Auto-tuner that builds optimized kernels for convolution layers on CNNs

Marcelo P. Novaes[1,2]

[1]Pervasive Parallelism Laboratory
Stanford University

[2]Department of Computer Science
Universidade Federal da Bahia (UFBA)

Summer, 2015

Outline

Outline

1. Introduction
   - Convolutional Neural Networks
   - Multi-stage programming with Lua and Terra
   - Project specification
2. Development
   - Direct
   - Lowering
   - FFT based (also called "Fast Convolution")
3. Experiments
   - Comparison between approaches
4. Conclusion
   - Limitations and suggested improvements
   - Products, possible future steps and summer experience
   - Acknowledgments

Convolutional Neural Networks

- Neural Network which exploit the spatially-local correlation of input
- one of the most promising techniques to tackle large scale learning problems
- core to applications such as as image and face recognition, audio and speech processing and natural language understanding
- composed by three main layers: convolution, pooling (sub-sampling) and fully-connected (usually the last)
- Their bottleneck are the convolution layers, responsible for most part of the computation on Convolutional Neural Networks (between 70% to 90%)

Convolutional Neural Networks

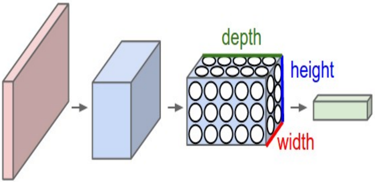

Figure: Convolutional Neural Network [6]

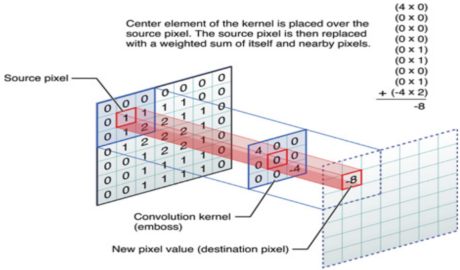

Figure: Spatial Convolution https://developer.apple.com/library/ios/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html

Discrete Convolution

$(f * g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f[m]\, g[n-m]$

Circular Discrete Convolution

$(f * g_N)[n] \equiv \sum_{m=0}^{N-1} \left(\sum_{k=-\infty}^{\infty} f[m+kN]\right) g_N[n-m]$, g_N is periodic, with period N

- Direct and Lowering compute the finite discrete convolution.
- FFT-based computes a circular discrete convolution

Convolutional Neural Networks

Trade-offs

- FFT-based has the best 2D Time Complexity $\theta(nlogn)$. It can have numerical problem (round-off leading to inaccuracy), windowing problems (circular conv.) and large amount of memory wasted with padding (e.g. power of 2 and same sizes).
- Lowering has pre-computation that also requires memory expansion and pos-computation. Relies on contiguous memory (cache friendly use) of blocked GEMM.
- Direct method has to deal with boundary cases (solution can be padding also), has non-contiguous memory problems and it is usually optimized only for part of the usual deep learning parameter space.

Outline

1. Introduction
   - Convolutional Neural Networks
   - Multi-stage programming with Lua and Terra
   - Project specification
2. Development
   - Direct
   - Lowering
   - FFT based (also called "Fast Convolution")
3. Experiments
   - Comparison between approaches
4. Conclusion
   - Limitations and suggested improvements
   - Products, possible future steps and summer experience
   - Acknowledgments

- Interoperability without Glue (e.g. shared lexical environment)
- Low level of abstraction suited for writing high-performance code (e.g. vector instruction, memory manually managed, prefetching intrinsics)
- Backwards compatible with C
- Others: Hygienic staging programming, type reflection. About Lua: tables and functions as first-class citizens.
- During development: active support, nice documentation, tests/ examples and functions such as disas() and printpretty().

```
function blockedComplexloop(M,N,blocksizes,bodyfn)
  local function generatelevel(n,ii,jj,bb0,bb1)
    if n > #blocksizes then
      return bodyfn(ii,jj)
    end
    local blocksize = blocksizes[n]
    return quote for i = ii,min(ii+bb0,M),blocksize do
                   for j = jj,min(jj+bb1,2*N),2*blocksize do
                      [ generatelevel(n+1,i,j,blocksize,blocksize) ]
                 end end end
  end
  return generatelevel(1,0,0,M,N)
end
```

