LocalBins: Improving Depth Estimation by Learning Local Distributions

Shariq Farooq Bhat[1], Ibraheem Alhashim[2], and Peter Wonka[1]

[1] KAUST
[2] National Center for Artificial Intelligence (NCAI), Saudi Data and Artificial Intelligence Authority (SDAIA), Riyadh, Kingdom of Saudi Arabia
shariq.bhat@kaust.edu.sa, ibraheem.alhashim@gmail.com, pwonka@gmail.com

Abstract. We propose a novel architecture for depth estimation from a single image. The architecture itself is based on the popular encoder-decoder architecture that is frequently used as a starting point for all dense regression tasks. We build on AdaBins which estimates a global distribution of depth values for the input image and evolve the architecture in two ways. First, instead of predicting global depth distributions, we predict depth distributions of local neighborhoods at every pixel. Second, instead of predicting depth distributions only towards the end of the decoder, we involve all layers of the decoder. We call this new architecture LocalBins. Our results demonstrate a clear improvement over the state-of-the-art in all metrics on the NYU-Depth V2 dataset. Code and pretrained models will be made publicly available.[3]

Keywords: single image depth estimation, encoder-decoder architecture, deep learning, dense regression, histogram prediction

1 Introduction

In this paper, we propose a new architecture for learning to estimate depth values given a single input image. In this line of work there are two main approaches that have been followed recently. Combining to train on multiple datasets at once while factoring out scale, e.g. [12,28] and training on a single dataset with consistent scale, e.g. [8,19,34,13,35,9,15,2,20,16,4]. Our approach falls in the second category. There are multiple published competing architectures, with AdaBins [4] currently being the most successful architecture on datasets such as NYU-Depth V2 [29]. Our newly proposed architecture, called *LocalBins* aims to improve upon this work.

The main idea of AdaBins is to predict adaptive bins that estimate one "global" depth distribution per image. This prediction works both as auxiliary supervision of depth estimation, but also directly influences the depth prediction.

We initially formulated two objectives in evolving AdaBins. First, we wanted to see if predicting local distributions around each pixel can improve upon predicting one global distribution for the complete input image. Second, we wanted

[3] https://github.com/shariqfarooq123/LocalBins

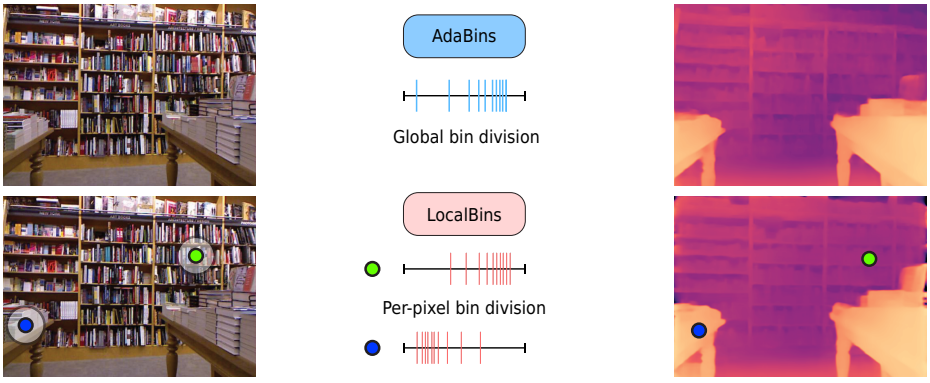

Fig. 1. Illustration of global adaptive bins vs local adaptive bins. While AdaBins predicts a depth distribution for a complete image, LocalBins predicts a depth distribution for the neighborhood of each pixel.

to design the architecture such that depth distribution supervision can be injected earlier in the network, preferably in a multi-scale fashion. AdaBins needs a special architecture design to work. Estimating global adaptive bins needs a transformer at "high resolution". Estimation of bins is done close to the output layer and most of the work is delegated to a specialized module based on a transformer. Even though this improves performance significantly, this offload of work may prevent earlier layers to fully exploit the "distribution supervision" to learn better representations. We call this the 'late injection problem' in our arguments. Any attempts to estimate global adaptive bins earlier in the network (e.g. near the bottleneck) or without a transformer leads to unstable training - divergence or convergence to a suboptimal point.

We realize these ideas in the following manner. To perform local predictions of depth distributions, we propose to use bin estimation at every pixel, and impose regularization on bin predictions via a query and response training scheme. Our proposed module is regularized to predict the depth distributions within the randomly selected bounding boxes within the image.

To perform multi-scale predictions of depth distributions, we let the network predict local depth distributions in a gradual step-wise manner throughout the decoder. Starting with a small N_{seed} number of bins, at the bottleneck each bin is subsequently split into two at every decoder layer, i.e., the i^{th} layer of the decoder estimates $2^i N_{seed}$ bins at every pixel position. Together with locality, this coarse-to-fine construction lets us avoid unstable training and simultaneously solves the late injection problem.

Our proposed LocalBins module is lightweight (adding only ~1M params) and can be used in conjunction with any encoder-decoder network.

To summarize, we make the following contributions:

- We propose a new architecture for single image depth estimation that improves upon the state-of-the-art in all metrics on the NYU-Depth V2 [29] dataset. Models and code will be made publicly available.
- We propose two novel architecture ideas to single image depth estimation: 1) estimating local histograms instead of a single global histogram and 2) estimating histograms in a multi-scale fashion to benefit from distribution supervision earlier in the pipeline. Even beyond depth estimation, we are not aware of existing similar concepts and we believe that these ideas could be beneficial beyond depth estimation.

