




A sparse mesh sampling scheme for graph-based relief pattern classification

G. Paolini¹ , N. Guiducci¹, C. Tortorici² , S. Berretti^{†1} 

¹Media Integration and Communication Center, University of Florence, Florence, Italy

²Technology Innovation Institute, Abu Dhabi, United Arab Emirates

Abstract

In the context of geometric deep learning, the classification of relief patterns involves recognizing the surface characteristics of a 3D object, regardless of its global shape. State-of-the-art methods leverage powerful 2D deep learning image-based techniques by converting local patches of the surface into a texture image. However, their effectiveness is guaranteed only when the mesh is simple enough to allow this projection onto a 2D subspace. Therefore, developing deep learning techniques that can work directly on manifolds represents an interesting line of research for addressing these challenges. The objective of our paper is to extend and enhance the architecture described in a recent GNN approach for a relief pattern classifier through the introduction of a new sampling technique for meshes. In their method, local mesh structures, referred to as SpiderPatches, are connected to form the nodes of a graph, called MeshGraph, that captures global structures of the mesh. These two data structures are then fed into a bi-level architecture based on Graph Attention Networks. The MeshGraph construction proves important in ensuring optimal classification results. By the proposed subsampling process, we tackle the problem of fine-tuning multiple hyperparameters inherent the MeshGraph by defining a graph structure that is aware of the mesh geometric details. We demonstrate that the graph constructed using this approach robustly captures the relief patterns on the surface, obviating the need for data augmentation during training. The resulting network is robust, easily customizable, and shows comparable performance to recent methods, all while operating directly on 3D data.

CCS Concepts

• **Computing methodologies** → **Neural networks; Object identification;**

1. Introduction

The problem of relief pattern classification concerns the analysis and recognition of the geometric properties over 3D surface. Similarly to the texture in a 2D image, the features of these patterns are independent to the overall shape of the manifold on which they rest. Examples of such patterns can be found in engravings, embroidered fabrics, and more broadly in regular decorations of ancient artifacts and everyday objects [MBG*20; OVSP13].

In recent years, the problem of classification, retrieval, and detection of relief patterns has gathered increasing attention, especially due to the advancement of algorithms in computer vision and the availability of datasets containing hundreds of both synthetic samples and objects derived from 3D scans [MBG*20; BTA*17]. However, most state-of-the-art methods avoid working directly on 3D

structures. Instead, they perform multi-view analysis techniques, where parts of the mesh are projected onto a 2D plane for further processing, often via learning-based approaches [MBG*20] that revealed to be effective in the 2D field. Despite these category of approaches achieved significant results, they still rely on 2D representation of 3D data, thus losing the essence of 3D information. In the recent years, the concepts like convolution or pooling have been extended to the 3D domain, in particular on mesh manifolds, point clouds, and graphs [BBL*17; HHH*19; MLR*20; SACO22; HLG*22; YLB*20]. Despite this progress, there has not been a corresponding advancement in the classification of relief patterns. Starting from the work of Werghe et al. [WBD14] based on a definition of Local Binary Patterns on mesh manifolds, many approaches have been studied that rely on ad-hoc descriptors [WTBD15; TBOW21; TWB21; TB18].

In Guiducci et al. [GTFB23], a relief pattern classifier based on Graph Neural Networks (GNN) was introduced. Their network features a bi-level architecture operating on both local and global graph structures, called SpiderPatch and MeshGraph, respectively. The idea is to use an individual SpiderPatch to capture a local and detailed representation of the mesh, while the MeshGraph, con-

[†] Corresponding author.

E-mail addresses: gabriele.paolinil@unifi.it (G. Paolini)

niccolo.guiducci@edu.unifi.it (N. Guiducci)

claudio.tortorici@tii.ae (C. Tortorici)

stefano.berretti@unifi.it (S. Berretti)

structured using feature vectors from several SpiderPatches as nodes, provides a macroscopic view of the mesh. The advantage of this approach is the ability to operate entirely on manifolds, achieving results in SHREC'17 and SHREC'20 tracks that are comparable, if not superior, to image-based methods.

In this paper, we develop on the approach proposed in Guiducci et al. [GTFB23] with the aim of making it more general and less demanding in terms of number of hyperparameters to tune. In particular, we have designed a mesh sampling algorithm capable of uniquely generating (except for the choice of the starting point) a global MeshGraph that retains sufficient information to characterize the relief patterns of the mesh. This way, we construct a more stable graph structure by halving the number of hyperparameters to fine-tune. Specifically, this sampling algorithm determines both the number of nodes (i.e., the SpiderPatches) and connectivity. Fine-tuning of this latter parameter is crucial to achieve high accuracy, so the development of an automatic method to estimate connectivity between SpiderPatches is essential. Finally, thanks to the MeshGraph constructed in this way, we are able to obtain accuracies that are close to the state-of-the-art without relying on data augmentation or voting systems.

We assessed the efficacy of the proposed method for relief pattern classification on meshes from the SHREC'17 relief pattern track dataset [BTA*17], wherein each mesh exhibits a single relief pattern. We showcased the potential and robustness of graph-based learning algorithms in this problem domain, achieving outcomes that are comparable with those reported in the foundational work of this paper.

Our main focus is directed towards the geometric texture classification task, thus we structured the rest of the paper as follows: In Section 2, we summarize related work in the literature of relief pattern classification; Section 3 describes in detail the proposed approach and the network architecture; experimental results are discussed in Section 4; finally, in Section 5, we draw conclusions and discuss potential future developments.

2. Related work

Many techniques have been proposed to describe texture patterns in the 2D image domain based on their repeatability, unpredictability, and orientation [DT05; Low04]. Widely recognized methods to extract such information typically rely on local descriptors resulting from convolution-based filtering operations or Local Binary Patterns (LBP) [OPH94]. While texture analysis is well-developed for 2D images, transferring these methods to 3D surfaces presents inherent challenges, including irregular data representation and the dichotomy between local and global characterizations of relief patterns. Such challenges have been highlighted in studies like that of Biasotti et al. [BTB*18]. In the domain of interest to us, namely the analysis of 3D texture patterns, methods can be categorized based on whether they utilize deep learning techniques. Traditional methods are briefly presented in Section 2.1, while learning-based methods are reported in Section 2.2. In alignment with the methods presented in SHREC'20 [MBG*20], non-deep learning approaches can be further classified based on two main adopted strategies. In the first strategy, methods work in the Euclidean domain either by

partially or fully projecting the 3D model onto a set of images and then applying texture image retrieval methods (see Section 2.1.1).

