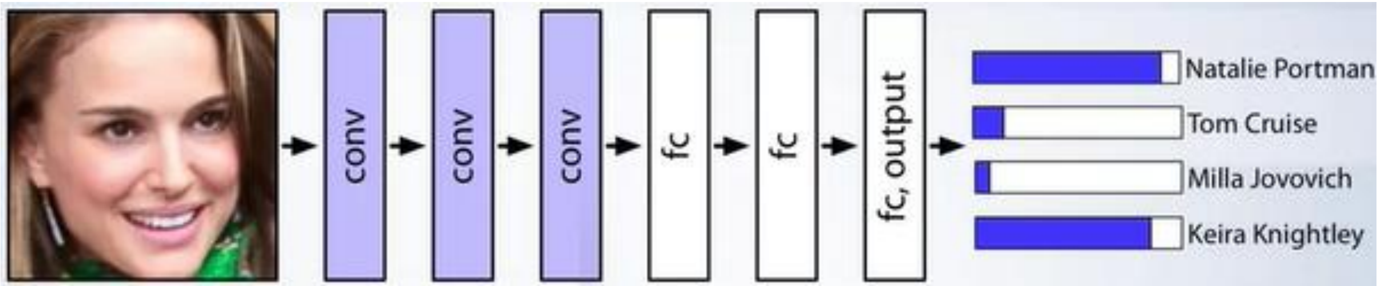
L3.1 CNN for Computer Vision

Zonghua Gu 2022



Acknowledgement: some contents taken from UC Berkeley CS231n https://cs231n.github.io
Coursera MOOC on CNN: https://www.coursera.org/learn/convolutional-neural-networks
Hung-yi Lee: https://speech.ee.ntu.edu.tw/~hylee/ml/2021-spring.html

Outline

- CNN Convolution layers
- Pooling and Fully-Connected layers
- CNN Case Studies

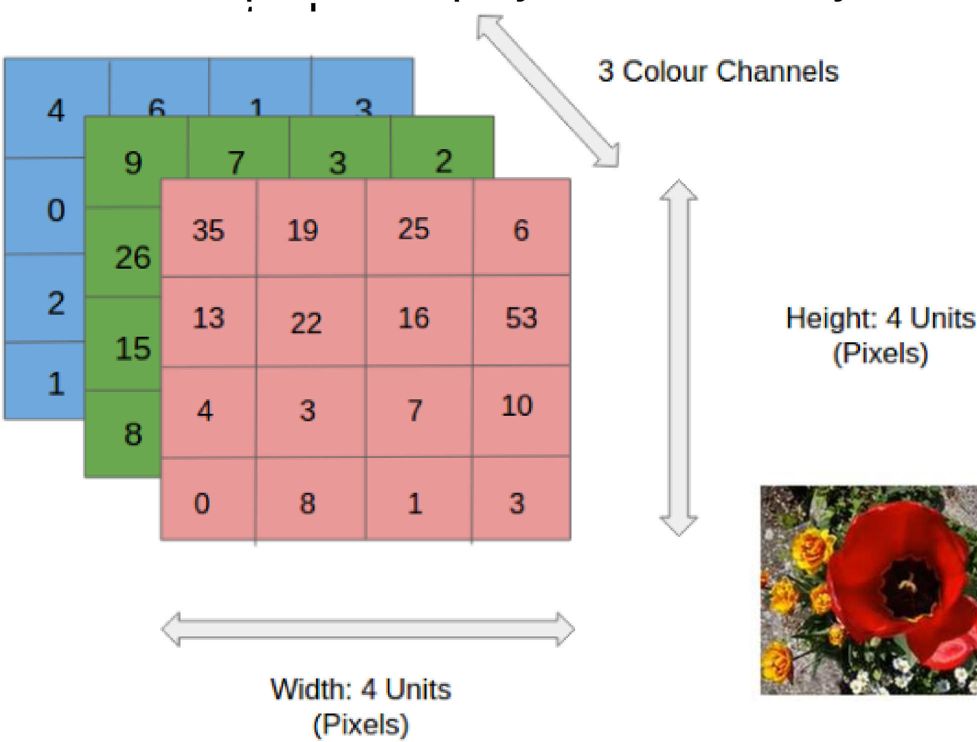
Classic Computer Vision

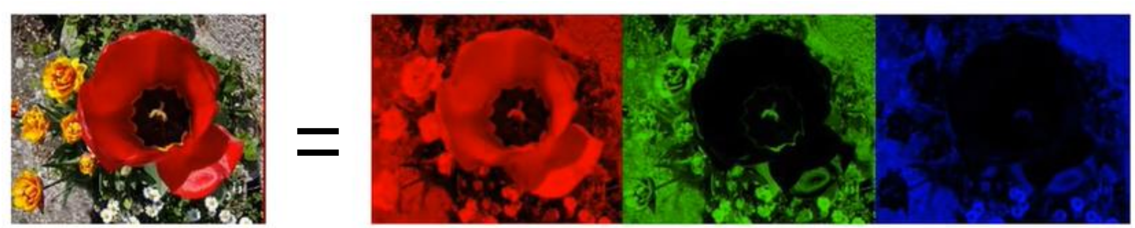
- Most “classic” (non-ML) CV algorithms are implemented in the OpenCV library, including
  - Core Operations:
    - basic operations on image like pixel editing, geometric transformations...
  - Image Processing
    - Thresholding, smoothing, edge detection, Hough Line Transform…
  - Feature Detection and Description
    - HOG, SIFT, SURF, BRIEF, ORB…
  - Video analysis
    - Object tracking w. optical flow
  - Camera Calibration and 3D Reconstruction
- They are simple, fast and reliable (e.g., for lane detection), and are often used in place of or in conjunction w. complex ML/DL algorithms, which may sometimes be unreliable and unpredictable.

OpenCV

Input Image Encoding

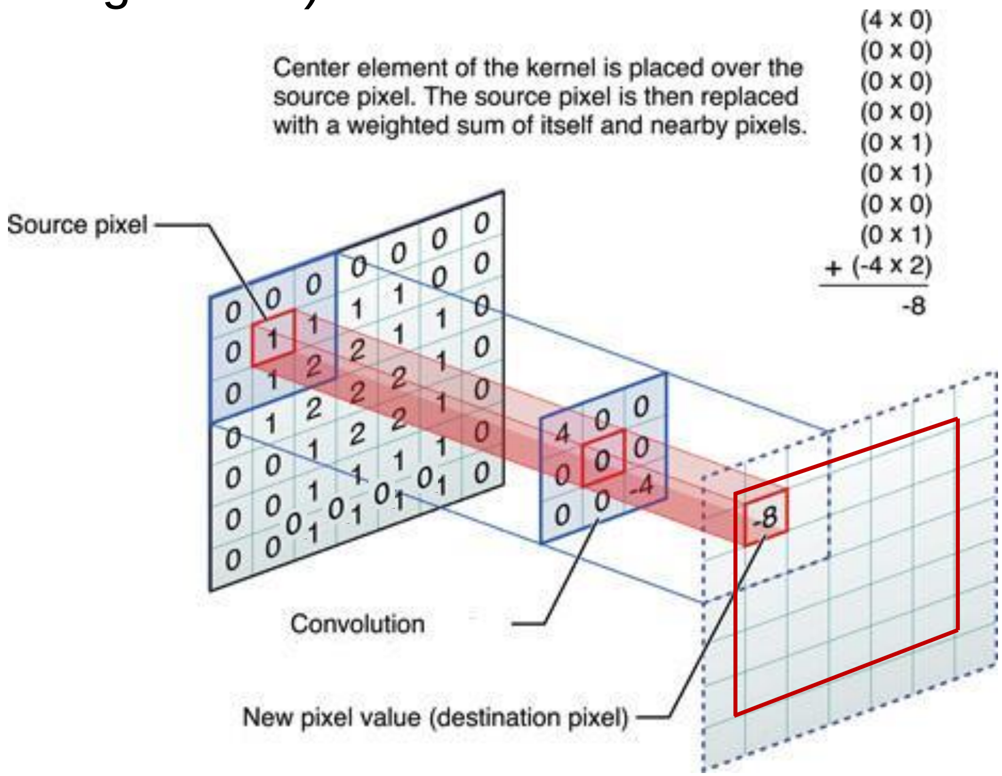
- A size $N \times N$ color image has volume $N \times N \times 3$, w. $N \times N$ pixels and 3 color components (Red, Green, and Blue, RGB) for each pixel
- A size $N \times N$ greyscale image has volume $N \times N \times 1$
- Color depth, or bit depth, is number of bits used for each color component of a single pixel
  - Typical value is 8, so pixel value has range $[0, 255]$
  - Larger depth is possible, e.g., true color (24-bit) is used in computer and phone displays for human eyes, but 8-bit is typically enough for CV

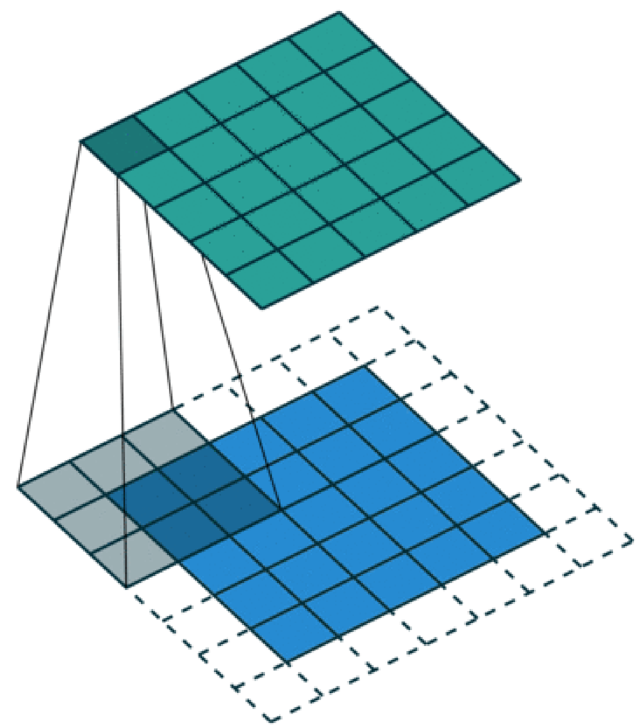




Filters/Kernels in Computer Vision

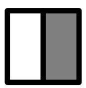
- Convolution operation: we slide each filter (also called kernel) across the width and height of the input volume, and compute dot products between the entries of the filter and the input. As we slide the filter over the width and height of the input volume, we will produce a 2D activation map (also called feature map) that gives the responses of that filter at every spatial position.
  - Dot product: elementwise multiplication of a filter w. corresponding input values, then summing them to generate one output value
- Used to extract features for downstream tasks (classification or regression)





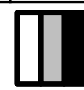
A Filter for Vertical Edge Detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0



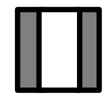
*

1	0	-1
1	0	-1
1	0	-1

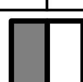


=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

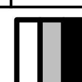


0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10



*

1	0	-1
1	0	-1
1	0	-1



=

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

Sobel Filter for Vertical Edge Detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
2	0	-2
1	0	-1

=

0	40	40	0
0	40	40	0
0	40	40	0
0	40	40	0

0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

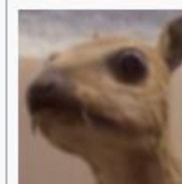
*

1	0	-1
2	0	-2
1	0	-1

=

0	-40	-40	0
0	-40	-40	0
0	-40	-40	0
0	-40	-40	0

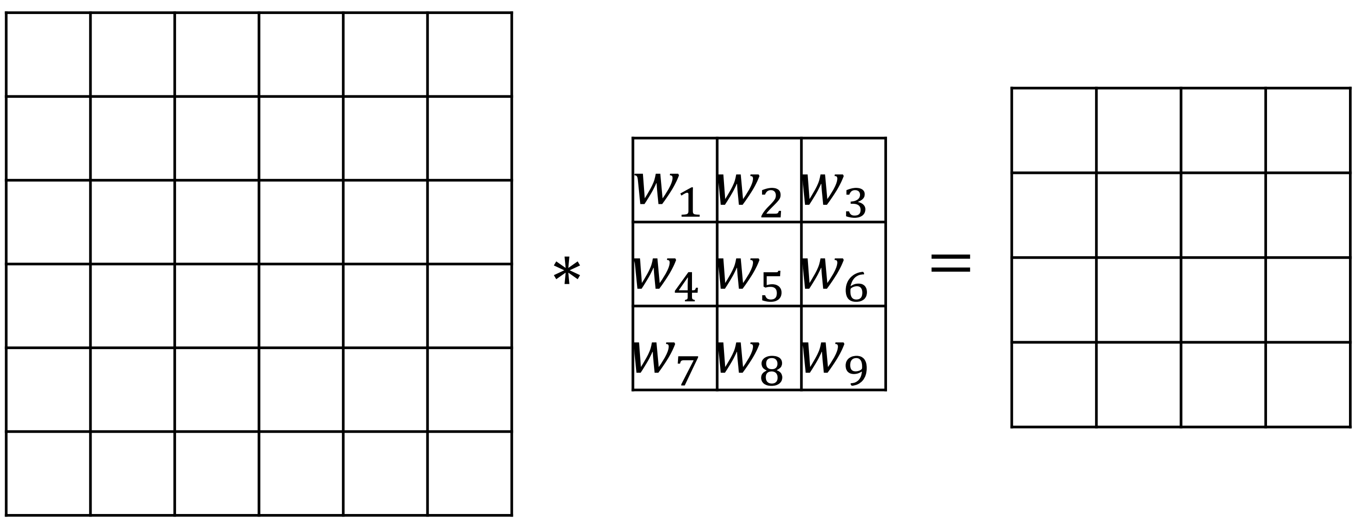
Common Filters in CV

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

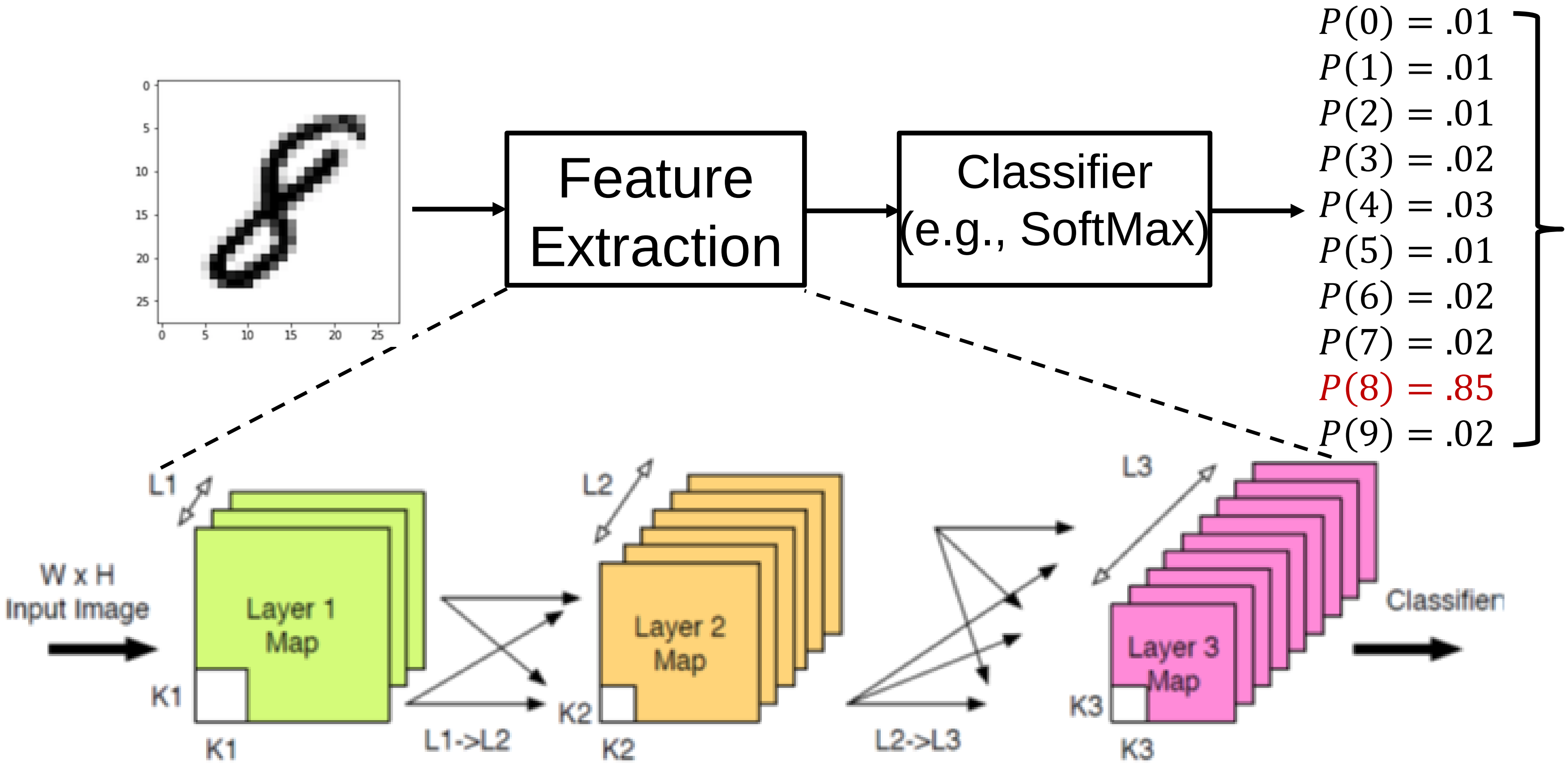
- These filters were designed, or “hand-crafted”, by CV researchers. They extract features used by downstream tasks such as classification, image segmentation, etc.

Machine Learning Meets CV

- Instead of hand-crafted filters in classic CV, why not learn custom filters from data by supervised learning?
  - For easy tasks like edge detection, learning may recover filters similar to hand-crafted ones.
  - For difficult tasks like cat vs. dog classification, learning is essential to achieving good results



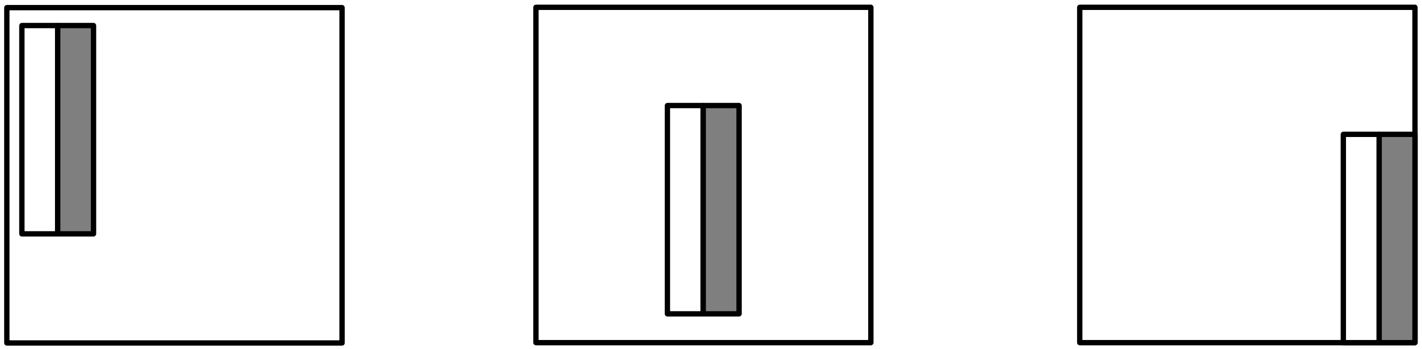
Convolutional Neural Networks (CNN)



- A CNN (also called ConvNet) is a sequence of Convolutional (CONV) Layers, Pooling (POOL) Layers and non-linear activation functions for feature extraction, followed by one or more Fully-Connected (FC) Layers for classification based on the extracted features

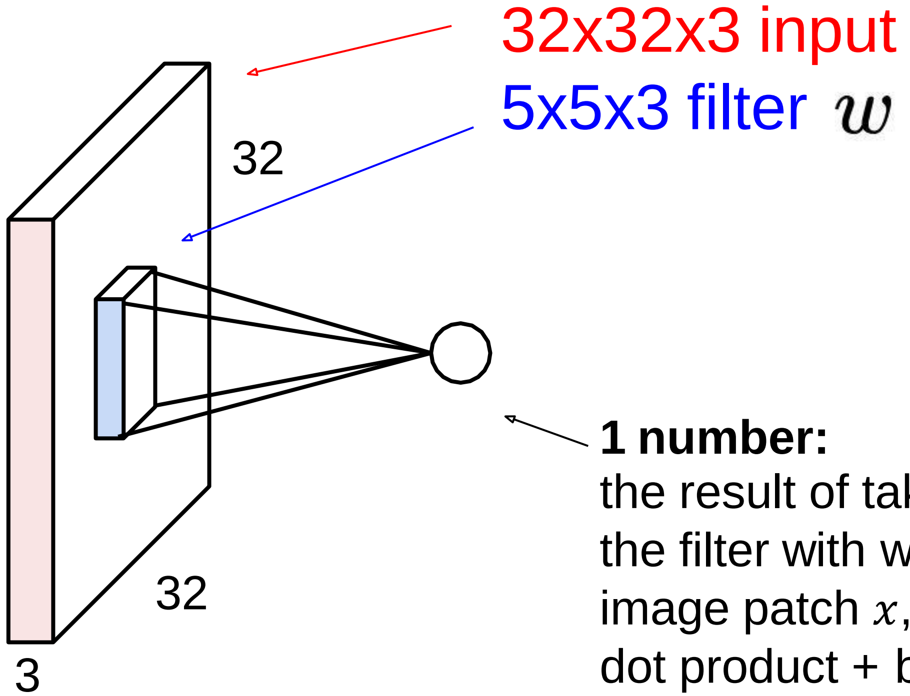
Receptive Field and Parameter Sharing

- Each neuron in a CONV layer has local, sparse connectivity to a small patch of the input volume w. size of the filter, called its Receptive Field
  - Each neuron covers a limited, narrow “field-of-view”
  - In contrast, each neuron in a FC layer has RF that covers the entire input volume
- Parameter sharing: all neurons in the same CONV layer share the same filter params w, b
  - It helps to reduce the number of params significantly compared to fully-connected networks
  - It gives translation invariance, e.g., an edge can be detected regardless of its location in the image

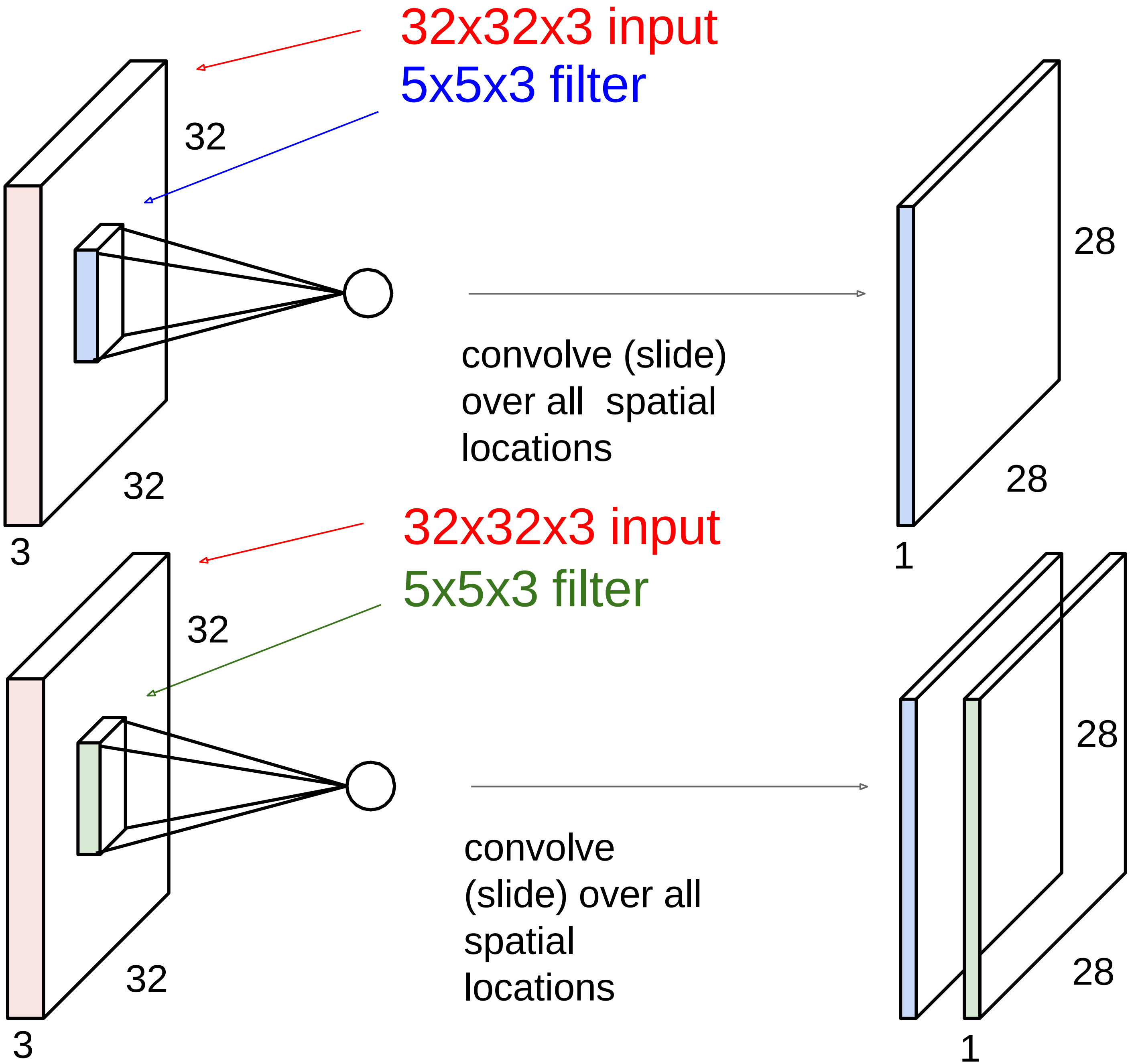


Convolution Operation

- Slide the filter over the image spatially, computing dot products $w^T x + b$ to generate an activation map as output
- The input may be an input RGB image w. 3 channels, hence depth=3, or intermediate activation maps generated by hidden layers of a CNN. We use the terms "input volume" and "output volume" to emphasize they may be 3D tensors



