CONVOLUTION METHODS

Keren Zhou

July 24, 2016

Institute of Computing Technology, Chinese Academy of Sciences

CONTENTS

MOTIVATION

After handling some CNN research with Prof. Tan, including CNN parallelism and architecture based optimizations, I figure out that various methods, which have different complexities, memory consumptions, and data access patterns, could compute convolution.

Therefore, I propose the slides to show two things:

1. How these method compute convolution?
2. What are these methods' advantages and disadvantages?

INTRODUCTION

The definition of 2-d convolution:

$$x_{i,j}^{l} = \sum_{u=0}^{fh} \sum_{v=0}^{fw} w_{u,v}^{l-1} x_{i+u,j+v}^{l-1}$$

- $x_{i,j}^{l}$: value of the i_{th} row and j_{th} column in layer l feature map.
- $w_{u,v}^{l-1}$: value of the u_{th} row and v_{th} column in layer $l-1$ filter.
- fh : height of the filter
- fw : width of the filter
- u : filter height index
- v : filter width index

DEFINITION

The definition of 2-d convolution:

$$x^{l}_{i,j} = \sum_{u=0}^{fh} \sum_{v=0}^{fw} w^{l-1}_{u,v} x^{l-1}_{i+u,j+v}$$

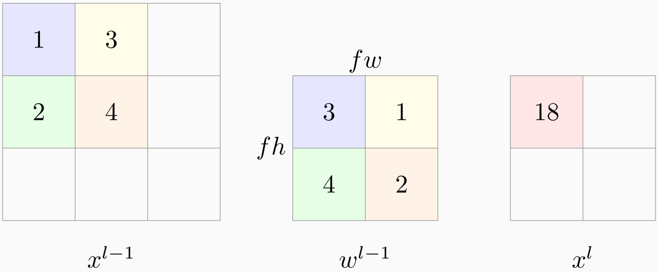


Figure: A convolution example

LIST OF ALGORITHMS

- Direct Convolution
  - Multi-stride Convolution
  - General Single Stride Convolution
- Matrix Multiplication
  - Batch Sensitive
  - Batch Independent
- FFT Based Convolution

ALGORITHMS

ZERO PADDING

Consider each input dimension to be [iw, ih]. We introduce two additional parameters:

- p : the padding size of each input
- s : the convolution stride

Before applying a convolution, we have to transform our input from [iw, ih] to [iw + 2 ∗ p, ih + 2 ∗ p]. We call some a paradigm as zero padding

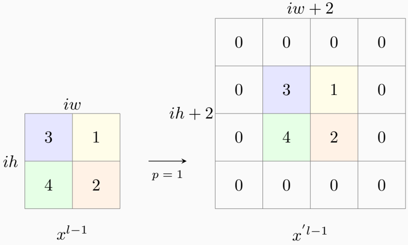


Figure: A zero padding example

This is what we called multi-stride convolution in contrast with the prior single stride convolution.

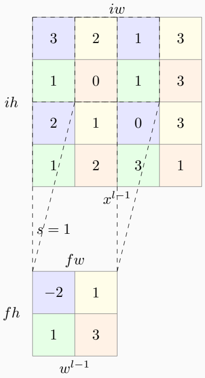


Figure: A single-stride example

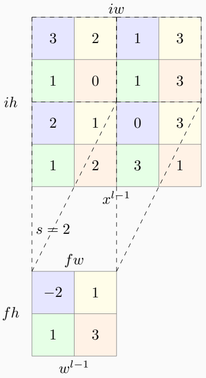


Figure: A multi-stride example

Adopting the stride parameter s, we have to transform the original formula.

Original:

$$x^{l}_{i,j} = \sum_{u=0}^{fh} \sum_{v=0}^{fw} w^{l-1}_{u,v} x^{l-1}_{i+u,j+v}$$

Add s and p:

$$x'^{l}_{i,j} = \sum_{u=0}^{fh} \sum_{v=0}^{fw} w^{l-1}_{u,v} x'^{l-1}_{s*i+u,s*j+v}$$

The MKL vslsConvExec function could calculate a single stride convolution very efficiently. But how could we use it to calculate the multi-stride convolution? We rearrange pixels beforehand.

