Lecture 9

Convolution Neural Network

Image Classification

100×100

$\rightarrow$ Model $\rightarrow$

$$\boldsymbol{y}' = \begin{bmatrix} \vdots \\ 0.2 \\ 0.7 \\ 0.1 \\ \vdots \end{bmatrix} \quad \longleftrightarrow \quad \hat{\boldsymbol{y}} = \begin{array}{r} \\ \text{dog} \\ \text{cat} \\ \text{tree} \\ \\ \end{array} \begin{bmatrix} \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$$

Cross entropy

(All the images to be classified have the same size.)

Cross-Entropy

Intuitively Understanding the Cross Entropy

$$H(P^*|\,P) = -\sum_i \underbrace{P^*(i)}_{\substack{\text{TRUE CLASS}\\ \text{DISTIRBUTION}}} \log \underbrace{P(i)}_{\substack{\text{PREDICTED CLASS}\\ \text{DISTIRBUTION}}}$$

Intuitively Understanding the KL divergence

KL is not symmetric

$$D_{KL}(P||Q) = \sum_i P(i)\log\frac{P(i)}{Q(i)}$$

$$p_1\log\frac{p_1}{q_1} + p_2\log\frac{p_2}{q_2}$$

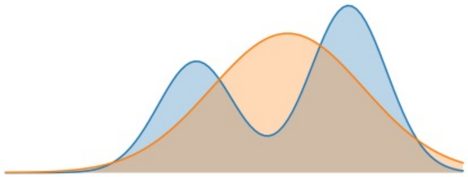
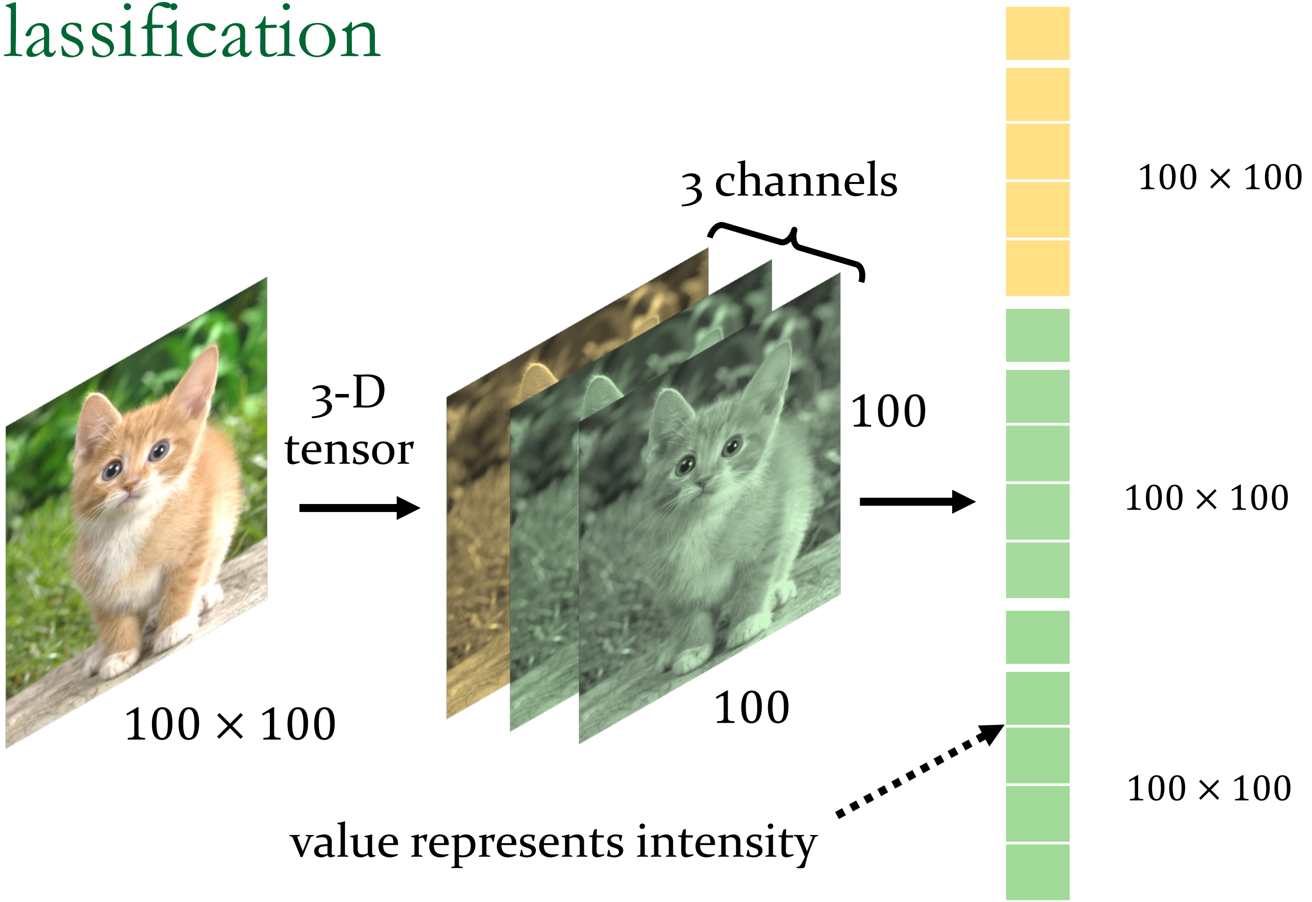
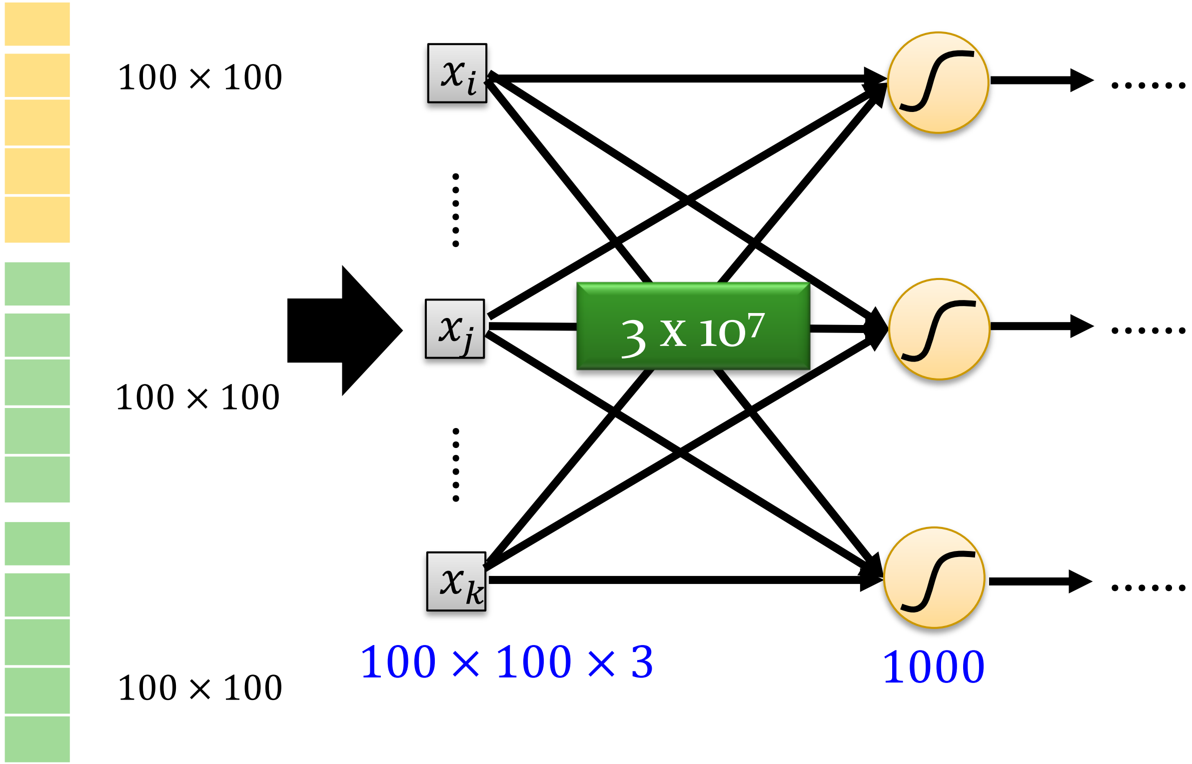