1 number:
the result of taking a dot product $wx + b$ between the filter with weights w, bias b, and a $5 \times 5 \times 3$ image patch x, with $5 * 5 * 3 = 75$-dimensional dot product + bias

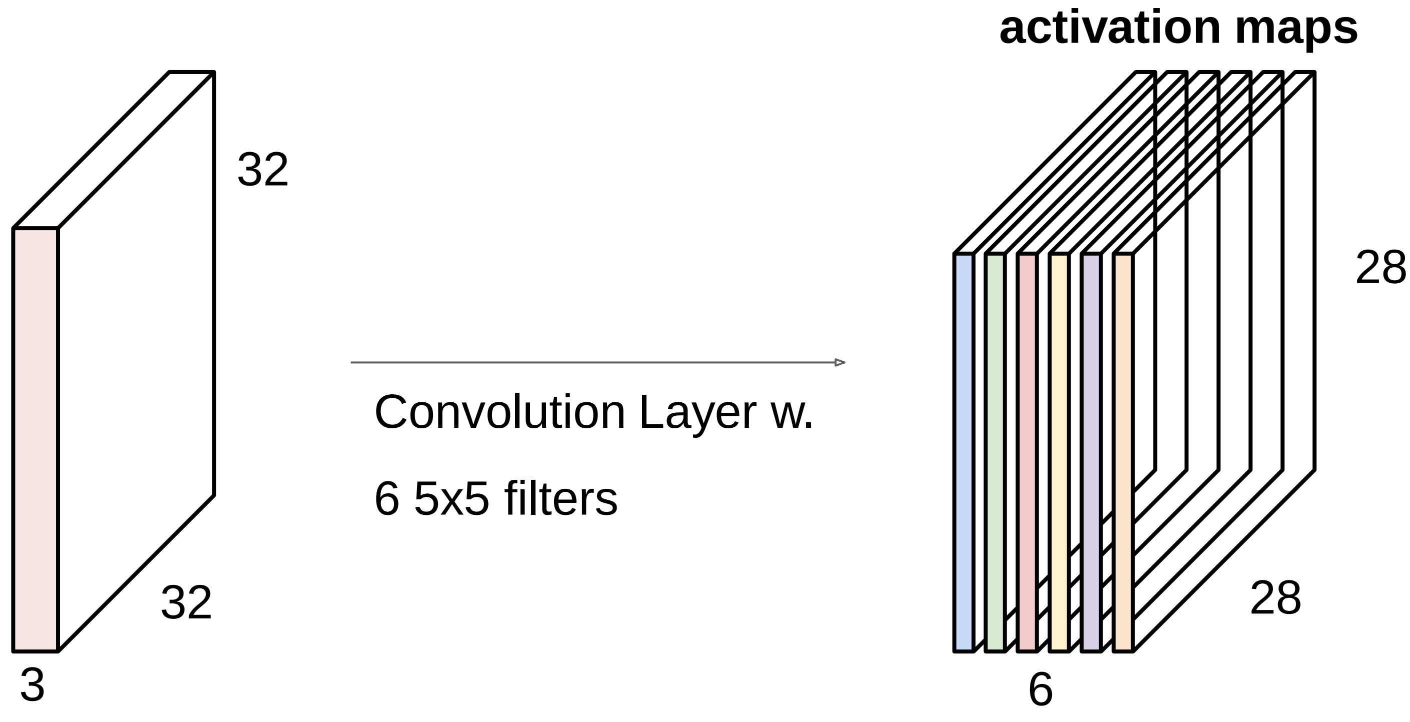


One (blue) filter generates one 2D activation map as output

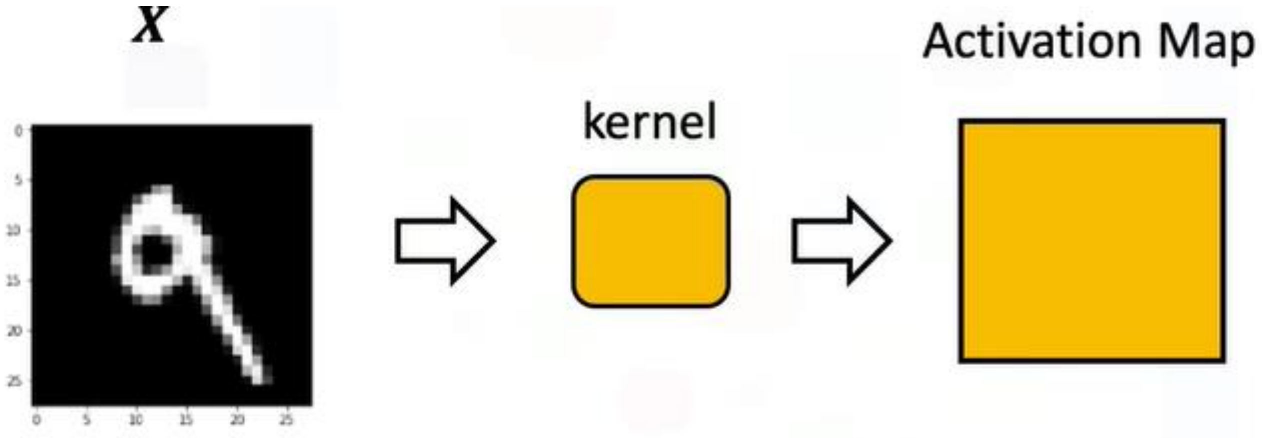
Two filters (blue and green) generate two 2D activation maps (blue and green), stacked along the depth dimension to produce a 3D output volume

Stacked Activation Maps

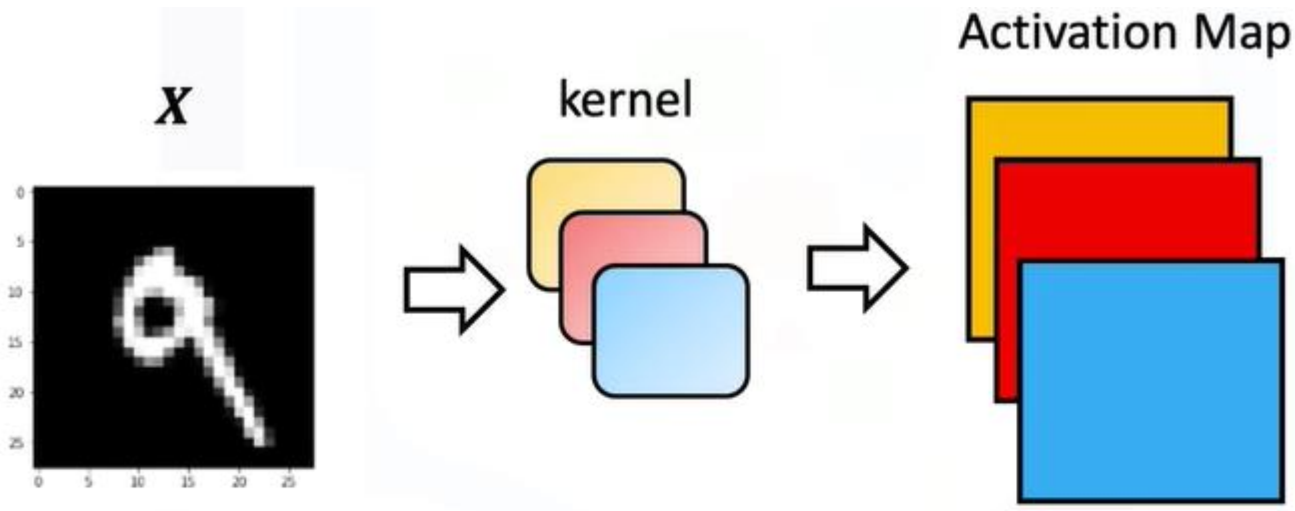
- If we have 6 5×5 filters, we'll get 6 different activation maps (feature maps), each computed by convolution of one filter with the input
  - For each 5×5 patch of the input, there are 6 different neurons looking at it, each extracting different features
- We stack these up to get an output volume (a new "image") of size $28 \times 28 \times 6$, an intermediate representation to be passed to subsequent layers



Activation Maps Illustration



1 filter/kernel, 1 output activation map

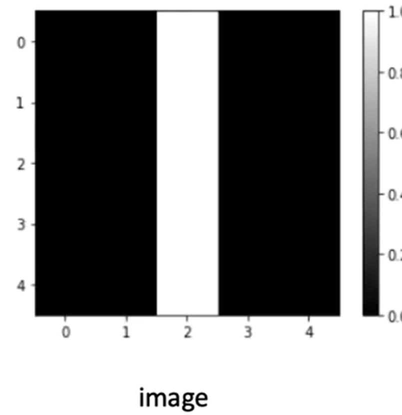


3 filters/kernels, 3 output activation maps

Concrete Example: 3 Filters

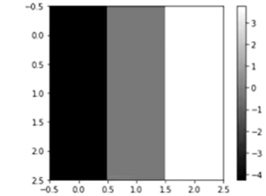
- 3 filters $\boldsymbol{W_0}$, $\boldsymbol{W_1}$, $\boldsymbol{W_2}$, each extracting different features. ($\boldsymbol{W_i} * \boldsymbol{X_j}$ denotes convolution of filter $\boldsymbol{W_i}$ w. input $\boldsymbol{X_j}$) (bias terms are assumed to be 0 here)
- Upper left: filter $\boldsymbol{W_0}$ extracts vertical line features $\boldsymbol{Z_0}$ from input image $\boldsymbol{X_1}$. (the other 2 filters do not extract any meaningful features)
- Lower left: filter $\boldsymbol{W_1}$ extracts horizontal line features $\boldsymbol{Z_1}$ from input image $\boldsymbol{X_2}$ (the other 2 filters do not extract any meaningful features)

$\mathbf{X}_1$



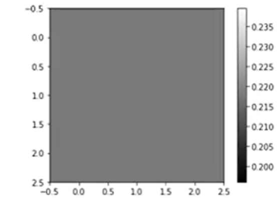
$\boldsymbol{Z}_0 = \boldsymbol{W}_0 * \mathbf{X}_1 + \boldsymbol{b}_0$

1	0	-1
1	0	-2
1	0	-1



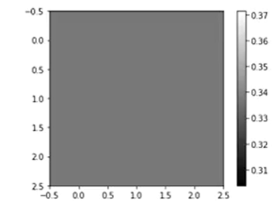
$\boldsymbol{Z}_1 = \boldsymbol{W}_1 * \mathbf{X}_1 + \boldsymbol{b}_1$

1	2	-1
0	0	0
-1	-2	-1

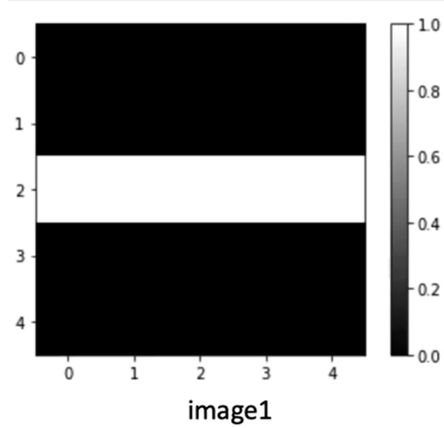


$\boldsymbol{Z}_2 = \boldsymbol{W}_2 * \mathbf{X} \quad + \boldsymbol{b}_2$

1	1	1
1	1	1
1	1	1

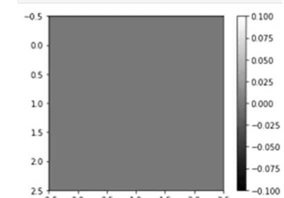


$\mathbf{X}_2$



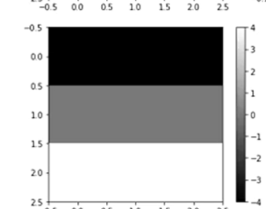
$\boldsymbol{Z}_0 = \boldsymbol{W}_0 * \mathbf{X}_2 + \mathrm{b}_0$

1	0	-1
1	0	-2
1	0	-1



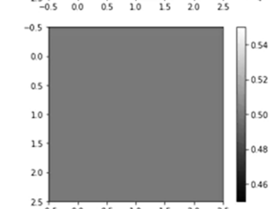
$\boldsymbol{Z}_1 = \boldsymbol{W}_1 * \mathbf{X}_2 + b_1$

1	2	-1
0	0	0
-1	-2	-1

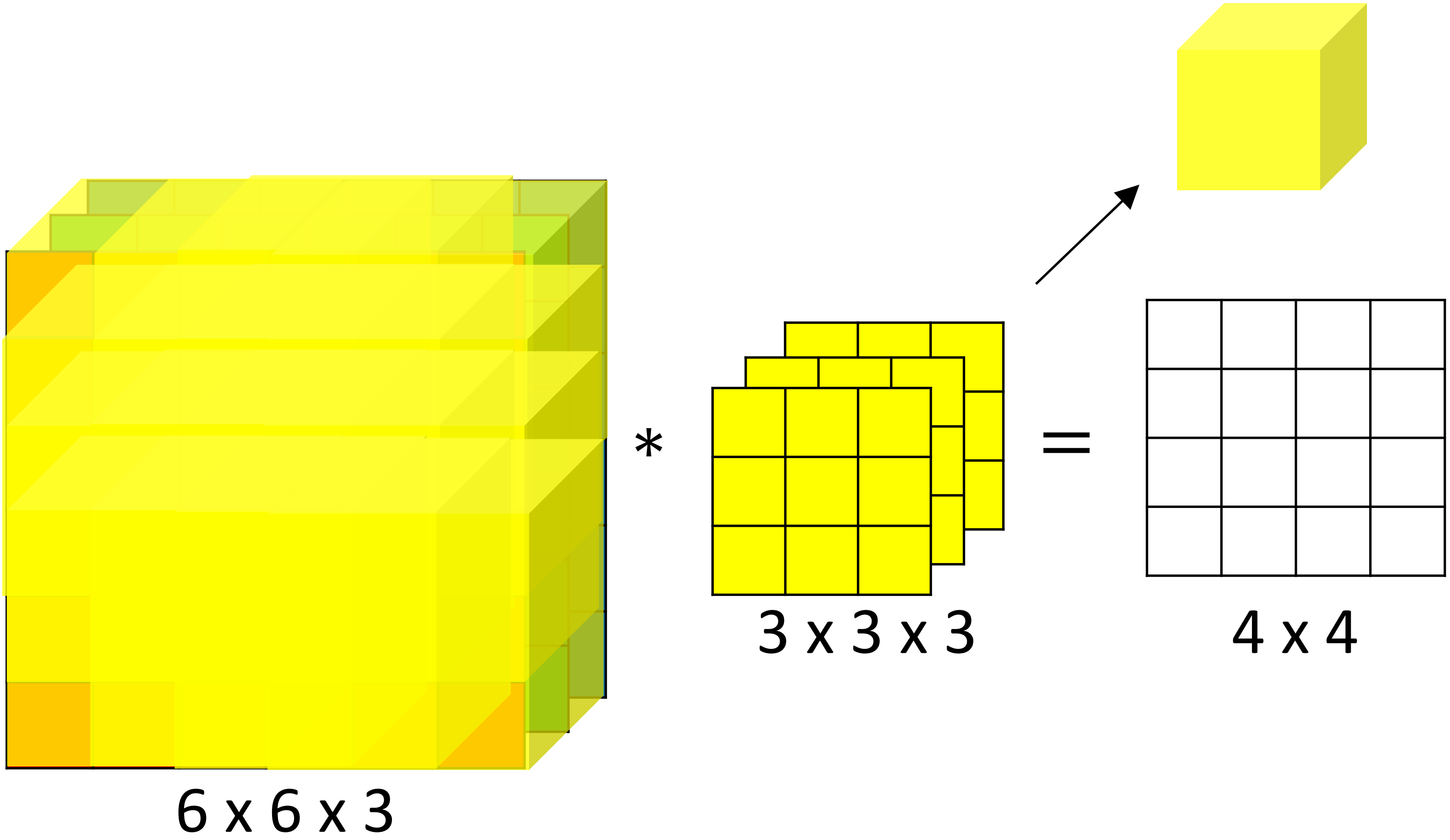


$\boldsymbol{Z}_2 = \boldsymbol{W}_2 * \mathbf{X}_2 + \mathrm{b}_2$

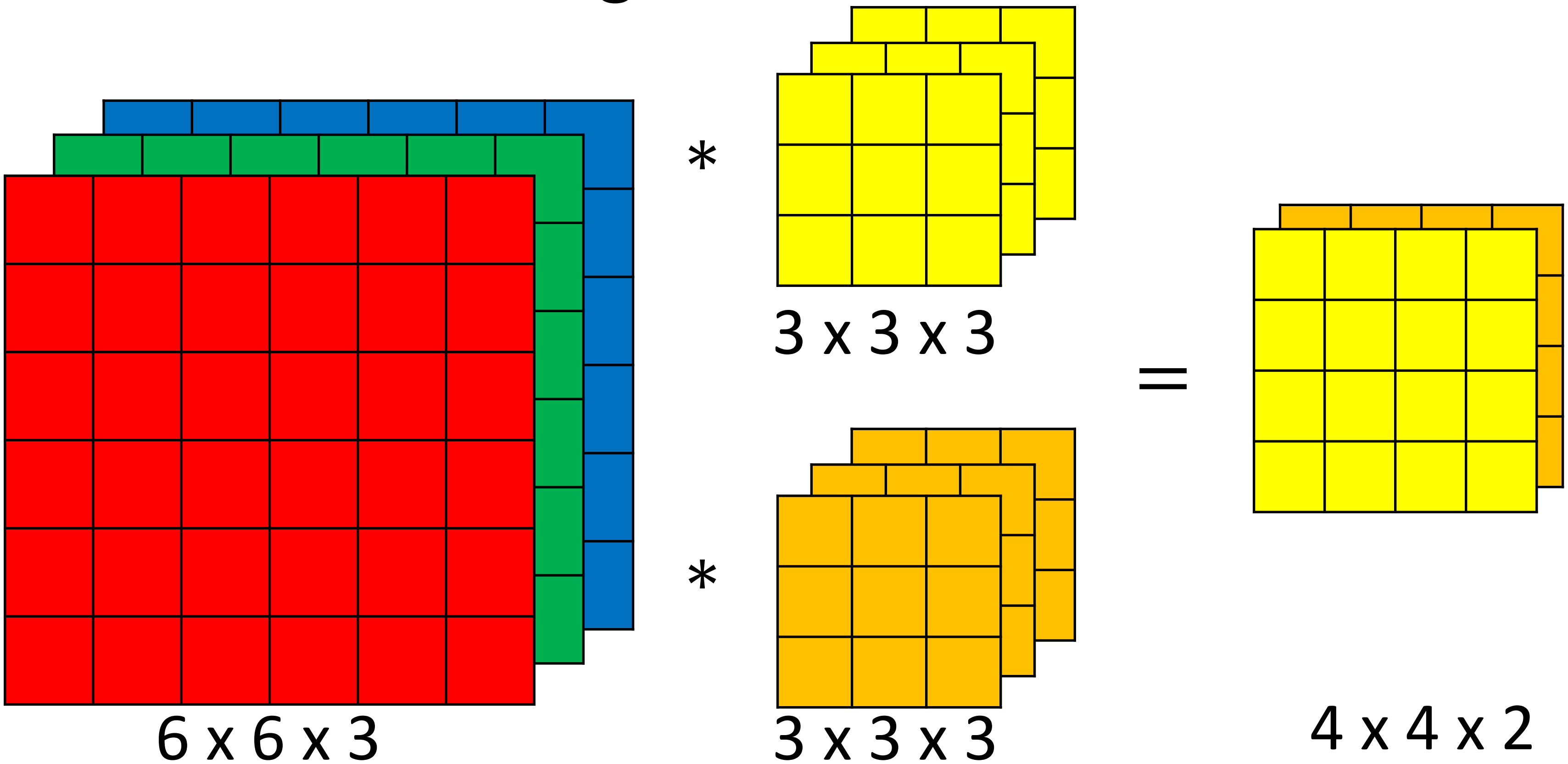
1	1	1
1	1	1
1	1	1



Convolution of a Filter on RGB Image w. 3 Channels



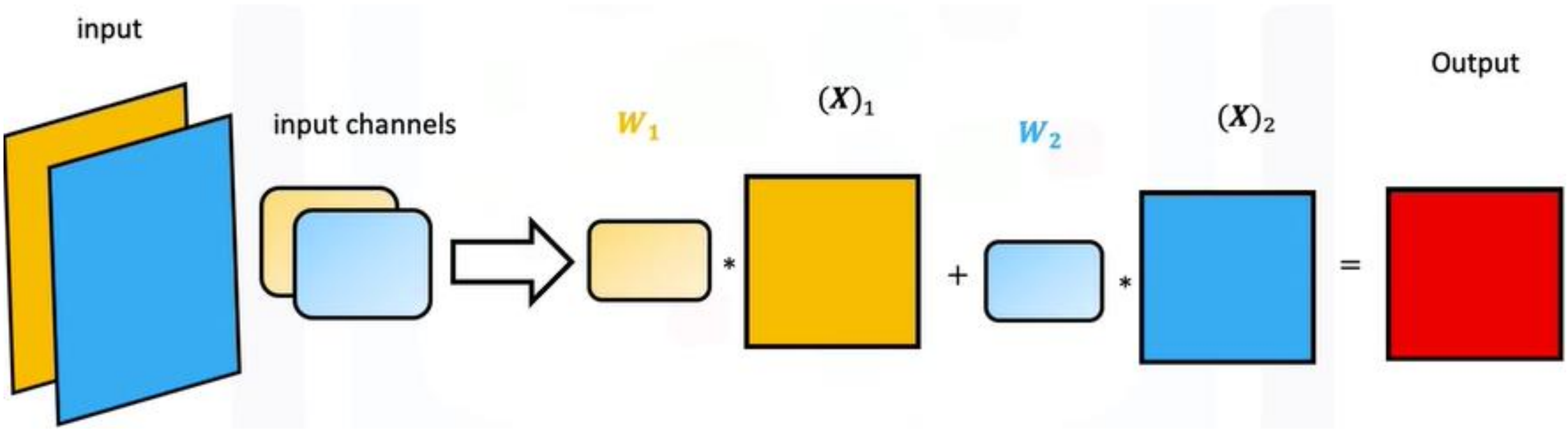
Convolution of 2 Filters on RGB Image w. 3 Channels



- 6x6 input feature map w. 3 channels; 2 3x3 filters with depth 3; 4x4 output feature map w. 2 channels
- # channels of input feature map == # depth of each filter (3)
- # channels of output feature map == # filters (2)

Convolution Example 1

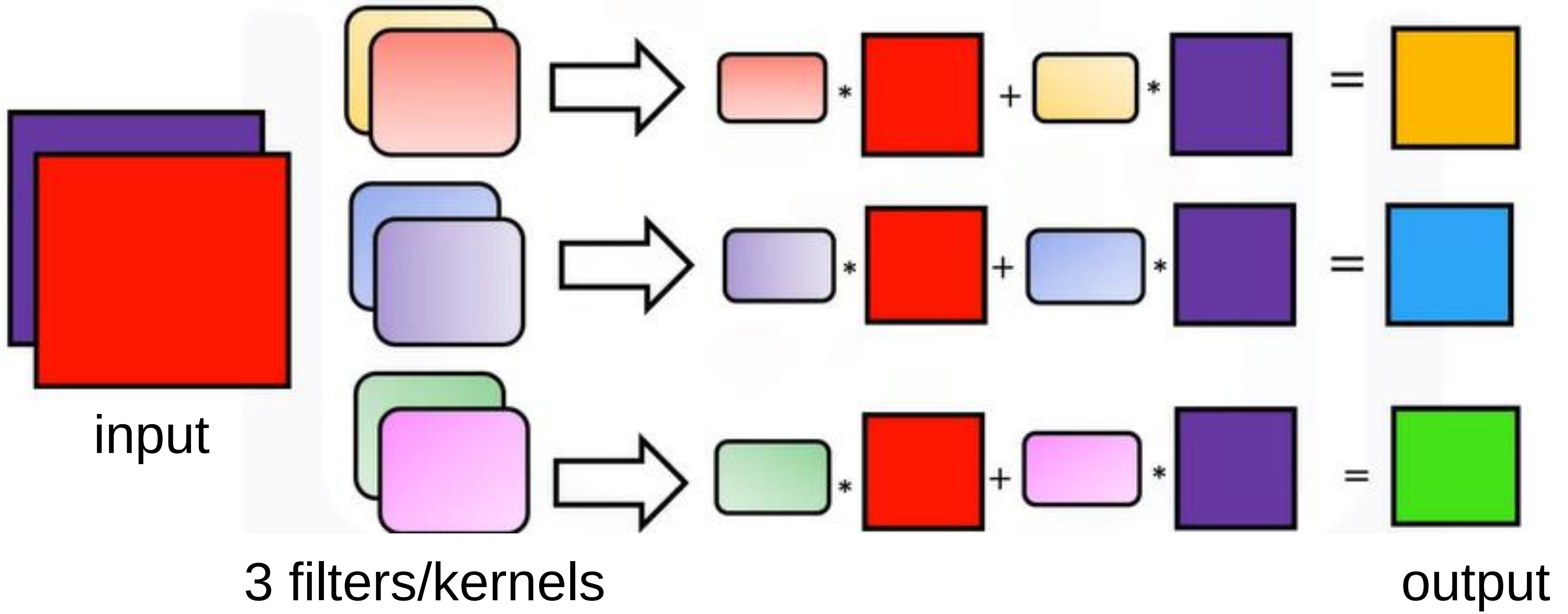
- `conv=nn.Conv2d(in_channels=2, out_channels=1, kernel_size=3)`
  - Pytorch code for a CONV layer with an input image with 2 channels (in_channels=2), 1 3×3 filter (with depth 2), 1 output activation maps (out_channels=1).
  - (The biases are assumed to be 0)



input 1 filter/kernel output

Convolution Example 2

- `conv4=nn.Conv2d(in_channels=2, out_channels=3, kernel_size=3)`
  - Pytorch code for a CONV layer with an input image with 2 channels (in_channels=2), 3 3×3 filters (with depth 2), 3 output activation maps (out_channels=3)
  - (The biases are assumed to be 0)