Figure: A blocking example for complex numbers

```
if number == double then
    terralib.saveobj("../bin/my_dconv.o", {my_numconv = my_numconv})
else
    terralib.saveobj("../bin/my_sconv.o", {my_numconv = my_numconv})
end
```

Figure: Single/Double Precision (all computation done over number)

Outline

1. Introduction
   - Convolutional Neural Networks
   - Multi-stage programming with Lua and Terra
   - Project specification
2. Development
   - Direct
   - Lowering
   - FFT based (also called "Fast Convolution")
3. Experiments
   - Comparison between approaches
4. Conclusion
   - Limitations and suggested improvements
   - Products, possible future steps and summer experience
   - Acknowledgments

Project specification

- Given necessity of different convolution methods for different convolution layers
- Given Terra flexibility

Build an auto-tuner that generates optimized kernels for convolution layers. Given convolution layer parameters, decide the best method and auto-tuned parameters.

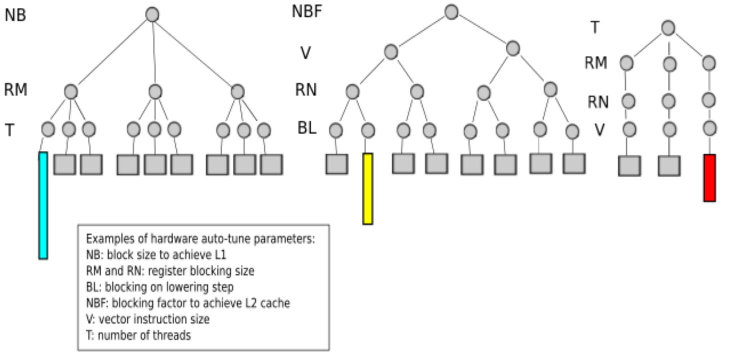

Figure: Search for the best execution time over the three methods

Outline

- Approach based on Terra's GEMM auto-tuner
- Features: 3-level blocking, variable reuse on blocking (pointer optimization), pre-load filter, data prefetching, Multi-thread (using C pthreads).
- Implemented, but not used to generate results (due to low performance): vector instruction based on AVX dot product, vector instruction based on multiplication and AVX hadd, vector instruction multiply and iteration (they are on tests/direct-vec).

Direct 2D

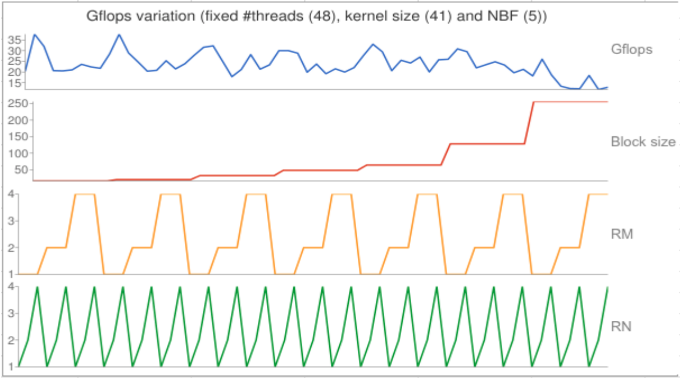

Figure: Example of Gflops variation over blocking parameters

Direct

2D Convolution (Spatial)

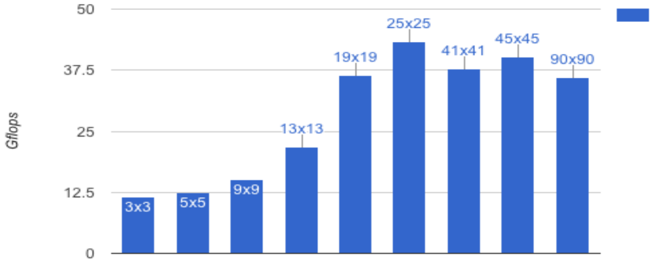

Figure: Optimum points (Double precision)

Direct

3D Convolution (filter batch)

- Extension for filter batch.
- Uses the maximum image reuse approach.
- Same features as Direct 2D plus the maximum image reuse during each 3rd level blocking.
- Drawback: slides over memory for each mini-block.

