Convolutional Neural Networks and Applications to Object Classification

Vahid Tarokh

ECE685D, Fall 2025

Introduction

•We next focus on convolutional neural networks. These have been extremely successful in image classification algorithms.

•Important Note: Source of some of my slides (with great appreciation and acknowledgements):

- **Dive into Deep Learning**
- **Professor David Carlson's Slides**
- **Professor Hugo Larochelle's slides**
- **Professor Ruslan Salakhutdinov's slides (available online)**

•Some tutorial slides were borrowed from Rob Fergus
https://sites.google.com/site/deeplearningsummerschool2016/speakers

•Marc'Aurelio Ranzato's CVPR 2014 tutorial on Convolutional Nets
https://sites.google.com/site/lsvrtutorialcvpr14/home/deeplearning

- **Much of the material in this lecture was borrowed from class on NN:**
  https://sites.google.com/site/deeplearningsummerschool2016

Computer Vision

• Design algorithms that can process visual data to accomplish a given task:

- For example, object recognition: Given an input image, identify which object it contains

Image Classification

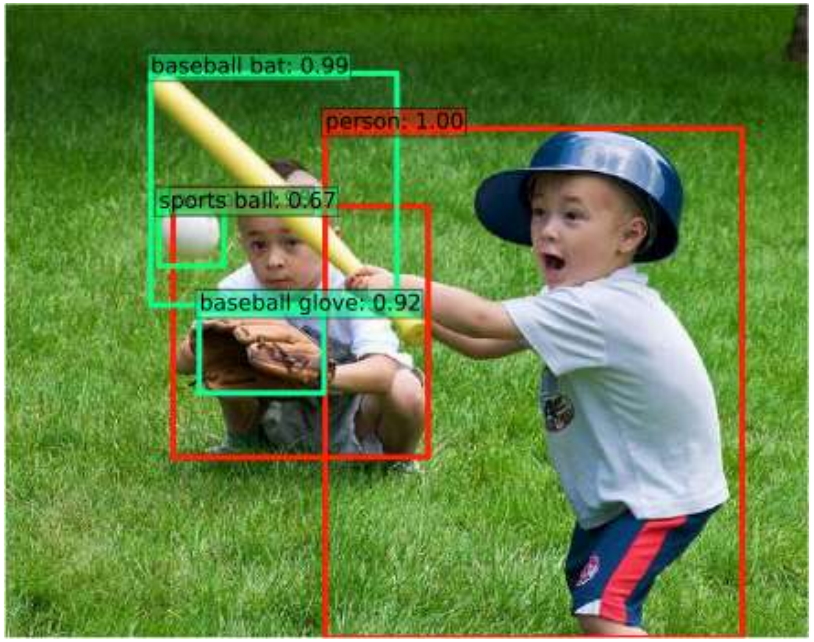
Question: “What is this an image of?”

Answer: “95% probability this is a ballplayer”

Object Detection

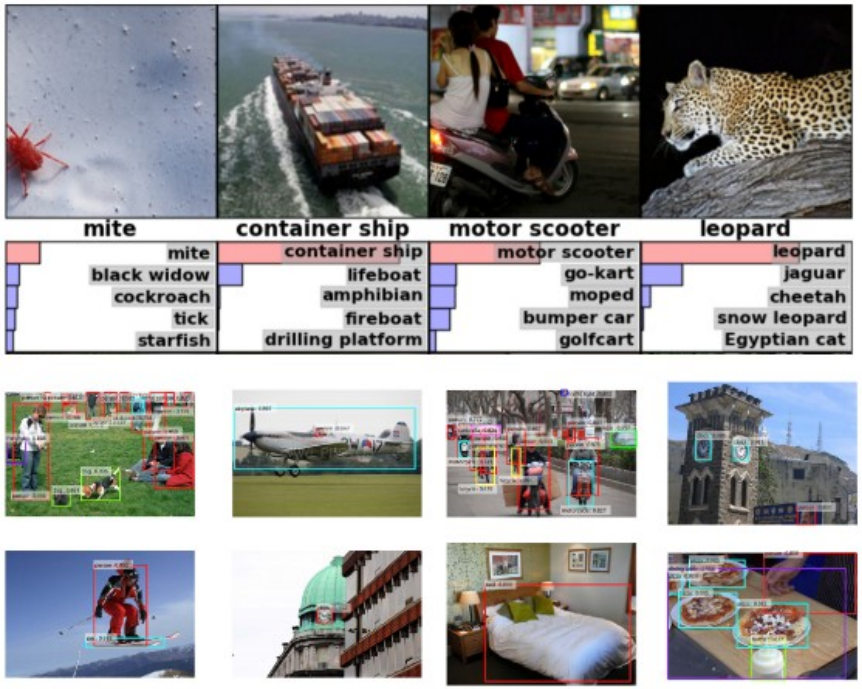
Question: “What are all the objects in this object and where are they?”

