Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Matematica

GEOMETRIC DEEP LEARNING FOR POLYGONAL MESH DENOISING

Relatore:
Chiar.ma Prof.ssa
SERENA MORIGI
Correlatore:
Chiar.ma Prof.ssa
DAMIANA LAZZARO

Presentata da:
ANNA DE GIROLAMI

I Sessione
Anno Accademico 2021/2022

Introduzione

Negli ultimi anni le convolutional neural networks (CNNs) hanno ottenuto ottimi risultati in molti campi relativi all'elaborazione delle immagini. Questo fenomeno ha portato molti ricercatori a interessarsi maggiormente alla generalizzazione degli operatori convoluzionali ad altre tipologie di dati, come ad esempio i grafi e le mesh, che possiamo definire dati di tipo non Euclideo. Le molteplici tecniche emerse da questi studi appartengono a un nuovo campo del machine learning, noto come geometric deep learning. I dati di tipo non Euclideo sono però caratterizzati da una alta dimensionalità e una irregolarità propria della struttura, queste due proprietà rendono l'estensione del concetto di convoluzione non immediata. In particolare, le maggiori difficoltà sono state riscontrate a causa dell'impossibilità di definire un filtro invariante per traslazione.
Le reti neurali per l'elaborazione di grafi concepite come generalizzazione delle CNNs sulle immagini sono dette graph neural networks (GNNs). Il presente lavoro si propone come descrizione dei più noti operatori convoluzionali che caratterizzano i layers delle GNNs. Trattata la mesh come caso particolare di grafo, analizziamo i risultati dell'applicazione degli operatori convoluzionali introdotti. La trattazione si divide in due parti, una teorica e una sperimentale. La parte teorica si estende nei primi cinque capitoli. Nel primo capitolo sono esposti i concetti preliminari di two-manifold e di mesh, inoltre viene fornita la definizione chiave dell'operatore di Lablace-Beltrami nel caso discreto. Nel secondo capitolo, seguendo la teoria ralativa ai grafi, illustriamo l'analogo sui segnali discreti della trasformata di Fourier. Questo ultimo

concetto è introduttivo alla definizione di convoluzione spettrale su mesh. I capitoli successivi sono dedicati all'esposizione delle GNNs. Innanzitutto, nel terzo capitolo della tesi definiamo quali sono gli elementi caratterizzanti una rete neurale, in particolare forniamo un noto esempio di architettura, la Graph U-Net. Nel quarto e quinto capitolo notiamo che le GNNs possono appartenere a due differenti categorie: in base all'operatore di convoluzione che le caratterizza si dividono in spectral convolutional neural networks e spatial convolutional neural networks. Per entrambe le categorie individuiamo le reti più note. Appartengono alla prima categoria le Chebyshev neural networks e le graph convolutional networks (GCNs), appartengono alla seconda categoria le reti caratterizzate dal modello Graph SAGE e le reti graph attention networks (GATs). In coda al quinto capitolo, viene anche esposto il tentativo di definire una struttura matematica unificata da cui far discendere gli operatori convoluzionali. La parte sperimentale occupa l'ultimo capitolo. In particolare riportiamo i risultati ottenuti durante il tirocinio in preparazione alla tesi conseguito all'interno dell'ateneo. Applichiamo la teoria approfondita nei precendenti capitoli ad uno dei maggiori problemi di computer grafica: il denoising delle superfici. L'obiettivo è quello di utilizzare le GNNs per ricostruire superfici danneggiate a causa delle misurazioni imperfette eseguite dagli scanner 3D. La fase di sperimentazione si sviluppa in cinque sezioni. Una prima sezione è dedicata alla scelta dell'operatore di convoluzione più adatto. Nella seconda sezione indaghiamo invece il tipo di architettura più idoneo, in particolare valutiamo i vantaggi di una architettura di tipo encoding e decoding come, ad esempio, la Graph U-Net. Nella terza sezione tentiamo di arricchire la loss function imponendo la fedeltà alla curvatura durante il training. Nella penultima sezione proviamo le reti addestrate su un dataset meno rumoroso. Infine l'ultima sezione è dedicata alla visualizzazione delle features in output dei layer convoluzionali intermedi che costituiscono una rete. I risultati ottenuti dalla ricostruzione di superfici perturbate utilizzando i diversi operatori convoluzionali suggeriscono una nuova direzione di ricerca e applicazione.

Contents

Chapter 1

The Geometry of Manifolds

A large field of computer graphics concerns the processing of 3D shapes, that is, two-manifolds embedded in $\mathbb{R}^3$. Basically, a *d-manifold* is a topological space with the property that each point has a neighborhood that is homeomorphic to an open subset of d-dimensional Euclidean space. In order to act on these topological spaces, we bring the two-manifolds back to a suitable discrete representation, the polygonal meshes.
In this chapter we give a brief description of the geometry of discrete manifolds, in particular, we introduce the notion of polygonal mesh and we generalise the concept of Laplacian operator on a discrete domain. Since we assume the basic notation of geometry, like Riemannian manifold, we invite the reader to consult [1] and [4] for more details. With regards to the definition of Laplacian operator in the discrete domain we recommend reading [2].

1.1 Discrete Manifolds

We devote this section to the definition of discrete manifold, that is, polygonal mesh. In particular, we introduce two fundamental matrices, the adjacency matrix and the degree matrix.

A two-manifold $\mathcal{X}$ embedded in $\mathbb{R}^3$ can be approximated in the discrete domain by polygonal mesh representations.

Definition 1.1.1. A *polygonal mesh* (or *discrete manifold*) is a triplet of finite sets $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, where, given that $N_v = |\mathcal{V}|$ and $N_e = |\mathcal{E}|$:

- $\mathcal{V} = \{ v_i \}_{i=1}^{N_v}$, with $v = (x, y, z) \in \mathbb{R}^3$, $\forall v \in \mathcal{V}$, its elements are the *vertex coordinates*;
- $\mathcal{E} = \mathcal{V} \times \mathcal{V} = \{(i, j),\ \forall i, j = 1, \ldots, N_v\}$, its elements are the indices to the vertices that define the *edges*;
- $\mathcal{T} = \mathcal{V} \times \mathcal{V} \times \mathcal{V} = \{(i, j, k),\ \forall i, j, k = 1, \ldots, N_v\}$, its elements are the indices to the vertices that make up the *flat facets*.

In addition, the following conditions must apply to define consistent polygonal meshes:

1. each edge is shared by a maximum of two adjacent faces;
2. one edge connects two vertices, faces are sequences of closed edges;
3. a vertex is shared by at least two edges.

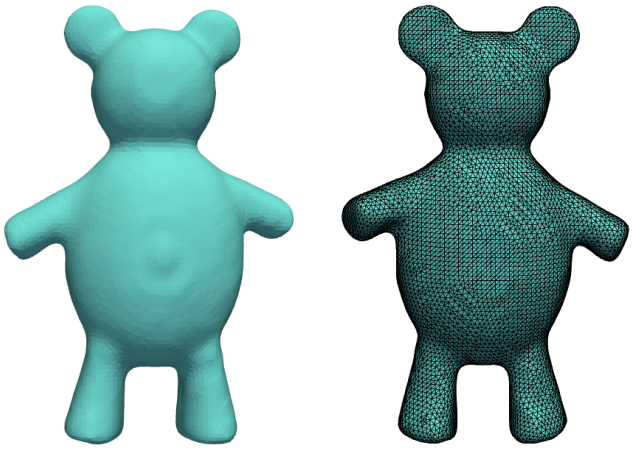
The most common two-manifold representation is given by the triangular mesh, i.e. a polygonal mesh with triangular faces.
From now on, we may refer to polygonal mesh simply as mesh.

We notice that in the definition of polygonal mesh, we also have a graph structure consisting of $(\mathcal{V}, \mathcal{E})$. We then define the adjacency matrix of a mesh as the adjacency matrix of its underlying graph.

Definition 1.1.2. The *adjacency matrix* W defined on a mesh $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ is an $N_v \times N_v$ matrix of edge weights where:

$$\begin{cases} w_{i,j} > 0 & \text{if } (i, j) \in \mathcal{E} \\ w_{i,j} = 0 & \text{otherwise} \end{cases}$$

for suitable $w_{i,j} \in \mathbb{R}$.



(a) *Topological manifold* (b) *Triangular mesh*

Figure 1.1: From two-manifold to discrete manifold

In this work any two vertices of the mesh will be assumed connected but without direction, i.e. $(i, j) \in \mathcal{E}$ if and only if $(j, i) \in \mathcal{E}$. In other words, the graph $(\mathcal{V}, \mathcal{E})$ underlying the mesh is supposed undirected. In this case we say the *mesh* is *undirected*. Moreover, let X be an undirected mesh, we assume $w_{i,j} = w_{j,i}$, $\forall i, j = 1, ..., N_v$, leading the adjacency matrix to be symmetric.

From the definition of adjacency matrix follows that of degree matrix. Let us define the *degree of a vertex* $v \in \mathcal{V}$ as the weighted sum of connecting edges. From this notion follows the definition of a very important matrix.

Definition 1.1.3. Given a polygonal mesh $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, with an adjacency matrix W, the *degree matrix* D is a $N_v \times N_v$ diagonal matrix whose elements are the degree of vertices:
$\forall i = 1, \ldots, N_v$

$$D_{i,i} = \sum_{j=1}^{N_v} w_{i,j} \tag{1.1}$$

In Fig. 6.9 we show an example of triangular mesh characterized by $X = (9548, 114552, 19092)$.

1.2 The Laplace-Beltrami Operator in Riemannian Geometry

In differential geometry, a d-manifold is provided with an additional structure, the Riemannian metric. The definition of a metric allows us to introduce the meaningful concepts of length, distance, angle, area and volume on manifolds. In this case, we refer to manifolds as Riemannian manifolds.
In this section we introduce the Laplace-Beltrami operator in classical Riemannian geometry, which is a generalization of the Euclidean Laplace operator. to functions defined on Riemannian manifolds. We invite the reader to consult [4] for more details.

Let us give the definition of a scalar field on a Riemannian manifold $\mathcal{X}$.

Definition 1.2.1. A *scalar field* defined on a Riemannian manifold $\mathcal{X}$ is a real function $f : \mathcal{X} \to \mathbb{R}$.

For any twice-differentiable real valued function f defined on Euclidean space $\mathbb{R}^n$, the Laplace operator (also known as the Laplacian) takes f to the divergence of its gradient vector field. Similarly, the Laplace-Beltrami operator is defined as the divergence of the gradient.

Definition 1.2.2. The *Laplace-Beltrami operator* acts on scalar field f and it is defined on a Riemannian manifold $\mathcal{X}$ as

$$\Delta_{\mathcal{X}} f = -\mathrm{div}\,(\nabla_{\mathcal{X}} f), \tag{1.2}$$

where the divergence and the intrinsic gradient on $\mathcal{X}$ are understood via the metric (see [5], pag. 24).

Remark 1.2.1. The Laplace-Beltrami is a *self-adjoint operator* (symmetric), i.e. the following formula holds:

$$\langle \nabla_{\mathcal{X}} f, \nabla_{\mathcal{X}} f \rangle_{\mathcal{L}^2(T\mathcal{X})} = \langle \Delta_{\mathcal{X}} f, f \rangle_{\mathcal{L}^2(\mathcal{X})} = \langle f, \Delta_{\mathcal{X}} f \rangle_{\mathcal{L}^2(\mathcal{X})} \tag{1.3}$$

1.3 The Mesh Laplacian Operator

We are now interested in the generalisation to a discrete domain of the Laplacian operator discussed above on a continuous domain. In particular, we will consider discrete manifolds, that is, polygonal meshes. We invite the reader to consult [2] for more details.

We need some preliminary definitions, such as the gradient and divergence operators, defined on the vertices and edges of the mesh, respectively.
Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a polygonal mesh, we define:

$$\mathcal{L}^2(\mathcal{V}) = \{f : \mathcal{V} \to \mathbb{R},\, v_i \mapsto f_i\}, \quad \mathcal{L}^2(\mathcal{E}) = \{F : \mathcal{E} \to \mathbb{R},\, (i,j) \mapsto F_{i,j}\}.$$

Notice that, once an order of $\mathcal{V}$ and of $\mathcal{E}$ are given we have $\mathcal{L}^2(\mathcal{V}) \cong \mathbb{R}^{N_v}$ and $\mathcal{L}^2(\mathcal{E}) \cong \mathbb{R}^{N_e}$. Let $f \in \mathcal{L}^2(\mathcal{V})$, $F \in \mathcal{L}^2(\mathcal{E})$ be two functions, we denote as f_i and $F_{i,j}$ the scalar value corresponding to the vertex $v_i \in \mathcal{V}$ and the edge $(i,j) \in \mathcal{E}$ realizing the above identifications $\mathcal{L}^2(\mathcal{V}) \cong \mathbb{R}^{N_v}$, $f = (f_i)$, $i = 1, \ldots, N_v$, $\mathcal{L}^2(\mathcal{E}) \cong \mathbb{R}^{N_e}$, $F = (F_{i,j})$, $(i,j) \in \mathcal{E}$.

With the notations fixed, the two definitions can be given.

Definition 1.3.1. Given a polygonal mesh $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, let $f \in \mathcal{L}^2(\mathcal{V})$ be a function on its vertices, we define the *mesh gradient operator* $\nabla_X : \mathcal{L}^2(\mathcal{V}) \to \mathcal{L}^2(\mathcal{E})$, at the edge $(i,j) \in \mathcal{E}$, as

$$\begin{cases} (\nabla_X f)_{i,j} := f_i - f_j & \text{if } (i,j) \in \mathcal{E}, \\ (\nabla_X f)_{i,j} := 0 & \text{otherwise.} \end{cases} \tag{1.4}$$

Definition 1.3.2. Given a polygonal mesh $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, with an adjacency matrix W, let $F \in \mathcal{L}^2(\mathcal{E})$ be a function on its edges. The *mesh divergence operator*, $\mathrm{div} : \mathcal{L}^2(\mathcal{E}) \to \mathcal{L}^2(\mathcal{V})$, at the vertex $v_i \in \mathcal{V}$ is

$$(\mathrm{div} F)_i := \frac{1}{a_i} \sum_{j:(i,j)\in\mathcal{E}} w_{i,j} F_{i,j}, \tag{1.5}$$

where $a_i > 0$ is a suitable weight on the vertex v_i.

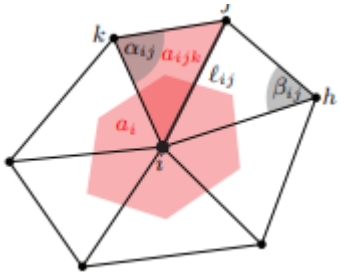


Figure 1.2: Triangular mesh

Finally, combining the two equations (1.4) and (1.5), we obtain the definition of mesh Laplacian operator.

Definition 1.3.3. Given a polygonal mesh $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, with an adjacency matrix W and the weights $a_i > 0$ associated to the vertices $v_i \in \mathcal{V}$, let $f \in \mathcal{L}^2(\mathcal{V})$ be a function on its vertices, the *mesh Laplacian operator* $\Delta_X : \mathcal{L}^2(\mathcal{V}) \to \mathcal{L}^2(\mathcal{V})$ at the vertex v_i is

$$(\Delta_X f)_i := (\operatorname{div}(\nabla_X f))_i := \frac{1}{a_i} \sum_{j:(i,j)\in\mathcal{E}} w_{i,j}(f_i - f_j), \tag{1.6}$$

Remark 1.3.1. Such a construction can be shown to converge to the continuous Laplacian of the underlying manifold, only when the vertex and edge weights satisfy some technical conditions.

Let us consider the particular case of a triangular mesh. First, the optimal weights assigned to the edges are the cotangent weights, that is, $\forall\,(i,j) \in \mathcal{E}$,

$$w_{i,j} := \frac{1}{2}(cot\alpha_{i,j} + cot\beta_{i,j}), \tag{1.7}$$

where α and β are two angles as in Figure (1.2).

Secondly, the vertex weight has to be interpreted as the local area element: given $v_i \in \mathcal{V}$, its weight is

$$a_i := \frac{1}{3} \sum_{j,k:(i,j,k)\in\mathcal{T}} a_{i,j,k} \tag{1.8}$$

where $a_{i,j,k} := \sqrt{s_{i,j,k}(s_{i,j,k} - \ell_{i,j})(s_{i,j,k} - \ell_{j,k})(s_{i,j,k} - \ell_{i,k})}$ is the area of triangle $(i,j,k) \in \mathcal{T}$ given by the Heron formula, $\ell_{i,j} := ||v_i - v_j||_{\mathbb{R}^3}$ is the distance

between the two vertices and $s_{i,j,k} := \frac{1}{2}(\ell_{i,j} + \ell_{j,k} + \ell_{k,i})$ is the semi-perimeter of triangle (i, j, k).

