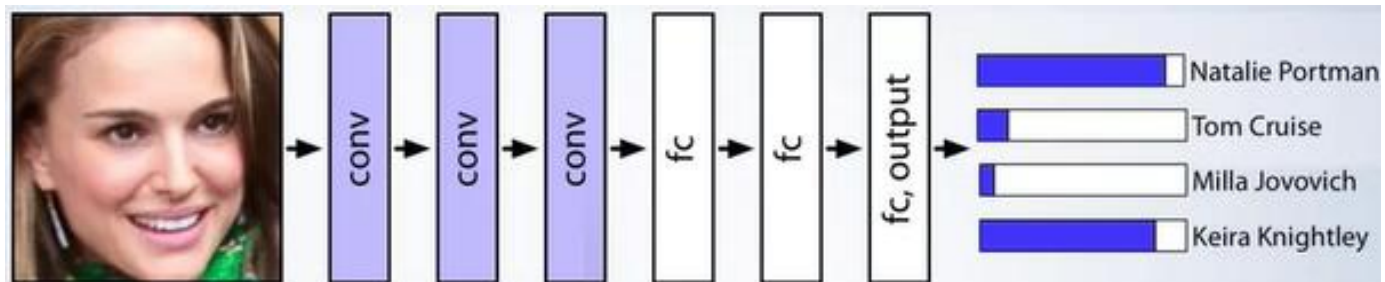


# LX CNNs and RNNs

Z. Gu 2021

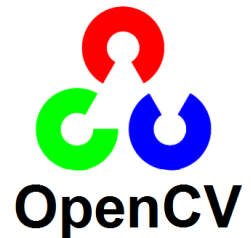


# Outline

- CNN Convolution layers
- Pooling and Fully-Connected layers
- CNN case studies
- RNNs

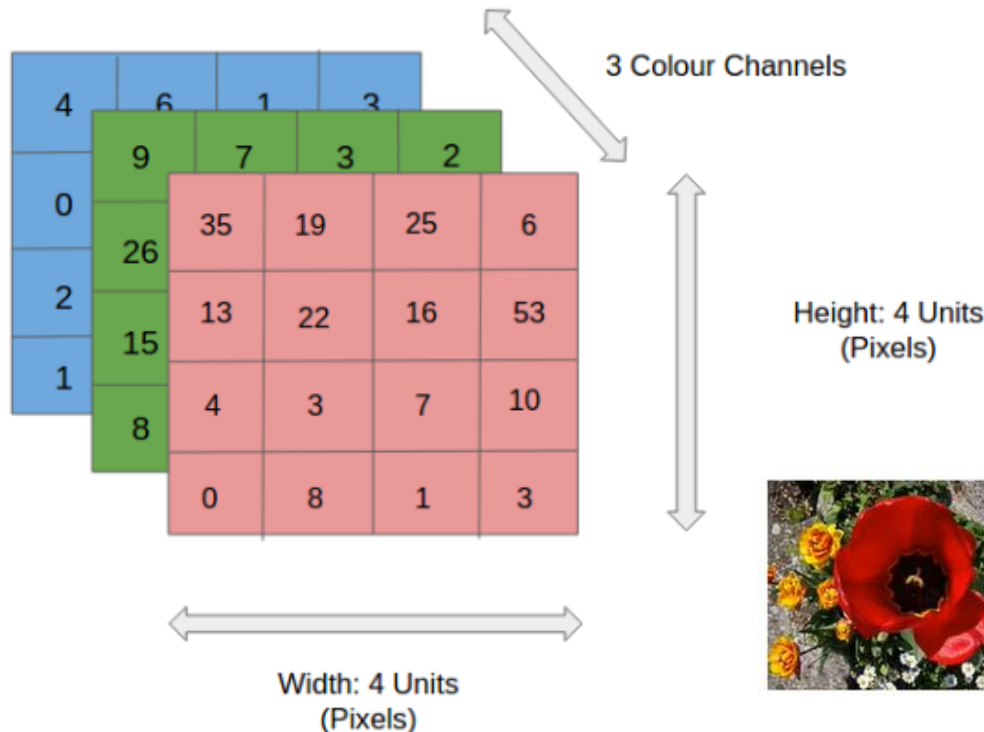
# Classic Computer Vision

- Most “classic” (non-ML) CV algorithms are implemented in the OpenCV library, including
  - Core Operations:
    - basic operations on image like pixel editing, geometric transformations, code optimization, some mathematical tools etc.
  - 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 conjunction w. complex ML/DL algorithms, which may be sometimes unreliable and unpredictable.

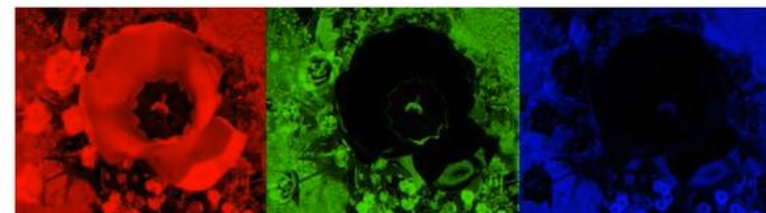


# Input Image Encoding

- 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
  - There are a number of color space encodings, incl. RGB, HSV, CMYK, etc.
- 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, i.e., true color (24-bit) is used in computer and phone display for human eyes, but 8-bit is typically enough for CV tasks

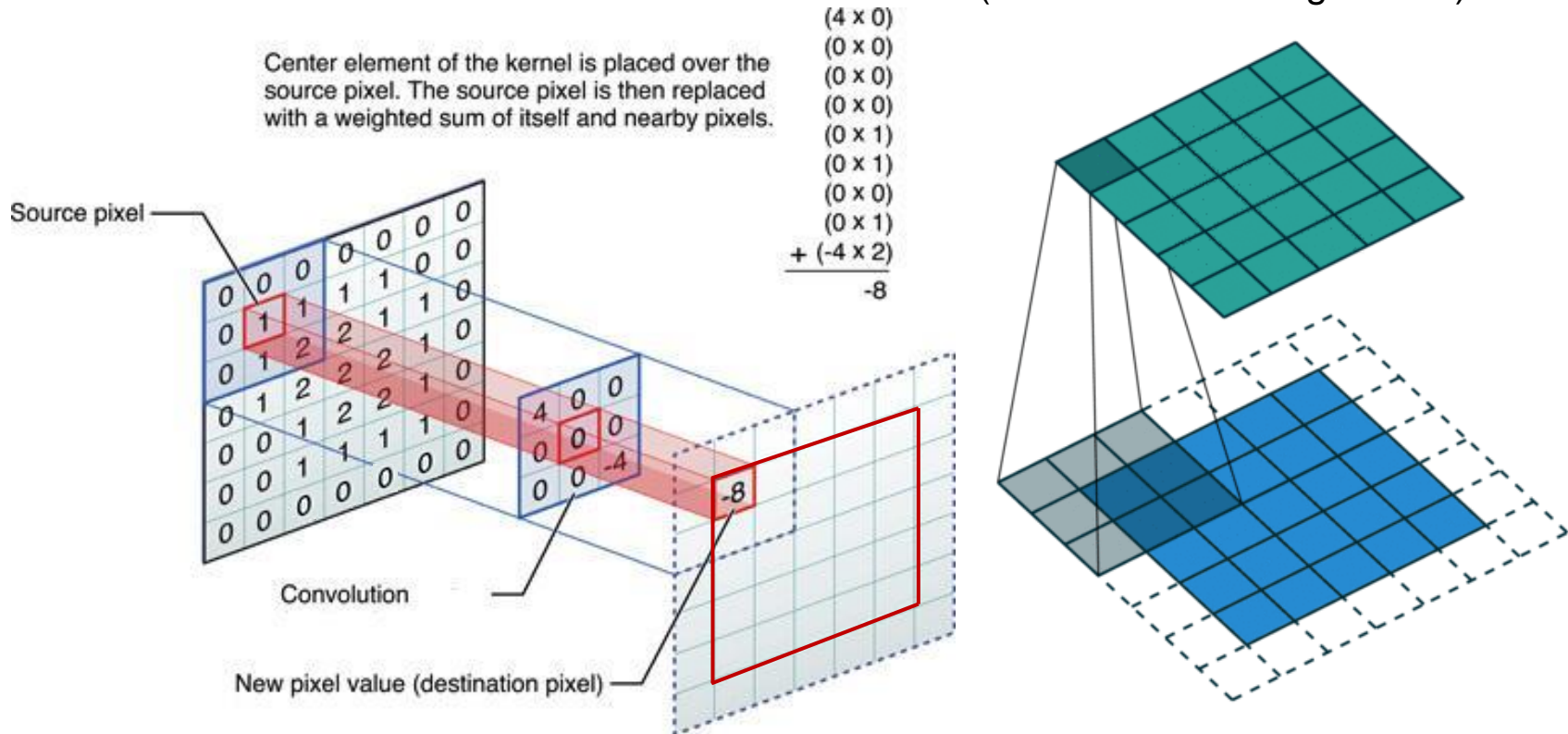


=












# Filters/Kernels in Computer Vision

- Convolution operation: we slide (convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter (kernel) and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position.
  - dot product operation: 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)

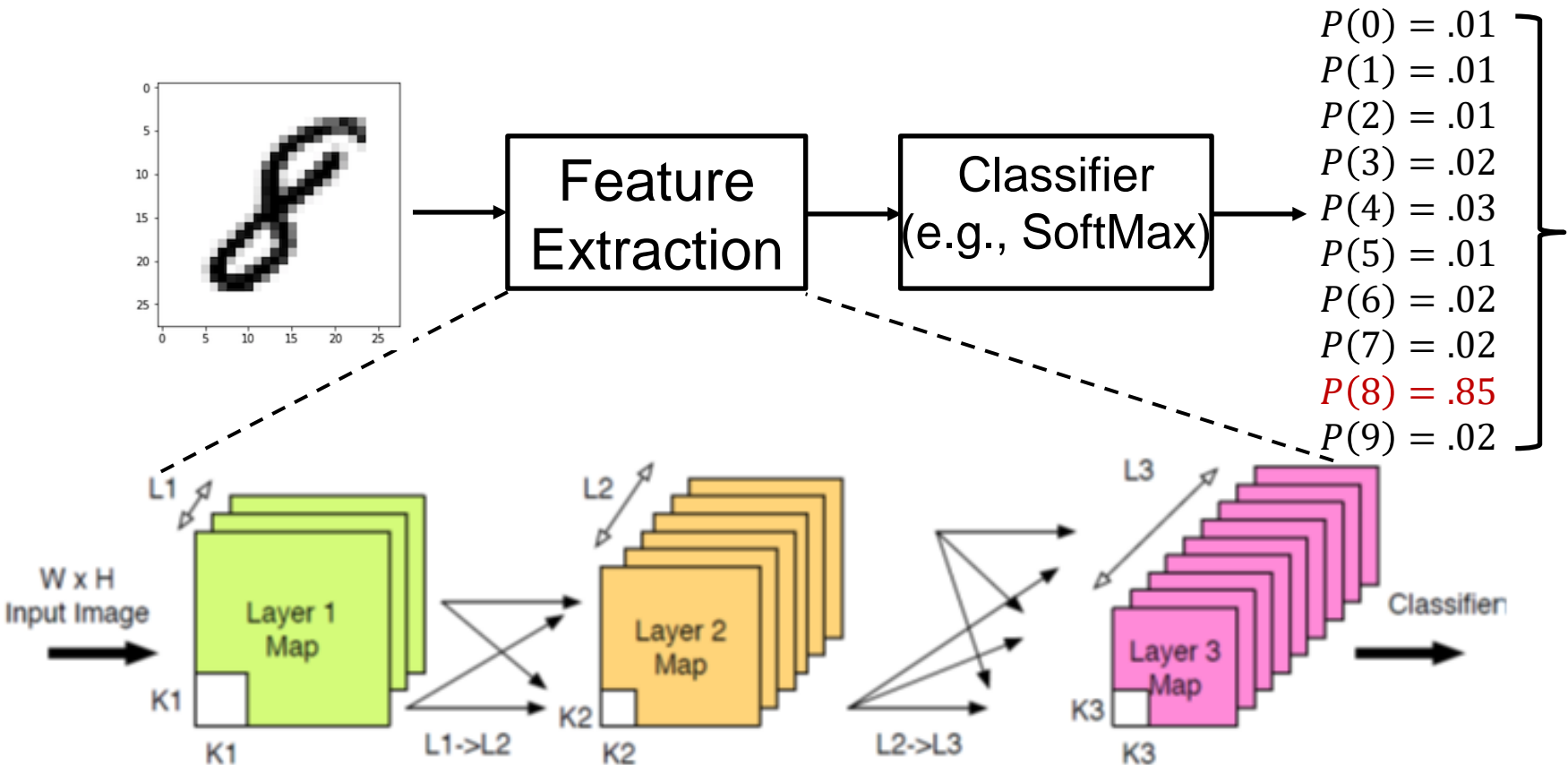


# Effects of Convolution w. Common Filters

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

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

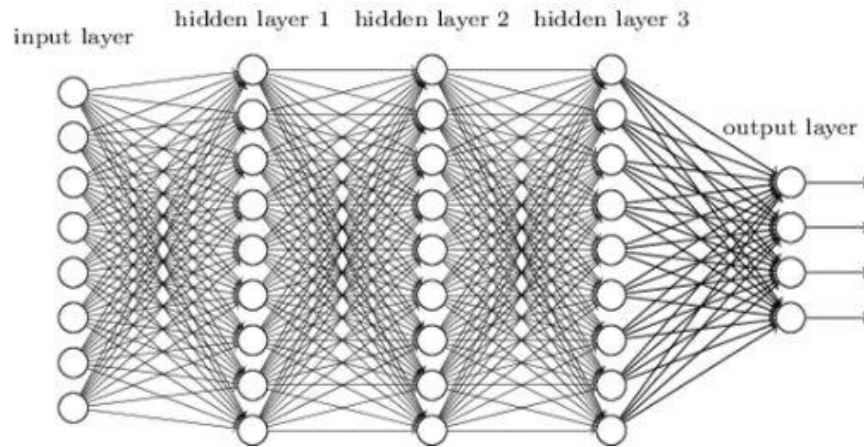
# Convolutional Neural Networks (CNN)



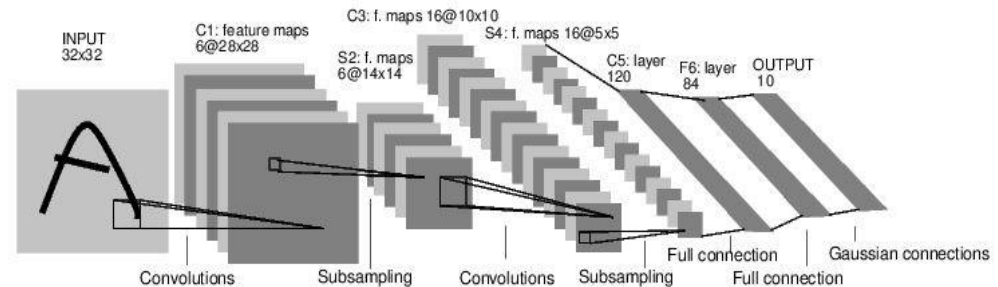
- A CNN (also called ConvNet) is a sequence of Convolutional (CONV) Layers, Pooling (POOL) Layers and non-linear activation functions for feature extraction, followed by one or more Fully-Connected (FC) Layers for classification based on the extracted features
- Difference from classic CV: learn **custom filters** in CONV layers from data by supervised learning, instead of using **hand-crafted filters**



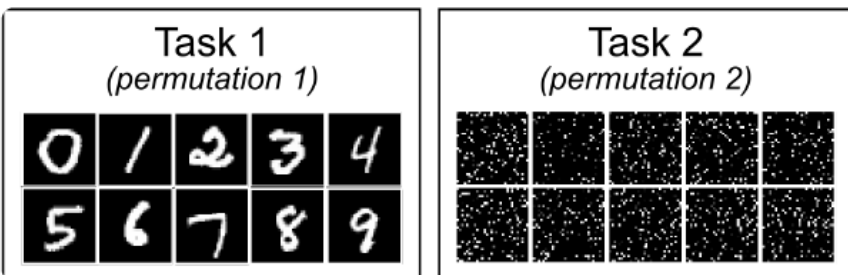
# Fully-Connected NN vs. CNN



- In a FCNN:
- Cannot alter input image size
- Permutation invariance
  - Cannot recognize geometry/locality of input, hence not useful for image recognition
  - Below two images, with same set of pixels but different permutations, will generate the exact same output in an MLP
- Num. params can grow very large



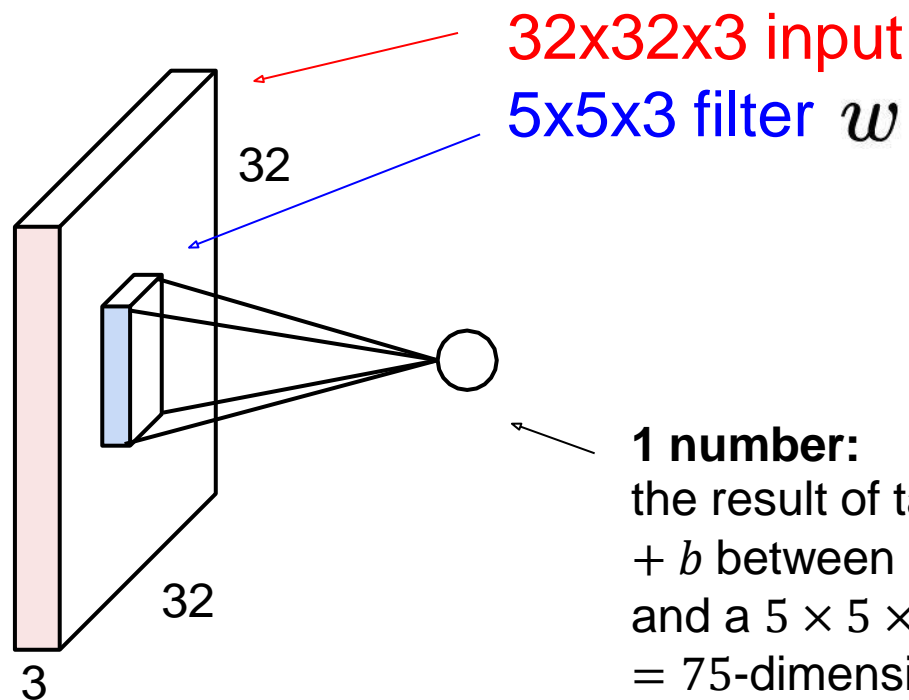
- In a CNN, only the last few (typically  $\leq 3$ ) layer(s) are Fully-Connected
- CONV layers can handle images of arbitrary size (but not the last MLP layer(s))
- Not permutation invariance
  - Can recognize geometry/locality of input
- Translation invariance
- Fewer params than MLP (less prone to overfitting)





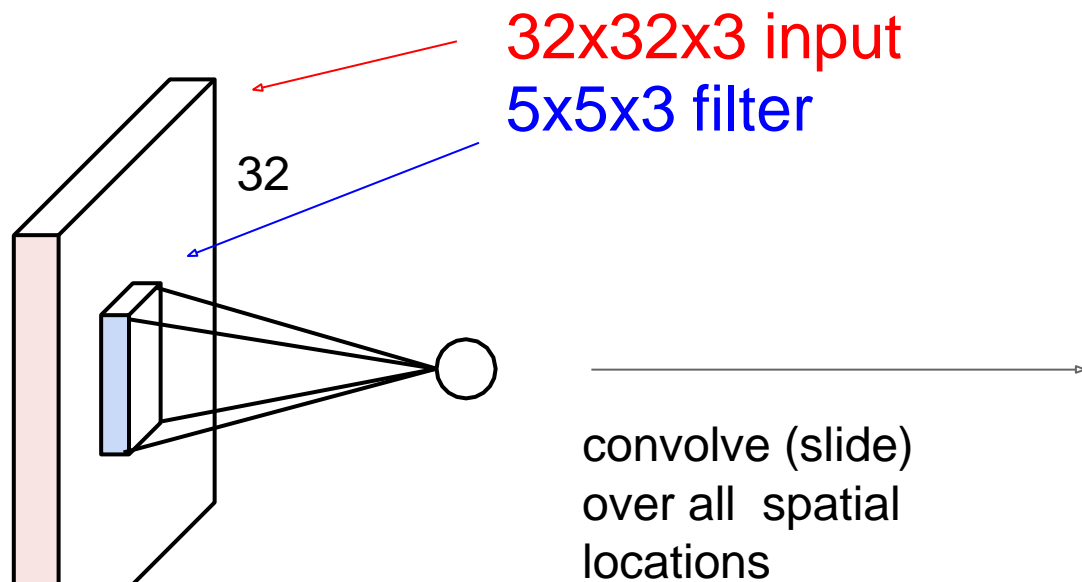
# Convolution Operation

- Filters always have the same depth as the input volume
  - The input may be a color image w. 3 channels (e.g., RGB), hence depth is 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 matrices
- Convolution: slide the filter over the image spatially, computing dot products  $w^T x + b$  to generate an activation map as output

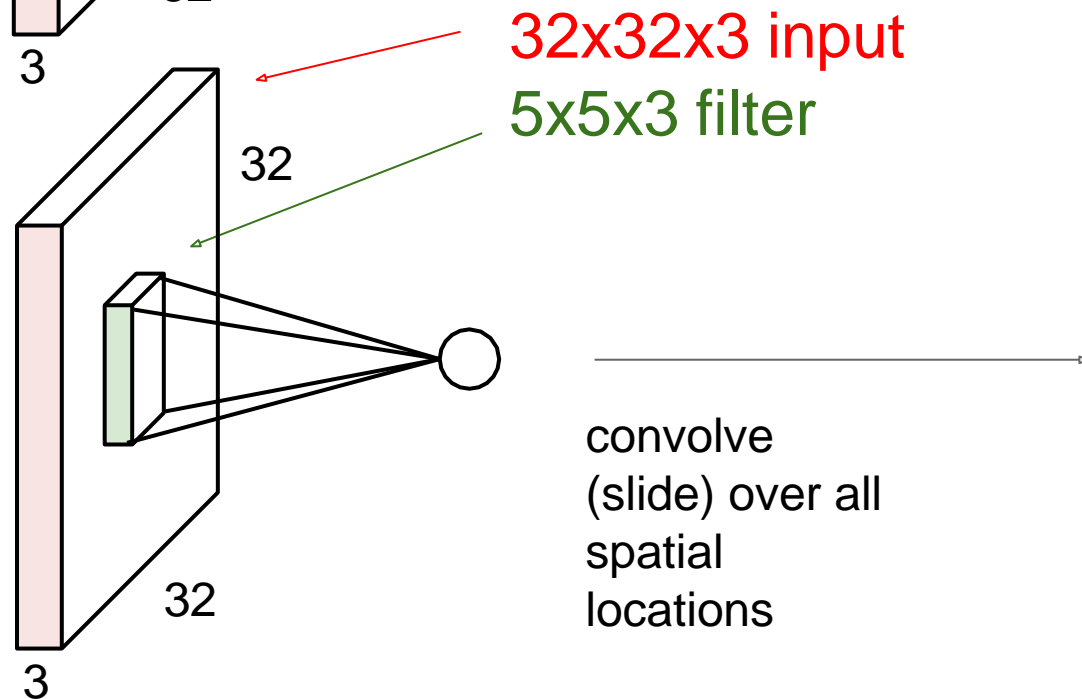


**1 number:**

the result of taking a dot product  $w^T x + b$  between the filter w. weights  $w$ , bias  $b$ , and a  $5 \times 5 \times 3$  image patch  $x$ , w.  $5 * 5 * 3 = 75$ -dimensional dot product + bias



