# Mesh R-CNN Literature Review

Cam2 Tensorflow Team | Author: Zach Ghera

**Model Name:** Mesh R-CNN
**Development Year:** 2019
**Authors Contact Information:**


| | |
|---|---|
| Georgia Gkioxari | georgia.gkioxari@gmail.com |
| Jitendra Malik | malik@eecs.berkeley.edu |
| Justin Johnson | justincj@umich.edu |


**Model Purpose**: 3D Shape Prediction from 2D RGB images.

**Is the code available? Where?** : (use https://paperswithcode.com)

- https://github.com/facebookresearch/meshrcnn
- https://github.com/IMAC-projects/style-transfer (not by original creator)

**Library Used to Develop Model**: Pytorch

**Networks Used/Referenced**: (Is another network referenced by name or used for construction? (ex: ResNet, and NAS is referenced in SpineNet paper) )

- **Mask R-CNN**: Built on top of. Mesh R-CNN contains the same box and mask branches/heads. These box and mask losses are also added to the voxel and mesh losses to train the network.
- **ResNet-50-FPN:** Backbone CNN.
- **Fully convolutional networks (FCNs):** Used in the voxel prediction branch.
- **Pixel2Mesh**: Network that inspired the mesh refinement branch.
- **Spatial Transformer Networks**: Used in the Vertex Alignment operation.
- **Semi-Supervised Classification with Graph Convolutional Networks**: Graph convolution operation used in the mesh refinement branch.
- **A pointset generation network for 3d object reconstruction from a single image:** Used as a standard for scaling ground-truth meshes in the ShapeNet model evaluation.

## Model Design

**Model Architecture**: (image and description)

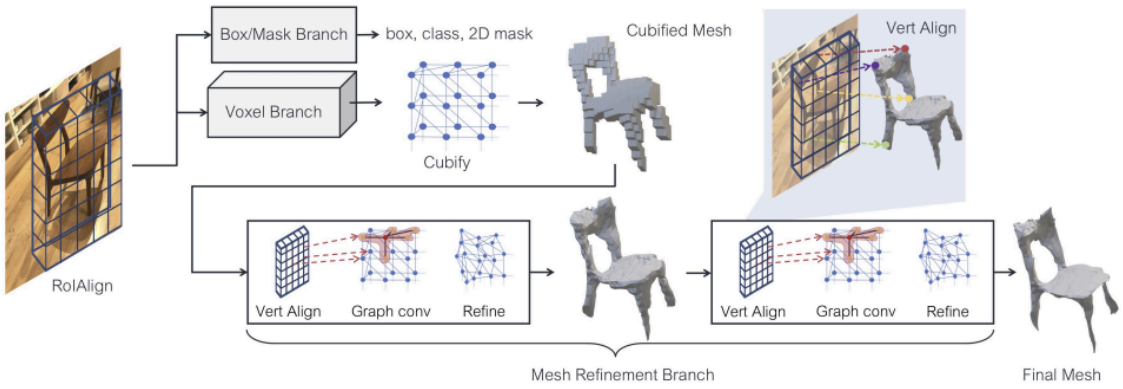


*Fig. 1: System Overview of Mesh R-CNN*

Mesh R-CNN reuses the same backbone network (ResNet-50-FPN), region proposal network (RPN), RoIAlign, as well as the box and mask branches from Mask-RCNN to produce a class label, bounding box, and instance segmentation mask. Mesh R-CNN directly extends this model with a novel branch to infer 3D meshes/shapes. This mesh branch network is composed of three primary sub-networks. A voxel branch that predicts a 3D voxel representation of the object, an operation called Cubify that

converts the voxel grid into a coarse 3D triangle mesh, and finally a mesh refinement branch that refines the vertex positions of the initial mesh to more accurately model the fine structure of the object. The *Model Sub-Networks* section provides additional details on the voxel prediction and mesh refinement branches/sub-networks.

