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; for all kernel
  - Exists a neural network that is sample-efficient.

Separation between NN and kernel

<u>Defn</u> Kernel method is a linear method with an embedding $\phi: \mathbb{R}^d \to \mathcal{H}$ Hilbert space

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

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

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

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

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

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 $\mathbb{E}_{x \sim \mu}\left[(c(x) - \langle f, \phi(x) \rangle)^2\right] \le \frac{1}{4}$

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

ii) $\exists$ simple procedure s.t. it can output

the true c as long as $n \geq d$

this procedure can be simulated/approximated

by NN + GD

Separation between NN and kernel

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

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

part ii) choose a basis: $\underbrace{\begin{pmatrix} -1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}}_{e_1}, \underbrace{\begin{pmatrix} 1 \\ -1 \\ \vdots \\ 1 \end{pmatrix}}_{e_2}, \dots, \underbrace{\begin{pmatrix} 1 \\ \vdots \\ 1 \\ -1 \end{pmatrix}}_{e_d}$

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

$\notin S \Rightarrow y_i = 1$

$\Rightarrow$ know whether i is in S or not

$\Rightarrow$ identify $S \Rightarrow C_S$

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

Separation between NN and kernel

Part i) $\mathcal{C}$ is a basis for $\{f : \{-1,1\}^d \to \mathbb{R}\}$

with distribution

by symmetry

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

Goal: to compute

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

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

$$\Rightarrow f = \sum_{i=1}^n a_i \phi(x_i)$$

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

Separation between NN and kernel

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

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

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

by assumption error $\leq \frac{1}{4}$

$$\Rightarrow \left(1 - \sum_{i=1}^{n} a_i \lambda_{i,S^*}\right)^2 \leq \frac{1}{4}$$

$$\& \sum_{S \neq S^*}\left(\sum_i a_i \lambda_{i,S}\right)^2 \leq \frac{1}{4}$$

$$\Longrightarrow n \geq 2^{d-1}$$

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

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

$A : n \times 2^d$

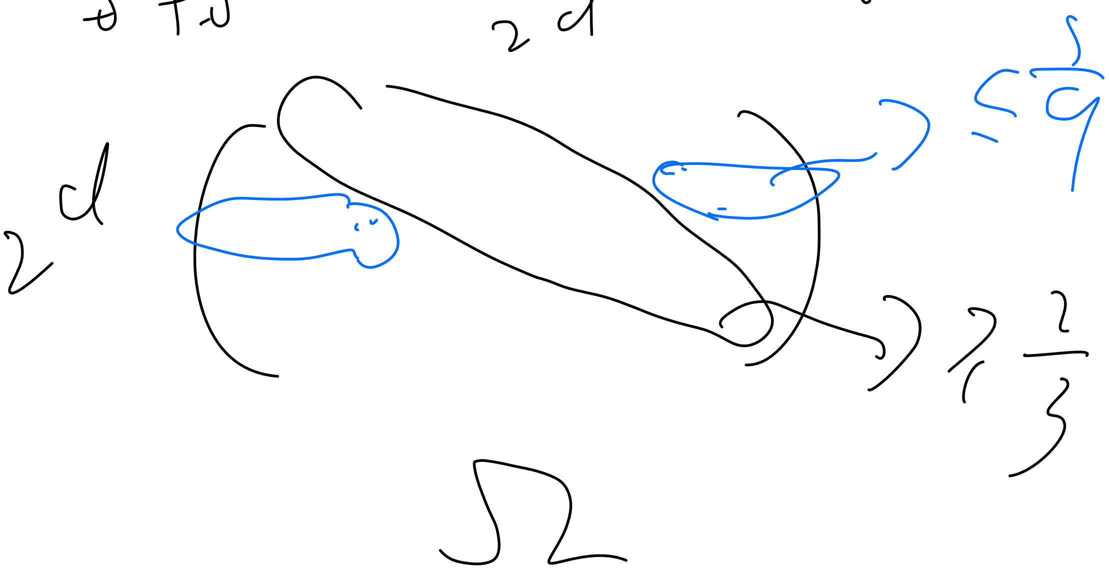
$A_{i,s^*} = a_{i,s^*}$

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

(1) $(1 - \Omega_{s^*,s^*})^2 \le \frac{1}{9} \rightarrow \Omega_{s^*,s^*} \ge \frac{2}{3}$

$$\sum_{s \ne s^*} \Omega^2_{s,s^*} \le \frac{1}{9}$$

2^d



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

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

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

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

$\Rightarrow$ Consider subspace with eigenvalues $< \frac{2}{3}$, which has dimension at least $\frac{3}{4} \cdot 2^d$

