Graph-based image representation learning

Thèse N° 9267

Présentée le 27 juin 2019

à la Faculté des sciences et techniques de l'ingénieur
Laboratoire de traitement des signaux 4
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Renata KHASANOVA

Acceptée sur proposition du jury

Dr F. Fleuret, président du jury
Prof. P. Frossard, directeur de thèse
Dr T. Maugey, rapporteur
Prof. Y. Bastanlar, rapporteur
Prof. A. Alahi, rapporteur

2019

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

There is no path to happiness:
happiness is the path.
— T. N. Hanh

To my family…

Acknowledgements

I would like to first thank my PhD advisor, Prof. Pascal Frossard, for giving me the great opportunity to join his research group. I am very grateful for him for giving me the freedom to choose my topic, which I enjoyed to work on, for his help and for providing detailed feedbacks during my PhD journey. I would also like to thank Xiaowan for working with me at the beginning of my PhD and for his patience and helpfulness during this time.

Further, I would like to thank the members of my thesis committee, Prof. Alexandre Alahi, Prof. Yalin Baştanlar, Dr. Thomas Maugey and Prof. François Fleuret for the time they have taken to review my manuscript and for their helpful comments.

I would like to thank my colleagues and collaborators from Google Research Zürich: Jyrki Alakuijala, Jan Wassenberg and Zoltan Szabadka, as well as all the others, who I worked with. I really enjoyed my time there and I am very grateful for the help and support that they provided.

Further, I would also like to give thanks to all the present and past members of LTS4 laborotary: Beril, Eda, Hermina, Seyed (Mohsen), Stefano, Mattia, Ersi, Isabela, Apostolos, Guillermo, Clement, Mireille, Roberto, Bastien, Guiseppe, Francesca, Xing, Hussein, Dorina, Andreas, Sofia, Ana, Xiaowan, Laura, Thomas, Pinar, Arun for the great time I have spent in the lab, for our fun game nights and other activities. In particular, I would like to thank my officemate Mireille for making our office fun and warm, for being always supportive and positive. I would also like to thank all the members of our baby-foot team (Stefano, Roberto, Clement, Seyed, Mireille, Arun, Hermina, Beril, Mattia) for our sportive breaks. I thank Eda and Ersi for sharing yoga and ballet passions with me. I would also like to give special thanks to Rosie and Anne for their kindness and assistance.

Then I would like to give great thanks to all my Lausanne friends. It would not be the same without you. In particular, I would like to thank Marwa, Rafah, Ajay, Elie and Elena for all the great adventures, hiking, traveling and ski weekends that we did together. I would like to thank also my Russian friends from Zürich for motivating me to come to Switzerland and welcoming me here. Especially, I would like to thank Valja and Dima for being always there for me, for inspiring me and for all the fun moments that we had together.

I am also very thankful to my husband Artem, for not only being my best friend during this PhD journey, bringing love, calmness and happiness in my life, being excited about all my ideas and believing in me, but also for deciding to marry me during my thesis writing time. I also greatly thank to my grandparents, aunts Raya and Elena, uncle Airat for being always supportive. I would also like to thank my new family members: Tatiana, Viktor, Kostya, Lena, Vladimir and Svetlana for sharing amazing time together and for your support during all the time. Finally, I would like to give a special thanks to my parents and my sister Adelina for their unconditional love and support in all times, for their kindness, help, for believing in me and making me believe in myself. Without them this PhD will be absolutely impossible.

Zürich, 21 January 2019

Abstract

Though deep learning (DL) algorithms are very powerful for image processing tasks, they generally require a lot of data to reach their full potential. Furthermore, there is no straightforward way to impose various properties, given by the prior knowledge about the target task, on the features extracted by a DL model. Therefore, in this thesis we propose several techniques that rely on the power of graph representations to embed prior knowledge inside the learning process. This allows to reduce the solution space and leads to faster optimization convergence and higher accuracy in the representation learning.

In our first work, inspired by the ability of a human to correctly classify rotated, shifted or flipped objects, we propose an algorithm that permits to inherently encode invariance to isometric transformations of objects in an image. Our DL architecture is based on graph representations and consists of three novel layers, which we refer to as graph convolutional, dynamic pooling and statistical layers. Our experiments on the image classification tasks show that our network correctly recognizes isometrically transformed objects even though such types of transformation are not seen by the network at training time. Standard DL techniques are typically not able to succeed in solving such a problem without extensive data augmentation.

Then, we propose to exploit the properties of graph-based approaches to efficiently process images with various types of projective geometry. In particular, we are interested in increasingly popular omnidirectional cameras, which have a 360 degree field of view. Despite their effectiveness, such cameras create images with specific geometric properties, which require special techniques for efficient processing. We propose an efficient way of adjusting the weights of the graph edges to adapt the filter responses to the geometric image properties introduced by omnidirectional cameras. Our experiments prove that using the proposed graph with properly adjusted edge weights permits to reach better performance as compared to using regular grid graph with equal weights.

Finally, the approach described above relies on the isotropic filters, which work well within our transformation invariant architecture for image classification. However, for other problems (e.g. image compression) or even when used without dynamic pooling and statistical layers that are defined within the proposed architecture, these filters are unable to efficiently encode the information about the object. Thus, we introduce a different technique based on anisotropic filters that adapt their shape and size according to the omnidirectional image geometry. The main advantage of this approach compared to the previous one is the ability to encode the orientation of an image pattern, which is important for various tasks such as image compression. Our experiments show that our approach adapts to different image projective geometries and achieves state-of-the-art performance on image classification and compression tasks.

Overall we propose several methods, which combine the power of DL and graph signal processing towards incorporating prior information about the target task inside the optimization procedure. We hope that the research efforts presented in this thesis will help the development of efficient DL algorithms that can use various types of prior knowledge to make them efficient even when the available training data is scarce, such as medical imaging and omnidirectional camera processing applications.

Keywords: deep learning, graph signal processing, invariant feature representation, omnidirectional images, geometry-aware architectures.

Résumé

Malgré la puissance des algorithmes d'apprentissage profond (DL), le grand volume des données reste un défi pour pouvoir atteindre une performance maximale. De plus, il n'existe aucun moyen simple pour imposer des informations qu'on connaît a priori sur la tâche cible et sur les caractéristiques extraites par un modèle DL. Cette thèse apporte une contribution fondamentale pour modéliser et intégrer des connaissances a priori dans le processus d'apprentissage, en se basant sur des méthodes graphiques. Les méthodes employées présentent l'avantage de pouvoir réduire l'espace de la solution, d'augmenter la vitesse de convergence et d'avoir une plus grande précision dans l'apprentissage de la représentation.

L'idée principale s'inspire de la capacité d'un être humain à classifier correctement des objets pivotés, décalés ou retournés. Dans ce travail, nous proposons un algorithme offrant une grande flexibilité quant aux transformations isométriques des objets dans une image. Notre architecture DL est basée sur des représentations de graphes et se compose de trois nouvelles couches : une couche de convolution sur les graphes, une couche de pooling (« mise en commun ») dynamique et une couche statistique. Dans les résultats de simulation fournis, nous démontrons l'efficacité de notre approche pour la classification des images. Le réseau considéré offre une grande capacité de reconnaître correctement les objets transformés de manière isométrique, même si ces transformations ne sont pas vies par le réseau au moment de l'apprentissage. Les techniques DL classiques sont généralement incapables de résoudre un tel problème sans recours à une augmentation importante des données.

Dans une seconde partie, nous proposons d'exploiter les propriétés des approches basées sur les graphes pour traiter efficacement des images avec divers types de géométrie projective. En particulier, nous nous intéressons aux caméras omnidirectionnelles qui sont de plus en plus populaires pour leur champ de vision de 360 degrés. Malgré leur efficacité, ces caméras créent des images avec des propriétés géométriques spécifiques. Nous proposons un moyen efficace d'ajuster les poids des bords du graphe pour adapter les réponses du filtre aux propriétés géométriques de l'image introduites par les caméras omnidirectionnelles. Nos expériences prouvent que l'utilisation des graphes avec de poids adaptés pour les bords et correctement ajustés„ permet d'atteindre des meilleures performances que l'état de l'art, où les méthodes existantes se limitent à l'utilisation d'un graphe régulier avec des poids uniformes.

L'approche décrite ci-dessus repose sur des filtres isotropes, qui sont efficuces avec notre architecture invariante pour la classification des images qui contiennent des objets transformés. Cependant, pour d'autres types de problèmes (par exemple la compression d'image), ces filtres ne peuvent pas coder de manière efficace les informations relatives à l'objet. Dans

cette dernière partie, nous introduisons une technique différente basée sur des filtres anisotropes qui s'adaptent (forme et taille) à la géométrie de l'image omnidirectionnelle. L'avantage principal de cette approche par rapport à la précédente réside dans la possibilité de coder l'orientation d'un motif dans une image, ce qui est important pour diverses applications comme la compression d'image. Nos expériences montrent que notre approche s'adapte à différents types de géométries projectives. Nos résultats ont permis de valider l'efficacité de cette méthode pour la classification et la compression d'images.
Globalement, nous proposons plusieurs méthodes, qui combinent la puissance du traitement du signal DL et du graphe pour incorporer des informations données a priori sur la tâche cible dans la procédure d'optimisation. Nous espérons que les efforts de recherche présentés dans cette thèse contribuent aux développements des algorithmes DL et permettent d'ouvrir des possibilités pour l'incorporation de nouveaux modèles pour éviter le problème de manque des données dans des applications telles que l'imagerie médicale et le traitement des images omnidirectionnelles.

Mots clés : apprentissage profond, traitement du signal sur les graphes, représentation des entités invariantes, images omnidirectionnelles, modèles géométriques.

Contents

Acknowledgements **v**

Abstract **vii**

List of figures **xiv**

List of tables **xv**

1 Introduction **1**

1.1 Motivation 1

1.2 Thesis outline 4

1.3 Summary of contributions 7

2 Graph signal processing meets deep learning in vision **9**

2.1 Graph signal processing elements 9

2.2 Graph-based ConvNets 11

3 Isometric transformation invariant image representation learning **17**

3.1 Introduction 17

3.2 Transformation invariant techniques 18

3.3 Graph-based convolutional network 19

3.3.1 Spectral convolutional layer 20

3.3.2 Dynamic pooling layer 22

3.3.3 Upper layers 23

3.3.4 Training 25

3.4 Equivariance of graph filters 26

3.4.1 Graph isometric transformation 26

3.4.2 General isometric transformations 29

3.5 Experiments 39

3.5.1 Experimental settings 39

3.5.2 TIGraNet Analysis 41

3.5.3 Influence of the image resolution on the quasi-equivariance 43

3.5.4 Performance evaluation 45

3.6 Conclusion 48

4 Graph-Based Classification of Omnidirectional Images **51**
4.1 Introduction 51
4.2 Wide field of view image processing 52
4.3 Graph-based representation 54
4.3.1 Image model 54
4.3.2 Weight design 55
4.4 Experiments 60
4.5 Conclusion 64

5 Projective geometry-aware anisotropic convolutional filters **65**
5.1 Introduction 65
5.2 Omnidirectional image processing 66
5.3 Geometry-aware CNN 67
5.4 Results 70
5.4.1 Image Classification 70
5.4.2 Image Compression 73
5.5 Conclusion 75

6 Conclusion **79**
6.1 Achievements 79
6.2 Future directions 80

A An appendix **83**
A.1 Modified spherical surface projection 83

Bibliography **92**

Curriculum Vitae **93**

List of Figures

1.1 Sample convolutional operation 2
1.2 Sample max-pooling operation 2
1.3 Sample omnidirectional image projeciton 3

3.1 Illustrative transformation-invariant handwritten digit classification task 18
3.2 TIGraNet architecture 20
3.3 Spectral convolutional layer 20
3.4 Dynamic pooling layer 22
3.5 Illustration of the statistical layer 24
3.6 Rotation and translation of signal pattern 32
3.7 Sample images from ETH-80 dataset 41
3.8 Illustration of feature maps 42
3.9 Sample trained filters in the spectral domain 43
3.10 Feature maps after pooling 43
3.11 Influence of the image resolution on the quasi-equivariance property 44
3.12 Network feature maps visualization 47
3.13 Performance of the different approaches depending on the number of random rotations 49

4.1 Filter response of the proposed graph construction method 52
4.2 Example of the gnomonic projection 55
4.3 Example of the equirectangular representation of the image 56
4.4 Visualisation of pattern shape changes on the spherical surface 57
4.5 Example of the feature maps of the last spectral convolutional layer extracted from equirectangular images 61
4.6 Example of feature maps of equirectangular images from **ETH-80** datasets . . . 62
4.7 Illustration of the Euclidean distances between the resulting features from two appoaches 63

5.1 Sample original and decompressed images encoded with graph-based isotropic filters 65
5.2 Filter size and shape changes depending on the elevation level 66
5.3 Illustration of 3 × 3 GA-filter kernel, defined on the tangent plane 69
5.4 Surfaces with different geometries and respective equirectangular images . . . 71

5.5 Cube-map projection 72
5.6 Architecture of classification algorithm 72
5.7 Architecture of compression algorithm 74
5.8 Omnidirectinal image compression results 75
5.9 Cube-map projection's artifacts 76
5.10 Decompression result 77

List of Tables

3.1 Architectures used for the experiments. 40

3.2 Classification accuracy of ConvNet, STN and TIGraNet on MNIST-012. The methods are trained without and with transformed images. We average the performance of all the methods across 10 runs with different transformations of the test data. 46

3.3 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. 46

3.4 Performance evaluation for ConvNet, STN, DeepScat, HarmNet and TIGraNet on classification of images from the ETH-80 dataset. 46

4.1 Comparison to the state-of-the-art methods on the **ETH-80** datasets. We select the architecture of different methods to feature similar number of convolutional filters and neurons in the fully-connected layers. 62

5.1 Evaluation of different approaches on Spherical (S), Mod-Spherical (MS1, MS2, MS2), Fish-eye (F) and Cube-Map (CM) datasets. 73

1 Introduction

1.1 Motivation

Deep learning methods have become widely used for image processing applications in the recent years. They have achieved state-of-the-art performance in various computer vision tasks including but not limited to image classification [1], object detection and image segmentation [2]. Despite being very effective, deep learning methods typically require large amounts of data to achieve their full potential. Large amounts of training examples is generally beneficial for various machine learning techniques, but deep learning methods more critically depend on the data as they typically do not rely on any prior knowledge about the data or the task at hand. Further, even though in many of the real-world problems one has access to some prior knowledge (e.g., camera parameters etc.), there is no straight-forward way of incorporating such information inside deep neural networks.

In this thesis, we propose several methods of incorporating the prior knowledge about the available data or target task, into the learning process. This permits to reduce the solution search space and, thus to significantly reduce the size of the training set (or the necessary amount of data augmentation) and leads to important benefits compared to the approaches that have to learn everything purely based on the data. In this work we focus in particular on two types of prior information: isometric transformations of data and camera projective geometry. In order to incorporate these types of prior information in the deep learning procedure we rely on the power of graph-based representations. In the remainder of this section we discuss in more details the prior knowledge considered in this work and then briefly introduce our framework to incorporate these types of information in the deep neural network architecture.

Isometric transformation of objects. One of the types of prior information, which we consider in this work, is the invariance of classification tasks to isometric transformations of objects. This property plays an important role in vision, where the outcome of analysis or classification algorithm shall be independent of the orientation of the object in the image.

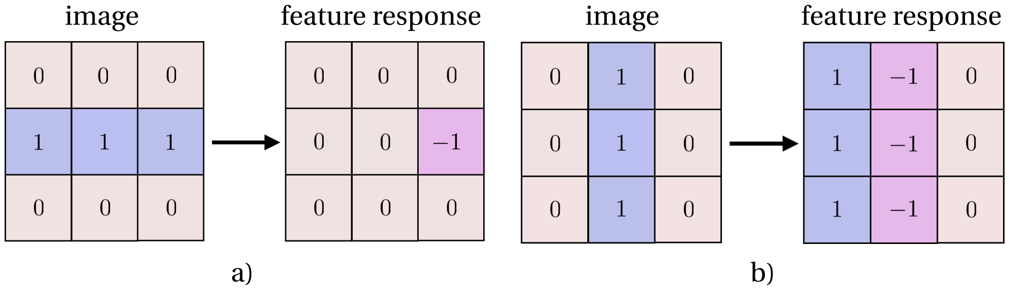


Figure 1.1 – Illustration of the result of convolutional operation with elementary filter $[-1, 1]$ on the images with a pattern in horizontal (a) or vertical (b) orientation. The resulting feature representation is not invariant to rotation.

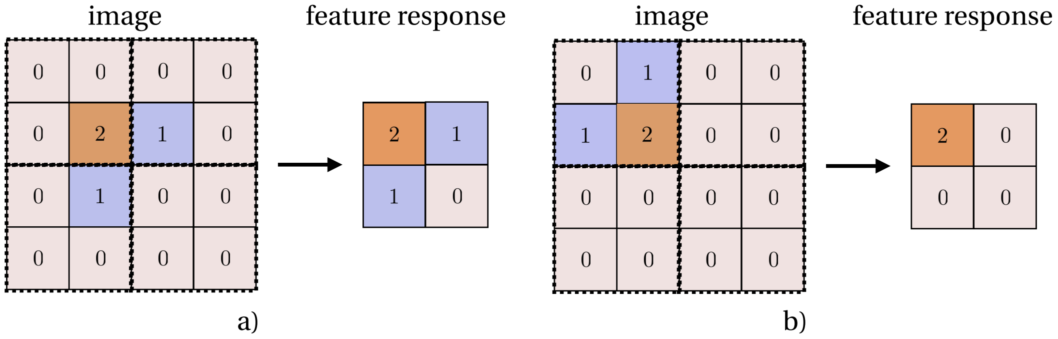


Figure 1.2 – Illustration of the result of max-pooling operation with non-overlapping patches, which is widely use in standard ConvNets on the image (a) and the image with the pattern rotated around the pixel with value “2” (b). The resulting feature representation is not invariant to rotation.

This is particularly true for the classification of images of objects that can be seen from different perspectives or camera viewpoints, or when processing biomedical and/or astronomical images, where the same objects may appear with a broad variety of orientations. In all these cases, it is often preferable to have an algorithm that produces the same feature representation for an object, independently of the isometric transformation (i.e., rotation, flip and translation) that this object undergoes in the observed image.

There is, however, no straightforward way of incorporating such prior knowledge inside the classic Convolutional Neural Network (ConvNet) [3] due to the following reasons. Even though the ConvNets are partially invariant to small translations by their construction of convolutional and pooling layers, they are not invariant to rotations and flips. This is illustrated in Fig. 1.1 and 1.2, where we can clearly see that convolutional and max-pooling operations that are frequently used in ConvNets do generally not produce invariant feature representation for an object that undergoes different isometric transformations.

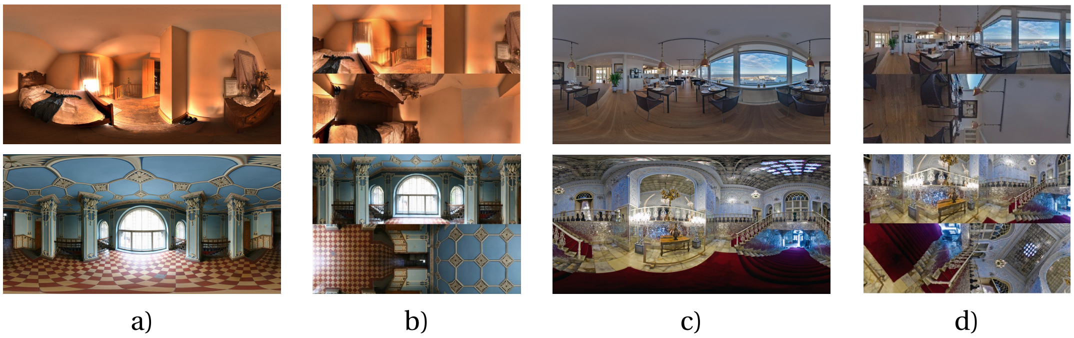


Figure 1.3 – Examples of equirectangular (a,c) and cube-map projection (b,d) representations of omnidirectional images. The distortion of equirectangular image (a,c) is stronger closer to the pole and the distortion of cube-map projection (b,d) mostly occurs due to the discontinuity on the borders of the faces of the cube.

Camera geometry. The second type of prior knowledge that we consider in this work relates to the projective geometry of the camera. This information is directly connected with various distortion effects that appear in the images captured by non-classical cameras, like omnidirectional ones. To efficiently use such prior knowledge we developed several methods that modify deep learning architectures in order to adapt to the geometry distortion effects introduced by the camera lens and produce features that are independent of such distortions. In particular, we are interested in omnidirectional cameras, which are frequently used in areas such as robot navigation and virtual reality due to their wide 360-degree field of view.

Fig. 1.3 illustrates several examples of distortion effects that are introduced by different representations of omnidirectional images. One of the most popular ways of representing such images is to use equirectangular projection [4], which maps the spherical image to a rectangular one that can then be processed with standard techniques. However, due to the process of unwrapping of the spherical surface onto a planar one, such representation of omnidirectional images contains strong distortion artifacts that can be seen in Fig. 1.3 (a,c). This essentially leads to the same object having different shapes depending on the position at which it appears on the omnidirectional image.

In order to reduce the influence of such geometric distortion effects, the cube-map projection [5] maps the image from the spherical surface to an imaginary cube that surrounds it. Then, this cube is unwrapped and its faces are rearranged in a special order to form a rectangular image. Such an approach prevents the appearance of severe artifacts that are seen in the case of the equirectangular projection. However, it introduces other artifacts due to the discontinuity on the boundaries of the cube faces. An example of the cube-map projection of the omnidirectional image can be seen in Fig. 1.3 (b,d)

Our approach. In this work, we propose to use the power of graph-based representations to integrate geometric priors into deep learning architectures. We represent an image as a signal on a graph, where each node and the signal value that is associated with it, corresponds to a pixel location and its intensity value respectively. This representation is very efficient for our task, due to the following reasons.

First, an undirected graph that represents an image does not have any orientation in space, as the ordering of the nodes (that represent pixels) can be set arbitrarily. This permits for development of convolutional filters that operate in the spectral domain and are equivariant to isometric transformations, which are applied to the image. We present an efficient way of learning such filters, in Chapter 3. In order to compress this feature representation, we introduce a special dynamic pooling layer that preserves the original graph structure and makes the signal sparser. All the aforementioned operations result in learning features that are equivariant to geometric transformations of the input signal. We further develop a special statistical layer, which learns multi-scale statistics that are independent of the ordering of graph nodes and therefore transformation invariant. Combining all these steps results in a deep learning architecture that is inherently invariant to isometric transformation of the image signal.

Then, graphs have the natural flexibility to encode the relations between the nodes (pixels) via the modification of the edge weights that connect these nodes and/or by changing the connectivity pattern between them. This is particularly useful for incorporating the knowledge about the geometry of omnidirectional images inside the deep learning framework. In this work, we suggest two different approaches to achieve this objective. Our first method extends our previous approach [6] to omnidirectional images by adjusting the weights of the edges between the graph nodes in such a way that the resulting response of a spectral convolutional filter is similar for the same image pattern independently of its position on the image. Similarly to [6], this method operates on images represented as signals on an undirected graph and, apart from adapting features to the geometry of omnidirectional cameras, it also makes them invariant to isometric transformations. While effective for such tasks as image classification, enforcing such invariance is not necessarily ideal in other tasks, such as image compression, where the orientation of the pattern plays an important role. We, therefore, develop an alternative approach that relies on multiple directed graphs, which in turn permits learning anisotropic graph filters. This approach further permits to adjust relative weights of different filter elements, and also to change the filter's shape to adapt to the image projective geometry in the neighborhood of a node (pixel).

1.2 Thesis outline

The goal of this thesis is to present a solution for extending deep learning techniques with a set of predefined priors using powerful graph-based representations.

As graph signal processing is a central part of the thesis, we start by an introduction of this topic in Chapter 2. We discuss the generalization of Fourier transform and the filtering operation, which is performed in the spectral domain of the graph. Then, we review the recent trends in the literature related to our approach, where deep learning marries with graph signal processing tools.

In Chapter 3 we propose a novel graph-based approach to learn image feature representations. We define quasi-eqivariance property and show that graph filtering operation permits to learn features that are quasi-equivariant to isometric transformations up to pixel resolution. This permits to have equivariant representation at the input of fully-connected layer. However, it is not enough to preserve invariance to isometric transformations, as the neurons of the fully-connected layers have to learn the representation of the input depending on the isometric transformation it undergoes. Thus, we suggest a novel statistical layer, which creates inherently isometric invariant representations. These can then be efficiently processed by a fully-connected layer, since the deformed objects have the same feature representation as the original ones. Our experiments on the image classification tasks show that our network is able to correctly recognize isometrically transformed objects even though such type of transformation was not seen by the network at training time, while standard deep learning techniques are typically not able to succeed in solving this problem.

Next, in Chapters 4 and 5 we present two different approaches that make the features learned via a deep neural network invariant to the geometric distortion of the omnidirectional camera. In Chapter 4, we depart from the proposed approach in Chapter 3 and extend a deep network architecture to images with a wide field of view, such as omnidirectional images. As images are represented by a graph, we have the flexibility of choosing different weights of the edges, which permits to change the response of the graph-based filter with its location in the image. We benefit from this property by introducing specific weights of the edges between the graph nodes that force the filter to have similar response for the same image pattern seen at different positions on the omnidirectional image. It permits learning transformation invariant features, which are effective for tasks such as image classification. Our experiments show that various techniques that operate on the graph with properly adjusted edge weights are able to reach higher accuracy for omnidirectional image classification task as compared to using regular grid graph with equal weights. This is achieved due to the fact that the network becomes aware about geometry of the camera.

Finally, processing the orientations of the image patterns is important in some applications like compression. Therefore, in Chapter 5, we suggest another representation learning algorithm that relies on multiple directed graphs. It is able to learn anisotropic filters that adapt their shape and size to the geometry of the images. To achieve this, we introduce a novel anisotropic filter based on multiple directed graphs. In contrary to the previous method, these filters are not isotropic and, therefore, are able to encode the orientation of the object in the image, which is important for various tasks such as omnidirectional image compression. Unlike standard convolutional neural networks, which use the information from a fixed neighborhood of each pixel, our filters are capable of adapting their size and shape to the geometry introduced by omnidirectional cameras, which ultimately results in a more efficient representation of the object in the image. The design of our graph structure is flexible and capable of modeling different image projective geometries. We, finally, show that our approach can be efficiently used in a more general scenario, where the representation of the image is non-spherical. For example, we have experimented with a cube-map projection (which is becoming popular nowadays for representing 360-degree images) and showed that our approach is able to efficiently cope with the distortion effects that arise in this case. Our network achieves state-of-the-art performance on different image classification tasks, where images are projected to various surfaces. Further, when applied to an image compression problem, our method is able to avoid various artifacts, which appear when using conventional compression methods, due to the knowledge about the image geometry.

1.3 Summary of contributions

We propose the following main contributions in the thesis:

- a novel graph-based image representation learning framework, which is inherently invariant to isometric transformation, and therefore able to correctly classified transformed objects in images, even if such transformations are not present in the training dataset;
- a new graph-based statistical layer for deep network architectures that leads to an effective transformation-invariant classification of images;
- a principled way for constructing graphs based on the projective geometry of images, which leads to a similar feature representation for the same object appearing at various positions of an image with particular geometry;
- a generic directed graph construction technique that allows developing anisotropic filters and permits adapting their shapes and sizes to various image projective geometries;
- a novel approach, which exploits image projective geometry and permits reducing compression artifacts that are specific for cube-map projected images compared to standard techniques.

