DS603
Project Report
Generalizing MeshRCNN for novel data categories

Prakanshul Saxena
11940860

1. Introduction

Due to the rapid progress in computer vision algorithms, 3D reconstruction has also picked up pace. This advancement in both 2D and 3D vision, has led to development of various detection and segmentation based algorithms. However, the object detection algorithms make predictions in 2D, ignoring the 3D structure of the world. Concurrently, advances in 3D shape prediction have mostly focused on synthetic benchmarks and isolated objects. The 2D view ignores one critical fact that the world and the objects within it are 3D and extend beyond the XY image plane. Also the 3D shape prediction methods have only been developed on synthetic benchmarks such as ShapeNet consisting of rendered objects in isolation, which are less complex than natural-image benchmarks used for 2D object recognition like ImageNet.

So this work combines both of these aspects while providing significant improvement in the shape prediction task. This is done by augmenting Mask R-CNN with a mesh prediction branch that outputs meshes with varying topological structure by first predicting coarse voxel representations which are converted to meshes and refined with a graph convolutional network op- erating over the mesh's vertices and edges.

This work has trained its model on the object categories - Chair, sofa, table, bed, desk, wardrobe, tool, miscellaneous, bookcases. Now we aim to incorporate a new category in the model, and obtain segmentation and shape prediction results similar to the categories already present.

2. BACKGROUND AND RELATED WORKS

This work deals with two mainstream problems in computer vision: Object Semantic Segmentation and 3D Shape Prediction/Construction.

There have been numeros works which perform object recognition. Works have also focused on localizing the recognized object with bounding boxes or masks, as per the requirement of the problem being tackled.
In the 3D domain, there have been works which focus on 3D shape prediction that too in different formats - voxels, meshes, point clouds etc. This work focuses on mesh based shape prediction.

Mask-RCNN is used in this work for object recognition and segmentation. And 3D shape prediction is slightly based on the concepts used by Pixel2Mesh.

3. SYSTEM MODEL

I tried running it on paperspace, but there is a notebook limit of 10 notebooks. So I chose to shift the platform to AWS Sagemaker.

The repository of this work is available at - https://github.com/facebookresearch/meshrcnn.
The whole code is implemented through two main libraries/frameworks - Pytorch3D and Detectron2.

3.1 Dataset

3.1.1 ShapeNet

This dataset provides a collection of 3D shapes, represented as textured CAD models organized into semantic categories following WordNet and has been widely used as a benchmark for 3D shape prediction. It covers 55 common object categories with about 51,300 unique 3D models.

The task on this dataset is to input a single RGB image of a rendered ShapeNet model on a blank background, and output a 3D mesh for the object in the camera coordinate system. During training the system is supervised with pairs of images and meshes.

The split for this dataset used by the work is -

Train - 35,011 models (840,189 images)
Test - 8,757 models (210,051 images)
Also 5% of the training models are reserved as a validation set.

The evaluation metrics for the task on this ShapeNet Dataset are - Chamfer Distance, Normal Consistency and F1 at various threshold distances of points.

3.1.2 Pix3D

This dataset consists of 10069 real-world images and 395 unique 3D models. The task is to jointly detect and predict 3D shapes for known object categories. The work implements its own splits on this Pix3D dataset.

S1 split randomly allocates 7539 images for training and 2530 for testing.

S2 is the split in which the 3D models appearing in training and testing images are disjoint.

These splits are significantly different from the ShapeNet dataset, and also more challenging as the same model can appear with varying appearance (e.g. color, texture), in different orientations, under different lighting conditions, in different contexts, and with varying occlusion, which is not the case in ShapeNet as the objects appear in it in blank backgrounds. Thus we can conclude that among the two splits in the paper : S1 and S2, S2 becomes more challenging as it is also novel 3D shapes of known categories: for example a model may see kitchen chairs during training but must recognize armchairs during testing. This split is possible due to Pix3D's unique annotation structure, and poses interesting challenges for both 2D recognition and 3D shape prediction.