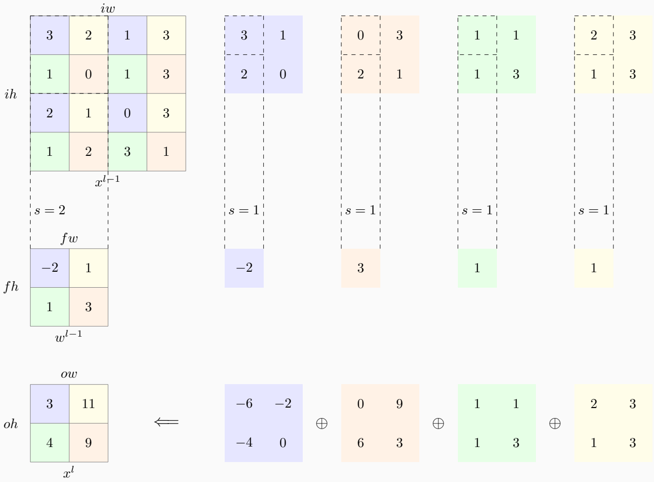


Figure: A sparse convolution example

Sparse convolution formulas:

Original:

$$x^l_{i,j} = \sum_{u=0}^{fh} \sum_{v=0}^{fw} w^{l-1}_{u,v} x^{l-1}_{s*i+u,s*j+v}$$

Sparse:

$$x^l_{i,j} = \sum_{y=0}^{s} \sum_{x=0}^{s} \sum_{u=0}^{fh/s} \sum_{v=0}^{fw/s} w^{l-1}_{u',v'} x^{l-1}_{i'+u',j'+v'}$$

- u'=s*u
- v'=s*v
- i'=s*i
- j'=s*j
- fh > s and fw > s

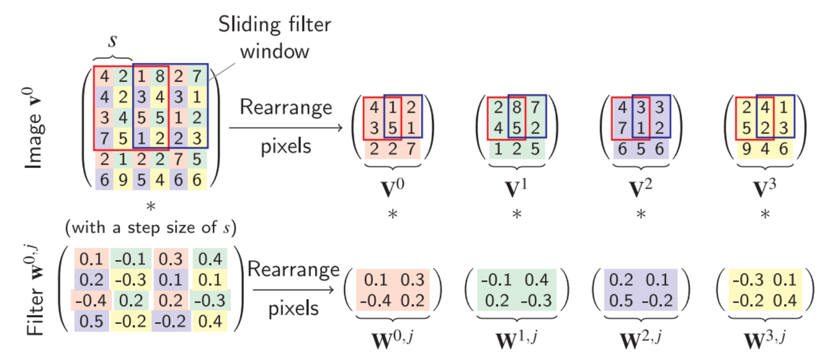


Figure: A concrete example [BT14]

SIGNIFICANCE

- Utilize existing high performance convolution libraries.
- Utilize a direct FFT implementation [ZLS15].
- Design large-scale convolution networks [ZLS15].
- Provide great improvement in pixel-wise convolution applications [LZW14].

In convolution architectures, each input holds raw pixel values of an image. (eg. an image of width 32, height 32, and three color channels R,G,B.)

We assign each channel a filter and compute them accordingly. Further, we add results to form an output channel.

