# Geometric deep learning for multimodal remote sensing data analysis

Student : **Marine MERCIER**
Supervisor : **Andrea MARINONI**
Referent teacher : **Florence TUPIN**

June 2021

# Contents

# Introduction

Global warming is one of the most sensitive contemporary issues. The increase of anthropogenic greenhouse gases in the atmosphere is the main cause of this rapid change in temperature, which favors the occurrence of natural, health or ecological disasters [8]. One of the major consequences of global warming is the melting of Arctic ice [9]. The Center for Integrated Remote Sensing and Forecasting for Arctic Operations uses remote sensing to monitor ice evolution for environmental and industrial purposes.

I would like to work in innovation for computer vision and remote sensing, on subjects related to environmental protection. After a first internship in a Parisian start-up, I wanted to discover the world of academic research. During this theoretical research internship at UiT the Arctic University of Norway, I worked on the use of Geometric deep learning for multimodal image analysis. Geometric Deep Learning was introduced by Bronstein et al. to work on non-Euclidean data [1]. This field of machine learning is in full development and its results are very promising.

In this report, I will present the main issues of Geometric deep learning, as well as its most classical tools. Then I will highlight two specific architectures, the Scalable Inception Graph Neural Networks architecture (SIGN) and Higher-Order Label Homogeneity and Spreading in Graphs (HOLS), on which I chose to focus during this internship. I will highlight how these methods, usually used on graphs representing social networks, have been adapted to perform pixel-based classification in multimodal remote sensing.

This report is divided in four parts. The first part presents the context of this research, the advantages of multimodal analysis and the datasets on which I worked. The second part explains the difference between classical deep learning and geometric deep learning. The third and fourth parts of this report each introduce a specific method of geometric deep learning, its adaptation to the problem of pixel classification and finally the potential outcomes.

# 1 Context

## 1.1 Working in CIRFA

Earth observation through remote sensing images allows to accurate characterization and identification of materials on the surface of Earth. The Center for Integrated Remote Sensing and Forecasting for Arctic Operations (CIRFA) aims at becoming a knowledge hub for research development on Arctic surveillance technologies. Its leading expertise in remote sensing takes advantage of the infrastructure that has been built up in Tromsø. The research carried out in CIRFA focuses on monitoring and forecasting variations of the sea ice cover in the Arctic. Indeed, the decrease in the extent of sea ice enables the development of industrial activities in the Artic, including oil and gas, shipping, fishing tourism and mining. Knowledge of ice evolution in the Arctic is also essential to validate and improve climate models, simulating future climate impacts on the Earth's environment. CIRFA will create knowledge and develop new methodologies and technical innovations which can enable safer maritime operations in the environmentally sensitive Arctic area. Among many innovations, CIRFA has contributed to the development of system processing ocean wind field from SAR, the modelling and prediction of oil-spill drift, detailed mapping of snow depth ...

## 1.2 Multimodal imaging

This mapping of icy Arctic water is obtained from satellite instruments, mainly space-borne radar systems [10]. Multiple and heterogeneous image sources can be available for the same geographical region: multispectral, hyperspectral, radar, multitemporal, and multiangular images can today be acquired over a given scene. Multimodal imaging refers to the combination of these signals in order to improve classification of the materials on the surface [11]. In particular, it is worth noting that each sensor can grasp different physical characteristics of the region of interest. Hence, integrating the information acquired by the multiple sensors can improve the understanding of the physical phenomena occurring on Earth surface. As such, the investigation of Arctic scenario can definitely benefit of multimodal remote sensing data analysis, so to enable a sustainable development of the region.

During this project, I focused on merging two specific imaging methods, hyperspectral imaging (HS) and LiDAR, for pixel based classification. In pixel-based classification, each individual image pixel is analysed and assigned to a class. The fusion of these imaging methods aims to combine the height information of LiDAR and the spectral information of HSI in order to improve the pixel classification performance. Indeed, hyperspectral image (HSI) is helpful to detect the type of the land cover, but is limited if different land covers are composed of the same material, for instance roads and roofs, because the spectral curves look alike. In this case, knowing the height of a structure thanks to LiDAR can be really useful. Therefore, by integrating the advantages of the two types of data, we get a better identification than with a single sensor.

- Hyperspectral sensors (HS) record the amount of reflected energy over a continuous and wide portion of the electromagnetic spectrum, therefore each pixel has many dimensions instead of the three assigned to primary colors (red, green, blue). Hyperspectral sensors collect information as a set of images, each image corresponds to a narrow wavelength range of the electromagnetic spectrum (range of 1-10nm). These images are combined into a three-dimensional data cube (X,

y, $\lambda$), where x and y represent the two spatial dimensions and $\lambda$ the spectral dimension.

- LiDAR, acronym of "light detection and ranging", is a method that measures ranges using a laser. It targets an object with a laser, then measures the time for the reflected light to return to the receiver. This method can be used to record the Digital Surface Model (DSM) that represent the elevation of the tallest surface at one point, or the Digital Terrain Model (DTM) that represent the elevation of the ground. Lidar can use ultraviolet, visible or near infrared light to target wide range of materials.

## 1.3 Datasets

For this work, I used two datasets in rural and urban areas to test the fusion effect of the proposed approach:

- The "Trento dataset" was taken from a rural area in Trento, a southern city in Italy. It includes an HSI and a LiDAR-derived DSM. Both sensors are equipped with a spatial resolution of 1 m and a spatial size of 600×166 pixels. The HSI has 63 spectral channels varying from 402.89 to 989.09 nm and its spectral resolution is 9.2 nm. This scene contains six different land covers: Apple Tree, Building, Ground, Wood, Vineyard, and Road.

- The "2018 Houston dataset" is composed of a hyperspectral image and a LiDAR derived Digital Surface Model (DSM), both at the same spatial resolution (2.5m). The size of the image is 1202x4768 pixels. The hyperspectral imagery has 48 spectral bands in the 380 nm to 1050 nm region. The data was acquired over the University of Houston campus and the neighboring urban area. The area contains 20 land cover classes : Healthy grass, Stressed grass, Artificial turf, Evergreen trees, Deciduous trees, Bare earth, Water, Residential buildings, Non-residential buildings, Roads, Sidewalks, Crosswalks, Major thoroughfares, Highways, Railways, Paved parking lots, Unpaved parking lots, Cars, Trains and Stadium seats.

- The "2013 Houston dataset" contains 144 hyperspectral bands, and one LiDAR band, it covers a spatial region of 349 x 1905 pixels with a spatial resolution of 2.5m per pixel. Data includes 15 labeled classes : grass-healthy, grass-stressed, grass-synthetic, tree, soil, water, residential, commercial, road, highway, railway, parking lot 1, parking lot 2, tennis court, running track.

These datasets represent benchmark for the remote sensing community, so that validation and assessment of multimodal remote sensing data analysis methods can be reliably conducted. As such, the geometric deep learning algorithms used in this work were first tested on these datasets. Nevertheless, future works will focus on employing the algorithms investigated in this project on other datasets, such as multimodal datasets acquired over Arctic regions.

# 2 From Classic Deep Learning to Geometric Deep Learning

## 2.1 Euclidean Domains

### 2.1.1 Deep Learning in Euclidean domains

Machine learning models have been particularly successful when dealing with signals in which there is an underlying Euclidean structure [1]. Supervised machine learning consist in estimating a function $f$ from some hypothesis class that fits well the outputs of a given training dataset and allows to predict the outputs of a previously unseen dataset [12]. Neural networks can produce a dense class of functions even when using a few layers of the simplest architecture like the Perceptron, therefore they are a good choice to represent functions. This property is known as Universal Approximation, but does not imply an absence of inductive bias. The inductive bias on $f$ is imposed through the construction of a function class $\mathcal{F}$ and the use of geometric priors. We want $f$ to be locally smooth, which means that if we slightly perturb the input, the output won't change much.

