CSPN++: Learning Context and Resource Aware Convolutional Spatial Propagation Networks for Depth Completion

Xinjing Cheng, Peng Wang,* Chenye Guan, Ruigang Yang
Robotics and Auto-driving Lab (RAL), Baidu Research
{chengxinjing, wangpeng54, guanchenye, yangruigang}@baidu.com

Abstract

Depth Completion deals with the problem of converting a sparse depth map to a dense one, given the corresponding color image. Convolutional spatial propagation network (CSPN) is one of the state-of-the-art (SoTA) methods of depth completion, which recovers structural details of the scene. In this paper, we propose CSPN++, which further improves its effectiveness and efficiency by learning adaptive convolutional kernel sizes and the number of iterations for the propagation, thus the context and computational resource needed at each pixel could be dynamically assigned upon requests. Specifically, we formulate the learning of the two hyper-parameters as an architecture selection problem where various configurations of kernel sizes and numbers of iterations are first defined, and then a set of soft weighting parameters are trained to either properly assemble or select from the pre-defined configurations at each pixel. In our experiments, we find weighted assembling can lead to significant accuracy improvements, which we referred to as "context-aware CSPN", while weighted selection, "resource-aware CSPN" can reduce the computational resource significantly with similar or better accuracy. Besides, the resource needed for CSPN++ can be adjusted w.r.t. the computational budget automatically. Finally, to avoid the side effects of noise or inaccurate sparse depths, we embed a gated network inside CSPN++, which further improves the performance. We demonstrate the effectiveness of CSPN++ on the KITTI depth completion benchmark, where it significantly improves over CSPN and other SoTA methods [1].

Introduction

Image guided depth completion, or depth completion for short in this paper, is the task of converting a sparse depth map from devices such as LiDAR or algorithms such as structure-from-motion (SfM) and simultaneously localization and mapping (SLAM) to a per-pixel dense depth map with the help of reference images. The technique has a wide range of applications for the perception of indoor/outdoor moving robots such as self-driving vehicles, home/indoor robots, or applications such as augmented reality.

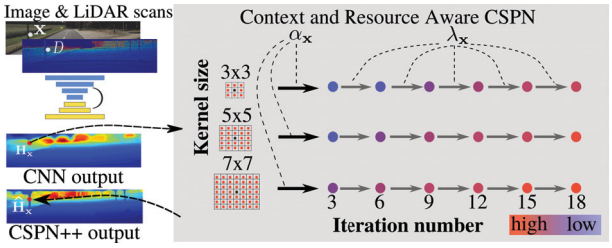

Figure 1: Output assembling or selection over an unrolled CSPN. The color of each dot indicates the computational resources need at the point, where blue indicates low resource usage while red indicates high resource usage.

One of the state-of-the-art (SoTA) methods for this task is CSPN (Cheng, Wang, and Yang 2018a), which is an efficient local linear propagation model with learned affinity from a convolutional neural network (CNN). CSPN claims three important properties should be considered for the depth completion task, 1) depth preservation, where the depth value at sparse points should be maintained, 2) structure alignment, where the detailed structures, such as edges and object boundaries in estimated depth map, should be aligned with the given image, and 3) transition smoothness, where the depth transition between sparse points and their neighborhoods should be smooth.

In real applications, depths from devices like LiDAR, or algorithms such as SfM or SLAM could be noisy (Van Gansbeke et al. 2019) due to system or environmental errors. Datasets like KITTI adopt stereo and multiple frames to compensate the errors for evaluation. Here in this paper, we do not assume that the sparse depth map is the ground truth, rather, we consider that it may include errors as well. So the depth value at sparse points should be conditionally maintained with respect to its accuracy. Secondly, all pixels are considered equally in CSPN, while intuitively the pixels at geometrical edges and object boundaries should be more focused for structure alignment and transition smoothness. Therefore, in CSPN++, we propose to find a proper propa-

*denotes the corresponding author

[1]http://www.cvlibs.net/datasets/kitti/eval_depth.php?benchmark=depth_completion

gation context, to further improve the performance of depth completion.

To be specific, as illustrated in Fig. 1, in CSPN++, numerous configurations of convolutional kernel size and number of iteration are first defined for each pixel $\mathbf{x}$, then we utilize $\alpha_{\mathbf{x}}$ to weight different proposals of kernel size, and use $\lambda_{\mathbf{x}}$ to weight outputs after different iterations. Based on these hyper-parameters, we induce context-aware and resource-aware variants for CSPN++. In context-aware CSPN (CA-CSPN), we propose to assemble the outputs, and CSPN++ is structurally similar to networks such as InceptionNet (Szegedy et al. 2016) or DenseNet (Huang et al. 2017a), where gradient from the final output can be directly back-propagated to earlier propagation stages. We find the model learns stronger representation yielding significant performance boost comparing to CSPN.

In resource-aware CSPN (RA-CSPN), CSPN++ sequentially selects one convolutional kernel and one number of iteration for each pixel by minimizing the computational resource usage, where the learned computational resource allocation speeds up CSPN significantly ($2\times$ $\sim 5\times$ in our experiments) with improvements of accuracy. In addition, RA-CSPN can also be automatically adapted to a provided computational budget with the awareness of accuracy through a budget rounding operation during the training and inference.

In summary, our contribution lies in two aspects:

1. Base on the observation of error sparse depths, we propose a gate network to guide the depth preservation, and make the output more robust to noisy sparse depths.
2. We propose an effective method to adapt the kernel sizes and iteration number for each pixel with respect to image content for CSPN, which induces two variants, named as context-aware and resource-aware CSPN. The former significantly improves its performance, and the later speeds up the algorithm and makes the CSPN++ adapt to computational budgets.

Related Work

Depth estimation, completion, enhancement/refinement and models for dense prediction with dynamic context and compression have been center problems in computer vision and robotics for a long time. Here we summarize those works in several aspects without enumerating them all due to space limits, and we majorly clarify their core relationships with CSPN++ proposed in this paper.

