

# Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

⋮

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

⋮

$$a^{(L+1)} = g(z^{(L+1)})$$

$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\mathcal{L}(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} \triangleq \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(l+1)}}$$

$$g'(z) = g(z)(1 - g(z))$$

# Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

⋮

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

⋮

$$a^{(L+1)} = g(z^{(L+1)})$$

$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\delta_i^{(L+1)} = \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} [y \log(g(z^{(L+1)})) + (1 - y) \log(1 - g(z^{(L+1)}))]$$

$$\mathcal{L}(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

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

$$\delta_i^{(l+1)} \triangleq \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(l+1)}}$$

$$g'(z) = g(z)(1 - g(z))$$

# Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

⋮

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

⋮

$$a^{(L+1)} = g(z^{(L+1)})$$

$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\begin{aligned} \delta_i^{(L+1)} &= \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} [y \log(g(z^{(L+1)})) + (1 - y) \log(1 - g(z^{(L+1)}))] \\ &= \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1 - y}{1 - g(z^{(L+1)})} g'(z^{(L+1)}) \end{aligned}$$

$$\mathcal{L}(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} \triangleq \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(l+1)}}$$

$$g'(z) = g(z)(1 - g(z))$$

# Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

⋮

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

⋮

$$a^{(L+1)} = g(z^{(L+1)})$$

$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\begin{aligned} \delta_i^{(L+1)} &= \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} [y \log(g(z^{(L+1)})) + (1 - y) \log(1 - g(z^{(L+1)}))] \\ &= \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1 - y}{1 - g(z^{(L+1)})} g'(z^{(L+1)}) \\ &= y - g(z^{(L+1)}) = y - a^{(L+1)} \end{aligned}$$

$$\mathcal{L}(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

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

$$g'(z) = g(z)(1 - g(z))$$

$$\delta_i^{(l+1)} \triangleq \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(l+1)}}$$

# Backprop

Recursive Algorithm!

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

⋮

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

⋮

$$\hat{y} = a^{(L+1)}$$

$$\delta^{(L+1)} = y - a^{(L+1)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\mathcal{L}(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} \triangleq \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(l+1)}}$$

# Backpropagation

Set  $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$  (Used to accumulate gradient)

For each training instance  $(x_k, y_k)$

Set  $\mathbf{a}^{(1)} = x_k$

Compute  $\{\mathbf{a}^{(2)}, \dots, \mathbf{a}^{(L)}\}$  via forward propagation

Compute  $\delta^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors  $\{\delta^{(L-1)}, \dots, \delta^{(2)}\}$

Compute gradients  $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Compute avg regularized gradient  $D_{ij}^{(l)} = \begin{cases} \frac{1}{n} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n} \Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

