Efficient Convolutions using Box-Filters for Low-Latency Image Processing

Susim Mukul Roy
University at Buffalo
susimmuk@buffalo.edu

Bharat Yalavarthi
University at Buffalo
byalavar@buffalo.edu

Nalini Ratha
University at Buffalo
nratha@buffalo.edu

***Abstract*—The convolution operator in Convolutional Neural Networks (CNNs) has been a crucial aspect of success in various computer vision tasks. There has been a recent surge in the use of CNNs for generative modeling. However, such techniques usually have a long inference time which is mainly due to the costly convolution operation. Thus, we bring our attention to reducing the total number of mathematical operations during convolution using the Summed Area Table (SAT) and box filters. In this work, we find the redundant box filters that are being repeated across the filter dimension and prune them to save computation. Additionally, we also show that this does not take a toll on the accuracy and instead increases it after finetuning the remaining architecture. Thus, our proposed method computes convolution output with lower latency as compared to using all the box filters, increases classification accuracy, and can be applied with arbitrarily shaped kernels.**

***Index Terms*—Summed Area Table, convolution, box filters**

I. Introduction

Convolutional Neural Networks(CNN) have been long-standing the to-go method for various tasks including image classification, object recognition, image segmentation, and image captioning. Whilst these problems have been solved in terms of their accuracies or precision, an open-ended question remains in decreasing the latency of these models during inference. Several works have been proposed to decrease the training time [5]–[7] through techniques like filter pruning, weight quantization, and sparse representations which have pushed the boundaries of faster training methods. However, accelerating the convolution operation during inference has been a less explored area. Given the widespread application of CNNs, especially as foundational components in popular generative models, it is essential to develop methods that mitigate their computational overhead. Reducing this overhead allows for greater emphasis on downstream tasks, minimizing the need for extensive optimization of inference times in CNN-based architectures. As shown by [18], the convolution operation takes up the majority of the running time which further focuses our attention on the convolution operator. Therefore, to solve this challenge, we come up with an alternative way to do convolution in a much faster way during inference whilst maintaining accuracy.

II. Related Work

Various methods have been proposed to accelerate CNN inference. These methods either exploit redundancies of CNN models to reduce the number of parameters and computations or introduce lightweight model structures for a given task. An important direction of significance is Low-Rank Decomposition which decomposes the original weight matrices to several low rank matrics [14] or low dimensional tensors [15]. These tensor decomposition methods rely on rank selection, which is an ill-posed problem, while the matrix factorization methods have limited speedup since redundancies in the standalone weight values are not considered. Alternatively, pruning methods like [16] propose pruning weights by comparing the magnitude with a threshold while [17] prune the redundant weight coefficients based on the magnitude after decomposing the filters into basis functions and finetuning them. However, these approaches depend on the granularity of the pruning technique and most of them still require additional finetuning. Summed Area Tables(SATs) are well-established concepts in computer vision, enabling the rapid computation of the sum of values within any arbitrary subset of a grid, maintaining a constant time complexity [8]. SATs are used to speed up computations for various tasks including texture mapping [9], decomposition of networks [10], and accelerating convolutions with binary and large kernels [11], [12]. Our approach is based on the box filtering idea that has long been mainstream in computer vision. Through the 2000s, a large number of architectures that use box filtering to integrate spatial context information have been proposed. The landmark work that started the trend was the Viola-Jones face detection system [2]. These methods capitalized on the ability of box filtering to be performed very efficiently through the use of the integral image trick [3]. Another method, introduced by [1], box filters have been used for approximating image filters via Exhaustive Search for small filters which forms a closed-form solution. However, one still needs to consider all the box filters in this case and the convolution is then done in the usual sliding-window fashion.

III. Methodology

Our proposed method has two parts as shown in Algorithm 1. First, we calculate all the box filters for all the filters for all the layers in line 3. Next, while inference, we iterate over the filters for each layer and check whether we have already encountered a particular box filter or it is unique. Accordingly, we calculate the sum of the rectangular region if that region is new, otherwise, we simply access it and store it in our

