Convolutional Neural Networks

Multi-layer Neural Network

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

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

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

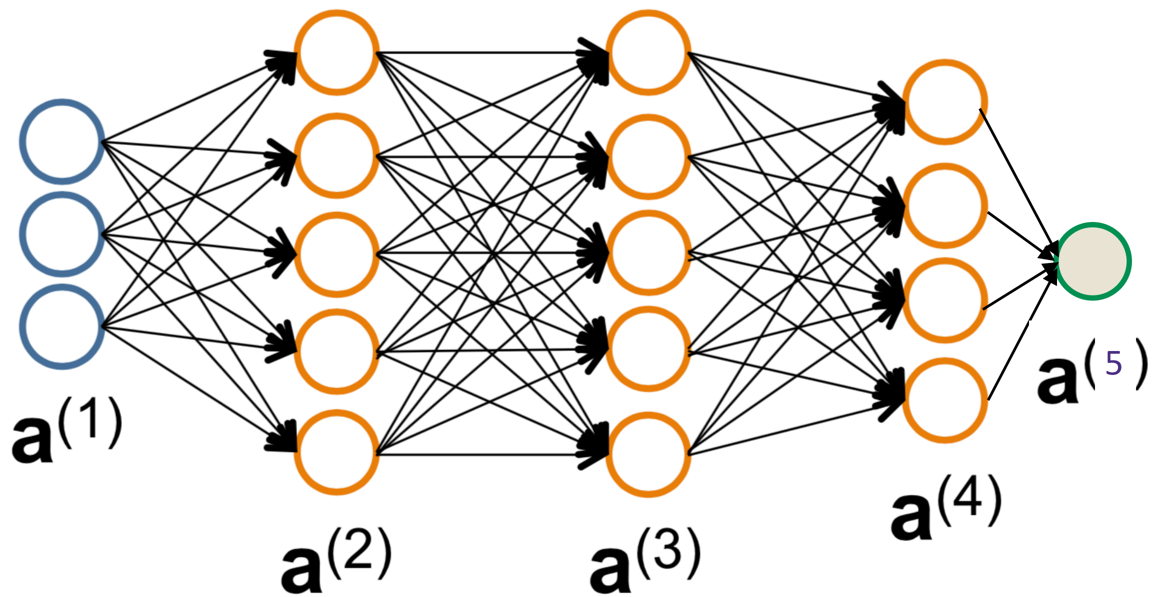
$$\vdots$$

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

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

$$\vdots$$

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



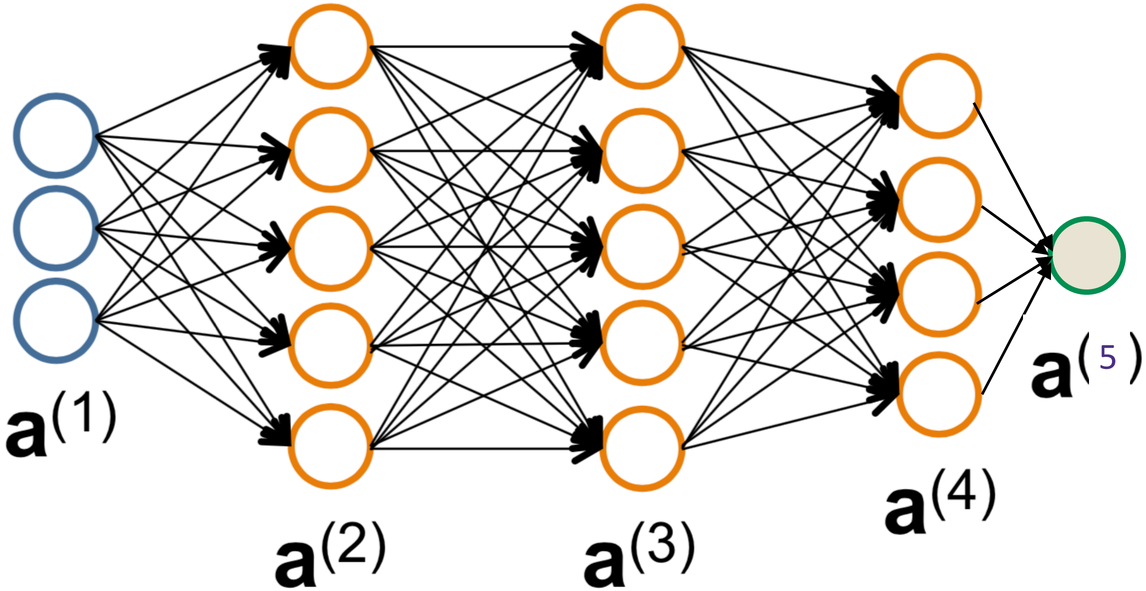
$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$$

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

Binary Logistic Regression

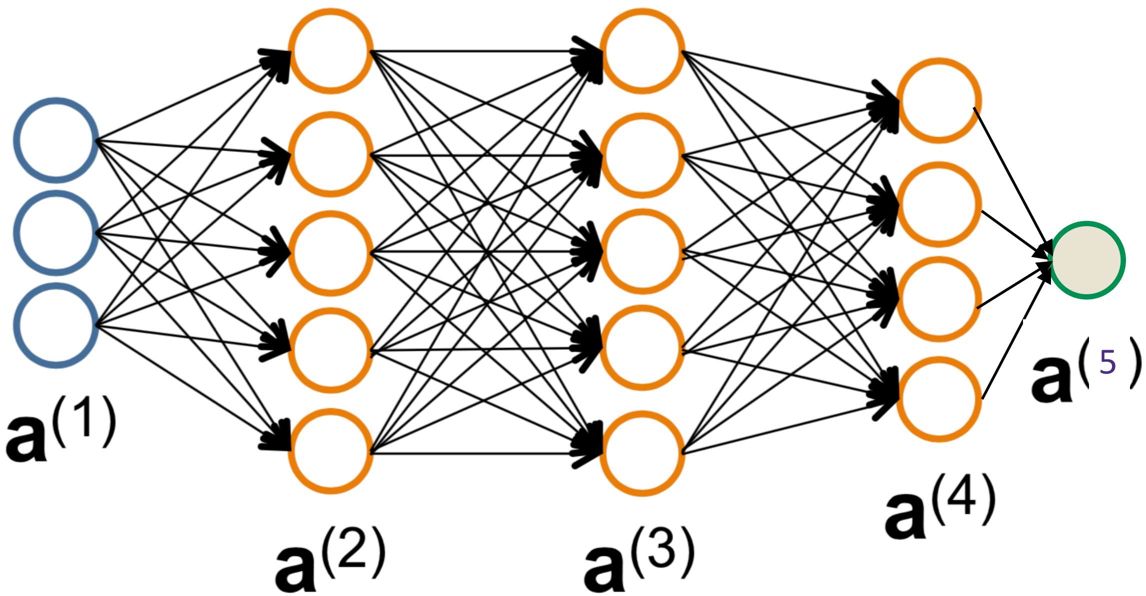
Neural Network Architecture

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



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**.



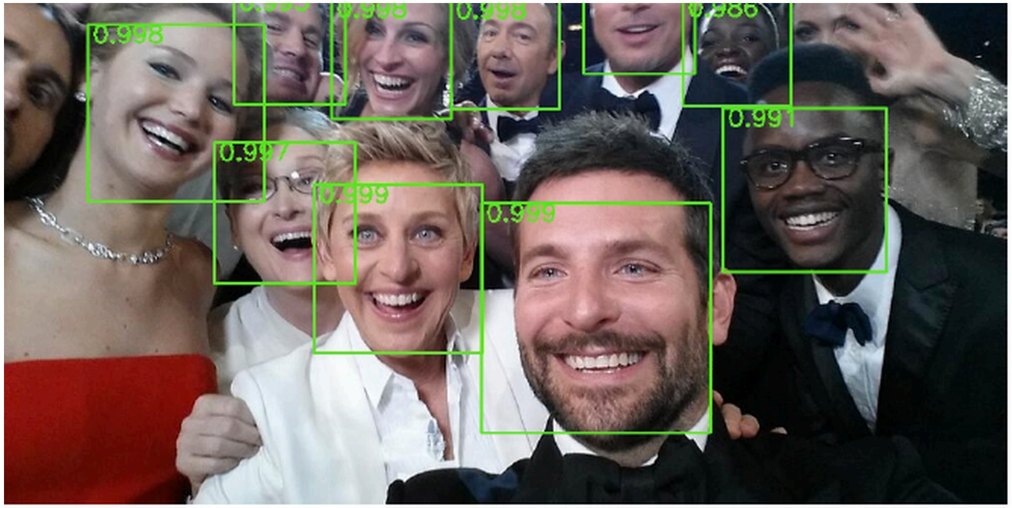
We say a layer is **Fully Connected (FC)** if all linear mappings from the current layer to the next layer are permissible.