Intercept do not have regularizer

Average loss +  $\ell_2$  regularizer

$$\frac{1}{n} \sum_{k=1}^n L(y_k, \hat{y}) + \lambda \|\Theta\|_2^2$$

# Convolutional Neural Networks

---



# Multi-layer Neural Network

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

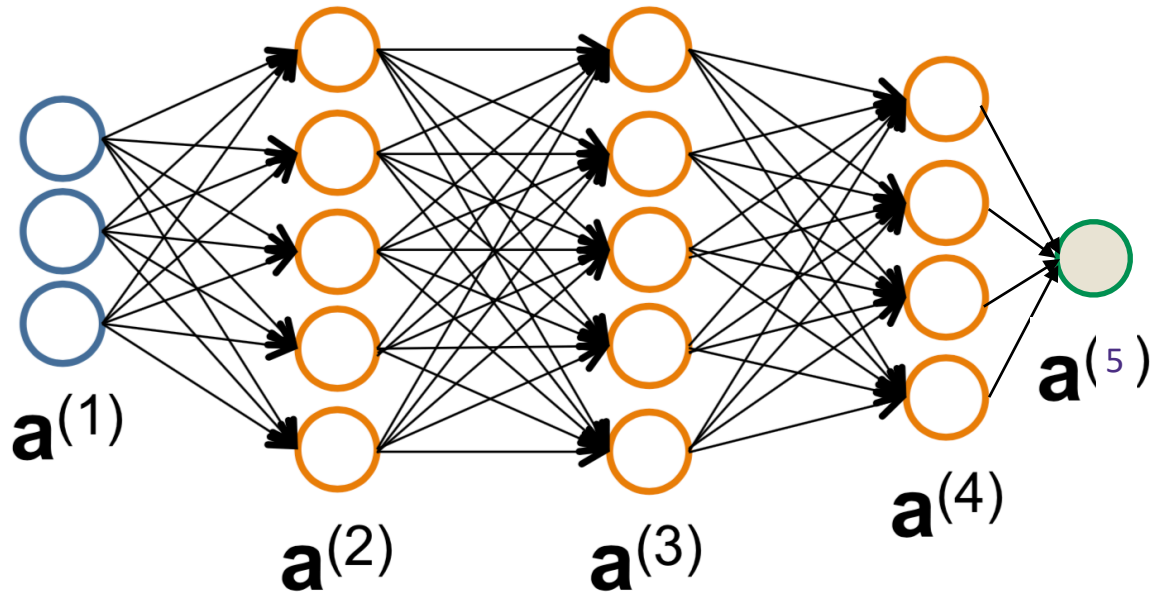
⋮

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

⋮

$$\hat{y} = a^{(L+1)}$$



$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

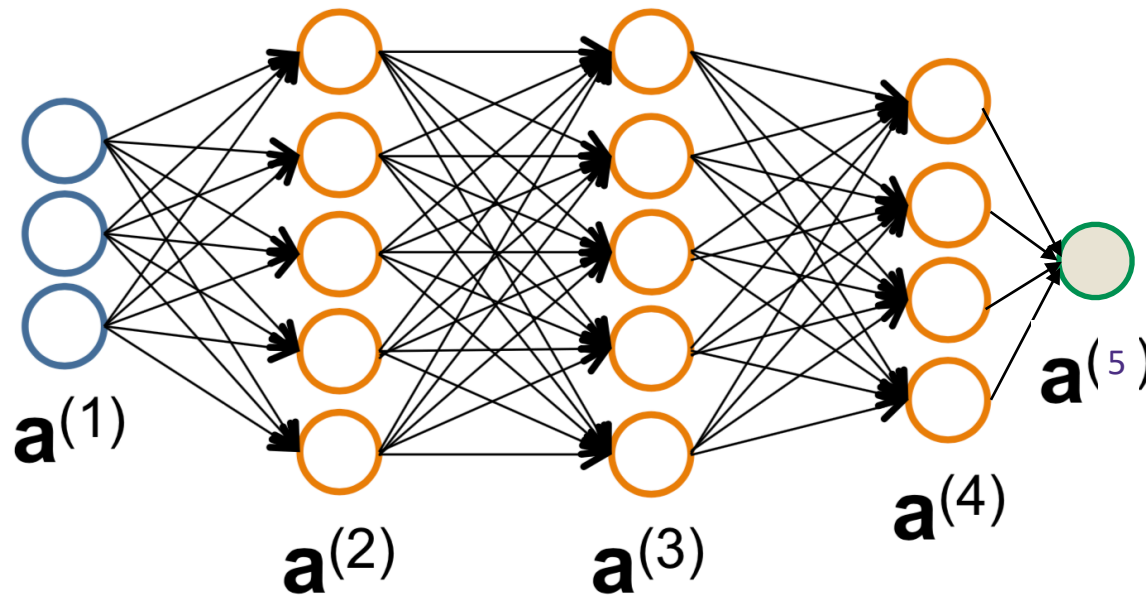
$$g(z) = \frac{1}{1 + e^{-z}}$$

Binary  
Logistic  
Regression



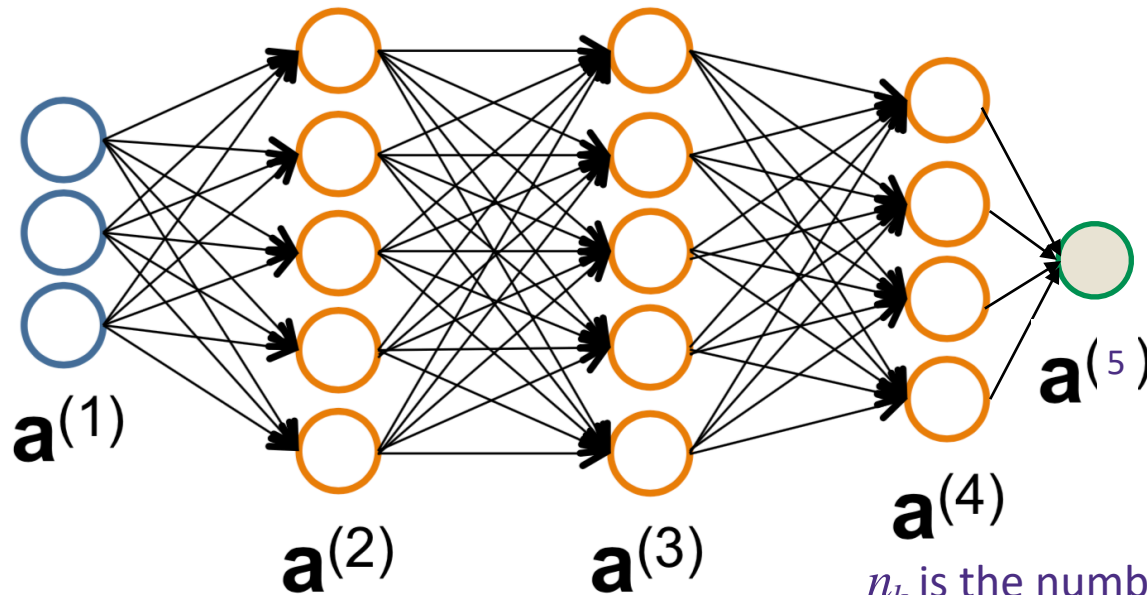
# Neural Network Architecture

- The neural network architecture is defined by
  - the number of layers (depth of a network),
  - the number of nodes in each layer (width of a layer),
  - and also by **allowable edges** and **shared weights**.



# Neural Network Architecture

The neural network architecture is defined by the number of layers, and the number of nodes in each layer, and also by **allowable edges** and **shared weights**.



$n_k$  is the number of nodes in layer  $k$

We say a layer is **Fully Connected (FC)** if all linear mappings from the current layer to the next layer are permissible.