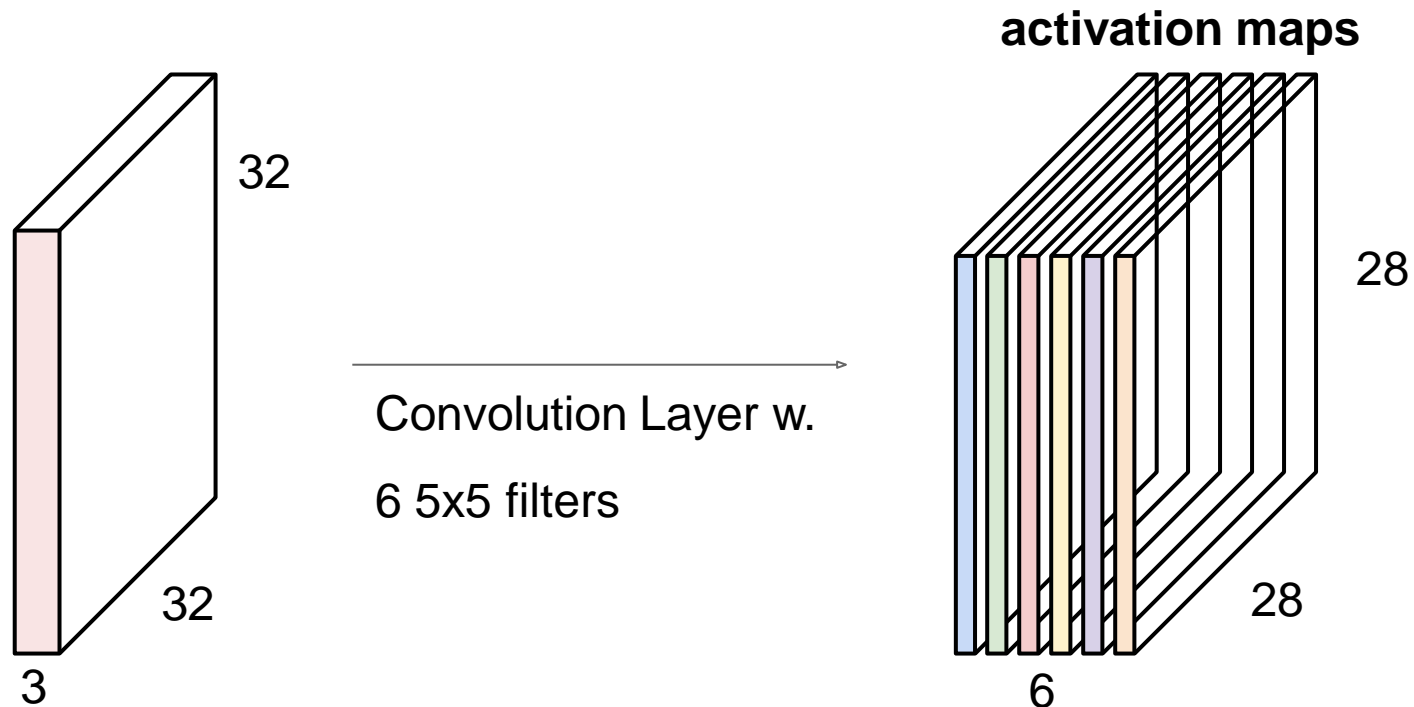
One (blue) filter  
generates one  
2D activation  
map as output



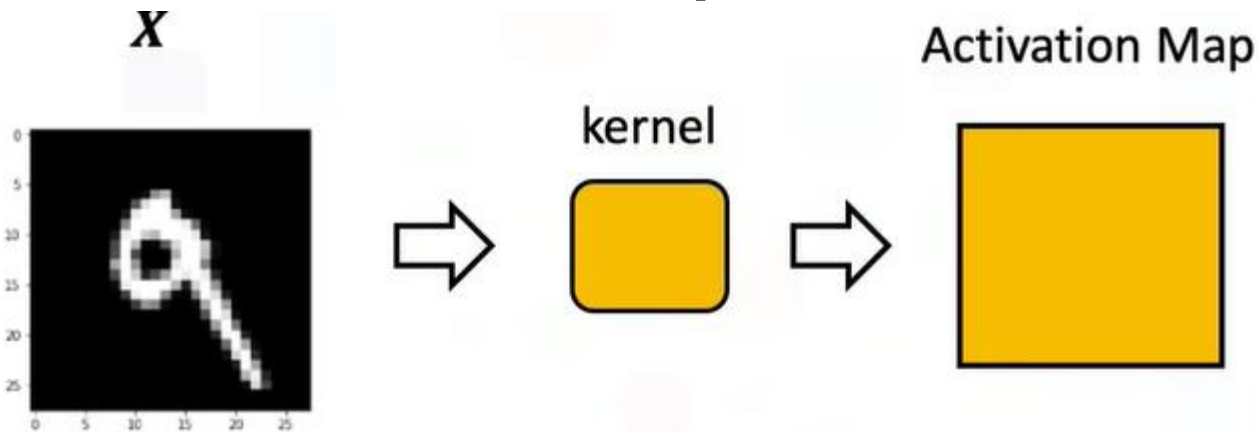
Multiple (blue  
and green)  
filters generate  
multiple (blue  
and green) 2D  
activation maps,  
stacked along  
the depth  
dimension to  
produce the 3D  
output volume

# Stacked Activation Maps

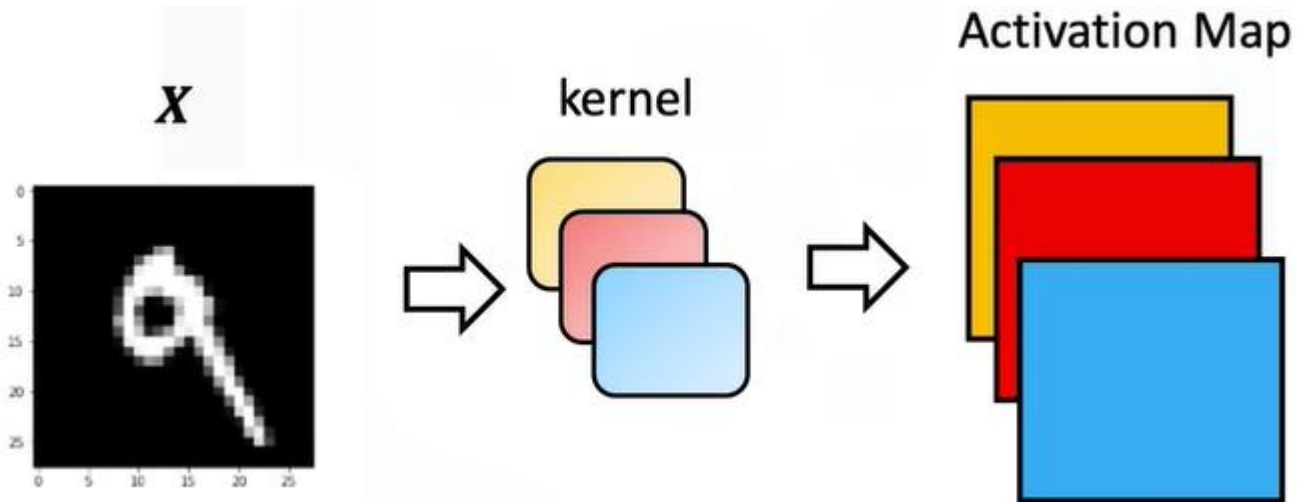
- If we have 6  $5 \times 5$  filters, we'll get 6 separate activation maps (also called feature maps).
- We stack these up to get an output volume (a new “image”) of size  $28 \times 28 \times 6$ , an intermediate representation to be passed to subsequent layers



# Activation Maps Illustration



1 filter/kernel, 1 output activation map

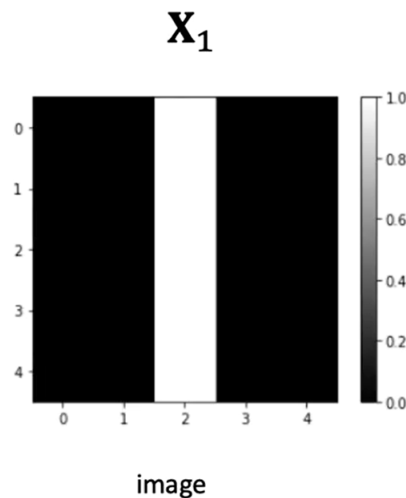


3 filters/kernels, 3 output activation maps

# Example: 3 Filters

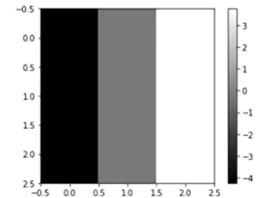
## Filters

- 3 filters  $W_0, W_1, W_2$ , each extracting different features. ( $W_i * X_j$  denotes convolution of filter  $W_i$  w. input  $X_j$ )
- Upper left: filter  $W_0$  extracts vertical line features  $Z_0$  from input image  $X_1$ . (the other 2 filters do not extract any meaningful features)
- Upper left: filter  $W_1$  extracts horizontal line features  $Z_1$  from input image  $X_2$  (the other 2 filters do not extract any meaningful features)



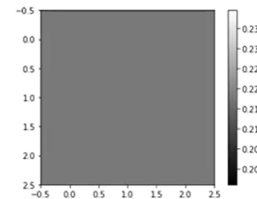
$$Z_0 = W_0 * X_1 + b_0$$

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



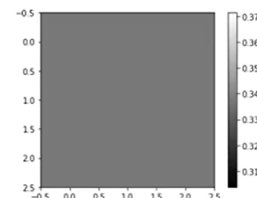
$$Z_1 = W_1 * X_1 + b_1$$

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

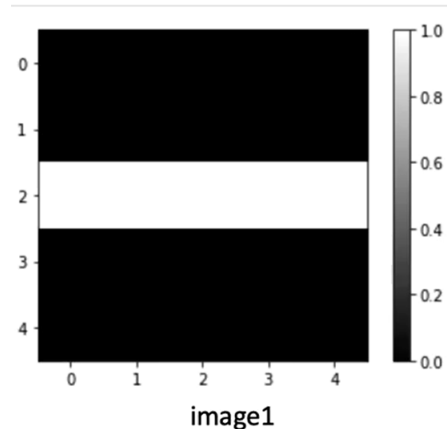


$$Z_2 = W_2 * X_1 + b_2$$

|   |   |   |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

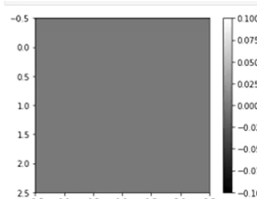


**$X_2$**



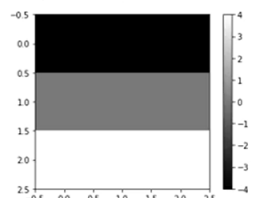
$$Z_0 = W_0 * X_2 + b_0$$

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



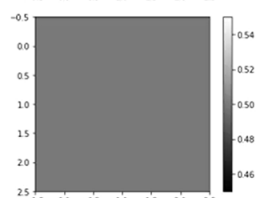
$$Z_1 = W_1 * X_2 + b_1$$

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



$$Z_2 = W_2 * X_2 + b_2$$

|   |   |   |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

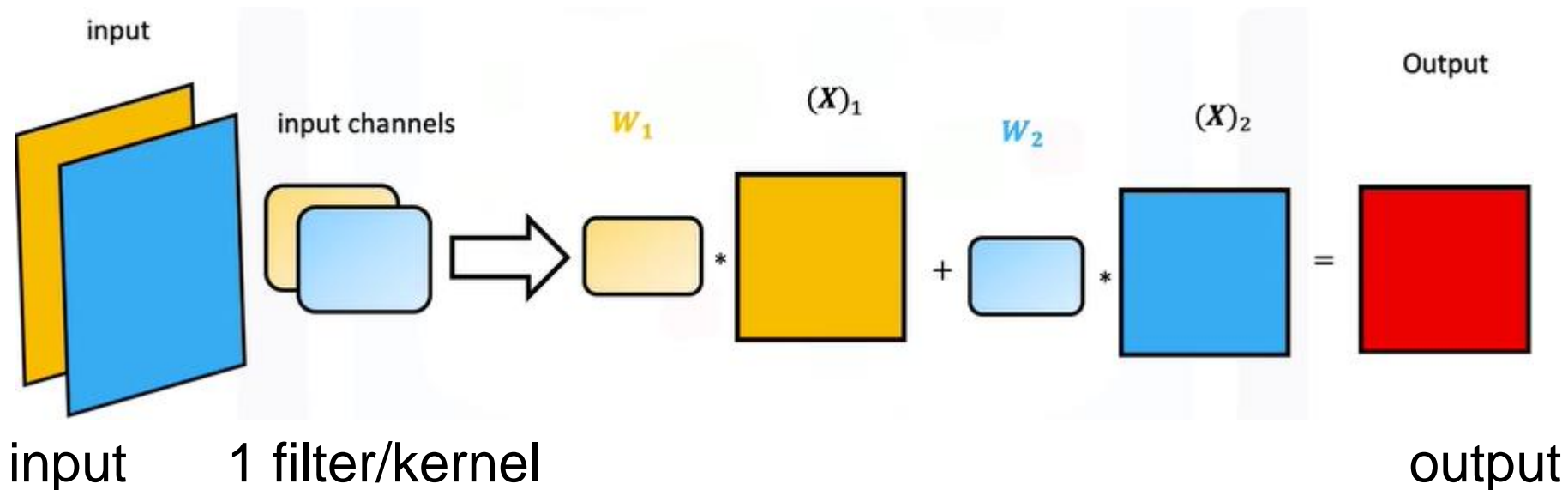


# Receptive Field and Weight Sharing

- Each neuron in a CONV layer has local 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
- Weight 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

# Convolution Example 1

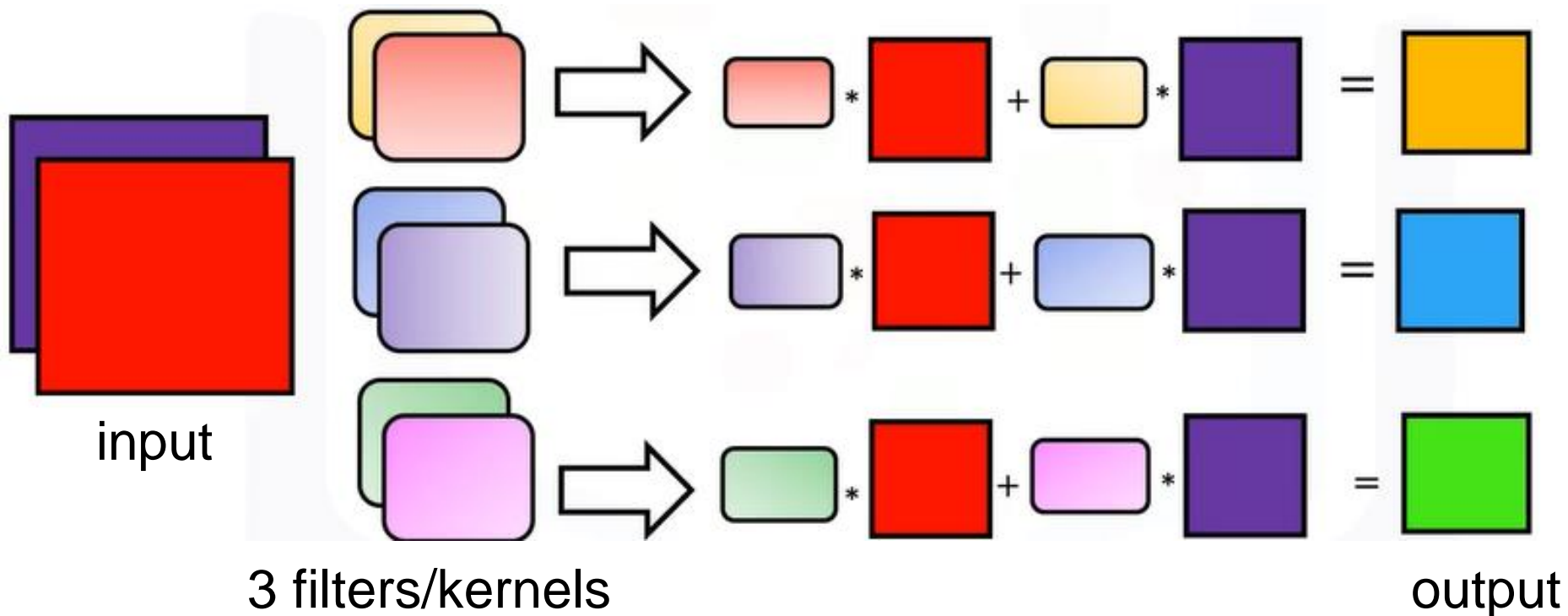
- `conv=nn.Conv2d(in_channels=2, out_channels=1, kernel_size=3)`
  - Pytorch code for a CONV layer with an input image with 2 channels (`in_channels=2`), 1  $3 \times 3$  filter, 1 output activation maps (`out_channels=1`).
  - The biases are assumed to be 0





# 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  $3 \times 3$  filters, 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 |

`Image4[1,0,:,:]`

`Image4[1,1,:,:]`

`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 |

Channel 1

Channel 2

|   |   |   |   |   |
|---|---|---|---|---|
| 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 |

|   |   |   |   |   |
|---|---|---|---|---|
| 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 |

`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  $3 \times 3$  filters

input image with 2 channels

# Convolution

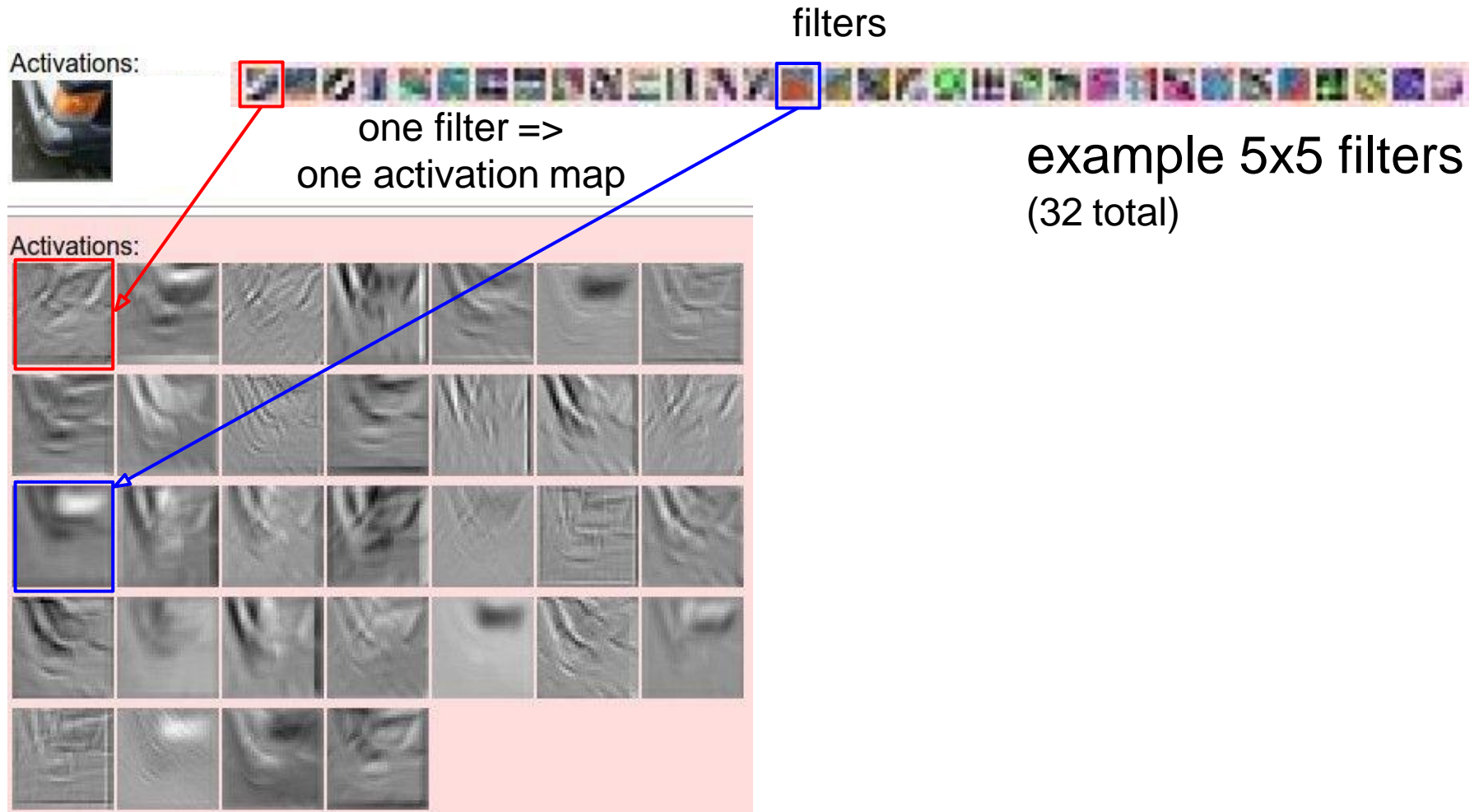
## Example 2: Output

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

$$\begin{aligned}
 (Z)_0 &= \begin{matrix} W_{0,0} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} * \begin{matrix} (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} \end{matrix} + \begin{matrix} W_{0,1} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} * \begin{matrix} (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} \end{matrix} \\
 &= \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} \\
 (Z)_1 &= \begin{matrix} W_{1,0} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} * \begin{matrix} (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} \end{matrix} + \begin{matrix} W_{1,1} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} * \begin{matrix} (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} \end{matrix} \\
 &= \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} \\
 (Z)_2 &= \begin{matrix} W_{2,0} \\ \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \end{matrix} * \begin{matrix} (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} \end{matrix} + \begin{matrix} W_{2,1} \\ \begin{bmatrix} 1 & 2 & -1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \end{matrix} * \begin{matrix} (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} \end{matrix} \\
 &= \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}
 \end{aligned}$$

