Independent Work Final Report, Spring 2019

3D Shape Estimation with User Input

Samuel Arnesen '20
Adviser: Prof. Jia Deng

Abstract

This paper investigates whether supplementing a partial depth map with a small amount of user input could allow a convolutional neural network to more accurately estimate the full shape of a 3D object. Our proposed user-input model was able to slightly outperform the baseline model, suggesting that there does exist a potential for user input to be helpful for the task of 3D shape estimation. These results could be useful in developing an application to allow an individual to convert a personal two-dimensional photograph into a fully-realized three-dimensional scene.

1 Introduction

The field of 3D shape estimation chiefly concerns itself with the question of how best to approximate the complete form of a three-dimensional object given that part of the object is obscured or corrupted in some way. At one level, this task is impossible to perform completely accurately as two objects may be identical in their revealed regions but differ in their obscured regions. For this reason, information beyond just a partial view of an object is necessary to fully reconstruct its shape. However, this is not to say that the problem is completely intractable for partial scans of an object can nonetheless still convey meaningful information about what the obscured region is likely to look like. After all, the human brain implicitly performs this task every waking moment: despite being only able to actually see the front-side of an object, humans generally have enough contextual knowledge to confidently predict what the backside of the object is likely to resemble.

This problem has relevance beyond just intellectual inquiry. Notably, accurate 3D shape estimation enables robots to more reliably calculate the best way to grasp and pick up an object [1]. Similarly, 3D shape estimation could also enable self-driving cars to better reason about their surroundings as their knowledge of upcoming obstacles is less limited by the vehicle's singular viewpoint [2]. 3D shape estimation also plays a role in research and gaming tasks that require the generation of a large library of fully-realized 3D objects. For instance, the artificial worlds of video games must be populated by innumerable plausible 3D objects to achieve any semblance of realism. Likewise, researchers might also desire an artificial world strewn with plausible 3D objects when simulating how a new program or device would operate out in the real world.

However, most pertinent to the discussion that follows are the more personal use-cases of 3D shape estimation. That is, there is undoubtedly an interest on the part of families, friend groups, and photographers to turn their 2D photos into fully-realized 3D scenes. Such an application could allow people to more fully capture a moment as it appears in their memory.

Fortunately, highly successful methods already exist to allow the reconstruction of the shape of the front-side of an object from just a single-view photograph [3, 4, 5]. The question then arises as to how to translate that front-side partial scan of the object into a complete 3D object. Already, some techniques exist to perform this task [6, 7, 8]. Although the particulars vary, these methods all take in as input just a partially reconstructed 3D shape and output an approximation of the full shape. For complete generalizability, it is important that no extra input is required as there should be no expectation that any other relevant information is available.

However, for the specific case of 3D shape estimation for personal photographs, this restriction can be relaxed. In these use-cases, not only is it reasonable to assume that the person performing the shape reconstruction has additional contextual knowledge about the backside of the shape but they are also unlikely to be demanding shape estimation at a scale such that they cannot expend a little extra effort to input small amounts of additional information. It is thus the purpose of this project to design a means of taking advantage of this extra information to better construct complete 3D objects and in doing so, determine if a realistic photograph-to-3D-scene application could exist without the need for advancement in more general 3D shape estimation techniques.

More precisely, we designed a 3D convolutional neural network (CNN) with an encoder-decoder architecture that takes in as input a partial view of the front-side of an object, a mask indicating known and unknown regions, and a minimal amount of user input and outputs an estimation of the full 3D shape. The specific form of user input that we included was a small collection of points (N=20) on and off the backside of the object. Ultimately, we found that this user input did empower our network to slightly outperform a baseline model with no user input.

The rest of the paper will be as follows. Section 2 describes related works and recent advancements in the field. In particular, Section 2.1 describes research into the task of image inpainting, a research field that provided the conceptual foundation for later research into 3D shape estimation. Section 2.2 then explicitly investigates the most relevant research in the field of 3D shape estimation.

