Training Monocular Depth Estimation Models on a Budget

Thatcher Freeman
Department of Computer Science
Stanford University
tfr@stanford.edu

Schuyler Tilney-Volk
Department of Electrical Engineering
Stanford University
stilneyv@stanford.edu

Abstract

Monocular Depth Estimation is an underspecified problem, as a given 2D image can correspond to multiple 3D scenes. Deep Learning has been shown to attain good performance in monocular depth estimation, but the state-of-the-art models are costly to train. In this paper, we present a series of modifications and optimizations that can be used to reduce the cost of training models, making monocular depth estimation models more accessible for financially constrained teams.

1. Introduction

Depth estimation from 2D images is critical for a myriad of applications such as autonomous driving, robotics, virtual reality, and surveillance. In more restrictive applications, models that can estimate depth based on a mere single 2D image are becoming increasingly popular. However, monocular depth estimation is a mathematically underspecified problem: for a given image, there are multiple depth maps and real-world geometry that would result in the same projection. However, given a few assumptions, the problem of mapping a single image to a depth map becomes tractable.

Due to the complexity and non-linear structure of the problem, many proposed solutions for monocular depth estimation (MDE) involve the use of various types of neural networks. In addition, the goal of a depth map presents some potential changes in loss functions. Traditional loss functions that only examine pixel-to-pixel changes only sense for depth changes in each pixel individually; they fail to account for changes in surrounding neighbors. This makes them less robust to depth image shifts on the x-y axes and easily manipulated by high-frequency noise. In order to combat such issues, many papers, as discussed below, have proposed the use of loss functions that take advantage of the gradient- and normal-vectors of an image. This paper experiments with both the architecture and loss function optimizations within the tighter resource constraints to find more accessible solutions to the MDE problem.

2. Related Work

Hu et al. [2] discuss an architecture involving a convolutional auto-encoder, followed by a method through which several resized versions of the input image are passed into a convolutional block. [2] also proposes a composite of three loss functions, based on the per-pixel depth error, smoothness similarity, and accuracy of normal vectors in the surfaces of the depth map.

In addition, other works have reframed the problem as an ordinal regression problem. Fu et. al. [1] recognized that using raw pixel MSE led to slow convergence and suboptimal local minima. They argued that most modern systems lost resolution as a result of deep networks of convolutional and pooling layers, and this issue was not completely resolved by the industry standard of including skip connections. Consequently, they opted for a spacing-increasing discretization strategy that recast the depth network learning as an ordinal regression problem, using ordinal regression loss. However, as discovered by Swami et. al. [6], this method led to a formulation that was not entirely differential. [6] then built on the method from [1] to create a full differentiable function, allowing them to include boundary and smoothness constraints in their optimization objective.

Wu et al. [7] approach the depth estimation task using a U-net and a similar loss function to [2]. They trained on a synthetic dataset consisting of rendered images, and also used their model to design an optical component in a camera.

In addition, Yin et. al. [8] discuss using gradient and normal vectors in the loss function that are determined by randomly sampled points in the reconstructed 3D space to improve the depth prediction accuracy and computational efficiency. Their results were of such high fidelity that the team was able to accurately reconstruct a 3D point cloud without training additional sub-models as other research teams had done. As they demonstrated, using gradient and normal vectors in the objective makes the model more robust to noise, including noise from the sensors in the ground truth labels.

In the space of shrinking networks to reduce computational load, Iandola et al. [3] discuss the creation of an image classification model with AlexNet performance, but with $50\times$ fewer parameters. They describe three main techniques in reducing the size of convolutional neural networks without significantly impacting its expressiveness: reducing the kernel size of the convolutional layers, reducing the number of input channels to larger convolutional layers, and downsampling late in the network.

3. Data

One of the standard datasets for the monocular depth estimation task is the NYU-depth-v2 dataset [5]. It is comprised of images of a variety of indoor household locations as recorded by both the RGB and Depth cameras from the Microsoft Kinect, providing about 47k RGB images and their ground truth depth maps.

4. Approach

4.1. Data Augmentation

For each image, we applied four forms of data augmentation. Random horizontal flipping with an 0.5 probability, Color Jitter where brightness, contrast, and saturation are randomly scaled by a value in the range of 0.6 to 1.4, random cropping shrinking the image by 0 to 20% in size, and rotation by between -5 and 5 degrees.

