From Pixel to Mesh

Tong ZHAO Oscar CLIVIO
2019/05/04
tong.zhao@eleves.enpc.fr oscar.clivio@eleves.enpc.fr

1. Introduction

The underlying topic of this project is the reconstruction of a 3D object from a single RGB image. It is a difficult task, as some parts of the object are not visible in the view but still have to be found.

Among the most used 3D representations, we may choose between voxels, point clouds and meshes. Voxels are 3D extensions of pixels, being widely used, but their memory efficiency is poor [2]. Point clouds are easier to scale but applying textures or lighting on them is complicated as they don't have surfaces [2]. In this context, meshes may be considered as a compromise. Moreover, they are easy to deform and accurate in modelling shape details [3]. This is why this representation is used in all the articles suggested by the project presentation [1][2][3].

We chose to work on Pixel2Mesh (P2M) [3]. Given a 2D picture of an object, it is able to compute a 3D mesh representation after several deformations of an original ellipsoid mesh.

Our goals are:

- Entirely re-implement the model in PyTorch, as its original code is written in TensorFlow ;
- Reproduce its results on one class ;
- Test different regularization methods and play with different parameters ;
- Improve the structure of the network.

The report is organized as follows: in Section 2, we describe the P2M model, including its principles, modules, structure and losses; in Section 3, we introduce our improved version - an Auto-Encoder based P2M model; in Section 4, we present several results with respect to our implementation; in Section 5, we draw some discussions as well as a conclusion.

2. P2M Model

2.1. Main Ideas

Before grasping the concepts of Pixel2Mesh, we need to understand the representation of a mesh. It consists in a graph $\mathcal{M} = (\mathcal{V}; \mathcal{E}; F)$, where $\mathcal{V}$ denotes the vertices, represented as nodes, $\mathcal{E}$ the connections between these points, represented as edges, and F the features for each vertex.

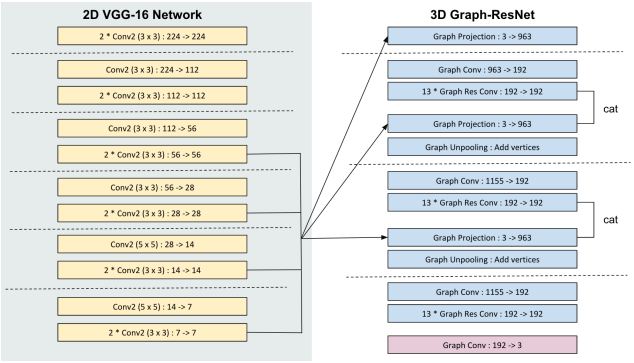


Figure 1: P2M Model - The left branch represents the structure of VGG-16 Net and the right branch represents the one of Graph-ResNet.

The core idea of Pixel2Mesh is to deform a simple mesh (here an ellipsoid) step after step guided by the features from its image. As we deform it, we start with a few vertices to bring stability, then we add more vertices to get a refined result. The figure 1 shows the detailed network structure.

2.2. The network

We quickly describe Pixel2Mesh's network. It consists in two parallel networks : a VGG-16 2D CNN and a 3D Graph Convolutional Network (GCN) which deforms an ellipsoid to the desired mesh.

There are three essential blocks in the GCN:

- **Mesh deformation block**: consists of 14 graph convolutional layers [4] with ResNet-like shortcut connections.
- **Perceptual feature pooling layer**: projects mesh vertices on the plane of the 2D image using camera intrinsic matrix. The corresponding 2D feature are retrieved using bilinear interpolation.
- **Graph unpooling layer**: adds the middle points of all edges to refine the mesh.

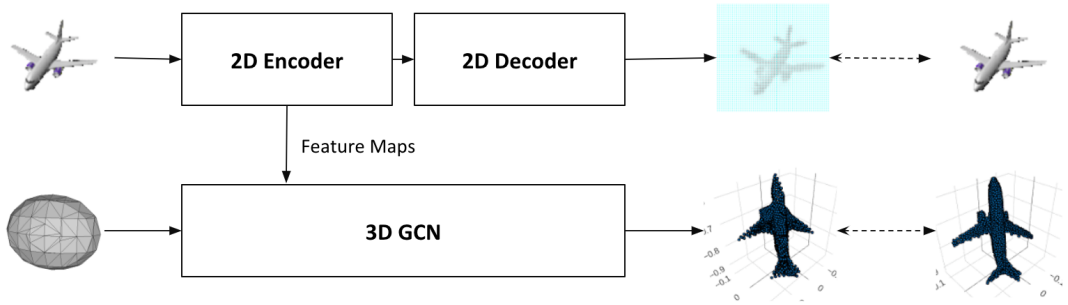


Figure 2: Auto-Encoder Based P2M Model. A image and a initial mesh (ellipsoid) are passed to the model. A reconstructed image and a deformed point cloud is produced. Then the model is optimized by several losses which guarantee both the image quality and the mesh quality.

