Designing by Training: Acceleration Neural Network for Fast High-Dimensional Convolution

Longquan Dai
School of Computer Science and Engineering
Nanjing University of Science and Technology
dailongquan@njust.edu.cn

Liang Tang
CASA Environmental Technology Co., Ltd
CASA EM&EW IOT Research Center
tangl@casaet.com

Yuan Xie
Institute of Automation
Chinese Academy of Sciences
yuan.xie@ia.ac.cn

Jinhui Tang[*]
School of Computer Science and Engineering
Nanjing University of Science and Technology
jinhuitang@njust.edu.cn

Abstract

The high-dimensional convolution is widely used in various disciplines but has a serious performance problem due to its high computational complexity. Over the decades, people took a handmade approach to design fast algorithms for the Gaussian convolution. Recently, requirements for various non-Gaussian convolutions have emerged and are continuously getting higher. However, the handmade acceleration approach is no longer feasible for so many different convolutions since it is a time-consuming and painstaking job. Instead, we propose an Acceleration Network (AccNet) which turns the work of designing new fast algorithms to training the AccNet. This is done by: 1, interpreting splatting, blurring, slicing operations as convolutions; 2, turning these convolutions to gCP layers to build AccNet. After training, the activation function g together with AccNet weights automatically define the new splatting, blurring and slicing operations. Experiments demonstrate AccNet is able to design acceleration algorithms for a ton of convolutions including Gaussian/non-Gaussian convolutions and produce state-of-the-art results.

1 Introduction

The high-dimensional convolution undoubtedly is a common and elementary computation unit in machine learning, computer vision and computer graphics. Krähenbühl and Koltun [2011] conducted efficient message passing in the fully connected CRFs inference by the high-dimensional Gaussian convolution. Elboer et al. [2013] expressed the generalized Laplacian distance for visual matching as cascaded convolutions. Paris and Durand [2009] converted the bilateral filter [Tomasi and Manduchi, 1998] into convolution in an elevated high-dimensional space. However, the computational complexity for a d-D convolution (1) is proportional to r^d, where r denotes the radius of the box filtering window Ω, $\mathcal{K}_{pq}$ represents the weight between p and q, $\mathcal{I}_p$ and $\mathcal{I}'_p$ are the values of input $\mathcal{I}$ and output $\mathcal{I}'$ at p. Therefore the running cost will become unacceptable for large r or d.

$$\mathcal{I}'_p = (\mathcal{K} * \mathcal{I})_p = \sum_{q \in \Omega_p} \mathcal{K}_{pq} \mathcal{I}_q \tag{1}$$

A lot of work was devoted to solving the computational shortcoming. But most of them focus on the Gaussian filtering. This is because not only the Gaussian convolution itself serves as building blocks

[*]Corresponding Author.

for many algorithms [Baek and Jacobs, 2010, Yang et al., 2015] but also its acceleration approaches play important roles in defocus [Barron et al., 2015], segmentation [Gadde et al., 2016], edge-aware smoothing [Barron and Poole, 2016], video propagation [Jampani et al., 2017].

In the literature, the most popular Gaussian blur acceleration algorithm should be the Splatting-Blurring-Slicing pipeline (SBS), which is first proposed by Paris and Durand [2006], Adams et al. [2010] coined its current name. We attribute its success to *data reduction* and *separable blurring*. In SBS, pixels are "splatted"(downsampled) onto the grid to reduce the data size, then those vertexes are blurred, finally the filtered values for each pixel are produced via "slicing"(upsampling). Due to the separable blurring kernel, the d-D Gaussian blurring performed on those vertexes can be deemed as a sum of separable 1-D filters [Szeliski, 2011] and therefore the computational complexity per pixel is reduced from $O(r^d)$ to $O(rd)$. As the filtering window becomes small after splatting, the computational complexity can be roughly viewed as $O(d)$ which is irrelevant to the radius r.

According to our investigation SBS has two problems: *1, how to approximate non-Gaussian blur?* SBS is designed for the Gaussian convolution. However, the requirements for non-Gaussian blurs emerge from local Laplacian filtering [Aubry et al., 2014] and mean-field inference [Vineet et al., 2014] recently. *2, how to improve the approximation error?* Previous SBS based methods just claim that their results are good approximations for the Gaussian filtering and prove this by experiments. Since current SBS has drawbacks, how can we generalize SBS to get a better result?

We recast SBS as a neural network (AccNet) to address above two problems in this paper. The benefits are threefold: 1, our AccNet offers a unified perspective for SBS based acceleration methods; 2, the layer weights together with the activation function g define the splatting, blurring and slicing convolution. So we can easily derive new splatting, blurring and slicing operations from the trained network for arbitrary convolutions. This ability entitles our network the End-to-End feature; 3, the optimal approximation error is guaranteed by AccNet in training.

2 Related Work

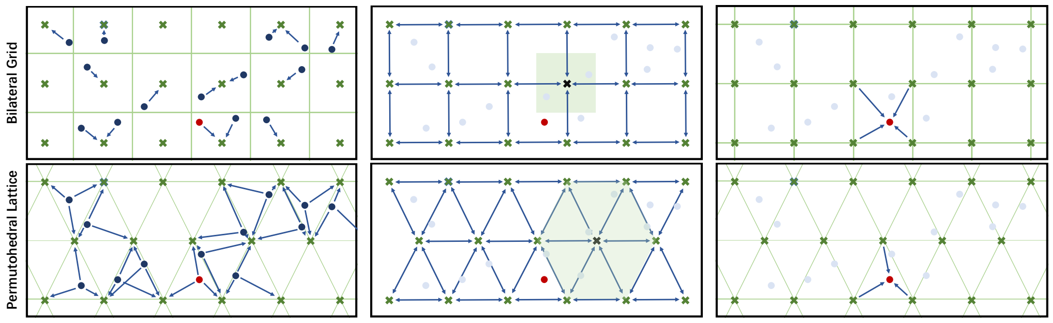
Few papers discussed acceleration algorithms for general high-dimensional convolution. Szeliski [2011] recorded a separable filtering method by SVD. Extending SVD to high-dimensional cases, we can generalize the separable filtering to high-dimensional convolution. In bilateral filtering literatures [Chaudhury and Dabhade, 2016, Dai et al., 2016], shiftable functions are exploited to approximate 1-D range kernels. This technique can also be extended to high-dimensional cases via outer product. However, its approximation terms are same as the separable filtering method and will exponentially increase with the dimension.

