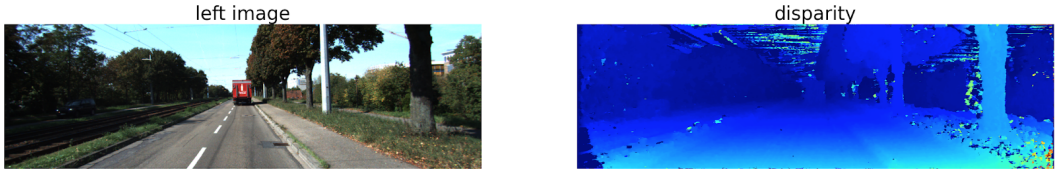
CS230

Binocular Stereopsis via Image Matching and Learned Filtering

Manoj Rajagopalan, Gordon Charles
rmanoj@stanford.edu, gcharles@stanford.edu



Abstract

We employ end-to-end deep learning to to calculate the left-disparity field of stereo images. We begin with pre-published work which uses a Siamese network which learns features for left- and right-images, and is unified by a simple inner-product operation in feature-space, per-pixel, to identify the disparities. We extend this work two ways: (1) we refine the disparities with a disconnected U-net style network, (2) we generalize the inner-product operation to include a metric tensor and use a gaussian-of-residuals approach to computing probabilities instead of the conventional softmax operation.

1. Introduction

Creating a real time model of one or more fields of view is critical to many applications such as self-driving cars and robotics. Solutions include sensors, which directly measure the distance over a field of view, such as LIDAR; however, these solutions are expensive while providing limited resolution. Given human, and in general animal, abilities to determine depth from stereo images separated by a distance, use of a pair of cameras is an intuitive and cost effective solution to provide higher resolution depth maps over a field of view.

Each image in a stereo pair views the same scene from a slightly different location: the separation between the two cameras is called the *baseline*. Both cameras are usually near-identical in construction and intrinsic calibration (focal length, aperture etc.). They are also extrinsically calibrated with rigid *baseline* separation and near-parallel viewing orientation. A *rectification* process corrects images for lens distortions and non-parallel viewing direction.

Once this is done, corresponding points in the left and right images appear displaced relative to each other and this per-pixel separation is called *disparity*. Some features can be occluded from the other camera's point-of-view so disparity information cannot always be computed for all pixels. Disparity is inversely proportional to depth, so a value of 0, which corresponds to infinite depth (uninteresting), is often repurposed to indicate occlusion. Computing depth information reduces to calculating disparities and our work does so using end-to-end Deep Learning methods.

The best known classical method prior to Deep Learning approaches uses a method called Semi-Global (Block) Matching [Hirschmuller] and is available in free implementations like OpenCV. Recently, there has been enormous interest in the use of Neural Networks for Disparity, Depth, Optical Flow computations and Semantic Segmentation.

This project attempts to apply network architectures developed for filtering / image classification to the problem of filtering disparity results from a patch-matching network. It also experiments with an original, alternate construction for the probabilities of disparities prior to the argmax.

2. Related Work

Deep Learning approaches to Stereo Matching fall into three categories [Zhou]. **Post-processing** based approaches use neural networks to compute feature vectors which are aggregated into a cost-volume. This is followed by non-deep-learning algorithms that "post-process" this information to arrive at disparity values. **End-to-end Deep Learning** methods seek to learn the post-processing transform. The cost-volume is constructed either using inner-products between left and right feature vectors, or from a concatenation of the two. Lastly, **Unsupervised Learning** methods rely on minimizing photometric warping error, as opposed to matching labeled disparity ground-truth.

Typically, end-to-end neural networks for stereo matching begin with a Siamese network to extract features of the image/patch pairs. The left and right feature tensors are then concatenated into a *cost volume* which is refined into a disparity field by subsequent layers which are either fully connected and/or stacked-U-net variants.

