

SRIB – KLE Tech. Univ. Worklet Projects Review

Low Memory GEMM Based Convolutions for Deep Neural Networks

SRIB Mentor Name : Vikram NR,

SRIB Mentor Dept. : S/W Products R&D Team @ SRIB

KLE Team Members :

a) Deepti Hegde

b) T Santoshkumar

Jan 19th , 2019



KLE Technological
University
Creating Value
Leveraging Knowledge

Project Overview / Summary

Project Domain :

- System Programming
- Embedded Learning

Project Goals / Objectives :

- Implement a GEMM-based convolution algorithm on a memory restricted device.
- Reduce memory footprint of a convolution function.

Problem Definition :

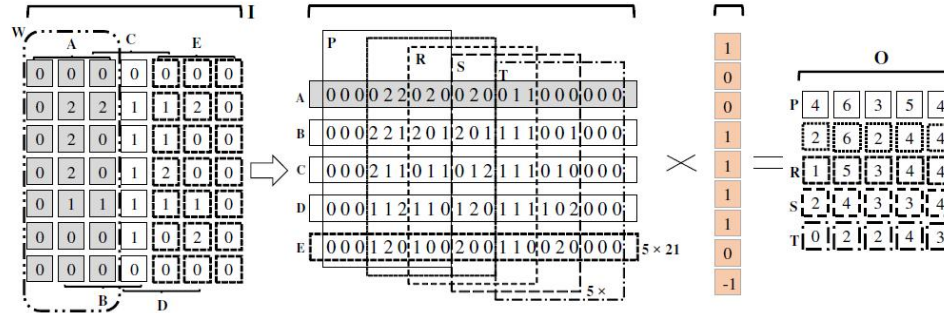
- "Memory Efficient GEMM based Convolution Implementation" - Memory footprint reduction for GEMM computation for memory restricted devices.

Technical Challenges :

- Challenge in implementation of customized kernel algorithms.
- Difficult to obtain accurate memory details for NEON assembly code.

Literature Survey

- From survey of various GEMM based convolution methods, Memory Efficient Convolutions by Cho et al is a method of multi-dimensional convolution with a two dimensional vector. This lends itself to implementation in NEON.



Algorithm 1 $O = \text{VanillaMEC}(I, K, s)$

- 1: Allocate O with $o_h o_w$ elements
- 2: Allocate L with $o_w i_h k_w$ elements
- 3: Interpret L as $o_w \times i_h \times k_w$ tensor
- 4: **for** $w \in 0 : o_w, h \in 0 : i_h$ **in parallel do**
- 5: $L[w, h, 0 : k_w] = I[h, s_w w : s_w w + k_w]$
- 6: **end for**
- 7: Interpret L as $o_w \times i_h k_w$ matrix
- 8: Interpret K as $k_h k_w \times 1$ matrix
- 9: Interpret O as $o_h \times o_w$ matrix
- 10: **for** $h \in 0 : o_h$ **in parallel do**
- 11: $O[h, 0 : o_w] =$
 $L[0 : o_w, s_h k_w h : s_h k_w h + k_h k_w] \times K$
- 12: **end for**
- 13: Return O

- Im2Col method is the predominant algorithm used for convolution based on matrix multiplication. All methods include either partial or full im2col unrolling.
- We propose to integrate these methodologies into a Compute Library Kernel.

Proposed Approach

- Optimal GEMM based algorithm is identified from memory profiling on x86_64 machine.
- For execution on a mobile device, we use Arm Compute Library for its optimized BLAS functions.
- We perform memory profiling of various possible implementations of convolution using ACL, and implement algorithm accordingly.

