Monocular Depth Estimation Using Neural Regression Forest

Anirban Roy
Oregon State University
Corvallis, OR 97331
royani@eecs.oregonstate.edu

Sinisa Todorovic
Oregon State University
Corvallis, OR 97331
sinisa@eecs.oregonstate.edu

Abstract

This paper presents a novel deep architecture, called neural regression forest (NRF), for depth estimation from a single image. NRF combines random forests and convolutional neural networks (CNNs). Scanning windows extracted from the image represent samples which are passed down the trees of NRF for predicting their depth. At every tree node, the sample is filtered with a CNN associated with that node. Results of the convolutional filtering are passed to left and right children nodes, i.e., corresponding CNNs, with a Bernoulli probability, until the leaves, where depth estimations are made. CNNs at every node are designed to have fewer parameters than seen in recent work, but their stacked processing along a path in the tree effectively amounts to a deeper CNN. NRF allows for parallelizable training of all "shallow" CNNs, and efficient enforcing of smoothness in depth estimation results. Our evaluation on the benchmark Make3D and NYUv2 datasets demonstrates that NRF outperforms the state of the art, and gracefully handles gradually decreasing training datasets.

1. Introduction

This paper address one of the basic vision problems, that of estimating depth from a single monocular image. Our goal is to predict the continuous depth values of every pixel. The image may show a natural outdoor scene with continuous depth values ranging from a few to 80 meters, or an indoor cluttered living space with continuous depths ranging between 0 and 10 meters, as shown in Fig. 1.

This problem is challenging, since there may be many distinct 3D scenes depicted in the same 2D image. Also, histograms of depths in typical outdoor and indoor scenes are "peaky" (Fig. 1). Hence, it is very likely that a training set may under-represent or simply not have examples of certain depth values, which could appear in test data.

To address these challenges, prior work typically uses graphical models for enforcing smoothness and taking into account spatial context in depth estimation [21, 22, 1, 14, 12]. More recent work resorts to convolutional neural networks (CNNs) [6, 15]. CNNs are appealing as they can efficiently incorporate multiscale contextual cues, in a feed-forward manner. However, one of the major bottlenecks of using CNNs for depth estimation is that their training typically requires big data. As Fig. 1 illustrates, both indoor and outdoor scenes may not have a sufficient number of examples of certain depth values, which makes the training of CNNs challenging.

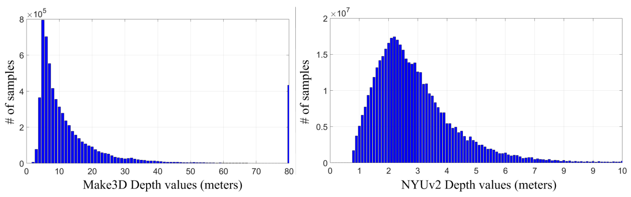

Figure 1. Depth histograms extracted from two benchmark datasets: Make3D dataset [22] (left), and NYUv2 dataset [18] (right). Depth values are in meters.

In this paper, we present a novel deep architecture, called neural regression forest (NRF), for monocular depth estimation. As shown in Fig. 2, NRF combines CNNs with Regression Forest [5, 3] for predicting depths in the continuous domain via regression. Robustness is achieved by processing a data sample with an ensemble of binary regression trees, which we call Convolutional Regression Trees (CRTs). We fuse individual regression results of every CRT into a final depth estimation.

Our approach scans a window $\boldsymbol{x}$ across every pixel location, and passes $\boldsymbol{x}$ down the CRTs for predicting the depth of the scanning-window's center. At every node of CRT, $\boldsymbol{x}$ is filtered with a CNN associated with that node, and then passed to left and right children nodes (i.e., CNNs) with a Bernoulli probability for further convolutional processing, until $\boldsymbol{x}$ reaches leaves of the tree. The probability that $\boldsymbol{x}$ reaches a particular leaf is equal to the product of all Bernoulli probabilities along the data's path from the root to that leaf. The sample undergoes the same procedure in other CRTs of NRF. Finally, depth estimations made in every leaf are weighted with the corresponding path probabilities for

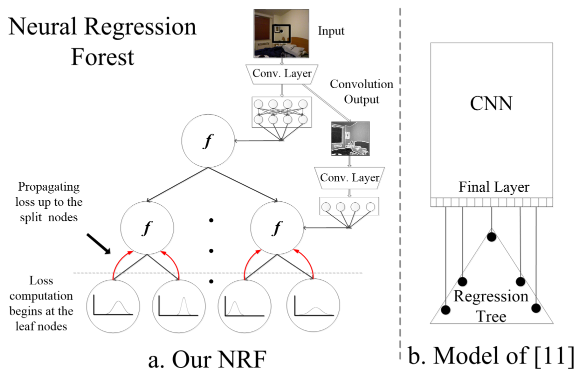

Figure 2. Neural Regression Forest. (a) A CNN is associated with every node of a binary Convolutional Regression Tree (CRT) for performing the convolutional processing of data samples. The CNN's output is passed to the left and right children nodes with a Bernoulli probability. (b) While our CNNs process data samples as they pass down the CRT, the related deep architecture of [11] uses a single deep CNN to fully process the data before passing them through a decision tree.

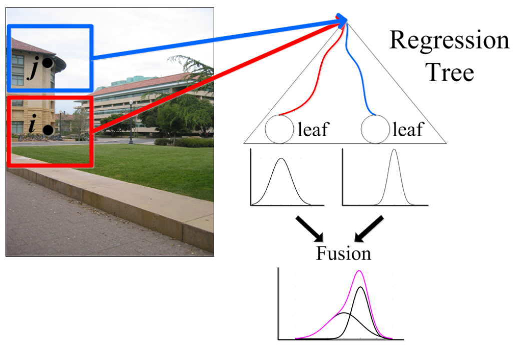

Figure 3. The window at pixel i and a window at neighboring pixel j have similar depths, but are different in appearance. Therefore, they could reach distinct NRF leaf nodes with different probabilities. The final depth prediction for the ith pixel is made by combining the Gaussian at the leaf corresponding to pixel i and also the Gaussian at the leaf corresponding to the neighboring pixel j. NRF allows us to effectively increase the (small) number of training samples by considering the pairs of samples and thus improve depth prediction of window $\boldsymbol{x}_i$ by additionally considering the NRF leaf reached by neighboring window $\boldsymbol{x}_j$.

estimating depth d of $\boldsymbol{x}$.