Section 3 reviews the overall architecture of our CNN. Section 3.1 describes the data used as input and output, Section 3.2 describes the form of the added user input, and Section 3.3

describes the loss function used for training. Finally, Section 3.4 describes the encoder-decoder structure of our network as well as additional salient features of the overall design.

Section 4 discusses the specific implementation of our network. In section 4.1, we discuss the code and how it was run. In section 4.2, we dive more explicitly into the various ways the user input was simulated in order to train our network.

Section 5 concerns the evaluation of our network. Section 5.1 focuses on the specific parameters used in training and testing while Section 5.2 reports the results and their implications.

2 Related Works

This section summarizes the relevant existing research with Section 2.1 reviewing the field of image inpainting and Section 2.2 discussing 3D shape estimation more directly.

2.1 Image Inpainting

Broadly speaking, image inpainting refers to the problem of how to best fill in missing pixels in an image. The most common usage of image inpainting techniques is in photo editing when a user wants to remove an object in the foreground from the picture. Such an edit creates a large block of pixels that now need to be filled in with an estimation of what would have been there had that previous foreground object not existed. This task is broadly similar to the task of 3D shape estimation: both concern themselves with using contextual clues to approximate features of an object or scene's obscured regions. However, most of image inpainting's advances come well prior to the maturation of the field of 3D shape estimation. This is because a) image inpainting has more self-evident real-world applications b) 2D photos are more widely available than 3D shapes and c) the number of terms that need to be estimated are smaller in two dimensions. As a result, the field of 3D shape estimation borrows heavily from the image inpainting research community.

Early scholarship in image inpainting focused on using geometric techniques to approximate the missing pixels [9]. That is, they made the assumption that patterns that existed elsewhere in the image were likely to be repeated in the missing region. The obvious downside of such a method is that this assumption is actually quite implausible in the real world: a foreground object may very well be blocking a unique pixel configuration that has no direct analog elsewhere in the image. Criminisi et al (2004) used a somewhat more sophisticated method where they propagated pixels from surrounding regions, creating novel configurations by using an estimation of their confidence in the inferred pixel's accuracy as part of their procedure [9]. However, this method too was limited by their exclusive use of information found in the image itself.

With the success of AlexNet in 2012 [10], convolutional neural networks and deep learning started getting applied to the task of image inpainting. Pathak et al (2016), for example, built an autoencoder that successively convolved the original corrupted image into a single, long vector before deconvolving it back to the original dimensions of the image [11]. Importantly, following Goodfellow (2014) [12], they then appended a generative adversarial network (GAN) to the end of their design and their loss function was a combination of the reconstructive loss and the GAN loss. A GAN is a neural network trained to differentiate between real images and synthetic images so including a GAN at the end of their network ensured that the outputted images were less blurry and harder to distinguish from real images. Unfortunately, due to the instability inherent to training a GAN and the number of parameters that need to be tuned, GANs have yet to be applied to 3D shape estimation [8], although they have been applied to other tasks in three dimensions [24].

At the moment, the state-of-the-art image inpainting algorithm belongs to Yu et al (2018) [13]. Their works combines the autoencoder+GAN architecture of Pathak et al. (2016) with a novel contextual attention layer whose kernels are initialized with patches from the rest of the image. The logic behind this move is that patterns that appear in the rest of the image still convey meaningful information about what is likely to be in the obscured region even if the early attempts that relied exclusively on this reasoning were inadequate. Contextual attention layers have also not yet been applied to 3D shape estimation.

2.2 3D Shape Estimation

In one of the earliest examples of divergence from the field of image inpainting, Rock et al. (2014) tackled the problem of 3D shape estimation by performing a nearest-neighbor search for similar objects and then filling out the obscured region of the object with the best database match [14]. The obvious drawback of such a method is that it cannot synthesize novel shapes and it relies on an enormously comprehensive shape database for even a semblance of accuracy.

Two years later, Firman et al. (2016) used a random forest algorithm on a small, custom dataset of everyday household objects to perform shape estimation [15]. These results improved on those of earlier non-convolutional methods but it was limited by its small training sample and inability to pick up on high-level features.