The mesh prediction branch of Mesh R-CNN is validated on the ShapeNet dataset. Additionally, the full Mesh R-CNN model is evaluated on the Pix3D dataset to both detect objects and infer their 3D shape. While the high-level architecture of Mesh R-CNN is the same for evaluating ShapeNet and Pix3D, two different Mesh R-CNN models exist to handle each task in an effort to mitigate memory constraints. See Fig. 2 and 3 below for the model architecture of Mesh R-CNN used for evaluating on the Pix3D and ShapeNet datasets, respectively.

| Index | Inputs | Operation | Output shape |
|---|---|---|---|
| (1) | Input | Input Image | $H \times W \times 3$ |
| (2) | (1) | Backbone: ResNet-50-FPN | $h \times w \times 256$ |
| (3) | (2) | RPN | $h \times w \times A \times 4$ |
| (4) | (2),(3) | `RoIAlign` | $14 \times 14 \times 256$ |
| (5) | (4) | Box branch: $2\times$ downsample, Flatten, Linear$(7 * 7 * 256 \rightarrow 1024)$, Linear$(1024 \rightarrow 5C)$ | $C \times 5$ |
| (6) | (4) | Mask branch: $4\times$ Conv$(256 \rightarrow 256, 3 \times 3)$, TConv$(256 \rightarrow 256, 2 \times 2, 2)$, Conv$(256 \rightarrow C, 1 \times 1)$ | $28 \times 28 \times C$ |
| (7) | (2), (3) | `RoIAlign` | $12 \times 12 \times 256$ |
| (8) | (7) | Voxel Branch | $24 \times 24 \times 24$ |
| (9) | (8) | `cubify` | $\|V\| \times 3, \|F\| \times 3$ |
| (10) | (7), (9) | Refinement Stage 1 | $\|V\| \times 3, \|F\| \times 3$ |
| (11) | (7), (10) | Refinement Stage 2 | $\|V\| \times 3, \|F\| \times 3$ |
| (12) | (7), (11) | Refinement Stage 3 | $\|V\| \times 3, \|F\| \times 3$ |

*Fig. 2: Mesh R-CNN Pix3D Model Architecture*

| Index | Inputs | Operation | Output shape |
|---|---|---|---|
| (1) | Input | Image | $137 \times 137 \times 3$ |
| (2) | (1) | ResNet-50 `conv2_3` | $35 \times 35 \times 256$ |
| (3) | (2) | ResNet-50 `conv3_4` | $18 \times 18 \times 512$ |
| (4) | (3) | ResNet-50 `conv4_6` | $9 \times 9 \times 1024$ |
| (5) | (4) | ResNet-50 `conv5_3` | $5 \times 5 \times 2048$ |
| (6) | (5) | Bilinear interpolation | $24 \times 24 \times 2048$ |
| (7) | (6) | Voxel Branch | $48 \times 48 \times 48$ |
| (8) | (7) | `cubify` | $\|V\| \times 3, \|F\| \times 3$ |
| (9) | (2), (3), (4), (5), (8) | Refinement Stage 1 | $\|V\| \times 3, \|F\| \times 3$ |
| (10) | (2), (3), (4), (5), (9) | Refinement Stage 2 | $\|V\| \times 3, \|F\| \times 3$ |
| (11) | (2), (3), (4), (5), (10) | Refinement Stage 3 | $\|V\| \times 3, \|F\| \times 3$ |

*Fig. 3: Mesh R-CNN ShapeNet Model Architecture*

The main difference between the models is the existence of the RPN, RoIAlign, Box Branch and Mask branch in the Pix3D model but not in the ShapeNet model. This was done because there is only one object per sample in the ShapeNet dataset and the only evaluation metric is the comparison of the predicted 3D mesh and the ground truth shape. Thus, there is no need to produce bounding boxes and instance segmentation masks nor generate region proposals for the voxel branch.

Differences between the sub-branches of the mesh prediction branch are detailed in the *Model Sub-Networks* section.

**Model Sub-Networks:** (defined structures like backbone, or encoder that can be built independently because they serve a specific purpose)

Sub-Networks re-used from Mask-RCNN:

- Feature extraction backbone -- ResNet-50-FPN
- Region Proposal Network
- Box branch
- Mask branch

Voxel Prediction Branch: Predicts a 3D voxel grid of occupancy probabilities. This visually represents a coarse 3D shape of each detected object.