Image Classification

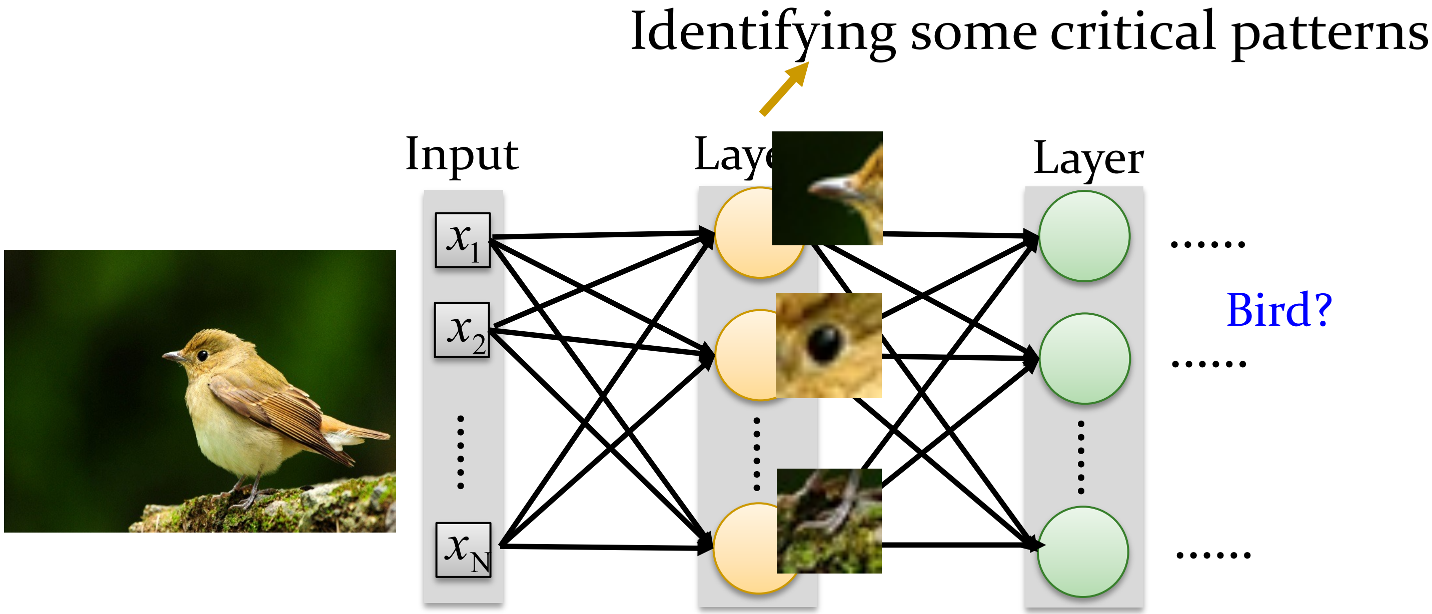


Fully Connected Network



Do we really need "fully connected" in image processing?

Observation 1

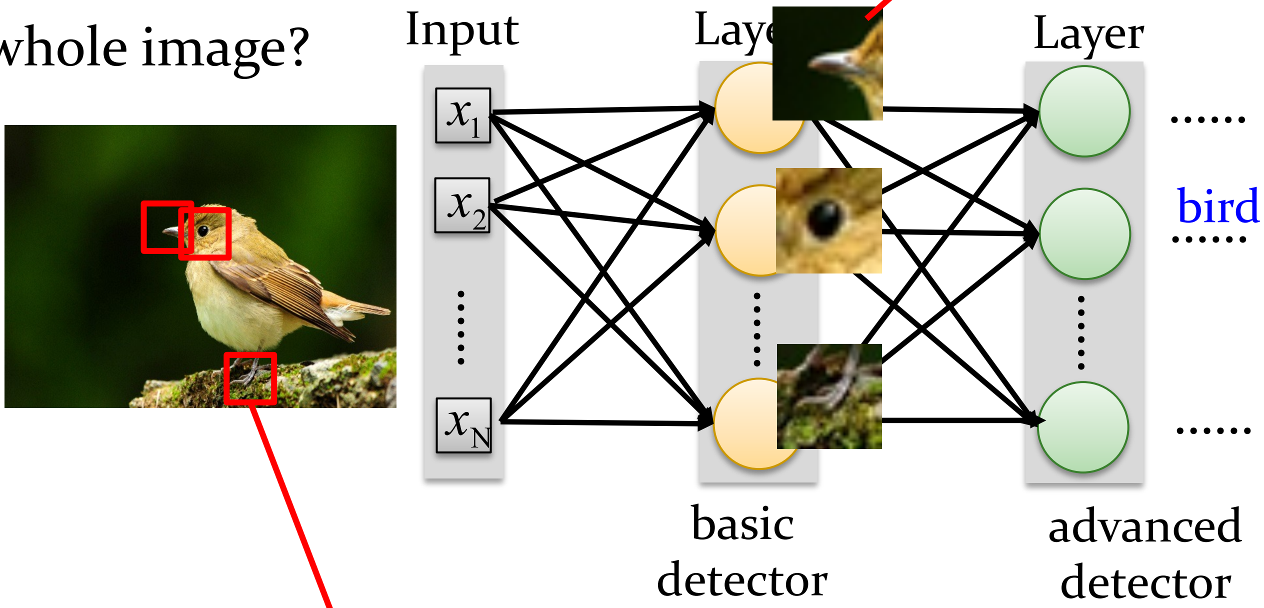


Perhaps human also identify birds in a similar way ... ☺

Observation 1

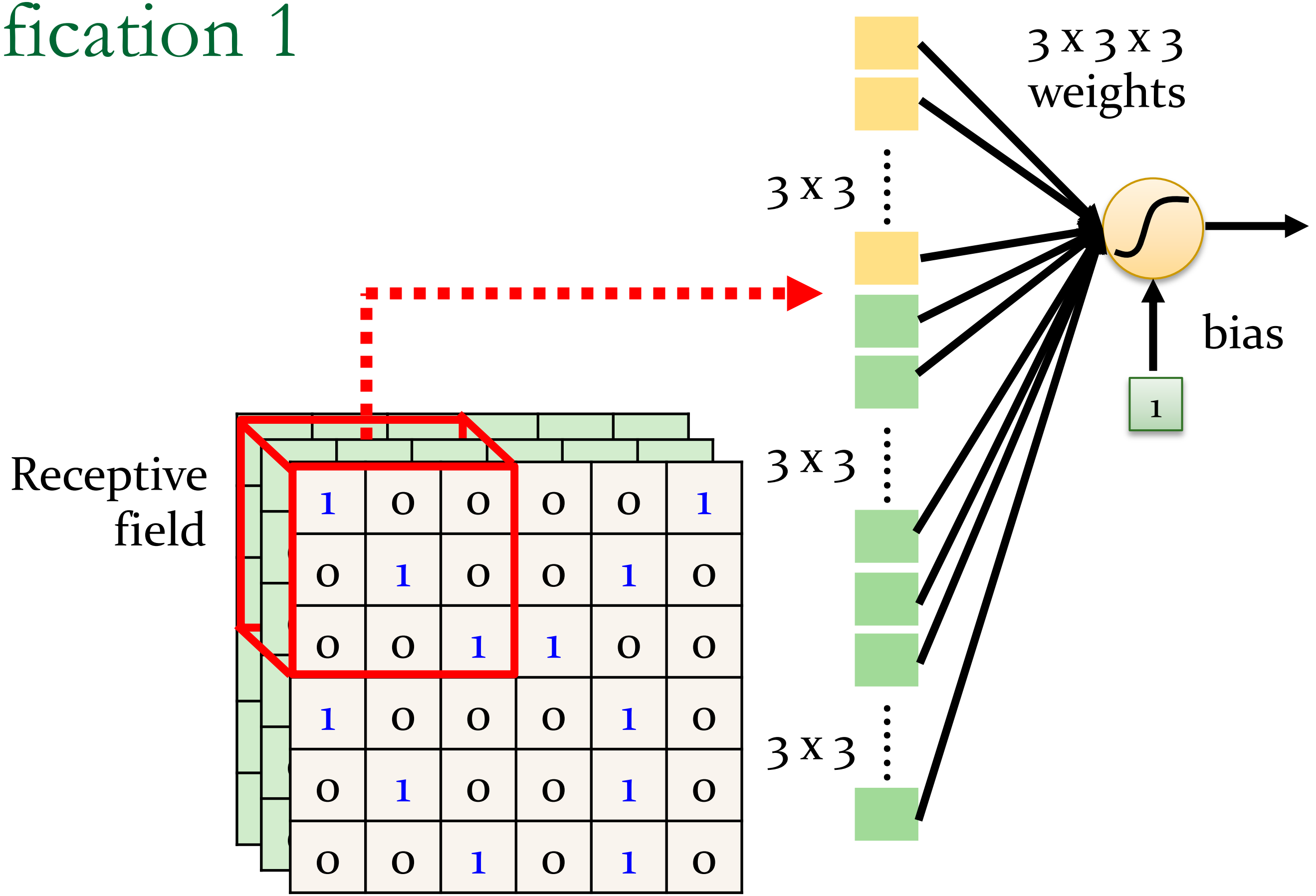
A neuron does not have to see the whole image.

Need to see the whole image?



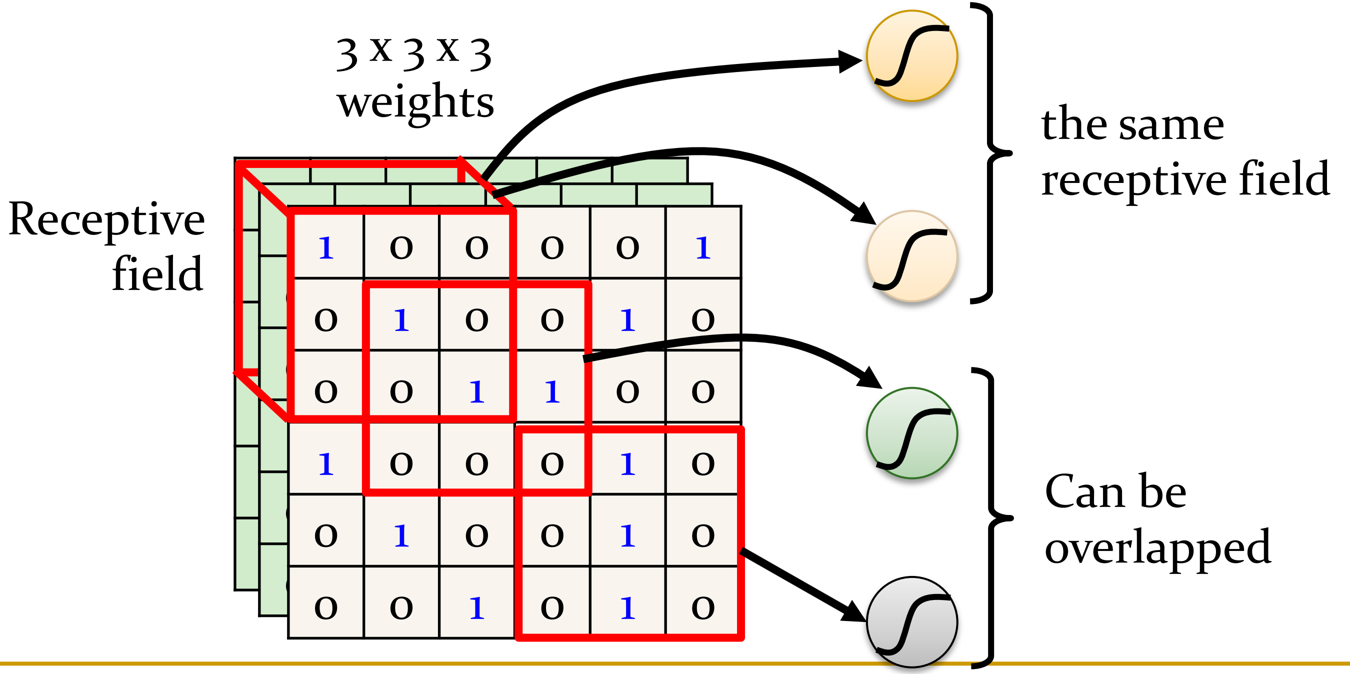
Some patterns are much smaller than the whole image.