Depth Completion. The task of depth completion (Uhrig et al. 2017) recently attracts lots of interests in robotics due to its wide application for enhancing 3D perception for robotics (Liao et al. 2017). The provided depths are usually from LiDAR, SfM or SLAM, yielding a map with valid depth partially available in some of the pixels. Within this field, some works directly convert sparse 3D points to dense ones without image guidance (Uhrig et al. 2017), which produce impressive results with deep learning. However, conventionally, jointly considering the structures from reference images for guiding depth completion/enhancement (Liu and Gong 2013) yields better results. With the rising the deep learning for depth estimation from a single image (Wang et al. 2016), researchers adopt similar strategies to image guided depth completion. For example, (Ma and Karaman 2018) propose to treat sparse depth map as an additional input to a ResNet based depth predictor (Laina et al. 2016), producing superior results than the depth output from CNN with solely image inputs. Later works are further proposed by focusing on improving the efficiency (Ku, Harakeh, and Waslander 2018), separately modeling the features from image and sparse depths (Tang et al. 2019), recovering the structural details of depth maps (Cheng, Wang, and Yang 2018a), combing with multi-level CRF (Xu et al. 2018) or adopting auxiliary training losses using normal (Zhang and Funkhouser 2018) or 3D representation (Chen et al. 2019) from self-supervised learning strategy (Ma, Cavalheiro, and Karaman 2019).

Among all of these works, we treat CSPN (Cheng, Wang, and Yang 2018a) as our baseline strategy due to its clear motivation and good theoretical guarantee in the stability of training and inference, and our resulted CSPN++ provides a significant boost both on its effectiveness and efficiency.

Context Aware Architectures. Assembling multiple contexts inside a network for dense predictions has been an effective component for recognition tasks in computer vision. In our perspective, the assembling strategies could be horizontal or vertical. Horizontal strategies assemble outputs from multiple branches in a single layer of a network, which include modules of Inception/Xception (Szegedy et al. 2016), pyramid spatial pooling (PSP) (Zhao et al. 2016), atrous spatial pyramid pooling (ASPP) (Chen et al. 2017), and vertical strategies assemble outputs from different layers include modules of HighwayNet (Srivastava, Greff, and Schmidhuber 2015), DenseNet (Huang et al. 2017a), etc. Some recent works combine these two strategies together such as networks of HRNet (Sun et al. 2019) or models of DenseASPP (Yang et al. 2018). Most recently, to make the context to be better conditioned on each pixel or provided image, attention mechanism with the cost of additional computation is further induced inside the network for context selection such as skipnet (Wang et al. 2018b), non-local networks (Wang et al. 2018a) or context deformation such as spatial transformer networks (Jaderberg et al. 2015) or deformable networks (Zhu et al. 2019).

In the field of depth completion, (Cheng, Wang, and Yang 2018b) propose the atrous convolutional spatial pyramid fusion (ACSF) module which extends ASPP by additionally adding affinity for each pixel, yielding stronger performance, which can be treated as a case of combining horizontal assembling with attention from affinity values. In our case, CA-CSPN of CSPN++ extends context assembling idea into CSPN with both horizontal and vertical strategies via attention. Horizontally, it assembles multiple kernel sizes, and vertically it assembles the outputs from different iteration stages as illustrated in Fig. 1. Here we would like to note that although mathematically in forward process, performing one step CA-CSPN with kernels of 7×7, 5×5, 3×3 together is equivalent to performing CSPN with a single 7×7 kernel since the full process are linear, the backward learn-

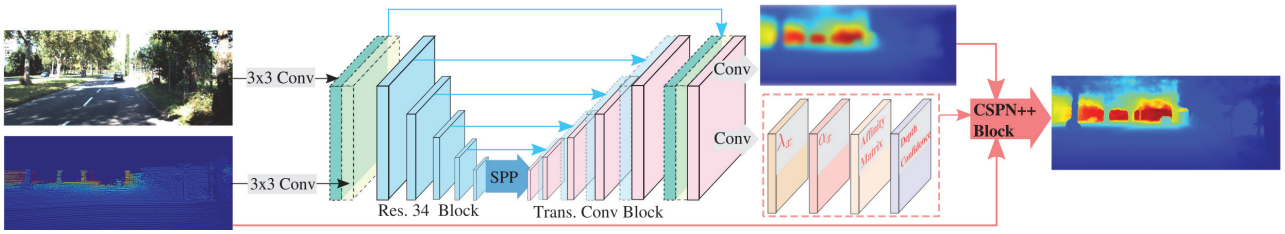

Figure 2: Framework of our networks for depth completion with resource and context aware CSPN (best view in color). At the end of the network, we generate the depth confidence for each sparse point, affinity matrix for CSPN, and weighting variables $\alpha_{\mathbf{x}}$ and $\lambda_{\mathbf{x}}$ for model assembling and selection.

ing process is different due to the auxiliary parameters ($\alpha_{\mathbf{x}}$, $\lambda_{\mathbf{x}}$), and our results are significantly better.

Resource Aware Inference. In addition, the dynamic context intuition can be also applied for efficient prediction by stopping the computation after obtained a proper context, which is also known as adaptive inference. Specifically, the relevant strategies have been adopted in image classification such as a multi-scale dense network (MSDNet) (Huang et al. 2018), object detection such as trade-off balancing (Huang et al. 2017b) or semantic segmentation such as regional convolution network (RCN) treating each pixel differently (Li et al. 2017).

In RA-CSPN of CSPN++, we first embed such an idea in depth completion, and adopt functionality of RCN in CSPN for efficient inference. To minimize the computation, each pixel chooses one kernel size and then one number of iterations sequentially from the proposed configurations. Besides, we can easily add a provided computation budget, such as latency or memory constraints, into our optimization target, which could be back-propagated for operation selection similar to resource constraint architecture search algorithms (Cai, Zhu, and Han 2019).

