PyTorch Implementation of Pixel2Mesh++

Samyak Jain
03752213
samyak.jain@tum.de

Zuzanna Trafas
03764759
zuzanna.trafas@tum.de

Hakan Alp
03765755
hakan.alp@tum.de

Abstract

Pixel2Mesh++ is the next step in multi-view 3D geometry generation as it is able to fill obscured area via hallucinating all the while using cross-view information. The official Pixel2Mesh++ implementation is in TensorFlow and this project ports it to PyTorch.

1. Introduction

Pixel2Mesh++ is an extension of the already existing paper Pixel2Mesh and it has an official implementation in TensorFlow. The aim of this project is to rewrite the Pixel2Mesh++ in PyTorch and compare it's performance. We talk briefly about the architecture of both Pixel2Mesh and Pixel2Mesh++ followed by the dataset used. Our work is discussed along with the losses and implementation details gives brief overview over the workflow followed swiftly by results and conclusion.

2. Previous work

Pixel2Mesh Generating meshes directly from 2-dimensional pictures has been a challenge in 3 dimension reconstruction and Pixel2mesh [4] provides a framework to train deep learning model end-to-end which generates 3D mesh from a single RGB image. It makes use of graph-based convolution neural networks to produce 3D mesh by progressively deforming an ellipsoid which is given as input. The network leverages the high-level perceptual features extracted from the image and it adopts the strategy of coarse-to-fine where the deformation starts from a small number of mesh nodes and progressively new nodes are added via unpooling and are deformed finely. The biggest drawback of this approach is that it cannot be extended to multiple images by design. Apart from that, the deformation method does not allow for breaks in between the mesh.

The architecture runs on the image side by passing the 2D image through VGG-16 like network and pooling together features. These features are produced from different layers output by projecting the current mesh's 3D coordinate onto the input image using the camera intrinsics and then pooling the features from nearby 4 pixels using bi-linear interpolation. These features are concatenated together with the 3-dimensional coordinates and passed onto the G-ResNet graph convolution neural network to produce the next deformed coordinates. Further, the graph unpooling layer increases the number of vertices using edge based unpooling where a vertex is added to an edge, in this case using mean, and connected to other new vertices formed in the face. In this way memory is saved as the mesh is unpooled only progressively and it is easier to place vertex coarsely in starting and fine-tune it with new vertices.

Pixel2Mesh++ The anticipated sequel to Pixel2Mesh, Pixel2Mesh++ [5] is the obvious extension that incorporates multiple images to generate a mesh. Multi-view geometry methods are well known to generate 3D shapes from correspondences across views however it still suffers from problems like having large baselines and poorly textured regions. Also having a smaller number of cross-view connections breaks the classical methods down. That's where using deep neural networks could be used, as they can learn cross-view connections. Idea is to extend Pixel2Mesh such that a coarse shape is generated by it and further deformation is applied by a new multi-view deformation network.

The multi-view deformation network starts with sampling possible new vertices for all the mesh vertices. This is done via deformation hypothesis sampling where a local graph of the 3D shape icosahedron is built with a mesh vertex as its center and all these local graphs are fed into a cross-view perceptual pooling layer. There each node is assigned features that are extracted from the multiple images using VGG-16 like network. Along with the statistics from different views, the deformation reasoning block gives a score to all the nodes in the local graph, of the original mesh coordinate which was at the center of the icosahedron, which causes a shift to the new center and that is how the mesh is deformed.

3. Dataset

As in the Pixel2Mesh++ and Pixel2Mesh papers, the same dataset provided by Choy et al. [2] is used. The dataset contains rendering images of 50k models belonging to 13 object categories from ShapeNet [1], which is a collection of 3D mesh models that are organized according to the WordNet hierarchy. In addition to that, it gives camera's pose and position along with ground truth mesh for objects. A model is rendered from various camera viewpoints but to keep consistent with TensorFlow implementation, only three particular viewpoints are considered. The same training/testing split as in Choy et . al. is utilised.

4. Our work

This project followed the official implementation of Pixel2Mesh++ which was written in TensorFlow 1. In this section, we discuss the challenges faced and the work that was done.

4.1. Graph Convolution network

The original Pixel2Mesh PyTorch implementation could not be used as it had to be extended to account for multi-view images. This is why the whole architecture was rewritten from scratch keeping the TensorFlow implementation in mind to be able to compare with it. Going through the TensorFlow version, placeholder variables were used to put data in every iteration and to pass different model parameters. This was solved by separating the training data and the parameters used to initialize the models. Some functions used in the TensorFlow version were not available in PyTorch and therefore implementations were found that best do the work and were validated independently. Their were too many parameters to learn which caused CUDA to run out of memory. To solve this, the perceptual network's weight were kept frozen with pretrained weights and CUDA cache was cleared after every iteration.

4.2. Losses

The losses used for Pixel2Mesh++ include all the losses used for Pixel2Mesh, with additional resampling used for Chamfer distance. In our implementation, we reused code from the Pixel2Mesh PyTorch implementation. They are included as follows:

Chamfer distance Chamfer distance is a metric for evaluating two cloud points. For each point in each cloud, it finds the closest neighboring point in the other set of points and sums the squares of the distances.

$$L_{chamfer}(S_1, S_2) = \frac{1}{|S_1|}\sum_{v_1 \in S_1} \min_{v_2 \in S_2} ||v_1 - v_2||_2^2 + \frac{1}{|S_2|}\sum_{v_2 \in S_2} \min_{v_1 \in S_1} ||v_1 - v_2||_2^2 \quad (1)$$

For Pixel2Mesh++ the point cloud generated by the model is resampled for each triangle uniformly using barycentric coordinates [3].

$$p = (1 - \sqrt{r_1})A + \sqrt{r_1}(1 - r_2)B + \sqrt{r_1}r_2C \quad (2)$$

for A, B, and C being the vertices of the triangle and random $r_1, r_2 \in [0, 1]$. The generated mesh is sampled for 4000 points that are added to the vertices.

