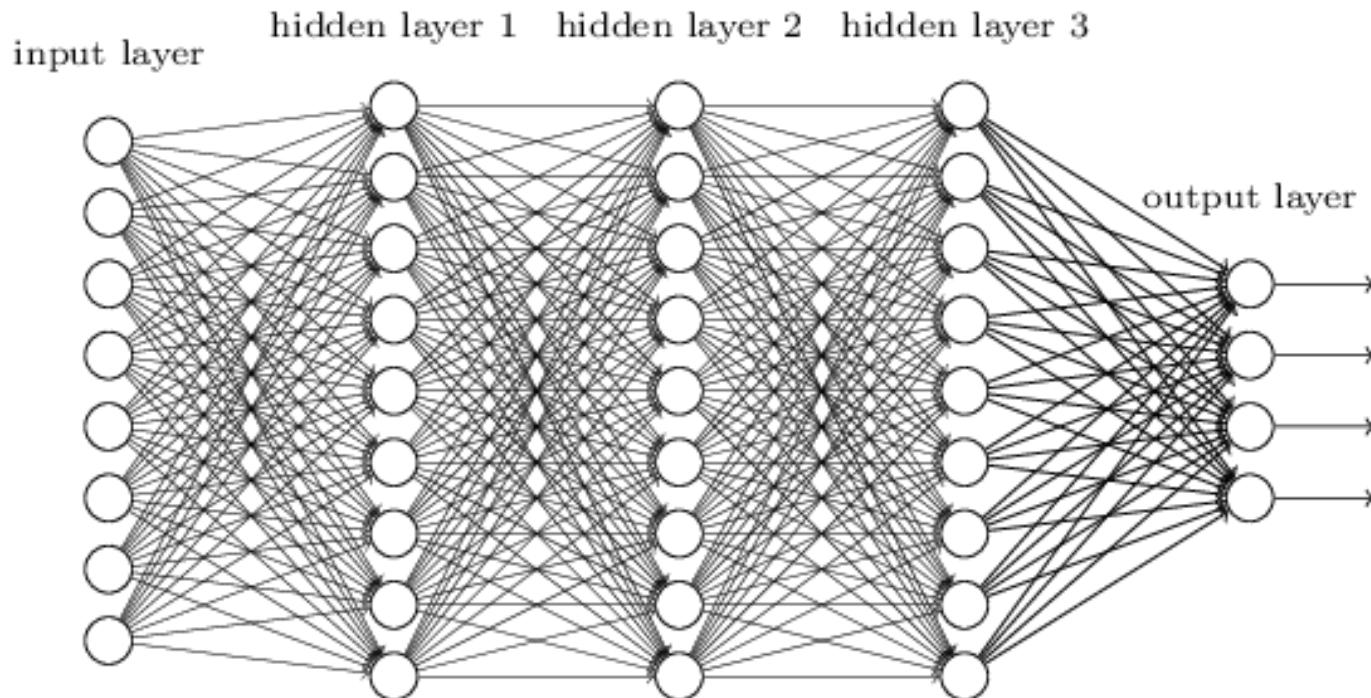


# CNN : Convolutional Neural Networks

- We know it is good to learn a small model.
- From this fully connected model, do we really need all the edges?
- Can some of these be shared?



# Consider learning an image:

- Some patterns are much smaller than the whole image

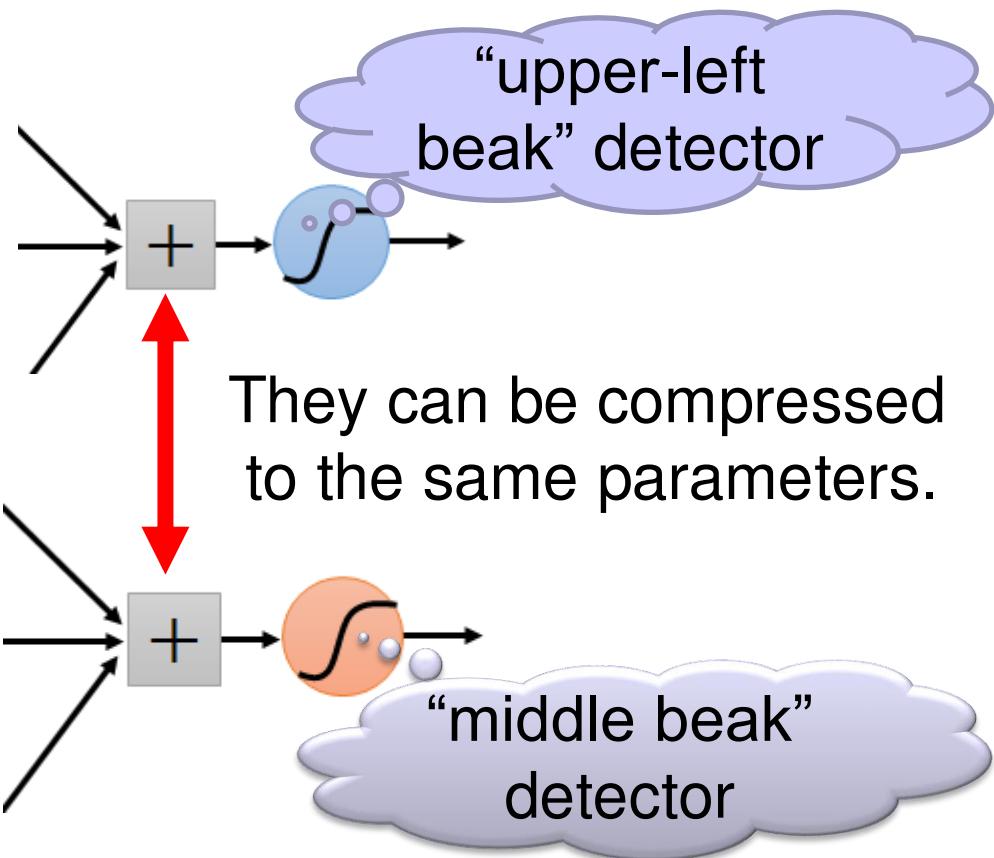
Can represent a small region with fewer parameters



Same pattern appears in different places:

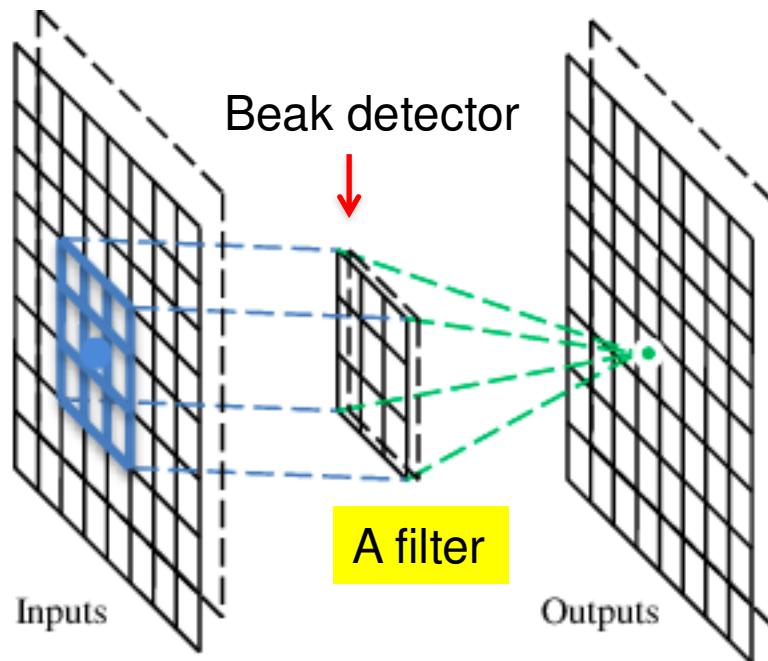
They can be compressed!

