Neural Networks 143 (2021) 657–668

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Content-aware convolutional neural networks

Yong Guo [a,b,c], Yaofo Chen [a], Mingkui Tan [a,c,*], Kui Jia [a], Jian Chen [a,*], Jingdong Wang [d]

[a] South China University of Technology, China
[b] Pazhou Laboratory, China
[c] Key Laboratory of Big Data and Intelligent Robot, Ministry of Education
[d] Microsoft Research Asia, China

A R T I C L E I N F O

Article history:
Received 20 October 2020
Received in revised form 1 April 2021
Accepted 29 June 2021
Available online 12 July 2021

Keywords:
Convolution
Neural networks
Redundancy reduction

A B S T R A C T

Convolutional Neural Networks (CNNs) have achieved great success due to the powerful feature learning ability of convolution layers. Specifically, the standard convolution traverses the input images/features using a sliding window scheme to extract features. However, not all the windows contribute equally to the prediction results of CNNs. In practice, the convolutional operation on some of the windows (*e.g.*, smooth windows that contain very similar pixels) can be very redundant and may introduce noises into the computation. Such redundancy may not only deteriorate the performance but also incur the unnecessary computational cost. Thus, it is important to reduce the computational redundancy of convolution to improve the performance. To this end, we propose a Content-aware Convolution (CAC) that automatically detects the smooth windows and applies a 1×1 convolutional kernel to replace the original large kernel. In this sense, we are able to effectively avoid the redundant computation on similar pixels. By replacing the standard convolution in CNNs with our CAC, the resultant models yield significantly better performance and lower computational cost than the baseline models with the standard convolution. More critically, we are able to dynamically allocate suitable computation resources according to the data smoothness of different images, making it possible for content-aware computation. Extensive experiments on various computer vision tasks demonstrate the superiority of our method over existing methods.

1. Introduction

Recently, convolutional neural networks (CNNs) have achieved remarkable performance in many computer vision tasks, including image classification (Guo et al., 2020; He, Zhang, Ren, & Sun, 2016), face recognition (Ozawa, Toh, Abe, Pang, & Kasabov, 2005; Schroff, Kalenichenko, & Philbin, 2015; Sun, Wang, & Tang, 2015), semantic segmentation (Ibtehaz & Rahman, 2020; Liu et al., 2020; Shelhamer, Long, & Darrell, 2017), and object detection (Ren, He, Girshick, & Sun, 2017; Wang, Dai, Cai, Sun, & Chen, 2018). Moreover, deep CNNs have also become the workhorse of many other tasks and real-world applications beyond computer vision, such as speech recognition (Schrauwen, D'Haene, Verstraeten, & Van Campenhout, 2008; Skowronski & Harris, 2007) and natural language processing (Duch, Matykiewicz, & Pestian, 2008; Gross & Murthy, 2014).

One of the key factors behind the success of CNNs lies in the powerful feature learning ability of convolution layers. Typically, the standard convolution transforms the input images/features into a set of windows and exploits a sliding window manner to extract features over them (Burrus & Parks, 1985). However, not all the windows contribute equally to the prediction results of CNNs. As shown in Fig. 1, the input images/features often contain a lot of smooth windows that consist of very similar pixels. These windows may contain very limited information about the data (Bar-Hillel & Weinshall, 2008; Fergus, Perona, & Zisserman, 2003) since a similar pattern may also appear in the surrounding areas. As a result, the computation on smooth windows may be very redundant. More critically, performing convolution on these windows may also introduce noises into the computation and thus deteriorate the performance (see results in Tables 1 and 2). Thus, it is important and necessary to reduce the computational redundancy of convolution to improve the performance.

Regarding this issue, existing methods improve the convolutional operation by reducing the redundant communications among different channels of feature maps (Chollet, 2017; Krizhevsky, Sutskever, & Hinton, 2012; Sifre & Mallat, 2014) or reducing the spatial size of some of the redundant channels (Chen et al., 2019). Specifically, group convolution (Krizhevsky et al., 2012) and depthwise separable convolution (Chollet, 2017; Sifre & Mallat, 2014) divide the channels of feature maps into multiple groups and perform convolution independently over each

* Corresponding authors.
E-mail addresses: guo.yong@mail.scut.edu.cn (Y. Guo), sechenyaofo@mail.scut.edu.cn (Y. Chen), mingkuitan@scut.edu.cn (M. Tan), kuijia@scut.edu.cn (K. Jia), ellachen@scut.edu.cn (J. Chen), jingdw@microsoft.com (J. Wang).

https://doi.org/10.1016/j.neunet.2021.06.030
0893-6080/© 2021 Elsevier Ltd. All rights reserved.

group. Recently, Chen et al. propose the Octave Convolution (OctConv) (Chen et al., 2019) method, which downscales the feature maps in some redundant channels into smaller sizes to reduce the computational cost. However, these methods only focus on the redundancy inside the channels of feature maps but ignore the spatial redundancy of the pixels in each window. Moreover, existing methods perform the same computation on the samples with different spatial redundancy, which makes the prediction not optimal and also very inefficient.

In this paper, we seek to reduce the computational redundancy on smooth windows to improve the performance of convolution. To this end, we propose a Content-aware Convolution (CAC) that uses a 1×1 convolution to replace the computation of original $k \times k$ convolution on smooth windows. In this sense, we are able to effectively avoid the redundant computation to improve the feature learning ability of convolution. To obtain the weights for the 1×1 convolutional kernels, we spatially aggregating the $k \times k$ kernel by summing up all the kernel parameters (see detailed analysis in Section 4.2). In order to automatically detect the smooth windows, we propose an effective training method that seeks for a trade-off between model performance and computational cost. More critically, CAC dynamically allocates computation resources for different samples based on the smoothness of their contents. Therefore, we are able to perform content-aware computation. In practice, our CAC models yield significantly better performance and lower computational cost than the models with the standard convolution. Extensive experiments on different computer vision tasks demonstrate the superiority of our method over existing methods.

In this paper, we make the following contributions.

- We propose a Content-aware Convolution (CAC) method that replaces the original $k \times k$ kernel with a 1×1 kernel on the smooth windows to improve the performance of convolution. With CAC, we are able to effectively reduce the computational redundancy of convolution and significantly improve the performance.
- We propose an effective training method to automatically detect the smooth windows for each layer. To achieve this goal, we solve a multi-objective optimization problem to find a trade-off between model performance and computational cost.
- Equipped with CAC, the resultant models achieve content-aware computation by dynamically allocating computation resources to different samples according to the data smoothness. Extensive experiments on different computer vision tasks demonstrate the effectiveness of the proposed method.

2. Related work

Recently, much effort has been made to reduce the redundancy of deep networks, including channel pruning, network quantization, and energy-efficient model design.

2.1. Channel pruning

Channel pruning is one of the predominant approaches for deep network compression. Li, Kadav, Durdanovic, Samet and Graf (2017) measure the importance of different channels by computing the sum of absolute values of weights to conduct channel selection. Hu, Peng, Tai, and Tang (2016) use the average percentage of zeros (APoZ) to select important channels. Several training based methods (Alvarez & Salzmann, 2016; Liu et al., 2017) have been proposed to automatically identify the redundant channels by introducing a sparsity regularizer in the training objective. The reconstruction methods (He, Zhang, & Sun, 2017; Luo, Wu, & Lin, 2017) seek to solve the channel pruning problem by minimizing the reconstruction error between the feature maps of the pretrained model and the compressed model. Recently, Zhuang et al. (2018) propose a discrimination-aware channel pruning (DCP) method to choose the channels that contribute to the discriminative power and obtain state-of-the-art results. Based on DCP, Liu et al. (2020) further propose a discrimination-aware kernel pruning (DKP) method by removing the redundant kernels according to the discrimination power. However, these methods only focus on the redundancy in model parameters but ignore the redundancy incurred by the input data.

2.2. Network quantization

Network quantization aims to convert the pretrained full-precision convolution networks into the low-precision versions to reduce the computational cost. Recently, Han, Mao, and Dally (2016) propose a three-stage deep compression pipeline, including pruning, trained quantization, and Huffman coding. DoReFa-Net (Zhou et al., 2016) seeks to quantize the full precision weights, activations, and gradients to the low bit ones for deep networks. In ternary weight networks (TWNs) (Li, Zhang, & Liu, 2016; Zhu, Han, Mao, & Dally, 2017), the parameters are constrained to $+1$, 0, and -1, and the model achieves higher accuracy than binary neural networks. Similar to channel pruning, network quantization also reduces the redundancy in model parameters and may yield limited performance.