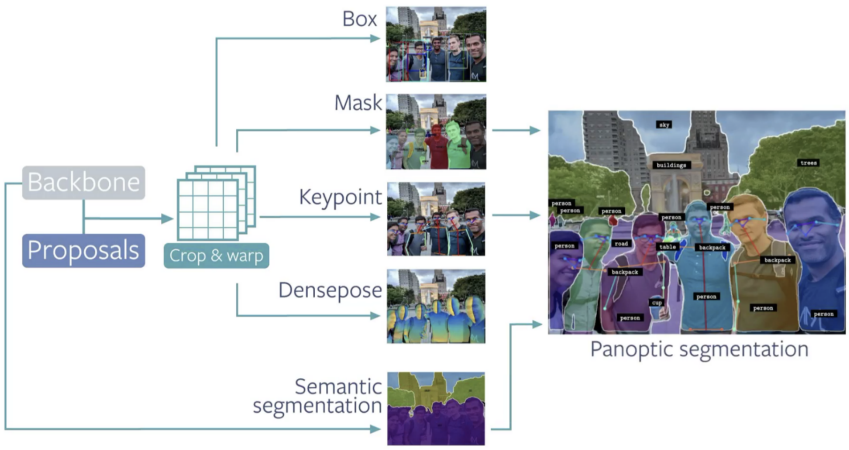
The evaluation metrics for this dataset are - AP Box, AP Mask, and AP Mesh. The first two being the standard object detection and instance segmentation IoU metrics. The third one is for 3D shape prediction. It is the mean area under the per-category precision-recall curves for $F1^{0.3}$ at 0.5. This annotation means that F1 score at the distance of 0.3 is 0.5. A mesh prediction is considered a true-positive if its predicted label is correct, it is not a duplicate detection, and the F1 condition is satisfied.

3.2 Detectron2

It is a library developed by FAIR using Pytorch. The Detectron 2 is a complete rewrite of the first Detectron which was released in the year 2018. The predecessor was written on Caffe2, a deep learning framework that is also backed by Facebook. Both the Caffe2 and Detectron are now deprecated. Caffe2 is now a part of PyTorch and the successor, Detectron 2 is completely written on PyTorch. It also provides a very easy API to extract scoring results.

Various object detection and segmentation tasks can be provided along with different backbone networks which are also customizable.

Below image depicts the versatility of Detectron2.



Also using Detectron2 we cannot exactly print the model architecture as readily as possible in Pytorch. This is due to the encapsulation it provides over Pytorch.

This work uses detectron2 for implementing the MaskRCNN branches and from that it uses Pytorch3D to build the mesh and voxel based layers.

3.3 Pytroch3D

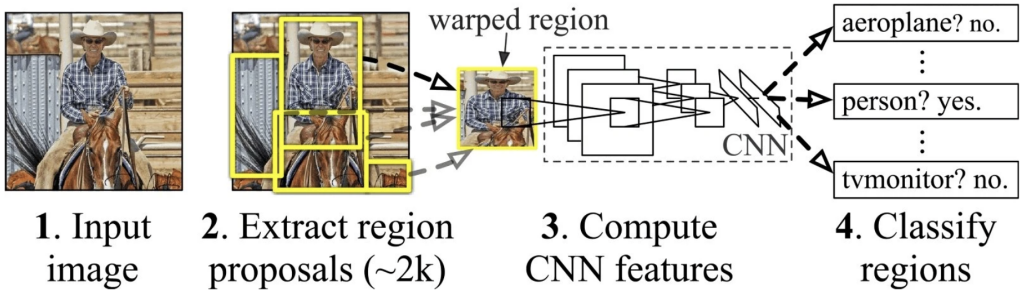
Pytorch3D is built with the aim of accelerating research at the intersection of deep learning and 3D. It encounters several challenges including 3D data representation, batching, and speed. PyTorch3D has included efficient 3D operators, heterogeneous batching capabilities, and a modular differentiable rendering API, to equip researchers in this field with a much needed toolkit to implement cutting-edge research with complex 3D inputs. There is a flexible interface for loading and saving point clouds and meshes from different formats. For meshes, this supports OBJ, PLY and OFF files. For point clouds, this supports PLY files.

