Summer Research Report

Speed Up Neural Network Inference

JIN Fenglei

Abstract

Convolutional neural networks (CNN) are widely applied in many fields, and thanks to the rapid development of GPU, different methods are proposed to accelerate the intensive computation. However, with the increasing demand of deploying CNN in mobile devices, speedup network interface for CPU computation becomes a popular topic.

In this paper, I implement Sparse-sparse Convolution proposed by Li [1] in caffe, and test it with some famous neural network to see the acceleration. The experimental result shows that this method leads to good speedup for some layers with sparse inputs and weights compared to traditional convolution method, and can reach the speed of Sparse-matrix-dense-matrix Multiplication(SpMDM)[2], which is the state of the art speedup now. To reduce matrix format conversion penalty, I modify this algorithm and proposed an algorithm which outputs matrix in sparse format.

1. Introduction

Among kinds of layers in DNN, the convolution layer is one of the most important, but the slowest and most memory-intensive ones in advanced/modern convolutional DNN[2]. Though CNNs become more and more powerful in a broad set of practical tasks, these deficiencies limit the development of CNN in a spectrum of platforms, especially smart phones and autonomous cars which are CPU-compute-only and have a strict requirement for memory.

To deal with this, many excellent researches have been done about weights sparsity. Wen et al. aim to accelerate neural network interface by making the weight matrix more sparse by adding group lasso regularization term to the loss function[3]. Cho et al. proposed a developed lowering method to compress the memory-overhead[4]. Park et al. have proposed a different perspective about sparse methods considering that the performance of the sparse matrix operations is typically memory bandwidth bound, so a Sparse-matrix-dense-matrix Multiplication(SpMDM)[2] accelerating method was applied, and becomes the state of the art speed. Park's experiment is based on the high sparsity of weights (showed in Figure1), and according to our research, the input matrix in some deep convolution layers always has a high sparsity, which is showed in Figure 2. The high sparsity of not only weight matrix but also input matrix make it possible to multiple two matrices in sparse format directly[1].

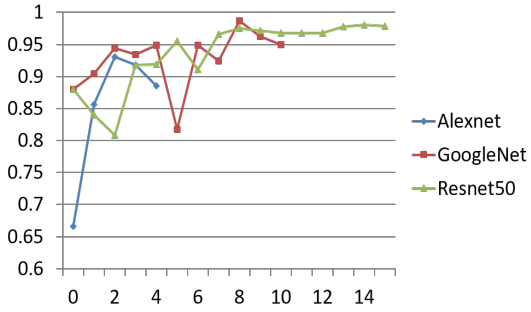


Figure 1: Weight sparsity

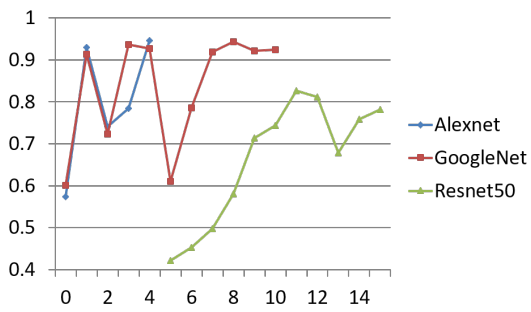


Figure 2: Input sparsity

And since weight matrices can be stored in CSC format after training, we only need to convert input matrices into CSC format in every convolution layer. We find this conversion costs much time and limits the performance, so we propose a new method which multiples two sparse matrices and outputs a matrix in sparse format. The output in sparse format can be multiplied directly as input matrix in the next layer and so on, which only needs once conversion and eliminate the conversion penalty of posterior layers.

Our paper makes the following contributions:

- We implement Sparse-sparse Convolution in caffe interface and achieve $2\times\sim3\times$ speedup in Resnet50 compared to default convolution.
- We proposed a Sparse-in-sparse-out Convolution algorithm to reduce conversion penalty and achieve successive convolution layers speedup theoretically.

The rest of the paper is organized as following: Section 2 presents the details of our Sparse-sparse Convolution design with time and space complexity compared to default convolution calculation algorithm. Section 3 demonstrates the performance of Sparse-sparse Convolution. Section 4 analyze how to improve the performance and propose Sparse-in-sparse-out Convolution algorithm.