2.3. Energy-efficient model design

Many energy-efficient modules have been proposed to reduce the computational cost of deep networks. Specifically, sparse convolution (Graham, Engelcke, & van der Maaten, 2018; Liu, Wang, Foroosh, Tappen, & Pensky, 2015) zeros out a large number of parameters to reduce the model size. Group convolution (Krizhevsky et al., 2012) and depthwise separable convolution (Sifre & Mallat, 2014) divide the input channels into groups to reduce the redundant communications among different groups. NAT (Guo et al., 2019) replaces redundant operations with identity mapping or directly remove them to obtain efficient models.

Related to our method, Li, Liu, Luo, Change Loy and Tang (2017) propose a Region Convolution (RC) that reduces the computational redundancy for semantic segmentation models. Specifically, given an input feature map, RC performs convolutions on the regions of hard pixels and discards the easy pixels according to the confidence of the predicted mask. However, it has some underlying limitations. First, RC relies on the predicted confidence of all pixels to construct the mask and cannot be applied to tasks without dense prediction, *e.g.*, image classification. Second, RC completely discards the easy regions and may influence the features learned in the deeper layers. Unlike RC, our CAC preserves the information in all regions/pixels and can be applied to most computer vision tasks, *e.g.*, image classification, semantic segmentation, and object detection.

Very recently, Chen et al. propose the octave convolution (OctConv) (Chen et al., 2019) method, which reduces the spatial resolution of some low-frequency feature maps to reduce the computational complexity. However, it has two major limitations. First, the low-frequency feature maps are predefined before training rather than detected according to the input data. Second, OctConv only considers the redundancy in the channels of the feature maps but ignores the redundancy in the data content, *e.g.*, pixels. Compared to OctConv, our CAC automatically detects the sharp/smooth windows from the input data to achieve content-aware computation. Moreover, CAC considers the pixel level redundancy caused by the input data.

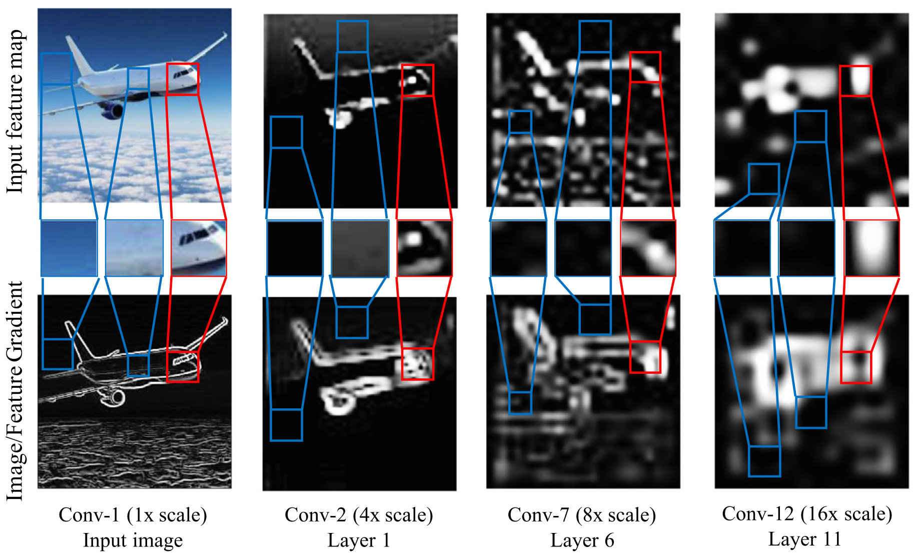

Fig. 1. Demonstration of computational redundancy in the input feature maps of different layers of ResNet18 (pretrained on ImageNet). The top row and bottom row show the input feature maps and the corresponding gradient in different layers, respectively. Red boxes denote the sharp windows that contain the main content of the image. Blue boxes denote the smooth windows that contain limited information about the image and require redundant computation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3. Notations and problem definition

In this paper, we assume that the convolution has the stride of 1 and is performed with padding to guarantee that the output feature maps have the same size as the input feature maps. In this paper, we consider the squared input images and feature maps. For simplicity, we consider one single-channel input feature map $\mathbf{X} \in \mathbb{R}^{n \times n}$ and one convolutional kernel $\mathbf{W} \in \mathbb{R}^{k \times k}$, where n and k denote the sizes of the feature maps and kernels, respectively. Thus, the standard convolution can be written as

$$\mathbf{Y} = \mathbf{X} \otimes \mathbf{W}, \tag{1}$$

where $\otimes$ denotes the convolutional operator.

In practice, the computation of convolution is often converted to the matrix–matrix or matrix–vector multiplication (Vasudevan, Anderson, & Gregg, 2017). Given a convolution with a $k \times k$ kernel and an input feature map $\mathbf{X} \in \mathbb{R}^{n \times n}$, there are n^2 windows that are convolved by the kernel (Ludwig, 2013). In this sense, we can represent $\mathbf{X}$ by a set of windows

$$\Psi := \{\mathbf{Q}_i \in \mathbb{R}^{k \times k} | \, i = 1, \ldots, n^2\}, \tag{2}$$

where $\mathbf{Q}_i$ denotes the ith window in Ψ. For any window $\mathbf{Q}_i$, we can reshape it into a vector $\mathbf{p}_i = \mathrm{vec}(\mathbf{Q}_i) \in \mathbb{R}^{k^2}$. For convenience, we define $\mathbf{P} := [\mathbf{p}_1, \ldots, \mathbf{p}_{n^2}] \in \mathbb{R}^{k^2 \times n^2}$ and $\mathbf{w} := \mathrm{vec}(\mathbf{W}) \in \mathbb{R}^{k^2}$. Note that the transformation from $\mathbf{X}$ to $\mathbf{P}$ is often called im2col (see Fig. 2). Thus, Eq. (1) can be written as a matrix–vector multiplication:

$$\mathbf{Y} = \mathrm{vec2mat}(\mathbf{P}^{\mathsf{T}}\mathbf{w}), \tag{3}$$

where the function vec2mat$(\cdot)$ denotes the operation to reshape a vector to a matrix.

Clearly, the complexity of the matrix–vector multiplication is $O(n^2k^2)$, which can be very expensive when n and/or k are very large. However, some of the windows are very smooth and only contain limited information. More critically, performing convolution on smooth windows may also introduce noises into the computation and thus hamper the performance. Thus, it is necessary and important to reduce the computational redundancy on smooth windows to improve the performance of convolution.

4. Proposed method

In this paper, we propose a Content-aware Convolution (CAC) that replaces the original kernel with a 1 × 1 kernel to perform convolution on smooth windows. Moreover, we present an effective method to automatically detect smooth windows. We show the overall scheme in Fig. 2 and the detailed computation method of CAC in Algorithm 1.

4.1. Motivation

The standard convolution convolves the input images/features by transforming them into a set of windows and adopts a sliding window scheme to extract features (Ludwig, 2013). However, not all the windows contribute equally to the prediction results of deep networks. As shown in Fig. 1, there are a large number of smooth windows in the feature maps of each layer. These windows often contain very similar pixels and come with very limited information about the data. As a result, performing convolution on smooth windows can be redundant. More critically, the computational redundancy on these smooth windows may also introduce noises into the computation and thus deteriorate the performance.

Instead of convolving all the windows using the same kernel, we seek to perform different convolutional computations on the windows according to their smoothness. Specifically, we first recognize the sharp and smooth windows. Then, we perform the standard convolution on the sharp windows and perform the convolution with a smaller kernel of 1 × 1 on a single pixel of the smooth windows to reduce the computational redundancy. Since the computation depends on the content of the input data in each layer, we call our method **Content-aware Convolution (CAC)**.

