

 Personal Open source Business Explore Pricing Blog Support


This repositorySearchSign inSign up


 Yangqing / **caffe**
forked from BVLC/caffe


Watch15Star42Fork5,552

<> Code

 Pull requests 0

 Wiki

 Pulse

 Graphs

Convolution in Caffe: a memo

Philip edited this page on 19 Jun 2015 · 11 revisions

In the last few months chatting with people about Caffe, a common comment I got was: "*Caffe's convolution has some memory issues.*"

While this is true in some sense, I am not sure whether it is truly an issue - rather, **it is a graduate-student level design choice when I was writing the Caffe framework in just 2 months' budget with a looming thesis deadline.** It turns out to have its own pros (faster than any trivial implementation unless you optimize really seriously) and cons (large memory consumption). A more detailed explanation follows, if you are interested.

First of all, convolution is, in some sense, quite hard to optimize. While the conventional definition of convolution in computer vision is usually just a single channel image convolved with a single filter (which is, actually, what Intel IPP's convolution means), in deep networks we often perform convolution with multiple input channels (the word is usually interchangeable with "depth") and multiple output channels.

Loosely speaking, assume that we have a $W \times H$ image with depth D at each input location. For each location, we get a $K \times K$ patch, which could be considered as a $K \times K \times D$ vector, and apply M filters to it. In pseudocode, this is (ignoring boundary conditions):

```
for w in 1..W
  for h in 1..H
    for x in 1..K
      for y in 1..K
        for m in 1..M
          for d in 1..D
            output(w, h, m) += input(w+x, h+y, d) * filter(m, x, y, d)
          end
        end
      end
    end
  end
end
end
end
```

Optimizing such a complex nested for loop is non-trivial. If you have tried optimizing matrix multiplication, you know that there is a lot of tricks involved in it. In my third year of PhD I did it in Berkeley's CS267 (parallel computers) class, and we got the highest maximum performance [1] among others after having two layers of caching, unrolled innermost computation, and SSE2 (AVX wasn't available on the cluster we used). Optimization of convolution could only be more complex.

I have long admired Alex Krizhevsky's great optimization in cuda-convnet [2]. Unfortunately, I had to finish my thesis at the time, and I probably was not as GPU savvy as he is. But I still needed a fast convolution. Thus, I took a simpler approach: reduce the problem to a simpler one, where others have already optimized it really well.


The trick is to just lay out all the local patches, and organize them to a $(W \times H, K \times K \times D)$ matrix. In Matlab, this is usually known as an `im2col` operation. After that, consider the filters being a $(M, K \times K \times D)$ matrix too, the convolution naturally gets reduced to a matrix multiplication (Gemm in BLAS) problem. We have awesome BLAS libraries such as MKL, Atlas, and CuBLAS, with impressive performances. This applies to GPUs as well, although GPU memory is indeed more "precious" than its CPU sibling. However, with reasonably large models such as ImageNet, Caffe has been working pretty OK. It is even able to process videos with such models, thanks to the recent advances in GPU hardware.


▼ Pages 5

Home
Convolution in Caffe: a memo
Development Hints
Publications
Setting up Caffe on Ubuntu 14.04

Clone this wiki locally

https://github.com/Yangqing/caffe/wiki/Convolution-in-Caffe:-a-memo



 Clone in Desktop

An additional benefit is that, since BLAS is usually optimized for all platforms by the BLAS distributors (like Intel and Nvidia), we don't need to worry about optimization with specific platforms, and can sit back comfortably assuming that a reasonably fast speed can be achieved on almost all platforms that those BLAS libraries support.

Somewhat surprising to me is that, such a "lazy optimization" is quite efficient, better than several other approaches and only recently been beaten (as expected) by the mighty Krizhevsky's optimized cuda-convnet2 code [3]:

<https://github.com/soumith/convnet-benchmarks> (as of Jul 27, 2014)

So this is the story of the memory issue that came into play. I didn't mean to defend it - in any means, it was designed as a **temporary** solution. Whenever this word pops up, it reminds me of this [4]:

```
// somedev1 - 6/7/02 Adding temporary tracking of Login screen
// somedev2 - 5/22/07 Temporary my ass
```

So we are having a lot of improvements being worked on by Berkeley folks and collaborators - things will apparently get better soon - expect a faster yet more memory-efficient convolution this fall.

For me, the lesson learned when writing Caffe as a grad student is simplicity: **reduce my problem to one already solved**. It sometimes burns [5], but could always be improved later if necessary.

[1] <http://www.cs.berkeley.edu/~ballard/cs267.sp11/hw1/results/>

[2] <http://code.google.com/p/cuda-convnet/>

[3] <https://code.google.com/p/cuda-convnet2/>

[4] <http://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered>

[5] http://www.reddit.com/r/Jokes/comments/27bgl8/a_math_joke/