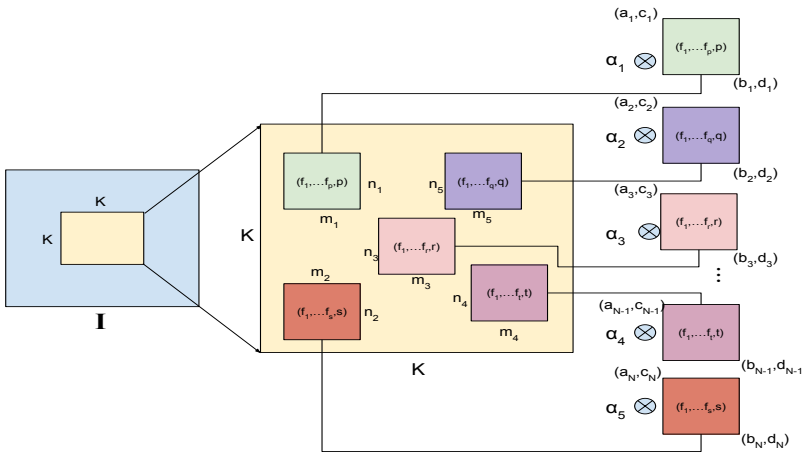

Fig. 1. Overview of our methodology. Each filter of size $K \times K$ is made up of N number of box filters with different α coefficients. Each box filter is represented by $(f_1, \ldots, f_k, k)$ which represents that the particular box was repeated in k filters. Once, we find those box filters, we also have the coordinates (a, b, c, d) with which we can figure out the region where the box filter needs to be applied on I.

Algorithm 1 Latency Improver

function REDUNDANCY REMOVER(Input Integral Image Channel x_0, Empty-Hashmap H, Model ϵ_θ, f_N filters in a Layer)
  Initialize: $x \leftarrow$ Empty-Tensor of size $(B \times f_N \times H \times W)$
  $B_i(a, b, c, d, \alpha) \leftarrow$ Box-Filter Approximation Algorithm(ϵ_θ)
  for each layer in layers **do**
    while $f_i \leq f_N$ **do**
      if $B_i(a, b, c, d)$ already in H **then**
        $output = \alpha_i \cdot H[B_i(a, b, c, d)]$
      else
        $output = \alpha_i \cdot RegionSum(x_1, y_1, x_2, y_2)$
        $H \leftarrow (B_i(a, b, c, d), RegionSum(x_1, y_1, x_2, y_2))$
      end if
      $x[region(x_1, y_1, x_2, y_2)] \leftarrow output$
    end while
  end for
  return x

output channel. Keeping in line with this method, we perform our experiments with ϵ_θ as resnet18 [19] on a simple RGB-channeled dataset.

A. Dataset Details:

We take the CIFAR-10 dataset [4]. It consists of 50000 training images and 10000 testing images. Each image is a 3-channel RGB image with a resolution of 32×32. For our purpose, we take the entire set of training images with a batch size of 64 for finetuning our network. Following, we take a batch size of 5000 for testing purposes. All experiments were done on a single NVIDIA H-100 GPU.

B. Box Filter Approximation

We approximate a kernel with a set of box filters by using the algorithm proposed by [1] to approximate any 2D filter with a set of box filters. This algorithm outputs a set of box filters which can then be stored to use during inference.
Each box filter is defined by four corner points, representing a rectangular region in the original filter. The coordinates of the top-left corner of the box are given as (a, c), and the bottom-right corner is (b, d). This geometric definition allows the filter to localize its effect on a specific region of the input image. The strength or contribution of each box filter is scaled by a factor, denoted as α which adjusts the intensity or weight of the filter within the overall approximation. The final kernel approximation $B_N(x, y)$, representing the original filter applied over the image, is constructed as a weighted sum of these individual box filters:

$$B_N(x, y) = \sum_{i=1}^{N} \alpha_i B_i(x, y) \tag{1}$$

Here, $B_N(x, y)$ is the original filter to be applied on the image, and then $B_i(x, y)$ are the individual box filters. Each of these individual box filters are approximated via the method used in [1] and then stored. The key advantage of this approach lies in its computational simplicity. Box filters allow us to replace a complex filter operation, which typically requires many floating-point multiplications and additions, with simpler arithmetic operations as explained in the next section.

C. Convolution on SAT

The first step in using SATs for convolution is constructing the summed area table. Given an image $I(x, y)$ of size MxN, the summed area table $S(x, y)$ is computed using $S(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i, j)$. Using the SAT, the sum of pixel values in any rectangular region of the image, defined by corners (x_1, y_1) and (x_2, y_2) can be computed in constant time. The sum Σ over this rectangle is as follows:

$$\Sigma(x_1, x_2, y_1, y_2) = S(x_2, y_2) - S(x_1 - 1, y_2) - S(x_2, y_1 - 1) + S(x_1 - 1, y_1 - 1).$$