4.2. Content-aware convolution

Given an input feature map or image, we divide the whole window set Ψ into two disjoint subsets, namely the **sharp window set** Λ and the **smooth window set** Φ. We will illustrate how to detect smooth/sharp windows from Ψ in Section 4.3. To reduce the computational redundancy on smooth windows, we seek to

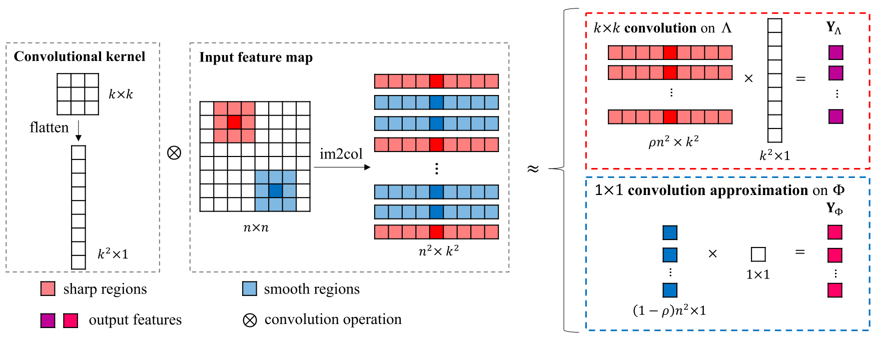

Fig. 2. The computation method of Content-aware Convolution. We first divide the input feature maps into two parts, namely the sharp windows (red boxes) and smooth windows (blue boxes). Then, we perform $k \times k$ convolution on sharp windows and 1×1 convolution on smooth windows. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Algorithm 1: Content-Aware Convolution (CAC).

```
Require: Input feature map $\mathbf{X} \in \mathbb{R}^{n\times n}$;
         Convolutional kernel $\mathbf{W} \in \mathbb{R}^{k\times k}$;
         Learnable parameters kernel $\gamma$ and $\beta$;
         Set of input windows $\Psi := \{\mathbf{Q}_i \in \mathbb{R}^{k\times k} \mid i = 1, ..., n^2\}$.
1: Compute the $1\times1$ kernel $w_\Phi$ using Eqn. (6);
2: Compute the average feature map using Eqn. (9);
3: Compute the gradient of feature maps $\mathbf{G}$ using Eqn. (10);
4: Compute the score map based on $\mathbf{G}$:
       $\mathbf{M} = \text{Sigmoid}\,(\gamma\mathbf{G} + \beta)$;
5: Obtain the set of sharp windows:
       $\Lambda = \{\mathbf{Q}_i \mid M_i > 0.5\}$;
6: Obtain the set of smooth windows:
       $\Phi = \Psi \setminus \Lambda$;
7: Perform convolution on $\Lambda$:
       $\mathbf{Y}_\Lambda = \text{Conv}(\Lambda; \mathbf{W})$;
8: Perform convolution on $\Phi$:
       $\mathbf{Y}_\Phi = \text{Conv}(\Phi; w_\Phi)$;
9: Combine $\mathbf{Y}_\Lambda$ and $\mathbf{Y}_\Phi$ to obtain the final output:
       $\mathbf{Y} = \text{Combine}(\mathbf{Y}_\Lambda, \mathbf{Y}_\Phi)$.
```

use a 1×1 kernel to replace the original large kernel. Given an input window $\mathbf{Q}_i \in \mathbb{R}^{k\times k}$ and a single-channel kernel $\mathbf{W} \in \mathbb{R}^{k\times k}$, the output of a convolution layer y_i can be computed by

$$y_i = \mathbf{Q}_i \otimes \mathbf{W} = \mathbf{p}_i^\top \mathbf{w} = \sum_{j=1}^{k^2} p_j w_j, \tag{4}$$

where $\mathbf{p}_i$ denotes vector presentation of $\mathbf{Q}_i$ and p_j denotes the jth element of $\mathbf{p}_i$. If $\mathbf{Q}_i$ is a smooth window, it implies that all the elements of the window should have very similar values, *i.e.*, for $\forall m, n \in \{1, \ldots, k^2\}$, $p_m \approx p_n$. Therefore, it follows that

$$\mathbf{p}_i^\top \mathbf{w} = \sum_{j=1}^{k^2} p_j w_j \approx \bar{p} \cdot \sum_{j=1}^{k^2} w_j, \tag{5}$$

where $\bar{p}$ can be the average value of all p_j ($\bar{p} = \frac{1}{k^2}\sum_{j=1}^{k^2} p_j$) or any element of this window. In this paper, we choose the center element to compute $\bar{p}$. Relying on Eq. (5), we approximate the original $k \times k$ convolution kernel using a 1×1 kernel:

$$w_\Phi = \sum_{j=1}^{k^2} w_j. \tag{6}$$

Note that the computation on similar pixels may introduce noises into the computation. Our CAC performs convolution on a single pixel in a window and thus effectively reduces the impact of the noisy information. In this sense, the CAC based models are often more robust than the models built with the standard convolution (see results in Table 5). Moreover, the computational cost of CAC on smooth windows can be reduced to $1/k^2$ of the cost with the $k \times k$ kernels.

Given the sharp windows Λ and smooth windows Φ, we obtain the output of CAC by performing a $k \times k$ convolution and a 1×1 convolution on Λ and Φ, respectively. Let $\mathbf{W} \in \mathbb{R}^{k\times k}$ be the parameters of a single-channel convolutional kernel, w_Φ be the 1×1 kernel obtained by Eq. (6). The computation on Λ and Φ can be formulated by

$$\mathbf{Y}_\Lambda = \text{Conv}(\Lambda; \mathbf{W}), \quad \mathbf{Y}_\Phi = \text{Conv}(\Phi; w_\Phi). \tag{7}$$

Then, we combine $\mathbf{Y}_\Lambda$ and $\mathbf{Y}_\Phi$ to obtain the final output according to the relative positions of the windows.

$$\mathbf{Y} = \text{Combine}(\mathbf{Y}_\Lambda, \mathbf{Y}_\Phi). \tag{8}$$

4.3. Sharp and smooth window recognition

Based on the smoothness of windows, we propose an effective method to automatically detect the sharp/smooth windows. In this paper, we measure the data smoothness using the gradients of the input images or features.

For any layer of a deep network, not all the channels are useful and some of them are noisy or irrelevant to the final prediction results (Zhuang et al., 2018). As a result, the features in these channels can be very noisy and thus may hamper the final prediction results (Wang, Ouyang, Wang, & Lu, 2015). Regarding this issue, we seek to compute the average feature map over different channels to alleviate the influence of noisy features:

$$\bar{\mathbf{X}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{X}_i, \tag{9}$$

where m denotes the number of channels. Then, we compute the gradient of the averaged feature map to compare the sharpness of different windows. In this paper, we use the Sobel operator (Kanopoulos, Vasanthavada, & Baker, 1988) to compute the gradient by performing two 1-d convolutions along the x- and y-axis, respectively:

$$\mathbf{G}_x = \bar{\mathbf{X}} \otimes \begin{bmatrix} -1 & 0 & +1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \quad \mathbf{G}_y = \bar{\mathbf{X}} \otimes \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} -1 \\ 0 \\ +1 \end{bmatrix}.$$

Thus, the total gradient becomes

$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}. \tag{10}$$

Given the feature map with m channels, computing gradients on the averaged feature map only yields $1/m$ cost of the computation on all the channels. Compared to the cost of convolution, the cost of computing gradients can be negligible in practice.

Based on the computed gradients, we may divide the windows into sharp and smooth windows according to some threshold. However, such a threshold has to be carefully selected for each layer, making it very time consuming and labor-intensive. To address this issue, we propose a learnable module to automatically discriminate the sharp windows from the smooth windows. Specifically, we exploit an affine function to transform the gradients and then apply the Sigmoid function to compute the probability of a window being sharp. The probability map $\mathbf{M}$ can be computed by

$$\mathbf{M} = \mathrm{Sigmoid}\left(\gamma \mathbf{G} + \beta\right), \tag{11}$$

where γ and β are trainable parameters.

Here, we consider the windows with the probability larger than 0.5 as sharp windows and the other windows as smooth windows. Formally, the sets of sharp and smooth windows can be represented by:

$$\Lambda = \{\mathbf{Q}_i \mid M_i > 0.5\}, \quad \Phi = \Psi \setminus \Lambda, \tag{12}$$

where M_i is the score of the window $\mathbf{Q}_i$ and Ψ denote the set of all the windows. By changing a hard threshold manner to a learnable scheme, the model is able to adjust γ and β to find the optimal number of smooth windows to perform 1×1 convolution. We will show the training method for γ and β in Section 4.4.

4.4. Training method

Note that CAC seeks to find a number of smooth windows to perform 1×1 convolution to improve the performance. Although we can reduce the computational cost, performing 1×1 convolution on too many windows may also hamper the performance (see results in Table 6). To find a good trade-off between model performance and computational cost, we propose to solve a multi-objective optimization problem.