2 Graph signal processing meets deep learning in vision

Convolutional neural networks (ConvNets) [7] have shown state-of-the-art performance in various computer vision tasks [1, 2]. They prove to be very effective when working with image data, which can be seen as type of data with regular (grid) structure. However, in reality there exist large number of tasks, that focus on working with irregular data, including social networks, brain signals, pointclouds, etc [8]. These data types typically have an underlying structure, which, however, cannot be represented as a regular grid. Therefore, applying ConvNets in this case is challenging as pooling and convolutional layers are not defined for such irregular data. This problem has recently gained a lot of attention from the research community. The class of methods that tackle the problem of efficient representation of irregular data and processing with deep learning technique is typically referred to as *geometric deep learning* [8]. In this thesis we focus on one of such representations, namely graph-based representation, which permits modeling irregular relations between items. Thus, in this chapter we review the most recent geometric deep learning techniques.

The remainder of the chapter is organized as follows. We start by discussing some graph signal processing elements that are crucial for the techniques discussed in the following chapters and for understanding the difference between the existing approaches. We then discuss in more details different ways of how each of the key ingredients of a convolutional neural network (i.e. convolutional and pooling layers) can be transformed such that the resulting architecture is capable of processing irregular data.

2.1 Graph signal processing elements

We now briefly review some elements of graph signal processing. We represent irregular data as signal y on a graph G, which is able to efficiently represent the underlying geometrical structure of the data. 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, $\mathcal{E}$ 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 $w(v_i, v_j) = w(v_j, v_i)$, which is usually chosen to capture the similarity 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 $y(v_n): \mathcal{V} \rightarrow \mathbb{R}$.

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 [9]. In that respect, we first consider the unnormalized graph Laplacian operator of the graph G, defined as

$$\mathscr{L}_{\mathrm{u}} = \begin{cases} d(v_i)\,, & \text{if } i = j, \\ -w(v_i, v_j)\,, & \text{if } i \neq j,\ v_i \sim v_j\,, \\ 0\,, & \text{otherwise}\,, \end{cases} \tag{2.1}$$

and then the normalized one, defined as

$$\mathscr{L} = \begin{cases} 1\,, & \text{if } i = j,\ d(v_i) \neq 0\,, \\ -\big(d(v_i)d(v_j)\big)^{-1/2}\,, & \text{if } i \neq j,\ v_i \sim v_j\,, \\ 0\,, & \text{otherwise}\,, \end{cases} \tag{2.2}$$

where $v_i \sim v_j$ denotes that node v_i is adjacent to the node v_j, and $d(v_i)$ is the degree of the vertex v_i, computed as

$$d(v_i) = \sum_{j=0, j\neq i}^{N} w(v_i, v_j)\,. \tag{2.3}$$

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 $\lambda_{N-1} = \lambda_{\max} = 2$ for the normalized Laplacian $\mathscr{L}$. 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:

$$\hat{y}(\lambda_i) = \sum_{n=1}^{N} y(v_n)\chi_i^*(v_n), \tag{2.4}$$

and

$$y(v_n) = \sum_{i=0}^{N-1} \hat{y}(\lambda_i)\chi_i(v_n). \tag{2.5}$$

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

$$(y_1 * y_2)(v_n) = \sum_{i=0}^{N-1} \hat{y}_1(\lambda_i)\hat{y}_2(\lambda_i)\chi_i(v_n). \tag{2.6}$$

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

$$\hat{y}_f(\lambda_i) = \hat{y}(\lambda_i)\hat{h}(\lambda_i), \tag{2.7}$$

where $\hat{h}(\lambda_i)$ is the spectral representation of the graph filter h 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 $H \in \mathbb{R}^{\mathbb{N}\times\mathbb{N}} : H = \chi\hat{H}\chi^T$, where $\hat{H}$ is a diagonal matrix constructed on the spectral representation of the graph filter:

$$\hat{H} = \mathrm{diag}(\hat{h}(\lambda_0), \ldots, \hat{h}(\lambda_{N-1})). \tag{2.8}$$

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 [10]:

$$\begin{aligned} T_{v_n} y &= \sqrt{N}(y * \delta_{v_n}) \\ &= \sqrt{N}\textstyle\sum_{i=0}^{N-1} \hat{y}(\lambda_i)\chi_i^*(v_n)\chi_i. \end{aligned} \tag{2.9}$$

Thus, convolutional filtering operation can be defined using the normalized or unnormalized Laplacian operators as follows:

$$\begin{aligned} &\mathscr{F}(y) = \textstyle\sum_{m=0}^{M} \alpha_m \mathscr{L}^m y, \\ &\mathscr{F}_{\mathrm{u}}(y) = \textstyle\sum_{m=0}^{M} \alpha_m \mathscr{L}_{\mathrm{u}}^m y, \end{aligned} \tag{2.10}$$

where a_m is a parameter of the convolutional filter and y is a signal defined on graph. Spatially, these filters operate on the defined m-hop neighbourhood of each vertex. More details about the above graph signal processing operators can be found in [9].

2.2 Graph-based ConvNets

We now review in more details the recent techniques that adapt convolutional operation, which is the key component of the classical ConvNets, to irregular data structures. Part of our chapter is based on the survey [8] extended with the recent state-of-the-art techniques.

In order to define the convolutional operation on a graph a number of methods have been proposed. These approaches can be roughly divided in three different categories:

1. **Spectral** that work with the spectral representation of the graph, following the convolutional theorem, which states that a convolutional operation in the spatial domain (i.e. is the domain of graph vertexes) is equal to the point-wise multiplication in the spectral domain.
2. **Spectrum-free** that are based on the polynomial of Laplacian filters both acts in the frequency domain of the graph and well-localized on it. These methods can be defined in spatial domain as well.
3. **Spatial** that work directly in the spatial domain and represent a filter as a template and define the convolutional operation as a template matching procedure.

In the remainder of this section we discuss in more details each of these types of approaches.

Convolutional filters in spectral domain. One of the pioneering efforts to define a convolutional filter in spectral domain, was done by [11]. They suggest to use eigendecomposition and apply Eq. (2.4) to transfer the signal y, defined on a graph, from vertex to spectral domain. The convolutional operation is then defined in this spectral domain using Eq. (2.6), where the size of the convolutional filter depends on the number of eigenvectors being used. In [11] the authors suggest using k first eigenvectors of y, as they represent the low frequencies of the signal. After the convolutional operation is applied to spectral representation of y, the result of this operation is converted back from spectral to vertex domain. While effective this method has certain limitations. First it is prone to overfitting, as it relies on a large number of parameters, which depends on the number of eigenvalues and eigenvectors. To overcome this problem and make the method more stable the authors of [12] propose to use smooth multipliers. This results in eigenvectors being ordered according to their respective eigenvalues and spectral multipliers being parametrized with a fixed interpolation kernel (e.g. a cubic spline). Further, even though this method has theoretical similarities with the standard convolutional filter, it is computationally non-effective and time-consuming as it requires computing expensive matrix multiplication steps $\mathcal{O}(N^2)$ during forward and back passes of the spectral CNN network, where N is the cardinality of the signal. To overcome the computational complexity of these approaches there were introduce methods, which allow to avoid explicit computation of Laplacian eigenvectors.

Spectrum-free convolutional filters. In order to avoid complex eigendecomposition, the method [13] shows that convolutional filters can be represented as Laplacian polynomials using Eq. (2.10). This permits to do spectral filtering of the signal without its actual conversion to spectral domain, thus avoiding the time-consuming eigendecomposition operation [6, 14]. This also leads to having a better control on the number of filter parameters, as they are directly related to the degree of the polynomial. To even further improve the speed of the convolutional operation the authors in [14] suggest approximating these functions by Chebyshev polynomials [15], which can be efficiently generated in a recurrent fashion.

Their approach, which is typically referred to as ChebNet, allows to significantly reduce the complexity of the filtering operation to $\mathcal{O}(NM)$, with M and N being the degree of the polynomial and the cardinality of the signal respectively.

Further, the work [16] simplifies the aforementioned work by fixing the degree of the polynomial filters to 1. The authors assume that the maximum eigenvalue $\simeq 2$, which permits obtaining a convolution filter, represented by a single parameter. For each graph node p in the vertex domain, such filter acts on its 1-hop neighborhood and effectively sums the values of the signal across all the direct neighbours of p and then computes the weighted sum of this value with the value of the signal at p. The authors in the work [16] show that having such relatively small filters within the deep learning architecture significantly improves the speed of the approach and is generally enough for various problems, as the multi-layer structure of the network allows to model complex relationships between the nodes of the graph.

The disadvantages of ChebNet is that the eigenvalues of the Laplacian, which are used for spectral filters are concentrated near very few values with the large gaps remained. Therefore, the authors from [17] propose to replace laplacian filters by the ones based on Cayley transform, which results in spectral filters on graphs that focus on frequency bands of interest, which allow to more evenly cover the whole spectrum of frequencies. These Cayley filters are special cases of filters based on general rational functions of the Laplacian, namely ARMA filters [18].

The aforementioned techniques assume that graph is undirected and its Laplacian matrix is symmetric and positive semi-definite with orthogonal eigendecomposition, which leads to their spectral interpretation. Contrary to these approaches there exist several graph-based techniques that relax the undirected graph requirement. The work [19] proposes a method that applies filtering operation directly on a graph, which can be both directed and undirected. The steady state requirement of this technique is then relaxed by the works [20, 21]. The work [20] further improves [19] by removing the need in constraining the model parameters to ensure convergence and the authors in [21] additionally drop the requirement of being recurrent. Further, the work [22] proposes a different deep learning architecture that they refer to as MotifNet, which operates on directed graphs by exploiting their motifs that is originally introduced in [23]. MotifNet outperforms approaches with Laplacian polynomial filters based on undirected graph for some nodes classification tasks, however, their main limitation is that they assume that the notion of direction can be derived from the local structural properties of the graph, which is not always the case.

Another issue with spectral approaches is that different graphs have different eigenvalue distributions. This essentially makes it impossible to use filters in a certain domain, if they are learned in a different one, which can be a severe limitation for such tasks as 3D point cloud segmentation. To overcome this problem the authors of [24] suggest mapping each of the individual domains to a canonic one using a so-called functional map tool. The main limitation of this approach is the requirement of eigendecomposition computation for both the Fourier and the Inverse Fourier Transforms. A different way of designing a domain independent architecture is to use the spatial techniques, which we discuss in the following section.

Convolutional filters defined in spatial domain. In this section we briefly discuss the most recent techniques that introduce different ways of developing a convolutional filter that operates in the vertex domain of the graph. A standard convolutional filter that operates on Euclidean data is typically defined as a patch, which has it is local system of coordinates and can be translated to each of the nodes. The convolutional operation is then defined at each vertex as a correlation between this defined patch and the neighbourhood of this vertex. Therefore, in order to define a convolutional operation on a graph, it is necessary to define both the coordinate system of the filter and the function that allows to compute the value of the filter for each node of the graph.

We start our discussion with the methods, designed for working with manifolds with the main focus on learning the correspondences between shapes. The authors in [25] work with manifold data, where they define the filter patch in a space tangent to the manifold. The filter is then defined in local polar coordinates of the patch with equal radius and angle steps. A different approach was taken by the authors in [26], who propose to design a convolutional filter directly on the manifold shape. They exploit the heat kernel theory on non-Euclidean data, which suggests that the rate of point temperature change is proportional to the difference between its own temperature and the temperature of the surrounding points. The authors propose to modify this heat equation by introducing special parameter, a so-called thermal conductivity tensor, which permits to model the heat flow depending on the position and orientation. The work [27] then proposes a specific choice of this tensor for a 2D manifold, which allows to rotate the flow depending on the maximum curvature direction and control the degree of anisotropy. Thus, this filter can be written using eigenfunctions and eigenvalues of the introduced anisotropic Laplacian operator. Further, the authors from [28] propose to use a discrete version of this operator and use it to form a deep convolutional neural network that operates on graphs.

Then, the authors of MoNet [29] propose an extension of the aforementioned techniques, which is able to work not only with manifolds but also with the generic graph structures. Briefly, they suggest to define a local coordinate system at each patch and the weights of the convolutional filter are obtained by applying a parametric Gaussian kernel with the mean vector and the covariance matrix being trainable parameters. The authors show that this method can be seen as generalization of such methods as ConvNets [3], Geodesic CNN [25] or Anisotropic CNN [27], and performs well on both shape correspondence problem and graph classification task. Another work, which can be seen as a particular instance of MoNet is Graph Attention Network (GAT) [30], where the authors propose to use attention mechanism to define the weight for a convolutional filter, which is defined as a neighborhood averaging function. Compared to [29], GAT does not rely on the fact that graph is given and uses node features measure the similarity between the graph nodes. The method [31] extends the work of [30] even further by introducing the notion of primal and dual graph, which permits to learn features both on the vertexes and the edges of the graph. A different direction was taken by the authors of SplineNet [32], who suggest training the network directly on the geometrical structures. Further, compared to general graph methods [14, 16, 17] SplineNet relies not only on the node's connectivity information, but also uses the knowledge about the relative positions of the nodes.

A different set of techniques was proposed for point cloud classification, segmentation and scene semantic parsing tasks. The pioneering work, which operates directly on the point sets is PointNet [33]. The idea of this method is to learn spatial local features of each point independently of the other points of the graph (point cloud). Then all these individual point representations are aggregated to form a global point cloud signature. This method was improved by PointNet++ [34], which for each vertex proposes exploring its neighborhood (according to the metric space distance) in order to enrich the learned local features, which consequently improves the results comparing to [33]. These methods, however, still treat the points from the local sets independently and do not consider the relationships between point pairs, therefore, the authors in [35] propose to perform graph-based edge convolution where the graph is used to represent the structure of the local neighborhood. Further, based on their experiments the authors conclude that having a dynamic graph structure that is able to adapt to the features learned at each layer of the network, as compared to the fixed graph with the neighborhoods defined according to the spatial distances between the nodes, is beneficial for the overall performance. Another interesting approach [36] suggests representing 3D points as a sparse set of samples on a high-dimensional lattice. The key advantage of the work is that they can jointly work with 2D and 3D points, which gives them the possibility to combine the information from both the point cloud and the image-based representation of a scene.

3 Isometric transformation invariant image representation learning

3.1 Introduction

Deep convolutional networks (ConvNets) have achieved impressive results for various computer vision tasks, such as image classification [1] and segmentation [2]. 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 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. 3.1. 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 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

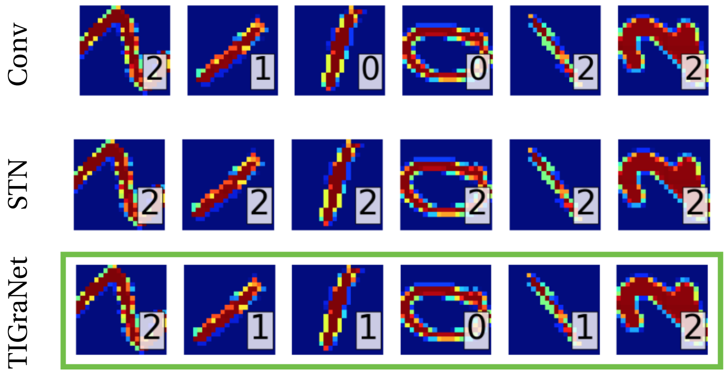


Figure 3.1 – Illustrative transformation-invariant handwritten digit classification task. Rotated test images, along with their classification label obtained from ConvNets (Conv) [37], Spatial-Transformer Network (STN) [38], and our method. (best seen in color)

deep neural network called TIGraNet[1], 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.

3.2 Transformation invariant techniques

Most of the recent architectures [3, 1] 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.

One intuitive way to make the classification architectures more robust to isometric transformations is to augment the training set with transformed data (e.g., [39]), 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 [40] construct deep neural networks that operate in parallel on the original and transformed images simultaneously with weight-shared convolutional filters. Then, the authors in [41] propose to use max-pooling to combine the outputs of these networks. A different approach was proposed in [38], where the authors introduce a new spatial transformer layer that deforms images according to a predefined transformation class. Then, the work in [42] suggests using rotated filter banks and a special max pooling operation to combine their outcomes and improve invariance to transformations. The authors in [43] propose a generalization of the ConvNets and introduce equivariance to 90° rotations and flips. Finally, the authors in [44]

---

[1] The code is available online: https://github.com/LTS4/TIGraNet

exploit rotation symmetry in the Convolutional Network for the specific problem of galaxy morphology prediction. This work has been extended in [45] 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 [46, 47, 48] are the closest in spirit to ours. In order to be invariant to local transformation, the works in [46, 47] 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, [49, 50] 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 [48] proposes a so called Harmonic Network, which uses specifically designed complex valued filters to make feature representations equivahereriant 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.

A different type of methods, which are very related to our technique are the graph-based deep learning approaches that are discussed in Chapter 2. However, most of these methods either integrate graph features directly to the fully-connected layer of the neural network or are designed for specific applications and cannot be directly applied to our task. Therefore, to the best of our knowledge, most of the existing 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.

3.3 Graph-based convolutional network

We now present the overview of our new architecture, which is illustrated in Fig. 5.6. The input to our system can be characterized by a normalized Laplacian matrix $\mathscr{L}$ 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.

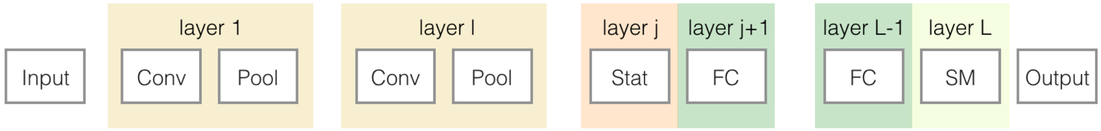


Figure 3.2 – **TIGraNet** architecture. The network is composed of an alternation of spectral convolution layers $\mathscr{F}^l$ and dynamic pooling layers $\mathscr{P}^l$, followed by a statistical layer $\mathscr{H}$, multiple fully-connected layers (FC) and a softmax operator (SM). The input of the network is an image that is represented as a signal y_0 on the grid-graph with Laplacian matrix $\mathscr{L}$. The output of the system is a label that corresponds to the most likely class for the input sample.

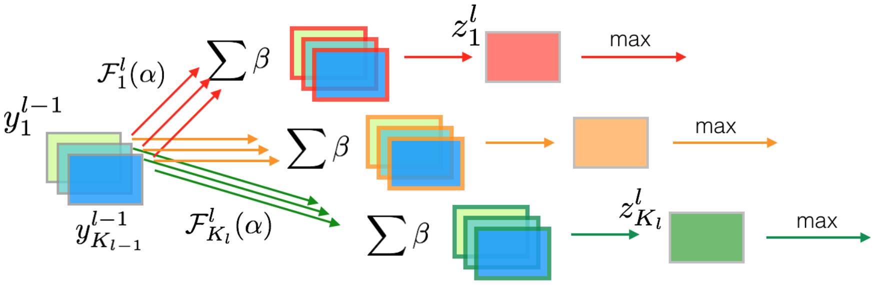


Figure 3.3 – Spectral convolutional layer $\mathscr{F}^l$ in **TIGraNet**. The outputs of the previous layer $l-1$ are fed to a set of filter operators $\mathscr{F}_i^l$. The outputs of $\mathscr{F}_i^l$ are then linearly combined to get the filter maps z_i^l that are further passed to the dynamic pooling layer.

In more details, our deep learning architecture consists of an alternation of spectral convolution layers $\mathscr{F}^l$ and dynamic pooling layers $\mathscr{P}^l$. They are followed by a statistical layer $\mathscr{H}$ and a sequence of fully-connected layers (FC) that precedes a softmax operator (SM) that produces a categorical distribution over labels to classify the input data. At layer l, both the spectral convolution and the dynamic pooling layers contain K_l operators denoted by $\mathscr{F}_i^l$ and $\mathscr{P}_i^l, i = 1, \ldots, K_l$, respectively. Each convolutional layer $\mathscr{F}_i^l$ 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 $\mathscr{H}$ 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.

3.3.1 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 $\mathscr{F}_i^l$, as illustrated in Fig. 3.3. However, each filter i operates in the graph spectral domain. In order to avoid computing the graph eigen-decomposition that is required to perform filtering through Eq. (2.6), we choose to design our graph filters as smooth polynomial filters of order M [10], which can be written as

$$\hat{h}(\lambda_l) = \sum_{m=0}^{M} \alpha_m \lambda_l^m. \tag{3.1}$$

As we mentioned before, following the notation of Eq. (2.8), each filter operator in the spectral convolutional layer l can be written as

$$\mathscr{F}_i^l = \sum_{m=0}^{M} \alpha_{i,m}^l \mathscr{L}^m, \tag{3.2}$$

where $\mathscr{L}^m$ 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 [10]. 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 parameter in the classical ConvNets.

The filtering operation then simply consists in multiplying the graph signal by the transpose of the operator defined in Eq. (3.2), namely

$$\tilde{y}_{i,k}^l = \left[\mathscr{F}_i^l|_{\mathcal{N}_i^{l-1}}\right]^T y_k^{l-1}, \tag{3.3}$$

where y_k^l and $\tilde{y}_{i,k}^l$ are the graph signals at the input and respectively the output of the l^{th} spectral convolutional layer (see Fig. 3.3). 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 l^{th} spectral convolutional layer is a set of K_l feature maps z_i^l. Each i^{th} feature map is computed as a linear combination of the outputs of the corresponding polynomial filter as follows:

$$z_i^l = \sum_{k=1}^{K_{l-1}} \beta_k^l \, \tilde{y}_{i,k}^l, \tag{3.4}$$

where the set of signals $\tilde{y}_{i,k}^l$ are the outputs of the i^{th} polynomial filter applied on the K_{l-1} input signals of the spectral convolutional layer with Eq. (3.3). The vector of parameters

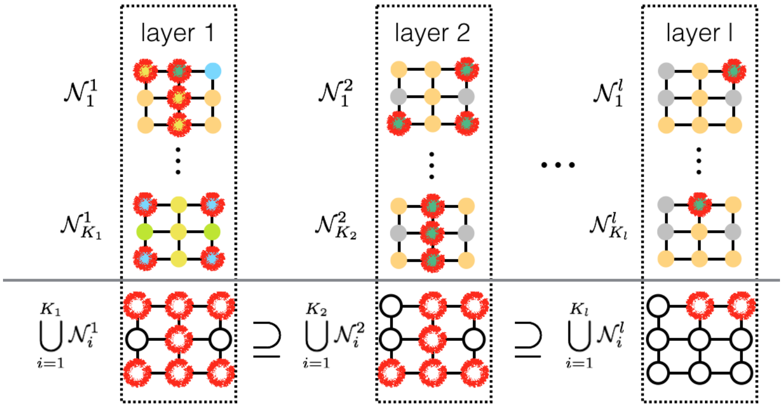


Figure 3.4 – Pooling process, with succession of dynamic pooling layers with operators $\mathscr{P}_i^l$ that each selects the vertices with maximum intensity according to Eq. (3.5).

$\{\beta_k^l\}$, 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. 3.3. Lastly, the complexity of spectral filtering can be computed based on the fact that $\mathscr{L}$ and thus the filters are sparse matrices. Then, the complexity is $O(|\mathscr{E}_M|N)$ where $|\mathscr{E}_M|$ is a maximum number of nonzero elements in the columns of $\mathscr{F}_i^l$.

3.3.2 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 [51] we introduce a novel layer that we refer to as dynamic pooling layer, which basically 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., $\Omega^{(1)} = \mathscr{V}$. 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 $\mathscr{N}_i^l$. The union of these sets for the different features maps z_i^l form the new set Ω^l, i.e.,

$$\Omega^l = \bigcup_{i=1}^{K_l} \mathscr{N}_i^l. \tag{3.5}$$

The set Ω^l drives the pooling operations at the next dynamic pooling layer $\mathscr{P}^{l+1}$. We note that, by construction, the different sets Ω^l are embedded, namely we have $\Omega^l \supseteq \Omega^{l+1}$, $\forall l \in [1..L]$. The Algorithm 1 summarize our approach, and Fig. 3.4 illustrates the effect of the pooling process through the different network levels.

Algorithm 1 Dynamic pooling layer at layer l.

```
Input:  Feature maps $z_i^l, i \in [1, K_l]$
        Set of nodes of interest $\Omega^{l-1}$
        Number of active nodes, $J_l$
for $i \in [1, K_l]$ do
    $\overline{\mathcal{N}_i^l} = \mathcal{V}$
    $\mathcal{N}_i^l = \emptyset$
    for $j \in [1, \max\left(J_l, |\Omega^{l-1}|\right)]$ do
        $v = \underset{v \in \left(\Omega^{l-1} \cap \overline{\mathcal{N}_i^l}\right)}{\operatorname{argmax}} z_i^l(v)$
        $\overline{\mathcal{N}_i^l} = \overline{\mathcal{N}_i^l} \setminus \{v\}$
        $\mathcal{N}_i^l = \mathcal{N}_i^l \cup \{v\}$
    end for
end for
$\Omega^l = \bigcup_{i=1}^{K_l} \mathcal{N}_i^l$
```

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. (3.3).

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

3.3.3 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 classical ConvNet architectures, we first insert a new statistical layer, whose output is then fed into fully-connected layers (see Fig. 5.6).

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 will 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 statistical layer, which computes transformation-invariant statistics of the

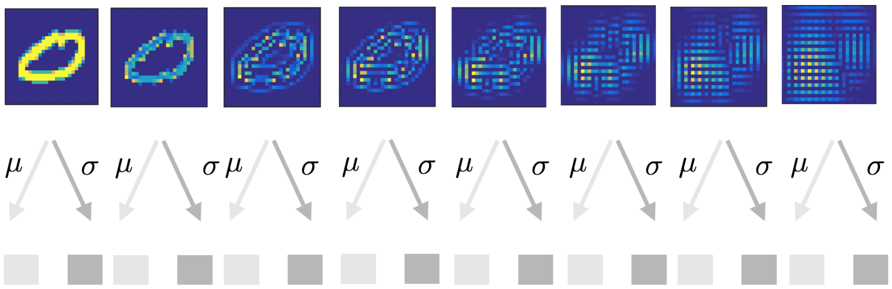


Figure 3.5 – Illustration of the statistical layer, which calculates multiscale statistics by filtering each input feature map by Chebyshev polynomials filters [15] and taking their second-order statistics. Resulting statistics are vectorized and build an invariant to graph isometric transformation.

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 $\mathscr{P}^{j-1}$ 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 the advantage of a fast computation due to their iterative construction, and they can be adapted to distributed implementations [15]. In order to construct these polynomials, we first shift the spectrum of the Laplacian $\mathscr{L}$ to the interval $[-1,1]$, which is the original support of Chebyshev polynomials. Equivalently, we set $\tilde{\mathscr{L}} = \mathscr{L} - I$.

As suggested in [14], for each input feature map $\tilde{z}_i$ we iteratively construct a set of signals $t_{i,k}$ using graph Chebyshev polynomials of order k, with $k \le K_{max}$, as

$$t_{i,k} = 2\tilde{\mathscr{L}}\, t_{i,k-1} - t_{i,k-2}, \tag{3.6}$$

with $t_{i,0} = \tilde{z}_i$ and $t_{i,1} = \tilde{\mathscr{L}}\tilde{z}_i$. 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 $\sigma^2_{i,k}$ for each signal $|t_{i,k}|$ (see Fig. 3.5). This forms a feature vector ϕ_i of $2K_{max}+2$ elements, i.e., $\phi_i = [\mu_{i,0}, \sigma^2_{i,0}, \dots, \mu_{i,K_{max}}, \sigma^2_{i,K_{max}}]$. 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 5.4.1. The output of the fully-connected layers is then fed to a softmax layer [52], which finally returns the probability distribution of a given input sample to belong to a given set of classes.

3.3.4 Training

We use supervised learning and train our network so that it maximizes the log-probability of estimating the correct class of training samples via logistic regression. Overall, we need to compute the values of the parameters in each convolutional and in fully-connected layers. The other layers do not have any parameter to be estimated. We train the network using a classical back-propagation algorithm and learn the parameters using ADAM stochastic optimization [53].