Simplification 1



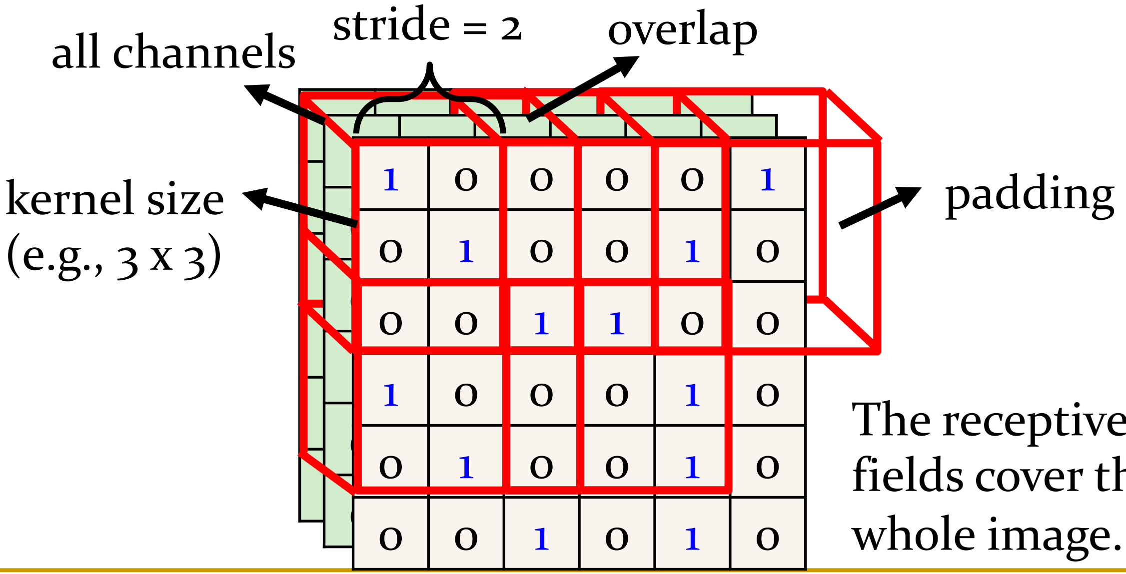
Simplification 1

- Can different neurons have different sizes of receptive field?
- Cover only some channels?
- Not square receptive field?



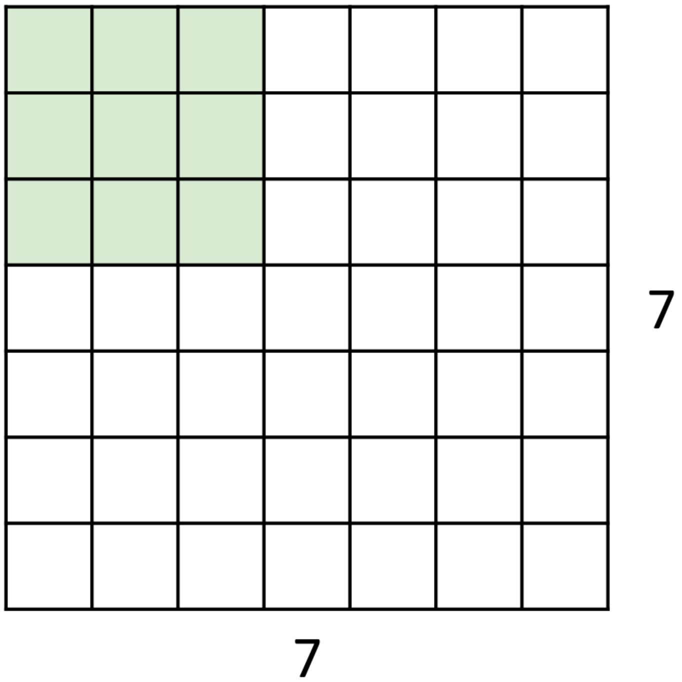
Simplification 1 – Typical Setting

Each receptive field has a set of neurons (e.g., 64 neurons).



The receptive fields cover the whole image.

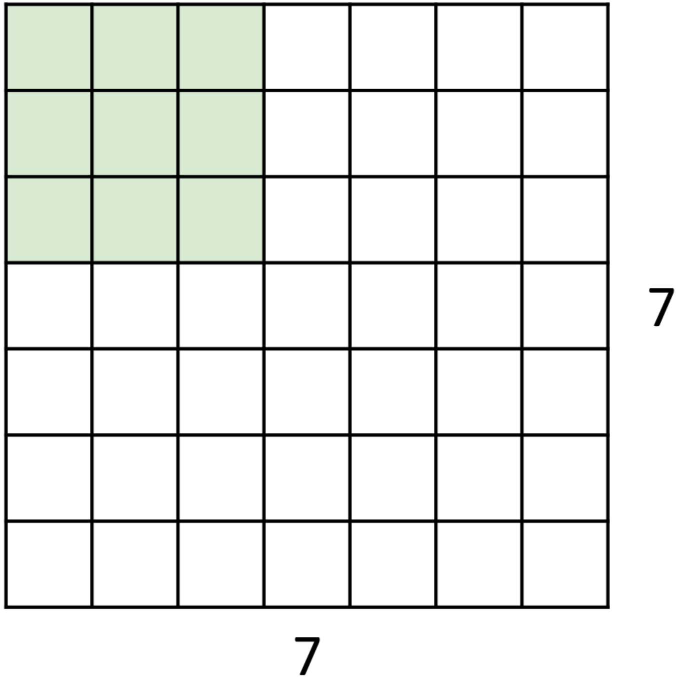
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

Q: How big is output?

Convolution Spatial Dimensions

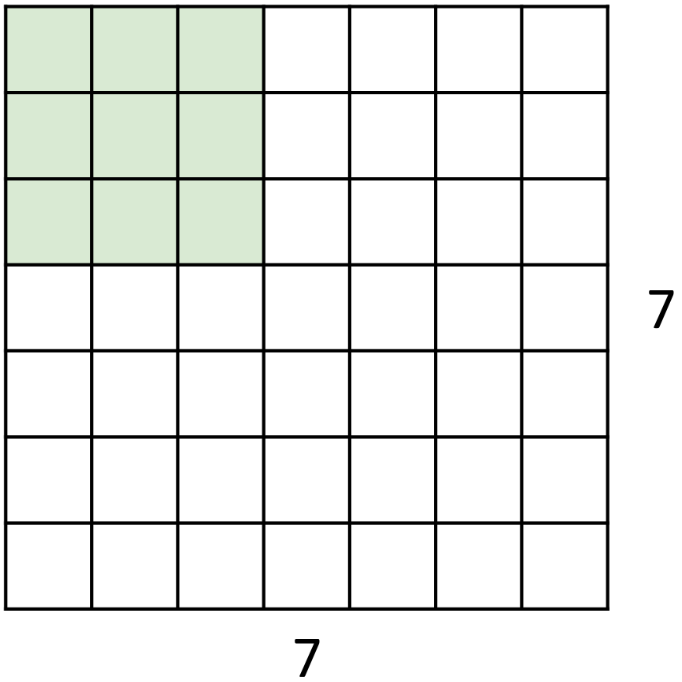


Input: 7x7
Filter: 3x3

Q: How big is output?

Output: 5x5

Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

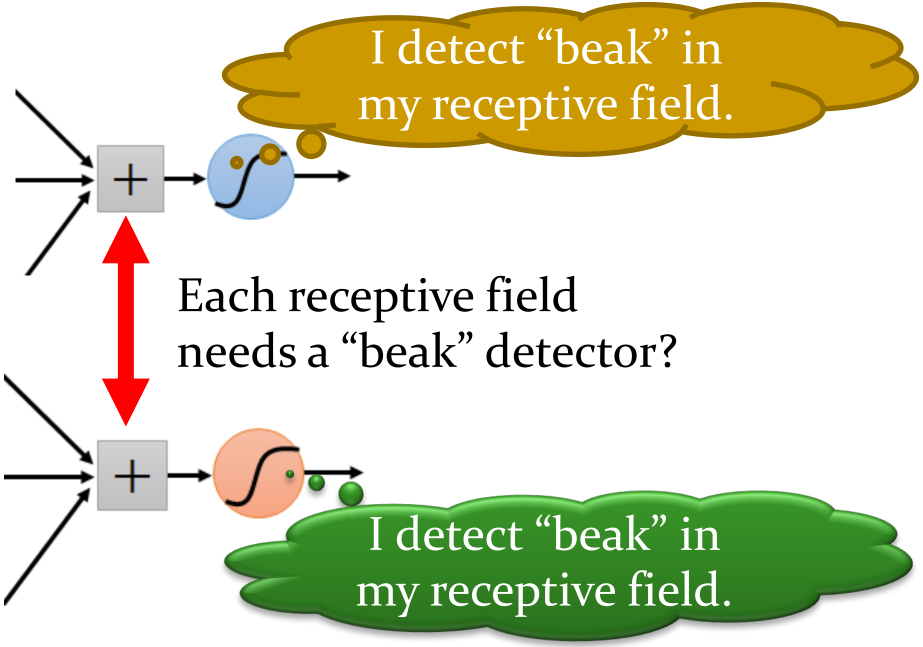
Q: How big is output?