The voxel branch architecture is given in Fig. 4. For ShapeNet V = 48 and for Pix3D V = 24 as the Pix3D model has greater memory constraints. TConv is a transpose convolution with stride 2.

| Index | Inputs | Operation | Output shape |
|---|---|---|---|
| (1) | Input | Image features | $V/2 \times V/2 \times D$ |
| (2) | (1) | Conv$(D \rightarrow 256, 3 \times 3)$, ReLU | $V/2 \times V/2 \times 256$ |
| (3) | (2) | Conv$(256 \rightarrow 256, 3 \times 3)$, ReLU | $V/2 \times V/2 \times 256$ |
| (4) | (3) | TConv$(256 \rightarrow 256, 2 \times 2, 2)$, ReLU | $V \times V \times 256$ |
| (5) | (4) | Conv$(256 \rightarrow V, 1 \times 1)$ | $V \times V \times V$ |

*Fig. 4: Voxel Prediction Branch*

Mesh Refinement Branch: Refines the vertex positions of the initial mesh to more accurately model the fine structure of the object. This process primarily consists of three operations:

- Vertex Alignment: Project vertices in current mesh onto the image plane, then samples a feature from the image plane corresponding to each vertex's position to obtain an image-aligned feature vector for each mesh vertex. See VertAlign in the *Model Custom Layers, or Activations* section for more details.
- Graph Convolution: Updates vertex feature vectors by propagating information along neighboring mesh edges for each vertex. See GraphConv in the Model *Custom Layers, or Activations* section for more details.
- Vertex Refinement: Computes updated vertex positions from the new feature vectors produced by the previous sequence of graph convolutions.

There are 3 versions of the mesh refinement branch. The mesh refinement stage for the Pix3D model is given by Fig. 5 while the ShapeNet model has two possible configurations (see Fig. 6 and Fig. 7). The Pix3D mesh refinement branch is similar to the non-residual Shapenet branch except for not having to project from a large concatenate output layer after the final VertAlign and Concatenate operation. There is also one additional concatenate layer after the final GraphConv layer in Fig. 5 (index 11).

| Index | Inputs | Operation | Output shape |
|---|---|---|---|
| (1) | Input | Backbone features | $h \times w \times 256$ |
| (2) | Input | Input vertex features | $\|V\| \times 128$ |
| (3) | Input | Input vertex positions | $\|V\| \times 3$ |
| (4) | (1), (3) | `VertAlign` | $\|V\| \times 256$ |
| (5) | (2), (3), (4) | Concatenate | $\|V\| \times 387$ |
| (6) | (5) | GraphConv$(387 \rightarrow 128)$ | $\|V\| \times 128$ |
| (7) | (3), (6) | Concatenate | $\|V\| \times 131$ |
| (8) | (7) | GraphConv$(131 \rightarrow 128)$ | $\|V\| \times 128$ |
| (9) | (3), (8) | Concatenate | $\|V\| \times 131$ |
| (10) | (9) | GraphConv$(131 \rightarrow 128)$ | $\|V\| \times 128$ |
| (11) | (3), (10) | Concatenate | $\|V\| \times 131$ |
| (12) | (11) | Linear$(131 \rightarrow 3)$ | $\|V\| \times 3$ |
| (13) | (12) | Tanh | $\|V\| \times 3$ |
| (14) | (3), (13) | Addition | $\|V\| \times 3$ |

*Fig. 5: Pix3D Single Mesh Refinement Stage*

The full model for ShapeNet (as well as Pixel2Mesh+ and Shpere-Init baselines) use the residual mesh refinement stage in Fig. 6. There is also a shallower design with half the GraphConv layers given by Fig. 7. These architectures were found to have similar performance on ShapeNet.