We provide more details here about the computation that are specific to our new architecture. We refer the reader to [54] for more details about the overall training procedure. The back-propagation in the spectral convolutional layer is performed by evaluating the partial derivatives with respect to the parameters $\alpha : \alpha \in \mathbb{R}^{K_{l-1} \times M}$ of the spectral filters, and to the parameters $\beta : \beta \in \mathbb{R}^{K_{l-1}}$ of the feature map construction. The partial derivatives read

$$\frac{\partial E}{\partial \alpha_{i,m}^{l}} = \sum_{k=0}^{K_{l-1}} \beta_k^l \left[\mathscr{L}^m |_{\mathscr{N}_i^{l-1}} \right] y_k^{l-1} \frac{\partial E}{\partial z_i^l}, \tag{3.7}$$

$$\frac{\partial E}{\partial \beta_j^l} = \sum_{m=0}^{M} \alpha_{i,m}^l \left[\mathscr{L}^m |_{\mathscr{N}_i^{l-1}} \right] y_j^{l-1} \frac{\partial E}{\partial z_i^l}, \tag{3.8}$$

where E is the negative log-likelihood cost function, $z_i^l = y_i^l$ is the output feature map of layer l, K_{l-1} denotes the number of feature maps at the previous layer of the network, M is the polynomial degree of the convolutional filter and $\mathscr{L}$ is the Laplacian matrix. Then, we further need to compute the partial derivatives with respect to the previous feature maps as follows

$$\frac{\partial E}{\partial y_j^{l-1}} = \beta_j^l \sum_{m=0}^{M} \alpha_{i,m}^l \left[\mathscr{L}^m |_{\mathscr{N}_i^{l-1}} \right] \frac{\partial E}{\partial z_i^l}. \tag{3.9}$$

Our new dynamic pooling layers, as well as our statistical layer do not have parameters to be trained. Similarly to the max-pooling operator our dynamic pooling layer permits back-propagation through the active nodes since the gradient is 0 for the non-selected nodes and not zero for the chosen ones. Further, the statistical layer back-propagates the gradients as follows:

$$\frac{\partial E}{\partial t_{i,k}} = \frac{1}{N} \frac{\partial E}{\partial \mu_{i,k}}, \tag{3.10}$$

$$\frac{\partial E}{\partial t_{i,k}} = \frac{2(N-1)}{N^2} \sum_{i=1}^{N} \left(t_{i,k} - \mu_{i,k}\right) \frac{\partial E}{\partial \sigma^2_{i,k}}, \tag{3.11}$$

where $\mu_{i,k}, \sigma^2_{i,k}$ are the inputs to the first fully-connected layer and the outputs of the statistical layer. The derivatives $\partial E / \partial \tilde{z}_i$ are then computed as:

$$\frac{\partial E}{\partial \tilde{z}_i} = \sum_{k=0}^{K_{max}} \frac{\partial E}{\partial t_{i,k}} \frac{\partial t_{i,k}}{\partial \tilde{z}_i}, \tag{3.12}$$

where $\partial t_{i,k} / \partial \tilde{z}_i$ are simply the derivatives of Chebyshev polynomials [15] with maximum order K_{max}. Please note that we use the non-linear absolute function $|t_{i,k}|$ before statistical layer, therefore, the gradient at $t_{i,k} = 0$ is not defined. In practice, however, we set it to 0, which gives us a nice property of encouraging some feature map values to be 0 and favors sparsity.

Finally, the parameters of the fully-connected layers are trained in a classical way, similarly to the training of fully-connected layers in ConvNet architectures [54].

3.4 Equivariance of graph filters

In this section we analyze the equivariance property of the features produced by the graph-based convolutional layer. We start our analysis by proving that independently of the input image y, graph filters are equivariant to isometric transformations of y such as image rotations by multiplier of 90° and translations by an integer number of pixels in horizontal and vertical directions. We refer to these transformations as the graph isometric ones.

We then show that given some constraints on the smoothness conditions on y, our graph-convolutional filters are *quasi-equivariant* to any isometric transformations (which comprise rotations by an angle of any degree and translations by any real number of pixels).

3.4.1 Graph isometric transformation

In this section we formally define isometric transformations g on regular graphs and show that the Laplacian polynomial filters, introduced in Eq. (3.2), produce inherently equivariant features with respect to such transformations g.

Definition 1. *A graph isometric transformation g is a bijective mapping $g : \mathcal{V} \to \mathcal{V}$ that preserves distances between all pairs of adjacent nodes $v_i \sim v_j : w(v_i, v_j) = 1$ and acts as a permutation function for signal y: $y_g = g(y)$ that can be formally described as*

$$\forall v_k \in \mathcal{V},\ \exists!\, v_j \in \mathcal{V} : y_g(v_k) = y(v_j)\,, \tag{3.13}$$

that preserves their neighbourhood and where $!$ indicates that the correspondence between *vertices v_k and v_j is a bijective mapping.*

In other words, a graph isometric transformation g is a permutation of the graph nodes $\mathcal{V}$. This results in g being a combination of the following basic operations applied to the image signal y, defined on the regular graph G:

- Image rotations by $\frac{\pi}{2}, \pi, \frac{3\pi}{2}$;
- Image reflection;
- Image translation by an integer number of pixels.

If we ignore the border effects, graph isometric transformations g satisfy the four fundamental group properties [55] and, therefore, form a group $\mathcal{G}$.

Theorem 1. *For any operation g from a group of graph isometric transformations $\mathcal{G}$ the response of a polynomial filter defined in Eq. (3.2) produces an equivariant feature representation.*

Proof of Theorem 1. Let us denote by y_g the transformed version of signal $y : y_g = g(y)$. Given the Def. 1 we can prove the theorem by induction. For simplicity and without loss of generality we prove the theorem for a 4-nn grid graph.

Base case: Let us first consider the polynomial filter $\mathcal{F}$ of degree $M = 1$. The result of the filtering operation $\mathcal{F}$, introduced in Eq. (3.2), applied to a signal y will then be the following:

$$\mathcal{F}(y) = (\alpha_0 + \alpha_1 \mathcal{L})y\,,\ \{\alpha_0, \alpha_1\} \in \mathbb{R}\,. \tag{3.14}$$

Here $\mathcal{L}$ is the normalized Laplacian matrix, introduced in Eq. (2.2). For the sake of simplicity and in order to avoid border effects we extend the graph by adding nodes to the image boundary and padding them with zeros. Hence all nodes v_j of the initial graph have degree $d(v_j) = 4$. Consequently, the filtered signal $\mathcal{F}(y)(v_j)$ at node v_j can be computed as:

$$f(v_j) \underset{df}{=} \mathcal{F}\left(y\right)(v_j) = \alpha_0 y(v_j) + \alpha_1 y(v_j) - \frac{\alpha_1}{4} \sum_{i: v_i \sim v_j} y(v_i), \tag{3.15}$$

where $v_i \sim v_j$ indicates adjacent nodes.

We then apply the same filter $\mathcal{F}$ to the transformed signal y_g, which is formed from y via a graph isometric transformation g. According to its definition (see Eq. (3.13)), g has the following property: $\forall v_k \mapsto v_j : y_g(v_k) = y(v_j)$ and the neighborhood of the node v_k maps to the neighborhood of the node v_j. Therefore, the result of $\mathcal{F}$, applied to y_g reads

$$f_g(v_k) \underset{df}{=} \mathcal{F}(y_g)(v_k) = \alpha_0 y_g(v_k) + \alpha_1 y_g(v_k) - \frac{\alpha_1}{4} \sum_{l: v_l \sim v_k} y_g(v_l). \tag{3.16}$$

Our goal is then to show that f_g from Eq. (3.16) is equivariant to f from Eq. (3.15). According to Eq. (3.13), the signal y_g in the neighborhood of the vertex v_k is identical to y in the neighborhood of v_j. Therefore Eqs. (3.15) and (3.16) are equal, which leads to

$$f_g(v_k) = f(v_j)\,, \tag{3.17}$$

where $v_k \in G$ is mapped to $v_j \in G$ via g. Further, Eq. (3.17) is valid for any node v_k, and for each of these v_k's there exists a mapping to v_j according to the graph isometric transformation g. Therefore, by Def. 1, f_g and f are related via g as $f_g = g(f)$ and we can write:

$$\mathscr{F}(y_g) = g\left(\mathscr{F}(y)\right) \text{ or, equivalently } \mathscr{F}\left(g(y)\right) = g\left(\mathscr{F}(y)\right)\,. \tag{3.18}$$

As Eq. (3.18) is obtained without any assumptions on the parameters (α_0, α_1) of the polynomial filter $\mathscr{F}$, we can conclude that the response of any Laplacian polynomial filter $\mathscr{F}$ of degree $M = 1$ is equivariant to graph isometric transformations g. The latter is, therefore, true for a special case of $\alpha_1 = 0$, which corresponds to the polynomial filters $\mathscr{F}$ of degree zero.

Inductive step: We now show that the polynomial filter $\mathscr{F}$ of any degree is equivariant to graph isometric transformation g. We assume that the filter

$$\mathscr{F}_{k-1} = \sum_{m=0}^{k-1} \alpha'_m \mathscr{L}^m \tag{3.19}$$

of degree $M = k-1$ is equivariant with respect to g. Our goal is then to show that the polynomial filter $\mathscr{F}_k$ of degree $M = k$ that has the following form:

$$\mathscr{F}_k(y) = \sum_{m=0}^{k} \alpha_m \mathscr{L}^m y = \alpha_0 y + \mathscr{L}\left(\sum_{m=0}^{k-1} \alpha_{m+1} \mathscr{L}^m y\right)\,, \tag{3.20}$$

is also equivariant with respect to g. To do so we take a polynomial filter $\mathscr{F}_{k-1}$, with coefficients $a'_m = a_{m+1}$ and rewrite Eq. (3.20) as

$$\mathscr{F}_k(y) = \alpha_0 y + \mathscr{L}\mathscr{F}_{k-1}(y). \tag{3.21}$$

We then apply the isometric transformation g to both parts of Eq. (3.21):

$$g(\mathscr{F}_k(y)) = g(\alpha_0 y + \mathscr{L}\mathscr{F}_{k-1}(y)) = \alpha_0 g(y) + \mathscr{L} g(\mathscr{F}_{k-1}(y)). \tag{3.22}$$

As by our assumption $\mathscr{F}_{k-1}$ is equivariant with respect to g and $y_g \underset{df}{=} g(y)$, we can rewrite (3.22) as

$$g(\mathscr{F}_k(y)) = \alpha_0 y_g + \mathscr{L}\mathscr{F}_{k-1}(y_g) \underset{df}{=} \mathscr{F}_k(y_g). \tag{3.23}$$

Based on Eqs. (3.22) and (3.23) we can conclude that a polynomial filter of any degree produces features that are equivariant to the graph isometric transformations.

□

3.4.2 General isometric transformations

In this section we extend the result of Section 3.4.1 to general isometric transformations. We start with the introduction of our image model, we then introduce the quasi-equivariance property and prove that under some assumptions on the image y defined on the graph nodes, Laplacian polynomial filters are quasi-equivariant to general isometric transformations. The latter can be represented by any combination of the following basic operations:

- Graph isometric transformations, defined in Section 3.4.1;
- Image rotation by an arbitrary angle γ;
- Image translation by a real number of pixels ξ.

Image model and quasi-equivariance

Camera captures 3D world and represents it as a 2D image. Therefore, an image is an approximation of a real scene with discrete pixel values. For the sake of simplicity let us represent the real 3D image signal y as a two-dimensional function $f = f(a,b)$, where $0 \leq a, b \leq 1$.

Thus, the transformed version of the signal, y_g, is also defined $\forall(a,b)$ for any transformation $g_{\gamma,\xi}$. However, the graph filter operations are defined only for such (a,b) that correspond to the graph nodes, therefore we cannot directly apply $g_{\gamma,\xi}$ to it if this condition is not satisfied. To overcome this problem we approximate γ and ξ with $\bar{\gamma}, \bar{\xi}$ that satisfy the following conditions:

$$\bar{\gamma} : \begin{cases} \bar{\gamma} = \frac{k\pi}{2},\ k \in \mathbb{Z}; \\ ||\bar{\gamma} - \gamma|| \leq \frac{\pi}{4}, \end{cases} \qquad \bar{\xi} : \begin{cases} \bar{\xi} = k,\ k \in \mathbb{Z}; \\ ||\bar{\xi} - \xi|| \leq \frac{1}{2}, \end{cases} \tag{3.24}$$

and introduce the graph isometric transformation $\bar{g}_{\gamma,\xi} = g_{\bar{\gamma},\bar{\xi}}$, which we refer to as *closest* graph isometric transformation of the original general transformation $g_{\gamma,\xi}$. As discussed in Section 3.4.1, such transformation is defined only on graph nodes, which sample signal with steps Δa and Δb. Finally, we introduce the quasi-equivariance property as follows:

Definition 2. *The quasi-equivariance of the operator $\mathscr{F}$ with respect to the isometric transformation $g_{\gamma,\xi}$ is defined as a small absolute value of the difference between signals $\mathscr{F}(g_{\gamma,\xi}(y))$ and the graph isometric transformation $\bar{g}_{\gamma,\xi}(\mathscr{F}(y))$ computed at node v in G. Equivalently, we have:*

$$\left|\mathscr{F}\big(g_{\gamma,\xi}(y)\big)(v) - \bar{g}_{\gamma,\xi}\big(\mathscr{F}(y)(v)\big)\right| \leq \epsilon, \forall v, \tag{3.25}$$

where ϵ is a small value, when $\mathscr{F}$ is a quasi-equivariant.

In order to prove the quasi-equivariance of the polynomial filters $\mathscr{F}$, defined in Eq. (3.2), with respect to general isometric transformations we show that the response of $\mathscr{F}$ is quasi-equivariant with respect to each of the basic operations, defined in the beginning of Section 3.4.2. In Section 3.4.1 we have shown that polynomial filters are equivariant to any graph isometric transformation, which also means that the quasi-equivariance property is satisfied. Therefore, in this section we focus on proving the quasi-equivariance of $\mathscr{F}$ with respect to arbitrary image rotations and image translations by a real number of pixels.

Rotation

In order to prove the quasi-equivariance of polynomial filters to rotation, we first observe that for every pixel location p in an image, any random image rotation around an arbitrary point can be decomposed into a set of two translations and one rotation around p. Therefore, in this section we focus on proving the quasi-equivariance of $\mathscr{F}$ with respect to rotation around the vertex $v \in G$, where $\mathscr{F}$ is applied. Then in Section 3.4.2 we discuss in more details the quasi-equivariance of $\mathscr{F}$ with respect to an arbitrary image translation. Further, without loss of generality, we consider the rotation angle γ to be in the range $[-\frac{\pi}{4}, \frac{\pi}{4})$, due to the fact that any random rotation can be decomposed into an integer number of rotations by $\pi/2$ and a rotation by $\gamma \in [-\frac{\pi}{4}, \frac{\pi}{4})$. As illustrated in Section 3.4.1, polynomial filters are equivariant with respect to rotations by $\pi/2$, thus we only need to prove their equivariance with respect to $\gamma \in [-\frac{\pi}{4}, \frac{\pi}{4})$.

Let $y_g = g_\gamma(y)$ denote the rotated version of the signal y by $\gamma \in [-\frac{\pi}{4}, \frac{\pi}{4})$ around a vertex v_r, where a polynomial filter $\mathscr{F}$ is applied. In the remainder of this section we show that under some assumptions on the second derivative of the signal y, the Eq. (3.25) is valid for any polynomial filter $\mathscr{F}$ and for any γ that defines the image rotation g_γ. Let us denote the equivariance gap by

$$\Delta_{\mathscr{F}} \underset{df}{=} \left|\mathscr{F}\big(g_\gamma(y)\big)(v) - \bar{g}_\gamma\big(\mathscr{F}(y)(v)\big)\right|, \tag{3.26}$$

where $\bar{g}_\gamma$ is the graph isometric transformation that is closest to g. Due to the fact that $\mathscr{F}$ is an isotropic filter and as we are considering rotations $\gamma \in [-\frac{\pi}{4}, \frac{\pi}{4})$ around vertex v_r, where $\mathscr{F}$ is applied, we obtain the following equality $\bar{g}_\gamma\big(\mathscr{F}(y)(v)\big) \equiv \mathscr{F}(y)(v)$ and rewrite Eq. (3.26) as

$$\Delta_{\mathscr{F}} \underset{df}{=} \left|\mathscr{F}\big(g_\gamma(y)\big)(v) - \mathscr{F}(y)(v)\right|, \forall v. \tag{3.27}$$

We now have the following theorem.

Theorem 2. *The absolute value of the difference $\Delta_{\mathscr{F}}$ between signals $\mathscr{F}(g_\gamma(y))$ and $\mathscr{F}(y)$ satis-*

fies the following inequality:

$$\Delta_{\mathscr{F}}(v) \leq \left(\sum_{k=1}^{M} |\alpha_k| 2^{k-3}\right) \left|(1-\sin\gamma-\cos\gamma)\bar{\mathcal{I}}\right|, \tag{3.28}$$

where M is the degree of $\mathscr{F} = \sum_{k=0}^{M} \alpha_k \mathscr{L}^k y$ *and* $\bar{\mathcal{I}}$ *depends on the second derivative of the image signal* $y \underset{df}{=} f(a,b)$ *as follows:*

$$\bar{\mathcal{I}} = \max_{0\leq a,b\leq 1} \left|\begin{bmatrix}\partial_a^2 f(a,b)\\ \partial_b^2 f(a,b)\end{bmatrix}\right|^{\mathsf{T}} \begin{bmatrix}\Delta a^2\\ \Delta b^2\end{bmatrix} + o(\Delta a^2) + o(\Delta b^2). \tag{3.29}$$

Here $\Delta a, \Delta b$ *are the distances between the graph nodes in horizontal and vertical directions respectively which corresponds to the resolution of the image; and* $o(\cdot)$ *is the "little-o" notation that describes function's asymptotic behavior [56].*

Proof of Theorem 2. We start the proof with the observation that any polynomial filter $\mathscr{F}$ of degree N is a sum of the *trivial* polynomial filters $\hat{\mathscr{F}}_k$ of degrees $k = [0..N]$, defined as:

$$\hat{\mathscr{F}}_k \underset{df}{=} \alpha_k \mathscr{L}^k y. \tag{3.30}$$

Therefore to prove Eq. (3.28), we need to show that

$$\Delta_{\hat{\mathscr{F}}_k}(v) \leq 2^{k-3} \left|\alpha_k(1-\sin\gamma-\cos\gamma)\bar{\mathcal{I}}\right|, \forall k, \tag{3.31}$$

where $\Delta_{\hat{\mathscr{F}}^k}(v)$ is the absolute difference between $\hat{\mathscr{F}}^k(y_g)$ and $\hat{\mathscr{F}}^k(y)$. Then $\Delta_{\mathscr{F}}(v)$ can be eventually approximated using the triangle inequality as

$$\Delta_{\mathscr{F}}(v) \leq \sum_{k=0}^{N} \Delta_{\hat{\mathscr{F}}^k}(v) \tag{3.32}$$

To prove Eq. (3.31) we use the induction method.

Base case: We first show that for any $\hat{\mathscr{F}}$ of degree 0 and 1 Eq. (3.31) is valid. In the case of a polynomial filter of a 0 degree, $\Delta_{\hat{\mathscr{F}}^0} = 0$, as

$$g_\gamma\left(\hat{\mathscr{F}}^0(y)\right) \underset{df}{=} g_\gamma(\alpha_0 y) = \alpha_0 g_\gamma(y) = \hat{\mathscr{F}}^0(g_\gamma(y)). \tag{3.33}$$

Then, as depicted by Eq. (3.15), for a given node $v_r \in G$, $\hat{\mathscr{F}}^1$ operates on the neighborhood of v_r, which can be represented as an image pattern, depicted by Fig. 3.6. For simplicity, we prove for the 4-nn graph case, however, the proof can be easily extended to the case of other regular graphs. For the sake of simplicity, we denote by p_j the values of y in the nodes of this pattern as follows:

$$p_j \underset{df}{=} f(a_j, b_j) = y(v_j),\ \forall v_j : j \in [2,4,5,6,8], \tag{3.34}$$

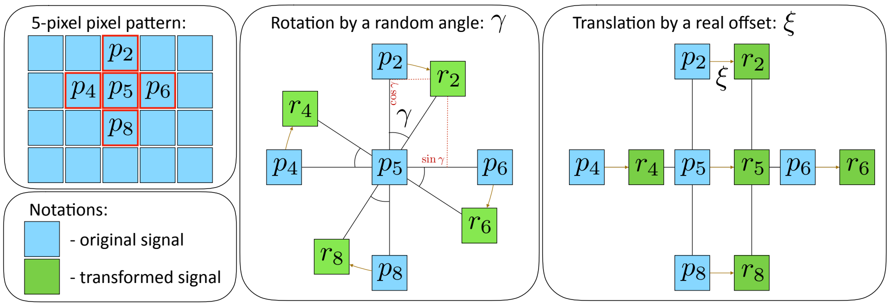


Figure 3.6 – [LEFT] Illustration of a 5-pixel image pattern that the polynomial filter $\mathscr{F}$ operates on, with $p_i, i \in [2,4,5,6,8]$ being the respective pixels intensities. [MIDDLE] The rotation from the original image signal y (blue rectangles) to the transformed image signal y_g (green rectangles), after applying the transformation g_γ around pixel p_5. The points $r_i, i \in [2,4,6,8]$ schematically show how the position of p_i change after applying g_γ. [RIGHT] The translation from the original to the transformed image signal, after applying the transformation g_ξ by a real number of pixel intensities ξ.

where (a_j, b_j) are the pixel coordinates of the node v_j. Assuming that f is differentiable (as it is generally done for natural images) we can express p_j using a Taylor approximation as

$$\begin{bmatrix} p_2 \\ p_6 \\ p_4 \\ p_8 \end{bmatrix} = \mathscr{I} p_5 - \begin{bmatrix} -\partial_b f(a_5, b_5)\Delta b \\ -\partial_a f(a_5, b_5)\Delta a \\ \partial_a f(a_5 - \Delta a, b_5)\Delta a \\ \partial_b f(a_5, b_5 - \Delta b)\Delta b \end{bmatrix} + \begin{bmatrix} R_2(b_2) \\ R_2(a_6) \\ R_2(a_4) \\ R_2(b_8) \end{bmatrix}, \tag{3.35}$$

where $\mathscr{I} = [1,1,1,1]^\intercal : \partial_a f, \partial_b f$ are the partial derivatives:

$$\partial_a f(a,b) \underset{df}{=} \frac{\partial f(a,b)}{\partial a}, \quad \partial_b f(a,b) \underset{df}{=} \frac{\partial f(a,b)}{\partial b}, \tag{3.36}$$

and $R_2(\cdot)$ is the remainder term of the Taylor expansion, which reads:

$$\begin{aligned} R_2(a_k) &= o(|a_k - a_5|^2) &&= o(\Delta a^2)\,, k \in [4,6]\,, \\ R_2(b_k) &= o(|b_k - b_5|^2) &&= o(\Delta b^2)\,, k \in [2,8]\,. \end{aligned} \tag{3.37}$$

We further denote the new node values of the rotated pattern $g_\gamma(y)$, depicted by Fig 3.6, as follows:

$$r_j \underset{df}{=} g_\gamma(y)(v_j), \; j \in [2,4,6,8]. \tag{3.38}$$

We can then approximate the values of r_j using bilinear interpolation [57] as illustrated by Fig. 3.6. This allows us to express the values r_j of each node of the rotated pattern based on the points p_j of the original pattern, which results in:

$$\begin{bmatrix} r_2 \\ r_4 \\ r_6 \\ r_8 \end{bmatrix} = \mathscr{I} r_5 - \mathscr{A} \sin\gamma - \mathscr{B} \cos\gamma + o(\Delta a^2) + o(\Delta b^2), \tag{3.39}$$

where

$$\mathscr{A} = \begin{bmatrix} -\partial_a f(a_5, b_5)\Delta a \\ -\partial_b f(a_5, b_5)\Delta b \\ \partial_b f(a_5, b_5 - \Delta b)\Delta b \\ \partial_a f(a_5 - \Delta a, b_5)\Delta a \end{bmatrix} \quad \mathscr{B} = \begin{bmatrix} -\partial_b f(a_5, b_5)\Delta b \\ \partial_a f(a_5 - \Delta a, b_5)\Delta a \\ -\partial_a f(a_5, b_5)\Delta a \\ \partial_b f(a_5, b_5 - \Delta b)\Delta b \end{bmatrix} \tag{3.40}$$

Given the introduced notations we can compute the response of a *trivial* filter $\hat{\mathscr{F}}^1$ at the graph node v_5 as:

$$\hat{\mathscr{F}}^1(y)(v_5) = -\frac{\alpha_1}{4}\mathcal{Z}(v_5)\,, \tag{3.41}$$

where $\mathcal{Z}$ reflects the smoothness of the image signal y and reads:

$$\begin{aligned} \mathcal{Z}(v_5) = &\begin{bmatrix} \partial_a f(a_5, b_5) - \partial_a f(a_5 - \Delta a, b_5) \\ \partial_b f(a_5, b_5) - \partial_b f(a_5, b_5 - \Delta b) \end{bmatrix}^\intercal \begin{bmatrix} \Delta a \\ \Delta b \end{bmatrix} \\ &+ o(\Delta a^2) + o(\Delta b^2)\,. \end{aligned} \tag{3.42}$$

Similarly, based on Eqs. (3.39) and (3.40) we can compute the response of the same filter $\hat{\mathscr{F}}^1$ applied to the rotated signal $y_g = g_\gamma(y)$ at the node v_5 as:

$$\hat{\mathscr{F}}^1(y_g)(v_5) = -\frac{\alpha_1}{4}(\sin\gamma + \cos\gamma)\mathcal{Z}(v_5) \tag{3.43}$$

We can then calculate the difference between the filter responses presented in Eqs. (3.41) and (3.43) as follows:

$$\Delta_{\hat{\mathscr{F}}^1}(v_5) = \frac{\alpha_1}{4}(1 - \sin\gamma - \cos\gamma)\mathcal{Z}(v_5). \tag{3.44}$$

In order to compute $\Delta_{\hat{\mathscr{F}}^1}$ in a general case (that is for an arbitrary rotation γ), we can rewrite $\mathcal{Z}$ as:

$$\mathcal{Z}(v_5) = \begin{bmatrix} \partial_a^2 f(a_5, b_5) \\ \partial_b^2 f(a_5, b_5) \end{bmatrix}^\intercal \begin{bmatrix} \Delta a^2 \\ \Delta b^2 \end{bmatrix} + o(\Delta a^2) + o(\Delta b^2), \tag{3.45}$$

which for any node v in G has its upper bound $\bar{\mathcal{Z}}$:

$$|\mathcal{Z}(v)| \le \bar{\mathcal{Z}} = \max_{0 \le a, b \le 1} \left| \begin{bmatrix} \partial_a^2 f(a, b) \\ \partial_b^2 f(a, b) \end{bmatrix} \right|^\intercal \begin{bmatrix} \Delta a^2 \\ \Delta b^2 \end{bmatrix} + o(\Delta a^2) + o(\Delta b^2)\,. \tag{3.46}$$

Based on Eqs. (3.44) and (3.46) we obtain the following condition:

$$\Delta_{\hat{\mathscr{F}}^1}(v) \leq 2^{-2}\left|\alpha_1(1-\sin\gamma-\cos\gamma)\bar{\mathscr{Z}}\right|, \forall v \in G, \tag{3.47}$$

for any *trivial* polynomial filter $\mathscr{F}$ of degree 1 and any image rotation g_γ, which concludes the proof of the base case.

