Graph-based Isometry Invariant Representation Learning

Renata Khasanova [1] **Pascal Frossard** [1]

Abstract

Learning transformation invariant representations of visual data is an important problem in computer vision. Deep convolutional networks have demonstrated remarkable results for image and video classification tasks. However, they have achieved only limited success in the classification of images that undergo geometric transformations. In this work we present a novel Transformation Invariant Graph-based Network (TIGraNet), which learns graph-based features that are inherently invariant to isometric transformations such as rotation and translation of input images. In particular, images are represented as signals on graphs, which permits to replace classical convolution and pooling layers in deep networks with graph spectral convolution and dynamic graph pooling layers that together contribute to invariance to isometric transformation. Our experiments show high performance on rotated and translated images from the test set compared to classical architectures that are very sensitive to transformations in the data. The inherent invariance properties of our framework provide key advantages, such as increased resiliency to data variability and sustained performance with limited training sets.

1. Introduction

Deep convolutional networks (ConvNets) have achieved impressive results for various computer vision tasks, such as image classification (Krizhevsky et al., 2012) and segmentation (Ronneberger et al., 2015). However, they still suffer from the potentially high variability of data in high-dimensional image spaces. In particular, ConvNets that are trained to recognize an object from a given perspective or
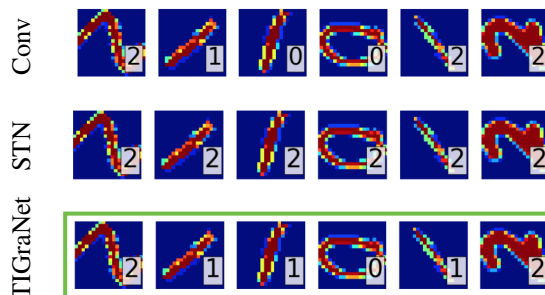

Figure 1. Illustrative transformation-invariant handwritten digit classification task. Rotated test images, along with their classification label obtained from ConvNets (Conv) (Boureau et al., 2010), Spatial-Transformer Network (STN) (Jaderberg et al., 2015), and our method. (best seen in color)

camera viewpoint, will likely fail when the viewpoint is changed or the image of the object is simply rotated. In order to overcome this issue the most natural step is to extend the training dataset with images of the same objects but seen from different perspectives. This however increases the complexity of data collection and more importantly leads to the growth of the training dataset when the variability of the data is high.

Instead of simply augmenting the training set, which may not always be feasible, one can try to solve the aforementioned problem by making the classification architecture invariant to transformations of the input signal as illustrated in Fig. 1. In that perspective, we propose to represent input images as signals on the grid graph instead of simple matrices of pixel intensities. The benefits of this representation is that graph signals do not carry a strict notion of orientation, while at the same time, signals on a grid graph stay invariant to translation. We exploit these properties to create features that are invariant to isometric transformations and we design new graph-based convolutional and pooling layers, which replace their counterparts used in the classical deep learning settings. This permits preserving the transformation equivariance of each intermediate feature representation under both translation and rotation of the input signals. Specifically, our convolutional layer relies on filters that are polynomials of the graph Laplacian for effective signal representation without computing eigendecompositions of the graph signals. We further introduce a new

[1] Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. Correspondence to: Renata Khasanova <renata.khasanova@epfl.ch>, Pascal Frossard <pascal.frossard@epfl.ch>.

statistical layer that is placed right before the first fully-connected layer of the network prior to the classification. This layer is specific to our graph signal representation, and in turn permits combining the rotation and translation invariance features along with the power of fully-connected layers that are essential for solving the classification task. We finally design a complete architecture for a deep neural network called TIGraNet, which efficiently combines spectral convolutional, dynamic pooling, statistical and fully-connected layers to process images represented on grid graphs. We train our network in order to learn isometric transformation invariant features. These features are used in sample transformation-invariant image classification tasks, where our solution outperforms the state-of-the-art algorithms for handwritten digit recognition and classification of objects seen from different viewpoints.

In summary, we propose the following contributions:

- We design a new graph-based deep learning framework that learns isometric invariant features;
- We propose a new sta tistical layer that leads to effective transformation-invariant classification of images described by graph-based features;
- Through experiments, we show that our method correctly classifies rotated or translated images even if such deformations are not present in the training data.

The remainder of the paper is organized as follows. In Section 2 we describe the related work. Section 3 reviews elements of graph signal processing, which are later used to design graph filters. Our new graph-based architecture is presented in details in Section 4. Finally, our experiments and their analysis are presented in Section 5.

2. Related work

Most of the recent architectures (LeCun et al., 2001; Krizhevsky et al., 2012) have been very successful in processing natural images, but not necessarily in properly handling geometric transformations in the data. We describe below some of the recent attempts that have been proposed to construct transformation-invariant architectures. We further quickly review the recent works that extend deep learning data represented on graphs or networks.

