10707
Deep Learning: Spring 2020

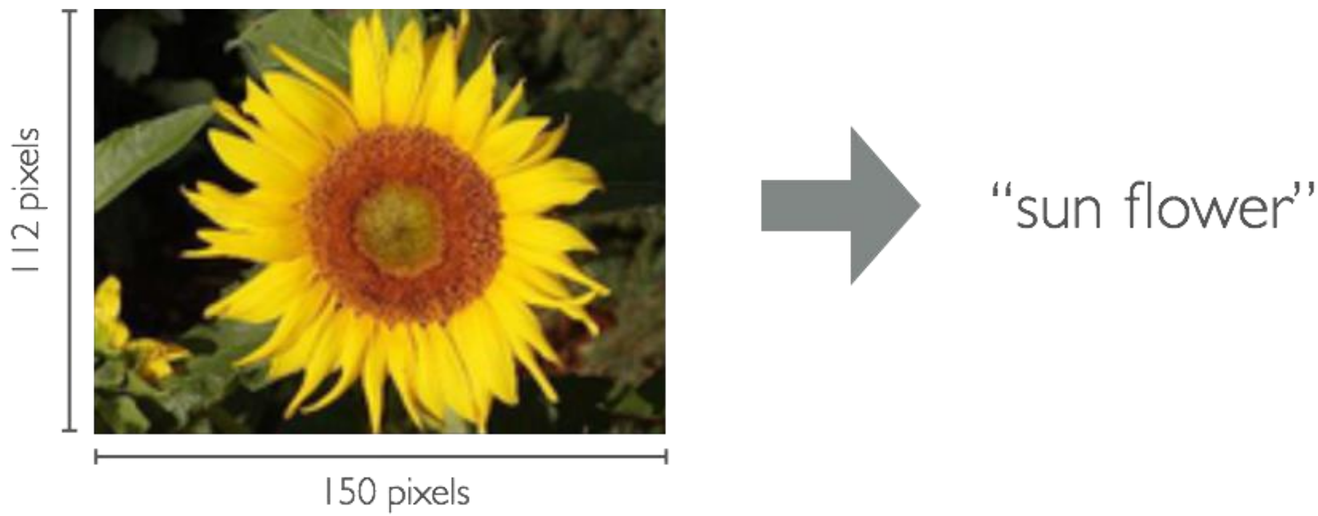
Andrej Risteski

Machine Learning Department

Lecture 4:
Convolutional architectures

Neural networks for vision

Prototypical task in vision is object recognition: given an input image, identify what kind of object it contains.



Are feedforward networks the right architecture for this?

Desiderata for networks for vision

- Inputs are very high-dimensional: 150 x 150 pixels = 22500 inputs, or 3 x 22500 if RGB pixels instead of grayscale.
- Should leverage the spatial locality (in the pixel sense) of data
- Build in invariance to natural variations: translation, illumination, etc.

Convolutional architectures are designed for this:

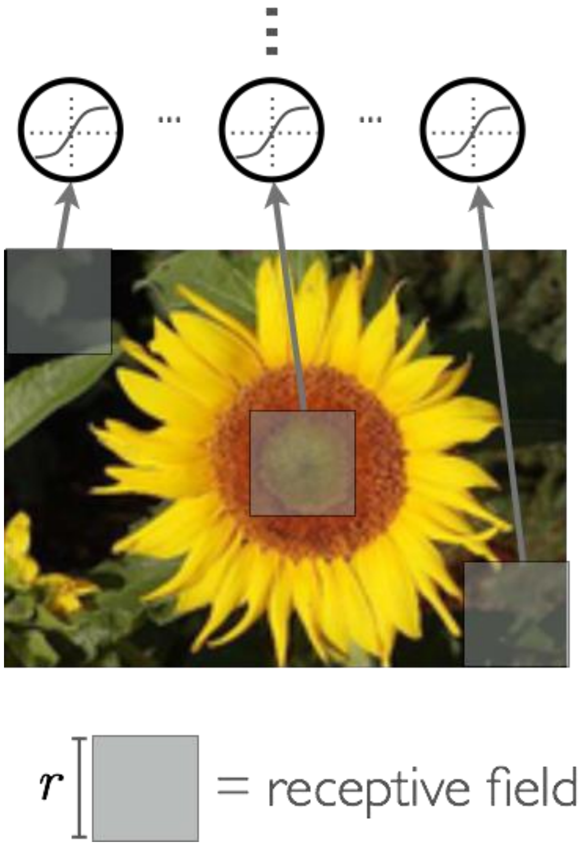
- Local connectivity (reflects spatial locality and decreases # params)
- Parameter sharing (further decreases # params)
- Convolution
- Pooling / subsampling hidden units

Local Connectivity

Use local connectivity of hidden units

- Each hidden unit is connected only to a sub-region (patch) of the input image.

  It is connected to all channels: 1 if grayscale, 3 (R, G, B) if color image

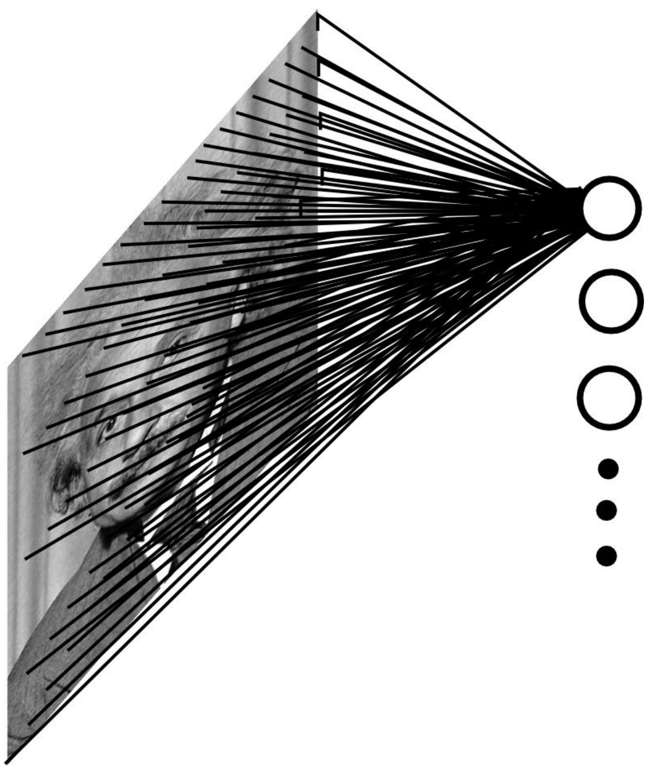


Why this is a good idea:

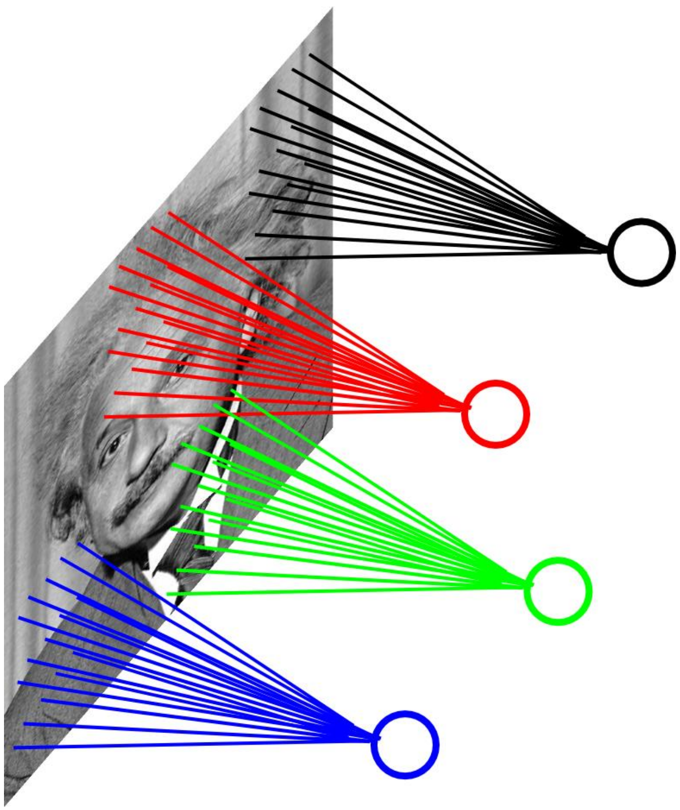
- Fully connected layer has a lot of parameters to fit, which requires a lot of training data
- Image data isn't arbitrary: neighboring pixels are "meaningfully related" – e.g. if a node is to be a "dog nose" detector – need to look at small patch of pixels.

Decrease in # of parameters

Fully connected: 200x200 image, 40K hidden units, ~2B parameters!



Convolutional: 200x200 image, 40K hidden units, window size 10x10, ~4M parameters!



Parameter Sharing

Prior approach makes weights sensitive to translations: e.g.

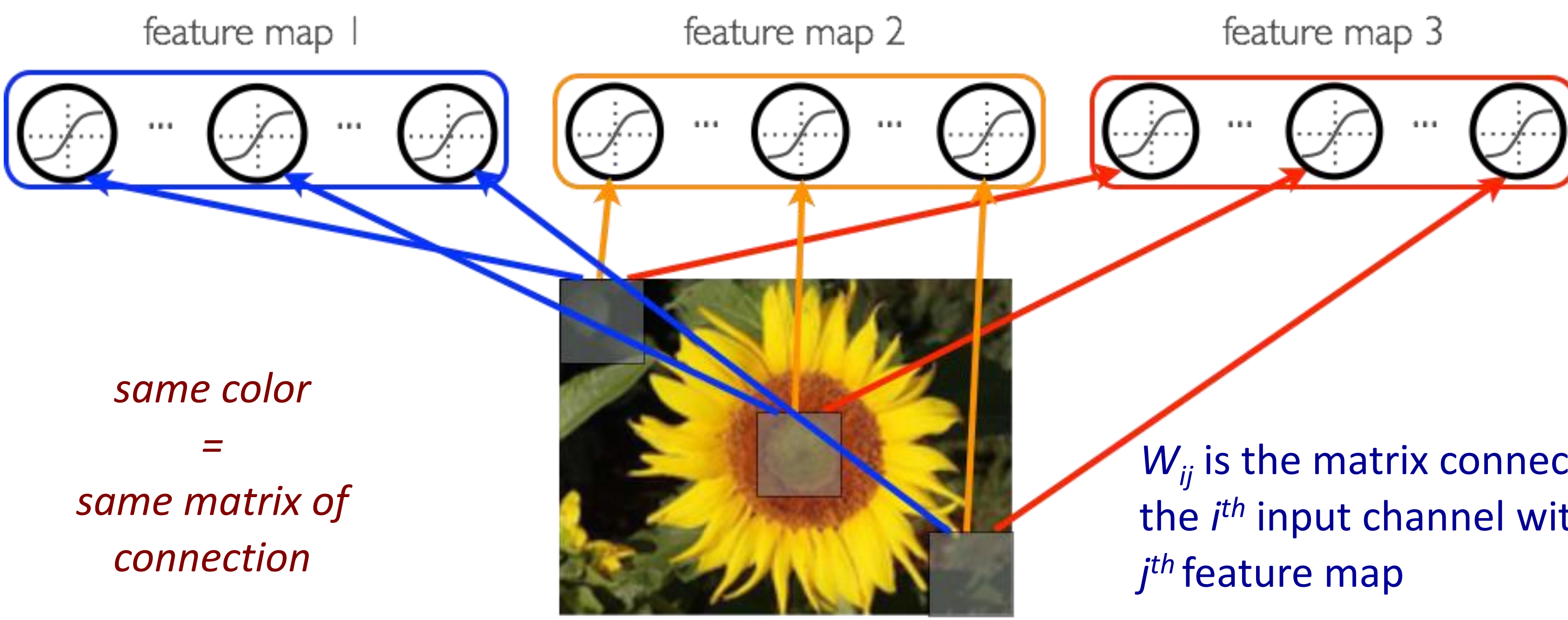
Might learn weights for nose detector here

But not learn weights for a nose detector here

Parameter sharing

Share matrix of parameters across some units

- Units that are organized into the "feature map" share parameters
- Hidden units within a feature map cover different positions in the image



same color
=
same matrix of connection

W_{ij} is the matrix connecting the i^{th} input channel with the j^{th} feature map

Computer Vision

Our goal is to design neural networks that are specifically adapted for such problems

- Must deal with very high-dimensional inputs: 150 x 150 pixels = 22500 inputs, or 3 x 22500 if RGB pixels
- Can exploit the 2D topology of pixels (or 3D for video data)
- Can build in invariance to certain variations: translation, illumination, etc.

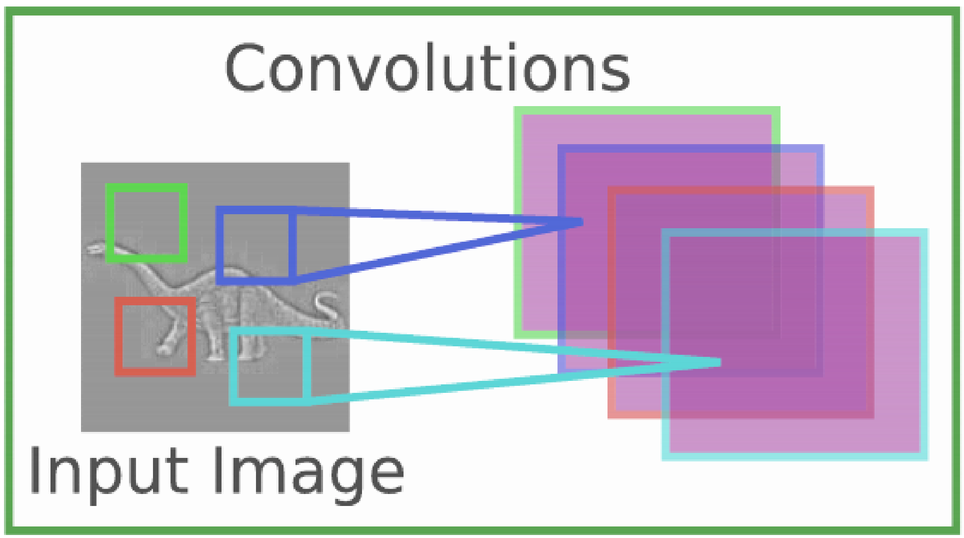
Convolutional networks leverage these ideas

- Local connectivity
- Parameter sharing
- Convolution
- Pooling / subsampling hidden units

Parameter Sharing

Each feature map forms a **2D grid of features**

Can be computed with a discrete convolution ($\ast$) of a kernel matrix k_{ij}



Jarret et al. 2009

$$y_j = g_j \tanh\left(\sum_i k_{ij} * x_i\right)$$

- x_i is the ith channel of input
- k_{ij} is the convolution kernel
- g_j is a learned scaling factor
- y_j is the hidden layer

Can add bias

Discrete Convolution

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{r-p,r-q}$$

Example:

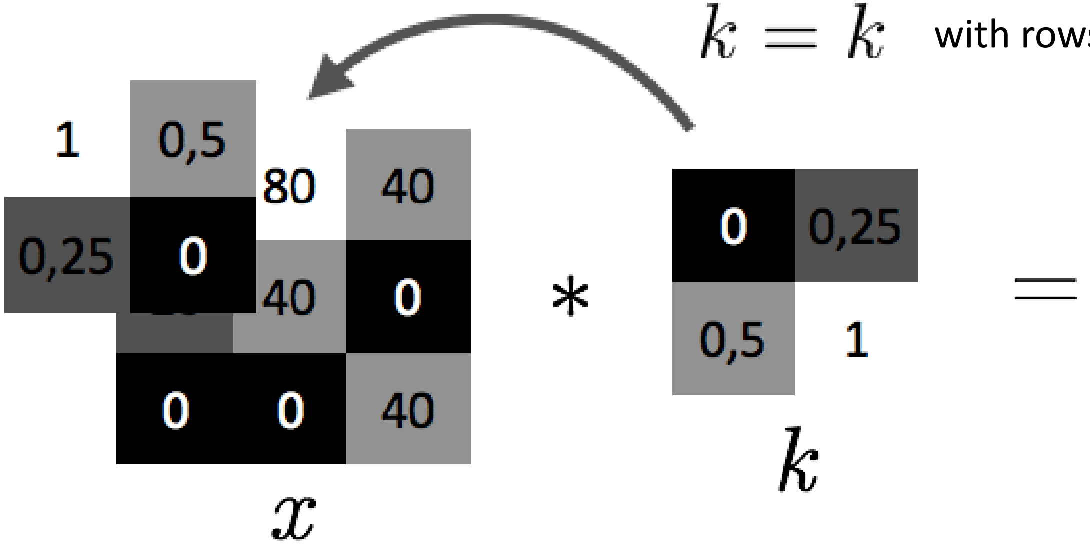
$$\underbrace{\begin{bmatrix} 0 & 80 & 40 \\ 20 & 40 & 0 \\ 0 & 0 & 40 \end{bmatrix}}_{x} * \underbrace{\begin{bmatrix} 0 & 0{,}25 \\ 0{,}5 & 1 \end{bmatrix}}_{k} =$$

Discrete Convolution

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{r-p,r-q}$$

Example:

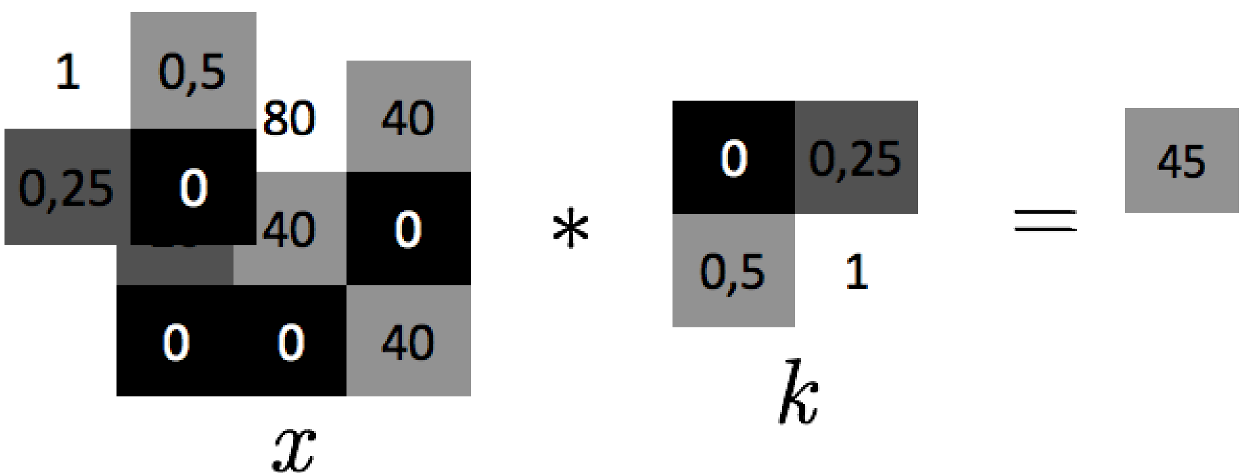
$\tilde{k} = k$ with rows and columns flipped



Discrete Convolution

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{r-p,r-q}$$

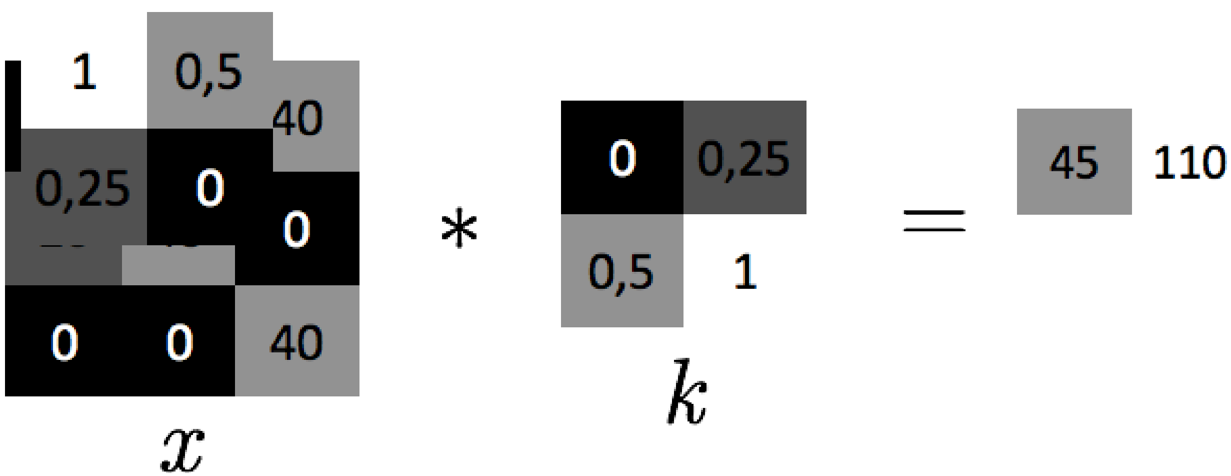
Example: $1 \times 0 + 0.5 \times 80 + 0.25 \times 20 + 0 \times 40 = 45$



Discrete Convolution

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{r-p,r-q}$$

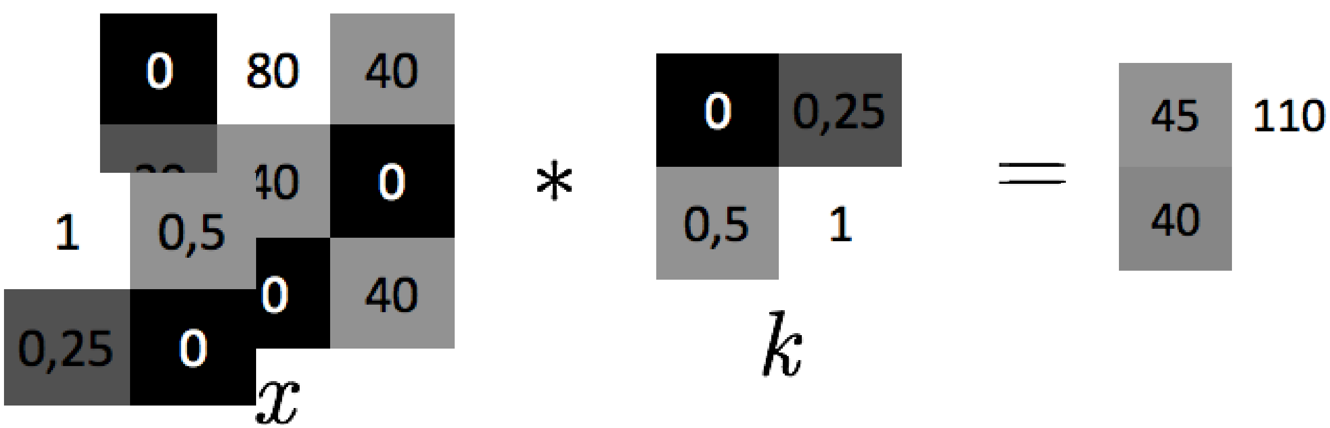
Example: $1 \times 80 + 0.5 \times 40 + 0.25 \times 40 + 0 \times 0 = 110$



Discrete Convolution

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{r-p,r-q}$$

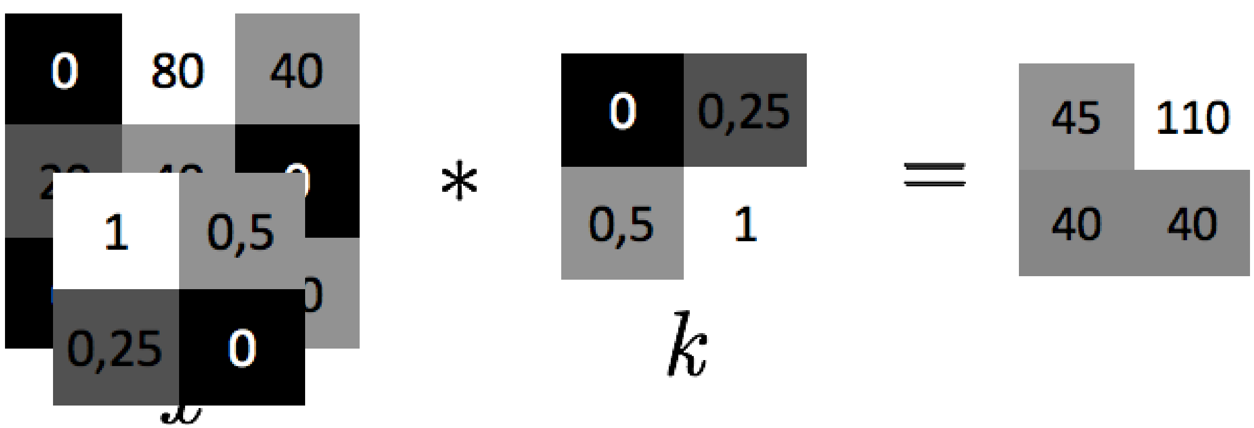
Example: $1 \times 20 + 0.5 \times 40 + 0.25 \times 0 + 0 \times 0 = 40$



Discrete Convolution

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{r-p,r-q}$$

Example: $1 \times 40 + 0.5 \times 0 + 0.25 \times 0 + 0 \times 40 = 40$



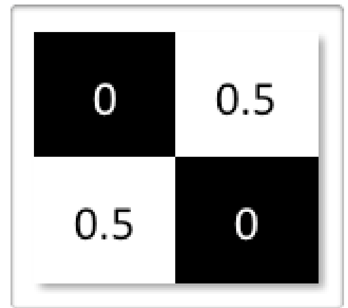
Example of a convolution

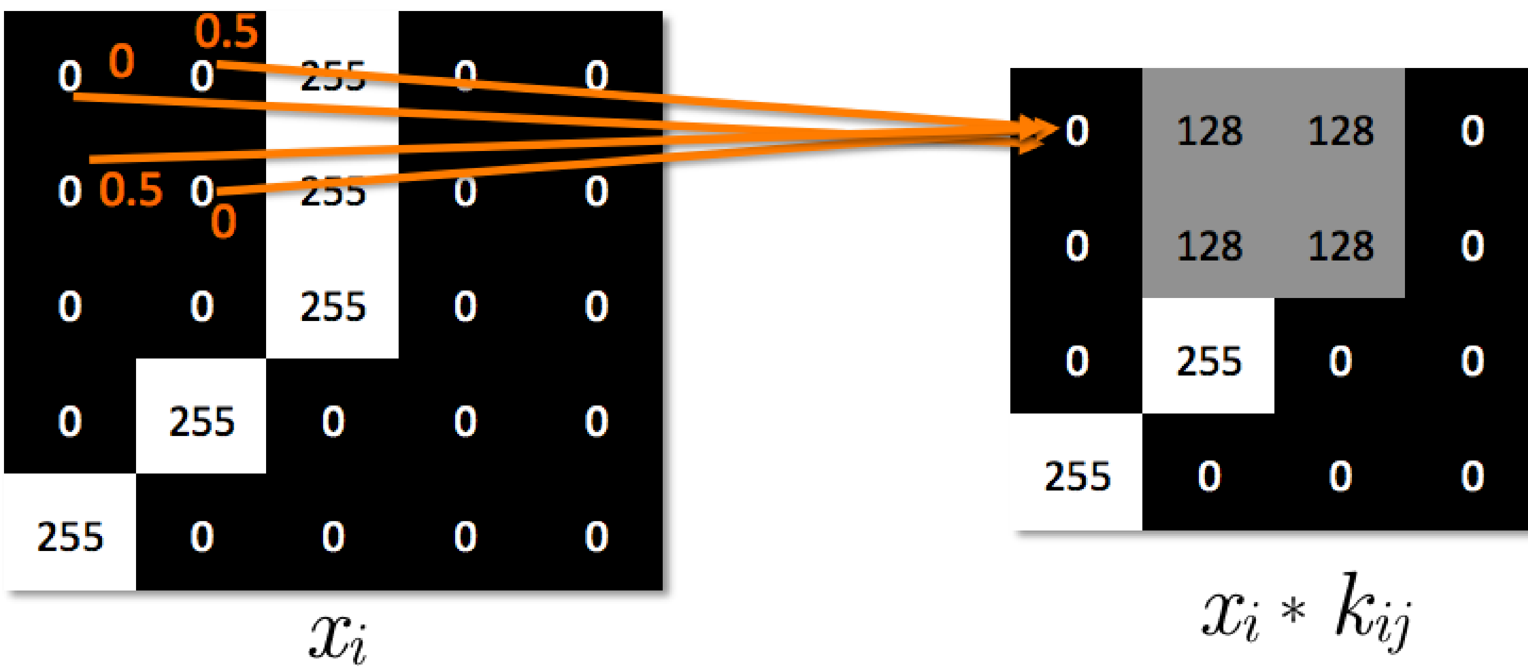
$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$

Adding non-linearity

With a non-linearity, we get a detector of a feature at any position in the image:

$$x * k_{ij},$$





Adding non-linearity

With a non-linearity, we get a detector of a feature at any position in the image:

$$x * k_{ij},$$

0	0.5
0.5	0

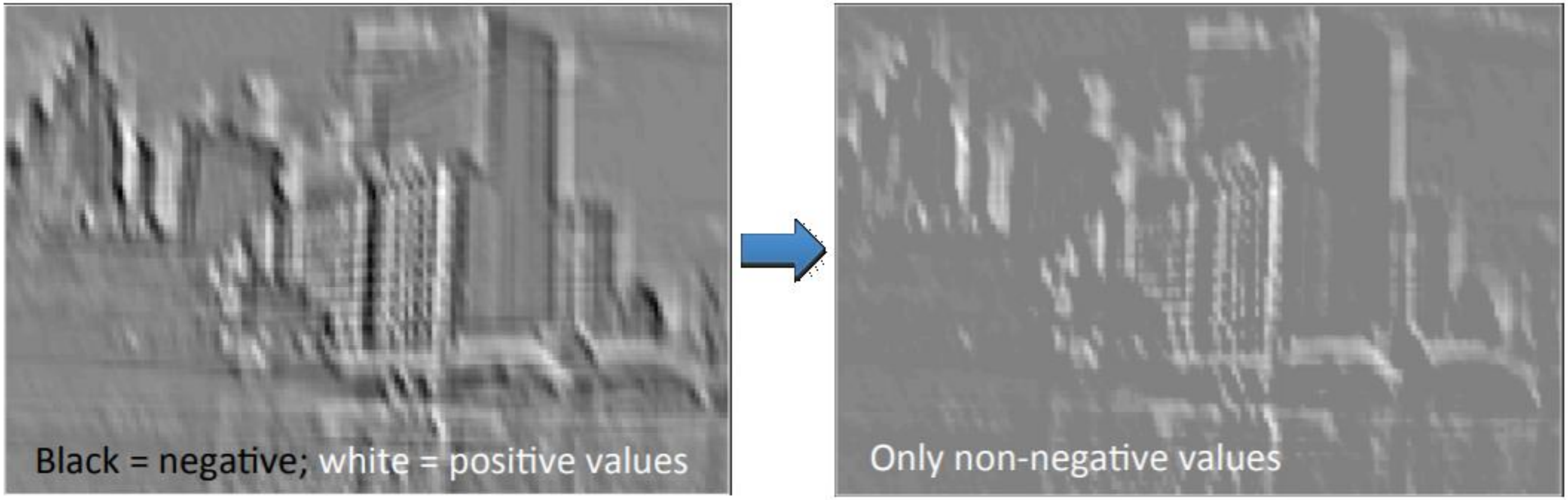
0	0	255	0	0
0	0	255	0	0
0	0	255	0	0
0	255	0	0	0
255	0	0	0	0