Current interest for fast high-dimensional convolution algorithms limits to Gaussian blur. Greengard and Strain [1991] provided the first fast Gaussian blur algorithm. Since the inception of the bilateral filter (BF) [Tomasi and Manduchi, 1998], the study for fast Gaussian convolution emerges in computer vision and computer graphics. Durand and Dorsey [2002] computed intermediate filtered images and synthesized final results by interpolation. The same approach was adopted by Porikli [2008] and Yang et al. [2009]. Paris and Durand [2006] implemented the first SBS which hints at more general approaches (bilateral grid and permutohedral lattice).

The bilateral grid [Chen et al., 2007] is a dense data structure that voxelizes the input space into a regular hypercubic lattice. By embedding inputs within the discretized space (splatting), they mix the values with a conventional Gaussian blur (blurring). The output image is extracted by resampling back into image space (slicing). The permutohedral lattice [Adams et al., 2010] is a sparse lattice that tessellates the space with simplices. By exploiting the fact that the number of vertices in a simplex grows slowly, it avoids the exponential growth of runtime that the bilateral grid suffers.

3 Design by Training for Fast High-Dimensional Convolution

Different from traditional output-focused neural networks, our AccNet implements the design-by-training strategy to automatically produce fast convolution pipeline and thus only interests in the activation function and weights as they define new splatting, blurring and slicing operations. In the following sections, we discuss how to transform the SBS into an AccNet as well as extensions for AccNets.



(a) Splatting (b) Blurring (c) Slicing

Figure 1: The splatting-blurring-slicing pipeline demonstration for bilateral grid and permutohedral lattice. The bilateral grid accumulates input values on a grid and factors the Gaussian-weighted gather into a separable Gaussian blur followed by multilinear sampling. The permutohedral lattice operates uses the permutohedral lattice. Barycentric weights within each simplex are used to resample into and out of the lattice. The separable blur is conducted along each axis.

3.1 Splatting, Blurring and Slicing Operations as Convolutions

Splatting voxelizes the space into a regular lattice and embeds inputs within the discretized vertices of the lattice to reduce the data size. Figure 1a illustrates the splatting operation of both bilateral grid and permutohedral lattice. The bilateral grid acceleration method trades accuracy for speed by accumulating constant values. The permutohedral lattice acceleration algorithm uses barycentric weights within each simplex to resample into the lattice. So the value of each vertice is the weighted sum of its nearby inputs. That is to say, the splatting operation conducts convolutions with a stride of s. Here s denotes the interval of lattice vertices.

Slicing as illustrated in Figure 1c is the inverse operation of splatting. SBS employs it to synthesize filtering results from the smoothed lattice values. The bilateral grid method does this by trilinear interpolation and the permutohedral lattice algorithm takes barycentric weights to resample out of the lattice. Since the slicing values are the weighted sum of neighbor vertices, the slicing operation equals to the convolution operation. Intuitively, slicing behaves as the deconvolution layer of the fully convolutional network [Shelhamer et al., 2017] which performs upsampling by convolution.

Blurring is an alias of convolution. In the d-D case, the full kernel implementation for a convolution requires r^d (multiply-add) operations per pixel, where r is the radius of the convolution kernel. This operation can be sped up by sequentially performing 1-D convolutions along each axis (which requires a total of dr operations per pixel) if the kernel is separable. Mathematically, a separable

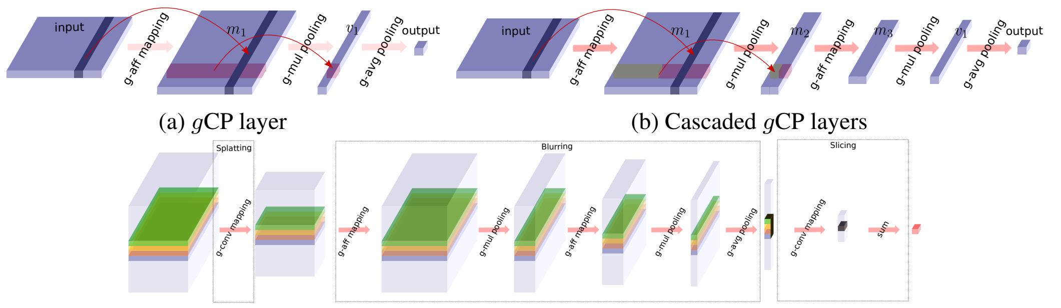
$$\boldsymbol{\mathcal{K}} = \boldsymbol{k}_1 \circ \boldsymbol{k}_2 \cdots \circ \boldsymbol{k}_d \tag{2}$$
$$\boldsymbol{\mathcal{I}}' = \boldsymbol{\mathcal{K}} * \boldsymbol{\mathcal{I}} = \boldsymbol{k}_1 * \boldsymbol{k}_2 \cdots \boldsymbol{k}_d * \boldsymbol{\mathcal{I}} \tag{3}$$

kernel $\boldsymbol{\mathcal{K}}$ is the rank-one tensor (the outer product of d vectors $\{\boldsymbol{k}_n, n = 1, \ldots, d\}$ (2)). Then the convolution with $\boldsymbol{\mathcal{K}}$ becomes (3). For arbitrary kernels, we can reformulate it as the sum of rank-one tensors by Canonical Polyadic (CP) decomposition [Sidiropoulos et al., 2017]. In this way, we have (4) and the computational complexity per pixel for the d-D case becomes $O(Ndr)$. Note that the

$$\boldsymbol{\mathcal{K}} = \sum_{i=1}^{N} w_i \circ \boldsymbol{k}_1^i \circ \boldsymbol{k}_2^i \cdots \circ \boldsymbol{k}_d^i \tag{4}$$

$$\boldsymbol{\mathcal{I}}' = \boldsymbol{\mathcal{K}} * \boldsymbol{\mathcal{I}} = \sum_{i=1}^{N} w_i \cdot \boldsymbol{k}_1^i * \boldsymbol{k}_2^i \cdots * \boldsymbol{k}_d^i * \boldsymbol{\mathcal{I}} \tag{5}$$

smoothing window usually is small after splatting, the computational complexity can be viewed as $O(Nd)$ which is irrelevant to r.