NRF has a number of advantages. First, each CRT node solves a binary problem, instead of a multiclass or regression problem. This allows for a robust training of our CNNs, since the left or right routing of data samples may combine a range of depth values, and thus compensate for some underrepresented (or missing) depth examples in the training data. Second, training of all CNNs in CRT is parallelizable. In particular, once a training sample reaches leaf nodes, we can readily compute the regression loss for every node of CRT, and then use these loss functions for simultaneously training the corresponding CNNs. Third, we design CNNs at every node of CRT to have significantly fewer parameters than seen in recent work. Specifically, we use "shallow" architectures with only a few convolutional (1-2) and fully connected levels (1-2). This, in turn, allows for robust training on smaller datasets, as necessary in our application domain (Fig. 1). It is note worthy that results of convolutional filtering at a node are used as input to its children. Hence, all CNN computations along a path from the root to the leaf amount to processing the data sample with a relatively deep CNN. Therefore, despite using "shallow" CNNs at each CRT node, we expect to have the usual benefits of deep processing of data as in related work which advocates the use of deeper architectures.

In our approach, we also consider more explicit ways of addressing the small number of training examples of certain depths. As illustrated in Fig. 3, depths in a neighborhood of scanning window $\boldsymbol{x}_i$ are often smooth. Therefore, for a more reliable depth prediction of $\boldsymbol{x}_i$, we also consider depth estimates of neighboring scanning windows $\boldsymbol{x}_j$. Neighborhood relationships for all pairs of pixels (i,j) are efficiently estimated using the standard bi-lateral filtering [25].

A comparison with the state of the art on the Make3D [22] and NYU v2 [18] datasets demonstrates our superior performance in terms of relative error, root mean squared error, and log-10 error. Also, our results show that NRF gracefully handles gradually decreasing training datasets.

In the following, Sec. 2 reviews prior work, Sec. 3 specifies our NRF, Sec. 4 explains how we enforce smoothness in depth predictions, Sec. 5 derives expressions for learning CNN and CRT leaf parameters, Sec. 6 specifies details of our CNN architecture, and Sec. 7 presents our results.

2. Related Work

The section reviews related work on monocular depth estimation, and combining CNNs and Random Forests.

Depth Estimation. Monocular depth estimation is a long-standing problem [21, 22, 1, 14, 12, 7, 6, 15]. A single CNN has been used for depth estimation [6, 15]; however, at a coarser resolution than that of the input image (e.g., 1/4 of the original resolution [6]). Depth estimation in the continuous domain has also been considered [24, 9, 27]. However, these approaches use a pre-constructed 3D shape database of known objects [24], or rely on class-specific object keypoints and object segmentations [9], whereas our method is aimed at general scenes.

Random Forests [5, 2, 4, 23] have a long-track record in computer vision. Recent work integrates decision trees with multilayer perceptrons for scene labeling [20]. Each node of a decision tree represents a multilayer perceptron aimed at learning the node's split function given hand-crafted image features. Instead, we use a tree of CNNs to jointly learn the feature representation and split functions for each node in the tree. A single deep CNN has been combined with a random forest for image classification [11]. Outputs of the top CNN layer are considered as nodes of the decision tree. Prediction loss is computed at each split node of the tree and back-propagated to the network for learning CNN parameters. In contrast, we learn a set of distinct "shallow" CNNs in every node of the decision tree. Importantly, we do not back-propagate the loss of depth prediction bottom-up as in [20]. Rather, we compute distinct loss for every CNN in the tree, and then use these losses for parallel training of all CNNs in the tree.

3. Neural Regression Forest

We consider a regression problem, where for each scanning window of the image, $\boldsymbol{x} \in \mathcal{X}$, there is a real-valued depth $d \in \mathcal{D} = [0,\ d_{\max}]$. As the regression model, we use NRF that consists of a set of CRTs, $\mathcal{F} = \{\mathcal{T}_i\}$. Each CRT represents a weak regressor. Depth prediction is made by averaging the predictions over the trees as

$$p_{\mathcal{F}}(d|\boldsymbol{x}) = \frac{1}{|\mathcal{F}|} \sum_{\mathcal{T} \in \mathcal{F}} p_{\mathcal{T}}(d|\boldsymbol{x}), \tag{1}$$

where $p_{\mathcal{F}}(\cdot)$ and $p_{\mathcal{T}}(\cdot)$ denote probability distributions.

Each $\mathcal{T}$ consists of a set of non-leaf decision nodes (also called split nodes), $\mathcal{V}_{\mathcal{T}} = \{v\}$, and a set of leaf nodes for prediction $\mathcal{L}_{\mathcal{T}} = \{l\}$. Each $v \in \mathcal{V}_{\mathcal{T}}$ represents a CNN with parameters $\boldsymbol{w}_v$. The parameters of all CNNs in $\mathcal{T}$ form a set $\mathcal{W}_{\mathcal{T}} = \{\boldsymbol{w}_v : v \in \mathcal{V}_{\mathcal{T}}\}$. Note that CNNs are not associated with leaf nodes. The architecture of CNNs is explained in Sec. 6.

