate filter weights to the local pseudo-coordinates. Our approach is closely related to the latter, though, instead of considering hand-designed local pseudo-coordinates,

in our approach we use the features of the preceding layer to map between local graph patches and filter weights.

Deep networks for 3D shape data. Besides spectral and local filtering approaches on graphs, a number of other techniques have been developed to handle 3D shape data in deep neural networks. Sinha *et al*. [21] use a spherical parametrization, filling holes in the mesh when needed, to map shapes onto octahedra. These octahedra are cut and unfolded to square images, which can then be processed using regular CNNs. Wei *et al*. [24] render depth maps of shapes, and process them with conventional CNNs to learn features that can be matched to establish shape correspondence. These two approaches transform 3D shape input data in order to adapt them to regular CNNs, in our work we instead propose a novel graph convolution that exploits the intrinsic graph structure of the data.

Recently other architectures have been proposed that are suitable for point clouds, another form of irregular input data that falls outside the scope of this paper. Klokov *et al*. [12] developed a deep network based on kd-trees over 3D point clouds, sharing parameters across the tree based on the depth of a split and the direction of split. Qi *et al*. [18] developed another approach for 3D point clouds, which combines local per-point processing layers, with global max-pooling layers. In the latter approach, spatial proximity information is only encoded in the input data and not used in the processing layers.

Data-adaptive convolutional networks. The convolutional layers in a conventional CNN multiply together activations of the preceding feature map and filter weights, and sum the results to obtain the output as a linear function of the input, after which a point-wise non-linearity is applied. In spatial transformer networks [9] and dynamic filter networks [3], a subnetwork, that takes the preceding feature map as input, replaces a standard convolutional layer with a data-adaptive transformation. In the former, a localization subnetwork computes the parameters of a spatial transformation, *e.g*. cropping and/or re-sizing, which is used to spatially re-sample the preceding feature map before convolution. In the latter, a subnetwork is used to generate the convolutional filters that will be applied to the preceding feature maps. Our approach uses similar techniques to define convolutions over graph structured data, in particular we use a subnetwork that locally assigns the elements of a local "patch" of the graph to the filter weights.

3 Graph convolutions using dynamic filters

Before we describe our approach to define convolutional filters over graph-structured data in Section 3.2, we first reformulate conventional CNNs in Section 3.1. We compare the number of parameters and computational cost to conventional CNNs in Section 3.3.

3.1 Reformulating convolutional layers in regular CNNs

In the case of standard CNNs for *e.g*. image data, the parameters of a convolutional layer that maps D-dimensional input features to E-dimensional output features are commonly represented as a set of $D \times E$ filters $\mathbf{F}_{d,e}$, each of size $h \times w$ pixels, see left panel of Figure 2. The computations in the convolutional layer to compute one of the E output channels can then be described as convolving each of the D input channels with the corresponding filters, summing the D convolution results and adding a constant bias to compute the output feature map.

In an equivalent, but less common representation, we rearrange the weights as a set of $M = h \times w$ weight matrices $\mathbf{W}_m \in \mathbb{R}^{E \times D}$, see right panel of Figure 2. Each of these weight matrices is used to project input features in $\mathbb{R}^D$ to output features in $\mathbb{R}^E$. The result of the convolution is obtained by summing for each of the M neighbors of the central pixel of the convolution the projection corresponding to its relative position (considering that pixel i is a neighbor of itself). The computation of the activation $\mathbf{y}_i \in \mathbb{R}^E$ of pixel i in the output feature map is then written as

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

where $\mathbf{b} \in \mathbb{R}^E$ denotes a vector of bias terms, and $j(m, i)$ gives the index of the pixel in the m-th relative position w.r.t. i. For example, $j(1, i) = i$ refers to the center pixel of the convolution, $j(2, i)$ refers to the pixel one position to the left and top w.r.t. pixel i, and so on. See also Figure 1 for an illustration of how in the conventional CNN each neighbor is mapped to a single weight matrix $\mathbf{W}_m$.

