LOW RANK APPROXIMATION

Low Rank Approximation

- Layer responses lie in a low-rank subspace
- Decompose a convolutional layer with d filters with filter size $k \times k \times c$ to
  - A layer with d' filters ($k \times k \times c$)
  - A layer with d filter ($1 \times 1 \times d'$)

$$A * B = C$$
$$B = U \Lambda V^T$$
$$C = A * (U \Lambda V^T) = (A * U) * \Lambda * V^T$$

SVD，CP分解、Tucker分解、Tensor Train分解、Block Term分解方法,現在很多網絡都是1x1的小的捲積，已經比較快了，用矩陣分解很難進行加速和壓縮

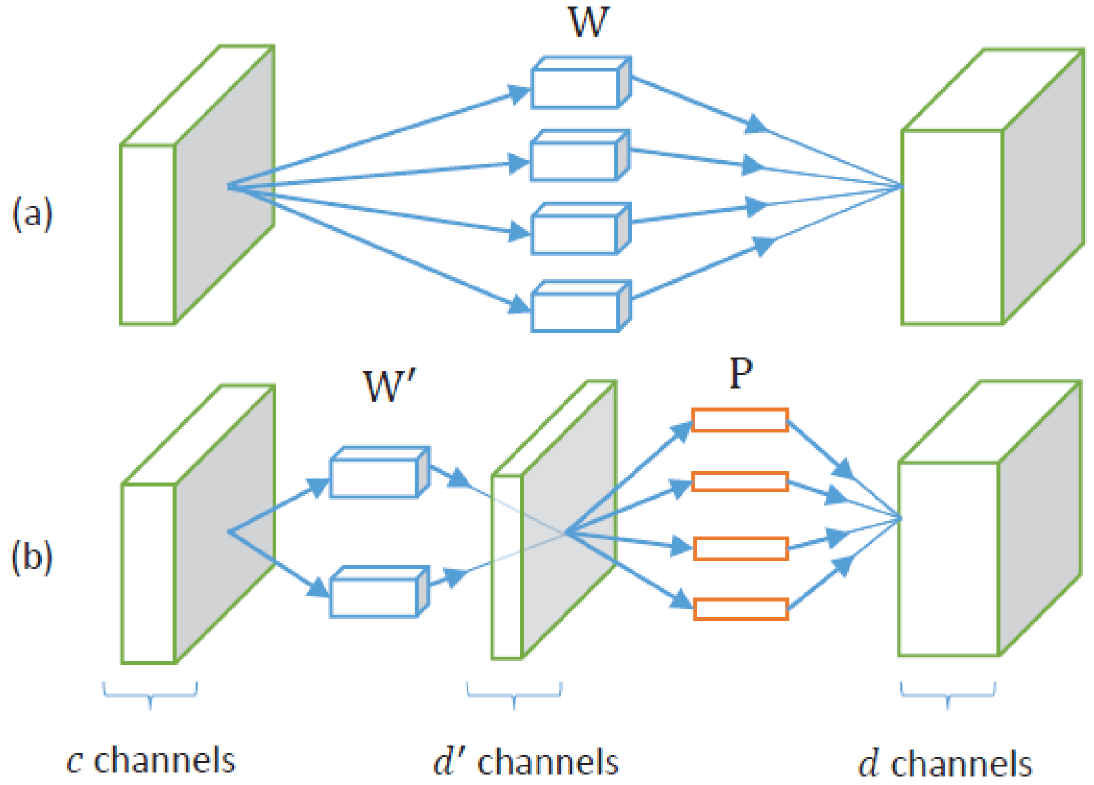


Figure 1. Illustration of the approximation. (a) An original layer with complexity $O(dk^2c)$. (b) An approximated layer with complexity reduced to $O(d'k^2c) + O(dd')$.