There is also a more general definition of the Laplacian operator that involves the introduction of a parameter $p \geq 1$ (see the article [12] for details).

Definition 1.3.4. Let the notations be as above, let $p \geq 1$, the *mesh p-Laplacian* operator, $\Delta_p : \mathcal{L}^2(\mathcal{V}) \to \mathcal{L}^2(\mathcal{V})$, at the vertex v_i is given by

$$(\Delta_p f)_i := (\text{div}\,(||\nabla_X f||^{p-2}\nabla_X f))_i. \tag{1.9}$$

where the norm $||\nabla_X f||^{p-2} \in \mathbb{R}^{N_e}$ is element wise.

Once again, from the definitions of gradient and divergence operators follows that:

$$(\Delta_p f)_i = \frac{1}{a_i} \sum_{j:(i,j)\in\mathcal{E}} w_{i,j}\, ||(\nabla_X f)_{i,j}||^{p-2}\,(f_i - f_j). \tag{1.10}$$

Notice that, if $p = 2$, it is equivalent to the classical definition of mesh Laplacian Beltrami operator defined in Def. 1.3.3.

Remark 1.3.2. Note that the Laplacian Δ_2 is a linear operator, while in general for $p \neq 2$, the p-Laplacian is nonlinear.

1.4 Laplacian Matrix Representation

Once defined the mesh Laplacian operator, in this section we move on to a most suitable matrix representation of it. For now, let us focus on the main case, the definition of Laplacian operator in which $p = 2$.

Definition 1.4.1. Given a polygonal mesh $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, the *mesh Laplacian matrix* is the $N_v \times N_v$ matrix

$$L = D - W, \tag{1.11}$$

where W and D are the adjacency matrix , Def. 1.1.2, and the degree matrix, Def. 1.1.3, respectively.

It is possible to consider sligthy different definitions of Laplacian matrix based on the normalization strategies. In particular, we are interested in the following:

- *unnormalized Laplacian*:

$$L_{un} = D - W; \tag{1.12}$$

- *normalized Laplacian*:

$$L_{norm} = D^{1/2} L D^{1/2} = I - D^{1/2} W D^{1/2}; \tag{1.13}$$

- *random walk Laplacian*:

$$L_{rw} = D^{-1} L = I - D^{-1} W. \tag{1.14}$$

The application of the described Laplacian operator to a function $f \in \mathcal{L}^2(\mathcal{V})$ on mesh vertices corresponds to a matrix L, such that:

$$\Delta f = Lf \tag{1.15}$$

in the identification $\mathcal{L}^2(\mathcal{V}) \cong \mathbb{R}^{N_v}$ which allows us to consider $\Delta \cong L : \mathbb{R}^{N_v} \to \mathbb{R}^{N_v}$.

Chapter 2

Mesh Spectral Convolution

Given the definition of Laplacian operator on meshes, described in the previous chapter, we are ready to introduce the analogue on meshes of the Euclidean Fourier transform, in order to provide the key definition of spectral mesh convolution. In this chapter we assume the basic notions of analysis, like Fourier transform, we invite the reader to consult [3] for more details. With regards to the definition of spectral mesh convolution, we recommend reading [2].

2.1 Laplacian Matrix Eigendecomposition

Prior the definition of the Fourier transform on meshes we must go over the eigendecomposition of the Laplacian matrix, the subject of this first section. We will follow [1] in our exposition.

We recall that the Laplacian operator is *self-adjoint positive-semidefinite*, which allows us to apply the spectral theorem. This implies that the Laplacian matrix, as defined in Sec. 1.4, admits an *eigendecomposition* on a compact domain (see [6]). Given a mesh $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, let $L \in \mathbb{R}^{N_v \times N_v}$ be its Laplacian matrix , we have

$$L = \Phi \Lambda \Phi^T, \tag{2.1}$$

- Λ is a diagonal matrix of real, non-negative *eigenvalues*

$$\Lambda = \text{diag}(\lambda_0, \ldots, \lambda_{N_v-1}), \tag{2.2}$$

$$0 = \lambda_0 \leq \ldots \leq \lambda_{N_v-1};$$

with $\lambda_i \in \mathbb{R}, \quad \forall i = 0, \ldots, N_v - 1$.

- Φ is a matrix of corresponding *orthonormal eigenfunctions*

$$\Phi = (\phi_0, \ldots, \phi_{N_v-1}), \tag{2.3}$$

such that $\forall i = 0, \ldots, N_v - 1, \ \phi_i \in \mathbb{R}^{N_v}$ and $L\phi_i = \lambda_i \phi_i$.
Furthermore, the eigenfunction corresponding to the eigenvalue $\lambda_0 = 0$ is $\phi_0 = \text{constant}$.

An example of Laplacian eigenfunctions on a non-Euclidean domain can be seen in Fig. 2.1.

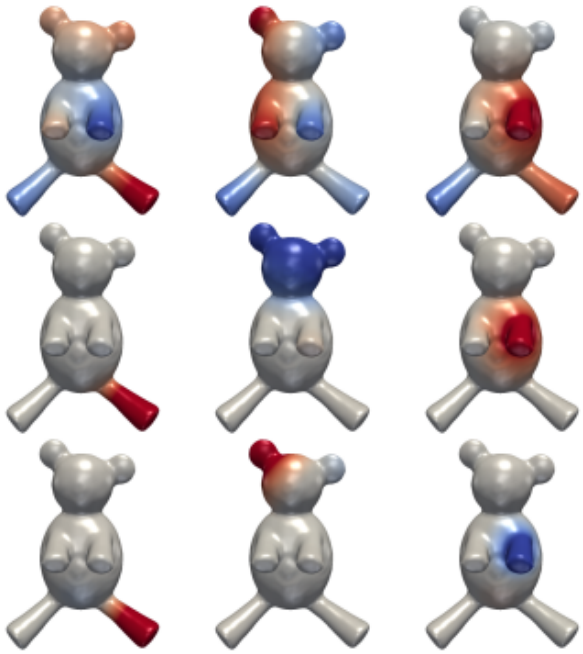


Figure 2.1: An example of Laplacian eigenfunctions on mesh

2.2 Dirichlet Energy

An interesting interpretation of the mesh Laplacian matrix eigenfunctions involves the generalization to the discrete domain of the concept of Dirichlet energy.

We first recall the classical definition.

Definition 2.2.1. Let $\mathcal{X}$ be a Riemannian manifold, given a differentiable function $f \in \mathcal{L}^2(\mathcal{X})$, its *Dirichlet energy* is

$$E(f) = \int_{\mathcal{X}} ||\nabla_{\mathcal{X}} f||^2 dx = -\int_{\mathcal{X}} f \Delta_{\mathcal{X}} f dx. \tag{2.4}$$

Note that the Dirichlet energy is a measure of how variable a function is. Then, given the classical optimization problem involving the Dirichlet energy:

$$\begin{cases} \arg\min\ E(\phi_0) \text{ s.t. } ||\phi_0|| = 1 \\ \arg\min\ E(\phi_i) \text{ s.t. } ||\phi_i|| = 1 \text{ and } \phi_i \perp \text{span}\{\phi_0, \dots, \phi_{i-1}\} \\ \phi_0, \phi_i \in \mathcal{L}^2(\mathcal{X}),\ \forall i = 1, \dots, K-1, \end{cases} \tag{2.5}$$

this will result in the smoothest possible base.

We turn to examine the generalisation in the discrete setting.
Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh, we look for an orthonormal basis on $\mathcal{L}^2(\mathcal{V})$ containing the $K \leq N_v$ functions $\{\phi_0^*, \dots, \phi_{K-1}^*\}$ that will solve the equivalent to the problem (2.5). Generalising the definition of Dirchlet energy in the discrete setting, (2.5) is equivalent to:

$$\Phi_K^* = \underset{\Phi_K \in R^{N_v \times K}}{arg\,min}\ \Phi_K^T \Delta_X \Phi_K,\ \text{s.t. } \Phi_K^T \Phi_K = \mathbf{I}, \tag{2.6}$$

Then $\Phi_K^* = (\phi_0^*, \dots, \phi_{K-1}^*)$.

It follows that the solution of the problem (2.6) $\Phi_K^* = (\phi_0^*, \dots, \phi{*}_{K-1}) \in \mathbb{R}^{N_v \times K}$ satisfy the condition:

$$\Delta_X \phi_k = \phi_k \lambda_k,\ \forall k = 0, \dots, K-1 \tag{2.7}$$

that is, the mesh Laplacian matrix eigenfunctions are the smoothest functions in the sense of the Dirichlet energy. In particular, they can be interpreted as a generalisation of the standard Fourier basis.

2.3 Mesh Fourier Transform

By exploiting the eigendecomposition of the mesh Laplacian matrix, we are able to define an operator that can be interpreted as the mesh analogue of the Euclidean Fourier transform.

Definition 2.3.1. Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh, given $f \in \mathcal{L}^2(\mathcal{V})$, we define the *mesh Fourier trasform* as

$$\hat{f} := \Phi^T f, \tag{2.8}$$

where Φ is the matrix of the mesh Laplacian eigenfunctions defined in (2.1). Similarly, we define its inverse operation:

$$f := \Phi \hat{f} \tag{2.9}$$

This means that the mesh Fourier transform projects the graph signal on the eigenvectors of the Laplacian matrix L. Thus, similarly to the classical Fourier transform, graph Fourier transform provides a way to represent a signal in two different domains, to which we refer as the *vertex (or spatial) domain*, where features are taken, and the *spectral domain*.

Remark 2.3.1. The definitions of the mesh Fourier transform and its inverse depend on the choice of Laplacian eigenvectors, which are not necessarily unique.

2.4 Mesh Convolutional Operator

The irregular structure of meshes makes the convolutions and filtering not as well-defined as on the Euclidean domain, however we can exploit the

convolution theorem to generalise this concept. The *convolution theorem* states that, in the Euclidean domain, the Fourier transform of convolution is the product of the Fourier transforms. It follows from the previous section the definition of mesh Fourier transform, then an analogous of the definition of convolution can be given.

Definition 2.4.1. Let $\mathcal{X} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh, given $f \in \mathcal{L}^2(\mathcal{V})$, the *mesh convolution* of f with a filter function h *in the spectral domain* is defined as follows:

$$g(f) = \Phi h(\Lambda)\Phi^T f, \tag{2.10}$$

where Φ is the matrix of the mesh Laplacian eigenfunctions and $h(\Lambda) \in \mathbb{R}^{N_v}$ is a diagonal matrix of spectral multiplers.

Remark 2.4.1. Note that $h(\Lambda)$ is a function of the eigenvalues of the Laplacian matrix: if h is the identity function, the convolution is equal to the application of the Laplacian operator.

Remark 2.4.2. The convolution and the Laplacian operators commutes, this means that the following formula holds

$$g(\Delta_X f) = \Delta_X g(f) \tag{2.11}$$

Remark 2.4.3. One of the key differences of such a construction from the classical convolution is the lack of shift-invariance. In terms of signal processing, it can be interpreted as a position-dependent filter.

While parametrized by a fixed number of coefficients in the frequency domain, the spatial representation of the filter can vary dramatically at different points.

Chapter 3

Graph Neural Networks

Graph neural networks (GNNs) are a class of neural networks for processing data best represented by graph data structures. In the following chapters, we exploit the interpretation of meshes as a special case of graphs in order to apply neural networks to the processing of meshes.
In this first chapter we give a brief introduction to graph neural network architecture and the layers application to meshes. In particular, we take a closer look to the graph U-Net architecture, that is an encoding-decoding neural network which provide the reduction and the increase of the number of vertices in the mesh data. We invite the reader to consult [15] for more details.

3.1 From Euclidean to non-Euclidean Domains

Graph neural networks (GNNs) are a class of neural networks for processing graph data, designed as a generalization of convolutional neural networks (CNNs) on images.

In the recent years, CNNs have achieved impressive results on many fields of image processing, as face detection, image segmentation, image classification and more.

These successes are due to an efficient architecture for extracting statistically significant patterns in large-scale, high-dimensional data sets. In fact, CNNs make it possible to extract local properties of the input data, i.e. local features shared throughout the data domain. These similar features are identified with localised convolutional filters, which are learnt from the data. These convolutional filters have compact support and are translation invariant, i.e. they are able to recognise identical features regardless of their spatial position.

Generalizing CNNs to graphs requires two fundamental steps:

- the design of localized convolutional filters on graphs,
- a graph pooling operation that exchange spatial resolution for higher filter resolution.

Remark 3.1.1. Each node of the graph may have a different number of neighbours. This fact prevents us from giving the definition of local and shift-invariant convolutional operators.

Due to the irregularity and the large size of the data, the generalisation of CNNs to graphs is not straightforward.

3.2 Basics of Deep Learning on Meshes

In this section we list the fundamental ingredients that make up a network.

A basic definition of machine learning is that of features. Generally, a mesh that is processed by a neural network have some features assigned to each node. The neural network acts directly on them, i.e. the node features are the values that the network takes as input.

Definition 3.2.1. Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh defined in Def. 1.1.1, we denote by $f \in \mathbb{R}^{N_v \times N_c}$ the *feature map*, where N_v is the number of the nodes in the graph and N_c is the number of features associated to each node.

Graph neural networks process mesh data acting on the features assigned to their nodes taking into account the structure of the underlying graph. The architecture of the network consists of an input layer, hidden layers and an output layer.
Below is a list of the elements that make up a convolutional network with a brief description of them.

- **Convolutional layer**. Since GNNs are a generalisation of CCNs, the main layers provide a convolutional operation, i.e. the features associated with each node are perturbed due to the influence of neighbouring values: for a given node, the features are multiplied by the weights of neighbours nodes and summed.

- **Activation function**. Each convolutional layer is followed by an activation function, in general it is taken to be a nonlinear function to do nonlinear regression and solve classification problems that are not linearly separable. Few example of activation functions: linear, sigmoid, tanh, Rectified Linear Unit (ReLU).

- **Pooling and unpooling layers**. The pooling operation reduce the size of the graph, the unpooling operation reverts the effect of the pooling operation. There are different types of pooling operators and they differ according to the sub-sampling criterion. In this work we will use the MAX-pooling and unpooling.

Furthermore, it is very common to divide a graph neural network including pooling and unpooling operators into two parts: encoding and decoding. An example of such neural network architecture is the graph U-NET described in the following section.

3.3 The Graph U-Net

The U-Net is an encoder-decoder architecture originally defined for image data based on convolutional, pooling and unpooling layers. For more details, we invite the reader to consult [16] . Since it has achieved impressive results on image pixelwise prediction tasks, it's interesting to develop a U-Net like architecture for mesh data and, in general, for graph data.
In the generalisation of the U-Net architecture to mesh data, the convolutional layers are generally assumed to be GCN, for our task we prefer to consider a more general definition and suppose the convolutional layers to be arbitrary. The next chapters are devoted to the definition of different mesh convolutional layers.
With regard to pooling and unpooling operators on meshes, since mesh nodes do not have the spatial locality required by normal pooling operations, extending these operations is very challenging. To address these challenges, we propose novel *graph pooling* (gPool) and *unpooling* (gUnpool) operations. The gPool layer adaptively selects some nodes to construct a smaller mesh based on their scalar projection values on a trainable projection vector. The gUnpool layer restores the mesh into its original structure using the position information of nodes selected in the corresponding gPool layer, we further propose the gUnpool layer as the inverse operation of the gPool layer. Based on our proposed gPool and gUnpool layers, we develop the encoder-decoder model on mesh known as the *graph U-Net*.

3.3.1 Graph U-Net Architecture

Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh with N_v vertices, each of which with N_c initial features assigned. Let $f \in \mathbb{R}^{N_v \times N_c}$ be its feature map.

A Graph-U-Net with depth P is defined as

$$\text{G U-Net} := \underbrace{T_{2P} \circ \cdots \circ T_{P+1}}_{decoder} \circ \underbrace{T_P \circ \cdots \circ T_1}_{encoder}$$

where $(T_l)_{1\leq l\leq 2P}$ are the layer blocks that make up the network.
We build the encoder by stacking P encoding blocks, $(T_l)_{1\leq l\leq P}$, each of which contains a gPool layer followed by a convolutional layer. In the decoder part, $(T_l)_{P+1\leq l\leq 2P}$, we stack the same number of decoding blocks as in the encoder part and each decoder block is similarly composed of a gUnpool layer and a convolutional layer. Convolutional layers are always followed by activation functions. Notice that there are skip-connections between corresponding blocks of encoder and decoder layers, which transmit spatial information to decoders for better performance.

Formally, the other layer blocks are characterized by the composition of:

- G, a graph convolutional layer, for example a GCN convolutional operator;
- σ, an activation function, for example a ReLU;
- p, a gPool/gUnpool operator.

