MeshCNN

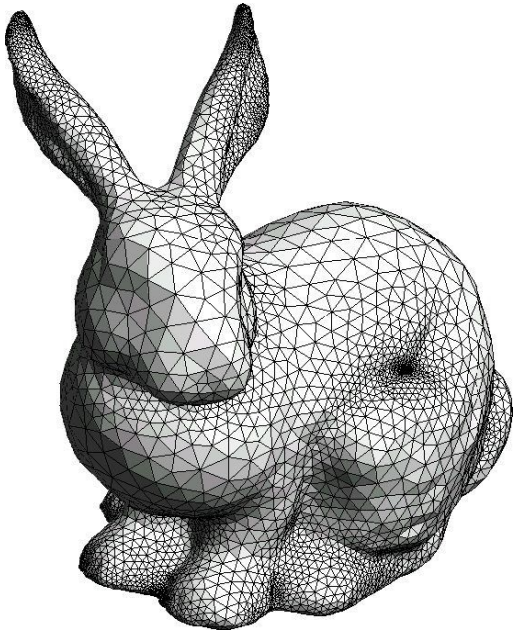
A Network with an Edge

Authors - Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, Daniel Cohen-Or

Presented By - Aradhya

Meshes

- Efficient
- Non - Uniform
- Small number of polygons can represent large, simple surfaces
- Captures connectivity information
- Captures topology of a surface: faithfully describing intricate structures while disambiguating proximity from nearby surfaces



Challenges

Images are represented on a regular grid of values, doing the same for meshes which have irregular structures is nontrivial.

How to account for local structure ?

How to capture the global structure ?

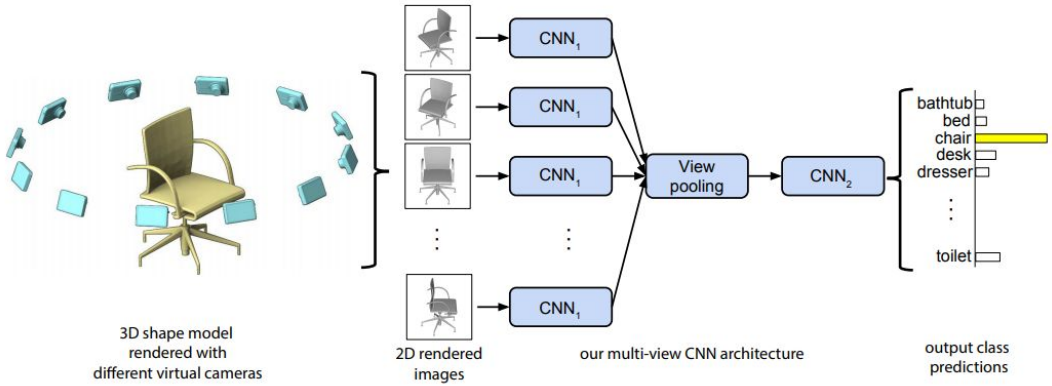
Combinatorial nature of adjacency matrix?

Defining operations on Graphs?

