DWM: A Decomposable Winograd Method for Convolution Acceleration

The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI), 2020

Presenter: Sang Heon Lee

August 4, 2020

Neural Network Acceleration Study Season #2

Contents of presentation

- **Introduction**
- **Winograd Algorithm**
- **Decomposable Winograd Method**
- **Experiments**
- **Conclusion**

Introduction

- Increasing amount of computations
  - AlexNet (2012) => 7 × 10^8 multiplications
  - VGG-16 (2014) => 1.5× 10^{10} multiplications
  - SENet-154 (2018) => 2.1× 10^{10} multiplications
- Reduce the number of multiplications in convolutions
  - Lavin (2016) applied Winograd's minimal filtering algorithm (Winograd 1980)
  - But Winograd's minimal filtering algorithm is only effective on 3×3 kernels with stride 1
  - So they propose the Decomposable Winograd Method(DWM) to extend the Winograd's minimal filtering algorithm into the cases of large kernels and stride > 1

Winograd Algorithm

- **Im2col + GEMM 합성곱**

  - Denoting the result of computing m outputs with an r-tap FIR filter as F(m, r)
  - 출력이 m개이고 필터 사이즈가 r인 1-D 합성곱을 F(m, r)로 표현할 수 있다.
  - F(2, 3) 일 때는 출력이 2, 필터사이즈가 3 이다. g는 filter와 관련, d는 input과 관련됨.

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} d_0 \boxed{\times} g_0 \oplus d_1 \boxed{\times} g_1 \oplus d_2 \boxed{\times} g_2 \\ d_1 \boxed{\times} g_0 \oplus d_2 \boxed{\times} g_1 \oplus d_3 \boxed{\times} g_2 \end{bmatrix}$$

  - F(2, 3)의 경우, 6개의 곱셈과 4개의 덧셈이 필요하다.
  - 이를 확장하여 m×n 출력과 r×s의 필터를 갖는 2-D 합성곱은 F(m×n, r×s)로 표현할 수 있다
  - 이때 2-D 합성곱에 필요한 곱셈의 개수는 m×n×r×s개 이다.

Winograd Algorithm

- **Winograd 합성곱**

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix}$$

Where,

$$m_1 = (d_o - d_2) \boxed{\times} g_0, \quad m_2 = (d_1 + d_2) \boxed{\times} \frac{g_0 + g_1 + g_2}{2}$$

$$m_4 = (d_1 - d_3) \boxed{\times} g_2, \quad m_3 = (d_2 - d_1) \boxed{\times} \frac{g_0 - g_1 + g_2}{2}$$

중복되는 연산 = $\dfrac{g_0+g_2}{2}$

- F(2, 3)의 경우, 4개의 곱셈과 11개의 덧셈이 필요하다.
- 2-D Winograd 합성곱 필요한 곱셈의 개수는 (m+r-1)(n+s-1)개이다.

=> 덧셈연산 보다 곱셈 연산이 더 오래 걸리는데,

winograd 합성곱을 사용하면 필요한 곱셈 연산의 수를 줄일 수 있다.

Winograd Algorithm

• Winograd 합성곱 표현방식

- 1-D convolutions

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

- 2-D convolutions

$$Y = A^T[(GgG^T) \odot (B^T dB)]A.$$

$\odot$= element-wise multiplication,
g = r×r filter
d = (m+r-1)×(m+r-1) image tile
A^T =output transform matrix
G =filter transform matrix
B^T =input transform matrix

