ECE 661 COMP ENG ML & DEEP NEURAL NETS

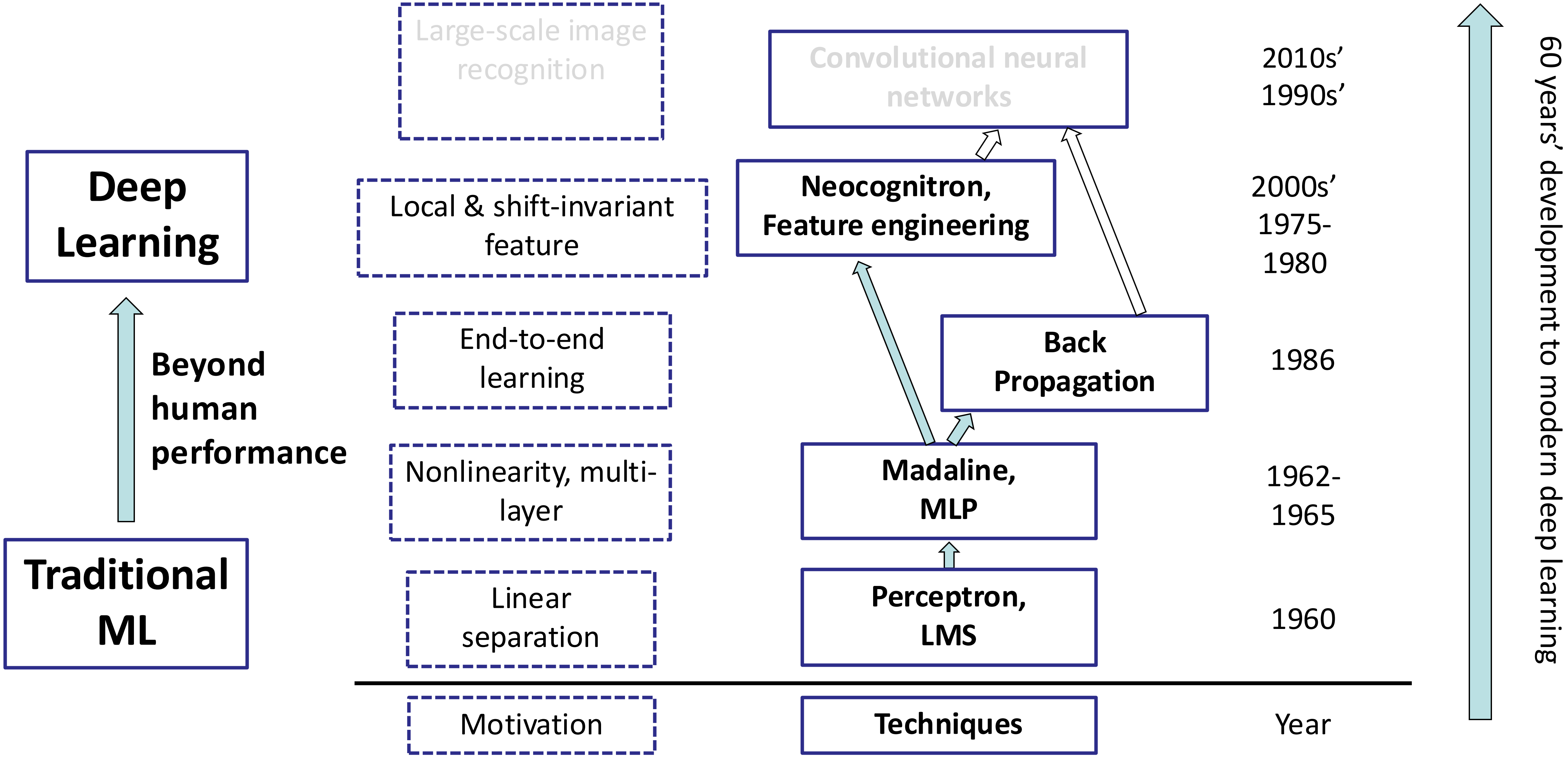
4. 60 YEARS OF NEURAL NETWORK (3/3): CONVOLUTIONAL NEURAL NETWORKS

HAI “HELEN” LI, SPRING 2025

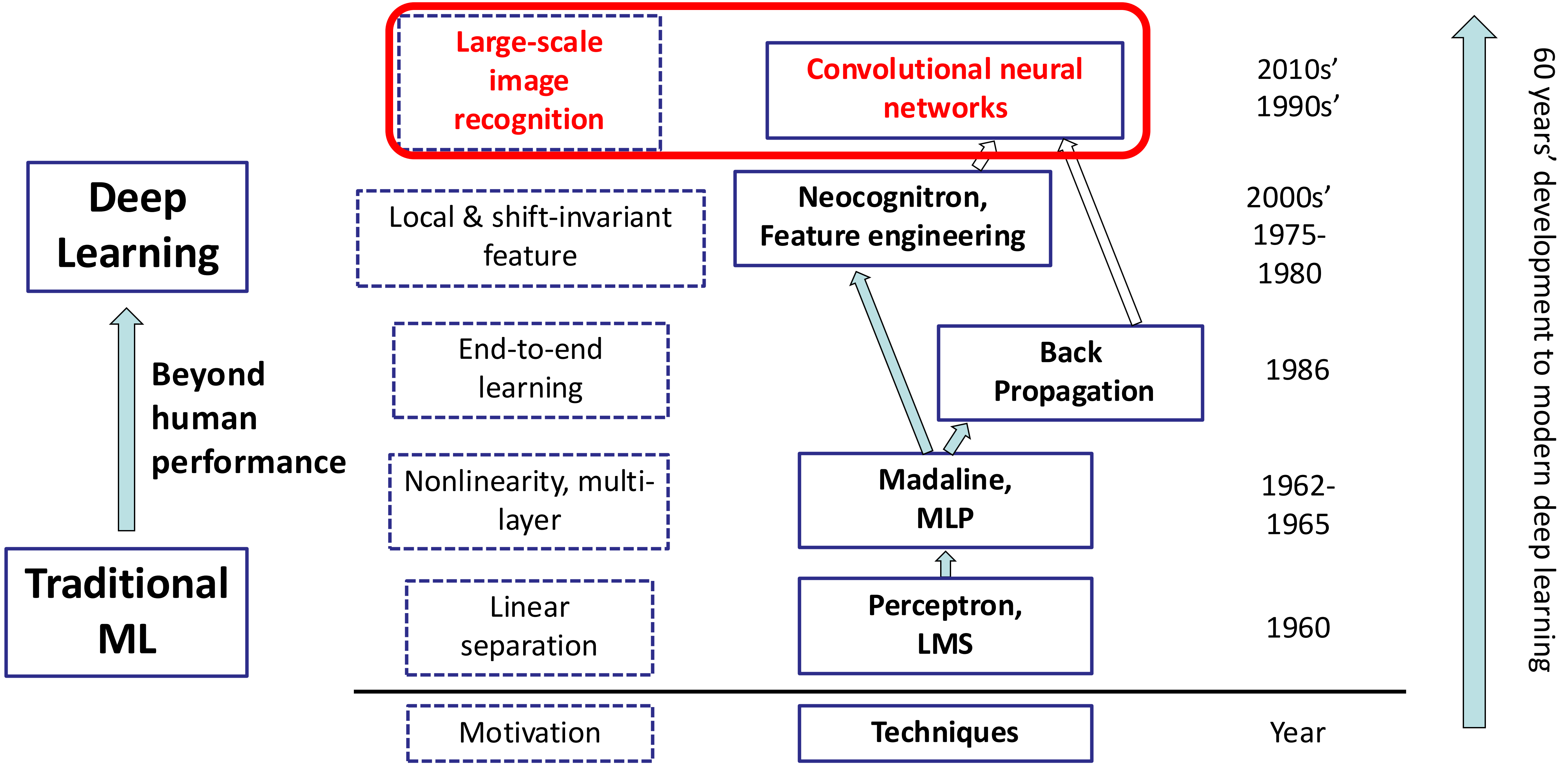
Getting Info

- **Slack workspace:** questions/answers
  - Use you Duke email to sign up at the following link
    - https://join.slack.com/t/ece661-25sp/shared_invite/zt-2xkdb7i8g-VJk6goye~M1Df1FZ_jXGcQ
  - Post all your questions here
  - Questions must be "public" unless good reason otherwise
  - No code in public posts!