### 2.1.2 The curse of dimensionality

Modern data analysis uses more and more information, which implies working in high dimensions. The number of dimensions is the number of features per datapoint in the dataset. The curse of dimensionality refers to the behaviors that appears when working in high dimension, and which make the optimization of functions more complex [5]. The number of observations needed to ensure the regularity or local smoothness of the function is exponential in the dimension d. The number of samples needed to approximate even a simple class without over-fitting grows 'too quickly' as the input dimension increases. Another issue is that a high number of dimensions causes the distances metric to loose their meaning, especially when using the euclidean distance [19].

### 2.1.3 The geometric priors

In modern data analysis tasks, several assumptions are made on the unknown function that we try to estimate :

- Stationarity : the function is either invariant or equivariant with respect to translation. For instance in image classification we want an invariant function, able to classify the object no matter where it is located in the image, whereas in image segmentation we want an equivariant function, where the mask follows the transformation of the input image.

- Local deformation and scale separation : most task studied in computer vision are stable with respect to local deformations like local translation, changes in point of view, rotations ... It follows that we can extract sufficient information at a lower spatial resolution by downsampling local filters. Therefore machine learning often uses hierarchical models in which spatial resolution is progressively reduced.

These properties are referred to as geometric priors.

### 2.1.4 Convolutional neural networks

Stationarity and stability to local deformation give the classic architecture of most Deep Learning algorithms. Convolutional Neural Networks (CNN) consist of many convolutional layers, applying filters with shared weights to the dataset. These are equivariant layers. In moreover pooling layers, like average or max-pooling, are used to aggregate results into a single output thanks to the scale separation property. Therefore geometric priors are the additional structure providing a solution for the curse of dimensionality and explaining the success of convolutional neural networks. Last but not least, the decomposition of convolution functions, or filters, thanks to wavelet or Fourier transforms can be used to obtain stability to local deformations and ensure the regularization of the function. For instance, smooth functions have Fourier transforms that decay rapidly to zero.

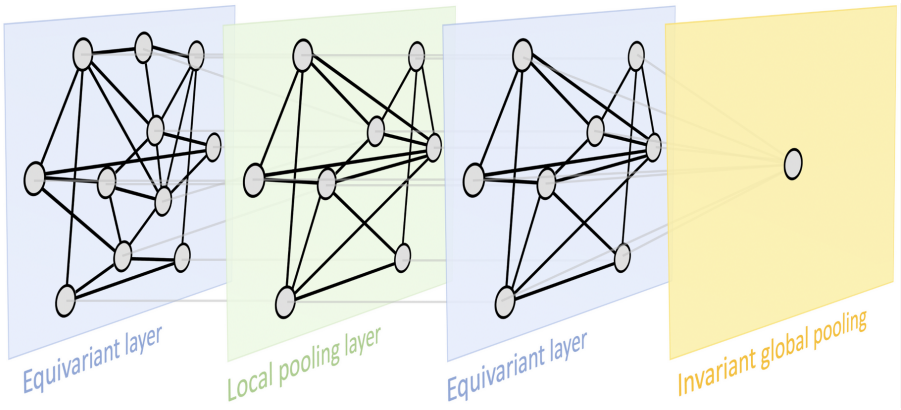



Figure 1: Geometric Deep Learning Blueprint [5]

Extending the CNN-type constructions to non-euclidean domains is the main goal of Geometric Deep Learning.

## 2.2 Non Euclidean domains

### 2.2.1 Graphs

Graphs are a non-euclidean type of data structure composed of nodes that are connected with edges and applied in a plethora of operational scenarios and research fields. This data structure represent individual features but also provides information regarding relationships between nodes since the edges also have weights.

Let $E$ be a collection of edges, $V$ the collection of nodes. We consider that all vertices of $V$ have the same weight. In the generic setting $E \neq \emptyset$, the graph connectivity can be represented by the $nn$ the weight matrix $W$, defined as $w_{uv} > 0$ if $(u, v) \in E$ and 0 otherwise, where $w_{uv}$ is the weight of the edge between $u$ and $v$. D is the degree matrix, defined by $D = diag(\sum_{u:u\neq v} W_{uv})$. $d_u$ s the degree of the node $u$.
$X_u$ are the features of node $u$. We are working with symmetric matrices so $(u, v) \in E \Leftrightarrow (v, u) \in E$. We denote by $A = D^{\frac{1}{2}} W D^{\frac{1}{2}}$ the normalized adjacency matrix. The normalized graph Laplacian is an

n×n positive semi-definite matrix $L = I - D^{\frac{-1}{2}} W D^{\frac{-1}{2}} = I - A$.

We also define the Hilbert spaces $L^2(V)$ and $L^2(E)$ such as :

$$< f, g >_{L^2(V)} = \sum_{u \in V} f_u g_u$$

$$< F, G >_{L^2(E)} = \sum_{(u,v) \in E} w_{uv} F_{uv} G_{uv}$$

The graph gradient is an operator $\bigtriangledown : L^2(V) \to L^2(E)$, defined as

$$(\bigtriangledown f)_{uv} = f_u - f_v$$

The graph divergence is the operator $div : L^2(E) \to L^2(V)$, defined as

$$(divF)_u = \sum_{u:(u,v) \in E} w_{uv} F_{uv}$$

THe gradient and divergence operators are adjoint with respect to the inner products.

$$< F, \bigtriangledown f >_{L^2(E)} = < -divF, f >_{L^2(V)}$$

The graph Laplacian is associated with the graph laplacian operator $\Delta : L^2(V) \to L^2(V)$, $\Delta = -div \bigtriangledown$.

$$(LX)_u = (\Delta X)_u = \frac{\sum_{(u,v) \in E} a_{u,v}(x_u - x_v)}{d_u}$$

The Laplacian operator can be interpreted as the difference between the local average of features around a point and the features at the point itself. It is used to describe physics phenomena like heat diffusion or wave propagation as it appears in Newton's law of cooling.

### 2.2.2 Geometric deep learning

Geometric deep learning is a new field of machine learning that can learn from complex data like graphs and multi-dimensional points which are non-euclidean [1]. In my work, I focused on graphs, but grids, groups, geodesics and gauges are also geometric domains.

A non-Euclidean environment has no dimensions, there is no more notion of right, left, up or down. In graphs, nodes are connected to an arbitrary number of nodes but dimension in the traditional sense of machine learning can refer to the number of features of each node. The collection of nodes is not provided in any particular order, therefore the functions acting on graphs should satisfy permutation invariance, which means that for two isomorphic graphs, the outcomes of these functions should be the same. For a functions on nodes, we want permutation equivariance, which means that we want the output of the function to be linked with the order of nodes [26].
Permutation invariance for graphs means that for any permutation matrix $P$,

$$f(PX, PAP^T) = f(X, A)$$

.
Permutation equivariance means that for any permutation $P$,

$$f(PX, PAP^T) = Pf(X, A)$$

.
The neighbourhood of a node $u$ is defined as $N_u = v : (u, v) \in E$. The neighbourhood features is the multiset containing the features of all the neighbourhood of a node $X_{Nu} = x_v : v \in N_u$. The 1-hop neighbourhood aligns well with the locality aspect of the geometric deep learning blueprint. Let $\phi(x_u, X_{Nu})$ be local function that works over the features of a node and its neighbourhood. Then we are looking for a permutation equivariant function $f$ that can be constructed using $\phi : f(X, A) = [\phi(x_1, X_{N1}), \phi(x_2, X_{N2}), ...\phi(x_n, X_{Nn})]$. If $\phi$ is independent of the ordering of nodes in $N_u$ then $f$ is permutation equivariant.