ex) F(4x4,3x3)에서의 상수행렬

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

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

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

Winograd Algorithm

- **Winograd 합성곱**

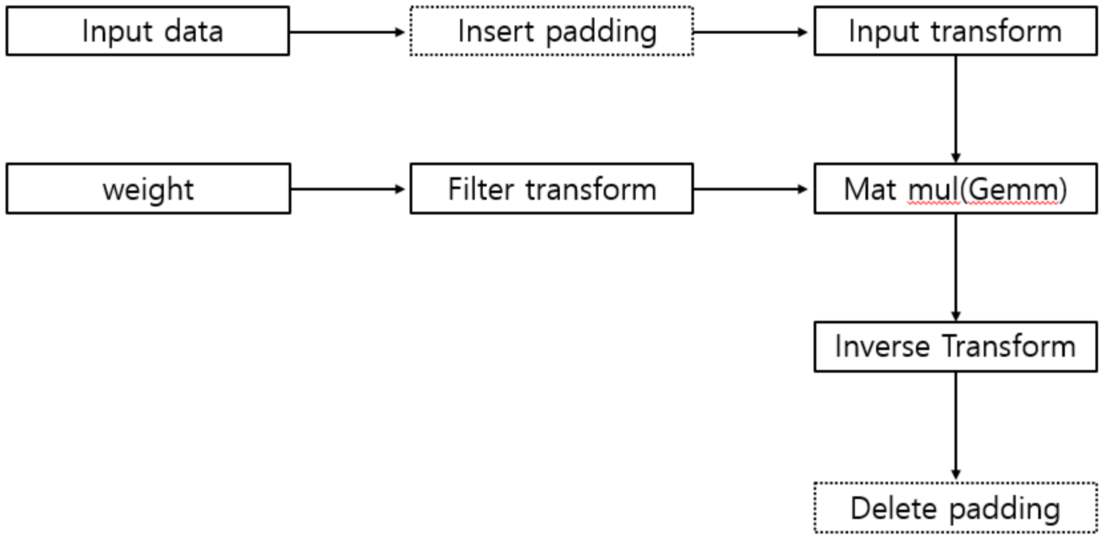
  - 2-D convolutions

Weight(Filter) transform Input transform

$$Y = A^T \left[\underbrace{\boxed{[GgG^T]}}_{\mathbf{U}} \odot \underbrace{\boxed{[B^T dB]}}_{\mathbf{V}} \right] A$$

Inverse transform

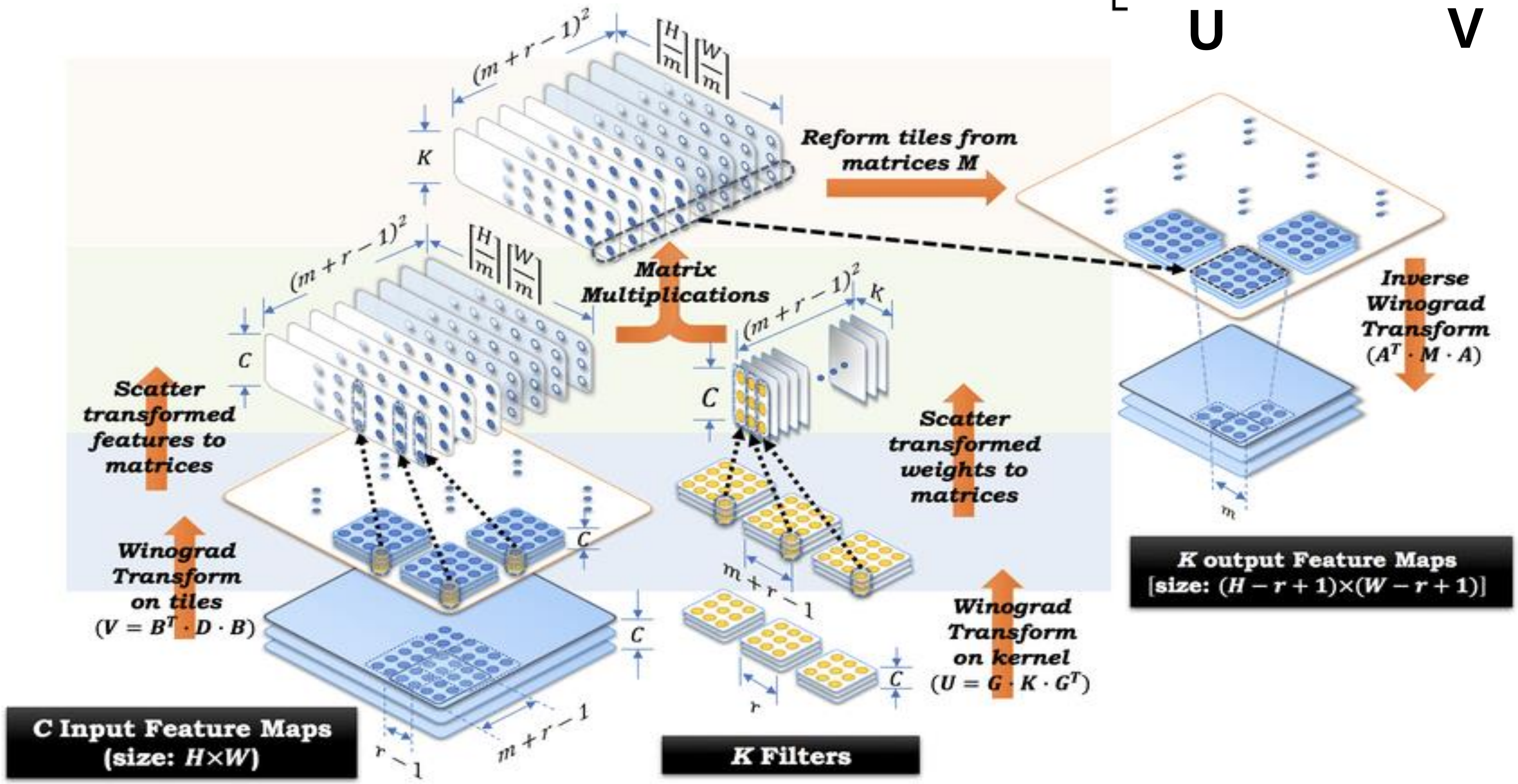
$$Y = \boxed{A^T \left[[GgG^T] \odot [B^T dB] \right] A}$$