2 Related Work

There are multiple categories of depth estimation methods. The first category are unsupervised methods [33,41,10,11,5,12,22,40,32]. These methods do not use ground truth depth data, but use self-supervision generally by some form of 3D reconstruction to learn depth values. These methods typically use videos or stereo videos as input. The second category of methods learn depth estimation from multiple datasets [23,28,27] jointly. Combining multiple datasets requires considering the different depth scales of the scenes. Therefore, methods that train on multiple datasets are generally not comparable to methods that train on a single dataset, because the test protocol is different. The third category of algorithms are domain transfer methods [3,38,31,7,1]. These techniques assume the availability of ground truth data in one domain during training, but the images in the target domain do not have ground truth depth (or only a few of them have ground truth depth [39]). The fourth category are depth estimation methods that learn a depth estimation network for each dataset separately [8,19,34,13,35,9,15,2,20,16,4]. Our method belongs to this category of supervised monocular depth estimation. These methods formulate the task as the regression of a depth map from a single RGB input image. The current dominant architecture follows an encoder-decoder network. Most high performing methods apply such architecture with some variations on the process of extracting of relevant feature maps during encoding and the fusion of these features with the intermediate maps produced during the decoding stage. Finally, we mention two very recent arXiv submissions that are concurrent to our work for the sake of completeness. The first is GLP-depth [17] which proposes a hierarchical transformer encoder that captures global features and a simple decoder that considers the local context. The second method [37] employs a neural window fully-connected Conditional Random Fields (CRFs) module for the decoder and a vision transformer for the encoder.

3 Methodology

3.1 Background

AdaBins [4] divides the depth interval (d_{min}, d_{max}) into bins. This bin-division is global (one proposed bin-division per image) and adaptive (varies from image

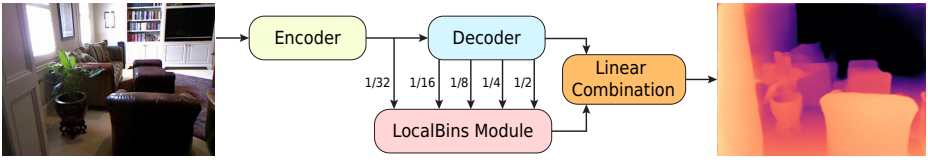

Fig. 2. Architecture overview. LocalBins module contains pixel-wise operations and estimate local neighborhood bin density for each pixel location.

to image), and reflects the global depth distribution for the input image. Each bin can have a different size and the predicted bin centers are closer to each other near more frequently occurring depth values. AdaBins employs an encoder-decoder architecture followed by a transformer based module to predict the adaptive bins. The final depth estimation is obtained by predicting the pixel-wise probability distribution over the bins and computing an expectation over the predicted global bin centers. This can also be seen as expressing the final depth value as a linear combination of bin centers. The reader is referred to [4] for more details.

We build upon the basic idea of AdaBins to estimate depth distributions, but we change the architecture design to incorporate two novel ideas. First, instead of predicting a single global depth bin-division, our architecture estimates a bin-division at every pixel, reflecting the depth distribution in the local neighborhood. Thus, the bin-divisions not only can vary from image to image (adaptiveness) but also from pixel to pixel (locality). Second, we do not use a transformer as a subsequent separate architecture block but integrate the depth prediction more tightly in the decoder. We utilize all the layers of the convolutional decoder to gradually refine the bin-division proposed for each pixel. The details of our architecture are described in the next section.

3.2 Architecture

Our architecture has two major components (see Fig. 2): 1) a standard encoder-decoder block and 2) our proposed LocalBins module.

Encoder-decoder We use the same encoder-decoder architecture as AdaBins to facilitate a fair comparison (EfficientNet-B5 with skip connections).

LocalBins module The LocalBins module uses the bottleneck features and the decoder features from the encoder-decoder block to estimate the local distribution of depth values at every pixel. As in AdaBins, the estimated distributions are encoded as the adaptive bin-divisions of the depth range interval, with the density of resulting bin-centers directly reflecting the density of the depth values in the local neighborhood. In practice, the bin-divisions are formulated as a vector of normalized bin-widths at every pixel from which the bin-centers can be easily obtained via Eq. 5. To estimate the bin-divisions at every pixel, we employ a *coarse-to-fine binning* strategy. Starting with N_{seed} number of bins for every pixel at the bottleneck, the number of bins doubles at every decoder layer.

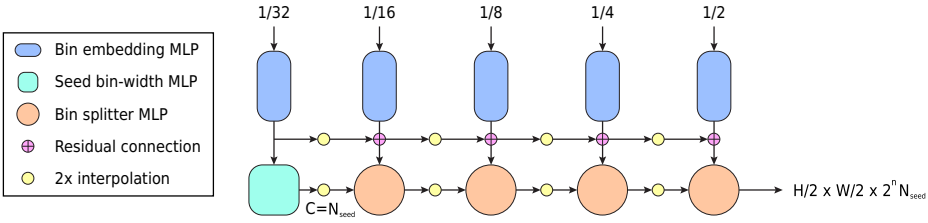

Fig. 3. Design of the LocalBins module. See Sec 3.2 for more details.

The LocalBins module consists of three main types of layers: a) Bin embedding layers b) Seed bin width estimators c) Bin splitters. All these layers are composed of pointwise MLPs (a.k.a 1×1 convolutional blocks) with two hidden layers with hidden dimension h.

a) Bin embedding layers All the feature blocks input to the LocalBins module (the bottleneck features from the encoder and the decoder features of all scales) are first fed to bin embedding layers. These layers project the features of varying channel dimensionality into the same space (dim=128) that we refer to as the 'bin embedding space'. The further layers in the LocalBins module 'decide' the bin-division for each pixel based on their bin-embeddings.

b) Seed bin width estimator This layer takes bin embeddings from the bottleneck as input and predicts N_{seed} number of bins at each pixel of the bottleneck. This bin-division estimate is taken as the seed and subsequently each bin is divided into two bins for every subsequent decoder layer by the bin splitters (along with the spatial 2x interpolation).

c) Bin splitters These pointwise MLPs are used to realize our coarse-to-fine binning strategy. Loosely speaking, bin splitters 'decide' where to put more bin-centers for each pixel based on their bin-embeddings. As illustrated in the Fig. 3, bin-embeddings and bin-widths from the previous layer are first bilinearly upsampled to match the spatial resolution of the current layer. A bin splitter MLP at layer k, denoted as $\mathcal{S}^k$, takes as input the 'current' layer bin-embeddings after a residual connection with the upsampled previous layer bin-embeddings. The output is then used to split each bin-width from the previous layer into two. Specifically, let $\boldsymbol{b}_{ij} \in \mathbb{R}^m$ be the normalized m-bin-widths at pixel location (i, j) (after 2x interpolation). Then, the new $2m$-bin-widths $\boldsymbol{b}'_{ij} \in \mathbb{R}^{2m}$ are given by:

$$\boldsymbol{\alpha}_{ij} = \sigma(\mathcal{S}^k(\mathbf{e}^{k-1}_{ij} + \mathbf{e}^k_{ij})) \tag{1}$$

$$\boldsymbol{b}'_{ij} = \{\alpha^0_{ij} b^0_{ij}, (1-\alpha^0_{ij}) b^0_{ij}, \alpha^1_{ij} b^1_{ij}, (1-\alpha^1_{ij}) b^1_{ij}, \ldots, \alpha^m_{ij} b^m_{ij}, (1-\alpha^m_{ij}) b^m_{ij}\} \tag{2}$$

where, $\sigma(\cdot)$ represents the splitter activation function that outputs values $\alpha \in (0, 1)$, $\mathbf{e}^k_{ij}$ is the bin-embedding of the pixel at (i, j) at the k^{th} layer and v^a represent the components of a vector $\mathbf{v}$.