2.1. Transformation-invariant deep learning

One intuitive way to make the classification architectures more robust to isometric transformations is to augment the training set with transformed data (e.g., (Dyk & Meng, 2012)), which however, increases both the training set and training time. Alternatively, there have been works that incorporate sort of data augmentation inside the network learning framework. The authors in (Fasel & Gatica-Perez, 2006) construct deep neural networks that operate in parallel on the original and transformed images simultaneously with weight-shared convolutional filters. Then, the authors in (Laptev et al., 2016) propose to use max-pooling to combine the outputs of these networks. A different approach was proposed in (Jaderberg et al., 2015), where the authors introduce a new spatial transformer layer that deforms images according to a predefined transformation class. Then, the work in (Marcos et al., 2016) suggests using rotated filter banks and a special max pooling operation to combine their outcomes and improve invariance to transformations. The authors in (Cohen & Welling, 2016) propose a generalization of the ConvNets and introduce equivariance to $90°$ rotations and flips. Finally, the authors in (Dieleman et al., 2015) exploit rotation symmetry in the Convolutional Network for the specific problem of galaxy morphology prediction. This work has been extended in (Dieleman et al., 2016) which introduces an additional layer that makes the network to be partially invariant to rotations. All the above methods, however, still need to be trained on a large dataset of randomly rotated images in order to be rotation invariant and achieve effective performance.

Contrary to the previous methods, we propose to directly learn feature representations that are invariant to isometric data transformations. With such features, our architecture preserves all the advantages of deep networks, but additionally provides invariance to isometric geometric transformations. The methods in (Oyallon & Mallat, 2015; Bruna & Mallat, 2013; Worrall et al., 2016) are the closest in spirit to ours. In order to be invariant to local transformation, the works in (Oyallon & Mallat, 2015; Bruna & Mallat, 2013) propose to replace the classical convolutional layers with wavelets, which are stable to some deformations. The latter achieves high performance on texture classification task, however it does not improve the performance of supervised ConvNets on natural images, due to the fact that the final feature representations are too rigid and unable to adapt to a specific task. Further, (Mathieu et al., 2014; Rippel et al., 2015) propose to use convolutional filters in Fourier domain to reduce complexity. The latter introduces spectral pooling to truncate the representation in the frequency domain. Finally, a recent work (Worrall et al., 2016) proposes a so called Harmonic Network, which uses specifically designed complex valued filters to make feature representations equivariant to rotations. This method, however, still requires the training dataset to contain examples of rotated images to achieve its full potential. On the other hand, we propose building features that are inherently invariant to isometric transformations, which allows us to train more compact networks and achieve state-of-the-art results.

2.2. Deep learning and graph signal processing

While there has been a lot of research efforts related to the application of deep learning methods to traditional data

like 1-D speech signals or 2-D images, it is only recently that researchers have started to consider the analysis of network or graph data with such architectures (Kipf & Welling, 2016; Henaff et al., 2015; Duvenaud et al., 2015; Jain et al., 2015). The work in (Bruna et al., 2014) has been among the pioneering efforts in trying to bridge the gap between graph-based learning and deep learning methods. The authors calculate the projection of graph signals onto the space defined by the eigenvectors of the Laplacian matrix of the input graph, which itself describes the geometry of the data. It however requires an expensive calculation of the graph eigendecomposition, which can be a strong limitation for large graphs, as it requires $O(N^3)$ operations with N being the number of nodes in the graph. The authors in (Defferrard et al., 2016) later propose an alternative to analyse network data, which is built on a vertex domain feature representation and on fast spectral convolutional filters. Both methods directly integrate the graph features into a fully-connected layer similarly to classical ConvNets, which is however not directly amenable to transformation-invariant image classification.

To the best of our knowledge, the current approaches to deep learning on graphs do not provide transformation-invariance in image classification. At the same time, the methods that specifically target transformation invariance in image datasets mostly rely on data augmentation, which largely remains an art. We propose to bridge this gap and present a novel method that uses the power of graph signal processing to add translation and rotation invariance to the image feature representation learned by deep networks.

3. Graph signal processing elements

We now briefly review some elements of graph signal processing that are important in the construction of our novel framework. We represent an input image as a signal $y(v_n)$ on the nodes $\{v_n\}$ of the grid graph G. In more details, $G = \{\mathcal{V}, \mathcal{E}, A\}$ is an undirected, weighted and connected graph, where $\mathcal{V}$ is a set of N vertices (i.e., the image pixels), $\mathcal{E}$ is a set of edges and A is a weighted adjacency matrix. An edge $e(v_i, v_j)$ that connects two nodes v_i and v_j is associated with the weight $a_{ij} = a_{ji}$, which is usually chosen to capture the distance between both vertices. The edge weight is set to zero for pairs of nodes that are not connected, and all the edge weights together build the adjacency matrix A. Every vertex v_n of G carries the luminance value of the corresponding image pixel. Altogether, the valued vertices define a graph signal $y(v_n) : \mathcal{V} \to \mathbb{R}$.