One of the earliest papers to apply CNNs to the task of 3D shape reconstruction actually predates Firman et al. and it was an attempt by Wu et al. (2014) to use shape estimation to aid in 3D object classification [16]. Wu et al.'s reasoning was that classifying an object properly and reconstructing its entire shape would require a CNN to pick up on similar high-level features and as a result, the same model could be used to perform both those tasks. Although their results would shortly be improved on by networks that more specifically concentrated on the task of 3D shape completion, their work nonetheless still demonstrated an important proof-of-concept that CNNs were a viable tool to perform tasks such as shape estimation.

Despite these advances, for the next few years, most shape-estimation deep learning systems required additional inputs beyond a single view depth map. For example, Girdhar et al (2016) required both a partial 3D voxel representation of the object and an additional photo to produce their estimate [17]. Similarly, Choy et al. (2016) required multiple views of the same object to be able to reliably approximate an object's obscured regions (although they did make an attempt at single-view reconstruction) [18].

The most significant recent advance in the field comes from the Stanford team of Angela Dai, Charles Ruizhongtai Qi, and Matthias Niessner [8]. They produced a network with three notable elements. First, they took advantage of the encoder-decoder structure that was successful in the 2D case with Pathak et al to produce their estimates. Second, they trained a parallel network to perform a classification operation and the resulting probability vector was appended to the encoded vector at the bottleneck layer in the network. The logic behind this decision was that an object's class has relevance as to what an object's backside is likely to look like for the straightforward reason that objects of the same class are likely to have similar backside shapes. Finally, they used a nearest neighbor search to confront the problem of low-resolutionality. That is, a frequent problem when working with three-dimensional shapes is that the input space can become intractably large at high resolutions. As a result, most shapes are represented at low-resolutions along the lines of 32x32x32. Dai et al. addressed this problem by upsampling their low-resolution estimate to 128x128x128 and then swapping out parts of their reconstructed object with high-resolution parts from a large object database so that the object loses its blurriness. Our model will borrow substantially from Dai et al.'s architecture, with the final classification and synthesis steps being omitted for the sake of simplicity.

3 Architecture

The following section describes the overall architecture of our model. In Section 3.1, we discuss the formatting and justification for the choice of input and output. In Section 3.2, we explore the user input layer and the logic behind how it should work. In Section 3.3, we briefly discuss the choice of loss function. Finally, in Section 3.4 we discuss the overall structure of the network.

3.1 Input/Output

The data source we used for our input was the ShapeNet Partial Image Scan Database [8]. The general ShapeNet database is a dataset of 3D CAD models that covers 3,135 different classes [25]. Dai et al (2017) obtained a partial front-side scan of 25,590 of their objects and converted them into an hd5 format for their Partial Image Scan Dataset.

Every object has three voxel grids (three dimensional tensors) associated with it, each 32x32x32. The first is a tensor of signed distance fields. That is, each element represents the distance from that box to the surface of the object, with known empty space encoded as a positive number and unknown space encoded as a negative number (voxels on the surface of the object are encoded as 0). The second is a binary mask indicating which voxel grid entries are known and unknown (1 for known, -1 for unknown). The last is the ground-truth: a voxel grid of unsigned distance fields (it is unsigned since all values are known and thus there is no need to encode unknown values as negative).

For our network, we concatenated together the signed distance field voxel grid and the binary mask and fed it in as input. The logic behind this decision is that both the signed distance field and the binary mask encode information relevant to the final shape estimation. The signed distance field of course provides the information as to the actual shape of the object as the voxel entries with zeros in them fully describe the outline of the object (in practice, no entry has a value of exactly 0 but entries with very small magnitudes nonetheless still offer a rough definition of the shape). The binary mask meanwhile tells the network which values to consider and which not to consider.

The real judgement call regards whether using a signed distance field is truly the best way to encode the shape of an object. After all, a simpler way exists to define an object's shape: encode the entry as a 0 if it is known to be off the object, 1 if it is known to be on the object, and a -1 if it is unknown. This would have the advantage of both simplicity and intelligibility. However, this option was ultimately rejected for two reasons. First, in experiments by Dai et al., using the signed distance field as input outperformed other possible input formats. Second, there is a loss in resolution when the object boundary gets stretched to the width of an entire voxel. Together these two factors suggested it was more prudent to use a signed distance field as input.