Output: 5x5

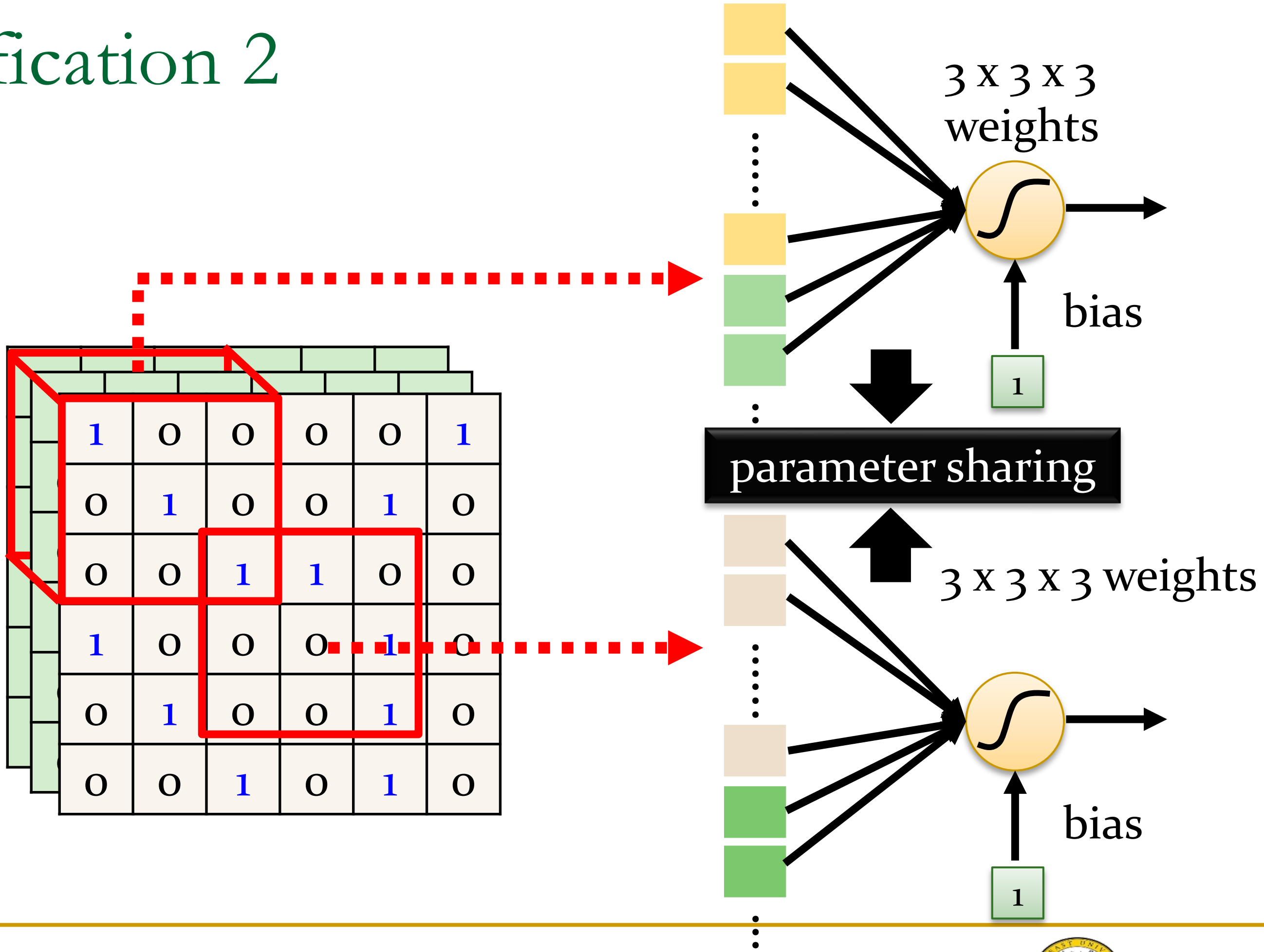
In general:
Input: W
Filter: K
Output: W – K + 1

Observation 2

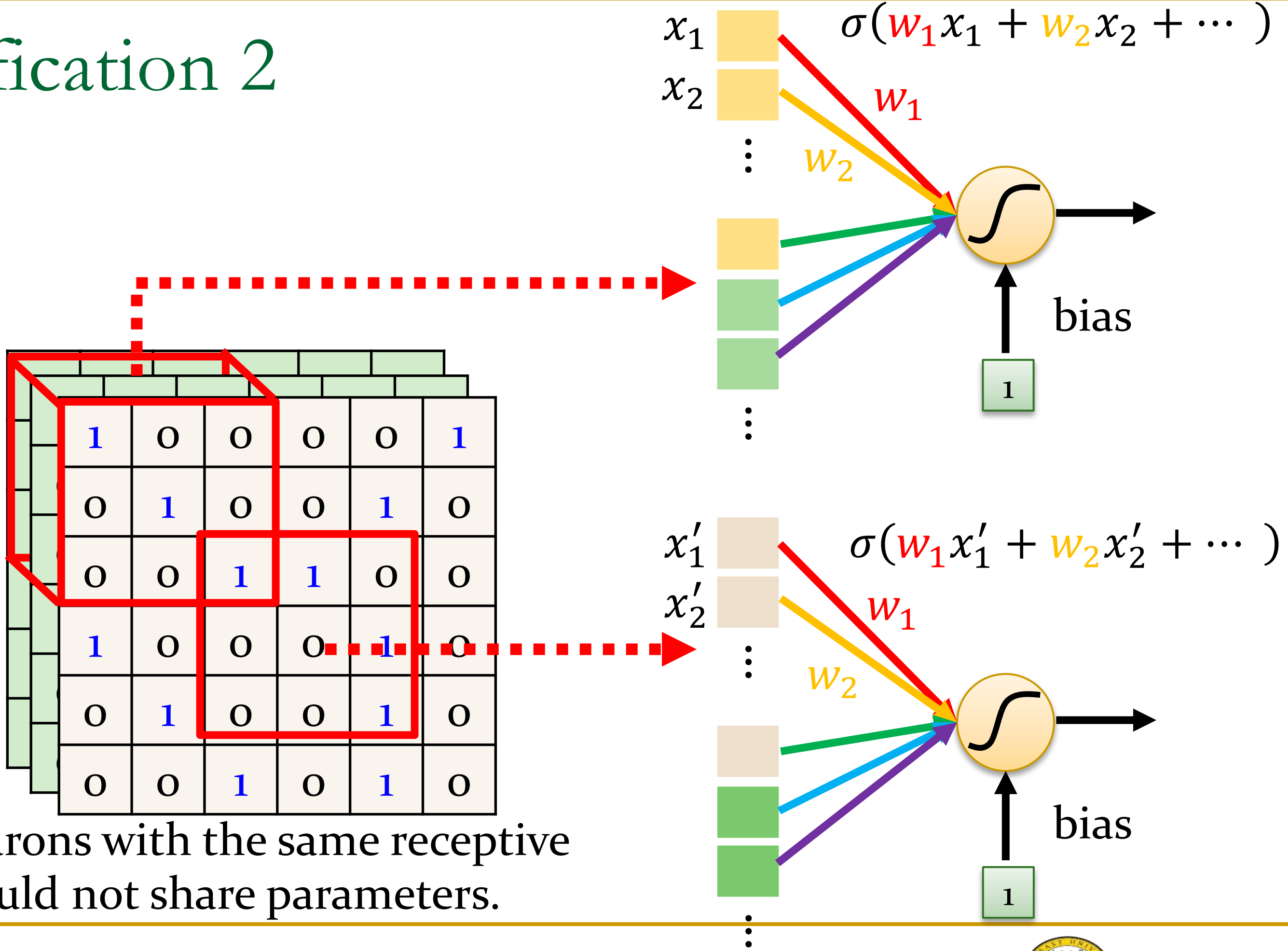
- The same patterns appear in different regions.