Seminal use of CNNs to extract per-pixel features was by [Zbontar]. It takes a square patch of pixels from the left image, in the range of 9×9 to 11×11 centered around the pixel of interest and calculates the cosine variance for each of the possible disparity locations in the corresponding right image, 201 possible disparities for the KITTI 2015 dataset [Menze]. It then filters this cost volume via semiglobal matching, argmin of the resulting volumes, followed by interpolation, sub-pixel enhancement, median filtering and bilateral filtering. While achieving very accurate results the post processing can take more than one minute.

[Mayer] first proposed an end-to-end approach, with pre-training using a synthetic dataset (SceneFlow) followed by fine-tuning (transfer) to a real-world dataset (KITTI 2012 [Geiger]). [Chang] propose Pyramidal Stereo Matching which makes use of Spatial Pyramid Pooling [He] inside the Siamese sub-network in order to better exploit multi-scale spatial features in the cost-volume construction.. In addition, they demonstrate a stacked-hourglass architecture for disparity-refinement on the cost volume. This stacked-hourglass sub-network resembles concatenated U-nets [Ronneberger] except that there are skip connections across the 'U' sub-networks. While very accurate, this is computationally expensive, requiring GPUs with huge amounts of memory (the paper mentions 16GB GPU RAM but our experiments on AWS required more - we had to use a p3.8xlarge instance with 64GB RAM (4 X V100 / SLI) costing $12/hr! [Luo] presents an idea to use image-patches, instead of entire images, to train and to significantly simplify the cost-volume calculation using a simple inner-product technique. The disparity is extracted using a Softmax over these inner-product values. This work forms the basis of our work.

3. Dataset

The KITTI Vision Stereo 2015 benchmark, [Menze], is the benchmark predominantly used in the field of binocular stereopsis in research. Relevant to this project the dataset includes matching sets of data, composed of left image (RGB), right image (RGB), left image per pixel non-occluded disparity (16-bit). An example of the left, right and ground truth left disparity images are shown in Figure 1.

Figure 2: Left, Right input images and Disparity (represented as grayscale)

The data set includes 200 training image pairs with corresponding ground truth disparities and additional 200 test image pairs without ground truth. Our work will split the training dataset into a 160 / 40 train / test split. Per pixel disparity provided as ground truth does not include every pixel in an image, but a subset of between 7% and 21% of the total pixels. A disparity ground truth image with a threshold applied to show all ground truth data points in white is shown in Figure 3.

Figure 3: Black and White Disparity Map with (19% populated)

The data set has been augmented with images generated from the dataset itself. Specifically, the disparity prediction output from the Siamese network incorporating 13×13 patches, disparity prediction output from the Siamese network incorporating 37×37 patches, a fully populated ground truth and a **ground truth mask layer** which provides a mask covering the region within 5 ground truth points, but excluding the larger regions without ground truth data. The fully populated ground truth is determined by a **filling filtering algorithm** which for every ground truth value of zero searches a growing square around the zero value until cumulatively at least four non-zero values have been found. The new value for this location is then calculated as the weighted average of all non-zero neighbors, where the weights are inversely proportional to the euclidean distance to the non-zero value.

4. Methods

The training methods and network architectures can be separated into the Siamese network (left hand side of Figure 1) used to determine the raw disparity values, the cost-volume construction and the subsequent filtering network which consumes the raw disparity values plus the left RGB input reference image.