Given an image, the features from four layers of the VGG-16 network are extracted and passed to the 3D network. The 3D GCN deforms the model in 3 steps: in a given step, the model projects image features to each vertex. Together with former mesh features (skipped in the first step) and raw coordinates , a mesh deformation block is performed and then a graph unpooling layer is used to refine the model (except the last step).

2.3. The losses

The following losses are computed to optimize the P2M model:

- **Chamfer loss :** it measures the discrepancy between two point clouds. However, it does not guarantee any connectivity of the graph.

$$l_c = \sum_p min_q ||p-q||^2 + \sum_q min_p ||p-q||^2$$

- **Tutte Laplacian loss :** it prevents the vertices from moving too freely after a deformation block.

$$l_{lap} = \sum_p ||\delta'_p - \delta_p||^2$$

where $\delta_p = p - \sum_{q \in \mathcal{N}(p)} \frac{1}{|\mathcal{N}(p)|} q$ is the distance between the point p and its Tutte laplacian embedding before the deformation, and δ'_p is the one after the deformation.

- **Edge Length Loss :** it penalizes long edges in the mesh by minimizing the following L_2 norm.

$$l_{loc} = \sum_p \sum_{q \in \mathcal{N}(p)} ||p-q||^2$$

3. Auto-Encoder Based P2M Model

One may notice that there is no loss having direct relation with 2D CNN. The only supervision is from the perceptual feature pooling layer via indexing functions, which is not differentiable with respect to the indices. Thus it takes a long time to make the 2D CNN converge. So the motivation of the auto-encoder-based model is to give a warm start (i.e. a good feature map) for the 3D GCN. The final structure is shown in figure 2.

3.1. Model

In the new model, we use an auto-encoder to replace the previous VGG-16 Net. While keeping all convolutional layers, we add 5 deconvolutional layers to reconstruct the image. U-net connections [5] are added to provide more detailed information for the image reconstruction task. The figure 3 shows the structure of the modified 2D CNN.

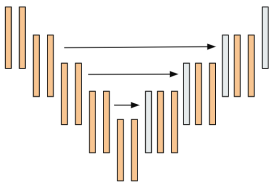


Figure 3: Auto-Encoder Based 2D CNN

3.2. Loss

The image reconstruction loss is the sum of two losses, namely the binary cross entropy loss and the L_1 loss. While the binary cross entropy loss minimizes the distance between two distributions of pixel values, the L_1 loss prevents the image being blurry. What's more, we minimize the image loss only in the first epoch, which corresponds to our purpose, i.e. "a warm start".

The total loss in the first epoch is thus $l = l_{mesh} + 0.0001 \times l_{img}$, where $l_{img} = l_{binary} + l_{l_1}$ and $l_{mesh} = 100 \times l_{chamfer} + 0.1 \times l_{edge} + 0.3 \times l_{lap}$. The total loss $l = l_{mesh}$ in the following epochs.

4. Experiment

4.1. Implementation

The code is implemented in Python and PyTorch. The source code can be found at `https://github.com/Tong-ZHAO/Pixel2Mesh-Pytorch`.

4.2. Training

We trained our model on the class - Plane. The training set contains 77662 images and the test set contains 19414 images. It takes 22 hours for the model to train 5 epochs on a single Nvidia GTX 1070 GPU.

The model has not yet well converged however we don't have enough time to make the training longer, limited by the deadline and the device. More experiments will be done in the near future.

4.3. Visualization

In order to evaluate our model, we picked some examples from the test set. We extract all three meshes in different scales, and compare them with the ground truth point cloud.

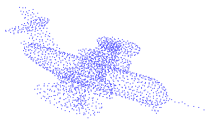


Figure 4: Ground Truth

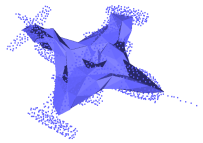


Figure 5: Mesh Scale 1

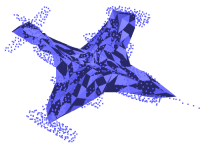


Figure 6: Mesh Scale 2

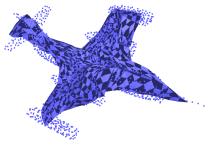


Figure 7: Mesh Scale 3

We observe that the multi-scale model refines the mesh step by step. Although there are still details which are not recovered at the final scale, the shape is well approximated. More examples can be found in Appendix.