Pytorch3D is equipped to perform heterogeneous batching of 3D meshes. Other operations of MeshRCNN such as Cubify and 3D IoU are readily incorporated in this framework.
Another aim of building Pytorch3D is to incorporate differentiable rendering for reducing computation costs and calculating losses in 2D space. Differentiable rendering allows to bridge the gap between 2D and 3D by allowing 2D image pixels to be related back to 3D properties of a scene. Also the implementation of this can also be used without GPU.

4. ALGORITHMS

4.1 Mask RCNN

It stands for Mask Region based Convolutional Neural Network. The architecture of Mask-RCNN was designed to solve image detection tasks. Also, R-CNN architecture forms the basis of Mask R-CNN and it was improved into what we know as Faster R-CNN.



Basic Architecture of Mask-RCNN

Fast R-CNN is an improved version of R-CNN architecture along with RPN and ROI Pool.

Now above this Faster R-CNN is an optimized form of R-CNN because it is built to enhance computation speed.

Mask RCNN is built upon Faster RCNN to perform the segmentation tasks. A task of instance segmentation is carried out in the paper. It deals with accurately identifying every object in an image and also segmenting each instance. Hence, it combines object localisation, object detection, and object classification. In other words, this kind of segmentation takes a step further to clearly distinguish each object that is categorized as a similar instance.

Basic Architecture of Faster-RCNN

MaskRCNN includes the addition of a new branch that outputs the object mask. The additional mask output is distinct from the class and box outputs, requiring the extraction of a much finer spatial layout of an object.

The work builds upon MaskRCNN to get the 2D segmentation masks of the localized objects.

4.2 Cubify

This operation runs on the output of the Voxel Branch. The voxel branch gives a VxVxV output with values containing probabilities of voxels being occupied. After this rough voxel based shape is formed, the cubify operation is applied to it. The cubify operation replaces each voxel with a cube of - 8 vertices, 12 faces, 18 edges (a normal cube with a single extra diagonal at each face).

This operation thus converts the image into a mesh, after which mesh refinement blocks are applied.

4.3 Vert Align

This is an important algorithm used by this wmork to obtain the 2D information from an image to construct the shape in 3D mesh representation. The intermediate feature maps of the 2D images are used to map a pixel of 2D image (using bilinear interpolation) to the corresponding vertex in the 3D mesh representation.

This VertAlign is the same algorithm as the perceptual feature pooling in Pixel 2 Mesh. This algorithm is based on the Spatial Transformer Networks. These networks adaptively transform their input to a canonical, expected pose. These include the concepts of bilinear interpolation and reverse mapping. That is, the vertex (3D) coordinates are first projected onto the xy-plane and de-normalized according to the feature map height and width. Then for each xy-coordinate, a feature vector is sampled using bilinear interpolation.
Resnet 50 is used as the backbone feature extractor in this work, thus VertAlign operator concatenates features from conv2_3, conv3_4, conv4_6, and conv5_3 before projecting to a vector of dimension 128.

