SparseMeshCNN with Self-Attention for Segmentation of Large Meshes

Bjørn Hansen*[1], Mathias Lowes*[1], Thomas Ørkild[1], Anders Dahl[1], Vedrana Dahl[1], Ole de Backer[2], Oscar Camara[3], Rasmus Paulsen[1], Christian Ingwersen†[1,4], and Kristine Sørensen†[1]

[1]Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kgs. Lyngby, Denmark
[2]The Heart Center, Rigshospitalet, University of Copenhagen, Copenhagen, Denmark
[3]BCN MedTech, Universitat Pompeu Fabra, Barcelona, Spain
[4]Trackman A/S, Vedbæk, Denmark

Abstract

In many clinical applications, 3D mesh models of human anatomies are important tools for visualization, diagnosis, and treatment planning. Such 3D mesh models often have a high number of vertices to capture the complex shape, and processing these large meshes on readily available graphic cards can be a challenging task. To accommodate this, we present a sparse version of MeshCNN called SparseMeshCNN, which can process meshes with more than 60 000 edges. We further show that adding non-local attention in the network can mitigate the small receptive field and improve the results. The developed methodology was applied to separate the Left Atrial Appendage (LAA) from the Left Atrium (LA) on 3D mesh models constructed from medical images, but the method is general and can be put to use in any application within mesh classification or segmentation where memory can be a concern. [1]

1 Introduction

A variety of complex anatomies exist in the human body and in many clinical settings patient-specific models are a key element in treatment and diagnosis. Triangulated meshes provide an efficient representation for these tasks, and can be obtained either directly from a 3D surface scanner or from isosurfacing in image volumes. In many clinical applications the meshes to be processed consist of many triangles and exhibit large variations in shape and size. Combining this with the fact that clinical datasets often have few examples, unbalanced class distributions, and contain noise, it is a difficult task to process such data. One example of such clinical application is the task of segmenting the Left Atrial Appendage (LAA) from the remaining Left Atrium (LA). The LAA is a tubular-like structure originating from the LA and exhibits large variations in shape and size. Patients suffering from Atrial Fibrillation (AF) have a 5-7 times higher risk of experiencing an ischemic stroke, and for these patients, the most common thrombus location site is inside the LAA, where up to 99% of the reported thrombi are located [3, 2]. To prevent strokes in AF-patients one can implant an occluding device into the LAA opening. We aim to demonstrate an automatic method for LAA segmentation, which can be a useful tool for medical doctors to plan and execute the LAA occlusion procedure.

Part segmentations of geometric data such as meshes are heavily investigated within the field of Geometric Deep Learning [5, 6, 10, 12, 15], but the methods are all developed for standardized datasets with limited variations and a small number of edges. These methods have not yet intruded into the medical domain, and we believe a

*Equal contribution

†Equal contribution

[1] Code available at Github here

https://doi.org/10.7557/18.6281

reason for this is their inability to process large meshes without infeasible large memory requirements. In this paper, we, therefore, introduce a memory-reduced version of the popular MeshCNN [5] and call it SparseMeshCNN. Recent modifications to the MeshCNN architecture includes increased representational power by leveraging a representation of the first and second fundamental form in MeshCNN Fundamentals [1], and increased memory efficiency on fine-grained medical data in MedMeshCNN [17] by utilizing a sparse book keeping matrix. However, both of these proposed improvements have drawbacks. MeshCNN Fundamentals lacks the ability to process larger meshes, and MedMeshCNN does not address the issues with increasing the number of edges while maintaining small convolutional kernels, which limits the receptive field to mainly capture local dependencies. To circumvent this we introduce a non-local block [23], which allows us to capture long-range dependencies without the need for large stacks of convolutions. We show that our SparseMeshCNN with attention can process meshes with more than 60 000 edges on a readily available 12GB GPU.

2 Related Work

In the following paragraphs, we will briefly introduce four different clusters of methods for processing geometric data: Conversion to euclidean data, point cloud processing, graph-based methods, and mesh-based methods. After this, we will introduce attention and shortly describe current efforts for part segmentation in medical applications.

Conversion to euclidean data. One can convert a 3D object into multiple 2D-views and use regular CNNs on each view [18]. Alternatively, the surface can be converted into a 3D occupancy grid, which can be processed with 3D U-nets [25] or octree-based CNN's [22]. These methods however struggle with the artifacts introduced when mapping into the euclidean domain and the high memory requirements of, especially 3D convolutions.

