Fast Algorithms for Convolutional Neural Networks

Keshav Vinayak Jha
M.Tech.(Research)
CSA, IISc

Index

Introduction

- Deep Convolutional Neural Networks (ConvNets) achieve state of the art results on image recognition problems but take several days of GPU time to train.
- State-of-the-art ConvNet architectures for image recognition user deep networks of 3x3 convolutional layers.
- Hence, there is a need for fast ConvNet algorithms for small filters and batch sizes.
- This paper introduces fast algorithms for CNNs based on the minimal filter algorithms proposed by Winograd[1].
  - The memory requirements are also lighter than conventional FFTs.
  - Achieves state-of-the-art throughput for all batch sizes measured(1-64), while using at most 16MB of workspace memory.

Background: Convolutional Neural Networks

- Correlates a bank of K filters with C channels and R x S (K, C, R, S) against a minibatch of N images with C channels and size H x W.

$$Y_{i,k,x,y} = \sum_{c=1}^{C} \sum_{v=1}^{R} \sum_{u=1}^{S} D_{i,c,x+u,y+v} G_{k,c,u,v} \qquad (1)$$

$$Y_{i,k} = \sum_{c=1}^{C} D_{i,c} * G_{k,c} \qquad (2)$$

Background: FIR Filters

- The minimal filtering algorithm for computing **m** outputs with an **r-tap FIR filter** is denoted by: **F(m,**

$$\mu(F(m,r)) = m + r - 1$$

# of minimum multiplications required

- Can nest two 1D algorithms F(m, r) and F(n, s)**[2]** such that

$$\begin{aligned}\mu(F(m \times n, r \times s)) &= \mu(F(m,r))\mu(F(n,s)) \\ &= (m + r - 1)(n + s - 1)\end{aligned}$$

- The minimal filtering algorithm requires 1 multiplication per input.

Fast Algorithms: F(2x2, 3x3)

- Standard algorithm for F(2, 3) would use 2x3 = 6 multiplications.
- Winograd's minimal algorithm:

$$F(2,3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix} \tag{5}$$

Where

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

Can be expressed as :

$$Y = A^T\left[(Gg) \odot (B^T d)\right]$$

Where

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

$$g = \begin{bmatrix} g_0 & g_1 & g_2 \end{bmatrix}^T$$

$$d = \begin{bmatrix} d_0 & d_1 & d_2 & d_3 \end{bmatrix}^T$$

Fast Algorithms: F(2x2, 3x3)

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

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

- g is now an r x r filter and d is an (m+r-1) x (m+r-1) image tile.
- This nesting technique can be generalized for non-square filters and outputs, F(m x n, r x s), by nesting an algorithm for F(m, r) with an algorithm of F(n x s).
- F(2 x 2, 3 x 3) uses 16 multiplications, but the standard multiplication would use 2x2x3x3 = 36.
- Algorithmic complexity reduction by a factor of **2.25**.

Fast Algorithms: F(2x2, 3x3)

- Algorithms for F(m x m, r x r) can be used to compute ConvNet layers with r x r kernels.
- Each image channel is divided into tiles of sizes (m+r-1, m+r-1) with r-1 elements of overlap between neighbouring tiles, yielding ***[H/m][W/m]*** tiles per channel ***C***.
- *F(mxm, rxr)* is then computed for each tile and filter combination in each channel, and the results are summed over all channels.
- Substituting $U = GgG^T$ and $V = B^T dB$

$$Y = A^T[U \odot V]A$$

$$\begin{aligned} Y_{i,k,\tilde{x},\tilde{y}} &= \sum_{c=1}^{C} D_{i,c,\tilde{x},\tilde{y}} * G_{k,c} \\ &= \sum_{c=1}^{C} A^T \left[U_{k,c} \odot V_{c,i,\tilde{x},\tilde{y}} \right] A \\ &= A^T \left[\sum_{c=1}^{C} U_{k,c} \odot V_{c,i,\tilde{x},\tilde{y}} \right] A \end{aligned}$$

$$M_{k,i,\tilde{x},\tilde{y}} = \sum_{c=1}^{C} U_{k,c} \odot V_{c,i,\tilde{x},\tilde{y}}$$

$$M_{k,b}^{(\xi,\nu)} = \sum_{c=1}^{C} U_{k,c}^{(\xi,\nu)} V_{c,b}^{(\xi,\nu)}$$

$$M^{(\xi,\nu)} = U^{(\xi,\nu)} V^{(\xi,\nu)}$$

Fast Algorithms: F(2x2, 3x3)

Algorithm 1 Compute Convnet Layer with Winograd Minimal Filtering Algorithm $F(m \times m, r \times r)$