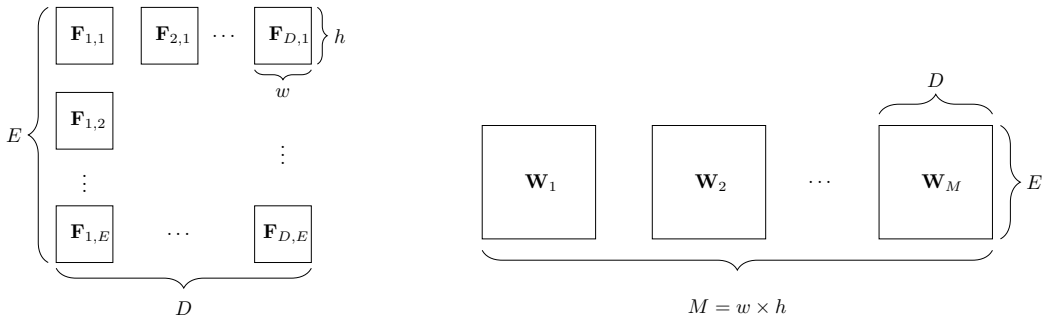


Figure 2: Weights in a convolutional layer of a conventional CNN. Left: Representation as $D \times E$ filters $\mathbf{F}_{d,e}$, each of size $w \times h$, that are used to convolve the input feature maps. Right: Representation as $M = w \times h$ weight matrices, each of size $E \times D$, that map input features to output features.

3.2 Generalization to non-regular input domains

In the case of CNNs for regular inputs, *e.g*. pixel grids, there is a clear one-to-one mapping between the neighbors and the weight matrices $\mathbf{W}_m \in \mathbb{R}^{E \times D}$. The main challenge in the case of non-regular data graphs is to establish this correspondence between neighbors and weight matrices. Previous work on local graph convolutions either uses a single weight matrix which obviates the need for a mapping [11], or the mapping is designed a-priori [2, 15], or learned based on a manually determined local coordinate systems over the graph [16]. In contrast, we propose to learn the mapping as an integral part of the neural network, using features computed in the preceding layer of the network rather than relying on an engineered representation.

Instead of assigning each neighbor j of a node i to a single weight matrix, we use a soft-assignment $q_m(\mathbf{x}_i, \mathbf{x}_j)$ across the M weight matrices. Given these soft-assignments, we define the function that maps the features from one layer to the next as

$$\mathbf{y}_i = \mathbf{b} + \sum_{m=1}^{M} \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} q_m(\mathbf{x}_i, \mathbf{x}_j) \mathbf{W}_m \mathbf{x}_j, \tag{2}$$

where $q_m(\mathbf{x}_i, \mathbf{x}_j)$ is the assignment of $\mathbf{x}_j$ to the m-th weight matrix, and $\mathcal{N}_i$ is the set of neighbors of i (including i), and $|\mathcal{N}_i|$ it's cardinal.

We define the weights as a function of the local feature vectors as

$$q_m(\mathbf{x}_i, \mathbf{x}_j) \propto \exp\left(\mathbf{u}_m^\top \mathbf{x}_i + \mathbf{v}_m^\top \mathbf{x}_j + c_m\right), \tag{3}$$

with $\sum_{m=1}^{M} q_m(\mathbf{x}_i, \mathbf{x}_j) = 1$. The weights involved in the update of node i sum to 1 regardless of the number of neighbors of a node, since $\sum_{j \in \mathcal{N}_i} \frac{1}{|\mathcal{N}_i|} \sum_{m=1}^{M} q_m(\mathbf{x}_i, \mathbf{x}_j) = \sum_{j \in \mathcal{N}_i} \frac{1}{|\mathcal{N}_i|} = 1$. Therefore, our formulation is robust to variations in the degree of the nodes. Instead of using a single linear transformation of the features in Eq. (3), more general transformations may be used, such as an MLP. Conventional CNNs over grid-graphs are recovered if $\forall_i |\mathcal{N}_i| = M$, and the assignments are binary, *i.e*. $q_m(\mathbf{x}_i, \mathbf{x}_j) \in \{0, 1\}$, based on the relative position of neighbors w.r.t. node i. In Figure 3 we give a schematic illustration of the computations in a standard CNN and in our graph convolutional network.

In our experiments, $\mathcal{N}_i$ contains vertex i and all vertices connected to i by an edge, *i.e*. the first *ring* around vertex i. Our approach, however, allows using larger neighborhoods, *e.g*. up to ring 2 or more. This is analogous to filters with a larger support in conventional CNNs. Importantly, and in contrast to standard CNNs, the above formulation decouples the neighborhood size $|\mathcal{N}_i|$ from the number M of weight matrices, and thus the number of parameters. As a consequence, filters with larger supports do not necessarily increase the number of parameters. Rather than forcing weight-sharing patterns or using dilated convolutions [27] for large filters with few parameters, in our approach we learn the mapping between an arbitrary number of neighbors and a limited fixed set of filter weights.

