Diagonalwise Refactorization: An Efficient Training Method for Depthwise Convolutions

2019-04-11

박상수

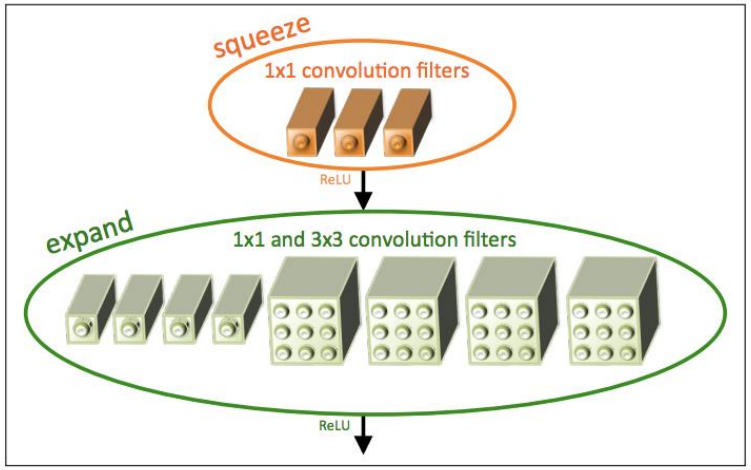
Research Area

- Parallel computation
  - Vector processor optimization (CPU/GPU/FPGA)
  - Vector processor design (RISC-V)

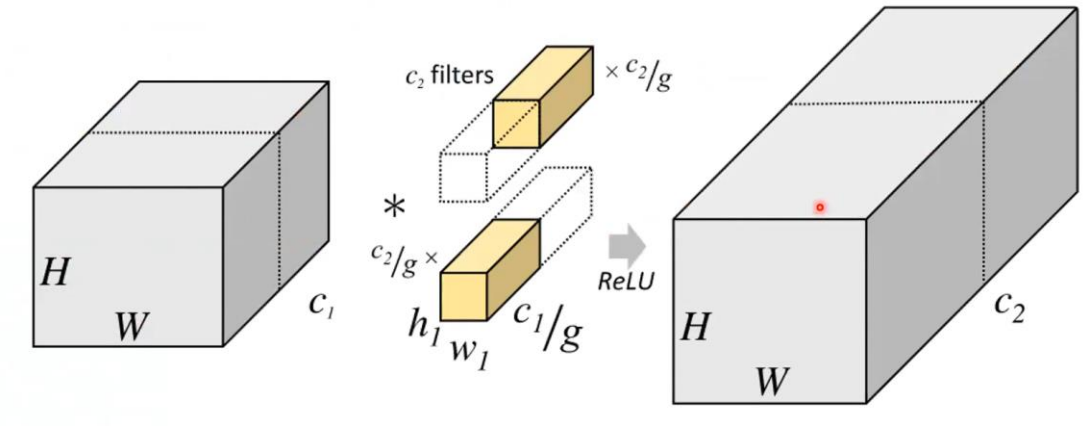
- DNN framework
  - Running on edge device (LG Robotic vacuum, Air conditioner)
  - AI speaker without any server connection (GPU/FPGA)
  - DNN accelerator

Lightweight CNN

- Designed for edge-device
  - Inference with Low cost (FLOPS, Parameters)
  - SqueezeNet, MobileNet-v1/2, ShuffleNet-v1/2, etc.



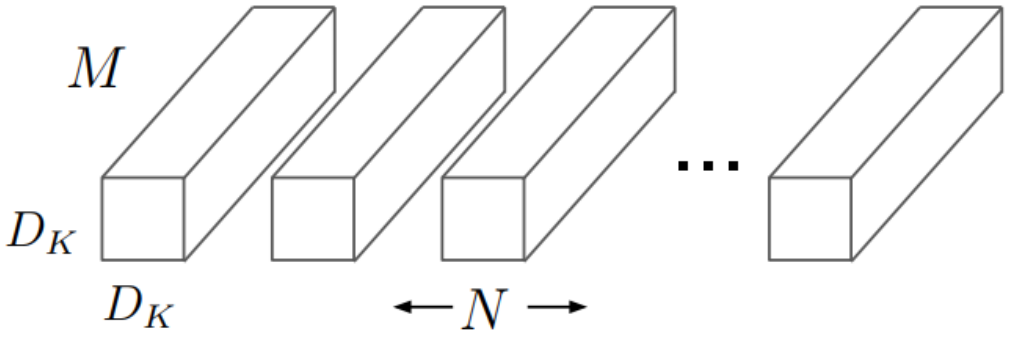
SqueezeNet (Fire module)



