Computers & Graphics 115 (2023) 69–80

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Section on 3DOR 2023

Learning graph-based features for relief patterns classification on mesh manifolds

Niccolò Guiducci [a], Claudio Tortorici [b], Claudio Ferrari [c], Stefano Berretti [a,*]

[a] University of Florence, Italy
[b] Technology Innovation Institute, Abu Dhabi, United Arab Emirates
[c] University of Parma, Italy

ARTICLE INFO

Article history:
Received 11 April 2023
Received in revised form 27 June 2023
Accepted 3 July 2023
Available online 16 July 2023

Keywords:
Graph Neural Networks
Graph Attention Networks
Relief pattern classification
Features learning

ABSTRACT

Relief patterns represent a surface characteristic that can be seen as the 3D counterpart of the texture concept in 2D images. Such characteristic is well distinct from the 3D object shape but represents a good information source to recognize the object itself. The majority of state-of-the-art techniques for 2D images rely on convolution-based filtering so, the idea of extending such techniques to the mesh manifold domain is quite intriguing as much as challenging. In this paper, we propose a novel approach based on Graph Neural Networks for 3D mesh relief pattern classification. To this end, we designed a bi-level architecture that learns on data structures computed thanks to a mesh resampling algorithm that allows us to represent local surface patches uniformly, while keeping a consistent points order. The local mesh structures are represented by *SpiderPatches*, that aim to capture local features of the 3D mesh surface, providing a fine-grained rich representation of the relief patterns; global structures are instead captured by *MeshGraphs*, whose nodes are SpiderPatches, representing the mesh at a macroscopic level. We tested our architecture using SpiderPatches and MeshGraphs on the original meshes of the SHREC'17 and SHREC'20 relief patterns track datasets, showing superior performance to that reported in the literature using a comparable experimental setting.

1. Introduction

Relief pattern classification is a challenging problem in computer vision and pattern recognition, with applications in areas such as archaeology, geology, and cultural heritage [1–3]. Relief patterns are an additional feature of the surface, which may be linked to texture, and provide complementary information to the shape. The peculiar characteristic of these patterns is that their style is independent of the general shape topology. They can be thought of as the 3D geometric equivalent of textures in 2D images; indeed, they are characterized by some type of regularity and repeatability across the surface. Knitted fabrics, artistic designs, artist styles, or naturally occurring structures like tree barks [4] rock types, or engravings [5] are a few examples. The problem of detecting, retrieving, and classifying relief patterns is now more relevant than ever due to the availability of datasets that include them [5,6]. After the pioneering work of Werghi et al. [7] based on defining Local Binary Patterns on mesh manifolds, a number of approaches have been proposed that rely on ad-hoc designed descriptors [8–11]. However, machine learning based approaches still struggle to succeed in such complex tasks.

Recently, Graph Neural Networks (GNNs) have emerged as a promising approach for processing graph-structured data, including 3D mesh surfaces. GNNs have the ability to capture the relationships between vertices, edges, and faces in the mesh, making them powerful tools for learning effective geometric features. However, analyzing and comparing different graphs demands for a consistent underlying mesh structure across meshes (or parts of them). Thus, an implicit ordering of the vertices is required.

In this work, we propose a novel approach based on Graph Neural Networks for 3D mesh relief pattern classification. We designed a bi-level architecture where local and global mesh structures are represented as graphs, named *SpiderPatch* and *MeshGraph*, respectively. A SpiderPatch aims to capture local features of the 3D mesh surface, providing a fine-grained local representation of the relief pattern; on the other hand, MeshGraph is a graph whose nodes are SpiderPatches, and represent the mesh at the macroscopic level. In order to train the above GNNs, we also developed a mesh sampling algorithm that allows us to represent local surface patches uniformly, while keeping a consistent points order. In this way we can safely represent the mesh (or its parts) as a graph such that the spatial relationships among points is the same across instances.

* Corresponding author.
E-mail addresses: niccolo.guiducci@stud.unifi.it (N. Guiducci), claudio.tortorici@tii.ae (C. Tortorici), claudio.ferrai2@unipr.it (C. Ferrari), stefano.berretti@unifi.it (S. Berretti).

https://doi.org/10.1016/j.cag.2023.07.004

We evaluated the effectiveness of the above solution in the task of relief pattern classification, and demonstrated the advantages of using Graph Attention Networks for this task. Indeed, our results show that the combination of GNNs and SpiderPatches outperforms previous approaches in terms of accuracy and robustness, making it a promising solution for the classification of 3D relief patterns. The main contributions can be summarized as follows:

- We designed a sampling algorithm that induces a local consistent ordering of points based on the mesh surface, allowing us to represent local surface patches as graphs;
- We propose and explore Graph Neural Networks for the task of 3D relief pattern classification with a novel bi-level learning architecture, where two GNNs are hierarchically stacked and trained to capture local surface properties and their global relationships. A comprehensive evaluation of the proposed solution is reported, also in comparison with state-of-the-art approaches.

The rest of the paper is organized as follows: In Section 2, we summarized the related work in the literature of relief pattern classification; The proposed approach is concisely introduced in Section 3, and described in detail in Sections 4, and 5; experimental results are presented in Section 6; finally, conclusions are sketched in Section 7.

2. Related work

Numerous techniques for describing texture patterns based on repeatability, unpredictability, and orientation have been developed for the 2D image domain [12,13]. he most popular methods for extracting this information were local descriptors from convolution-based filtering operations like Gabor or Local Binary Pattern [14]. In the 2D realm, texture analysis is an established field; however, in the 3D domain, geometric pattern analysis presents numerous challenges, like the lack of a unique, predictable and regular data representation or the local–global nature of relief patterns characterization, as reported in the work of Biasotti et al. in [15]. In the following, we summarize works in the literature that are more related to our approach.

A 3D mesh convolution, designed as a direct extension of the 2D operation, was introduced in [9]. Similarly, the widely used 2D approaches, HOG [12] and LBP, were extended to the mesh domain as MeshHOG [16] and MeshLBP [17]. A mesh local binary pattern (meshLBP), which was inspired by LBP in 2D, was presented in [8,17]. This technique developed on the Ordered Ring Facets (ORF) structure on the mesh to have a proper ordered mesh support region for LBP computation, thus creating a regular grid-like structure resembling that of 2D images. This method computed many rings at each facet depending on the demand, and through subsampling, twelve facets were picked at each ring. Additionally, employing geometric attributes discovered at facets, LBP computation used the difference between the central and neighboring facets. This strategy showed some limitation in the case adjacent facets were missing from boundary facets. A couple more methods make use of images generated from 3D meshes, and then locate the texture pattern using image processing methods like morphological operations. Despite being straightforward and simple to use, these strategies outperformed many others in the relief patterns track of SHREC'17 [15]. Biasotti et al. [6] introduced a covariance descriptor utilizing 2D images created from a 3D geometry. To determine how similar texture patterns are, these covariance descriptors were generated for patches. Similar to this, Giachetti et al. [18] created a 2D raster image of 3D meshes before using Improved Fisher Vector (IFV) to the 2D image to obtain feature vectors. In order to create LBP images, Tatsuma and Aono [6] first extracted depth images from 3D meshes. Additionally, the LBP image was used to compute Kaze characteristics. The final feature was created by concatenating a few statistical features that were computed on the LBP image. An EdgeLBP on the mesh surface that uses concentric spheres with various radii at each facet, similar to MeshLBP was proposed in [11]. Twelve evenly spaced points were produced for each intersecting contour. Additionally, the local binary pattern was computed using these locations.

A few mesh convolution works are directly applicable to 3D meshes. Hanocka et al. [19] introduced a technique that performed all common operations, such as convolution and pooling, on 3D meshes, much like they would be done on 2D images. This method constructed a 5D relative feature vector at each facet edge; the convolution was then operated on the neighboring edges using symmetric functions to avoid the order ambiguity between them, although the features derived with this solution used tessellation modification rather than geometric surface attributes. The method described in [9] incorporated two mesh convolutions: the first one used a 2D grid computed at each facet; and the second one performed a 2D convolution on the generated grid. A different convolution proposed in this paper employed polar coordinates and ORF, thus resulting in a mesh support which is comparable to the regular grids and Cartesian coordinates used in 2D. A drawback of this solution is related to the computational complexity in calculating geodesic distances on the surface, with a linear increment with the number of vertices. A description was developed by Sun et al. [6] using the inner dihedral angle of mesh edges. To discover commonalities across texture patches, a histogram was built using the collected angles [6].

