CS 760: Machine Learning
Convolutional Neural Networks

University of Wisconsin-Madison

Outline

- **Convolutional Neural Networks (CNN) Basics**
    - motivation, convolution operator
- **CNN Components & Layers**
    - padding, stride, dilation, channels, pooling layers
- **CNN Tasks & Architectures**
    - MNIST, ImageNet, LeNet, AlexNet, ResNet

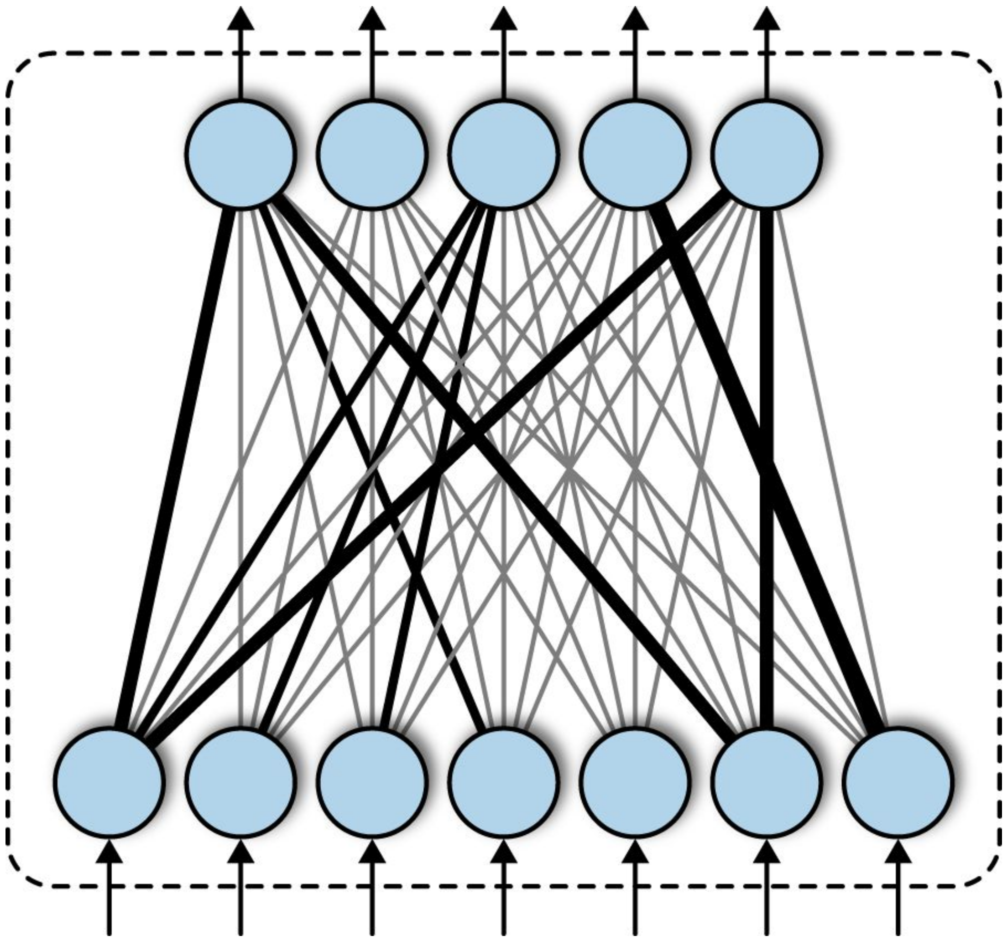
Outline

- **Convolutional Neural Networks (CNN) Basics**
  - motivation, convolution operator
- **CNN Components & Layers**
  - padding, stride, dilation, channels, pooling layers
- **CNN Tasks & Architectures**
  - MNIST, ImageNet, LeNet, AlexNet, ResNet

Review: Multi-layer perceptrons (MLPs)

- So far we've been using MLP networks, which consist of compositions of **fully-connected layers**, so named because every input unit is connected to every output unit

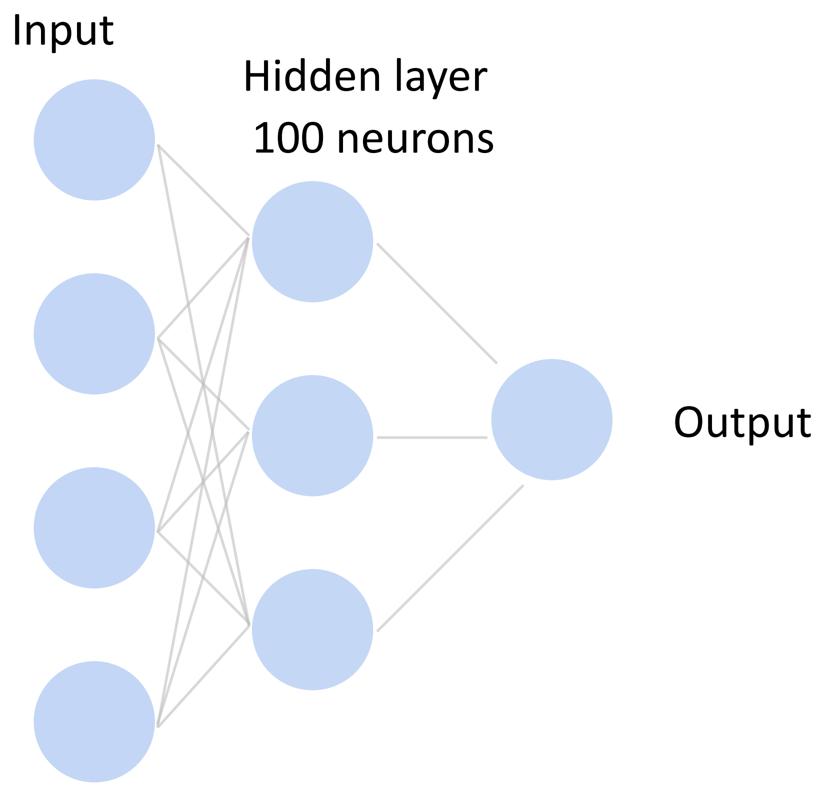
$$h^{L+1} = \sigma(Wh^L + b)$$



What if we have images as our inputs?

36M floats in a RGB image!

What if we have images as our inputs?



~ 36M input elements x 100 = ~**3.6B** parameters!

Convolutions to the rescue

- Convolution layers

- can process images with varying numbers of pixels

- have a parameter count that doesn't increase with image resolution, unlike $O(wh)$ or more for fully connected layers

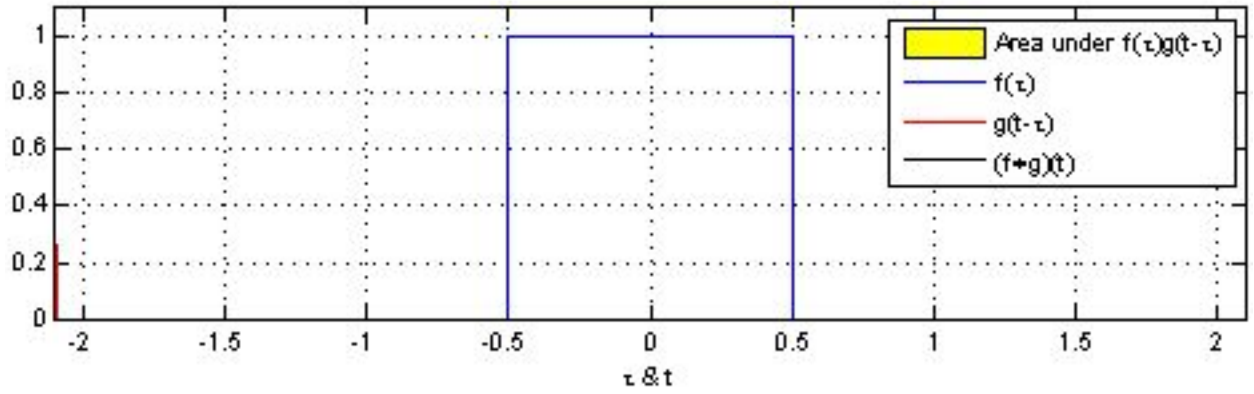
- have computational complexity $\tilde{O}(w + h)$ rather than $O(wh)$ or worse for fully connected layers

- are **translation equivariant**, i.e. extract the same feature from a translation of the image

Background: Convolution Operation

- Given array u_t and w_t, their convolution is a function s_t

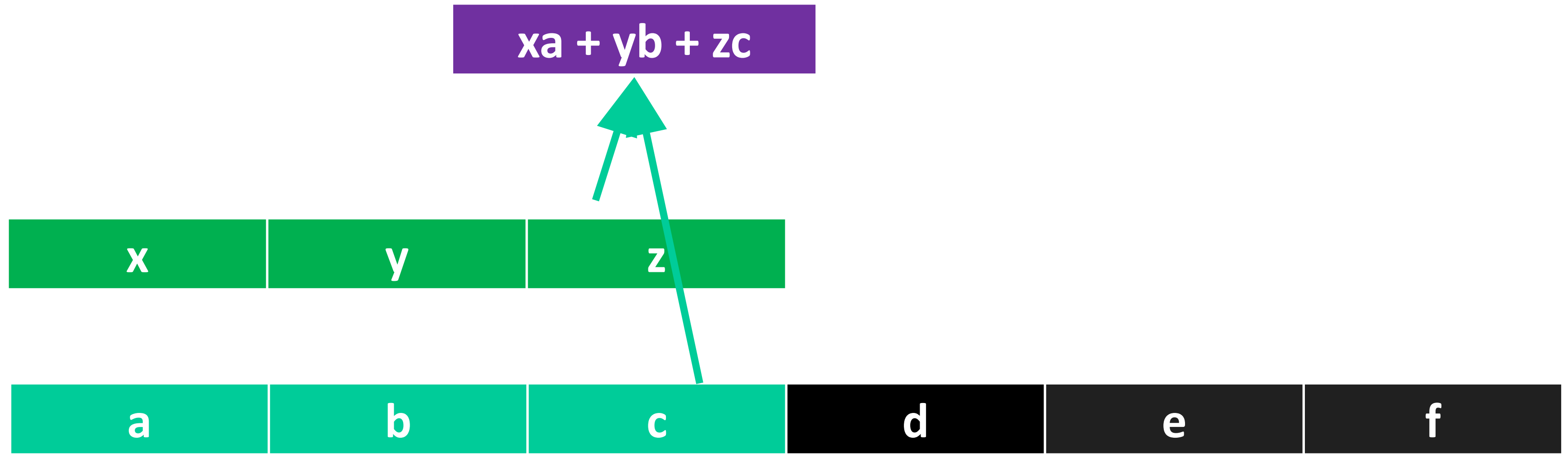
- Written as $s = (u * w)$ or $s_t = (u * w)_t$
- When u_t or w_t is not defined, assumed to be 0



Background: Convolution Operation

- Example:

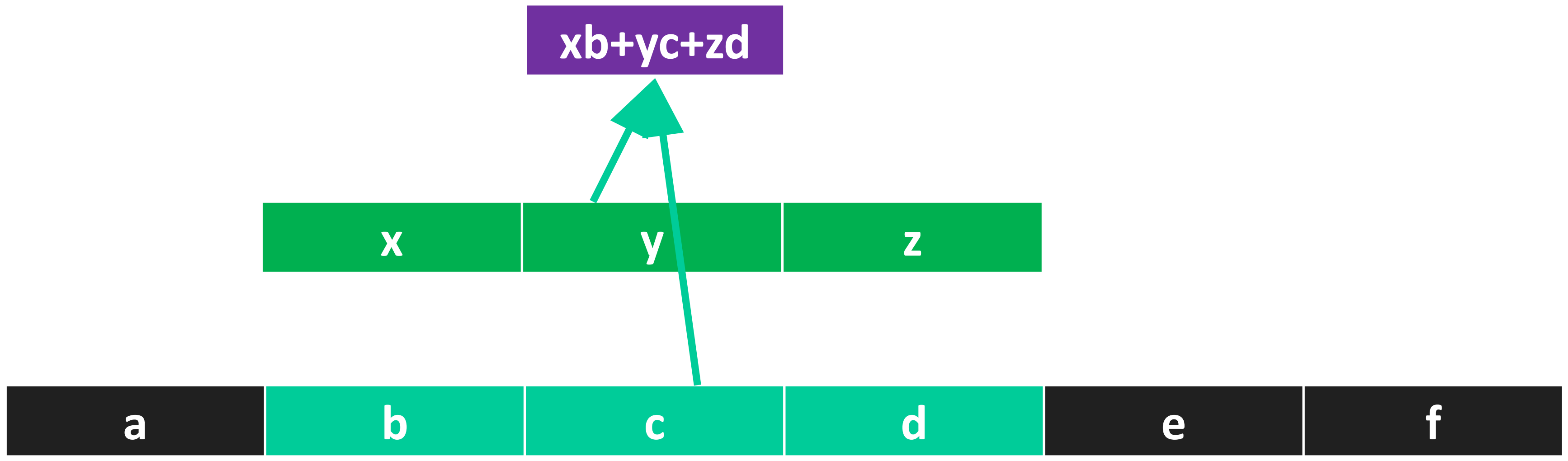
w = [z, y, x]
u = [a, b, c, d, e, f]



Background: Convolution Operation

- Example:

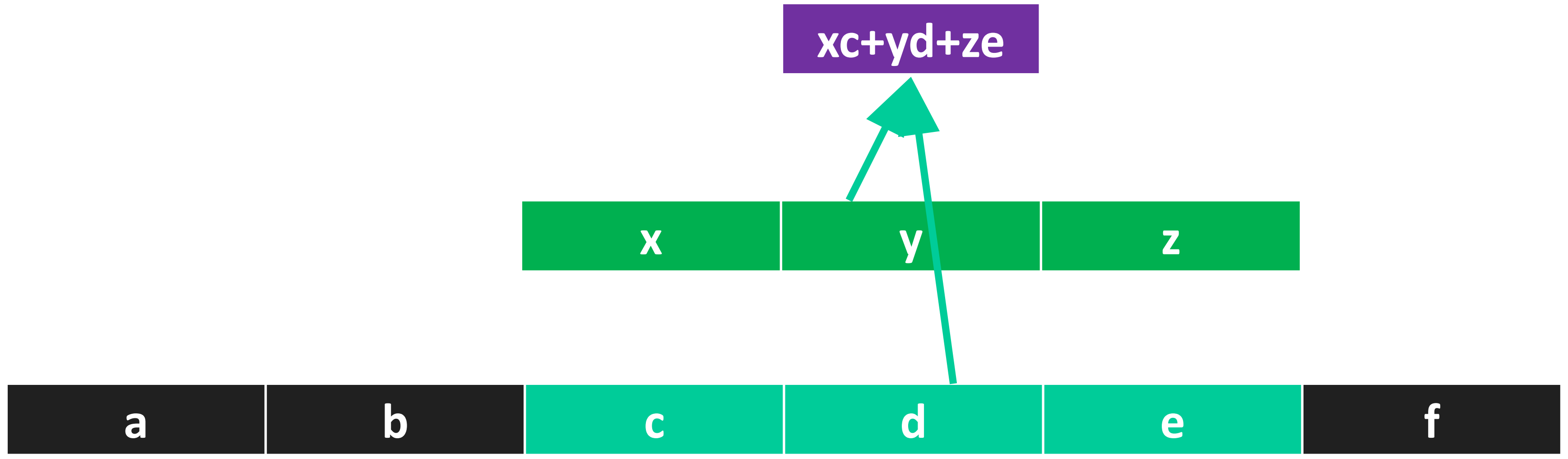
w = [z, y, x]
u = [a, b, c, d, e, f]



Background: Convolution Operation

- Example:

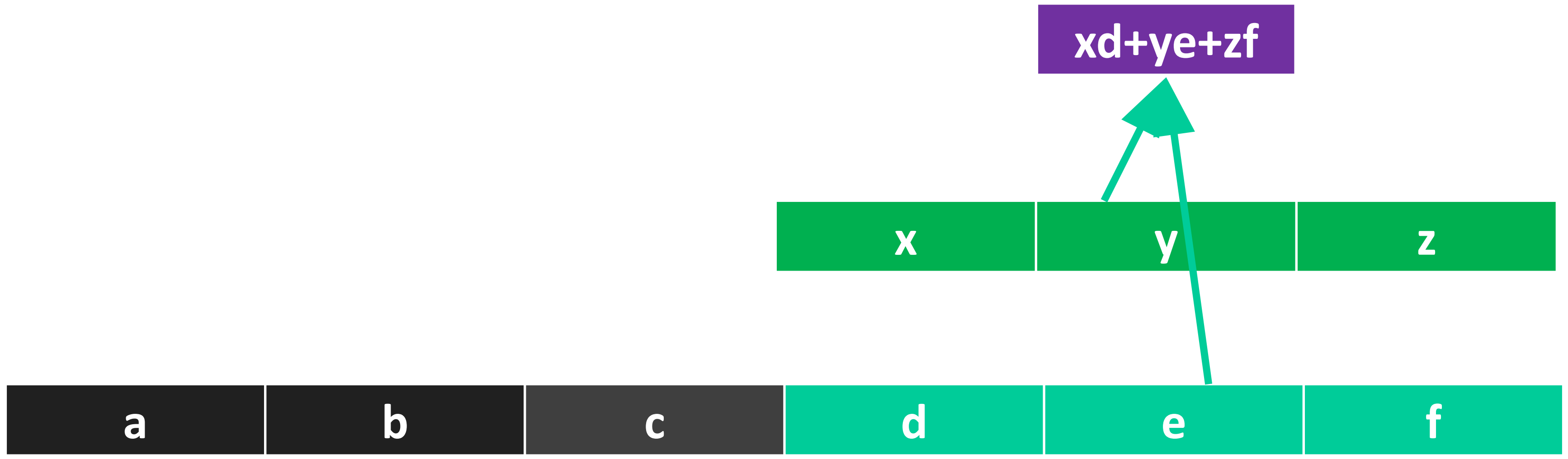
w = [z, y, x]
u = [a, b, c, d, e, f]