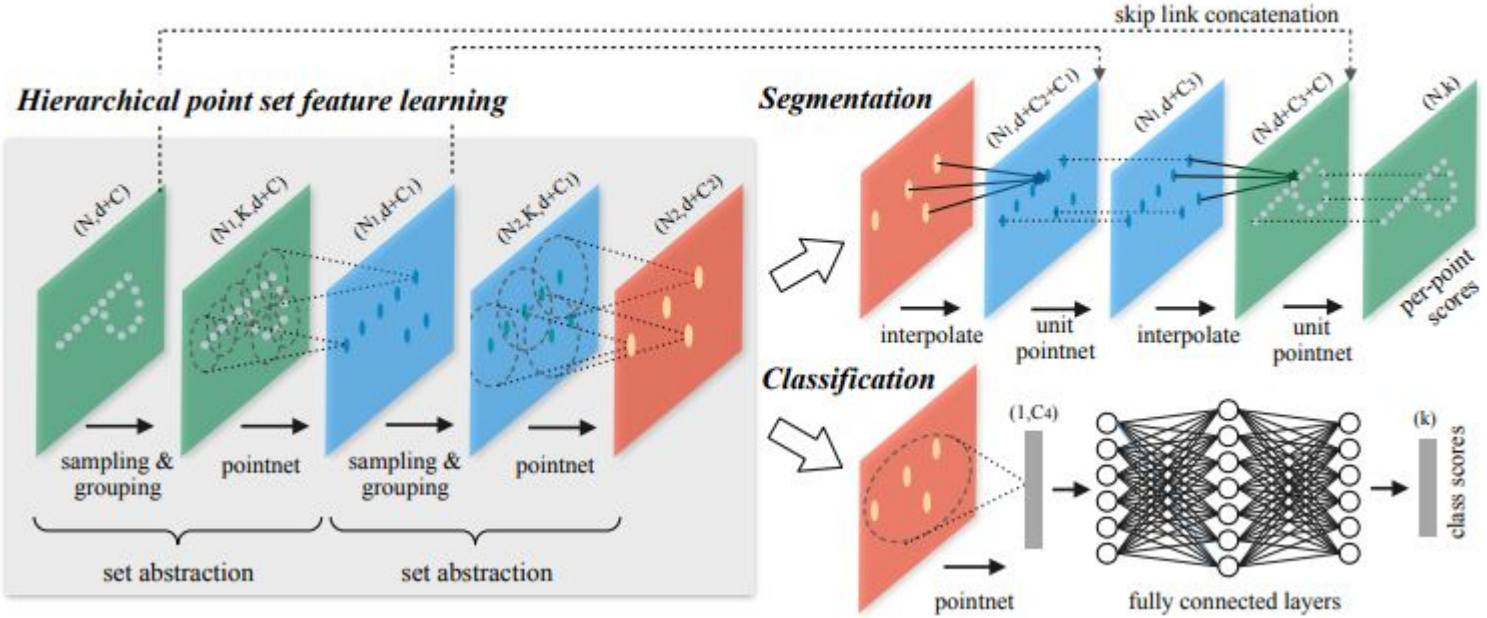
Previous Methods

- Point clouds
- Voxel based representation
- Image based
- Graph Based

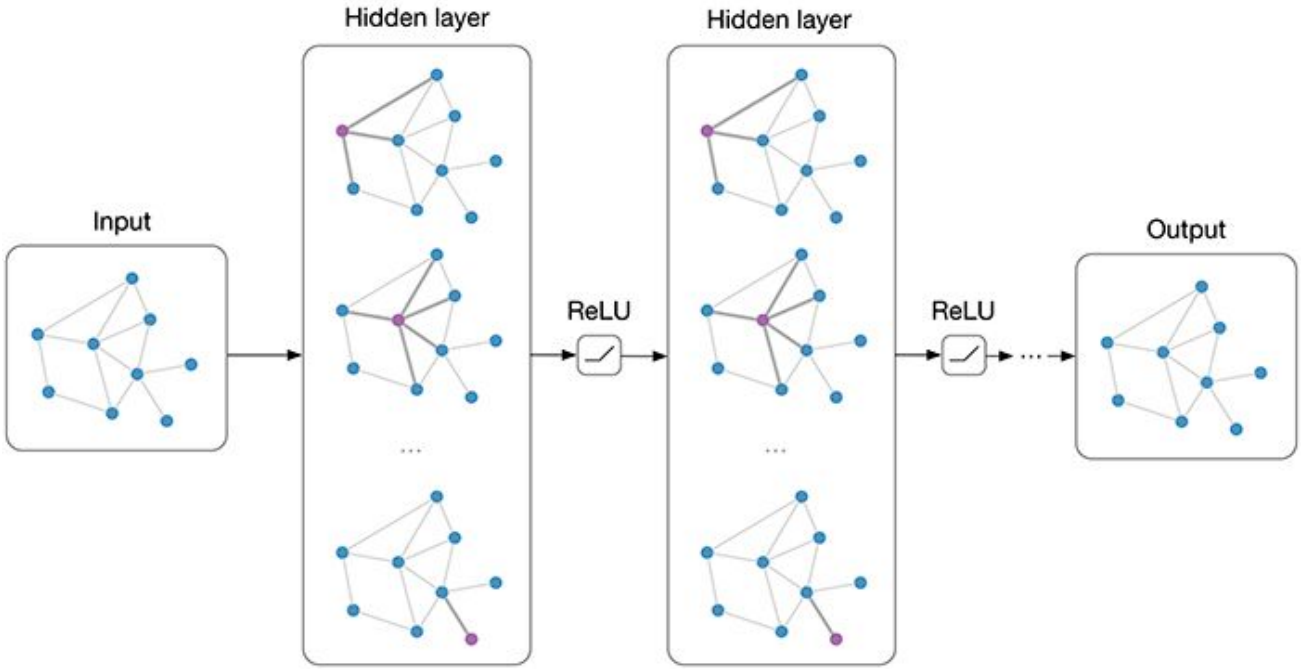
Multi-View Convolutional Neural Networks for 3D Shape Recognition



