Reminder: midterm evaluation

Separation between NN and kernel

- For approximation and optimization, neural network has no advantage over kernel. Why NN gives better performance: generalization.

- [Allen-Zhu and Li '20] Construct a class of functions $\mathscr{F}$ such that $y = f(x)$ for some $f \in \mathscr{F}$:
  - no kernel is sample-efficient;
  - Exists a neural network that is sample-efficient.

Separation between NN and kernel

Defn Kernel method is a linear method with embedding $\phi : \mathbb{R}^d \longrightarrow \mathcal{H}$ Hilbert space

$\Rightarrow$ it turns an element $f \in \mathcal{H}$ into a prediction function

$$\hat{y} = \langle f, \phi(x) \rangle$$

The method uses n samples, $\{x_i\}_{i=1}^n$, $x_i \in \mathbb{R}^d$

observes $\{y_i\}_{i=1}^n$

$$f \in \operatorname{span}\left(\phi(x_i)\right)_{i=1}^n, \quad i \in [n]$$

e.g. $\operatorname*{argmin}_f \frac{1}{n} \sum_{i=1}^n \left(y_i - \langle f, \phi(x_i) \rangle\right)^2 + \lambda \|f\|_{\mathcal{H}}$

n: # of samples

Separation between NN and kernel

Thm $\exists$ a class of functions $\mathcal{C} \subseteq \{c: \mathbb{R}^d \to \mathbb{R}\}$ and a distribution μ over $\mathbb{R}^d$ s.t.

i) $\forall$ kernel method, $\forall c \in \mathcal{C}$, given $y_i = c(x_i)$,

if $\mathbb{E}_{x \sim \mu}\left[\left(c(x) - \langle f, \phi(x) \rangle\right)^2\right] \leq \frac{1}{9}$

then $n \geq 2^{d-1}$ exp large

exp separation

ii) $\exists$ simple procedure s.t. it can output true c as long as $n \geq d$

this procedure can be simulated by a neural network trained by gradient descent

shallow

Separation between NN and kernel

Pf: μ: unif distribution over $\{1,-1\}^d$, 2^d elements

$$\mathcal{C} = \{ C_S = \prod_{s\in S} x_s,\ S \subset \{1,\dots,d\} \}$$

part ii) choose a basis

$$\begin{pmatrix} -1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ \vdots \\ 1 \end{pmatrix}, \dots, \begin{pmatrix} 1 \\ \vdots \\ 1 \\ -1 \end{pmatrix}$$

$e_1 \qquad e_2 \qquad\qquad e_d$

$$x = \underbrace{\begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ \vdots \end{pmatrix}}_{d\text{-dim}}$$

$\Rightarrow$ $y_i = c(e_i)$, if $i \in S \Rightarrow y_i = -1$

$i \notin S \Rightarrow y_i = 1$

$\Rightarrow$ whether i is in S or not

$\Rightarrow$ identify $S \Rightarrow$ recover C_S

Separation between NN and kernel

Part i) $\mathcal{C}$ is a basis for $\{f : \{-1,1\}^d \to \mathbb{R}\}$
with respect to distribution $\mathcal{U}$ by symmetry of $\mathcal{U}$

$$(*)\ \mathbb{E}_{x\sim\mathcal{U}}\left[C_S(x)\cdot C_{S'}(x)\right] = \begin{cases} 0 & \text{if } S \neq S' \\ 1 & \text{if } S = S' \end{cases}$$

$S,\ S' \subset \{1,\dots,d\}$

Goal: to compute

$$\mathbb{E}_{x\sim\mathcal{U}}\left[\left(C_{S^*}(x) - \langle f, \phi(x)\rangle\right)^2\right]$$

since $f \in \mathrm{span}\left(\phi(x_i)\right)_{i=1}^n$

$$\Rightarrow f = \sum_{i=1}^n a_i \phi(x_i),\quad f(x) = \langle f, \phi(x)\rangle = \sum_{i=1}^n a_i \langle \phi(x_i), \phi(x)\rangle$$

$$x \mapsto \langle \phi(x_i), \phi(x)\rangle = \sum_{S\subset[d]} \lambda_{i,S}\cdot C_S(x)$$

$[d] = \{1,\dots,d\}$

Separation between NN and kernel

$$\mathbb{E}_{x\sim\mu}\left[\left(C_{S^*}(x) - \langle f, \phi(x)\rangle\right)^2\right] \quad \text{error}$$

$$= \mathbb{E}_{x\sim\mu}\left[\left(C_{S^*}(x) - \sum_{S\subset[d]}\sum_{i=1}^{q} a_i\cdot\lambda_{i,S}\, C_S(x)\right)^2\right]$$

$$= \left(1 - \sum_{i=1}^{q} a_i\lambda_{i,S^*}\right)^2 + \sum_{S\neq S^*}\left(\sum_{i=1}^{q} a_i\lambda_{i,S}\right)^2$$

(use $\{C_S\}$ is a basis, property (*))

by assumption, error $\leq \frac{1}{q}$

$$\Rightarrow \left(1 - \sum_{i=1}^{q} a_i\lambda_{i,S^*}\right)^2 \leq \frac{1}{q}$$

$$\& \sum_{S\neq S^*}\left(\sum_{i=1}^{q} a_i\lambda_{i,S}\right)^2 \leq \frac{1}{q}$$

show $q \geq 2^{d-1}$ by linear algebra

Separation between NN and kernel

Notations: $\Lambda : 2^d \times n \quad (n \le 2^d)$

$\Lambda_{S,i} = \lambda_{i,S}$

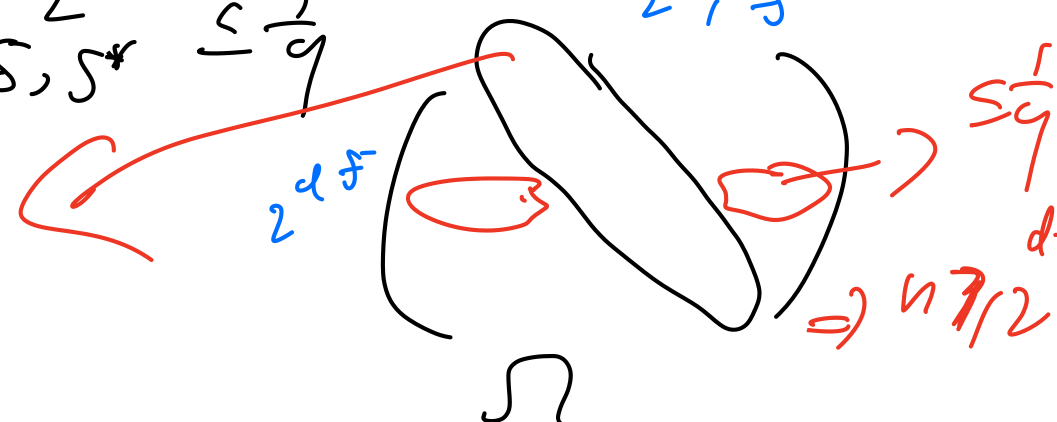
$A : n \times 2^d$

$A_{i,S^*} = a_{i,S^*}, \quad S^* \subset [d]$

$\Omega = \Lambda A : 2^d \times 2^d$ of rank-n

diagonal $\quad (1 - \Omega_{S^*,S^*}) \le \frac{1}{9} \longrightarrow \Omega^2_{S^*,S^*} \ge \frac{4}{9}$

off-diagonal $\quad \sum_{S \ne S^*} \Omega^2_{S,S^*} \le \frac{1}{9}$



Separation between NN and kernel

$\Omega = \mathrm{diag}(\Omega) + \Omega'$, Ω': off-diagonal

$\|\Omega'\|_F^2 \le \frac{2^d}{9}$

$= \sum \mathrm{eig}^2(\Omega')$

$\Rightarrow$ Ω' has at most $\frac{2^d}{4}$ eigenvalues $\geq \frac{2}{3}$

$\Rightarrow$ consider subspace of Ω' with eigenvalue $< \frac{2}{3}$ which has dimension at least $\frac{3}{4} \cdot 2^d$

$\forall x \in$ this space

$$\|\Omega x\|_2 = \|(\mathrm{diag}(\Omega) + \Omega')x\|_2$$
$$\geq \|\mathrm{diag}(\Omega)x\|_2 - \|\Omega' x\|_2$$
$$> \frac{2}{3}\|x\|_2 - \frac{2}{3}\|x\|_2 = 0$$