Zhang et al Efficient and Accurate Approximations of Nonlinear Convolutional Networks CVPR'15

- Method
  - SVD. tucker. CP

speedup	rank sel.	Conv1	Conv2	Conv3	Conv4	Conv5	Conv6	Conv7	err. ↑ %
2×	no	32	110	199	219	219	219	219	1.18
2×	**yes**	32	83	182	211	239	237	253	**0.93**
2.4×	no	32	96	174	191	191	191	191	1.77
2.4×	**yes**	32	74	162	187	207	205	219	**1.35**
3×	no	32	77	139	153	153	153	153	2.56
3×	**yes**	32	62	138	149	166	162	167	**2.34**
4×	no	32	57	104	115	115	115	115	4.32
4×	**yes**	32	50	112	114	122	117	119	**4.20**
5×	no	32	46	83	92	92	92	92	6.53
5×	**yes**	32	41	94	93	98	92	90	**6.47**

低秩估計方法存在一個待解決的問題，就是保留多少秩是不明確的

Zhang et al Efficient and Accurate Approximations of Nonlinear Convolutional Networks CVPR'15

WINOGRAD TRANSFORMATION

Convolution

Definition: Convolution in the time domain is equivalent to pointwise multiply in the frequency domain.

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

$\mathcal{F}\{f\}$ and $\mathcal{F}\{g\}$ are the Fourier transforms of f and g

The asterisk denotes convolution, not multiplication.

$$f = \begin{bmatrix} 0 & 0 & 1 & 2 \end{bmatrix}$$

$$g = \begin{bmatrix} 10 & 20 & 30 \end{bmatrix}$$

$$f * g = \begin{bmatrix} 30 & 80 \end{bmatrix}$$

Dot Product Approach

$$out[0] = 0 * 10 + 0 * 20 + 1 * 20 = 30$$

$$out[1] = 0 * 10 + 1 * 20 + 2 * 20 = 80$$

- 6 Multiplications
- 4 Additions

Shmuel Winograd (Winograd FFTs)

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

input

weight

where

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

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

- Data (d's): 4 ADDs
- Filter (g's): 3 ADDs, 2 MULs
- Outputs (m's): **4 MULs**, **4 ADDs**

Fast algorithms for convolutional neural networks

Winograd Convolution

- Winograd's minimal filtering algorithm for small filter sizes (3x3)
  - computing m outputs with an r-tap FIR filter, which we call F(m, r), requires $\mu(F(m, r)) = m + r - 1$ multiplications
  - minimal 2D algorithms: for computing m × n outputs with an r × s filter, which we call F(m × n, r × s). These require μ(F(m × n, r × s)) = μ(F(m, r))μ(F(n, s)) = (m + r − 1)(n + s − 1)

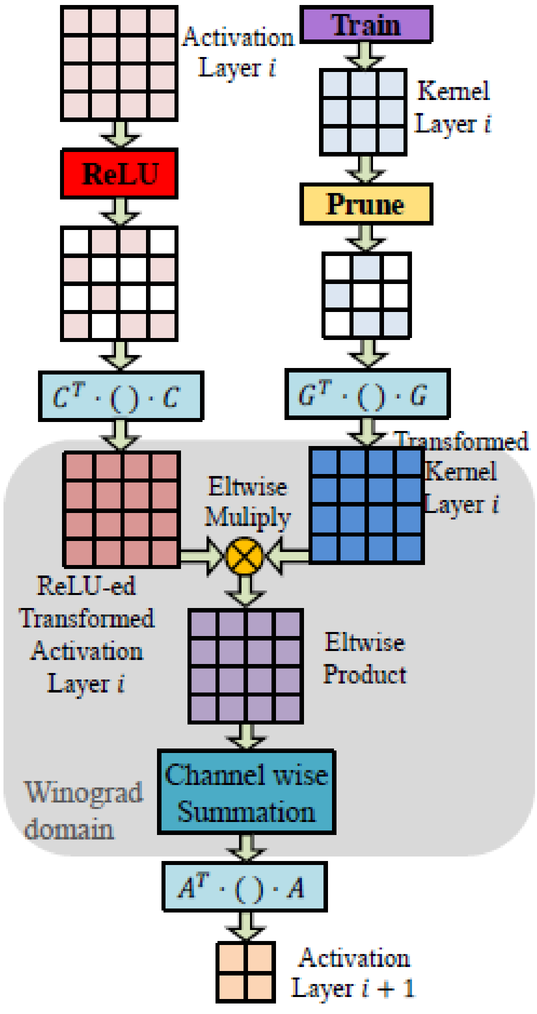
$\mu(F(2,3)) = 2 + 3 - 1 = 4.$ 4 mul v.s. 6 mul

