Convolutional Neural Networks

Session #6

A study group by dair.ai

Hello! 👋

- Salim Chemlal

/SalimChemlal

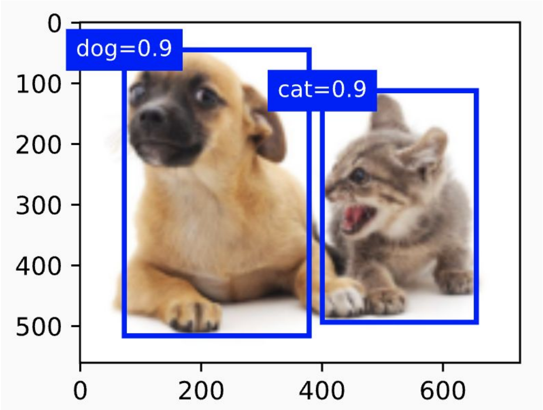
[dair.ai]

[@dair_ai]

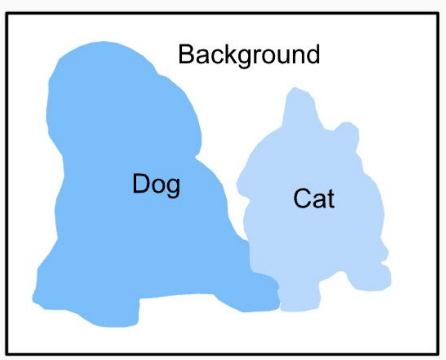
[Slack group]

[GitHub]

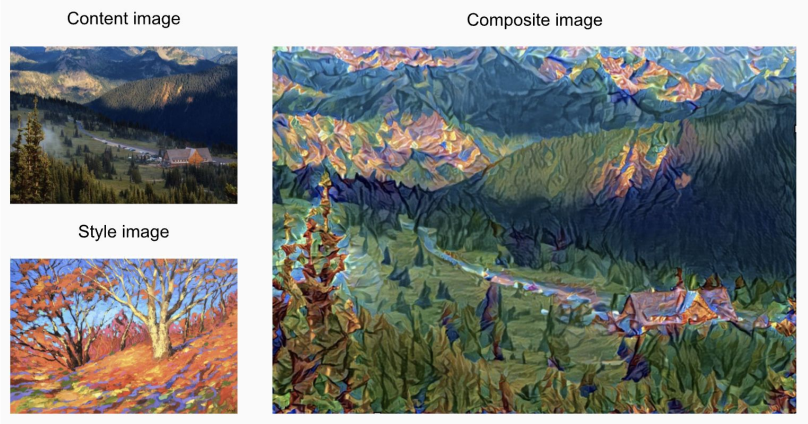
Convolution Neural Networks, ConvNets, CNNs



Classification

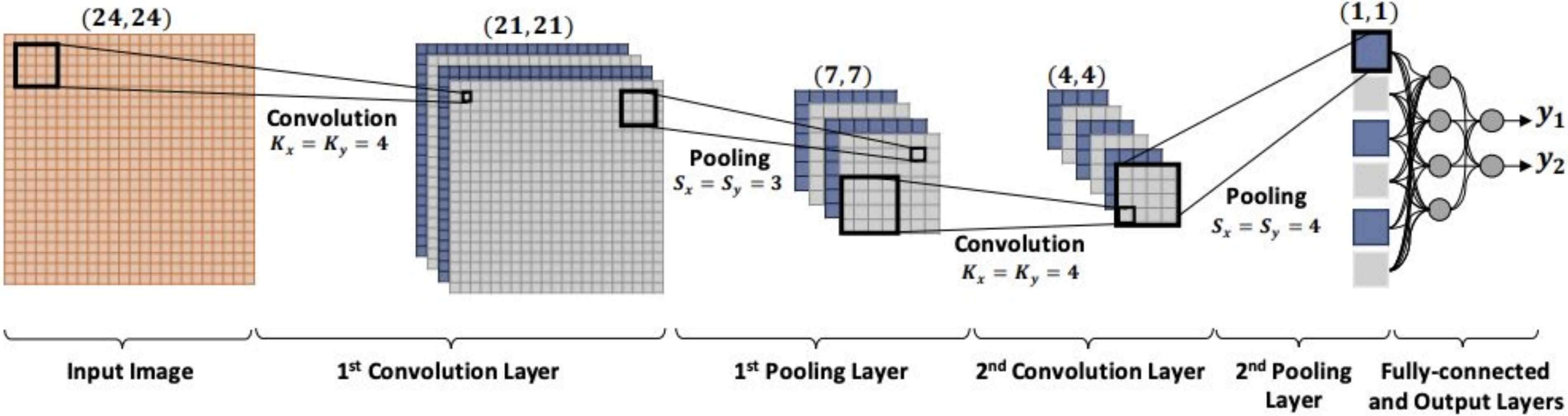


Segmentation



Neural Style Transfer

A Sample CNN



(24,24)
(21,21)
Convolution
$K_x = K_y = 4$
(7,7)
Pooling
$S_x = S_y = 3$
(4,4)
Convolution
$K_x = K_y = 4$
(1,1)
Pooling
$S_x = S_y = 4$
y_1
y_2
Input Image
1st Convolution Layer
1st Pooling Layer
2nd Convolution Layer
2nd Pooling Layer
Fully-connected and Output Layers

Fully-Connected Layers Constraints

- Assume

    1MP RGB Image

    &

    Model size of a single hidden MLP layer is 1000

- **What is size of weight matrix?**

Where is Waldo?

Intuitions behind CNNs

- **Translation invariance:**

  Network should respond similarly to the same patch, regardless of where it appears.

- **Locality:**

  Network should focus on local regions

MLP Recap:

Consider an MLP with a 2D image ***X*** and immediate hidden representation ***H*** in 2D, the fully-connected layer can be expressed as:

$$
\begin{aligned}
[\mathbf{H}]_{i,j} &= [\mathbf{U}]_{i,j} + \sum_k \sum_l [\mathsf{W}]_{i,j,k,l} [\mathbf{X}]_{k,l} \\
&= [\mathbf{U}]_{i,j} + \sum_a \sum_b [\mathsf{V}]_{i,j,a,b} [\mathbf{X}]_{i+a,j+b}
\end{aligned}
$$

where $[\mathsf{V}]_{i,j,a,b} = [\mathsf{W}]_{i,j,i+a,j+b}$

Principle 1: Translation Invariance

$$[\mathbf{H}]_{i,j} = [\mathbf{U}]_{i,j} + \sum_a \sum_b [\mathsf{V}]_{i,j,a,b}[\mathbf{X}]_{i+a,j+b}$$

A shift in **X** should lead to a shift in **H**

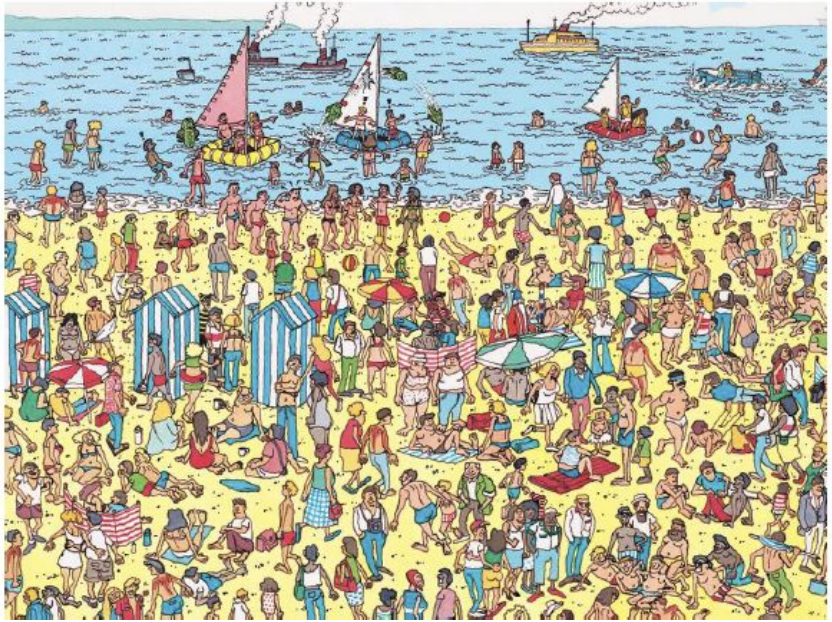
$\Rightarrow$ V and U do not actually depend on **(i, j)**

$$[\mathbf{H}]_{i,j} = u + \sum_a \sum_b [\mathbf{V}]_{a,b}[\mathbf{X}]_{i+a,j+b}$$

This is a ***convolution!***

Principle 2: Locality

$$[\mathbf{H}]_{i,j} = u + \sum_a \sum_b [\mathbf{V}]_{a,b}[\mathbf{X}]_{i+a,j+b}$$

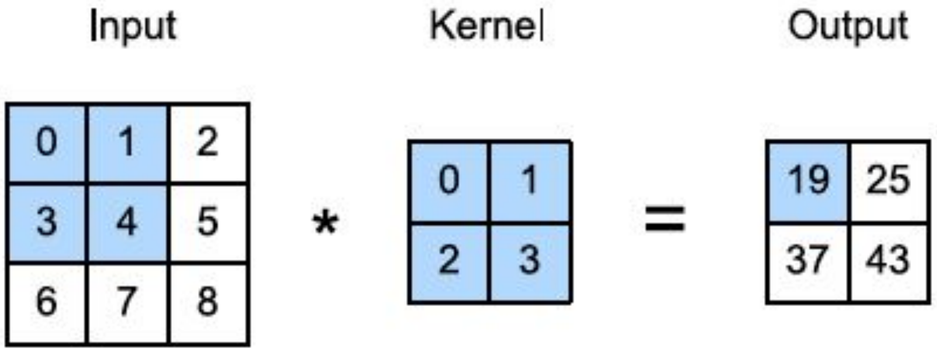


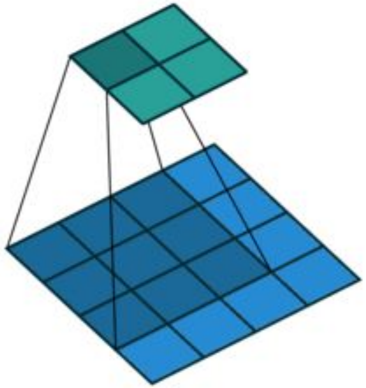
No need to look very far away from location ***(i, j)*** to assess what is going on at $[\mathbf{H}]_{i,j}$.

