Learning to Generate Chairs with Convolutional Neural Networks

Benjamin Ahlbrand
New York University
ba1404@nyu.edu

Hui Wei
New York University
hw1666@nyu.edu

Abstract

This work presents a differentiable renderer using convolutional neural networks in order to render 3D CAD models that are available online. Given some sparse sampling of discrete azimuth and elevation angle views of each object, it renders a 2D representation. This network is also capable of finding correspondences between objects of the same type in order to have a meaningful interpolation between them, in other words, semantically stable.

Figure 1. class 1, $\phi : 0°$, $\theta : 37°$

1. Introduction

We implemented the paper Learning to Generate Chairs by Dosovitskiy et. al.[4] in PyTorch. The only implementation available was the author's in Caffe with Lua, so we set off to create a clean open source implementation of the work. The work is an abstract differentiable renderer (and to our knowledge one of the earliest useful examples) where a network is provided with an object class, and camera intrinsics, it does a one shot through the network - rendering a 3D model. The novelty of this paper is that instead of performing the rasterization and the rest of the traditional pipeline, you show it a sufficient number of examples, and the network can imagine the views in between. Some weaknesses seem to be in 1) the resolution of our (and the author's) implementation, and 2) in capturing details not shown in the training dataset provided. Of course the latter is a reasonable shortcoming, as otherwise the network wouldn't have a concept of what would be underneath the arm of the chair.

2. Related Work

Differentiable rendering is a rapidly growing area of interest, at the intersection of graphics and machine learning (and work deriving from recent progress in computer vision). DeepMind published a popular paper recently in Science, called Generative Query Networks [5] as well as things like RenderNet[6] demonstrate the incredible potential of differentiable rendering in general. Instead of an explicit scene representation, where you need to specify and rasterize and perform elaborate computations of natural (or unnatural in the case of non-photorealistic rendering - which is quite unexplored to the best of our knowledge) phenomena - these allow a neural network to learn an abstract sense of the scenes and potentially offload otherwise impossible real-time renderings to less powerful hardware. There is also potential for using these techniques in order to synthesize datasets for other more complex neural networks to be trained. One use case of this is to compute a loss when training. In a recent paper on SVBRDF reconstruction[3], their network creates four texture maps (diffuse albedo, specular albedo, roughness, normal), but they were unable to compute loss across all four texture maps successfully. The target normals and the inferred normals were close enough for the loss curve to progress, but there was still a perceptible different to the human eye when actually rendered to lit surfaces. They then were able to compute this loss in the lit rendered space and then backprop through their original network successfully. So this complex juxtaposition of many parameters fed back to another network turns out to have immense potential. Generative Query Networks seem to learn to be able to predict view independence at a much deeper level than our implementation, inferring arbitrary views from an abstract description - somewhat similar to this work.

Figure 2. class 678, $\phi : 50°$, $\theta : 256°$

Figure 3. class 800, $\phi : 20°$, $\theta : 200°$

3. Implementation Detail

3.1. Preprocessing the Dataset

Since masks for the rendered chair images were not given by the original dataset, we generated them by assigning 0 to the white background pixels and 1 to chairs. At the same time, due to the input structure for our network and messed-up file name for each class, we first constructed a filename2idx dictionary, like word2idx in word embedding, to store the corresponding relationships between each class and its index varied from 1 to 809. In addition, we converted two kinds of viewpoint angles, azimuth angles θ, and elevation angles θ, to view vectors of length 4: $[\sin(\theta/180\pi), \cos(\theta/180\pi), \sin(\phi/180\pi), \cos(\phi/180\pi)]$. Since in the training set, transformed images are not provided, for transform parameter vector, we only feed in a ones vector for each training sample.

3.2. Model

PyTorch was used for constructing the entire model because of its more modern interface (as the field has progressed rapidly in the past few years), intuitive ways and good performance. According to 5, 2D convolutional layers and transposed convolutional layers were used. Specifically, we could not get the correct result using the log Softmax and negative log likelihood loss provided in PyTorch for segmentation part, we switched to sigmoid and binary cross entropy loss, which is equivalent.

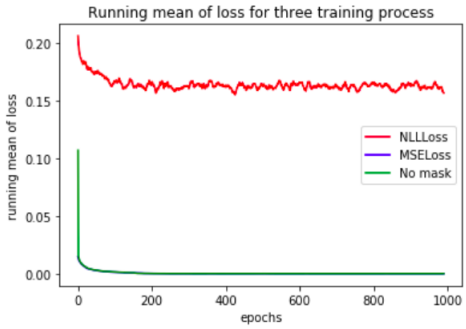

Figure 4. running mean of loss curve

3.3. Training

For the results shown, we trained our network for 1000 epochs taking 5 days, and saw little improvement beyond a point early on - we demonstrate epoch 417 in our ablation experiment. The loss function is compromised of a combination of mean-squared error for the target image, and a second loss function for the segmentation mask, scaled by a factor λ, which measures the relative importance between them. The running mean of the learning curve over the trained epochs can be observed from figure 4.