We explore three designs of the splitter activation function $\sigma(\cdot)$, namely:

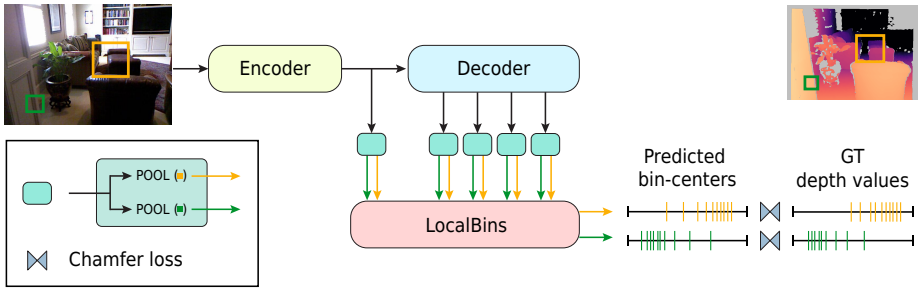

Fig. 4. Illustration of Query-Response training. See Sec 3.3 for more details.

1. Constant splitter: $\sigma(x) = 0.5\ \forall x$, that divides a bin in half irrespective of the splitter MLP output (and indirectly the bin-embeddings).
2. Sigmoid splitter: where $\sigma(\cdot)$ represents the sigmoid activation function.
3. Linear norm splitter: In this case, we let the splitter MLP $\mathcal{S}^k$ output two positive values (x_1, x_2) (via ReLU) for each bin. Then the linear norm split is given by:

$$\sigma(x_1, x_2) = \frac{x_1}{x_1 + x_2 + \epsilon} \tag{3}$$

where $\epsilon = 1e^{-4}$ is used for numerical stability.

Refer to Sec. 5.3 for their comparison.

Since the number of bins doubles every layer, we have $2^n N_{seed}$ bins at the end of an n-layer decoder. Therefore we use $output\_channels = 2^n N_{seed}$ for the last convolutional layer in the decoder. We then use the hybrid regression as in AdaBins, to obtain the final depth map. The difference is that now the bins change from pixel-to-pixel:

$$c(b_{ij}^k) = d_{min} + (d_{max} - d_{min})(b_{ij}^k/2 + \sum_{s=1}^{k-1} b_{ij}^s) \tag{4}$$

$$\tilde{d}_{ij} = \sum_{k=1}^{N} c(b_{ij}^k) p_{ij}^k \tag{5}$$

where $\tilde{d}_{ij}$ is the final estimated depth value, $\boldsymbol{b}_{ij}$ and $\boldsymbol{p}_{ij}$ are the bin-widths at the output layer and softmax scores respectively at location (i, j).

3.3 Training

Our network needs supervision in two forms. First, we need a pixel-wise loss ($\mathcal{L}_{pixel}$) to provide supervision for the final estimated depth values. For this we use the Scale-Invariant Loss as used in recent works [4,20]. Second, we need to supervise our network such that the bin predictions at a pixel actually reflect the density of depth values in its local neighborhood.

A naive way would be to just choose a fixed-sized local window around every pixel (say a 5×5 window) and directly impose the regularization on bin-predictions at a pixel location such that they reflect the corresponding ground truth depth distributions within the window. However, there are two major problems with this approach: 1) It is not clear how one would choose the size of the window. Empirical determination is not a scalable solution as the amount of detail within a fixed sized window varies with the spatial resolution of the image. Alternatively, choosing different sizes simultaneously would lead to inconsistent regularization (e.g. bin predictions at the same pixel location would be compared to GT distributions within, say, 5×5 and 7×7 windows at the same time) 2) Chamfer loss, which is used in AdaBins to implement the distribution loss, is not computationally scalable. In AdaBins, it is computationally feasible because there is only one bin-division proposal per image. While in our case, we have a bin-division proposal at every pixel. This would mean we would have to compute Chamfer loss between the point sets at ~300K locations per image (= total number of pixels) at the highest resolution, which is not feasible in terms of memory or computation. One can, in practice, subsample the number of locations. For example, for a batch size of 16, we could fit around ~2% of randomly selected pixel locations on four NVIDIA A100 GPUs. However, as expected, this leads to inferior performance (Sec. 5.3).

One obvious reason is the significant loss of the spatial coverage of locations at which the loss is computed. We initially identified two possible workarounds for this problem. Either investigate efficient ways for subsampling or let the gradients from the loss computation at a given pixel location directly flow to neighboring regions to increase the coverage. This work focuses on designing the latter solution. To increase the coverage, we propose to involve all the pixel locations within the window to compute the loss (instead of just the center one), while keeping the loss computation feasible. This means that instead of regularizing the bin-predictions at individual pixel locations, we propose to regularize the bin-predictions of the entire local window together, potentially solving the coverage problem. We achieve this via the following formulation that we call 'Query-Response' training.

Query-Response training We train the network via the following locality constraint regularization:

Consider a bounding box $\mathcal{B}$ at any given location in the input image. The LocalBins module, when applied on the spatial average of the features within $\mathcal{B}$, must predict the density of depth values within $\mathcal{B}$.

This is illustrated in Fig. 4. By using bounding boxes of different sizes, we can enforce the features to contain the local distributional information at multiple scales. Furthermore, since the spatial averaging operation covers the entire window, we can potentially achieve the complete coverage with a relatively smaller number of bounding boxes per-image.