Namely, let $\forall l = 1, \ldots, 2P$, $f^l \in \mathbb{R}^{N_v^l \times N_c^l}$ be the feature map that the l-th block takes as input, then the encoding/decoding block is a composition of functions

$$T_l : f^l \mapsto f^{l+1} = \sigma(G(p(f^l); \mathcal{W})),$$

where $\sigma(\cdot)$ is a nonlinear activation operator applied point-wise and $\mathcal{W}_l \in \mathbb{R}^{N_c^l \times N_c^{l+1}}$ is the trainable weight matrix for the convolutional operator.
The encoding-decoding network reads as follows:

$$\text{G U-Net} := \underbrace{T_{2P}^{gU} \circ \cdots \circ T_{P+1}^{gU}}_{decoder} \circ \underbrace{T_P^{gP} \circ \cdots \circ T_1^{gP}}_{encoder}. \tag{3.1}$$

In the decoding part, for $l = P+1, \ldots, 2P$, let g^{l+1} the output of the unpooling layer that the convolution layer takes as input, the operator is:

$$p : f^l \mapsto g^{l+1} := gUnpool(f^l),$$

and for $l = 1, \ldots, P$, the encoding part is defoined as

$$p : f^l \mapsto g^{l+1} := gPool(f^l).$$

. Fig. 3.1 is a visual representation of the stacked layers that make up the graph U-Net architecture. Note that the convolutional layers were assumed GCN.

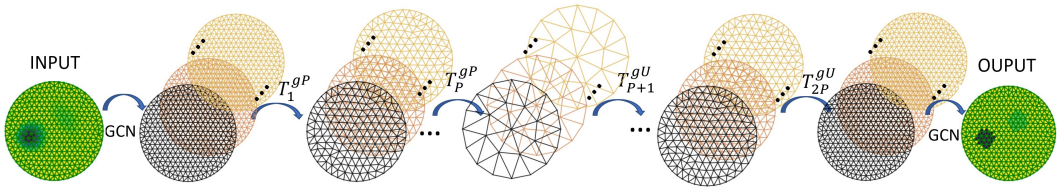


Figure 3.1: Graph U-Net architecture

3.3.2 Graph Pooling Layer

In order to reduce the size of the feature map and enlarge receptive fields, in this section, we propose the graph pooling (gPool) layer to enable down-sampling on mesh data. In this layer, we adaptively select a subset of veritices to form a new but smaller mesh. To this end, we employ a trainable projection vector $q \in \mathbb{R}^{N_c}$. Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh with N_v vertices, each of which with N_c initial features assigned. Let $f \in \mathbb{R}^{N_v \times N_c}$ be its feature map. Given a vertex $v \in \mathcal{V}$, we denote its feature vector $f(v) \in \mathbb{R}^{N_c}$. The scalar projection of $f(v)$ on q is

$$\phi(v) = \langle f(v), \frac{q}{||q||} \rangle \in \mathbb{R}. \tag{3.2}$$

with the Eucledean norm, and the Euclidean scalar product. Here, $\phi(v)$ measures how much informations of vertex v can be retained when projected onto the direction of q. By sampling vertices, we wish to preserve as much informations as possible from the original mesh. To achieve this, we select vertices with the largest scalar projection values on q to form a new mesh.

Let us consider the l-th pooling layer, with $l = 1, \ldots, P$. The input values are the feature associated to the verteces of a mesh X^l with feature map $f^l \in \mathbb{R}^{N_v^l \times N_c^l}$ and adjacency matrix $W^l \in R^{N_v^l \times N_v^l}$. The pooling layer returns a smaller mesh X^{l+1} characterized by a new feature map $g^{l+1} \in \mathbb{R}^{N_v^{l+1} \times N_c^l}$ and a new adjacency matrix $W^{l+1} \in R^{N_v^{l+1} \times N_v^{l+1}}$.
The layer-wise propagation rule of the mesh pooling layer l is defined as:

$$\phi = \langle f^l, \frac{q^l}{||q^l||} \rangle, \tag{3.3}$$

$$idx = rank(\phi, k), \tag{3.4}$$

$$\tilde{\phi} = sigmoid(S\phi), \tag{3.5}$$

$$\tilde{f}^l = Sf^l. \tag{3.6}$$

Then, the adjacency matrix and the feature map of the new down-sampled mesh can be updated.

$$W^{l+1} = SW^l S^T, \tag{3.7}$$

$$g^{l+1} = \tilde{f}^l \odot (\tilde{\phi} 1_{N_c^l}^T). \tag{3.8}$$

Where, $q^l \in \mathbb{R}^{N_c^l}$ is the trainable projection vector, $k = N_v^{l+1}$ is the number of vertices selected in the new mesh, $rank(\phi, k)$ is the operation of vertex ranking, which returns indices of the k-largest values in ϕ, the idx returned by $rank(\phi, k)$ contains the indices of nodes selected for the new mesh. $S \in \mathbb{R}^{k \times N_v^l}$ is the sampling matrix obtained by eliminating from the identity matrix $I_{N_v^l}$ the rows corresponding to the vertices selected by idx. $1_{N_c^l} \in \mathbb{R}^{\mathbb{N}_c^{\mathbb{l}}}$ is a vector of size N_c^l with all components being 1, and $\odot$ represents the Hadamard product (that is the element-wise matrix multiplication).

Using element-wise matrix product of $\tilde{f}^l$ and $\tilde{\phi} 1_{N_c^l}^T$, information of selected vertex is controlled. The i^{th} row vector in g^{l+1} is the product of the i^{th} row vector in $\tilde{f}^l$ and the i^{th} scalar value in $\tilde{\phi}$. Notably, the gate operation makes the projection vector p trainable by back-propagation. Fig. 3.2 provides an illustration of our proposed graph pooling layer.

Compared to pooling operations used in grid like data, our graph pooling layer employs extra training parameters in projection vector ϕ.

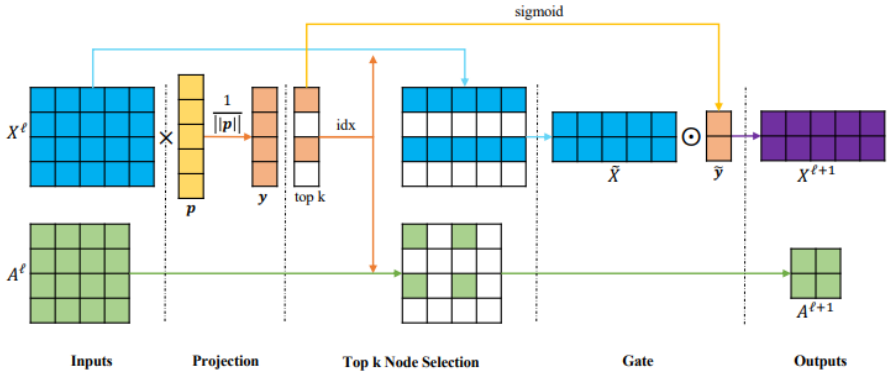


Figure 3.2: Graph Pooling Layer

3.3.3 Graph UnPooling layer

Up-sampling operations are important for encoder-decoder networks such as U-Net. The encoders of networks usually employ pooling operations to reduce feature map size and increase receptive field. While in decoders, feature maps need to be up-sampled to restore their original resolutions.

To enable up-sampling operations on mesh data, we propose the graph unpooling (gUnpool) layer, which performs the inverse operation of the gPool layer and restores the mesh into its original structure. To achieve this, we record the locations of vertices selected in the corresponding gPool layer and use this information to place vertices back to their original positions in the mesh. Let us consider the l-th unpooling layer, with $l = P + n$, and $n = 1 \ldots, 2P$. The corresponding pooling layer is the one of indice $l' = P - (n-1)$, let X^l be the initial mesh with feature map $f^l \in \mathbb{R}^{N_v^l \times N_c^l}$ and adjacency matrix $W^l \in R^{N_v^l \times N_v^l}$. Let us denote X^{l+1} the up-sampled mesh characterized by a new feature map $g^{l+1} \in \mathbb{R}^{N_v^{l+1} \times N_c^l}$ and a new adjacency matrix $W^{l+1} \in R^{N_v^{l+1} \times N_v^{l+1}}$.

Formally, we propose the layer-wise propagation rule of graph unpooling layer

as

$$g^{l+1} = S^T f^l, \tag{3.9}$$

where $S \in \mathbb{R}^{k \times N_v^l}$ is the same sampling matrix as the corresponding gPool layer, defined according to the vector $idx \in \mathbb{Z}^k$, which contains indices of selected vertices in the corresponding gPool layer that reduces the mesh size from N_v^l vertices to $k = N_v^{l+1}$ vertices. Finally, the updated Adjacent matrix is:

$$W^{l+1} = S^T W^l S. \tag{3.10}$$

Chapter 4

Frequency Domain Convolution Operators

In the following chapters we analyze some of the convolutional operators characterizing graph neural networks. Generally speaking, the graph convolutional neural networks can be divided into two categories. The first one refers to the definition of convolution based on the spectral graph theory, the other category exploits a second definition of convolution, named spatial domain convolution, which directly carries out the operation. We invite the reader to consult [6] for more details.
These operators can easily be adapted to convolutional operators on meshes. In this chapter, given a brief description of the convolution operator that characterises the layers of a convolution neural networks (CNNs) acting on a regular grid, we will take a closer look at some of the main convolution operators that belongs to the first category of GNN. In particular we will introduce the convolution operators of the Spectral GNN (SGNN), the Chebyshev neural network (ChebNet), the Graph Convolutional Network (GCN) and the p-Laplacian based graph neural networks (^{p}GNNs).We invite the reader to consult [7, 8, 12] for more details. Note that in the definition of the convolutional layer we consider the action of the activation function.

4.1 The Euclidean Convolutional Operator

In recent years, many researchers have designed the deep neural network architectures in graph data by referring to the CNNs on images. In fact, such networks have achieved great results on image processing. However, due to the irregularity (each vertex may have different number of neighbours) and the large scale of mesh data, a generalization of CNNs to graph is not straightforward. Let us define the CNN convolutional operator.

Definition 4.1.1. Let $\mathcal{I}$ be an image.
For each pixel $x \in \mathcal{I}$, we denote $f(x) = (f_1(x) \dots f_p(x))$ the p-dimensional input feature map and $g(x) = (g_1(x) \dots g_q(x))$ the q-dimensional output feature map. The *CNN operator* on the euclidean domain is defined:
$\forall l = 1 \dots q$,

$$g_l(x) = \sigma(\sum_{\bar{l}=1}^{p} (f_l * \omega_{l,\bar{l}})(x)) \tag{4.1}$$

where σ is the nonlinear activation function and $\mathcal{W} = (\omega_{\bar{l},l})$, $\bar{l} = 1 \dots p$, $l = 1 \dots q$, is a bank of learnable filters s.t. its components have compact spatial support.

The definition of feature map on images is analogous analogous to that on meshes, recall Def. 3.2.1

4.2 The Spectral Convolutional Operator

Similarly to the CNN convolutional operator (4.1) of a classical Euclidean CNN, we define the spectral convolutional operator of a Spectral GNN on meshes.

Definition 4.2.1. Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a *mesh*. Given $f = (f_1, \dots, f_p) \in \mathbb{R}^{N_v \times p}$ the input feature map associated to the vertices of a mesh and $g = (g_1, \dots, g_q) \in \mathbb{R}^{N_v \times q}$ the output feature map, the *spectral convolutional operator* is defined:

$\forall l = 1 \dots q$,

$$g_l = \sigma(\sum_{\bar{l}=1}^{p} \Phi_k \mathcal{W}_{l,\bar{l}} \Phi_k^T f_{\bar{l}}) \tag{4.2}$$

where σ is the nonlinear activation function and given $l = 1 \dots q$, $\mathcal{W}_{\bar{l},l} = \text{diag}(\omega_{\bar{l},l}) \in \mathbb{R}^{k \times k}$ is a matrix of spectral multipliers that represents a learnable filter in the frequency domain.

Remark 4.2.1. Using only the first k Laplacian eigenvectors (Φ_K) sets a cutoff frequency (tipically $K << N_v$), describing the smooth structure of the mesh. See the mesh Laplacian matrix eigendecomposition at Sec. 2.1.

Moreover, the Laplacian matrix eigenfunctions are domain-dependent. The filter coefficients learnt on one mesh domain cannot be applied to another one in a straightforward manner. If we learn a filter with respect to basis Φ_K on the mesh domain, and then lay to apply it on another mesh characterized by another basis Ψ_K, the result could be very different.

Figure 4.1: Same filter on different domains

An example illustrating the difficulty of generalizing spectral filtering across non-Euclidean domains is reported in the Fig. 4.1. On the left we can observe a function defined on a manifold (function values are represented by color), in the middle we have the result of the application of an edge-detection filter in the frequency domain. On the right we can observe the same filter, applied on the same function, but on a different (nearly-isometric) domain, it produces a completely different result.

Remark 4.2.2. The pooling operation in traditional CNN framework is replaced by graph coarsening procedure (input a graph with N_v vertices, and then produces a graph with $\bar{N_v} < N_v$ vertices and transfers featres from the vertices of the fine graph to those of the coarse one): the commonly used coarsening algorithm is Graclus.

Set a factor $\alpha < 1$, the matrix of Laplacian eigenvectors, from an original $\Phi \in \mathbb{R}^{N_v \times N_v}$, becomes a matrix $\tilde{\Phi} \in \mathbb{R}^{\alpha N_v \times \alpha N_v}$.
We invite the reader to consult [17] for more details.

4.3 The Chebyshev Convolutional Operator (Cheb-Conv)

The Chebyshev convolutional operator belongs to those spectral convolutional operators exploiting parametric filters. This kind of filters represents an improvement over Spectral GNN as they are responsible for overcoming some of the major drawbacks as we will deepen later.
Such convolutional operators correspond to define the spectral filter $\mathcal{W}_{\bar{l},l} \in \mathbb{R}^{k \times k}$ as a combination, such that,
$\forall\, \bar{l} = 1 \dots p,\ l = 1 \dots q,$

$$\mathcal{W}_{\bar{l},l} = h_{\omega_{\bar{l},l}}(\Lambda) = \sum_{j=0}^{r-1} (\omega_{\bar{l},l})_j \mathrm{pol}_j(\Lambda), \tag{4.3}$$

where $\Lambda \in \mathbb{R}^{k \times k}$ is the diagonal matrix of the first $k \leq N_v$ mesh Laplacian matrix eigenvalues.
In particular, the Chebyshev convolutional operator is based on the Chebyshev polynomial basis.

Definition 4.3.1. The *Chebyshev polynomials* $T_i : \mathbb{R} \to \mathbb{R}, \forall i \leq 0$, are defined in a recursive form as:

$$T_0(\lambda) = 1, \tag{4.4}$$

$$T_1(\lambda) = \lambda,$$

$$T_k(\lambda) = 2\lambda T_{k-1}(\lambda) - T_{k-2}(\lambda).$$

We can transponse the definition from $\mathbb{R}$ to the square matrices domain $\mathbb{R}^{N\times N}$, assuming that the first polynomial would be $T_0(\lambda) = \mathbb{I}_N$.

Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh, Λ be the $N_v \times N_v$ diagonal matrix of Laplacian eigenvalues, $T_j(\bar{\Lambda})$ is the Chebyshev polynomial of order j evaluated at $\bar{\Lambda} = 2\frac{1}{\lambda_{N_v}}\Lambda - \mathbb{I}$, aiming to reduce the size of the eigenvalues from $[0, \lambda_{N_v}]$ to [-1, 1], since the Chebyshev polynomial form an orthonormal basis that lies in [-1, 1]. Combining the recursive form of Chebyshev polynomial applied to the matrix of Laplacian eigenvalues, the filter in (4.3) describing Chebyshev convolution can be parametrized as the truncated expansion of order $r - 1$:

$$h_{\omega\bar{l},l}(\Lambda) = \sum_{j=0}^{r-1} (\omega_{\bar{l},l})_j \, T_j(\bar{\Lambda}) \tag{4.5}$$

where $\omega_{\bar{l},l} \in \mathbb{R}^r$ is the vector of Chebyshev coefficients for the input and output features with indices $\bar{l}, l$. Let $\Phi \in \mathbb{R}^{N_v\times N_v}$ be the matrix of Laplacian eigenvectors, due to the eigen-decomposition of the Laplacian matrix, the ChebNet's filtering operation can be written as:

$$\Phi h_{\omega\bar{l},l}(\lambda)\Phi^T = \sum_{j=0}^{r-1} (\omega_{\bar{l},l})_j T_j(\bar{L}) \tag{4.6}$$

where $\bar{L} = 2\frac{1}{\lambda_{Nv}}L - \mathbb{I}$ is the scaled Laplacian.

Once the filter has been described, the definition of the Chebyshev convolutional operator is given.