Simplification 2



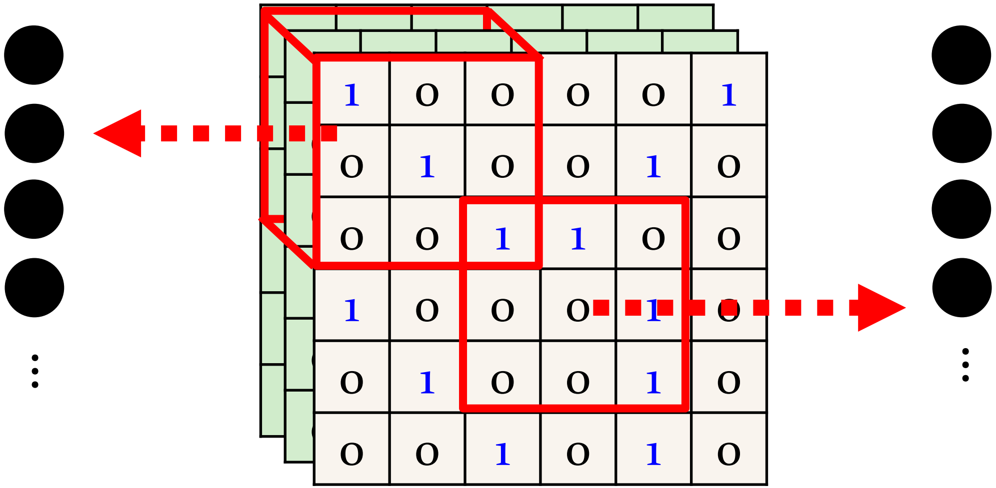
Simplification 2



Two neurons with the same receptive field would not share parameters.

Simplification 2 – Typical Setting

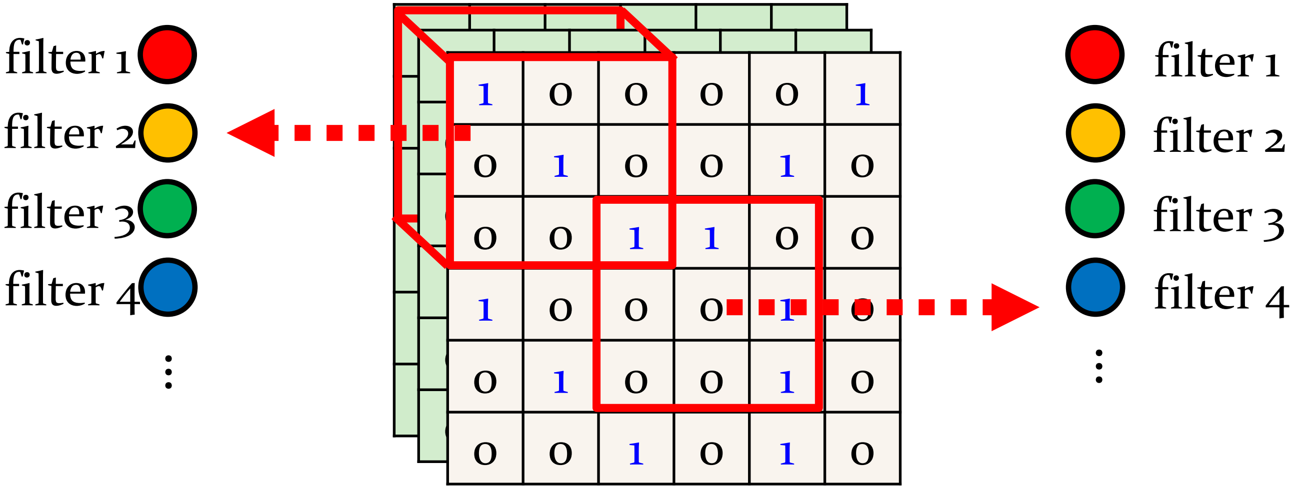
Each receptive field has a set of neurons (e.g., 64 neurons).



Simplification 2 – Typical Setting

Each receptive field has a set of neurons (e.g., 64 neurons).

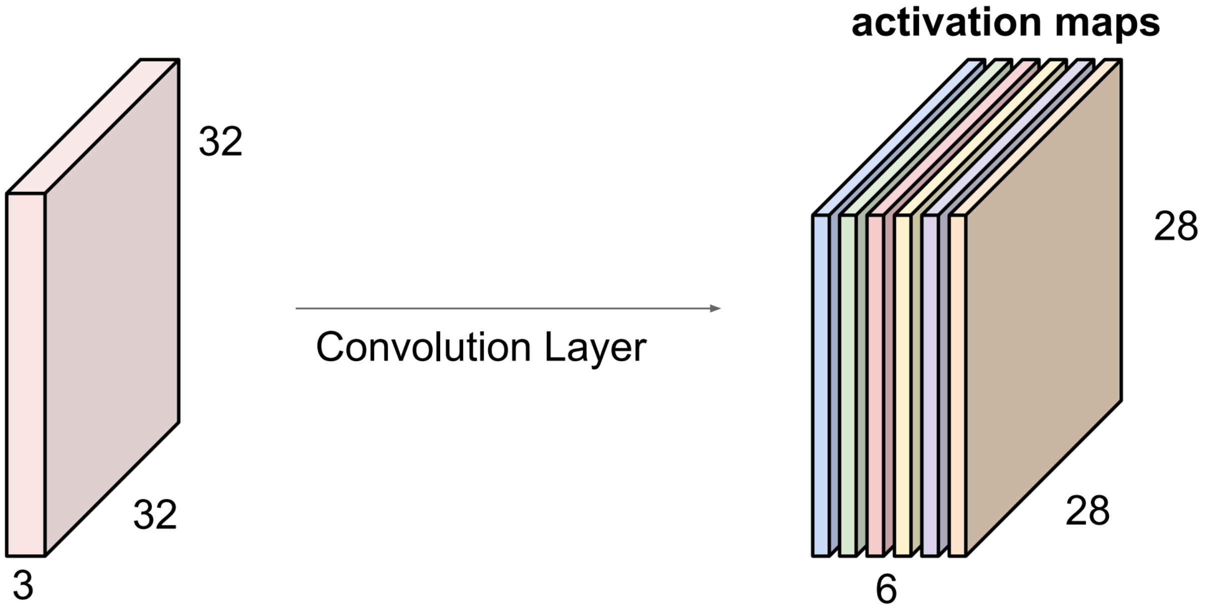
Each receptive field has the neurons with the same set of parameters.



https://cs231n.github.io/assets/conv-demo/

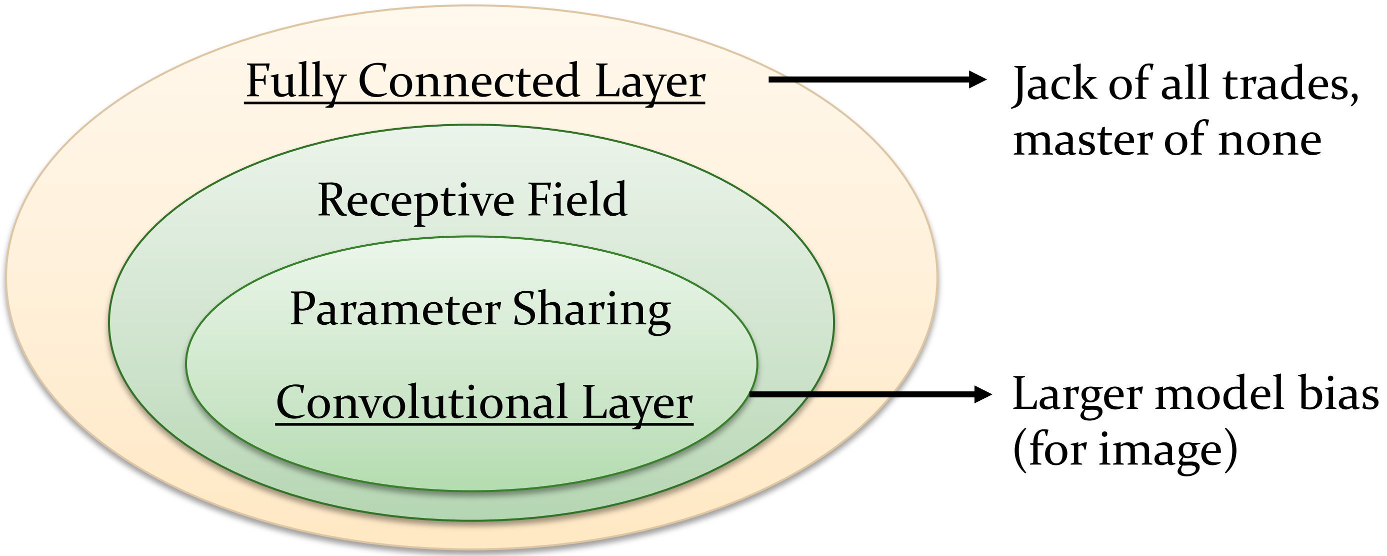
Stacking Convolution Filters

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Benefit of Convolutional Layer



- Some patterns are much smaller than the whole image.
- The same patterns appear in different regions.

Convolutional Layer

Another story based on filter ☺

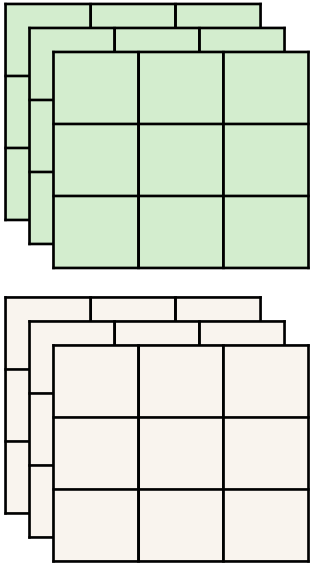
Each filter detects a small pattern (3 x 3 x channel).

Convolution

⋮ ⋮

channel = 3 (colorful)

channel = 1 (black and white)



Filter 1

3 x 3 x channel tensor

Filter 2

3 x 3 x channel tensor

⋮ ⋮

Convolutional Layer

Consider channel = 1
(black and white image)

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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

(The values in the filters are unknown parameters.)

