SparseMeshCNN with Self-Attention for Segmentation of Large Meshes

Christian K. Ingwersen et. al - cin@trackman.com

Segmentation of 3D shapes has recently been a hot topic within deep learning and several approaches to how to solve it has been presented. When working with shape data computational efficiency is often a strict requirement due to the hugely increased complexity compared to simple images. We present a new computationally efficient extension of MeshCNN architecture which allows the architecture to be used on large meshes, which is usually required in a production setting.

Introduction

We present an extension of MeshCNN which introduces a way to do convolutions, pooling, and unpooling of meshes. Our contribution to the architecture is a sparse reimplementation of the pooling operation that further allows it to pool edges in batches instead of sequentially.

Convolutions and Pooling on meshes

As meshes are not ordered in a regular grid-like 2D images we cannot apply a traditional convolutional operation on the mesh. In MeshCNN features for each edge are instead defined as the two inner angles, edge-length ratios and the dihedral angle. These features are then used to generate an image like structure of dimension, *b x c x e x n+1.*

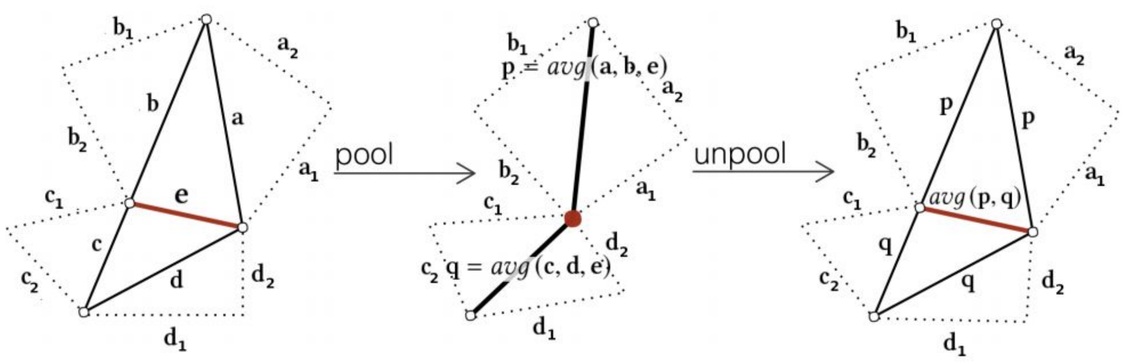

Figure 1: Illustration of the MeshCNN pooling and unpooling operation, where the edge with the minimum activation is being collapsed.

In order to keep track of the collapsed edges we use a matrix ***G*** of size (N_p x N_q), where each entry $\mathbf{G}_{i,j}$ denotes if edge *j* has been collapsed into edge *i*. For large meshes, this is a very sparse matrix, and we thus implemented it as such. We also modified the pooling operation to make it be done in bulk, rather than iteratively, as was the original implementation made by the author.

$$\mathbf{s} = \begin{bmatrix}0\\1\\4\\3\end{bmatrix} \quad \mathbf{t} = \begin{bmatrix}3\\2\\0\\4\end{bmatrix} \quad \mathbf{G} = \begin{bmatrix}1&0&0&0&1\\0&1&0&0&0\\0&1&1&0&0\\1&0&0&1&0\\1&0&0&1&1\\&&&&0\end{bmatrix}$$

$$\mathbf{s} = \begin{bmatrix}0\\1\\4\\3\end{bmatrix} \quad \mathbf{t} = \begin{bmatrix}3\\2\\0\\4\end{bmatrix} \begin{matrix}\}\text{ Batch 1}\\ \\ \\ \text{Batch 2}\end{matrix}$$

Figure 2: Illustration of the MeshCNN pooling operation where the source edges, **s**, are collapsed into the target edges, **t**. In the matrix **G** row 4 it is seen why the pooling operation cannot be fully vectorised. We instead present a batch operation which makes the operation more efficient.