Group convolution

Standard Convolution

- Convolution between feature map and kernels



(a) Standard Convolution Filters

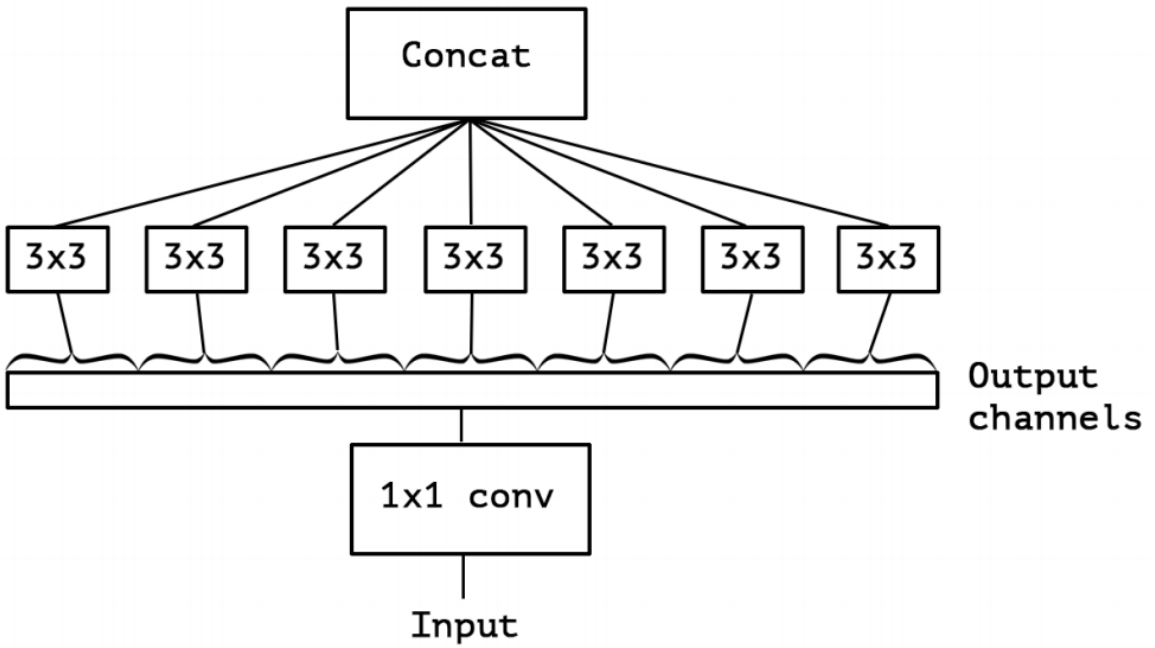
$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{2}$$

where the computational cost depends multiplicatively on the number of input channels M, the number of output channels N the kernel size $D_k \times D_k$ and the feature map size $D_F \times D_F$. MobileNet models address each of these terms and their interactions. First it uses depthwise separable convolutions to break the interaction between the number of output channels and the size of the kernel.