A split function $f_v(\boldsymbol{x}, \boldsymbol{w}_v) : \mathcal{X} \to [0,\ 1]$ is defined for each split node of $\mathcal{T}$. This function is computed by vth CNN with a single output node, which produces the value of the split function through a sigmoid function. Instead of deterministic, we use a stochastic split function, as in [11]. Specifically, $f_v(\boldsymbol{x}, \boldsymbol{w}_v)$ returns the Bernoulli probability of directing $\boldsymbol{x}$ to the left or right child node. In this way, $\boldsymbol{x}$ is passed through $\mathcal{T}$ reaching every leaf $l \in \mathcal{L}_{\mathcal{T}}$ with the corresponding probability $P(l|\boldsymbol{x}, \mathcal{W}_{\mathcal{T}})$, where $\sum_{l \in \mathcal{L}_{\mathcal{T}}} P(l|\boldsymbol{x}, \mathcal{W}_{\mathcal{T}}) = 1$. Following [11], we define

$$P(l|\boldsymbol{x}, \mathcal{W}_{\mathcal{T}}) = \prod_{v \in \mathcal{V}_{\mathcal{T}}} f_v(\boldsymbol{x}; \boldsymbol{w}_v)^{\mathbb{L}(l,v)} (1 - f_v(\boldsymbol{x}; \boldsymbol{w}_v))^{\mathbb{R}(l,v)}, \tag{2}$$

where $\mathbb{L}(l,v)$ and $\mathbb{R}(l,v)$ indicate whether l belongs to the left and right subtree of v, respectively. Note that the product is computed by considering all the split nodes in the tree. For split nodes v which do not lie on the path to l, we have $\mathbb{L}(l,v) = \mathbb{R}(l,v) = 0$. Therefore, the product in (2) effectively computes the probability of the sample's path from the root to the leaf l.

Each leaf $l \in \mathcal{L}_{\mathcal{T}}$ holds a Gaussian probability distribution, $p(d; \boldsymbol{\theta}_l)$, with parameters $\boldsymbol{\theta}_l = (\mu_l, \sigma_l)$, over depth values $\mathcal{D}$. The set of all Gaussian parameters, $\Theta_{\mathcal{T}} = \{\boldsymbol{\theta}_l : l \in \mathcal{L}_{\mathcal{T}}\}$, are computed from training samples that reached leaf nodes of $\mathcal{T}$ during training.

When $\boldsymbol{x}$ reaches the leaves in $\mathcal{T}$, prediction for its depth is given by

$$p_{\mathcal{T}}(d|\boldsymbol{x}) = \sum_{l \in \mathcal{L}_{\mathcal{T}}} p(d; \boldsymbol{\theta}_l) P(l|\boldsymbol{x}, \mathcal{W}_{\mathcal{T}}). \tag{3}$$

4. Enforcing Smoothness in Depth Predictions

NRF allows an efficient way of enforcing smoothness in pixel depth predictions across the image. As mentioned in Sec. 1, and illustrated in Fig. 3, neighboring pixels in the image at similar depths, i and j, may define image windows $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ with significantly different appearance. When passing $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ through $\mathcal{T}$, the convolutional neural processing in every split node v of $\mathcal{T}$ may estimate different values of split functions $f_v(\boldsymbol{x}_i, \boldsymbol{w}_v)$ and $f_v(\boldsymbol{x}_j, \boldsymbol{w}_v)$, resulting in different probabilities of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ reaching a particular leaf $l \in \mathcal{L}$.

To enforce smoothness in depth predictions, we modify the depth likelihood of every $\boldsymbol{x}_i$ given by (3). The modification is based on the depth likelihoods of neighboring samples $\boldsymbol{x}_j$, $j \in \mathtt{N}(i)$, and appearance similarity between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ estimated by bi-lateral filtering [25]. Thus, when $\boldsymbol{x}_i$ reaches the leaves in $\mathcal{T}$, we compute the modified probability distribution of depth d of $\boldsymbol{x}_i$ as

$$\bar{p}_{\mathcal{T}}(d|\boldsymbol{x}_i) = \sum_{j \in \mathtt{N}(i)} \kappa_{ij} p_{\mathcal{T}}(d|\boldsymbol{x}_j), \tag{4}$$

where i's neighborhood $\mathtt{N}(i)$ is defined to also include i, $i \in \mathtt{N}(i)$. κ_{ij} is the weight of bi-lateral filtering, estimated based on the Euclidean distance between the HSV color histograms of windows $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and the Euclidean distance between locations of i and j in the image as

$$\kappa_{ij} \propto \exp(-\frac{\| \boldsymbol{z}_i - \boldsymbol{z}_j \|^2}{\sigma_z^2})\ \exp(-\frac{\| \boldsymbol{h}_i - \boldsymbol{h}_j \|^2}{\sigma_h^2}) \tag{5}$$

where $\boldsymbol{z}_i = (x_i, y_i)$ is the pixel location in the image, $\boldsymbol{h}_i$ is the HSV color histogram of window $\boldsymbol{x}_i$, and $\sigma_z = 3$ and $\sigma_h = 30$ control sensitivity [25]. Note that for $j = i$, the weight κ_{ii} has the largest value. We normalize the weights such that $\sum_{j \in \mathtt{N}(i)} \kappa_{ij} = 1$.

The effect of using bi-lateral filtering in (4) is that only neighboring pixels of i with high color similarity with i contribute to the depth prediction of i. This means that our bi-lateral filtering *is not a post-processing step*. Rather, it is integrated with our training and inference. Note that bi-lateral filtering can be done only once per image. Hence, the above modification of computing $\bar{p}_{\mathcal{T}}(d|\boldsymbol{x}_i, \mathcal{W}, \Theta)$, given by

(4), minimally increases our complexity. We use an off-the-shelf linear-time implementation for the bi-lateral filtering [19]. Our results show that this modification in (4) improves depth estimation, producing smoother depth maps.