The second strategy work by extending the characterization of image textures directly within the 3D model, or more generally a non-Euclidean domain, which includes graphs, point clouds, and manifolds (see Section 2.1.2).

2.1. Traditional methods

The conventional approach to texture analysis involves the study and application of ad-hoc shape descriptors. Subsequently, the operation of texture retrieval or classification is executed using a specific metric applied to these descriptors.

2.1.1. Euclidean domain

The majority of the methodologies presented in the SHREC'17 and SHREC'20 tracks convert the 3D information of the mesh into 2D images. Subsequently, they identify texture patterns using image processing techniques like morphological methods. These methods often demonstrate superior performance compared to other proposed solutions. For example, Biasotti et al. [BTA*17] utilized covariance descriptors generated from 2D images derived from 3D geometries to compare texture patterns. Giachetti et al. [Gia18] followed a similar methodology, producing a 2D raster image from 3D meshes, then applying the Improved Fisher Vector (IFV) on the 2D image to extract feature vectors. Tatsuma and Aono [BTA*17] utilized depth images from 3D meshes to generate LBP images, further extracting Kaze features and statistical attributes from the LBP image.

2.1.2. Non-Euclidean domain

Non-Euclidean domains prove more challenging to address, as several properties, such as shift invariance, that are assured when working on grid structures, are lost. To counter this challenge, many techniques introduce local reference frames on the mesh to extend well-studied 2D methods. Renowned techniques like HOG [DT05] and LBP have seen their 3D counterparts as MeshHOG [ZBVH09] and MeshLBP [WBD14]. The meshLBP technique, inspired by LBP in 2D, leverages the Ordered Ring Facets (ORF) structure, aiming to emulate a consistent mesh support region akin to 2D grids. This approach encounters limitations when adjacent facets were missing from boundary facets.

2.2. Deep learning methods

More recent methods have begun to incorporate classifiers and feature extractors based on learning techniques. For example, the SHREC'20 competition [MBG*20] highlighted a hybrid technique, termed Deep Patch Metric Learning (DPML), which starts by translating a mesh to a graph and subsequently samples the mesh's surface to produce patches. These patches are converted to images, which are then fed into traditional CNNs for relief pattern classification.

In the field of geometric deep learning, different techniques explicitly designed for 3D meshes have also been proposed. For instance, Hanocka et al. [HHF*19] presented an approach that mir-

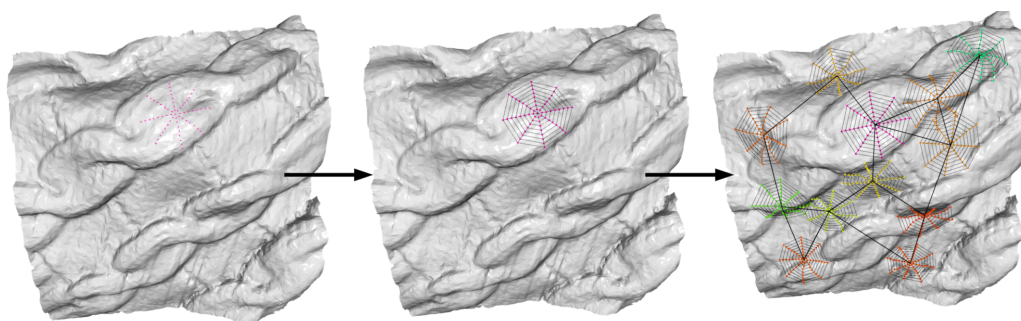


Figure 1: *MeshGraph construction: Initially, points are sampled on concentric rings (left). Subsequently, the SpiderPatch graph is assembled (middle) and, by combining multiple SpiderPatches, the final MeshGraph is constructed (right).*

rored traditional 2D convolution and pooling on 3D meshes. Convolution is performed on a 5D feature vector constructed at each facet by concatenating inner dihedral angles and edge properties. The application of symmetric functions to this 5D vector removes the order ambiguity between facet edges. Defining convolution on edges has the advantage of a natural pooling definition through collapse of non-significative edges. Sharp et al. [SACO22] bypassed the problem of defining convolution and pooling on non-Euclidean data by learning to simulate a diffusion process on the geometry. Each vertex is assigned a feature vector obtained from the corresponding Heat Kernel Signature value [SOG09], which is a shape descriptor derived from the heat kernel properties. The network performs a convolution-like operation by diffusing each feature vector to its neighbors based on a learned diffusion period, followed by integration through a Multi Layer Perceptron (MLP). Despite their superior performance in tasks such as shape correspondence, shape classification and segmentation, these recent convolution-based approaches have never been employed for the relief pattern classification task and thus, they constitute an interesting field of research to explore.

Nonetheless, in recent developments several methods have emerged that focus on mesh data by employing flexible structures like graphs. For example, in [GJFW22] the authors put forth a graph learning technique that classifies the texture of each facets within a 3D model, leveraging the versatility of graph-based representations. Here, the 3D mesh is transformed into a graph where each node symbolizes an aspect of the initial mesh. Feature vectors for each facet are derived from nearby facets and their geometric attributes. Subsequently, this graph is input into a Graph Neural Network (GNN) to determine if each node pertains to a textured or non-textured category. The method on which our work is based on [GTFB23] represents a successful example of GNN applied to non-Euclidean data.

In conclusion, while modern methodologies for relief pattern classification predominantly lean on techniques originally crafted for 2D images, they miss important information intrinsic to the manifold space. The results observed so far while using graph structures are promising. For this reason, it is essential to delve deeper into the potential of learning directly in the domain of manifolds. Such insights form the cornerstone of our proposed solution, which we elaborate upon in the subsequent sections.