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

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

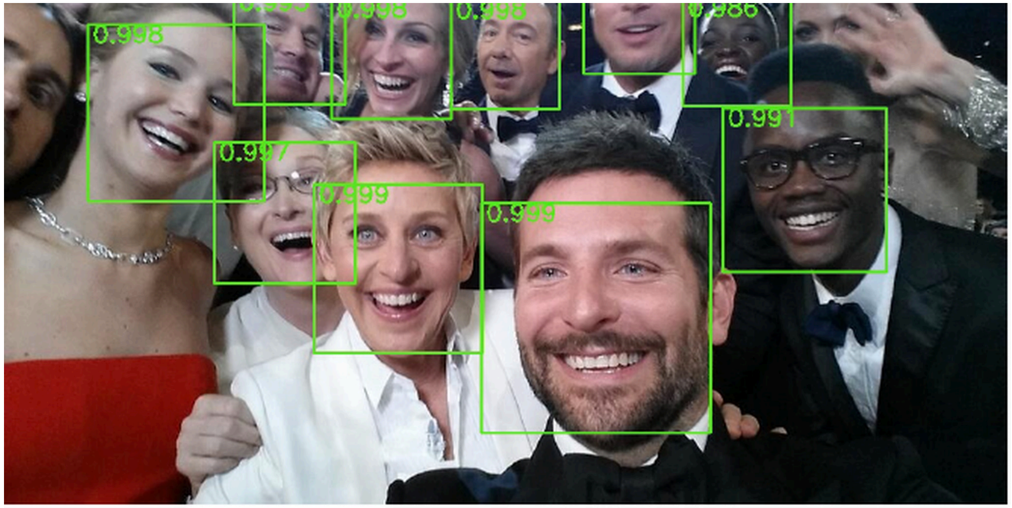
Neural Network Architecture

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 detector inside a window** across an image.

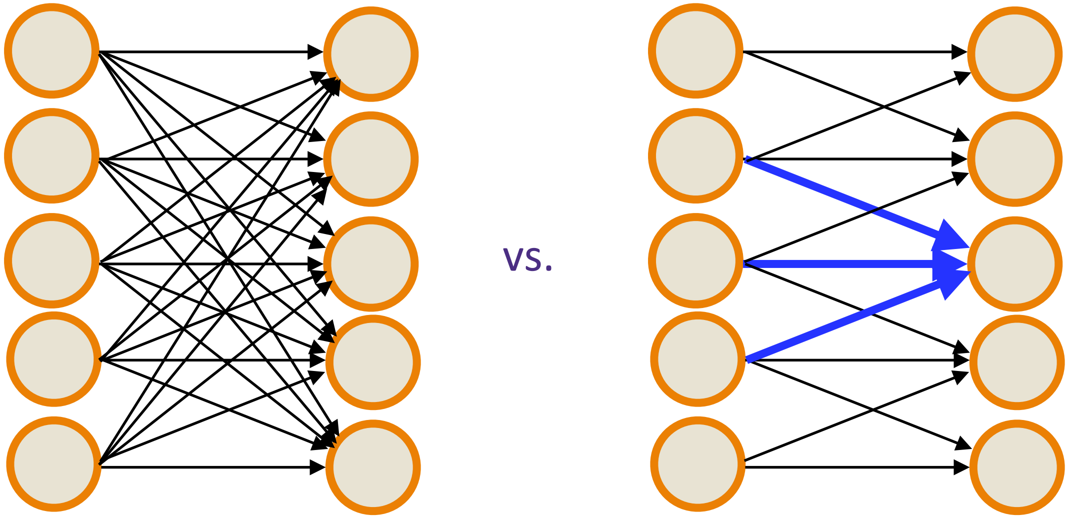


Neural Network Architecture

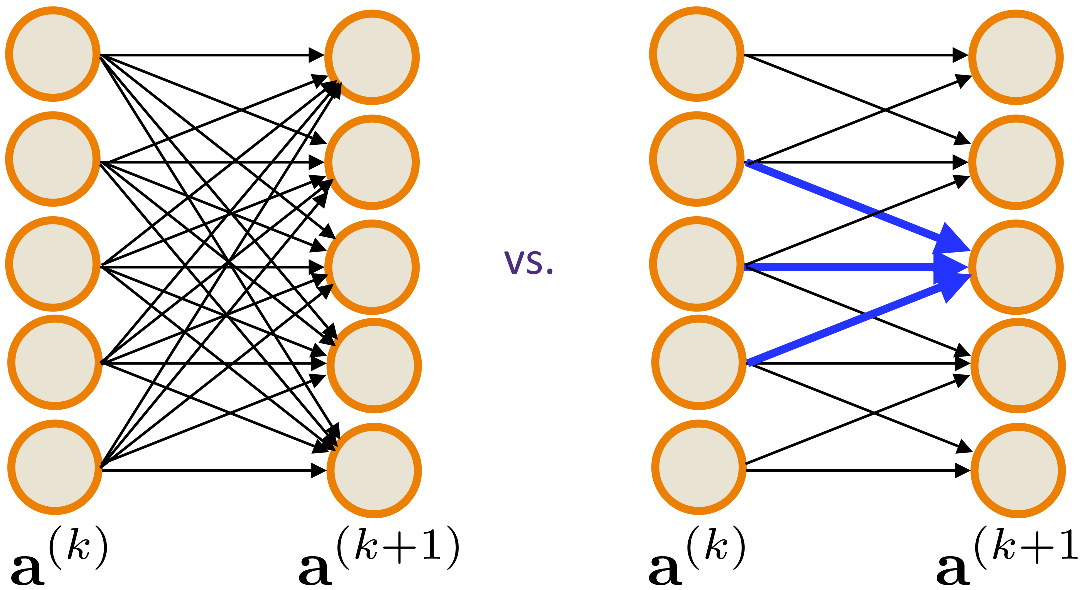
Objects are often **localized in space** so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.



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



Neural Network Architecture

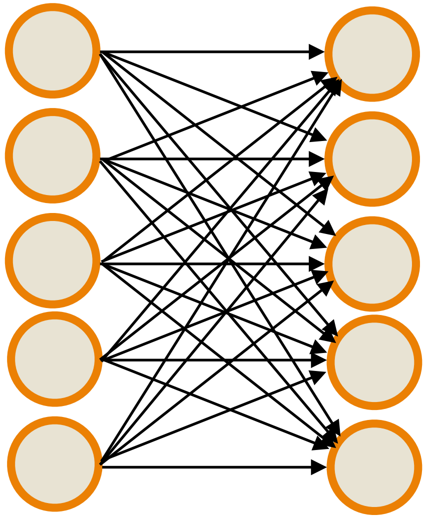


$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3}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

Parameters: n^2 $3n-2$

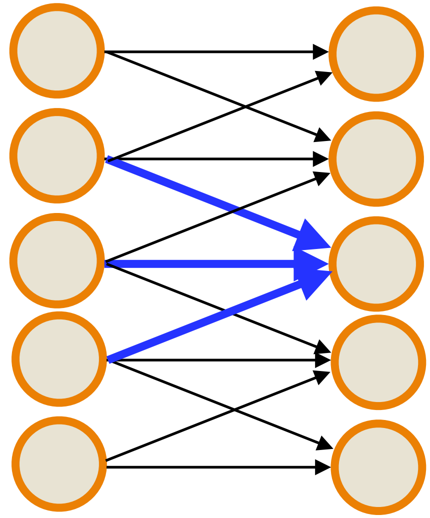
$$\mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{n-1} \Theta_{i,j}\mathbf{a}_j^{(k)}\right)$$

Neural Network Architecture



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

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 $3n-2$ 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)

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

$[x_1, \ldots, x_n]$

$i = 1$:

$$\Rightarrow (\theta * x)_1 = \theta_0 x_1 + \theta_1 x_2 + \theta_2 x_3$$

x_1	x_2	x_3	x_4	x_5
1	1	1	0	0

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

θ_1	θ_2	θ_3
1	0	1

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

Output $\theta * x$

Example (1d convolution)

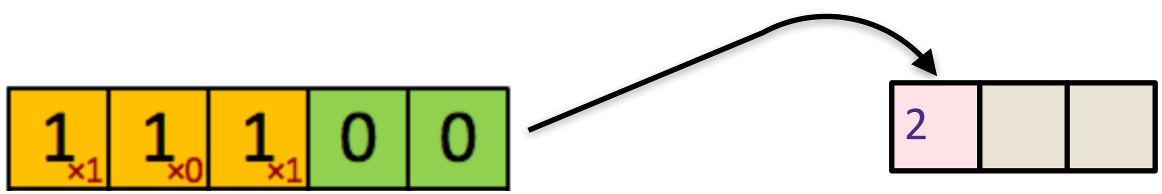
1	1	1	0	0

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

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

1	0	1

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



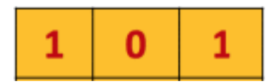
Output $\theta * x$

$$\Rightarrow (\theta * x)_1 = \theta_0 x_1 + \theta_1 x_2 + \theta_2 x_3$$

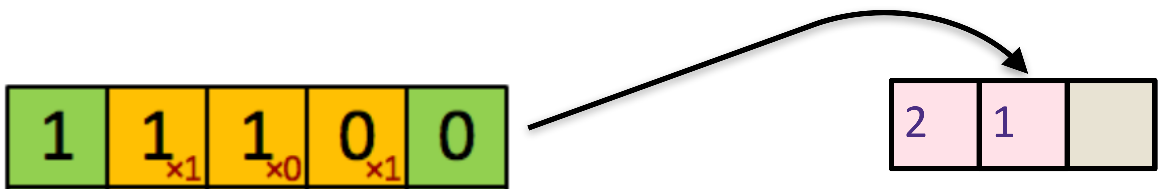
Example (1d convolution)

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

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



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



Output $\theta * x$

Example (1d convolution)