$\forall x \in$ this subspace

$\|\Omega x\|_2 = \|\mathrm{diag}(\Omega) x + \Omega' x\|_2$

$\geq \|\mathrm{diag}(\Omega) x\|_2 - \|\Omega' x\|_2$

$> \frac{2}{3} x - \frac{2}{3} x = 0$

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

Convolutional Neural Networks

Multi-layer Neural Network

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

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

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

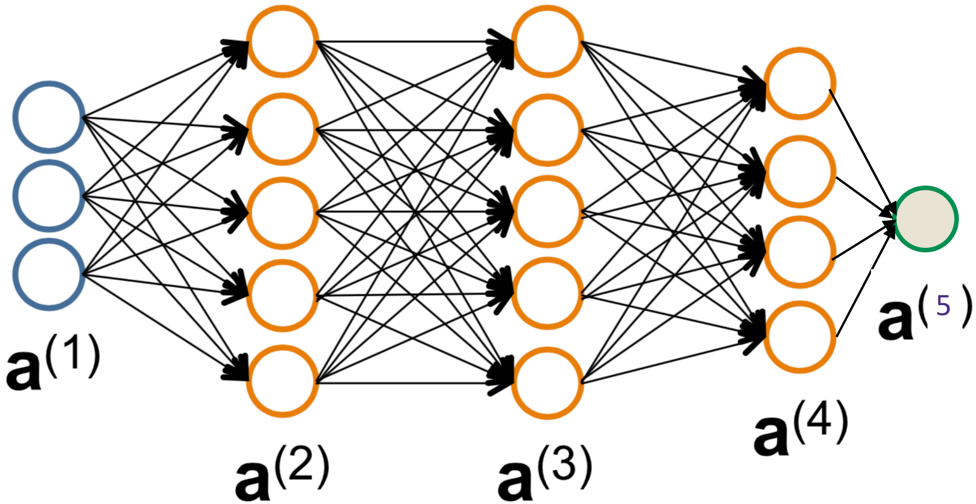
$$\vdots$$

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

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

$$\vdots$$

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



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

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

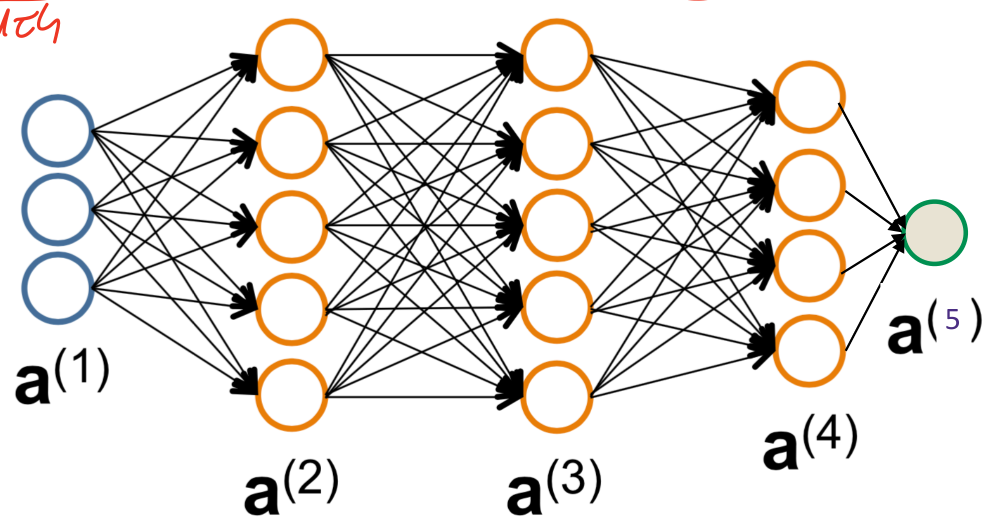
Binary
Logistic
Regression

Neural Network Architecture

depth

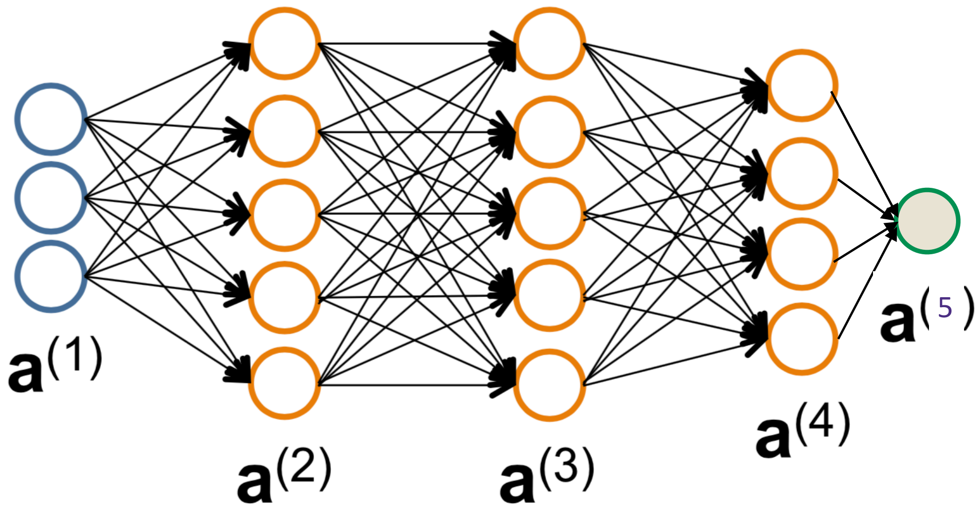
The neural network architecture is defined by the number of layers, and the number of nodes in each layer, but also by **allowable edges**.

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**.



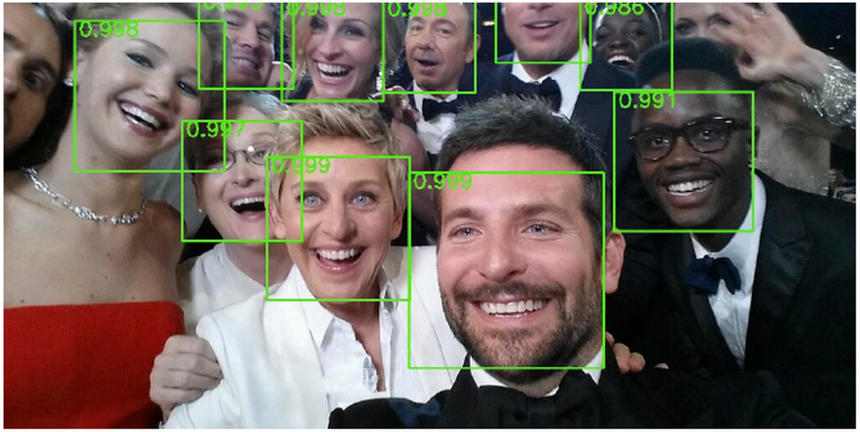
We say a layer is **Fully Connected (FC)** if all linear mappings from the current layer to the next layer are permissible.

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

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

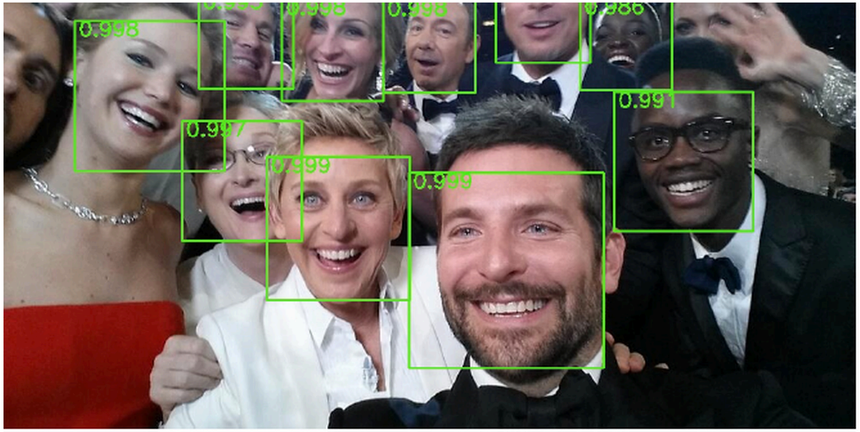
Neural Network Architecture

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.

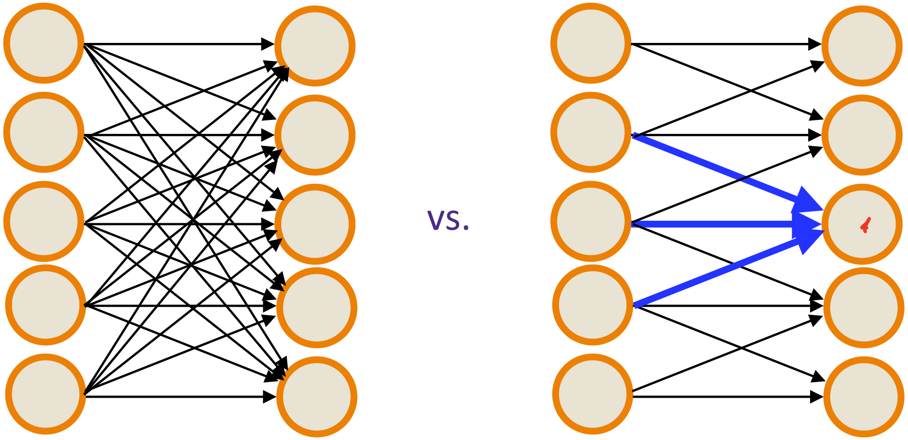


Neural Network Architecture

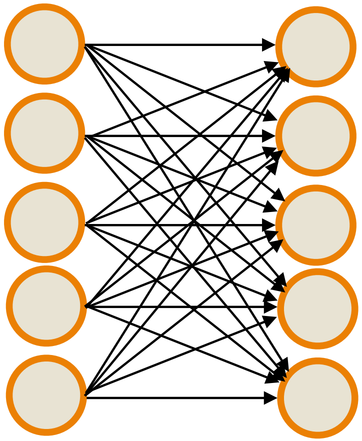
Objects are often **localized in space** so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.



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

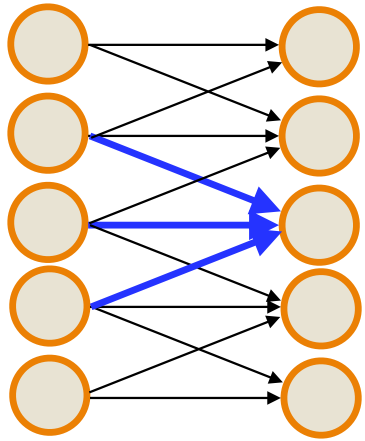


Neural Network Architecture



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

vs.



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

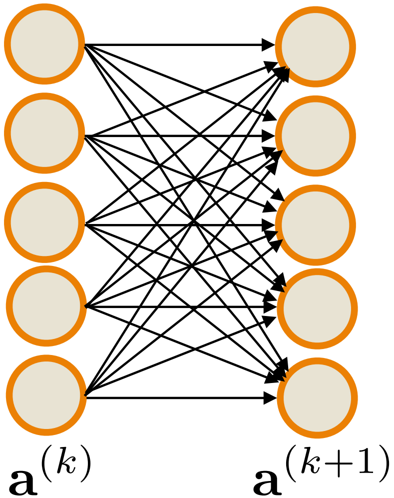
hardcode

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

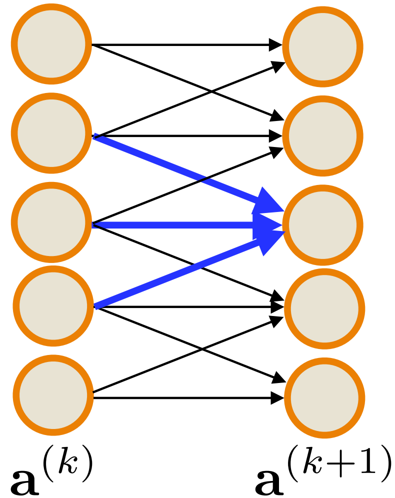
Parameters: n^2 $3n-2$

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

Neural Network Architecture



vs.