Answer:



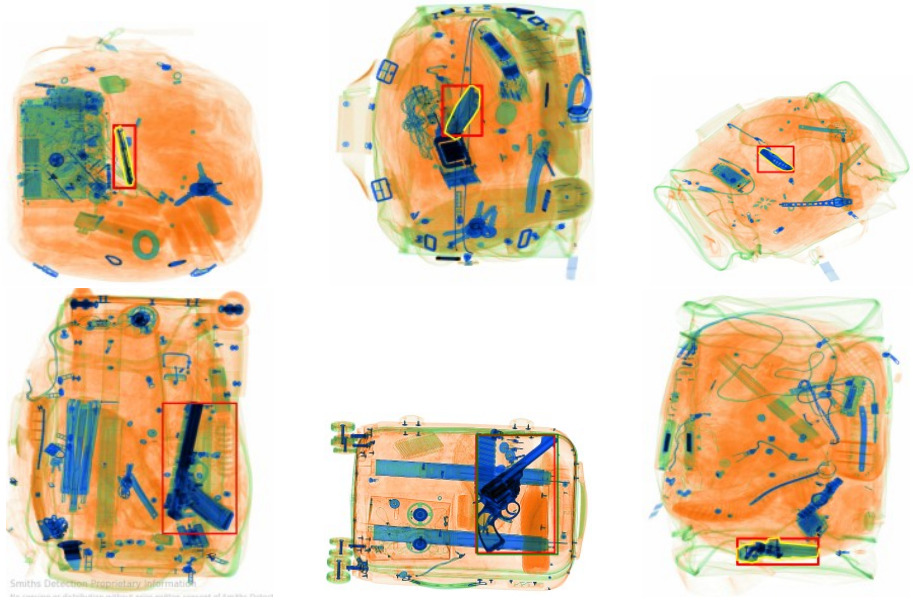
Classification versus detection

- Very related, but different problems.
- Many aspects will be shared between the two problems
- In detection algorithms, we try to draw a bounding box around the object of interest to locate it within the image.
- There could be many bounding boxes representing different objects of interest within the image and you would not know how many beforehand.



Application: Self-Driving Cars

Application: TSA



Useful Properties

We have some important properties that are useful

- Translation Invariance
- Scale Invariance
- Rotation Invariance

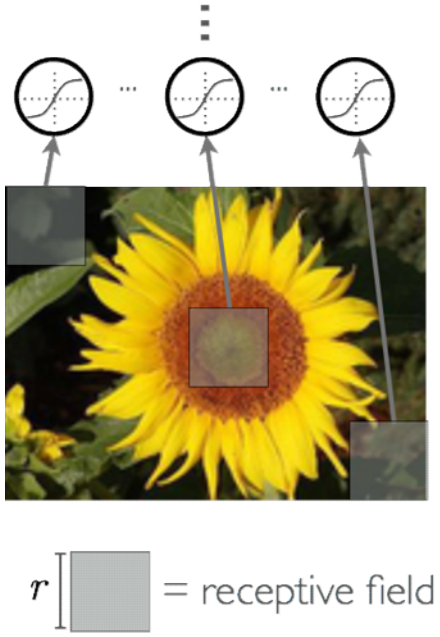
Some are built into the algorithm, and some come from the structure of the dataset

Convolutional Neural Networks

- 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 on invariance to certain variations: translation, illumination, etc.
- Convolutional networks leverage these ideas
  - Local connectivity
  - Parameter sharing
  - Convolution
  - Pooling / subsampling hidden units

