Fast Algorithms for Convolutional Neural Networks

Andrew Lavin  
alavin@acm.org

Scott Gray  
Nervana Systems  
sgray@nervanasys.com

CVPR 2016
Shameless plug:
As a DNN researcher, how do I make sure that I’m using the fastest NN package?

GitHub, Inc. [US]  https://github.com/soumith/convnet-benchmarks

<table>
<thead>
<tr>
<th>Library</th>
<th>Class</th>
<th>Time (ms)</th>
<th>forward (ms)</th>
<th>backward (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CuDNN[R4]-fp16 (Torch)</td>
<td>cudnn.SpatialConvolution</td>
<td>71</td>
<td>25</td>
<td>46</td>
</tr>
<tr>
<td>Nervana-neon-fp16</td>
<td>ConvLayer</td>
<td>78</td>
<td>25</td>
<td>52</td>
</tr>
<tr>
<td>CuDNN[R4]-fp32 (Torch)</td>
<td>cudnn.SpatialConvolution</td>
<td>81</td>
<td>27</td>
<td>53</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>conv2d</td>
<td>81</td>
<td>26</td>
<td>55</td>
</tr>
<tr>
<td>Nervana-neon-fp32</td>
<td>ConvLayer</td>
<td>87</td>
<td>28</td>
<td>58</td>
</tr>
<tr>
<td>fbfft (Torch)</td>
<td>fbnn.SpatialConvolution</td>
<td>104</td>
<td>31</td>
<td>72</td>
</tr>
<tr>
<td>Chainer</td>
<td>Convolution2D</td>
<td>177</td>
<td>40</td>
<td>136</td>
</tr>
<tr>
<td>cudaconvnet2*</td>
<td>ConvLayer</td>
<td>177</td>
<td>42</td>
<td>135</td>
</tr>
<tr>
<td>CuDNN[R2] *</td>
<td>cudnn.SpatialConvolution</td>
<td>231</td>
<td>70</td>
<td>161</td>
</tr>
<tr>
<td>Caffe (native)</td>
<td>ConvolutionLayer</td>
<td>324</td>
<td>121</td>
<td>203</td>
</tr>
<tr>
<td>Torch-7 (native)</td>
<td>SpatialConvolutionMM</td>
<td>342</td>
<td>132</td>
<td>210</td>
</tr>
<tr>
<td>CL-nn (Torch)</td>
<td>SpatialConvolutionMM</td>
<td>963</td>
<td>388</td>
<td>574</td>
</tr>
<tr>
<td>Caffe-CLGreenTea</td>
<td>ConvolutionLayer</td>
<td>1442</td>
<td>210</td>
<td>1232</td>
</tr>
</tbody>
</table>
This paper is about…

reducing the number of floating point operations when computing a convolution.
Convolution Theorem
Convolution Theorem

- **Definition:** Convolution in the time domain is equivalent to pointwise multiply in the frequency domain.

\[
f \ast g = \mathcal{F}^{-1} \{ \mathcal{F}\{f\} \cdot \mathcal{F}\{g\} \} \]

\(\mathcal{F}\{f\}\) and \(\mathcal{F}\{g\}\) are the Fourier transforms of \(f\) and \(g\).

The asterisk denotes convolution, not multiplication.

- **Example:**

\[
f = \begin{bmatrix} 0 & 0 & 1 & 2 \end{bmatrix}
\]

\[
g = \begin{bmatrix} 10 & 20 & 30 \end{bmatrix}
\]

\[
f \ast g = \begin{bmatrix} 30 & 80 \end{bmatrix}
\]
Dot Product Approach

\[ f = \begin{bmatrix} 0 & 0 & 1 & 2 \end{bmatrix} \]
\[ g = \begin{bmatrix} 10 & 20 & 30 \end{bmatrix} \]
\[ f \ast g = \begin{bmatrix} 30 & 80 \end{bmatrix} \]

- 6 Multiplications
- 4 Additions

Convolutions Theorem Approach

\[
\begin{align*}
\text{out}[0] &= 0 \times 10 + 0 \times 20 + 1 \times 20 = 30 \\
\text{out}[1] &= 0 \times 10 + 1 \times 20 + 2 \times 20 = 80 \\
\end{align*}
\]

- 6 Multiplications
- + more operations for FFT/IFFT, ... what gives?

\[
\begin{align*}
>>> f_0 &= [0,0,1] \\
>>> f_1 &= [0,1,2] \\
>>> g &= [10,20,30] \\
>>> gg &= [30,20,10] \\
>>> \text{ifft(fft(f_0)*fft(gg))} \\
array([20.+0.j, 10.+0.j, 30.+0.j]) \\
>>> \text{ifft(fft(f_1)*fft(gg))} \\
array([50.+0.j, 50.+0.j, 80.+0.j]) \\
\end{align*}
\]
Shmuel Winograd (Winograd FFTs)

\[ F(2, 3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix} \]

(5)

where

\[ m_1 = (d_0 - d_2)g_0 \]
\[ m_2 = (d_1 + d_2) \frac{g_0 + g_1 + g_2}{2} \]
\[ m_4 = (d_1 - d_3)g_2 \]
\[ m_3 = (d_2 - d_1) \frac{g_0 - g_1 + g_2}{2} \]