### 2.2.3 Eigendecomposition using the Laplacian

The Dirichlet energy is a measure of how smooth a function is [1]. It can therefore be used as a regularisation. Given a function $f$ on the domain $\mathcal{X}$, the Dirichlet energy measures how smooth it is:

$$\varepsilon_{Dir}(f) = \int_{\mathcal{X}} \| \bigtriangledown f(x)\|^2_{T_x\mathcal{X}} dx = \int_{\mathcal{X}} f(x)\Delta f(x) dx$$

where $T_x\mathcal{X}$ is the tangent space of $\mathcal{X}$ at the point $x$

The orthogonal basis on $\mathcal{X}$ containing the k smoothest possible functions is obtained by solving the following optimization problem.

$$min_{\phi_0} \varepsilon_{Dir}(\phi_0) s.t. \|\phi_0\| = 1$$

$$min_{\phi_i} \varepsilon_{Dir}(\phi_i) s.t. \|\phi_i\| = 1, \phi_i \perp span\phi_0, ...\phi_{i-1}, i = 1, 2, ...k-1$$

In the euclidean domain, this basis is the Fourier basis composed of sinusoids of increasing frequency.

In our problem, $\mathcal{X} = V$ the domain of the vertices is sampled at $n$ points and the norm is on $L^2(V)$. The problem can be written :

$$min_{\phi_k \in \mathbb{R}^{n \times k}} trace(\Phi_k^T \Delta \Phi_k) s.t. \Phi_k^T \Phi_k = I$$

where $\Phi_k = (\phi_0, ...\phi_{k-1})$ and $I$ the identity matrix.

The solution is given by the k eigenvectors of $\Delta$ associated with the smallest eigenvalues $0 = \lambda_0 \leq \lambda_1 \leq ... \leq \lambda_{k1}$ such that $\Delta\Phi_k = \Phi_k\Lambda_k$ with $\Lambda_k = diag(\lambda_0, ..., \lambda_{k1})$. The eigenvalues can be interpreted as the frequencies of the smoothest functions over $V$, and $\phi_0$ is the constant function [1].

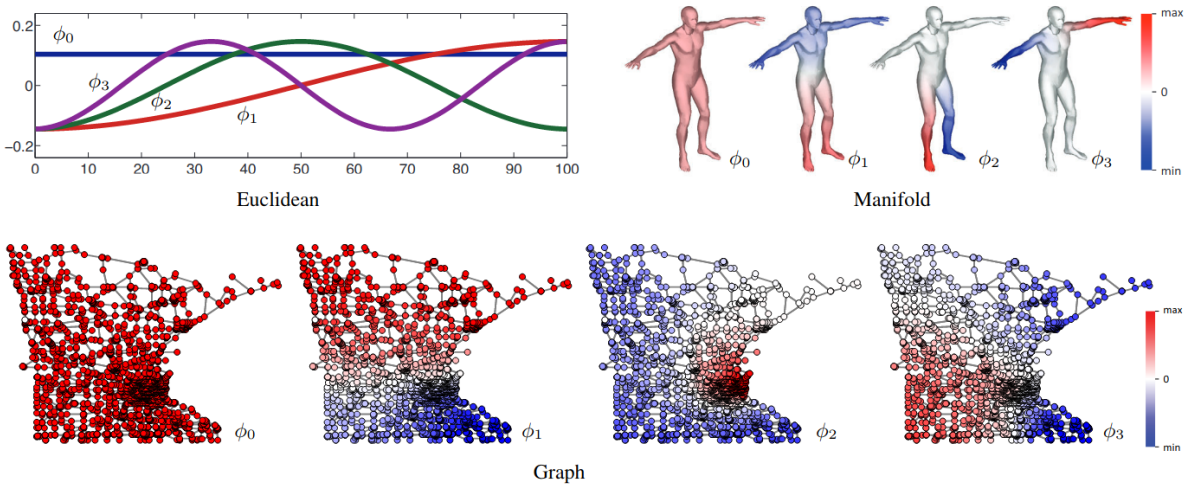



Figure 2: Examples of the first four Laplacian eigenfunctions on different domains [1]

### 2.2.4 Convolution like operators on graph

From the properties defined above, we can construct different operators on graphs that are similar to convolutions [20].

- Spectral CNN are using the link between the Laplacian decomposition and the Fourier transform to generalize CNN to graphs [13]. However this approach has some important limitations, the first being its limitation to a single domain. The coefficient of the spectral filters are basis dependent, and a small change in the basis can therefore lead to very different results, it can't be generalized across graphs. Another key drawback is the high complexity of this method is the high complexity, at least $O(n^2)$ with $n$ the number of nodes. Last but not least, the graph has to be undirected, for the Laplacian to be symmetric and diagonalizable.

- ChebNet considers the fact that a polynomial of the Laplacian acts on its eigenvalues. Rather than working in the frequency domain, it is possible to represent filters via polynomial expansion [14]. Filters computed with this method are stable under graph perturbation. Therefore convolution can be computed in the spatial domain using $Y = \tilde{g}(\Delta)X = \sum_{k=0}^{r} \theta_k \Delta^k X$, where $\tilde{g}$ is a filter function and $\theta_k$the coefficients of its polynomial expansion. This representation automatically creates localized filters as Laplacian because Laplacian is a local operator acting on 1-hop neighborhood, and its j$th$ power is working on j-hops. As a consequence a combination of powers behave like a diffusion operator. This decomposition in polynomials can in particular be carried out thanks to the Chebyshev polynomials, hence the name of this method. It can be computed in $O(nr)$.

- Graph Convolution Network works like ChebNet with r=1, the models only uses the node and its neighbors features [34]. $Y = \tilde{D}^{\frac{1}{2}}\tilde{W}\tilde{D}^{\frac{1}{2}}X\Theta = \tilde{A}X\Theta$ with $\tilde{W}$ the weight matrix with self loops, $\tilde{D}$ the matrix degree associated and $\tilde{A}$ the normalized adjacency matrix associated.

### 2.2.5 Node sampling

Graph sampling can be used to scale Graph Neural Networks (GNN) to large graphs. Indeed, for huge graphs the diffusion matrix cannot be stored in memory.

- Node-wise sampling consists of applying convolutions on partial node neighborhoods and is paired with mini-batch training [16]. A batch of nodes is chosen, and expanded with the most relevant neighbors, they can be selected thanks to pageRank algorithm for instance, then the loss is optimized on this selection of nodes.
- Layer-wise sampling avoid the exponential growth of neighborhood size with number of hops known as the small world effect [17]. In each layers, nodes only have edges directed towards nodes of the next layer. This method avoid the replication of nodes due to common neighbors.
- Graph-wise sampling uses subgraph as batches, and optimizes the GNN over all the nodes of the subgraph at each step [18]. Different methods can be used to sample subgarphs during the pre-processing.

# 3 SIGN - Scalable Inception Graph Neural Networks architecture

## 3.1 Theory

Most of the graph convolution techniques presented above only work on small graphs, or involve graph sampling in order to reduce the computational and memory complexity. As its name suggests, the SIGN method is scalable and, without using any sampling method, achieves effective training and low inference time on large graphs.

### 3.1.1 The SIGN architecture

This new architecture [2] proposed by F. Frasca and E. Rossi is effective on large graphs. It uses a set of diffusion operators $A_1, ..., A_r$, which are $n$ x $n$ matrices, $n$ being the number of nodes in the graph, applied to the node-wise features $X$. The products $A_1X, ..., A_rX$ can be precomputed. The architectures for node-wise classification looks like :

$$Z = \sigma([X\Theta_0, A_1X\Theta_1, ..., A_rX_r]),$$

$$Y = \xi(Z\Omega),$$

where $\Theta_0, ..., \Theta_r$ and $\Omega$ are learnable matrices while $\sigma$ and $\xi$ are non-linearities. This architecture allows to precompute $A_1X, ...A_rX$ which considerably reduces its computational complexity.