x_i

0.02	0.19	0.19	0.02
0.02	0.19	0.19	0.02
0.02	0.75	0.02	0.02
0.75	0.02	0.02	0.02

$$\mathrm{sigm}(0.02\ x_i * k_{ij}\ \text{-}4)$$

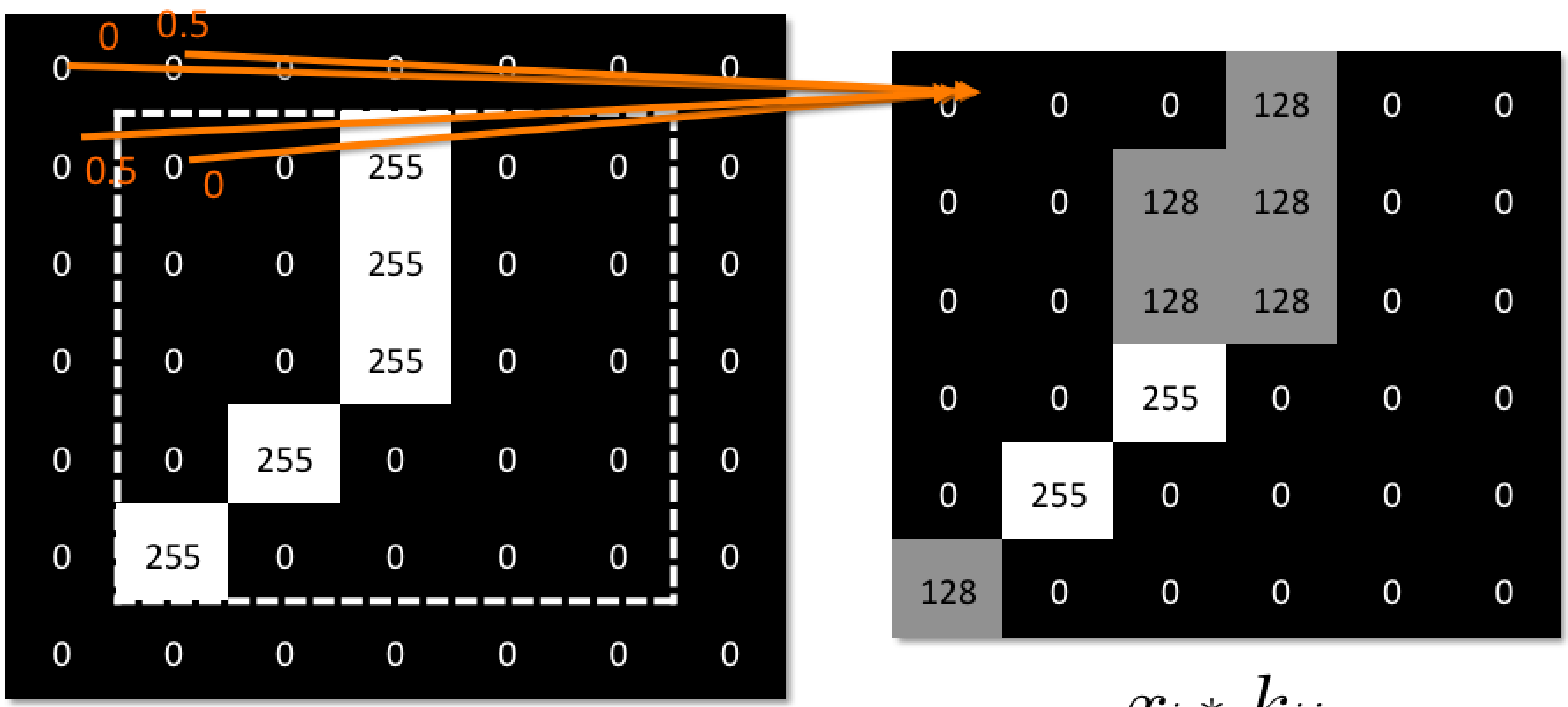
Example of ReLU non-linearity



From Rob Fergus tutorial (http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf)

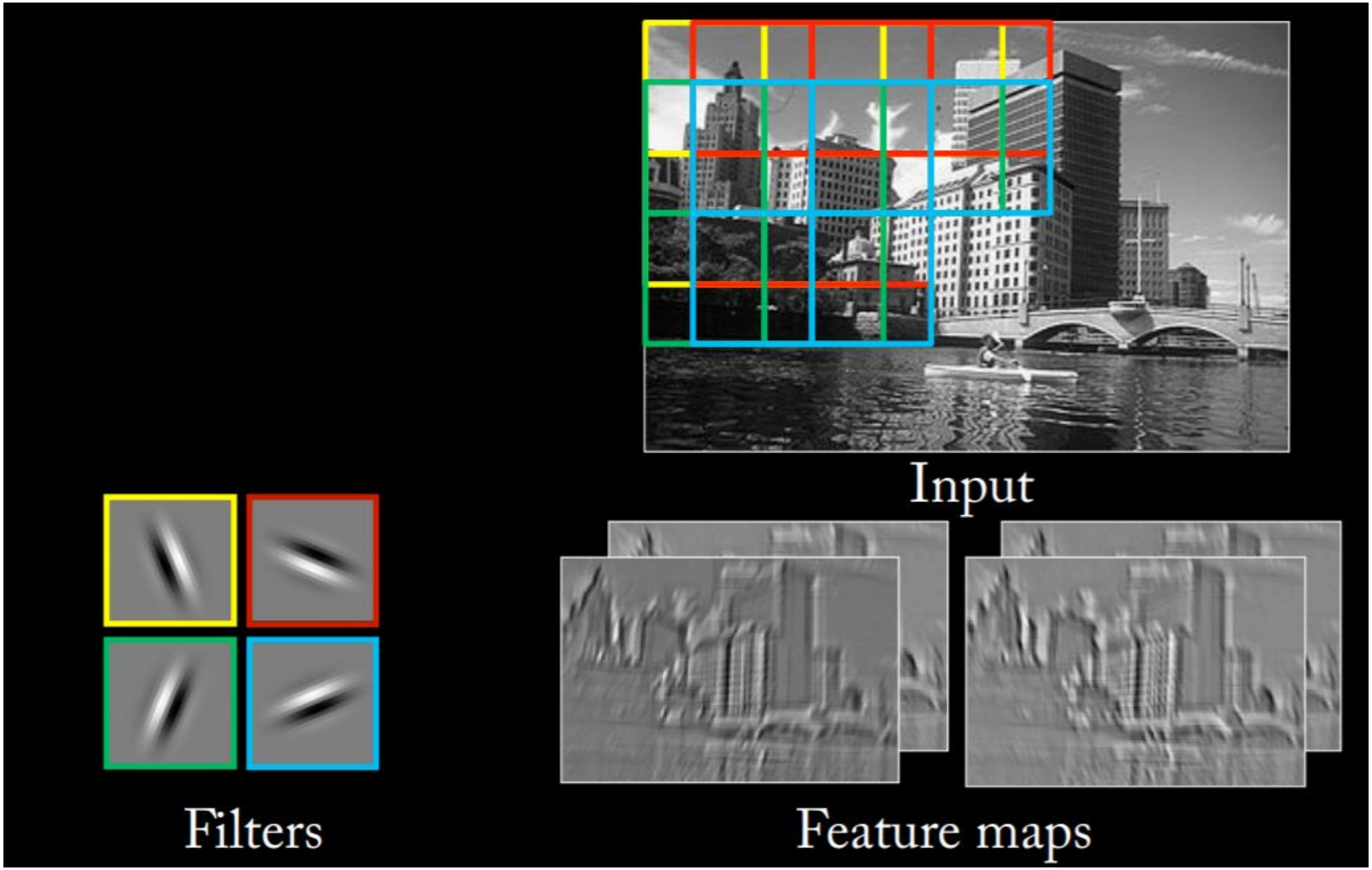
Padding

- Can use “zero padding” to allow going over the borders (*)



x_i $\qquad$ $x_i \underline{*} k_{ij}$

The picture so far



From Rob Fergus tutorial (http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf)

Computer Vision

Our goal is to design neural networks that are specifically adapted for such problems

- Must deal with very high-dimensional inputs: 150 x 150 pixels = 22500 inputs, or 3 x 22500 if RGB pixels
- Can exploit the 2D topology of pixels (or 3D for video data)
- Can build in invariance to certain variations: translation, illumination, etc.

Convolutional networks leverage these ideas

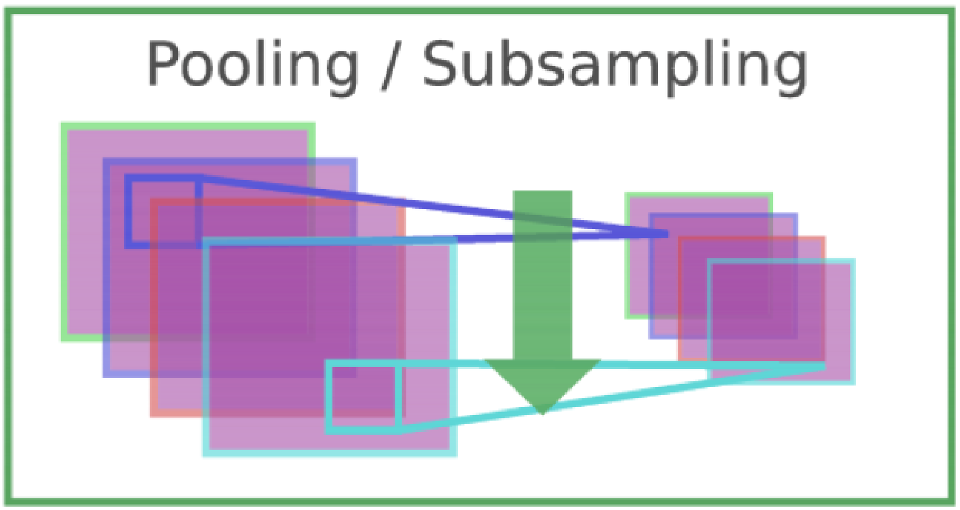
- Local connectivity
- Parameter sharing
- Convolution
- Pooling / subsampling hidden units

Pooling

Pool hidden units in same neighborhood

Pooling is performed in non-overlapping neighborhoods (subsampling)

$$y_{ijk} = \max_{p,q} x_{i,j+p,k+q}$$



Jarret et al. 2009

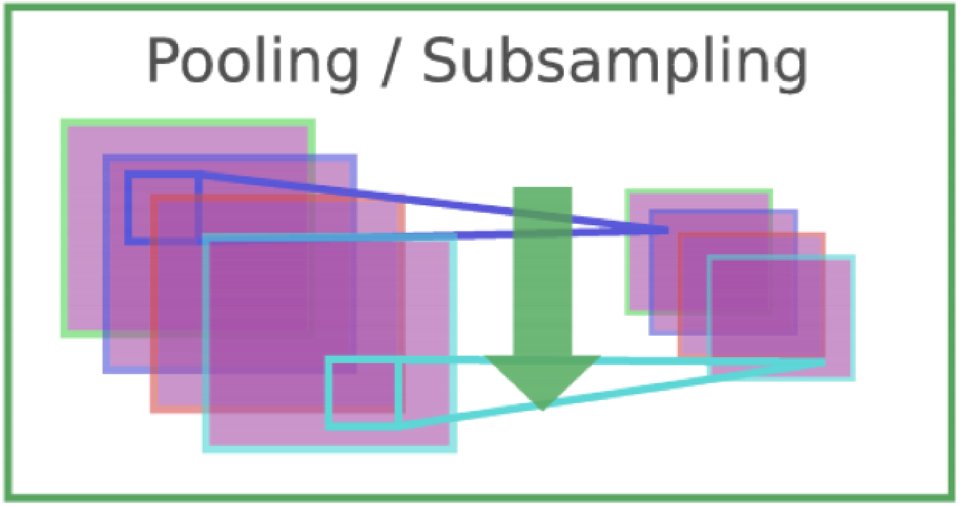
- x_i is the i^{th} channel of input
- $x_{i,j,k}$ is value of the i^{th} feature map at position j,k
- p is vertical index in local neighborhood
- q is horizontal index in local neighborhood
- y_{ijk} is pooled / subsampled layer

Pooling

Pool hidden units in same neighborhood

An alternative to “max” pooling is “average” pooling

$$y_{ijk} = \frac{1}{m^2} \sum_{p,q} x_{i,j+p,k+q}$$

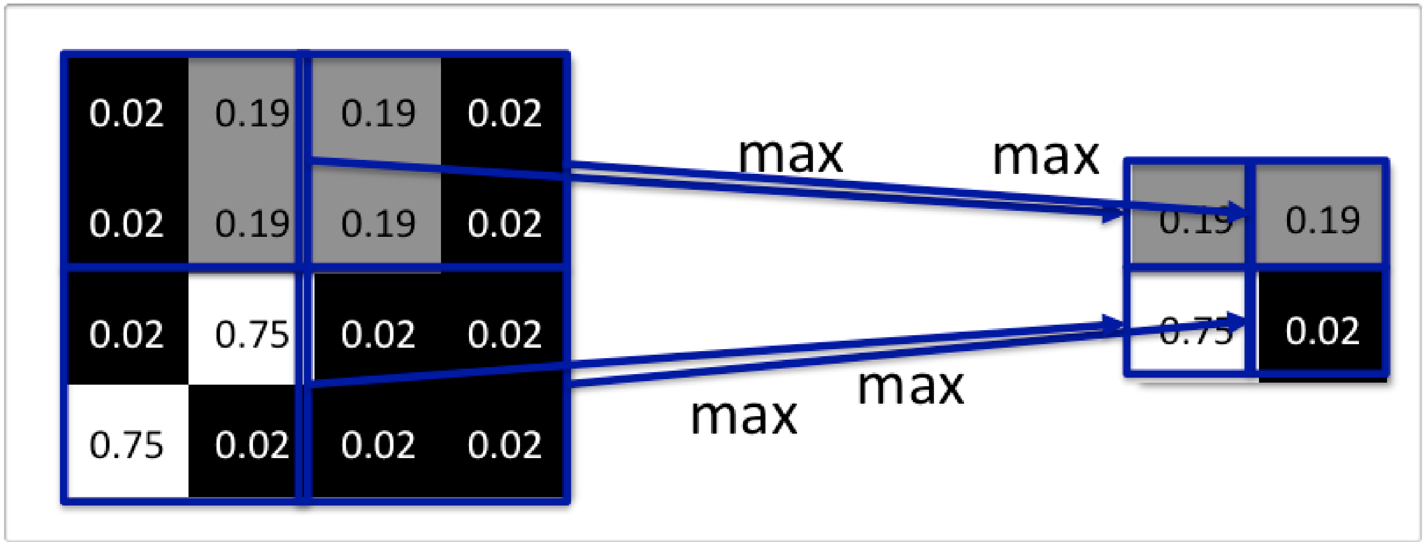


Jarret et al. 2009

- x_i is the i^{th} channel of input
- $x_{i,j,k}$ is value of the i^{th} feature map at position j,k
- p is vertical index in local neighborhood
- q is horizontal index in local neighborhood
- y_{ijk} is pooled / subsampled layer
- m is the neighborhood height/width

Example: Pooling

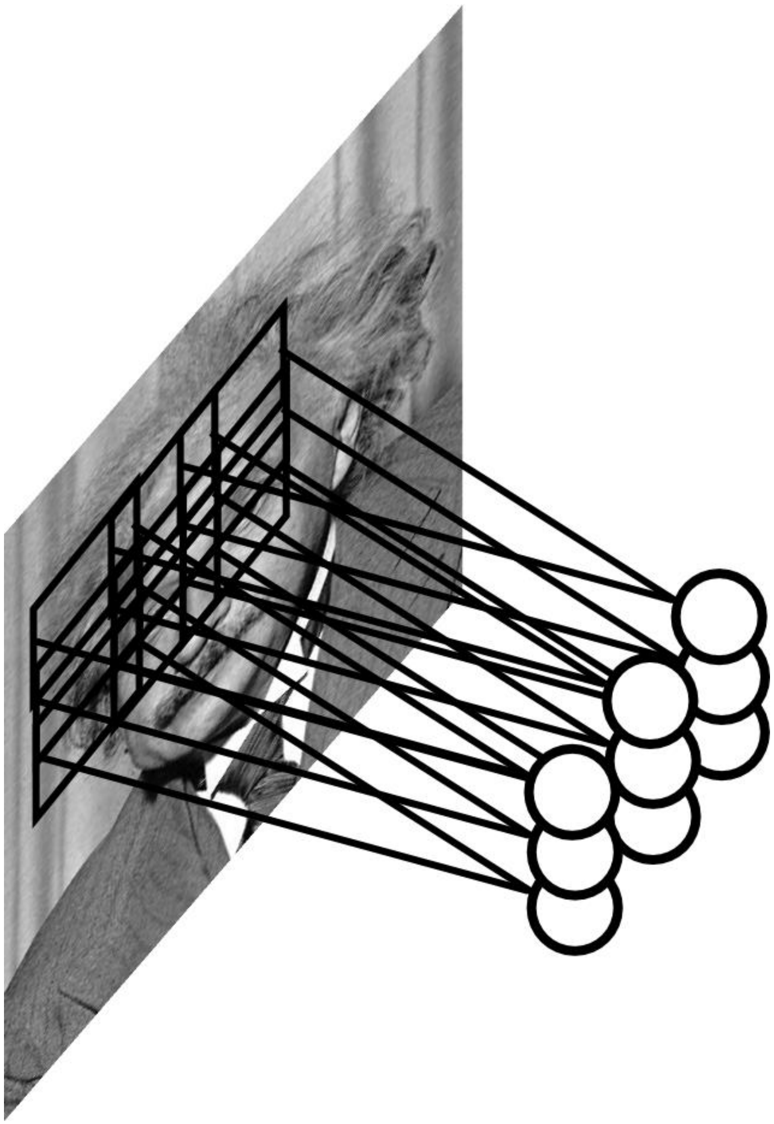
Illustration of pooling/subsampling operation



Why pooling?