We set $[\mathbf{V}]_{a,b} = \mathbf{0}$ for $|a|, |b| > \Delta$

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{V}]_{a,b}[\mathbf{X}]_{i+a,j+b}$$

2-D Cross Correlation





$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

2-D Convolution Layer

$\mathbf{X} : n_h \times n_w$ input matrix

$\mathbf{W} : k_h \times k_w$ kernel matrix

b: scalar bias

$\mathbf{Y} : (n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

W and *b* are learnable parameters

PyTorch Code:

- Cross Correlation
- Convolution Layer
- Object Edge Detection
- Learning a Kernel

Cross-correlation vs Convolution

- Identical operations except that the kernel is flipped in convolution.

  - 2-D Cross Correlation

$$y_{i,j} = \sum_{a=1}^{h} \sum_{b=1}^{w} w_{a,b} x_{i+a,j+b}$$

  - 2-D Convolution

$$y_{i,j} = \sum_{a=1}^{h} \sum_{b=1}^{w} w_{-a,-b} x_{i+a,j+b}$$

If the kernel is symmetric, then they are identical.

Cross-correlation vs Convolution

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

$$\begin{matrix} & w & \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

Full correlation result

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 8 & 7 & 0 & 0 \\ 0 & 0 & 6 & 5 & 4 & 0 & 0 \\ 0 & 0 & 3 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Full convolution result

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 & 0 \\ 0 & 0 & 7 & 8 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

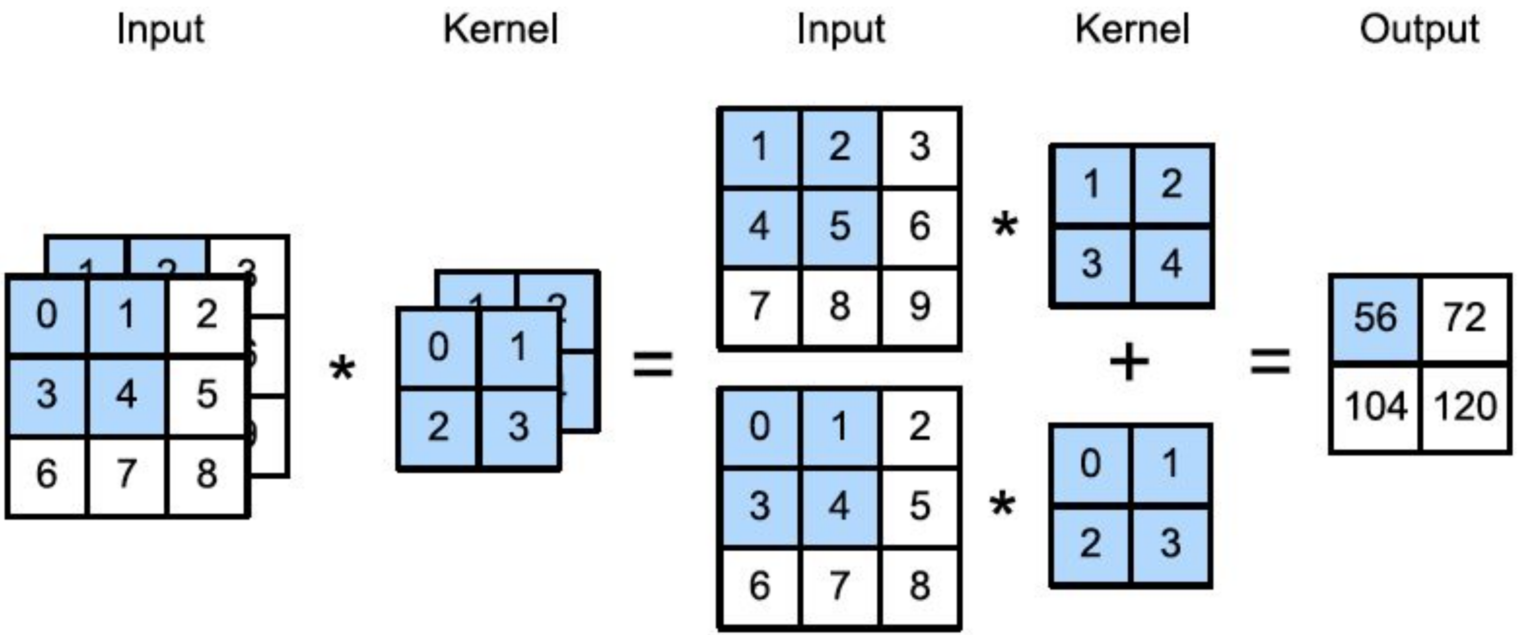
Many machine learning libraries implement cross-correlation but call it convolution.

In Deep Learning, since Kernels are learned, it does not matter!

Cool Edge Detection Demo

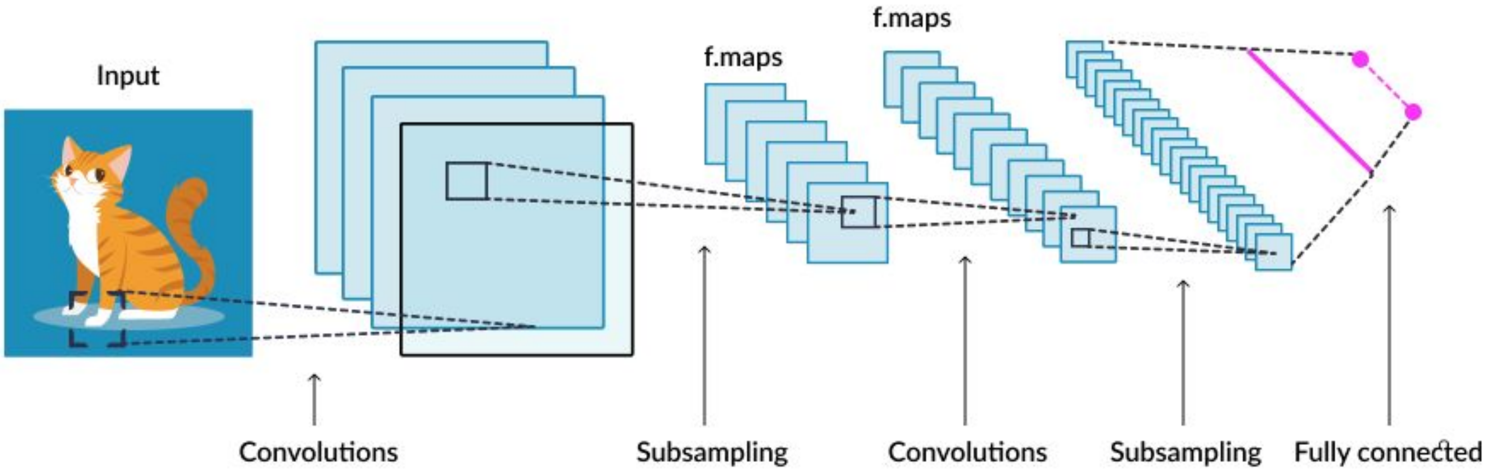
https://setosa.io/ev/image-kernels/

Multiple Input Channels



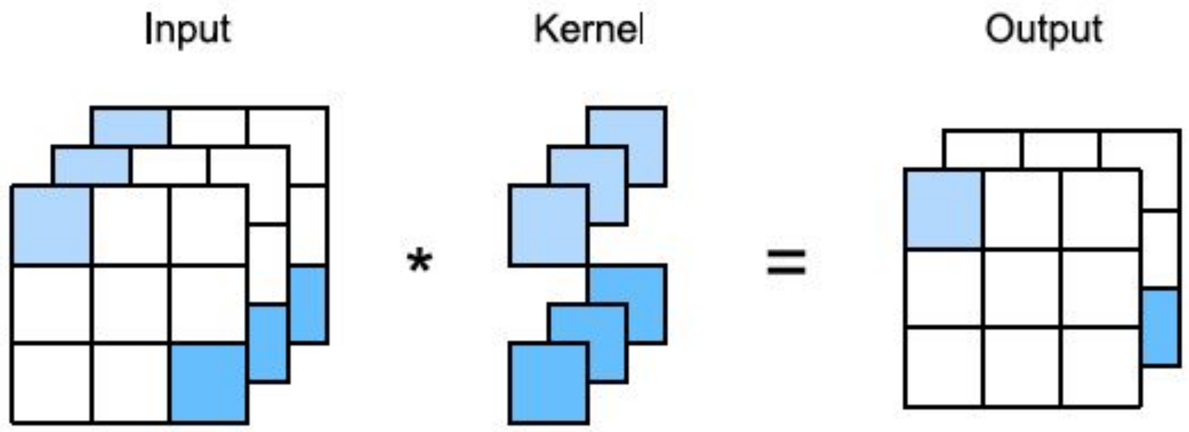
Note: ***Kernel must have same number of channels as input to perform cross-correlation***

Multiple Output Channels



Note: ***We typically increase channel dimension as we go higher up in the network***

Multiple Output Channels: 1x1 Convolutional Layer



Used to adjust number of channels between network layers and to control model complexity.