Background: Convolution Operation

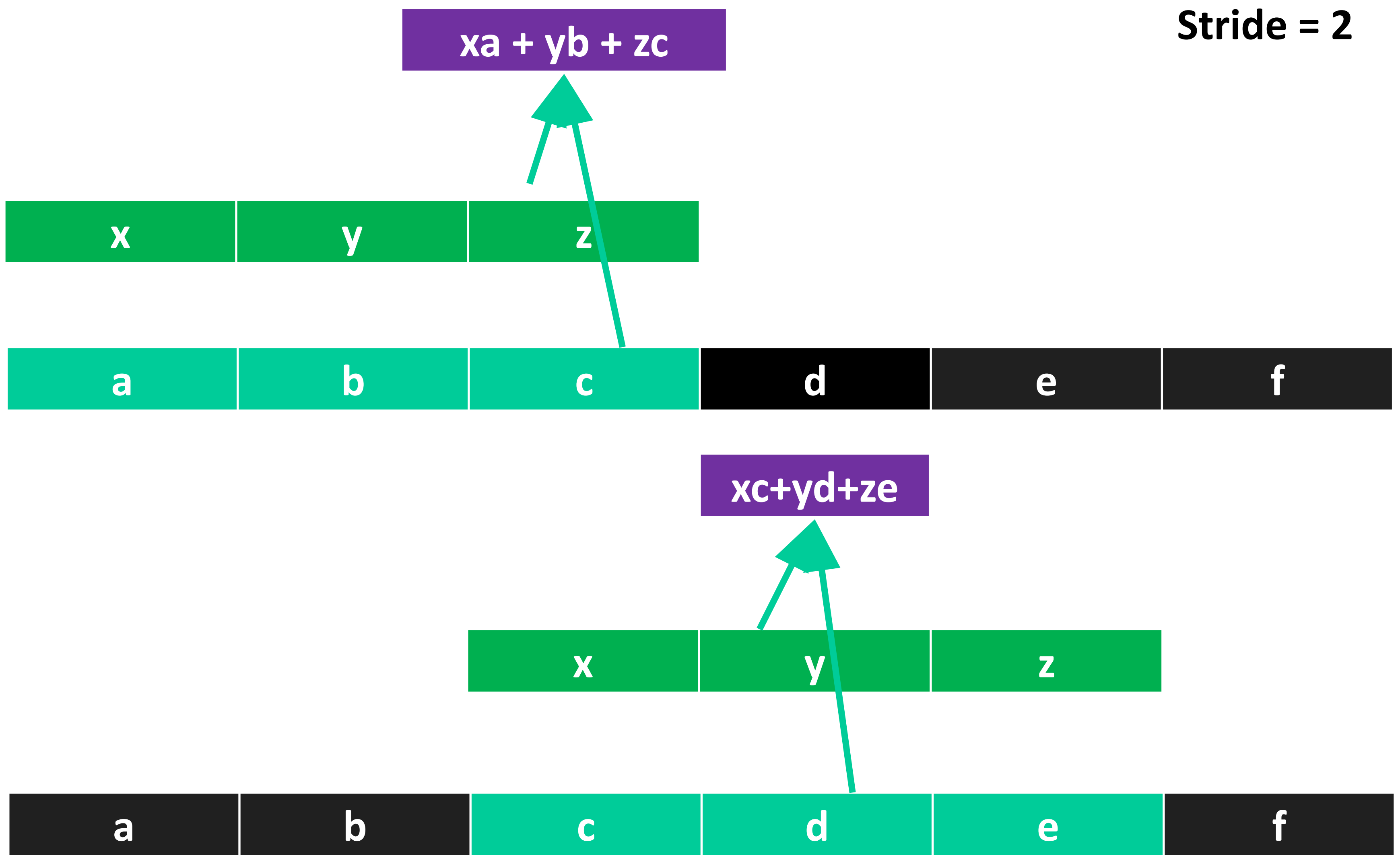
- Example:

w = [z, y, x]
u = [a, b, c, d, e, f]



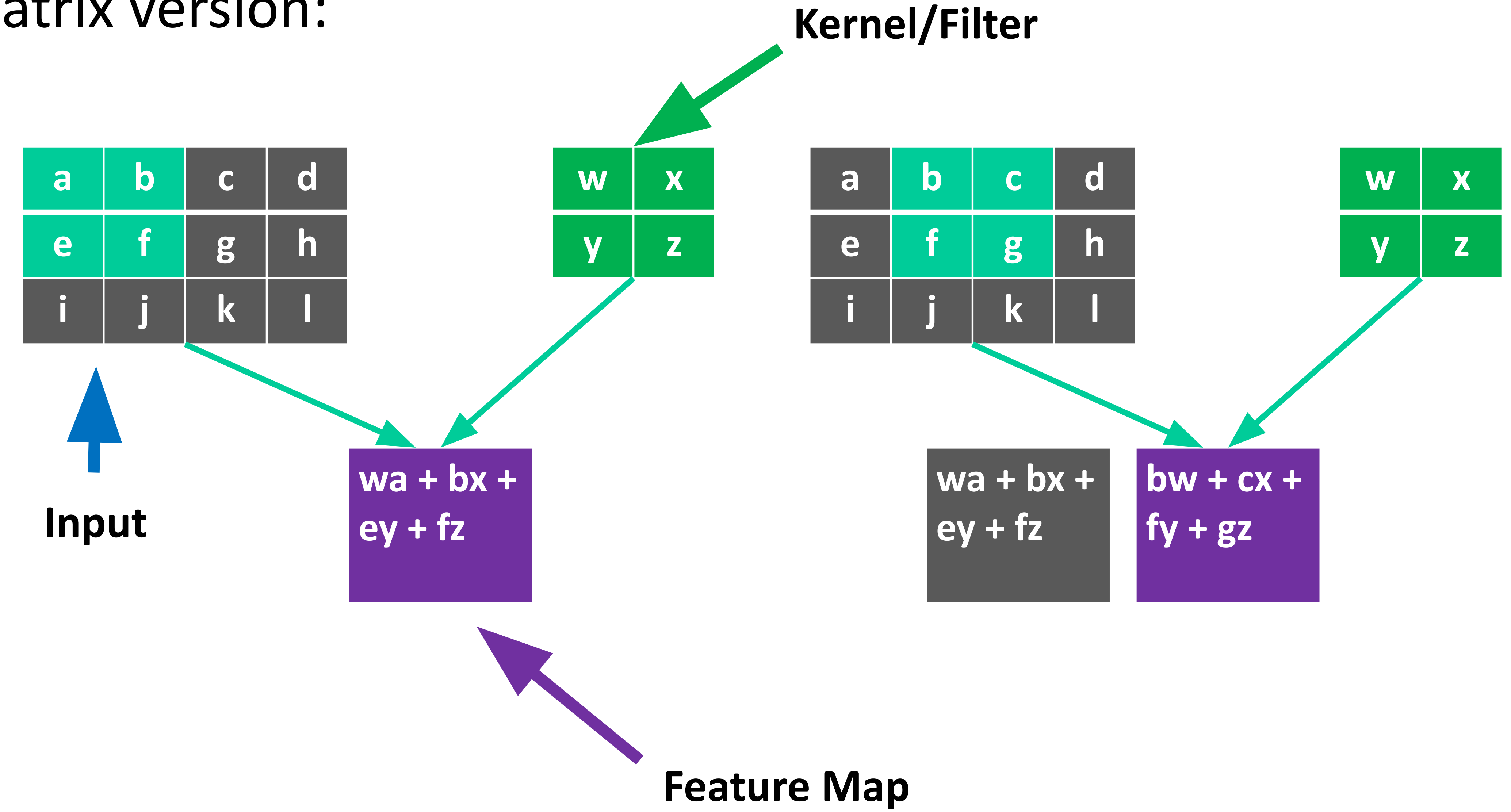
Background: Convolution Operation

- Stride: # of positions we move per step



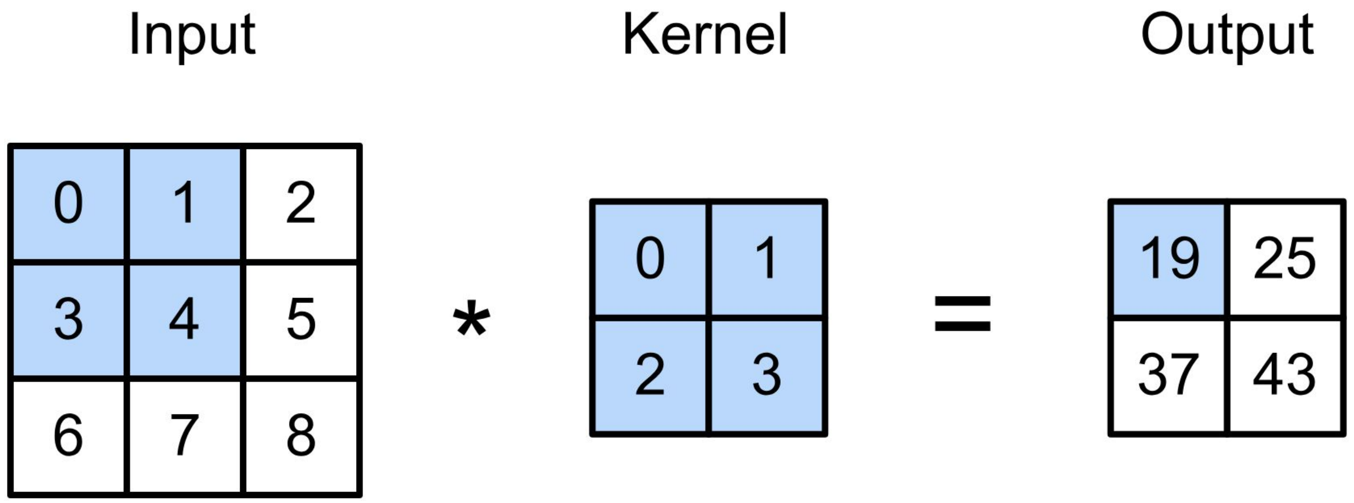
Background: Convolution Operation

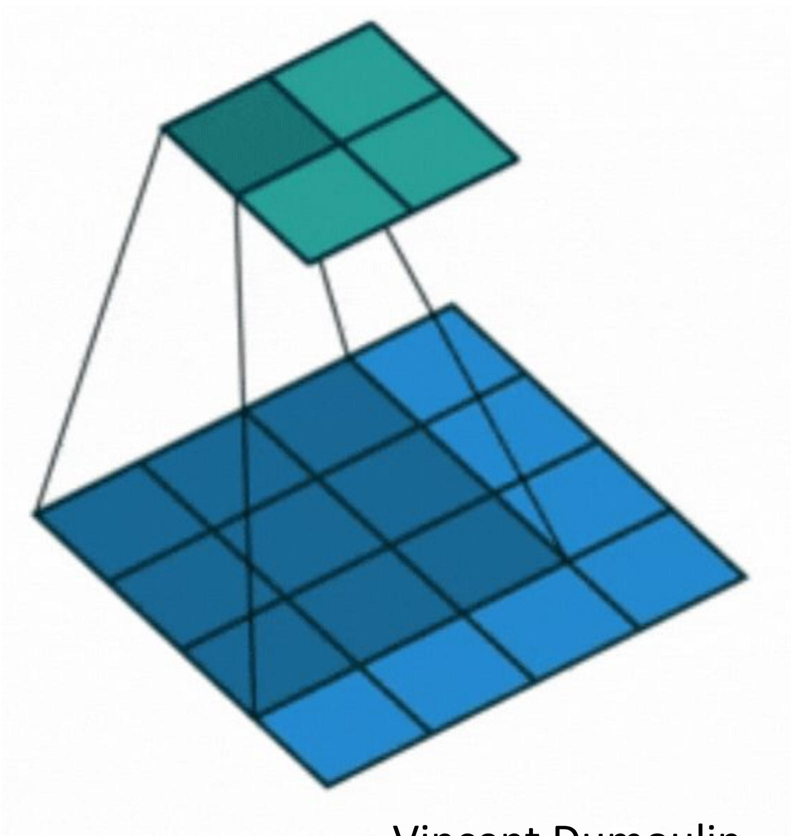
- Matrix version:



2-D Convolutions

Example:

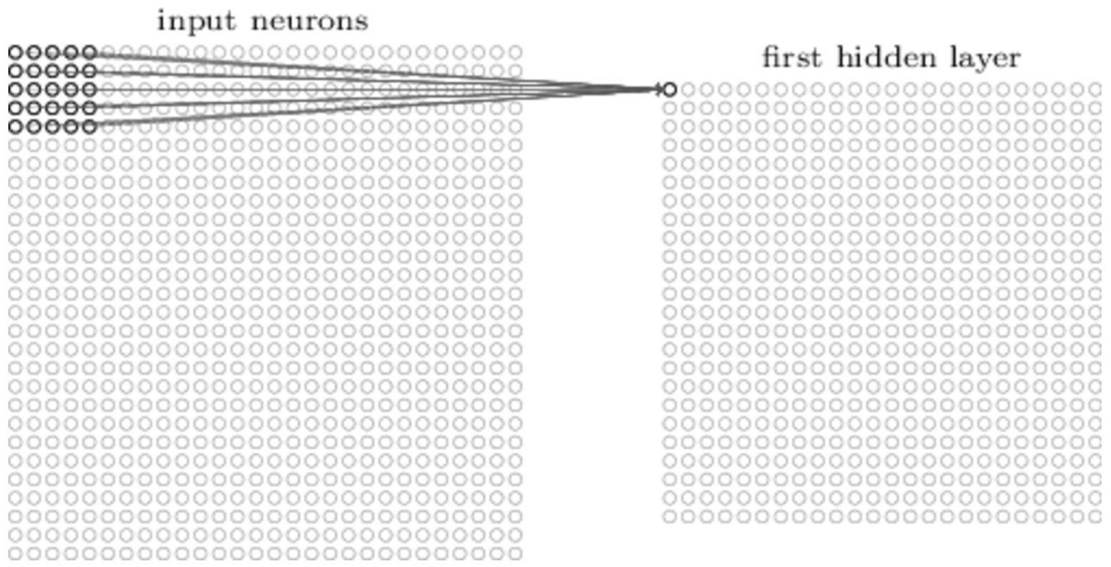


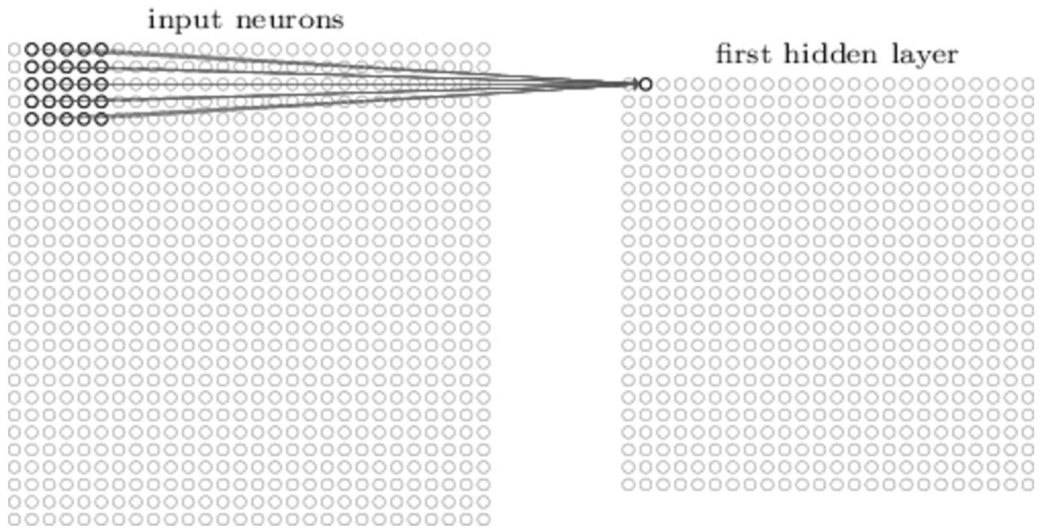

Vincent Dumoulin

$$
\begin{aligned}
0\times0+1\times1+3\times2+4\times3&=19,\\
1\times0+2\times1+4\times2+5\times3&=25,\\
3\times0+4\times1+6\times2+7\times3&=37,\\
4\times0+5\times1+7\times2+8\times3&=43.
\end{aligned}
$$

Convolution Operation

- All the units used the same set of weights (kernel)
- The units detect the same "feature" but at different locations



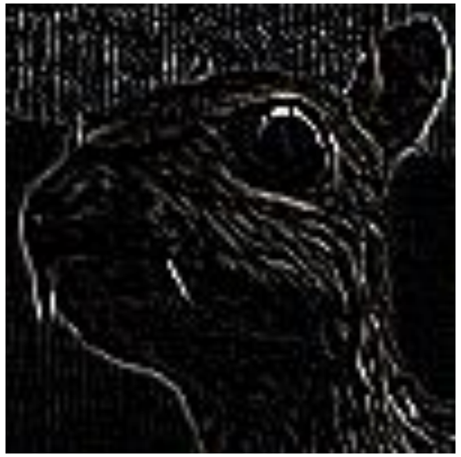


neuralnetworksanddeeplearning.com

Kernels: Examples

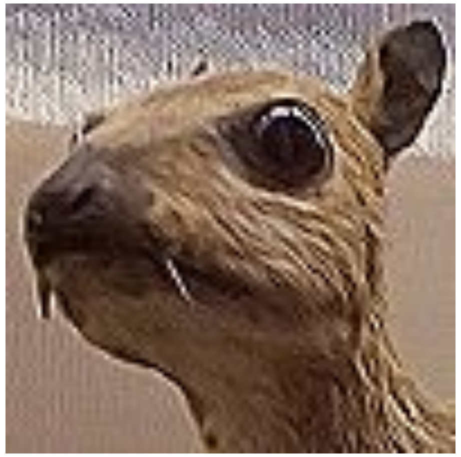
(Wikipedia)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



