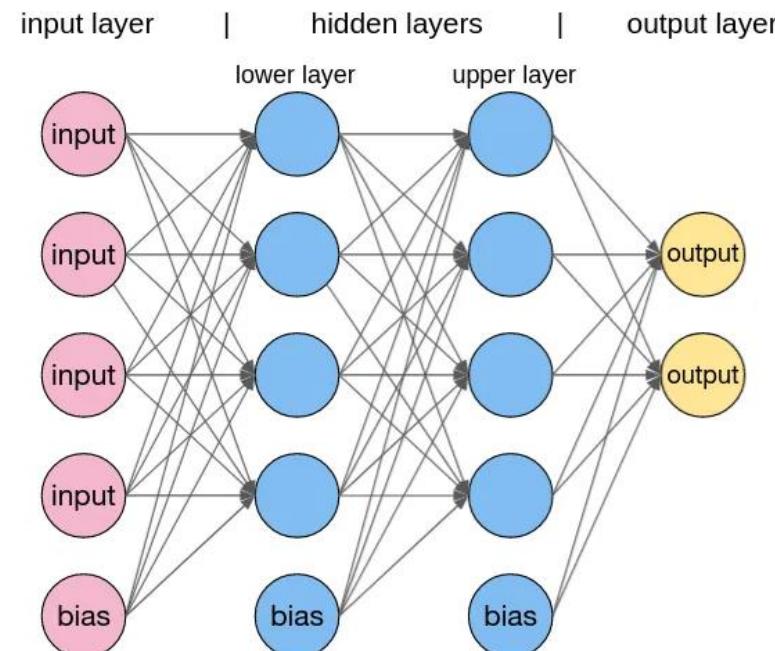
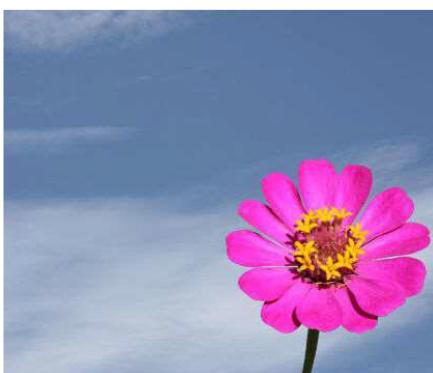


Convolution Neural Network Architecture

DEEP LEARNING COURSE
E. FATEMIZADEH
FALL 2023

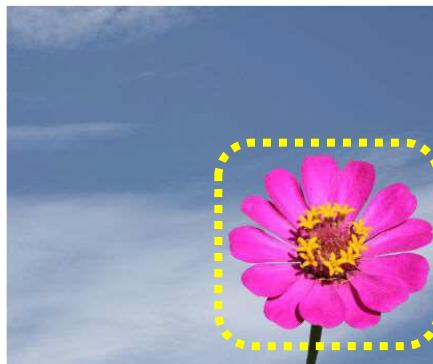
A Problem

- Can a MLP identify two same types of flowers in these images?



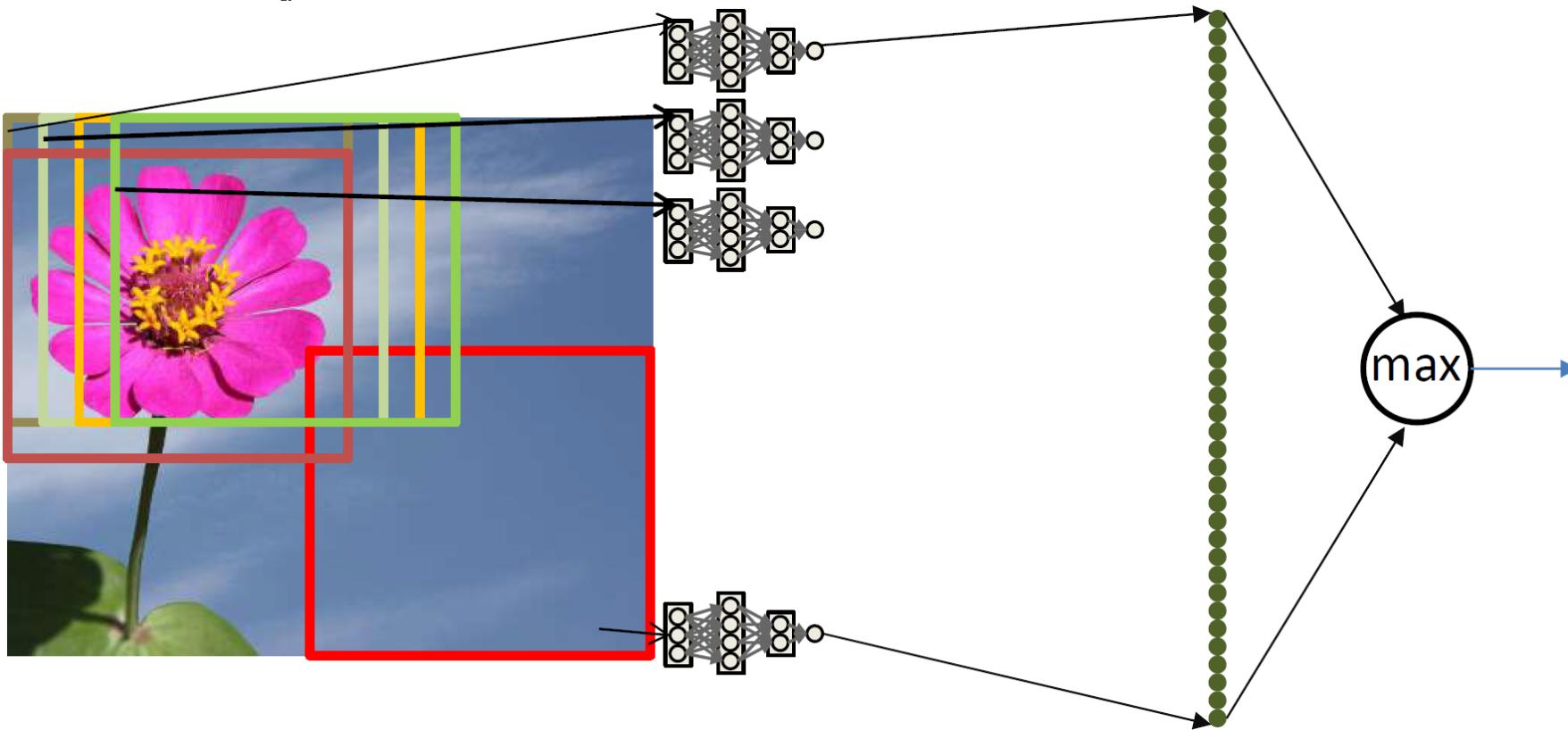
A Problem

- We need a network that will activate, regardless of the exact location of the desired object?
- We need shift invariance.



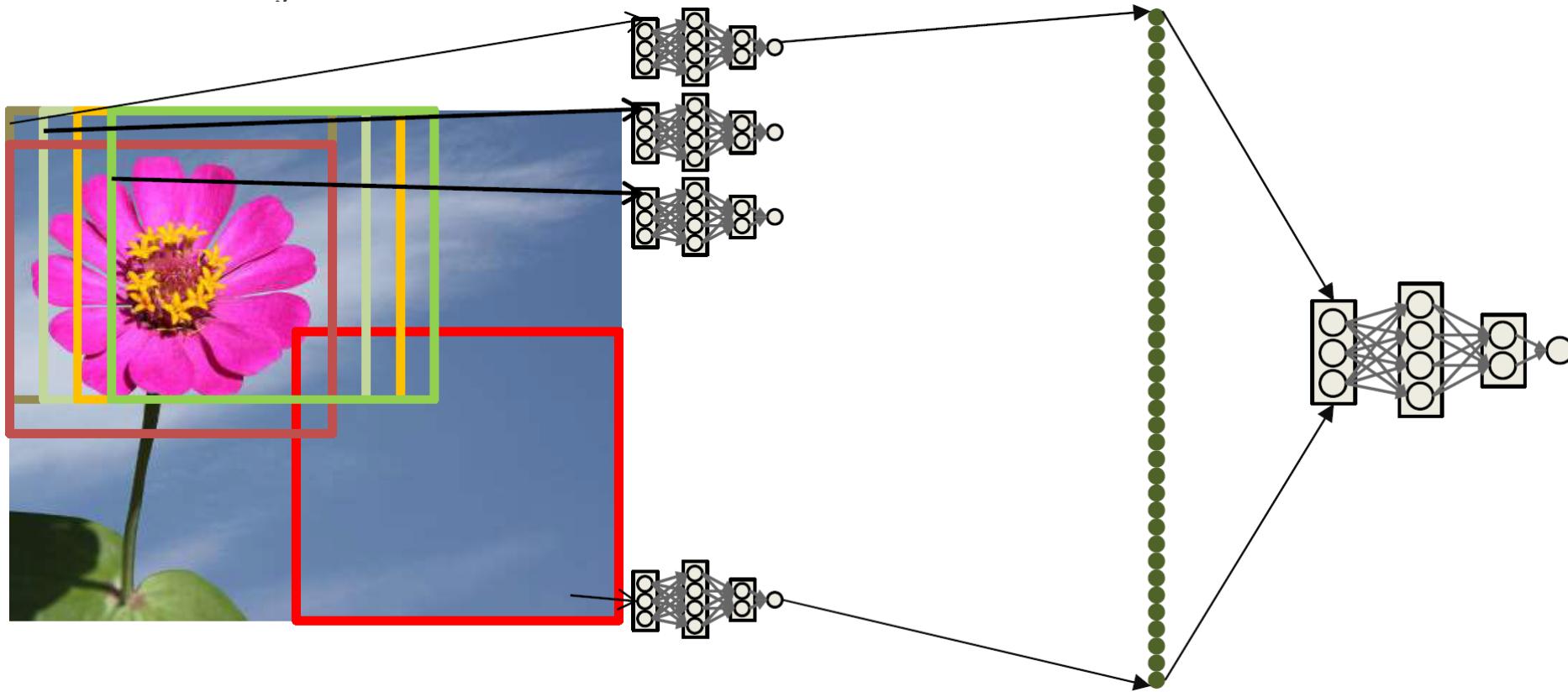
Solution - Scanner

- Scan for desired object



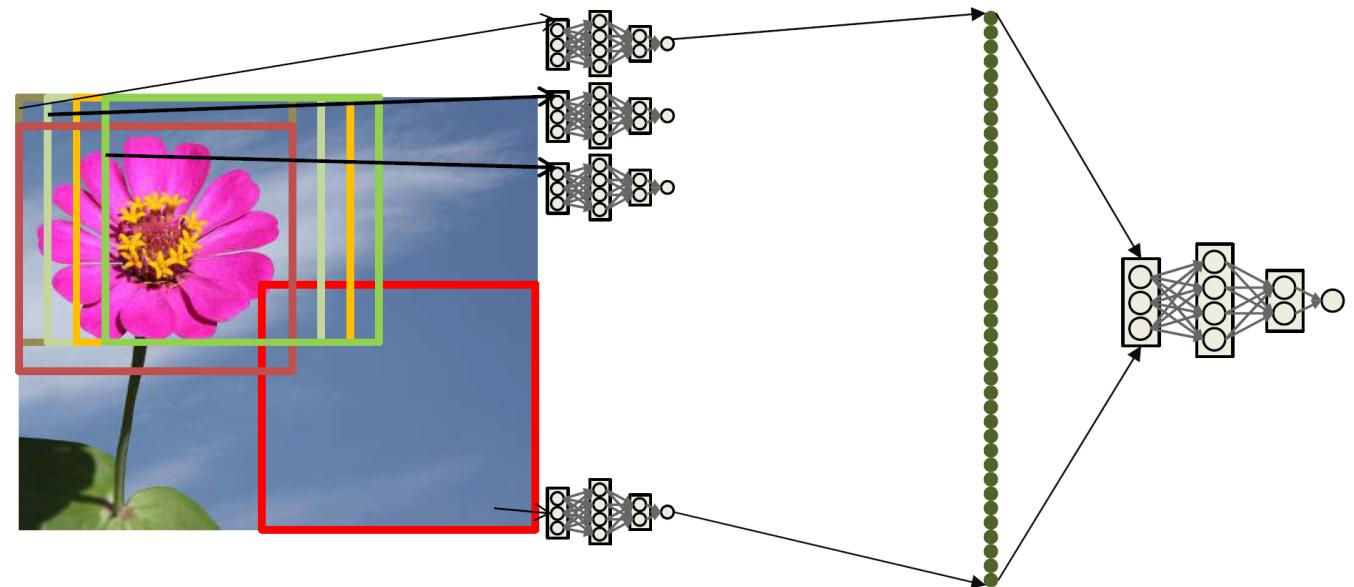
Solution - Scanner

- Scan for desired object



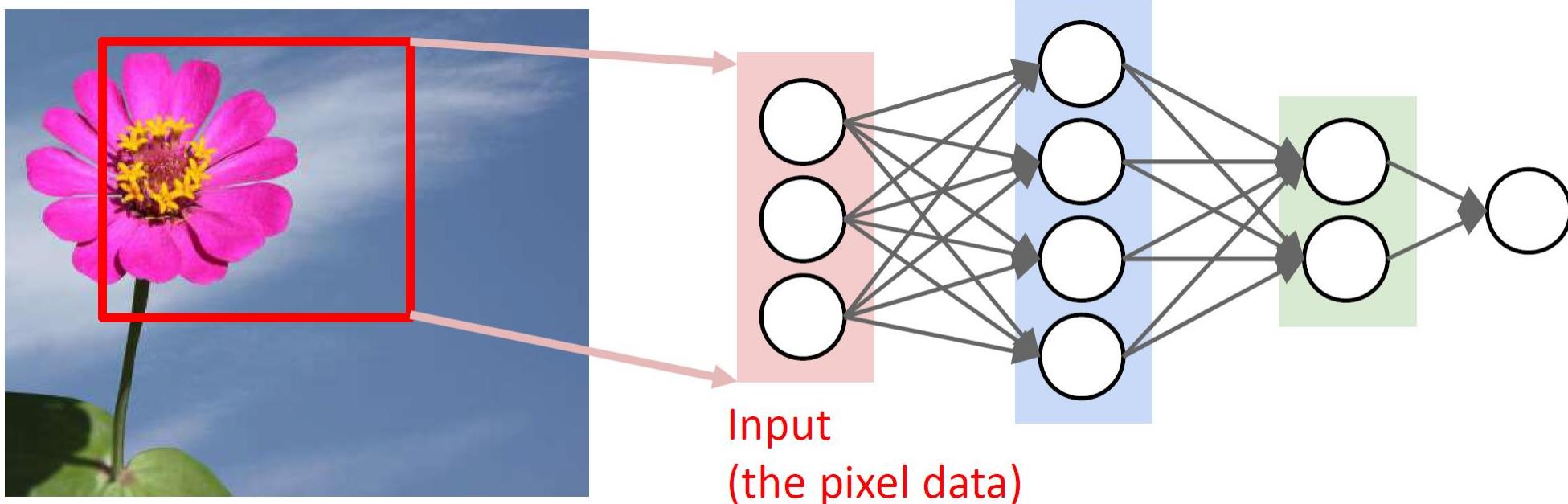
Solution - Scanner

- The entire operation can be viewed as a single giant network
 - Composed of many “subnets” (one per window)
 - With one **key feature**: all subnets are **identical**
 - These are shared parameter networks



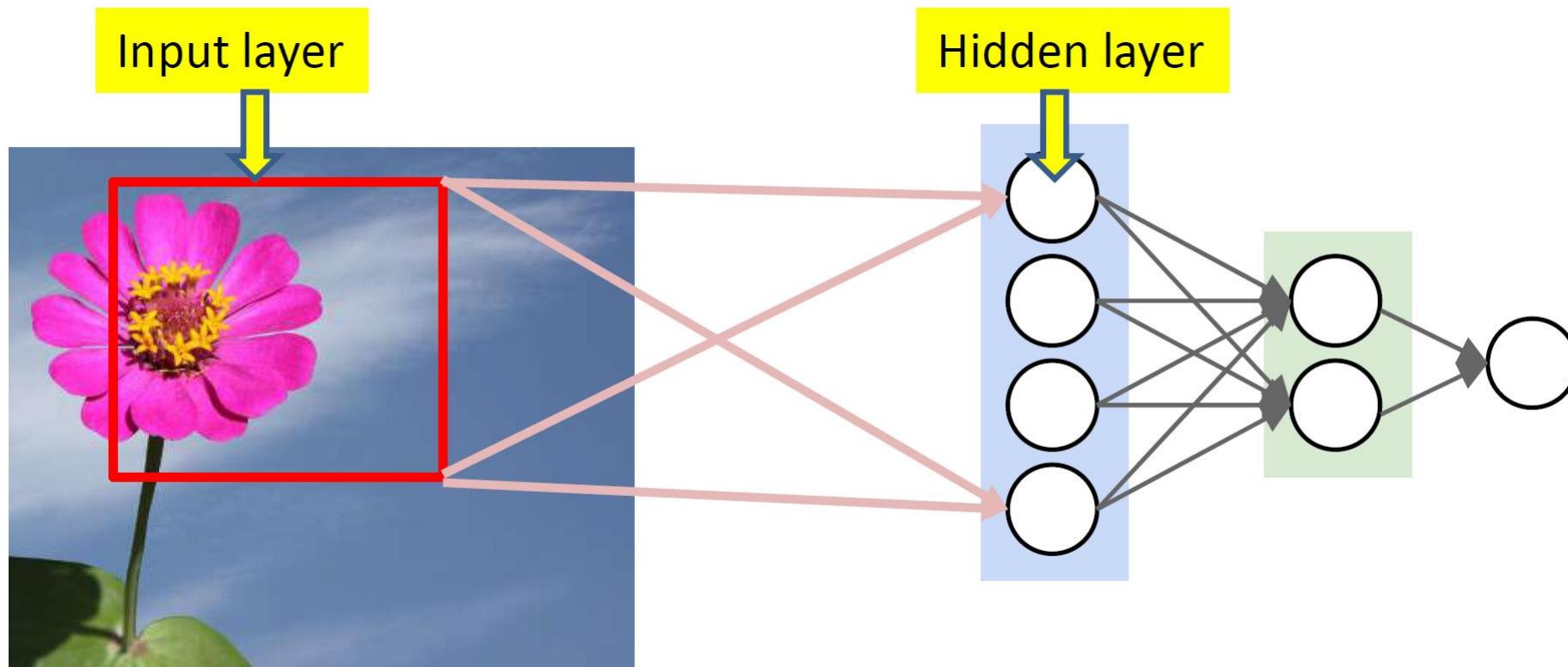
Two dimension scanning

- Consider this schematic:



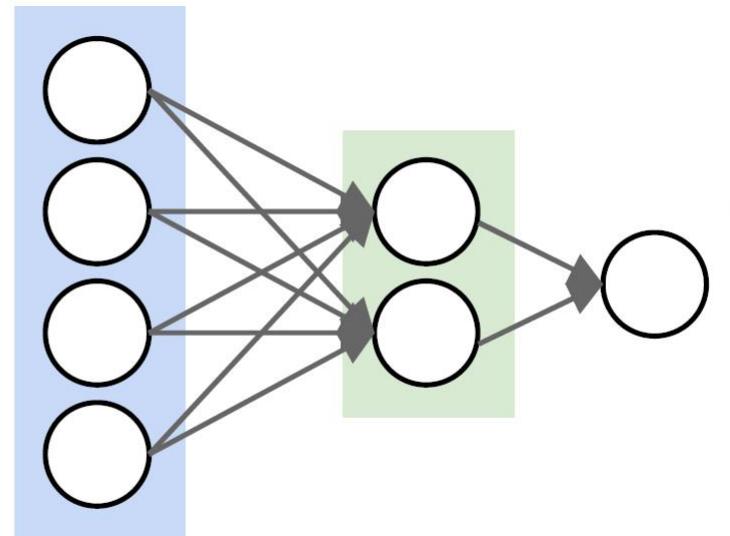
Two dimension scanning

- Consider this schematic:



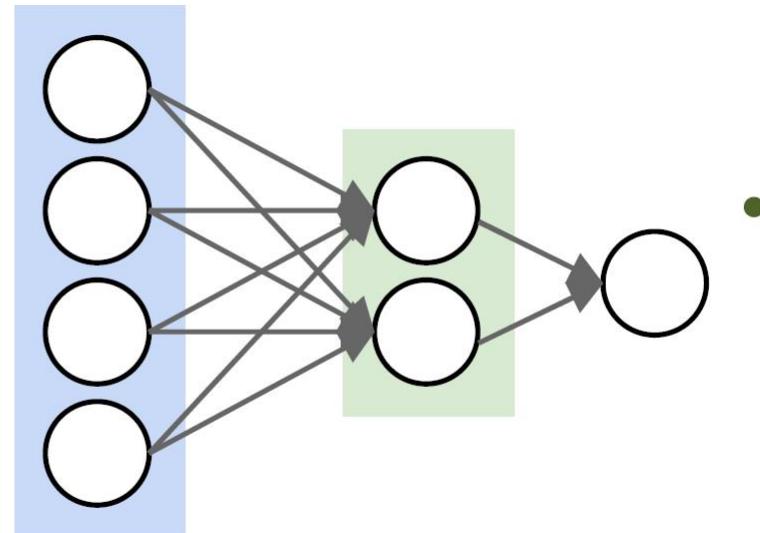
Two dimension scanning

- How feature map shape



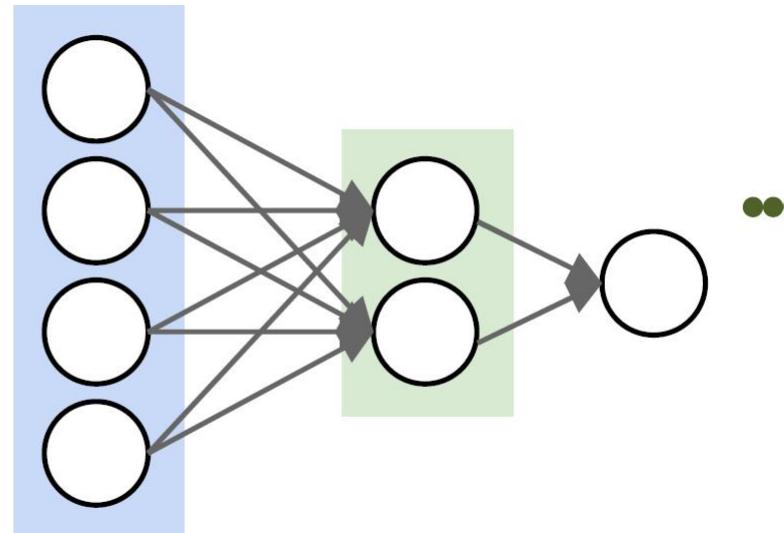
Two dimension scanning

- Consider this schematic:



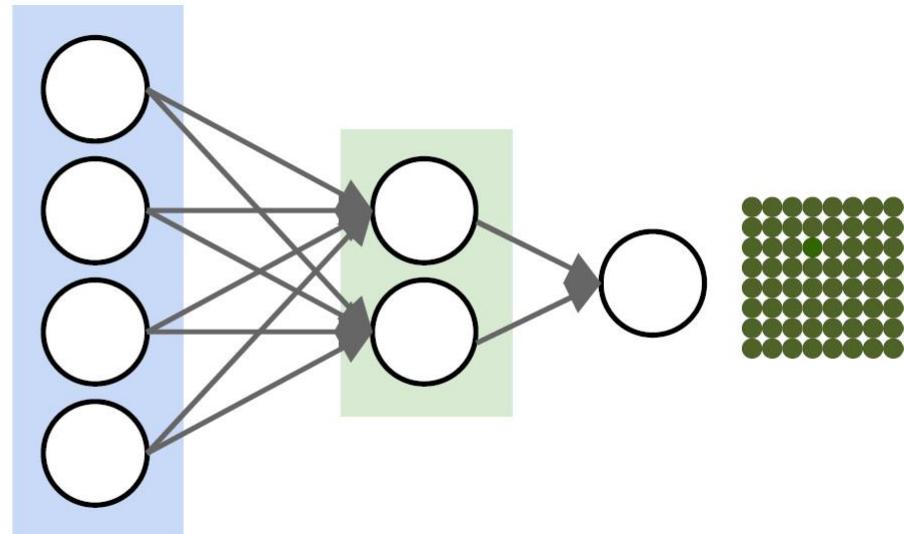
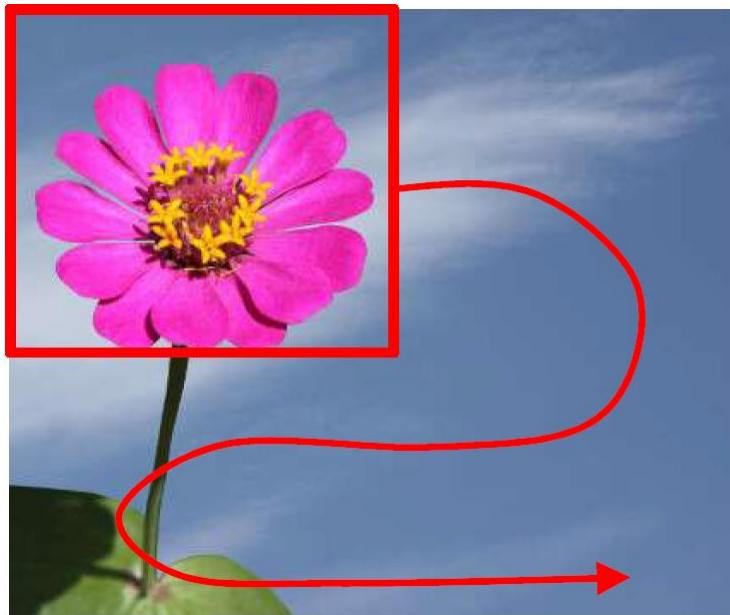
Two dimension scanning

- Consider this schematic:



Two dimension scanning

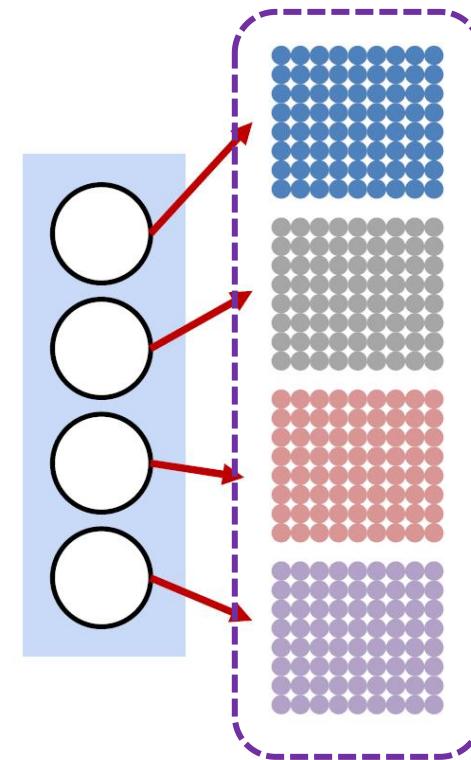
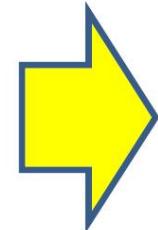
- Consider this schematic:



- Produce an output for every window analyzed → Pass collection of outputs through a softmax

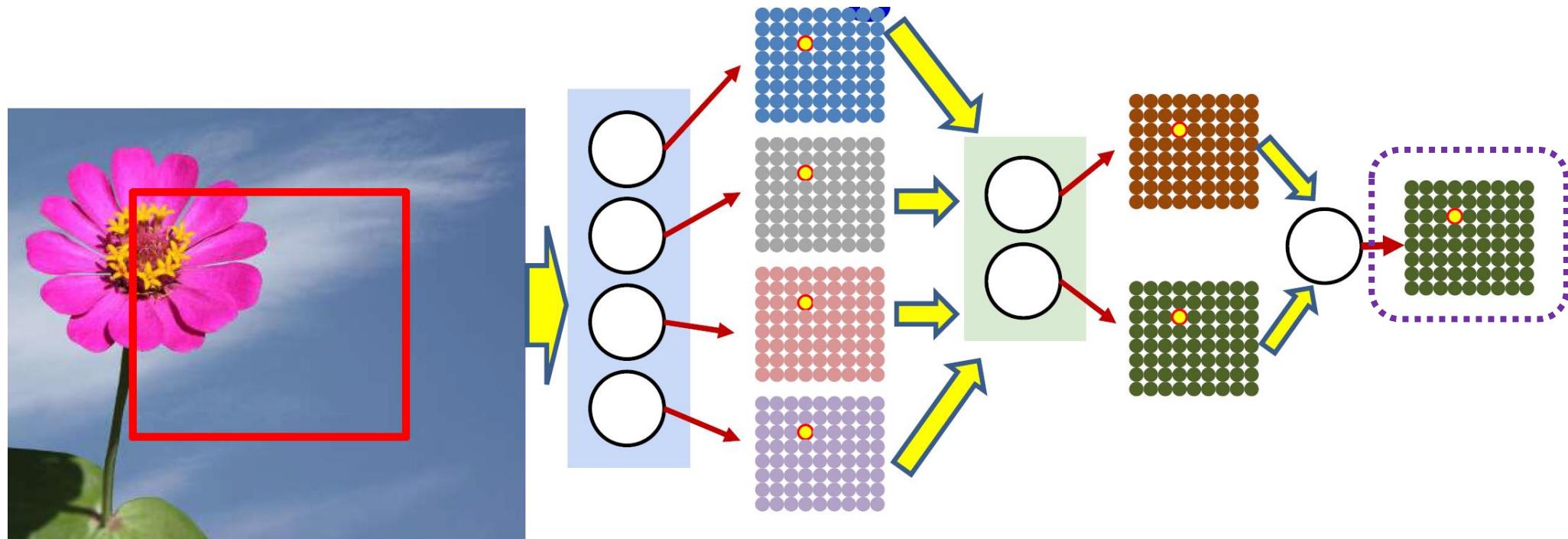
Two dimension scanning – Channel!

- Look at individual hidden neuron:



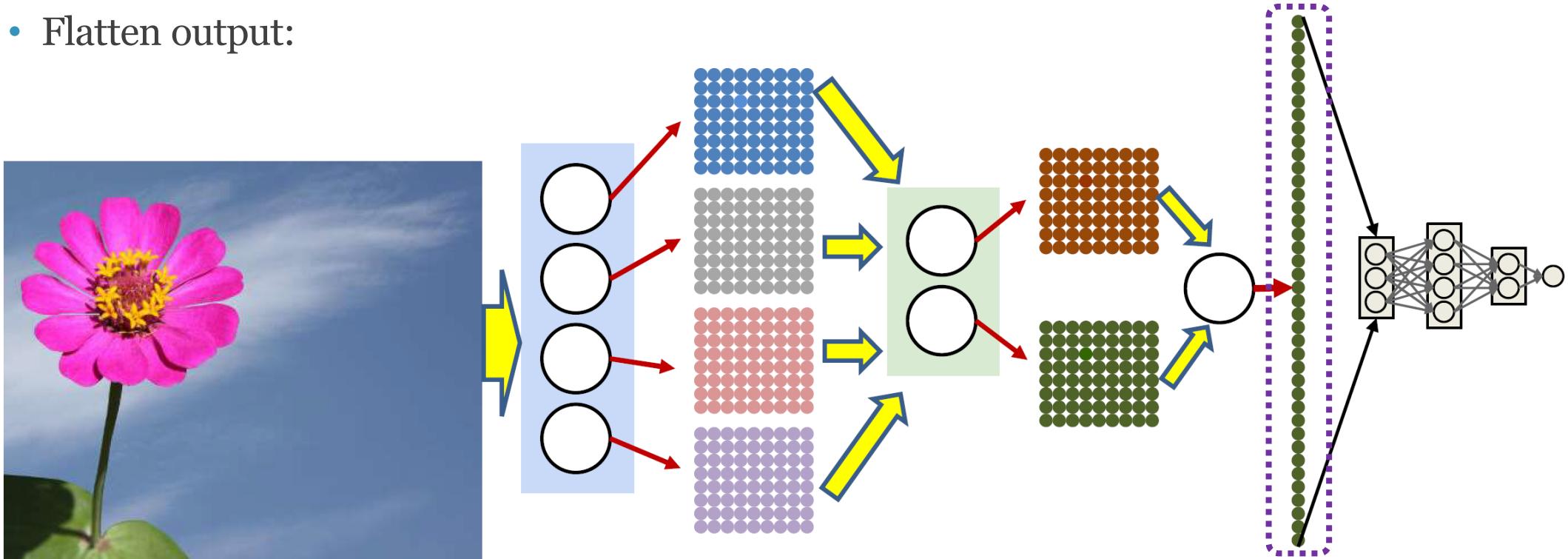
Two dimension scanning

- Look at hidden layers:



Two dimension scanning

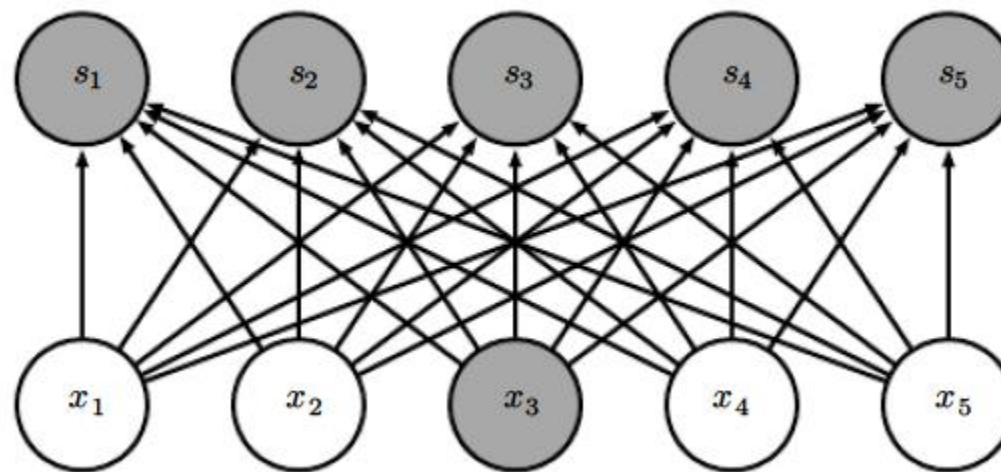
- Flatten output:



From FCN to CNN

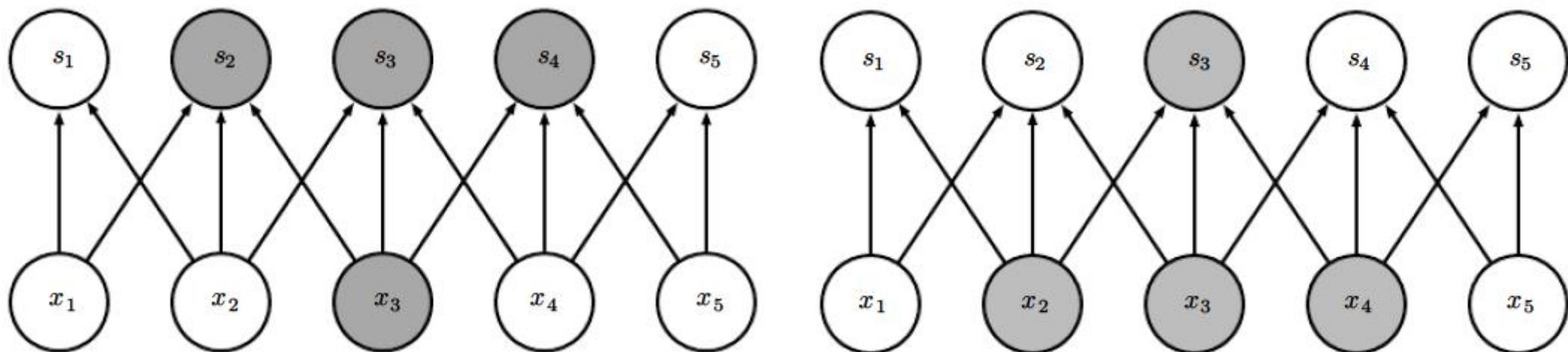
- Fully Connected (FC) or Dense simple matrix representation

$$S = \text{ReLU}(\mathbf{W}\mathbf{X}), w_{ij}'s \text{ are arbitrary}$$



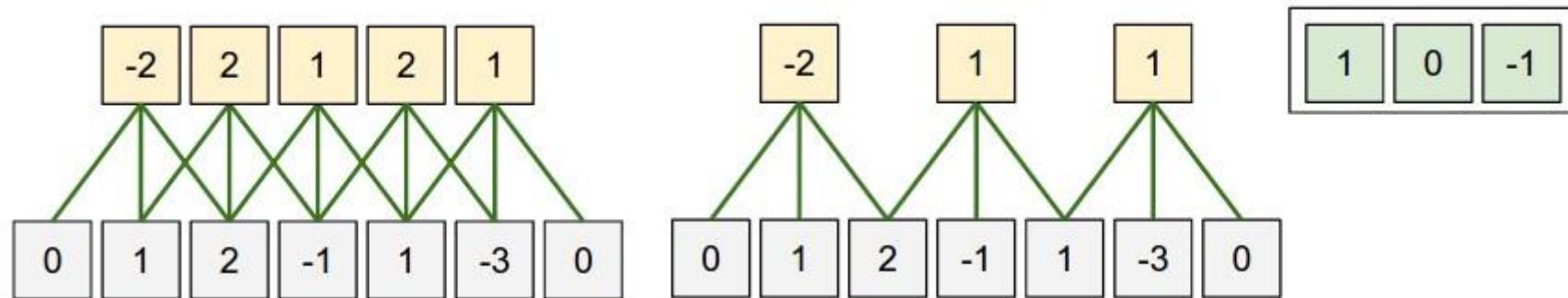
How to reduce number of parameters?

- Sparse Connection/interaction



How to reduce number of parameters?

- Parameter Sharing (plus Stride):



Convolution/Correlation

- Linear Convolution (Linear Shift Invariant Systems)

$$y[n] = \sum_m x[m] h[n - m] = \sum_k h[k] x[n - k]$$

- Correlation:

$$y[n] = \sum_m x[m] h[n + m] = \sum_k h[k] x[n + k]$$

Convolution/Correlation

- Sparse Connectivity:
 - Yes, short length kernel (FIR) , $h[n]$
- Parameter Sharing:
 - Yes, Linear Shift Invariant, a single $h[n]$ and NOT $h[n, m]!$
- Equivariant Representations:
 - Linearity and Shift Invariant! $f(g(x)) = g(f(x))$, some certain g – operator!

Matrix-Multiplication Representation

- Linear Convolution, Toeplitz Matrix:
 - Output size: full, same, valid

$$\begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ y[3] \\ y[4] \\ y[5] \end{bmatrix} = \begin{bmatrix} h[0] & 0 & 0 \\ h[1] & h[0] & 0 \\ h[2] & h[1] & h[0] \\ h[3] & h[2] & h[1] \\ 0 & h[3] & h[2] \\ 0 & 0 & h[3] \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \end{bmatrix}$$
$$Toeplitz(h) = \begin{bmatrix} h[0] & 0 & 0 \\ h[1] & h[0] & 0 \\ h[2] & h[1] & h[0] \\ h[3] & h[2] & h[1] \\ 0 & h[3] & h[2] \\ 0 & 0 & h[3] \end{bmatrix}$$

Numerical Example

- Linear Convolution:

$$x[n] = \{1, 2, 2\}, h[n] = \{1, -1\}, N_1 = 3, N_2 = 2$$

- The matrix representation:

$$\mathbf{y} = \mathbf{Hf} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ -2 \end{bmatrix}$$

- Signal representation:

$$y[n] = \{1, 1, 0, -2\}$$

Numerical Example

- Circular Convolution, Circulant Matrix:

$$C = \begin{pmatrix} c_0 & c_1 & c_2 & \cdots & c_{n-1} \\ c_{n-1} & c_0 & c_1 & c_2 & \cdots \\ c_{n-2} & c_{n-1} & c_1 & \cdots & \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ c_1 & c_2 & \cdots & c_{n-1} & c_0 \end{pmatrix}$$

Numerical Example

- Circular Convolution:

$$x[n] = \{1, 2, 2\}, h[n] = \{1, -1\}, N_1 = 3, N_2 = 2$$

- The matrix representation:

$$\mathbf{y} = \mathbf{Hf} = \begin{bmatrix} 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$

- Signal representation:

$$y[n] = \{1, 1, 0\}$$

2D Convolution and Correlation

- Convolution:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n).$$

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n).$$

- Correlation:

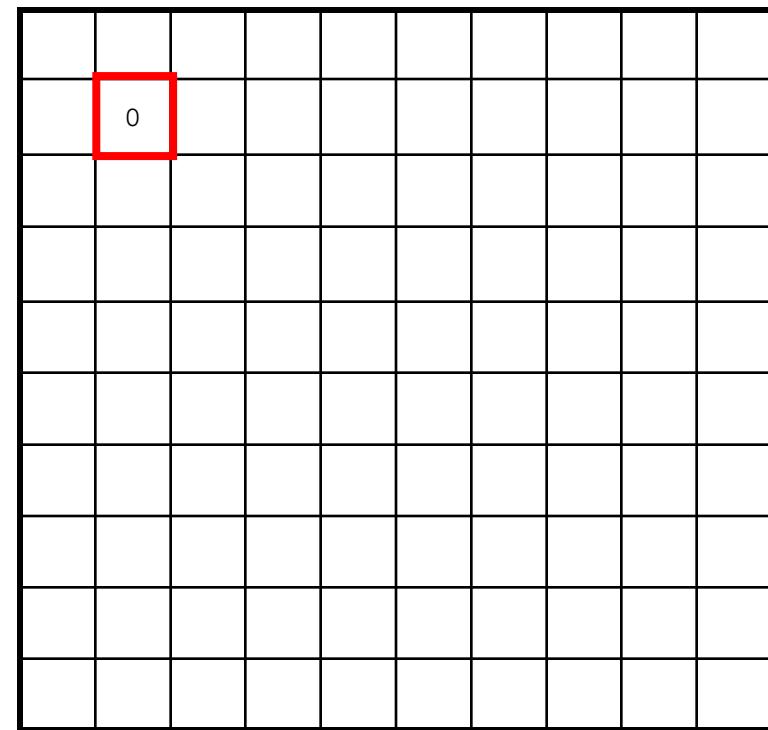
$$S(i, j) = (I \odot K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n).$$

Image/Matrix Filtering

- Step #1:

Valid

| | | | | | | | | | | | |
|---|---|----|----|----|----|----|----|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

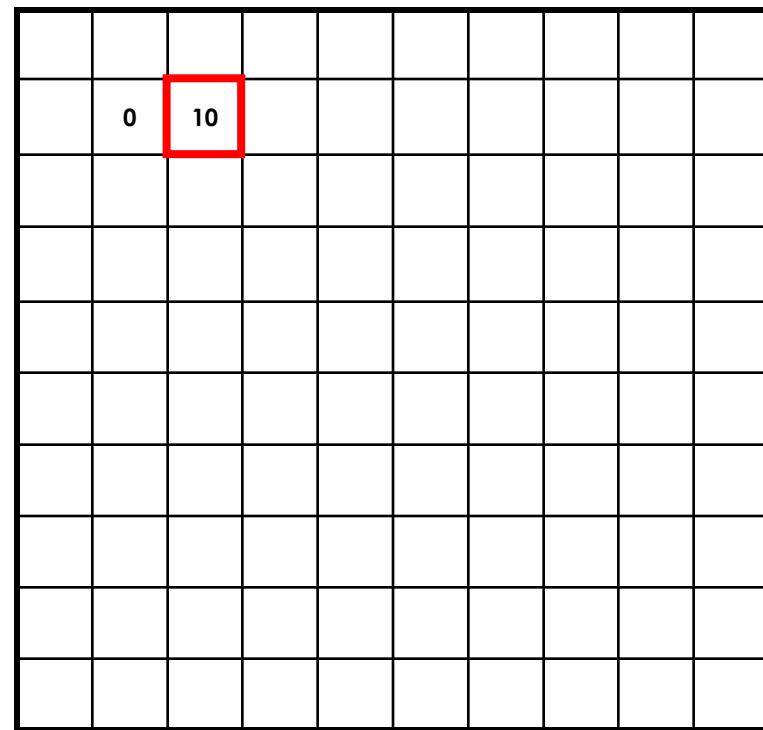


$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Image/Matrix Filtering

- Step #2

| | | | | | | | | | | |
|---|---|----|----|----|----|----|----|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Image/Matrix Filtering

- Step #3

| | | | | | | | | | | |
|---|---|----|----|----|----|----|----|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | | | |
|---|----|----|--|--|--|--|--|--|--|--|
| | | | | | | | | | | |
| | | | | | | | | | | |
| 0 | 10 | 20 | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Image/Matrix Filtering

- Step #4

| | | | | | | | | | | |
|---|---|----|----|----|----|----|----|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | | | |
|---|----|----|----|--|--|--|--|--|--|--|
| | | | | | | | | | | |
| | | | | | | | | | | |
| 0 | 10 | 20 | 30 | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Image/Matrix Filtering

- Step #5

| | | | | | | | | | | |
|---|---|----|----|----|----|----|----|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | | | |
|--|---|----|----|----|----|--|--|--|--|--|
| | | | | | | | | | | |
| | 0 | 10 | 20 | 30 | 30 | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Image/Matrix Filtering

- Step #6

| | | | | | | | | | | |
|---|---|----|----|----|----|----|----|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

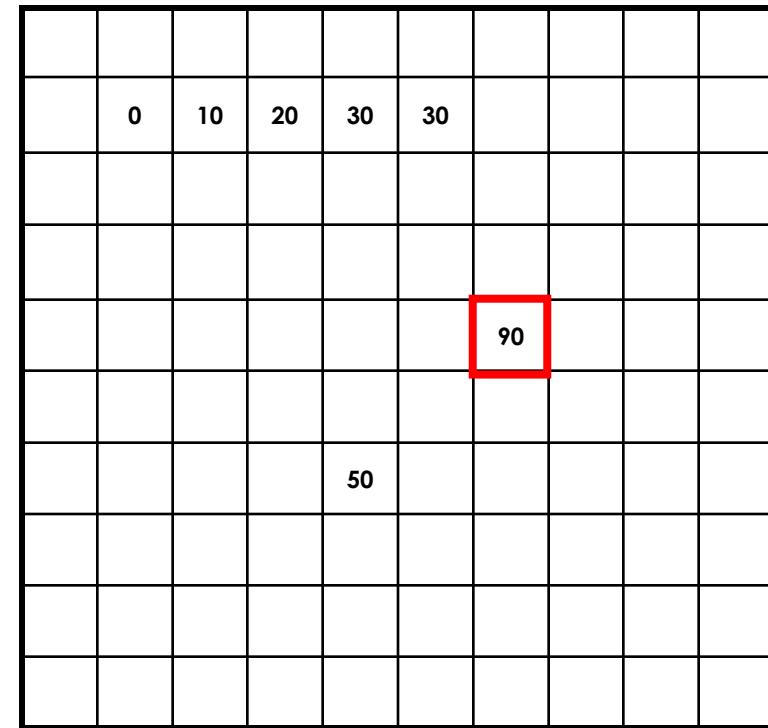
| | | | | | | | | | | |
|--|---|----|----|----|----|--|--|--|--|--|
| | | | | | | | | | | |
| | 0 | 10 | 20 | 30 | 30 | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Image/Matrix Filtering

- Step #7

| | | | | | | | | | | |
|---|---|----|----|----|----|----|----|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Image/Matrix Filtering

- Final Results

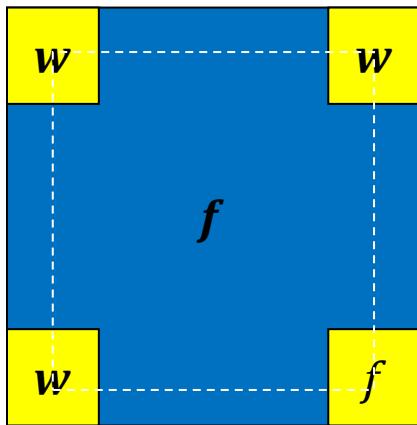
| | | | | | | | | | | |
|---|---|----|----|----|----|----|----|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|--|--|
| | | | | | | | | | | |
| | 0 | 10 | 20 | 30 | 30 | 30 | 20 | 10 | | |
| | 0 | 20 | 40 | 60 | 60 | 60 | 40 | 20 | | |
| | 0 | 30 | 60 | 90 | 90 | 90 | 60 | 30 | | |
| | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 | | |
| | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 | | |
| | 0 | 20 | 30 | 50 | 50 | 60 | 40 | 20 | | |
| | 10 | 20 | 30 | 30 | 30 | 30 | 20 | 10 | | |
| | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | | |
| | | | | | | | | | | |

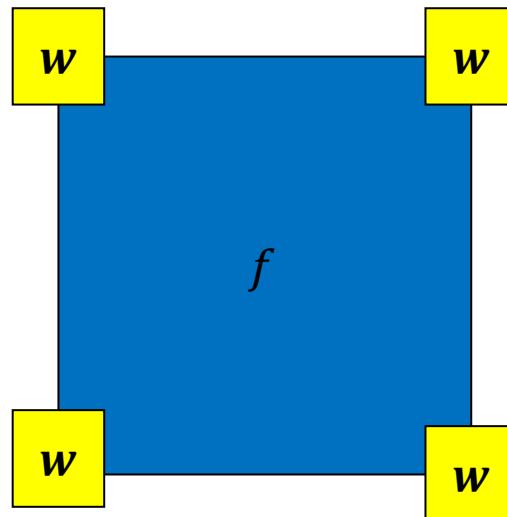
$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Practical Matters (Padding)

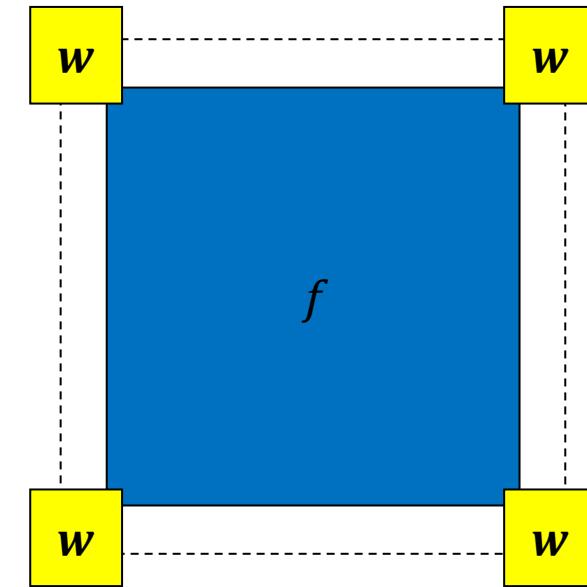
- Padding: Full, Same (Half), and Valid Scenario!



Valid (Smaller)



Same (Equal)



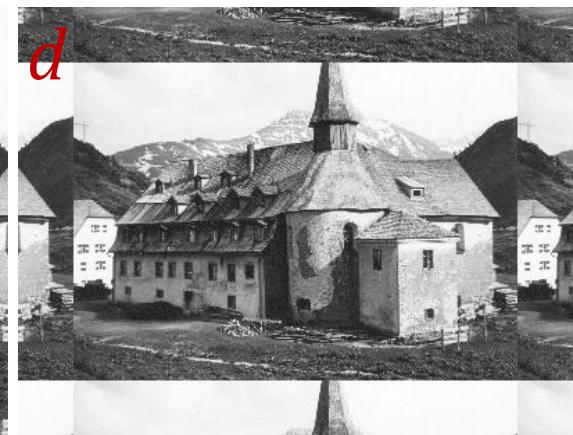
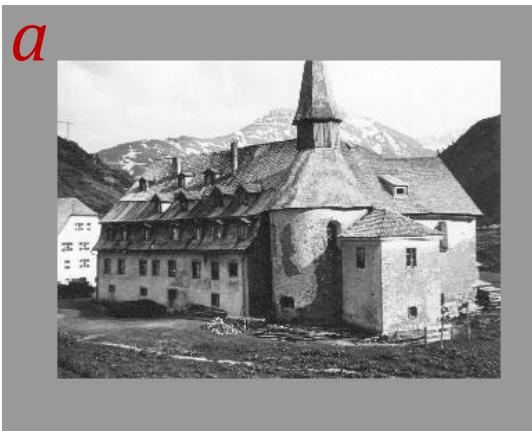
Full (Larger)

Practical Matters (Padding)

- To prevent the output from shrinking, we need to use padding:
 - Zero pad (Constant): values outside the boundary of the image are set to zero or any constant)
 - Circular (Wrap-around): values outside the boundary are computed by implicitly assuming the input is periodic.
 - Replicate (Extend): values outside the boundary are set equal to the nearest image border value.
 - Symmetric (Mirror): values outside the boundary of the image are obtained by mirror-reflecting the image across its border.
 - Kernel-Crop: Any pixel in the kernel that is outside of the input image isn't used and the normalizing is adjusted to compensate.

