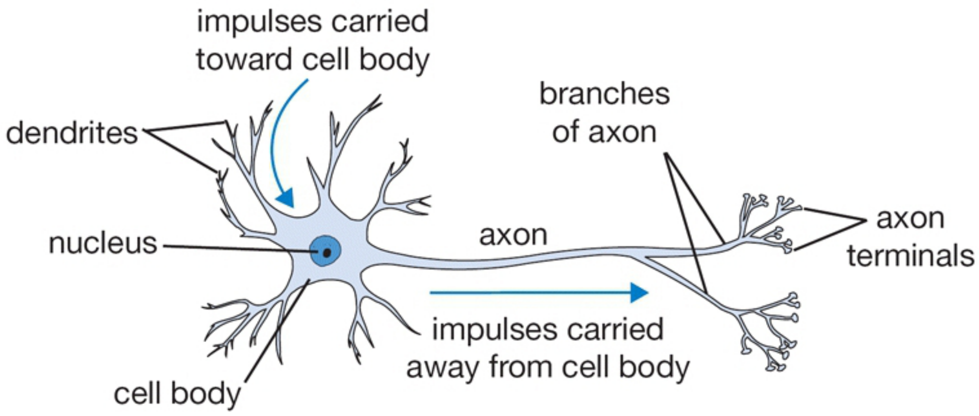
Introduction to Neural Networks

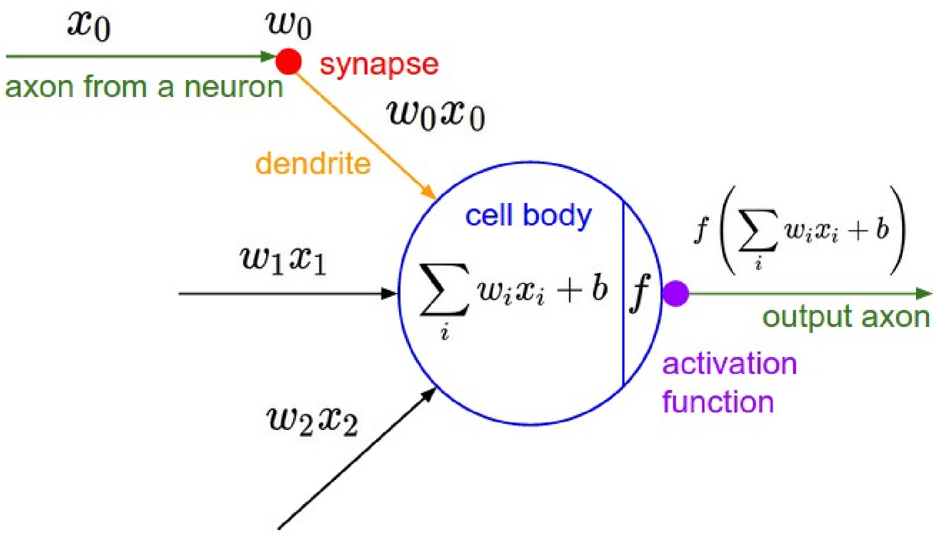
Jakob Verbeek

2017-2018

Biological motivation

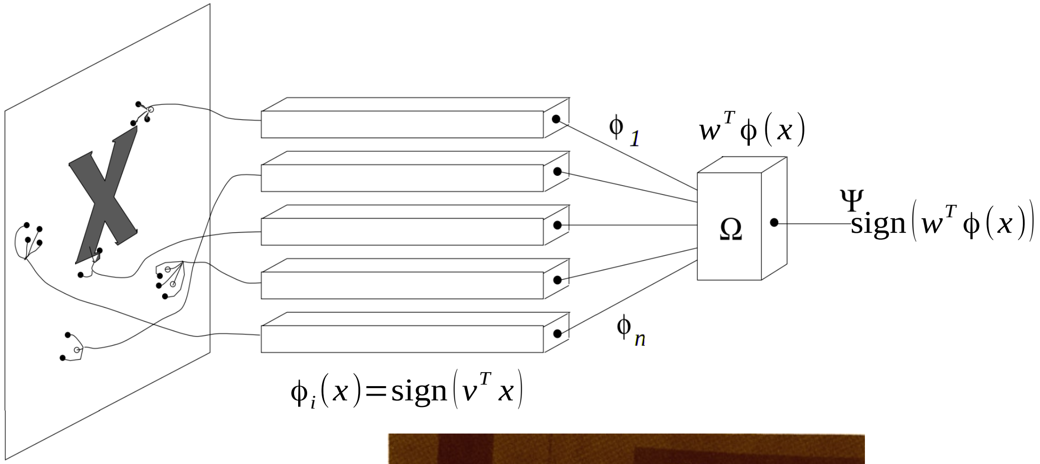
- Neuron is basic computational unit of the brain
  - about 10^11 neurons in human brain
- Simplified neuron model as linear threshold unit (McCulloch & Pitts, 1943)
  - Firing rate of electrical spikes modeled as continuous output quantity
  - Connection strength modeled by multiplicative weight
  - Cell activation given by sum of inputs
  - Output is non linear function of activation
- Basic component in neural circuits for complex tasks



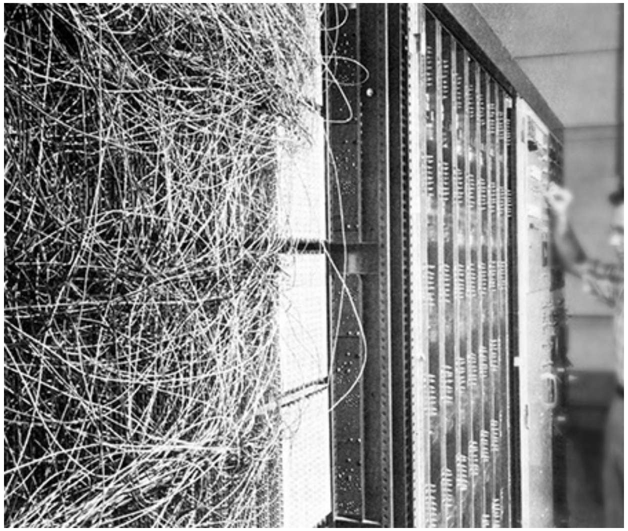


1957: Rosenblatt's Perceptron

- Binary classification based on sign of generalized linear function
  - Weight vector w learned using special purpose machines
  - Fixed associative units in first layer, sign activation prevents learning



20x20 pixel sensor



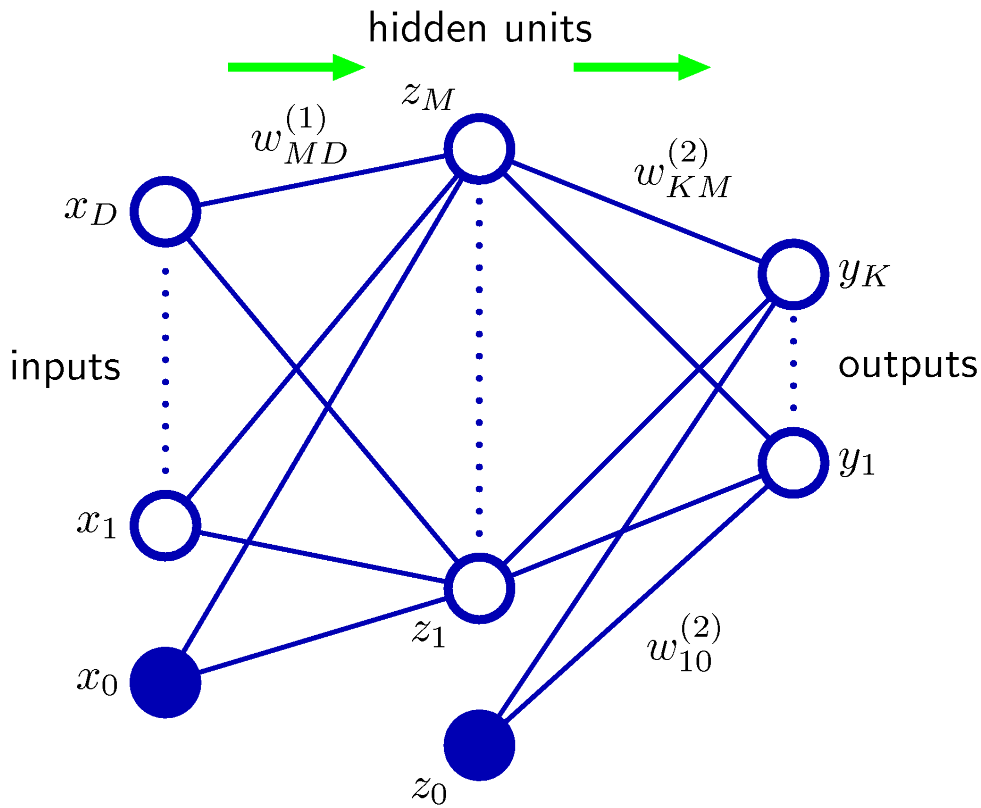
Random wiring of associative units

Multi-Layer Perceptron (MLP)

- Instead of using a generalized linear function, learn the features as well
- Each unit in MLP computes
  - Linear function of features in previous layer
  - Followed by scalar non-linearity
- Do **not** use the “step” non-linear activation function of original perceptron

$$z_j = h\left(\sum_i x_i w_{ij}^{(1)}\right)$$
$$z = h\left(W^{(1)} x\right)$$

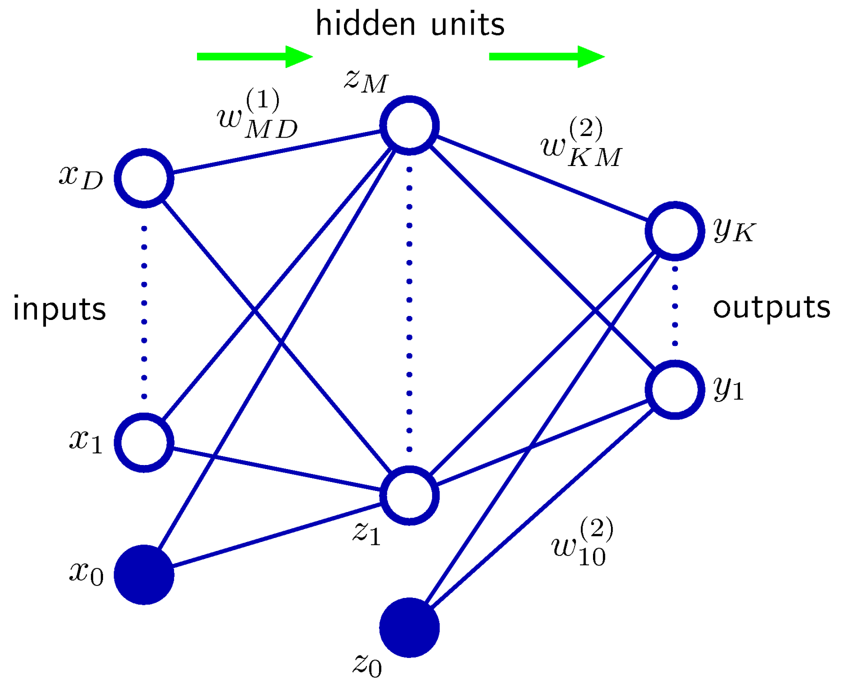
$$y_k = \sigma\left(\sum_j z_j w_{jk}^{(2)}\right)$$
$$y = \sigma\left(W^{(2)} z\right)$$



Multi-Layer Perceptron (MLP)

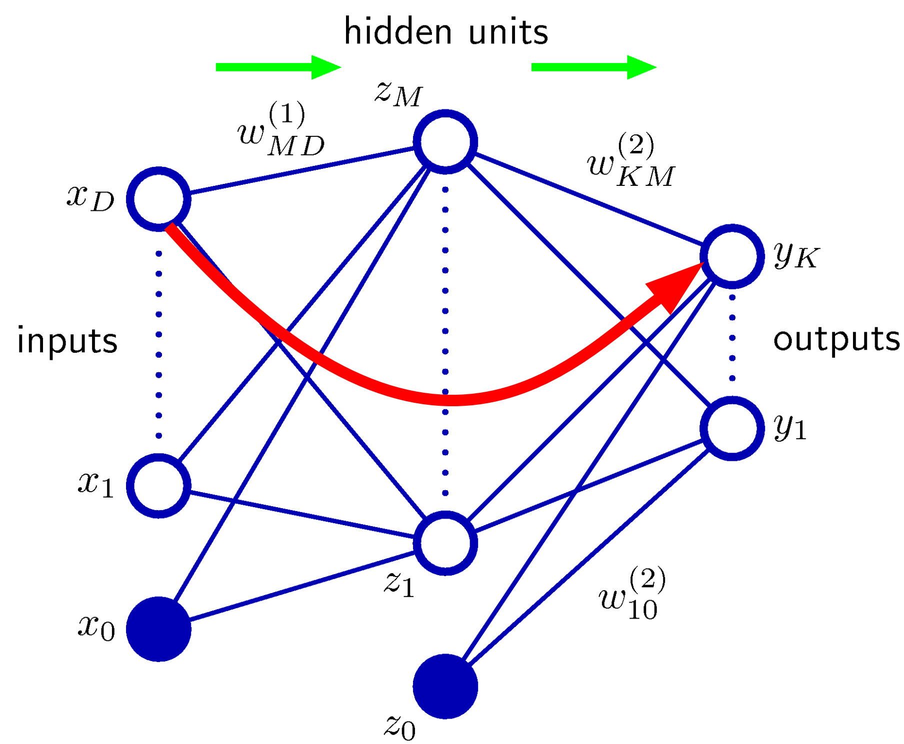
- Linear activation function leads to composition of linear functions
  - Remains a linear model, layers just induce a certain factorization

- Two-layer MLP can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units
  - Holds for many non-linearities, but not for polynomials



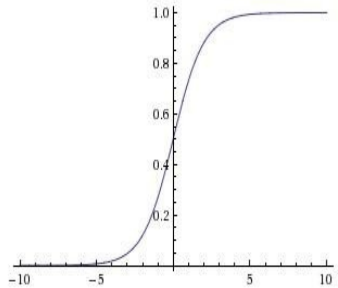
Feed-forward neural networks

- MLP Architecture can be generalized
  - More than two layers of computation
  - Skip-connections from previous layers
- Feed-forward nets are restricted to directed acyclic graphs of connections
  - Ensures that output can be computed from the input in a single feed-forward pass from the input to the output
- Important issues in practice
  - Designing network architecture
    - Nr nodes, layers, non-linearities, etc
  - Learning the network parameters
    - Non-convex optimization
  - Sufficient training data
    - Data augmentation, synthesis