3. Proposed approach

To provide context, we first briefly summarize the work of Guiducci et al. [GTFB23] in Section 3.1. In the next sections, we proceed by describing our developments based on their architecture. In Section 3.2, we introduce an algorithm to generate MeshGraphs with meaningful connectivity.

3.1. Bi-level GAT architecture

The first step to be able to input a mesh surface into the network is to generate the corresponding graph structures, denoted MeshGraphs and SpiderPatches. Different MeshGraphs on the same mesh are obtained by randomly selecting a pre-determined number of SpiderPatches. The latter can be described as an analogous to the patch operator introduced by Masci et al. [MBBV15], and it is obtained from a resampling algorithm devised by Tortorici et al. [TRBW20]. Examples of the MeshGraph construction are shown in Figure 1.

Next, each individual SpiderPatch is fed into a Graph Attention (GAT) [VCC*18] based network in order to produce meaningful embeddings of each local patch. The second GAT network receives these embeddings as node features of the related MeshGraph, producing a single embedding. A fully-connected network classifies each Meshgraph from these embeddings, producing a set of predicted labels.

A voting mechanism is introduced to mitigate the potential impact of randomly generated MeshGraphs on the classification results. With this voting method, a mesh is categorized into a specific class by applying a majority vote based on the classifications of the MeshGraphs generated from the mesh itself. While effective, this voting process requires a sufficient number of MeshGraphs to extract meaningful statistical properties of the mesh. Moreover, this method is not suitable for surfaces with non-uniform relief patterns, as information from each pattern type is averaged out. The whole process is summarized in Figure 2.

3.2. MeshGraph generation

Although the voting system is an effective mechanism to disambiguate the bias introduced by the use of random MeshGraphs, it

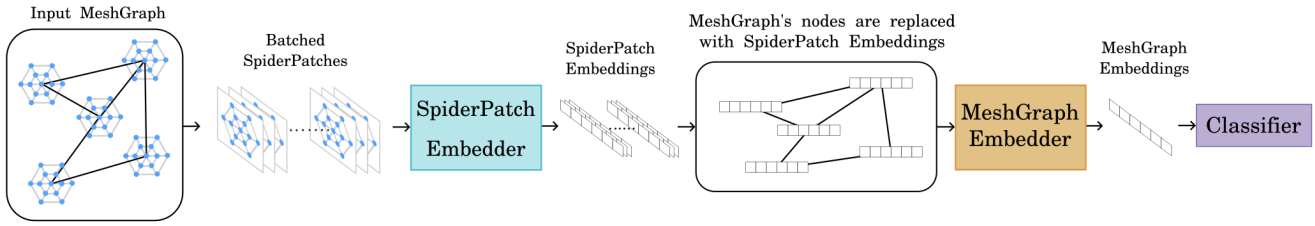


Figure 2: The bi-level procedure for MeshGraph classification: the first level GAT analyzes all the SpiderPatches, independently of a MeshGraph, producing embeddings; these embeddings are then used as node features in the second level GAT that analyzes the MeshGraph producing a single embedding; finally, this is used to classify the MeshGraph with a final layer.

necessitates a statistical sample of MeshGraphs to operate effectively, it is not repeatable, and mostly it requires generating and processing multiple Meshgraphs per mesh (25 as per Guiducci et al.). To enhance the system’s efficiency, we have implemented a parsimonious sampling method for the mesh, leading to the generation of MeshGraphs with more meaningful connectivity. This modified MeshGraph generation step grounds on the properties of the *Circle-Surface Intersection* (CSI) algorithm as proposed by Torrici et al. [TRBW20]. We choose a random vertex as the initial seed point of the first SpiderPatch. Then, we select the points in the outermost ring and add them as points belonging to the MeshGraph. The process repeats by iterating through these newly added points. A candidate point is rejected if it falls within the boundary defined by the MeshGraph, as well as when it is less than a given non-geodesic distance away from a MeshGraph node. For simplicity, this distance was chosen to be equal to the radius used to compute the SpiderPatches. The algorithm continues until there are no more nodes to add.

At this point, we obtained a set of SpiderPatches representing the MeshGraph nodes. To generate the final MeshGraph, we need to connect these nodes in a meaningful way. To define the graph connectivity, we opted for a triangulation algorithm of the nodes, so that the edges would connect spatially close nodes. There are several algorithms that generate a triangulation in 3D from a finite set of points. In our context, we require that the algorithm does not alter the position of the given point set. Thus, we choose the classic ball-pivoting algorithm, which ensures a watertight triangulation in most scenarios [BMR*99]. The ball radius is automatically determined by dividing the diagonal of the bounding box by the square root of the number of nodes [SO20]. This heuristic assumes a uniform distribution of points on the mesh. Given that the points reside on a 2D surface, the average distance between them is inversely proportional to the square root of their total number. Thus, the ball radius should also be proportional to this value. To achieve an appropriate radius, we scale this average distance using a characteristic length of the mesh, which we have selected as the diagonal of the oriented bounding box. The bounding box is based on the PCA of the convex hull of the mesh. By construction, a minimum geodesic distance is guaranteed between a sampled point and the nearest one. In this way, the ball-pivoting algorithm with the specified radius generates a watertight mesh.

To better understand the difference between our sampling and

the one proposed by Guiducci et al., we show an example of MeshGraph construction in Figure 3.

This MeshGraph generation has some advantages over the one described in [GTFB23]:

- The SpiderPatches cover the whole mesh, capturing enough information to process entire relief patterns. Changing the sampling distance affects the overlap area between SpiderPatches and thus the redundancy of information;
- Despite this redundancy, the number of SpiderPatches generated during the experiments is still close to that specified by Guiducci et al., making the computational cost comparable.
- Apart from the choice of the initial seed point, the MeshGraph is constructed in a deterministic manner, making the use of the voting system unnecessary. The bias introduced by the random seed point is negligible, as the final MeshGraph structure densely capture the geometric details of the pattern reliefs;
- The number of parameters to define during the pre-processing stage has been reduced. More precisely, we no longer need to define the number of MeshGraphs per mesh, as we construct a single MeshGraph that covers the whole mesh. Furthermore, the number of SpiderPatches now depends only on the result of the sampling method. Finally, the degree of each SpiderPatch in the MeshGraph is now computed via the ball-pivoting algorithm.