For training details, Adam with momentum $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and the regularization parameter $\epsilon = 10^{-6}$ were utilized for optimization. We used mini-batch size of 128 and set the initial learning rate to be 0.0005, with the help of reduce learning rate on plateau scheduler with factor 0.5 and patience 2 to adjust the learning rate automatically according to the validation error. Also, as suggested in the original paper, we initialized the weight for convolutional and linear layers with kaiming normal distribution with fan out mode and ReLU non-linearity.

4. Network Architecture

Refer to Figure 5 for a visualization of the network's architecture from the original paper. For all the convolution layers, kernel size of 3 and padding of 1 were used. Transposed conv layers (also known as de-convolutional layers) of kernel size 4 and stride 2 were for upsampling the feature map size to recover it to the image size in the dataset. Both of theses are shown in the dots part in 5. Moreover, ReLU were added after each layer to add the non-linearity. The network accepts as input, a set of classes representing the object's type, a view vector representing the axes angles, and a set of translation parameters. Pass each of these given parameters to individual fully connected layers of size 512, and concatenate the output together, to feed to more fully connected layers of size 1024. Please note that for the colored image output, the output channel for that branch is 3, while one channel is for masks since they are grayscale. This network is a rather lightweight architecture

density wise from our experience, the up-convolution process is quite fascinating since we've studied classification process much more deeply, seems obvious in retrospect, but flipping the architecture upside down in order to achieve a generative approach is rather powerful.

5. Evaluation

5.1. Ablation Study

We chose to evaluate our model based on three different metrics of the loss function for the segmentation mask, with mean-squared error loss, cross-entropy loss per pixel, and without use of the segmentation masks entirely. We found the segmentation mask is mostly unnecessary in general - but does start to introduce small artifacts without it, so it appears to be necessary for the finishing touches of the rendering. For the image visualizations over various viewpoints (see figures 1, 2, 3), it's a bit difficult to separate artifacts that result from the compression, versus the blitting / scaling from matplotlib, the additional scaling from Latex, and the network itself. We can rest assured that each image is created the same way so, we observe the same 'shadows on the wall', and can make some judgement calls on what impact the different decisions had on the results. All theta and phi angles in the displayed results do not appear in our training set (i.e. most of these examples are angles between the extrema), thus this demonstrates the interpolation the network performs between views - so you can see some artifacts around the legs due to missing some of these finer details. One general observation is that our model can create new view angles instead of just remembering the training data provided.

5.2. Comparison

We discuss a comparison of the different experiments for the ablation study. One might notice, that not using a segmentation mask on this particular example seems to thin out and bend the front right chair leg somewhat.

Some general observations we noted were:

- NLLLoss for mask: high-quality masks, coarser images
- MSELoss for mask: coarser masks, smoother images
- no mask: smoother images, but subtle issues with the finer details along the edges of model

5.3. Discussion

In figure 6, we attempted to normalize based on the min and max values the network was outputting. Once we clamped the values to 0 and 1, the issue was resolved as we rescaled the color values to the typical 0 to 255 as 8 bit color channels typically expect. The resolution of 128 by 128 pixels is quite limited, but the interesting work was getting this up and running, as opposed to scaling to high quality 4096 pixel resolutions, which would be interesting to see how this scales once you add physically based renderings. We discussed using a nonlinear layer at the top, like tanh, in order to smooth things out, but that would lose color accuracy, and we were able to fix it with our visualization method to begin with. Viewing angles beyond those seen in the dataset appear to work reasonably well, but you can observe some feature loss at the corners, where the viewing angles didn't project light / thus aren't seen by the network at any point.

5.4. Dataset

We divided the original dataset of chair renderings gathered from the paper Seeing 3D Chairs[1], into training and validation, whose ratio is 3:1. In the original dataset, they provided 1393 rendered classes of chairs, each class includes 2 elevation angles (20° and 30°) and 31 azimuth angles (from 0°to 348°with the step of 11°), thus 62 viewpoints in total. According to the Learning to Generate Chairs[4], we randomly selected 809 out of 1393 categories as the training and validation sets. The examples are in figure 10.