In order to implement such a regularization we make use of ROIAlign aggregation [14], a popular operation used in object detection pipelines. ROIAlign

aggregation allows us to extract and pool the features at different layers (bottleneck and decoder features) corresponding to a given bounding box. Details on how ROIAlign works is found in [14].

Given a bounding box $\mathcal{B}$ (*a.k.a* query), we apply ROIAlign aggregation and use the pointwise MLPs (Bin embbeding, seed bin width estimator and bin splitter layers from the LocalBins module) on the pooled features to get the bins $\boldsymbol{b}(\mathcal{B})$ (*a.k.a* the response). The resulting bins $\boldsymbol{b}(\mathcal{B})$ are then 'forced' to match the ground truth depth distribution within that bounding box. We use the 1D bi-directional Chamfer loss as in [4] as the matching loss: $chamfer(\boldsymbol{b}(\mathcal{B}), Depth(\mathcal{B}))$.

"Foveated" Loss We now have a few choices on how to aggregate losses from different bounding boxes to calculate the final loss. We choose to have smaller bounding boxes to have more influence than the larger ones. We therefore use the loss weights that exponentially decay with the bounding box size. Suppose we generate N different sets of bounding boxes $\mathcal{Q}_1, \ldots, \mathcal{Q}_N$ with different box sizes such that $\text{size}(\mathcal{B}_a \in \mathcal{Q}_i) < \text{size}(\mathcal{B}_b \in \mathcal{Q}_j)\ \forall a, b$ and $i < j$, the total loss is given by:

$$\mathcal{L}_{bins} = \sum_{L=1}^{n} \gamma_l^{n-L} \sum_{k=1}^{N} \gamma_b^{k-1} \sum_{\mathcal{B} \in \mathcal{Q}_k} chamfer(\boldsymbol{b}_L(\mathcal{B}), Depth(\mathcal{B})) \tag{6}$$

where n is the number of layers and $b_L(\mathcal{B})$ is the response at layer L (running from bottleneck to output layer). We use $\gamma_l = \gamma_b = 0.3$ in our experiments and use 5 different sizes of bounding boxes as described below. In summary, we follow the steps:

1. Generate five sets of random bounding boxes of sizes, 3×3, 7×7, 15×15, 31×31, 63×63, each containing $M = 200$ boxes.
2. Extract the depth values from the ground truth depth map corresponding to these bboxes to use in the Chamfer loss calculation.
3. Use ROIAlign with average pooling to get the corresponding pooled features at the bottleneck and decoder layers.
4. Use bin embedding MLPs to get the corresponding bin-embeddings. At this stage, we have one bin-embedding vector at each layer corresponding to each bbox.
5. Use seed bin estimator and bin splitter MLPs as usual to get the resulting bins.
6. Calculate the unweighted Chamfer loss for the predicted bins against their ground truth (step 2).
7. Calculate the weights and compute the weighted sum to get the final Chamfer loss, with the weighting scheme in Eq. 6.

Finally, we define the total loss as:

$$\mathcal{L}_{total} = \mathcal{L}_{pixel} + \beta \mathcal{L}_{bins}, \tag{7}$$

where we set $\beta = 0.02$ in all our experiments.

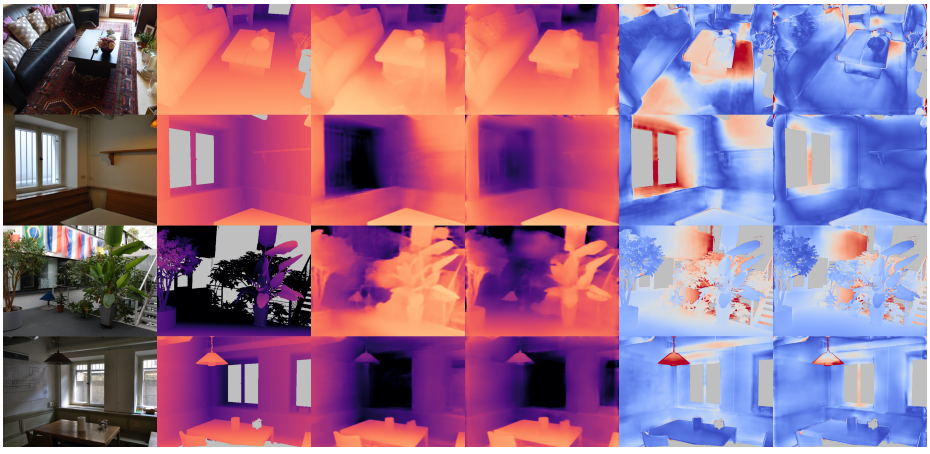

Fig. 5. Qualitative results on iBims-1 benchmark without fine-tuning.

4 Implementation Details

We implement the model in PyTorch [25]. We use hidden dimension $h = 256$ for the Seed bin width MLP and $h = 128$ for Bin embedding MLPs and Bin splitter MLPs. We train the network with batch size of 16 and use the AdamW [24] optimizer with weight decay of 10^{-1}. We use a learning rate of 3.57×10^{-4} which is decayed by a factor of 10^4 in the last 30% iterations using a cosine decay schedule. We train the models for 10 epochs in all our experiments.

5 Experiments and Results

5.1 Comparison to state-of-the-art

NYU-Depth V2. We use NYU-Depth V2 [29] as the most important dataset for evaluation. Table. 1 presents the performance comparison on the official test set of NYU-Depth V2. Our proposed model shows the state-of-the-art performance across all metrics, with ∼4% reduction in the absolute relative error metric. Qualitatively, as shown in Fig. 7, our model is better at predicting depths of thin objects as well as planar surfaces. Note that our final model is able to beat the current published state-of-the-art model AdaBins while having the same encoder-decoder backbone. In total, our model has even fewer parameters since we do not use global attention or transformers. We attribute this performance improvement to the proposed LocalBins design and better utilization of the depth statistics via our novel training scheme.

5.2 Zero-shot performance

To evaluate the generalization performance of our model, we use the model pretrained on the NYU-Depth V2 dataset and evaluate it on other datasets without fine-tuning.