Point Clouds. The lightweight point cloud representation allows for processing a large number of points with methods such as PointNet [14] and PointNet++ [15]. The point clouds do not have any information about connectivity and can therefore not represent the topology of complex anatomies.

Graph-based methods. Graph-based methods consider meshes as graphs with vertices as nodes and edges as relationships. The Mixture Model Networks (MoNet) [11] showed that many of the proposed CNNs for non-euclidean manifolds are similar methods with different definitions of local patches and weighing kernels. The Dual-Primal Graph Convolutional Network (DPGCNN) [12] further builds on this by adding Graph Attention Networks (GATs) [21] and combining information from the primal and the dual graph. Common to many graph-based methods is that they do not consider the geometrical properties of the meshes. This is especially evident in the pooling operations, which often build on classic graph-coarsening techniques aiming to minimize a non-task-specific error if pooling is done at all.

Mesh-based methods. The mesh-based methods are designed to exploit the geometrical properties of triangulated meshes. The method from [19] suggested using 2D CNNs to process features defined on the tangent planes, whereas the SpiralNet [4] defines the convolutions on spirals around each mesh vertex. Hanocka et al. [5] introduced MeshCNN, where convolutions are defined on the edges based on symmetric aggregation of features from the 1-ring neighborhood of an edge. In addition, they propose a task-specific pooling operation that aims to collapse edges that contribute the least to the decision-making. Milano et al. [10] combined the feature extraction from MeshCNN with the training method of DPGCNN by performing alternating convolutions on the primal and dual graph. The MeshWalker [6] by Lahav et al. utilizes the mesh structure by feeding random walks on the surface into a RNN. The features from the walks are extracted as the translation in each step of the walk.

Attention. In general terms, the attention mechanisms allow for modeling dependencies in the input without regards to their distance in the input or output [20]. The Graph Attention Network (GAT) from [21] generalized attention mechanisms to graphs by gathering features from neighboring nodes and weighing them through a learnable attention parameter. The method showed promising results on network graph learning and mesh classification and segmentation tasks [12, 10]. Another variant of attention is found in the non-local block [23]. They demonstrate the effectiveness on image and video classification and object detection, but it has not yet been extended to geometrical data.

Medical Applications. One other method has been developed for automatic detection of the LAA orifice [7]. The method takes base in a voxel-wise segmentation of CT images and aims to find the narrowest part of the LAA neck along the centerline. Learning-based methods for mesh segmentation have however been explored in other medical fields. Yang et al. [24] segments aneurism on brain vessels and compares methods working on projected views, voxels, points, and mesh representations of the blood vessels. They show that the segmentation accuracy increases, when the number of edges in the mesh is increased, which emphasizes the need for our proposed SparseMeshCNN.

3 Materials and Methods

This section contains a presentation of the dataset and a short introduction to the main elements of MeshCNN [5]. We mainly focus on how sparse matrices can be used for bookkeeping in the pooling operation and describe attention using non-local blocks. Lastly, all relevant technical and implementation details are provided.

3.1 Data

The data is acquired and provided by Rigshospitalet, Copenhagen. It consists of 106 randomly selected participants, who had undergone a cardiac computed tomography angiography (CCTA) examination for research purposes in the period 2010–2013. The images are semi-manually annotated and the segmentation is converted to a triangulated mesh using Marching Cubes isosurfacing [8] and post-processed with one round of Laplacian smoothing (relaxation factor 0.1) and a 50% increase in triangle size as implemented in [13].

LAA segmentation. The part of the mesh that is considered LAA was annotated using an automatic method relying on point correspondence between all meshes in the dataset. An expert annotator marked the LAA neck points on the template mesh and these points were propagated to all other meshes. For each mesh, a plane was fitted to the propagated neck points and small displacements were added to the plane parameters to find the plane that slithers the LAA neck with the smallest possible cross-sectional area. The segmentation is deemed satisfactory by visual inspection of all examples by the expert.

Data split. We split the data into 76 meshes used for training, 10 meshes for validation, and 20 meshes for testing. The number of faces in the meshes ranges from 20 322 to 44 742, where 14.2% to 33.3% of the faces belong to the LAA. Because of these significant differences, we construct the data split based on the number of faces in the mesh such that a uniform distribution is approximately achieved among the sets.