Inductive step: We first denote the polynomial filter of degree M by $\hat{\mathscr{F}}^M = \alpha_M \mathscr{L}^M y$. Then we assume that Eq. (3.31) is satisfied for a filter $\hat{\mathscr{F}}^{M-1} = \alpha_M L^{M-1} y$ of degree $M-1$ for every node v in graph G, which brings us to the following inequality:

$$\Delta_{\hat{\mathscr{F}}^{M-1}}(v) < 2^{(M-1)-3}\left|\alpha_M(1-\sin\gamma-\cos\gamma)\bar{\mathscr{Z}}\right|, \tag{3.48}$$

where $\Delta_{\hat{\mathscr{F}}^{M-1}}$ is the difference between the responses of filter $\hat{\mathscr{F}}^{M-1}$ applied to the original and transformed signals. For the sake of simplicity we denote the right hand side of Eq. (3.48) by ϵ_γ. Then our goal is to prove that

$$\left|\Delta_{\hat{\mathscr{F}}^M}(v)\right| \leq 2\epsilon_\gamma, \quad \forall v \in G. \tag{3.49}$$

To do so, we use the same representation of the filter $\hat{\mathscr{F}}^M$, as in Eq. (3.21), which allows us to rewrite $\Delta_{\hat{\mathscr{F}}^M}$ as:

$$\Delta_{\hat{\mathscr{F}}^M} = \mathscr{L}\hat{\mathscr{F}}^{M-1}(y_g) - g_\gamma(\mathscr{L}\hat{\mathscr{F}}^{M-1}(y)), \tag{3.50}$$

where $\mathscr{L}$ is the Laplacian matrix. Further, due to the linearity of the isometric transformation g_γ, we can rewrite $\Delta_{\hat{\mathscr{F}}^M}$ as

$$\Delta_{\hat{\mathscr{F}}^M} = \mathscr{L}\left(\hat{\mathscr{F}}^{M-1}(y_g) - g_\gamma(\hat{\mathscr{F}}^{M-1}(y))\right) \underset{df}{=} \mathscr{L}\left(\Delta_{\hat{\mathscr{F}}^{M-1}}\right), \tag{3.51}$$

that can be expressed as

$$\Delta_{\hat{\mathscr{F}}^M}(v_j) = \Delta_{\hat{\mathscr{F}}^{M-1}}(v_j) + \frac{1}{4}\sum_{i:v_i\sim v_j}\Delta_{\hat{\mathscr{F}}^{M-1}}(v_i), \ \forall v_j \in G. \tag{3.52}$$

We can then apply triangle inequality to Eq. (3.52) and write:

$$\begin{aligned}\Delta_{\hat{\mathscr{F}}^M}(v) &\leq \max_{v\in G}\Delta_{\hat{\mathscr{F}}^M}(v) \leq \\ &\leq \max_{v\in G}\left(\Delta_{\hat{\mathscr{F}}^{M-1}}(v) + \frac{1}{4}\sum_{i:v_i\sim v}\Delta_{\hat{\mathscr{F}}^{M-1}}(v_i)\right)\end{aligned} \tag{3.53}$$

As every node v has at most 4 neighbors and given Eq. (3.48): $|\Delta_{\hat{\mathscr{F}}^{M-1}}(v)| \leq \epsilon_\gamma, \forall v \in G$, we can rewrite Eq. (3.53) as:

$$\Delta_{\hat{\mathscr{F}}^M}(v) \leq \max_{v\in G}\Delta_{\hat{\mathscr{F}}^M}(v) = \epsilon_\gamma + \frac{1}{4}(4\epsilon_\gamma) = 2\epsilon_\gamma. \tag{3.54}$$

Based on Eqs. (3.48) and (3.54), we can then write the general condition on $\Delta_{\hat{\mathcal{F}}^M}(v)$ for a *trivial* polynomial filter of any degree M, for every node v in G as follows:

$$\Delta_{\hat{\mathcal{F}}^M}(v) \leq 2^{M-3} \left|\alpha_M(1-\cos\gamma-\sin\gamma)\bar{\mathcal{Z}}\right| , \tag{3.55}$$

which is identical to Eq. (3.31). We then obtain Eq. (3.28) by simply combining Eq. (3.55) with Eq. (3.32), which concludes the proof of the theorem. □

It is worth noting that in the special case of γ being equal to one of the following values: $[0, \pi/2, \pi, 3\pi/2]$, the general isometric transformation g_γ becomes a graph isometric transformation, which is thoroughly discussed in Section 3.4.1. This essentially means that for any signal y the difference between filter responses is 0. As we can see from Eq. (3.28), this is indeed true, as $\Delta_{\mathcal{F}} = 0$ independently of $\bar{\mathcal{Z}}$.

Translation

We now prove the quasi-equivariance of $\mathcal{F}$ with respect to an arbitrary image translation g_ξ. We start with a simple observation that any arbitrary translation of an image signal by a real number of pixels can be decomposed into a graph isometric one (i.e translation by an integer number of pixels) and an image translation by ξ that is less than 1 pixel. As illustrated in Section 3.4.1, polynomial filters $\mathcal{F}$ are equivariant with respect to the graph isometric translation, therefore, in this section we prove their quasi-equivariance with respect to translation by $\xi \in [-\frac{1}{2}, \frac{1}{2})$ pixels.

Similarly to Section 3.4.2, let $y_g = g_\xi(y)$ denote the translated version of the signal y by $\xi \in [-\frac{1}{2}, \frac{1}{2})$. Let us then denote the equivariance gap by

$$\Delta_{\mathcal{F}} \underset{df}{=} \left|\mathcal{F}\big(g_\xi(y)\big)(v) - \bar{g}_\xi\big(\mathcal{F}(y)(v)\big)\right|, \tag{3.56}$$

where $\bar{g}_\xi$ is graph isometric transformation that is the closest to g. Due to the fact that $\mathcal{F}$ is an isotropic filter and as we are considering rotation $\gamma \in [-\frac{\pi}{4}, \frac{\pi}{4}]$ around vertex v_r, where $\mathcal{F}$ is applied], we obtain the following equality $\bar{g}_\xi\big(\mathcal{F}(y)(v)\big) \equiv \mathcal{F}(y)(v)$ and rewrite Eq. (3.56) as

$$\Delta_{\mathcal{F}} \underset{df}{=} \left|\mathcal{F}\big(g_\xi(y)\big)(v) - \mathcal{F}(y)(v)\right|, \forall v. \tag{3.57}$$

and prove the following theorem.

Theorem 3. *The absolute value of the difference $\Delta_{\mathcal{F}}(v)$ between signals $\mathcal{F}(g_\xi(y))$ and $\mathcal{F}(y)$ for any node v defined on the graph G, satisfies the following inequality:*

$$\Delta_{\mathcal{F}}(v) \leq \left(\sum_{k=1}^{M} |a_k| 2^{k-3}\right) \left|\bar{\mathcal{Z}} + o(\Delta a^2) + o(\Delta b^2)\right| , \tag{3.58}$$

where M is the degree of $\mathcal{F} = \sum_{k=0}^{M} \alpha_k \mathcal{L}^k y$ and $\bar{\mathcal{Z}}$ depends on the third partial derivatives of *the image signal* $y \underset{df}{=} f(a,b)$ *as*

$$\bar{\mathscr{I}} = \max_{0\le a,b\le 1} \left| \begin{bmatrix} \partial_a^3 f(a,b) \\ \partial_b^3 f(a,b) \\ \partial_a \partial_b^2 f(a,b) \\ \partial_b \partial_a^2 f(a,b) \end{bmatrix} \right|^{\top} \begin{bmatrix} \Delta a^3 \\ \Delta b^3 \\ \Delta b^2 \Delta a \\ \Delta a^2 \Delta b \end{bmatrix} \tag{3.59}$$

with $\Delta a, \Delta b$ *being the distances between the graph nodes in horizontal and vertical directions of the 2D image, respectively.*

Proof of Theorem 3. We follow similar steps as in the proof of Theorem 2 and show that, for each *trivial* filter $\hat{\mathscr{F}}^k = \alpha_k \mathscr{L}^k y$, the following relation is valid:

$$\Delta_{\hat{\mathscr{F}}^k} = 2^{k-3} \left| \alpha_k (\bar{\mathscr{I}} + o(\Delta a^2) + o(\Delta b^2)) \right| . \tag{3.60}$$

Here $\Delta_{\hat{\mathscr{F}}^k}$ is the difference between responses of $\hat{\mathscr{F}}^k = \alpha_k \mathscr{L}^k y$ applied to the original signal y and the transformed version y_g, and $\bar{\mathscr{I}}$ is defined in Eq. (3.59). Then, based on Eq. (3.60) and triangle inequality, we can show that Eq. (3.58) is valid for any polynomial filter $\mathscr{F}$ that can be written as a sum of trivial filters.

Similarly to Theorem 2, in order to prove Eq. (3.60), we use the induction method. Further, we use the fact that any random translation g_ξ can be represented as a sum of the translations in horizontal and vertical directions.

Base case: We need to prove that Eq. (3.60) is valid for any filter $\hat{\mathscr{F}}$ of degree 0 and 1. The proof of the following equality:

$$g_\gamma \left(\hat{\mathscr{F}}^0(y) \right) \underset{df}{=} g_\gamma(\alpha_0 y) = \alpha_0 g_\gamma(y) = \hat{\mathscr{F}}^0(g_\gamma(y)), \tag{3.61}$$

is identical to the proof of Theorem 2, which validates the base case for the *trivial* filter of degree 0. Then we need to show that Eq. (3.60) is valid for any *trivial* filter $\hat{\mathscr{F}}^1$ of degree 1. To do so, we first compute the response of $\hat{\mathscr{F}}^1(y) = \alpha_1 \mathscr{L} y$ at a random vertex $v_5 \in G$ with the coordinates (a_5, b_5) (see Fig. 3.6 (right)) as:

$$\hat{\mathscr{F}}^1(y)(v_5) = \alpha_1 \left(f(a_5, b_5) - \frac{1}{4} \sum_{i=[2,4,6,8]} f(a_i, b_i) \right) . \tag{3.62}$$

We then apply the same $\hat{\mathscr{F}}^1$ to the transformed signal y_g that is a version of the signal y, shifted horizontally by ξ pixels. Using a Taylor expansion, we obtain the following:

$$
\begin{aligned}
\hat{\mathscr{F}}^1(y_g)(v_5) = \alpha_1 &\left(f(a_5,b_5) - \frac{1}{4} \sum_{i=[2,4,6,8]} f(a_i,b_i) + o(\Delta a^2) \right) + \\
+ \frac{\alpha_1}{4} &\begin{bmatrix} \partial_a f(a_5,b_5) \\ \partial_a f(a_5-\Delta a,b_5) \\ \partial_a f(a_5+\Delta a,b_5) \\ \partial_a f(a_5,b_5+\Delta b) \\ \partial_a f(a_5,b_5-\Delta b) \end{bmatrix}^\top \begin{bmatrix} 4 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \Delta a \,.
\end{aligned}
\tag{3.63}
$$

Equipped with Eqs. (3.62) and (3.63) we compute the difference between filter responses $\Delta_{\hat{\mathscr{F}}^1}$ at node v_5 in G and obtain

$$
\Delta_{\hat{\mathscr{F}}^1}(v_5) = \frac{\alpha_1}{4} \begin{bmatrix} \partial_a f(a_5,b_5) \\ \partial_a f(a_5-\Delta a,b_5) \\ \partial_a f(a_5+\Delta a,b_5) \\ \partial_a f(a_5,b_5+\Delta b) \\ \partial_a f(a_5,b_5-\Delta b) \end{bmatrix}^\top \begin{bmatrix} 4 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \Delta a + o(\Delta a^2)\,.
\tag{3.64}
$$

After applying matrix multiplication and Taylor expansion, Eq. (3.64) boils down to

$$
\Delta_{\hat{\mathscr{F}}^1}(v_5) = \frac{\alpha_1}{4} \begin{bmatrix} \partial_a^3 f(a_5,b_5)\Delta a \\ \partial_a \partial_b^2 f(a_5,b_5)\Delta a \end{bmatrix}^\top \begin{bmatrix} \Delta a^2 \\ \Delta b^2 \end{bmatrix} + o(\Delta a^2) + o(\Delta b^2)\,,
\tag{3.65}
$$

which permits to write the following upper bound for all nodes v in the graph G:

$$
\begin{aligned}
&\Delta_{\hat{\mathscr{F}}^1}(v) \le \max_{v\in G} \Delta_{\mathscr{F}}(v) = o(\Delta a^2) + o(\Delta b^2) + \\
&+ \max_{0\le a,b\le 1} \left| \frac{\alpha_1}{4} \begin{bmatrix} \partial_a^3 f(a,b)\Delta a \\ \partial_a \partial_b^2 f(a,b)\Delta a \end{bmatrix} \right|^\top \begin{bmatrix} \Delta a^2 \\ \Delta b^2 \end{bmatrix}.
\end{aligned}
\tag{3.66}
$$

This concludes the derivation of the upper bound on $\left|\Delta_{\hat{\mathscr{F}}^1}(v)\right|$ for an arbitrary horizontal translation g_ξ. We then perform the exact same steps for the vertical image translation and obtain:

$$
\begin{aligned}
&\Delta_{\hat{\mathscr{F}}^1}(v) \le \max_{v\in G} \Delta_{\mathscr{F}}(v) = o(\Delta a^2) + o(\Delta b^2) + \\
&+ \max_{0\le a,b\le 1} \left| \frac{\alpha_1}{4} \begin{bmatrix} \partial_b^3 f(a,b)\Delta b \\ \partial_b \partial_a^2 f(a,b)\Delta b \end{bmatrix} \right|^\top \begin{bmatrix} \Delta b^2 \\ \Delta a^2 \end{bmatrix}.
\end{aligned}
\tag{3.67}
$$

Finally, for an arbitrary image translation we add Eqs. (3.66) and (3.67) and obtain Eq. (3.60), which proves the base case of the theorem.

Inductive step: To prove the inductive step of the theorem we need to show that for any filter $\hat{\mathscr{F}}^M$ of degree M and every node v in G the following holds:

$$
\Delta_{\hat{\mathscr{F}}^M}(v) \le 2\epsilon_\xi\,,
\tag{3.68}
$$

under the assumption that for a filter $\hat{\mathscr{F}}^{M-1} = \alpha_M \mathscr{L}^{M-1} y$ of degree $M-1$, the following is valid:

$$\Delta_{\hat{\mathscr{F}}^{M-1}}(v) \leq \epsilon_\xi,\ \epsilon_\xi = 2^{k-3} \left| \alpha_M (\bar{\mathscr{I}} + o(\Delta a^2) + o(\Delta b^2)) \right|. \tag{3.69}$$

The proof of this fact is identical to the inductive step of the Theorem 2, which concludes the proof of this theorem. □

As we can see from Theorem 3 polynomial filter quasi-equivariant to arbitrary image translation. Further, Eqs. (3.66) and (3.67) show that the equivariance gap depends on the image properties, such as the maximum value of the third derivative of the input signal y, which is directly related to the smoothness of y.

Discussion

The Theorems 2 and 3 permit to obtain the following result.

Lemma 1. *For a high resolution image signals y and y_g (i.e., the distances between graph nodes $\Delta a, \Delta b$ are small) the difference between the responses of the polynomial filters $\Delta_{\mathscr{F}}$, defined in Eqs. (3.28) and (3.58) is smaller than (or equal to) the difference of the responses of the same filter $\mathscr{F}$, applied to the lower resolution versions of the same images y and y_g.*

The proof of this naturally follows from the Theorems 2 and 3. Based on these theorems, we can derive a formal condition on the resolution of the image signal y that guarantees that the response of any polynomial filter $\mathscr{F}$ of a given degree is quasi-equivariant to random image rotations.

Lemma 2. *For a polynomial filter $\mathscr{F}$ of a given degree there exist positive values Δa and Δb, which define the resolution of an image signal $y = f(a,b)$, such that the equivariance gap $\Delta_{\mathscr{F}}$ between $\mathscr{F}(y)$ and its rotated version $\mathscr{F}(y_g)$ is lower then a predefined constant ϵ_{def}.*

Proof of Lemma 2. Our goal is to derive an upper bound on the values of Δa and Δb such that $\Delta_{\mathscr{F}}$ is lower then a predefined constant $\epsilon_{\text{def}} > 0$. The values Δa and Δb are directly related to the resolution of the image y as they define the distance between the neighboring nodes of the graph G, which is small for high resolution images and large for the low resolution ones. Here we show the derivation of the upper bounds on Δa and Δb for a polynomial filter $\mathscr{F}$ of degree 1. Similar proof can be done for the polynomial filters of any arbitrary degree.

Let us introduce the following notations:

$$\begin{aligned} \epsilon_a &\underset{df}{=} \left(\partial_a^2 f(a_5, b_5)\Delta a^2 + o(\Delta a^2)\right)(1 - \sin\gamma - \cos\gamma), \\ \epsilon_b &\underset{df}{=} \left(\partial_b^2 f(a_5, b_5)\Delta b^2 + o(\Delta b^2)\right)(1 - \sin\gamma - \cos\gamma). \end{aligned} \tag{3.70}$$

Based on Eqs. (3.44) and (3.29) we can represent the equivariance gap $\Delta_{\mathscr{F}}$ as

$$\Delta_{\mathscr{F}} \underset{df}{=} |\epsilon_a + \epsilon_b| \le |\epsilon_a| + |\epsilon_b| \le \frac{\epsilon_{\text{def}}}{2} + \frac{\epsilon_{\text{def}}}{2} = \epsilon_{\text{def}}\,. \tag{3.71}$$

Then, we can express the bounds on the distances between the adjacent grid nodes in horizontal and vertical directions as

$$\begin{aligned} \Delta a &\le \sqrt{\frac{\epsilon_{\text{def}}}{2} \left| \left(\partial_a^2 f + o(1)\right) (1 - \sin\gamma - \cos\gamma) \right|^{-1}}\,, \\ \Delta b &\le \sqrt{\frac{\epsilon_{\text{def}}}{2} \left| \left(\partial_b^2 f + o(1)\right) (1 - \sin\gamma - \cos\gamma) \right|^{-1}}\,. \end{aligned} \tag{3.72}$$

Therefore, for the image resolution defined by Δa, Δb from Eq. (3.72) the equivariance gap $\Delta\mathscr{F}$ is lower or equal than ϵ_{def}. □

To sum up, we have shown that our polynomial filters are quasi-equivariant to random image rotations and translations. Additionally, as shown in Section 3.4.1 our polynomial filter $\mathscr{F}$ is equivariant to reflection transformation, which is an even stronger property than quasi-equivariance. This, altogether, proves that our polynomial filters are quasi-equivariant to any isometric transformation as the latter can be represented as the combination of rotation, translation and reflection.

3.5 Experiments

In this section we analyze the results and compare our network to the state-of-the-art transformation-invariant classification algorithms. We first describe the experimental settings. We then analyze our architecture and the influence of the different design parameters. Further, we show that equivariance gap $\Delta_{\mathscr{F}}$ is lower for higher resolution images which confirms our theoretical results. Finally we compare our network to the state-of-the-art transformation-invariant classification algorithms.

3.5.1 Experimental settings

The initialization of the system may have some influence on the actual values of the parameters after training. We have chosen to initialize the parameters $\alpha_{i,m}^{l}$ (Eq. 3.2) of our spectral convolutional filters so that the different filters uniformly cover the full spectral domain. We first create a set of Z overlapping rectangular functions $z(\mu, a_i, b_i)$

$$z(\mu, a_i, b_i) = \begin{cases} 1 & \text{if } a_i < \mu < b_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3.73}$$

Method	Architecture
Experiments on MNIST-012	
ConvNet [37]	C[3]-P[2]-C[6]-P[2]-FC[50]-FC[30]-FC[10]
STN [38]	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 [37]	C[10]-P[2]-C[20]-P[2]-FC[500]-FC[300]-FC[100]
STN [38]	C[10]-ST[6]-C[20]-ST[6]-FC[500]-FC[300]-FC[100]
DeepScat [46]	W[2, 5]-PCA[20]
HarmNet [48]	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 3.1 – Architectures used for the experiments.

The non-zero regions for all functions have the same size, and the set of functions covers the full spectrum of the normalized laplacian $\mathscr{L}$, i.e., $[0,2]$. We finally approximate each of these rectangular functions by a M-order polynomial, which produces a set of initial coefficients $\alpha^l_{i,m}$ that are used to define the initial version of the spectral filter $\mathscr{F}^l_i$. Then, the initial values of the parameters β in the spectral convolutional layer are distributed uniformly in $[0,1]$ and those of the parameters in the fully-connected layers are selected uniformly in $[-1,1]$.

We run experiments with different numbers of layers and parameters. For each architecture, the network is trained using back-propagation with Adam [53] optimization. The exact formulas of the partial derivatives 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 [58]. 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 [58] 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

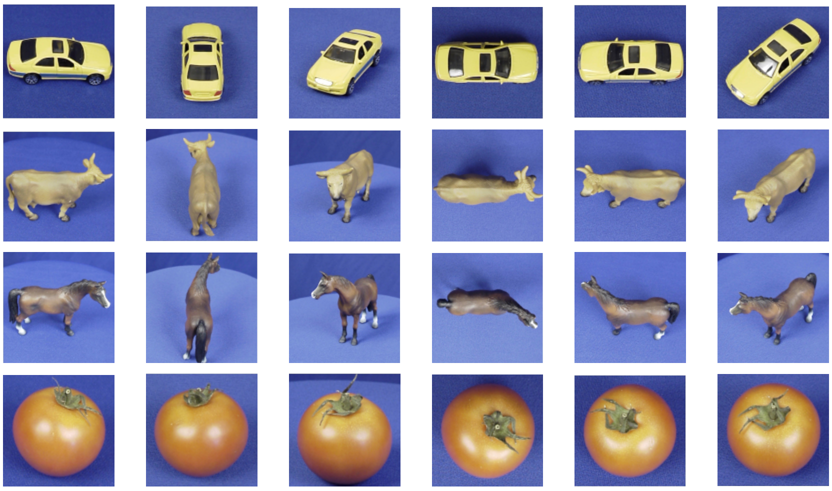


Figure 3.7 – Sample images from ETH-80 dataset.

MNIST-trans comprises randomly translated MNIST examples up to ±6 pixels in both vertical and horizontal directions.

- **ETH-80.** This dataset [59] 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 (see Fig. 3.7). 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 and 300 of them as the training and validation sets and we use the rest of the images 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.

3.5.2 TIGraNet Analysis

We analyze the performance of our new architecture on the MNIST-012 dataset. We first give some examples of feature maps that are produced by our network. We then illustrate the spectral kernels learned by our system, and discuss the influence of dynamic pooling operator.

We first confirm the transformation invariant properties of our architecture. Even though our classifier is trained on images without any transformations, it is able to correctly classify rotated images in the test set, since our spectral convolutional layer learns filters that are equivariant to isometric transformations. We illustrate this in Fig. 3.8, which depicts several examples of feature maps y_i^2 from the second spectral convolutional layer for randomly rotated input digits in the test set. Each row of Fig. 3.8 corresponds to images of a different digit, and we see that the corresponding feature maps are very close to each other (up to the image

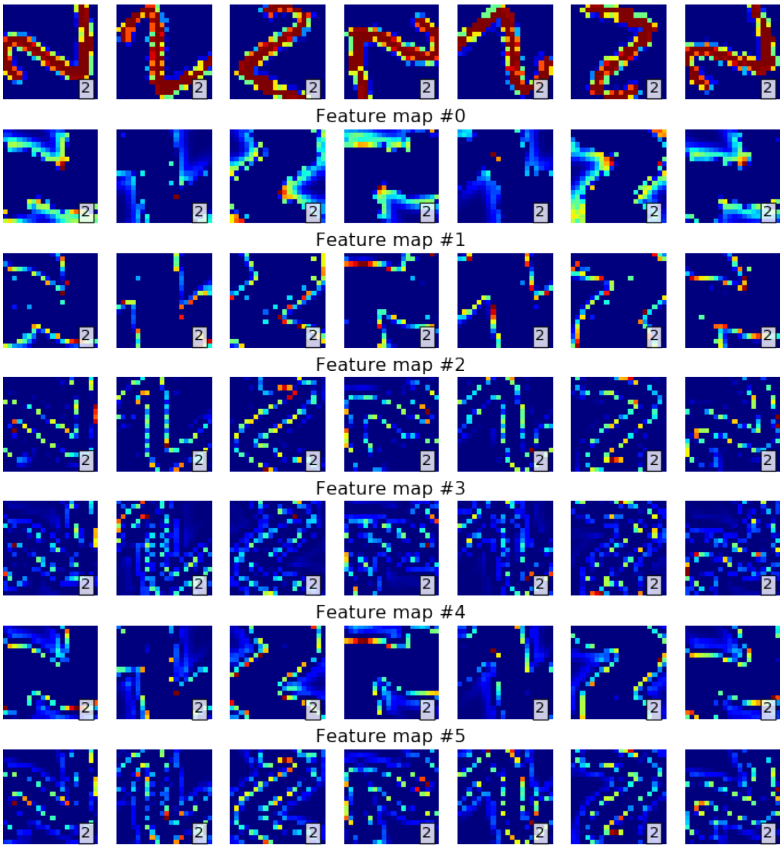


Figure 3.8 – **Feature maps** from the second spectral convolutional layer for test images that are rotated versions of an image of the digit '2'. The predicted label for each of the images is further shown in the right bottom corner of each image.

rotation) even when the rotation angle is quite large. This confirms that our architecture is able to learn features that are preserved with rotation, even if the training has been performed on non-transformed images. Despite important similarities in feature maps of rotated digits, one may however observe some slightly different values for the intensity. This can be explained by the fact that rotated versions of the input images may differ a bit from the original images due to interpolation artifacts.

Fig. 3.9 then shows the spectral representation of the kernels learned for the first two spectral convolutional layers of our network. As expected, the network learns filters that are quite different from each other in the spectral domain but that altogether cover the full spectrum. They permit to efficiently combine information in the different bands of frequency in the spectral representation of the input signal. Generally, the filters in the upper spectral convolutional layers are more diverse and represent more complicated features than those for the lower ones.

Further, we look at the influence of the new dynamic pooling layers in our architecture. Recall that dynamic pooling is used to reduce the network complexity and to focus on the

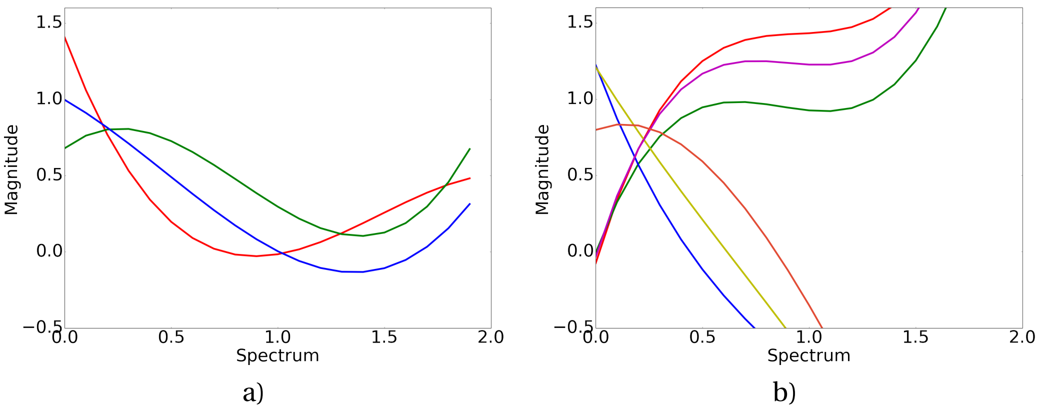


Figure 3.9 – **Sample trained filters** in the spectral domain for (a) first and (b) second convolutional layers. Different colors represent different filters on each of the layers.