More recently, some approaches proposed to directly work on the mesh using more versatile data structures such as graphs. For example, in [20], the authors suggested a graph learning-based method for categorizing the texture of each facet in a 3D model given the adaptability of graph representations. A 3D mesh was first converted into a graph structure, where each node represents an aspect of the original mesh. The nearby facets within a radius and their geometric characteristics were used to compute a feature vector for each facet. Then, a graph neural network is fed with the graph structure to categorize each node as belonging to the textured or non-textured class. In SHREC'20 competition [5] is reported a hybrid method, the Deep Patch Metric Learning(DPML), in which the mesh is first converted to a graph that is subsequently used to sample the surface of the mesh obtaining patches; these patches are then processed into images and used in combination with standard CNNs to classify relief patterns.

In summary, the current state-of-the-art solutions for relief patterns classification are still dominated by approaches based on techniques first developed for 2D images. Though effective results have been reported, they miss most of the additional information intrinsic to the mesh domain. Using a graph-based representation seems a natural way to take into account the adjacency information brought by the mesh support, while also opening the way to the use of learning methods that take graphs as an input. These are the motivating and funding ideas of our solution, as described in the rest of the paper.

3. Proposed approach

In the following, we describe a new relief pattern classification approach based on the definition of a hierarchical, bi-level Graph Neural Network equipped with attention layers. The overall idea is that a mesh surface can be represented as a graph, where each node encodes local geometric properties of a neighborhood of

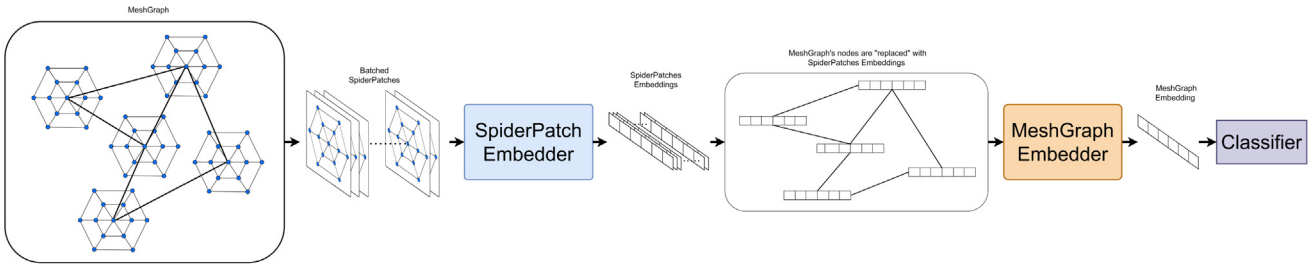

Fig. 1. The bi-level procedure of our proposed solution: the first level Graph Attention Network (SpiderPatch Embedder) analyzes all the SpiderPatches, independently of a MeshGraph, producing embeddings; these embeddings are then used as node features in the second level Graph Attention Network (MeshGraph Embedder) that analyzes the MeshGraph producing a single embedding; finally, this is used to classify the MeshGraph with a final layer.

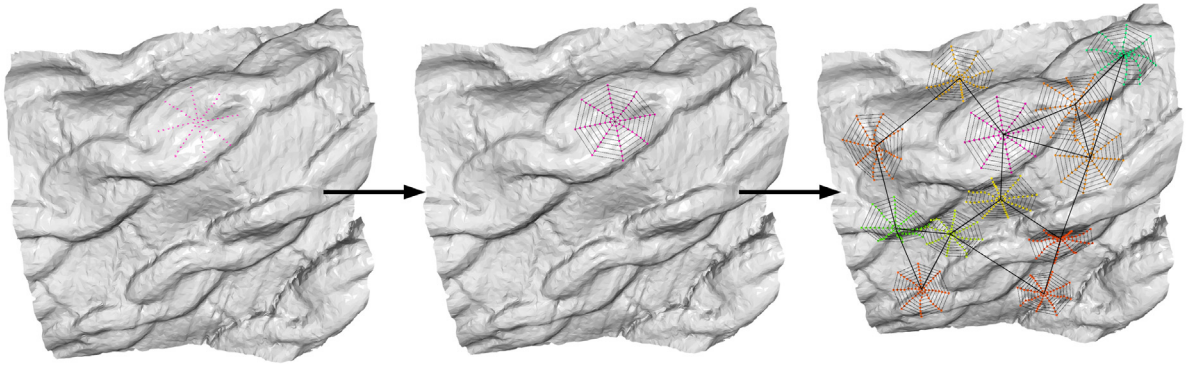

Fig. 2. Construction of a MeshGraph: first, we generate ConcentricRings (left); then, we build the SpiderPatch (middle); finally, by joining various SpiderPatches, the MeshGraph is constructed (right).

points (i.e., a local patch). However, given that relief patterns might be of different size and granularity, we define each local patch to be also a graph, where the size of the neighborhood can be defined dynamically. We will refer to the local graph as *SpiderPatch* (SP) and to the global one as *MeshGraph* (MG). The former can be considered as a local mesh descriptor as it gathers geometrical features of a local neighborhood of points, while MG is instead a global descriptor, which is built by connecting various SPs into a larger graph. To process these structures and learn relevant geometric features to classify the relief patterns, we designed two Graph Attention Networks (GATs), named *SpiderPath Embedder* and *MeshGraph Embedder*. These are intended to embed the graphs into latent features capturing discriminative characteristics to allow classifying the patterns. This framework is summarized in Fig. 1.

In order to build consistent graphs, we need a technique to sample a mesh surface as regularly as possible, so that the topology of the graphs can be defined uniquely. To this aim, we drew inspiration from the works of Tortorici et al. [9,21], in which a re-sampling process named Circle Surface Intersection Ordered Resampling (CSIOR) was used to extract local patterns on the mesh. More specifically, CSIOR is an algorithm for resampling a mesh to obtain a regular tessellation, while preserving the shape and local geometric properties of the mesh such as relief patterns and corrugations. This algorithm sorts the facets of the mesh in a polar fashion with respect to a seed point, thus maintaining a consistent ordering of points.

From the CSIOR algorithm, we borrow the Circle Surface Intersection (CSI) operation that can re-sample the mesh regularly by applying simple algebraic operations, thus creating ordered polygonal patterns on the mesh surface. In particular, to build our SpiderPatches, we developed a variation of CSI, named CSIRS (Circle Surface Intersection Regular Sampler), that locally samples the surface in a regular manner independently by the mesh tessellation through the use of CSI, thus obtaining a patch of points, denoted as *ConcentricRings*, which are ordered in a polar fashion. Based on this computation, a SpiderPatch graph is a data structure created to compensate for the lack of topology of ConcentricRings, and it is built by connecting vertices of the ConcentricRings in the shape of a spider's web. This specific choice derives from the similarity between the latter and the polygonal visual pattern that is created in the ConcentricRings (see Fig. 2 (Middle)). This new graph-based local operator has therefore opened the way to the possibility of using non-standard convolutions to process local surface patches.

Finally, a MeshGraph is built by connecting randomly sampled SpiderPatches, which represent the nodes, and therefore allows for connecting the various samplings of the mesh surface. In this way, while the SpiderPatches capture local surface features, their connection provides pattern clues to distinguish the global relief pattern. Fig. 2 summarizes the process of constructing a MeshGraph as composed of the generation of ConcentricRings and SpiderPatches.

4. Graph structures construction

In this section, we illustrate the necessary steps to build the SpiderPatch and MeshGraph structures that will be subsequently used to train the GNNs. The construction involves two macro steps: *(i)* re-sampling the mesh surface so to obtain a consistent and regular parameterization. To this aim, we defined a procedure, named *Circle-Surface Intersection Regular Sampler* (CSIRS). *(ii)* building the two above structures given the re-parameterized surface.

4.1. Surface resampling with CSIRS

The proposed CSIRS grounds on some ideas and properties of the *Circle-Surface Intersection* (CSI) algorithm defined by Tortorici et al. [21]. The goal of the CSI algorithm is to calculate a new mesh

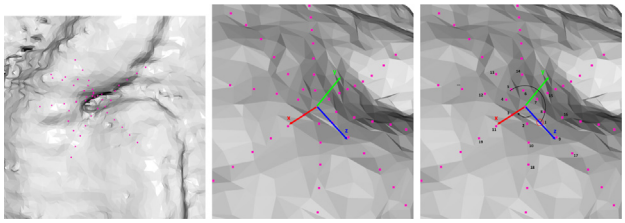

Fig. 3. LRF correction procedure: (Left) Example of an erroneous ConcentricRings generation; (Middle) LRF axes; (Right) First CSI iteration scheme and future order of ConcentricRings (Spiral Ordering).

vertex that lies on the original surface but is at a precise (non-geodesic) distance from a fixed point. The key idea is to compute the intersection between the mesh surface M with a circle C of fixed radius centered at some reference point p, orthogonal to M and aligned in the direction we want to determine the new vertex. Using the CSI properties and starting from an arbitrary mesh point, CSIRS generates a regular sequence of points that can be used as a regular sampling pattern, the *ConcentricRings*. Thanks to an iterative use of CSI, a "polygon" pattern formed by equilateral triangles can be reproduced on the mesh surface. Initially, the first polygon is generated from the seed point, i.e., the starting point of the procedure. Then, by expanding the vertices of this polygon on the mesh, ConcentricRings are generated. The result is a set of points, subdivided in concentric rings ordered following a polar fashion, that can be used as a local polar coordinate sorting operator. The characteristics of the resulting ConcentricRings depend on the following CSIRS parameters: the *radius*, i.e., the approximate geodesic distance between the seed point and a point in the last ring, the *number of rings* and the *number of points* per ring.