Data

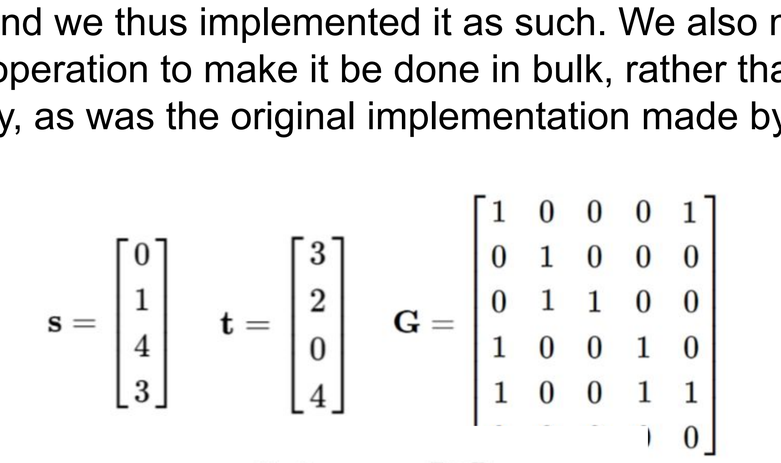

Figure 3: Left Illustration of the heart with the left atrial appendage marked. **Right** Our training data i.e. the left atrium cropped out with the left atrial appendage segmented.

Figure 4: Distribution of number of vertices in the meshes from the training split.

Model

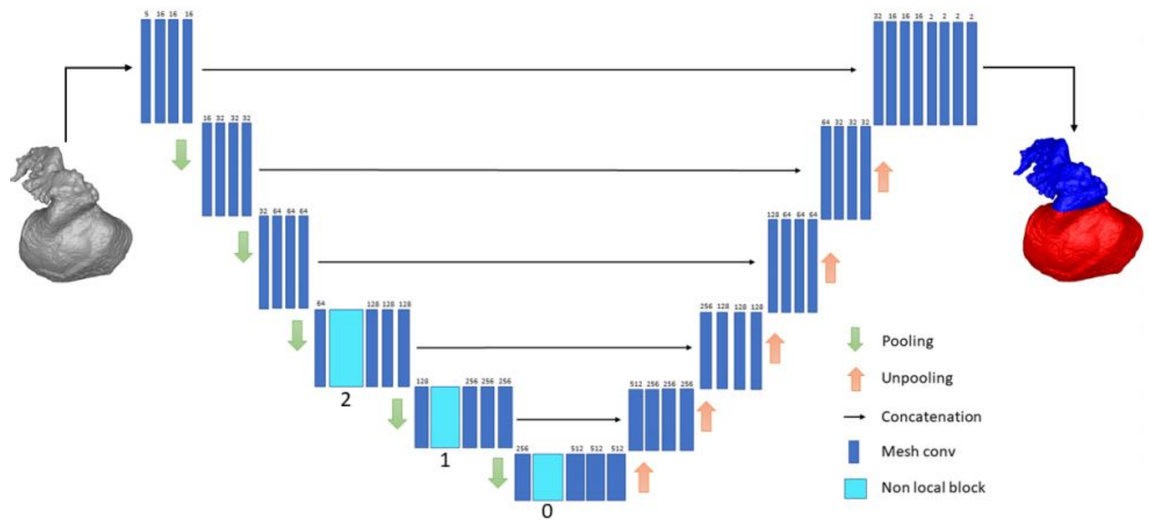

Figure 5: The used model architecture for the model with attention. The other models are based on similar U-net structure. Ideally we would have the attention block before the first pooling but due to memory constraints it is not possible on a consumer grade GPU.