Winograd Algorithm

Weight(Filter) transform Input transform

$$Y = A^T \left[\underbrace{\boxed{[GgG^T]}}_{\mathbf{U}} \odot \underbrace{\boxed{[B^T dB]}}_{\mathbf{V}} \right] A$$



Decomposable Winograd Method

• Winograd 합성곱의 단점

- Winograd algorithm can implement convolutions much more efficiently, it is always used on 3×3 and stride 1 convolutions only.

ex) F(2,5)에서의 상수행렬 => 계산이 더욱 복잡해지고, 24로 나누기 때문에 정확도 문제 발생

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

$$G = \begin{bmatrix} 1/4 & 0 & 0 & 0 & 0 \\ -1/6 & -1/6 & -1/6 & -1/6 & -1/6 \\ -1/6 & 1/6 & -1/6 & 1/6 & -1/6 \\ 1/24 & 1/12 & 1/6 & 1/3 & 2/3 \\ 1/24 & -1/12 & 1/6 & -1/3 & 2/3 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

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

Decomposable Winograd Method

- **Large Kernel Size에도 동작**

  - 1-D convolution에서 r이 filter의 size일 때 l = m + r −1

$$g(x) = g_{r-1}x^{r-1} + g_{r-2}x^{r-2} + ... + g_1 x + g_0,$$

$$d(x) = d_{l-1}x^{l-1} + d_{l-2}x^{l-2} + ... + d_1 x + d_0,$$

$$\begin{cases} g^{(0)}(x) = g_2x^2 + g_1x + g_0 \\ g^{(1)}(x) = (g_5x^2 + g_4x + g_3)x^3 \\ \vdots \\ g^{(\lfloor r/3 \rfloor)}(x) = \sum_{i=0}^{r-1 \ mod \ 3} g_{r-i-1}x^{(r-1 \ mod \ 3)-i}x^{3\lfloor r/3 \rfloor} \end{cases}.$$

Then from $g(x) = \sum_{i=0}^{\lfloor r/3 \rfloor} g^{(i)}(x)$ we can get

$$y(x) = g(x)d(x) = \sum_{i=0}^{\lfloor r/3 \rfloor} [g^{(i)}(x)d(x)] = \sum_{i=0}^{\lfloor r/3 \rfloor} y^i(x),$$