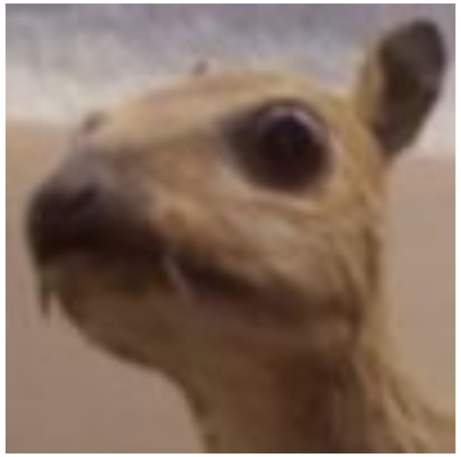
Edge Detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Sharpen

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

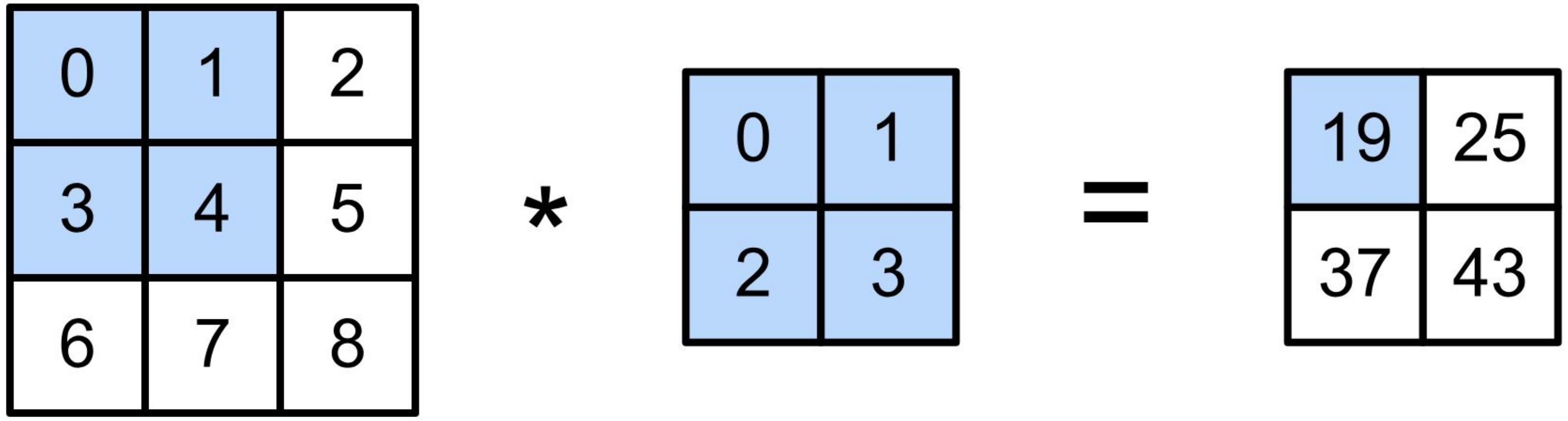


Gaussian Blur

Convolution Layers

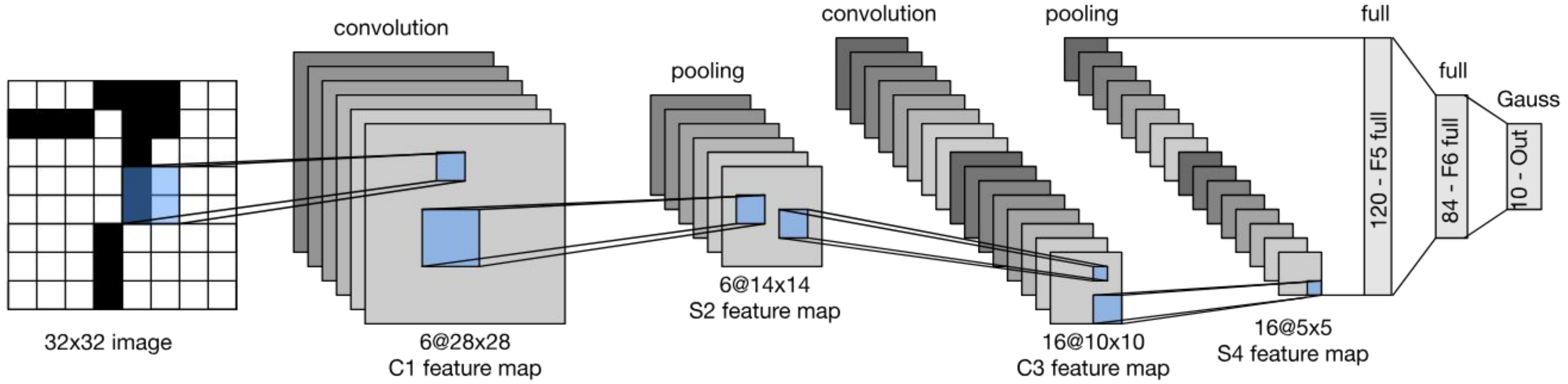
- Notation:
  - *X*: n_h x n_w input matrix
  - *W*: k_h x k_w kernel matrix
  - *b* : bias (a scalar)

- As usual *W*, *b* are learnable parameters



Convolutional Neural Networks

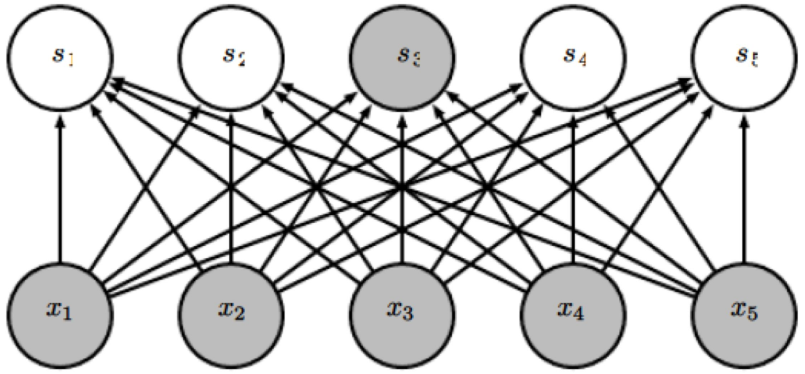
Convolutional networks: neural networks that use ***convolution*** in place of general matrix multiplication in at least one layer



- default approach for image tasks
- still used even in modern Transformer alternatives

CNNs: Advantages

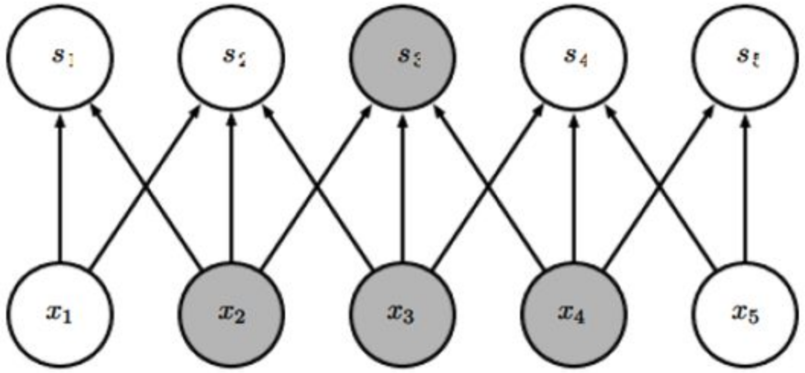
- Fully connected layer: *m* x *n* edges and parameters



m output nodes

n input nodes

- Convolutional layer: ≤ *m* x *k* edges, *k* parameters



m output nodes

k kernel size

n input nodes

Break & Quiz

Q: If the input size is NxN and the kernel/filter size is KxK, what is the size of the output matrix after performing convolution? Assume N>K, no padding, and stride (how much we move the kernel each time) = 1.

1. (N - K + 1) x (N - K + 1)
2. (N - K) x (N - K)
3. (N - K - 1) x (N - K - 1)
4. None of the above

Q: If the input size is NxN and the kernel/filter size is KxK, what is the size of the output matrix after performing convolution? Assume N>K, no padding, and stride (how much we move the kernel each time) = 1.

1. **(N - K + 1) x (N - K + 1)**
2. (N - K) x (N - K)
3. (N - K - 1) x (N - K - 1)
4. None of the above

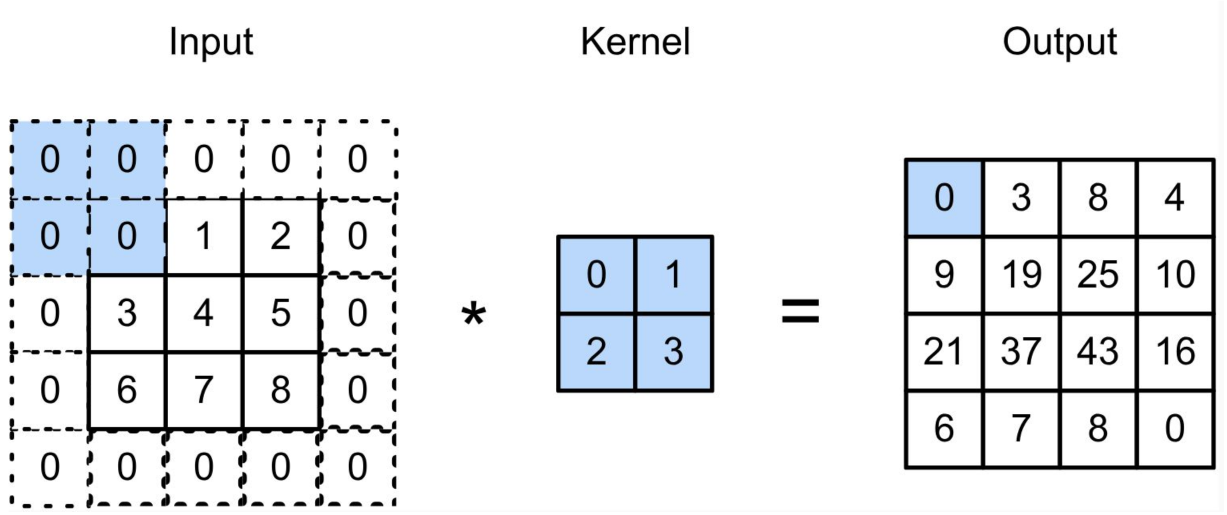
- when sliding to the right, we have N-K+1 positions
- similar when sliding downwards

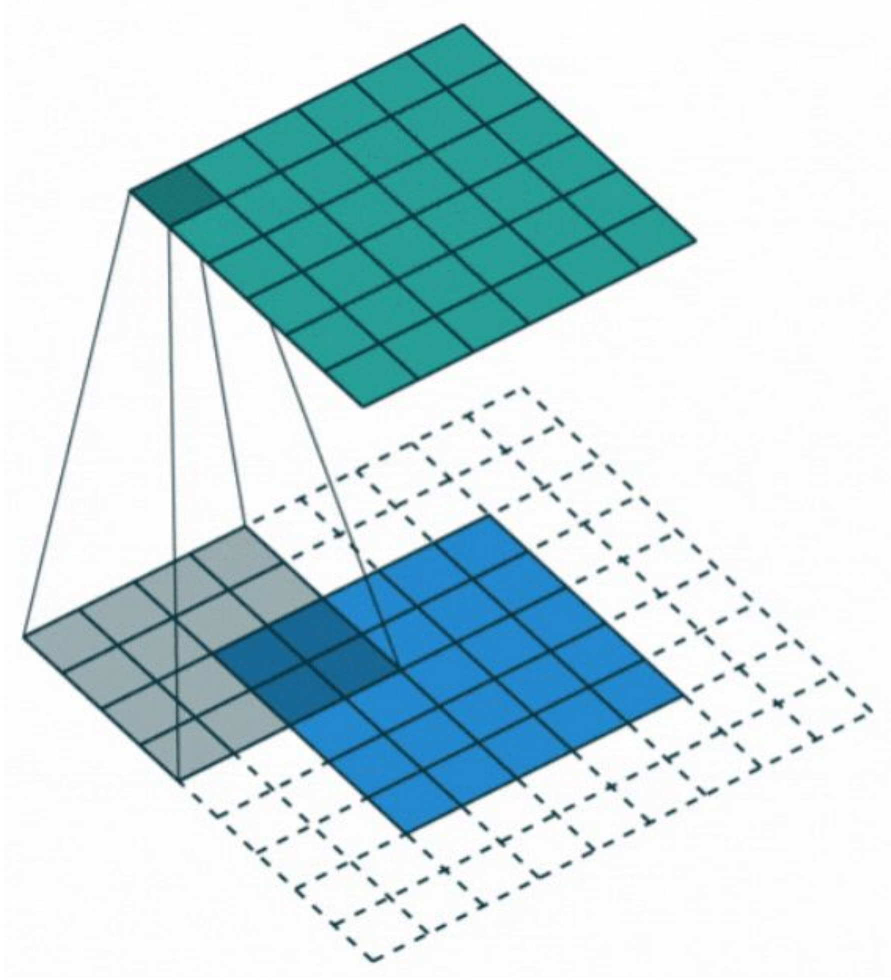
Outline

- **Convolutional Neural Networks (CNN) Basics**
  - motivation, convolution operator
- **CNN Components & Layers**
  - padding, stride, dilation, channels, pooling layers
- **CNN Tasks & Architectures**
  - MNIST, ImageNet, LeNet, AlexNet, ResNet

Convolutional Layers: Padding

Padding adds rows/columns around input





Convolutional Layers: Padding

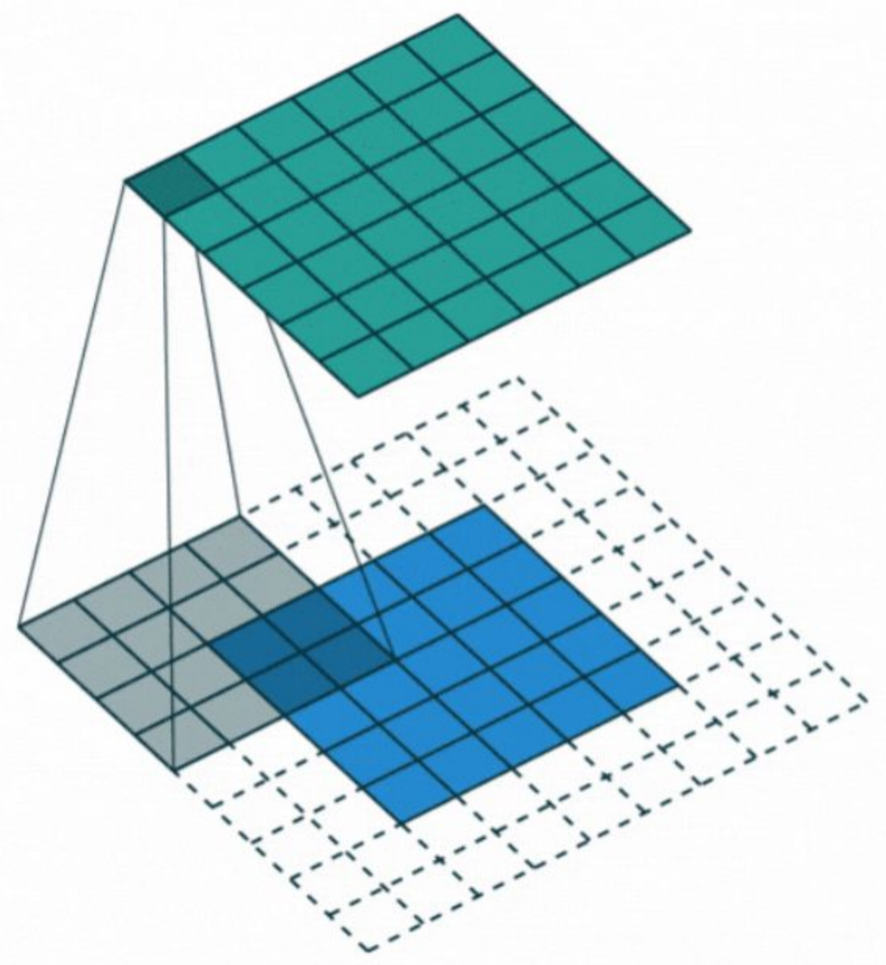
Padding adds rows/columns around input

Why?