- Data (d's): 4 ADDs
- Filter (g's): 3 ADDs, 2 MULs
- Outputs (m's): 4 MULs, 4 ADDs
4. Fast Algorithms

It has been known since at least 1980 that the minimal filtering algorithm for computing $m$ outputs with an $r$-tap FIR filter, which we call $F(m, r)$, requires

$$\mu(F(m, r)) = m + r - 1$$  \hspace{1cm} (3)
Nesting Minimal Filtering Algorithms

A minimal 1D algorithm \( F(m, r) \) is nested with itself to obtain a minimal 2D algorithm, \( F(m \times m, r \times r) \) like so:

\[
Y = A^T \left[ (GgG^T) \odot (B^T dB) \right] A
\]

(8)

where now \( g \) is an \( r \times r \) filter and \( d \) is an \((m + r - 1) \times (m + r - 1)\) image tile. The nesting technique can be generalized for non-square filters and outputs, \( F(m \times n, r \times s) \), by nesting an algorithm for \( F(m, r) \) with an algorithm for \( F(n, s) \).

\( F(2 \times 2, 3 \times 3) \) uses \( 4 \times 4 = 16 \) multiplications, whereas the standard algorithm uses \( 2 \times 2 \times 3 \times 3 = 36 \). This is an arithmetic complexity reduction of \( \frac{36}{16} = 2.25 \). The data transform uses 32 additions, the filter transform uses 28 floating point instructions, and the inverse transform uses 24 additions.

\[
I_{\text{padded}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 
\end{bmatrix}
\]

\[ F(2 \times 2, 3 \times 3) \]

\[
\text{out} = \begin{bmatrix}
o_0 & o_1 \\
o_2 & o_3 
\end{bmatrix} \quad \text{filter} = \begin{bmatrix}
1 & 2 & 1 \\
1 & 2 & 1 \\
1 & 2 & 1 
\end{bmatrix}
\]

\[ \text{input size} = (m+r-1) \times (m+r-1) = 4 \times 4 \]
You saved multiplications by a factor of 2.25. So what?
[1412.7580] Fast Convolutional Nets With fbfft: A GPU Performance ...

by N Vasilache - 2014 - Cited by 45 - Related articles

Dec 24, 2014 - ... another based on a Facebook authored FFT implementation, fbfft, that provides significant speedups over cuFFT (over 1.5x) for whole CNNs.

[1312.5851] Fast Training of Convolutional Networks through FFTs

by M Mathieu - 2013 - Cited by 57 - Related articles

Dec 20, 2013 - Training a large convolutional network to produce state-of-the-art results can take weeks, even when using modern GPUs. Producing labels ...
Figure 1: 3 × 3 kernel (K40m)

Figure 2: 5 × 5 kernel (K40m)

Figure 3: 7 × 7 kernel (K40m)

Figure 4: 9 × 9 kernel (K40m)

Figure 5: 11 × 11 kernel (K40m)

Figure 6: 13 × 13 kernel (K40m)
Figure 2: Number of operations required for computing (1) with different input image sizes and $S = 128$, $f = 96$, $f' = 256$, $k = 7$. 
\[ f \ast g = F^{-1} \{ F\{f\} \cdot F\{g\} \} \]

Thus we can reduce over \( C \) channels in transform space, and only then apply the inverse transform \( A \) to the sum. This amortizes the cost of the inverse transform over the number of channels.

The Fourier transform of a real signal has Hermitian symmetry, which reduces the number of unique products in each \( U \odot V \) by almost half. FFT based convnet implementations benefit from this fact. The magnitude of the transform matrix elements also increases with increasing tile size. This effectively reduces the numeric accuracy of the computation, so that for large
### Evaluation

<table>
<thead>
<tr>
<th>N</th>
<th>cuDNN msec</th>
<th>TFLOPS</th>
<th>F(2x2,3x3) msec</th>
<th>TFLOPS</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12.52</td>
<td>3.12</td>
<td>5.55</td>
<td>7.03</td>
<td>2.26X</td>
</tr>
<tr>
<td>2</td>
<td>20.36</td>
<td>3.83</td>
<td>9.89</td>
<td>7.89</td>
<td>2.06X</td>
</tr>
<tr>
<td>4</td>
<td>104.70</td>
<td>1.49</td>
<td>17.72</td>
<td>8.81</td>
<td>5.91X</td>
</tr>
<tr>
<td>8</td>
<td>241.21</td>
<td>1.29</td>
<td>33.11</td>
<td>9.43</td>
<td>7.28X</td>
</tr>
<tr>
<td>16</td>
<td>203.09</td>
<td>3.07</td>
<td>65.79</td>
<td>9.49</td>
<td>3.09X</td>
</tr>
<tr>
<td>32</td>
<td>237.05</td>
<td>5.27</td>
<td>132.36</td>
<td>9.43</td>
<td>1.79X</td>
</tr>
<tr>
<td>64</td>
<td>394.05</td>
<td>6.34</td>
<td>266.48</td>
<td>9.37</td>
<td>1.48X</td>
</tr>
</tbody>
</table>

Table 5. cuDNN versus $F(2 \times 2, 3 \times 3)$ performance on VGG Network E with fp32 data. Throughput is measured in Effective TFLOPS, the ratio of direct algorithm GFLOPs to run time.