Translation invariant assignments in feature space. As a special case, we can set $\mathbf{u}_m = -\mathbf{v}_m$ in Eq. (3), which results in $q_m^{ij} \propto \exp\left(\mathbf{u}_m^\top (\mathbf{x}_j - \mathbf{x}_i) + c_m\right)$, and leads to translation invariance

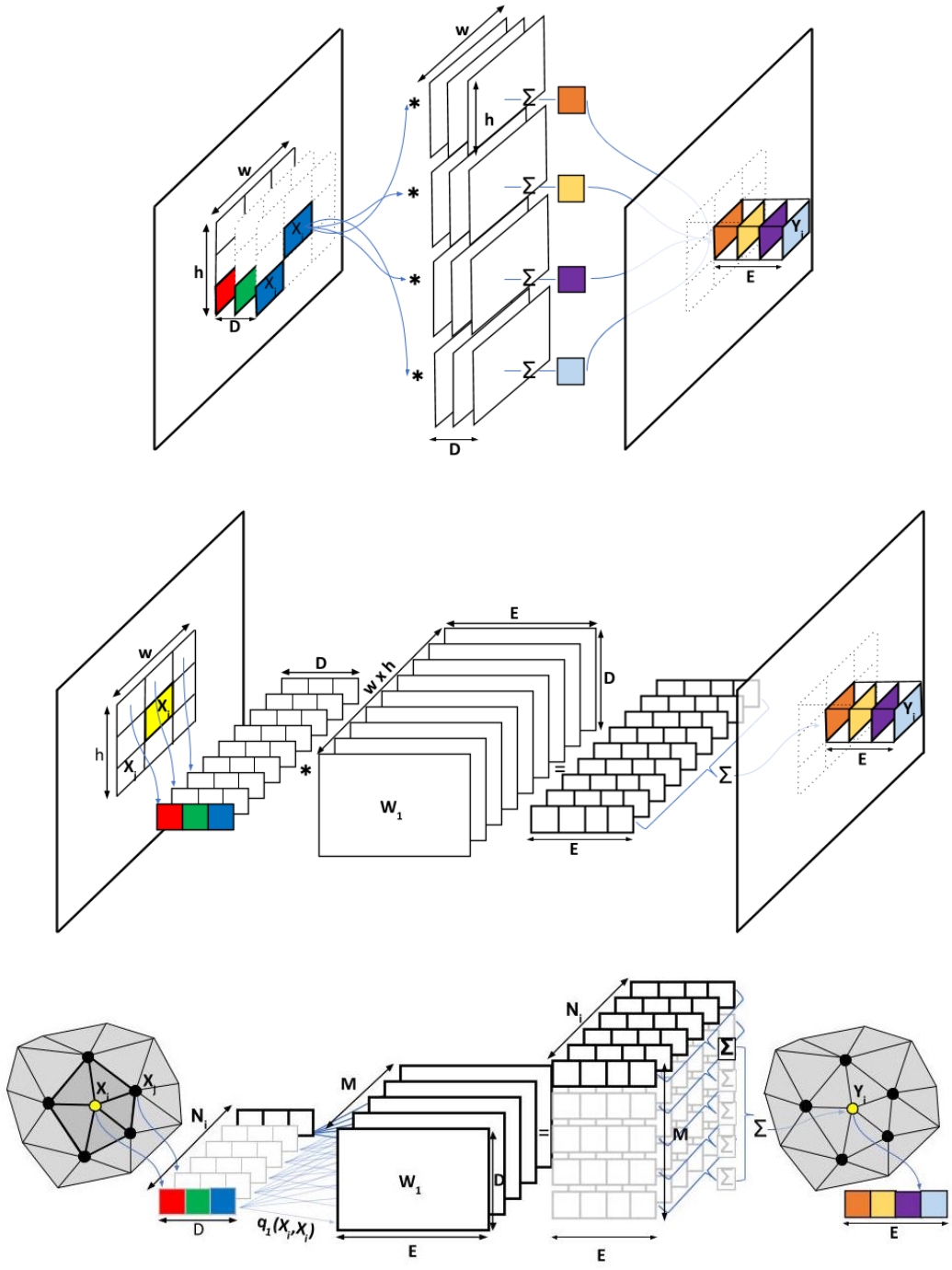


Figure 3: Top: Schematic illustration of a standard CNN where patches of $w \times h$ pixels are convolved with $D \times E$ filters to map the D dimensional input features to E dimensional output features. Middle: same, but representing the CNN parameters as a set of $M = w \times h$ weight matrices, each of size $D \times E$. Each weight matrix is associated with a single relative position in the input patch. Bottom: our graph convolutional network, where each relative position in the input patch is associated in a soft manner to each of the M weight matrices using the function $q(\mathbf{x}_i, \mathbf{x}_j)$.

of the weights in the feature space. This is of particular interest in applications where the input features include spatial coordinates, in which case it is natural to impose translation invariance on the assignment function. In our experiments we demonstrate that translation invariance is critical when directly using spatial coordinates as input features for 3D shape meshes.