(a) gCP layer (b) Cascaded gCP layers

(c) AccNet

Figure 2: Demonstration for gCP layer, cascaded gCP layers and AccNet. The inputs of (a) (b) are matrices formed by $[\boldsymbol{l}_1^{\boldsymbol{p}_i}, \dots, \boldsymbol{l}_d^{\boldsymbol{p}_i}]$ (refer to section 3.2.2) and their outputs are scalars. The color cube in (c) stands for $\boldsymbol{\mathcal{L}}^j$ (refer to section 3.2.4) and the color slice in the cube represents $\boldsymbol{L}_{\boldsymbol{p}_i}^j$, where the outputs of (a)(b)(c) are scalars, the stripes in (a)(b) and the slices in (c) with the same color present the input-output relationship.

3.2 Design by Training Acceleration Network (AccNet)

Essentially, our design-by-training approach is to decompose the filtering kernel (a tensor) by neural networks because a convolution can be fast computed according to (5) once (4) is obtained. In both equations, basic building blocks are multiplication and addition. If one of them is substituted by other operations, we obtain new CP decomposition (4) and separable convolution (5). That is to say, we get new kinds of splatting, blurring and slicing operations. In this section we follow the way of Cohen and Shashua [2016] to generalize (4) to $\boldsymbol{g}$CP decomposition and provide corresponding g-convolution. The gCP layer and cascaded $\boldsymbol{g}$CP layers are proposed for gCT and gHT decompositions.

3.2.1 $\boldsymbol{g}$CP ($\boldsymbol{g}$ Canonical Polyadic) Decomposition & $\boldsymbol{g}$-Convolution

In (4) each element $\boldsymbol{\mathcal{K}}_{j_1,j_2,\dots,j_d}$ is formulated as $\sum_{i=1}^N w_i \boldsymbol{k}_{1,j_1}^i \boldsymbol{k}_{2,j_2}^i \cdots \boldsymbol{k}_{d,j_d}^i$. Assuming the activation function $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ denotes multiplication, we have $\boldsymbol{k}_{1,j_1}^i \boldsymbol{k}_{2,j_2}^i \cdots \boldsymbol{k}_{d,j_d}^i = \boldsymbol{k}_{1,j_1}^i \times_g \boldsymbol{k}_{2,j_2}^i \times_g \cdots \times_g \boldsymbol{k}_{d,j_d}^i$, where $a \times_g b = g(a,b) = ab$. Let g be an activation function such that $\forall a,b,c \in \mathbb{R} : g(g(a,b),c) = g(a,g(b,c)), g(a,b) = g(b,a)$, the tensor decomposition (4) can be generalized by defining $\boldsymbol{\mathcal{K}}_{j_1,j_2,\dots,j_d} = \sum_{i=1}^N w_i \times_g \boldsymbol{k}_{1,j_1}^i \times_g \cdots \times_g \boldsymbol{k}_{d,j_d}^i$. So we have gCP decomposition (6), where $\circ_g$ denotes the generalized outer product by replacing multiplication with the activation function g.

$$\boldsymbol{\mathcal{K}} = \sum_{i=1}^{N} w_i \times_g \boldsymbol{k}_1^i \circ_g \boldsymbol{k}_2^i \cdots \circ_g \boldsymbol{k}_d^i \tag{6}$$

Further, we substitute g for multiplication in (1) and obtain the g-convolution (7).

$$\boldsymbol{\mathcal{I}}'_{\boldsymbol{p}} = (\boldsymbol{\mathcal{K}} *_g \boldsymbol{\mathcal{I}})_{\boldsymbol{p}} = \sum_{\boldsymbol{q} \in \Omega_{\boldsymbol{p}}} \boldsymbol{\mathcal{K}}_{\boldsymbol{pq}} \times_g \boldsymbol{\mathcal{I}}_{\boldsymbol{q}} \tag{7}$$

Applying (6) to (7), we get (8) which sequentially performs N 1-D g-convolutions.

$$\boldsymbol{\mathcal{I}}' = \boldsymbol{\mathcal{K}} *_g \boldsymbol{\mathcal{I}} = \sum_{i=1}^{N} w_i \times_g \boldsymbol{k}_1^i *_g \boldsymbol{k}_2^i \cdots *_g \boldsymbol{k}_d^i *_g \boldsymbol{\mathcal{I}} \tag{8}$$

