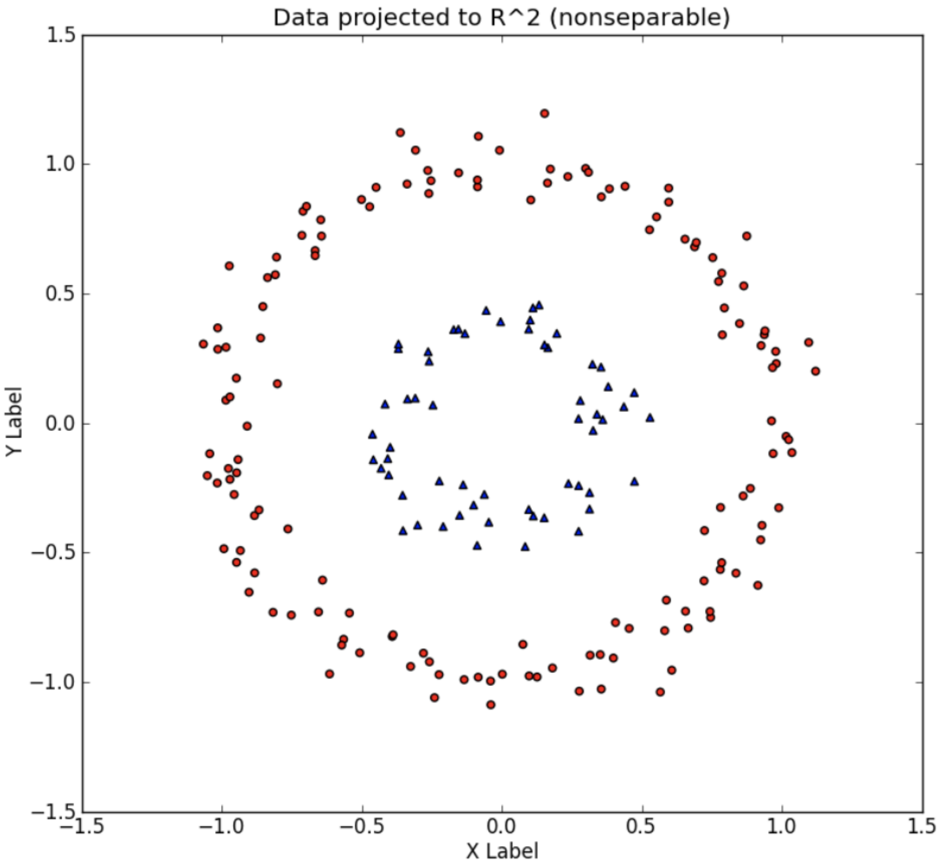
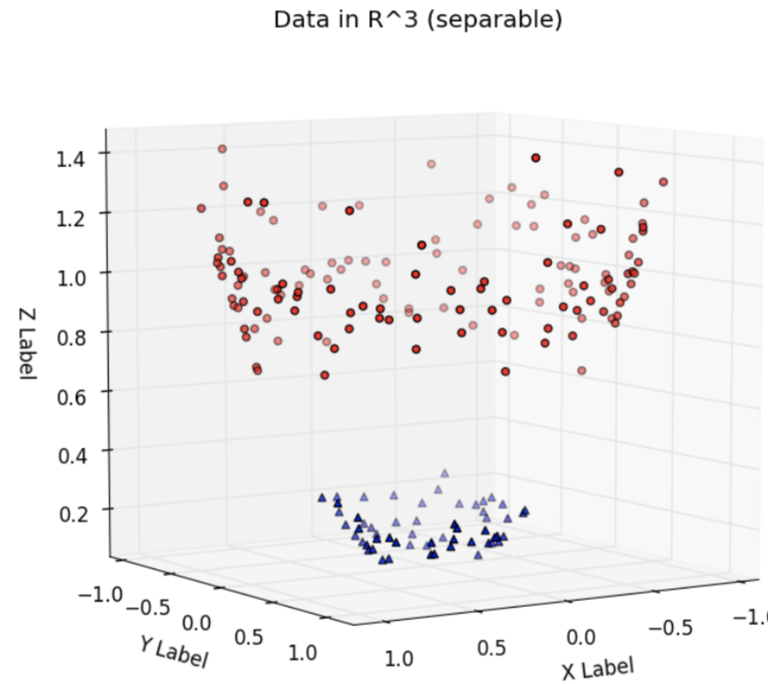
Kernel Methods


Data projected to R^2 (nonseparable)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Y Label
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
X Label


Data in R^3 (separable)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
Z Label
−1.0
−0.5
0.0
0.5
1.0
Y Label
1.0
0.5
0.0
−0.5
−1.0
X Label

- A basic concept in ML is the dot product. You often do dot products of the features of a data sample with some weigh ts , the parameters of your model. Instead of doing explicit ly this projection of the data in 3D and then evaluating the dot product, you can find a **kernel function** that simplifies this job by simply doing the dot product in the projected s pace for you, without the need to actually compute projec tions and then the dot product. This allows you to find a c omplex non linear boundary that is able to separate the cl asses in the dataset. This is a very intuitive explanation.

Feature Mapping

- **Example.** Non-linear classifiers.

$$x = (x_1, x_2)$$

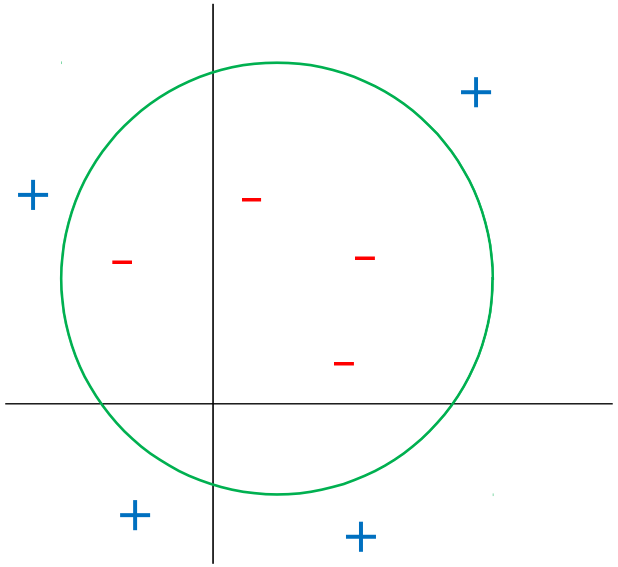
$$\phi(x) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}\, x_1 x_2, x_1^2, x_2^2\right)$$

$$\begin{aligned} h(x; \theta, \theta_0) & \\ &= \operatorname{sign}\left(\theta \cdot \phi(x)\right) \\ &= \operatorname{sign}\left(\theta_1 + \theta_2\sqrt{2}x_1 + \theta_3\sqrt{2}x_2 + \theta_4\sqrt{2}x_1 x_2 + \theta_5 x_1^2 + \theta_6 x_2^2\right) \end{aligned}$$

Feature Mapping

- **Example.** Non-linear classifiers.

$$h(x;\theta,\theta_0)$$
$$= \operatorname{sign}\big((x_1-1)^2+(x_2-2)^2-9\big)$$
$$= \operatorname{sign}(-4-2x_1-4x_2+x_1^2+x_2^2)$$



Challenges

- **High-Dimensional Features.**

$$x = (x_1, x_2, \dots, x_{1000}) \in \mathbb{R}^{1000}$$

$$\phi(x) = \left(1, \dots, \sqrt{2}x_i, \dots, \sqrt{2}x_i x_j, \dots, x_i^2, \dots\right) \in \mathbb{R}^{501501}$$

Complexity

Inner product: $\boldsymbol{O}(\boldsymbol{d})$

Computing $\phi(x) \cdot \phi(x')$ for $x, x' \in \mathbb{R}^{1000}$ requires about 2,004,000 floating point operations.

Kernel Functions

- Fortunately, many inner products simplify nicely.

$$
\begin{aligned}
K(x, x') = \phi(x) \cdot \phi(x') \\
&= 1 + 2\sum_i x_i x_i' + 2\sum_{i<j} x_i x_j x_i' x_j' + \sum_i x_i^2 x_i'^2 \\
&= 1 + 2\left(\sum_i x_i x_i'\right) + \left(\sum_i x_i x_i'\right)^2 \\
&= (x \cdot x' + 1)^2
\end{aligned}
$$

For $x, x' \in \mathbb{R}^{1000}$, computing this requires only about 2000 floating point operations, less than the 501,501 operations needed for $\phi(x)$.

Kernel function as similarity measure between input objects

Kernel Definition

- Theorem (Mercer): A function $K: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is a *kernel function* if and only if

1. it is symmetric: $K(x, y) = K(y, x)$ for all $x, y \in \mathbb{R}^d$,
2. Given $n \in \mathbb{N}$ and $x^{(1)}, x^{(2)}, \dots, x^{(n)} \in \mathbb{R}^d$,

   the *Gram matrix* K with entries

   $$K_{ij} = K\left(x^{(i)}, x^{(j)}\right)$$

   is positive semidefinite (positive eigenvalues)..