Previously


Deep Learning
Beyond human performance
Traditional ML
Large-scale image recognition
Local & shift-invariant feature
End-to-end learning
Nonlinearity, multi-layer
Linear separation
Motivation
Convolutional neural networks
Neocognitron, Feature engineering
Back Propagation
Madaline, MLP
Perceptron, LMS
Techniques
2010s'
1990s'
2000s'
1975-1980
1986
1962-1965
1960
Year
60 years' development to modern deep learning

This lecture



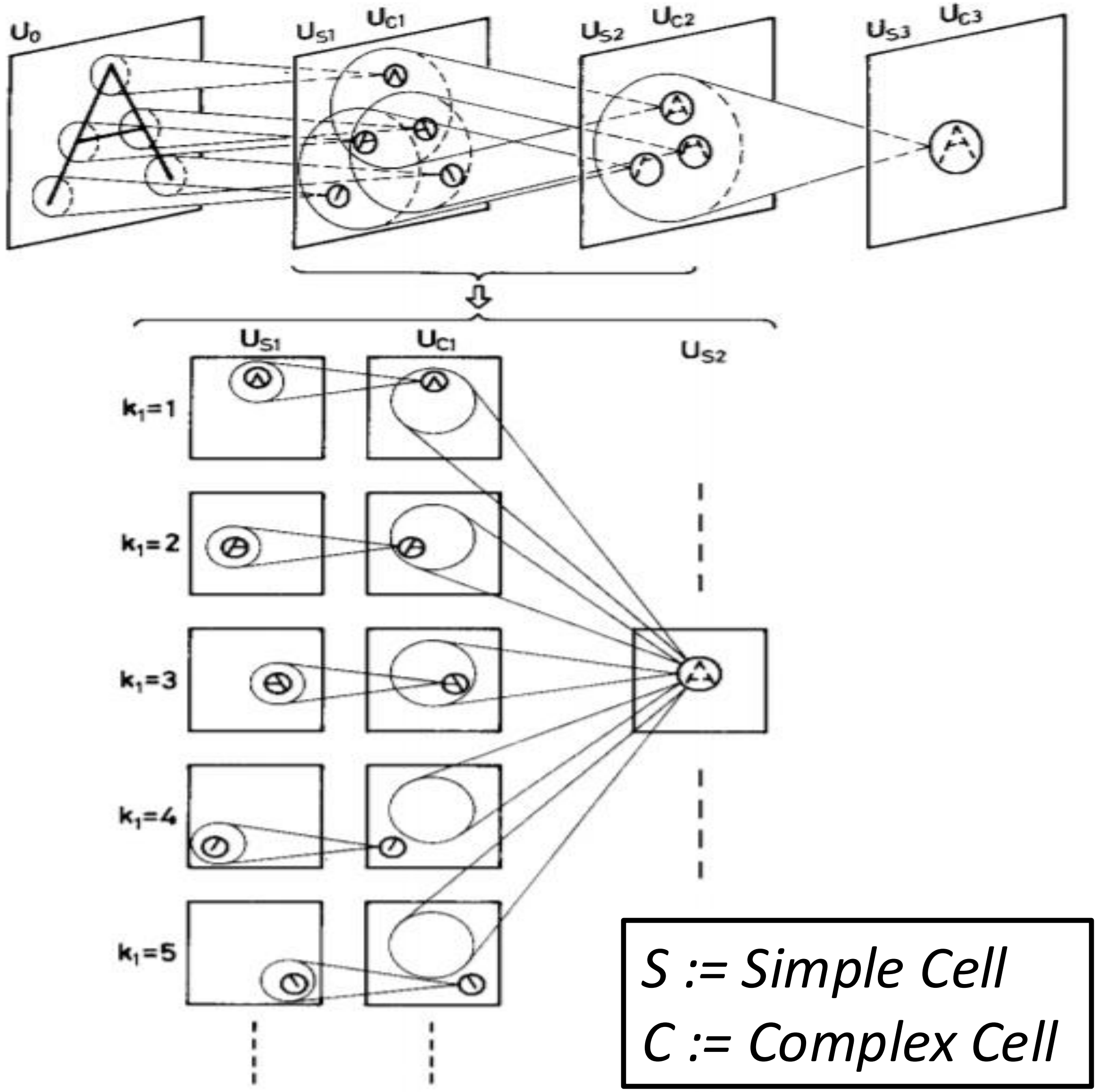
Outline

- Layers of Convolutional Neural networks
- Case study: Understanding LeNet-5
- Basic CNN design concepts
- How to implement CNN on PyTorch

Outline

- Layers of Convolutional Neural networks
- Case study: Understanding LeNet-5
- Basic CNN design concepts
- How to implement CNN on PyTorch

Recall: Neocognitron



S := Simple Cell
C := Complex Cell

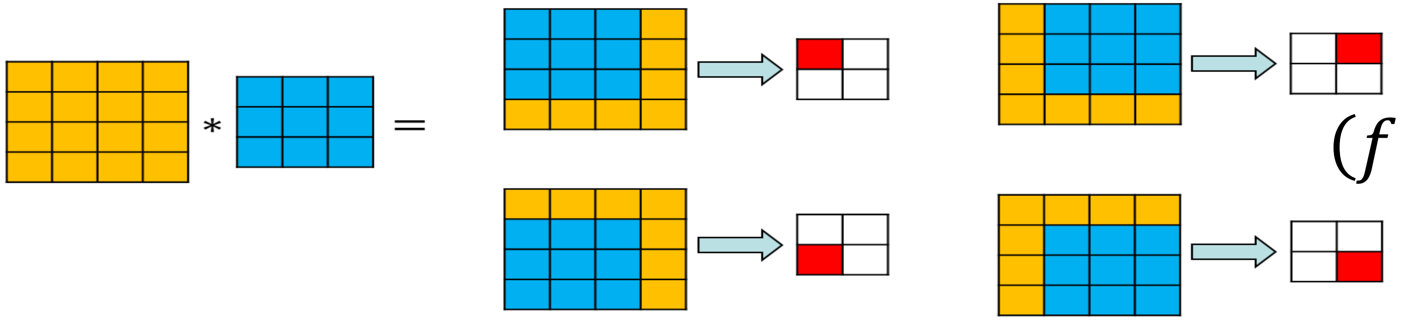
- S -> C: Down sampling/pooling
  - Down sample output features to fit more filter outputs into the receptive field (draw as circles on feature map) of a later filter
- C -> S: Convolution
  - Gather the extracted information from previous layer's filters to form a more complex feature
- Gradually captures more abstract features towards later layers

Components of a modern CNN model

- Convolutional layers
  - Perform convolution operation on input feature maps with multiple filters
- Pooling layers
  - Down sampling the feature map, gathering more information into the receptive field of later layer
- Fully Connected (FC) layer
  - Acts as the classifier
  - With high-level features extracted in later CONV layers, use those features to perform learning task
- We will formally introduce these layers in this lecture

Recall: Convolution operation

Convolution with Kernel size K; shift and apply filter



$$(f * g)[x, y] = \sum_{k,k} f[x + i, y + j]g[i, j]$$

Strides S: the amount by which filter shifts each step

Padding P: Zeros around input to maintain output size

Shape rule: input $\mathrm{H}_1 \times \mathrm{W}_1$, output $H_2 \times W_2$

$$W_2 = \left\lfloor \frac{W_1 - K + 2P}{S} \right\rfloor + 1$$

$$H_2 = \left\lfloor \frac{H_1 - K + 2P}{S} \right\rfloor + 1$$

These parameters work the same in a convolutional layer

2-D → 3-D: channel

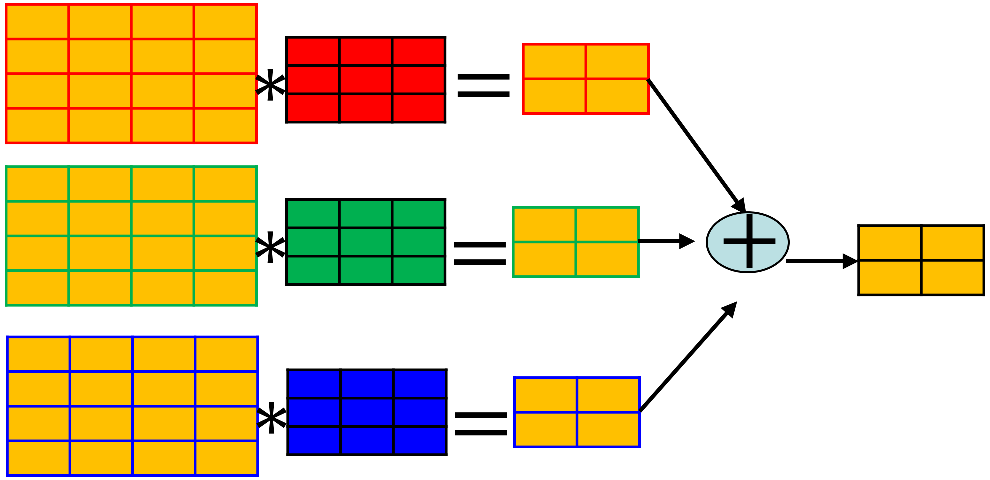
A color image is stored as a 3-D tensor with 3 **channels** (R,G,B)