5. Learning

Our learning of each $\mathcal{T}$ estimates the following parameters: 1) Θ - the set of Gaussian parameters in the leaf nodes, and 2) $\mathcal{W}$ - the set of CNN parameters of all the split nodes. For simplicity, below, we drop the explicit reference to $\mathcal{T}$ in our notation. This learning differs from the traditional supervised setting, because there is no ground truth for directing a training sample to the left or right in the tree. Thus, we learn $(\Theta, \mathcal{W})$ by maximizing the log-likelihood of the training data, as in [20, 11]. Given a training set $S \subset \mathcal{X} \times \mathcal{D}$, we define the empirical loss as

$$R(\mathcal{W}, \Theta; S) = \frac{1}{|S|} \sum_{(\boldsymbol{x},d)\in S} L(\mathcal{W}, \Theta; \boldsymbol{x}, d) \tag{6}$$

where $L(\mathcal{W}, \Theta; \boldsymbol{x}, d)$ is the negative log-likelihood of the depth prediction for $\boldsymbol{x}$

$$L(\mathcal{W}, \Theta; \boldsymbol{x}, d) = -\log p(d|\boldsymbol{x}), \tag{7}$$

where $p(d|\boldsymbol{x})$ is given by (3).

Note that we also specify an alternative training, when smoothing is enforced in depth estimation. For this training, we modify the definition of the empirical loss to $\bar{R}(\mathcal{W}, \Theta; S)$ specified in terms of the negative log-likelihoods of the smoothed depth predictions

$$\bar{L}(\mathcal{W}, \Theta; \boldsymbol{x}, d) = -\log \bar{p}(d|\boldsymbol{x}), \tag{8}$$

where $\bar{p}(d|\boldsymbol{x})$ is given by (4).

The learning objective is defined as

$$(\mathcal{W}^*, \Theta^*) = \arg\min_{\mathcal{W},\Theta} R(\mathcal{W}, \Theta; S), \tag{9}$$

and similarly for the case when we use $\bar{R}(\mathcal{W}, \Theta; S)$.

To solve (9), we alternate the following two steps. In the first step, we fix $\mathcal{W}$, and optimize for Θ, and in the second step, we fix Θ and optimize for $\mathcal{W}$. Learning iterations end after convergence, or when they reach a maximum number of iterations.

5.1. Learning CCN Parameters for the Split Nodes

In this section we describe how to learn the CNN parameters for the split nodes $v \in \mathcal{V}$, $\boldsymbol{w}_v$. We compute the gradient of the log-loss of the tree, $L(\mathcal{W}, \Theta; \boldsymbol{x}, d)$ given by (7), with respect to $\boldsymbol{w}_v$. The gradient is first computed for the final output node of the CNN and then passed to the lower layers of the CNN using the standard backpropagation [13]. The gradient at the final output node is computed as

$$\frac{\partial\, L(\mathcal{W}, \Theta; \boldsymbol{x}, d)}{\partial \boldsymbol{w}_v} = \frac{\partial\, L(\mathcal{W}, \Theta; \boldsymbol{x}, d)}{\partial f_v(\boldsymbol{x}, \boldsymbol{w}_v)} \frac{f_v(\boldsymbol{x}, \boldsymbol{w}_v)}{\partial \boldsymbol{w}_v}, \tag{10}$$

where only the first term depends on the tree, and the second term can be computed as the standard derivative of the sigmoid function. The first term can be computed as

$$\frac{\partial\, L(\boldsymbol{x}, d)}{\partial f_v(\boldsymbol{x})} = f_v(\boldsymbol{x}) \frac{p_{\mathcal{L}_{v_r}}(d|\boldsymbol{x})}{p_{\mathcal{T}}(d|\boldsymbol{x})} + (1 - f_v(\boldsymbol{x})) \frac{p_{\mathcal{L}_{v_l}}(d|\boldsymbol{x})}{p_{\mathcal{T}}(d|\boldsymbol{x})}, \tag{11}$$

where $L(\boldsymbol{x}, d)$ and $f_v(\boldsymbol{x})$ are the shorthand notation for $L(\mathcal{W}, \Theta; \boldsymbol{x}, d)$ and $f_v(\boldsymbol{x}, \boldsymbol{w}_v)$, respectively; $p_{\mathcal{T}}(d|\boldsymbol{x})$ is given by (3), and $p_{\mathcal{L}_{v_r}}(d|\boldsymbol{x})$ and $p_{\mathcal{L}_{v_l}}(d|\boldsymbol{x})$ are analogous to $p_{\mathcal{T}}(d|\boldsymbol{x})$ but defined for the sub-tree rooted at the right and left child of v, v_l and v_r, respectively: $p_{\mathcal{L}_{v_r}}(d|\boldsymbol{x}) = \sum_{l\in\mathcal{L}_{v_r}} p(d; \boldsymbol{\theta}_l) P(l|\boldsymbol{x}, \mathcal{W})$ and $p_{\mathcal{L}_{v_l}}(d|\boldsymbol{x}) = \sum_{l\in\mathcal{L}_{v_l}} p(d; \boldsymbol{\theta}_l) P(l|\boldsymbol{x}, \mathcal{W})$. Intuitively, the gradient of a split node is computed by combining the gradients of all the nodes in the subtree rooted at that split node. Note that this gradient computation can be easily extended for the loss function in (8) as the probability distribution in (4) is just a weighted sum of the distributions defined in (3).

From (11), the gradient $\frac{\partial L(\mathcal{W},\Theta;\boldsymbol{x},d)}{\partial f_v(\boldsymbol{x},\boldsymbol{w}_v)}$ at v can be recursively computed bottom-up by combining the loss gradients from v's children v_l and v_r. Thus, our gradient computation starts at the leaf nodes, and gets propagated to the root in a bottom up fashion. The gradient of loss at a leaf node $l \in \mathcal{L}$ is computed as

$$\frac{p(d; \boldsymbol{\theta}_l) P(l|\boldsymbol{x}, \mathcal{W})}{p_{\mathcal{T}}(d|\boldsymbol{x})}. \tag{12}$$

After computing $\frac{\partial L(\mathcal{W},\Theta;\boldsymbol{x},d)}{\partial f_v(\boldsymbol{x},\boldsymbol{w}_v)}$ at every node v, we proceed with simultaneous training of all CNNs in $\mathcal{T}$.