$\Rightarrow$ $\mathrm{rank}(\Omega) \geq \frac{3}{4} \cdot 2^d$ $\Rightarrow$ $n \geq \frac{3}{4} \cdot 2^d \geq 2^{d-1}$ $\square$

Convolutional Neural Networks

Multi-layer Neural Network

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

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

$$a^{(2)} = g\left(z^{(2)}\right)$$

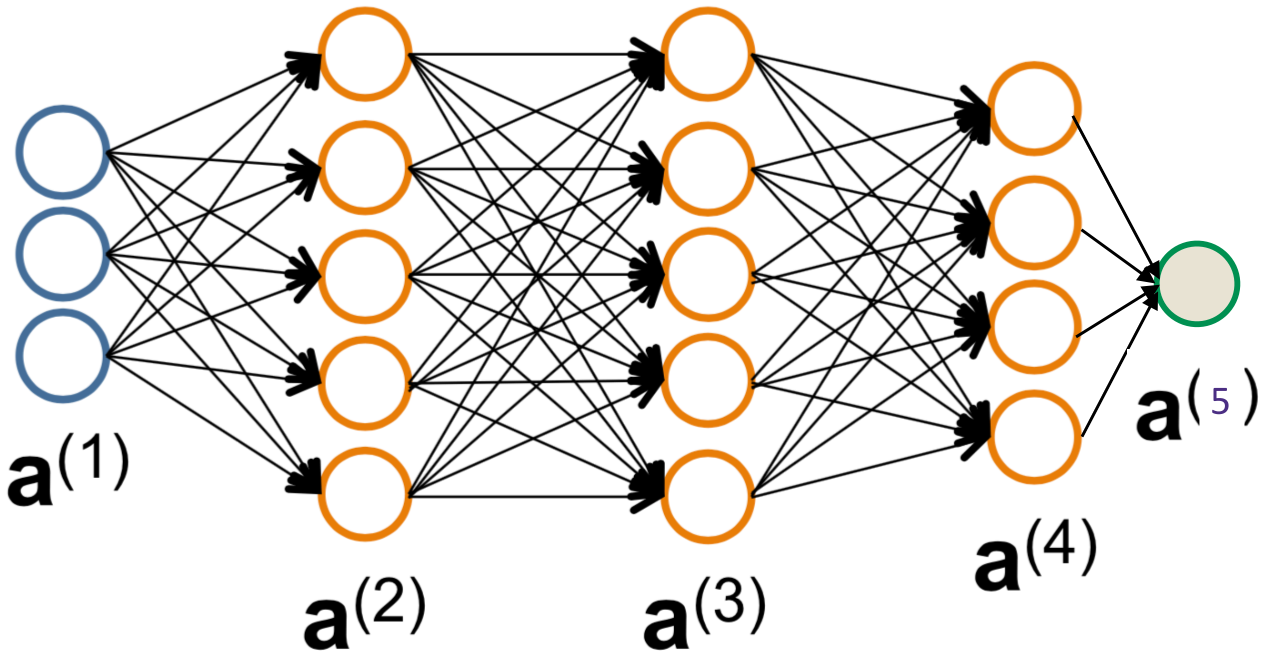
$$\vdots$$

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

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

$$\vdots$$

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



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

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

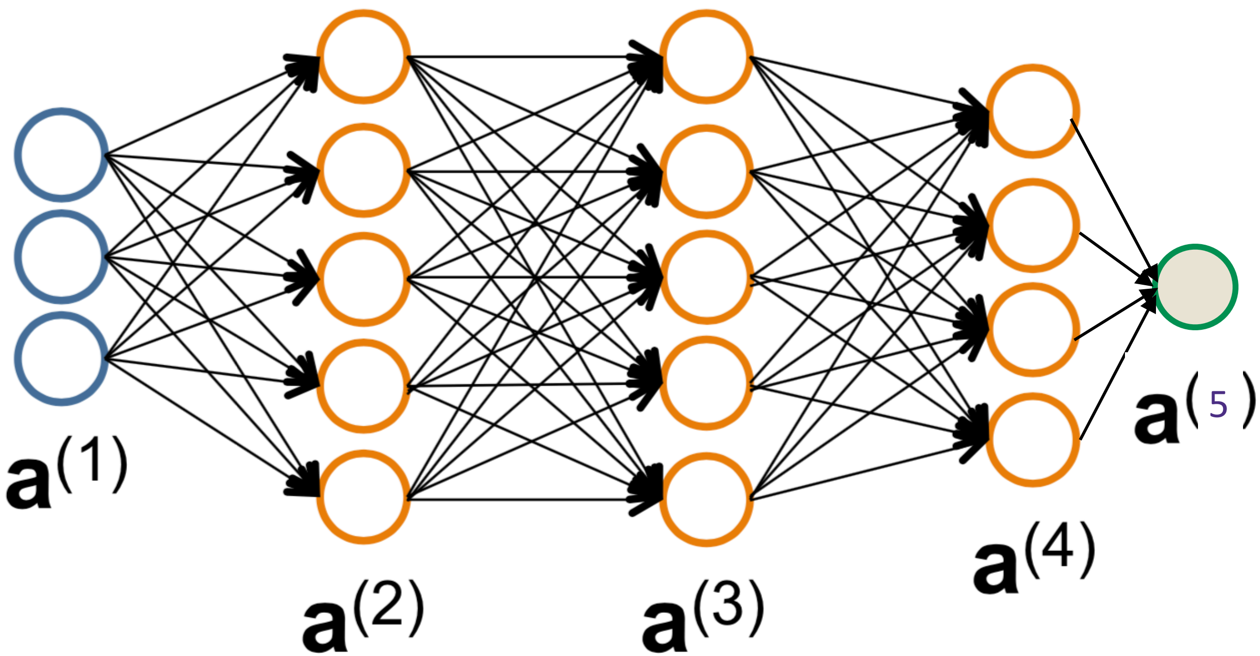
Binary
Logistic
Regression

Neural Network Architecture

depth

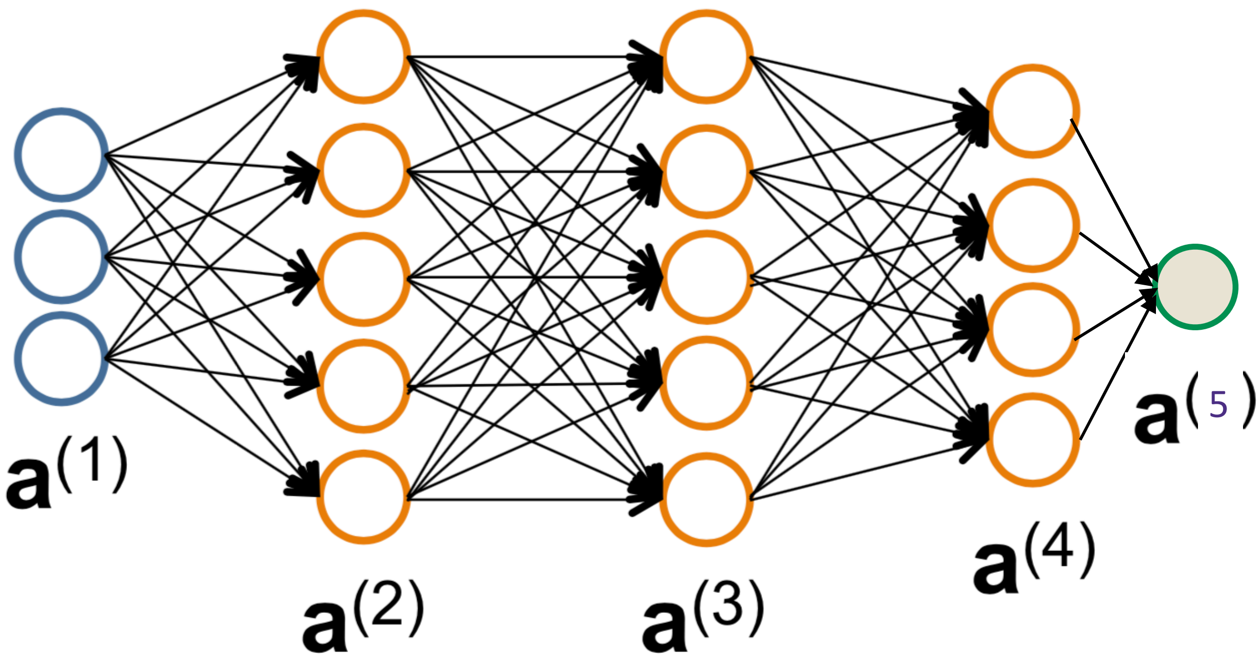
The neural network architecture is defined by the number of layers, and the number of nodes in each layer, but also by **allowable edges**.