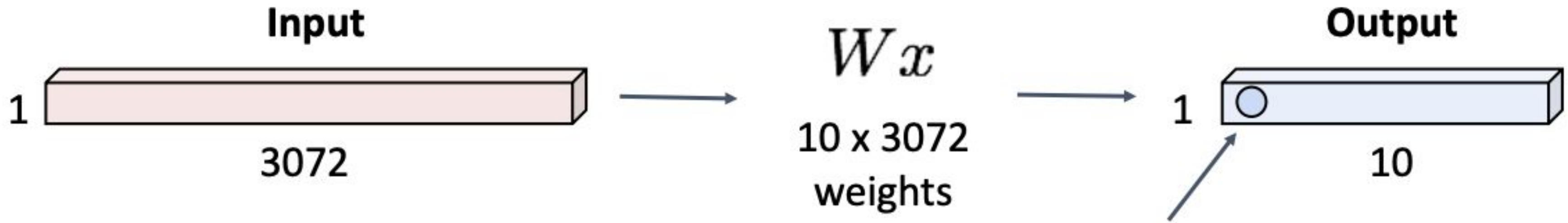
PyTorch Code:

- Multiple Input Channels
- Multiple Output Channels
- 1×1 Convolutional Layer

Recap & More

Fully-Connected Layer

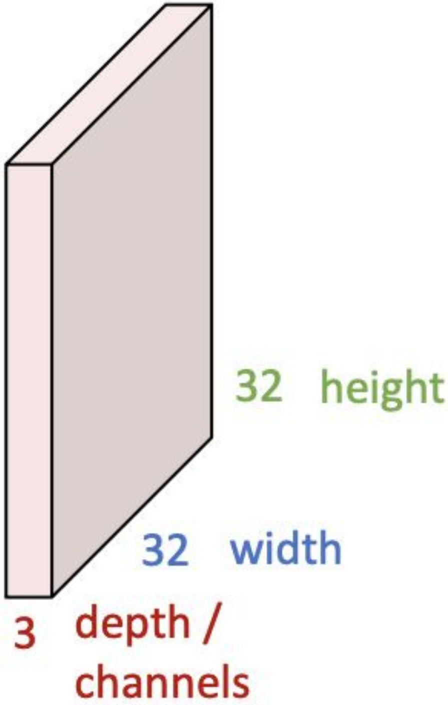
32x32x3 image -> stretch to 3072 x 1



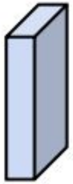
1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Convolutional Layer