Xception[1]

- Factorization of convolution kernels

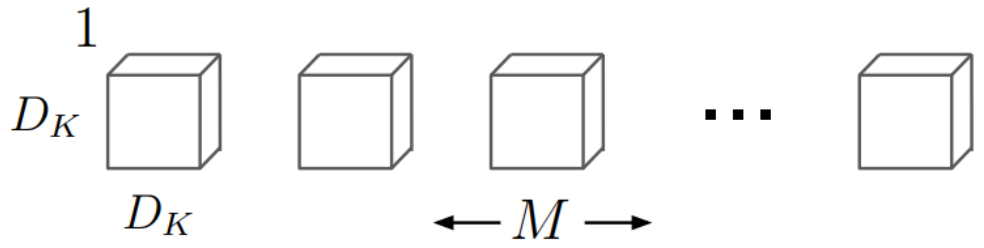
Figure 4. An "extreme" version of our Inception module, with one spatial convolution per output channel of the 1x1 convolution.



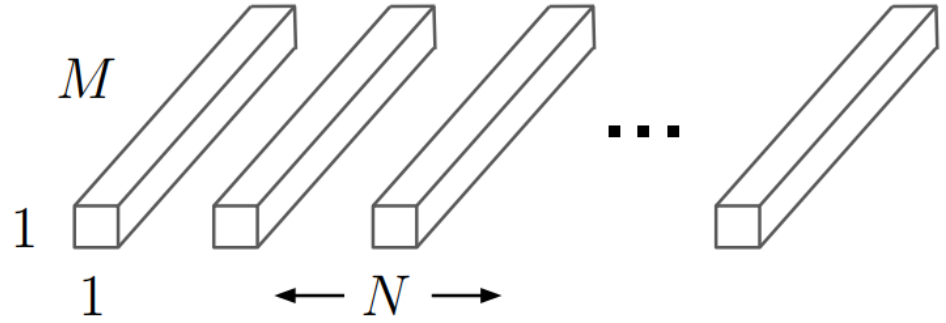
[1] Xception: Deep Learning with Depthwise Separable Convolution, Google

Depthwise Separable Convolution

- Depthwise + Pointwise (1x1 Conv)



(b) Depthwise Convolutional Filters



where $\hat{\mathbf{K}}$ is the depthwise convolutional kernel of size $D_K \times D_K \times M$ where the m_{th} filter in $\hat{\mathbf{K}}$ is applied to the m_{th} channel in $\mathbf{F}$ to produce the m_{th} channel of the filtered output feature map $\hat{\mathbf{G}}$.

Depthwise convolution has a computational cost of:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \tag{4}$$

Depthwise convolution is extremely efficient relative to standard convolution. However it only filters input channels, it does not combine them to create new features. So an additional layer that computes a linear combination of the output of depthwise convolution via 1×1 convolution is needed in order to generate these new features.

The combination of depthwise convolution and 1×1 (pointwise) convolution is called depthwise separable convolution which was originally introduced in [26].

Depthwise separable convolutions cost:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \tag{5}$$

In MobileNet Architecture #1

- DW Convolution accounts for most of the computation
  - But less parameters, Why ?

RATIOS OF MULT-ADDS, PARAMETERS, AND TRAINING TIME OF DIFFERENT LAYER TYPES FOR MOBILENETS ON CAFFE.