In order to induce a robust and consistent ordering of points on ConcentricRings when sampling on similar surfaces, the algorithm has been equipped with a Local Reference Frame (LRF), from which the directions of generation and ordering of the sampling points are chosen. More specifically, using as support points all the vertices of the mesh enclosed in a sphere with radius equal to the ConcentricRings and centered in the seed point, we calculate the eigenvectors of the Covariance matrix. The eigenvectors (x, y, z) are used as LRF axes, where x and z are parallel to the mesh surface, while y is perpendicular (conceptually similar to the surface normal). During the first CSI iteration of the procedure, the vector x is used as the normal of the first intersection circle, y is used as the rotation axis of the subsequent intersection circles and, finally, z is used to select the first point of the ordering. The direction of the ordering is instead given by the rotation around the y axis of the LRF. Therefore, being a rotation always positive and applied on a vector parallel to the surface normal, it maintains consistency on similar surfaces. Fig. 3 depicts the resulting axes.

In its basic implementation, this technique might fail in some difficult cases: for example, incorrect ordering of the LRF axes could arise if using too many (or too few) support points. To overcome these and possibly other issues, we used the actual local surface normals to align the y axis of the LRF with such normals, so tightly constraining the LRF. Fig. 4 shows some concentric rings generated in different regions of a mesh.

4.2. SpiderPatch

SpiderPatch is a data structure that was developed to compensate for the absence of a topology within ConcentricRings. It is therefore utilized for local mesh sampling, but it also has a well-defined topology, so can also be regarded as a graph

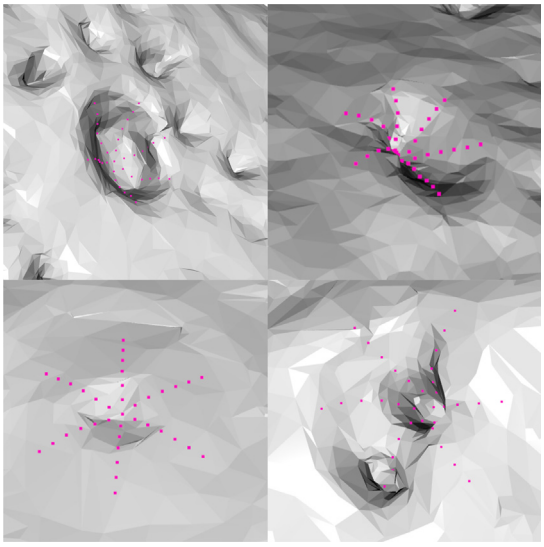

Fig. 4. Some examples of ConcentricRings generated in different regions of a mesh. It appears as ConcentricRings is capable to manage also "hard" spots if properly set.

whose nodes are connected through a spider web pattern. Every SpiderPatch has a seed point from which the generation of the graph starts: first, CSIRS is applied from the seed point, then the sample points obtained are connected following a spider web pattern. This algorithm generates the actual SpiderPatch giving to its nodes and edges the spider web pattern and some peculiar features like mesh spatial information or weights. Conceptually, the algorithm exploits the ordering of ConcentricRings to generate the SpiderPatch edges by connecting adjacent points of the same ring and the points of consecutive rings that are in the same radial direction with respect to the seed point.

A SpiderPatch is characterized primarily by three parameters: the radius, representing the maximum pseudo-geodesic distance between the seed point and a node belonging to the farthest ring from the seed point, the number of rings, and the number of nodes per ring. To clarify, the radius is used in the CSI part of the CSIRS algorithm to define the radius of the intersecting circumferences as *radius/number of rings*.

SpiderPatch nodes. The node of a graph is an abstract data structure that can virtually contain an infinite amount of information; SpiderPatch nodes inherit this trait. In addition, the nodes represent actual sampling points on the mesh that are generated thanks to the calculation of a ConcentricRings by using the CSIRS algorithm. During the SpiderPatch generation, thanks to the spider web pattern, some peculiar features are given to the nodes following a "weight decay" approach: every node is weighted based on its position in the spider web pattern relatively to the seed point (i.e., the center of the web) calculating the mean of nodes edges weights, see Section 4.2.

The following features are also stored for each node: the relative coordinates inside the web (derived from the coordinates inside the corresponding ConcentricRing); the spatial coordinates (3D), which represent the actual sampling point on the mesh; and, in addition, the Euclidean distance between node and seed point is calculated.

SpiderPatch nodes also contain Gaussian and mean curvatures, curvedness, max curvature (K2), and local depth (LD).

The fact that ConcentricRings possesses an intrinsic ordering of the generated points, and the way the SpiderPatch nodes are connected following the spider web pattern help to induce a

specific ordering of the SpiderPatch nodes: performing a visit of the SpiderPatch following the order of the node IDs (simply a number given in order of insertion of the nodes) a spiral path will be obtained, which starts from the center of the web (the seed point) and extends outwards. It is therefore possible to state that node IDs inherently have a "spiral" ordering, see Fig. 3 (Right).

SpiderPatch edges. The choice of using a connection pattern in the shape of a spider's web derives from the similarity between the latter and the visual pattern that is created in ConcentricRings. The edges that are generated can store an arbitrary number of features but, at the current stage, they only have a weight representing the type of connection, i.e., the edges that connect two nodes belonging to the same ring are considered different with respect to the edges that connect two nodes on different rings. More specifically: basic values are chosen for the two types of connection, 1 for an edge that connects two nodes belonging to different rings, and 0.85 for the edges that connect two nodes belonging to the same ring (subsequently this value is multiplied by a decay factor, which is set equal to 0.10 in the experiments). Concisely, the edge weight is computed as $E_{ij} = BW \cdot (1 - WD)^r$, where BW is the base weight defined on the edge type, WD is the weight decay, and r is the concentric ring of node i.

4.3. MeshGraph

A SpiderPatch can be regarded as a local descriptor that captures the local characteristics of relief patterns. For complex patterns classification or shape recognition tasks, SpiderPatches can be aggregated in another data structure that we called MeshGraph. Our idea here was to create a data structure that allows the exchange of information between different SpiderPatch samples on the mesh surface, while keeping intact the intrinsic spatial information of these particular graphs, thus potentially adding useful macro-information. MeshGraph is therefore a graph whose nodes are other graphs, the SpiderPatches, and edges are given a weight proportional to the Euclidean distance between the SpiderPatches seed points.

A MeshGraph can be characterized by two parameters: the number of SpiderPatches that compose it, and its degree of connectivity. It is possible to generate MeshGraphs that are completely connected or that have N-degree connectivity: simply, each SpiderPatch is connected to the closest N SpiderPatches in terms of Euclidean distance. In this way, it is possible to introduce information concerning the spatial proximity between SpiderPatches, which will be invariant on scale.

5. Network architecture

To process the SpiderPatch and MeshGraph structures, we developed a specific, bi-level architecture composed of two networks: the first, named *SpiderPatchEmbedder*, learns to embed the SpiderPatch graphs into a latent representation. The second, named *MeshGraphEmbedder*, embeds the MeshGraphs into a latent feature as well. The latter will be used to train the relief pattern classifier. Note that the MeshGraphs are built as described in Section 4.3, with the only difference that its nodes are the embedded SpiderPatches as resulting from the first networks. Both the modules have a common Graph Attention Network (GAT) [22] structure, depicted in Fig. 5. Differently from standard GCNs (Graph Convolutional Network) that use a simple average aggregation scheme, they aggregate the information from the one-hop neighborhood using an attention mechanism, weighing neighbor features with feature dependent and structure-free normalization,

The common structure of the basic building block of our architecture is composed of four main parts: the *convolutional layer*, the *normalization layer*, the *readout layer*, and the *jumping knowledge layer*. These elements are fully modular and are interspersed and repeated to compose the complete network. The two modules (SpiderPatchEmbedder and MeshGraphEmbedder) differentiate only in terms of number of the number of such block that are concatenated. Finally, note that this architecture is not specific for processing 3D meshes.