3.2.2 gCP Layer as $\boldsymbol{g}$CP Decomposition

$\boldsymbol{\mathcal{K}}_{\boldsymbol{pq}}$ and $\boldsymbol{\mathcal{I}}_{\boldsymbol{q}}$ in (7) form two d-order tensors. Taking $\hat{\boldsymbol{\mathcal{K}}}$ and $\hat{\boldsymbol{\mathcal{I}}}$ to denote them and putting (6) into (8), we have (9). Letting $\boldsymbol{l}_j$ be a vector and putting $\hat{\boldsymbol{\mathcal{I}}}_{v_1,\dots,v_d} = \prod_{j=1}^d \boldsymbol{l}_{j,v_j} = \boldsymbol{l}_{j,v_1} \times_g \boldsymbol{l}_{j,v_2} \times_g \cdots \times_g \boldsymbol{l}_{j,v_d}$ into (9), we obtain (10) which is consisted of three operations: 1, the g-affine mapping (g-aff mapping) defined by $\sum_v \boldsymbol{k}_{j,v}^i \times_g \boldsymbol{l}_{j,v}$; 2, the g-multiplication pooling (g-mul pooling) described by $\prod_{j=1}^d$; 3, the weighted average pooling (g-avg pooling) given by $\sum_{i=1}^N w_i$. The activation function g introduces

$$\begin{aligned}\boldsymbol{\mathcal{I}}'_{\boldsymbol{p}} = (\boldsymbol{\mathcal{K}} *_g \boldsymbol{\mathcal{I}})_{\boldsymbol{p}} &= \sum_{j_1,\dots,j_d} \hat{\boldsymbol{\mathcal{K}}}_{j_1,\dots,j_d} \times_g \hat{\boldsymbol{\mathcal{I}}}_{j_1,\dots,j_d} \\ &= \sum_{v_1,\dots,v_d} \sum_{i=1}^{N} w_i \times_g \prod_{j=1}^{d} \boldsymbol{k}^i_{j,v_j} \times_g \hat{\boldsymbol{\mathcal{I}}}_{v_1,\dots,v_d}\end{aligned} \tag{9}$$

$$\hat{\boldsymbol{\mathcal{I}}}'_{\boldsymbol{p}} = \sum_{i=1}^{N} w_i \times_g \prod_{j=1}^{d} \sum_{v} \boldsymbol{k}^i_{j,v} \times_g \boldsymbol{l}_{j,v} \tag{10}$$

nonlinearity to the three operations. Figure 2a plots the architecture, where the input is a matrix, the g-aff mapping transforms $\boldsymbol{l}_{j,v}$ denoted by the black line in the input to a new black vector in $\boldsymbol{m}_1$, the g-mul pooling maps each red vector in $\boldsymbol{m}_1$ to a scaler in vector $\boldsymbol{v}_1$ and the g-avg pooling reduces the element number of $\boldsymbol{v}_1$ to 1. In fact, the three operations belong to two categories. the g-avg pooling just is a special case of g-aff mapping. At last, we coin this layer as gCP layer as it implements the gCP decomposition for $\boldsymbol{\mathcal{K}}$.

3.2.3 Cascaded gCP Layers as gHT (g Hierarchical Turker) Decomposition

The expressive power of neural network has a close connection with the depth of layers. We cascade multiple gCP layers to extend the expressive ability in this section. The gCP layer maps a matrix to a scalar. As illustrated in Figure 2a, the g-aff mapping changes the element number of each red fiber, the g-mul pooling reduces the number of channels to 1 and the g-avg pooling decreases the element number of $\boldsymbol{v}_1$ to 1. If we replace the global pooling in the g-mul pooling by the local pooling, the output will become a matrix. Similarly, if we increase the output number of the last operation (the g-avg pooling is turned to the g-aff mapping), the output will be a vector. In this way, the gCP layer maps a matrix to another matrix and we can cascade two CP-layers together. Figure 2b provides a demo of two cascaded gCP layers, where the last g-aff mapping of the first gCP layer and the first g-aff mapping of the second gCP layer are merged as one g-aff mapping.

Cascaded gCP layers implement g hierarchical tucker decomposition [Hackbusch and Kühn, 2009], which replaces the multiplication by g in hierarchical tucker decomposition as we do for gCP. For example, a g hierarchical turker decomposition for a 4-order tensor $\boldsymbol{\mathcal{K}}$ with two layers can

$$\boldsymbol{\mathcal{K}} = \sum_{m=1}^{N_2} w_m \times_g \prod_{n=1}^{2} \boldsymbol{\mathcal{K}}^m_n \quad \text{with} \quad \boldsymbol{\mathcal{K}}^m_n = \sum_{i=1}^{N_1} w^m_{ni} \times_g \prod_{j=1}^{2} \boldsymbol{k}^{mi}_{nj} \tag{11}$$

be expressed as (11). Put (11) into convolution formula, we have (12). Comparing (12) to the

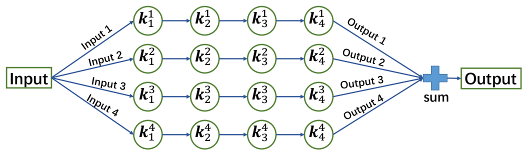
$$\boldsymbol{\mathcal{I}}'_{\boldsymbol{p}} = \sum_{m=1}^{N_2} w_m \times_g \prod_{n=1}^{2} \hat{\boldsymbol{\mathcal{I}}}^m_n \quad \text{with} \quad \hat{\boldsymbol{\mathcal{I}}}^m_n = \sum_{i=1}^{N_1} w^m_{ni} \times_g \prod_{i=1}^{2} \sum_{v} \boldsymbol{k}^{mi}_{nj,v} \times_g \boldsymbol{l}_{j,v} \tag{12}$$

architecture in Figure 2b, we can find that the operators $\sum_v \boldsymbol{k}^{mi}_{nj,v}$, $\prod_{i=1}^{2}$, $\sum_{i=1}^{N_1} w^m_{ni}$, $\prod_{n=1}^{2}$ and $\sum_{m=1}^{N_2} w_m$ corresponds the first g-aff mapping and g-mul pooling, the second g-aff mapping and g-mul pooling, the g-avg pooling, respectively.