F(2×2, 3×3) uses 4×4 = 16 multiplications
standard algorithm uses 2 × 2 × 3 × 3 = 36
an arithmetic complexity reduction of 36/16 = **2.25**

N	cuDNN msec	cuDNN TFLOPS	F(2x2,3x3) msec	F(2x2,3x3) TFLOPS	Speedup
1	12.52	3.12	5.55	7.03	2.26X
2	20.36	3.83	9.89	7.89	2.06X
4	104.70	1.49	17.72	8.81	5.91X
8	241.21	1.29	33.11	9.43	7.28X
16	203.09	3.07	65.79	9.49	3.09X
32	237.05	5.27	132.36	9.43	1.79X
64	394.05	6.34	266.48	9.37	1.48X

Table 5. cuDNN versus $F(2 \times 2, 3 \times 3)$ performance on VGG Network E with fp32 data. Throughput is measured in Effective TFLOPS, the ratio of direct algorithm GFLOPs to run time.

Winograd Convolution



(a)

Producing 4 output pixels:

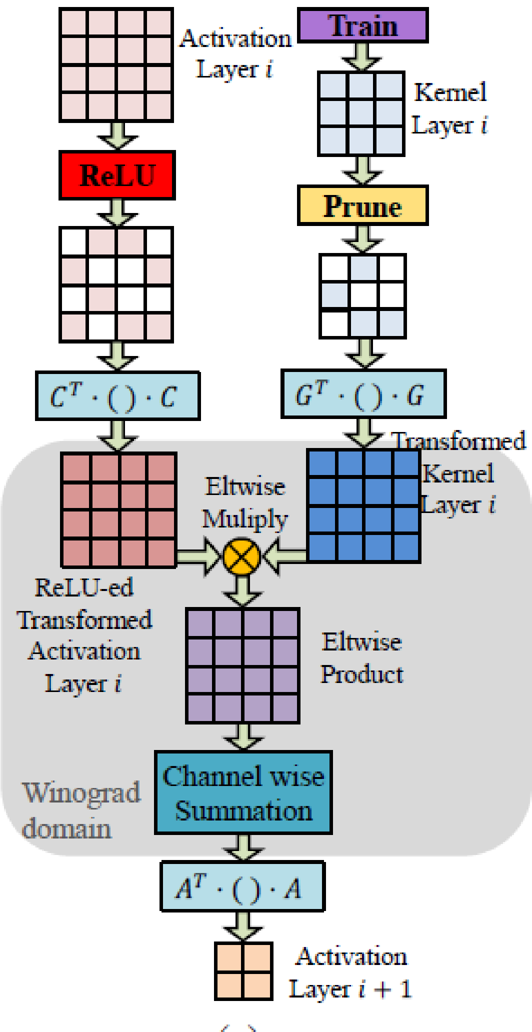
Direct Convolution:

- 4*9=36 multiplications (**1x**)

Winograd convolution:

- 4*4=16 multiplications (**2.25x** less)

Winograd Convolution v.s. Sparse



(a)

Producing 4 output pixels:

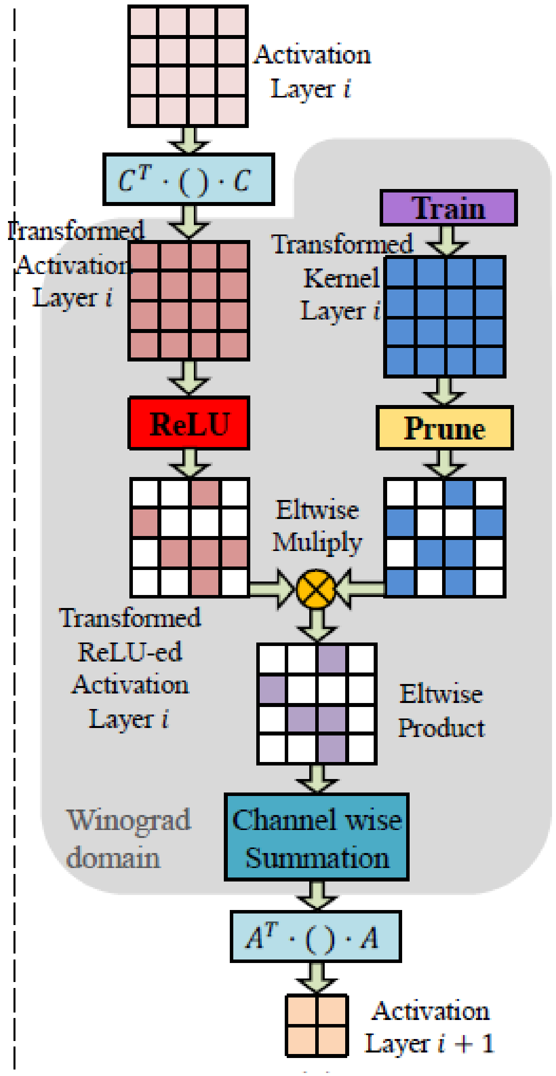
Direct Convolution:

- 4*9=36 multiplications (**1x**)
- sparse weight [NIPS'15] (**3x**)
- sparse activation (relu) (**3x**)
- Overall saving: **9x**

Winograd convolution:

- 4*4=16 multiplications (**2.25x** less)
- dense weight (**1x**)
- dense activation (**1x**)
- Overall saving: **2.25x**

Winograd Convolution + Sparse



Producing 4 output pixels:

Direct Convolution:

- 4*9=36 multiplications (**1x**)
- sparse weight [NIPS'15] (**3x**)
- sparse activation (relu) (**3x**)
- Overall saving: **9x**

Winograd convolution:

- 4*4=16 multiplications (**2.25x** less)
- sparse weight (**2.5x**)
- dense activation (**2.25x**)
- Overall saving: **12x**