Let M be the CAC-based model to be trained, M^b be the baseline model with the standard convolution, and $c(*)$ be the function to measure the computational cost of deep models, *e.g.*, the number of multiply-adds (MAdds). To train CAC-based models, we use the weighted product method[1] to build the objective:

$$L = \ell(M)\left(\frac{c(M)}{c(M^b)}\right)^{\lambda}, \tag{13}$$

where $\ell(M)$ denotes the standard loss function w.r.t. M (*e.g.*, the cross-entropy loss for classification models) and $\lambda \geq 0$ is a constant weight factor. When $\lambda = 0$, the objective is reduced to the standard loss for a specific task. When $\lambda > 0$, we seek to find a promising trade-off between model performance and computational cost. In this paper, to obtain a good balance, we use λ to control the importance of computational cost (see discussions on λ in Section 7.2).

5. More discussions

In this section, we first analyze the computational complexity of the proposed CAC convolution in Section 5.1. Then, we discuss the differences between the proposed methods and existing methods in Section 5.2.

[1] We use the weighted product method because it is easy to customize for different models. The weighted sum method is also appropriate.

5.1. Computational complexity analysis

To analyze the computational complexity of the proposed CAC, we consider the more general case in which a convolution layer contains multiple channels. Let $\mathcal{X} \in \mathbb{R}^{n\times n\times c_{in}}$ be the input feature maps of a convolution layer and the convolutional kernel be $\mathcal{W} \in \mathbb{R}^{k\times k\times c_{in}\times c_{out}}$, where c_{in} and c_{out} are the number of input and output channels, respectively. Then, the convolution in a standard convolution layer can be computed by

$$\mathcal{O} = \mathcal{X} \otimes \mathcal{W}, \quad \mathcal{O} \in \mathbb{R}^{n\times n\times c_{out}}, \tag{14}$$

where $\otimes$ denotes the convolution operation. The number of multiply-adds (MAdds) required by the standard convolution is given by:

$$\Omega_{\mathrm{Conv}} = c_{in} \cdot c_{out} \cdot k \cdot k \cdot n \cdot n. \tag{15}$$

Given a specific proportion of sharp windows (denoted by ρ), the computational complexity consists of three parts. **First**, there are $\rho \cdot n^2$ sharp windows in Λ where we perform the standard convolution using the $k \times k$ kernel. Thus, the complexity of the first part becomes $\rho\Omega_{\mathrm{conv}}$. **Second**, we perform 1×1 convolution on $(1-\rho)\cdot n^2$ smooth windows in Φ, each of which only requires $1/k^2$ complexity of the standard $k \times k$ convolution. Therefore, the computational complexity of the second part is $(1-\rho)\Omega_{\mathrm{conv}}/k^2$. **Third**, we perform two 1-d convolutions with a single output channel to compute the gradients along the x- and y-axis, respectively. Thus, there are a total of four 1×3 or 3×1 convolutions to compute the gradient.

Compared to the standard convolution with c_{in} input channels and c_{out} output channels, the complexity of computing gradients in the third part is

$$\underbrace{4 \cdot \frac{3}{k^2} \cdot \frac{1}{c_{in}c_{out}}\Omega_{\mathrm{Conv}}}_{\text{computing gradient}} + \underbrace{\frac{1}{k^2 c_{in}c_{out}}\Omega_{\mathrm{conv}}}_{\text{linear transformation}} = \frac{13}{k^2 c_{in}c_{out}}\Omega_{\mathrm{conv}}. \tag{16}$$

As a result, the total computational complexity of the CAC convolution becomes:

$$\Omega_{\mathrm{CAC}} = \left(\underbrace{\rho}_{k\times k\ \mathrm{conv}} + \underbrace{\frac{1-\rho}{k^2}}_{1\times 1\ \mathrm{conv}} + \underbrace{\frac{13}{k^2 c_{in}c_{out}}}_{\text{computing score map}}\right)\Omega_{\mathrm{Conv}}. \tag{17}$$

To accelerate the computation of the convolutions, according to Eq. (17), we have to satisfy the condition such that $\Omega_{\mathrm{CAC}}/\Omega_{\mathrm{Conv}} \leq 1$. In this sense, we can obtain the upper bound of the ratio ρ:

$$\rho \leq \overline{\rho} = 1 - \frac{13}{(k^2-1)\cdot c_{in}\cdot c_{out}}. \tag{18}$$

Specifically, for a 3×3 convolution (*i.e.*, $k = 3$), the upper bound is $\overline{\rho} = 1 - \frac{13}{8\cdot c_{in}\cdot c_{out}}$. Taking ResNet (He et al., 2016) as an example, the number of channels c_{out} ranges from 16 to 512. In this sense, the ratio only needs to be $\rho < 99.3\%$ when we substitute the smallest value $c_{in} = c_{out} = 16$ into Eq. (18). Thus, there is considerable potential to accelerate the computation of the standard convolution.

5.2. Differences from existing methods

The proposed CAC method has several essential differences from existing methods. **First**, the standard convolution performs convolution using a general kernel on all the windows and ignores the inherent redundancy, which may hamper the performance (see results in Tables 1 and 2). In contrast, the proposed CAC performs different convolutions on these windows to reduce the computational redundancy and improve the performance.

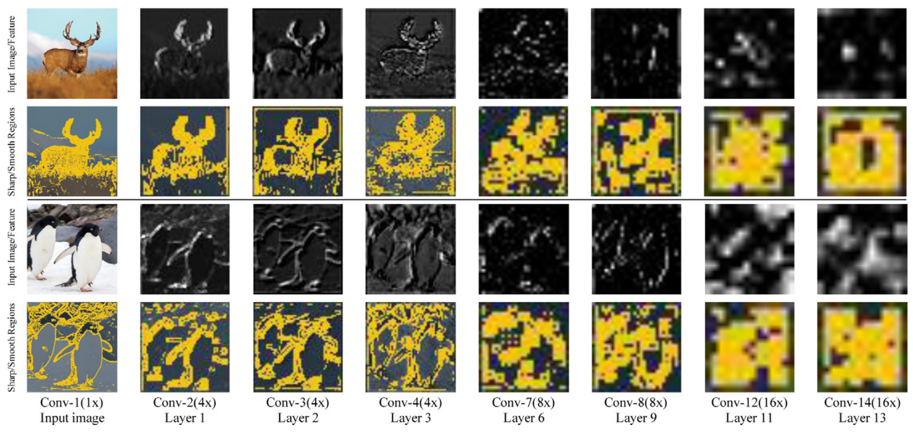

Fig. 3. Visualization of the feature maps of different layers in CAC-ResNet18 on ImageNet. For each image, the top row shows the feature map of different layers and the bottom row shows the corresponding map of sharp windows detected by CAC. In the bottom row, yellow regions denote the sharp windows to perform 3×3 convolution and the dark regions denote the smooth ones to perform 1×1 convolution. We scale the feature maps of different layers to the same spatial size for better visualization.

Table 1
Comparisons of different convolutions in terms of both computational complexity and testing error based on various architectures on CIFAR-10. "\" denotes the missing results of the models to which OctConv cannot be applied.

Conv type	ResNet20		ResNet32		ResNet56		DenseNet121		ShuffleNetV2	
	#MAdds (M)	Error (%)	#MAdds (M)	Error (%)	#MAdds (M)	Error (%)	#MAdds (M)	Error (%)	#MAdds (M)	Error (%)
Standard Conv	40.93	8.75	69.12	7.51	126.08	6.97	888.51	4.78	45.04	7.25
CAC	**25.79**	**8.37**	**46.58**	**7.25**	**87.27**	**6.51**	**733.82**	**4.60**	**41.33**	**7.13**
OctConv (Chen et al., 2019)	26.33	8.77	44.02	8.07	61.72	7.58	403.80	5.52	\	\
CAC-OctConv	**19.49**	**8.60**	**34.39**	**7.63**	**52.12**	**6.99**	**356.63**	**4.68**	\	\

Second, existing methods perform the same computation on the samples with different spatial redundancy. Unlike these methods, our CAC adopts a content-aware computation scheme that dynamically allocates suitable computational resources for different samples according to their data smoothness. It is worth noting that our CAC is more robust to the samples with adversarial perturbations than existing convolution methods. The main reason is that CAC replaces the large kernel convolution with a 1×1 convolution on smooth windows, which effectively reduces the influence incurred by the noises/attacks in these windows.