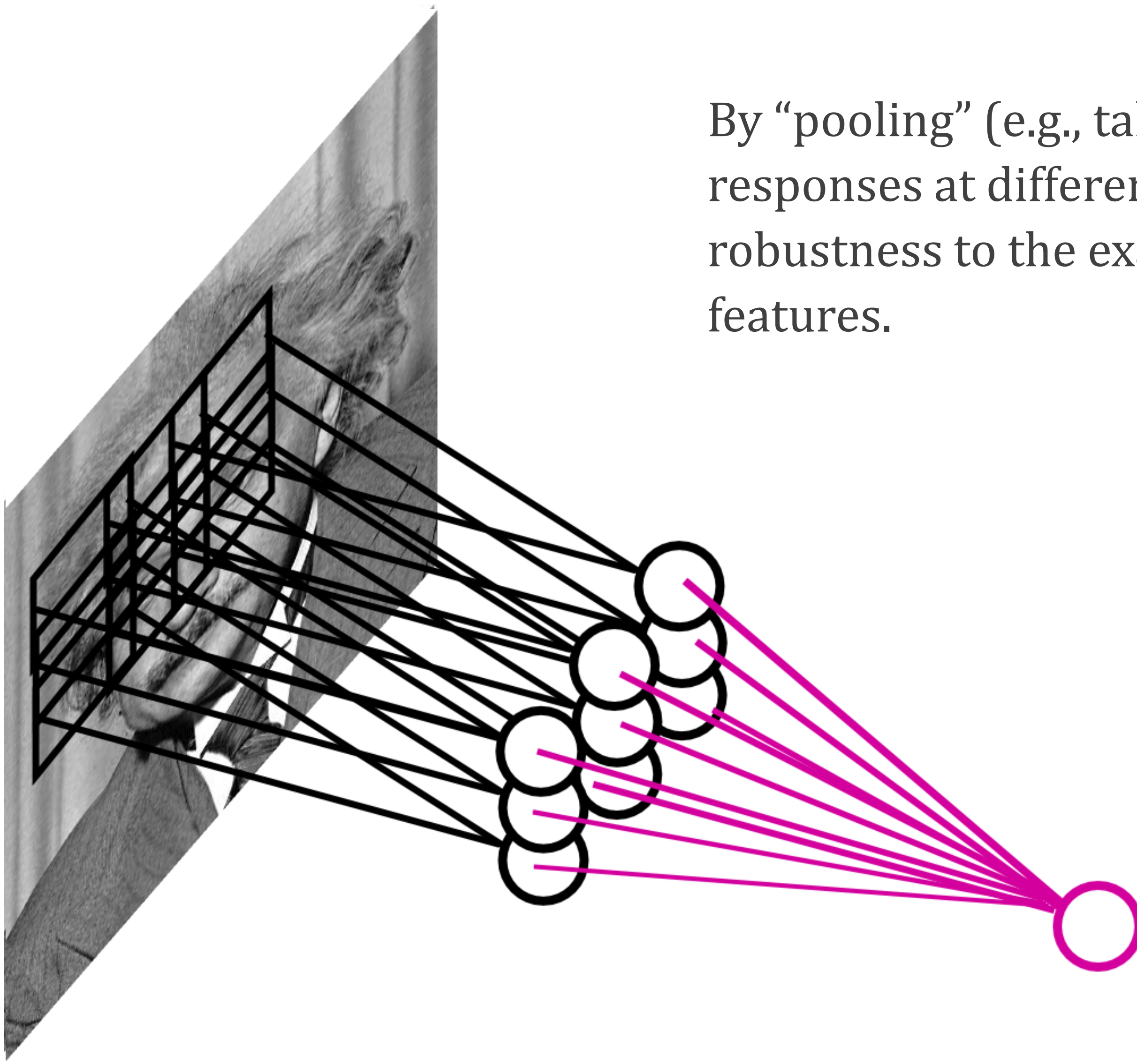
- Introduces invariance to local translations
- Reduces the number of hidden units in hidden layer

Example: Pooling



Can we make the detection robust to the exact location of the eye?

Example: Pooling

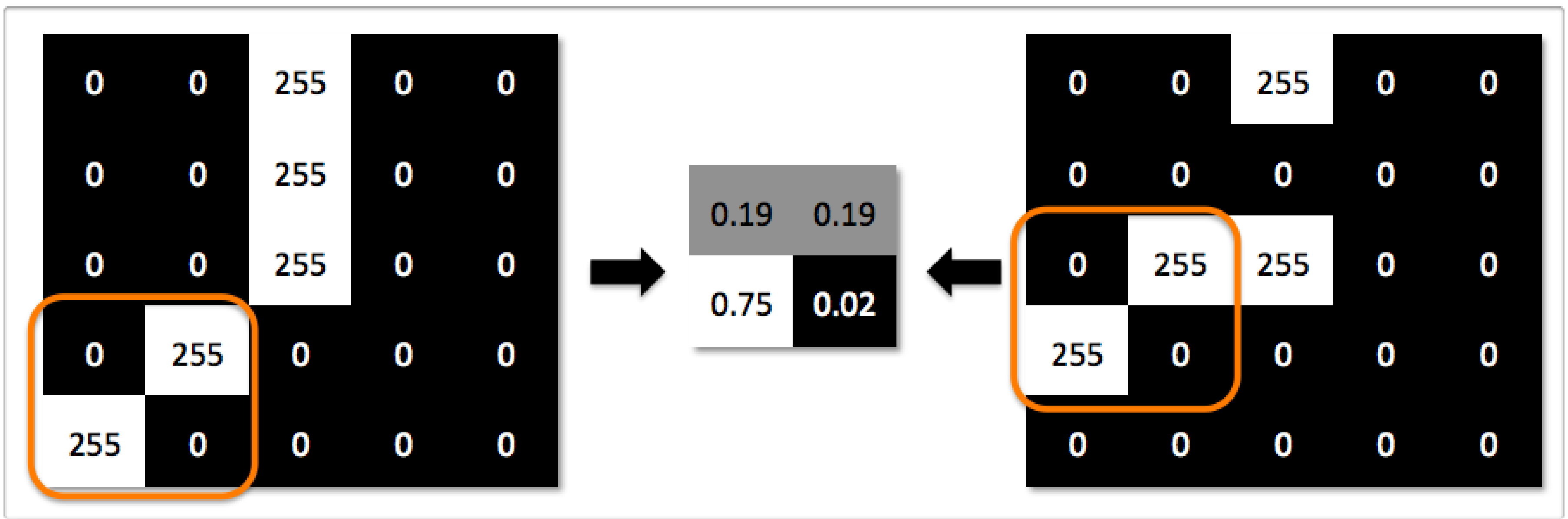


By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Translation Invariance

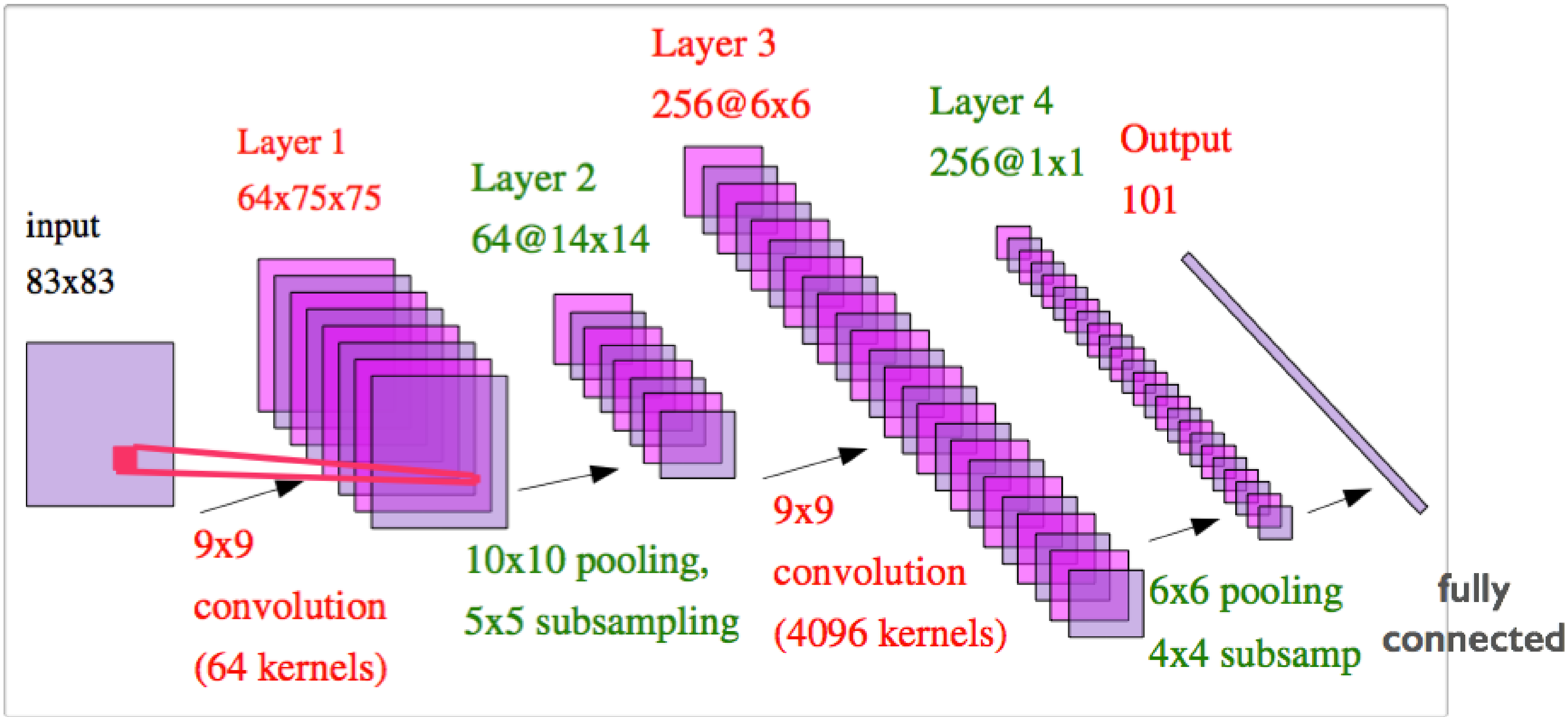
Illustration of local translation invariance