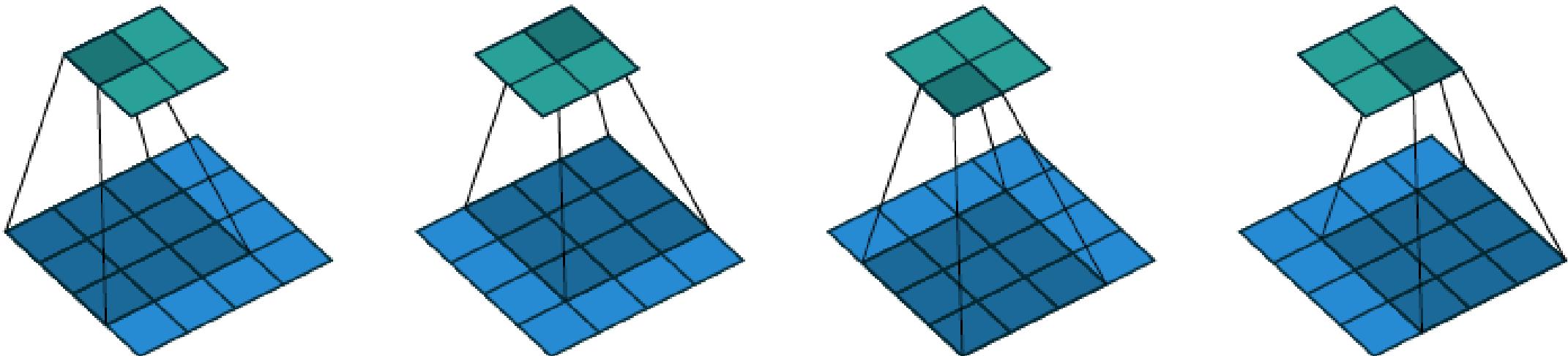
Practical Matters (Padding)

- a) Constant
- b) Replicate (nearest)
- c) Symmetric (mirror)
- d) Circular



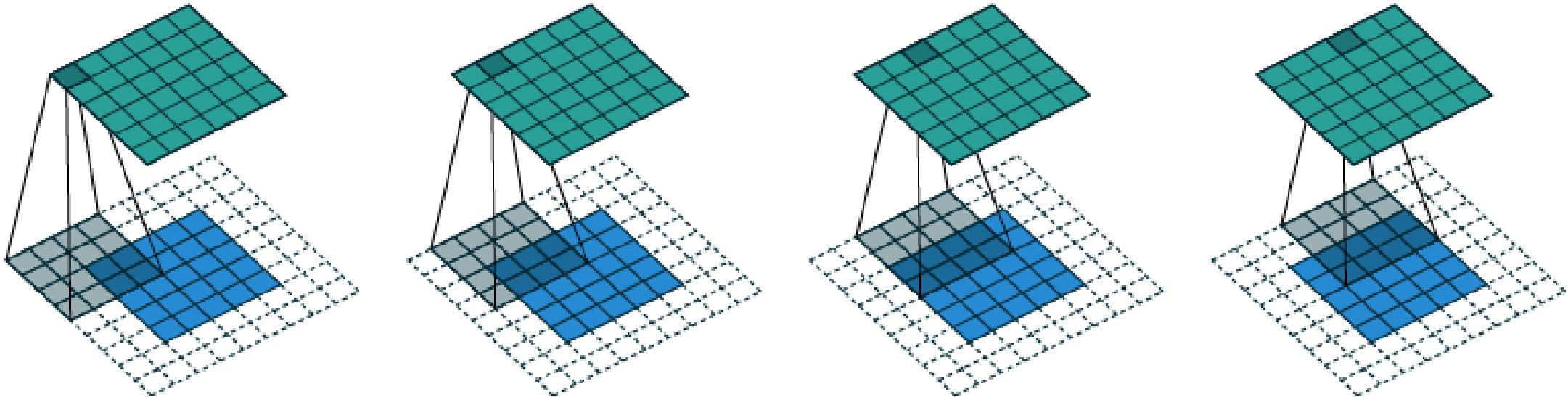
Practical matters – Padding and Stride

- No (zero) Padding (Valid) – Unit Stride



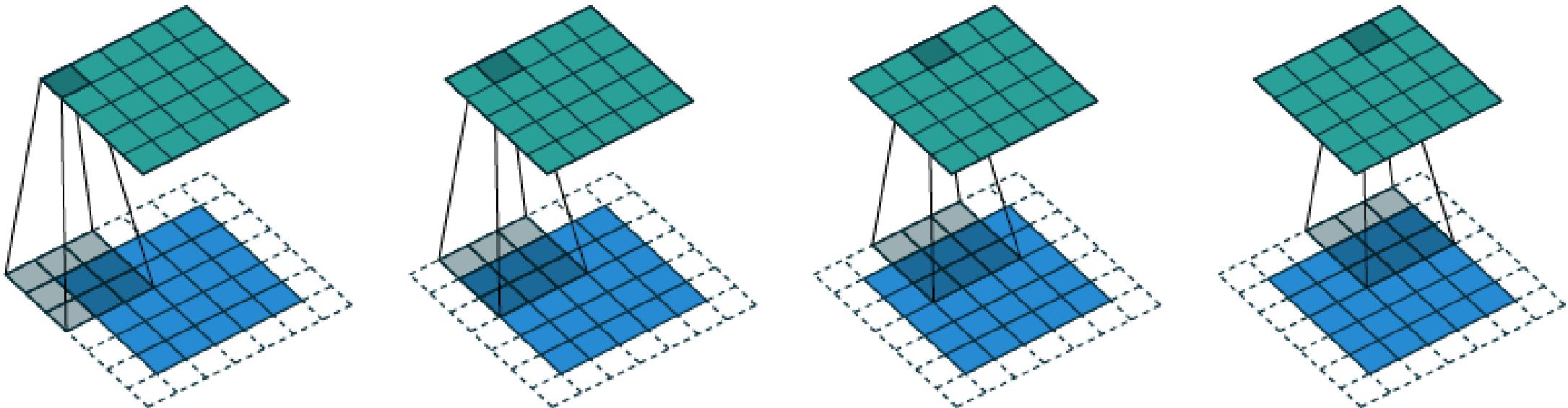
Practical matters – Padding and Stride

- Padding (2 pixel, each side) – Unit Stride



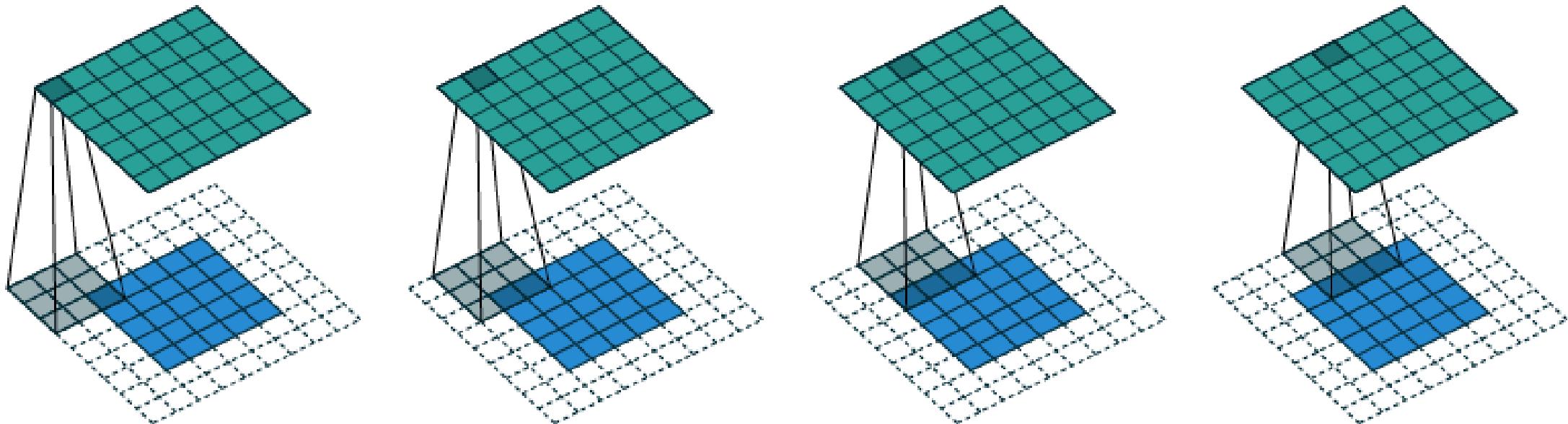
Practical matters – Padding and Stride

- Half Padding (Same) – Unit Stride



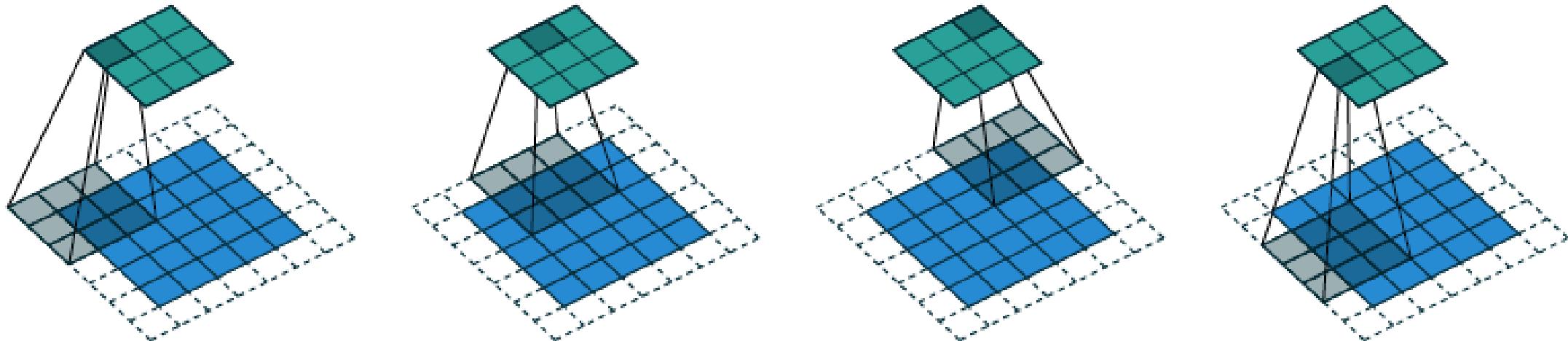
Practical matters – Padding and Stride

- Full Padding (Full) – Unit Stride



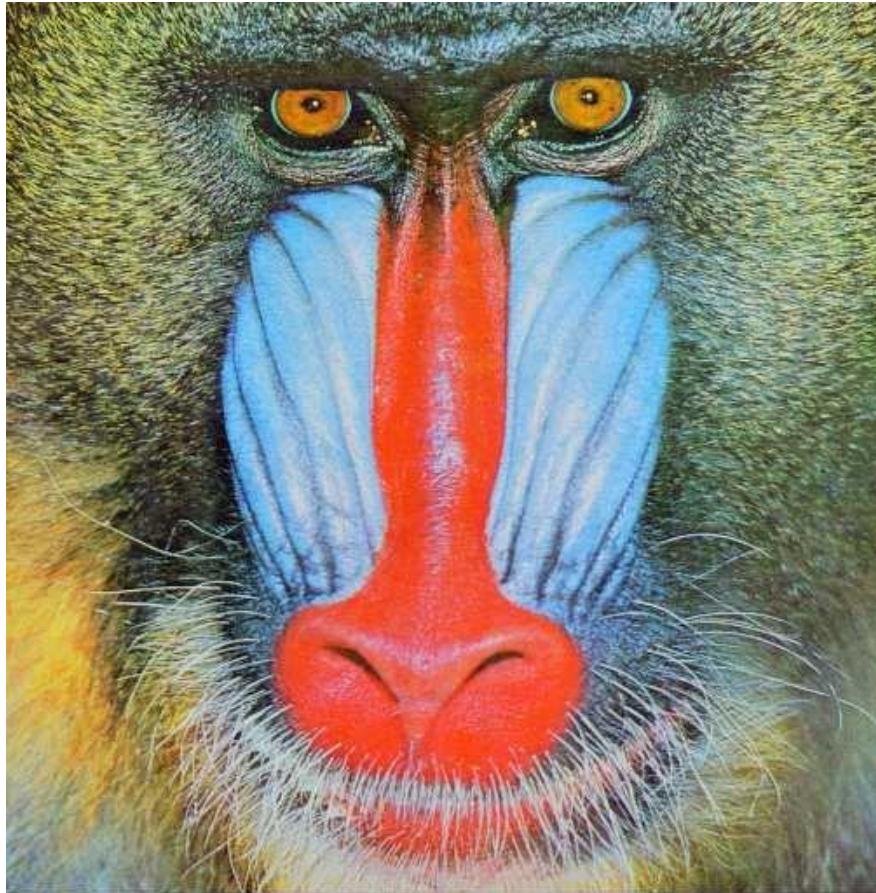
Practical matters – Padding and Stride

- Padding – 2x2 Stride (Subsampling)



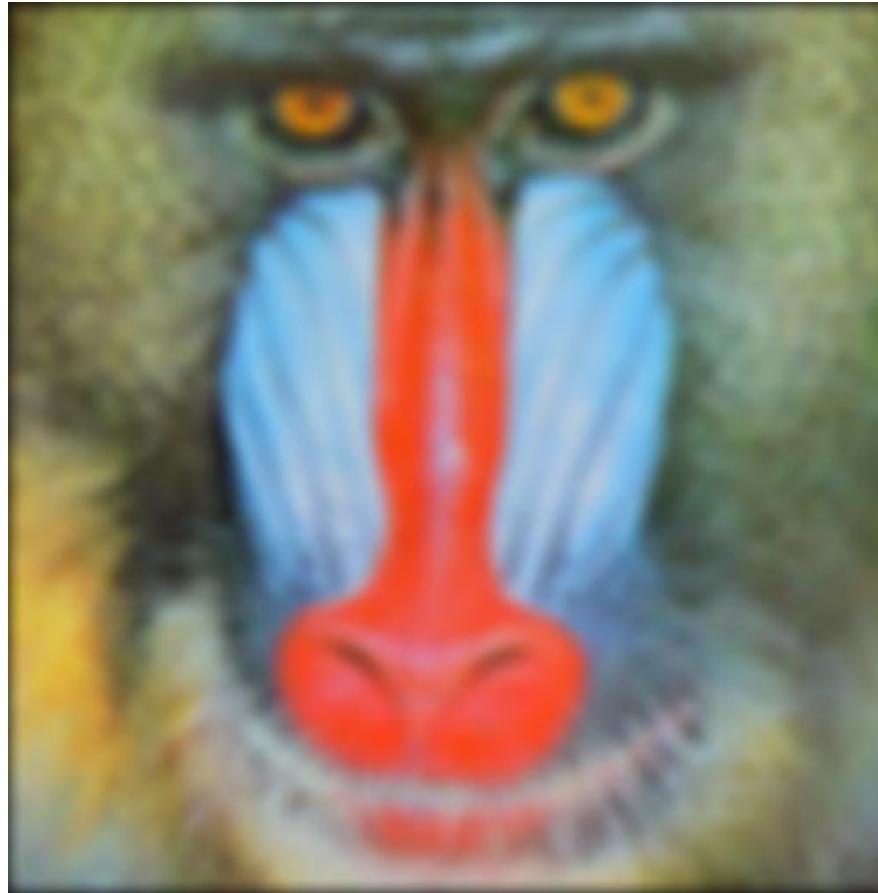
Convolution Effect!

$$h = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



Convolution Effect!

$$h = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



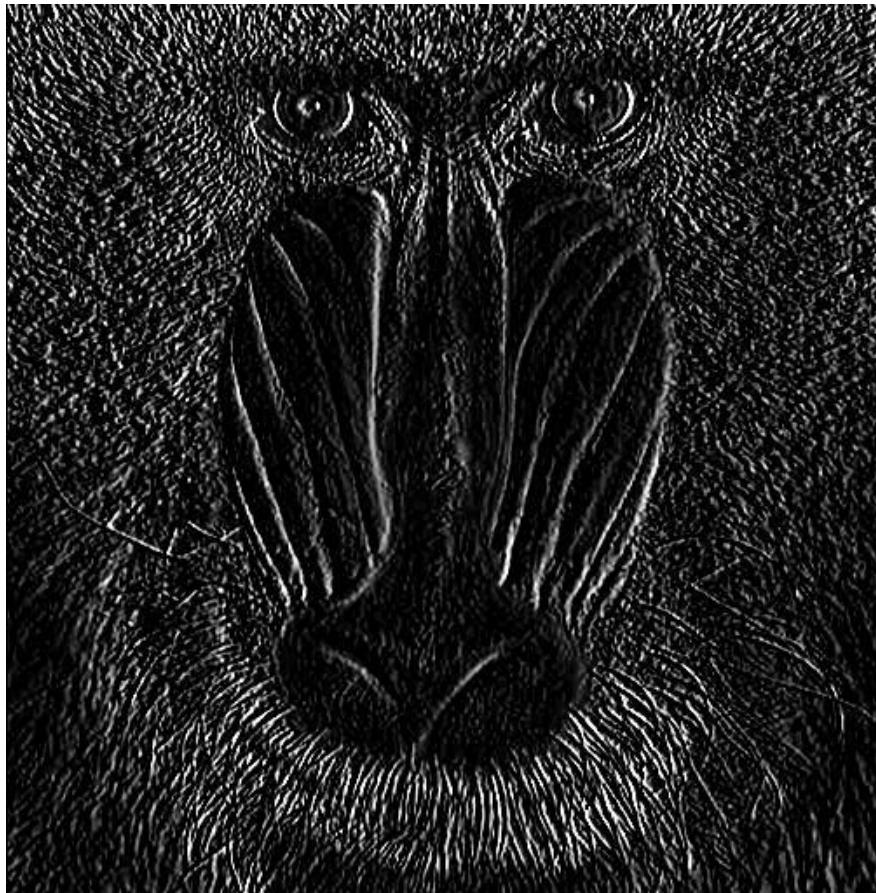
Convolution Effect!

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



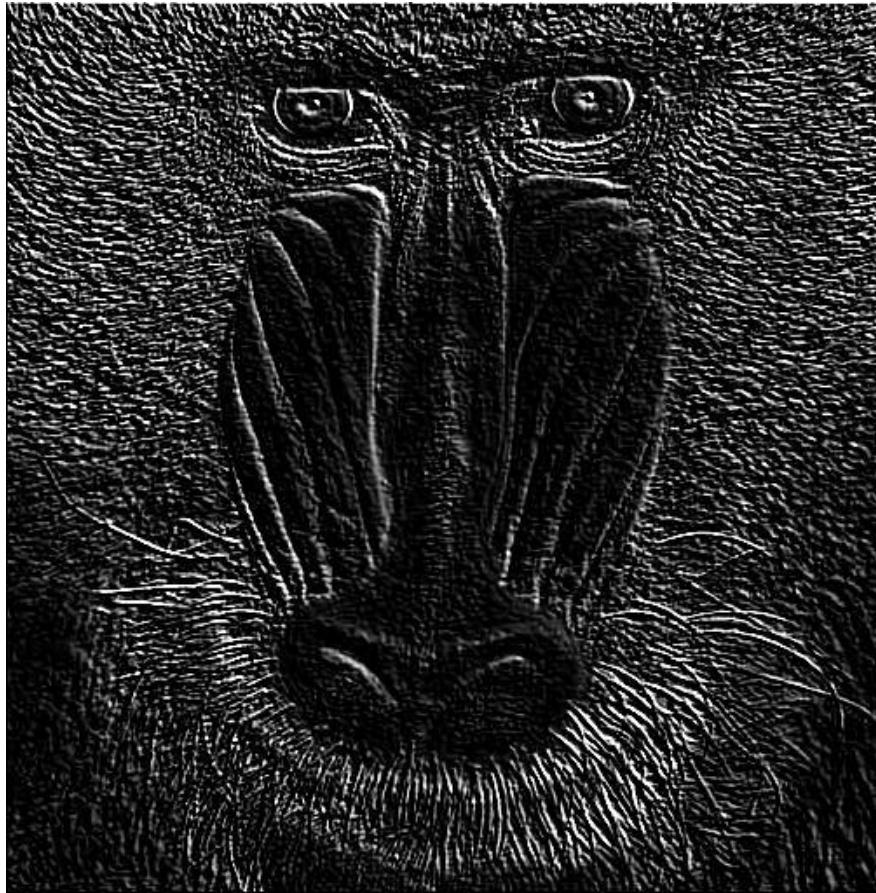
Convolution Effect!

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



Convolution Effect!

$h = [?]$



Convolution Effect!

$h = [?]$



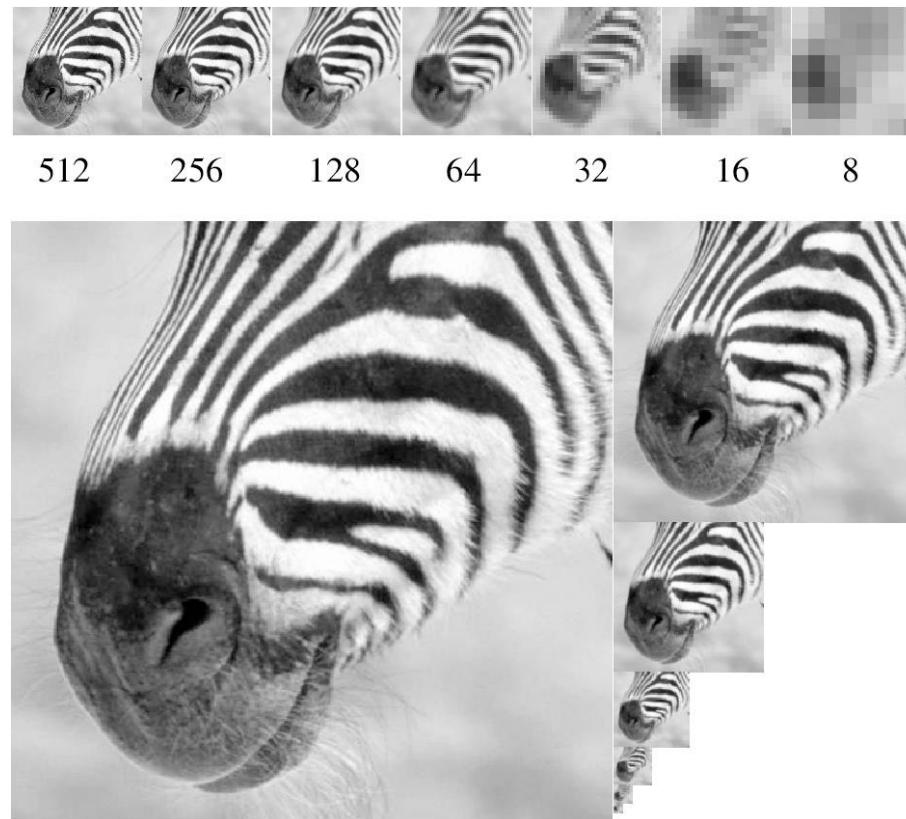
Convolution Effect!

$h = [?]$



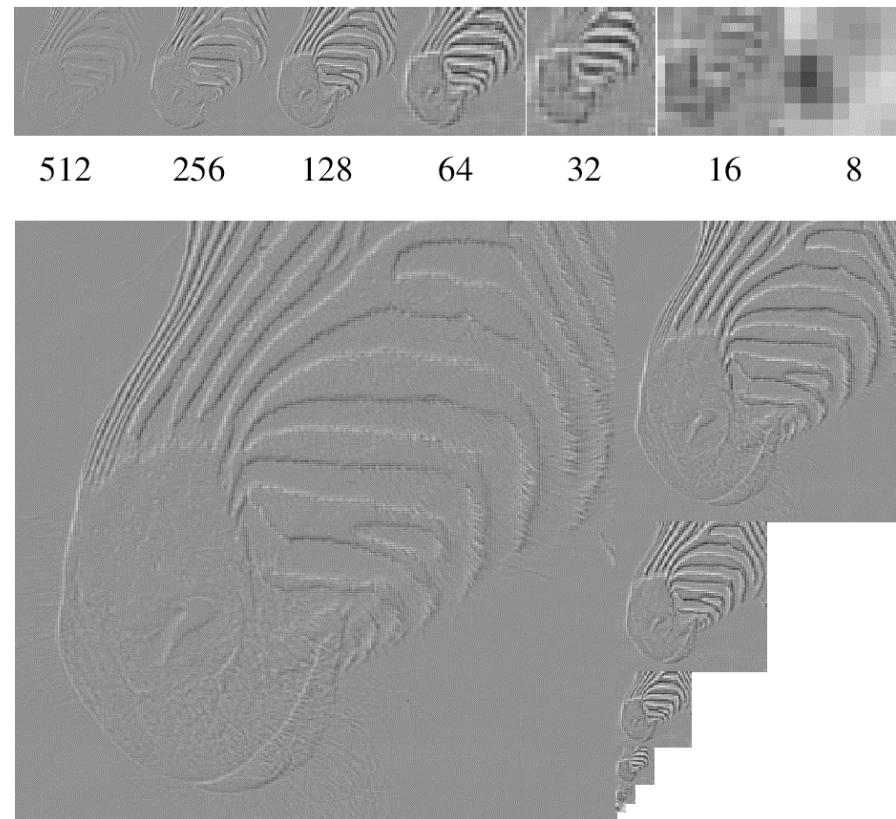
Convolution + Scaling!

- Lowpass filter and down sample:



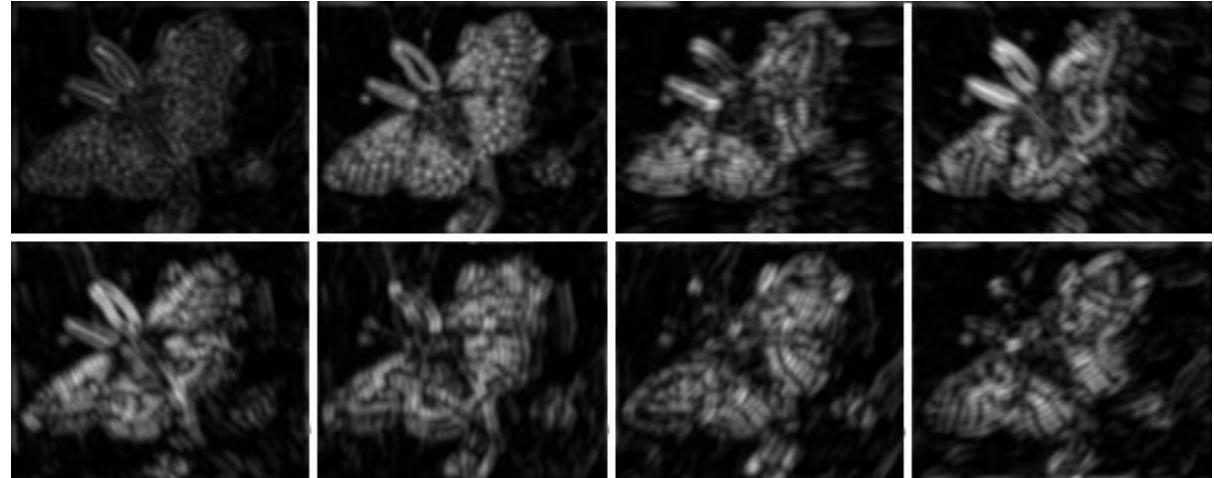
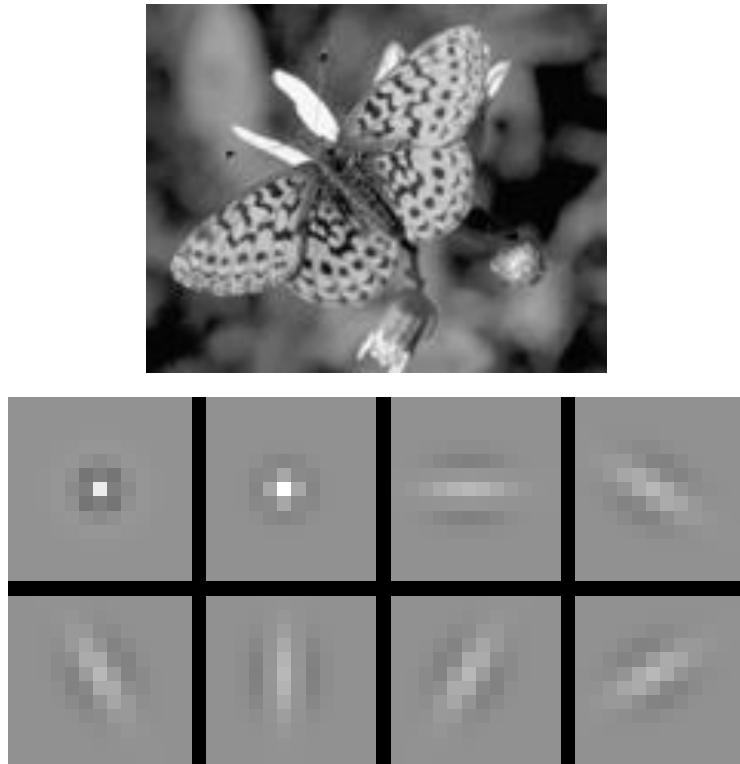
Convolution + Scaling!

- Lowpass filter and down sample:



Filter banks!

- Eight member filter bank:



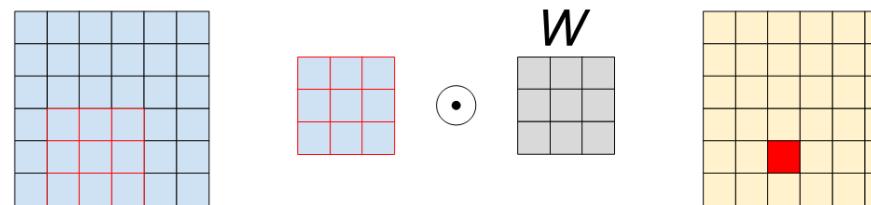
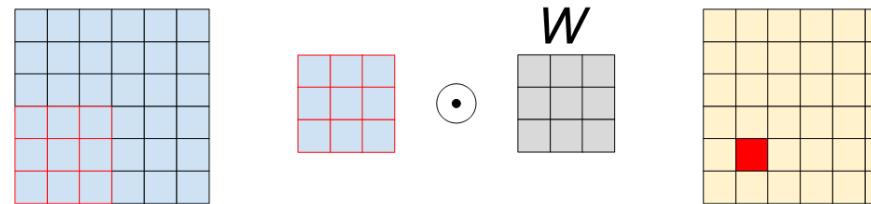
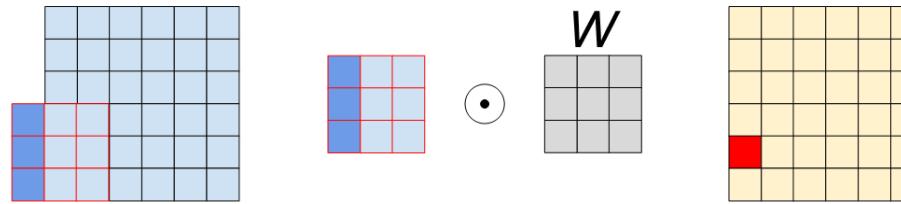
Vocabulary

- Input, x
- Kernel, h
- Feature Map (Output), y

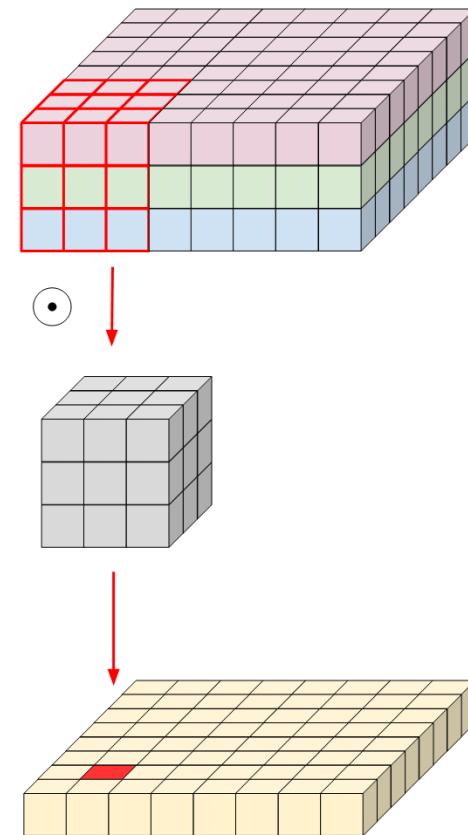
Different Type of Convolution

- 2D Conv, 3D Conv
- 1x1 Conv, BottleNeck
- Spatially Separable
- Depthwise Separable
- Grouped Convolution
- Shuffled Grouped

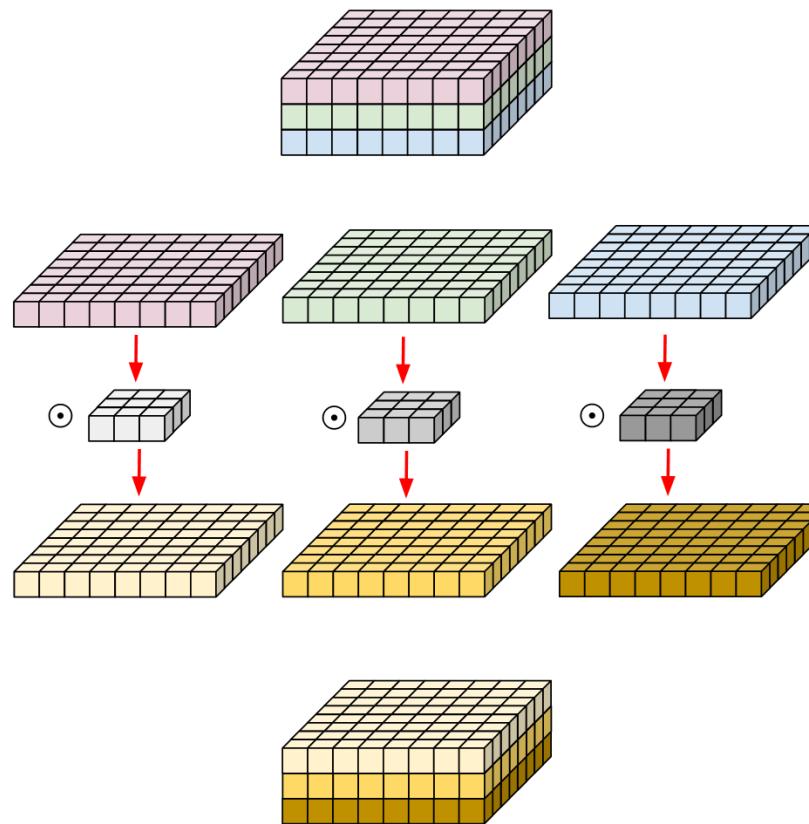
Basic 2D convolution (same and no stride)



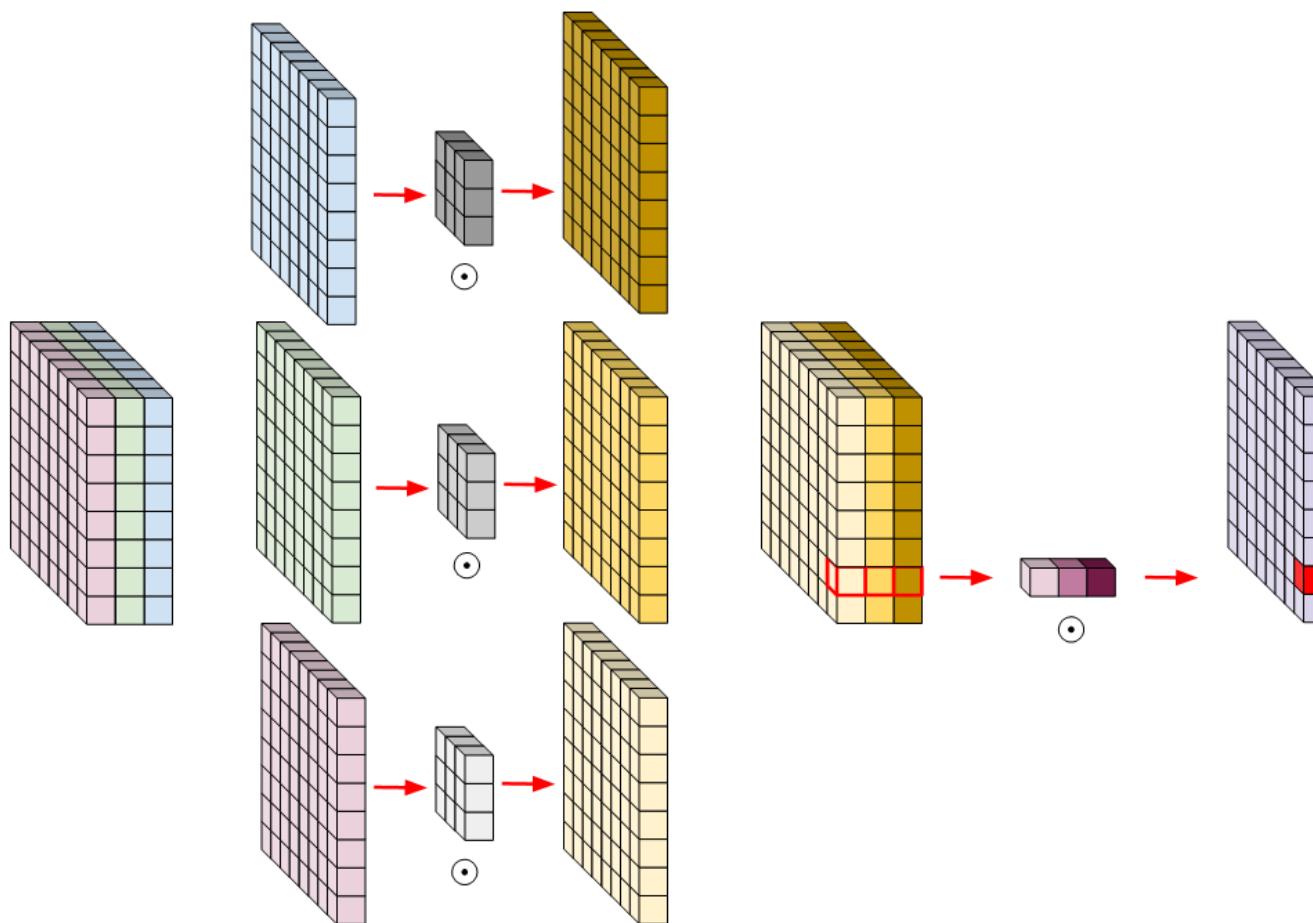
Basic 3D convolution (same and no stride)



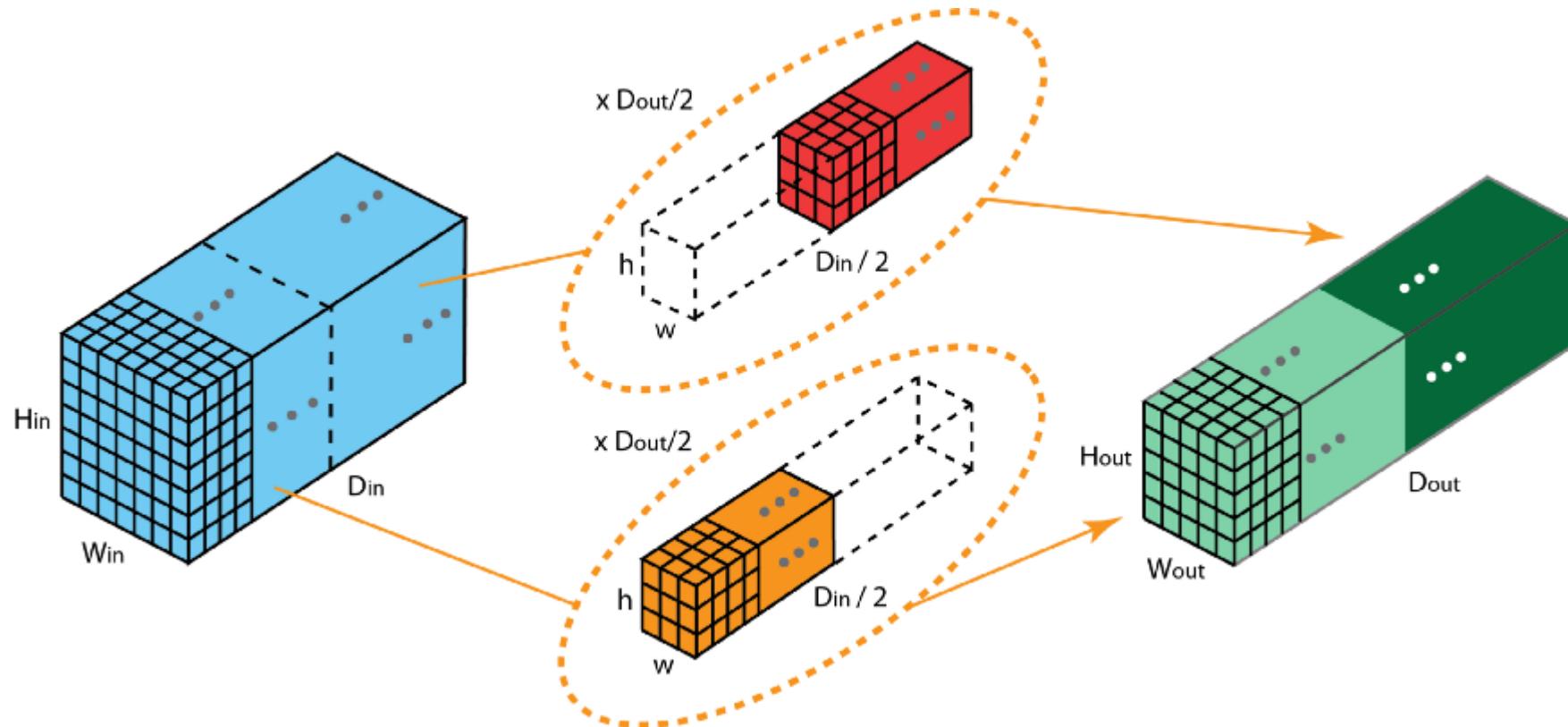
Depthwise Convolution (same and no stride)



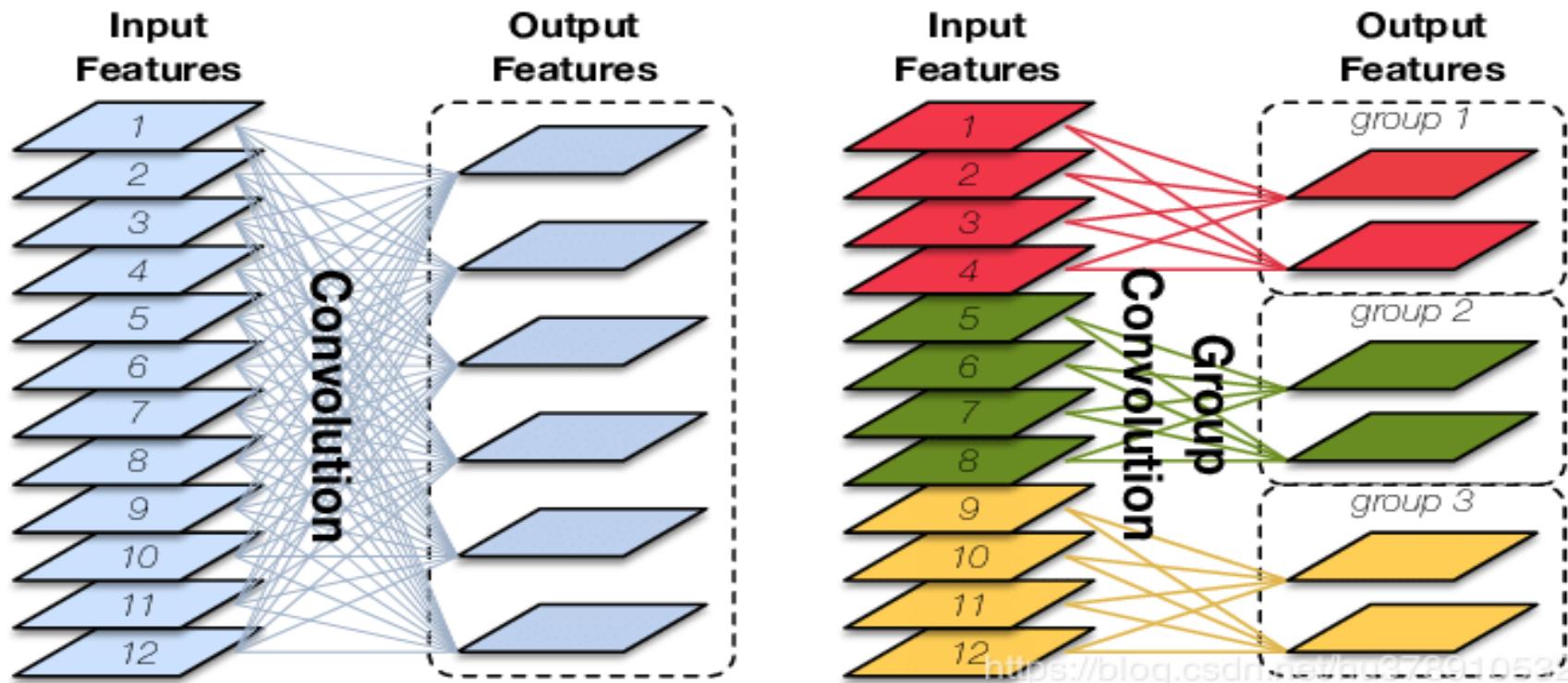
Depthwise Separable Convolution (same and no stride)



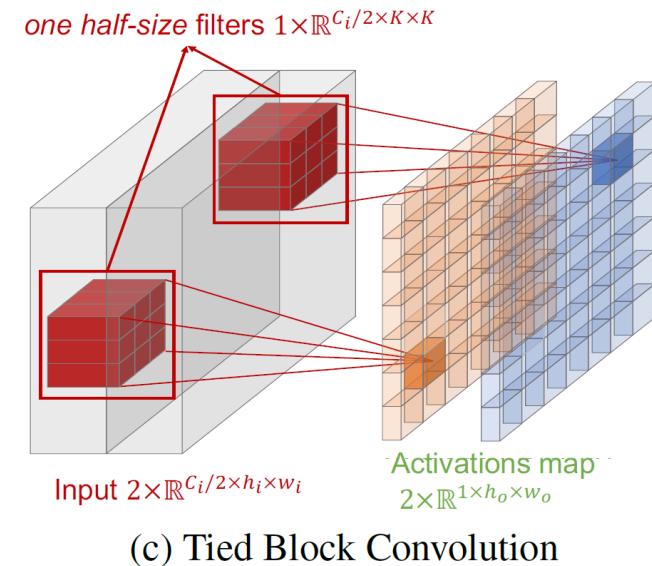
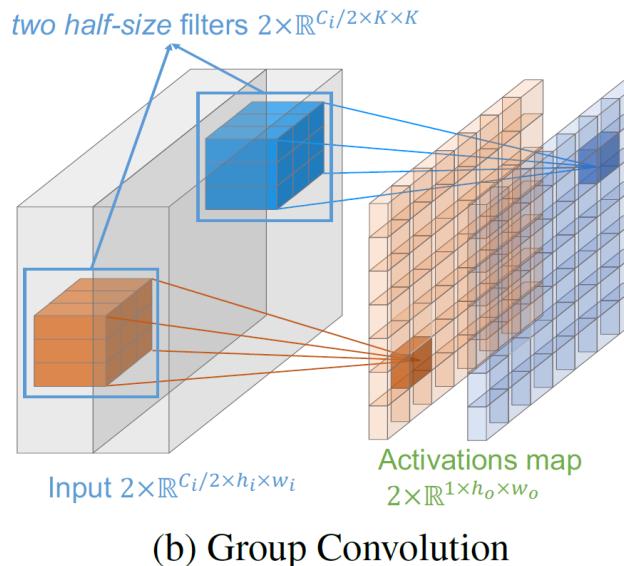
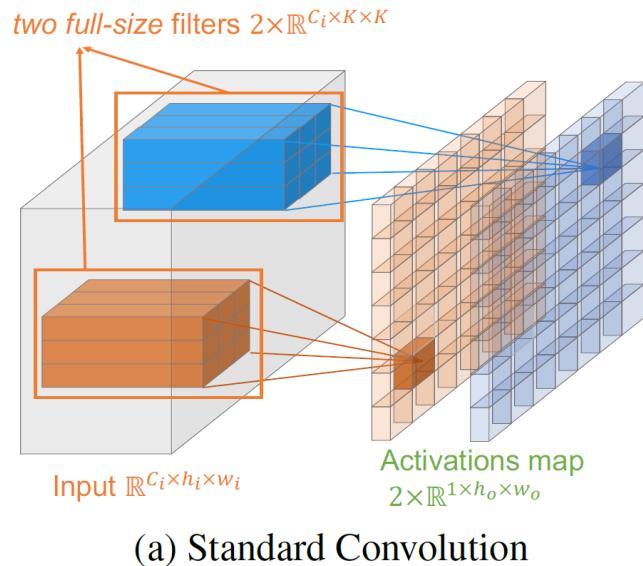
Group Convolution



Group Convolution

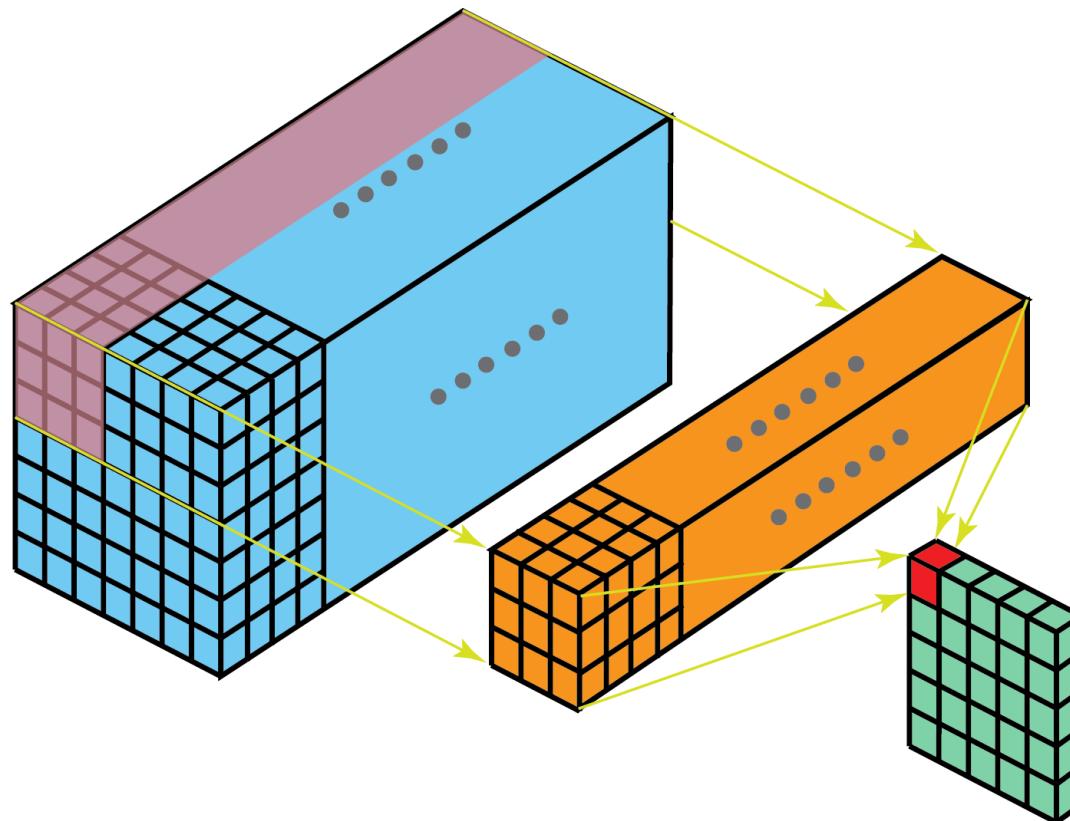


Group Convolution Variant (Tied)



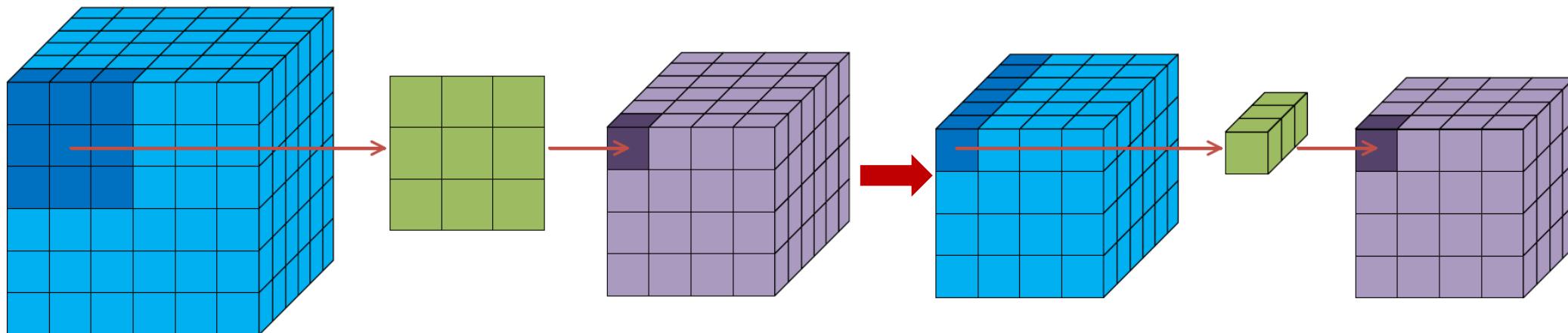
3D - ConvLayer

- Valid implementation:



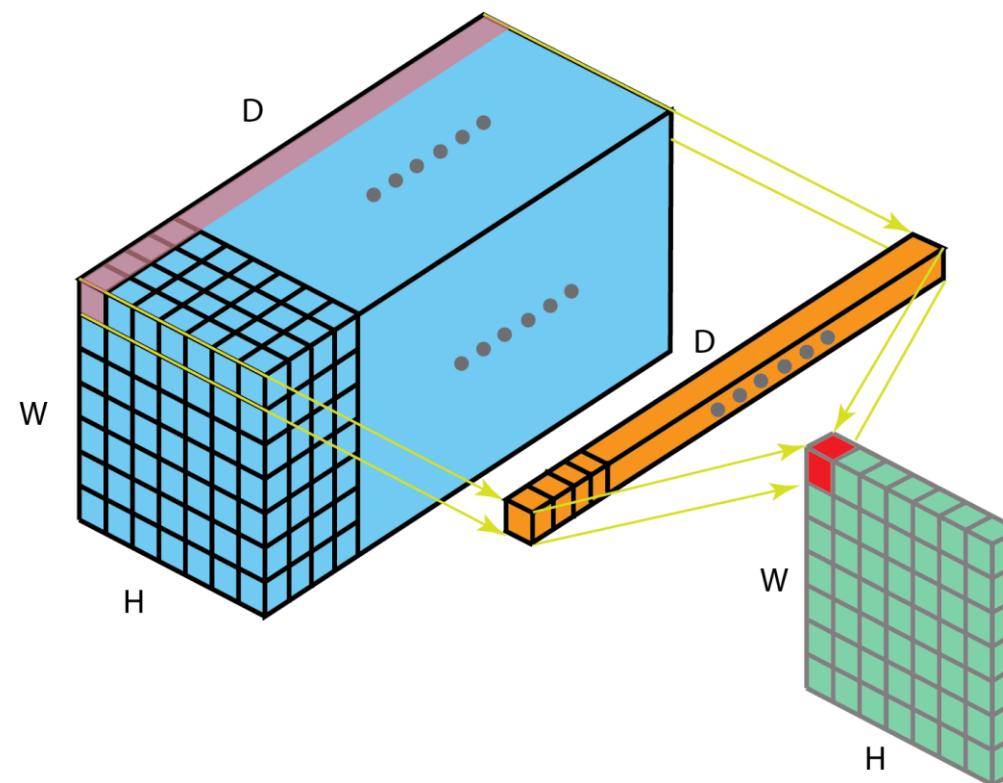
Pseudo-3D Convolution

- It is performed in two step



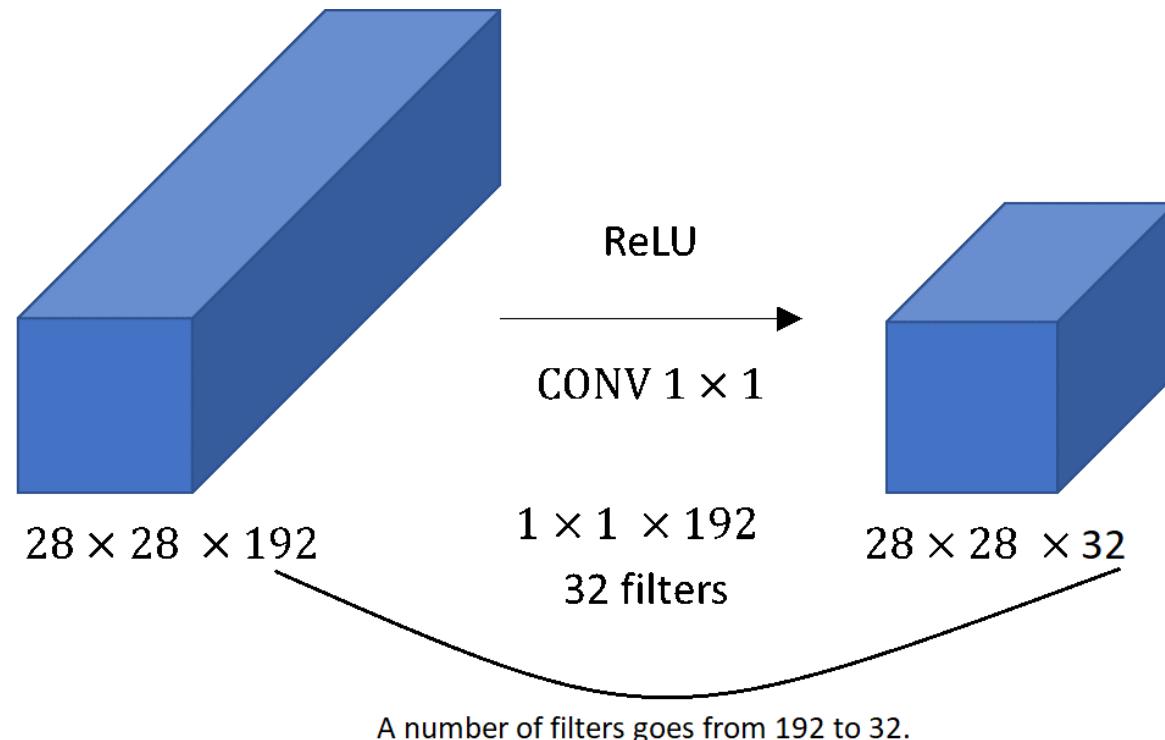
1D - ConvLayer

- 1D or 1×1 (Single) Convolutions (Bottleneck)

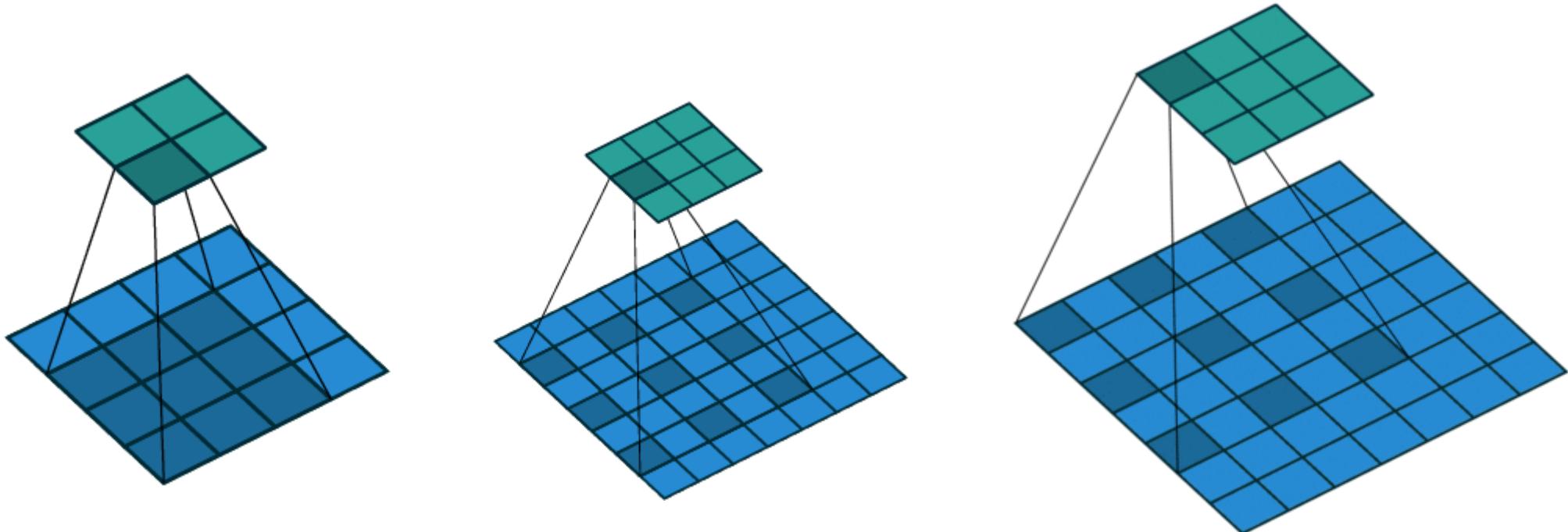


1D - ConvLayer

- 1D or 1×1 (Grouped) Convolutions, Channel Reduction

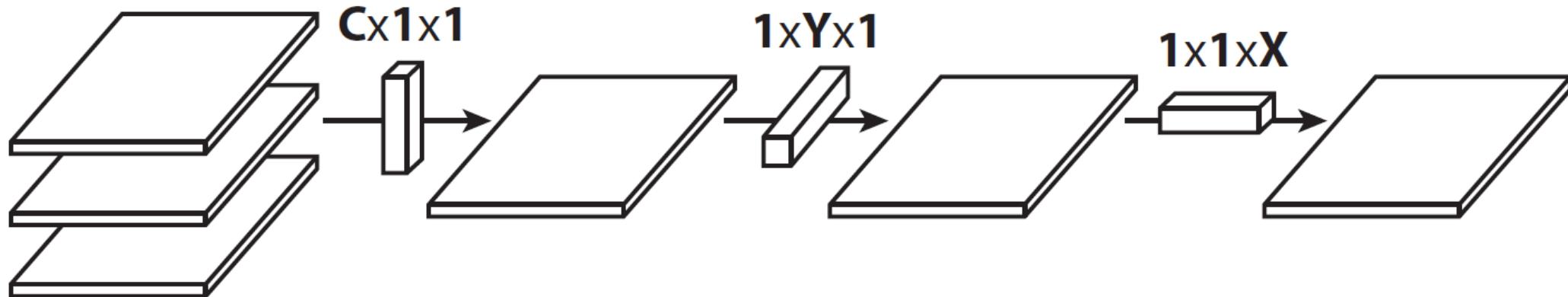


Dilated Convolution



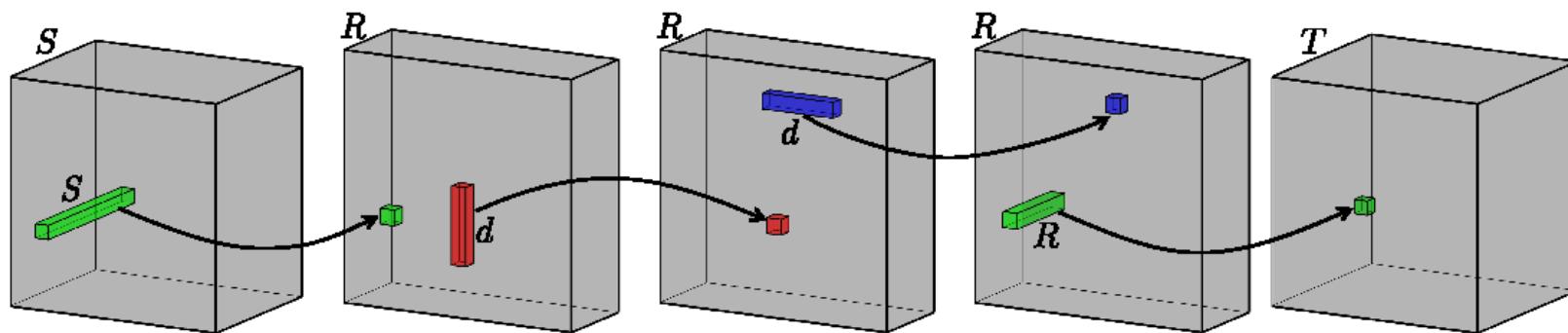
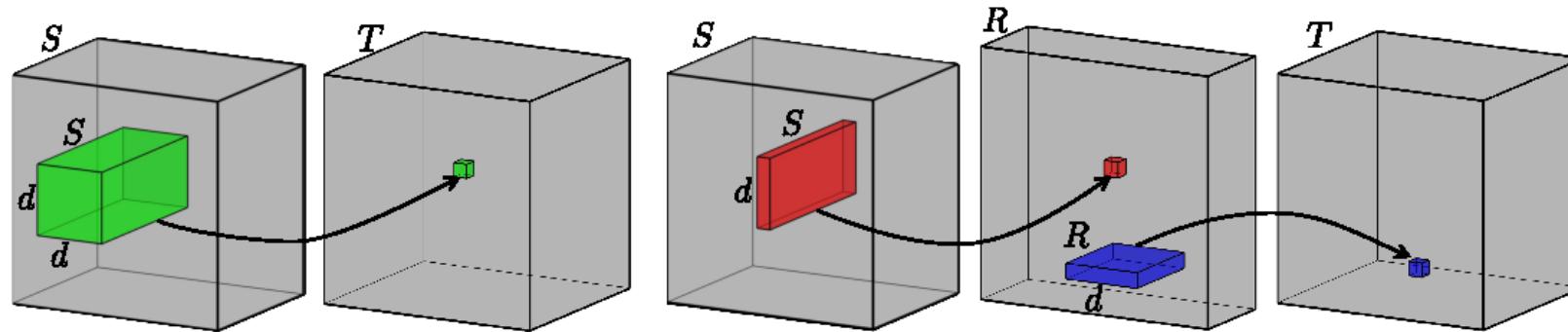
Flattened Convolution

- Use 1D convolution in different direction



Variations

- Higher Order (Complete or Not)



CNN Tailored Regularization

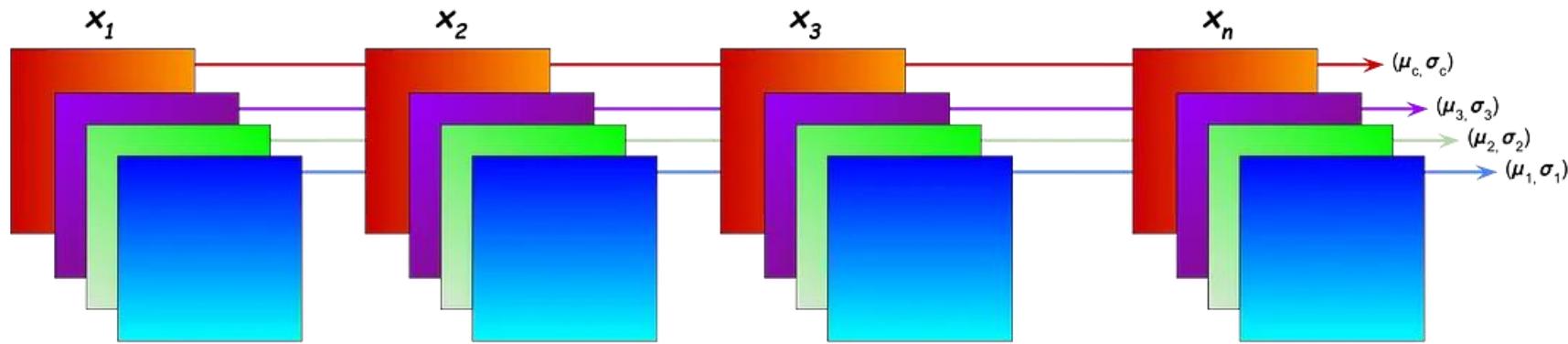
- Normalization
- Notation:
 - N = Batch Size
 - C = Number of Channels (filters)
 - H = Height of each activation map
 - W = Width of each activation map
- Feature map dimension:

$$x \in \mathbb{R}^{N \times C \times H \times W}$$

- Normalization:
$$\hat{x} = \frac{x - \mu}{\sigma}$$
- Batch Normalization, Instance Normalization and Layer Normalization differ in the manner these statistics are calculated.

CNN Tailored Regularization

- Batch Normalization:

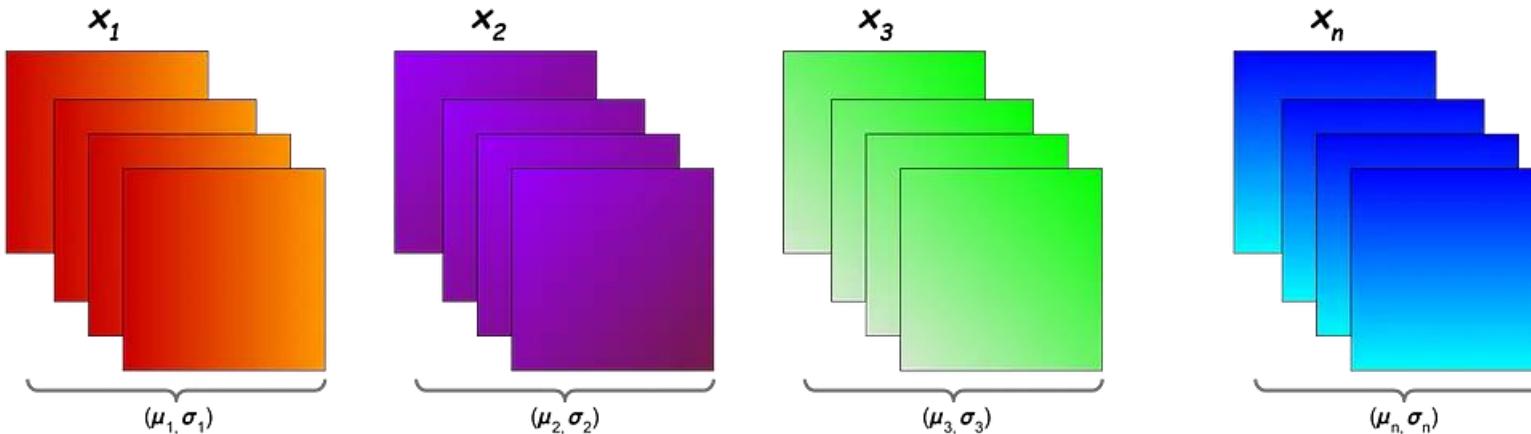


$$\mu_c = \frac{1}{NHW} \sum_{i=1}^N \sum_{j=1}^H \sum_{k=1}^W x_{icjk}, \quad \sigma_c^2 = \frac{1}{NHW} \sum_{i=1}^N \sum_{j=1}^H \sum_{k=1}^W (x_{icjk} - \mu_c)^2$$

- In Batch Normalization, mean and variance are calculated for each individual channel across *all samples* and *both spatial dimensions*.

CNN Tailored Regularization

- Layer Normalization:

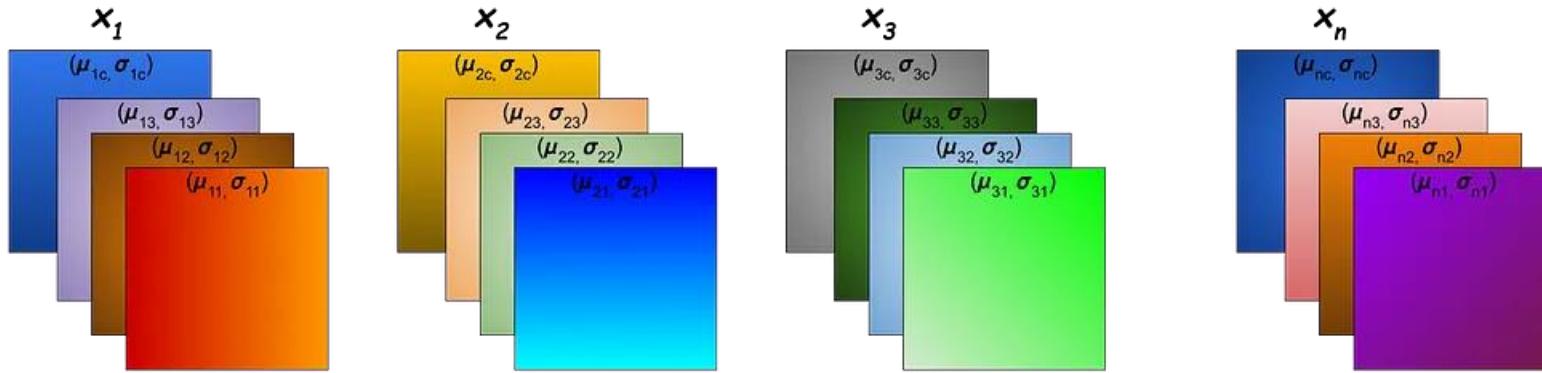


$$\mu_{\textcolor{red}{n}} = \frac{1}{CHW} \sum_{i=1}^C \sum_{j=1}^H \sum_{k=1}^W x_{nijk}, \quad \sigma_n^2 = \frac{1}{CHW} \sum_{i=1}^C \sum_{j=1}^H \sum_{k=1}^W (x_{nijk} - \mu_{\textcolor{red}{n}})^2$$

- In Layer Normalization, mean and variance are calculated for each individual sample across all *channels* and *both spatial dimensions*.

CNN Tailored Regularization

- Instant Normalization:

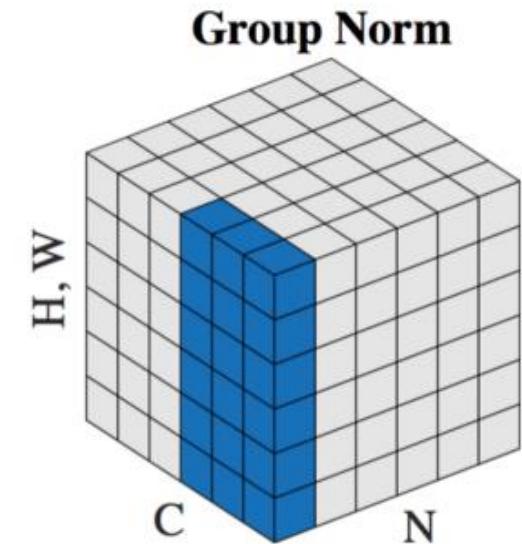
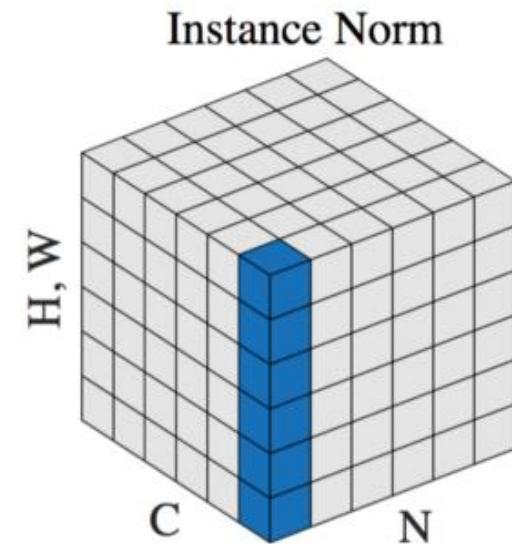
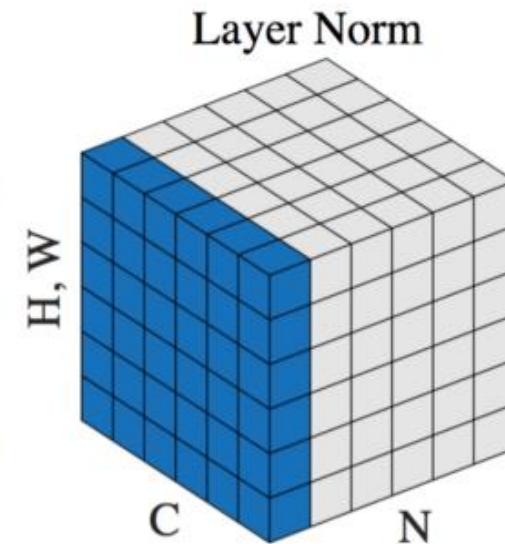
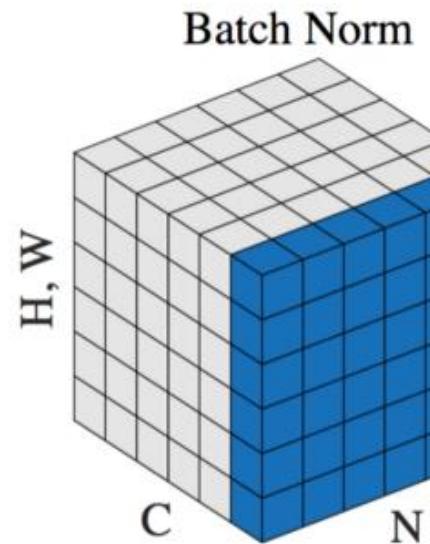


$$\mu_{nc} = \frac{1}{HW} \sum_{j=1}^H \sum_{k=1}^W x_{ncjk}, \sigma_{nc}^2 = \frac{1}{HW} \sum_{j=1}^H \sum_{k=1}^W (x_{ncjk} - \mu_{nc})^2$$

- In Instance Normalization, mean and variance are calculated for each individual *channel* for each *individual sample* across *both spatial dimensions*.

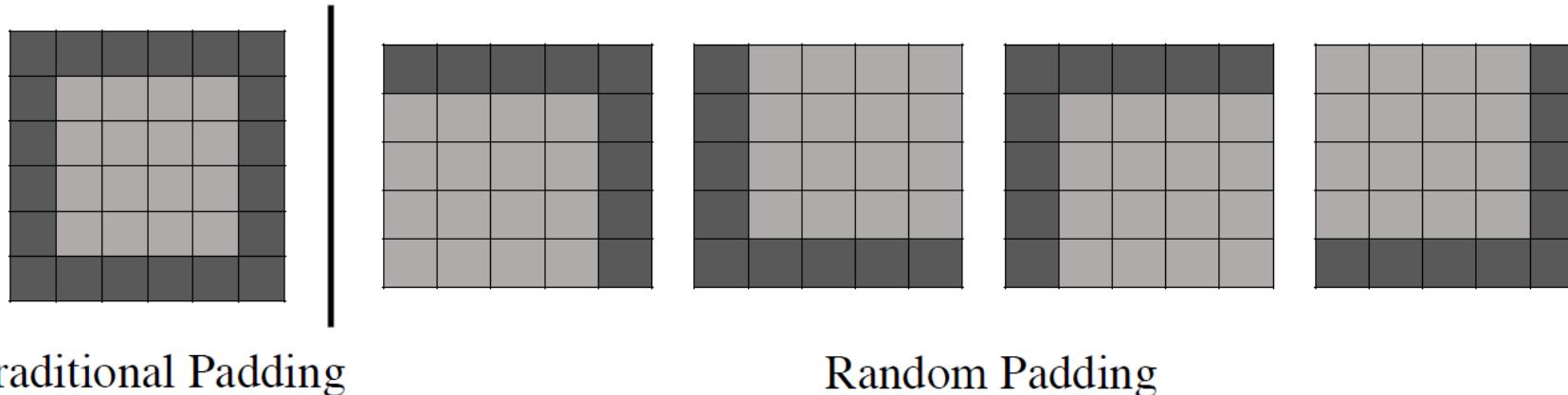
CNN Tailored Regularization

- All in One



CNN Tailored Regularization

- Random Padding (Act as Data Augmentation)

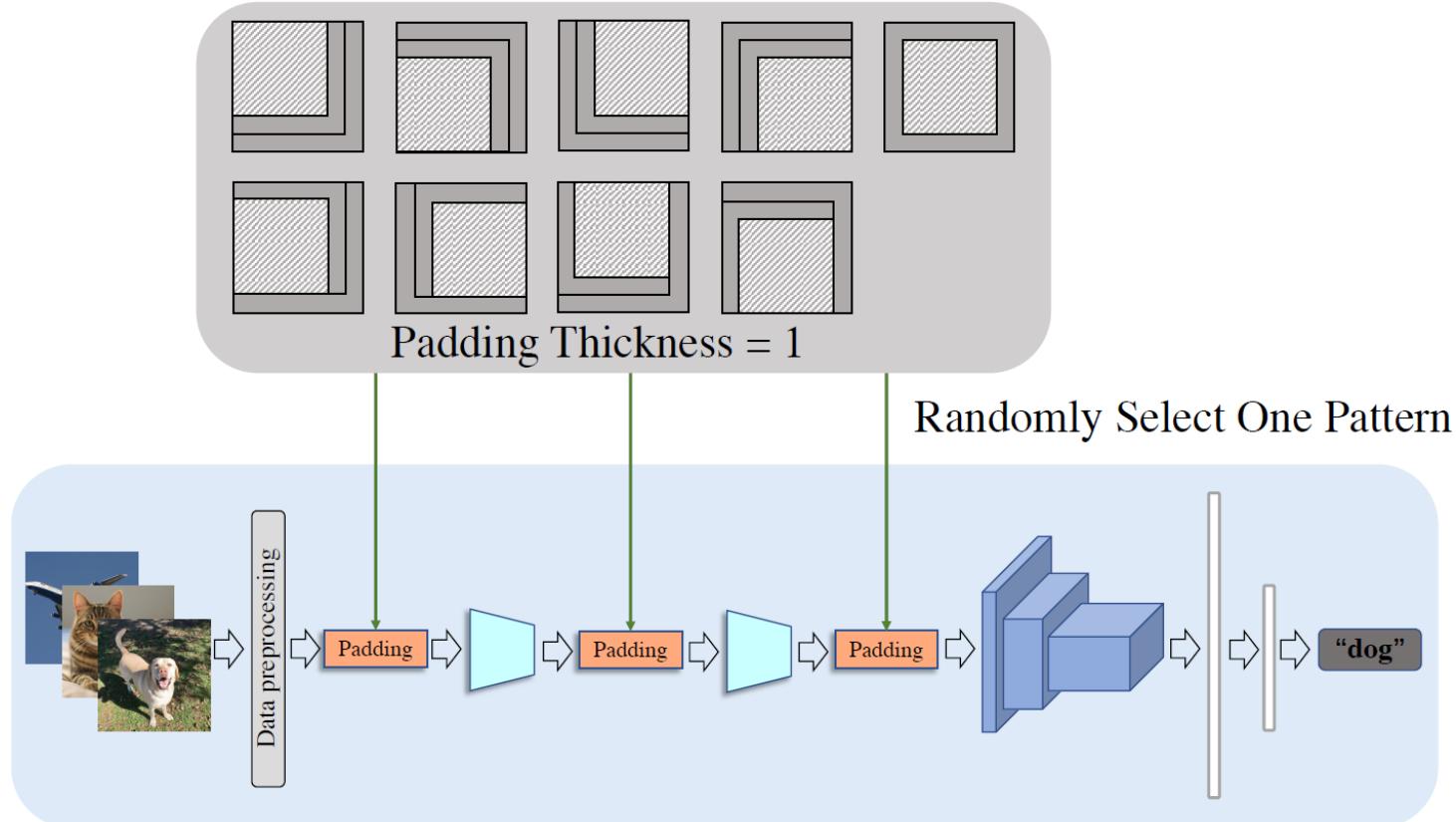


Traditional Padding

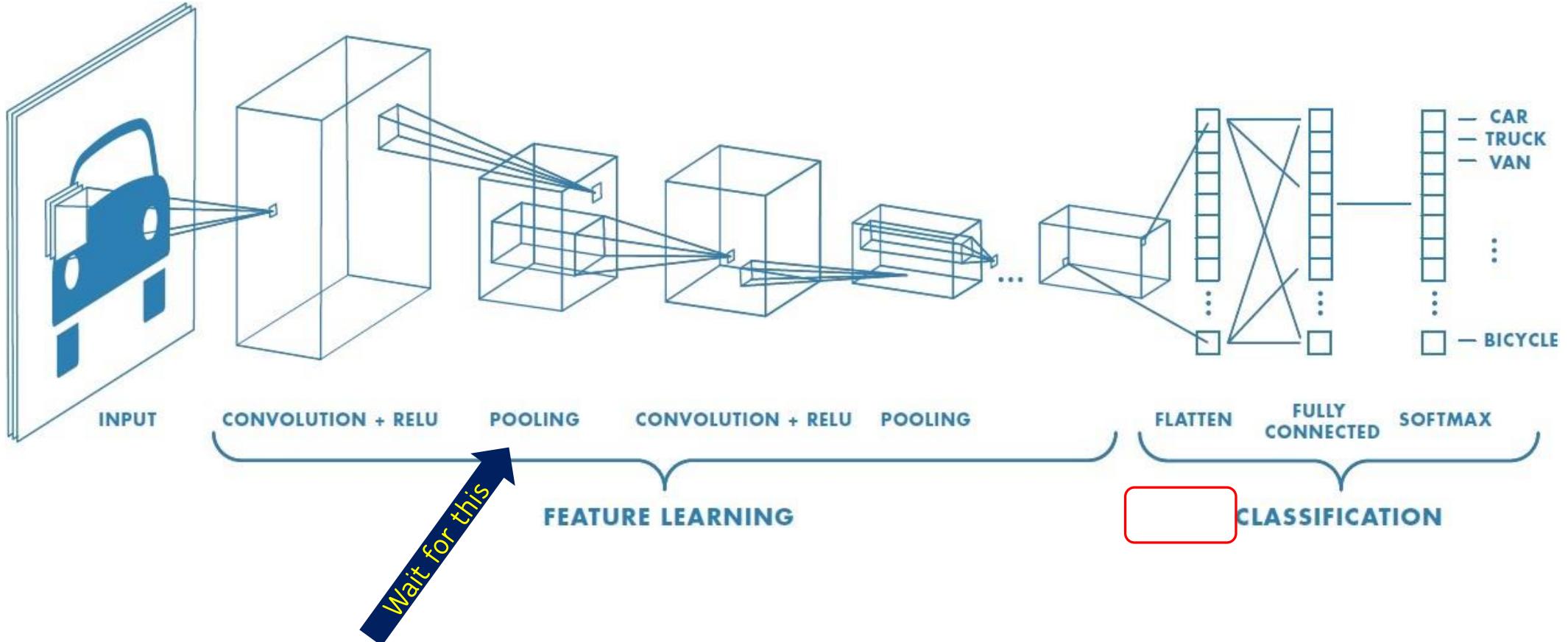
Random Padding

CNN Tailored Regularization

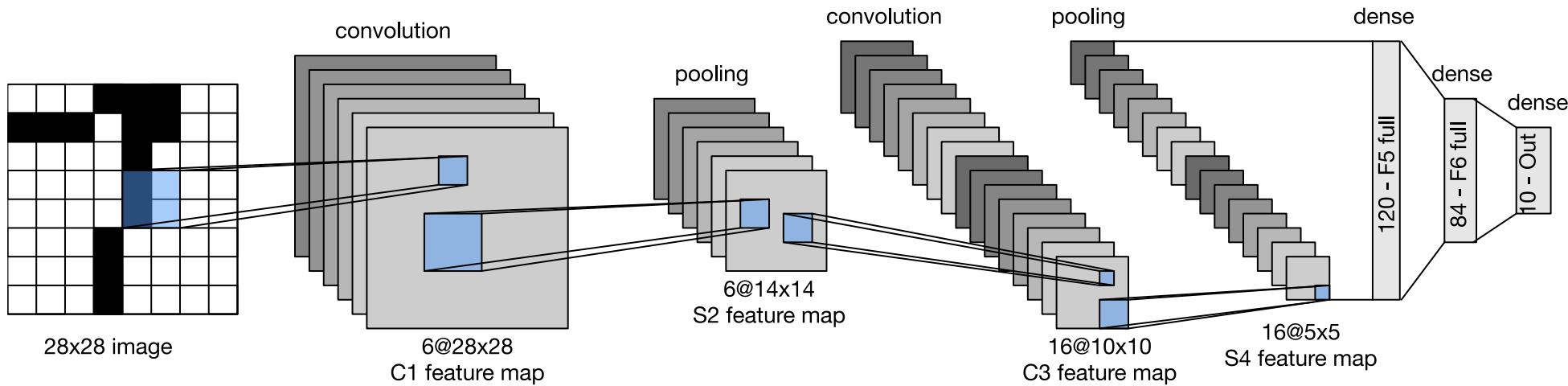
- Random Padding (Act as Data Augmentation)



A Typical CNN

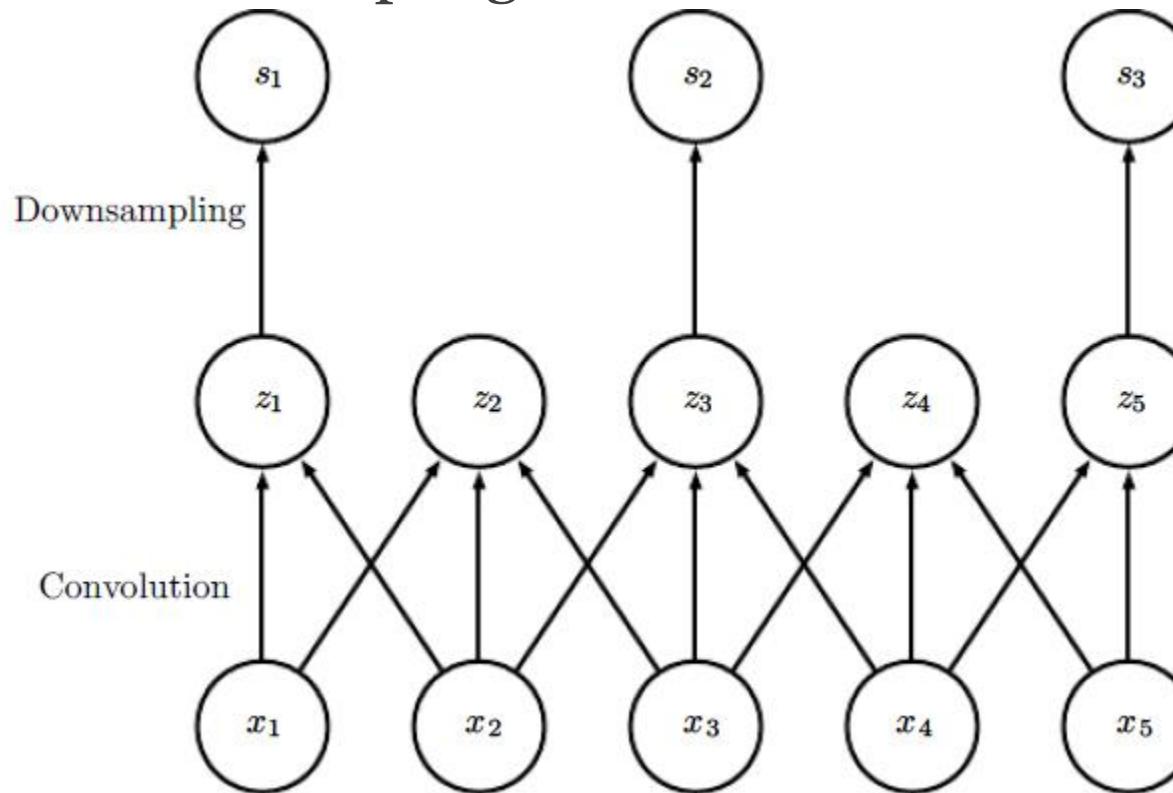


An Old CNN: LeNet (LeCun, 1998)



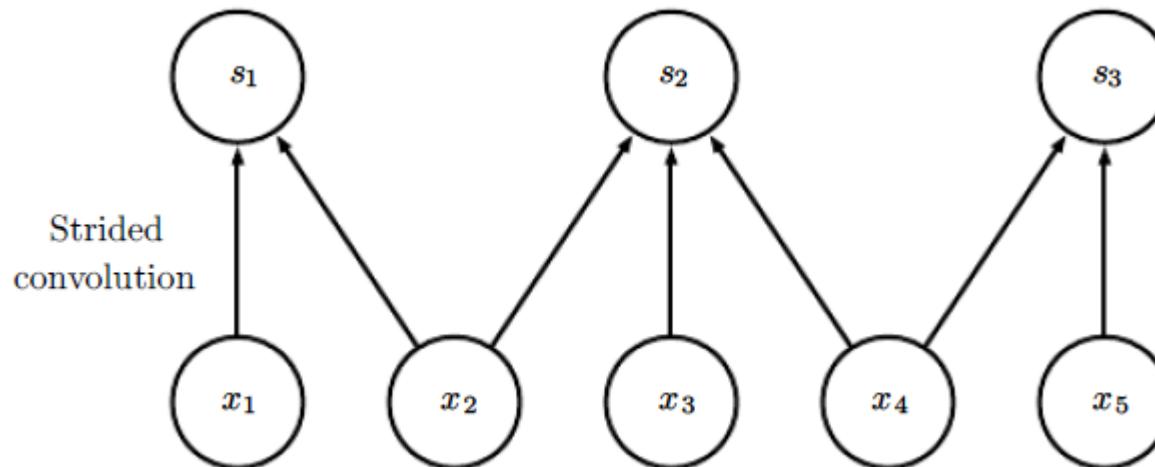
Variation

- Convolution and Downsampling:



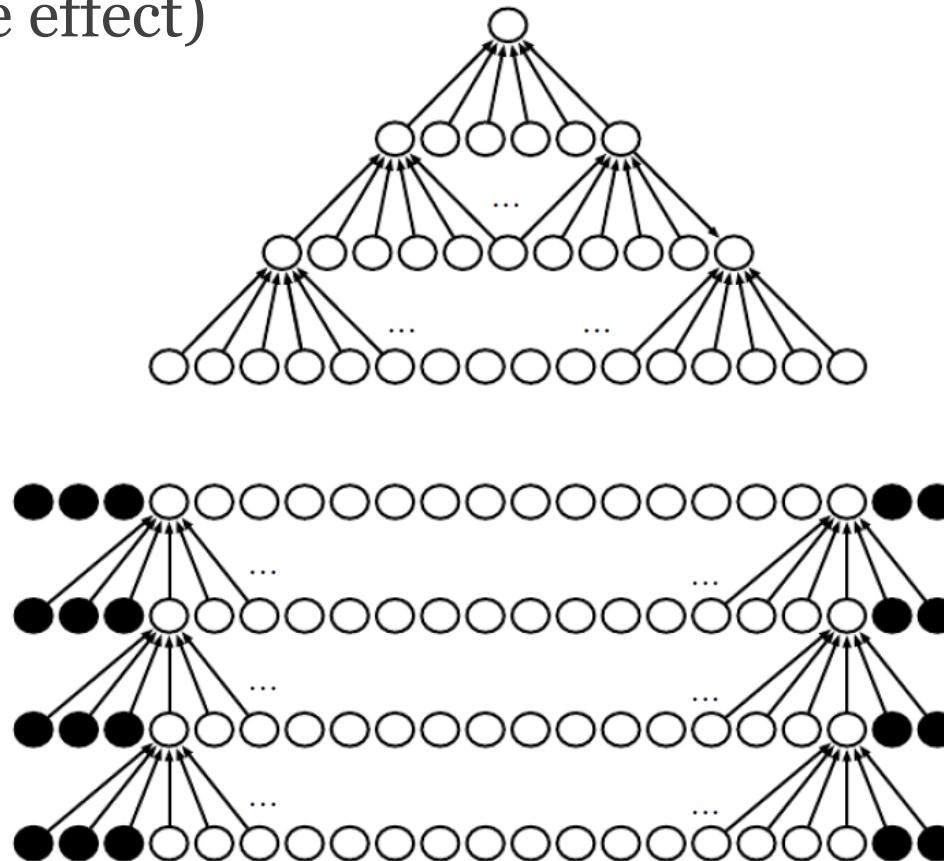
Variation

- Convolution and Downsampling:



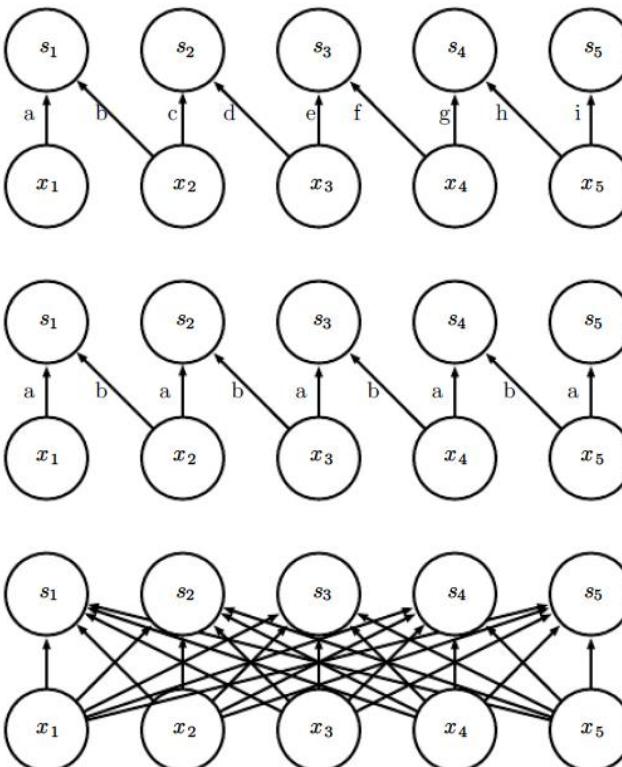
Variation

- Zero Padding (Size effect)



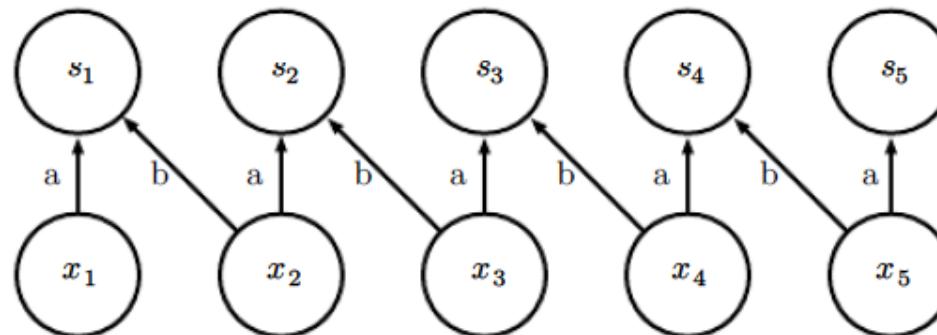
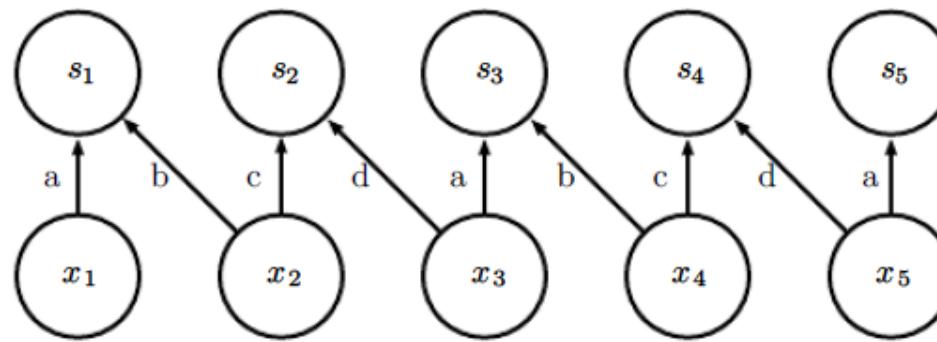
Variation

- Unshared Convolution (Shift Variant Convolution)
 - Each Node has its own weights (Locally Connected Layer)



Variation

- Tiled Convolution:
 - Two or More Kernel in each Layer

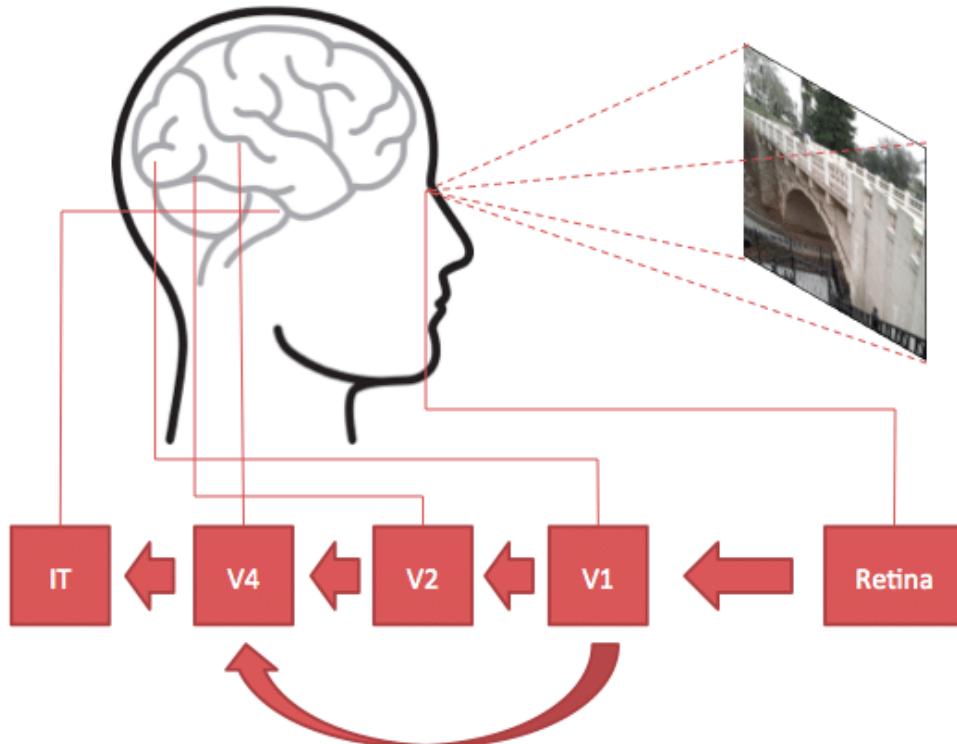


CNN History

- 1959: Neurobiological Experiments (Layers, Receptive Field,)
- 1980-1999: Neocognitron (Fukushima), ConvNet (LeCun)
- Early 2000: Winter Sleep!
- 2006-2011: Resurrection of CNN
- 2012-2014: Rise of CNN
- 2015-Present: Rapid Architectural Innovations and Applications

The Neuroscientific Basis

- Primary Visual Cortex, Theory: from 1959-1985



V1: Edge detection, etc.

V2: Extract simple visual properties (orientation, spatial frequency, color, etc)

V4: Detect object feature of intermediate complexity

TI: Object recognition

The Neuroscientific Basis

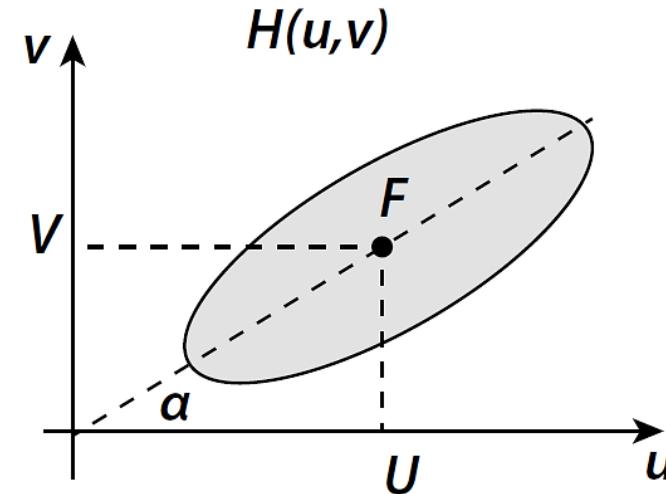
- V1 Cell in Primary Visual Cortex, Model by Gabor Transform
 - Image representation:
 - Spatial domain: show where but not what!
 - Frequency domain: show what but not where!
 - Need a local representation what and where.
- Core idea:
 - Characteristic spatial location nature, orientation, selectivity, and frequency.