6. Experiments

In this section, we use CAC to accelerate two popular convolution methods, namely the standard convolution and the octave convolution (OctConv) (Chen et al., 2019). We apply CAC to various architectures and demonstrate the performance on three computer vision tasks, including image classification, semantic segmentation, and object detection. All implementations are based on PyTorch.[2]

We organize the experiments as follows. First, we show the visual interpretation of each CAC layer in Section 6.1. Second, we evaluate our CAC on image classification tasks in Section 6.2. Third, we apply our CAC method to semantic segmentation models and evaluate the proposed method in Section 6.3. Fourth, we conduct experiments to show the effectiveness of our CAC method on object detection tasks in Section 6.4. Finally, we investigate the effect of the hyperparameter λ in Section 7.2.

6.1. Visual interpretation of CAC convolution

To better understand the proposed CAC method, we visualize the feature maps and the corresponding masks of the sharp windows of different CAC layers inside deep networks. In this experiment, we take the CAC-ResNet18 model as an example and show the results in Fig. 3.

From Fig. 3, the sharp windows (marked in yellow) are often located at the edges and contain the main information about the object. However, the smooth windows (marked in black) are often very smooth areas that only contain little information. As discussed in Section 4, when the input windows are very smooth, it is not necessary to use a large kernel to perform convolution. Thus, we can reduce the computational cost of convolution on the smooth windows using a 1×1 kernel to approximate the original output. In this way, our CAC method can greatly reduce the computational complexity without a loss of information. More critically, since the input images or features may have different numbers of smooth windows, the resultant CAC models can dynamically allocate suitable computation power to different input images. Thus, our CAC models can perform content-aware computation to improve the performance of convolution.

6.2. Experiments on image classification

In this experiment, we consider two popular convolution methods as the baseline methods, namely, the standard convolution and the OctConv (Chen et al., 2019). We apply the proposed CAC method to various image classification models, including ResNet (He et al., 2016), DenseNet (Huang, Liu, van der Maaten, & Weinberger, 2017), and ShuffleNetV2 (Ma, Zhang, Zheng, & Sun, 2018).

[2] The implementation of the proposed CAC method is available at https://github.com/guoyongcs/CAC.

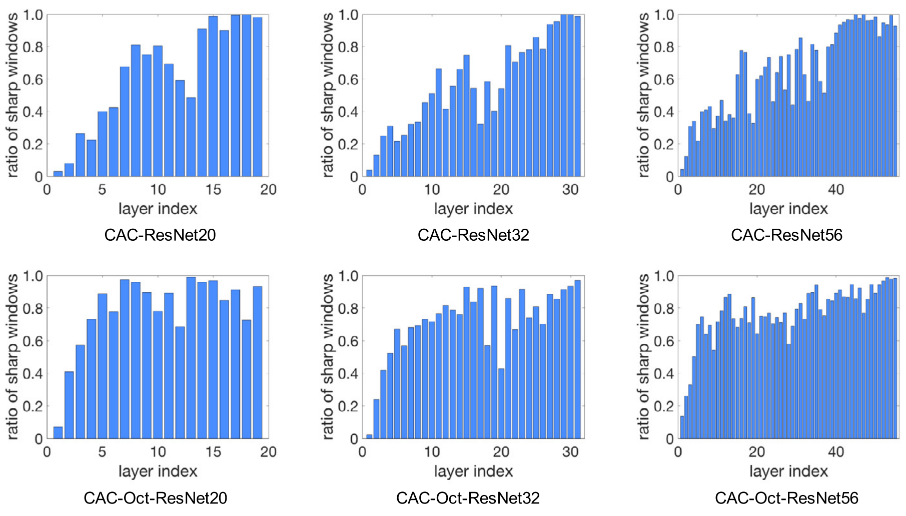

Fig. 4. Visualization of the ratios of sharp windows for different layers in ResNet20, ResNet32, and ResNet56 on CIFAR-10. We compute the ratios by averaging the ratios over 10,000 testing samples.

Table 2
Comparisons of different convolutions in terms of both computational complexity and validation error on ImageNet. "\" denotes the missing results of the models to which OctConv cannot be applied.

Conv type	ResNet18			ResNet50			DenseNet121			ShuffleNetV2		
	#MAdds	Error (%)		#MAdds	Error (%)		#MAdds	Error (%)		#MAdds	Error (%)	
	(G)	Top-1	Top-5	(G)	Top-1	Top-5	(G)	Top-1	Top-5	(G)	Top-1	Top-5
Conv	1.81	30.36	11.02	4.09	24.01	7.07	2.83	25.35	7.83	0.15	30.64	11.68
CAC	**1.41**	**30.19**	**10.87**	**3.75**	**23.79**	**6.81**	**2.52**	**24.49**	**7.37**	**0.13**	**30.13**	**11.27**
OctConv (Chen et al., 2019)	1.14	29.64	10.48	2.37	23.27	6.55	1.37	25.68	7.90			
CAC-OctConv	**0.96**	**29.43**	**10.27**	**2.19**	**23.05**	**6.37**	**1.28**	**24.82**	**7.61**			

6.2.1. Datasets and implementation details

We conduct experiments on two benchmark image classification datasets, including CIFAR-10 (Krizhevsky & Hinton, 2009), and ImageNet (Deng et al., 2009). CIFAR-10 consists of 50k training samples and 10k testing images with 10 classes. ImageNet contains 123k training samples and 50k testing images for 1000 classes.

We follow the settings in He et al. (2016) and use SGD with nesterov (Nesterov, 1983) for the optimization. The momentum and weight decay are set to 0.9 and 0.0001, respectively. On CIFAR-10, we train the models for 400 epochs using a mini-batch size of 128. The learning rate is initialized to 0.1 and divided by 10 at epochs 160 and 240, respectively. On ImageNet, we train the models for 90 epochs with a mini-batch size of 256. The learning rate is started at 0.1 and divided by 10 at epochs 30 and 60, respectively. We train the CAC based models with $\lambda = 0.3$. We use the number of multiply-adds (MAdds) to measure the computational complexity of deep models. Based on sharp window set Λ and the set of all the windows Ψ, we compute the ratio of sharp windows inside a convolution layer by $\rho = |\Lambda|/|\Psi|$, where $|\cdot|$ denotes the cardinality of a set. In general, a lower ratio implies that the more windows would be convolved with a 1×1 kernel.

6.2.2. Comparisons on CIFAR-10

In this experiment, we evaluate our CAC method on a small dataset CIFAR-10. From Table 1, the proposed CAC method greatly accelerates ResNet and DenseNet models equipped with different convolution types. Specifically, for both the models with the standard convolution and OctConv, our CAC consistently yields significantly better performance and lower computational cost. Moreover, we also show the ratios of sharp windows ρ of each layer based on several models in Fig. 4. From this figure, deep layers tend to have larger ratios than shallow layers due to their better representation ability and smaller feature map, yielding a smaller risk of containing smooth windows. These results show that the proposed CAC removes the redundancy caused by the smooth windows in each layer.

We also consider very compact models, *e.g.*, ShuffleNetV2, which mainly consists of 1×1 group convolution. However, since OctConv requires information exchange among groups, it would destroy the computation of group convolution and thus cannot be directly applied to ShuffleNetV2. Thus, we only compare the performance of the models equipped with the standard convolution and the proposed CAC. Even with such a compact model, our CAC further improves the validation accuracy and reduces the redundancy in the model. These results demonstrate the effectiveness of our method.

6.2.3. Comparisons on ImageNet

We also evaluate our method on a large-scale dataset ImageNet. Similar to the experiments on CIFAR-10, we apply our CAC to improve both the standard convolution and OctConv. In this experiment, we consider ResNet, DenseNet and ShuffleNetV2 as the baseline model. The results are shown in Table 2.

From Table 2, our CAC based models significantly outperform the baseline models with different architectures in terms of Top-1 and Top-5 error. More critically, the resultant models often have lower computational cost and thus become more compact. These results demonstrate the superiority of the proposed CAC method over the existing methods. We also show the ratios ρ of each layer for ImageNet models in Fig. 5. From this figure, due to the training difficulty on a large-scale dataset, ImageNet models are often hard to compress without performance degradation (Liu

Table 3
Comparisons of different models on each class for semantic segmentation. We adopt the FCN-ResNet18 model as the baseline model. "–" denotes the results that are not reported.

