CS 7643: UltraDepth

HAN UL (BRIAN) LEE
hanbrianlee@gatech.edu

Naveen Kumar Aproop
naproop3@gatech.edu

Nolan Foster
nfoster34@gatech.edu

Abstract

This paper aims to assess how much gain in depth estimation accuracy can be achieved by fusing in LIDAR with monocular camera. LIDAR data is sparse and semantically lacking but more accurate depth information, where its accuracy scales inversely proportional to the distance away from ego-vehicle, and camera data is dense and semantically rich but lacks in explicit depth information due to its 2D projection nature of the 3D world. We explore several existing approaches that solve depth estimation and show that early fusion techniques such as [5], which we call "reference paper" in this report, can help gain better accuracy by bringing in LIDAR, with some unique additional layers and additional considerations and details to building/training such network. We provide results indicate that we beat the current state of the art based on our data selection, but cannot conclusively claim that our network beats existing works due to difference in dataset selection.

1. Introduction/Background/Motivation

In autonomous driving domain, there is a need to deal with challenges in trying to localize the self-driving car on a map to within few centimeters. The key input that could help in achieving this is perception. By utilizing information from the host vehicle environment such as the distance to various traffic signs, lane edges, other moving vehicles, etc. a self-driving car can determine where it is on the road both longitudinally and laterally. Today, through the use of deep neural networks, perception systems have made great strides and can provide accurate results in semantic segmentation (i.e. freespace estimation) and object detection (2D and 3D) compared to the traditional computer vision techniques. But a 2D scene image whereupon the 3D world is projected onto camera, by itself, often cannot provide accurate enough depth information to enable 3D detection capability, which is crucial for planning, controls and also localization and mapping. There have been several methods proposed [17] using deep neural networks in recent years that are able to achieve state of the art results using supervised, unsupervised and semi-supervised learning methods through the use of monocular video stream, stereo pair and depth sensors such as LIDAR, RGB-D camera, etc.

In this paper, we are interested in studying how much of a gain in depth estimation accuracy can be achieved by fusing in LIDAR with monocular camera. Specifically, we explored depth estimation performance by using only monocular camera versus using LIDAR in the mix and compared their performances. Also, for monocular camera-only method, we evaluated a more sophisticated method, monodepth2 [7], as well for comparison which uses a monocular camera video stream and to estimate a dense depth map by utilizing the temporal information. Different hyperparameters, two different loss functions, and three different fusion methods were evaluated and discussed. We show that we were able to achieve better metrics than the reference paper although it should be taken with a grain of salt as the comparison is not completely apples to apples (even though the same KITTI [6] dataset pool was utilized, the exact split for train/validation has impact on the metrics).

2. Dataset

We use the KITTI [6] annotated depth maps for all experiments. KITTI data consists of City, Residential, Road, Campus and Person data from recordings of the Annieway autonomous driving platform. The platform consists of high resolution color and gray scale video cameras, a Velodyne lidar and GPS system. For monocular depth estimation the synchronized RGB camera images, sparse lidar scans with points from 0 to 80m and ground truth annotated depth maps were used (see Figure 6). The annotated depth maps dataset lacks the RGB camera images needed therefore the left images were copied from the "raw data" of the KITTI data set. This resulted in a data set with 43k training and 3k validation pairs of ground truth depth maps, lidar scans and left camera images. Since we were interested in monocular depth estimation and how much LIDAR fusion can improve the performance, the right camera images were omitted. From the validation set a held-out test set of $\sim$600 images was extracted for evaluation. The test set was carefully selected to ensure it contained a representative mix of the dataset categories. Each data pair was normalized and center-bottom-cropped, this helps speed up the learning and