Our method is summarized in Algorithm 1.

3.3. Network architecture

To process the SpiderPatches and MeshGraphs, we defined a multi-level architecture based on two networks. Analogous to what presented in [GTFB23], we term each network as an *Embedder*.

The first embedder aims to embed the SpiderPatches into a latent representation. The SpiderPatch nodes contain the following features; Gaussian and mean curvatures, curvedness, max curvature ($K2$), and local depth (LD). Curvature values were calculated on 5 spatial resolution levels (see in the Table 1). These resolutions are expressed as percentages relative to the maximum extension of the oriented bounding box. Unless otherwise specified, all resolutions were used in the following experiments.

The second embedder generates a single feature vector that will be used to train the relief pattern classifier (i.e., a fully-connected

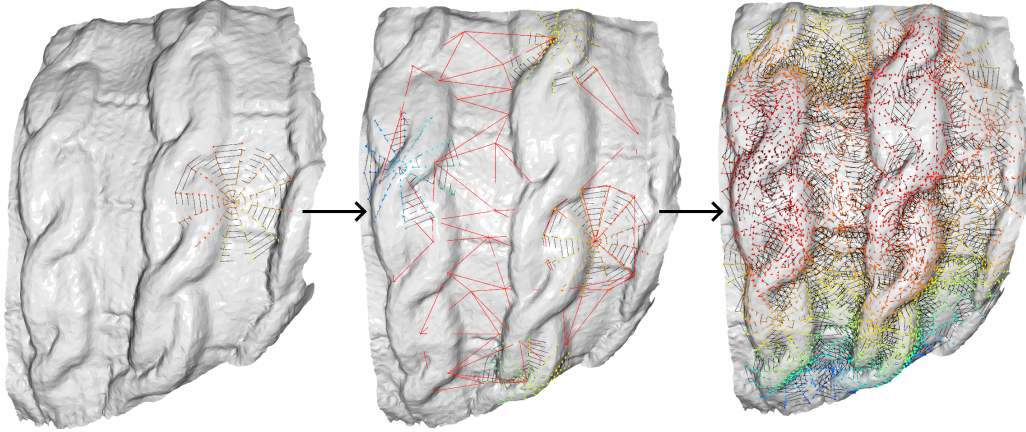


Figure 3: Our MeshGraph construction method: Initially, points are sampled on concentric rings and the SpiderPatch is assembled by connecting adjacent points (left). Subsequently, the MeshGraph nodes are generated by iteratively sampling the outermost rings of the SpiderPatches. The ball-pivoting algorithm generates the graph edges, highlighted in red (middle). Upon completion of the sampling process, a graph of graphs is obtained, wherein each node of the MeshGraph contains the SpiderPatches as attributes (right).

Algorithm 1 MeshGraph Generation

Require: Mesh M ; Initial mesh vertex $seed$; SpiderPatch radius $radius$; Number of rings per SpiderPatch $rings$; Points per rings $points$.

Ensure: MeshGraph object.

```

1:  $MG \leftarrow$  empty MeshGraph object
2:  $SP \leftarrow CSI(M, seed, radius, rings, points)$   $\triangleright$  SpiderPatch
   generation
3: Append  $SP$  to  $MG$  node attributes
4:  $new\_seeds \leftarrow$  outermost ring of  $SP$ 
5:  $sampler\_points \leftarrow [seed]$ 
6: Generate a KD-tree from  $sampler\_points$ 
7: while  $new\_seeds$  count  $> 0$  do
8:    $seed \leftarrow pop(new\_seeds)$ 
9:    $SP \leftarrow CSI(M, seed, radius, rings, points)$ 
10:  for all points  $p$  in outermost ring of  $SP$  do
11:     $neighbors \leftarrow$  KD-tree query for  $sampler\_points$ 
      within  $radius$  of  $p$ 
12:     $neighbors \leftarrow neighbors \setminus \{seed\}$ 
13:    if  $neighbors \cap sampler\_points \neq \emptyset$  then
14:      continue
15:    else
16:      Append  $p$  to  $new\_seeds$ 
17:    Append  $SP$  to  $MG$  node attributes
18:    Append  $seed$  to  $sampler\_points$ 
19:    Update KD-tree with new  $sampler\_points$ 
20: Generate connectivity in  $MG$  using ball-pivoting algorithm
21: Get final MeshGraph  $MG$ 

```

Table 1: Values of spatial resolutions used to compute curvatures. Each percentage refers to the maximum extent of the oriented bounding box.

Levels	Spatial resolutions				
	0	1	2	3	4
Percentage	0.1%	0.25%	1%	2.5%	5%

network). Each embedder shares the same common structure, composed of four main parts: *convolutional layer*, *normalization layer*, *readout layer*, and the *jumping knowledge layer*. An overview of the embedder structure is depicted in Figure 4.

Convolutional layer. It is implemented as a Multi-Head GAT [VCC*18], where the node embeddings $h_i^{(l+1)}$ in the layer $l+1$ are computed as:

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

where $N(i)$ is the set of one-hop neighbors of node i , a_{ij} the coefficient learned from the attention mechanism, $W^{(l)}$ is a shared weight matrix for node-wise transformation, and σ is the ReLU activation function. The expression in (1) represents a single-head attention layer; thus to define a multi-head attention layer, we parameterise different independent single-head layers and merge their outputs. For intermediary layers, we used concatenation as a merging operation, while for the final layer we averaged each output. In our architecture, each building block has a single two-headed GAT layer.