Similarly to regular 1-D or 2-D signals, the graph signals can be efficiently analysed via harmonic analysis and processed in the spectral domain (Shuman et al., 2013). In that respect, we first consider the normalized graph Laplacian

operator of the graph G, defined as

$$\mathcal{L} = I - D^{-1/2} A D^{-1/2},$$

where D is a diagonal degree matrix with elements $d_i = \sum_{n=0,n\neq i}^{N} A_{ni}$. The Laplacian operator is a real symmetric and positive semidefinite matrix, which has a set of orthonormal eigenvectors and corresponding eigenvalues. Let $\chi = [\chi_0, \chi_1, \ldots, \chi_{N-1}]$ denote these eigenvectors and $\{0 = \lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{N-1}\}$ denote the corresponding eigenvalues with $\lambda_{N-1} = \lambda_{\max} \leq 2$ for the normalized Laplacian $\mathcal{L}$. The eigenvectors form a Fourier basis and the eigenvalues carry a notion of frequencies as in the classical Fourier analysis. The Graph Fourier Transform $\hat{y}(\lambda_i)$ at frequency λ_i for signal y and respectively the inverse graph Fourier transform for the vertex $v_n \in \mathcal{V}$ are thus defined as:

$$\hat{y}(\lambda_i) = \sum_{n=1}^{N} y(v_n) \chi_i^*(v_n), \tag{1}$$

and

$$y(v_n) = \sum_{i=0}^{N-1} \hat{y}(\lambda_i) \chi_i(v_n), \tag{2}$$

where the superscript $*$ denotes the complex conjugate. Equipped with the above notion of Graph Fourier Transform, we can denote the generalized convolution of two graph signals y_1 and y_2 with help of the graph Laplacian eigenvectors as

$$(y_1 * y_2)(v_n) = \sum_{i=0}^{N-1} \hat{y_1}(\lambda_i) \hat{y_2}(\lambda_i) \chi_i(v_n). \tag{3}$$

By comparing the previous relations, we can see that the convolution in the vertex domain is equivalent to the multiplication in the graph spectral domain. Graph spectral filtering can further be defined as

$$\hat{y}_f(\lambda_i) = \hat{y}(\lambda_i) \hat{h}(\lambda_i), \tag{4}$$

where $\hat{h}(\lambda_i)$ is the spectral representation of the graph filter $h(v_n)$ and $\hat{y}_f(\lambda_i)$ is the Graph Fourier Transform of the filtered signal y_f. In a matrix form, the graph filter can be denoted by $H \in \mathcal{R}^{N \times N} : H = \chi \hat{H} \chi^T$, where $\hat{H}$ is a diagonal matrix constructed on the spectral representation of the graph filter:

$$\hat{H} = \text{diag}(\hat{h}(\lambda_0), \ldots, \hat{h}(\lambda_{N-1})). \tag{5}$$

The graph filtering process becomes $y_f = Hy$, with the vectors y and y_f being the graph signal and its filtered version in the vertex domain. Finally, we can define the generalized translation operator T_{v_n} for a graph signal y as the convolution of y with a delta function δ_{v_n} centered at vertex v_n (Thanou et al., 2014):

$$\begin{aligned} T_{v_n} y &= \sqrt{N}(y * \delta_{v_n}) \\ &= \sqrt{N} \sum_{i=0}^{N-1} \hat{y}(\lambda_i) \chi_i^*(v_n) \chi_i. \end{aligned} \tag{6}$$

More details about the above graph signal processing operators can be found in (Shuman et al., 2013).

4. Graph-based convolutional network

We now present the overview of our new architecture, which is illustrated in Fig. 2. The input to our system can be characterized by a normalized Laplacian matrix $\mathcal{L}$ computed on the grid graph G and the signal $y_0 = (y_0(v_1), \ldots, y_0(v_N))$, where $y_0(v_j)$ is the intensity of the pixel j in the input image and N is the number of pixels in the images. Our network eventually returns a class label for each input signal.

In more details, our deep learning architecture consists of an alternation of spectral convolution $\mathcal{F}^l$ and dynamic pooling $\mathcal{P}^l$ layers. They are followed by a statistical layer $\mathcal{H}$ and a sequence of fully-connected layers that precedes a softmax operator that produces a categorical distribution over labels to classify the input data. Both the spectral convolution and the dynamic pooling layers contain K_l operators denoted by $\mathcal{F}_i^l$ and $\mathcal{P}_i^l$, $i = 1, \ldots, K_l$, respectively. Each $\mathcal{F}_i^l$ is specifically designed to compute transformation-invariant features on grid graphs. The dynamic pooling layer follows the same principles as the classical ConvNet's max-pooling operation but preserves the graph structure in the signal representation. Finally, the statistical layer $\mathcal{H}$ is a new layer designed specifically to achieve invariance to isometric transformations on grid graphs. It does not have any correspondent in the classical ConvNets architectures. We discuss more thoroughly each of these layers in the remainder of this section.

4.1. Spectral convolutional layer

Similarly to the convolutional layers in classical architectures, the spectral convolutional layer l in our network consists of K_l convolutional filters $\mathcal{F}_i^l$, as illustrated in Fig. 3. However, each filter i operates in the graph spectral domain. In order to avoid computing the graph eigendecomposition that is required to perform filtering through Eq. (3), we choose to design our graph filters as smooth polynomial filters of order M (Thanou et al., 2014):

$$\hat{h}(\lambda_l) = \sum_{m=0}^{M} \alpha_m \lambda_l^m. \tag{7}$$

Following the notation of Eq. (5), each filter operator in the spectral convolutional layer l can be written as

$$\mathcal{F}_i^l = \sum_{m=0}^{M} \alpha_{i,m}^l \mathcal{L}^m, \tag{8}$$

where $\mathcal{L}^m$ denotes the Laplacian matrix of power m. The polynomial coefficients $\{\alpha_{i,m}^l\}$ have to be learned during

the training of the network, for each spectral convolutional layer l. Each column of this $N \times N$ operator corresponds to an instance of the graph filter centered at a different vertex of the graph (Thanou et al., 2014). The support of each graph filter is directly controlled by the degree M of the polynomial kernel, as the filter takes values only on vertices that are less than M-hop away from the filter center. Larger values of M require more parameters but allow training more complex filters. Therefore, M can be seen as a counterpart of the filter's size in the classical ConvNets.