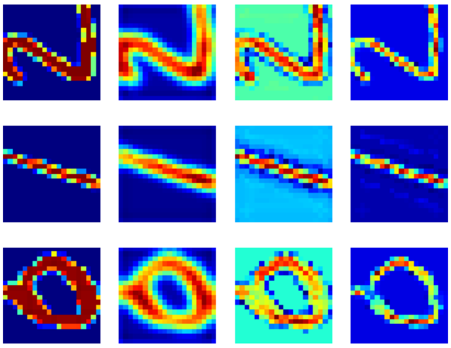


Figure 3.10 – **Feature maps after pooling** Each row shows different digits. The left most column depicts the original images, while the other columns show the features maps after dynamic pooling at the first, second and third layers respectively. The degree of the polynomial filters has been set to $M = 3$ for each layer in this experiment.

representative parts of the input signal. Fig. 3.10 depicts the intermediate feature maps of the network for sample test images. We can see that after each pooling operation the signal is getting more and more sparse, while structure of the data that is important for discriminating images in different classes is preserved. That shows that our dynamic pooling operator is able to retain the important information in the feature maps constructed by the spectral convolutional layers.

3.5.3 Influence of the image resolution on the quasi-equivariance

As we show in Section 3.4.2 the difference between filter responses of the polynomial filters $\Delta_{\mathscr{F}}$, defined in Eqs. (3.28) and (3.58) depends on the resolution of the image $\Delta a, \Delta b$. To verify our theoretical result we run the following experiment.

We down-sample 400 high resolution images from [60] using bicubic interpolation with several

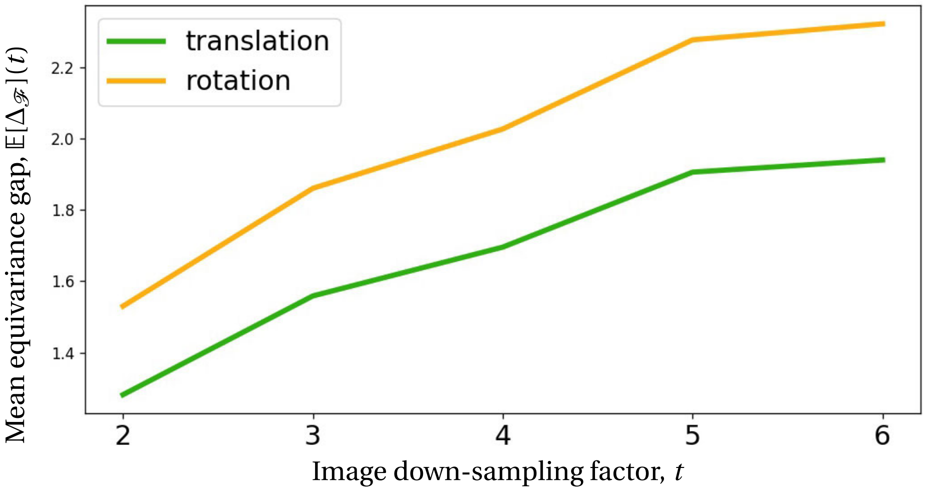


Figure 3.11 – **Influence of the image resolution on the quasi-equivariance property.** The x-axis illustrates the change in the image resolution that is defined by the down-sampling factor t of the original image. The y-axis shows the mean equivariance gap $\mathbb{E}[\Delta_{\mathscr{F}}](t)$ computed across a set of images with different transformations. The green and orange lines correspond to the set of image translations and rotations respectively. (best seen in color)

down-sampling factors $t \in \{2,3,4,5,6\}$. Thus, we obtain a set of images $y_t, t \in \{2,3,4,5,6\}$ with different resolutions.

For each of these images, we apply isometric transformation $\mathscr{T}(y_t)$ and filters $\mathscr{F}_i, i = 1,...,20$ of degree 4, with random coefficients $\alpha_i, m \in [-1,1]$. Then, we calculate the following differences

$$\mathbb{E}[\Delta_{\mathscr{F}}](t) = \frac{1}{20}\sum_{i=1}^{20} \mathscr{T}^{-1}\big(\mathscr{F}_i\big(\mathscr{T}(y_t)\big)\big) - \mathscr{F}_i(y_t), \tag{3.74}$$

where $\mathscr{T}^{-1}$ is inverse transformation. In our experiments we apply the following transformations:

- rotation by $\pi/18, \pi/9, \pi/6, \pi/4$;
- translation by $(0.1,0.1), (0.2,0.2), (0.3,0.3), (0.4,0.4)$ pixels;

and evaluate the mean equivariance gap $\mathbb{E}[\Delta_{\mathscr{F}}](t)$ across various transformations and images. Fig. 3.11 shows that $\mathbb{E}[\Delta_{\mathscr{F}}](t)$ is growing with the increase of the down-sampling factor t, or equivalently the decrease in resolution of the images. Also, we can see that translation gives smaller values of $\mathbb{E}[\Delta_{\mathscr{F}}](t), \forall t$ rather than rotation transformation, which confirms our results from Theorem 2 and 3, where it was shown that the upper bound on the equivariance gap depends on the maximum value of the second and third derivatives of the image signal y for rotation and translation transformations respectively.

3.5.4 Performance evaluation

Here, we compare TIGraNet to state-of-the art algorithms for transformation-invariant image classification tasks, i.e., ConvNet [37], Spatial Transformer Network (STN) [38], Deep Scattering (DeepScat) [46] and Harmonic Networks (HarmNet) [48]. 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. For 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 3.1, where we use the following notations to describe it: 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. Lastly, S[K_{max}] is a statistical layer with K_{max} the maximum order of Chebyshev polynomials.

The results of this first experiment are presented in Table. 3.2. 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%), due to its inherent transformation invariant characteristics. We further run experiments where a simple augmentation of the training set is implemented with randomly rotated images. 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 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 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 3.1. Table 3.3 shows that our algorithm significantly outperforms the competitor

	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	**94 ± 0.42**

Table 3.2 – Classification accuracy of ConvNet, STN and TIGraNet on MNIST-012. The methods are trained without and with transformed images. We average the performance of all the methods across 10 runs with different transformations of the test data.

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

Table 3.3 – 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.

	Accuracy (%)
STN [38]	45.1
ConvNet [37]	80.1
DeepScat [46]	87.3
HarmNet [48]	94.0
TIGraNet	**95.1**

Table 3.4 – Performance evaluation for ConvNet, STN, DeepScat, HarmNet and TIGraNet on classification of images from the ETH-80 dataset.

methods on both datasets due to its transformation invariant features.

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. 3.12, 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 the digit is shifted too close to the border due to an isometric transformation, then 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.

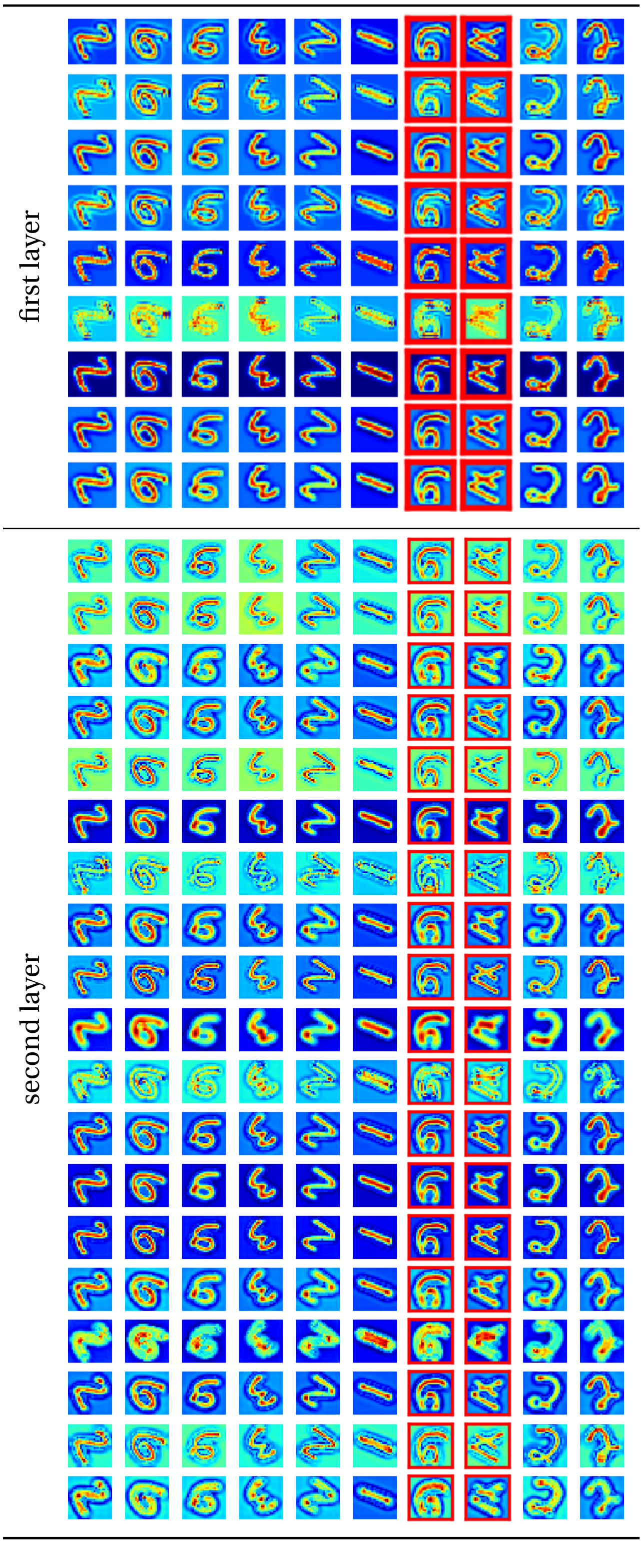


Figure 3.12 – **Network feature maps visualization.** Each row shows the feature maps of different digits after the first and the second spectral convolutional layers. The misclassified images are marked by red bounding boxes. (best seen in color)

Further, 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 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 3.1.

Table 3.4 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 the training of deep neural networks. This likely results in decrease of accuracy for methods such as ConvNets 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.

Finally, we run an additional experiment to show the influence of the amount of training data augmentation on the performance of our method. In this experiment we construct several training datasets $\mathscr{D}_i$ based on the training MNIST-rot images. To build each of these datasets $\mathscr{D}_i$ we randomly rotate its test images on one of $K_{\mathscr{D}_i}$ predefined angles. For example, $K_{\mathscr{D}_i} = 4$ means that training images are randomly rotated by $0, \pi/2, \pi, 3\pi/2$ degrees. The Fig. 3.13 shows the performance of different methods on the test dataset containing MNIST-rot images rotated on random angle with respect to the number $K_{\mathscr{D}_i}$. As we can notice, our method trained on the dataset without any transformation outperforms the convolutional network even if it contains examples of many different rotations in the training data. However, the Spatial Transformer Network outperforms our method on the dataset with $K_{\mathscr{D}_i} > 8$. It happens because our network is inherently equivariant to graph isometric transformations that preserve graph structure (see Section 3.4). Rotations on 45 degrees, however, introduces interpolation artifacts. Therefore, to be complete we also train our method on D_i, which allows our method to overcome these issues and again outperform other methods.

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 is able to reach sustained performance even if the training set is relatively small, or does not contain similar transformed images as in the test set. These are very important advantages in practice.

3.6 Conclusion

In this chapter, we have presented a new transformation invariant classification architecture, which combines the power of deep networks and graph signal processing to develop 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-

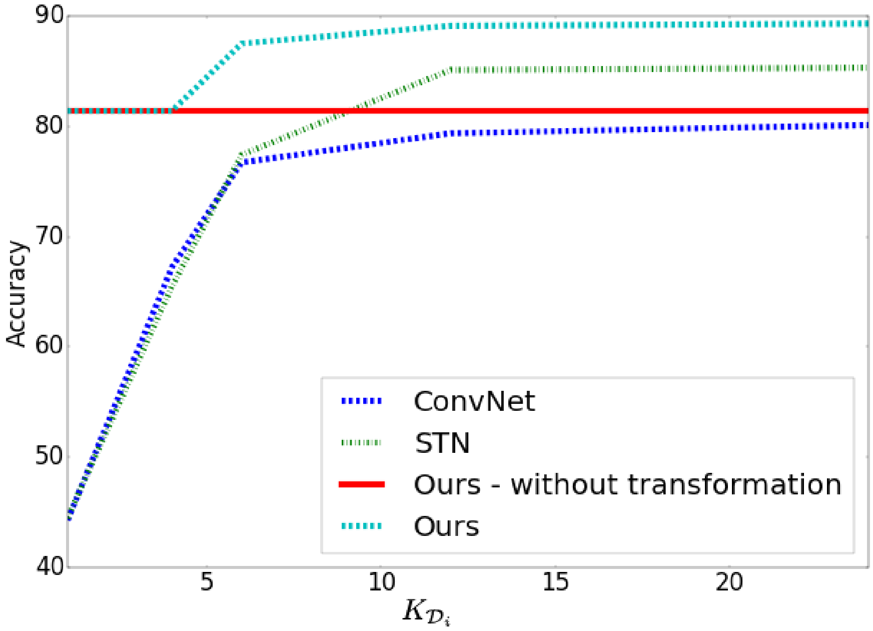


Figure 3.13 – Performance of the different approaches depending on the number of random rotations $K_{\mathscr{D}_i}$ in the training set. Each of the dataset $\mathscr{D}_i$ is build based of the MNIST-rot dataset. Red line corresponds to our method which is trained on the data without any augmentation.

the-art algorithms on various illustrative benchmarks. Our new method is able to correctly classify rotated and translated images even if such transformed images do not appear in the training set. This confirms its high potential in practical settings where the training sets are limited but where the data is expected to present high variability.

4 Graph-Based Classification of Omnidirectional Images

4.1 Introduction

In the previous chapter we showed that graphs can be efficiently used to incorporate some desired properties of the features (i.e. isometric transformation invariance) inside the deep learning framework. In this chapter we further extend graph-based representation learning to modeling a different type of prior information about the task at hand, namely the projective geometry of omnidirectional cameras.

Omnidirectional cameras are very attractive for various applications in robotics [61, 62] and computer vision [63, 64] thanks to their wide viewing angle. Despite this advantage, working with the raw images, taken by such cameras is difficult because of severe distortion effects introduced by the camera geometry or lens optics, which has a significant impact on local image statistics. Therefore, all methods that aim at solving different computer vision tasks (e.g. detection, points matching, classification) on the images from the omnidirectional cameras need to find a way of compensating for this distortion.

A 'naive', approach is to apply standard techniques directly to raw (distorted) images. However, algorithms proposed for the planar images lead to non-optimal solutions when applied to distorted images. One example of such standard techniques are the Convolutional Neural Networks (ConvNets) [37], which are primarily designed for regular domains [7] and have achieved remarkable success in various areas of computer vision [65, 66, 67]. The drawback of this solution is that ConvNets require a lot of training data for omnidirectional image classification task, as the same object will not have the same local statistics, for different image locations, which results in different filter responses. Therefore, the dataset should include images where same objects are seen in different parts of the image in order to reach invariance to distortions.

In this chapter we propose to design a solution for image classification that inherently takes into account the camera geometry. Developing such a technique based on the classic ConvNets is, however, complicated due to the two main reasons. First as we mentioned before the

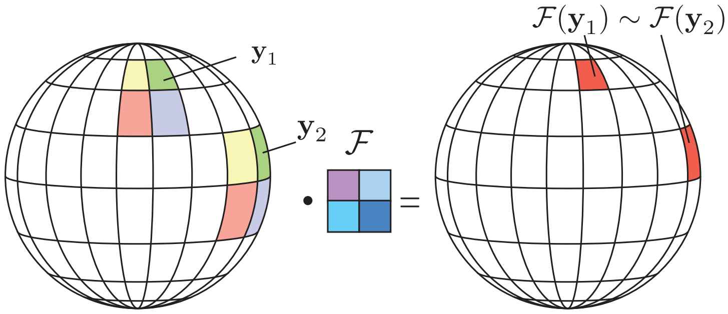


Figure 4.1 – The proposed graph construction method makes response of the filter similar regardless of different position of the pattern on an image from an omnidirectional camera.

features, extracted by the network, need to be invariant to positions of objects in the scene and different orientations with respect to the omnidirectional camera. Second, it is challenging to incorporate lens geometry knowledge in the structure of convolutional filters.

To tackle the first of the aforementioned challenges we propose to extend method from previous chapter. However, direct application of this approach is not able to take advantage of the knowledge about the geometry of omnidirectional images, as the same object seen at different positions of an omnidirectional image still remains different from the network point of view. Thus, we suggest to incorporate the knowledge about the geometry of the omnidirectional camera lens into the signal representation, namely in the structure of the graph (see Fig. 4.1). This permits filters output to remain the same regardless position of the object on the omnidirectional image. In summary, we therefore propose the following contributions:

- a principled way of graph construction based on geometry of omnidirectional images;
- graph-based deep learning architecture for the omnidirectional image classification task.

4.2 Wide field of view image processing

In this section we review several techniques designed for wide-angle cameras for different computer vision applications. We then briefly discuss the main difference of our proposed technique with the approaches discussed in Chapter 2.

A broad variety of computer vision tasks benefit from having wide-angle cameras. For example, images from fisheye [68], which can reach field of view (FOV) of more than 180°, or omnidirectional cameras, that provide 360° FOV [69, 70] are widely used in virtual reality and robotics [69, 71] applications. Despite their benefits these images are challenging to process due to the fact that most of the approaches are developed for planar images and suffer from distortion effects when applied to images captured by cameras with wide field of view [68].

There exist different ways of acquiring an omnidirectional image. First such an image can be built based on a set of multiple images, taken either by a camera that is rotated around its center of projection, or by multiple calibrated cameras. Rotating camera systems, however cannot be applied to dynamic scenes, while multi-camera systems suffer from calibration difficulties. Alternatively, one can obtain an omnidirectional image from dioptric or catadioptric cameras [72]. Most of the existing catadioptric cameras have the following mirror types: eliptic, parabolic and hyperbolic. The authors in [73] show that such mirror types allow for a bijective mapping of the lens surface to a sphere, which simplifies processing of omnidirectional images. We work with this spherical representation of catadioptric omnidirectional cameras. The analysis of images from wide-angle cameras remains however an open problem.

For example, the standard approaches for interest point matching propose affine-invariant descriptors such as SIFT [74], GIST [75]. However, designing descriptors that preserve invariance to geometric distortions for wide-angle camera's images is challenging. One of the attempts to achieve such invariance is proposed by [76], where the authors extend the GIST descriptor to omnidirectional images by exploiting their circular nature. Instead of using hand-crafted descriptors, the authors in [77] suggest to learn them from the data by creating a similarity preserving hashing function. Further, inspired by the aforementioned method, the work in [64] proposes to learn descriptors for images from omnidirectional cameras using a siamese neural network [78]. While this method is not using specific geometry of the lens, it significantly outperforms state-of-the-art as it encodes transformations that are present in the omnidirectional images. However, the method requires carefully constructed training dataset to learn all possible variations of the data.

Contrary to the previous approaches, the methods in [63, 79, 80] design a scale invariant SIFT descriptor for the wide-angle cameras based on the result of the work in [73] that introduced a bijection mapping between omnidirectional images and a spherical surface. In particular, the method in [63] maps images to a unit sphere, and those in [79] propose two SIFT-based algorithms, which work in spherical coordinates. The first approach (local spherical) matches points between two omnidirectional images, while the second one (local planar) works between spherical and planar images. Finally, the authors in [80] adapt a Harris interest point detector [81] to spherical nature of images from omnidirectional cameras. All the aforementioned works are designed for interest point matching task. In our work we use the similar idea of mapping omnidirectional images to the spherical surface [73] for omnidirectional image classification problem.

Omnidirectional cameras have also been widely used in other computer vision and robotics tasks. For example, the authors in [82] propose a segmentation method for catadioptric cameras. They derive explicit expression for edge detection and smoothing differential operators and design a new energy functional to solve segmentation problem. The work in [83] then develops a stereo-based depth estimation approach from multiple cameras. Further, the authors in [72] extend previous geometry-based calibration approach to compute depth and disparity maps from images captured by a pair of omnidirectional cameras. They also suggest an efficient way of sparse 3D scene representation. The works in [69, 70] then use omnidirectional cameras for robot self-localisation and reliable estimation of the 3D map of the environment (SLAM). Further, the authors in [84] suggest an object detection approach for omnidirectional images by modifying the traditional HOG-based approach both geometrically and by applying Riemannian metric on the sphere model. The authors in [85] propose a motion estimation method for catadioptric omnidirectional images by the evaluation of the correlation between them from arbitrary viewpoints. Finally, the work in [86] utilizes geometry of the omnidirectional camera to adapt the quantization tables of ordinary block-based transform codecs for panoramic images computed by equirectangular projection.

In summary, processing images from omnidirectional cameras becomes an important topic in the computer vision community. However, most of the existing solutions rely on methods developed for planar images. In this chapter we are particularly interested in image classification tasks and propose a solution based on a combination of powerful deep learning architecture and camera lens geometry that we encode using the graph-based representation of the input image. Therefore, our approach is also related to the methods that are designed for working with graphs that are summarized in Chapter 2. However, the main difference of our method with these approaches is that instead of designing a novel deep learning architecture that could generalize to any types of data, we focus on the development on a graph construction mechanism that embeds the knowledge of the omnidirectional camera lens geometry inside the neural network. The latter can eventually be incorporated in any graph-based deep learning architecture, as we show in this chapter on the example of [6, 14].

4.3 Graph-based representation

4.3.1 Image model

An omnidirectional image is typically represented as a spherical one (see Fig. 4.2), where each point $\mathbf{X}_k$ from 3D space is projected to the points $\mathbf{x}_k$ on the spherical surface $\mathbf{S}$ with radius r, which we set to $r = 1$ without loss of generality. The point $\mathbf{x}_k$ is then uniquely defined by its longitude $\theta_k \in [-\pi, \pi]$ and latitude $\phi_k \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and its coordinates can be written as:

$$\mathbf{x}_k : \qquad \begin{bmatrix} \cos\theta_k \cos\phi_k \\ \sin\theta_k \cos\phi_k \\ \sin\phi_k \end{bmatrix}, k \in [1..N]. \tag{4.1}$$

We consider objects on a plane that is tangent to the sphere $\mathbf{S}$. We denote by $\mathbf{X}_{k,i}$ a 3D space point on the plane $\mathbf{T}_i$ tangent to the sphere at (ϕ_i, θ_i). The point $\mathbf{X}_{k,i}$ is defined by the coordinates $(x_{k,i}, y_{k,i})$ on the plane $\mathbf{T}_i$. We, further, denote by $\mathbf{x}_{k,i} : (\phi_k, \theta_k)$ the points on the surface of the sphere that are obtained by connecting its center and the point $\mathbf{X}_{k,i}$ on $\mathbf{T}_i$. We can find coordinates of each point $\mathbf{X}_{k,i}$ on $\mathbf{T}_i$ by using the gnomonic projection, which

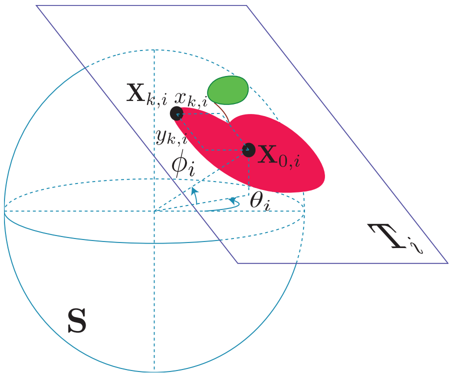


Figure 4.2 – Example of the gnomonic projection. An object from tangent plane $\mathbf{T}_i$ is projected to the sphere at tangency point $\mathbf{X}_{0,i}$, which is defined by spherical coordinates ϕ_i, θ_i. The point $\mathbf{X}_{k,i}$ is defined by coordinates $(x_{k,i}, y_{k,i})$ on the plane.

generally provides a good model for omnidirectional images [87]:

$$\begin{aligned} x_{k,i} &= \frac{\cos\phi_k \sin(\theta_k - \theta_i)}{\cos c}, \\ y_{k,i} &= \frac{\cos\phi_i \sin\phi_k - \sin\phi_i \cos\phi_k \cos(\theta_k - \theta_i)}{\cos c}, \end{aligned} \tag{4.2}$$

where c is the angular distance between the point $(x_{k,i}, y_{k,i})$ and the center of projection $\mathbf{X}_{0,i}$ and is defined as follows:

$$\begin{aligned} &\cos c = \sin\phi_i \sin\phi_k + \cos\phi_i \cos\phi_k \cos(\theta_k - \theta_i), \\ &c = \tan^{-1}\left(\sqrt{x_{k,i}^2 + y_{k,i}^2}\right). \end{aligned} \tag{4.3}$$

Fig. 4.2 illustrates an example of this gnomonic projection.

In order to easily process the signal defined on the spherical surface, it is typically projected to an equirectangular image (see Fig. 4.3). The latter represents the signal on the regular grid with step sizes $\Delta\theta$ and $\Delta\phi$ for angles θ and ϕ respectively. We work with these equirectangular images and assume that the object, which we are classifying, is lying on a plane $\mathbf{T}_i$ tangent to the sphere $\mathbf{S}$ at the point (ϕ_i, θ_i). Our work could however be adapted to other projection models, such as [88]. Finally, each point on the equirectangular image is considered as a vertex v_k in our graph representation. The graph then connects nearest neighbors of the equirectangular image $\mathbf{y}(v_i) = \mathbf{y}(\phi_i, \theta_i)$

4.3.2 Weight design

Our goal is to develop a transformation invariant system, which can recognize the same object on different planes $\mathbf{T}_i$ that are tangent to $\mathbf{S}$ at different points (ϕ_i, θ_i) without any extra training. The challenge of building such a system is to design a proper graph signal representation

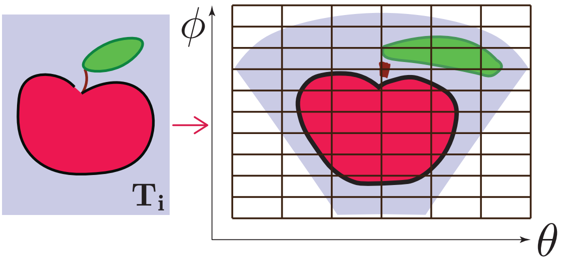


Figure 4.3 – Example of the equirectangular representation of the image. On the left, the figure depicts the original image on the tangent plane $\mathbf{T}_i$, on the right, projected to the points of the sphere. To build an equirectangular image the values points on the discrete regular grid are often approximated from the values of projected points by interpolation.

that allow compensating for the distortion effects that appear on different elevations of **S**. In order to properly define the structure, namely to compute the weights that satisfy the above condition we analyze, how a pattern projected a plane $\mathbf{T}_e$ at equator ($\phi_e = 0, \theta_e = 0$) varies on **S** with respect to the same pattern projected onto another plane $\mathbf{T}_i$ tangent to the sphere at (ϕ_i, θ_i). We use this result to minimize the difference between filter responses of two projected pattern versions. Generally, the weight choice depends on distances d_{ij} between neighboring nodes of graph $w_{ij} = g(d_{ij})$. In this section we show that the function $g(d_{ij}) = \frac{1}{d_{ij}}$ satisfies the above invariance condition.