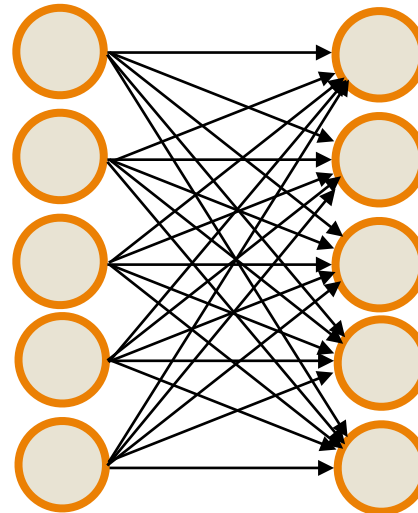
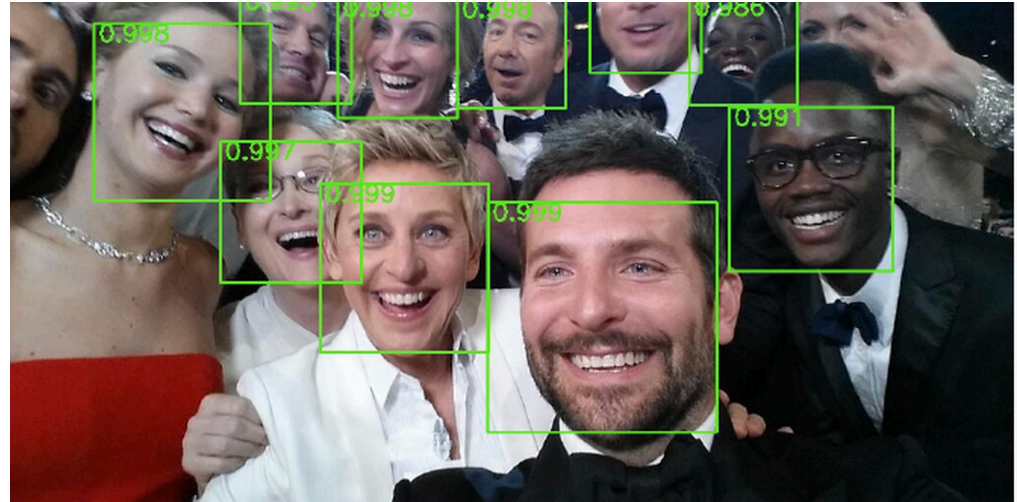
$$\mathbf{a}^{(k+1)} = g(\Theta \mathbf{a}^{(k)}) \quad \text{for any } \Theta \in \mathbb{R}^{n_{k+1} \times n_k}$$

A lot of parameters!!  $n_1 n_2 + n_2 n_3 + \cdots + n_L n_{L+1}$

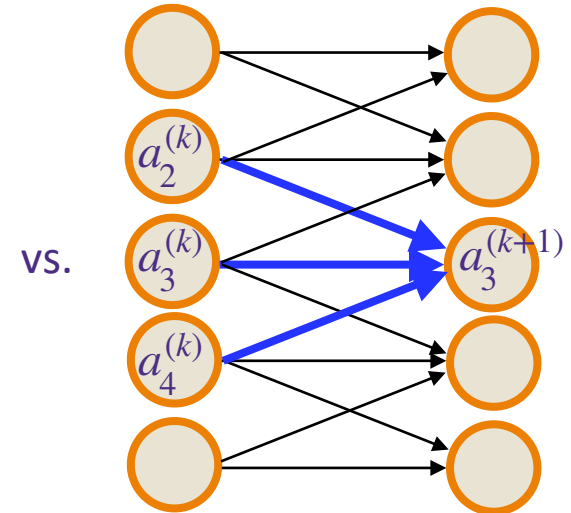
# Neural Network Architecture

Finding faces require only local patterns

- Objects in an image are often **localized in space** so to find the faces in an image, not every pixel is important for classification
- Makes sense to drag a window across an image, focusing a local region at a time
- Although images are two-dimensional, we use one-dimensional examples to illustrate the main idea
- Similarly, to identify edges or other local structure, it makes sense to only look at **local information**



This is a fully connected layer

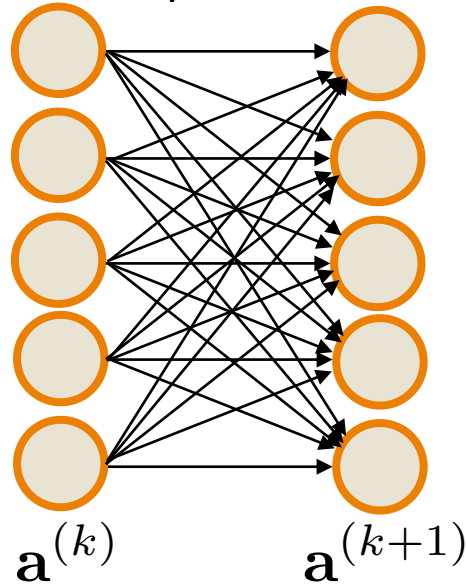


vs.

This has sparse and local connections

# Neural Network Architecture

Fully connected



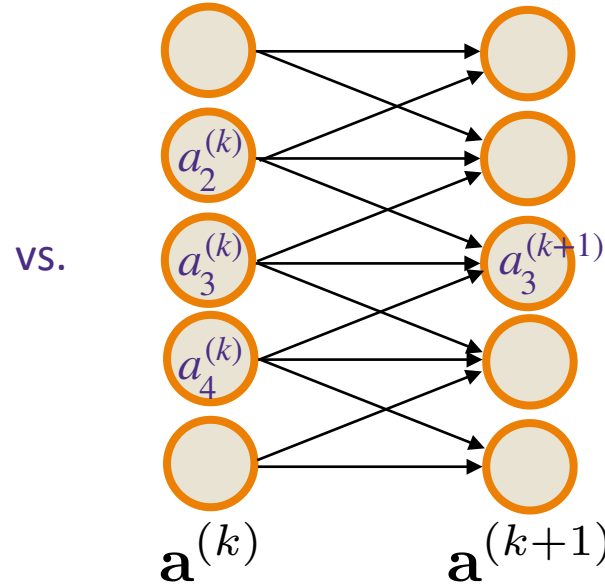
$$\Theta^{(k)} = \begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