Convolutional Layer

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

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

6 x 6 image

$\boxed{3}$	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
$\boxed{3}$	-2	-2	-1

Convolutional Layer

-1	1	-1
-1	1	-1
-1	1	-1

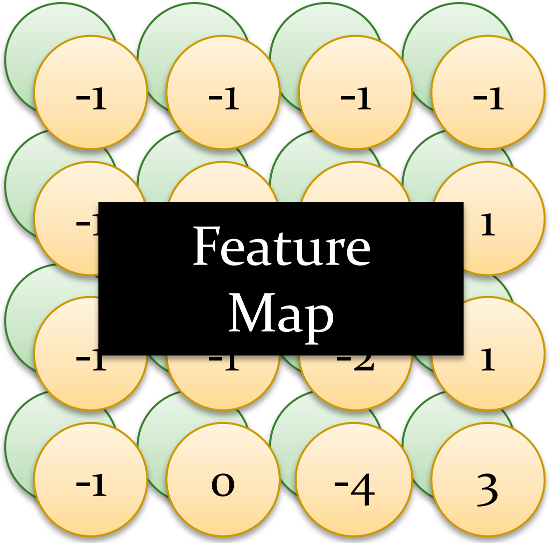
Filter 2

stride=1

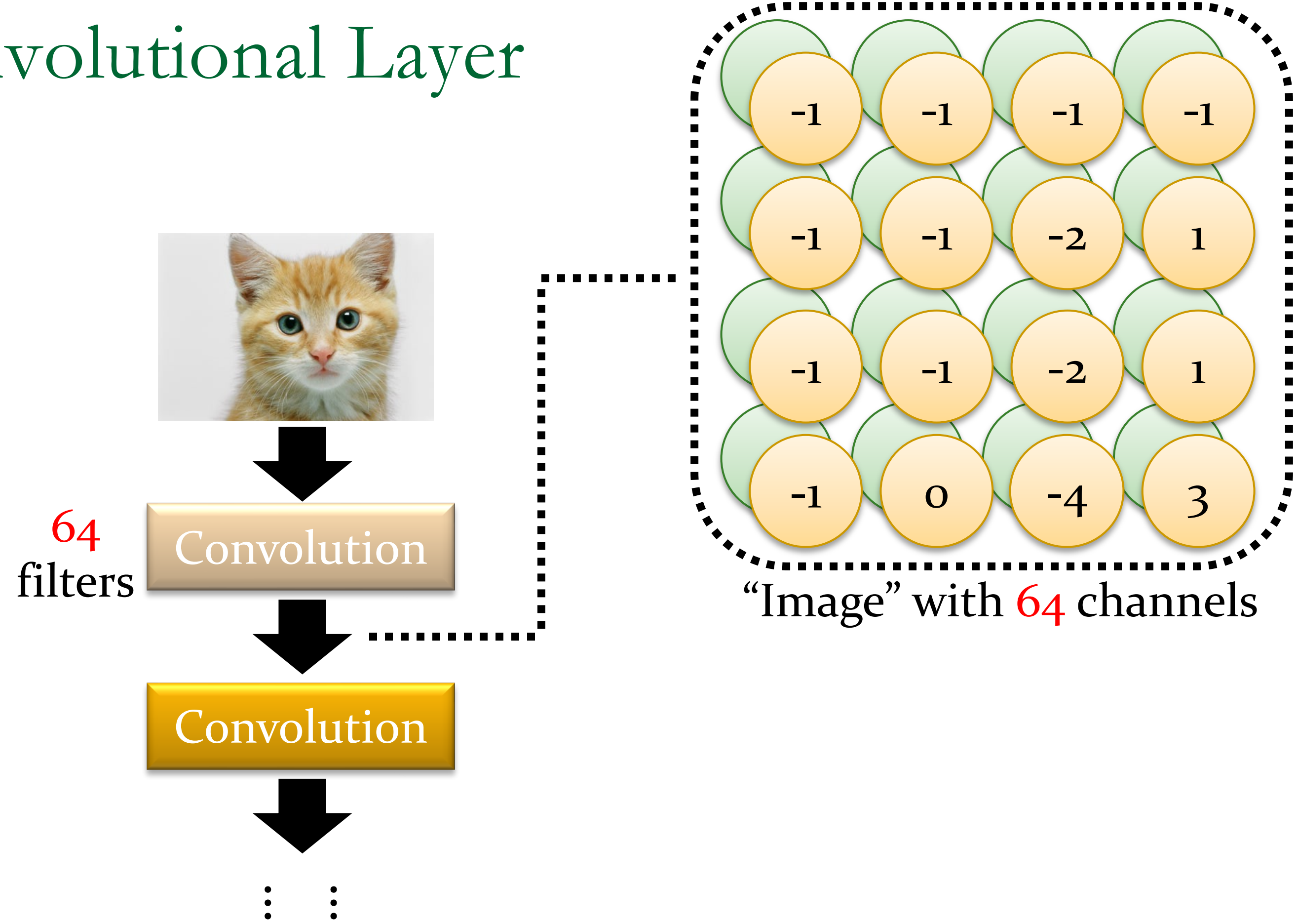
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

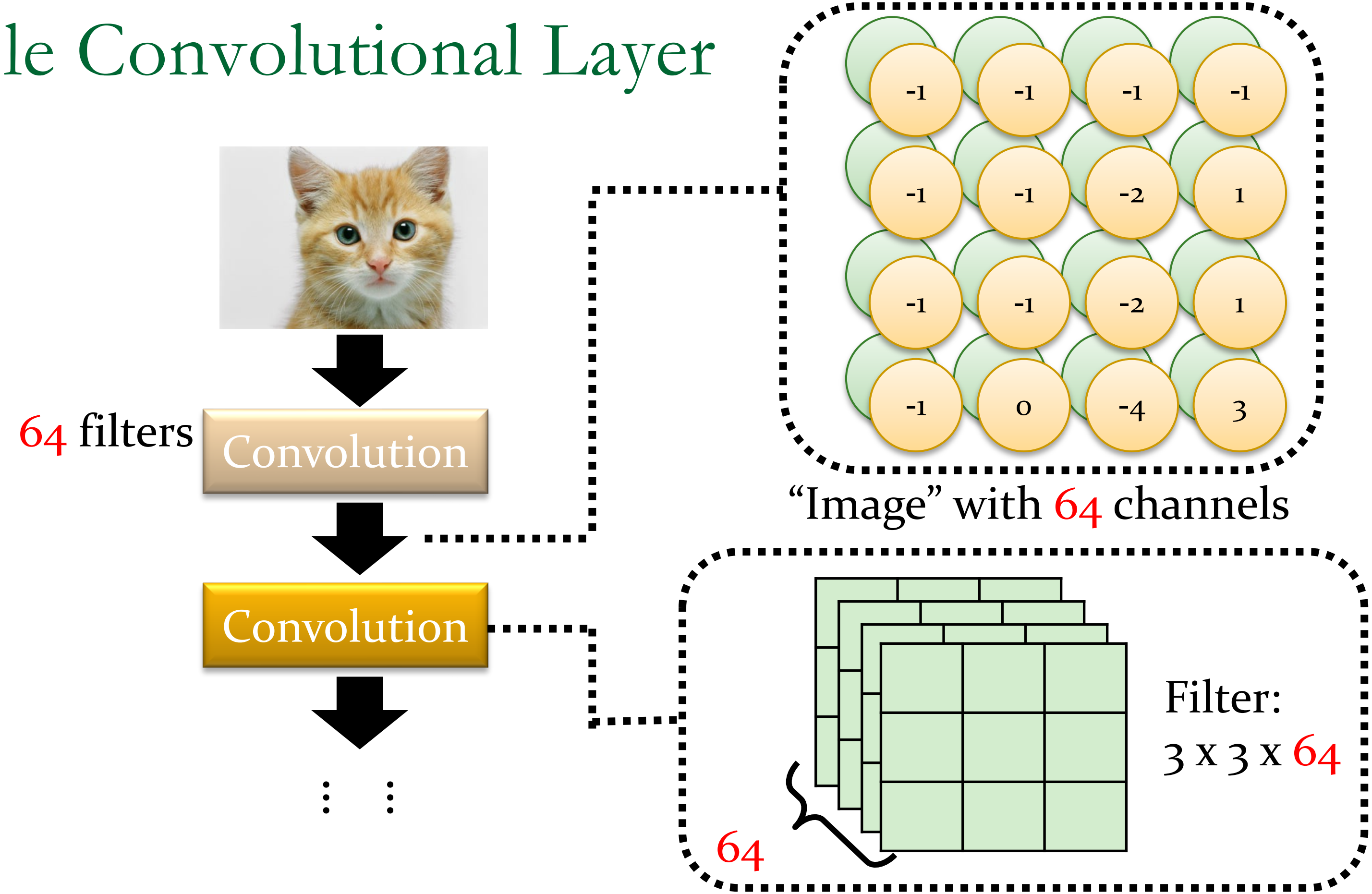
Do the same process for every filter



Convolutional Layer



Multiple Convolutional Layer



Multiple Convolutional Layer

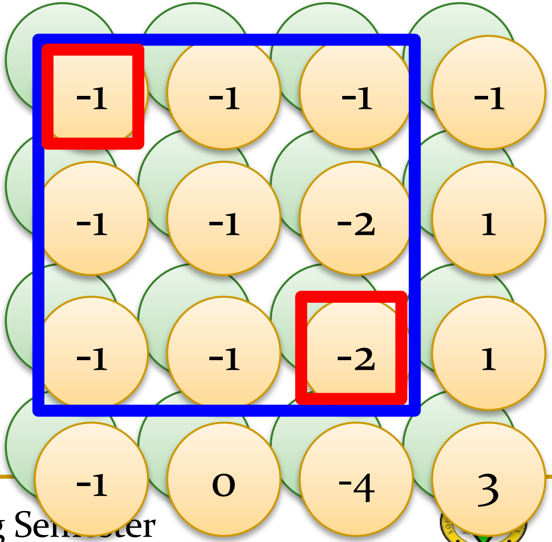
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