Method	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mIoU	#MAdds (G)
BONN-SVR (Carreira, Li and Sminchisescu, 2012)	54.3	23.9	39.5	35.3	42.6	65.4	53.5	46.1	15.0	47.4	30.1	33.9	48.8	54.4	46.4	28.8	51.3	26.2	44.9	37.2	43.3	
O_2P (Carreira, Caseiro, Batista and Sminchisescu, 2012)	64.0	27.3	54.1	39.2	48.7	56.6	57.7	52.5	14.2	54.8	29.6	42.2	58.0	54.8	50.2	36.6	58.6	31.6	48.4	38.6	47.8	–
SDS (Hariharan, Arbeláez, Girshick, & Malik, 2014)	63.3	25.7	63.0	39.8	59.2	70.9	61.4	54.9	16.8	45.0	48.2	50.5	51.0	57.7	63.3	31.8	58.7	31.2	55.7	48.5	51.6	–
MSRA-CFM (Dai, He, & Sun, 2015)	75.7	26.7	69.5	48.8	65.6	81.0	69.2	73.3	30.0	68.7	51.5	69.1	68.1	71.7	67.5	50.4	66.5	44.4	58.9	53.5	61.8	–
FCN (Long, Shelhamer, & Darrell, 2015)	83.5	29.7	68.7	59.8	50.2	80.3	71.9	**71.8**	28.6	59.6	44.4	**61.2**	59.0	66.7	**80.3**	40.8	**63.4**	42.5	74.4	66.9	61.7	45.0
RC-FCN (Li, Liu et al., 2017)	82.8	28.9	64.4	58.6	50.3	80.5	70.5	71.6	26.7	58.3	45.5	58.4	58.1	67.8	79.6	40.9	58.6	42.7	73.9	63.6	60.6	38.2
CAC-FCN	**84.1**	**29.9**	**69.1**	**60.5**	**51.9**	**82.2**	**73.1**	71.5	**28.9**	**59.7**	**48.0**	59.5	**60.2**	**69.2**	80.2	**41.8**	63.1	**44.1**	**76.9**	**67.0**	**62.5**	**37.2**

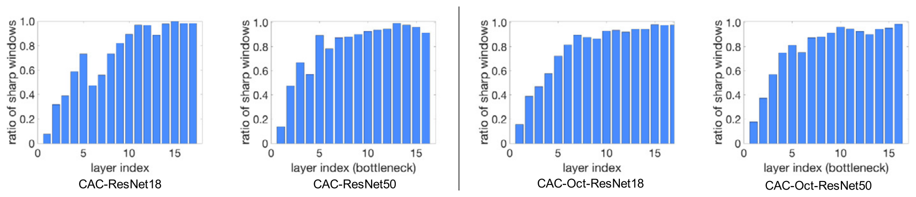

Fig. 5. Visualization of the ratios of sharp windows for different layers in ResNet18 and ResNet50 on ImageNet. The layer index for ResNet50 denotes the index of the bottleneck block. We compute the ratios for different layers by averaging the samples in the validation set.

et al., 2017; Luo et al., 2017; Zhuang et al., 2018), yielding larger ratios of the intermediate layers than the CIFAR-10 models.

6.3. Experiments on semantic segmentation

We further apply the proposed CAC to semantic segmentation models, *e.g.*, fully convolutional network (FCN) (Long et al., 2015). We compare the performance of the models with and without CAC based on a benchmark dataset PSACAL VOC 2012 (Everingham et al., 2015).

6.3.1. Compared methods

We adopt FCN as the baseline model and apply our CAC to show the effectiveness of CAC. In this experiment, we compare our CAC with a strong baseline region convolution (RC). Moreover, we also consider several semantic segmentation methods as the baselines, including BONN-SVR (Carreira, Li et al., 2012), O_2P (Carreira, Caseiro et al., 2012), SDS (Hariharan et al., 2014), and MSRA-CFM (Dai et al., 2015).

6.3.2. Datasets and implementation details

We conduct experiments on the benchmark semantic segmentation dataset PASCAL VOC 2012, which consists of 1464 training images and 1449 validation images. We measure the performance using the commonly used metric, *i.e.*, the mean intersection over union (mIoU), which computes the percent between the intersection and union of the ground truth segmentation mask and the prediction mask.

In this experiment, we use the ImageNet pretrained model as the backbone model, *e.g.*, ResNet18. Following the setting in Chen, Papandreou, Schroff, and Adam (2017), we make some modifications to adapt the model to the semantic segmentation task. First, we remove the last two subsampling layers in ResNet18 to upscale the size of the output feature map by 4×. Second, we replace the last four convolution layers with dilated convolutions. Third, we replace the linear layer with an interpolated upsampling layer. In the training, we first train the model on MS-COCO dataset and then finetune it on PASCAL VOC 2012. We make some modifications to adapt it to the semantic segmentation task. We apply data augmentation by randomly scaling the input images (from 0.5 to 2.0) and randomly left–right flipping in training. The input images are resized to 480×480 in testing. We finetune 30 epochs using mini-batch SGD with a weight decay of 0.0001 and a momentum of 0.9. The learning rate is started at 0.01. In all experiments, we train the CAC based models with $\lambda = 0.3$ and $C = 0.5$.

6.3.3. Performance comparison

In this section, we compare the proposed CAC method with the standard convolution and region convolution (RC) on semantic segmentation tasks. For convenience, we use FCN, RC-FCN, and CAC-FCN to represent the models with the standard convolution, RC, and CAC. We show the results in Table 3 and Fig. 6.

From Table 3, our CAC-FCN outperforms the baseline models with the standard convolution and RC on most of the categories. For the average performance in terms of mIoU, the CAC-FCN model yields significant performance improvement. We also provide a visual comparison of different models in Fig. 6. From this figure, our CAC-FCN produces more accurate segmentation masks than the FCN and RC-FCN baselines.

6.4. Experiments on object detection

In this section, we apply the proposed CAC method to object detection models. We evaluate the CAC-based models on the benchmark dataset MS COCO (Lin et al., 2014).

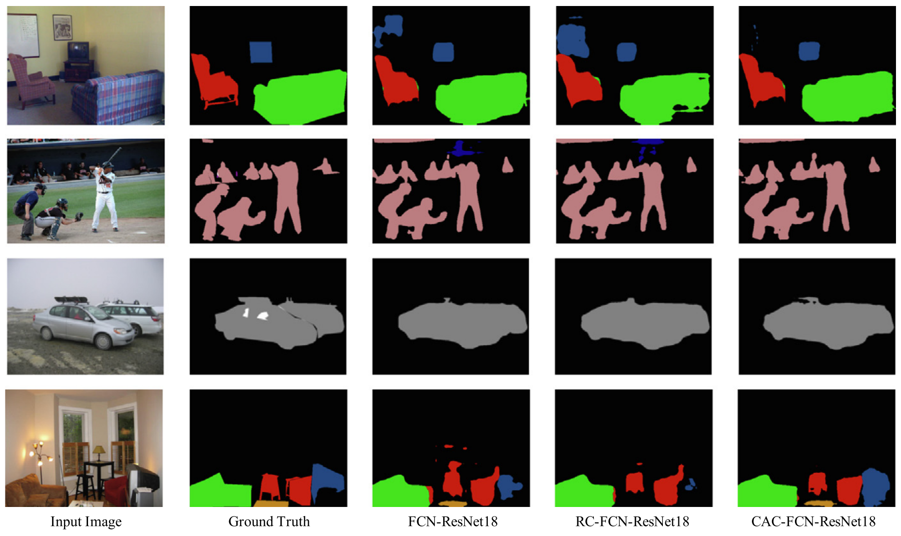

Fig. 6. Visual comparison of the segmentation masks produced by different methods. The first and the second columns show the input images and the corresponding ground truth segmentation masks, respectively. The last three columns show the segmentation masks predicted by different models.

6.4.1. *Compared methods*

In this experiment, we adopt the widely used model Faster-RCNN network as the baseline model. We compare our CAC based model with several state-of-the-art object detection models, including Fast-RCNN (Girshick, 2015), ION (Bell, Lawrence Zitnick, Bala, & Girshick, 2016), YOLOv2 (Redmon & Farhadi, 2017), SSD300 (Liu et al., 2016) and SSD512 (Liu et al., 2016).