PointNet++: Deep Hierarchical Feature Learning on Point Sets



Graph Neural Network



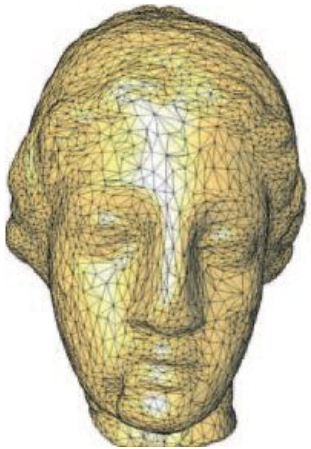
Mesh CNN

Enables performing convolutions and pooling operations with unique mesh properties.

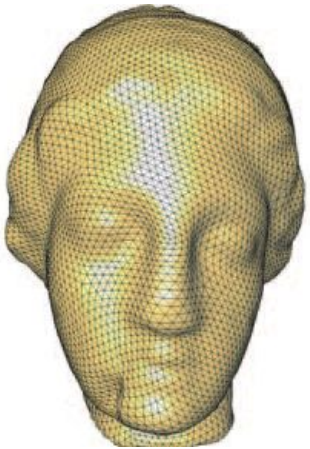
- Edges are analogous to pixels in an image
- Edges are the basic building blocks on which all operations are performed.
- Every edge are incident to exactly two faces, hence a fixed neighbourhood of 4 edges
- Utilize consistent face normal order to apply a symmetric convolution operation, which learns features invariant to rotation, scale and translation.

Kind of meshes

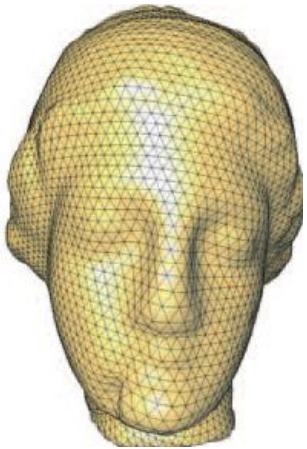
- Irregular and non-uniform meshes
- Manifold Mesh



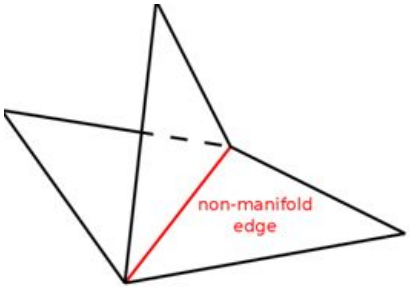
a) Irregular

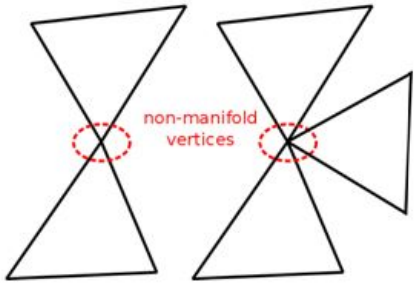


b) Semi-Irregular/Regular



c) Regular

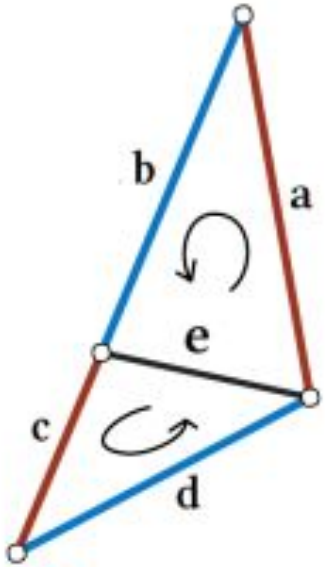




Invariant Convolution

The vertices of a face are ordered counter-clockwise, defining two possible orderings for the four neighbors of every edge.

the 1-ring neighbors of e can be ordered as (a, b, c, d) or (c, d, a, b) , depending on which face defines the first neighbor. This ambiguates the convolutional receptive field, hindering the formation of invariant features.