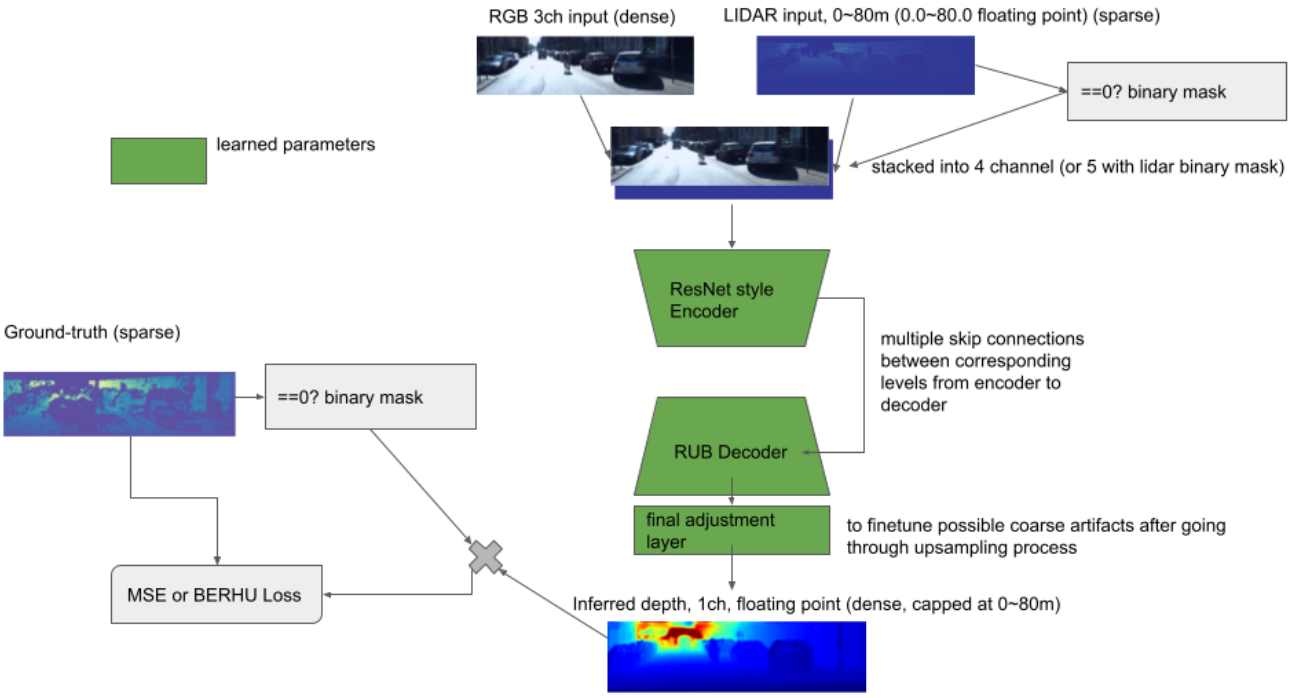


Figure 1: Our Ultradepth architecture. Encoder was a sequence of multiple convolution layers (kernel size=3) with max-pooling (kernel size=2), where the depths were [32, 64, 128, 256, 512], where the final encoded bottleneck feature map had 512 depth. This latent representation was then decoded by RUB in the reverse order via concatenation [256+128, 128+64, 64+32, 1], where the numbers on the right side of the plus sign indicate the decoded feature depth which gets concatenated to the corresponding features from the encoder layer with the same dimensions. The final output layer is depth of 1.

convergence. The crop reshaped the images to 928 (width) by 256 (height). This was necessary to remove the top of the images which did not have and lidar/ground-truth information.

3. Approach

Since depth estimation requires pixelwise inference, the required architecture has a lot of similarity with semantic segmentation networks, where most recent works such as [10], [1], [14], consist of encoder-decoder concepts where hierarchical semantic extraction is done on the encoder side and through skip connections from encoder to decoder, deep semantic understanding of the scene gets uncoiled whilst adding details from the encoder side. Hence, naturally we looked at architectures that are similar in idea with the popular Unet [14], where more specifically as we wanted to compare the performance gain with the addition of sparse, more accurate lidar data, we chose [5] as our base architecture, which from hereon we note as "reference paper". This architecture fuses in lidar and camera data early on from their raw forms, as opposed to extracting features separately and fusing them later on such as [4]. There are other approaches such as [7] as well where ground-truth depth data do not exist but rely on temporal difference with camera images and disparity based re-projection losses, but in order to keep the project as simple as possible for the purpose of comparing with lidar vs without lidar, and considering the dataset we chose, we decided to have our base model be inspired from [5]. The specific early-fusion approach made an intuitive sense due to the different nature of dense, yet inaccurate RGB data and sparse, but accurate lidar data. Instead of imposing our own bias on what loss targets should be used to extract desired features from the two sensor modalities, we thought letting the network decide how and what features extract in order to best improve the inference accuracy.

As for handling the holes in LIDAR data, we explored two different methods: 1. just using raw LIDAR data where 0 represents both holes and actual 0m distance, 2. also concatenating a binary mask along with the RGB and LIDAR data (thereby making 5 channel input data, see Figure 1). We thought giving more information to the network (i.e. which pixels contain LIDAR data that exists) can help the network figure out how to better handle sparse LIDAR data. We discuss the resulting differences in more detail in Section 4.4. We also explored adding 3x3 convolutional fine

adjustment layers as was done in [1], which we expect to be able to handle any coarse artifacts that occur as a result of upsampling process in a learnable way. The architecture we used is depicted in Figure 1, which overall looks similar to the architecture used in [5], but has slight differences in its implementation as the whole model was written from scratch using our own resnet-style [8] blocks with LeakyReLU with different depths (see caption in Figure 1 for more details). For decoding operation, the same Residual Up-Projection Block (RUB) [5] with scale of 2 was utilized as was used in the reference paper, to take advantage of the gradient highways during decoding process.

