Single Image Depth Prediction with Wavelet Decomposition Supplementary Material

Michaël Ramamonjisoa[1,*] Michael Firman[2] Jamie Watson[2]
Vincent Lepetit[1] Daniyar Turmukhambetov[2]
[1]LIGM, IMAGINE, Ecole des Ponts, Univ Gustave Eiffel, CNRS [2]Niantic
www.github.com/nianticlabs/wavelet-monodepth

1. Network Architectures and Losses

1.1. On direct supervision of wavelet coefficients

The previous work WaveletStereo [22] supervises its wavelet based stereo matching method with ground truth wavelet coefficients at the different levels of the decomposition. However, wavelets can only reliably be supervised when ground truth depth -or disparity- is provided and when it does not contain missing values or high-frequency noise, as they show on the synthetic SceneFlow [14] dataset. The sparsity of ground truth data in the KITTI dataset especially around edges makes it impossible to estimate reliably ground truth wavelet coefficients. On NYUv2, the noise in depth maps is also an issue for direct supervision of wavelets, e.g. with creases in the layout or inaccurate depth edges. This noise also prohibits the use of Semi Global Matching ground truth for wavelet coefficient supervision.

As we show in our work, supervising the network on wavelet reconstructions allows us to ignore missing values and be robust to noisy labels.

1.2. Experiments on KITTI

Architecture The architecture we use for our experiments is a modification of the architecture used in [7], as described in the main paper. In Table 1, we set out our decoder architecture in detail.

Self-supervised losses Our self-supervised losses are as described in [7], which we repeat here for completeness. Given a stereo pair of images (I_L, I_R), we train our network to predict a depth map D_L, pixel-aligned with the left image. We also assume access to the camera intrinsics K, and the relative camera transformation between the images in the stereo pair $T_{R\to L}$. We use the network's current estimate of depth to synthesise an image $I_{R\to L}$, computed as

$$I_{R\to L} = I_R\Big\langle \text{proj}(D_L, T_{R\to L}, K)\Big\rangle, \tag{1}$$

where proj() are the 2D pixel coordinates obtained by projecting the depths D_L into image I_R, and $\langle\rangle$ is the sampling operator. We follow standard practice in training the model under a photometric reconstruction error pe, so our loss becomes

$$L_p = pe(I_L, I_{R\to L}). \tag{2}$$

Following [7, 3] etc. we use a weighted sum of SSIM and L1 losses

$$pe(I_a, I_b) = \alpha\frac{1-\text{SSIM}(I_a, I_b)}{2} + (1-\alpha)\|I_a - I_b\|_1,$$

where $\alpha = 0.85$. We additionally follow [7] in using the smoothness loss:

$$L_s = |\partial_x d_L^*|\, e^{-|\partial_x I_L|} + |\partial_y d_L^*|\, e^{-|\partial_y I_L|}, \tag{3}$$

where $d_L^* = d_L/\overline{d_L}$ is the mean-normalized inverse depth for image I_L.

When we train on monocular and stereo sequences ('MS'), we again follow [7] — see our main paper for an overview, and [7] for full details.

Depth Hints loss When we train with depth hints, we use the proxy loss from [20], which we recap here. For stereo training pairs, we compute a proxy depth map $\tilde{D}_L$ using semi-global matching [10], an off-the-shelf stereo matching algorithm. We use this to create a second synthesized image

$$\tilde{I}_{R\to L} = I_R\Big\langle \text{proj}(\tilde{D}_L, T_{R\to L}, K)\Big\rangle, \tag{4}$$

We decide whether or not to apply a supervised loss using $\tilde{D}_L$ as ground truth on a *per-pixel* basis. We only add this supervised loss for pixels where $pe(I_L, \tilde{I}_{R\to L})$ is lower $pe(I_L, I_{R\to L})$. The supervised loss term we use is $\log L_1$, following [20]. For experiments where Depth Hints are used for training, we disable the smoothness loss term.

*Work done during an internship at Niantic

Depth Decoder						
layer	**k**	**s**	**chns**	**res**	**input**	**activation**
upconv5	3	1	256	32	econv5	ELU [4]
Level 3 coefficients predictions						
iconv4	3	1	256	16	↑upconv5, econv4	ELU
disp4	3	1	1	16	iconv4	Sigmoid
wave4	3	1	3	16	iconv4	Sigmoid
upconv4	3	1	128	16	iconv4	ELU
IDWT$_3$	-	-	1	8	**disp4**, **wave4**	-
Level 2 coefficients predictions						
iconv3	3	1	128	8	↑upconv4, econv3	ELU
wave3	3	1	3	8	iconv3	Sigmoid
upconv3	3	1	64	8	iconv3	ELU
IDWT$_2$	-	-	1	8	**IDWT$_3$**, **wave3**	-
Level 1 coefficients predictions						
iconv2	3	1	64	4	↑upconv3, econv2	ELU
wave2	3	1	3	4	iconv2	Sigmoid
upconv2	3	1	32	4	iconv2	ELU
IDWT$_1$	-	-	1	8	**IDWT$_2$**, **wave2**	-
Level 0 coefficients predictions						
iconv1	3	1	32	2	↑upconv2, econv1	ELU
wave1	3	1	3	2	iconv1	Sigmoid
IDWT$_0$	-	-	-	1	**IDWT$_1$**, **wave1**	-

Table 1: Our decoder network architecture for experiments on the KITTI [5] dataset using ResNet backbone Here **k** is the kernel size, **s** the stride, **chns** the number of output channels for each layer, **res** is the downscaling factor for each layer relative to the input image, and **input** corresponds to the input of each layer where ↑ is a 2× nearest-neighbor upsampling of the layer. **disp4** is used produce the low-resolution estimate LL_3, while **waveJ** is used to decode $\{\mathsf{LH_J}, \mathsf{HL_J}, \mathsf{HH_J}\}$ at level **J**. **disp4** and **waveJ** are convolution blocks detailed in Table 2.

disp4 Layer						
layer	**k**	**s**	**chns**	**res**	**input**	**activation**
disp4(1)	1	1	**chns**(iconv5) / 4	16	iconv5	LeakyReLU(0.1) [21]
disp4(2)	3	1	1	16	disp4-1	Sigmoid

Wavelet Decoding Layer - waveJ						
layer	**k**	**s**	**chns**	**res**	**input**	**activation**
wave**J**(1+)	1	1	**chns**(iconv[**J**+1])	2^J	iconv[**J**+1]	LeakyReLU(0.1)
wave**J**(2+)	3	1	3	2^J	wave**J**(1+)	Sigmoid
wave**J**(1-)	1	1	**chns**(iconv[**J**+1])	2^J	iconv[**J**+1]	LeakyReLU(0.1)
wave**J**(2-)	3	1	3	2^J	wave**J**(1-)	Sigmoid
substract	1	1	3	2^J	wave**J**(2+), wave**J**(2-)	Linear

Table 2: Architecture of our wavelet decoding layer used for KITTI experiments J denotes the level of the decoder. **disp4** is used produce the low-resolution estimate LL_3, while **waveJ** is used to decode $\{\mathsf{LH_J}, \mathsf{HL_J}, \mathsf{HH_J}\}$.