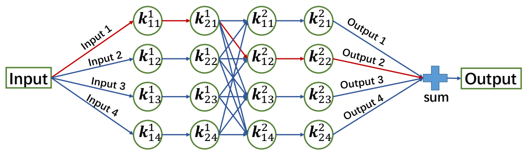
3.2.4 Proposed AccNet

Input: in sections (3.2.2) (3.2.3) we assumed $\boldsymbol{\mathcal{I}}_{\boldsymbol{p}_i} = \boldsymbol{l}^{\boldsymbol{p}_i}_1 \circ_g \cdots \circ_g \boldsymbol{l}^{\boldsymbol{p}_i}_d$ and form the matrix $[\boldsymbol{l}^{\boldsymbol{p}_i}_1, \cdots, \boldsymbol{l}^{\boldsymbol{p}_i}_d]$ as the network input for each point $\boldsymbol{p}_i$. To relax this assumption, we suppose $\boldsymbol{\mathcal{I}}_{\boldsymbol{p}_i} = \sum_{j=1}^{l} \boldsymbol{\mathcal{I}}^j_{\boldsymbol{p}_i}$ and $\boldsymbol{\mathcal{I}}^j_{\boldsymbol{p}_i} = \boldsymbol{l}^{\boldsymbol{p}_i}_{1j} \circ_g \cdots \circ_g \boldsymbol{l}^{\boldsymbol{p}_i}_{dj}$. The blurring value of each vertice $\boldsymbol{p}_i$ on the bilateral grid or permutohedral lattice depends on the values of its neighborhoods (an image batch $\boldsymbol{\mathcal{I}}_{\boldsymbol{p}_i}$). For slicing, we need m vertices $\boldsymbol{p}_i$ surrounding the target point to interpolate its filtering result. So total m image batches $\{\boldsymbol{\mathcal{I}}_{\boldsymbol{p}_i}, 1 \le i \le m\}$ are required to compute the results of target points encircled by $\{\boldsymbol{p}_i, 1 \le i \le m\}$. To synthesize filtering values of target points encircled by $\{\boldsymbol{p}_i\}$, we compose $\boldsymbol{\mathcal{L}}^j$ by concatenating $\{\boldsymbol{L}^j_{\boldsymbol{p}_i} = [\boldsymbol{l}^{\boldsymbol{p}_i}_{1j}, \cdots, \boldsymbol{l}^{\boldsymbol{p}_i}_{dj}], 1 \le i \le m\}$ vertically. Further $\{\boldsymbol{\mathcal{L}}^j, 1 \le j \le l\}$ are stacked together and serves as our AccNet input. Figure 2c illustrates this, where color regions denote different parts $\{\boldsymbol{L}^j_{\boldsymbol{p}_i}\}$ of $\boldsymbol{\mathcal{L}}^j$ and the light cube represents the 3-order input tensor.

Splatting: the splatting layer conducts the strided convolution. Theoretically, the convolution kernel $\boldsymbol{\mathcal{K}}$ is arbitrary. Here, we assume $\boldsymbol{\mathcal{K}} = \boldsymbol{k}_1 \circ_g \cdots \circ_g \boldsymbol{k}_d$ is a rank-one tensor in AccNet due to the



(a) gCP convolution



(b) gHT convolution

Figure 3: Illustration for fast filtering approaches based on gCP and gHT decompositions. Taking different tensor decomposition methods for the filtering kernel, we achieve different fast filtering algorithms. (a) plots the computation graph of fast filtering scheme (7) for the gCP decomposition $\boldsymbol{\mathcal{K}} = \sum_{i=1}^{4} \prod_{j=1}^{4} \boldsymbol{k}_j^i$. The path indicated by arrows presents the convolution sequence with kernels $\{\boldsymbol{k}_j^i\}$. Final result is the sum of the outputs of all 4 paths. (b) shows the flow chart of fast filtering scheme (12) for the gHT decomposition $\boldsymbol{\mathcal{K}} = \prod_{m=1}^{2} \boldsymbol{\mathcal{K}}_m$ with $\boldsymbol{\mathcal{K}}_m = \sum_{j=1}^{4} \prod_{i=1}^{2} \boldsymbol{k}_{ij}^m$. Each input connects to four outputs and thus produces four outputs. The red line indicates a convolution path. Final result is the sum of the outputs of all 16 paths.

reasons: 1, AccNet takes three layers to approximate the convolution result. Even though the splatting layer is simple, the approximation error can be reduced by increasing the complexity of the blurring layer; 2, each slice of the input tensor of the blurring layer must be a rank-one matrix and the filtering result $\boldsymbol{\mathcal{I}}'^j_{\boldsymbol{p}_i}$ with a rank-one kernel $\boldsymbol{\mathcal{K}}$ for a rank-one input tensor $\boldsymbol{\mathcal{I}}^j_{\boldsymbol{p}_i} = \boldsymbol{l}^{j,1}_{\boldsymbol{p}_i} \circ_g \cdots \circ_g \boldsymbol{l}^{j,d}_{\boldsymbol{p}_i}$ is also a rank-one tensor.

Blurring: we prefer to employ cascaded $\boldsymbol{g}$CP-layers to compose the blurring layer of AccNet as it has more powerful expressive ability than single $\boldsymbol{g}$CP-layer. Figure 2c provides a two $\boldsymbol{g}$CP-layers example. For each slice $\boldsymbol{L}^j_{\boldsymbol{p}_i} = [\boldsymbol{l}^{\boldsymbol{p}_i}_{1j}, \cdots, \boldsymbol{l}^{\boldsymbol{p}_i}_{dj}]$, the blurring layer produces a scalar value $z^j_{\boldsymbol{p}_i}$.

Slicing: let $\boldsymbol{z}^j = [z^j_{\boldsymbol{p}_1}, \ldots, z^j_{\boldsymbol{p}_m}]$, the slicing layer maps $\boldsymbol{z}^j$ to a vector $\boldsymbol{t}^j$, where each element of $\boldsymbol{t}^j$ corresponds to the interpolated values of the pixels surrounded by $\{\boldsymbol{p}_i\}$ and the value of each $\boldsymbol{p}_i$ are from $\boldsymbol{\mathcal{I}}^j_{\boldsymbol{p}_i}$. Since $\boldsymbol{\mathcal{I}}_{\boldsymbol{p}_i} = \sum_{j=1}^{l} \boldsymbol{\mathcal{I}}^j_{\boldsymbol{p}_i}$, there are total l different $\boldsymbol{z}^j$ and therefore we obtain l different $\boldsymbol{t}^j$. The final result is the sum of $\{\boldsymbol{t}^j, 1 \leq j \leq l\}$.

