Re-parameterization Making GC-Net-style 3DConvNets More Efficient

Takeshi Endo[1], Seigo Kaji[1], Haruki Matono[1], Masayuki Takemura[2], and Takeshi Shima[2]

[1] Hitachi, Ltd. Research & Development Group
{takeshi.endo.cw,seigo.kaji.vc,haruki.matono.dm}@hitachi.com
[2] Hitachi Astemo, Ltd.
{masayuki.takemura.gx,takeshi.shima.rb}@hitachiastemo.com

Abstract. For depth estimation using a stereo pair, deep learning methods using 3D convolution have been proposed. While the estimation accuracy is high, 3D convolutions on cost volumes are computationally expensive. Hence, we propose a method to reduce the computational cost of 3D convolution-based disparity networks. We apply kernel re-parameterization, which is used for constructing efficient backbones, to disparity estimation. We convert learned parameters, and these values are used for inference to reduce the computational cost of filtering cost volumes. Experimental results on the KITTI 2015 dataset show that our method can reduce the computational cost by 31–61% from those of trained models without any performance loss. Our method can be used for any disparity network that uses 3D convolution for cost volume filtering.

Keywords: Stereo matching · Re-parameterization · Efficient architecture.

1 Introduction

Depth estimation using a stereo pair is the core problem in computer vision and is applicable to autonomous driving. The depth can be obtained by calculating the disparity of a stereo pair, where disparity d is the horizontal displacement between a pair of corresponding pixels in the left and right images of the pair. If the position (x, y) in the left image corresponds to the position $(x - d, y)$ in the right image, then we can calculate the depth from $\frac{fB}{d}$, where f is the camera's focal length and B is the distance between the left and right cameras.

In recent years, deep learning has become the mainstream method for disparity estimation. Since DispNetC [12] was proposed, many methods have directly estimated disparities from stereo pairs. Specifically, DispNetC computes the correlation between left and right features to construct a cost volume which represents the matching cost. Then, it filters the cost volume with 2D convolutions to aggregate the cost. This method is efficient, but it loses rich information for channels. On the other hand, GC-Net [7] forms a concatenated cost volume

to keep channel information and aggregates the cost with 3D convolutions. This method offers improved accuracy by incorporating geometric and contextual information for disparity estimation. Subsequent 3D convolution-based methods have been proposed, such as PSMNet [2] and GWCNet [5]. These methods can easily be implemented on existing frameworks such as TensorFlow [1] because they have few network-specific operations. They can also be implemented on custom hardware, such as ASICs and FPGAs, with little additional module development. However, these methods filter the cost for each disparity d, which is computationally expensive. CRL [14] and HitNet [15] were thus proposed to speed up disparity estimation. While these 2D convolution-based approaches enable fast disparity estimation, they involve network-specific operations such as warping of feature maps to refine disparities. Accordingly, their implementation on custom hardware requires the development of new additional modules, which increases the implementation cost.

In this work, we aim to reduce the computational cost of 3D convolution-based methods because they can easily be implemented in various environments. For these methods, the most computationally expensive layer in the model is the first 3D convolution for the cost volume. Hence, we propose a method to re-parameterize the learned 3D convolution kernels for the cost volume and then use them during inference. Fig. 1 gives an overview of the proposed method. Our method reduces the computational cost without any performance loss, in contrast to network compression methods such as pruning [9], which degrade performance. Our method can easily be applied to any disparity network that uses 3D convolutions for cost volume filtering.

Our contributions are summarized below.

- We propose a re-parameterization method that can reduce the computational cost of any 3D convolution-based network for disparity estimation.
- We show experimentally that our method is effective for the networks of various model sizes.

2 Related Work

2.1 Disparity Estimation

Disparity estimation methods based on deep learning can be divided into two categories, depending on whether they are based on 2D or 3D convolution. 2D convolution-based methods compute the correlation between left and right feature maps to generate a cost volume, which is aggregated using 2D convolutions. In general, 2D methods are computationally inexpensive but have lower performance than 3D methods. Accordingly, structures that refine disparity maps have been proposed to achieve a favorable speed-accuracy tradeoff [14, 15]. In CRL [14] and HitNet [15], feature maps are warped to calculate error information, and the disparity map is updated by using that information. As these methods include network-specific operations, running them on custom hardware requires support for those unique operations.

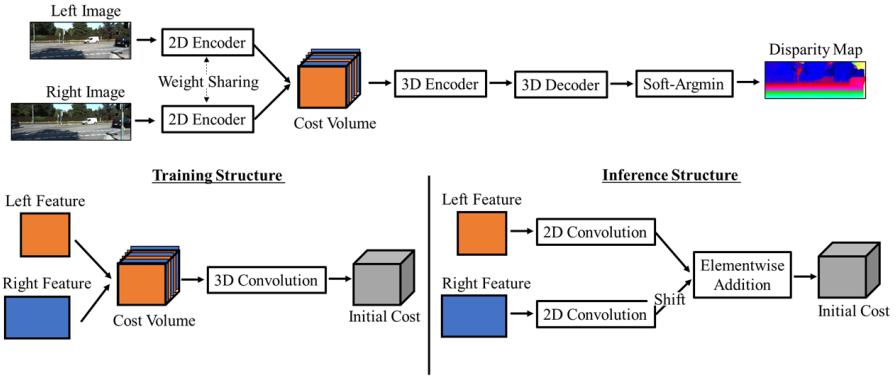


Fig. 1. Overview of the proposed method. (top) The 2D encoder extracts features by 2D convolution. Left and right features are concatenated to form the cost volume, which is then aggregated through the 3D encoder and decoder. Finally, a soft-argmin regresses the disparity. (bottom) Training and inference have different structures for computing the initial cost. During training, the initial cost is formed by 3D convolution on the cost volume. On the other hand, during inference, two 2D convolutions are used and these output values are added to form the initial cost.