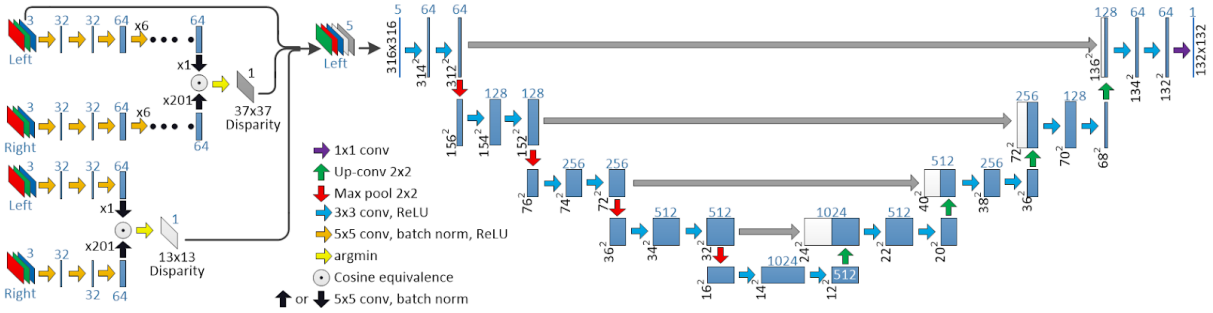


Figure 1: Proposed Network Architecture

4.1 Siamese Network Architecture

As shown in Figure 1, the Siamese Matching network architecture, consists of shared weights between the left and right network branches of a series of convolutional layers. The features of the left and right right channels are directly fed into the cost function. The kernel size in combination with the number of layers is designed to produce an output volume of 1×64 features on the left output. For a disparity range of 0 to d and a patch dimensions of $p \times p$, the right input is supplied with a patch size $n \times p$, where $n=p+d+1$, the resulting output is $(d+1) \times 64$ out where d is the disparity range.

4.2 Siamese Network Training

The training set of 160 left/right RGB image pairs plus disparity image are all cropped down to 370 x 1224 pixels. All non-zero points in the resulting disparity image are validated to determine if a centered bounding box defined by the image patch dimensions remains within the image space. The resulting valid locations and their corresponding image index value are shuffled to provide a list of training patches. For the 160 image set this results in roughly 11.6 million patches.

4.3 Cost-Volume and Loss Function

Two methods to construct the cost volume were attempted. One approach came from [Luo] where the loss function calculates, per-pixel, the inner-product between the left feature vector and each of the $d+1$ right feature vectors. A softmax operation over these results yields probabilities corresponding to the $d+1$ disparities. The loss is given by:

$$\mathbf{y} = softmax([\mathbf{l} \cdot \mathbf{r}_i, i \in [0, d])$$

$$L = -1 \sum_{i=0}^{d} p_{GT}(y_i) \log \hat{y}_i$$

$$p_{GT}(y_i) = 0.5 \text{ if } y_i = y_i^{GT},\ 0.2 \text{ if } \left| y_i - y_i^{GT} \right| = 1,\ 0.05 \text{ if } \left| y_i - y_i^{GT} \right| = 2,\ 0 \text{ otherwise}$$

The second approach was to try learning a metric tensor M for feature vector space (the previous approach assumes the space is Euclidean) and to use this to compute residuals between the left feature vector and the corresponding $d+1$ right feature vectors to find the best match. The scalar unnormalized Gaussian of this value would lie between 0 and 1 which can be interpreted as probabilities: residuals closer to zero (better match) should have higher probability.

$$\mathbf{y} = [exp(-||\mathbf{l} - \mathbf{r}_i||_M^2), i \in [0, d])]$$

We use the same loss function as above. The loss formula becomes especially simple (log of exponential ≡ identity).

4.4 Filtering Network Architecture

As shown on the right hand side of Figure 1, the U-net architecture, based on the work by [Ronneberger], includes a series of four "down conversion" segments, followed by four "up conversion" segments, followed by a final pair of 3×3 convolutions and a 1×1 convolution layer to generate a 2D feature map of 132×132 pixels. The architecture has been modified from the original Unet architecture in the final layer being reduced to a single channel and including the option for batch normalization following each convolutional layer, save the last layer.

4.5 Filtering Network Training

The filtering network training is performed by providing the network with a 5-channel input by stacking the left RGB image with the Siamese generated 13×13 and 37×37 patch disparities. The additional channels were added with the goal of providing context to the network with the expectation of improved performance. Optionally the network can be trained on only one dimension, the 37×37 patch disparity. The network architecture limits possible input dimensions to 188+16*n*, where *n* is a positive integer. To provide a significant training set size input volumes of 316×316 pixels were randomly selected from input coordinates fitting the volume inside the 370×1224 xy dimensions from across all 160 training image sets, providing roughly 7.8M distinct inputs.

