Monocular Depth Estimation by Deep Learning

Xingwei Liu xil128@eng.ucsd.edu
Chunyi Lyu lchunyi@ucsd.edu
Jiawei Li jil206@ucsd.edu

Abstract

In this paper we are focusing on analyzing the model to get 3D information from 2D image, which means we want to get a depth map from just one single image. We will present you three models with different structure, these three structures doesn't give us an out-performed result, which could also due to the limitation of the time and resources, but they did show an increasing quality on performance which could lead us to some further analyze, we will have a further discussion on how and why they could behave like that in this report.

Figure 1: Results by Eigen *et al.*[3]. Each columns from left to right are original image, predicted large-scale depth, predicted finer-scale depth, ground truth.

1. Introduction

2D image is one of the easiest data we can get in our daily life, however in most of time it tells us less information than 3D data, therefore generating more 3D data information from 2D data becomes one of a very important research in computer field.

This is actually a quite popular topic in computer vision, but most of the traditional methods requires two or more images to generate the depth by point correspondences and triangulation, which require complex set-ups such as calibration, we hope the deep learning could generate the depth just based on one image without any rectification or calibration.

We have found several papers that corresponding to this field, most of them are using deep learning method to train the model, but, unlike the traditional multi-view method, we generate the information from a single view, resulting in uncertainty on pixel's depth. So, some of the result they provide doesn't looks good, the distance or the objects in the original image aren't present correctly.

In a recent work by Eigen *et al.*[3], as shown in Figure 1, the result depth not only are they very fuzzy, but also lose many information from the image. The result from the last row shows that this model failed to recognize the two tables near the camera. In other words, it thinks the tables are at the same distance as the bookshelf behind from the camera. The accuracy of predicted depth they gave are 69.2%.

To get as much information as possible, we focus on several improvement ideas. In order to improve the performance, we hope we can generate a depth map that can clearly distinguish the segments inside the image. The model in the paper we found, uses two CNN network to generate the feature processing, which down-sampled the image and filtered many details of the image, for example, in some of the result, the depth map doesn't shown the switch on the wall, this is very hard since it's very hard to make a model sensible to the distance between a switch and a wall, especially if that wall is far enough, this may be one of the problem that prevent former work to get better result.

We also found that to recognize the depth map, we usually needs a lot of image or further information other than image to make the accuracy higher. But in reality we often don't possess such complicate data, so in our model we will only use 2D image data, to see whether a lighter dataset could also generate a better result through our model.

Our contributions for this project are:

1. We discover the effect from different structure towards the image depth detection, and find a way to balance the affect from different features in the image

2. We develop a model that doesn't require massive pretrain, but can also detect the image depth.

3. We proposed a way to control gradient flow to train multi-scale model requiring different learning rate more easily.

2. Background

There are several approaches that estimate depth from a single image that related to our work.

Eigen *et al.*[3] show how to directly regress on the depth using a neural network with two components: one that first estimates the global structure of the scene, then a second that refines it using local information. However, this method can't get high resolution due to the lack of normal estimation.

Single-image surface normal estimation has been addressed by Fouhey *et al.*[5],Ladicky *et al.* [8], Wang *et al.* [10] and recently Eigen *et al.* [2]

Fouhey *et al.* match to discriminative local templates [10] followed by a global optimization on a grid drawn from vanishing point rays[6], while Ladicky *et al.* learn a regression from over-segmented regions to a discrete set of normals and mixture coefficients.Wang *et al.* use convolutional networks to combine normals estimates from local and global scales, while also employing cues from room layout, edge labels and vanishing points.

Prior work on semantic segmentation includes many different approaches, both using RGB-only data[1] [4] as well as RGB-D [9] [7]

3. Dataset

3.1. Data Description

The data we connect are called NYU-Depth V2 provided by the lab from NYU. The NYU-Depth V2 data set is comprised of video sequences from a variety of indoor scenes as recorded by both the RGB and Depth cameras from the Microsoft Kinect. The feature for the data is as follow:

- 1449 densely labeled pairs of aligned RGB and depth images
- 464 new scenes taken from 3 cities
- 407,024 new unlabeled frames
- Each object is labeled with a class and an instance number (cup1, cup2, cup3, etc)

The total datasets has 428 GB which is too big consider the time limitation, therefore we only choose the labeled datasets which is 2.8 GB, the data it provides is as in Figure 2. The left image is original image, middle image is depth image, the right image is classification image, since in this project we want to see whether we can have a better result without using any further information but just depth-map, we will only choose the first two image data.