The filtering operation then simply consists in multiplying the graph signal by the transpose of the operator defined in Eq. (8), namely

$$\tilde{y}_{i,k}^l = \left[\mathcal{F}_i^l |_{\mathcal{N}_i^{l-1}} \right]^T y_k^l, \tag{9}$$

where y_k^l and $\tilde{y}_{i,k}^l$ are the kth graph signals at the input and respectively the output of the lth spectral convolutional layer (see Fig. 3). In particular, $y_k^{(1)} = y_0$ is the input image for the first level filter, while at the next levels of the network y_k^l is rather one of the feature maps output by the lower layers. We finally use the notation $A|_{\mathcal{N}_i^l}$ to represent an operator that preserves the columns of the matrix A, which have an index in the set $\mathcal{N}_i^l$, and set all the other columns to zero. This operator permits computing the filtering operations only on specific vertices of the graphs. It is important to note that the spectral graph convolutional filter permits equivariance to isometric transformations, which is a key property for designing a classifier that is invariant to rotation and translation.

Finally, the output of the lth spectral convolutional layer is a set of K_l feature maps z_i^l. Each ith feature map is computed as a linear combination of the outputs of the corresponding polynomial filter as follows:

$$z_i^l = \sum_{k=1}^{K_{l-1}} \beta_k^l \, \tilde{y}_{i,k}^l, \tag{10}$$

where the set of signals $\tilde{y}_{i,k}^l$ are the outputs of the ith polynomial filter applied on the K_{l-1} input signals of the spectral convolutional layer with Eq. (9). The vector of parameters $\{\beta_k^l\}$, for each spectral convolutional layer l is learned during the training of the network. The operations in the spectral convolutional layer are illustrated in Fig. 3. Lastly, the complexity of spectral filtering can be computed based on the fact that $\mathcal{L}$ and thus the filters are sparse matrices. Then, the complexity is $O(|\mathcal{E}_M|N)$ where $|\mathcal{E}_M|$ is a maximum number of nonzero elements in the columns of $\mathcal{F}_i^l$.

4.2. Dynamic pooling layer

In classical ConvNets the goal of pooling layers is to summarize the outputs of filters for each operator at the previous convolutional layer. Inspired by (Kalchbrenner et al.,
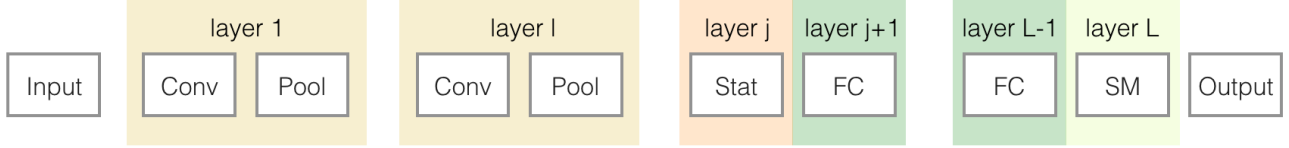
Figure 2. **TIGraNet** architecture. The network is composed of an alternation of spectral convolution layers $\mathcal{F}^l$ and dynamic pooling layers $\mathcal{P}^l$, followed by a statistical layer $\mathcal{H}$, multiple fully-connected layers (FC) and a softmax operator (SM). The input of the network is an image that is represented as a signal y_0 on the grid-graph with Laplacian matrix $\mathcal{L}$. The output of the system is a label that corresponds to the most likely class for the input sample.
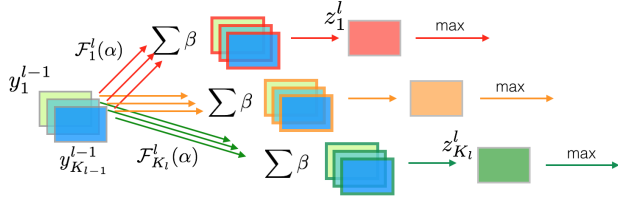

Figure 3. Spectral convolutional layer $\mathcal{F}^l$ in **TIGraNet**. The outputs of the previous layer $l-1$ are fed to a set of filter operators $\mathcal{F}_i^l$. The outputs of $\mathcal{F}_i^l$ are then linearly combined to get the filter maps z_i^l that are further passed to the dynamic pooling layer.

2014) we introduce a novel layer that we refer to as dynamic pooling layer, which consists in preserving only the most important features at each level of the network.

