Monocular Depth Estimation using a deep network

Renwen Cui, Zhenye Li

August 12, 2021

Abstract

Depth reconstruction is a fundamental problem in many applications. Monocular depth estimation with high resolution from a single RGB image is ill-posed. In this project, we learn and build an encoder-decoder network with the help of pretrained networks architecture. We use the high-performing pretrained DenseNet-169 as our encoder and develop the idea of feature reuse from the DenseNet to our decoder architecture. Besides, we define the weighted sum of three loss functions that lead to more accurate depth estimation with high-frequency details. In addition, we try to improve this decoder architecture and adjust the loss function. Our optimized models are able to achieve qualitatively better depth maps with high resolution, and even reduce the computing time.

1 Introduction

A dense depth map provides detail 3D geometric relations of objects and their environment within a scene that can be applied to recognition, navigation, image refocusing, and segmentation[1]. Depth estimation from 2D images is an important and basic task in many applications including scene understanding and reconstruction[2]. For this project, we want to construct a deep neural network to predict the depth map from a single RGB image in an end-to-end way. The input of the network is RGB images, and after applying the network the output of this system is the corresponding estimated depth maps. The two datasets we use for training the model, NYU Depth v2 and KITTI that include indoor and outdoor scenes respectively, contain RGB images and corresponding ground truth of depth maps. Because of the features of our datasets, this supervised monocular depth estimation can be regarded as a regression problem and we can apply machine learning algorithms to these two datasets. We first learn and analyze the existing network architecture proposed by I. Alhashim et al.[2], an encoder-decoder architecture with skip connections via transfer learning, as shown in Figure 1. Next, we will try to improve the decoder architecture and adjust different loss functions to build new models.

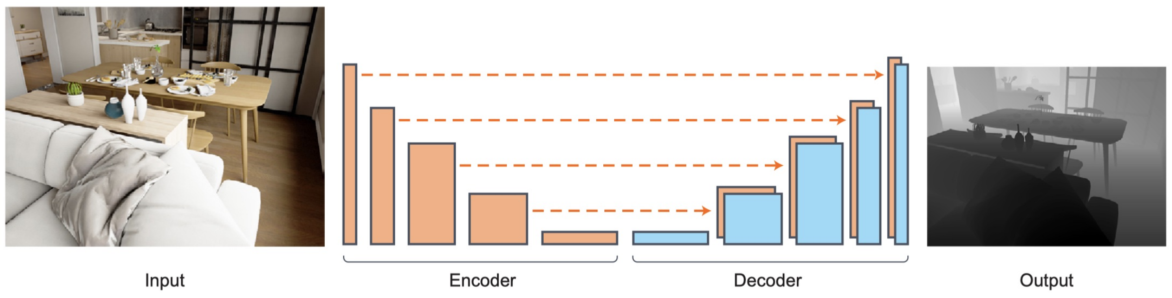

Figure 1: Network architecture[2]

1.1 Related work

There are many prior methods for computing the depth map of a scene. Stereo vision matching that simulates human eyes is a popular way to construct 3D depth information by observing the scene from two viewpoints. Using this geometry-based method needs to calibrate the two cameras in advance to obtain the transformation between them[3]. While the above method can efficiently perceive the depth information, binocular vision depends on image pairs. The other method of computing the depth map is based on depth sensors, such as light detection and ranging (LIDAR) that is able to directly capture depth images at high speed, high resolution, and long-range[4]. However, the depth map obtained by LIDAR is sparse rather than dense. Besides, considering the cost and size of this equipment, estimating the dense depth map from a single image attracts more attention.

Much work on monocular depth estimation is focusing on applying deep learning to perform 3D reconstruction from a single RGB image[1]. A variety of deep neural networks, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), show their outstanding performance on recovering pixel-level depth maps. Many studies aim to improve the resolution and quality of the estimated depth maps with more accurate boundaries using simple model architectures and few parameters. Eigen et al. first solved this problem using the multi-scale framework with two-component slacks, the global coarse-scale network and the local finescale network. The first estimates the global structure, and then a second refines it[5]. Mayer et al. proposed a fully CNN framework to solve this problem[6]. Laina et al. extended ResNet and introduced residual learning to construct a deeper network[7]. I. Alhashim et al. introduced an encoder-decoder network based on the concept of transfer learning[2], as shown in Figure 1.

2 Methods and Theory

In this section, we first describe the standard encoder-decoder network we used, and then propose modified decoder architectures. Next, we discuss a loss function for this task. Finally, we describe the inpainting method for the pre-processing of the KITTI dataset.

2.1 Network Architecture

Figure 1 shows an overview of encoder-decoder network. The encoder architecture compresses an input RGB image into a code with lots of feature maps and small size, and then in the decoder architecture, this code needs to be reconstructed depth map with one channel, a grayscale image.

Encoder. In this project, we use the DenseNet-169 pretrained on ImageNet[8] encoding an input RGB image to a feature vector. The Dense Convolutional Network (DenseNet) that connects each layer to every other layer has several compelling advantages: they can alleviate the vanishing-gradient problem when doing back-propagation, also encourage feature reuse[8]. The core of the DenseNet is feature concatenation or feature reuse, so Huang G et al. divided the network into multiple densely connected dense blocks to achieve both feature concatenation and down-sampling operation[8], as shown in Figure 2. The dense block performs feature concatenation. Inside each dense block, the size of feature maps remains the same, but the number of output feature maps is the sum of the number of input and all outputs of all convolution layers. The layers between the dense blocks as transition layers include the convolution and pooling that can perform the downsampling operation.