Assignment by Mahalanobis distance in feature space. Another interesting case occurs when considering a Mahalanobis distance to determine the assignments weights $q_m(\mathbf{x}_i, \mathbf{x}_j)$. The Mahalanobis distance, parameterized by a positive definite matrix Σ, between reference points $\mathbf{z}_m$ and a centered version of the neighbor features $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ is given by

$$d_\Sigma(\mathbf{x}_{ij}, \mathbf{z}_m) = (\mathbf{x}_{ij} - \mathbf{z}_m)^\top \Sigma (\mathbf{x}_{ij} - \mathbf{z}_m) \tag{4}$$

$$= -2\mathbf{x}_{ij}^\top \Sigma \mathbf{z}_m + \mathbf{z}_m^\top \Sigma \mathbf{z}_m + c, \tag{5}$$

where c is a constant independent of m. It is easy to verify that soft-assignments based on the above Mahalanobis distances fits the form of Eq. (3) with $c_m = \mathbf{z}_m^\top \Sigma \mathbf{z}_m + c$, and $\mathbf{u}_m = -2\Sigma\mathbf{z}_m$, and $\mathbf{v}_m = -\mathbf{u}_m$. These soft-assignments may be interpreted as the posterior assignments of the neighbor's centered feature vectors $\mathbf{x}_{ij}$ over the components of a Gaussian mixture model in feature space with components centered at the $\mathbf{z}_m$ and with a shared covariance matrix Σ^{-1}.

This mixture model interpretation of the soft-assignments also helps identifying the connection with the related work of Monti *et al*. [16]. In this work, a similar formulation is used in which centers $\mathbf{z}_m$ are learned along with covariance matrices Σ_m. This mixture is, however, defined over an a-priori defined local coordinate space over the graph (*e.g*. local log-polar coordinates over a mesh), rather than learned features as in our formulation.

Using this formulation, we can recover a conventional grid-graph CNNs as a special case by letting the pixel coordinates be part of the feature vectors $\mathbf{x}$, having the Mahalanobis distance depends only on these coordinates, and placing the centers $\mathbf{z}_m$ precisely on the relative positions of the neighboring pixels. Multiplying the Mahalanobis distances by a large constant will recover the hard-assignments used in the standard CNN model of Eq. (1).

3.3 Number of parameters and computational complexity

The weight matrices $\mathbf{W}_m$ are common between a conventional CNN and our approach, and contain MDE parameters. The, only, additional parameters in our approach w.r.t. a conventional CNN are the vectors $\mathbf{u}_m, \mathbf{v}_m$, which contain $2MD$ parameters. Thus the total number of parameters increases only by a factor $1 + 2/E$. Here we ignored bias terms, which contribute very few parameters.

To efficiently evaluate the activations, we first multiply all feature vectors $\mathbf{x}_i$ with the weight matrices $\mathbf{W}_m$ and weight vectors $\mathbf{u}_m$, and $\mathbf{v}_m$. This takes $\mathcal{O}(NMDE)$ operations, where N is the number of nodes in the graph. Let K denote the average number of neighbors of each vertex, we can then compute the weights in Eq. (3) and the activations in Eq. (2) in $\mathcal{O}(NMKE)$ operations. The total computational cost is thus $\mathcal{O}(NME(K + D))$.

The cost of a convolutional layer in a conventional CNN is $\mathcal{O}(NMED)$, *c.f*. Eq. (1). The computational cost of our approach is comparable, provided the number of neighbors K is comparable or smaller than the number of features D, as is typically the case in practice.

4 Experimental evaluation

We experimentally evaluated our approach on three different problems, handwritten digit classification, document classification, and 3D shape correspondence, as described in the sections below.

4.1 Handwritten digit classification

We first evaluate our approach on the classification of handwritten digits in the MNIST dataset [14]. The dataset consists of 28×28 images, split into 55k train samples, 5k validation samples and 10k test samples. We represent the images as regular 8-connected grid graphs with 784 vertices, and use three input channels: the pixel intensity and the 2D pixel coordinates.

For our baseline conventional CNN model, we use an architecture similar to LeNet5 [14] in which we use two convolutional layers (Conv32 + Conv64), followed by a fully connected layer (FC1024),

Table 1: Classification accuracies on MNIST for graph convolutional networks and CNN baseline.