3.1. Challenges

We anticipated some troubles in handling the sparse ground-truth data for KITTI [6] (while NYU-Depth V2 [12] has dense ground-truth data, KITTI has many holes where ground-truth data do not exist). In surveying different depth estimation papers that utilized KITTI dataset, we've come across solutions such as masking them by the presence of data or not, or using in-painting to fill the gaps [2]. In order to quickly get results and focus more on analysis, we did not spend time on in-painting or other methods of filling in the holes, and we opted to simply ignore the no-data regions so that the network will only learn from regions where ground-truth data do exist.

Our first attempt in training the network did not go well due to high learning rate causing too much fluctuations in the gradients and also with misinformed method on cropping the data (cropped the center instead of bottom-center, and with wrong resolutions), we were experiencing high RMSE values. We later on got around the problem by doing the proper crop, increasing the batch size for better batch statistics and losses, and eventually doing grid-search with viable range of parameters we found which are shared in Section 4.1.

4. Experiments and Results

4.1. Hyperparameters and Experiments

We mainly performed three fusion modes and associated grid search experiments to evaluate the effectiveness of using lidar in conjunction with images for depth estimation. The model was built with pytorch [13] and the hyper-parameters tuned with ray tune [9]. We used SGD as the optimizer as it provides more raw performances that can give us more direct comparisons between parameters for analysis. All experiments used the same model with different fusion modes which require changes in the first layer of the encoder and the input channel dimension. Each fusion mode was individually tuned, train and validated on a hyper performance compute platform with 4 NVIDIA Tesla P-100-SXM2 GPU's with 16GB memory each. The learning rate, batch size, number of layers and loss functions were tuned via grid-search. Each experiment was tuned on 5% of the training data to save time and compute resources, which seemed to agree with our preliminary manual search results, so we considered this amount of data to be sufficient in representing the overall distribution for hyper-parameter searching. Taking hints from the parameters used in the reference paper [5] the learning rates varied from 0.01 to 0.001. The batch sizes were tested with both 16 and 64 (per GPU this would equal 4 and 16 mini batch sizes) but in all cases 16 provided the better results. A larger batch size is beneficial in speeding up training if there are enough compute resources but it also can affect the performance of the final model. In practice, it is known that smaller batch sizes provide better generalization capability by feeding in more noisy data during training (data augmentation side-effect, see Figure 2) [11]. The best loss between MSE and BERHU was used for each experiment. The last parameter tuned was the number of layers in the encoder. The layers were varied between [2,2,2,2,2] and [2,4,6,8,10]. This was to see the effect of increasing the receptive field as well as having more parameters and compositional feature extraction capability. With the deeper network there was an opportunity to achieve lower RMSE because of the larger receptive field but it was prone to over fitting due to more parameters (Figure 3). If given enough time and resources a much larger variation of parameters could be used to potentially improve the results.

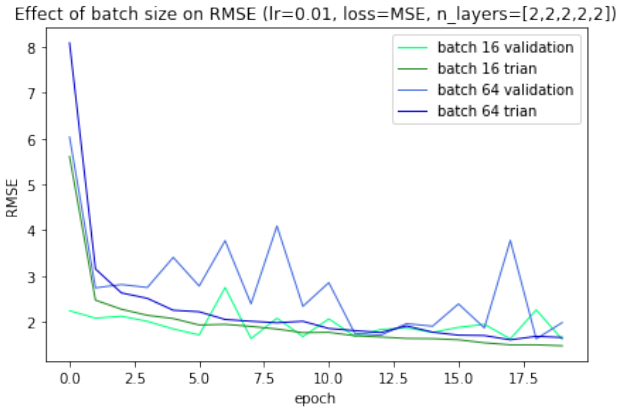


Figure 2: Batch size effect on RMSE for the lidar fusion mode. You can see that batch size 16 reaches lower RMSE (better) metric than the 64 for both train and validation sets.