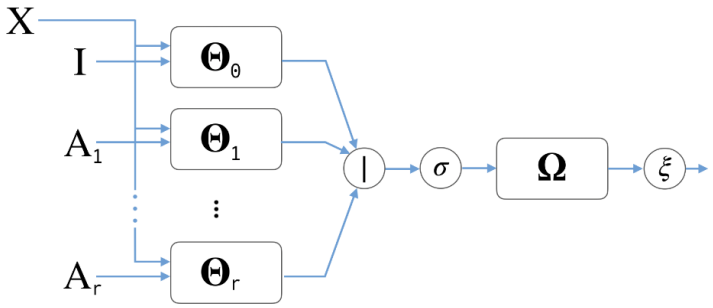



Figure 3: The SIGN architecture [2]

Unlike most graph convolution methods, SIGN doesn't use graph sampling techniques, operations which may create a bias.

Last but not least, this architecture can be used to learn automatically ChebNet, GCN and others classic Graph Convolution Methods if the right diffusion operator, non-linearity and activation function are used.

| | $\mathbf{B}_1, \ldots, \mathbf{B}_r$ | $\alpha$ | $\mathbf{\Theta}_0, \ldots, \mathbf{\Theta}_r$ | $\mathbf{\Omega}$ |
|---|---|---|---|---|
| *ChebNet* [15] | $\mathbf{\Delta}, \ldots, \mathbf{\Delta}^r$ | 1 | $\mathbf{\Theta}_0, \ldots, \mathbf{\Theta}_r$ | $[\mathbf{I}, \ldots, \mathbf{I}]^\top$ |
| *GCN* [34] | $r = 1,\ \tilde{\mathbf{A}}$ | 1 | $\mathbf{0}, \mathbf{\Theta}$ | $[\mathbf{0}, \mathbf{I}]^\top$ |
| *S-GCN* [59] | $r = 1,\ \tilde{\mathbf{A}^L}$ | 1 | $\mathbf{0}, \mathbf{\Theta}$ | $[\mathbf{0}, \mathbf{I}]^\top$ |

Figure 4: By appropriate configuration,the SIGN architecture is able to replicate some popular graph convolutional layers depending on the choice of the operator B such as $A_k = B^k$. The $\sigma$ non-linearity is set to PReLU and $\alpha$ represents the learnable parameter of the PReLU activation function.[2]

### 3.1.2 Choice of operators

Many different diffusion operators could be used to define B such as $A_k = B^k$, for $k = 1, ...r$.

The first intuition is to use normalized adjacency matrix or Laplacian matrix as well as their powers.

It is also possible to use operators induced by triangles or cliques, such as triangle-based adjacency matrix, which are efficient for representing weak or strong ties. In a triangle-based adjacency matrix, two nodes are connected by an edge if they are both part of a common triangle. In my work the transformation of images into graphs created fully connected graphs. Therefore this operator is not relevant for our problem.

The Personalized PageRank (PPR) algorithm was historically used by Google to rank web pages by measuring their importance. The algorithm uses Markov Chains to answer this problem, the initial probability distribution being $\Pi_0 = \frac{I}{n}$ and the transition matrix being $T = WD^{-1}$. We can imagine it as a random walker evolving on web-pages, choosing randomly his initial node, and jumping randomly to another node according to the adjacency matrix. The probability distribution evolves at each step with $\Pi_{t+1} = T\Pi_t$. To prevent dead end, the algorithm introduce a teleportation notion. At each step, there is a probability $(1 - \alpha)$ that the walker jumps according to the transition matrix, and a probability $\alpha$ that he jumps to some other nodes in the graph. $\alpha$ is often set between 0.1 and 0.2. The new transition matrix is defined as $R = (1 - \alpha)T + \alpha e$. $e$ is often equal to $\frac{I}{n}$, which means that the random walker can be teleported to every node with the same probability, but it is also possible to personalize the page rank algorithm by allowing teleportation only to certain nodes by changing $e$. $e$ can for instance be defined as a one hot encoder pointing toward one specific node. If any node can be reached from any other node, the probability distribution will converge to a stationary distribution $\Pi$ such as $\Pi = R\Pi$.

The heat kernel matrix of the graph represents the way in which information travels through the edges of the graph over time. We are interested in the classic heat equation solution ht : $\frac{\partial h_t}{\partial t} = \Delta h_t = -Lh_t$ where $t$ is time. The solution to this equation is $h_t = e^{tL}$. Since $L = I–A$, $h_t = e^{t(I-A)}$, which can be expanded as a polynomial $h_t = e^t \sum_{k=0}^{\infty} \frac{A^k t^k}{k!}$.

General graph diffusion is defined in [3] via the diffusion matrix $S = \sum_{k=0}^{\infty} \theta_k T^k$ with weighting coefficients $\theta_k$ and transition matrix $T$, chosen to ensure that $S$ converges. Given the two diffusion processes described above, we can define two diffusion matrices as the following :

- we define the PPR diffusion matrix with $T = WD^{-1}$ and $\theta_k = \alpha(1 - \alpha)^k$,

- we define the heat diffusion matrix with $T = WD^{-1}$ and $\theta_k = e^t \frac{t^k}{k!}$.

## 3.2 Implementation

### 3.2.1 From images to graphs

I transform the multimodal imaging datasets into graphs. Each pixel is a node, and the hyperspectral and LiDAR data associated with the pixels are its features. I defined the adjacency matrix of the graph as the distance matrix between pixels features, a square matrix containing the pairwise distances between the elements of the set. This matrix is symmetric, therefore the graph is undirected. The graph is fully connected, and the weight of an edge between two nodes or pixels is equal to the distance between these two nodes.

Every feature is scaled, therefore no dimension has more weight than the others. I have tested different scalers including Min Max Scaler, Standard Scaler, Max Abs Scaler and Robust Scaler. Indeed, one dimension shouldn't impact the model just because of its large magnitude. I chose the Standard Robust and, which is robust to outliers.

| Scaler | MaxAbs | MinMax | Standard | Robust |
|---|---|---|---|---|
| Accuracy Score | 72.51 | 72.6 | 73.32 | 74.38 |

Figure 5: Mean scores over ten runs of training on Houston, depending of the scaling chosen

Then we calculate the graph adjacency matrix using the Gaussian kernel for Euclidean distance. For two pixels $u$ and $v$, $w_{u,v} = e^{\frac{-\|u-v\|^2}{2\sigma^2}}$ with $\sigma$ the standard variation of the distance distribution. Last but not least, I normalized the adjacency matrix using the degree matrix : $A = D^{\frac{-1}{2}} W D^{\frac{-1}{2}}$.

This transformation of images generates very large graphs, which can be difficult to store and manipulate. One way to reduce the size of the diffusion operator matrices was to use superpixels as nodes of the graph. Among different segmentation algorithm, I tested waterpixels, a watershed based segmentation, and SLIC, an adaptation of k-mean to create superpixels [25]. However, for the following of the research to work, the superpixels must contain only pixels of the same class, which is in itself an important research topic. After some experiments, we decided to leave this part for possible future developments.

In node-wise predictions problem, there is a distinction between transductive and inductive settings. Transductive setting is when training and testing are performed on the same graph, which is our case. On the contrary, inductive setting is when different graph are used for training and testing.

### 3.2.2 The general code

We tested the architecture with different configurations for each of the diffusion operators described. The model is implemented by replacing the single layers performed by $\theta_i$ and $\Omega$ modules with multiple layers. The loss is calculated by using softmax and cross entropy loss. The softmax function produces a vector of the size of the number of class with components between 0 and 1. The softmax function

is defined as the following for each $i \in C$, C being the list of possible labels, $f(S)_i = \frac{e^{S_i}}{\sum_{j \in C}(e^{S_j}}$. Then the cross entropy loss is calculated as the following : $Loss = -\sum(L_i log(S_i))$ with S the output of the softmax function and L the labels expected in a hot vector encoder. The higher the cross-entropy loss, the less certain the model. The optimization of the model is done with the Adam optimized by minimizing this loss on mini-batchs. The data are normalized for each mini-batch.