By comparison, the decision regarding the output was much simpler: use the same unsigned distance field as the ground-truth. Although in theory processing could be performed on the ground-truth to convert it into some other format, keeping it as an unsigned distance field has the key advantage that it allows us to ultimately compare our results to the results achieved by Dai et al and others in the field.

3.2 User Input

At a very top-level, the form of user input that we used was a collection of 20 points on and off the backside of the object. Were this to be deployed in real-life, the user would be the one selecting the 20 points from their knowledge of what the object actually looks like. In the case of, say, a parent trying to convert a family photograph into a three-dimensional scene, this demand of the user is quite reasonable. Not only are they likely to have contextual knowledge as to shape of the object's backside, but they can also spare the time and effort to select the 20 points. In a commercial or research setting where 3D objects must be produced at scale, such an expectation may not be reasonable. Nonetheless, the success (or failure) of this kind of user input can still provide meaningful information regarding whether a scaled-down version of this input requirement could be viable for these more demanding shape refinement tasks.

In theory, the reason why this kind of user input could be useful is that even a highly sparse selection of points on the backside of the object can provide meaningful clues as to the shape of the object. For example, very few objects have unexplained holes cut out of their middle. As a result, if a point on the backside of the object is labelled as "empty," it is now substantially more likely that the regions behind it are empty as well. There is of course a risk that despite the potential for meaningful information to be encoded in the set of user input, the convolutional filters will only pick up on larger features and disregard this sparse input. One potential implication of our results then will not only be whether this particular form of user input can noticeably improve shape reconstruction accuracy but also what type of features are being identified by our convolutional filters.

The user input is not added as a separate layer to the input but rather it is just included as a modification to the signed distance field tensor. That is to say, the twenty entries that were selected have their values in the signed distance-field tensor replaced with the value found in the ground-truth tensor. The binary mask is also updated to reflect that these values are now known.

3.2 Loss Function

For our loss function, we chose the average L1 norm between the ground-truth and our predicted output. The reason for this decision was that it allows us to better compare our results to Dai et al. (2017) [8] and other older results like Wu et al (2016) [16] and Kazhdan et al (2016) [19].

Importantly, we also masked the output to consider only the part of the object that was previously obscured. Although this undoubtedly means that the entirety of the predicted output does not track the actual ground truth particularly well – the originally-known front-side of the object is unlikely to be reconstructed especially accurately – this is not a major concern since in post-production we can merge the originally known values from the input with the predicted unknown values to get a fully-realized 3D object. Masking the loss function ultimately allows for a more accurate reconstruction of the backside of the object because it means that the limited number of filters being trained can isolate in on the features that are relevant to estimating the shape of the backside and do not need to pick up on features that may only be relevant for front-side reconstruction.

3.3 Network Architecture

The overall architecture used in our CNN roughly tracks the encoder-decoder architecture used by Dai et al (2017) [8]. More precisely, it begins with five convolutional layers, followed by two fully-connected layers, and finally five deconvolutional layers. Because we used filters of dimension 4x4x4 and a stride of 2, during each of the convolutional layers, the input gets shrunk in half. However, the number of filters at each layer also increases as we get deeper into the network. As a result, after the fifth convolutional layer, we are left with a single-dimensional length-320 vector. It is the deconvolutional layers that allow us to end with the correct output dimensions. Each of the deconvolutional layers also have filters of 4x4x4 and strides of 2 so each of the layers doubles the size of the input. At the end of the fifth deconvolutional layer then, we are left with a single 32x32x32 outputted prediction.

This kind of bottleneck structure is commonly used to perform other tasks like image denoising [20]. The rough idea is that the vector produced in the bottleneck layer acts like a highly-compact compression of the original input. Each of the entries of that vector then is basically the degree to which a specific high-level feature is present in the input. For denoising, the applicability is straightforward as random noise would almost certainly be lost in the compression. As a result, after decompressing the vector during the deconvolutional stages, one is left with an image free of noise. For 3D shape estimation, the effect is a little subtler. The general idea is that the high-level features that comprise the elements of the compressed vector are all features that are salient to the reconstruction of the object's backside. Individual idiosyncrasies of the object's front-side get lost in the compression and only the major design elements ultimately influence the final prediction. Since it is likely that only major features of an object are truly indicative of what an object's backside looks like, treating these smaller contours as just noise thus has the advantage of improving output accuracy.