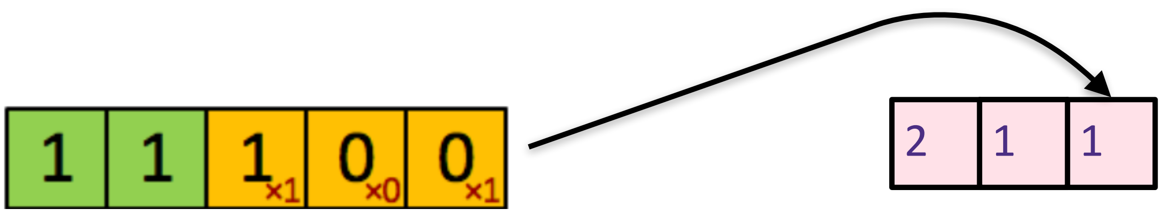
1	1	1	0	0

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

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

1	0	1

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



Output $\theta * x$

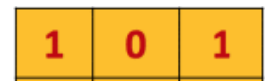
 # Stride is how much filter moves each time. We used? Stride = 1

 # Padding adds extra values (usually zeros) to the edges before convolving. Without it, output is smaller than input. Did we pad? No

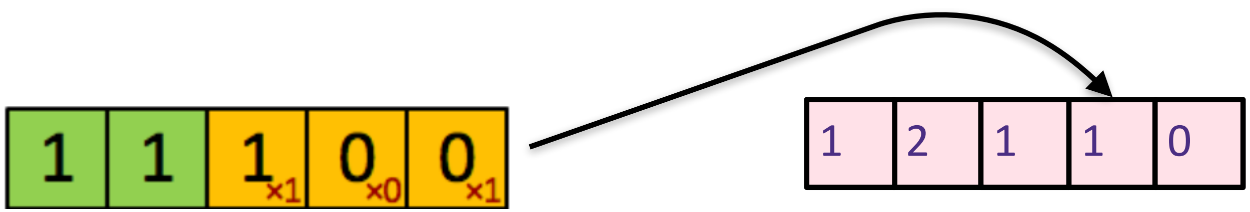
Example (1d convolution) - **with padding**

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

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



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



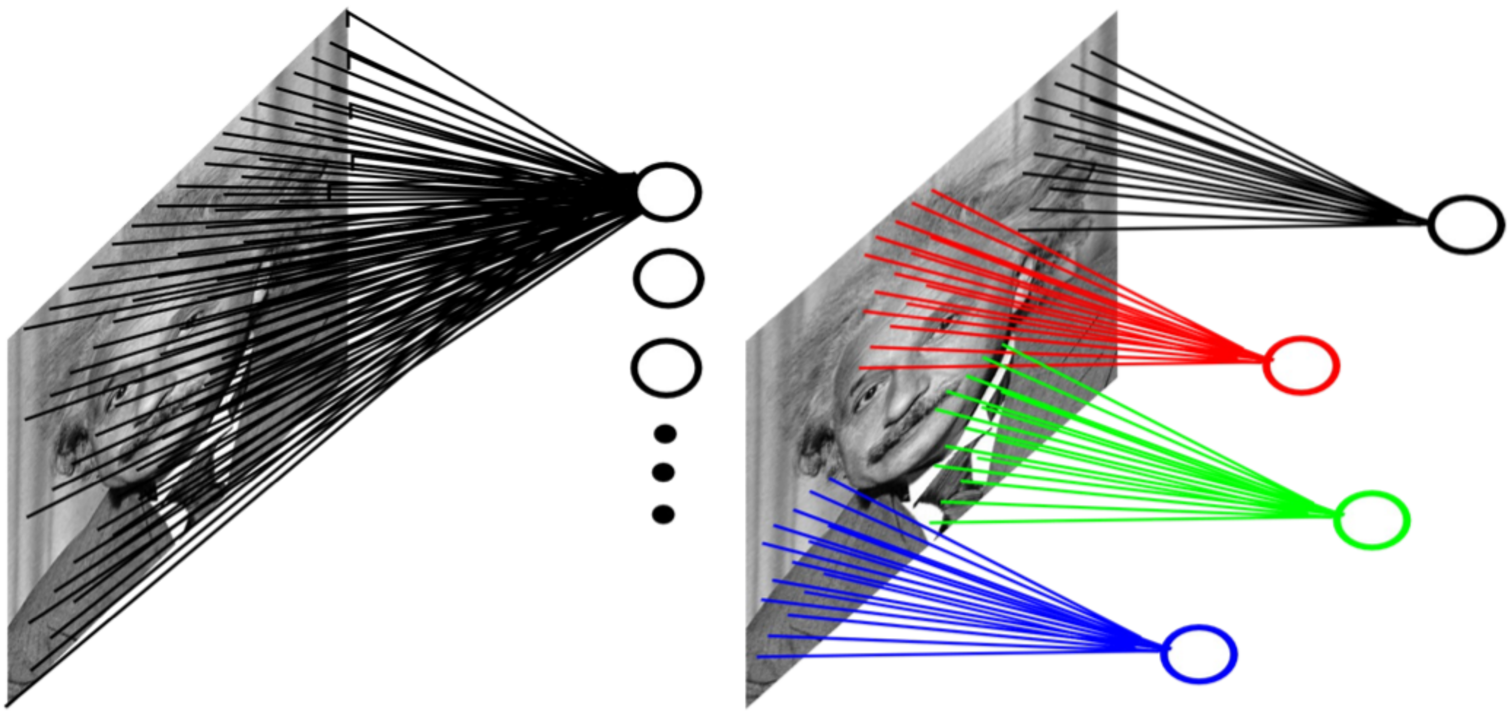
Output $\theta * x$

 # Stride is how much filter moves each time. We used? Stride = 1

 # Padding adds extra values (usually zeros) to the edges before convolving. Without it, output is smaller than input. Did we pad? Now we did!

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)

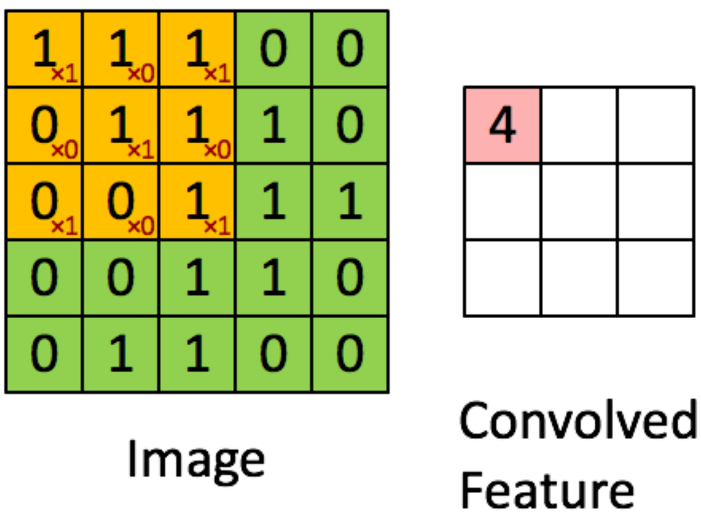
$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

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

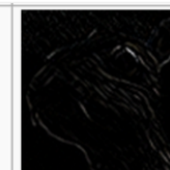
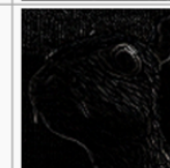
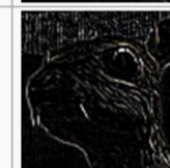
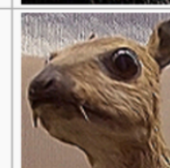
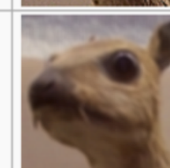
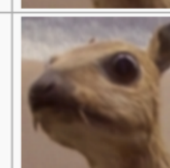


$I * K$

Convolution of images

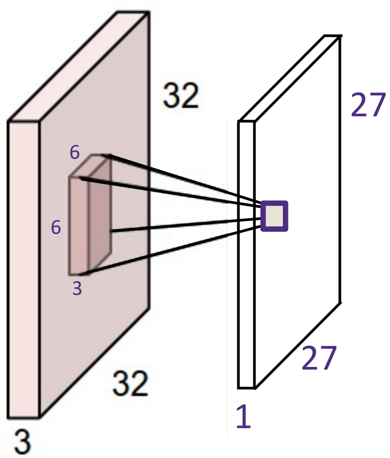
$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

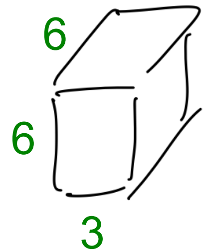
Image I

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

Because there are 3 channels, each 6x6 conv filter is a cube with depth 3





To compute a convolution output:

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

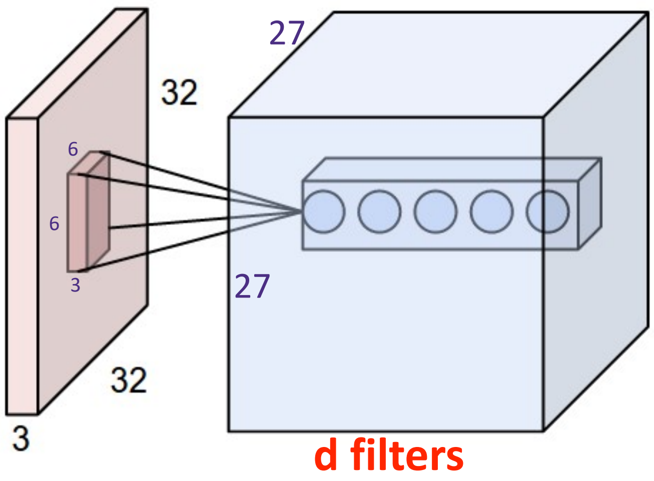
For all channels, slice the image at channel α, convolve with kernel for that channel, sum