3x32x32 image

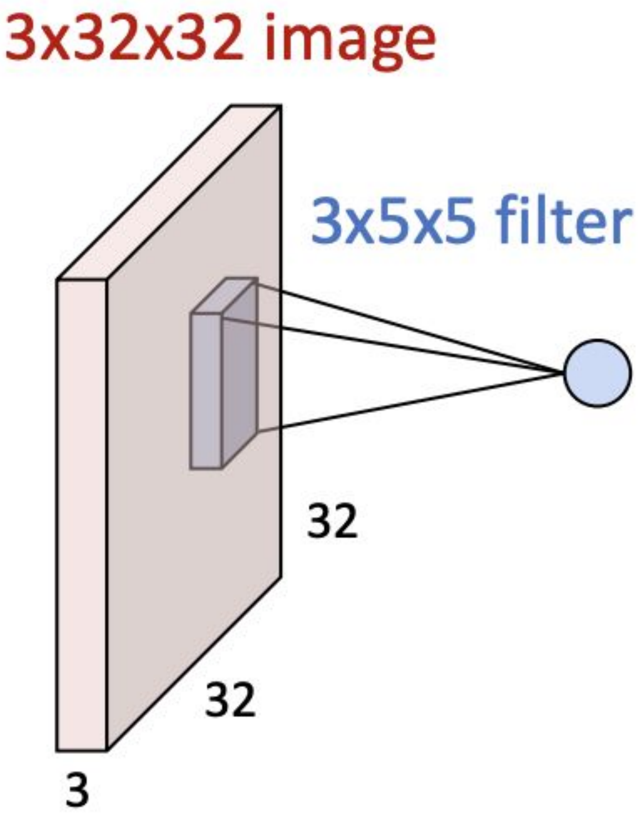


3x5x5 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

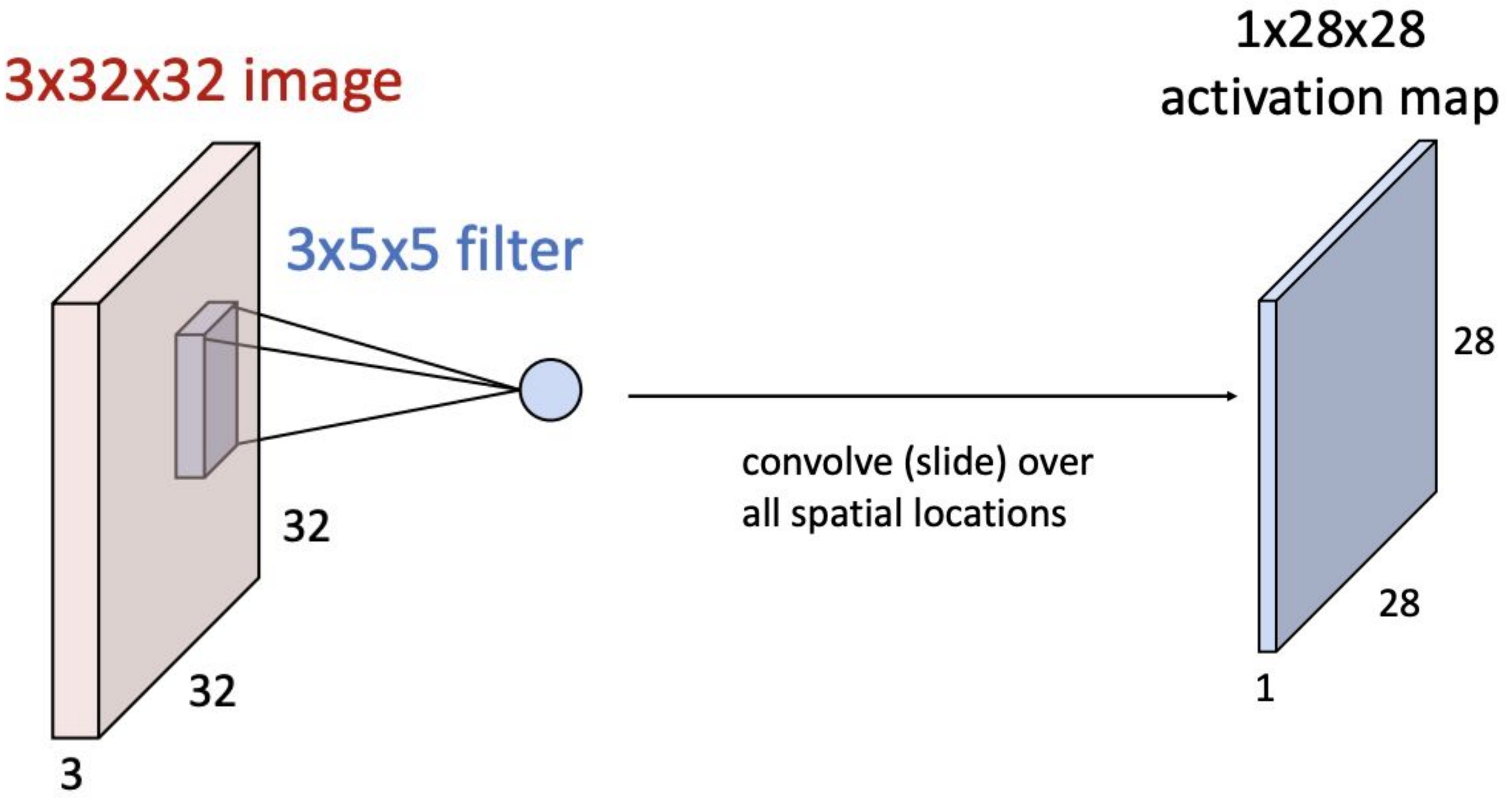
Convolutional Layer



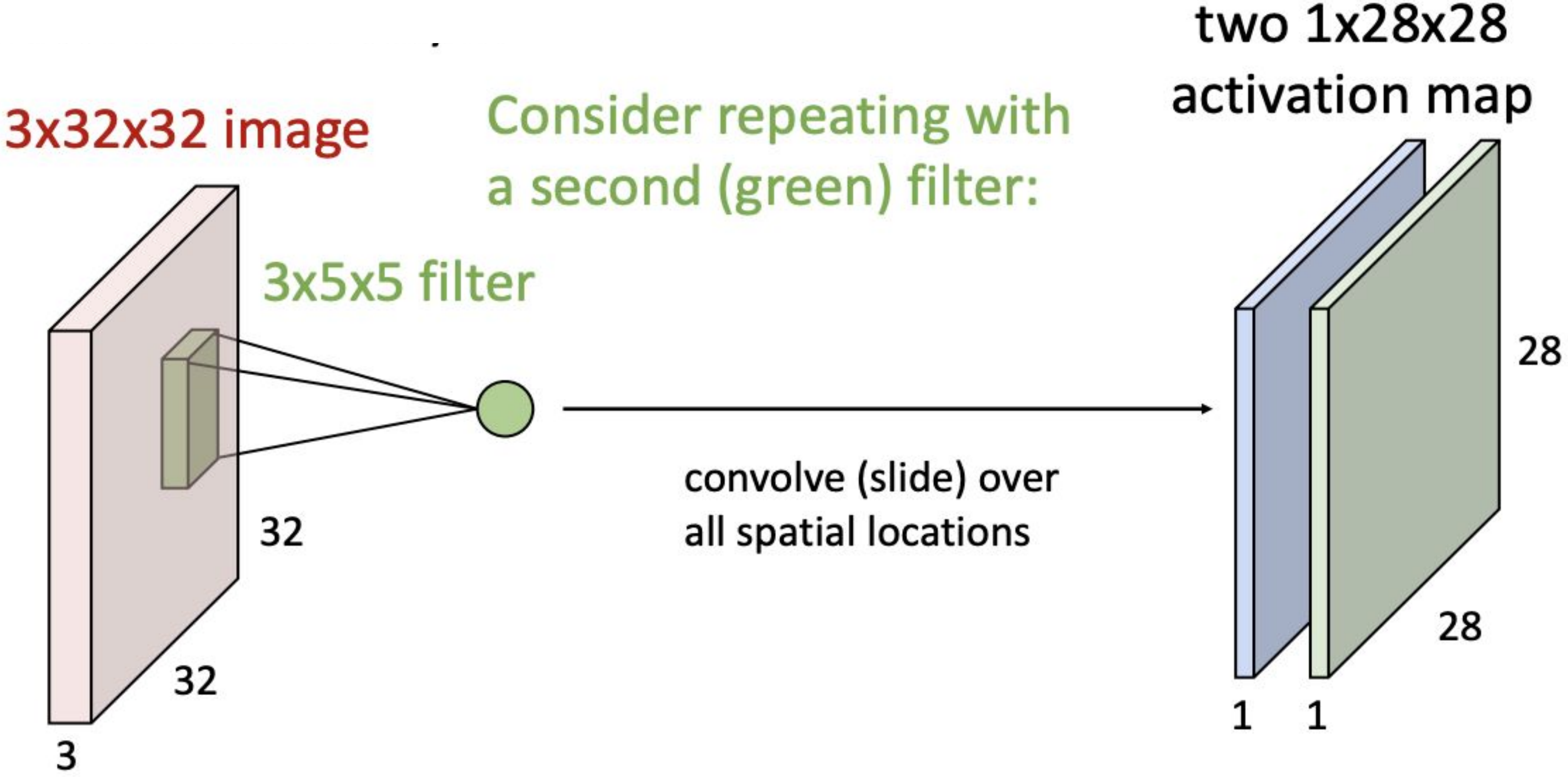
1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

