

3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction

Yimin Pan

Weixiao Xia

Linzhao Wang

Abstract

3D reconstruction from 2D images is a well-known task in computer vision. The goal of this task is to estimate 3D shape from the semantic attributes of the 2D inputs. However, single image based reconstruction is an extremely ill-posed problem as the final reconstruction is only constrained by a single view. Multi-view based approaches, in contrast, generally provides better reconstruction as it makes the problem more constrained. The aim of this project is to improve the performance of one of the classic multi-view reconstruction methods, 3D-R2N2, by modifying its architecture. Specifically, two variants are implemented: 1) In the first variant the standard CNN encoder is replaced with a Feature Pyramid Network to extract features at different scales. 2) In the second variant Transformer is used to predict the 3D hidden states and a Self-Attention module is used to fuse the information from different views instead of the 3D Convolutional LSTM. We evaluate these 2 variants and find that both outperforms the baseline method.

1. Introduction

Rapid and automatic 3D object prototyping has become a game-changing innovation in many applications related to e-commerce, visualization, and architecture, to name a few. Traditional 3D reconstruction methods rely on feature matching between different views of an object to create a sparse/dense reconstruction. However, the accuracy of the reconstruction highly depends on the quality of the hand crafted feature descriptors, the number of detected keypoints, and the accuracy of the triangulation process. Moreover, the latter requirement impede us from using images taken from arbitrary viewpoints as if the viewpoints are separated by a large baseline, establishing feature correspondences is extremely problematic due to local appearance changes or self-occlusions.

To overcome all these problems, a novel deep learning method was proposed by Choy et al. [1] to reconstruct 3D shape from 2D images. The proposed 3D-R2N2 takes in one or more images of an object instance from arbitrary viewpoints and outputs a reconstruction of the object in the

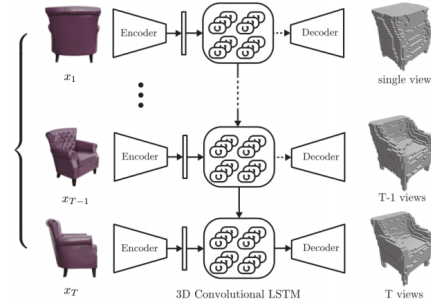


Figure 1. Overview of the 3D-R2N2's architecture.

form of a 3D occupancy grid. But, while 3D-R2N2 is able to reconstruct the coarse shape of the object, it fails to capture finer details. We believe this is mainly caused by 2 reasons: 1) the wrong choice of the loss function. 2) the incapability of the network architecture to capture finer details.

Thus, in this project we aim to improve the performance of 3D-R2N2 by solving the above problems, and our main contributions can be summarized as:

1. Replace the original BCE loss with 3D Dice loss which is more suitable for the task of 3D reconstruction.
2. Two variants of the 3D-R2N2 that are capable to capture finer details based on the usage of Feature Pyramid Network and Transformer.

2. Related work

3D Recurrent Reconstruction Neural Network 3D-R2N2 takes in one or more images of an object instance from arbitrary viewpoints and outputs a reconstruction of the object in the form of a 3D occupancy grid. The reconstruction is incrementally refined through the recurrent component as the network sees more views of the object, as illustrated in Figure 1.

The key part of the 3D-R2N2 is the 3D Convolutional LSTM. It consists of several LSTM units and each unit is responsible for processing a particular region of the final dense grid. During each time step, each LSTM unit takes

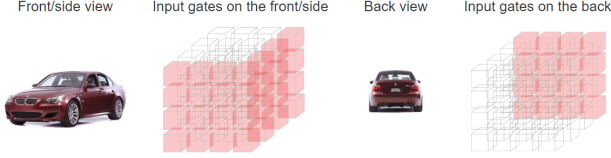


Figure 2. Selective update mechanism of the 3D-Convolutional LSTM module.

the feature vector extracted from the current view image, the current local neighborhood context, and the previous hidden state as input and updates its hidden state.

The update process is selective in the sense that, if the input image is taken from the front/side view, then only the input gates correspond to the front and side view will be activated. If the input image is taken from back view, then only input gates correspond to back view will be activated. This operation allows the network to put image features to the right position as illustrated in Figure 2.

Feature Pyramids Network The FPN was proposed by Lin et al. [2] to improve the performance of detection tasks. It proposes a topdown architecture with lateral connections for building high-level semantic feature maps at all scales as illustrated in Figure 3. This provides a hierarchical representation of features at multiple scales and enables the network to effectively detect objects of various sizes in an image by predicting at different scales.