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 $\qquad 3n-2$ $\qquad 3$

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

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)

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

1	1	1	0	0

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

1	0	1

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

Output $\theta * x$

$\in \mathbb{R}^{n-2}$

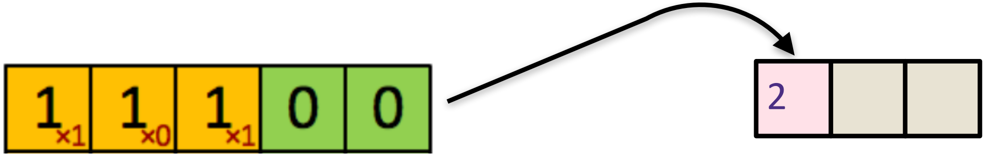
stride = 1

Example (1d convolution)

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

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

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



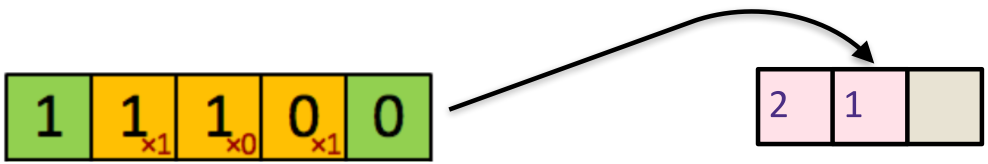
Output $\theta * x$

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 $\theta * x$

Example (1d convolution)

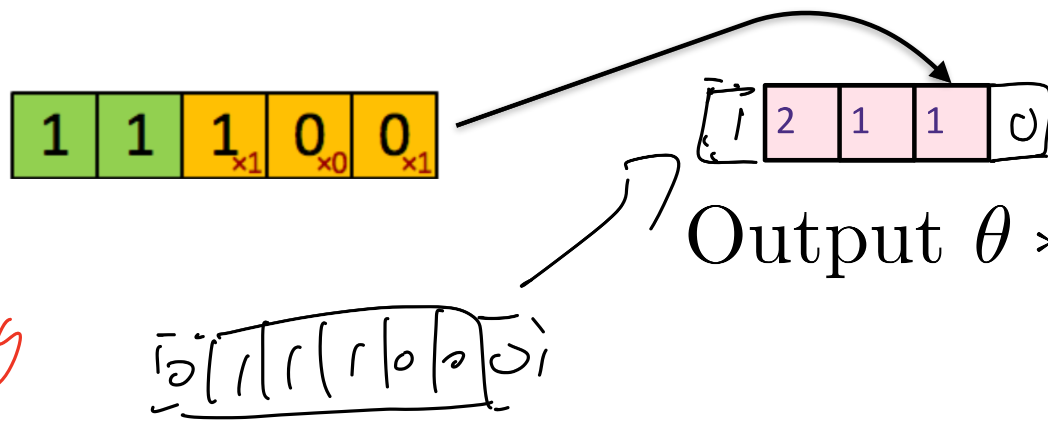
1	1	1	0	0

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

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

1	0	1

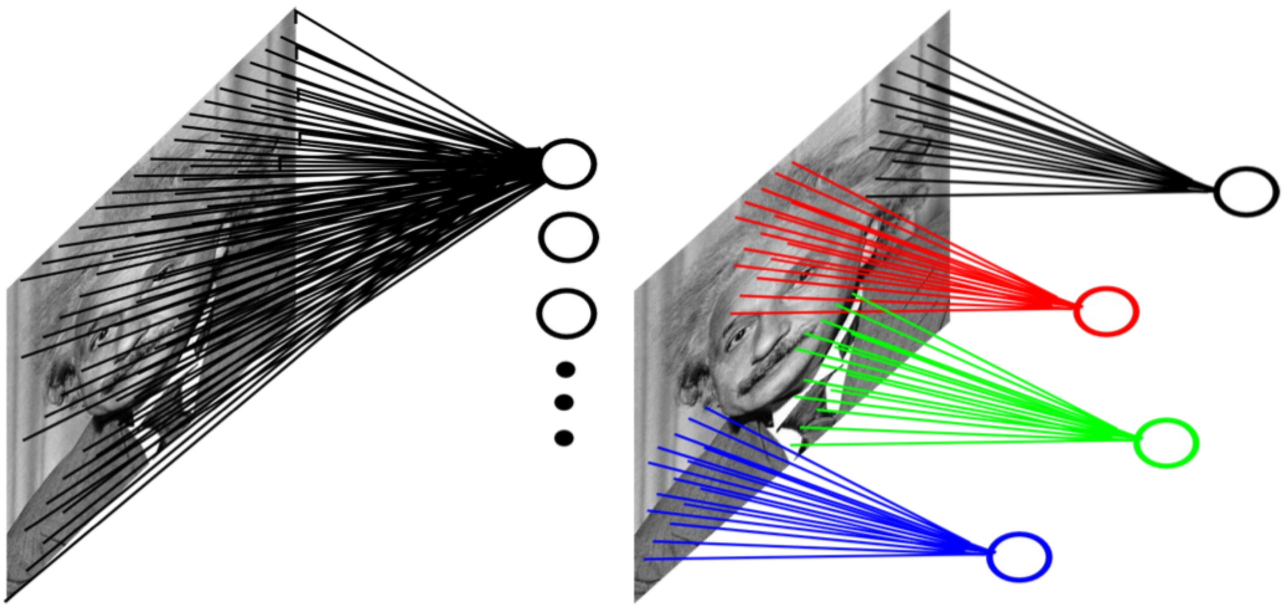
Filter $\theta \in \mathbb{R}^m$



Padding

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)

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

n

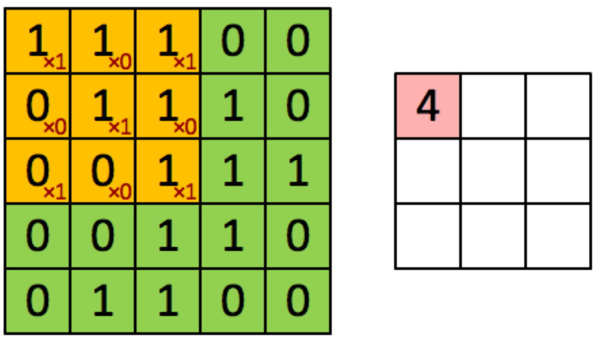
n

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



Image

Convolved Feature

$I * K$

$(n-2) \cdot (n-2)$

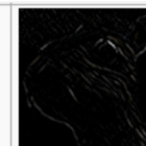
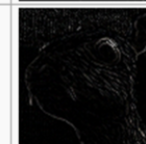
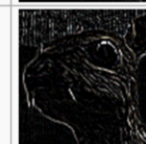
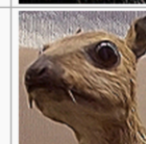
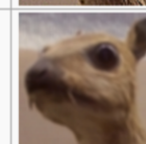
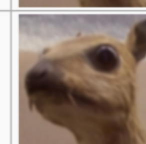
Convolution of images

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

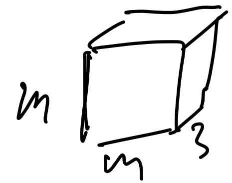
Image I

hand
crafted

NN : learned

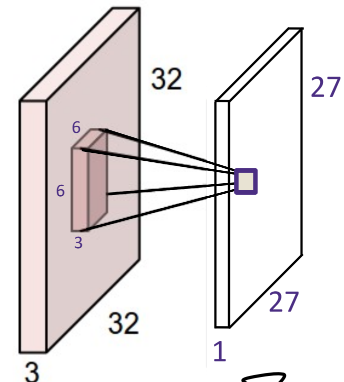
Operation	Filter K	Convolved Image $I * K$
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
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

$r = 3$



X $\quad Z$

channel

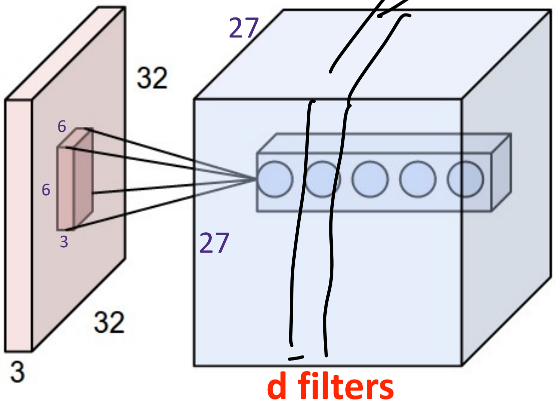
$$x \in \mathbb{R}^{n \times n \times \underline{r}}$$

R G B

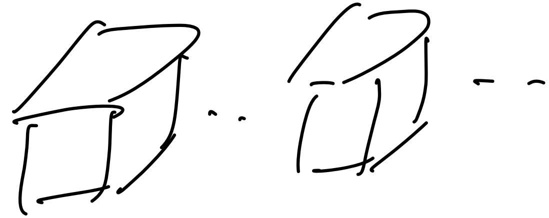
$$Z = \sum_{\alpha=1}^{r} X[:,:,\alpha] * K[:,:,\alpha]$$

Stacking convolved images

one filter

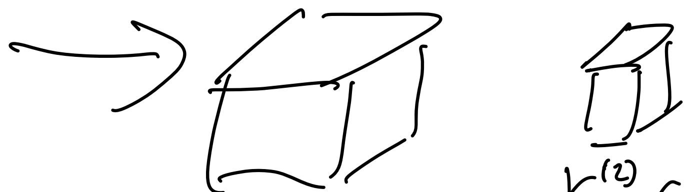


X $a^{(1)}$



K_1 K_d

Repeat with d filters!