# of Parameters  
in this layer:

$$n^2$$

$$\mathbf{a}_i^{(k+1)} = g \left( \sum_{j=0}^{n-1} \Theta_{i,j} \mathbf{a}_j^{(k)} \right)$$

sparse and local connections



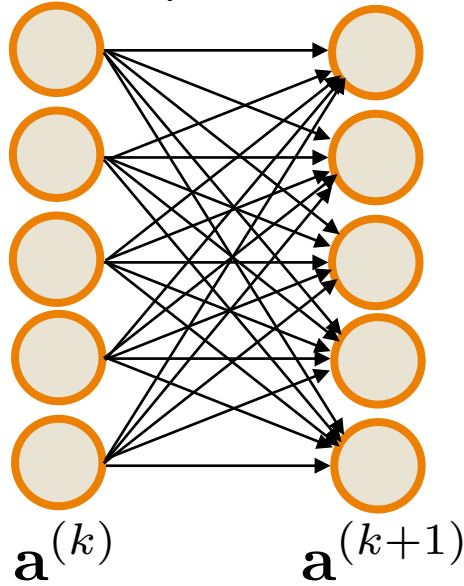
$$\Theta^{(k)} = \begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & 0 & 0 & 0 \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & 0 & 0 \\ 0 & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & 0 \\ 0 & 0 & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ 0 & 0 & 0 & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

$$3n - 2$$

# Neural Network Architecture

Shift invariance: A local pattern of interest can appear anywhere in the image

Fully connected



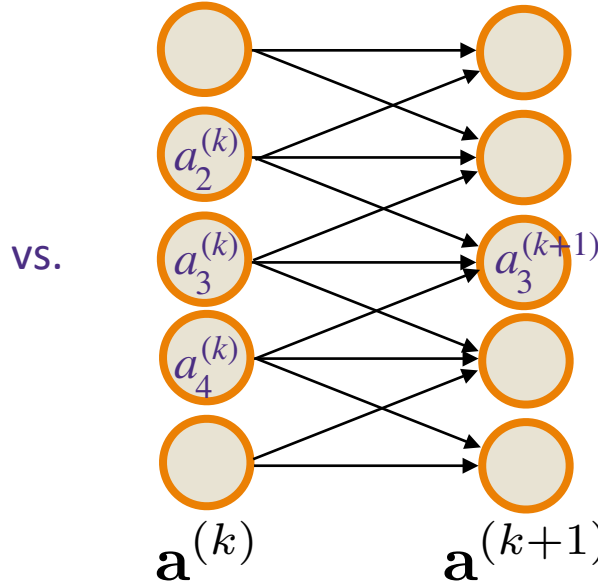
$$\Theta^{(k)} = \begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

# of Parameters in this layer:

$$n^2$$

$$\mathbf{a}_i^{(k+1)} = g \left( \sum_{j=0}^{n-1} \Theta_{i,j} \mathbf{a}_j^{(k)} \right)$$

sparse and local connections

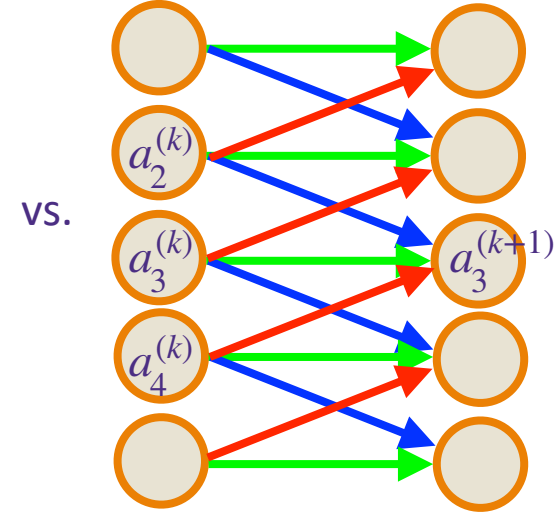


$$\Theta^{(k)} = \begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & 0 & 0 & 0 \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & 0 & 0 \\ 0 & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & 0 \\ 0 & 0 & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ 0 & 0 & 0 & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

$$3n - 2$$

$$\mathbf{a}_i^{(k+1)} = g \left( \sum_{j=-(m-1)/2}^{(m-1)/2} \theta_{j+(m-1)/2} \mathbf{a}_{i+j}^{(k)} \right)$$

sparse local connections and shared weights

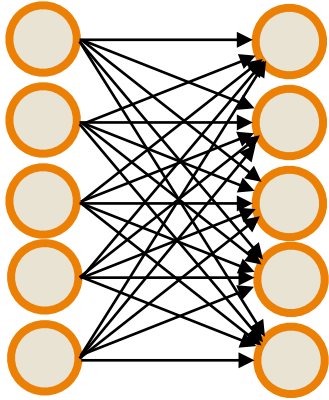


$$\Theta^{(k)} = \begin{bmatrix} \theta_1 & \theta_2 & 0 & 0 & 0 \\ \theta_0 & \theta_1 & \theta_2 & 0 & 0 \\ 0 & \theta_0 & \theta_1 & \theta_2 & 0 \\ 0 & 0 & \theta_0 & \theta_1 & \theta_2 \\ 0 & 0 & 0 & \theta_0 & \theta_1 \end{bmatrix}$$

$$3$$

# Neural Network Architecture

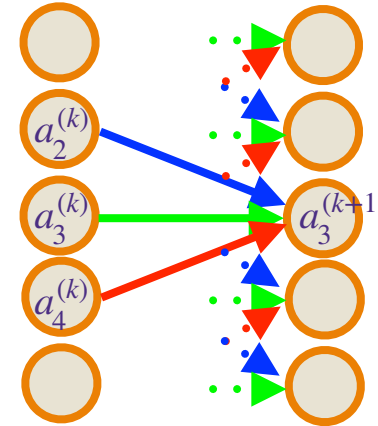
## Fully Connected (FC) Layer



$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

$$\mathbf{a}_i^{(k+1)} = g \left( \sum_{j=0}^{n-1} \Theta_{i,j} \mathbf{a}_j^{(k)} \right)$$

## Convolutional (CONV) Layer (1 filter)



Filter with  $m = 3$

$$\mathbf{a}_3^{(k+1)} \rightarrow \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & 0 & 0 \\ \theta_0 & \theta_1 & \theta_2 & 0 & 0 \\ 0 & \theta_0 & \theta_1 & \theta_2 & 0 \\ 0 & 0 & \theta_0 & \theta_1 & \theta_2 \\ 0 & 0 & 0 & \theta_0 & \theta_1 \end{bmatrix}$$

$$\mathbf{a}_i^{(k+1)} = g \left( \sum_{j=-(m-1)/2}^{(m-1)/2} \theta_{j+(m-1)/2} \mathbf{a}_{i+j}^{(k)} \right) = g([\theta \star \mathbf{a}^{(k)}]_i)$$

$\star$  = Convolution

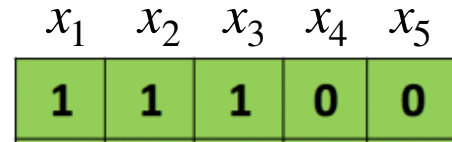
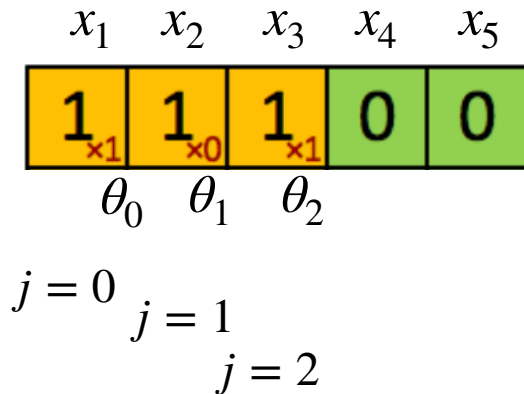
$\theta = (\theta_0, \dots, \theta_{m-1}) \in \mathbb{R}^m$  is referred to as a “filter”

Because of shift invariance and locality of computer vision tasks, convolution is extremely powerful

# Example (1d convolution)

- Notice that the indexing of the convolution is slightly different from previous slide
- There are many different ways to write the same convolution