Convolution Example 2: Filters and Input Image

conv4.state_dict()['weight'][0][0]

$W_{0,0}$

0	0	0
0	0.5	0
0	0	0

conv4.state_dict()['weight'][0][1]

$W_{0,1}$

0	0	0
0	0.5	0
0	0	0

conv4.state_dict()['weight'][2][0]

$W_{2,0}$

1	0	-1
1	0	-2
1	0	-1

conv4.state_dict()['weight'][2][1]

$W_{2,1}$

1	2	-1
0	0	0
-1	-2	-1

conv4.state_dict()['weight'][1][0]

$W_{1,0}$

0	0	0
0	1	0
0	0	0

conv4.state_dict()['weight'][1][1]

$W_{1,1}$

0	0	0
0	-1	0
0	0	0

3 3×3 filters

Image4[1,0,:,:]

Channel 1

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Image4[1,1,:,:]

Channel 2

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

input image with 2 channels

Convolution Example 2: Output

- Each of the 3 filters convolved with the input image generates an output activation map.
- The output volume consists of 3 3×3 activation maps, with volume $3 \times 3 \times 3$

$$(\boldsymbol{Z})_0 = \overset{\boldsymbol{W}_{0,0}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}} * \overset{(\boldsymbol{X})_0}{\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}} + \overset{\boldsymbol{W}_{0,1}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}} * \overset{(\boldsymbol{X})_1}{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}$$

$$= \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix}$$

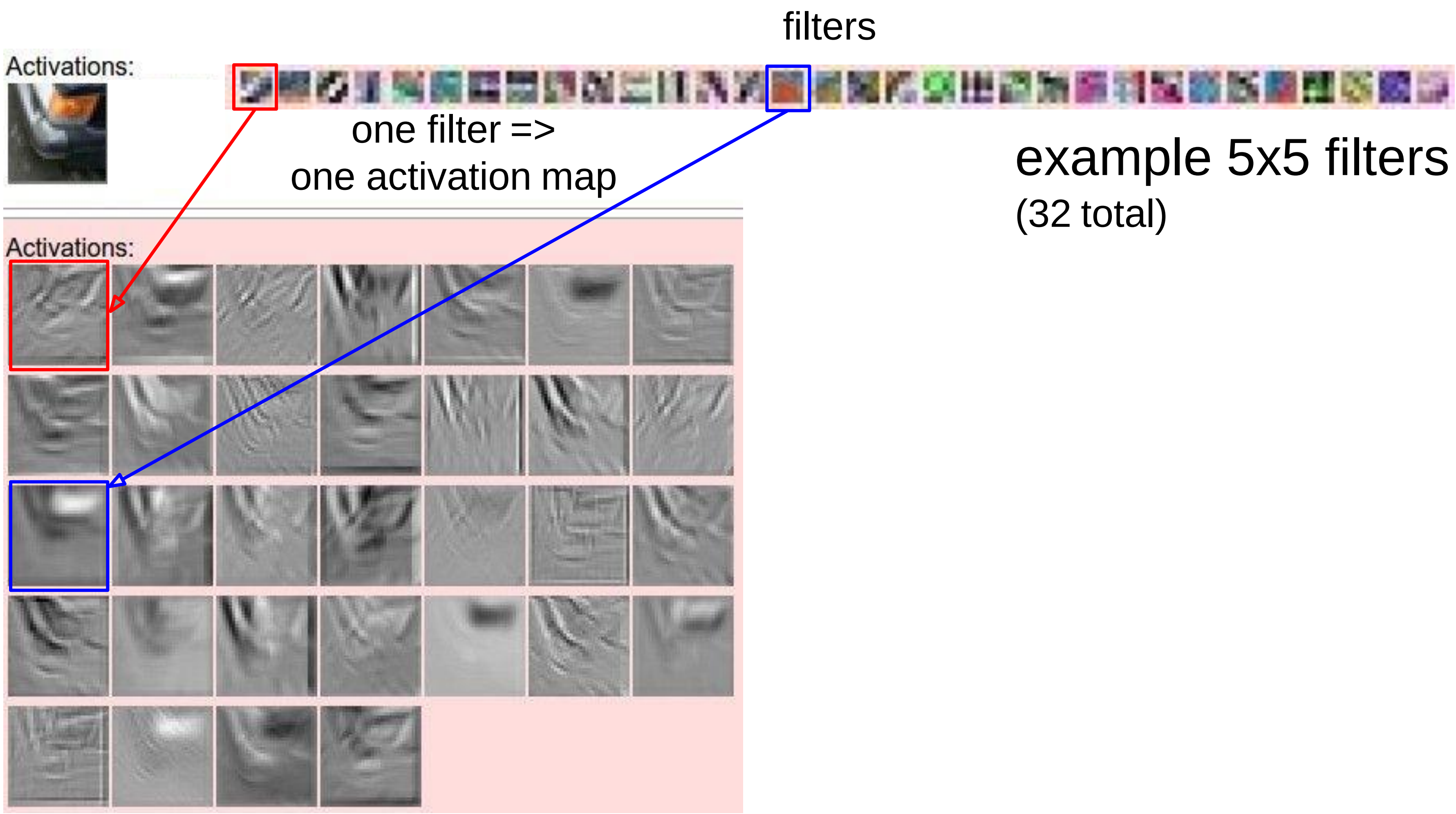
$$(\boldsymbol{Z})_1 = \overset{\boldsymbol{W}_{1,0}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}} * \overset{(\boldsymbol{X})_0}{\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}} + \overset{\boldsymbol{W}_{1,1}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}} * \overset{(\boldsymbol{X})_1}{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}$$

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

$$(\boldsymbol{Z})_2 = \overset{\boldsymbol{W}_{2,0}}{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}} * \overset{(\boldsymbol{X})_0}{\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}} + \overset{\boldsymbol{W}_{2,1}}{\begin{bmatrix} 1 & 2 & -1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}} * \overset{(\boldsymbol{X})_1}{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}$$

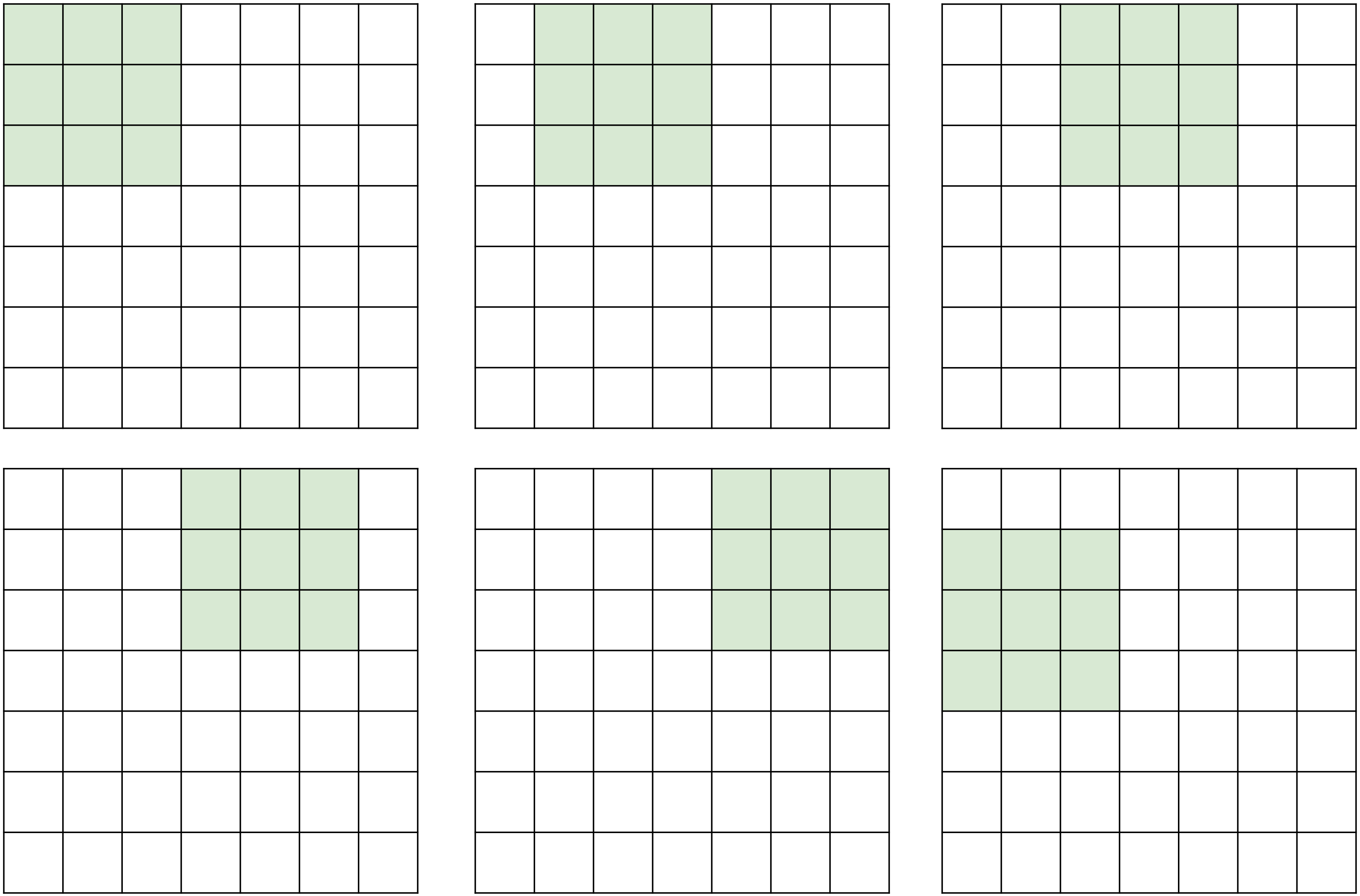
$$= \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Filters and Activation Maps Example

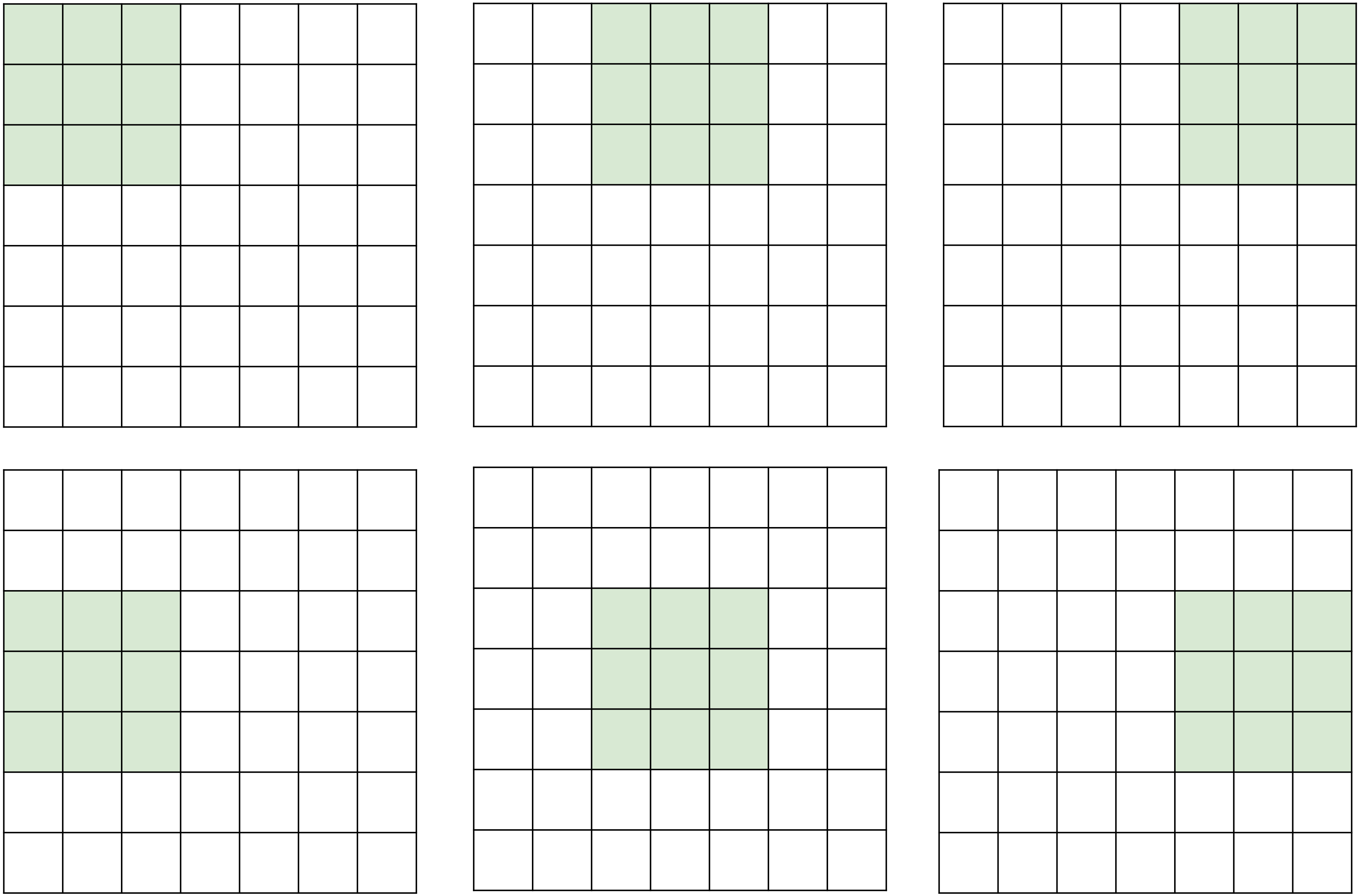


example 5x5 filters
(32 total)

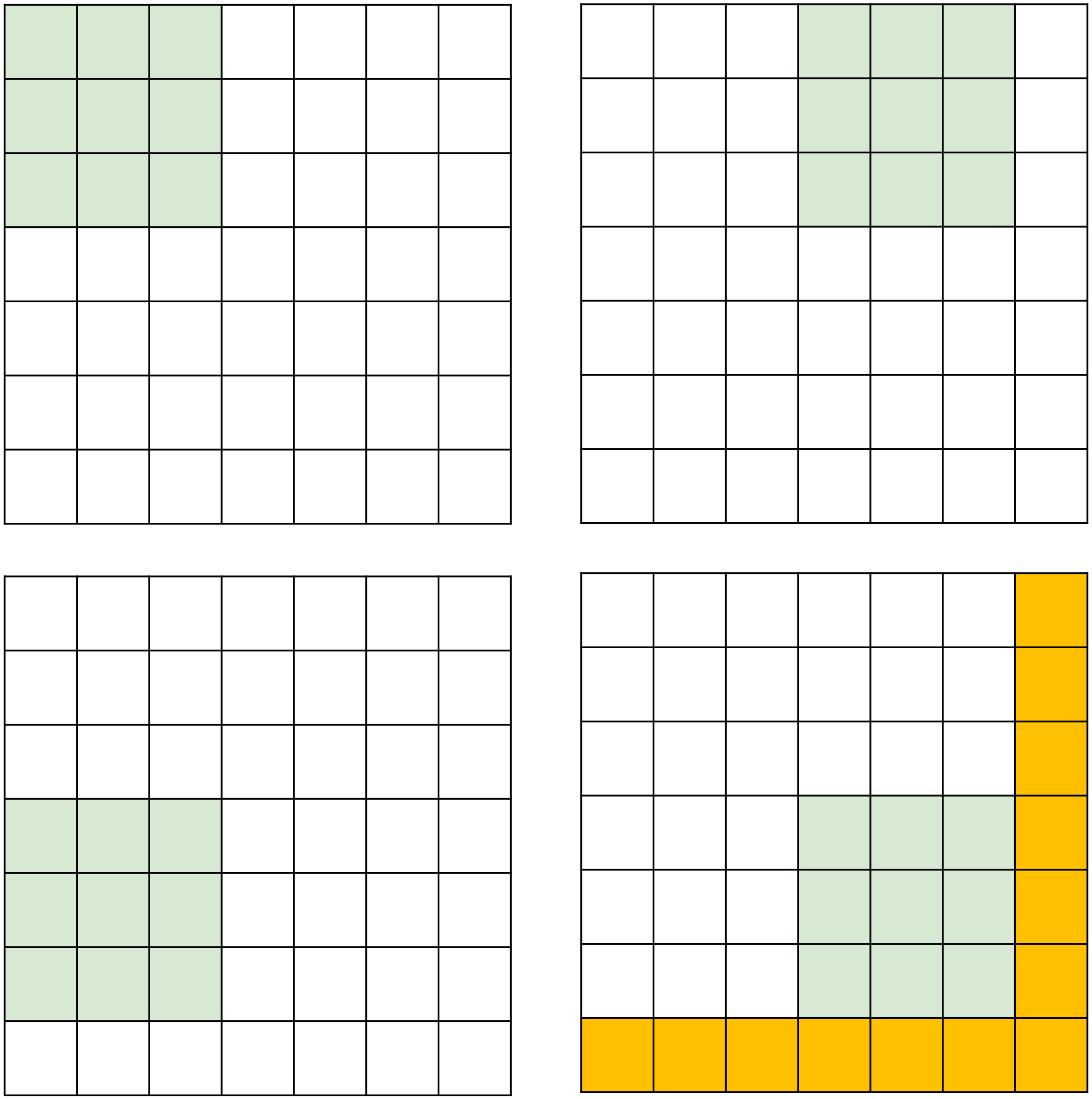
7x7 input, 3x3 filter, stride=1 ⇒ output: 5x5 filter



7x7 input, 3x3 filter, stride=2 ⇒ output: 3x3 filter



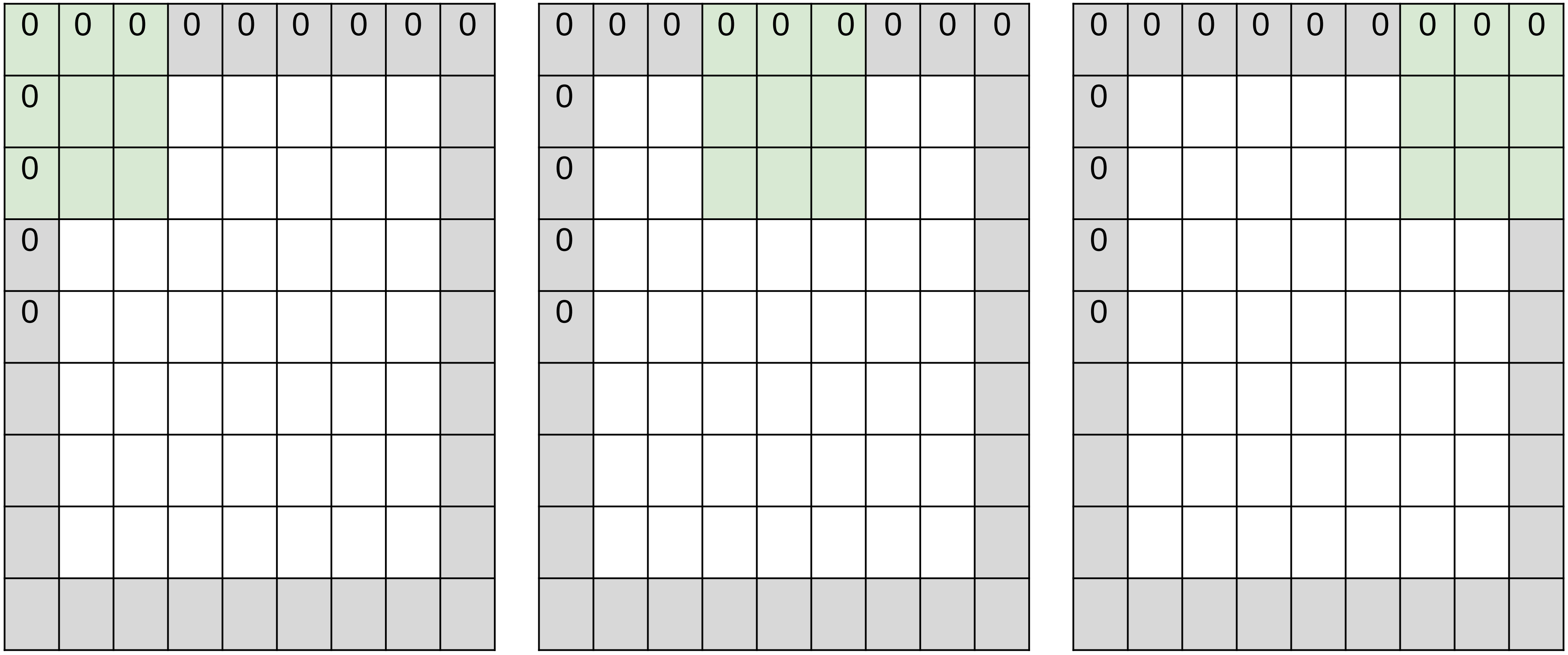
7x7 input, 3x3 filter, stride=3 ⇒ output: ???



The rightmost and bottom columns are not processed!

Solution: Add Padding

- 7x7 input, 3x3 filter, stride=3, zero padding w. 1 ⇒ output: 3x3 filter



Important

Computation of CONV Layer Sizes

Summary. To summarize, the Conv Layer:

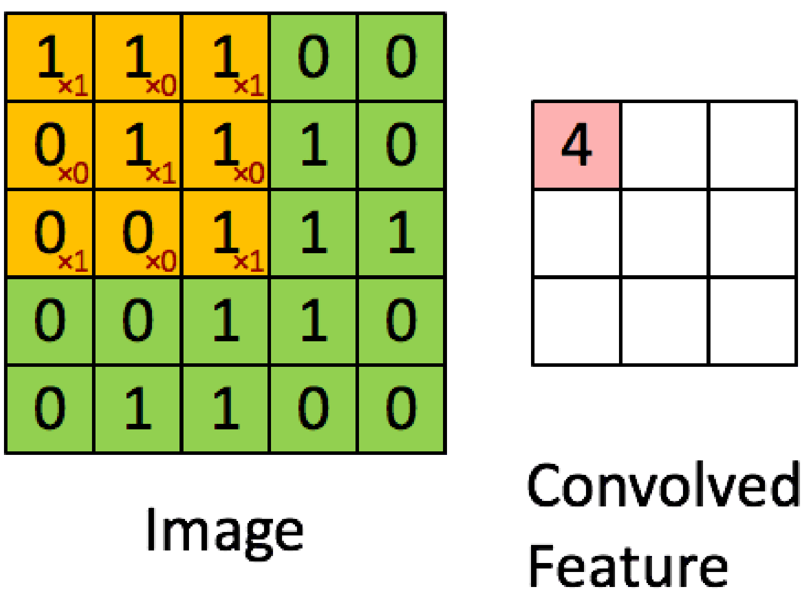
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:
K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 1, S = 1, P = 0

- If input has square shape, then we denote $N_1 = W_1 = H_1$; a filter is assumed to have square shape
- Each filter has the same depth D_1 as its input volume, and the number of filters K equals the depth D_2 of its output volume
- In practice, it is common to have stride $S = 1$, filter size $F \times F$, and zero-pad $P = \frac{1}{2}(F-1)$. Then output activation map has same spatial size as input. This is called “same padding”
  - $W_2 = \frac{1}{S}(W_1 + 2P - F) + 1 = \frac{1}{1}(W_1 + F - 1 - F) + 1 = W_1$; similarly, $H_2 = H_1$
  - e.g., $F < 3 \Rightarrow P = 0$; $F = 3 \Rightarrow P = 1$; $F = 5 \Rightarrow P = 2$