Both images result in the same feature map after pooling/subsampling



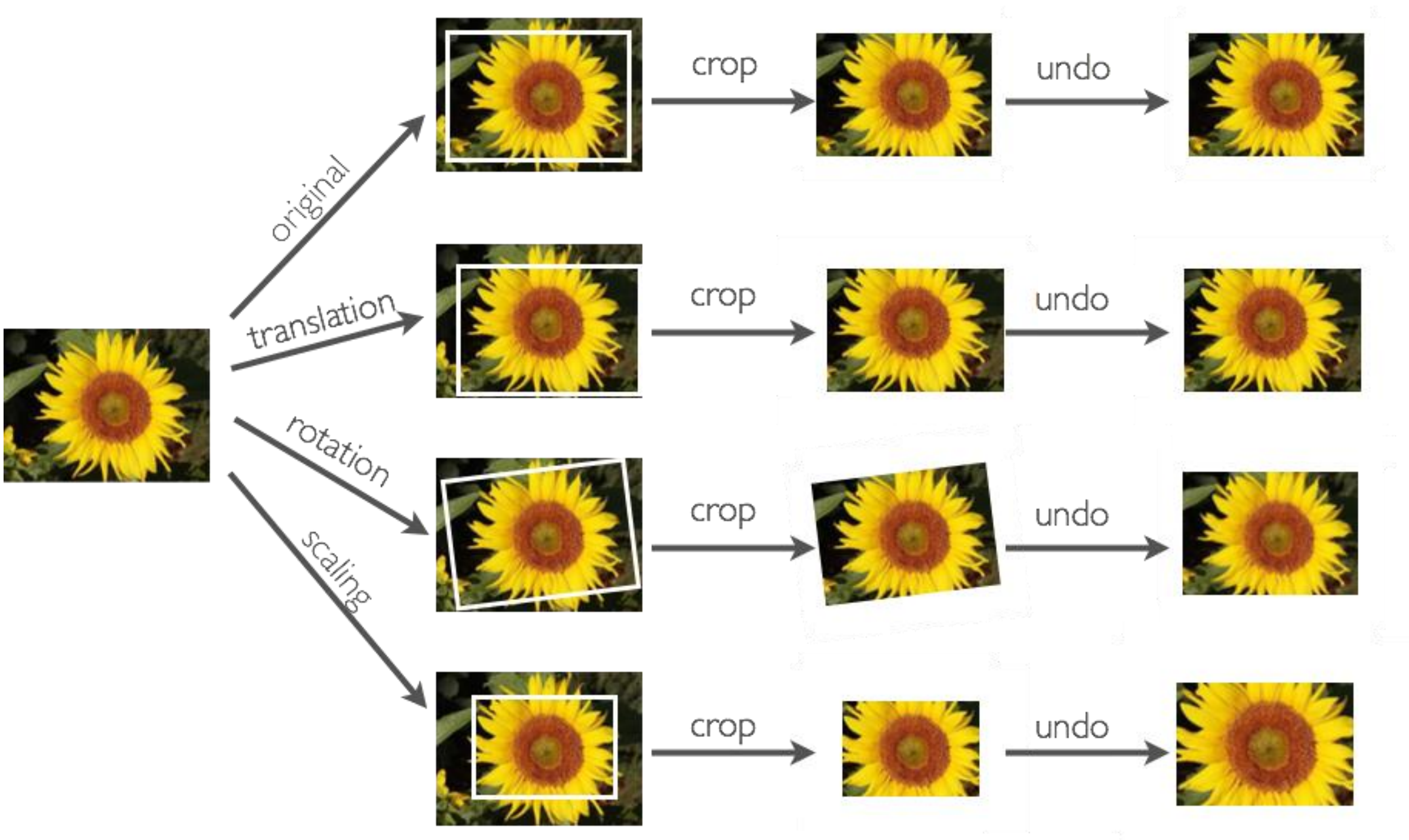
Convolutional Network

Convolutional neural network alternates between convolutional and pooling layers



From Yann LeCun's slides

Generating Additional Examples

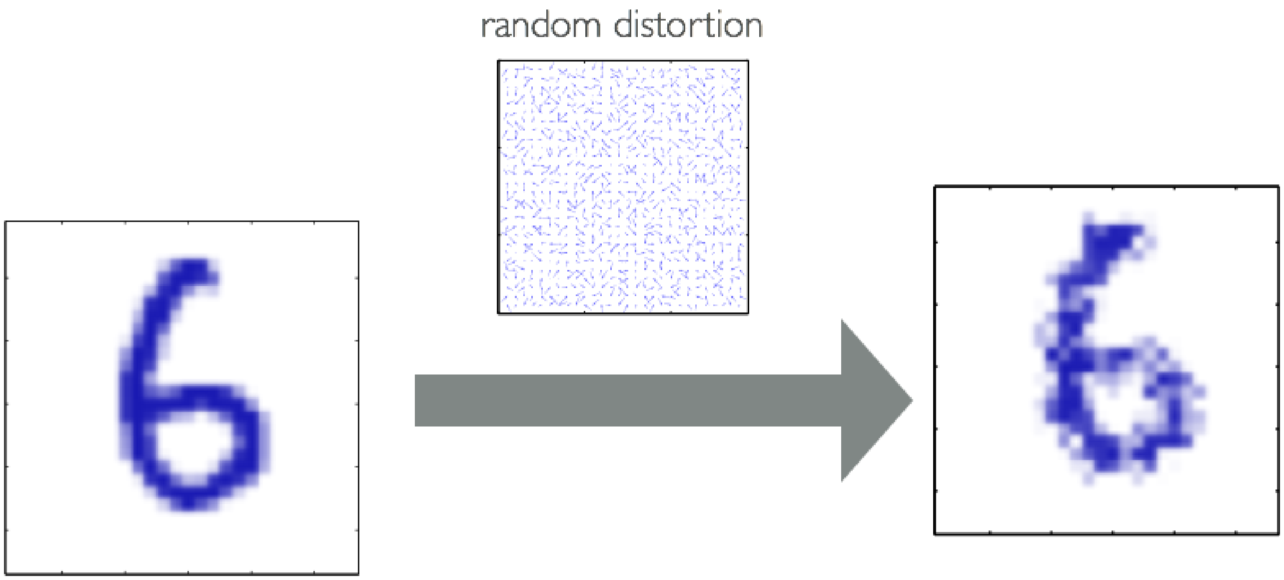


Elastic Distortions

Can add "elastic" deformations (useful in character recognition)

We can do this by applying a "distortion field" to the image

A distortion field specifies where to displace each pixel value



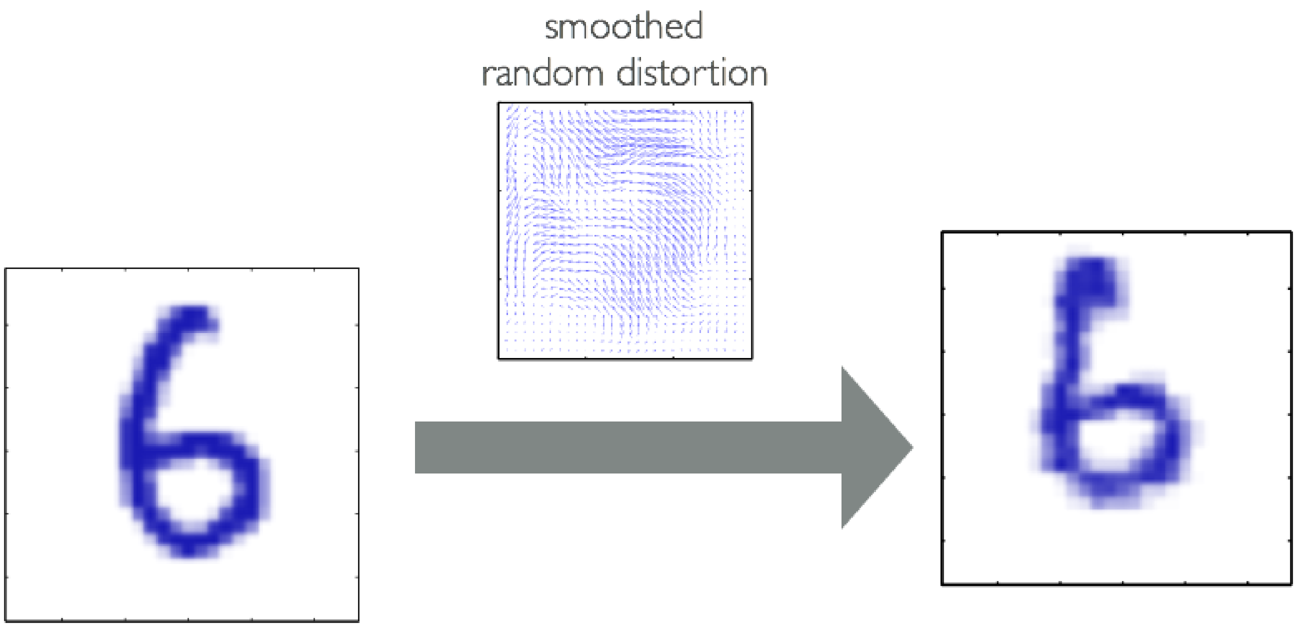
Bishop's book

Elastic Distortions

Can add "elastic" deformations (useful in character recognition)

We can do this by applying a "distortion field" to the image

A distortion field specifies where to displace each pixel value



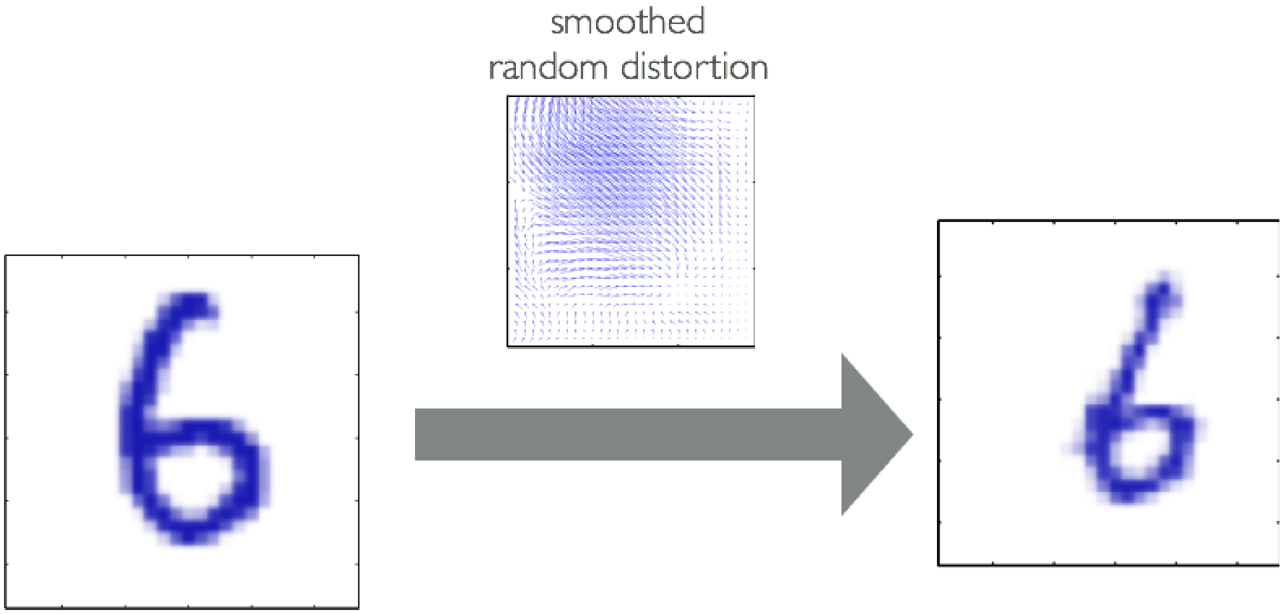
Bishop's book

Elastic Distortions

Can add "elastic" deformations (useful in character recognition)

We can do this by applying a "distortion field" to the image

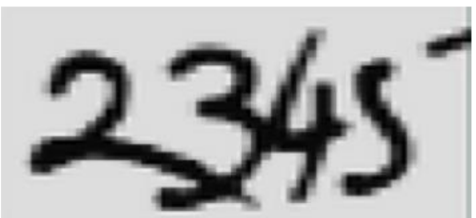
A distortion field specifies where to displace each pixel value



Bishop's book

Conv Nets: Examples

Optical Character Recognition, House Number and Traffic Sign classification



Ciresan et al. “MCDNN for image classification” CVPR 2012

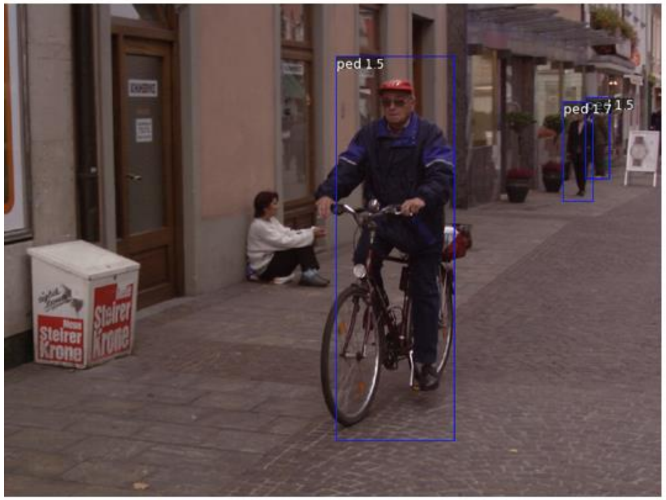
Wan et al. “Regularization of neural networks using dropconnect” ICML 2013

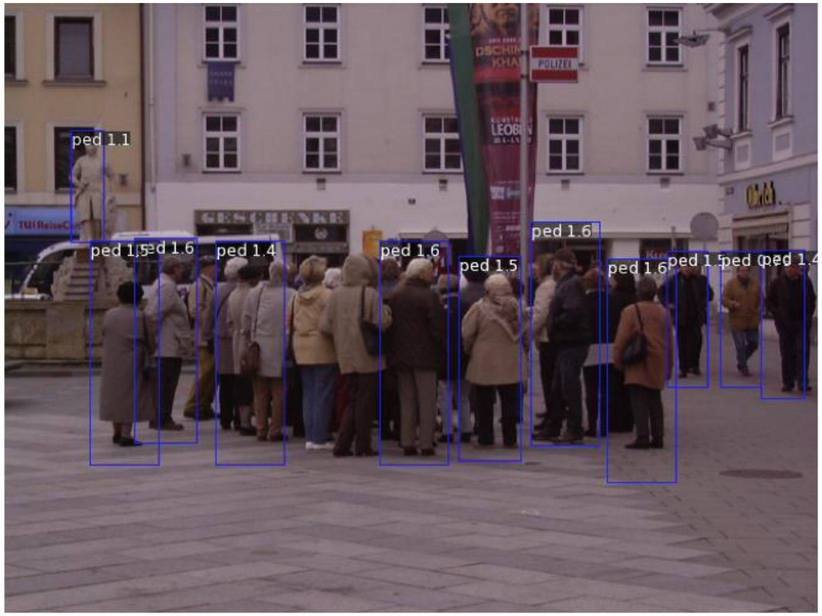
Goodfellow et al. “Multi-digit nuber recognition from StreetView...” ICLR 2014

Jaderberg et al. “Synthetic data and ANN for natural scene text recognition” arXiv 2014