1. Keeps **edge information**

2. Preserves sizes / allows deep networks
   - i.e. for a 32x32 input image, 5x5 kernel, after 1 layer, get 28x28, after 7 layers, **only 4x4**

3. Can combine different filter sizes



Convolutional Layers: Padding

- Padding p_h rows and p_w columns, output shape is

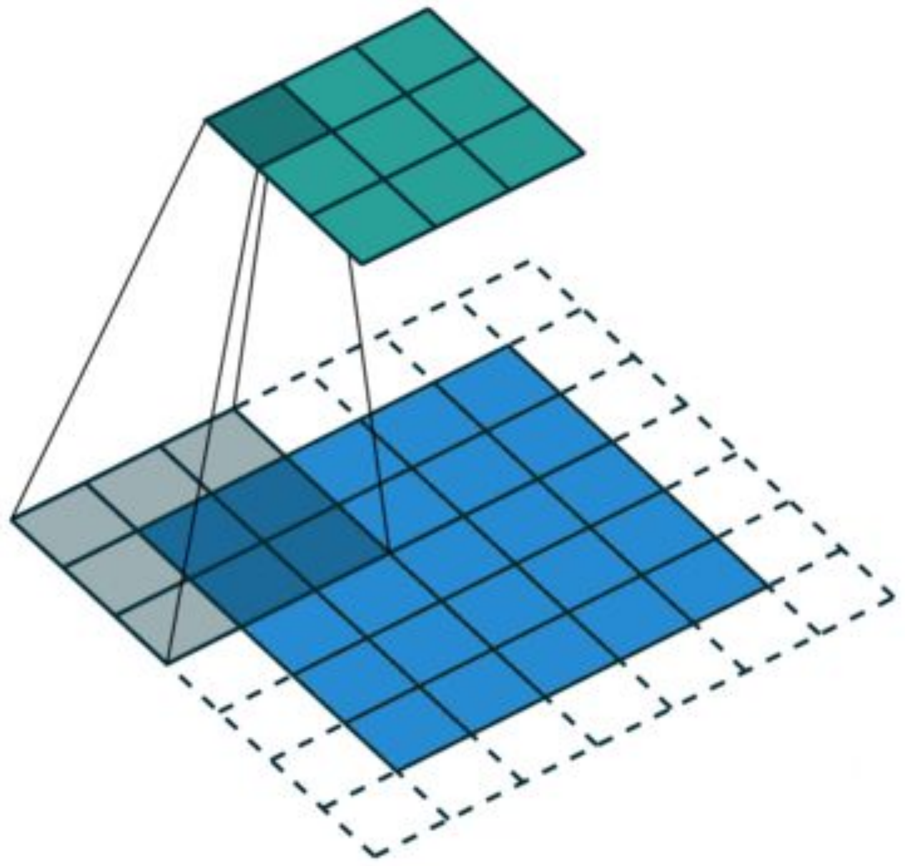
$$(n_h - k_h + p_h + 1) \text{ x } (n_w - k_w + p_w + 1)$$

- 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 ceil($p_h/2$) on top, floor($p_h/2$) on bottom

Convolutional Layers: Stride

Stride: #rows / #columns per slide

- On the right is a 3x3 kernel with row stride = column stride = 2



Convolutional Layers: Stride

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

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

- Set $p_h = k_h - 1$, $p_w = k_w - 1$, then get

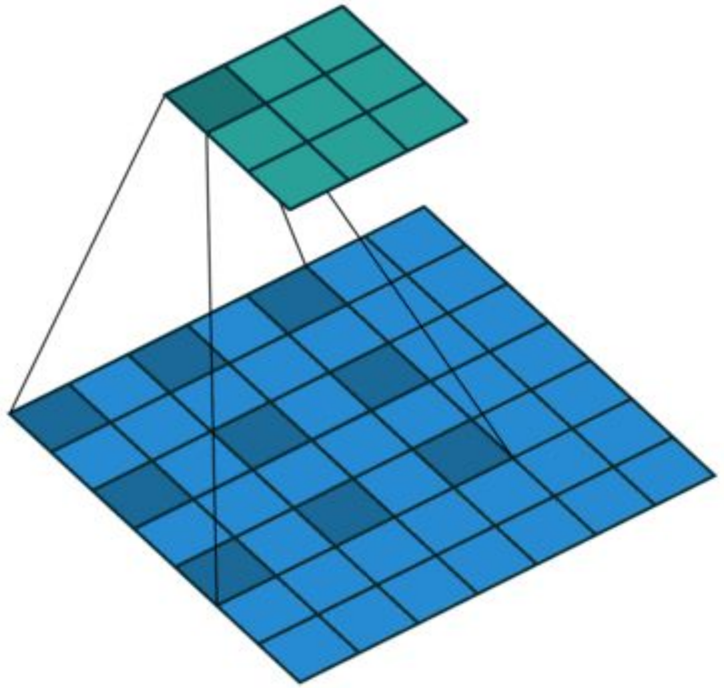
$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

Convolutional Layers: Dilated kernels

- Dilation rate: sets gaps between kernel elements

- on the right is a 3x3 kernel with dilation rate 2

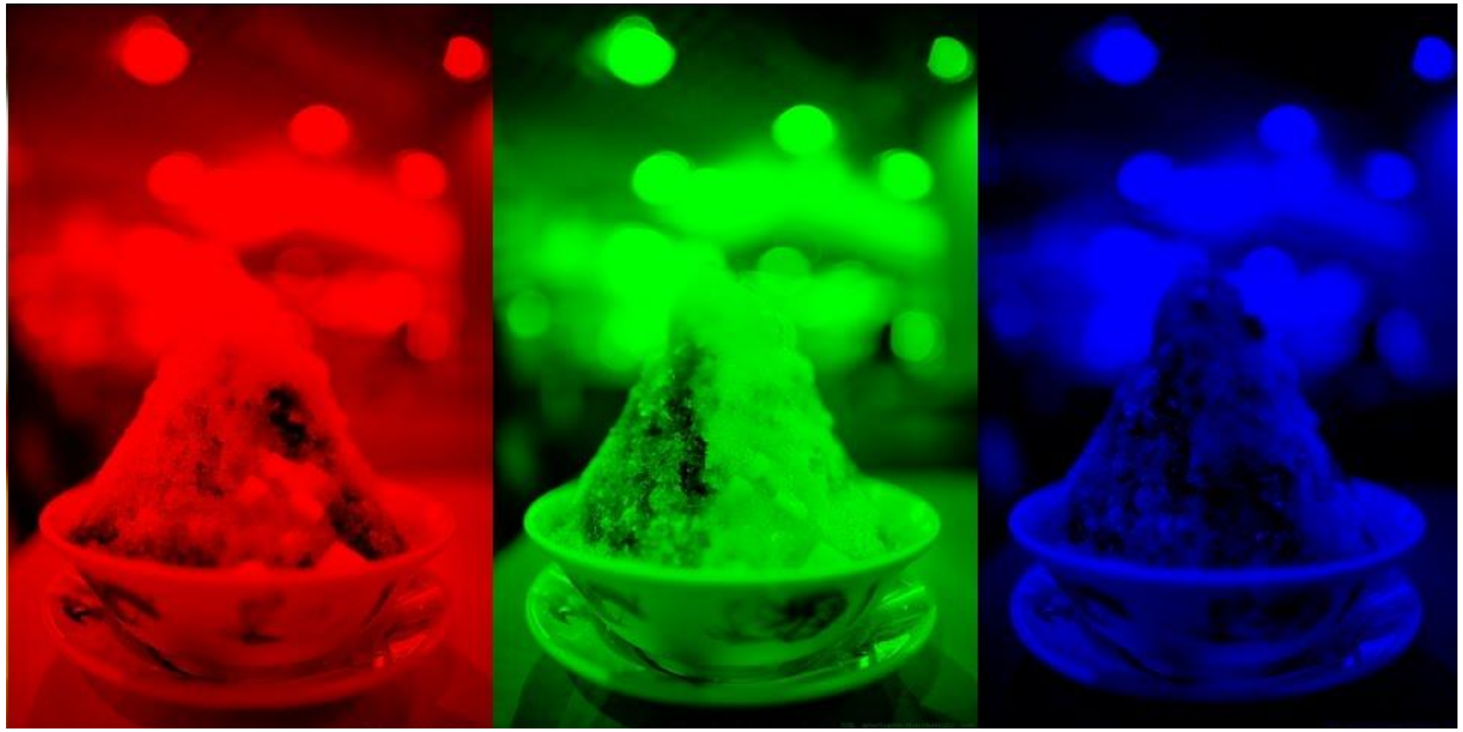
- a $k \times k$ kernel with dilation rate d is effectively a $(d(k-1)+1) \times (d(k-1)+1)$ kernel with the extra elements set to zero



Convolutional Layers: Channels

Color images are multi-channel, e.g. RGB:

=



Convolutional Layers: Channels

- How to integrate multiple channels?
- Have a kernel for each channel i, then sum results over c_i channels

$$\mathbf{X} : c_i \times n_h \times n_w$$
$$\mathbf{W} : c_i \times k_h \times k_w$$
$$\mathbf{Y} : m_h \times m_w$$

$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$

Convolutional Layers: Channels

- We can also have multiple **output** channels c_o
  - have a kernel for each of $c_i \times c_o$ pairs (i, o) of input channel i and output channel o
  - output channel o gets the sum over $i = 1, \dots, c_i$ over the applications of the kernels (i, o)

$$\mathbf{X} : c_i \times n_h \times n_w$$

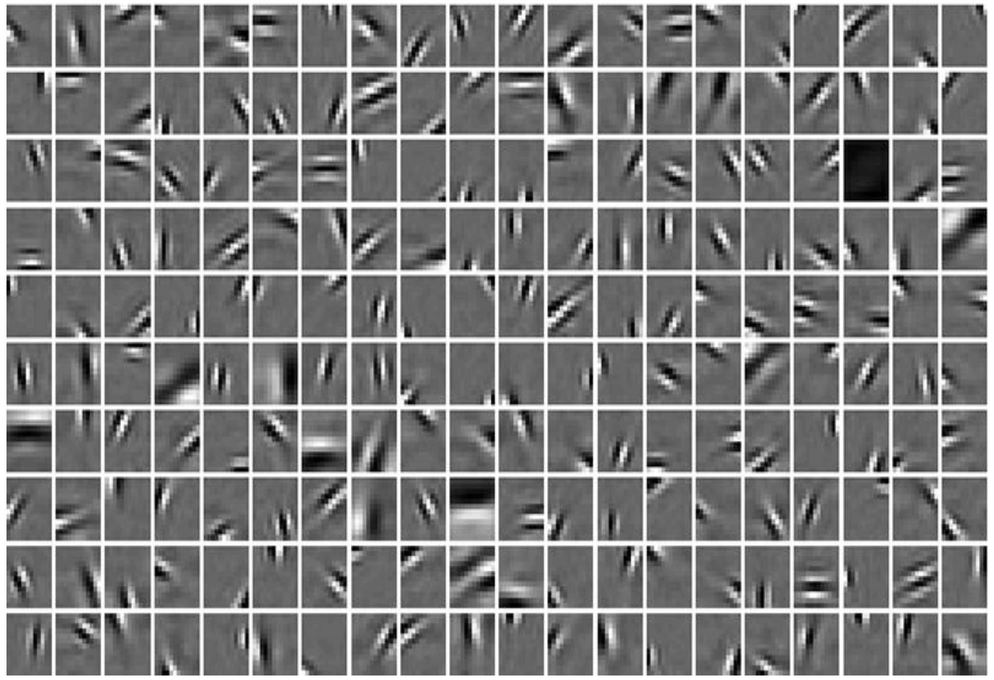
$$\mathbf{W} : c_o \times c_i \times k_h \times k_w$$

$$\mathbf{Y} : c_o \times m_h \times m_w$$

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$$

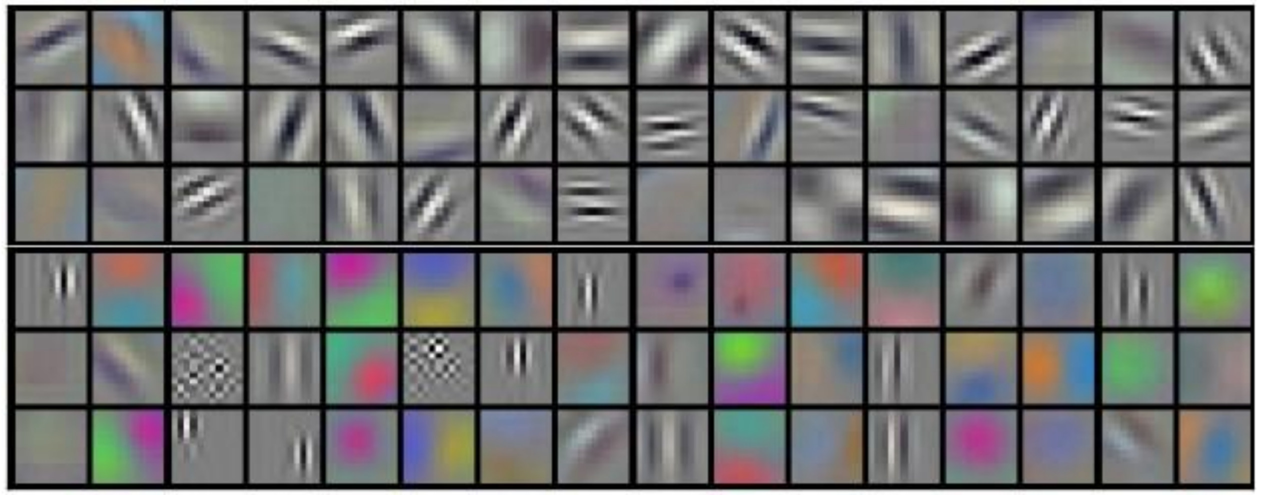
Convolutional Layers: Multiple Kernels

- Each kernel may recognize a particular pattern
  - Gabor filters



(Olshausen & Field, 1997)

[From Human Brain]



Krizhevsky et al

[Machine Learned CNN]

Convolutional Layers: Summary

Properties

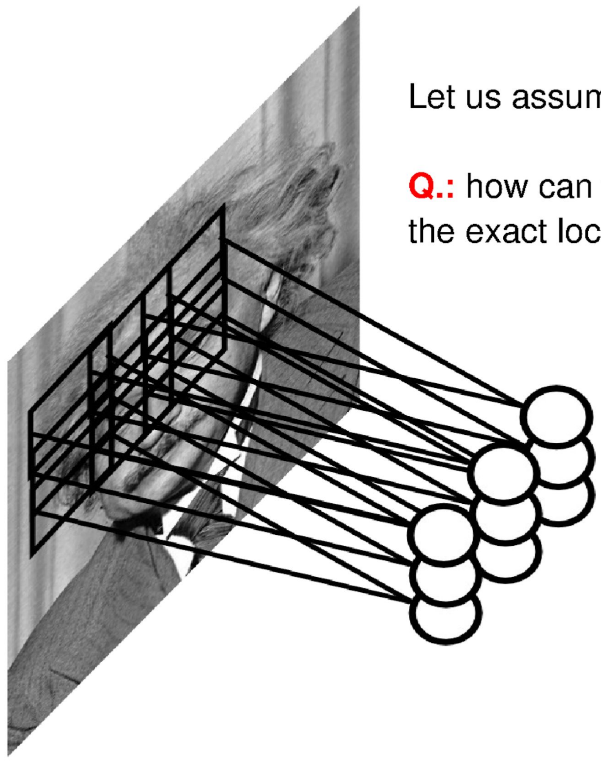
- **Input**: volume c_i x n_h x n_w (channels x height x width)
- **Hyperparameters**: # of kernels / filters c_o, size k_h x k_w, stride s_h x s_w, dilation rate d, zero padding p_h x p_w
- **Output**: volume c_o x m_h x m_w (channels x height x width)
- **Parameters**: k_h x k_w x c_i per filter, total $(k_h$ x k_w x $c_i)$ x c_o

How to pick these hyperparameters?

- trial-and-error
- hyperparameter tuning / architecture search
- often easiest to just use an off-the-shelf CNN

Other CNN Layers: Pooling

- Another type of layer

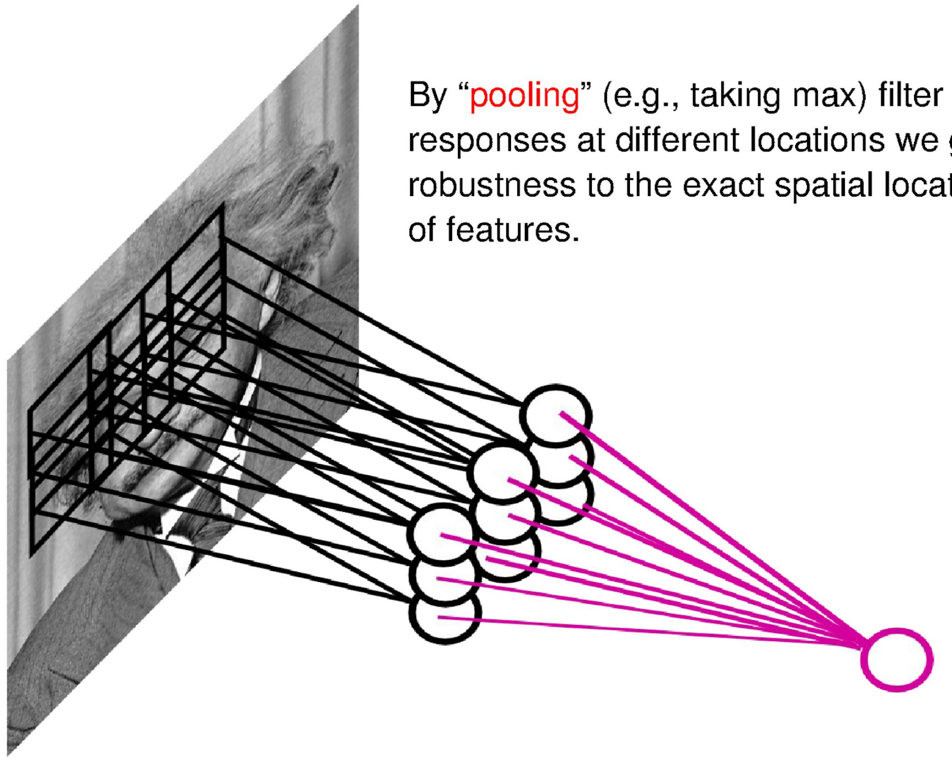


Let us assume filter is an "eye" detector.

Q.: how can we make the detection robust to the exact location of the eye?