Normalization layer. We chose to maintain the mixture of normalization methods, similar to the approach in [GTFB23]. The techniques incorporated in the mixture are node normalization, ad-

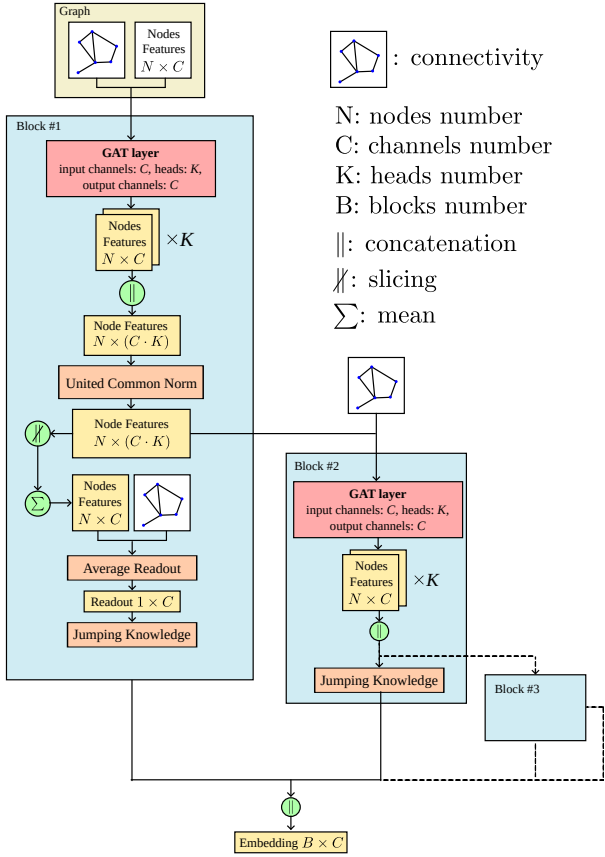


Figure 4: Design of the embedder. Each embedder is composed of several connected building blocks (blueish rectangles). Each unit incorporates a convolutional GAT, united common normalization, average readout, and jumping knowledge layers.

adjacent normalization, graph normalization and batch normalization. Each of these methods is weighted by a learned coefficient.

Readout layer. The readout process allows us to aggregate information across sets of nodes into a single fixed-length vector. Typically, the readout function is used as a final layer of the network. Here, we chose to simply average each node feature:

$$R_G = \frac{1}{|V|} \sum_{v \in V} h_v. \quad (2)$$

Jumping knowledge. In addition to standard adjacent layer connections, jumping knowledge involves connections between each layer's output and the final one, fostering a direct information pathway and enhancing the network's prediction capabilities. Let us consider a GAT with L layers and each layer's output $h^{(l)}$, where $l = 1, \dots, L$. In a jumping knowledge GAT, each $h^{(l)}$ is connected to the final layer through an aggregation function AG :

$$y = \text{classify}(AG(h^{(1)}, h^{(2)}, \dots, h^{(L)})). \quad (3)$$

In our design, each basic block produces an output of dimensions $K \times C^{(l)}$, where K represents the number of attention-heads, and

C corresponds to the number of channels in the convolution layer l . This output undergoes both concatenation and averaging, as depicted. Subsequent blocks utilize the concatenated output as their input (lines connecting different blocks in Figure 4), whereas the averaged output implements the jumping knowledge and it goes to the final layer.

Loss functions. In the training process of our network, we use the CETripletMG loss introduced by Guiducci et al. [GTFB23]. CETripletMG is a fusion of the traditional Cross Entropy loss and Triplet loss:

$$CETripletMG(y, \hat{y}, mg_{embed}) = \alpha \cdot \text{CrossEntropy}(y, \hat{y}) + \beta \cdot \text{Triplet}(y, mg_{embed}, m). \quad (4)$$

Here, y denotes the labels, \hat{y} are the predictions, and mg_{embed} represents the MeshGraph embeddings determined by the last embedder, with coefficients α and β both set to 0.5. The term $\text{Triplet}(y, mg_{embed}, m)$ embodies a triplet loss [SKP15] tailored for MeshGraphs. It leverages the online triplet mining strategy to compute the hardest-negative and hardest-positive examples within the batch of normalized MeshGraph embeddings provided. The margin m is set to 1.

4. Experimental results

In this section, we report the results of relief pattern classification in Section 4.1. In Section 4.2, we carried out an ablation study on SHREC'17 to determine the optimal parameters and assess their influence on the outcomes. Finally, experimental results are discussed in Section 4.3.

For our experiments, we sourced the raw mesh data from the SHREC'17 track dedicated to the *Retrieval of Surfaces with similar Relief Patterns* [BTA*17]. The full dataset encompasses 720 mesh surfaces, categorized into 15 classes with 48 elements each, as showcased in Figure 5. Each of these classes represents a distinct pattern derived from various poses and subjected to three distinct processing steps for every scan: two adaptive simplifications to 10K and 5K vertices and a re-sampling process to 15K vertices. These transformations intentionally alter the original mesh connectivity. Additionally, as the data were procured using a depth sensor, minor artifacts and subtle topological handles emerged. The collection of 180 distinctive raw scans, which include 15 patterns with 12 unique samples for each, is termed the *original dataset*.

Previous studies have harnessed the *full dataset*, comprising 720 meshes, for relief pattern retrieval tasks, noting perfect accuracy rates of up to 100% [TBOW21] or other significantly high scores [TB18; Gia18]. On the other hand, the **original dataset** presented a greater level of complexity, as evidenced by the findings in the literature (see Table 2). Consequently, we chose the original dataset as the foundation for our experiments.

4.1. Relief pattern classification

Among the participants in the SHREC'17 competition, the feature mapping method by [Gia18] exhibited the superior performance, when applied to the original dataset in the Tutte/meanC/SIFT/FV configuration. However, both the EdgeLBP by Thompson et

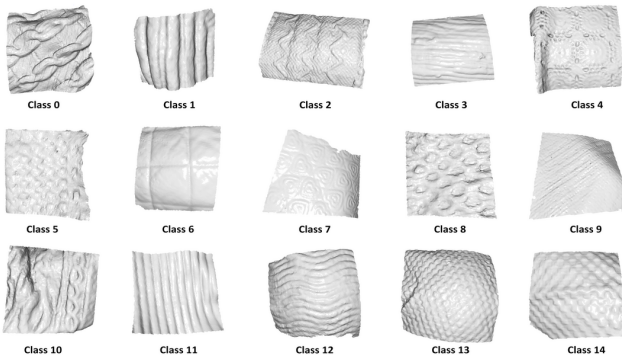


Figure 5: Representative meshes from the 15 classes in the SHREC'17 dataset.

al. [TB18] and Mesh-LBP+Mesh-Convolution by Tortorici et al. [TBOW21] showed superior performance in comparison. A potential explanation for its diminished performance might be attributed to the loss of depth information when the curvature values of the mesh are mapped onto the 2D plane. Meanwhile, the EdgeLBP, proposed by competition organizers, impressively achieved scores of 92.2% and 91.1% across its two runs. In line with other methods, the MeshLBP+Mesh-Convolution also displayed diminished performance when applied to the original dataset, with a drop in classification accuracy from 100% using the Sobel filter to 93.33%.