Kernel Properties

- Kernels can be constructed manually.
- Scalar product $\langle x, x' \rangle$ is a kernel
- Constant $K(x, x') \equiv 1$ is a kernel
- Product of kernels $K(x, x') = K_1(x, x')K_2(x, x')$is a kernel
- For every function $\varphi: X \to R$, the product $K(x, x') = \varphi(x)\varphi(x')$ is a kernel
- Linear combination of kernels
- $K(x, x') = \alpha_1 K_1(x, x') + \alpha_2 K_2(x, x')$ with positive coefficients is a kernel
- etc

Examples

- **Linear Kernel.**

$$K(x, x') = x \cdot x'$$

Polynomial Kernel.

$$K(x, x') = (x \cdot x' + 1)^k$$

Feature map $\phi(x)$ is infinite dimensional!

Radial Basis Kernel (Gaussian Kernel).

$$K(x, x') = \exp\left(-\frac{1}{2\sigma^2}\|x - x'\|^2\right)$$

The Gaussian Kernel

Online material https://en.wikipedia.org/wiki/Radial_basis_function_kernel

- Defined as $K(x, x') = \exp\left(-\frac{1}{2\sigma^2}\|x - x'\|^2\right)$ for $\sigma > 0$
- The most widely used kernel
- Corresponding feature space is infinite dimensional
- Very strong expression power
- Be careful not to overfit!

Kernel Trick in SVM

Kernel Trick

- The kernel trick refers to the strategy of converting a learning algorithm and the resulting predictor into ones that involve only the computation of the kernel $K(x, x') = \phi(x) \cdot \phi(x')$ but not of the feature map $\phi(x)$.

Support Vector Machines

- **Learning.**

maximize $\sum_{(x,y)} \alpha_{x,y} - \frac{1}{2} \sum_{(x,y)} \sum_{(x',y')} \alpha_{x,y}\, \alpha_{x',y'}\, y y' (x \cdot x')$

subject to $\alpha_{x,y} \geq 0$ for all (x, y)

Prediction.

$$h(x;\theta) = \mathrm{sign}(\theta \cdot x) = \mathrm{sign}\left(\sum_{(x',y')} \alpha_{x',y'} y' (x \cdot x')\right)$$

- Only inner product of data points necessary, no coordinates
- Kernel function: $K(x, x') = \phi(x) \cdot \phi(x')$
  - ϕ not necessary any more
  - possible to operate in any n-dimensional feature space
  - complexity independent of feature space

Support Vector Machines

- **Learning.**

  maximize $\sum_{(x,y)} \alpha_{x,y} - \frac{1}{2} \sum_{(x,y)} \sum_{(x',y')} \alpha_{x,y} \alpha_{x',y'} y y' (x \cdot x')$

  subject to $\alpha_{x,y} \geq 0$ for all (x, y)

Prediction.

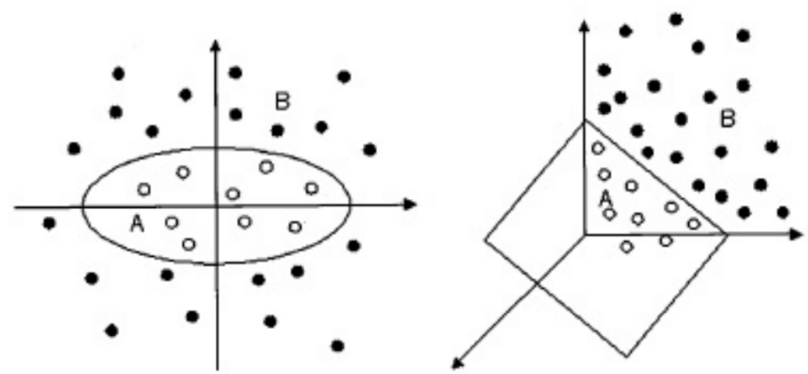
$$h(x;\theta) = \text{sign}(\theta \cdot x) = \text{sign}\left(\sum_{(x',y')} \alpha_{x',y'} y' (x \cdot x')\right)$$

- Data not linear separable in input space

  $\Rightarrow$ map into some feature space where data is linear separable

Mapping Example

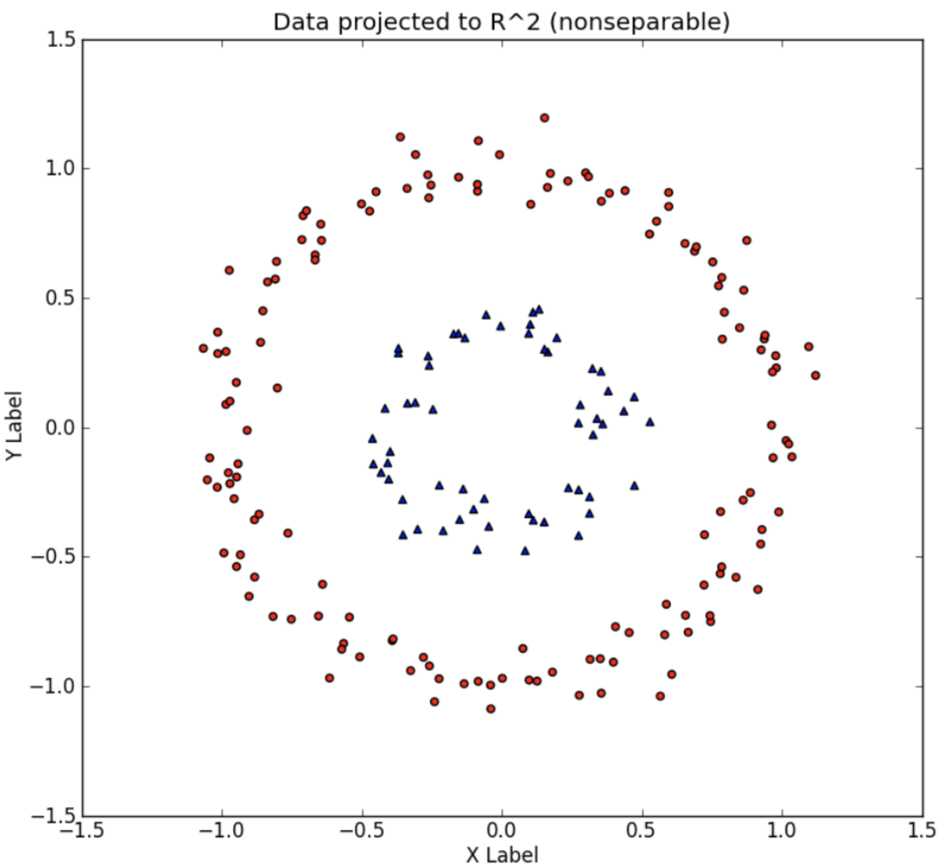
- Map data points into feature space with some function ϕ
- For example
  - ϕ: $R^2 \rightarrow R^3$
  - $(x_1, x_2) \rightarrow (z_1, z_2, z_3) := (x_1^2,$

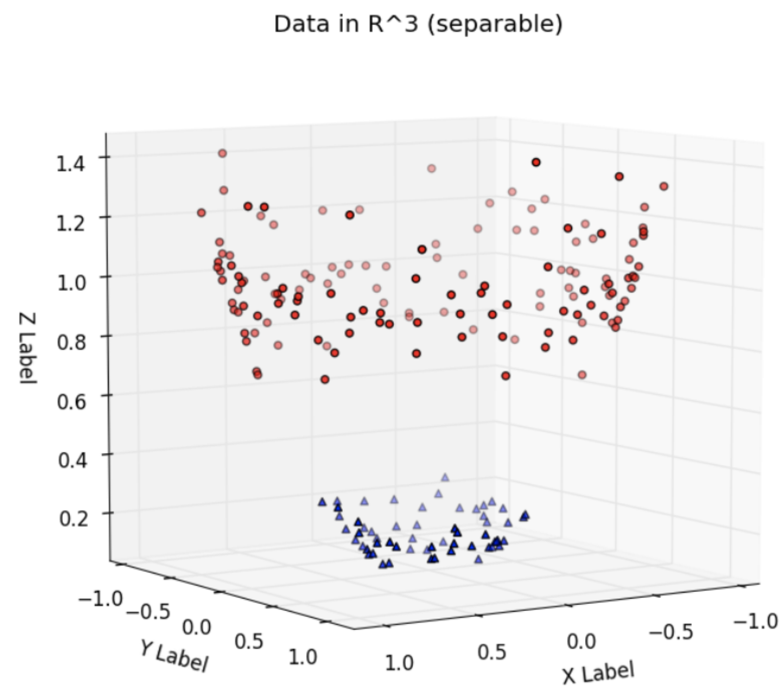


  - Hyperplane $\langle w \cdot z \rangle = 0$ can be written as a function of x:

$$w_1 x_1^2 + w_2 \sqrt{2} x_1 x_2 + w_3 x_2^2 = 0$$

Another Mapping Example





Kernel Support Vector Machines

- **Learning.**

$$\text{maximize } \sum_{(x,y)} \alpha_{x,y} - \frac{1}{2} \sum_{(x,y)} \sum_{(x',y')} \alpha_{x,y}\, \alpha_{x',y'}\, y y' K(x, x')$$

$$\text{subject to } \alpha_{x,y} \geq 0 \text{ for all } (x, y)$$

Prediction.

$$h(x;\theta) = \text{sign}(\theta \cdot \phi(x)) = \text{sign}\left(\sum_{(x',y')} \alpha_{x',y'}\, y' K(x, x')\right)$$

We may use the linear, polynomial, or radial basis kernels
to get different kinds of decision boundaries.