Definition 4.3.2. Given a mesh $\mathcal{X} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, let $F = (f_1, \ldots, f_p) \in \mathbb{R}^{N_v\times p}$ be the input feature map associated to the vertices of a mesh, denoted $G = (g_1, \ldots, g_q) \in \mathbb{R}^{N_v\times q}$ the output feature map, the *Chebyshev filtering operation* is defined as following:

$\forall l = 1 \dots q$,

$$g_l = \sigma(\sum_{\bar{l}=1}^{p} (\sum_{j=0}^{r-1} (\omega_{\bar{l},l})_j T_j(\bar{L}))\, f_{\bar{l}}) \tag{4.7}$$

where σ is the non-linear activation function, $\forall\, l = 1, \dots, q$,
$\omega_{\bar{l},l} \in \mathbb{R}^r$ is the vector of learnable parameters and $T_j(\bar{L})$ is the Chebyshev polynomial of order j applied on the scaled Laplacian matrix $\bar{L}$.

Remark 4.3.1. $T_j(\bar{L})$ is the $\bar{L}$'s polynomial of order j, which corresponds to the j-ring neighborhood.

Indeed, given two vertices denoted by the two indices $m, n \in \{1, \dots, N_v\}$, $d_G(m, n) > j$ implies $(L^j)_{m,n} = 0$, where d_G is the shortest path distance, i.e. the minimum number of edges connecting two vertices on the mesh. Consequently, spectral filters represented by $(r-1)^{th}$-order polynomials of the mesh Laplacian are exactly $(r-1)$-localized.

Finally, there are two remarkable improvement of Cheb-Conv operator over the spectral convolutional operator:

- unlike the spectral convolutional operator, that is very costly, the Chebyshev offers an efficient filtering scheme that does not require an explicit computation of the Laplacian eigenvectors.

- $T_k(\bar{L}) f_{\bar{l}}$ can operate locally on each vertex, which means that the filters of ChebNet is localized in spatial domain.

Despite the advantages there are still some drawbacks to overcome:

- the weights assigned to different neighbors in the same order neighborhood are exactly the same

- the learned weights of the model can't be adapted to different domains.

4.4 The Graph Convolutional Operator (GCN-Conv)

The graph convolutional operator can be defined as the first-order approximation of the Chebyshev convolutional operator, i.e. GCN simplifies ChebNet architecture by using filters operating on 1-ring neighborhood of the mesh. The definition of the GCN-Conv is given by the two following assumptions:

- $r = 2$,
- $\lambda_{N_v} = 2$.

Note that the second condition follows automatically from the choice of the normalized Laplacian $L^{norm} = \mathbf{I} - D^{1/2} W D^{1/2}$ (see Sec 1.4).
Then, let us consider the Cheb-Conv operator (4.7), the output of the GCN-Conv is:

$$\begin{aligned} g_l &= \sigma\Big(\sum_{\bar{l}=1}^{p} (\sum_{j=0}^{1} (\omega_{\bar{l},l})_j T_j(\bar{L})) \, f_{\bar{l}} \Big) \\ &= \sigma\Big(\sum_{\bar{l}=1}^{p} (\omega_{\bar{l},l})_0 \, f_{\bar{l}} + (\omega_{\bar{l},l})_1 \, \bar{L} \, f_{\bar{l}} \Big) \\ &= \sigma\Big(\sum_{\bar{l}=1}^{p} (\omega_{\bar{l},l})_0 \, f_{\bar{l}} + (\omega_{\bar{l},l})_1 \, (L^{norm} - \mathbf{I}) \, f_{\bar{l}} \Big) \\ &= \sigma\Big(\sum_{\bar{l}=1}^{p} (\omega_{\bar{l},l})_0 \, f_{\bar{l}} - (\omega_{\bar{l},l})_1 \, D^{-1/2} W D^{-1/2} \, f_{\bar{l}} \Big), \quad \forall l = 1 \dots q. \end{aligned} \tag{4.8}$$

In order to prevent the occurrence of overfitting due to the excessive number of parameters, a unified hypothesis is used $\omega_{\bar{l},l} = (\omega_{\bar{l},l})_0 = -(\omega_{\bar{l},l})_1$. Then the definition of GCN convolution operator becomes:

$$g_l = \sigma\Big(\sum_{\bar{l}=1}^{p} \omega_{\bar{l},l} \, (\mathbf{I} + \, D^{-1/2} W D^{-1/2}) \, f_{\bar{l}} \Big), \quad \forall l = 1 \dots q. \tag{4.9}$$

Then the set of learnable parameter of the network is $\Theta = \{\omega_{\bar{l},l}\}_{l=1,\dots,q}^{\bar{l}=1,\dots,p}$ with dimension $p \times q$.

Remark 4.4.1. Successive application of filters of this form then effectively convolve the K^{th}-order neighborhood of a node, where K is the number of successive filtering operations or convolutional layers in the neural network model.

4.5 The p-Laplacian Convolutional Operator (^{p}GNN-Conv)

Note that GCN-Conv consists on the application of the mesh Laplacian operator to the features in order to propagate the signal on the mesh. In this section, we define a convolutional operator using the mesh p-Laplacian defined in Def. 1.3.4. The strength of this method involves the fact that different choices of p result in different smoothness constraint on the vertex features. We invite the reader to consult [18] for more details.

The definition of this convolutional operator follows from the solution of a variational problem. Note that it doesn't involve learnable parameters. Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh and W be its adjacency matrix , $f \in \mathbb{R}^{N_v \times s}$ denotes the input feature map. Furthermore, only in this section we denote, $\forall i \in \{1, \dots N_v, \}$, $f_i \in \mathbb{R}^s$, the i-th row of the feature map, that is, the vector of features associated to the vertex $v_i \in \mathcal{V}$. Then, we denote $\forall j \in \{1, \dots, s\}$, $f^j \in \mathbb{R}^{N_v}$ the j-th column of the feature map, that is, a scalar function on the vertices. Note that in this section we don't denote the number of input features with each vertex as p anymore: the number of input features is s. Set $p \geq 1$, the aim is to update node features according to the following variational problem, the output of the convolutional layer is $g^* \in \mathbb{R}^{N_v \times s}$ solution of:

$$g^* = \underset{g \in R^{N_v \times s}}{arg\,min} \frac{1}{p} ||\nabla g||_p^p + \frac{\mu}{2} ||\bar{f} - g||_F^2. \tag{4.10}$$

Note that the number of features with each vertex doesn't change. The first term of the right-hand side is a measurement of variation of the feature map

over the mesh based on p-Laplacian, see the mesh p-laplacian Def. 1.3.4. The second term is the constraint that the optimal feature map g^* should not change too much from a given data $\bar{f}$. Finally, $\mu \in]0, \inf[$ provides a trade-off between these two constraints. Thus, the solution of the variational problem is a feature map on the mesh such that meets a smoothness criterion without departing from the initial map.

Recall the definition of $\nabla g \in \mathbb{R}^{N_e \times s}$ in the Sec. 1.3. We will denote $(\nabla g)_i := \{(\nabla g)_{i,j}, \forall j : (i,j) \in \mathcal{E}\} \in R^{\bar{N}}$, with $\bar{N} = |N_{v(i)}|$, and $N_{v(i)}$ the vertex neighborhood, the vector of partial derivatives at the vertex $v_i \in \mathcal{V}$, while we will denote $(\nabla g)_{i,j}$ the scalar value of the partial derivative of function g with respect to the direction defined by the edge (i,j).
Then, we define the p-norm as:

$$||\nabla g||_p^p := \frac{1}{p}\sum_{i=1}^{N_v} ||(\nabla g)_i||_2^p \tag{4.11}$$

$$= \frac{1}{p}\sum_{i=1}^{N_v} \Big(\sum_{j:(i,j)\in\mathcal{E}} (g_i - g_j)^2 \Big)^{\frac{p}{2}} \tag{4.12}$$

Remark 4.5.1. When $p = 1$, the formulation (4.10) is named total variation regularization.

Let us consider the easiest case $p = 2$, we require the first derivative to be equal to 0, then the solution of (4.10), g^*, satisfy

$$\Delta g + \mu\,(g - \bar{f}) = 0, \tag{4.13}$$

that is the discrete analogue of the Euler-Lagrange equation. Then, g^* is solution of a linear system,

$$(L + \mathbf{I}\mu)\, g = \mu \bar{f}. \tag{4.14}$$

where L is the mesh Laplacian matrix (1.4.1) that discretise the Laplacian operator.
Let us replace L with L_{norm} defined in (1.4), we obtain:

$$(\mathbf{I} - D^{-1/2} W D^{-1/2} + \mathbf{I}\mu)\, g = \mu \bar{f} \tag{4.15}$$

where D is the mesh degree matrix and W the adjacency matrix. Then, g^* can be given as a closed form solution:

$$g^* = ((1+\mu)\mathbf{I} - D^{-1/2}WD^{-1/2})^{-1}\mu\bar{f}. \tag{4.16}$$

Then g^* represents the output feature map of the convolutional layer.

In the more general case in which $p \geq 1$, imposing the optimality conditions, equation (4.10) is equivalent to:

$$\Delta_p g + \mu\,(g - \bar{f}) = 0, \tag{4.17}$$

$$(\Delta_p + \mathbf{I}\mu)\, g = \mu\bar{f}. \tag{4.18}$$

After recall the mesh p-laplacian definition, Def. 1.3.4, we carry out the discretization $\forall i = 1, \ldots, N_v$,

$$\frac{1}{a_i}\sum_{j:(i,j)\in\mathcal{E}} w_{i,j}\,||g_i - g_j||^{p-2}\,(g_i - g_j) + \mu g_i = \mu\bar{f}_i \tag{4.19}$$

$$\frac{1}{a_i}\,g_i\sum_{j:(i,j)\in\mathcal{E}} w_{i,j}\,||g_i - g_j||^{p-2} - \frac{1}{a_i}\sum_{j:(i,j)\in\mathcal{E}} w_{i,j}\,||g_i - g_j||^{p-2}g_j + \mu g_i = \mu\bar{f}_i$$

$$\Big(\frac{1}{a_i}\sum_{j:(i,j)\in\mathcal{E}} w_{i,j}\,||g_i - g_j||^{p-2} + \mu\Big)\,g_i = \frac{1}{a_i}\sum_{j:(i,j)\in\mathcal{E}} w_{i,j}\,||g_i - g_j||^{p-2}g_j + \mu\bar{f}_i.$$

Set $\alpha_i = \dfrac{1}{\left(\frac{1}{a_i}\sum_{j:(i,j)\in\mathcal{E}} w_{i,j}\,||g_i - g_j||^{p-2} + \mu\right)}$, $\beta_i = \alpha_i\mu \in \mathbb{R}$, (4.17) is equivalent to:

$$g_i = \alpha_i\frac{1}{a_i}\sum_{j:(i,j)\in\mathcal{E}} w_{i,j}\,||g_i - g_j||^{p-2}g_j + \beta_i\bar{f}_i, \qquad i = 1, \ldots, N_v \tag{4.20}$$

Let us consider the linear system (4.20) as the stationary state of an evolutionary process, the solution can be obtained through an iterative method, that is,

$$g_i^{k+1} = \alpha_i^k \sum_{j:(i,j)\in\mathcal{E}} M_{i,j}^k\, g_j^k + \beta_i^k\,\bar{f}_i\,,\ i = 1, \ldots, N_v \tag{4.21}$$

with $M^k \in \mathbb{R}^{N_v\times N_v}$, $\alpha^k, \beta^k \in \mathbb{R}^{N_v}$ s.t.

- $M_{i,j}^k = \frac{1}{a_i} w_{i,j} \, ||g_i^k - g_j^k||^{p-2} \in \mathbb{R}$,
- $\alpha_i^k = \frac{1}{\left(\frac{1}{a_i} \sum_{j:(i,j)\in\mathcal{E}} w_{i,j} \, ||g_i - g_j||^{p-2} + \mu \right)} \in \mathbb{R}$,
- $\beta_i^k = \alpha_i^k \mu \in \mathbb{R}$.

We rewrite the equation in a matrix form, we obtain the formula

$$g^{k+1} = \alpha^k M^k g^k + \beta^k \bar{f} \tag{4.22}$$

in which the α^k, M^k, β^k are updated according to (4.21). This iterative result is independent of the setting of the initial value. In this case the output of the convolutional layer is $g^* = g^K$, where K is the number of iterations.

Based on the defined convolutional operator, we introduce a network architecture.

Definition 4.5.1. Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh with an adjacency matrix W, $f = (f^1, \dots f^p)$ and $g = (g^1, \dots, g^q) \in \mathbb{R}^{N_v \times q}$ the input and output features. Set $p \geq 1$ and the maximum number of iteration K, we define the ^{p}GNNs as

$$\begin{aligned} &\tilde{f} = ReLU(f\Theta_0) \\ &g^{k+1} = \alpha^k M^k g^k + \beta^k \tilde{f}, \, \forall k = 0, \dots, K-1 \\ &g = softmax(\tilde{g}^K \Theta_1) \end{aligned} \tag{4.23}$$

where the learnable parameters are Θ_0, Θ_1.

Chapter 5

Spatial Domain Convolutional Operators

The second category of graph neural networks working on meshes is characterised by the emergence of a spatially-based convolutional operator. Basically, the spatial based convolution consists of a subdivision of the domain into charts. Then the convolution operation on images,

$$f * g,$$

can be expressed by:

$$\sum_j g_j D_j(x) f, \tag{5.1}$$

where g_j denotes the template coefficients applied on the patch extracted at each point and

$$D_j(x)f = \int_{\mathcal{X}} f(x')u_j(x, x')dx',\ j = 1 \dots J \tag{5.2}$$

is the *patch operator* specified by the weighting function u_j.
Spatial approaches define convolutions directly on the meshes based on the topology. They usually follow the same pattern:

- the vertex feature vectors are transformed using some sort of projection;
- they are aggregated by a permutation-invariant function;

- the feature vector of each vertex is updated based on its current values and the aggregated neighbourhood representation.

In this chapter, we will analyse two popular mesh convolutional operators, that of the GraphSAGE model and that of the graph attention network (GAT). Both belong to the spatial domain convolution category. For more information we invite the reader to consult [9] for the first convolutional operator, [10, 11] for the second.

5.1 SAGE Convolutional Operator (SAGEConv)

A very popular spatial based convolutional operator is the one characterizing the GraphSAGE (SAmple and aggreGatE) framework.
It introduces aggregation function to define the convolution in the spatial domain and learns the embedding of each vertex in an inductive way. The aggregation function is essentially the aggregation of the neighborhood information of the vertices, and its output should be invariant to the order of vertices. That is, the input order will not affect the output result of the aggregation function, such as mean, sum and max functions. In this way, each vertex is represented by the aggregate result of its neighborhood.

Definition 5.1.1. Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh, we consider the input feature map associated $f = (f_1, \dots f_p) \in \mathbb{R}^{N_v \times p}$. We denote $\forall v \in \mathcal{V}$, $f(v) = (f_1(v), \dots, f_p(v)) \in \mathbb{R}^p$ the vector of features associated to the vertex. All vertices $v \in \mathcal{V}$ are updated through the *SAGE convolutional operator* as following:

$$\tilde{f}(v) = AGGREGATE(f(u), \forall u \in N(v)), \tag{5.3}$$

$$\tilde{g}(v) = \sigma(\mathcal{W} \cdot CONCAT(f(v), \tilde{f}(v))), \tag{5.4}$$

$$g(v) = \frac{\tilde{g}(v)}{||\tilde{g}(v)||_2}, \tag{5.5}$$

where $g = (g_1, \dots g_q) \in \mathbb{R}^{N_v \times q}$ is the output feature map, $N(v)$ is the set of neighboring vertices of vertex $v \in \mathcal{V}$, $AGGREGATE$ represents the differentiable aggregator function, $CONCAT : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^{2p}$ is the function of concatenation, $\mathcal{W} \in \mathbb{R}^{q \times 2p}$ is a weight matrix of learnable parameters, σ is a nonlinear activation function.

In 5.4, an alternative to concatenation is the sum, the formula turns to

$$\tilde{g}(v) = \sigma(\mathcal{W}_0 f(v) + \mathcal{W}_1 \tilde{f}(v)), \tag{5.6}$$

where $\mathcal{W}_0, \mathcal{W}_1 \in \mathbb{R}^{q \times p}$ (note that the number of learnable parameter doesn't change).

Remark 5.1.1. A sequence of K SAGE convolutional layers allows to aggregate information on the K-ring neighborhood

$\forall k \in \{1, \dots, K\}$, the k-th layer is defined by its aggregation function $AGGREGATE_k$ and a weight matrix $\mathcal{W}_k$ used to propagate between different layers of the model.

5.2 Graph Attention Network Convolution (GAT-Conv)

Graph attention networks are spatial-based convolution networks in which attention mechanism is used to determine the importance of each neighbor vertex to the center vertex of a mesh when the neighbor information of vertices is aggregated.