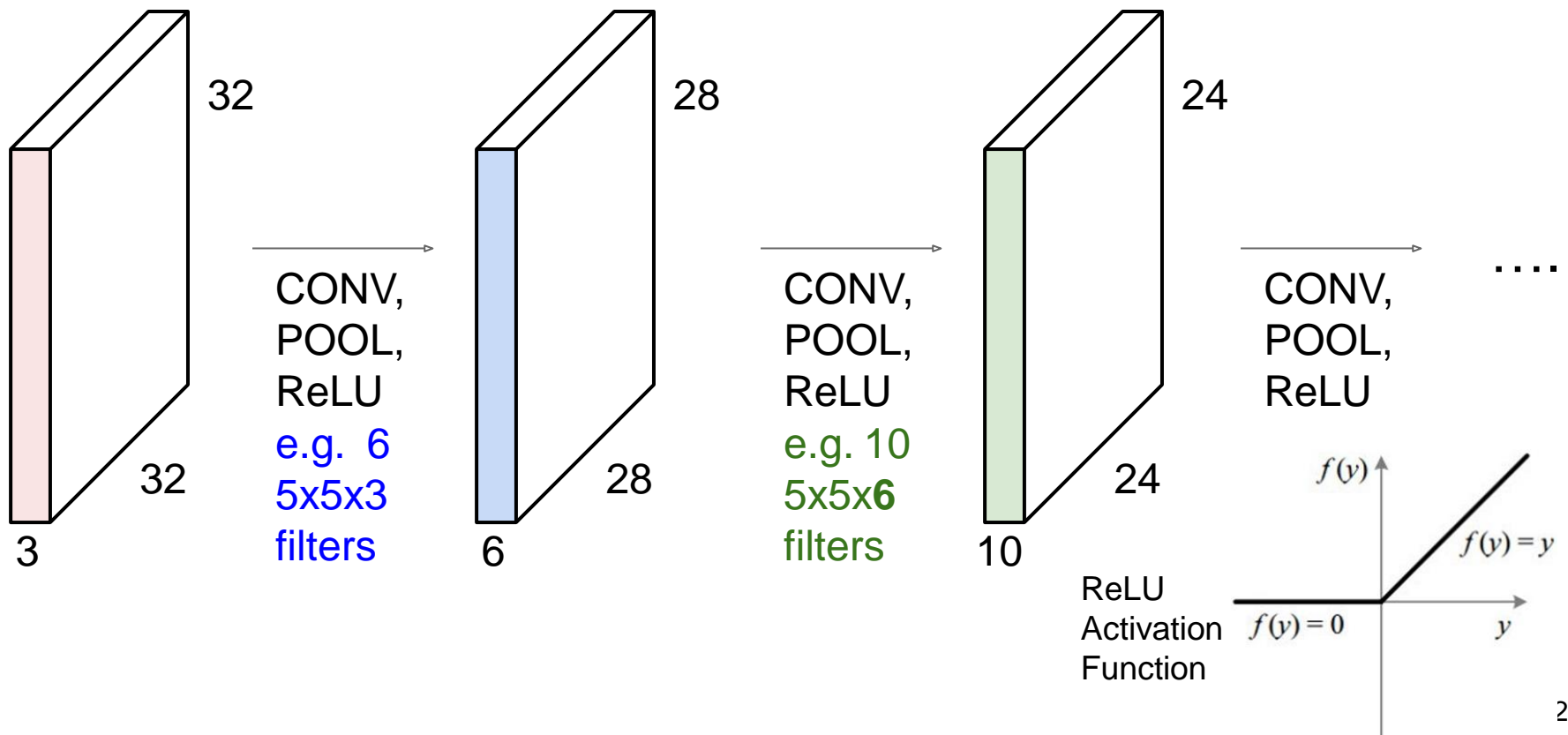
# Filters and Activation Maps

## Example



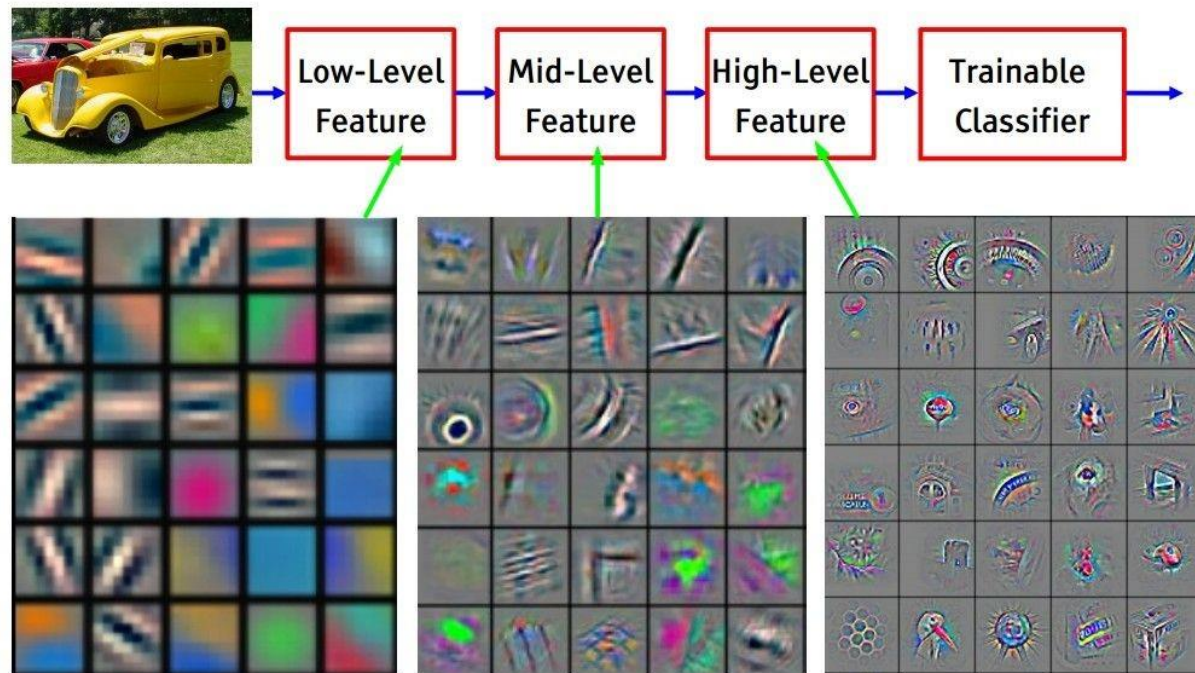
# Typical CNN Architecture

- 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



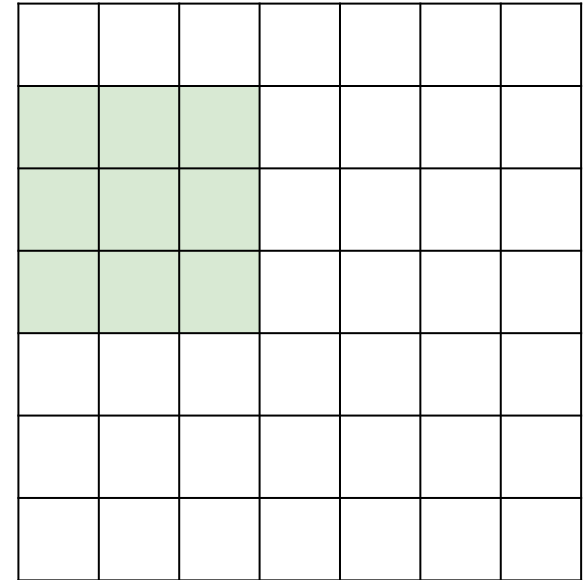
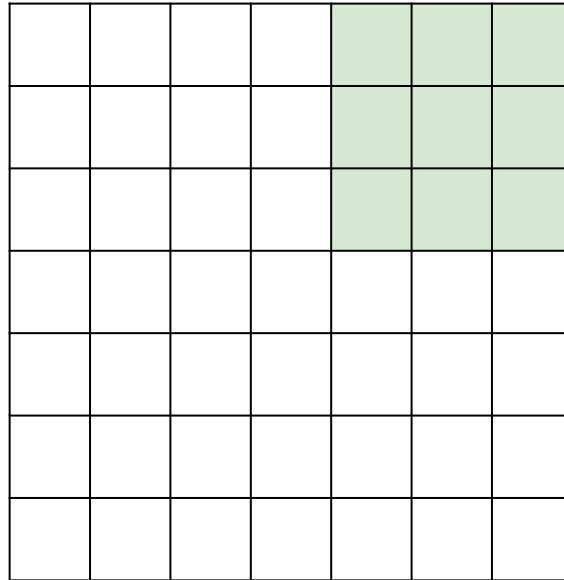
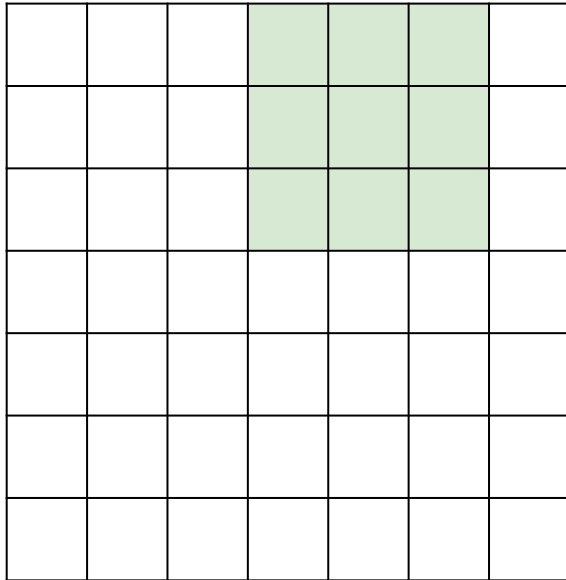
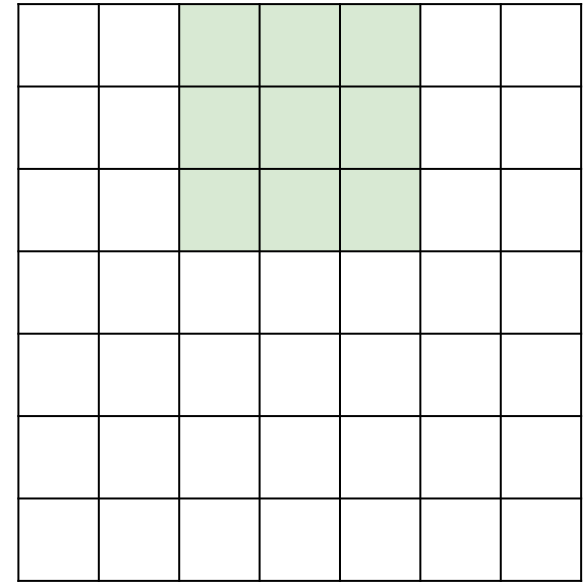
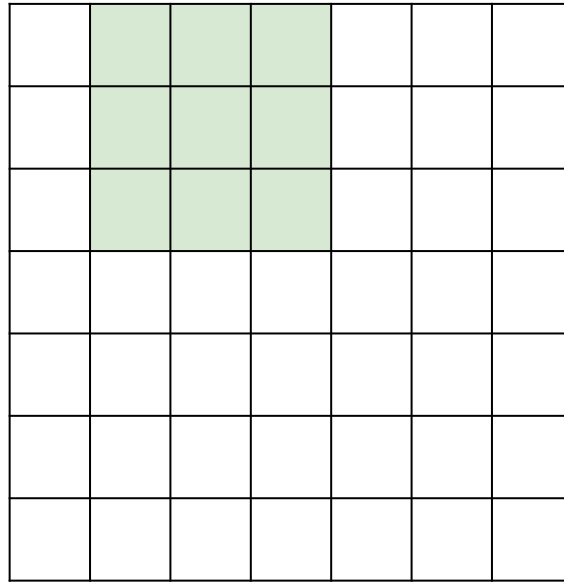
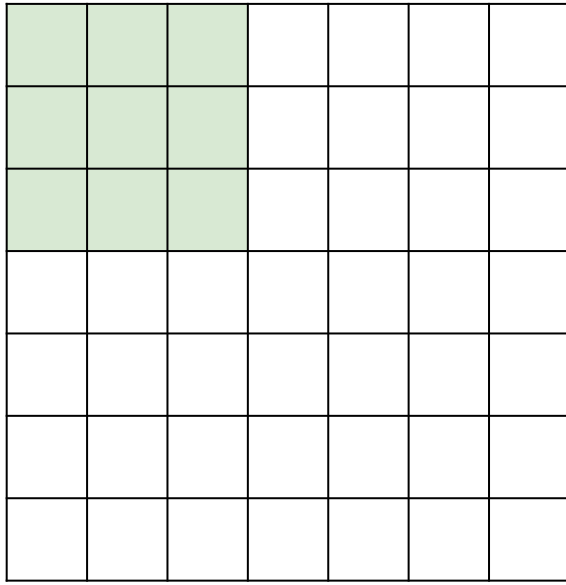
# Feature Hierarchy

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



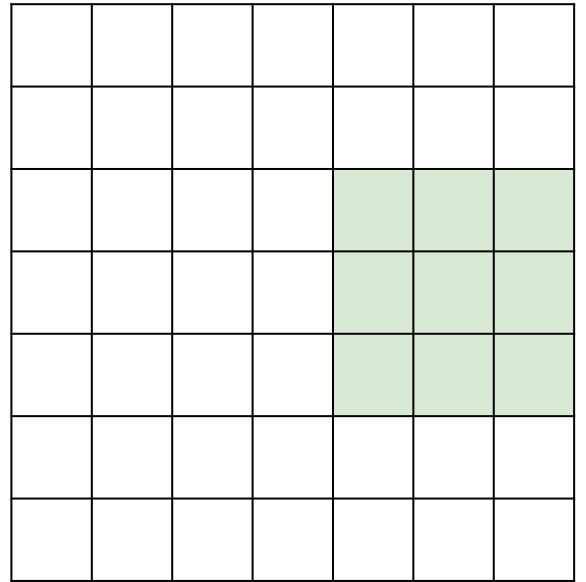
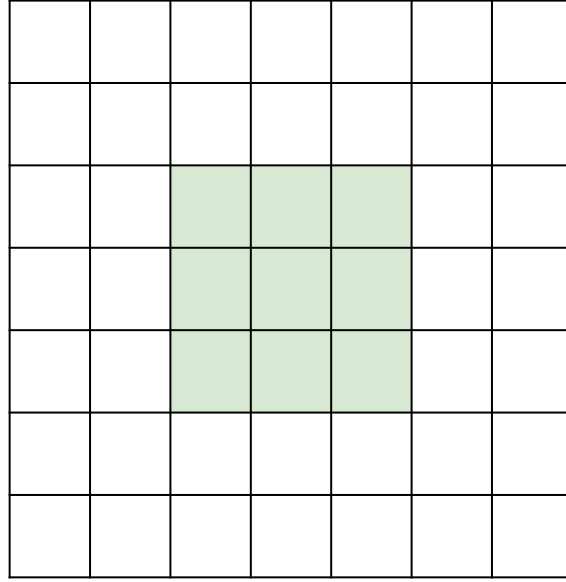
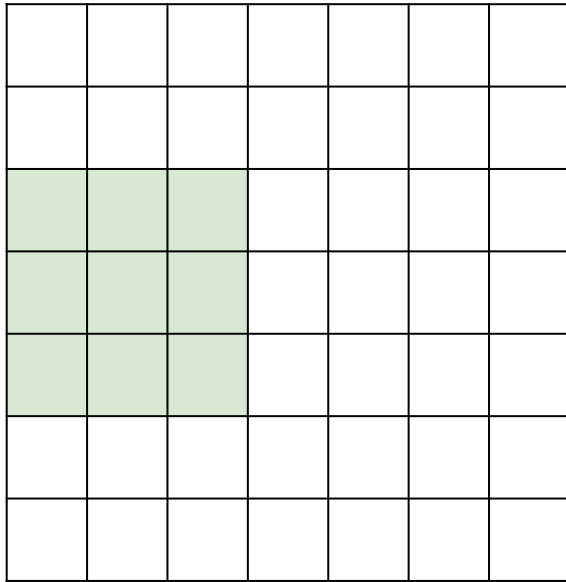
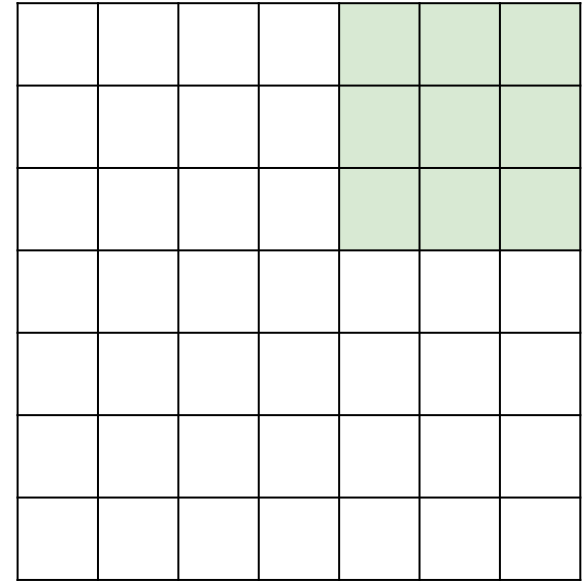
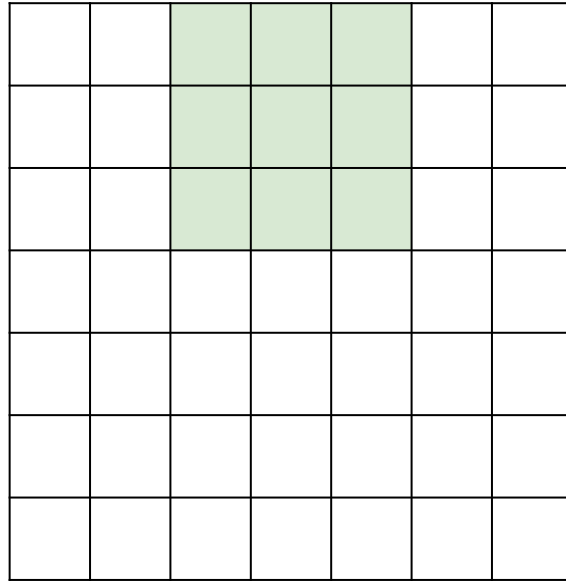
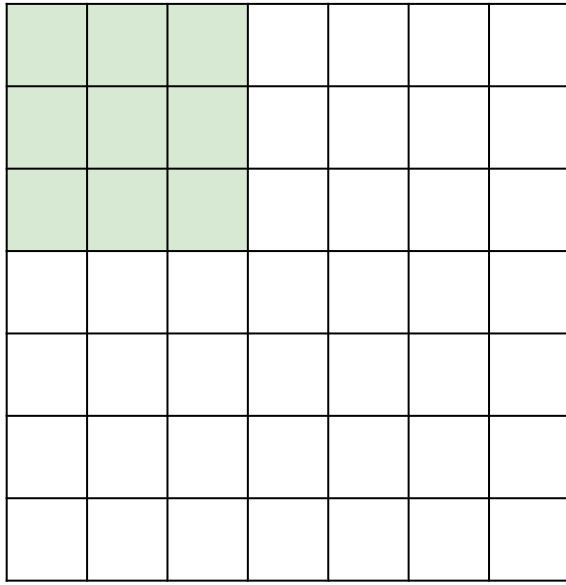
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

7x7 input, 3x3 filter, stride=1  $\Rightarrow$  output: 5x5 filter

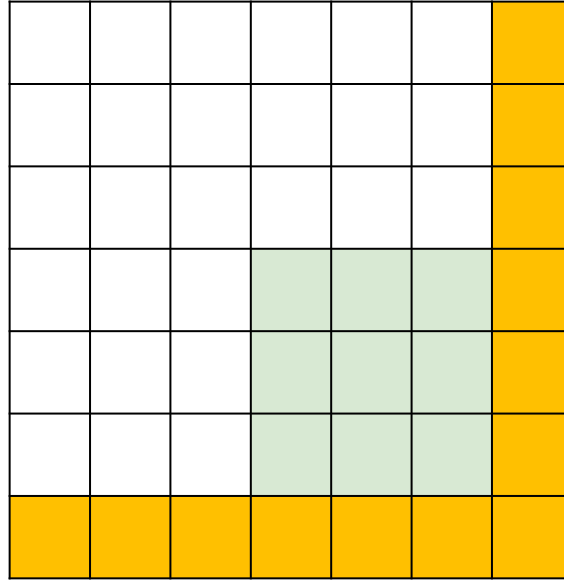
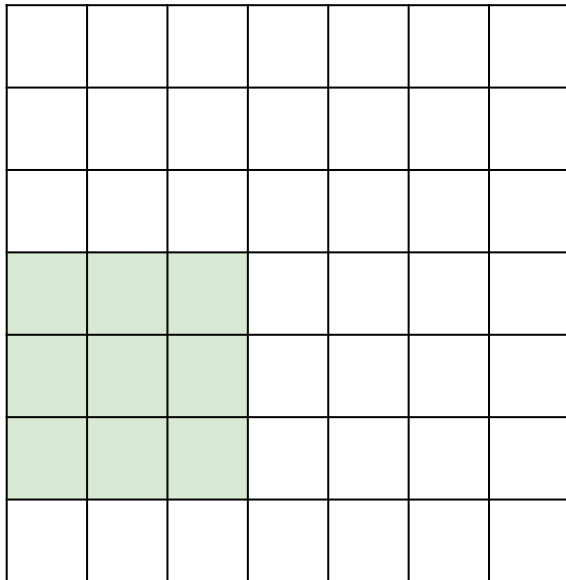
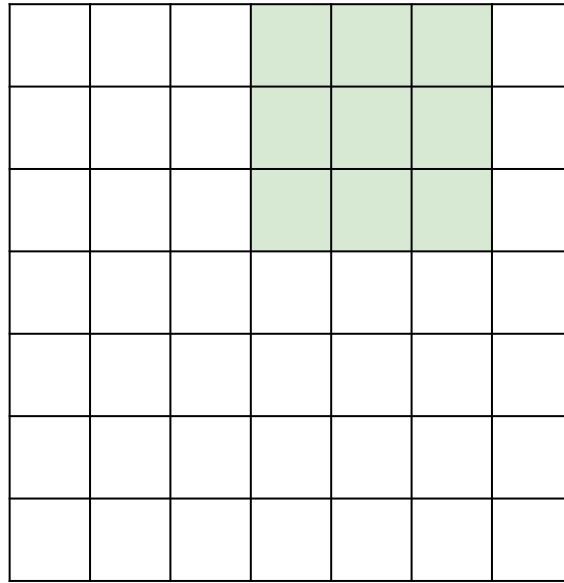
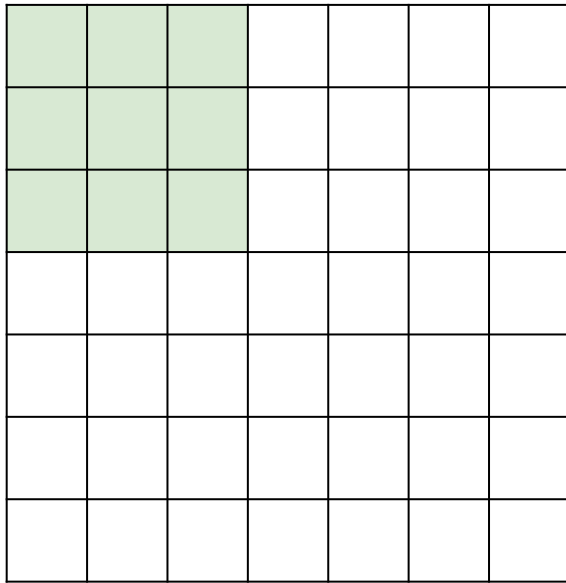




7x7 input, 3x3 filter, stride=2  $\Rightarrow$  output: 3x3 filter



7x7 input, 3x3 filter, stride=3  $\Rightarrow$  output: ???



The rightmost  
and bottom  
columns are not  
processed!

