Convolutional Neural Networks

Jian Tang

tangjianpku@gmail.com

Convolutional Neural Networks for Object Recognition

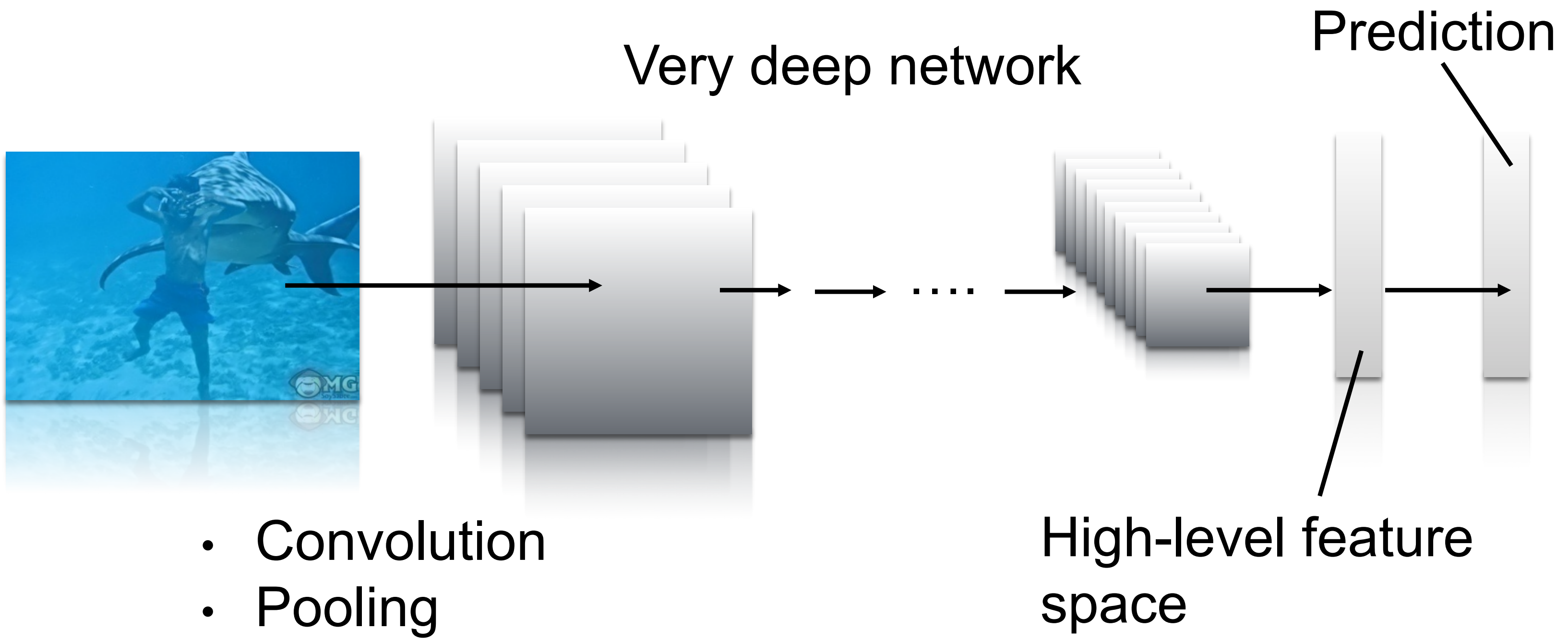
Object Recognition

Computer Vision

- Intuitions
  - Deal with very high-dimensional inputs: 150 x 150 pixels = 22500 inputs
  - Can exploit the 2D topology of pixels
  - Can build in invariance to certain variations, e.g., translation, etc.

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

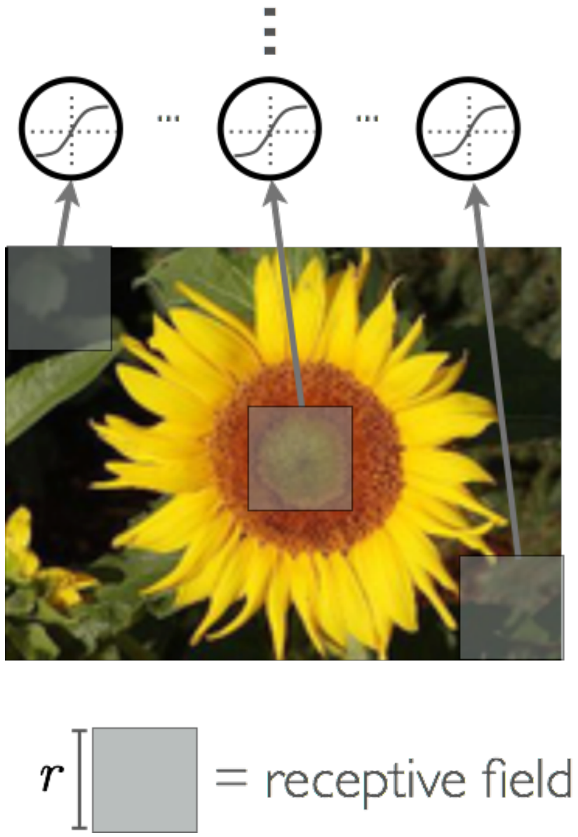
Deep Convolutional Neural Networks



- Convolution
- Pooling
- Normalization
- Densely connected

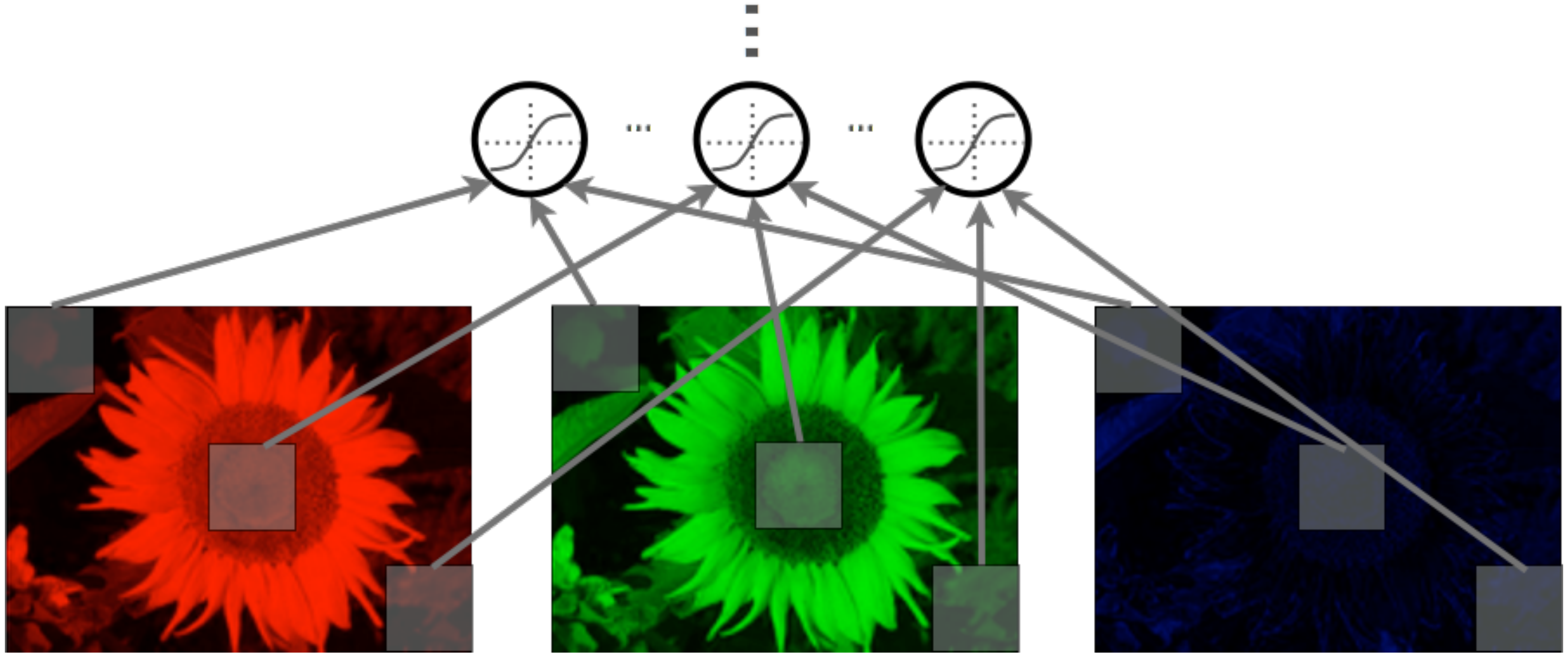
Local Connectivity

- Use a 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 local connectivity?
  - ➢ Fully connected layer has a lot of parameters to fit, requires a lot of data
  - ➢ Spatial correlation is local



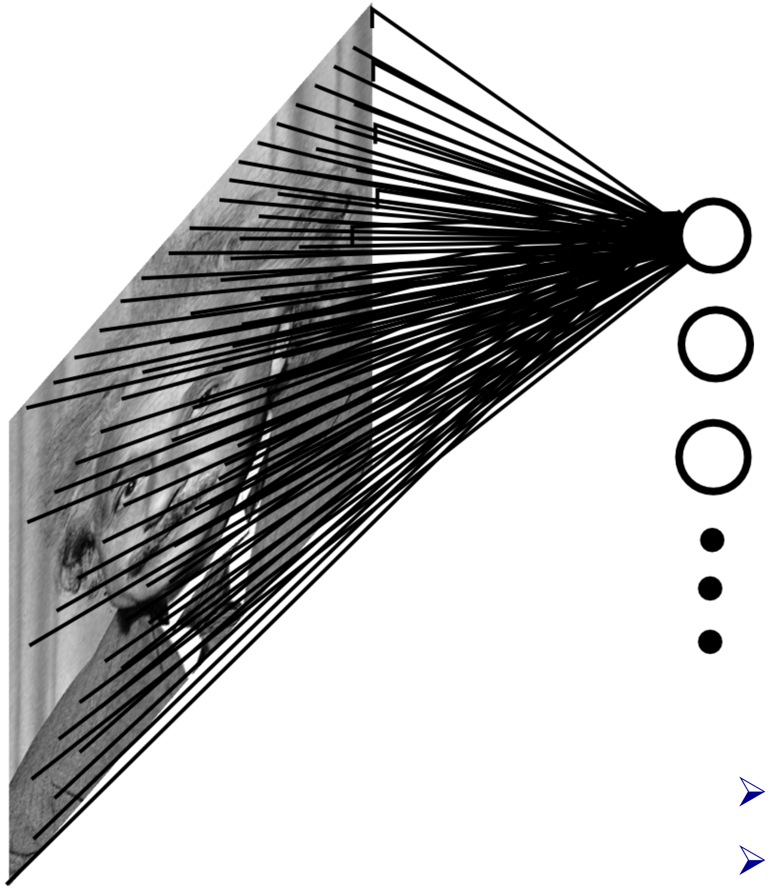
Local Connectivity

- Units are connected to all channels:
  - 1 channel if grayscale image,
  - 3 channels (R, G, B) if color image



Local Connectivity

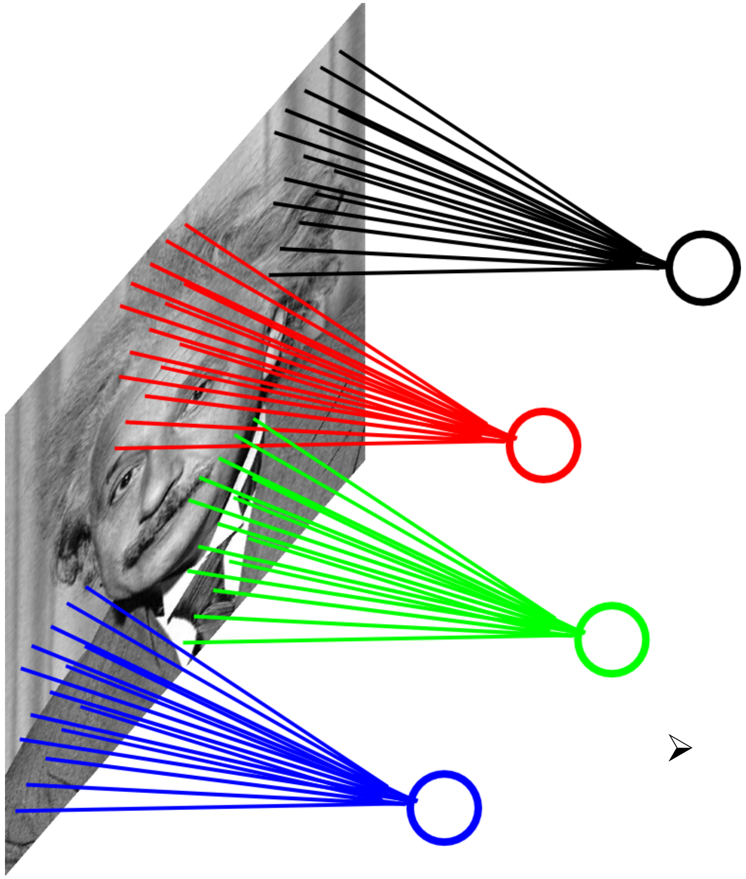
- Example: 200x200 image, 40K hidden units, ~2B parameters!