5.2. Learning the Leaf Distribution

For each leaf, we estimate the probability of the training samples reaching that node $P(l|\boldsymbol{x}, \mathcal{W})$. This probability represents a weight (i.e., relative significance) of training samples collected in the leaf. We use these weights to fit a Gaussian distribution on the *weighted* depth values of training samples that reached the leaf. Specifically, we follow the [26] to empirically estimate the Gaussian parameters $\boldsymbol{\theta}_l = (\mu_l, \sigma_l)$ for the leaf as

$$\mu_l = \frac{1}{\sum\limits_{(\boldsymbol{x}_i,d_i)\in S} P(l|\boldsymbol{x}, \mathcal{W})} \sum_{(\boldsymbol{x}_i,d_i)\in S} P(l|\boldsymbol{x}_i, \mathcal{W}) \cdot d_i$$

$$\sigma_l = \frac{1}{\sum\limits_{(\boldsymbol{x}_i,d_i)\in S} P(l|\boldsymbol{x}_i, \mathcal{W})} \sum_{(\boldsymbol{x}_i,d_i)\in S} P(l|\boldsymbol{x}_i, \mathcal{W})(d_i - \mu_l)^2, \tag{13}$$

where S is the set of training data.

6. CNN Architecture

We observe that as data samples are passed down the tree, the probability of data reaching a split node decreases. Moreover, the estimated depth distribution becomes more reliable as the data samples are becoming increasingly sorted by passing down the tree. Thus, we expect that it becomes easier to learn the split functions as we go down the tree [20].

To address this observation, we adjust the complexity of the CNN architecture along the tree height. The CNN architecture is determined by the number of: 1) convolution + pooling layers, and 2) fully connected perceptron layers. For the top one third of the tree height, we use CNNs with 2 convolution + pooling layers, and 2 fully connected perceptron layers. For the lower one third of the tree height (closer to the leaf nodes), we use CNNs with 2 convolution + pooling layers and 1 fully connected perceptron layer. Finally, for the bottom third of the tree height, we use CNNs with 1 convolution + pooling layer and 1 connected perceptron layer. We experimentally found this architecture suitable for depth prediction. We present the overview of the tree architecture in the Fig. 2. In our experiments, we use NRF with 100 binary trees, each with height 10.

A data sample is defined as a pair of an image window at a pixel location and its corresponding depth label. The feature of the window is computed by CNN. In our experiments, we consider two window sizes: 100×100 and 150×150 for the Make3D and NYUv2 datasets based on the size and complexity of the dataset. This window around the pixel is used to capture context around the pixel. The convolutional outputs of the CNN at a split node are used as the input window of the CNNs at the children split nodes. Thus an input window passes through a series of convolutions along the path from root to the leaf node. Fig. 2a shows that the input window size decreases for the CNNs at split nodes closer to the leaves. This is due to the convolution and pooling that reduce the size of the input window. This allows us to learn multi-scale features through the CNNs in the split nodes along the data path trough the tree. Multi-scale features are important for depth estimation [22, 21].

Unlike [20], we provide only raw color values around the pixel and use convolution kernels to automatically learn the features required to compute the split function. Our tree architecture effectively resembles a deep CNN framework which consists of a set of small CNNs. Our architecture is different from the neural decision forest architecture presented in [11], where a decision forest is used on top of a single deep CNN. In this case all the split nodes correspond to the same output layer of the CNN. We use the output of the parent split node as the input for the CNNs at the child split node.

We use a modified ImageNet initialization of convolution layers by sampling image patches. In our experiments, multiple random initializations produced statistically similar results to those reported in the Sec. 7. Note that the training and inference of the trees can be done in parallel. Inference takes less than 1sec per image and training takes 8-10 hours in our parallel implementation on a standard PC with a nvidia Tesla k80 graphics card.

7. Results

Datasets. We evaluate our approach on two benchmark datasets: the Make3D [22] and the NYUv2 [18]. These datasets are commonly used by the state of the art approaches for evaluating the depth estimation performance. The Make3D dataset consists of 534 images of outdoor scenes, where 400 images are used for training and 134 images are used for testing. Depth is estimated by a laser scanner and the depth ranges from 0 to 80 meters. The NYUv2 dataset consists of 1449 indoor images where the depth is estimated by a Kinect device. On this dataset, we follow the standard split of 795 training images and 654 test images. The depth ranges from 0 to 10 meters for this dataset.

Performance metrics. We use three standard error metrics which are commonly used by the state of the art methods [21, 22, 14, 6] to estimate the accuracy of monocular depth prediction. These errors are defined for each pixel and averaged across all the pixels in an image and all images in a dataset. Lets assume d^* and $\hat{d}$ are the ground-truth and predicted depth for a pixel then the errors are defined as 1) **relative error** (rel): $|d^* - \hat{d}|/d^*$, 2) **root mean squared error** (rms): $\sqrt{\sum(d^* - \hat{d})^2}$, and 3) $\log 10$ **error** ($\log 10$): $|\log d^* - \log \hat{d}|$.

Baselines. We propose the following baselines and the comparisons to the baselines are presented in Tab. 1.

CNN - regression forest (CNN-RF): In this baseline, we use a similar framework proposed by [11]. A single CNN is learned for the forest, and its outputs are treated as the split node for the regression trees of the forest. Thus, split functions in every node share the same CNN parameters through a single deep network. We consider same input windows for the pixels as we use for out experiments. NRF outperforms this baseline which suggests that a set of small CNN performs better than a large network for the small datasets.