The other major advantage of this bottleneck structure has to do with training time and memory usage. Hundreds if not thousands of high-level features are relevant to the estimation of an object's backside. To be sensitive to each of those features, one therefore needs hundreds of convolutional filters. If the input did not quickly get downsampled, the total memory used in the network would soon explode and the number of terms that would have to be updated during training would also increase enormously, slowing the training down noticeably. Moreover, in order to pick up on larger features without aggressive downsampling, one would require the use of larger kernels which would also create substantial memory and time costs.

Another important aspect of our architecture are skip connections. More specifically, for each of the deconvolutional layers, we concatenated the output from one of the convolutional layers to the end of the previous layer's output and then used that larger tensor as the input. The benefit of a skip connection is two-fold. First, it addresses the problem of the gradient declining to near-zero in the bottleneck layers, which would slow down learning immensely [21]. Second, it preserves some of the more minor features that were lost during the original convolutional stages but that potentially might have relevance to the ultimate shape estimation.

Lastly, we also include batch normalization after every convolutional and deconvolutional layer except for the final predictive layer. This is standard practice and helps decrease both the training time and the risk of overfitting [26].

4 Implementation

The section that follows will describe how our model was implemented. Section 4.1 will review the code used to produce our results while Section 4.2 will cover the specific parameters of our models with a special focus on how the user input was simulated.

4.1 Code

All code for this project was written in Python 2.7.12 with the TensorFlow 1.4.1 package. The code and data was stored on Google Cloud and run on a single NVIDIA Tesla K80 GPU using Google Cloud Platform's Compute Engine.

4.2 Models

This project has three models associated with it. The first, a baseline model, can fairly accurately be described as a stripped-down version of the state-of-the-art system built by Dai et al (2017). The second, the “random-user-input model” (RUI), follows the same design as the baseline model but included in the input are 20 points randomly selected from the obscured part of the object. The final model, the “smart-user-input model” (SUI), is very similar to the RUI model except that the user input is not selected randomly from the entire obscured region but rather from the specific part of the obscured region that is near the object boundary.

As described in Section 3.3, the baseline model contains five convolutional layers, 2 fully-connected layers, and five deconvolutional layers. This broadly follows the design of Dai et al. There are however a few key differences. Most notably, the parallel classification network and the synthesis model they appended to the end of the network are omitted. Both networks were left out for the simple reason that they would require increasing training time dramatically. Moreover, our network uses half the number of filters as Dai et al. So, for example, where their bottleneck vector is length 640, ours is only length 320. This decision was also made with an eye towards training time and memory usage. The upshot of this decision however is that even in the event where the additional user input does provide information that is meaningful to the training of the network, it is unlikely that we are going to be able to beat Dai et al.’s state-of-the-art results. However, since this baseline model still broadly follows Dai et al.’s design, if the RUI or SUI model significantly outperforms the baseline, then we can still be reasonably confident that the user input was in fact useful.

The RUI model operates very similarly to the baseline model except that 20 randomly selected points from the backside of the object are included in the input. This random-sampling process was done is a brute-force manner: we drew random x, y, and z coordinates from a uniform distribution and checked the given binary mask to see if the value of that voxel was unknown. If it was unknown, we assigned the value for the entry in the signed-distance-field tensor to be the same as the value in the ground-truth tensor and marked the voxel as known in binary mask. If the value was known, we did nothing. We then repeated this procedure until we found 20 points that were previously unknown. This model is meant to simulate the most-naïve way in which a person could enter their user input. This model then is designed to provide a floor on how poor the results that a model with user-input could achieve.