Method	Encoder	#p	$\delta_1\uparrow$	$\delta_2\uparrow$	$\delta_3\uparrow$	REL ↓	RMS ↓	log_{10} ↓
Eigen *et al.* [8]	-	141	0.769	0.950	0.988	0.158	0.641	–
Laina *et al.* [19]	ResNet-50	64	0.811	0.953	0.988	0.127	0.573	0.055
Hao *et al.* [13]	ResNet-101	60	0.841	0.966	0.991	0.127	0.555	0.053
Lee *et al.* [21]	-	119	0.837	0.971	0.994	0.131	0.538	–
Fu *et al.* [9]	ResNet-101	110	0.828	0.965	0.992	0.115	0.509	0.051
SharpNet [26]	-	-	0.836	0.966	0.993	0.139	0.502	0.047
Hu *et al.* [15]	SENet-154	157	0.866	0.975	0.993	0.115	0.530	0.050
Chen *et al.* [6]	SENet	210	0.878	0.977	0.994	0.111	0.514	0.048
Yin *et al.* [36]	ResNeXt-101	114	0.875	0.976	0.994	0.108	0.416	0.048
BTS [20]	DenseNet-161	47	0.885	0.978	0.994	0.110	0.392	0.047
AdaBins [4]	EfficientNet-B5	78	<u>0.903</u>	<u>0.984</u>	<u>0.997</u>	<u>0.103</u>	<u>0.364</u>	<u>0.044</u>
LocalBins (Ours)	EfficientNet-B5	74	**0.907**	**0.987**	**0.998**	**0.099**	**0.357**	**0.042**

Table 1. Comparison of performance on the NYU-Depth V2 dataset. The reported numbers are from the corresponding original papers. Best results are in bold, second best are underlined.

Method	$\delta_1\uparrow$	$\delta_2\uparrow$	$\delta_3\uparrow$	REL↓	RMS↓	log10↓
BTS	0.54	0.86	0.95	0.23	0.93	0.11
AdaBins	0.55	0.87	0.96	**0.21**	0.90	0.11
LocalBins (Ours)	**0.56**	**0.88**	**0.97**	**0.21**	**0.88**	**0.10**

Table 2. Quantitative results on the iBims benchmark without fine-tuning.

iBims-1 benchmark [18] (independent Benchmark images and matched scans version 1) is a high quality RGB-D dataset acquired using a digital single-lens reflex (DSLR) camera and a high-precision laser scanner. Table. 2 lists the performance on this benchmark, with our proposed model outperforming prior state-of-the-art methods. In addition, we show qualitative results in Fig. 5. AdaBins noticeably underestimates the depth range of the scenes with relative error growing with distance, whereas LocalBins more consistently predicts scale-accurate depths across varying depth ranges. This further emphasizes the generalization capability of our model.

SUN-RGBD [30] is an indoor dataset characterized by high scene diversity. We evaluate our model without fine-tuning on the official test set of 5050 images and report the results in Table. 5.

5.3 Analysis and Ablation Studies

Here, we present the results of the extensive experiments we performed to analyse the properties and the importance of various components in our proposed model.

LocalBins module. We first evaluate the importance of our LocalBins module. We remove the LocalBins module from the network and evaluate our base

	Enc-Dec	LBM	Naive	QR	Lbins	Foveated	**REL**	**RMS**
1	✓	✗	✗	✗	✗	✗	0.111	0.419
2	✓	✓	✗	✗	✗	✗	0.106	0.375
3	✓	✓	3×3	✗	✓	✗	0.108	0.381
4	✓	✓	5×5	✗	✓	✗	0.107	0.375
5	✓	✓	15×15	✗	✓	✗	0.108	0.377
8	✓	✓	✗	✓	✓	✗	0.099	0.364
9	✓	✓	✗	✓	✓	✓	0.099	0.357

Table 3. Ablation experiments showing the importance of various components in our proposed model. **Enc-Dec**: Base encoder-decoder model, **LBM**: LocalBins module, **Naive**: Naive training strategy discussed in Sec. 3.2, **QR**: Query-Response training, $\mathcal{L}_{bins}$: Chamfer loss supervision, **Foveated**: Foveated weighting in Chamfer loss. Data in the 'Naive' column indicates the bbox size used on GT to compute density.

	PSCI	#px	**Coverage (%)** ↑	**REL** ↓
Naive	4096	4096	1.33	0.1075
	8192	8192	2.67	0.1071
Query-Response	250	52,130	16.97	0.1043
	500	104,260	33.94	0.1002
	1000	208,520	67.88	0.0992

Table 4. Quantitative demonstration of the efficiency of Query-Response Training. **PSCI**: Point Set Comparisons per Image. **#px**: Total number of pixels covered for loss computation. **Coverage**: Percentage of #px with respect to image resolution.

encoder-decoder architecture. We use the pixel-wise loss ($\mathcal{L}_{pixel}$) to train the network. We also evaluate our proposed model (with LocalBins module) without the Chamfer loss to study the capacity of our design in absence of extra supervision. Results are reported in Table. 3.

Query-Response training and Foveated loss. We evaluate our proposed 'Query-Response' training scheme against the naive implementation discussed in Sec. 3.2. As listed in Table. 3, Query-Response training gives a significant boost to the performance, improving the absolute relative error by ~7%. We believe the Query-Response training scheme is a general, powerful regularization technique that can be directly used in tasks beyond depth estimation. The foveated weighting scheme further improves the squared error based metrics.

In order to demonstrate the power of Query-Response training, we compare it with the Naive scheme in terms of computation of Chamfer loss against the coverage (total number of pixel locations involved in loss computation - higher the better). We define 'PSCI' as the total number of *Point Set Comparisons performed per Image* and use it as an indicator of computational efficiency. Note

Method	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$	REL$\downarrow$	RMS$\downarrow$	$log_{10} \downarrow$
Chen [6]	0.757	0.943	0.984	0.166	0.494	0.071
Yin [36]	0.696	0.912	0.973	0.183	0.541	0.082
BTS [20]	0.740	0.933	0.980	0.172	0.515	0.075
AdaBins [4]	0.771	0.944	0.983	0.159	0.476	0.068
Ours	**0.777**	**0.949**	**0.985**	**0.156**	**0.470**	**0.067**

Table 5. Results of models trained on the NYU-Depth V2 dataset and tested on the SUN RGB-D dataset [30] without fine-tuning.