What's more, the meshes are not smooth, which may be due to the absence of the normal loss. It is introduced in the original article but is not implemented in our model because of its complexity.

5. Discussion

In this section we discuss the advantages and the shortcomings of P2M model.

5.1. Advantages

First of all, P2M model is a well formulated end-to-end model which produces a mesh from a single view image. The perceptual feature pooling layer enables the interaction between the 2D CNN and the 3D GCN.

Secondly, the ingenious application of graph convolutional layers allows the network to deal with the mesh directly. The operation takes into consideration the connections among points, thus the mesh can be deformed smoothly.

Thirdly, the regularization terms, i.e. laplacian loss and edge length loss, force the output mesh to have a good continuity and uniformity. They are essential in the whole training process. However, one should take care of the coefficients and keep balance among all three mesh losses.

5.2. Shortcomings

The first shortcoming is that the training batch size is always 1 because of the perceptual feature pooling layer. Thus the gradient is not stable and the model easily gets stuck in a local minimum.

The second one is the overuse of 2D image features. In the 3D GCN, we concatenate each time 2D image feature of dimension 960, 3D mesh features of dimension 192, and 3D coordinates of dimension 3. While the mesh features are updated all the time, the 2D image features are always the same, which dominates the graph feature.

5.3. Following work

In the following, we will at first produce a statistical result on the whole ShapeNet dataset and compare its performance with its TensorFlow version. After then, we will try more structures to overcome the above mentioned shortcomings.

References

[1] G. et al. Atlasnet: A papier-mch approach to learning 3d surface generation. *CVPR*, 2018.

[2] K. et al. Neural 3d mesh renderer. *CVPR*, 2018.

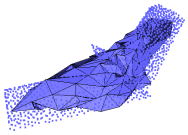
[3] W. et al. Pixel2mesh: Generating 3d mesh models from single rgb images. *ECCV*, 2018.

[4] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

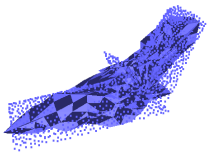
[5] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

Example 1

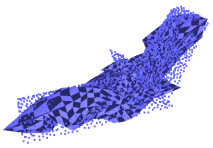
Ground Truth



Mesh Scale 1

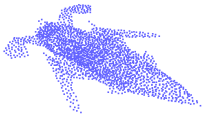


Mesh Scale 2

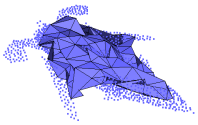


Mesh Scale 3

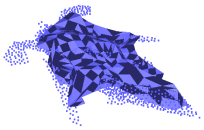
Example 2



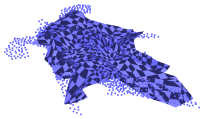
Ground Truth



Mesh Scale 1

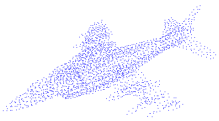


Mesh Scale 2



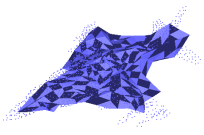
Mesh Scale 3

Example 3

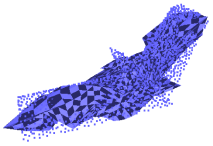


Ground Truth

Mesh Scale 1

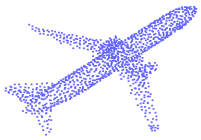


Mesh Scale 2

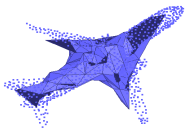


Mesh Scale 3

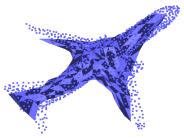
Example 4



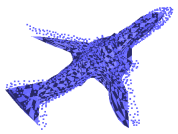
Ground Truth



Mesh Scale 1



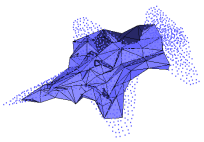
Mesh Scale 2



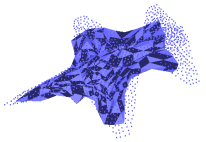
Mesh Scale 3

Example 5

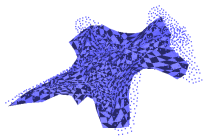
Ground Truth



Mesh Scale 1



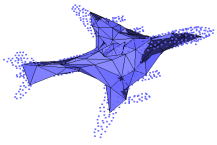
Mesh Scale 2



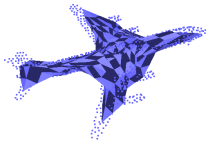
Mesh Scale 3

Example 6

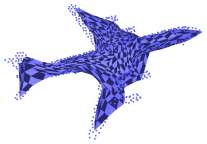
Ground Truth



Mesh Scale 1



Mesh Scale 2



Mesh Scale 3