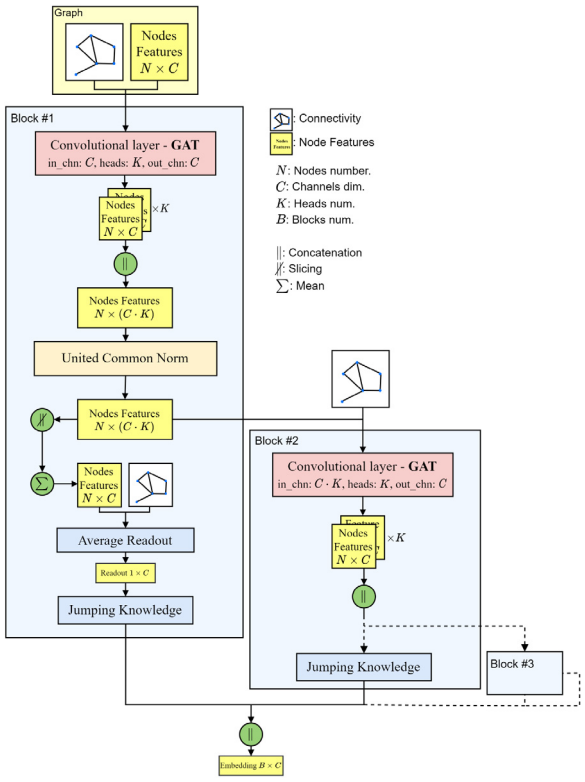

Fig. 5. Structure of the basic building block (blueish rectangles) and their interconnection. Each block is equipped with a convolutional GAT, united common normalization, average readout and jumping knowledge layers.

Convolutional layer. – It is implemented as a Multi-Head GAT [22]. In a standard Graph Convolution Network, the node embeddings $h_i^{(l+1)}$ in the layer $l+1$ are calculated as:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in N(i)} \frac{1}{c_{ij}} W^{(l)} h_j^{(l)}\right), \tag{1}$$

where $N(i)$ is the set of one-hop neighbors of i, $c_{ij} = \sqrt{|N(i)|}$ is a normalization constant based on the graph topology, σ is an activation function (often a ReLu), and $W^{(l)}$ is the shared weight matrix for node-wise transformation. The main difference in GAT layers is the introduction of an attention mechanism as a substitute for the statically normalized convolution. Practically, we substitute the term $\frac{1}{c_{ij}}$ with weights calculated as follows:

$$e_{ij}^{(l)} = LeakyReLu(\vec{a}^{(l)^T}(W^{(l)} h_i^{(l)} \parallel W^{(l)} h_j^{(l)})), \tag{2}$$

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k \in N(i)} \exp(e_{ik}^{(l)})}. \tag{3}$$

Where $\vec{a}^{(l)}$ is a learnable vector, $\parallel$ denotes concatenation, $e_{ij}^{(l)}$ represents the pair-wise not normalized attention score between two neighbors. We can finally substitute the term c_{ij} in (1) with

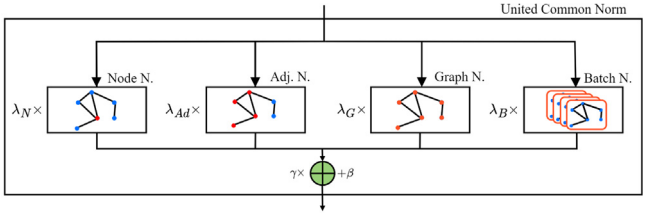

Fig. 6. United Common Normalization: different normalization coefficients are computed and jointly applied to the data as a weighted combination.

a_{ij} from (3) obtaining:

$$h_i^{(l+1)} = \sigma\left(\sum_{i\in N(i)} a_{ij} W^{(l)} h_j^{(l)}\right). \tag{4}$$

The expression in (4) represents a single-head attention layer; to obtain a multi-head attention layer, we use more independent single-head layers, each with its own parameters, and we merge their outputs. The merging operation can be performed in two ways (with K number of heads):

$$concatenation : h_i^{(l+1)} = ||_{k=1}^{K} \sigma\left(\sum_{i\in N(i)} a_{ij}^k W^{(l)k} h_j^{(l)}\right), \tag{5}$$

$$average : h_i^{(l+1)} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in N(i)} a_{ij}^k W^{(l)k} h_j^{(l)}\right), \tag{6}$$

where the concatenation is often used for intermediary layers, and the average for the final one. It should be noted that each multi-head GAT layer is also equipped with a residual connection.

Normalization layer. – Given the atypical nature of the data structures used, we decided to use a similar approach to that of [23], where a mixture of normalization methods is defined. Different normalization methods are used in a single module, and weighted by some learned coefficients. The main techniques incorporated in the mixture are node normalization, adjacent normalization, graph normalization and batch normalization (see Fig. 6).

Readout layer. – The term "readout" refers to the process of aggregating information across sets of nodes. The goal of the readout function is to summarize the information learned by the GCN into a fixed-length vector or tensor; the input to the readout function is typically the output of the final layer of the GCN, which includes node-level features for each node in the graph. There exist different readout functions. Here, we chose the *average* readout function: given a graph $G = (V, E)$, we can define the average node feature readout function as:

$$R_G = \frac{1}{|V|}\sum_{v\in V} h_v, \tag{7}$$

where h_v is the feature representation encoded in a node v.

Jumping knowledge. – The architecture is equipped with the process of Jumping Knowledge [24] that determines how the basic block output is treated. In GCNs, jumping knowledge involves connecting the output of each layer to the output of the final one, in addition to connections between adjacent layers. This creates a direct path for information to flow from early to later layers, allowing the network to use this information for making predictions. More specifically, let us consider a GCN with L layers, where the output of layer l is denoted as $h^{(l)}$. In a standard GCN, the output of the final layer is used as input to a classification layer to make predictions. In a jumping knowledge GCN instead, the output of each layer is also connected to the final layer, resulting in the following prediction function:

$$y = classify(AG(h^{(1)}, h^{(2)}, \ldots, h^{(L)})), \tag{8}$$

where AG is a generic aggregation function like mean or concatenation, but can be arbitrarily complex.

In our architecture, the output of each basic block, which is of size $K \times C^{(l)}$, being K the number of attention-heads, and C the number of channels in a convolution layer l, is both concatenated and averaged (see (5) and (6)). The concatenated output is used as input in the subsequent layer while the averaged output is used for jumping knowledge, i.e., it goes to the final layer.

5.1. SpiderPatch embedder

The SpiderPatchEmbedder is designed to embed the SpiderPatch graphs, i.e., learning a feature representation of a local patch of the mesh surface. The SpiderPatchEmbedder is composed by four concatenated basic blocks of Fig. 5, with the following parameters: 2 Heads; Average Readout; United Common Normalization; Jumping Concatenation; LeakyReLu Activation Function with slope set equal to 0.1. It is also equipped with a *NodeWeigher* module that weights the SpiderPatch nodes during readout.

5.1.1. NodeWeigher

The NodeWeigher module uses the attention mechanism to learn the best weight to apply to a specific node during the readout phase, relying both on the topological information of the node $p \in R^P$ (i.e., its relative position in the SpiderPatch), and on its features $f \in R^F$. The NodeWeigher architecture is summarized in Fig. 7.

The module is inspired by the Universal Readout function [25], yet the key difference is that the weights assigned to the nodes are not only based on current node features, but also on the topological information. As a result, the module has been designed with two fully connected layers. The first layer, denoted as *Expansion* layer, expands the topological information of the nodes (we remind that SpiderPatch nodes as defined in Section 4.2 have their own weight based on the distance from the seed point and coordinates within the SpiderPatch); the second layer, denoted as *Aggregation* layer, incorporates information relating to the node's current state, i.e., the features on which to perform the readout. The weights to use in the readout phase are finally calculated by an attention mechanism, using a Softmax activation on the output of the aforementioned fully connected layer.

5.2. MeshGraph embedder

The MeshGraph Embedder is again composed by stacking the basic blocks depicted in Fig. 5. This module though analyses graphs without specific topology (MeshGraph); therefore, it does not require any special adaptation as in the case of the SpiderPatch Embedder. The MeshGraph Embedder is built by concatenating three basic blocks. The parameters are the same as the SpiderPatchEmbedder, i.e., 2 Heads; Average Readout; United Common Normalization; Jumping Concatenation; LeakyReLu Activation Function with slope set equal to 0.2.