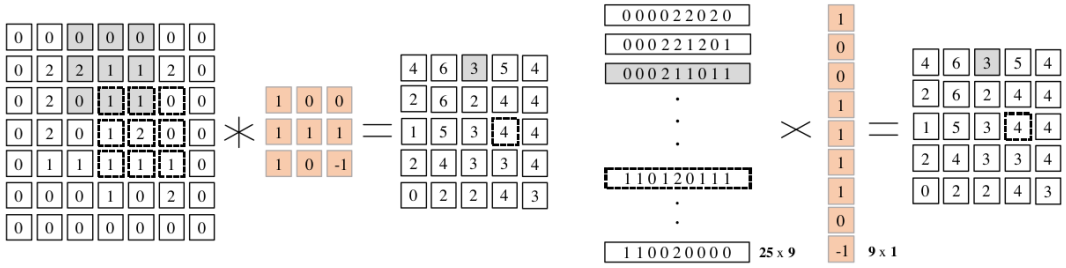
2. Algorithm

This section first demonstrates how convolution are actualized in caffe, then analyze the sparse-weight-sparse-input situation in popular networks. After that, it presents our efficient Sparse-sparse Matrix Multiplication algorithm. Finally, this section compares the time and space complexity with default convolution.

2.1. Image2Column Convolution

Due to geometry-specific constraint, the poor memory access pattern leads direct convolution(relatively short dot product) to poor performance. To achieve BLAS-friendly memory layout, im2col-based convolution transforms/lowers the input matrix into a Toeplitz matrix with redundancy (a.k.a, lowered matrix) such that convolution can be performed as fast

matrix-matrix multiplication(showed in Figure3), which can take advantage of highly optimized linear algebra packages including BLAS[4]. And this method is not only used in caffe, but also applicable for any convolution configuration on any platform. Im2col-based convolution makes it possible to speedup neural networks by optimizing matrices multiplication algorithms.



(a) direct convolution (b) im2col-based convolution with lowered matrix

Figure 3: Conventional convolution examples with 7×7 input size, 3×3 kernel size, 1 kernel stride in both dimensions.

2.2. Sparse Input and Weight

In this section, we will explain the feasibility of sparse input and weight in CNN models.

A lot of work has been done to increase the sparsity of weight matrices before. Structured sparsity learning[3] can increase the sparsity by group lasso which enforces several columns of weight to be all-zeros. Actually if use lasso instead will get higher sparsity because of less limitation. And network pruning has been widely studied to compress CNN models. In early work, network pruning proved to be a valid way to reduce the network complexity and over-fitting. Guided Pruning[2] has been tested in popular networks like Caffenet, GoogleNet and Resnet50. The weight sparsity of pruned models reaches 95% in some layers.

Because of the activation function like ReLU, actually the inputs of last few layers are already very sparse, especially when sparse weights are applied. Though input sparsity may vary from different training or testing data sets, we find that the sparsity of inputs always maintains a high level in last few layers of some deep networks(showed in Figure2), which is enough for our research.

2.3. Sparse-sparse Matrix Multiplication Algorithm

In this section, we will present our efficient sparse-sparse matrix multiplication algorithm.

Consider two sparse matrix M, N with size $m \times k$, $k \times n$, then the size of the output matrix O is $m \times n$. Then M is transformed into CSC(actually weight M can be directly trained and stored in CSC format). N is transformed into CSR format. And the calculation method of output matrix O is showed in Algorithm 1.

Algorithm 1 sparse-sparse matrix multiplication algorithm

Input: weight M is CSC format with 3 arrays: col_ptr, rowidx and val, and input N is CSR format with 3 arrays: row_ptr, colidx, val

Output: Output O

function CSC_MUL_CSR(M, N)
  allocate O with size $m \times n$
  for row_n in range size($N.row\_ptr$)-1 **do**
    for c in range (N.row_ptr[row_n],N.row_ptr[$row_n + 1$]) **do**
      for r in range (M.col_ptr[row_n],M.col_ptr[$row_n + 1$]) **do**
        $O(M.rowidx[r], N.colidx[c]) += M.val[r] \times N.val[c]$
      end for
    end for
  end for
  return O
end function

2.4. Analysis

In this section we analyze the time and size complexity compared to default convolution algorithm.