Gabor Filter

- Gabor filter (pair):

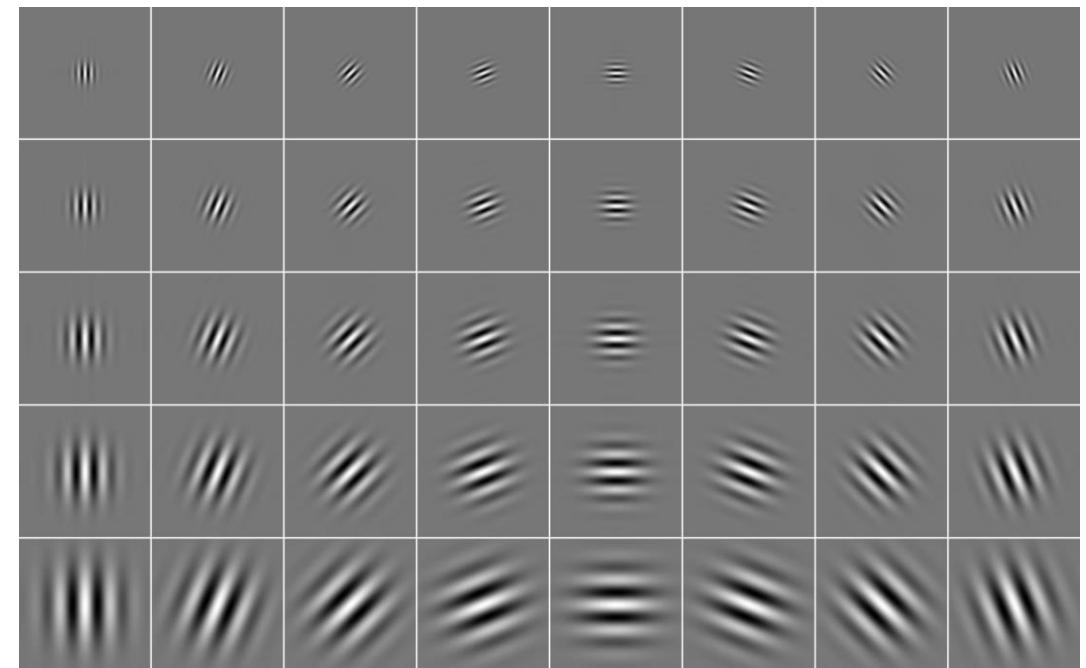
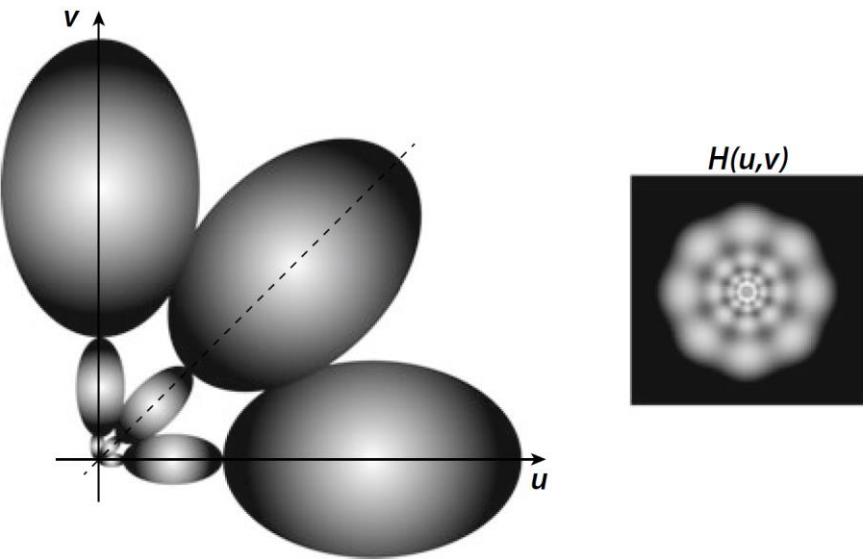
$$h_G(x, y) = \frac{1}{2\pi\sqrt{\sigma_x\sigma_y}} e^{-\frac{1}{2}\left(\frac{(x\cos\alpha+y\sin\alpha)^2}{\sigma_x^2} + \frac{(-x\sin\alpha+y\cos\alpha)^2}{\sigma_y^2}\right)} e^{2\pi j(Ux+Vy)}$$

- (σ_x, σ_y) : scaling factor
- α : orientation
- (U, V) : frequency
- $H_G(u, v) = e^{-2\pi^2[\sigma_x^2(\omega_u - U)^2 + \sigma_v^2(\omega_v - V)^2]}$
- $\omega_u = u\cos\alpha + v\sin\alpha$, $\omega_v = -u\sin\alpha + v\cos\alpha$



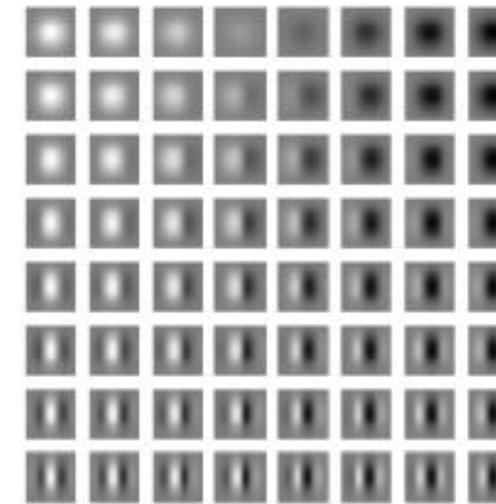
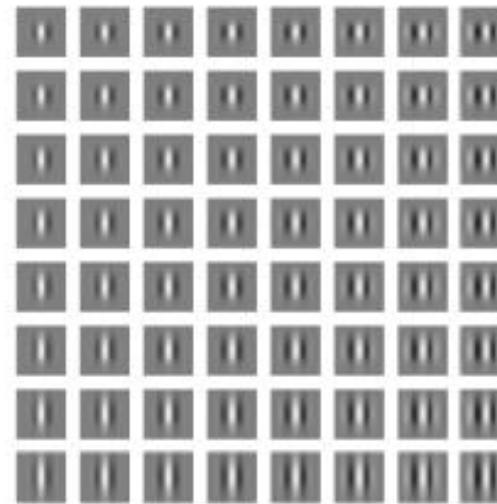
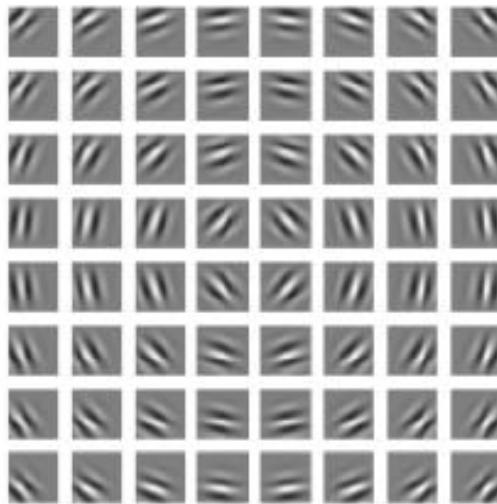
Gabor Filter

- Gabor *filter-bank* (orientation, scale, and central frequency):



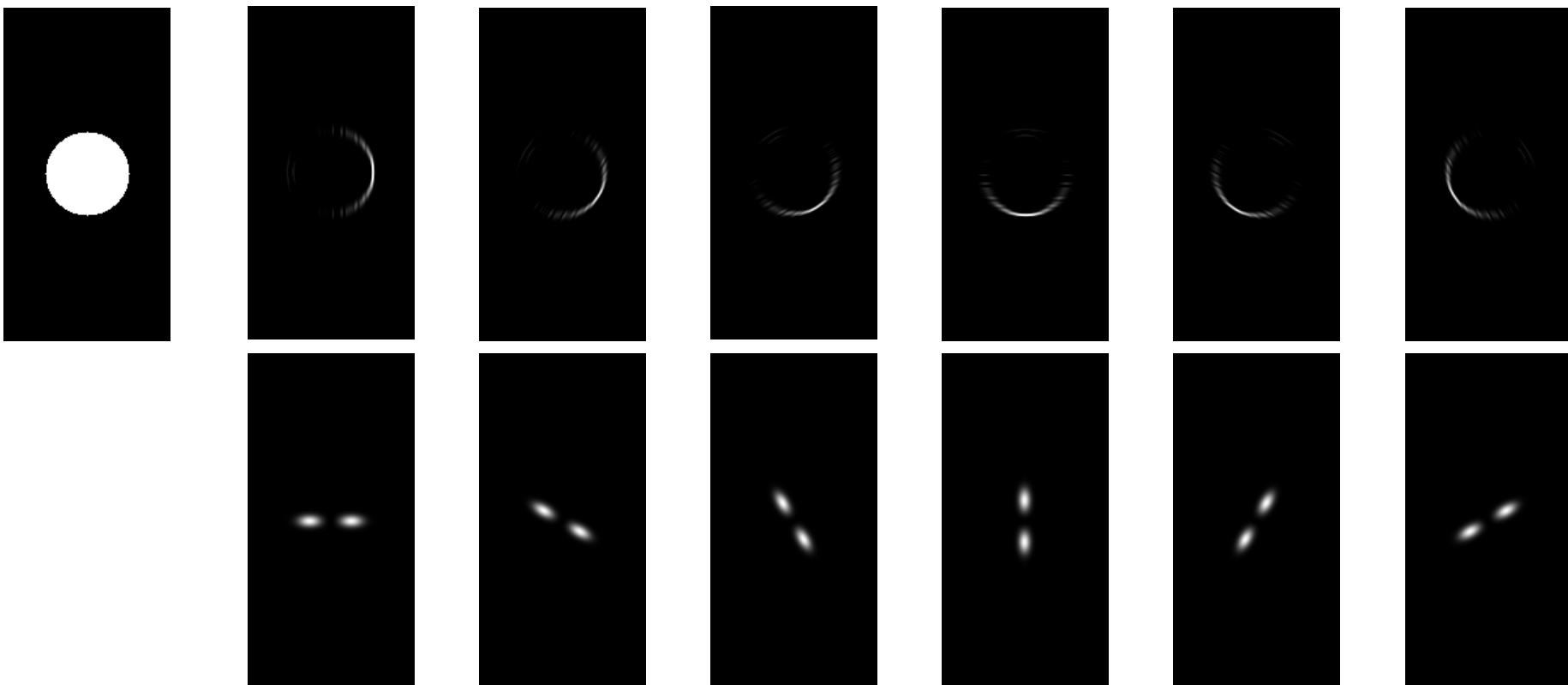
Gabor Function

- Example



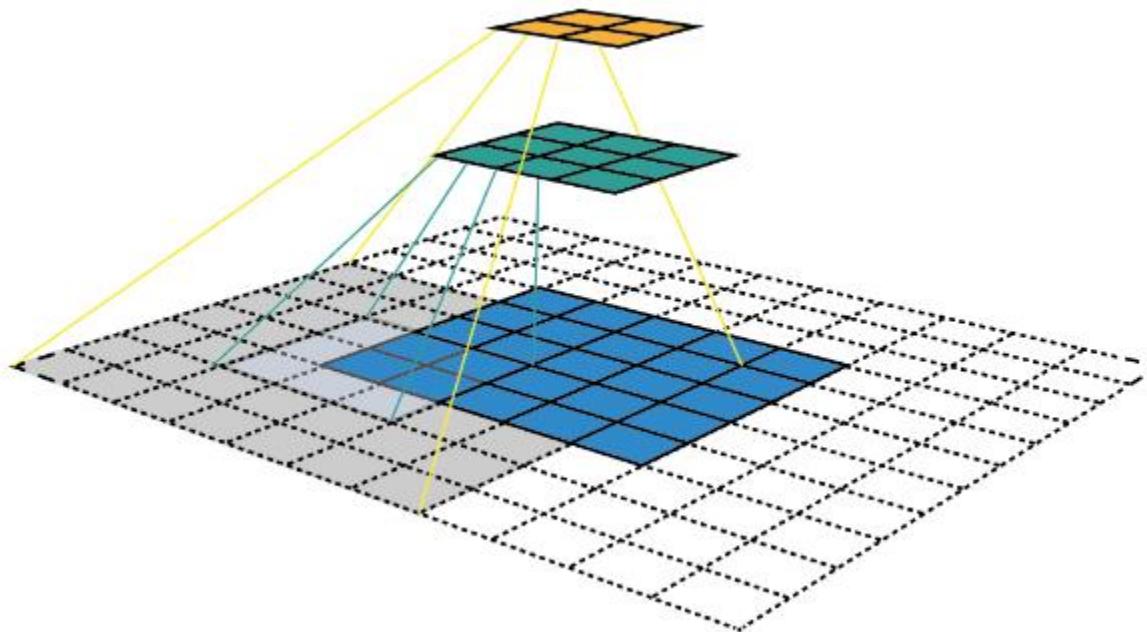
Gabor Filter Bank

- Example:



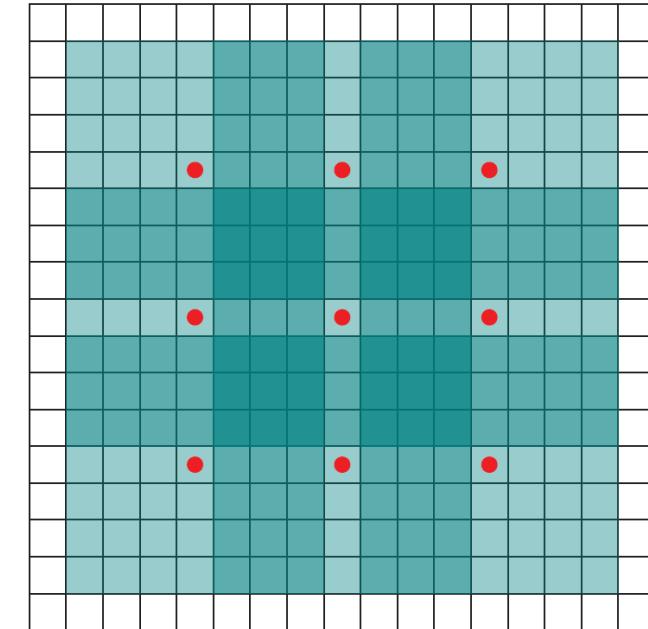
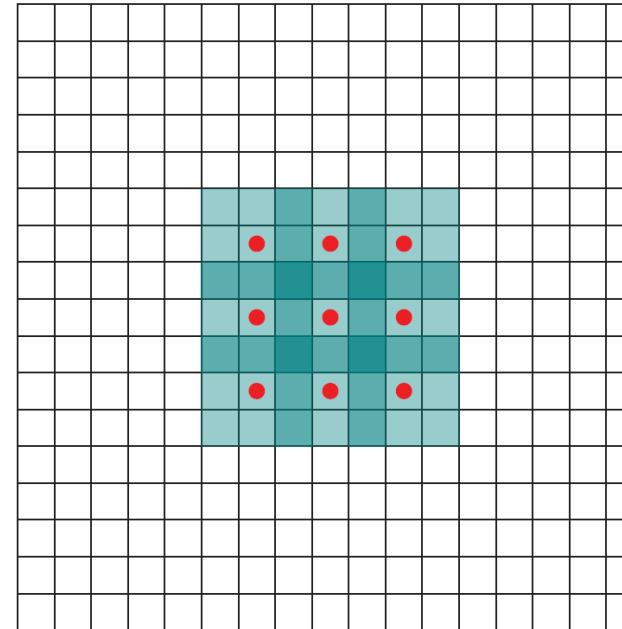
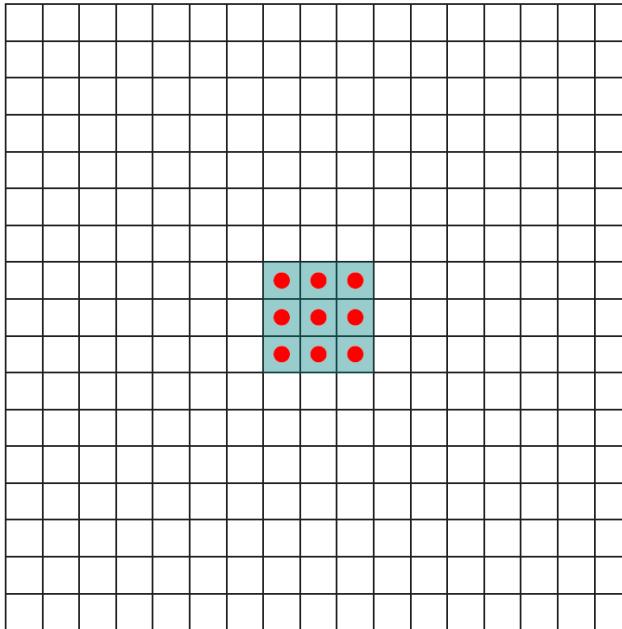
Receptive Field in CNN

- The region in the input space that a particular CNN's feature is looking at (or affected by)



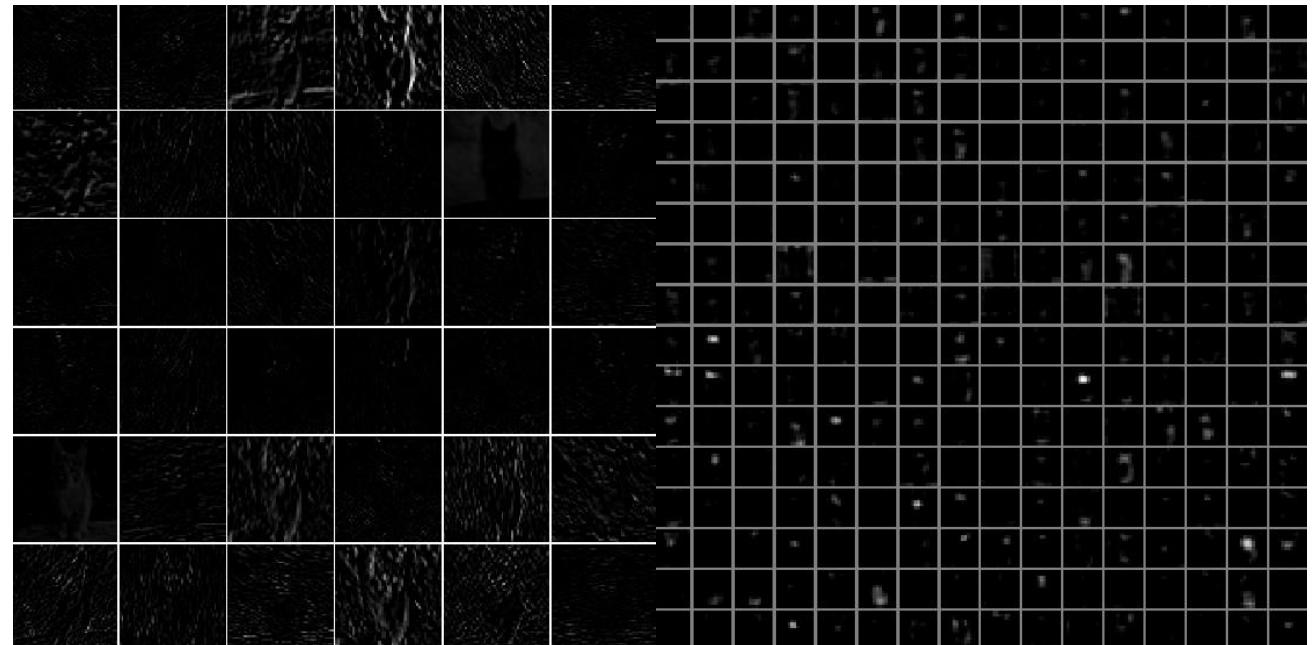
Receptive Field in CNN

- The region in the input space that a particular CNN's feature is looking at (or affected by)



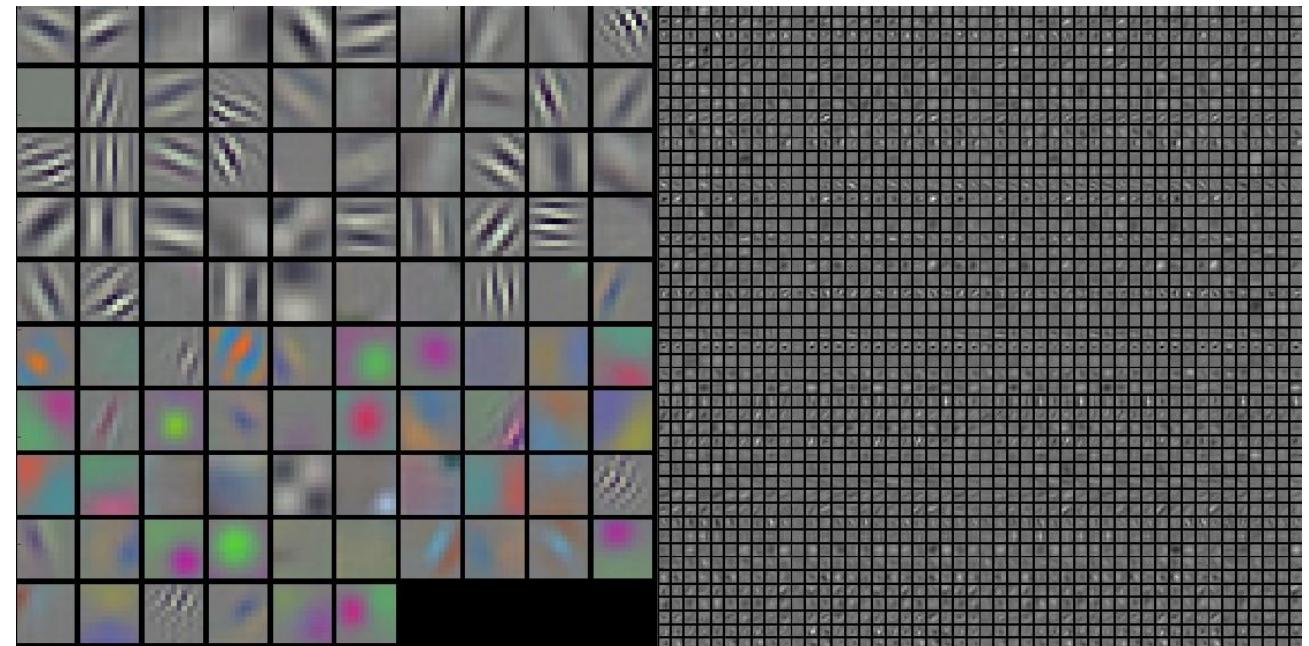
What Feature Layers Learn -Visualizing Strategies

- Method #1: Visualize the activations of the network in response to an input
 - Input is Cat!
 - Layer #1 and #5 of AlexNet
 - Map is Sparse



What Feature Layers Learn -Visualizing Strategies

- Method #2: Visualize the weights (filters)
 - 1st layer is informative
 - Layer #1/#2 of AlexNet



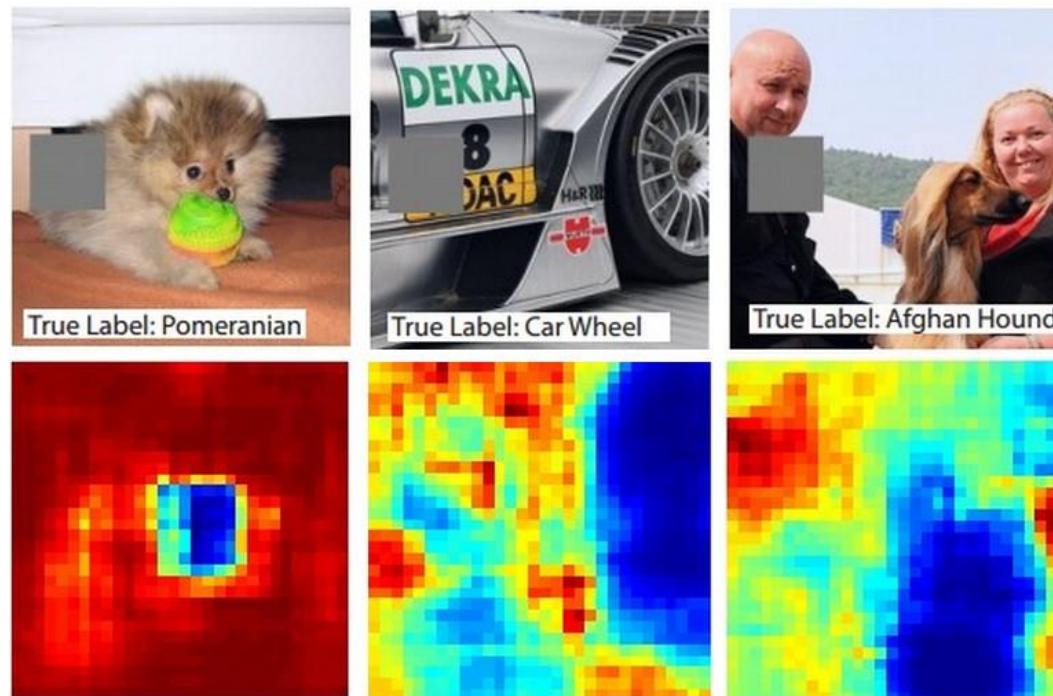
What Feature Layers Learn -Visualizing Strategies

- Method: #3: visualize the images to get an understanding of what the neuron is looking for in its receptive field (5th Layer of AlexNet)



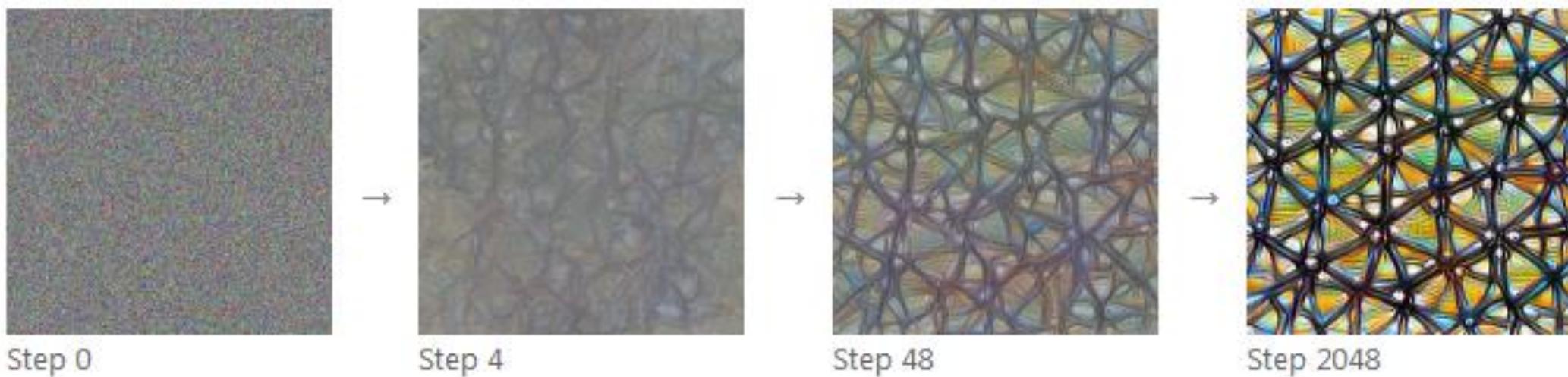
What Feature Layers Learn -Visualizing Strategies

- Method #4: Cover (by a $p \times p$ square) the input and plot the prediction probability as function of covering square



What Feature Layers Learn -Visualizing Strategies

- Method #5: Optimize input to activate a particular Neuron/Layer/....
- For Example: Maximizing (Neuron #11, Layer mixed4a,GoogleNet):



- See: <https://distill.pub/2017/feature-visualization/appendix/>

What Feature Layers Learn -Visualizing Strategies

- Method #5: Optimize input to activate a particular Neuron/Layer/....
- A DNN paints (DeepDream by Google)



Baseball—or stripes?
mixed4a, Unit 6



Animal faces—or snouts?
mixed4a, Unit 240



Clouds—or fluffiness?
mixed4a, Unit 453



Buildings—or sky?
mixed4a, Unit 492

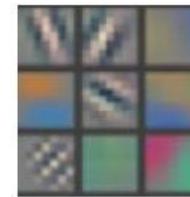
What Feature Layers Learn -Visualizing Strategies

- Method #5: Optimize input to activate a particular Neuron/Layer/....
- A DNN paints (DeepDream by Google)

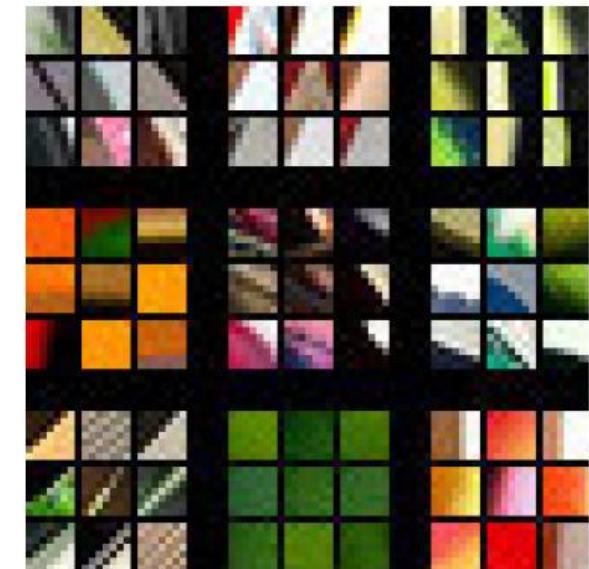


What Feature Layers Learn

- Layers Learn High Level Features
 - Top: Nine 7x7 filters in the 1st ConvLayer
 - Bottom: grid of 7x7 patches from actual images which maximally activated each of the 9 filters.

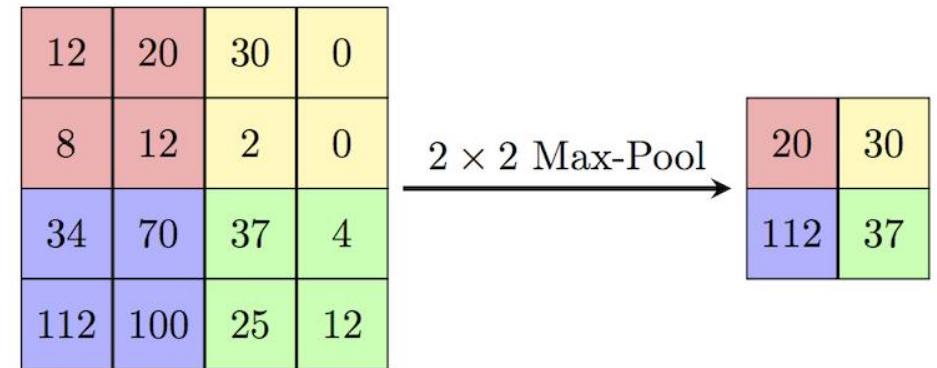


Layer 1

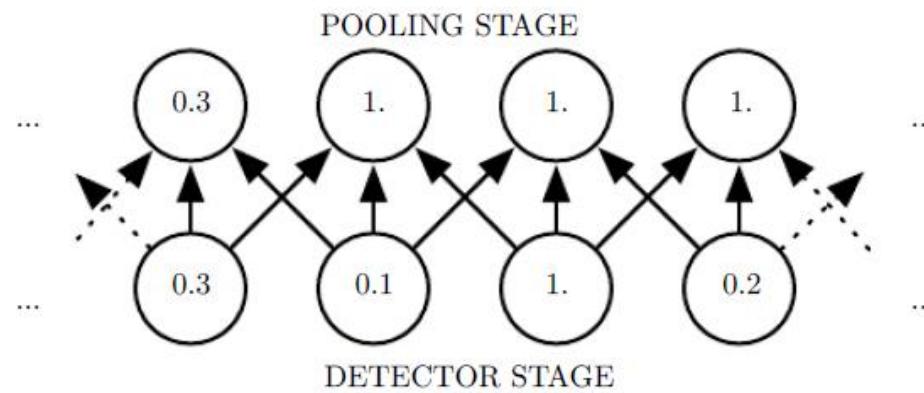
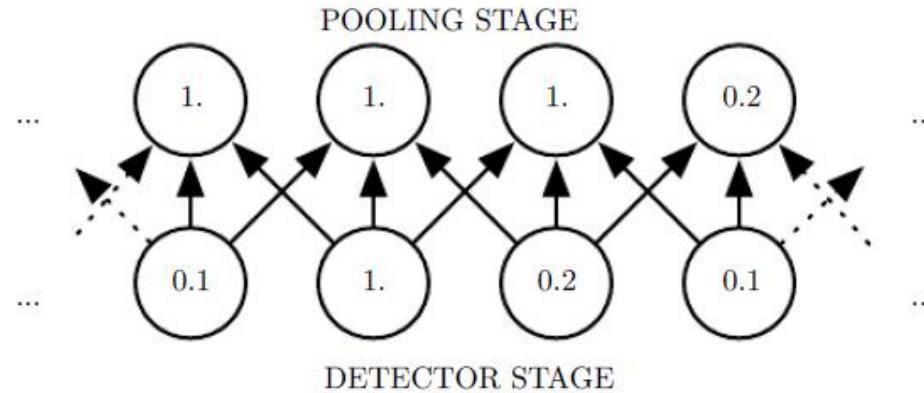


Operators - Pooling

- Modify and Subsample output of detector (ReLU):
 - Using neighbors
 - Max/min pooling
 - Means pooling
 - Median pooling
 - L_p -Pooling
 - Weighted Mean pooling
 - Make Invariant to small variation (like shift)
 - Reduce # of parameters
 - Regularization

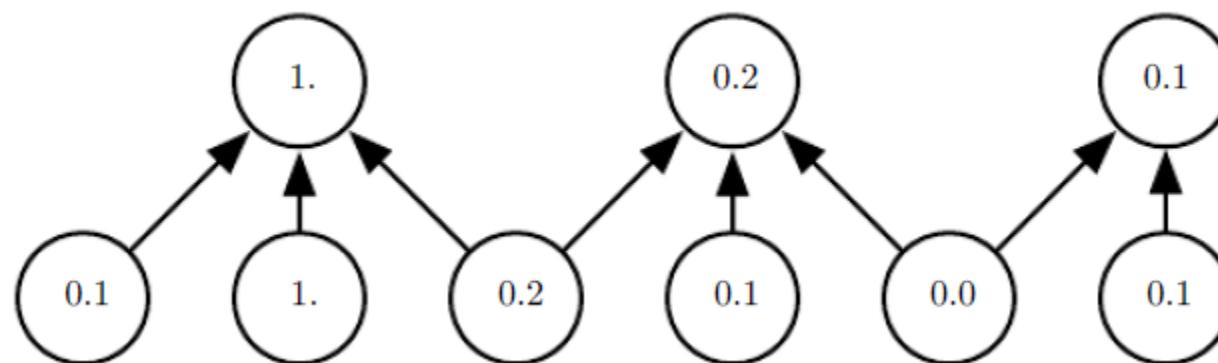


Pooling Example – One Pixel Shift



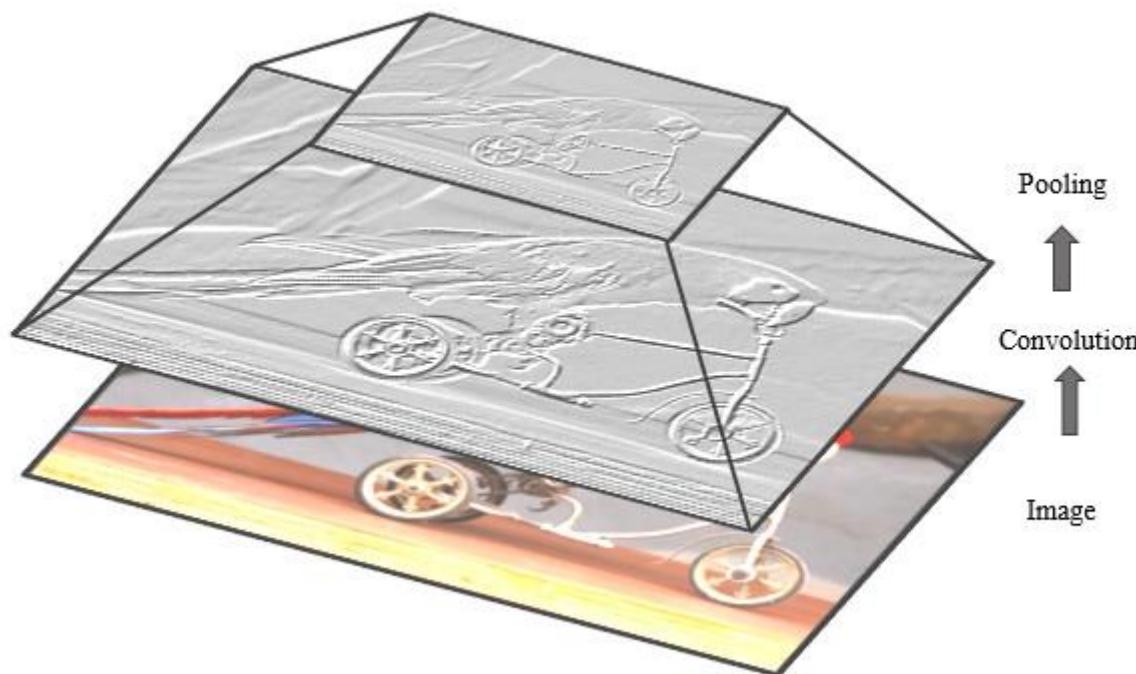
Pooling with Downsampling

- With or Without overlap/stride Pooling



Pooling Layer

- Pooling Effect



Pooling Strategies

- Deterministic
- Stochastic

Pooling Strategy - Deterministic

- Max pooling, idea

$$a_{kij} = \max_{(p,q) \in R_{ij}} (a_{kpq})$$

- Mean pooling:

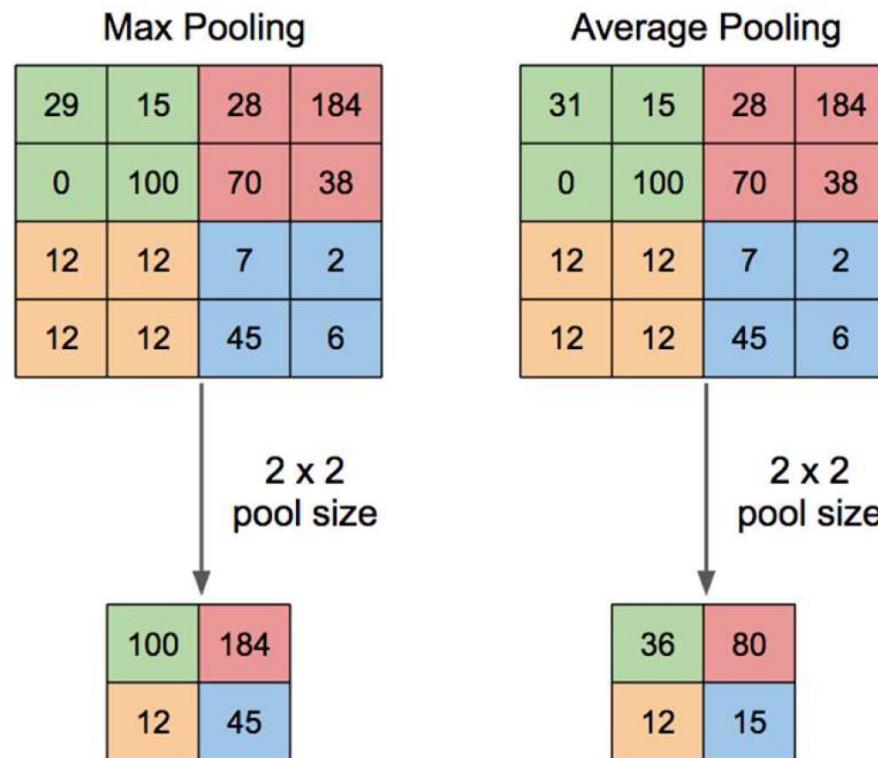
$$a_{kij} = \frac{1}{|R_{ij}|} \sum_{(p,q) \in R_{ij}} a_{kpq}$$

- Mixed pooling (λ : a random or deterministic parameter):

$$a_{kij} = \lambda \max_{(p,q) \in R_{ij}} (a_{kpq}) + (1 - \lambda) \frac{1}{|R_{ij}|} \sum_{(p,q) \in R_{ij}} a_{kpq}$$

Pooling Strategy - Deterministic

- Max Pooling vs Mean (Average) Pooling:

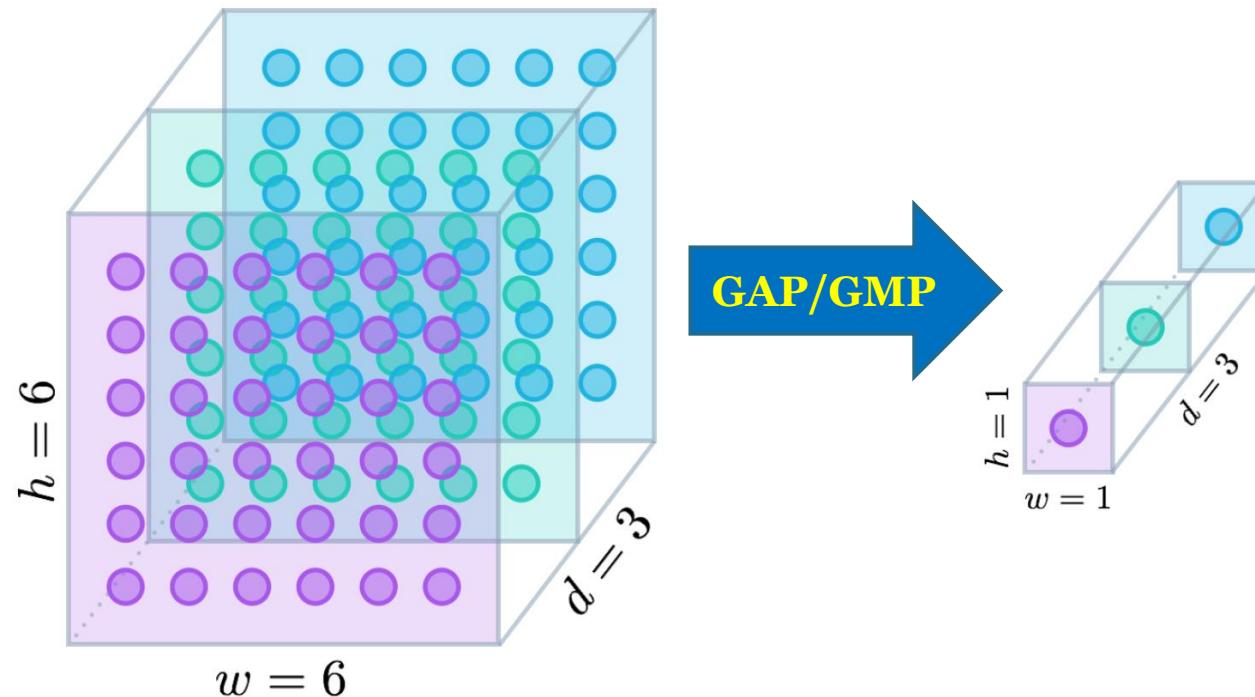


Pooling Strategy - Deterministic

- (Discrete Cosine Transform Spectral Pooling Layers for Convolutional Neural Networks)-ICAISC 2018
- **Spectral Pooling:**
 - C : # of channels,
 - Input: $X \in \mathbb{R}^{C \times m \times n}$
 - Output: $\hat{X} \in \mathbb{R}^{C \times p \times q}$
 - $Y = \text{Transform}(X) \in \mathbb{R}^{C \times m \times n}$, Transform: DFT/DCT/Wavelet/...
 - May be Transform and filter!
 - *ProperTruncate* Y to $\hat{Y} \in \mathbb{R}^{C \times p \times q}$
 - $\hat{X} = \text{InverseTransform}(\hat{Y}) \in \mathbb{R}^{C \times p \times q}$

Pooling Strategy - Deterministic

- Global Averaging Pooling (GAP) and Global Max Pooling (GMP)
 - Just before FC layer



Pooling Strategy - Stochastic

- Stochastic (Stochastic Pooling for Regularization of Deep Convolutional Neural Networks- ICLR2013)
- Roulette wheel Pooling (RWP), pick a random activation from region of pooling, activity related probability:

$$P_i = \frac{a_i}{\sum_{k \in Region} a_k}$$



- To avoid some noisy behavior:

$$S_j = \sum_{i \in R_j} a_i p_i$$

Pooling Strategy - Stochastic

- Results:

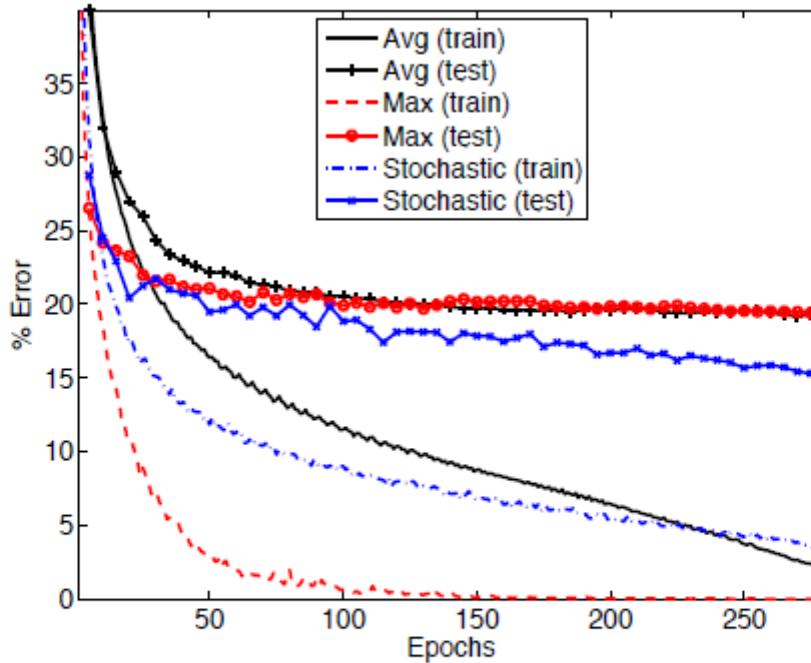


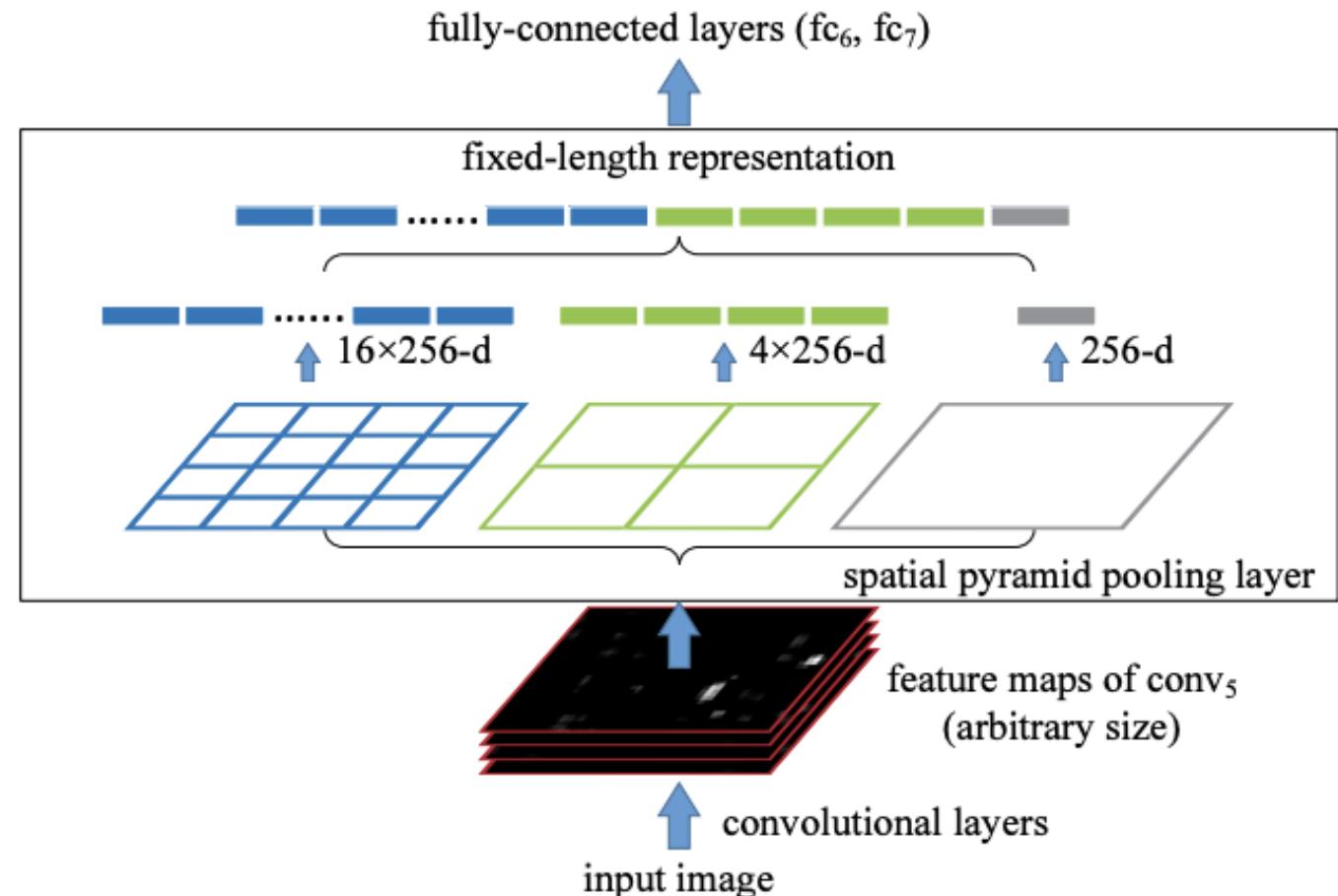
Figure 3: CIFAR-10 train and test error rates throughout training for average, max, and stochastic pooling. Max and average pooling test errors plateau as those methods overfit. With stochastic pooling, training error remains higher while test errors continue to decrease.¹

Pooling Strategy - Stochastic

- Stochastic:
 - Tournament pooling: Take any two (tournament size) nonzero activations with equal probability, and select larger one.
 - Fractional Max Pooling (FP): Random Region size and Location!

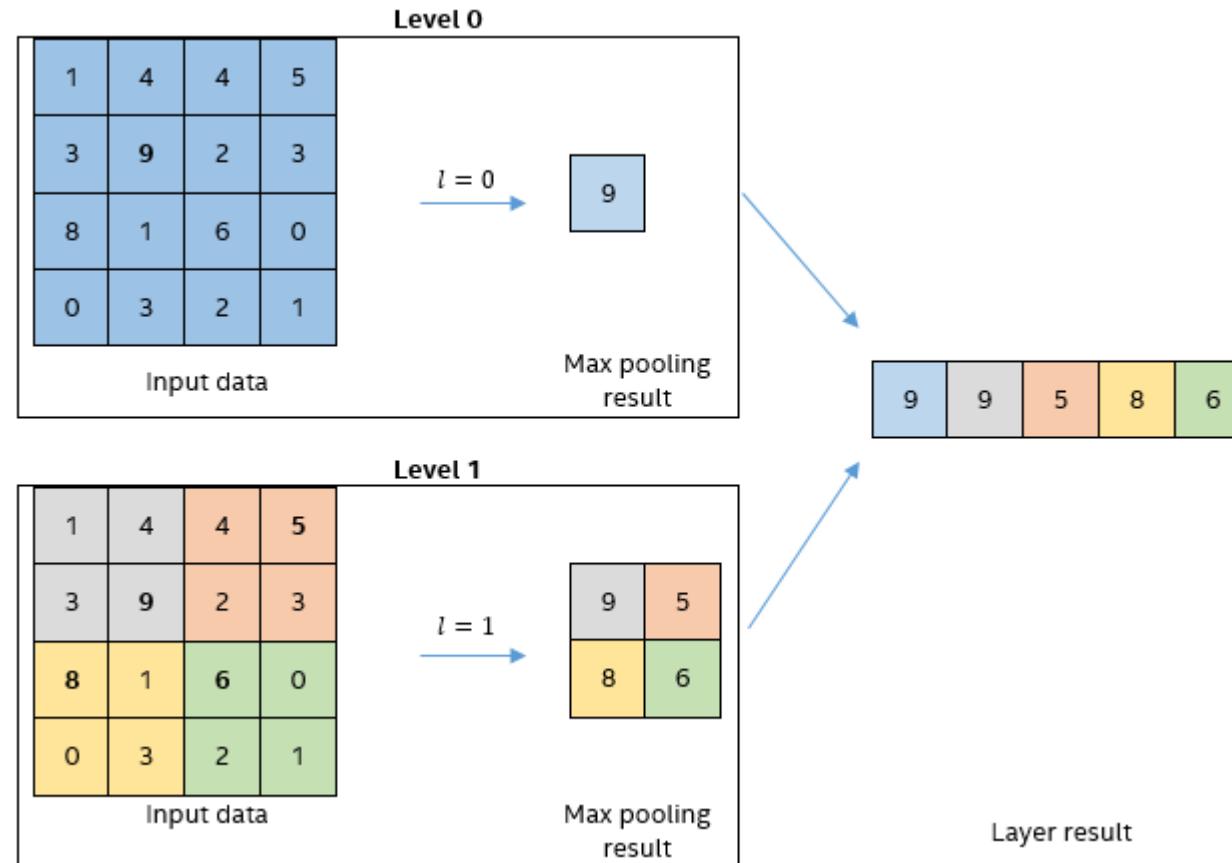
Pooling Strategy - Spatial Pyramid Pooling

- Spatial Pyramid Pooling
 - 256 is # of filters
 - FCL has fixed input size!



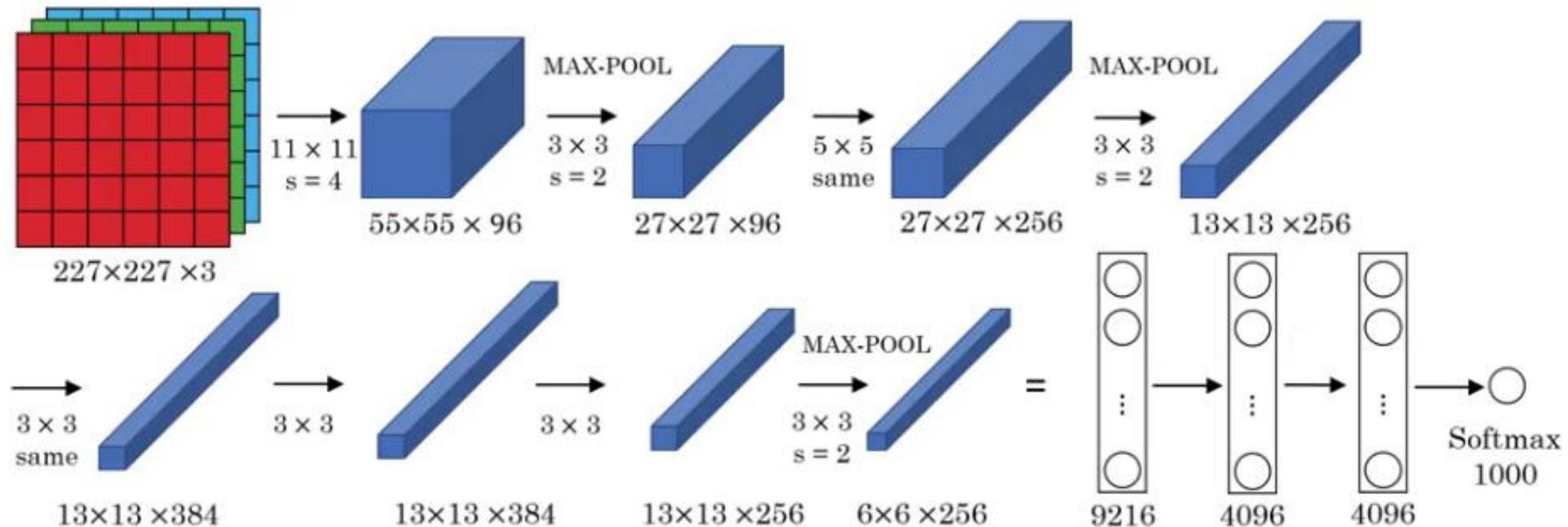
Pooling Strategy - Spatial Pyramid Pooling

- Example:



A Taste! AlexNet

- AlexNet
 - 1st Layer: $(11+4*(k-1))=227 \rightarrow k=55$



A Challenging Problem: IMAGENET!

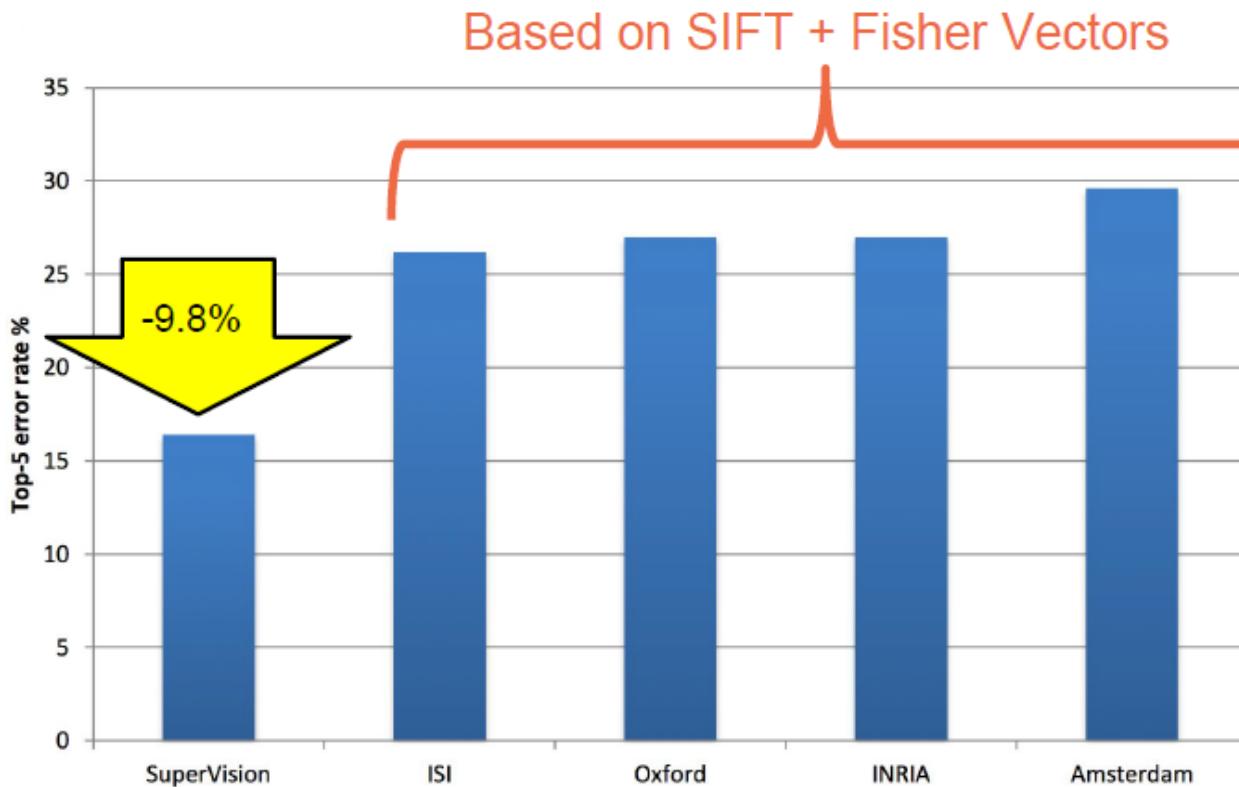
- 1000 Classes
- Train: 1.2 M
- Test: 100 K

IMAGENET



A Challenging Problem: IMAGENET!