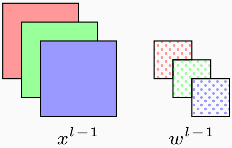


Figure: A three-channel input

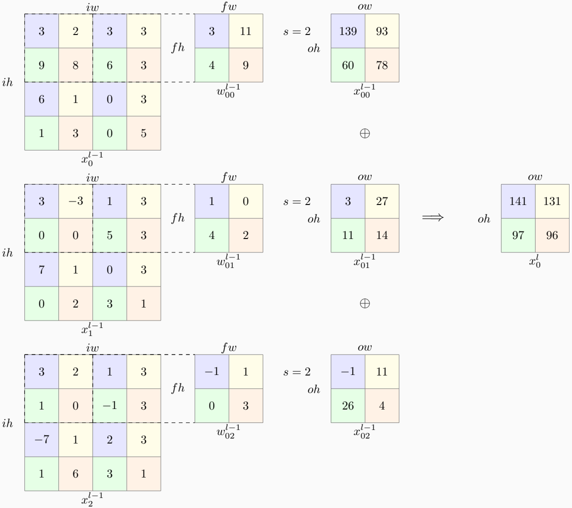


Figure: A multi-input convolution example

MULTIPLE OUTPUT CHANNELS

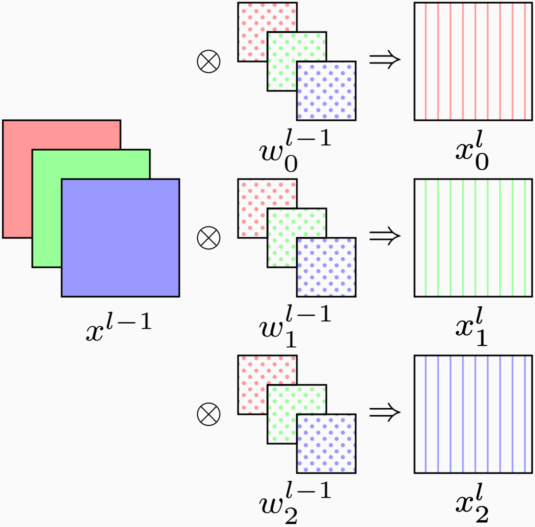


Figure: A multi-output example

DIRECT ALGORITHM

$$x^l_{k,i,j} = \sum_{c=0}^{C} \sum_{y=0}^{s} \sum_{x=0}^{s}$$

$$\sum_{u=0}^{fh/s} \sum_{v=0}^{fw/s} w^{l-1}_{k,c,u',v'} x^{l-1}_{c,i'+u',j'+v'}$$

- k: output channel index
- c: input channel index
- C: input channels

set up strides
for $i \leftarrow 0, oc$ **do**
  for $i \leftarrow 0, ic$ **do**
    $s \leftarrow \text{direct\_conv}(w_{oc,ic}, x^{l-1}_{ic})$
    $x^l_{oc} \leftarrow x^l_{oc} + s$
  end for
end for

Figure: Direct algorithm for multi-channel convolution

$$x^{l}_{n,k,i,j} = \sum_{c=0}^{C} \sum_{u=0}^{fh} \sum_{v=0}^{fw} w^{l-1}_{k,c,u,v} x^{l-1}_{n,c,s*i+u,s*j+v}$$

- n : batch size

Therefore, each batch could use the same filter, and they are calculated independently.

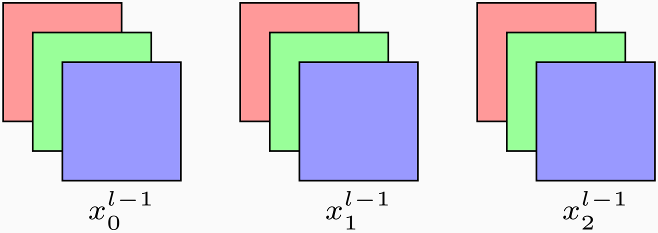


Figure: An input of three batches

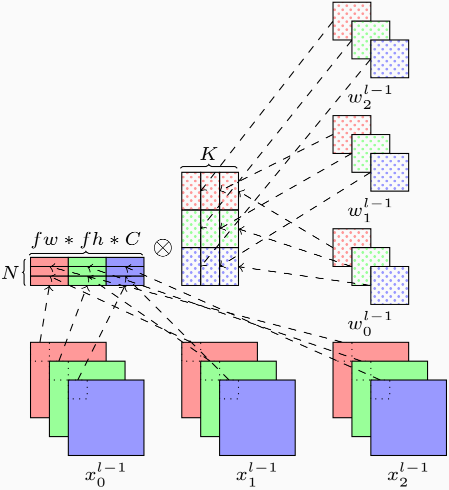


Figure: An example of batch sensitive convolution

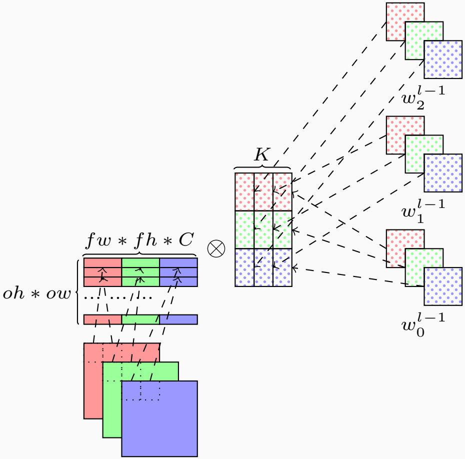


Figure: An example of batch independent convolution [Che+14]