Results

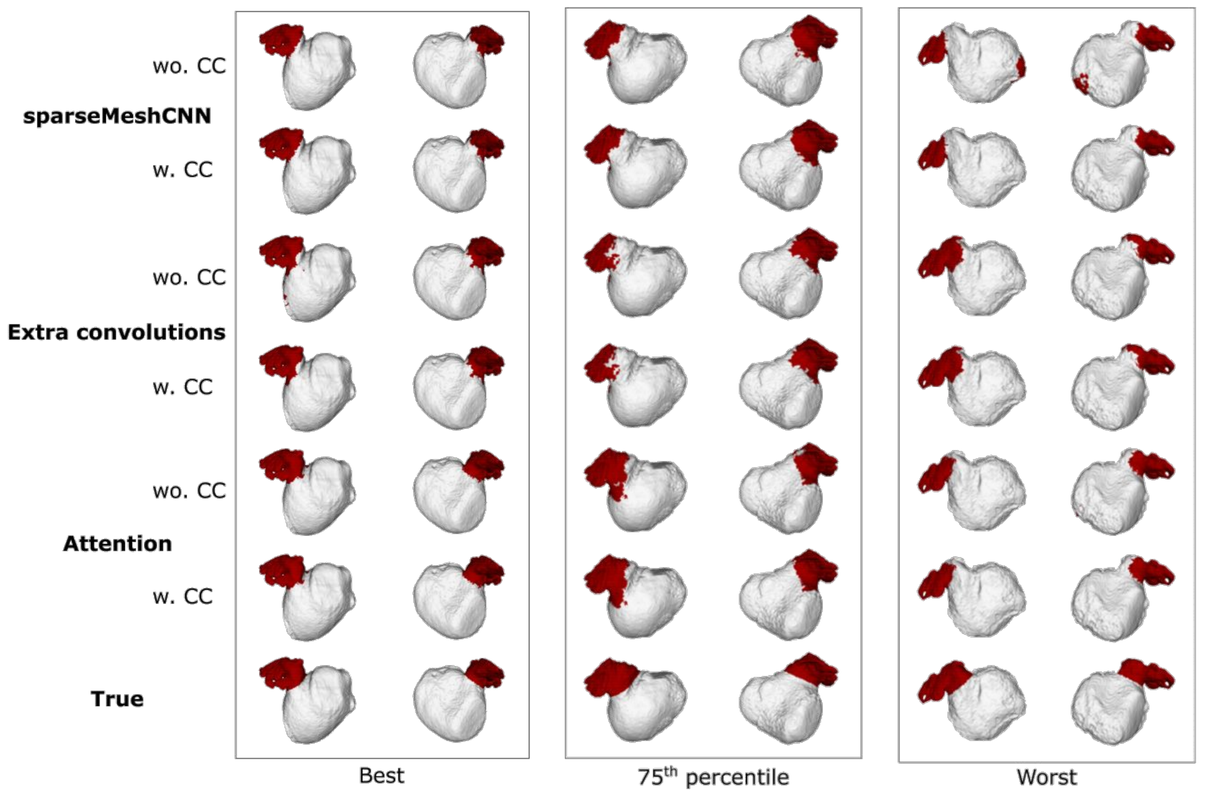

Figure 6: Results on the meshes with best, 75th percentile and worst segmentation results evaluated by Dice-score. It can be seen that the models with larger receptive field or attention manages to only segment the LAA without any post processing.

Acknowledgements

This poster presents the work of Mathias Lowes[1], Bjørn Hansen[1], Anders Dahl[1], Vedrana Dahl[1], Ole de Backer[3], Oscar Camara[2]. Rasmus Paulsen[1], Christian Ingwersen[1 4] and Kristine Sørensen[1]

[1]Visual Computing, Technical University of Denmark: s183983,s183986,abda,vand,rapa,ckin,kajul@dtu.dk
[2]Universitat Pompeu Fabra, Spain: oscar.camara@upf.edu
[3]Rigshopitalet, Denmark: ole.de.backer@regionh.dk
[4]TrackMan A/S, Denmark: cin@trackman.com