The convolutional layer is built as following.
Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh and $f = (f_1, \dots f_p) \in \mathbb{R}^{N_v \times p}$ the feature map that is the input to our layer. We denote $f(v) = (f_1(v), \dots f_p(v))$ the feature vector of a single vertex. The layer produces a new feature map (with potentially a different size), $g = (g_1, \dots g_q) \in \mathbb{R}^{N_v \times q}$, as its output.

1. As an initial step, a shared linear transformation, parametrized by a learnable weight matrix, $\mathcal{W} \in R^{qxp}$, is applied to each vertex.

2. Next, a shared learnable attention mechanism $a : \mathbb{R}^q x \mathbb{R}^q \to \mathbb{R}$ computes attention coefficients, that is $\forall\, v \in \mathcal{V}$

$$e_{v,u} = a(\mathcal{W}f(v), \mathcal{W}f(u)), \tag{5.7}$$

   with $u \in N_v$, that is, we only compute $e_{v,u}$ for vertices u s.t. $(u, v) \in \mathcal{E}$, including v.
   The attention coefficient indicate the importance of vertex u's features to vertex v.

3. To make coefficients easily comparable across different vertices, $\forall v \in \mathcal{V}$ we normalize them across all choices of $u \in N_v$ using the softmax function:

$$\alpha_{u,v} = softmax_u(e_{u,v}) = \frac{\exp(e_{u,v})}{\sum_{w \in N_v} \exp(e_{v,w})} \tag{5.8}$$

4. Once obtained, the normalized attention coefficients are used to compute a linear combination of the features corresponding to them, to serve as the final output features for every vertex. Finally, a non-linearity, σ, can be applied.

Now, we have the ingredients for define the convoluional layer:

Definition 5.2.1. Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh, let f, g be the input and output feature maps of the layer, $\mathcal{W}$ the weight matrix and $a_{u,v}$ the attention coefficient described previously. We define $\forall v \in \mathcal{V}$ the *GAT convolutional operator* as

$$g(v) = \sigma\Big(\sum_{u \in N_v} \alpha_{u,v} \mathcal{W} f(u) \Big). \tag{5.9}$$

Remark 5.2.1. Since it is beneficial to stabilize the learning process of self-attention, several networks use a multi-head attention.

Specifically, K independent attention mechanisms execute the transformation, and then their features are concatenated, resulting in the following output feature representation:

$$g(v) = CONCAT_{k=1...K}\Big(\sigma(\sum_{u\in N_v} \alpha^k_{u,v}\mathcal{W}f(u))\Big) \tag{5.10}$$

where $\alpha^k_{u,v}$ are normalized attention coefficients computed by the k^{th} attention mechanism and $\mathcal{W}^k$ is the corresponding input linear transformation's weight matrix.
If we perform multi-head attention on the final (prediction) layer of the network, we employ averaging:

$$g(v) = \sigma\Big(\frac{1}{K}\sum_{k=1}^{K}\sum_{u\in N_v} \alpha^k_{u,v}\mathcal{W}^k f(u)\Big). \tag{5.11}$$

5.3 Discrete Beltrami Flow on Meshes

The research activity is aimed at defining a unified mathematical structure that describes the convolutional operators on mesh. An attempt is to make the operators descend from the discretization of the Beltrami flow.

Given the definition of the Laplace operator on meshes, now we are able to discretize non-Euclidean diffusion PDEs. In particular in this section we will see the discretized Beltrami flow. For more information consult [14].

From the definition of the mesh Laplacian operator, Def. 1.3.3, follows that:

Definition 5.3.1. Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh, we define a function $f : [0, +\infty[\times\mathcal{V} \to \mathbb{R}^{N_c}$ s.t. $\forall\, t \in [0, +\infty[,\ \ f(t) \in \mathbb{R}^{N_v \times N_c}$ is the X feature map at time t. For all $v \in \mathcal{V}$ we denote $f(t, v)$ the vector of features associated to the vertex. We consider the *discrete Beltrami flow* to be the discrete diffusion equation:

$$\frac{\partial f(t,v)}{\partial t} = \sum_{u:(u,v)\in\mathcal{E}} a(f(t,u), f(t,v))\,(f(t,u) - f(t,v)) \tag{5.12}$$

where the function $a : \mathbb{R}^{N_c} \times \mathbb{R}^{N_c} \to \mathbb{R}$ is the diffusivity map controlling the diffusion strength between nodes and it is assumed to be normalised:
$\forall v \in \mathcal{V}$,

$$\sum_{u:(u,v)\in\mathcal{E}} a(f(t,u), f(t,v)) = 1. \tag{5.13}$$

The solution to the diffusion equation is computed by applying scheme multiple times in sequence, starting from some initial feature map $\bar{f}$. Equation (5.12) is solved numerically by replacing the continuous time derivative $\frac{\partial f(t)}{\partial t}$ with forward time difference:

$$\frac{f(v)^{(k+1)} - f(v)^{(k)}}{\tau} = \sum_{u:(u,v)\in\mathcal{E}} a(f(u)^{(k)}, f(v)^{(k)}) \, (f(u)^{(k)} - f(v)^{(k)}) \tag{5.14}$$

Here k denotes the discrete time index (iteration), while τ is the time step (discretiation parameter).

We rewrite the formula (5.12) in matrix form:

$$\begin{cases} \frac{\partial f(t)}{\partial t} = (A(t) - \mathbf{I}) f(t) \\ f(0) = \bar{f}, \end{cases} \tag{5.15}$$

where $t \geq 0$ and $A(t) = (a(v,u))(t)$ is s.t.

$$a(v,u)(t) = \begin{cases} a(f(t,v), f(t,u)) & \text{if } (v,u) \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{5.16}$$

Rewriting 5.14 in matrix form leads to the *explicit Euler scheme*:

$$f^{(k+1)} = f^{(k)} + \tau(A^{(k)} - I) f^{(k)} = Q^{(k)} f^{(k)}, \tag{5.17}$$

With f, A, τ, k defined as above, and Q defined as following:

$$Q(v,u)(t) = \begin{cases} \tau \, a^{(k)}(f(v)^{(k)}, f(u)^{(k)}) & \text{if } u \neq v \\ \tau \, a^{(k)}(f(v)^{(k)}, f(u)^{(k)}) + (1-\tau) & \text{otherwise.} \end{cases} \tag{5.18}$$

Remark 5.3.1. The network consisting of k GAT stacking layers can be interpreted as a method to solve numerically a particular setting of the discrete Beltrami Flow.

In fact, let us consider the learnable parametric attention functions play the role of diffusivity, assuming a discretization of time step τ, the result of each layer of the network represent the updated features of the mesh at the k-th iteration (time interval).

Chapter 6

Mesh Denoising using Graph U-Net

In the following we report a practical application of the graph neural networks theory read in the previous chapters to the reconstruction of noisy meshes. The task is to use the GNNs just mentioned to perform mesh denoising on surfaces damaged due to imperfect scanner measurements. The basic concepts related to graph neural networks were introduced. In particular, we describe the graph U-Net architecture in Sec. 3.3 and we list in Chapter 4 and in Chapter 5 some of the well-known mesh convolutional operators: Cheb-Conv and GCN-Conv as frequency domain convolutional operators, SAGE-Conv and GAT-Conv as spatial domain convolutional operators. Our aim is to test how convolutional operators work and to analyze the role of pooling/unpooling operators. For the tests we exploit PyTorch and PyTorch Geometric libraries, which provide a suitable representation of mesh as input of graph neural networks and an implementation of convolutional, pooling and unpooling operators.

6.1 Dataset Setting

We design neural networks to eliminate noise from corrupted meshes. To perform supervised learning we need a sample made up of noisy meshes accompanied by the corresponding ground truths. In this section we describe the dataset used to train, evaluate and test neural networks. Refer to Def. 1.1.1 for the definition of polygonal mesh.

For each model of the dataset we denote $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ the original mesh (the ground truth) and $\tilde{X} = (\tilde{\mathcal{V}}, \mathcal{E}, \mathcal{T})$ the mesh corrupted by noise. The original meshes are given, we have a set of 32 files available named as follows: 'Guided-trim-star_rescaled.obj', 'GuidedBunny.obj', 'GuidedJulius.obj', 'Macchia.obj', 'Original.obj', 'Original2.obj', 'angelo.obj', 'ant.ply', 'bird.ply', 'block.obj', 'bunny_st.obj', 'bunny_st_small.obj', 'bust.ply', 'cube_hole.obj', 'cube_hole_tri.obj', 'distcap.obj', 'dolphin.ply', 'fandisk.ply', 'foot-open.ply', 'gargo50k.obj', 'gargoyle.obj', 'hand.ply', 'hand38k.obj', 'horse.ply', 'igea-coarse.obj', 'igea.obj', 'mech1.ply', 'mech2.ply', 'mechpart.obj', 'oilpmp.obj', 'sphere6.ply', 'teddy.ply'.
Each file describes the ground truth, $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, of a specific model. It contains the vertices coordinates $v_i = (x_i, y_i, z_i)$, $\forall\, i = 1, \dots, N_v$ and the indices at the vertices that make up the faces, $(i, j, k) \in \mathcal{T}$, with $i, j, k = 1, \dots, N_v$.
From the ground truth recorded in the file, we need to build the corresponding noisy mesh to carry out the training. Recall that mesh denoising is an inverse ill posed problem since both the ground truth and the noise level are unknown. To make it tractable the noise on each vertex position is assumed to follow a Gaussian distribution and to be independent and identically distributed. The noisy mesh $\tilde{X} = (\tilde{\mathcal{V}}, \mathcal{E}, \mathcal{T})$ is obtained by the ground truth $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ perturbing vertex coordinates $v_i \in \mathcal{V}$, $\forall i = 1, \dots, N_v$. The vertex coordinates of the noisy mesh are defined as follows:
$\forall\, i = 1, \dots, N_v$,

$$\tilde{v}_i = v_i + \text{noise\_level}\,(\nu_i \odot n_i),$$

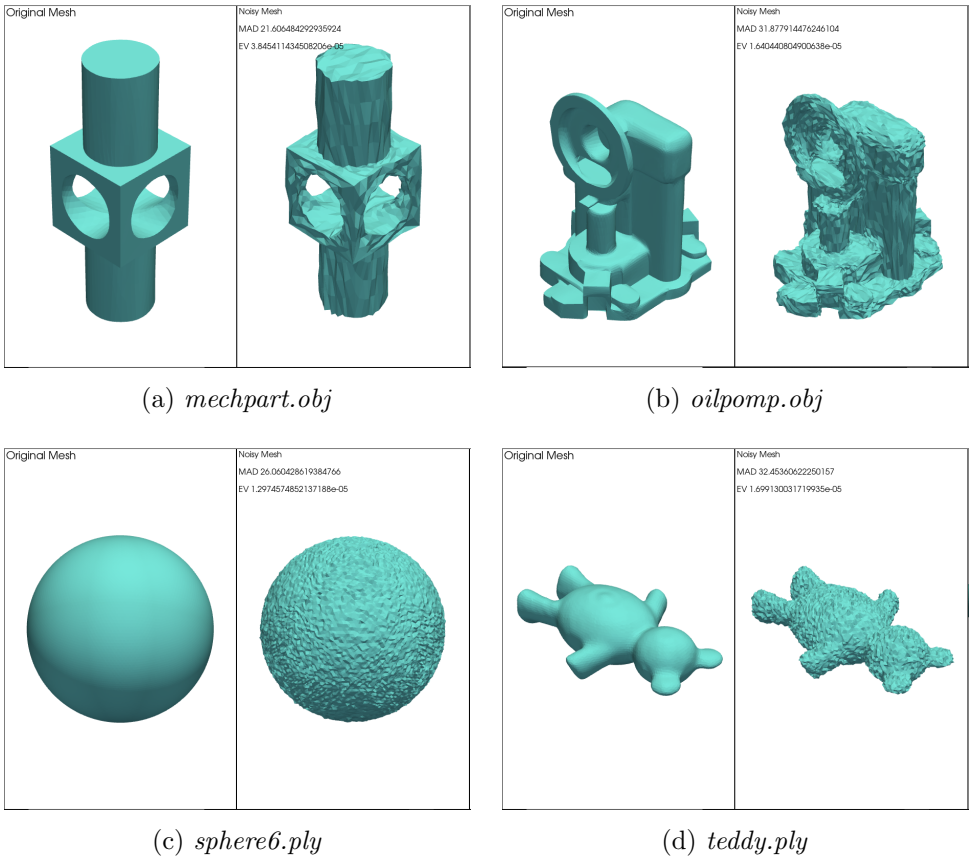
where $n_i \in \mathbb{R}^3$ is the normal vector to the vertex, $\nu_i \in \mathbb{R}^3$ is a vector of random variables independent and identically distributed as a Gaussian with mean 0 and standard deviation 1. The parameter noise_level is necessary to reduce the noise according to the length of the edges that make up the mesh, firstly we set the value at 0.005.

To obtain a data type compatible with machine learning techniques, it is desirable to represent all meshes as graphs: the graph nodes would correspond to the vertices of the mesh and two nodes would be associated if there was a face to which both respective vertices belong. In this regard, the PyTorch Geometric library provides the object *torch_geometric.data.Data*. We have implemented a function, called *mesh2Data*, that takes as input the mesh in the ".obj" or ".ply" file and the noise parameter noise_level, and returns a graph as *torch_geometric.data.Data*, which includes three tensors, x, *edge_index* and y, related to node features, edge indices and labels. In the feature tensor x are recorded the vertex coordinates of the noisy mesh, while in the label tensor y are recorded the coordinates related to the ground truth after centering and scaling. It directly follows that the size of both the two described tensors is the number of nodes that define the mesh multiplied by three. Finally, the *edge_index* tensor is a matrix of dimension two times the number of edges. The number of edges follows from the description of the faces in the input file and corresponds to the number of faces multiplied by three.
In some cases it's useful to record in the *Data* object, in addition to the three tensors, for each face of the model the indices to the three vertices, we denote the variable as *pos*. To clarify, we report below the data describing 'mechpart.obj':

$$Data(x = [1900, 3], edge\_index = [2, 22848], y = [1900, 3], pos = [3808, 3]).$$

Finally, to accomplish what we set out to do, our dataset needs to be split into three subsets: the training, the evaluating and the testing dataset. The meshes belonging the dataset are listed in Table 6.1. The Fig. 6.1 shows

the ground truth mesh and the noisy mesh with noise_level = 0.005 for each model of the testing dataset.



(a) *mechpart.obj* (b) *oilpomp.obj*

(c) *sphere6.ply* (d) *teddy.ply*

Figure 6.1: Testing dataset

filename	vertices	edges	faces
Train dataset			
Guided-trim-star_rescaled.obj	5192	62304	10384
GuidedBunny.obj	34834	416706	69451
GuidedJulius.obj	36201	431472	71912
Macchia.obj	14846	176622	29437
Original.obj	4610	55296	9216
Original2.obj	10443	125292	20882
angelo.obj	29670	351744	58624
ant.ply	7038	84432	14072
bird.ply	6475	77676	12946
block.obj	8771	105300	17550
bunny_st.obj	2503	29808	4968
bunny_st_small.obj	2503	29808	4968
bust.ply	25467	305580	50930
cube_hole.obj	10232	122880	20480
cube_hole_tri.obj	2552	30720	5120
distcap.obj	7513	89280	14880
dolphin.ply	7573	90852	15142
fandisk.ply	6475	77676	12946
foot-open.ply	10010	119844	19974
gargo50k.obj	25006	300000	50000
gargoyle.obj	100002	1200000	200000
hand.ply	6607	79260	13210
hand38k.obj	38219	458628	76438
horse.ply	8078	96912	16152
Eval dataset			
igea-coarse.obj	8268	99192	16532
igea.obj	134345	1612116	268686
mech1.ply	8759	105084	17514
mech2.ply	10400	124776	20796
Test dataset			
mechpart.obj	1900	22848	3808
oilpmp.obj	10274	123264	20544
sphere6.ply	16386	196608	32768
teddy.ply	9548	114552	19092

Table 6.1: Dataset

6.2 Metrics

To evaluate the neural networks we trained, it was necessary to define metrics on which to base ourselves to calculate the error made during denoising. In this section we introduce two metrics we used.

Suppose there are two meshes, the ground truth $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ and the mesh restored by the neural network $Y = (\bar{\mathcal{V}}, \mathcal{E}, \mathcal{T})$, evaluating a model involves calculating their degree of similarity. Therefore, we apply two different metrics:

- *MAD metric*: the value is related to the width of the angle between the normal vectors to the corresponding faces in the two meshes.
- *EV metric*: the value describes the distance between the corresponding vertices in the two meshes.

Basically, the MAD metric:

1. computes the unitary normals to the faces for both the two meshes;
2. for each face $(i, j, k) \in \mathcal{T}$, applies the inner product to the two normals, rispectively, of the restored mesh and the ground truth;
3. returns the mean of the arccosine of the values obtained for each face, i.e. the angles between the two normals.