- ImageNet ILSRVC 2012: (<https://cs.nyu.edu/%7Efergus/pmwiki/pmwiki.php>)



AlexNet, Alex Krizhevsky et al. 2012

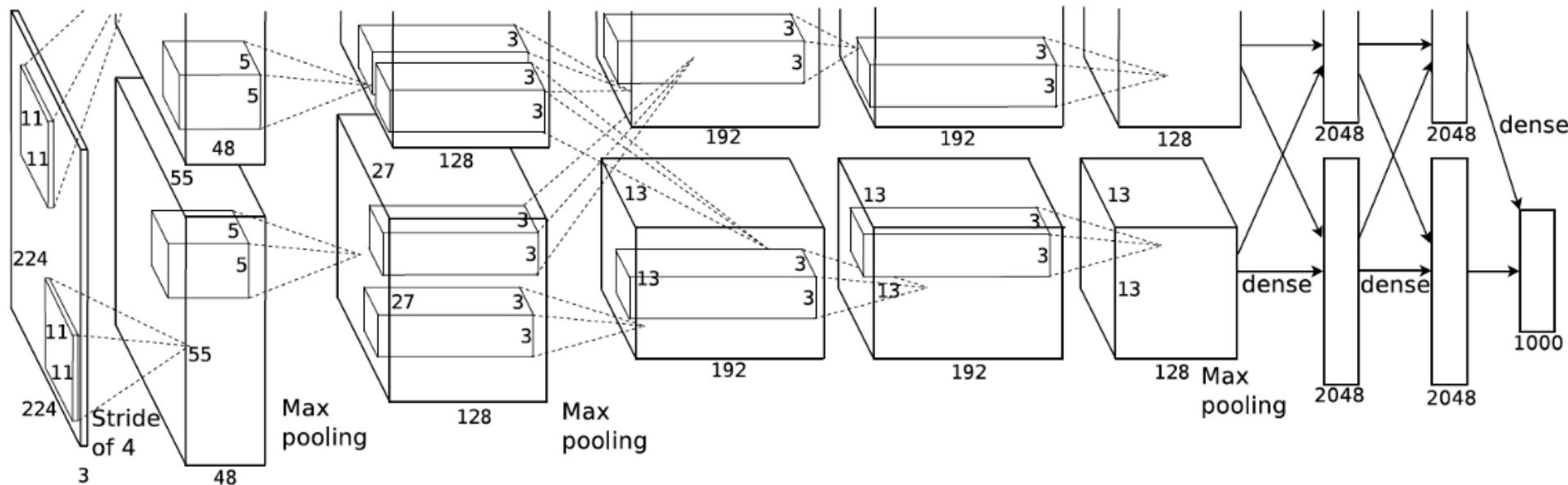
- In Brief:
 - ReLU (x6 faster than tanh)
 - Regularization: 50% Dropout (x2 training time)
 - 5 CL (Convolution Layer), 11×11, 5×5, 3×3
 - 3 FC (Fully Connected), 4096-4096-1000
 - 3 MaxPool Layer: 3×3
 - ReLu after all CL and FC
 - Image Size: 256x256
 - Input Size: 227x227x3

AlexNet, Alex Krizhevsky et al. 2012

- CONV1/MAX POOL1/NORM1
- CONV2/MAX POOL2/NORM2
- CONV3
- CONV4
- CONV5/Max POOL3
- FC6
- FC7
- FC8

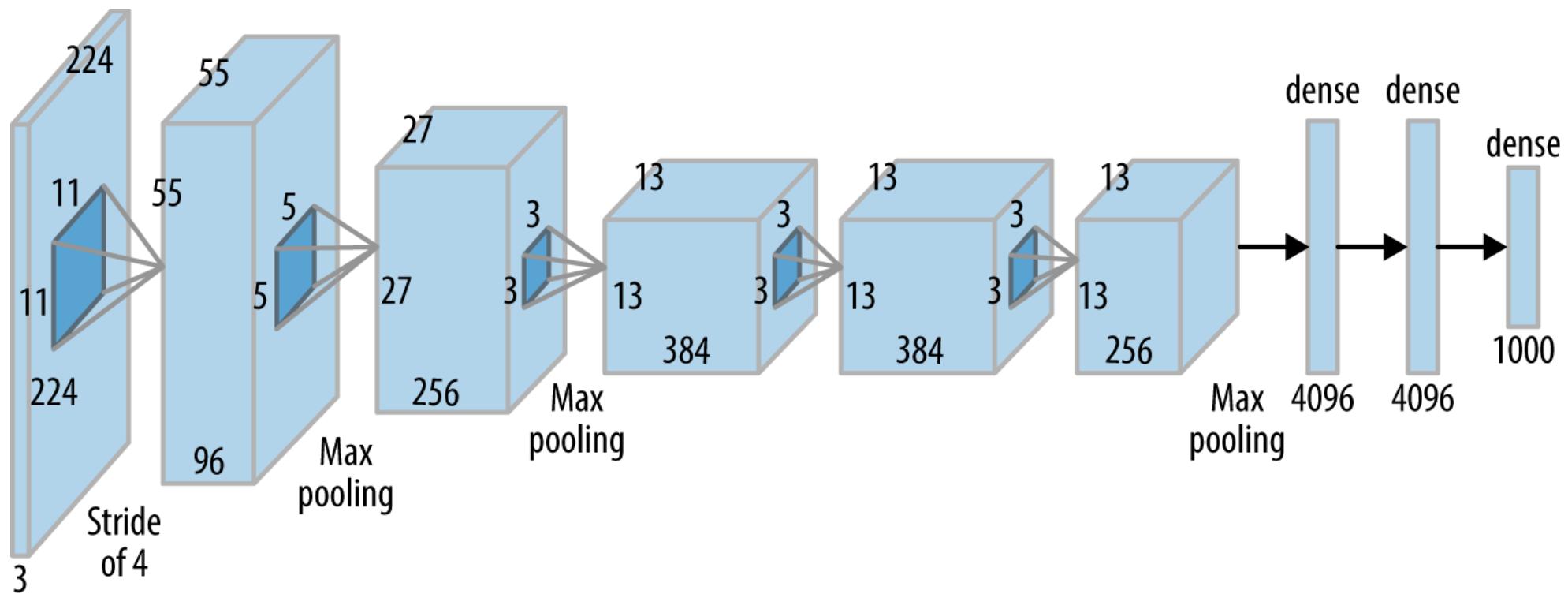
AlexNet, Alex Krizhevsky et al. 2012

- Original (paper) Figure:



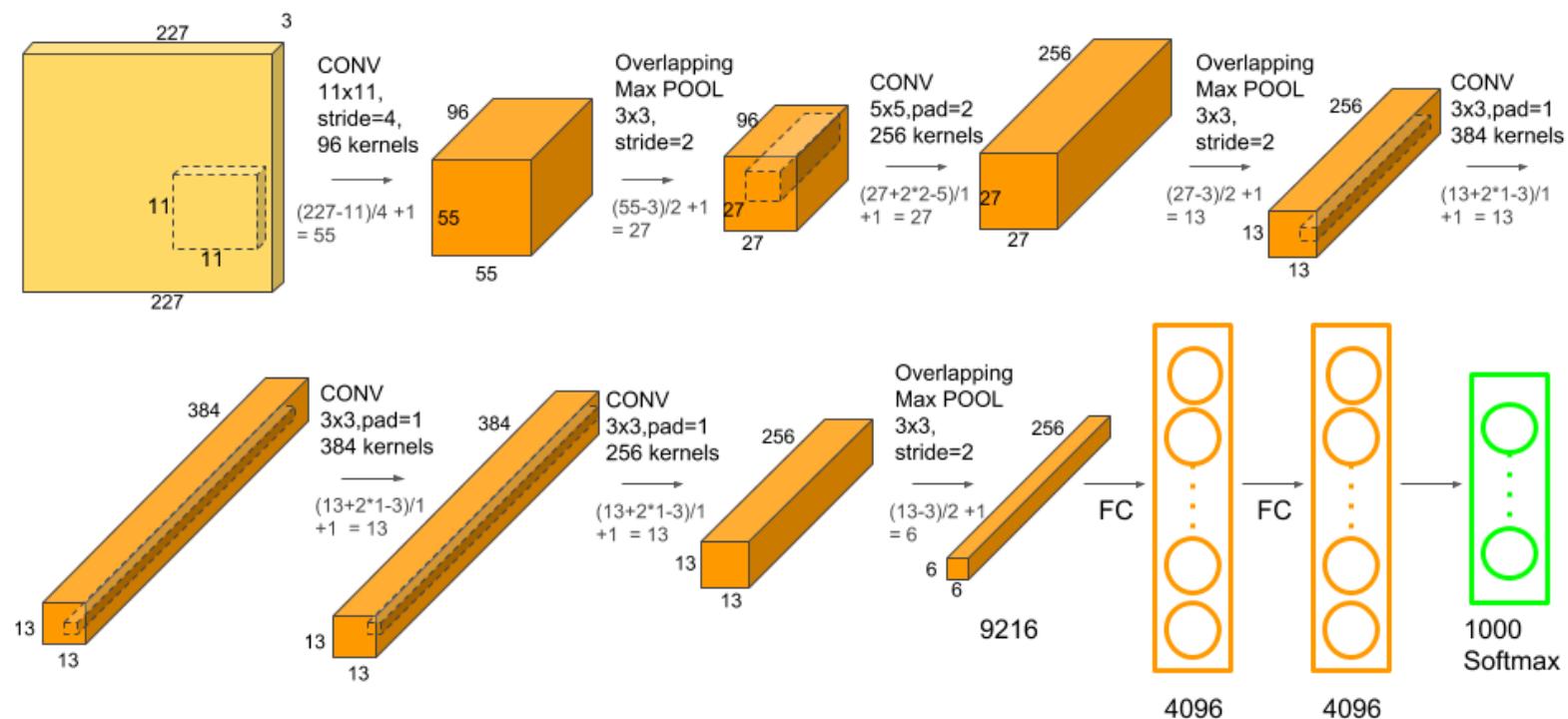
AlexNet, Alex Krizhevsky et al. 2012

- Better one:



AlexNet, Alex Krizhevsky et al. 2012

- [https://www.learnopencv.com/understanding-alexnet/\(\)](https://www.learnopencv.com/understanding-alexnet/)



AlexNet 2012 -Details

- Detail:
- First use of ReLU
- Normalize ReLu Output
- Several Data Augmentation
- Dropout: 0.5
- Batch Size: 128
- SGD Momentum: 0.9
- Learning rate 10^{-2} , reduced by 10
- ~60M parameters

AlexNet 2012 – Data Augmentation

- Random patches 224×224 (227×227 is true) from 256×256 input images
- Horizontal Reflection of all patches
- Add AGWN (Additive Gaussian White Noise) as follow:
 - PCA on RGB (Training set):

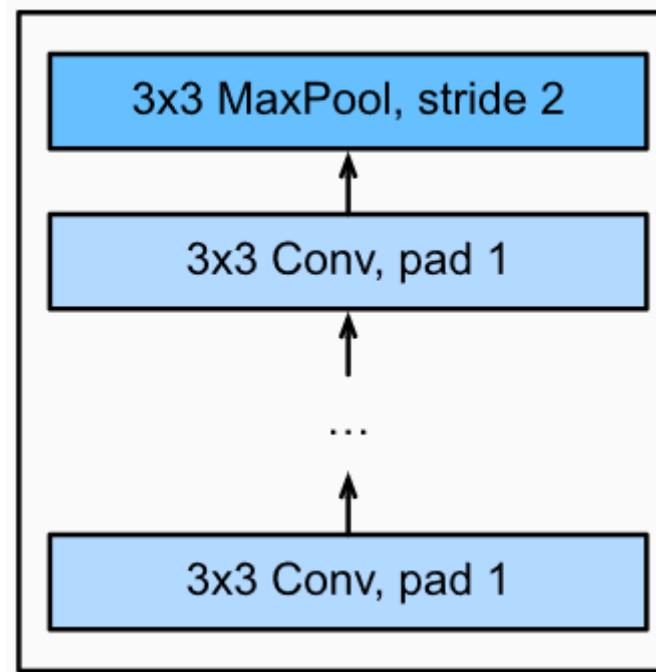
$$n_1\lambda_1\mathbf{v}_1 + n_2\lambda_2\mathbf{v}_2 + n_3\lambda_3\mathbf{v}_3$$
$$n_i \propto N(0,0.1)$$

VGGnet 16&19, Simonyan & Zisserman, 2014

- VGG: Visual Geometry Group
- Idea: Smaller filter but deeper!
 - 8 Layers in Alexnet → (16-19) ConvLayers
 - ConvLayer (16/19): Only 3×3 , Stride: 1, Padding: 1
 - More Non-linearity
 - MaxPooling (5): 2×2 , Stride: 2
 - 138M Parameters
 - 3×3 and deeper $\sim 7 \times 7$ shallower
 - $(3 \times 3)^*(3 \times 3)^*(3 \times 3) \sim 7 \times 7$ (Receptive Field)

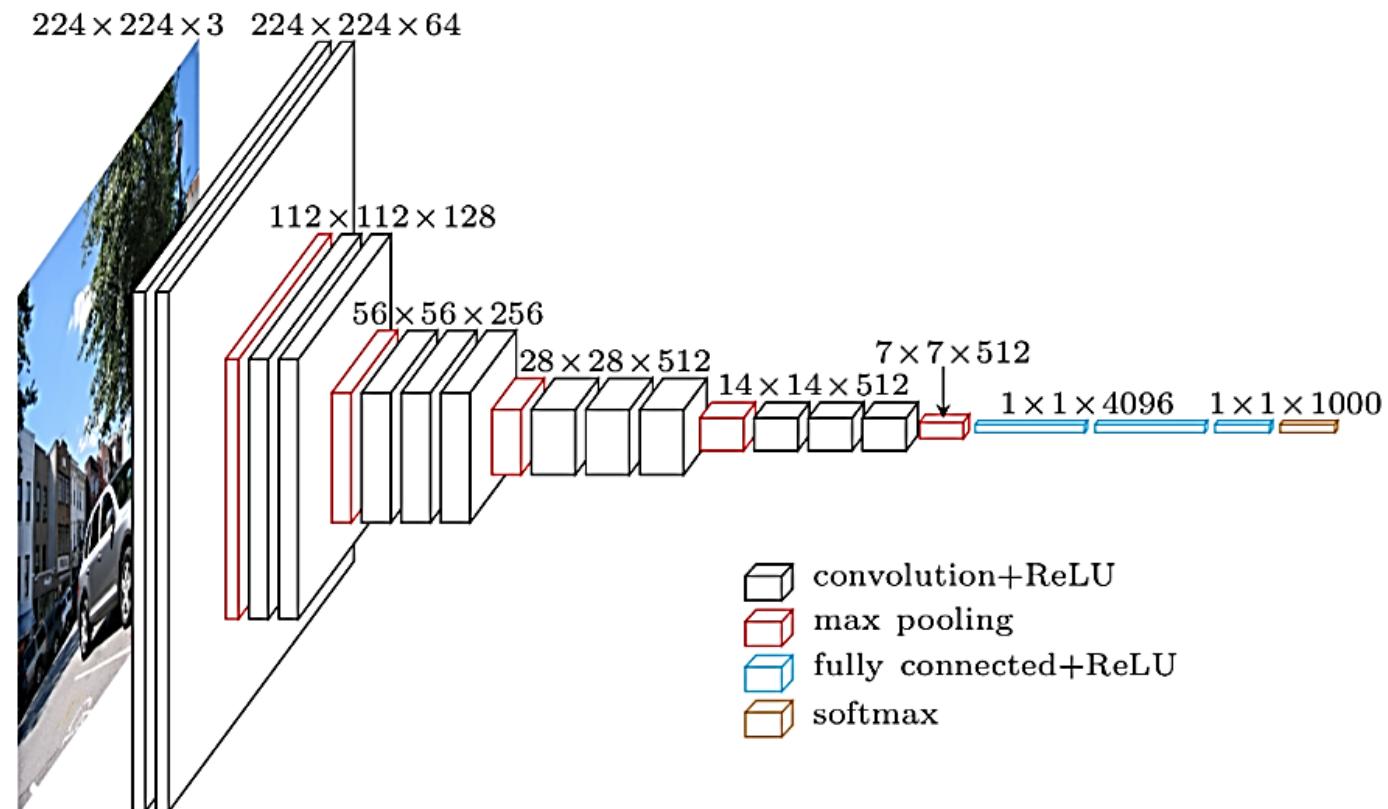
VGGnet 16&19, Simonyan & Zisserman, 2014

- VGG Main Block:



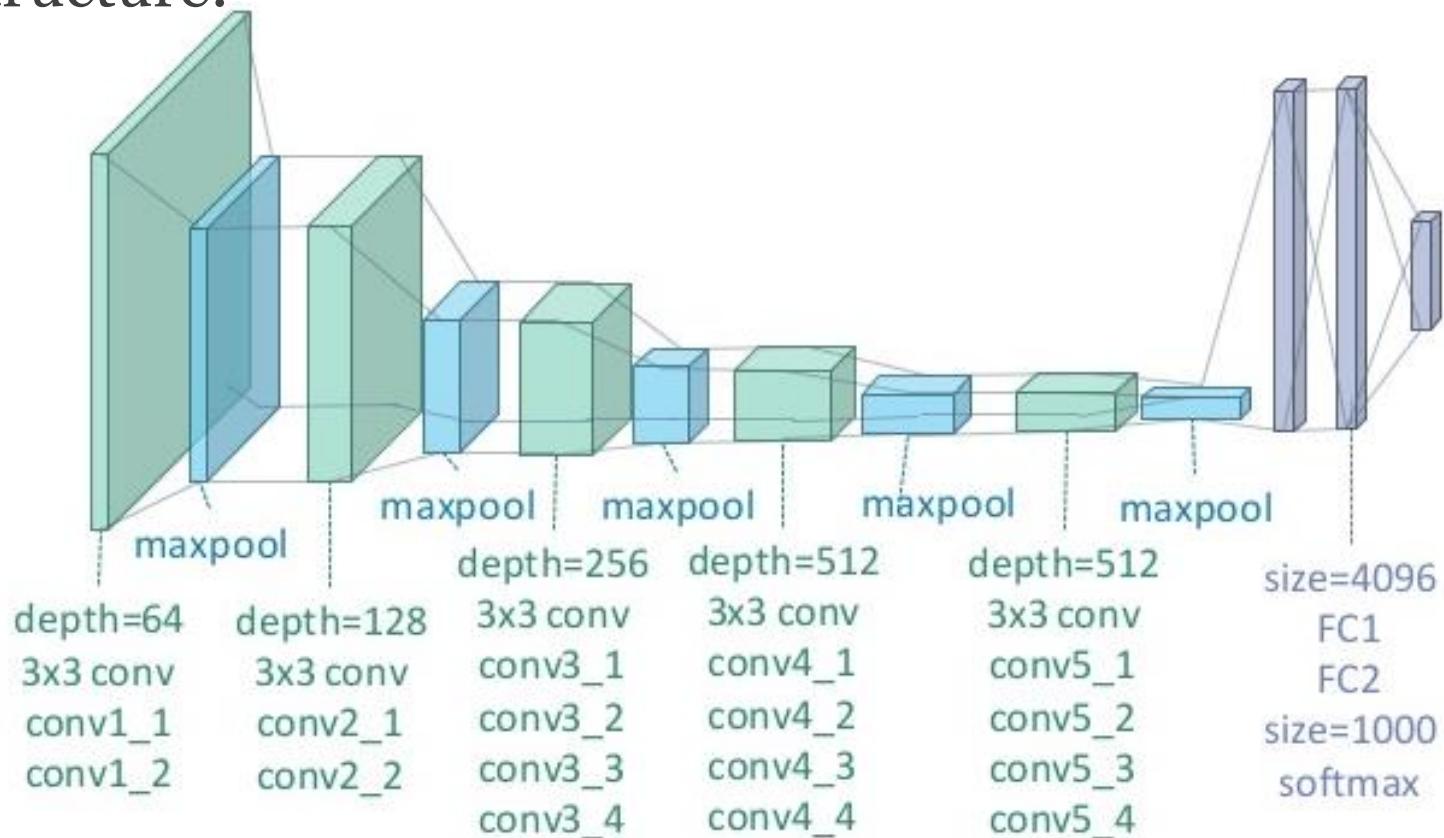
VGGnet 16, Simonyan & Zisserman, 2014

- VGG16 Structure:
- Input to 1st FC:
 - $7 \times 7 \times 512 = 25088$



VGGnet 16&19, Simonyan & Zisserman, 2014

- VGG19 Structure:

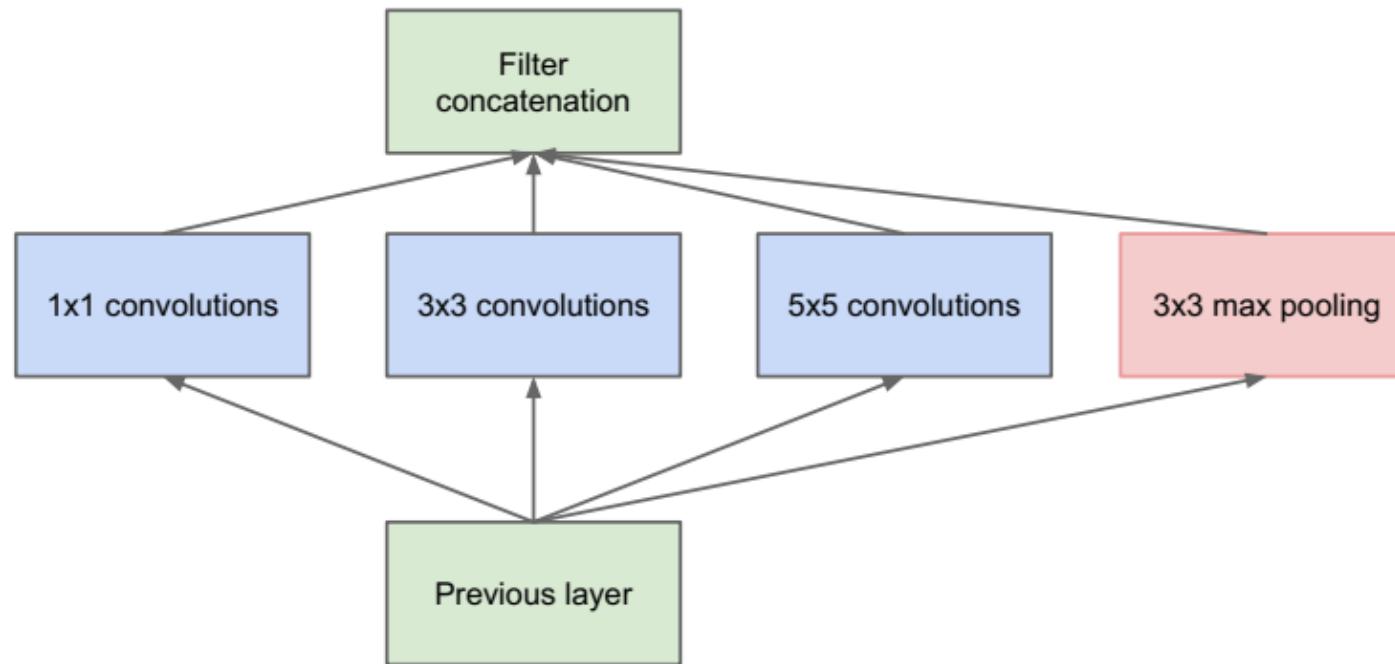


GoogLeNet - Going deeper with convolutions, 2014

- Inception Module V1
- Idea and Motivation:
 - Multiresolution Analysis (avoid cascade only convolution)
 - Global Average Pooling before FC (dimension reduction)
 - Bottleneck Layer (one dimension reduction)
 - Auxiliary Classifier (shallow/weak output)

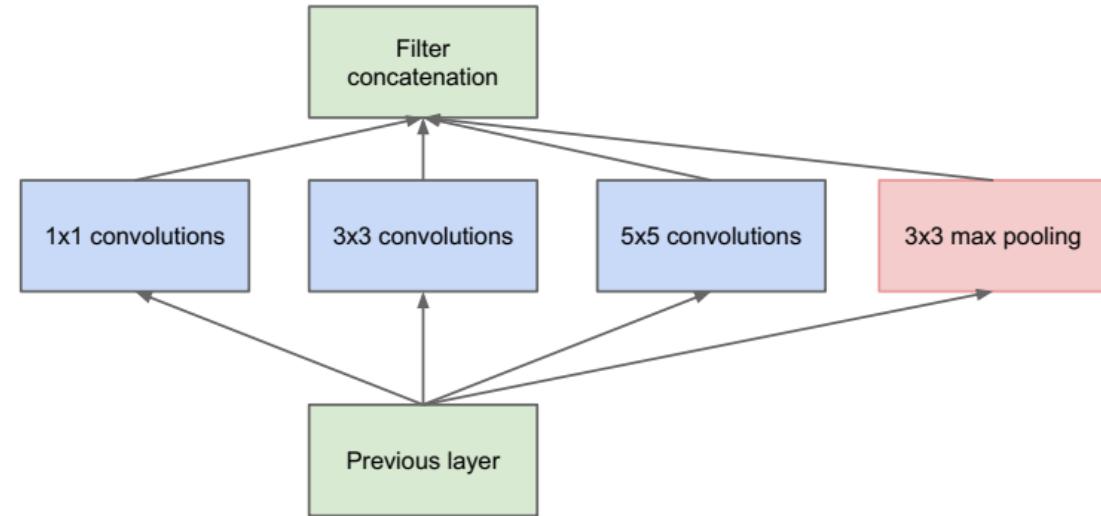
GoogLeNet - Going deeper with convolutions, 2014

- Idea#1: Why Resolution reduction in successive Layer?
- Naïve Inception Version:



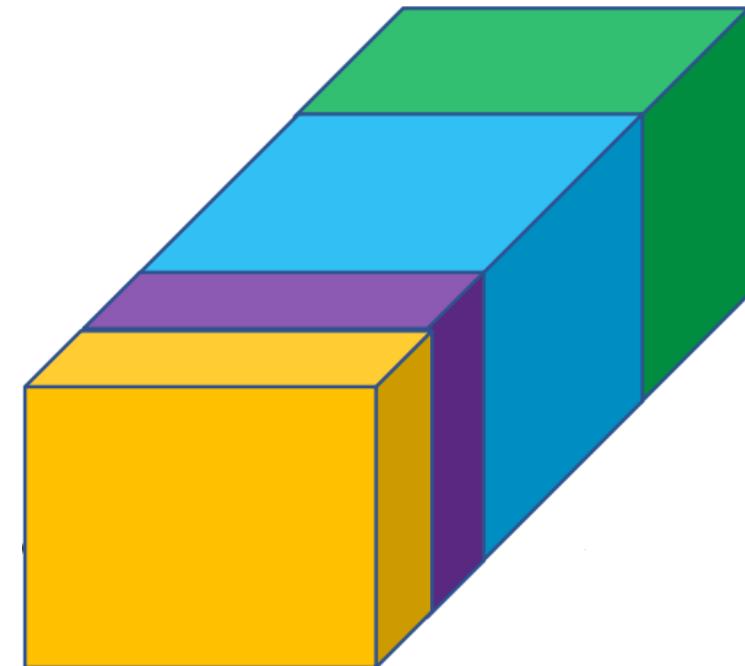
GoogLeNet - Going deeper with convolutions, 2014

- Inception V1 (Naïve Version):
- Previous Layer: $(28 \times 28) \times 256$
- $64 (1 \times 1) \rightarrow (28 \times 28) \times 64$
- $128 (3 \times 3) \text{ same} \rightarrow (28 \times 28) \times 128$
- $32 (5 \times 5) \text{ same} \rightarrow (28 \times 28) \times 32$
- $(28 \times 28) \text{ MaxPool } (3 \times 3), \text{ same} \rightarrow (28 \times 28) \times 256$
- Concatenated Tensor:
 - $(64 + 128 + 32 + 256) \times (28 \times 28) = 480 \times (28 \times 28) = 376320$



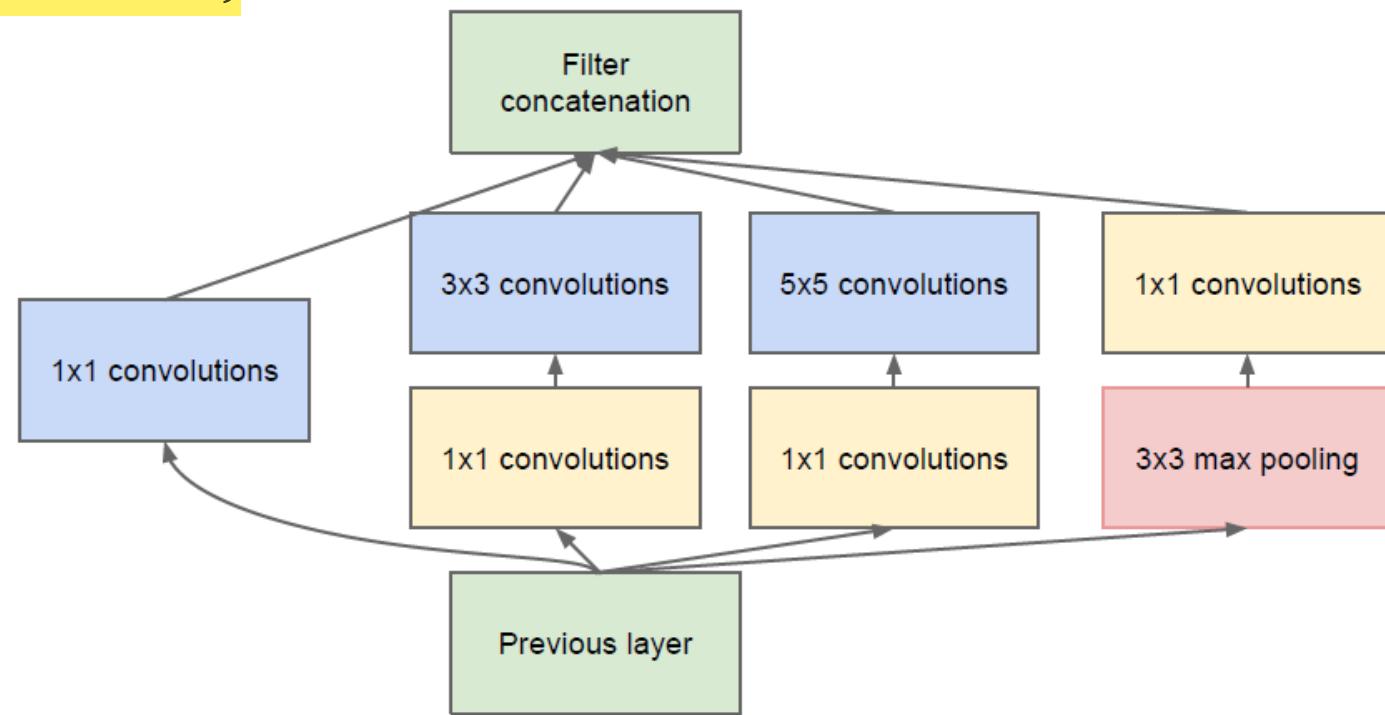
GoogLeNet - Going deeper with convolutions, 2014

- Idea#1: Resolution reduction in successive Layer
- Concatenated Tensor:
 - $(64+128+32+256) \times (28 \times 28) = 480 \times (28 \times 28) = 376320$
 - Memory and Computation Cost ☹



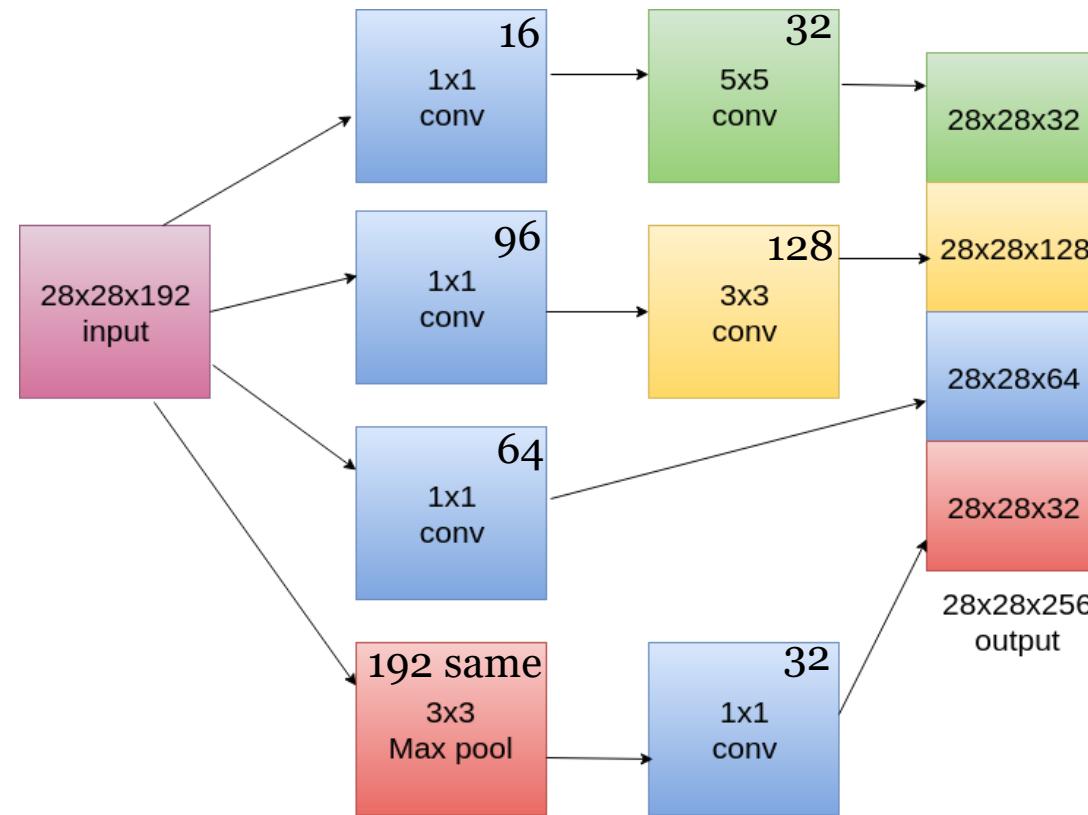
GoogLeNet - Going deeper with convolutions, 2014

- Idea#1: Inception V1 (Inception Module with dimension Reduction):
 - Bottleneck Layer (1×1 Convolution)
 - Reduce # of channels



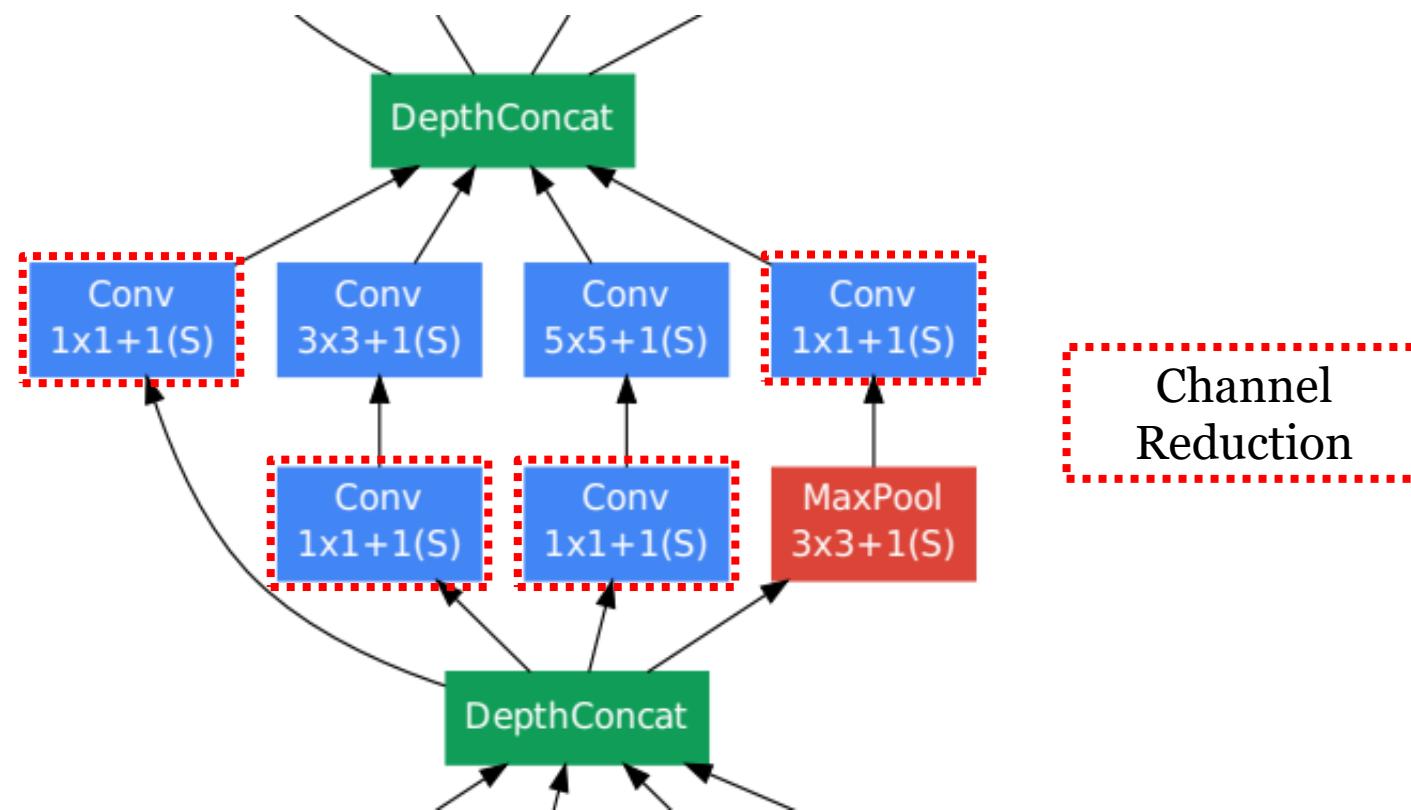
GoogLeNet - Going deeper with convolutions, 2014

- Example:



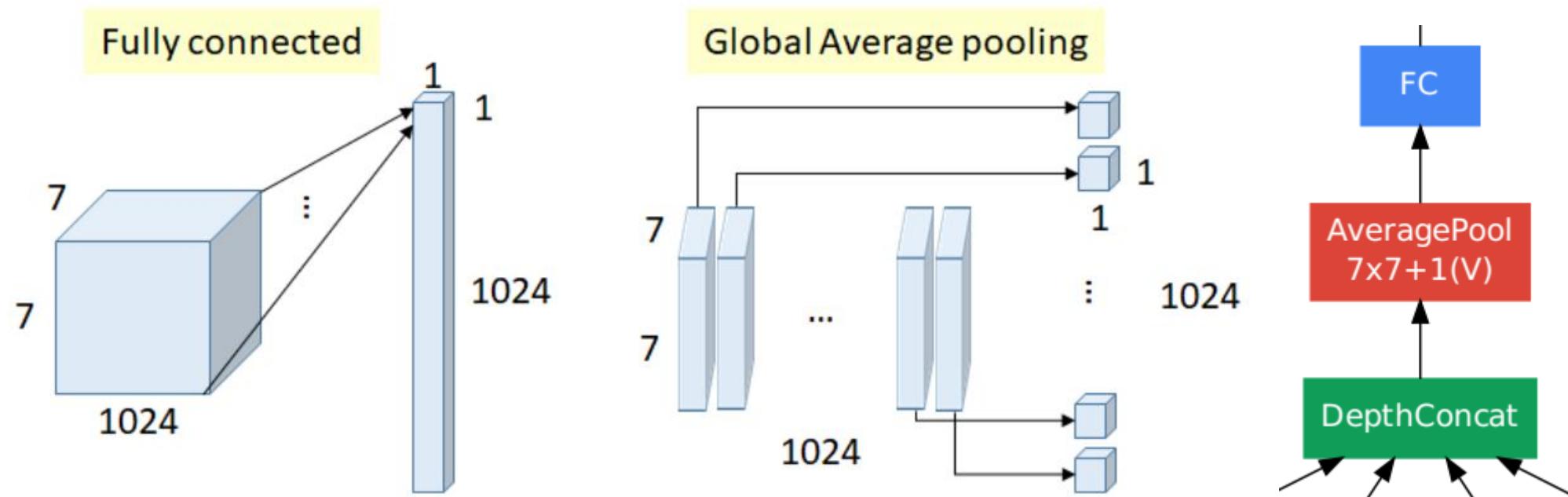
GoogLeNet - Going deeper with convolutions, 2014

- A Cross Section from GoogLeNet:



GoogLeNet - Going deeper with convolutions, 2014

- Idea#2: Global Average Pooling Before FC
- Input: $7 \times 7 \times 1024$
- Output: $1 \times 1 \times 1024$



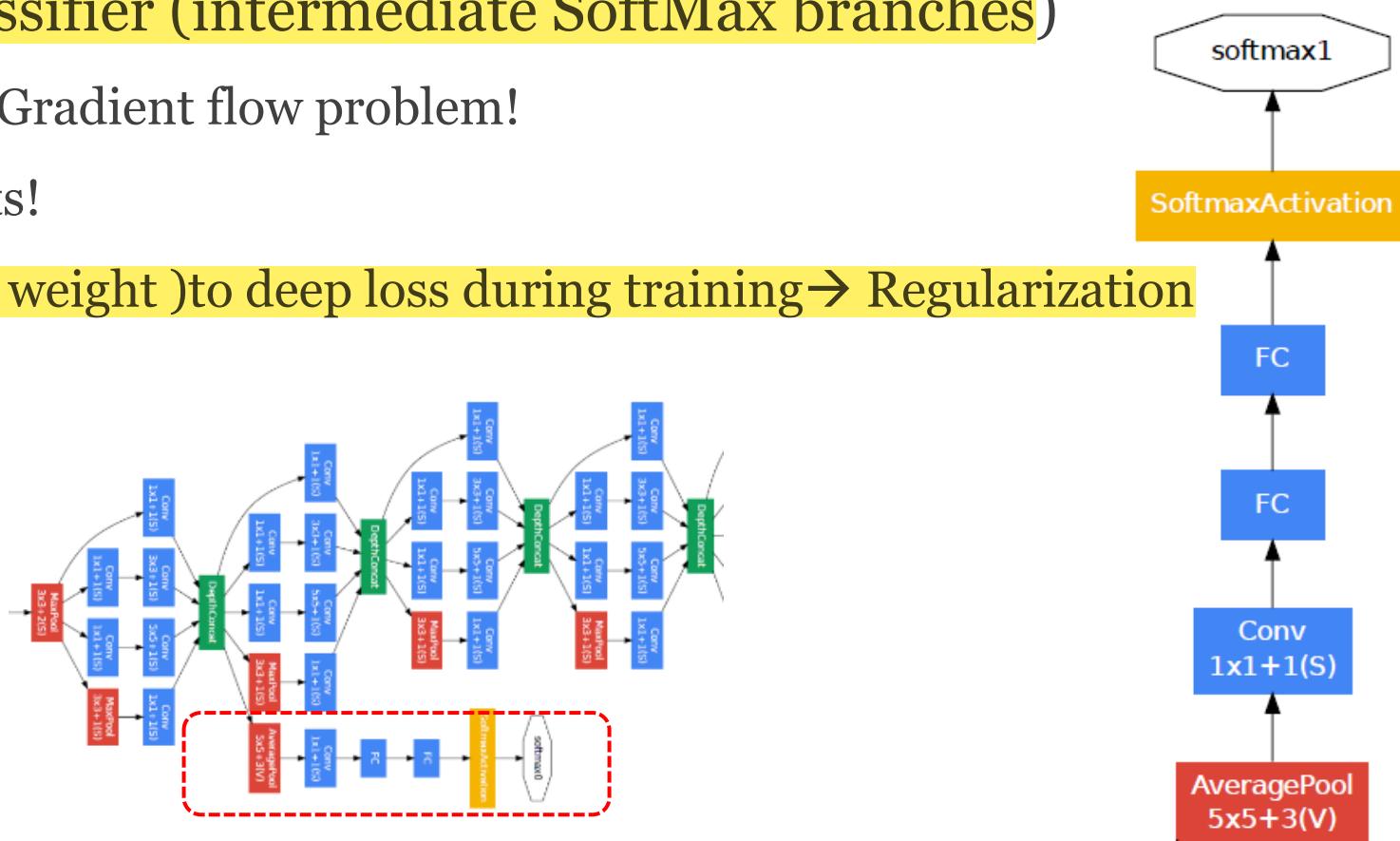
GoogLeNet - Going deeper with convolutions, 2014

- Idea #3: Auxiliary Classifier (intermediate SoftMax branches)

- Too Deep (22 Layers) → Gradient flow problem!

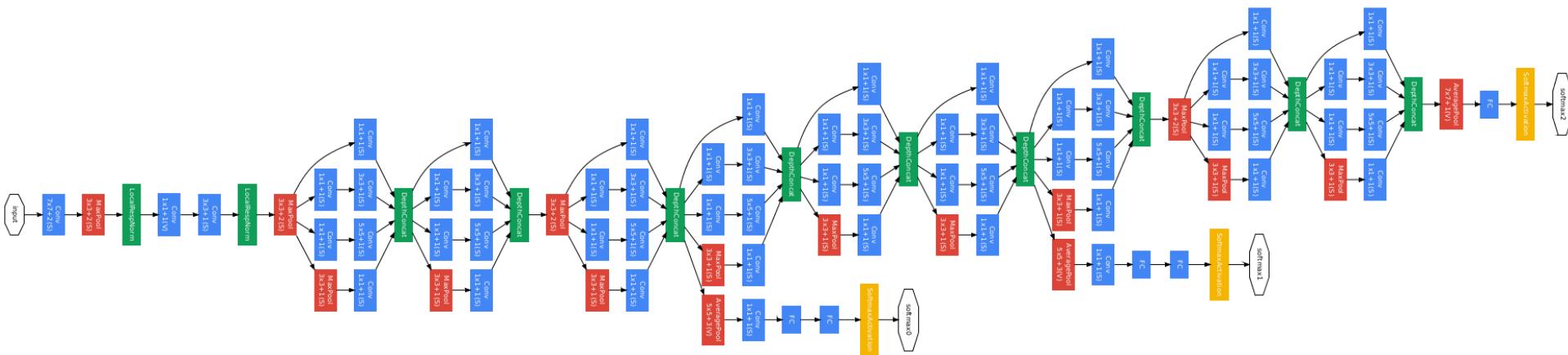
- Consider Shallow Outputs!

- How? Add their loss (0.3 weight) to deep loss during training → Regularization



GoogLeNet - Going deeper with convolutions, 2014

- Complete Structure:



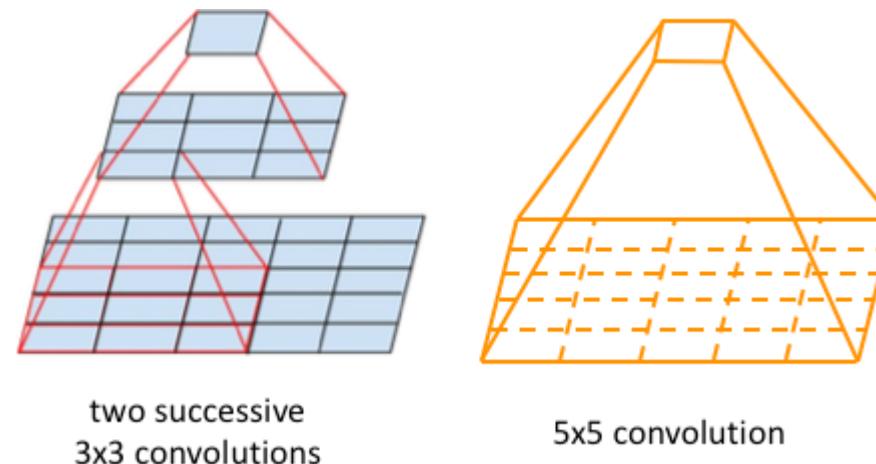
GoogLeNet - Going deeper with convolutions, 2014 Rethinking the Inception Architecture for Computer Vision, 2015

- Inception V2 and V3
- Motivation and Idea:
 - Factorizing Convolutions
 - Efficient Grid Size Reduction
 - Label Smoothing as Regularization

GoogLeNet - Going deeper with convolutions, 2014

Rethinking the Inception Architecture for Computer Vision, 2015

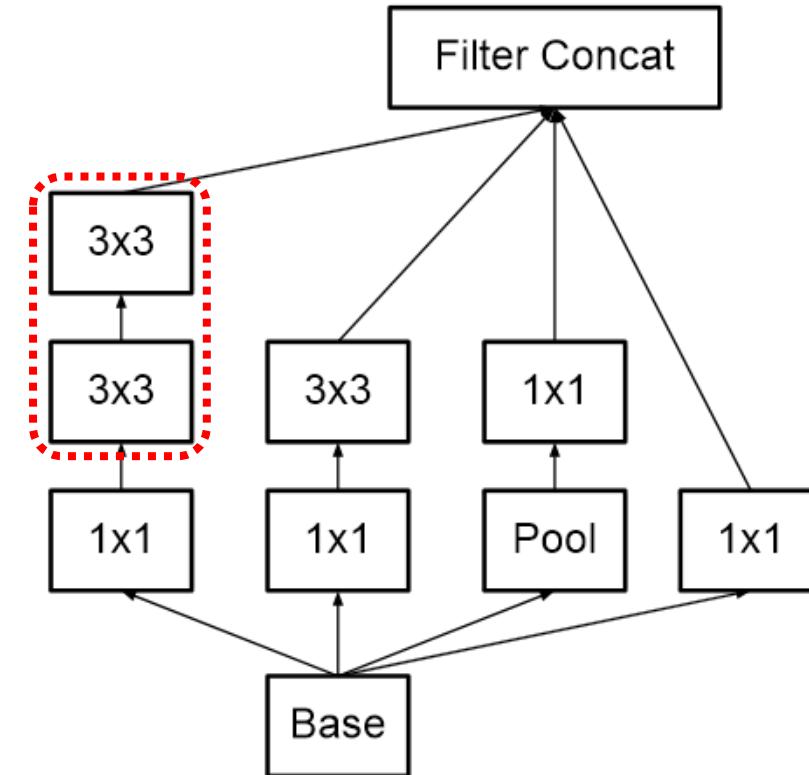
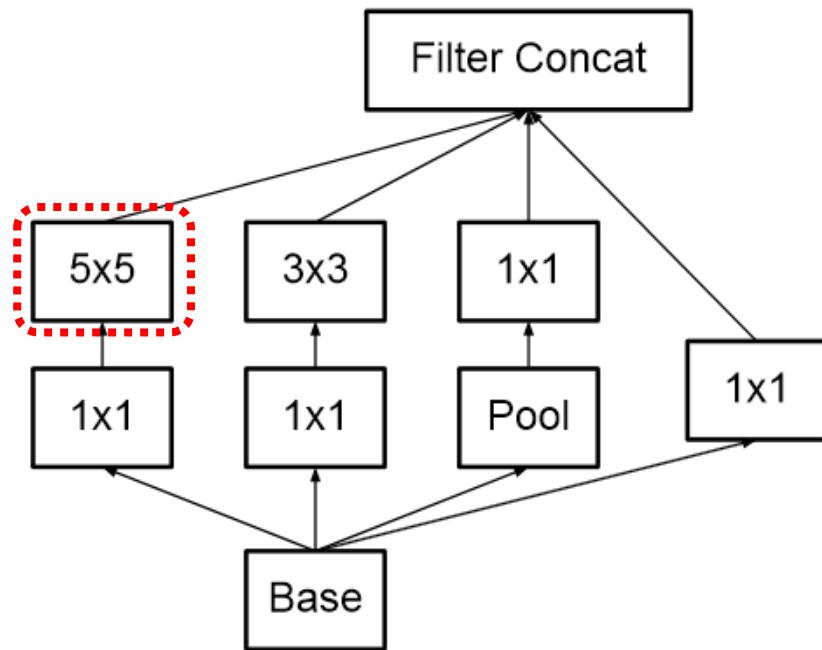
- Idea #1: (Symmetric Factorization)
 - A 5×5 Convolution \sim Two successive 3×3 convolution
 - # of weights Conv(5×5) $> 2 \times$ # of weights Conv(3×3), $(25 - 2 \times 9) / 25 = \%28$
 - # of weights Conv(3×3) $> 2 \times$ # of weights Conv(2×2), $(9 - 2 \times 4) / 9 = \%11$
 - Less parameter (28%), High Nonlinear ($2 \times$ ReLU)



GoogLeNet - Going deeper with convolutions, 2014

Rethinking the Inception Architecture for Computer Vision, 2015

- From Inception V1 to Inception V2

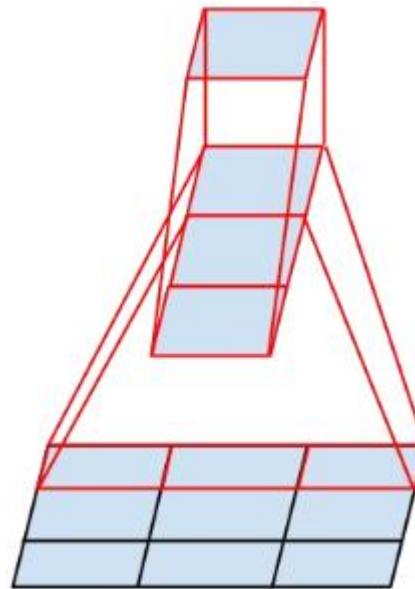


GoogLeNet - Going deeper with convolutions, 2014

Rethinking the Inception Architecture for Computer Vision, 2015

- Idea #1: More Factorization (**Asymmetric Factorization**)
 - A Conv (3×3) ~ Conv (3×1) Followed by Conv $(1 \times 3) \rightarrow (9-6)/6=33\%$
 - Less Parameter (33%), More Nonlinearity (2×ReLU)

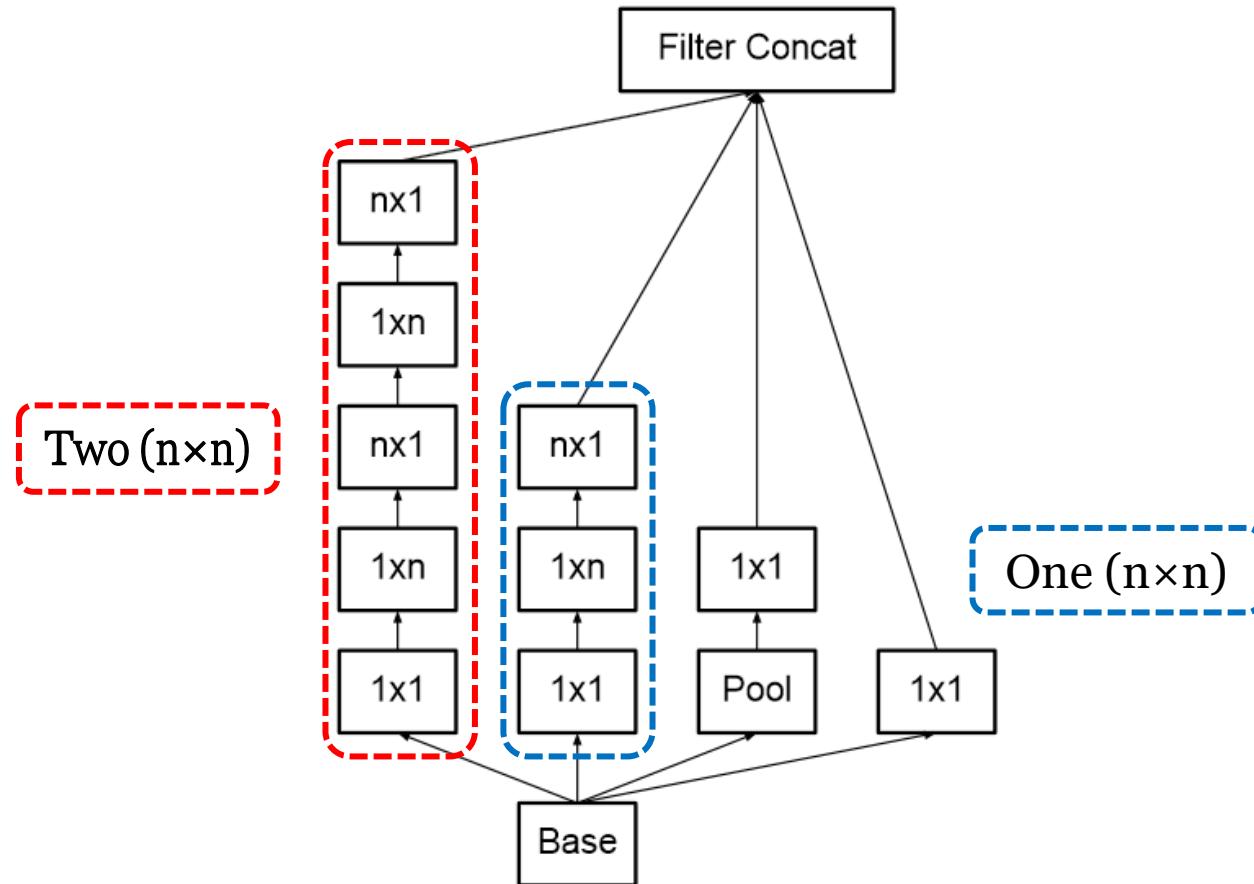
$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} * [4 \quad 5 \quad 6] = \begin{bmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{bmatrix}$$



GoogLeNet - Going deeper with convolutions, 2014

Rethinking the Inception Architecture for Computer Vision, 2015

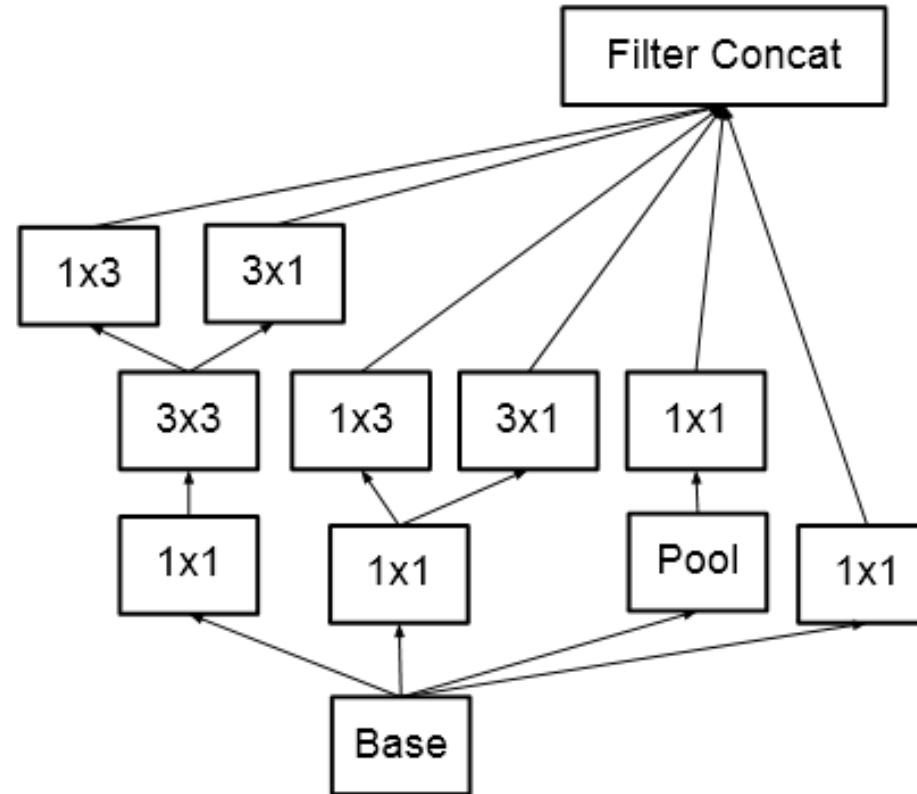
- Illustration:



GoogLeNet - Going deeper with convolutions, 2014

Rethinking the Inception Architecture for Computer Vision, 2015

- Make it wide (promote high dimensional representations)



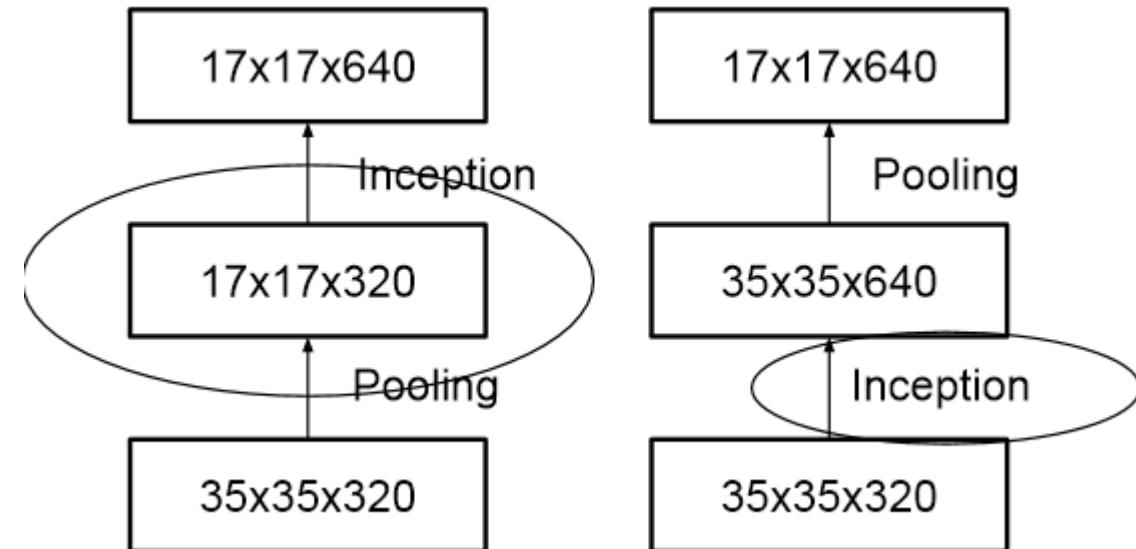
GoogLeNet - Going deeper with convolutions, 2014

Rethinking the Inception Architecture for Computer Vision, 2015

- Idea #2: Efficient Grid Size Reduction

- How to downsample?