Pattern choice. For simplicity we consider a 5-point pattern $\{p_0, \ldots, p_4\}$ on a tangent plane, which is depicted by the Fig. 4.4:

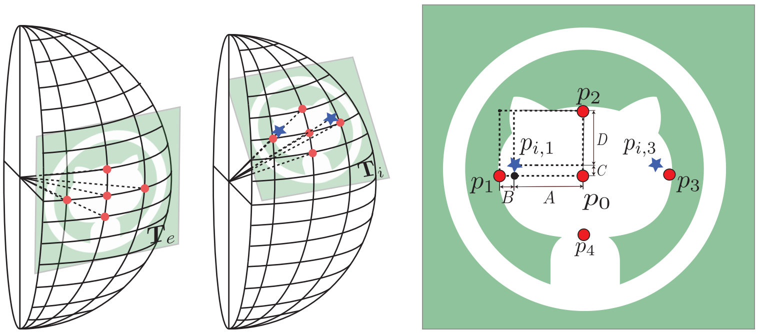
$$p_j := \mathbf{X}_{j,e}, \quad \forall j \in [0..4], \tag{4.4}$$

where $\mathbf{X}_{j,e}$ are the points on the plane $\mathbf{T}_e$ tangent to an equator point $\phi_e = 0, \theta_e = 0$ and $\mathbf{X}_{0,e} = \mathbf{x}_{0,e}$ is the tangency point. Further, pattern points $\{p_0, \ldots, p_4\}$ are also chosen in such a way that they are projected to the following locations on the sphere **S**:

$$\begin{aligned} p_0 \quad &\mapsto (0,0) \\ p_2, p_4 \quad &\mapsto (0 \pm \Delta\phi, 0) \quad . \\ p_1, p_3 \quad &\mapsto (0, 0 \pm \Delta\theta) \end{aligned} \tag{4.5}$$

These essentially correspond to the pixel locations of the equirectangular representation of the spherical surface introduced in Section 4.3.1. The chosen pattern has the following coordinates on the tangent plane at equator $\mathbf{T}_e$:

$$\begin{aligned} \mathbf{X}_{0,e} \quad &= (0,0) \\ \mathbf{X}_{2,e}, \mathbf{X}_{4,e} \quad &= (0, \pm \tan \Delta\phi) \quad . \\ \mathbf{X}_{1,e}, \mathbf{X}_{3,e} \quad &= (\pm \tan \Delta\theta, 0) \end{aligned} \tag{4.6}$$



a) b) c)

Figure 4.4 – a) We choose pattern $p_0,..,p_4$ from an object on tangent plane $\mathbf{T}_e$ at equator ($\phi_e = 0, \theta_e = 0$) (red points) and then, b) move this object on the sphere by moving the tangent plane $\mathbf{T}_i$ to point (ϕ_i, θ_i). c) Thus, the filter localized at tangency point (ϕ_i, θ_i) uses values $p_{i,1}, p_{i,3}$ (blue points) which we can obtain by interpolation.

Filter response. Our objective is to design a graph, which can encode the geometry of an omnidirectional camera in the final feature representation of an image. Ideally, the same object at different positions on the sphere should have the same feature response (see Fig. 4.1) or equivalently they should generate the same response to given filters. Therefore, we choose the graph construction, or equivalently the weights of the graph in such a way that the difference between the responses of a filter applied to gnomonic projection of the same pattern on different tangent planes $\mathbf{T}_i$ is minimized. We consider a graph where each node is connected with 4 of its nearest neighbours and take as an example the polynomial spectral filter $\mathscr{F} = \mathscr{L}_{\text{u}}$ of degree 1 ($\alpha_{0,j} = 0, \alpha_{1,j} = 1$), we can compute the filter response according to the Eq. (2.1) and Eq. (2.10):

$$\mathscr{F}(\mathbf{y}(v_i)) = D_{ii}\mathbf{y}(v_i) - \sum_{j \in \mathscr{E}} A_{ij}\mathbf{y}(v_j), \tag{4.7}$$

at the vertex p_0, one can write in particular:

$$\begin{aligned} \mathscr{F}(\mathbf{y}(p_0)) = \quad & 2(w_V + w_H)\mathbf{y}(p_0) - w_V(\mathbf{y}(p_2) + \mathbf{y}(p_4)) \\ & - w_H(\mathbf{y}(p_1) + \mathbf{y}(p_3)), \end{aligned} \tag{4.8}$$

where w_V, w_H are the weight of the 'vertical' and 'horizontal' edges of the graph. For the graph nodes representing points $p_l : (\theta_l, \phi_l)$ and $p_m : (\theta_m, \phi_m)$ we refer to edges as 'vertical' or 'horizontal' if $\theta_l = \theta_m, \phi_l \neq \phi_m$ or $\theta_l \neq \theta_m, \phi_l = \phi_m$ correspondingly.

We now calculate the filter response $\mathscr{F}(\mathbf{y}(p_{0,i}))$ for a point p_0 on the tangent plane $\mathbf{T}_i$ and compare the result with Eq. (4.8). For simplicity, we assume that we shift the position of the tangent plane by an integer number of pixel positions on the spherical surface, namely ϕ_i, θ_i corresponds to a node of the graph given by the equirectangular image.

According to the gnomonic projection (Eq. (4.2)), the locations of $p_{k,i}, k = [0,..,4]$ defined on the surface of $\mathbf{S}$ as $(\phi_i, \theta_i), (\phi_i \pm \Delta\phi, \theta_i), (\phi_i, \theta_i \pm \Delta\theta)$ correspond to the following positions $\mathbf{x}_{k,i} = (x_{k,i}, y_{k,i})$ on the tangent plane $\mathbf{T}_i$:

$$\begin{aligned} \mathbf{X}_{0,i} &= (0,0) \\ \mathbf{X}_{2,i}, \mathbf{X}_{4,i} &= (0, \pm\tan\Delta\phi) \\ \mathbf{X}_{1,i}, \mathbf{X}_{3,i} &= \left(\pm\frac{\cos\phi_i \sin\Delta\theta}{\sin^2\phi_i + \cos^2\phi_i \cos\Delta\theta}, \frac{\sin\phi_i \cos\phi_i (1-\cos\Delta\theta)}{\sin^2\phi_i + \cos^2\phi_i \cos\Delta\theta}\right) \end{aligned} \tag{4.9}$$

The tangent plane's positions of points $\mathbf{X}_{0,i}, \mathbf{X}_{2,i}, \mathbf{X}_{4,i}$ are independent of (ϕ_i, θ_i), therefore their values remain the same as those of p_0, p_2 and p_4 respectively. However, the positions of points $\mathbf{X}_{1,i}, \mathbf{X}_{3,i}$ depend on (ϕ_i, θ_i), so that we need to interpolate the values of the pattern signal at the vertices $p_{i,1}$ and $p_{i,3}$ (see Fig. 4.4).

We can approximate the values at $p_{i,1}$ and $p_{i,3}$ using the bilinear interpolation method [89]. We denote by A, B, C, D the distances between the corresponding points of the pattern $\mathbf{T}_i$, as shown in Fig. 4.4 (c). We can then express $p_{i,1}$ and $p_{i,3}$ as:

$$\begin{aligned} \mathbf{y}(p_{i,1}) = E^{-1}(AD\mathbf{y}(p_1) + BD\mathbf{y}(p_0) + CB\mathbf{y}(p_2)), \\ \mathbf{y}(p_{i,3}) = E^{-1}(AD\mathbf{y}(p_3) + BD\mathbf{y}(p_0) + CB\mathbf{y}(p_2)), \end{aligned} \tag{4.10}$$

where, using Eq (4.9),

$$\begin{aligned} E &= (C+D)(A+B), \\ A+B &= \tan\Delta\theta, \qquad . \\ C+D &= \tan\Delta\phi \end{aligned} \tag{4.11}$$

Using Eq. (4.10) we can then write the expression for the filter response $\mathscr{F}(\mathbf{y}(p_{0,i}))$, as follows:

$$\begin{aligned} \mathscr{F}(\mathbf{y}(p_{0,i})) &= 2(w_{i,V} + w_{i,H})\mathbf{y}(p_0) \\ &- w_{i,V}(\mathbf{y}(p_2) + \mathbf{y}(p_4)) \\ &- w_{i,H}(\mathbf{y}(p_{i,1}) + \mathbf{y}(p_{i,3})), \end{aligned} \tag{4.12}$$

where $w_{i,H}$ and $w_{i,V}$ are the weights of the 'horizontal' and 'vertical' edges of the graph at points with the elevation ϕ_i.

Objective function. We now want the filter responses a p_0 and $p_{0,i}$ in (Eq. (4.8) and Eq. (4.12)) to be close to each other in order to build translation-invariant features. Therefore, we need to find weights $w_H, w_V, w_{i,H}$ and $w_{i,V}$ such that the following distance is minimized:

$$\left|\mathscr{F}(\mathbf{y}(p_{0,e})) - \mathscr{F}(\mathbf{y}(p_{0,i}))\right|. \tag{4.13}$$

Additionally, as we want to build a unique graph independently of the tangency point of $\mathbf{T}_i$ and $\mathbf{S}$, we have additional constraint of $w_V = w_{i,V}$. The latter is important, as from Eq. (4.9) we can see that 'vertical' (or elevation) distances are not affected by translation of the tangent plane.

We assume that the camera has a good resolution, which leads to $\Delta\theta \simeq 0$. Therefore, based on Eq. (4.13), we can derive the following:

$$\begin{cases} w_H \simeq \left(\frac{\cos\phi_i \cos\Delta\theta}{\sin^2\phi_i + \cos^2\phi_i \cos\Delta\theta}\right) w_{i,H} = w_{i,H}\cos\phi_i, \\ \cos\Delta\theta \simeq 1. \end{cases} \tag{4.14}$$

Therefore, under our assumptions, we can conclude that the difference between filter responses, defined by Eq. (4.13) is minimized if the following condition is valid:

$$w_{i,H} = w_H \left(\cos\phi_i\right)^{-1}, \tag{4.15}$$

where w_H is the weight of the edge between points on the equator of the sphere $\mathbf{S}$.

Now, we can use this result to choose a proper function $g(d_{ij})$ to define the weights $w_{i,H}$ based on the Euclidean distances between two neighboring points $(\mathbf{x}_i, \mathbf{x}_j)$ on the sphere $\mathbf{S}$. For the case when $\phi_i = \phi_j = \phi_*, \theta_i \neq \theta_j$ the Euclidean distance can be expressed as follows:

$$d_{ij}^2 = r^2(1 - \cos\Delta\theta)(1 + \cos 2\phi_*) = r^2 \cos^2\phi_*. \tag{4.16}$$

For simplicity let us denote $d_{ij} = d_{\phi_*}$, where ϕ_* is the elevation of the points $\mathbf{x}_i, \mathbf{x}_j$. Using these notations, we can compute the proportion between distances d_{ϕ_e} and d_{ϕ_*}, which are the distances between neighboring points at equator $\phi_e = 0$ and elevations ϕ_* respectively. It reads:

$$\frac{d_{\phi_*}}{d_{\phi_e}} = \frac{\cos\phi_*}{\cos\phi_e}. \tag{4.17}$$

Given Eq. (4.15), we can rewrite Eq. (4.17) for elevation $\phi_* = \phi_i$ as:

$$\cos\phi_i = \frac{d_{\phi_i}}{d_{\phi_e}} = \frac{w_H}{w_{i,H}}. \tag{4.18}$$

As we can see, the distance between neighboring points on different elevation levels ϕ_i, is proportional to $\cos\phi_i$. Given Eq. (4.13), we can see that making weights inversely proportional to Euclidean distance allows to minimize difference between filter responses. Therefore, we propose using $w_{i,H}$ as:

$$w_{i,H} = \frac{1}{d_{\phi_i}}. \tag{4.19}$$

This formula can also be used to compute the weights for vertical edges, as the distance d between any pair neighboring points $(\mathbf{x}_i, \mathbf{x}_j)$, for which $\theta_i = \theta_j$ and $\phi_i \neq \phi_j$ is constant. This nicely fits with our assumption that the weights of 'vertical' edges should not depend on the tangency point of plane $\mathbf{T}_i$ and sphere $\mathbf{S}$.

Thus, summing it up we choose the weights w_{ij} of a graph based on the Euclidean distance between pixels on spherical surface d_{ij} as follows:

$$w_{ij} = \frac{1}{d_{ij}}. \tag{4.20}$$

The graph representation finally forms the set of signals $\mathbf{y}$ that are fed into the network architecture defined in chapter 3.

4.4 Experiments

In this section we present our experiments. We first describe the datasets that we use for evaluation of our algorithm. We then compare our method to state-of-the-art algorithms.

We have used the following two datasets for the evaluation of our approach.

MNIST-012 is based on a fraction of the popular MNIST dataset [58] that consists of 1100 images of size 28 × 28, subdivided in three different digit classes: '0', '1' and '2'. We then randomly split these images into training, validation and test sets of 600, 200 and 300 images respectively. In order to make this data suitable for our task we project them to the sphere at a point (ϕ_i, θ_i), as depicted by Fig. 4.2. To evaluate accuracy with the change of (ϕ_i, θ_i), for each image we randomly sample from 9 different positions: $\phi_i \in \{0, 1/8, 1/4\}, \theta_i \in \{\pm 1/8, 0\}$. Finally we compute equirectangular images (see Fig. 4.2) from these projections, as defined in Section 4.3.1 and use the resulting images to analyze the performance of our method.

ETH-80 is a modified version of the dataset introduced in [59]. It comprises 3280 images of size 128 × 128 that features 80 different objects from 8 classes, each seen from 41 different viewpoints. We further resize them to 50 × 50 and randomly split these images into 2300 and 650 training and test images, respectively. We use the remaining 330 ones for validation. Finally, we follow the similar procedure to project them onto the sphere and create equirectangular images as we do for MNIST-012 dataset.

For our first set of experiments we train the network in [6] with the following parameters. We use two spectral convolutional layers with 10 and 20 filters correspondingly, with global pooling which selects P_1 and P_2 nodes, where the parameters $P_1 = 2000$ and $P_2 = 200$ for MNIST-012 dataset and $P_1 = 2000$, $P_2 = 700$ for ETH-80 dataset. We then use a statistical layer with 12 × 2 statistics and three fully-connected layers with ReLU and 500, 300, 100 neurons correspondingly.

We have evaluated our approaches with respect to baseline methods in terms of classification accuracy. The **MNIST-012** dataset is then primarily used for the analysis of both the architecture and the graph construction approach. We then report the final comparisons to

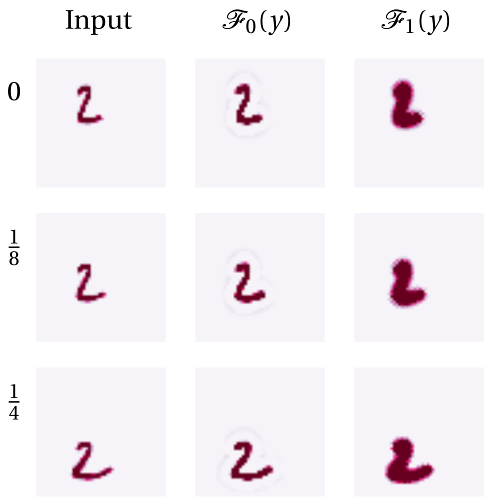


Figure 4.5 – Example of the feature maps of the last spectral convolutional layer extracted from equirectangular images. The first column corresponds to the original images created for the same object, which is projected from different tangent planes $\mathbf{T}_i$, with $\phi_i \in \{0, \frac{1}{8}, \frac{1}{4}\}$; the last two columns show the feature maps given by two randomly selected filters.

state-of-the-art approaches on the **ETH-80** dataset.

First of all, we visually show that feature maps on the last convolutional layer of our network are similar for different positions, namely for different tangent planes $\mathbf{T}_i$ with the same object. Fig. 4.5 and 4.6 depict some feature maps of images from MNIST-012 and ETH-80 correspondingly.

The first column of each figure shows original equirectangular images of the same object projected to different elevations $\phi_i = [0, 1/8, 1/4]$ and the rest visualize feature maps produced by two randomly selected filters. We can see, that the feature maps stay similar independently of the distortion of the corresponding input image. We believe that this, further, leads to closer feature representations, which is essential for good classification.

We recall that the goal of our new method is to construct a graph to process images from omnidirectional camera and use it to create similar feature vectors for the same object for different positions (ϕ_i, θ_i) of the tangent plane. To justify the advantage of proposed approach we design the following experiment. First of all, we randomly select three images of digits '2', '1' an '0' from the test set of MNIST-012. We then project each of these images to 9 positions on the sphere $\phi_i \in \{0, 1/8, 1/4\}, \theta_i \in \{\pm 1/8, 0\}$. We then evaluate Euclidean distances between the features that are given by the statistical layer of the network for all pairs of these 27 images. Fig. 4.7 presents the resulting $[27 \times 27]$ matrix of this experiment for a grid graph and for the proposed graph representation, which captures the lens geometry. Ideally we expect that images with the same digit give the same feature vector regardless of different elevations ϕ_i of the tangent plane. This essentially means that cells of the distance matrix should have low

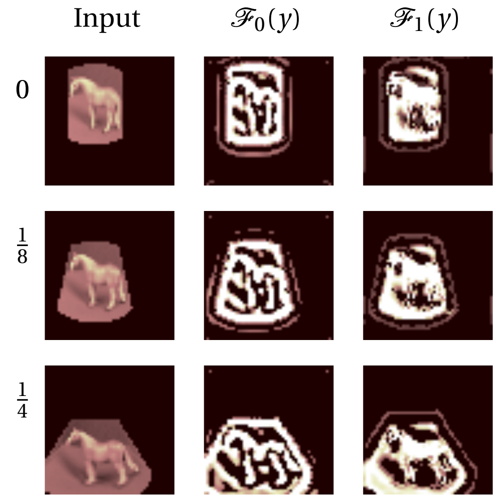
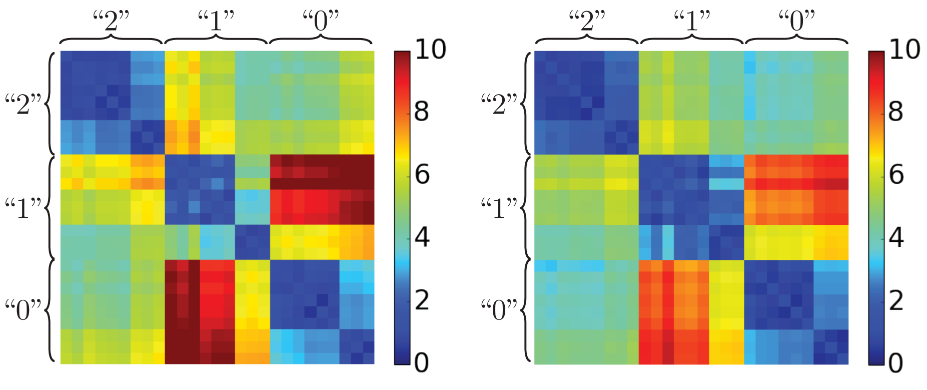


Figure 4.6 – Example of feature maps of equirectangular images from **ETH-80** datasets. Here, we randomly select an input image from the test set and project it on three elevation $\phi_i \in \{0, 1/8, 1/4\}$ and two spectral filters, which are named $\mathscr{F}_1$ and $\mathscr{F}_2$. The figure illustrates resulting feature maps given by the selected filters (second and third columns) and input images (first column).

method	graph type	# Parameters	**ETH-80** Accuracy (%)
classic Deep Learning:			
FC Nets	–	1.4M	71.3
STN [38]	–	1.1M	73.1
ConvNets [37]	–	1.1M	76.7
graph-based DLA:			
ChebNet [14]	grid	3.8M	72.9
TIGraNet [6]	grid	0.4M	74.2
ChebNet [14]	geometry	3.8M	78.6
Ours	geometry	0.4M	**80.7**

Table 4.1 – Comparison to the state-of-the-art methods on the **ETH-80** datasets. We select the architecture of different methods to feature similar number of convolutional filters and neurons in the fully-connected layers.

value on the [9 × 9] diagonal sub-matrices, which correspond to the same object, and high values on the rest of the matrix elements. Fig. 4.7 shows that our method gives more similar features for the same object compared to the approach based on a grid graph. This suggests that building graph based on image geometry, as described in Section 4.2, makes features less sensitive to image distortions. This consequently simplifies the learning process of the algorithm.



a) b)

Figure 4.7 – Illustration of the Euclidean distances between the features given by the networks of a) [6] and b) our geometry-aware graph. This figure depicts resulting matrix [27 × 27] for the images from 3 classes and 9 different positions, where axises correspond to the image indexes. Each diagonal [9 × 9] sub-matrix corresponds to the same object (digits "2", "1", "0"), the lowest value (blue) corresponds to the most similar features and the highest value (red) to the least similar (best seen in color).

We further evaluated our approach with respect to the state-of-the-art methods on ETH-80 dataset. The competing deep learning approaches can be divided in classical and graph-based methods. Among the former ones we use Fully-connected Networks (FCN), Convolutional Network (ConvNets) [37] and Spatial Transformer Networks (STN) [38]. STN has an additional to ConvNets layer which is able to learn specific transformation of a given input image. Among the graph-based methods, we choose ChebNet [14] and TIGraNet [6] for our experiments. ChebNet is a network designed based on Chebyshev polynomial filters. TIGraNet is a method invariant to isometric transformation of the input signal. The architectures are selected such that the number of parameters in convolutional and fully-connected layers roughly match each other across different techniques. More precisely, all networks have 2 convolutional layers with 10 and 20 filters, correspondingly, and 3 fully-connected layers with 300, 200 and 100 neurons. Filter size of the convolutional layer in classical architectures is 5×5. For ChebNet we try polynomials of degree 5 and 10 and pick the latter one as it produces better results. For TIGraNet we use polynomial filters of degree 5. The results of this experiment are presented in Table 4.1.

Table 4.1 further shows that ConvNet [37] outperforms TIGraNet [6]. This likely happens as [6] gathers global statistics and loses the information about the location of the particular object. This information, however, is crucial for the network to adapt to different distortions on omnidirectional images. We can see that the introduced graph construction method helps to create similar feature representations for the same object at different elevations, which results into different distortion effects but similar feature response. Therefore, the object looks similar for the network and global statistics become more meaningful compared to a method based on the regular grid graph [6].

Further, we can see that the proposed graph construction method allows to improve accuracy of both graph-based algorithms: ChebNet-geometry outperforms ChebNet-grid, and proposed algorithm based on TIGraNet outperforms the same method on the grid-graph [6]. Finally, we also notice that our geometry-based method performs better than ChebNet-geometry on the ETH-80 task due to the isometric transformation invariant features; these are an advantage for the image classification problems, where images are captured from different viewpoints.

Thus, we can conclude that our algorithm produces similar filter responses for the same object at different positions. This, in combination with global graph-based statistics, leads to the better classification accuracy.

4.5 Conclusion

In this chapter we have proposed a novel image classification method based on deep neural network that is specifically designed for omnidirectional cameras, which introduce geometric distortion effects. Our graph construction method allows for learning filters that respond similarly to the same object seen at different positions on the equirectangular image. We evaluated our method on challenging datasets and prove its effectiveness in comparison to state-of-the-art approaches that are agnostic to the geometry of the images.

Our discussion in this chapter was limited to specific type of the mapping projection. However, the graph-based solution has the potential to be extended to more general geometries of the camera lenses, which we explore in the next chapter.

5 Projective geometry-aware anisotropic convolutional filters

5.1 Introduction

In the previous chapter we showed that, by modifying the weights between edges in the graph that is used to represent an omnidirectional image, we can adapt the network to learn features that are aware of the geometry of such images. In Chapter 3 we presented one way of defining an isotropic convolutional filter that operates on an image represented as a signal on an undirected graph. While these filters can be efficiently used within the TIGraNet framework for such tasks as image classification, using these filters for other problems (for example compression or even for the same image classification task, but without dynamic pooling and statistical layers used in the TIGraNet framework) is challenging, as by construction they are not able to encode the orientation of object seen in the image. Fig. 5.1 illustrates this issue on a sample compression task. The images processed by the network that is composed of graph-based polynomial filters (Fig. 5.1(b,d)) are over-smoothed and have particular blocking artifacts, which appear due to the isotropic nature of the filters. To overcome this problem we introduce in this chapter a different way of constructing graph-based image representations. It not only permits to adapt to the geometry of an omnidirectional image, but also to learn anisotropic filters, which have a much broader application area than isotropic ones that are discussed in the previous chapters.

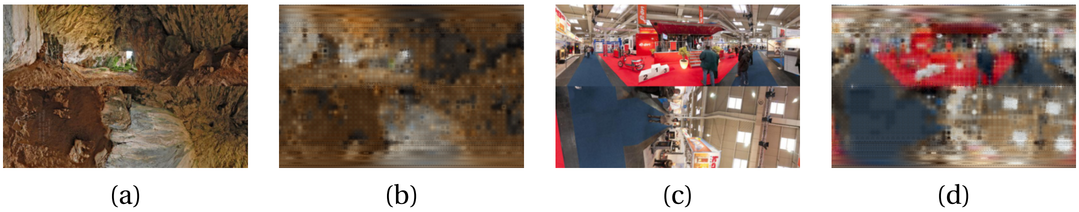

(a) (b) (c) (d)

Figure 5.1 – Example of original (a,c) and decompressed (b,d) image encoded with graph-based isotropic filters. The images are the result of deep network compression algorithm, [90], where convolutional filters are replaced by graph-based isometric ones introduced in Chapter 4.

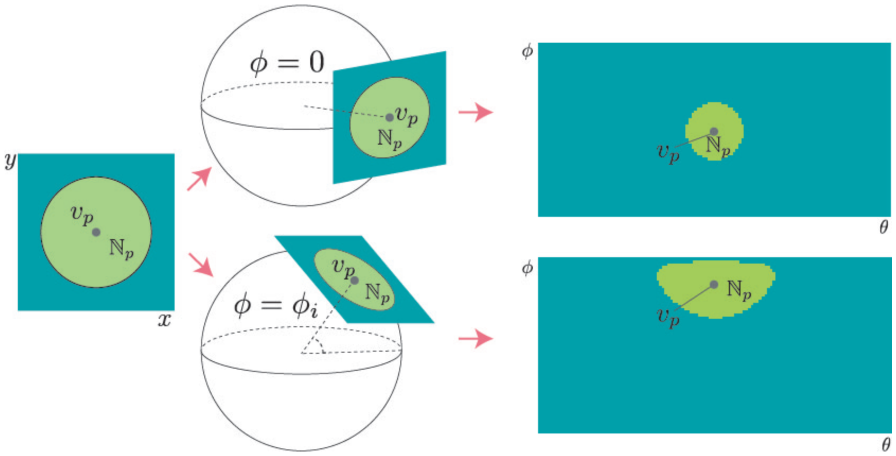


Figure 5.2 – Our network adapts the size and shape of the filter with respect to the elevation of omnidirectional image, where it is applied.

5.2 Omnidirectional image processing

As we discussed in the previous chapter, the most typical way of dealing with images taken by omnidirectional cameras is to apply standard image processing techniques directly on the equirectangular projection images [86]. However, due to the strong distortion effects introduced by the process of unwrapping of the projection surface to a plane, standard techniques lose much of their efficiency, as the appearance of the same object may change depending on its location in the image. To overcome this problem in the previous chapter we introduced an approach that relies on the graph-based special convolutional filers, which adapt to the geometry of omnidirectional images. This method, however, relies on the convolutional operation defined in the spectral domain of the graph, which leads to isotropic filters and may reduce the complexity of trained filters. In this chapter we, on the other hand, propose to build anisotropic graphs-based convolutional filters that do not have this limitation. Thus, in this section we discuss recent approaches, which use anisotropic filters to extend deep learning algorithms for the omnidirectional camera geometry.

The authors of [91] suggest adapting the size of the convolutional kernel to the elevation of the equirectangular image. The main limitation of this technique, however, is that it requires a significantly larger number of parameters than the competing techniques, as it does not have the weight sharing property of CNNs. It rather requires learning different convolutional filters for different elevations of the equirectangular image. Further, [92] propose to learn the shape of the convolutional filter, by learning the sampling locations (position offsets), where the elements of the filter are evaluated. The authors of [93] extend later this idea and suggest learning dynamic offsets of the filter elements depending on the image content, which allows the filters to adapt to different parts of the image. This method is quite flexible, however in the context of omnidirectional images requires an extensive training set, as the network needs to learn how it should react to various objects appearing at any possible elevation. In our work we rather take advantage of the knowledge of the image projective geometry and use this knowledge in the design of our architecture to adapt the size and shape of convolutional filters.