Formally, meshes are defined by the vertices coordinates and the indices to the vertices that make up the faces. Let's denote v_j a generic vertex and f_i a generic face of the ground truth X. To each vertex v_j of the ground truth corresponds a vertex $\bar{v}_j$ of the restored mesh Y with the restored vertex coordinates. Likely, to each face f_i of the ground truth we can associate a face $\bar{f}_i$ on the restored mesh made up by the correspondent vertices. Let N_f be the number of faces, we denote n_i, $\bar{n}_i$ the unitary normal vectors to the

faces f_i, $\bar{f}_i$, $\forall i = 1 \dots N_f$, then

$$MAD(X,Y)^2 = \frac{1}{N_f} \sum_{i=1}^{N} {}_f\langle n_i, \bar{n}_i \rangle.$$

The EV metric corresponds to the $\mathcal{L}^2$-metric . Given $\mathcal{V} = \{v_i : i = 1, \dots, N_v\}$ and $\bar{\mathcal{V}} = \{\bar{v}_i : i = 1, \dots, N_v\}$, the sets of vertex coordinates respectively of the ground truth X and the restored mesh Y. Then, the returned error is

$$EV(X,Y) = \sum_{i=1}^{N_v} ||v_i - \bar{v}_i||_2^2.$$

6.3 Numerical Experimentation Methodologies

Once described the dataset and the metrics used to compare neural networks performance, we introduce the results obtained from the trained neural networks. The experimental phase developed in the following points:

- **Graph convolutional operator choice** (Sec. 6.3.1): we train neural networks with graph U-Net architecture by changing the convolutional operator;
- **Architecture choice, Graph U-Net versus Basic Net** (Sec.6.3.2): we investigate the role of pooling and unpooling operators;
- **Loss function strategies**(Sec.6.3.3): we enrich the loss function with two additional terms to control the curvature;
- **Robustness to the noise**(Sec.6.3.4): we test the graph U-Net neural network with the GAT-Conv operator on a testing dataset with a lower noise level;
- **Layer output results**(Sec.6.3.5): we visualize the output of intermediate layers.

6.3.1 Graph Convolutional Operator Choice

First, we perform mesh denoising by observing how the choice of the convolutional operator affects the results. We train several neural networks with the same architecture replacing the convolutional layer, for the experiment we assume the graph U-Net, its definition is in the Sec. 3.3. The convolutional operators involved in this project are the Cheb-Conv, in which convolution concerns the first and second ring neighbors, the GCN-Conv, the SAGE-Conv and the GAT-Conv.

Let's consider the dataset as described in Sec. 6.1: each mesh is represented by the data type *torch_geometric.data.Data*. Our neural networks accept as input two tensors, x and $edge\_index$. The tensor x contains the vertex coordinates of the noisy mesh and it is the feature map that the network takes as input. The $edge\_index$ tensor defines the mesh structure, it is the analogue of the adjacency matrix.
For the graph U-Net architecture, in particular we use the PyTorch Geometric implementation with the following parameter setting:

- the number of input and output channels equal to three, that is the number of vertex coordinates;
- sixteen hidden channels (this value has been choosen experimentally);
- eight convolutional leyers, this value depends on the assigned depth, in this case we set the value of the depth to four, that is the number of layers on both the encoding part and the deconding part;
- the pool ratios equal to 0.5, the pool ratio concerns the pooling/unpooling operators,in this case the number of nodes is halved for each pooling layer.

The Table 6.2 records errors obtained with both the two metrics, the MAD metric and the EV metric (Section 6.2), from the graph U-Net denoiser using

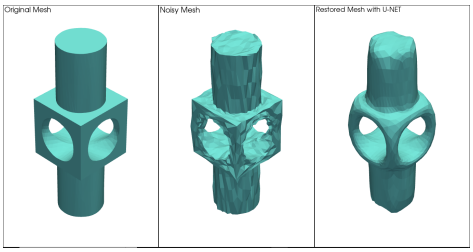
the convolutional operators just mentioned. We highlight in yellow the best result, in red the worst result for each model of the dataset.

	Noisy	Restored GCNConv	Restored ChebConv	Restored SAGEConv	Restored GATConv
MAD metric					
mechpart.obj	21.607	22.118	21.299	18.521	20.118
oilpomp.obj	31.878	21.568	30.354	17.596	13.540
sphere6.ply	26.060	4.655	22.126	10.119	3.691
teddy.ply	32.454	38.187	33.562	12.404	5.244
EV metric					
mechpart.obj	3.845e-05	12.244e-05	5.166e-05	9.330e-05	11.253e-05
oilpomp.obj	1.640e-05	4.783e-05	2.110e-05	4.197e-05	2.867e-05
sphere6.ply	1.298e-05	1.781e-05	1.419e-05	2.793e-05	1.078e-05
teddy.ply	1.699e-05	4.762e-05	1.927e-05	1.935e-05	0.885e-05

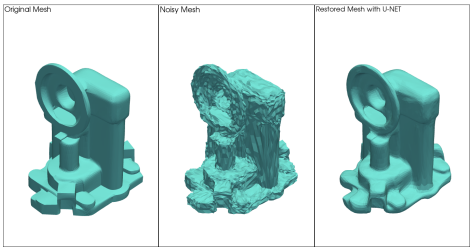
Table 6.2: Graph U-Net denoiser performance

We can observe that, indipendently form the metric, the convolutional operator GAT has the best performance, while the GCN and Chebyshev operators give the worst results. These results are not unexpected, in fact a huge drawback of the frequency domain convolutional operators is that for each vertex of the mesh, its coordinates are updated according to the coordinates of the neighboring vertices, but two vertices belonging to the same ring contribute in the same way. Moreover, the Cheb-Conv operator involves a greater number of vertices than the GCN-Conv in the computation, which does not necessarily imply a better performance, in fact the vertices involved could be too far away and describe other details of the 3D shape. On the other hand; GAT-Conv allows the network to learn the weight of each neighboring vertex regarding the similarity of features.
In the Fig. 6.2, for all the meshes that make up the test dataset, we show the ground truth, the noisy mesh and the mesh restored with the GAT-Conv operator.
It should be noticed, however, that if the mesh is characterized by sharp

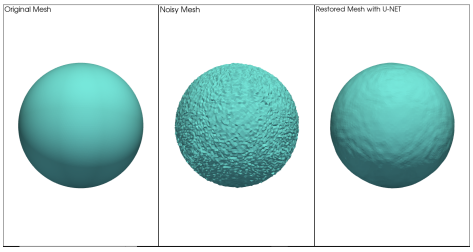
edges, the results change from the used metric: if we compare normal vectors to facets of the restored mesh and of the ground truth GAT achieves higher performance than the other operators, but with regard to vertices coordinates, a grater smoothing causes a move away from the original shape.



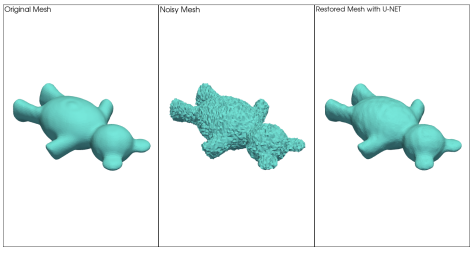
(a) *mechpart.obj*



(b) *oilpomp.obj*



(c) *sphere6.ply*



(d) *teddy.ply*

Figure 6.2: Graph U-Net with GAT-Conv operator obtained results

6.3.2 GNN Architecture Choice: Graph U-Net VS Basic Net

This section is intended to compare two neural network architectures, the graph U-Net and the Basic-Net. The aim is to determine an architecture that benefits our work.

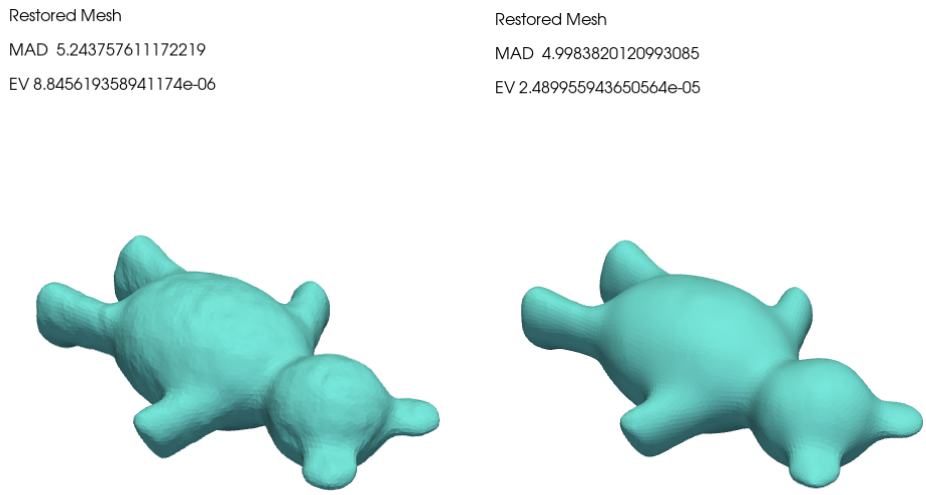
We call Basic-Net a neural network simply consisting of eight stacked convolutional layers. The difference between the two architectures is therefore the presence of pooling and unpooling operators in the Graph U-Net. The Table 6.3 shows the results of both architectures assuming GAT convolutional layers. We observe that the Graph U-Net denoiser is generally better than the Basic-Net denoiser. In the case of the 'teddy.ply' and 'sphere6.ply' meshes, however, the error calculated using the MAD metric is greater if we consider an encoding-decoding architecture.

	Noisy	Restored G-U-Net	Restored Basic Net
MAD metric			
mechpart.obj	21.607	20.118	30.310
oilpomp.obj	31.878	13.540	20.103
sphere6.ply	26.060	3.691	2.807
teddy.ply	32.454	5.244	4.998
EV metric			
mechpart.obj	3.845e-05	11.253e-05	27.750e-05
oilpomp.obj	1.640e-05	2.867e-05	5.651e-05
sphere6.ply	1.298e-05	1.078e-05	2.253e-05
teddy.ply	1.699e-05	0.885e-05	2.490e-05

Table 6.3: Comparison between the two architectures using GAT-Conv operator

As we printed the meshes restored through Graph U-Net neural Network, it's interesting to view the meshes restored without pooling and unpooling

operators intervention. The Fig. 6.3 shows how the Basic Net affects the model. Its stronger smoothing is suitable for model without sharp edges.



(a) *Graph U-Net* (b) *Graph Basic-Net*

Figure 6.3: Differences between graph U-Net and Basic-Net

6.3.3 Loss Function Strategies

One of the main problems of mesh restoration using the proposed neural networks is the strong smoothing that causes the loss of the original shape of the models. To contain this behaviour, we enrich the loss function by combining fidelity terms and we train new neural networks. For the experiment we use the Graph U-Net model with GAT-Conv operator.

We define the loss function as a combination

$$\mathcal{L}(\Theta) = \lambda_1 \mathcal{L}_1(\Theta) + \lambda_2 \mathcal{L}_2(\Theta) + \lambda_3 \mathcal{L}_3(\Theta), \tag{6.1}$$

where Θ represents the learnable parameters of the network and

- $\mathcal{L}_1$ is the *MSE loss function*;
- $\mathcal{L}_2$ is the *curvature fidelity loss function*;
- $\mathcal{L}_3$ is the *flatness loss function.*

We denote λ_i, $\forall i = 1, 2, 3$, the weights of the three loss function terms.

The MSE loss function

At each epoch of the training phase, all the corrupted meshes of the training dataset feed a new neural network. In order to update its weights, the denoiser accuracy is computed comparing each restored mesh Y with the corresponding ground truth X, then their difference is recorded on a loss variable useful to set the new weights of the next neural network.
The MSE provided by the PyTorch Geometric library computes the distance between vertex coordinates. Given the restored mesh $Y = (\bar{\mathcal{V}}, \mathcal{E}, \mathcal{T})$ and the corresponding ground truth $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, the MSE function takes as input the feature associated to each node, i.e. the vertex coordinates, and returns a real value as follows:

$$\mathcal{L}_1(\Theta) = \frac{1}{N_v} \sum_{i=1}^{N_v} \frac{1}{3}[(\bar{x}_i^2 - x_i^2) + (\bar{y}_i^2 - y_i^2) + (\bar{z}_i^2 - z_i^2)], \tag{6.2}$$

where $v_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ are the coordinates of the vertex $v_i \in \mathcal{V}$ of the ground truth and $\bar{v}_i = (\bar{x}_i, \bar{y}_i, \bar{z}_i) \in \mathbb{R}^3$ are the coordinates of the corresponding vertex $\bar{v}_i \in \bar{\mathcal{V}}$ on the restored mesh.

The curvature fidelity loss function

Given the restored mesh $Y = (\bar{\mathcal{V}}, \mathcal{E}, \mathcal{T})$ and its corresponding ground truth $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, let $\bar{g}$ and $g \in \mathbb{R}^{N_v \times 3}$ be two feature maps respectively of Y and X. The curvature fidelity loss function computes

$$\mathcal{L}_2(\Theta) = ||\bar{L}\bar{g} - Lg||_F, \tag{6.3}$$

where $\bar{L}$ and L are the Laplacian matrices associated to the meshes and $||\cdot||_F$ is the Frobenius norm. Recall that, in this case, each row of g is the vector $v_i = (x_i, y_i, z_i) \in \mathbb{R}^3$, similarly for $\bar{g}$.
Let $X = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a mesh and let $g \in \mathbb{R}^{N_v \times 3}$ the feature map, the link between curvature and Laplacian matrix is given by the formula:

$$Lg = hn \tag{6.4}$$

where h is the mean curvature field and n the normal vector field on X.

The flatness loss function

Given the restored mesh $Y = (\bar{\mathcal{V}}, \mathcal{E}, \mathcal{T})$, let $\bar{g} \in \mathbb{R}^{N_v \times 3}$ be its feature maps, the flatness loss function simply computes

$$\mathcal{L}_3(\Theta) = ||\bar{L}\bar{g}||_F, \tag{6.5}$$

this constraint causes the restored mesh to be flat.

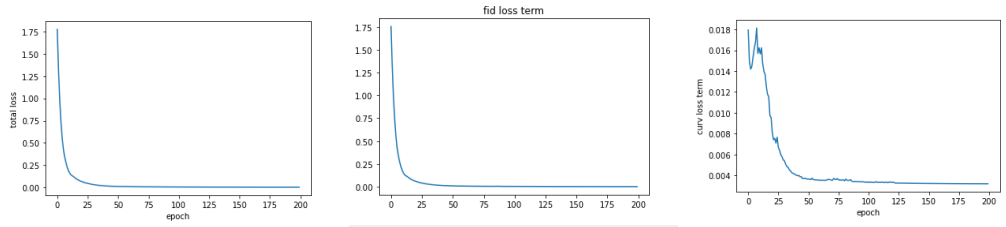
The main challenge is to give each term its proper weight. In the previous experiments we were assuming an MSE loss function, that is, $\lambda_1 = 1$ and $\lambda_2, \lambda_3 = 0$. In order to perform an edge preserve mesh denoising, we enrich the loss function with the two fidelity terms that control curvature.

The Table 6.4 shows the errors calculated by different networks trained with different loss functions. In particular $\lambda_1 = 1$, $\lambda_3 = 0$ and the weight relating to the curvature fidelity λ_2 is gradually increased. Moreover, let us consider the particular case in which $\lambda_2 = 1e - 6$. In the Fig. 6.4 we plot the total training loss function, the MSE loss function and the curvature fidelity loss function values with respect to the epochs: the contribution of the curvature loss function is minimal.