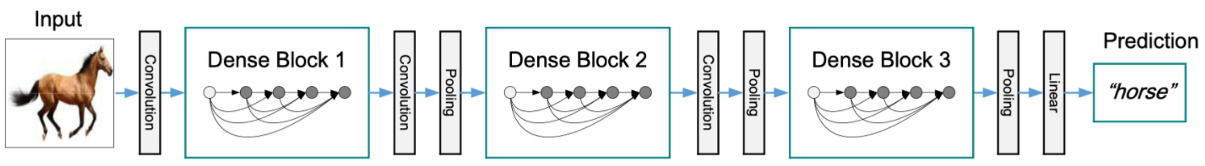

Figure 2: Encoder architecture[2]

In this monocular project, we remove the classification layer in the original DenseNet-169 architecture that is related to the classification task. The exact DenseNet-169 architecture we used configurations on ImageNet is shown in Table 1. The first part of the DenseNet-169 architecture consists of a 7x7 Convolution Layer with a stride 2 and a 3x3 MaxPooling layer, where each Convolution Layer corresponds to the sequence BatchNormalization-ReLu activation function-Convolution, followed by four dense blocks and transition layers. Inside each of the dense blocks, the 1x1 convolution layer is performed before the 3x3 convolution layer to reduce the number of input feature maps to the computing efficiency. For the NYU depth v2 dataset, the size of the input is 480x640x3, the output of the encoder architecture is 15x20x1664. The output code will be fed into the decoder architecture directly.

DenseNet-169		
Layer	Output	Function
Input	480x640x3	
Convolution1	240x320x64	7×7 conv, stride 2
Pooling1	120x160x64	3×3 max pool, stride 2
Dense BLock1	120x160x256	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition1	60x80x128	1×1 conv 2×2 average pool, stride 2
Dense BLock2	60x80x512	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition2	30x40x256	1×1 conv 2×2 average pool, stride 2
Dense BLock3	30x40x1280	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$
Transition3	15x20x640	1×1 conv 2×2 average pool, stride 2
Dense BLock4	15x20x1664	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$

Table 1: Encoder DenseNet169 Architecture along with their exact shapes for NYU depth v2 Dataset. Each conv layer corresponds the sequence: BatchNormalization-ReLU-Convolution.

Decoder. The decoder architecture has the same concept of feature maps concatenation from the encoder DenseNet-169[2]. It saves all outputs of every transition layer in the encoder architecture. As shown in Table 2, the decoder architecture consists of a series of upsampling blocks after the first 1x1 convolution layer. Each upsampling block is composed of one bilinear upsampling, one concatenation operation, and two convolution layers followed by a leaky ReLU activation function. The first convolutional layer in each upsampling block, ConvA, is applied on the concatenation of the output of the previous layer and the output of the transition layer in the encoder architecture with the same size. For the NYU depth v2 dataset, the size of the output grey depth map of this decoder network is 240x320x1, a half of RGB input resolution. We compare the output prediction with the ground truth downsampling by 2x during test time.

Simple Decoder Architecture		
Layer	Output	Function
Input	15x20x1664	
Convolution	15x20x1664	1×1 conv
Upsampling1	30x40x1664	2×2 upsampling
Concat1	30x40x1962	Concatenate Encoder-Transition3
Up1-ConvA	30x40x832	3×3 conv
Up1-ConvB	30x40x832	3×3 conv
Upsampling2	60x80x832	2×2 upsampling
Concat2	60x80x960	Concatenate Encoder-Transition2
Up2-ConvA	60x80x416	3×3 conv
Up2-ConvB	60x80x416	3×3 conv
Upsampling3	120x160x416	2×2 upsampling
Concat3	120x160x480	Concatenate Encoder-Transition1
Up3-ConvA	120x160x208	3×3 conv
Up3-ConvB	120x160x208	3×3 conv
Upsampling4	240x320x208	2×2 upsampling
Concat4	240x320x272	Concatenate Encoder-Convolution1
Up4-ConvA	240x320x104	3×3 conv
Up4-ConvB	240x320x104	3×3 conv
Convolution	240x320x1	3×3 conv

Table 2: Decoder Architecture along with their exact shapes for NYU depth v2 Dataset. Upsampling layer is bilinear upsampling. Each ConvB layer is followed by a leaky ReLU with parameter $\alpha = 0.2$.

2.2 Modified Decoder

Adding one more Upsampling layer. For the original encoder-decoder architecture, the output grey depth map is at half the resolution of the input RGB image, so we try to add one more 2x bilinear upsampling layer at the end of the original decoder architecture to make the resolution of output prediction be the same as the input.

Adding BatchNormalization layer. BatchNormalization layer has lots of benefits, for example, it is less affected by initialization and the effect of internal covariate shift, then it can speed training process. Although BatchNormalization layer is recommended in many CNN methods, the original decoder architecture does not contain BatchNormalization layer in each upsampling block. Therefore, we add BatchNormalization layer after each ReLU activation function.

Simple Decoder Architecture. We also examine the effect of removing the feature maps concatenation between layers of the encoder and decoder architectures. We define a simple decoder without skip-connections that only consists of upsampling, convolution layer and ReLU activation function, as shown in Table 3.

Decoder Architecture		
Layer	Output	Function
Input	15x20x1664	
Convolution	15x20x1664	1×1 convTranspose2d
Upsampling1	30x40x1664	2×2 upsampling
Up1-ConvA	30x40x832	3×3 conv
Up1-ConvB	30x40x832	3×3 conv
Upsampling2	60x80x832	2×2 upsampling
Up2-ConvA	60x80x416	3×3 conv
Up2-ConvB	60x80x416	3×3 conv
Upsampling3	120x160x416	2×2 upsampling
Up3-ConvA	120x160x208	3×3 conv
Up3-ConvB	120x160x208	3×3 conv
Upsampling4	240x320x208	2×2 upsampling
Up4-ConvA	240x320x104	3×3 conv
Up4-ConvB	240x320x104	3×3 conv
Convolution	240x320x1	3×3 convTranspose2d

Table 3: Simple Decoder Architecture along with their exact shapes for NYU depth v2 Dataset. Upsampling layer is bilinear upsampling. Each ConvB layer follows a leaky ReLU with parameter $\alpha = 0.2$.