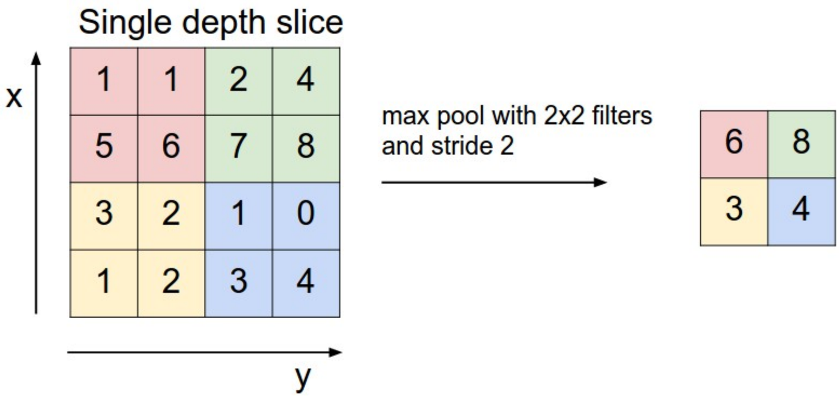
$K_j^{(2)} \in \mathbb{R}^{h \times h \times d}$

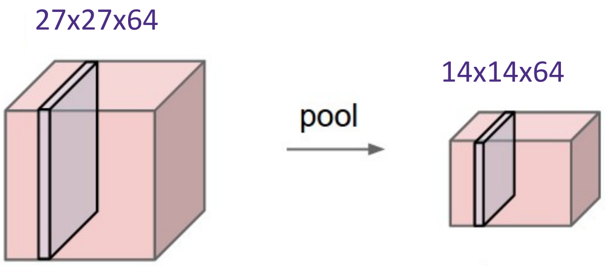
$$a^{(1)}_j = g\left(\sum_{\alpha=1}^{3} X[:,:,\alpha] * K_j[:,:,\alpha]\right)$$

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

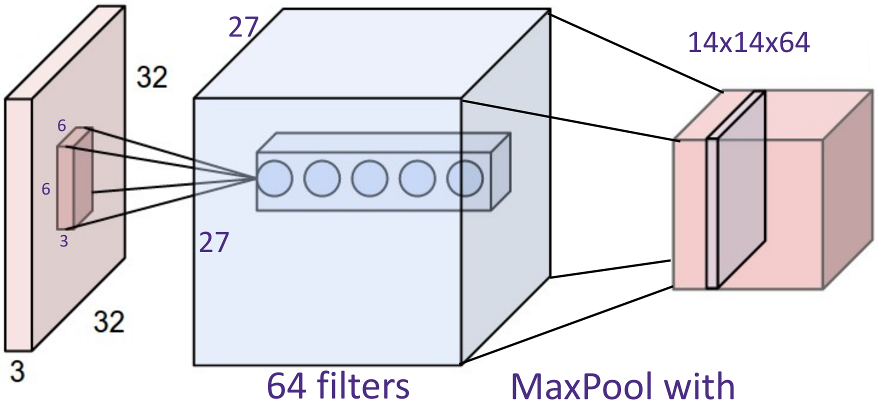
Pooling

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





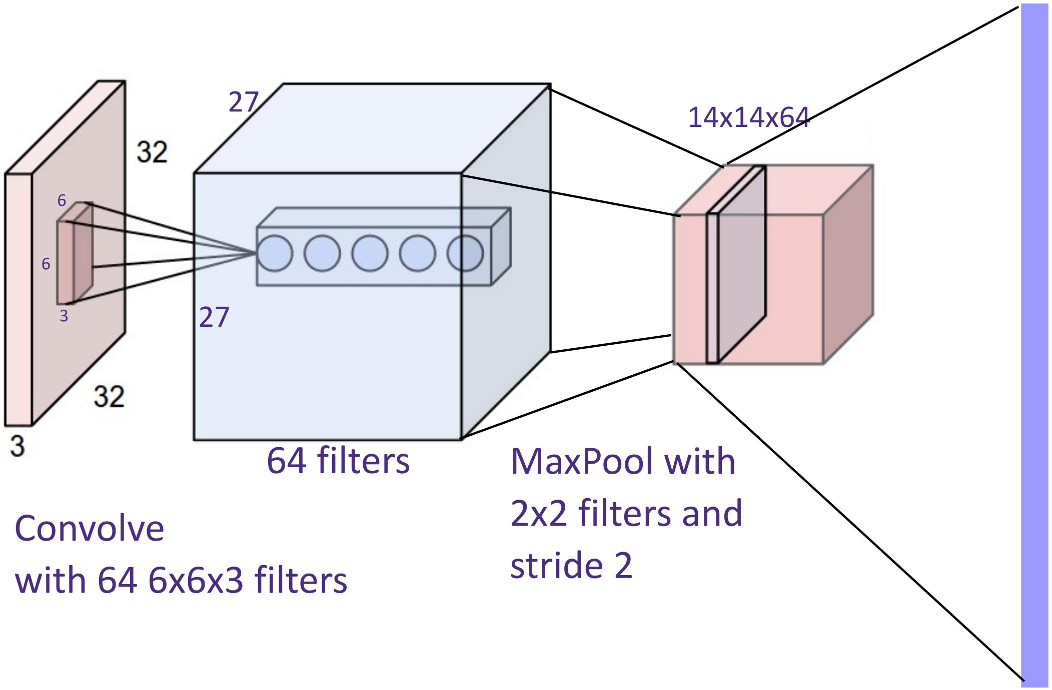
Pooling Convolution layer



MaxPool with
2x2 filters and
stride 2

Convolve
with 64 6x6x3 filters

Flattening

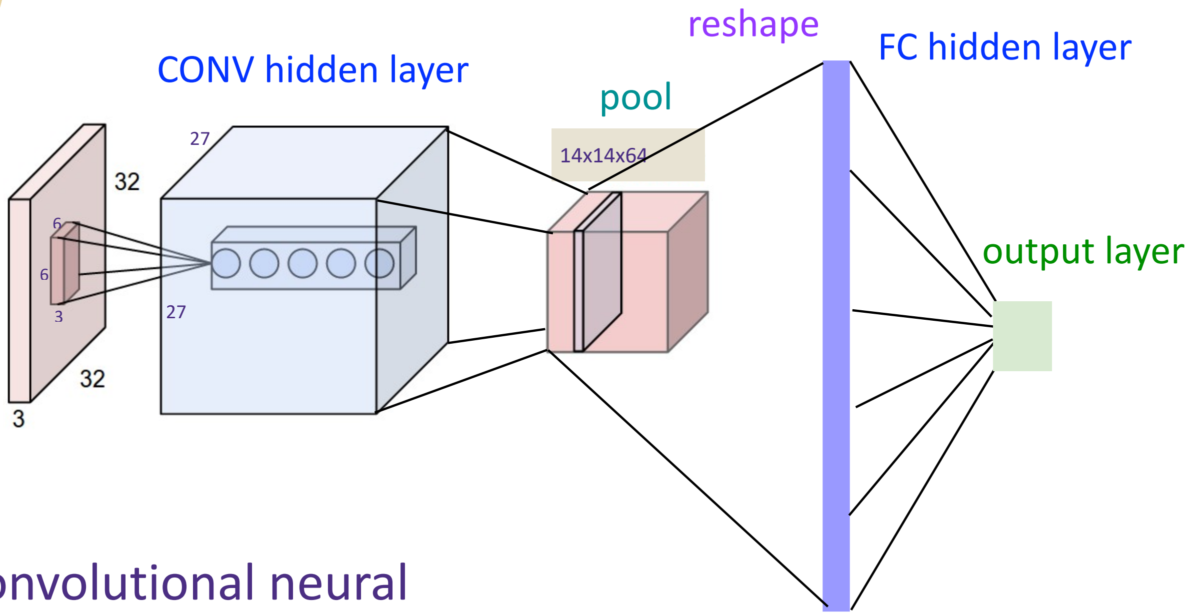


Convolve
with 64 6x6x3 filters

MaxPool with
2x2 filters and
stride 2

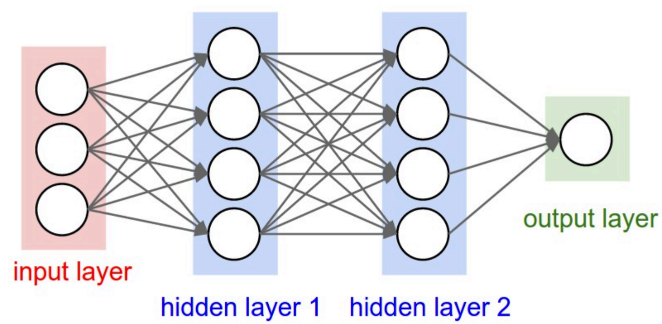
Flatten into a single
vector of size
14*14*64=12544