Additional experiments We additionally tried training using edge-aware sparsity constraints that penalize non-zero coefficients at non-edge regions, by replacing depth gradients with wavelets coefficients in Monodepth's [6] *dis-*

Depth Decoder						
layer	**k**	**s**	**chns**	**res**	**input**	**activation**
upconv5	3	1	1104	32	econv5	Linear
iconv4	3	1	552	16	↑upconv5, econv4	LeakyReLU(0.2)
iconv3	3	1	276	8	↑iconv4, econv3	LeakyReLU(0.2)
iconv2	3	1	138	4	↑iconv3, econv2	LeakyReLU(0.2)
iconv1	3	1	69	2	↑iconv2, econv1	LeakyReLU(0.2)
outconv0	1	1	1	2	iconv1	Linear

Table 3: Architecture of our DenseNet baseline decoder for experiments on the NYUv2 [18] dataset Note that as in DenseDepth [2] we produce a depth map at half-resolution. Table adapted from [7].

parity smoothness loss, which unfortunately made training unstable. We also tried to supervise wavelet coefficients using distillation [9, 1] from a teacher depth network, which resulted in lower performances.

1.3. Experiments on NYUv2

Architecture We adapted our architecture from the PyTorch implementation of DenseDepth [2]. Our implementation uses a DenseNet161 encoder instead of a DenseNet169, and a standard decoder with up-convolutions. We first design a baseline that does not use wavelets, using the architecture detailed in Table 3. Our wavelet adaptation of that baseline is then detailed in Table 4. For experiments reported in the main paper, we follow the DenseDepth strategy and predict outputs at half the input resolution. Hence, the last level of the depth decoder in Table 4 is discarded. For experiments using a light-weight decoder discussed later in Section 4.4, which predicts 224×224 depth maps given a 224×224 input image, we keep all four levels of wavelet decomposition.

Supervised losses For our NYU results in the main paper, we supervise depth using an L1 loss and SSIM:

$$L_D(y, y^*) = \lambda_1 \ell_1(y, y^*), \tag{5}$$

where y and y^* are respectively predicted and ground truth depth and $\lambda_1 = 0.1$. Similar to [16, 15], we clamp depth between 0.4 and 10 meters.

2. Scores on Improved KITTI Ground Truth

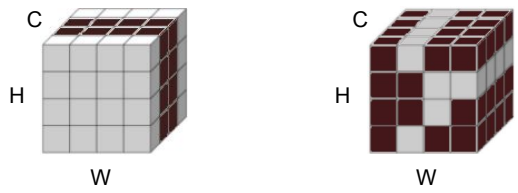
We report results on the improved KITTI ground truth [19] in Table 5. As we saw in the main paper, our method is competitive on scores with non-wavelets baselines, but as we have shown our wavelet decomposition enables more efficient predictions.

3. Qualitative Results

In this section, we show qualitative results of our method.

Depth Decoder						
layer	**k**	**s**	**chns**	**res**	**input**	**activation**
upconv5	3	1	1104	32	econv5	Linear
Level 3 coefficients predictions						
iconv4	3	1	552	16	↑upconv5, econv4	LeakyReLU(0.2)
disp4	1	1	1	16	upconv5	Linear
wave4	3	1	3	16	upconv5	Linear
IDWT$_3$	-	-	1	8	disp4, wave4	-
Level 2 coefficients predictions						
iconv3	3	1	276	8	↑iconv4, econv3	LeakyReLU(0.2)
wave3	3	1	3	8	iconv3	Linear
IDWT$_2$	-	-	1	4	**IDWT**$_3$, wave3	-
Level 1 coefficients predictions						
iconv2	3	1	138	4	↑iconv2, econv2	LeakyReLU(0.2)
wave2	3	1	3	4	iconv2	Linear
IDWT$_1$	-	-	1	2	**IDWT**$_2$, wave2	-

Table 4: Our decoder network architecture for experiments on the NYUv2 [18] dataset Note that since like in DenseDepth [2] we produce a depth map at half-resolution, we only need to predict wavelet coefficients until quarter-resolution. Table adapted from [7].



(a) **(b)**

Figure 1: Channel pruning (a) vs our sparse computation (b). Our sparse computation enabled by wavelets is complimentary with the channel pruning strategy to reduce the amount of computation, as both computation reduction methods operate in orthogonal dimensions.

In Figures 2-3-4 and Figures 5-6-7 we first show our sparse prediction process with corresponding sparse wavelets and masks, on the NYUv2 and KITTI datasets respectively. While we only need to compute wavelet coefficients in less than 10% of pixel locations in the decoding process, we show that our wavelets efficiently retain relevant information. Furthermore, we show that wavelets efficiently detect depth edges and their orientation. Therefore, future work could make efficient use of our wavelet based depth estimation method for occlusion boundary detection.

In Figure 8, we show comparative results between our baseline Depth Hints [20] and our wavelet based method.

4. Exploring Other Efficiency Tracks

Our paper mainly explores computation reduction in the decoder of a UNet-like architecture. However, this direction is orthogonal and complementary with all other complexity reduction lines of research.

Our approach is for example complementary with the FastDepth approach, which consists in reducing the overall complexity of a depth estimation network by compressing it in many dimensions such as (1) the encoder, (2) the decoder (3) the input resolution. They argue that the deep network introduced by Laina *et al.* [13] suffers from high complexity, while it could largely be reduced. Here we present a set of experiments we conducted to explore these different aspects of complexity reduction.

4.1. Experiment with a light-weight MobileNetv2 encoder

First, we replace the costly ResNet [8] or DenseNet [12, 11] backbone encoders with the efficient MobileNetv2 [17]. Indeed, in contrast with FastDepth, in the main paper, we report results using large encoder models (Resnet18/50 or Densenet161). Although this helps achieving better scores, we show in Table 6 and Table 7 that we can reach close to state-of-the-art results even with a small encoder such as MobileNetv2.

4.2. Separable convolutions

Secondly, FastDepth also shows that separable convolutions in their "NNConv" decoder provides the best score-efficiency trade-off. Since this approach is orthogonal to our sparsification method, it therefore complements our method and can be used to improve efficiency. Interestingly, we show in Table 7 that replacing sparse convolutions with sparse-depthwise separable convolutions works on par with standard convolutions. This can be explained by the fact that **IDWT** is also a separable operation, and therefore efficiently combines with depthwise separable convolutions.