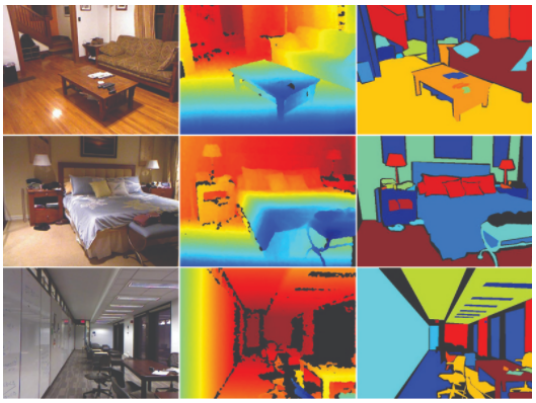


Figure 2: NYU Datasets

3.2. Preprocessing

We let our model predict the logarithm distance between camera and objects, so we convert the target depth into logarithm scale.

As what the normal training process did, we separate the data into two parts, 80% as training set, 20% as testing set.

The amount of data we get is not enough to make our model learn the depth map but rather remembering all the inputs, therefore, we augment the training data as following.

- Randomly scale image by $s \in [0.75, 1.25]$, also divide the depth by s to maintain geometric consistency.
- Randomly rotate image by $[-5, 5]$ degrees
- Randomly crop a 304×228 area from the image

While for the test set, we just do a center crop of size 304×228.

4. Evaluation

We use the following equation to calculate the difference between two pixel:

$$d_i = \log y_i - \log y_i^*$$

y_i represent the i th pixel on the predicted depth map, y^* represent the ground truth.Every time we calculate the error ,we flatter the image and calculate the difference between each pixel, the basic error function we used is scale-invariant mean squared error (in log space):

$$D(y, y^*) = \frac{1}{n}\sum_{i=1}^{n}(\log y_i - \log y_i^* + \alpha(y, y^*))^2$$

The α here is the value used to minimizes the error for a given (y, y^*), which we can represent as this:

$$\alpha(y, y^*) = \frac{1}{n}\sum_{i}(\log y_i^* - \log y_i)$$

It expresses the error by comparing relationships between pairs of pixels (i, j) in the output: to have low error,each pair of pixels in the prediction must differ in depth by an amount similar to that of the corresponding pair in the ground truth.

In addition to performance evaluation,we also tried using the scale-invariant error as a training loss:

$$L(y, y^*) = \frac{1}{n}\sum_i d_i^2 - \frac{\lambda}{n^2}(\sum_i d_i)^2$$

Note the output of the network is logy; that is, the final layer predicts the log depth. Setting = 0 reduces to element wise l2, while = 1 is the scale-invariant error exactly. We use the average of these, = 0.5,as what specified in report[1].

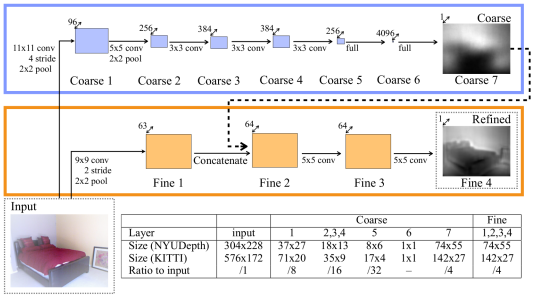
Then we need to find a way to judge how exactly our model performs, typically we are trying to find the which image has the minimum difference from the ground truth,

$$\begin{aligned}\Delta(y_i, y_i^*) &= \max(\frac{y_i}{y_i^*}, \frac{y_i^*}{y_i}) \\ &= \exp(\max(\log y_i - \log y_i^*, \log y_i^* - \log y_i)) \\ &= \exp|\log y_i - \log y_i^*|\end{aligned}$$