Perceptron

- **Learning.**

1. Initialize $\theta = 0$.
2. Repeat until no mistakes are found:
   Select data $(x, y) \in \mathcal{S}_n$ in sequence:
   If $y(\theta^\top x) \leq 0$, then $\theta \longleftarrow \theta + yx$.

Prediction.

$$h(x; \theta, \theta_0) = \operatorname{sign}(\theta \cdot x)$$

From the learning algorithm, we see that

$\theta = \sum_{x,y} \alpha_{x,y} yx$ for some $\alpha_{x,y} \in \mathbb{N}$.

Perceptron
online tutorial:
https://en.wikipedia.org/wiki/Kernel_perceptron

- **Learning.**

1. Initialize α to an all-zeros vector of length n, the number of training samples
2. Repeat until no mistakes are found:
   Select data $(x, y) \in \mathcal{S}_n$ in sequence:
   If $\sum_{x',y'} \alpha_{x',y'}\, yy'(x \cdot x') \leq 0$, then
   $\alpha_{x,y} \longleftarrow \alpha_{x,y} + 1.$

α is a mistake counter.

Prediction.

$$h(x; \theta, \theta_0) = \text{sign}\left(\sum_{x',y'} \alpha_{x',y'}\, y'(x \cdot x')\right)$$

Kernel Perceptron

- **Learning.**
  1. Initialize $\alpha = 0$.
  2. Repeat until no mistakes are found:
     Select data $(x, y) \in \mathcal{S}_n$ in sequence:
     If $\sum_{x',y'} \alpha_{x',y'}\, yy' K(x, x') \leq 0$, then
     $\alpha_{x,y} \longleftarrow \alpha_{x,y} + 1$.

Prediction.

$$h(x; \theta, \theta_0) = \operatorname{sign}\left(\sum_{x',y'} \alpha_{x',y'}\, y' K(x, x')\right)$$

Summary

- Kernel Functions
  - Feature Maps
  - Inner Products
  - Polynomial Kernel
  - Radial Basis Kernel
- Kernel Trick
  - Support Vector Machines
  - Perceptron

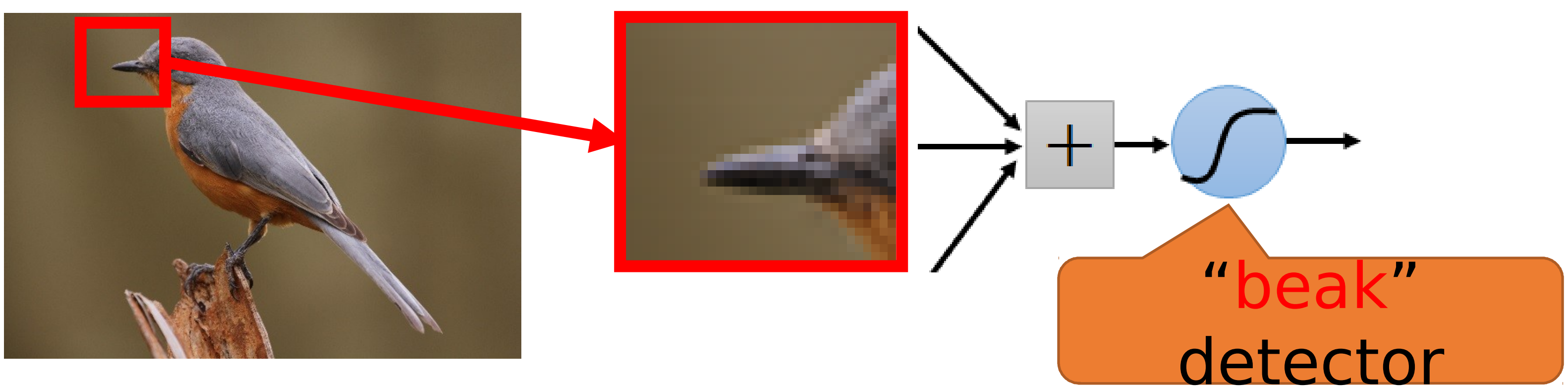
Convolutional Neural Networks

Online tutorial: http://cs231n.github.io/convolutional-networks/

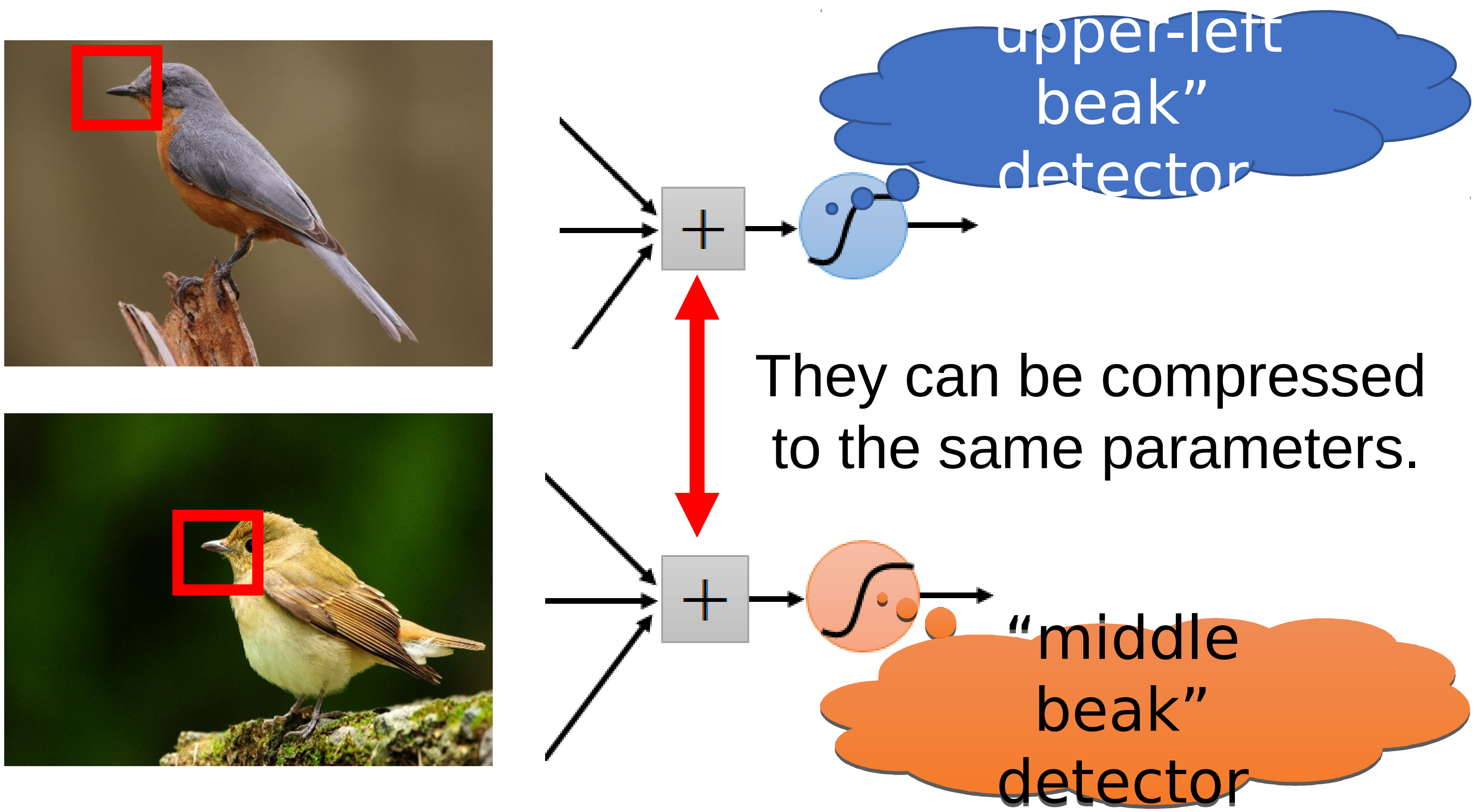
Consider learning an image:

- Some patterns are much smaller than the wh ole image

That’s why we use relatively small convolutional filters

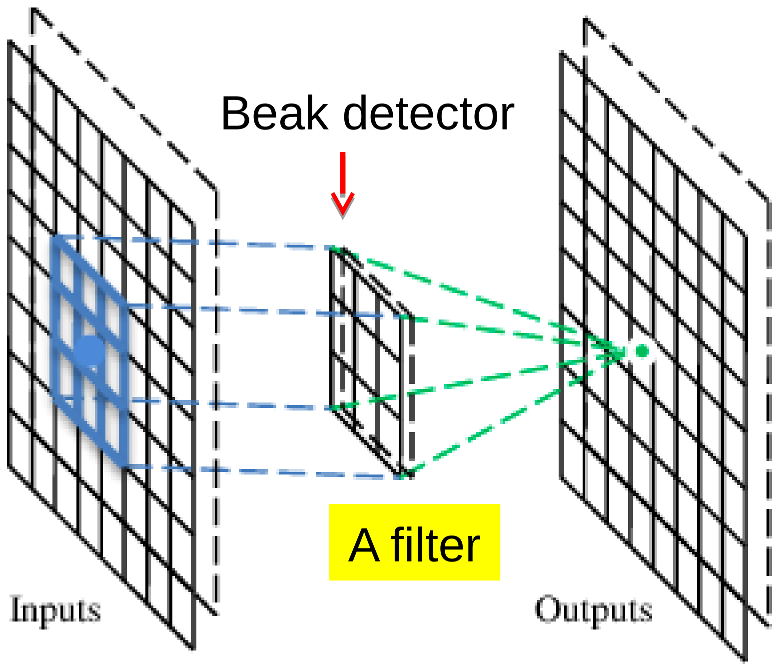


Same pattern appears in different places:
They can be compressed!
What about training a lot of such "small" detectors and each detector must "move around".