4.3. Channel pruning

A popular approach to complexity and memory footprint reduction is channel pruning, which aims at removing some of the unnecessary channel in convolutional layers. Note that our wavelet enabled sparse convolutions are complementary with channel pruning, as can be seen in Figure 1. While channel pruning can, in practice, greatly reduce both complexity and memory footprint, it requires heavy hyper-parameter search that we therefore choose to leave for future work.

4.4. Input resolution

Finally, one important factor that makes FastDepth efficient is that it is trained with 224×224 inputs, against our 640×480 input. While our method is best designed for higher-resolution regime where sparsity of wavelets is stronger, we still show that our method achieves decent results even at low-resolution, and report our scores in Table 8.

Cit.	Method	PP	Data	H × W	Abs Rel	Sq Rel	RMSE	RMSE log	δ<1.25	δ<1.25^2	δ<1.25^3
[7]	Monodepth2 Resnet18	✓	S	192 × 640	0.079	0.512	3.721	0.131	0.924	0.982	0.994
	WaveletMonodepth Resnet18	✓	S	192 × 640	0.084	0.523	3.807	0.137	0.914	0.980	0.994
	WaveletMonodepth Resnet50	✓	S	192 × 640	0.081	0.477	3.658	0.133	0.920	0.981	0.994
[20]	Depth Hints	✓	S_{SGM}	192 × 640	0.085	0.487	3.670	0.131	0.917	0.983	0.996
	WaveletMonodepth Resnet18	✓	S_{SGM}	192 × 640	0.083	0.476	3.635	0.129	0.920	0.983	0.995
	Depth Hints Resnet50	✓	S_{SGM}	192 × 640	0.081	0.432	3.510	0.124	0.924	0.985	0.996
	WaveletMonodepth Resnet50	✓	S_{SGM}	192 × 640	0.081	0.449	3.509	0.125	0.923	0.986	0.996
[7]	Monodepth2 Resnet18	✓	MS	192 × 640	0.084	0.494	3.739	0.132	0.918	0.983	0.995
	WaveletMonodepth Resnet18	✓	MS	192 × 640	0.085	0.497	3.804	0.134	0.912	0.982	0.995
[20]	Depth Hints	✓	MS + S_{SGM}	192 × 640	0.087	0.526	3.776	0.133	0.915	0.982	0.995
	WaveletMonodepth Resnet18	✓	MS + S_{SGM}	192 × 640	0.086	0.497	3.699	0.131	0.914	0.983	0.996
[7]	Monodepth2 Resnet18	✓	S	320 × 1024	0.082	0.497	3.637	0.132	0.924	0.982	0.994
	WaveletMonodepth Resnet18	✓	S	320 × 1024	0.080	0.443	3.544	0.130	0.919	0.983	0.995
	WaveletMonodepth Resnet50	✓	S	320 × 1024	0.076	0.413	3.434	0.126	0.926	0.984	0.995
[20]	Depth Hints	✓	S_{SGM}	320 × 1024	0.077	0.404	3.345	0.119	0.930	0.988	0.997
	WaveletMonodepth Resnet18	✓	S_{SGM}	320 × 1024	0.078	0.397	3.316	0.121	0.928	0.987	0.997
	Depth Hints Resnet50	✓	S_{SGM}	320 × 1024	**0.074**	0.363	3.198	**0.114**	**0.936**	**0.989**	**0.997**
	WaveletMonodepth Resnet50	✓	S_{SGM}	320 × 1024	**0.074**	**0.357**	**3.170**	**0.114**	**0.936**	**0.989**	**0.997**

Table 5: Quantitative results on the improved KITTI benchmark. We compare our method to our baselines on the KITTI [5] improved dataset introduced by [19], using the Eigen split. *Data column* (data source used for training): S is for self-supervised training on stereo images, MS is for models trained with both M (forward and backward frames) and S data and S_{SGM} refers to the extra stereo ground truth which was used in [20].

Cit.	Method	PP	Data	H × W	Abs Rel	Sq Rel	RMSE	RMSE log	δ<1.25	δ<1.25^2	δ<1.25^3
[20]	Depth Hints	✓	S_{SGM}	192 × 640	0.106	0.780	4.695	0.193	0.875	0.958	0.980
	WaveletMonodepth MobileNetv2	✓	S_{SGM}	192 × 640	0.109	0.851	4.754	0.194	0.870	0.957	0.980
	WaveletMonodepth Resnet18	✓	S_{SGM}	192 × 640	0.107	0.829	4.693	0.193	0.873	0.957	0.980
	WaveletMonodepth Resnet50	✓	S_{SGM}	192 × 640	0.105	0.813	4.625	0.191	0.879	0.959	0.981
[20]	Depth Hints	✓	S_{SGM}	320 × 1024	0.099	0.723	4.445	0.187	0.886	0.961	0.982
	WaveletMonodepth MobileNetv2	✓	S_{SGM}	320 × 1024	0.104	0.772	4.545	0.188	0.880	0.960	0.982
	WaveletMonodepth Resnet18	✓	S_{SGM}	320 × 1024	0.102	0.739	4.452	0.188	0.883	0.960	0.981
	Depth Hints Resnet50	✓	S_{SGM}	320 × 1024	**0.096**	**0.710**	4.393	0.185	0.890	**0.962**	0.981
	WaveletMonodepth Resnet50	✓	S_{SGM}	320 × 1024	0.097	0.718	**4.387**	**0.184**	**0.891**	**0.962**	**0.982**

Table 6: Quantitative results on the KITTI dataset using MobileNetv2 encoder. We evaluate results of our method using a lighter encoder on KITTI [5], using the Eigen split. *Data column* (data source used for training): S is for self-supervised training on stereo images, MS is for models trained with both M (forward and backward frames) and S data and S_{SGM} refers to the extra stereo ground truth which was used in [20].

Method	Encoder	Depthwise	H × W	Abs Rel	RMSE	log_{10}	δ_1	δ_2	δ_3	ϵ_{acc}	ϵ_{comp}
Dense baseline	DenseNet161	-	480 × 640	0.1277	**0.5479**	**0.0539**	0.8430	**0.9681**	**0.9917**	**1.7170**	**7.0638**
Ours	DenseNet161	-	480 × 640	**0.1258**	0.5515	0.0542	**0.8451**	**0.9681**	**0.9917**	1.8070	7.1073
Ours	DenseNet161	✓	480 × 640	0.1275	0.5771	0.0557	0.8364	0.9635	0.9897	2.0133	7.1903
Dense baseline	MobileNetv2	-	480 × 640	0.1772	0.6638	0.0731	0.7419	0.9341	0.9835	1.8911	7.7960
Ours	MobileNetv2	-	480 × 640	0.1727	0.6776	0.0732	0.7380	0.9362	0.9844	1.9732	7.9004
Ours	MobileNetv2	✓	480 × 640	0.1734	0.6700	0.0731	0.7391	0.9347	0.9844	2.3036	8.0538

Table 7: Quantitative results on NYUv2 [18] using depth-wise convolutions and light-weight encoder We show that our method is compatible with other efficiency seeking approaches such as depth-wise separable convolutions and lower complexity encoders.