Conv Nets: Examples

Pedestrian detection

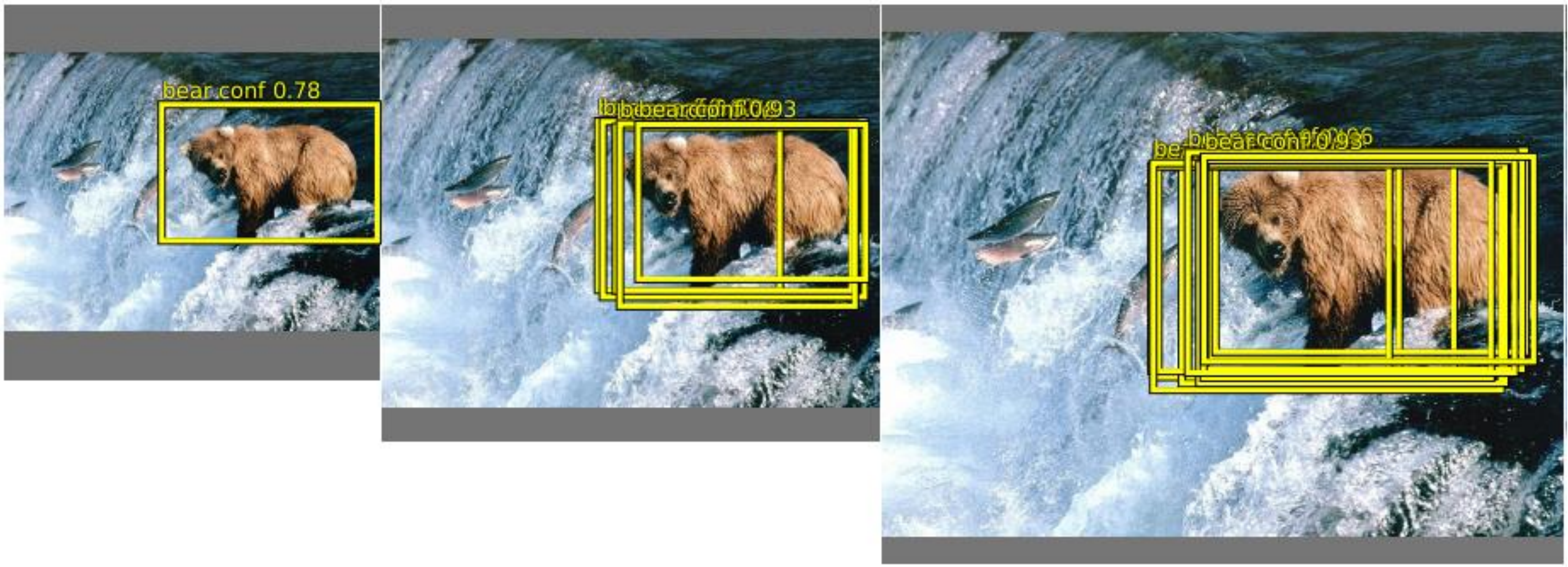




Sermanet et al. “Pedestrian detection with unsupervised multi-stage..” CVPR 2013

Conv Nets: Examples

Object Detection

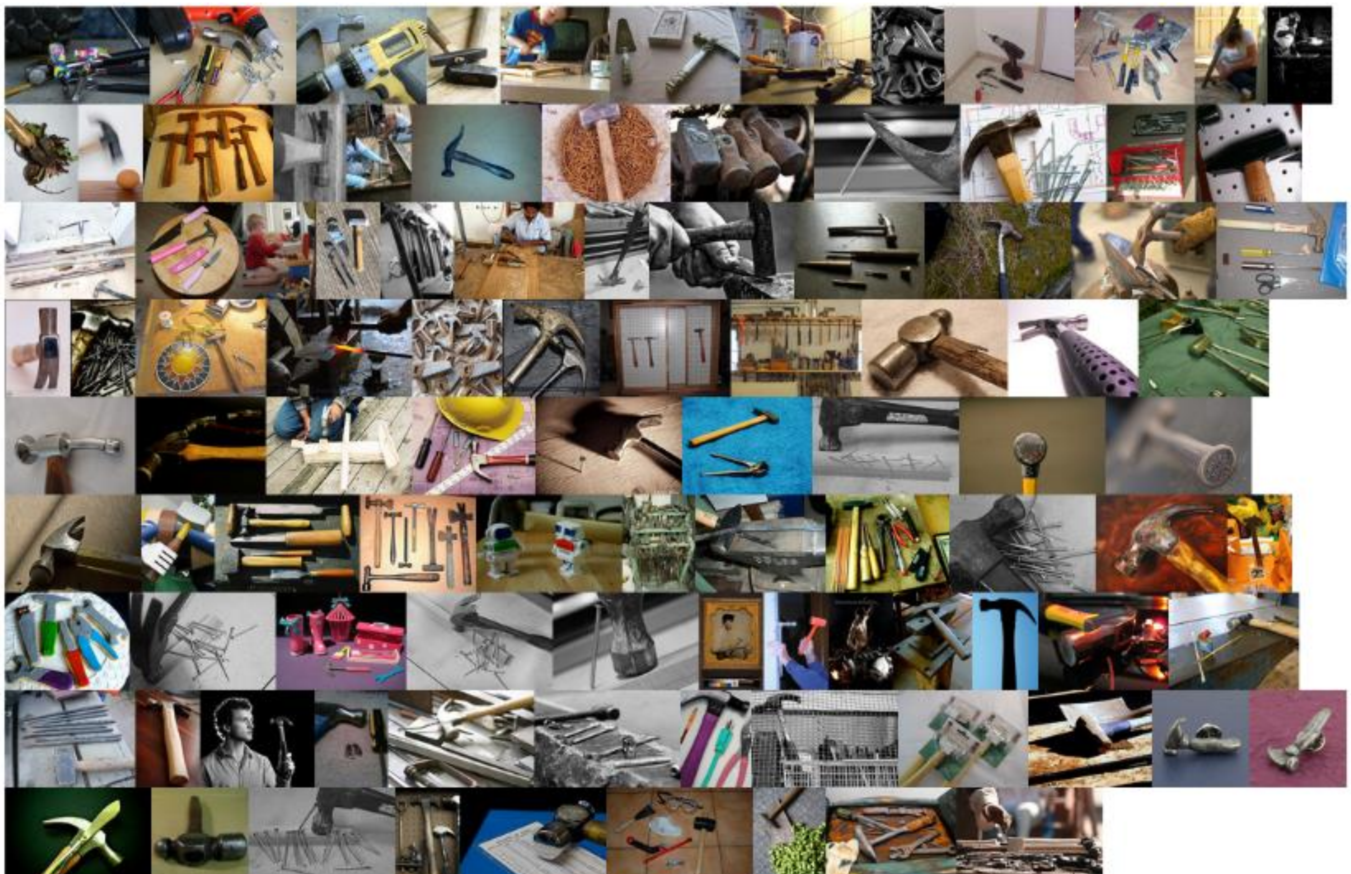


Sermanet et al. “OverFeat: Integrated recognition, localization” arxiv 2013
Girshick et al. “Rich feature hierarchies for accurate object detection” arxiv 2013
Szegedy et al. “DNN for object detection” NIPS 2013

ImageNet Dataset

1.2 million images, 1000 classes

Examples of Hammer

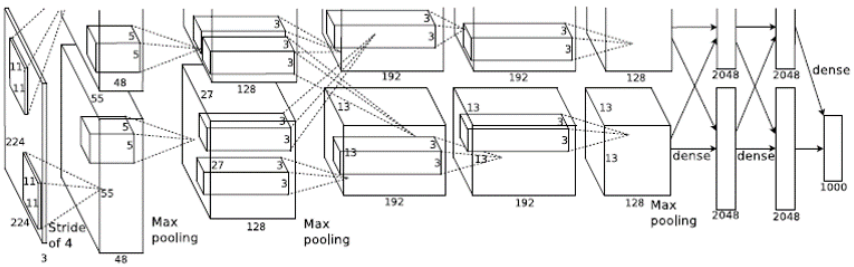


Deng et al. “Imagenet: a large scale hierarchical image database” CVPR 2009

Important Breakthroughs

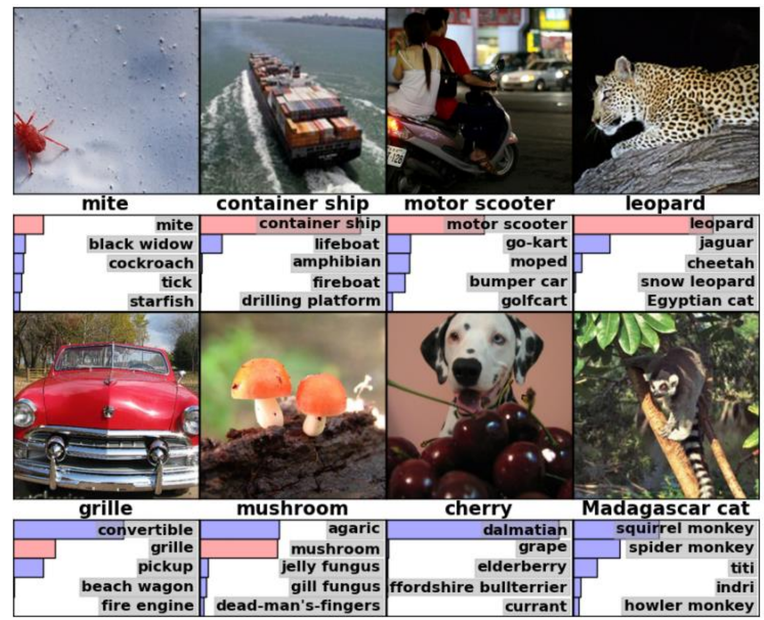
Deep Convolutional Nets for Vision (Supervised)

Krizhevsky, A., Sutskever, I. and Hinton, G. E., ImageNet Classification with Deep Convolutional Neural Networks, NIPS, 2012.



IMAGENET

1.2 million training images
1000 classes



Architecture

How can we select the “right” architecture:

Manual tuning of features is now replaced with the manual tuning of architectures

- Depth
- Width
- Parameter count

How to Choose Architecture

Many hyper-parameters:

Number of layers, number of feature maps

- Cross Validation
- Grid Search (need lots of GPUs)
- Smarter Strategies
  - Random search [*Bergstra & Bengio JMLR 2012*]
  - Bayesian Optimization

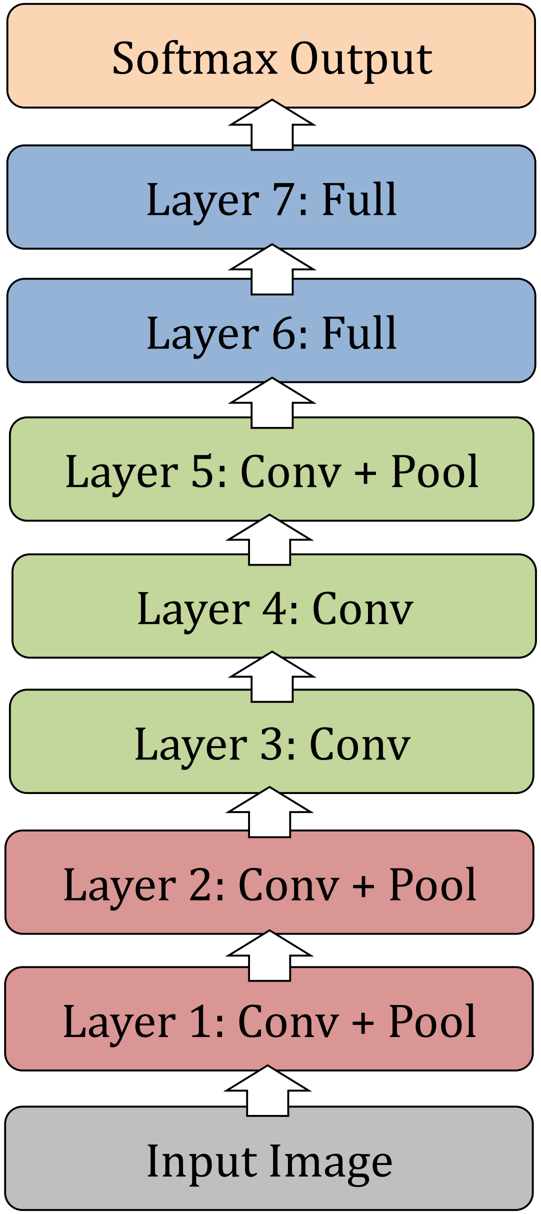
AlexNet

8 layers total

Trained on Imagenet dataset [Deng et al. CVPR'09]

18.2% top-5 error

[From Rob Fergus' CIFAR 2016 tutorial]

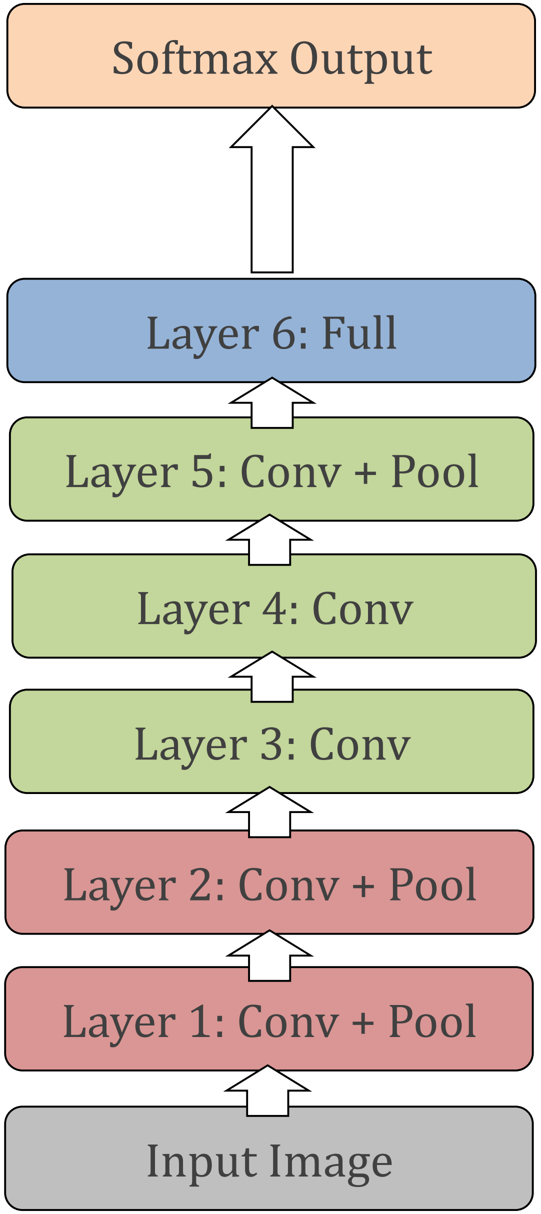


AlexNet: ablation study

Remove top fully connected layer 7

Drop ~16 million parameters

Only 1.1% drop in performance!



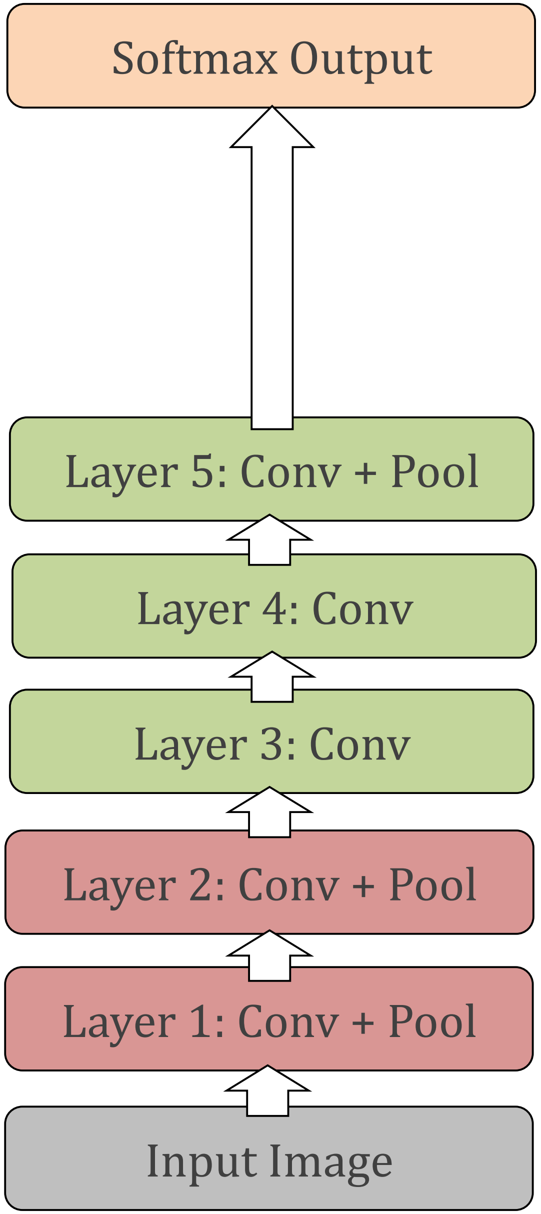
[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet: ablation study

Remove both fully connected layers 6,7

Drop ~50 million parameters

5.7% drop in performance!



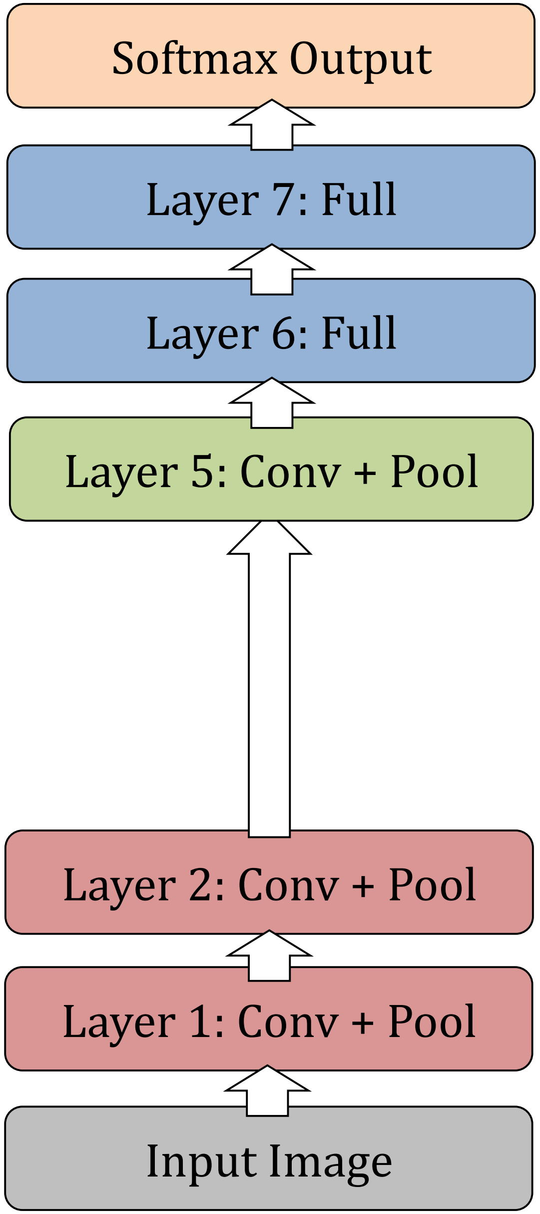
[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet: ablation study

Remove upper feature extractor layers (Layers 3 & 4)

Drop ~1 million parameters

3% drop in performance.