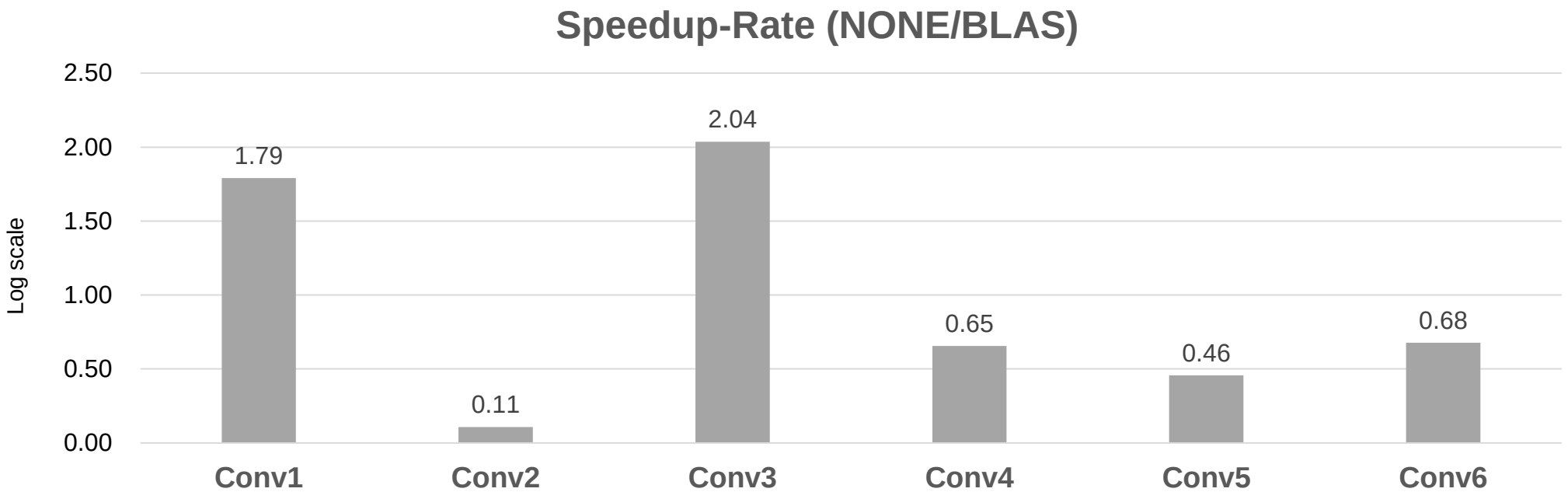
Type	Mult-Adds	Parameters	Training Time
Conv 1×1	94.86%	74.59%	16.39%
Conv DW 3×3	3.06%	1.06%	82.86%
Conv 3×3	1.19%	0.02%	0.72%
Fully Connected	0.18%	24.33%	0.03%

Conv DW: depthwise convolution layer.

In MobileNet Architecture #2

- Speedup rate
  - Standard vs Depthwise
  - OpenBLAS (Single precision at batch 1)

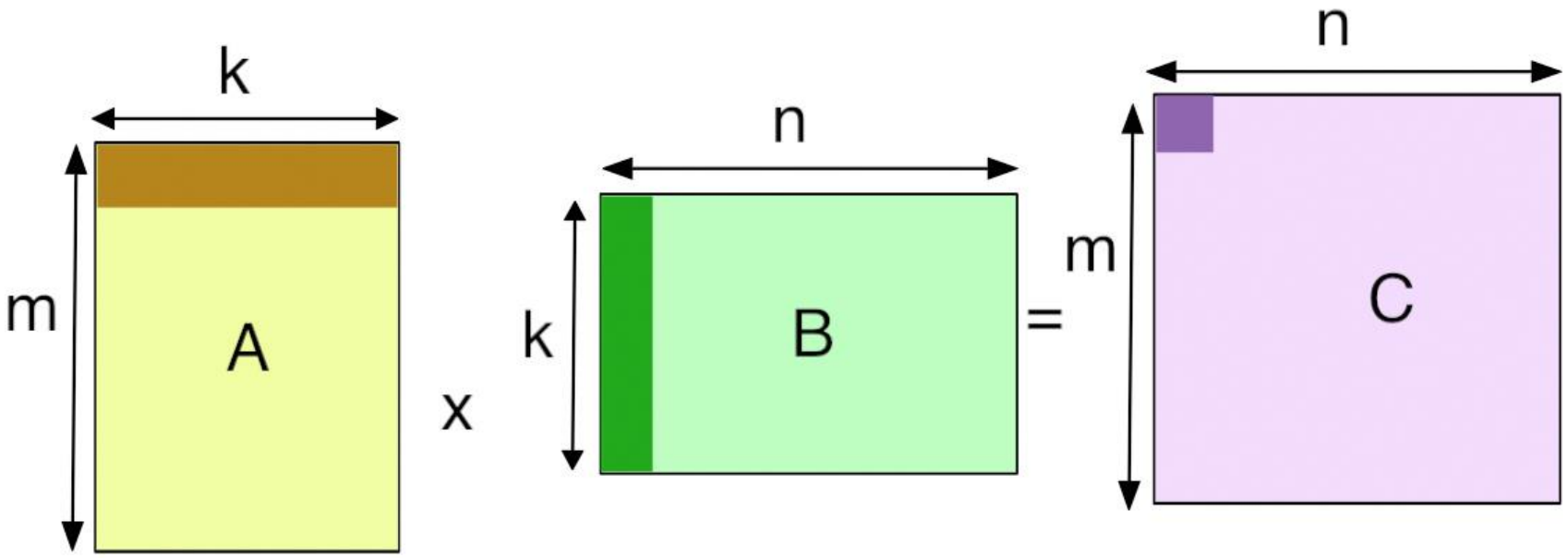
	M	N	K	CPU (NONE)	CPU (BLAS)
Conv1	50176	32	27	0.57	0.01
Conv2	50176	1	9	0.03	0.02
Conv3	50176	64	32	1.36	0.01
Conv4	12544	1	9	0.08	0.02
Conv5	12544	128	64	0.02	0.01
Conv6	12544	1	9	0.17	0.04