Here, $S(x_2, y_2)$ is the sum of region from $(0, 0)$ to (x_2, y_2), $S(x_1 - 1, y_2)$ subtracts the region to the left of (x_1, y_2), $S(x_2, y_1 - 1)$ subtracts the region above (x_2, y_1) and $S(x_1 -$

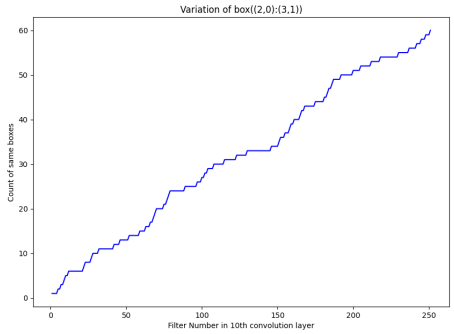

Fig. 2. Increase in the box-filter with coordinates $(2,0):(3,1)$ in 256 filters in the 10th convolution layer for the same input channel.

$1, y_1-1)$ adds back the region that was subtracted twice. Once, we know these values, we can directly access those indexes. Let us now have a box filter with the top-left coordinate being (x_1, y_1) and the bottom-right coordinate being (x_2, y_2). To perform convolution with a box filter over a region defined by a filter $[X_1, Y_1]\times[X_2, Y_2]$, we perform the following operation:

$$\sum_{X_1=0}^{H'}\sum_{Y_1=0}^{W'}\sum_{k=0}^{N}\alpha_k * \Sigma(X_1+x_1, X_1+x_2, Y_1+y_1, Y_1+y_2) \quad (2)$$

Here, H', W' are the height and width of the next output channel after convolution.

We, therefore, notice that if we had a kernel of size $N \times N$ which is performing convolution in a sliding window manner, we would have had to perform N^2 multiplications and N^2-1 addition operations, making the total number of mathematical operations to $2N^2-1$ for every image region. Now, we reduced it to 3 addition operations and 1 multiplication operations. The mathematical operations in this case are constant(**4**) and do not depend on the kernel size N per image region.

IV. Experiments and Results

We perform various experiments proving the efficiency and efficacy of our approach in computing convolution. These experiments are aimed at understanding the effects of using box filter approximated kernels in terms of error and latency. We found the optimal number of boxes for a 7×7 and 3×3 filter to be 32 and 5 respectively which has been used in the following experiments.

A. Redundancy of Box-Filters

As explained above, there are N box filters for a single filter corresponding to a single input channel. When finding these box filters, we noticed that a box filter with the same set of coordinates is being predicted over different filters for the same input channel. Figure 2 shows us a demonstration of the repetition of boxes across the filter dimension for an input channel. From the plot, by the time we reach the 256th filter, we already have 60 occurrences of the box with coordinates $(2,0):(3,1)$. As we can observe, we need to calculate the region sum of the box with coordinates $(2,0):(3,1)$ multiple times which is unnecessary. Note that this box may or may not occur in a filter for that channel. 60 signifies that it has occurred in 60 filters after going over all(here 256) filters.

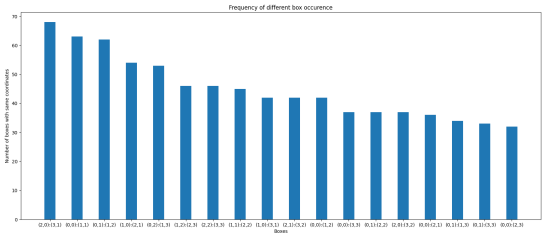

Fig. 3. Visualization of the number of same boxes across different filters for the same input channel

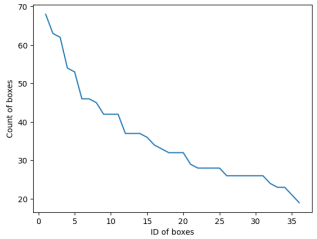

Fig. 4. Visualization of the total number of same boxes across the 10th convolution layer

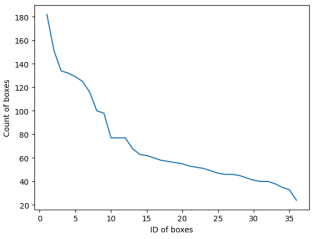

Fig. 5. Visualization of the total number of same boxes across the 14th convolution layer.