Table 2 summarizes the techniques employed for relief pattern classification on the original SHREC'17 dataset. The table is divided into techniques not leveraging deep learning and methods that apply deep learning on graphs. Results scored by our proposed method, as reported at the bottom of Table 2, are comparable to those of Tortorici et al. [TBOW21] when employing the Sobel filter. The method by Guiducci et al. [GTFB23] displays superior performance, irrespective of whether the voting system is used or not. In the case with no voting system, the network simply treated each MeshGraph generated from a mesh as an independent sample of the mesh itself. As will be elaborated upon in the subsequent sections, this difference in accuracy values can be ascribed to the redundancy of information when generating more than one MeshGraph per mesh and the effect of the triplet loss. We also observe that, given the limited number of models in the dataset (180) the percentage discrepancy is way less relevant than usual (a difference of about 2% is due to a misclassification of ~ 3 models wrt to [GTFB23] w/o voting, and of ~ 7 models with).

The percentages presented for our approach refer to the parameters in Table 3 that yield the best performance. The other parameters and protocols were set as follows. Our training spanned 60 epochs, utilizing an early stopping criterion. For robust generalization and comparability, especially with SHREC'17 outcomes, we implemented a 12-fold cross-validation. Consequently, 12 trained networks emerged, each classifying 11 samples and validating one. The network was trained using the AdamW optimizer [LH19] with an initial learning rate set to 0.01 and a learning rate decay of 0.5

every 20 epochs. A batch size of 16 was chosen for MeshGraphs and 512 for SpiderPatches.

Table 2: Comparison of our method with the results of SHREC'17 competition [BTA*17], EdgeLBP [TB18], Mesh-LBP and Mesh-Convolution [TBOW21], the IFV-based technique of [giachetti2018eff3ective], and lastly the framework proposed in [GTFB23]. Performance refers to the **original dataset** using Nearest Neighbor (NN) classification for No deep learning methods.

Traditional approaches		
Retrieval Method		NN
[BTA*17]	CMC-2	63.3%
	KLBO-FV-IWKS	52.2%
	KLBO-SV-IWKS	48.9%
[TB18]	EdgeLBP - run1	92.2%
	EdgeLBP - run2	91.1%
[Gia18]	P/mC/SIFT/FV	82.8%
	T/mC/SIFT/FV	87.2%
	MeshLDSift+FV	77.8%
[TBOW21]	Differential $r = 4$	87.2%
	Edge Detector (b)	90.0%
	Sobel	93.3%
	Haar H_2	90.0%
	Sharpen H_2	91.1%
Learning-based approaches		
Classification Method		Acc.
[GTFB23]	No Voting	95.0%
[GTFB23]	Voting	97.2%
	Ours	93.3%

Table 3: Test on different SHREC'17 MeshGraph datasets. Values in the Radius column are normalized by the number of rings, while "e" indicates that for each mesh the average edge length was used to compute the intersection radius. Blocks refers to the number of blocks in the last embedder.

SpiderP. params			Net. params	Statistics	
Radius	Rings	Points	Blocks	Acc.	Loss
e	6	8	3	80.0%	0.47
e	10	8	3	86.6%	0.65
e	10	12	3	93.3%	0.28
e	10	12	5	93.3%	0.31
0.02	10	12	3	86.6%	0.74
0.02	10	12	5	93.3%	0.46

4.2. Ablation study

In Table 3, we report an ablation study to assess the influence on the performance of the hyperparameters related to the MeshGraph generation and the GAT network. The rest of the parameters of the architecture are kept fixed. The accuracy values presented in the table were derived by employing 11 training meshes and 1 test mesh for each of the 15 classes. When using the proposed MeshGraph generation, the parameters to be calibrated are reduced to those defining the construction of individual SpiderPatches.

Table 4: Effect of different batch size and number of features on network performance. Each test was conducted using the following SpiderPatches parameters: radius equals to the average edge length; 10 rings per SpiderPatch; 12 points per ring. Sp. Res. refers to the spatial resolution levels of the curvature-based features.

Curvatures					Other params		Statistics	
Gauss	Mean	Curvedness	K2	LD	Sp. Res.	Batch Size	Acc.	Loss
✓	✓	✓	✓	✓	all	16	93.3%	0.28
✓	✓	×	×	✓	0,1	32	86.6%	0.42
×	×	×	×	✓	all	32	93.3%	0.35
✓	×	×	×	✓	0,1	64	53.3%	1.09
×	×	×	×	✓	all	64	93.3%	0.47

Several observations can be drawn from the table. Given the same patch size, the number of rings and points determines the density of information extracted from the mesh. Decreasing these two parameters results in a gradual decline in performance. In the experiments, two radius values were employed, which determine the extent of the SpiderPatches. The value denoted by e indicates that for each mesh, the radius was computed as the product of the average edge length and the number of SpiderPatch rings. The second value, set at 0.02, was determined by selecting the maximum value among the average edge length of the entire original dataset. The radius is one of the most critical parameters: excessively small values would render the MeshGraph construction process inefficient, while excessively large values would produce SpiderPatches with an overly broad coverage area, risking the loss of relief pattern details. From the tests conducted, determining the radius based on the average edge length of the individual mesh generally yields the best results. Lastly, the number of blocks in the final embedder influences the distance at which message passing occurs. Opting for a higher number of blocks enables the network to share features between distant relief patterns, thus enhancing the overall performance.