$P = N\lceil H/m \rceil \lceil W/m \rceil$ is the number of image tiles.
$\alpha = m + r - 1$ is the input tile size.
Neighboring tiles overlap by $r - 1$.
$d_{c,b} \in \mathbb{R}^{\alpha \times \alpha}$ is input tile b in channel c.
$g_{k,c} \in \mathbb{R}^{r \times r}$ is filter k in channel c.
G, B^T, and A^T are filter, data, and inverse transforms.
$Y_{k,b} \in \mathbb{R}^{m \times m}$ is output tile b in filter k.
for $k = 0$ to K **do**
  for $c = 0$ to C **do**
    $u = G g_{k,c} G^T \in \mathbb{R}^{\alpha \times \alpha}$
    Scatter u to matrices U: $U_{k,c}^{(\xi,\nu)} = u_{\xi,\nu}$
for $b = 0$ to P **do**
  for $c = 0$ to C **do**
    $v = B^T d_{c,b} B \in \mathbb{R}^{\alpha \times \alpha}$
    Scatter v to matrices V: $V_{c,b}^{(\xi,\nu)} = v_{\xi,\nu}$
for $\xi = 0$ to α **do**
  for $\nu = 0$ to α **do**
    $M^{(\xi,\nu)} = U^{(\xi,\nu)} V^{(\xi,\nu)}$
for $k = 0$ to K **do**
  for $b = 0$ to P **do**
    Gather m from matrices M: $m_{\xi,\nu} = M_{k,b}^{(\xi,\nu)}$
    $Y_{k,b} = A^T m A$

Fast Algorithms: *F(3x3, 2x2)*

- Training a network using stochastic gradient descent requires the computation of the gradient with respect to the inputs and weights.
- The gradient with respect to the inputs is a convolution of the next layers' backpropagated error, of dimension N x K x H x W, with a flipped version of the layer's R x S filters.
- Can be computed using the same algorithm that is used for forward propagation.

Fast Algorithms: *F(3x3, 2x2)*

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}, G = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

- Results in 36/16 = 2.25 arithmetic complexity reduction, as the corresponding forward propagation algorithm.

Fast Algorithms: *F(4x4, 3x3)*

- For *F(4, 3):*

$$B^T = \begin{bmatrix} 4 & 0 & -5 & 0 & 1 & 0 \\ 0 & -4 & -4 & 1 & 1 & 0 \\ 0 & 4 & -4 & -1 & 1 & 0 \\ 0 & -2 & -1 & 2 & 1 & 0 \\ 0 & 2 & -1 & -2 & 1 & 0 \\ 0 & 4 & 0 & -5 & 0 & 1 \end{bmatrix}$$

$$G = \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ -\frac{1}{6} & -\frac{1}{6} & -\frac{1}{6} \\ -\frac{1}{6} & \frac{1}{6} & -\frac{1}{6} \\ \frac{1}{24} & \frac{1}{12} & \frac{1}{6} \\ \frac{1}{24} & -\frac{1}{12} & \frac{1}{6} \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & 0 \\ 0 & 1 & 1 & 4 & 4 & 0 \\ 0 & 1 & -1 & 8 & -8 & 1 \end{bmatrix}$$

- Applying the nesting formula yields a minimal algorithm for *F(4x4, 3x3)* that uses 6x6 = 36 multiplies, compared to the standard 4x4x3x3 = 144 multiplies.
- The number of additions and constant multiplications required by the minimal Winograd transforms increases quadratically with the tile size.
- Thus, for large tiles, the complexity of the transforms will overwhelm any savings in the number of multiplications.
- Also, the magnitude of the transform matrix elements increases with increasing the tile size
- ConvNets require surprisingly little numeric precision

Fast Algorithms: **Fast Fourier Transform**

- FFT can be used to produce a tiled Conv algorithm just like Algorithm 1.
- Transform matrices are replaced by FFT and inverse FFT, and pointwise multiplication of complex FFT components yields cyclic convolution.
- Out of (m + r -1)x(n + s -1), mxn components are valid, rest must be discarded and recomputed: tiles are overlapped by r-1 and s-1.
- 1 multiply per input, but the operands are complex numbers; resulting in 4 real multiplications.
- The Fourier Transform of a real signal has Hermitian symmetry. Which reduced the number of unique products in each U*V by half.

Fast Algorithms: **Fast Fourier Transform**

- Using the standard algorithm for multiplying complex numbers, this equals $4\left(\text{floor}\left(\frac{\alpha}{2}\right)+1\right) > 2$ real multiplies per input.