In the second Table 6.5 we add the flatness loss function term to compute the total loss function, the proposal is to decrease the curvature. This

	Noisy	Restored $\lambda_2 = 0$	Restored $\lambda_2 = 1e-7$	Restored $\lambda_2 = 1e-6$	Restored $\lambda_2 = 1e-4$
MAD metric					
mechpart.obj	21.607	20.118	20.145	20.547	23.627
oilpomp.obj	31.878	13.540	13.677	13.837	21.098
sphere6.ply	26.060	3.691	3.798	5.189	26.233
teddy.ply	32.454	5.244	5.309	6.208	28.118
EV metric					
mechpart.obj	3.845e-05	11.253e-05	11.646e-05	11.586e-05	32.081e-05
oilpomp.obj	1.640e-05	2.867e-05	2.750e-05	3.186e-05	18.963e-05
sphere6.ply	1.298e-05	1.078e-05	1.075e-05	2.675e-05	30.669e-05
teddy.ply	1.699e-05	0.885e-05	0.9422e-05	2.314e-05	28.621e-05

Table 6.4: Mesh restored with different contributions of fidelity terms



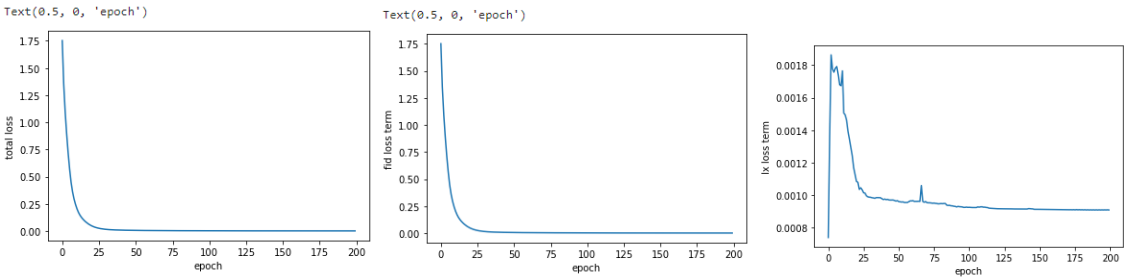
(a) *Total loss* (b) *MSE loss* (c) *Curvature fidelity loss*

Figure 6.4: Loss-plots with weights: $\lambda_1 = 1.0, \lambda_2 = 1e-6, \lambda_3 = 0.0$

procedure lead the mesh to be flat in order to preserve edges. We set $\lambda_1 = 1$, $\lambda_2 = 0$, as in the previous case, increasing the weight related to the flatness loss function term, λ_3, the denoiser performance get worse.
Let us consider the weight $\lambda_1 = 1.0, \lambda_2 = 0.0, \lambda_3 = 1e-07$, in Fig. 6.5 we plot the total training loss function, the MSE loss function term and the flatness fidelity loss function term with respect to the epochs.

	Noisy	Restored $\lambda_1 = 1$ $\lambda_2 = 0$ $\lambda_3 = 0$	Restored $\lambda_1 = 1$ $\lambda_2 = 0$ $\lambda_3 = 1e-7$	Restored $\lambda_1 = 1$ $\lambda_2 = 0$ $\lambda_3 = 1e-5$
MAD metric				
mechpart.obj	21.607	20.118	20.112	20.843
oilpomp.obj	31.878	13.540	13.528	14.554
sphere6.ply	26.060	3.691	4.173	5.392
teddy.ply	32.454	5.244	5.288	6.537
EV metric				
mechpart.obj	3.845e-05	11.253e-05	11.326e-05	17.090e-05
oilpomp.obj	1.640e-05	2.867e-05	2.786e-05	8.241e-05
sphere6.ply	1.298e-05	1.078e-05	1.407e-05	5.754e-05
teddy.ply	1.699e-05	0.885e-05	0.811e-05	4.709e-05

Table 6.5: Mesh restored with different contributions of fidelity terms

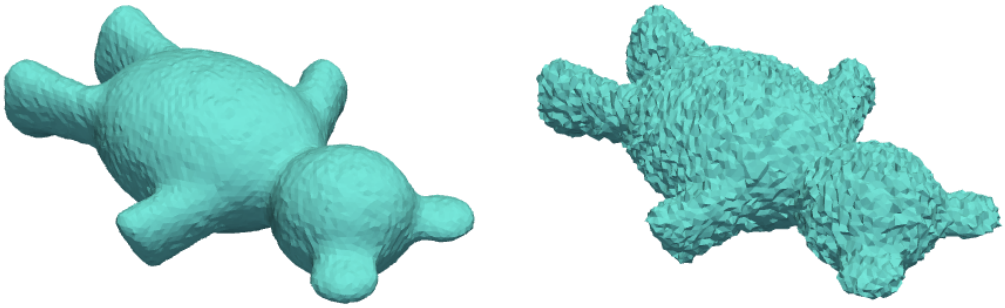


(a) *Total loss* (b) *L2 fidelity loss* (c) *Lx loss*

Figure 6.5: Loss-plots with weights: $\lambda_1 = 1.0, \lambda_2 = 0.0, \lambda_3 = 1e-07$

6.3.4 Robustness to the Noise

Let us consider a new testing dataset. The prevoius dataset has been generated as discribed in the Sec. 6.1. We generate the same dataset decreasing the level of perturbation, that is, we set the parameter noise_level equal to 0.001 (a lower value then 0.005). In Fig. 6.6 we show the model 'teddy', perturbed with both the two noise_level settings: $0.001, 0.005$.



(a) *noise_level = 0.001* (b) *noise_level = 0.005*

Figure 6.6: Mesh 'teddy' perturbed with two different noise levels

In Table 6.6 we record the results obtained by mesh denoing performed by the neural network with U-Net architecture and GAT convolutional layers trained with a dataset perturbed by noise_level 0.005.

	Noisy	Restored
MAD metric		
mechpart.obj	4.834	19.433
oilpomp.obj	7.439	12.246
sphere6.ply	5.600	1.236
teddy.ply	7.443	2.122
EV metric		
mechpart.obj	0.775e-05	11.216e-05
oilpomp.obj	0.330e-05	2.843e-05
sphere6.ply	0.259e-05	1.021e-05
teddy.ply	0.341e-05	0.746e-05

Table 6.6: Denoising with U-Net model and GAT convolutional layer

In Fig. 6.7 we show the 'teddy' model perturbed with noise_level= 0.001 and restored with a graph U-Net architecture and GAT convolutional layers.

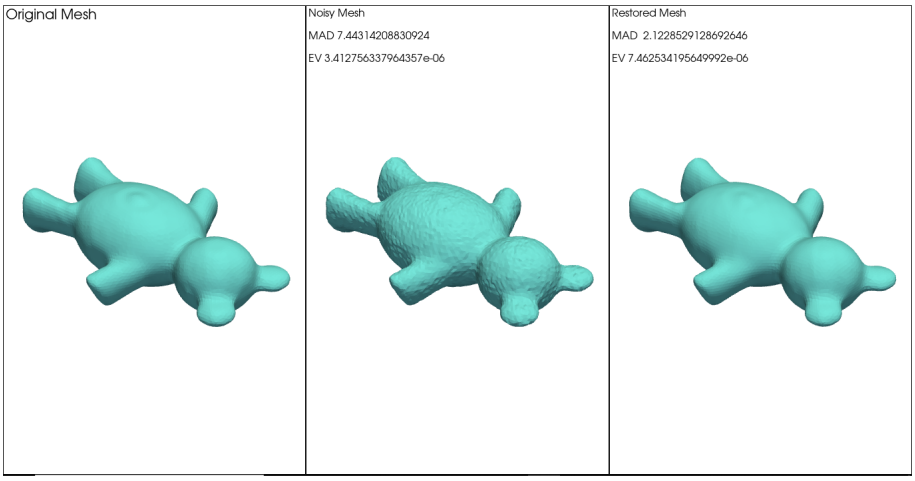


Figure 6.7: Restoration of 'teddy' with noise'level = 0.001

For completeness, in Fig. 6.8, 6.9, 6.11 and 6.10, we visualize the results obtained with the GAT-Conv operator on smooth meshes. In particular we enrich the testing dataset with two meshes:

- 'Julius.obj': $X = (36201, 431472, 143824)$;
- 'nicolo.obj': $X = (14846, 176622, 58874)$.

The denoising performances are recorded in Table 6.7.

6.3.5 Layer Output Results

The aim of this section is the convolutional layers monitoring. Let us consider the Basic Net model made up by eight GAT convolutional layers:

- Input layer: takes as input three features for each vertex and returns sixteen features for each vertex.
- Six hidden layers: take as input sixteen features for each vertex and return sixteen features for each vertex.
- Outout layer: takes as input sixteen features for each vertex and returns three features for each vertex.

	Noisy	Restored G-U-Net	Restored Basic Net
MAD metric			
sphere6.ply	26.060	3.691	2.807
teddy.ply	32.454	5.244	4.998
Julius.obj	52.061	9.893	8.337
nicolo.obj	39.952	8.038	9.203
EV metric			
sphere6.ply	1.298e-05	1.078e-05	2.253e-05
teddy.ply	1.699e-05	0.885e-05	2.490e-05
Julius.obj	0.873e-05	0.635e-05	1.662e-05
nicolo.obj	1.375e-05	1.350e-05	2.928 e-05

Table 6.7: Denoising of smooth meshes

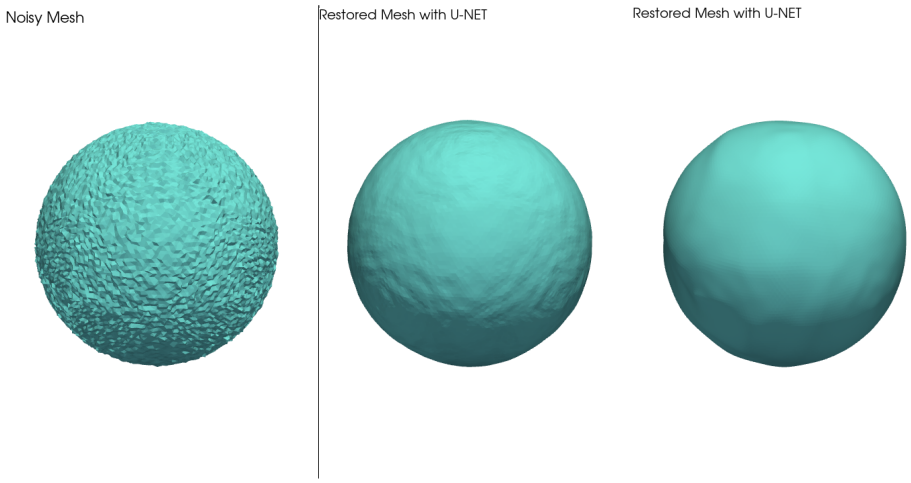


Figure 6.8: 'sphere' restoration

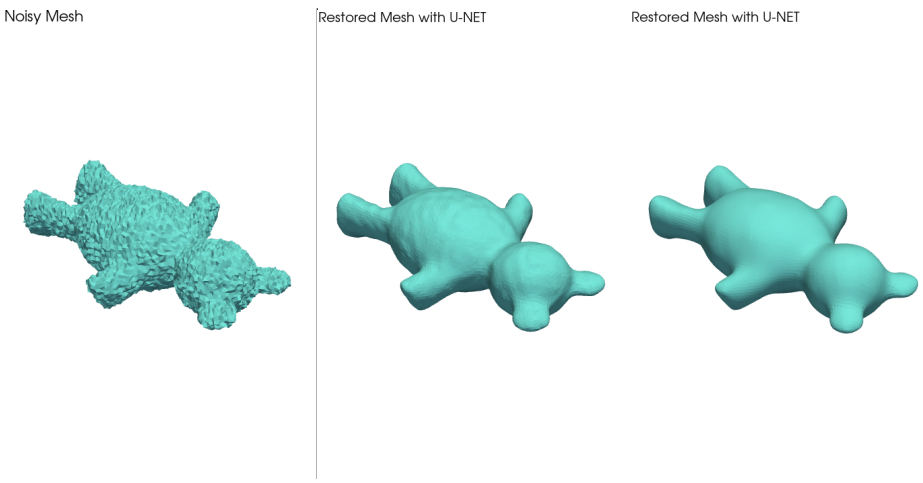


Figure 6.9: 'teddy' restoration

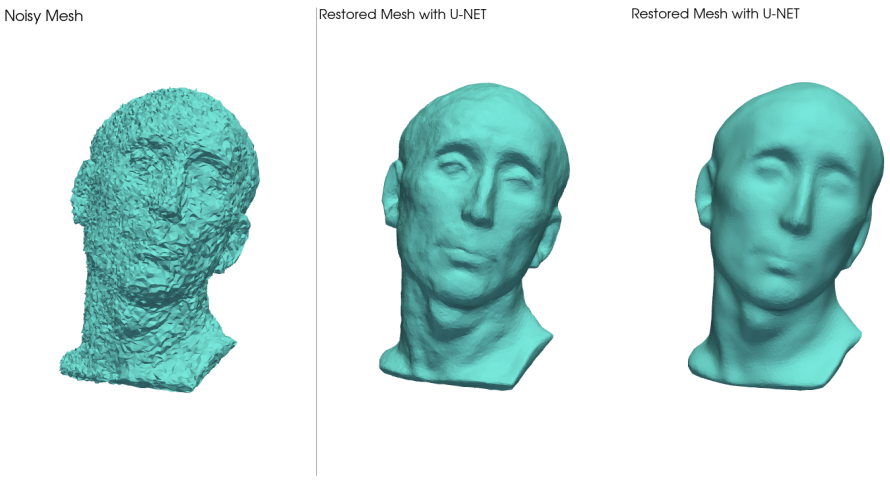


Figure 6.10: 'nicolo' restoration

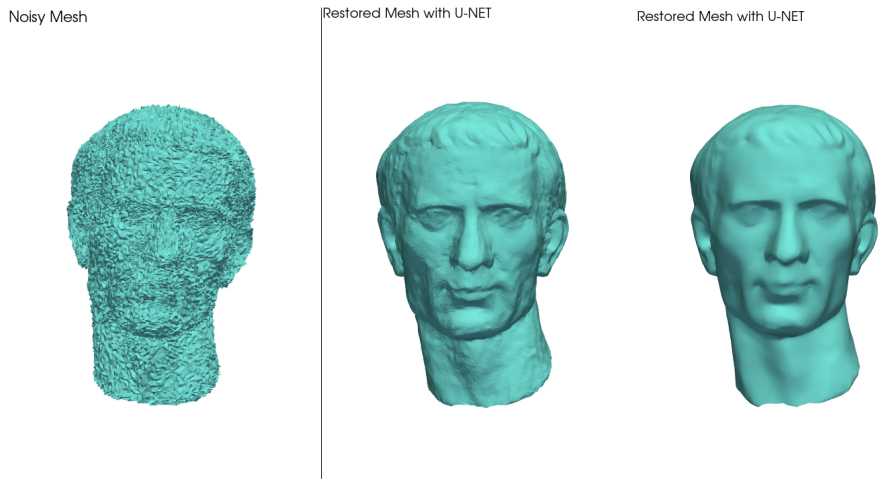
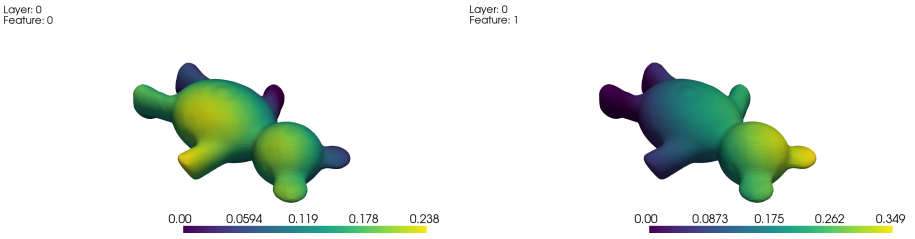


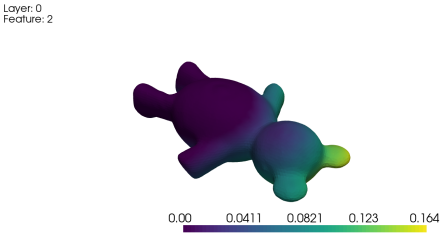
Figure 6.11: 'julius' restoration

We use the 'teddy' mesh to test the trained network, as the GAT convolutional operator is particularly efficient on it. In order to analyse the values of the output vertex features, at each layer we display as many meshes as there are output features (sixteen for the first seven layers, three for the last one) and we assign a colour to mesh vertices that coincides with the values of the feature associated. At the end of this chapter we show the meshes corresponding to the output features of the first, the fourth and the last GAT convolutional layers. In the first layer we can identify two kind of features, in some cases, features highlight a detail of the surface, as in Fig. 6.13.a (Feature 8), in other cases, features are an approximation of vertex coordinates, as in Fig. 6.12.g (Feature 6). It can also be seen that in the fourth layer there are already features that describe the coordinates as in Fig. 6.14.e (Feature 4), moreover greater is the number of the layer, grater is the number of features equal to zero over the entire domain, see both the two figures, Fig. 6.14 and Fig. 6.15 . A possible efficency improvement may be the choice of a smaller number of layers and a smaller number of output features. Finally, Fig. 6.16 shows the output of the last layer, that is vertex coordinates of the restored mesh. This check proof the efficacy of the network.