Convolution invariance to ordering

In order to guarantee the invariance to ordering they apply simple symmetric functions to the ambiguous pairs. The new set of convolutional neighbors that are guaranteed to be invariant.

$$(e^1, e^2, e^3, e^4) = (|a - c|, a + c, |b - d|, b + d)$$

Invariance to Similarity Transform

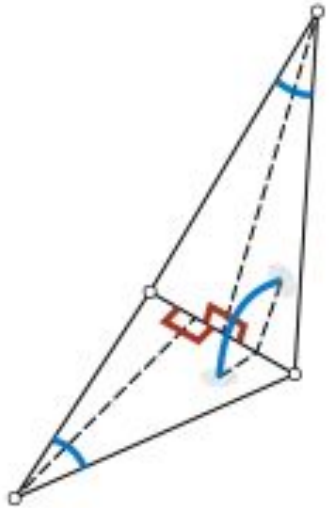
- Design input descriptor of an edge to contain only relative geometric features that are inherently invariant to similarity transformations.
- Aggregation of four 1-ring edges into two pairs of edges which have an ambiguity (eg a,c & b,d) and generate new features by simple symmetric functions on each pair. (eg. sum (a,c))

Convolution is then applied on the new symmetric features thus eliminating the ambiguity of order.

Input Features

The input edge feature is a 5-dimensional vector for every edge:

- the dihedral angle, two inner angles and two edge-length ratios for each face.
- The edge ratio is between the length of the edge and the perpendicular (dotted) line for each adjacent face.
- We sort each of the two face-based features (inner angles and edge-length ratios), thereby resolving the ordering ambiguity and guaranteeing invariance.
- Observe that these features are all relative, making them invariant to translation, rotation and uniform scale.



Global Ordering

- The global ordering of the edges is the order in which the edge data (input features) of a particular shape enters the network.

- This ordering has no influence during the convolutional stage since convolution is performed within local neighborhoods.

- For tasks that require global feature aggregation, such as classification, we follow the common practice suggested by Qi et al. [2017a] in PointNet, and place a global average pooling layer that connects between the convolutional part and the fully-connected part of the network.

Mesh Convolution

The convolution operator for edges is defined as the dot product between the kernel k and a neighborhood, thus the convolution for an edge feature e and its four adjacent edges

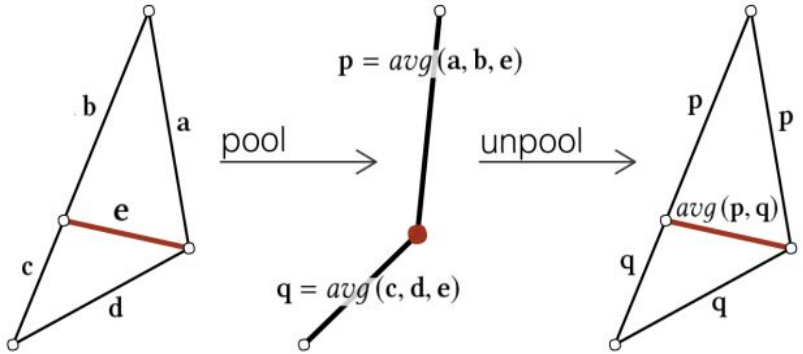
$$e \cdot k_0 + \sum_{j=1}^{4} k_j \cdot e^j$$

Mesh Pooling

- Mesh pooling operates on irregular structures and spatially adapts to the task.
- Unlike normal edge collapse operation which can be used as an alternative to pooling the authors propose task-specific edge collapse.
- The collapsed edges are the ones that least contribute to the objective.
- Since there can be combinations of edges, pooling simplifies to predetermined constant number of edges.

Mesh Pooling

5 edges are collapsed into 2 edges, the operation is prioritised by the norm of edge features, thereby allowing the network to select which parts of mesh to simplify and which to leave intact. Thus creating a task aware.



Mesh Pooling

Conventional pooling is extended to irregular data by identifying 3 core operations that help generalize the notion of pooling:

- Define pooling region given adjacency
- Merge features in each pooling region
- Redefine adjacency for the merged features

Mesh pooling is another case of generalized pooling where adjacency is determined from the topology.