The Adam optimizer is an update to the RMSProp (Root Mean Square Propagation) optimizer [24], in which the learning rate is adapted for each parameter. The gradient of the cost function evaluated on a random batch of data can be considered as a random variable. The first moment is the mean, the second is the variance. Let $g_t$ be the gradient of the function we want to minimize on the mini batch $t$, then the mean and the variance are defined as the following :

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t$$

.

We want $\mathbb{E}[m_t] = \mathbb{E}[g_t]$ and $\mathbb{E}[v_t] = \mathbb{E}[g_t^2]$, but, in the current state, these estimators are biased : $m_t = (1-\beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i$ so $\mathbb{E}[m_t] = \mathbb{E}[g_t](1-\beta_1^t)+\xi$ with $\xi$ the error emerging from the approximation of $g_i$ with $g_t$. Therefore $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ and $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$. Then, we use these moving average to update the learning rate for each parameters.

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where $w$ is the model weight, $\eta$ the step size and $\epsilon$ a small scalar used to prevent division by 0.

## 3.3 Testing of the architecture

### 3.3.1 Results

**Setup :** Only 5% of the dataset is used for training, 5% for validation, and 90% for testing. The hyper-parameters in the basic setup are the following : 2 layers, 256 hidden channels, a dropout of 0.2, learning rate of 0.0001 and batch size of 100. Each run is composed of 100 epochs, and for all experiences I do 10 runs and average the results. The effects of each hyper-parameter will be described in detail in the following part.

**Running time :** The computational time might be inconsistent because the machine used to do the calculus was shared with other people.

The prepossessing is quite long, especially when using the heat and page rank diffusion operators.

| Powers for diffusion operators | | | | Running time (s) | |
|---|---|---|---|---|---|
| Adjacency | Laplacian | Heat Kernel | Page Rank | Houston | Trento |
| 1 | 0 | 0 | 0 | 2.09 | 439.13 |
| 0 | 1 | 0 | 0 | 2.06 | 433.77 |
| 0 | 0 | 1 | 0 | 25.92 | 9631.08 |
| 0 | 0 | 0 | 1 | 25.73 | 9690.66 |
| 3 | 0 | 0 | 0 | 2.47 | 440.35 |
| 0 | 3 | 0 | 0 | 1.69 | 433.43 |
| 0 | 0 | 3 | 0 | 25.02 | 10102.31 |
| 0 | 0 | 0 | 3 | 23.46 | 10160.74 |
| 1 | 1 | 0 | 0 | 2.37 | 507.57 |
| 1 | 1 | 1 | 0 | 24.26 | 10035.83 |
| 1 | 1 | 0 | 1 | 24.12 | 9570.79 |
| 3 | 3 | 0 | 0 | 2.26 | 620.44 |
| 3 | 3 | 3 | 0 | 25.32 | 11936.66 |
| 3 | 3 | 0 | 3 | 28.43 | 13666.79 |
| 0 | 5 | 0 | 0 | 6.67 | 439.68 |
| 0 | 8 | 0 | 0 | 11.38 | 453.06 |

Figure 6: Preprocessing running time with different operators

**Results :** The Laplacian operator got the best results, the scores improving with the number of power used. Using heat kernel and page rank does not improve accuracy and need an higher preprocessing computational time. According to [2], the SIGN architecture is faster than other convolutionnal graph network on large graphs. It would also be interesting to compare the computational time of this SIGN method with other pixel classifications methods.

| Operators chosen | Adjacency power 3 | Laplacian power 3 | Page Rank power 3 | Heat kernel power 3 |
|---|---|---|---|---|
| Test accuracy | 95.7 | 96.55 | 95.44 | 95.06 |
| Running time (s) | 271.17 | 249.86 | 101.86 | 300.86 |

Figure 7: Mean scores and running time depending of the diffusion operators used, dataset Trento

I also tried to combine different diffusion operators.

| Adjacency power | Laplacian power | Heat Kernel power | Page Rank power | Test accuracy | Running time (s) |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 68.47 | 7.13 |
| 0 | 1 | 0 | 0 | 71.59 | 7.21 |
| 0 | 0 | 1 | 0 | 70.29 | 6.89 |
| 0 | 0 | 0 | 1 | 70.76 | 6.87 |
| 1 | 1 | 0 | 0 | 72.05 | 8.22 |
| 1 | 1 | 0 | 1 | 72.55 | 7.92 |
| 1 | 1 | 1 | 0 | 72.39 | 7.96 |
| 3 | 0 | 0 | 0 | 69.12 | 7.51 |
| **0** | **3** | **0** | **0** | **74.17** | **7.94** |
| 0 | 0 | 3 | 0 | 68.99 | 7.95 |
| 0 | 0 | 0 | 3 | 69.29 | 7.94 |
| **3** | **3** | **0** | **0** | **74.09** | **8.53** |
| 3 | 3 | 3 | 0 | 72.2 | 8 |
| 3 | 3 | 0 | 3 | 72.76 | 7.85 |
| 5 | 0 | 0 | 0 | 67.56 | 8.44 |
| **0** | **5** | **0** | **0** | **74.96** | **8.37** |

Figure 8: Mean scores and running time depending of the diffusion operators used, dataset Houston

The laplacian operator outperformed the other considered operators, and doesn't need any other diffusion operator. I realized that the more power of the Laplacian were used, the better the score get. It is more interesting to go as wide as possible rather than multiplying the different diffusion operators which can become repetitive. "Going wide" means using higher powers for the diffusion operator. Using high power of diffusion operators improves the accuracy scores. However, using high power of the diffusion operator also increases the computational complexity, a trade-off is necessary.

**Tested development paths :** I had the occasion to test many unsuccessful ideas to improve this algorithm.

- I used the *stratify* option of the python function *train-test-split* in order to create training, test and validation datasets containing the same proportions of each labels as the original dataset. In the original code, the split between the training, testing and validation dataset was done randomly.
- I added the possibility to adapt the weight for each label during the training. More specifically, I tried to balance the power given to each category. A more important weight is thus given to the rarer labels, in order to balance the learning on each category.
- I tried to add spatial information, which is the coordinates (x,y) of the pixels in the image, to the features. Thus each node contains spectral and spatial information of the corresponding pixel.

- I reduced the dimension of the features using Principal Component Analysis. It reduced the computational time effectively but the accuracy scores were considerably diminished.

These ideas, however, have not led to conclusive improvements in training.

**Limitation and future work :** The results are really different depending of the dataset, and I have not yet been able to find a valid reason.

The accuracy scores on Trento are clearly better than the ones on Houston 2013. Possible reasons why are that Houston 2013 is much smaller than Trento (2832 vs 30214 pixels). The number of features for each pixel is also bigger (144 hyperspectral bands and one Lidar for the Houston dataset compared to 63 HIS and 14 Lidar bands for the Trento dataset). Therefore we have less samples, with more information per sample, which can lead easily to over fitting. Also I noticed that the algorithm had trouble classifying specific categories. Looking at the confusion matrix, I noticed that the labels parking lot 1, parking lot 2 and road were often mixed.

Moreover, the algorithm performed really badly in Houston 2018, and was not learning at all. It might be due to a code problem, but despite my efforts I couldn't solve it. This dataset has lots of samples, I couldn't compute the algorithm on so many pixels. Therefore I had to select 30000 pixels randomly, while keeping the distribution of labels. The distribution of labels is heavily unequal compared to the previous datasets, so I thought it might be the reason why the algorithm was struggling. I have tried many different solutions to solve this problem :

- I tried to balance the labels of each class, and to get rid ot the classes with to few elements.
- On the other hand I tried to unbalance the label distribution in Trento's. The results of this training where not as good as usual, but still better than the ones on Houston 2018 and the algorithm was learning, therefore the problem is not the unbalanced labels.
- I thought that the euclidean distance used in the calculation of pairwise distance was maybe the problem. On one hand the problem is high dimension, the euclidean distance might not be the optimal choice because of the curse of dimensionality. On the other hand, euclidean distance worked well on the other datasets. I tested other classical distances such as Manhattan distance metric or fractional distance metric.
- I also tried to modify the Gaussian kernel, the standard deviation of pairwise distances on this dataset was larger than the one of the other dataset and was crushing the distances to 0.