4.2. Metrics

We computed standard depth error metrics similar to those in [17] and [7] and proposed in [3] for easier comparison of results with other works. These include: Root mean squared error (1), Log root mean squared error (2), Relative absolute error (3), Relative squared error (4), Accuracies

Fusion modes	Learning Rate	Batch Size	Loss	Layer Size	RMSE
cam only	**0.0075**	**16**	**BERHU**	**[2,2,2,2,2]**	**4.73**
	0.005	16	MSE	[2,2,2,2,2]	4.83
	0.0075	16	MSE	[2,2,2,2,2]	5.69
	0.005	16	BERHU	[2,2,2,2,2]	5.95
	0.01	16	BERHU	[2,2,2,2,2]	6.61
camera + lidar	**0.01**	**16**	**MSE**	**[2,2,2,2,2]**	**1.64**
	0.005	16	MSE	[2,2,2,2,2]	1.67
	0.005	64	MSE	[2,2,2,2,2]	1.71
	0.0075	64	MSE	[2,2,2,2,2]	1.77
	0.0075	16	BERHU	[2,2,2,2,2]	1.90
camera + lidar with binary mask	**0.01**	**16**	**MSE**	**[2,2,2,2,2]**	**1.36**
	0.0075	16	MSE	[2,2,2,2,2]	1.49
	0.0075	64	MSE	[2,2,2,2,2]	1.62
	0.005	64	MSE	[2,2,2,2,2]	1.67
	0.005	16	MSE	[2,2,2,2,2]	1.68

Table 1: Top 5 Hyperparameter combinations for each experiment

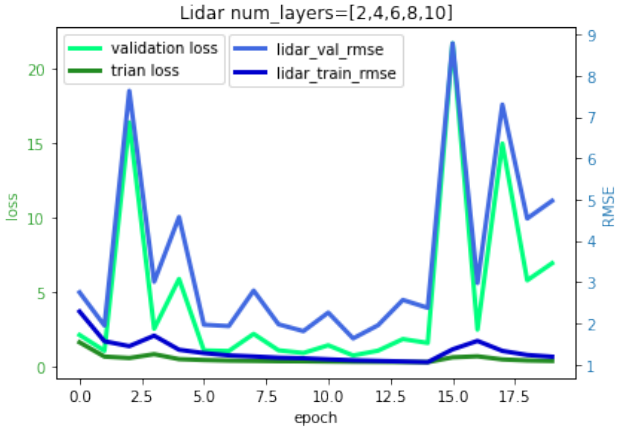


Figure 3: Results of Lidar trained with camera images and additional layers added. Refer to Lidar fusion mode in Table 1 for the remaining parameters.

$\mathbf{a1} = \mathbf{a} < \mathbf{1.25^1}$, $\mathbf{a2} = \mathbf{a} < \mathbf{1.25^2}$ and $\mathbf{a3} = \mathbf{a} < \mathbf{1.25^3}$ (5) as follows:

RMSE:

$$\sqrt{\frac{1}{|N|}\sum_{i \in N} ||d_i - d_i^*||^2} \tag{1}$$

RMSE Log:

$$\sqrt{\frac{1}{|N|}\sum_{i \in N} ||log(d_i) - log(d_i^*)||^2} \tag{2}$$

Abs Rel:

$$\frac{1}{|N|}\sum_{i \in N} \frac{|d_i - d_i^*|}{d_i^*} \tag{3}$$

Sq Rel:

$$\frac{1}{|N|}\sum_{i \in N} \frac{||d_i - d_i^*||^2}{d_i^*} \tag{4}$$

Accuracies:

$$\%\text{ of } d_i \text{ s.t. } max(\frac{d_i}{d_i^*}, \frac{d_i^*}{d_i}) = \mathbf{a} < thresh \tag{5}$$

Where $|N|$ is the number of pixels, d_i and d_i^* are the predicted and ground truth depths for pixel i respectively.

4.3. Losses

In terms of losses, we compared Mean Squared Error (**MSELoss**) of PyTorch and reverse Huber (**Berhu**) loss [18] which is calculated as follows:

$$L_{Berhu} = \begin{cases} |e|, & \text{if } |e| \leq c \\ \frac{e^2+c^2}{2c}, & \text{otherwise} \end{cases} \tag{6}$$

Where $e = d_i - d_i^*$ (difference between inferred and ground-truth depth at each pixel) and $c = 0.2 * max_i(|e|)$ is the *threshold*. Berhu loss will be same as the L_1 if the threshold $|e| \leq c$, otherwise it becomes L_2 loss.

MSE loss learns to smooth and blur the object boundaries, where one shifted pixel depth inference can create large error with respect to the ground-truth pixel, by penalizing large errors more heavily, while Berhu avoids such issues [5] by allowing the huge contrast happening in edges to be left more as is. In our experiments, Berhu showed more stable (less variance/noise) loss/metric convergence trend than MSE, which is expected because it only soft-penalizes those large boundary errors for instance. The plots comparing the two losses and RMSE metric over 100 epochs with our full dataset are shown in Figures 4 and 5 respectively. Our grid search results showed that there does not seem to be a clear trend in which loss prevailed over the other in all hyperparameter combinations.