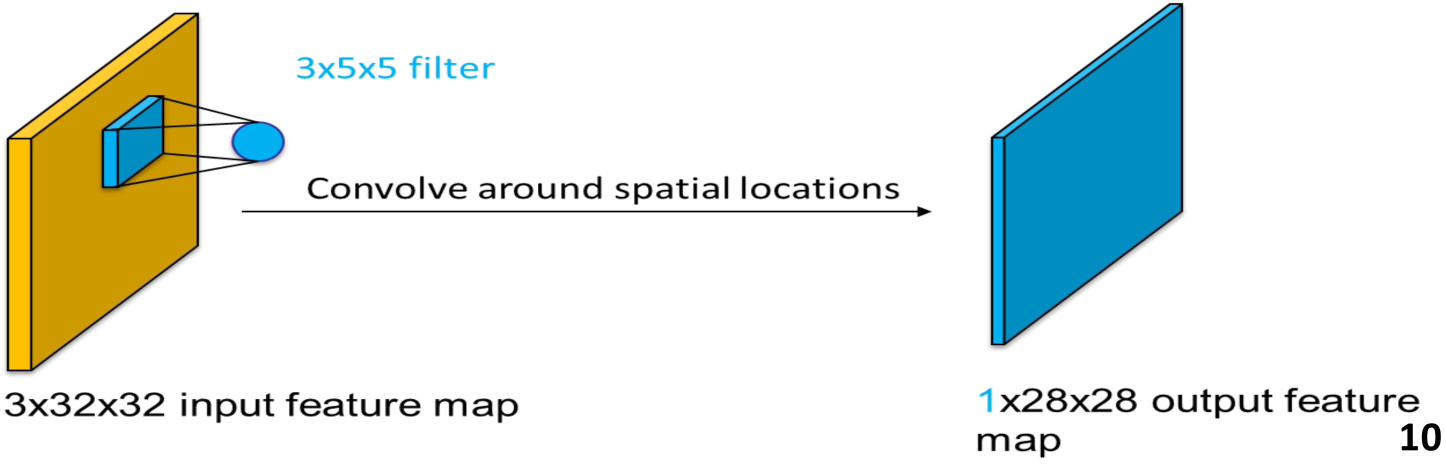
Locality property holds across channels

- Pixels at the same position in different channels should correspond to the same object

Convolution on 3-D input (***I* channels**)

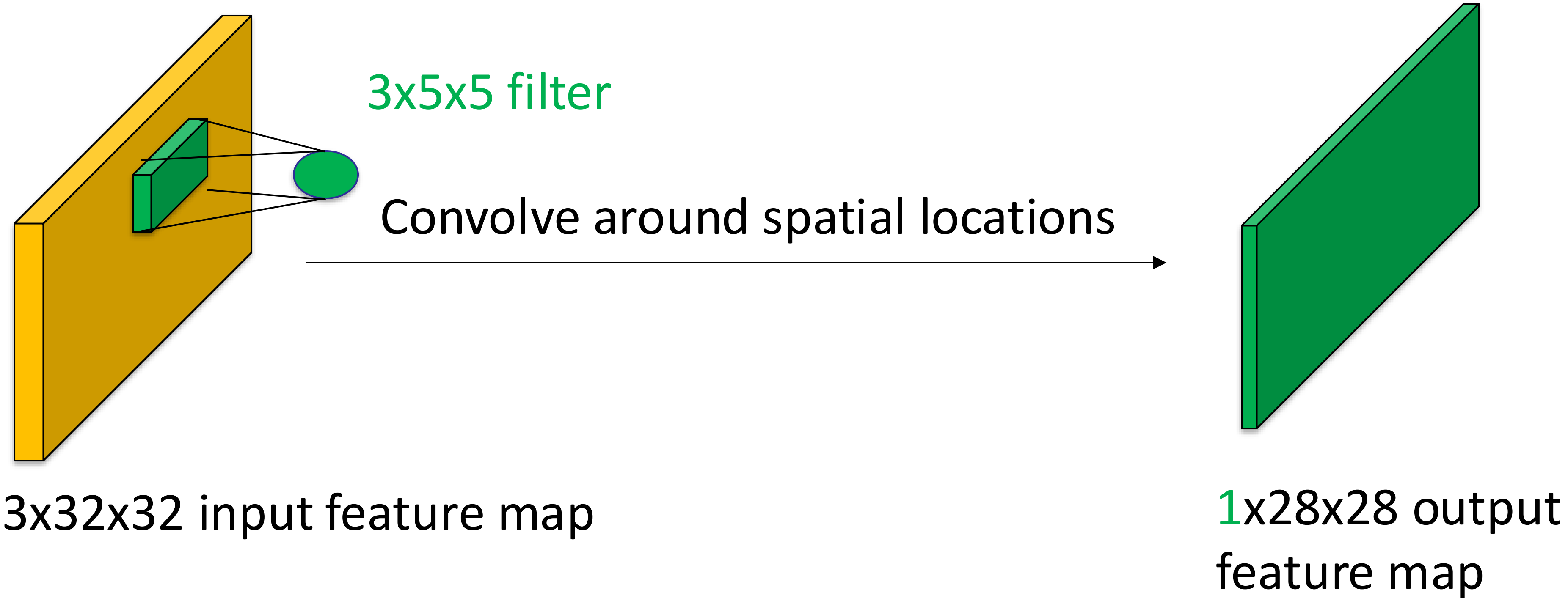
- Put a $K \times K$ filter on each channel, filter weights independent
- Lead to a **3-D filter** with a $I \times K \times K$ kernel
- Sum the convolution result across all input channels to get an output feature map, or an **output channel**





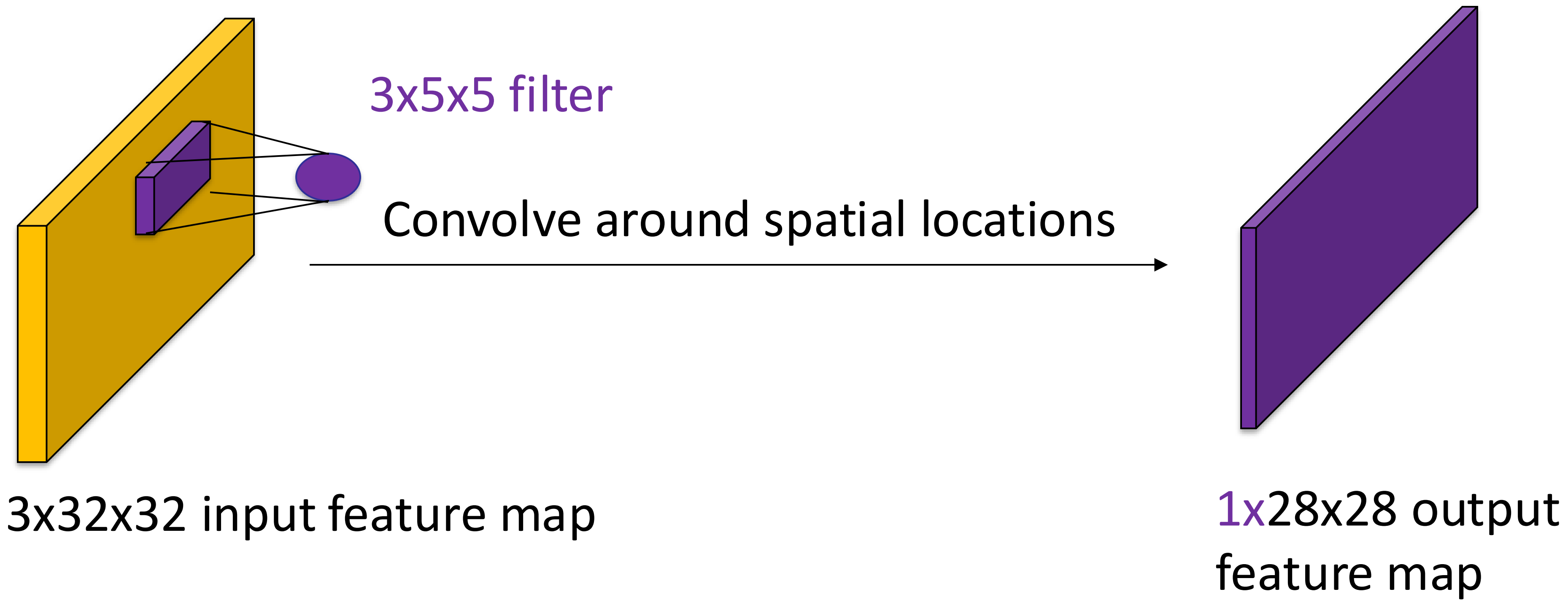
Applying another filter

- Different filters can learn to extract different features
- Let’s add another 3-D filter on the input feature map
- Slide the filter over the image spatially and obtain another 3-D output feature map.