# Solution: Add Padding

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

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 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 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 |   |   |   |   |   |   |   |   |
| 0 |   |   |   |   |   |   |   |   |
| 0 |   |   |   |   |   |   |   |   |
| 0 |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |

# 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  $P$ .

**Common settings:**

$K$  = (powers of 2, e.g. 32, 64, 128, 512)

- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 1, S = 1, P = 0$

- 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.
- It is common to have square input shape, i.e.,  $W_1 = H_1 = N_1$ , but general rectangle shape size is possible
- A filter always has square shape, and always has the same depth  $D_1$  as the input 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:  $5 \times 5 \times 1$  ( $W_1 = H_1 = N_1 = 32, D_1 = 3$ )(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:  $W_2 = H_2 = N_2 = \frac{1}{S}(N_1 + 2P - F) + 1 = \frac{1}{1}(5 - 3) + 1 = 3$
  - 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...

|                        |                        |                        |   |   |
|------------------------|------------------------|------------------------|---|---|
| 1<br><small>x1</small> | 1<br><small>x0</small> | 1<br><small>x1</small> | 0 | 0 |
| 0<br><small>x0</small> | 1<br><small>x1</small> | 1<br><small>x0</small> | 1 | 0 |
| 0<br><small>x1</small> | 0<br><small>x0</small> | 1<br><small>x1</small> | 1 | 1 |
| 0                      | 0                      | 1                      | 1 | 0 |
| 0                      | 1                      | 1                      | 0 | 0 |

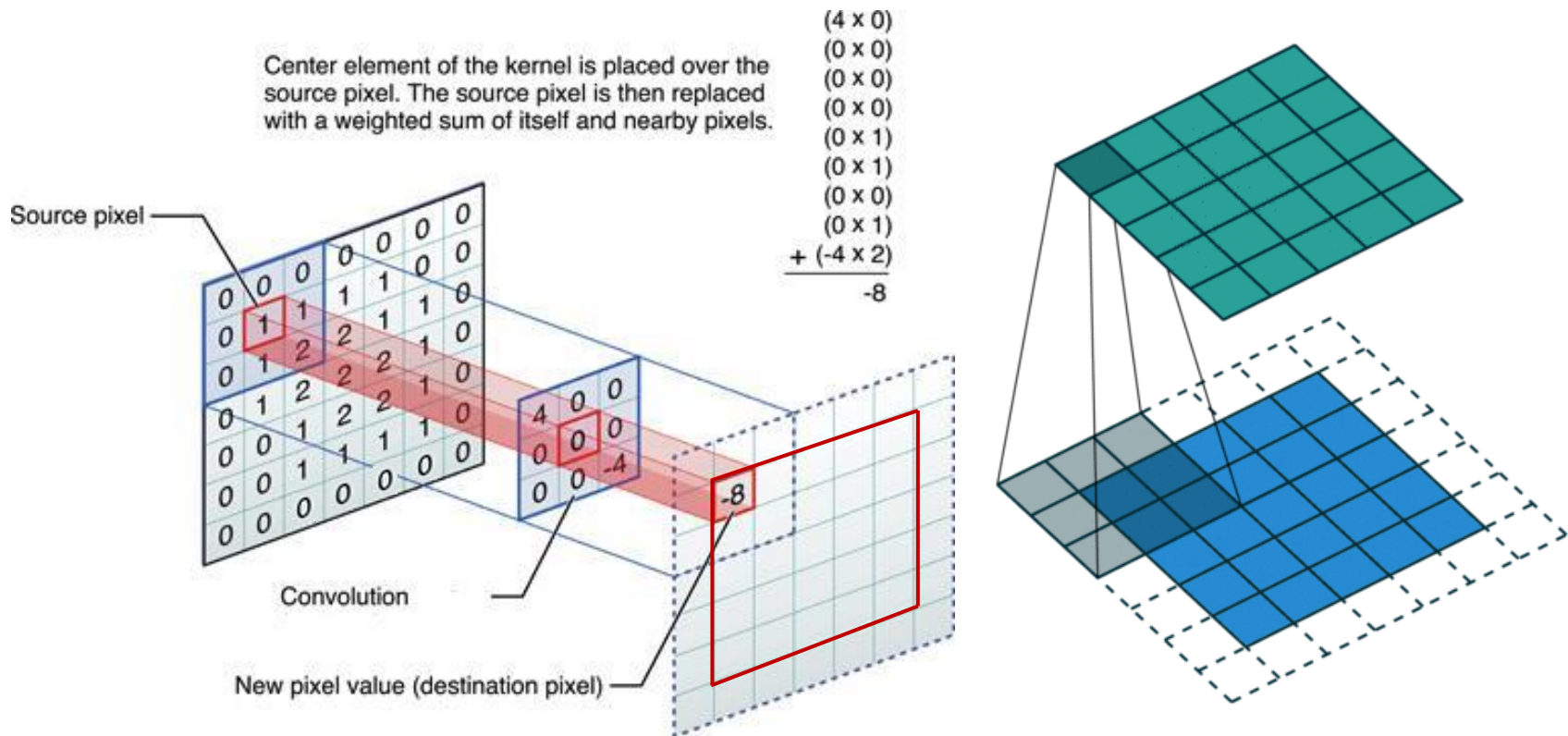
Image

|   |  |  |
|---|--|--|
| 4 |  |  |
|   |  |  |
|   |  |  |

Convolved  
Feature

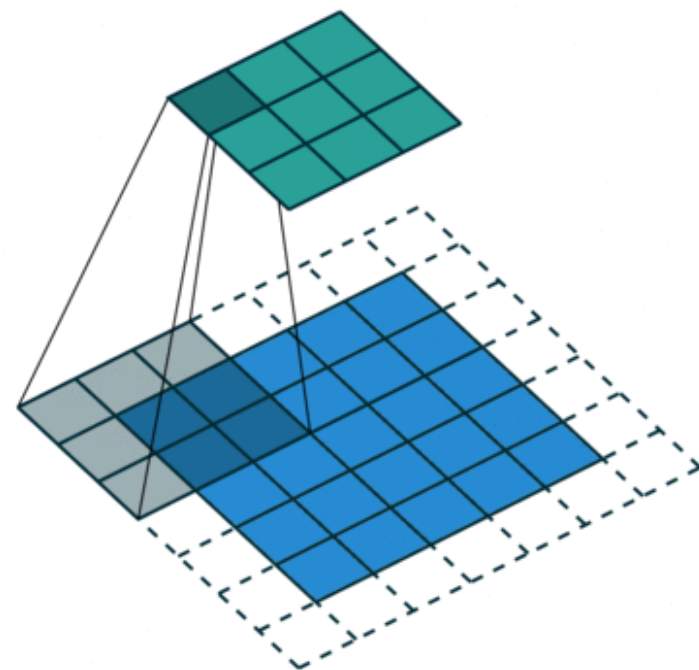
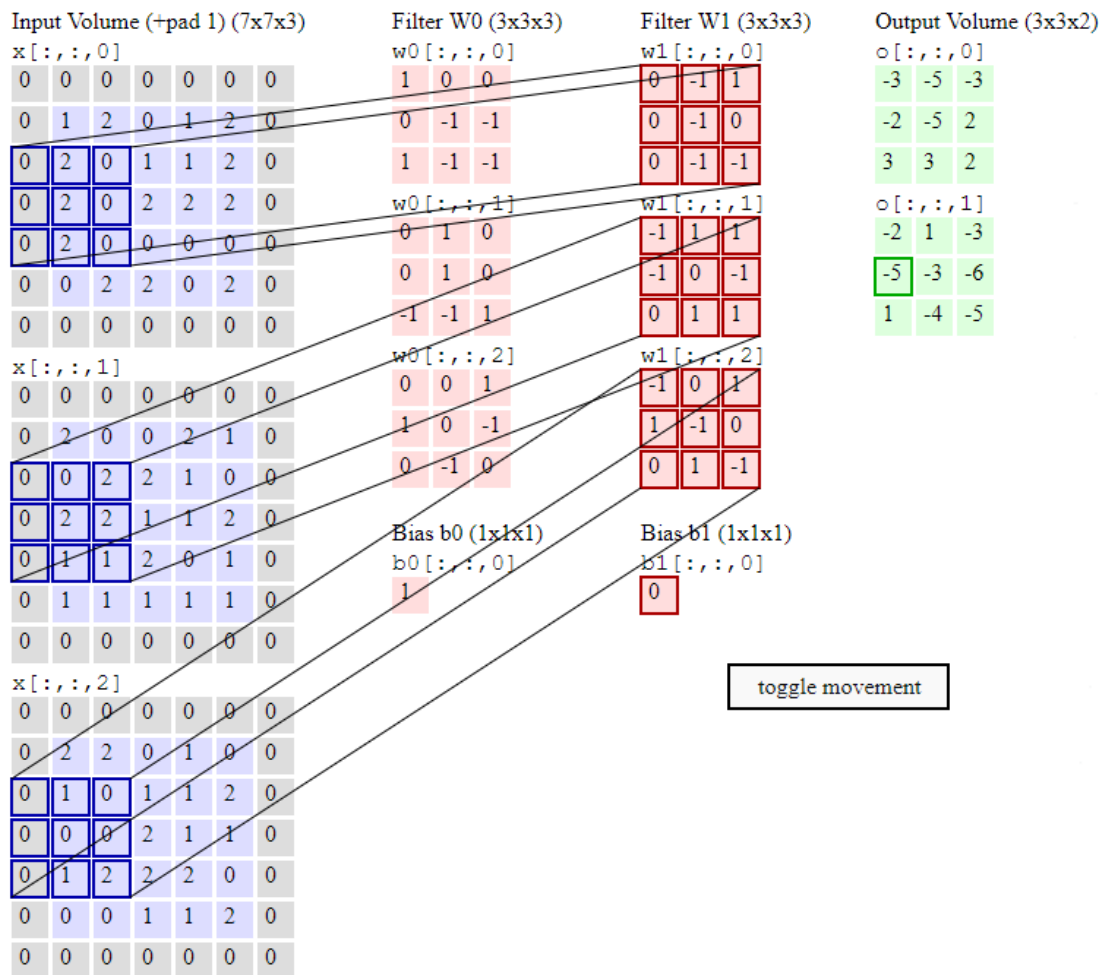
# CONV Example 2: Same Padding

- 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 ( $5 \times 5$ )



# CONV Example 3: Stride $S = 2$

- Input volume:  $5 \times 5 \times 3$
- $2 \times 3 \times 3 \times 3$  filters ( $K = 2, F = 3$ ) w. **stride  $S = 2$** , pad  $P = 1$
- Output volumes:  $2 \times 3 \times 3 \times 1$  (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)

|    |    |    |
|----|----|----|
| -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

+

-498

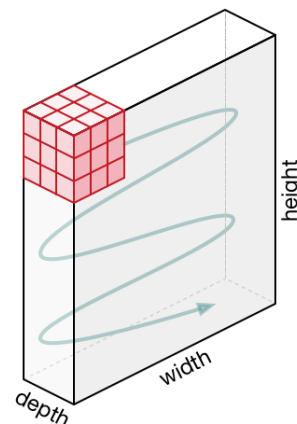
+

164

+

1 = -25

Bias = 1



Movement of the filter

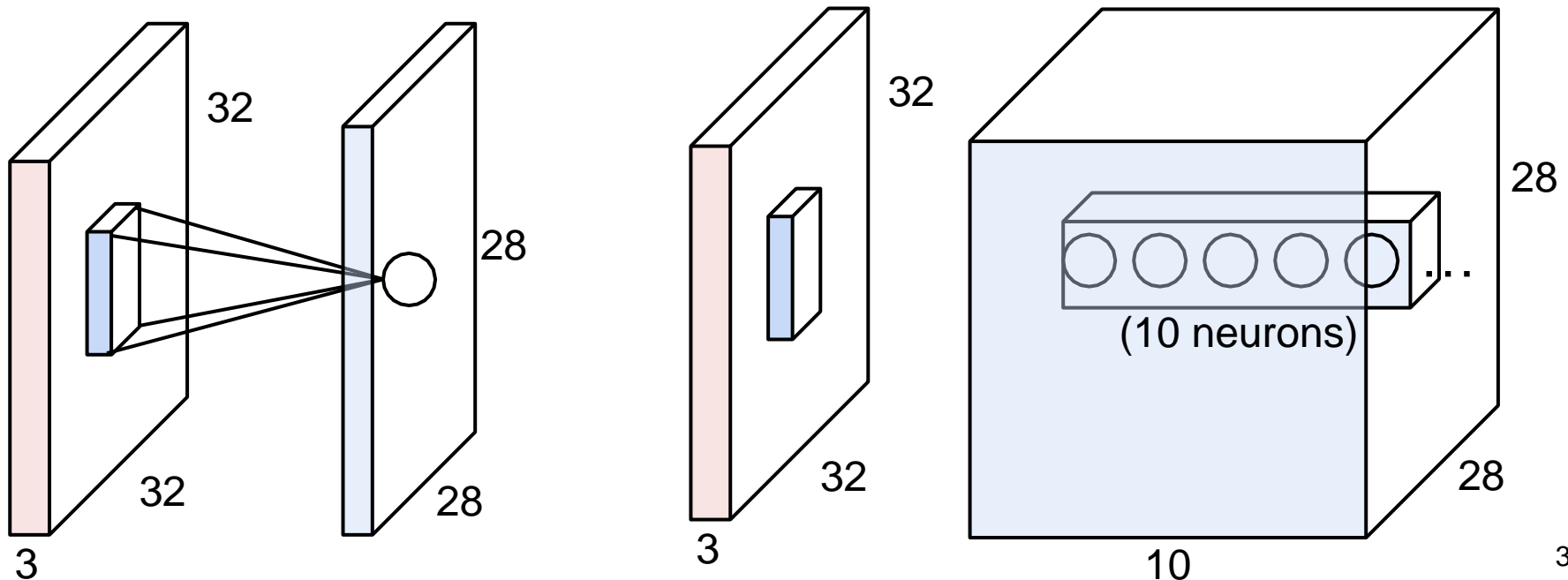
Output

|     |     |     |     |     |
|-----|-----|-----|-----|-----|
| -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:  $M \times N \times 3$  (since  $\frac{1}{1}(M + 2 * 1 - 3) + 1 = M$ ,  $\frac{1}{1}(N + 2 * 1 - 3) + 1 = N$ )

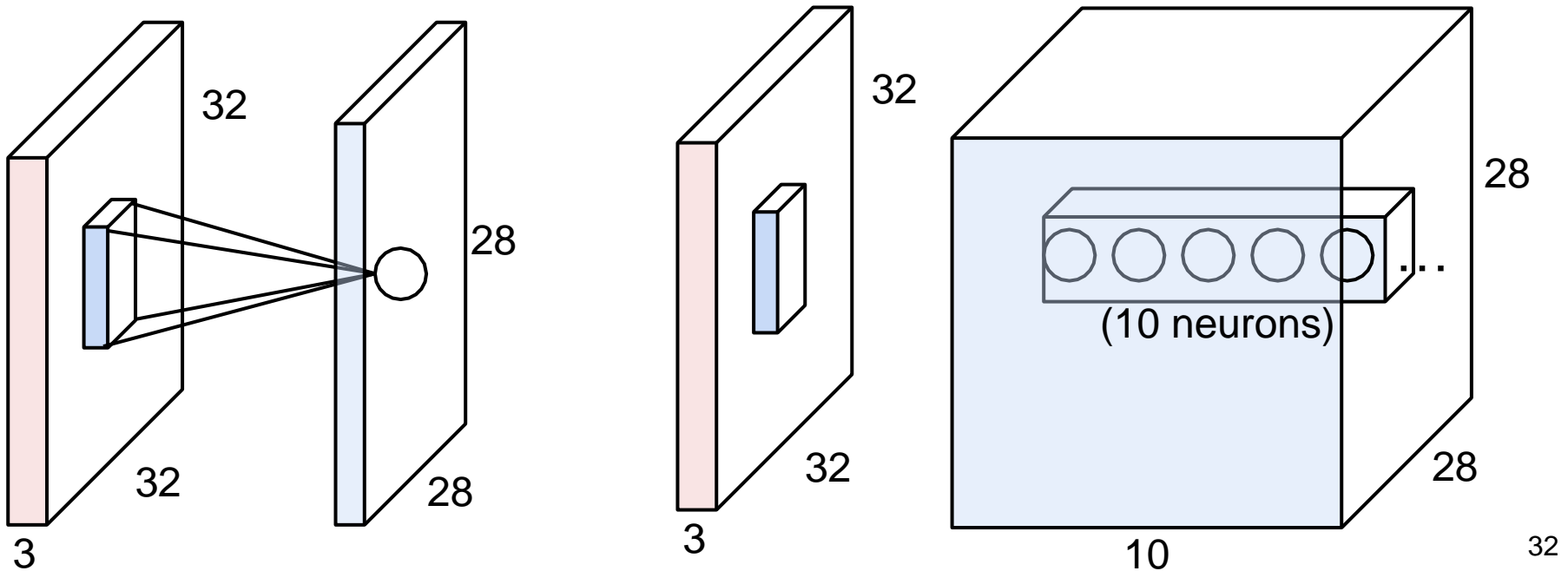
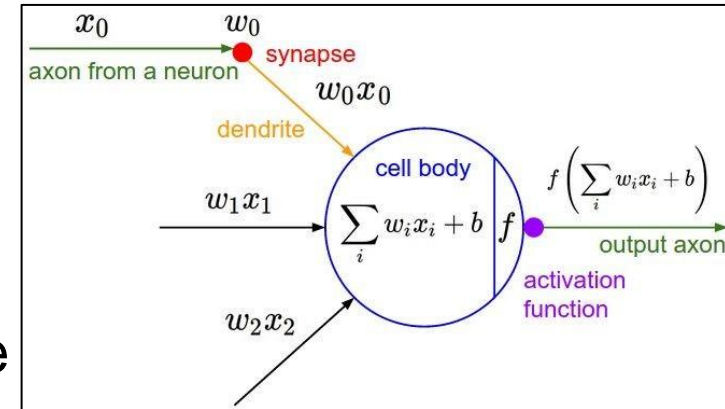
# CONV Example 5: Multiple Filters $K = 10$

- 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:  $W_2 = H_2 = N_2 = \frac{1}{S}(N_1 + 2P - F) + 1 = \frac{1}{1}(32 - 5) + 1 = 28$
  - 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 5: Neuron View

- One activation map is a  $28 \times 28$  sheet of neuron outputs.
- With 10 filters, the CONV layer consists of neurons arranged in a 3D grid ( $28 \times 28 \times 10$ ).
  - For each  $5 \times 5$  patch of the input, there are 10 different neurons all looking at it, each extracting different features

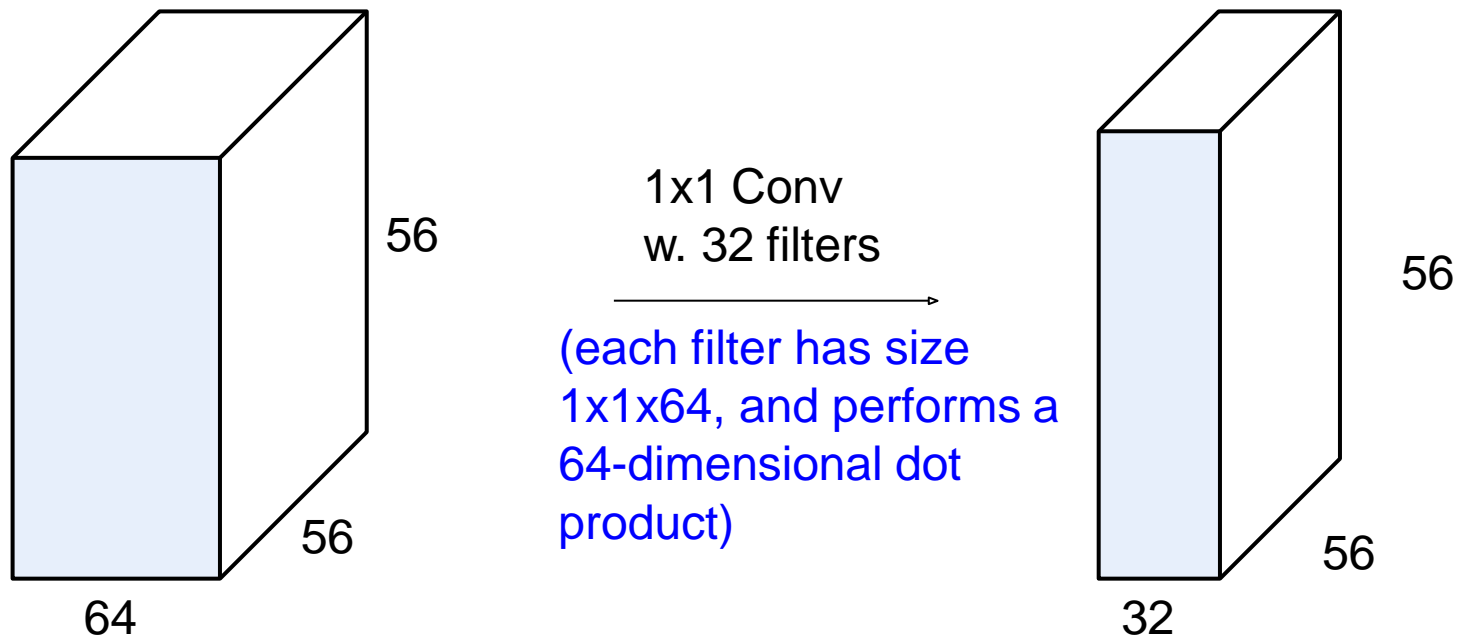


# CONV Example 6: Pad $P = 2$

- 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:  $W_2 = H_2 = N_2 = \frac{1}{S}(N_1 + 2P - F) + 1 = \frac{1}{1}(32 + 2 * 2 - 5) + 1 = 32$
  - 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

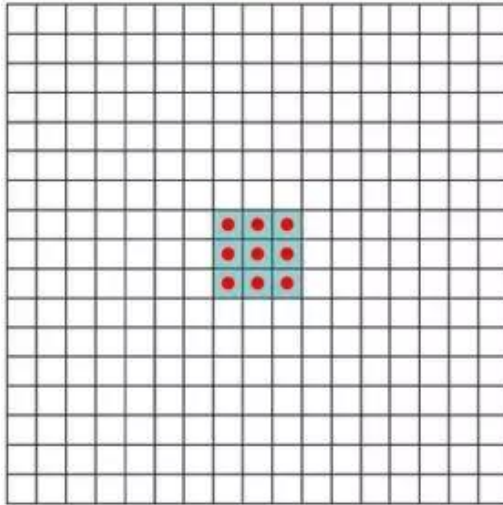
# CONV Example 7: $1 \times 1$ Filter

- 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:  $W_2 = H_2 = N_2 = \frac{1}{S}(N_1 + 2P - F) + 1 = \frac{1}{1}(56 - 1) + 1 = 56$
  - 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
- A  $1 \times 1$  filter performs “mixing” of the 64 input channels, then applies a non-linear activation function

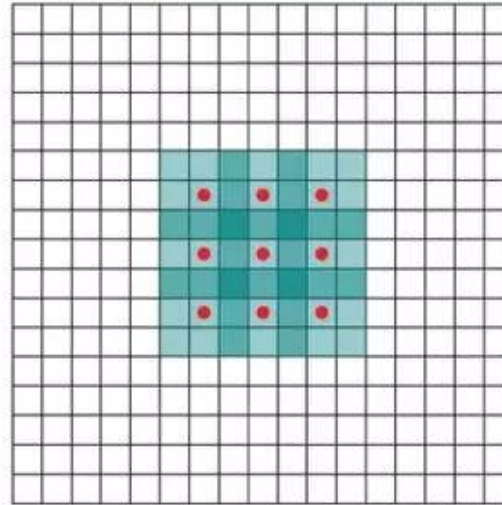


# Dilated Convolution

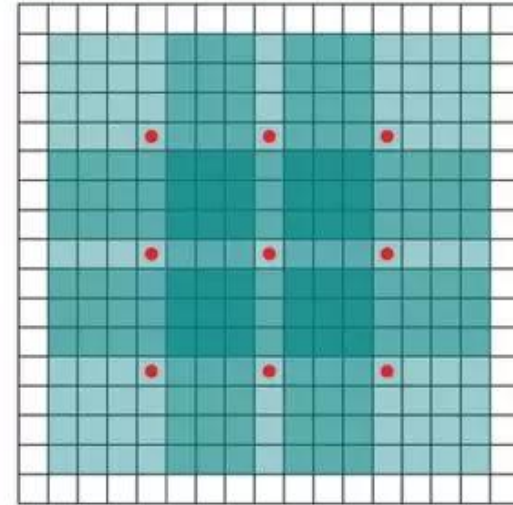
1 Dilated Convolution



2 Dilated Convolution

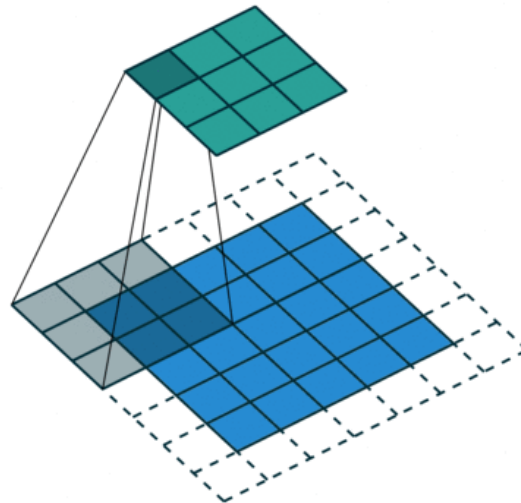


4 Dilated Convolution

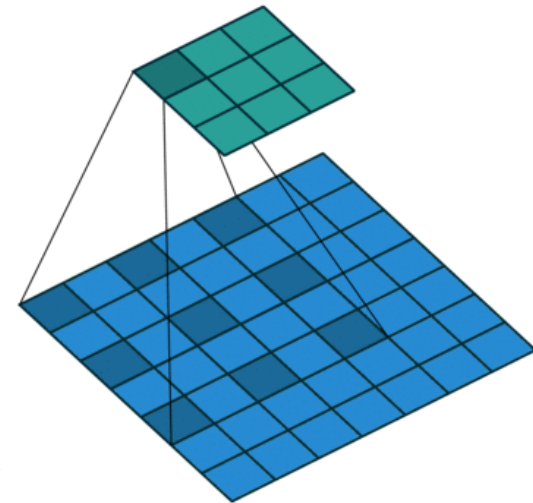


(a)