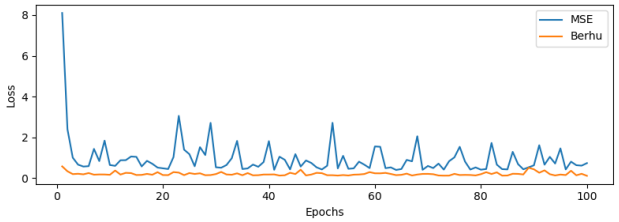


Figure 4: Loss plot for MSE vs Berhu losses for Validation set

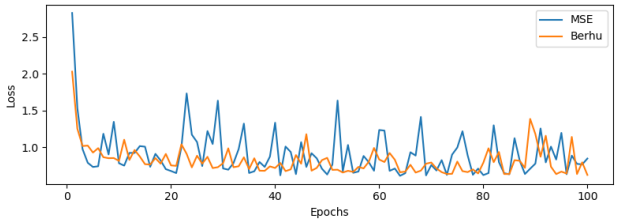


Figure 5: RMSE plot for MSE vs Berhu losses for Validation set

Experiment	**RMSE(m)**	**Abs Rel**	**a1(%)**	**a2(%)**	**a3(%)**
cam only	3.59	0.113	89.4	97.9	99.4
camera+lidar	**1.21**	0.0445	98.6	99.6	99.9
camera+lidar with final adjustment layer	1.24	**0.0379**	**99.1**	**99.8**	99.9
camera+lidar with mask	1.44	0.0422	99.0	99.8	99.9

Table 2: Experiment Results

4.4. Results

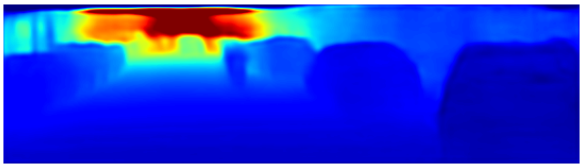
Here we compare performance of our experiments for depth estimation. The resulting metrics can be seen in Table 2 with the corresponding images in Figures 6 and 7.

The colors range from blue(0m) to red(80m). The far points in red align on the vantage point and the close areas around the edge and bottom of the image are properly marked as close in blue. These results were evaluated on the held-out test set using the model from the best epoch (RMSE as the criterion) which is denoted by the vertical line in Figures 8, 9 and 10. It can be seen that there are large spikes (especially those with LIDAR fusion) in losses at some epochs. We believe these are attributed to the network trying to figure out how to deal with sparse data and sometimes when it makes use of data non-existing pixels too much, it generates large errors, and also when the model tries to put too much emphasis on LIDAR data as opposed to smoother, semantic-rich RGB data. The model learned on camera only therefore has much smaller spikes since the input is all dense and so is the output. It is likely also related to using smaller batch sizes, which would have noisier losses. It can be observed in Figure 2 that reducing the batch size reduces the volatility of these spikes. Overfitting effect happens at different epoch per model, but it can be seen that from around 15 epochs, the validation and train losses start growing apart in Figure 9. But as mentioned, our final evaluations were done with the best epoch, which mostly happens before the overfitting region.

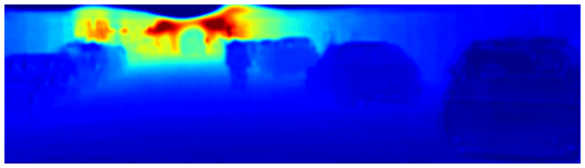
Image (a) in Figures 6 and 7 show the results for camera-only depth estimation, and it seems that camera images seem to be enough to produce rough semantic (object-level) representative depth estimation. But as can be seen in the far inferior metrics from Table 2 and looking closely to edges of the objects, they are slightly fuzzy as if Gaussian blur was applied. This fuzziness alludes to a lack of precision. This precision is improved when lidar data is fused (just LIDAR only in (b), LIDAR with mask in (c), and finally LIDAR with final adjustment layer in (d)). First, the lidar fused results show a deeper range of depth with the closer blues getting darker than what was seen in camera only and the far objects near the vantage point. Second, the edges and details are overall more defined and crisp, as seen by the more detailed shape for the bicyclist (b,c,d) which is more of a blur in (a) for Figure 6 and also the black car close to the camera having more defined edges (b,c,d) than the almost fading into the rest in (a) for Figure 7. Amongst the different experiments for LIDAR fusion, just fusing in LIDAR only (b) shows some undesired artifacts (i.e. holes in the rightmost car in Figure 6, while providing the mask as well (c) seems to improve this artifact. Using a final adjustment layer with LIDAR fusion (without mask) also seems to have similar effect of taking care of these artifacts. These artifacts likely occur because of the way ground-truth depth is sprase with holes, so the network isn't strictly learning to infer completely dense depth for every pixel. However, with information as to which pixels are data-less and which are, or by providing another fine-adjustment chance by giving it another layer at the end, the network seems to help bridge the gap thereby providing more dense depth estimation. While these sparsity induced artifacts are removed with the latter methods, looking closely, the contour of the car gets slightly over-flowing. This is another artifact induced by trying to do more dense depth estimation (since the network has to fabricate depth information where ground-truth doesn't exist, while learning to generalize, it overflows on edges a little bit).