Preliminaries

To make the paper self-contained, we first briefly review CSPN (Cheng, Wang, and Yang 2018b), and then demonstrate how we extend it with context and resource awareness. Given one depth map $D_o \in \mathbb{R}^{m\times n}$ that is output from a network taken input as an image $\mathbf{I} \in \mathbb{R}^{m\times n}$, CSPN updates the depth map to a new depth map D_n. Without loss of generality, we follow their formulation by embedding depth to a hidden representation $\mathbf{H} \in \mathbb{R}^{m\times n\times c}$, and the updating equation for one step propagation can be written as,

$$\begin{aligned}\mathbf{H}_{\mathbf{x},t+1} &= \phi_{CSP}(\mathbf{H}_{\mathbf{x},t}, \mathbf{H}_{\mathbf{x},0}|k) \\ &= \kappa_{\mathbf{x}}(\mathbf{x}) \odot \mathbf{H}_{\mathbf{x},0} + \sum_{\mathbf{x}_n\in\mathcal{N}_k} \kappa_{\mathbf{x}}(\mathbf{x}_n) \odot \mathbf{H}_{\mathbf{x}_n,t}\end{aligned}$$

where,

$$\begin{aligned}\kappa_{\mathbf{x}}(\mathbf{x}_n) &= \hat{\kappa}_{\mathbf{x}}(\mathbf{x}_n) / \sum_{\mathbf{x}_n\in\mathcal{N}} |\hat{\kappa}_{\mathbf{x}}(\mathbf{x}_n)|, \\ \kappa_{\mathbf{x}}(\mathbf{x}) &= \mathbf{1} - \textstyle\sum_{\mathbf{x}_n\in\mathcal{N}} \kappa_{\mathbf{x}}(\mathbf{x}_n)\end{aligned} \tag{1}$$

where $\phi_{CSP}()$ represents one step CSPN given a predefined size of convolutional kernel k. $\mathcal{N}_k$ is the neighborhood pixels in a $k\times k$ kernel, and the affinities output from a network $\hat{\kappa}_{\mathbf{x}}()$ are properly normalized which guarantees the stability of the module. The whole process will iterate N times to obtain the final results. Here, k, N needs to be tuned in the experiments, which impacts the final performance significantly in their paper.

For depth completion, CSPN preserves the depth value at those valid pixels in a sparse depth map D_s by adding a replacement operation at the end of each step. Formally, let $\mathbf{H}^s$ to be the corresponding embedding for D_s, the replacement step after performing Eq. (1) is,

$$\mathbf{H}_{\mathbf{x},t+1} = (1 - m_{\mathbf{x}})\mathbf{H}_{\mathbf{x},t+1} + m_{\mathbf{x}}\mathbf{H}^s_{\mathbf{x}}, \tag{2}$$

where $m_{\mathbf{x}} = \mathbb{I}(d^s_{\mathbf{x}} > 0)$ is an indicator for the validity of sparse depth at $\mathbf{x}$.

Context and Resource Aware CSPN

In this section, we elaborate how CSPN++ enhances CSPN by learning a proper configuration for each pixel by introducing additional parameters to predict. Specifically, predicting $\alpha_{\mathbf{x}} = \{\alpha_{\mathbf{x}}(k)\}$ for weighting various convolutional kernel size and $\lambda_{\mathbf{x}} = \{\lambda_{\mathbf{x}}(k,t)\}$ for weighting different number of iterations given a kernel size k. As shown in Fig. 2, both variables are image content dependent, and are predicted from a shared backbone with CSPN affinity and estimated depths.

Context-Aware CSPN

Given the provided $\alpha_{\mathbf{x}}$ and $\lambda_{\mathbf{x}}$, context-aware CSPN (CA-CSPN) first assembles the results from different steps. Formally, the propagation from t to $t+1$ could be written as,

$$\begin{aligned}\mathbf{H}^+_{\mathbf{x},t+1,k} &= \lambda_{\mathbf{x}}(k,t+1) * \phi_{CSP}(\mathbf{H}_t, \mathbf{H}_0|\mathbf{x},k) + \mathbf{H}^+_{\mathbf{x},t,k} \\ \lambda_{\mathbf{x}}(k,t) &= \sigma(\hat{\lambda}_{\mathbf{x}}(k,t)) / \textstyle\sum_{t\in\{1\cdot N\}} \sigma(\hat{\lambda}_{\mathbf{x}}(k,t))\end{aligned} \tag{3}$$

where, $\sigma()$ is the sigmoid function, and $\hat{\lambda}_{\mathbf{x}}$ is the outputs from the network. In the process, $\mathbf{H}^+_{\mathbf{x},t+1,k}$ progressively aggregates the output from each step of CSPN based on $\lambda_{\mathbf{x}}$. Finally, we assemble different outputs from various kernels

after N iterations,

$$\mathbf{H}^{+}_{\mathbf{x},N} = \sum\nolimits_{k\in\mathcal{K}} \alpha_{\mathbf{x}}(k)\mathbf{H}^{+}_{\mathbf{x},N,k}$$
$$\alpha_{\mathbf{x}}(k) = \sigma(\hat{\alpha}_{\mathbf{x}}(k))/\sum\nolimits_{k\in\mathcal{K}} \sigma(\hat{\alpha}_{\mathbf{x}}(k)) \quad (4)$$

Here, both $\alpha_{\mathbf{x}}$ and $\lambda_{\mathbf{x}}$ are properly normalized with their l_1 norm, so that our output $\mathbf{H}^{+}_{\mathbf{x},N}$ maintains the stabilization property of CSPN for training and inference.

When there are sparse points available, CSPN++ adopts a confidence variable $g_{\mathbf{x}}$ predicted at each valid depth in the sparse depth map, which is output from the shared backbone in our framework (Fig. 2). Therefore, the replacement step for CSPN++ can be modified accordingly,

$$\mathbf{H}^{+}_{\mathbf{x},t+1} = (1-g_{\mathbf{x}})\mathbf{H}^{+}_{\mathbf{x},t+1} + g_{\mathbf{x}}\mathbf{H}^{s}_{\mathbf{x}}, \quad (5)$$

where $g_{\mathbf{x}} = \mathbb{I}(d^{s}_{\mathbf{x}} > 0)\sigma(\hat{g}_{\mathbf{x}})$, where $\hat{g}_{\mathbf{x}}$ is predicted from a network after a convolutional layer.

