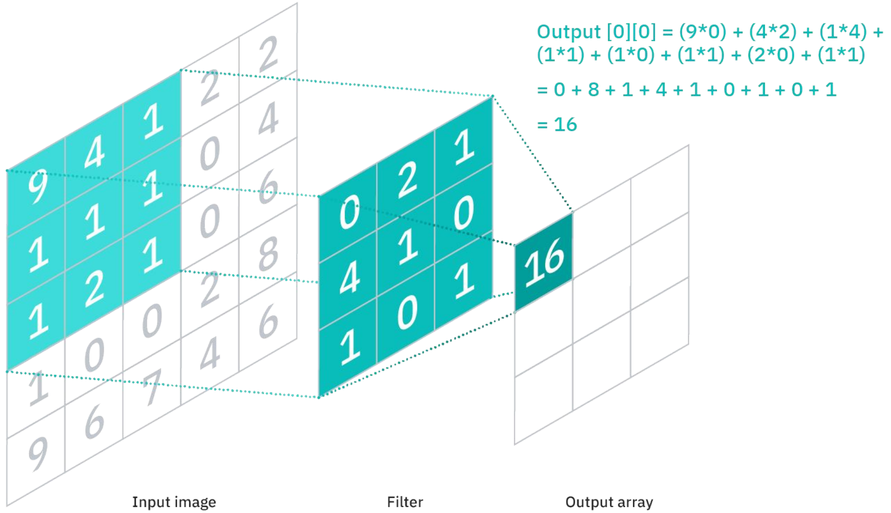
Fast Algorithms for Convolutional Neural Networks

Convolution Neural Networks

- Deep convolutional neural networks (convnets) achieve state of the art results on image recognition problems.
- The networks take several days of GPU time to train and require significant compute resources during classification as well. Larger data sets and models lead to better accuracy but also increase computation time.
- Therefore progress in deep neural networks is limited by how fast the networks can be computed.


Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16
Input image
Filter
Output array

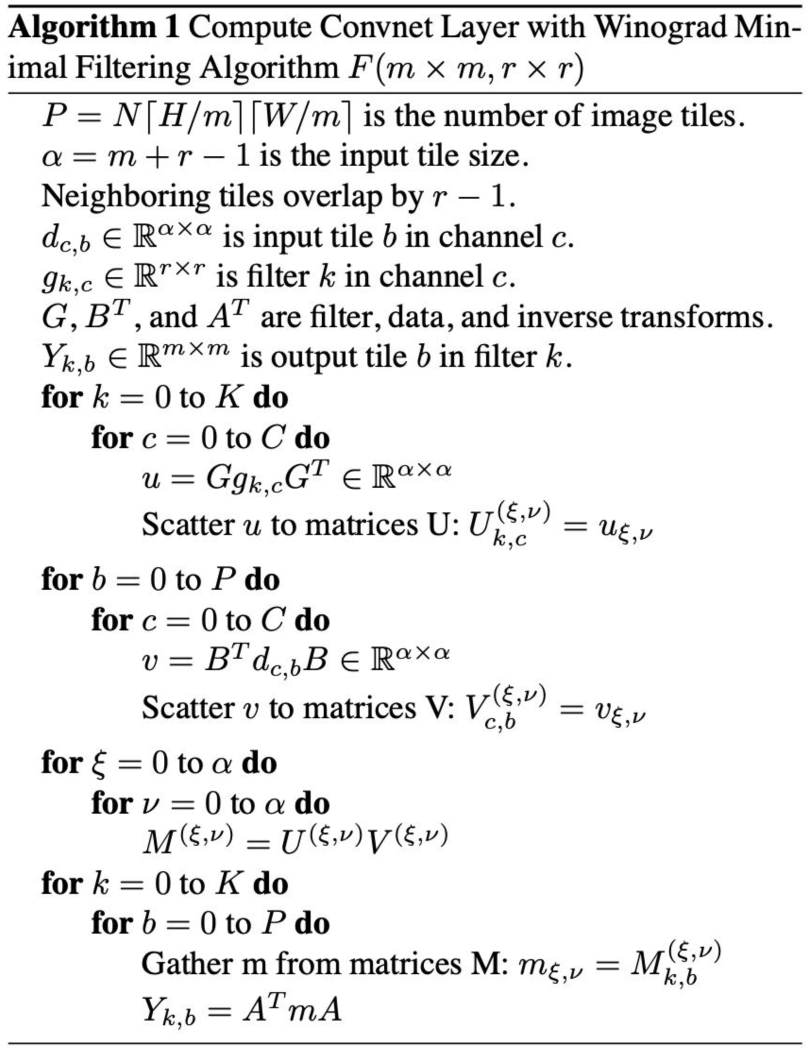
- By distributing each batch of examples among the nodes of a cluster and aggregating weight changes across the nodes, distributed training of convnets can be accomplished.

- Large batch sizes have a negative impact on network convergence, hence the smallest batch size that can be calculated effectively sets an upper limit on cluster size.
- Deep networks with 3 x 3 convolutional layers are used in state-of-the-art convnet designs for image recognition because they provide greater accuracy with fewer weights than shallow networks with bigger filters.
- As a result, fast convnet algorithms for small batch sizes and small filters are in high demand. Traditional convnet libraries, on the other hand, require big batch sizes and massive filters to operate quickly.

Fast Algorithms

- The efficiency of an Algorithm can be evaluated by the amount of computational resources it takes to compute output.
- We can use these algorithms for any dimensional arrays by just cascading the 1-D algorithm.
- Using Winograd's minimal filtering techniques, we present a new class of fast algorithms for convolutional neural networks to achieve efficiency in multiplication of matrices performed by each thread.

Winograd Algorithm



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 $\alpha$ **do**
        **for** $\nu = 0$ to $\alpha$ **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: $m_{\xi,\nu} = M_{k,b}^{(\xi,\nu)}$
            $Y_{k,b} = A^T m A$
```

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`

Experimental Observations

- 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.

Team

Medidoddi Vahini	Jaswanth Dharmana
Naman Jain	R. S. S. V. Prakash
Dhrumil Kavathia	Uppada Hemanth Kumar
Aayushi Vidyanta	Raj Gupta
Patel Devarshi Chandrakant	Kunal Katiyar