64 filters

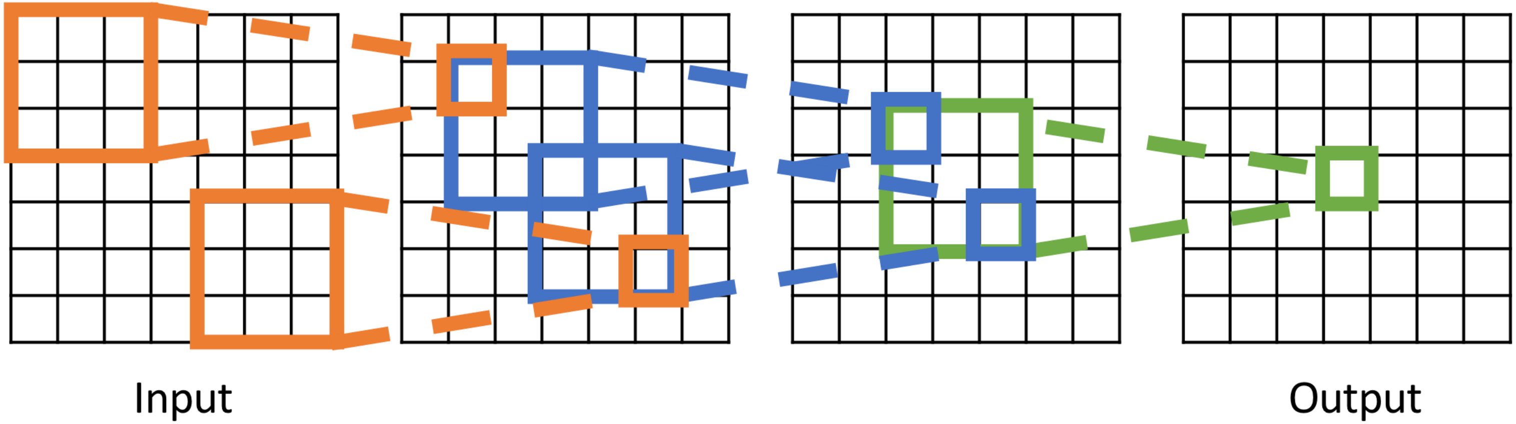
Convolution

Convolution

⋮ ⋮



Receptive Fields



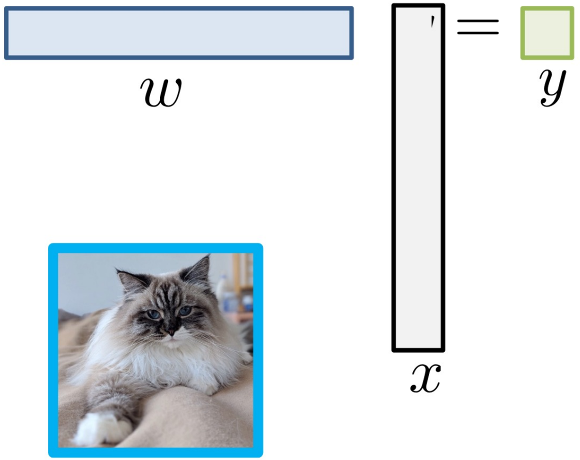
Convolutional Layer

Neuron Version Story	Filter Version Story
Each neuron only considers a receptive field.	There are a set of filters detecting small patterns.
The neurons with different receptive fields share the parameters.	Each filter convolves over the input image.

They are the same story.

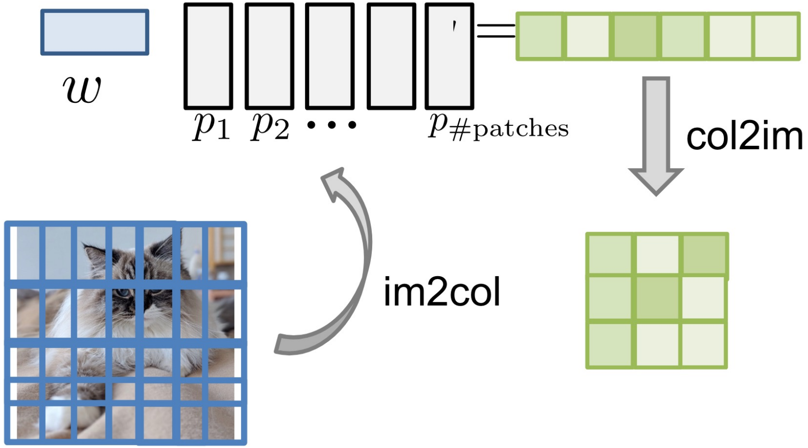
We're still doing matrix multiplications, just localized & shared

Recall one neuron in FC layer:



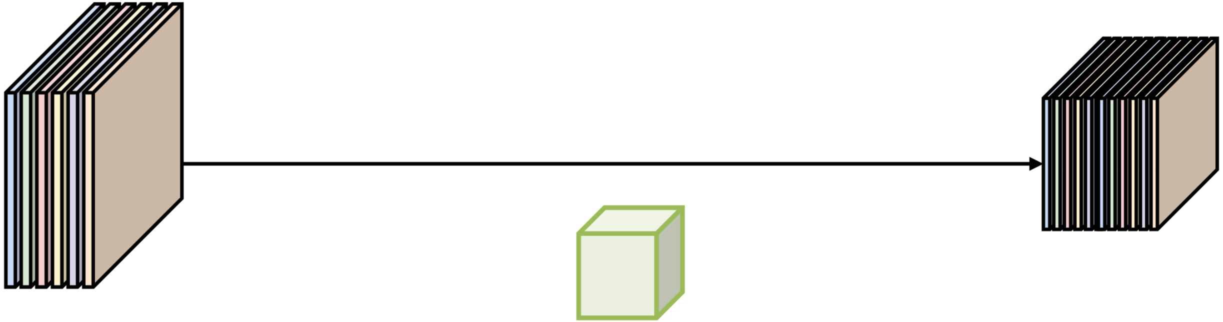
w takes the entire image!

With Conv layer:



Now w takes (overlapping) patches

What needs to be learned?



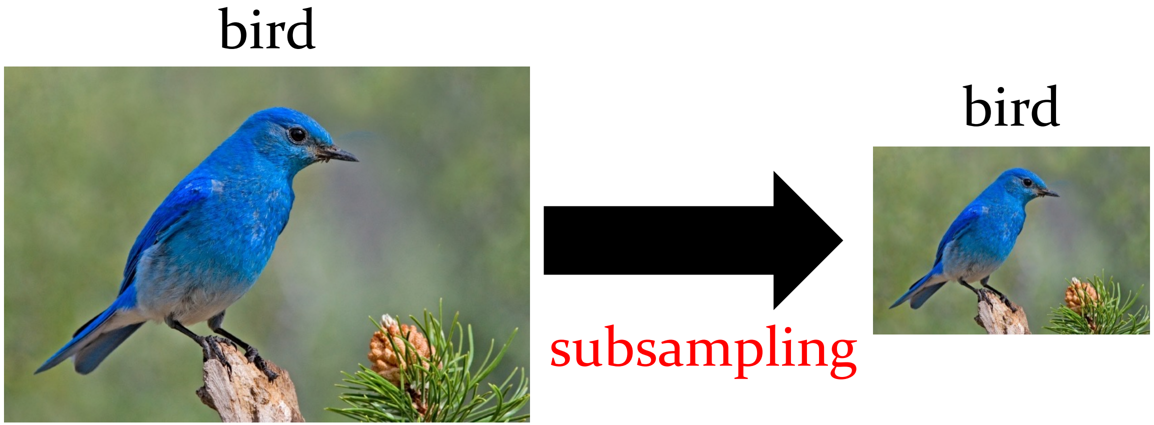
Input:
B, C_in, H, W

W
C_out, C_in, K, K

Output
B, C_out, H, W

Observation 3

- Subsampling the pixels will not change the object



Pooling – Max Pooling

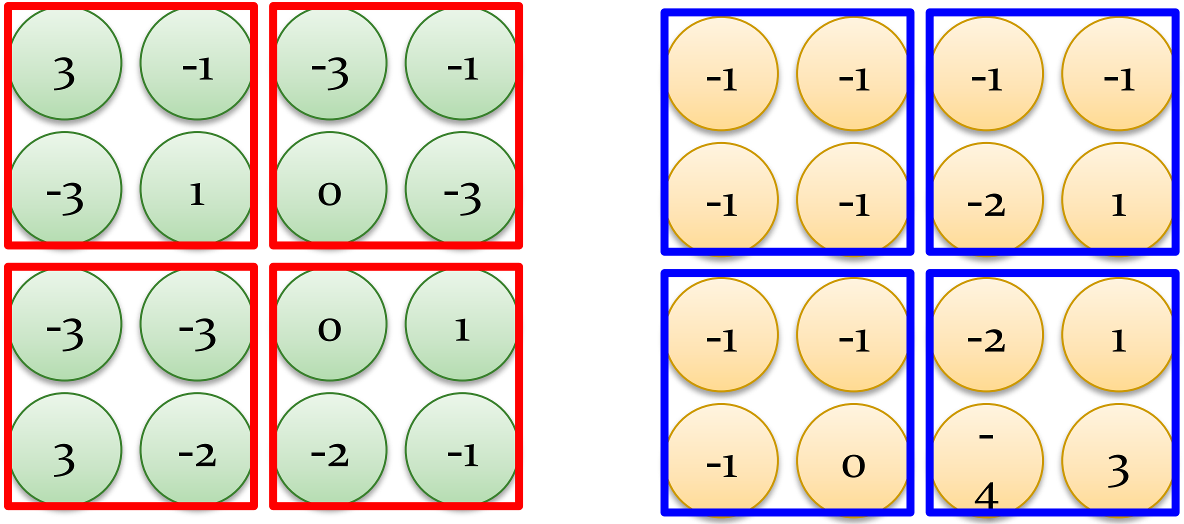
No learnable parameters!