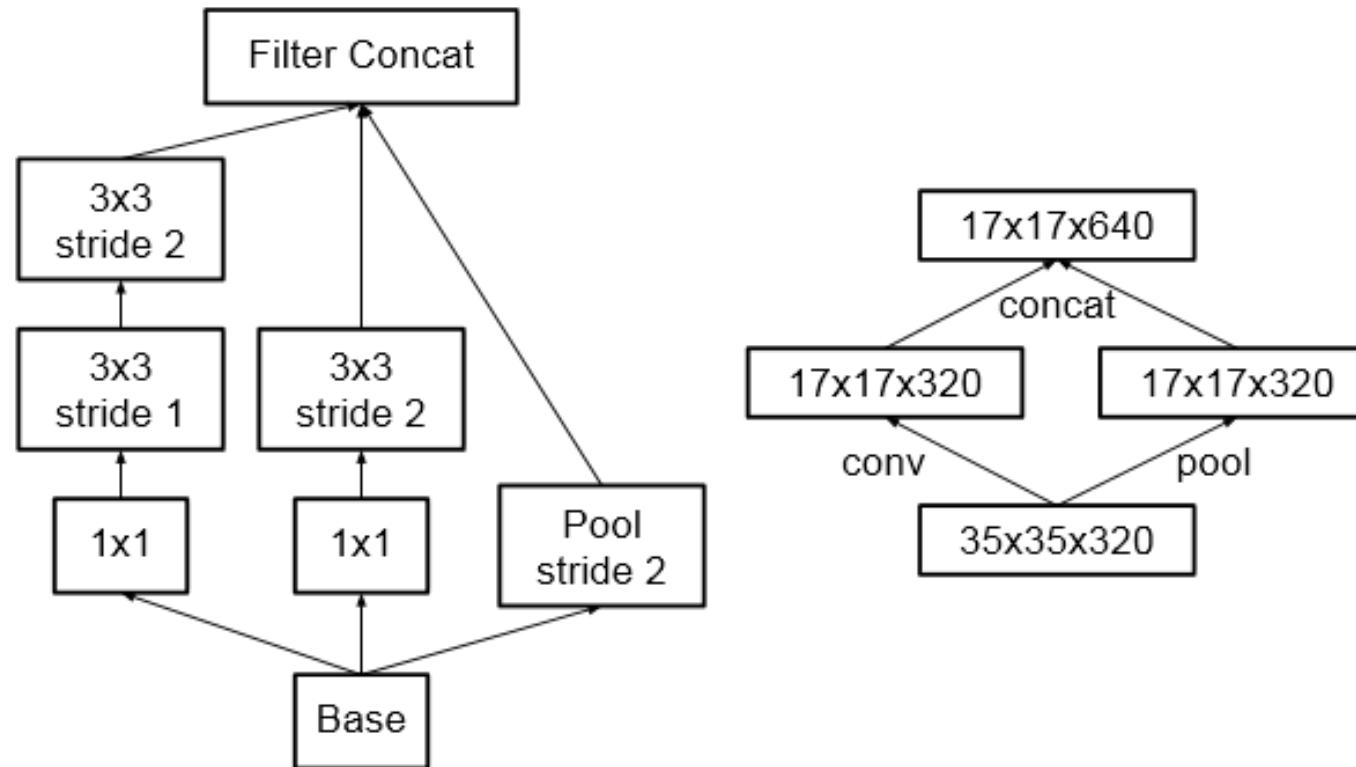
- Pooling then Convolution: Too Greedy (information loss)
 - Convolution then Pooling: High Cost



GoogLeNet - Going deeper with convolutions, 2014

Rethinking the Inception Architecture for Computer Vision, 2015

- Idea #2: Efficient Grid Size Reduction
 - Solution:



GoogLeNet - Going deeper with convolutions, 2014

Rethinking the Inception Architecture for Computer Vision, 2015

- Idea #3: Label Smoothing
 - Add noise to target (label), to regularize learning strategy:
 - $\varepsilon \approx 10^{-1}$, $K = \# \text{ of Classes}$

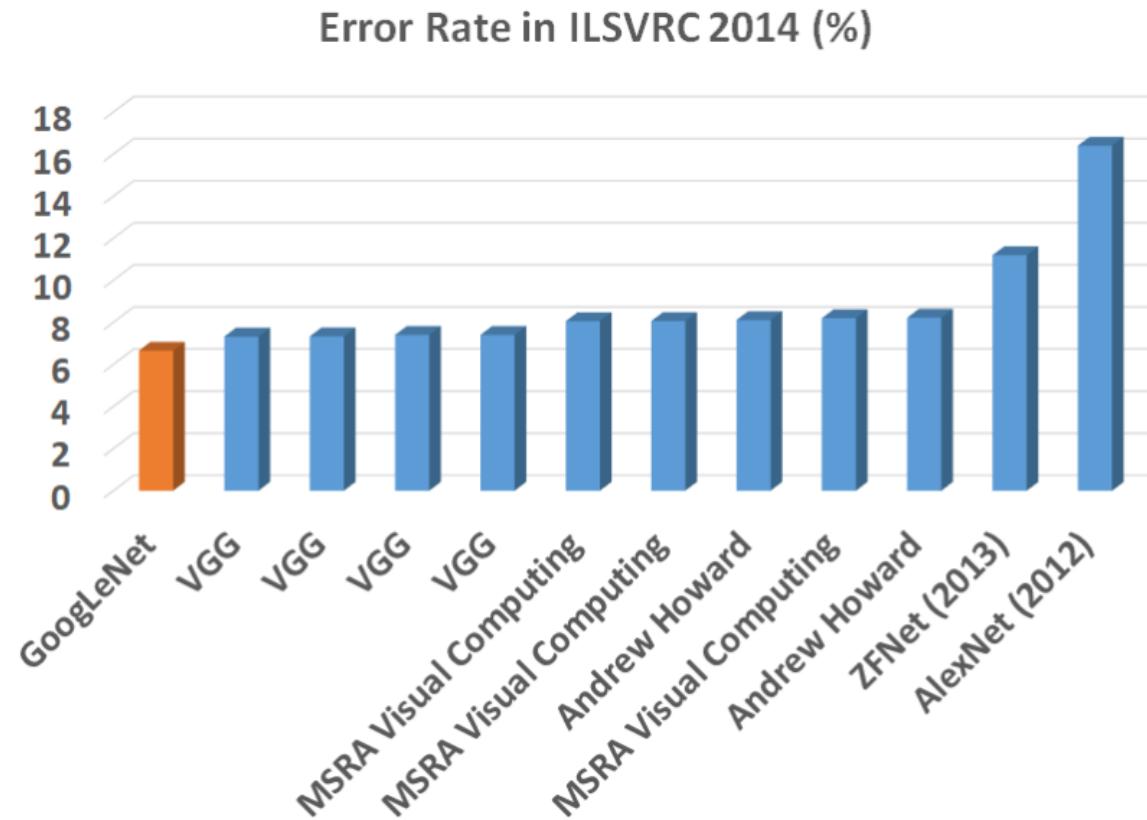
$$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} \frac{\varepsilon}{K} \\ \vdots \\ (1 - \varepsilon) \\ \vdots \\ \frac{\varepsilon}{K} \end{bmatrix}$$

|

Step#1: $\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0.02 \\ 0.94 \\ 0.02 \\ 0.02 \end{bmatrix}$, *Step#2:* $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 0.01 \\ 0.01 \\ 0.01 \\ 0.97 \end{bmatrix}$

GoogLeNet - Going deeper with convolutions, 2014

- Results:



GoogLeNet - Going deeper with convolutions, 2014

Rethinking the Inception Architecture for Computer Vision, 2015

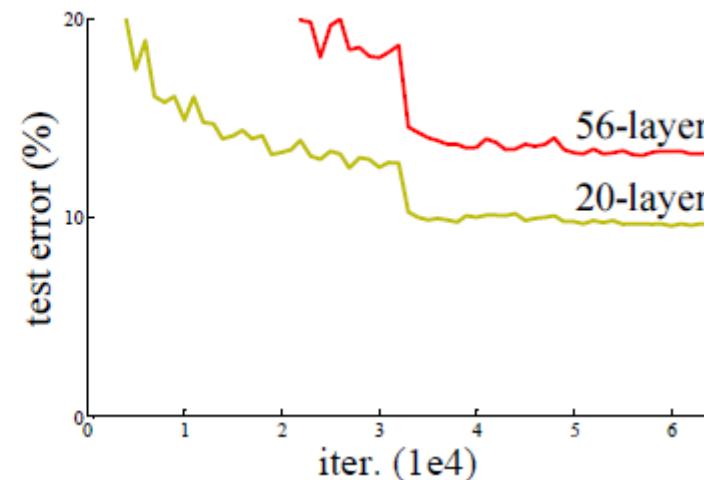
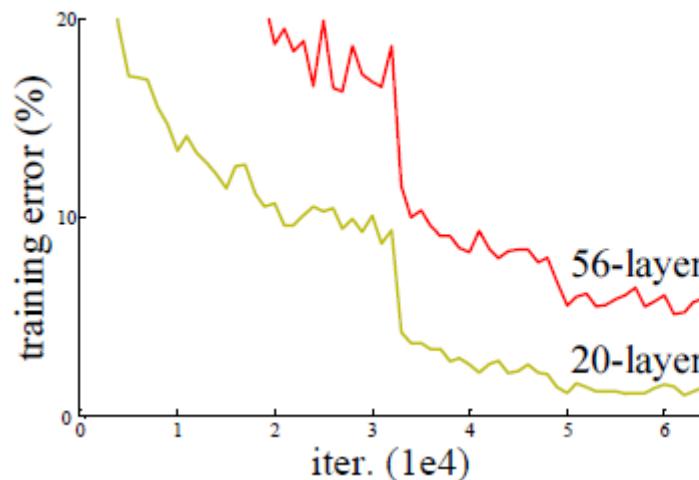
- Inception V3
- Motivation and Idea:
 - RMSProp Optimizer.
 - Factorized 7x7 convolutions.
 - BatchNorm in the Auxillary Classifiers.
 - Label Smoothing

GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

- Inception V4
 - Some block modifications and ResNet (Microsoft, next slides) module

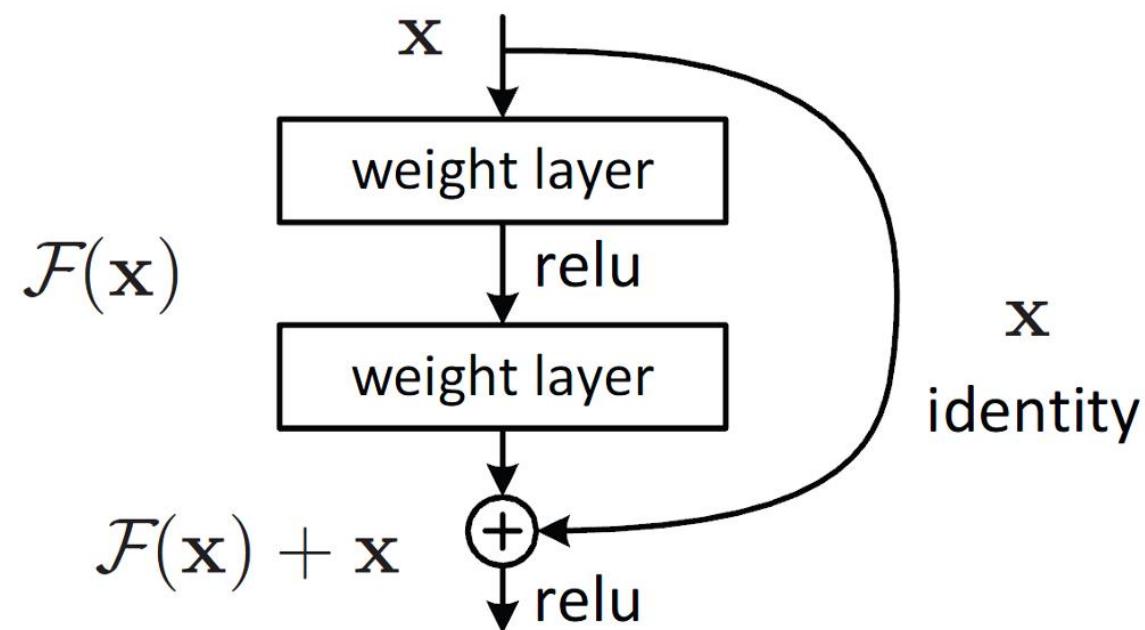
ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Problems with Deeper Networks:
 - Performance degradation, Not due to overfitting
 - Gradient vanishing/exploding
 - Increasing Depth → Accuracy Saturation then degrades rapidly!
 - For a PLAIN network (on CIFAR-10)



ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Idea: New Block (Shortcut Connection/Skip Connections)
 - Deep Residual Learning!



ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Why? A Identity map is hard to capture by highly nonlinear map.
- In Residual Learning:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \mathbf{W}) + \mathbf{x}$$

- $H(x) = F(x) + x$, the $F(x)$ may learn to be zero!
 - No Extra Parameter! ☺
 - No Computational Complexity! ☺
 - Problem of dimension F and X ☹
 - Solution:

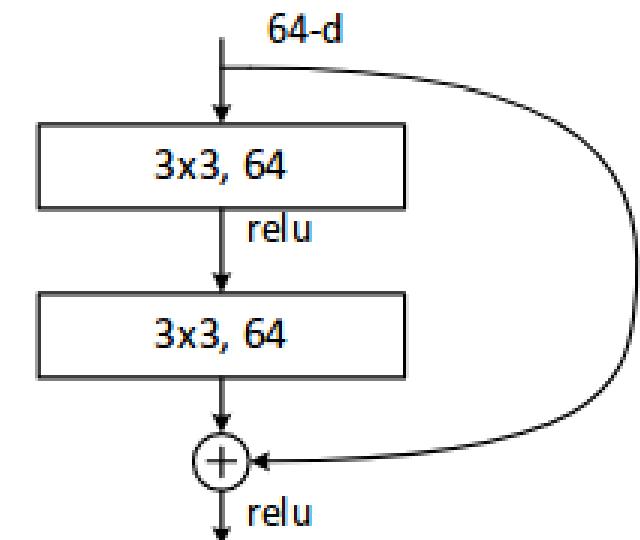
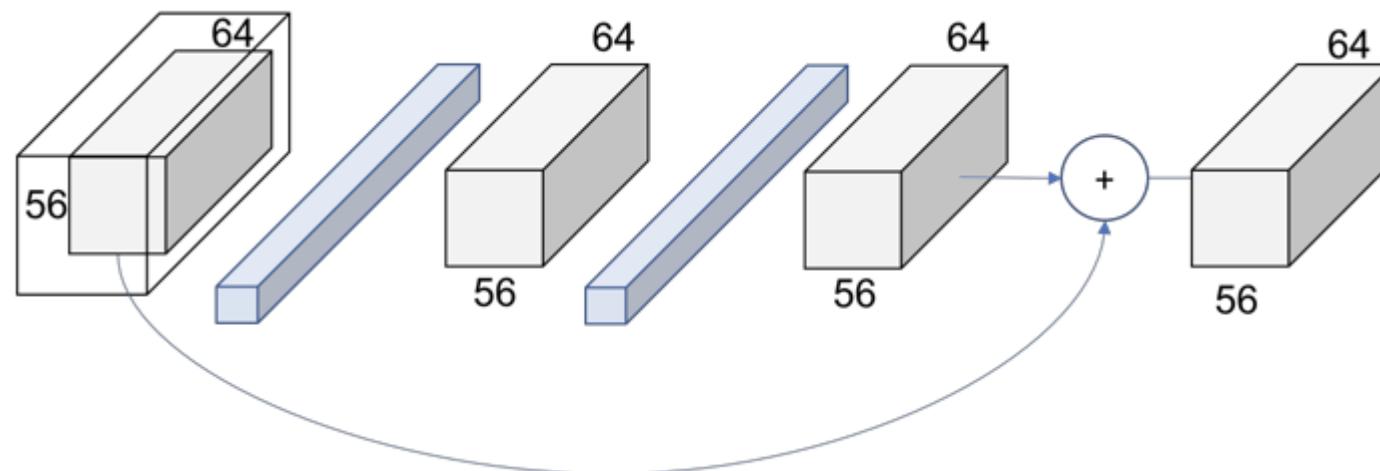
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \mathbf{W}) + \mathbf{W}_s \mathbf{x}$$

ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- ResNet Benefits:
 - Vanishing and exploding gradients
 - May increase # of layer

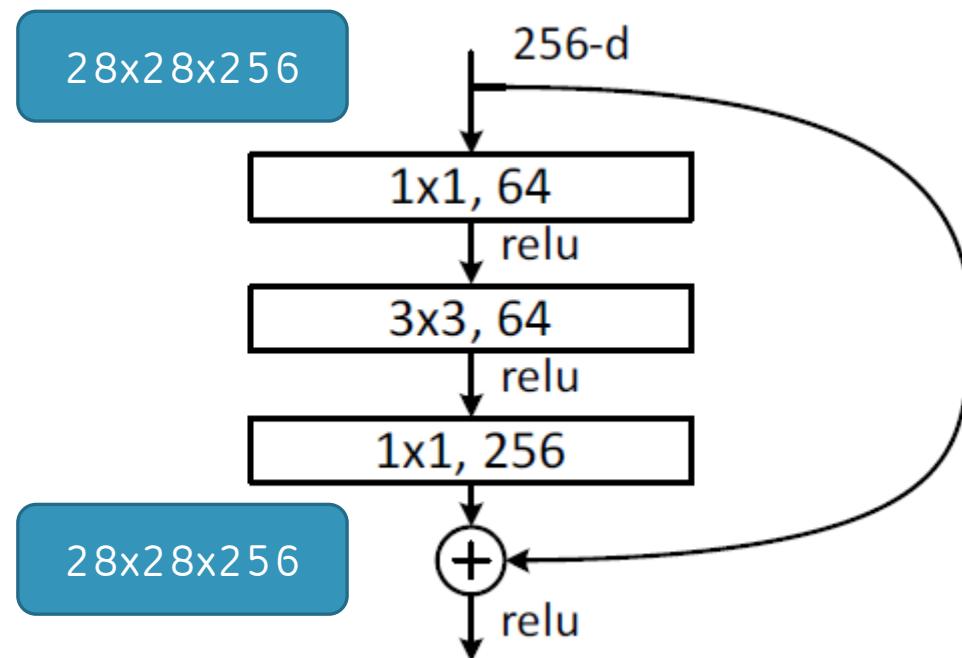
ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Residual Layer (Type I):
- Input: $(56 \times 56) \times 64$
- Kernel: (3×3) same $\times 64$



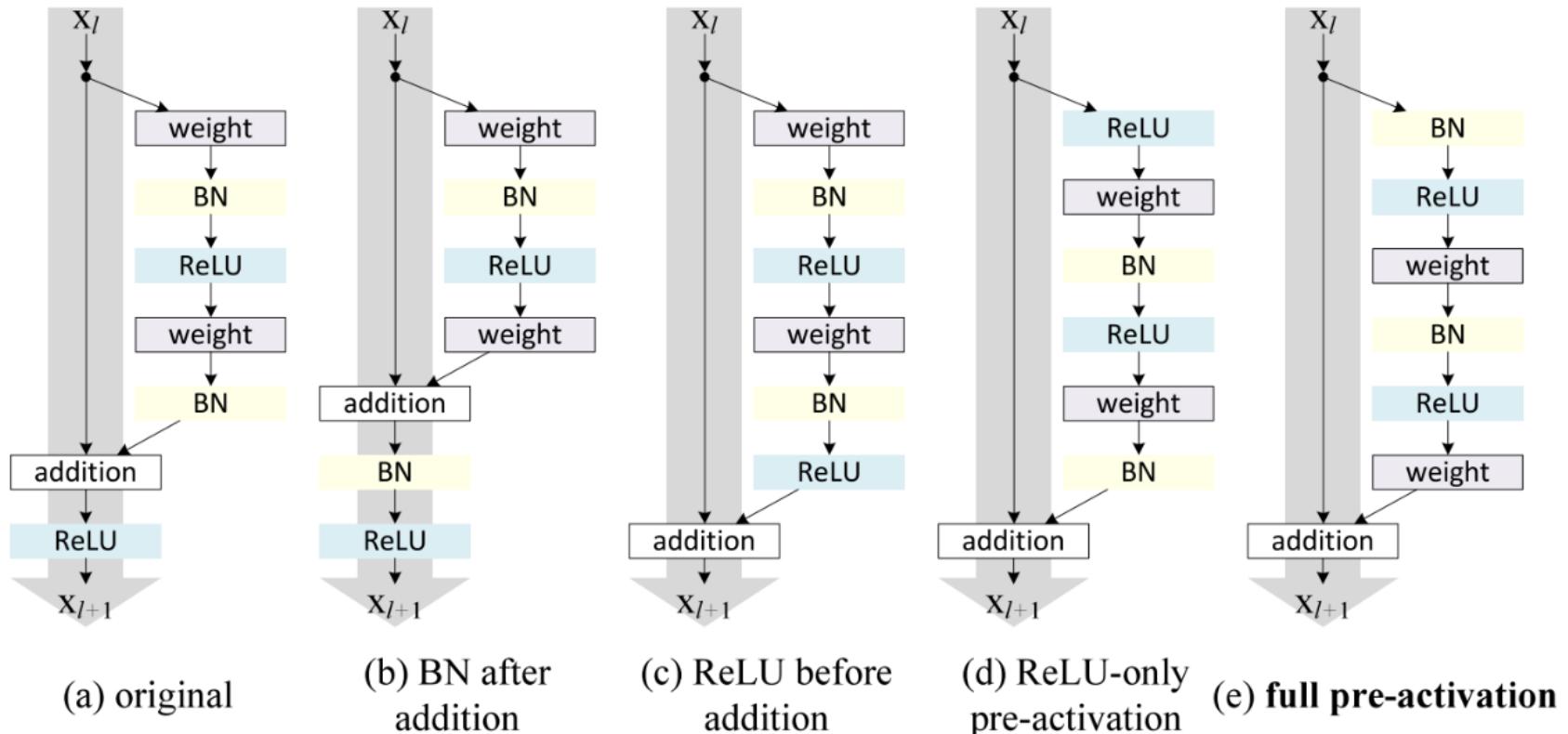
ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Residual Layer (Type II):
- Another idea for Deeper Net (Bottleneck Design):



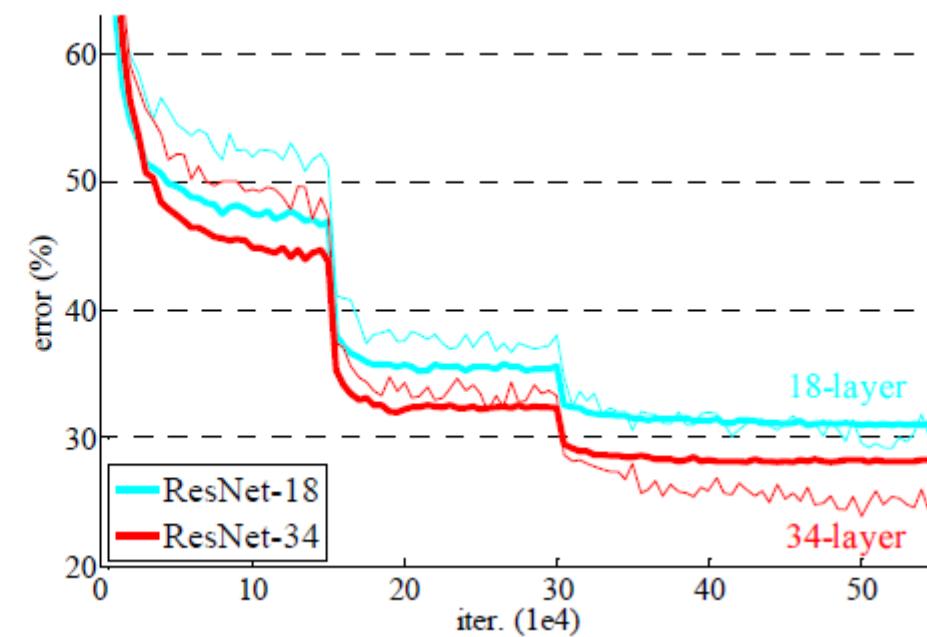
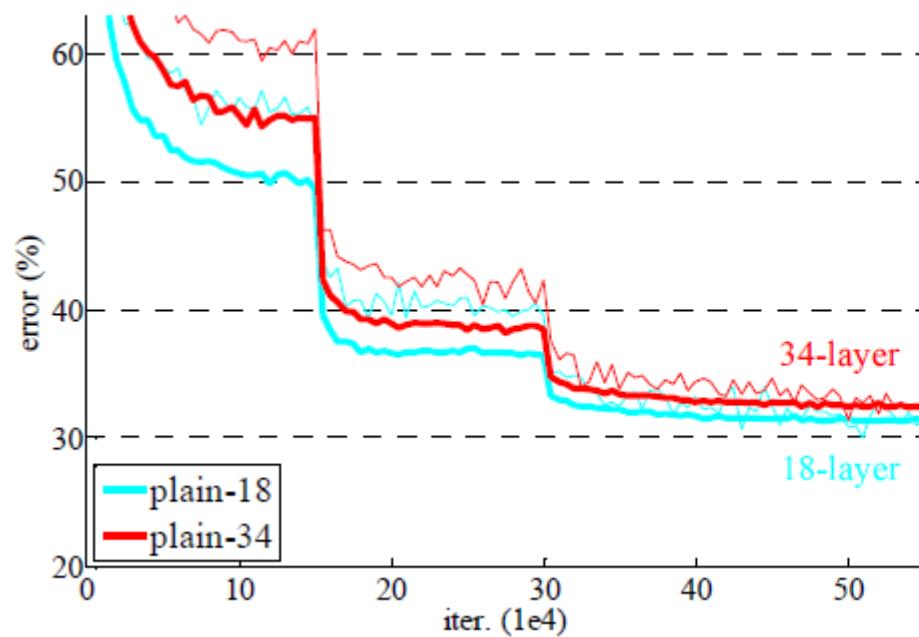
ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Variation:



ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Results (ImageNet, Not too deep, 18/34):



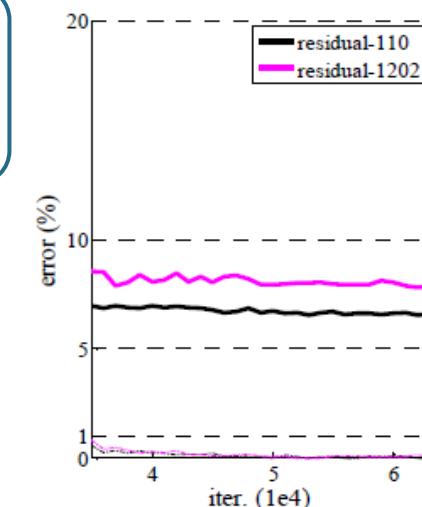
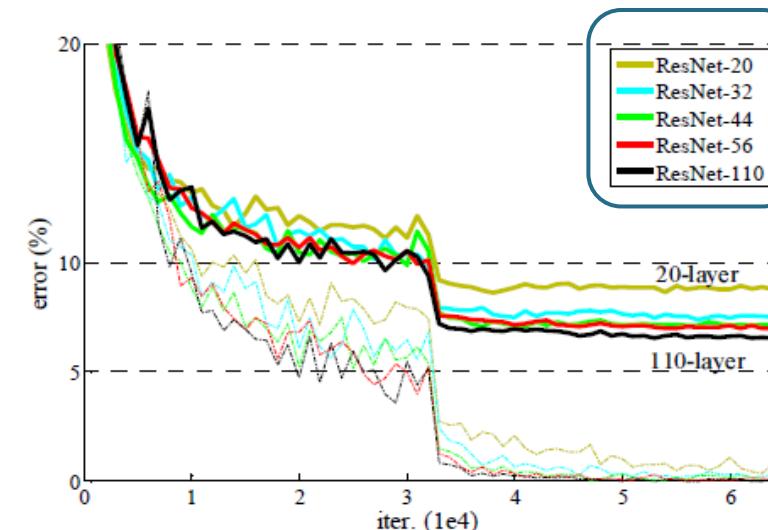
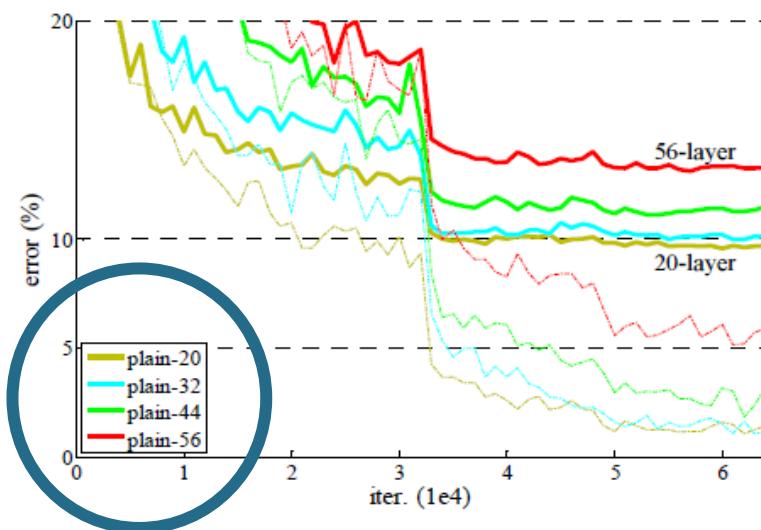
ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Results (ImageNet, very deep, 152):

| method | top-5 err. (test) |
|----------------------------|----------------------------|
| VGG [41] (ILSVRC'14) | 7.32 |
| GoogLeNet [44] (ILSVRC'14) | 6.66 |
| VGG [41] (v5) | 6.8 |
| PReLU-net [13] | 4.94 |
| BN-inception [16] | 4.82 |
| ResNet (ILSVRC'15) | 3.57 |

ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Results (CIFAR, too deep, 20/110/1202):

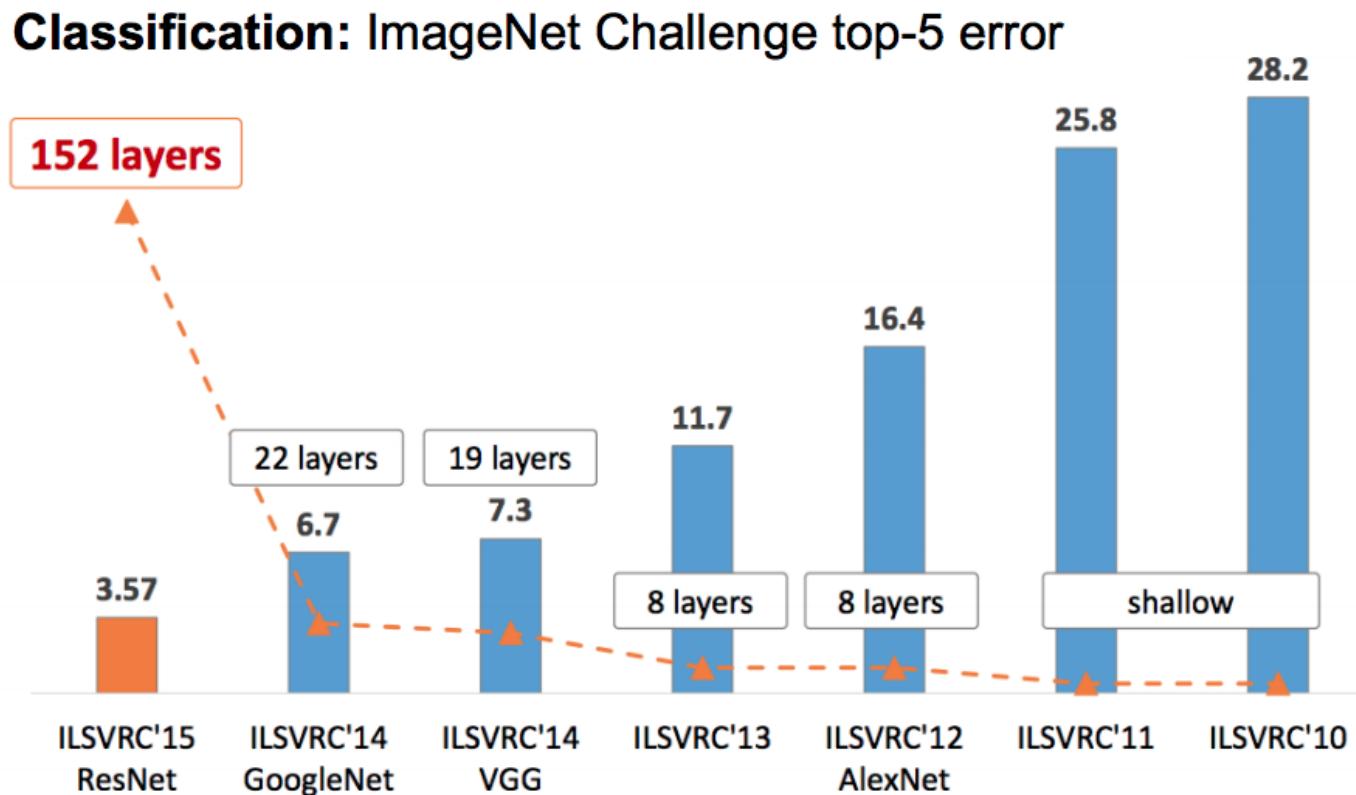


ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Specification:
 - Batch Normalization after each ConvLayer
 - SGD + momentum (0.9)
 - Minibatch Size: 256
 - No Dropout

ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Depth Effect!



Results - <https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/googlenet.html>

- ImageNet 2015

| 2015 ResNet (ILSVRC'15) 3.57 | | | | |
|------------------------------|--------------------------|-----------------|----------------|--|
| Year | Codename | Error (percent) | 99.9% Conf Int | |
| 2014 | GoogLeNet | 6.66 | 6.40 - 6.92 | |
| 2014 | VGG | 7.32 | 7.05 - 7.60 | |
| 2014 | MSRA | 8.06 | 7.78 - 8.34 | |
| 2014 | AHoward | 8.11 | 7.83 - 8.39 | |
| 2014 | DeeperVision | 9.51 | 9.21 - 9.82 | |
| 2013 | Clarifai [†] | 11.20 | 10.87 - 11.53 | |
| 2014 | CASIAWS [†] | 11.36 | 11.03 - 11.69 | |
| 2014 | Trimp [†] | 11.46 | 11.13 - 11.80 | |
| 2014 | Adobe [†] | 11.58 | 11.25 - 11.91 | |
| 2013 | Clarifai | 11.74 | 11.41 - 12.08 | |
| 2013 | NUS | 12.95 | 12.60 - 13.30 | |
| 2013 | ZF | 13.51 | 13.14 - 13.87 | |
| 2013 | AHoward | 13.55 | 13.20 - 13.91 | |
| 2013 | OverFeat | 14.18 | 13.83 - 14.54 | |
| 2014 | Orange [†] | 14.80 | 14.43 - 15.17 | |
| 2012 | SuperVision [†] | 15.32 | 14.94 - 15.69 | |
| 2012 | SuperVision | 16.42 | 16.04 - 16.80 | U. of Toronto, SuperVision, a 7 layers network |
| 2012 | ISI | 26.17 | 25.71 - 26.65 | |
| 2012 | VGG | 26.98 | 26.53 - 27.43 | |
| 2012 | XRCE | 27.06 | 26.60 - 27.52 | |
| 2012 | UvA | 29.58 | 29.09 - 30.04 | |

human error is around 5.1% on a subset

Microsoft ResNet, a 152 layers network

GoogLeNet, 22 layers network

U. of Toronto, SuperVision, a 7 layers network

GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

- Inception V4
 - Introduce 3 types of inception module, 2 types of reduction module
 - Build Inception-ResNet, Inception-ResNet reduction Block

GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

- Inception V4, Inception Modules A, B, C

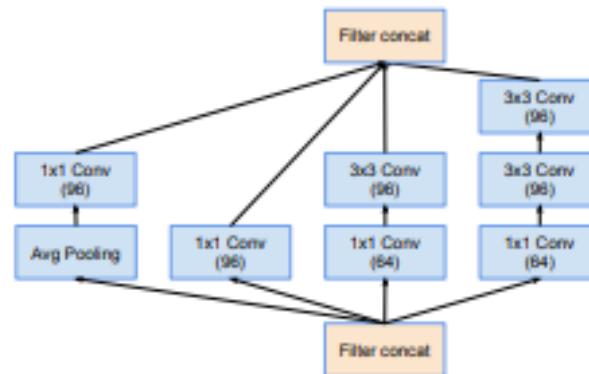


Figure 4. The schema for 35×35 grid modules of the pure Inception-v4 network. This is the Inception-A block of Figure 9.

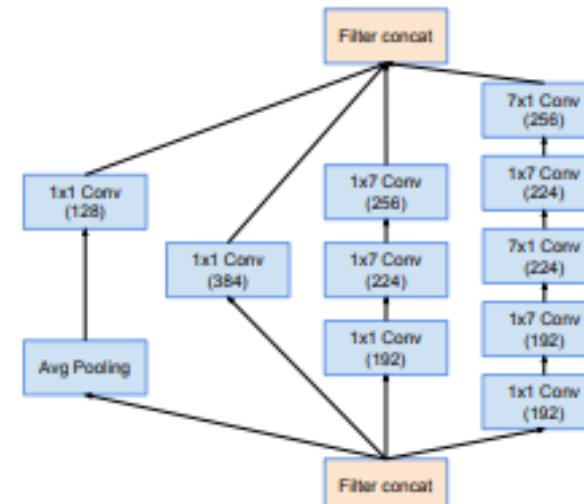


Figure 5. The schema for 17×17 grid modules of the pure Inception-v4 network. This is the Inception-B block of Figure 9.

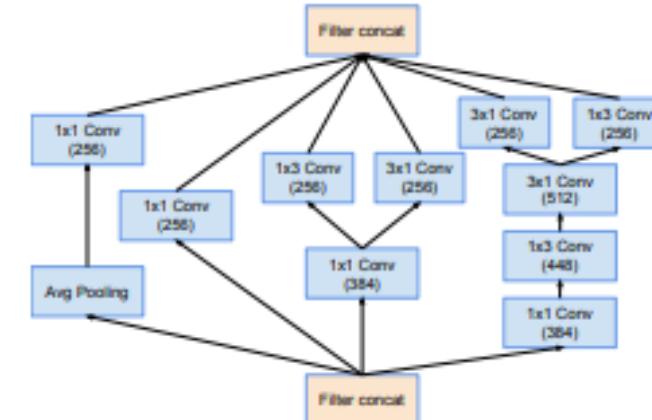


Figure 6. The schema for 8×8 grid modules of the pure Inception-v4 network. This is the Inception-C block of Figure 9.

GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

- Inception V4, Inception-Reduction Modules A, B

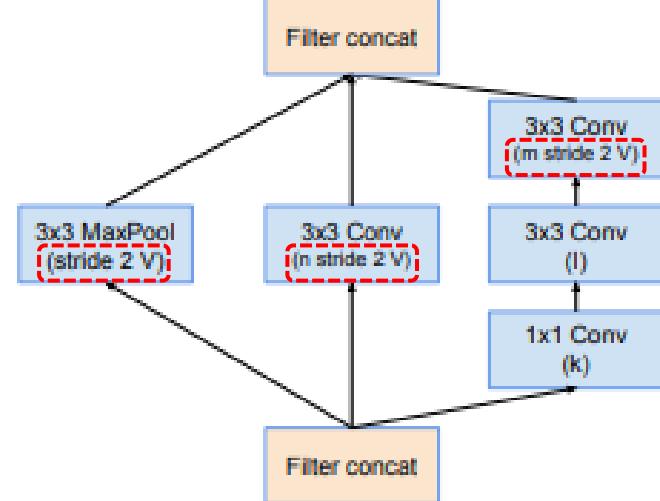


Figure 7. The schema for 35×35 to 17×17 reduction module. Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The k, l, m, n numbers represent filter bank sizes which can be looked up in Table 1.

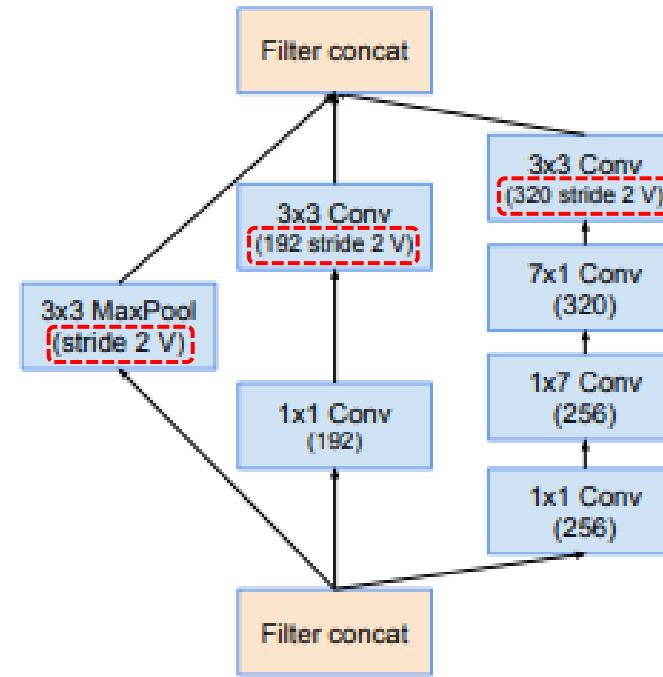


Figure 8. The schema for 17×17 to 8×8 grid-reduction module. This is the reduction module used by the pure Inception-v4 network in Figure 9.

GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

- Inception V4, Inception-ResNet Modules A, B, C

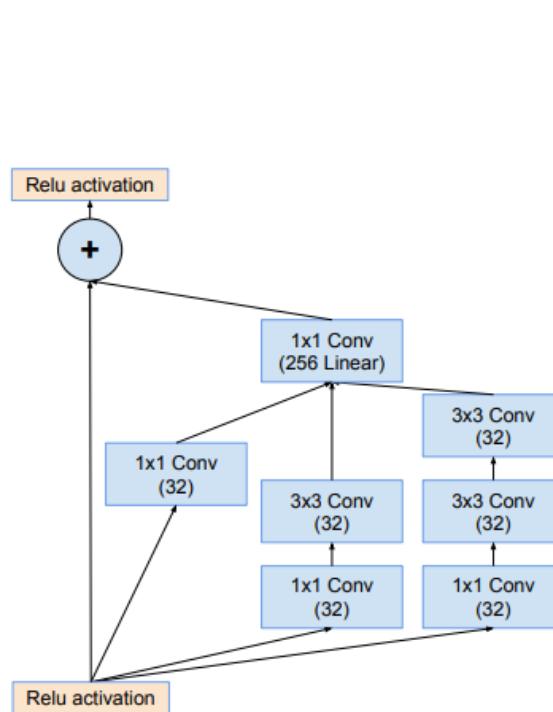


Figure 10. The schema for 35×35 grid (Inception-ResNet-A) module of Inception-ResNet-v1 network.

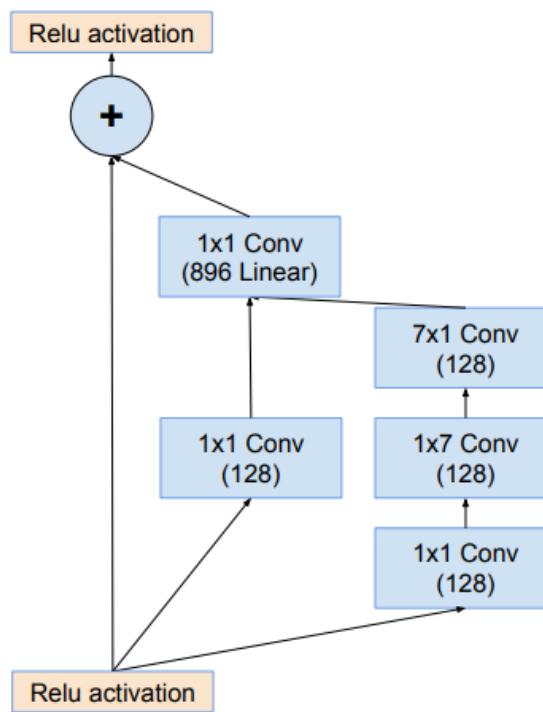


Figure 11. The schema for 17×17 grid (Inception-ResNet-B) module of Inception-ResNet-v1 network.

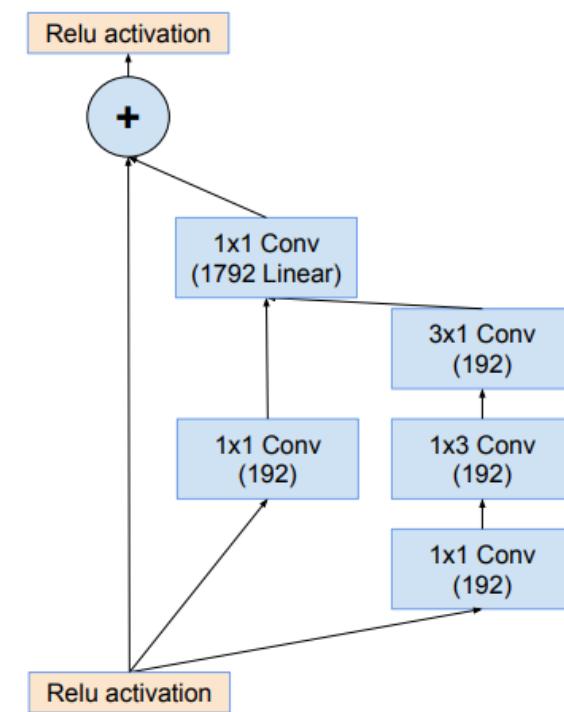
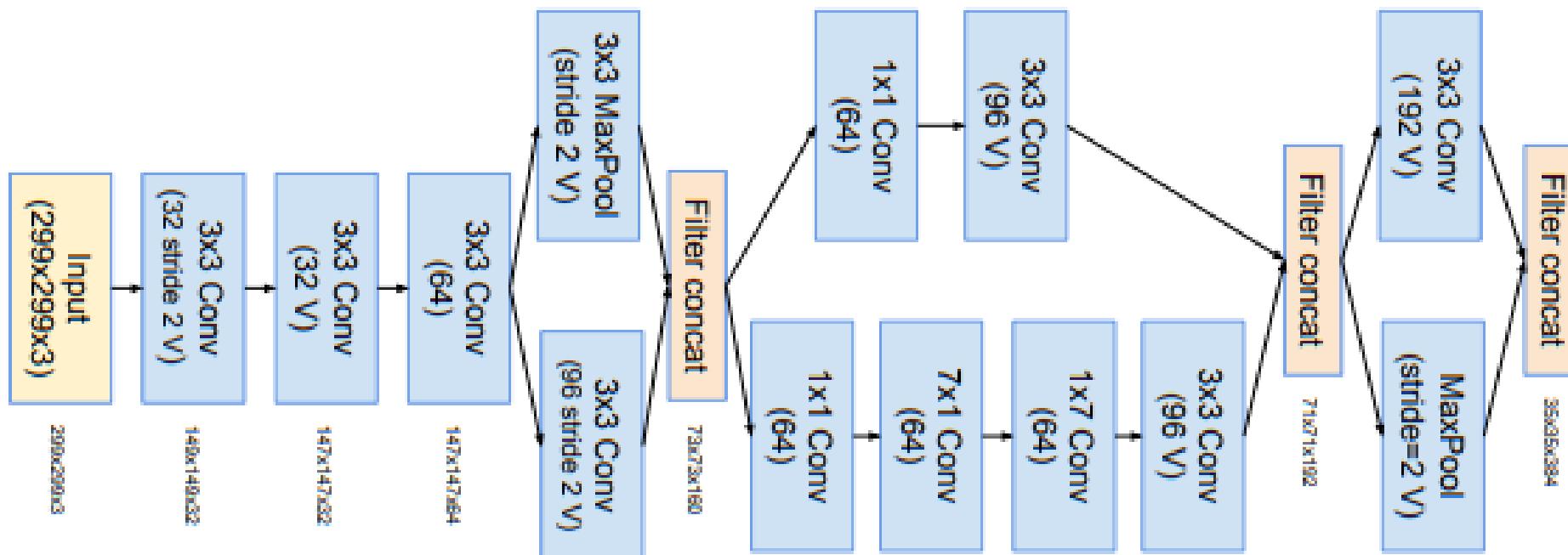


Figure 13. The schema for 8×8 grid (Inception-ResNet-C) module of Inception-ResNet-v1 network.