2.3 Loss Function

Different loss functions have significant influence on depth estimation performance and computing speed. The optimized loss function we use for this depth regression problem keeps a balance between the accuracy of depth map prediction and also the reconstruction of high frequency details in the depth map[2]. The loss function for training the network is the weighted sum of the three loss functions shown below.

$$L(y, \hat{y}) = w_1 L_{depth}(y, \hat{y}) + w_2 L_{grad}(y, \hat{y}) + w_3 L_{SSIM}(y, \hat{y})$$

The first term $L_{depth}(y, \hat{y})$ is the point-wise L1 loss. The second loss term $L_{grad}(y, \hat{y})$ is the gradient loss. The last term $L_{SSIM}(y, \hat{y})$ is the SSIM loss. Details about these three loss functions are described as follows.

L1 Loss. L1 Loss is used to minimize the sum of the mean absolute error between predicted depth value and ground truth.

$$L_{depth}(y, \hat{y}) = \frac{1}{n} \sum_{n}^{p} |y_p - \hat{y_p}|$$

Gradient Loss. Gradient loss penalizes distortions of high frequency details in the image domain of the depth map[2]. These details typically correspond to the boundaries of objects in the scene. Gx and Gy compute the differences in the x and y components for the depth image gradients of the predicted depth map and ground truth.

$$L_{grad}(y, \hat{y}) = \frac{1}{n} \sum_{n}^{p} |\boldsymbol{g_x}(y_p, \hat{y_p})| + |\boldsymbol{g_y}(y_p, \hat{y_p})|$$

Structural Similarity(SSIM) Loss. The SSIM index is used for measuring the similarity between two images, which is a good loss term for image comparison[9]. The SSIM incorporates comparisons between the entire predicted depth map with the ground truth from three perspectives of luminosity, contrast, and correlation(structural similarity).

(1) Luminosity. The average value of all pixels in the whole image is used as a measure of luminance that can be expressed as $\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i$, N is the number of all pixels. The comparison metric for luminosity is defined as:

$$l(y, \hat{y}) = \frac{2\mu_y\mu_{\hat{y}} + C_1}{\mu_y^2 + \mu_{\hat{y}}^2 + C_1}$$

C_1 is a constant to avoid instability when the denominator is close to zero.

(2) Contrast. Contrast is a measure of the degree of intensity distribution in an image. The standard deviation $\sigma_x = (\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2)^{\frac{1}{2}}$ is a measure of spread. The comparison metric for the spread is defined as:

$$c(y, \hat{y}) = \frac{2\sigma_y\sigma_{\hat{y}} + C_2}{\sigma_y^2 + \sigma_{\hat{y}}^2 + C_2}$$

C_2 is a constant to avoid instability.

(3) SSIM. Structrual similarity index(SSIM) stands for the correlation between the ground truth and depth estimation that can be expressed as:

$$s(y, \hat{y}) = \frac{\sigma_{y\hat{y}} + C_3}{\sigma_y\sigma_{\hat{y}} + C_3}$$

C_3 is a constant to avoid instability.

The **SSIM** can be obtained by multiplying the above three:

$$SSIM(y, \hat{y}) = \frac{(2\mu_y\mu_{\hat{y}} + C_1)(2\sigma_{y\hat{y}} + C_2)}{(\mu_y^2 + \mu_{\hat{y}}^2 + C_1)(\sigma_y^2 + \sigma_{\hat{y}}^2 + C_2)}$$

Because SSIM has an upper bound of one and we try to maximize the SSIM value, we define it as a **SSIM loss** as follows.

$$L_{SSIM}(y, \hat{y}) = \frac{1 - SSIM(y, \hat{y})}{2}$$

For the whole loss function, we give different weights to three loss terms. Besides, we also try to use L1 loss to train a model.

2.4 Inpainting Method

The ground truth depth information of KITTI is obtained by using a lidar sensor, so the depth information is actually in a 3D matrix. We use an inpainting method to fill missing or invalid depth values. The core of this colorization method is that neighboring pixels in space with similar intensities should have similar colors. Anat Levin et al. formalize this premise using a quadratic cost function and obtain an optimization problem that can be solved efficiently using standard techniques. The authors propose the constraint that two closed pixels should have similar colors if their intensities are similar[10]. The inpainting code is provided on the NYU website. As shown in Figure 3, we obtain the depth ground truth from the pair of one RGB image and corresponding dense depth point clouds using this inpainting method.