GEMM-based
convolution algorithm
from literature



Use computationally
efficient GEMM
kernels from ACL for
implementation



Optimized convolution
with low memory
footprint and reduced
space complexity

Experimental Results / Observations

Results of Code Profiling of Convolution - x86_64 Linux Machine (Python)

Method of Implementation	Image Size	Max. Memory Consumption (MiB)	Time Taken (seconds)
Standard TF Conv Function	[1960,1960,3]	250	24.4
	[300,300,3]	200	2.8
MEC	[1960,1960,3]	510	66.3
	[300,300,3]	53	1.8
Modified MEC	[1960,1960,3]	220	84.8
	[300,300,3]	52	1.9
Modified MEC with GEMM function	[1960,1960,3]	160	32.5
	[300,300,3]	50	1.7

Red: Best result for large images

Green: Best Result for small images

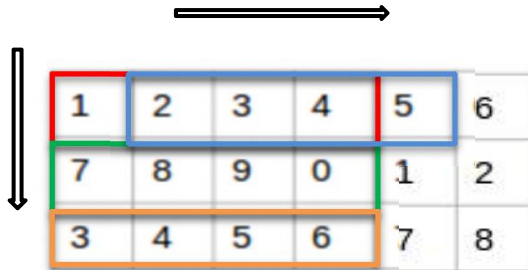
MEC = Memory Efficient Convolution

- Best performance in terms of time : **Standard TF conv function**
- Best performance in terms of memory footprint: **Modified MEC with GEMM function**
- Standard MEC algorithm fails in case of large images.

On Device - Proposed Method

For implementation on a CPU, NEON registers may be used to reduce memory access and footprint

Input Matrix



1	2	3	4	5	6
7	8	9	0	1	2
3	4	5	6	7	8

Kernel

1	2	3
4	5	6
7	8	9

Neon_REG (34x4)

step1	1x1	2x1	3x1	4x1
step2	7x4	8x4	9x4	0x4
step3	3x7	4x7	5x7	6x7
step4	2x2	3x2	4x2	5x2
step5	8x5	9x5	0x5	1x5
step6	4x8	5x8	6x8	7x8
step7	3x3	4x3	5x3	6x3
step8	9x6	0x6	1x6	2x6
step9	5x9	6x9	7x9	8x9

Experimental Results / Observations

Results of Memory Profiling - Aarch-64 armv8-A device (Android) – Redmi Note 5 PRO

Example	Method of Implementation	Native Heap(KB)		Total Memory(KB)
GEMM for 2D matrices - ACL example Basic CNN - ACL Example	ACL Convolution Layer	Before Allocation	80	703
		After Allocation	1392	2957
		Memory Aquire	1628	2821
		Memory Release	1636	2912
2D Convolution	ACL NEConvolution function	Before Allocation	70	201
		After Allocation	1632	2371
		Before Execution	1636	2379
		After Execution	1692	2579
2D Convolution Using Unrolling	NEON (arm_neon.h)	Before Allocation	80	695
		After Allocation	80	703
		Before Execution	76	691
		After Execution	80	695

Results of Memory Profiling

Aarch-64 armv8-A device (Android) – Redmi Note 5 PRO

Example	Method of Implementation	Native Heap(KB)		Total Memory(KB)
GEMM for 2D matrices - ACL example (640x48 , 480,480)	ACL NEGEMM fuction	Before Allocation	80	838
		After Allocation	6472	7448
		Before Execution	6464	7460
		After Execution	6472	7448
4x4 Matrix Multiplication	NEON (arm_neon.h)	Before Allocation	76	687
		After Allocation	80	699
		After Execution	84	732

Conclusion

- Having understood code flow of Arm Compute Library, we profiled various NEON implementations of GEMM matrix multiplication and convolutions to analyze memory usage.
- We found that the most optimal functions in ACL (for CPU) utilise NEON intrinsics to perform multiplication/convolution.
- Convolution in the Compute Library may be traced back to a number of implementations(may/may not be GEMM-based) depending on architecture and computational specifications.
- The existing kernels minimize memory usage by primarily dealing with pointers.
- Further improvements to the library, while minute, may be made by designing a more efficient method of memory access.

Thank You

SAMSUNG