6. Future Work

6.1. Extensions

Since we wrote a 3D renderer in Python, to run from the command line in order to iterate over a set of 3D models in a folder, and render from various angles and transformation parameters, it would be a trivial extension to add transformations, colors and zooming. To add lighting parameters, you'd just need to concatenate that into the network at the beginning layer and expand each layer accordingly. Furthermore, we didn't add interpolation between different object classes of the same type - such as blending between different chairs to generate entirely new objects.

6.2. Directions

Another interesting direction for this work, is exploring more complex scenes, where the render time for a scene is higher than that of the inference cost for this network. For instance, could this learn complex interactions between reflections / refractions and indirect lighting bounces? Given the complexity of optical physics, it would be an interesting exploration to see if instead of computing several individual parameters of the rendering equation, if it's possible to learn a more abstract representation for different physically based lighting models, and that aren't real-time approximations.

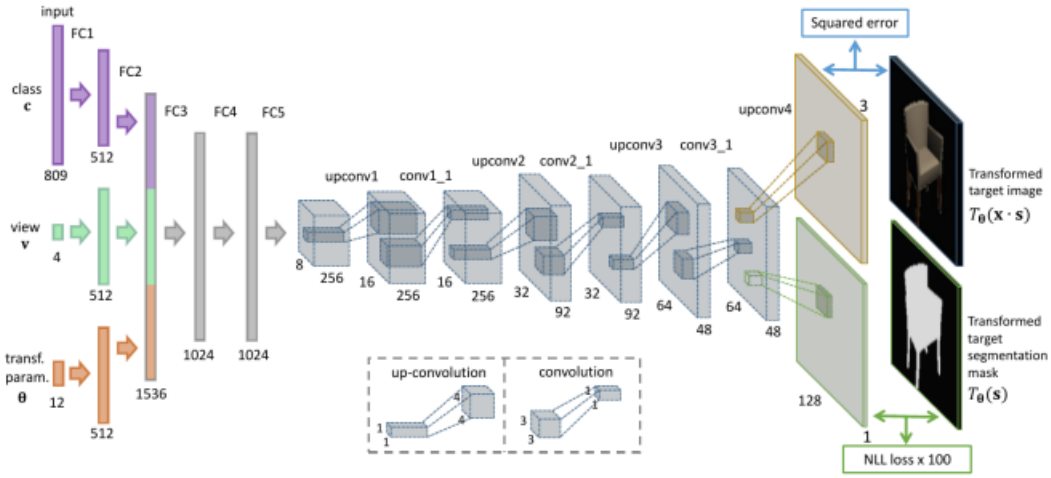

Figure 5. upside-down CNN architecture

Figure 6. Raw Normalization

6.3. Network Improvements

It would be interesting to explore this space with a generative adversarial network, or to see if any work has been done on this. A convolutional encoder with a classifier trained on the objects being generated could sharpen the features. Perhaps something along the lines of [2] would catch the sharper features across different resolutions due to the use of laplacian pyramids to pull features more consistently across viewing angles. Another avenue to explore is optimal structure for when expanding the network's memory across layers for additional parameters, is it strictly necessary to continue to add neurons or is there a better inherent structure that could be added which would be more effective in learning this mapping?

Figure 7. ablation: mean-squared loss, $\lambda = 100$. [left: image, right: mask]

References

[1] M. Aubry, D. Maturana, A. Efros, B. Russell, and J. Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. 2014. 3

[2] P. Bojanowski, A. Joulin, D. Lopez-Pas, and A. Szlam. Op-

Figure 8. ablation: cross-entropy loss, $\lambda = 0.01$ [left: image, right: mask]

Figure 9. ablation: no mask

timizing the latent space of generative networks. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 600–609, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 4

[3] V. Deschaintre, M. Aittala, F. Durand, G. Drettakis, and A. Bousseau. Single-image svbrdf capture with a rendering-aware deep network. *ACM Trans. Graph.*, 37(4):128:1–128:15, July 2018. 1

[4] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. *CoRR*, abs/1411.5928, 2014. 1, 3

[5] S. M. A. Eslami, D. Jimenez Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, D. P. Reichert, L. Buesing, T. Weber, O. Vinyals, D. Rosenbaum, N. Rabinowitz, H. King, C. Hillier, M. Botvinick, D. Wierstra, K. Kavukcuoglu, and D. Hassabis. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. 1

[6] T. Nguyen-Phuoc, C. Li, S. Balaban, and Y. Yang. Rendernet: A deep convolutional network for differentiable rendering from 3d shapes. *CoRR*, abs/1806.06575, 2018. 1

Figure 10. dataset examples. [left: image, right: mask]