60

Ranzato



By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

61

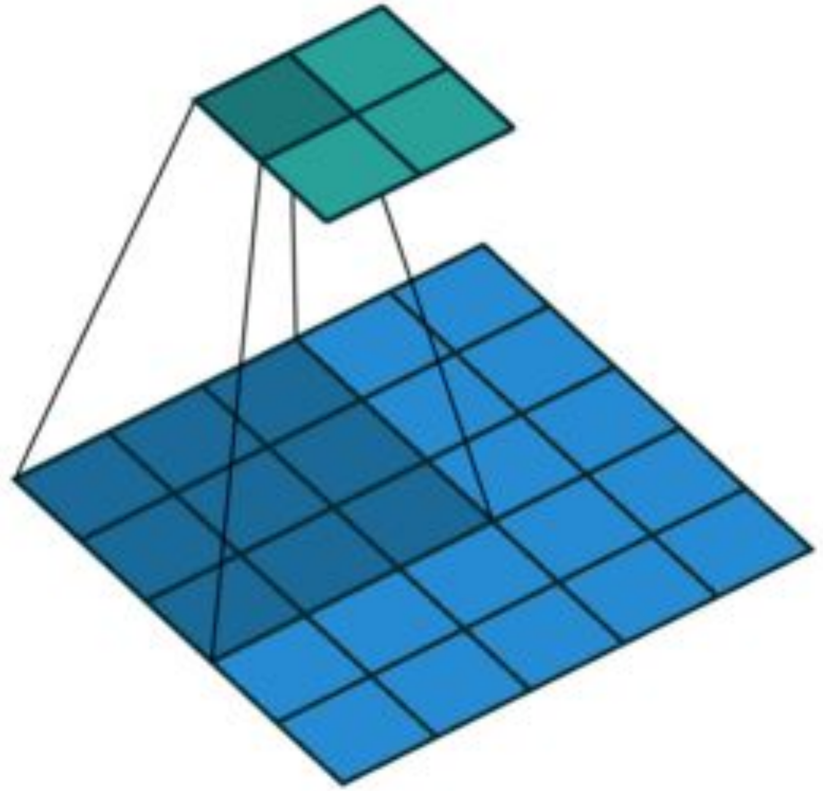
Ranzato

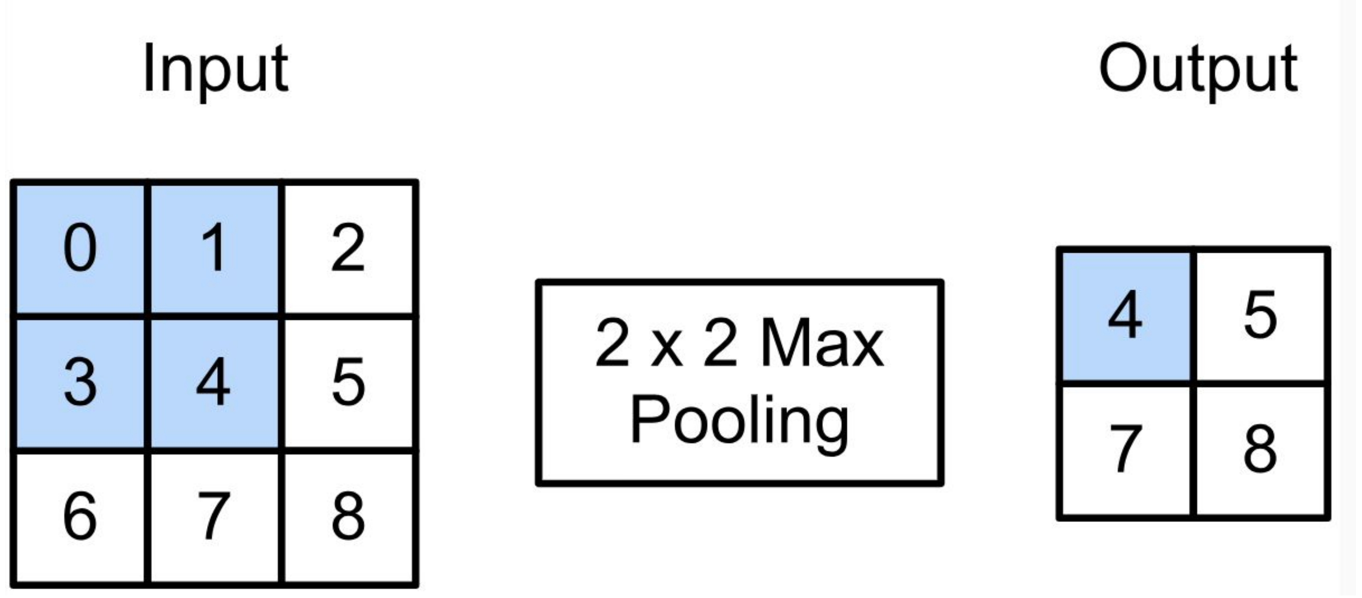
Credit: Marc'Aurelio Ranzato

Max Pooling

- Returns the maximal value in the sliding window

- Example:
  - max(0,1,3,4) = 4

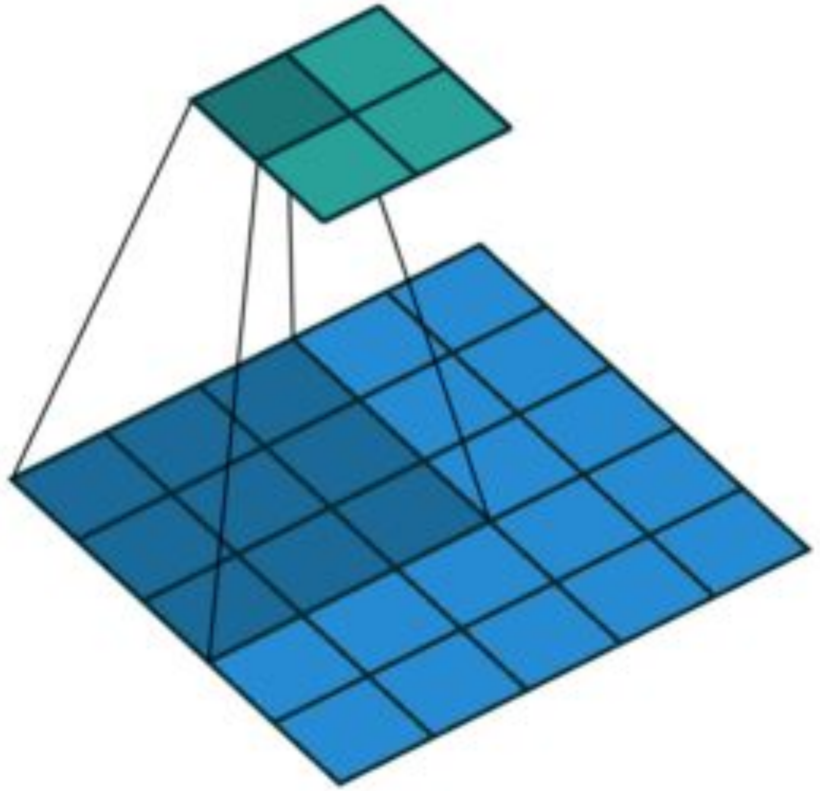


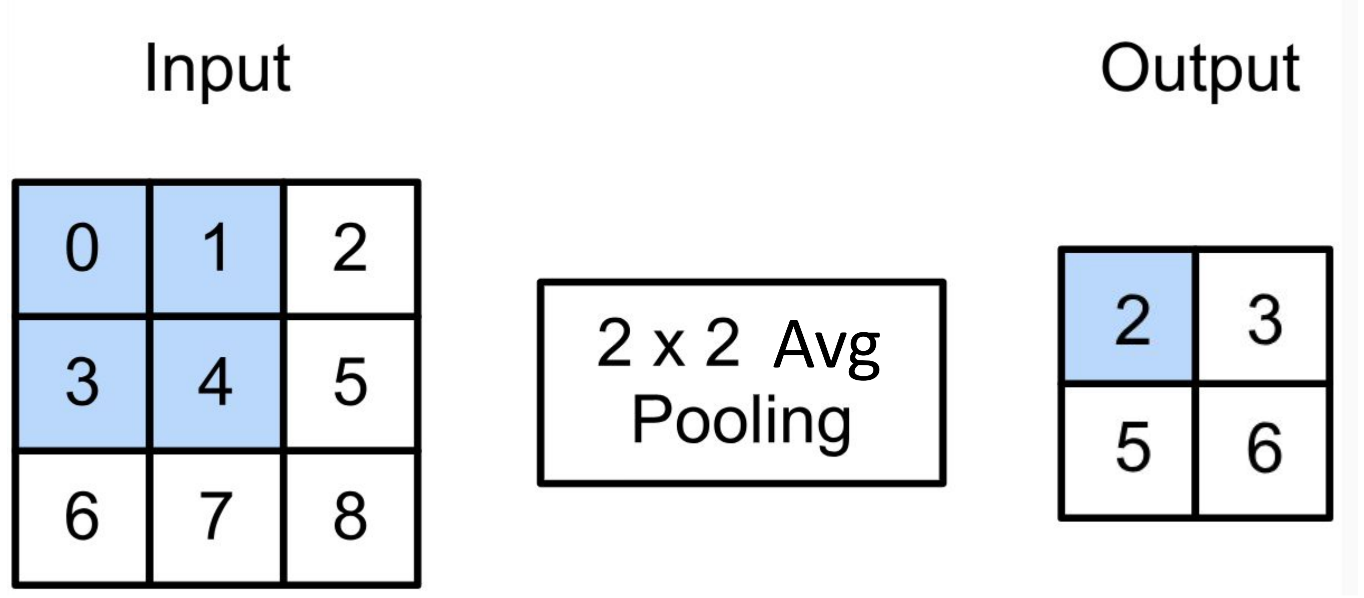


Average Pooling

- Returns the average value in the sliding window

- Example:
  - avg(0,1,3,4) = 2

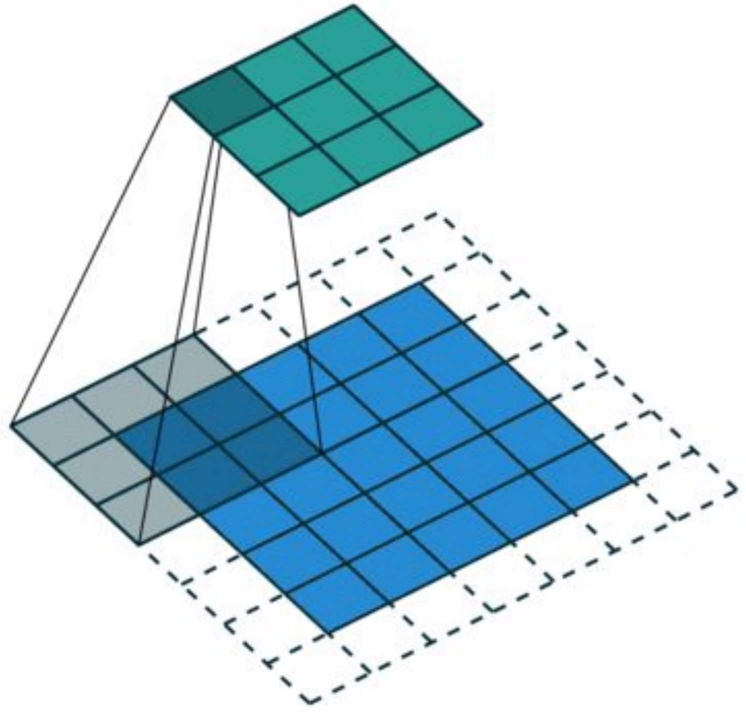




Other CNN Layers: Pooling

1. Pooling layers have similar padding and stride as convolutional layers
2. No learnable parameters
3. Apply pooling for each input channel to obtain the corresponding output channel

#output channels = #input channels



Break & Quiz

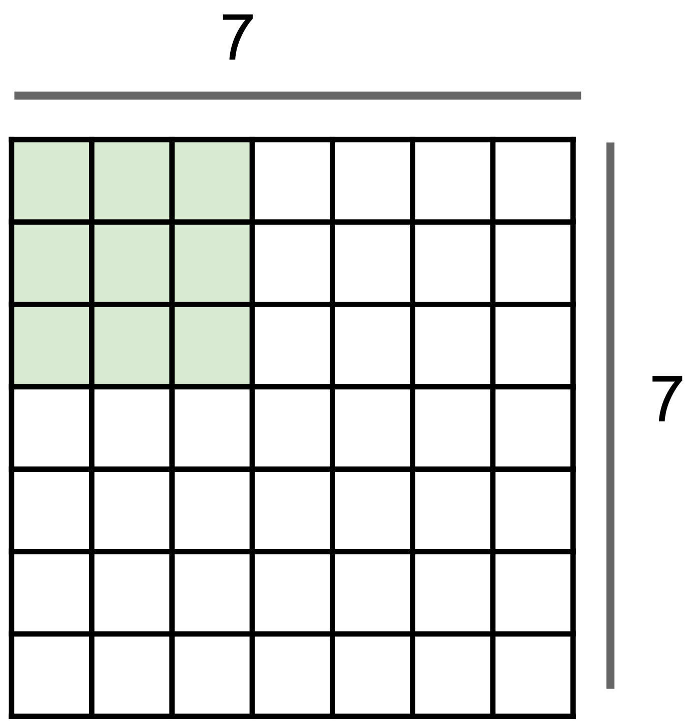
Q2-1. Suppose we want to perform convolution on a single channel image of size 7x7 (no padding) with a kernel of size 3x3, and stride = 2. What is the dimension of the output?

A. 3x3

B. 7x7

C. 5x5

D. 2x2



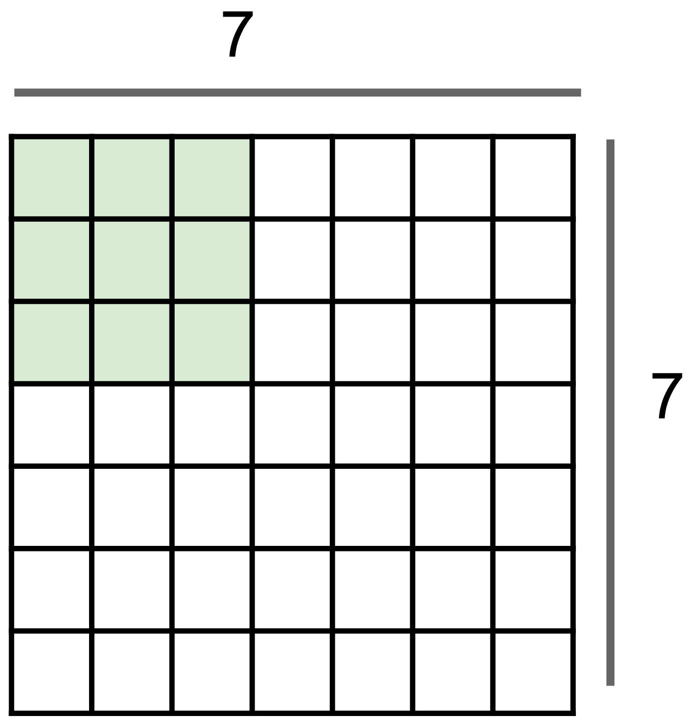
Q2-1. Suppose we want to perform convolution on a single channel image of size 7x7 (no padding) with a kernel of size 3x3, and stride = 2. What is the dimension of the output?

A. 3x3 ⬅

B. 7x7

C. 5x5

D. 2x2



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

Q2-2. Suppose we want to perform 2x2 average pooling on the following single channel feature map of size 4x4 (no padding), and stride = 2. What is the output?

12	20	30	0
20	12	2	0
0	70	5	2
8	2	90	3

A.

20	30
70	90

B.

16	8
20	25

C.

20	30
20	25

D.

12	2
70	5

Q2-2. Suppose we want to perform 2x2 average pooling on the following single channel feature map of size 4x4 (no padding), and stride = 2. What is the output?

12	20	30	0
20	12	2	0
0	70	5	2
8	2	90	3

A.

20	30
70	90

[✓ B.]

16	8
20	25

C.

20	30
20	25

D.

12	2
70	5

Outline

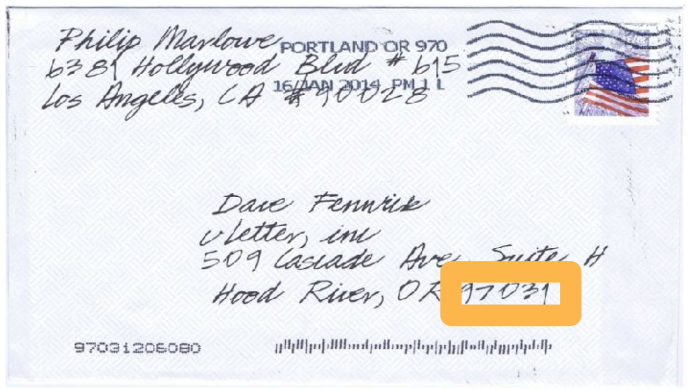
- **Convolutional Neural Networks (CNN) Basics**
  - motivation, convolution operator

- **CNN Components & Layers**
  - padding, stride, dilation, channels, pooling layers

- **CNN Tasks & Architectures**
  - MNIST, ImageNet, LeNet, AlexNet, ResNet

CNN Tasks

- Traditional tasks: handwritten digit recognition
- Dates back to the '70s and '80s
  - Low-resolution images, 10 classes

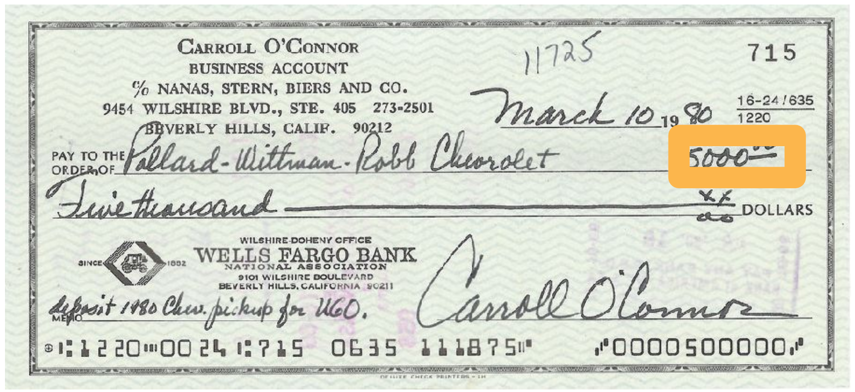


Philip Marlowe
6381 Hollywood Blvd #615
Los Angeles, CA #90028

PORTLAND OR 970
16 JAN 2014 PM 1 L

Dave Fennick
vletter, inc
509 Cascade Ave, Suite H
Hood River, OR 97031

97031206080



CARROLL O'CONNOR
BUSINESS ACCOUNT
% NANAS, STERN, BIERS AND CO.
9454 WILSHIRE BLVD., STE. 405 273-2501
BEVERLY HILLS, CALIF. 90212

11725

715

March 10 19 80

16-24/635
1220

PAY TO THE ORDER OF Pollard-Wittman-Robb Chevrolet $5000^{00}

Five thousand $\frac{xx}{00}$ DOLLARS

WILSHIRE-DOHENY OFFICE
SINCE 1852
WELLS FARGO BANK
NATIONAL ASSOCIATION
9101 WILSHIRE BOULEVARD
BEVERLY HILLS, CALIFORNIA 90211

MEMO deposit 1980 Chev. pickup for U60.