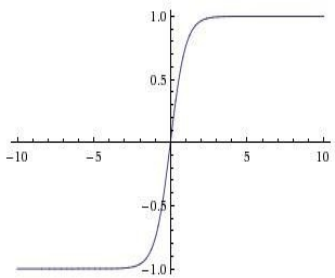


Activation functions

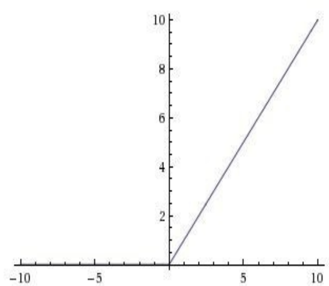
Sigmoid $1/(1+e^{-x})$



tanh

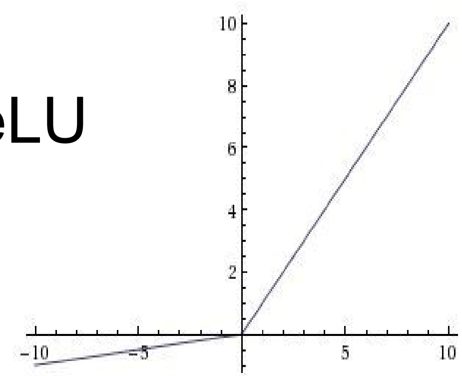


ReLU $\max(0, x)$

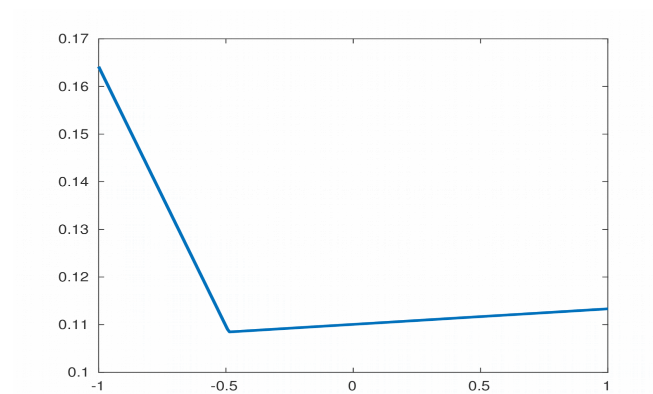


Leaky ReLU

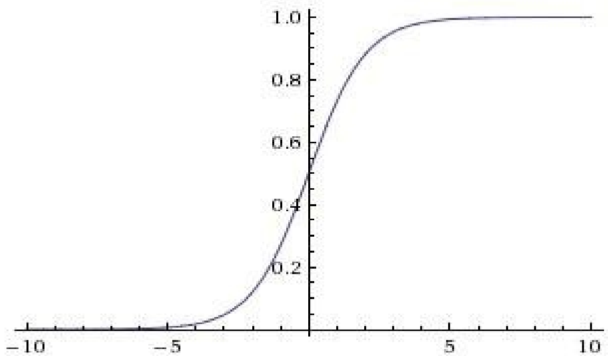
$\max(\alpha x, x)$



Maxout $\max(w_1^T x, w_2^T x)$

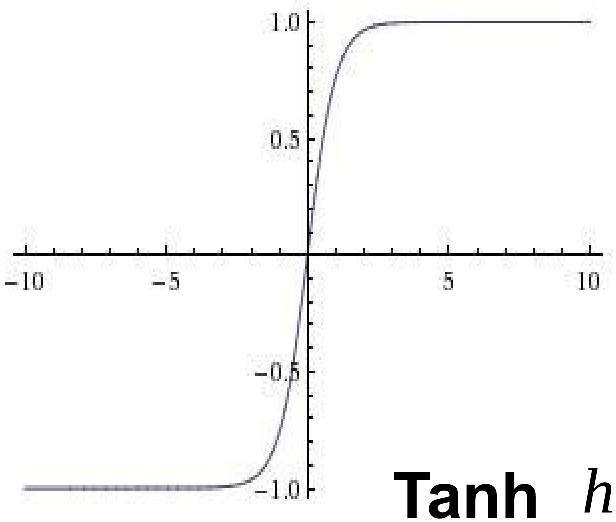


Activation Functions



- Squashes reals to range [0,1]
- Tanh outputs centered at zero: [-1, 1]
- Smooth step function
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

Sigmoid $\sigma(x) = 1/(1+e^{-x})$

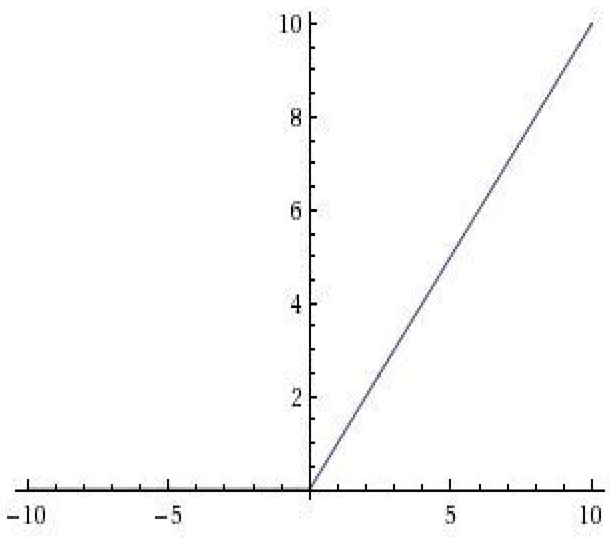


1. Saturated neurons "kill" the gradients, need activations to be exactly in right regime to obtain non-constant output
2. exp() is a bit compute expensive

Tanh $h(x) = 2\sigma(x) - 1$

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Activation Functions



ReLU
(Rectified Linear Unit)

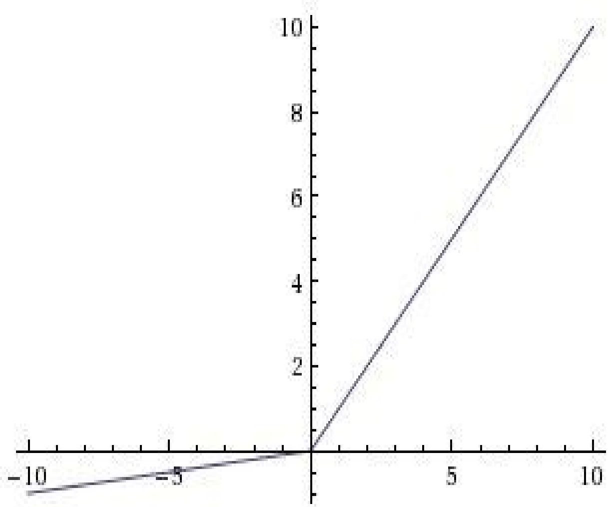
Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Most commonly used today

[Nair & Hinton, 2010]

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Activation Functions



Leaky ReLU

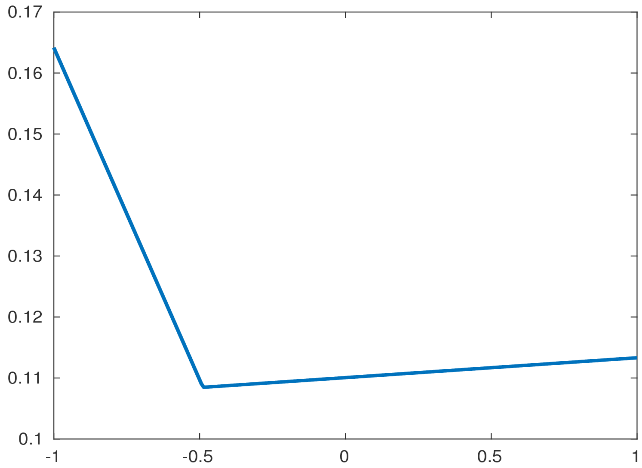
- Does not saturate: will not "die"
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)

$$f(x) = \max(0.01x, x)$$

[Mass et al., 2013] [He et al., 2015]

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Activation Functions



Maxout

$$\max\left(w_1^T x, w_2^T x\right)$$

- Does not saturate: will not "die"
- Computationally efficient

- Maxout networks can implement ReLU networks and vice-versa
- More parameters per node

[Goodfellow et al., 2013]

Training feed-forward neural network

- Non-convex optimization problem in general
  - Typically number of weights is very large (millions in vision applications)
  - Seems that many different local minima exist with similar quality

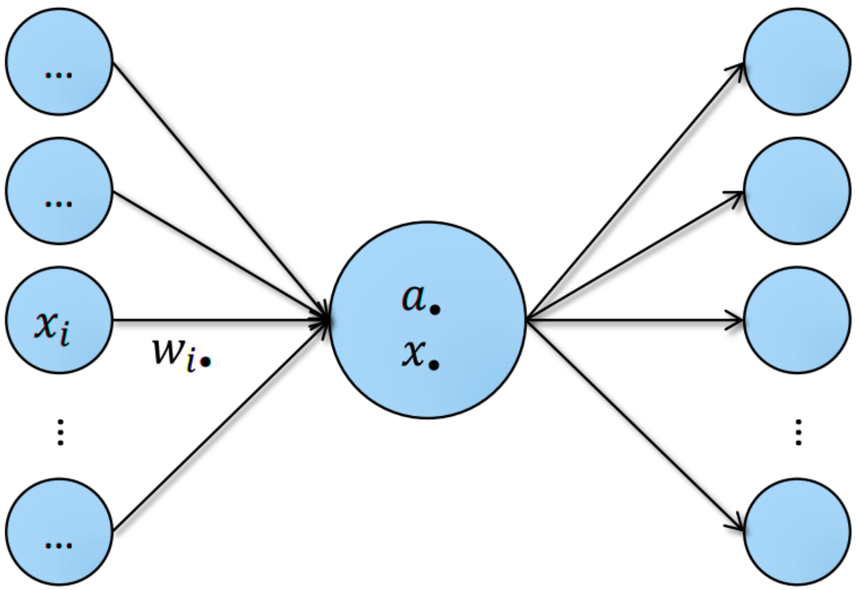
$$\frac{1}{N}\sum_{i=1}^{N} L\left(f\left(x_i\right), y_i; W\right) + \lambda\,\Omega\left(W\right)$$

- Regularization
  - L2 regularization: sum of squares of weights
  - “Drop-out”: deactivate random subset of weights in each iteration
    - Similar to using many networks with less weights (shared among them)

- Training using simple gradient descend techniques
  - Stochastic gradient descend for large datasets (large N)
  - Estimate gradient of loss terms by averaging over a relatively small number of samples

Training the network: forward propagation

- Forward propagation from input nodes to output nodes
  - Accumulate inputs via weighted sum into activation
  - Apply non-linear activation function f to compute output
- Use Pre(j) to denote all nodes feeding into j



$$a_j = \sum_{i \in Pre(j)} w_{ij} x_i$$

$$x_j = f(a_j)$$

Training the network: backward propagation

- Node activation and output

$$a_j = \sum_{i \in Pre(j)} w_{ij} x_i$$

$$x_j = f(a_j)$$

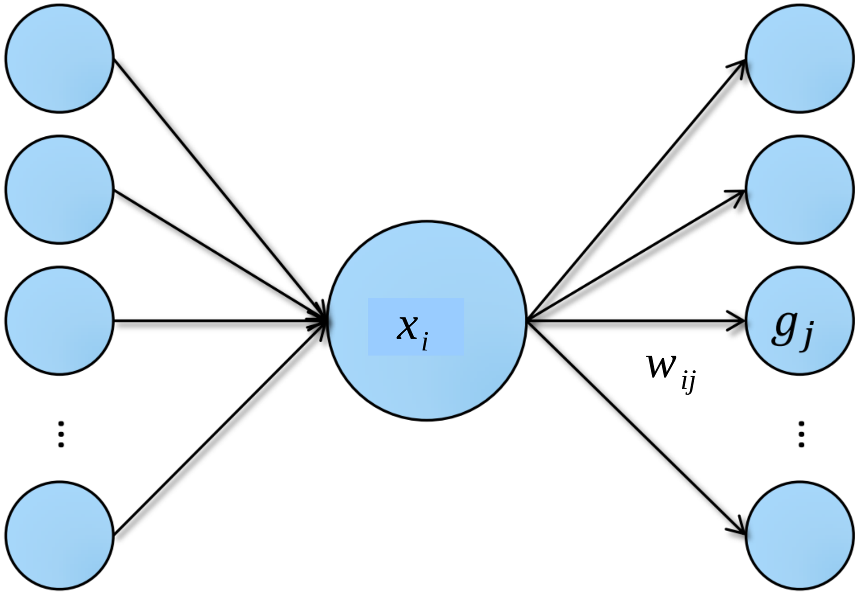
- Partial derivative of loss w.r.t. activation

$$g_j = \frac{\partial L}{\partial a_j}$$

- Partial derivative w.r.t. learnable weights

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = g_j x_i$$

- Gradient of weight matrix between two layers given by outer-product of x and g



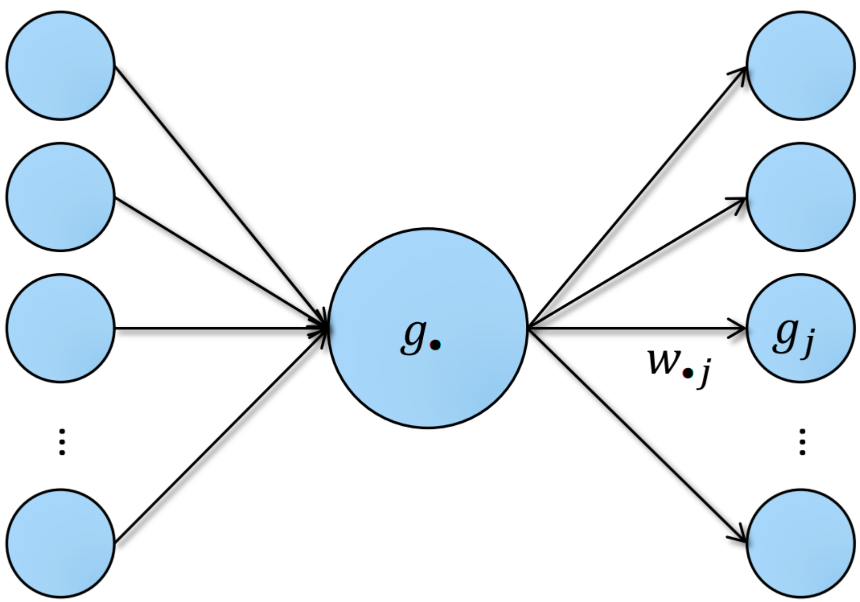
Training the network: backward propagation

- Back-propagation layer-by-layer of gradient from loss to internal nodes
  - Application of chain-rule of derivatives
- Accumulate gradients from downstream nodes
  - Post(i) denotes all nodes that i feeds into
  - Weights propagate gradient back
- Multiply with derivative of local activation function

$$a_j = \sum_{i \in Pre(j)} w_{ij} x_i$$

$$x_j = f(a_j)$$

$$g_i = \frac{\partial L}{\partial a_i}$$



$$\frac{\partial L}{\partial x_i} = \sum_{j \in Post(i)} \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial x_i} = \sum_{j \in Post(i)} g_j w_{ij}$$

$$g_i = \frac{\partial x_i}{\partial a_i} \frac{\partial L}{\partial x_i} = f'(a_i) \sum_{j \in Post(i)} w_{ij} g_j$$

Training the network: forward and backward propagation

- Special case for Rectified Linear Unit (ReLU) activations

$$f(a) = max(0, a)$$

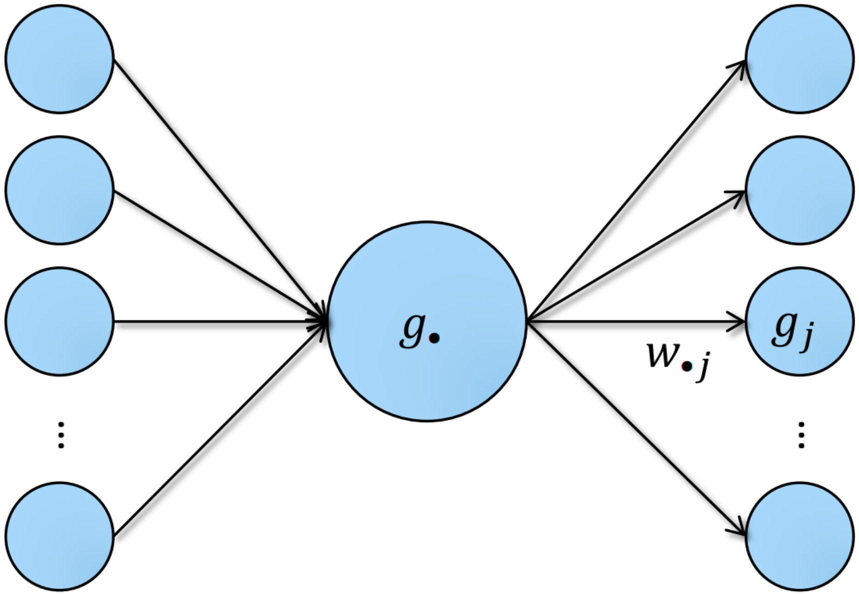
- Sub-gradient is step function

$$f'(a) = \begin{cases} 0 & \text{if } a \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

- Sum gradients from downstream nodes

$$g_i = \begin{cases} 0 & \text{if } a_i \leq 0 \\ \sum_{j \in Post(i)} w_{ij} g_j & \text{otherwise} \end{cases}$$

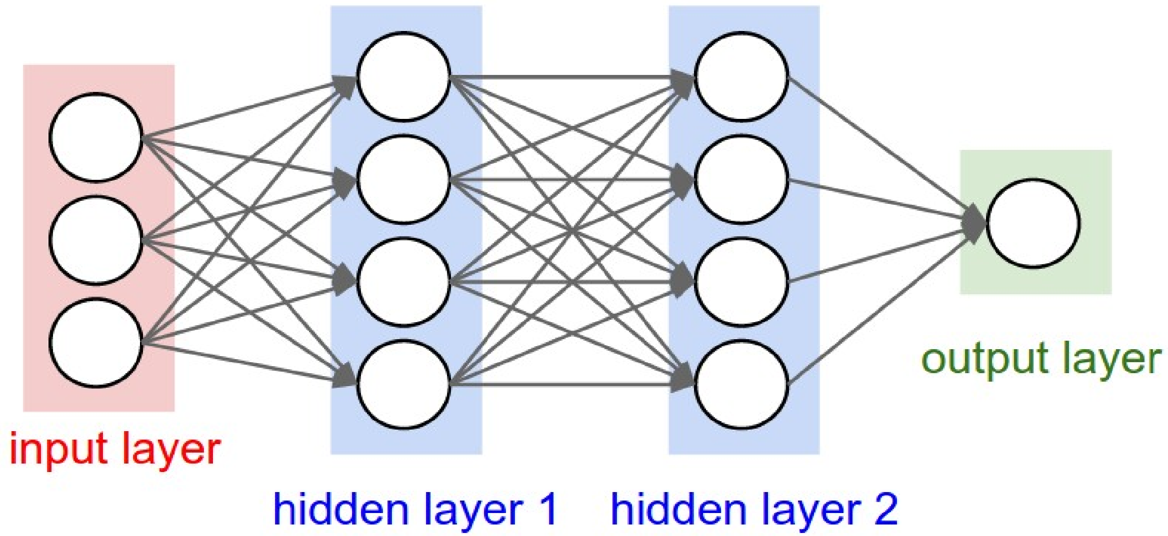
  - Set to zero if in ReLU zero-regime
  - Compute sum only for active units



- Gradient on incoming weights is “killed” by inactive units
  - Generates tendency for those units to remain inactive

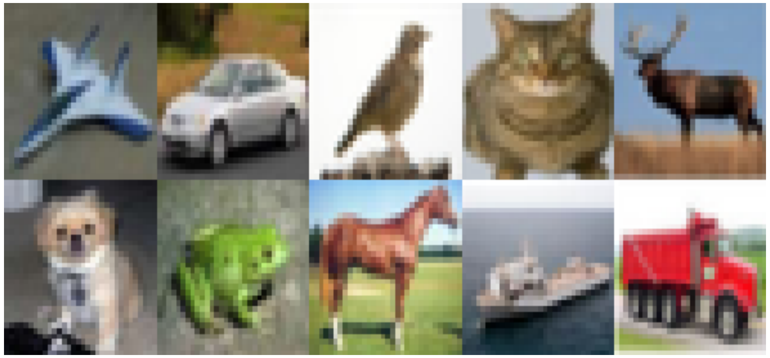
$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = g_j x_i$$

Convolutional Neural Networks



How to represent the image at the network input?