- Another technique in addition, used for multiplying complex numbers is:

$$\begin{aligned}(x_0 + ix_1)(y_0 + iy_1) &= [x_0y_0 - x_1y_1, i(x_0y_1 + x_1y_0)] \\ &= [u_cv_a + u_av_c, i(u_av_c - u_bv_b)]\end{aligned} \tag{16}$$

where

$$\begin{aligned}u_a &= x_0 \qquad v_a = y_0 \\ u_b &= x_0 + x_1, v_b = y_1 \\ u_c &= x_1 - x_0 \quad v_c = y_0 + y_1\end{aligned} \tag{17}$$

- An FFT based ConvNet incorporates this by modifying the FFT transforms of the filter and data to output (U_a, U_b, U_c) and (V_a, V_b, V_c) instead of the complex valued matrices U and V.

Fast Algorithms: **Fast Fourier Transform**

- Then we can calculated M=UV using 3 calls to SGEMM

$$T = U_a V_c \qquad M_0 = U_c V_a + T$$
$$M_1 = -U_b V_b + T,\ M = (M_0, iM_1)$$

- The temporary matrix T increases memory use by half, so the workspace size is approx. twice that of FFT based convolution with direct CGEMM.

- Combining Hermitian symmetry with fast CGEMM gives $3\left(\text{floor}\left(\frac{\alpha}{2}\right)+1\right) > 1.5$ real multiples per input.
- Still higher than Winograd, so FFT must use a significantly larger tile size in order to rival the arithmetic complexity of the Winograd algorithms.

Arithmetic Complexity Analysis

- The arithmetic complexity of the multiplication st: $M = N\lceil H/m \rceil \lceil W/n \rceil CK(m+R-1)(n+S-1)$
- When m=n-1, it becomes direct convolution: *F(1x1, RxS)*
- For simplification, W/m and H/n have no remainders, R=S, m=n (Square filters and blocks)

$$M = (m+R-1)^2/m^2 NHWCK$$
$$= \alpha' NHWCK$$

where $\alpha = (m+R-1)^2$ and $\alpha' = \alpha/m^2$

- The total arithmetic complexities of the data, filter and inverse transforms can be written as:

$$T(D) = \beta/m^2 NHWC$$
$$T(F) = \gamma CK$$
$$T(I) = \delta/m^2 NHWK$$

Where β, γ, δ are the number of floating-point instructions used by the corresponding transforms for single tiles

Dividing the complexity of each transform by M yields it's relative complexity

$$T(D)/M = \beta/(K\alpha^2) = \beta'/K$$
$$T(F)/M = \gamma/(NHW\alpha^2/m^2)$$
$$= \gamma/(P\alpha^2) = \gamma'/P$$
$$T(I)/M = \delta/(C\alpha^2) = \delta'/C$$

Arithmetic Complexity Analysis

$$T(D) = \beta/m^2 NHWC$$
$$T(F) = \gamma CK$$
$$T(I) = \delta/m^2 NHWK$$

Where β, γ, δ are the number of floating-point instructions used by the corresponding transforms for single tiles

Dividing the complexity of each transform by M yields it's relative complexity

$$T(D)/M = \beta/(K\alpha^2) = \beta'/K$$
$$T(F)/M = \gamma/(NHW\alpha^2/m^2)$$
$$= \gamma/(P\alpha^2) = \gamma'/P$$
$$T(I)/M = \delta/(C\alpha^2) = \delta'/C$$

- Here β', γ', δ' are the normalized arithmetic complexities of the data, filter and the inverse transforms respectively.
- $P = \frac{NHW}{m^2}$ is the number of tiles per channel.
- Adding the terms of each stage gives the total arithmetic complexity of the convnet layers.
- For direct convolution, $\alpha' = \alpha^2 = R^2 \; and \; \beta' = \gamma' = \delta' = 0$. Therefore the maximum speedup of a fast algorithm versus direct convolution is $\frac{R^2}{\alpha'}$

Arithmetic Complexity Analysis

Tile	Winograd				FFT	
	α'	β'	γ'	δ'	α'	β', γ', δ'
3	9.00	-	-	-		
4	4.00	2.00	1.75	1.50		
5	2.78	3.60	2.24	2.24		
6	**2.25**	**4.33**	**2.00**	**2.78**		
8	1.78	6.50	2.23	4.38	4.44	2.42
16					2.94	4.23
32					2.42	6.24
64					**2.20**	**8.30**
128					2.10	10.37
256					2.05	12.42

Table 1. Multiply (α'), data transform (β'), filter transform (γ'), and inverse transform (δ') normalized arithmetic complexity versus tile size, for both Winograd and FFT based convolution. F(4x4,3x3) has tile size 6. Direct convolutions has tile size 3.

Tile	FFT with Fast CGEMM			
	α'	β'	γ'	δ'
8	3.33	3.77	4.30	4.30
16	**2.20**	**6.23**	**6.82**	**6.82**
32	1.81	8.94	9.57	9.57
64	1.65	11.72	12.36	12.36
128	1.57	14.48	15.14	15.14
256	1.54	17.22	17.88	17.88

Table 2. Normalized arithmetic complexity for FFT filtering with fast CGEMM. Fast CGEMM uses 3 multiplies per complex multiply instead of 4, but has slightly greater transform overhead and uses more memory.