4.2. Architecture

We largely considered two kinds of architectures. The first was a U-Net architecture consisting of a ResNet encoder, and a decoder consisting of four Up-Projection blocks, similar to the ones used by Hu et al. [2] and introduced by Laina et al. [4], followed by two convolutional layers. In the later charts in this report, we will refer to this model as the "U-Net" model. The other architecture was based on the one used by Hu et al. [2], consisting of a ResNet encoder, a decoder consisting of four Up-Projection layers, an MFF module consisting of four Up-Projection layers and a convolutional layer, and a Refinement module that processes the concatenated outputs of the decoder and the MFF module using four convolutional layers. We will refer to this model later as the "Hu et al." model.

In their original forms, all convolutions outside of the encoders are 5x5 'same' convolutions. A diagram illustrating the architecture is visible in Figure 1 and our source code is available on github. [1]

The bulk of this report consists of the modifications we made to these models to improve inference speed during training time. When using the Hu et al. model, as described in the original paper, our profiling tools informed us that the vast majority of the inference time was spent through backpropagation through the convolutional layers, particularly in the Up-Projection layers. We therefore considered the following modifications:

4.2.1 Intermediate Resolution

While the NYU-Depth-V2 dataset consists of 640×480 images, it is not necessary to use the image's full resolution as input to the model. By resizing the input images to a lower resolution and outputting a depth map at the same, reduced resolution, the convolution operation will have to operate on a smaller number of windows and therefore reduce the computational load. We used Area interpolation to reduce the resolution of the input images to the target intermediate resolution, and we used Bicubic interpolation to upscale the model's predicted depth maps back to 640×480. Our loss functions and evaluation metrics are all computed at the final 640×480 resolution to maintain consistency. We experimented with five different intermediate resolutions and report our results in section 6.1.

4.2.2 Encoder Size

Hu et al. used a ResNet50 encoder, resulting in the four Up-Projection layers in the MFF module taking inputs with 256, 512, 1024, and 2048 channels respectively. As those Up-Projection layers rescale their inputs all the way up to the model's intermediate resolution, the fourth Up-Projection layer has a substantial computational load, taking 2048 channels as input. Similarly, in our U-Net architecture, when the encoder activations have many channels, our decoder up-projection layers would have to operate on a large number of channels as well.

We used a ResNet34 encoder in all our models simply because using a ResNet50 would be too expensive to train. The intermediate activations coming from the four blocks of a ResNet34 model have four times fewer channels as a ResNet50 encoder. We measured the inference speed on the Hu et al. model with both a ResNet34 and a ResNet50 encoder, and those results are available in section 6.3.

4.2.3 Convolutional Kernel Size

The model used by Hu et al. [2] consisted of a ResNet50 encoder, pretrained for the ImageNet image classification challenge, followed by three fully convolutional modules. Each of those modules consisted of standard convolutional layers, and Up-Projection layers very similar to the ones described by [4]. In [2], all convolutional layers in their model, outside of the encoder, were 5×5 'same' convolutions. Our initial U-Net and Hu et al. models used 5×5 convolutional layers as well.

[1] https://github.com/thatcherfreeman/CS231A-FinalProject

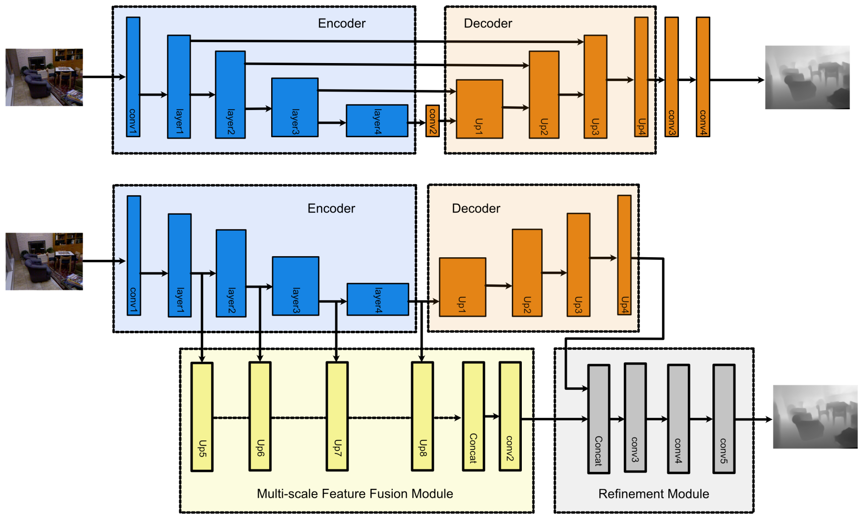

Figure 1. Comparison of the two model architectures we used. Top is the U-Net model, an the bottom is the Hu et al. model.

