Fast Convolution Algorithms for deep learning and computer vision

Sample slides only

Presenter: Prof. Ioannis Pitas
Aristotle University of Thessaloniki
pitas@csd.auth.gr
Outline

• 1D convolutions
  Linear & Cyclic 1D convolutions
  Discrete Fourier Transform, Fast Fourier Transform
  Winograd algorithm

• Linear & Cyclic 2D convolutions

• Applications in deep learning
  Convolutional neural networks
Motivation

• Fast implementation of 1D and 2D digital filters
  Image filtering
  Image feature calculation
    • Gabor filters
• Fast implementation of 1D and 2D correlation
  Template matching
  Correlation tracking
• Machine learning
  Convolutional Neural Networks
Linear 1D convolution

• The one-dimensional (linear) convolution of:
  • an input signal $x$ and
  • a convolution kernel $h$ (filter finite impulse response) of length $N$:

$$y(k) = h(k) * x(k) = \sum_{i=0}^{N-1} h(i)x(k - i)$$

• For a convolution kernel centered around 0 and $N = 2v + 1$, it takes the form:

$$y(k) = h(k) * x(k) = \sum_{i=-v}^{v} h(i)x(k - i)$$
Linear 1D convolution - Example

Linear 1D convolution - Example

Linear 1D correlation

- Correlation of template $h$ and input signal $x(k)$:

$$r(k) = \sum_{i=0}^{N-1} h(i)x(k + i)$$

- Input signal is not flipped.
- It is used for template matching and for object tracking in video.
- It is often confused with convolution: they are identical only if $h$ is centered at and is symmetric about $i=0$. 

Cyclic 1D convolution

- One-dimensional cyclic convolution of length $N$, $(k)_N = k \mod N$:
  
  $$y(k) = x(k) \ast h(k) = \sum_{i=0}^{N-1} h(i)x((k - i)_N)$$

- Embedding linear convolution in a cyclic convolution $y(n) = x(n) \otimes h(n)$ of length $N \geq L + M - 1$ and then performing a cyclic convolution of length $N$:
  
  $$y(k) = x(k) \ast h(k) = \sum_{i=0}^{N-1} x_N(i) h_n((k - i)_N)$$
Cyclic Convolution via DFT

Cyclic convolution can also be calculated using 1D DFT:

\[ y = IDFT(DFT(x)DFT(h)) \]
1D FFT

• There are a few algorithms to speed up the calculation of DFT.
• The most well known is the **radix-2** decimation-in-time (DIT) Fast Fourier Transform (FFT) (Cooley-Tuckey).

1. The DFT of a sequence $x(n)$ of length $N$ is:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{2\pi i}{N} nk}$$

where $k$ is an integer ranging from 0 to $N - 1$. 
1D FFT

- radix-2 FFT breaks a length-$N$ DFT into many size-2 DFTs called "butterfly" operations.
- There are $\log_2 N$ stages.
The Z-transform of a signal (function) $x(n)$ having domain $[0, ..., N]$ is given by:

$$X(z) = \sum_{n=0}^{N-1} x(n)z^{-n}$$

The domain of Z-transform is the complex plane, since $z$ is a complex number. The following relation holds for the Z-transform:

$$y(n) = x(n) * h(n) \iff Y(z) = X(z)H(z)$$
Cyclic convolution and Z-transform

\[ y(k) = x(k) \ast h(k) = \sum_{i=0}^{N-1} h(i) x((k - i)_N) \]

Where: \((k)_N = k \mod N\)

\[ y(n) = x(n) \ast h(n) \iff Y(z) = X(z)H(z) \mod(z^N - 1) \]
Winograd algorithm

Fast 1D cyclic convolution with minimal complexity

• The Winograd algorithm works on small tiles of the input image.
• The input tile and filter are transformed.
• The outputs of the transform are multiplied together in an element-wise fashion.
• The result is transformed back to obtain the outputs of the convolution.
Winograd algorithm
Fast 1D cyclic convolution with minimal complexity

• Winograd convolution algorithms or fast filtering algorithms:
  \[ Y = C(Ax \otimes Bh) \]

• They require only \( 2N - \nu \) multiplications in their middle vector product, thus having minimal complexity.

• \( \nu \): number of cyclotomic polynomial factors of polynomial \( z^N - 1 \) over the rational numbers \( Q \).