In contrast, 3D convolution-based methods concatenate left and right feature maps to form a cost volume, which is filtered by using 3D convolutions to aggregate the cost. GC-Net [7] was the first such 3D method. It aggregates the cost with 3D convolutions and regresses the disparity map with a soft-argmin process. PSMNet [2] uses a spatial pyramid module for feature extraction, which enables the cost volume to incorporate multi-scale information. The problem with GC-Net and PSMNet is the increased computational cost due to the 3D convolutions. As alternatives, a method forming a low-resolution cost volume [8] and a method using group-wise correlation to construct a cost volume [5] were proposed. These network architectures of 3D methods are simpler than those of 2D methods. In addition, their architectures comprise generic operations that are used in other networks. On the other hand, the 3D methods are more computationally expensive than the 2D methods. Here, we aim to reduce the computational cost of 3D convolution-based methods from the viewpoint of ease of implementation.

2.2 Kernel Re-parameterization

To reduce the computational cost, we convert the learned parameters and use the values for inference. This idea is similar to research on kernel re-parameterization. [16] parameterizes a kernel as $\hat{\mathrm{W}}$=diag(a)I+diag(b)$\mathrm{W}_{\mathrm{norm}}$, where $\mathrm{W}_{\mathrm{norm}}$ is the normalized weight, and a and b are learnable vectors. The model uses implicit skip-connections, and it has the same structures during learning and inference, whereas we use different model structures during training and inference. Our approach is closely related to ExpandNet [4] and RepVGG [3]. ExpandNet converts

a block consisting of multiple convolutions with a single convolution layer. This is based on the idea that multiple linear layers can be fused when there is no nonlinear function. RepVGG is trained with a multi-branch model and uses its parameters in a single-path model during inference. The difference between our method and RepVGG is in the model structures before and after re-parameterization: we train a single-path model and use its parameters in a two-branch inference model. The type of convolution is also different, as RepVGG re-parameterizes 2D convolution kernels, as opposed to 3D convolution kernels in our approach. To the best of our knowledge, this is the first work to apply re-parameterization to disparity networks.

2.3 Network Comprerssion

Network compression methods such as pruning also reduce the computational cost. [9] proposes a structured pruning method. They remove filters that are identified as having a small effect on the output accuracy, and then finetune the network. Another pruning method [11] forces channel-level sparsity and leverages learnable batch normalization parameters for pruning. [10] explores the rank of feature maps and measures the information richness of feature maps by using the rank. All of these network compression methods change the network outputs before and after compression. In contrast, our method matches the outputs before and after re-parameterization, which places it along a different research direction from network compression.

3 Re-parameterization for 3DConvNets

In this section, we first describe the architecture of 3D convolution-based networks such as GC-Net. Then, we describe our re-parameterization approach in detail, followed by the implementation tricks required for re-parameterization.

3.1 GC-Net-style 3DConvNet

We first describe a general 3DConvNet (3D Convolution-based Network) as represented by GC-Net [7]. The model shown in the upper part of Fig. 1 represents the basic structure of a 3DConvNet. A stereo pair of left and right images is input, and shared 2D convolutions are used to extract the left feature map $\mathbf{f}_l$ and the right feature map $\mathbf{f}_r$. Given $\mathbf{f}_l$ and $\mathbf{f}_r$, the cost volume is computed, as

$$\mathbf{C}_{\text{concat}}(d, x, y) = \text{Concat}\{\mathbf{f}_l(x, y), \mathbf{f}_r(x - d, y)\}, \tag{1}$$

where d denotes the disparity level. We omit the channel dimension here for simplicity.

The cost volume $\mathbf{C}_{\text{concat}}$ is formed from $\mathbf{f}_l$ and $\mathbf{f}_r$ shifted relative to $\mathbf{f}_l$. The 3D encoder aggregates the cost with 3×3×3 3D convolutions, while the 3D decoder uses 3×3×3 3D deconvolutions to upsample the aggregated cost. The disparity map is regressed by a soft-argmin process [7].

Table 1 lists the details of the model structures to which our method applies. H and W denote the height and width of an input image, respectively. F denotes the number of channels for the feature map, and D indicates the maximum disparity to be estimated. The structures described in Table 1 are almost the same as those in GC-Net: the only difference is that we add a Stem2 layer to the 2D encoder to adjust the model's computational cost. In our experiments, we varied the stride values of the layers to build models having different computational costs.

The layers of the 3D encoder are denoted as follows. The output of the first convolution on the cost volume $\mathbf{C}_{\text{concat}}$ is denoted as $\mathbf{C}_{\text{initial}}$, because it represents the initial matching cost. The layers that correspond to $\mathbf{C}_{\text{initial}}$ in Table 1 are the InitialCost1 and InitialCost2 layers of the 3D encoder. Next, we denote the output of the 3D encoder's subsequent convolutions as $\mathbf{C}_{\text{aggregated}}$, because it represents the aggregated cost.

The computational cost of 3D convolution is proportional to the output size, which is similar to the property of 2D convolution. In the 3D encoder described in Table 1, the bottom layers such as InitialCost1 are computationally expensive. Hence, in this work, we propose a re-parameterization method to reduce the computational cost of constructing the initial cost $\mathbf{C}_{\text{initial}}$.

3.2 Re-parameterization for Efficient Inference

In our approach, we re-parameterize the 3×3×3 3D convolution kernels used to construct the initial cost $\mathbf{C}_{\text{initial}}$. Specifically, the 3D kernels are converted to 2D convolution kernels.