- ➢ Spatial correlation is local
- ➢ Too many parameters, will require a lot of training data!

Local Connectivity

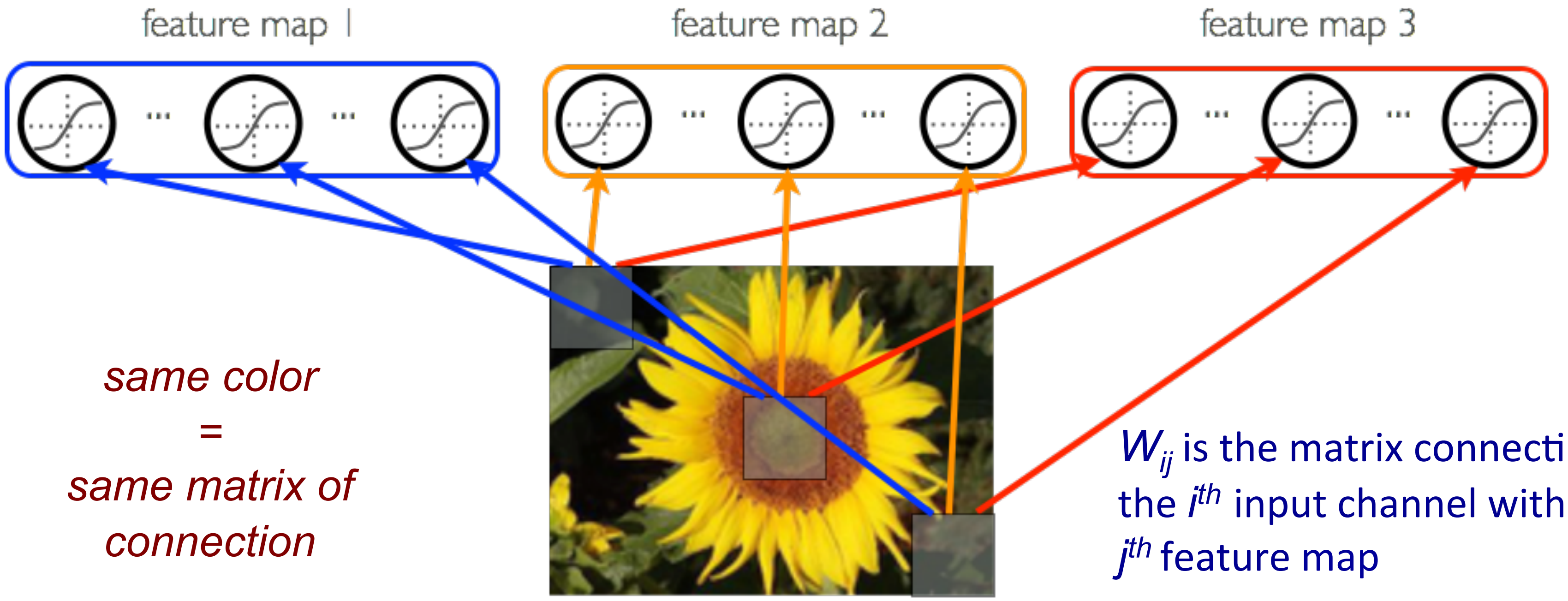
- Example: 200x200 image, 40k hidden units, filter size 10*10, ~4M parameters



➢ This parameterization is good when input image is registered

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

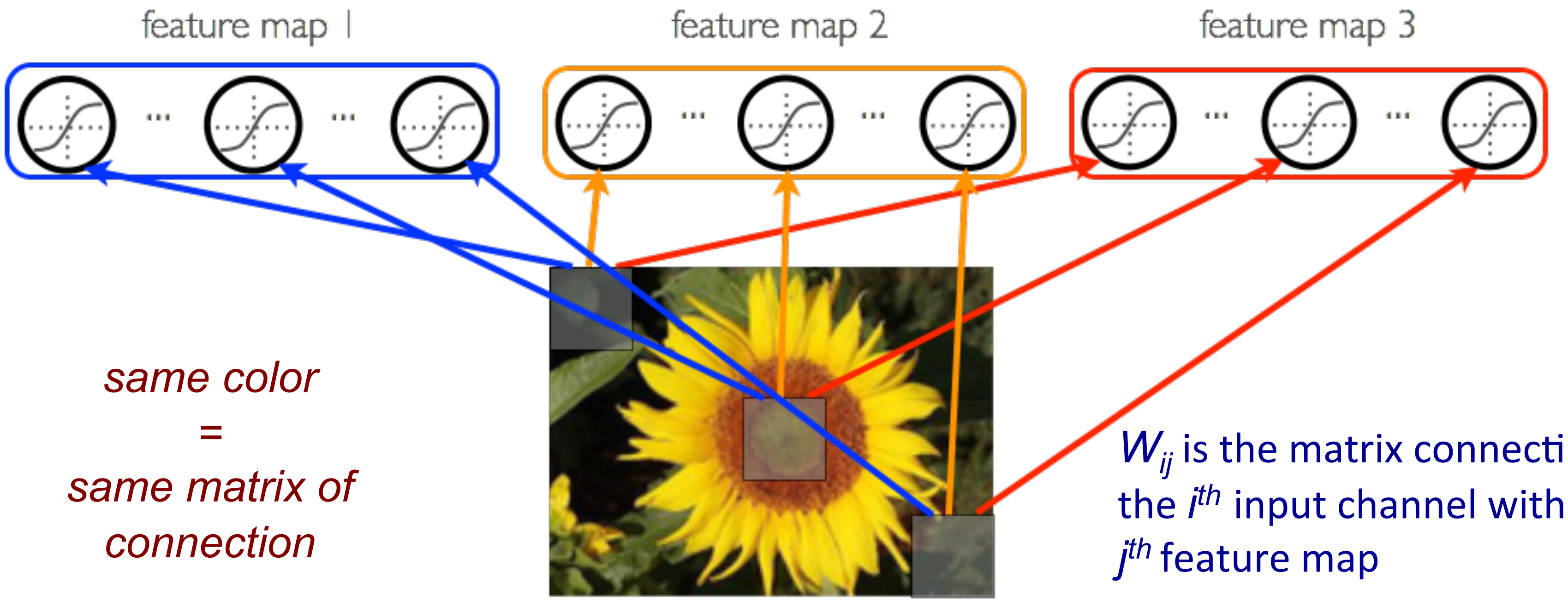


same color
=
same matrix of connection

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

Parameter Sharing

- Why parameter sharing?
  - ➢ Reduces even more the number of parameters
  - ➢ Will extract the same features at every position (features are “equivariant”)

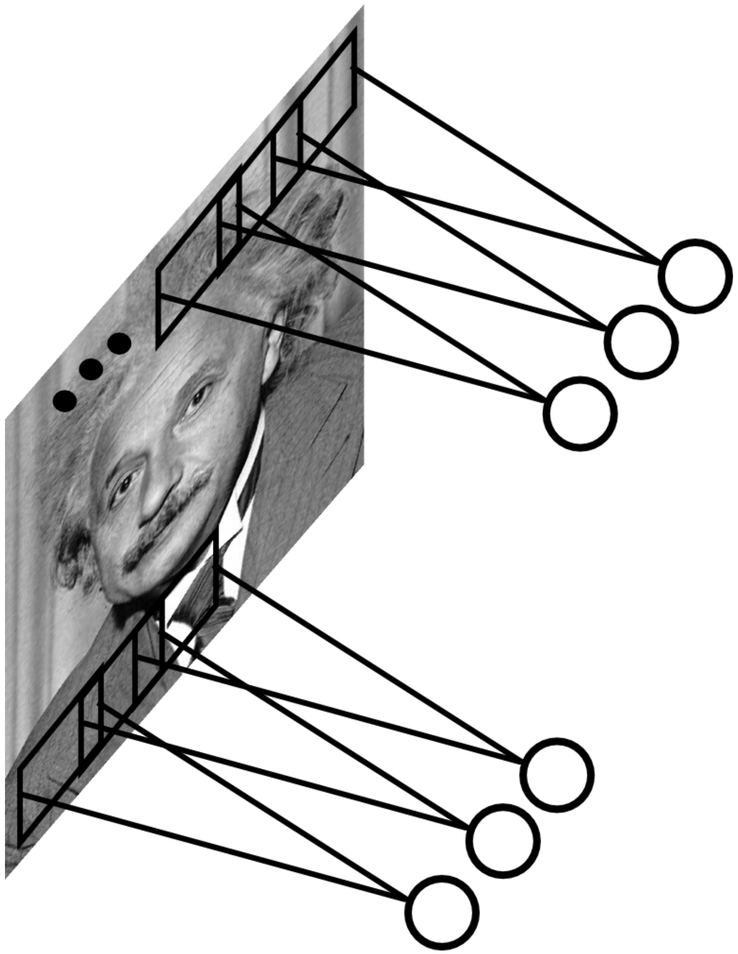


same color
=
same matrix of connection

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

Parameter Sharing

- Share matrix of parameters across certain units



➢ Convolutions with certain kernels

Discrete Convolution

- The convolution of an image x with a kernel k is computed as follows:

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

- Example:

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

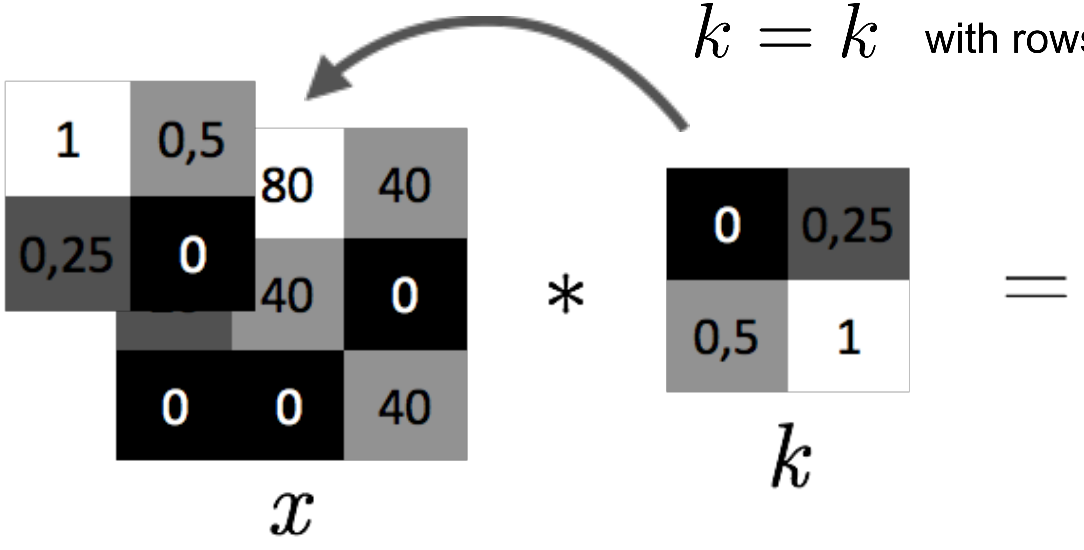
Discrete Convolution

- The convolution of an image x with a kernel k is computed as follows:

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

- Example:

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

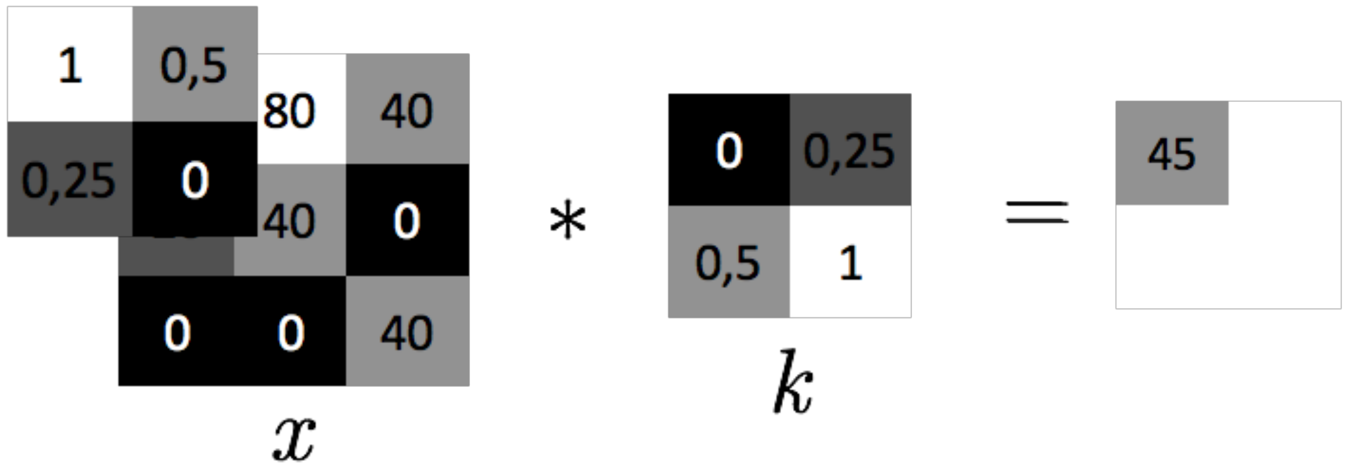


Discrete Convolution

- The convolution of an image x with a kernel k is computed as follows:

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

- Example: 1 x 0 + 0.5 x 80 + 0.25 x 20 + 0 x 40 = 45

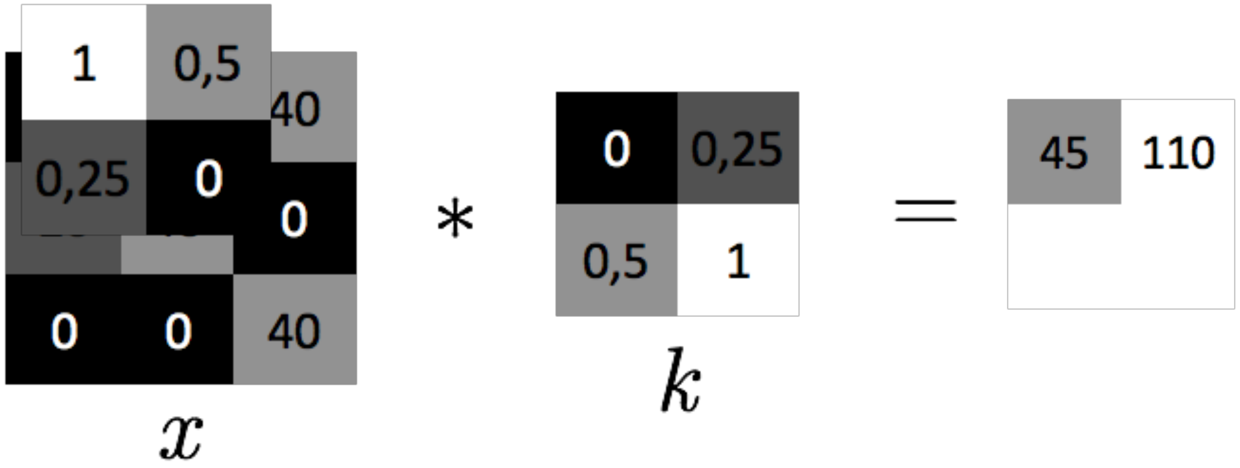


Discrete Convolution

- The convolution of an image x with a kernel k is computed as follows:

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

- Example: 1 x 80 + 0.5 x 40 + 0.25 x 40 + 0 x 0 = 110

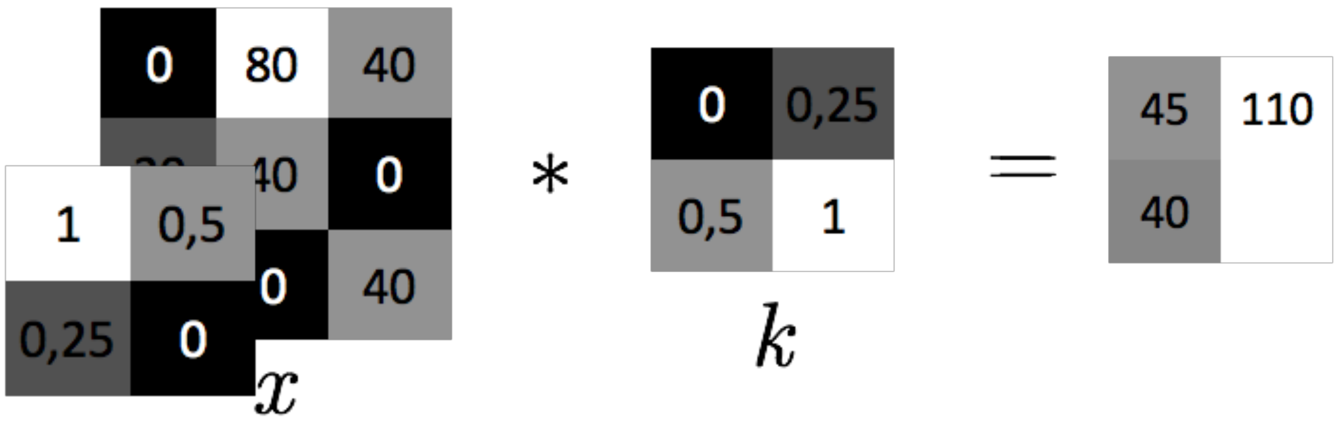


Discrete Convolution

- The convolution of an image x with a kernel k is computed as follows:

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

- Example: 1 x 20 + 0.5 x 40 + 0.25 x 0 + 0 x 0 = 40

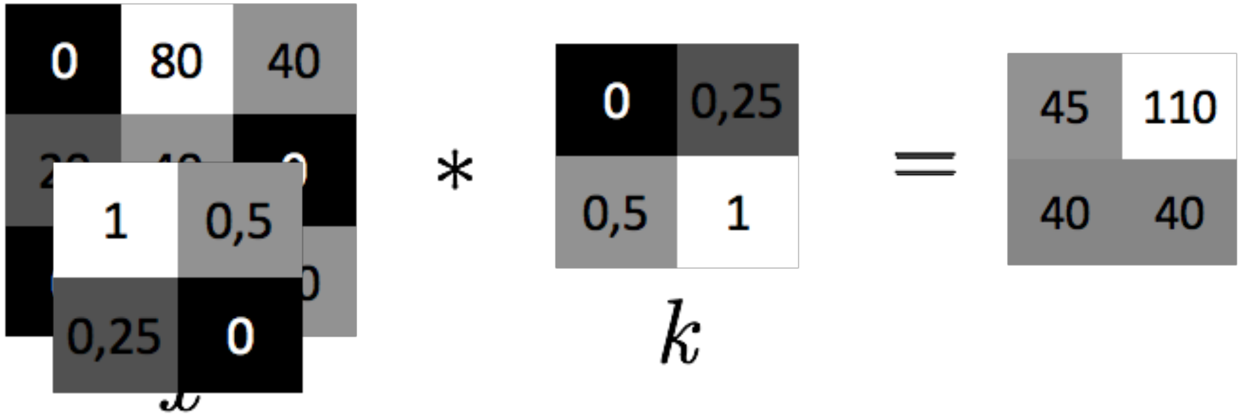


Discrete Convolution

- The convolution of an image x with a kernel k is computed as follows:

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

- Example: 1 x 40 + 0.5 x 0 + 0.25 x 0 + 0 x 40 = 40



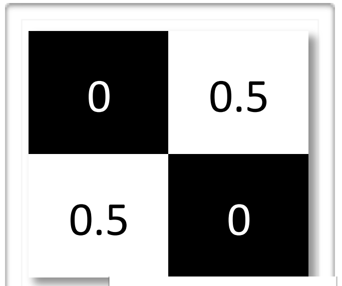
Discrete Convolution

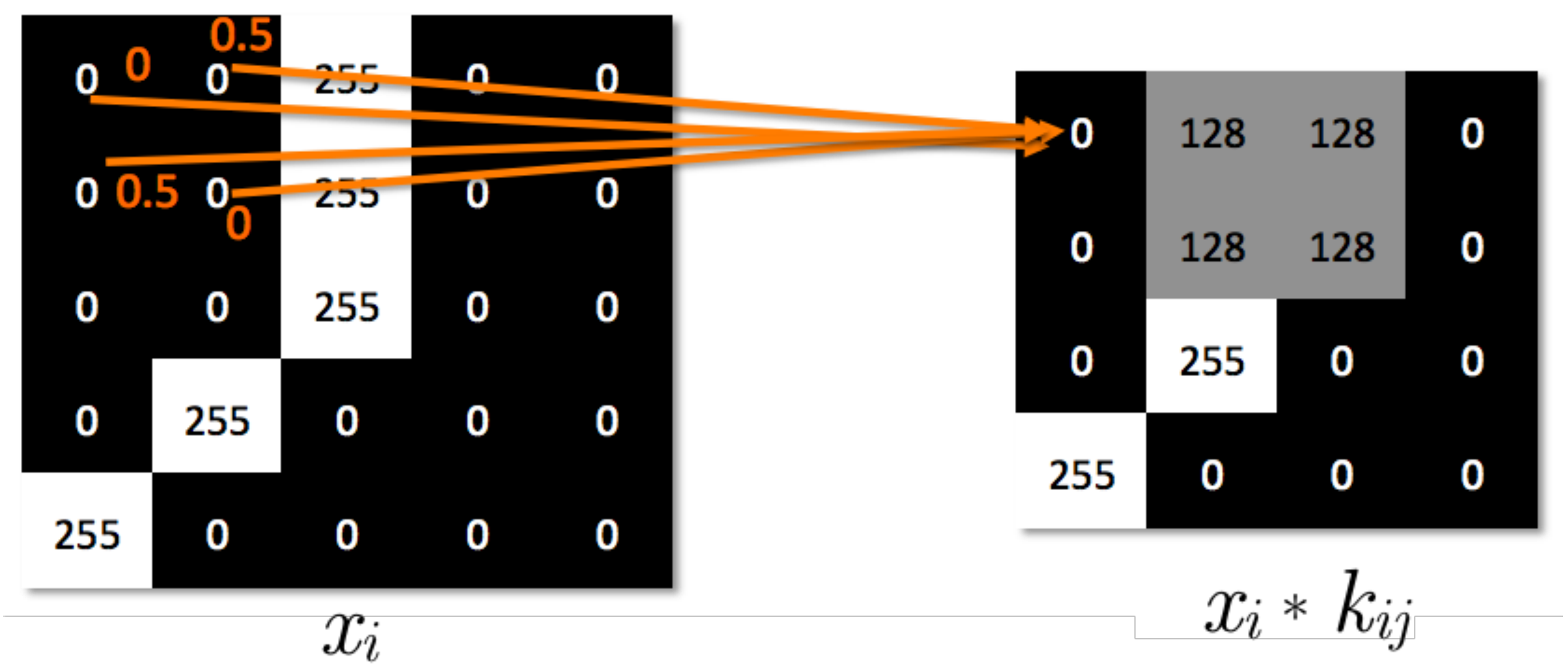
- Pre-activations from channel x_i into feature map y_j can be computed by:
  - getting the convolution kernel where $k_{ij} = \widetilde{W}_{ij}$ from the connection matrix W_{ij}
  - applying the convolution $x_i * k_{ij}$
- This is equivalent to computing the discrete correlation of x_i with W_{ij}