3.2 SparseMeshCNN

With MeshCNN, Hanocka et al. [5] introduced a general convolution, pooling, and unpooling operation for meshes. These operations are defined on the edges of the mesh and consider the 1-ring neighborhoods of the edges i.e. all of the edges in the two incident faces to a given edge. We shortly describe the definition of convolutions using 1-ring neighbors from the original MeshCNN and from there focus on our contribution to reduce the memory-consumption of the pooling and unpooling operations and how to introduce non-local attention.

Convolution on meshes. In MeshCNN, symmetric features are computed for each edge and used to generate an image-like structure representing the mesh, on which it is possible to apply ordinary 2D convolutional operations. Five features are computed for each edge: the dihedral angle, the two opposite angles in the 2 connecting faces, and the two ratios between the length of the edge and the two "heights" of the connecting faces. The neighborhood is defined as the 1-ring neighbors and one can therefore construct a 5-channel feature image of size $5 \times n_e \times (n_n + 1)$, where n_e is the number of edges in the mesh and n_n is the number of neighbors. When applying 2D convolutions to such an image, we combine the features from the edge itself and the features from the neighboring edges to create new features. For more information on the features and how to avoid ambiguities in ordering see the original MeshCNN [5].

Pooling & unpooling on meshes. After each convolution, a new feature vector is calculated for all edges. The ℓ_2-norm of the feature vector is calculated as a measure of importance for each edge. The edges with the lowest norm will be collapsed iteratively by merging the vertices corresponding to the edge and their features are averaged (see Figure 1).

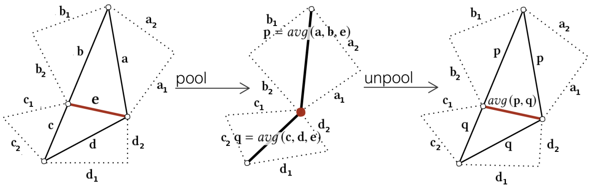

Figure 1: The pooling and unpooling operator collapsing and restoring edge e. Note the $\mathbf{p}$ and $\mathbf{q}$ denotes the feature vectors of edges p and q. The figure is modified from [5].

Because the unpooling operation has to restore the mesh to its state before pooling, it is essential to keep track of the collapsed edges during pooling. It is in this stage we can reduce memory consumption by introducing sparse matrices. To keep track of the pooling, a matrix $\mathbf{G}$ of size $n_e \times n_e$ is used. Each entry $\mathbf{G}_{i,j}$ denotes if edge i has been collapsed into edge j. To collapse edge e_i into e_j, we add row $\mathbf{G}_i$ to row $\mathbf{G}_j$. However, for large meshes the size of $\mathbf{G}$, scales quadratically with the number of edges, as the pooling sizes should be correlated with the size of the meshes. To reduce the size of $\mathbf{G}$, we have implemented it as a SparseTensor in PyTorch. PyTorch SparseTensors are stored in coordinate-format (COO), which stores the values, $\mathbf{v}$, and the indices of those values, $\mathbf{I}$. The matrix $\mathbf{I}$ is constructed as a $2 \times n_0$ such that the first row holds row indices of all non-zero elements in $\mathbf{G}$ and the second row holds column indices. The current state of SparseTensors in PyTorch is somewhat limited and we cannot easily make row addition in a SparseTensor. To add row $\mathbf{G}_s$ to row $\mathbf{G}_t$ and thus collapsing edge e_s into edge e_t, we need to find all row indices of $\mathbf{I}$ with the same value as row s, extract the value of these indices as well as the indices of the columns. The target row t and the column indices are now appended to $\mathbf{I}$ and the extracted values are appended to $\mathbf{v}$.

The process of finding the row indices of $\mathbf{I}$ equal to s, becomes less trivial when, instead of having a single source edge s, we have a list of source edges, $\mathbf{s}$, and a list of associated target edges, $\mathbf{t}$. Now we both have to find all the indices of row $\mathbf{I}$ that is also in $\mathbf{s}$, but we also need to keep track of what the target edge is for that specific source edge. To solve this, we generated an intermediate matrix, $\mathbf{M}$, of size $len(\mathbf{I}) \times len(\mathbf{s})$ where each entry $\mathbf{M}_{i,j}$ is true if $\mathbf{I}_{1,i} = \mathbf{s}_j$ i.e. the row index of i is equal to the source row of j, and false otherwise. By finding the indices of the true elements of $\mathbf{M}$, we are thus able to get elements of $\mathbf{G}$ and the matched source-target pair, which we can then use to update $\mathbf{G}$ in a vectorized manner. For large meshes $\mathbf{M}$ can be quite large. To restrict the memory requirement of $\mathbf{M}$, we therefore restrict the size to be at most $len(\mathbf{I}) \times len(\mathbf{s}_{1,2,\ldots,b})$ where b is a buffer size. By only selecting the first b elements of our source-target pairs, we make a trade-off between speed and memory. Through previous experiments we found that a buffer size of $b = 200$ yielded best results. Thus, we use this buffer size for the experiments discussed in this paper.