Method	Encoder	Depthwise	H × W	Abs Rel	RMSE	log_{10}	δ_1	δ_2	δ_3
Dense baseline	DenseNet161	-	224 × 224	0.1278	0.5715	0.0557	0.8368	0.9620	0.9901
Ours	DenseNet161	-	224 × 224	0.1279	0.5651	0.0549	0.8399	0.9652	0.9899
Ours	DenseNet161	✓	224 × 224	0.1304	0.5775	0.0564	0.8329	0.9613	0.9892
Dense baseline	MobileNetv2	-	224 × 224	0.1505	0.6221	0.0632	0.7984	0.9526	0.9878
Ours	MobileNetv2	-	224 × 224	0.1530	0.6409	0.0655	0.7844	0.9500	0.9864
Ours	MobileNetv2	✓	224 × 224	0.1491	0.6463	0.0646	0.7880	0.9506	0.9871

Table 8: Quantitative results on NYUv2 [18] using depth-wise convolutions and light-weight encoder We show that our method is compatible with other efficiency seeking approaches such as depth-wise separable convolutions and lower complexity encoders.

References

[1] Filippo Aleotti, Giulio Zaccaroni, Luca Bartolomei, Matteo Poggi, Fabio Tosi, and Stefano Mattoccia. Real-time single image depth perception in the wild with handheld devices. *Sensors*, 2021.

[2] Ibraheem Alhashim and Peter Wonka. High Quality Monocular Depth Estimation via Transfer Learning. *arXiv:1812.11941*, 2018.

[3] Yuhua Chen, Cordelia Schmid, and Cristian Sminchisescu. Self-supervised learning with geometric constraints in monocular video: Connecting flow, depth, and camera. In *ICCV*, 2019.

[4] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[5] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.

[6] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised Monocular Depth Estimation with Left-Right Consistency. In *CVPR*, 2017.

[7] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow. Digging into self-supervised monocular depth estimation. In *ICCV*, 2019.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.

[10] Heiko Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *CVPR*, 2005.

[11] Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens Van Der Maaten, and Kilian Weinberger. Convolutional networks with dense connectivity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[12] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[13] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper Depth Prediction with Fully Convolutional Residual Networks. In *3DV*, 2016.

[14] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4040–4048, 2016.

[15] Michael Ramamonjisoa, Yuming Du, and Vincent Lepetit. Predicting sharp and accurate occlusion boundaries in monocular depth estimation using displacement fields. In *CVPR*, 2020.

[16] Michael Ramamonjisoa and Vincent Lepetit. SharpNet: Fast and Accurate Recovery of Occluding Contours in Monocular Depth Estimation. In *ICCV Workshop*, 2019.

[17] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[18] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor Segmentation and Support Inference from RGBD Images. In *ECCV*, 2012.

[19] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant CNNs. In *3DV*, 2017.

[20] Jamie Watson, Michael Firman, Gabriel J. Brostow, and Daniyar Turmukhambetov. Self-supervised monocular depth hints. In *ICCV*, 2019.

[21] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[22] Menglong Yang, Fangrui Wu, and Wei Li. Waveletstereo: Learning wavelet coefficients of disparity map in stereo matching. In *CVPR*, June 2020.