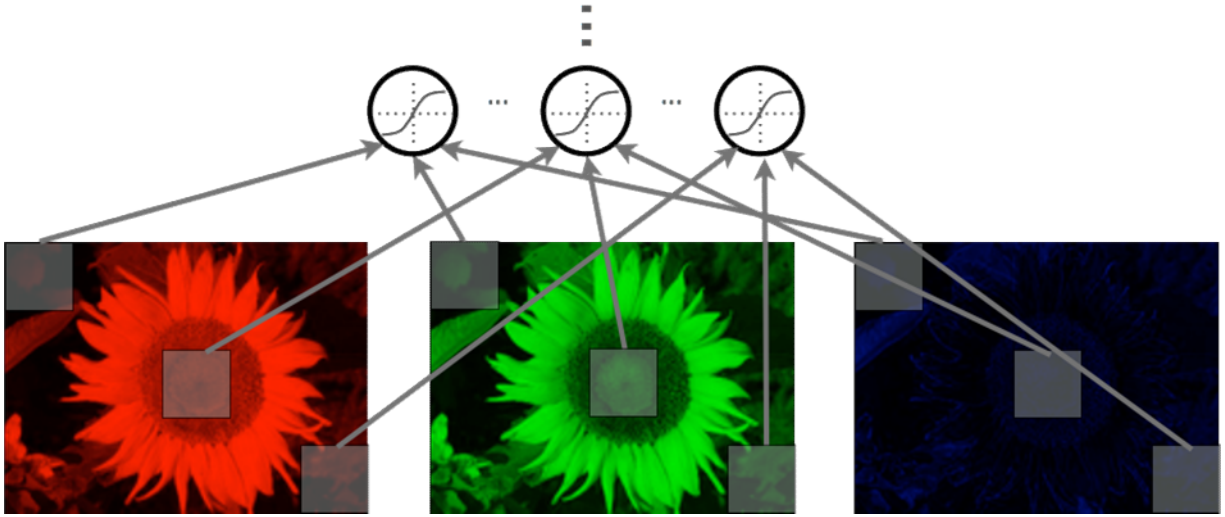
Local Connectivity

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



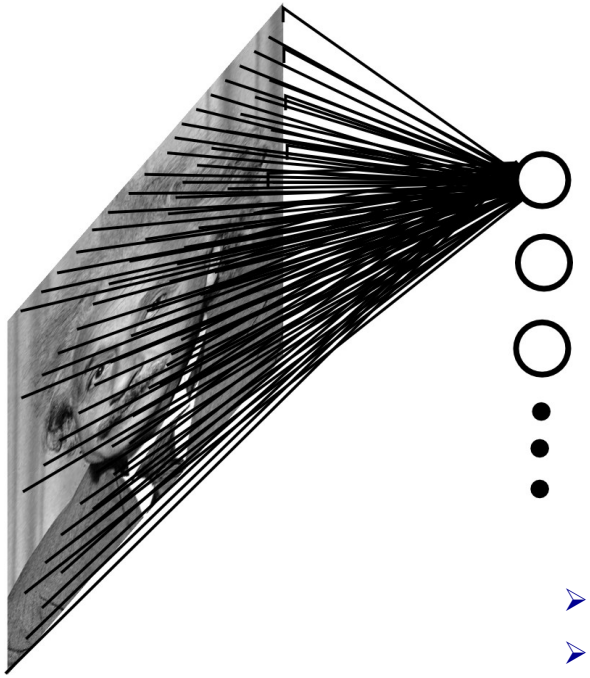
Local Connectivity

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



Local Connectivity

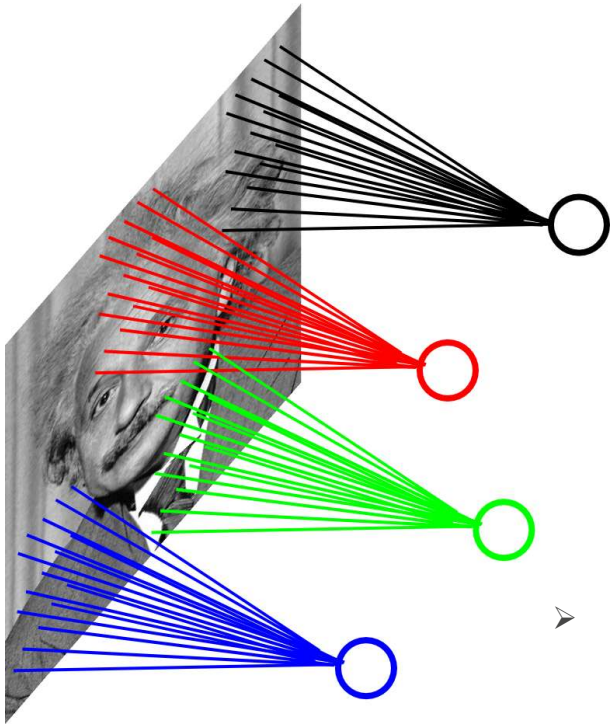