Method	Accuracy
Baseline CNN	99.2%
Ours, non-translation invariant	97.8%
Ours, translation invariant	98.4%
MoNet [16]	99.2%
ChebNet [5]	99.1%

and insert max-pooling layers after each convolutional layer. For our graph convolutional network we used a similar architecture with two convolutional layers (Conv32 + Conv64), followed by a fully connected layer (FC1024). The graph pooling layer proposed by Defferrard *et al*. [5] and also used in [16] can also be applied in our approach, but we did not include it for simplicity. The baseline CNN uses 3×3 filters, and for direct comparison we set $M = 9$ in our approach. We use the Adam [10] optimization algorithm, learning rate 10^{-3}, batch size 20, and did not use data augmentation.

We present the accuracy of our model, the CNN baseline, and other recent graph convolutional approaches [5, 16] in Table 1. The results clearly show that our approach learns effective filters shaped by dynamically computed features. As expected, using translation invariance has a positive effect on performance. Our results are somewhat below the baseline CNN and other graph convolutional approaches, which is probably explained by the lack of pooling layers in our architecture.

4.2 Semi-supervised document classification

In our second set of experiments we consider semi-supervised document classification on the Cora and PubMed datasets [20]. These datasets contain scientific papers divided into seven and three classes respectively. The data is organized in a citation graph, where documents correspond to vertices, and citations are reflected by edges. Each document (2,708 in Cora and 19,717 in PubMed) is represented by sparse bag-of-words feature vector of dimension 1,433 for Cora and 500 for PubMed. There are 5,429 and 44,338 citation links in Cora and PubMed respectively. We followed the experimental setup and the dataset split as given in [11, 26]. There are 20 training samples per class and 500 and 1,000 vertices in the validation and test set respectively. We used two convolutional layers (Conv16 + Conv16). We used the validation data to choose the number of weight matrices M, learning rate, and the weight-decay level.

For M, we tried values from 1 upto 32, and found $M = 1$ to give best results on the validation data. This suggests that for this data that comes as a single graph, our model did not succeed in learning documents features to define filter shapes, and instead uses a uniform filter over the graph, just like [11]. The results in Table 2 show that the classification accuracy we obtain is comparable to the results of other recent graph convolutional approaches [11, 16], and significantly improves over the graph embedding approach of [26].

4.3 3D shape correspondence

In our third set of experiments we followed the experimental setup in [2, 15, 16] based on the FAUST human shape dataset [1]. This dataset consists of 100 watertight meshes with 6,890 vertices each, corresponding to 10 shapes in 10 different poses each. The shape correspondence problem, between a given reference shape and any other shape, is here formulated as a vertex labeling problem where

Table 2: Classification accuracy on the Cora and PubMed document classification datasets.

Method	Cora	PubMed
Planetoid [26]	75.7%	77.2%
GCN [11]	81.6%	78.7%
MoNet [16]	81.7%	78.8%
Ours	81.6%	79.0%

Table 3: Comparison of input features and the effect of translation invariance, using $M = 9$.

Descriptor	Accuracy	Acc. Trans.-inv.
XYZ	28%	86%
SHOT	58%	63%

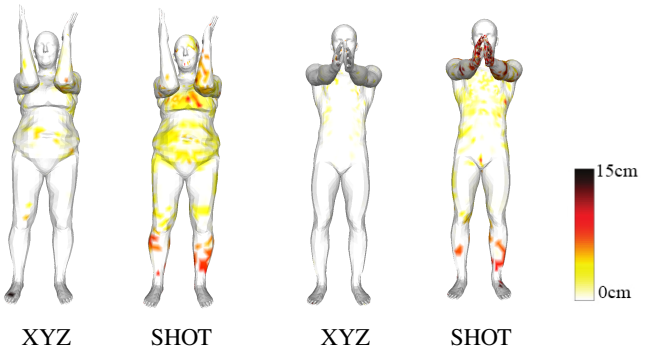


Figure 4: Geodesic errors between the true and the estimated vertex locations using XYZ and SHOT feature inputs for two test shapes.

the label set consists of all the 6,890 vertices on the reference shape. The first 80 shape meshes are used as training data, and the last 20 meshes are used as test data (corresponding to the 10 poses of two shapes not seen during training). Exact ground-truth correspondence is known, and the first shape in the first pose is used as reference. The output of the last soft-max layer at each vertex gives a probability distribution over corresponding vertex on the reference shape. We replicated the architecture used in [2], which is also similar to the one used in [15, 16], see Table 4 for details. The model is trained using the standard cross-entropy classification loss. We use learning rate as 10^{-2}, weight decay of 10^{-4}. We use the 544-dimensional SHOT descriptor used in earlier work, but also experiment with the raw 3D XYZ vertex coordinates as input. The accuracy is defined as the number of vertices for which the correspondence prediction is exact.