3D Convolution (filter batch)

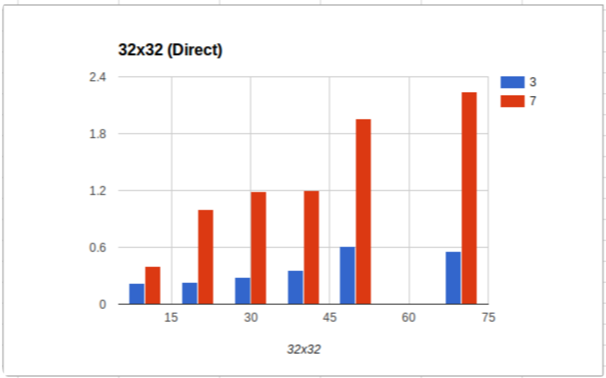

Figure: Execution time by number of kernels

Direct

3D Convolution (filter batch)

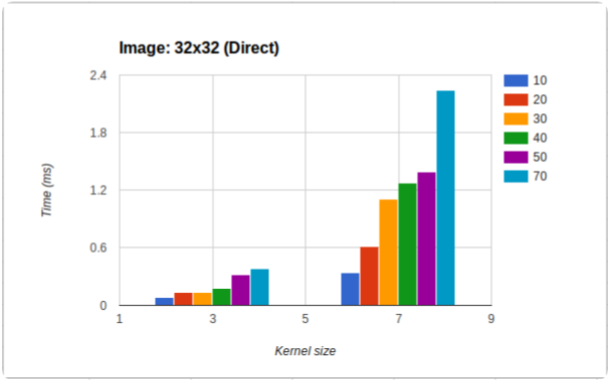

Figure: Execution time by kernel size

Outline

Lowering

- Goal: contiguous memory access when applying image to filter
- It is composed by three steps: (1) Lowering, (2) Matrix Multiplication, (3) Lifting
- Drawbacks: Memory expansion on image lowering. Input preparation (lowering of image and filter). Pos-computaiton lifting the result back.
- Relies on efficiency of GEMM libraries such as MKL GEMM, ATLAS, OpenBLAS (GOTO2BLAS), cuBLAS (for GPUs).
- Features: Blocked image lowering, Multi-threaded (by C pthreads) standard Terra's GEMM (variable reuse, data prefetching, vector instruction, 3-level blocking).

Lowering
Image

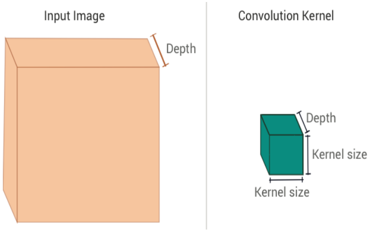

Figure: Tensors inputs

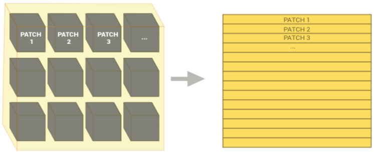

Figure: Image lowering

Lowering
Image

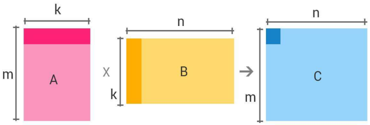

Figure: Output (C) obtained from lowered image (A) multiplied by lowered kernel (B)