5. Experiments, Results and Discussion

In this section, we present the obtained results training both networks. Our implementation can be found at:

https://github.com/manoj-rajagopalan/stanford-cs230-project

Given that our test set differs from the test sets used in our references, our results may deviate slightly from true scores it could receive on KITTI 2015. Randomizing the selection of data and test sets from KITTI 2015 training set have shown less than +/- 0.75% variance. Networks were trained on Tesla V100 GPUs or P100 GPUs each with a minimum of 16GB of storage, with training training cycle taking between 1.5 and 6 hours to complete. Implementations of both the Siamese Network and the Filtering Network were done either from scratch or ported from TensorFlow implementations which could not be run due to undocumented dependencies. Batch sizes were limited to fit within 16GB limitation; however, were optimal for smaller batch sizes. Datasets for both networks were large enough to train in a single epoch, multiple epochs did not materially change the performance. Results are compared against a KITTI benchmark metric of percentage of pixels with more than 3 pixels of disparity error from the ground truth, referred to as the "3+ pixel error".

5.1 Siamese Matching Network

Our Siamese Matching Network implementation's optimal performance, 8.06% 3+ pixel error, closely matched the performance reported by [Luo], 7.23% for 37×37 sized patches. [Zbontar] does not report on performance without post processing. Optimal performance for the 13×13 patches is 12.3% 3+ pixel error. The Siamese Matching Network's performance varied slightly over a batch sizes ranging from 32 to 384 and learning rates from 0.005 to 0.04. Both the referenced implementations reduce the learning rate by a factor of 5 at 24k iterations and another factor of 5 every 8k iterations after. In our implementation we had slightly better results when implementing a non-linear decreasing learning rate with learning_rate = 0.01 and a reduction_factor of 50, according to:

$$learning\ rate = initial\ learning\ rate \cdot \left(\frac{1}{(reduction\ factor)^{\frac{1}{number\ of\ steps}}} \right)^{step},$$

$$where\ the\ desired\ final\ learning\ rate\ is\ \frac{staring\ learning\ rate}{reduction\ factor}$$

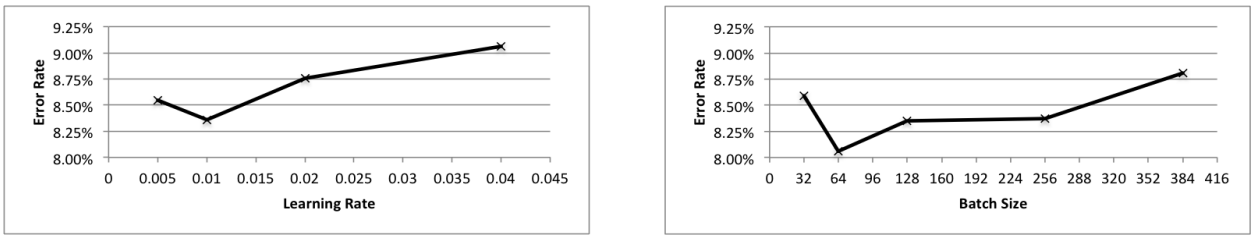


Figure 5: 37×37 Patch Learning Rate and Batch Hyperparameter Search Results

5.2 Cost Volume Methods

All results discussed before and after this section use the original softmax-of-inner-product formulation for cost-volume. We found the alternate cost volume formulation (exponential-of-residual-norm) to fail for various reasons:

- Learned metric was not symmetric positive definite, leading to large negative values for norms!
- Using ½ (M+M^T) instead of just M preserved symmetry but not diagonal dominance: negative eigenvalues caused negative values for the norm (undesirable)
- Implementing diagonal-dominance constraint (adding signed sum of absolute value of off-diagonal elements, to the diagonal) for the metric broke PyTorch's autograd tracing capabilities.
- Using a simple diagonal metric did give all positive values but they converged to zero over time.