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

These are the network parameters to be learned.

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

⋮ ⋮

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

Convolution

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

Dot product

3 -1

6 x 6 image

Convolution

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

Filter 1

If stride=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

3 -3

6 x 6 image

Convolution

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

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Convolution

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

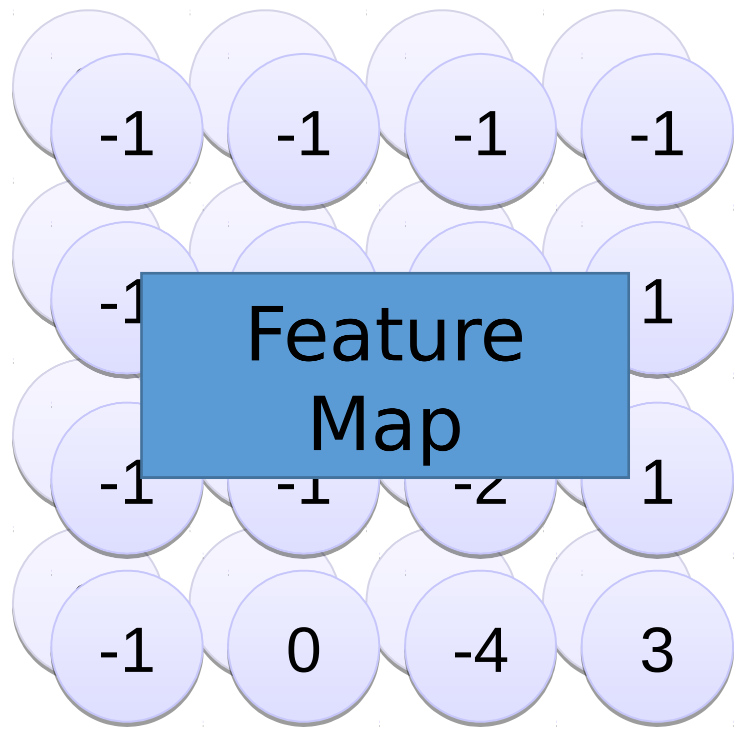
Filter 2

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

Repeat this for each filter



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

Color image: RGB 3 channels

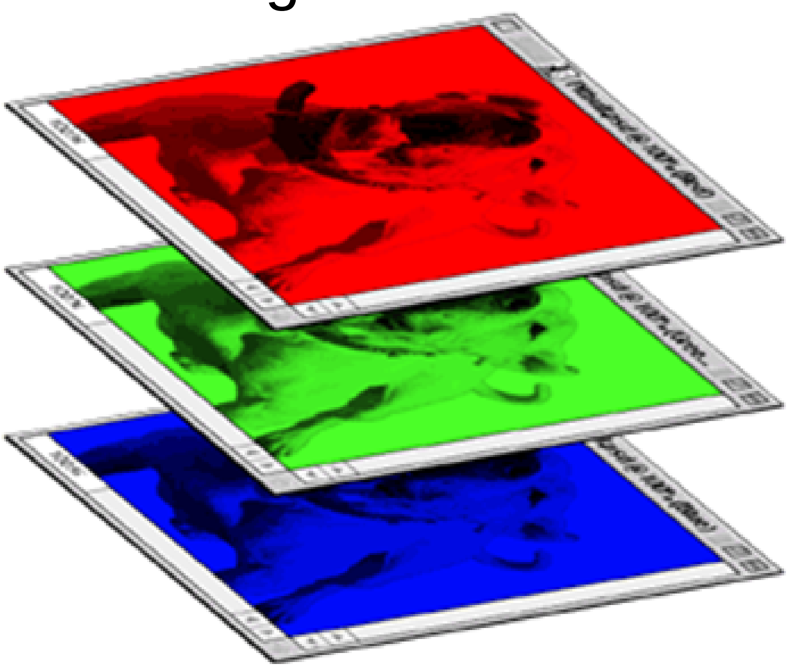
Filter 1

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

Filter 2

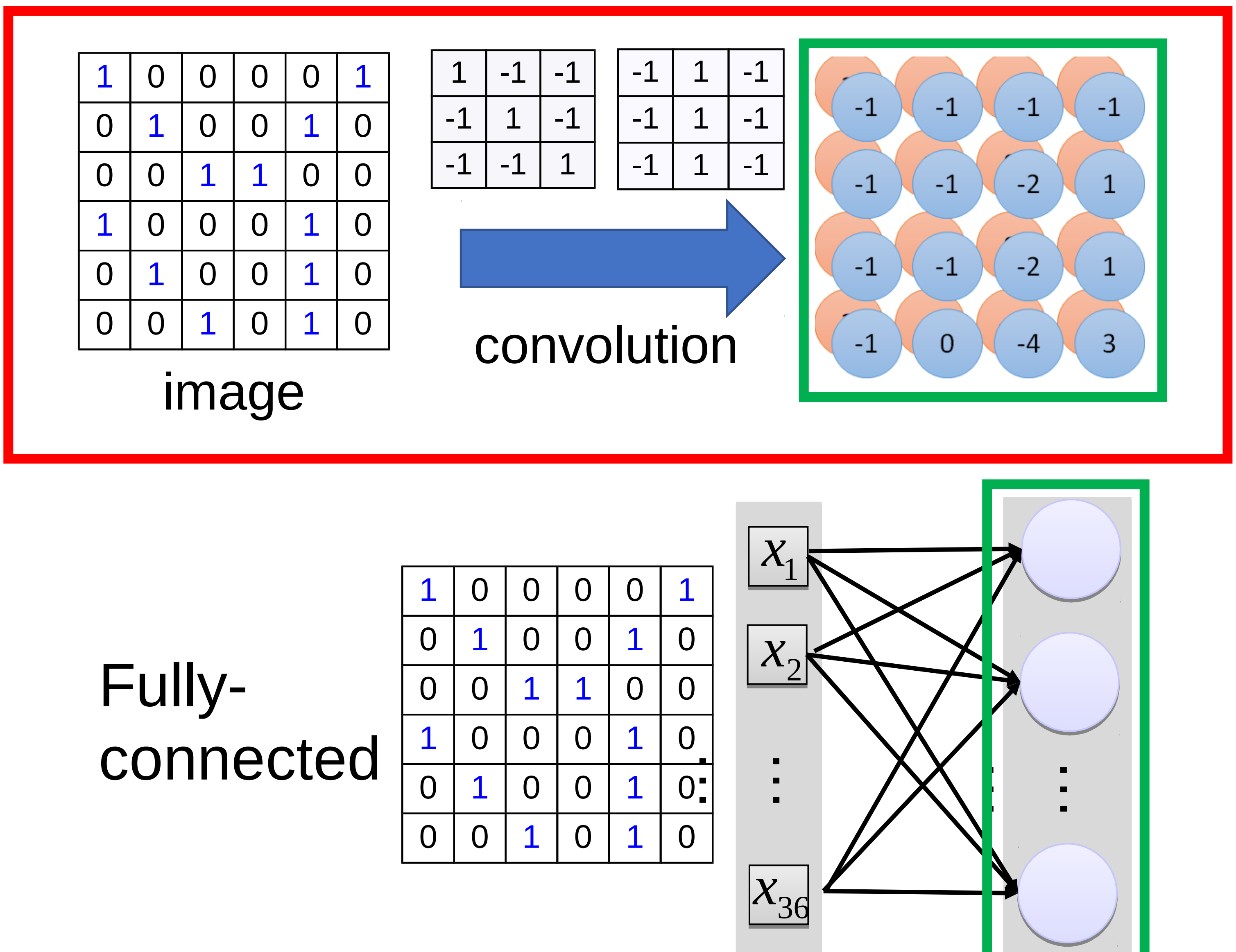
-1	1	-1
-1	1	-1
-1	1	-1

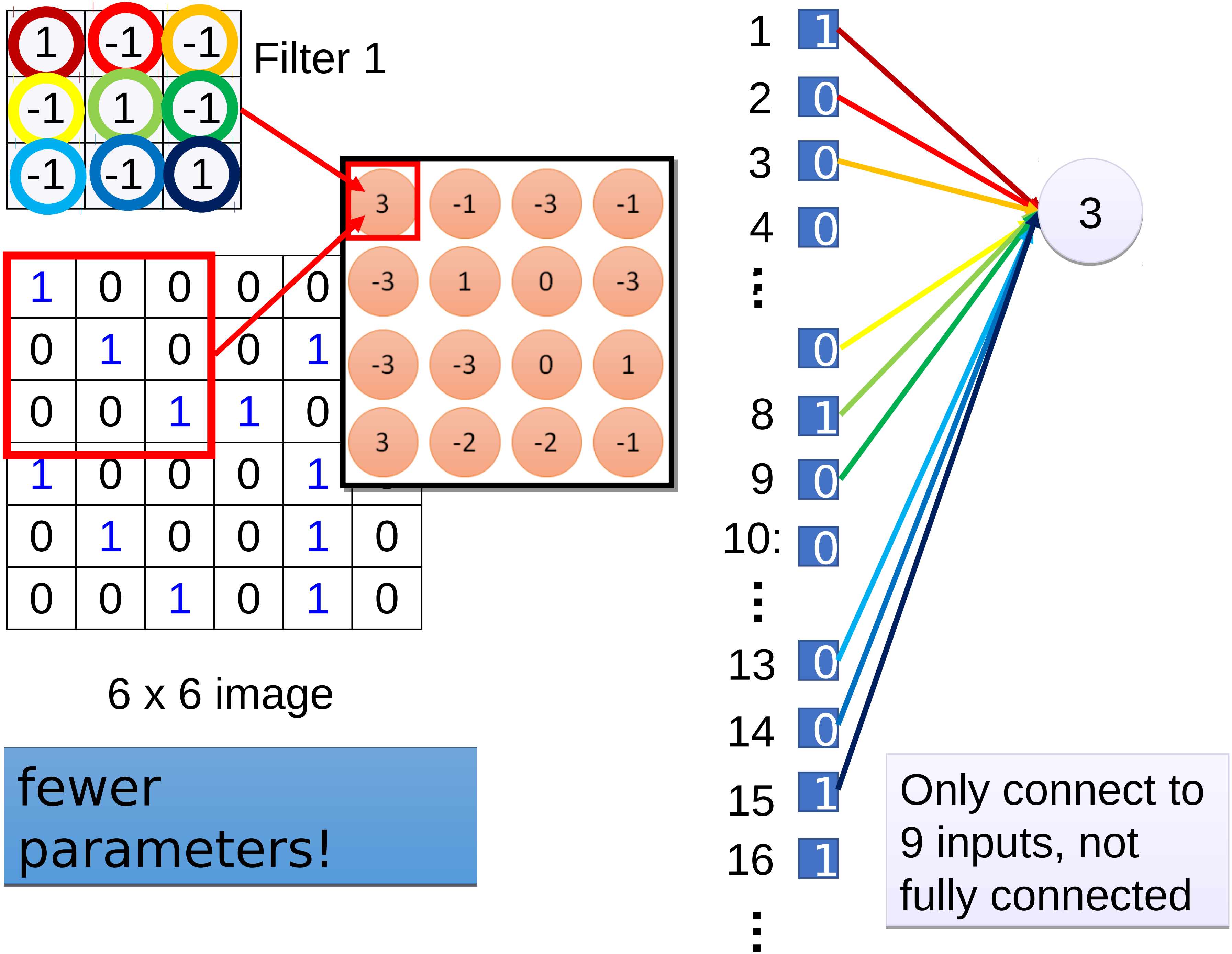
Color image



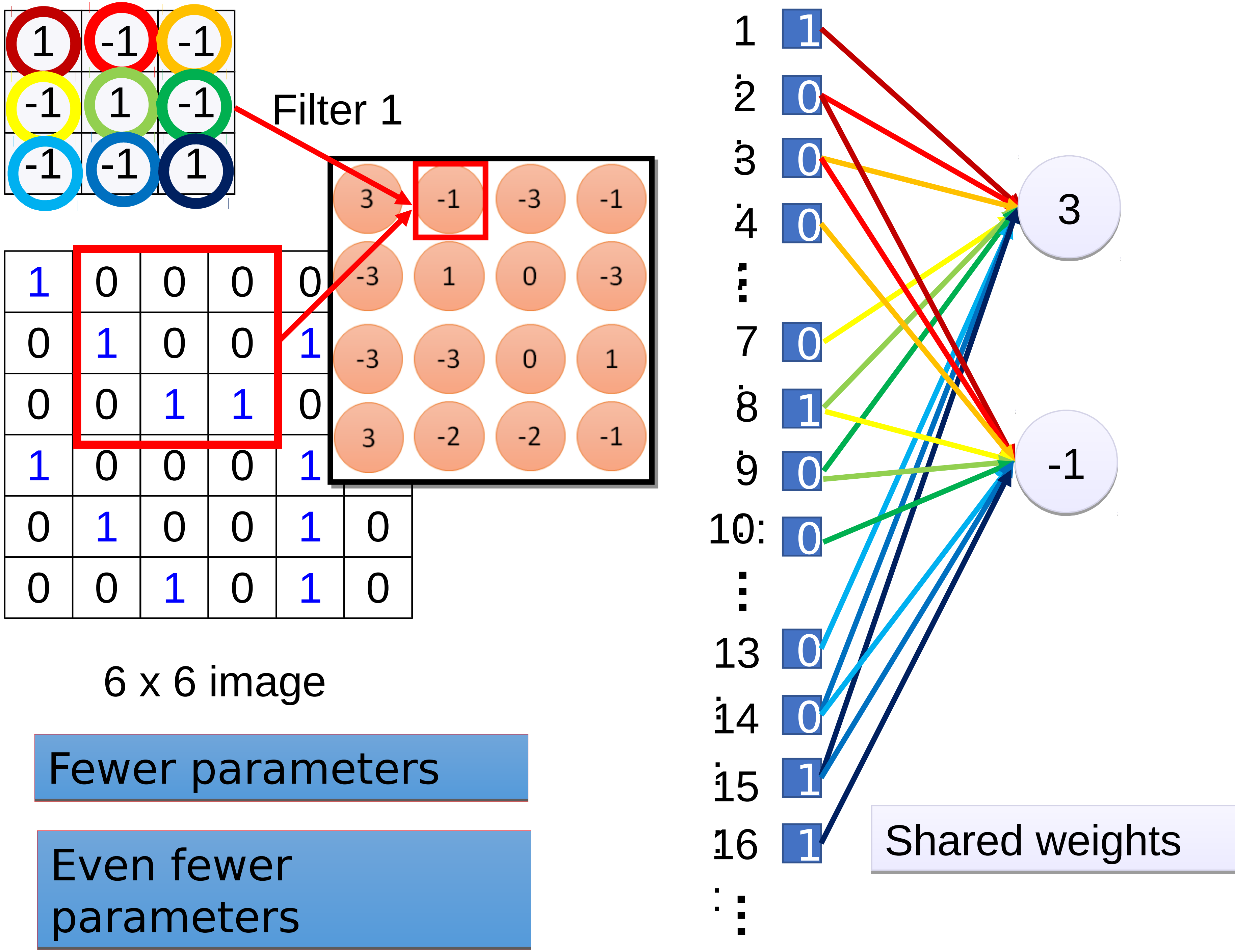
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Convolution v.s. Fully Connected