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



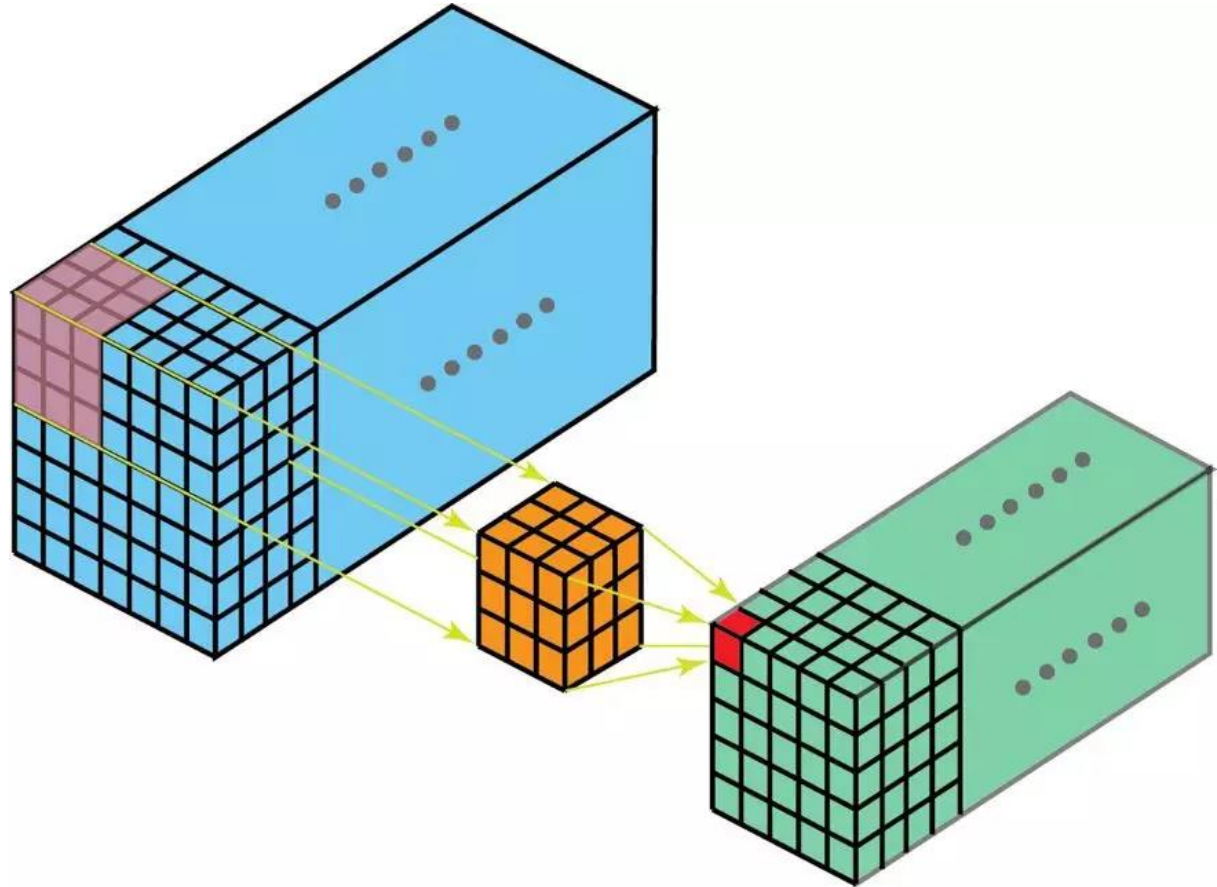
Regular convolution  
(1-dilated)



2-dilated  
convolution

# 3D Convolution

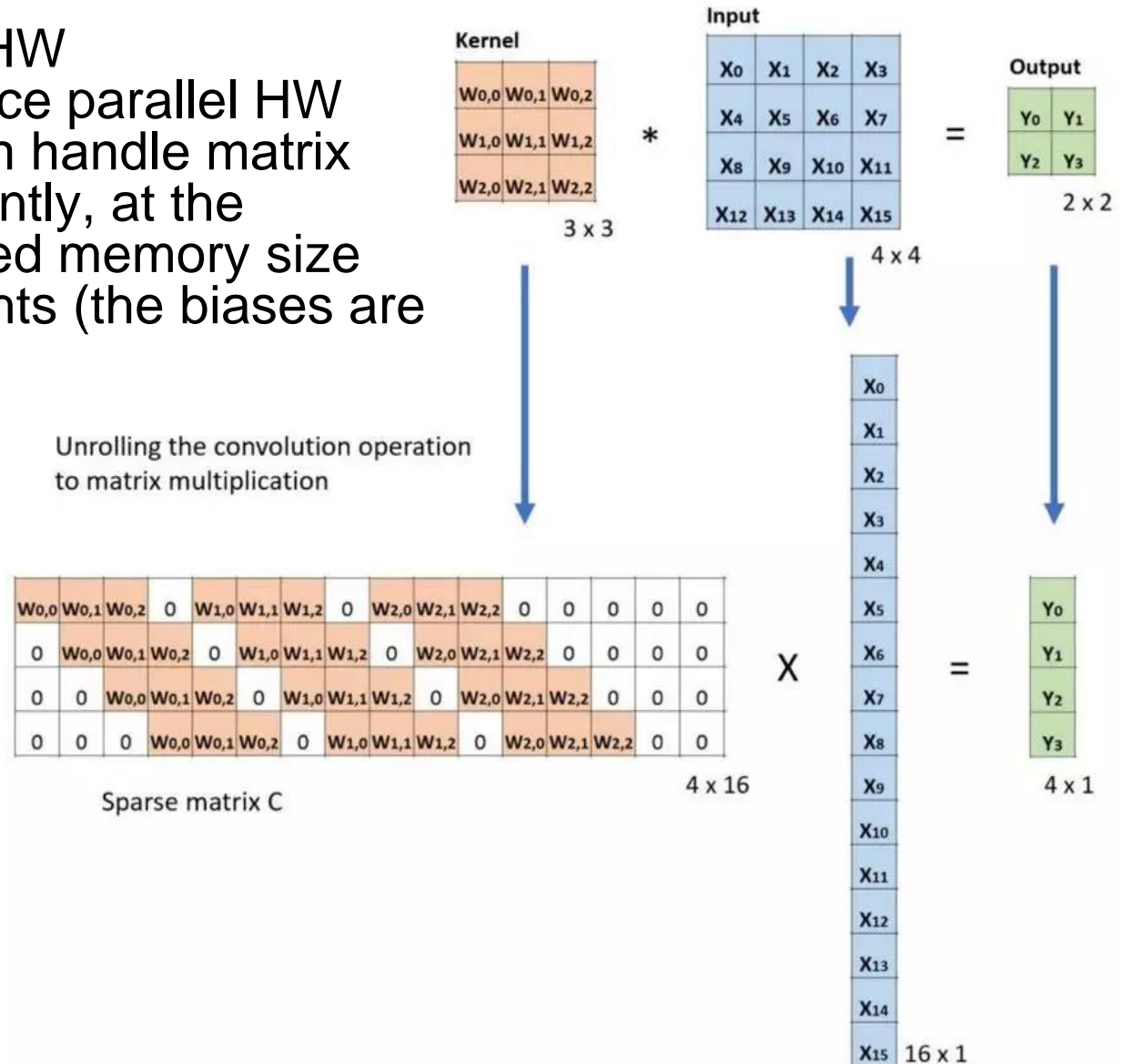
- 3D filter slides along all 3 axes (width, height, depth). Very computation intensive
- Useful for 3D images such as medical CT/MRI images, or Point Clouds from Lidar





# Converting Convolution to Matrix Multiplication

- Facilitates parallel HW implementation, since parallel HW (GPU, FPGA...) can handle matrix multiplication efficiently, at the expense of increased memory size for storing the weights (the biases are not shown in fig)



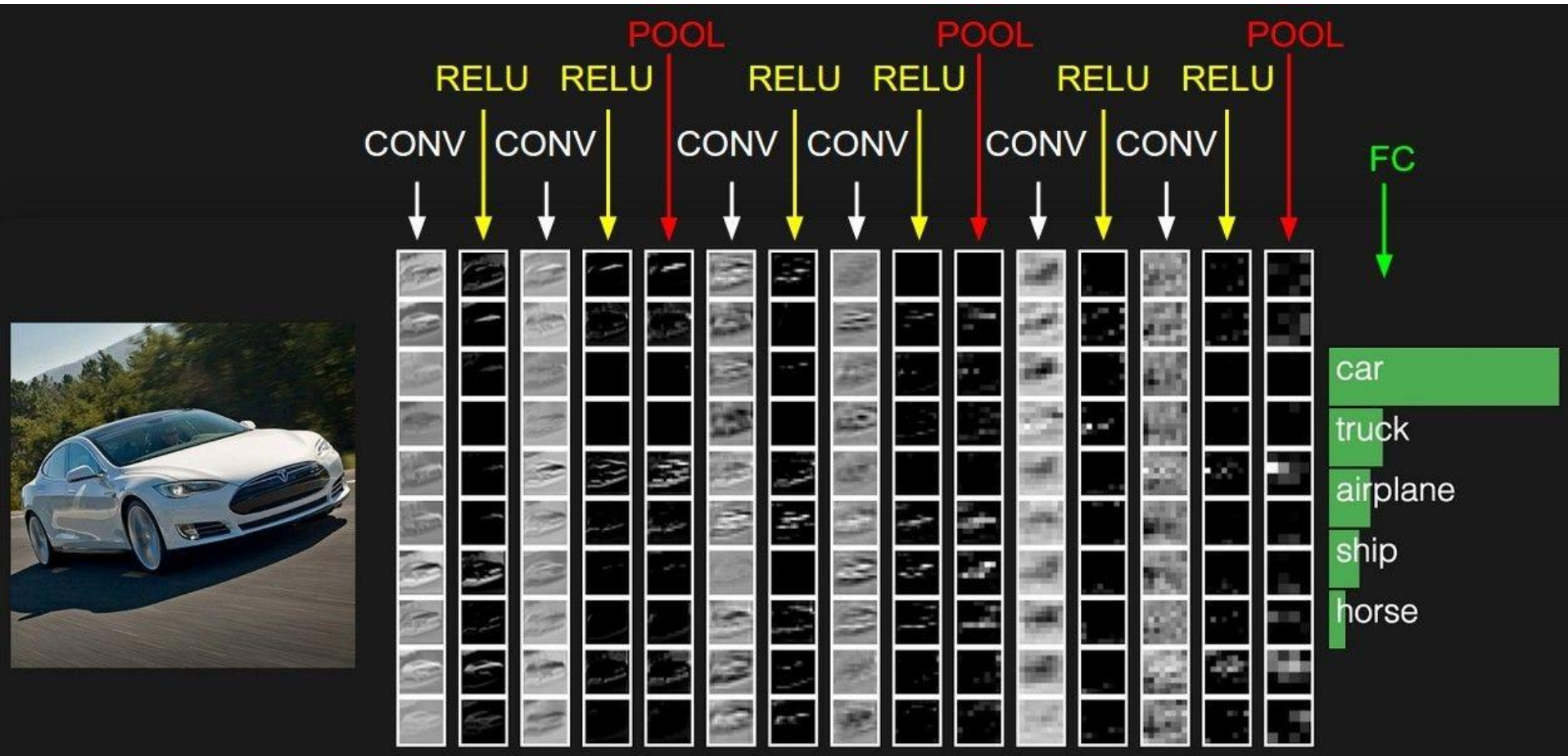
# Quiz

- What does a CONV layer w. filter volume  $K \times K \times D_1$  equal to its input volume  $N \times N \times D_1$ , i.e.,  $K = D$ , w. stride  $S = 1$  and no pad look like?
- ANS: It consists of a single neuron (since  $F = N_1 \Rightarrow N_2 = \frac{1}{S}(N_1 + 2P - F) + 1 = 1$ )

# Outline

- CNN Convolution layers
- Pooling and Fully-Connected layers
- CNN case studies
- RNNs

# Pooling and Fully-Connected Layers

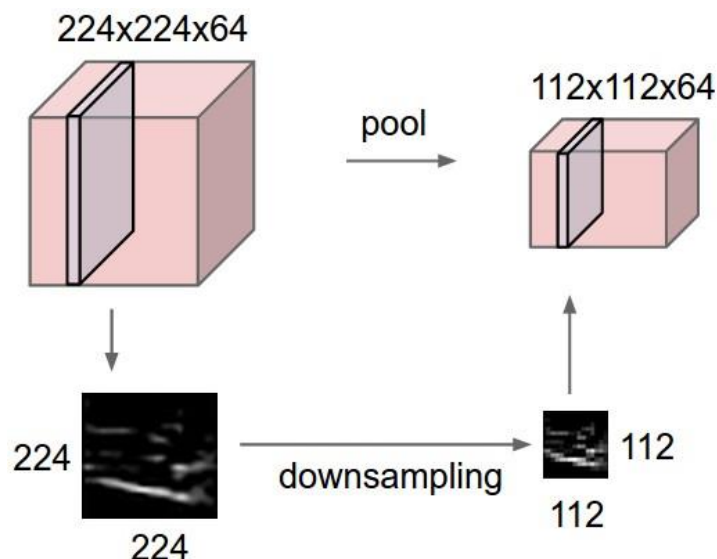
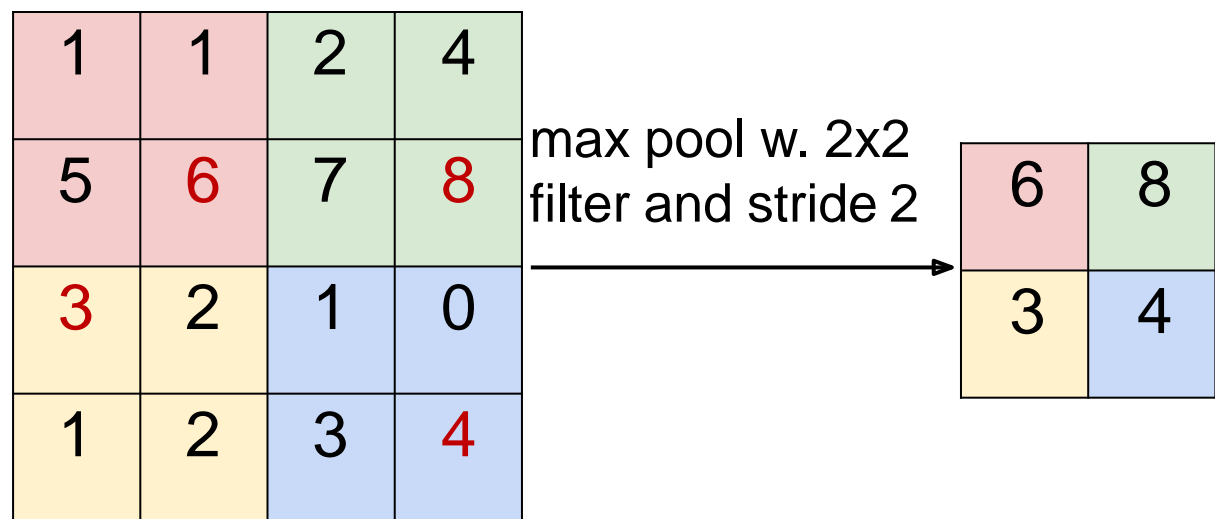


# Pooling (Sub-Sampling) Layer

- 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 always have the same depth  $D_1$  (unlike a CONV filter, which always has the same depth  $D_1$  as the input volume, and output volume has depth  $D_2 = K$ , the No. filters)
  - Common settings:  $F = 3, S = 2$ , or  $F = 3, S = 2$
- Example: pooling w. a  $2 \times 2$  filter w. stride  $S = 2$ , no pad
- Output volume:  $\frac{W_1}{2} \times \frac{H_1}{2} \times D_1$  (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
- Alternative: average pooling is less commonly used



# Overlapping Pooling

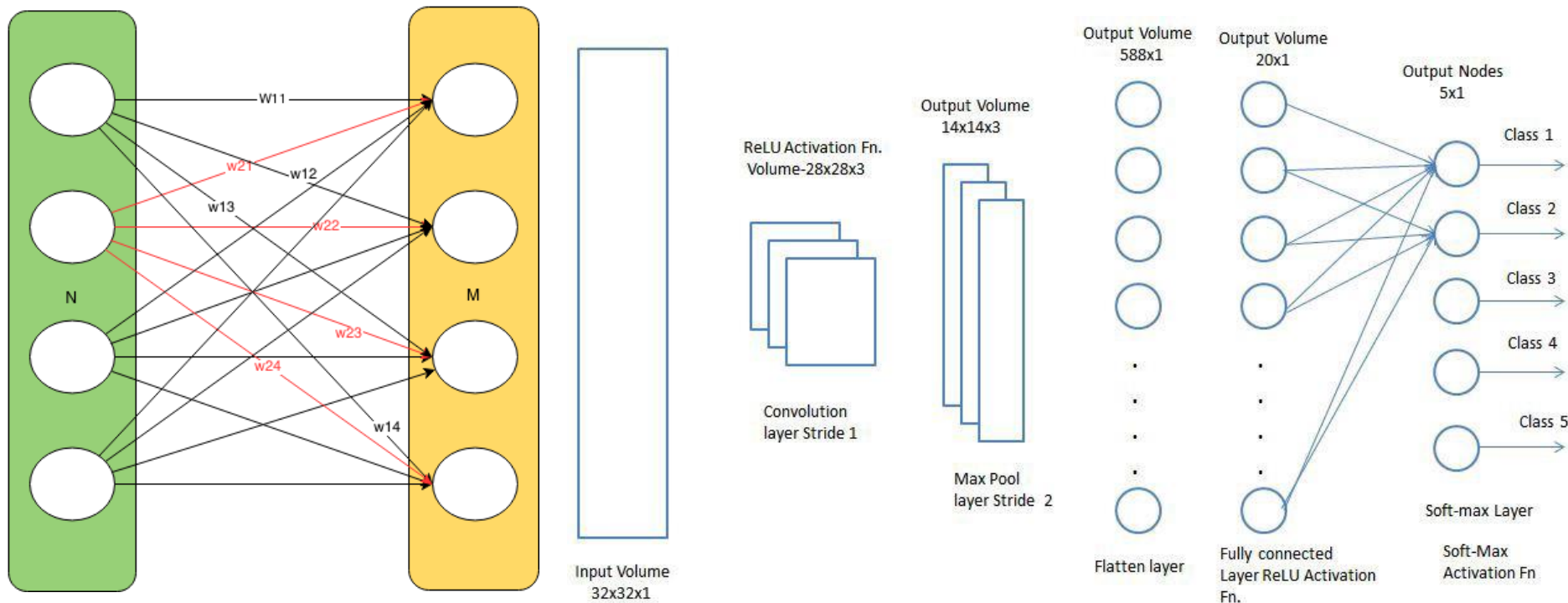
- Input volume:  $N \times N \times D_1$
- A  $3 \times 3$  filter w. stride  $S = 1$ , no pad
- Output volume:  $(N - 2) \times (N - 2) \times D_1$  (since  $\frac{1}{1}(N - 3) + 1 = N - 2$ )
- (This setting is uncommon. It is more common to have  $F = 3, S = 2$  for overlapping pooling)

|     |     |     |
|-----|-----|-----|
| 3.0 | 3.0 | 3.0 |
| 3.0 | 3.0 | 3.0 |
| 3.0 | 2.0 | 3.0 |

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | 3 | 2 | 1 | 0 |
| 0 | 0 | 1 | 3 | 1 |
| 3 | 1 | 2 | 2 | 3 |
| 2 | 0 | 0 | 2 | 2 |
| 2 | 0 | 0 | 0 | 1 |

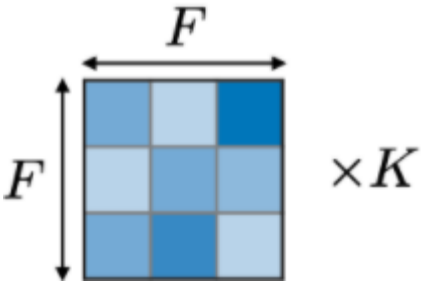
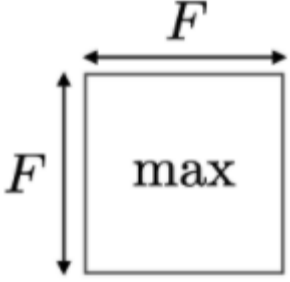
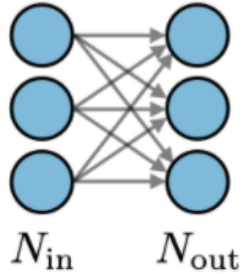
# FC Layer

- 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  $(N_{in} + 1) * N_{out}$





# No. Params in Each Layer

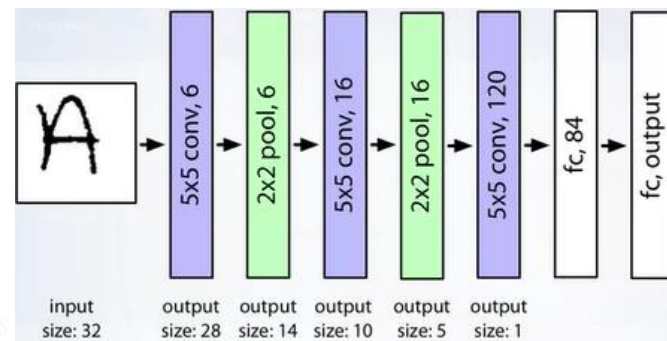
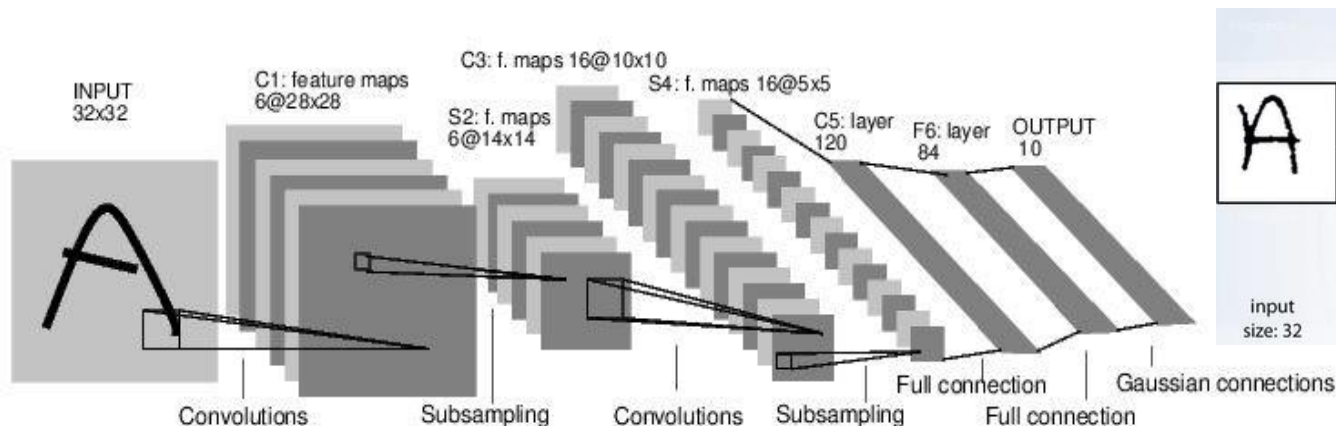
|             | CONV  | POOL  | FC  |
|-------------|---|---|---|
|             |  |  |  |
| Input size  | $N_1 \times N_1 \times D_1$   | $N_1 \times N_1 \times D_1$   | $N_{in}$  |
| Output Size | $N_2 \times N_2 \times K$   | $N_2 \times N_2 \times D_1$   | $N_{out}$   |
| No. params  | $(F * F * D_1 + 1) * K$   | 0   | $(N_{in} + 1) * N_{out}$  |

# Outline

- CNN Convolution layers
- Pooling and Fully-Connected layers
- CNN case studies
- RNNs