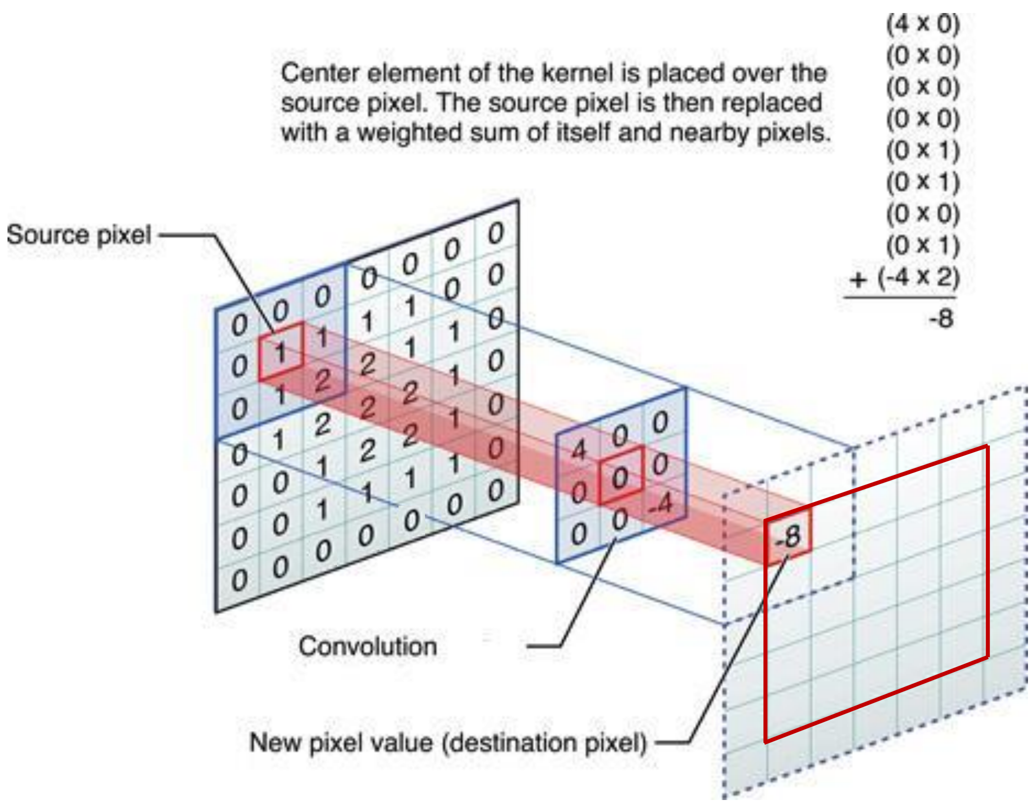
CONV Example 1: No Pad

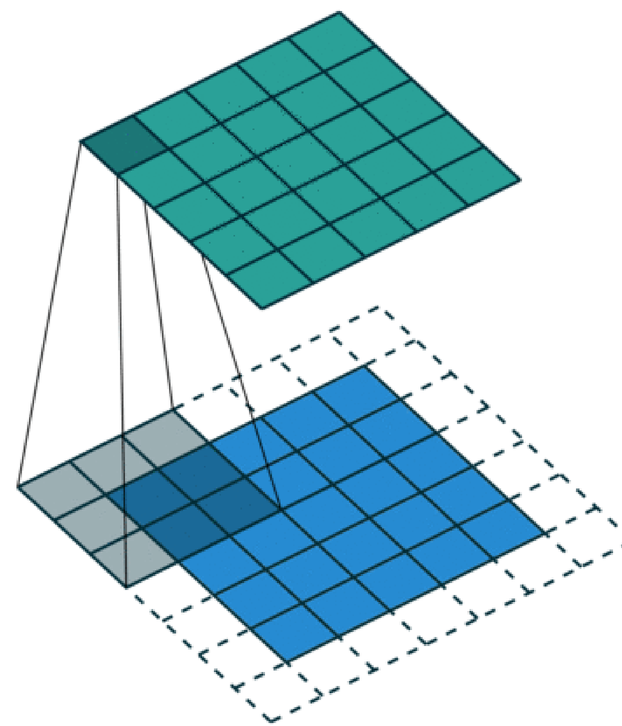
- Input volume: $5 \times 5 \times 1$ $(W_1 = H_1 = N_1 = 32, D_1 = 1)$(e.g., a greyscale image)
- A $3 \times 3 \times 1$ filter $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ $(K = 1, F = 3)$ w. stride $S = 1$, no pad
- Output activation map:
  - Spatial size: $\mathrm{W}_2 = \mathrm{H}_2 = \mathrm{N}_2 = \frac{1}{S}(N_1 + 2P - F) + 1 = \frac{1}{1}(5 - 3) + 1 = 3$
  - Depth: $D_2 = K = 1$
- Output volume: $3 \times 3 \times 1$
- Even though the fig shows sequential computation, convolution operations are inherently parallel, hence suitable for efficient implementation on parallel hardware, e.g., GPU, FPGA…



CONV Example 2: Same Padding

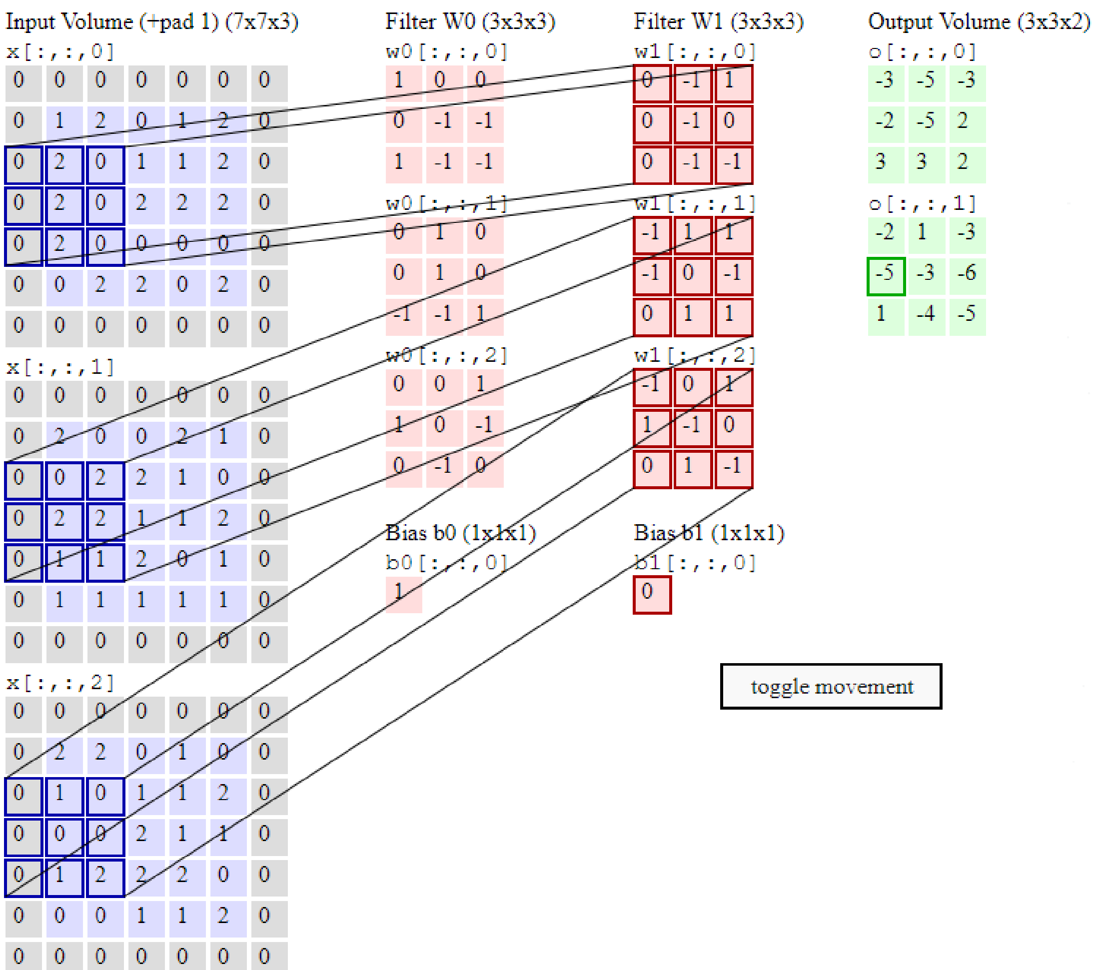
- Input volume: $5 \times 5 \times 1$
- A $3 \times 3 \times 1$ filter ($K = 1, F = 3$) w. stride $S = 1$, pad $P = 1$
- Output volume: $5 \times 5 \times 1$ (since $\frac{1}{1}(5 + 2 - 3) + 1 = 5$)
- Output activation map has the same spatial dimension as input (5×5)

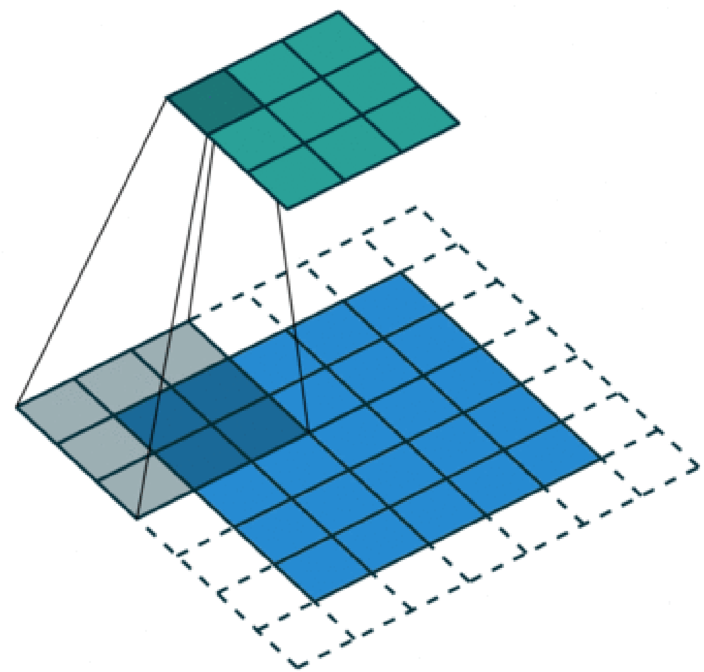




CONV Example 3: Stride $S = 2$

- Input volume: $5 \times 5 \times 3$
- 2 $3 \times 3 \times 3$ filters ($K = 2, F = 3$) w. stride $S = 2$, pad $P = 1$
- Output volumes: 2 $3 \times 3 \times 1$ (since $\frac{1}{2}(5 + 2 * 1 - 3) + 1 = 3$)
  - Animation: https://cs231n.github.io/convolutional-networks/





CONV Example 4: Input Depth $D_1 = 3$

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

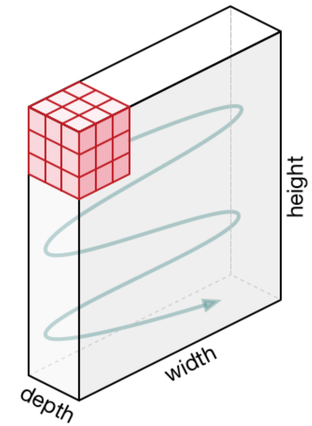
Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)



Movement of the filter

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$$308 \quad + \quad -498 \quad + \quad 164 \quad + 1 = -25$$

Bias = 1

Output

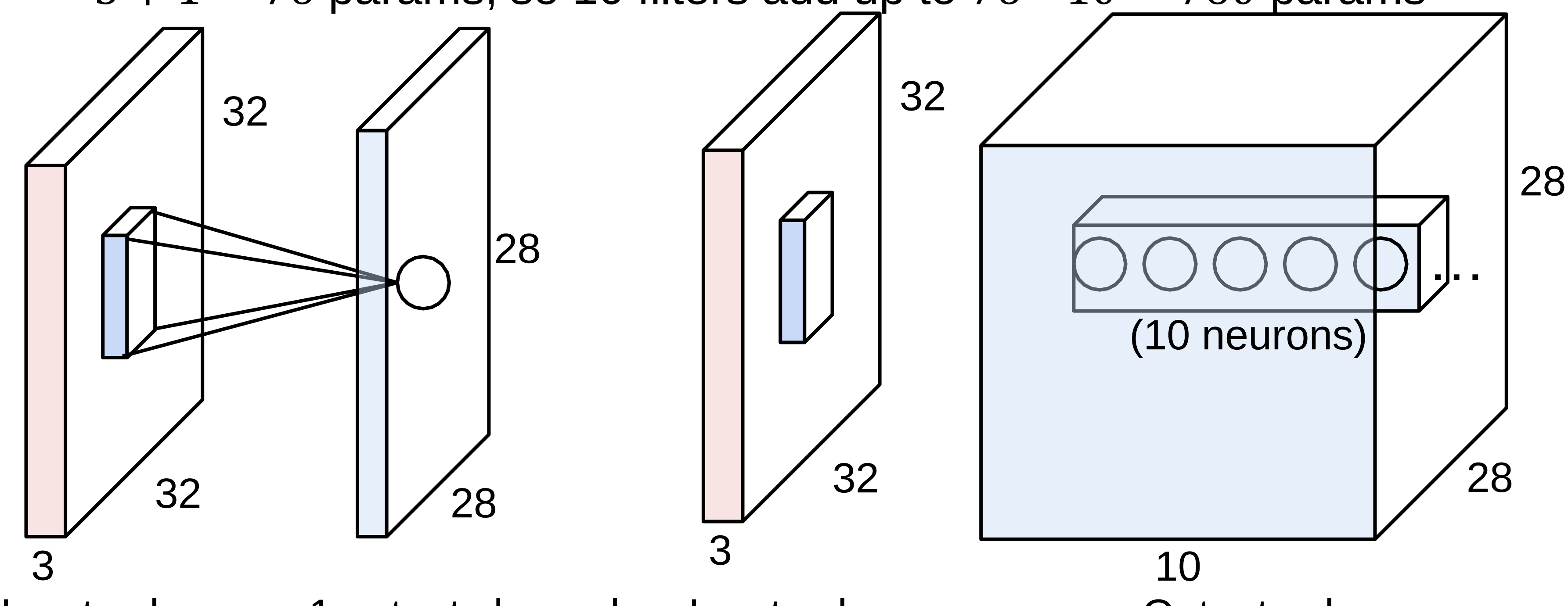
-25				...
				...
				...
				...
...	...	...	...	...

- Input volume: $M \times N \times 3$
- A $3 \times 3 \times 3$ filter ($K = 1, F = 3$) w. stride $S = 1,$ pad $P = 1$
- Output volume: $M \times N \times 1$ (since $\frac{1}{1}(M + 2 * 1 - 3) + 1 = M,$ $\frac{1}{1}(N + 2 * 1 - 3) + 1 = N$)

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

CONV Example 5: Multiple Filters $K = 10$

- Input volume: $32 \times 32 \times 3$ $(W_1 = H_1 = N_1 = 32, D_1 = 3)$
- 10 $5 \times 5 \times 3$ filters ($K = 10, F = 5$) w. stride $S = 1,$ no pad ($P = 0$)
- Each output activation map:
  - Spatial size: $\mathrm{W}_2 = \mathrm{H}_2 = \mathrm{N}_2 = \frac{1}{S}(N_1 + 2P - F) + 1 = \frac{1}{1}(32 - 5) + 1 = 28$
  - Depth: $D_2 = K = 10$
- Output volume: $28 \times 28 \times 10$
- No. params (weights and biases) in this layer: each filter has $5 * 5 * 3 + 1 = 76$ params, so 10 filters add up to $76 * 10 = 760$ params

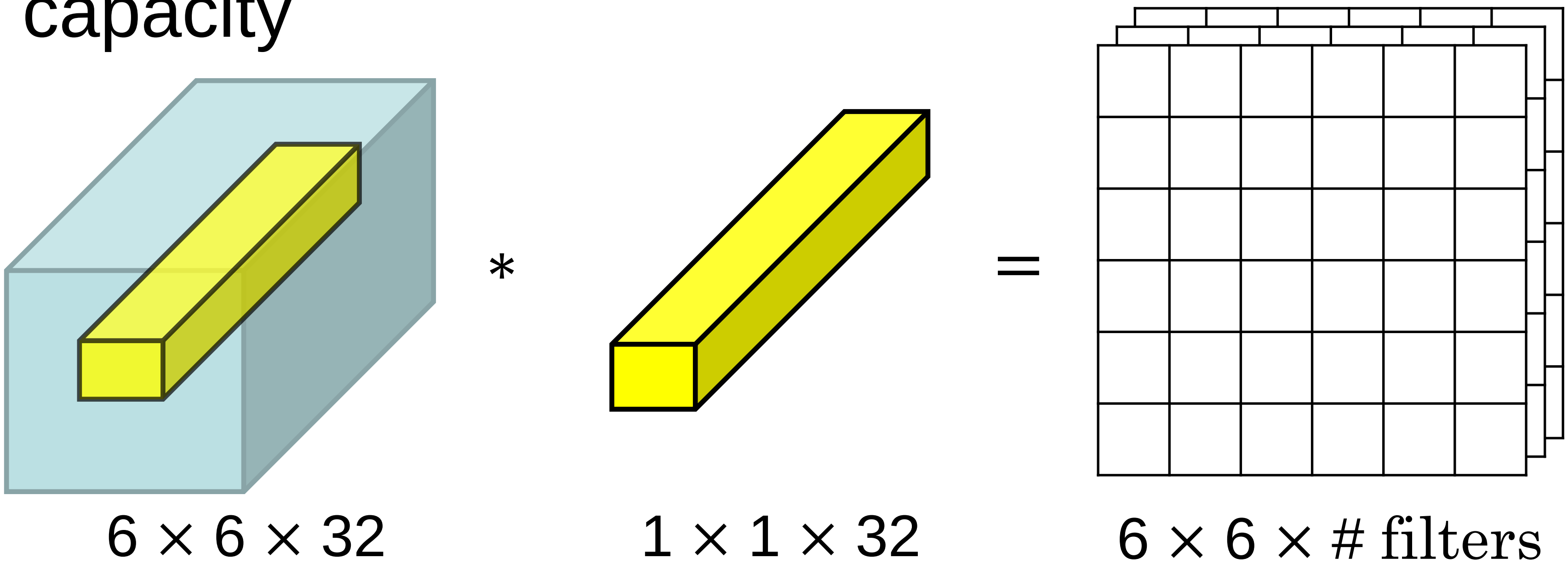


CONV Example 6: Pad $P = 2$

- Input volume: $32 \times 32 \times 3$ $(W_1 = H_1 = N_1 = 32, D_1 = 3)$
- 10 $5 \times 5 \times 3$ filters ($K = 10, F = 5$) w. stride $S = 1$, pad $P = 2$
- Each activation map:
  - Spatial size: $\mathrm{W}_2 = \mathrm{H}_2 = \mathrm{N}_2 = \frac{1}{S}(N_1 + 2P - F) + 1 = \frac{1}{1}(32 + 2 * 2 - 5) + 1 = 32$
  - Depth: $D_2 = K = 10$
- Output volume: $32 \times 32 \times 10$
- No. params: each filter has $5 * 5 * 3 + 1 = 76$ params, so 10 filters add up to $76 * 10 = 760$ params

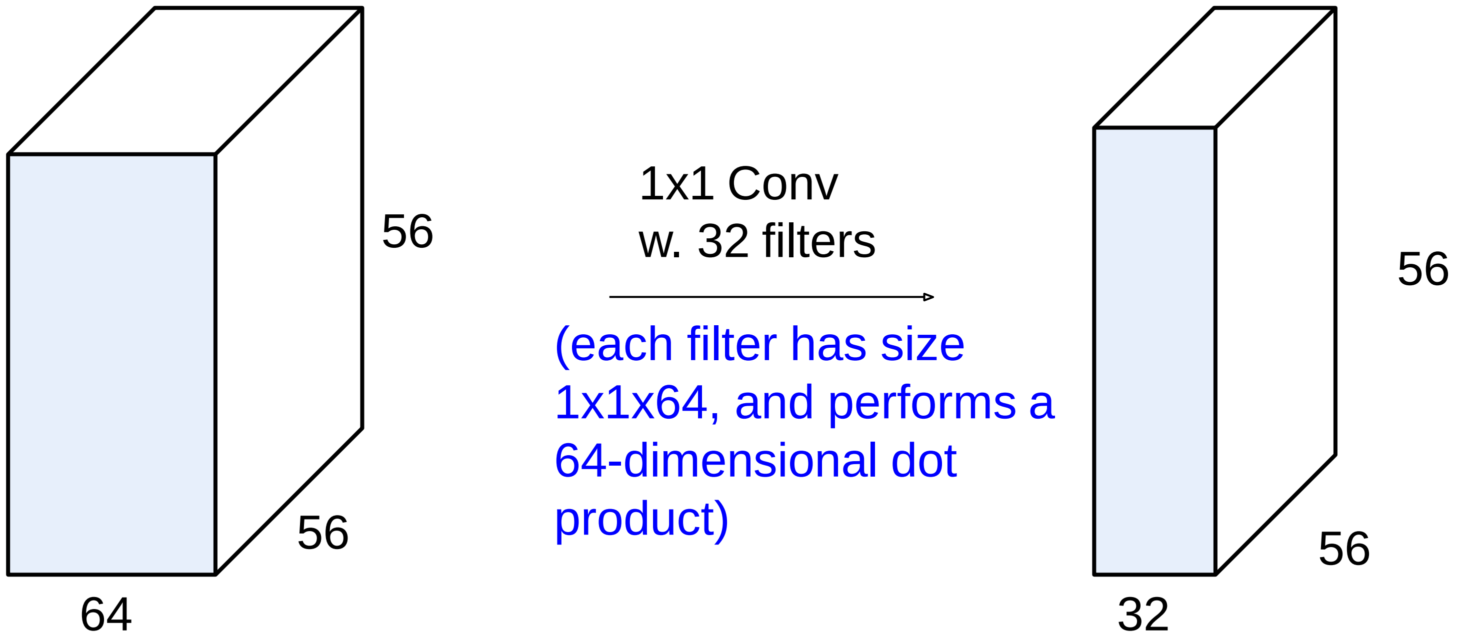
Pointwise Convolution with 1×1 Filter

- A 1×1 filter performs "mixing" of the input channels, then applies a non-linear activation function
- Can be used to reduce the number of channels (volume depth); the non-linear activation function also helps increase model capacity

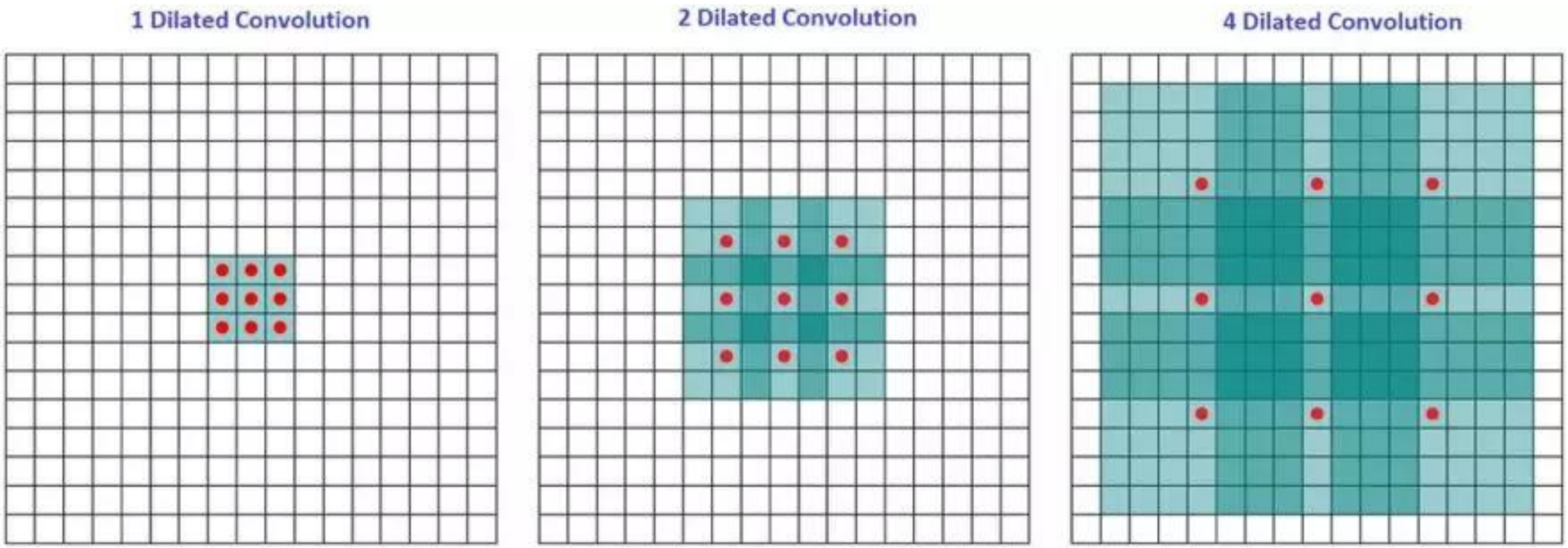


1×1 Filter Example

- Input volume: $56 \times 56 \times 64$ $(W_1 = H_1 = N_1 = 56, D_1 = 64)$
- 32 $1 \times 1 \times 64$ filters ($K = 32, F = 1$) w. stride $S = 1,$ no pad
- Each activation map:
  - Spatial size: $\mathrm{W}_2 = \mathrm{H}_2 = \mathrm{N}_2 = \frac{1}{S}(N_1 + 2P - F) + 1 = \frac{1}{1}(56 - 1) + 1 = 56$
  - Depth: $D_2 = K = 32$
- Output volume: $56 \times 56 \times 32$
- No. params: each filter has $1 * 1 * 64 + 1 = 65$ params, so 32 filters add up to $65 * 32 = 2080$ params

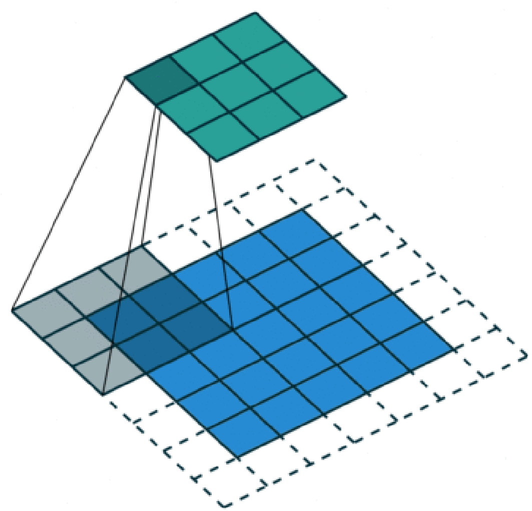


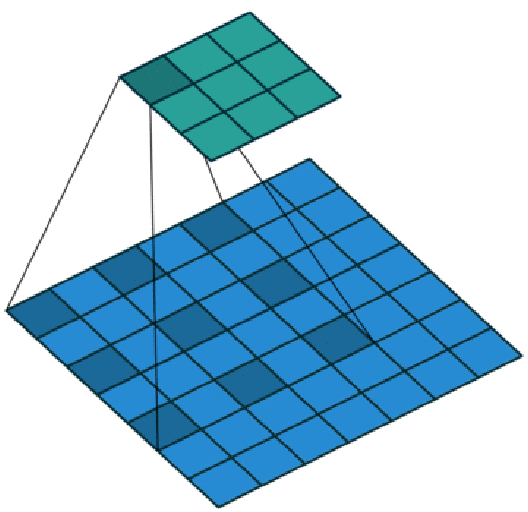
Dilated Convolution



(a)

- Insert 0s between input elements to increase receptive field size without increasing # params