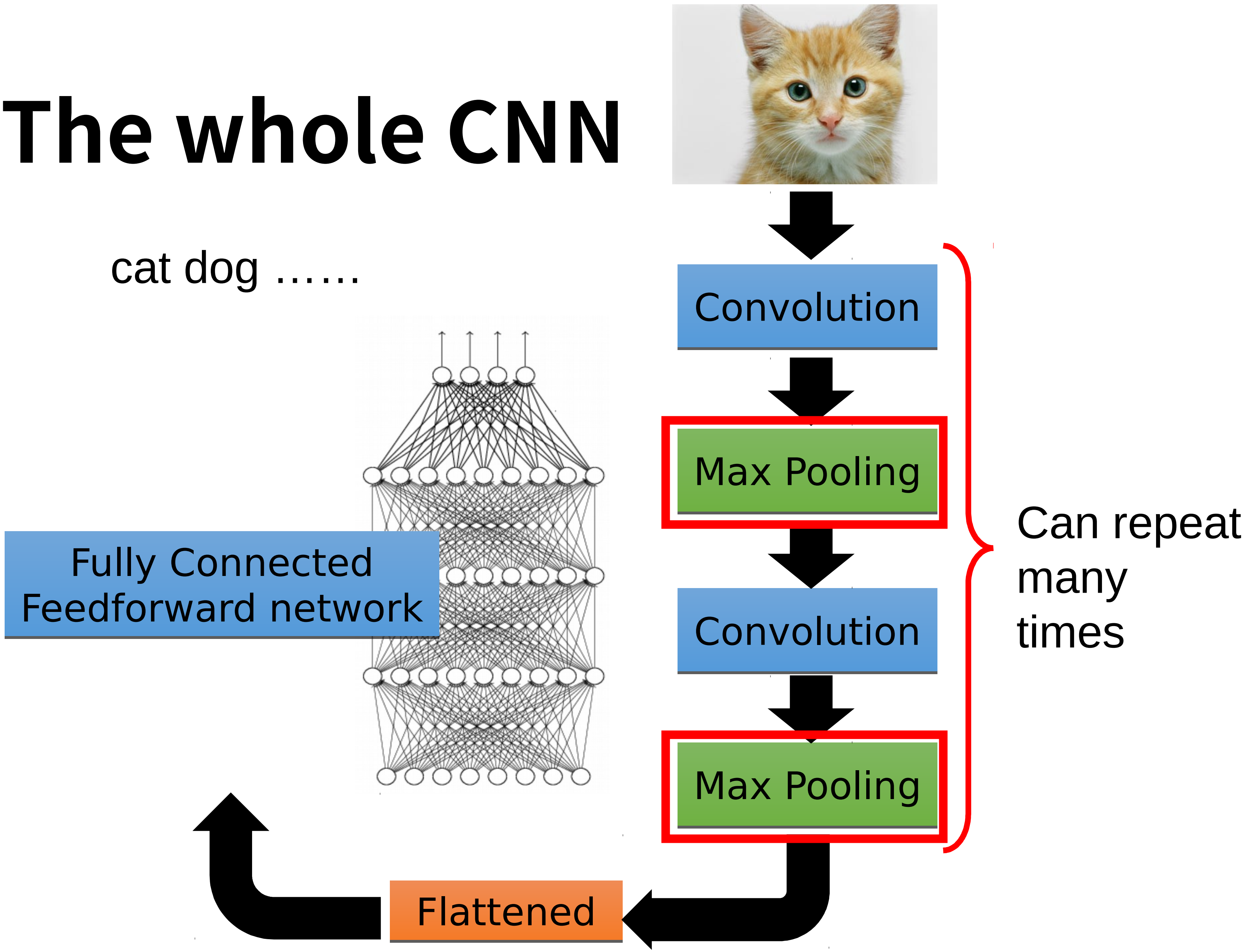




Filter 1
1 -1 -1
-1 1 -1
-1 -1 1
6 x 6 image
1 0 0 0 0
0 1 0 0 1
0 0 1 1 0
1 0 0 0 1
0 1 0 0 1 0
0 0 1 0 1 0
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 1
2 0
3 0
4 0
0
8 1
9 0
10: 0
13 0
14 0
15 1
16 1
3
fewer parameters!
Only connect to 9 inputs, not fully connected


Filter 1
1 -1 -1
-1 1 -1
-1 -1 1
6 x 6 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
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 2 3 4 7 8 9 10 13 14 15 16
3
-1
Shared weights
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

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	-4	3

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:

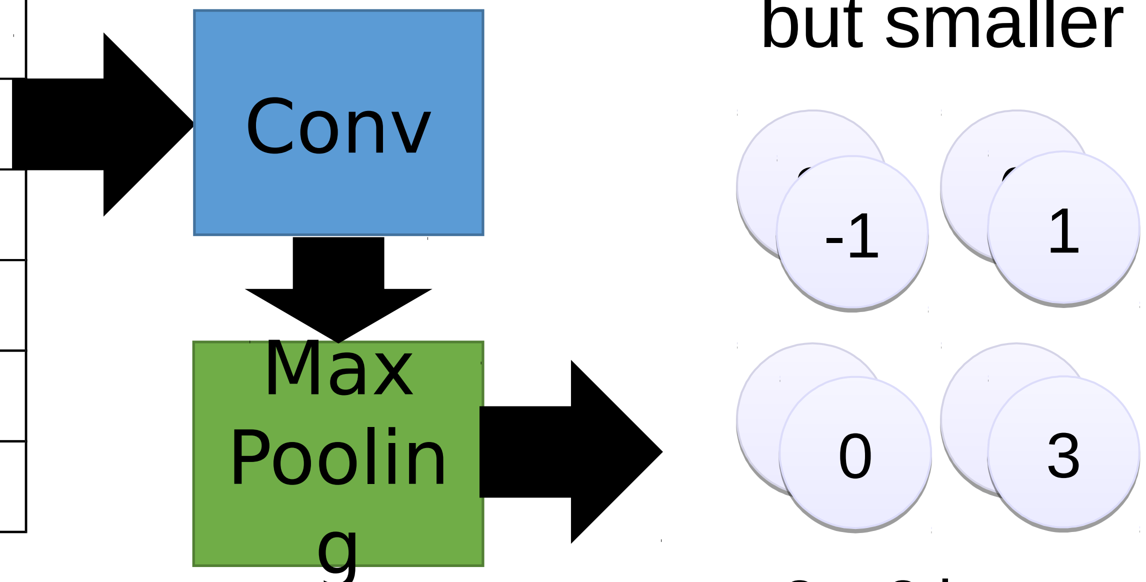
- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