The SUI model offers a more intelligent approach to selecting user input. Instead of accepting every possible user-inputted voxel that was previously unknown, the procedure here only accepts inputted voxels that were both previously unknown and that are near the object boundary. In more precise terms, after randomly drawing a voxel from a uniform distribution, we assigned the value for that voxel in the signed-distance-field tensor to be the same as the value in the ground-truth tensor if and only if that value was previously unknown and the distance in the ground-truth tensor was less than 0.1. Since the values in the ground-truth vector represent the distance from that voxel to the object boundary, only including voxels with values less than 0.1 ensures that all the user input comes from voxels that are near the object edge. Whether one can reasonably expect a real-life user to have enough fine-grained knowledge of the object backside to input data with that level of precision is debatable. However, nonetheless, the success or failure of this model should still provide meaningful information regarding whether user input can be used to improve the accuracy of a 3D shape estimation network.

5 Evaluation

This section will consider the evaluation protocol for our three models. The first section, Section 5.1., will go into detail on the specific parameters used in the training and testing procedure. The final section, 5.2., will report and comment on the results.

5.1 Training and Testing Procedure

The batch size we used for the training and testing procedure was just 16. This was done out of a concern for memory usage and timing. Already, given the number of terms that must be inferred during training (~10 million), each epoch takes a substantial amount of time. Even with a batch size of just 16, just over seven minutes elapsed per epoch. Although an even smaller batch size would speed this up substantially, it would come at a cost of decreasing the representativeness of each mini-batch. Overall then, we decided that a batch size of 16 was the best balance for our specific purposes.

We trained our 3 models for 400 epochs each, meaning that the cumulative training procedure took just around 6 days. The choice of 400 was determined experimentally and was influenced by the observation that the overall loss improved steadily until about 350 iterations, after which it began to flatline. The downside of using only 400 epochs however is that each model ended up training on only 6400 shapes, a small fraction of the overall number of objects available.

The only other hyperparameter of note was our choice for the momentum used in batch normalization. We ultimately chose 0.9 in order to balance two considerations. The first consideration is that our smaller batch size should mean that we use a higher momentum so that the idiosyncrasies of each particular mini-batch do not overly influence the gradient descent calculation. The second consideration is that the relatively small number of epochs used in training should mean that we use a lower momentum so that convergence can happen before the maximum iteration limit is hit. We reasoned that 0.9 was an appropriate balance of these two factors and small experiments supported this intuition.

We decided to train using the Adam optimizer introduced by Kingma and Ba (2015) with a learning rate of 0.001 [22]. The Adam optimizer is one of the optimizers that is best-suited for computer vision tasks [23] in large part because it is adept at handling sparse inputs and noisy gradients [22]. The learning rate was chosen to match the learning rate of Dai et al. [8].

For our testing procedure, we ran our models on 1600 objects that were held out of the training data set. The specific 1600 number was chosen to accommodate the fact that the input objects largely fell into 8 different classes and so we chose a high multiple of 8 so that each class would equally influence the final result. The reported results below are an average of the loss for the 1600 tested objects.

5.2 Results

Our results are reported in the table below. As mentioned in Section 3.2, the loss here refers to the average per-voxel difference between the estimated and ground-truth distance field value for voxels whose distances were not previously known.

Table 1: Results

Model	L1 Distance
Dai et al (2017) [8]	0.309
Kazhdan et al (2016) [19]	1.91
Baseline Model	1.68
RUI Model	1.70
SUI Model	1.64

Notably, none of the three models achieved the loss reached by Dai et al (2017) [8] although all three were able to beat the geometric methods used by Kazhdan and Hoppe (2013) [19]. This is unsurprising. Dai et al (2017) have over twice as many learnable parameters, train on exactly 50% more objects, and include synthesis and classification modules that we omit [8]. As a result, the baseline model, which approximates a stripped-down version of Dai et al.'s model, should be our reference point for whether the inclusion of user input was useful to the final shape estimation.