Complexity and computational resource analysis. From CSPN, we know that theoretically with sufficient amount of GPU cores and large memory storage, the overall complexity for CSPN with a kernel size of k and iteration N is $O(\log(k^2)N)$. In CA-CSPN, with induced K convolutional kernels, the computation complexity is $O(\log(k^2_{max})N)$, where k_{max} is the maximum kernel size since all branch can be performed simultaneously.

However, in the real application, the expected computational resource is limited and latency of memory request with large convolutional kernel could be time consuming. Therefore, we need to utilize a better metric for estimating the cost. Here, we adopt the popularly used memory cost and Mult-Adds/FLOPs as an indicator of latency or computational resource usage on a device. Specifically, based on the CUDA implementation of convolution with `im2col`, performing CSPN with a kernel k would require memory cost of $O(k^2)$, and FLOPs of $O(Nk^2)$, given a single feature block with a size of $h \times w \times c$. In summary, given K kernels, the latency from big O estimation for CA-CSPN would be $O(Nk^2_{max})$. Finally, we would like to note that the memory and computational configuration varies with given devices, so does the latency estimation. A better strategy would be directly testing over the target device as proposed in (Cai, Zhu, and Han 2019). Here, we just provide a reasonable estimation with the commonly used GPU.

Network architectures. As illustrated in Fig. 2, for the backbone network, we adopt the same ResNet-34 structure proposed in (Ma, Cavalheiro, and Karaman 2019). The only modification is at the end of the network, it outputs the per-pixel estimation of assembling parameters $\lambda_{\mathbf{x}}$, $\alpha_{\mathbf{x}}$, noisy guidance for replacement $g_{\mathbf{x}}$ and affinity matrix $\kappa_{\mathbf{x}}$ using a convolutional layer with a 3×3 kernel. For handling the affinity values for various propagation kernels, we use a shared affinity matrix since the affinity between different pixels should be irrelevant to the context of propagation, which saves the memory cost inside the network.

Training context-aware CSPN. Given the proposed architecture, based on our computational resource analysis w.r.t. latency, we add additional regularization term inside the general optimization target, which minimizes the expected

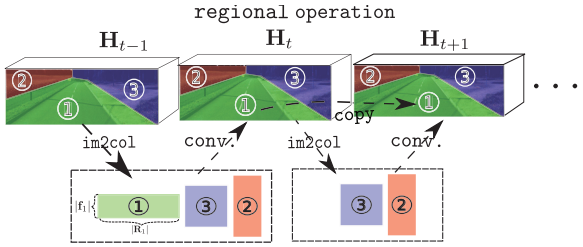

Figure 3: The proposed regional `im2col` and `conv.` operation for efficient testing. Here, let the regions of green (①), red (②) and blue (③) have kernel size of 3, 7, 5 and iteration number of t, t+1, t+1 respectively. We convert each region to a matrix of $|\mathbf{f}_i| \times |R_i|$ for performing parallel `conv.` through `im2col`, where $|\mathbf{f}_i| = k_i \times k_i \times c$ is the feature dimension, and $|R_i|$ is the number of pixels in the corresponding region. If pixels belong to a region does not need propagation (*i.e.* region ① at time step t as illustrated), we direct `copy` its feature to next step.

computational cost c by treating $\alpha_{\mathbf{x}}, \lambda_{\mathbf{x}}$ as probabilities of configuration selection. It is shown to be effective in improving the final performance in our experiments. Formally, the overall target for training CA-CSPN can be written as,

$$\min_{\mathbf{w}} \mathcal{L}_{train}(D, D^*|\mathbf{w}) + \eta_1\|\mathbf{w}\|^2_2 + \eta_2\mathbf{E}(c|\{\alpha_{\mathbf{x}}, \lambda_{\mathbf{x}}\})$$
$$\mathcal{L}_{train}(D, D^*|\mathbf{w}) = \|D - D^*\|^2_2$$
$$\mathbf{E}(c|\{\alpha_{\mathbf{x}}, \lambda_{\mathbf{x}}\}) = \frac{1}{hw}\sum\nolimits_{\mathbf{x}} \mathbf{E}(c_{\mathbf{x}}|\alpha_{\mathbf{x}}, \lambda_{\mathbf{x}}) \quad (6)$$
$$\mathbf{E}(c_{\mathbf{x}}|\alpha_{\mathbf{x}}, \lambda_{\mathbf{x}}) = \frac{1}{(Nk^2_{max})}\sum\nolimits_{k,t} \lambda_{\mathbf{x}}(k,t)\alpha_{\mathbf{x}}(k)tk^2$$

where $\mathbf{w}$ is the network parameters, and $\|\mathbf{w}\|^2_2$ is weight decay regularization. $\mathbf{E}(c|\cdot)$ is the expected computational cost given the assembling variables based on our analysis. h, w are height and width of the feature respectively. D and D^* is the output depth map from CA-CSPN and ground truth depth map correspondingly. Here, our system can be trained end-to-end.

Resource Aware Configuration

As introduced in our complexity analysis, CSPN with large kernel size and long time propagation is time consuming. Therefore, to accelerate it, we further propose resource-aware CSPN (RA-CSPN), which selects the best kernel size and number of iteration for each pixel based on the estimated $\alpha_{\mathbf{x}}, \lambda_{\mathbf{x}}$. Formally, its propagation step can be written as,

$$\mathbf{H}_{\mathbf{x},t+1} = \phi_{CSP}(\mathbf{H}_t, \mathbf{H}_0|\mathbf{x}, k^*)$$
$$\text{where } k^* = \arg\max_k \alpha_{\mathbf{x}}(k), t \leqslant \arg\max_t \lambda_{\mathbf{x}}(k,t) \quad (7)$$

Here each pixel is treated differently by selecting a best learned configuration, and we follow the same process of replacement as Eq. (2) for handling depth completion.

Computational resource analysis. Given the selected configuration of convolutional kernel and number of iteration at each pixel, the latency estimation for each image that we proposed in Sec. is changed to $O(\hat{k}^2\hat{t})$, where $\hat{k} = \frac{1}{hw}\sum_{\mathbf{x}} k^*_{\mathbf{x}}$ and $\hat{t} = \frac{1}{hw}\sum_{\mathbf{x}} t^*_{\mathbf{x}}$ are the average iteration step and kernel size in the image respectively. Both of the numbers are guaranteed to be smaller than the maximum number of iteration N and kernel size k_{max}.