What about training a lot of such “small” detectors  
and each detector must “move around”.



# A convolutional layer

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



# Convolution

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

These are the network parameters to be learned.

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

Filter 1

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

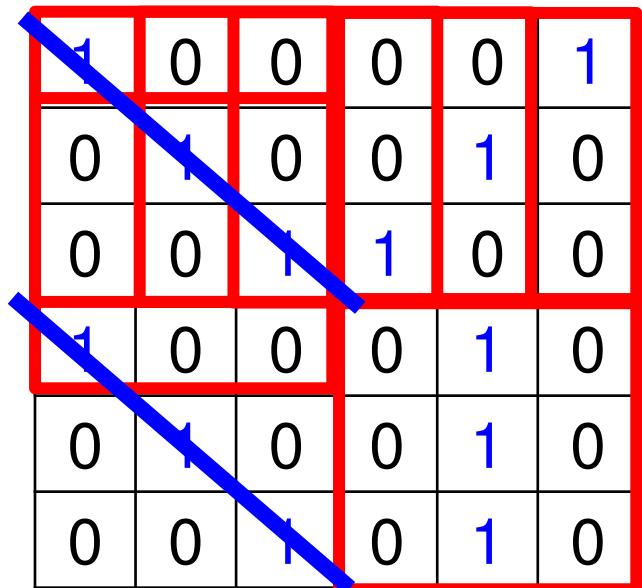
Filter 2

⋮ ⋮

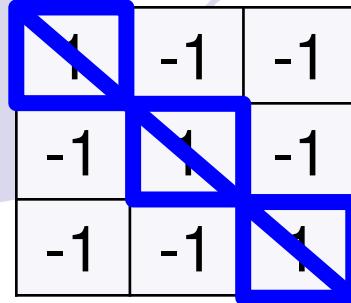
Each filter detects a small pattern (3 x 3).

# Convolution

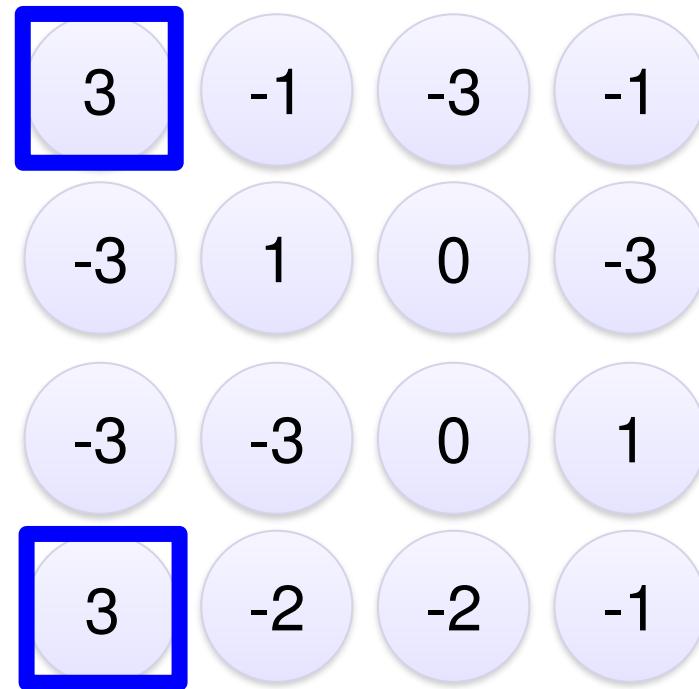
stride=1



6 x 6 image



Filter 1



# Convolution

stride=1

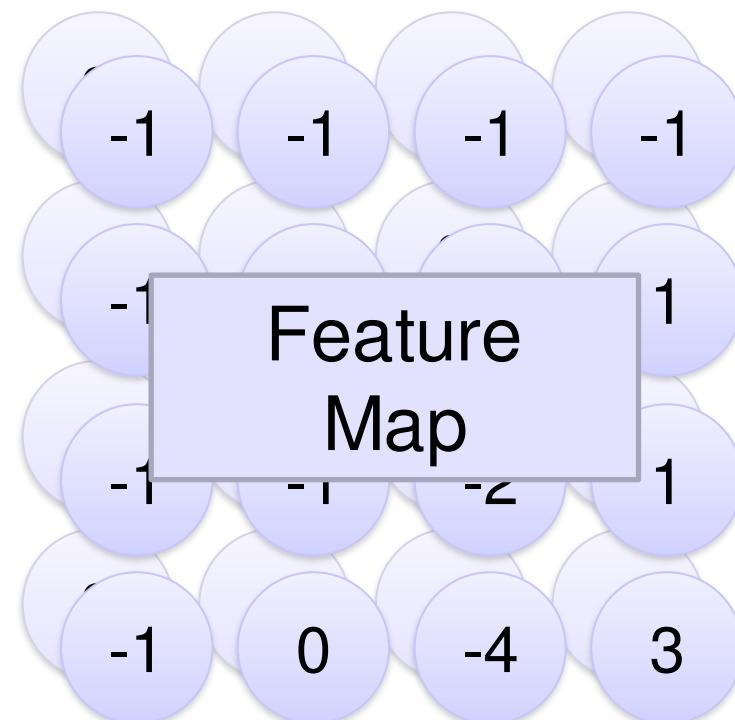
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