A different approach is suggested by [94], who introduces a CNN that is designed for spherical shapes and define filters directly on its surface. This method, however, is specifically designed for processing spherical images, while our approach is easily adapted to different kind of shapes, which we show in our experiments. Further, the methods of [95, 96] suggest a different way of compensating for the distortion of omnidirectional image. They suggest adapting the sampling locations of the convolutional filters to the geometry of the lens by projecting kernels to the sphere and using interpolated pixel values on the projected locations for implementing the convolutional filters. While these works are the closest to in spirit to ours, we propose a more general architecture, which permits to adapt the shape and size of the convolutional kernel to the location of omnidirectional image, and therefore to use the information about all the pixels and not only of a subset of them.

Then, the authors in [97, 98] suggest a completely different approach to tackle the distortion of omnidirectional images. Instead of working with equirectangular images, they propose to project an omnidirectional image to a cube, where each of its faces represents an image that would have been seen through a regular perspective camera, with the optical center located in the center of the cube [5]. Representing an omnidirectional image in this way allows having less noticeable distortion effects as compared to equirectangular images. This representation, however, suffers from another type of distortion that appears due to discontinuity effect on the borders between the faces of the cube. To mitigate this issue, [97] propose to apply a smoothing filter as a post-processing step and [98] suggest an algorithm that enforces consistency between the neighboring facets of the cube. Contrary to the mentioned approaches, as we model cube surface as a graph, our algorithm can easily handle the discontinuity problem and adapt to image distortions introduced by the cube-map projection.

5.3 Geometry-aware CNN

In this section we describe our algorithm, which adapts convolutional filters to the distortion of omnidirectional images. In order to achieve this, we build a graph $\mathscr{G}$ in such a way that the neighbourhood of each node is different for different elevations of the omnidirectional image. In the following section we describe two approaches that rely on undirected and directed graphs and define isotropic and anisotropic geometry-aware (GA) filters respectively.

Undirected graph construction for adaptive filtering. To adapt $\mathscr{F}$ to the elevation level we construct a graph $\mathscr{G}$ with nodes that have different neighborhoods depending on the elevation. To do so we define a circular area on a tangent plane $\mathscr{T}$, centered in the tangency point. Then we move $\mathscr{T}$ such that it becomes tangent to the sphere $\mathscr{S}$ in different positions v_p and project the circular area onto $\mathscr{S}$. For every point of $\mathscr{S}$ this creates a neighborhood N_p, which changes its shape and size together with the elevation, as can be seen in Fig. 5.2.

Based on this geometry adaptive neighborhood, we then construct the graph $\mathcal{G}$ in the following way. We connect the node $v_p \in \mathcal{G}$, corresponding to a tangent point on the sphere, with the node $v_j \in N_p$. The corresponding edge e_{pi} has a weight w_{pi} that is inversely proportional to the Euclidean distance between v_p and v_i, which are defined on the sphere:

$$\begin{aligned} &w_{pi} = ||v_p - v_i||_{L_2}^{-1}, \quad && v_i \in N_p\,, \\ &w_{pi} = 0, && v_i \notin N_p\,. \end{aligned} \tag{5.1}$$

This allows us to vary the size of the neighbourhood according to the geometry of the omnidirectional image for each node in $\mathcal{G}$, and weight the contribution of the nodes to the final filter response according to their distances to v_p. Therefore, depending on the elevation the filter is changing its shape and size.

While effective, filter $\mathcal{F}$ does not have a defined orientation in space as according to Eq. (2.10) the filter applies the same weights α_l to all nodes in the l-hoop neighborhood, with the contribution of each node being weighted by the distance to v_p. This results in $\mathcal{F}$ being isotropic, which leads to suboptimal representation, as the network is not able to encode the orientation of the object.

Directed graphs construction. In order to overcome the limitations of isotropic filters, we propose to replace a single undirected graph $\mathcal{G}$ with multiple directed graphs $\mathcal{G}_k$, where each $\mathcal{G}_k$ defines its own orientation. Let us consider the case of a 3x3 classical convolutional filter. In this case the filter has 9 distinct elements. To mimic the same structure with our graph convolutional filters we employ the following algorithm. First we define 9 non-overlapping areas $s_k, k = 1..9$ on the tangent plane, which together form a rectangular region that is centered in the tangency point v_p of T as defined in the Fig. 5.3. This rectangular region effectively defines the receptive field of the filter on the tangent plane. Then we build a set of nine directed graphs $G_k, k = 1..9$ in the similar way, as mentioned in the previous section for undirected graph. In particular in order to build graph $\mathcal{G}_k$ we do as follows. For the area s_k and for every node v_p we move the tangent plane at point v_p and then project s_k from T onto the sphere. This operation defines a specific neighborhood $N_k(p)$ on the sphere that consists of the points that belong to the projection of the region s_k from the plane T. We then connect v_p with a directed edge to each of these points, where the weight of the edge is defined in Eq. (5.1). Note that the direction of the edge is very important, because connecting v_p and v_i with an undirected edge forces v_p to be part of the neighborhood $N_k(i)$. This, however, is not possible, as the neighborhood $N_k(i)$ is computed by projecting the area s_k from the plane T that is tangent to the sphere at point v_i and does not include v_p.

This results in construction of the directed graph $\mathcal{G}_k$, which corresponds to the k^{th} region of the filter, illustrated in Fig. 5.3. We repeat this operation for all the areas $s_k, k = 1..9$ of our filter, which leads to creation of 9 directed graphs $\mathcal{G}_k, k = 1..9$. Given this set of graphs $\mathcal{G}_k$ we define the resulting convolutional operation F as follows:

$$F = \sum_{k=1}^{9} \mathscr{F}_k, \tag{5.2}$$

where $\mathscr{F}_k$ is the filtering operation defined on the graph $\mathscr{G}_k$. Note that this filtering operation is slightly different from the operation that is used when working with undirected graphs and is discussed in more details in the following section.

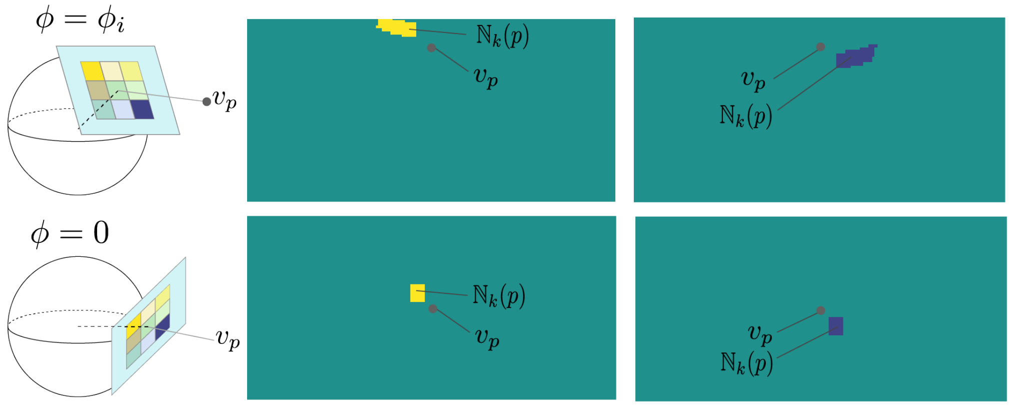


Figure 5.3 – Illustration of 3 × 3 GA-filter kernel, defined on the tangent plane. Projection of area s_k forms the neighborhood $N_k(p)$ of the node v_p, where the GA-filter is applied. The right part of the figure illustrates the change of the neighborhood $N_k(p)$ depending on the location of v_p and the chosen filter area s_k.

To sum up, the introduced graph construction process allows having anisotropic filters F, defined in Eq. (5.2) that are capable of capturing the orientation of the object and therefore learn more meaningful feature representation for an image compared to the isotropic graph-based filters. It is important to note that in our work we use the set of 9 non-overlapping rectangular areas defined on the tangent plane, as shown by Fig. 5.3, due to their rough correspondence to the elements of a 3 × 3 convolutional filter. However, our method can be easily extended to an arbitrary number of such areas with arbitrary shapes.

Geometry aware anisotropic filters. For directed graphs $\mathscr{G}_k$ Laplacian matrix is not defined, therefore, we use the polynomial filters proposed in [99]. Instead of the Laplacian matrix these filters rely on the normalized adjacency matrix $\mathscr{A}_k$, which is defined as follows:

$$\mathscr{A}_k = D_k^{-1} A_k, \tag{5.3}$$

where A_k and D_k are the weighted adjacency and the diagonal degree matrices of graph $\mathscr{G}_k$ respectively. The elements of D_k are computed as $D_k(m,m) = \sum_n A_k(m,n)$. Then, we define filters in the following way:

$$\mathscr{F}_k = \alpha_0^{(k)} + \alpha_1^{(k)} \mathscr{A}_k, \tag{5.4}$$

where $\alpha_0^{(k)}, \alpha_1^{(k)}$ are the training parameters of our filter. Here, we use polynomial filters of degree 1, as they achieve good balance between speed and performance.

Network architecture. The introduced approach focuses on the modification of the convolutional layer to incorporate the knowledge about the image projective geometry inside the neural network. Thus, it can be applied for a broad variety of tasks. In this chapter we focus on the image classification and compression problems. For the former one we use a relatively standard architecture that consists of a sequence of convolutional layers with the introduced graph-based filters, followed by a sequence of the fully connected layers. For the compression task we use the architecture proposed in [90] and replace its convolutional filters with the proposed graph-based ones.

Discussion. Our method can be seen as a generalization of different approaches that have been developed for omnidirectional images. For example, if the node v_p at elevation ϕ_i has only one neighbor in each direction and the weight of the edges between nodes is always equal to one, it will be the standard CNN method [3]. Further, if these neighbors correspond to the projected points it becomes the recently proposed algorithm of [95]. Finally, if we replace directed graphs with a single undirected one we get the same behavior of the polynomial filters as described in graph-based deep learning methods [14, 16].

5.4 Results

In this section we illustrate the performance of our approach. We start by evaluating our method with respect to competing approaches on the task of classification images that are projected to different surfaces. Finally, to show the generality of our approach and illustrate the effectiveness of the anisotropic graph-based filters, we evaluate our method on the image compression task.

5.4.1 Image Classification

In this section we first introduce the datasets that we used for the evaluation of our method. We then discuss the baseline approaches and architectures, which we use in our experiments. Finally, we show the quantitative comparison of our method with the competing ones.

Datasets. We evaluate our method on three different types of data, which represent different surface geometries, where images are projected:

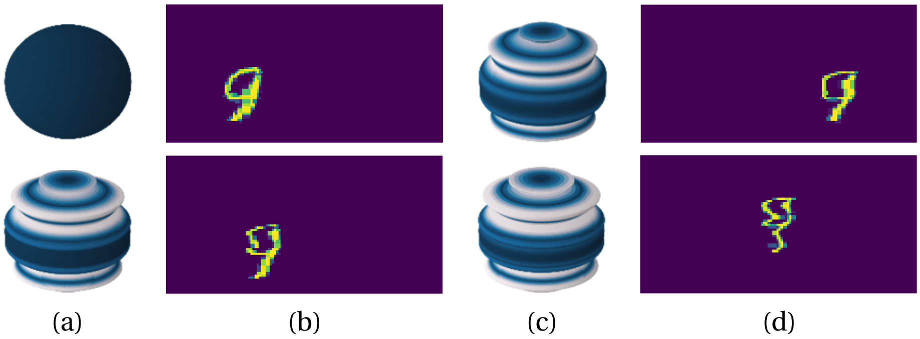


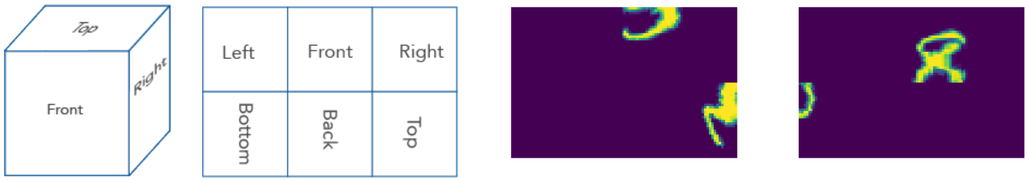
Figure 5.4 – Illustration of surfaces (a,c) and corresponding equirectangular images (b,d) of the digit 9 projected to these surfaces at random positions. White color of surfaces highlights the furthest points to the spherical surface of the same radius in terms of Euclidean distance; blue color indicates the closest points.

- **Spherical** dataset (S) consists of images projected on different locations on a sphere. The resulting spherical images are then unwrapped to equirectangular images, as described in Section 4.3.1.
- **Mod-spherical** dataset (MS) features image projections on complicated surfaces that are depicted by Fig. 5.4 together with the representative examples of projected images. This dataset itself consists of three different versions: MS1, MS2, MS3 which correspond to the surfaces which are getting further away from the spherical one. A more detailed discussion about the type of projection and the surface geometry used in these datasets can be found in the Appendix.
- **Fish-eye** dataset (F) consists of images projected on different locations on a sphere using stereographic projection [100], which is frequently used in fish-eye cameras.
- **Cube-map** dataset (CM) features projection of the images on the cube as shown by Fig. 5.5. This type of projection has recently gained popularity for handling omnidirectional images, due to its ability to reduce distortion artifacts that appear due to the spherical geometry.

In all these datasets we use MNIST images from [58], which are divided into training, validation and test sets with 54k, 6k and 10k samples respectively.

Architecture. We compare our approach with standard ConvNets, the algorithm proposed in [94] and other graph-based methods. For the graph-based methods, we investigate three possible ways of constructing the graph $\mathcal{G}$:

- Regular grid-graph with 8 neighbors and all equal weights $w_{ij} = 1$;



(a) (b) (c)

Figure 5.5 – **Cube-map projection**: (a) schematic illustration of the unwrapping process of the cube surface with a baseball arrangement [5] onto a planar surface; (b,c) sample projections of images from the MNIST dataset on the cube surface unwrapped into rectangular images.

- Regular grid-graph with 8 neighbors and weights that depend on the Euclidean distance d_{ij} between the nodes, as proposed in [101], i.e., $w_{ij} = d_{ij}^{-1}$;
- Irregular GA-graph with $w_{ij} = d_{ij}^{-1}$ (isotropic filters from Section 5.3).

For all of them we build a normalized Laplacian matrix and use polynomial filters of degree 1, which is equivalent to using 3×3 convolutional kernels. Therefore, for the standard ConvNet architecture we similarly rely on filters of size 3×3. We present our results in Table 5.1[1].

Fig. 5.6 illustrates the architecture of our classification network. All the competing approaches use networks of roughly the same complexity. For all the methods we use the architectures of similar structure and roughly the same number of parameters. For all the graph-based approaches we use the graph-based convolutions with stride two on each layer, which in turn requires building a graph for each new layer according to its respective sampling. For the method of [94] we used the architecture proposed in the paper with roughly the same number of parameters as in the competing approaches that we evaluate.

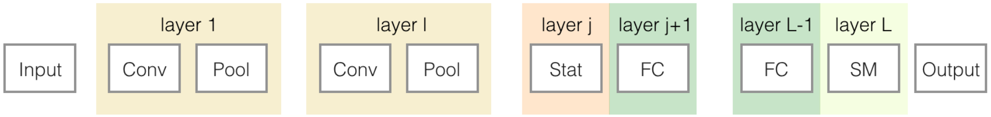


Figure 5.6 – Our classification network consists of three geometry-aware graph-based convolutional layers (GAL) with stride 2. Each of the GAL $[X \times Y$, stride $2]$ convolutional layers contains Y filters with X parameters. The convolutional layers are followed by an average pooling operation AP and two fully-connected layers FC$[Z]$ with Z neurons each.

Evaluation. We compare the performance of our approach with the one of baseline methods in Table 5.1. Our method significantly outperforms the standard ConvNets, as it is designed to

---

[1] We were unable to compare our method to the recent work of [95] as to the best of our knowledge, there is no publicly available implementation.

Table 5.1 – Evaluation of different approaches on Spherical (S), Mod-Spherical (MS1, MS2, MS2), Fish-eye (F) and Cube-Map (CM) datasets.

Method	**S**	**MS1**	**MS2**	**MS3**	**F**	**CM**
regular graph ($w_{ij} = 1$)	69.4	64.3	64.1	62.8	71.8	40.0
regular graph ($w_{ij} = 1/d_{ij}$)	69.8	63.4	64.5	62.5	70.2	40.5
GA graph ($w_{ij} = 1/d_{ij}$)	70.2	63.9	62.5	62.8	72.1	44.2
ConvNets	94.2	91.3	91.2	90.5	93.4	79.4
SphereNet [94]	95.2	84.5	83.3	80.9	94.9	–
Ours	**96.9**	**95.1**	**95.3**	**94.9**	**95.7**	**84.3**

explicitly use the geometry of the omnidirectional images. Further it shows a much higher accuracy then other graph-based techniques, which rely on isotropic filters. Further, our method achieves comparable accuracy to [94] on spherical image representation and it outperforms [94] on other datasets. Finally, we are able to run our approach on cub-map projection while the SphericalCNN by design is not applicable to such kind of images.

5.4.2 Image Compression

In all our previous experiments we have focused on evaluating our approach on the image classification task. To show the generality of our method and better illustrate the effectiveness of anisotropic graph-based filters, we now evaluate their performance on an image compression problem. For this task, we choose to modify the architecture introduced in [90] by replacing the ordinary convolutional layers with our own graph-based convolutions. In this section we first introduce our approach and then compare the performance of the two graph-based methods, which rely on isotropic and anisotropic graph-based filters respectively.

Image compression framework. The method introduced in [90], presents the process of image compression is an optimization of the tradeoff between having small distortion of the pixel intensity values and the small number of bits that are required for storing the compressed representation of these values. As described in [102, 90], this optimization can be represented as a variational autoencoder.

We briefly describe the compression approach, proposed by [90]. An input image x is encoded using a function $g_a(x;\alpha)$, which results in the respective latent representation y. Then, y is quantized into $\hat{y}$, which can be losslessly compressed using entropy coding algorithms. The quantized representation $\hat{y}$ is then passed to the decoder $g_s(\hat{y};\beta)$ at the decompression step, which results in a decompressed image $\hat{x}$. Here, we denote by α and β the parameters of the encoding and decoding algorithms respectively. While both encoder and decoder can be represented as a differentiable function, the process of quantization is non-differentiable.

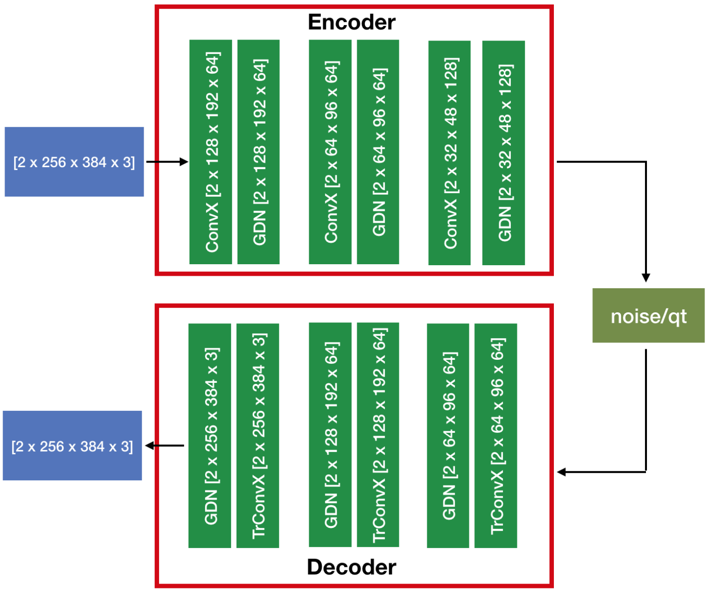


Figure 5.7 – **Architecture of compression algorithm**. For our experiment we use omnidirectional images projected to the cube surface with a baseball arrangement with size $[256 \times 384 \times 3]$ grouped in batches of size 2. Here, ConvX and TrConvX denote convolutional and transpose convolutional layers with stride 2, where X corresponds to the possible choice of either standard convolutional or proposed geometry-aware graph-based filters; and GDN is a normalization layer, which is proposed by [102].

Therefore the authors of [102] propose to replace the quantization with an additive uniform noise at the training step as follows:

$$\tilde{y}_i = y + \Delta y, \tag{5.5}$$

where Δy denotes additive i.i.d uniform noise. This trick allows to perform the end-to-end optimization of both the encoder and decoder parameters using the following loss function:

$$L(\alpha,\beta) = E_{x,\Delta y}\left[-\sum_i log_2 p_{\tilde{y}_i}(g_a(x;\alpha)+\Delta y) + \quad \lambda d(g_s(g_a(x;\alpha)+\Delta y;\beta),x)\right], \tag{5.6}$$

where g_s, g_a are convolutional deep neural networks, d represents the distance between the images and λ is a weighting parameter. Thus, during the training step, we add noise (according to Eq. (5.5)) to be able to back propagate the error. At the inference time we apply quantization to the latent representation y. The overall architecture that we use is similar to the one proposed in [90] and is summarized in Fig. 5.7. The method of [90] relies on the standard convolutional layers, which are practical for ordinary images: they allow learning local image structures independently of their location in the image. Instead, we replace the standard convolutional filters by our new GA-filters.

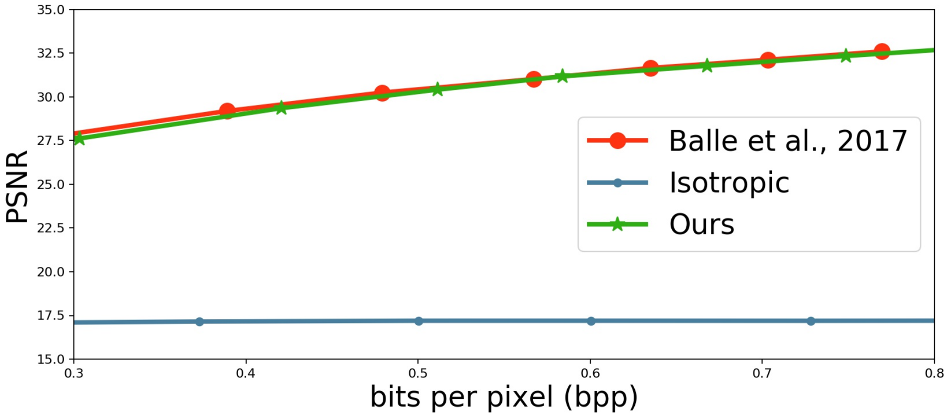


Figure 5.8 – PSNR results of the decompressed images for different filter types with respect to bit per pixel values.

Evaluation. We now evaluate the performance of our approach. For this experiment we have implemented two versions of our system. One with isotropic graph-based filters and the other one with the anisotropic ones. We further evaluate the original method [90] for the sake of completeness. All three methods are trained and tested on the same splits of the modified version of the dataset [103], which consists of omnidirectional images projected onto a cube. From this dataset we use 3900 images for training and 1000 for testing of the methods.

We compare the methods in terms of the Peak Signal to Noise Ratio (PSNR) with respect to the average number of bits per pixel (bpp). The results of the evaluation are presented in Fig. 5.8. As we can see, our method with anisotropic filters and [90] show similar PSNR values and significantly outperform the architecture with isotropic filters. Further, due to the fact that PSNR value depends on the average difference in pixel values between the compressed image and the original one it is not able to reliably detect small artifacts that appear in the cube-map images, which are noticeable for humans. These artifacts are clearly seen in Fig. 5.9, which shows that, due to the knowledge of the image projective geometry, our approach correctly reconstructs the areas along cube borders, while the method of [90] over-smooths these areas.

Fig. 5.10 further illustrates some visual comparisons of the methods. We can see that isotropic filters produce over-smoothed decompressed images, which do not look realistic and result in very low PSNR values. On the other hand our method with anisotropic filters is able to produce sharp results, which is an important property for VR applications.