width



Neural Network Architecture

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



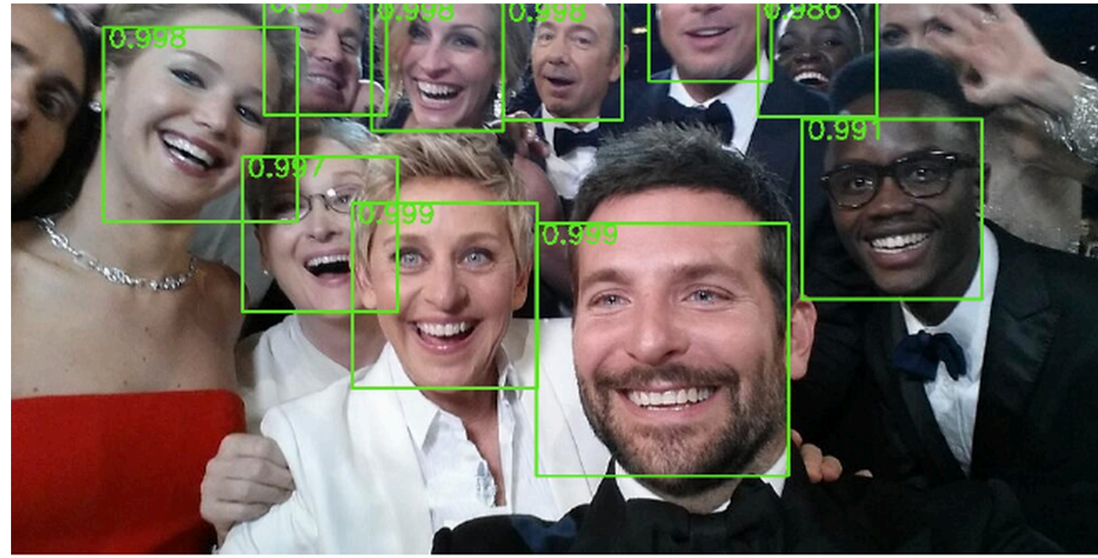
We say a layer is **Fully Connected (FC)** if all linear mappings from the current layer to the next layer are permissible.

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

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

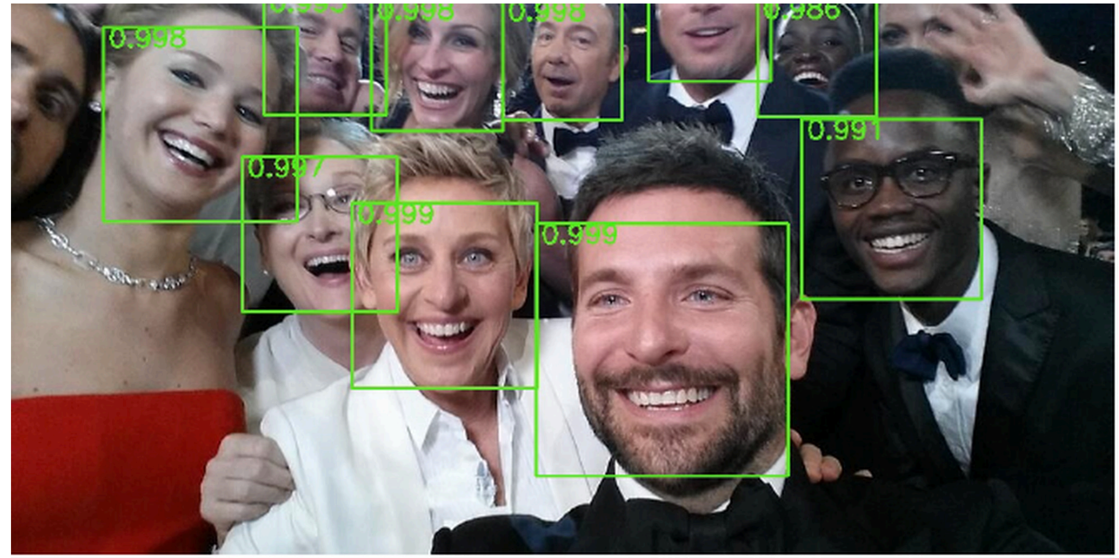
Neural Network Architecture

Objects 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.

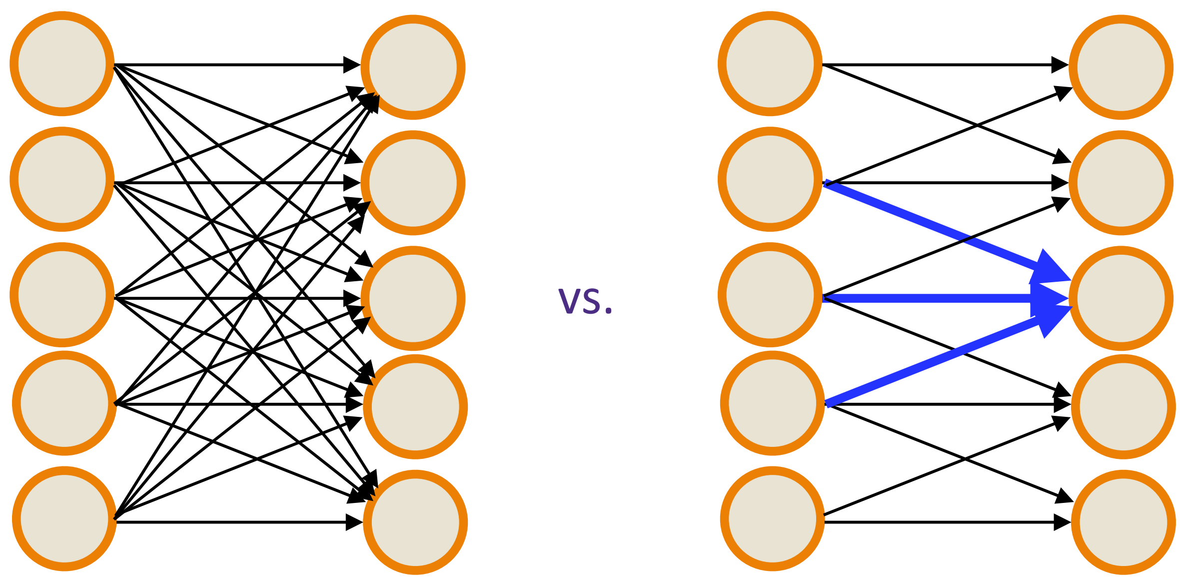


Neural Network Architecture

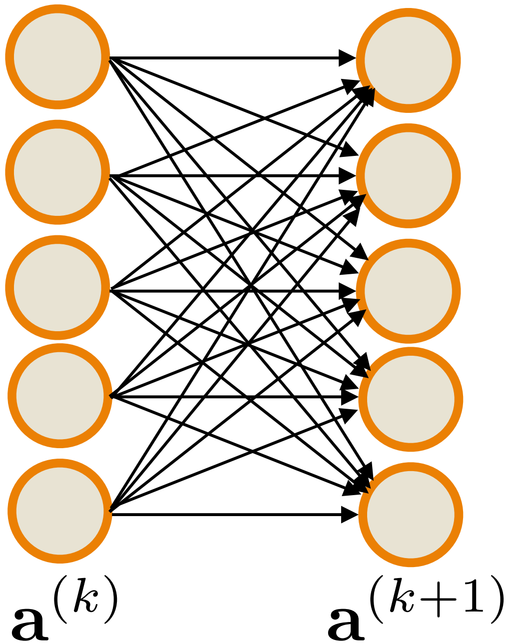
Objects are often **localized in space** so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.