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

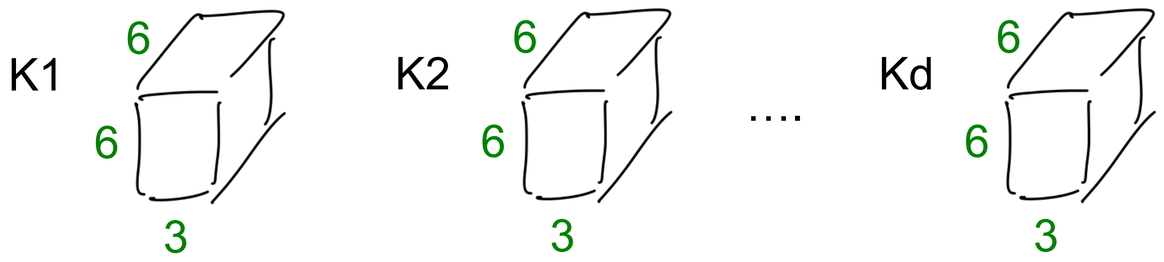
r = # channels.
Often RGB

n x n is image pixels. Often 0-255 range

Stacking convolved images

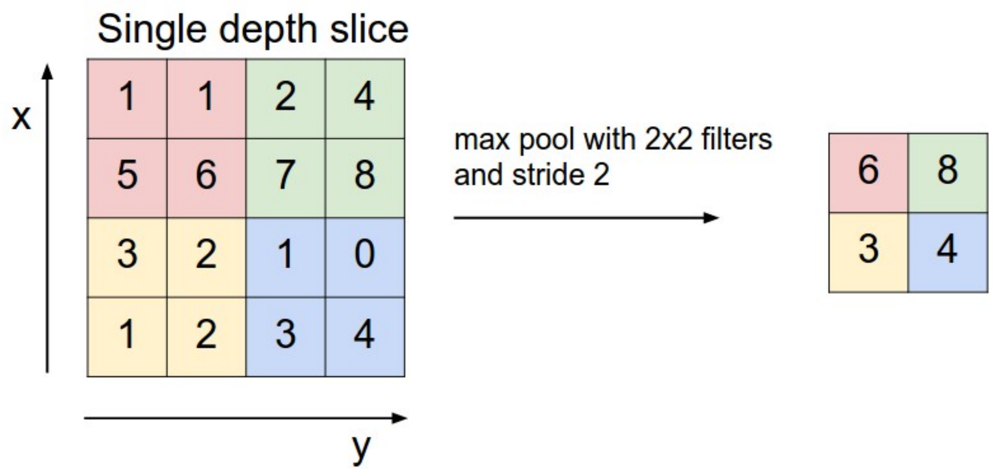


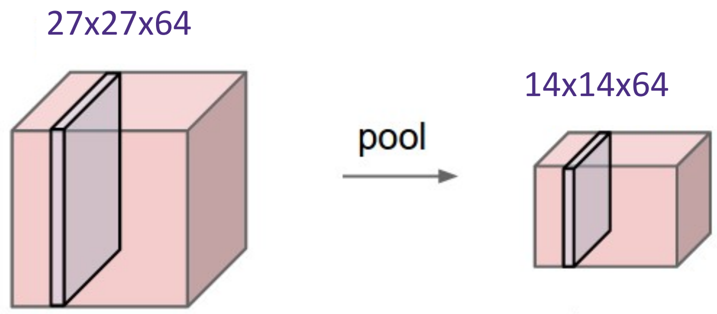
Repeat with d filters!