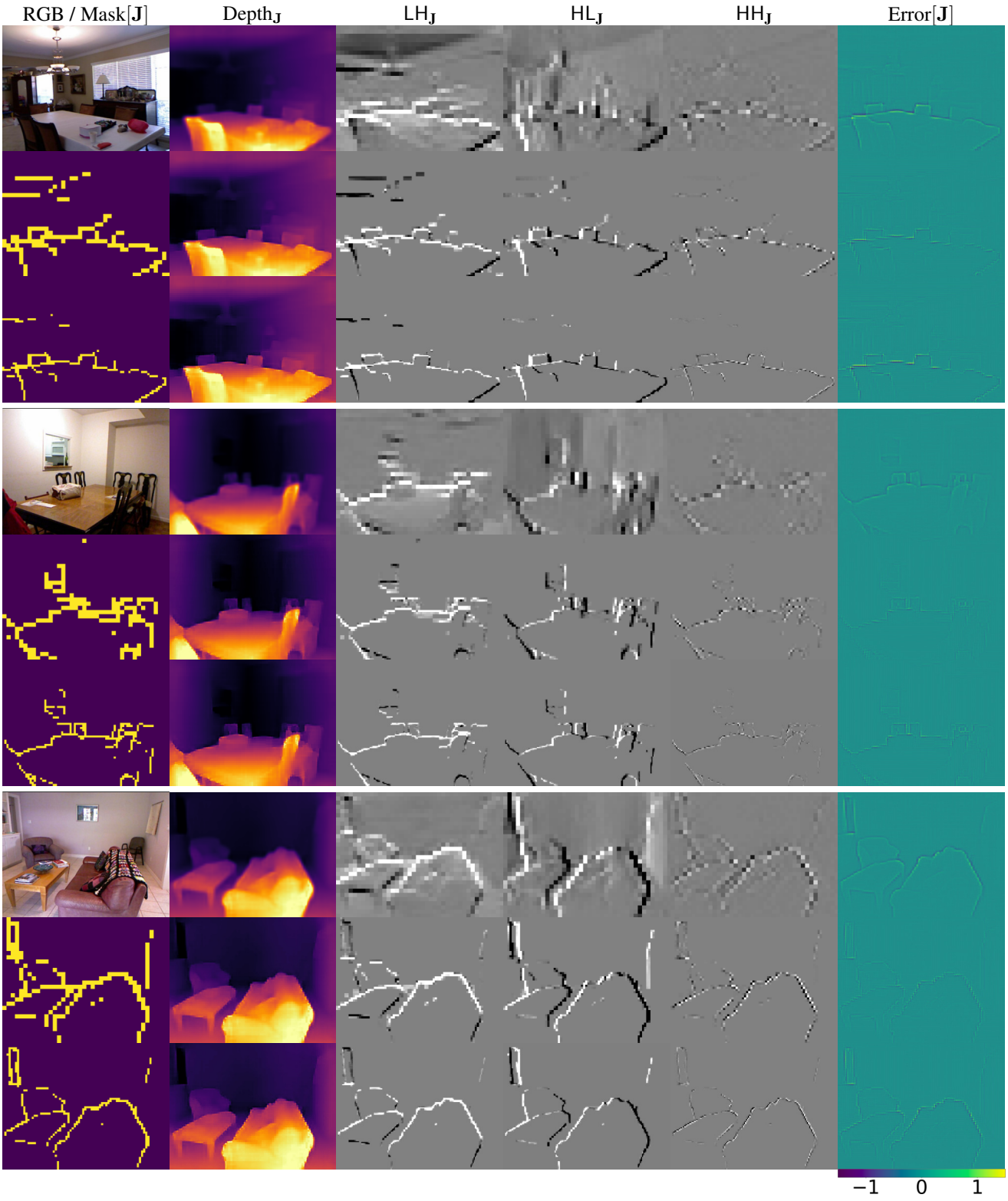


Figure 2: Qualitative results of predicted wavelets coefficients of depth maps on the NYU dataset (1/3). Our predicted wavelets have two desirable properties: they are sparse, allowing for efficient computation, and they are accurately located around depth edges without needing to supervise them. Results are obtained with $\eta = 0.04$. For each block of results, each row shows coefficients and depth maps obtained at scale **J** in the decoder from lowest to highest scale (decreasing **J**), as well as the (signed-)error between $\text{Depth}_{\mathbf{J}}$ and the Depth map obtained with dense wavelet coefficients at all scales. Error is displayed within range $[-1.5m, 1.5m]$.

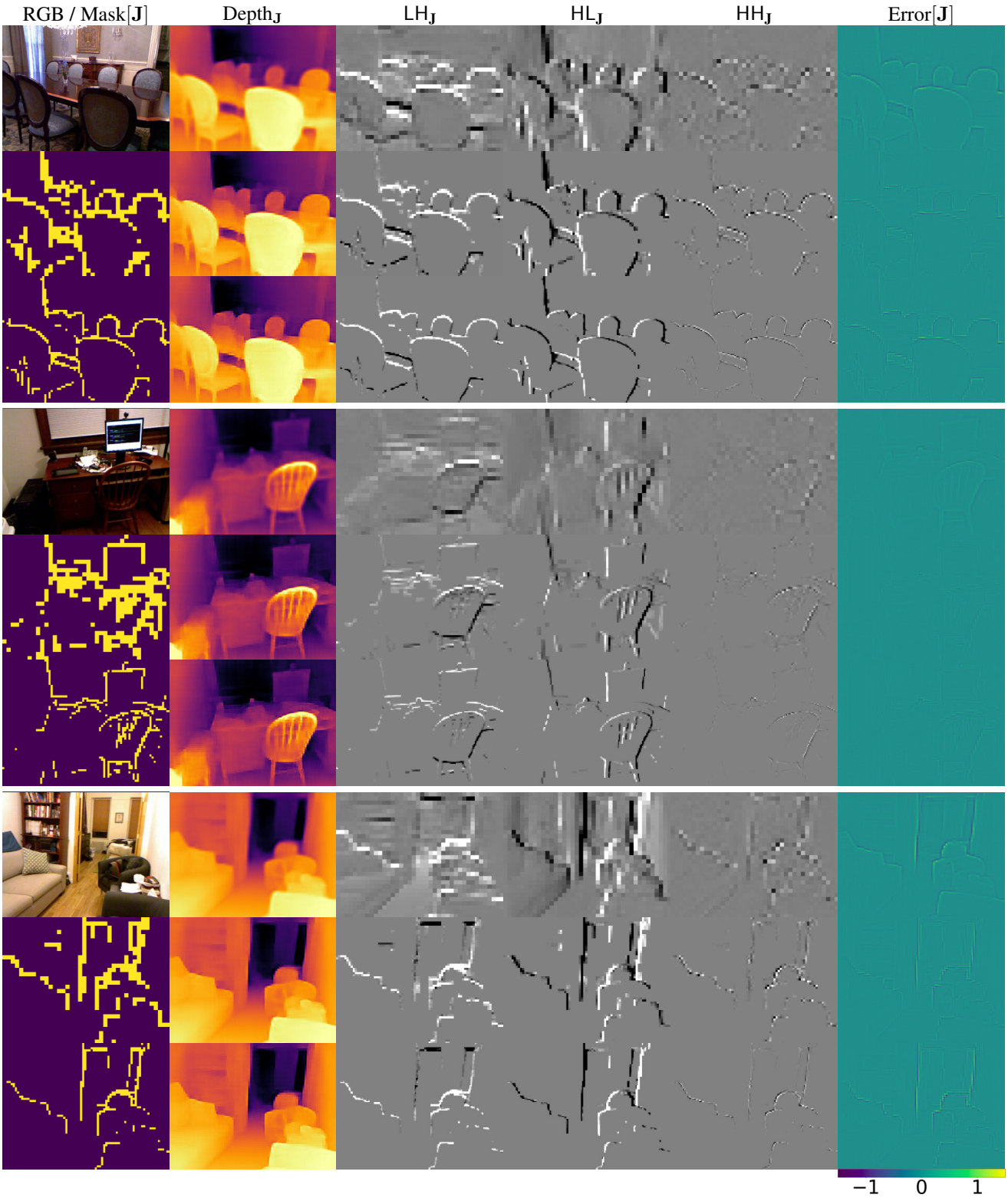


Figure 3: Qualitative results of predicted wavelets coefficients of depth maps on the NYU dataset (2/3). Our predicted wavelets have two desirable properties: they are sparse, allowing for efficient computation, and they are accurately located around depth edges without needing to supervise them. Results are obtained with $\eta = 0.04$. For each block of results, each row shows coefficients and depth maps obtained at scale $\mathbf{J}$ in the decoder from lowest to highest scale (decreasing $\mathbf{J}$), as well as the (signed-)error between $\text{Depth}_\mathbf{J}$ and the Depth map obtained with dense wavelet coefficients at all scales. Error is displayed within range $[-1.5m, 1.5m]$.