The initial cost $\mathbf{C}_{\text{initial}}$ is computed as follows,

$$\mathbf{C}_{\text{initial}}(d,x,y) = \sum_{-1\le l,m,n\le 1} \mathbf{W}_{3d}(l,m,n)\cdot \mathbf{C}_{\text{concat}}(d+l,x+m,y+n), \quad (2)$$

where $\mathbf{W}_{3d}$ is the 3D convolution kernel. From (1), (2) is equivalent to the following:

$$\begin{aligned}\mathbf{C}_{\text{initial}}(d,x,y) = &\sum_{-1\le l,m,n\le 1} \mathbf{W}_{3d,l}(l,m,n)\cdot \mathbf{f}_l(x+m,y+n) \\ &+ \sum_{-1\le l,m,n\le 1} \mathbf{W}_{3d,r}(l,m,n)\cdot \mathbf{f}_r(x-d+m-l,y+n), \quad (3)\end{aligned}$$

where $\mathbf{W}_{3d,l}$ and $\mathbf{W}_{3d,r}$ are 3D convolution kernels and parts of $\mathbf{W}_{3d}$. $\mathbf{W}_{3d,l}$ and $\mathbf{W}_{3d,r}$ are used for filtering $\mathbf{f}_l$ and $\mathbf{f}_r$, respectively. If the number of channels in $\mathbf{f}_l$ and $\mathbf{f}_r$ is F, then $\mathbf{W}_{3d}$ has 2F channels. The kernel corresponding to channels $0\cdots(\mathrm{F}-1)$ is $\mathbf{W}_{3d,l}$, and the one corresponding to channels $\mathrm{F}\cdots(2\mathrm{F}-1)$ is $\mathbf{W}_{3d,r}$. The first term in (3) is the operation on the left feature map $\mathbf{f}_l$ and the second term is the operation on the right feature map $\mathbf{f}_r$.

In the first term of (3), the distributive law holds. Therefore, $\mathbf{W}_{3d,l}$ can be fused in the disparity direction l. This yields the following equation for the first

Table 1. Summary of our 3DConvNet architecture, which has a 2D encoder, cost volume layer, 3D encoder, and 3D decoder. Each convolutional layer comprises convolution, batch normalization, and ReLU activation. The layer properties consist of the kernel size, convolution type, number of kernels, and stride, in this order.

(a) 2D encoder

Name	Layer Property	Output Size
Stem1	5×5, conv2d, 32, 2	$\frac{H}{2}\times\frac{W}{2}\times F$
Stem2	3×3, conv2d, 32, 1	$\frac{H}{2}\times\frac{W}{2}\times F$
Resblocks	$\left.\begin{matrix}(3\times3, \text{conv2d}, 32, 1)\times2\\ \text{skip connect}\end{matrix}\right\}\times8$	$\frac{H}{2}\times\frac{W}{2}\times F$
Conv1	3×3, conv2d, 32, 1 (no ReLU and BN)	$\frac{H}{2}\times\frac{W}{2}\times F$

(b) Cost volume layer and 3D encoder

Name	Layer Property	Output Size
	Cost Volume	
ConcatCost	Concat left & right features	$\frac{D}{2}\times\frac{H}{2}\times\frac{W}{2}\times2F$
	3D Encoder	
InitialCost1	From ConcatCost: $3\times3\times3$, conv3d, 32, 1	$\frac{D}{2}\times\frac{H}{2}\times\frac{W}{2}\times F$
AggCost1	$3\times3\times3$, conv3d, 32, 1	$\frac{D}{2}\times\frac{H}{2}\times\frac{W}{2}\times F$
InitialCost2	From ConcatCost: $3\times3\times3$, conv3d, 64, 2	$\frac{D}{4}\times\frac{H}{4}\times\frac{W}{4}\times2F$
AggCost2-1	$(3\times3\times3, \text{conv3d}, 64, 1)\times2$	$\frac{D}{4}\times\frac{H}{4}\times\frac{W}{4}\times2F$
AggCost3-1	From InitialCost2: $3\times3\times3$, conv3d, 64, 2	$\frac{D}{8}\times\frac{H}{8}\times\frac{W}{8}\times2F$
AggCost3-2	$(3\times3\times3, \text{conv3d}, 64, 1)\times2$	$\frac{D}{8}\times\frac{H}{8}\times\frac{W}{8}\times2F$
AggCost4-1	From AggCost3-1: $3\times3\times3$, conv3d, 64, 2	$\frac{D}{16}\times\frac{H}{16}\times\frac{W}{16}\times2F$
AggCost4-2	$(3\times3\times3, \text{conv3d}, 64, 1)\times2$	$\frac{D}{16}\times\frac{H}{16}\times\frac{W}{16}\times2F$
AggCost5-1	From AggCost4-1: $3\times3\times3$, conv3d, 128, 2	$\frac{D}{32}\times\frac{H}{32}\times\frac{W}{32}\times4F$
AggCost5-2	$(3\times3\times3, \text{conv3d}, 128, 1)\times2$	$\frac{D}{32}\times\frac{H}{32}\times\frac{W}{32}\times4F$