T represent the difference between the predict depth map and the ground truth depth map, we set threshold as 1.25 which means, every predicted result with difference below 1.25 can be consider as successfully predicted, then we calculate for each set of data, how many prediction can be consider as success, then we calculate the accuracy. The whole formula is shown below:

$$\text{Accuracy} = \text{Percentage of } y_i \text{ s.t. } \Delta(y_i, y_i^*) < \delta$$

5. Model



Layer	input	Coarse 1	Coarse 2,3,4	Coarse 5	Coarse 6	Coarse 7	Fine 1,2,3,4
Size (NYUDepth)	304x228	37x27	18x13	8x6	1x1	74x55	74x55
Size (KITTI)	576x172	71x20	35x9	17x4	1x1	142x27	142x27
Ratio to input	/1	/8	/16	/32	–	/4	/4

Figure 3: Network structure by Eigen *et al.*[3]

Based on the CNN model by Eigen *et al.*[3] shown in Figure 4, we can see from section 1 that it doesn't give a very good result, and the way it train the model is to firs train the Coarse part, than use the pretrain data to retrain the Refine part, which we think is not that efficient and doesn't really perform well when we test it, in that case we decided to change its structure step by step to see how our new model works.

5.1. Model 1 (Auto-encoder Refine Network)

From Figure[3] you can see the new model we changed based on the previous model. Since we think the FCN(refined) part won't do a very great job, we decided to change this part to an auto-encoder, it has three blocks for each encoder and decoder part, every blocks contain 2 to 3 layers, the coarse part is fixed.

We also find that in the original model, the fine part has a dimension of 74X55X64, while the data from coarse that concatenate with it has only 74X55X1 dimension, which means when we do the decoder for the concatenated part, the information from the coarse will be highly covered by the information from fine part, to reduce the effect of that, we plan to reduce difference between these two dimensions, since increase the dimension of the coarse part could result in a really massive calculation problem due to the fully connected layers, we decided to reduce the dimension of the encoder part, therefore in the concatenated part of model1, we change the dimension to $38 \times 28 \times 31 + 38 \times 28 \times 1$

We hope by reducing the dimension from the fine part, we could balance the information we get from these two part.The result of the first model doesn't act well, which bring us to our next model, We will have a further discussion on the performance of this model in section 4.

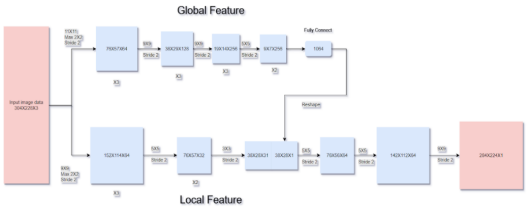


Figure 4: New network Model1

5.2. Model 2 (Residual CNN Coarse Prediction)

For this part we present a new structure of the model, which we modified the coarse part from the original model, since in our original model we found with the current convolution layers, the coarse part lose some information from the original image(you can see the detail from section6), therefore we hope the coarse part could keep more information than before, in order to do that, we add more convolution layers, so it can learn much more features, the model is as shown in Figure 5.

Here we keep the refine part fixed, so we can see whether the coarse part will have a major effect on the result, and we also keep the last two fully connected layer, so after the decoding, all the information from the image could be passed to the refine part.

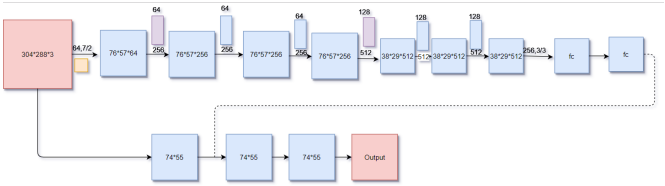


Figure 5: Model 2

5.3. Model 3 (Multi-Scale Residual CNN)

After changing parts of original model without significant improvements, we tried to design our own network for this task. We adapted the multi-scale idea and extended it to k-scale depth generation model.

As described in Figure **??**, our network has a expendable structure where each layer's input is a down-sampled version of the last layer's input. Each layer is composed by a series of Residual Blocks and output a single depth prediction according to its down-sampled input. A concatenation layer combined outputs from different layers and process the final result by another series of Residual blocks.