Detection of large objects are mostly happening at higher pyramidal levels while detection of small objects are mostly happening at lower pyramidal levels. Since lower pyramidal levels have higher resolution, it is then feasible to detect small objects. Higher pyramidal levels, however, have lower resolution and is dominated by features of large objects since features of small objects are lost during the sub-sampling process.

Detection Transformer Transformer was first proposed by Vaswani et al. [4] for the language translation task. Carion et al. [3] later proposed DETection TRAnsformer, or simply DETR, which brings Transformer to the object detection task and achieve state-of-the-art performance. This new method streamlines the detection pipeline, effectively removing the need for many hand-designed components like a non-maximum suppression procedure or anchor generation that explicitly encode our prior knowledge about the task of detection.

The overall architecture of DETR is illustrated in 9. DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional

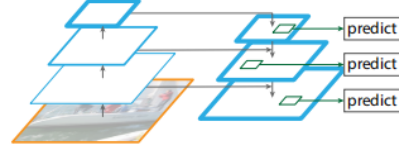


Figure 3. A top-down architecture with skip connections, with predictions made independently at all levels.

embeddings, which we call object queries, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a “no object” class.

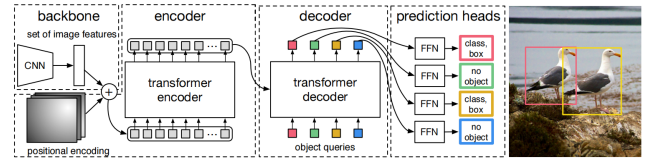


Figure 4. Overview of the DETR's architecture.

3. Proposed Modifications

In this section, we will first give a detailed description about the architecture of the baseline. Then, we will go through each of the modifications we made explaining what exactly did we modify and the reason behind it, i.e., why such a modification would lead to better performance.

3.1. 3D-R2N2

In the original paper several architectures of 3D-R2N2 was proposed. But in this project only the Res3D-GRU-3 was implemented since according to the original paper this one got the highest score. The overall structure of the Res3D-GRU-3 is illustrated at Figure 5 and it consists of:

A deep residual 2D-CNN encoder Build from standard residual blocks, pooling layers, and leaky rectified linear units followed by a fully-connected layer.

A 3D Convolutional GRU The network is made up of a set of structured GRU units with restricted connections as illustrated in Figure 6. The 3D-GRU units are spatially distributed in a 3D grid structure, with each unit responsible for reconstructing a particular part of the final output. Inside the 3D grid, there are $N \times N \times N$ 3D-GRU units where N is the spatial resolution of the 3D-GRU grid. Each 3D-GRU unit, indexed (i, j, k) , has an independent hidden state $h_{t,(i,j,k)} \in \mathcal{R}^{N_h}$. Formally, the update rule for each 3D-

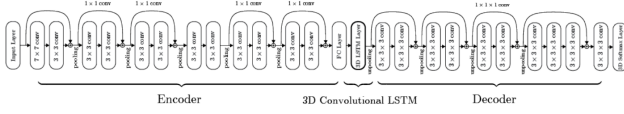


Figure 5. Overall architecture of Res3D-GRU-3.

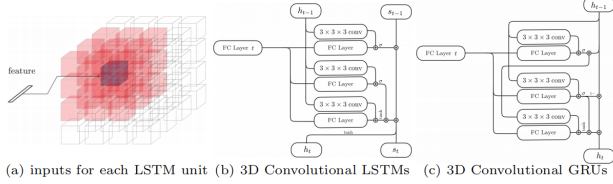


Figure 6. (a) At each time step, each unit (purple) in the 3D-LSTM receives the same feature vector from the encoder as well as the hidden states from its neighbors (red) by a $3 \times 3 \times 3$ convolution ($W_s * h_{t-1}$) as inputs. (b) 3D-LSTMs without output gates. (c) 3D Gated Recurrent Units (GRUs)

GRU unit can be expressed as

$$\begin{aligned} u_t &= \sigma(W_{fx}\mathcal{T}(x_t) + U_f * h_{t-1} + b_f) \\ r_t &= \sigma(W_{ix}\mathcal{T}(x_t) + U_i * h_{t-1} + b_i) \\ h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot \tanh(W_h\mathcal{T}(x_t) + \\ &\quad U_h * (r_t \odot h_{t-1}) + b_h) \end{aligned}$$

where u_t , r_t , h_t represent the update, reset, and hidden state respectively, and the $*$ operator denotes for 3D convolution. For vanilla LSTM/GRUs, all elements in the hidden layer h_{t-1} affect the current hidden state h_t , whereas a spatially structured 3D Convolutional LSTM/GRU only allows its hidden states $h_{t,(i,j,k)}$ to be affected by its neighboring 3D-LSTM units.

A 3D-CNN decoder made from stacked blocks of 3D Transposed Convolutions and 3D Convolutions.