We also observe that this is not only for a specific box instance but for multiple boxes as shown in Figure 3. In this figure, we notice that we are calculating ≈60 in the same box with coordinates $(0,1):(1,2)$ with different alphas. Here same boxes mean that the four corner points of that box are the same. Therefore, we are unnecessarily calculating the sum of the elements in that box region multiple times. The graph shows only 50% of the boxes that are being repeated over the filter dimension in a single layer along with the number of filters it has occurred in. Figure 4 and Figure 5 shows us a more extensive demonstration of how many times different boxes are being repeated over the entire filter dimension. For example, the highest occurring box occurs in $\approx$ 180 filters out of 1280 (256×5) filters in the 14th convolution layer. We also note that the minimum times a box occurs in the 10th convolution layer is 21. This means that we will encounter every box at least 21 times after traversing over the filter dimension. Moreover, we also note that as we go deeper into the network, the frequency of common boxes or redundancy increases as in the right figure(Figure 5), we see that the most occurring box has occurred $\approx$ 180 times. Therefore, we aim to calculate the sum over the region covered by these boxes only once and store them in memory so that they can be accessed instantly if we encounter them again in a different filter.

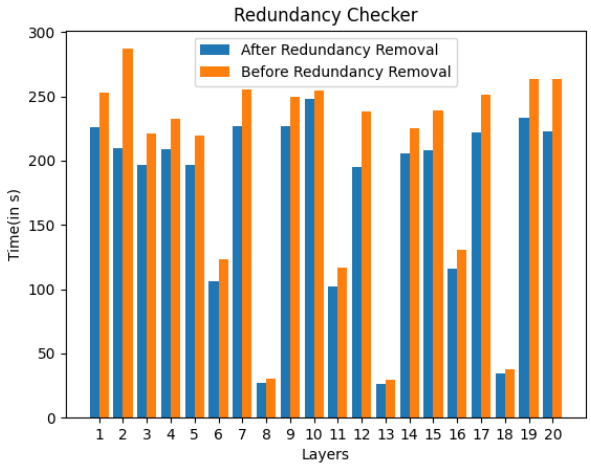

Fig. 6. Comparison between the running time of each convolution layer in resnet18 before and after redundancy removal.

TABLE I
WE REPORT THE 10-CLASS CLASSIFICATION ACCURACY ON THE CIFAR-10 DATASET USING THE STANDARD CONVOLUTION AND OUR PROPOSED BOX-FILTER CONVOLUTION.

Method	Accuracy
Sliding-Window Convolution	79.98%
Box-Filter Convolution	79.97%
Box-Filter Convolution(finetuned)	**82.27%**

B. Latency in Convolution

To avoid recalculating the region-sum, we calculate that sum as and when we encounter it for the first time. In this way, whenever we come across the same set of four corner points, we can directly access the region sum in $O(1)$ time instead of again calculating the region sum using 3 additional addition operations. This reduces the computation power needed further. To demonstrate this numerically, we compare the time taken by each convolutional layer of resnet18 in two scenarios:(1) When computing the box region sum as and when we are given a box using all the boxes every time (orange bar)(2) When calculating a box region-sum only when we encounter it for the first time and then reusing it thereafter(blue bar).
Figure 6 shows us the improvement in time gained by our method. As we can observe, we are improving the running time of every convolutional layer as compared to when recalculating box region-sum every time. We also notice a significant reduction in time in the $2nd$ convolutional layer. Overall, we are decreasing the inference time by 12.35% in resnet18.

C. Accuracy using Box-Filter method

For sanity check, we also compare the classification accuracies obtained by replacing the regular filters with our box filters in Table I. Here, box-filter convolution is performed in a SIMD fashion which avoids the sliding-window manner. Since we have already precomputed the box coordinates, we directly find the indexes in the input channel where we need to operate on and do the operation for every set of boxes. For example, in the first convolution layer, we have 64 filters for each of the 3 channels. Hence, we iterate 192 times and perform convolution each time separately using the set of boxes stored for that particular input channel and filter.

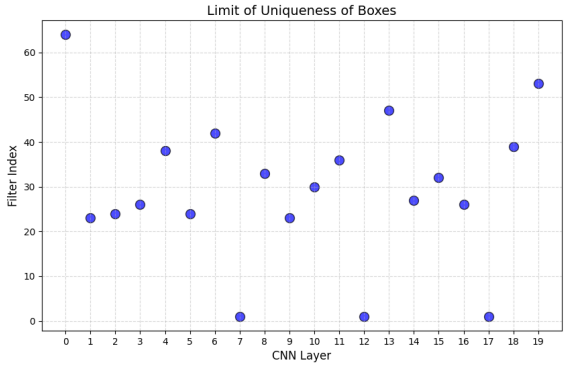

Fig. 7. Visualization of the filter index at which we exhausted all the possible boxes in all the layers for a single channel.