To help each layer to train, we use an additional variable, represented by Train Gate in the graph, to control which output is used as the starting point of back-propagation at training time. At test time, the Train Gate is fixed to 0, to select the output of concatenation layer as the model output.

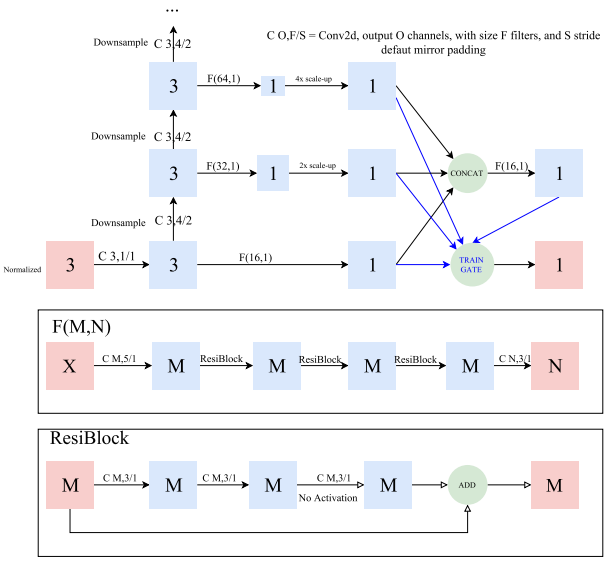


Figure 6: Model 3

Table 1: Model 1

Epoch	50	150
Loss	0.5173	0.2916
Accuracy	0.2337	0.3506

6. Performance

6.1. Model 1

For the first model, we mainly change the fine and refine part to an auto-encoder, although the loss of the test set decrease in a very high speed at first, it converged to around 0.3 after 100 epochs, here we present two results from epoch=50 and epoch=150, you can see the loss and accuracy from table Model 1.

There are three images for each set, the left one is the original image, the center one is our prediction, the right one is the ground-truth. From the result you can see that although the result of loss decrease from 0.5 to 0.3, the performance doesn't changed much, you can see that the predicted depth map from epoch=150 showed much more curves than the one at epoch=50, which could represent as the details on the table, which means it could learn more details after more iteration, but the whole pattern of the depth map has really large difference compare to the ground-truth.

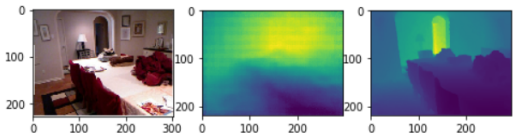


Figure 7: Result for epoch=50

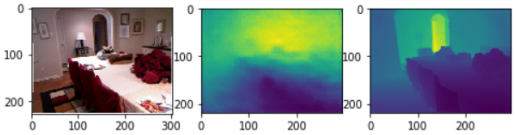


Figure 8: Result for epoch=150

To find out which part cause this problem we check the feature our model learned from each part, by directly output the feature image it learns, which turn out the original coarse has some problems, here we directly decode the feature come from coarse which has dimension 38X28X1, and we get the following result. The left and right image is the original image and ground-truth, the middle one is the result we decode from coarse, you can see that compare to the original image, our coarse part did learn the global fea-

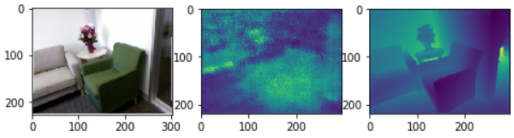


Figure 9: Result for epoch=150

tures we want, you can see some patterns of the sofa, and the vase, but it actually miss judge the distance of the objects, it appears it think compare to the wall, the sofa and vase is much further from us(object is much further when color is much lighter). Therefore, in models2 we will mainly focusing on changing the structure of the coarse part.

The Figure 10, show some result where we get the pattern of the image but lose the details.