Input example : an image

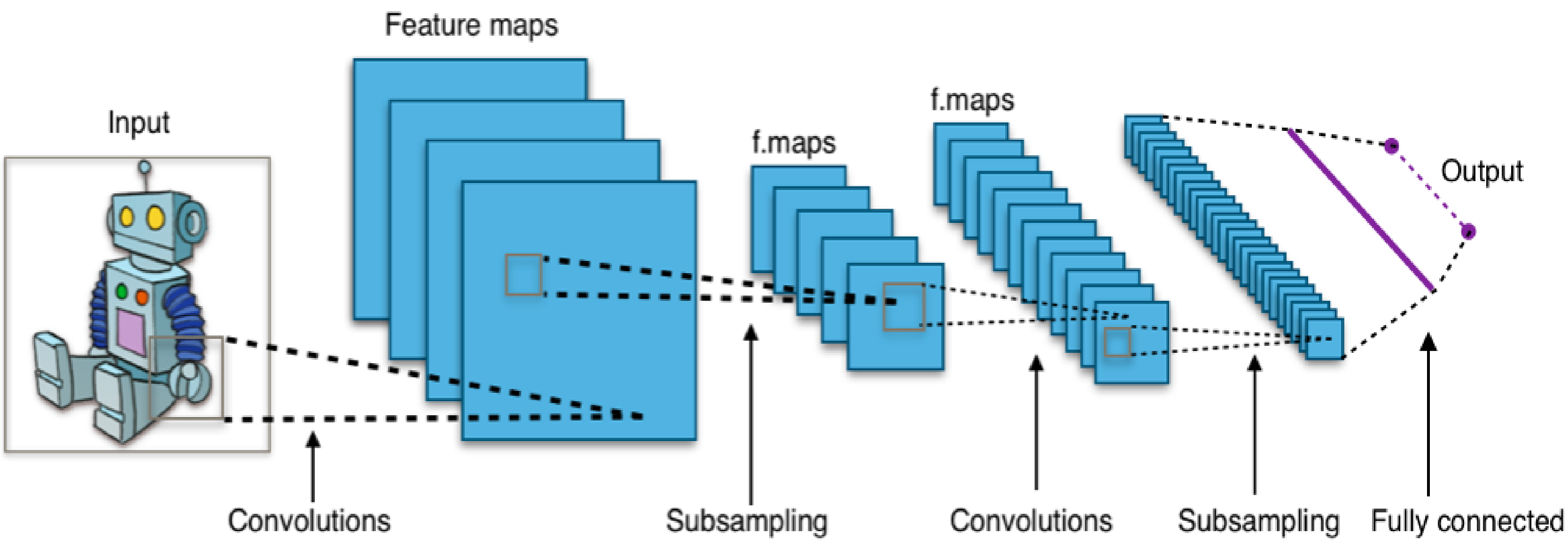


Output example: class label

airplane	dog
automobile	frog
bird	horse
cat	ship
deer	truck

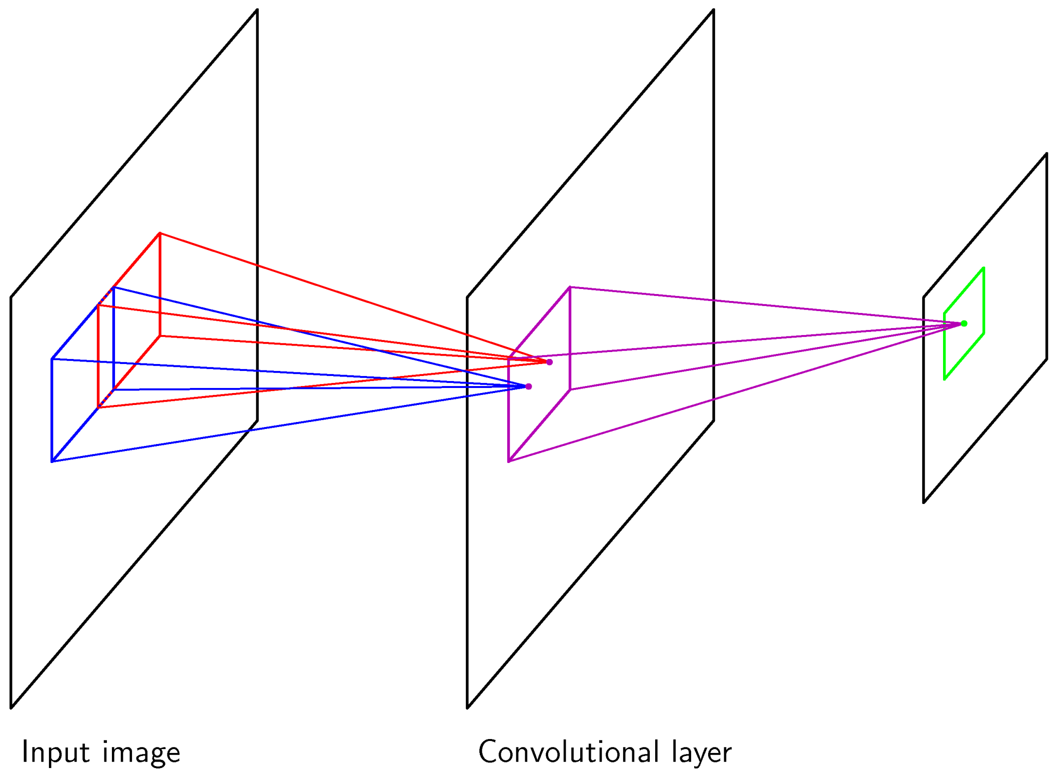
Convolutional neural networks

- A convolutional neural network is a feedforward network where
  - Hidden units are organizes into images or “response maps”
  - Linear mapping from layer to layer is replaced by convolution

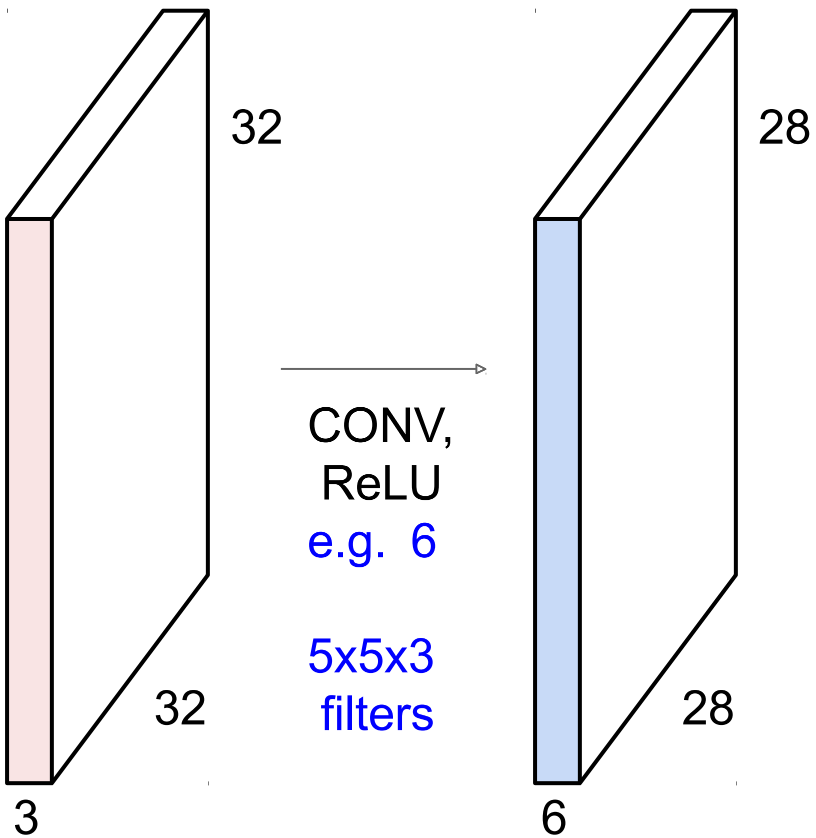


Convolutional neural networks

- Local connections: motivation from findings in early vision
  - Simple cells detect local features
  - Complex cells pool simple cells in retinotopic region
- Convolutions: motivated by translation invariance
  - Same processing should be useful in different image regions

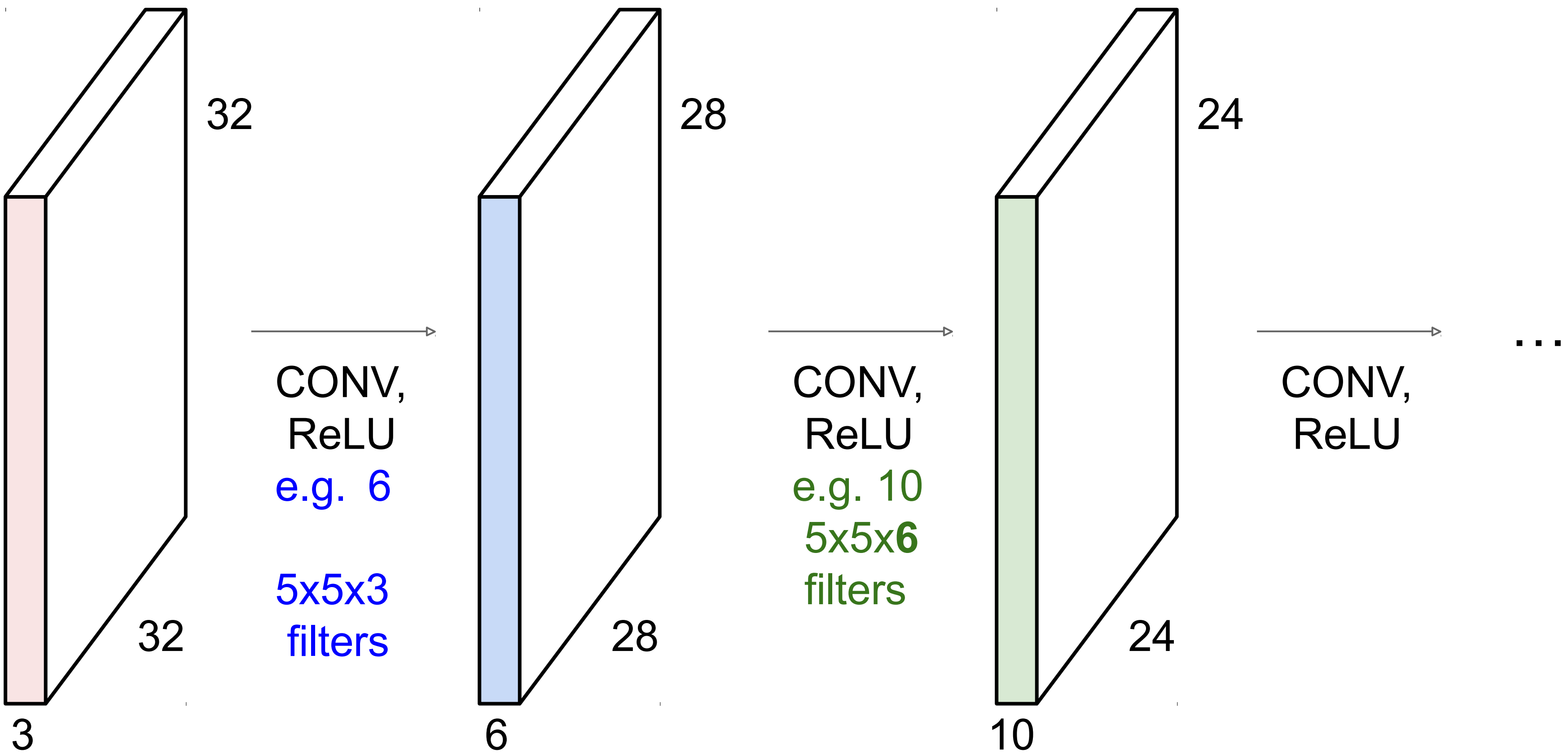


Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



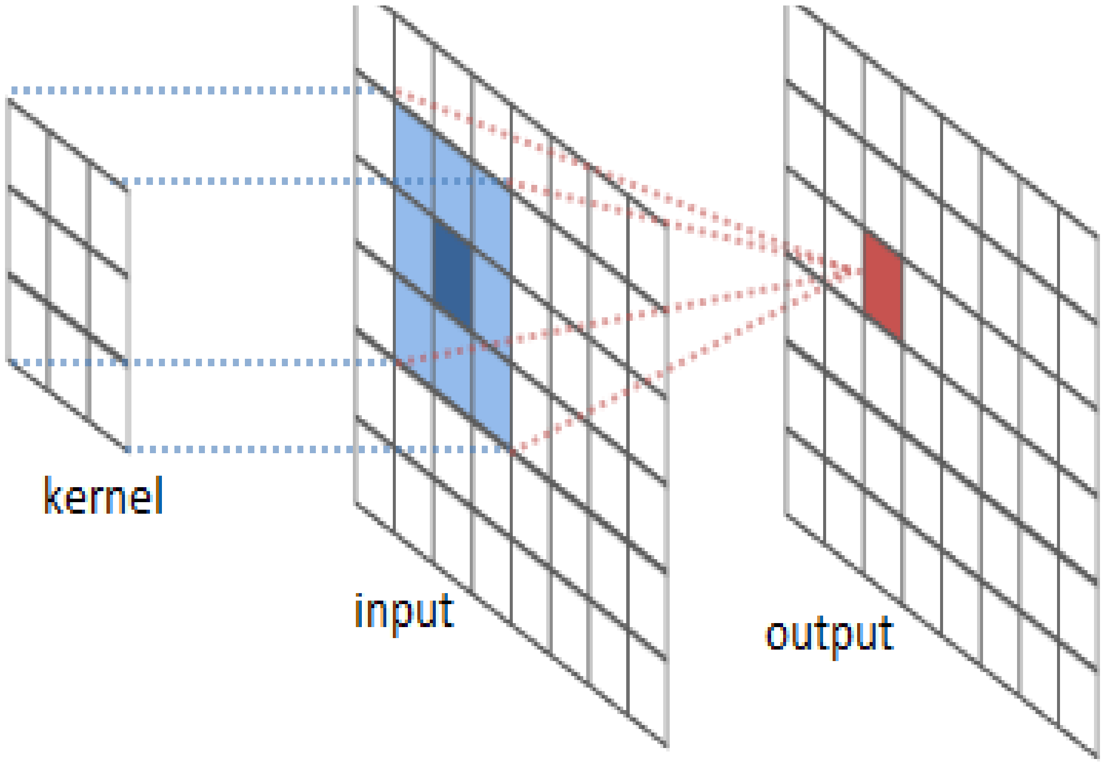
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