3.3 Attention using non-local blocks

Wang et al. [23] introduced an approach to capture long range dependencies in data using a non-local block. The progressive behavior of a convolution operation is in the non-local block replaced by directly computed interactions between any two edges regardless of their distance. The interactions between edges are captured with a generic non-local operation given by

$$\mathbf{y}_i = \frac{1}{\mathcal{C}(\mathbf{x})} \sum_{\forall j} f\left(\mathbf{x}_i, \mathbf{x}_j\right) g\left(\mathbf{x}_j\right), \qquad (1)$$

where f is a pairwise and g a unary function. The input $\mathbf{x}_i$ to the non-local block consists of the feature vector of the edge i itself and the symmetric feature vectors of its 1-ring neighborhood. We are thus computing interactions between each edge and its 1-ring neighborhood and all other edges. As the function f and the normalization $\mathcal{C}$ are chosen to be the embedded Gaussian $f\left(\mathbf{x}_i, \mathbf{x}_j\right) = e^{\theta(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}$ and $\mathcal{C}(\mathbf{x}) = \sum_{\forall_j} f(\mathbf{x}_i, \mathbf{x}_j)$ the non-local operation becomes the self-attention module from [20], with $\theta\left(\mathbf{x}_i\right)$, $\phi\left(\mathbf{x}_j\right)$ and $g\left(\mathbf{x}_j\right)$ corresponding to the queries, keys and values. These embeddings are learned through the training of the model.

We add a residual connection such that the output from the non-local block is

$$\mathbf{z}_i = W_z \mathbf{y}_i + \mathbf{x}_i, \tag{2}$$

where $\mathbf{y}_i$ is given in Equation 1.

The matrix storing the computed interaction between any two edges contains $n_e \times n_e$ elements, and can therefore be quite memory heavy. To reduce the amount of pairwise computations a subsampling trick is used, where Equation 1 is modified to

$$\mathbf{y}_i = \frac{1}{\mathcal{C}(\hat{\mathbf{x}})} \sum_{\forall j} f\left(\mathbf{x}_i, \hat{\mathbf{x}}_j\right) g\left(\hat{\mathbf{x}}_j\right), \tag{3}$$

where $\hat{\mathbf{x}}$ is a subsampled version of $\mathbf{x}$. This reduces the amount of pairwise computation to 1/4 and does not alter the non-local behavior. The subsampling is done by applying a maxpooling layer after the ϕ and g embeddings. To further reduce computations the number of channels in the θ, ϕ and g embeddings are set to half of the number of channels in the input.

3.4 Implementation details

Network architecture. Using the defined convolution, pooling and unpooling operations we can use a U-Net [16] architecture as our network. The non-local blocks are placed after the last and second to last pooling operations as illustrated in Figure 2. At each stage in the U-Net, the number of edges is pooled to a fixed number, which is given as a hyper parameter to the model.

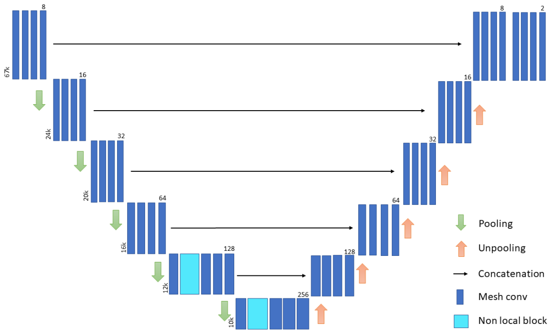

Figure 2: The network architecture for SparseMeshCNN with attention. The number of edges after each pooling operation is provided in the lower left of each stage and the number of channels is denoted in the top right of each stage.