Our RUI model did not outperform the baseline model (1.70 v. 1.68). Two factors likely contributed to this failure. First, the kind of features that the model was sensitive to were features that were much larger than a single voxel. As a result, the specific type of input added by the user was not especially useful to the model since it did not meaningfully change the filter responses. In some ways this is desirable since any partial scan of an object may have noise or errors associated with individual voxels and we would not want our ultimate shape estimation to be highly sensitive to that noise. Especially considering that this exact same encoder-decoder architecture is used for de-noising tasks, it was predictable that a random collection of individual voxels would not make a large difference in the model accuracy. The second reason for the underwhelming results of the RUI model has to do with the random nature of the input. At a minimum, the randomness means that it is difficult for the network to identify backside features that reliably predict the final shape of the object since the specific points selected by the user differ dramatically and arbitrarily from object to object. In the worst case, this could actually make training more difficult since two near-identical objects might have very different input representations if their user-inputted points were substantially different. Overall then, the sparsity and randomness of the user input doomed the RUI model to mediocrity.

Meanwhile, our SUI model did narrowly outperform both the RUI model and the baseline model. The difference was not particularly large but the number of test cases – 1600 – suggests that this differential was due to more than just random variation. The best explanation for the differing performances of the RUI and SUI models is that having the simulated user only select points that were near the object boundary meant that the model had much more relevant information at its disposal. This is in part because we suspect that the voxels near the object boundary are likely to be the voxels with the highest associated uncertainty. Whereas voxels far away from the object or in the middle of the object can likely be predicted with high degrees of confidence even without user input, voxels near the boundary are more likely to be approximated incorrectly. As a result, including this information provides the model with information it might actually need to make an accurate estimation. Compounding this effect further is the fact that boundary information can propagate throughout the model more effectively than other types of information. In short, if you know the outline of the object's shape, it is fairly simple to estimate the contents of the remaining voxels: the voxels inside the outline are likely to be filled whereas voxels outside the voxel are likely to be empty. These two considerations are what likely led to the improved performance of the SUI model. Unfortunately, the twin problems that plagued the RUI model – sparsity and unpredictability – also damaged the effectiveness of the SUI model. In sum then, the SUI model presents some advantages relative to the baseline model but not the results were not significantly different due to offsetting factors.

6 Conclusion

Overall, this project attempted to investigate whether adding user input could accelerate the existence of a realistic photograph-to-3D-scene application. By building a convolutional neural network that took in as input the front side of an object and a simulated selection of points on the backside of the object, we hoped to retrieve an accurate estimation of the entirety of the object's shape.

On this front, our results were mixed. One version of our model was able to achieve a better test-case loss than the baseline model that roughly approximated a bare-bones version of the state-of-the-art. However, the results were not dramatically different and it is unclear if the burden placed on the user –selecting 20 points very close to the object boundary – was realistic. To that end, we believe the strongest defensible claim we can advance is that our results demonstrate a potential for user input to improve the performance of 3D shape estimation CNNs.

The goal of developing a usable photograph-to-3D-scene application would benefit substantially from future research. One easy way to do this would be to use the exact architecture developed by Dai et al (2017) and then add the small collection of user inputted points to determine if this form of user input could actually improve on the state-of-the-art. There exists a potential concern that the improvement would be negligible given their network's existing level of sophistication however there also exists a possibility that their large increase in trainable parameters means that this network would actually be able to pick up on features hinted at by the user input.

Another possible direction for research could be to change the type of user input. For example, asking the user to provide a class label for the object might also improve the ultimate accuracy of the estimate. After all, we know from earlier works that concatenating an estimated class vector to the end of an encoded vector in the bottleneck layer of the encoder-decoder architecture can improve the results of the final estimate [8]. However, it remains an open question as to whether a user inputted class vector would be a substantial improvement over an estimated class vector given how accurate state-of-the-art classification techniques are already. Another way to alter the user input could be to have the user input a collection of voxel neighborhoods rather than just a collection of individual voxels (i.e. each input would be, say, 20 different 4x4x4 blocks of voxels instead 20 singular voxels). This could provide more information to the network since a 4x4x4 patch is large enough for a filter to pick up on real features.