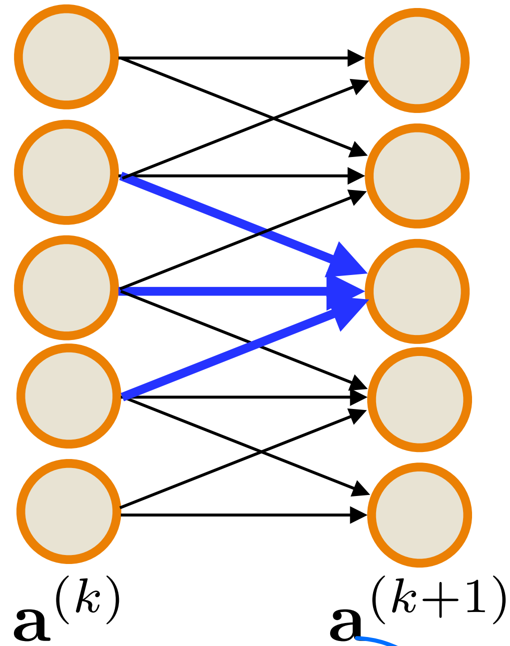
Similarly, to identify edges or other local structure, it makes sense to only look at **local information**



Neural Network Architecture



vs.



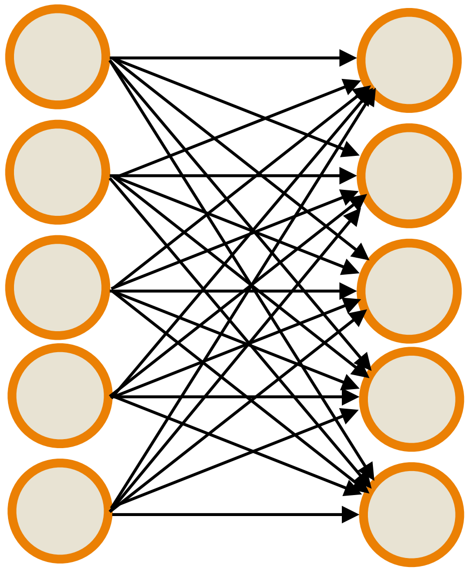
hardcode

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

Parameters: n^2 $\qquad$ $3n-2$

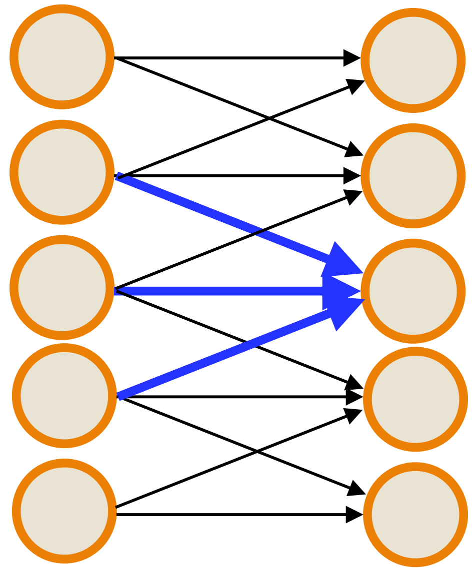
$$\mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{n-1} \Theta_{i,j} \mathbf{a}_j^{(k)}\right)$$

Neural Network Architecture



$\mathbf{a}^{(k)}$ $\mathbf{a}^{(k+1)}$

vs.



$\mathbf{a}^{(k)}$ $\mathbf{a}^{(k+1)}$

Mirror/share local weights everywhere (e.g., structure equally likely to be anywhere in image)

$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix} \quad \begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & 0 & 0 & 0 \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & 0 & 0 \\ 0 & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & 0 \\ 0 & 0 & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ 0 & 0 & 0 & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix} \quad \begin{bmatrix} \theta_1 & \theta_2 & 0 & 0 & 0 \\ \theta_0 & \theta_1 & \theta_2 & 0 & 0 \\ 0 & \theta_0 & \theta_1 & \theta_2 & 0 \\ 0 & 0 & \theta_0 & \theta_1 & \theta_2 \\ 0 & 0 & 0 & \theta_0 & \theta_1 \end{bmatrix}$$

Parameters: n^2 $\quad 3n-2$ $\quad 3$

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

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)

$$\begin{bmatrix} \theta_1 & \theta_2 & 0 & 0 & 0 \\ \theta_0 & \theta_1 & \theta_2 & 0 & 0 \\ 0 & \theta_0 & \theta_1 & \theta_2 & 0 \\ 0 & 0 & \theta_0 & \theta_1 & \theta_2 \\ 0 & 0 & 0 & \theta_0 & \theta_1 \end{bmatrix}$$

m=3

$$\mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{m-1} \theta_j \mathbf{a}_{i+j}^{(k)} \right) = g([\theta * \mathbf{a}^{(k)}]_i)$$

Convolution*

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

Example (1d convolution)

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

stride = 1

1	1	1	0	0

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

1	0	1

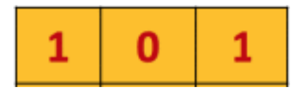
Filter $\theta \in \mathbb{R}^m$

Output $\theta * x$

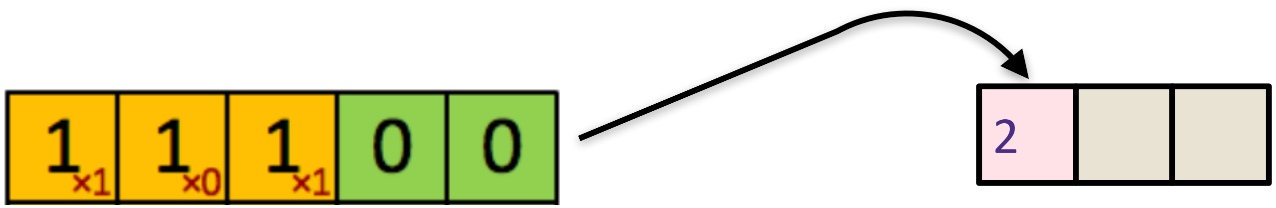
Example (1d convolution)

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

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



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

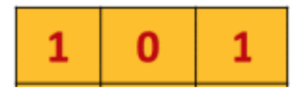


Output $\theta * x$

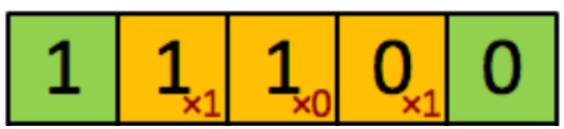
Example (1d convolution)

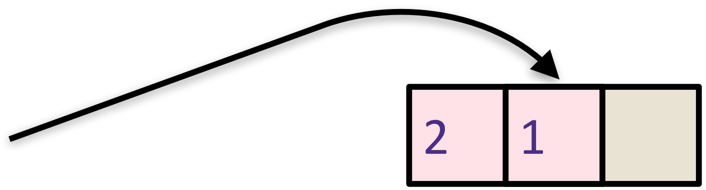
Input $x \in \mathbb{R}^n$

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



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





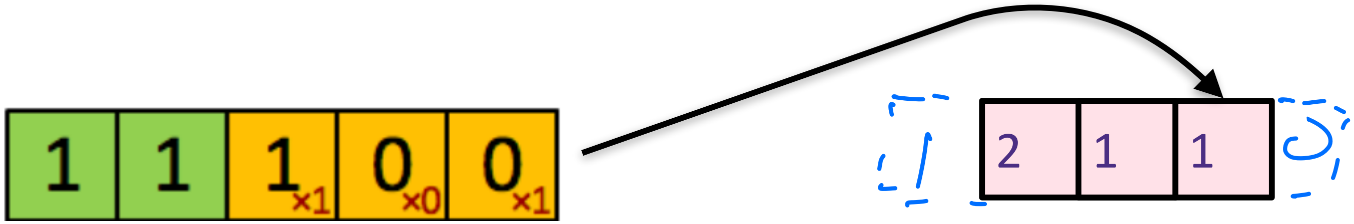
Output $\theta * x$

Example (1d convolution)