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$



Input  $x \in \mathbb{R}^n$

$\theta_0 \ \theta_1 \ \theta_2$



Filter  $\theta \in \mathbb{R}^m$

$i = 1$



Output  $\theta * x \in \mathbb{R}^{n-m+1}$

# Example (1d convolution)

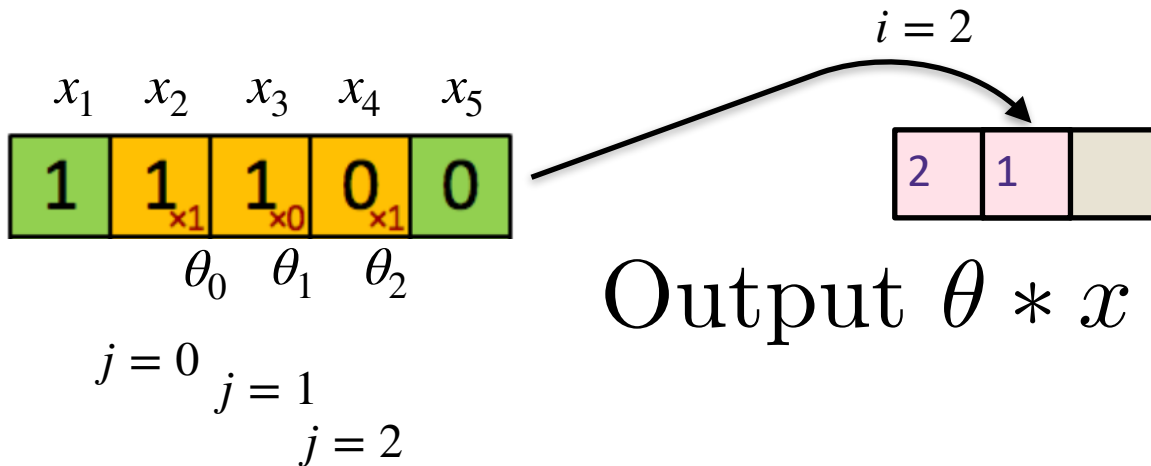
$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
1	1	1	0	0

Input  $x \in \mathbb{R}^n$

$\theta_0$	$\theta_1$	$\theta_2$
1	0	1

Filter  $\theta \in \mathbb{R}^m$

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$



Output  $\theta * x \in \mathbb{R}^{n-m+1}$



# Example (1d convolution)

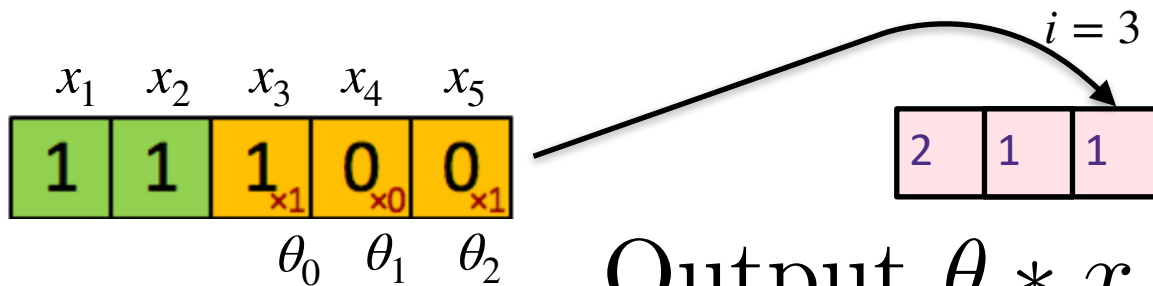
$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$

$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
1	1	1	0	0

Input  $x \in \mathbb{R}^n$

$\theta_0$	$\theta_1$	$\theta_2$
1	0	1

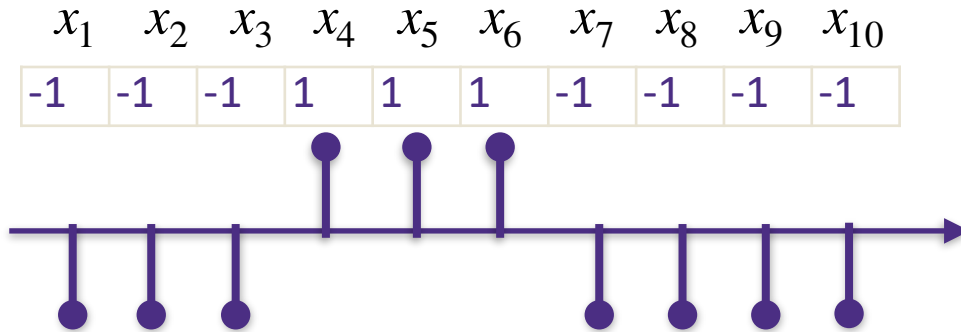
Filter  $\theta \in \mathbb{R}^m$



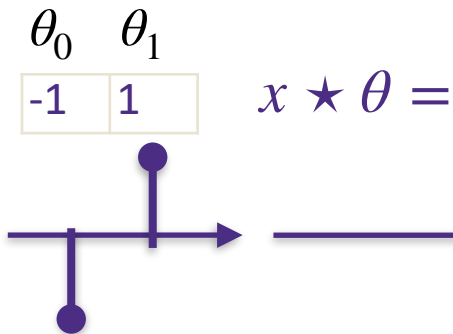
Output  $\theta * x \in \mathbb{R}^{n-m+1}$

# 1d convolution

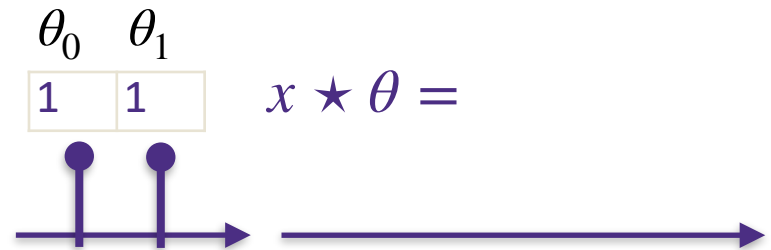
- Each filter finds a specific pattern over the input



Filter 1



Filter 2



- We use many such convolutional filters per layer in practice
- Each convolutional filter output vector (or a matrix if 2D convolution) is called a channel

# Convolution of images (2d convolution)

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

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

Image  $I$

1	0	1
0	1	0
1	0	1

Filter  $K$

1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	1	0
0 <sub>x1</sub>	0 <sub>x0</sub>	1 <sub>x1</sub>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved  
Feature

$$I * K$$

# Convolution of images (2d convolution)

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

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

Image  $I$

1	0	1
0	1	0
1	0	1

Filter  $K$

1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	1	0
0 <sub>x1</sub>	0 <sub>x0</sub>	1 <sub>x1</sub>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved  
Feature

$$I * K$$

# Convolution of images

- These are hand-crafted filters, to illustrate what the weights of a filter mean
- Filter in a Convolutional Neural Network (CNN) is learned, and we might be able to interpret what we learned

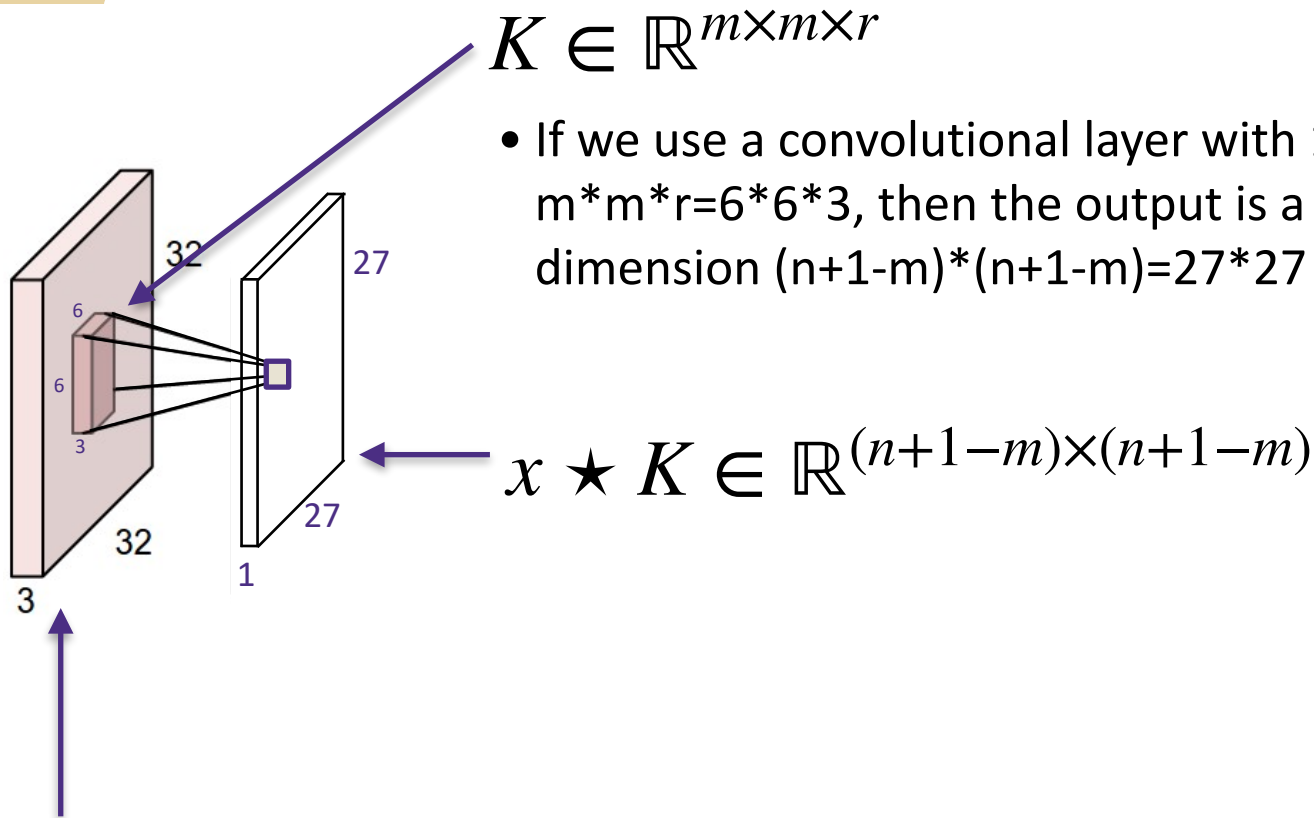
$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