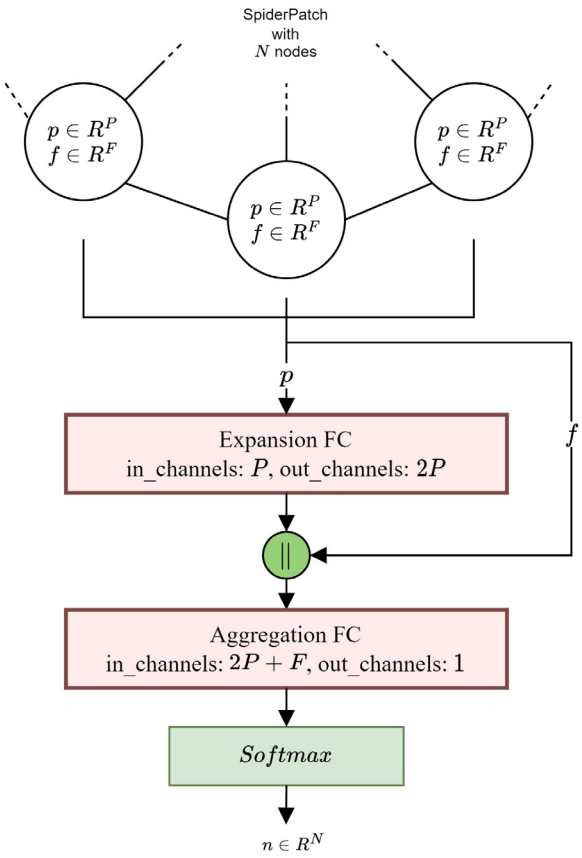

Fig. 7. NodeWeigher architecture. Every node of a SpiderPatch is analyzed: only the topological information p is used in the first FC layer.

5.3. Loss functions

To train the proposed network, we defined the CETripletMG loss that combines the classic Cross Entropy loss and a Triplet loss. The CETriplet loss is defined as follows:

$$CETripletMG(y, \hat{y}, mg_{embed}) = \alpha \cdot CrossEntropy(y, \hat{y}) + \beta \cdot Triplet(y, mg_{embed}, m), \quad (9)$$

where y are the labels, $\hat{y}$ are the predicted values, mg_{embed} are the MeshGraph embeddings computed by the MeshGraph Embedder, and $\alpha = \beta = 0.5$. $Triplet(y, mg_{embed}, m)$ is a triplet loss [26] adapted to MeshGraphs that uses the online triplet mining strategy to calculate the hardest-negative and hardest-positive examples in the batch of normalized embedded MeshGraphs passed to the function. The margin m is set to 1.

6. Experimental results

In this section, we report relief pattern classification results on the SHREC'17 and SHREC'20 competition datasets. An ablation study was also conducted on SHREC'17 to find the best parameters, and investigate how they impact on the results.

6.1. SHREC'17

In this section, we first present the data used in the experiments on the SHREC'17 dataset and their pre-processing, respectively, in Sections 6.1.1 and 6.1.2. Then, we report results for our method also in comparison with state-of-the-art solutions in Section 6.1.3. Additional analysis for different configurations of our methods are given in Section 6.2.

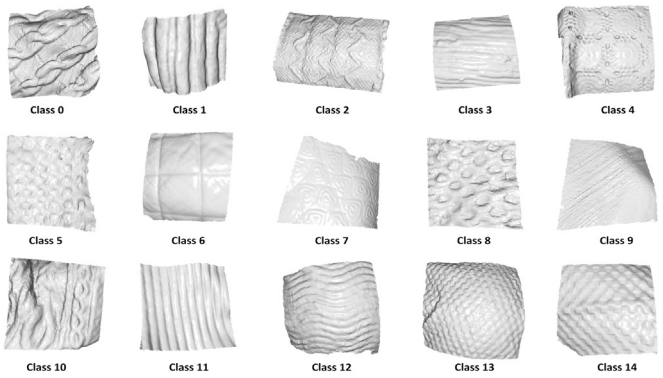

Fig. 8. Example meshes from the 15 classes of the SHREC'17 dataset.

Table 1
Characteristics of the SpiderPatch datasets.

SpiderPatch datasets				
Name	Radius	Rings	Points	Norm & Align
SHREC17R0.1RI4P6	0.1	4	6	✓
SHREC17R0.1RI6P8	0.1	6	8	✓
SHREC17R10RI4P6	10	4	6	✗
SHREC17R10RI6P8	10	6	8	✗
SHREC20R0.1RI6P8	10	6	8	✗
SHREC20R0.15RI6P8	10	6	8	✗

6.1.1. Raw data

The raw mesh data used in our experimentation are those included in the SHREC'17 track on *Retrieval of Surfaces with similar Relief Patterns* [27]. The (*full*) dataset consists of 720 mesh surfaces, grouped in 15 classes of 48 elements each (see Fig. 8 for some examples). Each class represents a single pattern and was created by acquiring it in different poses, and applying three processing operations to each scan (two adaptive simplifications to 10K and 5K vertices and one re-sampling operation to 15K vertices). Transformations were designed to alter the connectivity of the original meshes. Since data were acquired with a depth sensor, small artifacts and small topological handles are also present. The dataset constituted by the 180 unique original raw scans (15 patterns, each with 12 samples, all different from each other) is referred to as *original dataset*.

In the literature, the *full dataset* of 720 meshes was used in a relief pattern retrieval task, with a reported accuracy that reached 100% [9] or very high scores [11,18]. The **original dataset** proved to be more challenging, as evidenced by the results reported in the literature (see Table 3), so we opted to utilize the original dataset for our experiments.

6.1.2. Raw data sampling with SpiderPatches and MeshGraphs

In order to use the SHREC data to train our model, we need to process them so to build the SpiderPatch and MeshGraph data structures. To this end, we derived different *SpiderPatch datasets* and *MeshGraphs dataset* from the data of the SHREC'17 *original dataset*. Different datasets are generated by varying the parameters of the SpiderPatches, e.g., radius, number of rings, and MeshGraphs, e.g., connectivity degree.

SpiderPatch dataset – It is a collection of SpiderPatches generated by applying the CSIRS procedure to the SHREC data. The dataset is constituted by 1,000 SpiderPatches, generated for each scan in SHREC'17 using a uniform sampling. The dataset contains only SpiderPatches that have at least the first two rings completed, i.e., rings in which all the sampling points have been generated and which are therefore closed, In this way, the relief

Table 2
Characteristics of the MeshGraph datasets. With the parameter Norm&Align we refer to a MeshGraph dataset generated starting from a set of meshes that have been normalized and aligned (i.e., meshes have unitary cube bounding boxes and are aligned along their major extension dimension.

SHREC'17 MeshGraph datasets							
Name	SpiderPatch (SP) params				MeshGraph (MG) params		
	Radius	Rings	Pts	Norm&Align	MG Num.	#SP	Connec.
A	0.1	4	6	✓	25	50	5
B	0.1	6	8	✓	25	50	5
C	10	4	6	✗	25	50	5
D	10	6	8	✗	25	50	10
SHREC'20 MeshGraph datasets							
E	0.1	6	8	✗	25	50	10
F	0.15	6	8	✗	25	50	10

Table 3
Comparison of our method with the results of SHREC'17 competition [27], EdgeLBP [11], Mesh-LBP and Mesh-Convolution [9], and the approaches of [18]. Performance refers to the original dataset using Nearest Neighbor (NN) classification for No deep learning methods.

NO Deep Learning		
Retrieval Method		NN
[27]	CMC-2	63.3%
	KLBO-FV-IWKS	52.2%
	KLBO-SV-IWKS	48.9%
[11]	EdgeLBP - run1	92.2%
	EdgeLBP - run2	91.1%
[18]	P/mC/SIFT/FV	82.8%
	T/mC/SIFT/FV	87.2%
	MeshLDSift+FV	77.8%
[9]	Differential r = 4	87.22%
	Edge Detector (b)	90.00%
	Sobel	**93.33%**
	Haar H_2	90.00%
	Sharpen H_2	91.11%
DEEP LEARNING		
Classification Method		Acc.
No Voting	Spider-GAT	95.0%
Voting	Spider-GAT	**97.2%**

pattern is sampled by possibly different types of SpiderPatches. A SpiderPatch dataset can be described by the characteristics of its SpiderPatches: in particular, we generated 4 datasets, 2 using normalized and aligned SHREC'17 meshes, and 2 where the original SHREC'17 data were left unchanged, as reported in Table 1.

MeshGraphs dataset – It is a collection of MeshGraphs generated from the SpiderPatch datasets. For every mesh in SHREC'17, we generated a certain number of MeshGraphs using homogeneous SpiderPatches from a SpiderPatch dataset. A MeshGraph dataset is mainly characterized by: number of MeshGraph per SHREC'17 mesh; number of SpiderPatches per MG (MG nodes); type of SpiderPatch; MG connectivity. The main characteristics of the datasets are summarized in Table 2.