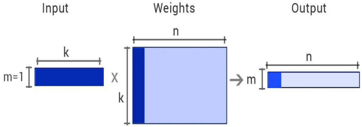

Figure: Convolution correspondence of each output element

Lowering

Results

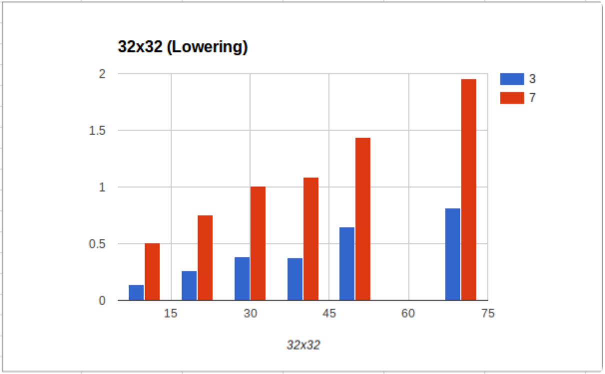

Figure: Execution time by number of kernels

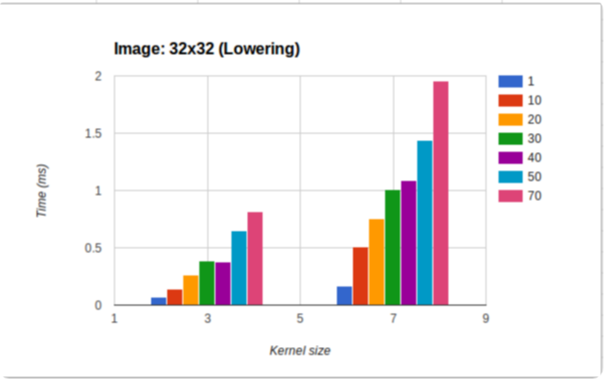

Figure: Execution time by kernel size

Lowering
Results

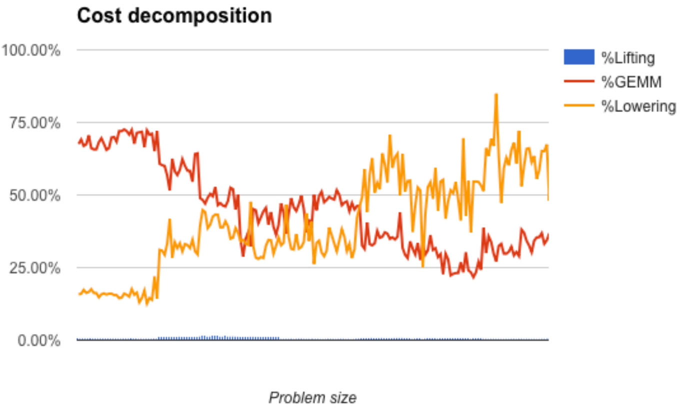

Figure: Decomposition cost (before blocking on image lowering step)

Outline

1. Introduction
   - Convolutional Neural Networks
   - Multi-stage programming with Lua and Terra
   - Project specification
2. Development
   - Direct
   - Lowering
   - FFT based (also called "Fast Convolution")
3. Experiments
   - Comparison between approaches
4. Conclusion
   - Limitations and suggested improvements
   - Products, possible future steps and summer experience
   - Acknowledgments

FFT based (also called "Fast Convolution")

Multi-stage approach

- Convolution in geometric space amounts to point-wise multiplication (modulation) in frequency space. Theorem on DSP: $f * g = FT_i(FT(f).FT(g))$
- We are concerned on Periodic and Discrete Fourier Transform and its inverse. The naive DFT takes $\theta(n^2)$.
- By the Colley-Turkey divide-and-conquer algorithm, we can compute it in $\theta(nlogn)$. The algorithm is called Fast Fourier Transform (FFT). There are many variations of Colley-Turkey algorithm, the chosen one was the radix-2 (easier implementation and better accuracy, according to [1]).

FFT based (also called "Fast Convolution")

Steps

- Convolution Steps
  - 2D FFT on image
  - 2D FFT on filter
  - Point-wise multiplication
  - 2D FFT_i of the result
- 2D FFT steps
  - 1D FFT over one dimension of the matrix (e.g. on each row)
  - Transpose resulting matrix
  - 1D FFT again over same dimension
  - Transpose back the matrix
- The transposition was done in order to improve locality and therefore better cache use.
- Features: Multi-stage computation on n-point kernels, Complex multiplication auto-tuned kernel (3-level blocking, variable reuse), multi-thread.