We looked at the method described by [3], which described an approach for designing smaller network architectures. Applying their techniques to our U-Net model and the Hu et al. model, we experimented with shrinking the convolutional kernel size to 3×3 in all convolutional layers outside of the encoder. It should be noted that while [4] justified the use of 5×5 convolution layers in the Up-Projection layers because they followed a 2×2 unpool layer, [2] simply uses bilinear interpolation instead of an unpooling layer, therefore avoiding the concern that the kernel would potentially operate on the zeros outputted by a unpooling layer. Thus, using 3×3 convolutional layers in the Up-Projection layers is acceptable. The performance and inference speeds of our models with and without this change are discussed in 6.3

4.2.4 Modifying the Up-Projection Layers

The Up-Projection layers used by [2] consisted of a bilinear interpolation to upscale the $C \times H \times W$ input along the height and width, followed by a series of convolutional layers that reduced the number of channels to $C/2$. We observed that this is computationally expensive during training, as the activations leaving the interpolation layer can be quite large. Again, in following the method used by SqueezeNet [3], we considered the effect of introducing a 1×1 convolutional layer before the interpolation step, whose sole purpose was to reduce the number of channels from C to $C/2$. This allowed the later convolutional layers to operate with smaller filters, on a smaller input, therefore improving performance. This change made a larger difference in inference speed with the Hu et al. model than with our U-Net model, and the results of this experiment will be discussed in section 6.3.

4.3. Loss Functions

The simplest loss function commonly used in monocular depth estimation is to simply treat the problem as a per-pixel regression task. If y is the ground truth depth map and $\hat{y}$ is the model's predicted depth map, with T pixels, then we can simply compute the L1 loss in the following way:

$$\mathcal{L}_{L1}(\hat{y}, y) = \frac{1}{T} \sum_{i,j} |\hat{y}_{ij} - y_{ij}| \tag{1}$$

However, simply using the above loss function typically results in jagged, distorted depth maps, and incorrectly penalizes inaccuracies for distant objects as much as nearby ones, where nearby objects should be given more importance in accuracy. A simple solution to this is to instead take the log. As we expect errors to be larger on distant objects, this penalizes a 1cm increase in error at a distance less than a 1cm increase in error near the camera.

$$\mathcal{L}_{\text{log error}}(\hat{y}, y) = \frac{1}{T} \sum_{i,j} \log(|\hat{y}_{ij} - y_{ij}| + \alpha) \tag{2}$$

We have also implemented the two other loss functions from [2], which provide the model with more information about the surfaces in the ground truth depth map. The first loss function compares the surface gradients in the horizontal and vertical directions between the ground truth and estimated depth maps, as so:

$$\mathcal{L}_{\text{grad}}(\hat{y}, y) = \frac{1}{T} \sum_{i,j} \log(\nabla_x |\hat{y}_{ij} - y_{ij}| + \alpha) + \log(\nabla_y |\hat{y}_{ij} - y_{ij}| + \alpha)$$