2.4.1. Time Complexity

We firstly consider the time complexity of Sparse-sparse Matrix Multiplication. The key point is that this algorithm ignores all element multiplications with zero. Consider two sparse matrix M(CSC format), N(CSR format) with size $m \times k(W_{size})$, $k \times n(I_{size})$, and their sparsity are α, β respectively. For each element in M, they need to multiply with elements in N whose row index equals to their own column index. Because of the CSR format, multiplication has no need to go through all elements in N but only elements with equal row index. Therefore each element will occupy $(1-\beta) \times n$ operations. The whole time complexity of the matrix multiplication is $mkn(1-\alpha)(1-\beta)$. Cause we need to transform N(input matrix) into CSR format first, and this operation has to go through all elements in N, the time complexity is $k \times n$. So the whole number of floating point operations is:

$$T_{sparse} = kn + mkn(1-\alpha)(1-\beta) = I_{size} + \frac{I_{size}W_{size}(1-\alpha)(1-\beta)}{k}, \tag{1}$$

The time complexity of default matrix multiplication is mkn. Therefore this algorithm achieve $\frac{1}{(1-\alpha)(1-\beta)}\times$ speedup compared to default matrix multiplication theoretically. However, because CPU architecture is very friendly to matrix multiplication, the algorithm works when matrices have high sparsity in reality. In the experiment, the speedup is obvious when the two sparsity reach 0.8.

2.4.2. Space Complexity

Considering that the weight can be trained and stored in CSC format, the space complexity reduce to $W_{size}(1-\alpha)$. The required input should be transformed into CSR format,

so the memory overhead is $I_{size}(1-\beta)$. The space complexity is:

$$S_{sparse} = W_{size}(1-\alpha) + I_{size}(2-\beta), \tag{2}$$

Compared to the space complexity of default multiplication which is $W_{size} + I_{size}$, in our experiment, our algorithm only costs around 65% memory compared to default method when sparsity reach 0.9 and 0.75.

3. Experimental Result

We have implemented the Sparse-sparse Convolution in caffe interface and test the speedup on some convolution layers of Caffenet and Resnet50. The trained weights can be easily download at SkimCaffe[1]. The test input are 1000 images download at ImageNet of 20 different classes. The test is done on Ubuntu18.04 with CPU Intel Core i5-8250U. The result of Caffenet and Resnet-50 are shown in Figure 4. The red cylinder part represents the time of converting input matrix to CSR format, and the blue cylinder part represents the time of Sparse-sparse Matrix Multiplication. The green cylinder represents the origin time of default convolution. The purple cylinder represents the time of SpMDM.

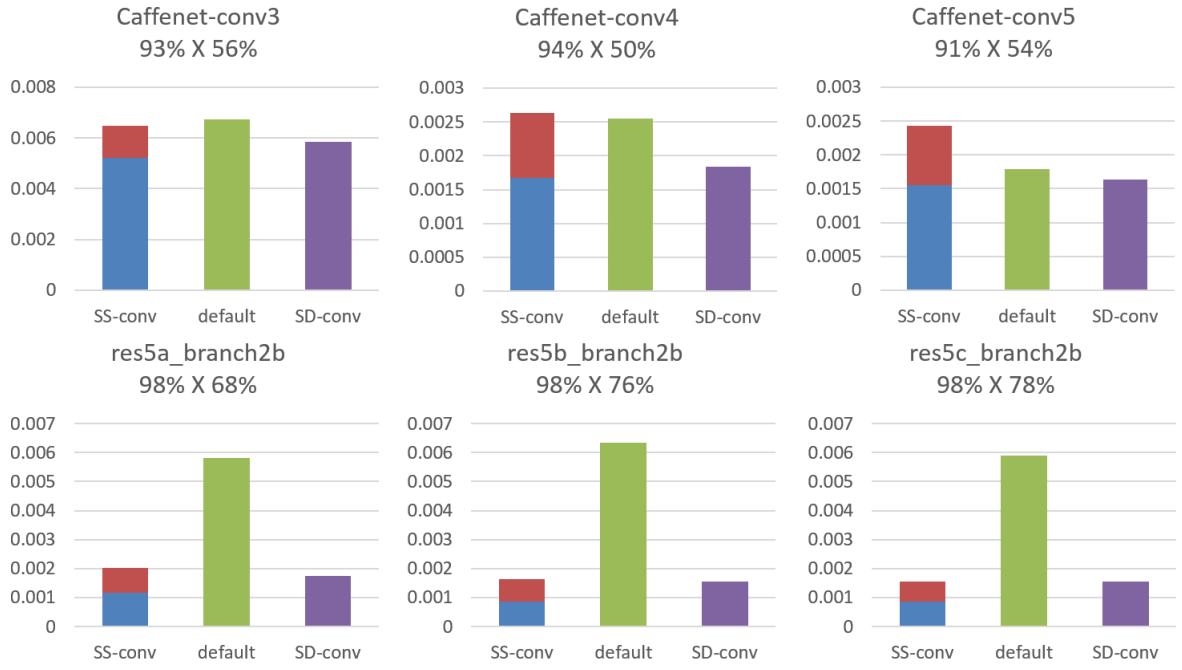


Figure 4: The left percentage is the sparsity of weight matrix, while the right is input sparsity. The first cylinder represents the time of Sparse-sparse Convolution: input2CSR(red) plus Sparse-sparse Matrix Multiplication(blue), the middle cylinder represents default time, and the purple one represents SpMDM time.