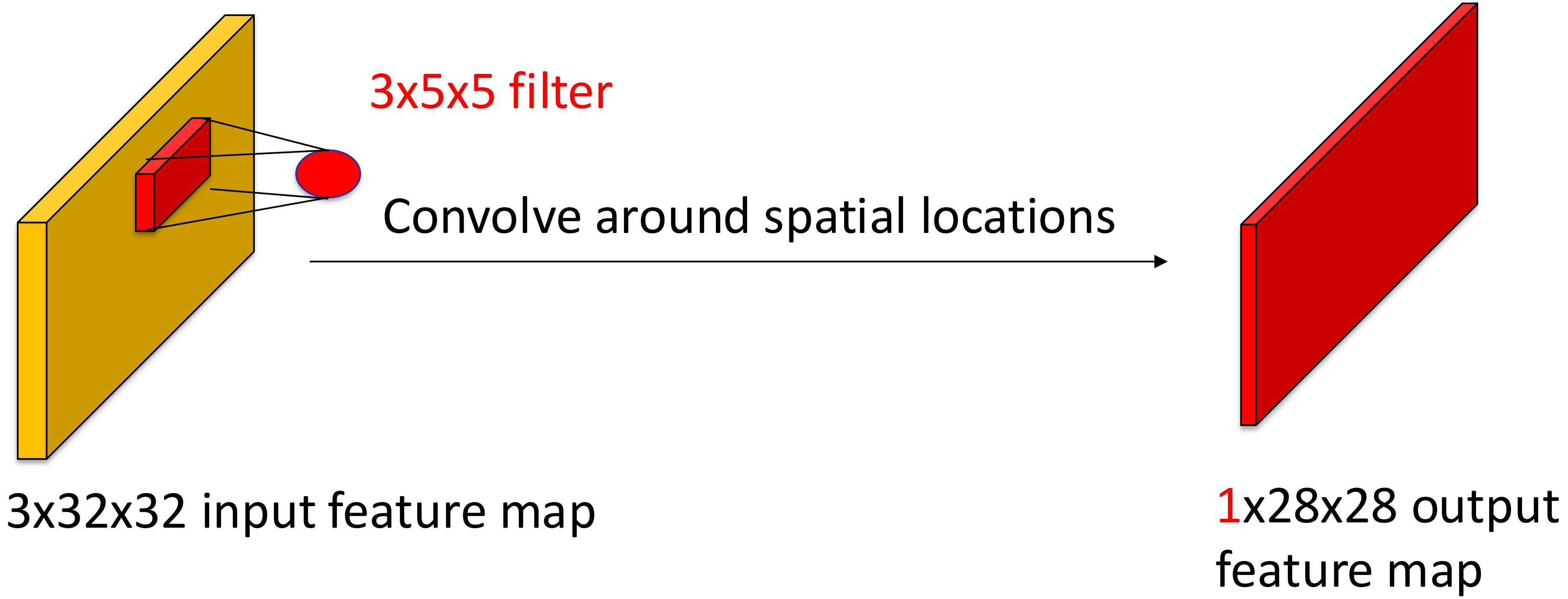
Applying more filters

- Let's add another 3-D filter on the input feature map.
- Slide the filter over the image spatially and obtain another 3-D output feature map.



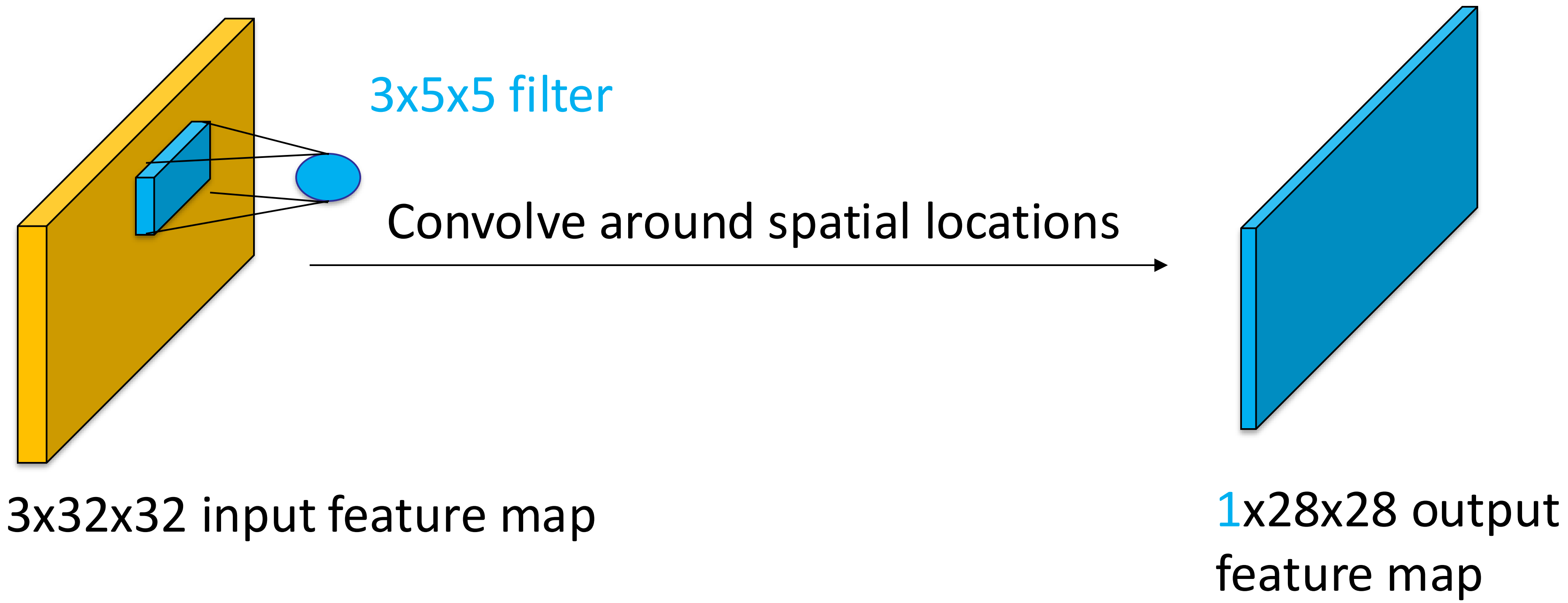
Applying more filters

- Let’s add another 3-D filter on the input feature map.
- Slide the filter over the image spatially and obtain another 3-D output feature map.



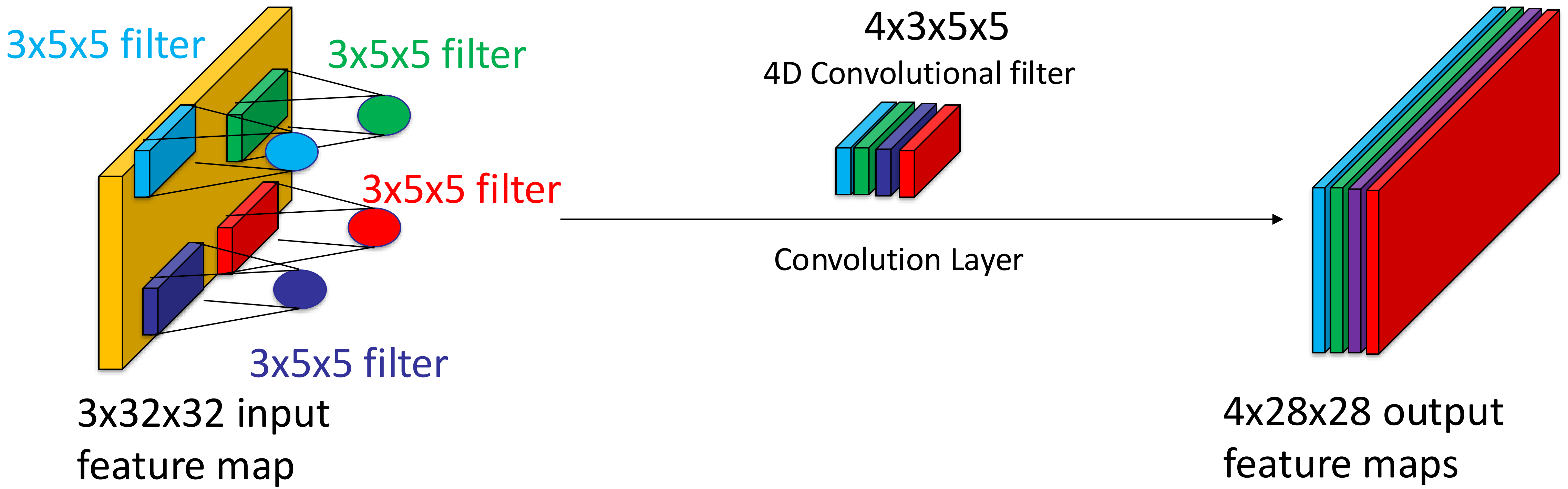
Applying more filters

- Let’s add another 3-D filter on the input feature map.
- Slide the filter over the image spatially and obtain another 3-D output feature map.