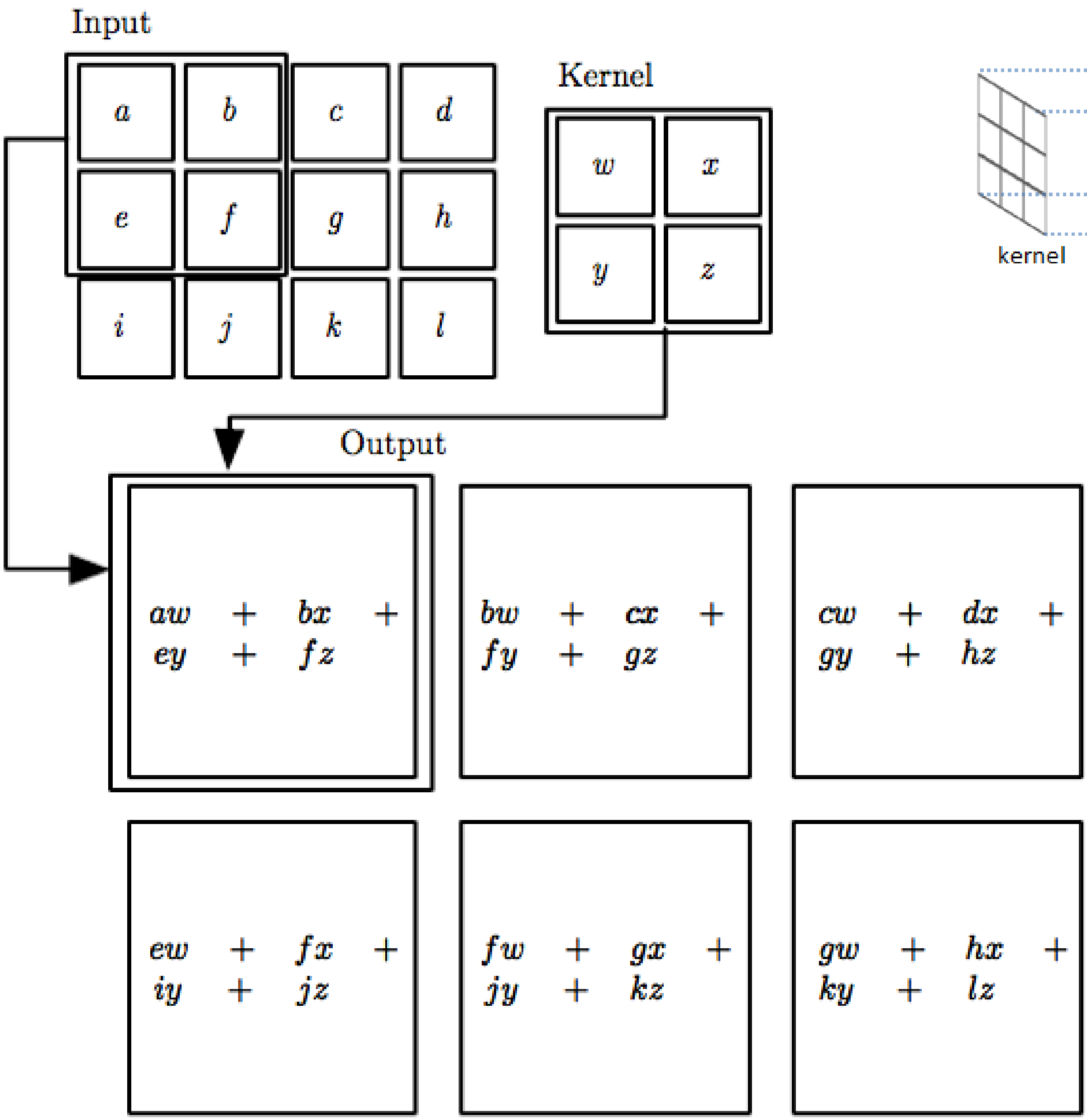


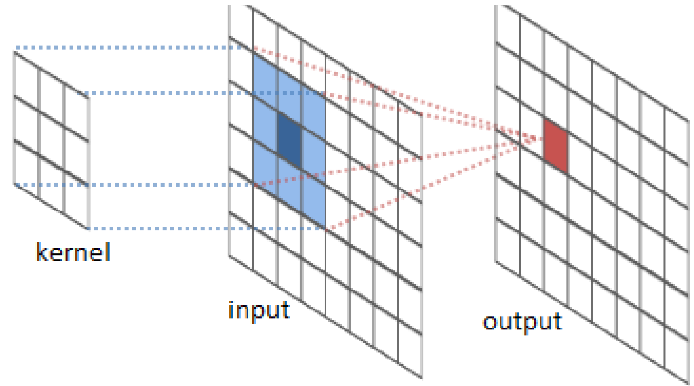
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

The convolution operation



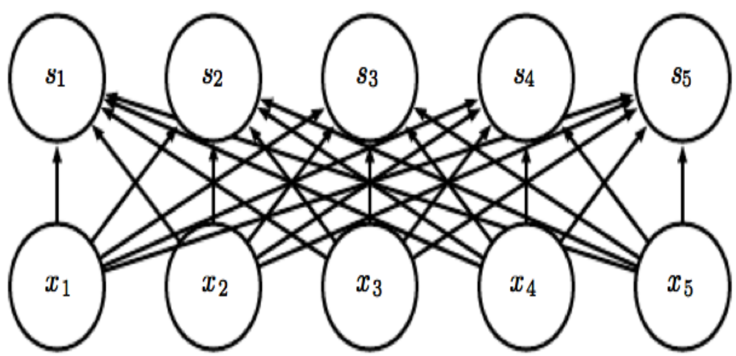
The convolution operation





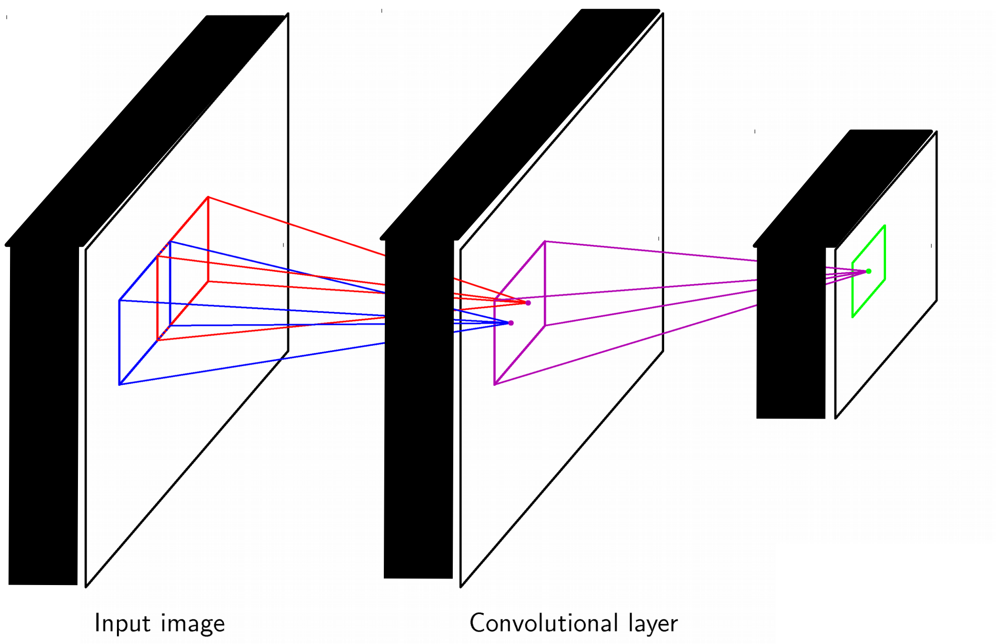
Local connectivity

Fully connected layer
as used in MLP



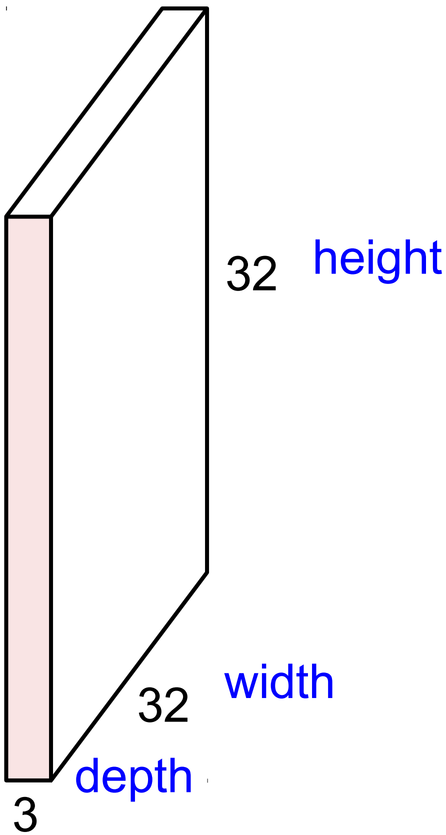
Convolutional neural networks

- Hidden units form another “image” or “response map”
  - Followed by point-wise non-linearity as in MLP
- Both input and output of the convolution can have multiple channels
  - E.g. three channels for an RGB input image
- Sharing of weights across spatial positions decouples the number of parameters from input and representation size
  - Enables training of models for large input images



Convolution Layer

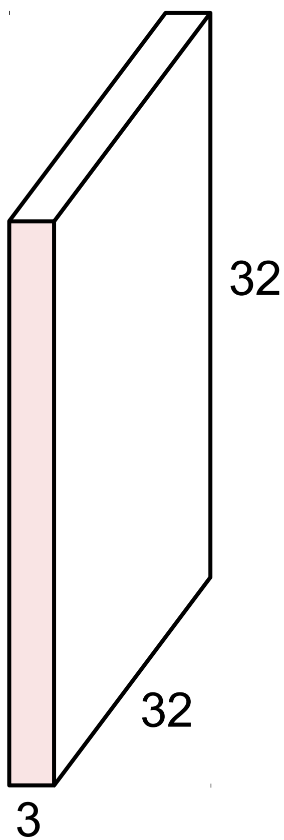
32x32x3 image



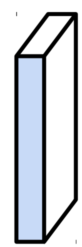
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolution Layer

32x32x3 image



5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

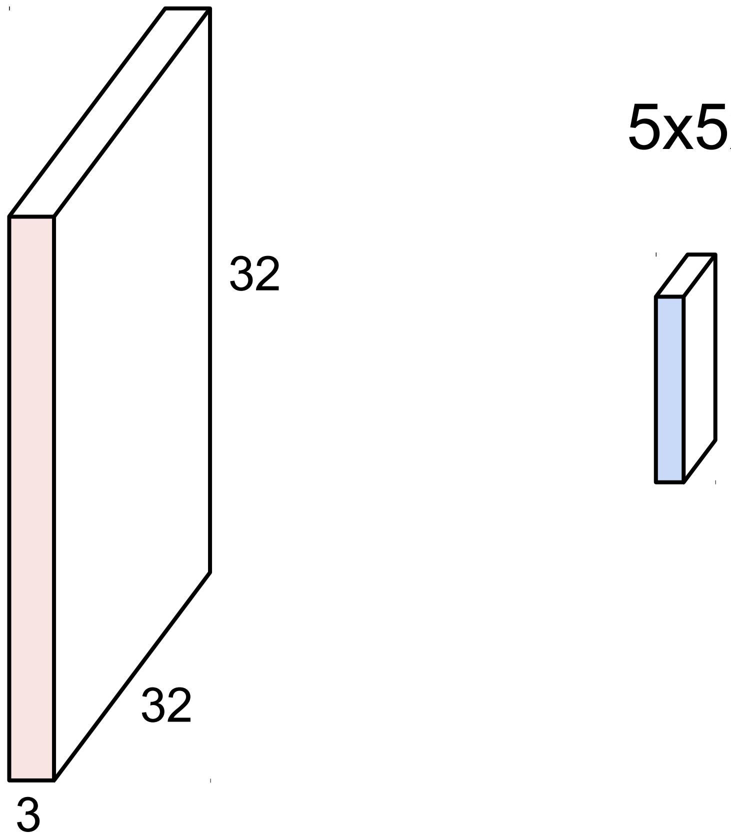
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolution Layer

32x32x3 image

Filters always extend the full depth of the input volume

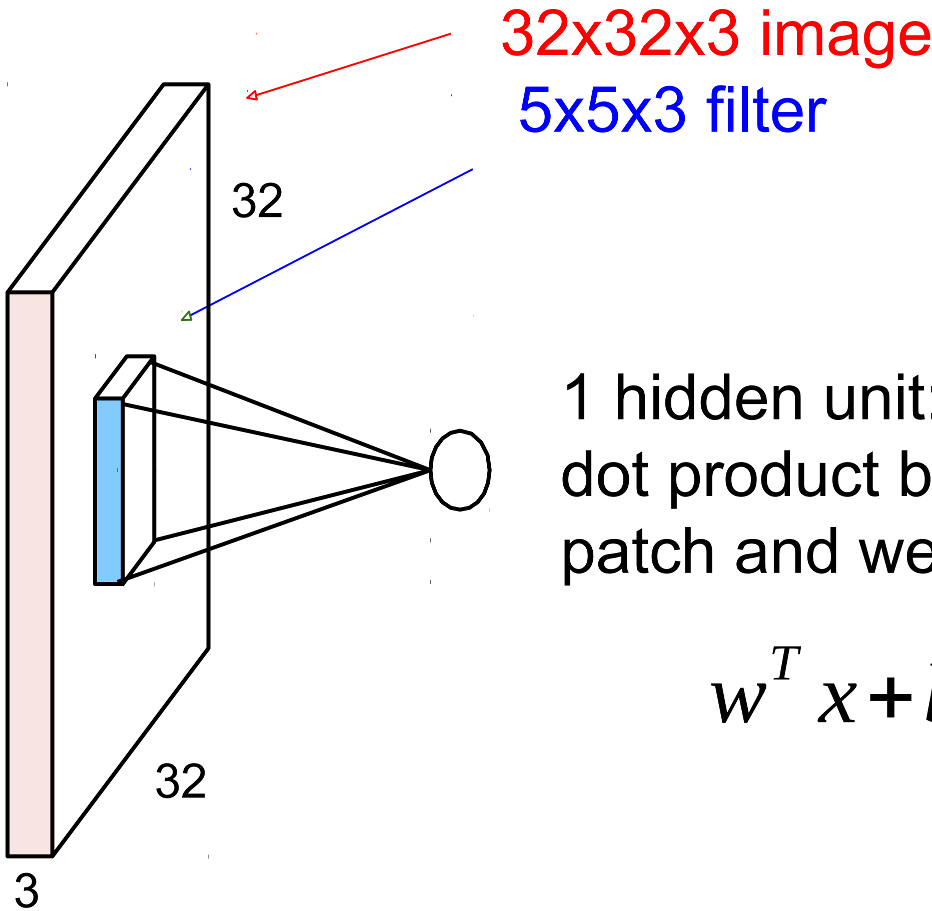
5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolution Layer