Image  $I$



Operation	Filter $K$	Convolved Image $I * K$
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

# Stacking convolved images



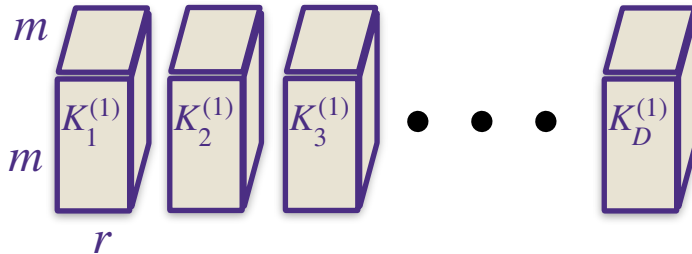
- If we use a convolutional layer with 1 filter of size  $m \times m \times r = 6 \times 6 \times 3$ , then the output is a matrix of dimension  $(n+1-m) \times (n+1-m) = 27 \times 27$

- Input is a multi-array or a tensor, because it has 3 color channels

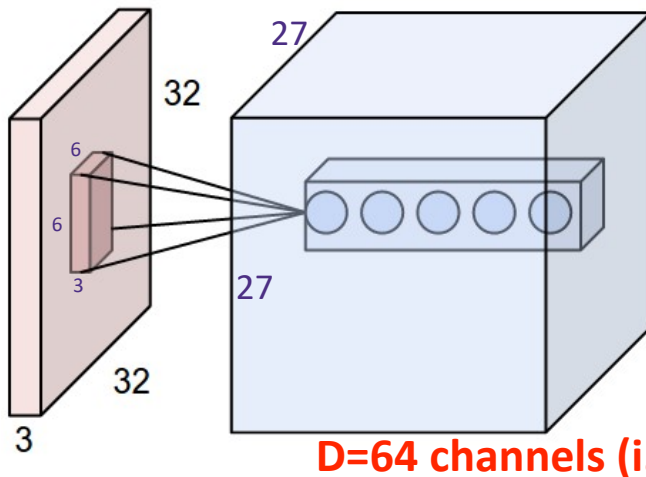
$$x \in \mathbb{R}^{n \times n \times r}$$

# Stacking convolved images

- Typical convolutional layer has multiple filters to capture multiple patterns
- Each one is called a channel
- Each channel has a filter of the same size  $m*m*r$  but with different weights



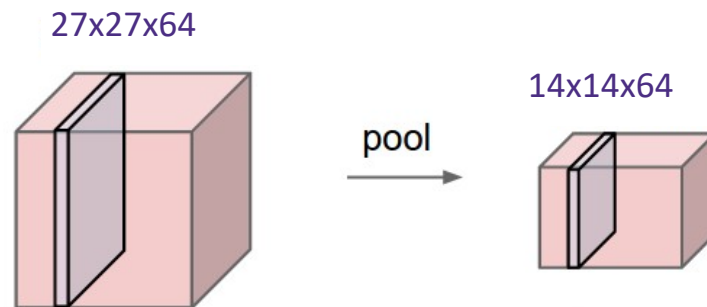
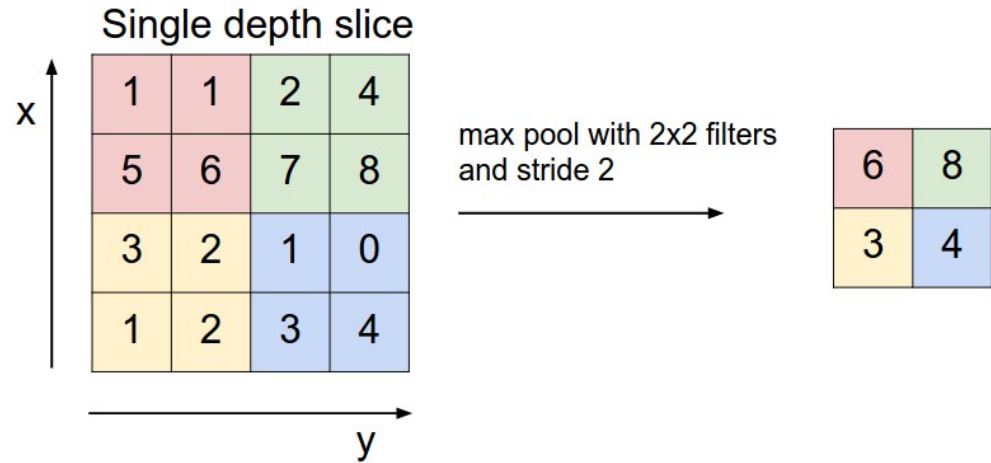
- Each channel outputs a matrix of dimension  $(n+1-m)*(n+1-m)$
- Put together the output is a tensor of dimension  $(n+1-m)*(n+1-m)*D$



**Repeat with D filters!**

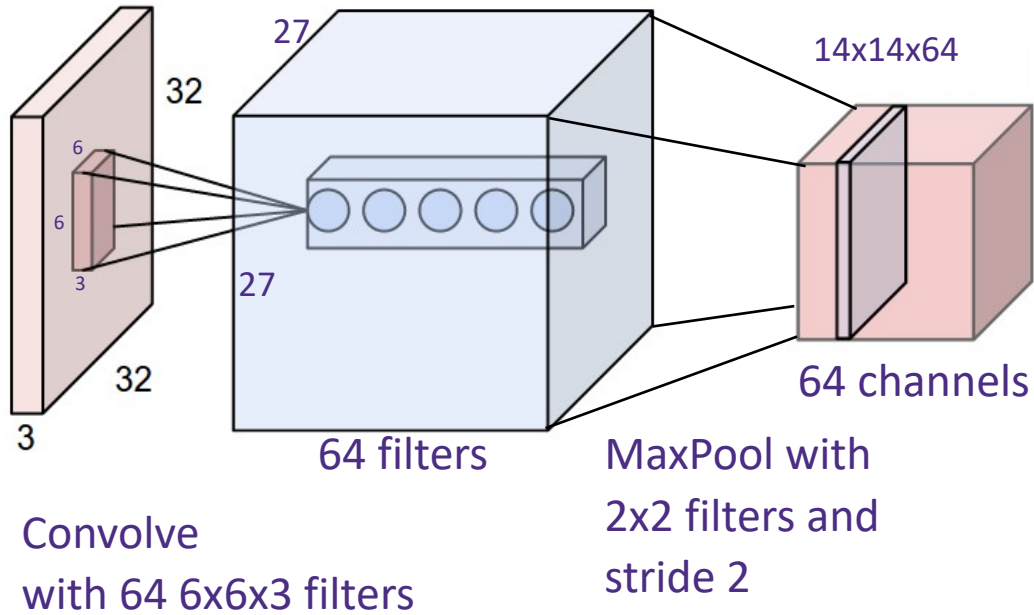
# Max Pooling gives a summary of a region

Pooling reduces the dimension and can be interpreted as “This filter had a high response in this general region”