Looking at the metrics in Table 2, the RMSE-wise using LIDAR only by itself for fusion seems to be prevailing. However, this is slightly misleading because the RMSE is computed against sparse ground-truth data (non-existent pixels are ignored) so the RMSE metric does not penalize against non-dense depth estimation really. Looking at the qualitative results, it seems that lidar with final adjustment layer strikes a good balance between proper dense depth estimation and good RMSE score (accuracy at each pixel with respect to the sparse ground-truth). Due to lack of time, the lidar with final adjustment layer variation did not have its own hyperparmeter grid-searching, but by just using the best parameters selected for "lidar" fusion mode (see Table 1), it worked well. If given more time, it would be interesting to see if applying both final adjustment layer and providing binary LIDAR data mask can benefit from each other, or if it will just further increasing the overflowing artifacts discussed here. Finally, it's worth noting that while "lidar with final adjustment layer" had slightly higher RMSE than just "lidar" mode, its Abs Rel, a1, and a2 metrics are superior meaning at each pixel depth, the relative errors were better (which would be more useful overall in applications such as autonomous vehicle driving where we desire the depth information to be good regardless of how far certain objects are from the ego-vehicle).

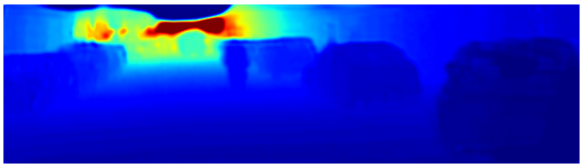
Left: RGB Camera, Middle: Lidar, Right: Ground truth



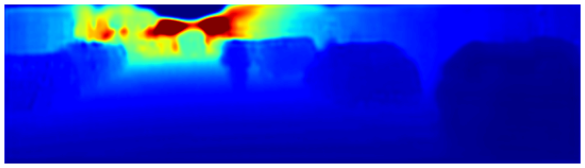
(a) Camera Only



(b) Camera + LIDAR



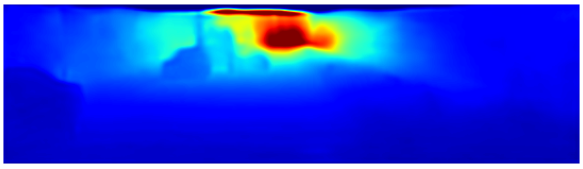
(c) Camera + LIDAR With Binary Mask



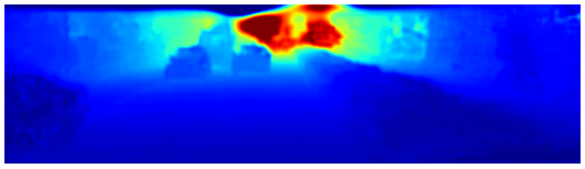
(d) Camera + LIDAR With Final Adjustment Layer

Figure 6: Notice the dark (0m) artifacts on the top side, this is induced by ground-truth data having no data on the top side, causing the dense depth inference to fail to generalize properly. This artifact can be removed by correcting the crop region to be lower, but since we decided to follow the crop size from the reference paper, we did not change the crop size. Note that the artifacts on the top side do not factor into metric computation because non-existent pixels are ignored.

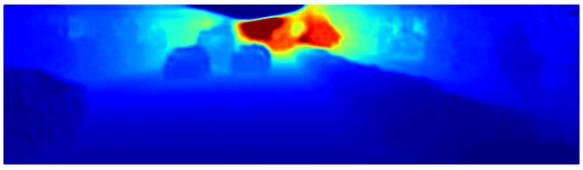
Left: RGB Camera, Middle: Lidar, Right: Ground truth



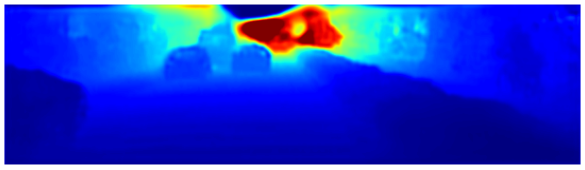
(a) Camera Only



(b) Camera + LIDAR



(c) Camera + LIDAR With Binary Mask



(d) Camera + LIDAR With Final Adjustment Layer

Figure 7: The same as the above figure but with a different example.