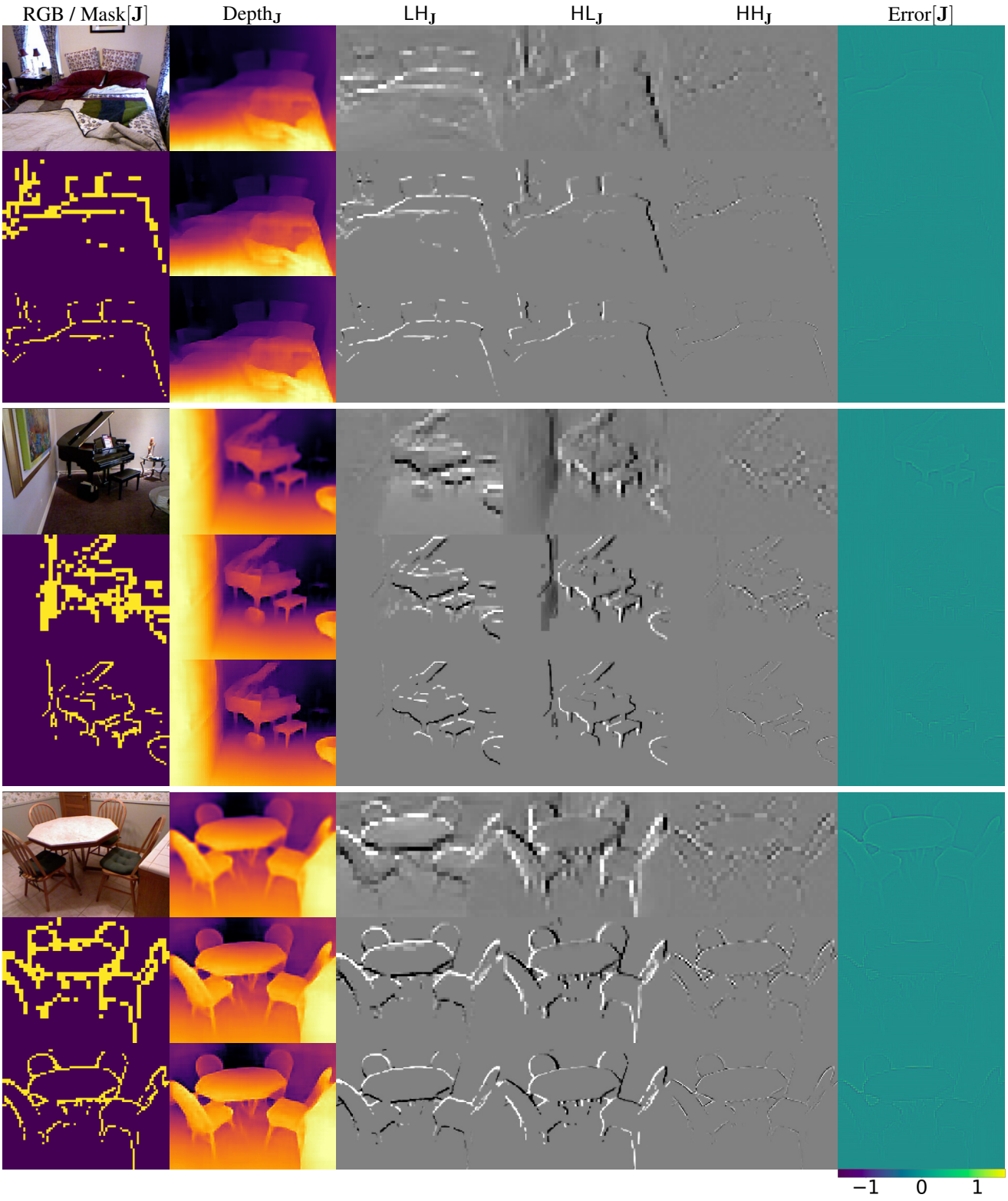


Figure 4: Qualitative results of predicted wavelets coefficients of depth maps on the NYU dataset (3/3). Our predicted wavelets have two desirable properties: they are sparse, allowing for efficient computation, and they are accurately located around depth edges without needing to supervise them. Results are obtained with $\eta = 0.04$. For each block of results, each row shows coefficients and depth maps obtained at scale **J** in the decoder from lowest to highest scale (decreasing **J**), as well as the (signed-)error between $\text{Depth}_{\mathbf{J}}$ and the Depth map obtained with dense wavelet coefficients at all scales. Error is displayed within range $[-1.5m, 1.5m]$.

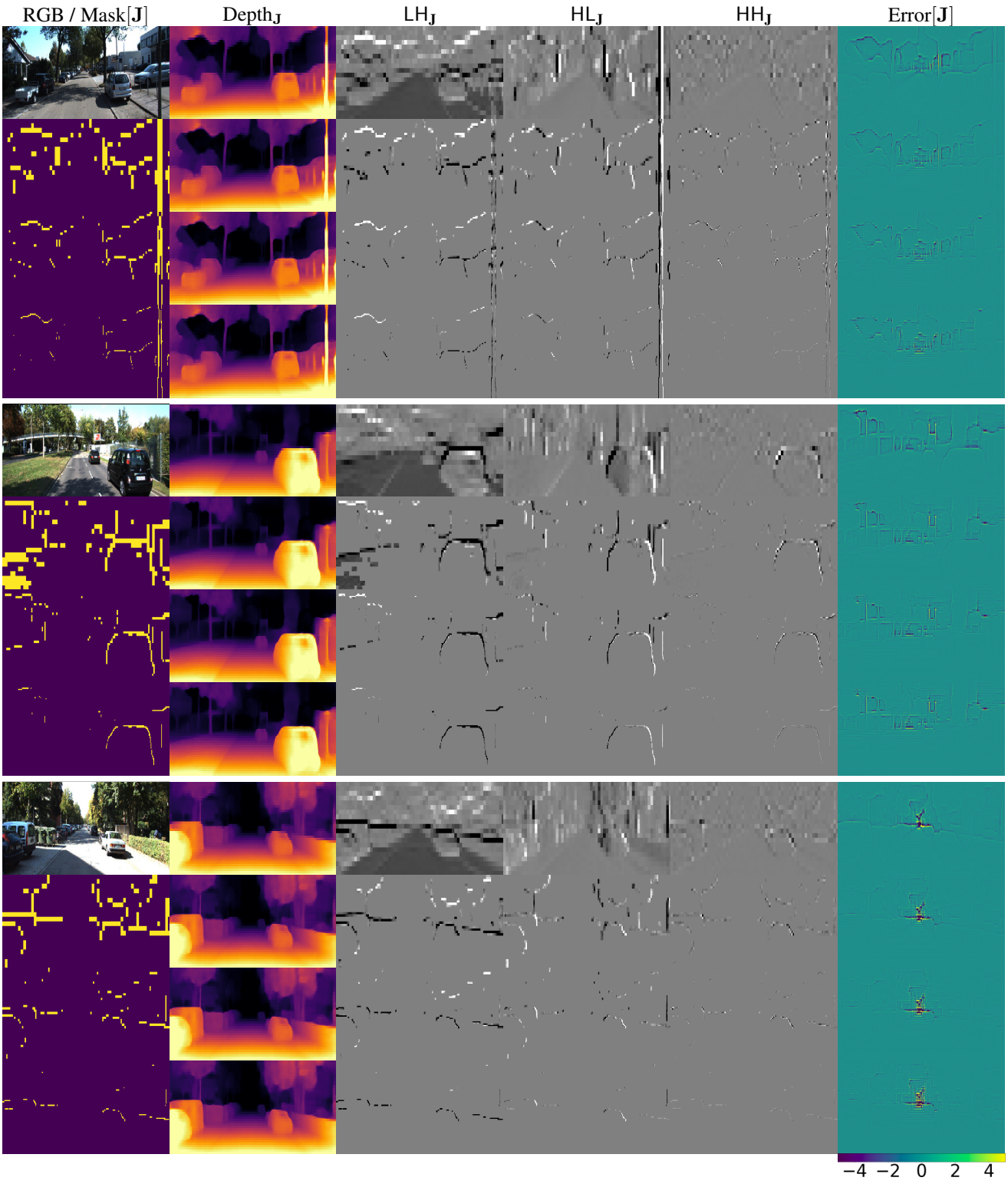


Figure 5: Qualitative results of predicted wavelets coefficients of depth maps on the KITTI dataset (1/3). Our predicted wavelets have two desirable properties: they are sparse, allowing for efficient computation, and they are accurately located around depth edges without needing to supervise them. Results are obtained with $\eta = 0.05$. For each block of results, each row shows coefficients and depth maps obtained at scale **J** in the decoder from lowest to highest scale (decreasing **J**), as well as the (signed-)error between $\text{Depth}_{\mathbf{J}}$ and the Depth map obtained with dense wavelet coefficients at all scales. Error is displayed within range $[-5m, 5m]$.