| Index | Inputs | Operation | Output shape |
|---|---|---|---|
| (1) | Input | conv2_3 features | $35 \times 35 \times 256$ |
| (2) | Input | conv3_4 features | $18 \times 18 \times 512$ |
| (3) | Input | conv4_6 features | $9 \times 9 \times 1024$ |
| (4) | Input | conv5_3 features | $5 \times 5 \times 2048$ |
| (5) | Input | Input vertex features | $\|V\| \times 128$ |
| (6) | Input | Input vertex positions | $\|V\| \times 3$ |
| (7) | (1), (6) | VertAlign | $\|V\| \times 256$ |
| (8) | (2), (6) | VertAlign | $\|V\| \times 512$ |
| (9) | (3), (6) | VertAlign | $\|V\| \times 1024$ |
| (10) | (4), (6) | VertAlign | $\|V\| \times 2048$ |
| (11) | (7),(8),(9),(10) | Concatenate | $\|V\| \times 3840$ |
| (12) | (11) | Linear($3840 \rightarrow 128$) | $\|V\| \times 128$ |
| (13) | (5), (6), (12) | Concatenate | $\|V\| \times 259$ |
| (14) | (13) | ResGraphConv($259 \rightarrow 128$) | $\|V\| \times 128$ |
| (15) | (14) | $2\times$ ResGraphConv($128 \rightarrow 128$) | $\|V\| \times 128$ |
| (16) | (15) | GraphConv($128 \rightarrow 3$) | $\|V\| \times 3$ |
| (17) | (16) | Tanh | $\|V\| \times 3$ |
| (18) | (6), (17) | Addition | $\|V\| \times 3$ |

*Fig. 6: ShapeNet Single Residual Mesh Refinement Stage*

| Index | Inputs | Operation | Output shape |
|---|---|---|---|
| (1) | Input | conv2_3 features | $35 \times 35 \times 256$ |
| (2) | Input | conv3_4 features | $18 \times 18 \times 512$ |
| (3) | Input | conv4_6 features | $9 \times 9 \times 1024$ |
| (4) | Input | conv5_3 features | $5 \times 5 \times 2048$ |
| (5) | Input | Input vertex features | $\|V\| \times 128$ |
| (6) | Input | Input vertex positions | $\|V\| \times 3$ |
| (7) | (1), (6) | VertAlign | $\|V\| \times 256$ |
| (8) | (2), (6) | VertAlign | $\|V\| \times 512$ |
| (9) | (3), (6) | VertAlign | $\|V\| \times 1024$ |
| (10) | (4), (6) | VertAlign | $\|V\| \times 2048$ |
| (11) | (7),(8),(9),(10) | Concatenate | $\|V\| \times 3840$ |
| (12) | (11) | Linear($3840 \rightarrow 128$) | $\|V\| \times 128$ |
| (13) | (5), (6), (12) | Concatenate | $\|V\| \times 259$ |
| (14) | (13) | GraphConv($259 \rightarrow 128$) | $\|V\| \times 128$ |
| (15) | (6), (14) | Concatenate | $\|V\| \times 131$ |
| (16) | (15) | GraphConv($131 \rightarrow 128$) | $\|V\| \times 128$ |
| (17) | (6), (16) | Concatenate | $\|V\| \times 131$ |
| (18) | (17) | GraphConv($131 \rightarrow 128$) | $\|V\| \times 128$ |
| (19) | (18) | Linear($128 \rightarrow 3$) | $\|V\| \times 3$ |
| (20) | (19) | Tanh | $\|V\| \times 3$ |
| (21) | (6), (20) | Addition | $\|V\| \times 3$ |

*Fig. 7: ShapeNet Non-residual Mesh Refinement Stage*

**Model Sub-Network Flow/Connections:** See Fig. 1.

**Model Building Blocks/Repeated Structures:** (found by analyzing model architectures above, indicate where you found each block)

The only repeated layer structure that is not already formalized by Mesh R-CNN is the Concatenate / GraphConv layers in the non-residual version of the mesh refinement stage (see Fig. 5 and Fig. 7).

**Model Custom Layers, or Activations:** (layers not included in tensorflow, but required to build Building blocks. (Ex. "attention layer transformers"))

- RoIAlign: Re-used from in Mask R-CNN.
- Cubify: Takes as input the 3D voxel grid with occupancy probabilities (output of voxel branch) and a threshold value. Each voxel in the grid is replaced with a "water-tight cuboid triangle mesh with 8 vertices, 18 edges,and 12 faces. Shared vertices and edges between adjacent occupied voxels are merged, and shared interior faces are eliminated.

  The cubify algorithm is provided as seen in Algorithm 1 below. This implementation is inefficient as it uses nested for loops. The authors implement the same operations with 3D convolutions and vectorize other computations which results in an approximately 10x speedup.