Thus, Given one or more images of an object from arbitrary viewpoints, the 2D-CNN first encodes each input image x into low dimensional features $\mathcal{T}(x)$. Then, given the encoded input, a set of newly proposed 3D Convolutional GRU (3D-GRU) units either selectively update their cell states or retain the states by closing the input gate. Finally, the 3D-CNN decodes the hidden states of the GRU units and generates a 3D probabilistic voxel reconstruction.

3.2. Loss function

In the original implementation voxel-wise BCE was used as the loss function. However, due to the large data imbalance between empty and filled voxels, voxel-wise BCE does not force network to capture details as losing some small part of the shape does not increase loss sharply. Thus, we

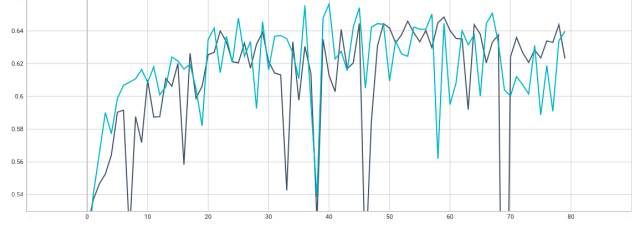


Figure 7. Comparison between BCE (dark green curve) and Dice loss (blue curve). x-axis: epoch. y-axis: iou score over test set.

replaced the BCE loss with the 3D Dice loss instead. Compared to BCE, Dice is more sensitive to the completeness of the shape as it focus more on improving the IoU score.

The Figure 7 illustrates the testing curve of baseline trained with BCE (dark green) and Dice (blue) respectively. We can see that baseline trained with Dice loss achieves higher IoU score over the test set compared to BCE.

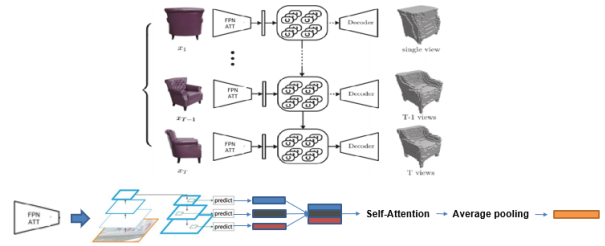


Figure 8. Overview of the Res3D-GRU-3 integrated with FPN.

3.3. 3D-R2N2 with FPN

From the Figure 5 we can see that the base Res3D-GRU-3 performs feature extraction at one single scale. If we think this as a detection task, it could be that the encoder is detecting some important features that are useful for the reconstruction. As we are only extracting features at the finest level, we will only capture features that describe the global shape. Just like in detection tasks, if we predict at the finest level only, then we will only be able to detect large objects since small objects are lost during the sub-sampling process. This also explains why the original 3D-R2N2 was not able to capture details.

Thus, the same reason why FPN could improve the performance of detection networks also applies in this case. The only difference is that, instead of detecting objects of different sizes, here we detect features of different 'sizes'. Features extracted from the finest level better describes the global shape, while features extracted from coarser levels better describe details.

Once we have extracted all the features, these feature vectors will be then passed to a self-attention module to en-

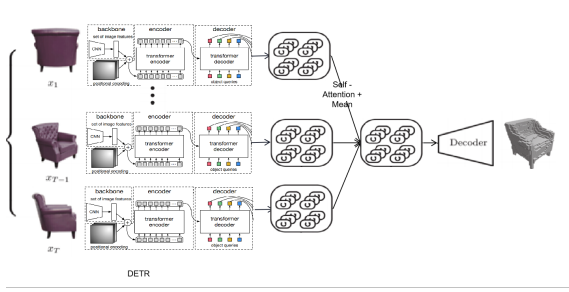


Figure 9. Overview of the Res3D-GRU integrated with DETR.

hance features that contributes most to the final reconstruction. After that, it will be fed to an average pooling layer to get the final fused feature vector, which is then passed to the 3D-GRU module.

The overall architecture of this variant is illustrated in the Figure 8.

3.4. 3D-R2N2 with DETR

The reason why we choose Transformer is not that obvious compared to the FPN. In simple words, the architecture of DETR suits perfectly the way how 3D-R2N2 constructs the base 3D hidden state.

Given that the size of the base 3D hidden state is fixed, we demonstrate how the architecture of DETR is able to generate the same 3D hidden state. From the Figure 9 we can see that the Transformer decoder takes as input the output of the Transformer encoder and a fixed set of learnt object queries and outputs a fixed set of object embeddings which is later used to detect objects.