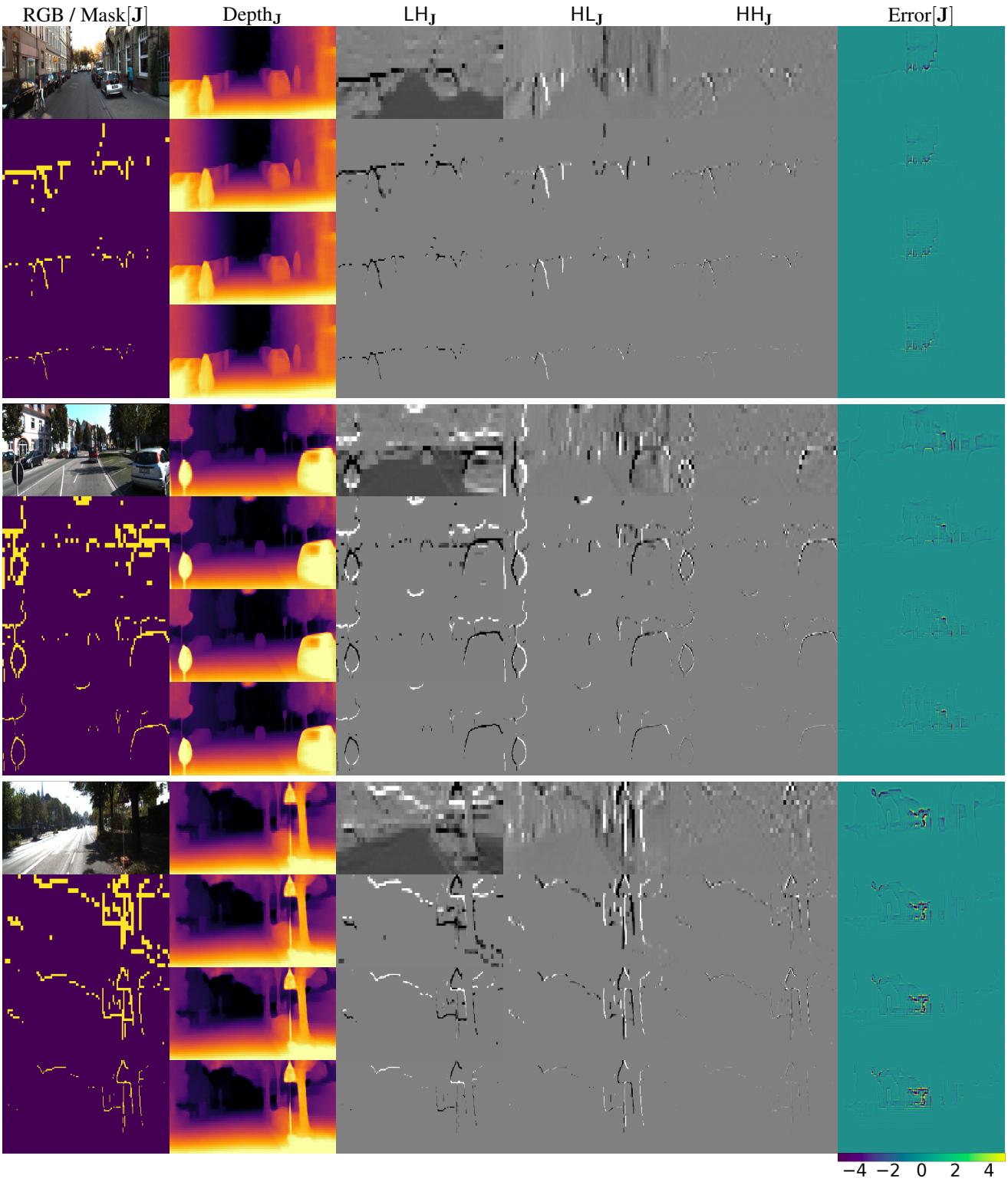


Figure 6: Qualitative results of predicted wavelets coefficients of depth maps on the KITTI dataset (2/3). Our predicted wavelets have two desirable properties: they are sparse, allowing for efficient computation, and they are accurately located around depth edges without needing to supervise them. Results are obtained with $\eta = 0.05$. For each block of results, each row shows coefficients and depth maps obtained at scale $\mathbf{J}$ in the decoder from lowest to highest scale (decreasing $\mathbf{J}$), as well as the (signed-)error between $\text{Depth}_{\mathbf{J}}$ and the Depth map obtained with dense wavelet coefficients at all scales. Error is displayed within range $[-5m, 5m]$.