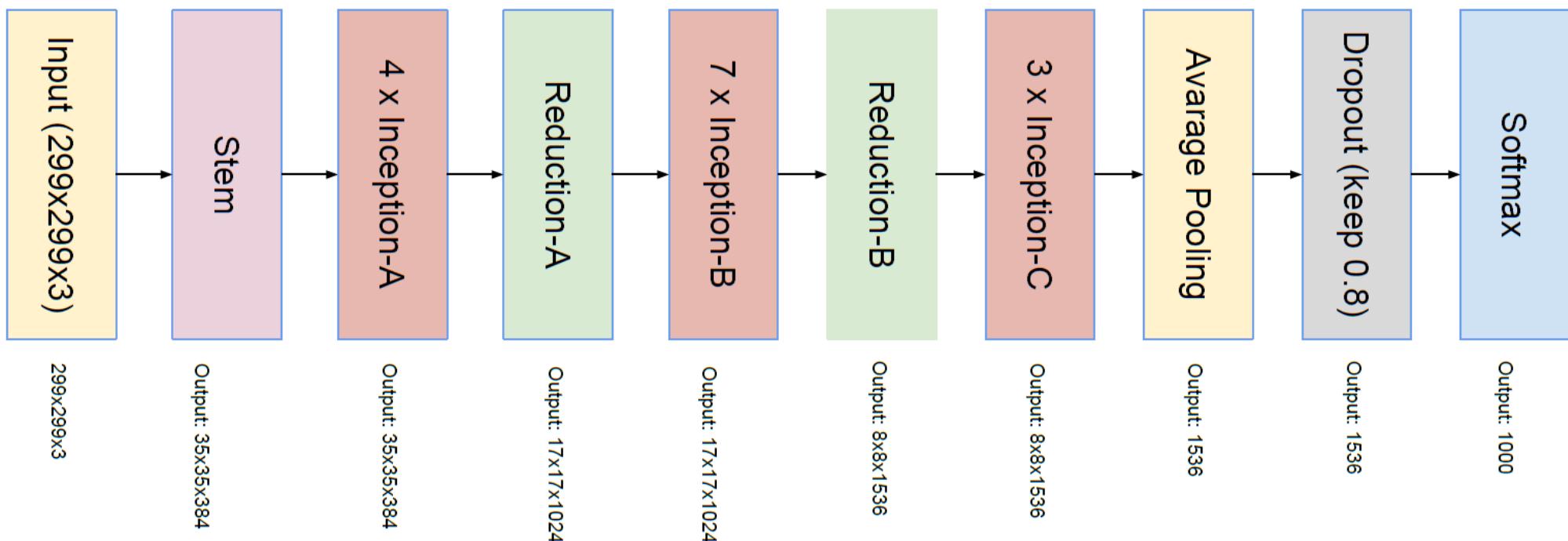
GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

- Inception V4, **stem** module



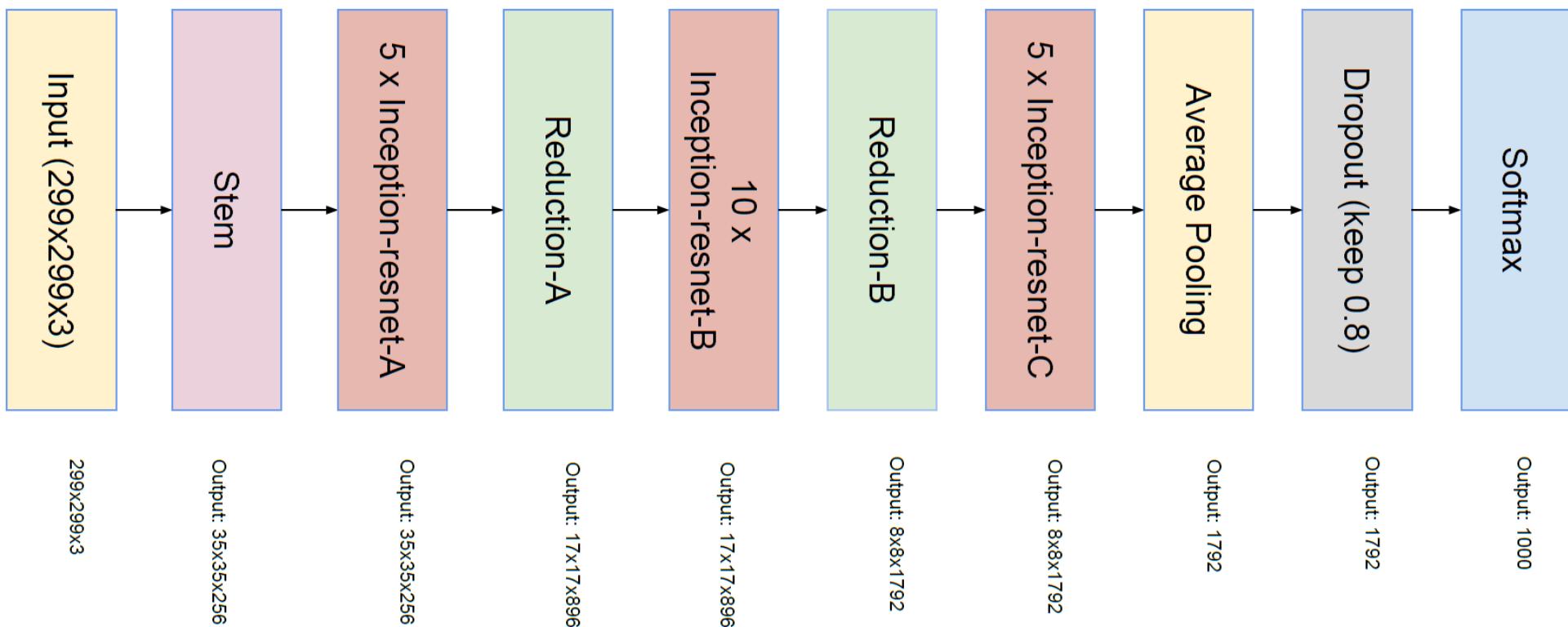
GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

- Inception V4, Overall Scheme



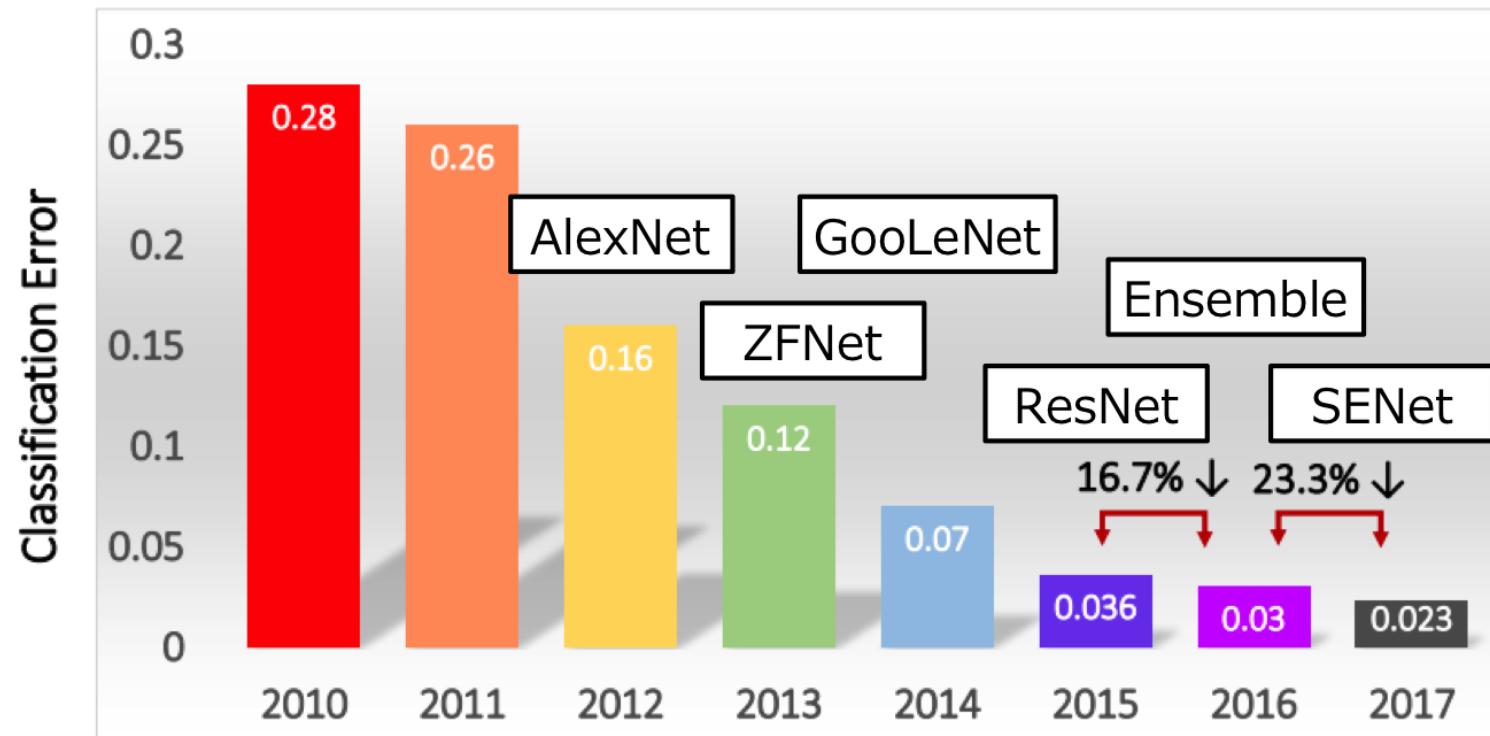
GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

- Inception-ResNet V4, Overall Scheme



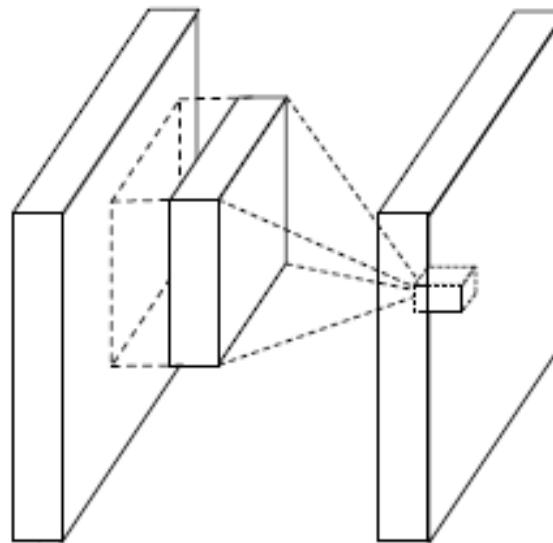
GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

- 2010-2017



NiN - Network In Network, 2013

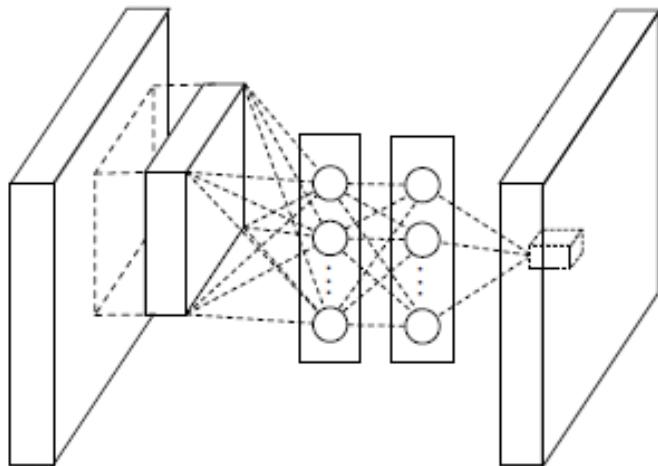
- Network in Network (NiN)
 - Conventional Convolution Layer:



$$f_{i,j,k} = \max(w_k^T x_{i,j}, 0).$$

NiN - Network In Network, 2013

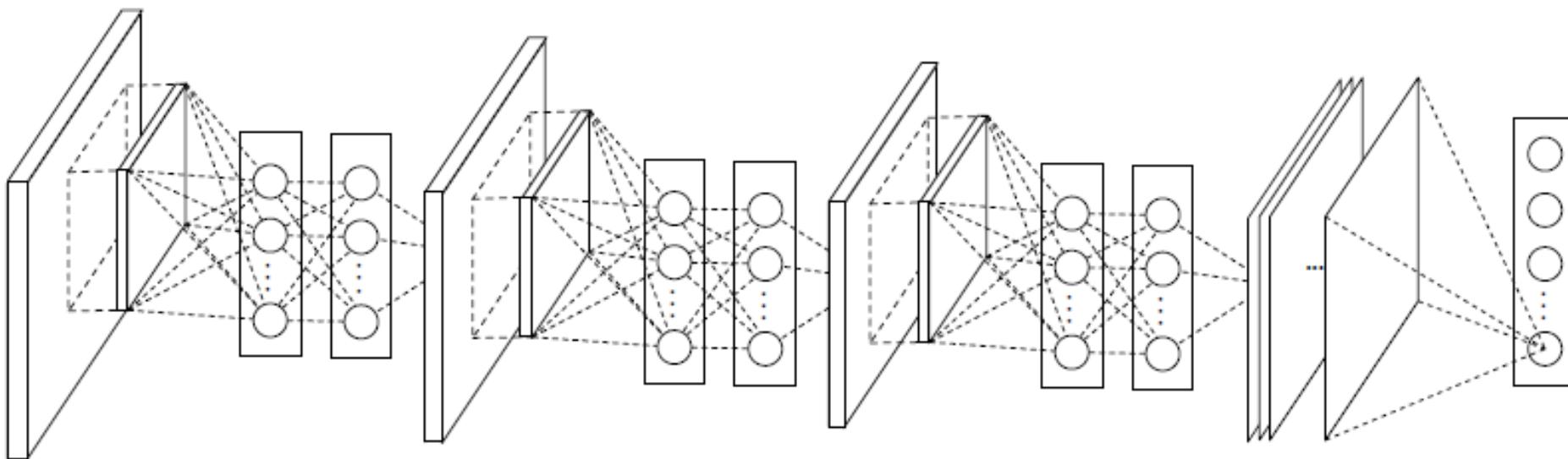
- Network in Network (NiN)
 - MLP Convolution Layer:



$$\begin{aligned} f_{i,j,k_1}^1 &= \max({w_{k_1}^1}^T x_{i,j} + b_{k_1}, 0). \\ &\vdots \\ f_{i,j,k_n}^n &= \max({w_{k_n}^n}^T f_{i,j}^{n-1} + b_{k_n}, 0). \end{aligned}$$

NiN - Network In Network, 2013

- Network in Network (NiN)
 - Global Average Pooling Before FC



DenseNet - Densely Connected Convolutional Networks, CVPR 2017

- Idea:
- Conventional CNN:

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}).$$

- ResNet:

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}.$$

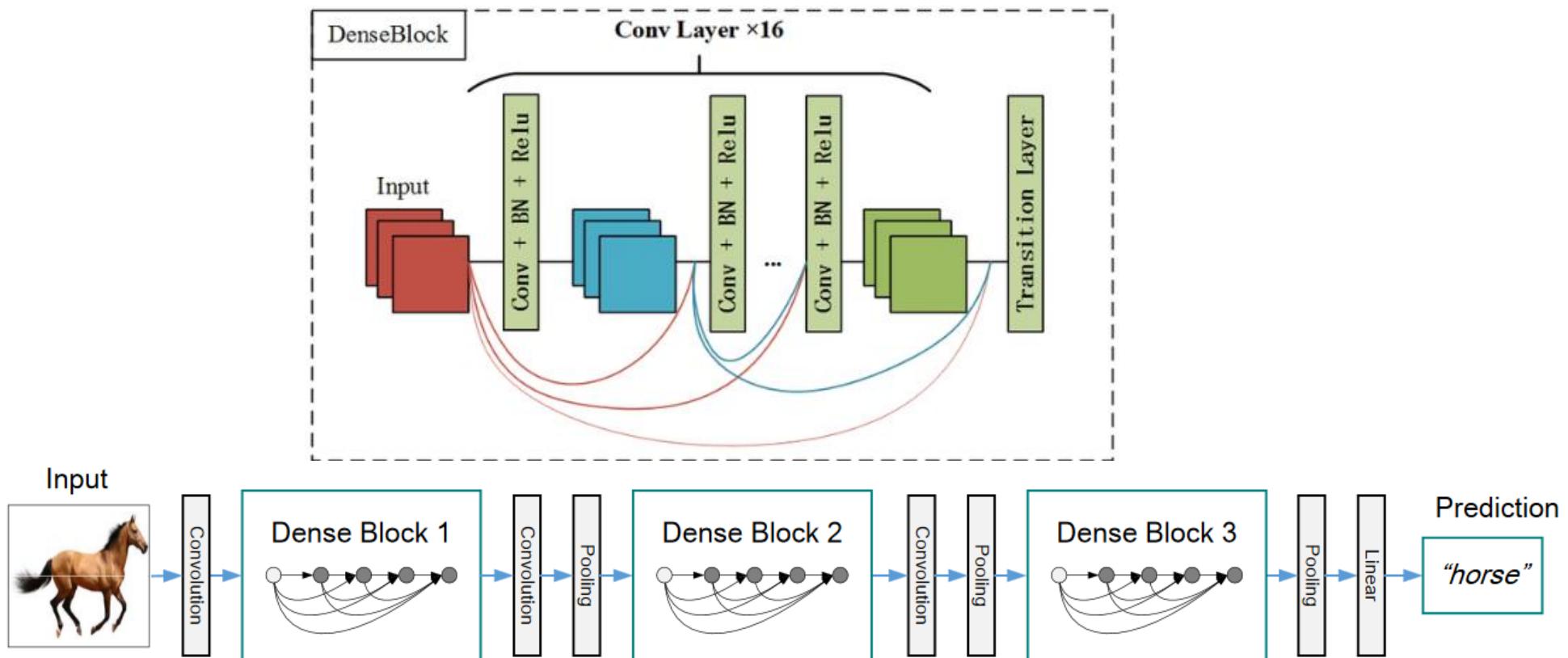
- DenseNet:

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{\ell-1}]),$$

- $[\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{\ell-1}]$: Concatenation of the feature-maps in previous layer

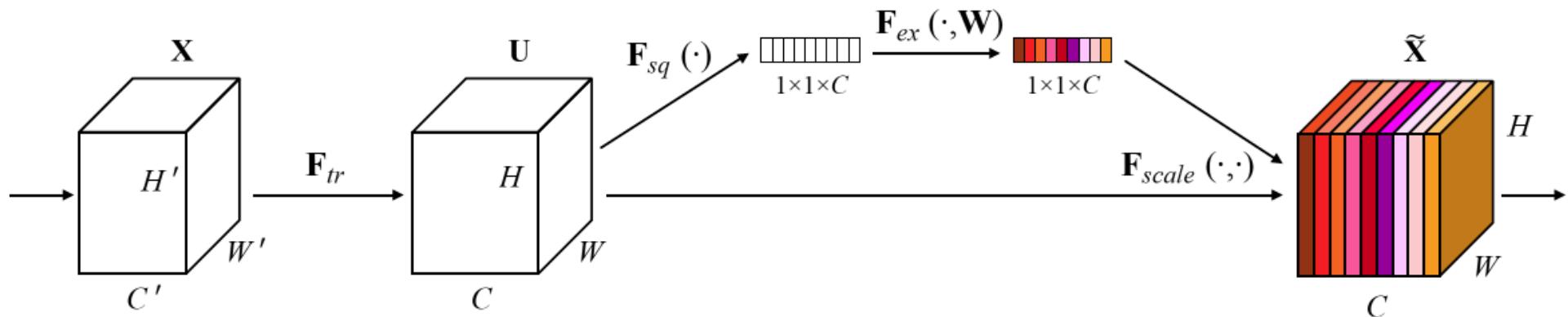
DenseNet - Densely Connected Convolutional Networks, CVPR 2017

- $H(\cdot)$ functionality: Batch Normalization, ReLU, (3×3) Conv



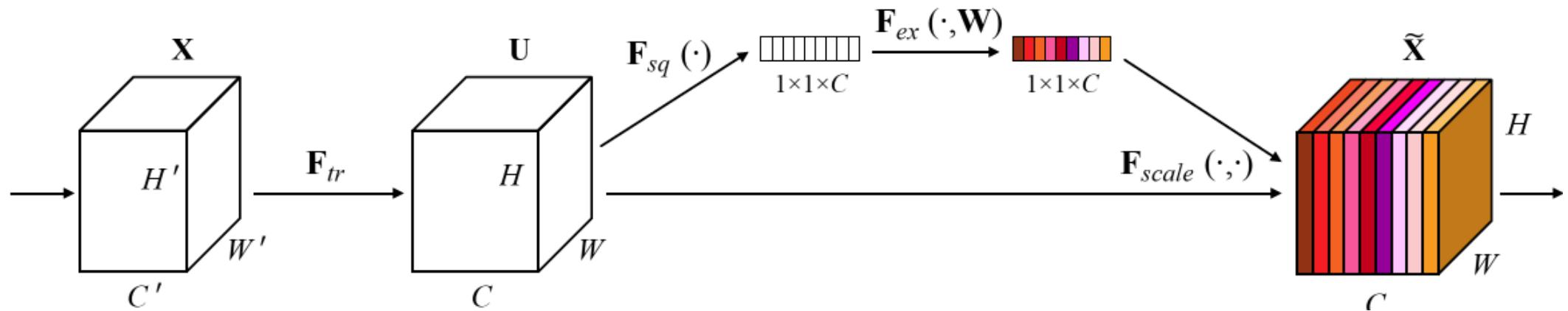
SENet - Squeeze-and-Excitation Networks, CVPR 2017

- Main Idea: Weight Channels by learning
- Squeeze-and-Excitation (SE) Block:



SENet - Squeeze-and-Excitation Networks, CVPR 2017

- Squeeze-and-Excitation (SE) Block:



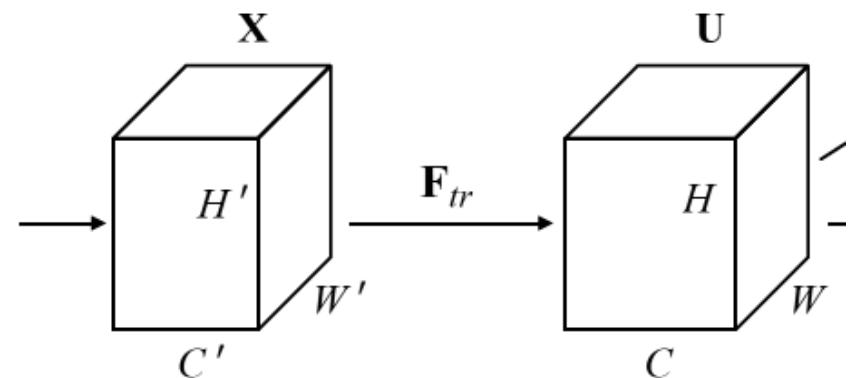
- Transform: $U = F_{tr}(X)$: $\mathbb{R}^{H' \times W' \times C'} \rightarrow \mathbb{R}^{H \times W \times C}$
- Squeezing: $z = F_{sq}(U)$: $\mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R}^{1 \times 1 \times C}$
- Excitation: $s = F_{ex}(U)$: $\mathbb{R}^C \rightarrow \mathbb{R}^C$

SENet - Squeeze-and-Excitation Networks, CVPR 2017

- Transform: 3D Convolution
- Transform: $U = F_{tr}(X)$: $\mathbb{R}^{H' \times W' \times C'} \rightarrow \mathbb{R}^{H \times W \times C}$

$$\mathbf{u}_c = \mathbf{v}_c * \mathbf{X} = \sum_{s=1}^{C'} \mathbf{v}_c^s * \mathbf{x}^s$$

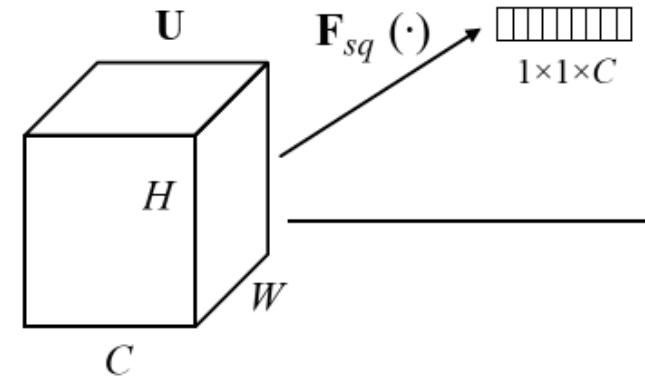
- Implement by Residual or Inception Module



SENet - Squeeze-and-Excitation Networks, CVPR 2017

- Squeeze: Global Information Embedding

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W u_c(i, j)$$

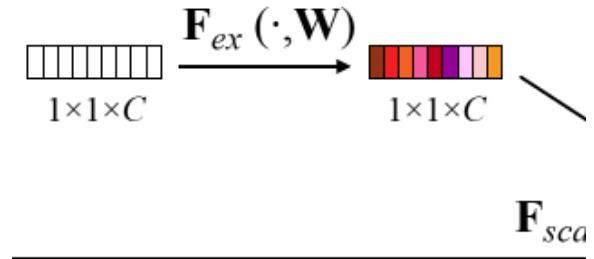


- Global pooling, A Channel Descriptor!

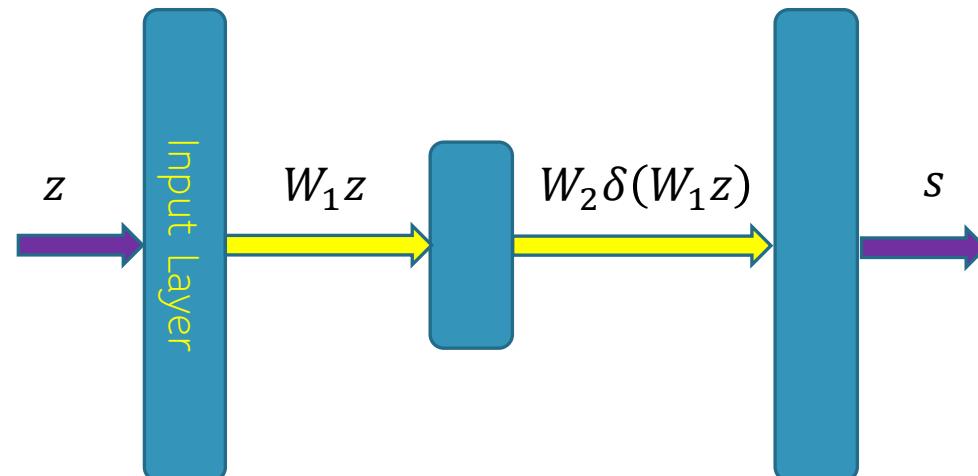
SENet - Squeeze-and-Excitation Networks, CVPR 2017

- Excitation (SE) Block:

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})),$$

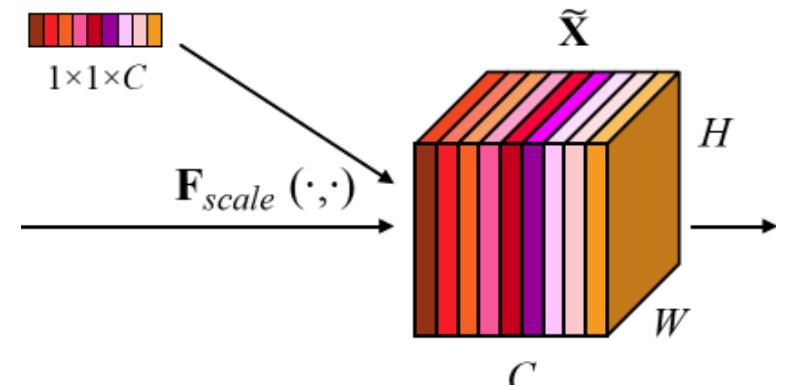


- $W_1 \in \mathbb{R}^{\frac{C}{r} \times C}$: FC Layer
- δ : ReLU
- $W_2 \in \mathbb{R}^{C \times \frac{C}{r}}$: FC Layer
- σ : sigmoid



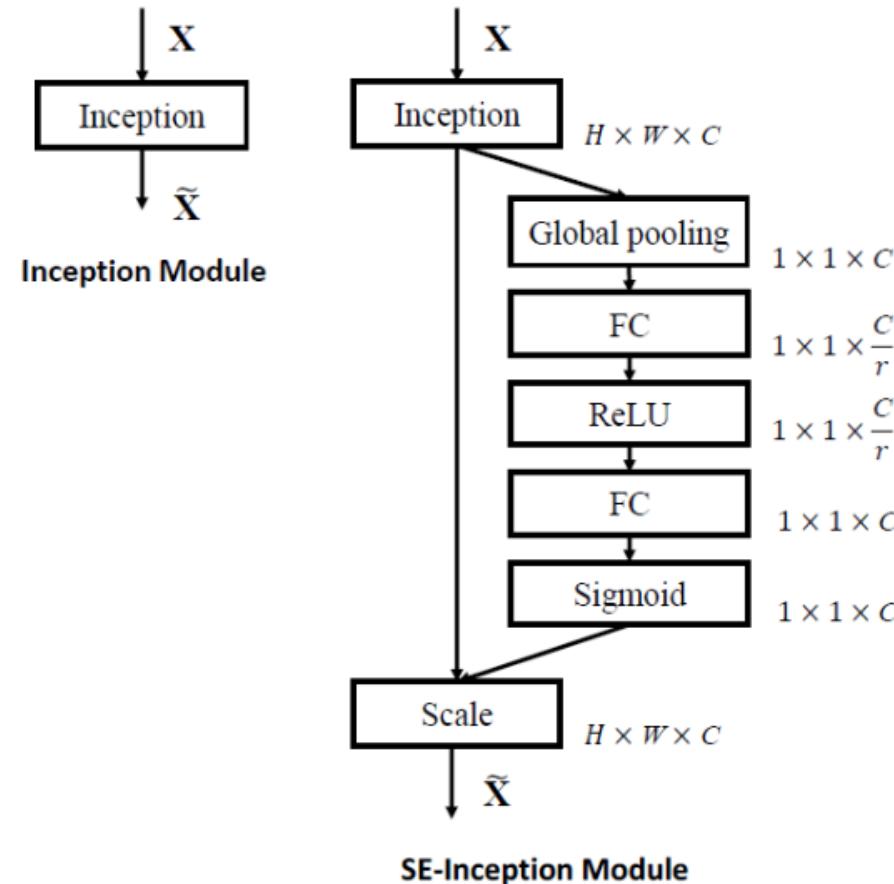
SENet - Squeeze-and-Excitation Networks, CVPR 2017

- Scaling:
- Channel-wise multiplication
- S_i (scalar) multiply by $X(:, :, i) \rightarrow \tilde{X}(:, :, i)$



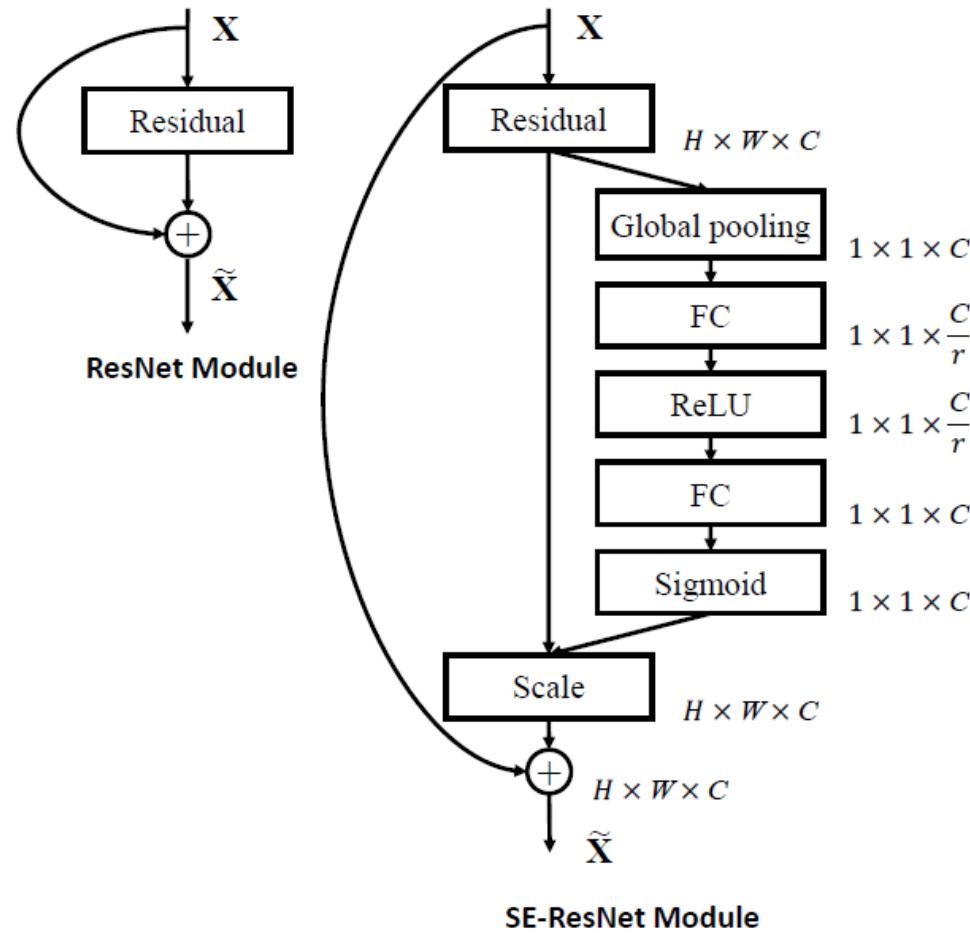
SENet - Squeeze-and-Excitation Networks, CVPR 2017

- SE-Inception



SENet - Squeeze-and-Excitation Networks, CVPR 2017

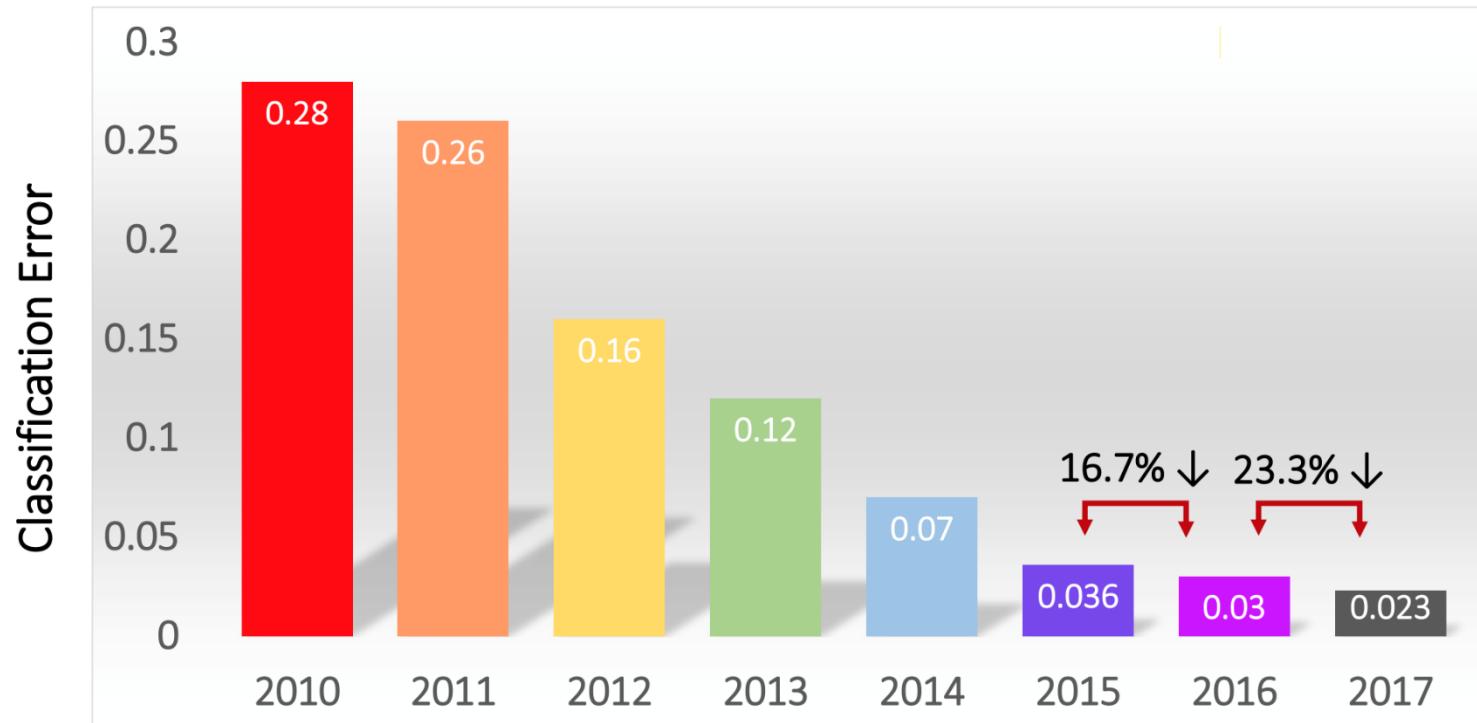
- SE-Residual



SENet - Squeeze-and-Excitation Networks, CVPR 2017

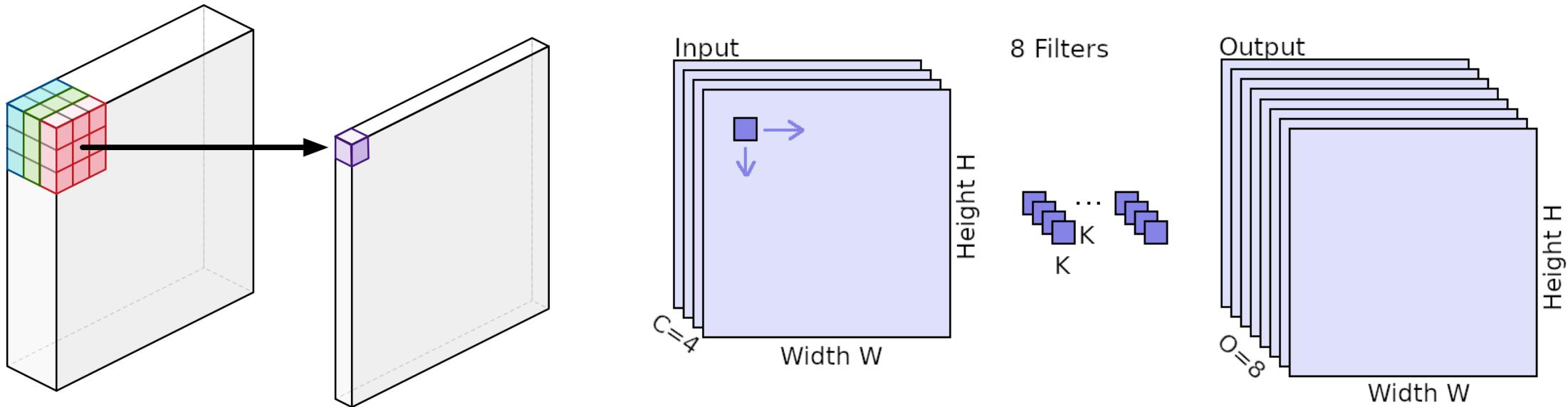
- Results

Classification Results (CLS)



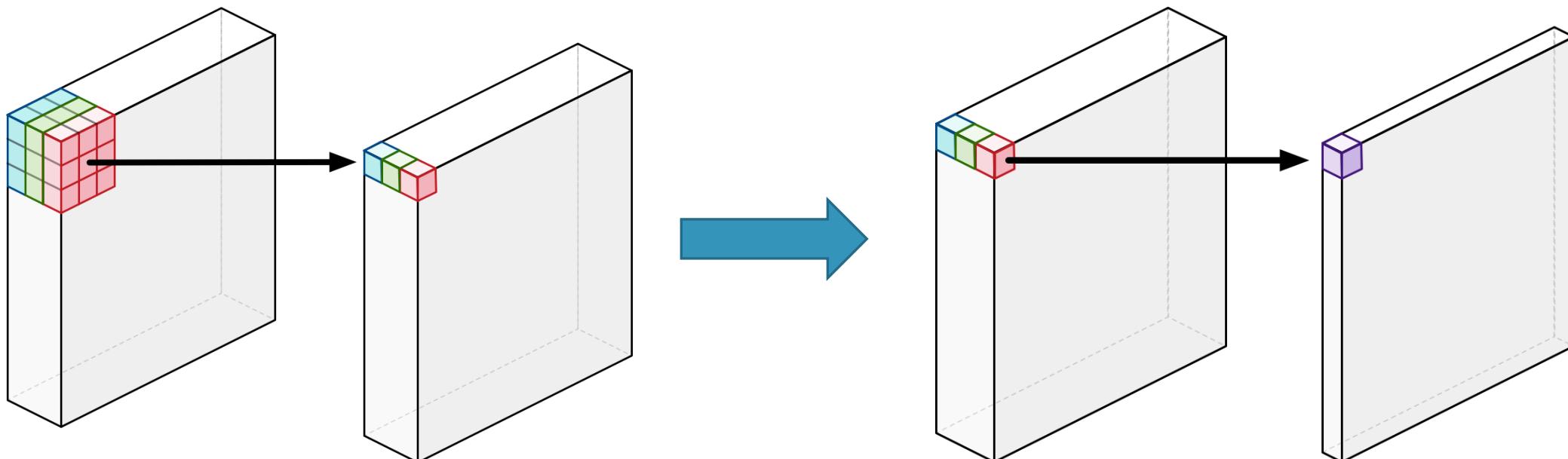
Xception

- Core idea: New Depthwise Separable Convolution!
- Conventional 3D convolution:



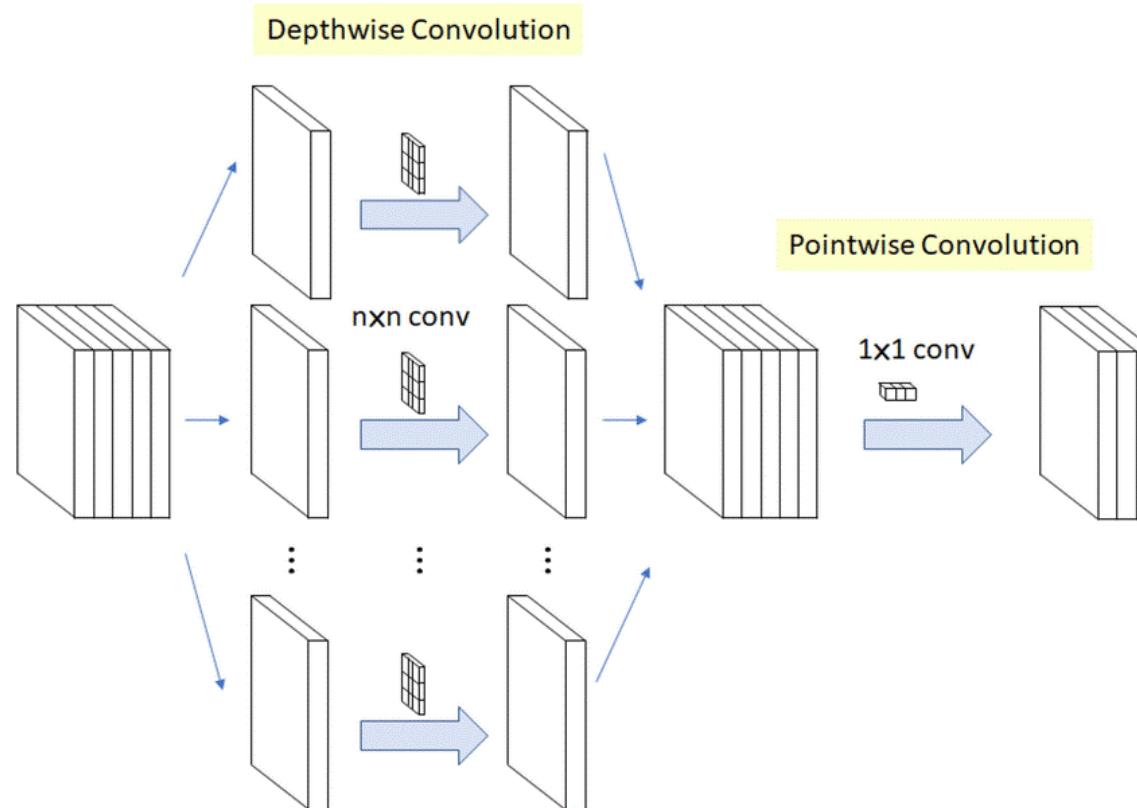
Xception

- Original Depthwise Separable Convolution!



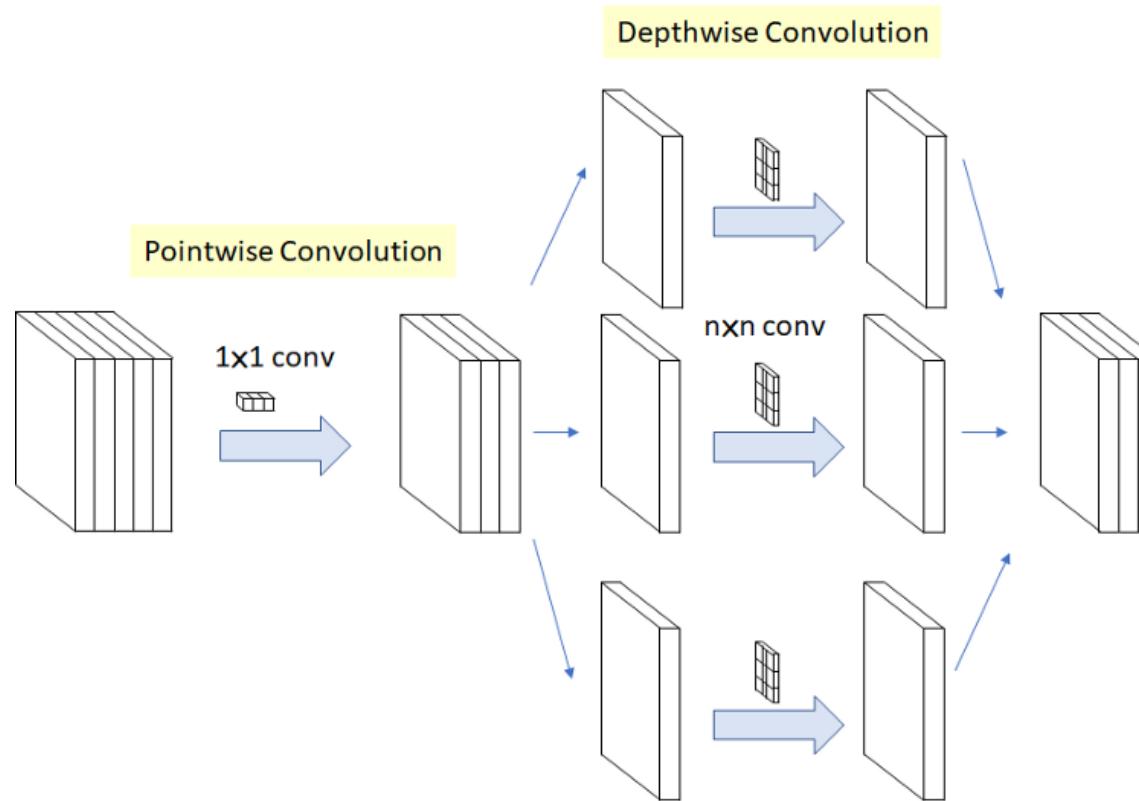
Xception

- Original Depthwise Separable Convolution!



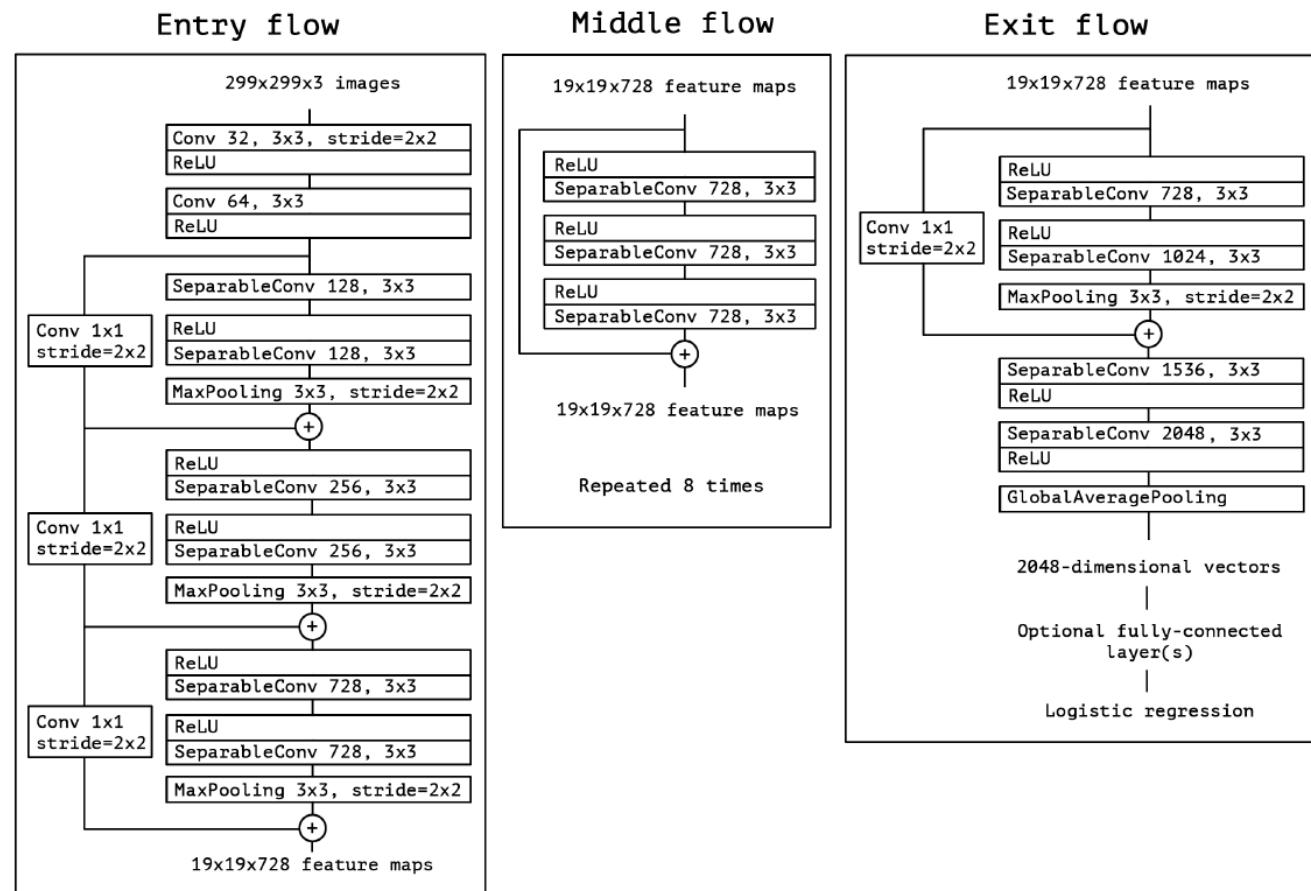
Xception

- Xception Model Depthwise Separable Convolution!