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

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

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

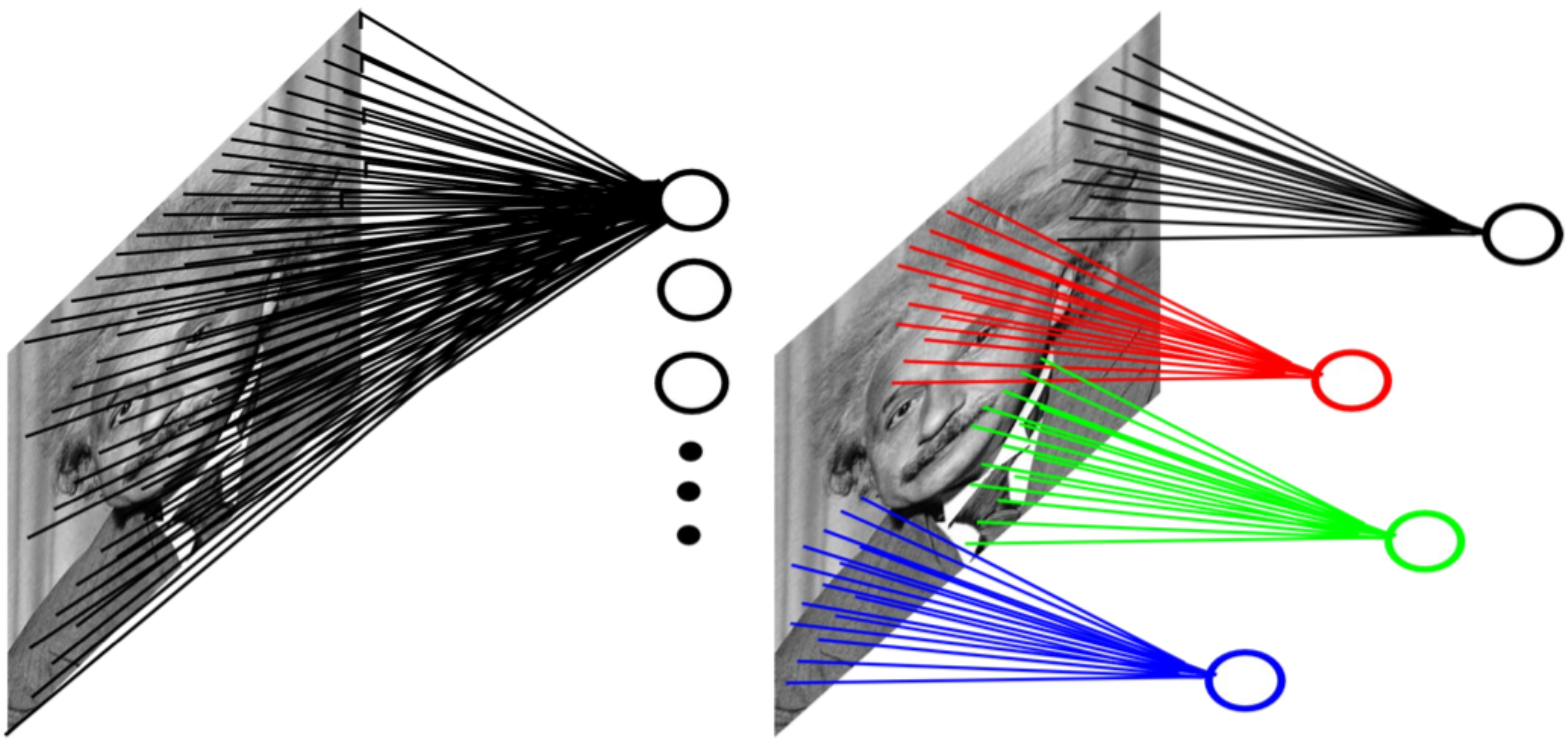


Output $\theta * x$

stride =1

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
  - Local connections capture local dependencies



Convolution of images (2d convolution)

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

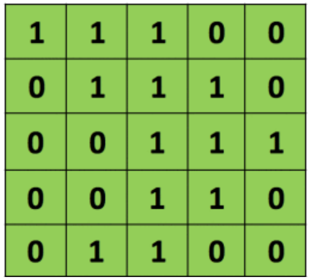
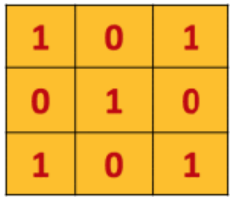
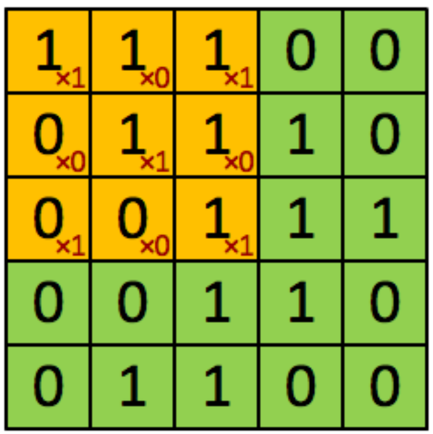


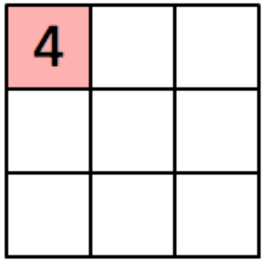
Image I



Filter K



Image



Convolved Feature

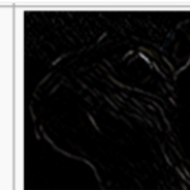
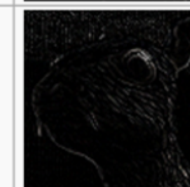
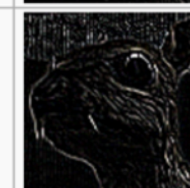
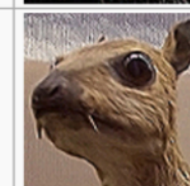
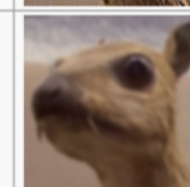
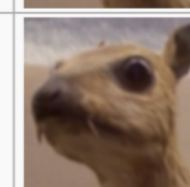
$I * K$

Convolution of images

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

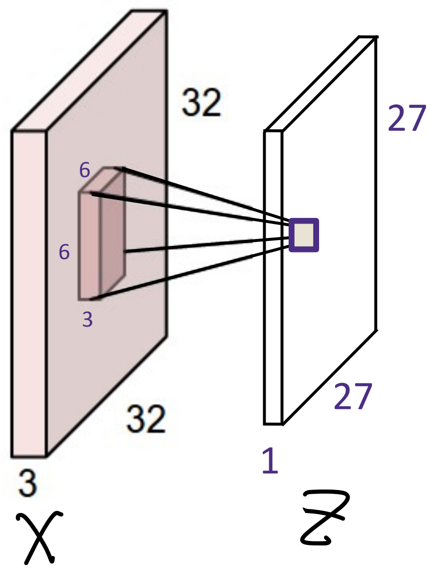
Image I

NN: learn

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

$K: k \times k \times r$

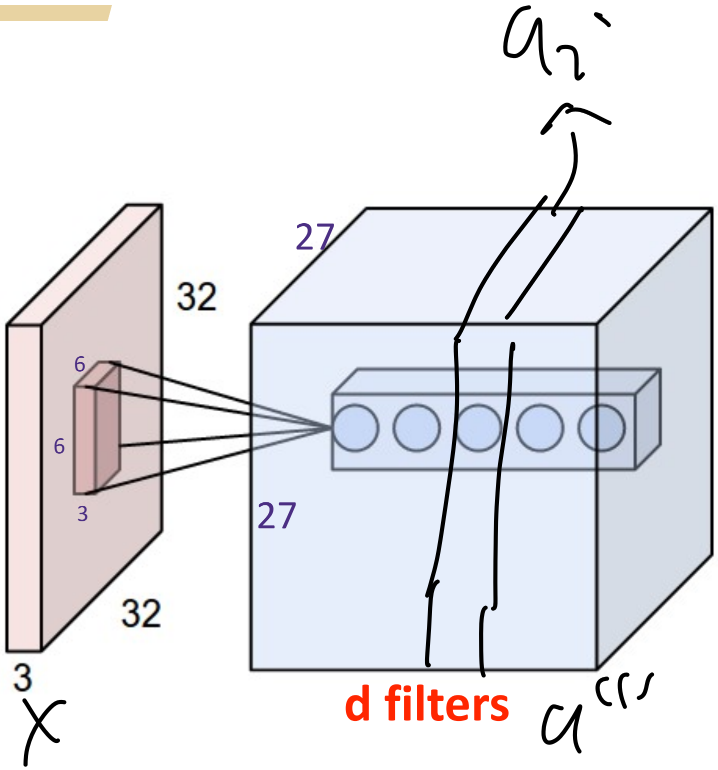


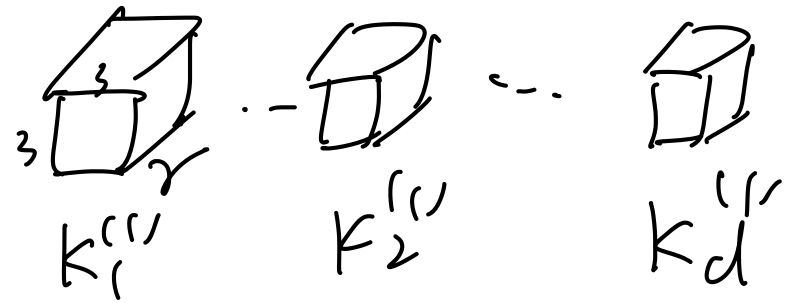
$$x \in \mathbb{R}^{n \times n \times r}$$