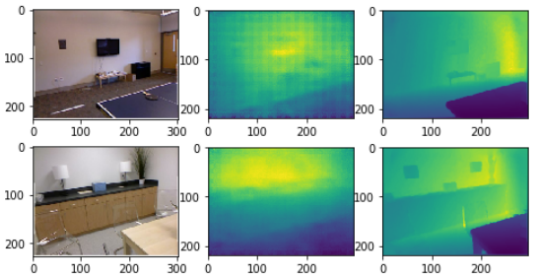


Figure 10: Result for model1 that lost the details

6.2. Model 2

Here is the result(Figure 11) after we changed the structure of the coarse part:

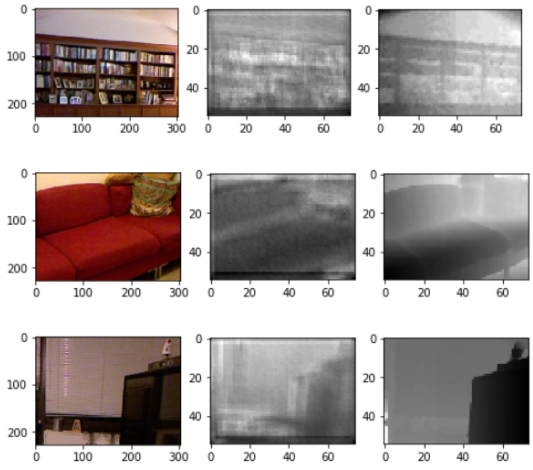


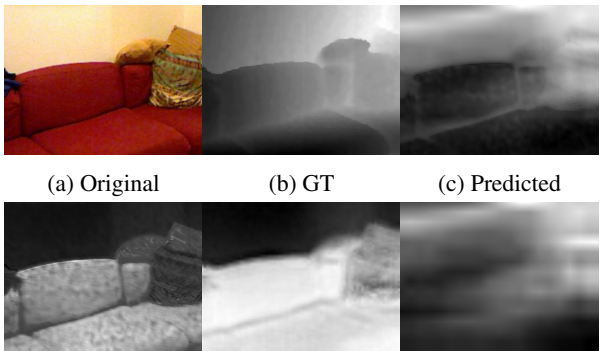
Figure 11: Result for model2

The result is still shown following the order: original image, prediction, and ground truth. The loss of the model 2 converged after 100 epoch, and it's accuracy began to float between 0.43 after that, the result we showed here are from epoch=150, you can see that compare to the result of model1, the model2 learned more details than we thought, since that there is a really clear block for each prediction depth map, we think this may because during the convolution, although we save the details from the original image, we actually lose the information from the global feature, since the length of the coarse is too much to keep the information.

The final loss for the testing set reach 0.1189 after 200 epochs. The accuracy will show in the section 7.

6.3. Model 3

This model converged to 40.63% accuracy within 150 epochs. From the result shown in Figure 12, we can see that



(a) Original (b) GT (c) Predicted

(d) Layer0 (e) Layer1 (f) Layer2

Figure 12: A sample output from Model 3

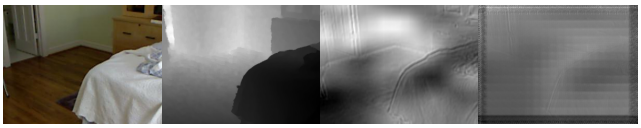
the model successfully learned depth in different scales, and using those information for a finer scale prediction of depth. However, it seems that the model overrated the edges, so that the result image has overly bright edges, which appears artificial for a depth image.

Upscale Method

We tested two different upscale method for this model, namely bilinear interpolation and up-convolution. As shown in Figure 13, the up-convolution method results in severe grid-like artifact. We proposed that this may because the convolution method has no enough information to derive the sub-pixel level information to reconstruct higher resolution image. As in the bi-linear interpolation case, because it uses fixed up-sampling algorithm, it requires less parameter to train and also results in smoother depth interpolation.

7. Conclusion

From this report, we present three different structures for detecting the image depth, based on just one single image,



(a) Original (b) GT (c) Bi-Interp (d) Up-Conv

Figure 13: Comparison between Up-convolution and Bi-linear Interpolation for Upscaling. Both trained for 100 epochs.

Table 2: Best accuracy for each model

	Threshold
Auto-encoder Refine Network(Model 1)	0.3557
Residual CNN Coarse Prediction(Model 2)	0.43
Multi-Scale Residual CNN(Model 3)	0.4063

and we find how does each structure affect the performance of the model, which could lead us to the feature work of the improvement to our current model.

The best accuracy for each model we tried is shown in Table 2.