- Pruned Winograd-transformed weights
- Moving ReLU to the Winograd domain

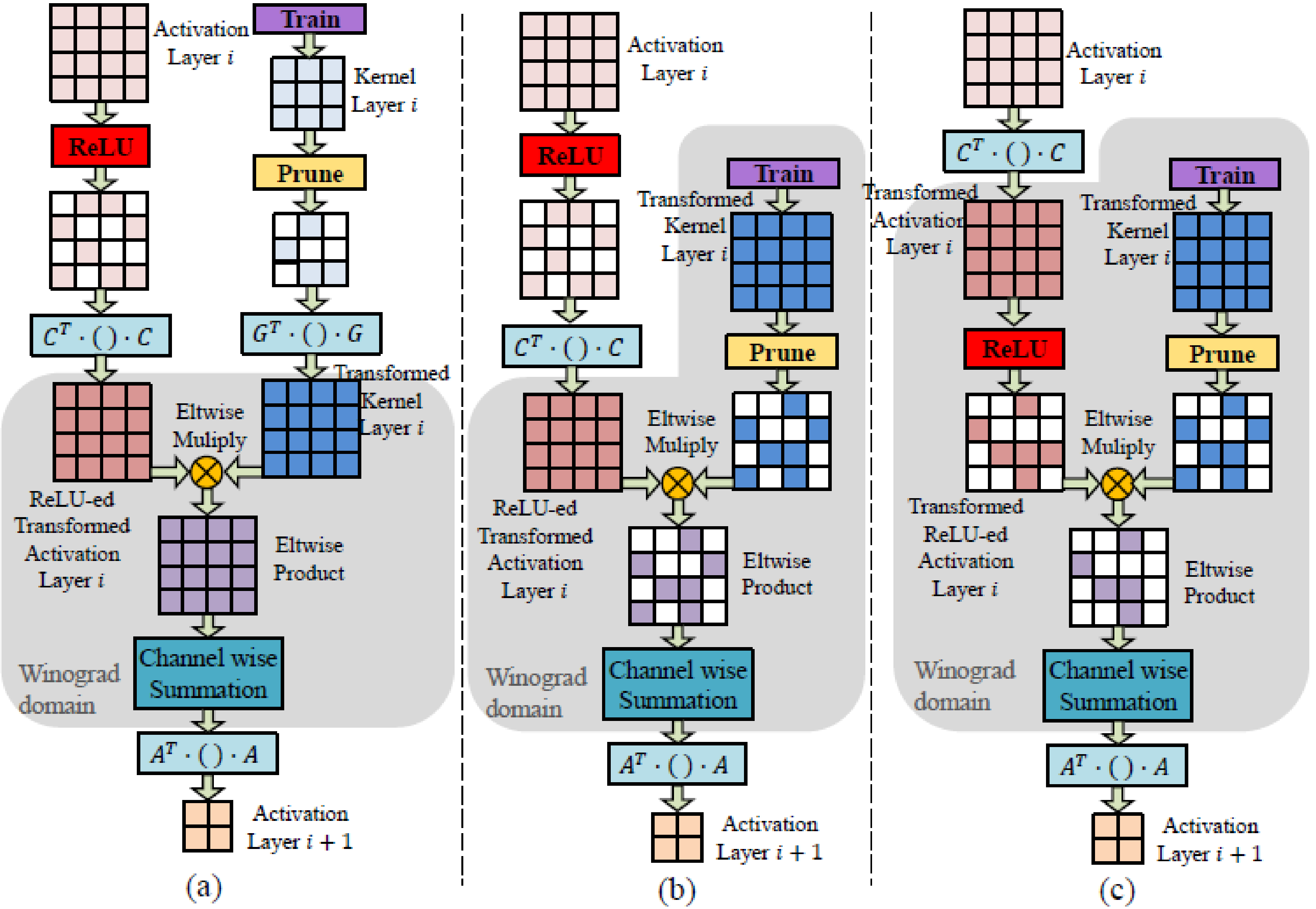
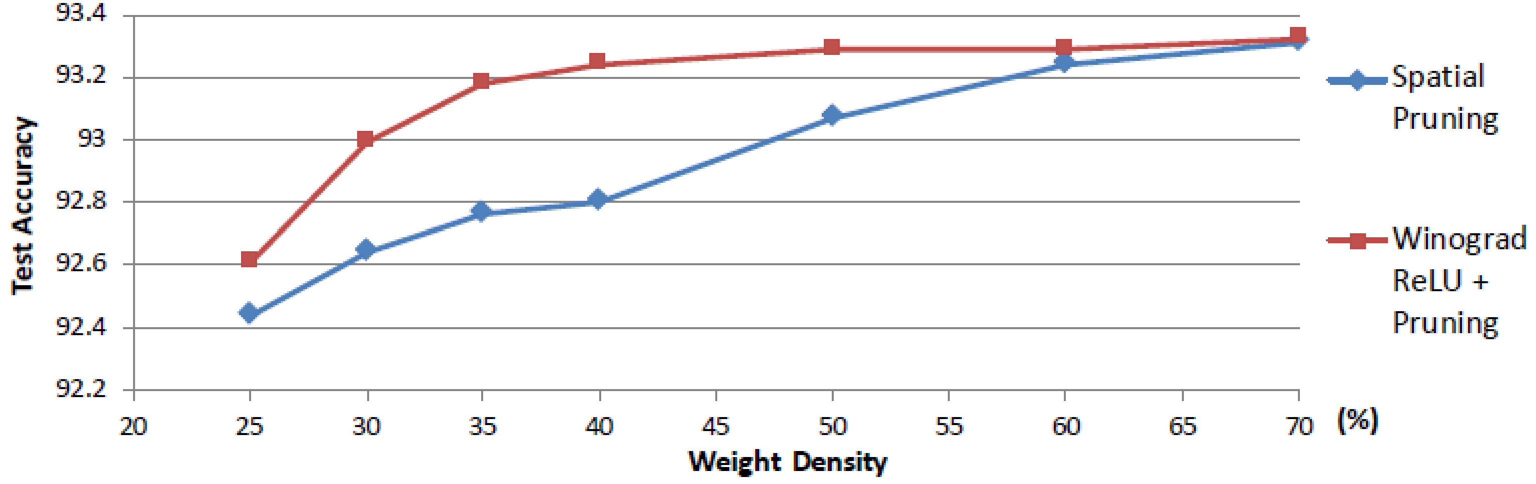
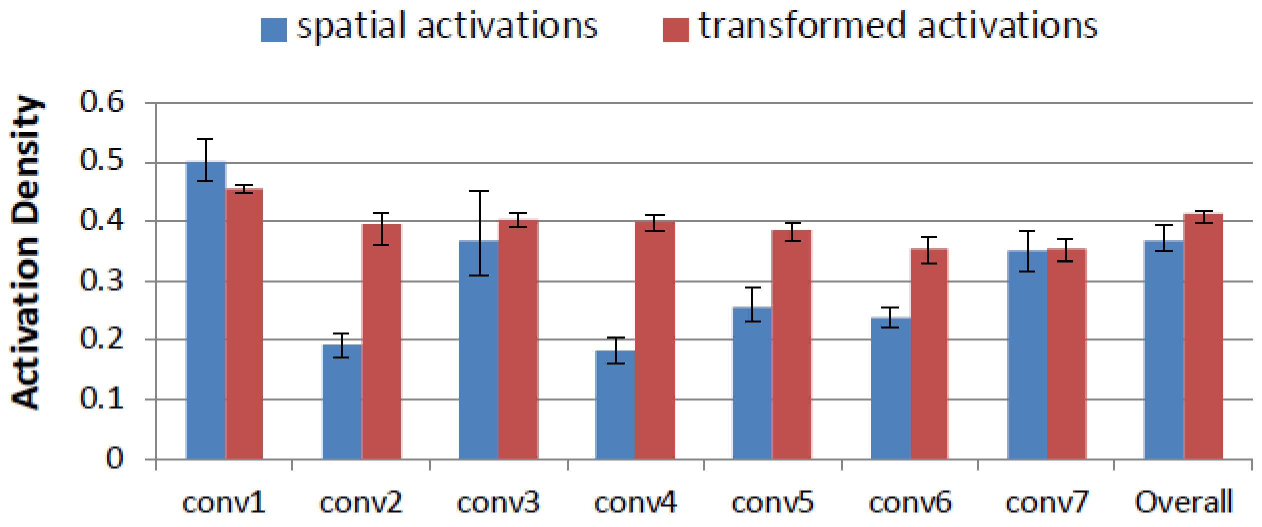


Figure 1: Combining Winograd convolution with sparse weights and activations. (a) Original Winograd-based convolution proposed by Lavin (2015) fills in the non-zeros in both the weights and activations. (b) Pruning the 4 × 4 transformed kernel restores sparsity to the weights. (c) Moving the ReLU layer after Winograd transformation also restores sparsity to the activations.

Winograd Convolution + Sparse

pruned to 40% density without significant (> 0:1%) loss of accuracy





overall activation density of 41:1% compared to 36:9% density for the spatial activations

Compressed neural architecture utilizing dimensionality reduction and quantization