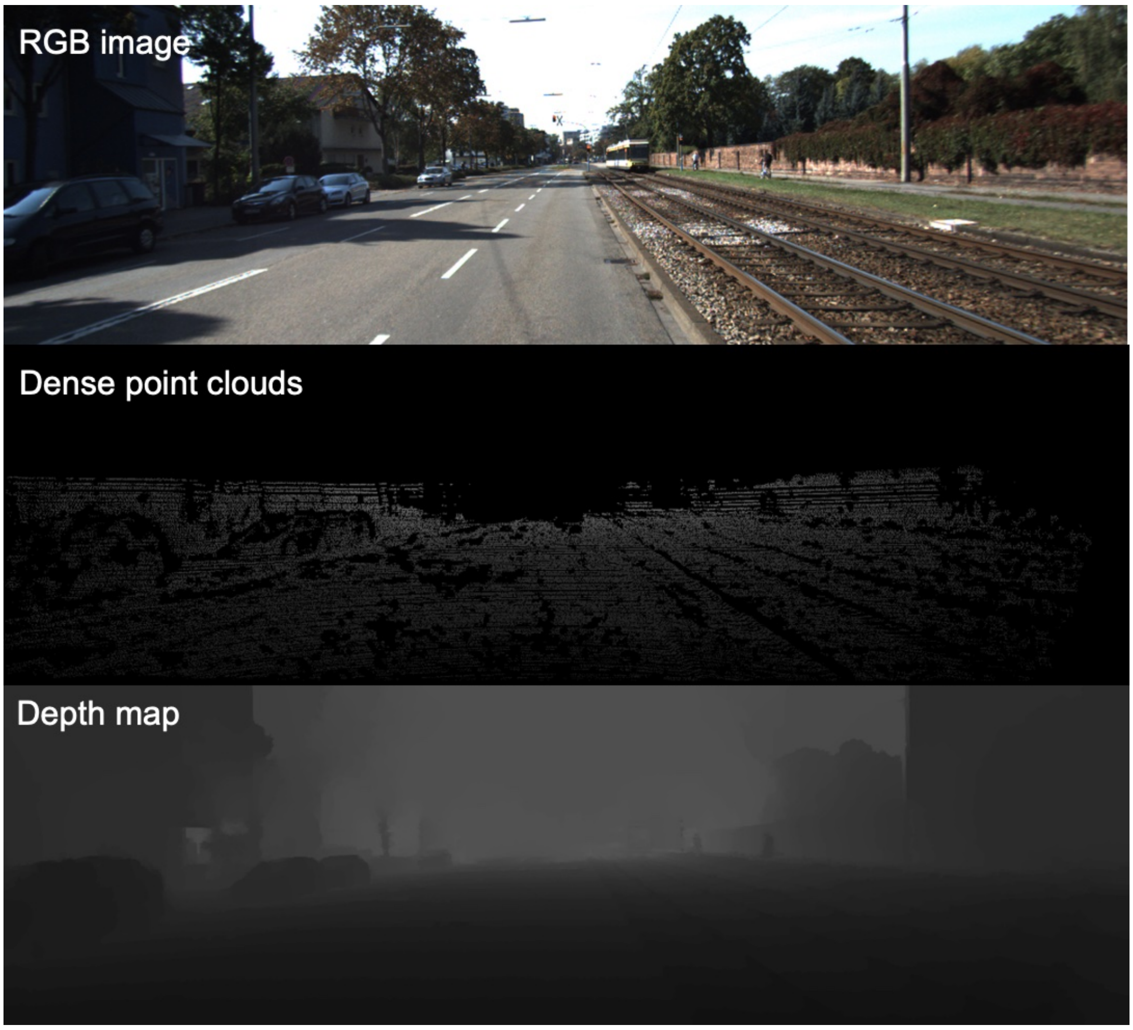

Figure 3: Filling the missing or invalid depth values in groundtruth using the inpainting method: input RGB image, corresponding dense depth point clouds and output ground truth.

3 Results

3.1 Implementation Details

We implemented our encoder-decoder network using PyTorch on CoLab and trained on one Tesla T4 or Tesla P100 GPU with 13GB memory. The encoder is the DenseNet-169 pretrained on ImageNet. We adjust different decoder architectures to train models and compare their performance. The decoders we use are the original decoder with skip-connections, simple decoder without skip-connections, original decoder adding one more upsampling, original decoder adding batch normalization layer, original decoder with equal-weighted loss function, and original decoder with L1 loss function. We use the ADAM optimizer with learning rate 0.0001 and batch size of 8 in our experiments. Training for the NYU depth v2 dataset needs 90 minutes per epoch for most of our decoder networks. When using the original decoder architecture with L1 loss function, it takes about 70 minutes for one epoch. However, for the modified decoder architecture adding one more upsampling layer spends longer time training the model, which takes about 2 hours for one epoch. Training for the KITTI dataset that contains 3500 pairs of RGB images and corresponding depth maps needs 10 minutes per epoch when using the original decoder architecture with skip-connections and the equal-weighted loss function.

3.2 Datasets

We train the model on the raw datasets of both NYU Depth v2 and KITTI that are the benchmark and commonest training datasets for the monocular depth estimation.

3.2.1 NYU Depth v2

The NYU Depth dataset v2 is composed of 464 indoor scenes with the resolution of 640x480 as recorded by both the RGB and Depth cameras from the Microsoft Kinect that can collect ground truth of depth directly. The upper bound of the depth maps is 10 meters. The dataset contains 120K training data and 654 testing data. Because sampling rates of the RGB and depth cameras are different, depth maps and RGB images are not one-to-one mapping. Each depth image is matched with its closest RGB image in time as RGB-depth map pair[5]. Besides, depth maps of the raw dataset contain missing or invalid depth values caused by shadows or low albedo surfaces. Using the inpainting method can fill missing depth values[10].

3.2.2 KITTI

The KITTI is a large dataset that is composed of 56 outdoor scenes including the "city", "residential" categories of the raw data, and so on. Each scene consists of stereo RGB images captured by cameras mounted on a moving vehicle and corresponding sparse 3D laser scans that are sampled at irregularly spaced points by a rotating LIDAR depth sensor[11]. The depth maps have an upper bound of 80 meters. The RGB images are 1224x376 and corresponding depth maps have a very low density with lots of missing depth values. Due to depth maps are sparse, we need to construct the ground truth depths before training the model. Here, we plan to fill missing depth values using the inpainting method[10].

3.3 Evaluation

An indicator of **RMSE** is often used in evaluating and comparing the performance of different depth estimating networks. This evaluation indicator is defined as:

$$\textbf{Root Mean Squared Error(RMSE):} \sqrt{\frac{1}{n}\sum_{i\in n}||d_i - \hat{d}_i||^2}$$

where d_i stands for the predicted depth value of pixel i, $\hat{d}_i$ is the true depth value of pixel i, and n is the total number of pixels for the whole depth image.

3.4 Results on NYU v2

Pre-processing. The NYU depth v2 dataset we use contains about 50k pairs of RGB images and corresponding depth maps. Also, we assume the upper bound for indoor scenes is 10 meters, so corresponding target depth maps are clipped to the range [0, 10] in meters.

Many variations on our standard model are considered. We train different models and batch sizes for five epochs and look at their effect on performance, as shown in Figure 4. The validation losses for the original decoder with skip-connections, the decoder add an upsampling layer, the decoder with batchnormalization layer and the simple decoder without skip-connections are displayed on the left. We find that using the original decoder with skip-connections has the lowest validation loss except for the decoder architecture adding one more upsampling layer. on the right, we can see the validation loss for different batch size 4,6 and 8. Setting batch size to 6 or 8 results in a better performance than the value of 4.