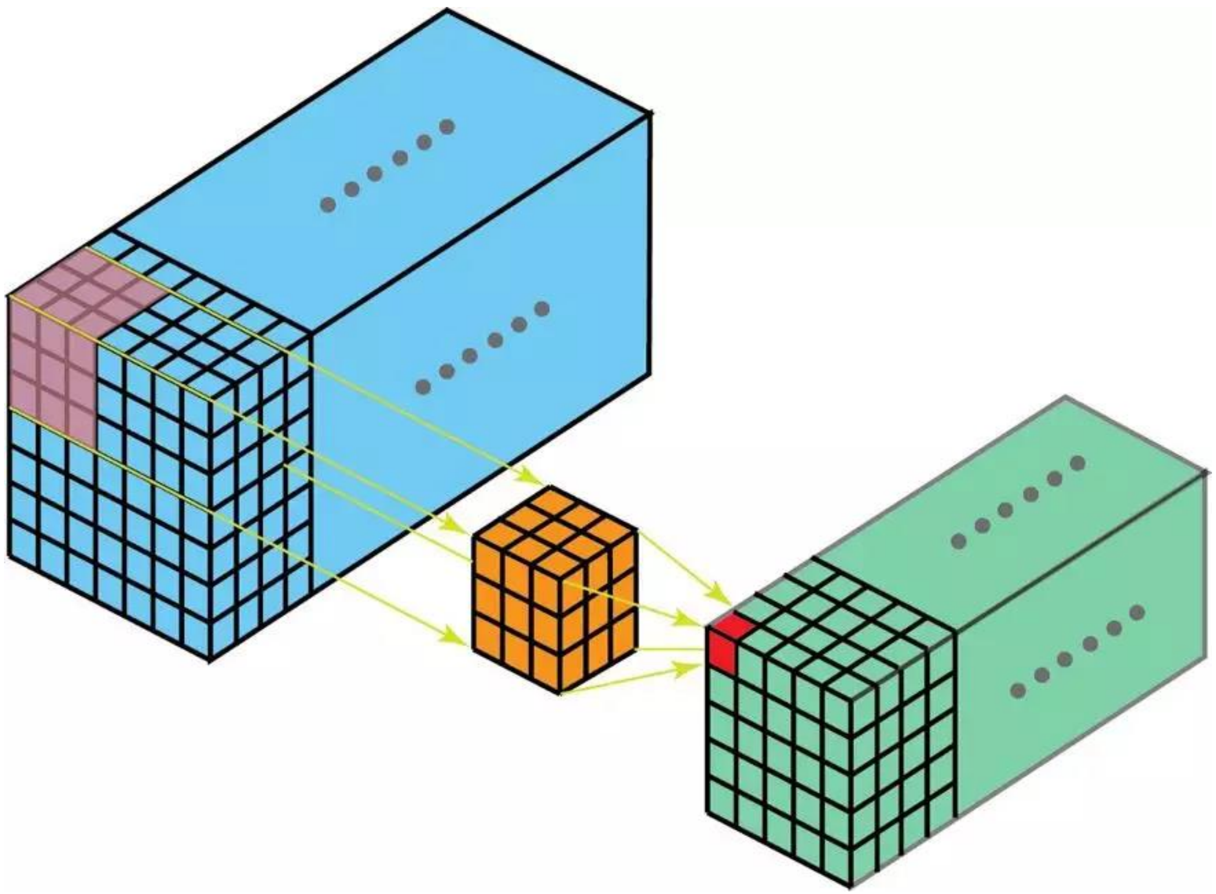


Regular convolution (1-dilated)

2-dilated convolution

3D Convolution

- 3D filter slides along all 3 axes (width, height, depth). Very computation intensive
- Useful for 3D images such as medical CT/MRI images, or Point Clouds from Lidar



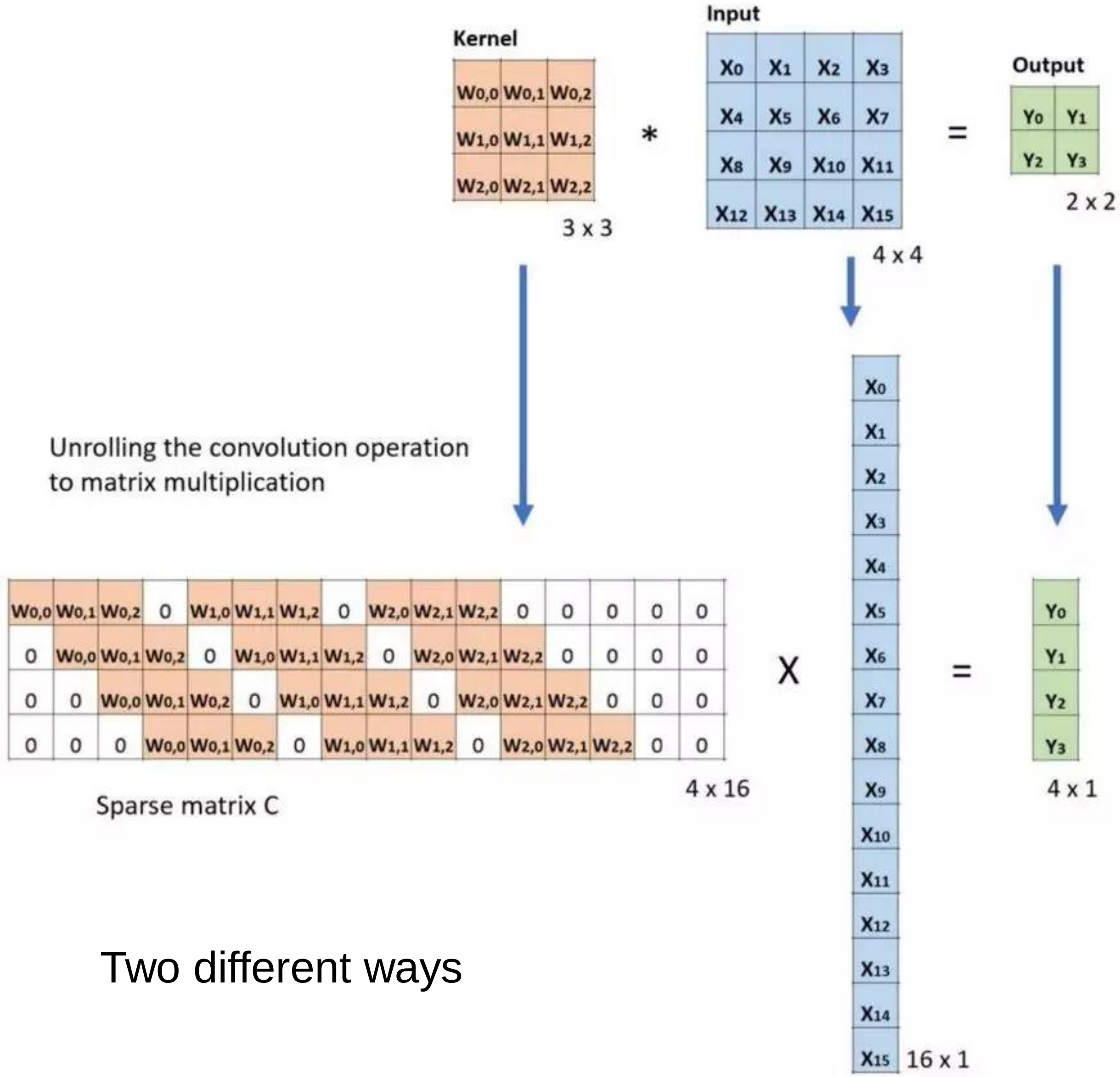
Converting Convolution to Matrix Multiplication: 1D CONV Example

- Since parallel hardware (GPU, FPGA…) can handle matrix multiplication efficiently, this conversion increases computation efficiency at the expense of increased memory size for storing the weights (the biases are not shown in fig)

$$\begin{array}{ccccc} \text{Kernel} & & \text{Input} & & \text{Output} \\ \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} & * & \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} & = & \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \end{array}$$

$$\begin{bmatrix} w_0 & w_1 & w_2 & 0 & 0 \\ 0 & w_0 & w_1 & w_2 & 0 \\ 0 & 0 & w_0 & w_1 & w_2 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Converting Convolution to Matrix Multiplication: 2D CONV Example



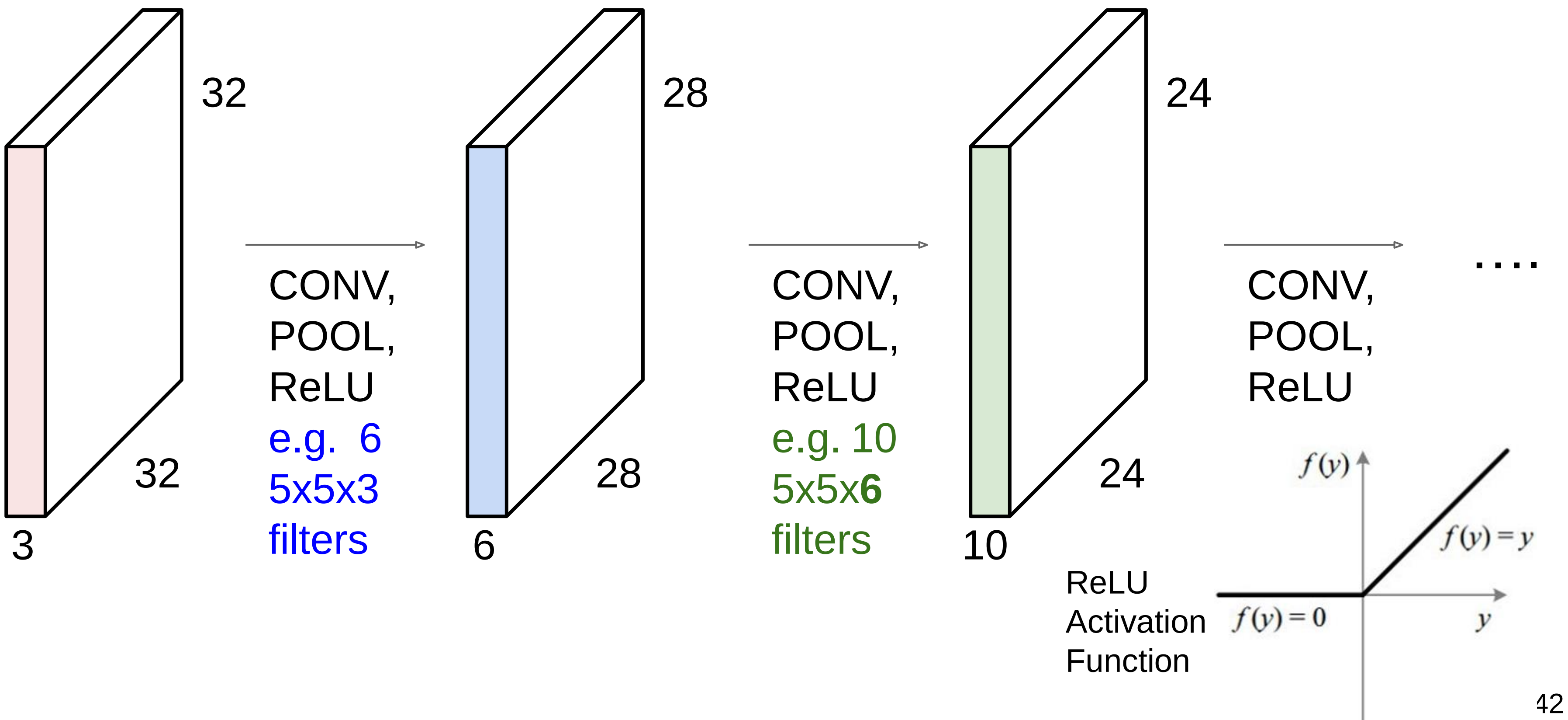
Two different ways

Outline

- CNN Convolution layers
- Pooling and Fully-Connected layers
- CNN Case Studies

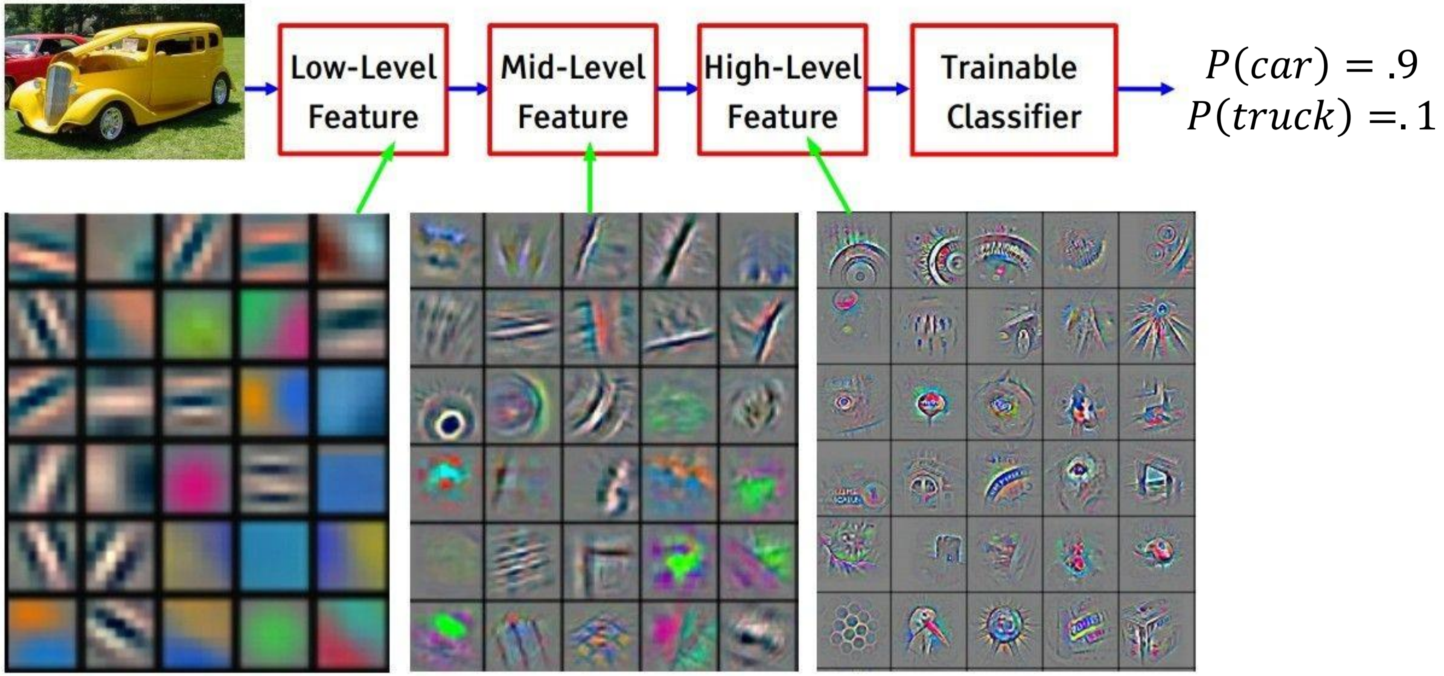
Typical CNN Architecture

- Multiple layers, each consisting of CONV, POOL and non-linear activation functions (e.g., ReLU), are stacked into a deep network
  - Many variants possible, e.g., multiple CONV layers can be stacked without POOL and activation functions in-between



Feature Hierarchy

- Multiple hidden layers extract a hierarchy of increasingly-abstract features layer-by-layer, until the last layer produces a classification result



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Important

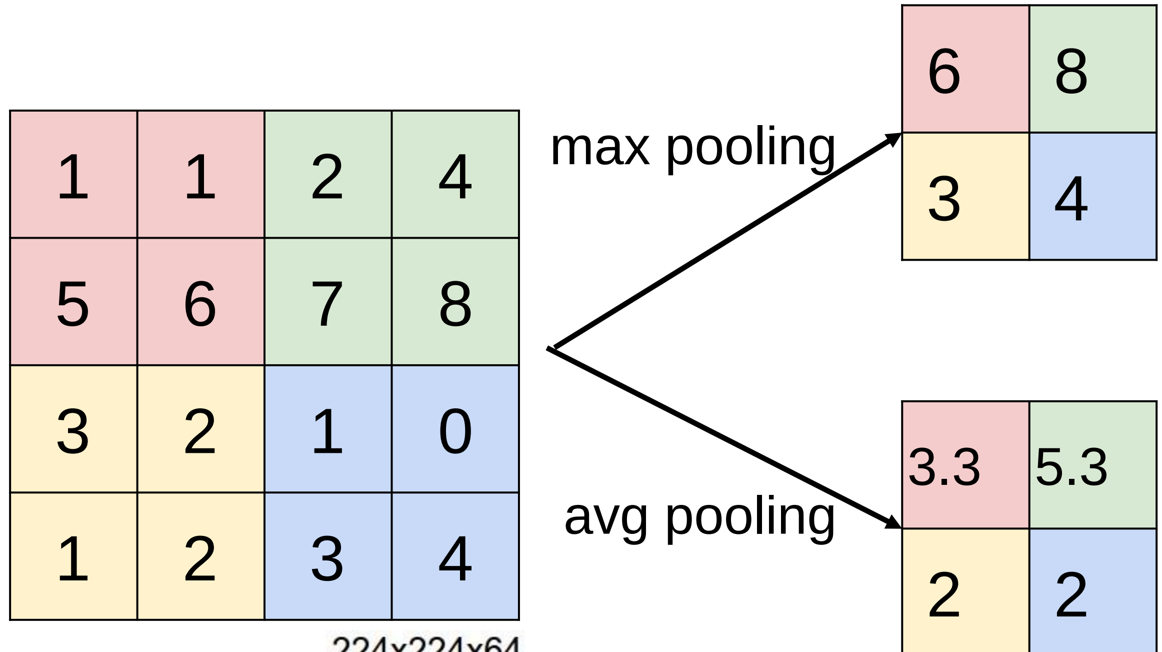
Pooling (Sub-Sampling) Layer

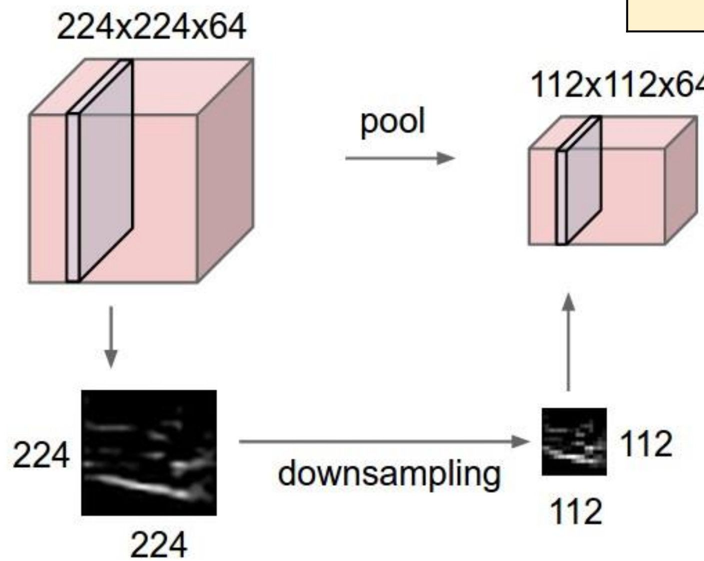
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.

- A pooling filter has depth 1, and operates over each activation map independently, hence the input volume and output volume have the same depth $D_1 = D_2$
  - In contrast, a CONV filter has the same depth D_1 as its input volume, and the number of filters K equals the depth D_2 of its output volume
  - Common settings: $F = 2, S = 2$, or $F = 3, S = 2$
- Example: pooling w. a 2×2 filter w. stride $S = 2$, no pad
- Output volume: $\frac{W_1}{2} \times \frac{H_1}{2} \times \mathrm{D}_1$(since $\frac{1}{2}(W_1 - 2) + 1 = \frac{W_1}{2}, \frac{1}{2}(H_1 - 2) + 1 = \frac{H_1}{2}$)

Max Pooling w. Examples

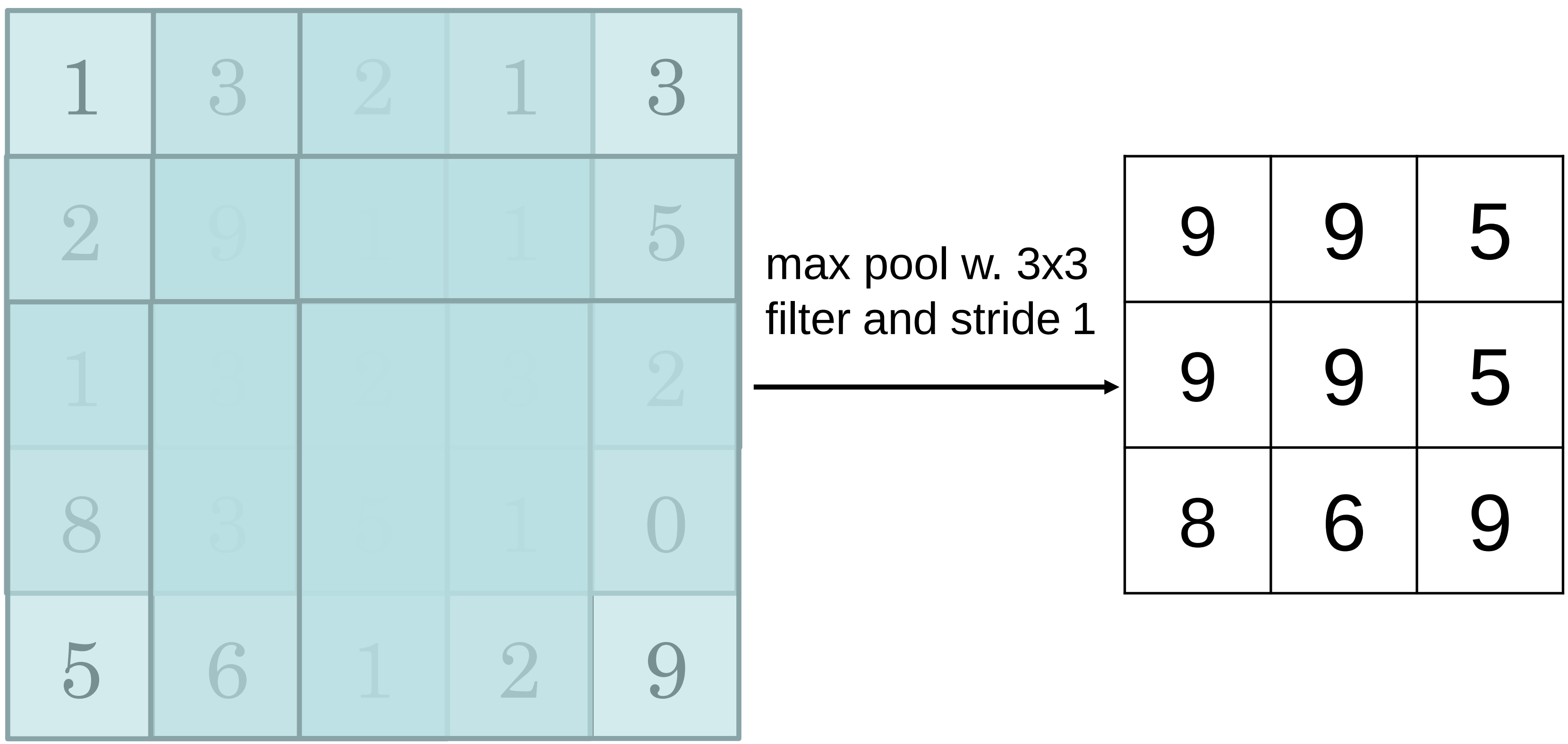
- Max pooling: take the max element among the $F * F$ elements in each $F \times F$ patch of each input activation map to reduce its dimension ($F = 2, S = 2$)
- Alternative: average pooling is less commonly used
- Pooling is also called subsampling or downsampling
  - Max pooling selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image. Average pooling method smooths out the image and hence the sharp features may not be identified when this pooling method is used.