We first evaluated performances using either the XYZ coordinates or the SHOT descriptor as input features, and with and without translation invariance in our model. Here we used $M = 9$ as a reference value. The results in Table 3 show that translation invariance improves results for both descriptors and is critical for the raw XYZ inputs, for which translation invariance is natural. We note that the XYZ inputs clearly outperforms the SHOT descriptor inputs in our settings. Geodesic correspondence errors can be visualised in Figure 4 for both descriptors, with the translation invariant model.

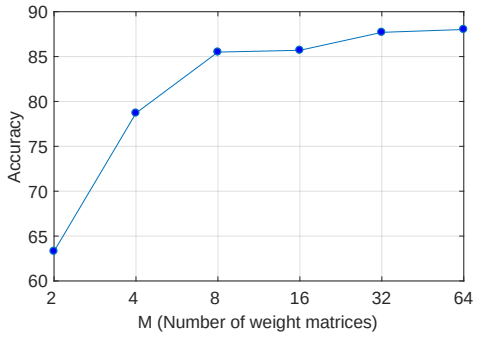


Figure 5: Accuracy as a function of the number of weight matrices for the FAUST dataset.

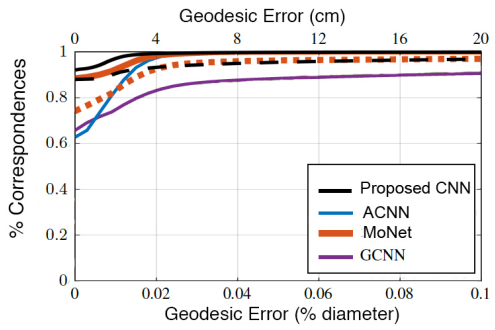


Figure 6: Shape correspondence on FAUST, solid curves include post-processing.

Table 4: Raw classification accuracy for 3D shape correspondence on the FAUST dataset, without correspondence refinement post-processing. Accuracies for [2, 15, 16] are estimated from graphs in the corresponding papers, see Figure 6 for a more detailed comparison.

Method	Input	Architecture	Accuracy
Baseline	SHOT	Linear classifier	40%
ACNN [2]	SHOT	Lin64+Conv64+Conv128+Conv256+ Lin1024+Lin512+Lin6890	$\approx 60\%$
GCNN [15]	SHOT	Lin16+Conv32+Conv64+Conv128+Lin256+Lin6890	$\approx 60\%$
MoNet [16]	SHOT	3 Convolutional layers, similar to [2, 15]	$\approx 70\%$
Ours	XYZ	Same as [15], translation invariant, $M = 32$	88%

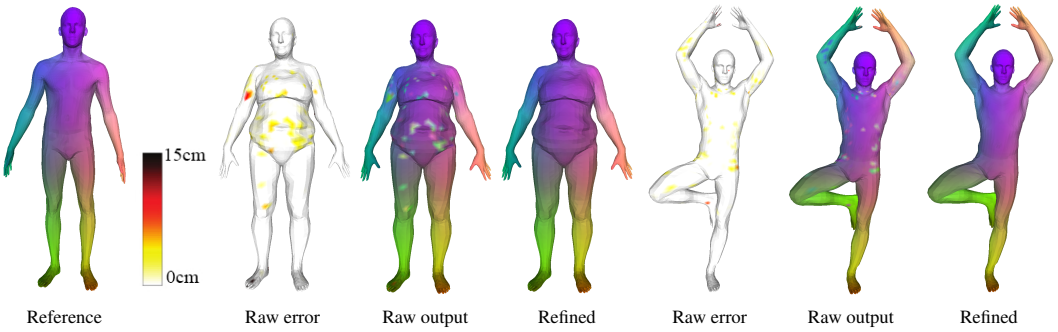


Figure 7: Texture transfer from the reference shape (left) to two different shapes using the predicted correspondences before and after refinement with with [23], as well as point-wise geodesic errors.