$$
\begin{aligned}
f(x,y) &= \frac{y_2-y}{y_2-y_1} f(x,y_1) + \frac{y-y_1}{y_2-y_1} f(x,y_2) \\
&= \frac{y_2-y}{y_2-y_1}\left(\frac{x_2-x}{x_2-x_1} f(Q_{11}) + \frac{x-x_1}{x_2-x_1} f(Q_{21})\right) + \frac{y-y_1}{y_2-y_1}\left(\frac{x_2-x}{x_2-x_1} f(Q_{12}) + \frac{x-x_1}{x_2-x_1} f(Q_{22})\right) \\
&= \frac{1}{(x_2-x_1)(y_2-y_1)} \left(f(Q_{11})(x_2-x)(y_2-y) + f(Q_{21})(x-x_1)(y_2-y) + f(Q_{12})(x_2-x)(y-y_1) + f(Q_{22})(x-x_1)(y-y_1)\right) \\
&= \frac{1}{(x_2-x_1)(y_2-y_1)} \begin{bmatrix} x_2-x & x-x_1 \end{bmatrix} \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2-y \\ y-y_1 \end{bmatrix}.
\end{aligned}
$$

This is how we can sample a point from VertAlign using Bilinear Interpolation given the camera intrinsics (Q)

4.4 Graph Convolution

We use graph convolutions to learn the relationships between adjacent vertices. In this each vector (on which we apply graph convolutions) is the 128 dimension vector associated with its vertex.

Multiple graph convolutional layers (along with residual graph conv layers) are applied in each mesh refinement stage. To retrieve the updated xyz coordinates, a tanh activation is used. The vertices can then move in 3D space and the mesh can be improved by adding those changes to the original vertex positions.

4.5 Losses

4.5.1 Cross Entropy

Binary cross-entropy loss is used between the voxel occupancy grid and the ground-truth voxels to train the voxel branch.

4.5.2 Chamfer Distance

Chamfer Distance (CD) is an evaluation metric for two point clouds. It takes the distance of each point into account. For each point in each cloud, CD finds the nearest point in the other point set, and sums the square of distance up.

P,Q - two point clouds
p,q - P is in point cloud P and q is the closest point to p in point cloud Q.

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

4.5.3 Normal

The normal distance penalizes mismatched normals between two point clouds.

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

u_p and u_q are the normals of points.

4.5.4 Edge loss/Laplacian

Shape regularizers are also required for constructing a fine high quality mesh. Thus we use an Edge Loss for the same. As an alternative to the edge loss, laplacian can also be applied.

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

Final loss for the model is calculated based on the weighted sum of the above three losses.

5. EXPERIMENTS AND RESULTS

To implement this work and add new object categories to this model, I have first tried to learn Pytorch 3D and then understand the concept behind the Detectron2 library. After understanding their codebase a bit, I have then set up the environment required for this on AWS Sagemaker. This task was taxing as different dependencies (including new libraries and frameworks such as Detectron2 and Pytorch3D respectively) along with the appropriate CUDA version are difficult to install. Currently the environment is completely built and ready in AWS Sagemaker along with the appropriate CUDA Version - 11.4.

The .ipynb file submitted last time just displayed the use of the mesh-rcnn pretrained model. But now we are moving forward in different directions. After this I have worked in two directions - one of which is to train the model and evaluate the training results, and the other is to study the codebase in order to incorporate a new data category.

5.1 Training the Model

In this direction of the project, I have worked on the Pix3D dataset as the ShapeNet dataset took a long time to get aptly approved for downloading and also the ShapeNet Core dataset (the one used in the paper) has a size of 54 GB. And this is too large as compared to the resources available. The Pix3D dataset is 3GB and thus can be easily used to produce the implementation results associated with the Pix3D dataset.

Now I ran the training using two different AWS Sagemaker instances - g4dn.xlarge and g4dn.4xlarge and keeping the same image - “Pytorch 1.13 GPU Optimized”. However, after this I realized that both the instances only offer a single T4 GPU which is only able to provide 16GB of GPU memory. This is not enough to train the Mesh RCNN model.