- VertAlign: While Mesh R-CNN explained the VertAlign operation at a high level, *Pixel2Mesh*'s description of the "Perceptual feature pooling layer" was much more clear and descriptive. For each of the 3D vertices, we do the following:
  - Project the vertex onto the input image plane (using camera intrinsics).
  - In the image space, we obtain (num channels/kernels) dimension vector for each of the 'conv3_3', 'conv4_3', and 'conv5_3' feature maps where that vector is obtained through bilinear interpolation of the four closest pixels to the projected vertex pixel in that feature map (like ROI Align). See Fig. 7.

**Data:** $V : [0, 1]^{N \times D \times H \times W}$, $\tau \in [0, 1]$
unit_cube $= (V_{\text{cube}}, F_{\text{cube}})$
**for** *(n, z, y, x)* $\in$ *range(N, D, H, W)* **do**
  **if** $V[n, z, y, x] > \tau$ **then**
    add unit_cube at $(n, z, y, x)$
    **if** $V[n, z - 1, y, x] > \tau$ **then**
      remove back faces
    **end**
    **if** $V[n, z + 1, y, x] > \tau$ **then**
      remove front faces
    **end**
    **if** $V[n, z, y - 1, x] > \tau$ **then**
      remove top faces
    **end**
    **if** $V[n, z, y + 1, x] > \tau$ **then**
      remove bottom faces
    **end**
    **if** $V[n, z, y, x - 1] > \tau$ **then**
      remove left faces
    **end**
    **if** $V[n, z, y, x + 1] > \tau$ **then**
      remove right faces
    **end**
  **end**
**end**
merge shared verts
**return** A list of meshes $\{T_i = (V_i, F_i)\}_{i=0}^{N-1}$

**Algorithm 1:** `Cubify`

- Concatenate the vectors obtained from the bilinear pooling in each feature map and then project that to a 128 dim vector that we call the “image aligned feature vector.”

In the first mesh refinement stage, for each vertex, this 128 dim feature vector is concatenated with the vertex position. In subsequent mesh refinement stages, these two vectors are also concatenated with the vertex feature vector calculated in the previous mesh refinement stage.

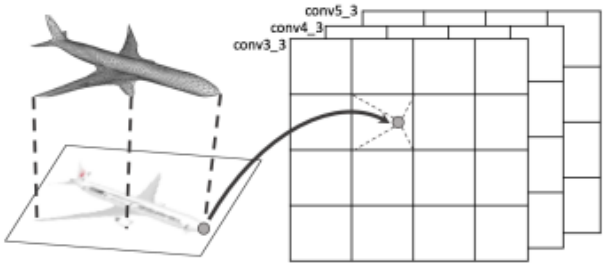


*Fig. 7: VertAlign Operation*

- GraphConv: The graph convolution operation is given by Equation 1 below. This computation is analogous to traditional 2D convolution as it captures relationships between local/neighboring regions of the input. In the case of a 3D triangle mesh, this means that the vertex feature vectors will be updated by combining information from the feature vectors of the neighboring vertices. More specifically, the new feature vector for the current vertex is the product of the current feature vector and a learned weight matrix added to the same product (learned weight matrix times feature vec) for all neighboring vertices.

$$f_i' = \mathrm{ReLU}\left(W_0 f_i + \sum_{j \in \mathcal{N}(i)} W_1 f_j\right)$$

*Equation 1: Graph Convolution*

**Model Loss Functions (custom or not):** (the equation if given, if not just note the loss function used by looking at paper or repo)

As stated in Mesh R-CNN, “ The voxel and mesh losses... are added to the box and mask losses and the whole system is trained end-to-end.”

Classification, Box, and Mask Loss for each RoI from Mask R-CNN are

- Classification and Bounding-Box Loss:

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1] L_{\text{loc}}(t^u, v)$$

where

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i),$$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

p and u are classification vectors. t and v are bounding boxes.

- Mask Loss: Average binary cross entropy loss on output of a per-pixel sigmoid where each RoI encodes K binary masks of resolution m×m, one for each of the K classes.

  For an RoI associated with ground-truth class k, this loss is only defined on the k-th mask.