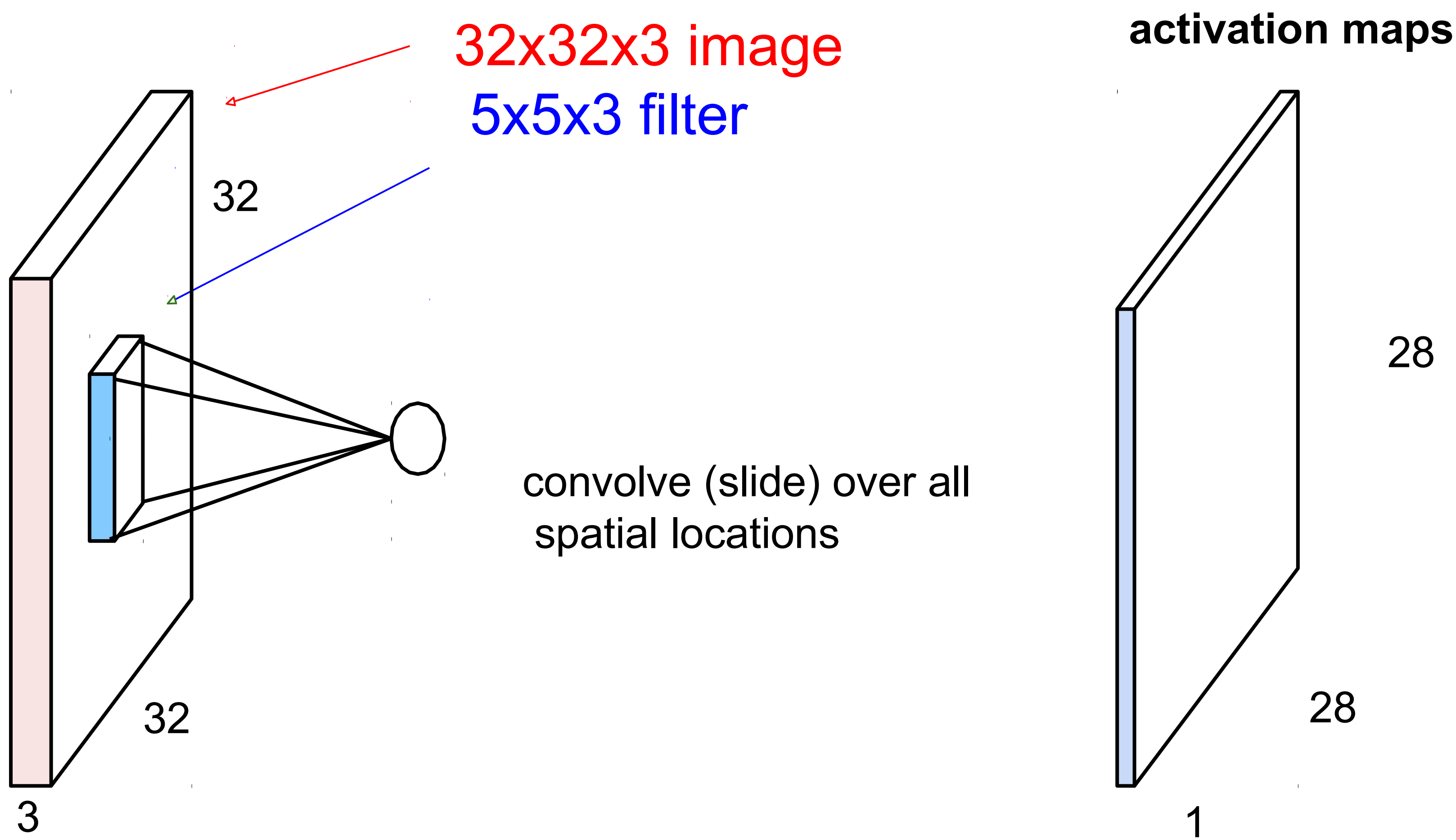


1 hidden unit:
dot product between 5x5x3=75 input patch and weight vector + bias

$$w^T x + b$$

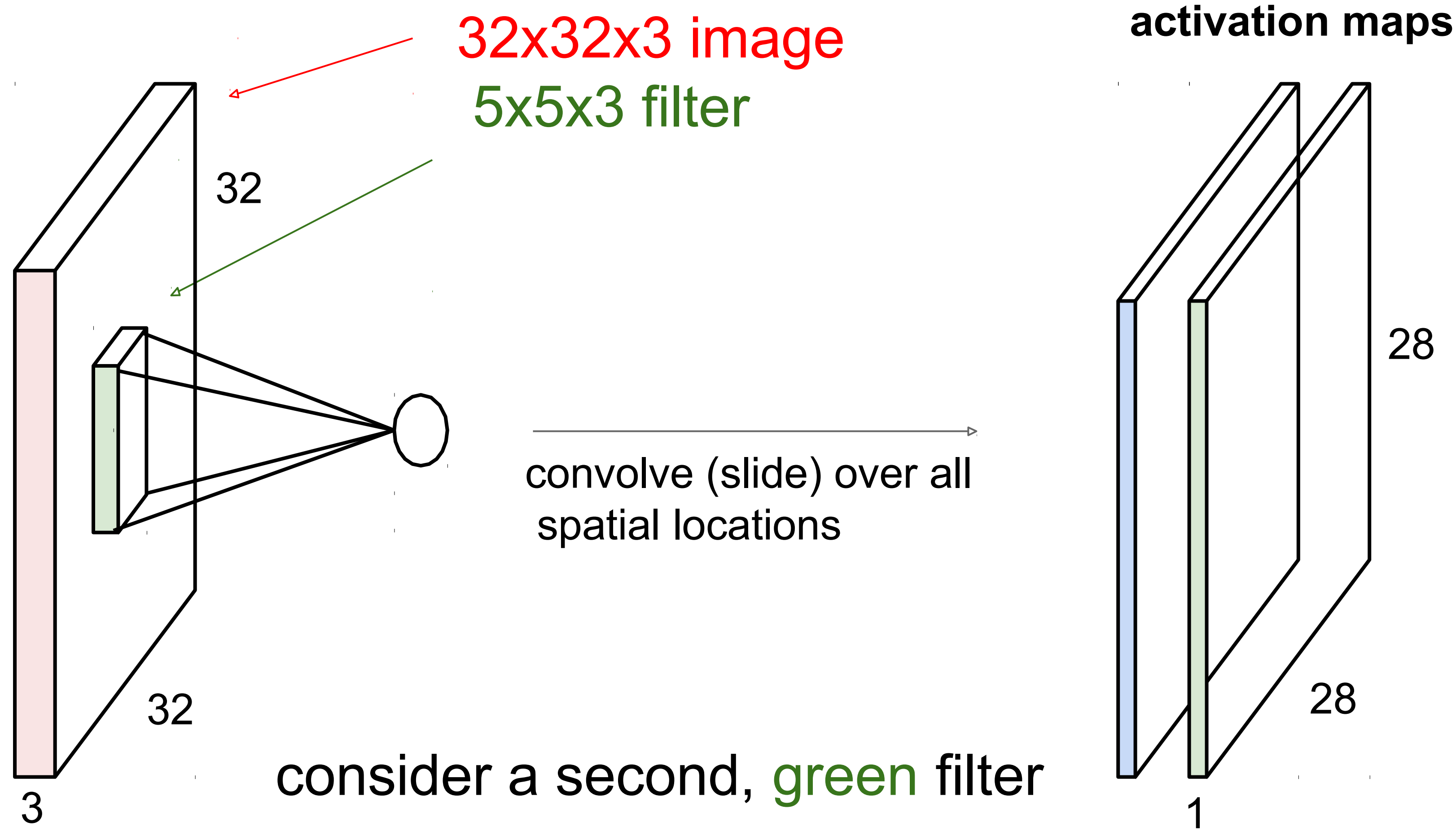
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolution Layer



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

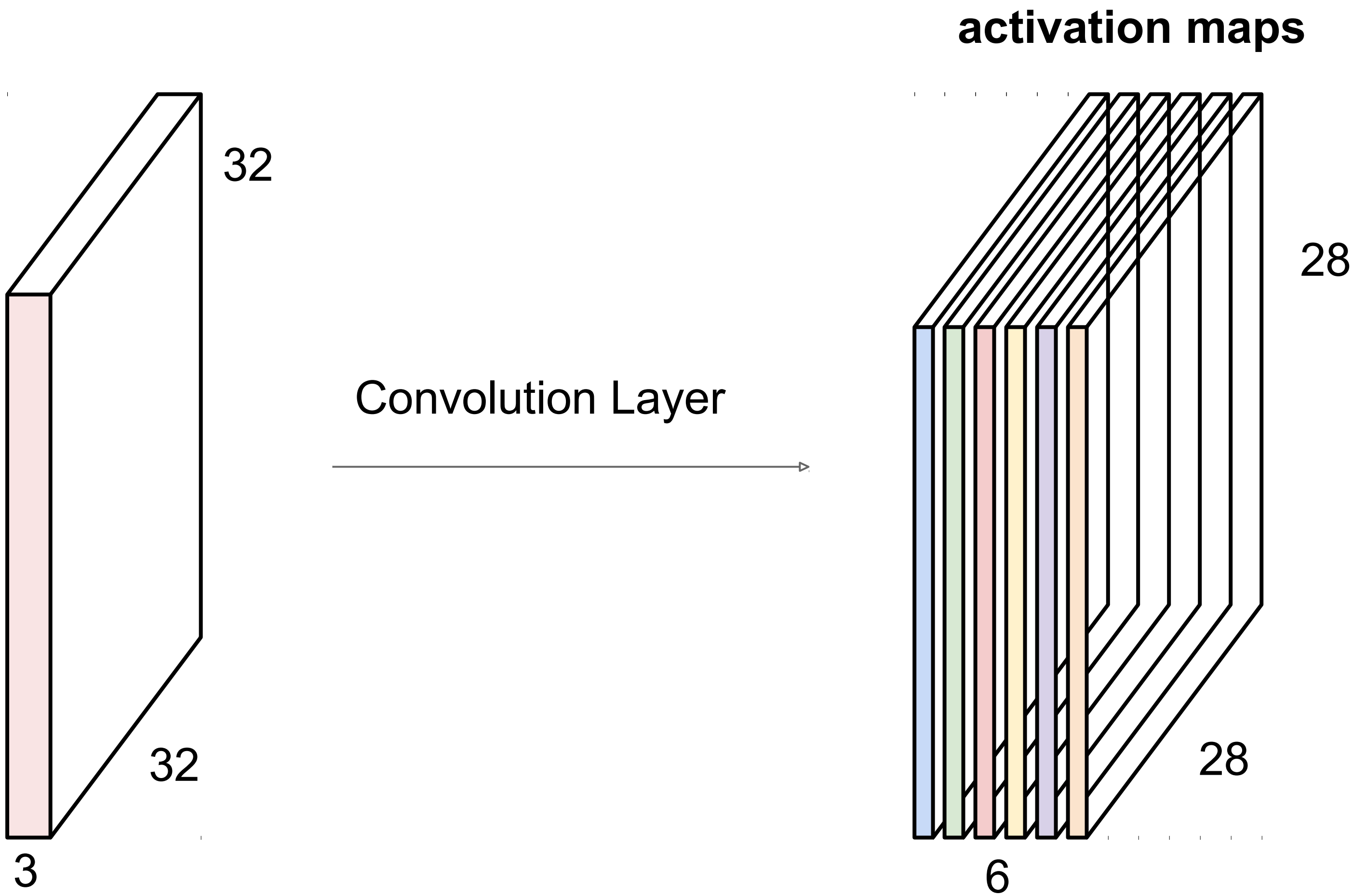
Convolution Layer



consider a second, green filter

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

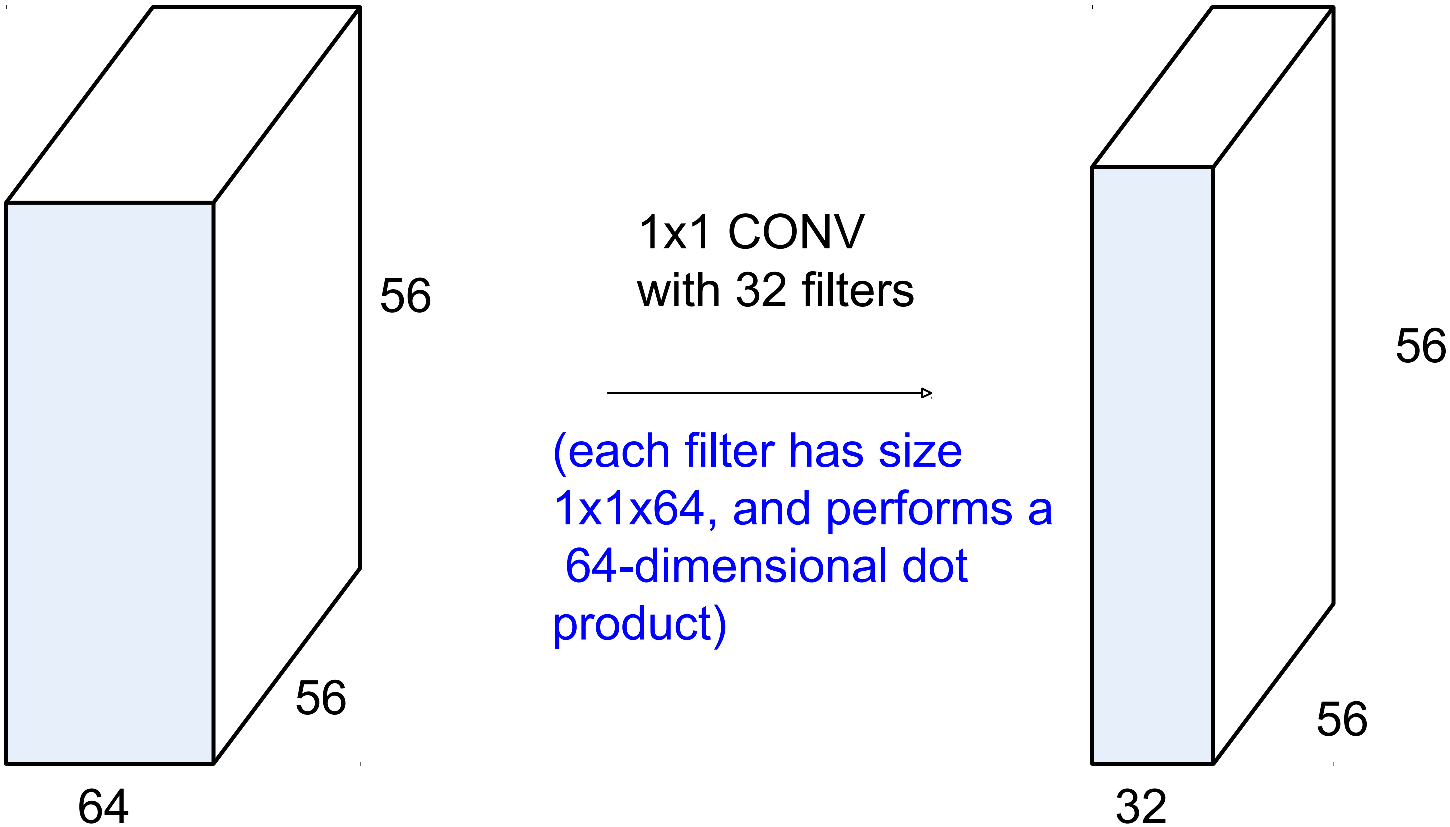
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

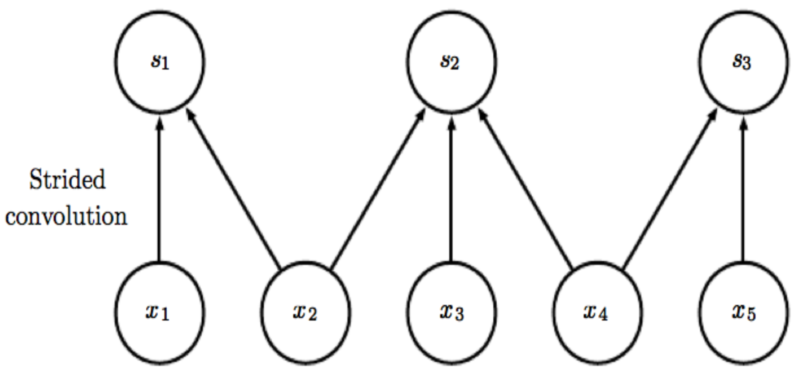
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

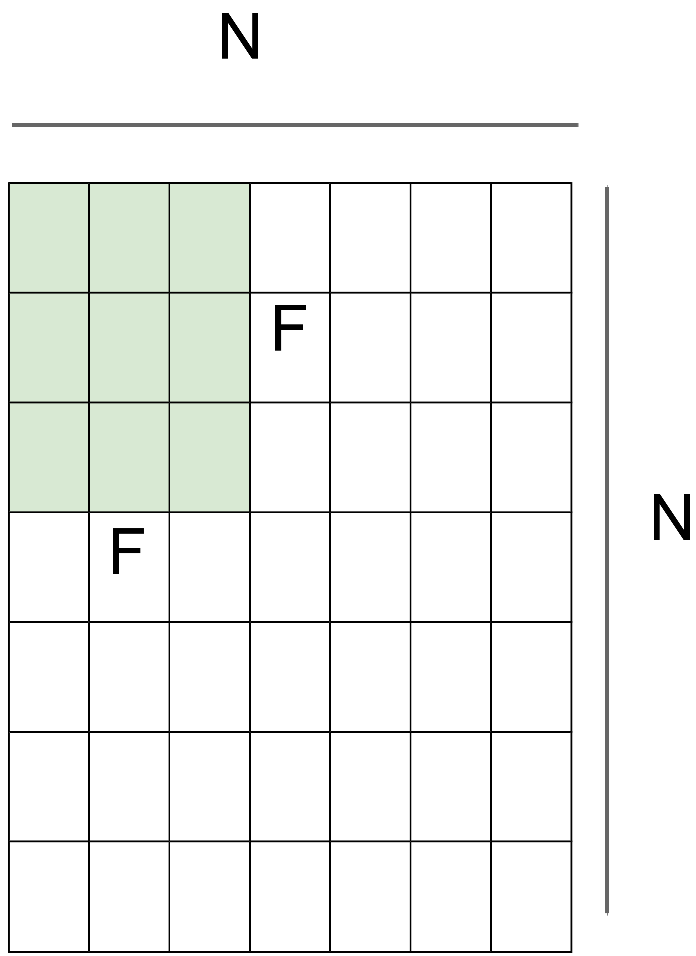
Convolution with 1x1 filters makes perfect sense