(c) 3D decoder

Name	Layer Property	Output Size
AggCost6	$3\times3\times3$, deconv3d, 64, 2 skip connect (from AggCost4-2)	$\frac{D}{16}\times\frac{H}{16}\times\frac{W}{16}\times2F$
AggCost7	$3\times3\times3$, deconv3d, 64, 2 skip connect (from AggCost3-2)	$\frac{D}{8}\times\frac{H}{8}\times\frac{W}{8}\times2F$
AggCost8	$3\times3\times3$, deconv3d, 64, 2 skip connect (from AggCost2-1)	$\frac{D}{4}\times\frac{H}{4}\times\frac{W}{4}\times2F$
AggCost9	$3\times3\times3$, deconv3d, 32, 2 skip connect (from AggCost1)	$\frac{D}{2}\times\frac{H}{2}\times\frac{W}{2}\times F$
AggCost10	$3\times3\times3$, deconv3d, 1, 2 (no ReLU and BN)	$D\times H\times W\times1$

term of (3):

$$\sum_{-1\le l,m,n\le1}\mathbf{W}_{3d,l}(l,m,n)\cdot\mathbf{f}_l(x+m,y+n)=\sum_{-1\le m,n\le1}\mathbf{W}'_{2d,l}(m,n)\cdot\mathbf{f}_l(x+m,y+n),\quad(4)$$

where $\mathbf{W}'_{2d,l}(m,n)$ is defined as

$$\mathbf{W}'_{2d,l}(m,n)=\sum_{-1\le l\le1}\mathbf{W}_{3d,l}(l,m,n).\quad(5)$$

Next, for the second term in (3), we let the 3×5 2D kernel $\mathbf{W}'_{2d,r}$ be given by,

$$\mathbf{W}'_{2d,r}(m,n)=\sum_{-1\le l\le1}\mathbf{W}'_{3d,r}(l,m+l,n),\quad(6)$$

where

$$\mathbf{W}'_{3d,r}(l,m,n)=\begin{cases}\mathbf{W}_{3d,r}(l,m,n), & -1\le m\le1\\ 0, & \text{otherwise.}\end{cases}\quad(7)$$

Substituting (6) into the second term in (3), we obtain the following equation:

$$\sum_{-1\leq l,m,n\leq 1} \mathbf{W}_{3d,r}(l,m,n)\cdot\mathbf{f}_r(x-d+m-l,y+n) = \sum_{\substack{-2\leq m\leq 2\\-1\leq n\leq 1}} \mathbf{W}'_{2d,r}(m,n)\cdot\mathbf{f}_r(x-d+m,y+n). \quad (8)$$

From (4) and (8), $\mathbf{C}_{\text{initial}}(d,x,y)$ is given by,

$$\mathbf{C}_{\text{initial}}(d,x,y) = \sum_{-1\leq m,n\leq 1} \mathbf{W}'_{2d,l}(m,n)\cdot\mathbf{f}_l(x+m,y+n) + \sum_{\substack{-2\leq m\leq 2\\-1\leq n\leq 1}} \mathbf{W}'_{2d,r}(m,n)\cdot\mathbf{f}_r(x-d+m,y+n). \quad (9)$$

Convolution is not affected by translation. As the order of shift and convolution operations can be interchanged, we have

$$\mathbf{W} * \text{Shift}(\mathbf{f},d) = \text{Shift}(\mathbf{W}*\mathbf{f},d), \quad (10)$$

where $\text{Shift}(\mathbf{X},d)$ denotes shifting a tensor $\mathbf{X}$ by d pixels. Hence, we derive the following:

$$\mathbf{C}_{\text{initial}}(d) = \mathbf{W}'_{2d,l} * \mathbf{f}_l + \text{Shift}(\mathbf{W}'_{2d,r} * \mathbf{f}_r, d), \quad (11)$$

where $*$ is the convolution operator.

(11) indicates that the initial cost $\mathbf{C}_{\text{initial}}$ can be computed only from 2D convolutions, without 3D convolutions. It is obvious that we only need to calculate $\mathbf{W}'_{2d,l} * \mathbf{f}_l$ and $\mathbf{W}'_{2d,r} * \mathbf{f}_r$ once to form $\mathbf{C}_{\text{initial}}$. Hence, we compute $\mathbf{C}_{\text{initial}}$ as shown in Fig. 1. We first filter the left feature map $\mathbf{f}_l$ and the right feature map $\mathbf{f}_r$ by using the respective 2D convolution kernels $\mathbf{W}'_{2d,l}$ and $\mathbf{W}'_{2d,r}$. The filtered right feature map is then shifted and added to the filtered left feature map, which allows us to construct the initial cost $\mathbf{C}_{\text{initial}}$.

Here, we re-parameterize the trained model's 3D convolution kernels into 2D convolution kernels, as shown in Fig. 2. The 3D kernels are decomposed to filter the left and right feature maps. As seen in (5), the kernels for left feature maps are re-parameterized by element addition, while those for right feature maps are re-parameterized according to (6) and (7). The re-parameterized 2D convolution kernels are then used for inference.

The re-parameterized 2D convolutions reduce the computational cost. We measure the computational cost in terms of FLOPs, the number of multiply-adds required for convolutions. For the model described in Table 1, the computation of the InitialCost1 layer requires $3^3\cdot\frac{1}{2}\mathrm{D}\cdot\frac{1}{2}\mathrm{W}\cdot\frac{1}{2}\mathrm{H}\cdot 2\mathrm{F}^2$ FLOPs. On the other hand, the re-parameterized model requires $3^2\cdot\frac{1}{2}\mathrm{W}\cdot\frac{1}{2}\mathrm{H}\cdot\mathrm{F}^2 + 3\cdot 5\cdot\frac{1}{2}\mathrm{W}\cdot\frac{1}{2}\mathrm{H}\cdot\mathrm{F}^2$ FLOPs. Thus, the computational cost is reduced to $\frac{8}{9\cdot\mathrm{D}}$. As D is 192 in general, the layer can be computed with 0.5% of the original computational cost in FLOPs.