We used the same module as the original implementation to calculate the Chamfer distance.

Normal loss Normal loss is defined on the surface normal for vertices v_1 and v_2

$$L_{normal} = \sum_{v_1}\sum_{v_2} || < v_1 - k, n_{v_2} > ||_2^2 \quad (3)$$

Where $k \in N(v_1)$ and v_2 is the closest neighbor of v_1 found when calculating the Chamfer distance. n_{v_2} is the observed surface normal from the ground truth.

Laplacian loss Laplacian loss is added as a regularization to prevent the vertices from moving too much. To calculate it first we need to calculate the laplacian coordinate for the vertex v

$$\delta_v = v - \sum_{k \in N(v)} \frac{1}{||N(v)||} k \quad (4)$$

Next, the laplacian regularization is defined as:

$$L_{laplace} = \sum_v ||\delta'_v - \delta_v||_2^2 \quad (5)$$

Where δ_v and δ'_v are the laplacian coordinates of a vertex before and after the deformation block.

Edge loss Edge loss adds regularization on the edge length

$$L_{edge} = \sum_v \sum_{k \in N(v)} ||v - k||_2^2 \quad (6)$$

The final loss is a weighted sum of all the losses

$$\begin{aligned} L = 3000 * L_{chamfer} + 0.5 * L_{normal} \\ +359 * L_{egde} + 1500 * L_{laplace} \end{aligned} \quad (7)$$

4.3. Implementation details

The implementation of architecture was tested using an overfitting script. The Pixel2Mesh++ paper talks about training the multi-view Pixel2Mesh followed by end-to-end training of Pixel2Mesh++ network but their TensorFlow implementation had these trained uncoupled. Following the TensorFlow implementation, two overfitting scripts were written and trained for the case of sofa and plane. The

Loss	Sofa		Plane	
	Ours	**Original**	**Ours**	**Original**
Chamfer	0.6197	1.3168	0.2127	224.5983
Edge	0.0014	0.0002	0.0003	0.0001
Normal	0.2558	0.2601	0.3119	0.2644
All	1.2642	1.5169	0.4793	224.7690

Table 1. Loss comparison for TensorFlow and PyTorch

whole dataset wasn't trained because of limited cluster GPU access and so overfit model were used as prototype. The comparisons of the overfit model is done with learned TensorFlow model because there is no overfit model available in TensorFlow implementation.

5. Results

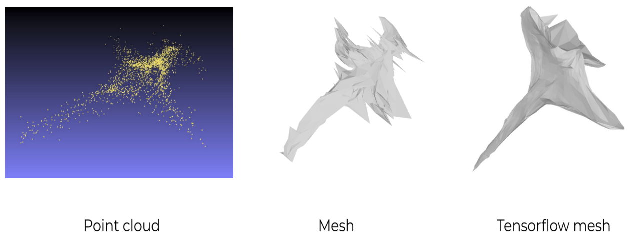

Figure 1. Our implementation and TensorFlow output comparison for plane

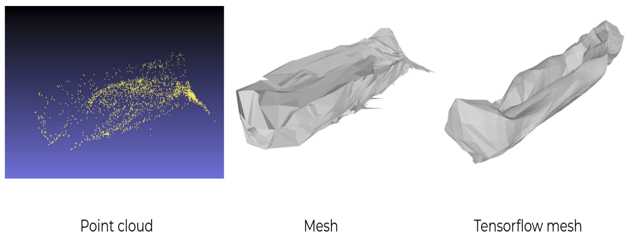

Figure 2. Our implementation and TensorFlow output comparison for sofa

The image results in Figure 1 and 2 shows that the PyTorch model was able to generate a good point cloud but the mesh had jitter.

The loss values, as seen in Table 1, suggest that during training original Pixel2Mesh++ implementation might have used loss weights different than the ones from their GitHub repository. The Chamfer loss is smaller for our implementation, even though the result looks worse. Edge and normal losses are bigger, but the weights are very small so they are not as important during the training.

We didn't include Laplace loss as we couldn't obtain vertices before deformation for the original implementation.

6. Conclusion

We managed to port Multi-View Pixel2Mesh and Multi-View Deformation Network model architectures to PyTorch and debug the overfitting script. The model was able to learn the general shapes of objects but produced a lot of spikes. The poor results and time constraint of project didn't allow for training the model on the whole dataset. For future works, a thorough look at the cause of spikes and fixing it can bring this project to a fruitful end.

7. Acknowledgments

This work exists thanks to Pixel2Mesh [4] and Pixel2Mesh++ [5]. During to project we use TUM's computer clusters that has dedicated RTX3090 GPU. Also PyTorch 1.12.1 with NVIDIA CUDA 11.3 used throughout the project.

References

[1] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 2

[2] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*, pages 628–644. Springer, 2016. 2

[3] Lubor Ladicky, Olivier Saurer, SoHyeon Jeong, Fabio Maninchedda, and Marc Pollefeys. From point clouds to mesh using regression. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 3913–3922. IEEE Computer Society, 2017. 2

[4] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–67, 2018. 1, 3

[5] Chao Wen, Yinda Zhang, Zhuwen Li, and Yanwei Fu. Pixel2mesh++: Multi-view 3d mesh generation via deformation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1042–1051, 2019. 1, 3