Mesh Pooling

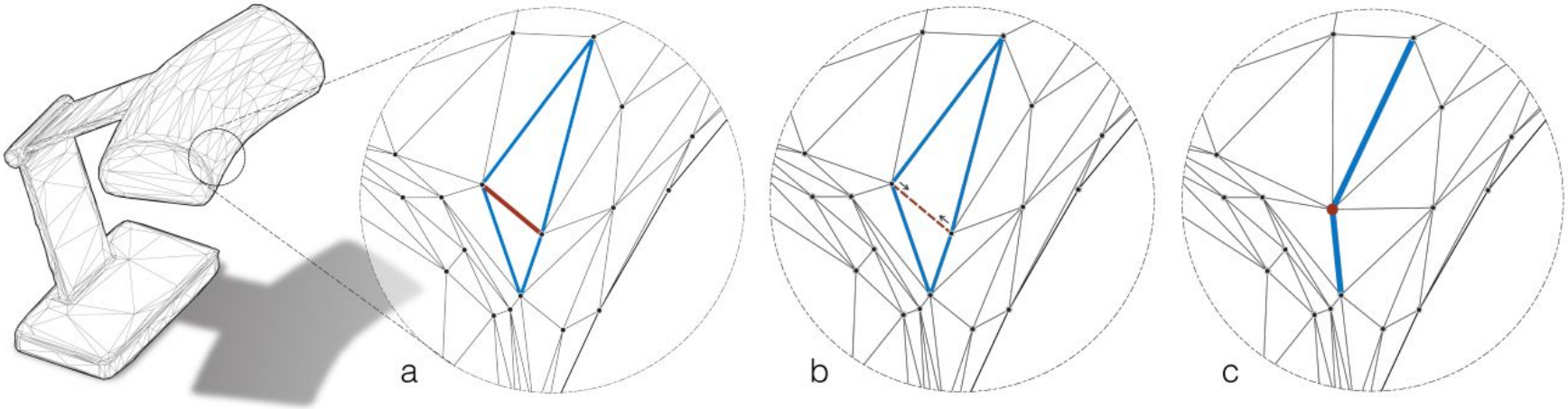
The edge magnitude based collapsing allows for non-uniformly collapse certain regions which are least important for the loss.

Each face contains three edges: the minimum edge and two adjacent neighbors of the minimum edge.

Each of the features of the three edges in each face are merged into a new edge feature by taking the average over the feature channel.

The strength of the feature of the edge is taken as the L2-norm.

Note: Edge collapse yielding to non-manifold face is not allowed it would violate the four convolutional neighbors assumption.



Features are computed on the edges by applying convolutions with neighborhoods made up of the four edges of the two incident triangles of an edge, b) and c) show the pooling step

Mesh Unpooling

- Pooling operation records the history from the merge operations and uses them to expand feature activation. Thus, unpooling does not have learnable parameters and is combined with convolutions to recover the original resolution lost in the pooling operation.
- Mesh unpooling layer reinstates the upsampled topology by storing the connectivity prior to pooling.
- Each unpooled edge feature is then a weighted combination of the pooled edge features.

Data Processing for experiments

- simplified each mesh to roughly the same number of edges. Note that as mentioned earlier, Mesh-CNN does not require the same number of edges across all samples.
- geometric mesh decimation helps to reduce the input resolution and with it the network capacity required for training.

- the classification task learns a global shape description, we usually use a lower resolution (750 edges)
- compared with the segmentation task (2250 edges)

Augmentation

since our input features are similarity-invariant, applying rotation, translation and isotropic scaling (same in x, y and z) does not generate new input features.

Apply anisotropic scaling on the vertex locations in x, y and z, < S x , S y , S z >which will change the input features to the network. We also shift vertices to different locations on the mesh surface.