The Voxel and Mesh losses are provided in Mesh R-CNN:

- Voxel Loss: Average binary cross-entropy between predicted voxel occupancy probabilities and true voxel occupancies.
- Mesh Loss: Average weighted sum of $\mathcal{L}_{\text{cham}}(P^i, P^{gt})$, $\mathcal{L}_{\text{norm}}(P^i, P^{gt})$ and $\mathcal{L}_{\text{edge}}(V^i, E^i)$ (see below).

  These mesh losses are based on a differentiable mesh sampling operation to sample points and their normal vectors uniformly from the surface of the ground truth and predicted meshes. Points are uniformly sampled from the surface of the mesh by sampling a face *f* = (v1, v2, v3) from the mesh where the probability distribution of faces defined as:

$$P(f) = \frac{area(f)}{\sum_{f' \in F} area(f')}$$

  Once a face is selected. We sample a point *p* uniformly from the interior of *f* by $p = \sum_i w_i v_i$ where

$$w_1 = 1 - \sqrt{\xi_1}, \quad w_2 = (1 - \xi_2)\sqrt{\xi_1}, w_3 = \xi_2\sqrt{\xi_1}, \text{ and } \xi_1, \xi_2 \sim U(0, 1)$$

  This use of the reparameterization trick (VAEs) allows gradients of the loss wrt. points *p* can propagate back to gradients wrt. the face vertices $v_i$.

  - Chamfer Distance: Goal is to penalize distance between points in the ground truth vs predicted.

$$\mathcal{L}_{\text{cham}}(P, Q) = |P|^{-1} \sum_{(p,q) \in \Lambda_{P,Q}} \|p - q\|^2 + |Q|^{-1} \sum_{(q,p) \in \Lambda_{Q,P}} \|q - p\|^2 \quad (1)$$

    …

  - Normal Distance: Goal is to penalize angle between normal vectors in the ground truth vs predicted.

$$\mathcal{L}_{\text{norm}}(P, Q) = -|P|^{-1} \sum_{(p,q) \in \Lambda_{P,Q}} |u_p \cdot u_q| - |Q|^{-1} \sum_{(q,p) \in \Lambda_{Q,P}} |u_q \cdot u_p|. \quad (2)$$

    …

  - Edge Loss: Goal is to regularize predicted mesh.

$$\mathcal{L}_{\text{edge}}(V, E) = \frac{1}{|E|} \sum_{(v,v') \in E} \|v - v'\|^2$$

    …

**Model Outputs Structure:** (outputs of each subnetwork and final output, output shape, and how information is organized)

See figures in *Model Architecture* and *Model Sub-Networks* sections.

## Training and Evaluation

**Datasets Used and Location/Availability:**

ShapeNet: https://shapenet.org/

Pix3D: http://pix3d.csail.mit.edu/

**Input Preprocessing Steps:** (the measures used to pre-process images prior to network input):

Dataset augmentation was not listed in the Mesh RCNN paper. However, I was able to get some information from looking at the official Pytorch implementation. The elements that I saw included in the data pipelines include:

- (Pix3D only) Parsing the COCO-format annotations in to the Detectron2 format
- Apply transformation including image resizing and flipping
- Shuffle and create the mini batches for training
- Normalize the mean and variance of images
- Project vertices onto image plane
- Compute voxels

**Training Steps and Parameters for Each Training Phase:**

ShapeNet

- 25 epochs with 32 images per batch
- Adam with learning rate 10−4
- Cubify threshold to 0.2 and weight the losses with λvoxel = 1, λcham = 1, λnorm = 0, and λedge = 0.2.