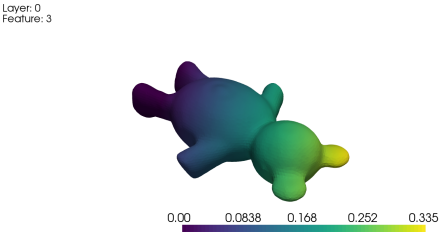


(a) *Feature 0*

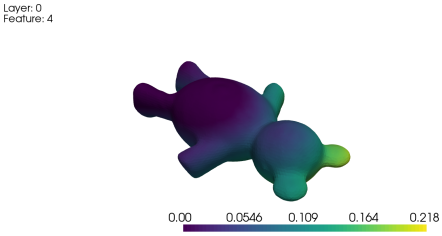
(b) *Feature 1*



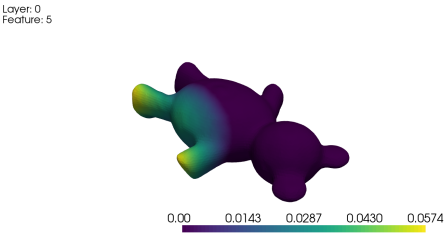
(c) *Feature 2*



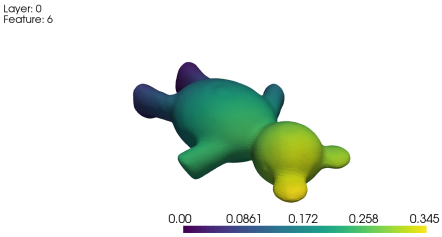
(d) *Feature 3*



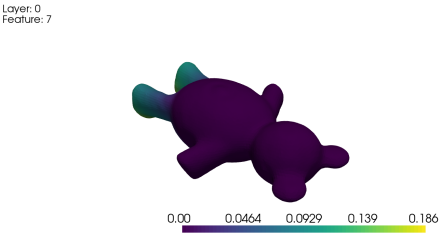
(e) *Feature 4*



(f) *Feature 5*

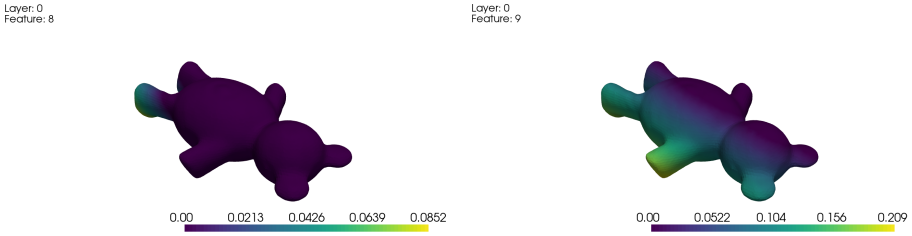


(g) *Feature 6*



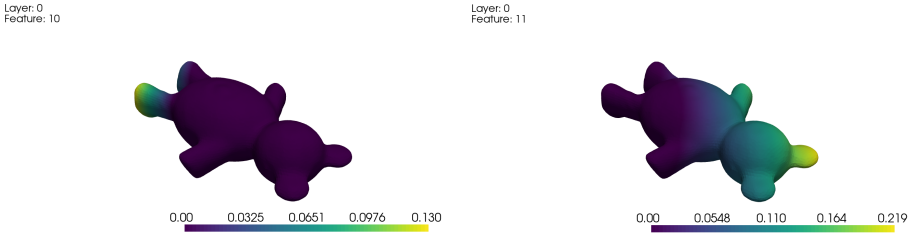
(h) *Feature 7*

Figure 6.12: First layer output, part 1



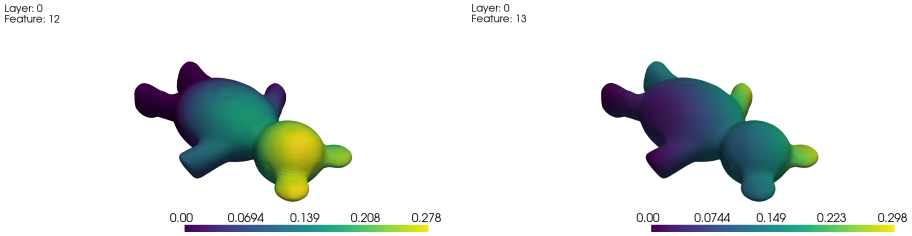
(a) *Feature 8*

(b) *Feature 9*



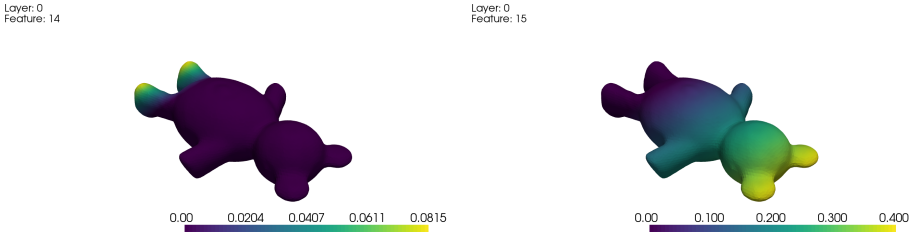
(c) *Feature 10*

(d) *Feature 11*



(e) *Feature 12*

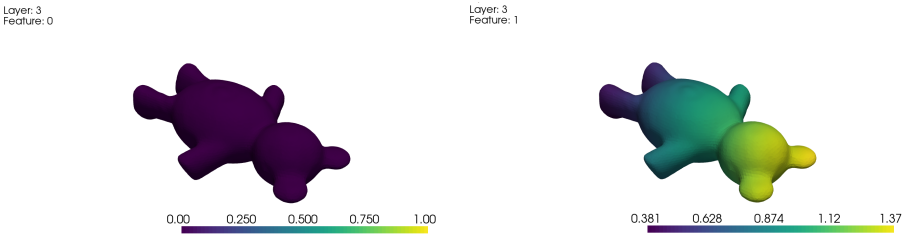
(f) *Feature 13*



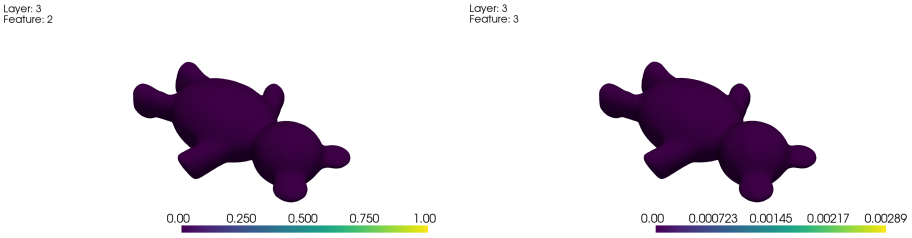
(g) *Feature 14*

(h) *Feature 15*

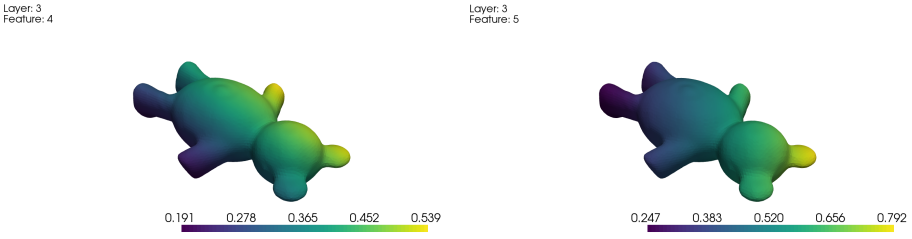
Figure 6.13: First layer output, part 2



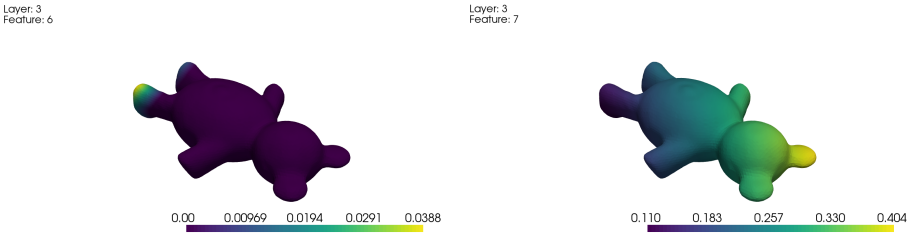
(a) *Feature 0* (b) *Feature 1*



(c) *Feature 2* (d) *Feature 3*

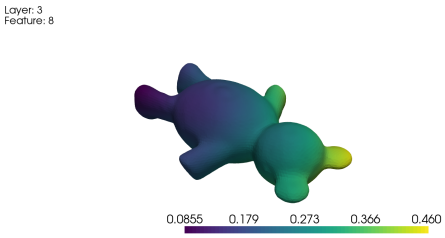


(e) *Feature 4* (f) *Feature 5*

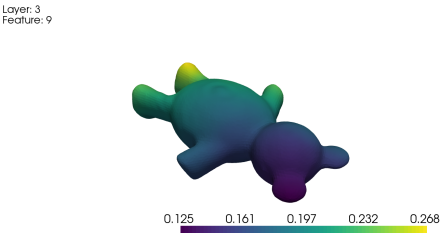


(g) *Feature 6* (h) *Feature 7*

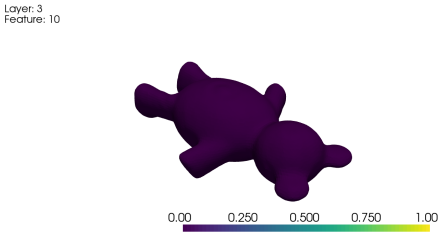
Figure 6.14: Fourth layer output, part 1



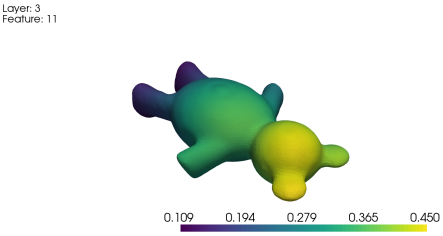
(a) *Feature 8*



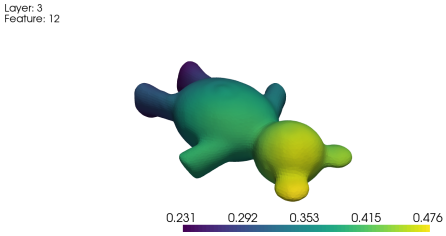
(b) *Feature 9*



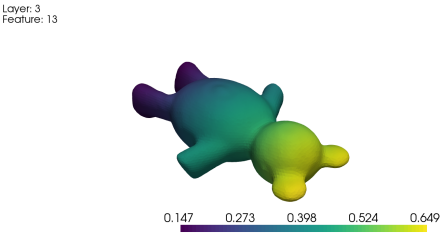
(c) *Feature 10*



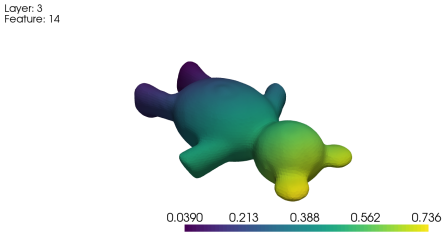
(d) *Feature 11*



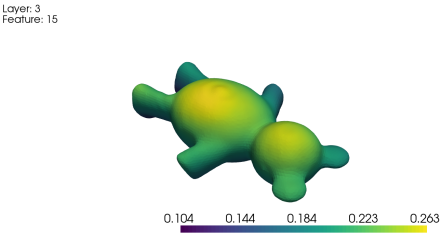
(e) *Feature 12*



(f) *Feature 13*

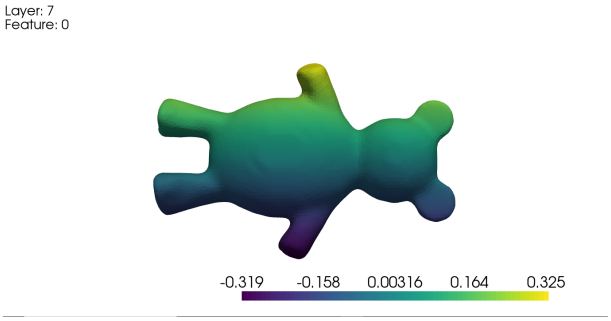


(g) *Feature 14*

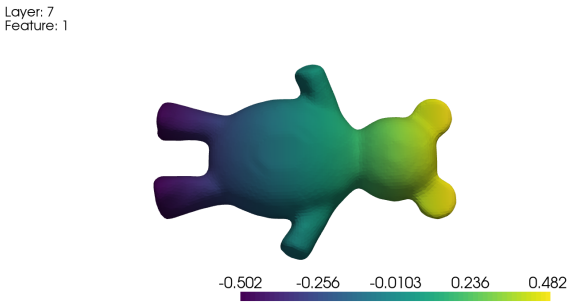


(h) *Feature 15*

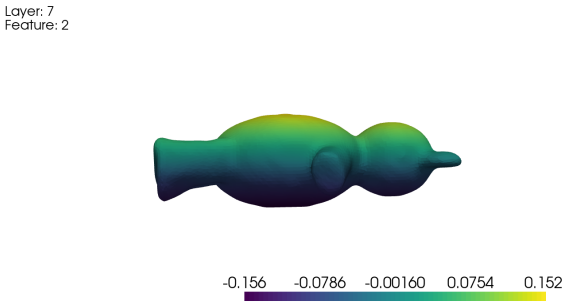
Figure 6.15: Fourth layer output, part 2



(a) *Feature 0*



(b) *Feature 1*



(c) *Feature 2*

Figure 6.16: Last layer output

Conclusions

The aim of this thesis was to analyse the convolutional operators that characterise graph neural networks and to apply them to the denoising of surfaces corrupted due to imperfect scanner measurement. First, we have represented the surfaces as meshes, i.e. a special case of graph. This representation allows us to process 3D shapes with deep learning techniques. Following the investigation of several convolutional operators, four were selected: the GCN-Conv, the Cheb-Conv, the SAGE-Conv and the GAT-Conv. The study of these operators revealed their strengths, in particular the superiority of the GAT operator. In fact, the operator just mentioned is able to aggregate the features of the nodes of a mesh (and in general a graph) by giving the nodes a degree of importance through an attention mechanism. This property also yielded impressive results in the experimental part of the thesis. We have used the GAT-Conv operator both within a network with a graph U-Net type architecture and in the case of a simpler type architecture, which did not include pooling and unpooling operators. Given the good results achieved with GAT convolutional operator on smooth meshes we are interested to continuing our research and further investigate the neural networks which exploit the definition of mesh p-Laplatian reported in Sec 4.5.

Bibliography

[1] Loring W.Tu, *An Introduction to Manifold*, Springer, second edition, 2014

[2] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, Pierre Vandergheynst, *Geometric deep learning: going beyond Euclidean data*, IEEE Signal Processing Magazine, Volume 34, Number 4, Pages 18-42, Jul 2017

[3] Walter Rudin, *Principles of mathematical analysis*, McGraw-Hill Education, third edition, 2015

[4] Loring W.Tu, *Differential Geometry: Connections, Curvature, and Characteristic Classes*, Springer, second edition, 2010

[5] Peter Petersen, *Riemannian Geometry*, Springer New York, NY, 2006

[6] Zonghan Wu and Shirui Pan and Fengwen Chen and Guodong Long and Chengqi Zhang and Philip S. Yu, *A Comprehensive Survey on Graph Neural Networks*, IEEE Transactions on Neural Networks and Learning Systems,Volume 32, Number 1, Pages 4-24, Jan 2021

[7] Defferrard, MichaÃ«l and Bresson, Xavier and Vandergheynst, Pierre, *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*, arXiv, 2016

[8] Kipf, Thomas N. and Welling, Max, *Semi-Supervised Classification with Graph Convolutional Networks*, arXiv, 2016

[9] Hamilton, William L. and Ying, Rex and Leskovec, Jure, *Inductive Representation Learning on Large Graphs*, arXiv, 2017

[10] Monti, Federico and Boscaini, Davide and Masci, Jonathan and Rodolá, Emanuele and Svoboda, Jan and Bronstein, Michael M. *Geometric deep learning on graphs and manifolds using mixture model CNNs*, arXiv, 2016

[11] Velickovic, Petar and Cucurull, Guillem and Casanova, Arantxa and Romero, Adriana and Lió, Pietro and Bengio, Yoshua, *Graph Attention Networks*, arXiv, 2017

[12] Fu, Guoji and Zhao, Peilin and Bian, Yatao, *p-Laplacian Based Graph Neural Network*, arXiv, 2021

[13] Chamberlain, Benjamin Paul and Rowbottom, James and Eynard, Davide and Di Giovanni, Francesco and Dong, Xiaowen and Bronstein, Michael M, *Beltrami Flow and Neural Diffusion on Graphs*, arXiv, 2021

[14] Chamberlain, Benjamin Paul and Rowbottom, James and Gorinova, Maria and Webb, Stefan and Rossi, Emanuele and Bronstein, Michael M., *GRAND: Graph Neural Diffusion*, arXiv, 2021

[15] Gao, Hongyang and Ji, Shuiwang, *Graph U-Nets*, arXiv, 2019

[16] Ronneberger, Olaf and Fischer, Philipp and Brox, Thomas, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, arXiv, 2015

[17] Wei-Lin Chiang and Xuanqing Liu and Si Si and Yang Li and Samy Bengio and Cho-Jui Hsieh, *Cluster-GCN*, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & amp Data Mining, ACM, Jul 2019

[18] Dengyong Zhou and Bernhard Scholkopf, *Regularization on discrete spaces*, 27th DAGM Symposium, Vienna, Austria, August 31 - September 2, 2005, volume 3663, pp. 361-368, 2005.