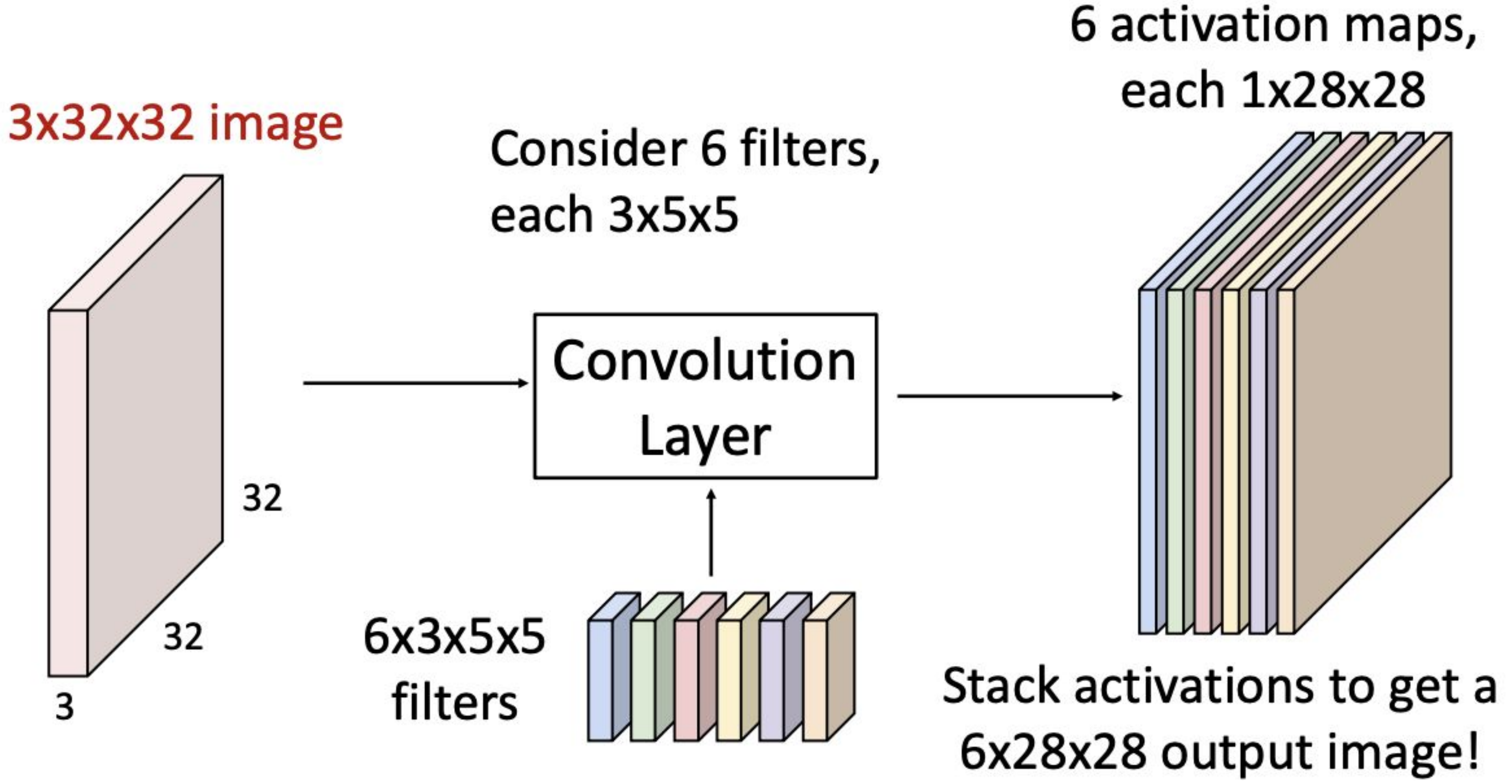
Convolutional Layer



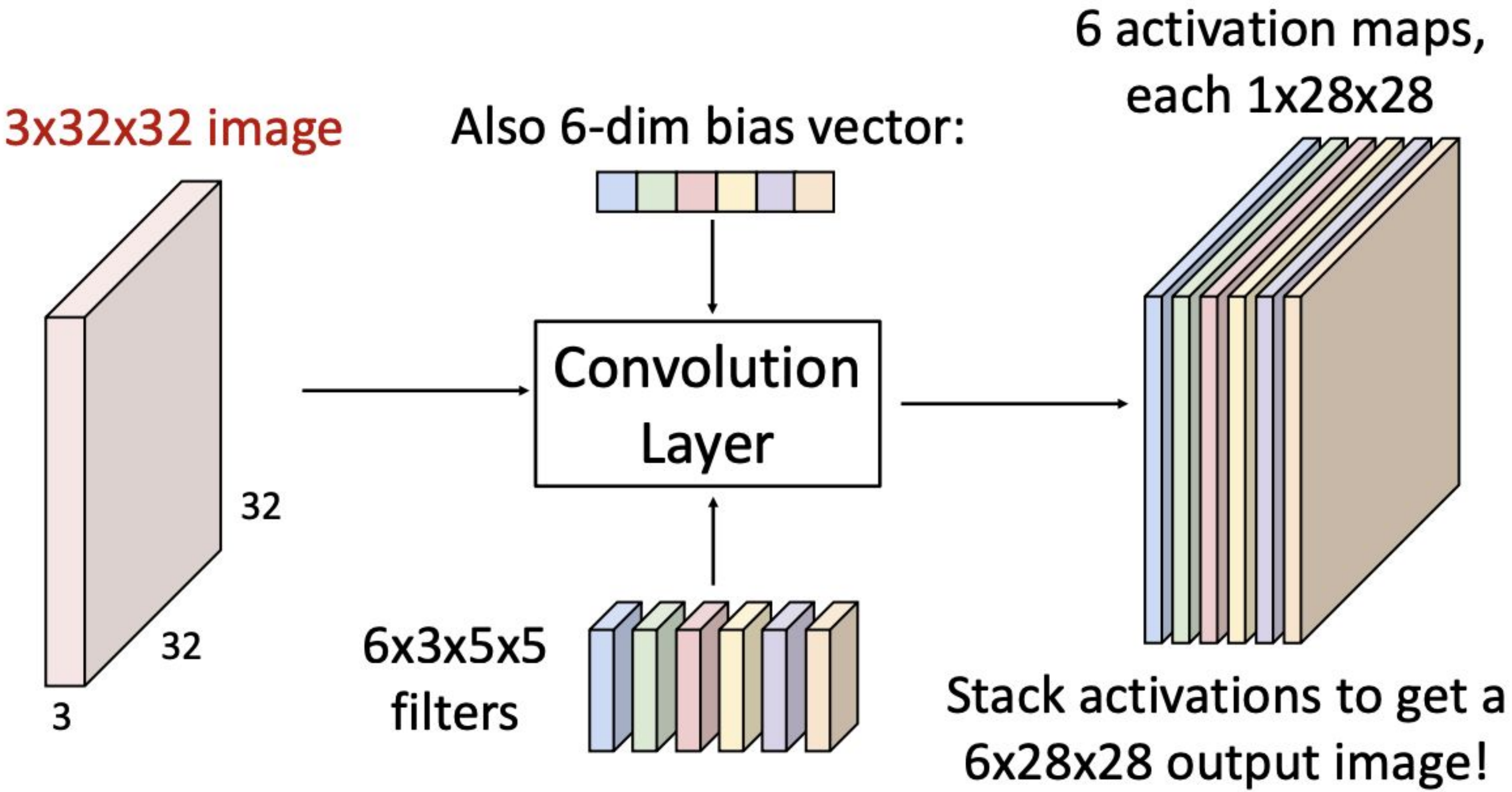
Convolutional Layer



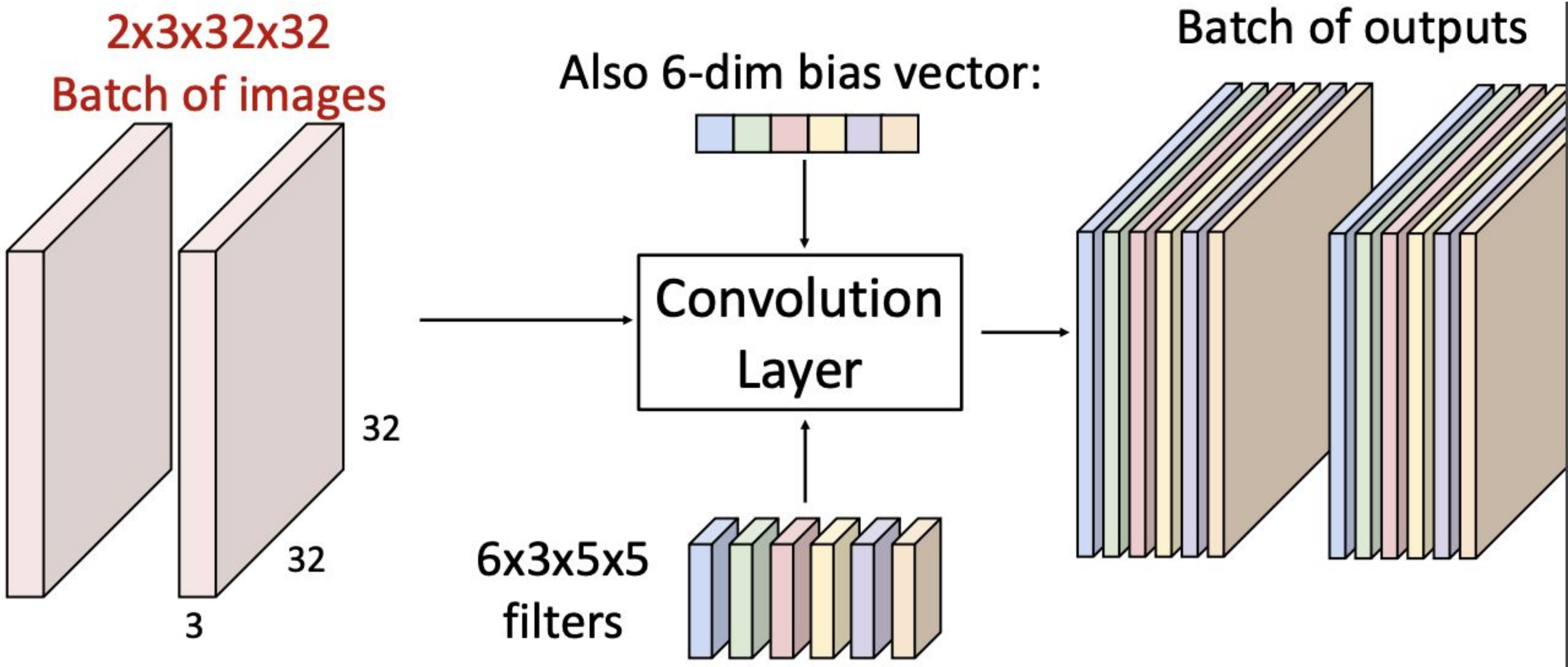
Convolutional Layer



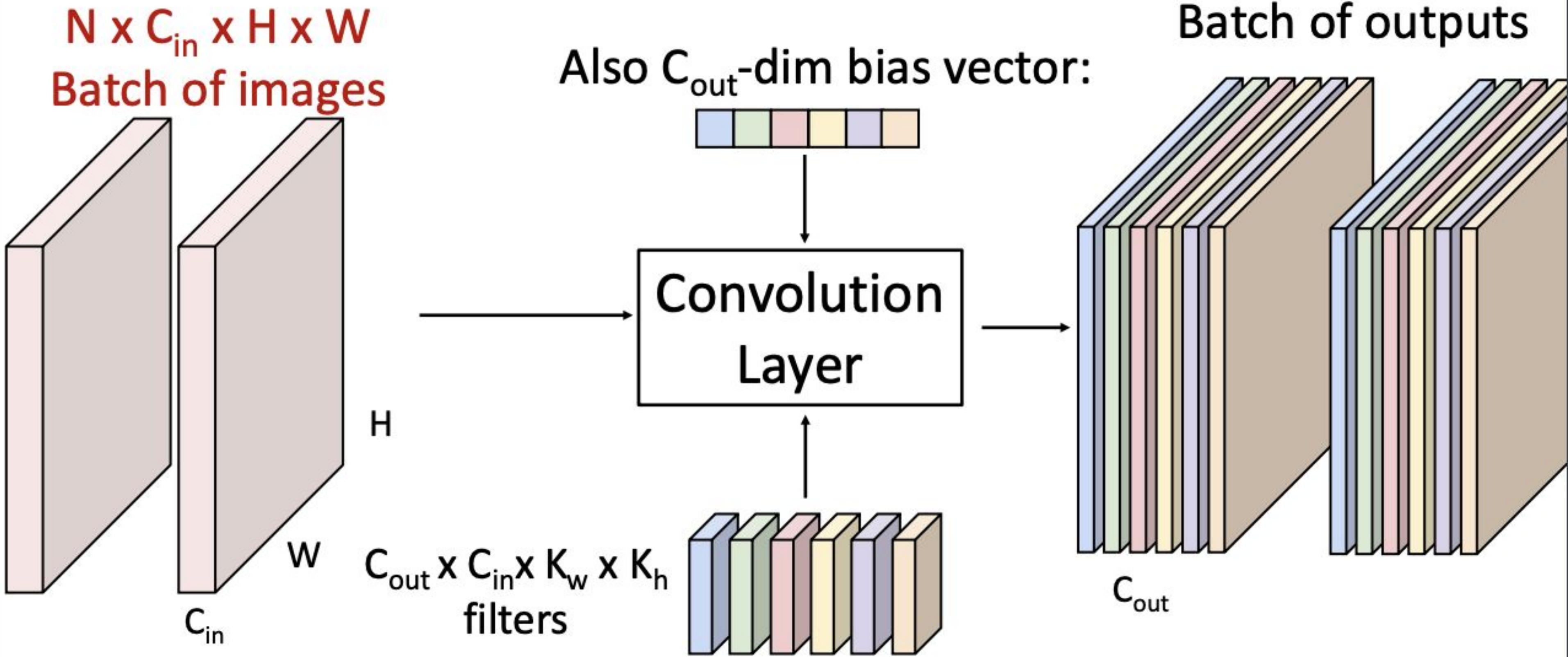
Convolutional Layer



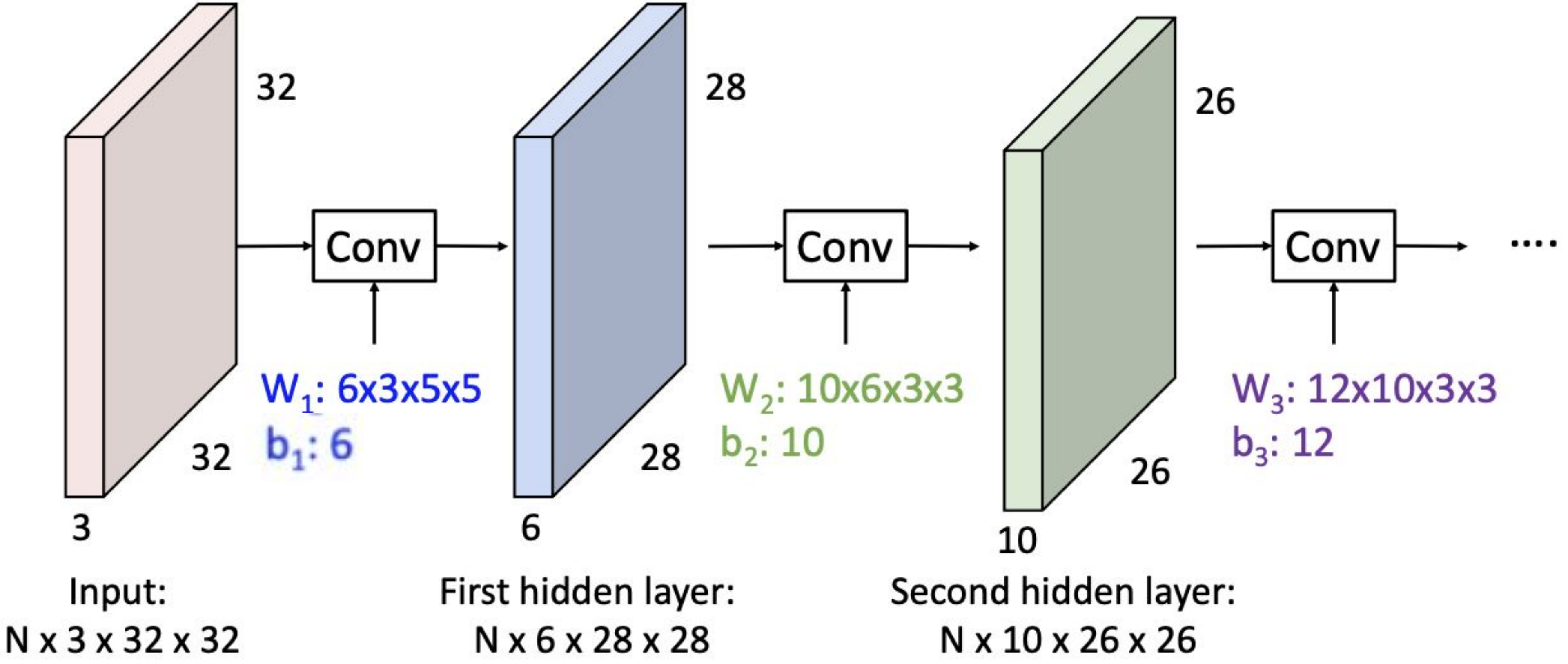
Convolutional Layer



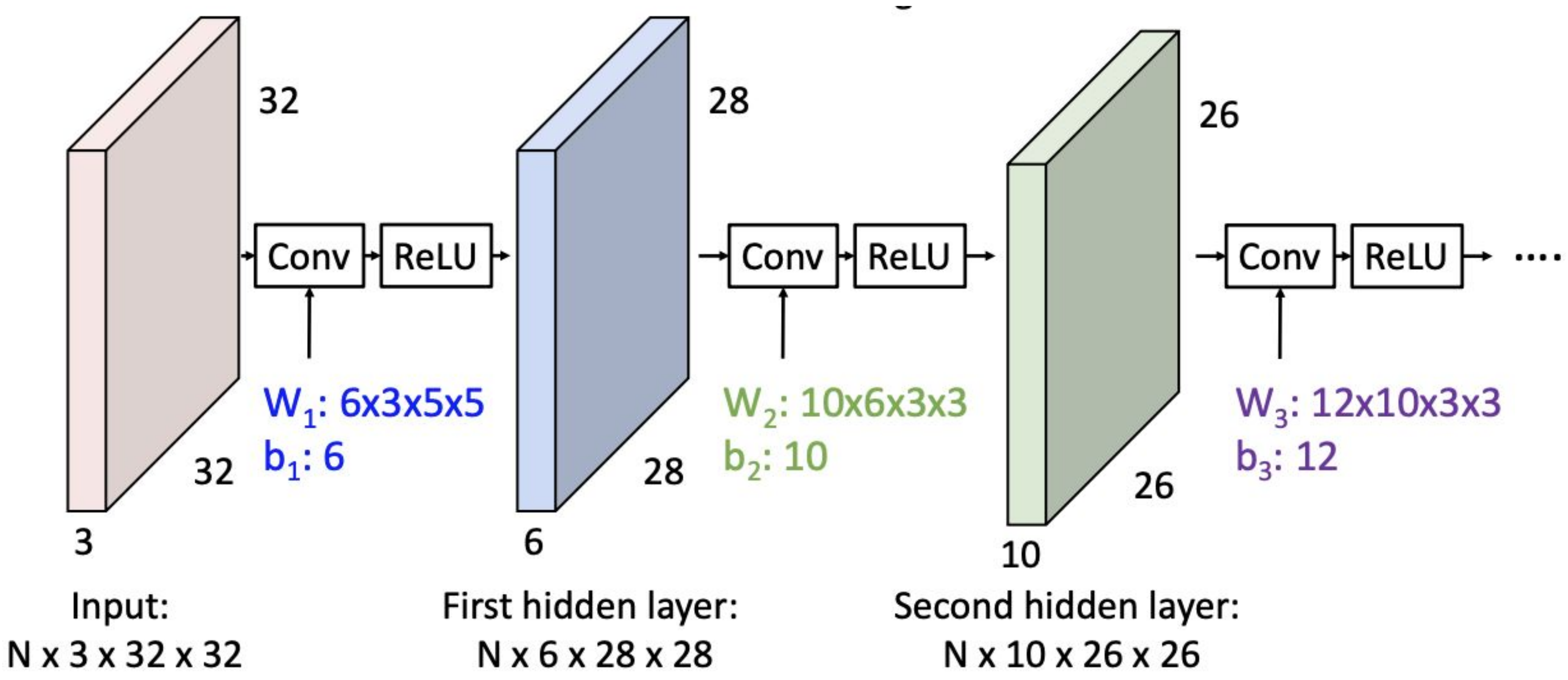
Convolution Layer



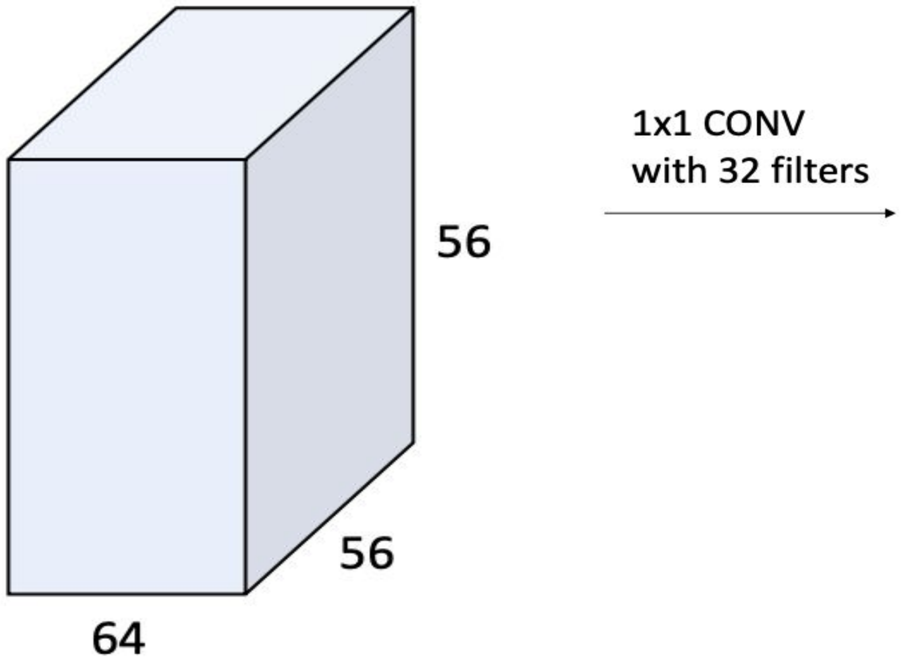
Stacking Convolutions



Stacking Convolutions

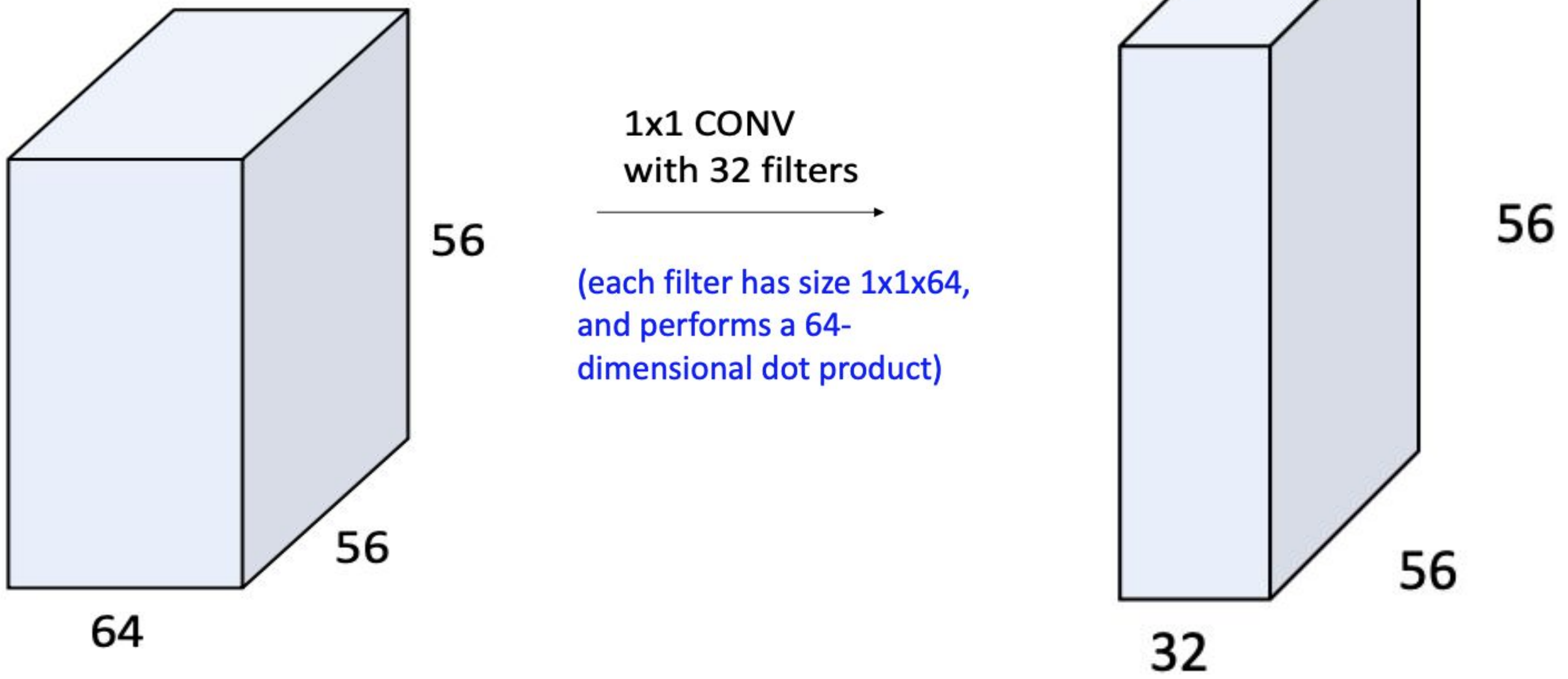