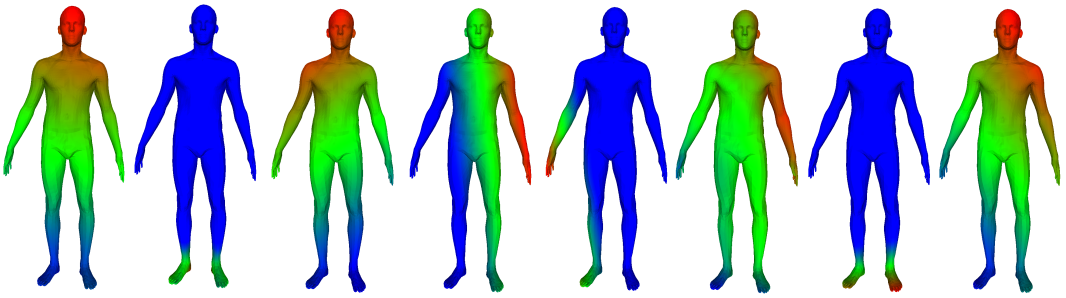
In our next experiment we evaluate the impact of the number of weight matrices M. The results in Figure 5 show that the performance quickly improves from $M = 2$ to $M = 8$, after which the improvements are smaller. These results show that our model learns effective non-uniform filters over the graph, without relying on pre-defined local coordinates. We use $M = 32$ for the remaining experiments.

In Table 4, we compare the classification accuracy obtained with our model with other recent state-of-the-art graph convolutional methods. In Figure 6 we plot the percentage of correspondences that are within a given geodesic distance from the ground-truth on the reference shape. Besides evaluating the quality of the matches directly predicted by our model, we also assessed the impact of a post-processing the matches using a correspondence refinement algorithm [23]. Dashed curves are obtained before correspondence refinement post-processing, solid curves are obtained after refinement. The correspondences predicted by our network significantly improve over earlier results (dashed curves). Also after refinement our results are still superior (solid curves). In Figure 7 we visualize the correspondence errors for two shapes, and the effectiveness of correspondence refinement to correct the sparsely distributed errors.

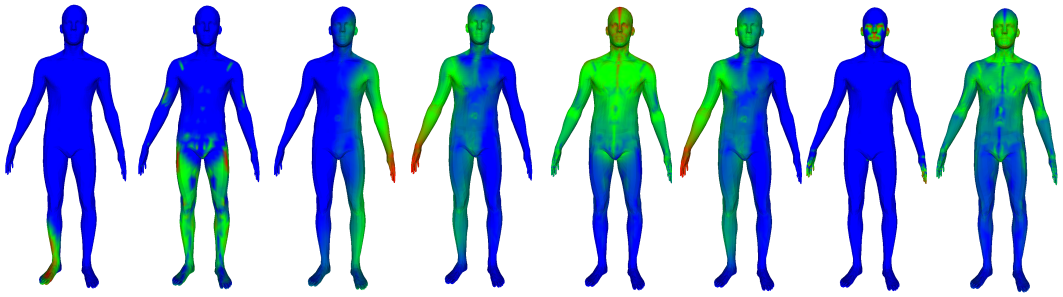
In Figure 8, we show the feature activations across the layers of our graph convolutional network. In the earlier layers the activations are smooth, and gradually become more sparse and localized as required by the task.

5 Conclusion

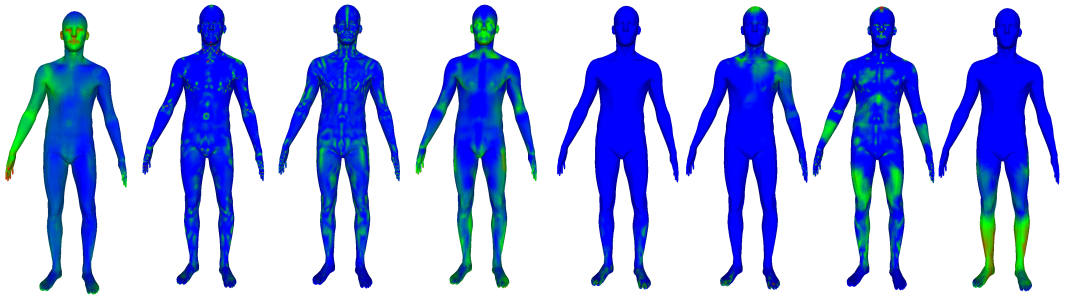
We have presented a novel graph-convolutional architecture that is based on local filtering and applies to generic graph structures, both regular and irregular. The main novelty is that our architecture determines local filters dynamically based on the features in the preceding layer of the network. The network thus learns features that are (i) effective to shape the local filters, and (ii) informative for the prediction task. Experimental results validate our architecture and shows that it yields performances comparable to recent state-of-the-art graph-convolutional approaches for digit recognition and document classification and clearly outperforms the state-of-the-art for 3D shape correspondence. Importantly, we obtain this improvement over previous work, while only using 3D spatial coordinates as input, where previous work relied on precomputed 3D shape descriptors. In the future we plan to