Another speculative approach could be to try to teach a network to identify the most useful points for a user to provide their input. In production, this would appear to the user as if a heat map were overlaid onto the backside of the object where the most useful points for user input are clearly marked. Such a system could not only guarantee that the points inputted by the user are nontrivial but also provide greater certainty to the network as to the points that are to be inputted, which would in turn address the unpredictability problem that plagued both the RUI and SUI models. However, this is likely to be a longer-term goal and its technical feasibility is very much not yet established.

Ultimately, the results we found indicate that the goal of creating a photo-to-3D-scene application using user input is a realistic one. However, more work needs to be done on network architecture and user input selection before any such application can actually be realized.

Works Cited

[1] Agrawal, Amit, et al. "Vision Guided Robot System for Picking Objects by Casting Shadows." *Mitsubishi Electric Research Laboratories* (December 2009).

[2] Miao, Yanen et al. "Joint 3-D Shape Estimation and Landmark Localization From Monocular Cameras of Intelligent Vehicles." *IEEE Internet of Things* Vol 6. No. 1 (February 2019).

[3] Koch, Tobias. "Evaluation of CNN-based Single-Image Depth Estimation Methods." *arXiv* (3 May 2018).

[4] Saxena, Ashutosh, Sung H. Chung, and Andrew Y. Ng. "3-D Depth Reconstruction from a Single Still Image." (2007).

[5] Eigen, David, Christian Puhrsch, and Rob Fergus. "Depth Map Prediction from a Single Image using a Multi-Scale Deep Network." (2014).

[6] Wu, Zhirong, et al. "3D ShapeNets: A Deep Representation for Volumetric Shapes." (2014).

[7] Choy, Christopher, et al. "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction." (2 April 2016).

[8] Dai, Angela, Charles Ruizhongtai Qi, and Matthias Niessner. "Shape Completion using 3D-Encoder-Predictor CNNs and Shape Synthesis." *arXiv* (11 April 2017).

[9] Criminisi, A., P. Perez, and K. Toyama. "Region Filling and Object Removal by Exemplar-Based Image Inpainting." *IEEE Transactions on Image Processing* Vol. 13 No. 9 (Sept. 2004).

[10] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." (2012).

[11] Pathak, Deepak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, Alexei Efros. "Context Encoders: Feature Learning by Inpainting." (2016).

[12] Goodfellow, Ian, et al. "Generative Adversarial Networks." (10 June 2014).

[13] Yu, et al. "Generative Image Inpainting with Contextual Attention." (21 March 2018).

[14] Rock, Jason et al. "Completing 3D Object Shape from One Depth Image." (2014).

[15] Firman, Michael, Mac Aodha, Simon Julier, Gabriel Brostow. "Structured Prediction of Unobserved Voxels from a Single Depth Image." (2016).

[16] Wu, Zhirong, et al. "3D ShapeNets: A Deep Representation for Volumetric Shapes." (2014).

[17] Girdhar, Rohit et al. “Learning a Predictable and Generative Vector Representation for Objects.” (31 August 2016).

[18] Choy, Christopher B., Danfei Xu, JunYoung Gwak, Kevin Chen, Silvio Savarese. “3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction.” (2 April 2016).

[19] M. Kazhdan, M. Bolitho, H. Hoppe. “Poisson Surface Reconstruction.” *Eurographics Symposium on Geometry Processing* (2016).

[20] Vincent, Pascal, Hugo Larochelle, Yoshua Bengio, Pierre-Antoine Manzagol. “Extracting and Composing Robust Features with Denoising Autoencoders.” (2008).

[21] Balduzzi, David et al. “The Shattered Gradients Problem: If resnets are the answer, then what is the question?” (6 June 2018).

[22] Kingma, Diederik and Jimmy Lei Ba. “Adam: A Method for Stochastic Optimization.” *ICLR* (2015).

[23] “Introduction to Optimizers.” *Algorithmia* (7 May 2018).

[24] Wu, Jiajun, et al. “Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling.” (4 January 2017).

[25] Chang, Angel et al. “ShapeNet: An Information-Rich 3D Model Repository.” *Shapenet* (2015).

[26] Ioffe, Sergey and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” (2 March 2015).

I pledge my honor that I did not violate the Honor Code during this completion of this independent work.