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

Filter 2

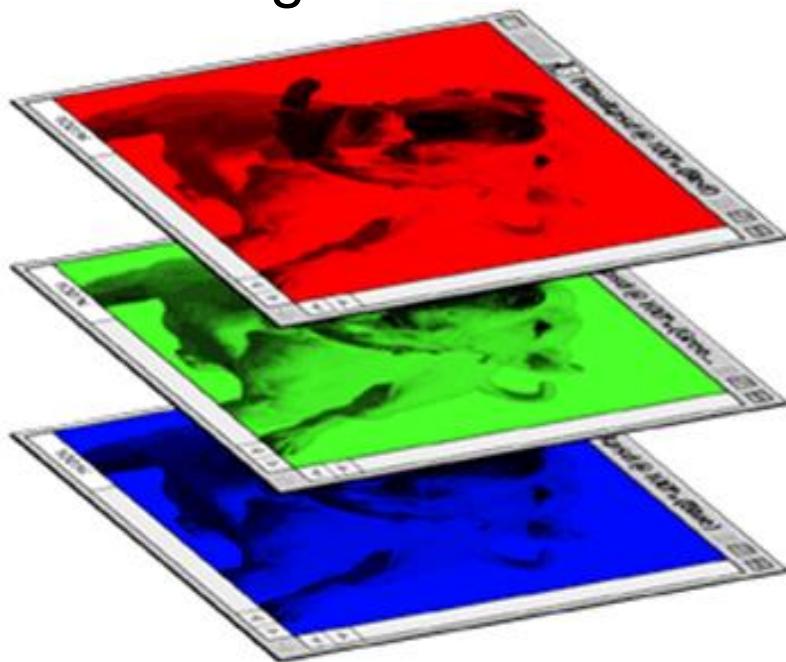
Repeat this for each filter



Two 4 x 4 images  
Forming 2 x 4 x 4 matrix

# Color image: RGB 3 channels

Color image



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

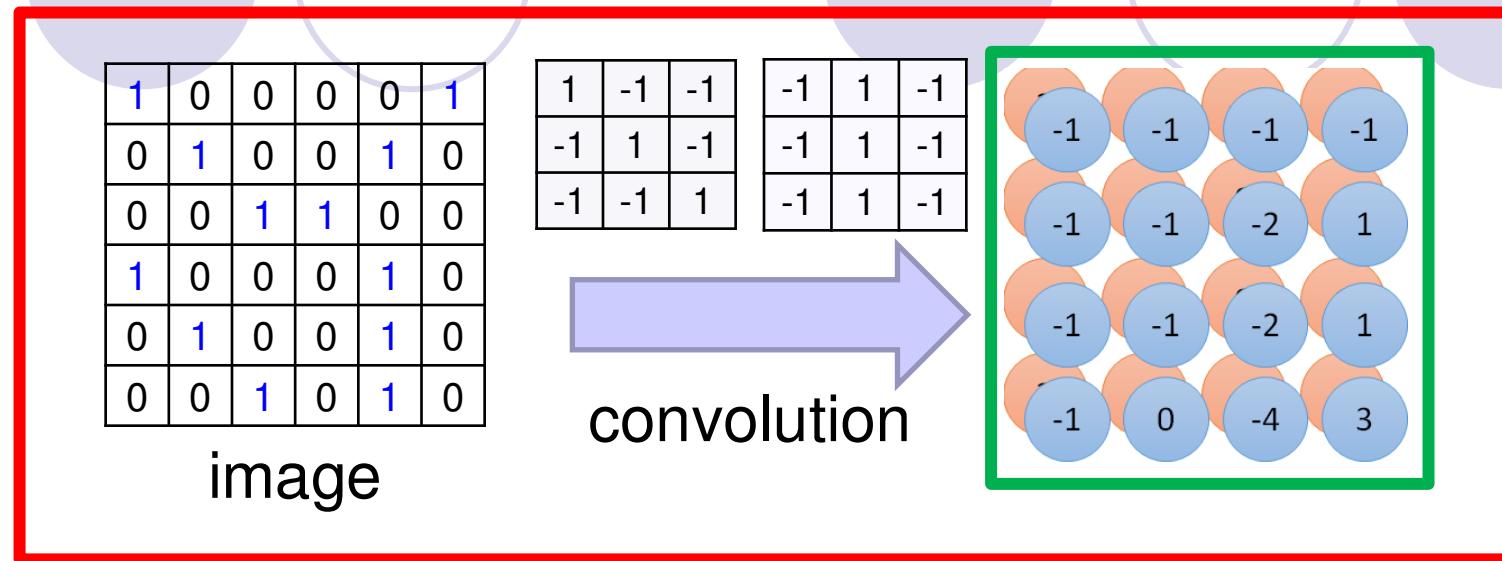
Filter 1

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

Filter 2

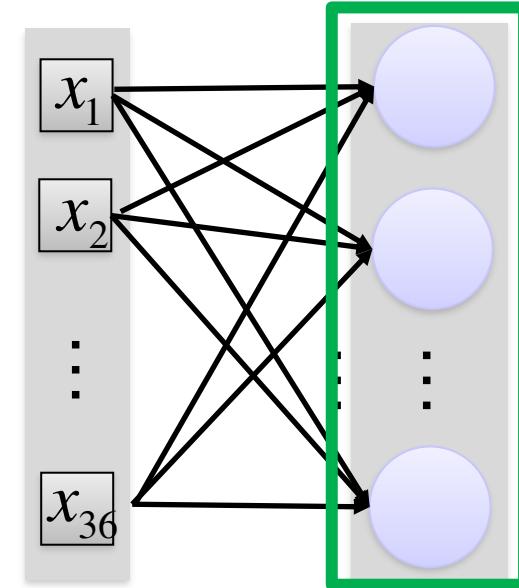
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