An important point of development would be to determine the cause of these differences between performances on different datasets, and to find a solution to this problem.

### 3.3.2 Choice of hyper-parameters

**Over fitting** is a classic error in machine learning, that occurs when the estimated function is too closely aligned with a a limited amount of training points. Therefore the function is useless for predicting other points than the training dataset. Over fitting can be detected when the training, testing and validation curves diverge [21].

The **learning rate** define the size of the step taken during back-propagation. If the learning rate is too big, we might miss the minimum by jumping over it, it also increases the odds of over fitting . If the learning rate is too small, we might find a local minimum, the model convergence to a local minima will be slower [22].

The **batch-size** indicates the number of sample in each batch. Knowing that the loss is optimized on mini-batches, the batch-size is the number of samples "seen" by the algorithm between two optimizations. A small batch-size can generate over fitting, and instability in the learning curves. Intuitively, large batches would converge to a better estimation since they are more representative of the actual dataset. Nonetheless, optimization of non convex function is not that simple, and small batches converge to a "flat minima" which tend to give a better generalization, as explained in [7]. We can see in our experiment that reducing the batch size increases a little over fitting but still gives better results, but with a higher computational cost.

The **number of layers** qualifies the depth of the deep learning architecture. Each layer trains on a distinct set of features based on the output of the previous layer. The Sign architecture has at least two layers, the input and output layer. One of the strengths of this algorithm is that going deep, which means adding lots of layers, is not useful and going wide, using powers of diffusion operators is more important.

| Number of layers | 7 | 5 | 3 | 2 |
|---|---|---|---|---|
| Accuracy Score | 96.67 | 97.32 | 96.57 | 96.66 |
| Running time (s) | 171.68 | 135.84 | 65.81 | 142.86 |

Figure 11: Mean scores and running time over ten runs of training on Trento, depending of the number of layers

| Number of layers | 7 | 5 | 3 | 2 |
|---|---|---|---|---|
| Accuracy Score | 69.36 | 74.69 | 75.02 | 73.22 |
| Running time (s) | 12.15 | 14.53 | 13.44 | 13.57 |

Figure 12: Mean scores and training running time over ten runs of training on Houston 2013, depending of the number of layers

The **number of hidden channels** is the size of inputs and outputs in the hidden layers. The input size of the first layer is the number features given to the model, and the output size of the last layer is the number of classes. The size of inputs and output layers in between can be seen as the number of characteristics that we want to learn how to identify.

The **dropout** defines the proportion of layers output which are randomly ignored. It helps avoiding over-fitting and forces the network to learn more robust features, however it slows down the training considerably. The best effect is obtained with 0.5, too much dropout is useless, as lots of connections would be cut [23].

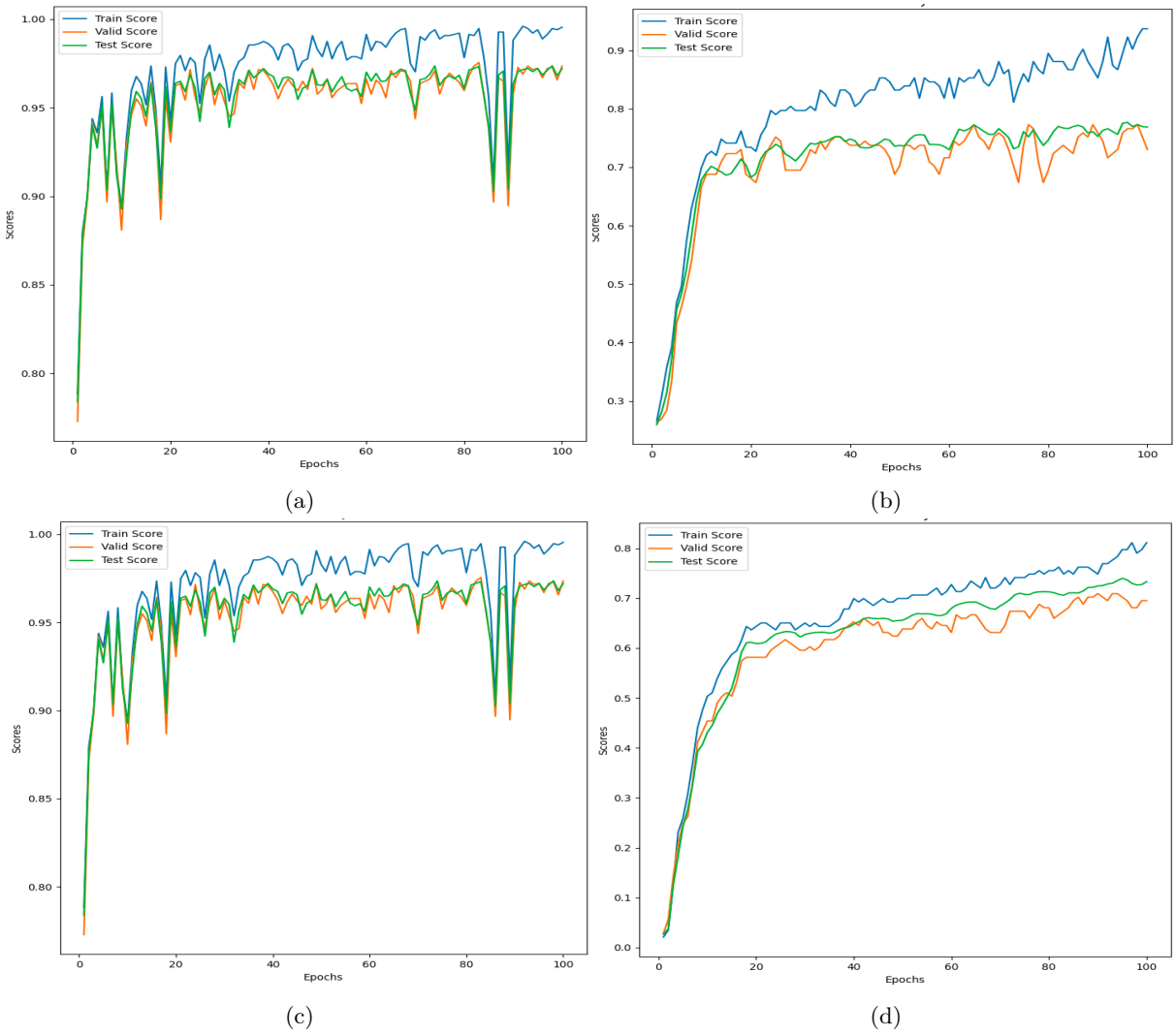



Figure 9: Accuracy score over epochs depending of the learning rate. Trento dataset : (a) lr = $10^{-3}$, (c) lr = $10^{-5}$; Houston 2013 dataset : (b) lr = $10^{-3}$, (d) lr = $10^{-5}$