Training Convolutional Networks

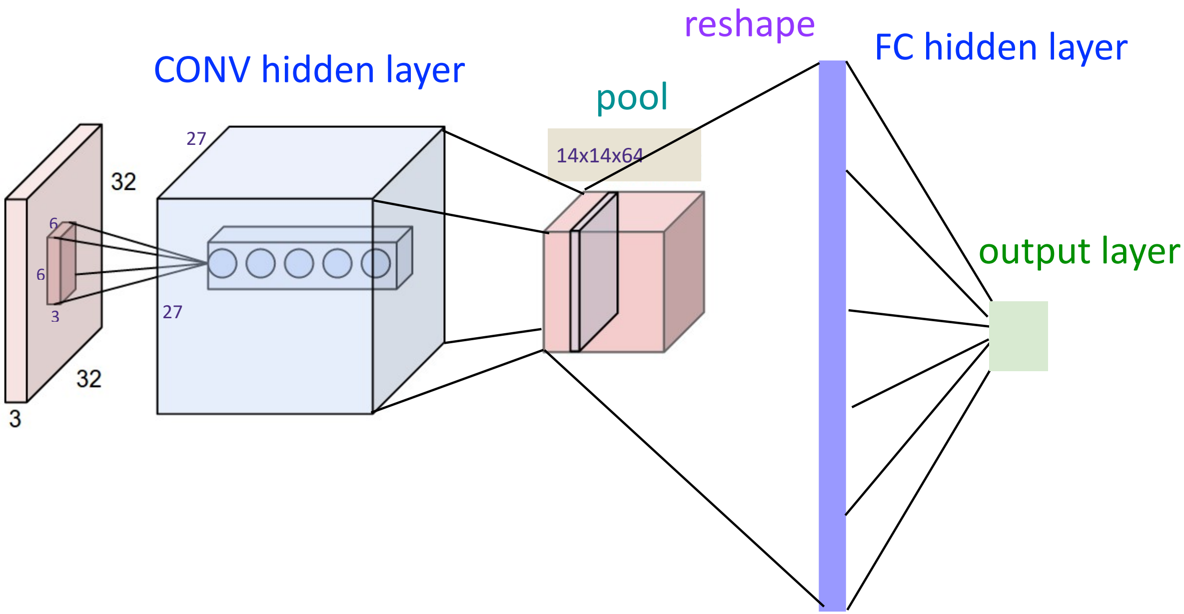


Recall: Convolutional neural networks (CNN) are just regular fully connected (FC) neural networks with some connections removed.

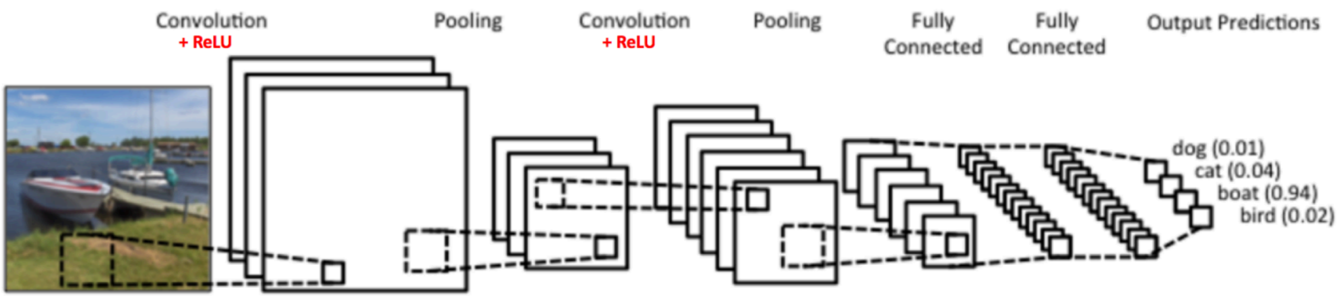
Train with SGD!