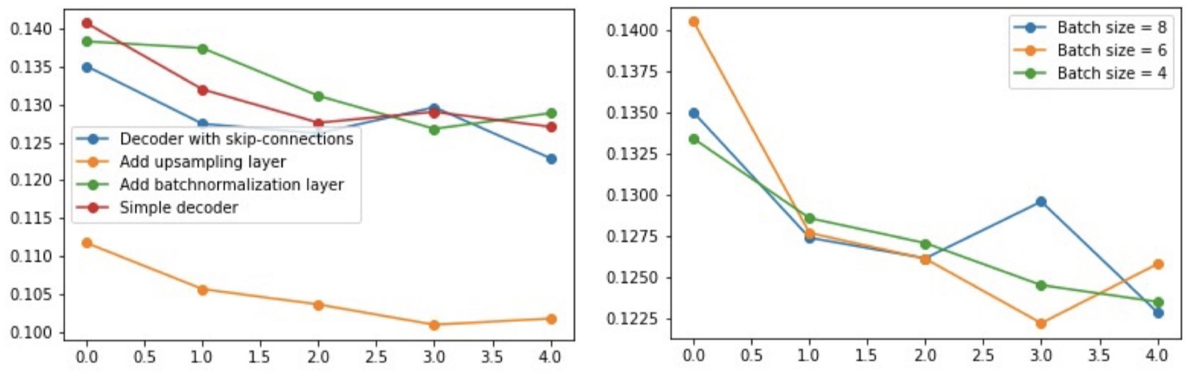

Figure 4: Comparisons of different models and batch size.

Considering modified loss function brings an influence on the validation loss, it's unfair to compare the validation loss for all models with different loss functions. Therefore, we set the batch size to 8 for all models, and compare their performance using an evaluation indicator of RMSE, as shown in Table 4. The best result is marked in red, and the second-best is underlined. The result of using the original decoder architecture with the equal-weighted loss function is better than those produced by other models. The depth estimation on the NYU depth v2 dataset using all models we propose is shown in the appendix. We find that these models can produce a good performance, especially for the original encoder-decoder architecture with the equal-weighted loss function and L1 loss. Compared to the equal-weighted loss function that reconstructs clear edges, the L1 loss can restore indistinct details, such as the bathroom ladder behind the bathtub having a similar color to the wall, as shown in the second column in Figure 9.

In addition, we plot training loss and validation loss for the original decoder architecture with various loss functions in Figure 5. We can observe a downtrend for these two losses: the training loss keeps decreasing, but the validation loss becomes steady after dropping fast at first.

Root Mean Squared Error(RMSE)						
Method	Standard Decoder	Simple Decoder	Add Upsampling	Add BN	Equal Weighted Loss	L1 Loss
RMSE	0.9539	0.9616	0.973	1.0748	0.9459	<u>0.9508</u>

Table 4: Comparisons of different methods on the NYU Depth v2 dataset.

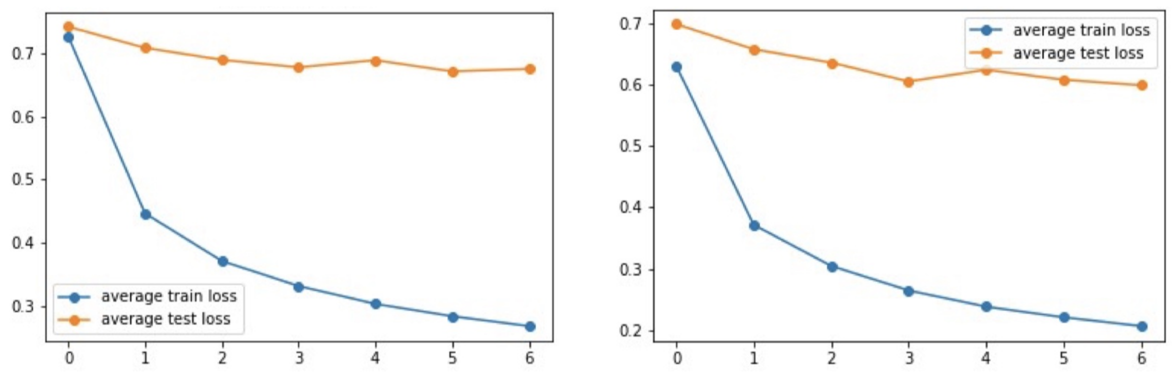

Figure 5: Training loss and validation loss for different loss functions: **Left.** Equal weighted loss function. **Right.** L1 Loss function.

We test four pictures taken in the Engineering Hall using the original encoder-decoder architecture with the equal-weighted loss function in Figure 6. The prediction is indeed a good depth reconstruction of the environment, and the boundaries of objects are restored. However, this model is sensitive to color changes caused by light and shadows. We can find that the depth estimation on the plane floor in the second column changes dramatically due to the shadow caused by the trash bin. Besides, it's hard to distinguish the depth of closed objects with similar colors, for example, in the third column the pendant light and the cabinet behind it have similar but unreasonable depth predictions. On the other hand, we observe that the model is not applicable for the second scene because it may be seen as an outdoor scene due to transparent glass.