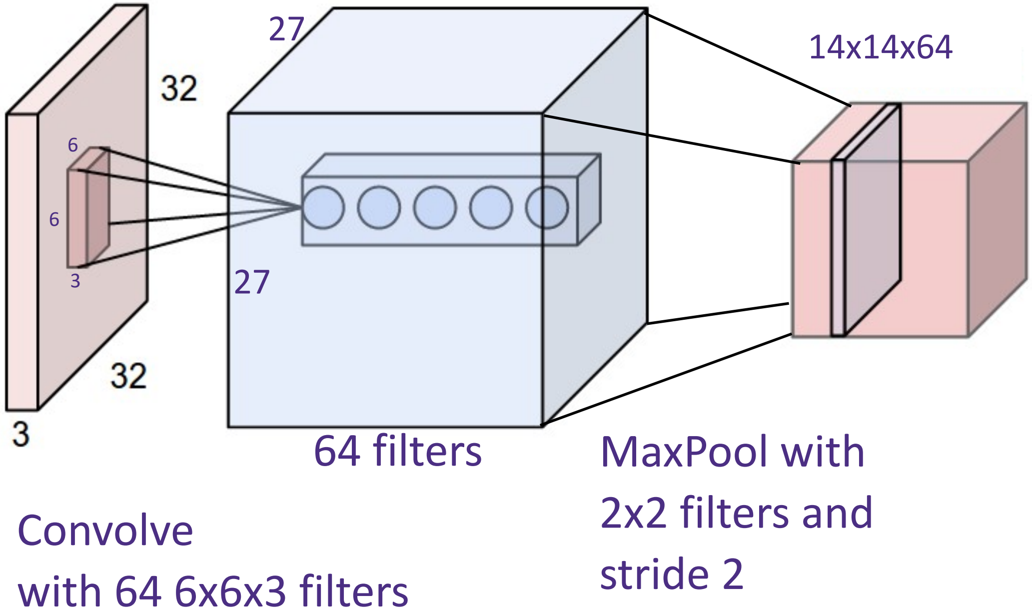
Pooling

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

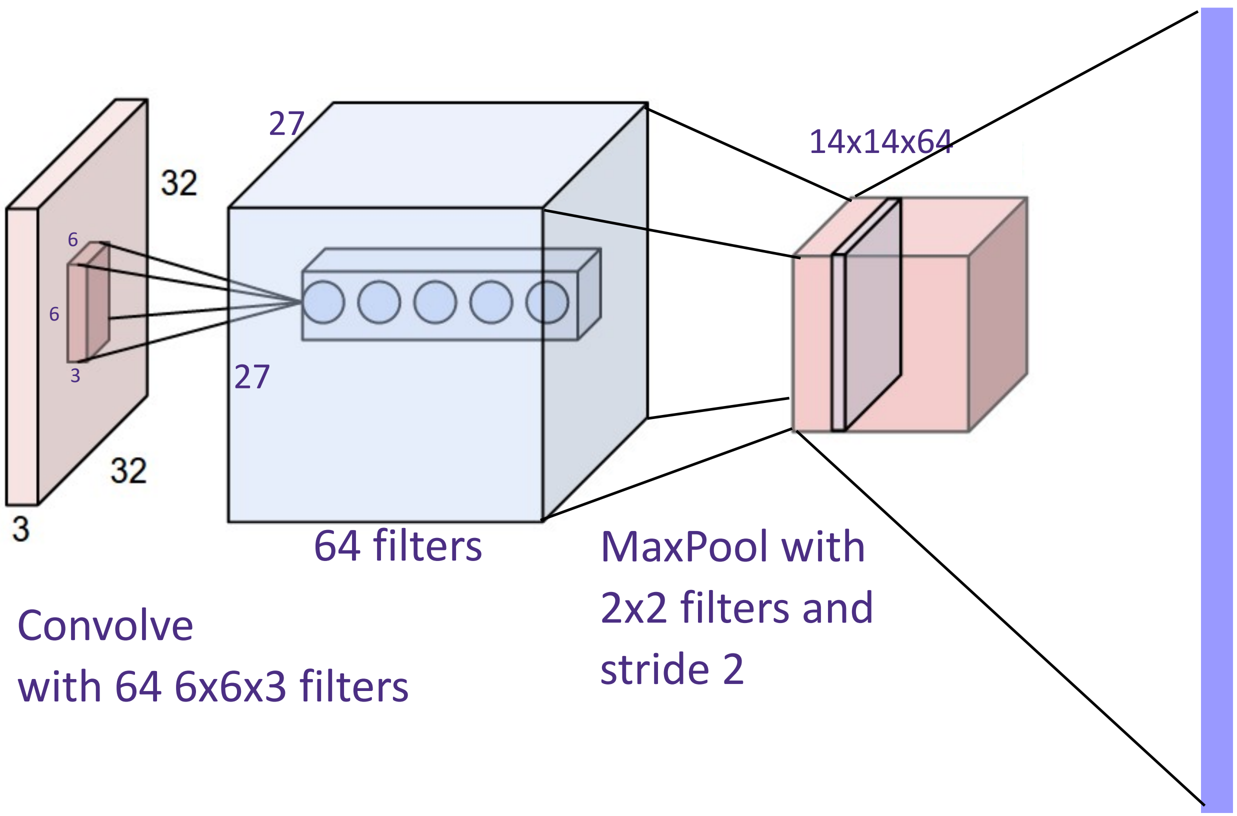




Pooling Convolution layer

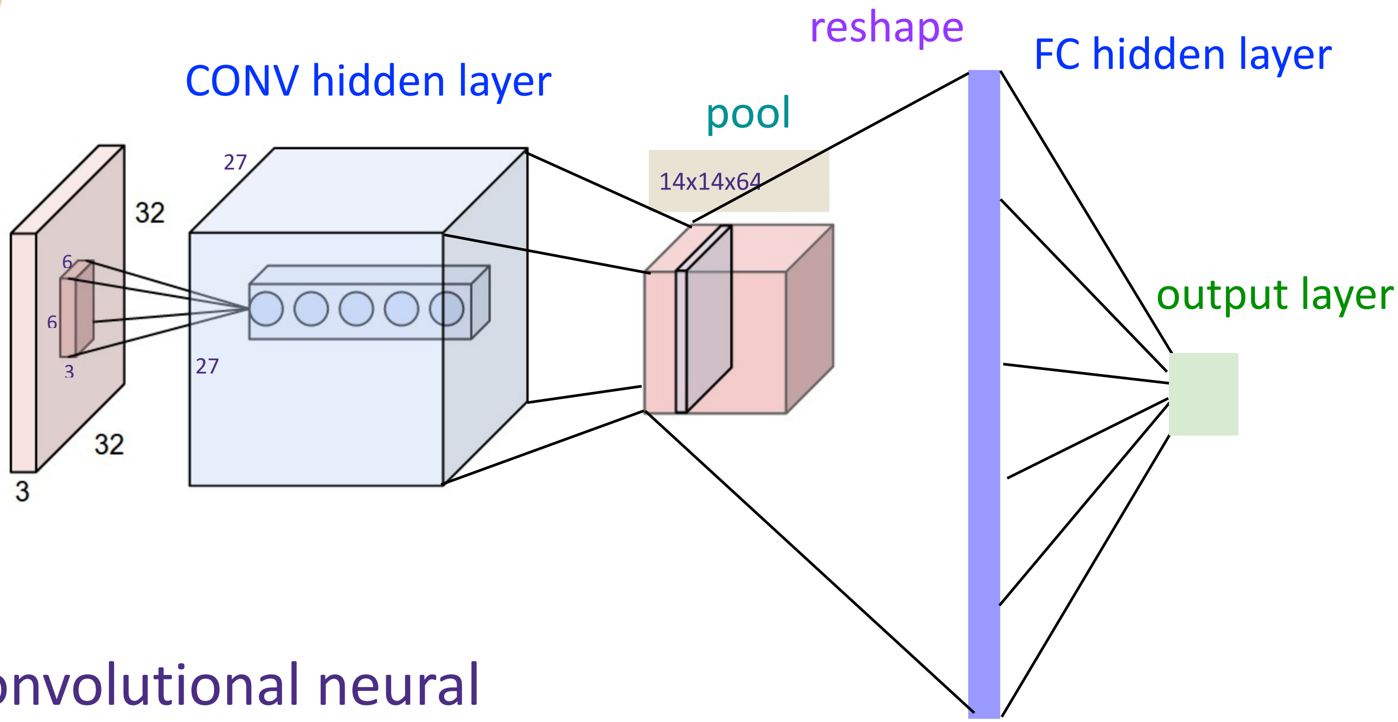


Flattening



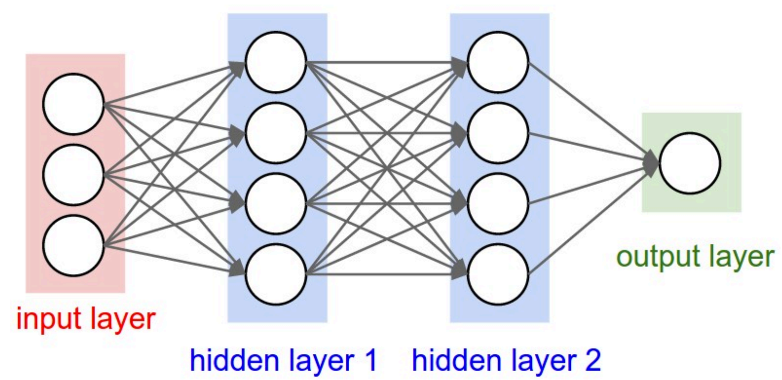
Flatten into a single vector of size 14*14*64=12544

Training Convolutional Networks

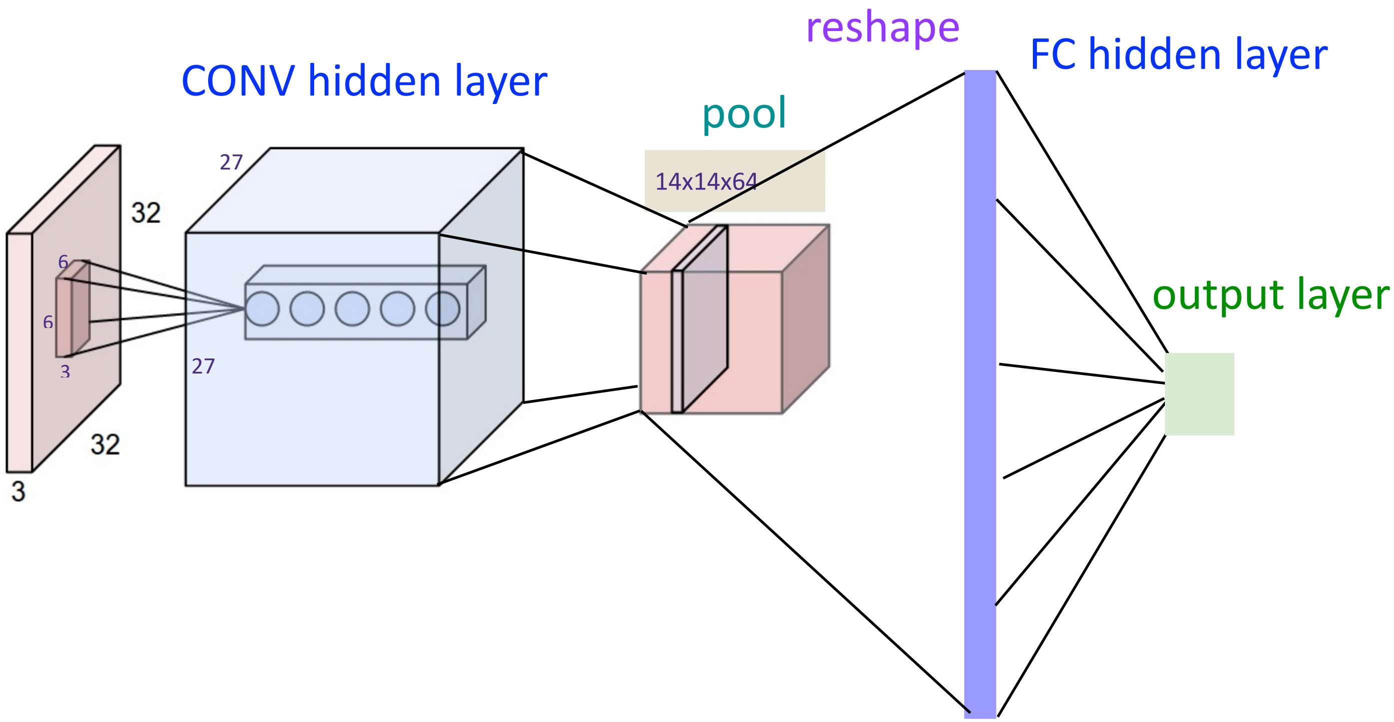


Recall: Convolutional neural networks (CNN) are just regular fully connected (FC) neural networks with some connections removed.

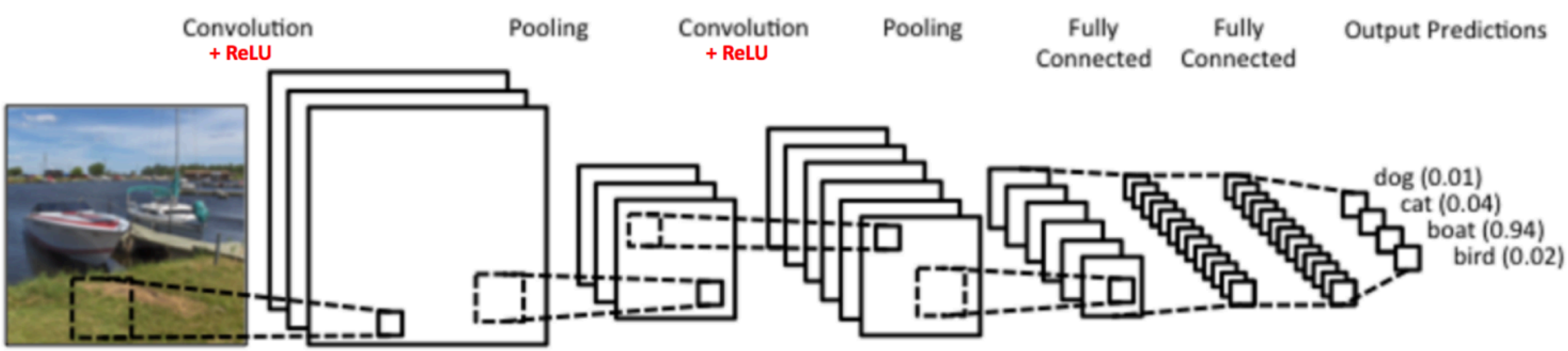
Train with SGD!