6.1.3. Relief pattern classification

As previously mentioned, the results on the full dataset have perfect or extremely high accuracy in the literature; however, we notice that all the SHREC'17 methods drastically decreased their performance when applied to the original dataset, except for the KLBOFV-IWKS method that achieved a maximum score of 63.3% (CMC-2) [27]. The feature mapping method of [18], on the other hand, performed better than the SHREC'17 participants, earning 87.2% on its Tutte/meanC/SIFT/FV configuration. This strategy performed worse than the EdgeLBP of Thompson et al. [11] and the Mesh-LBP+Mesh-Convolution approach of Tortorici et al. [9].

The mapping method used in such a solution, which maps to the 2D domain and thus loses some information on the mesh manifold, may be one reason for this. Instead, the EdgeLBP suggested by the competition's organizers demonstrated its potential by obtaining 92.2% and 91.1% in its two runs. The convolution-based method proposed by Tortorici et al. [9] also reported a performance decline on the original dataset, which is consistent with the rest of the literature. However, in their work discrete filters showed more resilience, and they deemed Sobel to be the best feature descriptor because it outperformed the state-of-the-art for this dataset, achieving 93.33% in classification accuracy.

The Mesh-LBP was computed with radius $r = 10$. This is somewhat in accordance with our method that scored its best results when a SpiderPatch radius of 10 was used. In contrast, the MeshLBP approach compared two meshes by collecting a global histogram on all the mesh facets, then comparing these histograms with the Bhattacharyya distance. Differently, in our case, the sample points on the mesh that were needed to build the MeshGraph are just a small fraction compared to the number of facet of the mesh.

In Table 3, we summarize the techniques described above for relief pattern classification on the SHREC'17 original dataset. The table is divided into two main parts: the first part summarizes the state-of-the-art methods that do not use deep learning, while the second part reports the results of our method that applies deep learning on graphs. The first part of the table is also divided into four groups, thus evidencing some methods that participated in the SHREC'17 contest [27], the two method variants of [11], the method variants of [18], and some discrete convolution filters proposed in [9]. Our network was trained on the MeshGraphs included in the "D" dataset of Table 2. This dataset was chosen thanks to the execution of small-scale tests and seems to be the one containing the type of MeshGraphs that best characterize the SHREC17's relief patterns.

To evaluate the performance of the new feature descriptors and network architecture, we followed two approaches: first, we measured the ability of the model to classify individual MeshGraphs, i.e., considering a MeshGraph as a unique and separate representation of a relief pattern; in a second test, we considered the MeshGraphs generated from a mesh as samples of the mesh itself, while the mesh classification was obtained by a majority voting of the individual MeshGraph predictions.

We train the network for a total of 60 epochs, adopting an early-stopping paradigm. In order to better generalize the results, and compare with those present in the literature, and with those obtained in the SHREC17 competition in particular, we used a 12-fold cross-validation (i.e., for each class in the dataset there are 12 distinct samples). In this way, we obtained 12 trained networks in which, for each class, 11 samples were used as training and 1 sample was used as validation.

For the network training, we used the AdamW optimizer, which is a variant of the classic Adam optimizer that uses a decoupling weight decay technique in the gradient update, with an initial learning rate equal to 0.01. The batch size was very dependent on the implementation of the used batching techniques: a batch size of 128 was chosen for the batching of the MeshGraphs, and 512 for the batching of the SpiderPatches. A high value for MeshGraphs batching resulted necessary for an optimal behavior of the proposed CETriplet loss.

6.2. Discussion

We note that the results reported in Table 3 for the SHREC17 track and the other retrieval methods were obtained with the specific protocol defined in the competition. The different setup, *i.e.,* classification instead of retrieval, prevented us from adopting exactly the same protocol as in SHREC17. The latter indeed

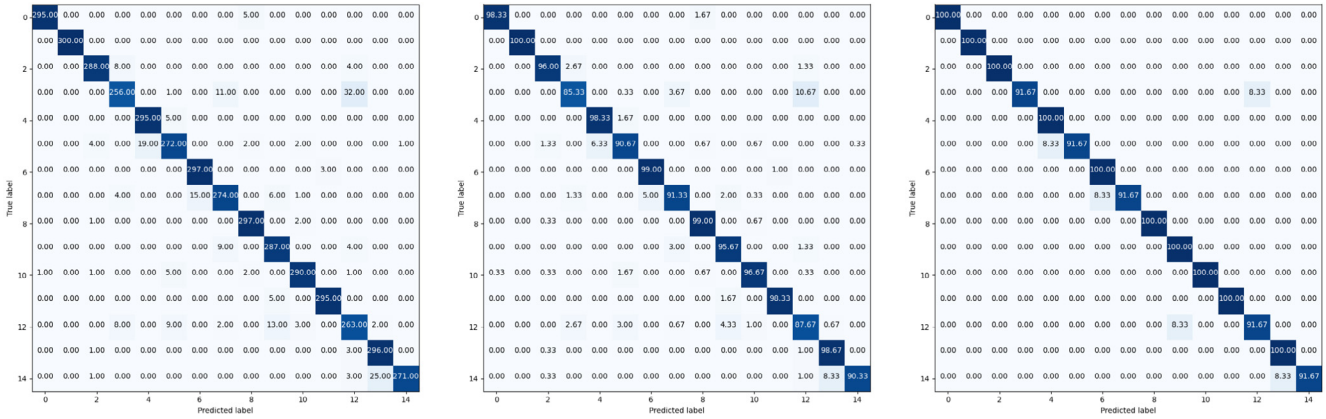

Fig. 9. Confusion matrices for the experiments conducted on SHREC'17 using a 12-fold cross validation (11 meshes were used for training, 1 mesh for validation, repeated 12 times so that each mesh is used once for test). (Left) Confusion matrix with the number of correctly classified MeshGraphs for each class (for each validation mesh, 25 MeshGraphs were generated; thus, for the entire cross validation process, there were $25 \cdot 12 = 300$ MeshGraphs to be classified for each class). (Middle) Mean accuracy confusion matrix. (Right) Voting confusion matrix obtained by implementing a voting approach on MeshGraphs: a mesh is correctly classified if at least half of the MeshGraphs representing it are correctly classified.

does not come with a predefined train/test split, necessary for training the classifier. Methods performance is evaluated using the Nearest-Neighbor criterion for the "Retrieval" task, i.e., given a mesh (Query) of class find, among all the other meshes in the dataset (Gallery), the most similar one based on some similarity metric. We argue this is the most comparable setting with respect to a classification one where, for a given mesh, the model directly outputs its predicted class. Clearly, this requires dividing the dataset in two sets, one to train the model, and the other for testing. In order to collect results for all the samples, we performed a cross-validation as per the standard procedure. So, we performed a 12-fold cross-validation, and gathered results obtained for all the folds, so to cover the entire dataset.

The performance of the MeshGraph classification is reported in Table 3. For this particular task, we reached an accuracy of 95.0% outperforming existing methods. In Fig. 9, we reported the corresponding confusion matrices, reaching a perfect classification score for class 1. Classes that obtained a lower score, like class 3 or 12 (above 85%) are similar, and their relief patterns would need different MeshGraph configurations to be better classified. Further, these two classes include quite irregular meshes, where the relief pattern is barely visible.

As discussed above, to balance the effect that the random generation of the MeshGraphs might have on the relief pattern classification results based on the individual MeshGraphs, and therefore not create a bias on the prediction performance measures, we implemented a voting mechanism. Using this voting approach, a mesh is assigned to a certain relief pattern class by applying a majority vote on the classification of the MeshGraphs generated by the mesh itself. With this configuration, the accuracy increased to 97.2%, and the classes with a perfect score increased to 10 out of 15. This is shown in Fig. 9, where the 5 classes that do not have a perfect score still show an accuracy of 91.7%, meaning that only one out of the 12 meshes was misclassified during the cross validation.

6.3. Ablation study

For completeness, we report in Table 4 an ablation study to evaluate how the hyperparameters related to the generation of the MeshGraph datasets influenced the performance. The parameters of the network architecture are instead fixed. Due to computational limitations, a cross-validation was not used in this case. Instead, an 80/20 splitting was used: the 80% of the meshes in the original dataset on which the MeshGraphs were generated, were used for training, and 20% for test (i.e., 10 training meshes and 2 test meshes for each of the 15 classes of SHREC17). It is interesting to observe that, keeping unchanged the other hyperparameters, the connectivity of the MeshGraph plays a fairly important role in decreeing good or bad performance; balanced values allow the achievement of an excellent accuracy. Another important parameter is the radius chosen for building the SpiderPatches. Below a certain threshold, it appears the performance remains unchanged (for example, in the case of radius 6 and 10); instead, higher values capturing an excessively large area, lead to the risk of missing the relief pattern. Ultimately, the choice of the radius is critical yet reasonably robust; a wrong choice can still compromise the effectiveness with which some patterns are recognized nonetheless.