Example

- Illustration:

$$(x_i * k_{ij}) \text{ where } k_{ij} = \tilde{W}_{ij}$$





Example

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

$$(x_i * k_{ij}), \quad \text{where } W_{ij} = \tilde{W}_{ij}$$

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

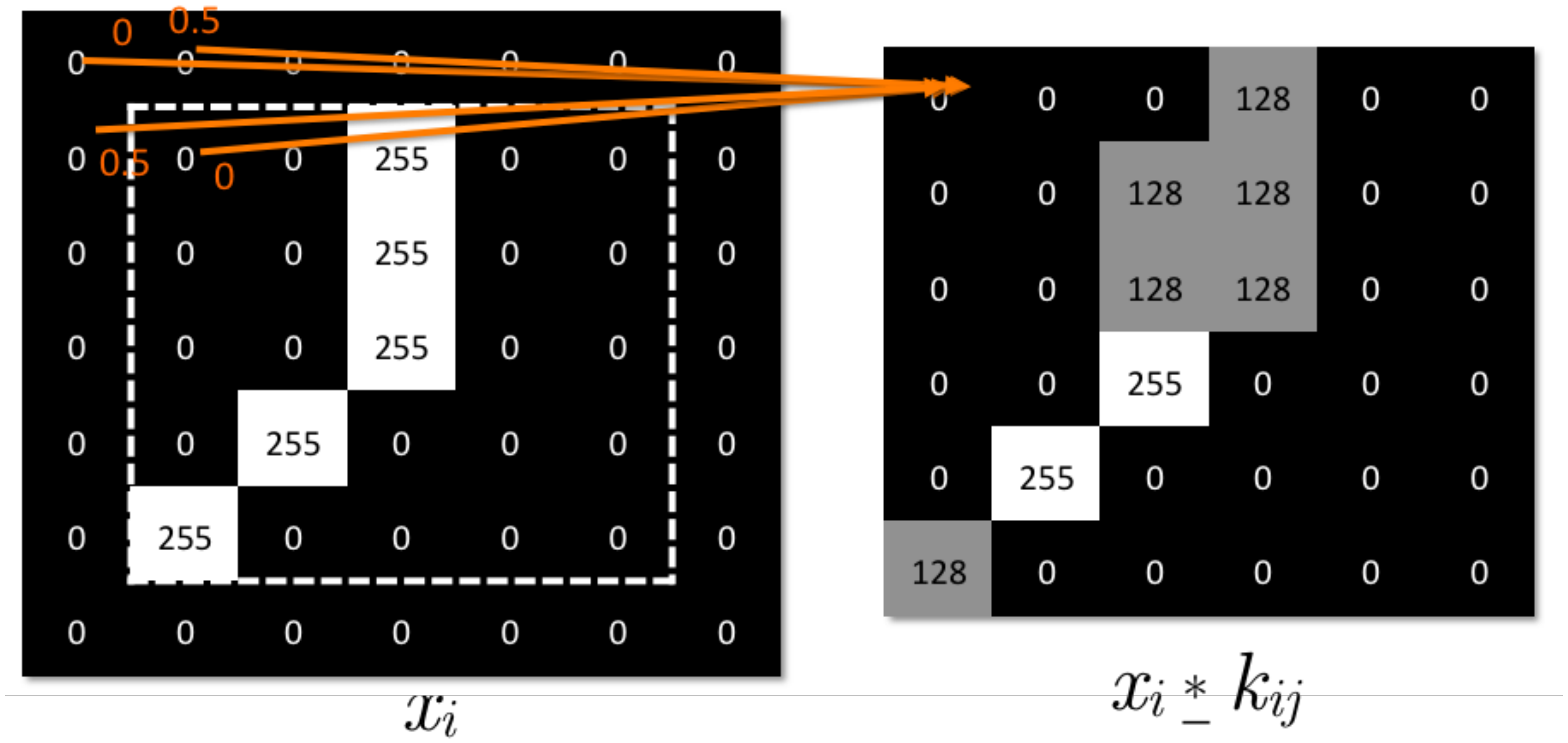
x_i

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

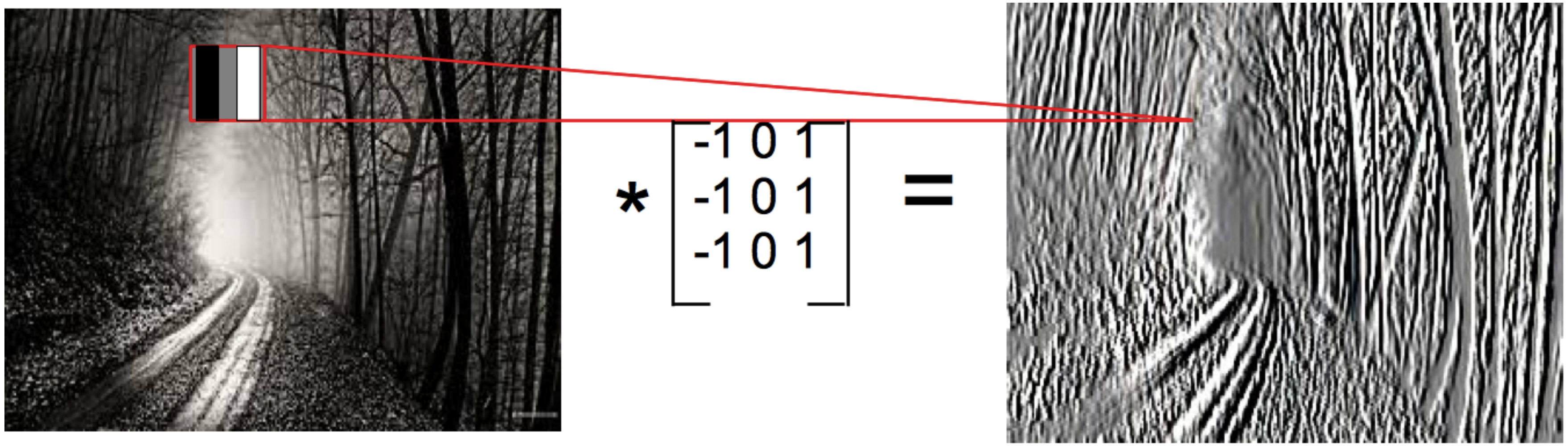
$\mathrm{sigm}(0.02\ x_i * k_{ij} \text{ -}4)$

Example

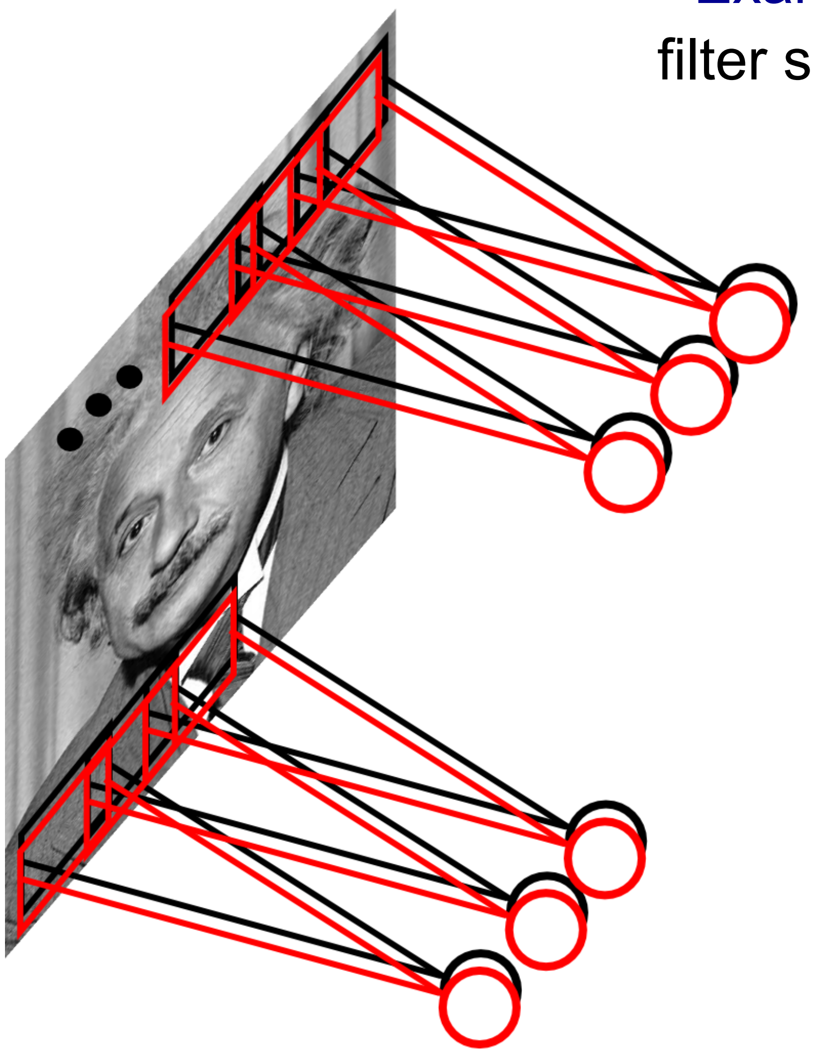
- Can use “zero padding” to allow going over the borders (*)



Example



Multiple Feature Maps

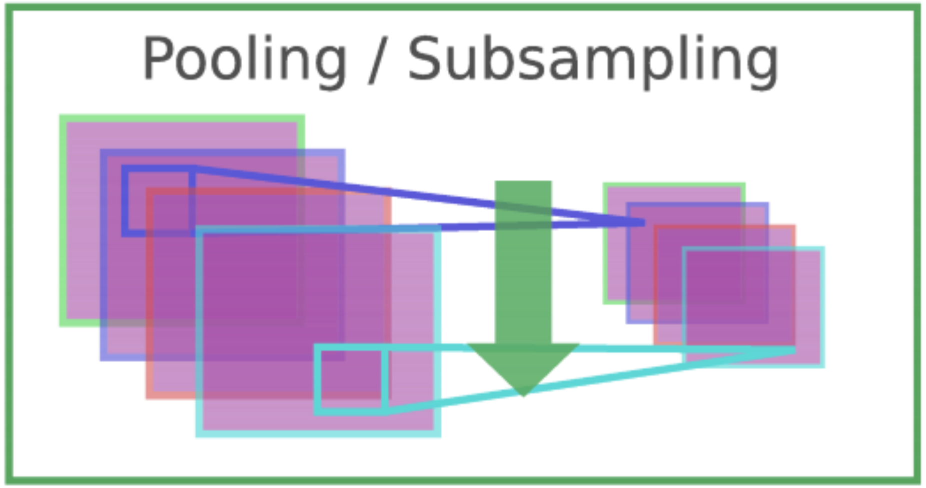


- Example: 200x200 image, 100 filters, filter size 10x10, 10K parameters

Pooling

- Pool hidden units in same neighborhood
  - Pooling is performed in non-overlapped neighborhoods (subsampling)