- Example: 200x200 image, 40K hidden units, ~2B parameters!



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

Local Connectivity

• Example: 200x200 image, 40K hidden units, filter size 10x10, 4M parameters!



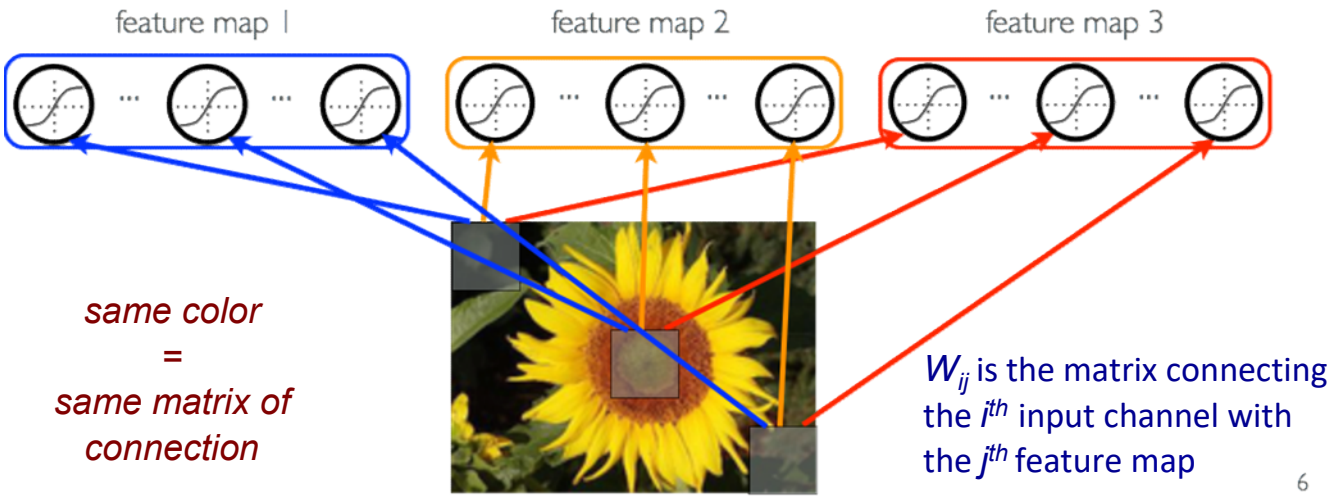
- This parameterization is good when input image is registered