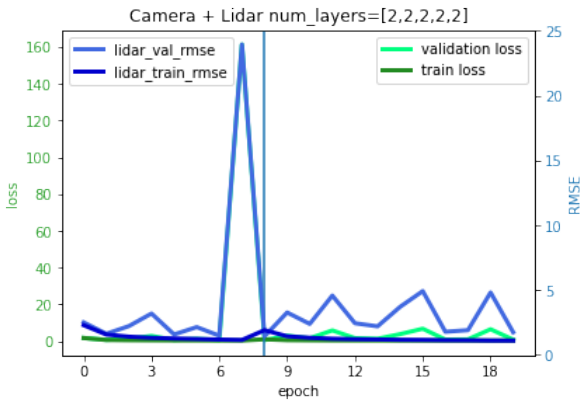


Figure 8: RMSE and loss plots for Lidar + camera. Refer to Table 1 for the parameters used.

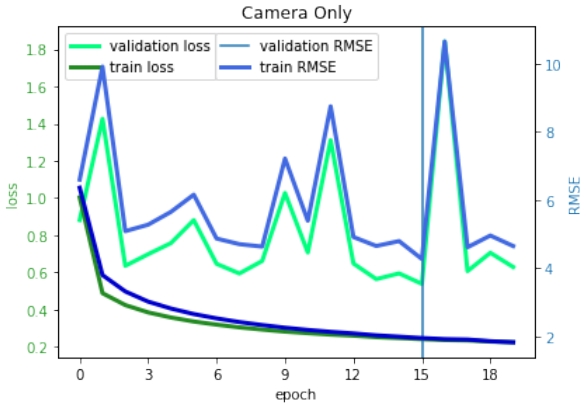


Figure 9: RMSE and loss plots for RGB camera images only. Refer to Table 1 for the parameters used.

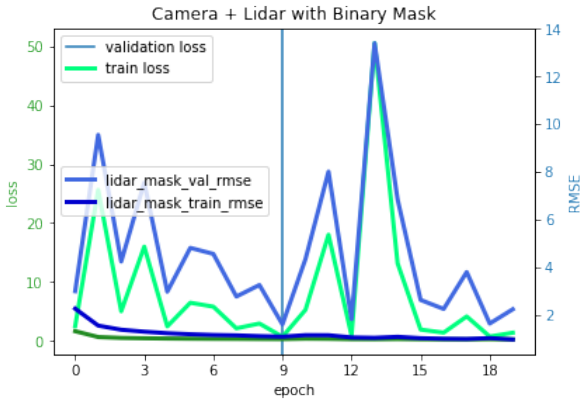


Figure 10: RMSE and loss plots for Lidar + camera images and binary mask. Refer to Table 1 for the parameters used.

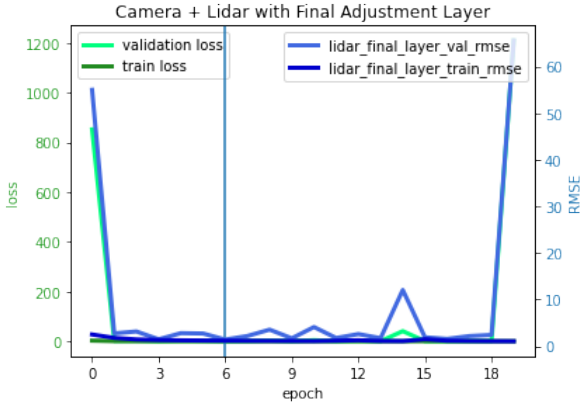


Figure 11: RMSE and loss plots for Lidar + camera with additional final adjustment layer. Refer to Table 1 for the parameters used.

4.5. Comparisons to Other Works

In this section, we compare our best results against the reference [5] and Monodepth2 [7]. The comparison is not apples-to-apples, since Monodepth2 is a camera only self-supervised method with different input image resolutions and the reference paper used data derived from KITTI odometry dataset with different splits, while we used the more post-processed depth estimation data from "KITTI depth dataset". Our test set is manually procured by ourselves (as mentioned in Section 2) because the official test set by KITTI depth estimation task is hidden and cannot be tested against unless we submit our work with significant contribution to the architecture together with a conference paper. We have yet decided whether our contributions are significant since the only major difference compared to the reference paper is the use of additional binary mask, adding final adjustment layer, and using LeakyReLU for activations in the encoder with slightly different layer depths.

As shown in Table 3, our metric seems to significantly beat the other works, but this comparison needs to be taken with a grain of salt because different data splits or different image dimensions/cropping can all cause quite different resulting metrics (as can be seen in the works such as [16]). Hence, we do not claim that the additional architectural decisions we made on top of the reference paper definitely improved the result, but looking at qualitative result comparisons, our result does seem to remove the hole artifacts that are present in [5].