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



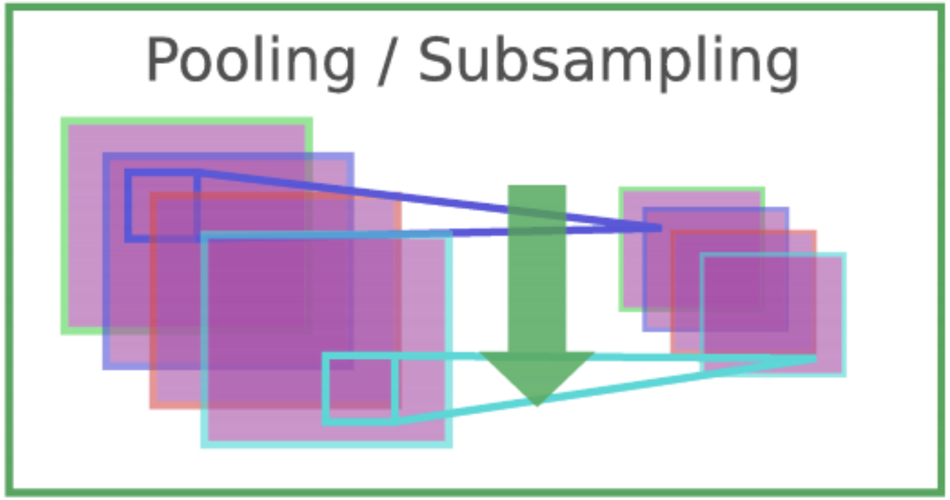
Jarret et al. 2009

- 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

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

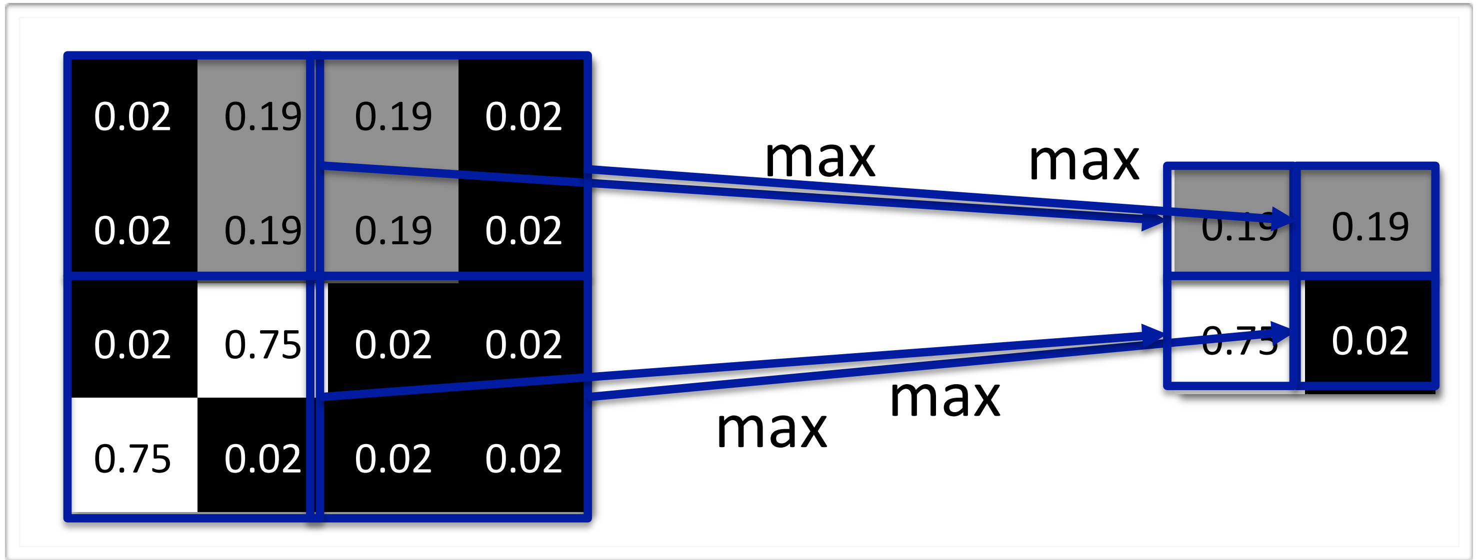


Jarret et al. 2009

- 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

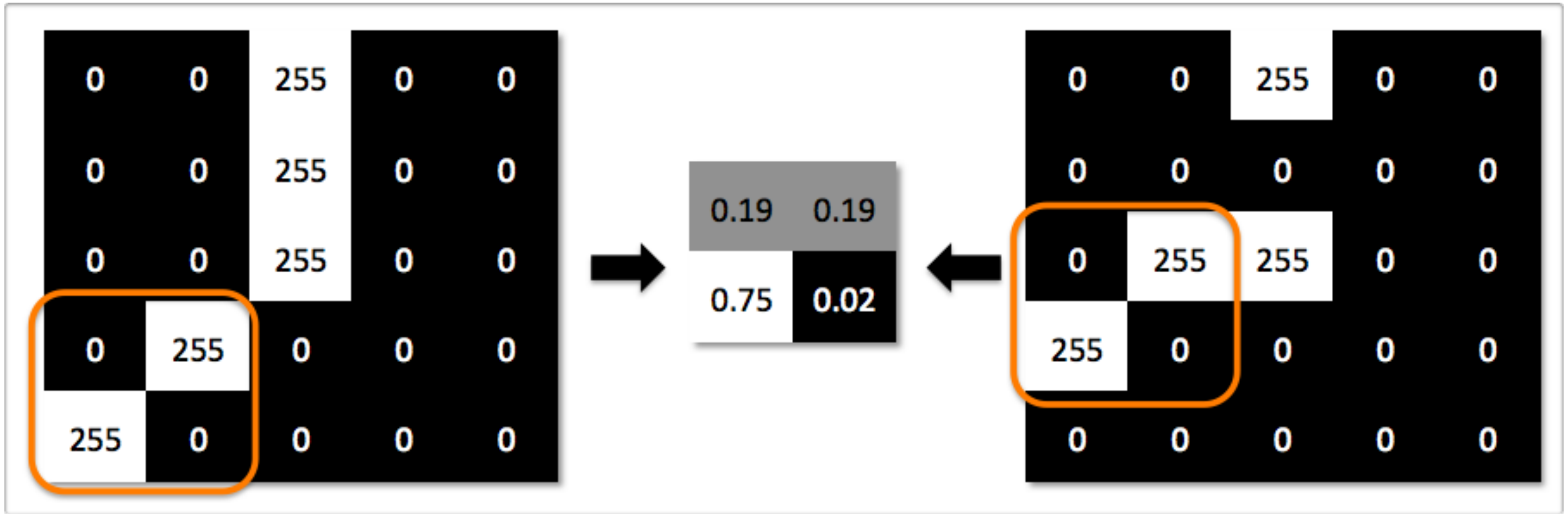
- Illustration of pooling/subsampling operation



- Why pooling?
    - ➢ Introduces invariance to local translations
    - ➢ Reduces the number of hidden units in hidden layer

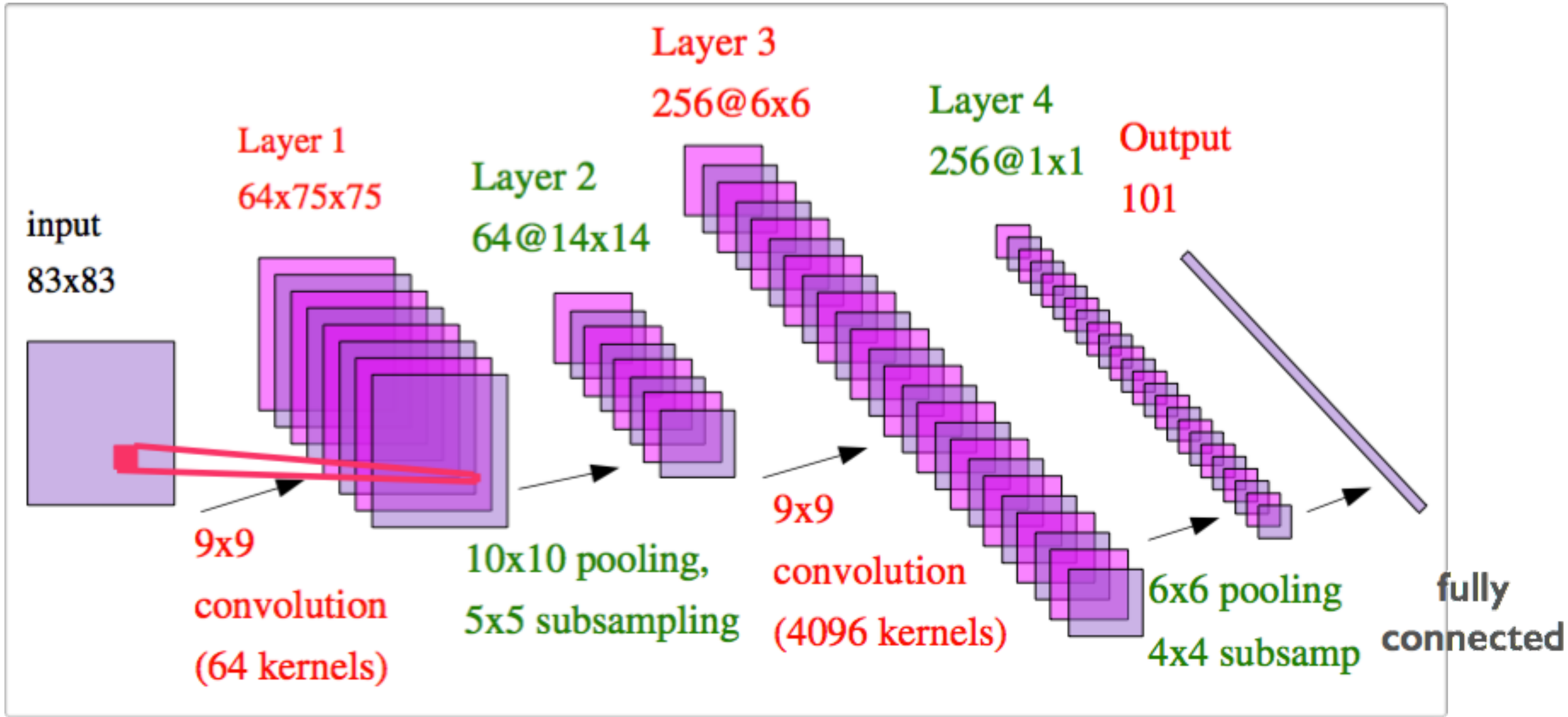
Translation Invariance

- Illustration of local translation invariance
  - ➢ both images result in the same feature map after pooling/ subsampling



Convolutional Neural Network

- Convolutional neural network alternates between the convolutional and pooling layers



From Yann LeCun's slides

Convolutional Neural Network

- For classification: Output layer is a regular, fully connected layer with softmax non-linearity
  - ➢ Output provides an estimate of the conditional probability of each class

- The network is trained by stochastic gradient descent
  - ➢ Backpropagation is used similarly as in a fully connected network
  - ➢ We have seen how to pass gradients through element-wise activation function
  - ➢ We also need to pass gradients through the convolution operation and the pooling operation