Training details. The models were trained on a 12 GB GTX Titan X graphics card. The loss function is binary cross-entropy loss and the network is updated using the Adam optimizer with a learning rate of 0.001 for the first 200 epochs and linear decay to zero over the next 400 epochs. An epoch takes around 18 minutes, while a single forward pass can be computed in 5-10 seconds.

Post processing. We utilize that the LAA is one connected part of the mesh to improve the segmentations. With connected component analysis of the segmentations, we can extract the largest connected LAA and LA components. If this leaves unlabelled faces, these are filled with the label of its nearest neighbor.

3.4.1 Data augmentation.

To compensate for the few available data examples, we implement local augmentation and thin plate spline augmentation.

Local augmentation. The features of the MeshCNN are invariant to similarity transformations, i.e. scaling, rotation and translation do not generate new features. However, we can apply anisotropic scaling, where each vertex coordinate is scaled separately. We scale all (x, y, z) coordinates with (s_x, s_y, s_z) with each s_j being sampled randomly from $\mathcal{N}(0, 0.1^2)$. In places where the mesh is approximately flat, we apply vertex shifting and edge flipping to augment the mesh without changing the morphology.

Thin plate spline augmentation. The thin-plate spline (TPS) data augmentation is an augmentation of the whole mesh, and the output of the augmentation is a new mesh with new features. We generate a grid ranging from the smallest x, y, z coordinate values $(x_{min}, y_{min}, z_{min})^{\mathrm{T}}$ to the largest $(x_{max}, y_{max}, z_{max})^{\mathrm{T}}$ and denote the grid points our source points. By making a random displacement to each of the source points we can acquire the target points as

$$t_{ij} = k \cdot u \cdot s_{ij}, \tag{4}$$

where t_{ij} is the j'th coordinate of the i'th point, k is a scalar and $u \sim U(-1, 1)$. We find a mapping $f : s \mapsto t$ using the TPS basis functions $\phi(r) = r^2 \log r$ and a linear polynomial basis. The acquired mapping from the source points to the target points can now be used on all vertices in a mesh, thus a new training example has been made.

	SparseMeshCNN	**Attention**
Pooling res.	[28k,24k,20k,16k,12k,10k]	[24k,20k,16k,12k,10k]
# Features	[16,32,64,128,256,256,512]	[8,16,32,64,128,256]
# Conv. Layers	4	4
NL block	None	Two bottom layers

Table 1: Settings for the best performing models with and without attention.

	SparseMeshCNN		**+Attention**	
		+CC		+CC
Area accuracy[%]	92.36	95.36	93.23	96.19
Area DICE[%]	79.79	88.65	81.53	90.71
COM distance[mm]	11.61	5.25	11.14	4.87

Table 2: Area weighted face accuracy, DICE-score and center-of-mass distance for our proposed SparseMeshCNN with and without attention, before and after connected component analysis (CC).

4 Experiments and Results

Since we are segmenting surfaces, we argue that the most representative evaluation methods should operate on the faces of the mesh. Since the faces can have different area, we present an area weighted face accuracy and dice score. We furthermore compute the center-of-mass (COM) of the poly-ring separating the LAA from the remaining LA and compare it between the true and the predicted segmentation. The poly-ring is extracted as all edges that are located between two faces that are assigned to different classes. This COM is important to locate precisely as it can be interpreted as the intended position of the occluding device.

Our SparseMeshCNN can process all meshes in the dataset with no need to downsample the meshes to decrease the number of edges. To evaluate the effect of introducing non-local attention, we compare to a standard SparseMeshCNN. To ensure fair comparison we added an extra pooling layer at the beginning of the SparseMeshCNN with 16 features and a resolution of 28 000 edges and doubled the number of features in each of the remaining layers. See Table 1 for the exact pooling layers and number of features. The results are visualized Figure 3 and quantitatively shown in Table 2.

5 Discussion

Our SparseMeshCNN has shown to be able to learn a complex anatomical segmentation task on meshes with more than 60 000 edges while running on readily available GPUs. The large increase in the number of edges however creates issues with a small receptive field due to the small convolutional kernels and the limitations of manifold meshes making it impossible to pool to very low resolutions in the bottle-neck layers. The non-local attention seemed to increase the segmentation accuracy. Due to our restrictions in memory requirements, it was only possible to place the non-local block in the lower layers of the network, which means interactions can only be computed between higher-level features in a low-resolution version of the mesh. The benefit of the non-local block is that it can easily be plugged in at any stage as a substitution for a normal convolutional layer. The non-local block can therefore easily be added at earlier stages in applications where memory is less of an issue.