1 x 1 Convolutional Layer



Expected Output size ?

1 x 1 Convolutional Layer



Spatial Dimensions

- Padding and Stride
- Pooling

Padding

- Given a 32 x 32 input image, apply convolutional layer with 5x5 kernel

  ⇨ Output with 1 layer is 28 x 28

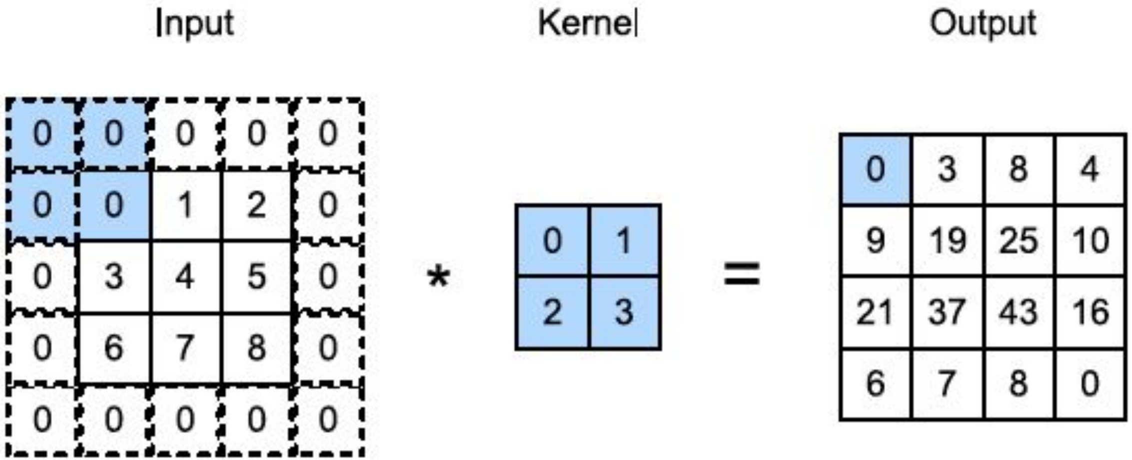
  ⇨ Output with 7 layers is 4 x 4

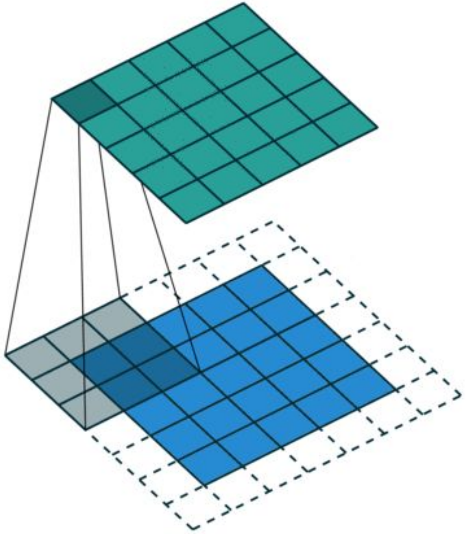
- Shape decreases faster with large kernels

  from $n_h \times n_w$ to $(n_h - k_h + 1) \times (n_w - k_w + 1)$