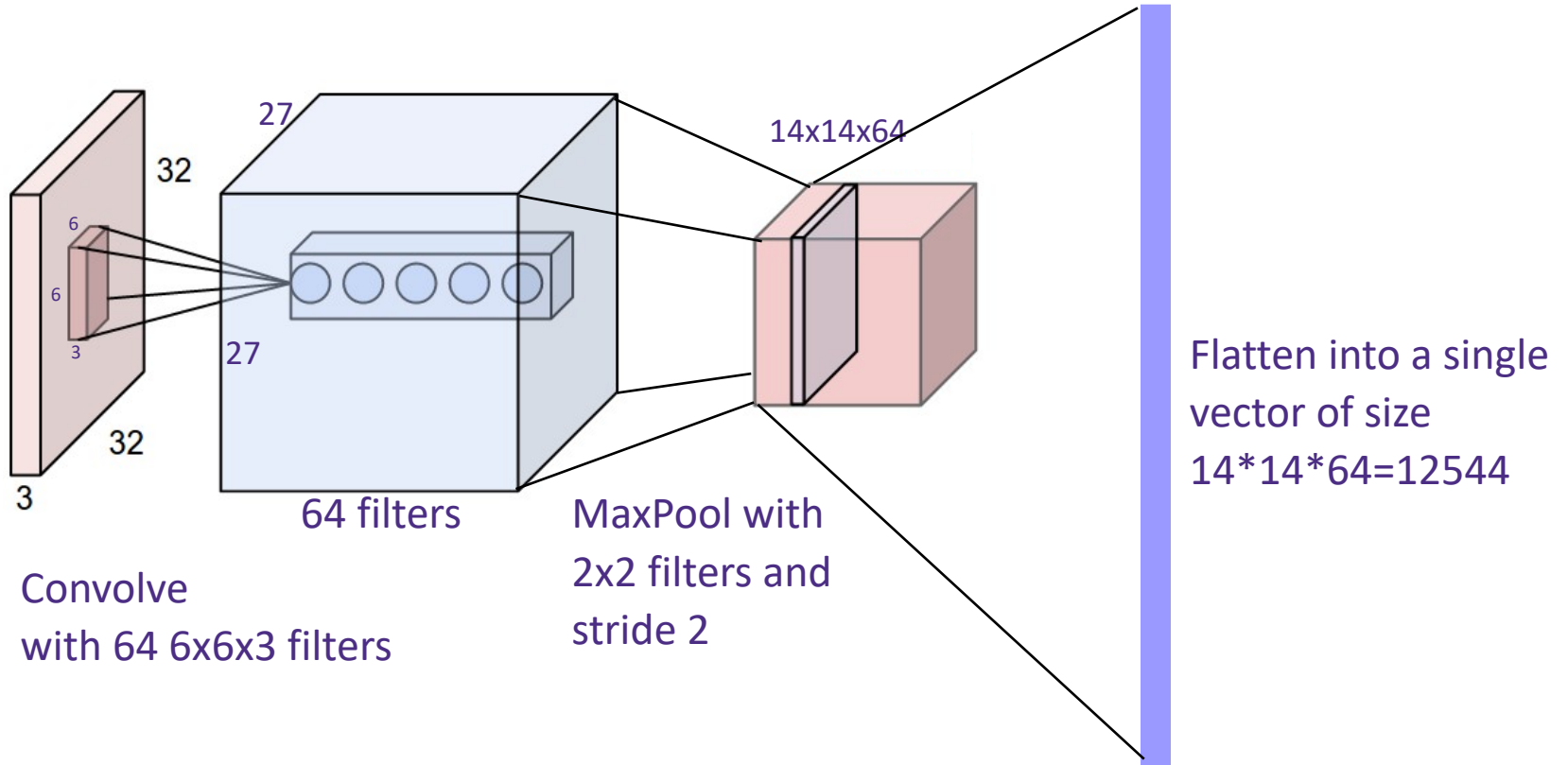




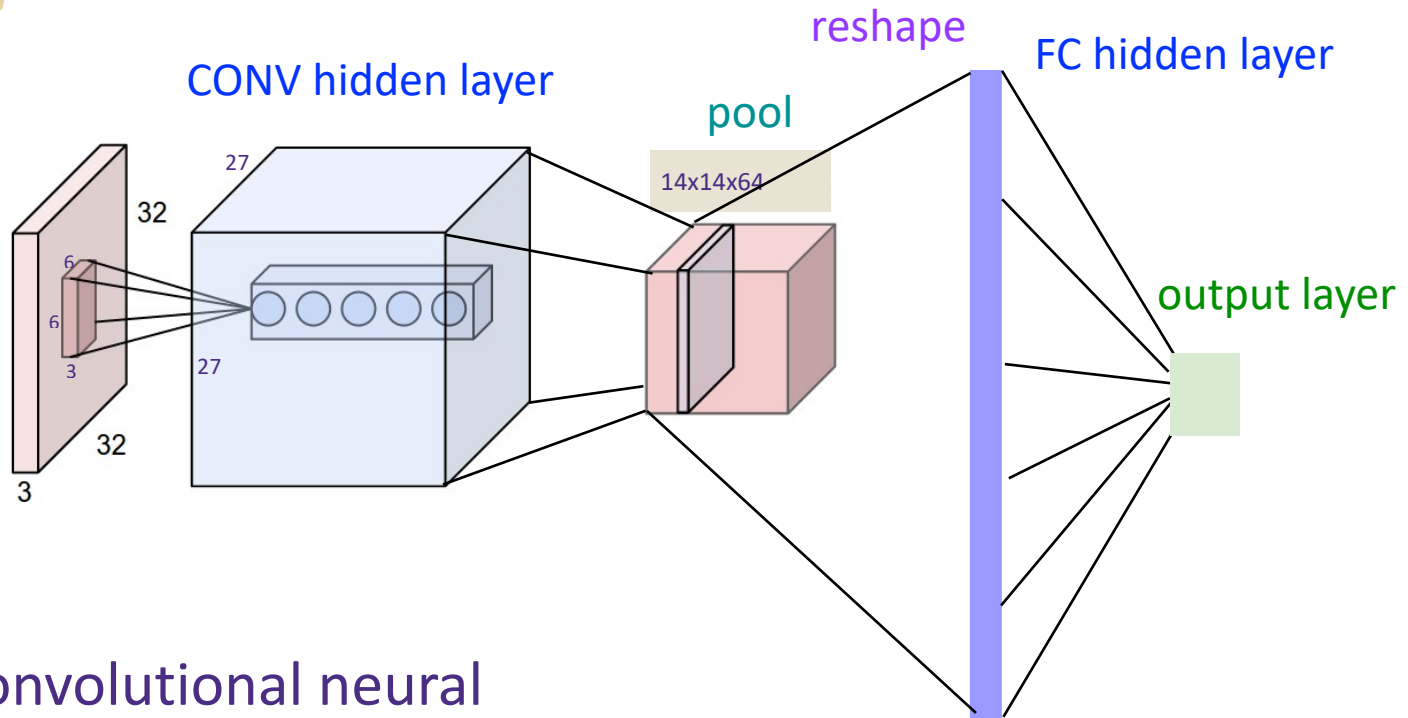
# Pooling Convolution layer



# Flattening



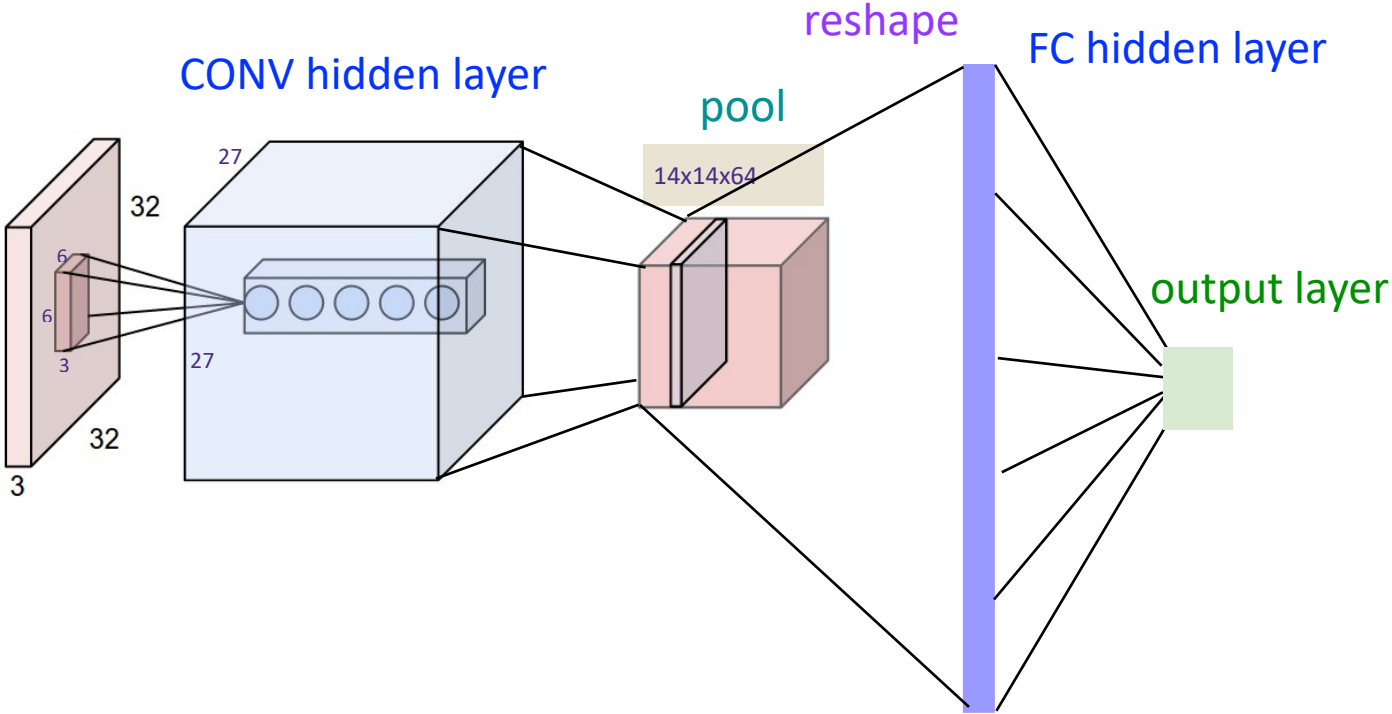
# Training Convolutional Networks



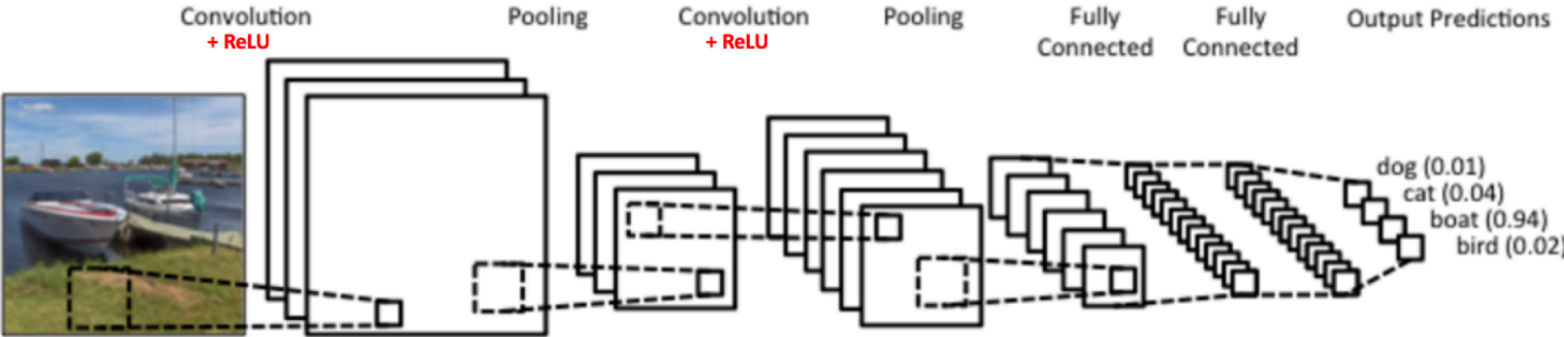
Recall: Convolutional neural networks (CNN) are just regular fully connected (FC) neural networks with some connections removed and some weights shared.