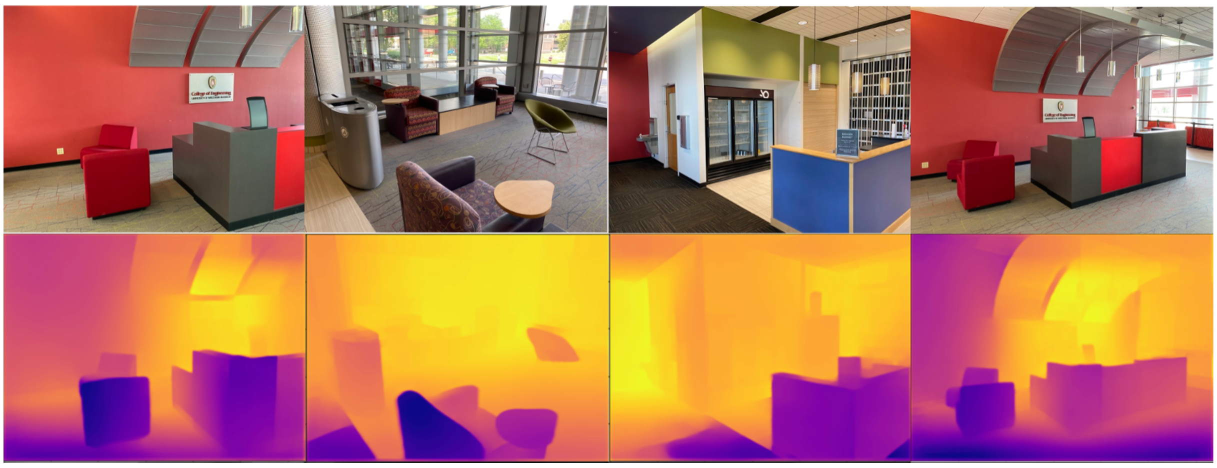
Figure 6: Estimated depth maps of indoor scene in Engineering Hall.

3.5 Results on KITTI

Pre-processing. The KITTI dataset we use contains about 3500 pairs of RGB images and corresponding depth maps. Besides, we assume the upper bound for outdoor scenes is 80 meters, so corresponding target depth maps are clipped to the range [0, 80] in meters. On the other hand, our encoder's architecture expects image dimensions to be divisible by 32 due to the downsampling operation in encoder architecture. Therefore, we need to resize input RGB images before training if their width or height cannot be divisible by 32. For the KITTI dataset, the resolution is 1224×376, so we need to resize it to 1280×384 before feeding it to the encoder-decoder network.

We use the original encoder-decoder architecture with the equal-weighted loss function to train a new model for the KITTI dataset. Considering the dataset we use is relatively small, we offer 50 epochs for training a model. As shown in Figure 7, we see the training and validation losses have a similar downtrend to the NYU model: the training loss keeps decreasing, but the validation loss drops slowly. It is worth mentioning that no overfitting occurred during training time because we do not find that the validation loss has an upward tendency with the increase of epochs.

In Figure 8, we test two pictures taken on the state street with the help of the pre-trained KITTI model. This model also achieves great performance. We can observe pedestrians and people cycling through the street in the distance, and the buildings and trees can be predicted reasonably. However, compared to the depth estimation on the NYU depth v2 dataset, the prediction for outdoor scenes loses lots of details and clear boundaries due to the complexity of the training scene and the small training set. The predicted results on the KITTI dataset as displayed in Figure 10 in the appendix. We can see the predicted images are pretty good compared to the ground truth. People, cars, and trees are reconstructed clear.

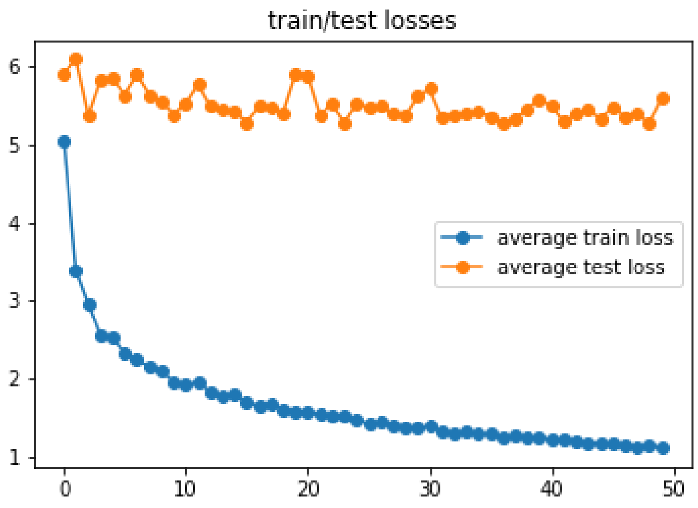

Figure 7: The average loss of the train and validation sets at each epoch for KITTI dataset.

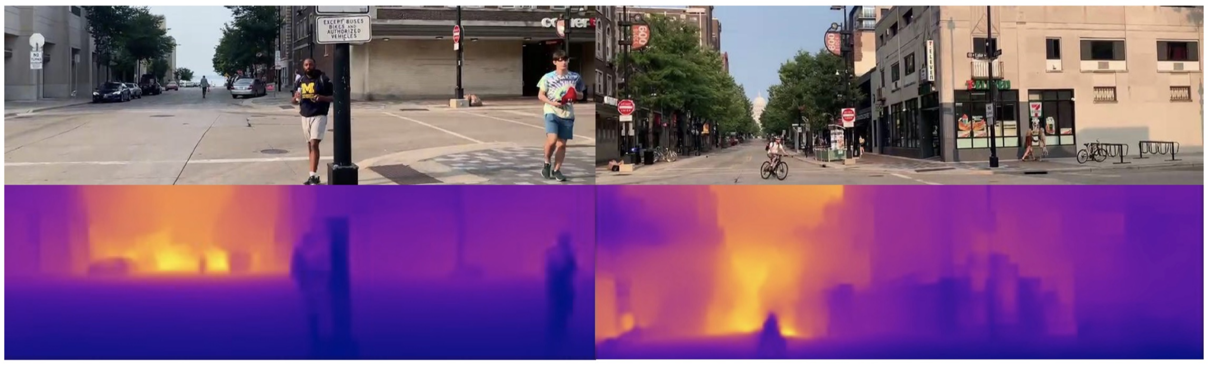

Figure 8: Estimated depth maps of outdoor scene on the state street in Madison.