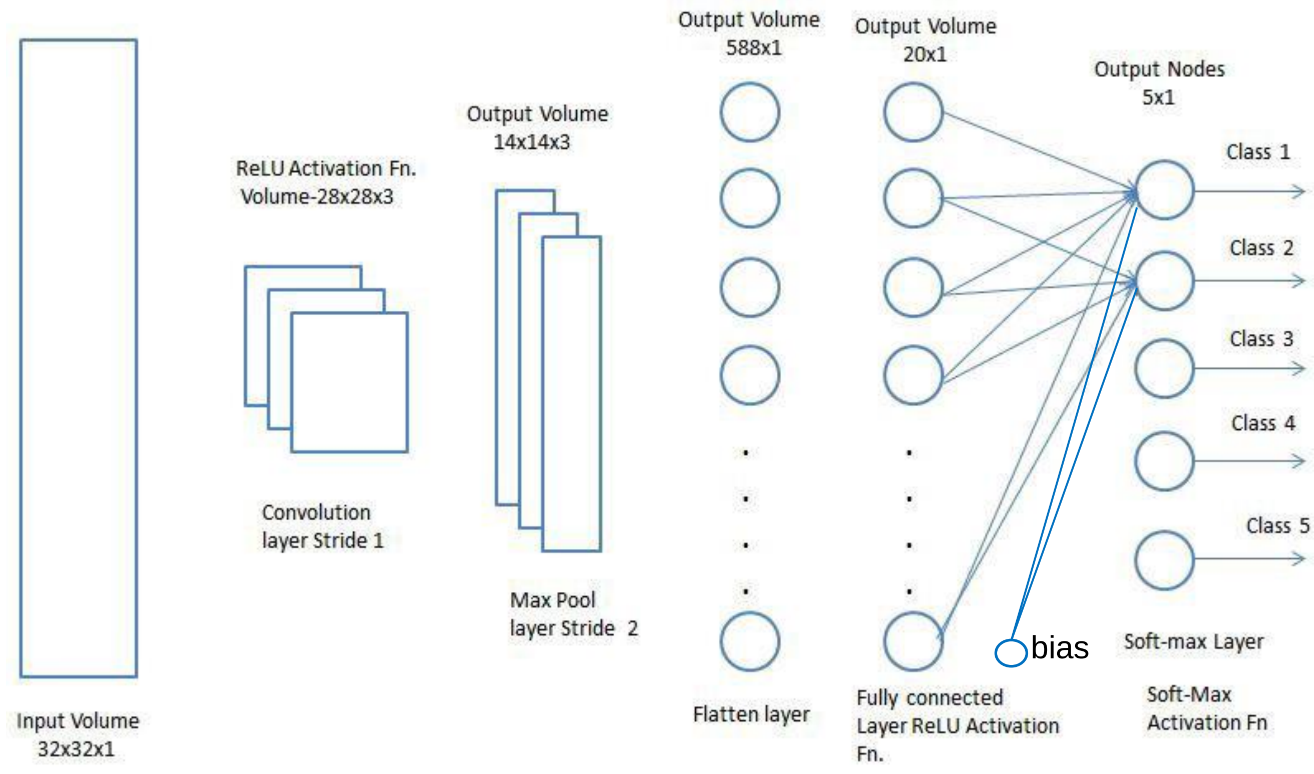
Overlapping Pooling

- Input volume: $N \times N \times \mathrm{D}_1$
- A 3×3 pooling filter w. stride $S = 1,$ no pad
- Output volume: $(N-2) \times (N-2) \times \mathrm{D}_1$(since $\frac{1}{1}(N-3)+1 = N-2$)
  – In practice, it is more common to have $F = 3, S = 2$ for overlapping pooling



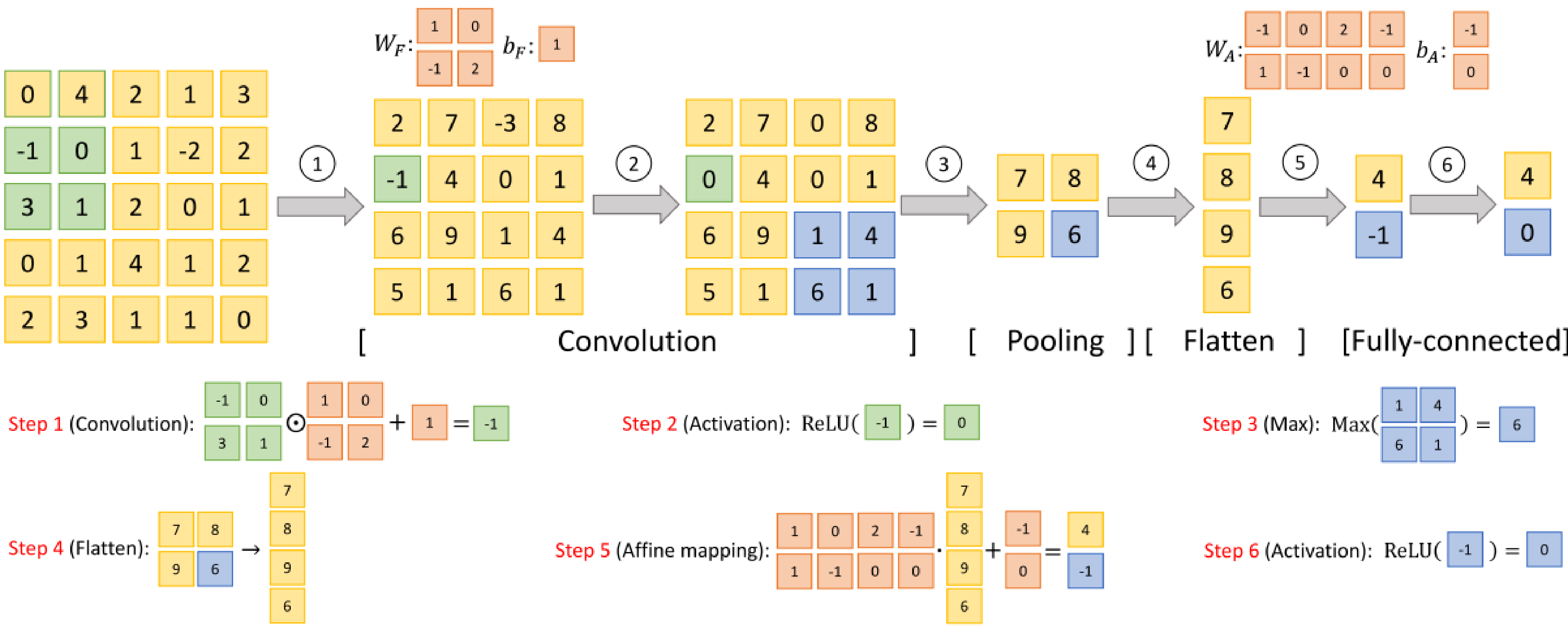
FC Layer

- Contains neurons that connect to the entire input volume w. no weight sharing
  - No. params for FC layer of size N_{out} connected to input layer of size N_{in} is $(N_{in} + 1) * N_{out}$



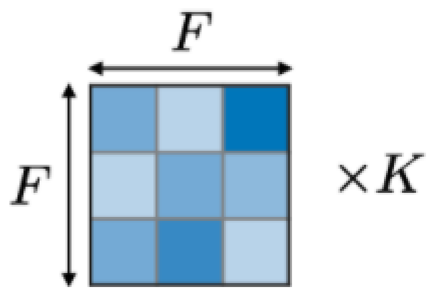
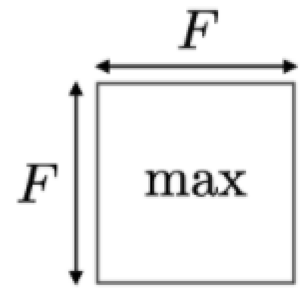
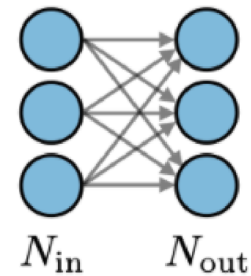
CNN Toy Example

- A CNN with 1 CONV layer and 1 FC layer



Huang C, Fan J, Chen X, et al. Divide and slide: Layer-wise refinement for output range analysis of deep neural networks[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(11): 3323-3335.

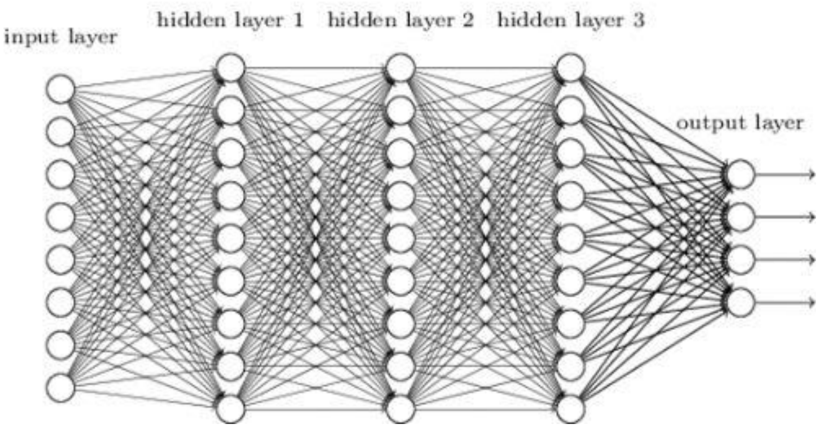
Summary of 3 Types of CNN Layers

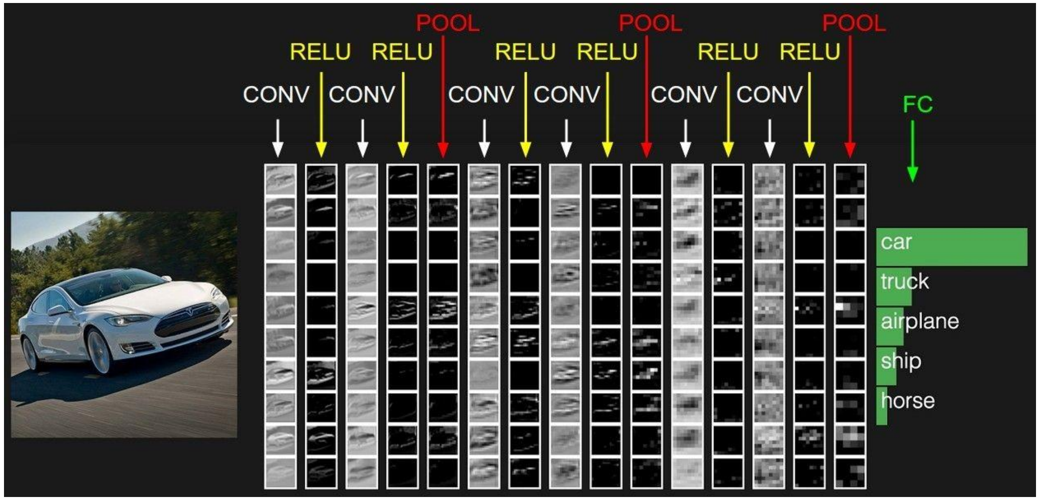
	CONV	POOL	FC
			
Input volume	$W_1 \times H_1 \times D_1$	$W_1 \times H_1 \times D_1$	N_{in}
Output volume	$W_2 \times H_2 \times K$	$W_2 \times H_2 \times K$	N_{out}
No. params	$(F * F * D_1 + 1) * K$	0	$(N_{in} + 1) * N_{out}$
No. MULs	$(F * F * D_1 + 1) * K * W_2 * H_2$	0	$(N_{in} + 1) * N_{out}$

(1) No. MULs for CONV layer: $(F * F * D_1 + 1)$ MULs to compute each output element; $K * W_2 * H_2$ output elements
(2) The bias term $+1$ is often omitted

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

Fully-Connected NN vs. CNN





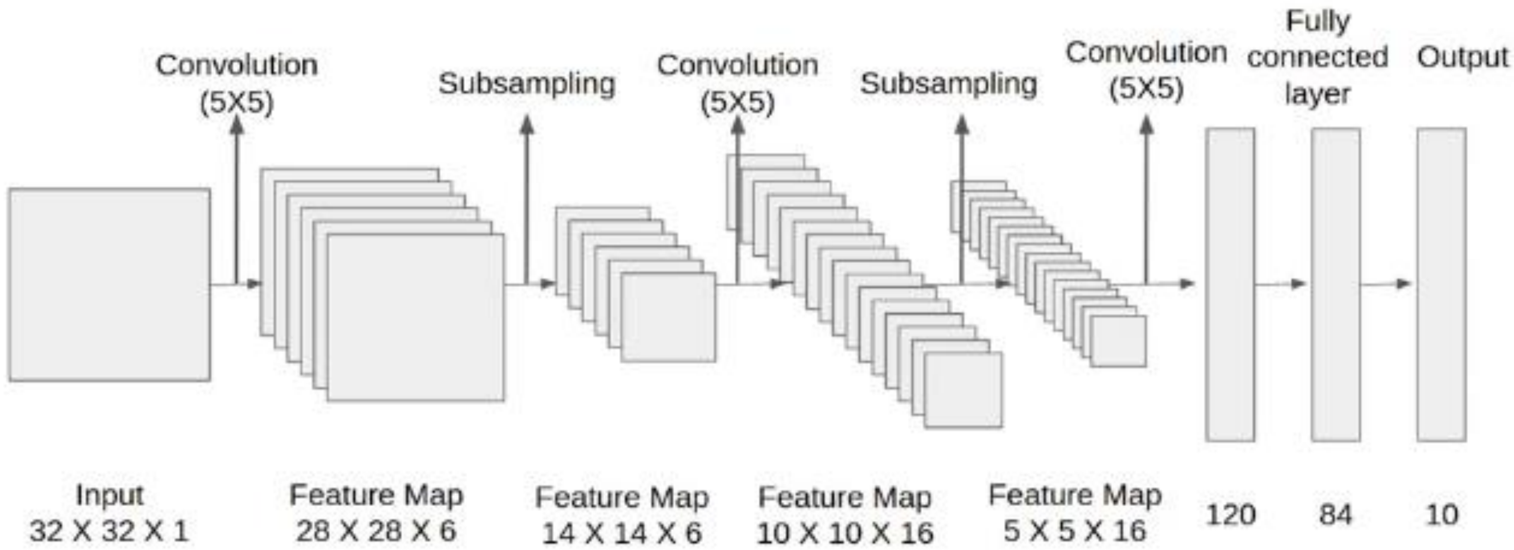
- In a FCNN, all layers are Fully-Connected
- Cannot alter input image size
- No translation invariance
- No. params can grow very large, prune to overfitting

- In a CNN, only the last few (typically <=3) layer(s) are FC
- CONV layers can handle images of arbitrary size
- Translation invariance
- Fewer params than MLP

Outline

- CNN Convolution layers
- Pooling and Fully-Connected layers
- CNN Case Studies

LeNet-5



Layer	Input $W_1 \times H_1 \times D_1$	No. Filters	Filter $K \times K \times D/S$	Output $W_2 \times H_2 \times D_2$	No. params
C1:CONV	$32 \times 32 \times 1$	6	$5 \times 5 \times 1$	$28 \times 28 \times 6$	156
S2:POOL	$28 \times 28 \times 6$	6	$2 \times 2 \times 1/2$	$14 \times 14 \times 6$	0
C3:CONV	$14 \times 14 \times 6$	16	$5 \times 5 \times 6$	$10 \times 10 \times 16$	2416
S4:POOL	$10 \times 10 \times 16$	16	$2 \times 2 \times 1/2$	$5 \times 5 \times 16$	0
C5:CONV	$5 \times 5 \times 16$	120	$5 \times 5 \times 16$	$1 \times 1 \times 120$	48120
F6	FC	-	$-$	84	10164
Output	FC			10	850

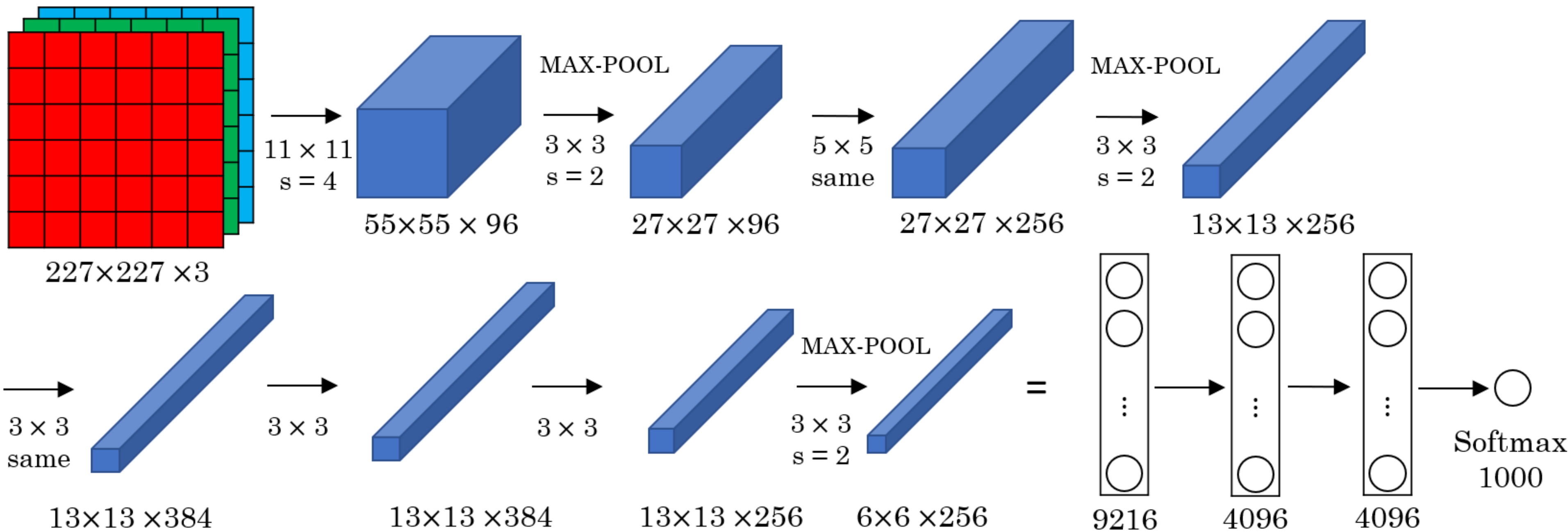
LeNet-5 Details

- Input image: $32 \times 32 \times 1$ (grey-scale images of hand-written digits w. size 32×32 pixels)
- Conv filters $5 \times 5 \times 1$ w. stride 1; Pooling filters 2×2 w. stride 2
- Conv layer C1 maps from input volume $32 \times 32 \times 1$ to 6 feature maps w. volume $28 \times 28 \times 6$ (since $\frac{1}{1}(32 - 5) + 1 = 28$). No params: $(5 * 5 * 1 + 1) * 6 = 156$
- Pooling layer S2 maps from input volume $28 \times 28 \times 6$ to 6 feature maps w. volume $14 \times 14 \times 6$ (since $\frac{1}{2}(28 - 2) + 1 = 14$).
- Conv layer C3 maps from input volume $14 \times 14 \times 6$ to 16 feature maps w. volume $10 \times 10 \times 16$ (since $\frac{1}{1}(14 - 5) + 1 = 10$). No params: $(5 * 5 * 6 + 1) * 16 = 2416$
- Pooling layer S4 maps from input volume $10 \times 10 \times 16$ to 16 feature maps w. volume $5 \times 5 \times 16$ (since $\frac{1}{2}(10 - 2) + 1 = 5$)
- Conv layer C5 maps from input volume $5 \times 5 \times 16$ to 120 feature maps w. volume $1 \times 1 \times 120$ (since $\frac{1}{1}(5 - 5) + 1 = 1$). No params: $(5 * 5 * 16 + 1) * 120 = 48120$
  - You can also view it as an equivalent Fully-Connected layer that maps from the flattened input of size 400×1 ($5 * 5 * 16 = 400$) to output of size 120×1. For details, refer to L4.2 “Turning FC layer into CONV Layers”
- FC layer F6 maps from input of size 120×1 to output of size 84×1. No params: $(120 + 1) * 84 = 10164$
- Output layer (SoftMax) maps from input of size 84×1 to output of size 10. No params: $(84 + 1) * 10 = 850$

https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/
https://towardsdatascience.com/understanding-lenet-a-detailed-walkthrough-17833d4bd155

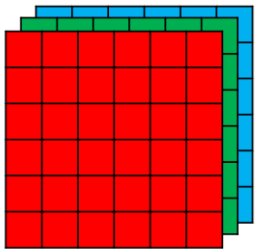
AlexNet [Krizhevsky et al. 2012]

- Input image: $227 \times 227 \times 3$
- 1st layer (CONV1): 96 11×11 filters w. stride $S = 4$, w. ReLU activation function
- Output volume: $55 \times 55 \times 96$ (since $\frac{1}{4}(227 - 11) + 1 = 55$).
- 2nd layer (POOL1): 3×3 filters w. stride $S = 2$ (overlapping)
- Output volume: $27 \times 27 \times 96$ (since $\frac{1}{2}(55 - 3) + 1 = 27$)
- $\dots$
- Total No. params: 60M
- Introduced ReLU activation function



VGGNet [Simonyan 2014] (the best performing variant VGG-16)

CONV = 3×3 filter, s = 1, same MAX-POOL = 2×2 , s = 2

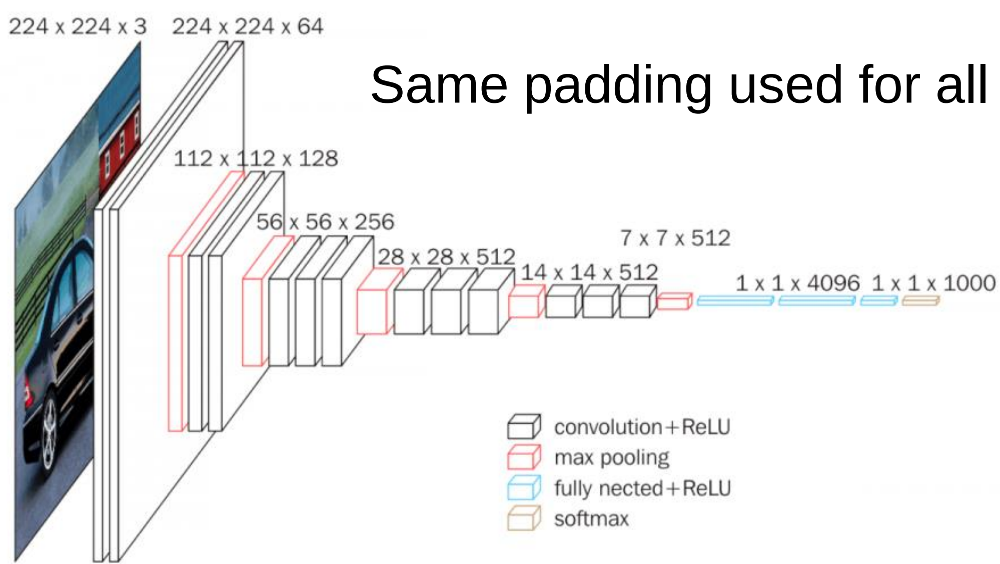


$224\times224\times3 \xrightarrow[\times 2]{\text{[CONV 64]}} 224\times224\times64 \xrightarrow{\text{POOL}} 112\times112\times64 \xrightarrow[\times 2]{\text{[CONV 128]}} 112\times112\times128 \xrightarrow{\text{POOL}} 56\times56\times128$

$\xrightarrow[\times 3]{\text{[CONV 256]}} 56\times56\times256 \xrightarrow{\text{POOL}} 28\times28\times256 \xrightarrow[\times 3]{\text{[CONV 512]}} 28\times28\times512 \xrightarrow{\text{POOL}} 14\times14\times512$

$\xrightarrow[\times 3]{\text{[CONV 512]}} 14\times14\times512 \xrightarrow{\text{POOL}} 7\times7\times512 \longrightarrow \underset{4096}{\text{FC}} \longrightarrow \underset{4096}{\text{FC}} \longrightarrow \underset{1000}{\text{Softmax}}$

Same padding used for all CONV layers

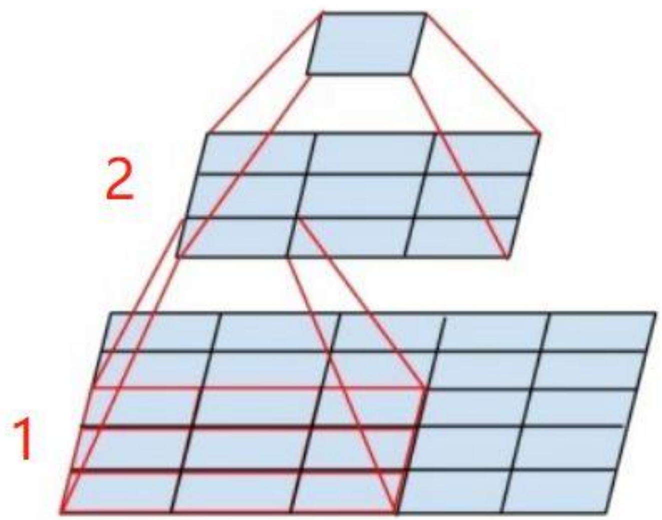


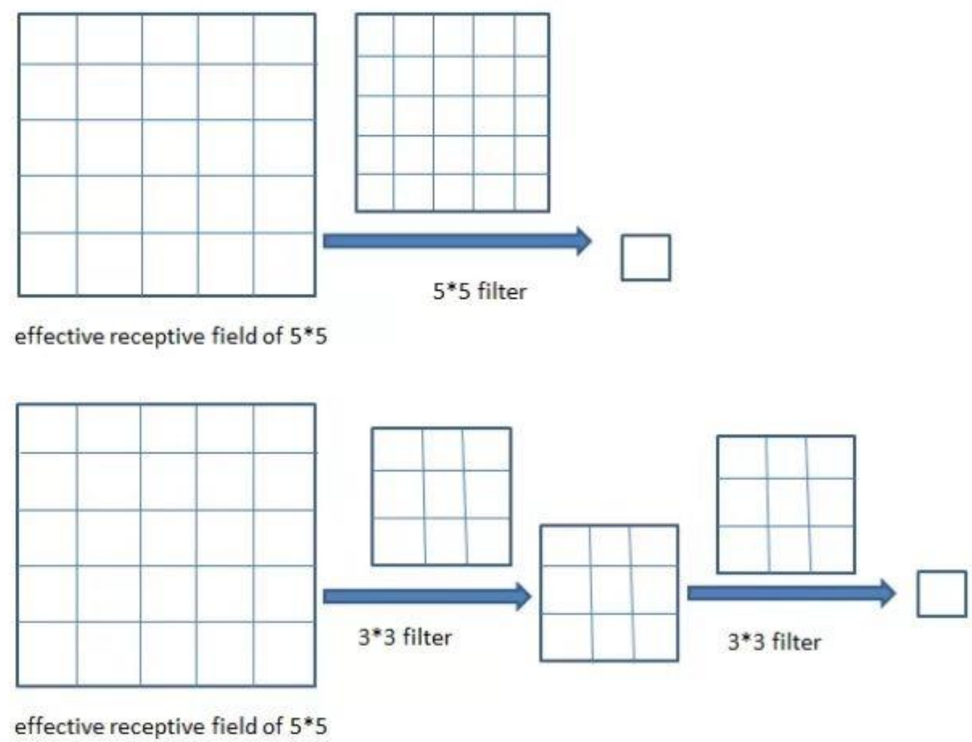
VGG-16 Details

- VGG-16 has 16 weight layers, not including POOL layers w. 0 weight
- Input image: $224 \times 224 \times 3$
- 1st and 2nd CONV layers: $64\ 3 \times 3$ filters w. stride $S = 1$, pad $P = 1$
  - Output volume: $224 \times 224 \times 64$ (since $\frac{1}{1}(224 + 2 * 1 - 3) + 1 = 224$)
- 3rd POOL layer: 2×2 filters w. stride $S = 2$
  - Output volume: $112 \times 112 \times 64$ (since $\frac{1}{2}(224 - 2) + 1 = 112$)
- 4th and 5th CONV layers: $128\ 3 \times 3$ filters w. stride $S = 1$, pad $P = 1$
  - Output volume: $112 \times 112 \times 128$ (since $\frac{1}{1}(112 + 2 * 1 - 3) + 1 = 112$)
- 6th POOL layer: 2×2 filters w. stride $S = 2$
  - Output volume: $56 \times 56 \times 128$ (since $\frac{1}{2}(112 - 2) + 1 = 56$)
- Total No. params: 60M
- ImageNet top 5 error: 7.3%