***g* function:** the function plays an important role in our AccNet. First, it introduces nonlinearity to AccNet. This strengthens the expressive power of our AccNet. Second, it defines new convolutions. Employing g-conv operation, we can easily define novel splatting, blurring and slicing operations. There are many possible $\boldsymbol{g}$ functions meeting the associativity $g(g(a,b),c) = g(a,g(b,c))$ and commutativity $g(a,b) = g(b,a)$ requirements. Here we list two of them used in AccNet: 1, $g(a,b) = \max\{a,0\}\max\{b,0\}$; 2, $g(a,b) = \max\{ab,0\}$.

Gradients: The gradients of both sum and g function can be easily obtained. Therefore, AccNet as a composition of the two basis calculations can be easily trained by the back-propagation algorithm.

4 Approximation & Fast Filtering

In section 3, we discussed the layers of AccNet as well as the way to transform the SBS to an AccNet. Here, we describe an approach to compose an expressive powerful AccNet and to turn it back to SBS.

Expressive Powerful AccNet: the expressive power of AccNet determines the approximation error. We have two ways to increase this power. One is to introduce the nonlinear activation function to AccNet. Unlike traditional SBS taking the CP decomposition for acceleration, we implement gCP decomposition in AccNet. Another way is to make AccNet deeper. In this way, gCP becomes gHT. At last, we note that we can choose different activation functions in different layers. This is because splatting, blurring and slicing operations are essentially convolutions therefore we can take different gCTs and gHTs to accelerate their computation.

From AccNet to SBS: the weights as well as the activation function of three AccNet layers define the splatting, blurring and slicing kernels. The correspondences between AccNet weights and convolution kernels are determined by (7) (8) for the gCP decomposition and (11) (12) for the gHT decomposition. For easy understanding, we visualize the computation graph of an AccNet in Figure 3. Figure 3a

Table 1: Filtering accuracy comparison for the bilateral grid acceleration method (BG), the permutohedral lattice acceleration method (PL) and our AccNet, where the sampling period of splatting is 3, the radius of blurring is 1 and the radius of original convolution is 5.

	2D				3D				5D			
	$\sigma = 2$	$\sigma = 4$	$\sigma = 8$	$\sigma = 16$	$\sigma = 2$	$\sigma = 4$	$\sigma = 8$	$\sigma = 16$	$\sigma = 2$	$\sigma = 4$	$\sigma = 8$	$\sigma = 10$
BG	0.952	0.768	0.587	0.288	1.225	1.085	0.813	0.668	1.804	1.552	1.179	0.878
AccNet	0.309	0.249	0.165	0.054	0.336	0.276	0.267	0.171	0.853	0.465	0.349	0.259
PL	0.541	0.657	0.419	0.239	1.107	0.893	0.733	0.604	1.712	1.488	1.005	0.854
AccNet	0.273	0.175	0.142	0.051	0.381	0.243	0.203	0.153	0.528	0.423	0.299	0.213

takes the gCT decomposition $\mathcal{K} = \sum_{i=1}^{4} \prod_{j=1}^{4} \boldsymbol{k}_j^i$ to implement the fast convolution algorithm and Figure 3b records the fast convolution for the gHT decomposition $\mathcal{K} = \prod_{m=1}^{2} \sum_{j=1}^{4} \prod_{i=1}^{2} \boldsymbol{k}_{ij}^m$, where circles denote convolution operations with specific filtering kernels $\boldsymbol{k}$ and arrows indicate the computation order.

The two examples in Figure 3 disclose the superiority of gHT decomposition based acceleration algorithms. In Figure 3a, each convolution kernel is only used by one computation path. In contrast, the convolution kernel in Figure 3b is used by multiple times. The *reuse* advantages are twofold: 1, we can reduce the approximation error because more terms can be used to approximate original kernels; 2, we can reduce the execution time by reusing the convolution result sharing the same convolution node. For example, the filtering path $\boldsymbol{k}_{11}^1 \rightarrow \boldsymbol{k}_{21}^1 \rightarrow \boldsymbol{k}_{11}^2 \rightarrow \boldsymbol{k}_{21}^2$ and $\boldsymbol{k}_{11}^1 \rightarrow \boldsymbol{k}_{21}^1 \rightarrow \boldsymbol{k}_{12}^2 \rightarrow \boldsymbol{k}_{22}^2$ share the filtering results of $\boldsymbol{k}_{11}^1 \rightarrow \boldsymbol{k}_{21}^1$.

5 Experiments

AccNet is the first neural network producing fast convolution algorithms. To reveal its advantages, three experiments are conducted: 1, we compare our AccNet designed acceleration method to the handmade bilateral grid and permutohedral lattice acceleration methods; 2, we provide a new neural network to automatically design fast algorithm and compare it to AccNet; 3, we employ AccNet to design new acceleration algorithms for non-Gaussian convolution and demonstrate their applications. In the following experiments, the blurring layer of AccNet is composed by two cascaded gCP layers and the activation function is $g(a, b) = \max(ab, 0)$.

Fast Gaussian convolution comparison: Both bilateral grid acceleration method (BG) and permutohedral lattice acceleration method (PL) are designed for fast Gaussian convolution. The major difference between them is the underlying grid. Our AccNet can be applied to both bilateral grid and permutohedral lattice. To illustrate the filtering accuracy of the methods produced by AccNet, we keep their convolution number same to BG and PL and evaluate their filtering accuracy. Table 1 records the quantitative comparison results, where the first row denotes the dimension of the Gaussian kernel, σ denotes the bandwidth of kernel, the accuracy is measured by MSE (the mean-square error), the first two rows record the results of BG and AccNet on the bilateral grid and the last two rows plot the results of PL and AccNet on the permutohedral lattice.