- Here β', γ', δ' are the normalized arithmetic complexities of the data, filter and the inverse transforms respectively.
- $P = \frac{NHW}{m^2}$ is the number of tiles per channel.
- Adding the terms of each stage gives the total arithmetic complexity of the convnet layers.
- For direct convolution, $\alpha' = \alpha^2 = R^2 \; and \; \beta' = \gamma' = \delta' = 0$. Therefore the maximum speedup of a fast algorithm versus direct convolution is $\frac{R^2}{\alpha'}$
- The normalized transform complexities are listed for different tile sizes and algorithms in table 1 and 2.

Arithmetic Complexity Analysis

Tile	Winograd				FFT	
	α'	β'	γ'	δ'	α'	β', γ', δ'
3	9.00	-	-	-		
4	4.00	2.00	1.75	1.50		
5	2.78	3.60	2.24	2.24		
6	**2.25**	**4.33**	**2.00**	**2.78**		
8	1.78	6.50	2.23	4.38	4.44	2.42
16					2.94	4.23
32					2.42	6.24
64					**2.20**	**8.30**
128					2.10	10.37
256					2.05	12.42

Table 1. Multiply (α'), data transform (β'), filter transform (γ'), and inverse transform (δ') normalized arithmetic complexity versus tile size, for both Winograd and FFT based convolution. F(4x4,3x3) has tile size 6. Direct convolutions has tile size 3.

Tile	FFT with Fast CGEMM			
	α'	β'	γ'	δ'
8	3.33	3.77	4.30	4.30
16	**2.20**	**6.23**	**6.82**	**6.82**
32	1.81	8.94	9.57	9.57
64	1.65	11.72	12.36	12.36
128	1.57	14.48	15.14	15.14
256	1.54	17.22	17.88	17.88

Table 2. Normalized arithmetic complexity for FFT filtering with fast CGEMM. Fast CGEMM uses 3 multiplies per complex multiply instead of 4, but has slightly greater transform overhead and uses more memory.

- Even with fast CGEMM, the larger tile size compared to Winograd means FFT-based ConvNet implementations must have a large memory workspace to hold transformed data to amortize transform cost and to generate matrices with large enough dimensions so that the multiply stage is efficient.
- This is problematic for current GPUs, which have a limited amount of on-chip memory, CPUs have larger caches and might compute FFT based convolution more efficiently.

GPU Implementation

- Implemented *F(2x2, 3x3)* for NVIDIA Maxwell GPUs and tested on the NVIDIA Titan X model.
- *F(2x2, 3x3)* enables fused implementation of algorithm stage, data and filter transform 16 batched GEMMs, and inverse transforms in the same block.
- The 16 batched GEMMs compute *32x32* outputs, which enables to fit the workspace in the registers and shared mem of a single block and still have *multiple* active blocks per SM for latency hiding.
- Image Data is stored as *(C, H, W, N)* to facilitate contiguous and aligned memory loads, reducing over-fetch.
- 32 tiles of *4x4* are loaded from a configurable number of images, N >= 32 implies loading 32 tiles from different images, whereas for N < 32, a superblock of *X * Y = 32/N* tiles per image are loaded.
- This strategy facilitates efficient loads with small batch sizes, as *W x N* dimensions of the input data are contiguous in memory.
- Furthermore, the 2-pixel overlap between adjacent tiles causes high hits in the L1 cache hit rates when using several tiles in a super block.

GPU Implementation

- Since the number of image tiles is generally larger than the number of filters, the block mapping iterates over a group of up to 128 filters in the innermost loop and then iterates over all image tiles in the second loop.
- All channels of the filter group fit in the L2 cache, so each filter will only be loaded from DDR memory once, and each image tile will be loaded *ceil(K/128)* times.
- One version of the kernel loads fp16 data, and another version, called “FX” runs the transform kernel first and stores the result in a workspace buffer.
- The convolution kernel loads transformed filter from the workspace as needed, the size of the workspace is only 16MB when K=C=512, and the data is fp32.

Experiments and Results

- Refer to the last couple pages of the paper.

Thanks!

References:

- [1] Shmuel Winograd. Arithmetic complexity of computations, volume 33. Siam, 1980
- [2] Shmuel Winograd. On multiplication of polynomials modulo a polynomial. SIAM Journal on Computing, 9(2):225–229, 1980.
- [3] cuDNN. https://developer.nvidia.com/cudnn. Accessed: 2015-11-01.
- [4] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. CoRR, abs/1312.5851, 2013.