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Stride



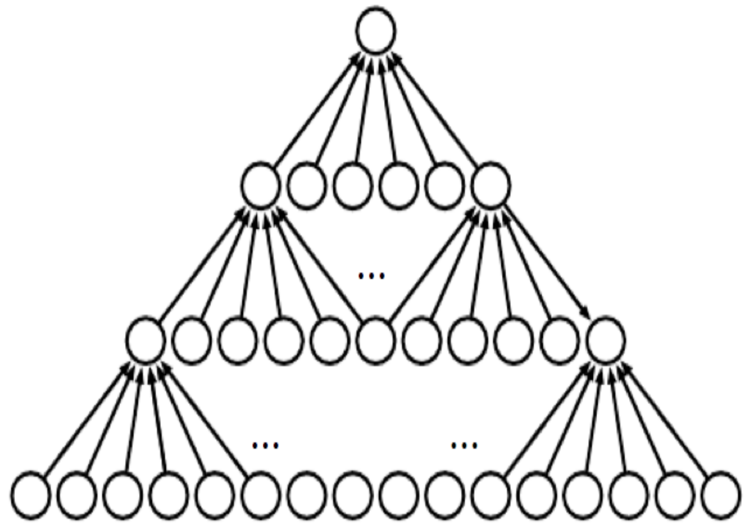


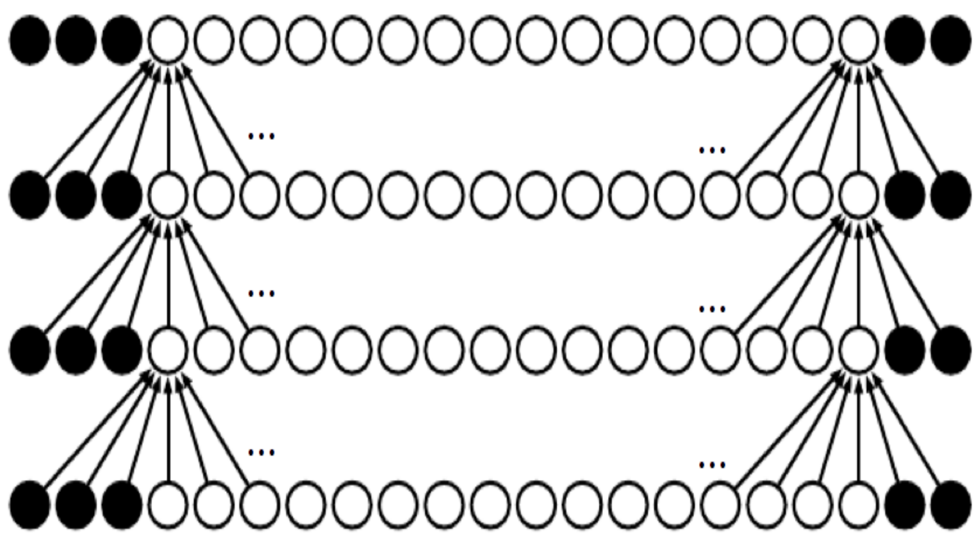
Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

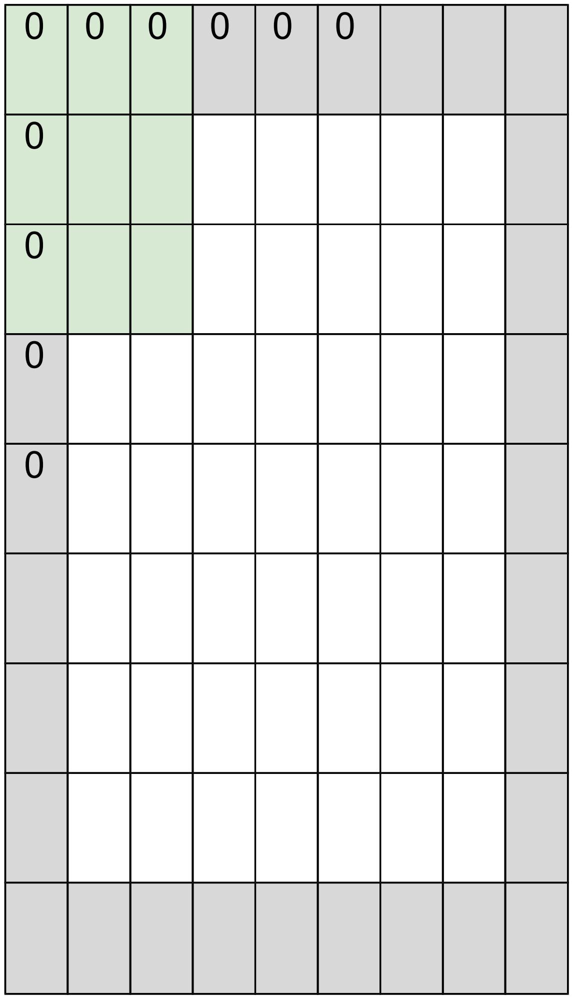
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

(Zero)-Padding





Zero-Padding: common to the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

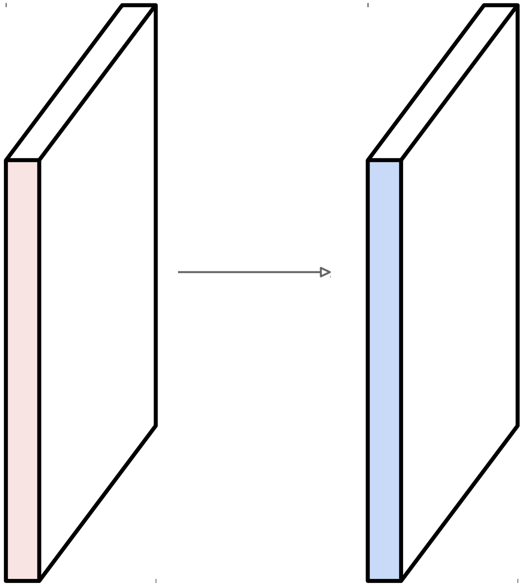
7x7 output!

In general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)

e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Example:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

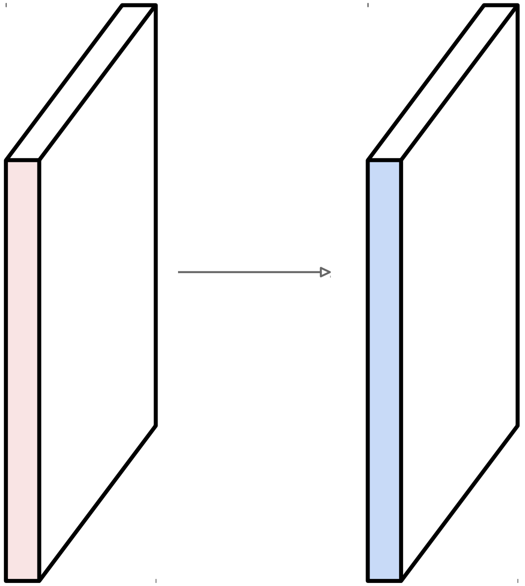
Output volume size: ?

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Example:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



Output volume size:

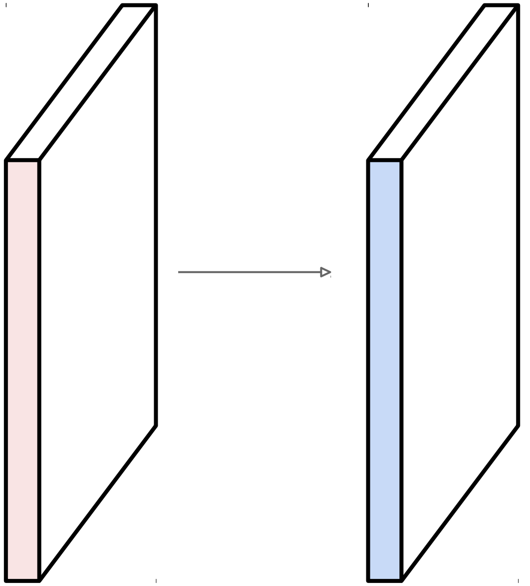
(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Example:

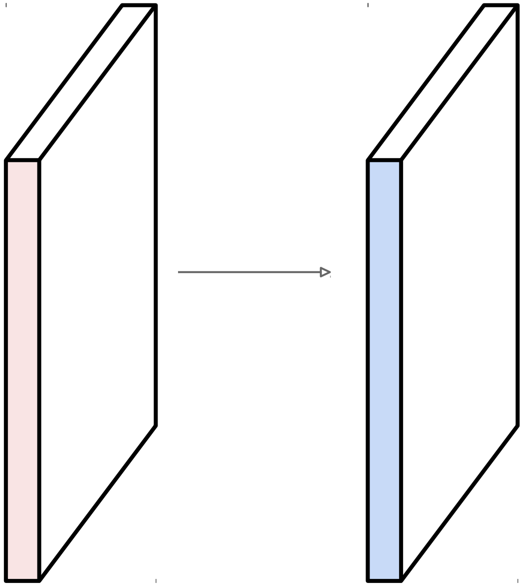
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Example:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params

(+1 for bias)

=> 76*10 = **760**

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

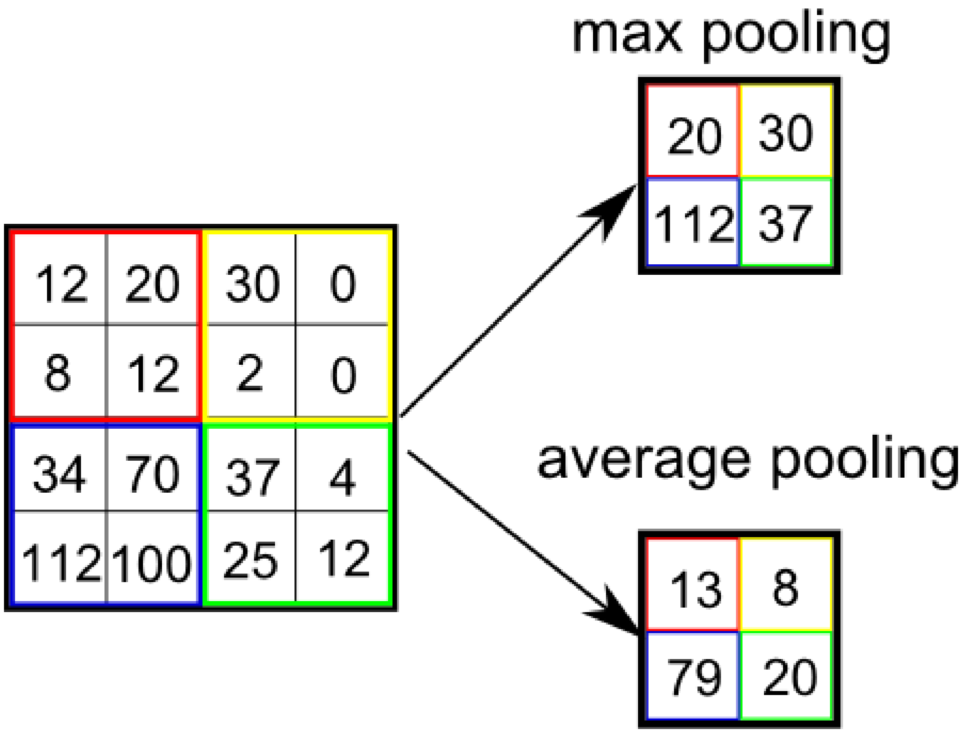
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

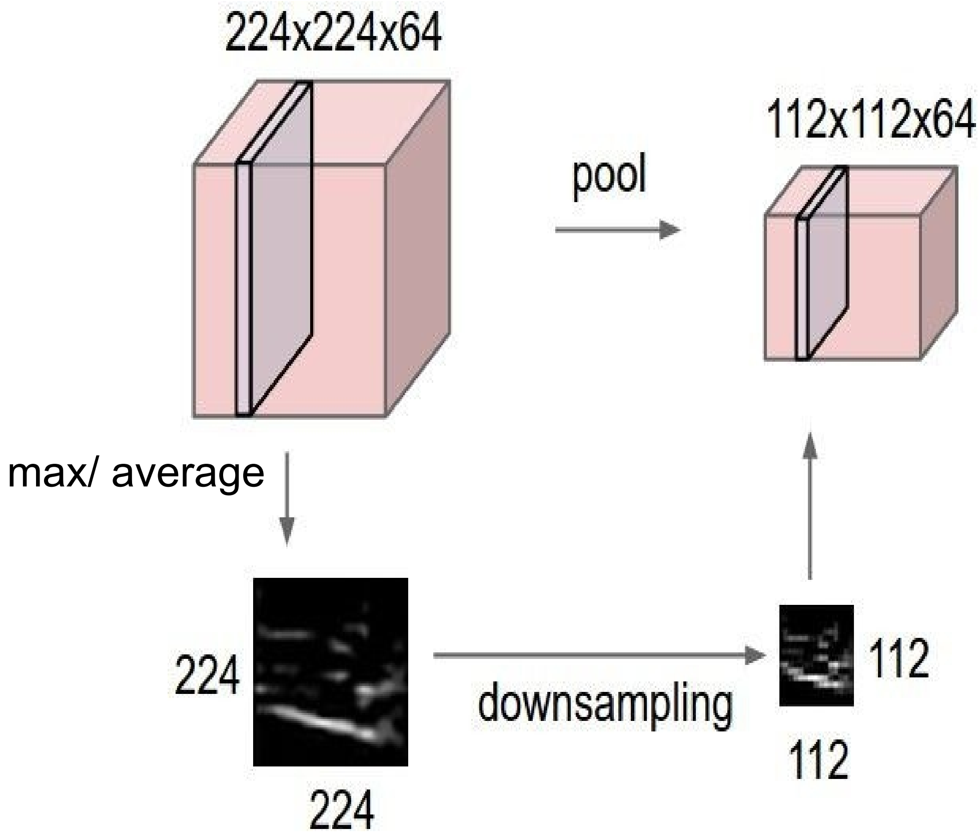
Pooling



Effect = invariance to small translations of the input

Pooling

- Makes representation smaller and computationally less expensive
- Operates over each activation map independently



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Summary

Common settings:

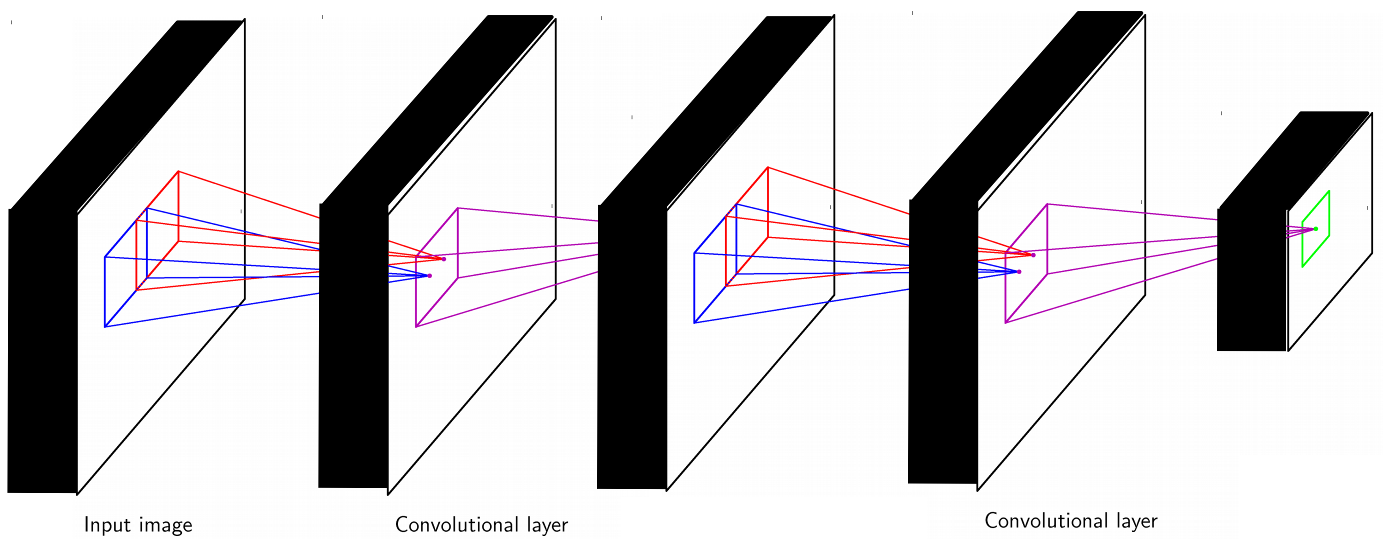
F = 2, S = 2
F = 3, S = 2

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

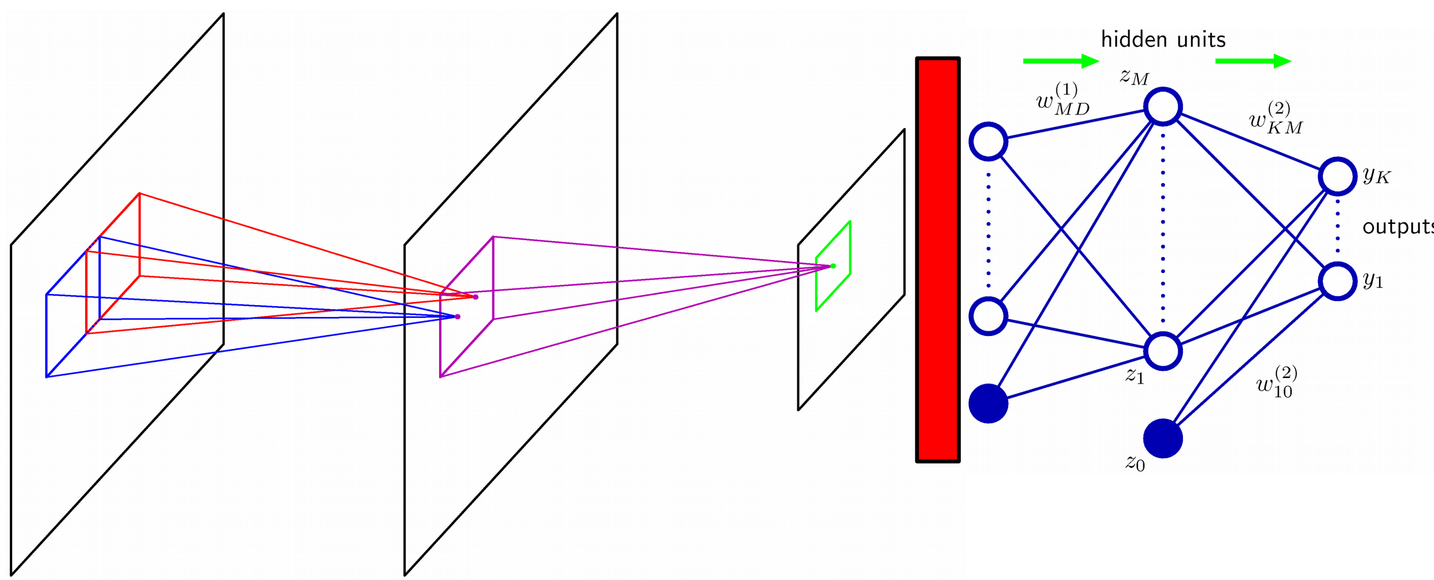
Receptive fields

- "Receptive field" is area in original image impacting a certain unit
  - Later layers can capture more complex patterns over larger areas
- Receptive field size grows linearly over convolutional layers
  - If we use a convolutional filter of size w x w, then each layer the receptive field increases by (w-1)
- Receptive field size increases exponentially over layers with striding
  - Regardless whether they do pooling or convolution



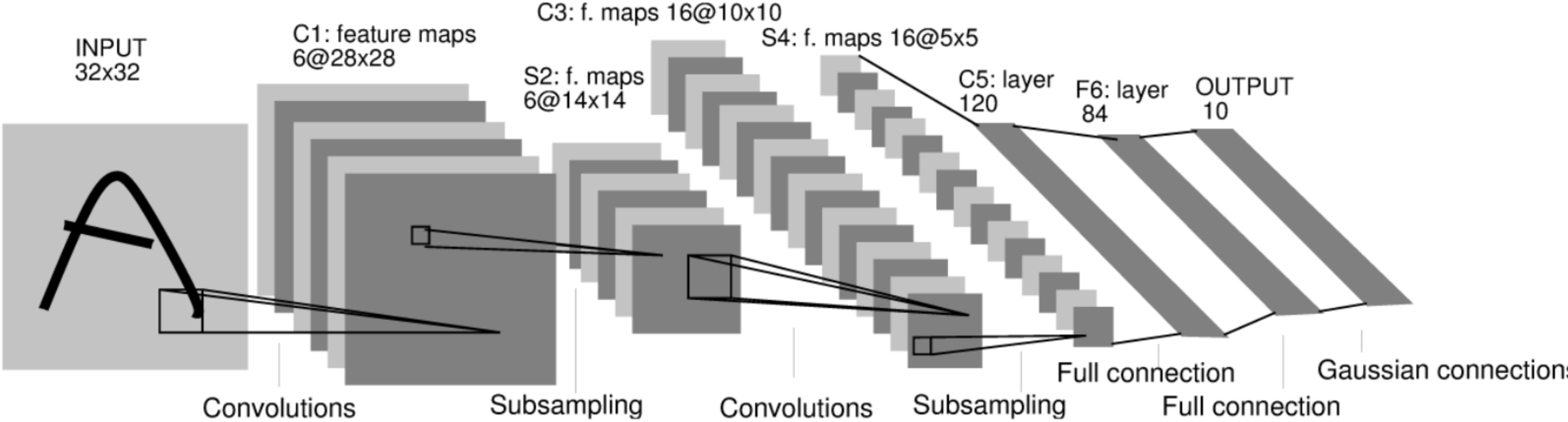
Fully connected layers

- Convolutional and pooling layers typically followed by several “fully connected” (FC) layers, i.e. a standard MLP
  - FC layer connects all units in previous layer to all units in next layer
  - Assembles all local information into global vectorial representation
- FC layers followed by softmax for classification
- First FC layer that connects response map to vector has many parameters
  - Conv layer of size 16x16x256 with following FC layer with 4096 units leads to a connection with 256 million parameters !
  - Large 16x16 filter without padding gives 1x1 sized output map



Convolutional neural network architectures

- LeNet by LeCun et al 1998
- Surprisingly little difference between todays architectures and those of late eighties and nineties
  - Convolutional layers, same
  - Nonlinearities: ReLU dominant now, tanh before
  - Subsampling: more strided convolution now than max/average pooling



Handwritten digit recognition network. LeCun, Bottou, Bengio, Haffner, Proceedings IEEE, 1998

Convolutional neural network architectures

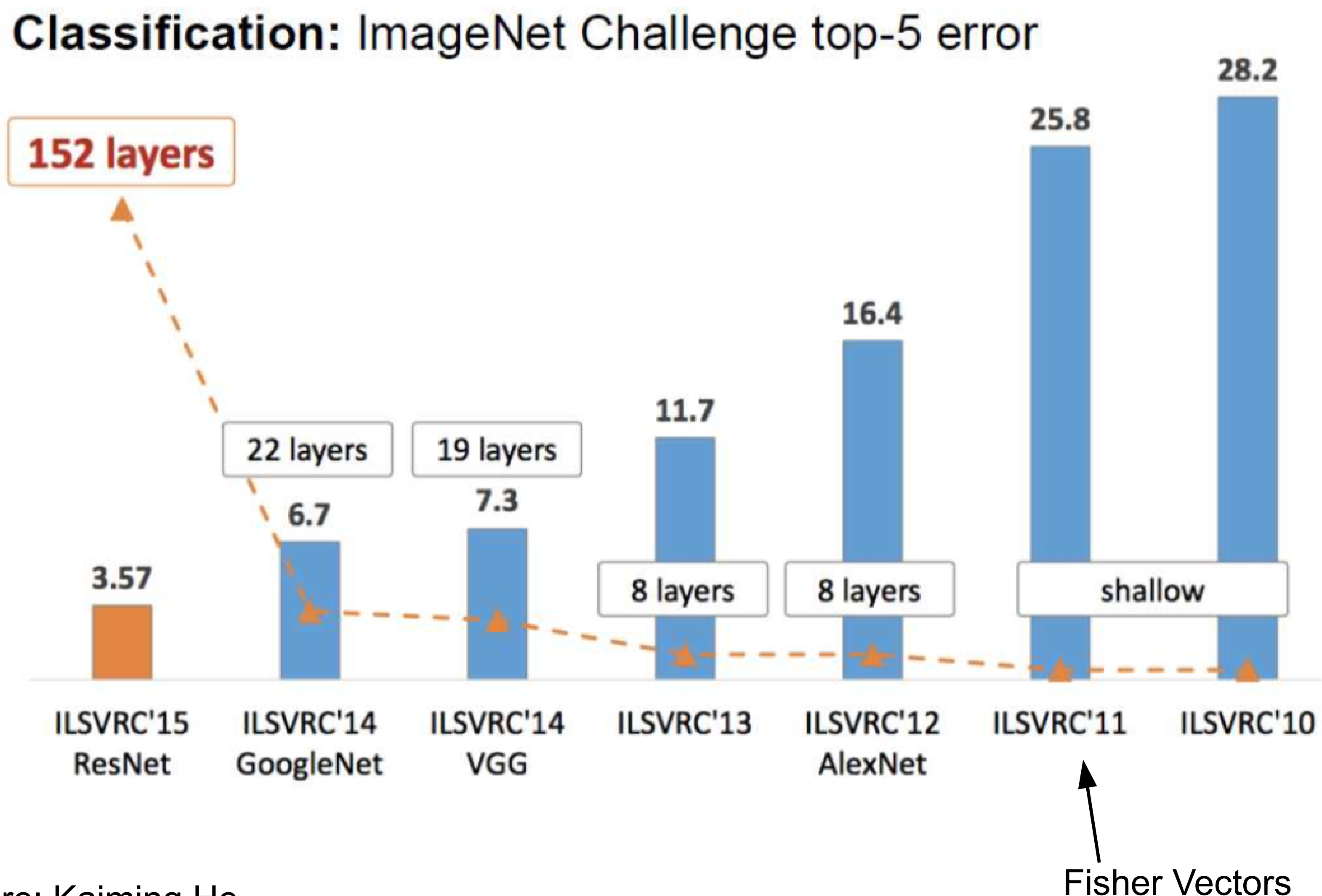
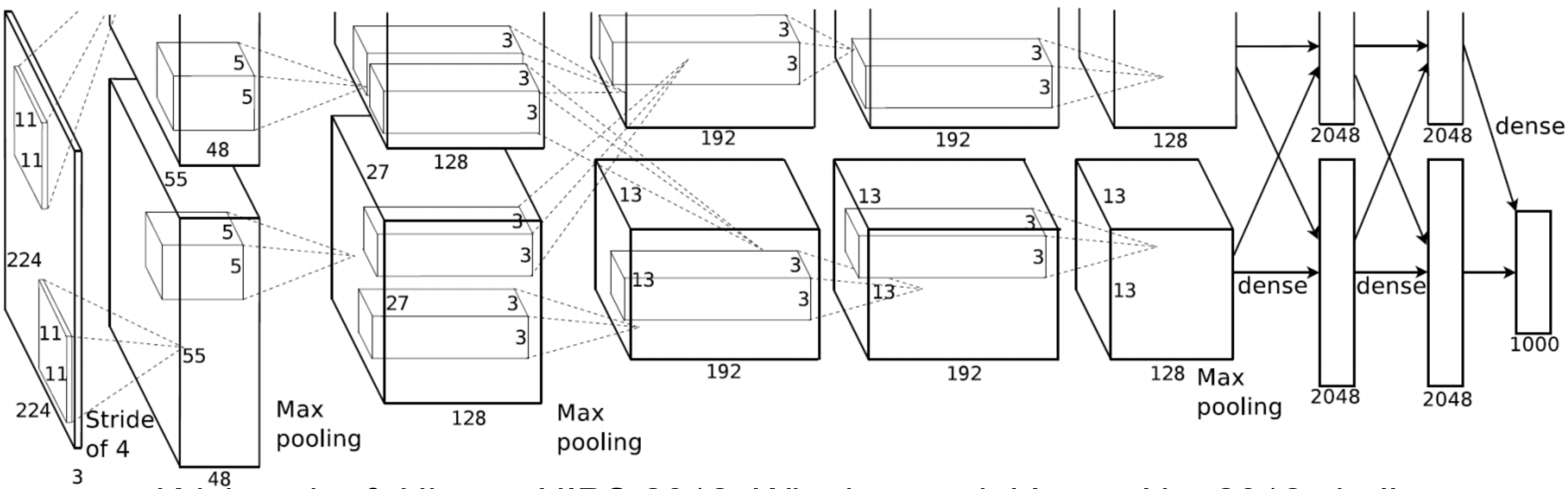


Figure: Kaiming He

Convolutional neural network architectures

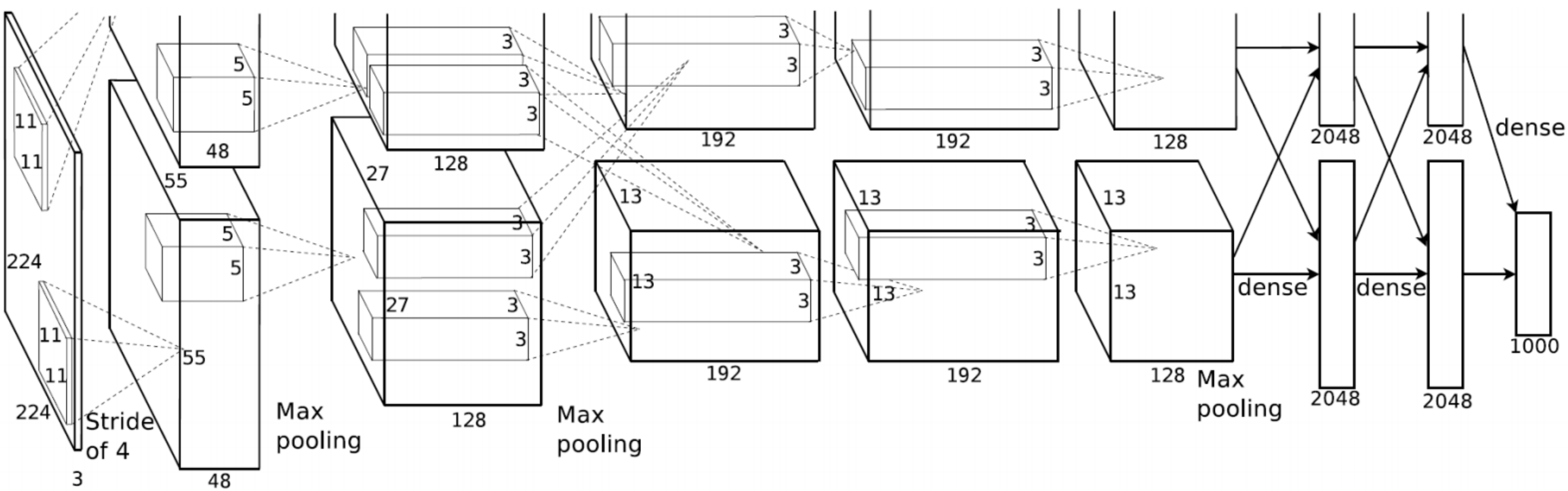
- Recent success with deeper networks
  - 19 layers in Simonyan & Zisserman, ICLR 2015
  - Hundreds of layers in residual networks, He et al. ECCV 2016
- More filters per layer: hundreds to thousands instead of tens
- More parameters: tens or hundreds of millions



Krizhevsky & Hinton, NIPS 2012, Winning model ImageNet 2012 challenge

Other factors that matter

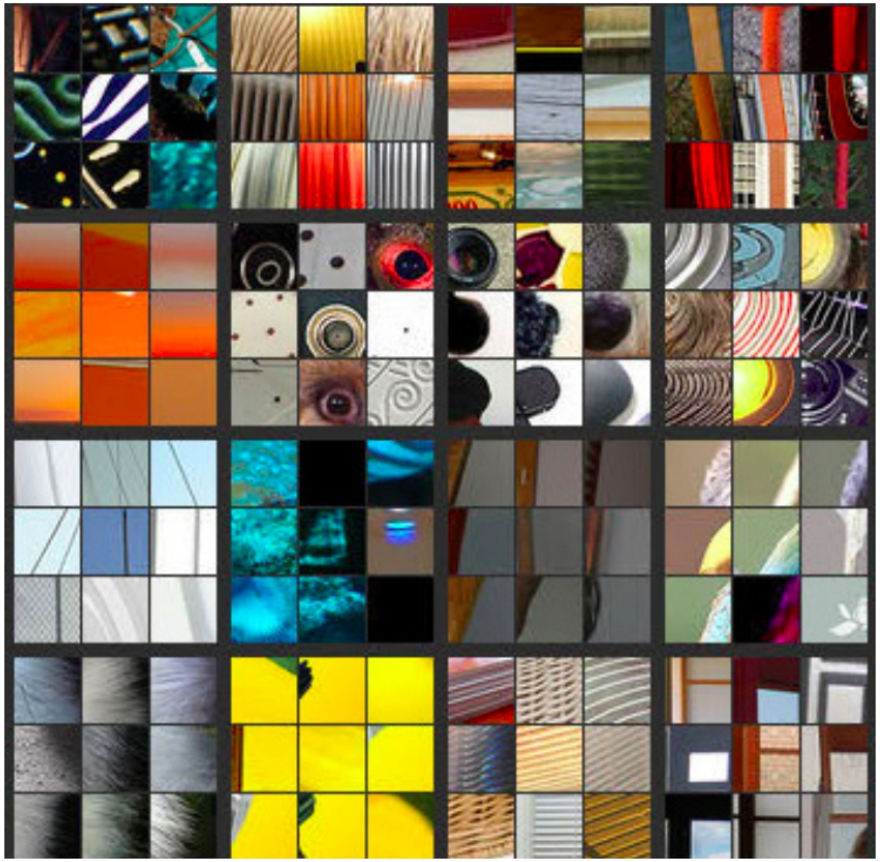
- More training data
  - 1.2 millions of 1000 classes in ImageNet challenge
  - 200 million faces in Schroff et al, CVPR 2015
- GPU-based implementations
  - Massively parallel computation of convolutions
  - Krizhevsky & Hinton, 2012: six days of training on two GPUs
  - Rapid progress in GPU compute performance



Krizhevsky & Hinton, NIPS 2012, Winning model ImageNet 2012 challenge

Understanding convolutional neural network activations

- Patches generating highest response for a selection of convolutional filters,
  - Showing 9 patches per filter
  - Zeiler and Fergus, ECCV 2014
- Layer 1: simple edges and color detectors



- Layer 2: corners, center-surround, ...