For example, when $r = 5$, we can split it into two parts

$$\begin{cases} g^{(0)}(x) = g_2x^2 + g_1x + g_0 \\ g^{(1)}(x) = (g_4x + g_3)x^3 \end{cases}, \quad (10)$$

and $g(x) = g^{(0)}(x) + g^{(1)}(x)$. Then we get:

$$y(x) = g(x)d(x) = g^{(0)}(x)d(x) + g^{(1)}(x)d(x). \quad (11)$$

Decomposable Winograd Method

- **2-D convolution에도 적용**

  - split the large kernel into small parts
  - can process a common large kernel convolution in five steps

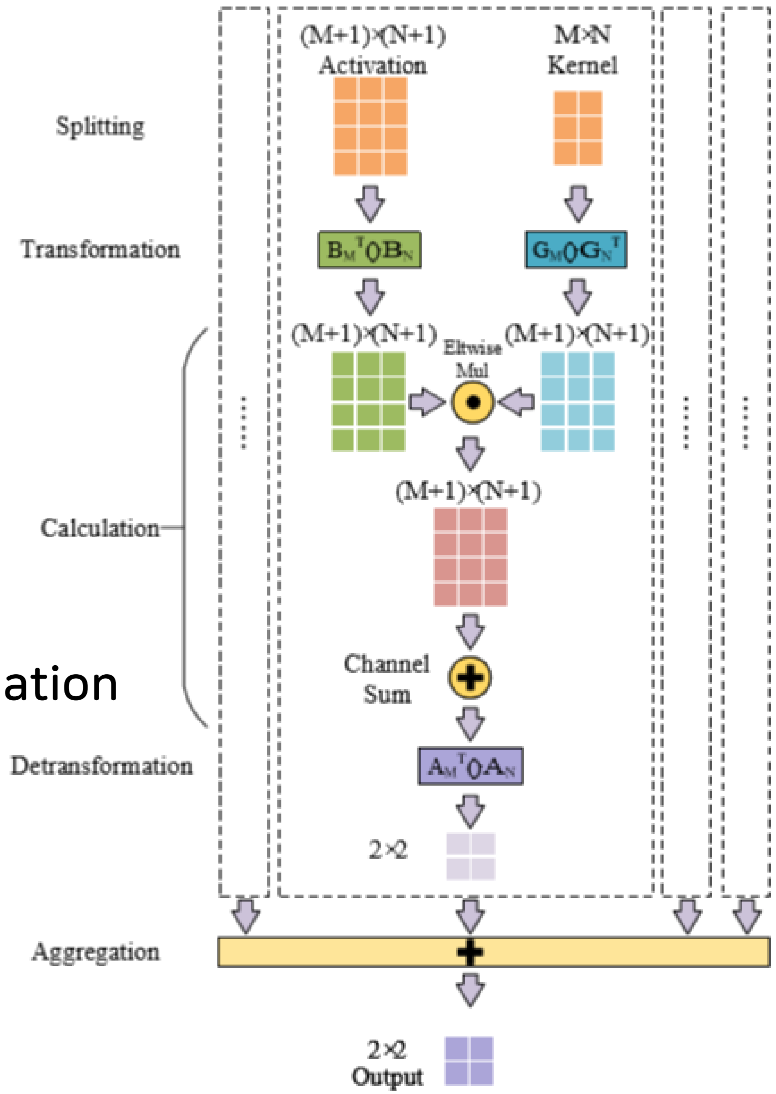
1. Splitting => Split the convolution kernel into several parts