R G B

$$Z = \sum_{\alpha=1}^{r} X[:,:,\alpha] * K[:,:,\alpha]$$

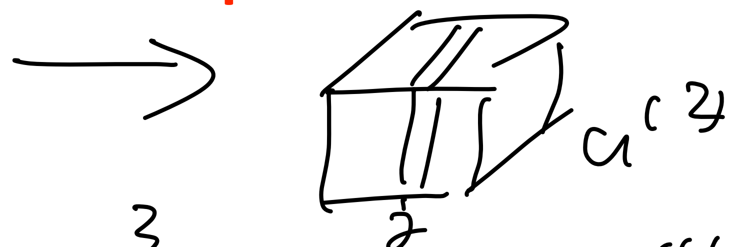
Stacking convolved images





$\{K_j^{(2)}\}_{j=1}^{d}$

Repeat with d filters!



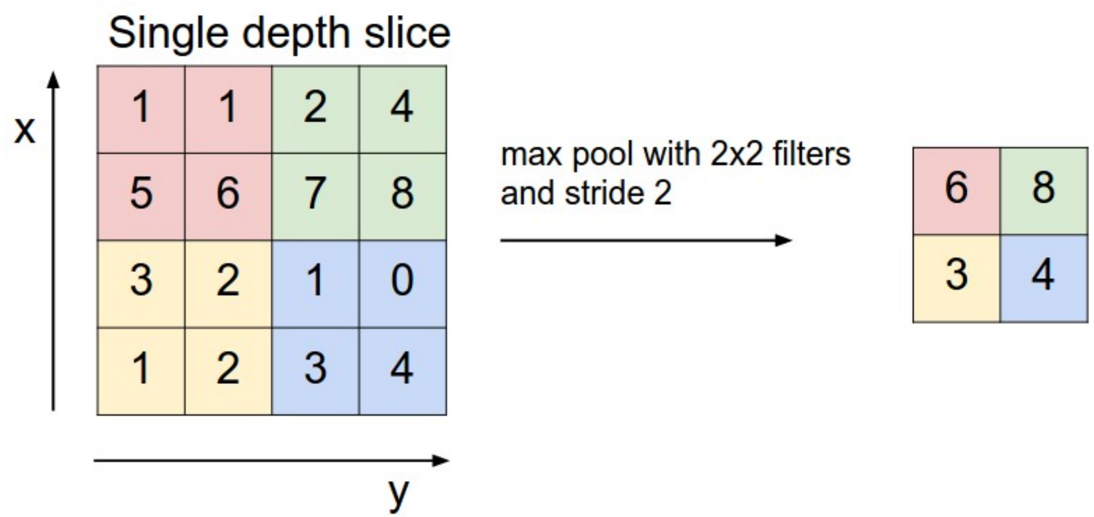
$$a_i^{(1)} = g\left(\sum_{\alpha=1}^{3} X[:,:,\alpha] * K_i^{(1)}[:,:,\alpha]\right)$$