In more details, we perform a dynamic pooling operation, which is essentially driven by the set of graph vertices of interest Ω^l. This set is initialised to include all the nodes of graph, i.e., $\Omega^1 = \mathcal{V}$. It is then successively refined along the progression through the multiple layers of the network. More particularly, for each dynamic pooling layer l, we select the J_l vertices that are part of Ω^{l-1} and that have the highest values in z_i^l. The indexes of these largest valued vertices form a set of nodes $\mathcal{N}_i^l$. The union of these sets for the different features maps z_i^l form the new set Ω^l, i.e.,

$$\Omega^l = \bigcup_{i=1}^{K_l} \mathcal{N}_i^l. \qquad (11)$$

The sets Ω^l drives the pooling operations at the next dynamic pooling layer $\mathcal{P}^{l+1}$. We note that, by construction, the different sets Ω^l are embedded, namely we have $\Omega^l \supseteq \Omega^{l+1}$, $\forall l \in [1..L]$. Fig. 4 illustrates the effect of the pooling process through the different network levels.

The sets $\mathcal{N}_i^l$ are used to control the filtering process at the next layer. The spectral convolutional filters $\mathcal{F}_i^{l+1}$ compute the output of filters centred on the nodes in $\mathcal{N}_i^l$ that are selected by the dynamic pooling layer, and not necessarily for all the nodes in the graph. The filtering operation is given by Eq. (9).

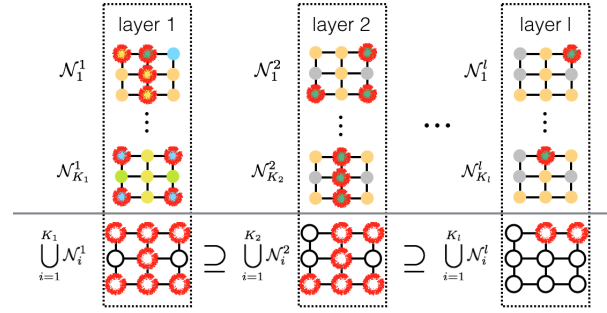Finally, we note that one of the major differences with the

Figure 4. Pooling process, with succession of dynamic pooling layers with operators $\mathcal{P}_i^l$ that each selects the vertices with maximum intensity according to Eq. (11).

classical max-pooling operator is that our dynamic pooling layer is not limited to a small neighbourhood around each node. Instead, it considers the set of nodes of interest Ω^l which is selected over all graph's nodes. The dynamic pooling operator $\mathcal{P}^l$ is thus equivariant to the isometric transformations R, similarly to the spectral convolutional layers, which is a key property in building a transformation-invariant classification architecture. The complexity of $\mathcal{P}^l$ is comparable with the classical pooling operator as the task of $\mathcal{P}^l$ is equivalent to finding J_l highest statistics. Using the selection algorithm (Knuth, 1998) we can reach the average computational complexity of $O(N)$.

4.3. Upper layers

After the series of alternating spectral convolutional and dynamic pooling layers, we add output layers that compute the label probability distributions for the input images. Instead of connecting directly a fully-connected layer as in ConvNet architectures, we first insert a new statistical layer, whose output is fed to fully-connected layers (see Fig. 2).

The main motivation for the statistical layer resides in our objective of designing a transformation-invariant classification architecture. If fully-connected layers are added directly on top of the last dynamic pooling layers, their neurons would have to memorize large amounts of information corresponding to the different positions and rotation of the visual objects. Instead, we propose to insert a new statis-

tical layer, which computes transformation-invariant statistics of the input signal distributions.

In more details, the statistical layer estimates the distribution of values on the active nodes after the last pooling layer. The inputs of the statistical layer j are denoted as $\tilde{z}_i$, which correspond to the outputs z_i^{j-1} of the last pooling layer $\mathcal{P}^{j-1}$ where the values on non-active nodes (i.e., the nodes in $\overline{\mathcal{N}_i^l}$) are set to zero. We then calculate multiscale statistics of these input features maps using Chebyshev polynomials of the graph Laplacian. These polynomials have two important benefits. First, they permit to capture the relation between different neighbors of the graph nodes. In particular, polynomials of order 1 consider direct neighbors, polynomials of order 2 include second neighbors and so on. Gathering all this information permits to efficiently discriminate different objects. As the statistics are calculated from the entire feature map, the final feature representation is independent of the actual signal transformation. Secondly, Chebyshev polynomials can be calculated in an iterative manner, which make them readily amenable to effective computation. (Shuman et al., 2011). In order to construct these polynomials, we first shift the spectrum of the Laplacian $\mathcal{L}$ to the interval $[-1, 1]$, which is the original support of Chebyshev polynomials. Equivalently, we set $\tilde{\mathcal{L}} = \mathcal{L} - I$.

As suggested in (Defferrard et al., 2016), for each input feature map $\tilde{z}_i$ we iteratively construct a set of signals $t_{i,k}$ using Chebyshev polynomials of order k, with $k \le K_{max}$:

$$t_{i,k} = 2\tilde{\mathcal{L}}t_{i,k-1} - t_{i,k-2}, \tag{12}$$

with $t_{i,0} = \tilde{z}_i$ and $t_{i,1} = \tilde{\mathcal{L}}\tilde{z}_i$. We finally compute a feature vector that gathers the first order statistics of the magnitude of these signals, namely the mean $\mu_{i,k}$ and variance $\sigma_{i,k}^2$ for each signal $|t_{i,k}|$. This forms a feature vector ϕ_i of $2K_{max} + 2$ elements, i.e., $\phi_i = [\mu_{i,0}, \sigma_{i,0}^2, \ldots, \mu_{i,K_{max}}, \sigma_{i,K_{max}}^2]$. We choose these particular statistics as they are prone to efficient gradient computation, which is important during back propagation. Furthermore, we note that such feature vectors are inherently invariant to transformation such as translation or rotation.