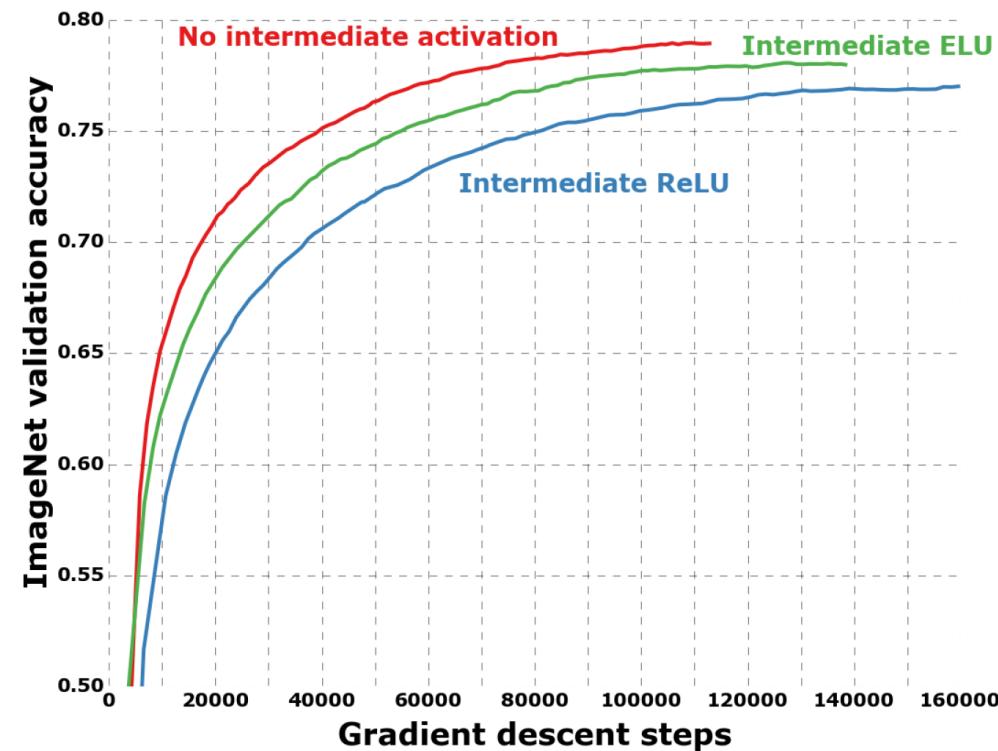
Xception: Deep Learning with Depthwise Separable Convolutions - 2017

- Three Level of models:



Xception: Deep Learning with Depthwise Separable Convolutions - 2017

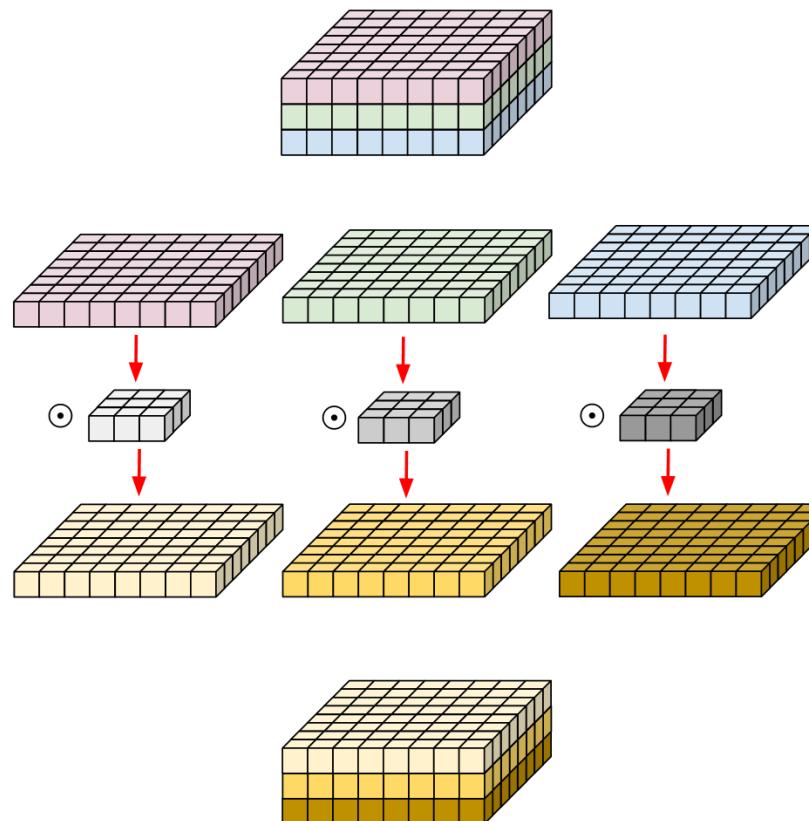
- No activations between the depthwise and pointwise operations of the separable convolution layers.



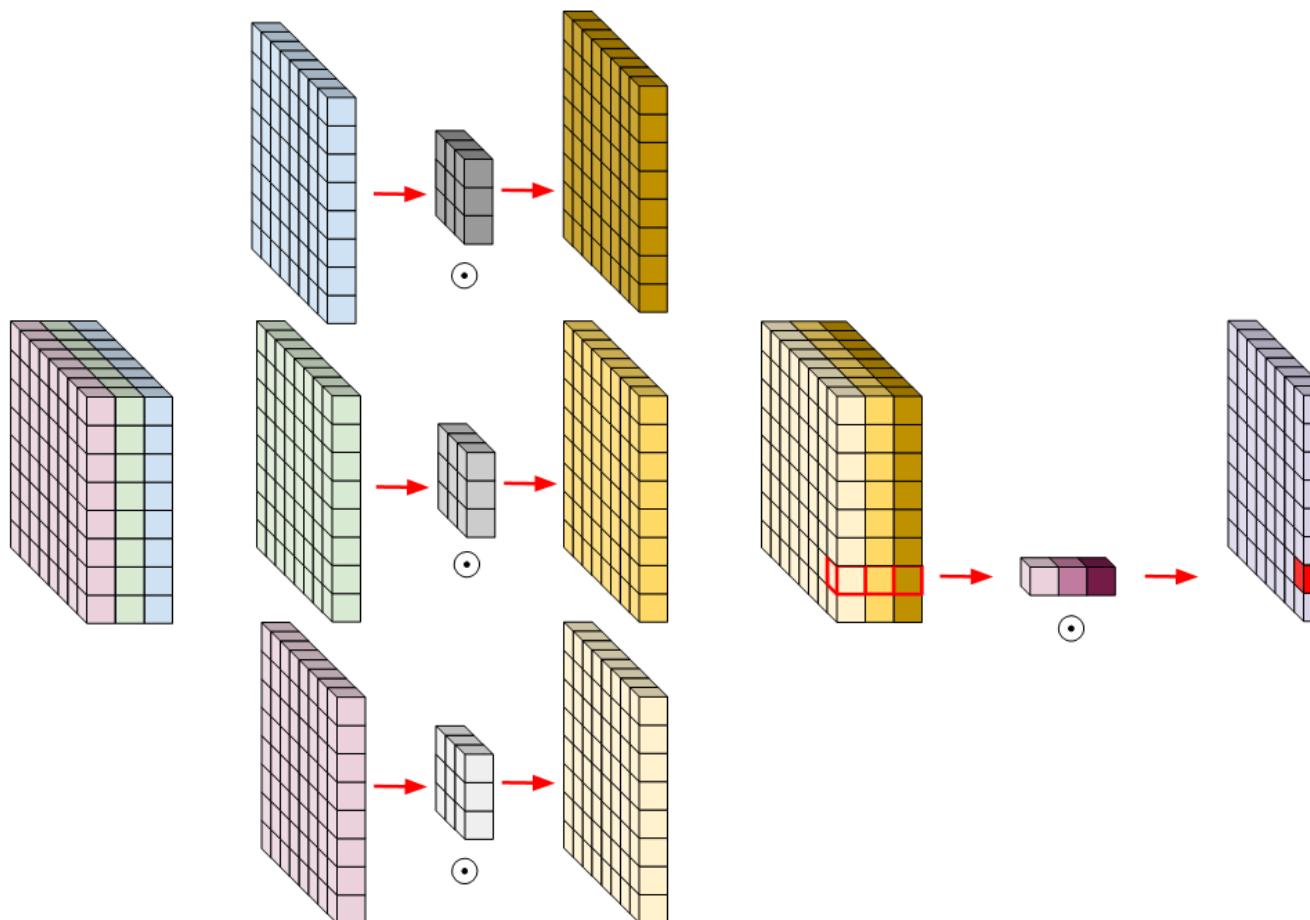
Mobilenet and MobilenetV2

- In order to implement a deep learning model suitable to use it for low machine performance like mobile devices, Google developed a model named MobileNet:
- Key idea:
 - Depth-wise separable convolutions (Mobilenet)
 - Invented residual block (MobilenetV2):
 - which allow a shortcut connection directly between the bottleneck layers

Remember: Depthwise Convolution

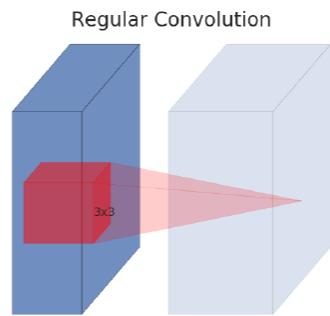


Remember: Depthwise Convolution

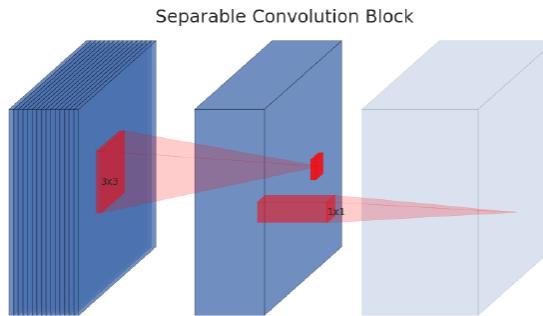


Four types of separable convolution blocks

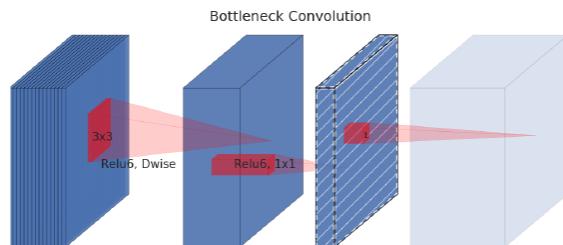
(a) Regular



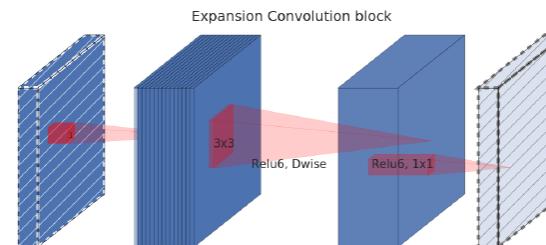
(b) Separable



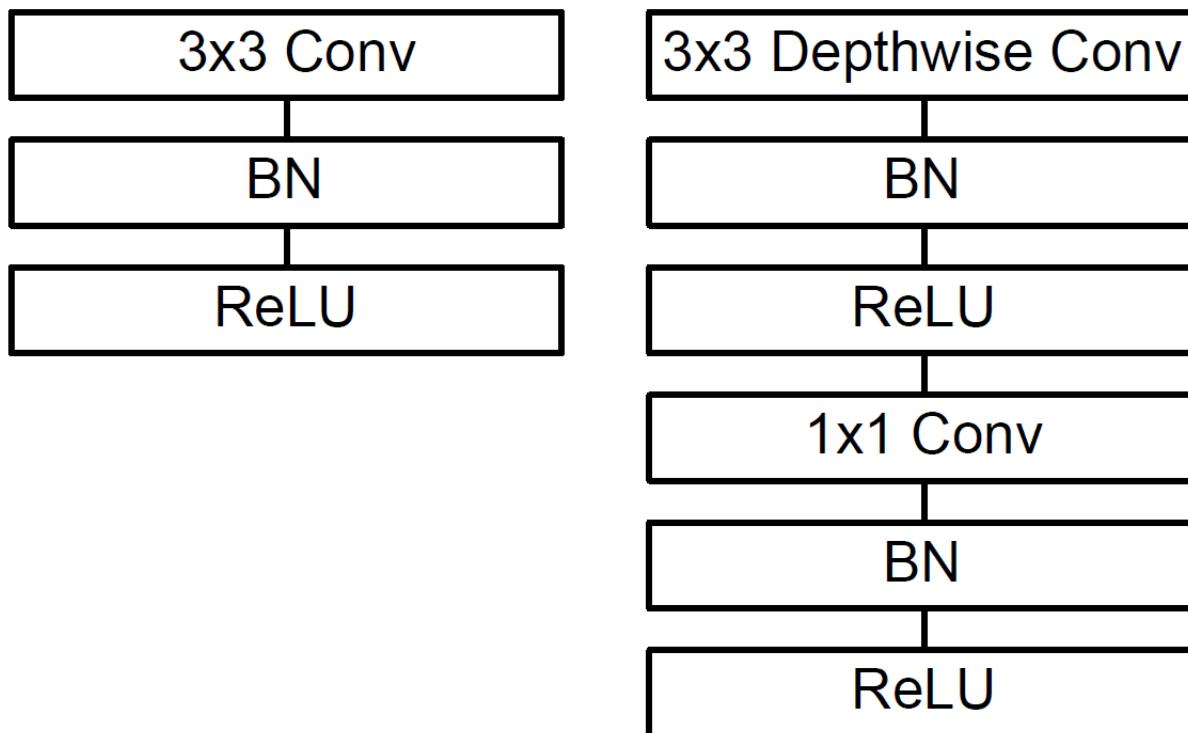
(c) Separable with linear bottleneck



(d) Bottleneck with expansion layer

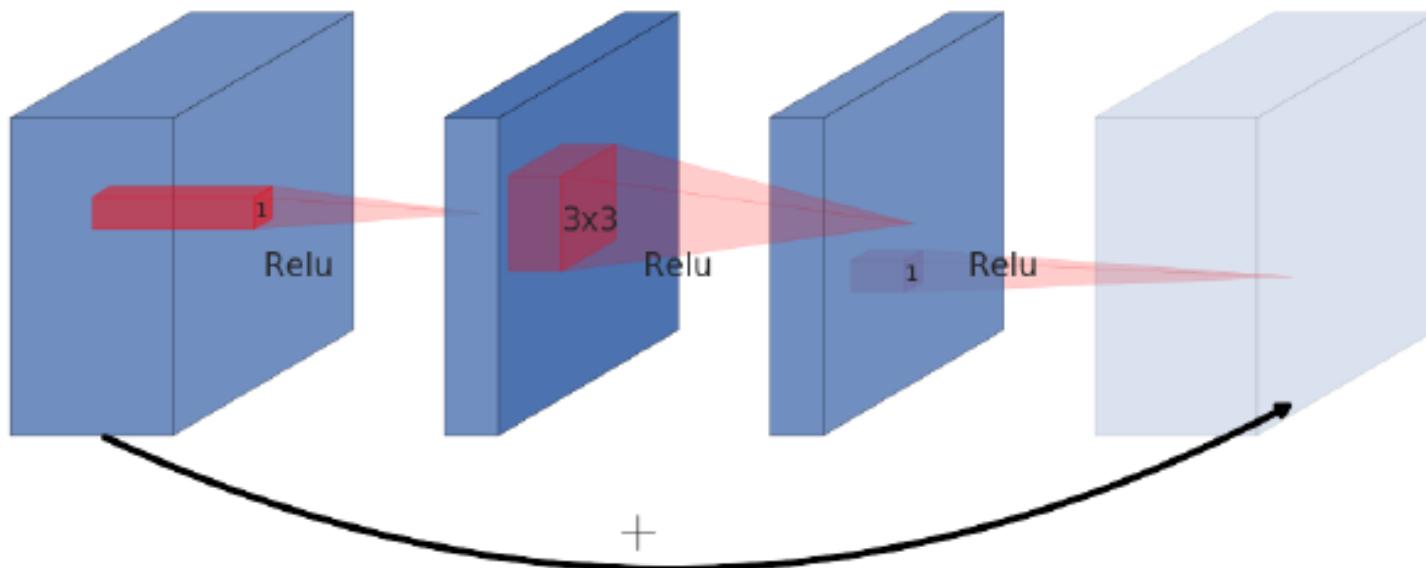


Mobilenet



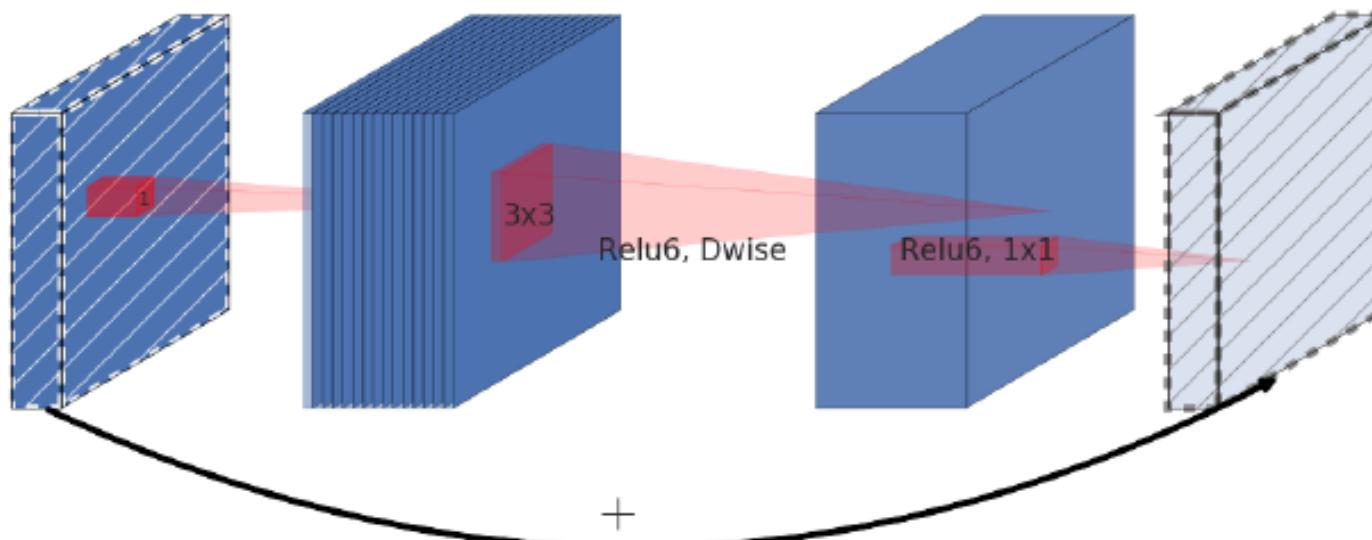
MobileNetV2: Inverted Residuals and Linear Bottlenecks

- Conventional Residual Block (Wide-Narrow-Wide)



MobileNetV2: Inverted Residuals and Linear Bottlenecks

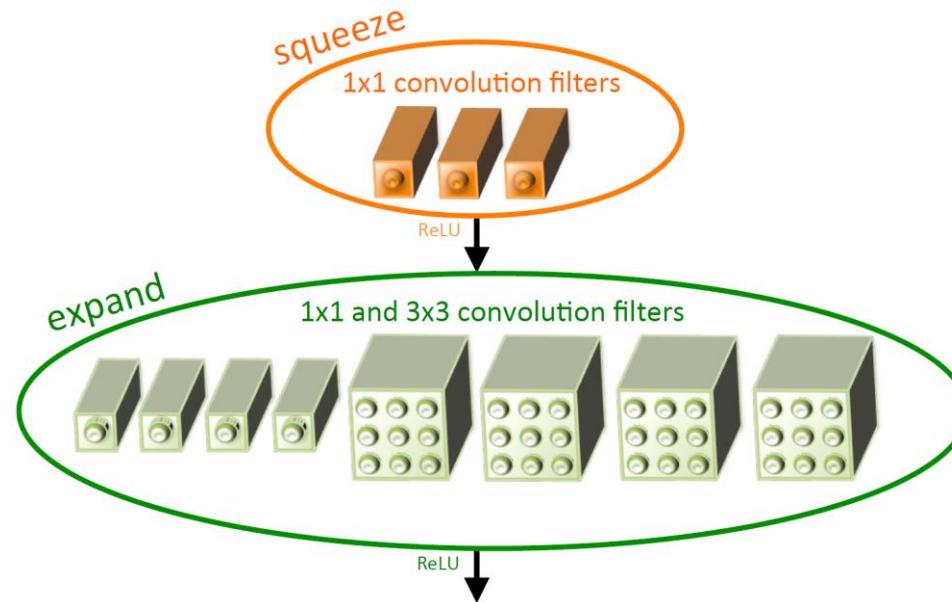
- Inverted Residual Block (Narrow-Wide-Narrow)



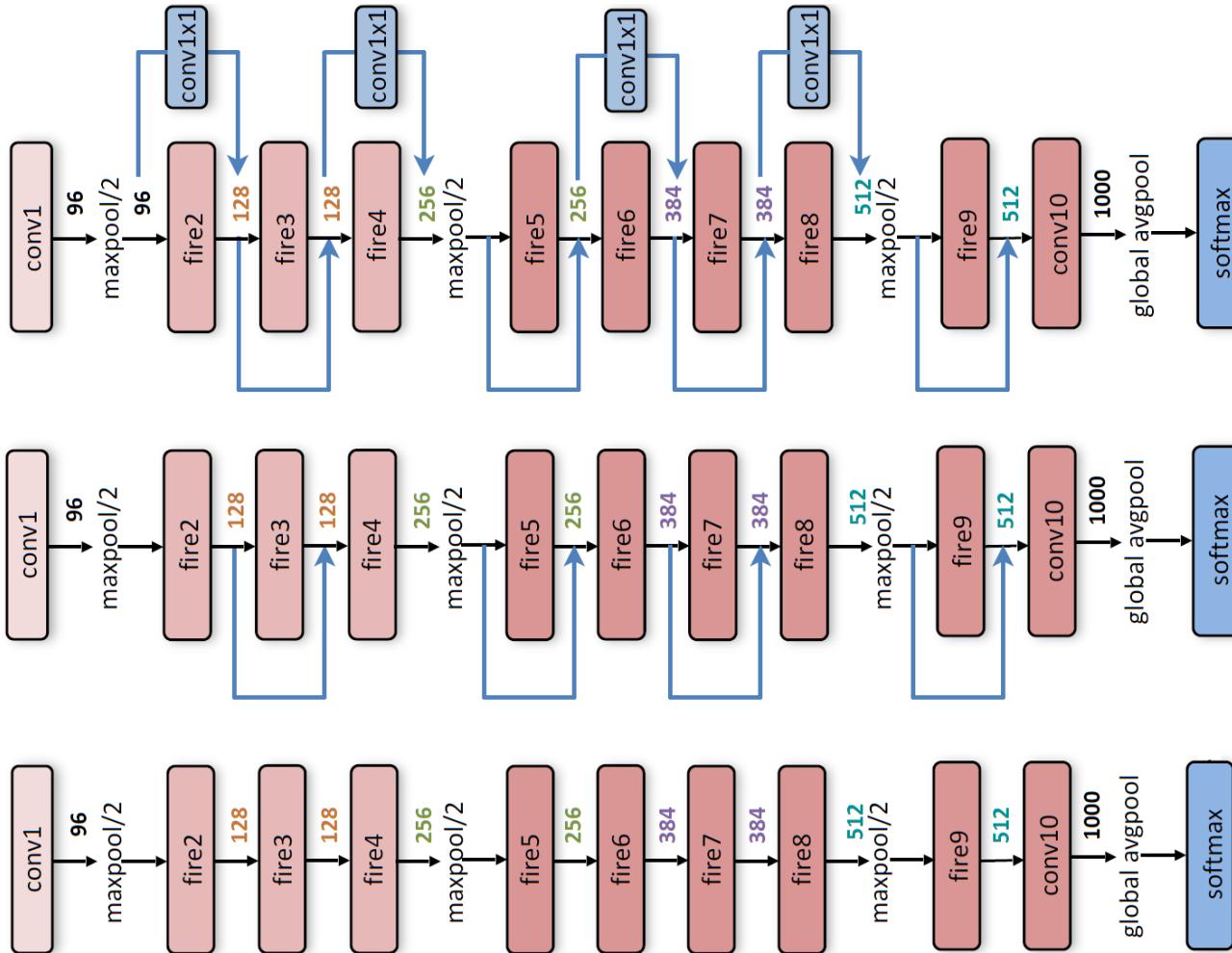
- Relu6 definition : $ReLU6(x) = \min(\max(0, x), 6)$

SqueezeNet

- Replace 3×3 filters with 1×1 filters in *almost* layers
- Decrease the number of input channels to 3×3 filters using squeeze layers
- Downsample late in the network so that convolution layers have large activation maps



SqueezeNet: Three proposed architectures

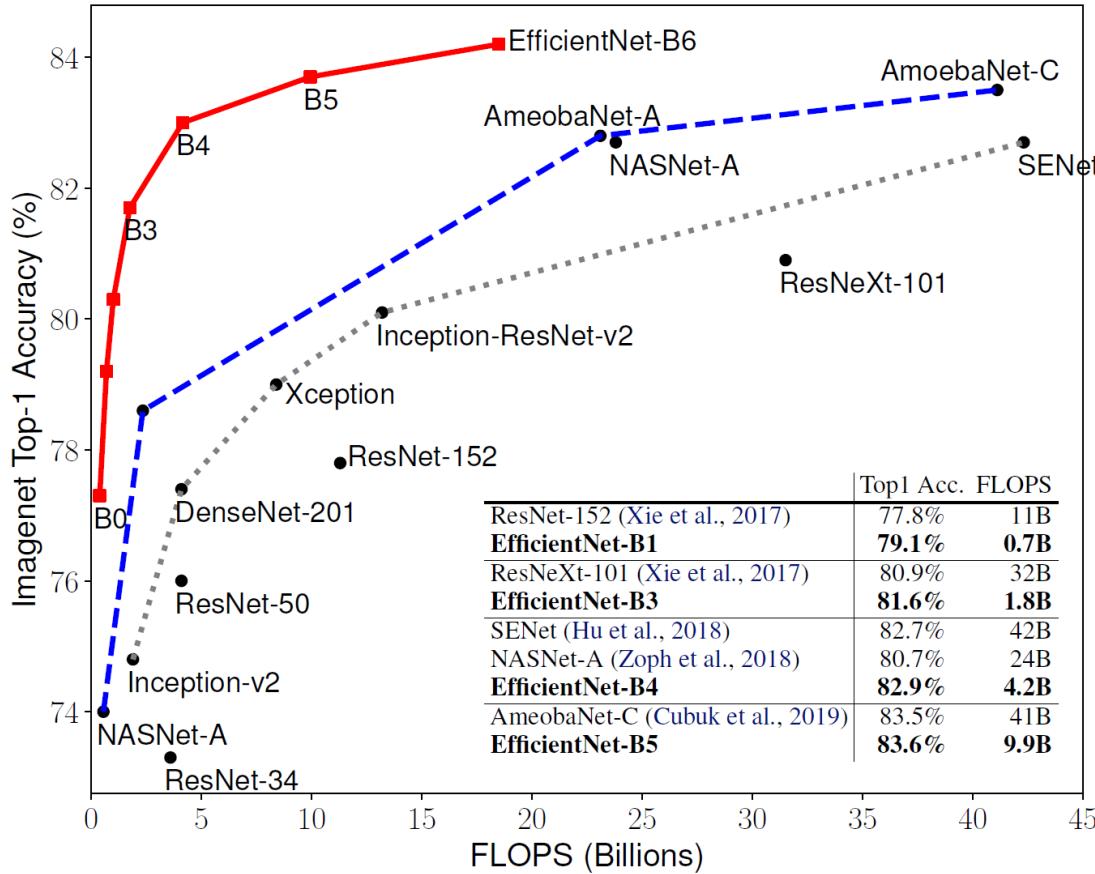


EfficientNet*

- There are three ways to scale a model:
 - Depth (# of layers)
 - Width (# of channels)
 - Resolution (# input image size)
- Target is to maximize the model accuracy for any given resource (memory and flops) constraints,
- In EfficientNet a hubristic optimization performed and eight models (B0 to B7) introduced (5.4 M to 66 M parameters)

EfficientNet

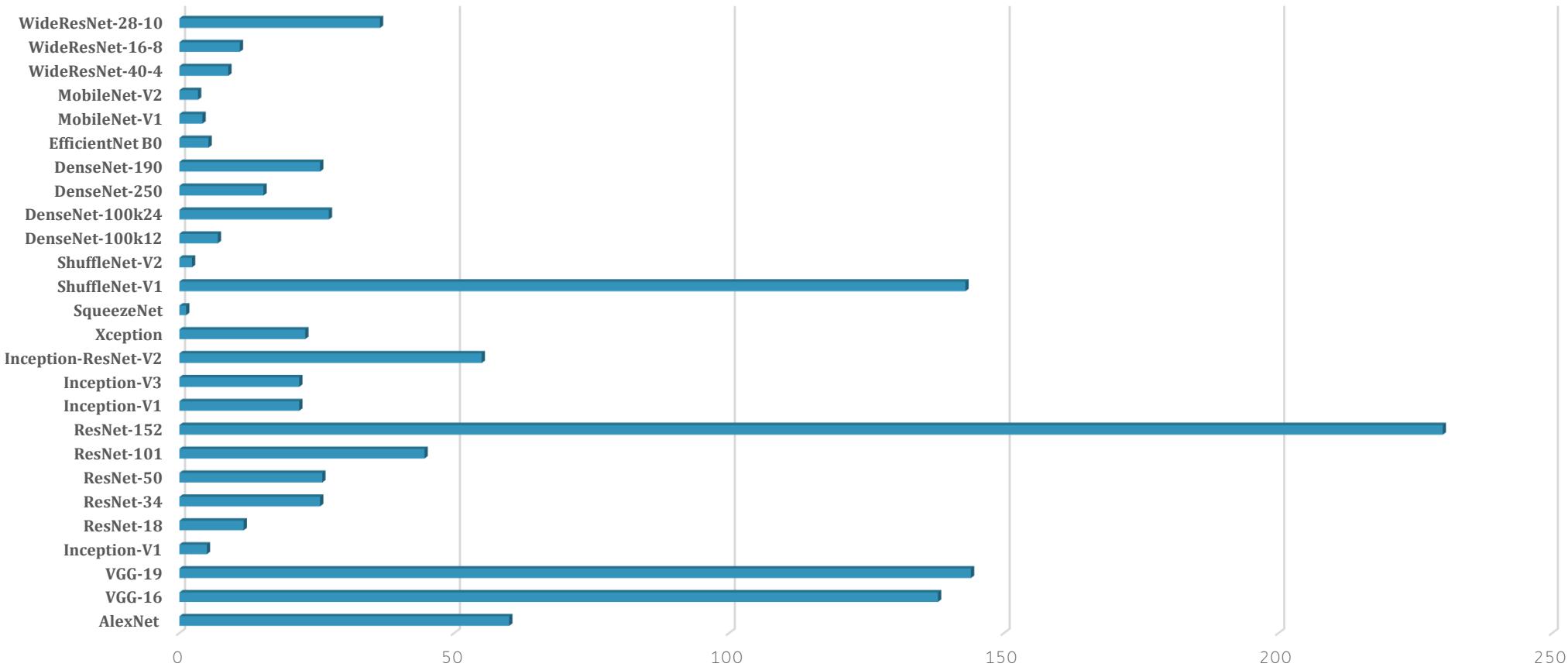
- Performance



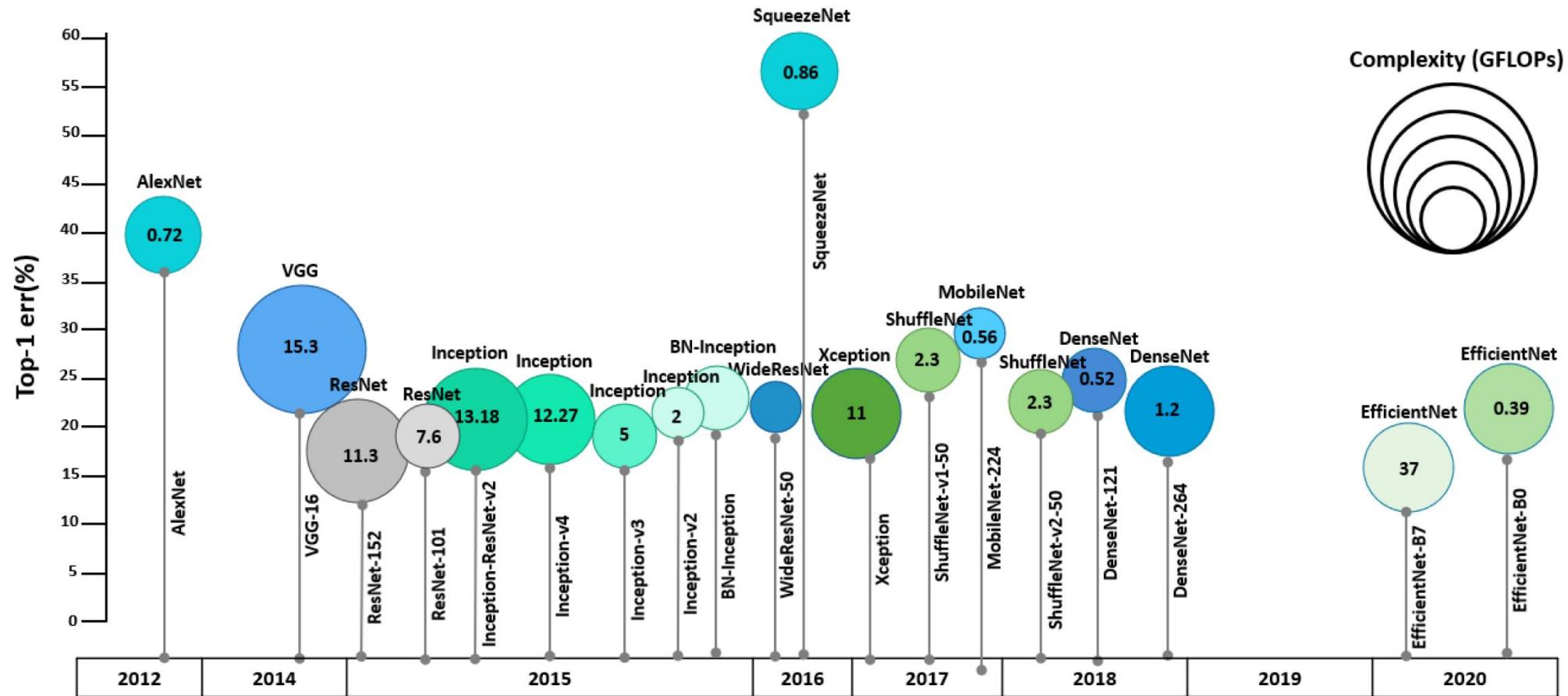
Other backbone

- ResNext
- DarkNet-19
- WideResNet
- SWideRNet
- HRNetV1 and HRNetV2
- See: Backbones-Review: Feature Extraction Networks for Deep Learning and Deep Reinforcement Learning Approaches (arXiv:2206.08016v1)

Model vs Number of parameters (M)



Timeline and performance accuracies of the proposed networks on ImageNet

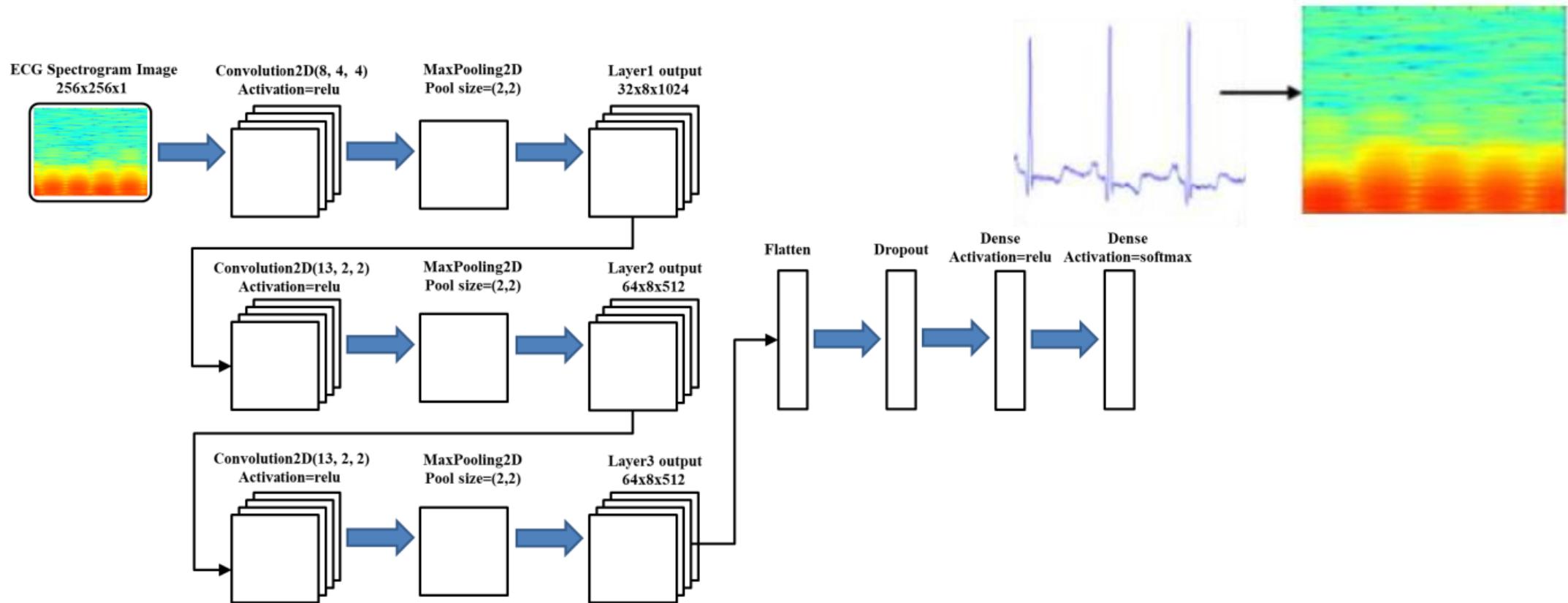


CNN for Signal Processing

- CNN and 1D signals:
 - 2D reshape → Conventional CNN architecture:
 - Multichannel Signals (Biological signals, market data, array sensor.,)
 - Group of Subjects
 - 2D (time and transform/feature/...) representation (STFT1, EMD2, ...)
 - 1D CNN:
 - One dimensional Convolution, Pooling,
- Time series deep structure (RNN, LSTM, ...)
- Short Time Fourier Transform
- Empirical Mode Decomposition

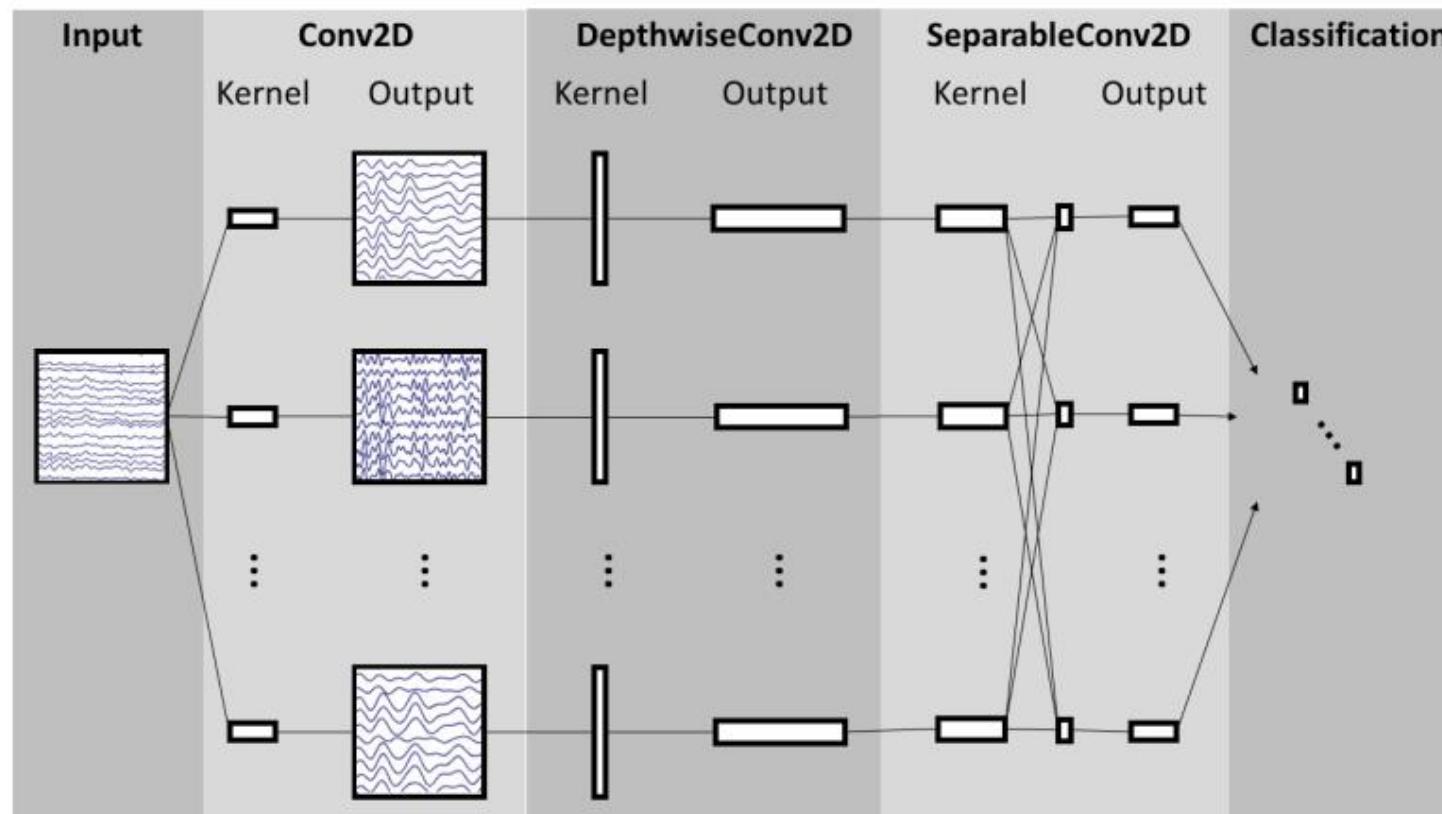
CNN for Signal Processing - ECG Arrhythmia Classification Using STFT-Based Spectrogram and CNN - 2019

- 2D representation via STFT

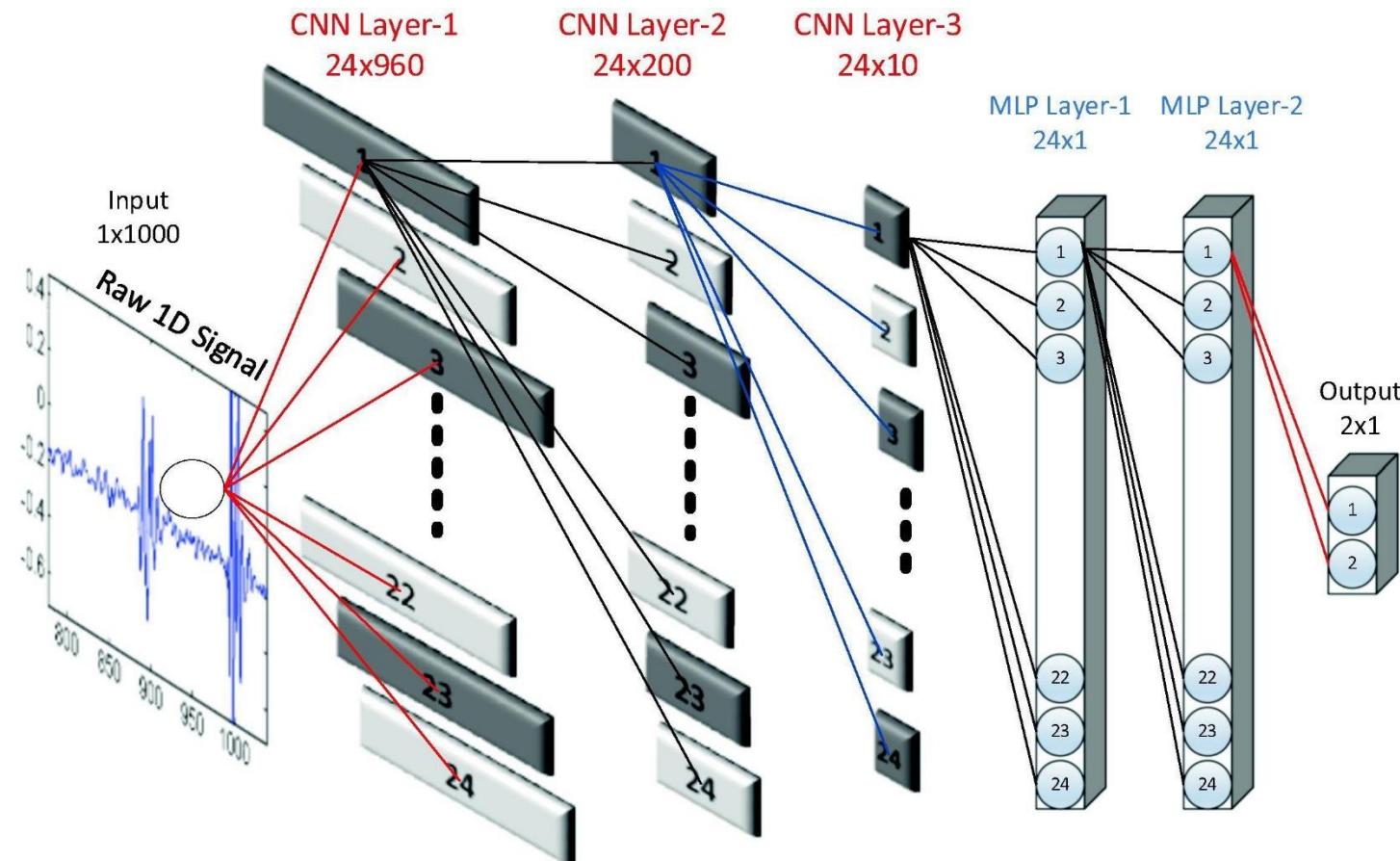


CNN for Signal Processing - EEGNet: A Compact Convolutional Neural Network for EEG-based Brain-Computer Interfaces

- 2D representation via Multichannel recording

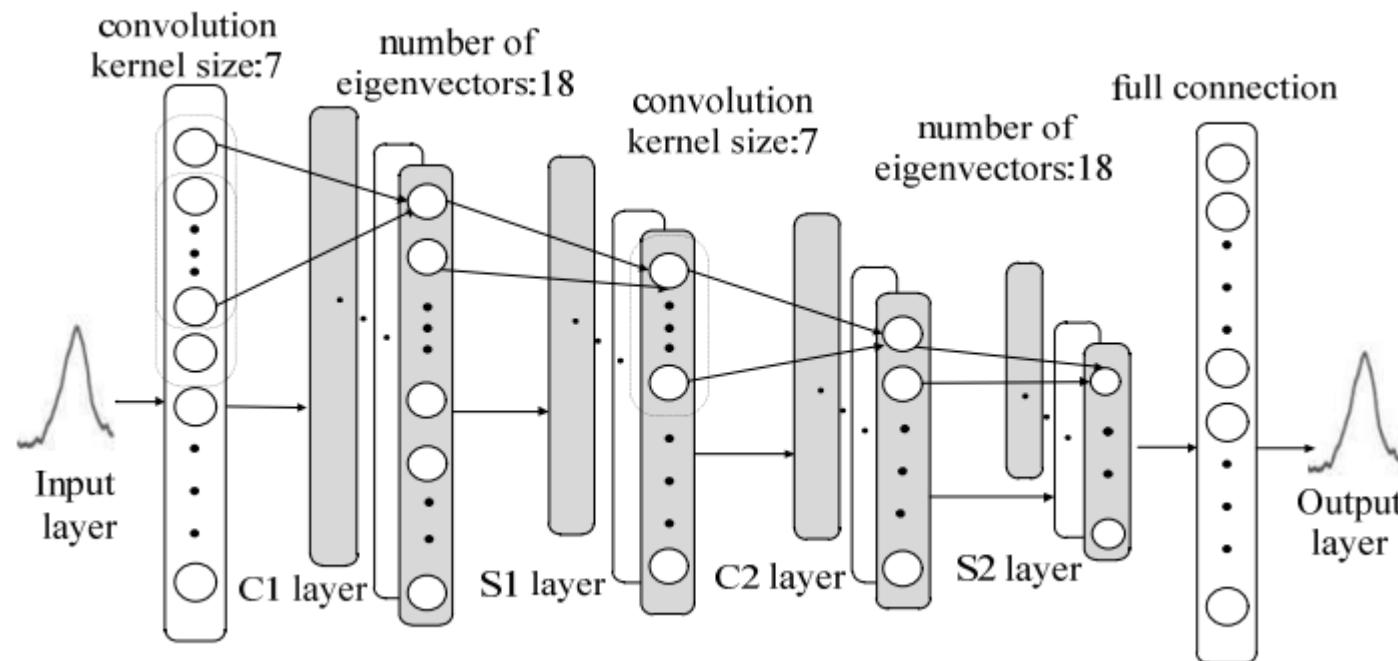


1D Signal Processing with CNN



CNN for Signal Processing - Classification of ECG Signals Based on 1D Convolution Neural Network

- 1D Signal processing



Transfer Learning

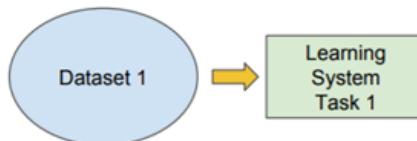
- A trained model one dataset#1 used for decision on dataset#2
- Vocabulary
 - Pre-trained Model
 - Fine-Tuning
 - Freezing

Transfer Learning

- Definition:

Traditional ML

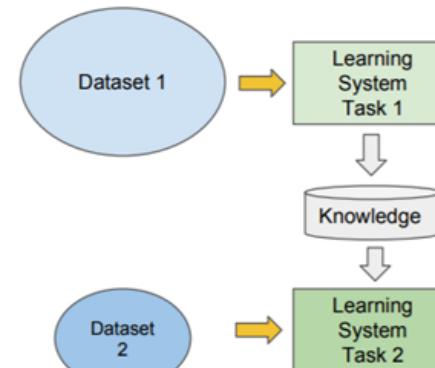
- Isolated, single task learning:
 - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



vs

Transfer Learning

- Learning of a new tasks relies on the previous learned tasks:
 - Learning process can be faster, more accurate and/or need less training data



Transfer Learning

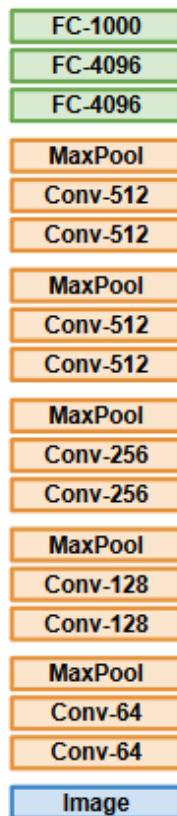
- Methodology in CNN:
 - ConvNet as fixed feature extractor, remove FC layer (4096-d feature vector)
 - Fine-tuning the ConvNet, fine-tune the weights of the pre-trained network by continuing the backpropagation with new data (all or some layers)
 - Pretrained models: Use without modification

Transfer Learning - <http://cs231n.github.io/transfer-learning/#tf>

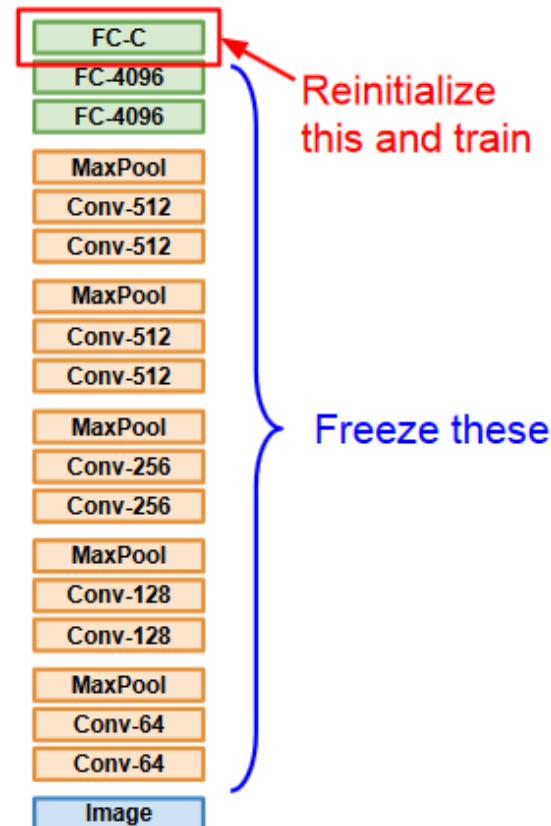
- When and How:
- New dataset is small and similar to original dataset:
 - Use final feature layer and train a linear or simple classifier
- New dataset is large and similar to the original dataset:
 - Fine-tune the full network
- New dataset is small but very different from the original dataset:
 - Train a SVM classifier from activations (middle layer, not final) in the network
- New dataset is large and very different from the original dataset:
 - Train from scratch (Initialize with weights from a pre-trained model)

Transfer Learning - cs231n_2019_lecture09.pdf

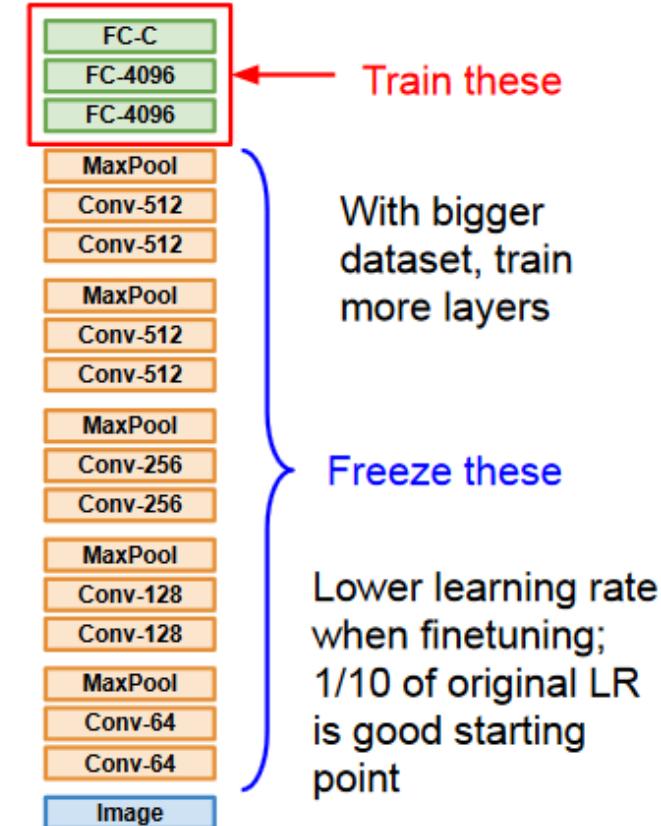
1. Train on Imagenet



2. Small Dataset (C classes)



3. Bigger dataset



AutoML

- Automated machine learning (AutoML) is the process of automating the tasks of applying machine learning to real-world problems.
 - The user provides preprocessed and cleaned data that is ready for use in training a ML model.
 - The AutoML system preprocesses (feature engineering and normalization) the data.
 - The AutoML system **trains** multiple machine learning models on the preprocessed data, using a variety of **algorithms** and **hyperparameters**. This allows the system to find the model that performs best on the data.
 - The AutoML system **evaluates** the **performance** of the trained models, and selects the one that performs the **best**. This model is then used as the output of the AutoML system.
 - The user can then use the trained model to make predictions or take actions based on new, unseen data.

AutoML

- Packages for Neural Architecture Search (NAS) and hyperparameter optimization for deep learning:
 - Run:ai
 - Google Cloud AutoML
 - Auto-SKLearn
 - MLBox
 - Auto-Keras
 - H2O AutoML
 - Amazon Lex
 - Azure AutoML
 - Auto-PyTorch
 - Talos
 - <https://www.automl.org/nas-overview/>
 - <https://www.automl.org/automl/>

CNN - Summary

- This is never-end story!
- Lots of models and growing
- Depth/Width is growing!
- New architecture are developing!
- ResNet and SENet: good choice!
- Model Zoo:
 - <https://modelzoo.co/framework/tensorflow>
 - <https://github.com/BVLC/caffe/wiki/Model-Zoo>
 - <https://github.com/onnx/models>
- New sight: Capsule Net, Meta Learning (Learn the architecture, ...)