Splitter activation	**REL**	**RMS**
Constant	0.117	0.454
Sigmoid	0.100	0.361
Linear norm	0.099	0.364

Table 6. Types of splitters.

that in practice, the computations are batched, and PSCI indicates the total number of 'samples' in the batch contributed per image. For the Naive scheme, PSCI is equal to the total number of subsampled locations. For the Query-Response training scheme, PSCI is equal to the total number of bounding box queries per image. We take random bounding boxes of sizes $\{3\times3, 7\times7, 15\times15, 31\times31, 63\times63\}$ and compute their average total area. The results are given in Table. 4. Our proposed training scheme performs ~7.6% better with 8× fewer point set comparisons compared to the naive scheme.

Splitter activation function. We evaluate the three types of splitter activation functions as discussed in Sec. 3.2. The results are given in Table. 6.

Effect of N_{seed}. We analyse the effect of varying the N_{seed} and hence the total number of bins, and compare with AdaBins. The results are plotted in Fig. 6. We find that our model is more robust to the total number of bins used and generally has better performance.

6 Conclusions

We introduced a new network architecture for depth estimation from a single image. We build on an encoder-decoder architecture and evolve the current state-of-the-art model AdaBins in two aspects. First, we add three building blocks to estimate local depth distributions in the neighborhood of a pixel. These three building blocks, bin embedding layers, seed bin width estimator, and bin splitters are tightly integrated with the decoder in a multi-scale fashion. Second, we

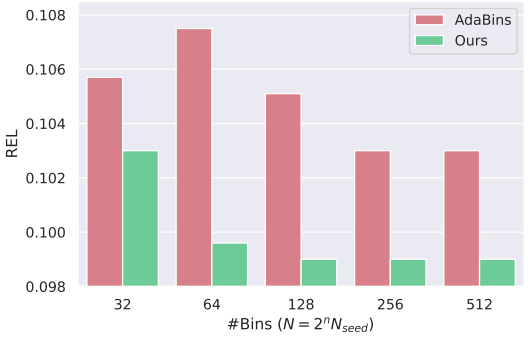

Fig. 6. REL vs #Bins.

propose a query - response acceleration strategy for training, since a naive implementation of the idea would be highly time and memory consuming. In future work, we would like to adapt the LocalBins concept to other dense regression algorithms, such as image segmentation or inpainting.

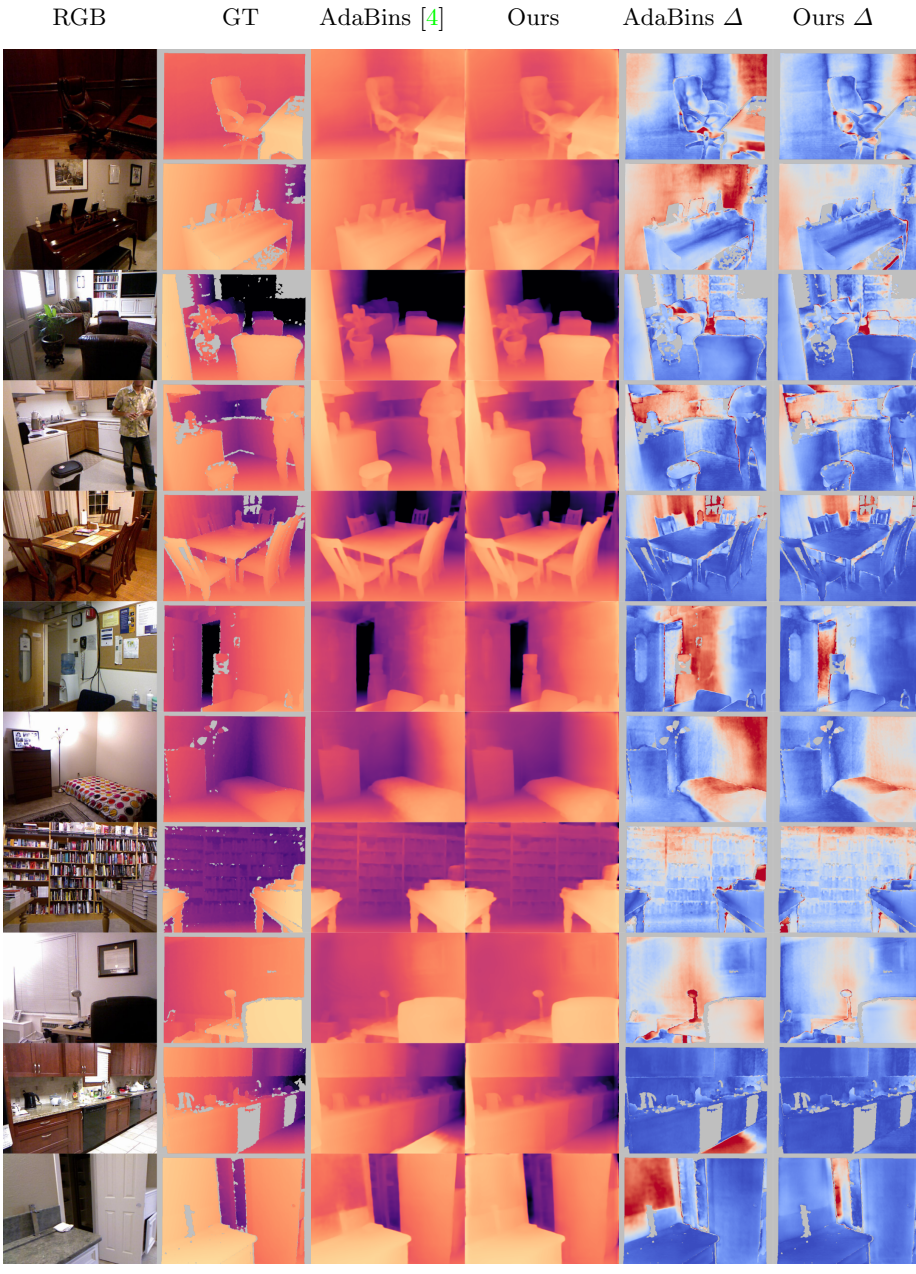

Fig. 7. Qualitative results on NYU-Depth V2.