5.3 Filtering Network

Our initial architecture on the filtering network borrowed the architecture from [Liu], a CNN based Unet with two down conversion and up conversions with skip connections at each level and one skip connection spanning the entire network, targeting 48×48 pixel patch sizes. All loss functions described below were swept over a range of learning rates. See **Appendix A** for a complete table of experiments. Initially the loss function was the MSE of the difference between the ground truth and the output; however, this did not converge and at this time we realized the sparse nature of the ground truth data. The loss function was modified to mask out the non-valid disparity locations from the MSE function and only providing 48×48 pixel patches with more than three disparity differences; however, no improvement was seen. In retrospect the implementation of the masking function may have prevented gradient propagation during training. The loss function was modified to represent the sum of pixels with 3+ pixel disparity error, which also demonstrated no improvement and provided zero gradient for back propagation.

Next the filtering architecture was changed to the Unet [Ronneberger] architecture with the addition of a skip connection from the input to the output. Tests with the 3+ pixel disparity error metric function also failed to converge. Using a MSE loss function also failed to converge. A full ground truth image was created by applying the **filling filtering algorithm** and masking with the **ground truth mask**. Pixels zeroed by the **ground truth mask** were replaced with 37×37 patch pixels, so the MSE for those would be zero. Initial runs did not converge, so the network was modified to operate on a single channel input of just the 37×37 patch layer. At this point the training loss began to converge, eventually arriving at an optimal point of 2.0% 3+pixel disparity error with a batch size of 4, an initial learning rate of 0.001 and a reduction factor of 50. Learning rates and batch sizes were varied around this point without additional improvement. The use of an L1Smooth loss function also converged but did not improve performance. Returning to the 5-channel input, the training loss converged around the hyperparameter point of batch size of 2, learning rate of 0.0001 and a reduction factor of 100; however, performance, at 2.6% 3+ pixel disparity error, was worse than the optimal single channel implementation. While the additional information available to the network from the additional channels might enable improved performance, the optimal operating point required a much lower learning rate, likely due to the difficulty in the optimizer determining a direction. The complete solution performed well when compared to the hand crafted filtering functions of [Zbontar] and [Luo] as shown in Table 1 below.

Solution	> 3 pixel disparity error (non-occluded)
Unary+CA+SGM+Post+Slant [Luo]	3.07%
Unary+SGM, I, SPE, MF, BLF [Zbontar]	2.43%
Ours (Single 37×37 channel, batch_size=4, lr=0.000	**2.0%**

Table 1: Solution Performance Comparison

6. Conclusion and Future Work

In this project we constructed a concatenated two network solution to accurately predict pixel resolution disparities between binocular stereo images and specifically applied a convolutional neural network architecture to learn a filtering implementation specific to the filtering of disparity values, producing results demonstrating improvement over hand crafted filters. The original multi-channel design, created a larger search domain for the network, resulting in poorer performance when compared to a single-channel design. Due to resource constraints the current implementation does not tile over the entire input image. For future development implementing radial padding and implementing tiling to support filtering of the complete image would be a natural next step. In addition closing the link between the two networks to attempt to optimize the complete solution would be a worthwhile experiment; however, it would require greater GPU resources than we have available at this time.

7. Contributions

Effort during the project development was regularly divided between team members, with each having responsibilities in literature search, proposal, environment development, dataset gathering, milestone delivery, coding, debug, brainstorming and data augmentation. Manoj took a greater responsibility during the AWS environment development and milestone delivery, which Gordon was recovering from knee surgery (out for two weeks) and Gordon took a greater responsibility during the testing phase. Due to the challenges faced in the development, with issues arriving in the adoption of various network code bases, six in total, both Manoj and Gordon brought up networks in both TensorFlow and Pytorch implementations. The effort was truly collaborative with regular check points and re-division of responsibilities as the project dictated.

8. References

J.Chang, Y.Chen, "Pyramid Stereo Matching Network", *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* (CVPR), 2018, pp. 5410-5418.

A.Geiger, P.Lenz, R.Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision BenchmarkSuite", *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* (CVPR) 2012.