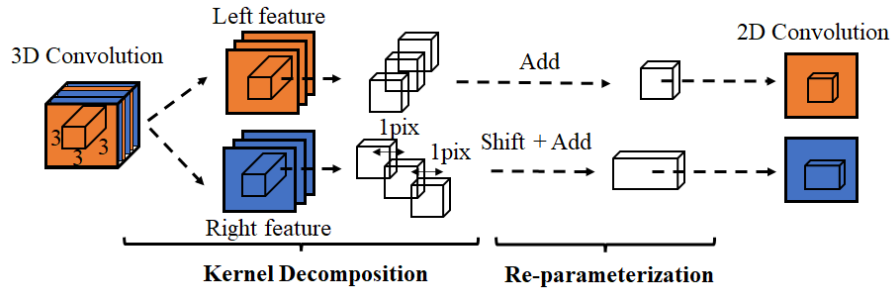


Fig. 2. Kernel re-parameterization, in which 3D convolution kernels are converted into 2D kernels. Each 3D kernel is decomposed into left and right kernels. The left kernels corresponding to each disparity are fused by elementwise addition, while the right kernels corresponding to each disparity are fused by a shift operation and elementwise addition. The re-parameterized 2D kernels are then used for inference.

3.3 Implementation Tricks

Because of the effect of padding, the outputs before and after re-parameterization do not match. Hence, we describe implementation tricks to avoid this issue, namely, left image cropping and initial cost cropping. Because these tricks reduce the output size of the disparity map, our approach cannot estimate disparities at left and right sides of an image. However, in the case of a real application, disparity estimation is performed on a partial region of interest. Accordingly, we assume that disparity calculations at the left and right sides are not necessarily required.

Left Image Cropping. To construct the cost volume $\mathbf{C}_{\text{concat}}$, the left and (shifted) right feature maps are concatenated in the channel direction, as seen in (1). The left side of the right feature map is padded with zeros when the feature map is shifted to the right. The number of padded columns is equal to the number of shifts. In the left feature map, the same number of columns is also padded. Thus, the left feature map $\mathbf{f}_l$ is padded differently at each disparity level d. This is also true for the right feature map $\mathbf{f}_r$.

In our method, however, $\mathbf{f}_l$ must not be padded differently at different disparity levels d. This is because $\mathbf{W}'_{2d,l} * \mathbf{f}_l$ is repeatedly used for constructing the initial cost $\mathbf{C}_{\text{initial}}(d)$ at any disparity level. This is also true for $\mathbf{f}_r$. To satisfy these conditions, we use stereo pairs with different widths. Given the shift of the right feature map, the width of the right image is set to be D pixels larger than that of the left image. Thus, we crop the left image so that its width is $\mathrm{W_l} = \mathrm{W_r} - \mathrm{D}$, where $\mathrm{W_r}$ is that of the right image. This gives the difference between the widths of the left and right feature maps. Hence, we crop the right feature $\mathbf{f}_r$ to the same width as the left feature map $\mathbf{f}_l$. Then, we concatenate them in the channel direction.

Initial Cost Cropping. In general, a cost volume is padded with zeros in the spatial and disparity directions when a 3D convolution is performed on it. Accordingly, we must handle the effect of padding in the x and d directions so that the outputs before and after re-parameterization match. During training, we use the 3D convolution to filter the cost volume. The convolution outputs at position $x = 0$, W and $d = 0$, D are affected by padding. During inference, we use the shift operation and elementwise addition to construct the initial cost, as seen in (11). Because these operations lead to a different padding effect, we crop the cost volume to eliminate pixels that do not match in training and inference, as shown in Fig. 3.

During training, we first crop the left and right sides of the left feature map $\mathbf{f}_l$. This operation is necessary to keep the original disparity range ($d = 0 \cdots \mathrm{D}$). Then, the left and right feature maps are concatenated, and the 3D convolution is performed to construct the initial cost. Finally, we crop the initial cost in the x and d directions to remove the pixels that are affected by padding.

During inference, we use a similar procedure. First, the left feature map is cropped. Then, the re-parameterized 2D convolutions are performed on the left and right feature maps, and the shift operation and elementwise addition are used to construct the initial cost. Finally, we crop the initial cost in the x and d directions.

3.4 Disparity Regression

We use the disparity regression method proposed in [7] to estimate disparities. The soft-argmin operation enables us to estimate disparities with sub-pixel accuracy. The probability of each disparity level d is used for estimation. The negative of the predicted cost c_d is converted to the probability via the softmax operation. The predicted disparity $\hat{d}$ is then calculated as the sum of each disparity level d weighted by its probability:

$$\hat{d} = \sum_{d=0}^{D} d \times \sigma(-c_d), \tag{12}$$

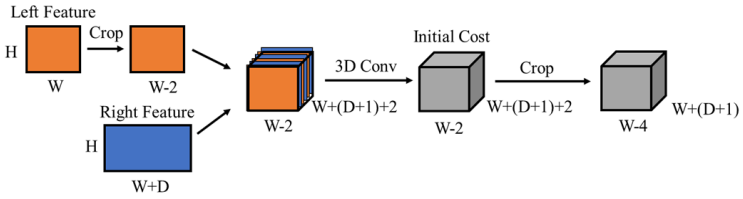
where σ denotes the softmax operation.

3.5 Loss Function

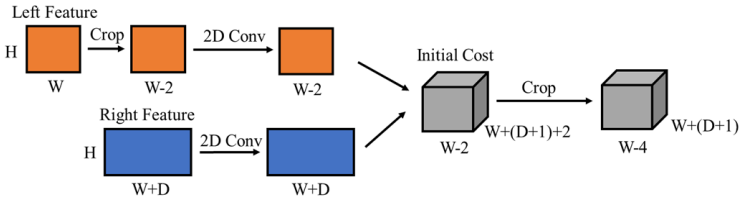
We use the loss function proposed in [7], which is defined as follows:

$$Loss = \frac{1}{N} \sum_{n=0}^{N} \|d - \hat{d}\|_1, \tag{13}$$

where N is the number of labeled pixels, $\hat{d}$ is the predicted disparity, and d is the ground-truth disparity.