6.4.2. *Datasets and implementation details*

We conduct experiments on the MS COCO dataset which contains 117k training images and 5k validation images with 80 classes. We use the ImageNet pretrained ResNet18-based FPN network for comparing different methods on MS COCO dataset. We follow the setting in Ren, He, Girshick, and Sun (2015). The networks are optimized for 13 epochs using SGD with a weight decay of 0.0001 and a momentum of 0.9. The learning rate is initialized with 0.02 and divided by 10 at 8 and 11 epochs.

We evaluate different object detection models using the COCO's standard metric, namely mAP@0.5 and mAP@0.75. These two metrics represent the mean average precision (mAP) computed at the IoU thresholds of 0.5 and 0.75, respectively. We also compute mmAP for different models by averaging multiple mAP values with the IoU thresholds ranging from 0.5 to 0.95 with the step of 0.05. In all experiments, we train the CAC based models with $\lambda = 0.3$ and $C = 0.5$.

6.4.3. *Performance comparison*

In this experiment, we use CAC to replace the standard convolution layers in the Faster-RCNN model. We show the quantitative and visual results in Table 4 and Fig. 7.

From Table 4, CAC can obtain very compact models with better performance and lower computational cost than the baseline models equipped with the standard convolution. Specifically, the resultant CAC-Faster-RCNN has a lower computational cost and consistently outperforms the baseline model on all the considered metrics, including mmAP, mAP@0.5, and mAP@0.75. We also show the visual comparison of different object detection models in Fig. 7. From this figure, our CAC model generates more accurate bounding boxes than the Faster-RCNN baseline. The results demonstrate the effectiveness of the proposed CAC method on object detection tasks.

Table 4
Comparisons of different object detection models on MS COCO dataset. We use the ResNet18 model as the backbone. "–" denotes results that are not reported.

Model	mmAP	mAP@0.5	mAP@0.75	#MAdds (G)
Fast-RCNN (Girshick, 2015)	18.9	38.6	–	–
ION (Bell et al., 2016)	23.6	43.2	23.6	–
YOLOv2 (Redmon & Farhadi, 2017)	21.6	44.0	27.8	–
SSD300 (Liu et al., 2016)	23.2	41.2	23.4	–
SSD512 (Liu et al., 2016)	26.8	46.5	27.8	–
Faster-RCNN (Ren et al., 2015)	28.6	49.2	29.6	23.82
CAC-Faster-RCNN	**29.9**	**49.9**	**31.1**	**20.49**

7. Further experiments

In this section, we first compare the accuracy of our CAC method with the standard convolution method under adversarial perturbations to investigate the robustness of our method. Then, we conduct more experiments to investigate the effect of the hyperparameter λ.

7.1. *Comparisons of Robustness*

We investigate the robustness of the proposed CAC by comparing the accuracy on adversarial samples generated by four different attack methods, including FGSM (Goodfellow et al., 2015), MI-FGSM (Dong et al., 2018), PGD10 (Madry et al., 2018) and PGD100 (Madry et al., 2018). We train the standard convolution models (namely ResNet20 and ResNet56) and the CAC based models with adversarial samples on CIFAR-10. We report the adversarial accuracy that is evaluated on adversarial samples. The higher adversarial accuracy the model has, the more adversarially robust the model will be.

Following the setting in Madry et al. (2018), we train all the models for 200 epochs and use an SGD optimizer with a momentum of 0.9 and a weight decay of 0.0005. The initialized learning rate is set to 0.1 and divided by 10 at epochs 90, 140

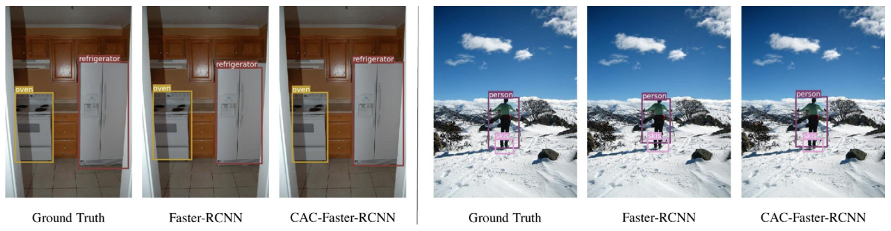

Fig. 7. Visual comparison of different object detection methods. The first column shows the input images with the corresponding ground truth bounding boxes. The second and third columns show the bounding boxes predicted by different models.

Table 5
Comparisons of robustness of deep models with different convolutions on CIFAR-10.

Model	Method	Error on adversarial examples (%)			
		FGSM (Goodfellow, Shlens, & Szegedy, 2015)	MI-FGSM (Dong et al., 2018)	PGD-10 (Madry, Makelov, Schmidt, Tsipras, & Vladu, 2018)	PGD-100 (Madry et al., 2018)
ResNet20	Standard Conv	29.82	54.89	57.99	65.41
	CAC	**28.97**	**53.67**	**57.13**	**64.74**
ResNet56	Standard Conv	26.91	51.91	55.33	64.81
	CAC	**25.58**	**51.03**	**54.72**	**63.75**

Table 6
Comparisons of the CAC-ResNet20 models with different values of λ on CIFAR-10.

Method	Standard Conv	CAC			
λ	–	0.3	0.5	0.7	1.0
Error (%)	8.75	**8.37**	9.92	10.07	11.81
#MAdds ↓ (%)	0	36.98	40.38	51.68	69.39

and 160, respectively. All attacks are with a total perturbation scale of 8/255 (0.03) and a step size of 2/255 (0.01). We set the number of attack iterations to 10, 10 and 100 for MIFGSM (Dong et al., 2018), PGD10 (Madry et al., 2018) and PGD100 (Madry et al., 2018), respectively. From Table 5, our CAC based models achieve higher adversarial accuracy than the standard convolution ones under four different attack perturbations. These results demonstrate that the proposed CAC convolution is more robust than the standard convolution. The main reason is that our CAC replaces the original large kernel with a 1×1 kernel to perform convolution on smooth windows. In this sense, we are able to effectively reduce the influence incurred by the noises and attacks in these windows.

7.2. Effect of λ on CAC

We investigate the effect of the hyperparameter λ in Eq. (13) on the performance of CAC models. In this experiment, we use ResNet20 as the baseline model and train the models with different values of $\lambda \in \{0.3, 0.5, 0.7, 1.0\}$ on CIFAR-10. We show the results in Table 6.

From Table 6, the reduction of MAdds would increase when we gradually increase λ. However, due to the redundancy incurred by the smooth windows, it is possible to simultaneously reduce the computational cost and improve the performance, *e.g.*, when setting $\lambda = 0.3$. If we further increase the value of λ, the objective in Eq. (13) encourages the model to focus more on the computational cost but compromise the performance. For example, when λ is set to 1.0, CAC reduces the computational cost by 69.39% but incur significant performance degradation. Thus, we set $\lambda = 0.3$ to train CAC models in the experiments.

8. Conclusion

In this paper, we have proposed a Content-aware Convolution (CAC) to reduce the computational redundancy incurred by the smooth windows when performing convolution. To reduce the computational redundancy and improve the performance, CAC replaces the original $k \times k$ kernel with a 1×1 kernel to perform convolutions on the smooth windows. Moreover, we propose an efficient algorithm to automatically recognize the sharp and smooth windows. Given different samples, the resultant CAC models could allocate different computation resources according to their data smoothness, which makes it possible for content-aware computation. Extensive results on image classification, semantic segmentation, and object detection tasks show that our CAC based models yield significantly better performance and lower computational cost than the baseline models with the standard convolution.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by Ministry of Science and Technology Foundation Project (2020AAA0106900), National Natural Science Foundation of China (62072190, 62072186), Key Realm R&D Program of Guangzhou (202007030007), Fundamental Research Funds for the Central Universities (D2191240), Program for Guangdong Introducing Innovative and Entrepreneurial Teams (2017ZT07X183), Key-Area Research and Development Program of Guangdong Province (2019B010155002), Guangdong Basic and Applied Basic Research Foundation (2019B1515130001), Tencent AI Lab Rhino-Bird Focused Research Program (JR201902), Guangzhou Science and Technology Planning Project (201904010197), Opening Project of Guangdong Key Laboratory of Big Data Analysis and Processing.

References

Alvarez, J. M., & Salzmann, M. (2016). Learning the number of neurons in deep networks. In *Advances in neural information processing systems*(pp. 2270–2278).

Bar-Hillel, A., & Weinshall, D. (2008). Efficient learning of relational object class models. *International Journal of Computer Vision*, *77*(1–3), 175–198.