The feature vectors ϕ_i's are eventually sent to a series of fully-connected layers similarly to classical ConvNet architectures. However, since our feature vectors are transformation invariant, the fully-connected layers will also benefit from these properties. This is in opposition to their counterparts in classical ConvNet systems, which need to compute position-dependent parameters. The details about fully-connected layer parameters are given in the Section 5. The output of the fully-connected layers is then fed to a softmax layer (Bishop, 2006), which finally returns the probability distribution of a given input sample to belong to a given set of classes.

5. Experiments

In this section we compare our network to the state-of-the-art transformation-invariant classification algorithms.

5.1. Experimental settings

We run experiments with different numbers of layers and parameters. For each architecture, the network is trained using back-propagation with Adam (Kingma & Ba, 2014) optimization. The exact formulas of the partial derivatives and explanation about the initialization of the network parameters are provided in the supplementary material. Our architecture has been trained and tested on different datasets, namely:

- **MNIST-012.** This is a small subset of the MNIST dataset (LeCun & Cortes, 2010). It includes 500 training, 100 validation and 100 test images selected randomly from the MNIST images with labels '0', '1' and '2'. This small dataset permits studying the behavior of our network in detail and to analyze the influence of each of the layers on the performance.

- **Rotated and translated MNIST.** To test the invariance to rotation and translation of the objects in an image we create MNIST-rot and MNIST-trans datasets respectively. Both of these datasets contain 50k training, 3k validation and ~9k test images. We use all MNIST digits (LeCun & Cortes, 2010) except '9' as it is rotated version resembles '6'. In order to be able to apply transformation to the digits, we resize the MNIST-rot to the size 26×26 and MNIST-trans to the 34×34. The training and validation data of these datasets contain images of digits without any transformation. However, the testing set of MNIST-rot contains randomly rotated digits by angles in range $[0°, 360°]$, while the testing set of MNIST-trans comprises randomly translated MNIST examples up to ± 6 pixels in both vertical and horizontal directions.

- **ETH-80.** The dataset (Leibe & Schiele, 2003) contains images of 80 objects that belong to 8 classes. Each object is represented by 41 images captured from different viewpoints located on a hemisphere. The dataset shows a real life example where isometric transformation invariant features are useful for the object classification. We resize the images to $[50 \times 50]$ and randomly select 2300, 300 of them as the training, validation sets and we use the rest of them for testing.

For all these datasets, we define G as a grid graph where each node corresponds to a pixel location and is connected with 8 its nearest neighbors with a weight that is equal to 1. The pixel luminance values finally define the signal y on the graph G for each image.

Method	Architecture
Experiments on MNIST-012	
ConvNet (Boureau et al., 2010)	C[3]-P[2]-C[6]-P[2]-FC[50]-FC[30]-FC[10]
STN (Jaderberg et al., 2015)	C[3]-ST[6]-C[6]-ST[6]-FC[50]-FC[30]-FC[10]
TIGraNet	SC[3, 3]-DP[300]-SC[6, 3]-DP[100]-S[10]-FC[50]-FC[30]-FC[10]
Other experiments	
ConvNet (Boureau et al., 2010)	C[10]-P[2]-C[20]-P[2]-FC[500]-FC[300]-FC[100]
STN (Jaderberg et al., 2015)	C[10]-ST[6]-C[20]-ST[6]-FC[500]-FC[300]-FC[100]
DeepScat (Oyallon & Mallat, 2015)	W[2, 5]-PCA[20]
HarmNet (Worrall et al., 2016)	HRC[1, 10]-HCN[10]-HRC[10, 10]-HRC[10, 20]-HCN[20]-HRC[20, 20]
TIGraNet	SC[10, 4]-DP[600]-SC[20, 4]-DP[300]-S[12]-FC[500]-FC[300]-FC[100]

Table 1. Architectures used for the experiments on (Leibe & Schiele, 2003). We use the following notations to describe the architectures of the networks. $C[X_1]$, $P[X_2]$, $FC[X_3]$ correspond to the convolutional, pooling and fully-connected layers respectively, with X_1 being the number of 3×3 filters, X_2 – the size of the max-pooling area and X_3 – the number of hidden units. $ST[X_4]$ denotes the spatial transform layer with X_4 affine transformation parameters. $W[O, J]$ and $PCA[X_5]$ denote the parameters of DeepScat network with wavelet-based filters of order O and maximum scales J, with dimension of the affine PCA classifier X_5. $HRC[X_6, X_7]$ depicts the harmonic cross correlation filter operating on the X_7 neighborhood with X_6 feature maps. $HCN[X_8]$ is the complex nonlinearity layer of HarmNet with X_8 parameters. Finally, $SC[K_l, M]$ is a spectral convolutional layer with K_l filters of degree M, $DP[J_l]$ is a dynamic pooling that retains J_l most important values. $S[K_{max}]$ is a statistical layer with K_{max} the maximum order of Chebyshev polynomials.