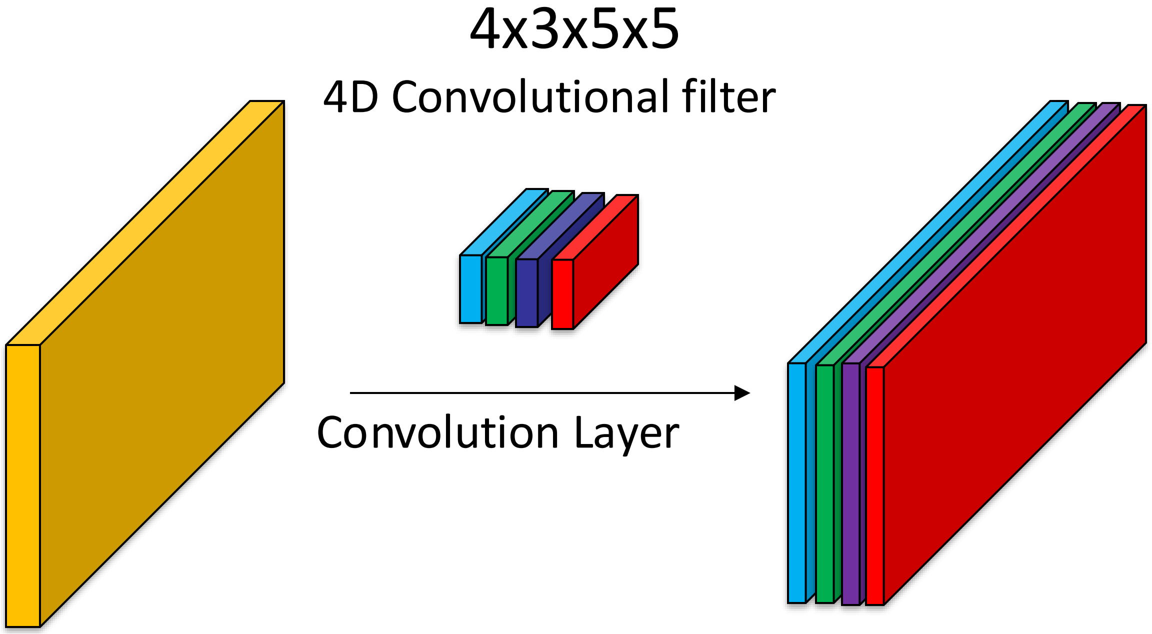
Combing the results

- Stack all the 4 3-D filters into a **4-D filter** with shape 4x3x5x5.
- The output feature maps are also concatenated together into a 3-D output feature map with shape 4x28x28.



Summary of convolution

- 3-D input feature map
- 4-D convolutional filter
- 3-D output feature map



3x32x32 input feature map

4x28x28 output feature maps

Convolutional Filter shape:
(N, I, H, W)<->(4, 3, 5, 5)

N: number of 3-D filters in the convolutional layer
I: number of input channels to the convolution operation
H: height of convolutional kernel
W: width of convolutional kernel

(N, I) are more related to **channel-wise** feature extraction, whereas **(H,W)** are more related to **spatial** feature extraction.

Convolution layer shape rule

- Assume the input feature map has a shape of $\mathrm{C}_1 \times \mathrm{H}_1 \times \mathrm{W}_1$.
- Convolution Layer configuration:
  – Number of filters N
  – Convolution kernel size K
  – Stride for convolution S
  – Padding for each boarder P

- The output feature map has the following shape $\mathrm{C}_2 \times H_2 \times W_2$, where

$$W_2 = \left\lfloor \frac{W_1 - K + 2P}{S} \right\rfloor + 1$$

$$H_2 = \left\lfloor \frac{H_1 - K + 2P}{S} \right\rfloor + 1$$

$$C_2 = \mathrm{N}$$

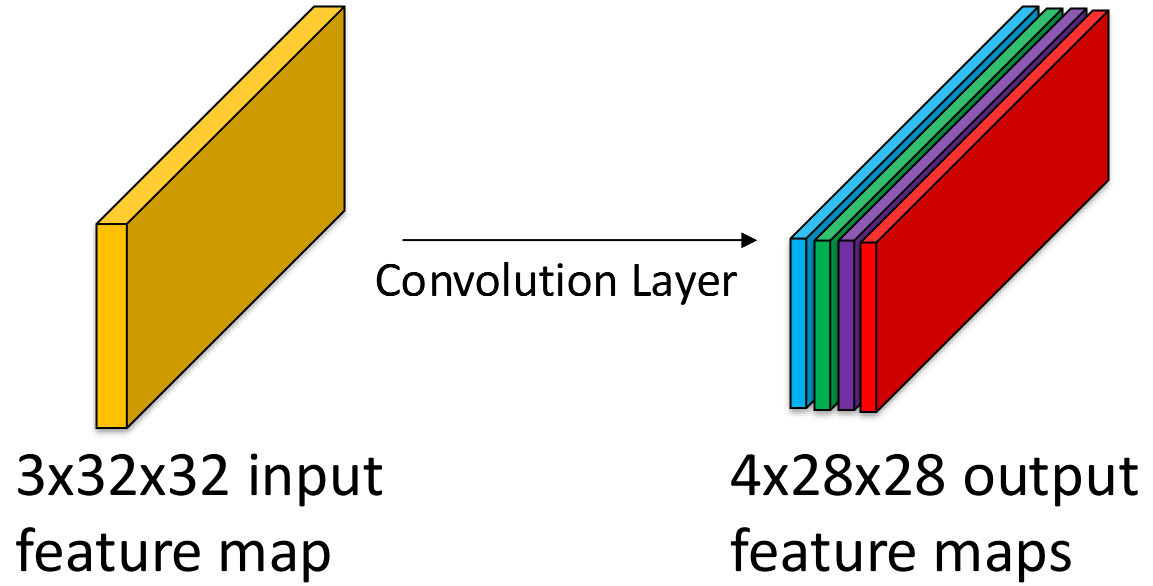
Weight and computation amount

Input feature map $C_1 \times \mathrm{H}_1 \times \mathrm{W}_1$, output $C_2 \times H_2 \times W_2$, kernel size $K \times K$

Weight sharing: All values within 1 output channel comes from the same 3-D convolutional filter, with weight # $C_1 \times K \times K$

C_2 output channels comes from C_2 3-D filters

Total weight elements: $C_2 \times C_1 \times K \times K$



Total weight elements:
$4 \times 3 \times 5 \times 5$

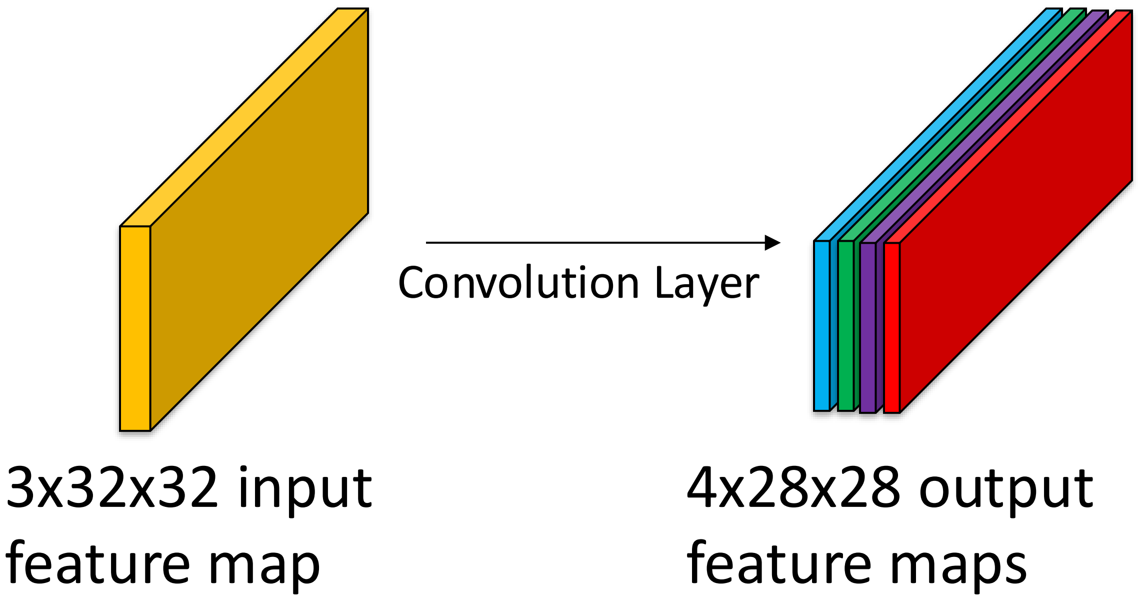
Weight and computation amount

Input feature map $C_1 \times \mathrm{H}_1 \times \mathrm{W}_1$, output $C_2 \times H_2 \times W_2$, kernel size $K \times K$