You can see that the best performed model is model2, which may because it's focusing on learning the details of the image, when we use the mini-difference to calculate the accuracy, it will have the less difference between each pixels, to make a much clear clear visualization of how each model perform, we have the comparison from Figure 11.

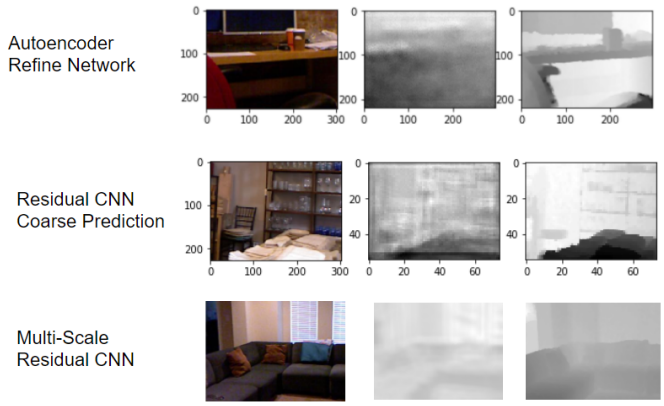


Figure 14: Comparison between three models

From this comparison, we can see that for model 1, when we change the refine part to auto-encoder, it learns much clear pattern of the object, but will lose more details. While for model 2 when we add more convolution layers to the coarse part, it shows a better performance on capture the details of the who image, however, since it weighted the details of the image, you can see the predicted-depth map is not smooth compare to the other two, it always have a very clear edge between objects.

Therefore in model 3, we add a switch node that could balance the affect from coarse and refine part(details in section 5.3), we also add multiple-scales so our model won't just learn from some specific features, the result turns to be fine, compare to the previous two result, it did learned from multiple features, but those informations are highly covered to each other, you can see from the result that although it showed the pattern of the sofa, but doesn't show which part is much closer, since it's trying to find the balance between each feature, and it will reduce the affect of the weighted features.

Finally, our work shows the different learning pattern from three different model, using these pattern we can develop other models that could present better result, however, due to the limitation of the image resources(only 1449 images) and the limitation of the learning time, our model doesn't show a result that could beat the original model. Even though, our model detect the basic image depth without doing pre-training, which may suggest that, if we use the same method as what the base model did, we could have a better result, but since our goal is to generate a better result with less cost, we didn't try that. Also, since our datasets only contain the depth image from indoor scene, it's hard for us to detect the image depth of out-door scene and our model could lose some details when the image is complicated. Some further improvements we can implement will be discussed in the Section 8.

8. Future work

8.1. Better Loss Function

We noticed that the loss function we currently use mainly reflect the pixel-wise accuracy of predicted depth, lacking the statistical and structural information. For example, if a predicted depth gets relatively proximity relationship correct but magnitude wrong, it should get lower loss than get part of pixels correct but other parts totally wrong. We are working on deriving a loss function that can describe the structural difference between predicted depth and target depth.

8.2. Utilizing Segmentation Information

Depth prediction is highly correlated with the segmentation of objects. Pixel from same objects often having linear depth relationship. Because our training data containing label information, we proposed that a separately trained segmentation network could provide more information to refine the depth prediction.

References

[1] J. Carreira and C. Sminchisescu. Cpmc: Automatic object segmentation using constrained parametric min-cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(7):1312–1328, July 2012.

[2] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[3] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.

[4] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Scene parsing with multiscale feature learning, purity trees, and optimal covers. *CoRR*, abs/1202.2160, 2012.

[5] D. F. Fouhey, A. Gupta, and M. Hebert. Data-driven 3D primitives for single image understanding. In *ICCV*, 2013.

[6] D. F. Fouhey, A. Gupta, and M. Hebert. Unfolding an indoor origami world. In *ECCV*, 2014.

[7] D. Fox. Rgb-(d) scene labeling: Features and algorithms. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 2759–2766, Washington, DC, USA, 2012. IEEE Computer Society.

[8] L. Ladický, B. Zeisl, and M. Pollefeys. Discriminatively trained dense surface normal estimation. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 468–484, Cham, 2014. Springer International Publishing.

[9] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[10] X. Wang, D. F. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. *CoRR*, abs/1411.4958, 2014.