Padding

- Add zeros around the input





Padding

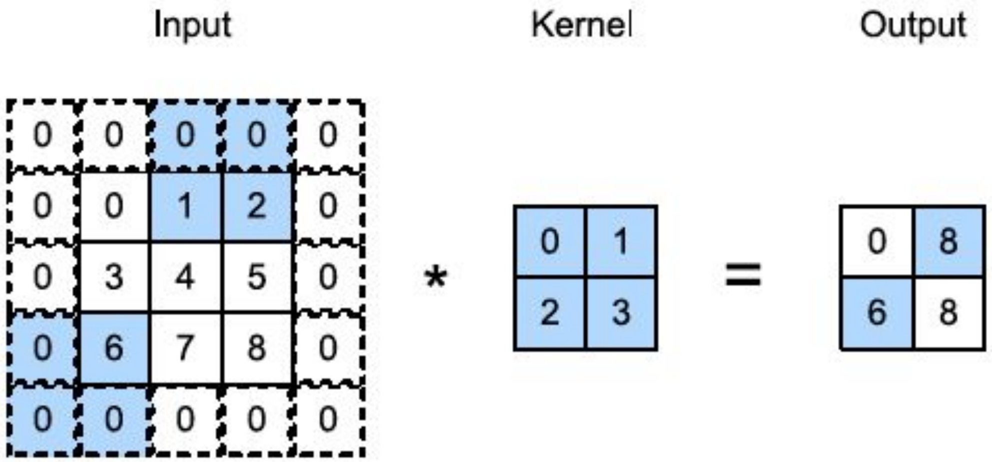
- Padding p_h rows and p_w columns, output shape will be

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

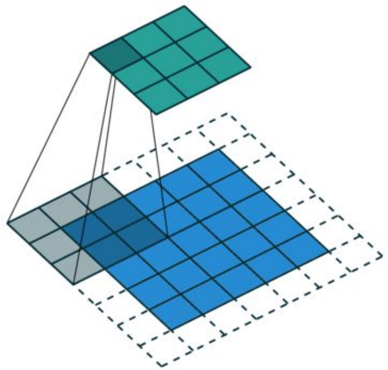
- A common choice is $p_h = k_h - 1$ and $p_w = k_w - 1$
  - Odd k_h : pad $p_h/2$ on both sides
  - Even k_h : pad $\lceil p_h/2 \rceil$ on top, $\lfloor p_h/2 \rfloor$ on bottom

Stride

- Dictates the slide of the convolution window.