D. Analysis of the results

1) Classification accuracy: As is evident, using the box-filter convolution we achieve the same as the original accuracy and even better if we further finetune the network. This finetuning is done only on the BatchNorm and MLP layers. Our rationale behind this is that since the filter weights have deviated a bit(hence a 0.01% loss), the BatchNorm and MLP layers must also be adjusted accordingly. On finetuning them for 5 epochs with a batch size of 64 on the training set, we obtain the reported accuracy. Thus, not only do we maintain the accuracy but finetuning it can further increase the accuracy.

2) Uniqueness of Boxes: We note that the total possible boxes for a filter of size $K \times K$ is $K^2(K^2+1)/4$. A major reason for the reduction in running time can be attributed to the fact that after a few filters, we have already come across all the possible boxes and hence save computing over the box region in the majority of the filters. This is shown by Figure 7 where each point represents the filter index where we achieve saturation at that convolution layer for a particular input channel.

V. CONCLUSION

As is evident from our extensive experiments, we have successfully reduced the inference time of a popular CNN-based architecture by changing the root method of a convolution operation. In future work, we hope to build more optimized methods of performing the box-filter convolution in a SIMD manner so that we can get rid of the sliding-window technique altogether. We also look forward to decreasing the training time of such architectures using a similar approach without any toll on the classification accuracy.

REFERENCES

[1] B. R. Pires, K. Singh and J. M. F. Moura, "Approximating image filters with box filters," 2011 18th IEEE International Conference on Image Processing, Brussels, Belgium, 2011, pp. 85-88, doi: 10.1109/ICIP.2011.6116693.

[2] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, HI, USA, 2001, pp. I-I, doi: 10.1109/CVPR.2001.990517.

[3] Lewis, J.P.. (1994). Fast Template Matching. Vis. Interface. 95.

[4] Krizhevsky, Alex. "Learning Multiple Layers of Features from Tiny Images." (2009).

[5] Ding, Xiaohan et al. "ResRep: Lossless CNN Pruning via Decoupling Remembering and Forgetting." 2021 IEEE/CVF International Conference on Computer Vision (ICCV) (2020): 4490-4500.

[6] Chen, Hao et al. "Conv-Adapter: Exploring Parameter Efficient Transfer Learning for ConvNets." 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) (2022): 1551-1561.

[7] Mostafa, Hesham and Xin Wang. "Parameter Efficient Training of Deep Convolutional Neural Networks by Dynamic Sparse Reparameterization." International Conference on Machine Learning (2019).

[8] Franklin C. Crow. 1984. Summed-area tables for texture mapping. SIGGRAPH Comput. Graph. 18, 3 (July 1984), 207–212. https://doi.org/10.1145/964965.808600

[9] Barron, Jonathan T. et al. "Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields." 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2021): 5460-5469.

[10] Babiloni, Francesca et al. "Factorized Dynamic Fully-Connected Layers for Neural Networks." 2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW) (2023): 1366-1375.

[11] Viola, P., Jones, M.J. Robust Real-Time Face Detection. International Journal of Computer Vision 57, 137–154 (2004). https://doi.org/10.1023/B:VISI.0000013087.49260.fb

[12] Zhang, Linguang et al. "Accelerating Large-Kernel Convolution Using Summed-Area Tables." ArXiv abs/1906.11367 (2019): n. pag.

[13] Ernesto Tapia. 2011. A note on the computation of high-dimensional integral images. Pattern Recogn. Lett. 32, 2 (January, 2011), 197–201. https://doi.org/10.1016/j.patrec.2010.10.007

[14] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. 2016. Accelerating Very Deep Convolutional Networks for Classification and Detection. IEEE Trans. Pattern Anal. Mach. Intell. 38, 10 (October 2016), 1943–1955. https://doi.org/10.1109/TPAMI.2015.2502579

[15] Kim, Yong-Deok et al. "Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications." CoRR abs/1511.06530 (2015): n. pag.

[16] Han, Song et al. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding." arXiv: Computer Vision and Pattern Recognition (2015): n. pag.

[17] Shiyu Li, Edward Hanson, Hai Li, and Yiran Chen. 2020. PENNI: pruned kernel sharing for efficient CNN inference. In Proceedings of the 37th International Conference on Machine Learning (ICML'20), Vol. 119. JMLR.org, Article 544, 5863–5873.

[18] Cong, Jason and Bing-Yu Xiao. "Minimizing Computation in Convolutional Neural Networks." International Conference on Artificial Neural Networks (2014).

[19] He, Kaiming et al. "Deep Residual Learning for Image Recognition." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 770-778.