Multiply & Accumulation (MAC): Every time the 3-D filter is applied on the input feature map involves $C_1 \times K \times K$ MACs, and lead to 1 value in the output feature map

$C_2 \times H_2 \times W_2$ values in the output feature map

Total MACs: $C_1 \times K \times K \times C_2 \times H_2 \times W_2$



Total MACs:

$3 \times 5 \times 5 \times 4 \times 28 \times 28$

Convolutional layers are **Computation-intensive**

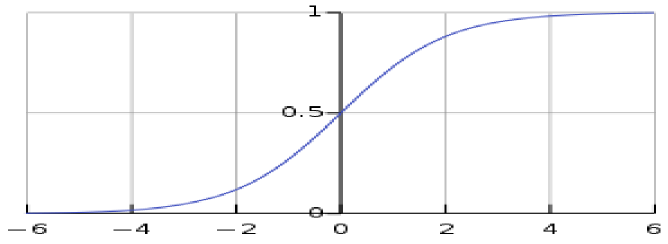
Activation functions

Stacking multiple convolutional layers

- Each convolutional layer should be followed by a nonlinear activation function to create nonlinear functional mappings.

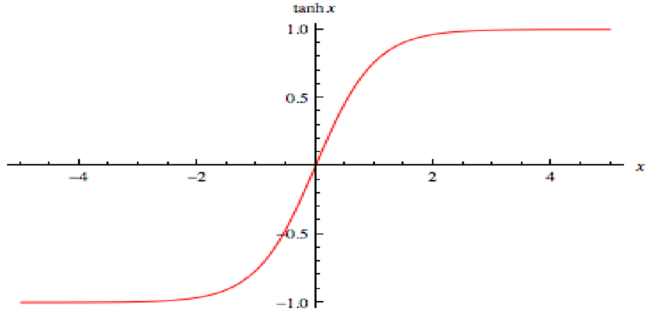
Sigmoid

$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$



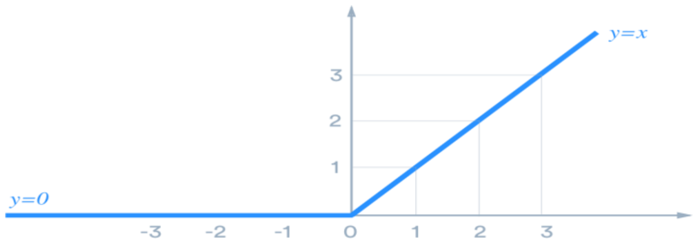
Hyperbolic tangent (tanh)

$$\tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$



Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}(z) = \max(z, 0)$$

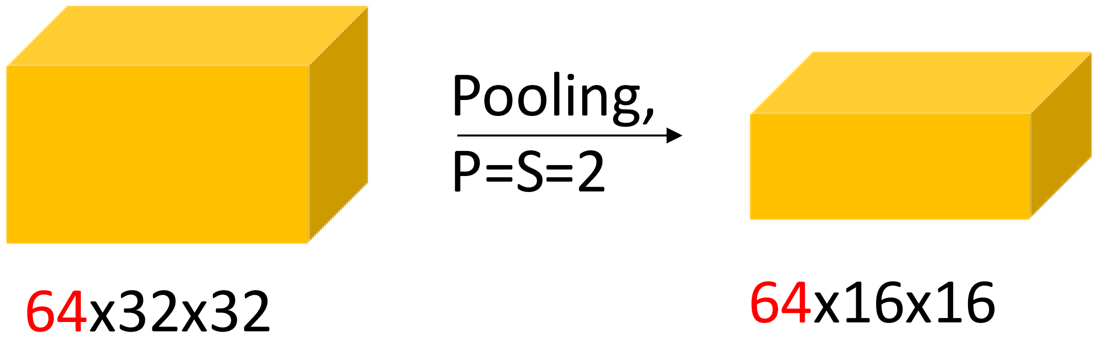


We will cover more details about activation functions in **CNN: Training**.

Pooling: Reducing input size regularly

Input feature map must be down-scaled regularly to prevent CNNs from being too large.

- Pooling down-samples input feature map to extract a concise and manageable knowledge representation.



Pooling is conducted independent of channels

Pooling size (P)
The region to down-sample features according to pooling policy.

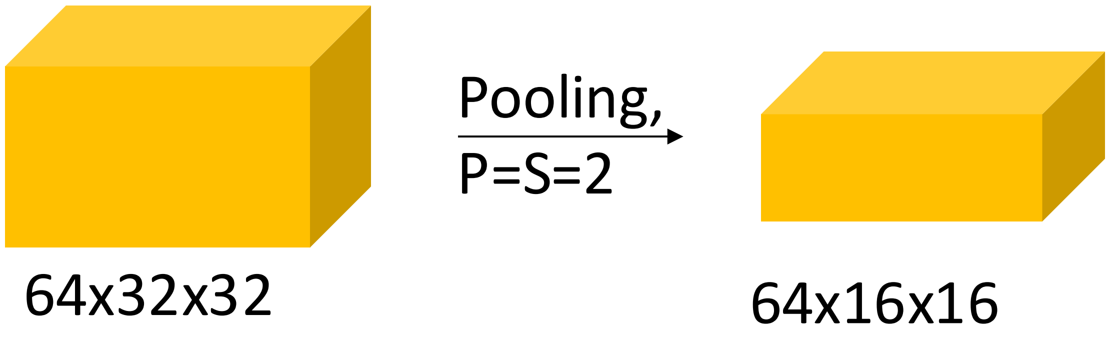
Strides (S)
The amount by which pooling region slides over the input feature map.

Generally, pooling size is equal to the stride in the pooling layer.

Max pooling

- **Max Pooling** only selects the maximum value within each pool region.

Pooling conducted along heights and width dimension of the feature map, not changing the depth.



Pooling size (P)
the region to down-sample features according to pooling policy.

1	3	2	4	5	-1
7	-6	-3	-2	-6	-3
2	0	9	3	8	6
0	1	2	4	-1	-3
-2	0	1	1	-4	-1
-3	4	0	-1	-5	-2

Max Pooling
P=2, S=2

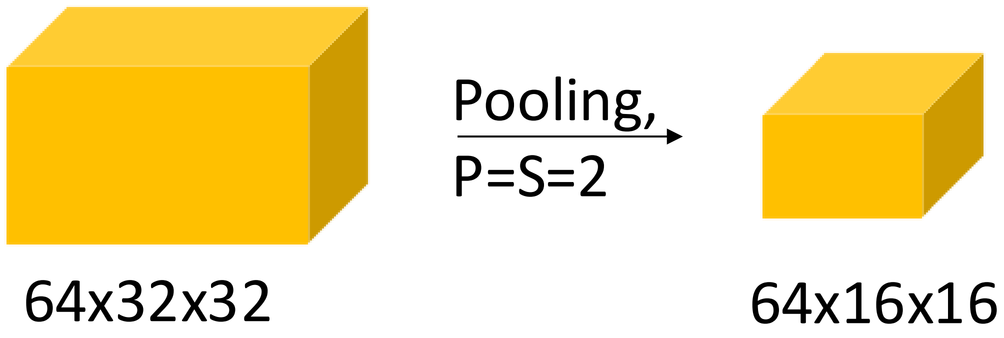
7	4	5
2	9	8
4	1	-1

Strides (S)
The amount by which pooling region slides over the input feature map.

Generally, pooling size is equal to strides in the pooling layer.

Average pooling

- Average Pooling: average the response in each region. However, average pooling introduces **extra computation**.



Instead of selecting the maximum element, we use an average of all items within each pool region as final result.

1	3	2	4	5	-1
7	-6	-3	-2	-6	-3
2	0	9	3	8	6
0	1	2	4	-1	-3
-2	0	1	1	-4	-1
-3	4	0	-1	-5	-2

Average Pool
P=2, S=2

1.25	0.25	-1.25
0.75	4.50	2.50
-0.25	0.25	-3.00

Average pooling consumes extra computation.

((-2)+(-3)+0+4)/4=-0.25

Fully connected layers in CNN

Can 2D convolution layers generate final prediction logits?