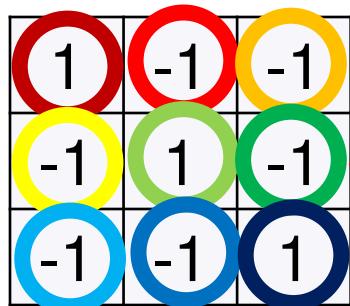
# Convolution v.s. Fully Connected



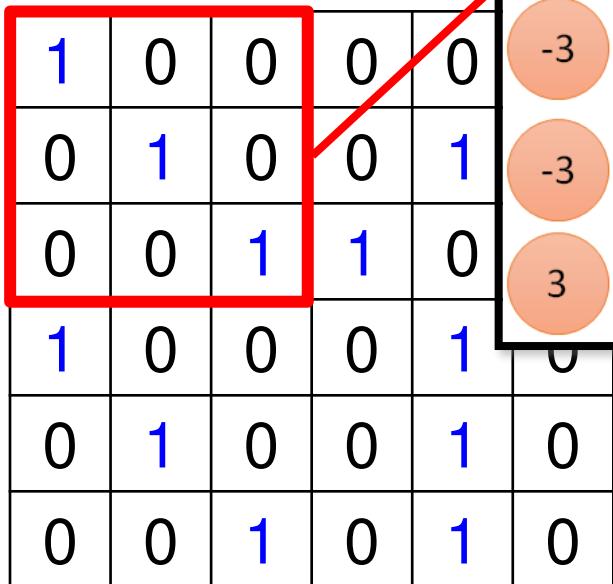
Fully-  
connected

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



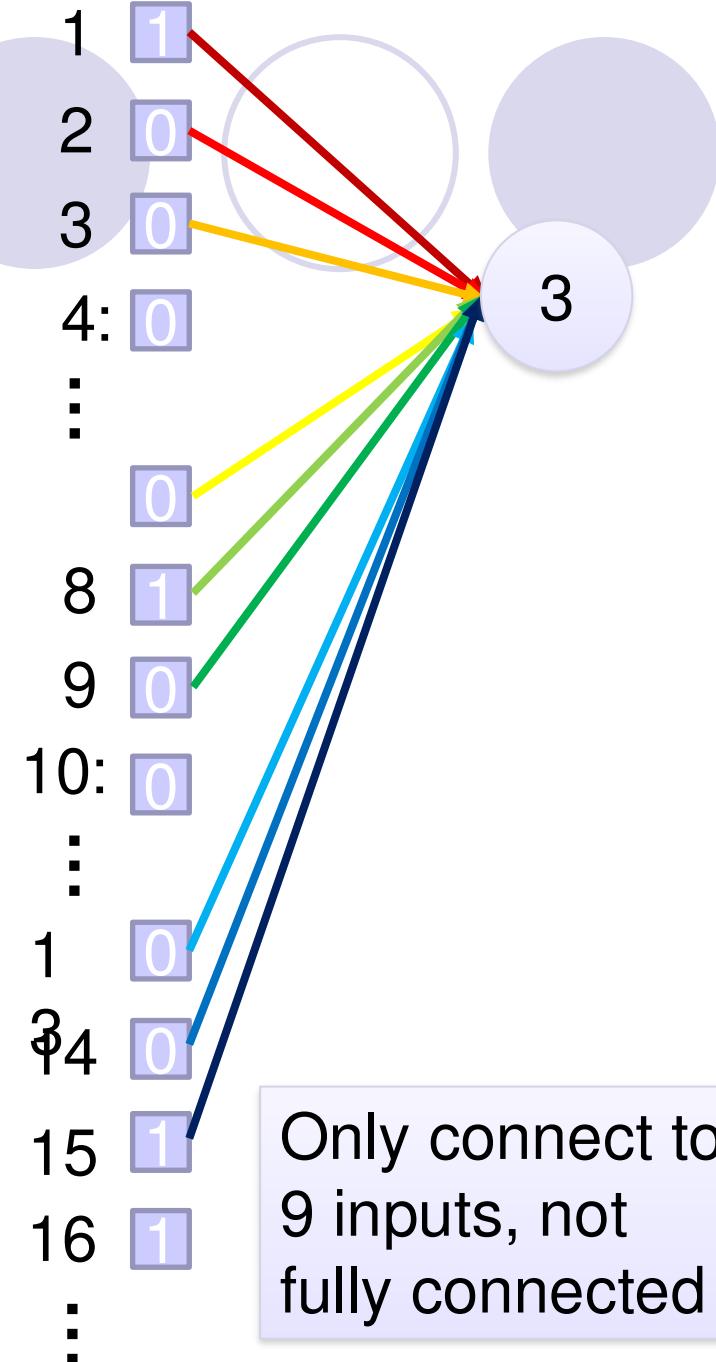
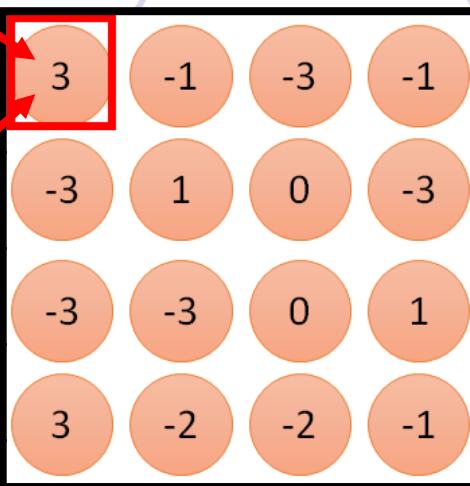


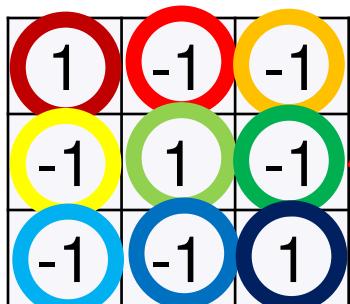
Filter 1



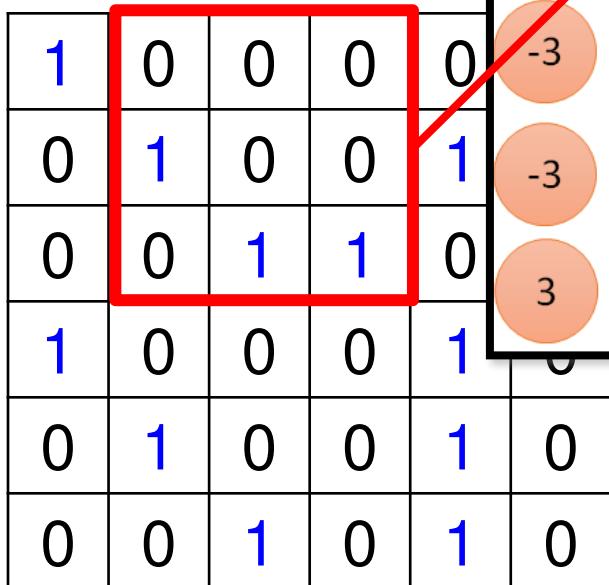
6 x 6 image

fewer parameters!





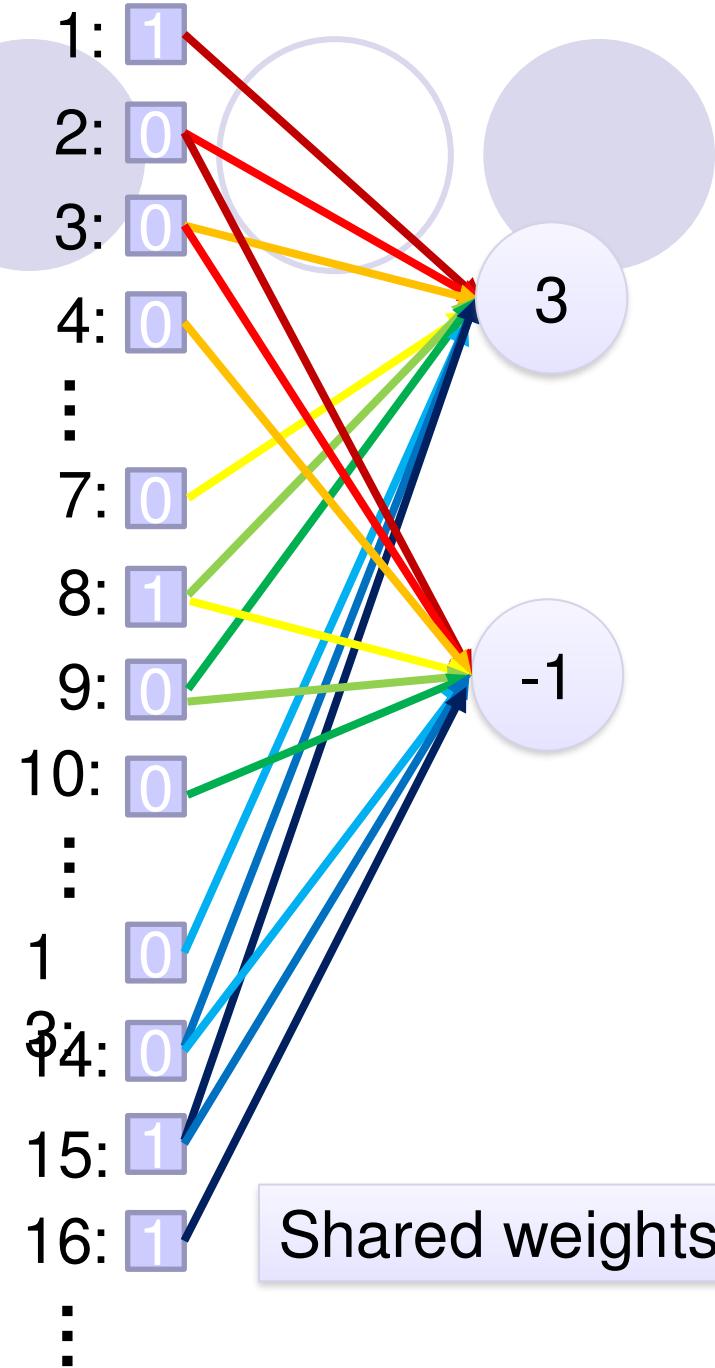
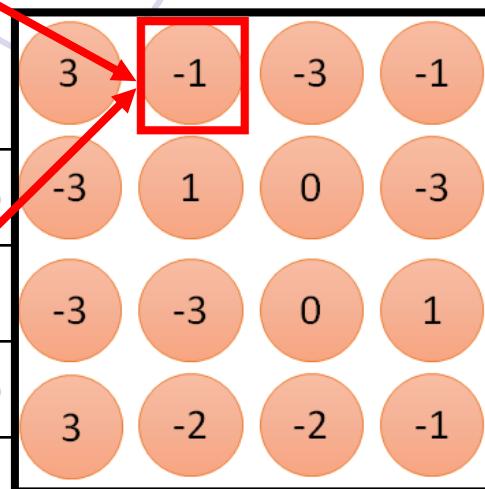
Filter 1