Table 4
Test on different SHREC'17 MeshGraph datasets.

SpiderP. params				MeshGraph params			Statistics	
Radius	Rings	Points	N&A	# MG	# SP	Connec.	Acc.	Loss
0.1	4	6	✓	25	50	5	91.3%	0.52
0.1	6	8	✓	25	50	5	90.7%	0.47
0.1	6	8	✓	25	50	10	93.6%	0.35
6	4	6	✗	25	50	5	93.0%	0.36
10	4	6	✗	25	50	5	92.9%	0.44
10	6	8	✗	25	50	5	93.9%	0.43
10	6	8	✗	25	50	10	96.7%	0.26
10	6	8	✗	50	10	full	84.0%	0.88
10	6	8	✗	50	25	full	88.9%	0.56
15	4	6	✗	25	50	5	96.8%	0.31

6.4. SHREC'20 experiments

In this section, we present the results of the experiments performed on the SHREC'20 dataset [5]. First, we summarize the peculiarities of the data used and their pre-processing in Section 6.4.1. Then, we report results for our method utilizing the best configuration derived for the SHREC'17 dataset in Section 6.4.2, and provide further discussion in Section 6.4.3.

6.4.1. Raw data

To better evaluate our method, we applied it to the complex synthetic dataset released for the SHREC 2020 challenge

Table 5
Comparison of our method with the results of SHREC20 competition [5]. Retrieval performance of methods that participated in the SHREC20 competition were measured using Nearest Neighbor (NN), which is comparable with our classification accuracy measure:.

NO Deep Learning		
Retrieval Method		NN
[5]	MeshLBP-So	90.9%
	kd-tree FLANN	68.6%
	SRNA	92.3%
	OH	71.4%
	APPFD-FK	18.6%
DEEP LEARNING		
Classification Method		NN.
[5]	DPML	98.2%
	PointNet+SQFD	17.3%
	DFE	98.2%
		Acc.
No Voting	Spider-GAT	90.7%
Voting	Spider-GAT	**100**%

on *Retrieval of digital surfaces with similar geometric reliefs* [5]. The SHREC 2020 dataset includes 220 triangular meshes of 20 synthetically modeled objects like pots, goblets and mugs, each one of them characterized by one of 11 different geometric reliefs derived from natural textures (bricks, floors, roofs, etc.). Each mesh is given by a single connected component, it is regular, and made of 50K vertices. Every different object is covered with all 11 different relief patterns and, depending on the object represented, some meshes have non-trivial topology (they may have handles or tunnels) and may even have boundaries.

Similar to the experiments on SHREC17, our aim here is to classify relief patterns using the proposed SpiderPatch and MeshGraphs structures. To this end we pre-processed the data to derive SpiderPatch and MeshGraph data following the same procedure as explained in Section 6.1.2. This resulted into 1,000 SpiderPatches for every mesh in SHREC'20, from which we generated the MeshGraph dataset. To compensate for the differences in mesh dimension between SHREC17 (bounding box volume around 3 m^3) and SHREC'20 (bounding box volume around 4.5 cm^3), and better compare the results, to generate our data structures from SHREC20, we used the same configuration used for SHREC17, only changing the SpiderPatch radius, as reported in Tables 1 and 2.

6.4.2. Relief pattern classification

A total of eight methods participated in the SHREC20 challenge [5]. They can be divided in two groups: geometric-based and learning-based methods. For the sake of brevity, we will provide a detailed comparison only with learning-based methods. Among the geometric-based approaches, the best performance was obtained by Mesh-LBP and SRNA with a score of 90.9% and 92.3%, respectively. The Mesh-LBP method is similar to the one used in the SHREC17 competition, with the difference that several histograms are computed rather than a global one. To compare two meshes, the respective patches are therefore compared. SRNA also uses histograms, but calculated on the basis of per-vertex quantity. Both methods require pre-processing on the raw data (smoothing for SRNA and resampling for Mesh-LBP) and use a good fraction, if not all, of the vertices/faces of the mesh.

For learning-based methods, the best ones turned out to be DFE and DPML, both with an accuracy of 98.2%. DFE extracts 2D images starting from surface patches of the mesh and uses them as input to different pre-trained CNNs (ResNet [28], DenseNet [29], etc.). DPML, instead, uses graphs to represent the mesh. Differently from our solution, DPML represents the entire mesh as a graph and uses it for the calculation of surface patches. These patches are then transformed into 2D images and used for training a Siamese CNN (a VGG16 architecture [30]). In Table 5, we summarize results on SHREC'20 for the above techniques. The table explicitly separates learning-based methods (including ours) from geometric ones.

For training our network, we followed the steps described in Section 6.1.3, with the obvious exception of a different choice of MeshGraph dataset and a different cross-validation protocol. In this case, in fact, the dataset "E" of Table 2 was used, which we recall was generated with the specifications of the best MeshGraph dataset for SHREC17 ("D" Table 2) and a 5-fold cross-validation was used as a training protocol, which translates into 16 meshes used as training dataset and 4 as validation. The rest of the hyperparameters have been set as described in the last part of Section 6.1.3.

6.4.3. Discussion and additional analysis

The results of the experiments based on SHREC20 are reported in Table 5. Differently from SHREC'17, in SHREC'20 [5] classification results are instead reported, yet only in form of graphs without numerical values. Given this, we could not directly compare with those results; so, we resorted to using the performance measurement protocol described in Section 6.2, and compare with the same measure as that used for SHREC'17 *i.e.* nearest-neighbor retrieval.

The performance of the classification of the individual MeshGraphs is shown in Table 5 and stands at 90.7%, marking a worsening compared to 95% of SHREC17; this is reasonably due to the more complex shape of the meshes, as well as to the random sampling during the generation of the SpiderPatches. Similar to SHREC17, we can observe from the confusion matrices in Fig. 10 (Center) that there is a predominance of some classes (1 and 7) over others, indicating that a different MeshGraph configuration may favor some classes. It is also interesting to note that SHREC20 classes 1 and 7 (Fig. 8 represent similar patterns, respectively, to SHREC17 classes 1 and 8 [5]. In Table 5, we also report the classification accuracy using the voting mechanism: with this configuration we get a perfect score, 100%. Such a result is due to the voting mechanism, canceling the effect that an unfortunate placement of the SpiderPatches has on the classification of the single MeshGraphs.

7. Conclusions

In this paper, we presented new mesh surface descriptors, namely SpiderPatch and MeshGraph that used together with our bi-level GAT, shown to outperform existing SHREC'17 relief pattern classification methods obtaining an accuracy of 97.2% (i.e., misclassifying just 5 meshes over a total of 180 meshes).

Considering the current definition and practical implementation of the SpiderPatch structure, we can enumerate pros and cons as emerged from our reported results and further experimental analysis that we did not report in the paper. As pros, SpiderPatches can be computed via simple geometric computations; they are scale-invariant with respect to the mesh (but they are not with respect to the scale of the relief pattern), they are not influenced by the mesh tessellation; they are invariant to rotations and can be applied to any mesh without preprocessing. As cons, the local patches are affected by the scale of the relief patterns.

The MeshGraph representation revealed both merits and limitations: on the positive side, it can be computed on a quite ample spectra of meshes, thus being adaptable to different tasks; on the negative one, we did not exploit yet a generation with a predefined sampling as well as the edges do not account for geodesic distance.