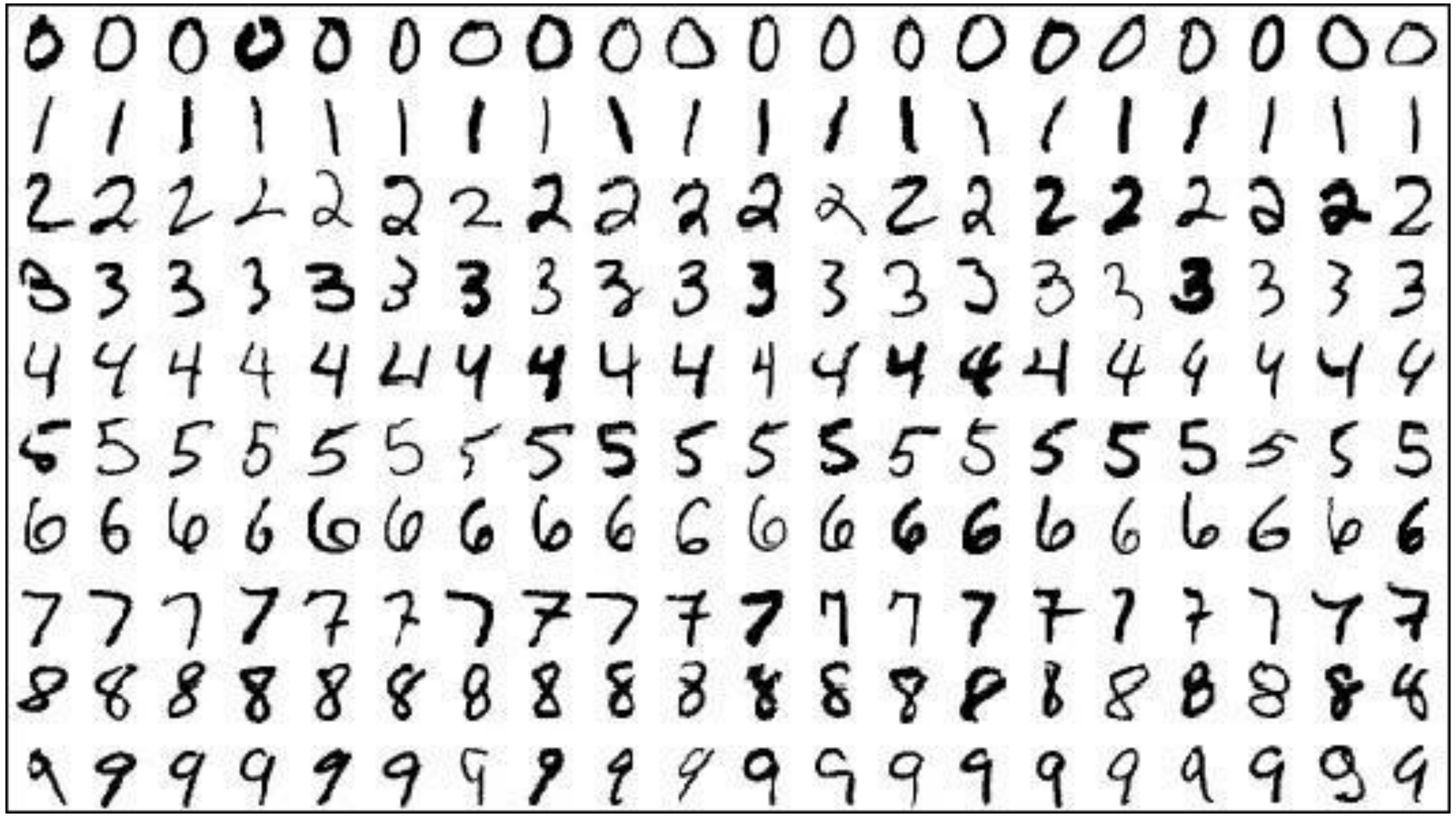
Carroll O'Connor

⑆1220⑉0024⑆715 0635 111875⑈ ⑈0000500000⑈

CNN Tasks

- Traditional tasks: handwritten digit recognition
- Classic dataset: MNIST

- Properties:
  - 10 classes
  - 28 x 28 images
  - Centered and scaled
  - 50,000 training data
  - 10,000 test data

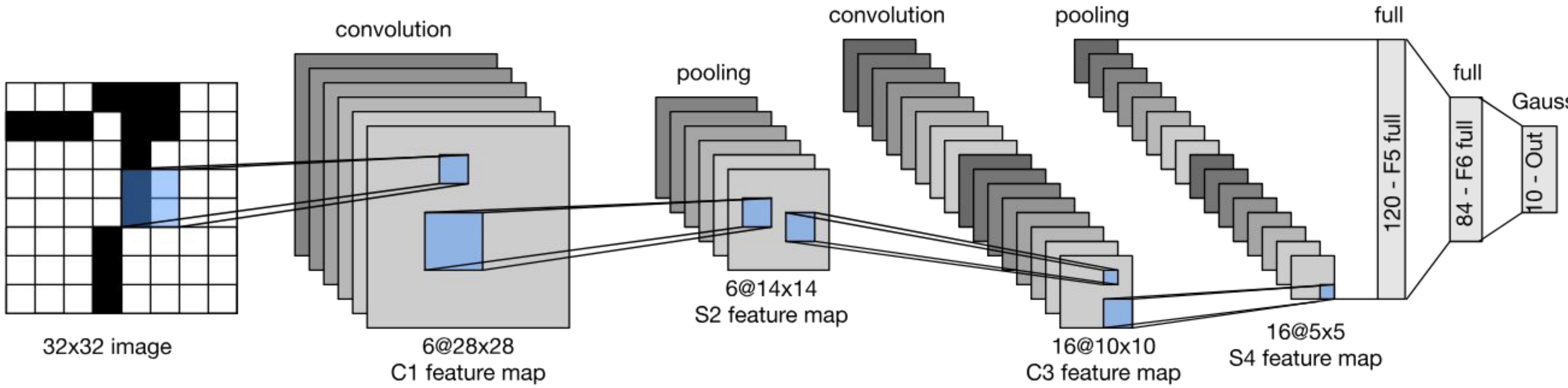


CNN Architectures

- Traditional tasks: handwritten digit recognition
- Classic dataset: MNIST
- 1989-1999: LeNet model

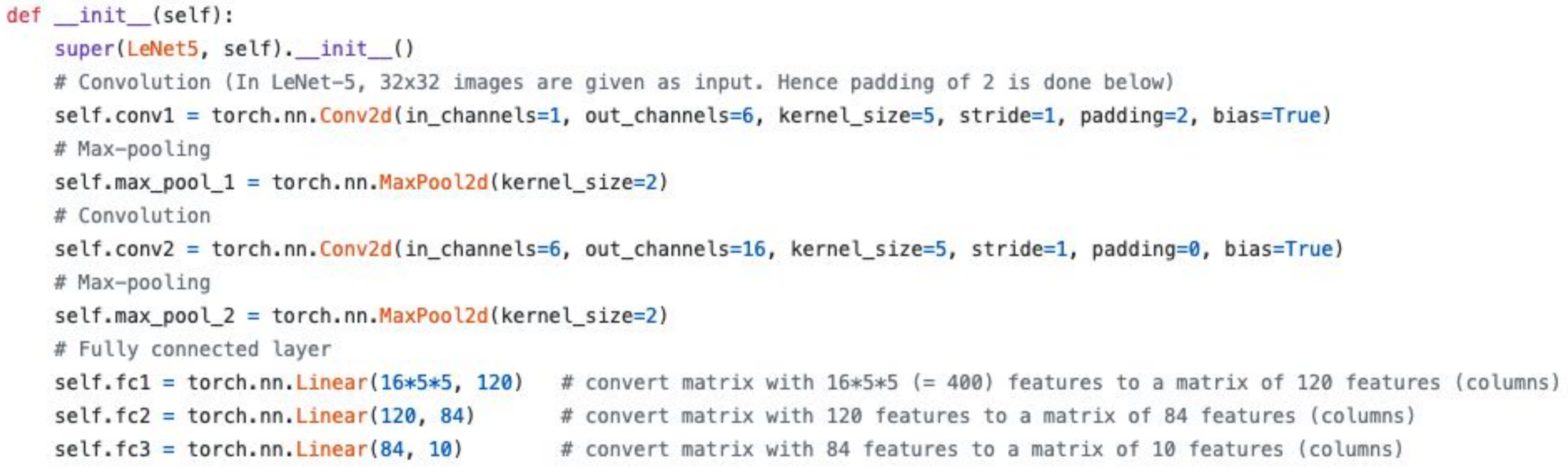
LeCun, Y et al. (1989). Backpropagation applied to handwritten zip code recognition. Neural Computation

LeCun, Y.; Bottou, L.; Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proc. IEEE



LeNet in PyTorch

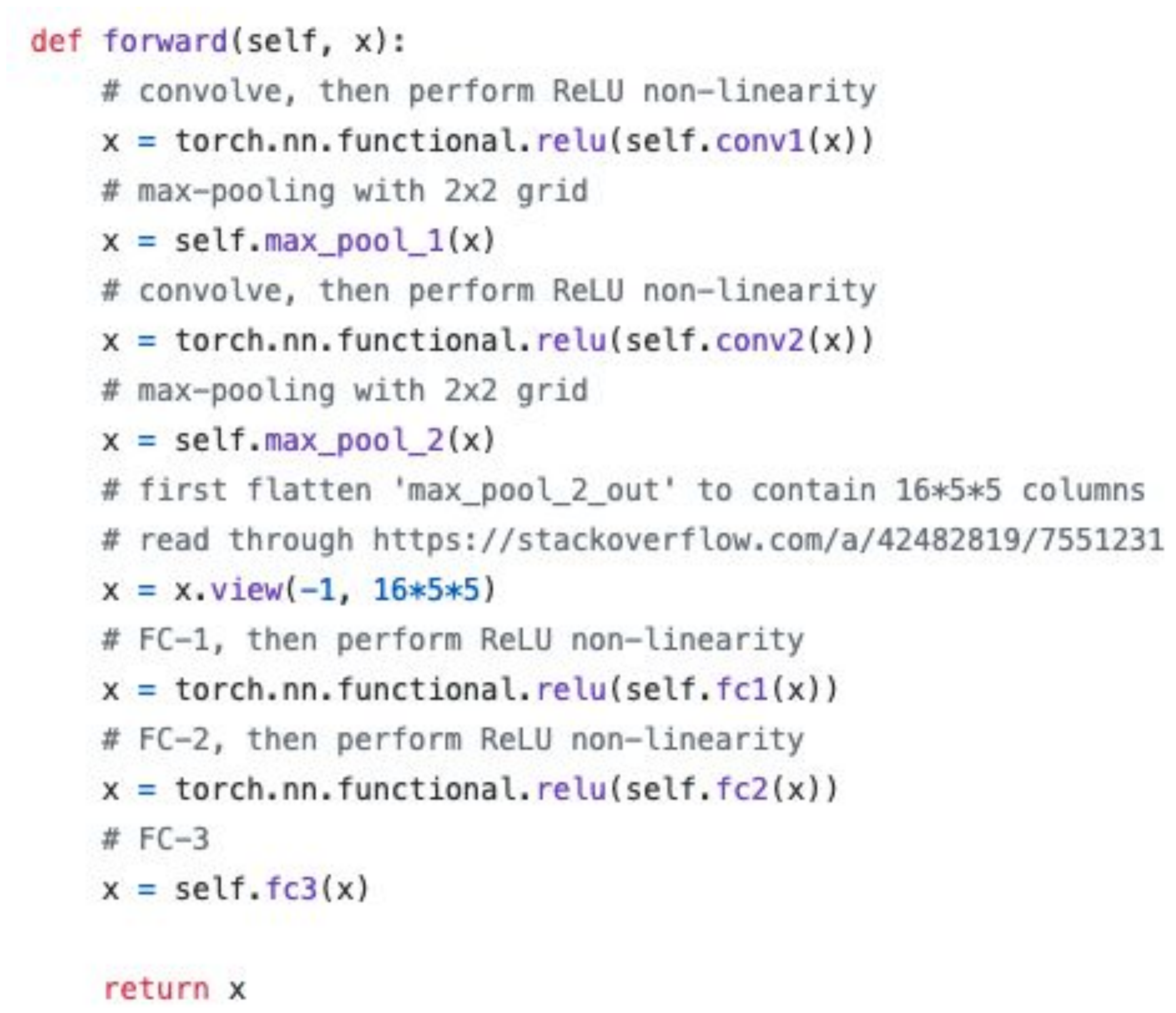
- Object setup:



```
def __init__(self):
    super(LeNet5, self).__init__()
    # Convolution (In LeNet-5, 32x32 images are given as input. Hence padding of 2 is done below)
    self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride=1, padding=2, bias=True)
    # Max-pooling
    self.max_pool_1 = torch.nn.MaxPool2d(kernel_size=2)
    # Convolution
    self.conv2 = torch.nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1, padding=0, bias=True)
    # Max-pooling
    self.max_pool_2 = torch.nn.MaxPool2d(kernel_size=2)
    # Fully connected layer
    self.fc1 = torch.nn.Linear(16*5*5, 120)   # convert matrix with 16*5*5 (= 400) features to a matrix of 120 features (columns)
    self.fc2 = torch.nn.Linear(120, 84)       # convert matrix with 120 features to a matrix of 84 features (columns)
    self.fc3 = torch.nn.Linear(84, 10)        # convert matrix with 84 features to a matrix of 10 features (columns)
```

LeNet in PyTorch

- Forward pass:

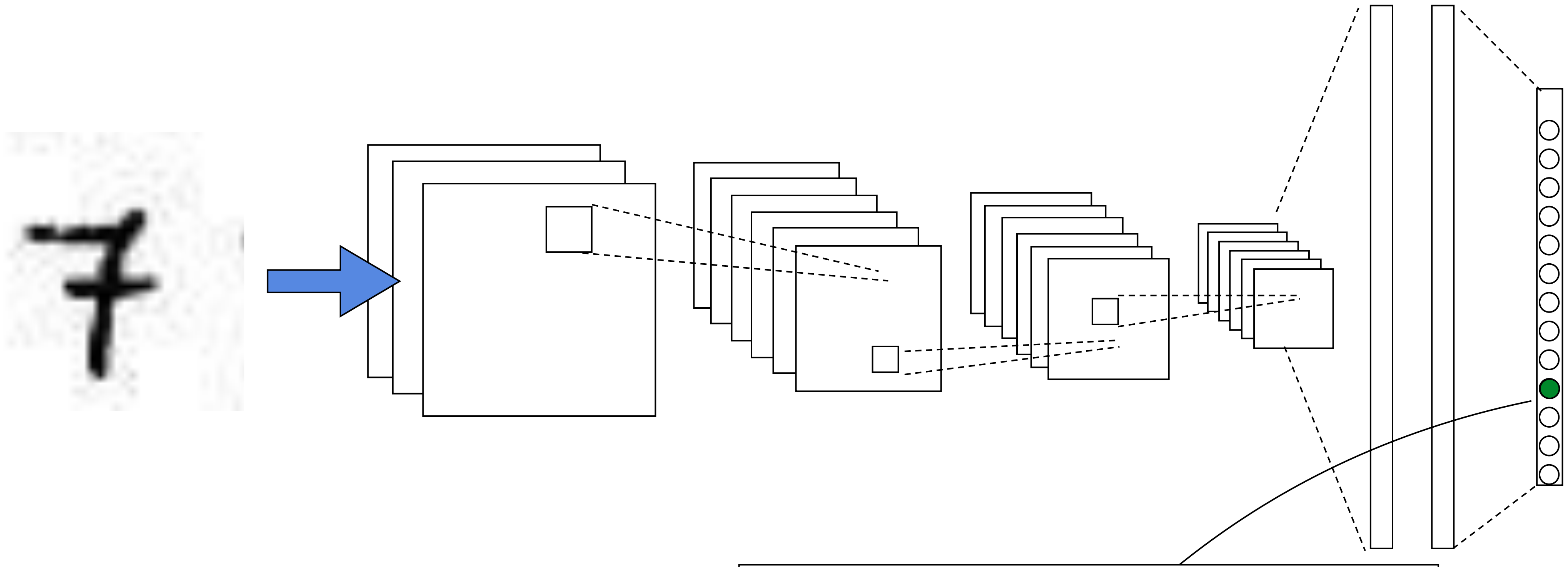


```
def forward(self, x):
    # convolve, then perform ReLU non-linearity
    x = torch.nn.functional.relu(self.conv1(x))
    # max-pooling with 2x2 grid
    x = self.max_pool_1(x)
    # convolve, then perform ReLU non-linearity
    x = torch.nn.functional.relu(self.conv2(x))
    # max-pooling with 2x2 grid
    x = self.max_pool_2(x)
    # first flatten 'max_pool_2_out' to contain 16*5*5 columns
    # read through https://stackoverflow.com/a/42482819/7551231
    x = x.view(-1, 16*5*5)
    # FC-1, then perform ReLU non-linearity
    x = torch.nn.functional.relu(self.fc1(x))
    # FC-2, then perform ReLU non-linearity
    x = torch.nn.functional.relu(self.fc2(x))
    # FC-3
    x = self.fc3(x)

    return x
```