In MobileNet Architecture #3

- Because Matrix multiplication !
  - **BLAS API** is not always efficient

	M	N	K	CPU (NONE)	CPU (BLAS)
Conv1	50176	32	27	0.57	0.01
Conv2	50176	1	9	0.03	0.02
Conv3	50176	64	32	1.36	0.01
Conv4	12544	1	9	0.08	0.02
Conv5	12544	128	64	0.02	0.01
Conv6	12544	1	9	0.17	0.04

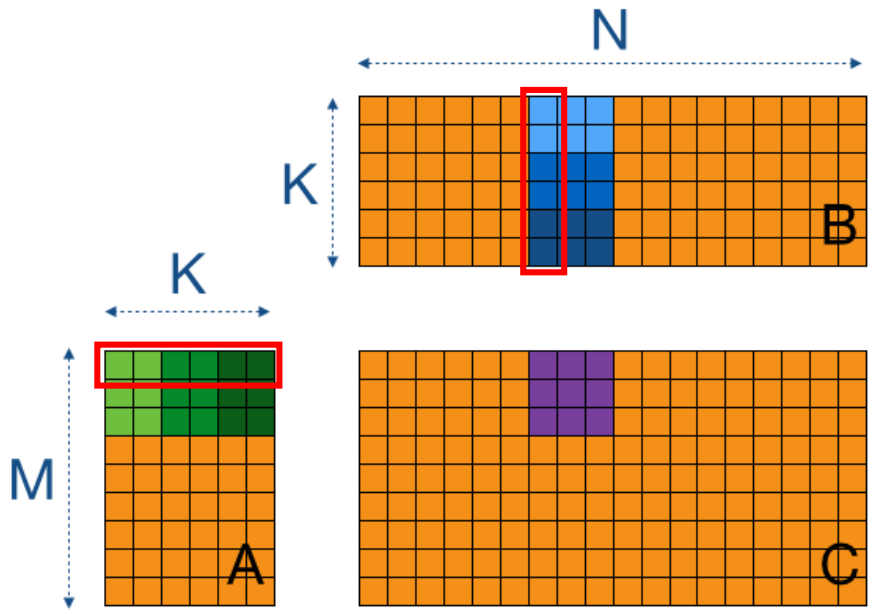


In MobileNet Architecture #3

- Depend on ***N*** size
  - **Different speedup rate**
  - **Solution: change to standard conv**