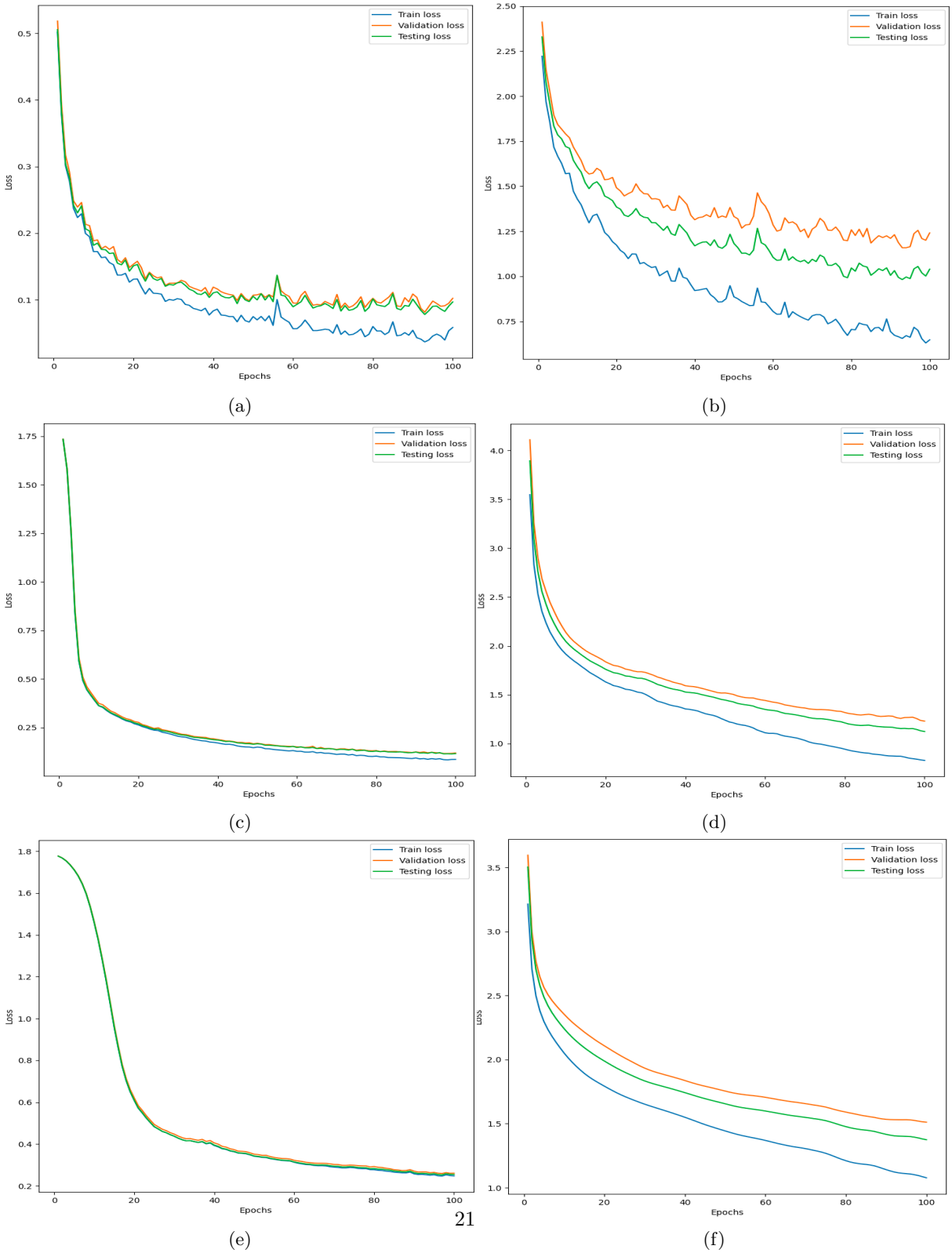





Figure 10: Loss function over epochs according to the batch size. Trento dataset : (a) batch-size = 10, (c) batch-size = 100, (e) batch-size = 500; Houston 2013 dataset : (b) batch-size = 10, (d) batch-size = 100, (f) batch-size = 500

**Conclusion :** In order to test the effect of each hyper-parameter on the accuracy score and the running time, I chose a basic setup and modified one hyper-parameter at a time. I trained the model with different diffusion operators and averaged the results in the following table:

| Hyper-parameter | Test accuracy | Running time (s) |
|---|---|---|
| Classic setup | 72.6 | 6.73 |
| Number of layers = 5 | 71.5 | 9.45 |
| Number of layers = 3 | 73.61 | 7.91 |
| Hidden channels = 500 | 73.78 | 7.9 |
| Hidden channels = 100 | 67.22 | 6.5 |
| Dropout = 0.5 | 70.86 | 7.15 |
| Dropout = 0 | 73.32 | 7.22 |
| Learning rate = $5x10^{-5}$ | 69.33 | 7.1 |
| Learning rate = 0.001 | 72.23 | 7.2 |
| Batch size = 500 | 68.44 | 6.33 |
| Batch size = 10 | 72.7 | 11.34 |

Figure 13: Mean scores and training running time when modifying only one hyper-parameter, reminder the classic setup is 2 layers, 256 hidden channels, a dropout of 0.2, learning rate of 0.0001 and batch size of 100.

We can see that adding more hidden channels and layers increases the computational cost, indeed it increases the number of learnable parameters of the model. According to my experiences, I chose the following hyper-parameters : the laplacian diffusion operator with 8 powers, a learning rate of $10^{-4}$, batch-size of 50, 500 hidden channels, a dropout of 0.2 and 3 layers. The best hyper-parameters can change a little according to the dataset. Especially, the number of epochs used in the above examples is exaggeratedly large, 50-60 epochs are usually enough. With these hyper-parameters, for Houston, the average test accuracy over 10 runs is 79.99, and for Trento 97.32, when using the standard scaler.

The method developed during this internship, transforming an image into a graph and exploiting the SIGN architecture seems promising. According to the remarks concerning the choice of the operators and the hyper-parameters, here are the best scores I obtained during training.

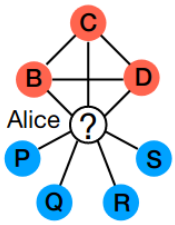
Nevertheless, this method still needs to be further developed, especially in the understanding of the heterogeneity of the results obtained on different datasets.

# 4 HOLS - Higher-Order Label Homogeneity and Spreading in Graphs

## 4.1 Theory

### 4.1.1 Empirical evidence

This architecture described in [6] uses strong connections to determine the label of a node. Traditional supervised learning leverage the homophily of vertices, stating that nearby vertices are likely to have the same label. HOLS is based on the fact that strong connection often participate in high-order structures.




| Network structure (clique) | $K_2$ | $K_3$ | $K_4$ |
|---|---|---|---|
| = | 3 | 3 | 1 |
| = | 4 | 0 | 0 |

Figure 14: Approach to correctly classify the unlabeled node "Alice" [6]

Label configurations indicates if vertices of a same clique share the same labels. For instance, a 3-clique has three label configurations : '1-1-1' where each vertex has a different label, '2-1' where only two vertices have the same label, and '3' where all vertices have the same label. A graph is said to have a high order label homogeneity if the label configurations 'k' for k-cliques happens more often than with a random label distribution. On the contrary, if the label configuration '1-1-...-1' happens more often in a labeled graph than if its labels are randomly chosen, vertices mix disassortatively.

### 4.1.2 Higher-order label spreading

Let $K$ be the set of network structures. For each motif $k \in K$, for instance "3-cliques", $|k|$ is its size, $Q_k$ the set of occurrences for this $k$, and for each $q \in Q_k, w_q$ is the sum of all incident edge weights.

The k-participation matrix is defined as $W^{(k)} = [w_{ij}^{(k)}]$ and $w_{ij}^{(k)} = \frac{\eta}{2} \sum_{q \in Q_k} w_q \mathbb{1}[i \in q \wedge j \in q$ the total weight of k-motifs that vertices i and j participate in. Then $w'_{ij} = \sum_{k \in K} \alpha_k w_{ij}^{(k)}$.

Let $y_i$ the one-hot vector of provided label for vertex i, which means that $y_{ic} = 1$ if vertex i has label c, and $x_i$ the inferred labels probability. We minimize the following loss :

$$L = (1 - \eta) L_s + \eta L_g$$

$$L_s = \frac{1}{2} \sum_i \|x_i - y_i\|^2$$

$$L_g = \frac{\eta}{2} \sum_{i,j} w'_{ij} \|x_i - x_j\|^2$$

Where $L_s$ is the supervised loss and $L_{g,k}$ the graph loss with respect to motif k, which enforces the inferred labels to be smooth and $\sum_{k\in K} \alpha_k = 1$.