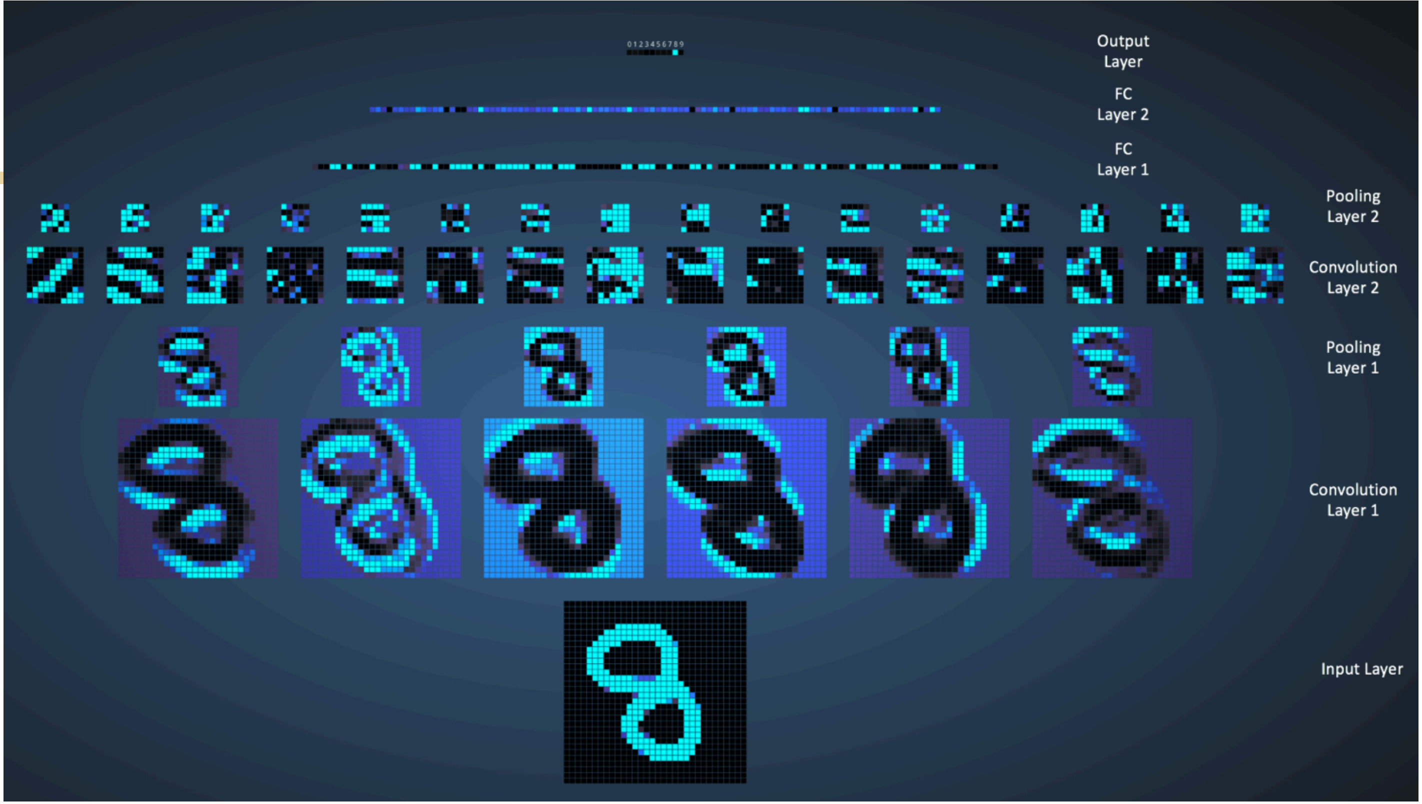


Training Convolutional Networks

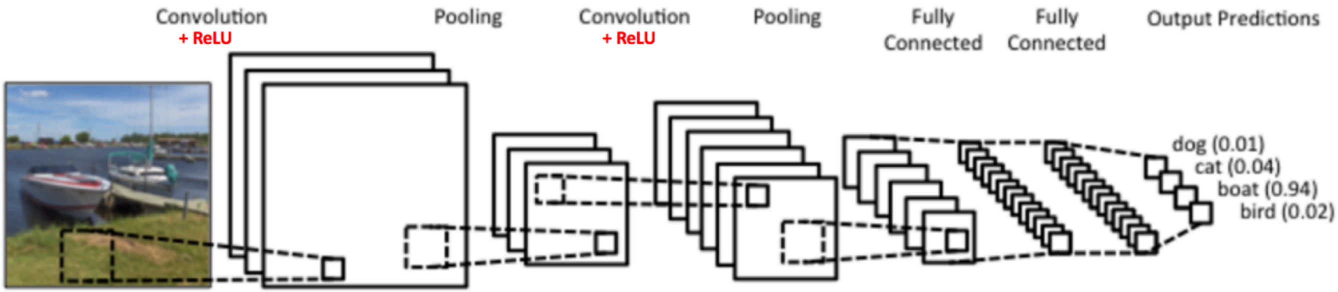


Real example network: LeNet





Real example network: LeNet



Famous CNNs

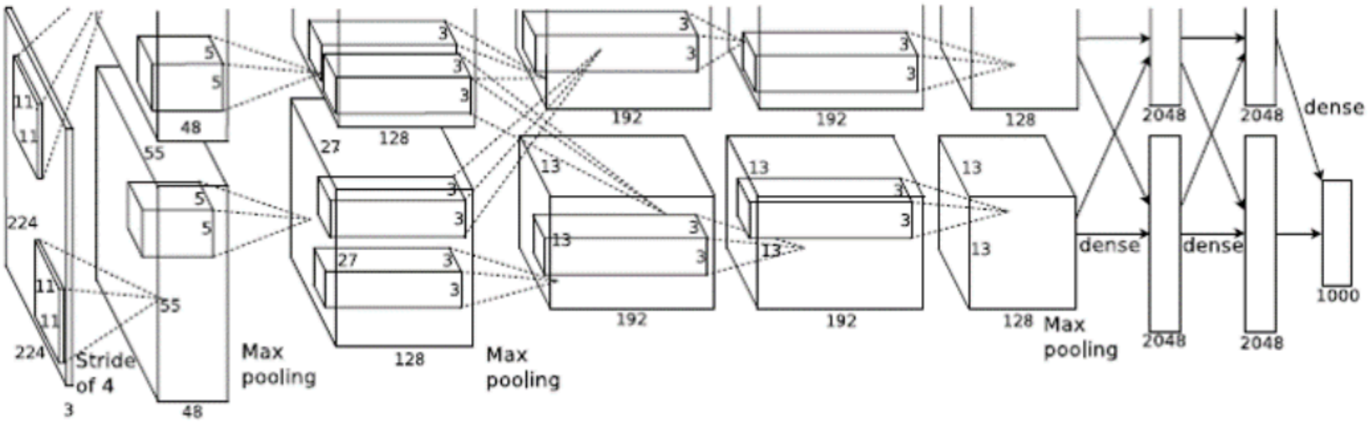
ImageNet Dataset

~14 million images, 20k classes

Deng et al. “Imagenet: a large scale hierarchical image database” ‘09

AlexNet

Breakthrough on ImageNet: ~the beginning of deep learning era



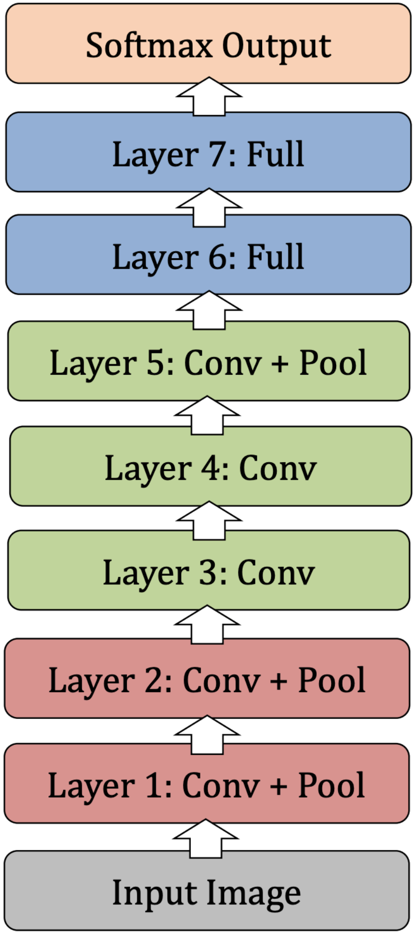
Krizhevsky, Sutskever, Hinton "ImageNet Claasification with Deep Convolutional Neural Networks", NIPS 2012.

AlexNet

8 layers, ~60M parameters

Top5 error: 18.2%

Techniques used:
ReLU activation, overlapping pooling, dropout, ensemble (create 10 patches by cropping and average the predictions), data-augmentation (intensity of RGB channels)



[From Rob Fergus' CIFAR 2016 tutorial]

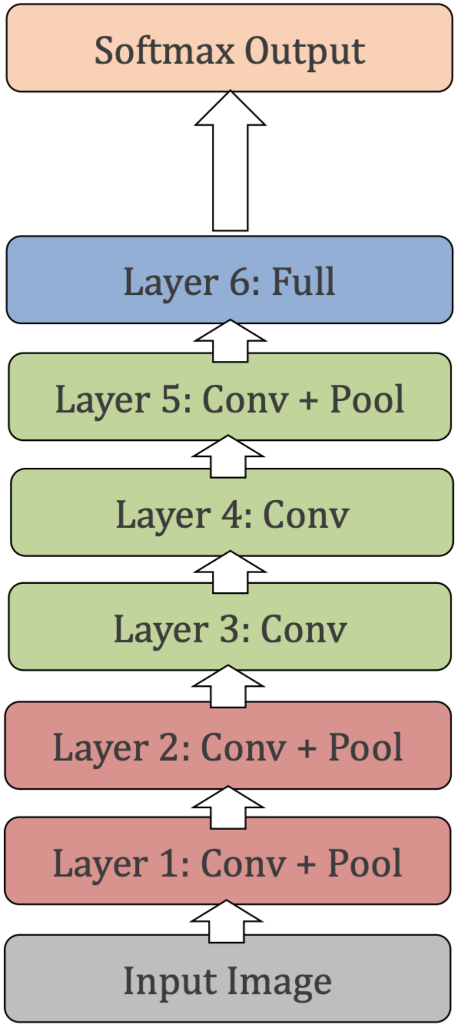
AlexNet

Remove top fully-connected layer 7

Drop ~16 million parameters

1.1% drop in performance

[From Rob Fergus' CIFAR 2016 tutorial]