5.5 Conclusion

In this chapter we have presented generic method of graph construction that allows incorporating the information about the image geometry inside the neural network architecture. Further, we have introduced the graph-based geometry aware convolutional filters that adapt

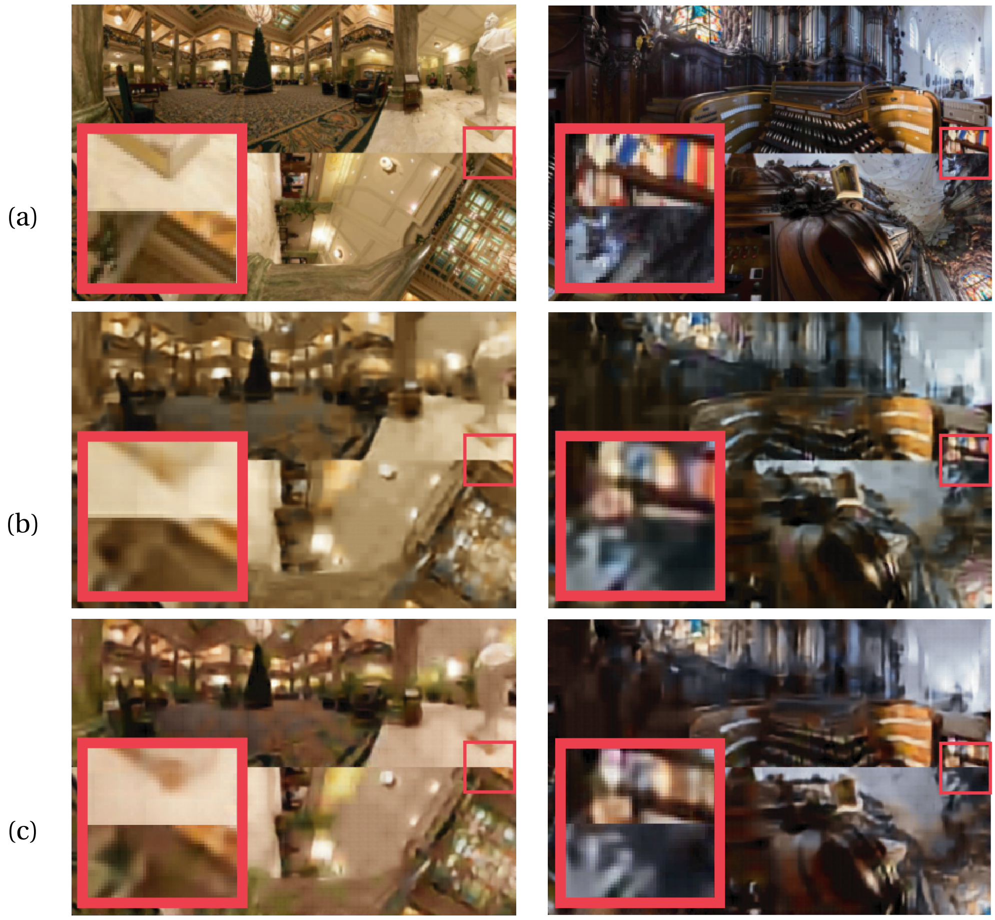


Figure 5.9 – **Cube-map projection's artifacts** appear due to the discontinuity between unwrapped face's borders. On the left corners of original (a) and decompressed images (b,c) we show zoomed version of the image patch that illustrates the borders of the cube faces. The decompressed results (c), obtained by our proposed anisotropic geometry-aware filters does not smooth border of the faces as our method has access to the information about cube-map geometry, while the result (b) obtained with convolutional filter from [90] smooths these borders. This smoothing can lead to perceptutally unpleasant result in various applications (e.g., virtual reality). (best seen in color)

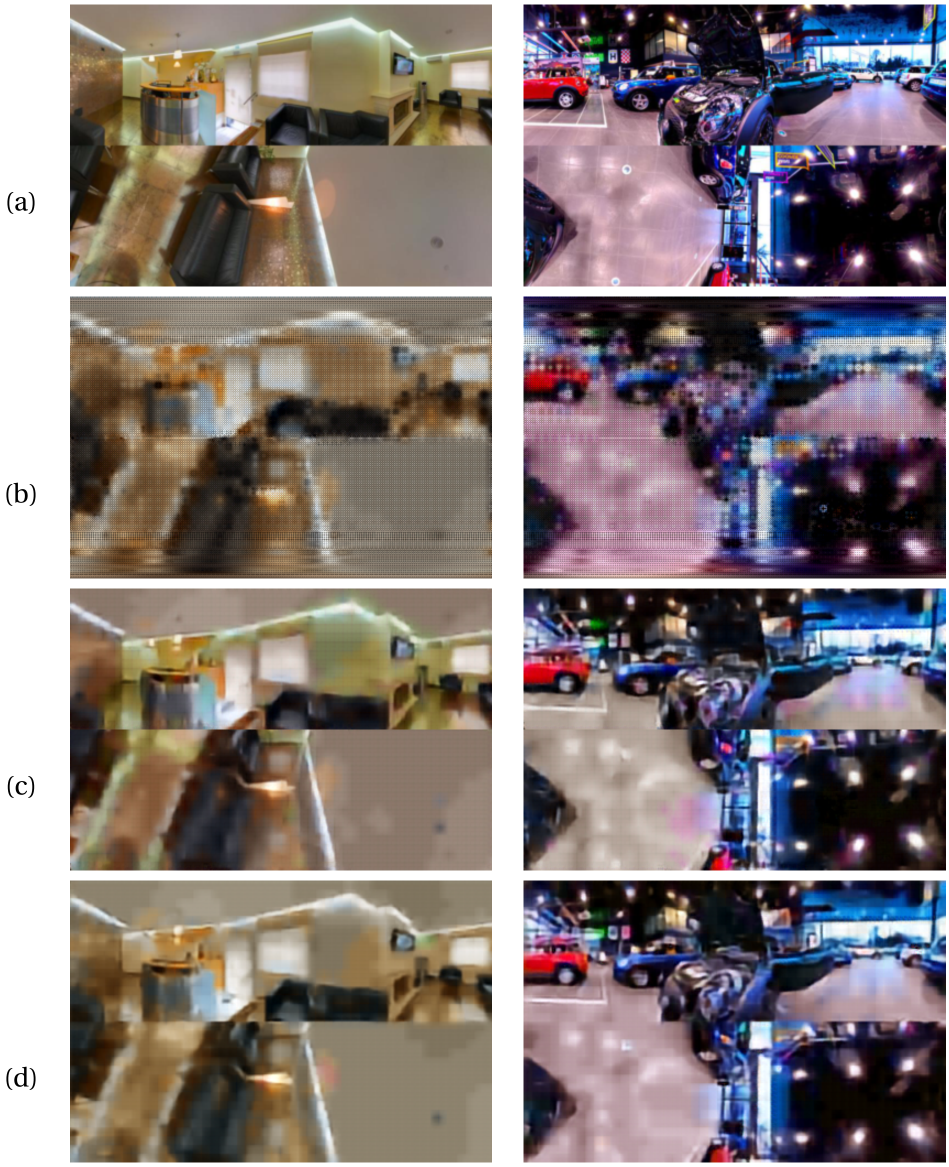


Figure 5.10 – **Decompression result**: original images (a); decompressed image obtained by our algorithm with isotropic geometry-aware (b), proposed anisotropic geometry-aware (c) and convolutional filters from [90] (d). (best seen in colors)

their shape and size to the geometry of the image projection surface. In contrast to many existing graph-based filters, our filters are anisotropic, which allows to better adjust to the specific properties of the problem. Our illustrative experiments show state-of-the-art performance of our approach applied to image classification and compression tasks in the presence of various types of image distortions.

6 Conclusion

6.1 Achievements

In this thesis we have addressed the problem of using prior knowledge about a given task to improve representation learning and proposed several frameworks that are all based on effective combination of deep networks and graph-based representation. In particular, we propose an architecture that achieves invariance to isometric transformations by using isotropic graph-based filters. Further, we extend our approach to work with omnidirectional images by modifying the weights of the edges of the graph in such a way that graph-based filters are able to adapt to distortions that appear at various elevations of equirectangular images. This permits to have a similar feature representation of an object regardless of its position on the image. Finally, we show that, even though isotropic filters are effective for such tasks, where isometric feature invariance is important, there exist other tasks (e.g. image compression) where correctly recovering the orientation of the image pattern is crucial and therefore relying on isotropic filter is detrimental to the overall quality of the approach. Therefore, we introduce another method that builds anisotropic graph-based filters. These filters are able to adapt their size and shape to any image projective geometry that can be encoded by a graph. Below, we summarize each of our contributions in more details.

Our first contribution resides in leveraging graph signal processing tools to create isometry invariant deep neural network for a classification task. To achieve this, we rely on the polynomial spectral graph filters that operate on an undirected graph. These filters are isotropic by nature and permit us to create feature representations of an input signal that are equivariant to rotations, translations and flips. This in turn permits to avoid training networks on all the possible transformations, because equivariance is inherently encoded inside the filter. To preserve this equivariance property we also introduce a special graph pooling layer that makes the signal sparser on the graph. Further, we propose a specific statistical layer, which collects the equivariant representation of the input signal and creates a new one that remains the same for all isometric transformations of an object. Our experiments show that our network is able to predict correct classification labels even when an object appears under unseen transformations. It further outperforms classical ConvNet architectures on various classification tasks.

Then, we extend our previous approach to omnidirectional camera geometry. We theoretically show that, under some assumptions about the image signal, we can change the weights of the graph in such a way that polynomial filters respond similarly regardless of the object locations and corresponding distortion effects. This property combined with our TIGraNet architecture permits to create a network that learns a feature representation that is invariant to geometric distortion effects in an omnidirectional camera.

Finally, we extend our framework to a wider class of tasks. We show that such problems as deep learning based compression require anisotropic filters to be able to decode sharp image details. We therefore introduce a principled way of building anisotropic graph-based filters that are able to adapt their size and shape to the geometry of omnidirectional images. In our experiments we show that our method performs well on both the classification and compression tasks and is able to adapt to different types of image projective geometry.

Overall, in this thesis, we have combined the power of deep learning and graph signal processing tools that altogether permit to incorporate prior knowledge about the target task inside the learning procedure.

6.2 Future directions

While this thesis has demonstrated the effectiveness of adding graph-based priors to the tasks solved by deep learning networks, there remain many opportunities for extending the scope of this work. In this section we discuss some of the possible future directions.

Scale invariance. In Chapter 3 we introduced the image classification system that is invariant to isometric transformations of the input signal. However, in the real world applications, objects not only be captured from different view points but can also appear at different distances from the camera. Therefore, an interesting extension of this work would be to design a system, which is also invariant to the scale changes of an object in a scene. This property of the network may lead to improving the accuracy and give the network the ability to correctly classify objects that appear at unseen scales. This will ultimately lead to further decrease in the size of the necessary training dataset and the amount of augmentation that is applied to the data.

Adaptation to any geometry. In Chapters 4 and 5 we propose approaches that adapt the filters of the network to different object elevations in omnidirectional images, assuming that the projective and sampling geometry is known. An interesting extension of these approaches is to have the network infer the underlying image geometry and/or camera parameters on its own. This is a considerably more complex problem for the network. Therefore, the algorithm may require more examples to make the network infer the image geometry. Such an approach, however, will allow the network to adapt to any type of images and will not require the camera calibration, which is often a tedious procedure.

Geometrical transfer learning. Another interesting direction is to train a network on one type of camera geometry (e.g., planar images) and then adapt this knowledge to another domain (e.g., fish-eye camera). This direction is very interesting as there exist a large number of datasets with standard (undistorted) images. However, to the best of our knowledge, there are very few datasets available that comprise images from fish-eye and omnidirectional cameras. In this case having a system that would be able to easily adapt to any kind of distortions will be very useful for various applications.

Other target tasks. In this thesis, we show that anisotropic filters permit solving a broader variety of problems as compared to isotropic ones. Therefore, another possible extension of the method proposed in Chapter 5 is to apply the suggested approach to different applications including but not limited to object detection, segmentation, noise reduction or omnidirectional images generation. All these omndirectional image problems may benefit from using the geometrical prior.

A An appendix

A.1 Modified spherical surface projection

In Chapter 5 in order to compute the projections of the MNIST digits on the modified spherical surfaces we have used the following projective mapping, which is a modification of the mapping defined in Eq. (4.2):

$$\begin{aligned} x &= \cos(\phi_i)\sin(\theta_i - \theta_0) \\ y &= (\cos(\phi_0)\sin(\phi_i + p(\phi_i, r, l)) - \sin(\phi_0 + p(\phi_0, r, l))\cos(\phi_i)\cos(\theta_i - \theta_0))/c, \\ c &= \sin(\phi_i + p(\phi_i, r, l))\sin(\phi_0 + p(\phi_0, r, l)) + \cos(\phi_i)\cos(\phi_0)\cos(\theta_i - \theta_0), \end{aligned} \tag{A.1}$$

where (x, y) are the coordinates on the tangent plane and $p(\phi, r, l)$ is the perturbation function that can be written as

$$p(\phi, r, l) = r\sin^{-1}(\sin(l\phi)), \tag{A.2}$$

where ϕ is the elevation level; r is the parameter that regulates the perturbation magnitude and l defines frequency of the perturbation signal. In our experiments we have set $l = 10$. Note that for a specific case of $r = 0$ we get the ordinary spherical surface.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[2] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Medical Image Computing and Computer-Assisted Intervention*, vol. 9351, 2015, pp. 234–241.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” in *Intelligent Signal Processing*, 2001, pp. 306–351.

[4] F. De Simone, P. Frossard, N. Birkbeck, and B. Adsumilli, “Deformable block-based motion estimation in omnidirectional image sequences,” in *International Workshop on Multimedia Signal Processing*, 2017, pp. 1–6.

[5] Z. Chen, Y. Li, and Y. Zhang, “Recent advances in omnidirectional video coding for virtual reality: Projection and evaluation,” vol. 146, 2018.

[6] R. Khasanova and P. Frossard, “Graph-based Isometry Invariant Representation Learning,” in *International Conference on Machine Learning*, vol. 70, 2017, pp. 1847–1856.

[7] Y. LeCun and Y. Bengio, “The handbook of brain theory and neural networks.” MIT Press, 1998, ch. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258.

[8] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep llearning: Going beyond euclidean data,” *Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[9] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains,” *Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[10] D. Thanou, D. I. Shuman, and P. Frossard, “Learning parametric dictionaries for signals on graphs,” *Transactions on Signal Processing*, vol. 62, no. 15, pp. 3849–3862, 2014.

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral Networks and Locally Connected Networks on Graphs,” in *International Conference for Learning Representations*, 2014.

[12] M. Henaff, J. Bruna, and Y. LeCun, “Deep Convolutional Networks on Graph-Structured Data,” *arXiv preprint*, 2015.

[13] J. S. Alexander and R. Kondor, “Kernels and regularization on graphs,” in *Conference on Learning Theory*, vol. 2777, 2003, pp. 144–158.

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

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

[16] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *International Conference on Learning Representations*, 2017.

[17] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, “Cayleynets: Graph convolutional neural networks with complex rational spectral filters,” *arXiv preprint*, 2017.

[18] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, “Autoregressive moving average graph filtering,” *Transactions on Signal Processing*, vol. 65, no. 2, pp. 274–288, 2017.

[19] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[20] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated Graph Sequence Neural Networks,” *arXiv preprint*, 2015.

[21] S. Sukhbaatar, A. Szlam, and R. Fergus, “Learning multiagent communication with backpropagation,” in *Advances in Neural Information Processing Systems*, 2016, pp. 2244–2252.

[22] F. Monti, K. Otness, and M. M. Bronstein, “Motifnet: a motif-based graph convolutional network for directed graphs,” *arXiv preprint*, 2018.

[23] A. R. Benson, D. F. Gleich, and J. Leskovec, “Higher-order organization of complex networks,” *Science*, vol. 353, no. 6295, pp. 163–166, 2016.

[24] L. Yi, H. Su, X. Guo, and L. J. Guibas, “Syncspeccnn: Synchronized spectral CNN for 3d shape segmentation,” in *Conference on Computer Vision and Pattern Recognition*, 2017.

[25] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst, “Geodesic Convolutional Neural Networks on Riemannian Manifolds,” in *International Conference on Computer Vision Workshops*, 2015, pp. 832–840.

[26] M. Andreux, E. Rodolà, M. Aubry, and D. Cremers, “Anisotropic laplace-beltrami operators for shape analysis,” in *European Conference on Computer Vision*, 2014, pp. 299–312.

[27] D. Boscaini, J. Masci, E. Rodolà, M. M. Bronstein, and D. Cremersl, “Anisotropic diffusion descriptors,” in *Computer Graphics Forum*, vol. 35, no. 2, 2016, pp. 431–441.

[28] D. Boscaini, J. Masci, E. Rodolà, and M. M. Bronstein, “Learning shape correspondence with anisotropic convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2016, pp. 3189–3197.

[29] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, “Geometric deep learning on graphs and manifolds using mixture model cnns,” in *Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5425–5434.

[30] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” in *International Conference on Learning Representations*, 2018.

[31] F. Monti, O. Shchur, A. Bojchevski, O. Litany, S. Günnemann, and M. M. Bronstein, “Dual-primal graph convolutional networks,” *arXiv preprint*, 2018.

[32] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, “Splinecnn: Fast geometric deep learning with continuous b-spline kernels,” in *Conference on Computer Vision and Pattern Recognition*, 2018.

[33] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” in *Conference on Computer Vision and Pattern Recognition*, 2017.

[34] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” in *Advances in Neural Information Processing Systems*, 2017.

[35] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, “Dynamic graph cnn for learning on point cloufds,” *arXiv preprint*, 2018.

[36] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz, “Splatnet: Sparse lattice networks for point cloud processing,” in *Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2530–2539.

[37] Y. L. Boureau, J. Ponce, and Y. LeCun, “A Theoretical Analysis of Feature Pooling in Visual Recognition,” in *International Conference on Machine Learning*, 2010, pp. 111–118.

[38] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, “Spatial Transformer Networks,” in *Advances in Neural Information Processing Systems*, 2015, pp. 2017–2025.

[39] D. A. Dyk and X.-L. Meng, “The art of data augmentation,” *Journal of Computational and Graphical Statistics*, vol. 10, pp. 1–111, 2012.

[40] B. Fasel and D. Gatica-Perez, "Rotation-invariant neoperceptron," in *International Conference on Pattern Recognition*, vol. 3, 2006, pp. 336–339.

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

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

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

[44] S. Dieleman, K. W. Willett, and J. Dambre, "Rotation-invariant convolutional neural networks for galaxy morphology prediction," *Monthly notices of the royal astronomical society*, vol. 450, no. 2, pp. 1441–1459, 2015.

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

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

[47] J. Bruna and S. Mallat, "Invariant scattering convolution networks," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1872–1886, 2013.

[48] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow, "Harmonic networks: Deep translation and rotation equivariance," *arXiv preprint*, 2016.

[49] M. Mathieu, M. Henaff, and Y. Lecun, "Fast training of convolutional networks through ffts," in *International Conference on Learning Representations*, 2014.

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

[51] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A Convolutional Neural Network for Modelling Sentences," *arXiv preprint*, 2014.

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

[53] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint*, 2014.

[54] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[55] I. Herstein, *Topics in Algebra*, 2nd ed. Wiley India Pvt. Limited, 2006.

[56] E. Landau, *Handbuch der Lehre von der Verteilung der Primzahlen.* Teubner, 1909.

[57] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

[58] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[59] B. Leibe and B. Schiele, "Analyzing Appearance and Contour Based Methods for Object Categorization," in *Conference on Computer Vision and Pattern Recognition*, 2003, pp. 409–415.

[60] E. Agustsson and R. Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study," in *Conference on Computer Vision and Pattern Recognition Workshops*, 2017.

[61] D. Scaramuzza, *Omnidirectional Vision: From Calibration to Robot Motion Estimation.* ETH, 2008.

[62] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision based MAV navigation in unknown and unstructured environments," in *International Conference on Robotics and Automation*, 2010, pp. 21–28.

[63] P. Hansen, P. Corke, W. W. Boles, and K. Daniilidis, "Scale invariant feature matching with wide angle images," in *International Conference on Intelligent Robots and Systems*, 2007, pp. 1689–1694.

[64] J. Masci, D. Migliore, M. M. Bronstein, and J. Schmidhuber, "Descriptor Learning for Omnidirectional Image Matching," in *Registration and Recognition in Images and Videos*, 2014, pp. 49–62.

[65] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[66] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint*, 2014.

[67] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3D Bounding Box Estimation Using Deep Learning and Geometry," *arXiv preprint*, 2016.

[68] L. Posada, K. K. Narayanan, F. Hoffmann, and T. Bertram, "Semantic classification of scenes and places with omnidirectional vision," in *European Conference on Mobile Robots*, 2013, pp. 113–118.

[69] T. Goedemé, T. Tuytelaars, L. V. Gool, G. Vanacker, and M. Nuttin, "Omnidirectional sparse visual path following with occlusion-robust feature tracking," in *Workshop on Omnidirectional Vision, Camera Networks and Non-classical Cameras*, 2005.

[70] D. Scaramuzza and R. Siegwart, “Appearance-Guided Monocular Omnidirectional Visual Odometry for Outdoor Ground Vehicles,” *Transactions on Robotics*, vol. 24, no. 5, pp. 1015–1026, 2008.

[71] L. B. Marinho, J. S. Almeida, J. W. M. Souza, V. H. C. de Albuquerque, and P. P. R. Filho, “A novel mobile robot localization approach based on topological maps using classification with reject option in omnidirectional images,” *Expert Systems with Applications*, vol. 72, pp. 1–17, 2017.

[72] I. Tosic and P. Frossard, “Spherical Imaging in Omnidirectional Camera Networks,” in *Multi-Camera Networks: Concepts and Applications*. Elsevier, 2009.

[73] C. Geyer and K. Daniilidis, “Catadioptric Projective Geometry,” *International Journal of Computer Vision*, vol. 45, no. 3, pp. 223–243, 2001.

[74] D. G. Lowe, “Object Recognition from Local Scale-Invariant Features,” in *International Conference on Computer Vision*, 1999, pp. 1150–1157.

[75] A. Torralba, “Contextual Priming for Object Detection,” *International Journal of Computer Vision*, vol. 5, no. 2, pp. 169–191, 2003.

[76] A. Rituerto, A. C. Murillo, and J. J. Guerrero, “Semantic labeling for indoor topological mapping using a wearable catadioptric system,” *Robotics and Autonomous Systems*, vol. 62, no. 5, pp. 685–695, 2014.

[77] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua, “LDAHash: Improved Matching with Smaller Descriptors,” *Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 1, pp. 66–78, 2012.

[78] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality Reduction by Learning an Invariant Mapping,” in *Conference on Computer Vision and Pattern Recognition*, 2006, pp. 1735–1742.

[79] J. Cruz-Mota, I. Bogdanova, B. Paquier, M. Bierlaire, and J. Thiran, “Scale Invariant Feature Transform on the Sphere: Theory and Applications,” *International Journal of Computer Vision*, vol. 98, no. 2, pp. 217–241, 2012.

[80] H. Hadj-Abdelkader, E. Malis, and P. Rives, “Spherical image processing for accurate visual odometry with omnidirectional cameras,” *Workshop on Omnidirectional Vision, Camera Networks and Non-classical Cameras*, 2008.

[81] C. Harris and M. Stephens, “A Combined Corner and Edge Detector,” in *Alvey Vision Conference*, 1988, pp. 1–6.

[82] I. Bogdanova, X. Bresson, J. Thiran, and P. Vandergheynst, “Scale Space Analysis and Active Contours for Omnidirectional Images,” *Transactions on Image Processing*, vol. 16, no. 7, pp. 1888–1901, 2007.

[83] Z. Arican and P. Frossard, “Dense disparity estimation from omnidirectional images,” in *International Conference on Advanced Video and Signal-based Surveillance*, 2007, pp. 399–404.

[84] I. Cinaroglu and Y. Bastanlar, “A direct approach for object detection with catadioptric omnidirectional cameras,” *Transactions on Signal, Image and Video Processing*, vol. 10, no. 2, pp. 413–420, 2016.

[85] I. Tosic, I. Bogdanova, P. Frossard, and P. Vandergheynst, “Multiresolution motion estimation for omnidirectional images,” in *European Signal Processing Conference*, 2005, pp. 1–4.

[86] F. De Simone, P. Frossard, P. Wilkins, N. Birkbeck, and A. C. Kokaram, “Geometry-driven quantization for omnidirectional image coding,” in *Picture Coding Symposium*, 2016, pp. 1–5.

[87] F. Pearson, *Map Projections Theory and Applications*. Taylor & Francis, 1990.

[88] K. Miyamoto, “Fish Eye Lens,” *Journal of the Optical Society of America*, vol. 54, no. 8, p. 1060, 1964.

[89] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. Cambridge University Press, 2007.

[90] J. Ballé, V. Laparra, and E. Simoncelli, “End-to-end optimized image compression,” in *International Conference on Learning Representations*, 2017.

[91] Y.-C. Su and K. Grauman, “Learning spherical convolution for fast features from 360-degree imagery,” in *Advances in Neural Information Processing Systems*, 2017, pp. 529–539.

[92] Y. Jeon and J. Kim, “Active convolution: Learning the shape of convolution for image classification,” in *Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1846–1854.

[93] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, “Deformable convolutional networks,” in *International Conference on Computer Vision*, 2017, pp. 764–773.

[94] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, “Spherical CNNs,” in *International Conference on Learning Representations*, 2018.

[95] B. Coors, A. P. Condurache, and A. Geiger, “Spherenet: Learning spherical representations for detection and classification in omnidirectional images,” in *European Conference on Computer Vision*, 2018.

[96] K. Tateno, N. Navab, and F. Tombari, “Distortion-aware convolutional filters for dense prediction in panoramic images,” in *European Conference on Computer Vision*, 2018.

[97] R. Monroy, S. Lutz, T. Chalasani, and A. Smolic, "Salnet360: Saliency maps for omnidirectional images with cnn," *Signal Processing: Image Communication*, 2018.

[98] M. Ruder, A. Dosovitskiy, and T. Brox, "Artistic style transfer for videos and spherical images," *International Journal of Computer Vision*, pp. 1–21, 2018.

[99] A. Sakiyama, T. Namiki, and Y. Tanaka, "Design of polynomial approximated filters for signals on directed graphs," in *Global Conference on Signal and Information Processing*, 2017, pp. 633–637.

[100] F. Bettonvil, "Fisheye lenses," *WGN, Journal of the International Meteor Organization*, vol. 33, pp. 9–14, 2005.

[101] R. Khasanova and P. Frossard, "Graph-based classification of omnidirectional images," in *International Conference on Computer Vision Workshops*, 2017.

[102] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimization of nonlinear transform codes for perceptual quality," in *Picture Coding Symposium*, 2016, pp. 1–5.

[103] J. Xiao, K. Ehinger, A. Oliva, and A. Torralba, "Recognizing scene viewpoint using panoramic place representation," in *Computer Vision and Pattern Recognition*, 2012.

Renata Khasanova

École polytechnique fédérale de Lausanne (EPFL)
EPFL-STI-IEL-LTS4
Station 11
CH-1015 Lausanne

Phone number: +41(0) 78 727-76-39
E-mail: renata.khasanova@epfl.ch
Date of birth: 08.02.1989
Website: http://lts4.epfl.ch/khasanova

Research interests: My research interests reside in the areas of (1) physically and/or biologically inspired algorithms for Artificial Intelligence and (2) extension of the deep neural networks to various weakly-explored domains, such as graph signal processing.

Work experience:

2017 — 2018 (6 months) **Google Research**, Zurich, Switzerland. *SWE Intern in Compression Team*

I explored new context, texture and noise models for the PIK image compression algorithm. I used Python to quickly prototype a large variety of approaches (including clustering, filtering, and optimization) and integrated the best candidate algorithms into the C++ code at: *https://github.com/google/pik*

2013 – present **EPFL**, Lausanne, Switzerland. *PhD in Computer Science and Teaching Assistant*

I am working towards the extension of the deep learning architectures to processing irregular data structures, which can be modelled as a graph.

Projects:
- Development of the graph-based deep learning architecture
- Development of the deep transformation-invariant features for object classification (presented on ICML'17 conference)
- Development of deep learning algorithms for efficient processing of raw images taken by the omnidirectional cameras (presented on ICCV'17 workshop)

Tools: Python, MATLAB

2011 – 2013 **Yandex,** Moscow, Russia. *Automated Test Engineer in Advertising Technology Department*

Project: Developing and implementation of the automated test system which is used to inspect the engine of the advertising system

Tools: Python, MS SQL

2011 (three months) **LG Research**, Moscow, Russia. *Algorithm Consultant, internship*

Project: Gyroscope noise suppression for robotic vacuum cleaner
Tools: MATLAB

2010 – 2011 (6 months) **Yandex School of Data Analysis**, Moscow, Russia. *Teaching Assistant*

Subject: Algorithms and Data Structure
Tools: C++

Education:

2013 – present **EPFL**, Lausanne, Switzerland. *PhD*
Electrical Engineering Institute, Signal Processing laboratory (LTS4)

2010 – 2012 **Bauman Moscow State Technical University (BMSTU)**, Moscow, Russia. *MSc*
Computer Science department, Computer-Aided Design program GPA: 4.9/5

2009 – 2011	**Moscow Institute of Physics and Technology**, Moscow, Russia. *Advanced Training* Computer Science department of Yandex School of Data Analysis, Internet Data Analysis programme
2006 – 2010	**Bauman Moscow State Technical University (BMSTU),** Moscow, Russia. *Bsc* Computer Science department, Computer-Aided Design program

Technical Skills:

C/C++, STL	**5 years** of experience in Google, Yandex School of Data Analysis, BMSTU
Python (Numpy, Scipy)	**7 years** of experience at Google, EPFL, Yandex, BMSTU
Theano	**2 year** of experience at EPFL
Tensorflow	**5 months** of experience at EPFL
Matlab	**2 year** of experience in BMSTU and LG projects

Publications:

- Renata Khasanova, Pascal Frossard, “Isometric Transformation Invariant Graph-based Deep Neural Network”, arXiv preprint, 2018
- Renata Khasanova, Jan Wassenberg, Jyrki Alakuijala, “Noise generation for compression algorithms”, arXiv preprint, 2018
- Renata Khasanova, Pascal Frossard, “Graph-Based Classification of Omnidirectional Images”, In IEEE International Conference on Computer Vision Workshops (ICCVW). Venice, Italy, 2017
- Renata Khasanova, Pascal Frossard, “Graph-based Isometry Invariant Representation Learning”, In International Conference on Machine Learning (ICML). Sydney, Australia, 2017
- Renata Khasanova, Xiaowen Dong and Pascal Frossard, “Multi-modal image retrieval with random walks on multi-layer graphs”. In IEEE International Symposium on Multimedia (ISM). San Jose, California, USA, 2016

Extracurricular:

- Google Student Blog post, 2018: https://students.googleblog.com/2018/11/getting-to-know-research-intern-renata.html
- ML Google summit, Switzerland, 2017
- EPFL & ETH Summer School, Key Insights in Networks and Graphs, Switzerland, 2015
- Microsoft computer vision school, Russia, 2011
- Yandex summer school in machine translation, Russia, 2010

Languages:

- English – fluent
- Russian – native
- French – B1
- Italian – A1

Ce document a été imprimé au Centre d'impression EPFL,
imprimerie climatiquement neutre, certifiée myClimate.