Stacked 3×3 CONV Layers

- 2 stacked 3×3 CONV layers w. pad $P = 1$ have the same effective receptive field as a 5×5 CONV layer; 3 stacked 3×3 CONV layers w. pad $P = 1$ have RF of 7×7; L stacked 3×3 CONV layers w. pad $P = 1$ have RF of $1 + 2L$
- Benefits:
  - Fewer params. Suppose all volumes have the same depth D, then a 7×7 CONV layer has $(7 * 7 * D + 1) * D \approx 49D^2$ params, while three stacked 3x3 CONV layers have only $(3 * 3 * D + 1) * D * 3) \approx 27D^2$ params
  - Two layers of non-linear activation functions increases CNN depth, hence larger model capacity





https://zhuanlan.zhihu.com/p/79258431

VGGNet No. Params

```
INPUT: [224x224x3]        memory:  224*224*3=150K   weights: 0
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64]  memory:  112*112*64=800K   weights: 0
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory:  56*56*128=400K   weights: 0
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory:  28*28*256=200K   weights: 0
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory:  14*14*512=100K   weights: 0
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory:  7*7*512=25K  weights: 0
FC: [1x1x4096]  memory:  4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory:  4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory:  1000 weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

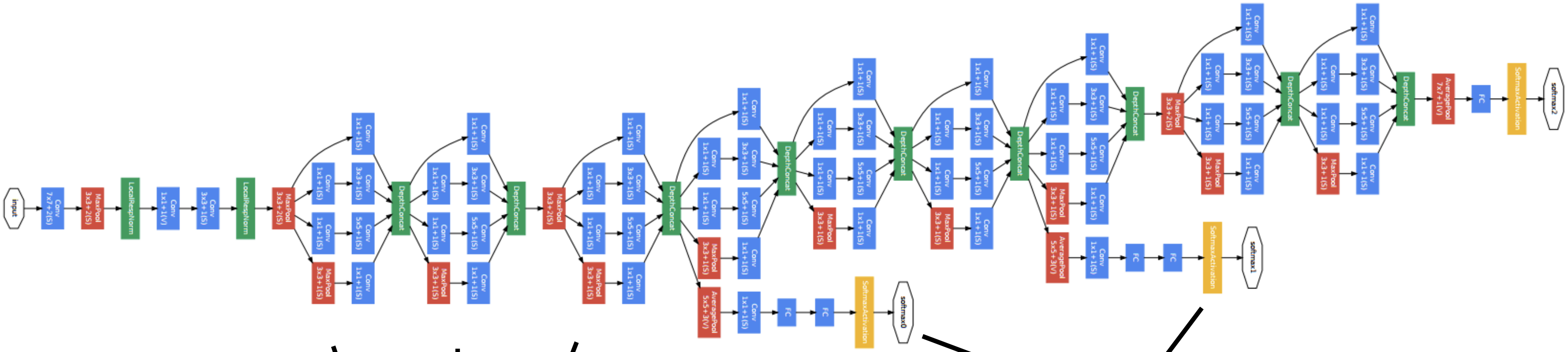
- Memory refers to memory size of activation maps
- For ease of calculation, only the No. weights are counted, not the biases

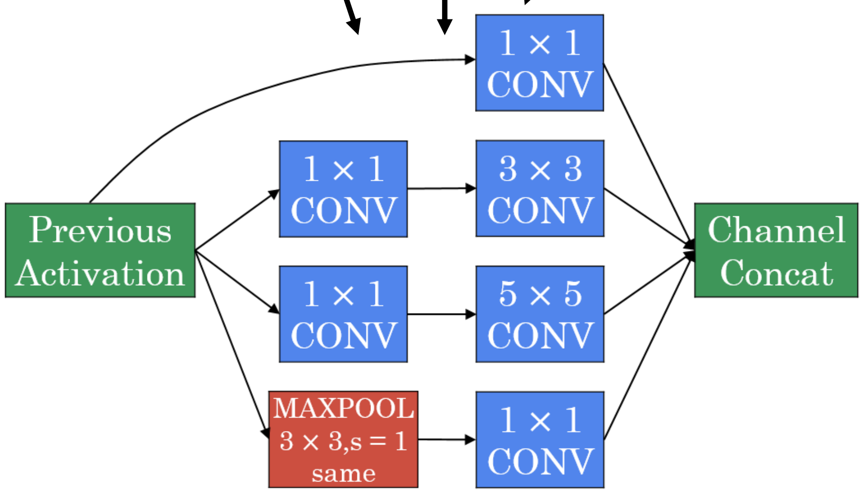
VGGNet Variants

Best performing variant
VGG-16

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

GoogLeNet [Szegedy et al., 2014]



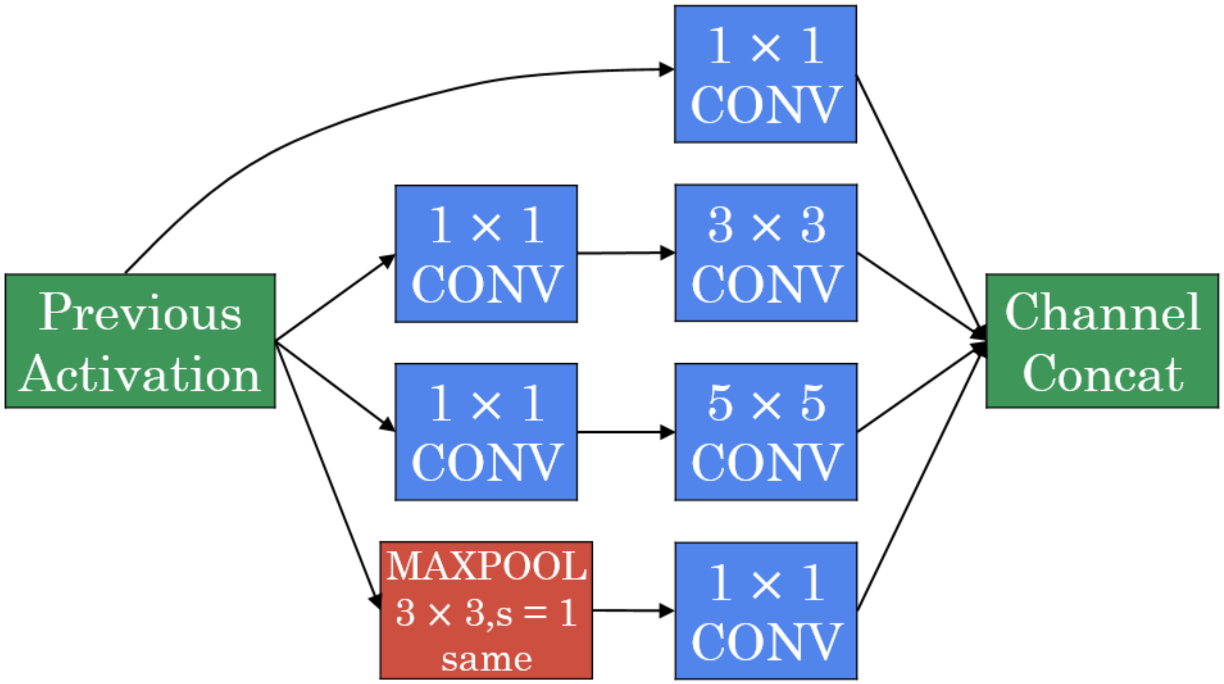


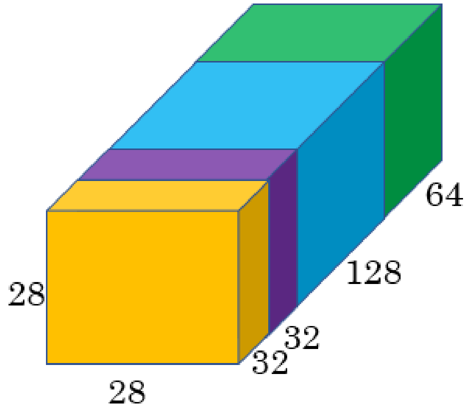
Inception Module

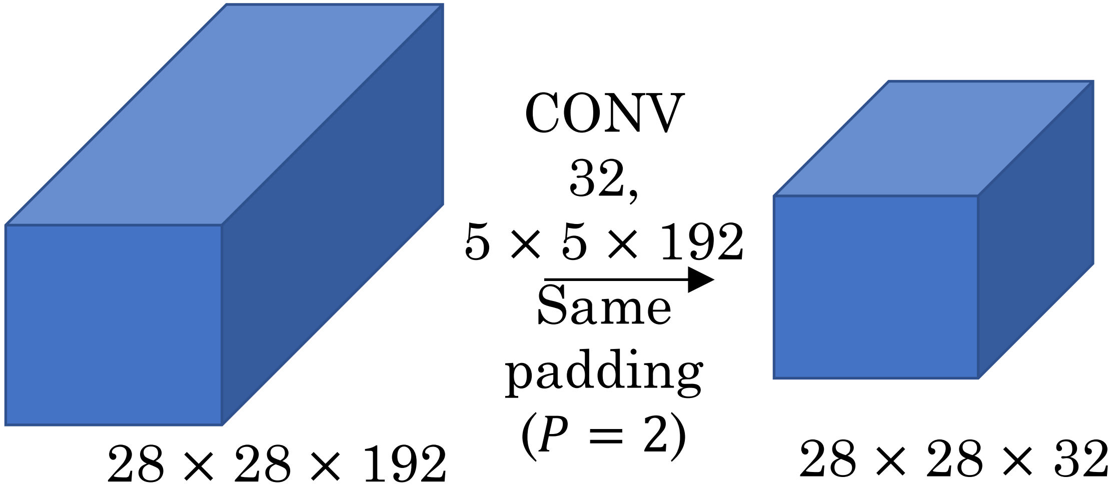
Additional classification heads for regularization

Inception Module

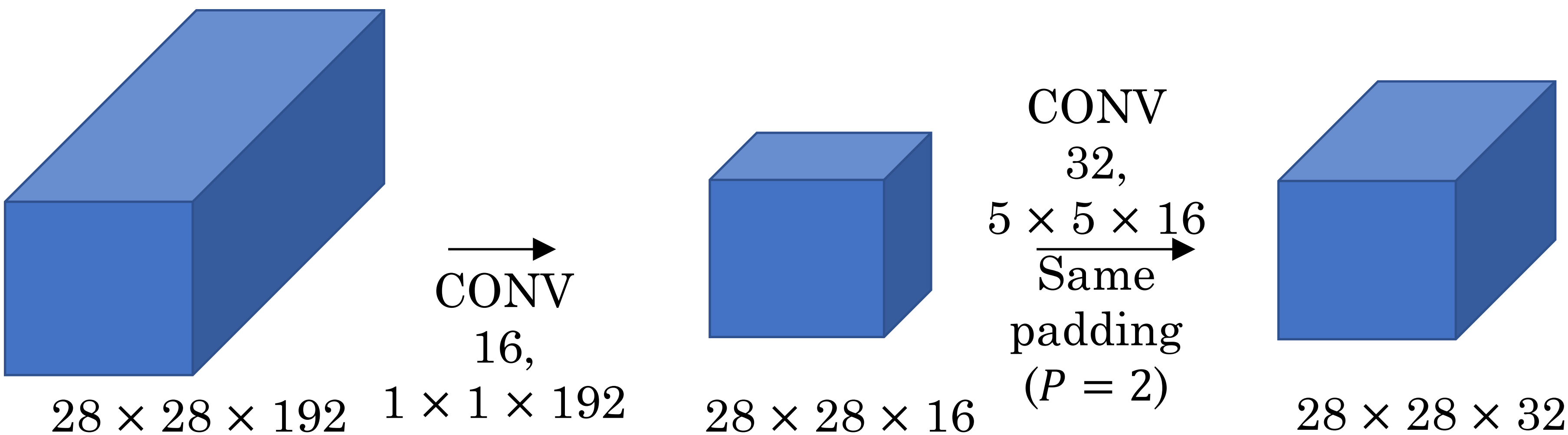
- Can't make up your mind about filter size? Have them all in the Inception Module!
  - But this increases computation load
- Additional 1×1 CONV layers serve as bottleneck to reduce number of parameters and computation load







- Without the bottleneck layer: No. params: $5 * 5 * 192 * 32 = 153600$; No. MULs: $(5 * 5 * 192) * (32 * 28 * 28) = 120\text{M}$



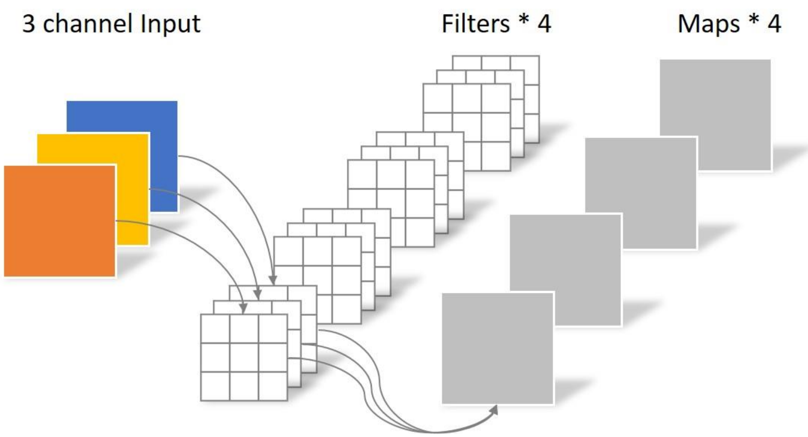
- With the bottleneck layer: No. params: $1 * 1 * 192 * 16 + 5 * 5 * 16 * 32 = 15872$; No. MULs: $(1 * 1 * 192) * (16 * 28 * 28) + (5 * 5 * 16) * (32 * 28 * 28) = 12.4\text{M}$

GoogLeNet Size

- Compared to AlexNet:
  - 12x less params (only 5M, due to no FC layers), 2x more compute (due to more CONV layers)

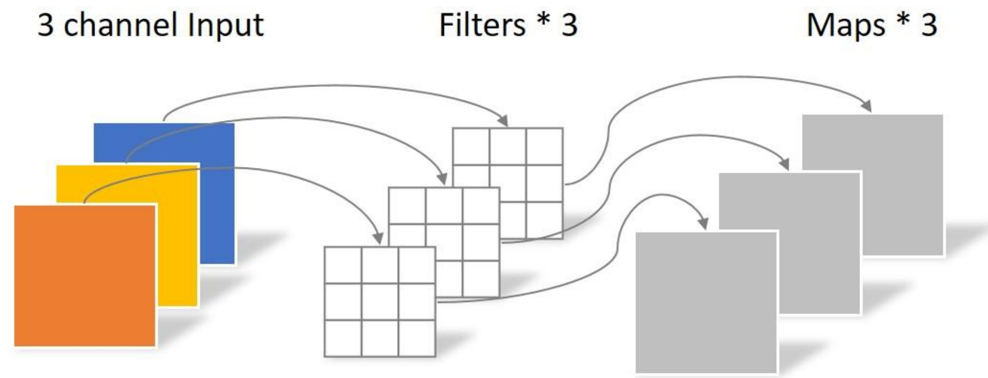
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Xception [Chollet 2017] MobileNets [Howard et al. 2017] : Depthwise Separable Convolution

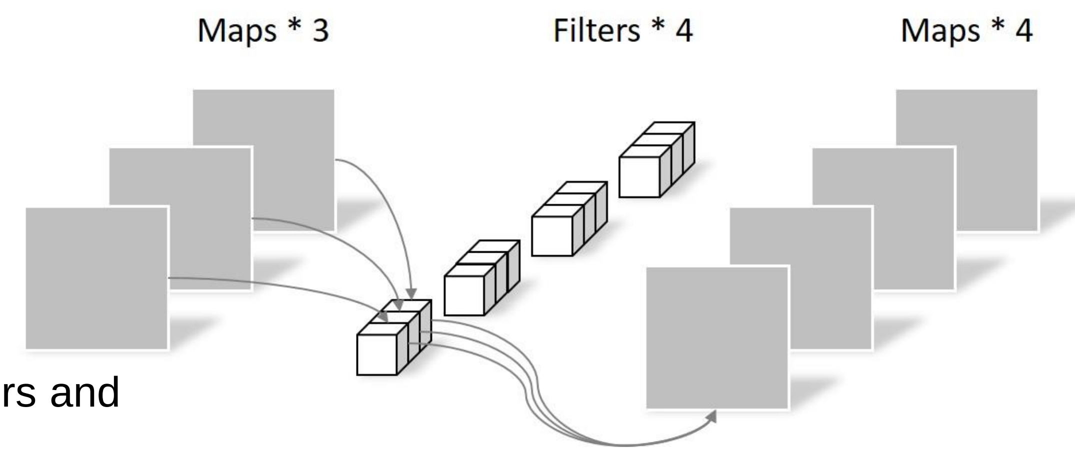


Each filter is convolved with all input channels

Regular Convolution



Each filter is convolved with one input channel



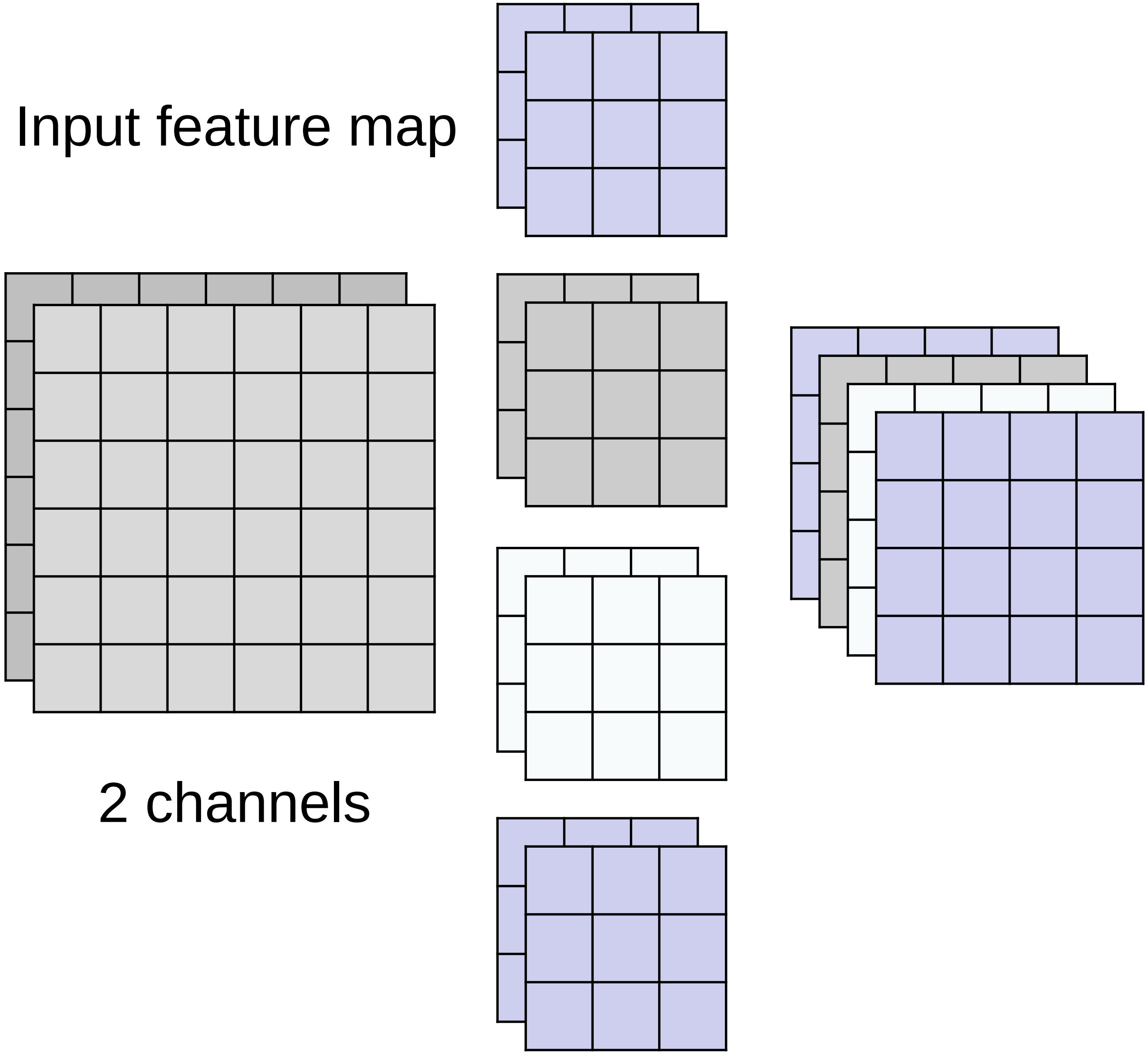
Followed by pointwise convolution

The intermediate feature maps serve as bottleneck to reduce number of parameters and computation load

(Optional) Depthwise Separable Convolution - A FASTER CONVOLUTION!

https://www.youtube.com/watch?v=T7o3xvJLuHk

Example: Regular Convolution

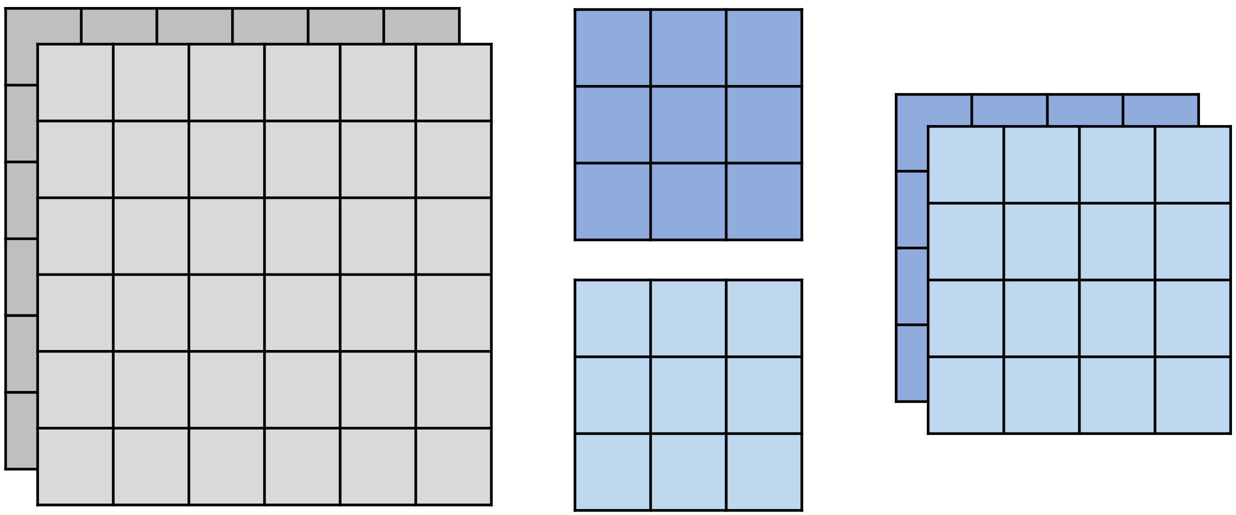


No. params: $3 * 3 * 2 * 4 = 72$ (Four $3 \times 3 \times 2$ filters) (not counting biases)

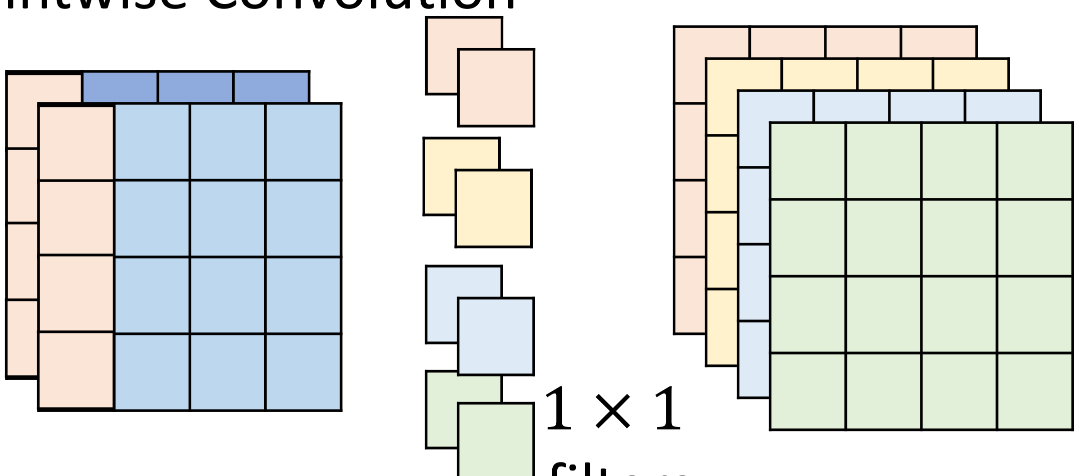
No. MULs: $(3 * 3 * 2) * (4 * 4 * 4) = 1152$ ($3 * 3 * 2$ MULs to compute each output element; $4 * 4 * 4$ output elements)

Example: Depthwise Separable Convolution

1. Depthwise Convolution



2. Pointwise Convolution

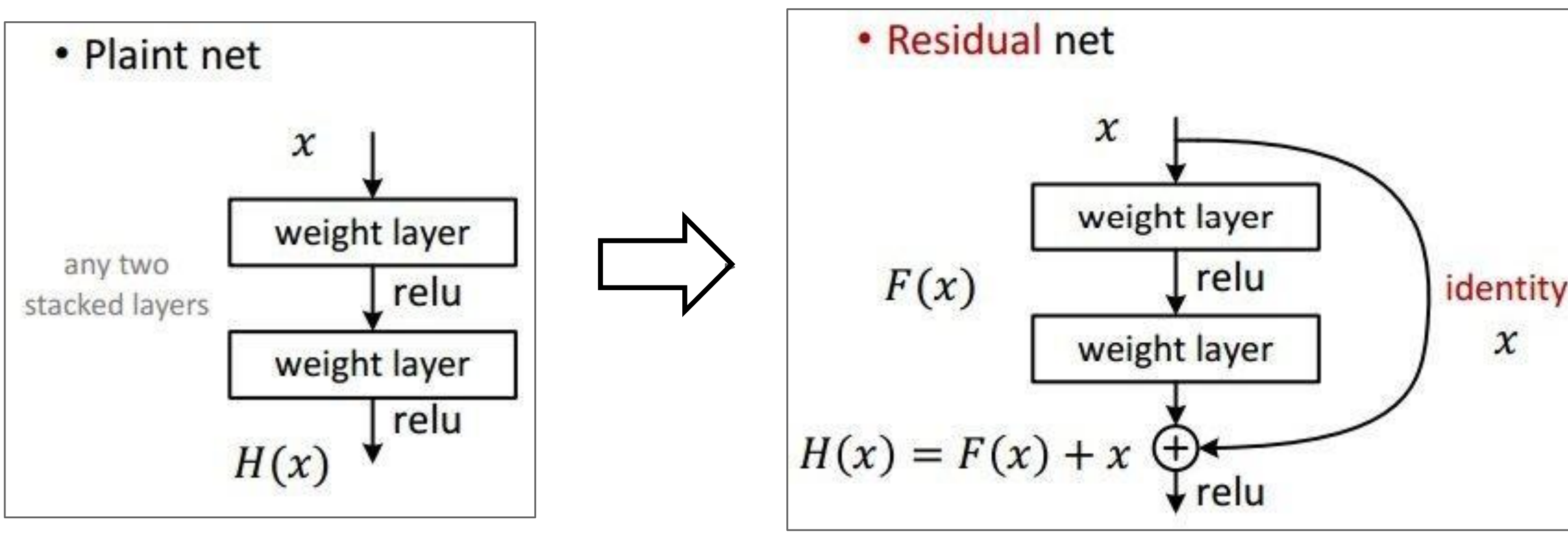


No. params: $3 * 3 * 2 + 2 * 4 = 26$ (Two $3 \times 3 \times 1$ filters and four $1 \times 1 \times 2$ filters) (not counting biases)

No. MULs: $(3 * 3 * 1) * (2 * 4 * 4) + (1 * 1 * 2) * (4 * 4 * 4) = 416$ (Depthwise Conv: $3 * 3 * 1$ MULs to compute each output element; $2 * 4 * 4$ output elements; Pointwise Conv: $1 * 1 * 2$ MULs to compute each output element; $4 * 4 * 4$ output elements)

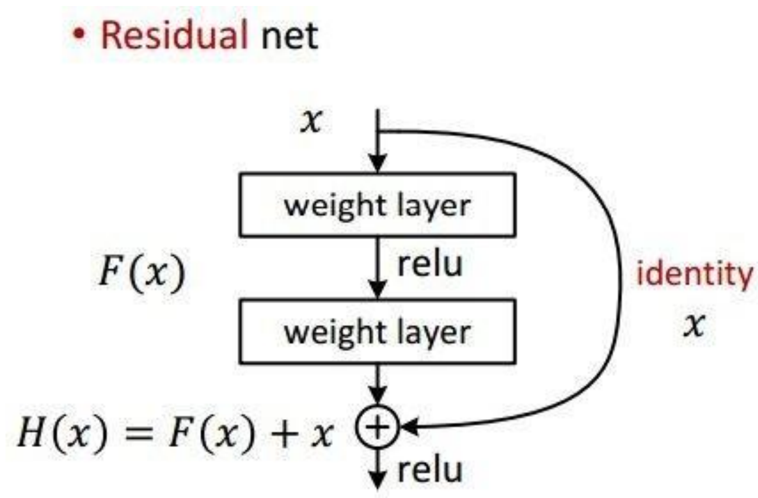
Residual Networks (ResNet) [He et al. 2015]

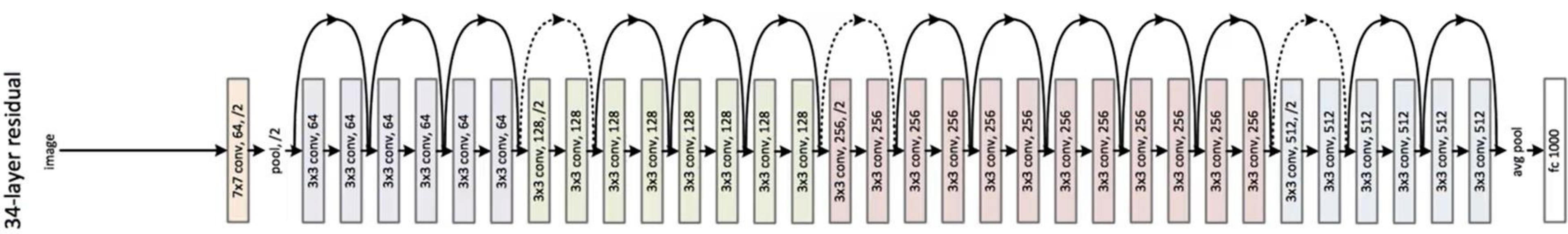
- Based on VGG-19, adding more layers and skip connections
- ImageNet top 5 error: 3.6%



ResNet Skip Connection

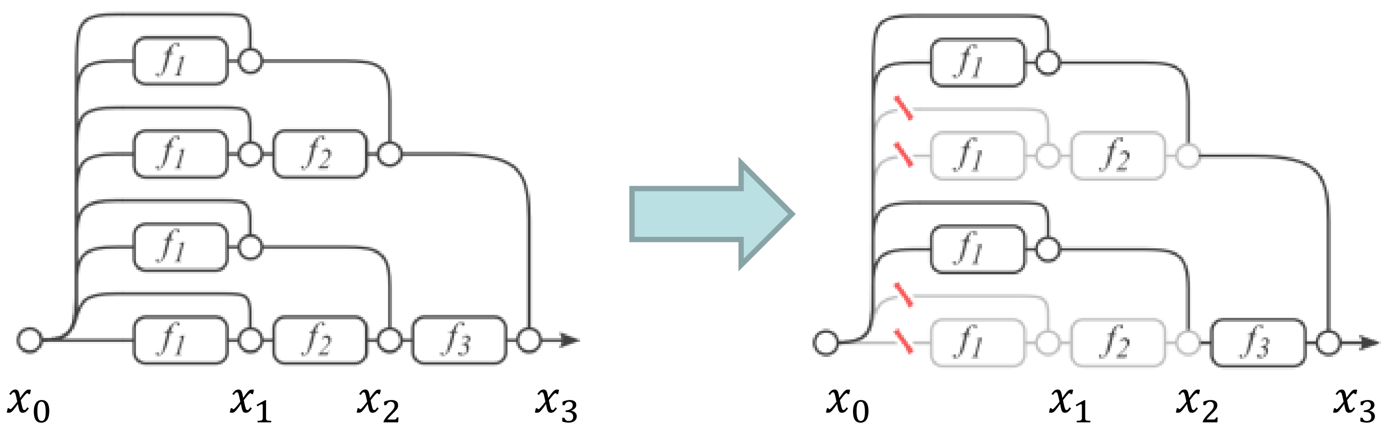
- In a standard network, output from a given layer is $F(x)$
- In ResNet w. the identity skip (or short-cut) connection, output from a given layer is $H(x) = F(x) + x$
- Benefits:
  - Residual connections help in handling the vanishing gradient problem in very deep NNs
  - If identify mapping is close to optimal, then weights can be small to capture minor differences only, in other words, “unnecessary layers” can learn to be identity mapping. This allows stacking many layers (e.g., 152) without overfitting





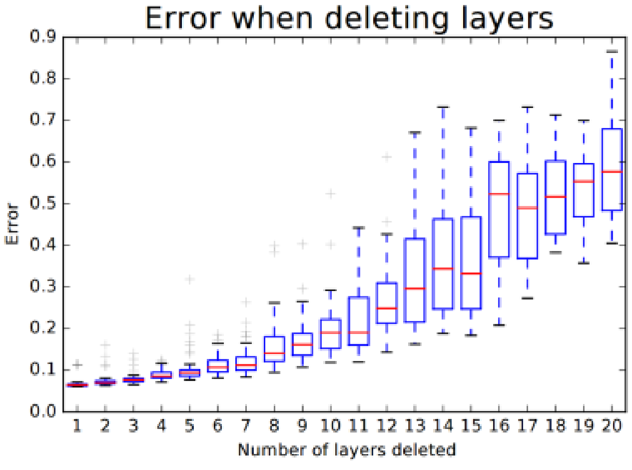
Consider a 3-layer Network

- Standard NN:
  - $x_3 = f_3(f_2(f_1(x_0)))$
- ResNet:
  - $x_1 = f_1(x_0) + x_0$
  - $x_2 = f_2(x_1) + x_1 = f_2(f_1(x_0) + x_0) + f_1(x_0) + x_0$
  - $x_3 = f_3(x_2) + x_2 = f_3(f_2(f_1(x_0) + x_0) + f_1(x_0) + x_0) + f_2(f_1(x_0) + x_0) + f_1(x_0) + x_0$
- Suppose $f_2(x_1)$ is a vector of very small values (layer 2 is "off"/skipped), then it looks like the input x_0 bypassed the second layer completely on its way to the output x_3