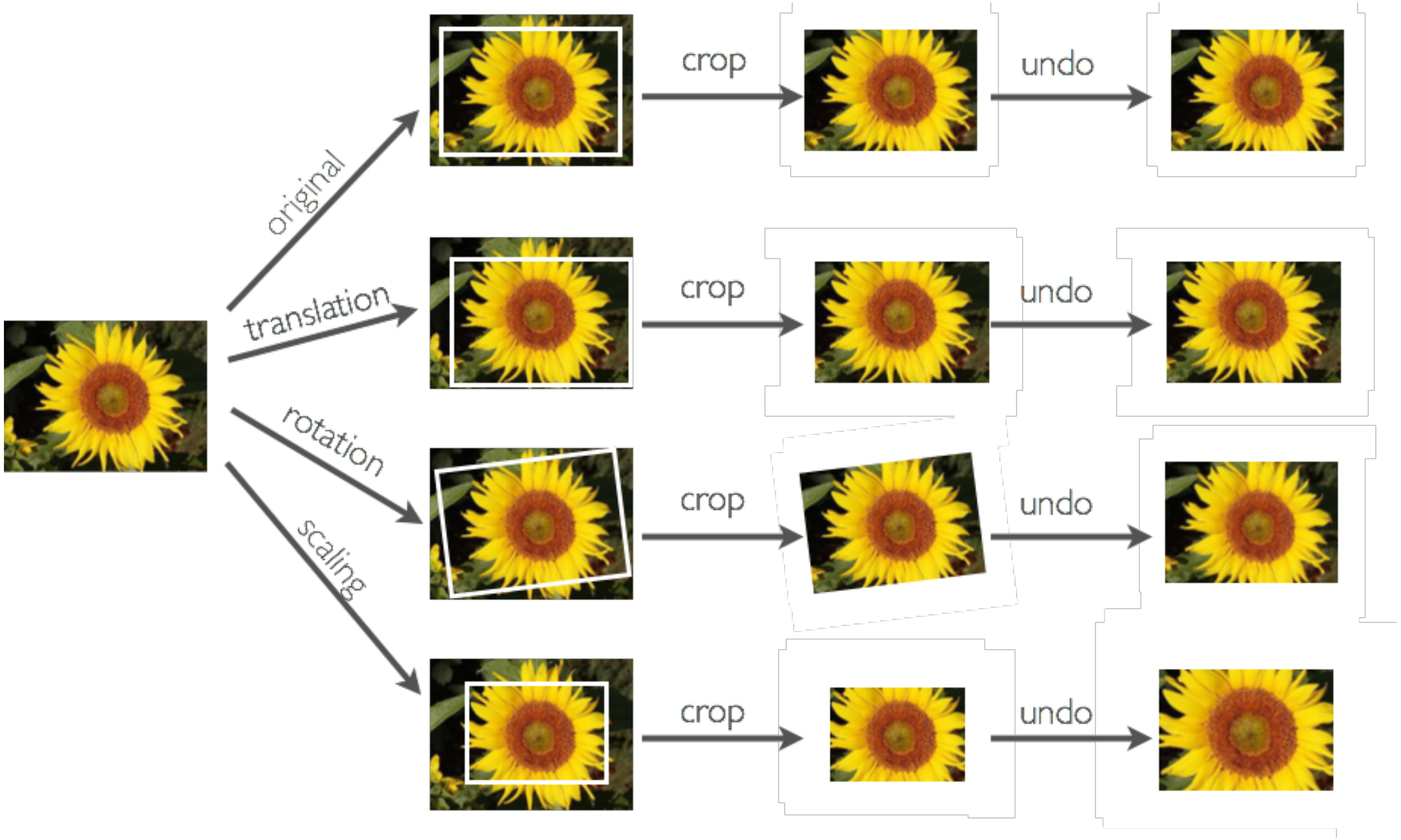
Invariance by Dataset Expansion

- Invariances built-in in convolutional network:
  - small translations: due to convolution and max pooling
  - small illumination changes: due to local contrast normalization

- It is not invariant to other important variations such as rotations and scale changes

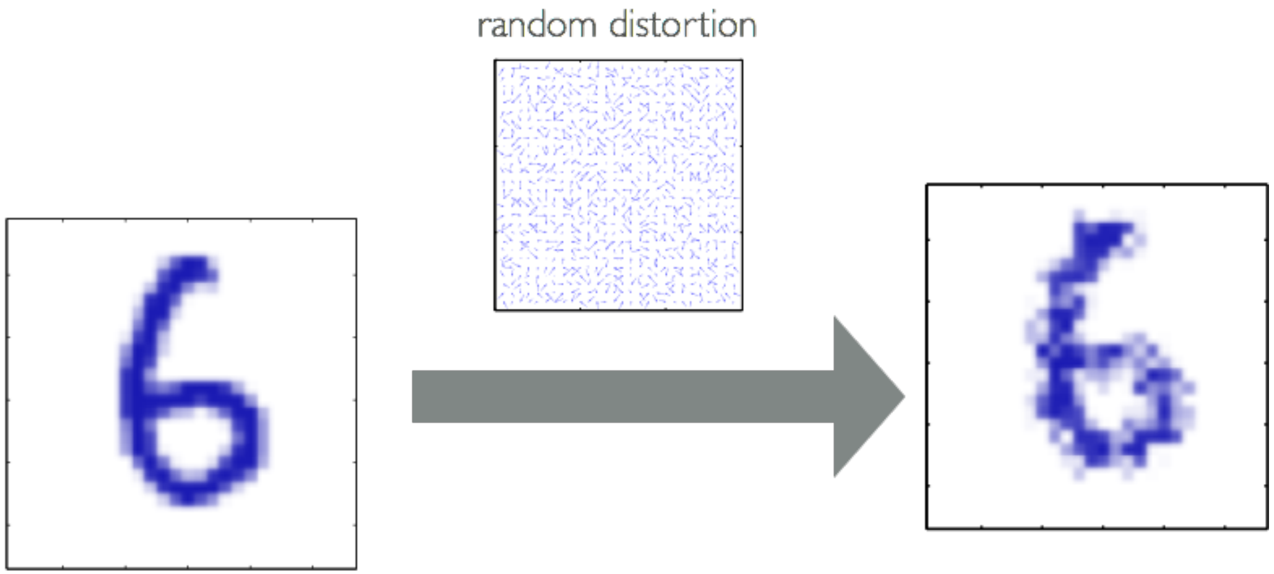
- However, it's easy to artificially generate data with such transformations
  - could use such data as additional training data
  - neural network can potentially learn to be invariant to such transformations

Generating Additional Examples



Elastic Distortions

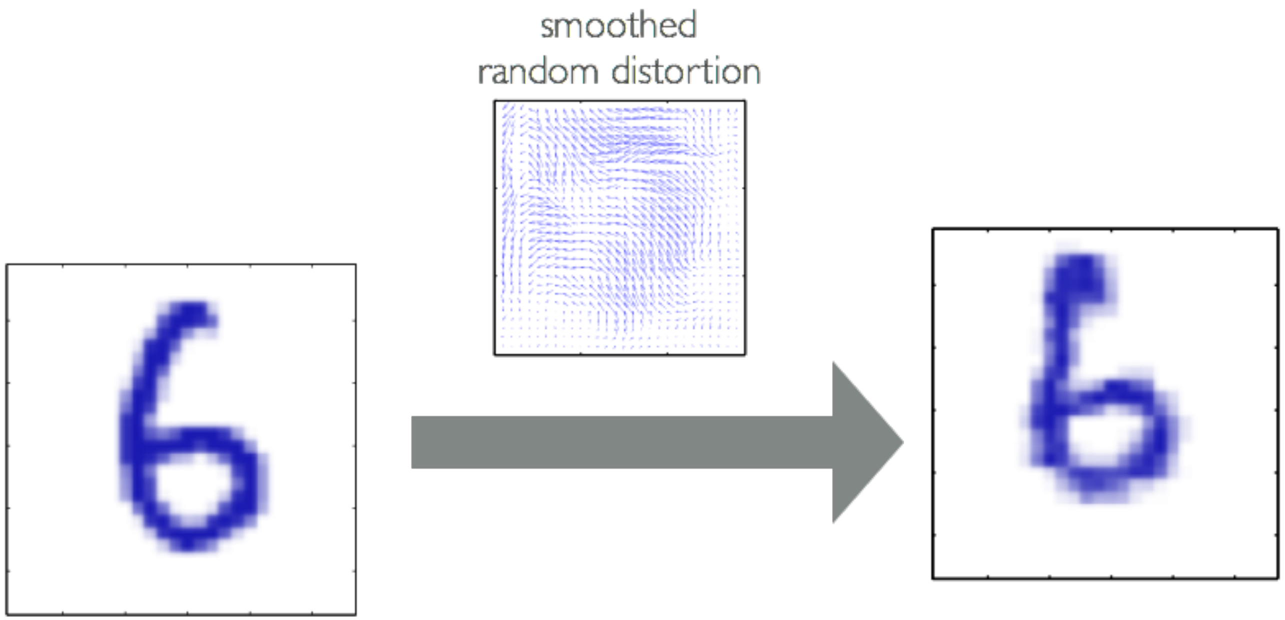
- Can add "elastic" deformations (useful in character recognition)
- We can do this by applying a "distortion field" to the image
  - ➢ a distortion field specifies where to displace each pixel value



Bishop's book

Elastic Distortions

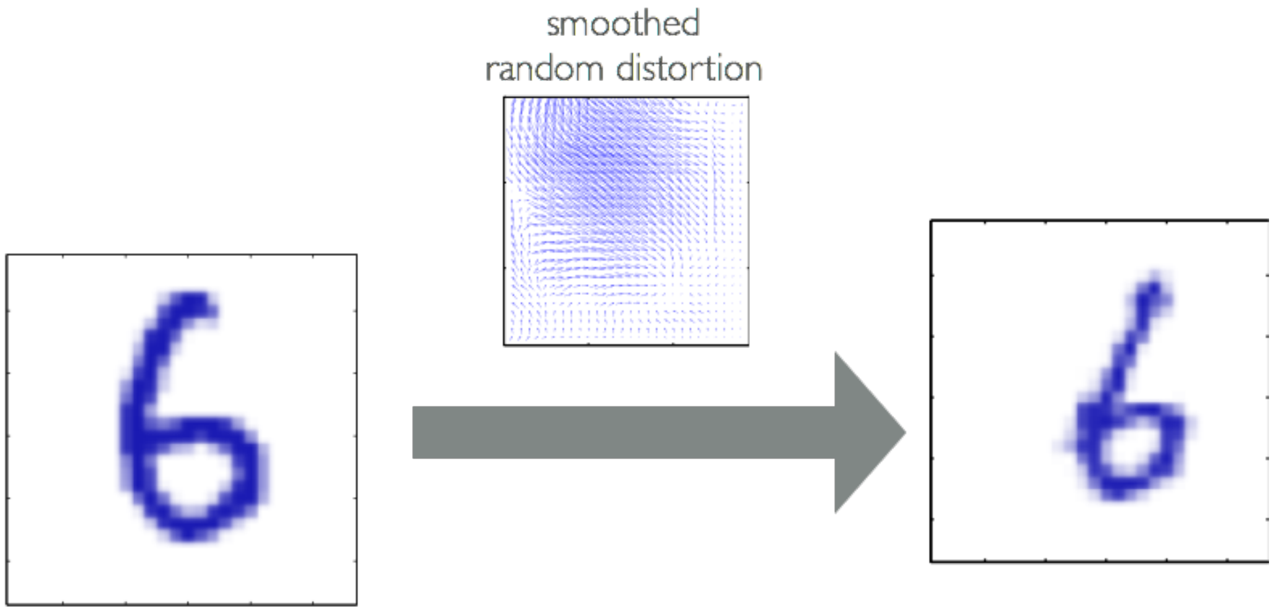
- Can add “elastic” deformations (useful in character recognition)
- We can do this by applying a “distortion field” to the image
  - ➢ a distortion field specifies where to displace each pixel value