• GEneral Matrix Multiplication (GEMM) BLAS or CUBLAS routines can be used.
Linear and cyclic 2D convolutions

- Two-dimensional linear convolution with convolutional kernel $h$ of size $N_1 \times N_2$ is given by:

$$y(k_1, k_2) = h(k_1, k_2) \ast \ast x(k_1, k_2) = \sum_{i_1}^{N_1} \sum_{i_2}^{N_2} h(i_1, i_2)x(k_1 - i_1, k_2 - i_2)$$

- Its two-dimensional cyclic convolution counterpart of support $N_1 \times N_2$ is defined as:

$$y(k_1, k_2) = h(k_1, k_2) \ast \ast x(k_1, k_2) = \sum_{i_1}^{N_1} \sum_{i_2}^{N_2} h(i_1, i_2)x((k_1 - i_1)_{N_1}, (k_2 - i_2)_{N_2})$$
2D Convolution - Example

• With Padding
Applications

• Convolutional neural networks

• Signal processing
  Signal filtering
  Signal restoration
  Signal deconvolution

• Signal analysis
  Time delay estimation
  Distance calculation (e.g., sonar)
  1D template matching
Convolutional Neural Networks

Convergence of machine learning and signal processing processing

- Two step architecture:
  - First layers with sparse NN connections: convolutions.
  - Fully connected final layers.
- Need for fast convolution calculations.
Convolutional Layer

For RGB images

- For a convolutional layer $l$ with an activation function $f_l(\cdot)$, multiple incoming features $d_{in}$ and one single output feature $o$.

## Multiple input features to single feature $o$ transformation

$$y^{(l)}(i, j, o) = f_l \left( b^{(l)} + \sum_{r=1}^{d_{in}} \sum_{k_1=-q_1}^{q_1} \sum_{k_2=-q_2}^{q_2} w^{(l)}(k_1, k_2, r, o) x^{(l)}(i - k_1, j - k_2, r) \right)$$

## Convolutional Layer Activation Volume (3D tensor)

$$a^{(l)}_{ij}(o) = f_l \left( b^{(l)}(o) + \sum_{r=1}^{d_{in}} W^{(l)}(r, o) \ast X^{(l)}_{ij}(r) \right) \quad A^{(l)} = [a^{(l)}_{ij}(o): i = 1, \ldots, n^{(l)}, j = 1, \ldots, m^{(l)}, o = 1, \ldots, d_{out}]$$

where $A^{(l)}$ is the activation volume for the convolutional layer $l$, $W^{(l)}(r, o)$ is a 2D slice of the convolutional kernel $W^{(l)} \in \mathbb{R}^{h_1 \times h_2 \times d_{in} \times d_{out}}$ for input feature $r$ and output feature $o$, $b^{(l)}(o)$ a scalar bias and $X^{(l)}_{ij}(r)$ a region of input feature $r$ centered at $[i, j]^T$, e.g. $X^{(1)}(1)$ the R channel of an image $d_{in} = C = 3$. 
# Deep Learning Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>User Interface</th>
<th>Data Parallelism</th>
<th>Model Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td>protobuf, C++, Python</td>
<td>Yes</td>
<td>Limited</td>
</tr>
<tr>
<td>CNTK</td>
<td>BrainScript, C++, C#</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Python, C++</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Theano</td>
<td>Python</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Torch</td>
<td>LuaJIT</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Image Source: Heehoon Kim, Hyoungwook Nam, Wookeun Jung, and Jaejin Le - Performance Analysis of CNN Frameworks for GPUs
Deep Learning Frameworks

- All 5 frameworks work with cuDNN as backend.
- cuDNN unfortunately not open source
- cuDNN supports FFT and Winograd

<table>
<thead>
<tr>
<th>Framework</th>
<th>User Selectable</th>
<th>Heuristic-based</th>
<th>Profile-based</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Heuristic-based</td>
</tr>
<tr>
<td>CNTK</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Profile-based</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Heuristic-based†</td>
</tr>
<tr>
<td>Theano</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>GEMM</td>
</tr>
<tr>
<td>Torch</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>GEMM</td>
</tr>
</tbody>
</table>

†TensorFlow uses its own heuristic algorithm

Image Source: Heehoon Kim, Hyoungwook Nam, Wookeun Jung, and Jaejin Le - Performance Analysis of CNN Frameworks for GPUs
The Neon story

- Developed by Nervana in 2015
- Written in Python and C
- Doesn’t support Windows
- Uses MKL for CPU (highly optimized by Intel)
- Supports CUDA for GPU
- Known mostly to be the first to implement Winograd faster than others.
Thank you very much for your attention!

Contact: Prof. I. Pitas
pitas@csd.auth.gr
www.multidrone.eu