NRF without input forwarding for CNNs (NRFw/oF): In this baseline, for each CNN in the split node, we use only RGB input window instead of convolutional outputs from the parent split node. As suggested by [20], the size of input windows for the split nodes is gradually reduced as we go down the tree along its depth. NRF outperforms this baseline, which suggests forwarding the convolutional outputs as the inputs to the lower split nodes can effectively model a better feature learning framework then considering RGB inputs to each split node.

	Make3D			NYU v2		
	rel	log10	rms	rel	log10	rms
CNN-RF	0.361	0.148	15.10	0.35	0.131	1.2
NRFw/oF	0.312	0.128	13.8	0.24	0.09	0.95
NRFw/oN	0.29	0.126	13.7	.22	0.088	0.936
Ours	0.26	0.119	12.40	0.187	0.078	0.744

Table 1. Comparison with the baselines on Make3D and NYUv2 datasets.

	Make3D			NYU v2		
	rel	log10	rms	rel	log10	rms
[22]	0.370	0.187	-	0.349	-	1.214
[1]	0.362	0.168	15.8	-	-	-
[16]	0.338	0.134	12.60	0.335	0.127	1.06
[10]	0.361	0.148	15.10	0.35	0.131	1.2
[12]	0.364	0.148	-	-	-	-
[14]	0.379	0.148	-	-	-	-
[6]	-	-	-	0.215	-	0.907
[15]	0.307	0.125	12.89	0.230	0.095	0.824
[27]	-	-	-	0.305	0.122	1.04
Ours	0.26	0.119	12.40	0.187	0.078	0.744

Table 2. Comparison with the state of the art on Make3D and NYU v2 datasets.

NRF without the neighborhood information (NRFw/oN): In this baseline, we predict the depth of a pixel only based on its appearance without considering the information of its neighboring pixels. Specifically, we use (3) instead of (4) for the depth prediction in a tree. In our full approach, for each pixel, we consider 10 neighboring windows for capturing the neighborhood depth information. The results are shown in the Tab. 1. This proves that the neighboring pixels provide important information for depth prediction. Moreover, considering neighboring information results in smoother depth maps (Fig. 5 and 6).

Comparison with state of the art: A comparison with the state of the art is presented in Tab. 2. For fair comparison, we follow the same experimental setup, and use the same standard metrics as done by the state-of-the-art methods. We achieve better performance on both datasets in terms of standard error metrics.

Robustness against the amount of training data: To evaluate the sensitivity of NRF against the amount of training, we simulate the behavior of our approach on both datasets by gradually reducing the training data. Instead of using the complete training set, we randomly sample a fraction of training data for training. The plots in Fig. 4 show that the variation of the error values (‘rel’ and ‘log 10’) while the amount of training data is gradually reduced. Note that both error metrics increase slowly as the amount of training data is gradually reduced. We believe that this is due to our ensemble architecture of the forest, and accounting for the smoothness of neighboring depths.

Qualitative results: We present the qualitative results on the Make3D and the NYUv2 datasets in Fig. 5 and Fig. 6 respectively. We compare our approach with the baseline which does not consider smoothness is depths of neighboring pixels. We observe that accounting for smoothness produces more accurate depth maps. Recall that the neighborhoods are estimated by bi-lateral filtering. Since the bilateral filter is able to preserve edges, the resulting depth maps tend to respect object boundaries.

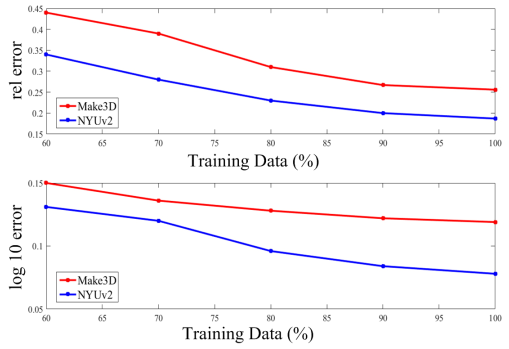

Figure 4. Variation of ‘rel’ and ‘log 10’ errors with respect to the amount of training data on Make3D and NYUv2 datasets.

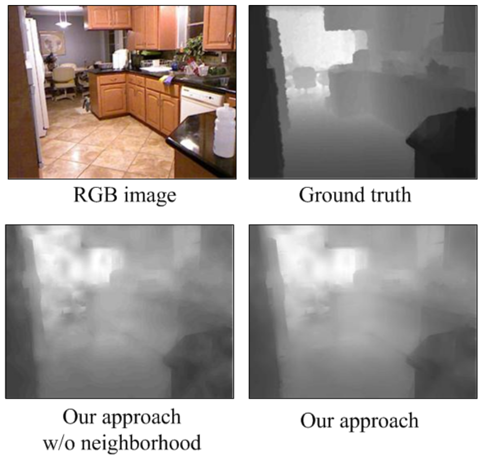

Figure 7. A failure case for an image from the NYUv2 dataset with highly textured regions and object clutter.

Failure case: In Fig. 7, we present a case where our approach fails to produce an accurate depth map for an image in the NYUv2 dataset. As our depth prediction is guided by appearance, regions with high and random texture might create confusion in our prediction. Note that the image is highly textured and cluttered with many different objects. This causes error as shown in Fig. 5 and Fig. 6.