4 Conclusion

In this work, we build an encoder-decoder network with the help of pretrained DenseNet-169 network to estimate depth map from a single RGB image for indoor and outdoor scenes and optimize the decoder architecture based on the idea of feature reuse from the DenseNet. Besides, we define the weighted sum of three loss functions that balances between the accuracy of depth map prediction and also the reconstruction of high-frequency details in the depth map. Our optimized models achieve good and stable performance with high resolution on NYU depth v2 dataset and KITTI dataset. Even though the photo processing speed is not enough for a real-time system, the outputs of our models are still reasonable. However, many questions are on the limits of our models. Our model is sensitive to light and color, for example, it is really hard for the model to distinguish them if the objects and background have a similar color. In addition, there are several things we can do to improve our project. First, due to time and resource limitations, we only used equal weights for the loss function for each model. Ideally, we should use a loop to determine the best weights for each loss function term. Second, we use the only L1 loss for the last training run. Even though the evaluation result was not the best, it has the best visual results compared to other models. We could use various combinations of loss function weights in the future. Third, we only use 3500 outdoor scenes to train a model that is a small fraction of the entire KITTI dataset. We should use the entire KITTI dataset to improve the outdoor model next. Finally, although the monocular depth estimation method has high accuracy, the estimated depth map we obtained is still fake. It may be easily deceived by a photo posted on the wall. We can try to learn a depth completion network to solve this problem that combines estimated depth maps with true sparse depth information to predict more real and accurate depth maps[12].

4.1 Acknowledgements

Renwen Cui is responsible for the construction of the encoder-decoder architectures, code for data reading of the NYU depth v2 dataset, and code for training a model.
Zhenye Li is responsible for the definition of the loss function, code for the loss function, preprocessing of filling missing depth values for the KITTI dataset, code for data reading of the KITTI dataset.
Renwen Cui and **Zhenye Li** evaluate and analyze the results together.

References

[1] Zhao, ChaoQiang, et al. "Monocular depth estimation based on deep learning: An overview." Science China Technological Sciences (2020): 1-16.

[2] I. Alhashim and P. Wonka, "High quality monocular depth estimation via transfer learning," arXiv:1812.11941, 2018.

[3] L. Zou and Y. Li, "A method of stereo vision matching based on opencv," in 2010 International Conference on Audio, Language and Image Processing. IEEE, 2010, pp. 185–190.

[4] K. Yoneda, H. Tehrani, T. Ogawa, N. Hukuyama, and S. Mita, "Lidar scan feature for localization with highly precise 3-d map," in 2014 IEEE Intelligent Vehicles Symposium Proceedings. IEEE, 2014, pp. 1345–1350.

[5] 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, 2014, pp. 2366–2374.

[6] E. Shelhamer, J. T. Barron, and T. Darrell, "Scene intrinsics and depth from a single image," in Proceedings of the IEEE International Conference on Computer Vision Workshops, 2015, pp. 37–44.

[7] 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). IEEE, 2016, pp. 239–248.

[8] Huang G, Liu Z, Laurens V, et al. Densely Connected Convolutional Networks[J]. IEEE Computer Society, 2016.

[9] J. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing, 13:600612, 2004.

[10] Levin, Anat, Dani Lischinski, and Yair Weiss. "Colorization using optimization." ACM SIGGRAPH 2004 Papers. 2004. 689-694.

[11] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. "Vision meets robotics: The kitti dataset." I. J. Robotics Res., 32:1231–1237, 2013.

[12] Bergman, A. W., D. B. Lindell, and G. Wetzstein. "Deep Adaptive LiDAR: End-to-end Optimization of Sampling and Depth Completion at Low Sampling Rates." 2020 IEEE International Conference on Computational Photography (ICCP) IEEE, 2020.