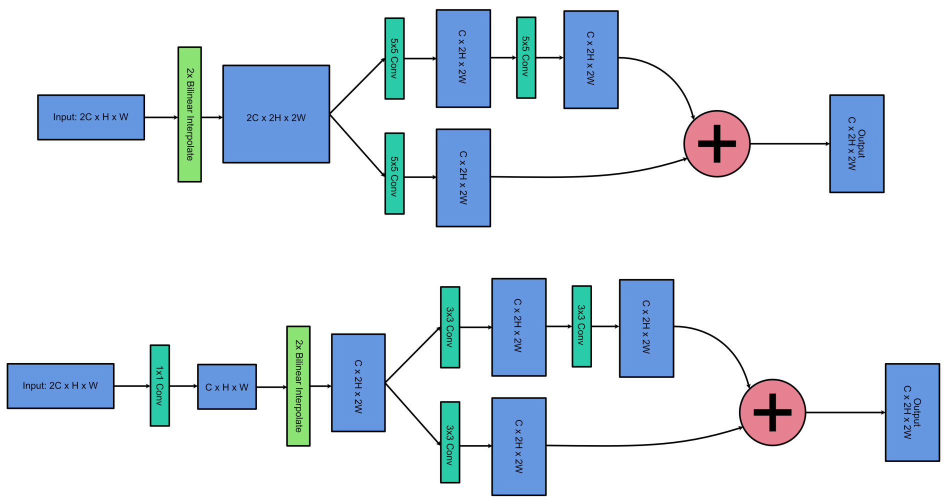

Figure 2. Comparison of the Up-Projection layer used by Hu et al. [2], based on [4] (Top), and our modified Up-Projection layer with a 1×1 channel reduction layer and 3×3 convolutions (Bottom).

This loss function is expected to encourage smoothness in the model's predicted depth map. The final loss function compares the cosine similarity of the per-pixel normal vectors in the ground truth and estimated depth maps.

$$\mathcal{L}_{\text{normal}}(\hat{y}, y) = \frac{1}{T}\sum_{i,j} |1 - \cos\text{sim}(n^{\hat{y}_{ij}}, n^{y_{ij}})|$$

Where $\cos\text{sim}(A, B)$ is the cosine similarity function $A \cdot B/(||A||\ ||B||)$ and $n^{y_{ij}}$ is the normal vector given by $[-\nabla_x y_{ij}, -\nabla_y y_{ij}, 1]^T$. This loss function encourages the model's estimated depth map to match the surface normals of the objects in the scene.

We sum these three loss functions together to get our overall training loss:

$$\mathcal{L} = \mathcal{L}_{\text{depth}} + \mathcal{L}_{\text{grad}} + \mathcal{L}_{\text{normal}}$$

We experimented with $\mathcal{L}_{\text{depth}} = \mathcal{L}_{\text{log error}}$ and $\mathcal{L}_{\text{depth}} = \mathcal{L}_{L1}$. Additionally, we considered the effect of just using $\mathcal{L}_{L1}$ as the only loss function. The results of these experiments are available in section 6.2.

4.3.1 Improving Gradient and Normal Vector Loss Functions

Yin et al. [8] made an astute observation that the NYU-Depth-V2 dataset is captured using a cheap consumer device, the Kinect, and therefore while the ground truth depth map distances are overall accurate, the surface gradients can be noisy. Yin et al. address this issue by constructing a point cloud from the depth maps and sampling normal vectors over larger surfaces, rather than simply in the local neighborhood of individual pixels.

The loss function used by Hu et al. and described above in section 4.3 calculates gradients based on adjacent pixels in the predicted and ground-truth depth maps. If the ground truth normal vectors and gradients are excessively noisy, then these noisy labels could impede training performance. As following the method used in [8] would be computationally expensive and difficult, we propose a naive data augmentation method to expand the neighborhood used to calculate the ground truth surface normal vectors. Our method is to simply apply a subtle gaussian blur to the 640×480 ground truth depth maps, but only for the $\mathcal{L}_{\text{grad}}$ and $\mathcal{L}_{\text{normal}}$ portions of the loss function. We applied a gaussian blur with a standard deviation of 1.4 and a kernel size of 7. When this data augmentation is used, we will refer to it as a "blurred normal" loss function. The results of this experiment are discussed in section 6.2.