References

1. Akada, H., Bhat, S.F., Alhashim, I., Wonka, P.: Self-supervised learning of domain invariant features for depth estimation. In: IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022, Waikoloa, HI, USA, January 3-8, 2022. pp. 997–1007. IEEE (2022). https://doi.org/10.1109/WACV51458.2022.00107, https://doi.org/10.1109/WACV51458.2022.00107
2. Alhashim, I., Wonka, P.: High quality monocular depth estimation via transfer learning. CoRR **abs/1812.11941** (2018), http://arxiv.org/abs/1812.11941
3. Atapour-Abarghouei, A., Breckon, T.P.: Real-time monocular depth estimation using synthetic data with domain adaptation via image style transfer. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2800–2810 (2018)
4. Bhat, S.F., Alhashim, I., Wonka, P.: Adabins: Depth estimation using adaptive bins. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4008–4017. IEEE Computer Society, Los Alamitos, CA, USA (jun 2021). https://doi.org/10.1109/CVPR46437.2021.00400, https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.00400
5. Casser, V., Pirk, S., Mahjourian, R., Angelova, A.: Unsupervised monocular depth and ego-motion learning with structure and semantics. In: CVPR Workshop on Visual Odometry and Computer Vision Applications Based on Location Cues (VOCVALC) (2019)
6. Chen, X., Chen, X., Zha, Z.J.: Structure-aware residual pyramid network for monocular depth estimation. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. pp. 694–700. International Joint Conferences on Artificial Intelligence Organization (7 2019). https://doi.org/10.24963/ijcai.2019/98, https://doi.org/10.24963/ijcai.2019/98
7. Chen, Y.C., Lin, Y.Y., Yang, M.H., Huang, J.B.: Crdoco: Pixel-level domain transfer with cross-domain consistency. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
8. Eigen, D., Puhrsch, C., Fergus, R.: Depth map prediction from a single image using a multi-scale deep network. In: NIPS (2014)
9. Fu, H., Gong, M., Wang, C., Batmanghelich, N., Tao, D.: Deep ordinal regression network for monocular depth estimation. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition pp. 2002–2011 (2018)
10. Godard, C., Aodha, O.M., Brostow, G.J.: Unsupervised monocular depth estimation with left-right consistency. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 6602–6611 (2017)
11. Godard, C., Aodha, O.M., Brostow, G.J.: Digging into self-supervised monocular depth estimation. CoRR **abs/1806.01260** (2018)
12. Gordon, A., Li, H., Jonschkowski, R., Angelova, A.: Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019)
13. Hao, Z., Li, Y., You, S., Lu, F.: Detail preserving depth estimation from a single image using attention guided networks. 2018 International Conference on 3D Vision (3DV) pp. 304–313 (2018)
14. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: 2017 IEEE International Conference on Computer Vision (ICCV). pp. 2980–2988 (2017). https://doi.org/10.1109/ICCV.2017.322

15. Hu, J., Ozay, M., Zhang, Y., Okatani, T.: Revisiting single image depth estimation: Toward higher resolution maps with accurate object boundaries. 2019 IEEE Winter Conference on Applications of Computer Vision (WACV) pp. 1043–1051 (2018)
16. Huynh, L., Nguyen-Ha, P., Matas, J., Rahtu, E., Heikkila, J.: Guiding monocular depth estimation using depth-attention volume. arXiv preprint arXiv:2004.02760 (2020)
17. Kim, D., Ga, W., Ahn, P., Joo, D., Chun, S., Kim, J.: Global-local path networks for monocular depth estimation with vertical cutdepth. arXiv preprint arXiv:2201.07436 (2022)
18. Koch, T., Liebel, L., Fraundorfer, F., Körner, M.: Evaluation of cnn-based single-image depth estimation methods. In: Proceedings ECCV 2018 Workshops (2019)
19. Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., Navab, N.: Deeper depth prediction with fully convolutional residual networks. 2016 Fourth International Conference on 3D Vision (3DV) pp. 239–248 (2016)
20. Lee, J.H., Han, M.K., Ko, D.W., Suh, I.H.: From big to small: Multi-scale local planar guidance for monocular depth estimation. arXiv preprint arXiv:1907.10326 (2019)
21. Lee, W., Park, N., Woo, W.: Depth-assisted real-time 3d object detection for augmented reality. ICAT'11 **2**, 126–132 (2011)
22. Li, H., Gordon, A., Zhao, H., Casser, V., Angelova, A.: Unsupervised monocular depth learning in dynamic scenes. arXiv preprint arXiv:2010.16404 (2020)
23. Li, Z., Snavely, N.: Megadepth: Learning single-view depth prediction from internet photos. In: Computer Vision and Pattern Recognition (CVPR) (2018)
24. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), https://openreview.net/forum?id=Bkg6RiCqY7
25. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32, pp. 8026–8037. Curran Associates, Inc. (2019), https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf
26. Ramamonjisoa, M., Lepetit, V.: Sharpnet: Fast and accurate recovery of occluding contours in monocular depth estimation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops (Oct 2019)
27. Ranftl, R., Bochkovskiy, A., Koltun, V.: Vision transformers for dense prediction. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 12179–12188 (October 2021)
28. Ranftl, R., Lasinger, K., Hafner, D., Schindler, K., Koltun, V.: Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) (2020)
29. Silberman, N., Hoiem, D., Kohli, P., Fergus, R.: Indoor segmentation and support inference from rgbd images. In: Computer Vision – ECCV 2012. pp. 746–760. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
30. Song, S., Lichtenberg, S.P., Xiao, J.: Sun rgb-d: A rgb-d scene understanding benchmark suite. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 567–576 (2015). https://doi.org/10.1109/CVPR.2015.7298655