# Case Study: LeNet-5 [LeCun et al. 1998]

- Architecture is [CONV-POOL-CONV-POOL-CONV-FC-FC]
- Input image:  $32 \times 32 \times 1$  (grey-scale images of hand-written digits w. size  $32 \times 32$  pixels)
- Conv filters  $5 \times 5 \times 1$  w. stride 1; Subsampling (Pooling) layers were  $2 \times 2$  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  $\frac{1}{1}(32 - 5) + 1 = 28$ )
- Subsampling layer S2 maps from input volume  $28 \times 28 \times 1$  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  $\frac{1}{1}(14 - 5) + 1 = 10$ )
- Subsampling 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  $\frac{1}{1}(5 - 5) + 1 = 1$ )
  - Fig shows “Full Connection” before C5, but it is actually a Conv layer w. filter size equal to input size, followed by a “flatten” layer to turn it from  $1 \times 1 \times 120$  to a linear array of 120 neurons
- Subsequent FC layers F6 and Output have 84 and 10 neurons, respectively



# LeNet-5 Summary

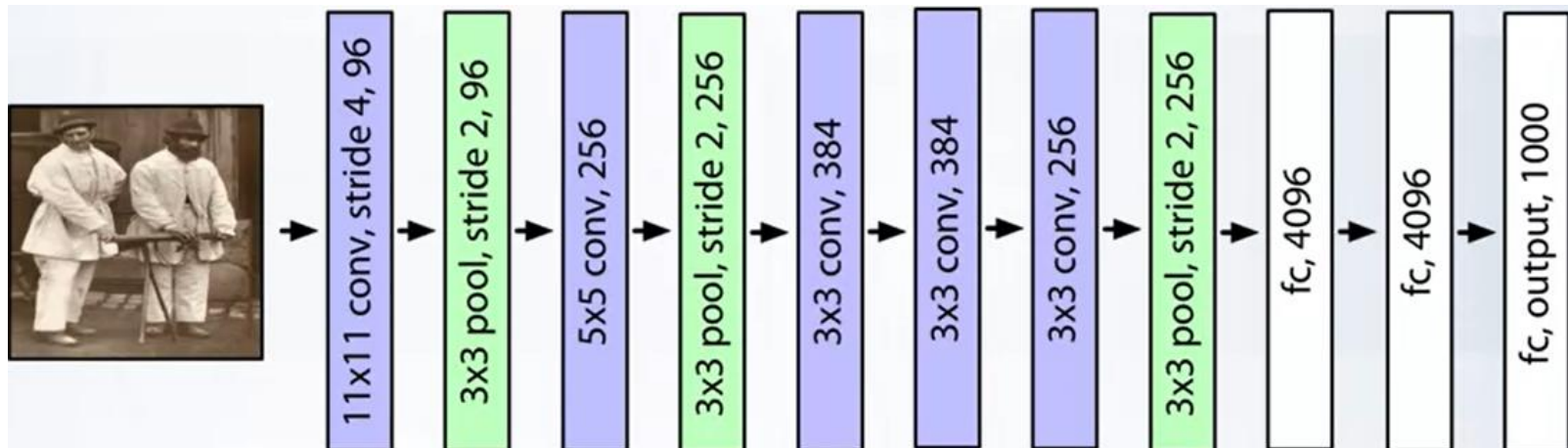
| Layer  | Layer Type | No. Filters | Filter Size  | Layer Size               | No. params |
|--------|------------|-------------|--------------|--------------------------|------------|
| Input  | Image      | -           | -            | $32 \times 32 \times 1$  | -          |
| C1     | Conv       | 6           | $5 \times 5$ | $28 \times 28 \times 6$  | 156        |
| S2     | Subsamp    | 6           | $2 \times 2$ | $14 \times 14 \times 6$  | 0          |
| C3     | Conv       | 16          | $5 \times 5$ | $10 \times 10 \times 16$ | 2416       |
| S4     | Subsamp    | 16          | $2 \times 2$ | $5 \times 5 \times 16$   | 0          |
| C5     | Conv+      | 120         | $5 \times 5$ | $1 \times 1 \times 120$  | 48120      |
|        | Flatten    | -           | —            | 120                      | 0          |
| S6     | FC         | -           | —            | 84                       | 10164      |
| Output | FC         |             |              | 10                       | 850        |

- C1: No. params:  $(F^2 D_1 + 1) * K = (5 * 5 * 1 + 1) * 6 = 156$
- C3: No. params:  $(5 * 5 * 6 + 1) * 16 = 2416$
- C5: No. params:  $(5 * 5 * 16 + 1) * 120 = 48120$
- S6: No. params:  $(120 + 1) * 84 = 10164$
- Output: No. params:  $(N_{in} + 1) * N_{out} = (84 + 1) * 10 = 850$
- Total No. params: 61706

# Case Study: AlexNet [Krizhevsky et al. 2012]

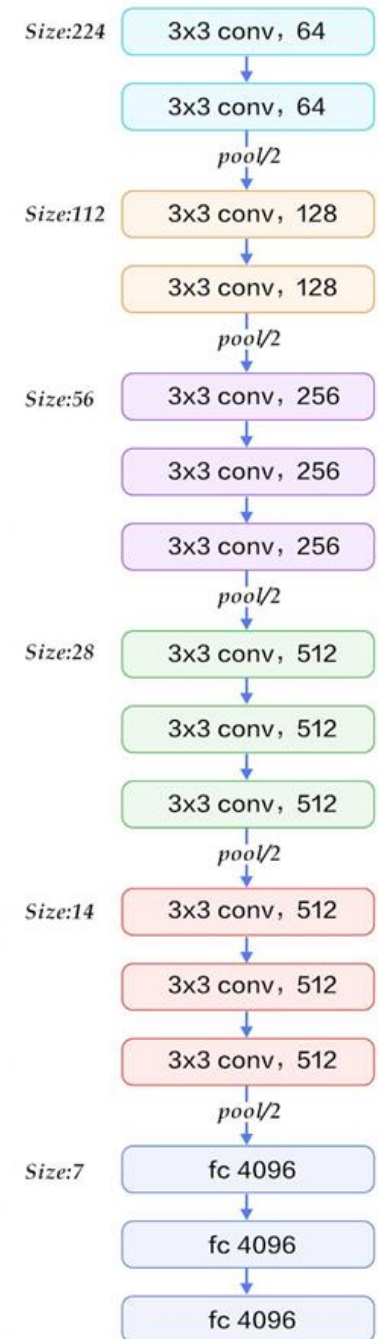
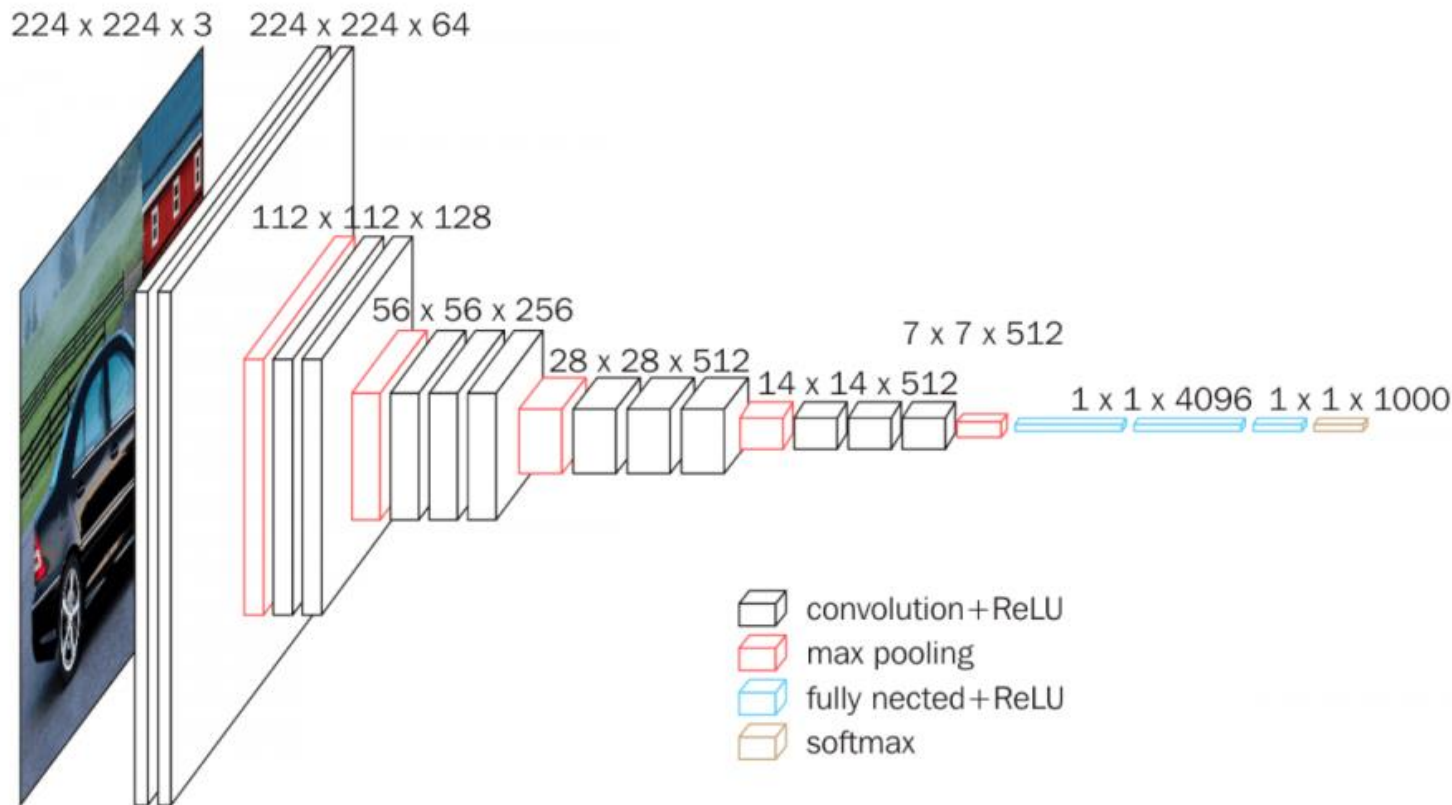
[227x227x3] INPUT  
 [55x55x96] **CONV1**: 96 11x11 filters w. stride 4, pad 0  
 [27x27x96] **MAX POOL1**: 3x3 filters w. stride 2  
 [27x27x96] **NORM1**: Normalization layer  
 [27x27x256] **CONV2**: 256 5x5 filters w. stride 1, pad 2  
 [13x13x256] **MAX POOL2**: 3x3 filters at stride 2  
 [13x13x256] **NORM2**: Normalization layer  
 [13x13x384] **CONV3**: 384 3x3 filters w. stride 1, pad 1  
 [13x13x384] **CONV4**: 384 3x3 filters w. stride 1, pad 1  
 [13x13x256] **CONV5**: 256 3x3 filters w. stride 1, pad 1  
 [6x6x256] **MAX POOL3**: 3x3 filters w. stride 2  
 [4096] **FC6**: 4096 neurons  
 [4096] **FC7**: 4096 neurons  
 [1000] **FC8**: 1000 neurons (class scores)  
 ImageNet top 5 error: 15.4%

- Input image:  $227 \times 227 \times 3$
- 1<sup>st</sup> layer (CONV1): 96  $11 \times 11$  filters w. stride  $S = 4$ , w. ReLU activation function
- Output volume:  $55 \times 55 \times 96$  (since  $\frac{1}{4}(227 - 11) + 1 = 55$ )
  - No. params:  $(11 * 11 * 3 + 1) * 96 = 35K$
- 2<sup>nd</sup> layer (POOL1):  $3 \times 3$  filters w. stride  $S = 2$  (overlapping)
- Output volume:  $27 \times 27 \times 96$  (since  $\frac{1}{2}(55 - 3) + 1 = 27$ )
  - No. params: 0
- ...
- Total No. params: 60M



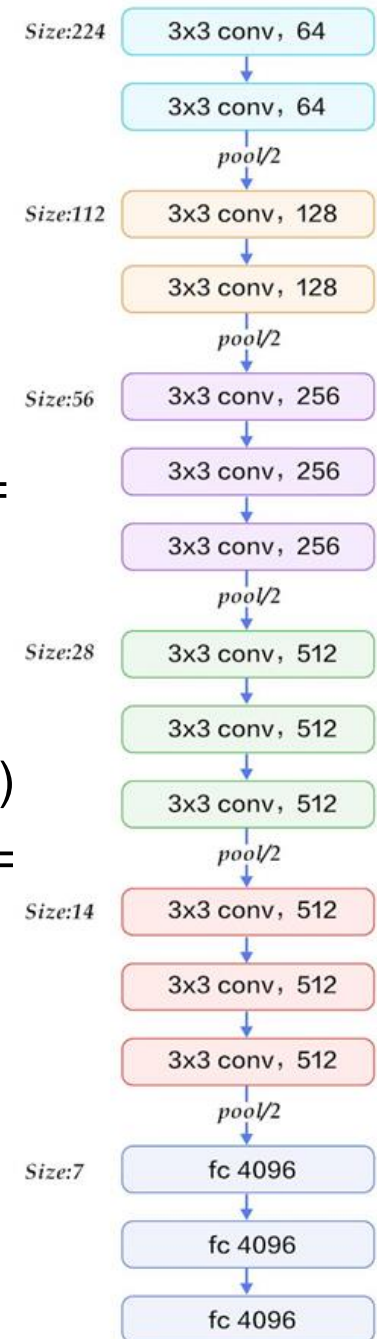
# Case Study: VGGNet [Simonyan and Zisserman, 2014]

- We show the best performing variant VGG-16



# VGG-16 Details

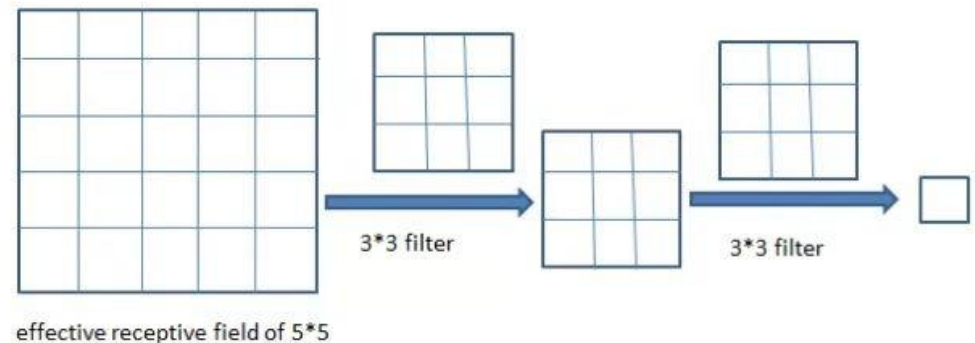
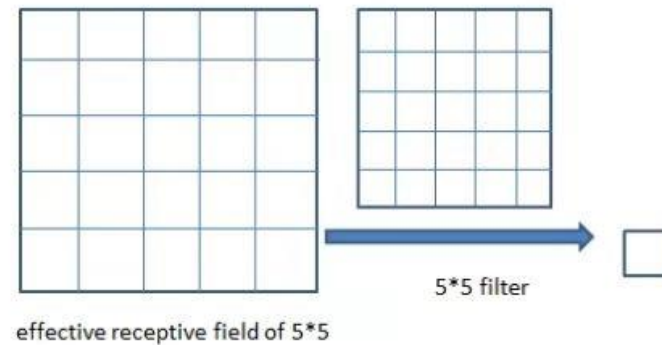
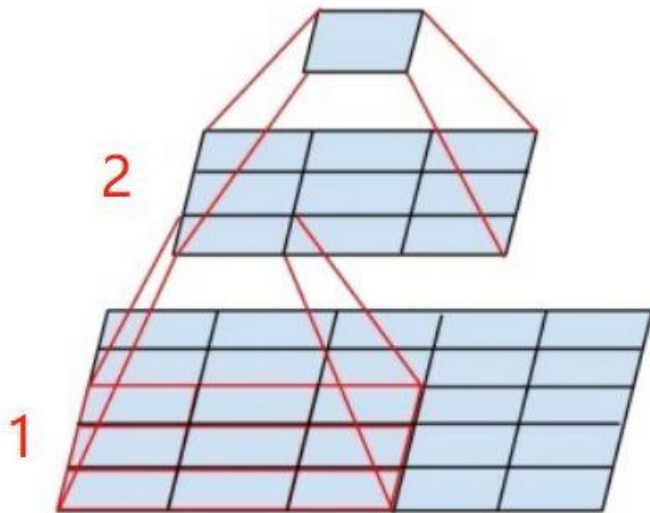
- VGG-16 has 16 weight layers, not including POOL layers w. 0 weight
- Input image:  $224 \times 224 \times 3$
- 1<sup>st</sup> and 2<sup>nd</sup> CONV layers: 64  $3 \times 3$  filters w. stride  $S = 1$ , pad  $P = 1$ 
  - Output volume:  $224 \times 224 \times 64$  (since  $\frac{1}{1}(224 + 2 * 1 - 3) + 1 = 224$ )
- 3<sup>rd</sup> POOL layer:  $2 \times 2$  filters w. stride  $S = 2$ 
  - Output volume:  $112 \times 112 \times 64$  (since  $\frac{1}{2}(224 - 2) + 1 = 112$ )
- 4<sup>th</sup> and 5<sup>th</sup> CONV layers: 128  $3 \times 3$  filters w. stride  $S = 1$ , pad  $P = 1$ 
  - Output volume:  $112 \times 112 \times 128$  (since  $\frac{1}{1}(112 + 2 * 1 - 3) + 1 = 112$ )
- 6<sup>th</sup> POOL layer:  $2 \times 2$  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 $3 \times 3$ CONV Layers

- 2 stacked  $3 \times 3$  CONV layers w. pad  $P = 1$  have the same effective receptive field as a  $5 \times 5$  CONV layer; 3 stacked  $3 \times 3$  CONV layers w. pad  $P = 1$  have RF of  $7 \times 7$ ;  $L$  stacked  $3 \times 3$  CONV layers w. pad  $P = 1$  have RF of  $1 + 2L$
- Benefits:
  - Fewer params. Suppose all volumes have the same depth  $D$ , then a  $7 \times 7$  CONV layer has  $(7 * 7 * D + 1) * D \approx 49D^2$  params, while three stacked  $3 \times 3$  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





# 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

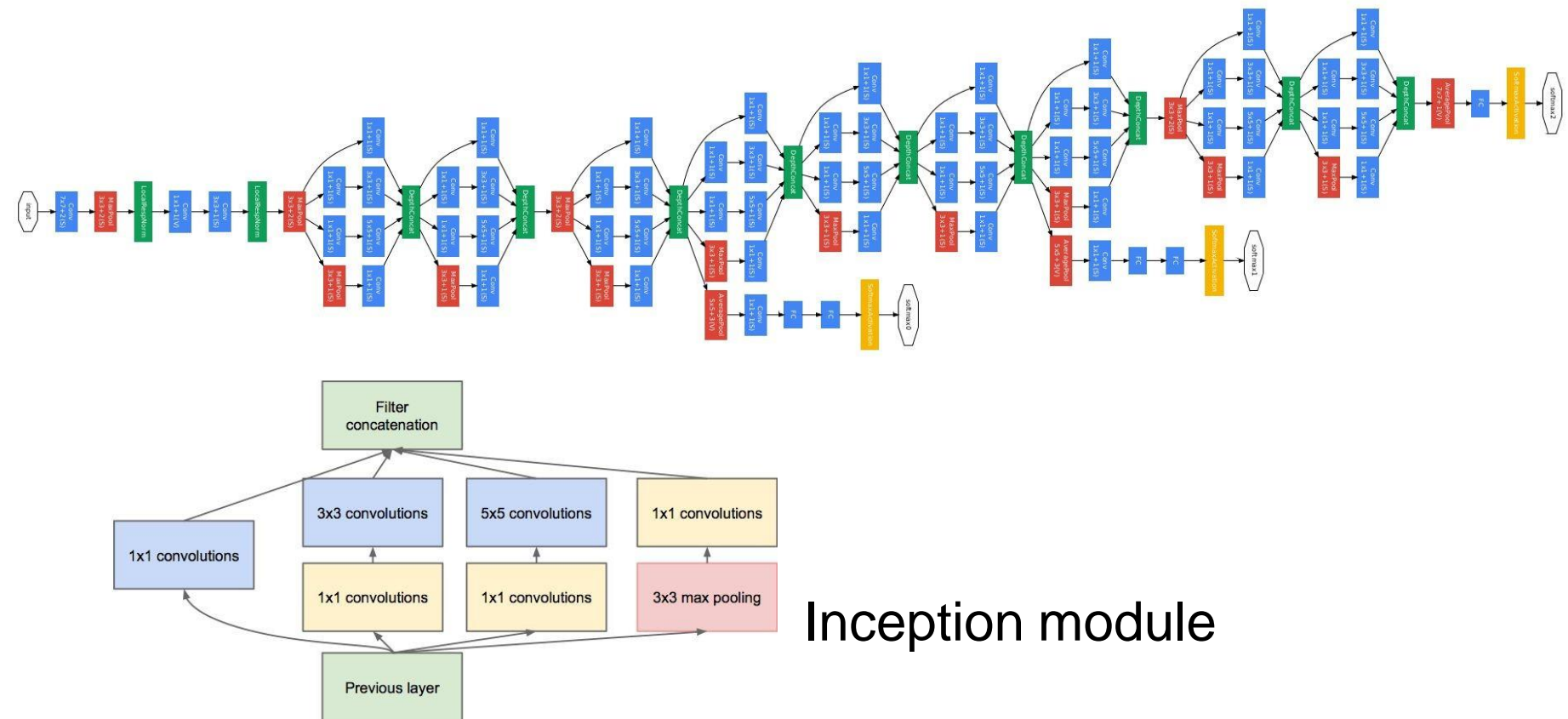
# VGGNet Variants

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<br>LRN        | conv3-64<br><b>conv3-64</b>   | conv3-64<br>conv3-64                       | conv3-64<br>conv3-64                       | conv3-64<br>conv3-64                                    |
| maxpool                     |                        |                               |  |  |   |
| conv3-128                   | conv3-128              | conv3-128<br><b>conv3-128</b> | conv3-128<br>conv3-128                     | conv3-128<br>conv3-128                     | conv3-128<br>conv3-128                                  |
| maxpool                     |                        |                               |  |  |   |
| conv3-256<br>conv3-256      | conv3-256<br>conv3-256 | conv3-256<br>conv3-256        | conv3-256<br>conv3-256<br><b>conv1-256</b> | conv3-256<br>conv3-256<br><b>conv3-256</b> | conv3-256<br>conv3-256<br>conv3-256<br><b>conv3-256</b> |
| maxpool                     |                        |                               |  |  |   |
| conv3-512<br>conv3-512      | conv3-512<br>conv3-512 | conv3-512<br>conv3-512        | conv3-512<br>conv3-512<br><b>conv1-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br>conv3-512<br><b>conv3-512</b> |
| maxpool                     |                        |                               |  |  |   |
| conv3-512<br>conv3-512      | conv3-512<br>conv3-512 | conv3-512<br>conv3-512        | conv3-512<br>conv3-512<br><b>conv1-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br>conv3-512<br><b>conv3-512</b> |
| maxpool                     |                        |                               |  |  |   |
| FC-4096                     |                        |                               |  |  |   |
| FC-4096                     |                        |                               |  |  |   |
| FC-1000                     |                        |                               |  |  |   |
| soft-max                    |                        |                               |  |  |   |