Appendix

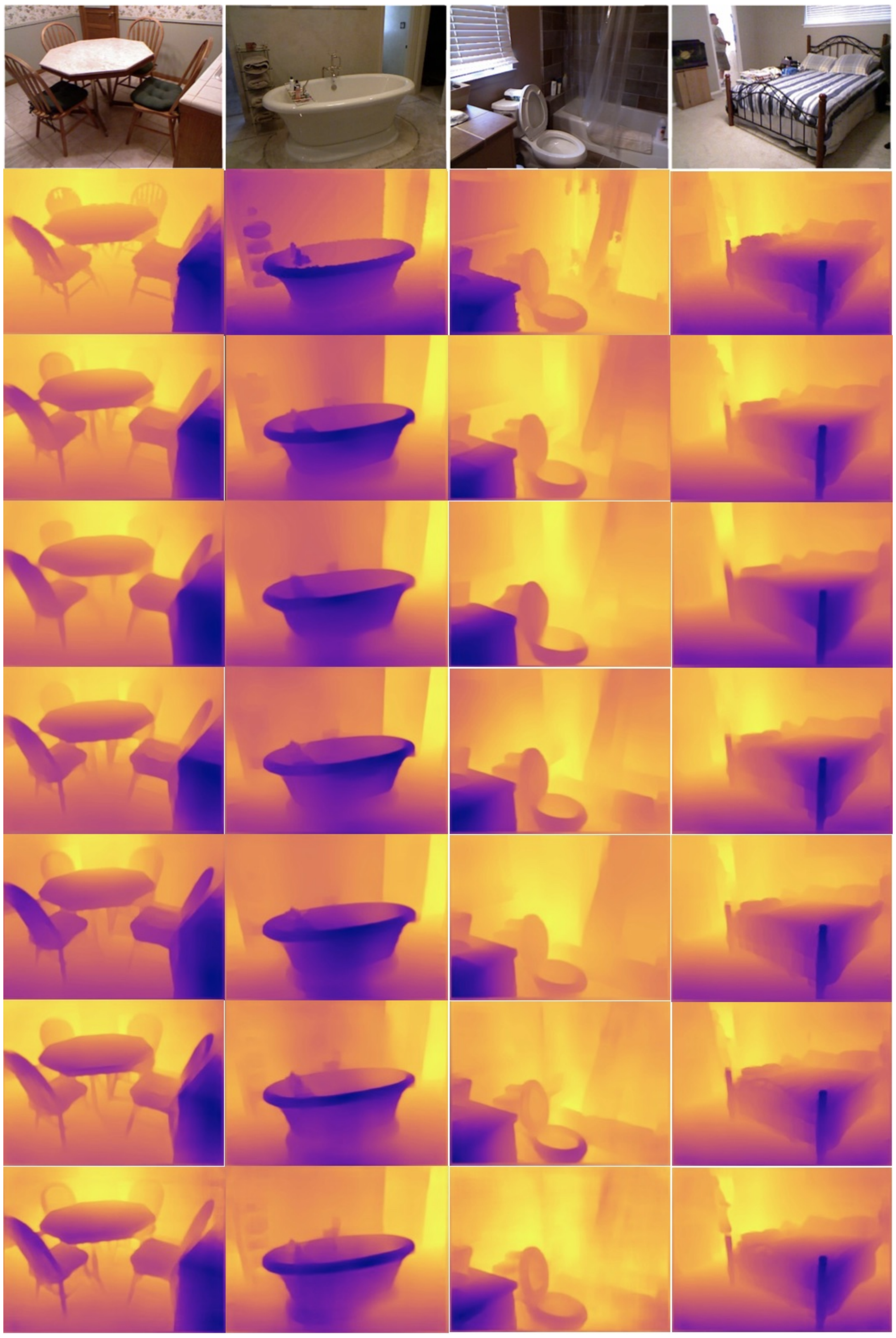

Figure 9: Indoor scene from the KITTI dataset: the order from top to bottom is input RGB images, ground truth, original decoder with skip-connections, simple decoder without skip-connections, original decoder adding one more upsampling, original decoder adding batch normalization layer, original decoder with equal weighted loss function, and original decoder with L1 loss function.

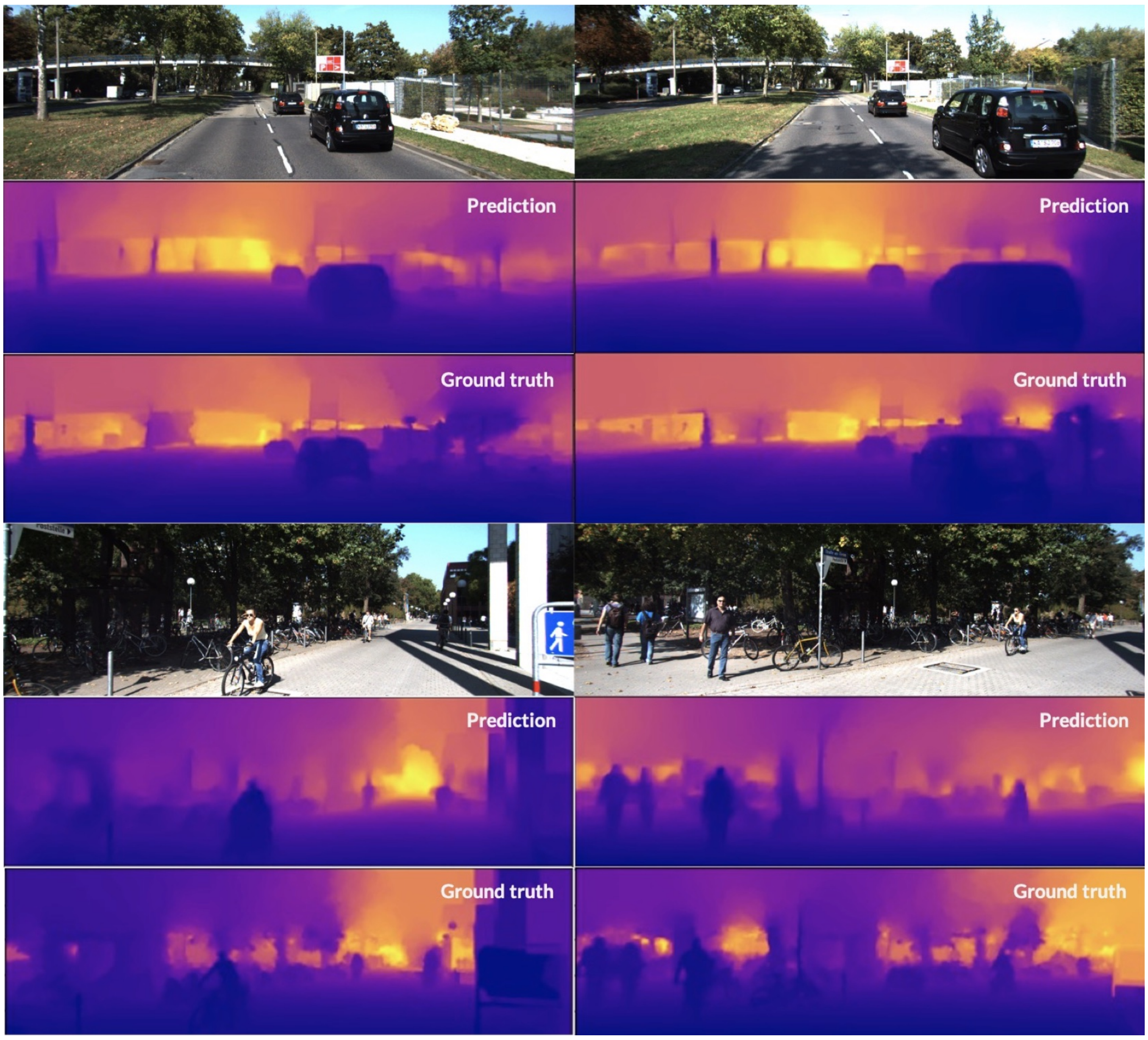

Figure 10: Outdoor scene from the KITTI dataset: input RGB images, our estimated depth maps, and ground truth.