2. Transformation => $B^T(\cdot)B \ and \ G(\cdot)G^T$ (Input, filter transform)

3. Calculation => element-wise multiplication and channel-wise summation

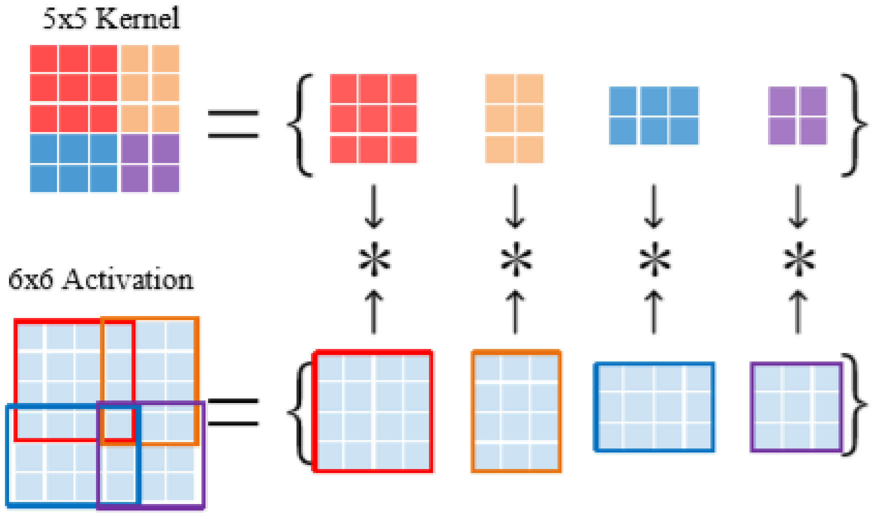
4. Detransformation => $A^T(\cdot)A$ (Inverse transform)

5. Aggregation => Sum the calculation results of each part



Decomposable Winograd Method

- **Splitting(Large Kernel Size)**



(a)

$$
\begin{cases}
g^{(0)}(x) = g_2 x^2 + g_1 x + g_0 \\
g^{(1)}(x) = (g_5 x^2 + g_4 x + g_3) x^3 \\
\vdots \\
g^{(\lfloor r/3 \rfloor)}(x) = \sum_{i=0}^{r-1 \ mod \ 3} g_{r-i-1} x^{(r-1 \ mod \ 3)-i} x^{3\lfloor r/3 \rfloor}
\end{cases}
.
$$

Then from $g(x) = \sum_{i=0}^{\lfloor r/3 \rfloor} g^{(i)}(x)$ we can get

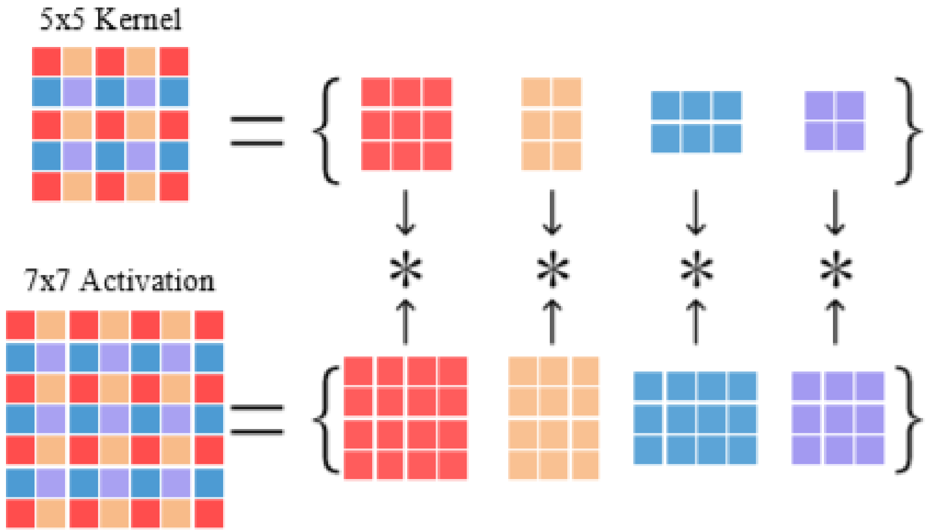
$$
y(x) = g(x)d(x) = \sum_{i=0}^{\lfloor r/3 \rfloor} [g^{(i)}(x)d(x)] = \sum_{i=0}^{\lfloor r/3 \rfloor} y^i(x),
$$