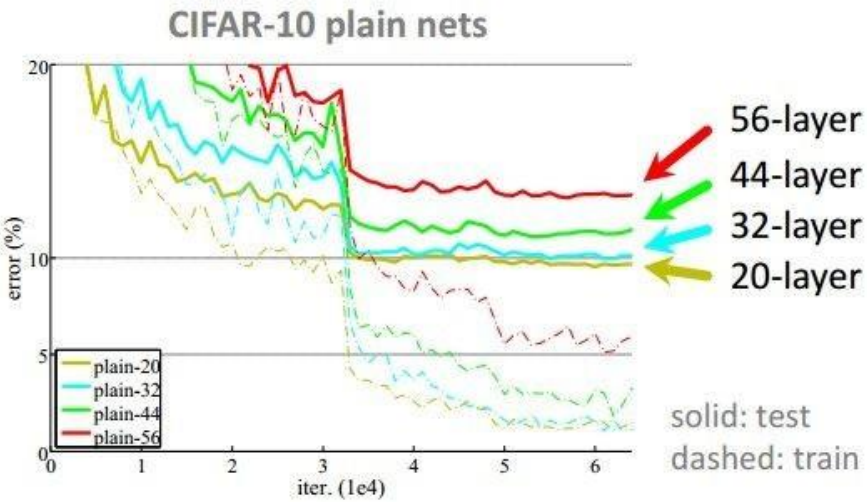
ResNet is an Ensemble of Models

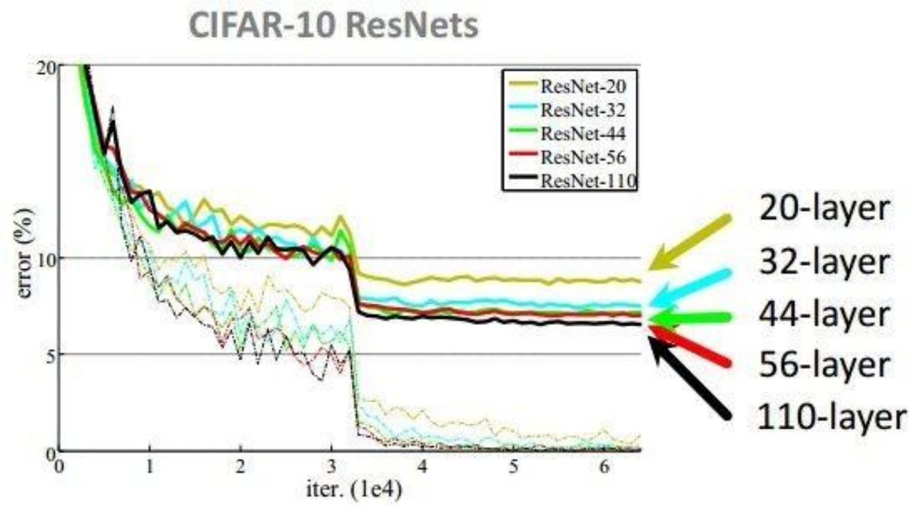
- Every input x_0 to ResNet may activate a unique path to the output. Total number of possible paths is 2^N, where N is the total number of layers in the network, since each layer may be either "on" or "off" for a given input x_0
  - Compare w. a standard network, where there is only one single path for any input corresponding to all layers being "on", and no layer is skipped
- Consequences:
  - Resilience to layer deletion: deleting 1-3 layers in a large ResNet introduces only around 6-7% error
  - Shortening of effective paths: w. 152-layer ResNet, most paths are only 20-30 levels deep!



Deeper Nets have Better Performance

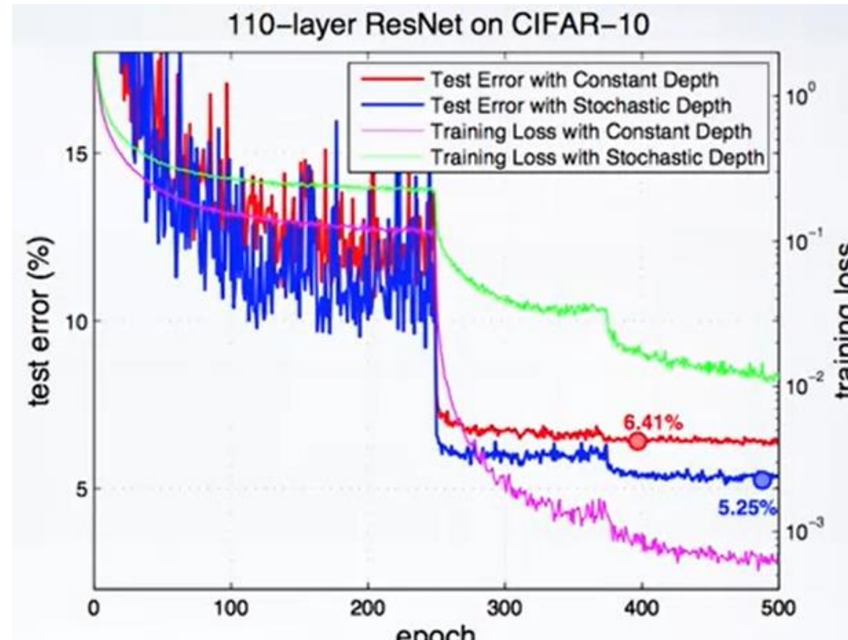
CIFAR-10 experiments



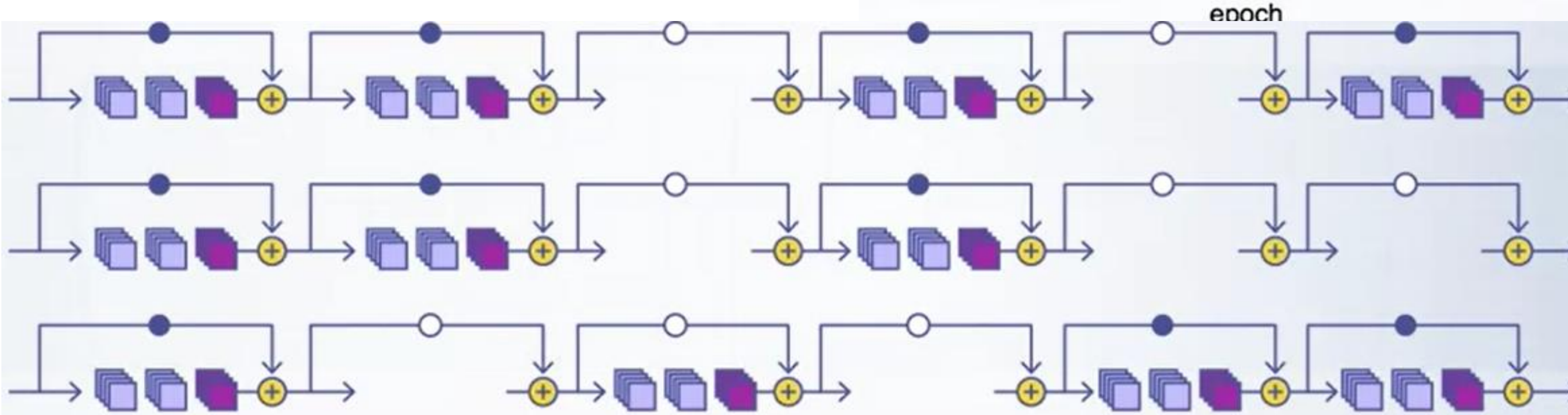


ResNet Training with Stochastic Depth

- For each minibatch of inputs, randomly skip some layers (replaced w. identity mapping)
- Reduced network depth during training; full depth during inference

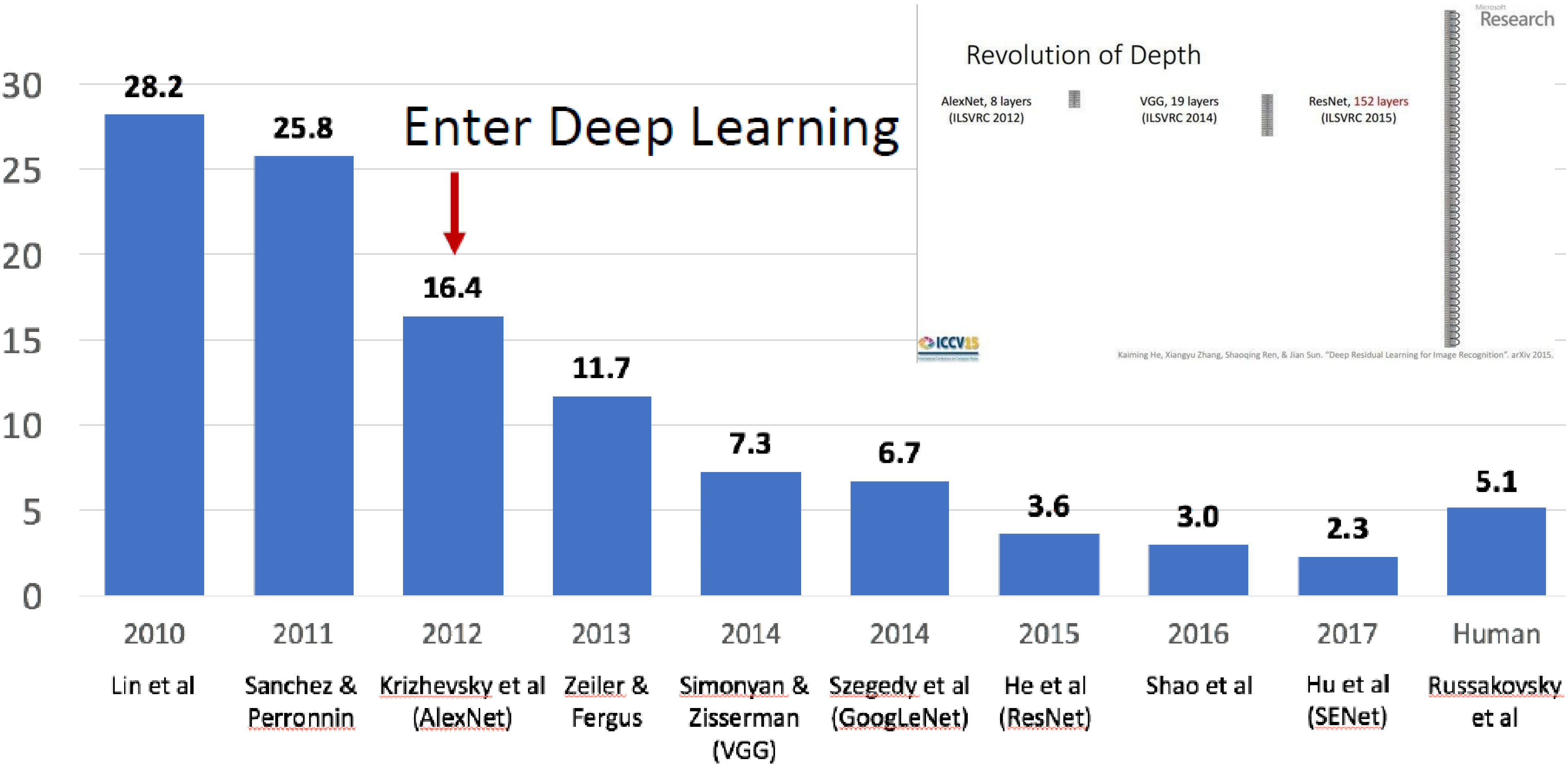


MB1
MB2
MB3



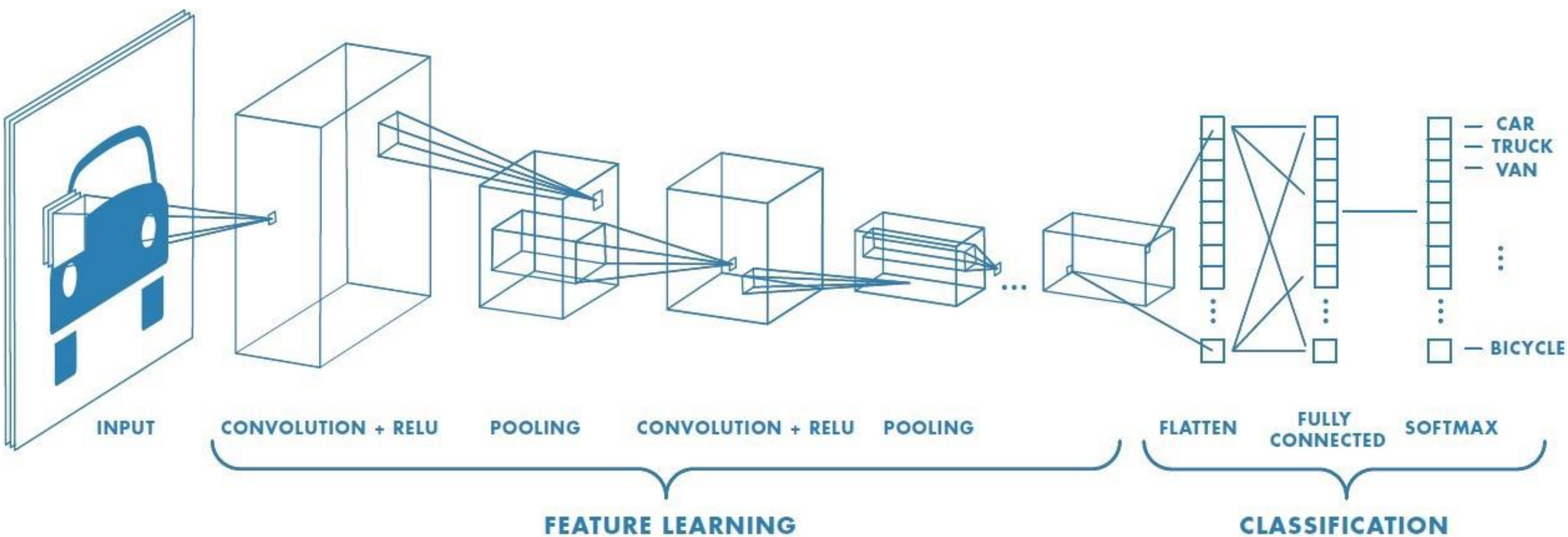
ImageNet Large Scale Visual Recognition Challenge

- 1,000 object classes, 1.4 M labeled images



CNN Layer Patterns

- A typical CNN architecture looks like: `INPUT->[[CONV->RELU]*N->POOL?]*M->[FC->RELU]*K->FC`
  - where `*` indicates repetition, and `POOL?` indicates an optional pooling layer. $N \geq 0$ (usually $N \leq 3$), $M \geq 0$, $K \geq 0$ (and usually $K < 3$)
- Some common architectures:
  - `INPUT->FC`, implements a linear classifier. Here N = M = K = 0.
  - `INPUT->CONV->RELU->FC`
  - `INPUT->[CONV->RELU->POOL]*2->FC->RELU->FC` (fig below). There is a single CONV layer between every POOL layer.
  - `INPUT->[CONV->RELU->CONV->RELU->POOL]*3->[FC->RELU]*2->FC` There are two `CONV` layers stacked before every `POOL` layer, e.g., two stacked 3×3 CONV Layers. This is generally a good idea for larger and deeper networks, because multiple stacked `CONV` layers can develop more complex features of the input volume before the destructive pooling operation.



Layer Sizing Rules-of-Thumb

- The input layer (that contains the image) should be divisible by 2 many times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. ImageNet), 384, and 512.
- The CONV layers should use small filters (e.g. 3x3 or at most 5x5), stride S=1. The input volume should have “same padding”, i.e., the conv layer does not alter the spatial size of the input. For any F, pad P=(F−1)/2 preserves the input size, e.g., when F=3, P=1; when F=5, P=2. This means the CONV layers only transform the input volume depth-wise, but do not perform downsampling. (c.f. CONV Example 3 and VGGNet).
- The POOL layers are in charge of downsampling the spatial dimensions of the input. The most common setting is to use max-pooling with 2x2 receptive fields (F=2), with stride of 2 (S=2). A less common setting is to use F=3, S=2. It is uncommon to see receptive field sizes for max pooling that are larger than 3, because the pooling is then too lossy and aggressive.
- In some cases (especially in early layers), the memory size can build up very quickly with the rules of thumb presented above. For example, filtering a 224x224x3 image with three 3x3 CONV layers with 64 filters each and padding 1 would create 3 activation volumes, each with size 224x224x64. This amounts to a total of about 10 million activations, or 72MB of memory (per image, for both activations and gradients). Since GPUs are often bottlenecked by memory, it may be necessary to compromise. In practice, make the compromise at only the first CONV layer that is looking at the input image. For example, AlexNet uses filter size of 11x11 and stride of 4 in the first CONV layer.

Memory Size Considerations

- From the intermediate volume sizes:
  - These are the raw number of activations at every layer of the CNN, and also their gradients (of equal size). Usually, most of the activations are on the earlier CONV layers of a CNN. These are kept around because they are needed for backpropagation during training, but for inference, we can store only the current activations at the current layer and discarding the activations from previous layers.
- From the parameter sizes:
  - These are the weights and biases, and their gradients during backprop, and also a step cache if the optimization is using momentum, Adagrad, or RMSProp. Therefore, the memory to store the parameter vector alone usually should be multiplied by a factor of at least 3 or so.
- Each number may need 4 B storage space for floating point, 8 B for double, or 1 B or smaller for optimized fixed-point implementations.

Transfer Learning

- Instead of training your CNN from scratch, start from a pre-trained CNN (e.g., ResNet) and fine-tune it for your task
- First, replace SoftMax layer (classification head) with your own
- Next, train the CNN while keeping parameters frozen for
  - all CONV layers and only train the SoftMax layer
  - or part of the earlier CONV layers close to the input layer (since earlier layers extract lower-level features that are more likely to be common among different tasks)
  - or none of the layers
  - The decision depends on how much training data you have, and how similar your task is to that of the pre-trained CNN