Understanding convolutional neural network activations

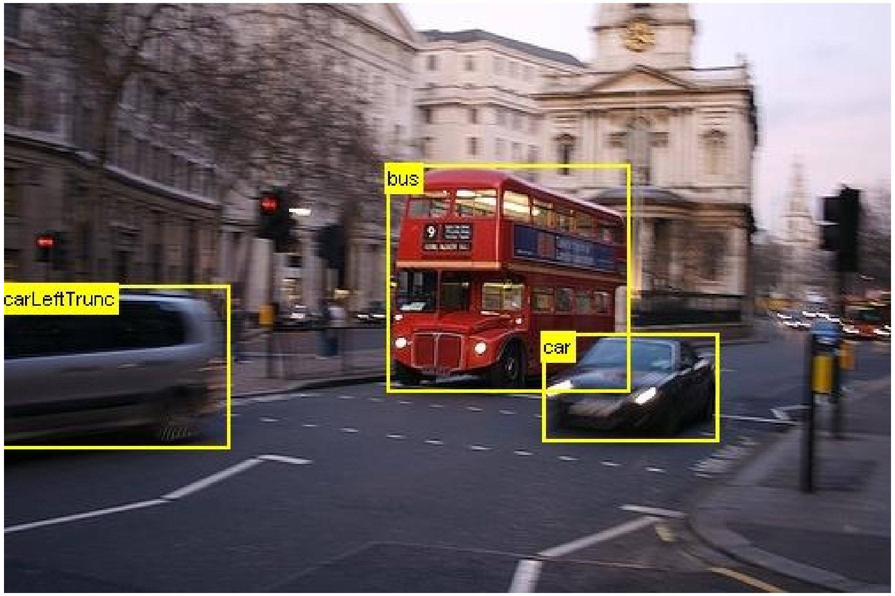
- Layer 3: various object parts

Understanding convolutional neural network activations

- Layer 4+5: selective units for entire objects or large parts of them

Convolutional neural networks for other tasks

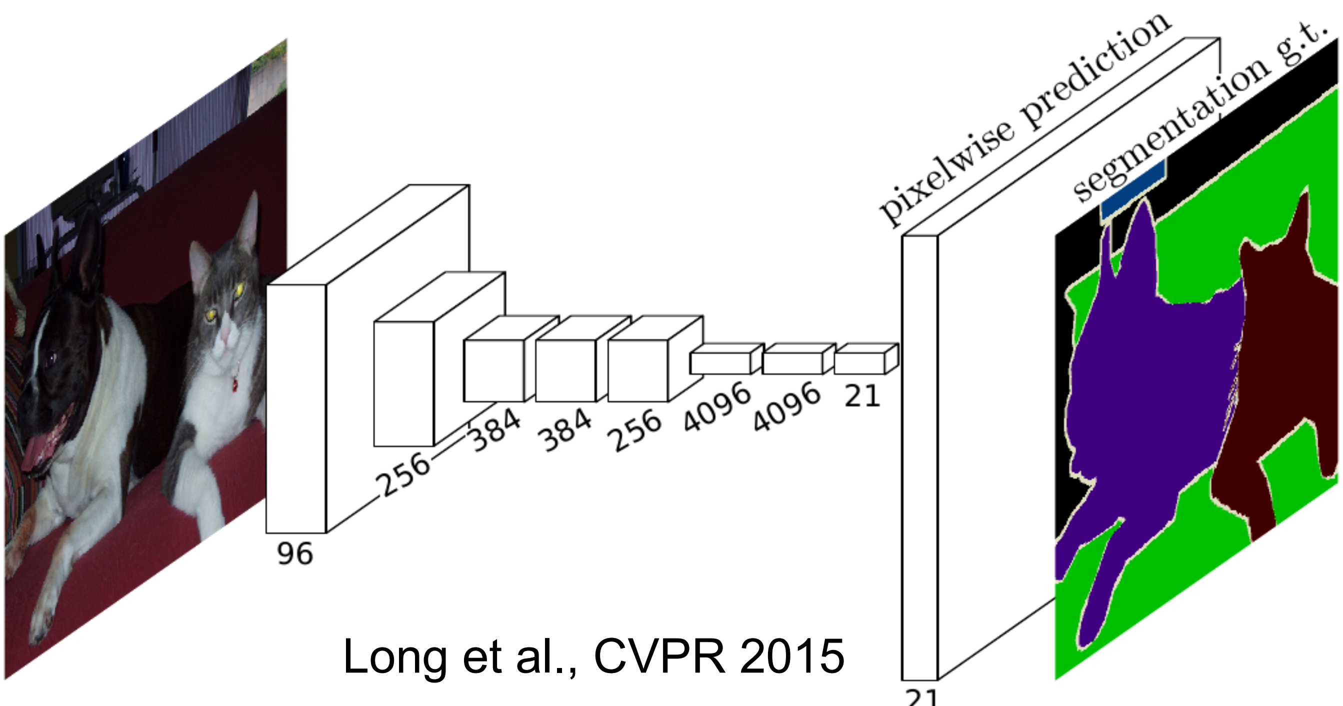
- Object category localization



- **Semantic segmentation**

Application to semantic segmentation

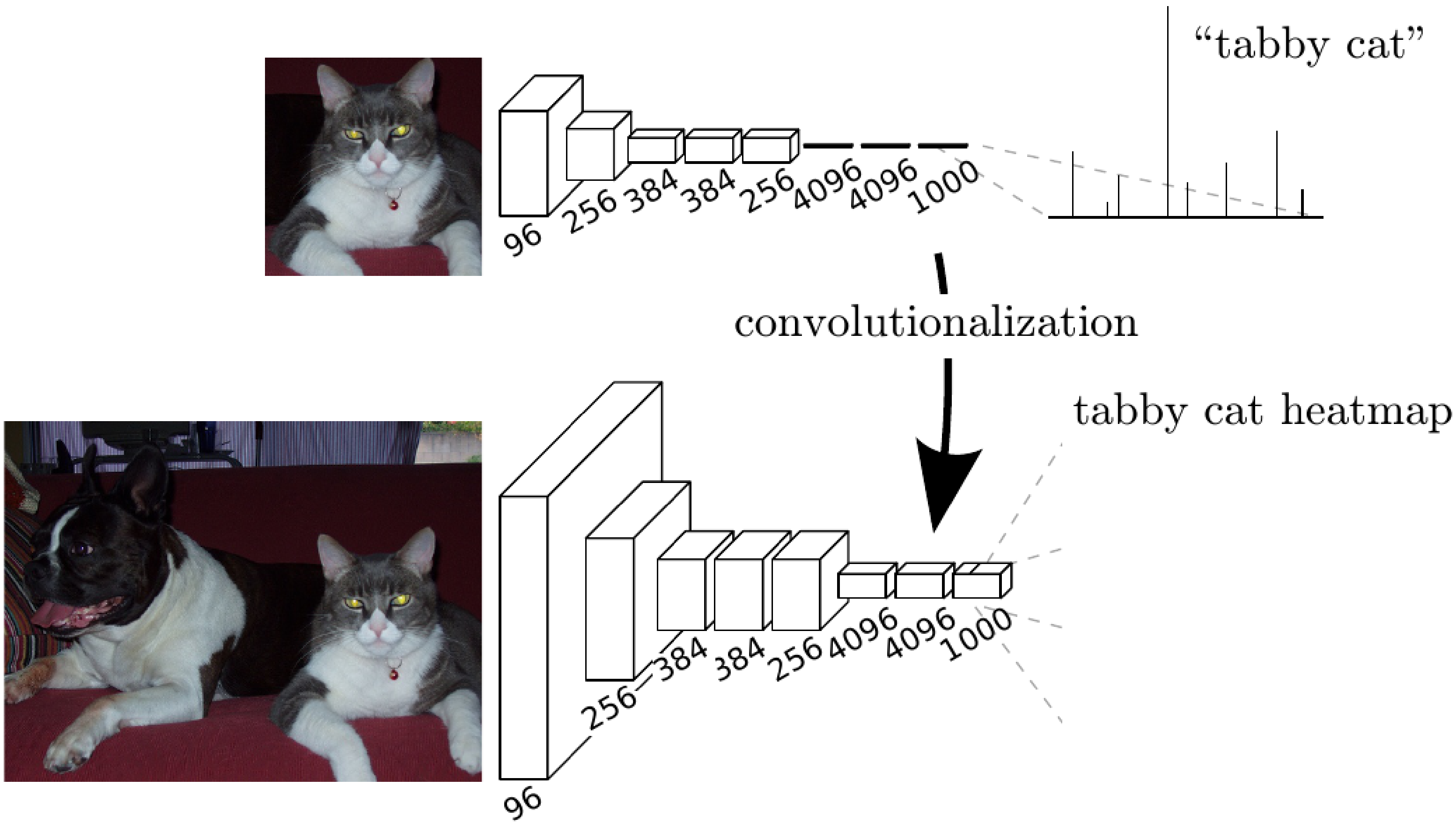
- Assign each pixel to an object or background category
  - Consider running CNN on small image patch to determine its category
  - Train by optimizing per-pixel classification loss
- Similar to SPP-net: want to avoid wasteful computation of convolutional filters
  - Compute convolutional layers once per image
  - Here all local image patches are at the same scale
  - Many more local regions: dense, at every pixel



Long et al., CVPR 2015

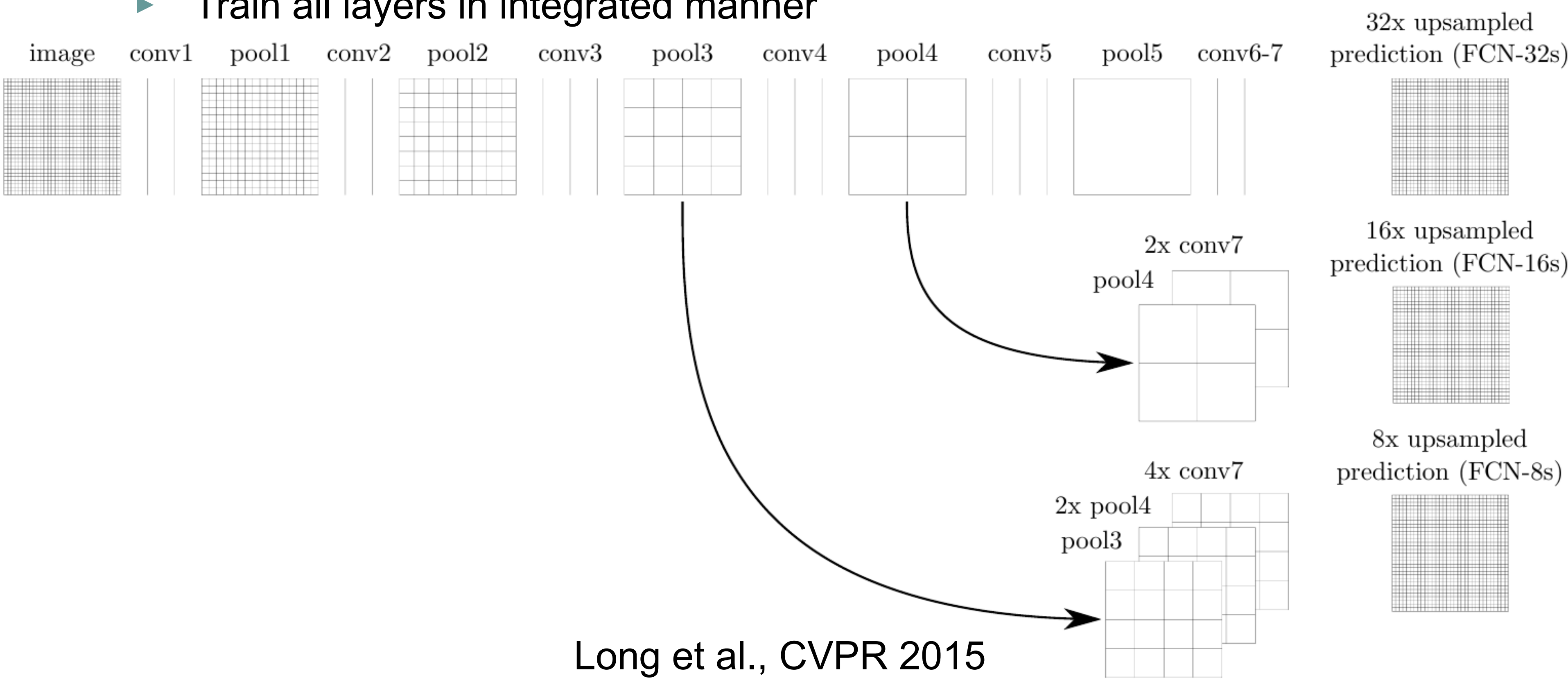
Application to semantic segmentation

- Interpret fully connected layers as 1x1 sized convolutions
  - Function of features in previous layer, but only at own position
  - Still same function is applied across all positions
- Five sub-sampling layers reduce the resolution of output map by factor 32



Application to semantic segmentation

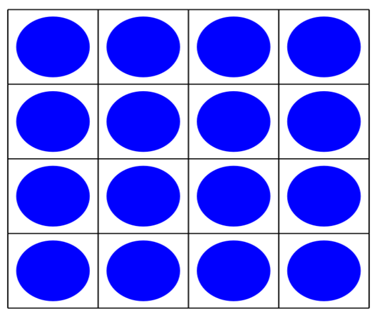
- Idea 1: up-sampling via bi-linear interpolation
  - Gives blurry predictions
- Idea 2: weighted sum of response maps at different resolutions
  - Upsampling of the later and coarser layer
  - Concatenate fine layers and upsampled coarser ones for prediction
  - Train all layers in integrated manner

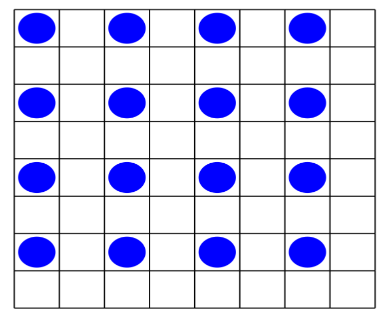


Long et al., CVPR 2015

Upsampling of coarse activation maps

- Simplest form: use bilinear interpolation or nearest neighbor interpolation
  - Note that these can be seen as upsampling by zero-padding, followed by convolution with specific filters, no channel interactions
- Idea can be generalized by learning the convolutional filter
  - No need to hand-pick the interpolation scheme
  - Can include channel interactions, if those turn out be useful





$$\text{Bi-linear:} \quad \frac{1}{4}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \qquad \text{Nearest neighbor:} \quad \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- Resolution-increasing counterpart of strided convolution
  - Average and max pooling can be written in terms of convolutions
  - See: “Convolutional Neural Fabrics”, Saxena & Verbeek, NIPS 2016.

Application to semantic segmentation

- Results obtained at different resolutions
  - Detail better preserved at finer resolutions