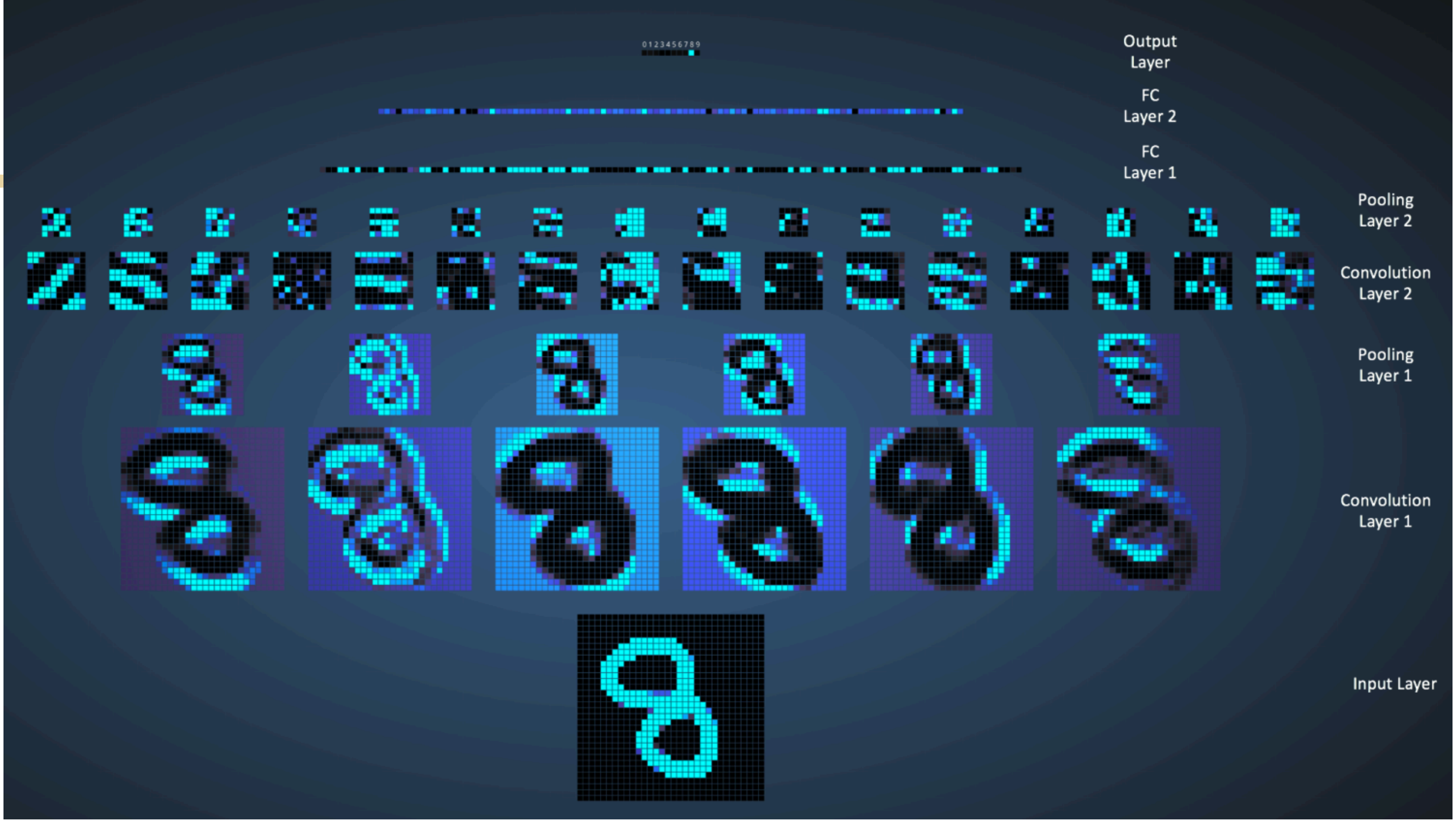


Training Convolutional Networks

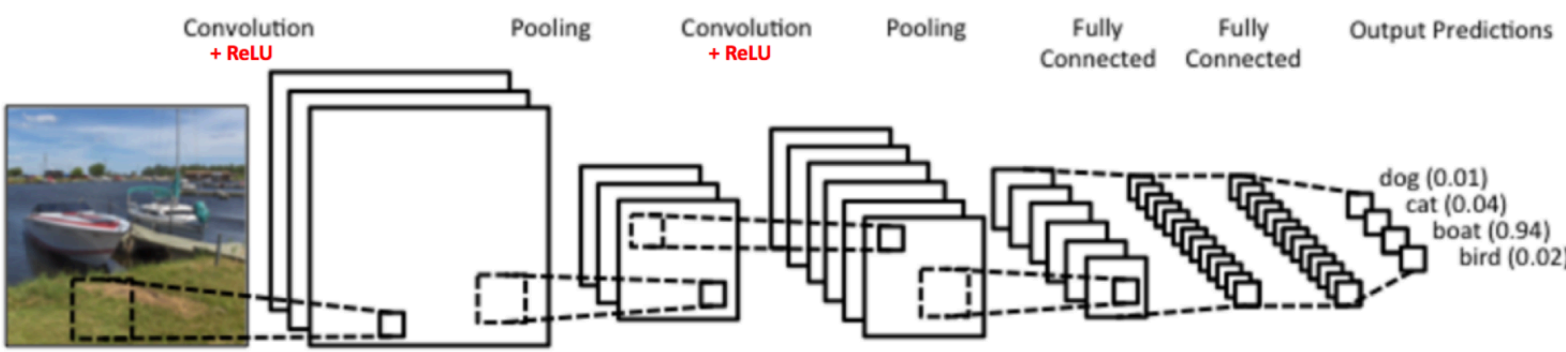


Real example network: LeNet





Real example network: LeNet



Famous CNNs

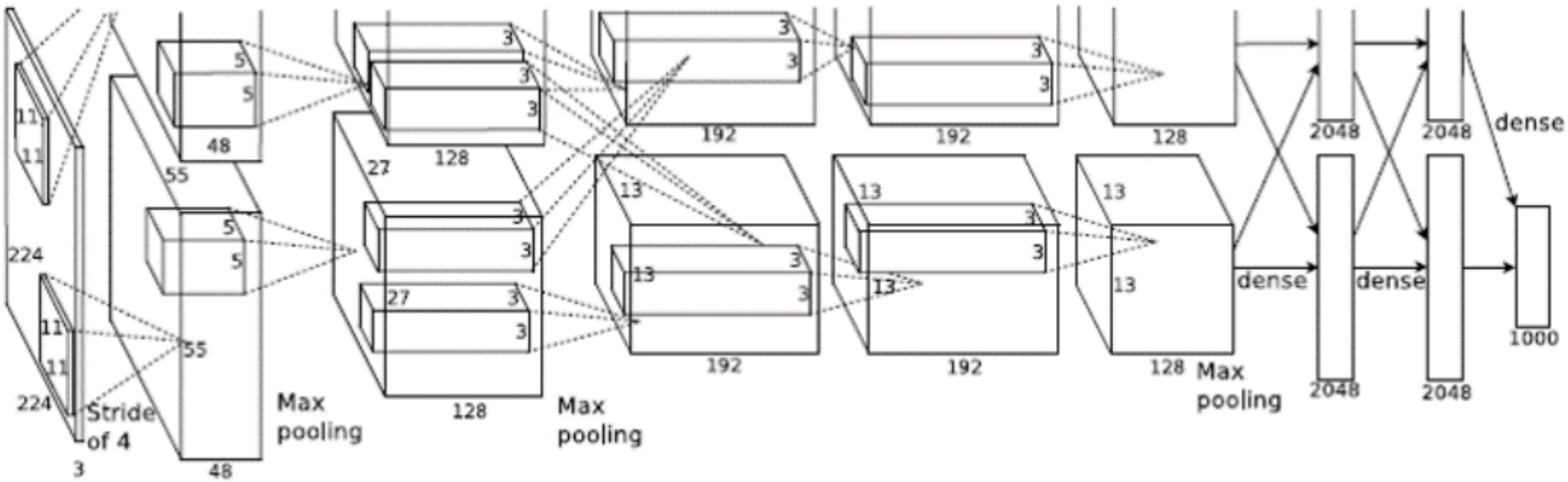
ImageNet Dataset

~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.