Training RA-CSPN. In our case, training RA-CSPN does not need to modify the multi-branch architecture shown in Fig. 1, but switches from the weighted average assembling as described in Eq. (3) and Eq. (4) to max selection that only one path is adopted for each pixel. In addition, we need modify our loss function in Eq. (6) by changing the expected computational cost as,

$$\mathbf{E}(c_{\mathbf{x}}|\alpha_{\mathbf{x}}, \lambda_{\mathbf{x}}) = (k^*_{\mathbf{x}})^2 t^*_{\mathbf{x}}/(Nk^2_{max})$$
$$\text{where } k^* = \arg\max_k \alpha_{\mathbf{x}}(k), t^* = \arg\max_t \lambda_{\mathbf{x}}(k^*, t) \quad (8)$$

In practice, to implement configuration selection, we can reuse the same training pipeline as CA-CSPN via converting the obtained soft weighting values in $\alpha_{\mathbf{x}}$ and $\lambda_{\mathbf{x}}$ to one-hot representation through an `argmax` operation.

Efficient testing. Practically, there are two issues we need to handle when making the algorithm efficient at testing: 1) how to perform different convolution simultaneously at different pixels, and 2) how to continue the propagation for pixels whose neighborhood pixels stop their diffusion/propagation process. To handle these issues, we follow the idea of regional convolution (Li et al. 2017).

Specifically, as shown in Fig. 3, to tackle the first one, we group pixels to multiple regions based on our predicted kernel size, and prepare corresponding matrix before convolution for each group using region-wise `im2col`. Then, the generated matrix can be processed simultaneously at each pixel using region-wise convolution. To tackle the second issue, when the propagation of one pixel $\mathbf{x}$ stops at time step t, we directly copy the feature of $\mathbf{x}$ to the next step $t+1$ for computing convolution at later stages. In summary, RA-CSPN can be performed in a single forward pass with less resource usage.

Learning with provided computational budget. Finally, in real applications, rather than providing an optimal computational resource, usually there is a hard constraint for a deployed model, either the memory or latency of inference. Thanks to the adaptive resource usage of CSPN++, we are able to directly put the required budget into our optimization target during training. Formally, given a target memory cost C_m and a latency cost C_l for resource-aware CSPN, our optimization target in Eq. (6) could be modified as,

$$\min_{\mathbf{w}} \mathcal{L}_{train}(D, D^*|\mathbf{w}) + \eta_1\|\mathbf{w}\|_2^2 + \eta_2\mathbf{E}(c|\{\alpha_{\mathbf{x}}, \lambda_{\mathbf{x}}\})$$
$$\text{s.t. } \mathbf{E}(cm|\{\alpha_{\mathbf{x}}, \lambda_{\mathbf{x}}\}) \leqslant C_m, \mathbf{E}(c|\{\alpha_{\mathbf{x}}, \lambda_{\mathbf{x}}\}) \leqslant C_l \quad (9)$$

where $\mathbf{E}(cm|\{\cdot,\cdot\}) = \frac{1}{hwk^2_{max}}\sum_{\mathbf{x}}(k^*_{\mathbf{x}})^2$ is the expected memory cost, and $\mathbf{E}(c|\{\cdot,\cdot\})$ is the expected latency cost defined in Eq. (8). The two constraints can be added to our target easily with Lagrange multiplier. Formally, our optimization target with resource budges is,

$$\min_{\mathbf{w}} \mathcal{L}_{train}(D, D^*|\mathbf{w}) + \eta_1\|\mathbf{w}\|_2^2 + \quad (10)$$
$$\eta_2'[\mathbf{E}(c|\{\cdot,\cdot\} - C_l]_+ + \eta_3[\mathbf{E}(cm|\{\cdot,\cdot\}) - C_m]_+$$

where the hinge loss $[x]_+ = \max(x, 0)$ is adopted as our surrogate function for satisfying the constraints.

Last but not the least, since our primal problem, *i.e.* optimization with deep neural network, is highly non-convex, thus during training, there is no guarantee that all samples will satisfy the constraints. In addition, during testing, the predicted configuration might also violate the given constraints, *e.g.* $\mathbf{E}(c|\{\cdot,\cdot\}) - C_l > 0$. Therefore, for these cases, we propose a resource rounding strategy to hard constraint its overall computation within the budgets. Specifically, we calculate the average cost at each pixel, and for the pixels violating the cost, as illustrated in Fig. 1, we are are able to find the Pareto optimal frontier (Mock 2011) that satisfying the constraint, and we pick the one with largest iteration since it obtains the largest reception field.

Experiments

In this section, we will first introduce the dataset, metrics and our implementation details. Then, extensive ablation study of CSPN++ is conducted on the validation set to verify our insight of each proposed components. Finally, we provide qualitative comparison of CSPN++ versus other SoTA method on testing set.

Experimental setup

KITTI Depth Completion dataset. The KITTI Depth Completion benchmark is a large self-driving real-world dataset with street views from a driving vehicle. It consists 86k training, 7k validation and 1k testing depth maps with corresponding raw LiDAR scans and reference images. We use the official 1k validation images in as our validation set while merge the remained images to training set. The sparse depth maps are obtained by projecting the raw LiDAR points through the view of camera, and the ground truth dense depth maps are generated by first projecting the accumulated LiDAR scans of multiple timestamps, and then removing outliers depths from occlusion and moving objects through comparing with stereo depths from image pairs.

NYU v2 dataset. The NYU-Depth-v2 dataset consists of RGB and depth images collected from 464 different indoor scenes. We use the official split of data, where 249 scenes are used for training and we sample 50K images out of the training set with the same manner as (Cheng, Wang, and Yang 2018a). For testing, following the standard setting (Cheng, Wang, and Yang 2018a), the small labeled test set with 654 images is used the final performance. The original image of size 640×480 are first downsampled to half and then center-cropped, producing a network input size of 304×228.