- Convolution layers conduct feature extraction. We usually need fully connected layers to generate final output logits.
- Fully connected layers are usually attached as final layers of CNNs.
- Before attaching CNN layers, note that FC layers can only process 1D inputs. Thus, a reshape (flatten) operation is needed.
  - Reshape flattens all dimensions of the input tensor and transforms 3D outputs to 1D inputs for the FC layers.

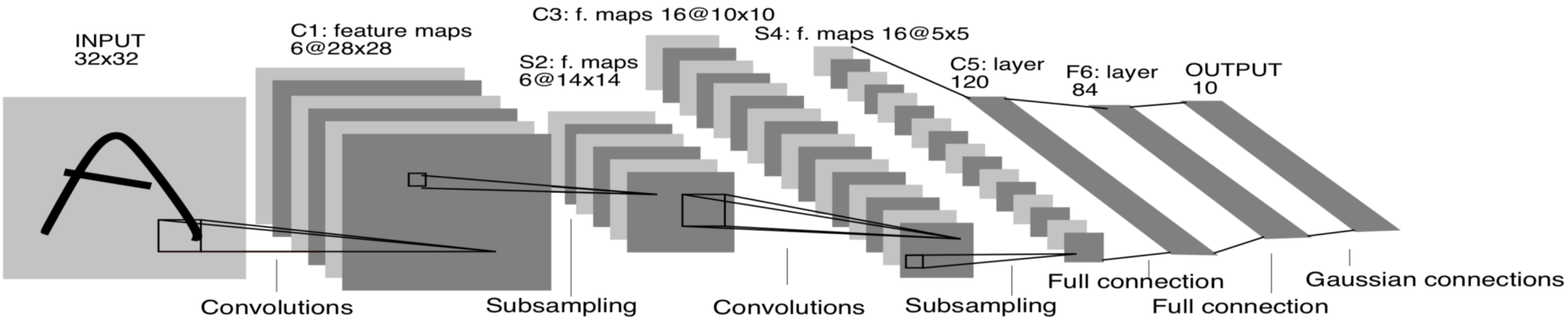
$$D = C_{out} \times H_{out} \times W_{out}$$

Outline

- Layers of Convolutional Neural networks
- Case study: Understanding LeNet-5
- Basic CNN design concepts
- How to implement CNN on PyTorch

LeNet-5: Popular CNN for digits

- Case-study: LeNet-5 for MNIST handwritten digits classification



LeNet Structure:

Convolution – MaxPool – Convolution – MaxPool – Fully connected – Fully connected – Classification

CONV-POOL-CONV-POOL-FC-FC-FC-OUTPUT

LeCun, Yann et al. "Gradient-based learning applied to document recognition." (1998)

Understanding LeNet-5

Q1: What is the output size after the first convolution?

INPUT	32x32
CONV,6	5x5 kernel, no padding
tanh	
POOL	2x2 strides & pooling size
CONV,16	5x5 kernel, no padding
tanh	
POOL	2x2 strides & pooling size
Reshape	5x5x16=400 units
FC,120	
tanh	
FC,84	
tanh	
FC,10	
output	

Understanding LeNet-5

Q1: What is the output size after the first convolution?

A1: Feature width/height: 32-5+1=28

output channels: 6; Output size: 28x28x6

INPUT	32x32
CONV,6	5x5 kernel, no padding
tanh	
POOL	2x2 strides & pooling size
CONV,16	5x5 kernel, no padding
tanh	
POOL	2x2 strides & pooling size
Reshape	5x5x16=400 units
FC,120	
tanh	
FC,84	
tanh	
FC,10	
output	

Understanding LeNet-5

Q1: What is the output size after the first convolution?

A1: Feature width/height: 32-5+1=28

output channels: 6; Output size: 28x28x6

Q2: What is the output size after the first pooling layer?

INPUT	32x32
CONV,6	5x5 kernel, no padding
tanh	
POOL	2x2 strides & pooling size
CONV,16	5x5 kernel, no padding
tanh	
POOL	2x2 strides & pooling size
Reshape	5x5x16=400 units
FC,120	
tanh	
FC,84	
tanh	
FC,10	
output	

Understanding LeNet-5

Q1: What is the output size after the first convolution?

A1: Feature width/height: 32-5+1=28

output channels: 6; Output size: 28x28x6

Q2: What is the output size after the first pooling layer?

A2: Pooling applied to height and width only, not channels; Width/height: 28/2=14

Output size: 14x14x6

INPUT	32x32
CONV,6	5x5 kernel, no padding
tanh	
POOL	2x2 strides & pooling size
CONV,16	5x5 kernel, no padding
tanh	
POOL	2x2 strides & pooling size
Reshape	5x5x16=400 units
FC,120	
tanh	
FC,84	
tanh	
FC,10	
output	

Understanding LeNet-5

Q1: What is the output size after the first convolution?

A1: Feature width/height: 32-5+1=28
output channels: 6; Output size: 28x28x6

Q2: What is the output size after the first pooling layer?

A2: Pooling applied to height and width only, not channels; Width/height: 28/2=14
Output size: 14x14x6

Q3: How many parameters are there in the first fully connected layer?

Understanding LeNet-5

Q1: What is the output size after the first convolution?

A1: Feature width/height: 32-5+1=28
output channels: 6; Output size: 28x28x6

Q2: What is the output size after the first pooling layer?

A2: Pooling applied to height and width only, not channels; Width/height: 28/2=14
Output size: 14x14x6

Q3: How many parameters are there in the first fully connected layer?

A3: Feature width/height for the second pooling is (14-5+1)/2=5. Number of output channels is 16. Therefore, there are 5x5x16x120=48000 parameters in the first fully-connected layer.

Outline

- Layers of Convolutional Neural networks
- Case study: Understanding LeNet-5
- Basic CNN design concepts
- How to implement CNN on PyTorch

How to design CNNs?

What can we learn from LeNet-5?

CNN design rules

What can we learn from LeNet-5?

- CNN has 3 types of fundamental layers: Convolutional layers, Pooling layers and fully-connected layers.
- Convolutional layers are responsible for extracting spatial features and computing output of neurons that are connected to local regions in the input.
- Pooling layers will perform a down-sampling operation along the spatial dimensions.
- Fully connected layers are responsible to produce the classification results.
- Typical Design: (CONV*N-POOL)*M-FC*K-OUTPUT
- **Remember to add non-linearity between layers!**

CNN design recommendations

- DO NOT stack more than 3 fully connected layers.

- Stacking convolutional layers gives significant performance boost. The size of receptive fields grows quickly with the number of convolutional layers. Stacking 3-4 convolutional layers before applying the pooling operation may give you the optimal results.

- More design ideas will be discussed in **CNN Architectures** and **Neural Architecture Search**.

Outline

- Layers of Convolutional Neural networks
- Case study: Understanding LeNet-5
- Basic CNN design concepts
- How to implement CNN on PyTorch

Convolution: PyTorch implementation

torch.nn.Conv2D https://pytorch.org/docs/stable/nn.html

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

```
class MyConv(nn.Module):
    def __init__(self):
        super(MyConv, self).__init__()
        self.my_conv = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3)

    def forward(self, x):
        x = self.my_conv(x)
        return x
```

Initialize a 3x3 convolution kernel with 32 input channels and 64 output channels.

You may also want to add a nonlinearity (activation function).

```
return
torch.nn.functional.relu(self.my_conv(x))
```

Other parameters (i.e. dilation, groups etc.) lead to more advanced architectures, will introduce if we need them in future lectures

Pooling: PyTorch implementation

torch.nn.MaxPool2d

https://pytorch.org/docs/stable/nn.html

CLASS torch.nn.MaxPool2d(*kernel_size*, *stride=None*, *padding=0*, *dilation=1*, *return_indices=False*, *ceil_mode=False*) [SOURCE]

```
class MyMaxPool(nn.Module):
    def __init__(self):
        super(MyMaxPool, self).__init__()
        self.my_max_pool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        return self.my_max_pool(x)
```

Initialize a Max Pooling layer with pool size 2 and stride 2.

torch.nn.AvgPool2d

CLASS torch.nn.AvgPool2d(*kernel_size*, *stride=None*, *padding=0*, *ceil_mode=False*, *count_include_pad=True*, *divisor_override=None*) [SOURCE]

```
class MyAvgPool(nn.Module):
    def __init__(self):
        super(MyAvgPool, self).__init__()
        self.my_avg_pool = nn.AvgPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        return self.my_avg_pool(x)
```

Initialize an Average Pooling layer with pool size 2 and stride 2.