FFT based (also called "Fast Convolution")

Code

```
repeat
    -- #stages that type of kernel absorb is lower or equal lacking
    if ker[k][1] <= rest then
        K_W = ker[k][0]
        skernel = ker[k][1]
        base  = 0
        ublocks = NELEMS/K_W
        rest = rest - skernel
        for i=0,ublocks-1 do -- loop over big ublocks
            base = i*(Ns*K_W*2) -- iterate over the big ublocks, K_
            for j=0, Ns-1 do -- inside each block
                if k == 0 then
                    exec:insert(quote
                        FFT_4(base + 2*j,A,Ns,NFFT,signal)
                    end)
                elseif k == 1 then
                    exec:insert(quote
                        FFT_2(i*Ns+j,base + 2*j,A,Ns,NFFT,signal)
                    end)
                end
            end
        end
        -- jump stages  (min. skernel == 1 and K_W = 2)
        s = s + skernel
        Ns = Ns * K_W
        NELEMS = NELEMS / K_W
    else
        k = k + 1
    end
until rest == 0

bitreversal:insert(quote
    cbit.reversal(NFFT,A)
end)

return terra([A] : &double)
    [exec];
    [bitreversal];
end
```

Figure: Multi-staged kernel for FFT

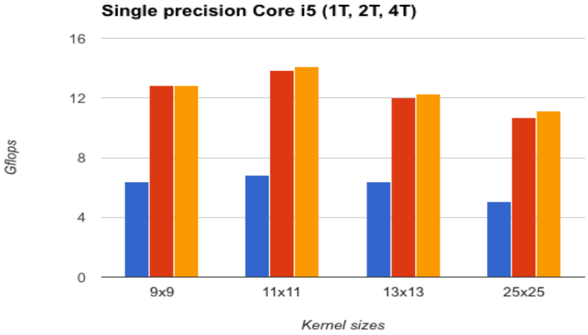

Figure: Gflops table for Core i5

Comparison with the state of the art
Lowering

	Speed up over CcT		
32x32	3(p1)	7(p3)	13(p6)
3	3.366901792	4.04963704	6.17078521
15	6.53631488	3.807272381	1.717765592
40	20.2459685	9.285587372	11.16347907
96	22.385713	9.570350035	3.779976059
256	18.21301395	5.476782931	2.117950723
128x128	3(p1)	7(p3)	13(p6)
3	15.73386751	12.28365235	10.21441127
15	20.44453337	8.065545275	6.260054264
40	17.89114576	10.20093719	7.622179521
96	9.47757057	7.099851183	4.272144582
256	13.76905301	8.033597631	4.220747433

Figure: Lowering vs Caffe con Troll. Considering only one image with depth = 1 (b=1) and stride = 1 (s=1).

Comparison with the state of the art
FFT based

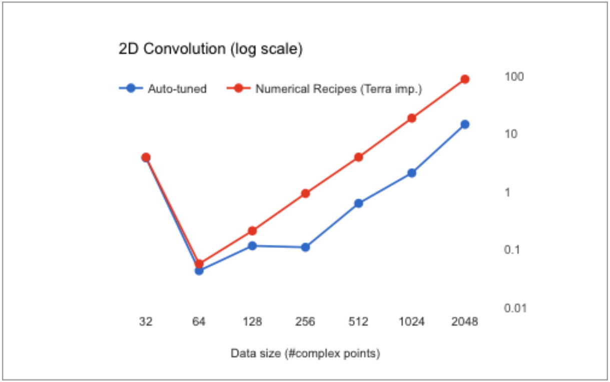

Figure: 2D FFT-based convolution

Outline

Comparison between approaches

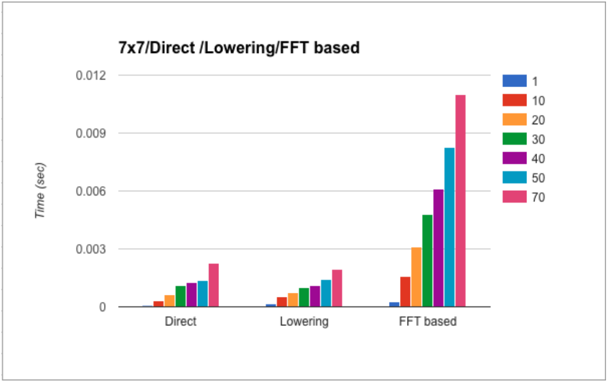

Figure: Small image (32x32) and small kernel (7x7). As expected Direct and Lowering better than FFT.