5.2. Performance evaluation

Here, we compare TIGraNet to state-of-the art algorithms for transformation-invariant image classification tasks, i.e., ConvNet (Boureau et al., 2010), Spatial Transformer Network (STN) (Jaderberg et al., 2015), Deep Scattering (DeepScat) (Oyallon & Mallat, 2015) and Harmonic Networks (HarmNet) (Worrall et al., 2016). Briefly, ConvNet is a classical convolutional deep network that is invariant to small image translations. STN compensates for image transformations by learning the affine transformation matrix. Further, DeepScat uses filters based on rich wavelet representation to achieve transformation invariance, however, it does not contain any parameters for the convolutional layers. Finally, HarmNet trains complex valued filters that are equivariant to signal rotations. For the sake of fairness in our comparisons, we use versions of these architectures that have roughly the same number of parameters, which means that each of the approaches learns features with a comparable complexity. For the DeepScat we use the default architecture. Further for the HarmNet we preserve the default network structure, keeping the same number of complex harmonic filters, as the number of spectral convolutional filters that we have in TIGraNet.

We first compare the performance of our algorithm to the ones of ConvNet and STN for the small digit dataset MNIST-012. The specific architectures used in this experiments are given in Table 1. The results of this first experiment are presented in Table. 2. We can see that if we train the methods on the dataset that does not contain rotated images and test on the rotated images of digits, our approach achieves a significant increase in performance (i.e., 86%),

	Training set	Validation set	Rotated test set
Training set with data augmentation			
ConvNet	99	94	78 ± 2.1
STN	100	97	93 ± 0.97
Training set without data augmentation			
ConvNet	100	100	55 ± 5
STN	100	98	50 ± 5
TIGraNet	98	97	$\mathbf{94 \pm 0.42}$

Table 2. Classification accuracy of ConvNet, STN and TIGraNet on MNIST-012 averaged over 10 runs.

	MNIST-rot	MNIST-trans
ConvNet	44.3	43.5
STN	44.5	67.1
TIGraNet	**83.8**	**79.6**

Table 3. Evaluation of the accuracy of the ConvNet, STN and TIGraNet on the MNIST-rot and MNIST-trans datasets. All the methods are trained on sets without transformed images.

due to its inherent transformation invariant characteristics. We further run experiments where a simple augmentation of the training set is implemented with randomly rotated each image of digits. This permits increasing the performance of all algorithms, as expected, possibly at the price of more complex training. Still, due to the rotation invariant nature of its features, TIGraNet is still able to achieve higher classification accuracy than all its competitors.

We then run experiments on the MNIST-rot and MNIST-trans datasets. Note that both of them do not contain any
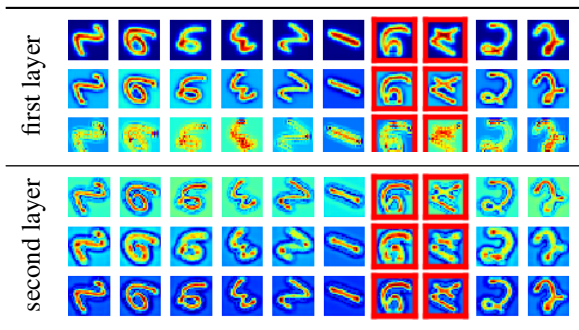
Figure 5. **Network feature maps visualization.** Each row shows the feature maps of different digits after the first and the second spectral convolutional layers. The misclassified images are marked by red bounding boxes. (best seen in color)

STN (Jaderberg et al., 2015)	45.1
ConvNet (Boureau et al., 2010)	80.1
DeepScat (Oyallon & Mallat, 2015)	87.3
HarmNet (Worrall et al., 2016)	94.0
TIGraNet	**95.1**

Table 4. Classification accuracy of ConvNet, STN, DeepScat, HarmNet and TIGraNet on the ETH-80 dataset.

isometric transformation in training and validation sets, but the test set contains transformed images. For all the methods we have used the architectures defined in Table 1. Table 3 shows that our algorithm significantly outperforms the competitor methods on both datasets due to its transformation invariant features. The other architectures have only limited capabilities with respect to such transformation as rotation. Additionally, we run experiments for our MNIST-rot with different types of data augmentation. We have tried several settings, when input image is rotated by an angle that is a multiple of X degrees, with X being 360, 90, 60, 30 and 15. As expected, the accuracy of ConvNets increases to $44.3\%, 67.2\%, 76.7\%, 79.4\%, 80.1\%$ respectively with the increasing amount of data augmentation. We observe similar dynamics for STN method and its accuracy matches our approach for $X = 15$ degrees rotations.

To further analyze the performance of our network we illustrate several sample feature maps for the different filters of the first two spectral convolutional layers of TIGraNet in Fig. 5, for the MNIST-rot and MNIST-trans datasets. We can see a few examples of misclassification of our network; for example, the algorithm predicts label '5' for the digit '6'. This mostly happens due to the border artifacts: if an isometric transformation shifts the digit too close to the border, the neighborhood of some nodes may change. This problem can be solved by increasing the image borders or applying filters only to the central pixel locations.

Finally, we evaluate the performance of our algorithm in

more realistic settings where the objective is to classify images of objects that are captured from different viewpoints. This task requires having a classifier that is invariant to isometric transformations of the input signal. We therefore run experiments on the ETH-80 dataset and compare the classification performance of TIGraNet to those of ConvNet, STN, DeepScat and HarmNet. The architectures of the different methods are described in Table 1.