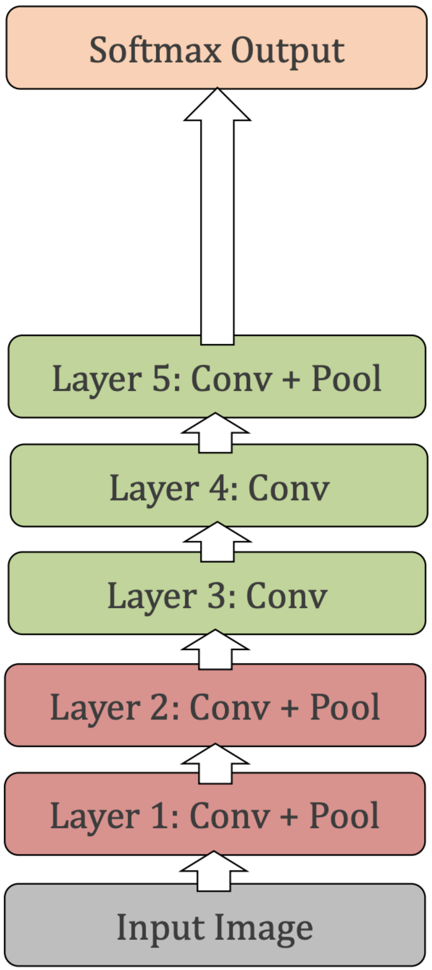
AlexNet

Remove both fully connected layers 6 and 7

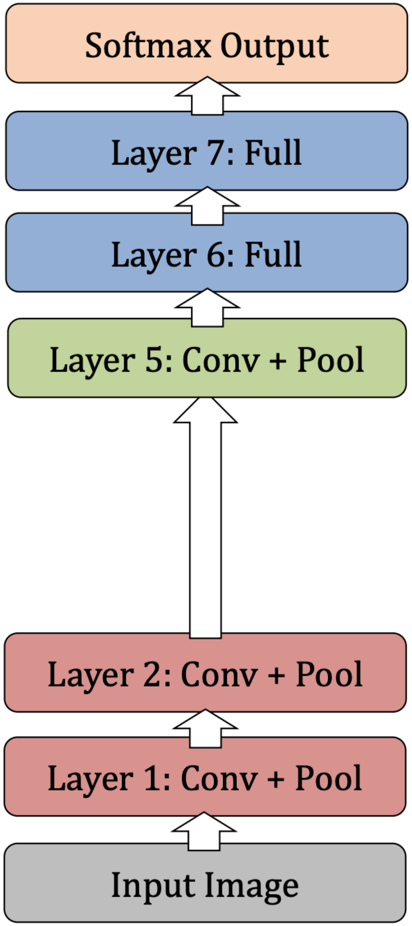
Drop ~50 million parameters

5.7% drop in performance

[From Rob Fergus' CIFAR 2016 tutorial]



AlexNet



Remove upper convolutio / feature extractor layers (layer 3 and 4)

Drop ~1 million parameters

3% drop in performance

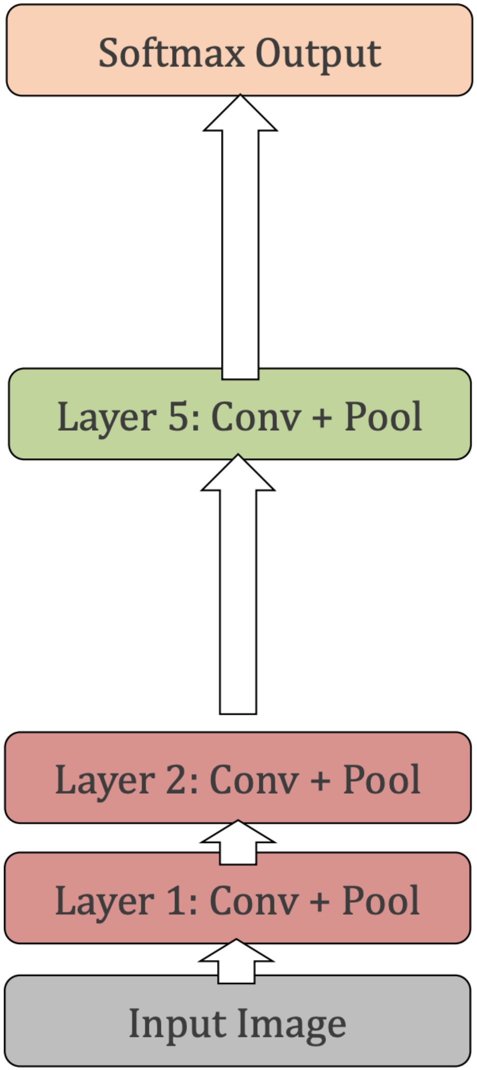
[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet

Remove top fully connected layer 6,7 and upper convolution layers 3,4.

33.5% drop in performance.

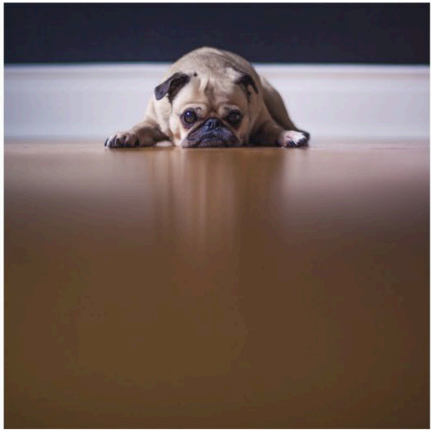
Depth of the network is the key.



[From Rob Fergus' CIFAR 2016 tutorial]

GoogLeNet

Motivation: multiscale nature of images

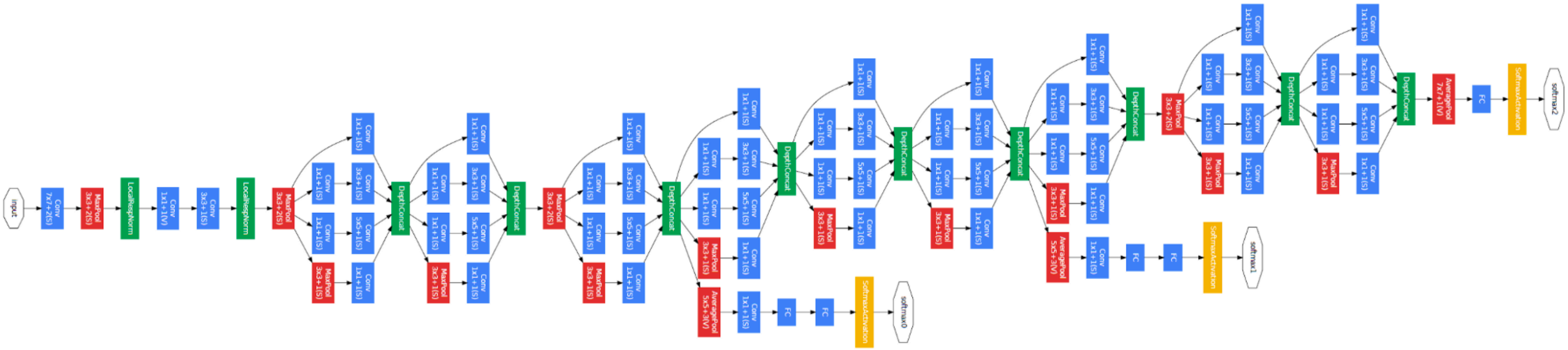
  

Large kernel for global features, and **smaller kernel** for local features.

Idea: have multiple different-size kernels at any layer.

[Going Deep with Convolutions, Szegedy et al. ’14]

GoogLeNet

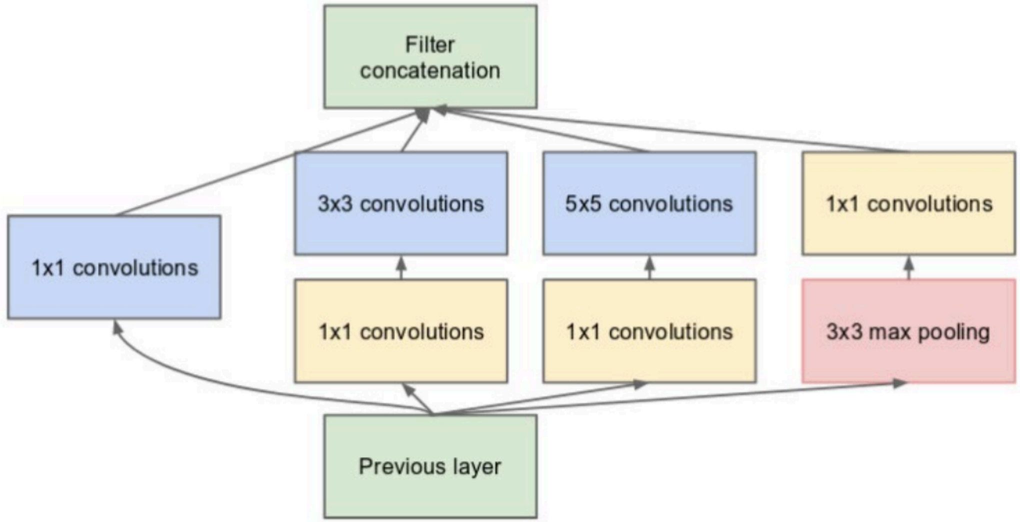


Large kernel for global features, and **smaller kernel** for local features.

Idea: have multiple different-size kernels at any layer.