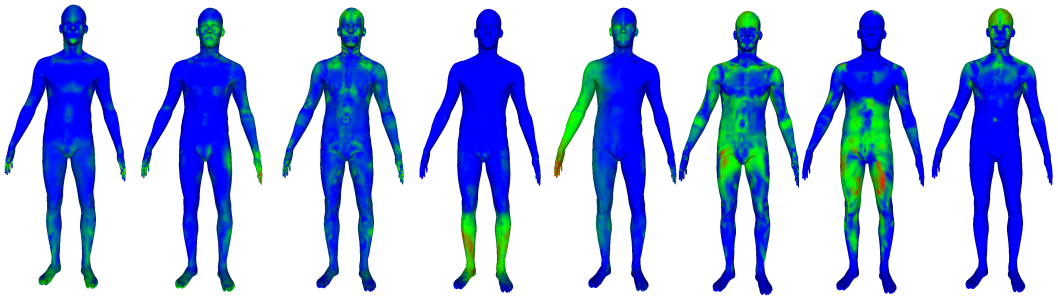
Layer 1 (Lin16)



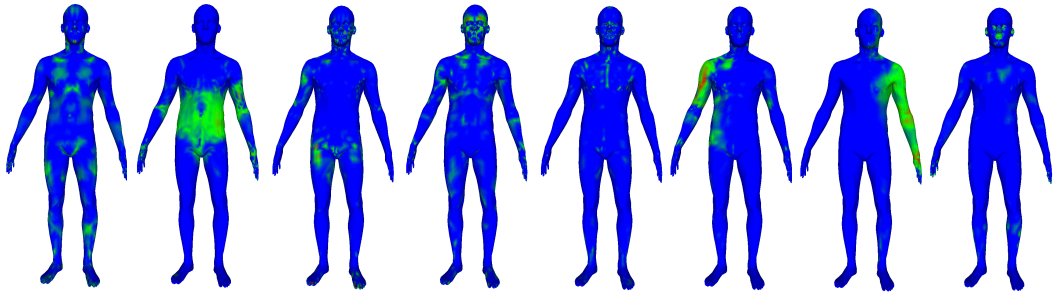
Layer 2 (Conv32)



Layer 3 (Conv64)



Layer 4 (Conv128)



Layer 5 Lin(256)

Figure 8: Visualization of activations of random features across the layers of our graph convolutional network (translation-invariant, xyz-coordinate input).

extend our architecture to model other properties of 3D shapes, such as appearance or motion patterns. Another intriguing future direction is the use of these type of models for 3D shape acquisition.

Acknowledgment. We would like to thank NVIDIA for the donation of GPUs used in this research. This work was partially supported by the grant ANR-16-CE23-0006 Deep in France, and the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01).

References

[1] F. Bogo, J. Romero, M. Loper, and M. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *CVPR*, 2014.

[2] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *NIPS*, 2016.

[3] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool. Dynamic filter networks. In *NIPS*, 2016.

[4] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014.

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

[6] R. Girdhar, D. Fouhey, M. Rodriguez, and A. Gupta. Learning a predictable and generative vector representation for objects. In *ECCV*, 2016.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[8] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[9] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *NIPS*, 2015.

[10] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[11] T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[12] R. Klokov and V. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3D point cloud models. *arXiv preprint arXiv:1704.01222*, 2017.

[13] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 52:436–444, 2015.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, pages 2278–2324, 1998.

[15] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on Riemannian manifolds. In *ICCV Workshops*, 2015.

[16] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR*, 2017.

[17] D. Palaz, M. Magimai-Doss, and R. Collobert. Analysis of CNN-based speech recognition system using raw speech as input. In *InterSpeech*, 2015.

[18] C. Qi, H. Su, K. Mo, and L. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[19] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[20] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[21] A. Sinha, J. Bai, and K. Ramani. Deep learning 3d shape surfaces using geometry images. In *ECCV*, 2016.

[22] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3D convolutional networks. In *ICCV*, 2015.

[23] M. Vestner, R. Litman, E. Rodola, A. Bronstein, and D. Cremers. Product manifold filter: Non-rigid shape correspondence via kernel density estimation in the product space. *arXiv preprint arXiv:1701.00669*, 2017.

[24] L. Wei, Q. Huang, D. Ceylan, E. Vouga, and H. Li. Dense human body correspondence using convolutional networks. In *CVPR*, 2016.

[25] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

[26] Z. Yang, W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

[27] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.