5. Experiments

5.1. Evaluation Methods

There are several evaluation metrics commonly used to evaluate the quality of depth maps. The most common one is Root-Mean-Squared (RMS) error. Given a model's output depth map $\hat{y}$ and a ground truth depth map y, over T pixels, we simply compute:

$$\text{RMS} = \sqrt{\frac{1}{T}\sum_{i,j}(\hat{y}_{ij} - y_{ij})^2}$$

And the RMS error is averaged over all images in the test dataset. Another metric is the Mean Relative Error (REL).

$$\text{REL} = \frac{1}{T}\sum_{i,j} |\hat{y}_{ij} - y_{ij}|$$

Mean Log10 error is similar, deprioritizing accuracy for points far from the camera.

$$\text{Log10} = \frac{1}{T}\sum_{i,j} ||\log_{10}(\hat{y}_{ij}) - \log_{10}(y_{ij})||_1$$

And the final commonly used metric is the Threshold, measuring the fraction of pixels in the model's prediction that deviate from the ground truth label by less than a certain amount.

$$\text{Threshold } \delta = \frac{1}{T}\sum_{i,j} \mathbb{1}\left\{\max\left(\frac{\hat{y}_{ij}}{y_{ij}}, \frac{y_{ij}}{\hat{y}_{ij}}\right) < \delta\right\}$$

We evaluate the above with $\delta_1 = 1.25$, $\delta_2 = 1.25^2$, and $\delta_3 = 1.25^3$. All of the above error metrics iterate over all T pixels in the ground truth depth map that are nonzero.

We will evaluate our results by measuring all of the above metrics on the test split of the NYU-depth V2 dataset, in accordance to the standard benchmark on that task. We would expect the depth maps to visually appear to be smooth, with clean edges around objects.

5.2. Training

As we had two overall model architectures, two loss functions, the choice of blurring the ground truth depth maps, five intermediate resolutions, two different encoders, two convolutional kernel sizes, and the choice of whether to include the 1×1 filter in the Up-Projection layers, resulting in an enormous number of possible experiments, we decided to approach the experiments in a linear fashion. We made a reasonable guess about the rest of the model and compared all options for a given experiment with all other factors kept constant, and then we used the best outcome to inform our later experiments.

We trained our models on a Tesla T4 with the Adam optimizer and a learning rate of 1e-4 and a batch size of 1, for 3 epochs. We partitioned the training split of the NYU-Depth-V2 dataset so that 3% were reserved for validation and the other 46156 examples were used for training. When we had run all the small experiments and selected the best choice for each one, we trained the resulting models for 20 epochs.

6. Results

6.1. Intermediate Resolution

To compare intermediate resolutions, we used our U-net model with a ResNet-34 encoder, 3×3 convolutions, and the 1×1 channel reduction in the Up-Projection layers. We trained for three epochs against the Hu et al loss function ($\mathcal{L}_{\text{depth}} = \mathcal{L}_{\text{log error}}$) without blurring the ground truth depth maps at the resolutions of 640×480, 512×384, 384×288, 256×192, and 128×96, and measured the model's performance against the test dataset using the 6 evaluation measures. We also inspected the qualitative appearance of the model's predicted depth maps.

Numerically, we observed that the 384×288 model and the 256×192 model had very similar performance, but upon inspection, the depth maps from the larger resolution were substantially more detailed and had fewer artifacts. A comparison of some depth maps are in Figure 3. We therefore conducted all later tests using a resolution of 384×288, except when indicated otherwise.

Resolution	RMS	REL	LOG10	δ_1	δ_2	δ_3
640×480	0.999	0.201	0.110	0.580	0.831	0.924
512×384	0.982	0.192	0.104	0.617	0.851	0.929
384×288	**0.933**	0.173	0.096	0.666	**0.878**	**0.936**
256×192	0.944	**0.170**	**0.095**	**0.670**	0.872	0.935
128×96	0.996	0.188	0.105	0.626	0.847	0.919

Table 1. Validation time stats vs Intermediate Resolution, using Hu et al. loss function and U-Net 34 model

Resolution	Speedup
640x480	1.0x
512x314	1.4x
384x288	2.6x
256x192	5.2x
128x96	12.0x

Table 2. Relative inference speed with U-Net model, using a ResNet 34 encoder and 3×3 convolutions, running on CPU. These speedup figures were similar to the speedup when running on a GPU.