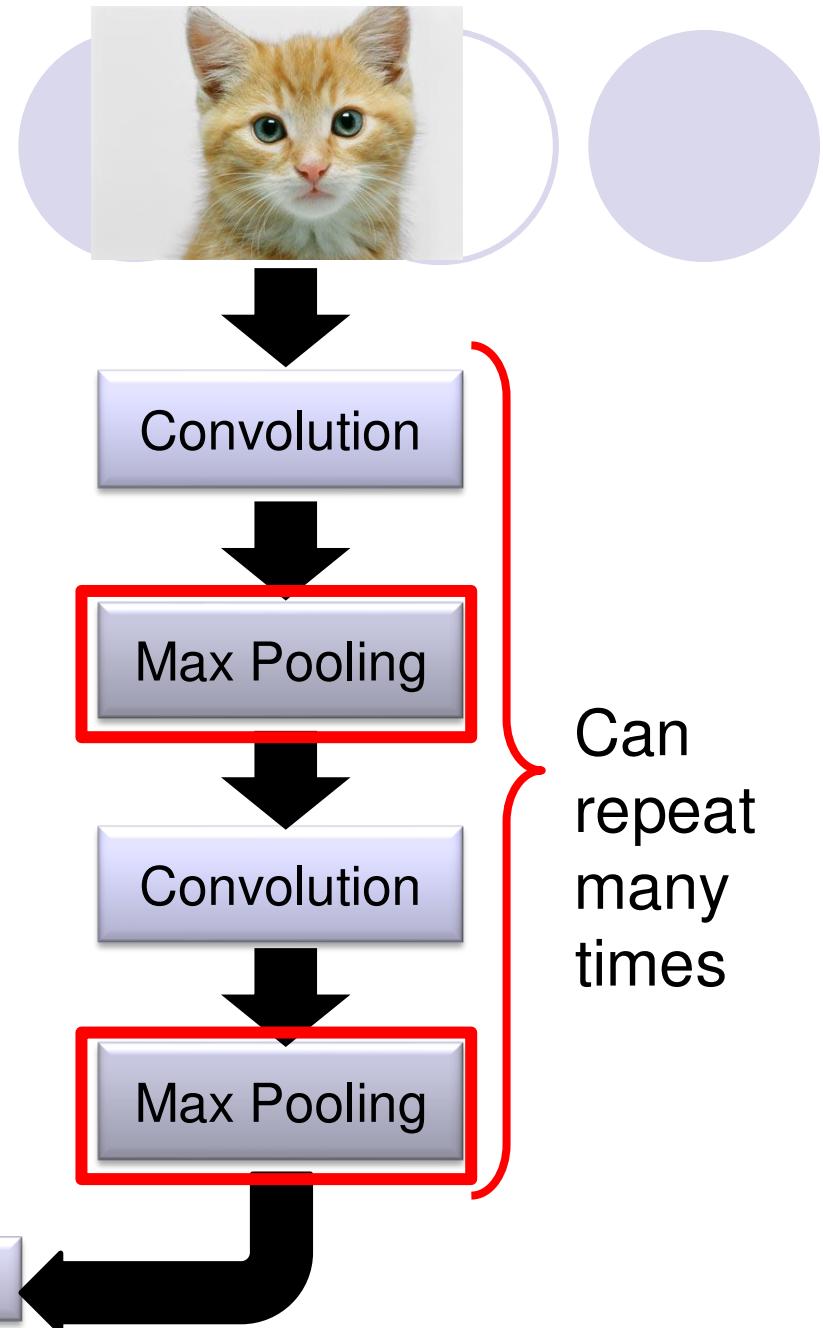
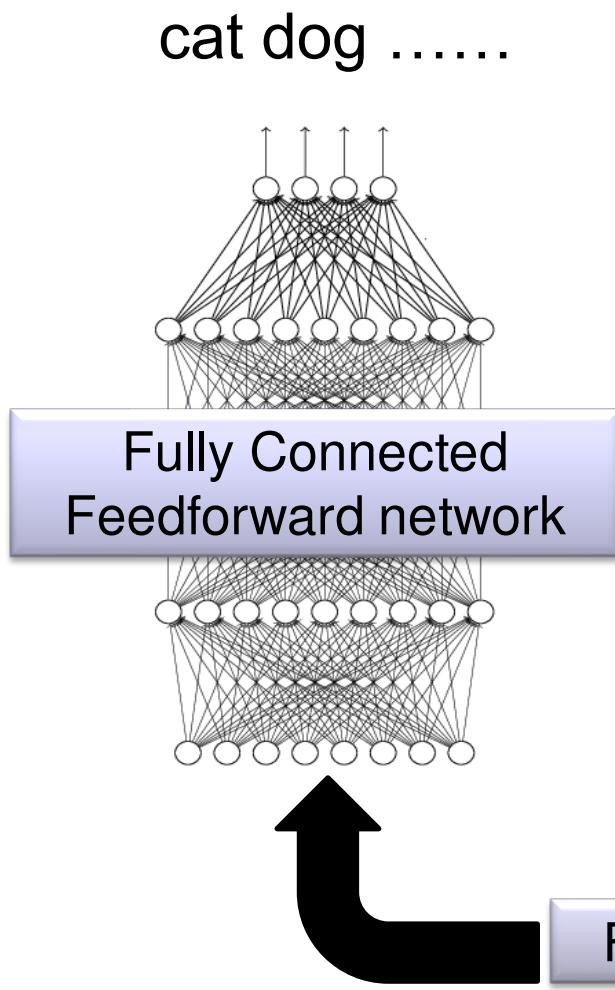
6 x 6 image