CNN applications

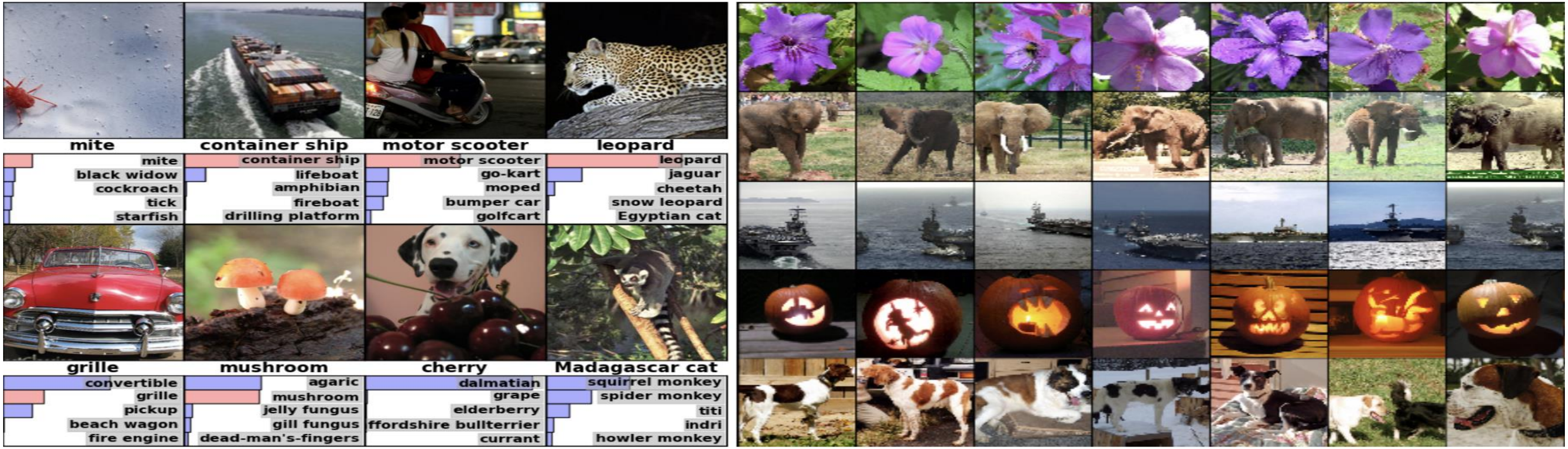
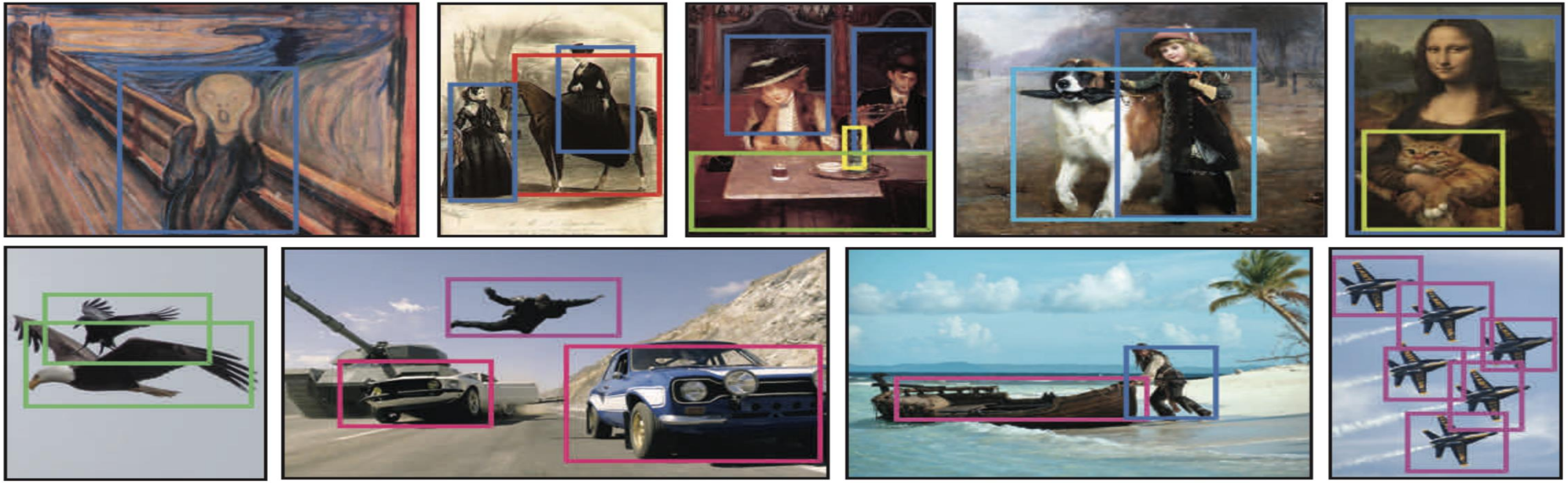


Image Classification

Source of Images: Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

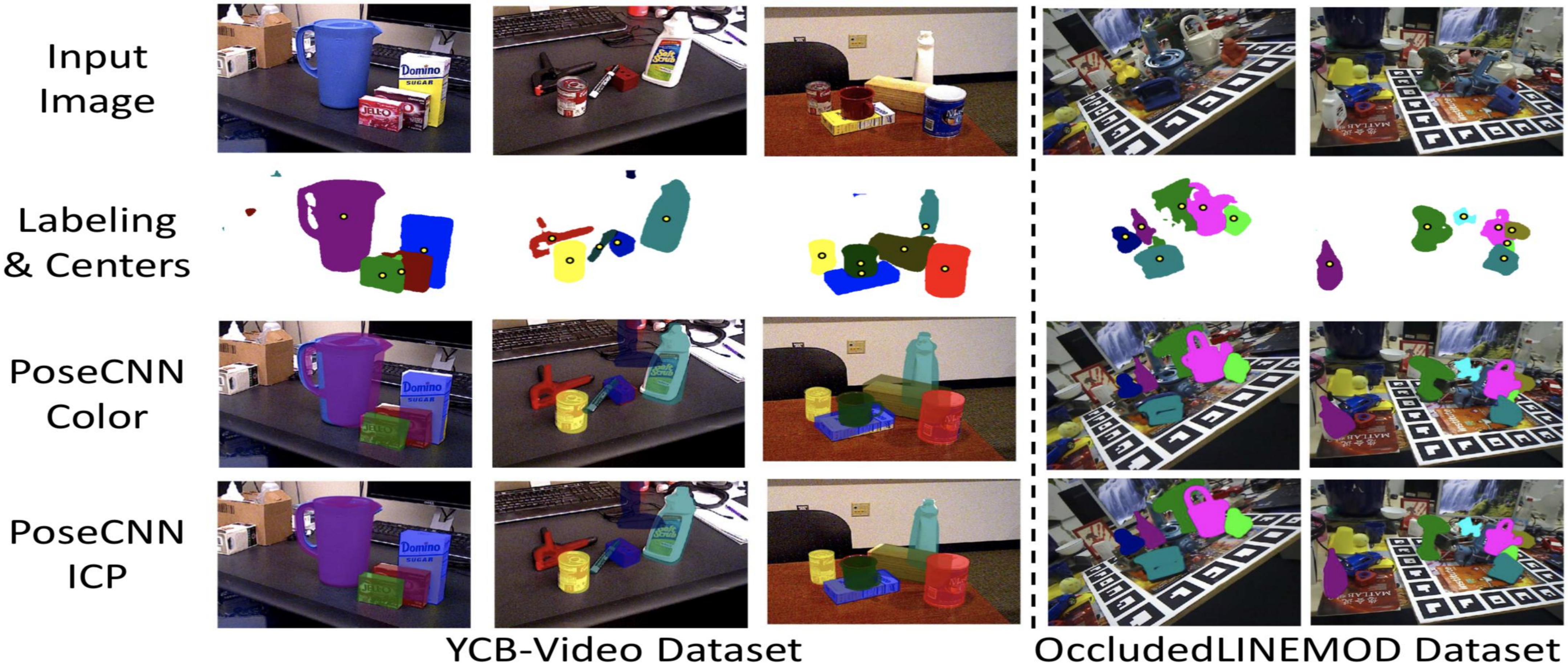
CNN applications



Object Detection

Source of Images: Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

CNN applications



Pose Estimation

Source of Images: Xiang, Yu, et al. "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes." *arXiv preprint arXiv:1711.00199* (2017).

In this lecture, we learned:

- Convolutional Neural networks
  – Calculate Convolution on 3D Image Space.
  – Receptive field & shared weight connections.
  – Down-sampling modules
- Image Classification Applications
  – Handwritten digit recognition: LeNet-5
  – Parameters and MACs in a CNN layer
- Basic CNN design concepts
  – Stack convolutional layers instead of fully-connected layers.
  – Typical design rule:
    (CONV*N-POOL)*M-FC*K-OUTPUT
- How to implement CNN on PyTorch

- Coming up: CNN training (Lec 5, 6 & 7)