(a): Splitting a 5×5 and stride 1 convolution into four smaller convolutions

=> 2-D, 5 × 5 convolution case, we can split the kernel into 4 parts:3×3, 3×2, 2×3 and 2×2,

Decomposable Winograd Method

- **Splitting (Stride > 1)**



(b)

$$
\begin{cases}
g^{(0)}(x) = \displaystyle\sum_{i=0}^{\lfloor (r-1)/s \rfloor} g_{s*i} x^{s*i} \\
g^{(1)}(x) = \displaystyle\sum_{i=0}^{\lfloor (r-2)/s \rfloor} g_{s*i+1} x^{s*i+1} \\
\vdots \\
g^{(s-1)}(x) = \displaystyle\sum_{i=0}^{\lfloor (r-s-1)/s \rfloor} g_{s*i+s-1} x^{s*i+s-1}
\end{cases}.
$$

(b): Splitting a 5 × 5 and stride 2 convolution into four stride 1 convolutions

Decomposable Winograd Method

• Comparison and Discussion

- 기존에는 large kernel size problem을 해결하기 위해 kernel에 패딩을 추가하는 방법을 사용했다. (Lu et al. 2017)

=> non-zero values 값으로 채워져서 추가적인 계산이 필요함 => 소요 시간 증가

- DWM 방법에서 convolution operations을 적절히 나눴기 때문에 패딩을 넣어줄 필요X

- Achieves the best acceleration without any numerical accuracy loss

F(2,3)

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

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

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

F(2,5)

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

$$G = \begin{bmatrix} 1/4 & 0 & 0 & 0 & 0 \\ -1/6 & -1/6 & -1/6 & -1/6 & -1/6 \\ -1/6 & 1/6 & -1/6 & 1/6 & -1/6 \\ 1/24 & 1/12 & 1/6 & 1/3 & 2/3 \\ 1/24 & -1/12 & 1/6 & -1/3 & 2/3 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

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

Decomposable Winograd Method

- **Experiments (Setup)**

*NVIDIA V100 GPU

- Doing convolution with the standard normal distribution random numbers on one single layer
- and then calculate the mean squared error(MSE) with FP64 convolution results
- ProxylessNAS (Cai, Zhu, and Han 2018) 적용
- The Networks' accuracy was measured on ImageNet 2012 (Russakovsky et al. 2015).

Decomposable Winograd Method

- **Experiments (Numerical Accuracy of Single Layer)**

Table 1: Mean squared error (MSE) between different acceleration algorithms and the FP64 result by running a forward convolution, i.e FP64 convolution result is the baseline. H/W means the size of featur map, and FP32/FP16 indicates doing convolution in FP32/FP16 format.

Kernel Size	H/W	Channel	Filters	FP32	FP16	Winograd FP32	Winograd FP16	DWM FP32	DWM FP16
3x3	14	256	256	2.21E-08	2.71E-04	5.24E-10	1.44E-03	5.32E-10	3.42E-02
3x3	28	128	128	1.11E-10	1.41E-04	1.43E-10	7.48E-04	1.47E-10	9.08E-03
5x5	14	256	256	1.05E-02	6.93E-04	7.14E-08	1.07E-01	1.47E-09	9.72E-02
5x5	28	128	128	3.15E-10	3.80E-04	2.00E-08	5.78E-02	4.33E-10	2.83E-02
7x7	14	256	256	6.13E-10	1.25E-03	1.47E-03	NaN	2.97E-09	1.97E-01
7x7	28	128	128	5.61E-10	7.16E-04	4.24E-04	NaN	8.86E-10	5.88E-02
9x9	14	256	256	9.90E-10	1.90E-03	5.31E-03	NaN	3.67E-09	2.36E-01
9x9	28	128	128	8.52E-10	1.14E-03	1.62E-03	NaN	1.18E-09	7.33E-02
11x11	14	256	256	1.47E-09	2.60E-03	1.48E-02	NaN	5.30E-09	3.46E-01
11x11	28	128	128	1.15E-09	1.63E-03	4.35E-03	NaN	1.81E-09	1.15E-01

*NVIDIA V100 GPU

Decomposable Winograd Method

- **Experiments (Numerical Accuracy of Single Layer)**

Table 2: Top-1 accuracy, FLOPs and speedup of several acclerating algorithm on different networks. Origin means the original top-1 accuracy and FLOPs of networks.