Acceleration network comparison: SBS sequentially conducts three convolutions. We can turn it to a CNN with three layers and further transform each CNN layer to d cascaded 1-D convolution according to the CP decomposition (4) (5). The differences between this network and our AccNet are that: 1, the depth of each layer of this CNN model is proportional to the dimension of filtering kernel. In contrast, the layer depth of AccNet only depends on the desired expressive power of the layer and the expressive power of the simplest AccNet layer equal to the expressive power of CNN layer. 2, the CNN model is hard to express the gHT decomposition (11) as its straightforward processing pipeline is similar to Figure 3a and could not reuse intermediate results as AccNet does in Figure 3b.

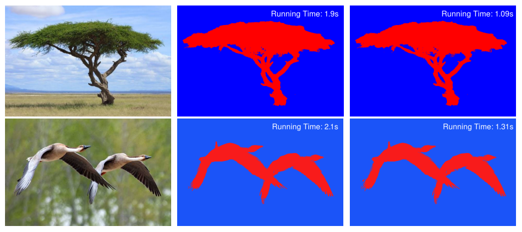
The first shortcoming makes the CNN model deeper for high-dimensional convolution. We thus have to spend more time to tweak it. What's worse, the depth does not increase the expressive power of

Table 2: Two acceleration neural networks (CNN and AccNet) comparisons. The bandwidth of target Gaussian kernel is 5 and the underlying lattice is the bilateral grid.

	2D		3D		5D	
	Filtering Error	Training Time	Filtering Error	Training Time	Filtering Error	Training Time
CNN	0.245	12.5h	0.283	13.1h	0.473	14h
AccNet	0.239	7.2h	0.271	7.3h	0.461	7.6h

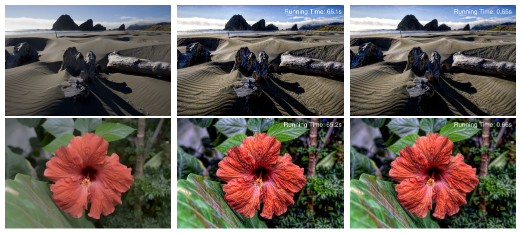
this model because its expressive power is determined by the number N of cascaded 1-D convolution pipelines. The second weakness causes its inferiority of the expressive power when we limits its convolution number equal to AccNet. This usually means larger filtering errors in filtering. To prove these, we plot Table 2 which records the training time as well as the filtering error measured by MSE, where the dimension of filtering kernel varies from 2-D to 5-D.

Fast non-Gaussian filtering: Non-Gaussian blur becomes popular recently. To illustrate the power of our AccNet, we demonstrate three applications of fast non-Gaussian filtering in machine learning, computer vision and computer graphics, respectively.



(a) Input (b) Krähenbühl (c) Ours

Figure 4: Pixel-level segmentation results of two fully connected CRF implementations. (a) is input images. (b) is the segmentation results of Krähenbühl. (c) records our segmentation results.



(a) Input (b) Paris (c) Ours

Figure 5: Detail enhancement of two local Laplace filtering implementations. (a) is input images. (b) is the filtering results of Paris. (c) denotes our results.

Table 3: Stereo matching quantitative comparison.

	All			NoOcc		
	bad 1%	MAE	RMS	bad 1%	MAE	RMS
[Zbontar and LeCun, 2015]	20.07	5.93	18.36	10.42	1.94	9.07
[Barron and Poole, 2016]	19.49	2.81	8.44	11.33	1.40	5.23
Ours	19.21	2.13	7.79	10.41	1.34	4.96

CRF inference: The pairwise edge potentials used in the fully connected CRFs [Krähenbühl and Koltun, 2011] is the Gaussian mixture kernels. Krähenbühl and Koltun [2011] provided a highly efficient approximate inference algorithm by showing a mean field update of all variables in a fully connected CRF can be performed using Gaussian filtering in the feature space. In order to speed up the computation via the separability of the Gaussian kernel G_i, Krähenbühl has to perform multiple times Gaussian filtering. Employing AccNet, we can accelerate the Gaussian mixture kernels directly. Compared to the original method, we save 60% of the time while producing the same segmentation results as shown in Figure 4.

Bilateral solver: Bilateral solver [Barron and Poole, 2016] allows for some optimization problems with bilateral affinity terms to be solved quickly, and also guarantees that the solutions are smoothed within objects, but not smooth across edges. Although the prior used by bilateral solver is arbitrary in theory, bilateral solver can only take the Gaussian function as it is the only function can be presented by SBS before our work. Here we take the smooth exponential family prior [Zhang and Allebach, 2008] to construct non-Guassian bilateral solver and apply it stereo post-processing procedure of MC-CNN [Zbontar and LeCun, 2015] following the way of [Barron and Poole, 2016]. In Table 3, we record the quantitative results, where "bad 1%" presents the percent of pixels whose disparities are wrong by more than 1, "MAE" stands for the mean absolute error and "RMS" is the root mean square error.

Local Laplace filtering: Local Laplacian filter [Paris et al., 2011] is an edge-aware operator that defines the output image $\bar{\boldsymbol{I}}$ by constructing its Laplacian pyramid $\{L[\bar{\boldsymbol{I}}]\}$ coefficient by coefficient. Aubry et al. [2014] present the Laplacian coefficient at level l and position $\boldsymbol{p}$ as the nonlinear convolution $\{L_l[\bar{\boldsymbol{I}}](\boldsymbol{p})\} = \sum_{\boldsymbol{q}\in\Omega_{\boldsymbol{p}}} D_l(\boldsymbol{q}-\boldsymbol{p})f(\boldsymbol{I_q}-g)(\boldsymbol{I_q}-g)$, where f is a continuous function, D_l is the difference-of-Gaussians filter defining the pyramid coefficients at level l and g is the coefficient of the Gaussian pyramid at $(l,\boldsymbol{p})$. Obviously, this convolution can be accelerated by

AccNet and achieves speed-ups on the order of 100 times. Figure 5 visualizes the similar detail enhancement results of Paris and ours.