Pix3D

- 12 epochs with 64 images per batch
- SGD with momentum. Linearly increasing the learning rate from 0.002 to 0.02 over the first 1K iterations, then decaying by a factor of 10 at 8K and 10K iterations
- Initialize from a model pre--trained for instance segmentation on COCO
- Cubify threshold to 0.2 and weight the losses with λvoxel = 3, λcham = 1, λnorm = 0.1, and λedge = 1.
- Weight decay 10−4
- Detection loss weights are identical to Mask R-CNN

**Output Processing Steps:** N/a.

**Testing/Training Metrics and Callbacks:**

ShapeNet

- Chamfer distance:

$$\mathcal{L}_{\text{cham}}(P,Q) = |P|^{-1} \sum_{(p,q)\in\Lambda_{P,Q}} \|p-q\|^2 + |Q|^{-1} \sum_{(q,p)\in\Lambda_{Q,P}} \|q-p\|^2 \quad (1)$$

- Normal consistency: 1 - $L_{norm}$ where

$$\mathcal{L}_{\text{norm}}(P,Q) = -|P|^{-1} \sum_{(p,q)\in\Lambda_{P,Q}} |u_p \cdot u_q| - |Q|^{-1} \sum_{(q,p)\in\Lambda_{Q,P}} |u_q \cdot u_p|. \quad (2)$$

- $F1^T$, at various distance thresholds τ. Where $F1^T$ is the "harmonic mean of the precision at τ (fraction of predicted points within τ of a ground-truth point) and the recall at τ (fraction of ground truth points within τ of a predicted point)."

Pix3D

- AP_box and AP_mask at intersection-over-union (IoU) 0.5.
- AP_mesh: A mesh prediction is considered a true-positive if its predicted label is correct, it is not a duplicate detection, and the mean area under the per-category precision-recall curves for F1_0.3 > 0.5.

**Testing Steps:**

ShapeNet:

- Sample 10k points uniformly at random from the surface of predicted and ground-truth meshes, and use them to compute the metrics list in *Testing/Training Metrics and Callbacks* for ShapeNet above.
- Because Chamfer distance and $F1^T$ depend on the scale of the meshes, the meshes are scaled based on the testing procedure in *A pointset generation network for 3d object reconstruction from a single image* (longest edge of the
- ground-truth mesh's bounding box has length 10) and Pixel2Mesh (rescale by a factor of 0.57)

Pix3D

- Compare predicted and ground-truth meshes in the camera coordinate system assumed from known camera intrinsics for VertAlign.
- In addition to predicting the box of each object on the image plane, Mesh R-CNN predicts the depth extent by appending a 2-layer MLP head, similar to the box regressor head. As a result, Mesh RCNN predicts a 3D bounding box for each object (more details in Appendix section E).

**Target Metrics for Completion:** (the evaluation or testing stats that will indicate model building is complete)

ShapeNet

| | | Full Test Set | | | | | | | Holes Test Set | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Chamfer(↓) | Normal | $F1^{0.1}$ | $F1^{0.3}$ | $F1^{0.5}$ | $\|V\|$ | $\|F\|$ | Chamfer(↓) | Normal | $F1^{0.1}$ | $F1^{0.3}$ | $F1^{0.5}$ | $\|V\|$ | $\|F\|$ |
| | Pixel2Mesh [69]‡ | 0.205 | 0.736 | 33.7 | 80.9 | 91.7 | 2466±0 | 4928±0 | 0.272 | 0.689 | 31.5 | 75.9 | 87.9 | 2466±0 | 4928±0 |
| | Voxel-Only | 0.916 | 0.595 | 7.7 | 33.1 | 54.9 | 1987±936 | 3975±1876 | 0.760 | 0.592 | 8.2 | 35.7 | 59.5 | 2433±925 | 4877±1856 |
| Best | Sphere-Init | **0.132** | 0.711 | 38.3 | 86.5 | **95.1** | 2562±0 | 5120±0 | 0.138 | 0.705 | 40.0 | 85.4 | 94.3 | 2562±0 | 5120±0 |
| Best | Pixel2Mesh+ | **0.132** | 0.707 | 38.3 | 86.6 | **95.1** | 2562±0 | 5120±0 | 0.137 | 0.696 | 39.3 | 85.5 | 94.4 | 2562±0 | 5120±0 |
| Best | Ours (light) | 0.133 | 0.725 | **39.2** | **86.8** | **95.1** | 1894±925 | 3791±1855 | **0.130** | 0.723 | 41.6 | **86.7** | 94.8 | 2273±899 | 4560±1805 |
| Best | Ours | 0.133 | **0.729** | 38.8 | 86.6 | **95.1** | 1899±928 | 3800±1861 | **0.130** | **0.725** | **41.7** | **86.7** | **94.9** | 2291±903 | 4595±1814 |
| Pretty | Sphere-Init | 0.175 | 0.718 | 34.5 | 82.2 | 92.9 | 2562±0 | 5120±0 | 0.186 | 0.684 | 34.4 | 80.2 | 91.7 | 2562±0 | 5120±0 |
| Pretty | Pixel2Mesh+ | 0.175 | **0.727** | 34.9 | 82.3 | 92.9 | 2562±0 | 5120±0 | 0.196 | 0.685 | 34.4 | 79.9 | 91.4 | 2562±0 | 5120±0 |
| Pretty | Ours (light) | 0.176 | 0.699 | 34.8 | 82.4 | 93.1 | 1891±924 | 3785±1853 | 0.178 | 0.688 | 36.3 | 82.0 | 92.4 | 2281±895 | 4576±1798 |
| Pretty | Ours | **0.171** | 0.713 | **35.1** | **82.6** | **93.2** | 1896±928 | 3795±1861 | **0.171** | **0.700** | **37.1** | **82.4** | **92.7** | 2292±902 | 4598±1812 |