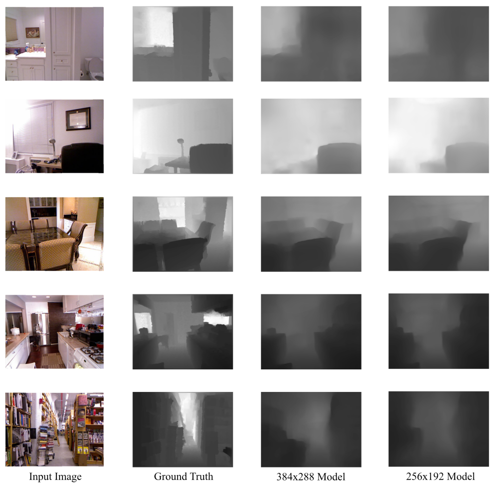

Figure 3. Comparison of depth maps after 3 epochs of training, with the U-Net model with a kernel size of 3 and the channel reduction filter in the up-projection layers, trained at 384×288 and 256×192. These five images are randomly sampled from the test split of NYU-Depth V2.

6.2. Loss Function

To compare loss functions, we trained the same model from section 6.1, but at an intermediate resolution of 384×288. We again trained for three epochs with each configuration of our loss function. The most naive configuration is the Per-pixel L1 norm loss function. Then, we train with the Hu et al. loss function, and with that loss function used where the ground truth depth map is slightly blurred when computing the gradients. We also trained two more times with the $\mathcal{L}_{\text{depth}}$ portion of the loss function replaced with the $\mathcal{L}_{L1}$ function, with and without blurring the ground-truth depth map for the gradient and normal loss functions. We evaluated the six metrics and then inspected the same five images as before.

Qualitatively, we felt that the Hu et al. loss function and the blurred version of the same function (with $\mathcal{L}_{\text{depth}} = \mathcal{L}_{\text{log error}}$) looked the best and had the fewest artifacts.

6.3. Architectural Changes

We began by benchmarking inference speed on several models. As we only had access to $350 of GCP credits and one other GPU, for expediency we had to prioritize models that created both the highest quality depth maps, and also fit within our limited budget, time, and hardware constraints. The inference speeds of the different models we considered is visible in Table 4. The "Hu et al" model refers to the model used in the original paper, with 5×5 convolutions and a ResNet 50 encoder, and the various benchmarked alterations are listed as well. The "U-Net" model represents a ResNet 34 encoder, with 5×5 convolutions, the models listed had no 1×1 channel reduction filter unless otherwise specified. All models were trained at 384×288 unless otherwise specified.

The five fastest models at the 384×288 resolution were the U-Net models as well as the Hu et al. model with all the architectural changes suggested in section 4.2. We trained each of these models to 3 epochs and quantitatively and qualitatively analyzed their results. Their evaluation metrics are available in Table 5. Based on the quantitative results, the U-Net model performed best without the 1×1 channel reduction filter, which made only a minor impact on inference speed. The modified Hu et al. model was one of the slower models of these five, but had decent performance.

6.4. Finished Model

Of the five models in the previous section, we ultimately decided to continue with the U-Net model with a convolutional kernel size of 3, as well as the Hu et al. model with the ResNet34 encoder, 1×1 channel reduction filter, and the kernel size of 3. We trained these two models for 20 epochs using the Blurred variant of the Hu et al. loss function, and we also trained the U-Net model again with the unblurred version of the same loss function. The test results for these three runs are reported in table 6 and final images are displayed in Figure 5.

Qualitatively, the two models trained on the blurred gradient depth maps had the best results, with the miniature version of the Hu et al. model doing a decent job of estimating the depth in distant parts of the image.

7. Future Work

Future research directions would be in applying these techniques to other State-of-the-art models, and in increasing the depth of the networks while even further reducing the size of the convolutions as in [3]. Additionally, while this work generated depth maps at the same resolution as the input images, one could choose to output full-sized depth maps (with a smaller input image), or vice-versa to experiment with performance characteristics.

8. Conclusion

In this work we have demonstrated techniques to get a usable monocular depth estimation model under limited time and budget constraints. We apply the techniques of reducing the image resolution, shrinking kernel size, and removing activation channels in a variety of ways. We also propose a novel data augmentation method that provides slight improvements to surface normals in monocular depth estimation models, with a negligible increase in computation time.

References

[1] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao. Deep ordinal regression network for monocular depth estimation, 2018.

[2] J. Hu, M. Ozay, Y. Zhang, and T. Okatani. Revisiting single image depth estimation: Toward higher resolution maps with accurate object boundaries. 2019.

[3] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size, 2016.

[4] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision (3DV)*, pages 239–248. IEEE, 2016.