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

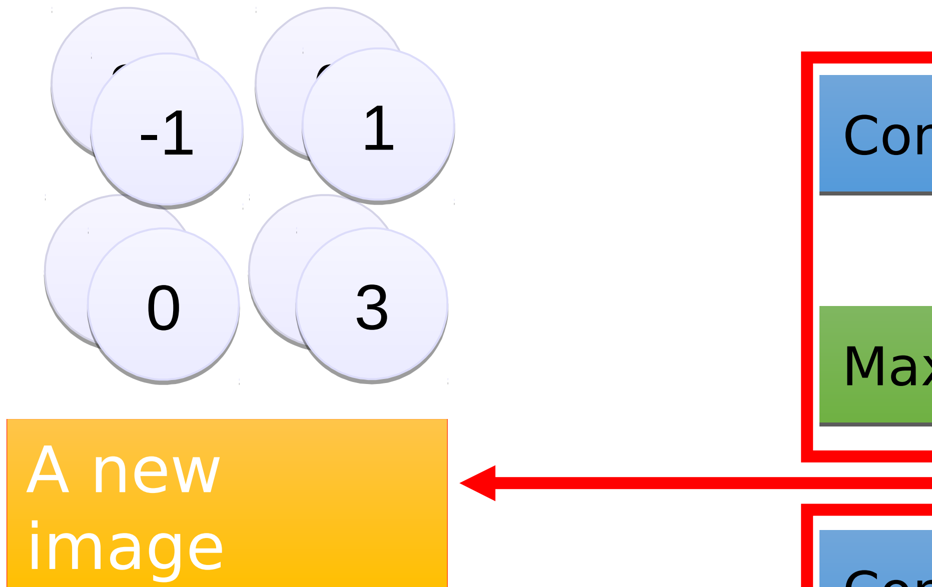
New image but smaller

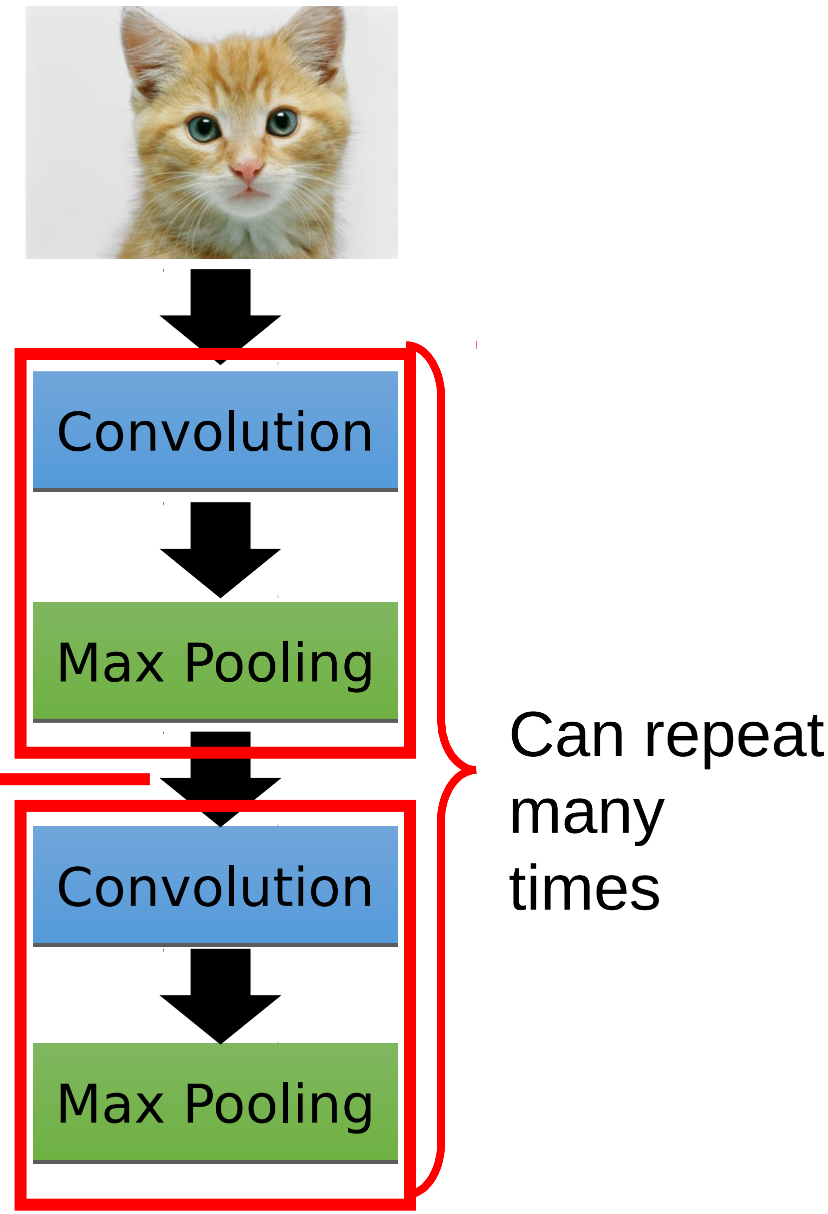


2 x 2 image

Each filter produces a channel

The whole CNN





Smaller than the original image

The number of channels is the number of filters

The whole CNN

Convolution

Max Pooling

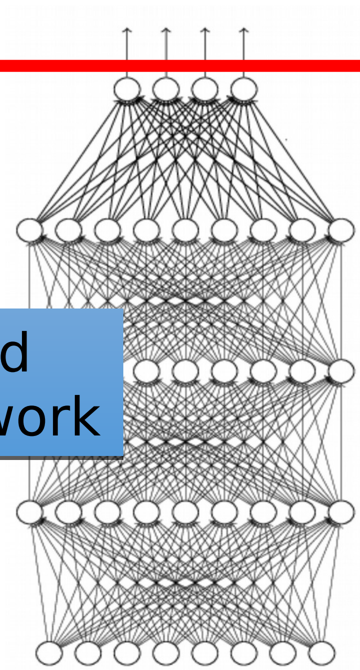
Convolution

A new image

Max Pooling

A new image

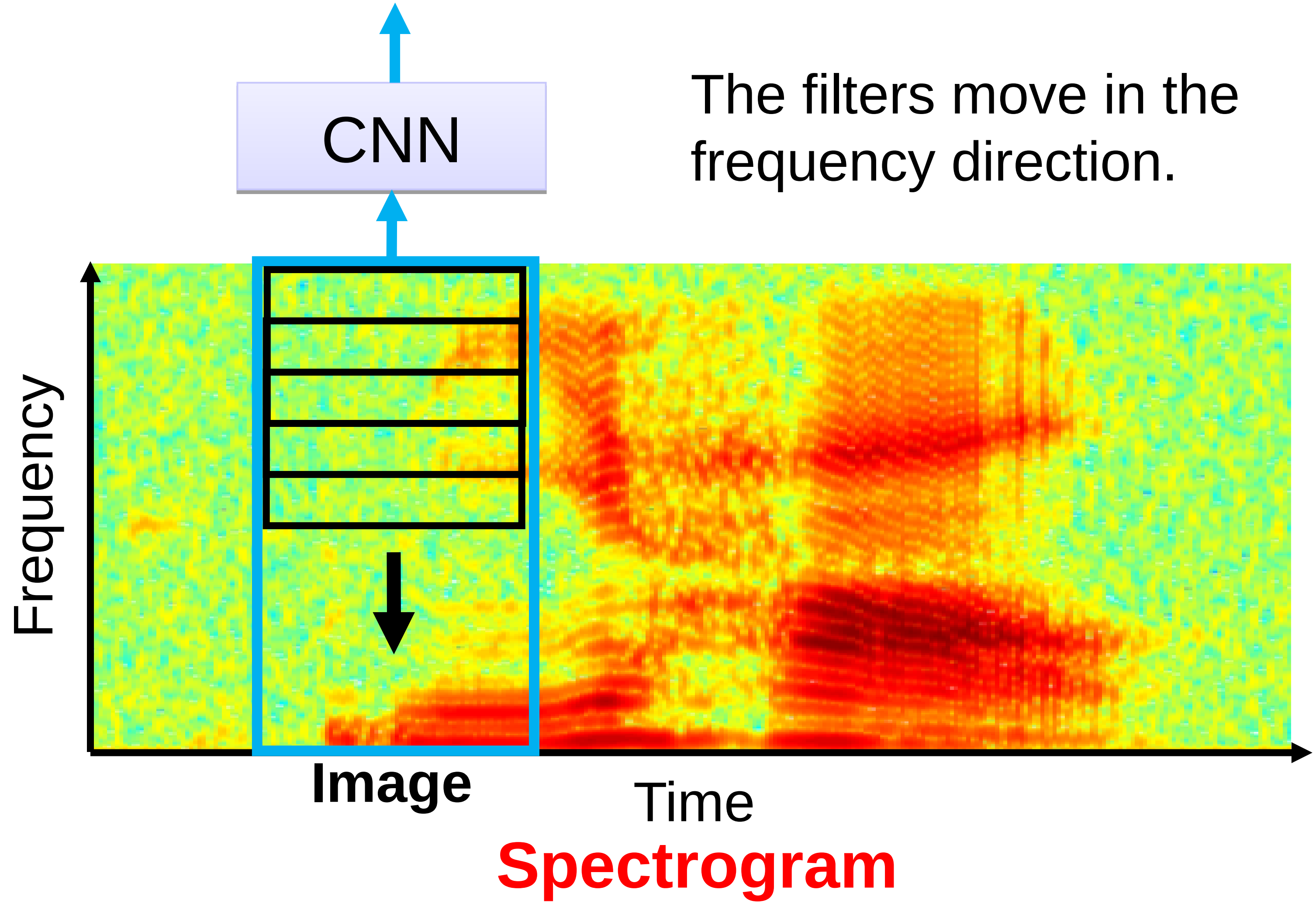
cat dog ……



Fully Connected Feedforward network

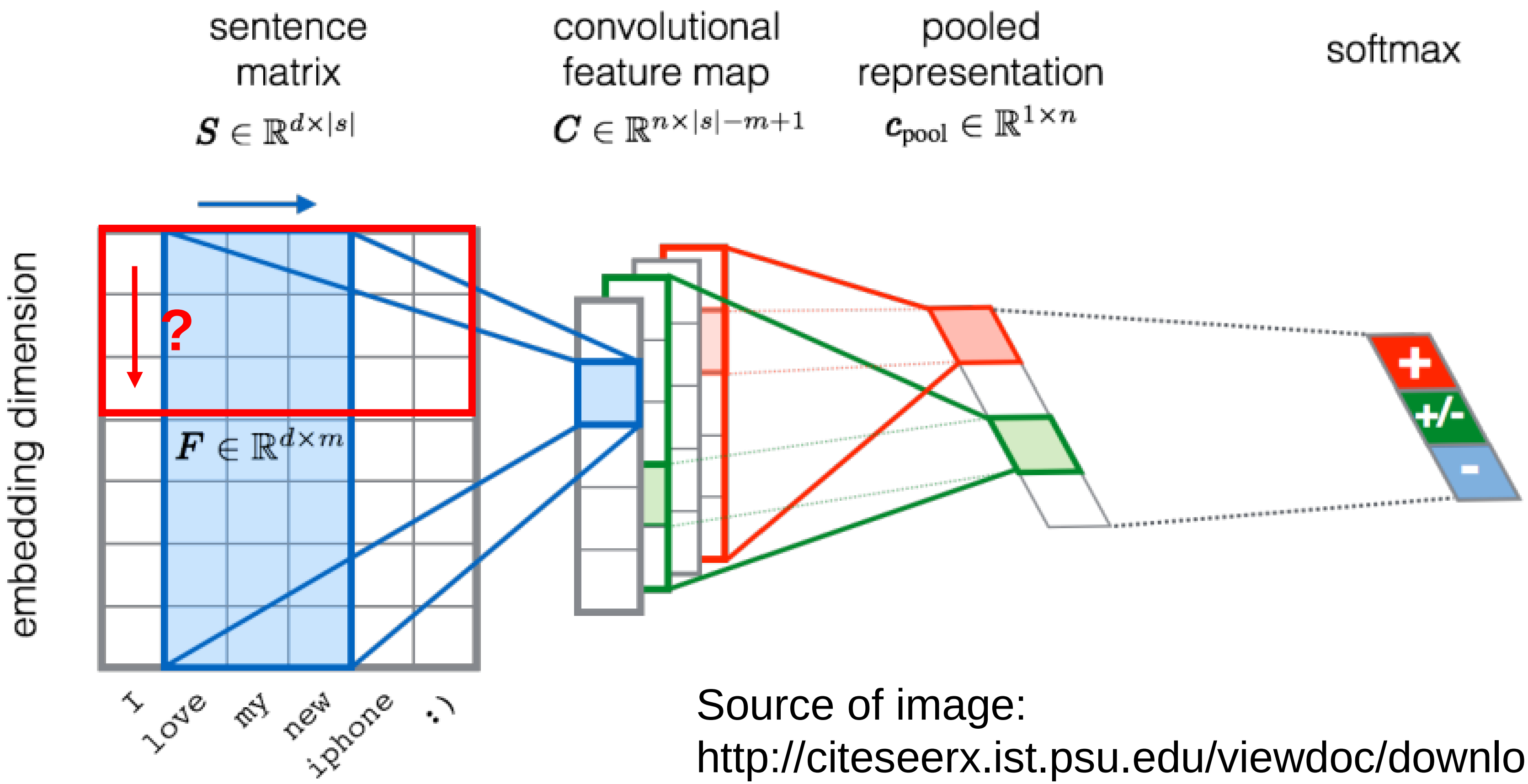
Flattened

CNN in speech recognition



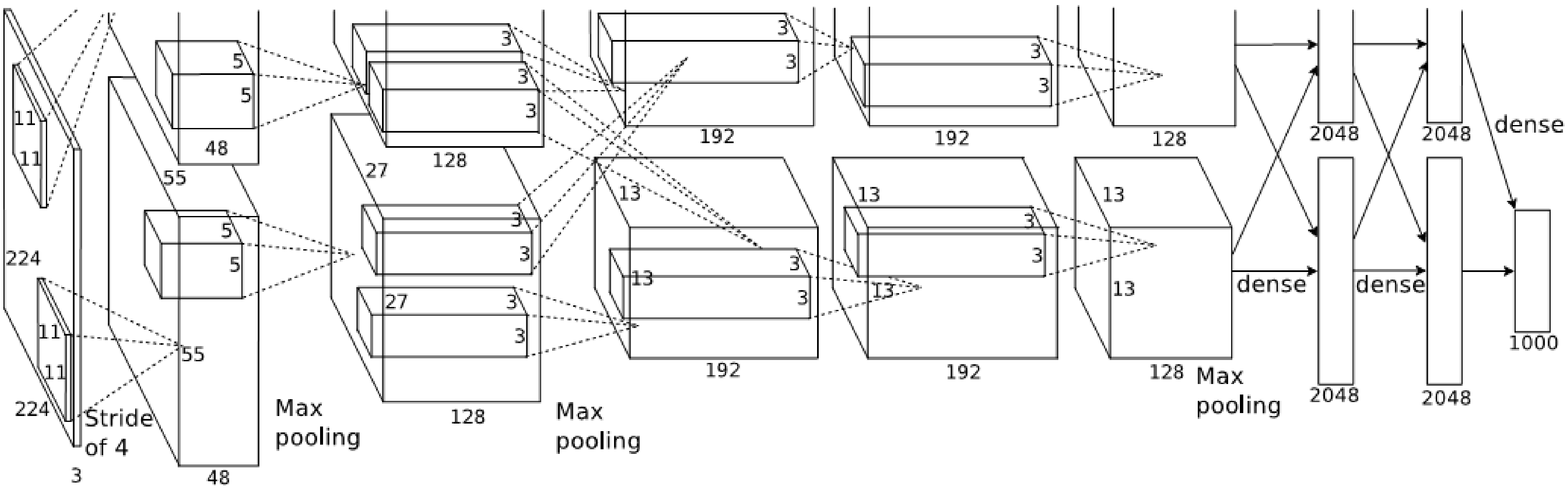
The filters move in the frequency direction.

CNN in text classification



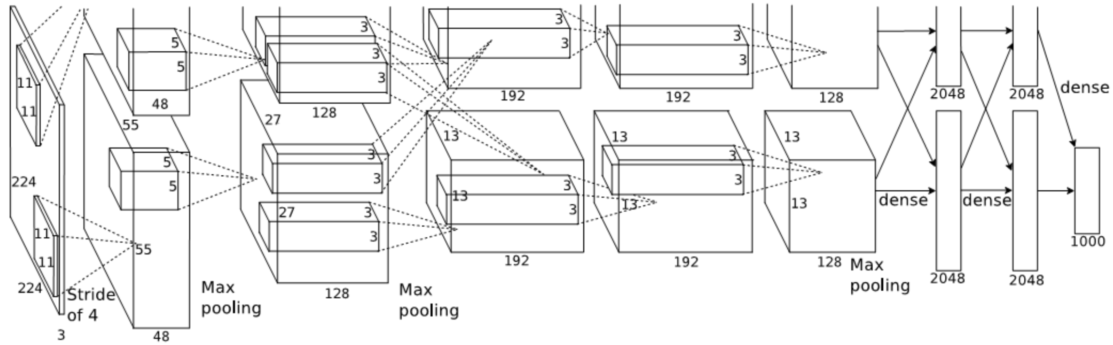
Source of image:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf

Case Study - AlexNet [Krizhevsky et al. 2012]



Case Study - AlexN

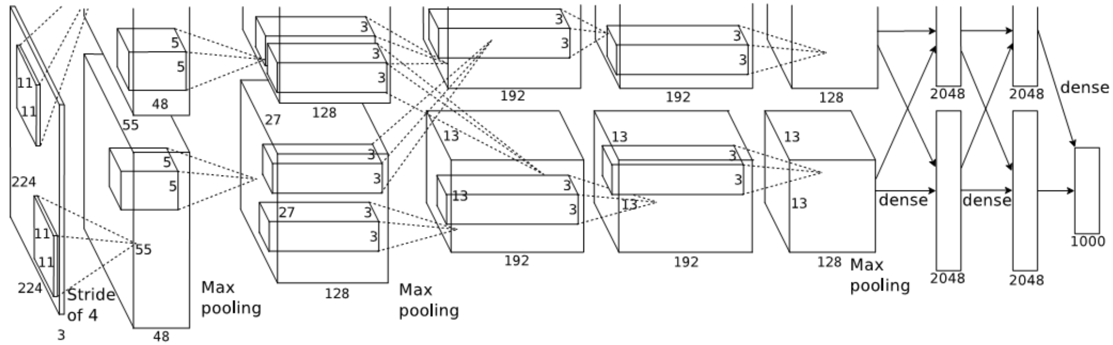
[Krizhevsky et al. 2012]



- Input: 227x227x3 images
- **First layer** (CONV1): 96 11x11 filters applied at stri
  de 4
- Q: what is the output volume size?
- Hint: (227-11)/4+1 = 55
- Output volume [55x55x96]
- Q: What is the total number of parameters in this l