(a) Cost volume cropping during training.



(b) Cost volume cropping during inference.

Fig. 3. Cost volume cropping. (a) During training, the left and right sides of the left feature map are cropped. Then, the initial cost is constructed by a 3D convolution, and it is finally cropped in the x and d directions. (b) During inference, the cropping is almost the same as during training; the difference is the way to construct the initial cost.

4 Experiments

In this section, we describe our experimental conditions and our results on the KITTI 2015 dataset [13]. In these experiments, we compared several models having different computational costs. We also performed ablation studies to further evaluate our method.

4.1 Experimental Conditions

During training, we used the Monkaa dataset [12] for pre-training and the KITTI 2015 dataset [13] for finetuning. For evaluation, we used the KITTI 2015 dataset. As we evaluated several models, we did not use the KITTI 2015 official evaluation dataset. Instead, we split the original 200 training images into 160 training images and 40 evaluation images.

For training on these datasets, we cropped the left and right images to input sizes of 256×512 and 256×704, respectively. We set $\mathrm{D} = 192$ and $\mathrm{F} = 32$. RMSprop [6] was used as the optimizer, with a learning rate of 1×10^{-3}. These settings are the same as in [7]. For evaluation, we padded zeros at the top and on the right side of an image. The input sizes were 384×1056 for left images and 384×1248 for right images.

Table 2. Model structures of the (a) 3DConvNet-B1, (b) 3DConvNet-B0, and (c) Rep3DConvNet-B0 models.

(a) 3DConvNet-B1

Name	Layer Property	Output Size
	3D Encoder	
InitialCost1	From ConcatCost: $3\times3\times3, \text{conv3d}, 64, 1$	$\frac{D}{4}\times\frac{H}{4}\times\frac{W}{4}\times2F$
AggCost1	$3\times3\times3, \text{conv3d}, 64, 1$	$\frac{D}{4}\times\frac{H}{4}\times\frac{W}{4}\times2F$
InitialCost2	From ConcatCost: $3\times3\times3, \text{conv3d}, 64, 2$	$\frac{D}{8}\times\frac{H}{8}\times\frac{W}{8}\times2F$
AggCost2-1	$(3\times3\times3, \text{conv3d}, 64, 1)\times2$	$\frac{D}{8}\times\frac{H}{8}\times\frac{W}{8}\times2F$
AggCost3-1	From InitialCost2: $3\times3\times3, \text{conv3d}, 64, 2$	$\frac{D}{16}\times\frac{H}{16}\times\frac{W}{16}\times2F$
AggCost3-2	$(3\times3\times3, \text{conv3d}, 64, 1)\times2$	$\frac{D}{16}\times\frac{H}{16}\times\frac{W}{16}\times2F$
AggCost4-1	From AggCost3-1: $3\times3\times3, \text{conv3d}, 128, 2$	$\frac{D}{32}\times\frac{H}{32}\times\frac{W}{32}\times4F$
AggCost4-2	$(3\times3\times3, \text{conv3d}, 128, 1)\times2$	$\frac{D}{32}\times\frac{H}{32}\times\frac{W}{32}\times4F$
	3D Decoder	
AggCost5	$3\times3\times3, \text{deconv3d}, 64, 2$ skip connect (from AggCost3-2)	$\frac{D}{16}\times\frac{H}{16}\times\frac{W}{16}\times2F$
AggCost6	$3\times3\times3, \text{deconv3d}, 64, 2$ skip connect (from AggCost2-1)	$\frac{D}{8}\times\frac{H}{8}\times\frac{W}{8}\times2F$
AggCost7	$3\times3\times3, \text{deconv3d}, 64, 2$ skip connect (from AggCost1)	$\frac{D}{4}\times\frac{H}{4}\times\frac{W}{4}\times2F$
AggCost8	$3\times3\times3, \text{deconv3d}, 1, 2$ (no ReLU and BN)	$\frac{D}{2}\times\frac{H}{2}\times\frac{W}{2}\times1$

(b) 3DConvNet-B0

Name	Layer Property	Output Size
	3D Encoder	
InitialCost1	From ConcatCost: $3\times3\times3, \text{conv3d}, 64, 1$	$\frac{D}{4}\times\frac{H}{4}\times\frac{W}{4}\times2F$
AggCost2-1	From InitialCost1: $3\times3\times3, \text{conv3d}, 64, 2$	$\frac{D}{8}\times\frac{H}{8}\times\frac{W}{8}\times2F$
AggCost2-2	$(3\times3\times3, \text{conv3d}, 64, 1)\times2$	$\frac{D}{8}\times\frac{H}{8}\times\frac{W}{8}\times2F$
AggCost3-1	From AggCost2-1: $3\times3\times3, \text{conv3d}, 64, 2$	$\frac{D}{16}\times\frac{H}{16}\times\frac{W}{16}\times2F$
AggCost3-2	$(3\times3\times3, \text{conv3d}, 64, 1)\times2$	$\frac{D}{16}\times\frac{H}{16}\times\frac{W}{16}\times2F$
	3D Decoder	
AggCost4	$3\times3\times3, \text{deconv3d}, 64, 2$ skip connect (from AggCost2-2)	$\frac{D}{8}\times\frac{H}{8}\times\frac{W}{8}\times2F$
AggCost5	$3\times3\times3, \text{deconv3d}, 1, 2$ (no ReLU and BN)	$\frac{D}{4}\times\frac{H}{4}\times\frac{W}{4}\times1$