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

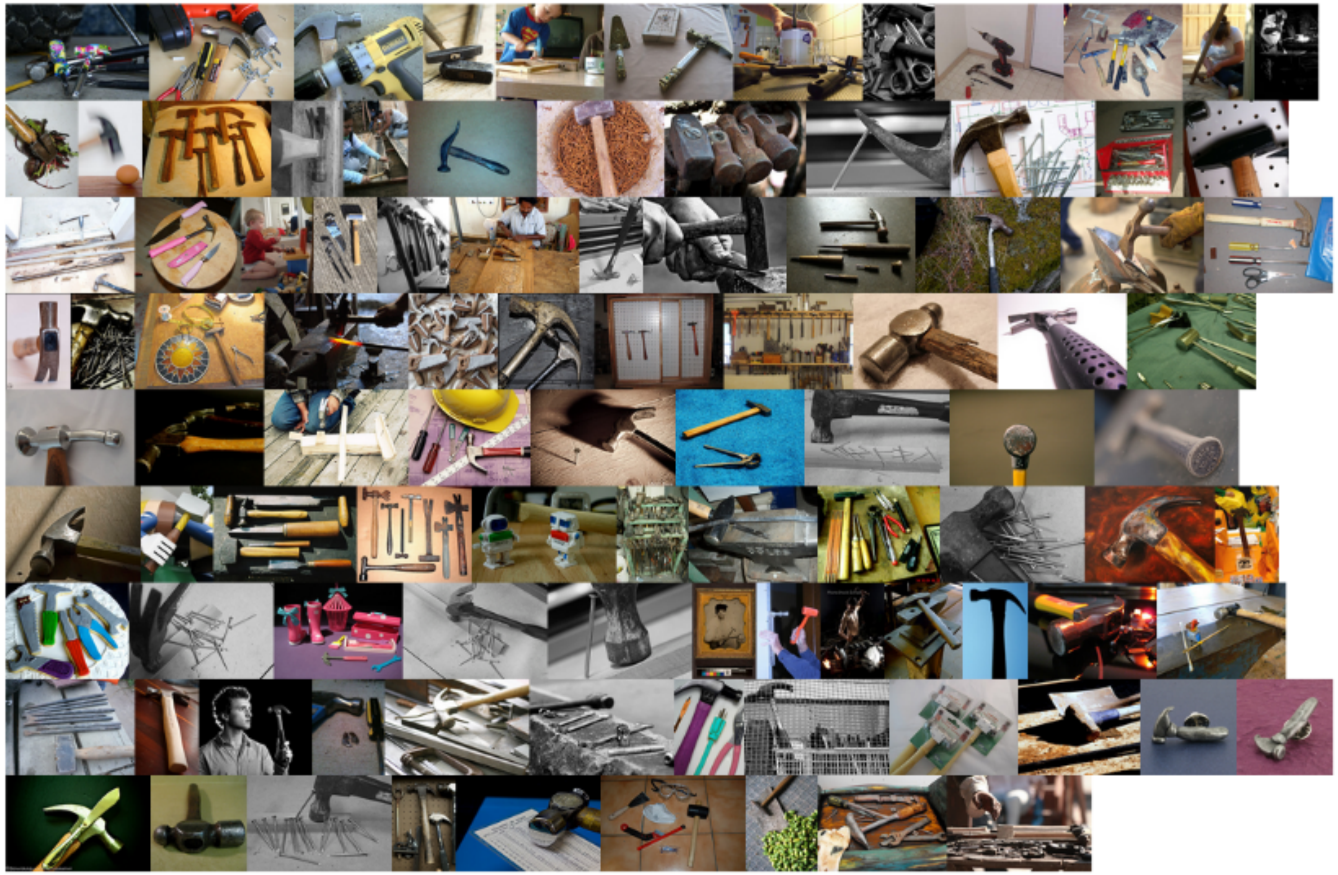


Bishop’s book

ImageNet Dataset

- 1.2 million images, 1000 classes

Examples of Hammer

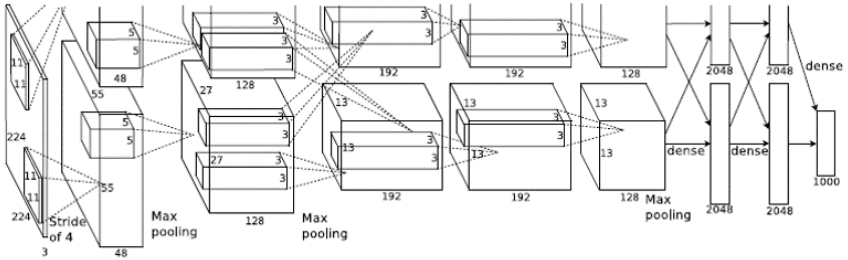


Deng et al. “Imagenet: a large scale hierarchical image database” CVPR 2009

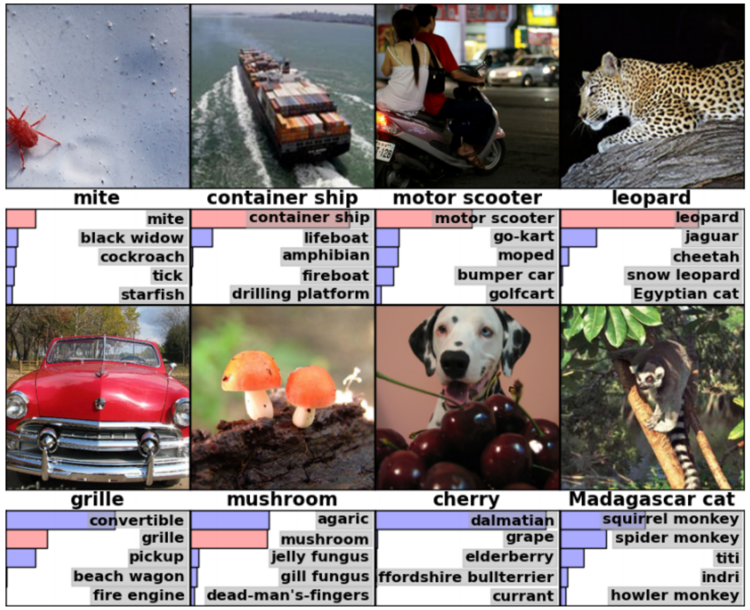
Important Breakthroughs

- Deep Convolutional Nets for Vision (Supervised)

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



1.2 million training images
1000 classes



How to Select the Right Architecture?

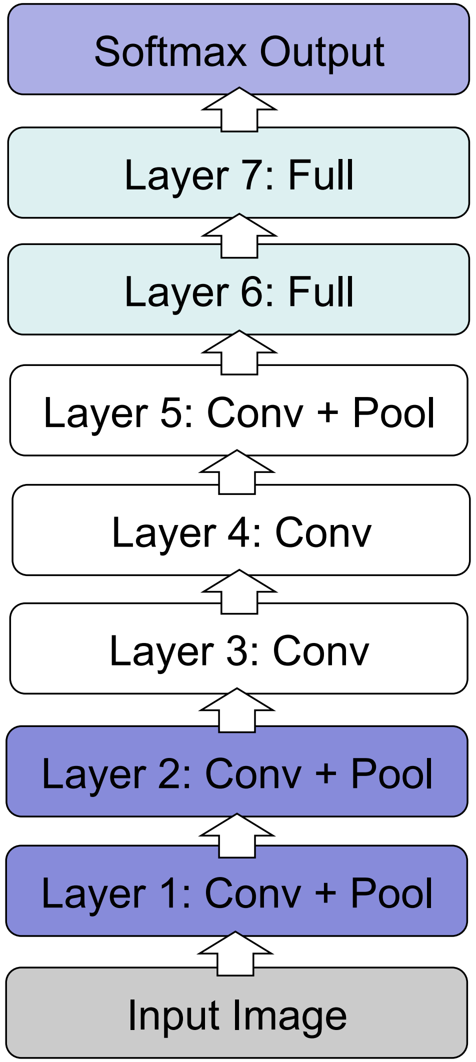
- From manual tuning features => manual tuning architectures
- Many hyper-parameters
  - Number of layers (depth), number of feature maps (width)
- Cross validation
- Grid search (need lots of GPUs)
- Smarter Strategies
  - Random search
  - Bayesian optimization
  - Reinforcement Learning (Zoph et al. 2016)

AlexNet

- 8 layers total

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

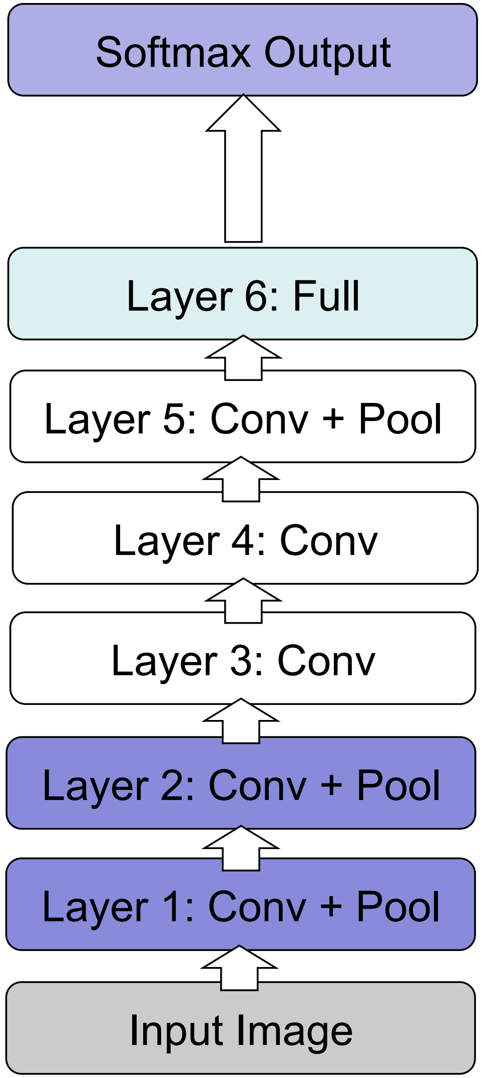
- 18.2% top-5 error



[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet

- Remove top fully connected layer 7
- Drop ~16 million parameters
- Only 1.1% drop in performance!



[From Rob Fergus’ CIFAR 2016 tutorial]

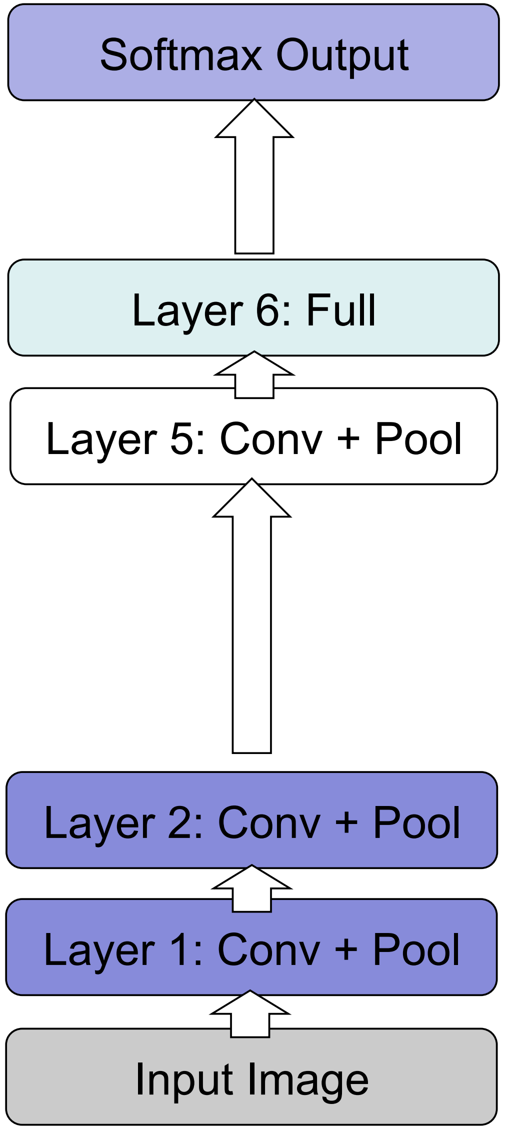
AlexNet

- Let us remove upper feature extractor layers and fully connected:
    - ➢ Layers 3,4, 6 and 7

- Drop ~50 million parameters

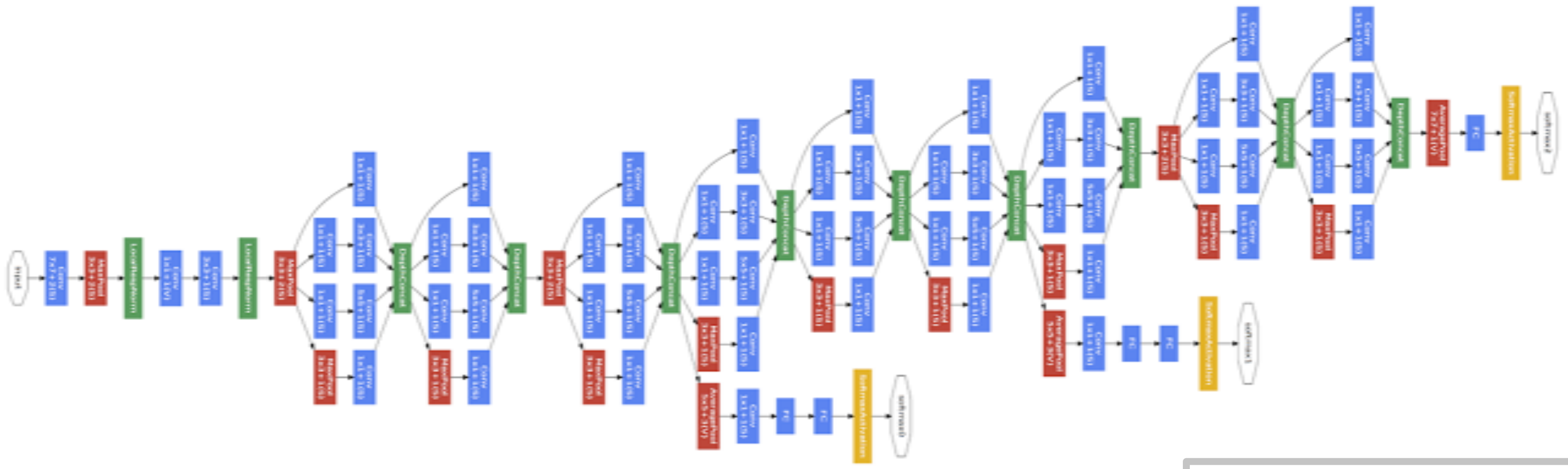
- **33.5 drop in performance!**

- Depth of the network is the key.