[From Rob Fergus' CIFAR 2016 tutorial]

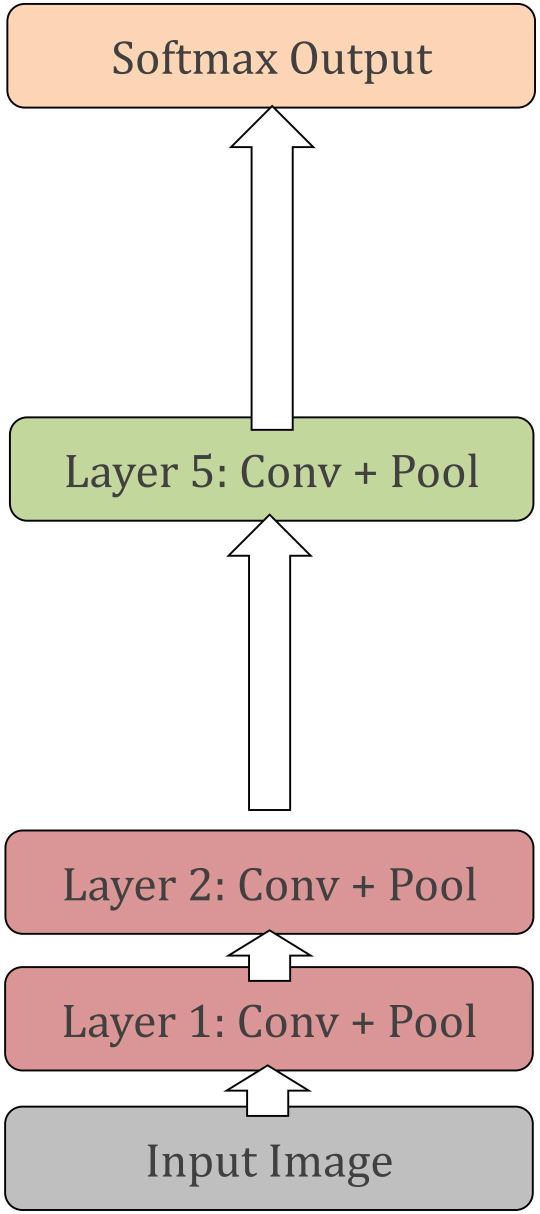
AlexNet: ablation study

Let us remove upper feature extractor layers and fully connected:

- Layers 3,4, 6 and 7

33.5% drop in performance!

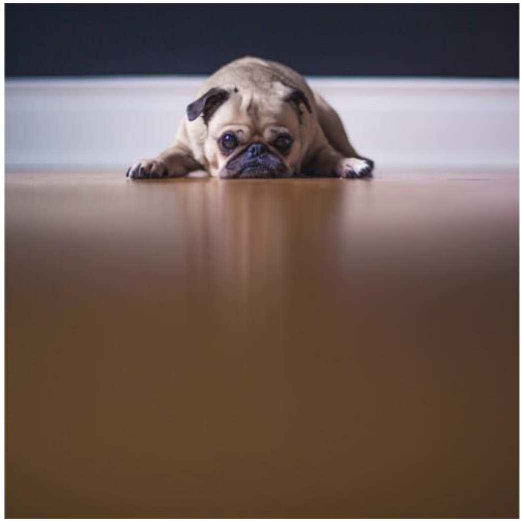
Depth of the network is the key.



[From Rob Fergus' CIFAR 2016 tutorial]

GoogLeNet

Issue: multiscale nature of images

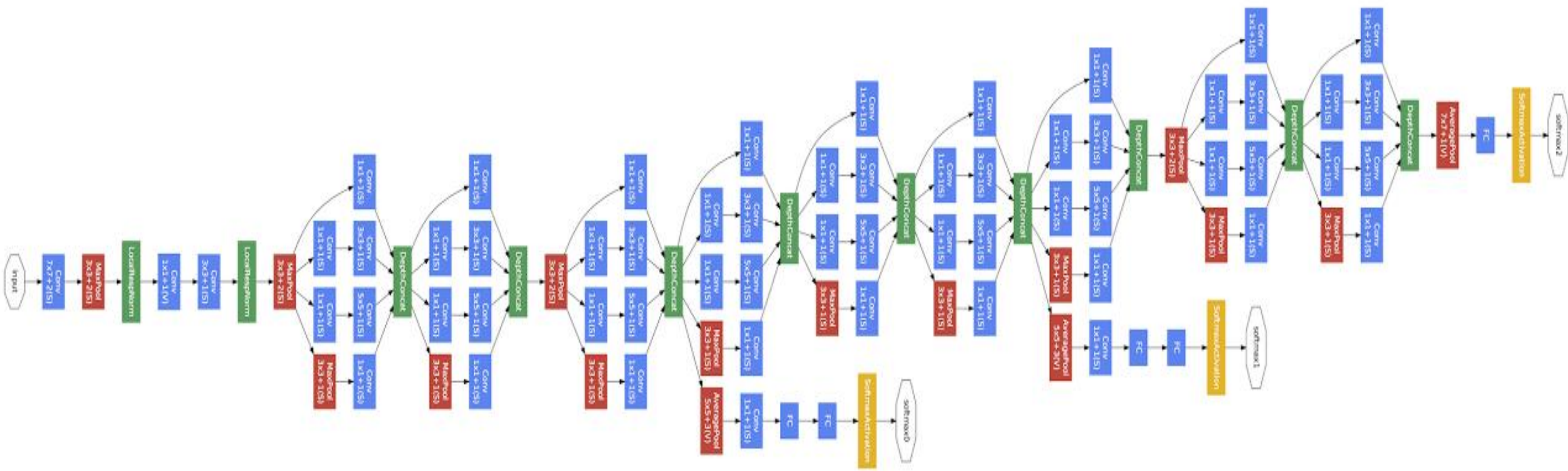


https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

Larger kernel good for global features, and smaller kernel for local features.

Idea: have multiple different-size kernels at any given level.

GoogLeNet



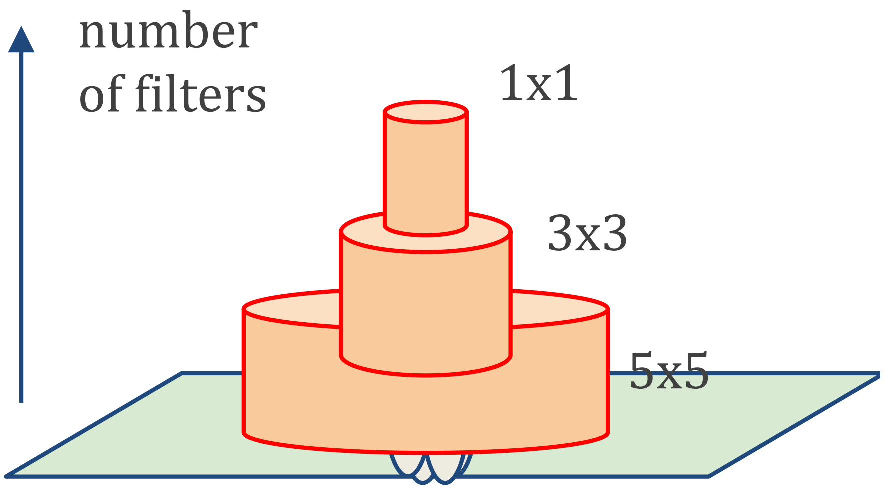
24-layer model that uses so-called **inception module**.

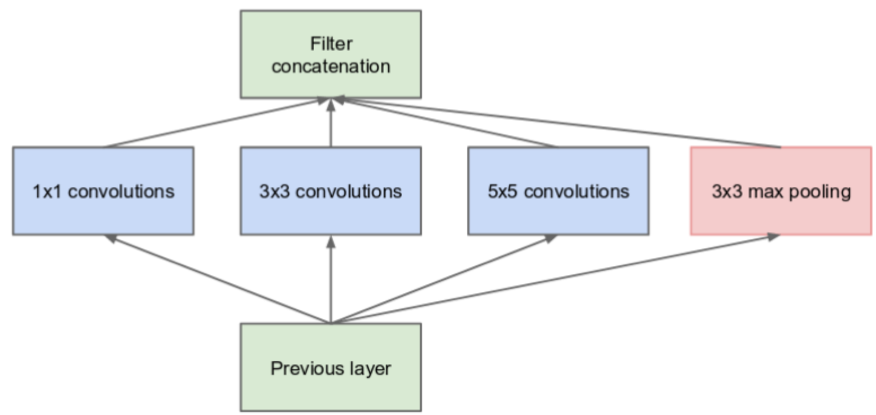
[Going Deep with Convolutions, Szegedy et al., arXiv:1409.4842, 2014]

GoogLeNet

GoogLeNet inception module:

- Multiple filter scales at each layer





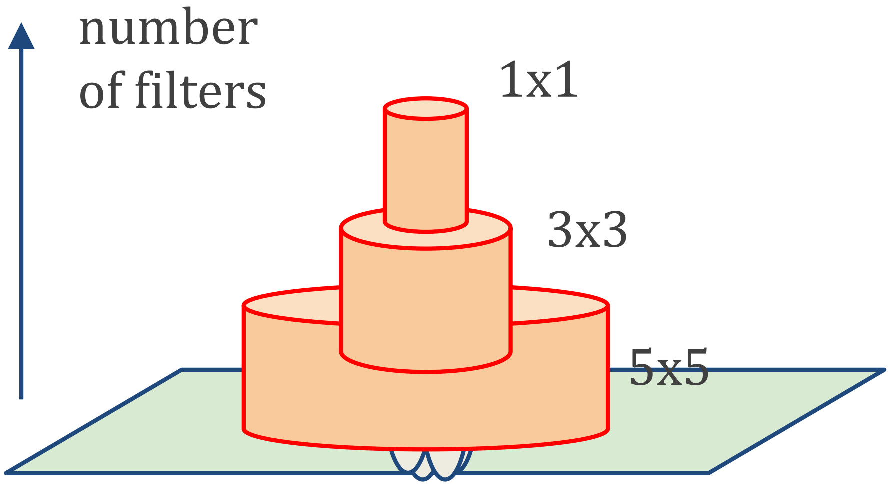
(a) Inception module, naïve version

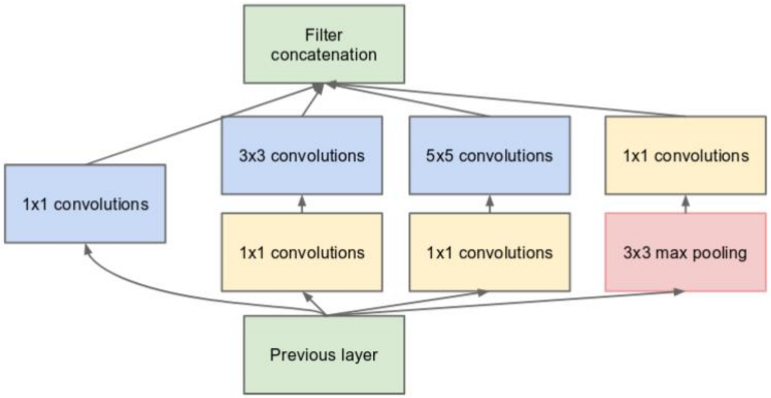
[Going Deep with Convolutions, Szegedy et al., arXiv:1409.4842, 2014]

GoogLeNet

GoogLeNet inception module:

- Multiple filter scales at each layer
- Dimensionality reduction to keep computational requirements down

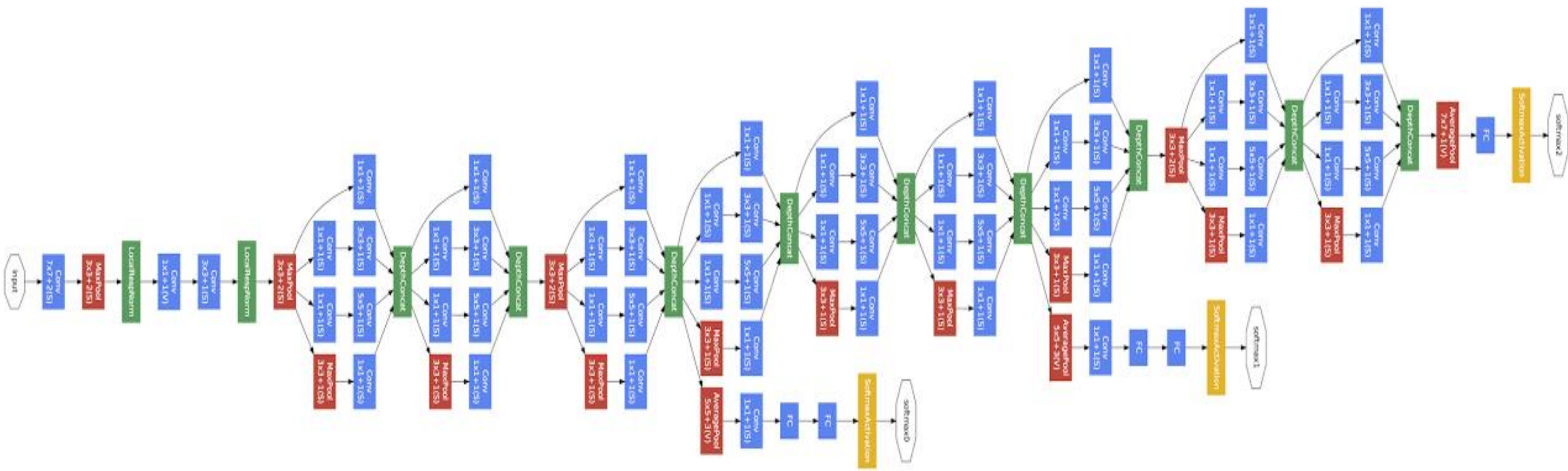




(b) Inception module with dimension reductions

[Going Deep with Convolutions, Szegedy et al., arXiv:1409.4842, 2014]

GoogLeNet

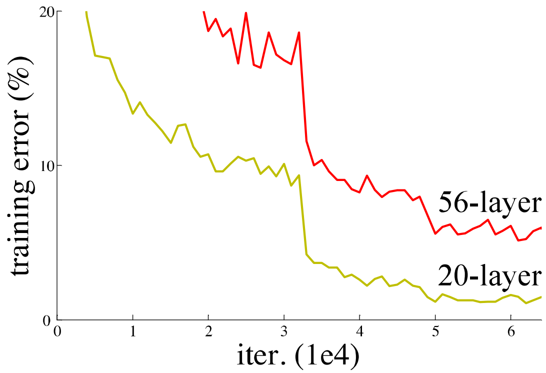


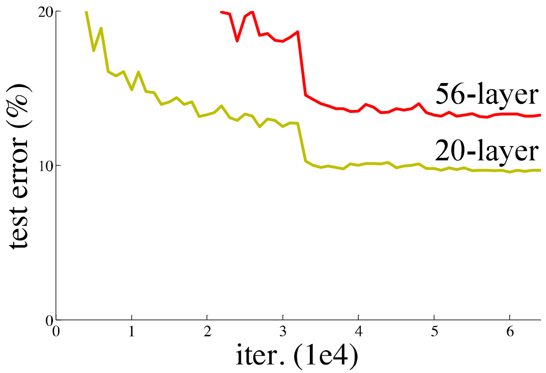
- Width of inception modules ranges from 256 filters (in early modules) to 1024 in top inception modules.
- Can remove fully connected layers on top completely
- Number of parameters is reduced to 5 million
- 6.7% top-5 validation error on Imagnet

[Going Deep with Convolutions, Szegedy et al., arXiv:1409.4842, 2014]

Residual Networks

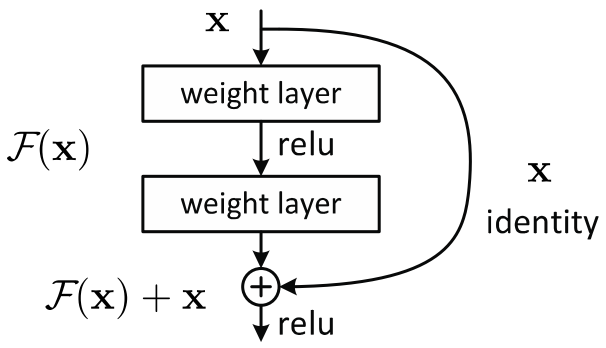
Really, really deep convnets do not train well, E.g. CIFAR10:





Reason: gradients involve multiplications of a # of matrices proportional to depth.