Method	**RMSE(m)**	**Abs Rel**	**a1(%)**	**a2(%)**	**a3(%)**
Monodepth2 (camera-only with temporal information)	4.701	0.115	87.9	96.1	98.2
Reference (camera+lidar)	3.67	0.072	92.3	97.3	98.9
Ours (camera + lidar + finetuning layer)	1.24	0.0379	99.1	99.8	99.9

Table 3: Comparison to other works

5. Future Works

For potential future work, we would like to see a combination of lidar fusion techniques such as those discussed in this paper along with other techniques such as fusing in optical flow information [15] and temporal information (as done in [7]) that seem each to improve upon depth estimation to see how much of the performance boundary can be pushed. It would be interesting to see which of these techniques augment each other and oppose each other for better depth estimation.

6. Conclusion

In this paper, we investigated how much LIDAR fusion can improve depth estimation compared to using camera alone and some of the existing methods as well as our own additional architectural choices and their impacts. As can be seen in Table 2, we see a significant gain in metrics while fusing in LIDAR as opposed to using camera only. Figures 6 and 7 show that qualitatively fusing in LIDAR helps make the edges more crisp and more detailed, and further use of binary LIDAR mask or final adjustment layer can help remove random hole artifacts that can be introduced by sparse LIDAR fusion.

7. Work Division

See Table 4.

References

[1] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. 2, 3

[2] C. DanielHerrera, Juho Kannala, L. Ladicky, and J. Heikkilä. Depth map inpainting under a second-order smoothness prior. In *SCIA*, 2013. 3

[3] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. 3

[4] Chen Fu, Chiyu Dong, Christoph Mertz, and John M Dolan. Depth completion via inductive fusion of planar lidar and monocular camera. *arXiv preprint arXiv:2009.01875*, 2020. 2

[5] Chen Fu, Christoph Mertz, and John M Dolan. Lidar and monocular camera fusion: On-road depth completion for autonomous driving. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 273–278. IEEE, 2019. 1, 2, 3, 4, 7

[6] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset, 2013. 1, 3

[7] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow. Digging into self-supervised monocular depth prediction. October 2019. 1, 2, 3, 7, 8

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 3

[9] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018. 3

[10] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 2

[11] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *CoRR*, abs/1804.07612, 2018. 3

[12] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 3

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 3

[14] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]). 2

[15] Jiaxin Xie, Chenyang Lei, Zhuwen Li, L. Li, and Qifeng Chen. Video depth estimation by fusing flow-to-depth proposals. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10100–10107, 2020. 8

[16] Jiaxin Xie, Chenyang Lei, Zhuwen Li, Li Erran Li, and Qifeng Chen. Video depth estimation by fusing flow-to-depth proposals. *CoRR*, abs/1912.12874, 2019. 7

[17] Chaoqiang Zhao, Qiyu Sun, Chongzhen Zhang, Yang Tang, and Feng Qian. Monocular depth estimation based on deep learning: An overview, 2020. 1, 3

[18] L. Zwald and S. Lambert-Lacroix. The berhu penalty and the grouped effect. 4

Student Name	Contributed Aspects	Details
HAN UL (BRIAN LEE)	Led dataset selection, network architecture implementation, and initial training pipeline implementation. Analysis.	Surveyed datasets and existing papers to make initial selection on what type of architecture to implement. Implemented the architecture and succeeded initial training with small dataset, with initial hyperparameter searches. Took charge of writing Abstract, sections 3, 5, 6, analytical discussions regarding lidar fusion in 4.4, and overall report coherency edits and analytical reasoning.
Naveen Kumar Aproop	Supported initial development, training and coarse tuning. Analysis. Train and evaluated monodepth2.	Supported Brian's development making changes to data loader, plots, crop and adding metrics and loss functions to run on GPU server with full dataset. Performed several runs and provided feedback until stable results are achieved. Reviewed papers for monocular video sequence, selected and trained monodepth2 (and built KITTI eigen split dataset) in case we do optical flow with LIDAR. Completed report sections 1, 4.2, 4.3 and 4.5.
Nolan Foster	Build final training pipeline, Hyperparameter tuning, Results Analysis	Built full data set from KITTI sources and created data loader with augmentations. Implemented final training pipeline to work on high performance compute cluster based on Brian and Naveens initial pipeline work. Set up the pipeline to load full dataset, tune hyperparameters record metrics and evaluate the trained model. Ran experiments for lidar, camera only and lidar with mask and generated plots and result images for analysis and wrote sections 2, 4.4 and 4.1

Table 4: Contributions of team members.