8. Conclusion

We have formulated NRF for the problem of monocular depth estimation. A “shallow” CNN with maximum two convolutional layers is associated with every node in the regression trees of NRF. The convolutional output of a CNN is used as input to the left and right children CNNs in the tree, effectively allowing for deep convolutional processing of data samples. In this way, scanning windows

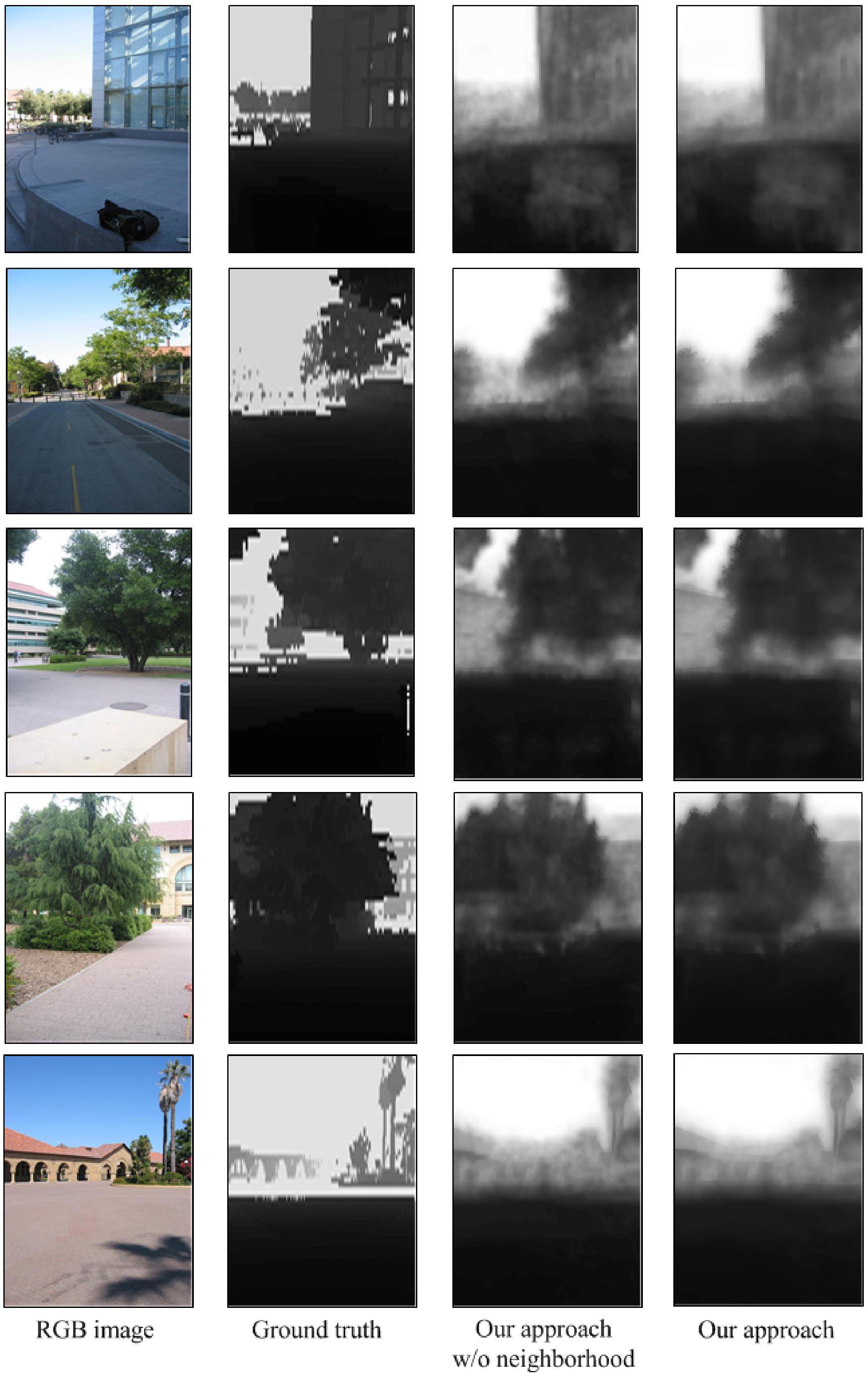

Figure 5. Qualitative results on the Make3D dataset. Note that accounting for smoothness in a neighborhood results in more accurate depth maps which better respect object boundaries.

