Fast Algorithms for Convolutional Neural Networks

- Group 2

1. Introduction

Convolutional Neural Networks are most commonly employed in Computer Vision and Natural Language Processing. Graphic Processing Units (GPUs) are used to accelerate Convolution Neural Networks (CNNs) due to their cheaper prices and greater availability. We frequently utilize more complicated architectures and bigger volumes of training data to get higher accuracies. Because GPUs frequently have limited physical memory, training larger CNNs is more challenging.

To lower the memory footprint of CNNs, memory optimization is essential. Memory optimization methods now in use have a negative impact on performance. Memory optimization approaches must be designed in such a way that they not only minimize memory consumption but also aren't so computationally intensive that they slow down execution time.

In Image Processing, convolution plays an important role in manipulating images by applying filters in the form of masks. We generally use small sized masks(3x3 or 5x5). So, using small masks involves a large number of iterations which in turn takes large computation time when they are convoluted with image. So, in this project, our objective is to implement an efficient fast algorithm to compute convolution of masks with images using Winograd's minimal filtering algorithm.

2. Approaches

- An algorithm(which is using standard matrix multiplication), in order to compute for m outputs with r-tap FIR-filter, requires mxr multiplications. But here, we will be using fast algorithms i.e winograd min-filtering algorithm and reduce the multiplications to just (m+r-1).
- Lets say, the function representing this algo is F(m, r). Now consider F(2, 3). So, using the winograd algorithm, we compute 2+3-1 = 4 multiplications which are used to build output.
- Now we can nest these 1-D Algorithms to form multidimensional fast algorithms which require the number of multiplications is equal to the number of inputs. So, to compute F(mxn, rxs)(represents cascading of two 1-D functions), we require (m+r-1)*(n+s-1) number of multiplications which is far better than m*n*r*s computation by standard multiplication method.

3. Implementation

Code Structure

kernel

- **conv.cu** file contains the basic implementation of CUDA algo for convolutional neural networks where (x, y, z) values calculated by global thread values are used to compute the (x, y)th element in the output image using the zth filter for each sample image in batch n.

  Then the implementation of Winograd algorithm is shown in the following steps:
  - P → Number of image tiles per channel
  - d(c, b) → input tile b in channel C
  - g(k, c) → filter k in channel C
  - Y(k, b) → output tile b in filter k
  - G is a filter, BT and AT are data and inverse transforms respectively.
  - we compute U and V as U = GgGT and V = BTbB
  - the required sum values can be computed by just the matrix multiplication of U and V i.e. M = UV
  - Now, we compute each output tile b in filter k as Y(k, b) = ATmA where m is gathered from M(m = M(k, b))

- utils
  - G, A, B are constants that are dependent on F(m,r)
  - The logic for the calculation of above constants is given by the authors in https://github.com/andravin/wincnn.git
  - The helper.cpp file implements the above logic and outputs G,A,B values for given F(m, r)
- winograd
  - It implements the algorithm that is given in the paper.

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: $m_{\xi,\nu} = M_{k,b}^{(\xi,\nu)}$
    $Y_{k,b} = A^T m A$

- parallel algo
  - It contains the parallelized version of the above algorithm, where each thread computes the output for every tile in the image.

4. Results

Simple Conv Layer :

input parameters : n =1 , h = 100 , w = 100 , c = 3; k =3 ; r = 3 ; m = 3 ;

time taken : `1249.0560 µs`

Winograd Algo :

input parameters : n =1 , h = 2 , w = 3 , c = 1; k =1 ; r = 3 ; m = 2 ;
time taken : `65.1840 µs`

5. Conclusion

- The Winograd algorithm uses less number of multiplications and hence works faster than the basic simple convolution layer.
- For larger input images, the winograd algorithm performs much better than the simple conv layer because of the larger difference in the number of multiplications.

References

1. Andrew L. & Scott G. (2015) : Fast Algorithms for Convolutional Neural Networks
2. Understanding 'Winograd Fast Convolution'
3. Gan T. & Libo H. (Nov, 2021) : Fast Convolution based on Winograd Minimum Filtering : Introduction and Development
4. wincnn : A simple python module for computing minimal Winograd convolution algorithms for use with convolutional neural networks

Source Code: https://github.com/namanjain0501/cuda-term-project.git

Contributions

Naman Jain :	1st part (basic conv layer)
Raj Gupta :	1st part (basic conv layer)
Vahini :	2nd part (winograd algorithm)
Devarshi :	2nd part (winograd algorithm)
Dhrumil :	2nd part (winograd algorithm)
Aayushi :	2nd part (winograd algorithm)
Jaswanth:	report and testing
Satya:	report and testing
Hemanth:	report and testing
Kunal:	report and testing