FFT

For 2 length N sequences, each FFT takes Nlog(N) computations, while the naive method takes N^2 computations.

$F\{x * w\} = F\{f\}.F\{w\}$
$x * w = F^{-1}\{F\{f\}.F\{w\}\}$
$*$: convolution
. : multiplication
F : fourier transform
F^{-1} : inverse fourier transform

Table: FFT vs naive method [LLC16]

N	FFT	naive
4	176	16
32	2560	1024
64	5888	4096
128	13312	16384
256	29696	65536

In practice, we could apply mkl_convolution to compute a single channel convolution by FFT.

set up strides
for $i \leftarrow 0, oc$ **do**
  for $i \leftarrow 0, ic$ **do**
    $s \leftarrow \text{direct\_conv}(w_{oc,ic}, x_{ic}^{l-1}, \text{FFT})$
    $x_{oc}^{l} \leftarrow x_{oc}^{l} + s$
  end for
end for

Figure: Direct FFT algorithm for multi-channel convolution

PARALLELISM

Based on multi-stride convolution:

$$x^{l}_{n,k,i,j} = \sum_{c=0}^{C} \sum_{u=0}^{fh} \sum_{v=0}^{fw} w^{l-1}_{k,c,u,v} x^{l-1}_{n,c,s*i+u,s*j+v}$$

The output channel (k) and batch (n) could be computed in parallel without synchronization.

If c, fh, or fw is computed in parallel, we say the parallelism is fine grain. Otherwise, If k or n is computed in parallel, we say the parallelism is coarse grain [Tal16].

FINE GRAIN

Fine grain parallelism induces:

- Redundant pack
- Synchronization cost

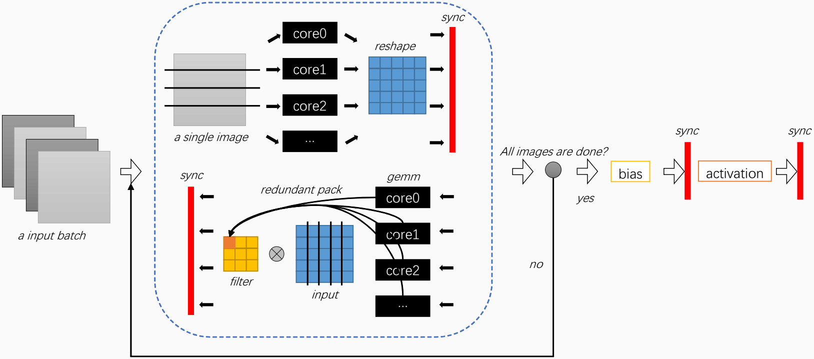


Figure: A fine-grain parallel convolution

Coarse grain parallelism aims to:

- Fuse bias and activation
- Reduce synchronization cost
- Avoid redundant packing