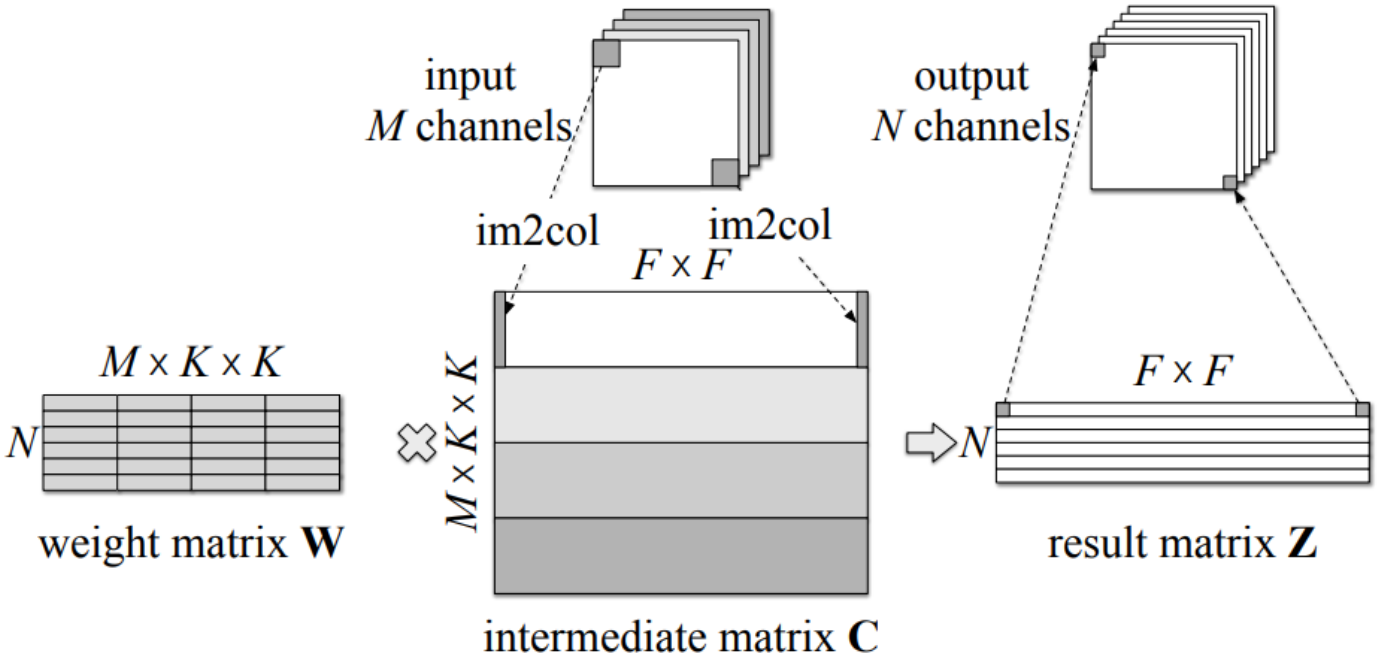
	M	N	K	CPU (NONE)	CPU (BLAS)
Conv1	50176	32	27	0.57	0.01
Conv2	50176	1	9	0.03	0.02
Conv3	50176	64	32	1.36	0.01
Conv4	12544	1	9	0.08	0.02
Conv5	12544	128	64	0.02	0.01
Conv6	12544	1	9	0.17	0.04

```
/* Multiply n x n matrices a and b */
void mmm(int n, double a[n][n], double b[n][n], double c[n][n]) {
 int i, j, k;
 for (i = 0; i < n; i+=B)
   for (j = 0; j < n; j+=B)
      for (k = 0; k < n; k+=B) /* B x B mini matrix multiplications */
         for (i1 = i; i1 < i+B; i1++)
            for (j1 = j; j1 < j+B; j1++)
               for (k1 = k; k1 < k+B; k1++)
                  c[i1][j1] += a[i1][k1]*b[k1][j1]; }
}
```