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

[6] K. Swami, P. V. Bondada, and P. K. Bajpai. Aced: Accurate and edge-consistent monocular depth estimation, 2020.

[7] Y. Wu, V. Boominathan, H. Chen, A. Sankaranarayanan, and A. Veeraraghavan. Phasecam3d—learning phase masks for passive single view depth estimation. In *2019 IEEE International Conference on Computational Photography (ICCP)*, pages 1–12. IEEE, 2019.

	RMS	REL	LOG10	δ_1	δ_2	δ_3
L1 loss	0.774	0.174	0.128	0.674	0.887	0.940
Hu et al.	0.933	0.173	**0.096**	0.666	0.878	0.936
Blurred Hu et al.	0.844	0.178	0.118	0.637	0.870	0.937
L1 Hu et al.	**0.706**	**0.152**	0.109	**0.718**	**0.904**	**0.946**
L1 Blurred Hu et al.	0.795	0.172	0.102	0.661	0.886	0.945

Table 3. Validation scores for each loss function, using Baseline34 model at 288x384 resolution.

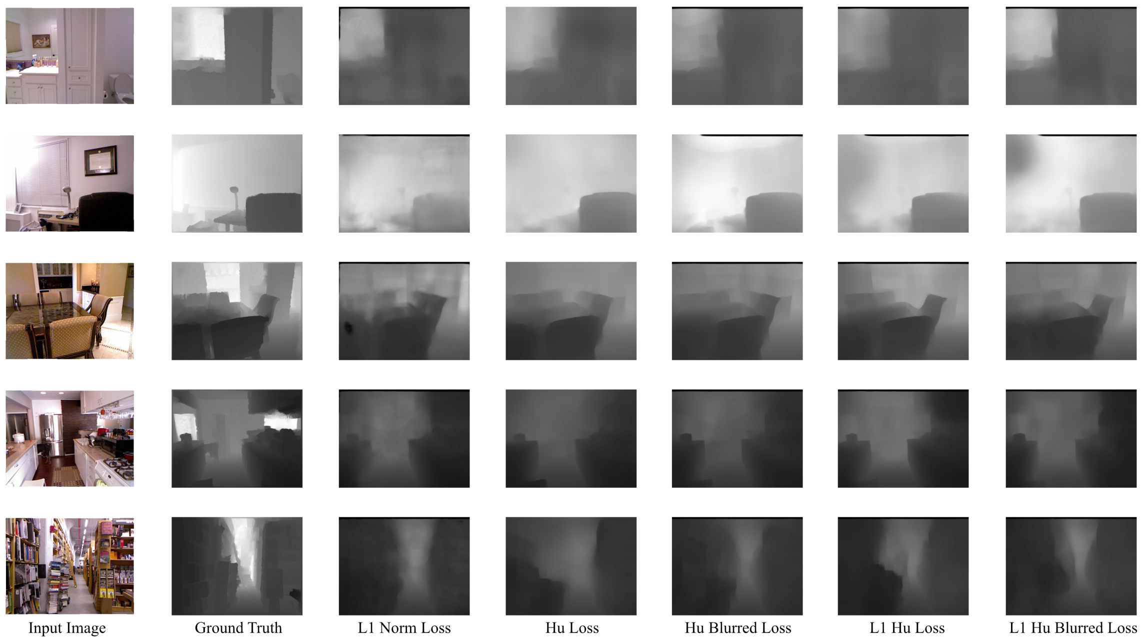

Figure 4. Comparison of the U-Net model trained with each loss function for three epochs.

[8] W. Yin, Y. Liu, C. Shen, and Y. Yan. Enforcing geometric constraints of virtual normal for depth prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5684–5693, 2019.