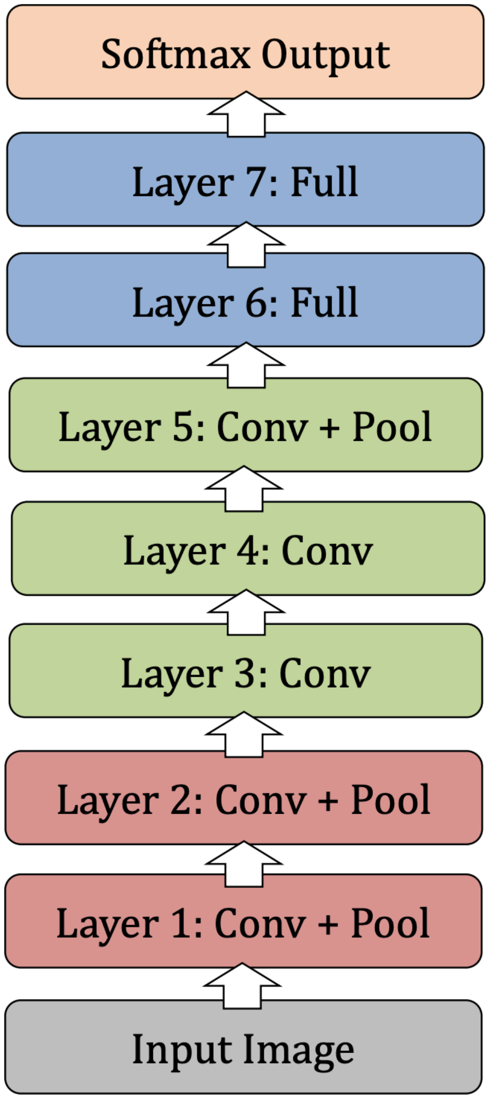
AlexNet

8 layers, ~60M parameters

Top5 error: 18.2%

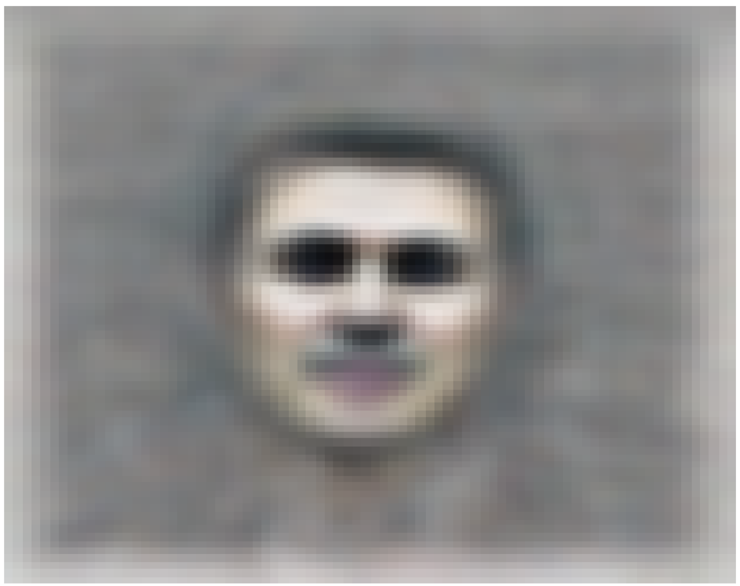
Techniques used:
ReLU activation, overlapping pooling, dropout, ensemble (create 10 patches by cropping and average the predictions), data-augmentation (intensity of RGB channels)

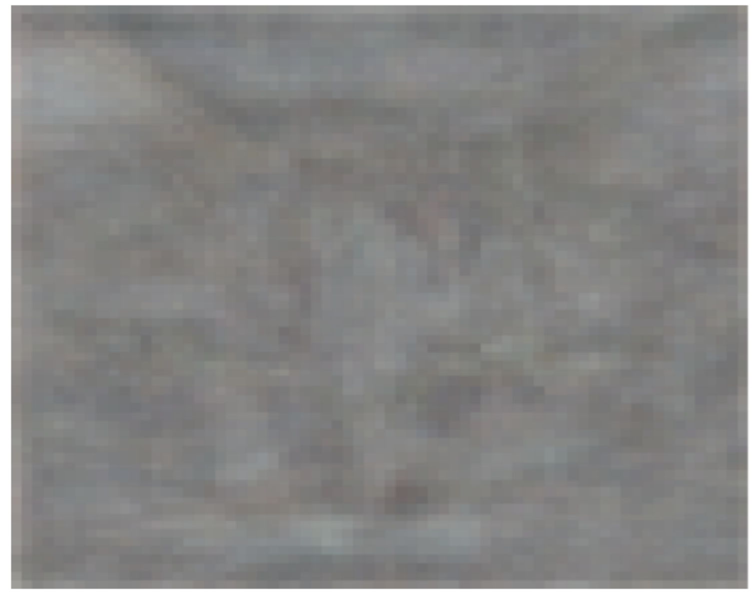
[From Rob Fergus' CIFAR 2016 tutorial]



The Cat Neuron

Train a conv net *unsupervised* on 10 million YouTube images, what features do your neurons learn?





"Contrary to what appears to be a widely-held intuition, our experimental results reveal that it is possible to train a face detector without having to label images as containing a face or not."

Such early days of representation learning… 🥹

[Building high-level features using large scale unsupervised learning, Quoc Le et al. '12]

GoogLeNet

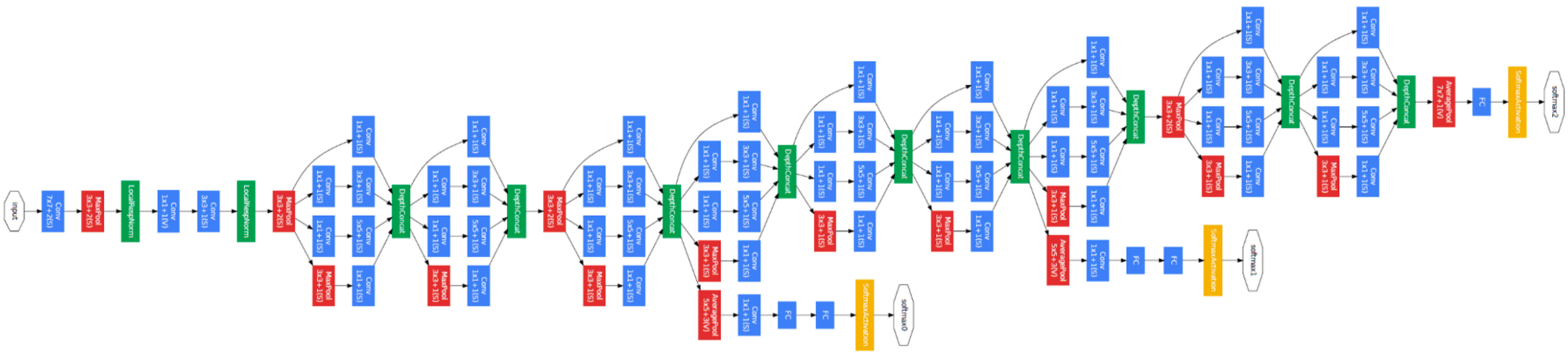
Motivation: multiscale nature of images

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

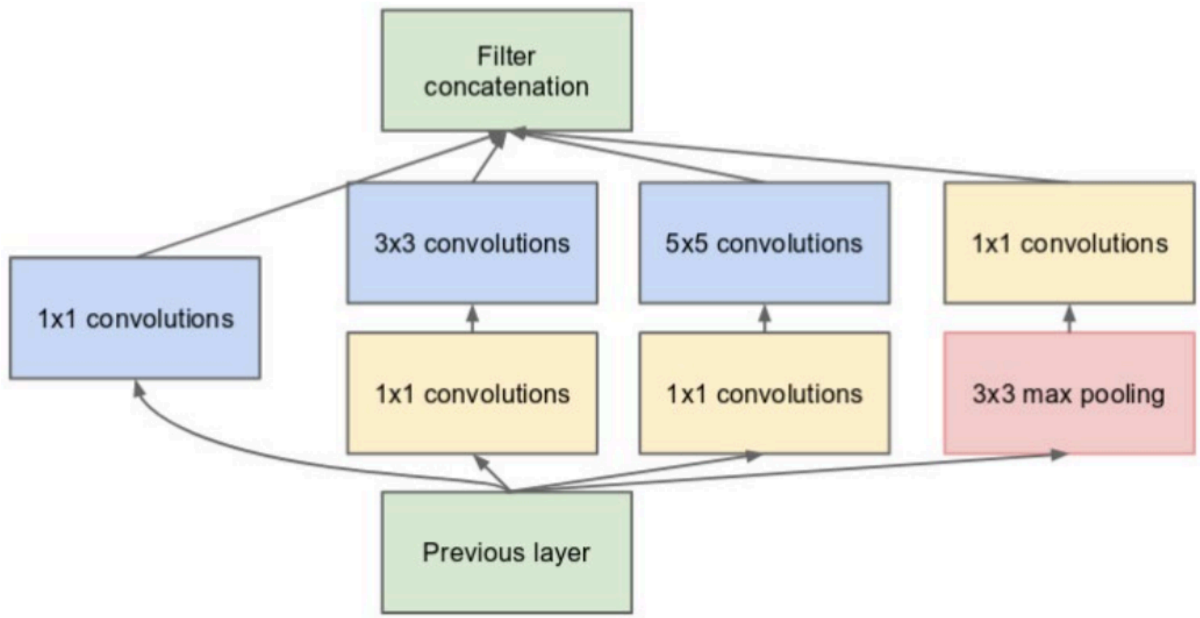


Large kernel for global features, and **smaller kernel** for local features.