[Going Deep with Convolutions, Szegedy et al. '14]

Inception Module



Multiple filter scales at each layer

Dimensionality reduction to keep computational requirements down

[Going Deep with Convolutions, Szegedy et al. '14]

Residual Networks

Motivation: extremely deep nets are hard to train (gradient explosion/ vanishing)

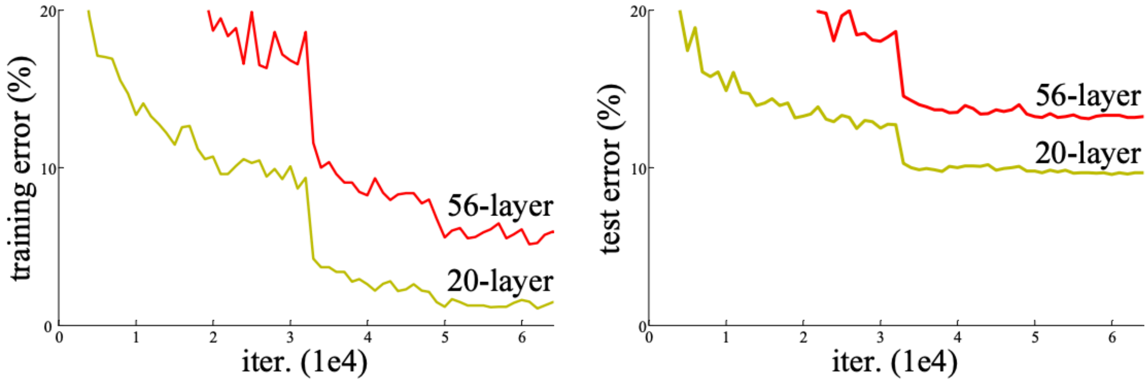


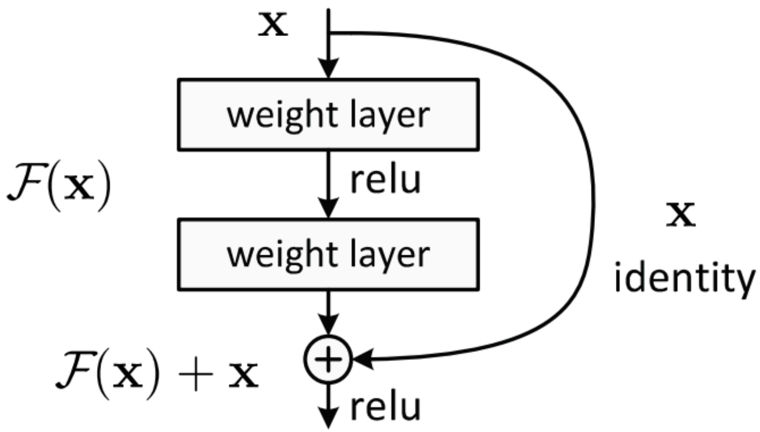
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

[He, Zhang, Ren, Sun, '16]

Residual Networks

Idea: identity shortcut, skip one or more layers.

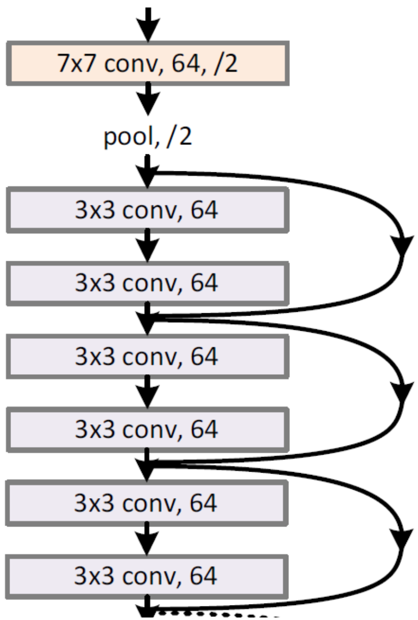
Justification: network can easily simulate shallow network ($F \approx 0$), so performance should not degrade by going deeper.

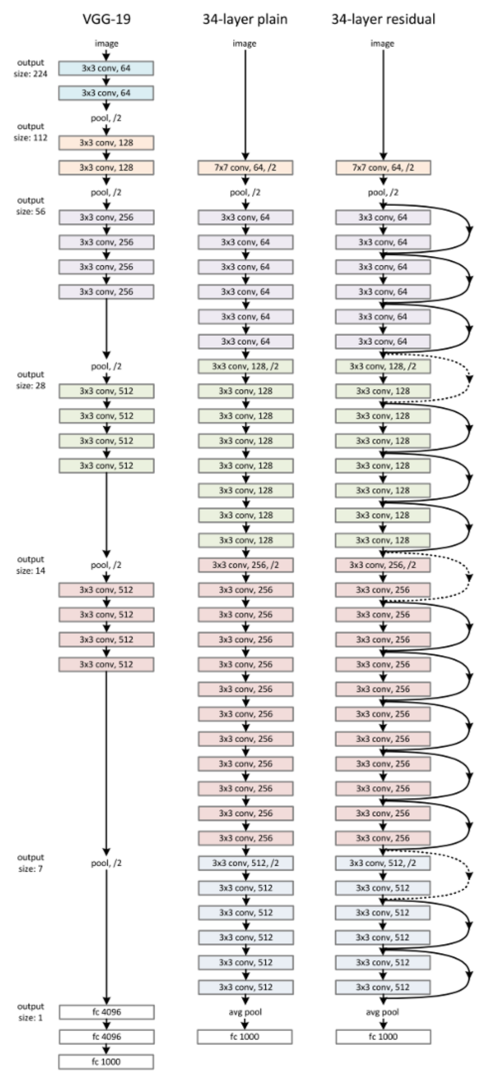


[He, Zhang, Ren, Sun, '16]

Residual Networks

- 3.57% top-5 error on ImageNet
- First deep network with > 100 layers.
- Widely used in many domains (AlphaGo)



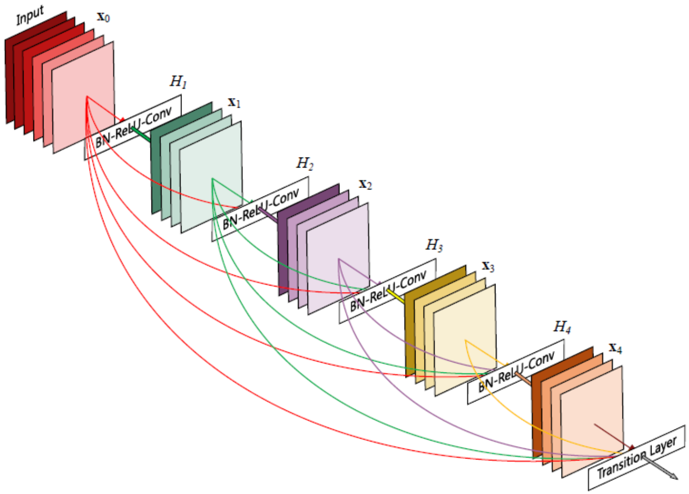


[He, Zhang, Ren, Sun, ’16]

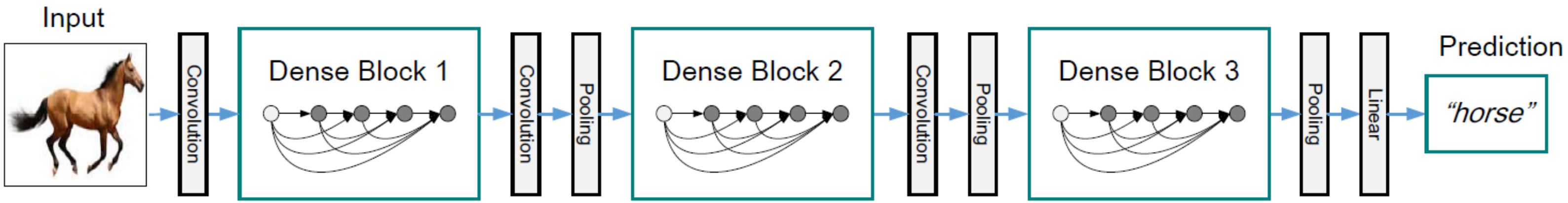
Densely Connected Network

Idea: explicit forward output of layer to all future layers (by concatenation)

Intuition: helps vanishing gradients, encourage reuse features (reduce parameter count)



Issues: network maybe too wide, need to be careful about memory consumption



[He, Zhang, Ren, Sun, '16]

Neural Architecture / Hyper-Parameter Search

Many design choices:
- Number of layers, width, kernel size, pooling, connections, etc.
- Normalization, learning rate, batch size, etc.

Strategies:
- Grid search
- Random search [Bergestra & Bengio ’12]
- Bandit-based [Li et al. ’16]
- Gradient-based (DARTS) [Liu et al. ’19]
- Neural tangent kernel [Xu et al. ’21]
- …