4.3. Discussion

The methods outlined in Table 2 adhere to the pattern retrieval protocol designated for the SHREC'17 competition. The performance of these techniques is assessed using the Nearest-Neighbor criterion; that is, given a query mesh from a certain class, the dataset is queried to find the mesh most akin to the presented one based on a specific similarity metric. In this context, to be able to compare our approach with the proposed ones, we conducted a standard 12-fold cross-validation (as there are 12 samples per class). For the classification task, we reached an accuracy of 93.3%, slightly outperformed by the method of Guiducci et al. [GTFB23] without voting system. We argue that this discrepancy in the results can be attributed to the fact that even without voting system, their network can learn from a greater batch of MeshGraphs. More specifically, their batch size of 512 MeshGraphs is better suited to work with triplet loss, since it can draw hardest-negative and hardest-positive examples from a larger set of samples. The use of a smaller batch size of 16 MeshGraphs was due to computational limitations. By reducing the number of features, it was possible to run several tests with a larger batch size. Interestingly, the spatial resolutions used during training plays an important role in the accuracies obtained. As shown in Table 4, keeping more curvature-based features, while

limiting the spatial resolution levels deteriorates the performance. On the other hand, it is sufficient to train the network on all spatial resolutions of the local depth to achieve accuracy values comparable to our best outcome. This is to some extent in agreement with the results obtained by [TBOW21], as it was observed that local depth outperforms K2 curvature in capturing small local variation of the surface. Ultimately, we chose a batch size of 16 to retain all features and allow a more direct comparison with the original work.

As an additional analysis, we compared our sampling algorithm with the farthest point sampling (FPS) [MD03], given its ubiquity in computer graphics and computational geometry for data simplification, particularly in point clouds. We first computed the average ratio of the number of points sampled with our method over the total vertex count, which for the original dataset is 0.5% with a standard deviation of 0.02%. Then, we processed each mesh in the dataset initialising the FPS algorithm with the computed ratio and SpiderPatch parameters set as in Table 4. The sampled points are then connected to create a MeshGraph using the ball-pivoting algorithm, with the radius set as described in Section 3.2. We observed a drop in accuracy levels, from 93.3% down to 69.3 on a 12-fold cross validation process. We argue that FPS generate more regular spaced sampling points, which can introduce aliasing effect relative to highly detailed reliefs. Further studies could shed more light on the reasons for this discrepancy in results.

5. Conclusions

In this paper, we introduced a novel sampling method for mesh structures, specifically tailored for graph-based pattern relief classifiers. The integration of a sampling algorithm for the MeshGraph generation has streamlined the complexity of the prior approach, diminishing both preprocessing time and the number of hyperparameters to be estimated, which are crucial to achieve high accuracies. The classifier, integrated with the proposed method, readily achieves accuracy values comparable to the state-of-the-art in pattern retrieval techniques. Furthermore, the current definition of SpiderPatch ensures rotational invariance and robustness against varied mesh tessellations.

To ensure invariance across distinct mesh samplings and invariance to the scale of relief patterns, future work will explore ways to automatically determining the SpiderPatch radius based on intrinsic characteristics of the relief patterns.