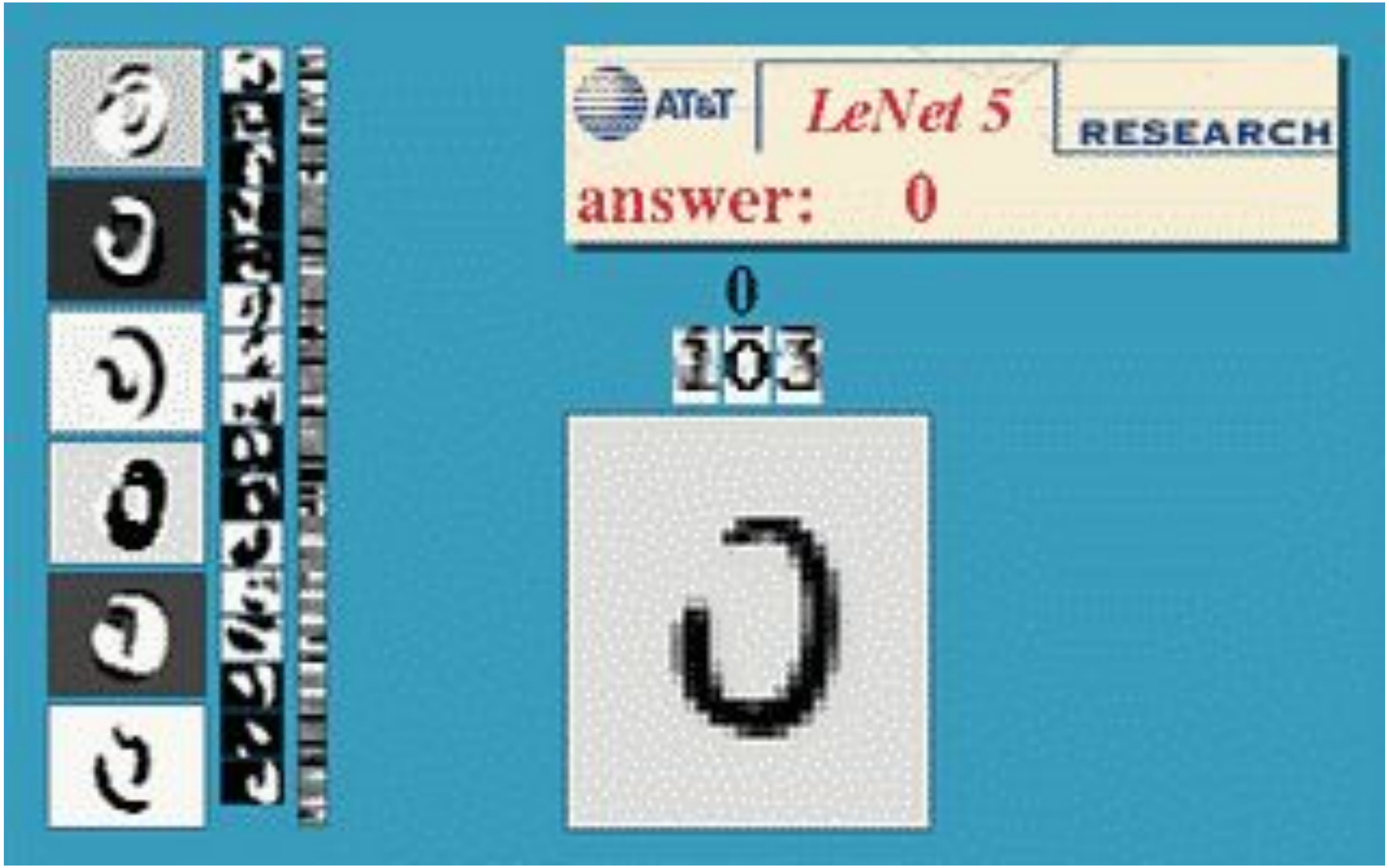
Training a CNN

- Q: So we have a bunch of layers. How do we train?
- A: Same as before. Apply softmax at the end, use backprop.



$$p_i(\boldsymbol{x}) = \frac{\exp\left(f_i(\boldsymbol{x})\right)}{\sum_{j=1}^{N} \exp\left(f_j(\boldsymbol{x})\right)},$$

Result:

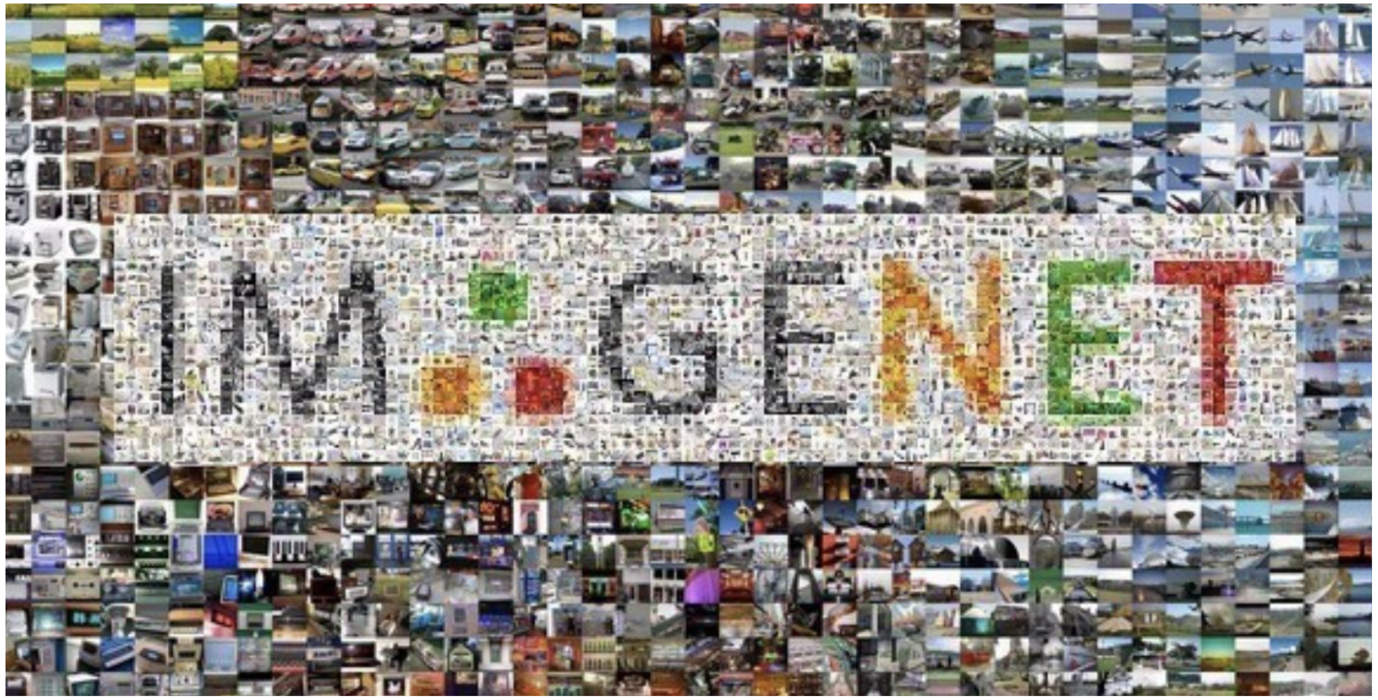


More CNN Architectures: ImageNet Task

- Next big task/dataset: image recognition on ImageNet
- Large Scale Visual Recognition Challenge (ILSVRC) 2012-2017

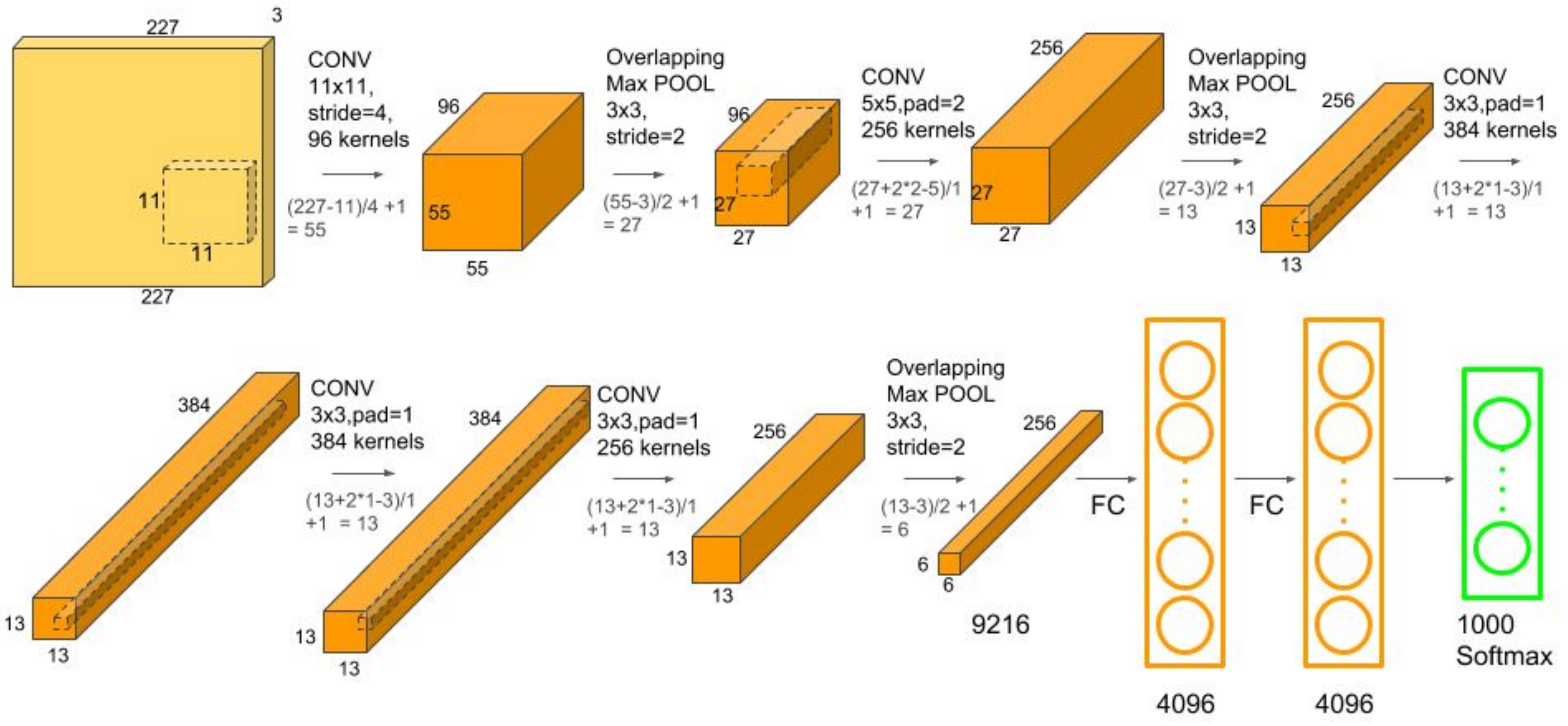
- Properties:
  - Thousands of classes
  - Full-resolution
  - 14,000,000 images

- Started 2009 (Deng et al)



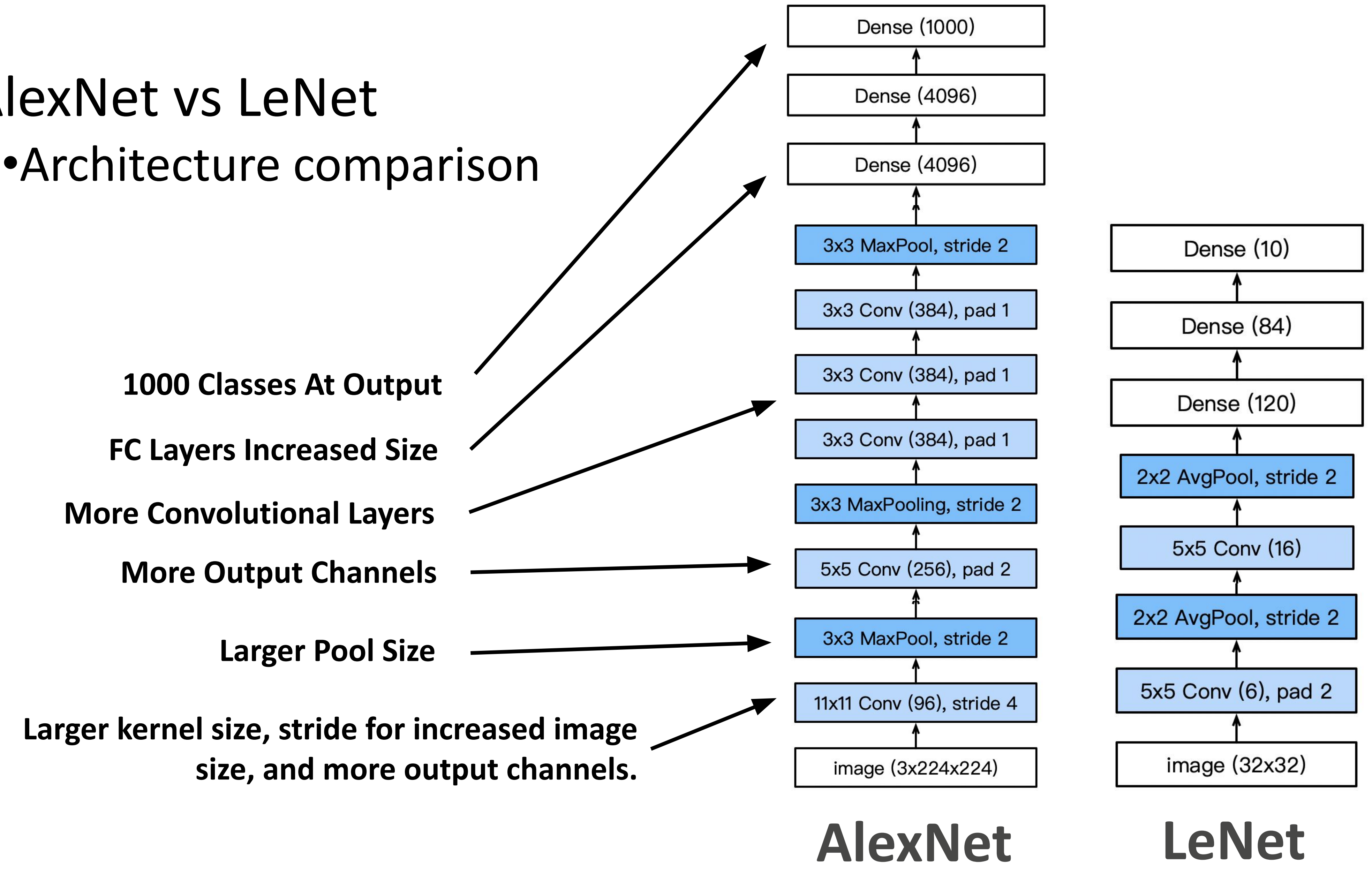
CNN Architectures: AlexNet

- AlexNet winning ImageNet 2012 was a paradigm shift in computer vision
- Major advances: deeper and bigger version of LeNet



More CNN Architectures

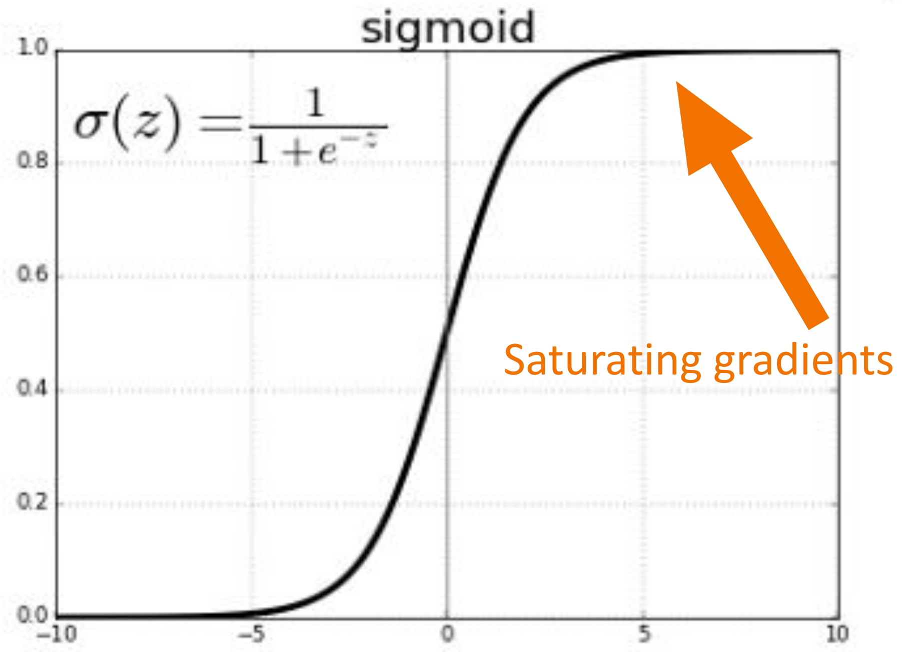
- AlexNet vs LeNet
  - Architecture comparison



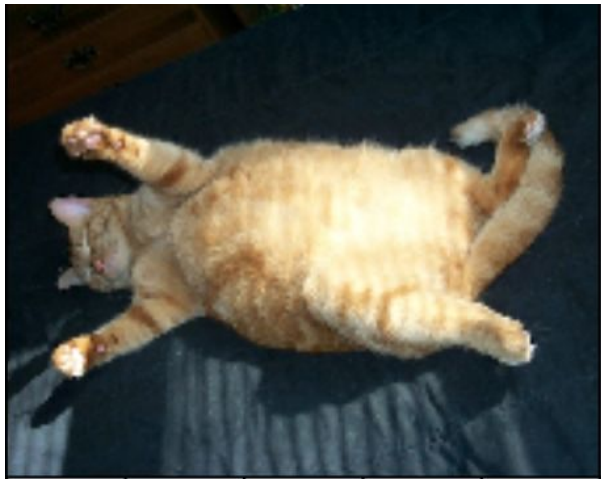
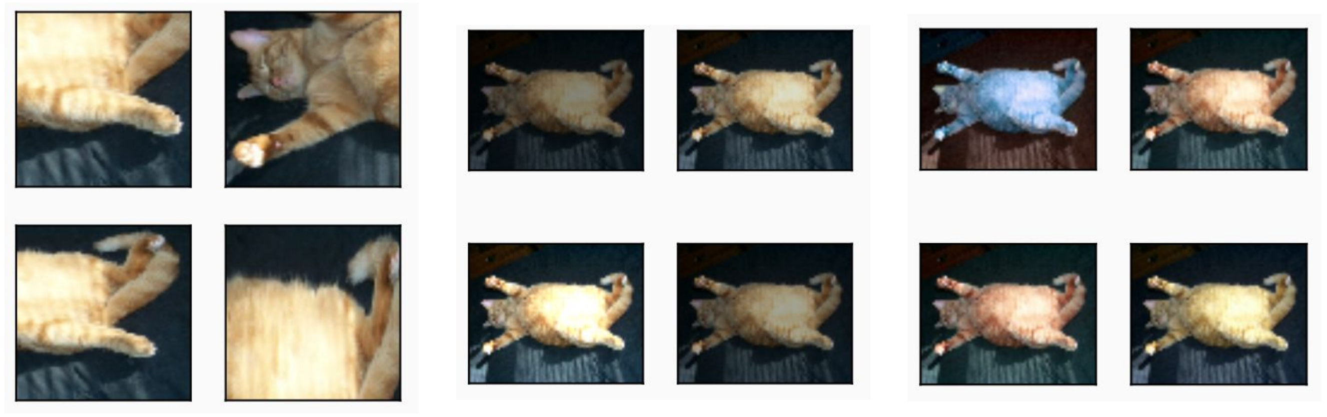
More Differences