(c) Rep3DConvNet-B0

Name	Layer Property	Output Size
Conv2D-L	From 2D encoder's left feature: $3\times3, \text{conv2d}, 64, 1$	$\frac{H}{4}\times\frac{W}{4}\times2F$
Conv2D-R	From 2D encoder's right feature: $3\times5, \text{conv2d}, 64, 1$	$\frac{H}{4}\times\frac{W+D}{4}\times2F$
InitialCost1	For each disparity d: Conv2D-L+Shift(Conv2D-R,d)	$\frac{D}{4}\times\frac{H}{4}\times\frac{W}{4}\times2F$
	3D Encoder	
	AggCost$*$ layers in 3D Encoder of 3DConvNet-B0	
	3D Decoder	
	AggCost$*$ layers in 3D Decoder of 3DConvNet-B0	

4.2 Experimental Results

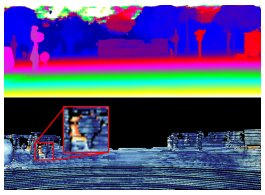
We trained four models with different computational costs. We refer to the model described in section 3.1 as 3DConvNet-B3. We created three other models with reduced computational costs: 3DConvNet-B2, 3DConvNet-B1, and 3DConvNet-B0. First, 3DConvNet-B2 limited the number of convolutions of the 3D encoder, as listed in Table 1. Specifically, the AggCost1 is removed, and the number of convolution layers was reduced to one in the AggCost2-1, AggCost3-2, AggCost4-2, and AggCost5-2 layers. Next, 3DConvNet-B1 and 3DConvNet-B0 used a stride of two for the 2D encoder's Stem2 layer to reduce the spatial resolution. The resulting structures of 3DConvNet-B1 and 3DConvNet-B0 are given in Table 2. The structure of 3DConvNet-B1 was similar to that of 3DConvNet-B3, but the 3D encoder's input size and the 3D decoder's output size were different. 3DConvNet-B0 was the lightest model: its 3D encoder and 3D decoder were shallow, and the output disparity size was one-fourth of the input size.

As described in section 3, the proposed method uses 3D convolutions to compute the initial cost during training. The 3D convolutions are re-parameterized, and the re-parameterized kernels are then used for inference. We refer to the re-parameterized model that uses 2D convolutions to compute the initial cost as Rep3DConvNet. In Table 2(c), we show the structure of Rep3DConvNet-B0. The difference between the model and 3DConvNet-B0 is in the initial cost construction process. As described in section 3.3, our methods cannot estimate

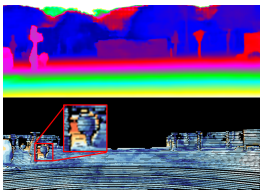
Table 3. Experimental results on 40 evaluation images from the KITTI 2015 dataset. The abbreviations 'bg' and 'fg' refer to the results on background and dynamic object pixels, respectively. Each model was evaluated before and after re-parameterization. The column 'FLOPs' gives the computational cost of the entire network, while the column 'FLOPs†' shows the cost for the initial cost construction. The runtime and memory consumption were measured on an NVIDIA Tesla V100 GPU (16 GB).

Model	Inference Structure		D1 error(%)			FLOPs	FLOPs†	Runtime	Memory
	3DConv	2DConv	D1-bg	D1-fg	D1-all	(B)	(G)	(sec)	(GiB)
3DConvNet-B0	✓		3.19	9.15	4.01	0.23	141.9	0.059	3.1
Rep3DConvNet-B0		✓				0.09	1.4	0.041	2.8
3DConvNet-B1	✓		2.58	7.76	3.30	0.52	160.8	0.102	4.9
Rep3DConvNet-B1		✓				0.36	1.7	0.077	3.1
3DConvNet-B2	✓		2.48	4.69	2.79	1.68	695.8	0.378	15.3
Rep3DConvNet-B2		✓				0.98	4.2	0.225	9.9
3DConvNet-B3	✓		2.20	5.18	2.62	2.10	695.8	0.462	15.3
Rep3DConvNet-B3		✓				1.41	4.2	0.311	10.4

(a) Input image



(b) Rep3DConvNet-B3



(c) Rep3DConvNet-B0

Fig. 4. Example of qualitative results on KITTI 2015, showing (a) the input image, and the output results from (b) Rep3DConvNet-B3 and (c) Rep3DConvNet-B0. In (b) and (c), the top and bottom images are the disparity and error maps, respectively.

disparities at the left and right sides of an image. Thus, in our evaluation, the accuracy was measured for pixels excluding those at the left and right sides. For evaluation, we used the D1 error metric, which counts the number of pixels that satisfy $|d - \hat{d}| > 3$pix, or $\frac{|d-\hat{d}|}{d} > 5\%$, for the ground-truth d and estimated disparity $\hat{d}$. Better models have lower scores for the D1 error. Note that our models estimate disparity maps at different scales. If the estimated disparity maps are smaller than the input images, we upsample the outputs by bilinear interpolation.

The evaluation results on the KITTI dataset are summarized in Table 3. Specifically, we give the results for 3DConvNet and Rep3DConvNet. The results show that our method could reduce the computational cost without any performance loss. In particular, the re-parameterization reduced the computational cost of the lightest model, 3DConvNet-B0, by 61%, and that of 3DConvNet-B3, the largest model, by 33%. The reduction rate depends on the computational cost for the initial cost construction. Overall, our experiment showed that the proposed method was effective for various models with different computational costs. We also measured the actual run time and memory consumption. As seen

Table 4. Ablation study results. (a) D1 error for 3DConvNet-B3 in various training settings, where 'Trick1' and 'Trick2' denote left image cropping and initial cost cropping, respectively. (b) Accuracy comparison between 2DConvNet-B1, which is trained with the same structure as the re-parameterized model, and Rep3DConvNet-B1.