Fewer parameters

Even fewer parameters



# The whole CNN



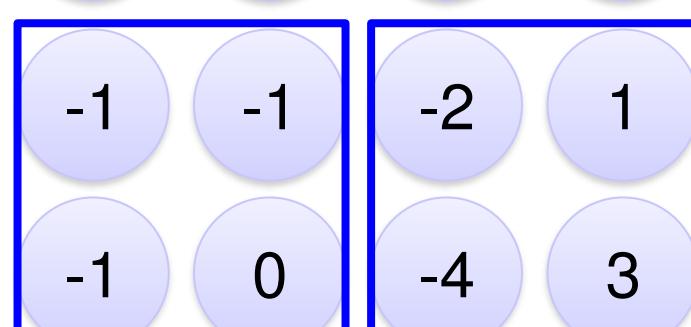
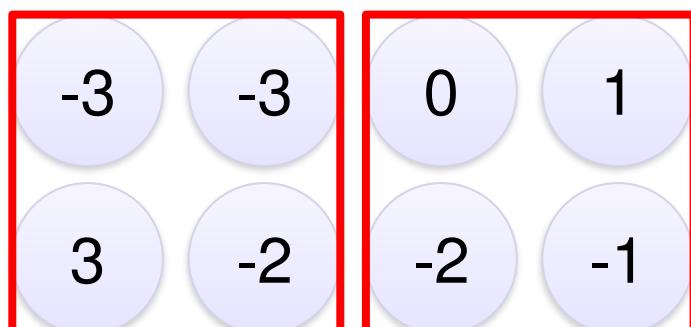
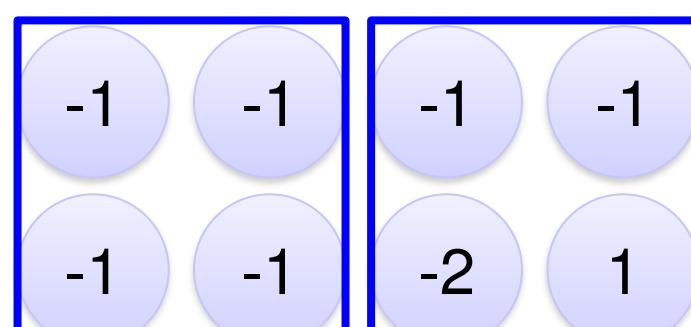
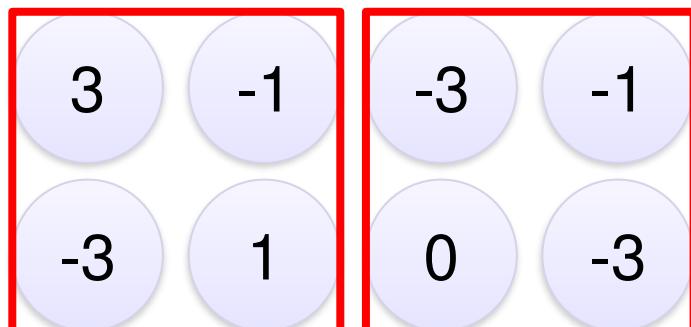
# Max Pooling

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

Filter 1

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

Filter 2



# Why Pooling

- Subsampling pixels will not change the object bird



Subsampling



bird

We can subsample the pixels to make image  
smaller  
→ fewer parameters to characterize the image

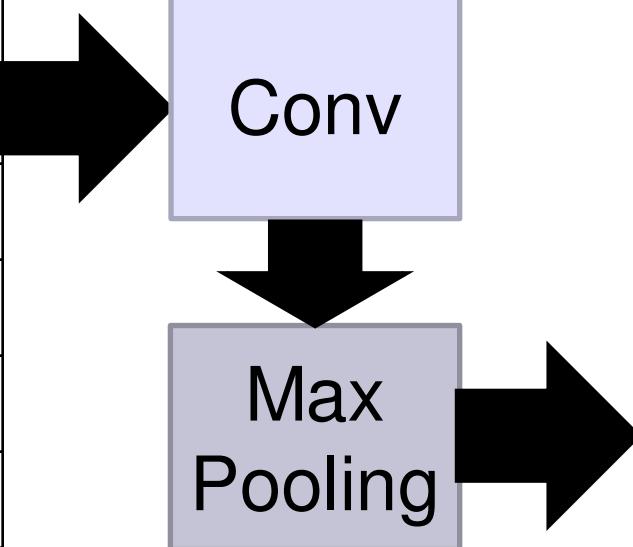
A CNN compresses a fully connected network in two ways:

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

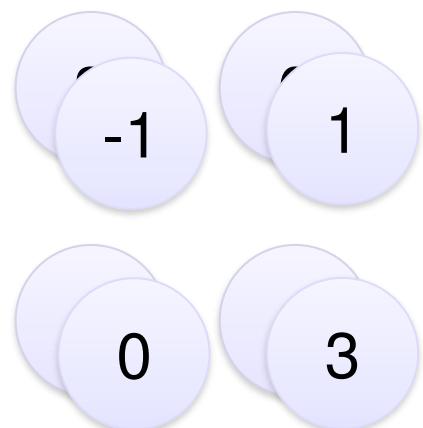
# Max Pooling

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



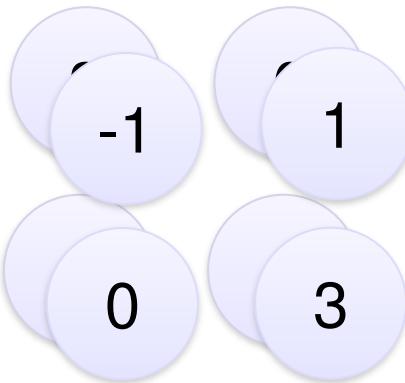
New image  
but smaller



2 x 2 image

Each filter  
is a channel

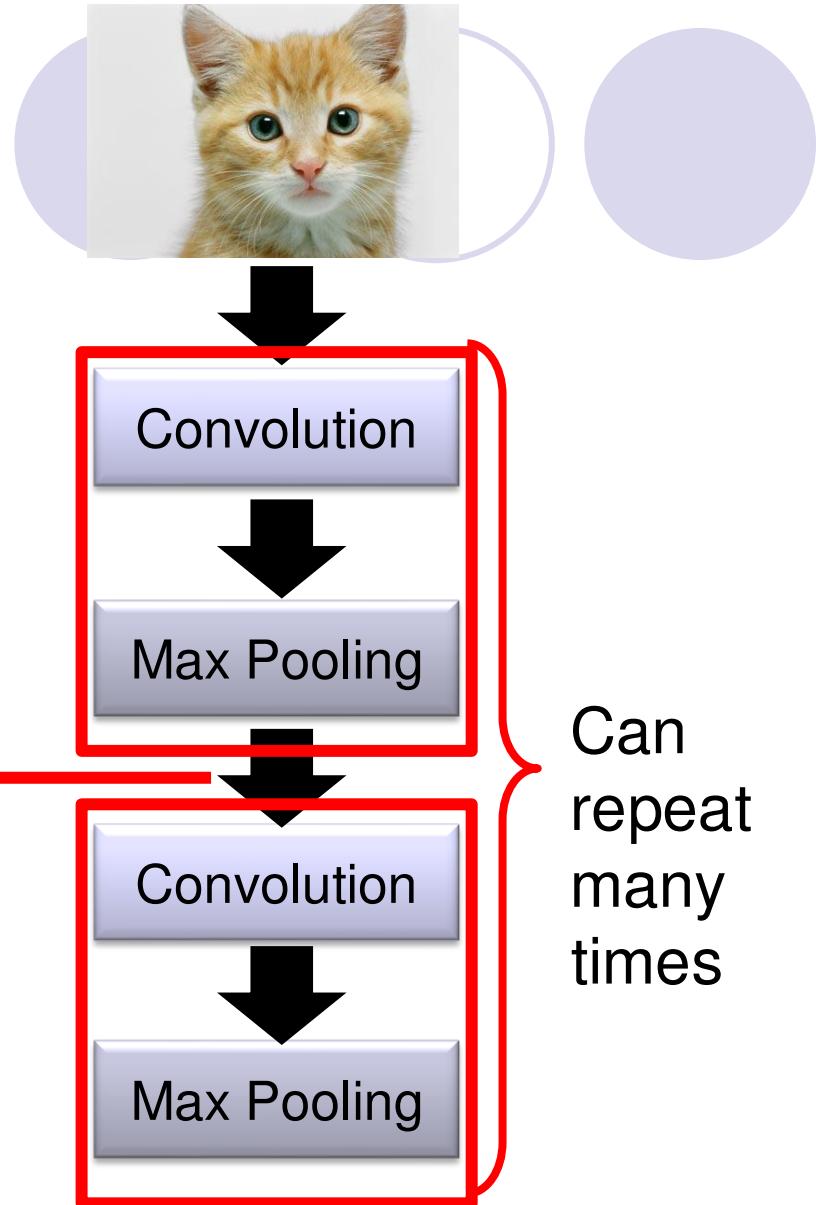
# The whole CNN



A new image

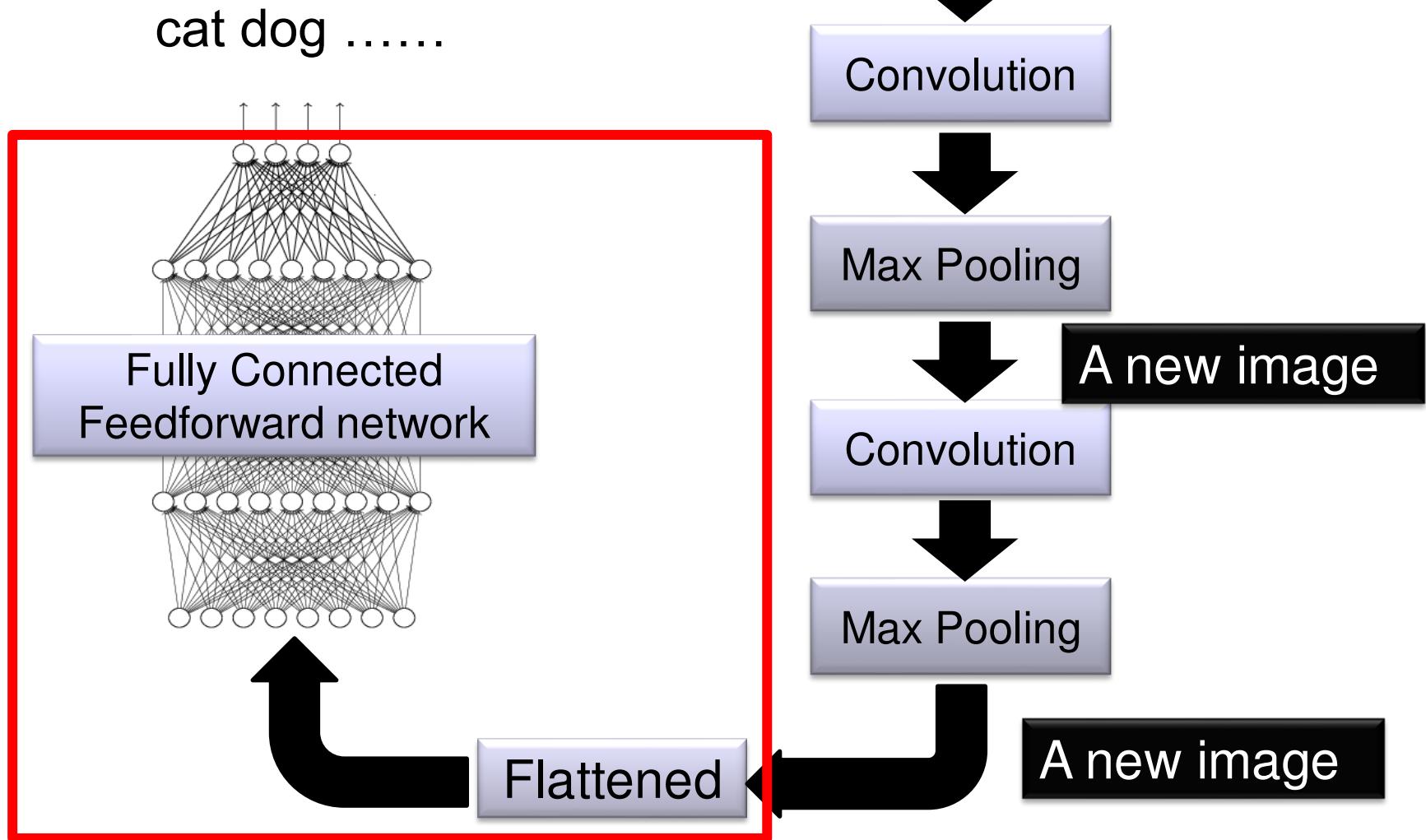
Smaller than the original image

The number of channels  
is the number of filters

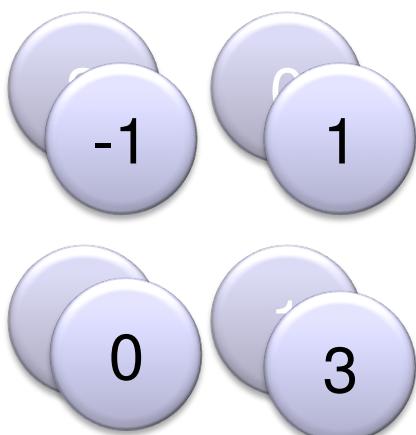


Can  
repeat  
many  
times

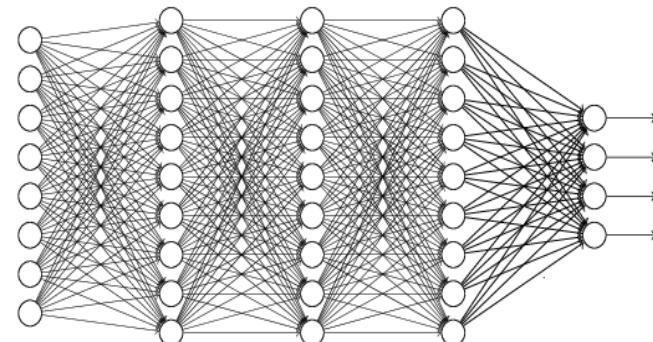
# The whole CNN



# Flattening



Flattened



Fully Connected  
Feedforward network

# CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
model2.add( Convolution2D( 25, 3, 3,  
                           input_shape=(28, 28, 1)) )
```

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

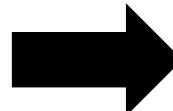
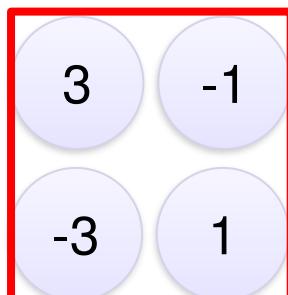
There are  
**25 3x3**  
filters.

Input\_shape = ( 28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

```
model2.add(MaxPooling2D( (2, 2) ))
```



input  
↓

Convolution  
↓

Max Pooling  
↓

Convolution  
↓

Max Pooling

# CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***

How many parameters for each filter?