Metrics. We adopt error metrics same as KITTI depth completion benchmark, including root mean square error (RMSE), mean abosolute error (MAE), inverse RMSE (iRMSE) and inverse MAE (iMAE), where inverse indicates inverse depth representation, *i.e.*converting $d_{\mathbf{x}}$ to $1.0/d_{\mathbf{x}}$.

Table 1: Ablation Study on KITTI Depth Completion validation dataset. 'GR' stands for guided replacement.'LR' stands for latency regularization for the model. 'CPSN++' is our proposed strategies.

Method	SPP	CSPN configuration			GR	LR	Results (Lower the better)	
		Normal	assemble kernel	assemble iter.			RMSE(mm)	MAE(mm)
(Ma, Cavalheiro, and Karaman 2019)							799.08	265.98
(Ma, Cavalheiro, and Karaman 2019)	✓						788.23	247.55
CSPN	✓	✓					765.78	213.,54
CSPN	✓	✓			✓		756.27	215.21
CA-CSPN	✓		✓		✓		732.46	210.61
CA-CSPN	✓		✓	✓	✓		732.34	209.20
CA-CSPN	✓		✓	✓	✓	✓	**725.43**	**207.88**

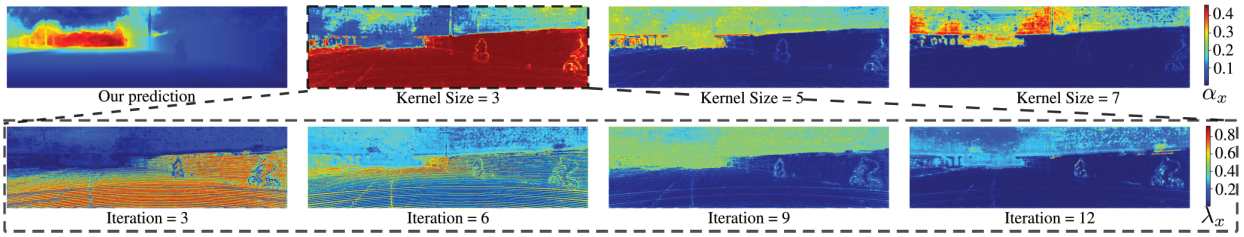

Figure 4: Framework of our networks for depth completion with resource and context aware CSPN(best view in color).

Implementation details. For kitti dataset, we train our network with four NVIDIA Tesla P40, and use batchsize of 8. In all our experiments, we adopt kernel sizes of 3×3, 5×5 and 7×7, and sample outputs after $3, 6, 9, 12$ times of propagation. All our models are trained with Adam optimizer with $\beta_1 = 0.9, \beta_2 = 0.999$. The learning rate start from 10^{-5} and reduce by half for every 5 epochs. Here, for training context-aware CSPN in Eq. (6), the parameter for weight decay, *i.e.* η_1, is set to 0.0005, and the parameter for resource regularization, *i.e.* η_2 is set to 0.1. For training resource-aware CSPN in Eq. (8), we set $\eta_2' = 1.0$ and $\eta_3 = 1.0$. All our parameters are induced for balancing value scale of different losses without exhaustively tuning. While training on NYU dataset, we keep the same configuration with CSPN (Cheng, Wang, and Yang 2018a), and adopt same kernel and iteration configuration with kitti dataset.

Ablation studies

Ablation study of context-aware CSPN (CA-CSPN). Here, we conduct experiments to verify each module adopted in our framework, including our baselines, *i.e.* CSPN with spatial pyramid pooling(SPP), and our newly proposed modules in context-aware CSPN. Specifically, to make the validation efficient, we only train each network 10 epochs to obtain its results. For SPP, we adopt pooling sizes of $12, 6, 4, 2$ and for CSPN, we use the kernel size of 7×7 and set the number of iteration as 12. As shown in Tab 1, by adding SPP and CSPN module to the baseline from (Ma, Cavalheiro, and Karaman 2019), we can significantly reduce the depth error due to the induced pyramid context in SPP and refined structure with CSPN. With additional confidence guided replacement(GR) (Eq. (5)), our module better handles the noisy sparse depths, and the RMSE is significantly reduced from 765.78 to 756.27. Then, at rows with 'assemble kernel', we add the component of learning to horizontally assemble predictions from different kernel size via the learned $\alpha_{\mathbf{x}}$. It further reduce the error from 756.27 to 732.46. At rows with 'assemble iter.', we include the component of learning to vertically assemble outputs after different iterations via the learned $\lambda_{\mathbf{x}}$. Finally, at rows with 'LR', we add our proposed latency regularization term (Eq. (6)) into the training losses, yielding the best results of our context-aware CSPN.

In Fig. 4, we visualize the learned configurations of $\alpha_{\mathbf{x}}$ and $\lambda_{\mathbf{x}}$ at each pixel. Typically, we find majority pixels on ground and walls only need small kernel and few iterations for recovery, while pixels further away and around object and surface boundary need large kernels and more iterations to obtain larger context for reconstruction. This agrees with our intuition since in real cases, sparse points are denser close by and the structure is simpler in planar regions, thus it is easier for depth estimation.

Ablation study of resource-aware CSPN (RA-CSPN). To verify the efficiency of our proposed RA-CSPN, we study the computational improvement w.r.t. vanilla CSPN and CA-CSPN. As list in Tab 2, at row 'CSPN', we list its memory cost and latency on device. At row 'CA-CSPN', although the memory cost and latency are in practice larger, but the expected kernel size $\mathbf{E}(k)$ and iteration steps $\mathbf{E}(t)$ are much smaller using our latency regularization terms. This indicates that most pixels only need small kernel and few iteration for obtaining better results. At row of 'RA-CSPN', we train with resource-aware objective as in Eq. (8), and show that RA-CSPN not only outperforms CSPN for efficiency (almost $3\times$ faster), but also improves RMSE from 756.27 to 732.32. More importantly, we can train RA-CSPN with

Table 2: Comparison of efficiency between CSPN and CSPN++. $\mathbf{E}(k)$ is the expected kernel size and $\mathbf{E}(t)$ is the expected number of iterations using learned $\alpha_{\mathbf{x}}$ and $\lambda_{\mathbf{x}}$. Cm is the real cost of memory and Cl is the real time latency on device. $m.c.$ is short for memory constraints and $l.c.$ is short for latency constraints. Both constraints and expected values are normalized by the corresponding resource used in the CSPN baseline. Note here the number of memory cost is not proportion to $\mathbf{E}(k)$ since the majority is taken by affinity matrix in our case. Here, we set a minimum cost of using kernel size of 3×3 and propagation steps of 3, and one may achieve additional acceleration by dropping the minimum cost.