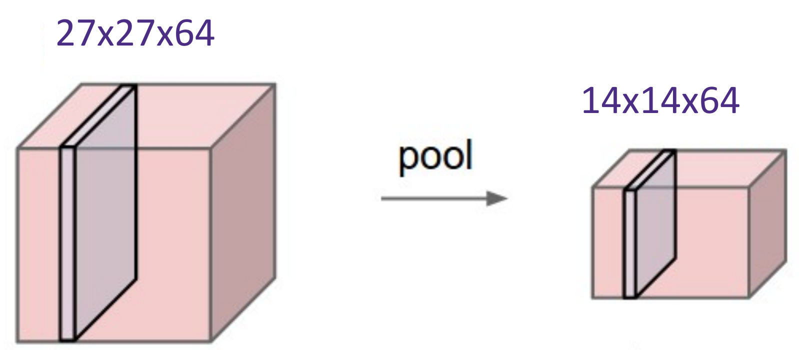
$K_j^{(2)}: k \times k \times d$

$$a_j^{(2)} = g\left(\sum_{i=1}^{d} a^{(1)}[:,:,i] * K_j^{(2)}[:,:,i]\right)$$

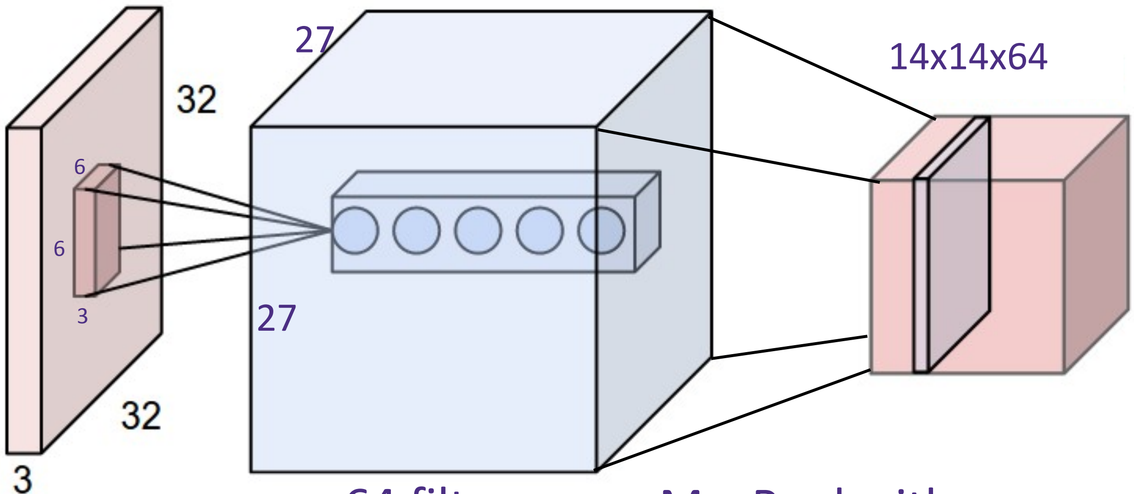
Pooling

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





Pooling Convolution layer

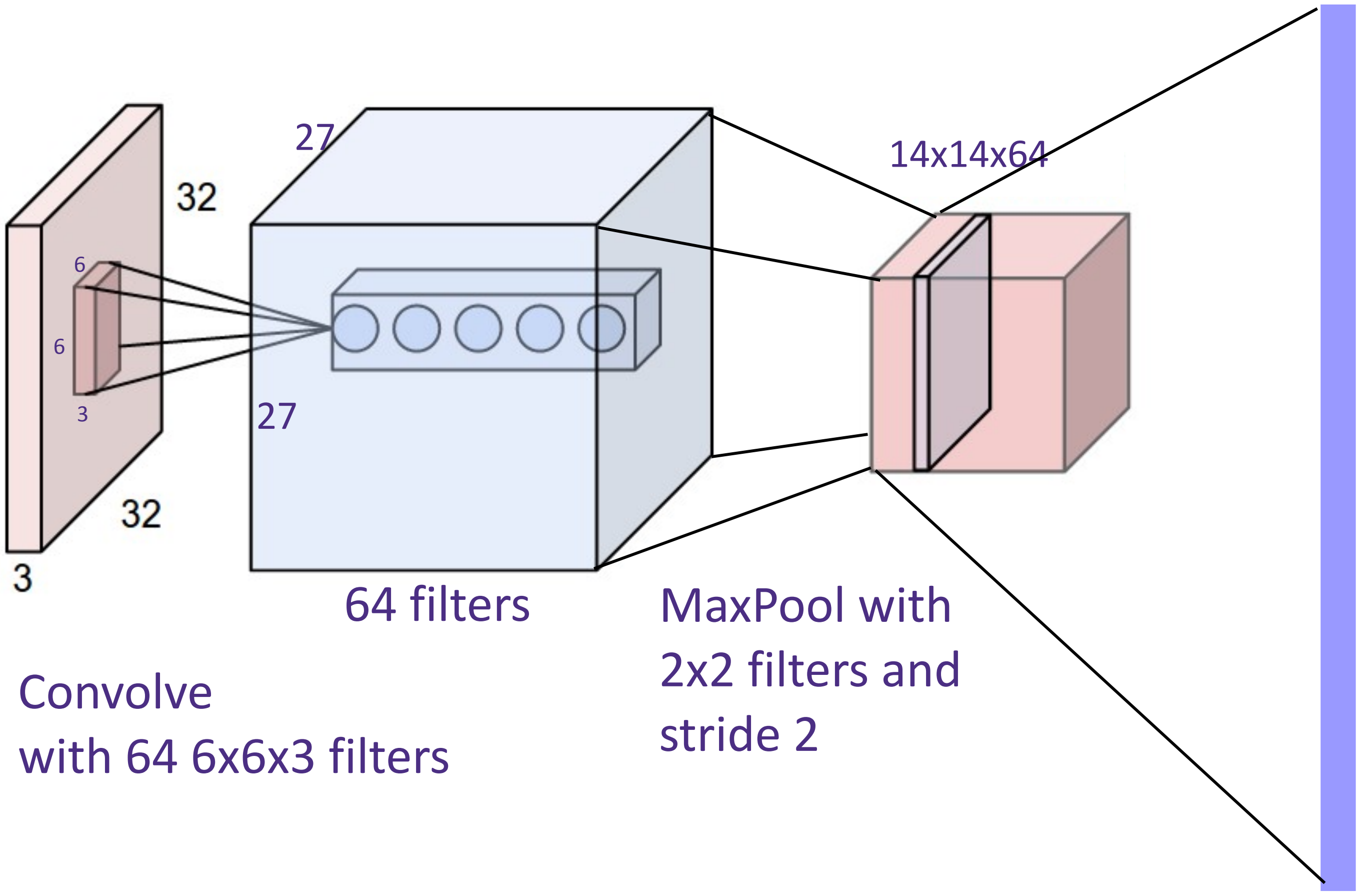


64 filters

MaxPool with 2x2 filters and stride 2

Convolve with 64 6x6x3 filters

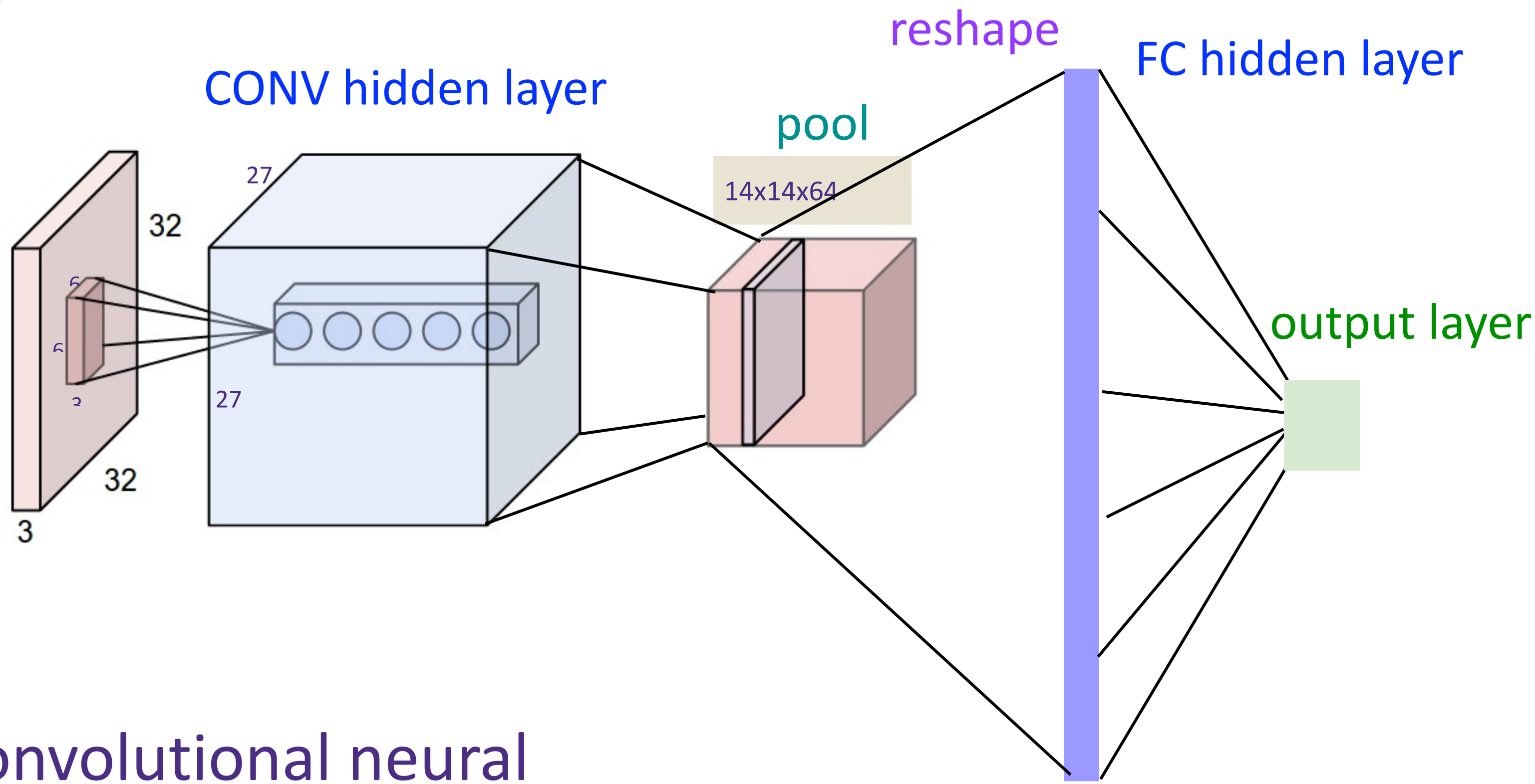
Flattening



Convolve
with 64 6x6x3 filters

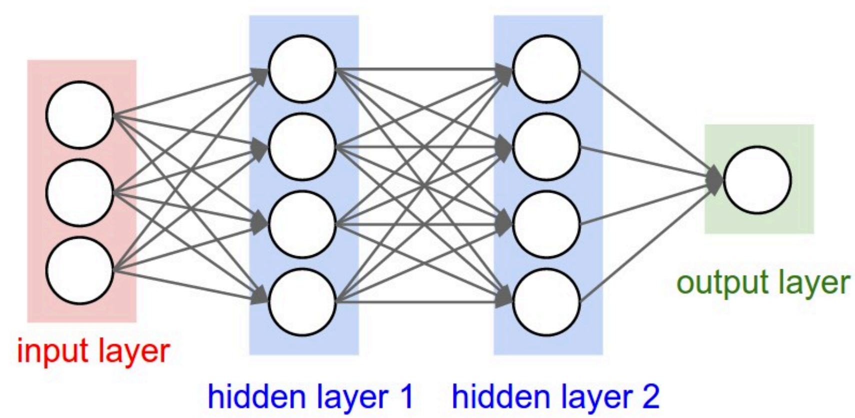
Flatten into a single
vector of size
14*14*64=12544