We can define a modified adjacency matrix $W' = \sum_{k\in K} \alpha_k W^{(k)}$, and let $D'$ and $L'$ the degree and normalized Laplacian matrix of this new graph. With $X = [x_1...x_N]^T$ and $Y = [y_1...y_N]^T$, $N$ the number of vertices :

$$L = \frac{1-\eta}{2}\|x_i - x_j\|^2\|X - Y\|^2 + \frac{\eta}{2}X^T L' X$$

.

The exact solution to this minimization problem is $X = (1-\eta)(1-\eta(I-L'))^{-1}Y$ but matrix inversion is a task with high computational complexity. Instead, an iterative approach is used :

$$X \leftarrow \eta(I - L')X + (1-\eta)Y$$

The convergence to the unique point defined above is guaranteed.

## 4.2 Adaptation to pixel classification

The graph defined in the above section, using the spectral distance between pixels to compute the graph weights, gives a fully connected graph. In order to get relevant cliques and strong connections, we need to transform this graph by removing some edges. To solve this issue, I had two options, either keep only the connections between a node and its k nearest neighbors, or remove all the links whose weight is lower than a given threshold.

The first step to use this algorithm on our problem was to check if the graph from the remote sensing images had a high order label homogeneity or not. For this purpose, I planned to obtain the label configuration chart, and to use the Algorithm of Chiba and Nishizieki to list K-cliques.

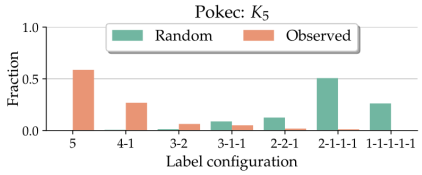



Figure 15: Label configuration chart. Comparison between random and observed label configuration to determine if the graphs presents high order label homogeneity, like it does in this example [6]

Unfortunately, I did not have the time to go through with the implementation of this method.

# Conclusion

Geometric machine learning is a promising field, hat allows us to work beyond the Euclidean domains and thus to consider data such as multimodal images in a new way.

During this internship, I discovered what geometric machine learning is and a set of graph convolution algorithms. I studied more particularly two methods the Scalable Inception Graph Neural Networks architecture and the Higher-Order Label Homogeneity and Spreading in Graphs. In order to adapt these methods to a pixel classification problem, we had to conceptualize effective transformations of multimodal images into graphs, different according to the method used.

The SIGN architecture presented many interesting results, which deserve to be investigated. However, a more precise understanding of the algorithm is needed to address its main limitation, which is the large performance gap depending on the datasets used. Indeed, the method performs very well on the Trento dataset, moderately well on Houston 2013 and totally ineffective on Houston 2018.

Due to an accident and medical complications, I did not have the opportunity to implement and test the HOLS method as I wished. However, I was able to learn how it works and think about an adaptation for pixel classification.

Finally, this internship allowed me to discover the world of research, I really enjoyed working in autonomy, and testing new ideas. Despite the frustrations caused by the problems I was not able to solve, and many trials and errors, I am happy with my work and what I learned during this experience.

# Acknowledgements

I would first like to express my gratitude to my supervisor, Mr. Andrea Marinoni, for his expertise and his unfailing support throughout this internship. Thanks to his trust, I was able to fully achieve my objectives.

I would also like to thank Mrs. Florence Tupin, who helped me a lot during my internship search. Her listening and her advises allowed me to target my candidatures, and to find this internship which was in total adequacy with my expectations.

Finally, I would like to thank all the CIRFA team for their welcome, the time spent together and sharing their knowledge with me, in particular Eduard Khachatrian, Saloua Chlaily and the other interns with whom I worked.

# Bibliographie

[1] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, Pierre Vandergheynst. 2017. Geometric deep learning : going beyond Euclidean data.

[2] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael M. Bronstein, Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks.

[3] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. In Conference on Neural Information Processing Systems (NeurIPS).

[4] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Benjamin Chamberlain, Michael Bronstein, Federico Monti. 2020. ICML 2020 Workshop on Graph Representation Learning and Beyond, url: https://github.com/twitter-research/sign

[5] Michael M. Bronstein. "Geometric foundations of Deep Learning". In: towards data science, 2021. url: https://towardsdatascience.com/graph-theory-and-deep-learning-know-hows-6556b0e9891b

[6] Dhivya Eswaran, Srijan Kumar, Christos Faloutsos. 2020. Higher-Order Label Homogeneity and Spreading in Graphs.

[7] Nitish Shirish Keskar, Jorge Nocedal, Dheevatsa Mudigere, Mikhail Smelyanskiy, Ping Tak Peter Tang. 2017. On large-batch training for deep learning : generalization gap and sharp minima.

[8] The Intergovernmental Panel on Climate Change (IPCC). 2013. AR5 Synthesis Report: Climate Change. url : https://www.ipcc.ch/report/ar5/syr/.

[9] Arctic Monitoring  Assessment Programme (AMAP). 2012. Arctic Climate Issues 2011: Changes in Arctic Snow, Water, Ice and Permafrost. url : https://www.amap.no/documents/doc/arctic-climate-issues-2011-changes-in-arctic-snow-water-ice-and-permafrost/129.

[10] Centre for Integrated Remote Sensing and Forecasting for Arctic Operations (CIRFA). url : https://cirfa.uit.no/research/.

[11] Saloua Chlaily, Mauro Dalla Mura, Jocelyn Chanussot, Christian Jutten, Paolo Gamba, Andrea Marinoni. 2020. Capacity and Limits of Multimodal Remote Sensing: Theoretical Aspects and Automatic Information Theory-Based Image Selection.

[12] Aidan Wilson. 2019. A Brief Introduction to Supervised Learning. url : https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590

[13] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks andlocally connected networks on graphs. In ICLR.

[14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neuralnetworks on graphs with fast localized spectral filtering. In NIPS.

[15] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neuralnetworks on graphs with fast localized spectral filtering. In NIPS.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learningon Large Graphs. In NIPS.

[17] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive Sampling Towards-Fast Graph Representation Learning. In NIPS.

[18] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2019. GraphSAINT: Graph Sampling Based Inductive Learning Method. arXiv : 1907.04931(2019).

[19] "z_ai". 2021. The Surprising Behaviour of Distance Metrics in High Dimensions. url : https:// towardsdatascience.com/the-surprising-behaviour-of-distance-metrics-in-high-dimensions-c2cb72779ea6.

[20] Flawnson Tong. 2019. Graph Convolutional Networks for Geometric Deep Learning. url : https://towardsdatascience.com/graph-convolutional-networks-for-geometric-deep-learning-1faf17dee008.

[21] Jason Brownlee. 2019. How to use Learning Curves to Diagnose Machine Learning Model Performance. Deep Learning Performance. url : https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

[22] Hafidz Zulkifli. 2018. Understanding Learning Rates and How It Improves Performance in Deep Learning. url : https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10.

[23] Baldi, P., Sadowski, P. J. (2013). Understanding dropout. In Advances in neural information processing systems (pp. 2814–2822).

[24] Akshay L Chandra. 2019. Learning Parameters, Part 5: AdaGrad, RMSProp, and Adam. url : https://towardsdatascience.com/learning-parameters-part-5-65a2f3583f7d

[25] Radhakrishna Achanta, Appu Shaji, Kevin Smith,Aurelien Lucchi, Pascal Fua, and Sabine S usstrunk. 2012. SLIC Superpixels Compared to State-of-the-artSuperpixel Methods. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[26] Haggai Maron, Heli Ben-Hamu, Nadav Shamir Yaron Lipman. 2019. Invariant and equivariant graph networks. In ICLR.
[34] Thomas N Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In ICLR.

[59] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019.Simplifying Graph Convolutional Networks. In ICML.