DataSet	Method	kernel	iter.	m. c.	l. c.	Lower the Better				
						$\mathbf{E}(k)$	$\mathbf{E}(t)$	Cm(MB)	Cl(ms)	RMSE(mm)
KITTI	CSPN	7x7	12			1.0	1.0	829	28.88	756.27
	CA-CSPN	assemble	12			0.680	1.0	2125	67.23	732.46
	CA-CSPN	assemble	assemble			0.316	0.446	2125	67.23	**725.43**
	RA-CSPN	select	select			**0.268**	0.439	626.29	10.03	732.32
	RA-CSPN	select	select	0.35	0.35	0.333	**0.303**	**625.30**	**9.84**	742.17
NYU v2	CSPN	7x7	12			1.0	1.0	628	21.03	121.49
	CA-CSPN	assemble	assemble			**0.373**	0.451	1691	50.47	**115.73**
	RA-CSPN	select	select	0.40	0.40	0.386	**0.395**	**531.27**	**10.03**	118.70

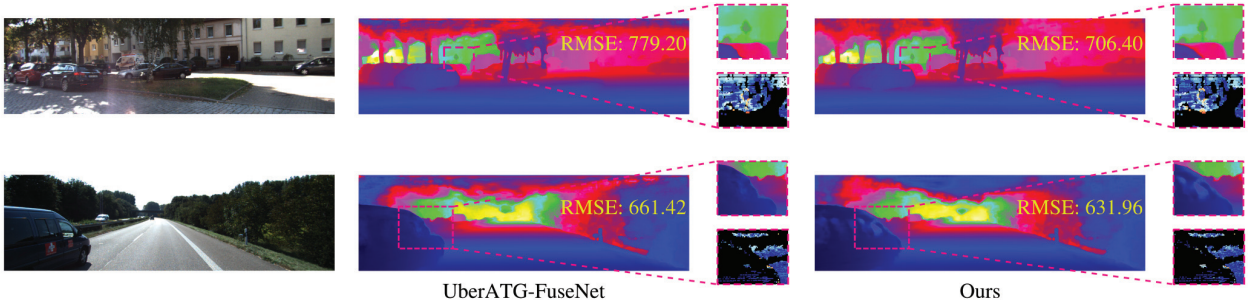

Figure 5: Qualitative comparison with UberATG-FuseNet on KITTI test set, where the zoom regions show that our method recover better and detailed structure.

computational budget to fit different devices as proposed in Eq. (10). At the last row, with a hard constrain that the m.c. and l.c. is less than 35% of the vanilla CSPN, we found that, our method will adjust kernel sizes and iteration actively. In this case, the $\mathbf{E}(t)$ reduce from 0.439 to 0.303 but $\mathbf{E}(k)$ increase from 0.268 to 0.333, which means that the network chooses larger kernel sizes with less iteration automatically to satisfied our hard constraints, while still produces better results and demonstrate the effectiveness of our method.

Experiments on NYU v2 dataset

To verify the generalization capability of our method in indoor scenes, we adopt same experiments on NYU v2 dataset. As shown in Tab. 2, we draw similar conclusions with the KITTI dataset, which shows the effectiveness of our method again.

Comparisons against other methods

Finally, to compare against other SoTA methods for depth estimation accuracy, we use our best obtained model from CA-CSPN, and finetune it with another 30 epochs before submitting the results to KITTI test server. As summarized in Tab. 3, CA-CSPN outperforms all other methods significantly and currently rank 2nd on the bench mark. However, our results are better in three out of the four metrics. Here, we would like to note that our results are also better than methods adopted additional dataset, *e.g.* DeepLiDAR (Qiu et al. 2019) uses CARLA (Dosovitskiy et al. 2017) to better learn dense depth and surface normal tasks jointly, and FusionNet (Van Gansbeke et al. 2019) used semantic pretrained segmentation models on CityScape (Cordts et al. 2016). Our plain model only trained on KITTI dataset and outperforms all other methods.

In Fig. 5, we qualitatively compare the dense depth maps estimated from our proposed mehtod with UberATG-FuseNet (Chen et al. 2019) together with the corresponding error maps. We found our results are better at detailed scene structure recovery.

Conclusion

In this paper, we propose CSPN++ for depth completion, which outperforms previous SoTA strategy CSPN (Cheng, Wang, and Yang 2018b) by a large margin. Specifically, we elaborate two variants using the same framework of model selection, *i.e.* context-aware CSPN and resource-aware CSPN. The former significantly reduces estimation error, while the later achieves much better efficiency with comparable accuracy with the former. We hope CSPN++

Table 3: Comparisons against state-of-the-art methods on KITTI Depth Completion benchmark.

Method	iRMSE (1/km)	iMAE (1/km)	RMSE (mm)	MAE (mm)
SC (Uhrig et al. 2017)	4.94	1.78	1601.33	481.27
CSPN (Cheng, Wang, and Yang 2018a)	2.93	1.15	1019.64	279.46
NC (Eldesokey, Felsberg, and Khan 2019)	2.60	1.03	829.98	233.26
StD (Ma, Cavalheiro, and Karaman 2019)	2.80	1.21	814.73	249.95
FN (Van Gansbeke et al. 2019)	2.19	0.93	772.87	215.02
DL (Qiu et al. 2019)	2.56	1.15	758.38	226.25
Uber (Chen et al. 2019)	2.34	1.14	752.88	221.19
CA-CSPN	**2.07**	**0.90**	**743.69**	**209.28**

could motivate researchers to better adopt data-driven strategies for effective learning hyper-parameters in various tasks. In the future, we would like merge the two variants, and consider replacing more modules in network with CSPN for multiple tasks such as segmentation and detection.