Vanishing/exploding gradients: gradients get very small/very large.



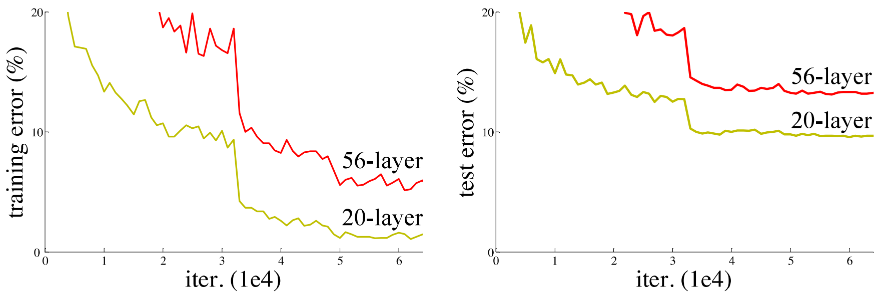
Key idea: introduce "identity shortcut" connection, skipping one or more layers.

Intuition: network can easily simulate shallower network (at initialization, F is not too far from 0 map), so performance should not degrade by going deeper.

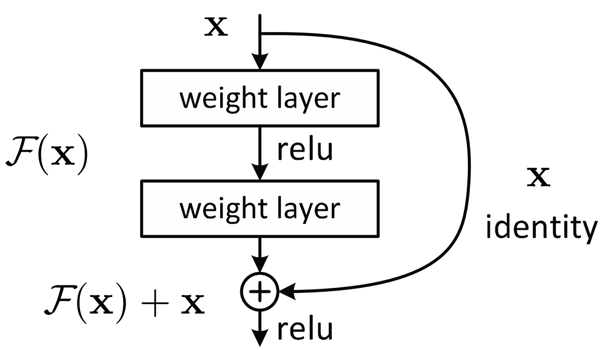
[He, Zhang, Ren, Sun, CVPR 2016]

Residual Networks

Really, really deep convnets do not train well,
E.g. CIFAR10:



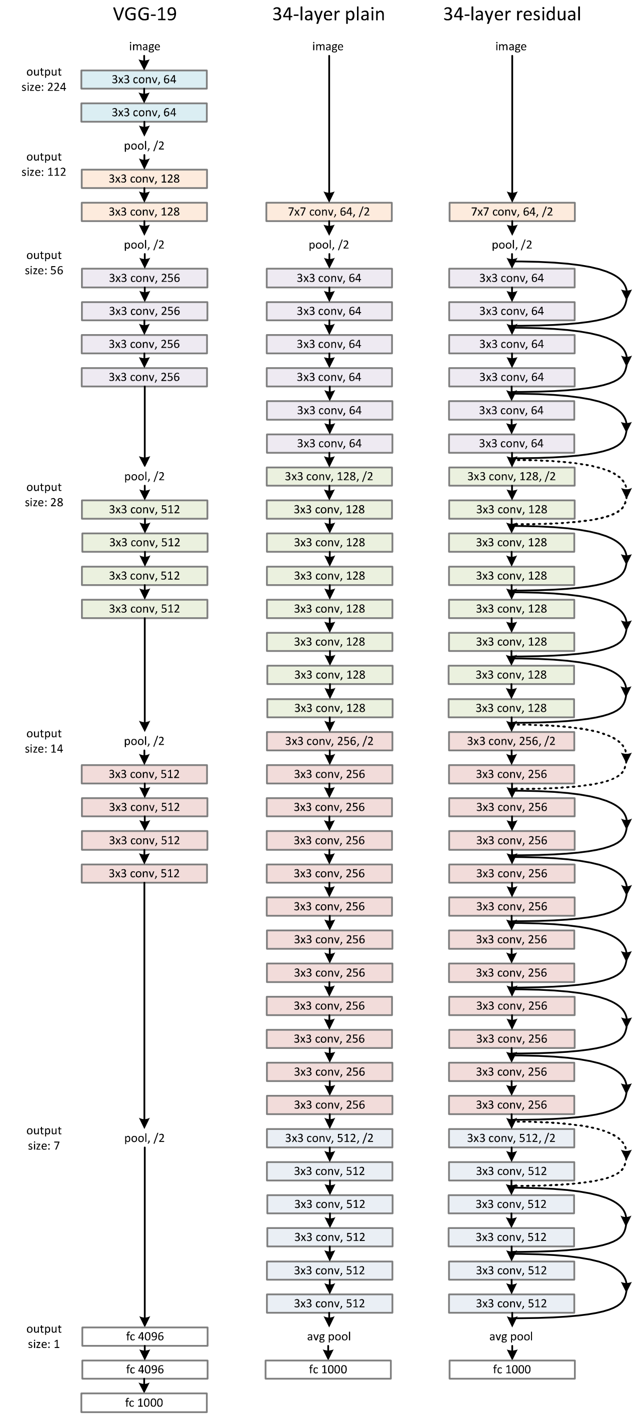
Key idea: introduce "identity shortcut"



method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

With ensembling, 3.57% top-5 test error on ImageNet



[He, Zhang, Ren, Sun, CVPR 2016]

Dense Convolutional Networks

Information in ResNets is only carried implicitly, through addition.

Idea: explicitly forward output of layer to ***all*** future layers (by **concatenation**).

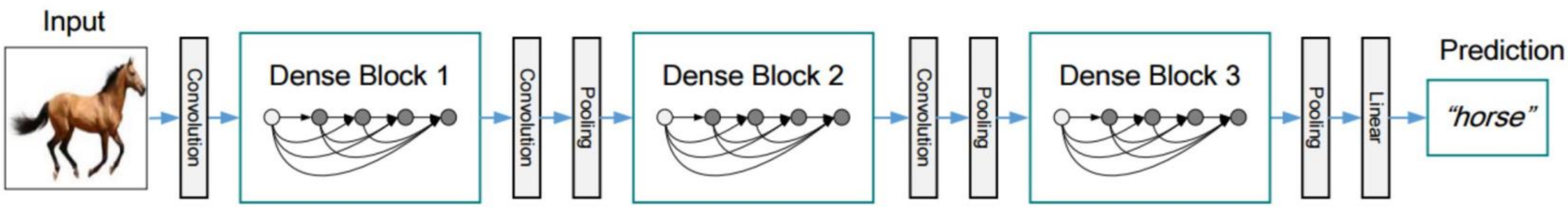
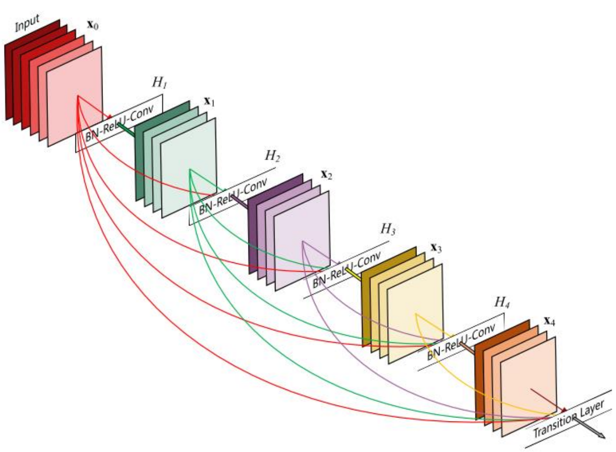


Figure 2. A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature map sizes via convolution and pooling.

Intuition: helps vanishing gradients; encourage reuse features (& hence reduce parameter count);



Layers	Output Size	DenseNet-121 $(k = 32)$	DenseNet-169 $(k = 32)$	DenseNet-201 $(k = 32)$	DenseNet-161 $(k = 48)$
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 36$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

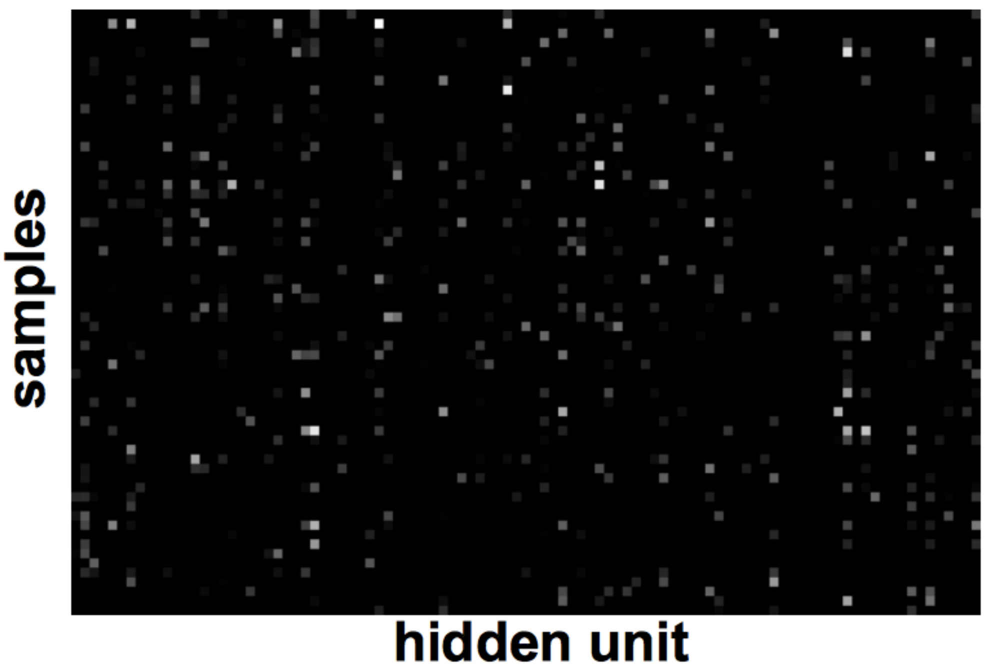
Full architecture for Imagenet

[Huang, Liu, Weinberger, van der Maaten, CVPR 2017]

Debugging hints

- Check gradients numerically by finite differences

- Visualize features (feature maps need to be uncorrelated) and have high variance



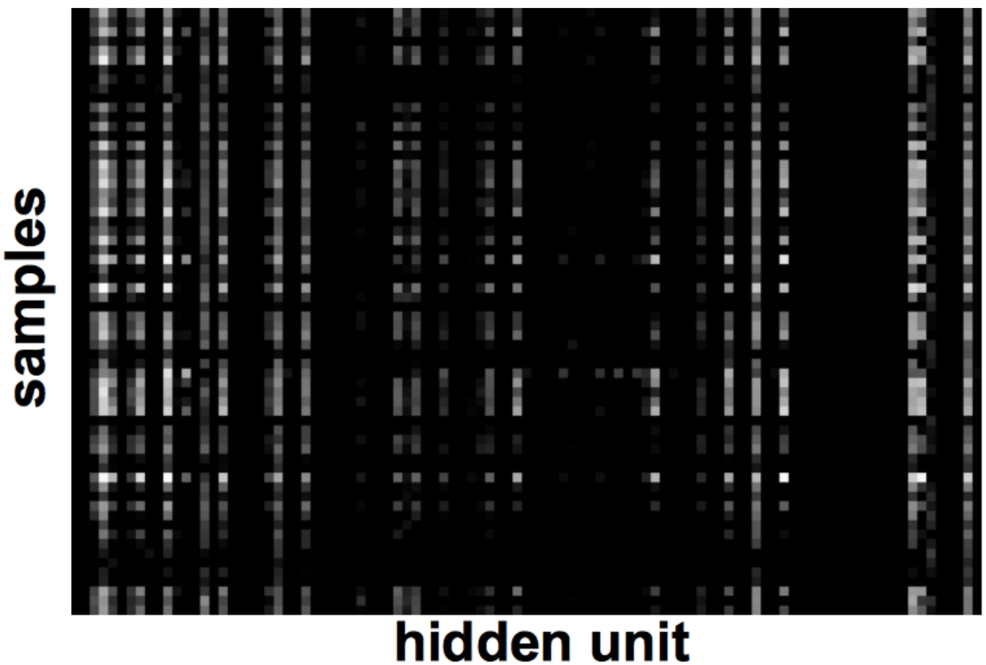
Good training: hidden units are sparse across samples

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Debugging hints

- Check gradients numerically by finite differences

- Visualize features (feature maps need to be uncorrelated) and have high variance

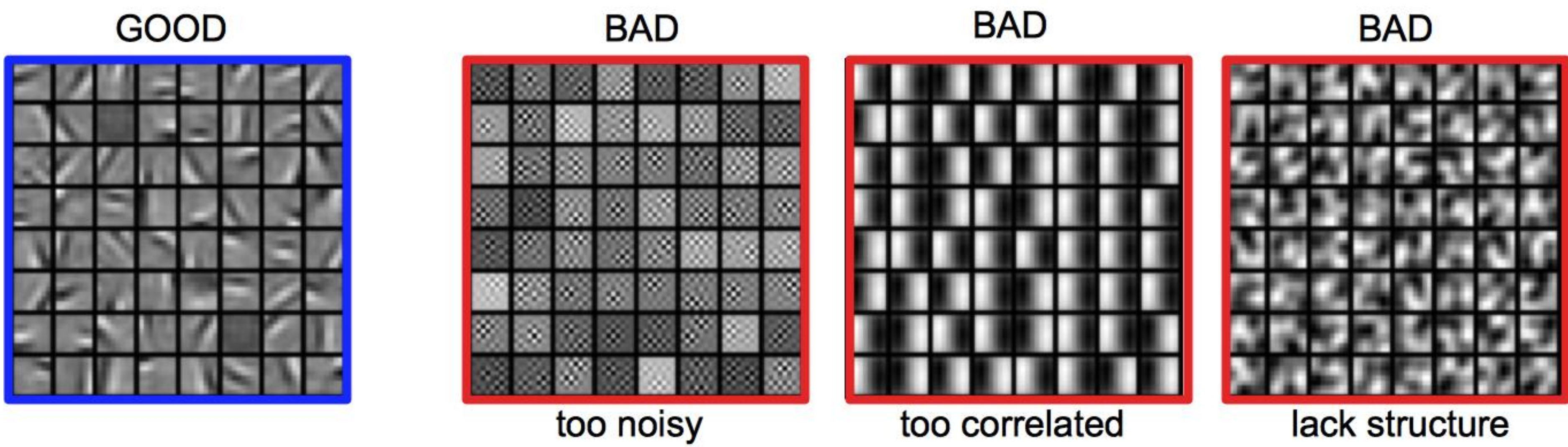


Bad training: many hidden units ignore the input and/or exhibit strong correlations

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Debugging hints

- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance
- Visualize parameters: learned features should exhibit structure and should be uncorrelated and are uncorrelated



[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Debugging hints

- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance
- Visualize parameters: learned features should exhibit structure and should be uncorrelated and are uncorrelated
- Measure error on both training and validation set
- Test on a small subset of the data and check the error $\rightarrow 0$.

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

When it does not work

Training diverges:

- Learning rate may be too large → decrease learning rate
- Backprop is buggy → numerical gradient checking

Parameters collapse / loss is minimized but accuracy is low

- Check loss function: is it appropriate for the task you want to solve?
- Does it have degenerate solutions?

Network is underperforming

- Compute flops and nr. params. → if too small, make net larger
- Visualize hidden units/params → fix optimization

Network is too slow

- GPU, distrib. framework, make net smaller

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]