Model	Inference Speed (speedup)
U-Net w/ 1x1 filter, ksize 3	11.2 it/s (59x)
U-Net w/ ksize 3	10.2 it/s (54x)
U-Net w/ 1x1 filter	3.09 it/s (16x)
U-Net	2.9 it/s (15x)
Hu et al. w/ Resnet34 Encoder, 1x1 filter, ksize 3	4.8 it/s (25x)
Hu et al. w/ Resnet34 Encoder, 1x1 filter	1.6 it/s (8.4x)
Hu et al. w/ Resnet34 Encoder, ksize 3	1.77 it/s (9.3x)
Hu et al. w/ Resnet34 Encoder	1.35 it/s (7.1x)
Hu et al. w/ 1x1 filter, ksize 3 (256x192)	3.2 it/s (17x)
Hu et al. w/ 1x1 filter, ksize 3	1.5 it/s (7.9x)
Hu et al. (256x192)	0.35 it/s (1.8x)
Hu et al.	0.19 it/s (1.0x)

Table 4. Relative inference speed using Hu et Al. blurred loss function

	RMS	REL	LOG10	δ_1	δ_2	δ_3
Hu et al. w/ ResNet 34 Encoder, 1x1 filter, ksize 3	**0.804**	**0.170**	0.115	**0.671**	0.881	0.939
U-Net w/ 1x1 filter, ksize 3	0.844	0.178	0.118	0.637	0.870	0.937
U-Net w/ ksize 3	0.830	0.176	0.119	0.652	0.877	0.937
U-Net w/ 1x1 filter	0.886	0.192	0.123	0.620	0.852	0.928
U-Net	0.813	0.177	**0.114**	0.666	**0.884**	**0.941**

Table 5. Validation scores for each model, using Blurred Hu et al. loss function

	Train Time	RMS	REL	LOG10	δ_1	δ_2	δ_3
U-Net w/ ksize 3, Blurred Loss	25 Hours	**0.846**	**0.209**	0.123	0.659	**0.890**	0.957
U-Net w/ ksize 3, Unblurred Hu et al. Loss	25 Hours	0.850	0.209	0.124	**0.665**	0.890	0.956
Hu et al. w/ ResNet 34 Encoder, 1x1 filter, ksize 3, Blurred Loss	54 Hours	0.876	0.210	**0.106**	0.660	0.885	**0.959**

Table 6. Test scores for each model after 20 epochs of training

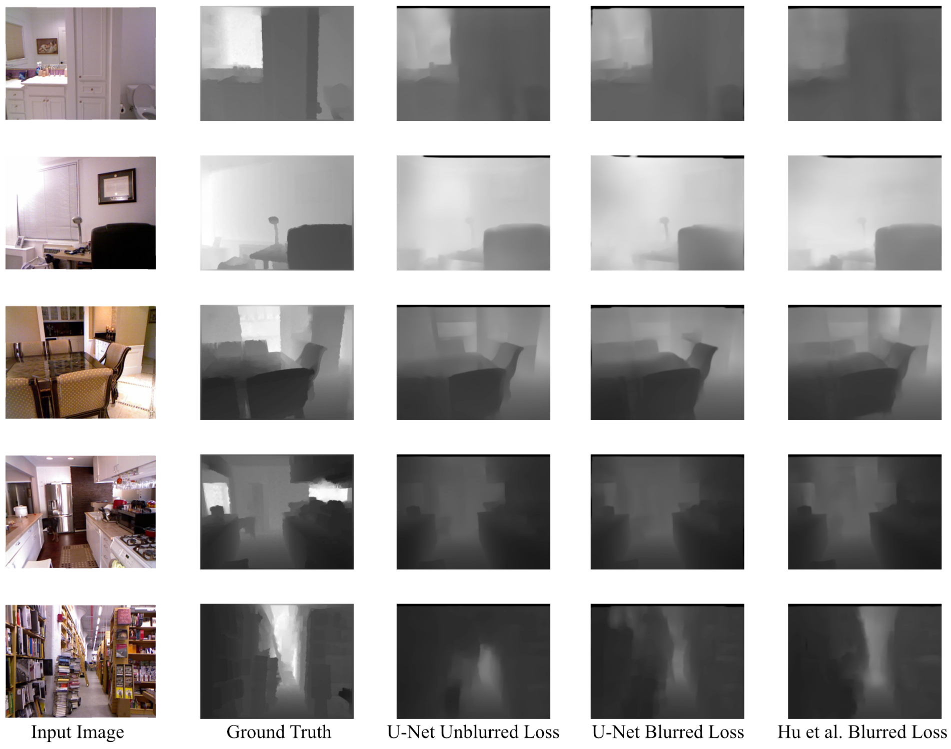

Figure 5. Predicted images by the three models after 20 epochs of training.