# Case Study: GoogLeNet [Szegedy et al., 2014]

- ImageNet top 5 error: 6.7%



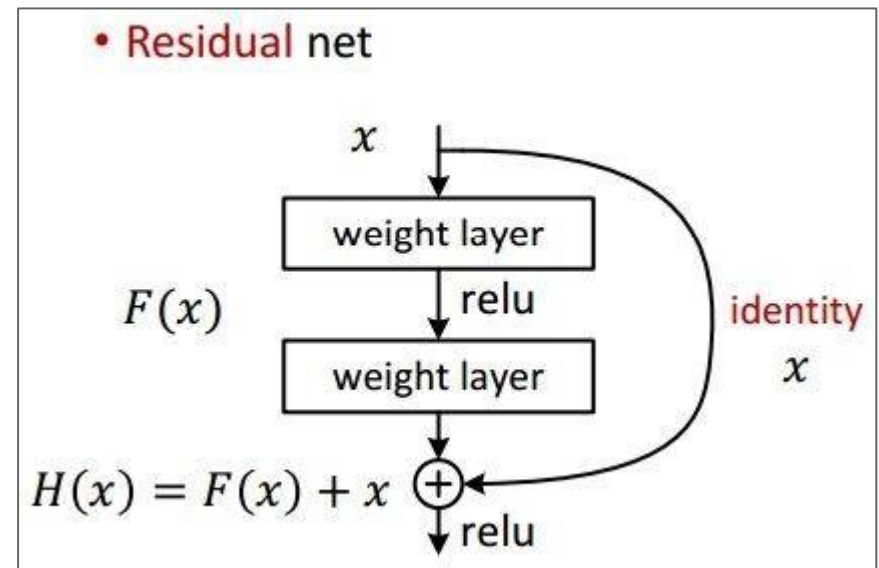
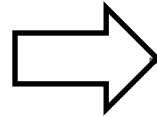
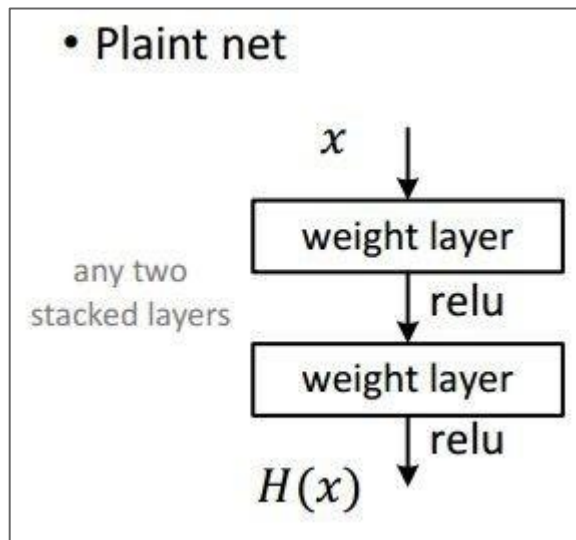
# GoogLeNet Size

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

| type           | patch size/<br>stride | output<br>size | depth | #1×1 | #3×3<br>reduce | #3×3 | #5×5<br>reduce | #5×5 | pool<br>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     |      |                |      |                |      |              |        |      |

# Case Study: Residual Networks (ResNet) [He et al. 2015]

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





- Right fig shows 34-layer ResNet

The diagram illustrates the VGG-16 architecture, which consists of the following layers and operations:

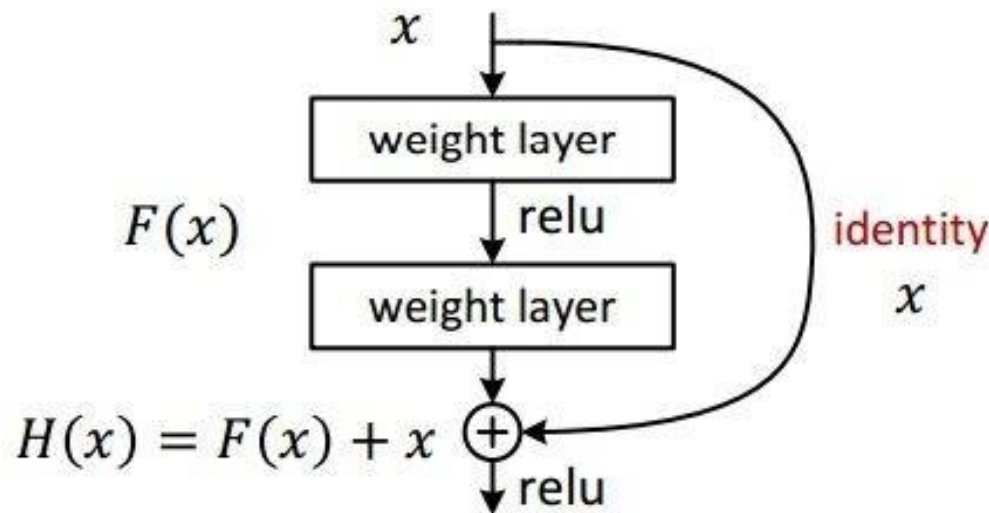
- Input Layer:** An arrow pointing down to the first convolutional layer.
- Layer 1:** 7x7 conv, 64, /2 (shaded orange).
- Layer 2:** pool, /2.
- Layer 3:** 3x3 conv, 64 (shaded purple).
- Layer 4:** 3x3 conv, 64 (shaded purple).
- Layer 5:** 3x3 conv, 64 (shaded purple).
- Layer 6:** 3x3 conv, 64 (shaded purple).
- Layer 7:** 3x3 conv, 64 (shaded purple).
- Layer 8:** 3x3 conv, 128, /2 (shaded green).
- Layer 9:** 3x3 conv, 128 (shaded green).
- Layer 10:** 3x3 conv, 128 (shaded green).
- Layer 11:** 3x3 conv, 128 (shaded green).
- Layer 12:** 3x3 conv, 128 (shaded green).
- Layer 13:** 3x3 conv, 128 (shaded green).
- Layer 14:** 3x3 conv, 128 (shaded green).
- Layer 15:** 3x3 conv, 256, /2 (shaded red).
- Layer 16:** 3x3 conv, 256 (shaded red).
- Layer 17:** 3x3 conv, 256 (shaded red).
- Layer 18:** 3x3 conv, 256 (shaded red).
- Layer 19:** 3x3 conv, 256 (shaded red).
- Layer 20:** 3x3 conv, 256 (shaded red).
- Layer 21:** 3x3 conv, 256 (shaded red).
- Layer 22:** 3x3 conv, 256 (shaded red).
- Layer 23:** 3x3 conv, 256 (shaded red).
- Layer 24:** 3x3 conv, 512, /2 (shaded blue).
- Layer 25:** 3x3 conv, 512 (shaded blue).
- Layer 26:** 3x3 conv, 512 (shaded blue).
- Layer 27:** 3x3 conv, 512 (shaded blue).
- Layer 28:** 3x3 conv, 512 (shaded blue).
- Layer 29:** avg pool.
- Layer 30:** fc 1000.

Connections between layers are shown with arrows. Solid arrows indicate the main flow, while dashed arrows indicate skip connections from the output of a convolutional layer to the input of a pooling layer (e.g., from Layer 3 to Layer 2, Layer 7 to Layer 8, Layer 14 to Layer 15, Layer 23 to Layer 24).

# ResNet Skip Connection

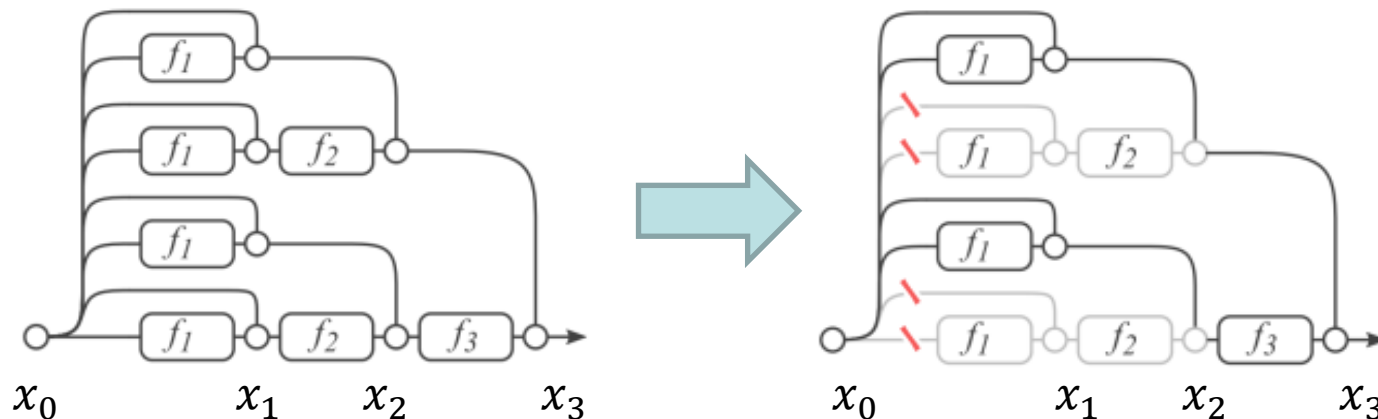
- In a standard network, output from a given layer is  $F(x)$
- In ResNet w. the identity skip (or short-cut) connection, output from a given layer is  $H(x) = F(x) + x$ 
  - If identity mapping is close to optimal, then weights can be small to capture minor differences only
- Benefits:
  - Residual connections help in handling the vanishing gradient problem in very deep NNs
  - It is safe to train very deep NNs, because “unnecessary layers” can learn to be identity mapping and do not cause overfitting

## • Residual net



# Consider a 3-layer Network

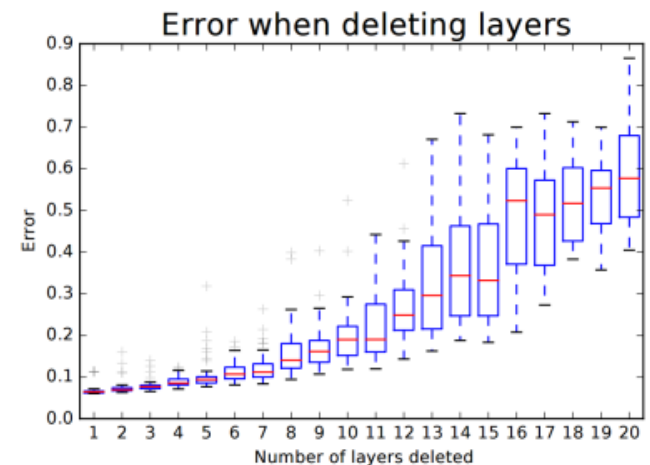
- Standard NN:
  - $x_3 = f_3(f_2(f_1(x_0)))$
- ResNet:
  - $x_1 = f_1(x_0) + x_0$
  - $x_2 = f_2(x_1) + x_1 = f_2(f_1(x_0) + x_0) + f_1(x_0) + x_0$
  - $x_3 = f_3(x_2) + x_2 = f_3(f_2(f_1(x_0) + x_0) + f_1(x_0) + x_0) + f_2(f_1(x_0) + x_0) + f_1(x_0) + x_0$
- Suppose  $f_2(x_1)$  is a vector of very 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  $x_3$





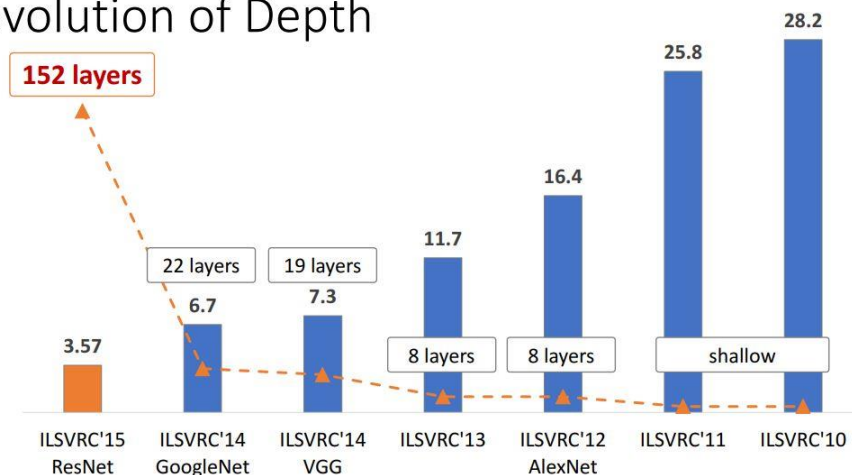
# ResNet is Ensemble of Models

- 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

Revolution of Depth



Revolution of Depth

AlexNet, 8 layers  
(ILSVRC 2012)

VGG, 19 layers  
(ILSVRC 2014)

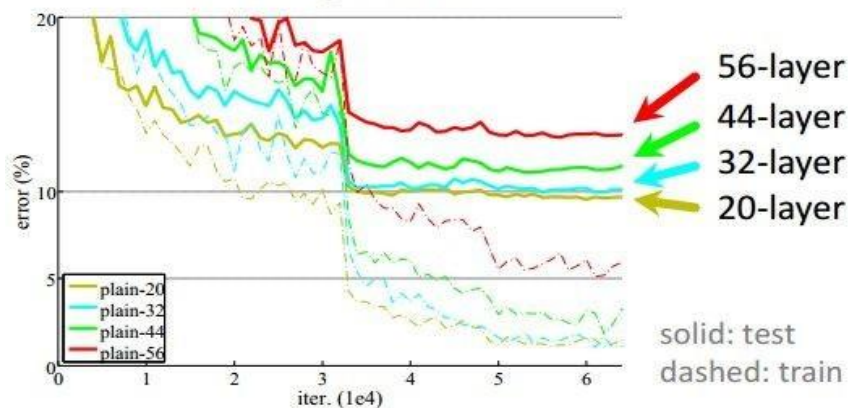
ResNet, 152 layers  
(ILSVRC 2015)



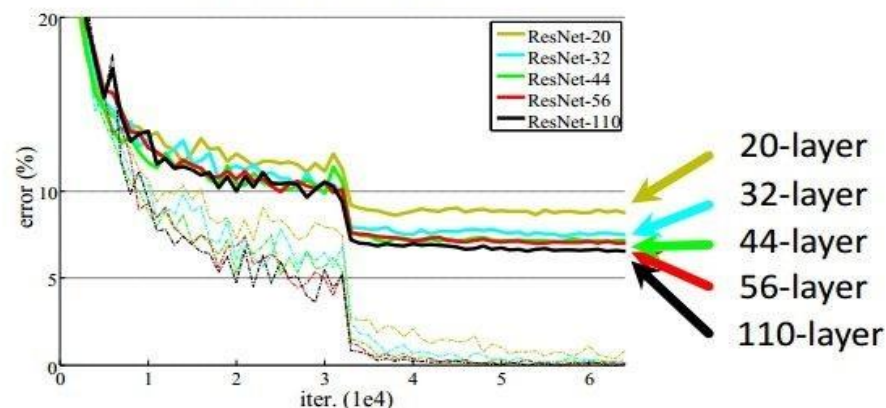
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

## CIFAR-10 experiments

CIFAR-10 plain nets



CIFAR-10 ResNets

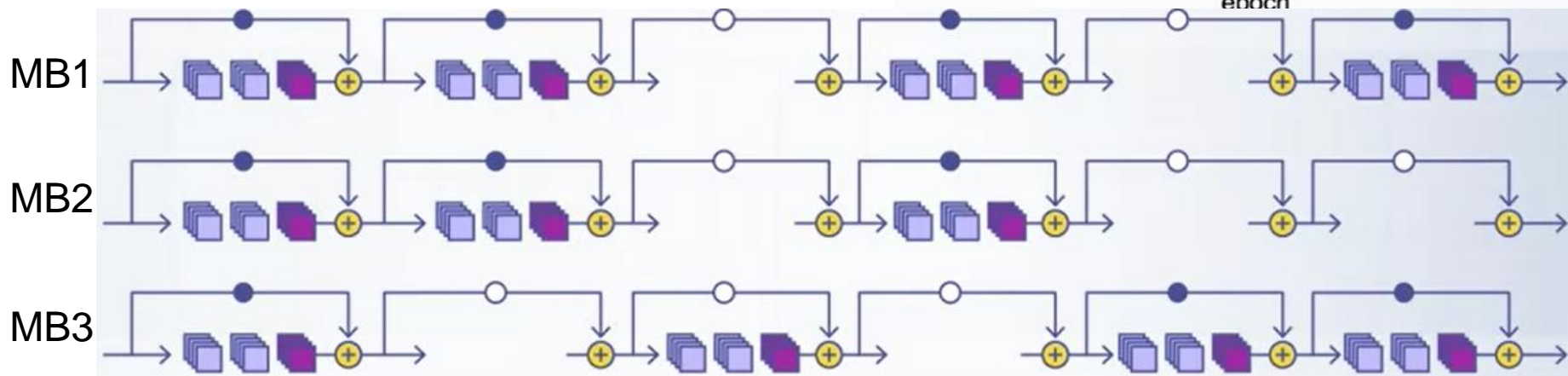
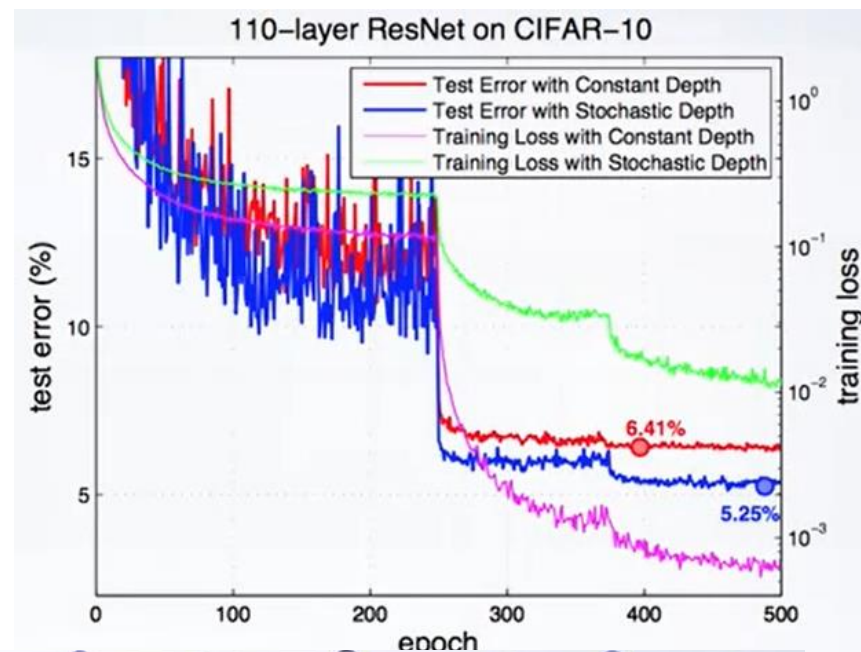


# Case Study: Policy Network in AlphaGo

- Input volume: [19x19x48]
- CONV1: 192 5x5 filters, stride 1, pad 2
- Output volume: [19x19x192] (since  $\frac{1}{1}(19 + 2 * 2 - 5) + 1 = 19$ )
- CONV2..12: 192 3x3 filters, stride 1, pad 1
- Output volume: [19x19x192] (since  $\frac{1}{1}(19 + 2 * 1 - 3) + 1 = 19$ )
- CONV: 1 1x1 filter, stride 1, pad 0
- Output volume: [19x19] (since  $\frac{1}{1}(19 - 1) + 1 = 19$ )

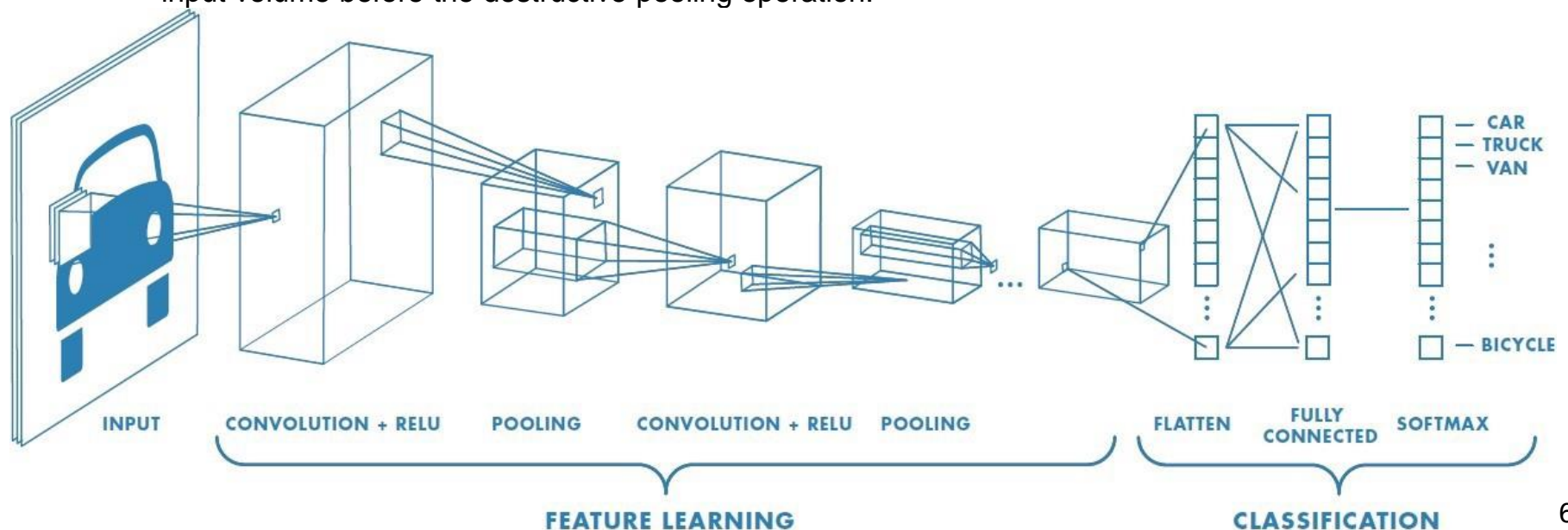
# Case Study: Stochastic Depth

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



# CNN Layer Patterns

- A typical CNN architecture looks like:  $\text{INPUT} \rightarrow [ [\text{CONV} \rightarrow \text{RELU}]^N \rightarrow \text{POOL?}]^M \rightarrow [ \text{FC} \rightarrow \text{RELU} ]^K \rightarrow \text{FC}$ 
  - where  $*$  indicates repetition, and  $\text{POOL?}$  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:
  - $\text{INPUT} \rightarrow \text{FC}$ , implements a linear classifier. Here  $N = M = K = 0$ .
  - $\text{INPUT} \rightarrow \text{CONV} \rightarrow \text{RELU} \rightarrow \text{FC}$
  - $\text{INPUT} \rightarrow [ \text{CONV} \rightarrow \text{RELU} \rightarrow \text{POOL} ]^2 \rightarrow \text{FC} \rightarrow \text{RELU} \rightarrow \text{FC}$  (fig below). There is a single CONV layer between every POOL layer.
  - $\text{INPUT} \rightarrow [ \text{CONV} \rightarrow \text{RELU} \rightarrow \text{CONV} \rightarrow \text{RELU} \rightarrow \text{POOL} ]^3 \rightarrow [ \text{FC} \rightarrow \text{RELU} ]^2 \rightarrow \text{FC}$  There are two CONV layers stacked before every POOL layer. 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 amount of memory 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 backpropagation, 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.