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)

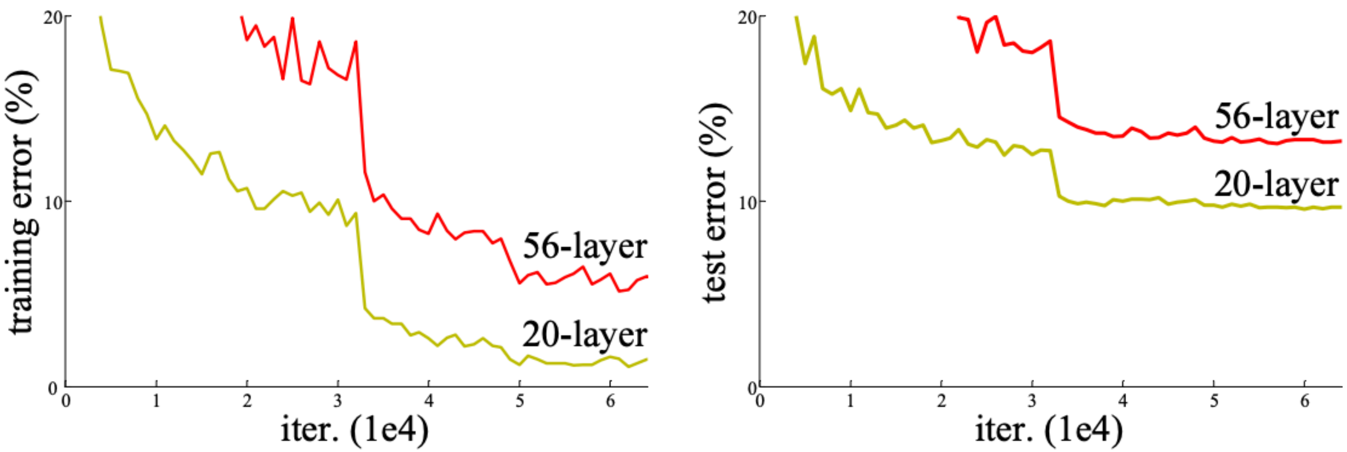


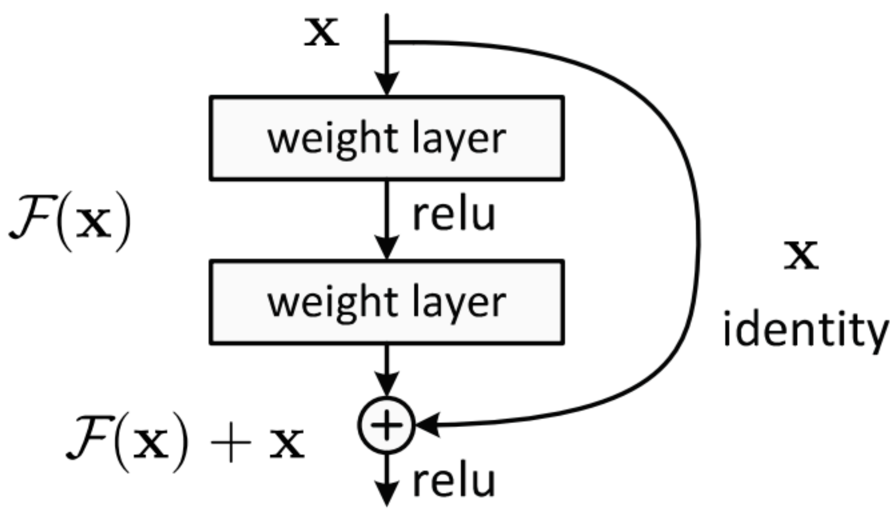
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

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

Residual Networks

Idea: identity shortcut, skip one or more layers.

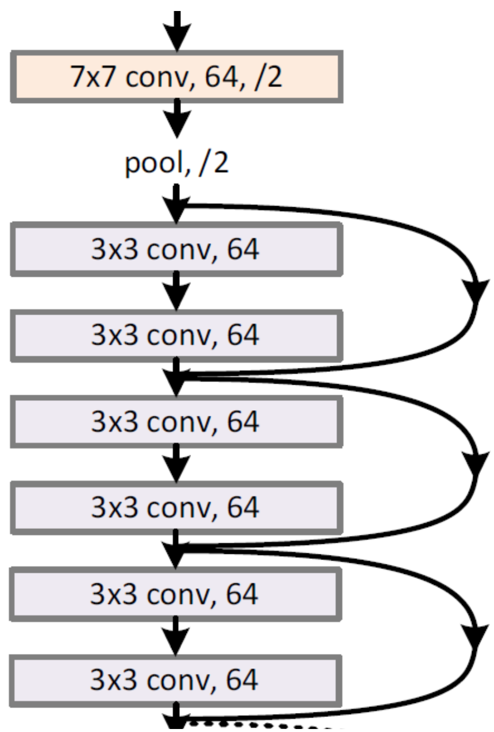
Justification: network can easily simulate shallow network ($F \approx 0$), so performance should not degrade by going deeper.

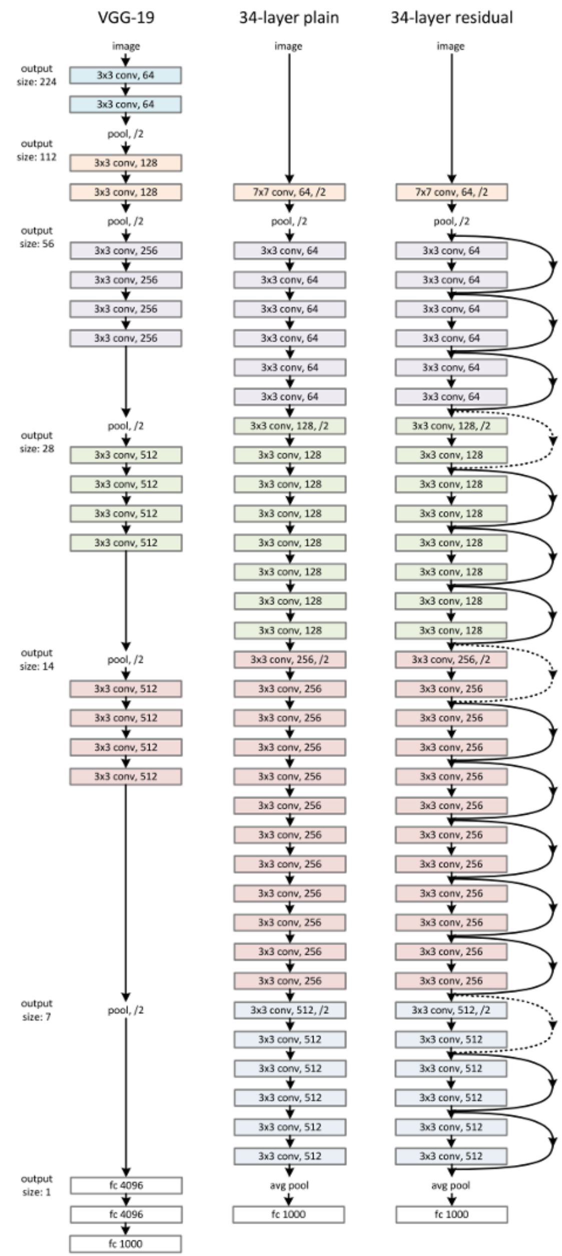


[He, Zhang, Ren, Sun, '16]

Residual Networks

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





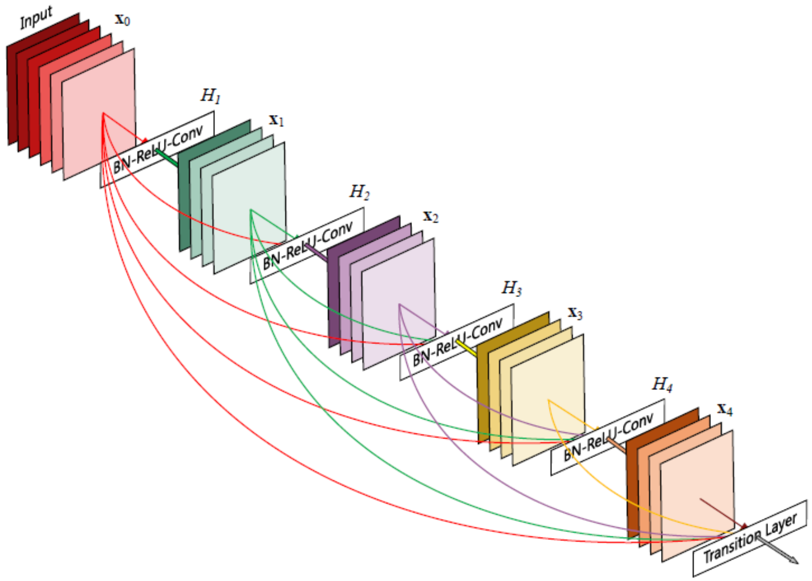
[He, Zhang, Ren, Sun, ’16]

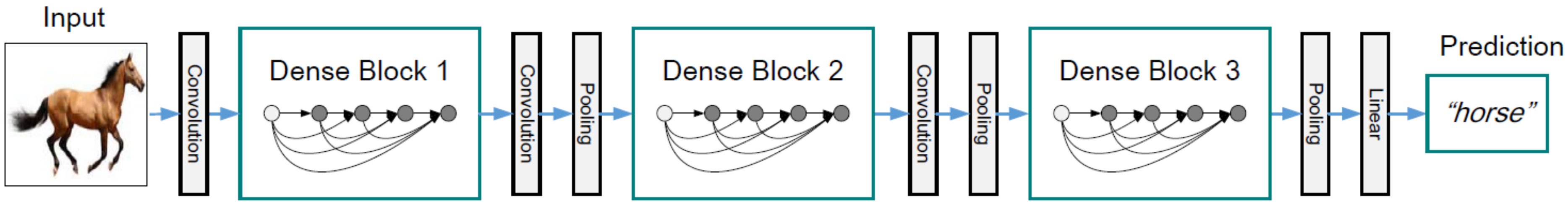
Densely Connected Network

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]
- …