Bell, S., Lawrence Zitnick, C., Bala, K., & Girshick, R. (2016). Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2874–2883).

Burrus, C. S., & Parks, T. (1985). *Convolution algorithms*. Citeseer.

Carreira, J., Caseiro, R., Batista, J., & Sminchisescu, C. (2012). Semantic segmentation with second-order pooling. In *The European conference on computer vision* (pp. 430–443).

Carreira, J., Li, F., & Sminchisescu, C. (2012). Object recognition by sequential figure-ground ranking. *International Journal of Computer Vision*, *98*(3), 243–262.

Chen, Y., Fan, H., Xu, B., Yan, Z., Kalantidis, Y., Rohrbach, M., et al. (2019). Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. In *The IEEE international conference on computer vision* (pp. 3434–3443).

Chen, L., Papandreou, G., Schroff, F., & Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation. Arxiv abs/1706.05587.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *The IEEE conference on computer vision and pattern recognition*(pp. 1800–1807).

Dai, J., He, K., & Sun, J. (2015). Convolutional feature masking for joint object and stuff segmentation. In *The IEEE conference on computer vision and pattern recognition* (pp. 3992–4000).

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *The IEEE conference on computer vision and pattern recognition*.

Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., et al. (2018). Boosting adversarial attacks with momentum. In *The IEEE conference on computer vision and pattern recognition* (pp. 9185–9193).

Duch, W., Matykiewicz, P., & Pestian, J. (2008). Neurolinguistic approach to natural language processing with applications to medical text analysis. *Neural Networks*, *21*(10), 1500–1510.

Everingham, M., Eslami, S. M. A., Gool, L. V., Williams, C. K. I., Winn, J. M., & Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, *111*(1), 98–136.

Fergus, R., Perona, P., & Zisserman, A. (2003). Object class recognition by unsupervised scale-invariant learning. In *2003 IEEE computer society conference on computer vision and pattern recognition, 2003. Proceedings, Vol. 2* (p. II). IEEE.

Girshick, R. (2015). Fast R-CNN. In *The IEEE international conference on computer vision* (pp. 1440–1448).

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples. In Y. Bengio, & Y. LeCun (Eds.), *International conference on learning representations*.

Graham, B., Engelcke, M., & van der Maaten, L. (2018). 3D semantic segmentation with submanifold sparse convolutional networks. In *The IEEE conference on computer vision and pattern recognition* (pp. 9224–9232).

Gross, A., & Murthy, D. (2014). Modeling virtual organizations with Latent Dirichlet Allocation: A case for natural language processing. *Neural Networks*, *58*, 38–49.

Guo, Y., Chen, J., Du, Q., Van Den Hengel, A., Shi, Q., & Tan, M. (2020). Multi-way backpropagation for training compact deep neural networks. *Neural Networks*.

Guo, Y., Zheng, Y., Tan, M., Chen, Q., Chen, J., Zhao, P., et al. (2019). Nat: Neural architecture transformer for accurate and compact architectures. In *Advances in neural information processing systems* (pp. 737–748).

Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *International conference on learning representations*.

Hariharan, B., Arbeláez, P., Girshick, R., & Malik, J. (2014). Simultaneous detection and segmentation. In *The European conference on computer vision*(pp. 297–312).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *The IEEE conference on computer vision and pattern recognition* (pp. 770–778).

He, Y., Zhang, X., & Sun, J. (2017). Channel pruning for accelerating very deep neural networks. In *The IEEE international conference on computer vision*(pp. 1398–1406).

Hu, H., Peng, R., Tai, Y.-W., & Tang, C.-K. (2016). Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. ArXiv abs/1607.03250.

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *The IEEE conference on computer vision and pattern recognition* (pp. 2261–2269).

Ibtehaz, N., & Rahman, M. S. (2020). MultiResUNet: Rethinking the U-net architecture for multimodal biomedical image segmentation. *Neural Networks*, *121*, 74–87.

Kanopoulos, N., Vasanthavada, N., & Baker, R. L. (1988). Design of an image edge detection filter using the sobel operator. *IEEE Journal of Solid-State Circuits*, *23*(2), 358–367.

Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images*: *Tech Report*.

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1106–1114).

Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, H. P. (2017). Pruning filters for efficient convnets. In *International conference on learning representations*.

Li, X., Liu, Z., Luo, P., Change Loy, C., & Tang, X. (2017). Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *The IEEE conference on computer vision and pattern recognition* (pp. 6459–6468).

Li, F., Zhang, B., & Liu, B. (2016). Ternary weight networks. ArXiv abs/1605.04711.

Lin, T., Maire, M., Belongie, S. J., Hays, J., Perona, P., Ramanan, D., et al. (2014). Microsoft COCO: Common objects in context. In *The European conference on computer vision* (pp. 740–755).

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., et al. (2016). Ssd: Single shot multibox detector. In *The European conference on computer vision* (pp. 21–37). Springer.

Liu, L., Cao, J., Liu, M., Guo, Y., Chen, Q., & Tan, M. (2020). Dynamic extension nets for few-shot semantic segmentation. In *Proceedings of the 28th ACM international conference on multimedia*.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., & Zhang, C. (2017). Learning efficient convolutional networks through network slimming. In *The IEEE international conference on computer vision* (pp. 2755–2763).

Liu, B., Wang, M., Foroosh, H., Tappen, M., & Pensky, M. (2015). Sparse convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition* (pp. 806–814).

Liu, J., Zhuang, B., Zhuang, Z., Guo, Y., Huang, J., Zhu, J., et al. (2020). Discrimination-aware network pruning for deep model compression. arxiv preprint arXiv:2001.01050.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *The IEEE conference on computer vision and pattern recognition* (pp. 3431–3440).

Ludwig, J. (2013). *Image convolution*. Portland State University.

Luo, J.-H., Wu, J., & Lin, W. (2017). ThiNet: A filter level pruning method for deep neural network compression. In *The IEEE international conference on computer vision* (pp. 5068–5076).

Ma, N., Zhang, X., Zheng, H.-T., & Sun, J. (2018). ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In *The European conference on computer vision* (pp. 122–138).

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. In *International conference on learning representations*.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate O (1/k2). In *Soviet math. dokl, Vol. 27* (pp. 372–376).

Ozawa, S., Toh, S. L., Abe, S., Pang, S., & Kasabov, N. (2005). Incremental learning of feature space and classifier for face recognition. *Neural Networks*, *18*(5–6), 575–584.

Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In *The IEEE conference on computer vision and pattern recognition* (pp. 7263–7271).

Ren, S., He, K., Girshick, R. B., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (pp. 91–99).

Ren, S., He, K., Girshick, R. B., & Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149.

Schrauwen, B., D'Haene, M., Verstraeten, D., & Van Campenhout, J. (2008). Compact hardware liquid state machines on FPGA for real-time speech recognition. *Neural Networks*, *21*(2–3), 511–523.

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *The IEEE conference on computer vision and pattern recognition* (pp. 815–823).

Shelhamer, E., Long, J., & Darrell, T. (2017). Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(4), 640–651.

Sifre, L., & Mallat, S. (2014). *Rigid-motion scattering for image classification* (Ph.D. thesis), Citeseer.

Skowronski, M. D., & Harris, J. G. (2007). Automatic speech recognition using a predictive echo state network classifier. *Neural Networks*, *20*(3), 414–423.

Sun, Y., Wang, X., & Tang, X. (2015). Deeply learned face representations are sparse, selective, and robust. In *The IEEE conference on computer vision and pattern recognition* (pp. 2892–2900).

Vasudevan, A., Anderson, A., & Gregg, D. (2017). Parallel multi channel convolution using general matrix multiplication. In *2017 IEEE 28th international conference on application-specific systems, architectures and processors (ASAP)* (pp. 19–24). IEEE.

Wang, H., Dai, L., Cai, Y., Sun, X., & Chen, L. (2018). Salient object detection based on multi-scale contrast. *Neural Networks*, *101*, 47–56.

Wang, L., Ouyang, W., Wang, X., & Lu, H. (2015). Visual tracking with fully convolutional networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 3119–3127).

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., & Zou, Y. (2016). Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. Arxic abs/1606.06160.

Zhu, C., Han, S., Mao, H., & Dally, W. J. (2017). Trained ternary quantization. In *International conference on learning representations*.

Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., et al. (2018). Discrimination-aware channel pruning for deep neural networks. In *Advances in neural information processing systems* (pp. 883–894).