KITTI 2015, http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo

K. He, X. Zhang, S. Ren, J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition", *IEEE transactions on pattern analysis and machine intelligence* 37.9 (2015): 1904-1916.

H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information", *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* (CVPR) 2005, pp. 807–814.

D. Liu, B. Wen, X. Liu, Z. Wang, and T. Huang. "When Image Denoising Meets High-Level Vision Tasks: A Deep Learning Approach. *International Joint Conferences on Artificial Intelligence Organization*, 2018.

W. Luo, A. G. Schwing and R. Urtasun, "Efficient Deep Learning for Stereo Matching," *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* (CVPR) 2016, Las Vegas, NV, 2016, pp. 5695-5703.

N.Mayer, E.Ilg, P.Häusser, D.Cremers, A.Dosovitsky, T.Brox, "A Large Dataset to Train ConvolutionalNetworks for Disparity, Optical Flow, and Scene Flow", Estimation, *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* (CVPR) 2016.

M.Menze, C.Heipke, A.Geiger, "Joint 3D Estimation of Vehicles and Scene Flow", *ISPRS Workshop onImage Sequence Analysis* (ISA), 2015.

O. Ronneberger, P. Fischer, and T. Brox. "U-net: Convolutional networks for biomedical image segmentation. *Intl Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

J. Zbontar, Y. LeCun, "Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches", *J. Machine Learning Research*, vol. 17, pp. 1-32, 2016.

K. Zhou, X. Meng, B. Cheng, "Review of Stereo Matching Algorithms Based on Deep Learning", *Computational Intelligence and Neuroscience*, vol. 2020, Article ID 8562323