[From Rob Fergus' CIFAR 2016 tutorial]

GoogLeNet



Convolution
Pooling
Softmax
Other

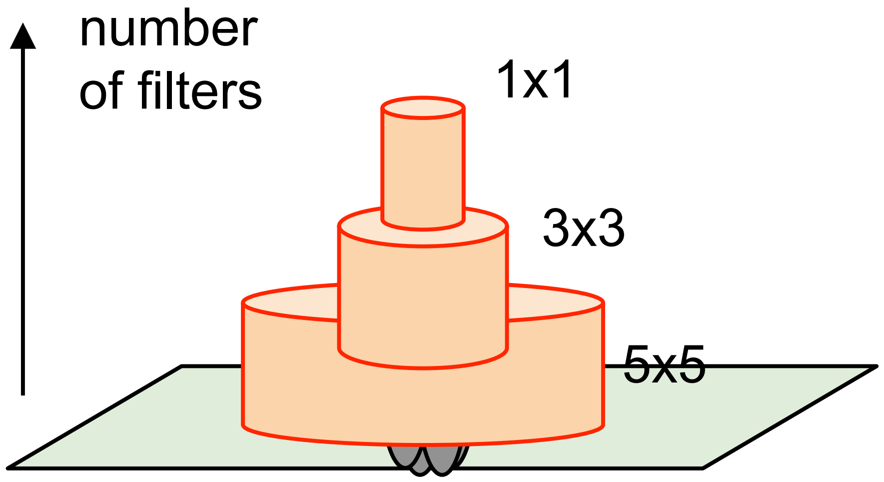
- 24 layer model that uses so-called inception module.

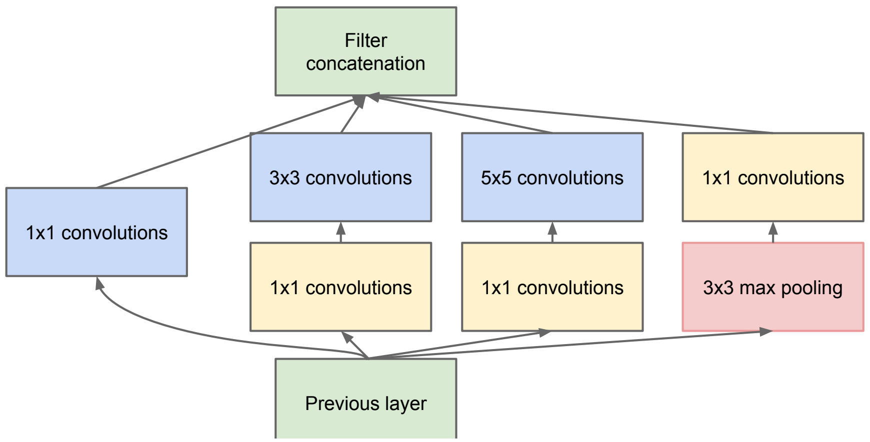
[Going Deep with Convolutions, Szegedy et al., arXiv:1409.4842, 2014]

GoogLeNet

- GoogLeNet inception module:

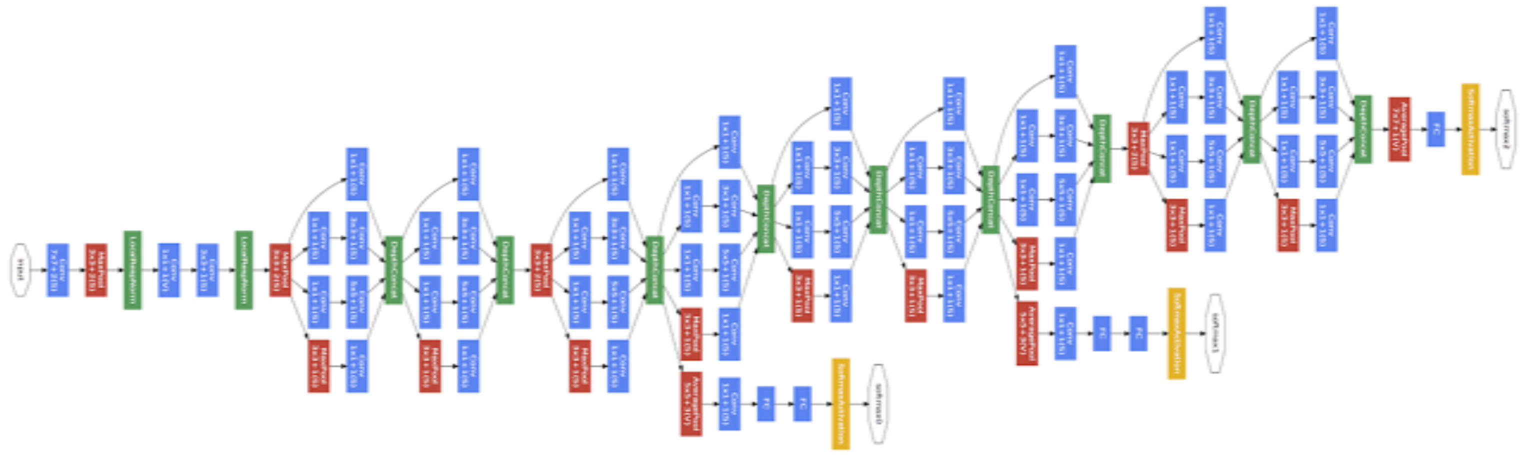
  - Multiple filter scales at each layer
  - Dimensionality reduction to keep computational requirements down





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

GoogLeNet

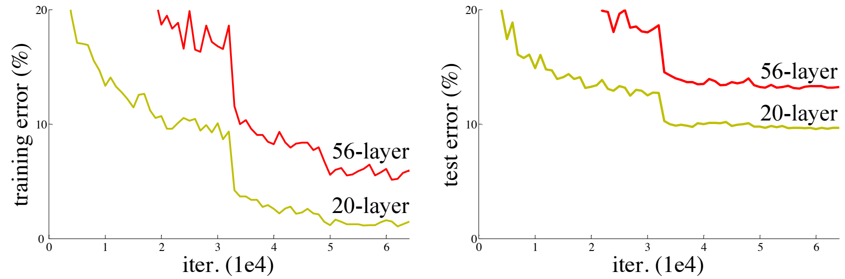


- Width of inception modules ranges from 256 filters (in early modules) to 1024 in top inception modules.
- Can remove fully connected layers on top completely
- Number of parameters is reduced to 5 million
- 6.7% top-5 validation error on Imagnet

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

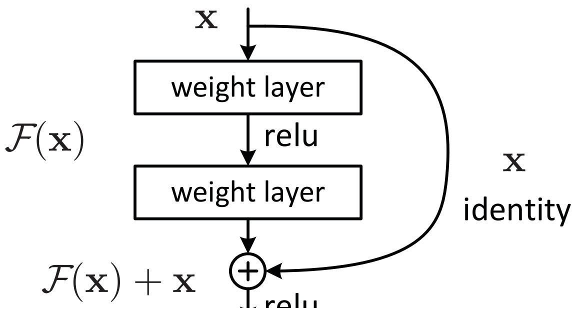
Residual Networks

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



Key idea: introduce "pass through" into each layer

Thus only residual now needs to be learned

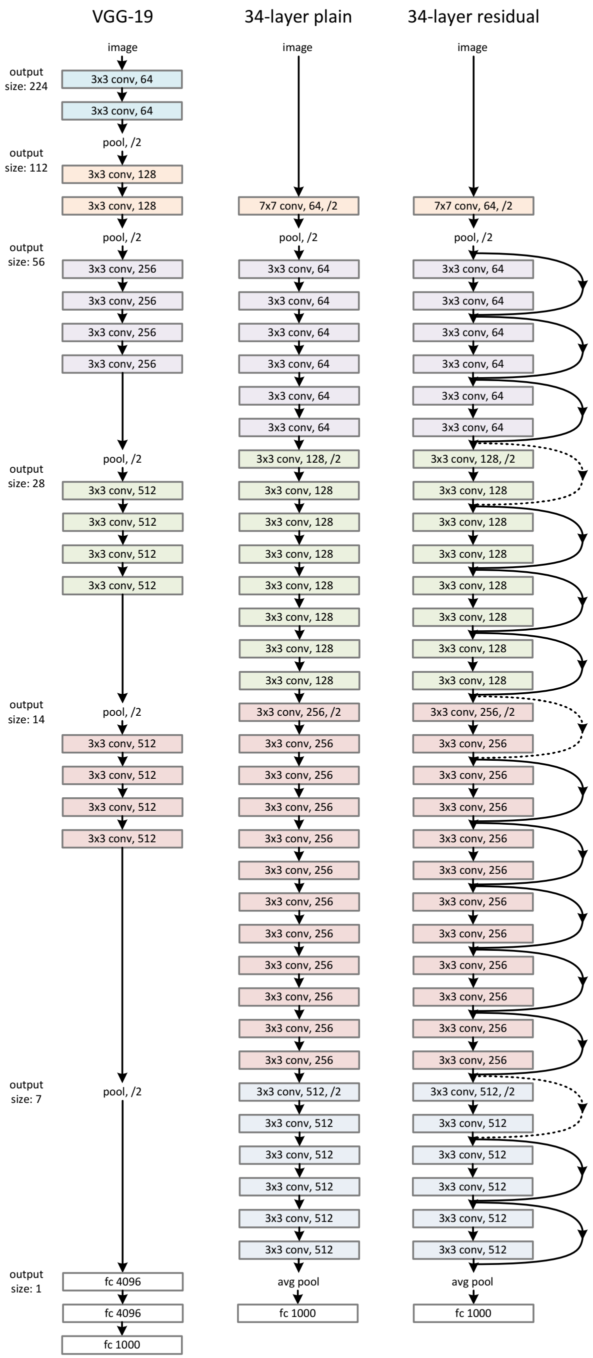


method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

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]



Selecting the Architecture

- Task dependent
- Cross-validation
- [Convolution $\rightarrow$ pooling]* + fully connected layer
- The more data: the more layers and the more kernels
  - Look at the number of parameters at each layer
  - Look at the number of flops at each layer
- Computational resources

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Optimization Tricks

- SGD with momentum, batch-normalization, and dropout usually works very well

- Pick learning rate by running on a subset of the data
  - Start with large learning rate & divide by 2 until loss does not diverge
  - Decay learning rate by a factor of ~100 or more by the end of training

- Use ReLU nonlinearity

- Initialize parameters so that each feature across layers has similar variance. Avoid units in saturation.

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Improve Generalization

- Weight sharing (greatly reduce the number of parameters)
- Data augmentation (e.g., jittering, noise injection, etc.)
- Dropout
- Weight decay (L2, L1)
- Sparsity in the hidden units
- Multi-task (unsupervised learning)

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

References

- Chapter 9, deep learning book