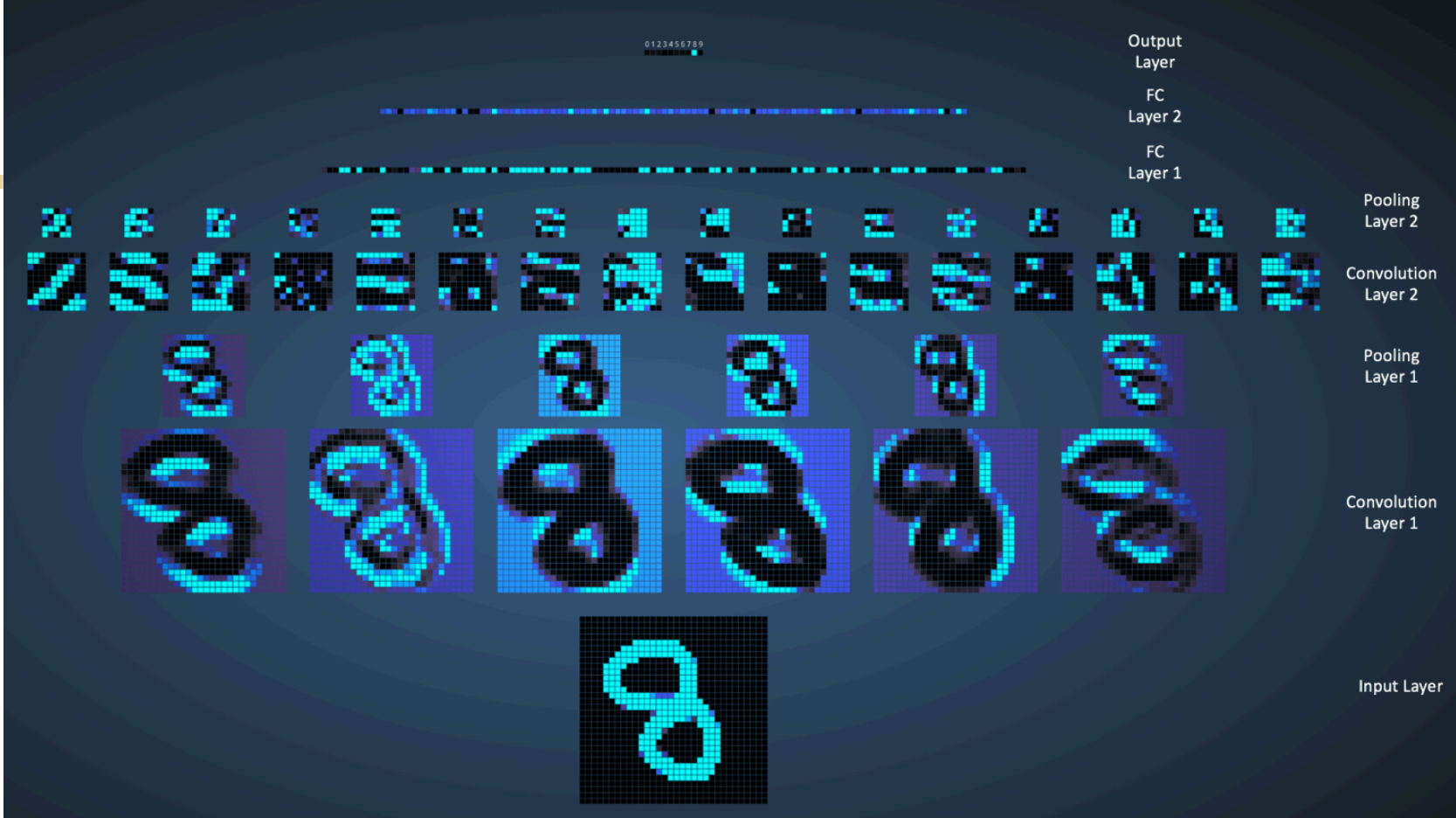
**Train with SGD!**

# Training Convolutional Networks

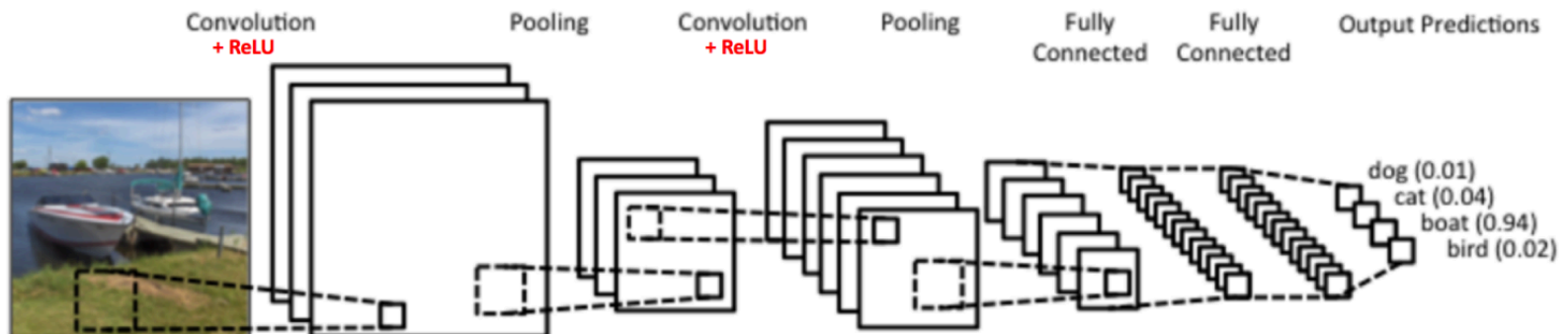


## Real example network: LeNet





Real example network: LeNet

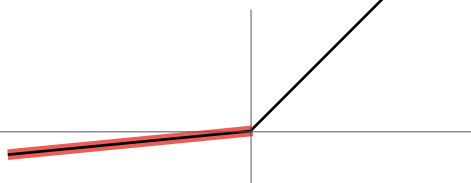


# Real networks

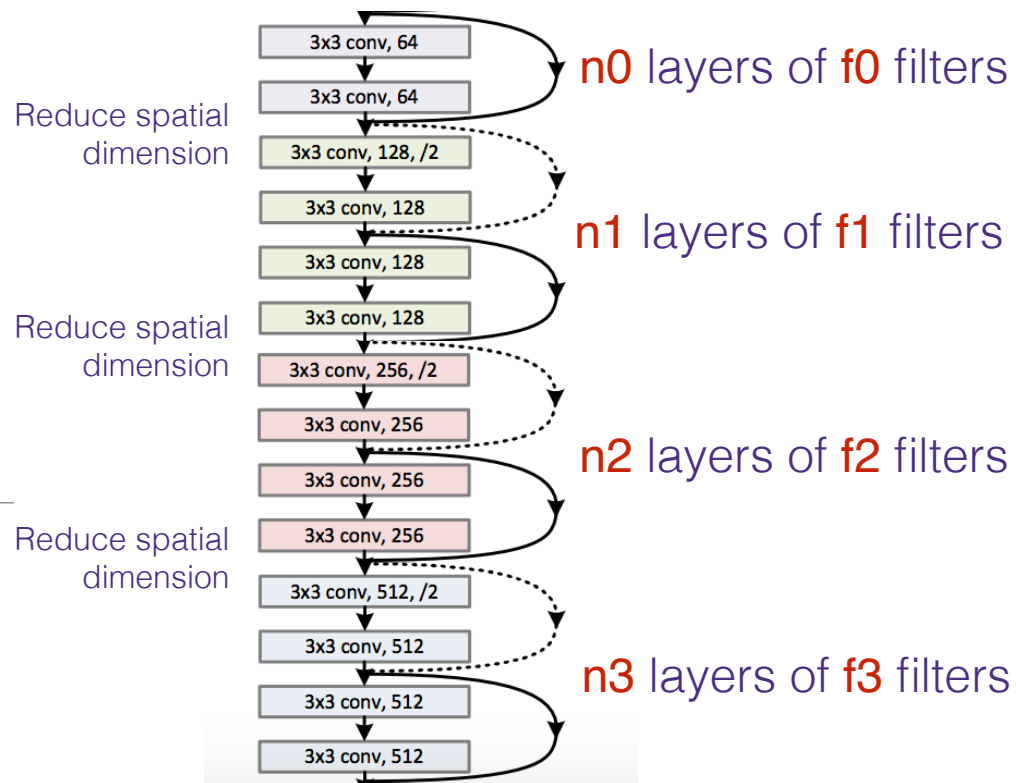
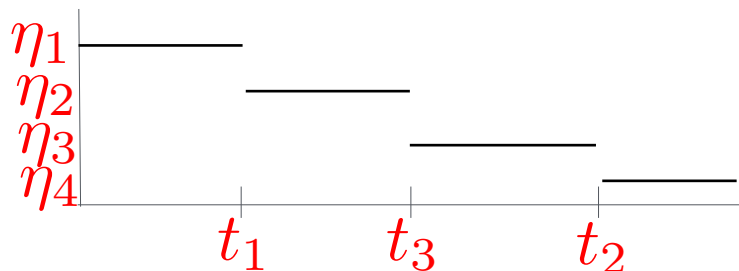
Modern networks have dozens of parameters to tune.

Data augmentation?  
Batch norm?

RELU leakiness  
slope



Learning rate schedule



# Remarks

---

- Convolution is a fundamental operation in signal processing. Instead of hand-engineering the filters (e.g., Fourier, Wavelets, etc.) **Deep Learning *learns* the filters and CONV layers with back-propagation**, replacing fully connected (FC) layers with convolutional (CONV) layers
- **Pooling** is a dimensionality reduction operation that summarizes the output of convolving the input with a filter
- Typically the last few layers are **Fully Connected (FC)**, with the interpretation that the CONV layers are feature extractors, preparing input for the final FC layers. Can replace last layers and retrain on different dataset+task.
- Just as hard to train as regular neural networks.
- More exotic network architectures for specific tasks

# Vision transformers

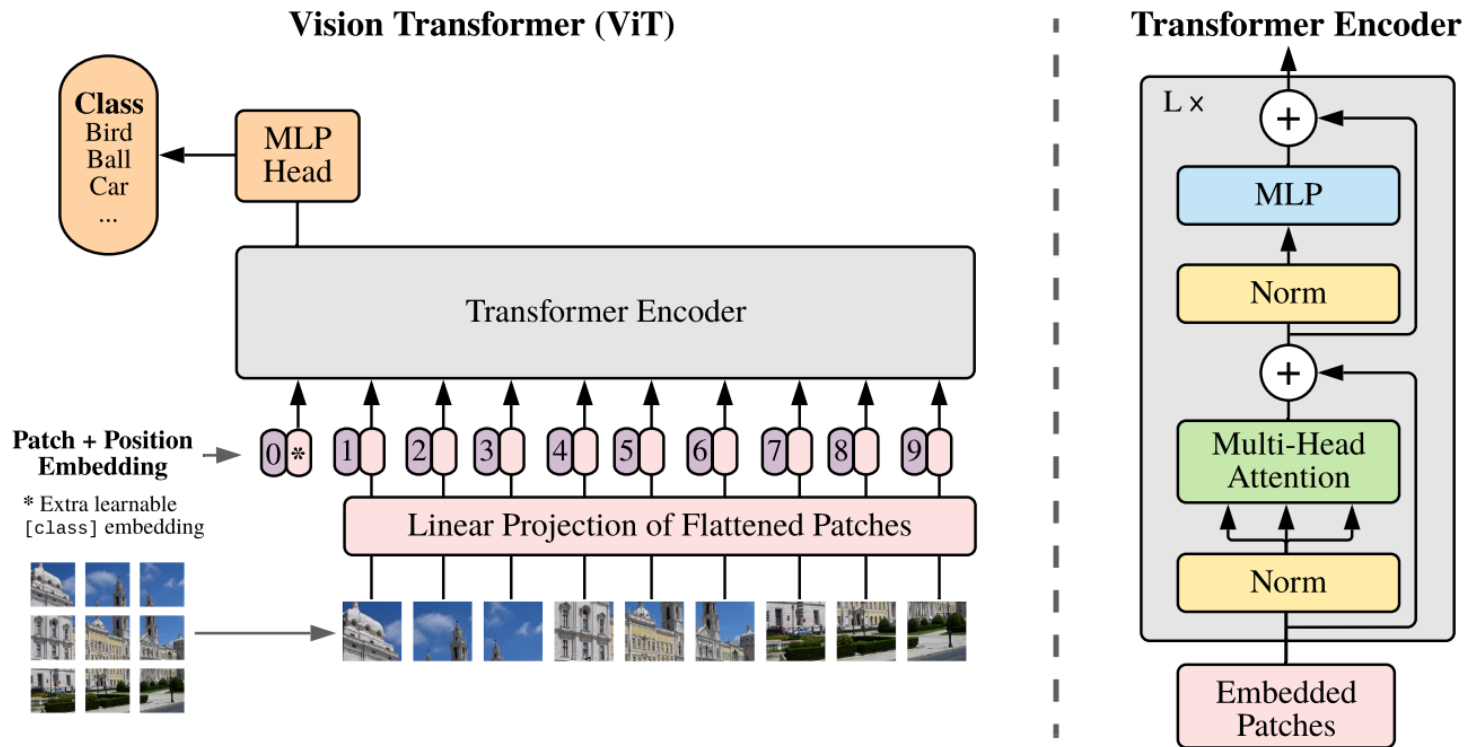


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).



# 2d Convolution Layer

## ■ Example: 200x200 image

- ▶ Fully-connected, 400,000 hidden units = 16 billion parameters
- ▶ Locally-connected, 400,000 hidden units 10x10 fields = 40 million params or channels or filter
- ▶ Local connections capture local dependencies