Appendix A

Network	Learning Rate	Red Fact	Training Length	Batch Size	Patch Size Height	Patch Size Width	Batch Norm	Loss Function	Loss Behavior	Image Feed Forward (Around NN, into Loss)	Input Channels (R, G, B, 13, 37)	Ground Truth Filled / Sparse	Error Rate	Notes
FilterNet	0.001	25	1.00E+05	32	48	48	no	MSE	noisy	yes	R, G, B, 13, 37	Sparse	~8%	Loss initially reduces and then bounces around
FilterNet	**0.01**	25	1.00E+05	32	48	48	no	MSE	exploded	yes	R, G, B, 13, 37	Sparse	-	
FilterNet	0.0001	25	1.00E+05	32	48	48	no	MSE	noisy	yes	R, G, B, 13, 37	Sparse	~8%	Loss initially reduces and then bounces around
FilterNet	0.00001	25	1.00E+05	32	48	48	no	MSE	noisy	yes	R, G, B, 13, 37	Sparse	~8%	Loss initially reduces and then bounces around
FilterNet	0.0001	25	1.00E+05	32	48	48	yes	MSE	noisy	yes	R, G, B, 13, 37	Sparse	~8%	Loss initially reduces and then bounces around
FilterNet	0.0001	25	1.00E+05	16	48	48	yes	MSE	noisy	yes	R, G, B, 13, 37	Sparse	~8%	Loss initially reduces and then bounces around
FilterNet	0.0001	25	1.00E+05	4	48	48	yes	MSE	noisy	yes	R, G, B, 13, 37	Sparse	~8%	Loss initially reduces and then bounces around
FilterNet	**0.001**	25	1.00E+05	32	48	48	yes	**MSE with disparity mask**	exploded	yes	R, G, B, 13, 37	Sparse	~8%	Loss initially reduces and then bounces around
FilterNet	0.0001	25	1.00E+05	32	48	48	yes	MSE with disparity mask	noisy	yes	R, G, B, 13, 37	Sparse	~8%	Loss initially reduces and then bounces around
FilterNet	0.00001	25	1.00E+05	32	48	48	yes	MSE with disparity mask	noisy	yes	R, G, B, 13, 37	Sparse	~8%	Loss initially reduces and then bounces around
FilterNet	0.001	25	1.00E+05	32	48	48	yes	**disparity mask + count > 3 pixel error**	exploded	yes	R, G, B, 13, 37	Sparse	~8%	Loss did not materially change
FilterNet	0.0001	25	1.00E+05	32	48	48	yes	disparity mask + count > 3 pixel error	noisy	yes	R, G, B, 13, 37	Sparse	~8%	Loss did not materially change
FilterNet	0.00001	25	1.00E+05	32	48	48	yes	disparity mask + count > 3 pixel error	noisy	yes	R, G, B, 13, 37	Sparse	~8%	Loss did not materially change
Unet	0.001	25	1.00E+05	4	**316**	**316**	no	**disparity mask + count > 3 pixel error**	noisy	yes	R, G, B, 13, 37	Sparse	-	Loss did not materially change
Unet	0.0001	25	1.00E+05	4	316	316	no	disparity mask + count > 3 pixel error	noisy	yes	R, G, B, 13, 37	Sparse	-	Loss did not materially change
Unet	0.00001	25	1.00E+05	4	316	316	no	disparity mask + count > 3 pixel error	noisy	yes	R, G, B, 13, 37	Sparse	-	Loss did not materially change
Unet	0.001	25	1.00E+05	4	316	316	no	**MSE**	noisy reduction	yes	R, G, B, 13, 37	**Filled**	50%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	0.0001	25	1.00E+05	4	316	316	no	MSE	noisy reduction	yes	R, G, B, 13, 37	Filled	6%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	**0.00001**	25	**4.00E+05**	4	316	316	no	MSE	noisy reduction	yes	R, G, B, 13, 37	Filled	3.9%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	0.0001	25	1.00E+05	4	316	316	no	MSE	noisy reduction	**no**	R, G, B, 13, 37	Filled	4.2%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	**0.00001**	25	1.00E+05	4	316	316	no	MSE	noisy reduction	no	R, G, B, 13, 37	Filled	3.90%	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	0.0001	**50**	1.00E+05	4	316	316	no	MSE	noisy reduction	no	R, G, B, 13, 37	Filled	3.8%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	0.0001	**100**	**4.00E+05**	4	316	316	no	MSE	noisy reduction	no	R, G, B, 13, 37	Filled	3.3%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	**0.00001**	100	**8.00E+05**	**8**	316	316	no	MSE	noisy reduction	no	R, G, B, 13, 37	Filled	3.0%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	0.00001	100	8.00E+05	**4**	316	316	no	MSE	noisy reduction	no	R, G, B, 13, 37	Filled	2.6%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	0.00001	100	**4.00E+05**	**2**	316	316	no	MSE	noisy reduction	no	R, G, B, 13, 37	Filled	3.0%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	0.0001	25	1.00E+05	4	316	316	no	MSE	noisy reduction	no	**37**	Filled	2.6%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	0.0001	**50**	4.00E+05	4	316	316	no	MSE	noisy reduction	no	37	Filled	2.0%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	0.0001	50	4.00E+05	4	316	316	**yes**	MSE	noisy reduction	no	37	Filled	2.1%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	**0.00001**	**100**	**8.00E+05**	4	316	316	no	MSE	noisy reduction	no	37	Filled	2.5%*	Loss reduced by ~20x, error is only on a 132 x 132 patch
Unet	0.0001	25	4.00E+04	4	**316**	**1224**	no	MSE	noisy reduction	no	37	Filled	15%*	Loss reduced by ~20x, error is only on a 132 x 1028 patch
Unet	**0.00001**	**50**	**1.00E+05**	4	**316**	**1224**	no	MSE	noisy reduction	no	37	Filled	20%*	Loss reduced by ~20x, error is only on a 132 x 1028 patch
Unet	0.0001	25	1.00E+05	4	316	316	no	L1Smooth	noisy reduction	no	37	Filled	3.0%*	**Loss reduced by ~30x, error is only on a 132 x 132 patch**
Unet	**0.00001**	**100**	**8.00E+05**	4	316	316	no	L1Smooth	noisy reduction	no	37	Filled	2.4%*	**Loss reduced by ~30x, error is only on a 132 x 132 patch**