References

Cai, H.; Zhu, L.; and Han, S. 2019. Proxylessnas: Direct neural architecture search on target task and hardware. *ICLR*.

Chen, L.; Papandreou, G.; Schroff, F.; and Adam, H. 2017. Rethinking atrous convolution for semantic image segmentation. *CoRR* abs/1706.05587.

Chen, Y.; Liang, M.; Yang, B.; and Urtasun, R. 2019. Learning joint 2d-3d representations for depth completion. *ICCV*.

Cheng, X.; Wang, P.; and Yang, R. 2018a. Depth estimation via affinity learned with convolutional spatial propagation network. In *ECCV*, 103–119.

Cheng, X.; Wang, P.; and Yang, R. 2018b. Learning depth with convolutional spatial propagation network. *arXiv preprint arXiv:1810.02695*.

Cordts, M.; Omran, M.; Ramos, S.; Rehfeld, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; and Schiele, B. 2016. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 3213–3223.

Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; and Koltun, V. 2017. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*.

Eldesokey, A.; Felsberg, M.; and Khan, F. S. 2019. Confidence propagation through cnns for guided sparse depth regression. *TPAMI*.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017a. Densely connected convolutional networks. In *CVPR*, 4700–4708.

Huang, J.; Rathod, V.; Sun, C.; Zhu, M.; Korattikara, A.; Fathi, A.; Fischer, I.; Wojna, Z.; Song, Y.; Guadarrama, S.; et al. 2017b. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7310–7311.

Huang, G.; Che, D.; Li, T.; Wu, F.; van der Maaten, L.; and Weinberger, K. 2018. Multi-scale dense networks for resource efficient image classification. arxiv preprint arxiv: 170309844. *ICLR*.

Jaderberg, M.; Simonyan, K.; Zisserman, A.; et al. 2015. Spatial transformer networks. In *NIPS*, 2017–2025.

Ku, J.; Harakeh, A.; and Waslander, S. L. 2018. In defense of classical image processing: Fast depth completion on the cpu. In *CRV*, 16–22. IEEE.

Laina, I.; Rupprecht, C.; Belagiannis, V.; Tombari, F.; and Navab, N. 2016. Deeper depth prediction with fully convolutional residual networks. In *3D Vision (3DV), 2016 Fourth International Conference on*, 239–248. IEEE.

Li, X.; Liu, Z.; Luo, P.; Change Loy, C.; and Tang, X. 2017. Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *CVPR*, 3193–3202.

Liao, Y.; Huang, L.; Wang, Y.; Kodagoda, S.; Yu, Y.; and Liu, Y. 2017. Parse geometry from a line: Monocular depth estimation with partial laser observation. *ICRA*.

Liu, J., and Gong, X. 2013. Guided depth enhancement via anisotropic diffusion. In *Pacific-Rim Conference on Multimedia*, 408–417. Springer.

Ma, F., and Karaman, S. 2018. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. *ICRA*.

Ma, F.; Cavalheiro, G. V.; and Karaman, S. 2019. Self-supervised sparse-to-dense: Self-supervised depth completion from lidar and monocular camera. In *ICRA*, 3288–3295. IEEE.

Mock, W. B. 2011. Pareto optimality. *Encyclopedia of Global Justice* 808–809.

Qiu, J.; Cui, Z.; Zhang, Y.; Zhang, X.; Liu, S.; Zeng, B.; and Pollefeys, M. 2019. Deeplidar: Deep surface normal guided depth prediction for outdoor scene from sparse lidar data and single color image. In *CVPR*, 3313–3322.

Srivastava, R. K.; Greff, K.; and Schmidhuber, J. 2015. Highway networks. *arXiv preprint arXiv:1505.00387*.

Sun, K.; Xiao, B.; Liu, D.; and Wang, J. 2019. Deep high-resolution representation learning for human pose estimation. *CVPR*.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the inception architecture for computer vision. In *CVPR*, 2818–2826.

Tang, J.; Tian, F.-P.; Feng, W.; Li, J.; and Tan, P. 2019. Learning guided convolutional network for depth completion. *arXiv preprint arXiv:1908.01238*.

Uhrig, J.; Schneider, N.; Schneider, L.; Franke, U.; Brox, T.; and Geiger, A. 2017. Sparsity invariant cnns. *3DV*.

Van Gansbeke, W.; Neven, D.; De Brabandere, B.; and Van Gool, L. 2019. Sparse and noisy lidar completion with rgb guidance and uncertainty. *arXiv preprint arXiv:1902.05356*.

Wang, P.; Shen, X.; Russell, B.; Cohen, S.; Price, B.; and Yuille, A. L. 2016. Surge: Surface regularized geometry estimation from a single image. In *NIPS*, 172–180.

Wang, X.; Girshick, R.; Gupta, A.; and He, K. 2018a. Non-local neural networks. In *CVPR*, 7794–7803.

Wang, X.; Yu, F.; Dou, Z.-Y.; Darrell, T.; and Gonzalez, J. E. 2018b. Skipnet: Learning dynamic routing in convolutional networks. In *ECCV*, 409–424.

Xu, D.; Wang, W.; Tang, H.; Liu, H.; Sebe, N.; and Ricci, E. 2018. Structured attention guided convolutional neural fields for monocular depth estimation. In *CVPR*, 3917–3925.

Yang, M.; Yu, K.; Zhang, C.; Li, Z.; and Yang, K. 2018. Denseaspp for semantic segmentation in street scenes. In *CVPR*, 3684–3692.

Zhang, Y., and Funkhouser, T. 2018. Deep depth completion of a single rgb-d image. In *CVPR*, 175–185.

Zhao, H.; Shi, J.; Qi, X.; Wang, X.; and Jia, J. 2016. Pyramid scene parsing network. *CVPR*.

Zhu, X.; Hu, H.; Lin, S.; and Dai, J. 2019. Deformable convnets v2: More deformable, better results. In *CVPR*, 9308–9316.