```
model2.add( Convolution2D( 25, 3, 3,  
    input_shape=(28,28,1)) )
```

9

25 x 26 x 26

```
model2.add(MaxPooling2D( (2,2) ))
```

25 x 13 x 13

```
model2.add(Convolution2D(50, 3, 3))
```

$225 =$   
 $25 \times 9$

50 x 11 x 11

```
model2.add(MaxPooling2D( (2,2) ))
```

50 x 5 x 5

Input

Convolution

Max Pooling

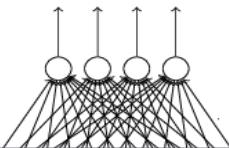
Convolution

Max Pooling

# CNN in Keras

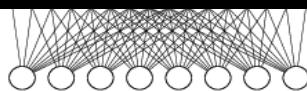
Only modified the ***network structure*** and ***input format (vector -> 3-D array)***

Output



Fully connected  
feedforward network

```
model2.add(Dense(output_dim=100))  
model2.add(Activation('relu'))  
model2.add(Dense(output_dim=10))  
model2.add(Activation('softmax'))
```



1250

Flattened

```
model2.add(Flatten())
```

Input

$1 \times 28 \times 28$

Convolution

$25 \times 26 \times 26$

Max Pooling

$25 \times 13 \times 13$

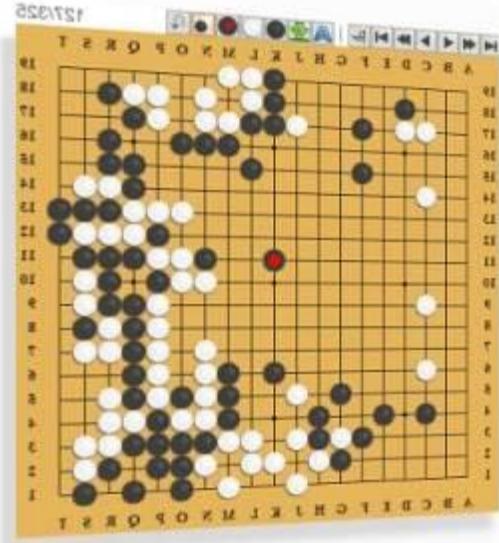
Convolution

$50 \times 11 \times 11$

Max Pooling

$50 \times 5 \times 5$

# AlphaGo

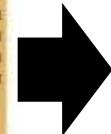


19 x 19 matrix

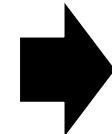
Black: 1

white: -1

none: 0



Neural  
Network



Next move  
(19 x 19  
positions)

Fully-connected feedforward  
network can be used

But CNN performs much better

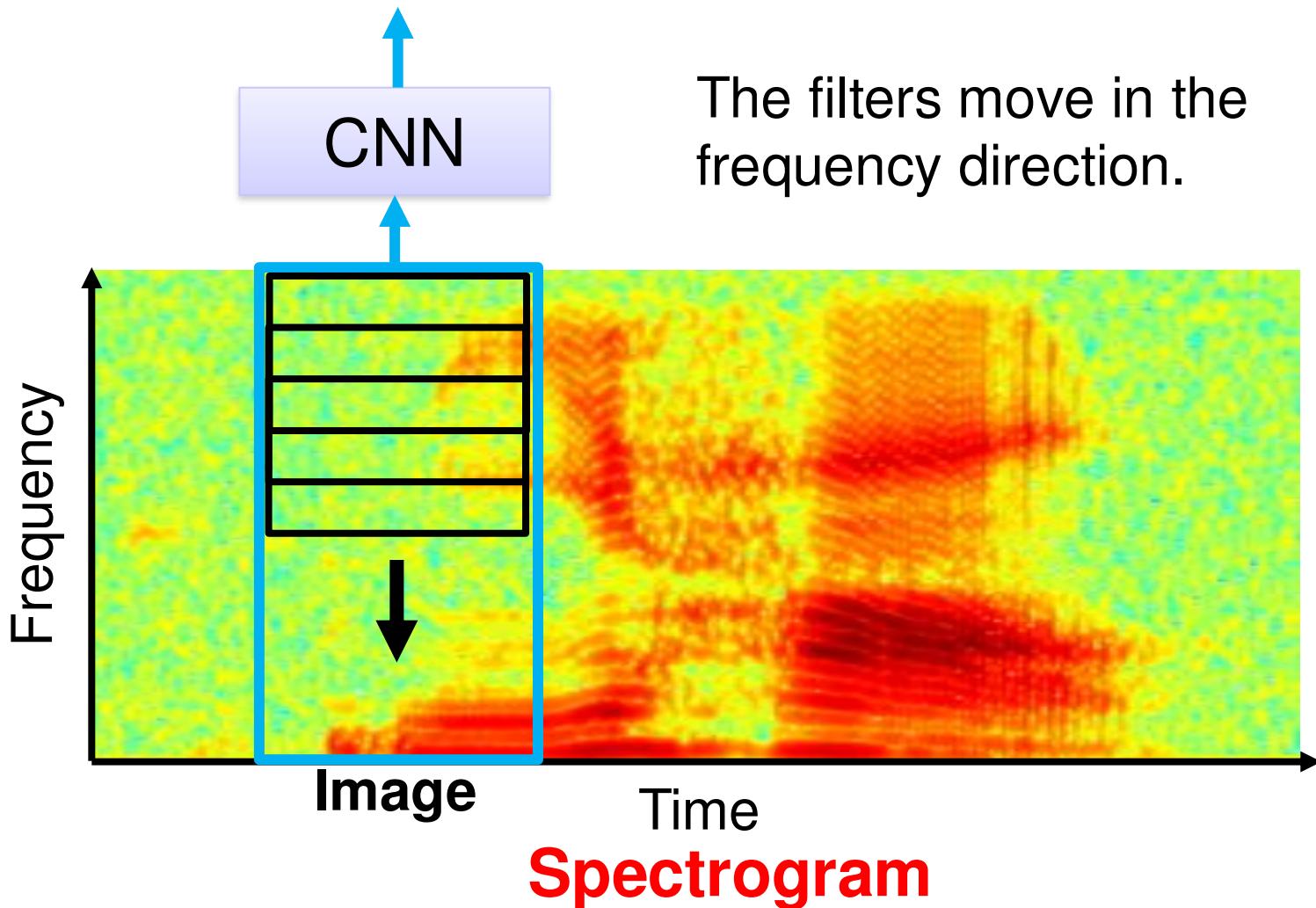
# AlphaGo's policy network

The following is quotation from their Nature article:

Note: AlphaGo does not use Max Pooling.

**Neural network architecture.** The input to the policy network is a  $19 \times 19 \times 48$  image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a  $23 \times 23$  image, then convolves  $k$  filters of kernel size  $5 \times 5$  with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a  $21 \times 21$  image, then convolves  $k$  filters of kernel size  $3 \times 3$  with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size  $1 \times 1$  with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used  $k = 192$  filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with  $k = 128, 256$  and  $384$  filters.

# CNN in speech recognition



# CNN in text classification