(a) Ablation 1

		D1 error(%)		
Trick1	Trick2	D1-bg	D1-fg	D1-all
		2.59	7.43	3.26
✓		2.33	5.31	2.74
✓	✓	2.20	5.18	2.62

(b) Ablation 2

	D1 error(%)		
Model	D1-bg	D1-fg	D1-all
2DConvNet-B1	2.77	8.19	3.52
Rep3DConvNet-B1	2.58	7.76	3.30

in the table, the re-parameterization accelerated the inference and reduced the memory footprint.

Fig. 4 shows an example of the output results for Rep3DConvNet-B0 and Rep3DConvNet-B3. Rep3DConvNet-B3 had more layers and a higher resolution for disparity maps. Thus, it could suppress depth mixing, which means the errors between foreground and background pixels, better than Rep3DConvNet-B0 could. On the other hand, the overall disparity was similar to that for Rep3DConvNet-B3, even though Rep3DConvNet-B0 had 6% of the convolution operations in Rep3DConvNet-B3. The required accuracy depends on the application for disparity estimation. The results demonstrate that our method can reduce the computational cost from computationally expensive models to much smaller models.

4.3 Ablation Study

Table 4(a) lists the results of this ablation study, where 'trick1' and 'trick2' denote left image cropping and initial cost cropping, respectively. It can be seen that the performance was improved by using trick1. This was because trick1 eliminates occlusion pixels, enabling the model to find better correspondences. We can also see that trick2 did not degrade the performance. Thus, these tricks are unlikely to have negative impacts on the accuracy.

Next, although we use different model structures during training and inference, it is also possible to train a network with the same structure as the re-parameterized model. We refer to the resulting 2D convolution-based model, which is not re-parameterized, as 2DConvNet. Table 4(b) lists the results of a comparison between 2DConvNet-B1 and Rep3DConvNet-B1, which shows that the re-parameterized model had better accuracy. In our re-parameterization process, the number of parameters during training is larger than the number during inference. This can be viewed as learning an over-parameterized model and then using a compact model during inference. In ExpandNet [4], over-parameterization stabilizes the gradients during training. It is also known that the loss landscape becomes flat, which gives better generalization performance.

Accordingly, it is important to train models by using 3D convolution and then re-parameterize them during inference, as in our proposed method.

5 Conclusion

We have proposed a kernel re-parameterization method to reduce the computational cost of 3D convolution-based disparity networks. Our method re-parameterizes learned 3D convolution kernels and uses them as 2D convolution kernels. Our experimental results show that the proposed method can reduce the computational cost without degrading the trained model's performance. Specifically, the computational cost could be reduced by 31–61% for models with different sizes. In addition, the proposed method dramatically reduces the computational cost for constructing the initial cost, which makes it feasible to construct many initial costs. Hence, a future work will be to develop a network structure in which our method is effective.

Acknowledgements We want to thank Atsushi Yokoyama and Kumud Shishir for their helpful comments on this paper.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: Large-scale machine learning on heterogeneous systems. In: Software available from tensorflow.org (2015)
2. Chang, J., Chen, Y.: Pyramid stereo matching network. In: The IEEE International Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5410–5418 (2018)
3. Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., Sun, J.: Repvgg: Making vgg-style convnets great again. In: The IEEE International Conference on Computer Vision and Pattern Recognition (CVPR). pp. 13733–13742 (2021)
4. Guo, S., Alvarez, J., Salzmann, M.: Expandnets: Linear over-parameterization to train compact convolutional networks. In: Advances in Neural Information Processing Systems (NeurIPS) 33. pp. 1298–1310 (2020)
5. Guo, X., Yang, K., Yang, W., Wang, X., Li, H.: Group-wise correlation stereo network. In: The IEEE International Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3273–3282 (2019)
6. Hinton, G., Srivastava, N., Swersky, K.: Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Cited on **14**(8), 2 (2012)
7. Kendall, A., Martirosyan, H., Dasgupta, S., Henry, P., Kennedy, R., Bachrach, A., Bry, A.: End-to-end learning of geometry and context for deep stereo regression. In: The IEEE International Conference on Computer Vision (ICCV). pp. 66–75 (2017)

8. Khamis, S., Fanello, S., Rhemann, C., Kowdle, A., Valentin, J., Izadi, S.: Stereonet: Guided hierarchical refinement for real-time edge-aware depth prediction. In: European Conference on Computer Vision (ECCV). pp. 573–590 (2018)
9. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: arXiv pre-print arXiv: 1608.08710 (2016)
10. Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., Shao, L.: Hrank: Filter pruning using high-rank feature map. In: The IEEE International Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1529–1538 (2020)
11. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutonal networks through network slimming. In: The IEEE International Conference on Computer Vision (ICCV). pp. 2736–2744 (2017)
12. Mayer, N., Ilg, E., Hausser, P., Fischer, P., Cremers, D., Dosovitskiy, A., Brox, T.: A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In: The IEEE International Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4040–4048 (2016)
13. Menze, M., Geiger, A.: Object scene flow for autonomous vehicles. In: The IEEE International Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3061–3070 (2015)
14. Pang, J., Sun, W., Ren, J.S., Yang, C., Yan, Q.: Cascade residual learning: A two-stage convolutional neural network for stereo matching. In: The IEEE International Conference on Computer Vision (ICCV). pp. 887–895 (2017)
15. Tankovich, V., Hane, C., Zhang, Y., Kowdle, A., Fanello, S., Bouaziz, S.: Hitnet: Hierarchical iterative tile refinement network for real-time stereo matching. In: The IEEE International Conference on Computer Vision and Pattern Recognition (CVPR). pp. 14362–14372 (2021)
16. Zagoruyko, S., Komodakis, N.: Diracnets: Training very deep neural networks without skip-connections. In: arXiv pre-print arXiv: 1706.00388 (2018)