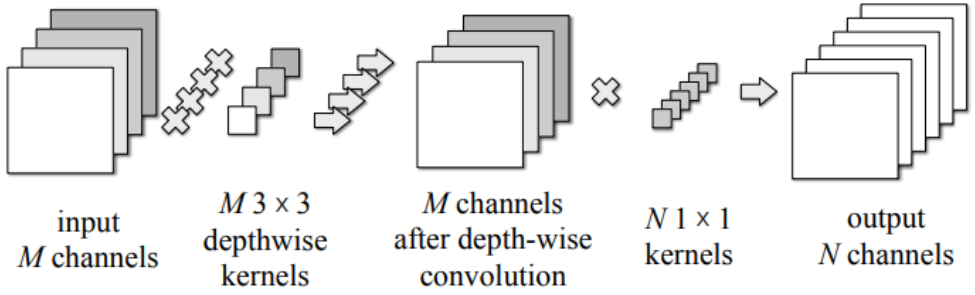
GEMM based acceleration #1

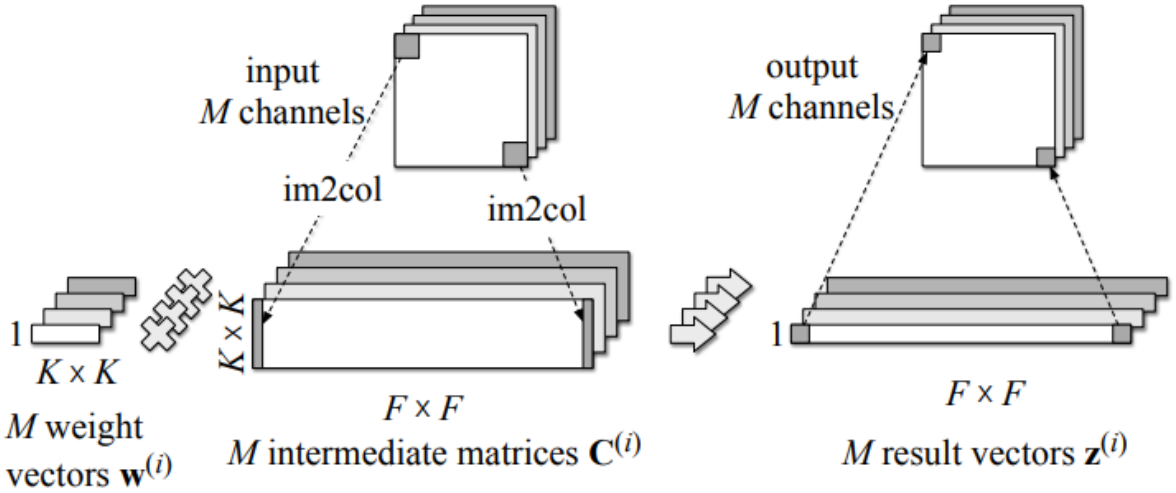
- Matrix multiplication in standard convolution