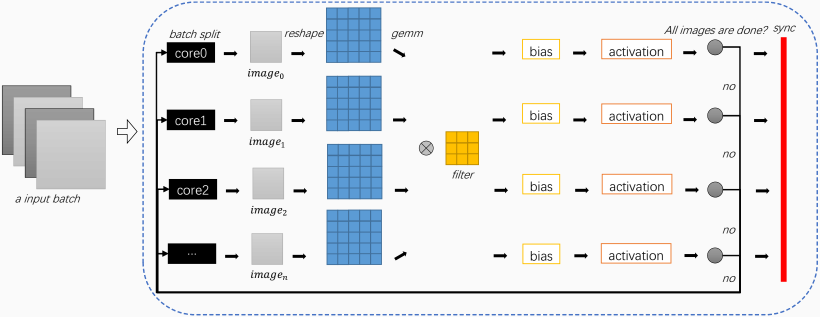
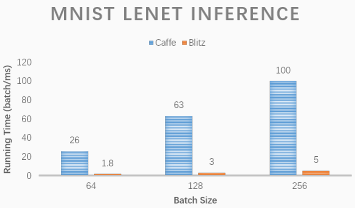
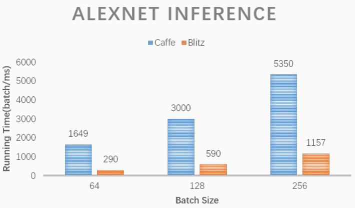
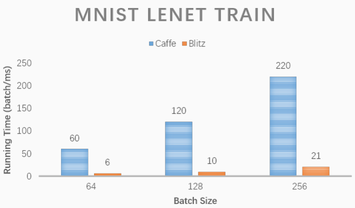


Figure: A coarse-grain convolution example

E5-2670, ICC 16.0, MKL 11.3.2







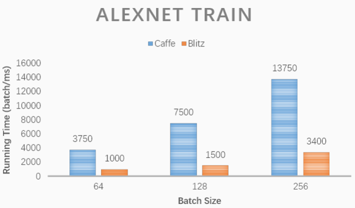


Figure: Compare coarse grain and fine grain

ANALYSIS

Table: FFT vs naive method

	Pre-packing	Computation
Multi-stride	0	2 * N * ow * oh * fh * fw * C * K
Direct	0	2 * N * ow * oh * fh * fw * C * K
Direct FFT	0	3 * N * C * ow * oh * log(ow) * (C + K + C * K)
Batch Sensitive Gemm	N * C * fw * fh * ow * oh	2 * N * ow * oh * fh * fw * C * K
Batch insensitive Gemm	N * C * fw * fh * ow * oh	2 * N * ow * oh * fh * fw * C * K

USAGE

Table: Compare different frameworks

Framework	CPU-Parallelism	CPU	GPU-Parallelism	GPU
Caffe	fine grain	batch insensitive gemm	corase grain	cudnn
Torch	most coarse grain	multi-stride	corase grain	cudnn
Tensorflow	fine grain	direct	corase grain	multi-stride
Neon	most fine grain	batch sensitive gemm	corase grain	multi-stride

PROS AND CONS

- **Multi-stride** is simple to understand but lack of efficient optimizations.
- **Direct** has vendor implementations but depends on the shape of the input and the filter.
- **FFT** is theoretically effective but only performs well on large kernels.
- **GEMM** is clear and efficient but needs extra memories and a packing process.

CONCLUSION

SUMMARY

- Convolution Methods
  - Direct Convolution
  - Matrix Multiplication
  - FFT Based Convolution
- Parallelism
  - Coarse grain
  - Fine grain
- Analysis
  - Complexity
  - Usage
  - Pros and Cons

Questions?

Tom Brosch and Roger Tam. “Efficient training of convolutional deep belief networks in the frequency domain for application to high-resolution 2D and 3D images”. In: Neural computation (2014).

Sharan Chetlur et al. “cudnn: Efficient primitives for deep learning”. In: arXiv preprint arXiv:1410.0759 (2014).

Hongsheng Li, Rui Zhao, and Xiaogang Wang. “Highly efficient forward and backward propagation of convolutional neural networks for pixelwise classification”. In: arXiv preprint arXiv:1412.4526 (2014).

Aleksandar Zlateski, Kisuk Lee, and H Sebastian Seung. “ZNN-A fast and scalable algorithm for training 3D convolutional networks on multi-core and many-core shared memory machines”. In: arXiv preprint arXiv:1510.06706 (2015).

MultiMedia LLC. FFT Convolution vs Direct. 2016. URL: `https://ccrma.stanford.edu/~jos/ReviewFourier/FFT_Convolution_vs_Direct.html` (visited on 07/22/2016).

Marc Gonzalez Tallada. “Coarse grain parallelization of deep neural networks”. In: Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM. 2016, p. 1.