Strides of 3 and 2 for height and width



Stride

- Given stride s_h for the height and stride s_w for the width, the output shape is

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

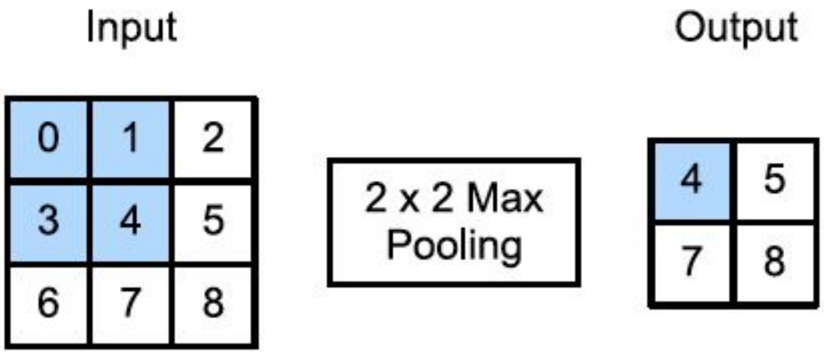
- With $p_h = k_h - 1$ and $p_w = k_w - 1$

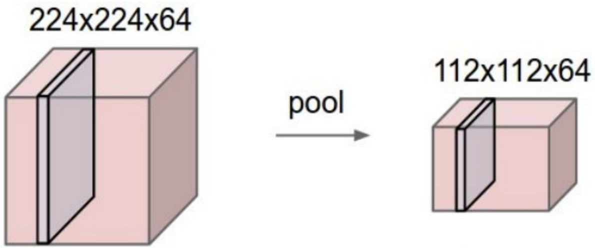
$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

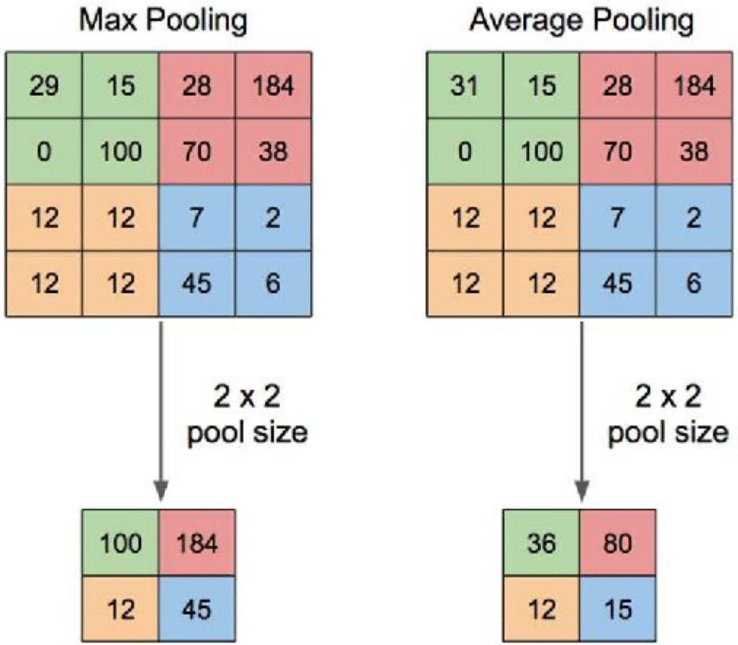
- If input height/width are divisible by strides

$$(n_h/s_h) \times (n_w/s_w)$$

Pooling:


Input
0 1 2
3 4 5
6 7 8
2 x 2 Max Pooling
Output
4 5
7 8


224x224x64
pool
112x112x64


Max Pooling
29 15 28 184
0 100 70 38
12 12 7 2
12 12 45 6
2 x 2
pool size
100 184
12 45
Average Pooling
31 15 28 184
0 100 70 38
12 12 7 2
12 12 45 6
2 x 2
pool size
36 80
12 15

PyTorch Code:

- Padding
- Stride
- Pooling

Convolutional Summary

Input: C_{in} x H x W

Hyperparameters:

- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Weight matrix: C_{out} x C_{in} x K_H x K_W
giving C_{out} filters of size C_{in} x K_H x K_W

Bias vector: C_{out}

Output size: C_{out} x H' x W' where:

- $H' = (H - K + 2P) / S + 1$
- $W' = (W - K + 2P) / S + 1$

Common settings:

$K_H = K_W$ (Small square filters)

$P = (K - 1) / 2$ ("Same" padding)

C_{in}, C_{out} = 32, 64, 128, 256 (powers of 2)

K = 3, P = 1, S = 1 (3x3 conv)

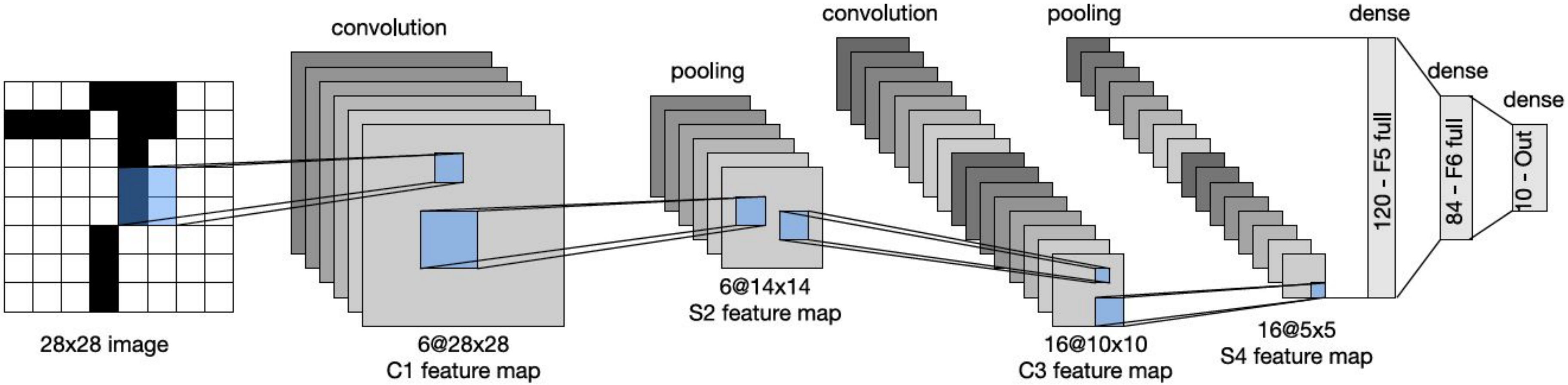
K = 5, P = 2, S = 1 (5x5 conv)

K = 1, P = 0, S = 1 (1x1 conv)

K = 3, P = 1, S = 2 (Downsample by 2)

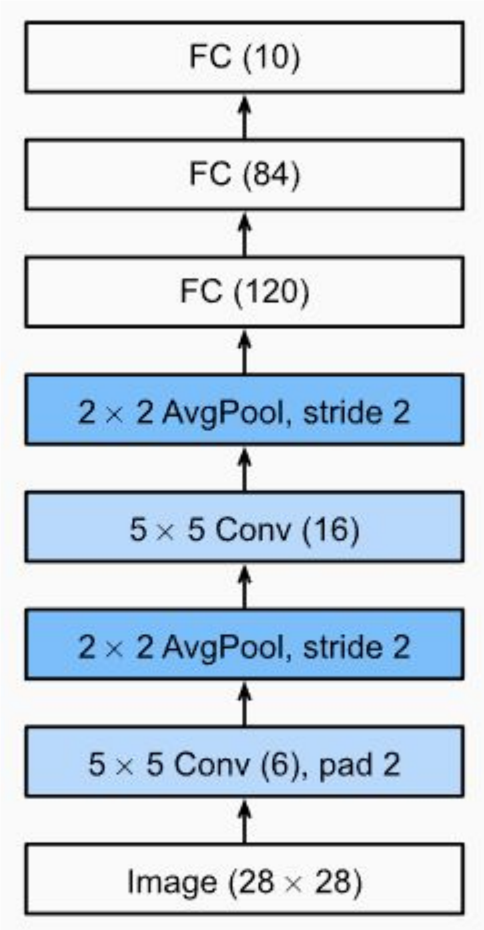
LeNet Architecture

One of the first successful applications of CNN developed by Yann LeCun in the 1990’s.



LeNet Architecture

```
Reshape output shape:        torch.Size([1, 1, 28, 28])
Conv2d output shape:         torch.Size([1, 6, 28, 28])
Sigmoid output shape:        torch.Size([1, 6, 28, 28])
AvgPool2d output shape:      torch.Size([1, 6, 14, 14])
Conv2d output shape:         torch.Size([1, 16, 10, 10])
Sigmoid output shape:        torch.Size([1, 16, 10, 10])
AvgPool2d output shape:      torch.Size([1, 16, 5, 5])
Flatten output shape:        torch.Size([1, 400])
Linear output shape:         torch.Size([1, 120])
Sigmoid output shape:        torch.Size([1, 120])
Linear output shape:         torch.Size([1, 84])
Sigmoid output shape:        torch.Size([1, 84])
Linear output shape:         torch.Size([1, 10])
```



PyTorch Code:

- LeNet Architecture
  on Fashion-MNIST

Questions / Discussion

References:

- Material is from the book Dive into Deep Learning
- Some visualizations are from:
    - https://web.eecs.umich.edu/~justincj/slides/eecs498/FA2020/598_FA2020_lecture07.pdf
    - https://github.com/vdumoulin/conv_arithmetic