Error - `torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 1.39 GiB (GPU 0; 14.76 GiB total capacity; 12.74 GiB already allocated; 559.75 MiB free; 13.15 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation. See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF`

The minimum memory required to train the MeshRCNN has not been mentioned in either their paper or on their repository (https://github.com/facebookresearch/meshrcnn/issues/92). However, in one issue the developer has mentioned the maximum GPU allocation (https://github.com/facebookresearch/meshrcnn/issues/37#issuecomment-640110545) (https://github.com/facebookresearch/meshrcnn/issues/37) per Tesla V100 GPU . So since the

model is trained using 8 V100 GPUs, which is when keeping the batch size to 16 images (*mentioned in the issue, but in paper an implementation of 64 images per batch is done on the Pix3D implementation*). We can probably assume that the total GPU memory required is 56GB if training on a single GPU with a batch size of 16 images. This can be incorporated into a 16GB single GPU, if we set the batch size to 4 images (Calculation Below). But some space is also reserved for GPU Optimized Pytorch. So this needs to be adjusted using the *os* library.

7 GB per GPU for 8 GPUs for a Batch size of 16 (2 images per GPU)
So for a single GPU with a Batch size of 16 : 56GB will be required.
So for a single GPU with a Batch size of 4 : 14GB will be required (4 images per GPU or the single present GPU).

So if we compromise on time, a slight increase in the GPU memory should be enough for the training. Thus I have submitted the same issue to AWS support in order to enquire about solutions for this issue.

The reply from AWS support is the same which I have speculated already. And thus I have again told them to provide me with better computation options to use for the same task. I am waiting for their reply and incorporating the current speculated logic for running training.

Reply from AWS -

```
I understand your concerns and also apologize for the issue. Regarding your error, PyTorch is trying to load the model into GPU0, it
reports that it sees W GiB total capacity (the GPU's memory), and it has already allocated X GiB of that GPU's memory. It only has Y GiB
free and needs Z GiB in order to complete its action.

You can try the following to remediate the issues :
1. Set the "max_split_size_mb" parameter.
import os
os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "max_split_size_mb:<enter-size-here>"
Please try out different sizes here and check which one works.

2. Incrementally reduce the batch size. For one particular value, this error should be mitigated.

3. Tensors usage: Minimize the number of tensors that you create. The garbage collector won't release them until they go out of scope.

4. Reduce the size of your model.

5. try using pytorch methods to split models across your GPUs. Note that splitting models mean that your GPUs and/or CPUs may be idle at
times while waiting for other GPUs/CPUs to finish processing, as such, this may end up more expensive that using a newer instance type.
Please make use of the following guides :
1. https://pytorch.org/tutorials/beginner/former_torchies/parallelism_tutorial.html
2.  https://discuss.pytorch.org/t/sharding-model-across-gpus/1065/2

6. Please do try a different instance type- one with a higher memory / computation power. This will definitely enable your model more
performance capabilities.
```

5.2 Incorporating new Data Category

I have studied their codebase for incorporating new data categories to generalize the model, and found two helping issues - https://github.com/facebookresearch/meshrcnn/issues/6 and https://github.com/facebookresearch/meshrcnn/issues/19 . https://github.com/facebookresearch/meshrcnn/issues/6#issuecomment-582041953 explains it completely.

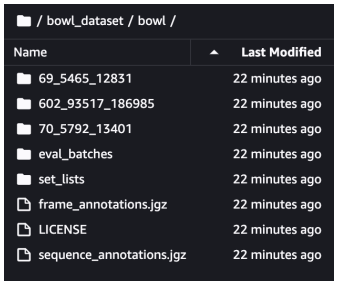
So to train I have to edit the following file - https://github.com/facebookresearch/meshrcnn/blob/main/meshrcnn/data/datasets/builtin.py and then save my 3D dataset (image and 3D annotations) in the COCO format as the Pix3D has been saved in the coco format. After that we have to create a config file for the new dataset added and then we can train the model with the new added categories. This is a challenging task as we have to go extensively through the detectron2 tutorials in order to facilitate this.