  ayer?
- Parameters: (11*11*3)*96 = 35K

Case Study - AlexN

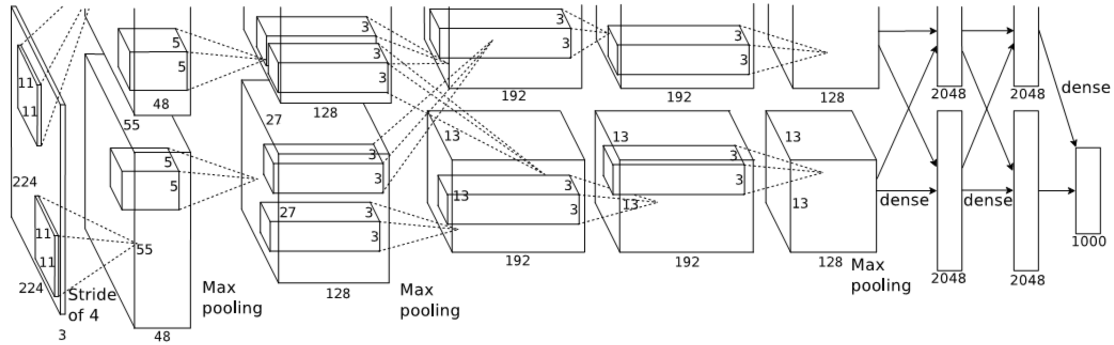
[Krizhevsky et al. 2012]



- Input: 227x227x3 images
- Input: 227x227x3 images
- After CONV1: 55x55x96
- **Second layer** (POOL1): 3x3 filters applied at stride 2
- Q: what is the output volume size? Hint: (55-3)/2+1 = 27
- Output volume: 27x27x96
- Q: what is the number of parameters in this layer?
- Parameters: 0!

Case Study - AlexNe

[Krizhevsky et al. 2012]



- Full (simplified) AlexNet architecture:
- [227x227x3] INPUT
- [55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
- [27x27x96] MAX POOL1: 3x3 filters at stride 2
- [27x27x96] NORM1: Normalization layer [27x27x256]
- CONV2: 256 5x5 filters at stride 1, pad 2 [13x13x256]
- MAX POOL2: 3x3 filters at stride 2 [13x13x256]
- NORM2: Normalization layer
- [13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
- [13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
- [13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
- [6x6x256] MAX POOL3: 3x3 filters at stride 2
- [4096] FC6: 4096 neurons
- [4096] FC7: 4096 neurons
- [1000] FC8: 1000 neurons (class scores)

More Details:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Thank you