Network	Origin		Winograd			DWM		
	Acc	GFLOPs	Acc	GFLOPs	speedup	Acc	GFLOPs	speedup
AlexNet (Krizhevsky, Sutskever, and Hinton 2012)	56.52	0.71	56.51	0.56	1.28	56.51	0.45	1.57
GoogLeNet (Szegedy et al. 2015)	69.79	1.51	69.79	0.97	1.55	69.77	0.92	1.65
Inception-V3 (Szegedy et al. 2016)	69.54	2.86	69.47	2.34	1.22	69.46	1.92	1.49
ResNet-152 (He et al. 2016)	78.31	11.62	78.31	8.78	1.32	78.31	8.60	1.35
DenseNet-161 (Huang et al. 2017)	77.14	7.88	77.13	6.32	1.25	77.12	6.23	1.26
ProxylessGPU (Cai, Zhu, and Han 2018)	75.08	0.49	74.77	0.49	1.01	75.06	0.47	1.05
ProxylessMobile (Cai, Zhu, and Han 2018)	74.59	0.35	74.47	0.34	1.01	74.57	0.33	1.06

*NVIDIA V100 GPU

Decomposable Winograd Method

- **Experiments (FLOPs Estimation on Single Layer)**

Table 3: The speedup of several acclerating algorithms on different kinds of convolutions. We assume that the output size is fixed to 14×14.

Kernel Size	Stride	Direct	Winograd		DWM	
		FLOPs	FLOPs	speedup	FLOPs	speedup
3x3	1	1.76E+03	784	2.25	784	2.25
5x5	1	4.90E+03	1.48E+04	0.33	2.40E+03	2.04
7x7	1	9.60E+03	9.72E+04	0.10	4.90E+03	1.96
9x9	1	1.59E+04	3.16E+05	0.05	7.06E+03	2.25
11x11	1	2.37E+04	1.07E+06	0.02	1.10E+04	2.15
3x3	2	1.76E+03	N/A	N/A	1.23E+03	1.44
5x5	2	4.90E+03	N/A	N/A	2.40E+03	2.04
7x7	2	9.60E+03	N/A	N/A	4.90E+03	1.96
9x9	2	1.59E+04	N/A	N/A	8.28E+03	1.92
11x11	2	2.37E+04	N/A	N/A	1.10E+04	2.15

*NVIDIA V100 GPU

Decomposable Winograd Method

- **Experiments (FLOPs Estimation on Single Layer)**

Table 4: The actual runtime of several acclerating algorithms on different kinds of convolutions tested by nvprof. The batch size, the channels and the filters are 256. The input size is fixed to 14×14.

Kernel Size	DWM(ms)	Winograd(ms)	cuDNN(ms)	Wino/DWM	cuDNN/DWM
3x3	3.67	3.35	2.80	0.91	0.76
5x5	11.37	69.70	11.26	6.13	0.99
7x7	22.83	133.34	24.51	5.84	1.07
9x9	30.67	248.71	50.92	8.11	1.66
11x11	48.29	349.33	94.63	7.23	1.96

*NVIDIA V100 GPU

Conclusion

- Winograd’s minimal filtering algorithm has been widely used to reduce the number of multiplications for faster processing

- However, it has the drawbacks of sufferring from significantly increased FLOPs and numerical accuracy problem for kernel size larger than 3x3 and failing on convolution with stride larger than 1

- 기존의 Winograd 방법이 kernel size as 3x3 and stride as 1 일 때 효율적으로 작동

=> To solve this problems, we propose DWM to break through the limitation of original Winograd’s minimal filtering algorithm on convolutions of large kernel and large stride

- Experimental results show that the proposed DWM is able to support all kinks(?) of convolutions with a speedup of ~2, without affecting the numerical accuracy.

Thank you