Activations: from sigmoid to ReLU

- Deal with vanishing gradient issue

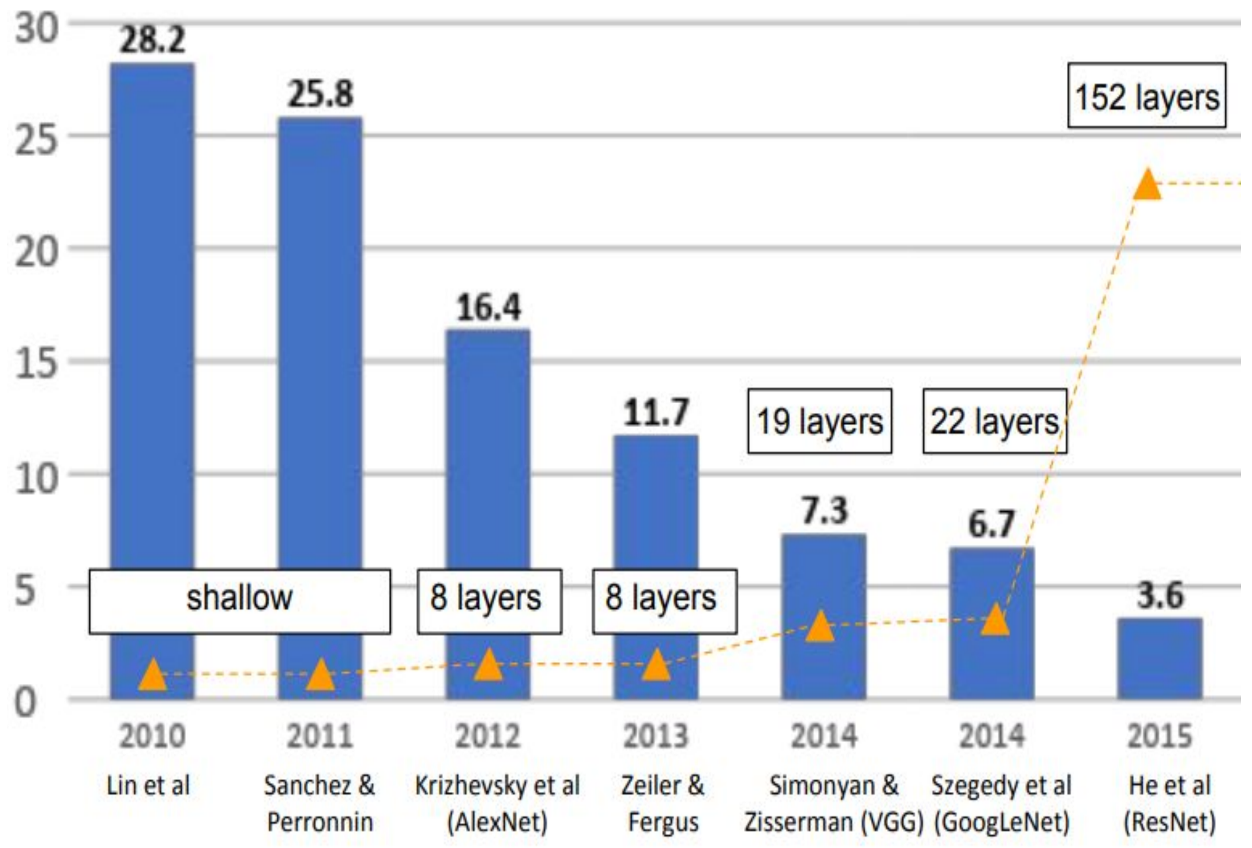


Data Augmentation

Going Further

ImageNet error rate by year; note layer count on right.

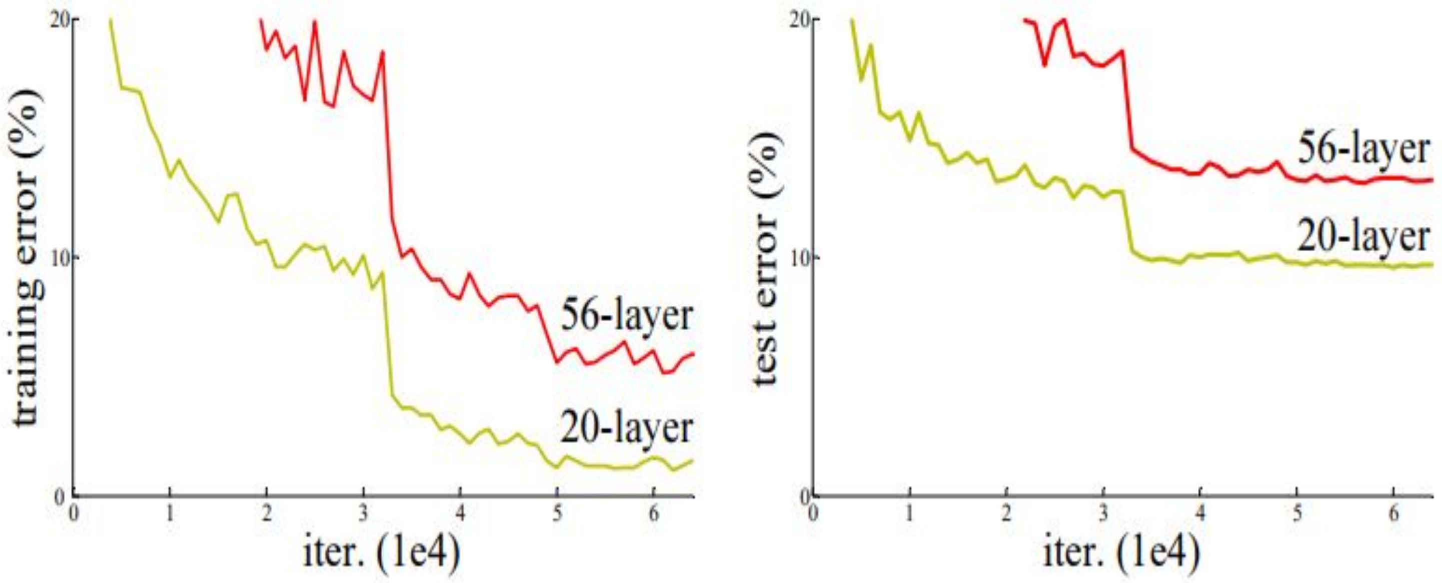


Credit: Stanford CS 231n

How to make neural networks deep

Adding too many layers leads to optimization issues:

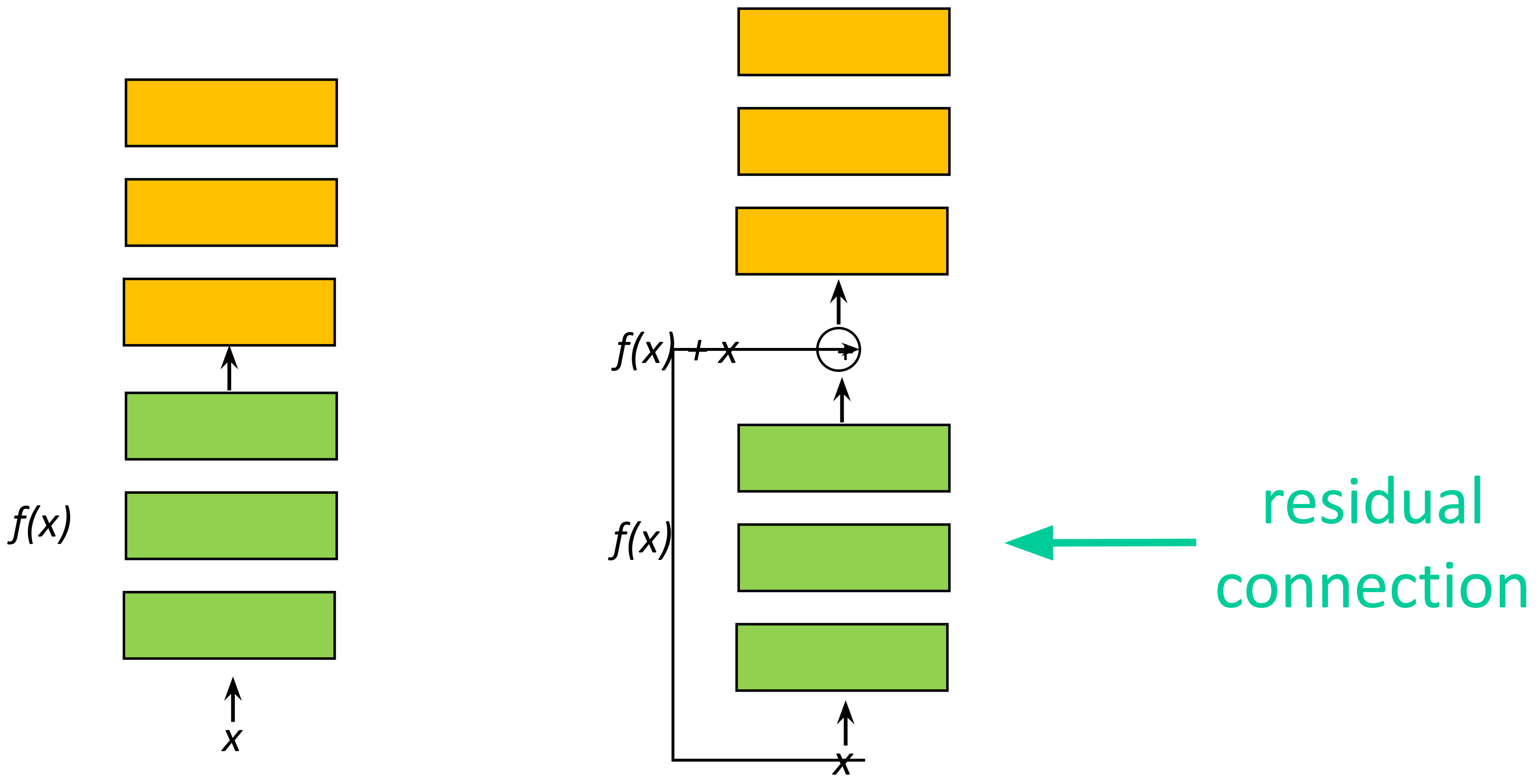
- Vanishing gradients
- Unstable training



He et al: “Deep Residual Learning for Image Recognition”

Residual Connections

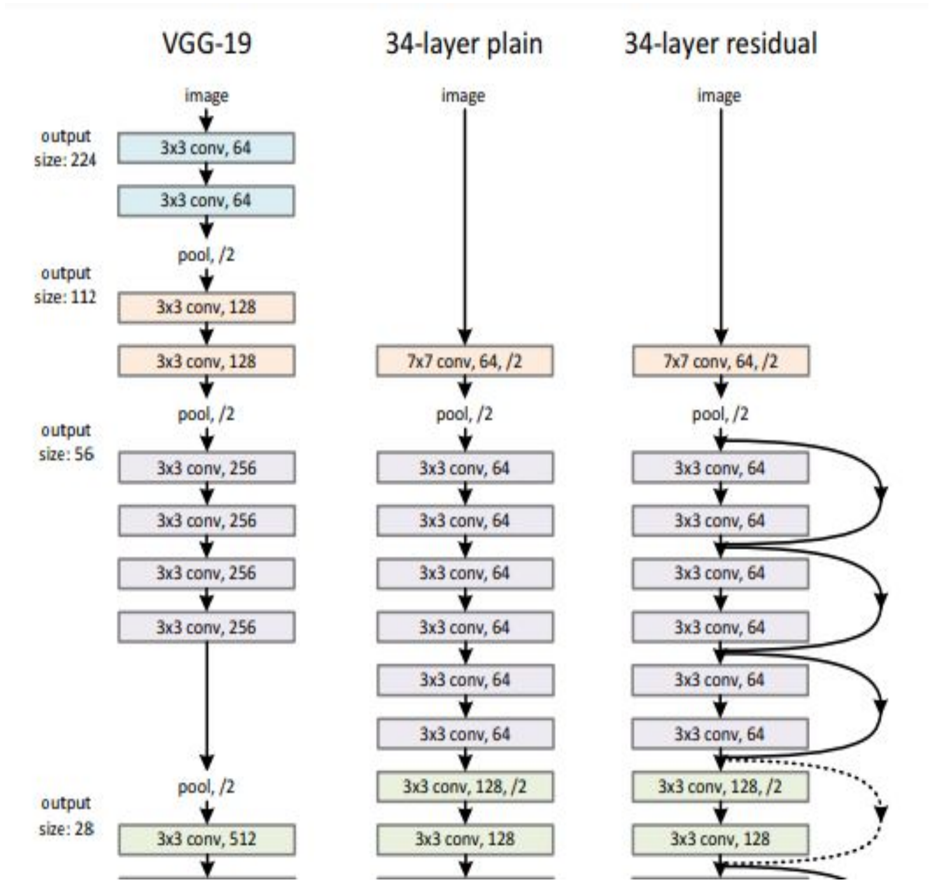
Idea: instead of transforming the input, learn a correction of the identity



ResNet Architecture

Residual or skip connections help make learning easier

- Vastly better performance
  - No additional parameters!
  - Records on many benchmarks
- Have been used in many other models, including Transformers

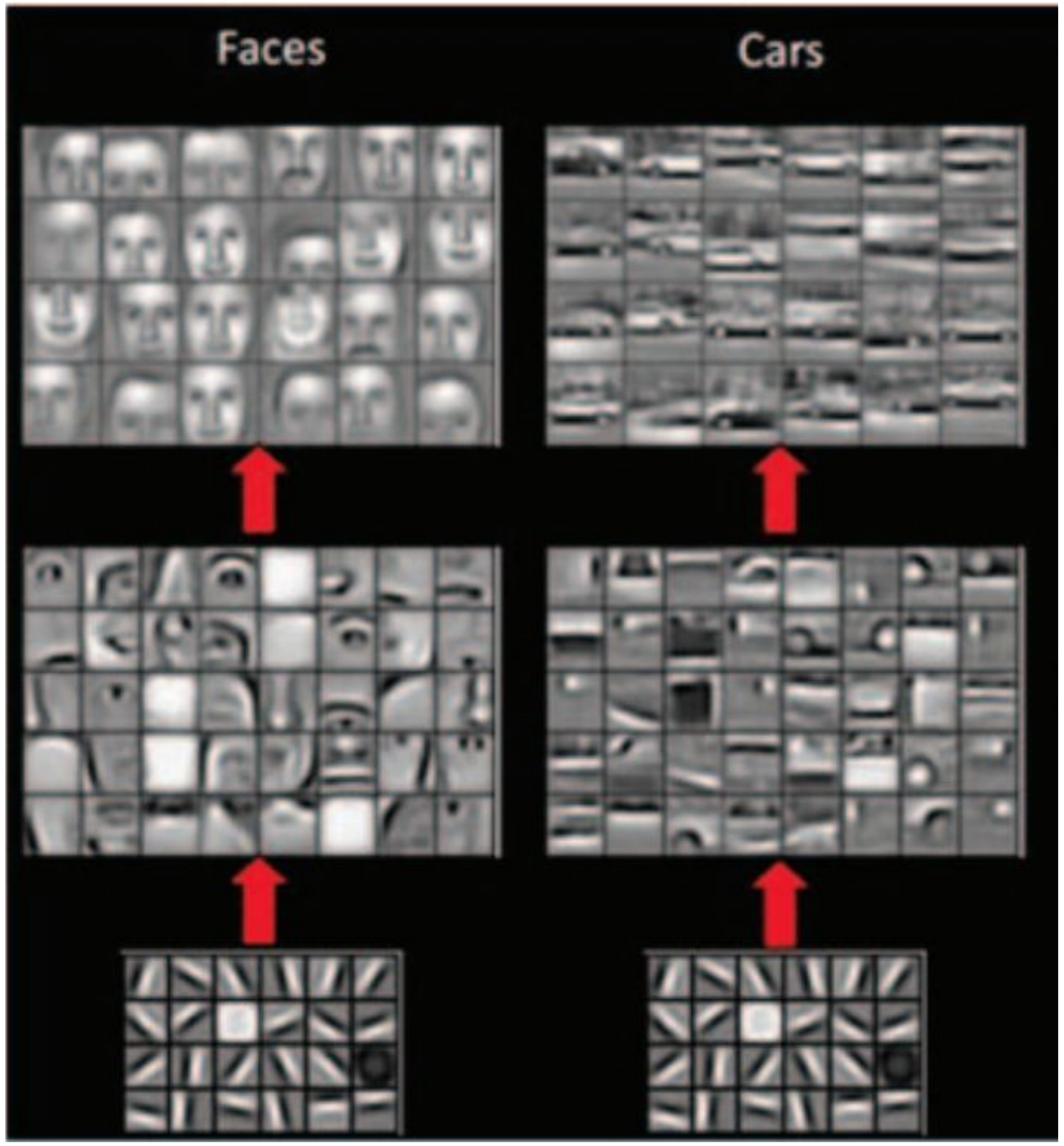


He et al: “Deep Residual Learning for Image Recognition”

Why “**deep**” learning?

Hierarchical feature learning:

- Last layer detect complete objects
- Middle layers detect parts of objects
- Early layers detect simple patterns



Seigel et al., 2016.

Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Misha Khodak, Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov, Sharon Li, Fred Sala, Kirthi Kandasamy, Josiah Hanna