References

- [BBL*17] BRONSTEIN, MICHAEL M., BRUNA, JOAN, LECUN, YANN, et al. “Geometric Deep Learning: Going beyond Euclidean data”. *IEEE Signal Processing Magazine* 34.4 (2017), 18–42. DOI: [10.1109/MSP.2017.2693418](https://doi.org/10.1109/MSP.2017.2693418) 1.
- [BMR*99] BERNARDINI, F., MITTLEMAN, J., RUSHMEIER, H., et al. “The ball-pivoting algorithm for surface reconstruction”. *IEEE Transactions on Visualization and Computer Graphics* 5.4 (1999), 349–359. DOI: [10.1109/2945.8173514](https://doi.org/10.1109/2945.8173514).
- [BTA*17] BIASOTTI, SILVIA, THOMPSON, E MOSCOSO, AONO, MASAKI, et al. “Shrec’17 track: Retrieval of surfaces with similar relief patterns”. *10th Eurographics Workshop on 3D Object Retrieval*. 2017 1, 2, 6, 7.
- [BTB*18] BIASOTTI, SILVIA, THOMPSON, E MOSCOSO, BARTHE, LOIC, et al. “SHREC’18 track: Recognition of geometric patterns over 3D models”. *Eurographics workshop on 3D object retrieval*. 2018 2.
- [DT05] DALAL, N. and TRIGGS, B. “Histograms of oriented gradients for human detection”. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. 2005, 886–893. DOI: [10.1109/CVPR.2005.1772](https://doi.org/10.1109/CVPR.2005.1772).
- [Gia18] GIACHETTI, ANDREA. “Effective characterization of relief patterns”. *Computer Graphics Forum*. Vol. 37. 5. Wiley Online Library. 2018, 83–92 2, 6, 7.
- [GJFW22] GANAPATHI, IYYAKUTTI IYAPPAN, JAVED, SAJID, FISHER, ROBERT BOB, and WERGI, NAOUFEL. “Graph Based Texture Pattern Classification”. *Int. Conf. on Virtual Reality (ICVR)*. 2022, 363–369. DOI: [10.1109/ICVR55215.2022.9847889](https://doi.org/10.1109/ICVR55215.2022.9847889) 3.
- [GTFB23] GUIDUCCI, NICCOLÒ, TORTORICI, CLAUDIO, FERRARI, CLAUDIO, and BERRETTI, STEFANO. “Learning graph-based features for relief patterns classification on mesh manifolds”. *Computers & Graphics* (2023). URL: <https://api.semanticscholar.org/CorpusID:259941818> 1–8.
- [HHF*19] HANOCKA, RANA, HERTZ, AMIR, FISH, NOA, et al. “MeshCNN”. *ACM Transactions on Graphics* 38.4 (Aug. 2019), 1–12. DOI: [10.1145/3306346.3322959](https://doi.org/10.1145/3306346.3322959). URL: <https://doi.org/10.1145/3306346.3322959> 1, 2.
- [HLG*22] HU, SHI-MIN, LIU, ZHENG-NING, GUO, MENG-HAO, et al. “Subdivision-based mesh convolution networks”. *ACM Transactions on Graphics (TOG)* 41.3 (2022), 1–16 1.
- [LH19] LOSHCILOV, ILYA and HUTTER, FRANK. *Decoupled Weight Decay Regularization*. 2019. arXiv: [1711.05101](https://arxiv.org/abs/1711.05101) [cs.LG] 7.
- [Low04] LOWE, DAVID G. “Distinctive image features from scale-invariant keypoints”. *International journal of computer vision* 60 (2004), 91–110 2.
- [MBBV15] MASCI, JONATHAN, BOSCAINI, DAVIDE, BRONSTEIN, MICHAEL, and VANDERGHEYNST, PIERRE. *Shapenet: Convolutional neural networks on non-euclidean manifolds*. Tech. rep. 2015 3.
- [MBG*20] MOSCOSO THOMPSON, ELIA, BIASOTTI, SILVIA, GIACHETTI, ANDREA, et al. “SHREC 2020: Retrieval of digital surfaces with similar geometric reliefs”. *Computers & Graphics* 91 (2020), 199–218. ISSN: 0097-8493. DOI: <https://doi.org/10.1016/j.cag.2020.07.011>. URL: <https://www.sciencedirect.com/science/article/pii/S0097849320301138> 1, 2.
- [MD03] MOENNING, CARSTEN and DODGSON, NEIL A. “Fast Marching farthest point sampling”. *Eurographics*. 2003. URL: <https://api.semanticscholar.org/CorpusID:11826820> 8.
- [MLR*20] MILANO, FRANCESCO, LOQUERCIO, ANTONIO, ROSINOL, ANTONI, et al. “Primal-Dual Mesh Convolutional Neural Networks”. (2020). DOI: [10.48550/ARXIV.2010.12455](https://arxiv.org/abs/2010.12455). URL: <https://arxiv.org/abs/2010.12455> 1.
- [OPH94] OJALA, TIMO, PIETIKAINEN, MATTI, and HARWOOD, DAVID. “Performance evaluation of texture measures with classification based on Kullback discrimination of distributions”. *Proceedings of 12th international conference on pattern recognition*. Vol. 1. IEEE. 1994, 582–585 2.
- [OVSP13] OTHMANI, AHLEM, VOON, LEW FC LEW YAN, STOLZ, CHRISTOPHE, and PIBOULE, ALEXANDRE. “Single tree species classification from terrestrial laser scanning data for forest inventory”. *Pattern Recognition Letters* 34.16 (2013), 2144–2150 1.
- [SACO22] SHARP, NICHOLAS, ATTAIKI, SOUHAIB, CRANE, KEENAN, and OVSJANIKOV, MAKS. “DiffusionNet: Discretization Agnostic Learning on Surfaces”. 41.3 (Mar. 2022). ISSN: 0730-0301. DOI: [10.1145/3507905](https://doi.org/10.1145/3507905). URL: <https://doi.org/10.1145/3507905> 1, 3.
- [SKP15] SCHROFF, FLORIAN, KALENICHENKO, DMITRY, and PHILBIN, JAMES. “FaceNet: A unified embedding for face recognition and clustering”. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, 815–823. DOI: [10.1109/CVPR.2015.7298682](https://doi.org/10.1109/CVPR.2015.7298682) 6.
- [SO20] SHARP, NICHOLAS and OVSJANIKOV, MAKS. “PointTriNet: Learned Triangulation of 3D Point Sets”. *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020 4.
- [SOG09] SUN, JIAN, OVSJANIKOV, MAKS, and GUIBAS, LEONIDAS. “A Concise and Provably Informative Multi-Scale Signature Based on Heat Diffusion”. *Computer Graphics Forum* 28.5 (2009), 1383–1392. DOI: <https://doi.org/10.1111/j.1467-8659.2009.01515.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01515.x>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01515.x> 3.
- [TB18] THOMPSON, ELIA MOSCOSO and BIASOTTI, SILVIA. “Description and retrieval of geometric patterns on surface meshes using an edge-based LBP approach”. *Pattern Recognition* 82 (2018), 1–15 1, 6, 7.
- [TBOW21] TORTORICI, CLAUDIO, BERRETTI, STEFANO, OBEID, AHMAD, and WERGI, NAOUFEL. “Convolution operations for relief-pattern retrieval, segmentation and classification on mesh manifolds”. *Pattern Recognition Letters* 142 (2021), 32–38 1, 6–8.
- [TRBW20] TORTORICI, CLAUDIO, RIAHI, MOHAMED KAMEL, BERRETTI, STEFANO, and WERGI, NAOUFEL. “CSIOR: Circle-surface intersection ordered resampling”. *Computer Aided Geometric Design* 79 (2020), 101837 3, 4.
- [TWB21] TORTORICI, CLAUDIO, WERGI, NAOUFEL, and BERRETTI, STEFANO. “Representing and analyzing relief patterns using LBP variants on mesh manifold”. *Pattern Analysis and Applications* 24 (2021), 557–573 1.
- [VCC*18] VELIČKOVIĆ, PETAR, CUCURULL, GUILLEM, CASANOVA, ARANTXA, et al. *Graph Attention Networks*. 2018. arXiv: [1710.10903](https://arxiv.org/abs/1710.10903) [stat.ML] 3, 5.
- [WBD14] WERGI, NAOUFEL, BERRETTI, STEFANO, and DEL BIMBO, ALBERTO. “The mesh-lbp: a framework for extracting local binary patterns from discrete manifolds”. *IEEE Transactions on Image Processing* 24.1 (2014), 220–235 1, 2.
- [WTBD15] WERGI, NAOUFEL, TORTORICI, CLAUDIO, BERRETTI, STEFANO, and DEL BIMBO, ALBERTO. “Representing 3D texture on mesh manifolds for retrieval and recognition applications”. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, 2521–2530 1.
- [YLB*20] YANG, ZHANGSIHAO, LITANY, OR, BIRDAL, TOLGA, et al. *Continuous Geodesic Convolutions for Learning on 3D Shapes*. 2020. arXiv: [2002.02506](https://arxiv.org/abs/2002.02506) [cs.CV] 1.
- [ZBVH09] ZAHARESCU, ANDREI, BOYER, EDMOND, VARANASI, KIRAN, and HORAUD, RADU. “Surface feature detection and description with applications to mesh matching”. *2009 IEEE conference on computer vision and pattern recognition*. IEEE. 2009, 373–380 2.