1	-1	-1
-1	1	-1
-1	-1	1

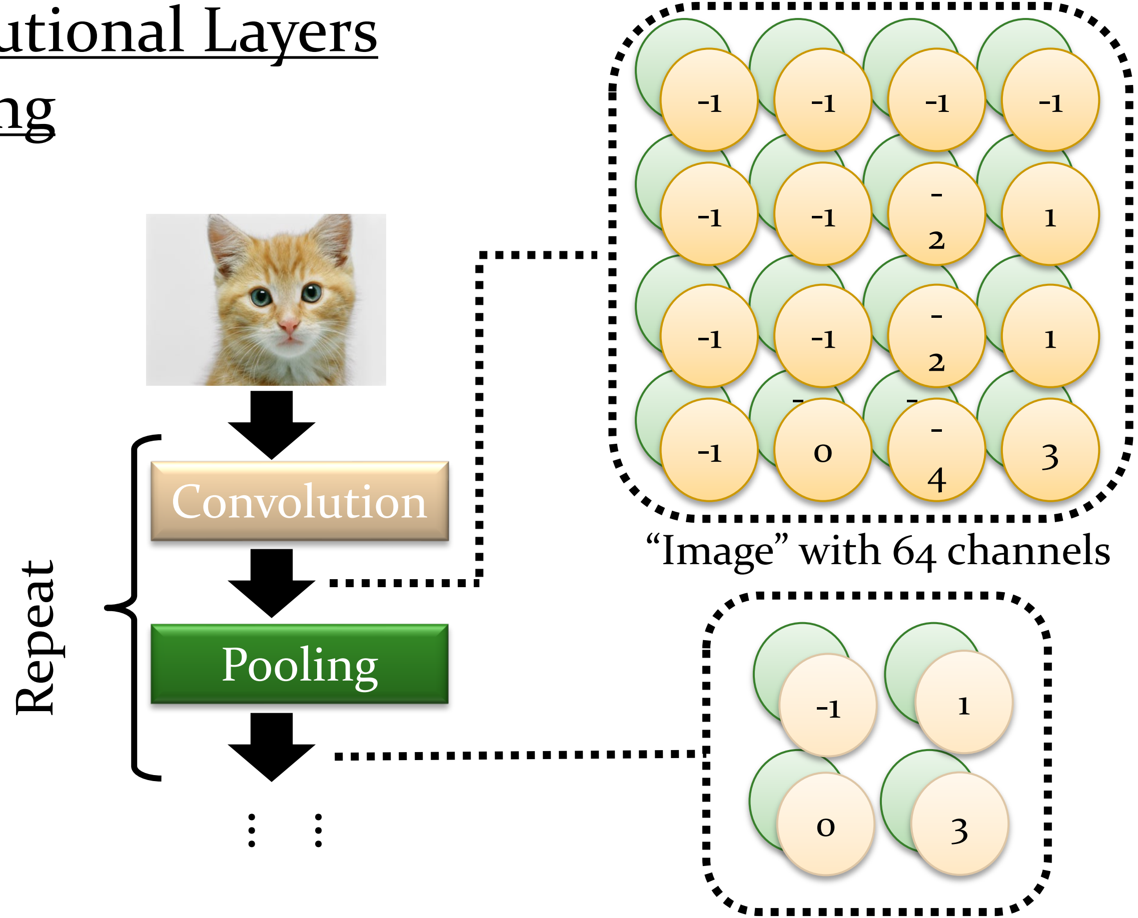
Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

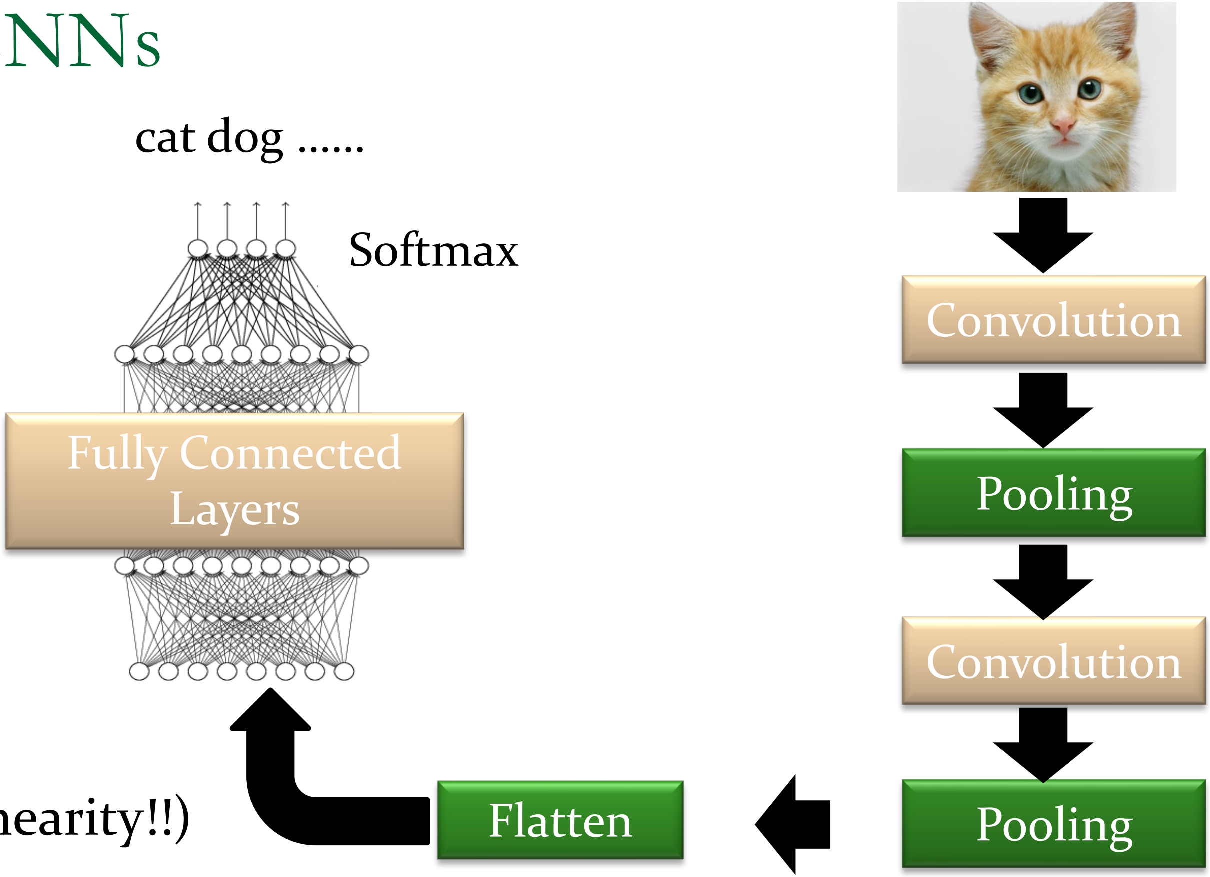
Filter 2



Convolutional Layers + Pooling

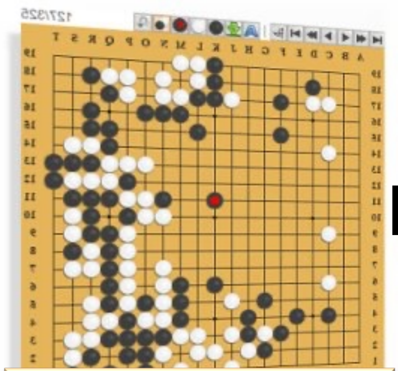


The whole CNNs



(they still need to
go through non-linearity!!)

Application: Playing Go



19 × 19 matrix
(image)

Network

Next move
(19 × 19 positions)
19 × 19 classes

48 channels in
Alpha Go

Black: 1
white: −1
none: 0

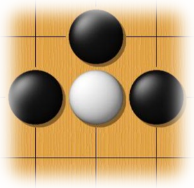
Fully-connected
network can be used

But CNN performs much
better.

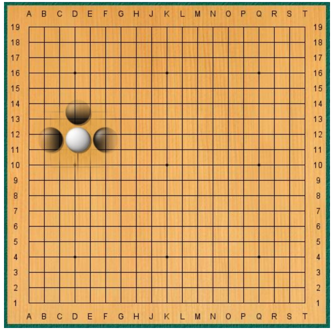
Why CNN for Go playing?

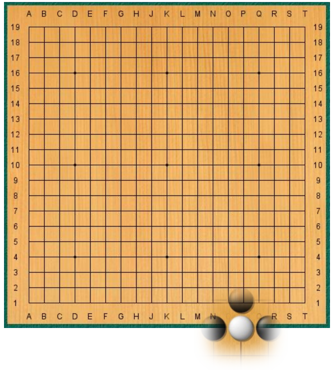
- Some patterns are much smaller than the whole image

Alpha Go uses 5 × 5 for first layer



- The same patterns appear in different regions.





```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

```python
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

```python
for epoch in range(2):  # 多批次循环

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # 获取输入
        inputs, labels = data

        # 梯度置0
        optimizer.zero_grad()

        # 正向传播，反向传播，优化
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

To learn more …

- CNN is not invariant to scaling and rotation (we need data augmentation ☺).