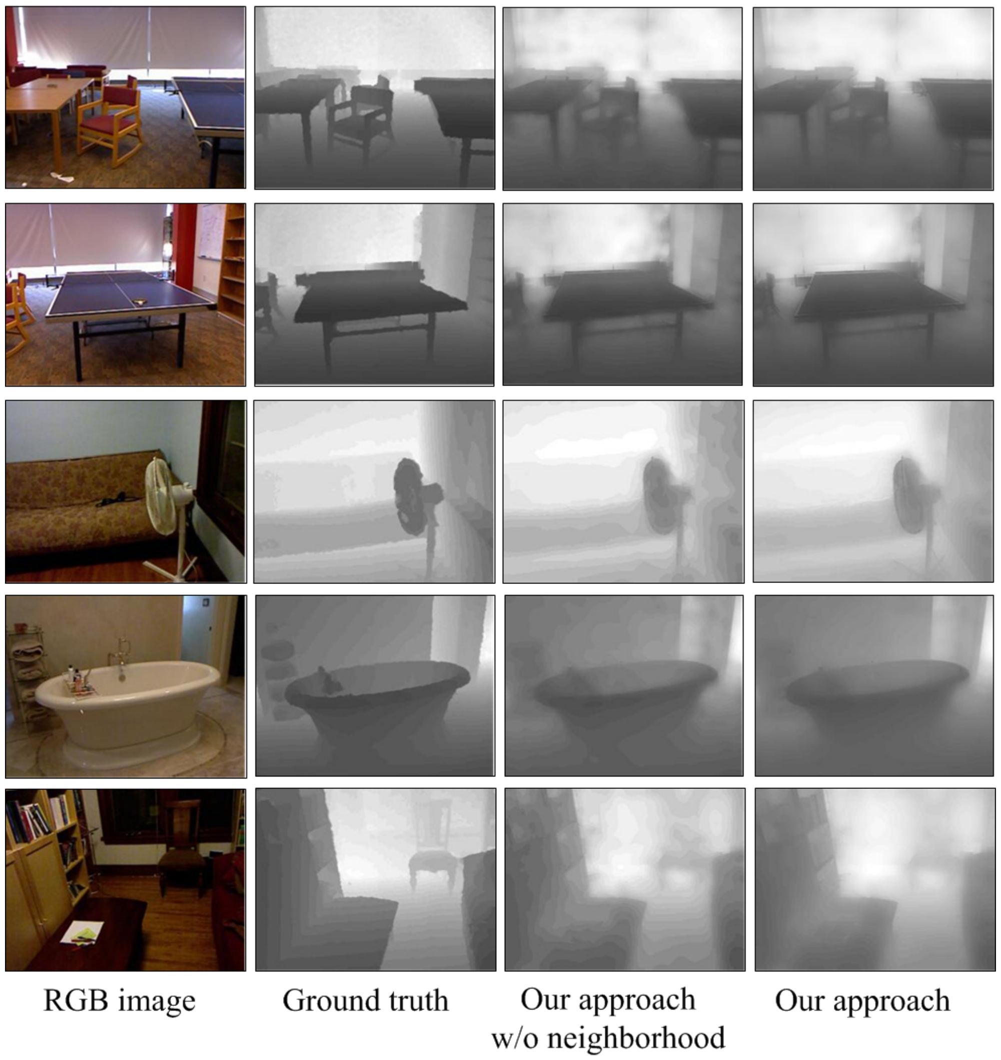

Figure 6. Qualitative results on the NYUv2 dataset. Note that accounting for smoothness in a neighborhood results in more accurate depth maps which better respect object boundaries.

extracted from the image are passed down the tree, until the leaves representing weak depth estimators. We have additionally accounted for smoothness in depths across pixel neighborhoods, which in turn are estimated using bilateral filtering. We have evaluated our approach on two benchmark datasets: Make3D and NYUv2. Both datasets provide a relatively small and unbalanced set of training depth examples. Results demonstrate that NRF is able to robustly address these challenges, and outperform the state of the art, due to: (a) its ensemble architecture consisting of the forest of trees, (b) "shallow" CNN architecture with significantly few parameters than seen in recent work, and (c) accounting for smoothness in depths. Also, our experiments show that NRF's performance gracefully downgrades when the size of the training dataset gradually decreases.

Acknowledgment

This work was supported in part by grant NSF RI 1302700. We would like to thank Jose Roberto Alvarez and Bennett Wilburn from Huawei Media Technologies Lab, CA for fruitful discussions and feedback. A part of the work was developed during Roy Anirban's internship in Huawei Media Technologies Lab, CA.

References

[1] D. Batra and A. Saxena. Learning the right model: Efficient max-margin learning in laplacian CRFs. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2136–2143. IEEE, 2012. 1, 2, 6

[2] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *Computer Vision, 2007.*

ICCV 2007. IEEE 11th International Conference on, pages 1–8. IEEE, 2007. 2

[3] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 1

[4] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla. Segmentation and recognition using structure from motion point clouds. In *Computer Vision–ECCV 2008*, pages 44–57. Springer, 2008. 2

[5] A. Criminisi and J. Shotton. *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media, 2013. 1, 2

[6] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems*, pages 2366–2374, 2014. 1, 2, 5, 6

[7] S. R. Fanello, C. Keskin, S. Izadi, P. Kohli, D. Kim, D. Sweeney, A. Criminisi, J. Shotton, S. B. Kang, and T. Paek. Learning to be a depth camera for close-range human capture and interaction. *ACM Transactions on Graphics (TOG)*, 33(4):86, 2014. 2

[8] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1915–1929, 2013.

[9] A. Kar, S. Tulsiani, J. Carreira, and J. Malik. Category-specific object reconstruction from a single image. *CVPR*, 2015. 2

[10] K. Karsch, C. Liu, and S. B. Kang. Depth transfer: Depth extraction from video using non-parametric sampling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(11):2144–2158, 2014. 6

[11] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulo. Deep neural decision forests. In *ICCV*, 2015. 2, 3, 4, 5

[12] M. Lam, J. R. Doppa, S. Todorovic, and T. G. Dietterich. HC-search for structured prediction in computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4923–4932, 2015. 1, 2, 6

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. 4

[14] B. Liu, S. Gould, and D. Koller. Single image depth estimation from predicted semantic labels. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1253–1260. IEEE, 2010. 1, 2, 5, 6

[15] F. Liu, C. Shen, and G. Lin. Deep convolutional neural fields for depth estimation from a single image. 2015. 1, 2, 6

[16] M. Liu, M. Salzmann, and X. He. Discrete-continuous depth estimation from a single image. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 716–723. IEEE, 2014. 6

[17] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015.

[18] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor segmentation and support inference from RGBD images. In *ECCV*, 2012. 1, 2, 5

[19] F. Porikli. Constant time O(1) bilateral filtering. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 4

[20] S. Rota Bulo and P. Kontschieder. Neural decision forests for semantic image labelling. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 81–88. IEEE, 2014. 2, 3, 4, 5

[21] A. Saxena, S. H. Chung, and A. Y. Ng. Learning depth from single monocular images. In *Advances in Neural Information Processing Systems*, pages 1161–1168, 2005. 1, 2, 5

[22] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3D scene structure from a single still image. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):824–840, 2009. 1, 2, 5, 6

[23] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, et al. Efficient human pose estimation from single depth images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(12):2821–2840, 2013. 2

[24] H. Su, Q. Huang, N. J. Mitra, Y. Li, and L. Guibas. Estimating image depth using shape collections. *ACM Transactions on Graphics (TOG)*, 33(4):37, 2014. 2

[25] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998. 2, 3

[26] P. Vincent, Y. Bengio, et al. Locally weighted full covariance gaussian density estimation. Technical report, Technical report 1240, 2003. 4

[27] W. Zhuo, M. Salzmann, X. He, and M. Liu. Indoor scene structure analysis for single image depth estimation. In *CVPR*, 2015. 2, 6