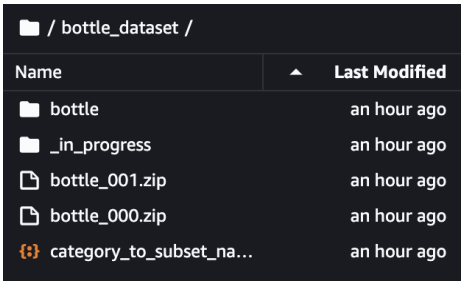
And this would be the secondary task for adding a new dataset category. The first is to source such data. For this another dataset we have - https://ai.facebook.com/datasets/CO3D-dataset/. This dataset includes many sourced images and their 3D annotations. This dataset spans to 50 MS-COCO categories. Thus we can use object categories from this dataset and then incorporate them into Mesh RCNN for generalizing it. Therefore I am currently working on downloading this dataset (https://github.com/facebookresearch/co3d) and extracting a single category from it and then assuring that it is saved according to COCO format for Mesh RCNN incorporation.

—----------Further Progress—----------

5.3 Dataset sourcing and Downloading complete

The download of the dataset for a single category is done. I studied the repository - https://github.com/facebookresearch/co3d and found out that we can download the dataset in different sizes pertinent to our needs and resources. The full dataset has a size of 5.5TB. So I found that installing the co3d package and modifying the file - https://github.com/facebookresearch/co3d/blob/main/co3d/download_dataset.py , we can download the data for a single category. Also the issue explains it in some detail - https://github.com/facebookresearch/co3d/issues/68#issuecomment-1475242239 . Thus I implemented this in my AWS environment and downloaded the dataset for ***bottles*** and ***bowls***. It can be seen that these images are sourced from the public to create a large dataset of images and its 3d masks.





Also the full versions of the dataset are of sizes - 17 to 25 GBs, so I have downloaded a subset of the dataset consisting only of the sequences selected for the many-view single-sequence task where both training and evaluation are commonly conducted on a single image sequence. In this current dataset - bottle occupies 14.5 MB and bowl occupies 523 MB. Also these categories are not present in the pix3d dataset. The Pix3d dataset only has 9 categories in which bowl and bottle are not present.

I am currently working on putting these dataset along with the mesh-rcnn repository.

5.4 Final result after incorporating the dataset

I had incorporated the Bottle co3d dataset in the main model and the input to the model (left) and outputs (middle and right) are as follows -

6. References

1) Sun, X., Wu, J., Zhang, X., Zhang, Z., Zhang, C., Xue, T., Tenenbaum, J.B. and Freeman, W.T., 2018. Pix3d: Dataset and methods for single-image 3d shape modeling. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2974-2983).
2) Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H. and Xiao, J., 2015. Shapenet: An information-rich 3d model repository. arXiv preprint arXiv:1512.03012.

3) Wang, N., Zhang, Y., Li, Z., Fu, Y., Liu, W. and Jiang, Y.G., 2018. Pixel2mesh: Generating 3d mesh models from single rgb images. In Proceedings of the European conference on computer vision (ECCV) (pp. 52-67).
4) Ravi, N., Reizenstein, J., Novotny, D., Gordon, T., Lo, W.Y., Johnson, J. and Gkioxari, G., 2020. Accelerating 3d deep learning with pytorch3d. arXiv preprint arXiv:2007.08501.
5) https://research.fb.com/wp-content/uploads/2019/12/4.-detectron2.pdf
6) Jaderberg, M., Simonyan, K. and Zisserman, A., 2015. Spatial transformer networks. Advances in neural information processing systems, 28.
7) Gkioxari, G., Malik, J. and Johnson, J., 2019. Mesh r-cnn. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 9785-9795).
8) https://ai.facebook.com/datasets/CO3D-dataset/
9) He, K., Gkioxari, G., Dollár, P. and Girshick, R., 2017. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 2961-2969).
10) Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C.L., 2014. Microsoft coco: Common objects in context. In Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13 (pp. 740-755). Springer International Publishing.