Training Convolutional Networks

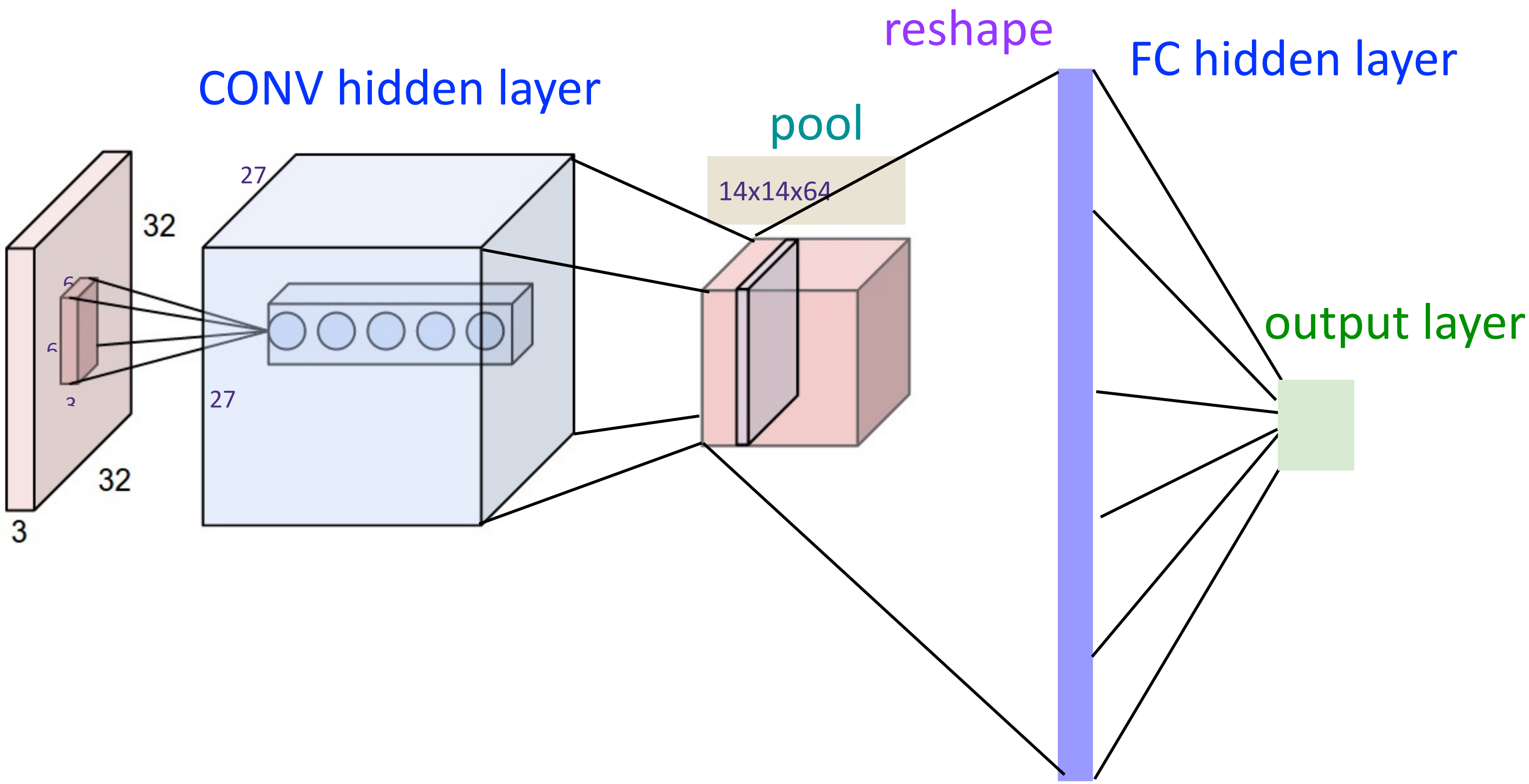


Recall: Convolutional neural networks (CNN) are just regular fully connected (FC) neural networks with some connections removed.

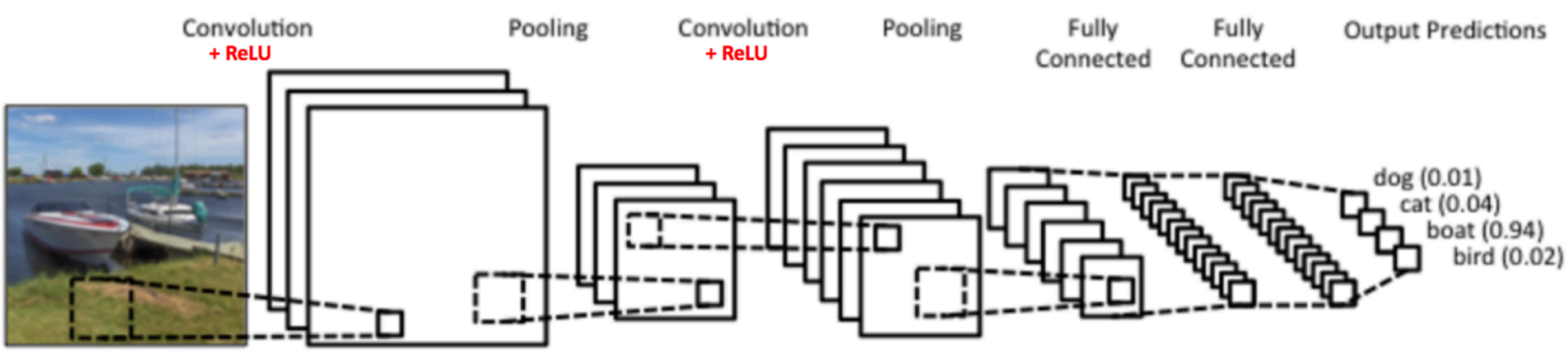
Train with SGD!

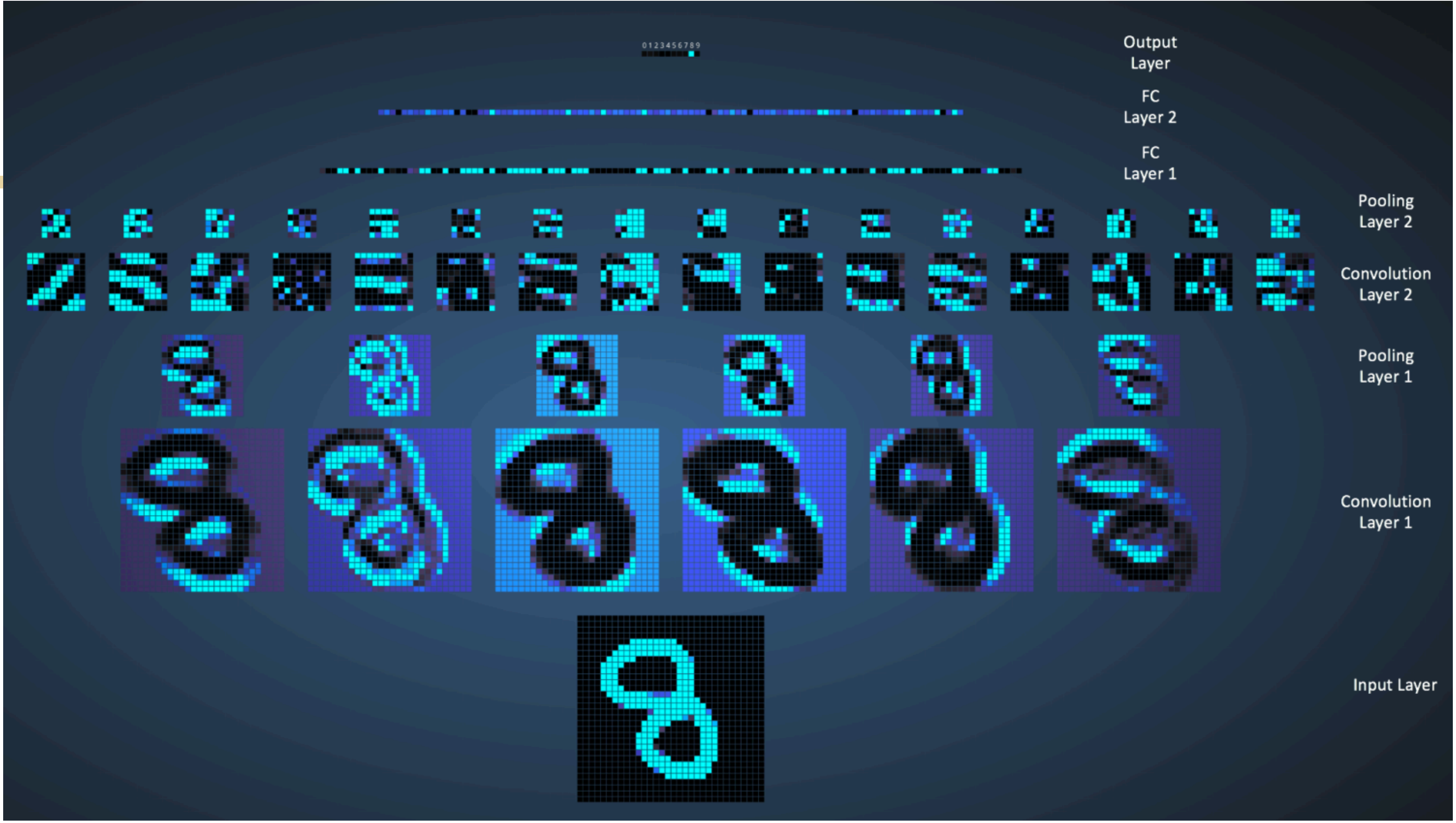


Training Convolutional Networks



Real example network: LeNet





Real example network: LeNet