Comparison between approaches

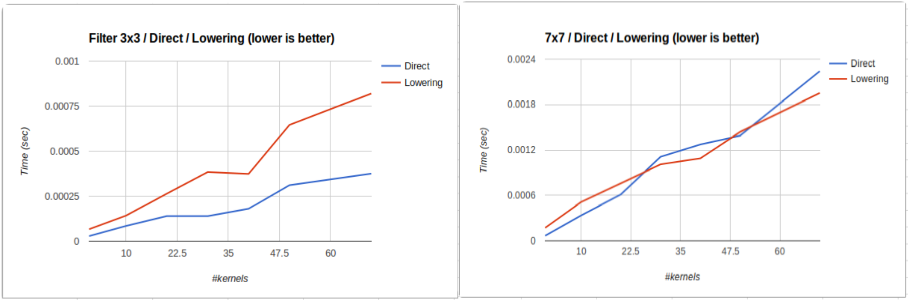

Figure: Direct vs Lowering

Outline

Limitations and suggested improvements

- The optimization was focused on one image. So, it will only be efficient for a small depth, e.g. a RGB image (depth $= 3$). For greater ones, more optimizations should be done.
- Direct: Re-think strategies for vector instruction, e.g. activate it only for some kernel sizes.
- Lowering: Multi-thread lowering, multi-thread and block transpose kernel. Maybe implement the fusion changing the Terra's GEMM kernel (it would be easy once GEMM kernel is in Terra also).
- FFT: CGEMM instead of the CMULT (such as fbfft [4]), twidle factors reuse, twidle factors during staging. Ovearlap-add and overlap-save as "blocking" methods for the small kernels.
- FFT has accuracy problems, one has to define an upper bound of this round-off error.

Outline

Product, possible future steps and summer experience

- Code and documentation on GitHub
- Possible future steps: Optimization for image batches. Provide the others CNN layers.

Outline

Acknowledgments

Summary

- An integrated multi-staged implementation of the three convolution approaches was done.
- Each one of the methods seems to be efficient compared with the state of the art (more tests are necessary for FFT).
- The implementation of the methods seems to be complementary considering the usual deep learning parameter space.

- Outlook
  - Apply possible optimization on patch of images
  - Necessary a faster FFT (base code is there, do more n-kernels and suggested optimizations)

For Further Reading I

Teukolsky, Vetterling and Flannery
The Art of Scientific Computing.
Cambridge University Press, 1992.

DeVito Z., Hegarty J., Aiken A., Hanrahan P. and Vitek J.
Terra: A Multi-Stage Language for High-Performance Computing
PLDI, 2013.

Hadjis S., Abuzaid F., Zhang C. and Ré C.
Caffe con Troll: Shallow Ideas to Speed Up Deep Learning
arXiv:1504.04343, 2015

Vasilache N., et al.
On Fast Convolutional Nets With fbfft: A GPU Performance Evaluation.
arXiv: 1412.7580, 2015.

For Further Reading II

Chetlur S., et al.
cuDNN: Efficient Primitives for Deep Learning
arXiv:1410.0759v3, 2014.

Li F. and Karpathy A.
Convolutional Neural Networks for Visual Recognition (CS231n)
Stanford University, 2014.

Iandola, F.N.; Sheffield, D.; Anderson, M.J.; Phothilimthana, P.M.; Keutzer, K.,
Communication-minimizing 2D Convolution In GPU Registers
International Conference on Image Processing (ICIP), 2013.

AMD Developer Center
OpenCL Optimization Case Study Fast Fourier Transform - Parts 1 and 2
AMD Articles, 2011.