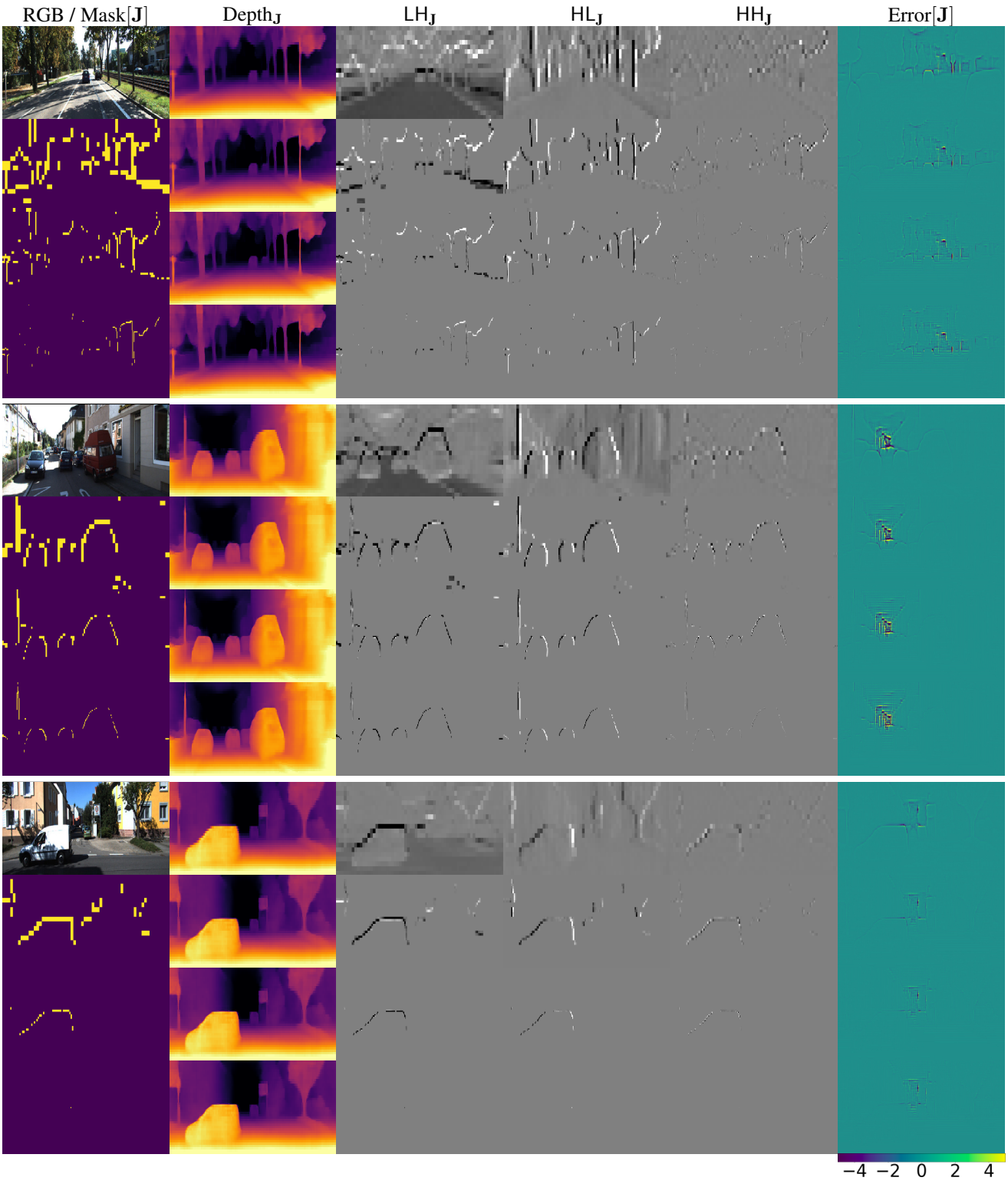


Figure 7: Qualitative results of predicted wavelets coefficients of depth maps on the KITTI dataset (3/3). Our predicted wavelets have two desirable properties: they are sparse, allowing for efficient computation, and they are accurately located around depth edges without needing to supervise them. Results are obtained with $\eta = 0.05$. For each block of results, each row shows coefficients and depth maps obtained at scale **J** in the decoder from lowest to highest scale (decreasing **J**), as well as the (signed-)error between $\text{Depth}_{\mathbf{J}}$ and the Depth map obtained with dense wavelet coefficients at all scales. Error is displayed within range $[-5m, 5m]$.

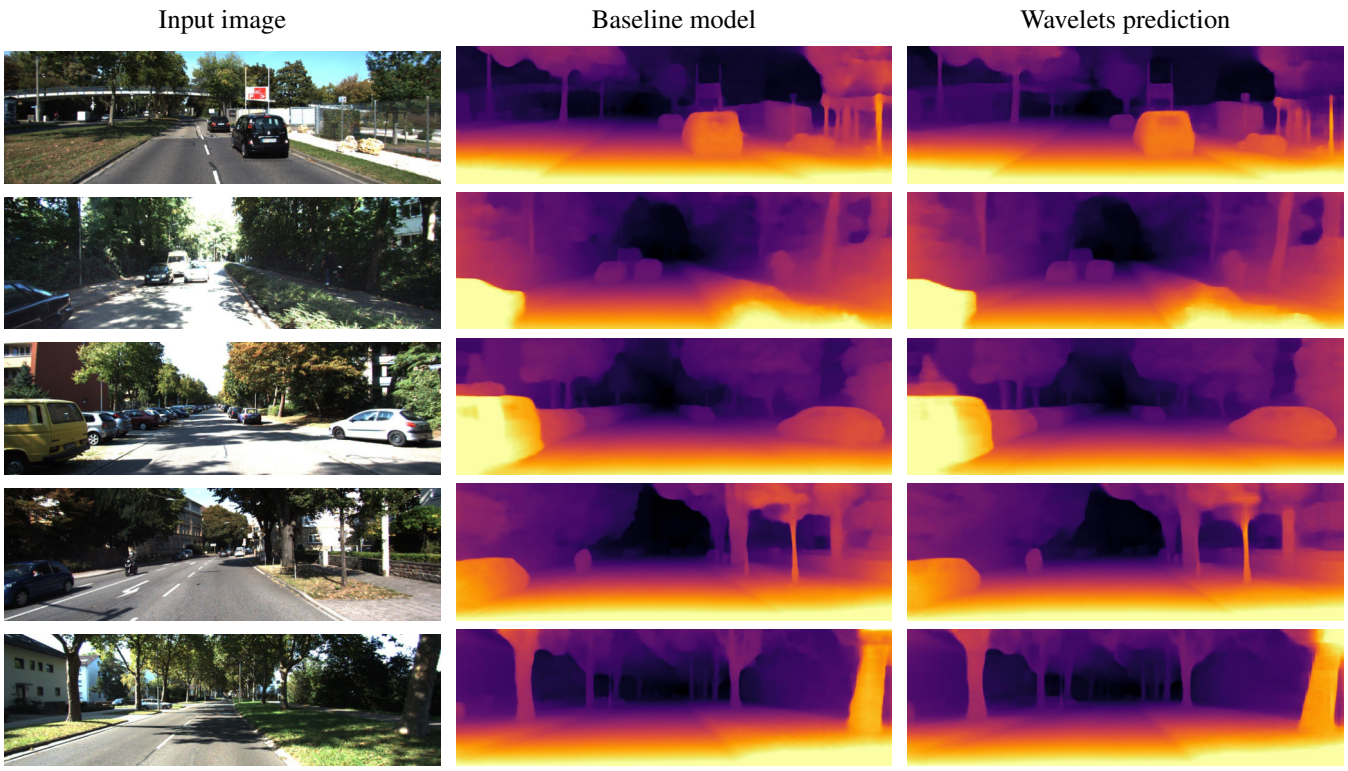


Figure 8: Comparing wavelet predictions to a baseline model on the KITTI dataset. On the left we show the input image, and in the middle column we show the prediction from an off-the-shelf Depth Hints ResNet 50 model [20]. On the right we show an equivalently trained ResNet 50 model, but with our wavelets in the decoder. We see that our predictions retain the high quality of the baseline predictions, but are more efficient to predict.