6 Conclusion

In this paper, we propose the first neural network producing fast high-dimensional convolution algorithms. We take AccNet to express the approximation function of SBS and generalize SBS by changing the architecture of AccNet. Once training is finished, new fast convolution algorithm can be easily derived from the weights and activation functions of each layer. Experiments prove the effectiveness of our algorithm.

7 Acknowledgment

This work was supported by the 973 Program (Project No. 2014CB347600), the National Natural Science Foundation of China (Grant No. 61701235, 61732007, 61522203, 61772275, 61873293 and 61772524), the Fundamental Research Funds for the Central Universities (Grant No. 30917011323) and the Beijing Municipal Natural Science Foundation (Grant No. 4182067).

References

Andrew Adams, Jongmin Baek, and Myers Abraham Davis. Fast high-dimensional filtering using the permutohedral lattice. *Computer Graphics Forum*, 29(2):753–762, may 2010. 2

Mathieu Aubry, Sylvain Paris, Samuel W. Hasinoff, Jan Kautz, and Frédo Durand. Fast local laplacian filters: Theory and applications. *ACM Transactions on Graphics*, 33(5):1–14, sep 2014. 2, 8

Jongmin Baek and David E. Jacobs. Accelerating spatially varying gaussian filters. *ACM Transactions on Graphics*, 29(6):1, dec 2010. 2

Jonathan T. Barron and Ben Poole. The fast bilateral solver. In *European Conference on Computer Vision*, pages 617–632. Springer International Publishing, 2016. 2, 8

Jonathan T. Barron, Andrew Adams, YiChang Shih, and Carlos Hernandez. Fast bilateral-space stereo for synthetic defocus. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2015. 2

Kunal N. Chaudhury and Swapnil D. Dabhade. Fast and provably accurate bilateral filtering. *IEEE Transactions on Image Processing*, 25(6):2519–2528, jun 2016. 2

Jiawen Chen, Sylvain Paris, and Frédo Durand. Real-time edge-aware image processing with the bilateral grid. *ACM Transactions on Graphics*, 26(3):103, 2007. 2

Nadav Cohen and Amnon Shashua. Convolutional rectifier networks as generalized tensor decompositions. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 955–963, New York, New York, USA, 20–22 Jun 2016. PMLR. 4

Longquan Dai, Mengke Yuan, and Xiaopeng Zhang. Speeding up the bilateral filter: A joint acceleration way. *IEEE Transactions on Image Processing*, 25(6):2657–2672, jun 2016. 2

Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Transactions on Graphics*, 21(3):257–266, jul 2002. 2

Elhanan Elboer, Michael Werman, and Yacov Hel-Or. The generalized laplacian distance and its applications for visual matching. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2013. 1

Raghudeep Gadde, Varun Jampani, Martin Kiefel, Daniel Kappler, and Peter V. Gehler. Superpixel convolutional networks using bilateral inceptions. In *European Conference on Computer Vision*, pages 597–613. Springer International Publishing, 2016. 2

Leslie Greengard and John Strain. The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, jan 1991. 2

W. Hackbusch and S. Kühn. A new scheme for the tensor representation. *Journal of Fourier Analysis and Applications*, 15(5):706–722, oct 2009. 5

Varun Jampani, Raghudeep Gadde, and Peter V. Gehler. Video propagation networks. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, jul 2017. 2

Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected CRFs with gaussian edge potentials. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 109–117. Curran Associates, Inc., 2011. 1, 8

Sylvain Paris and Frédo Durand. A fast approximation of the bilateral filter using a signal processing approach. In *European Conference on Computer Vision*, pages 568–580. Springer Nature, 2006. 2

Sylvain Paris and Frédo Durand. A fast approximation of the bilateral filter using a signal processing approach. *International Journal of Computer Vision*, 81(1):24–52, dec 2009. 1

Sylvain Paris, Samuel W. Hasinoff, and Jan Kautz. Local laplacian filters: edge-aware image processing with a laplacian pyramid. *ACM Transactions on Graphics*, 30(4):1, jul 2011. 8

Fatih Porikli. Constant time o(1) bilateral filtering. In *IEEE Conference on Computer Vision and Pattern Recognition*. Institute of Electrical and Electronics Engineers (IEEE), jun 2008. 2

Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, apr 2017. 3

Nicholas D. Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E. Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, jul 2017. 3

Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag GmbH, 2011. ISBN 1848829345. 2

C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *IEEE International Conference on Computer Vision*. Narosa Publishing House, 1998. 1, 2

Vibhav Vineet, Jonathan Warrell, and Philip H. S. Torr. Filter-based mean-field inference for random fields with higher-order terms and product label-spaces. *International Journal of Computer Vision*, 110(3):290–307, mar 2014. 2

Qingxiong Yang, Kar-Han Tan, and Narendra Ahuja. Real-time o(1) bilateral filtering. In *IEEE Conference on Computer Vision and Pattern Recognition*. Institute of Electrical and Electronics Engineers (IEEE), jun 2009. 2

Qingxiong Yang, Narendra Ahuja, and Kar-Han Tan. Constant time median and bilateral filtering. *International Journal of Computer Vision*, 112(3):307–318, sep 2015. 2

Jure Zbontar and Yann LeCun. Computing the stereo matching cost with a convolutional neural network. In *IEEE Conference on Computer Vision and Pattern Recognition*. Institute of Electrical and Electronics Engineers (IEEE), jun 2015. 8

Buyue Zhang and J.P. Allebach. Adaptive bilateral filter for sharpness enhancement and noise removal. *IEEE Transactions on Image Processing*, 17(5):664–678, may 2008. 8