Pix3D

| Pix3D $\mathcal{S}_1$ | $AP^{box}$ | $AP^{mask}$ | $AP^{mesh}$ | *chair* | *sofa* | *table* | *bed* | *desk* | *bkcs* | *wrdrb* | *tool* | *misc* | $\|V\|$ | $\|F\|$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Voxel-Only | 94.4 | 88.4 | 5.3 | 0.0 | 3.5 | 2.6 | 0.5 | 0.7 | 34.3 | 5.7 | 0.0 | 0.0 | $2354_{\pm 706}$ | $4717_{\pm 1423}$ |
| Pixel2Mesh$^+$ | 93.5 | 88.4 | 39.9 | 30.9 | 59.1 | 40.2 | 40.5 | 30.2 | 50.8 | 62.4 | 18.2 | 26.7 | $2562_{\pm 0}$ | $5120_{\pm 0}$ |
| Sphere-Init | 94.1 | 87.5 | 40.5 | 40.9 | **75.2** | 44.2 | 50.3 | 28.4 | 48.6 | 42.5 | **26.9** | 7.0 | $2562_{\pm 0}$ | $5120_{\pm 0}$ |
| Mesh R-CNN (ours) | 94.0 | 88.4 | **51.1** | **48.2** | 71.7 | **60.9** | **53.7** | **42.9** | **70.2** | **63.4** | 21.6 | **27.8** | $2367_{\pm 698}$ | $4743_{\pm 1406}$ |
| # test instances | 2530 | 2530 | 2530 | 1165 | 415 | 419 | 213 | 154 | 79 | 54 | 11 | 20 | | |
| Pix3D $\mathcal{S}_2$ | | | | | | | | | | | | | | |
| Voxel-Only | 71.5 | 63.4 | 4.9 | 0.0 | 0.1 | 2.5 | 2.4 | 0.8 | 32.2 | 0.0 | 6.0 | 0.0 | $2346_{\pm 630}$ | $4702_{\pm 1269}$ |
| Pixel2Mesh$^+$ | 71.1 | 63.4 | 21.1 | 26.7 | 58.5 | 10.9 | 38.5 | 7.8 | 34.1 | **3.4** | **10.0** | 0.0 | $2562_{\pm 0}$ | $5120_{\pm 0}$ |
| Sphere-Init | 72.6 | 64.5 | 24.6 | 32.9 | **75.3** | 15.8 | 40.1 | 10.1 | 45.0 | 1.5 | 0.8 | 0.0 | $2562_{\pm 0}$ | $5120_{\pm 0}$ |
| Mesh R-CNN (ours) | 72.2 | 63.9 | **28.8** | **42.7** | 70.8 | **27.2** | **40.9** | **18.2** | **51.1** | 2.9 | 5.2 | 0.0 | $2358_{\pm 633}$ | $4726_{\pm 1274}$ |
| # test instances | 2356 | 2356 | 2356 | 777 | 504 | 392 | 218 | 205 | 84 | 134 | 22 | 20 | | |

**Model Miscellaneous Items:** (important elements that do not fall under any category)