# Outline

- CNN Convolution layers
- Pooling and Fully-Connected layers
- CNN case studies
- RNNs

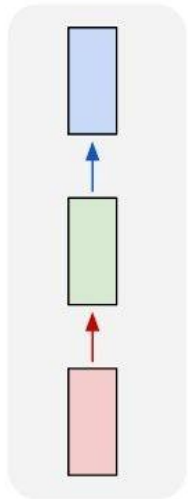


# Recurrent Neural Network (RNN)

- An RNN has connections between nodes that form a directed graph along a temporal sequence. This allows it to process variable-length input sequences and take into account dynamic temporal behavior
  - e.g., To take into account temporal sequence of consecutive frames in a video clip, we can either stack together a fixed number of frames as input to a CNN, or we can use an RNN (combined w. CNN) to process any variable-length sequence of frames
  - (Optional) Michael Phi, Illustrated Guide to Recurrent Neural Networks: Understanding the Intuition  
<https://www.youtube.com/watch?v=LHXXI4-IEns>

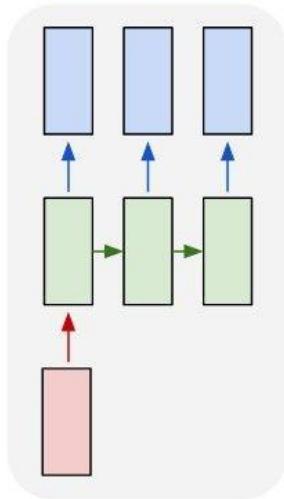
# RNN Architecture Variants

one to one



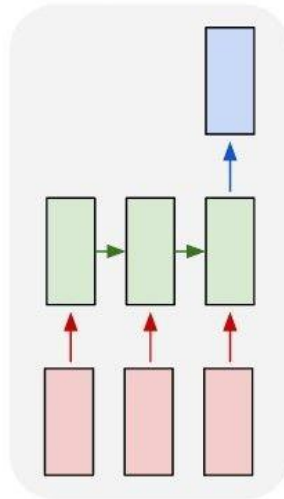
Regular  
Feedforward  
NN

one to many



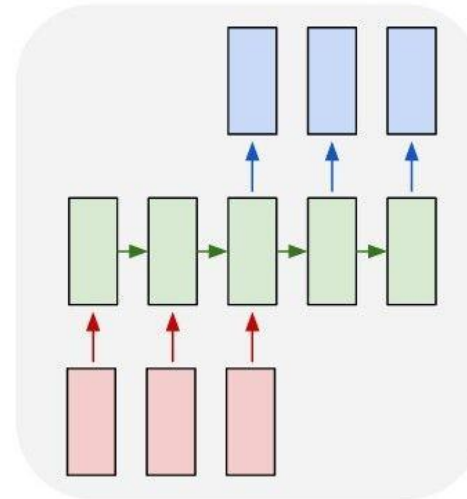
e.g. Image  
Captioning  
image ->  
sequence of  
words

many to one



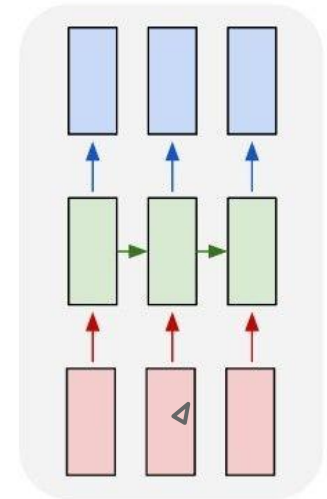
e.g. Sentiment  
Classification  
sequence of  
words ->  
sentiment

many to many



e.g. Machine Translation  
seq of words -> seq of  
words

many to many



e.g., video  
classification  
on frame level

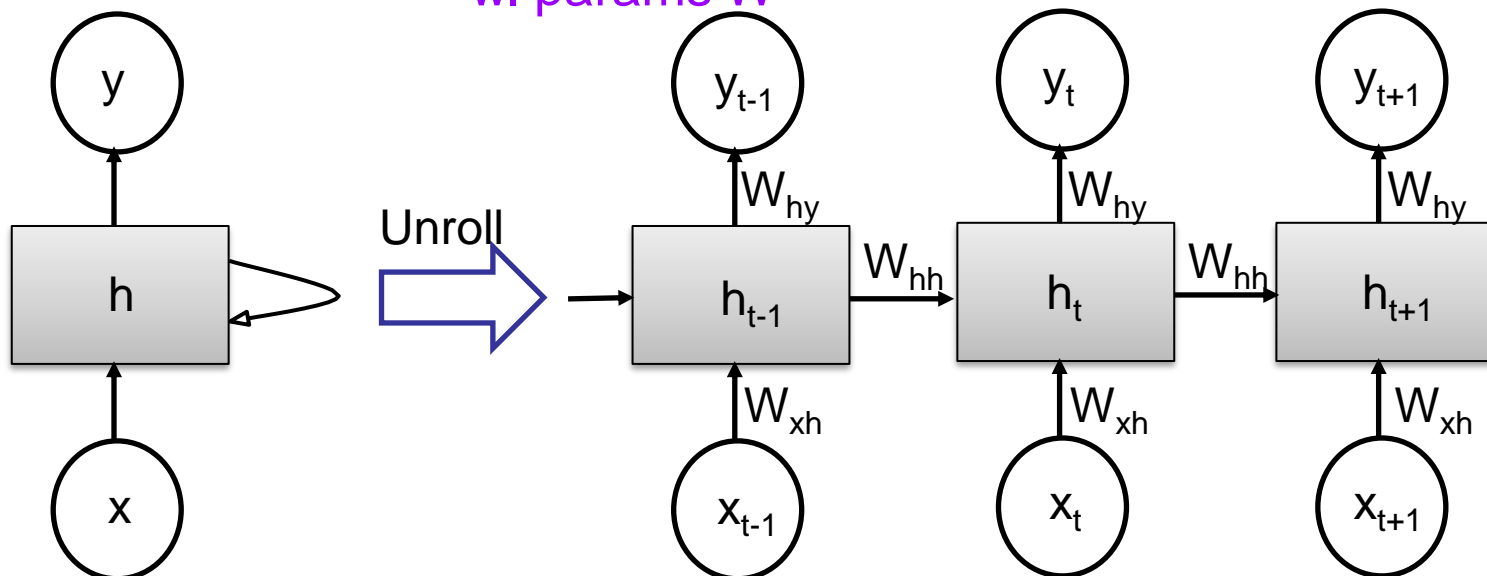
# RNN Many-to-Many Architecture

- RNN can process a sequence of inputs  $x$  recurrently at every time step, w. the same activation function and parameters  $f_W$ 
  - e.g.  $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$ ,  $y_t = W_{hy}h_t$
  - $h_t$  can even be a large CNN (without the last classification layer)

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

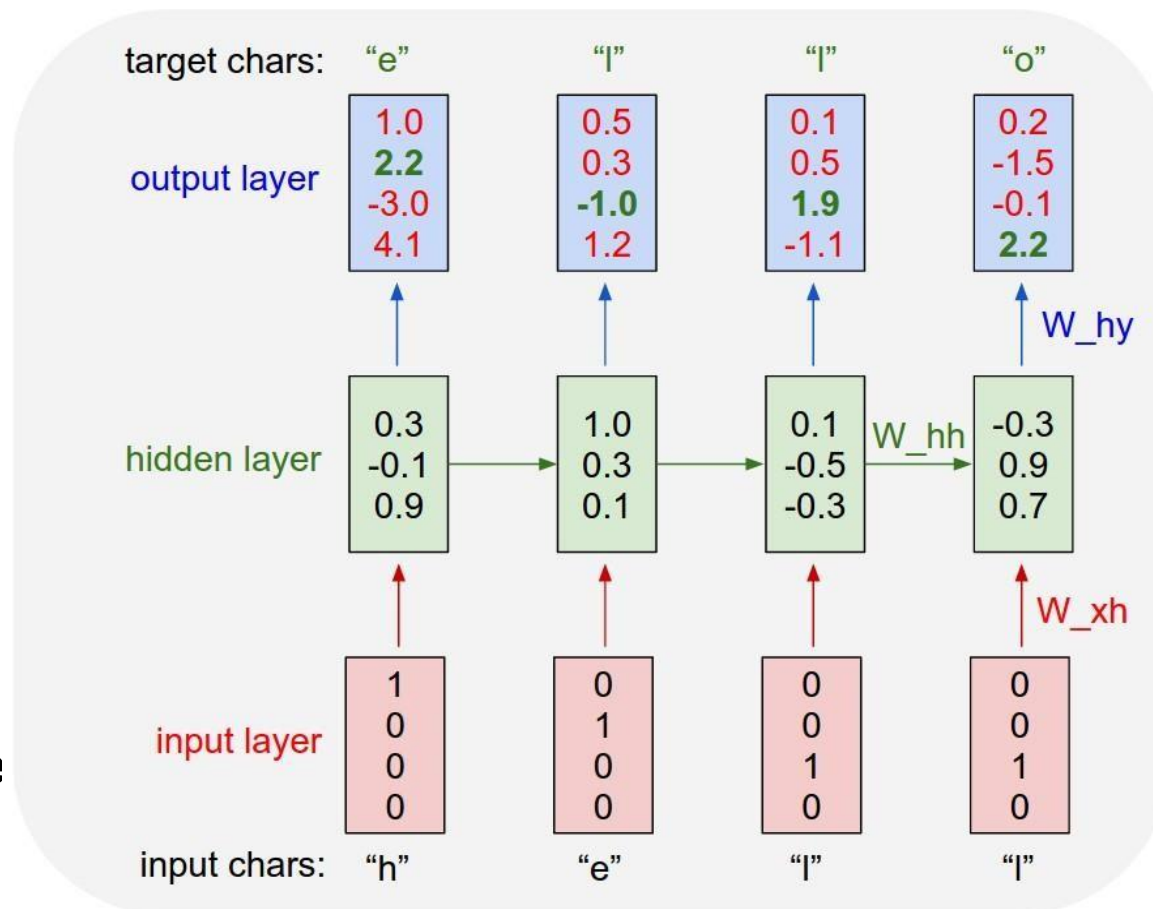
new state at timestep  $t$       old state at  $t - 1$       input at timestep  $t$

Activation function  
w. params  $W$



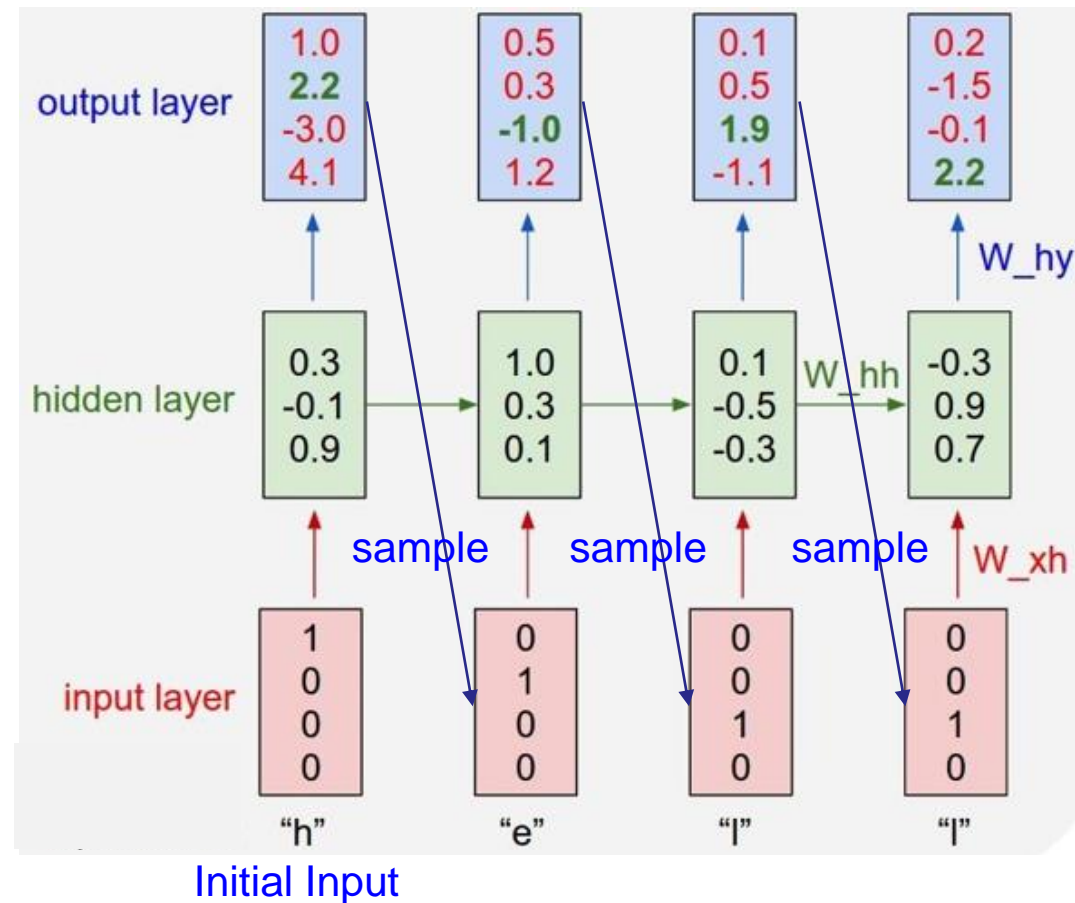
# Example: Character-Level Language Model, Training Time

- Task: predict the next char from current char sequence
- Example training sequence: “hello” w. vocabulary: [h,e,l,o]
  - Input is one-hot encoding of each char
  - Hidden layer is learned embedding
  - Output layer is a probability vector w. size 4, denoting prob distribution of next char (Fig shows the activation values before applying the SoftMax function for computing probabilities).



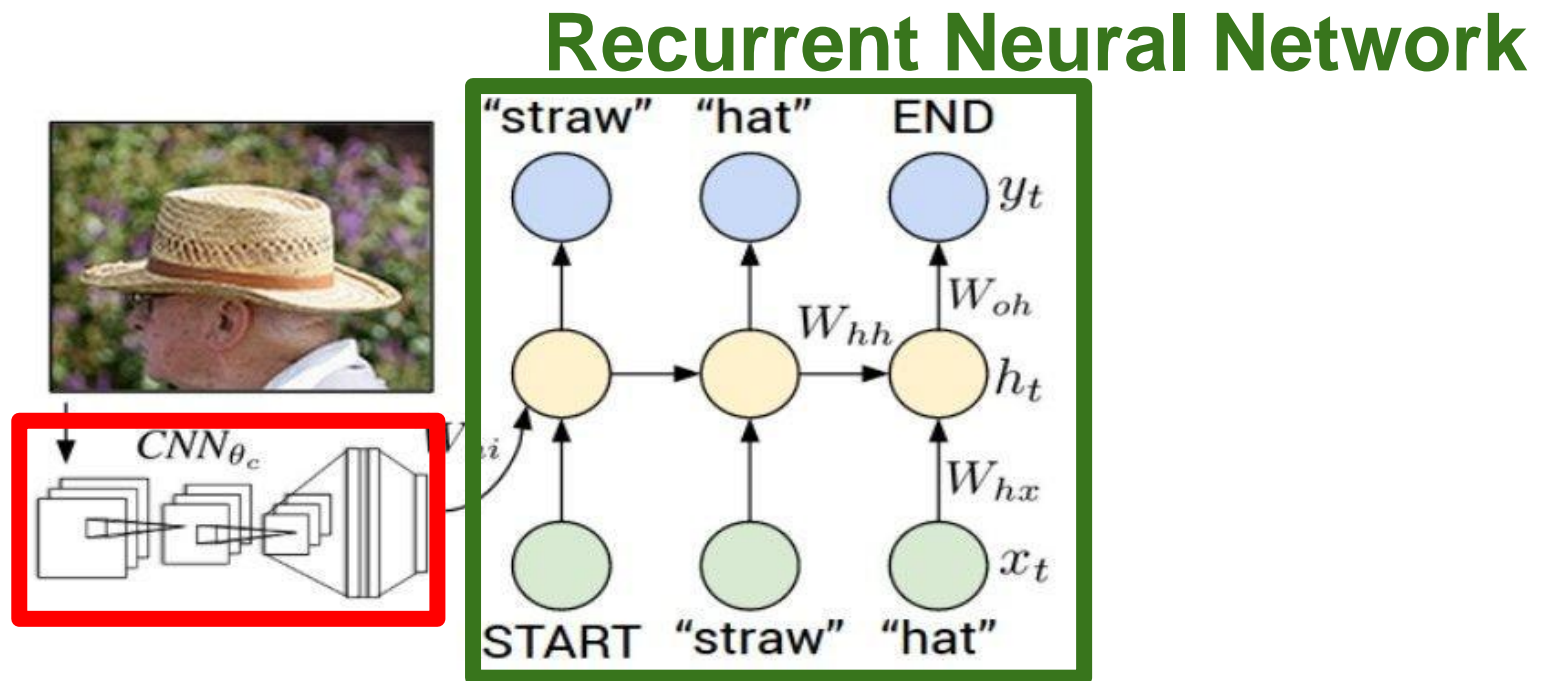
# Example: Character-Level Language Model, Inference Time

- Initial input is char “h”
- At each timestep, sample from the prob vector of the output  $y_t$  to generate the next input char
  - Here we assume the char w. highest prob is always selected as output at each timestep, i.e., “e”, “l”, “l” in sequence, but it is possible to select the other choices, e.g., “l”, “e”, “e”)



# Example: Image Captioning

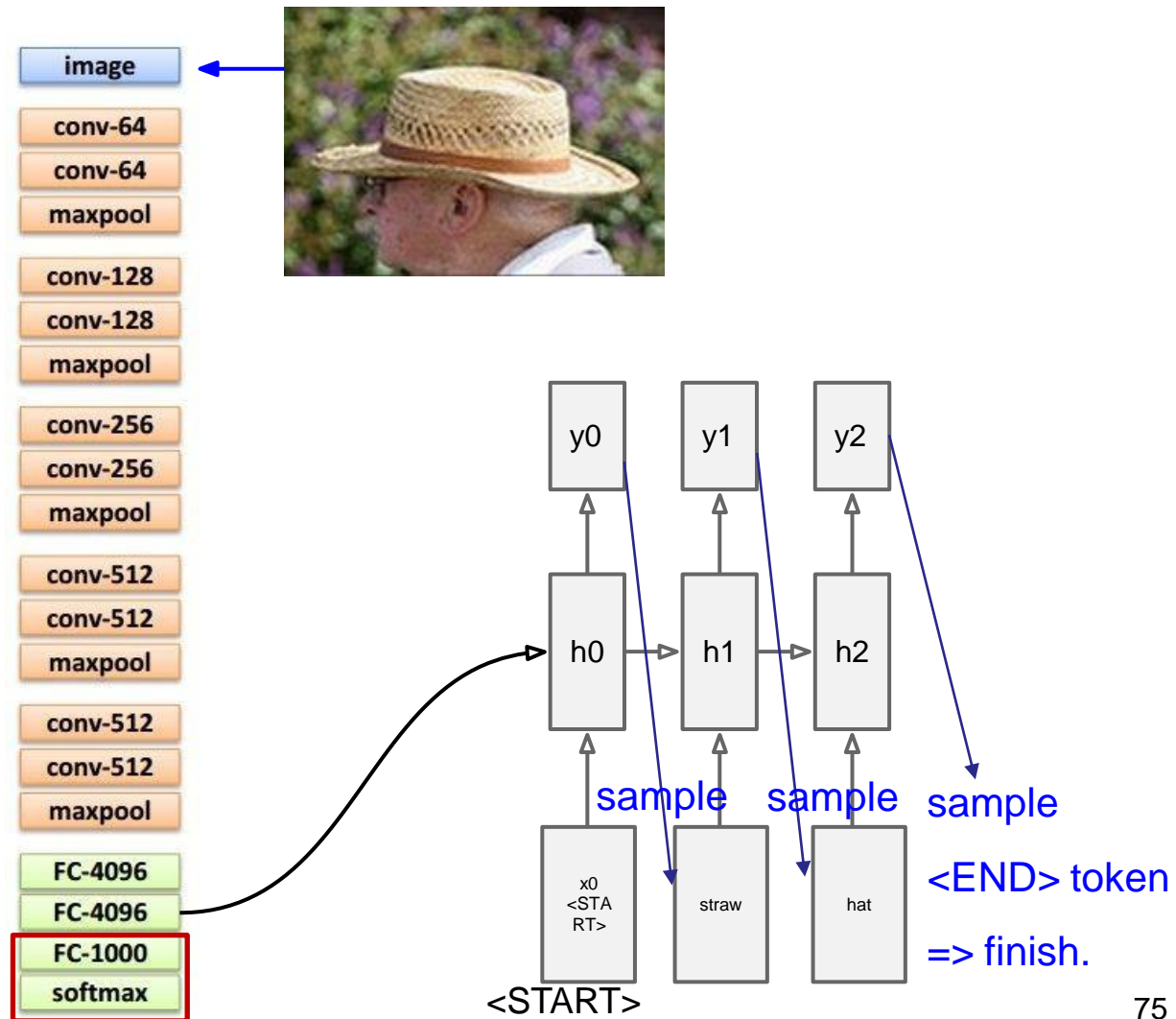
- CNN processes the input image and generates a feature vector as input to RNN



**Convolutional Neural Network**

# Image Captioning: Word-Level Language Model, Inference Time

- The last 2 layers of the CNN for classification (FC-1000 and SoftMax) are not used, since we only need the extracted features from the layer FC-4096
- At each timestep, sample from the prob vector of the output  $y_t$  to generate the next input word





# Image Captioning Examples



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"a young boy is holding a baseball bat."



"a cat is sitting on a couch with a remote control."



"a woman holding a teddy bear in front of a mirror."



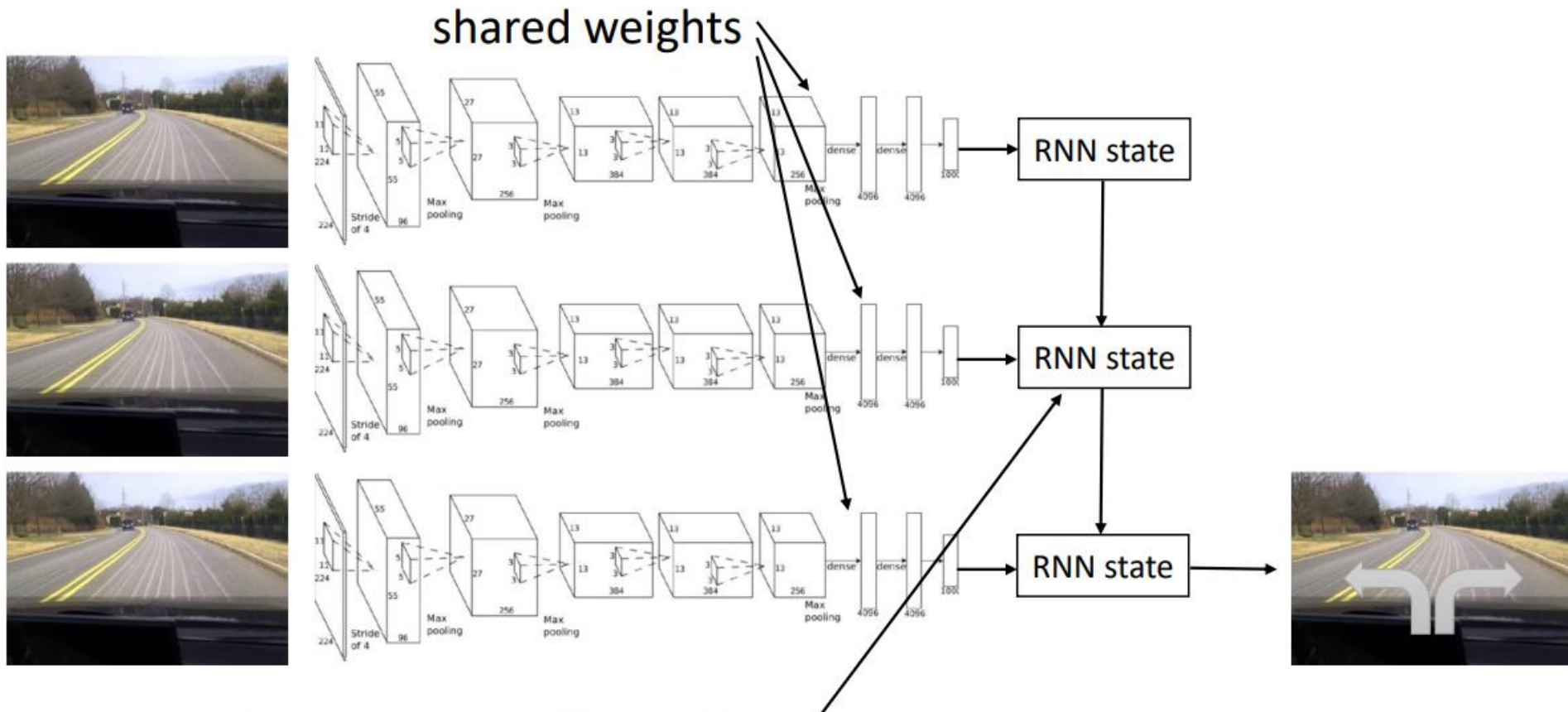
"a horse is standing in the middle of a road."

Bottom row shows failure cases



# RNN in AD

- Combined with CNN, RNN can handle Non-Markovian behavior, i.e, the current action depends not just on the current observation (input image), but on a recent history of observations



Typically, LSTM cells work better here

# RNN Summary

- Training of RNNs requires back propagation through time, which may cause exploding or vanishing gradient problems
- More sophisticated architectures are more practical
  - LSTM (Long Short-Term Memory Model) or GRU (Gated Recurrent Unit)
- RNNs are most widely used in Natural Language Processing, but it is also useful for processing videos, w. applications in autonomous driving