Attention was implemented to increase our models receptive field, which was problematic in early experiments. Another way to increase the receptive field is by having a deeper network with many more convolutions. Experiments with this has shown an almost equally as good performance. We were not able to compare the significance of the sparseness of our method on the LAA segmentation since we cannot process the data without it.

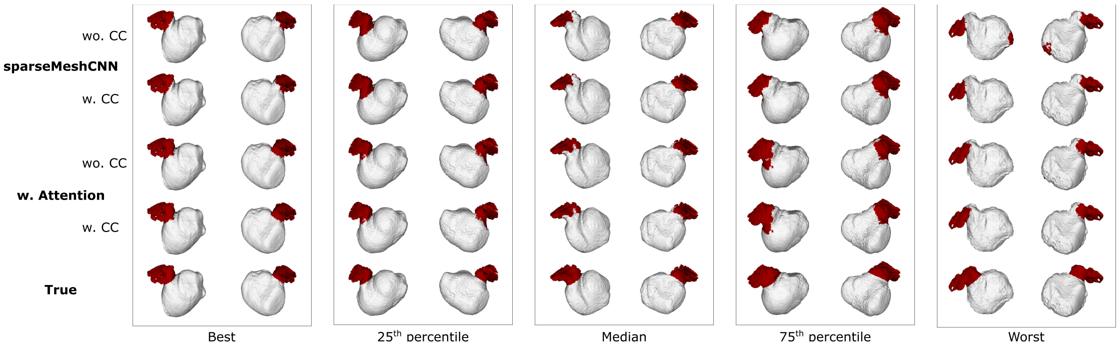

Figure 3: Visual results on the three different models showing (from the left) the best, 75^{th}, 50^{th}, 25^{th} percentile and worst segmentation results evaluated by Dice-score on the model using attention. We show results with and without connected component (CC) post-processing.

The only other LAA ostium detection method available [7] solely evaluated their results using the COM distance. Compared to our ≈ 4.70 mm, they reported a distance of ≈ 2.51 mm. It should however be taken into account that their method only takes a small patch around the LAA into account as opposed to our method processing the entire LA. The COM-distance measure also favors their methods since they restrict the position to be on the centerline derived with manual interaction. Alternatively one can see our results in relation to the results on benchmark datasets such as the human body segmentation [9]. In the original MeshCNN [5] the meshes each consisted of up to 2250 edges and they reported state-of-the-art segmentation accuracy of 92.3% on the dataset. Using the SparseMeshCNN with Attention on the human body dataset we obtain an accuracy of 93.9%, which is a signifucant improvement to the MeshCNN. However, newer methods such as MeshWalker [6] by Lahav et al. reports an accuracy of 94.8%. With these results in mind, we argue that the 96% accuracy we obtained on the LAA segmentation task is acceptable considering the challenges we encounter with a small data set, large meshes, large shape differences, and possibly noisy labels.

6 Conclusion

The SparseMeshCNN is capable of processing large meshes with more than 60 000 edges without needing very large graphic cards that are difficult and expensive to procure. The task of LAA part segmentation is a typical clinical problem with few data examples, large meshes, and highly variable shapes. Nevertheless, we demonstrate results comparable to what has been achieved on idealized benchmark data sets and take a step in the direction of automatic LAA ostium detection to aid the LAA occlusion interventions.

References

[1] A. Barda, Y. Erel, and A. H. Bermano. Meshcnn fundamentals: Geometric learning through a reconstructable representation. *CoRR*, abs/2105.13277, 2021. URL `https://arxiv.org/abs/2105.13277`.

[2] A. Cresti, M. A. García-Fernández, H. Sievert, P. Mazzone, P. Baratta, M. Solari, A. Geyer, F. De Sensi, and U. Limbruno. Prevalence of extra-appendage thrombosis in non-valvular atrial fibrillation and atrial flutter in patients undergoing cardioversion: a large transoesophageal echo study. *EuroIntervention*, 15: e225–e230, 2019. doi: 10.4244/eij-d-19-00128.

[3] M. Glikson, R. Wolff, G. Hindricks, J. Mandrola, A. J. Camm, G. Y. Lip, L. Fauchier, T. R. Betts, T. Lewalter, J. Saw, A. Tzikas, L. Sternik, F. Nietlispach, S. Berti, H. Sievert, S. Bertog, and B. Meier. EHRA/EAPCI expert consensus statement on catheter-based

left atrial appendage occlusion - An update. *EuroIntervention*, 15(13):1133–1180, 2020. doi: 10.4244/EIJY19M08_01.