GEMM based acceleration #2

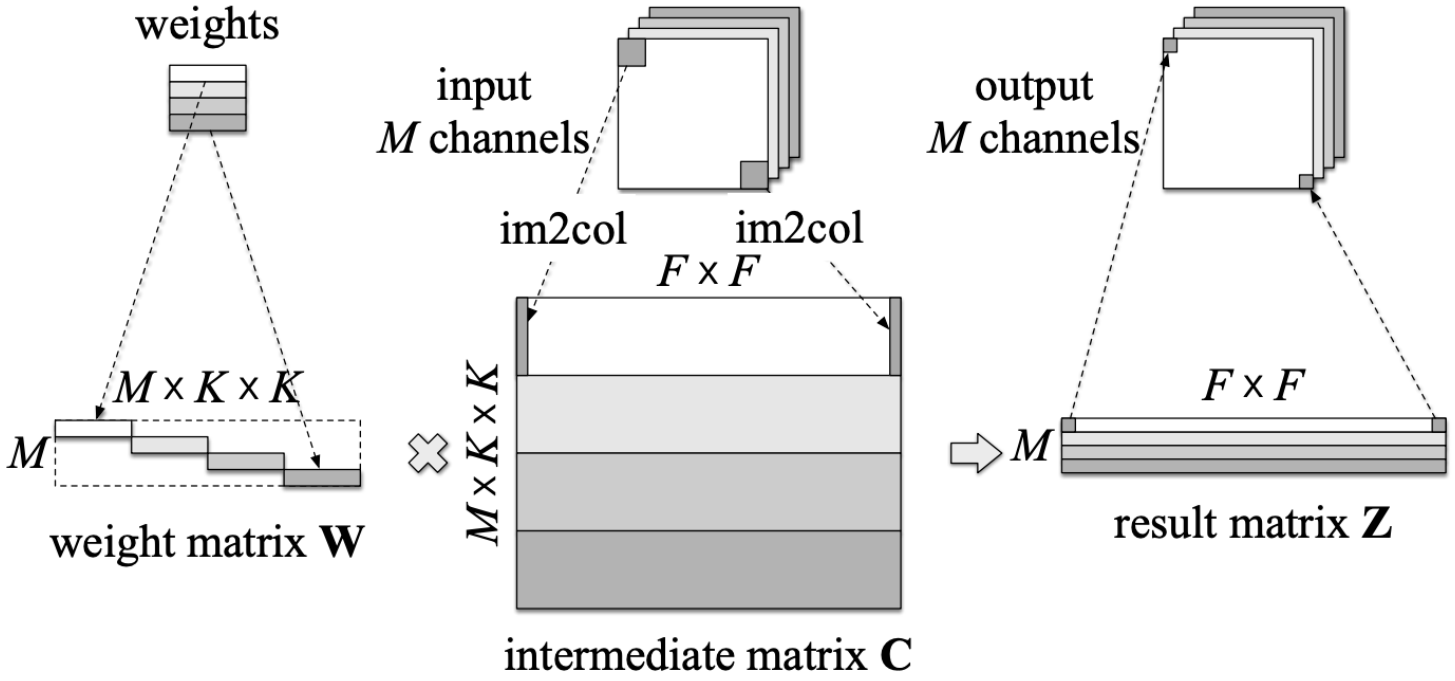
- Matrix multiplication in depthwise convolution
  - Channel-by-channel method





Diagonalwise Refactorization #1

- Matrix multiplication with 4-channel diagonalwise group
  - Input channels ***M*** into ***G*** (1~16) group (all to 128)



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}^{(1)} & & & \\ & \mathbf{w}^{(2)} & & \\ & & \ddots & \\ & & & \mathbf{w}^{(M)} \end{bmatrix}$$

Diagonalwise Refactorization #2

- Forward and Backward Propagation time (ms)
  - C-by-C: channel-by-channel
  - Specialized: cuDNN

Layer Number	Layer Configuration	Forward			Backward			
		C-by-C GEMM	*Specialized Kernel*	*Diagonalwise cuDNN*	*C-by-C GEMM*	*Specialized Kernel*	*Specialized Kernel**	*Diagonalwise cuDNN*
2	$3 \times 3/1, 112 \times 112 \times 32$	19.63	10.29	**7.45**	194.55	186.79	23.22	**22.84**
4	$3 \times 3/2, 112 \times 112 \times 64$	16.56	4.63	**3.94**	109.85	46.14	13.51	**15.93**
6	$3 \times 3/1, 56 \times 56 \times 128$	32.03	8.51	**7.45**	209.66	81.44	17.47	**16.50**
8	$3 \times 3/2, 56 \times 56 \times 128$	22.72	2.06	**2.05**	78.16	12.73	6.59	**7.02**
10	$3 \times 3/1, 28 \times 28 \times 256$	45.54	3.90	**3.86**	150.51	20.74	8.42	**7.79**
12	$3 \times 3/2, 28 \times 28 \times 256$	37.17	**1.06**	1.19	75.52	5.22	3.40	**3.86**
14, 16, 18, 20, 22	$3 \times 3/1, 14 \times 14 \times 512$	79.31	**2.00**	2.06	159.60	7.21	4.21	**4.60**
24	$3 \times 3/2, 14 \times 14 \times 512$	69.42	**0.59**	1.10	133.97	**2.32**	1.84	4.41
26	$3 \times 3/1, 7 \times 7 \times 1024$	139.03	**1.06**	1.50	268.62	**3.24**	2.28	3.49
Total		461.42	34.10	30.61	1380.43	365.84	80.95	86.44