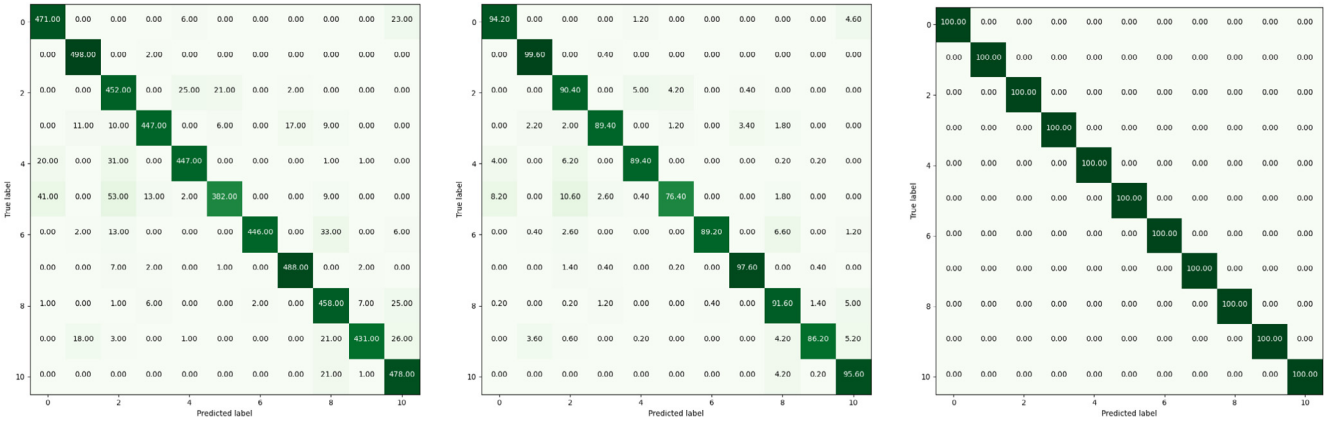

Fig. 10. Confusion matrices for the experiments performed on SHREC'20 using a 5-fold cross validation. (Left) Confusion matrix with the number of correctly classified MeshGraphs for each class (16 training meshes, 4 validation meshes, for each validation mesh 25 MeshGraphs were generated so that $25 \cdot 4 \cdot 5 = 500$ MeshGraphs were classified in the entire cross validation process). (Middle) Mean accuracy confusion matrix. (Right) Voting confusion matrix obtained by implementing a voting approach on the MeshGraphs: a mesh is correctly classified if at least half of the MeshGraphs representing it are correctly classified.

The proposed GAT architecture has some positive properties: though we experimented it in the task of relief pattern classification, in principle it can be applied to different tasks (e.g., 3D shape retrieval); it is a completely modular architecture, which makes it easily adaptable and open to improvements; it allowed us to achieve effective performance that outperformed state-of-the-art methods using a comparable protocol with a relatively small number of training samples.

Overall, based on the analyses performed so far, we have built some very ductile and adaptable descriptors as well as a network that can be easily remodeled for other tasks. As future work, we plan to explore new fields of application for the proposed approach.

CRediT authorship contribution statement

Niccolò Guiducci: Software, Validation, Investigation, Writing – original draft, Data curation. **Claudio Tortorici:** Investigation, Writing – review & editing, Formal analysis, Conceptualization. **Claudio Ferrari:** Methodology, Writing – review & editing, Visualization. **Stefano Berretti:** Supervision, Writing – review & editing, Visualization.

Declaration of competing interest

We declare that the content of the paper is the result of the research effort of all the authors listed in the accompanying letter. All authors have approved the paper submission.

Data availability

Data will be made available on request

Acknowledgments

The research is funded by the Technology Innovation Institute (TII), Abu Dhabi, UAE under grant TII/ARRC/2053/2021.

References

[1] Andreetto M, Brusco N, Cortelazzo GM. Automatic 3D modeling of textured cultural heritage objects. IEEE Trans Image Process 2004;13(3):354–69.
[2] Hu S, Li Z, Wang S, Ai M, Hu Q. A texture selection approach for cultural artifact 3D reconstruction considering both geometry and radiation quality. Remote Sens 2020;12(16):2521.
[3] Gillespie D, Welham K. Subjective and objective assessment of 3D textured and non-textured cultural heritage artefacts. IEEE Comput Graph Appl 2020.
[4] Othmani A, Voon LFLY, Stolz C, Piboule A. Single tree species classification from terrestrial laser scanning data for forest inventory. Pattern Recognit Lett 2013;34(16):2144–50.
[5] Thompson EM, Biasotti S, Giachetti A, Tortorici C, Werghi N, Obeid AS, Berretti S, Nguyen-Dinh H-P, Le M-Q, Nguyen H-D, et al. SHREC 2020: Retrieval of digital surfaces with similar geometric reliefs. Comput Graph 2020;91:199–218.
[6] Biasotti S, Moscoso Thompson E, Aono M, Ben Hamza A, Bustos B, Dong S, Du B, Fehri A, Li H, Limberger FA, Masoumi M, Rezaei M, Sipiran I, Sun L, Tatsuma A, Velasco-Forero S, Wilson RC, Wu Y, Zhang Y, Zhao T, Formasa F, Giachetti A. Shrec'17 Track: Retrieval of surfaces with similar relief patterns. In: 10th Eurographics workshop on 3D object retrieval. 2017, p. 95–103.
[7] Werghi N, Tortorici C, Berretti S, del Bimbo A. Local binary patterns on triangular meshes: Concept and applications. Comput Vis Image Underst 2015;139:161–77. http://dx.doi.org/10.1016/j.cviu.2015.03.016, URL https://www.sciencedirect.com/science/article/pii/S1077314215000843.
[8] Werghi N, Tortorici C, Berretti S, Del Bimbo A. Representing 3D texture on mesh manifolds for retrieval and recognition applications. In: Proceedings of the IEEE Conference on computer vision and pattern recognition. 2015, p. 2521–30.
[9] Tortorici C, Berretti S, Obeid A, Werghi N. Convolution operations for relief-pattern retrieval, segmentation and classification on mesh manifolds. Pattern Recognit Lett 2021;142:32–8.
[10] Tortorici C, Werghi N, Berretti S. Representing and analyzing relief patterns using LBP variants on mesh manifold. Pattern Anal Appl 2021;24:557–73.
[11] Thompson EM, Biasotti S. Description and retrieval of geometric patterns on surface meshes using an edge-based LBP approach. Pattern Recognit 2018;82:1–15.
[12] Dalal N, Triggs B. Histograms of oriented gradients for human detection. In: IEEE Conference on computer vision and pattern recognition, vol. 1. 2005, p. 886–93. http://dx.doi.org/10.1109/CVPR.2005.177.
[13] Lowe DG. Distinctive image features from scale-invariant keypoints. Int J Comput Vis 2004;60:91–110.
[14] Ojala T, Pietikainen M, Harwood D. Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In: Proceedings of 12th International conference on pattern recognition, vol. 1. IEEE; 1994, p. 582–5.
[15] Biasotti S, Thompson EM, Barthe L, Berretti S, Giachetti A, Lejemble T, Mellado N, Moustakas K, Manolas I, Dimou D, et al. Shrec'18 track: Recognition of geometric patterns over 3D models. In: Eurographics workshop on 3D object retrieval. 2018.
[16] Zaharescu A, Boyer E, Varanasi K, Horaud R. Surface feature detection and description with applications to mesh matching. In: 2009 IEEE Conference on computer vision and pattern recognition. IEEE; 2009, p. 373–80.
[17] Werghi N, Berretti S, Del Bimbo A. The mesh-lbp: a framework for extracting local binary patterns from discrete manifolds. IEEE Trans Image Process 2014;24(1):220–35.
[18] Giachetti A. Effective characterization of relief patterns. In: Comput Graph Forum. 37, (5):Wiley Online Library; 2018, p. 83–92.

[19] Hanocka R, Hertz A, Fish N, Giryes R, Fleishman S, Cohen-Or D. Meshcnn: a network with an edge. ACM Trans Graph 2019;38(4):1–12.
[20] Ganapathi II, Javed S, Fisher RB, Werghi N. Graph based texture pattern classification. In: Int. conf. on virtual reality. ICVR, 2022, p. 363–9. http://dx.doi.org/10.1109/ICVR55215.2022.9847889.
[21] Tortorici C, Riahi MK, Berretti S, Werghi N. CSIOR: Circle-surface intersection ordered resampling. Comput Aided Geom Design 2020;79:101837.
[22] Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y. Graph attention networks. 2017, arXiv preprint arXiv:1710.10903.
[23] Chen Y, Tang X, Qi X, Li C-G, Xiao R. Learning graph normalization for graph neural networks. Neurocomputing 2022;493:613–25.
[24] Xu K, Hu W, Leskovec J, Jegelka S. How powerful are graph neural networks? 2019, arXiv:1810.00826.
[25] Navarin N, Van Tran D, Sperduti A. Universal readout for graph convolutional neural networks. In: 2019 International joint conference on neural networks. IJCNN, IEEE; 2019, p. 1–7.
[26] Schroff F, Kalenichenko D, Philbin J. Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE Conference on computer vision and pattern recognition. 2015, p. 815–23.
[27] Biasotti S, Thompson EM, Aono M, Hamza AB, Bustos B, Dong S, Du B, Fehri A, Li H, Limberger FA, et al. Shrec'17 track: Retrieval of surfaces with similar relief patterns. In: 10th Eurographics workshop on 3F object retrieval. 2017.
[28] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on computer vision and pattern recognition. 2016, p. 770–8.
[29] Huang G, Liu Z, Pleiss G, Van Der Maaten L, Weinberger KQ. Convolutional networks with dense connectivity. IEEE Trans Pattern Anal Mach Intell 2019;44(12):8704–16.
[30] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In: International conference on learning representations. 2015, p. 1–14.