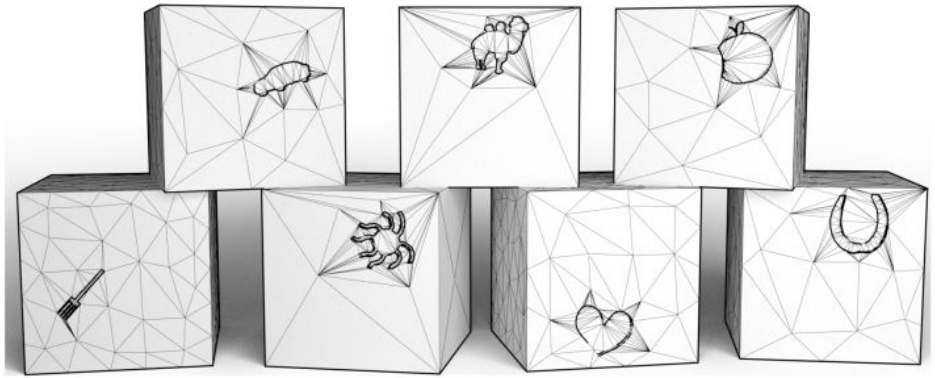
Experiments

Engraved cube classification

Accuracy : 91%

Number of epochs: 200

Number of classes: 30

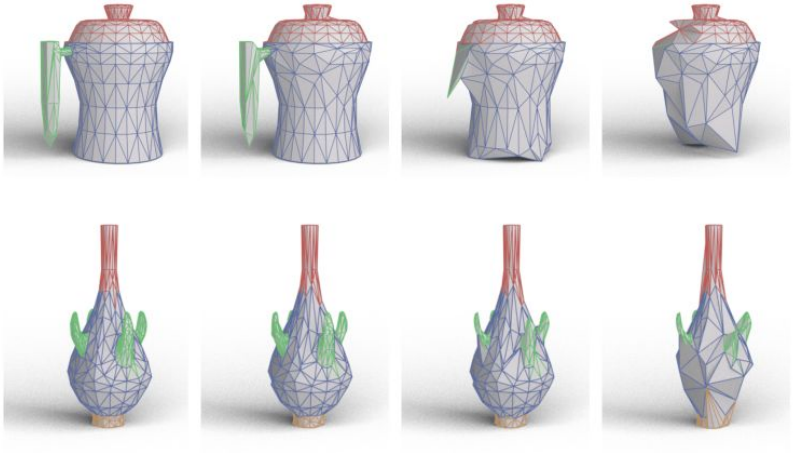


Experiments

COSEG Segmentation

evaluate the performance of MeshCNN on the task of segmentation on the COSEG dataset, which contains three large sets:

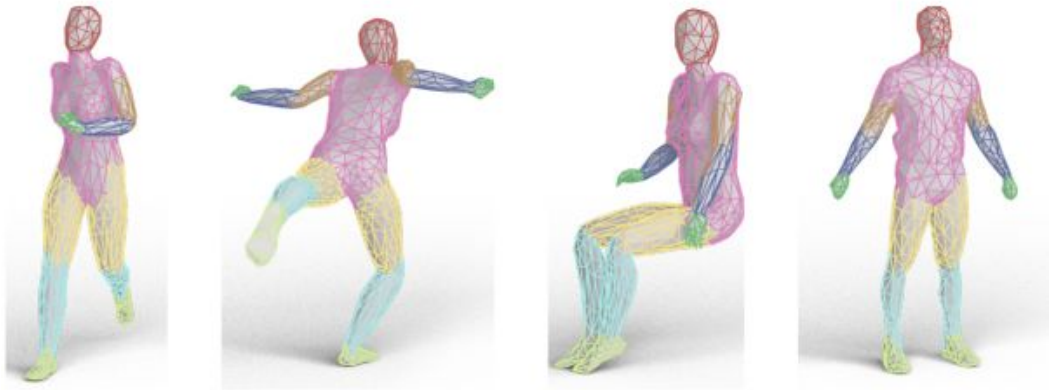
aliens, vases and chairs containing 200, 300 and 400 models in each respectively, obtained accuracies of 97% 99.6% and 97% respectively.



Experiments

Human shape segmentation

MeshCNN 92.30%



Code [source:github official repo]:

```python
def merge_vertices(self, edge_id):
    self.remove_edge(edge_id)
    edge = self.edges[edge_id]
    v_a = self.vs[edge[0]]
    v_b = self.vs[edge[1]]
    # update pA
    v_a.__iadd__(v_b)
    v_a.__itruediv__(2)
    self.v_mask[edge[1]] = False
    mask = self.edges == edge[1]
    self.ve[edge[0]].extend(self.ve[edge[1]])
    self.edges[mask] = edge[0]
```

```python
def remove_edge(self, edge_id):
    vs = self.edges[edge_id]
    for v in vs:
        if edge_id not in self.ve[v]:
            print(self.ve[v])
            print(self.filename)
        self.ve[v].remove(edge_id)
```

Equivariant convolution [github repo link]

```
# apply the symmetric functions for an equivariant conv
x_1 = f[:, :, :, 1] + f[:, :, :, 3]
x_2 = f[:, :, :, 2] + f[:, :, :, 4]
x_3 = torch.abs(f[:, :, :, 1] - f[:, :, :, 3])
x_4 = torch.abs(f[:, :, :, 2] - f[:, :, :, 4])
f = torch.stack([f[:, :, :, 0], x_1, x_2, x_3, x_4], dim=3)
return f
```

Thank You