From the result we can find that:

[1]https://github.com/IntelLabs/SkimCaffe

- Different layer architecture consult in different speed in Sparse-sparse Convolution. The higher the ratio of weight size to input size, the faster the compute.
- Cause the large input size and low sparsity of Caffenet convolution layers, the algorithm even slow down the convolution. The penalty of input2CSR stunts the performance.
- Sparse-sparse Convolution achieves good speed up on the last few layers of Resnet50. Cause the network is deep enough, the input matrices reach high sparsity. The speedup reaches 2.92×∼3.88×, which is similar to the state-of-art speed. However, the input2CSR part is still a big penalty.

4. Advancement and Future Work

As showed in the result, the penalty of input2CSR limits the performance of Sparse-sparse Convolution. To eliminate this penalty, a new algorithm called Sparse-in-sparse-out Convolution is proposed. This algorithm outputs a matrix in CSR format, which can be directly used and the input sparse matrix of the next convolution layer, and continue. Therefore, this algorithm needs only once input2CSR conversion which reduces the conversion penalty of behind layers. The details are showed in Algorithm 2.

Algorithm 2 Sparse-sparse Matrix Multiplication(output CSR)

Require: Weight M and Input N in CSR format with 3 arrays: row_ptr, colidx, val
Ensure: Output O in CSR format with 3 arrays: row_ptr, colidx and val

function CSR_MUL_CSR(M, N)
  for row_m in range size($M.row\_ptr$)-1 **do**
    Initialize temp[$N.col.size$]
    for c in range (M.row_ptr[row_m],M.row_ptr[$row_m + 1$]) **do**
      for r in range (N.col_ptr[$M.colidx[c]$],M.col_ptr[$M.colidx[c] + 1$]) **do**
        $temp(N.colidx[r]) += M.val[c] \times N.val[r]$
      end for
    end for
    Detect whether the elements in $temp$ equals to zero, update 3 arrays of O
  end for
  return O
end function

The key idea is also ignoring all multiplications with zero and directly detecting whether the output row has non-zero elements. If so, add them to output in CSR format. Therefore, the time complexity of the converting convolution layer is also $I_{size}+I_{size}W_{size}(1-\alpha)(1-\beta)/k$, but the time complexity reduce to $I_{size}W_{size}(1-\alpha)(1-\beta)/k$ in behind convolution layers. And cause the posterior inputs are now in CSC format, the total space complexity reduce to $W_{size}(1-\alpha)+I_{size}(1-\beta)$.

After some naive test on Sparse-in-sparse-out Convolution, we find that this algorithm does achieve some speedup compared to the Sparse-sparse Matrix Multiplication. However, further work should be done to beat SpMDM:

- Implementing Sparse-in-sparse-out Convolution in caffe means not only modifying the convolution part, but also the whole network stream and data flow.
- Some Output needs to pad zeros or go through ReLU layer before entering the next convolution layer, special padding and activation algorithm should be studied.

5. Acknowledgment

My deeper gratitude goes from first and foremost to my summer research advisor Prof. Yu and PhD mentor Mr. Yuzhe Ma. Their guidance and experience help a lot to finish my research. And thanks to all authors in reference papers, I can "stand on their shoulders" to study more interesting methods though I am new to this field. Last but not the least, I would thank Engineering Faculty for providing me this chance. This experience encourages me to think and study in a researcher's perspective.

6. Reference

[1] Li, Wei. "Sparse-sparse Convolution Acceleration in Deep Convolutional Neural Network." CUHK Summer Research. 2017.
[2] Park, Jongsoo, et al. "Faster CNNs with Direct Sparse Convolutions and Guided Pruning." arXiv preprint arXiv:1608.01409 (2016).
[3] Wen, Wei, et al. "Learning structured sparsity in deep neural networks." Advances in Neural Information Processing Systems. 2016.
[4] Cho, Minsik, and Daniel Brand. "MEC: memory-efficient convolution for deep neural network." arXiv preprint arXiv:1706.06873 (2017).