Table 4 shows the classification results in this experiment. We can see that our approach outperforms the state-of-the-art methods due to its transformation invariant features. The closest performance is achieved by Harmonic Networks, since this architecture also learns equivariant features. It is important to note that the ETH-80 dataset contains less training examples than other publicly available datasets that are commonly used for training of deep neural networks. This likely results in decrease of accuracy for such methods as ConvNet and STN. On the contrary, our method is able to achieve good accuracy even with small amounts of training data, due to its inherent invariance to isometric transformations.

Overall, all the above experiments confirm the benefit of our transformation invariant classification architecture, which learns features that are invariant to transformation by construction. Classification performance improves with these features, such that the algorithm reaches sustained performance even if the training set is relatively small, or does not contain similar transformed images as the test set. These are very important advantages in practice.

6. Conclusion

In this paper we present a new transformation invariant classification architecture, which combines the power of deep networks and graph signal processing, which allows developing filters that are equivariant to translation and rotation. A novel statistical layer further renders our full network invariant to the isometric transformations. This permits outperforming state-of-the-art algorithms on various illustrative benchmarks. Having inherently invariant to transformation features gives our network the ability to learn well from a few training examples and to generalize to unseen transformations. This confirms its high potential in practical settings where the training sets are limited but where the data is expected to present high variability.

Acknowledgments

References

Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

Boureau, Y. L., Ponce, J., and LeCun, Y. A Theoretical Analysis of Feature Pooling in Visual Recognition. In *International Conference on Machine Learning*, 2010.

Bruna, J. and Mallat, S. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1872–1886, 2013.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral Networks and Locally Connected Networks on Graphs. In *International Conference for Learning Representations*, 2014.

Cohen, T. S. and Welling, M. Group equivariant convolutional networks. *arXiv preprint*, 2016.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems*, pp. 3837–3845, 2016.

Dieleman, S., Willett, K. W., and Dambre, J. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notices of the royal astronomical society*, 450(2):1441–1459, 2015.

Dieleman, S., Fauw, J. D., and Kavukcuoglu, K. Exploiting cyclic symmetry in convolutional neural networks. In *International Conference on Machine Learning*, 2016.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pp. 2224–2232, 2015.

Dyk, D. A. and Meng, X.-L. The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10: 1–111, 2012.

Fasel, B. and Gatica-Perez, D. Rotation-invariant neoperceptron. In *International Conference on Pattern Recognition*, volume 3, pp. 336–339, 2006.

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

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. Spatial Transformer Networks. *arXiv Preprint*, 2015.

Jain, A., Zamir, A. R., Savarese, S., and Saxena, A. Structural-rnn: Deep learning on spatio-temporal graphs. *arXiv Preprint*, 2015.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. A Convolutional Neural Network for Modelling Sentences. *arXiv Preprint*, 2014.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *arXiv Preprint*, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint*, 2016.

Knuth, D. E. *The Art of Computer Programming: Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., 1998.

Krizhevsky, A., Sutskever, I., and Geoffrey, E. H. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

Laptev, D., Savinov, N., Buhmann, J.M., and Pollefeys, M. TI-Pooling: Transformation-Invariant Pooling for Feature Learning in Convolutional Neural Networks. In *Conference on Computer Vision and Pattern Recognition*, 2016.

LeCun, Y. and Cortes, C. MNIST handwritten digit database, 2010. URL http://yann.lecun.com/exdb/mnist/.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-Based Learning Applied to Document Recognition. In *Intelligent Signal Processing*, pp. 306–351, 2001.

Leibe, B. and Schiele, B. Analyzing appearance and contour based methods for object categorization. In *Conference on Computer Vision and Pattern Recognition*, volume 2, pp. 11–409, 2003.

Marcos, D., Volpi, M., and Tuia, D. Learning rotation invariant convolutional filters for texture classification. *arXiv preprint*, 2016.

Mathieu, M., Henaff, M., and Lecun, Y. Fast training of convolutional networks through ffts. In *International Conference on Learning Representations*. 2014.

Oyallon, E. and Mallat, S. Deep roto-translation scattering for object classification. In *Conference on Computer Vision and Pattern Recognition*, pp. 2865–2873, 2015.

Rippel, O., Snoek, J., and Adams, R. P. Spectral representations for convolutional neural networks. In *Advances in Neural Information Processing Systems 28*, pp. 2449–2457. Curran Associates, Inc., 2015.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, volume 9351, pp. 234–241, 2015.

Shuman, D. I., Vandergheynst, P., and Frossard, P. Chebyshev polynomial approximation for distributed signal processing. In *IEEE International Conference on Distributed Computing in Sensor Systems*, pp. 1–8, 2011.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. 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, 2013.

Thanou, D., Shuman, D. I., and Frossard, P. Learning parametric dictionaries for signals on graphs. *IEEE Transactions on Signal Processing*, 62(15):3849–3862, 2014.

Worrall, D. E., Garbin, S. J., Turmukhambetov, D., and Brostow, G. J. Harmonic networks: Deep translation and rotation equivariance. *arXiv Preprint*, 2016.