For illustration, we can think object queries as queries that request information about a particular spatial region of the input image: "Do this region contain any object? If so, what it's the class of the object and the bbox?" Of course it works in a more complex way, but if we think it this way, then the object queries can be smoothly translated to voxel queries by just changing the question to "Do this voxel contain any part of the object? If so, filled it with corresponding information". So, in order to predict the base 3D hidden state, we just need to use $N \times N \times N$ object (voxel) queries to request embedding for each voxel, where N denotes spatial resolution of the base 3D hidden state.

Furthermore, by using self- and encoder-decoder attention over these embeddings, DETR globally reasons about all objects (or voxel in our case) together using pair-wise relations between them, while being able to use the whole image as context.

Thus, the key idea here is that, we first use DETR (with out prediction heads since it's used for class and bbox prediction) to generate the base 3D hidden state for each view image. These base 3D hidden states are then feed to a self-attention module followed by an average pooling layer to

| | Baseline | Base. with FPN | Base. with DETR |
|-----|----------|----------------|-----------------|
| IoU | 0.64 | 0.72 | 0.71 |

Table 1. Quantitative comparison between baseline and the proposed variants in term of IoU score computed over the test set.

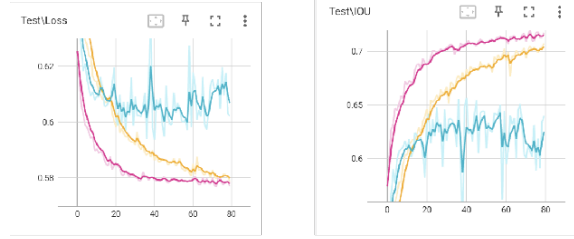


Figure 10. Testing curve. Left: Loss. Right: IoU. Baseline (blue). Baseline with FPN (pink). Baseline with DETR (yellow)

get the final fused 3D hidden state, which is then fed to the 3D decoder.

Note that in this variant the 3D-Convolutional GRU is replaced with the self-attention module, which means the update process becomes parallel instead of sequential. The overall architecture of this variant is illustrated in the Figure 9.

4. Experiment

All the models are trained for 80 epochs with a batch size of 32 and the training took 80 hours approximately. We used the same dataset provided in the official page of 3D-R2N2 with a split of 0.9 (training set) and 0.1 (test set). For this experiment, the number of input view images is set to 5 and the confidence threshold of whether a voxel is filled or not is set to $t = 0.5$.

Quantitative comparison From the Table 1 we can see that both FPN-based and DETR-based variants outperform the baseline with a 10% boost in the IoU score. The Figure 10 shows the testing curve of the 3 methods.

Qualitative comparison From the Table 2 we can see that both FPN and DETR based variants are able to capture smaller details and reconstruct global shape better.

5. Conclusion

In this project we proposed two variants of the 3D-R2N2, one based on FPN and another one based on DETR. While both variants outperform the baseline, from the Figure 10 we can see by the shape of the testing curve that the DETR based variant is definitely not converged yet. This is the common problem of the Transformers since they require

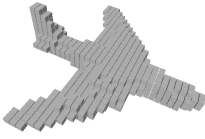
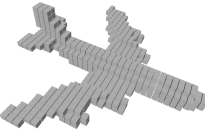
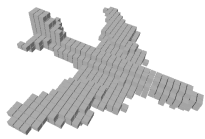
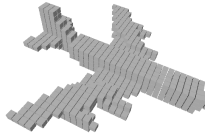
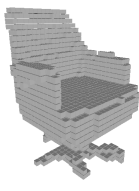
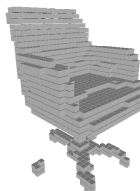
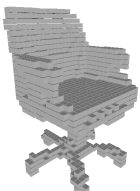





| Baseline | Baseline with FPN | Baseline with DETR | Ground Truth |
|---|---|--|---|
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table 2. Examples of multi-view 3D reconstruction from the ShapeNet dataset.

large training time. But, even though it is still not converged, the performance is already close to the FPN based variant, which is already fully converged. Thus, our guess is that, once fully converged the Transformer based approach should have best performance.

There are also other modifications that could potentially lead to better performance. For example, in the FPN based variant, instead of averaging feature vectors extracted from different scales after the self-attention module, one can simply concatenate them and let the 3D-Convolutional GRU module to choose the best features instead.

References

- [1] JunYoung Gwak Kevin Chen Silvio Savarese Christopher B. Choy, Danfei Xu. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. 2016. 1
- [2] Ross Girshick Kaiming He Lin., Piotr Dollár. Feature pyramid networks for object detection. 2017. 2
- [3] Gabriel Synnaeve Nicolas Usunier Alexander Kirillov Nicolas Carion, Francisco Massa and Sergey Zagoruyko. End-to-end object detection with transformers. 2020. 2
- [4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 2