31. Tonioni, A., Poggi, M., Mattoccia, S., di Stefano, L.: Unsupervised domain adaptation for depth prediction from images. CoRR **abs/1909.03943** (2019), http://arxiv.org/abs/1909.03943
32. Watson, J., Mac Aodha, O., Prisacariu, V., Brostow, G., Firman, M.: The temporal opportunist: Self-supervised multi-frame monocular depth. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1164–1174 (June 2021)
33. Xie, J., Girshick, R.B., Farhadi, A.: Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In: ECCV (2016)
34. Xu, D., Ricci, E., Ouyang, W., Wang, X., Sebe, N.: Multi-scale continuous crfs as sequential deep networks for monocular depth estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5354–5362 (2017)
35. Xu, D., Wang, W., Tang, H., Liu, H.W., Sebe, N., Ricci, E.: Structured attention guided convolutional neural fields for monocular depth estimation. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition pp. 3917–3925 (2018)
36. Yin, W., Liu, Y., Shen, C., Yan, Y.: Enforcing geometric constraints of virtual normal for depth prediction. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019)
37. Yuan, W., Gu, X., Dai, Z., Zhu, S., Tan, P.: NeW CRFs: Neural Window Fully-connected CRFs for Monocular Depth Estimation. arXiv e-prints arXiv:2203.01502 (Mar 2022)
38. Zhao, S., Fu, H., Gong, M., Tao, D.: Geometry-aware symmetric domain adaptation for monocular depth estimation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9788–9798 (2019)
39. Zhao, Y., Kong, S., Shin, D., Fowlkes, C.: Domain decluttering: Simplifying images to mitigate synthetic-real domain shift and improve depth estimation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
40. Zhou, H., Greenwood, D., Taylor, S.: Self-supervised monocular depth estimation with internal feature fusion. In: British Machine Vision Conference (BMVC) (2021)
41. Zhou, T., Brown, M.R., Snavely, N., Lowe, D.G.: Unsupervised learning of depth and ego-motion from video. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 6612–6619 (2017)

A Appendix

A.1 Visualizing bin predictions

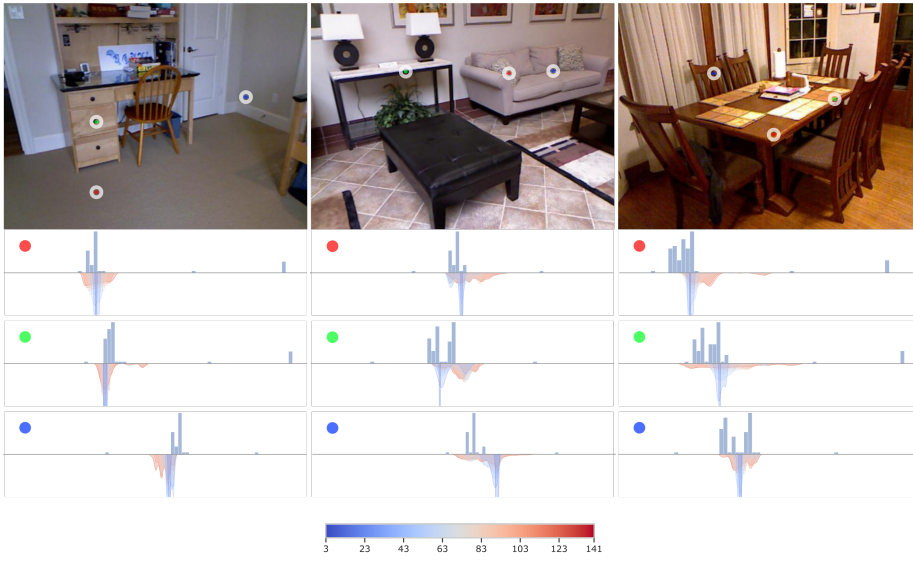

Fig. 8. Visualization of bin predictions, showing input RGB (first row) and density plots of local depth distributions at selected locations across the depth range interval. Density plots consisting of density of bin centers predicted by LocalBins module at the selected pixel location (**top**) and density of ground truth depth values (**bottom**) for various window sizes (indicated by the colorbar).

Fig. 8 shows a qualitative comparison of the bin predictions of the LocalBins module against the ground truth depth distribution in the local neighborhoods of various sizes.

Our main goal in this work was to have the LocalBins module predict the local depth distribution at each pixel. However, in Query-Response training, windows of various sizes are generated, and regularization is imposed on the bin predictions of the entire window together rather than for each individual pixel location separately. While being vital (refer to 'Query-Response training' section in the main paper), this renders the 'size' of the 'local neighborhood' rather indeterministic. The model may either end up using a fixed size and always predict the bins at each pixel that reflect the density of depth values within a fixed-sized neighborhood, or the model may as well choose to cover different sized neighborhoods depending upon the context. Visualizations (See Fig. 9) indicate the latter case.

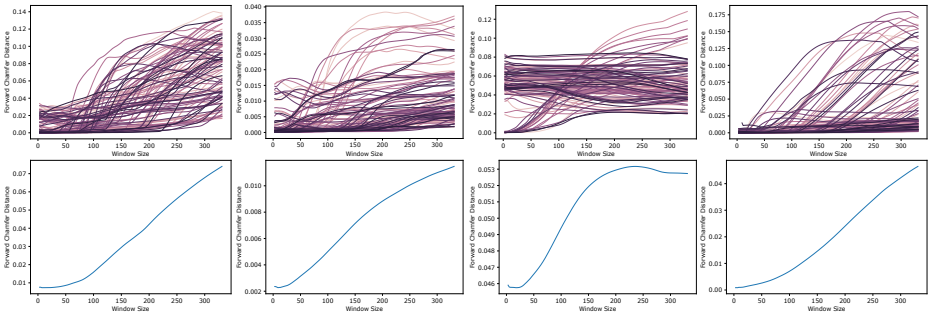

Fig. 9. How *"local"* are the bin predictions of the LocalBins module? Plots show the average absolute difference between the ground truth depth values and their nearest bin centers predicted by the LocalBins module. Top row shows the average differences plotted for 100 random locations for four randomly selected input images (columns). GT depth values are taken from the 'concentric' bounding boxes and compared against the bins predicted at the center. Bottom row shows the mean across the 100 locations for each image.

A.2 Different Backbones

Backbone	#params(M)	d1	d2	d3	REL	RMS	log10
MobileNetV2-100	20.26	0.812	0.963	0.992	0.141	0.480	0.060
EfficientNetV2-S	38.71	0.894	0.981	0.995	0.106	0.376	0.045
EfficientNetV2-M	71.53	0.898	0.983	0.996	0.102	0.365	0.044
EfficientNetV2-L	136.3	0.910	0.986	0.997	0.098	0.351	0.042

Table 7. Performance of LocalBins with various backbone encoders

We switch the EfficientNet-b5 encoder in our default model with various other backbones and list the performance in Table. 7