[4] S. Gong, L. Chen, M. Bronstein, and S. Zafeiriou. Spiralnet++: A fast and highly efficient mesh convolution operator. *Proceedings - 2019 International Conference on Computer Vision Workshop, Iccvw 2019*, 2019. doi: 10.1109/ICCVW.2019.00509.

[5] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. Meshcnn: A network with an edge. *Acm Transactions on Graphics*, 38(4):90, 2019. doi: 10.1145/3306346.3322959.

[6] A. Lahav and A. Tal. Meshwalker: Deep mesh understanding by random walks. *CoRR*, abs/2006.05353, 2020. URL `https://arxiv.org/abs/2006.05353`.

[7] H. Leventić, D. Babin, L. Velicki, D. Devos, I. Galić, V. Zlokolica, K. Romić, and A. Pižurica. Left atrial appendage segmentation from 3D CCTA images for occluder placement procedure. *Computers in Biology and Medicine*, 104:163–174, 2019. doi: 10.1016/j.compbiomed.2018.11.006.

[8] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987. doi: 10.1145/37402.37422.

[9] H. Maron, M. Galun, N. Aigerman, M. Trope, N. Dym, E. Yumer, V. G. Kim, and Y. Lipman. Convolutional neural networks on surfaces via seamless toric covers. *Acm Transactions on Graphics*, 36(4):71, 2017. doi: 10.1145/3072959.3073616.

[10] F. Milano, A. Loquercio, A. Rosinol, D. Scaramuzza, and L. Carlone. Primal-dual mesh convolutional neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020. URL `https://github.com/MIT-SPARK/PD-MeshNet`.

[11] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *Proceedings - 30th Ieee Conference on Computer Vision and Pattern Recognition, Cvpr 2017*, pages 5425–5434, 2017. doi: 10.1109/CVPR.2017.576.

[12] F. Monti, O. Shchur, A. Bojchevski, O. Litany, S. Günnemann, and M. M. Bronstein. Dual-primal graph convolutional networks. volume abs/1806.00770, 2018. URL `http://arxiv.org/abs/1806.00770`.

[13] R. R. Paulsen, J. A. Baerentzen, and R. Larsen. Markov random field surface reconstruction. *IEEE transactions on visualization and computer graphics*, 16(4):636–646, 2009. doi: 10.1109/TVCG.2009.208.

[14] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. volume abs/1612.00593, 2016. URL `http://arxiv.org/abs/1612.00593`.

[15] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017. URL `http://arxiv.org/abs/1706.02413`.

[16] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL `http://arxiv.org/abs/1505.04597`.

[17] L. Schneider, A. Niemann, O. Beuing, B. Preim, and S. Saalfeld. Medmeshcnn - enabling meshcnn for medical surface models. *CoRR*, abs/2009.04893, 2020. URL `https://arxiv.org/abs/2009.04893`.

[18] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. *Proceedings of the IEEE International Conference on Computer Vision, 2015*, 2015. doi: 10.1109/ICCV.2015.114.

[19] M. Tatarchenko, J. Park, V. Koltun, and Q. Y. Zhou. Tangent convolutions for dense prediction in 3d. *Proceedings of the Ieee Computer Society Conference on Computer Vision and Pattern Recognition*, 2018. doi: 10.1109/CVPR.2018.00409.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

[21] P. Veličković, A. Casanova, P. Liò, G. Cucurull, A. Romero, and Y. Bengio. Graph attention networks. *6th International Conference on Learning Representations, Iclr 2018 - Conference Track Proceedings*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

[22] P. S. Wang, Y. Liu, Y. X. Guo, C. Y. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *Acm Transactions on Graphics*, 36(4), 2017. doi: 10.1145/3072959.3073608.

[23] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. *Proceedings of the Ieee Computer Society Conference on Computer Vision and Pattern Recognition*, 2018. doi: 10.1109/CVPR.2018.00813.

[24] X. Yang, D. Xia, T. Kin, and T. Igarashi. INTRA: 3D intracranial aneurysm dataset for deep learning. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2653–2663. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020. doi: 10.1109/CVPR42600.2020.00273.

[25] Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9901:424–432, 2016. doi: 10.1007/978-3-319-46723-8_49.