Convolutional Neural Networks

- 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

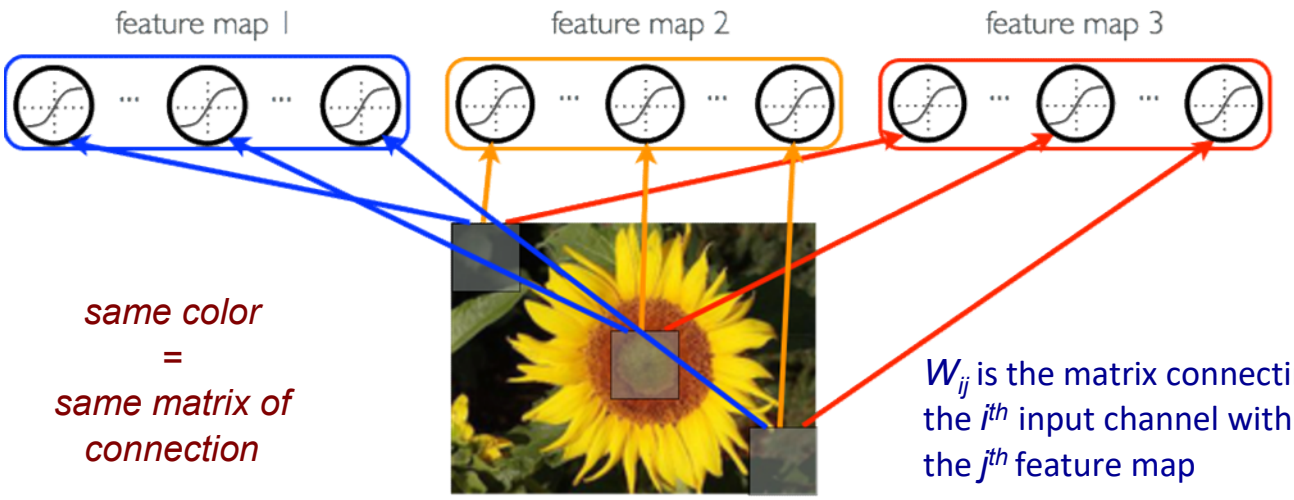
Parameter Sharing

- 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



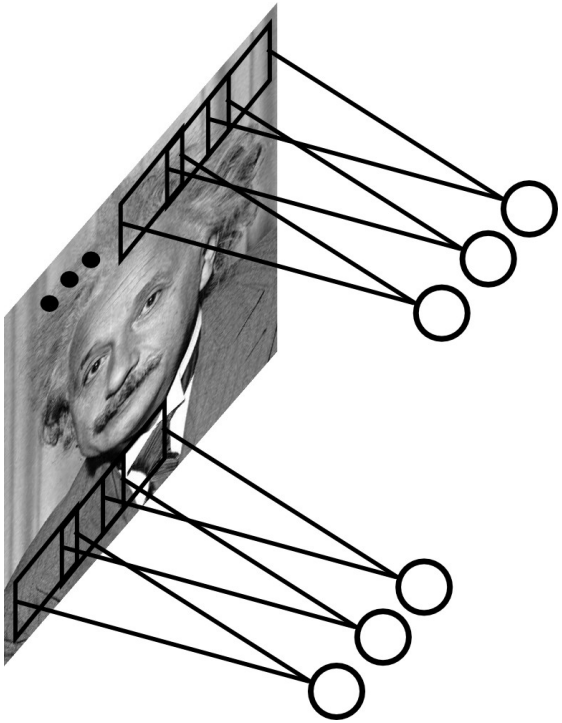
Parameter Sharing

- Why parameter sharing?
  - Reduces even more the number of parameters
  - Will extract the same features at every position (features are "equi-variant")



Parameter Sharing

- Share matrix of parameters across certain units



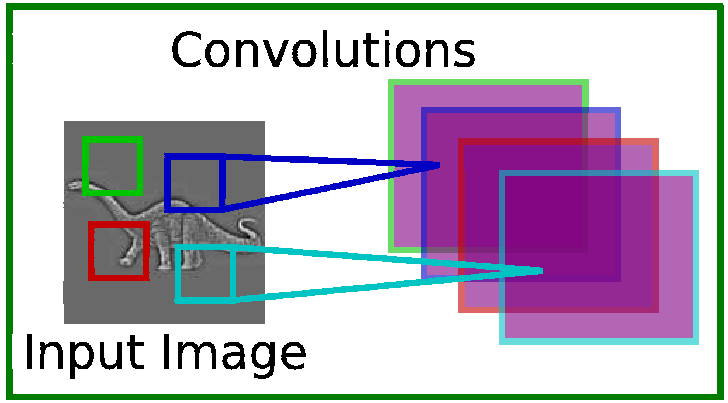
- Convolutions with certain kernels

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 / sub-sampling hidden units

Parameter Sharing

- Each feature map forms a 2D grid of features
  - can be computed with a discrete convolution ($*$) of a kernel matrix k_{ij} which is the hidden weights matrix W_{ij} with its rows and columns flipped



Jarret et al. 2009

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

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

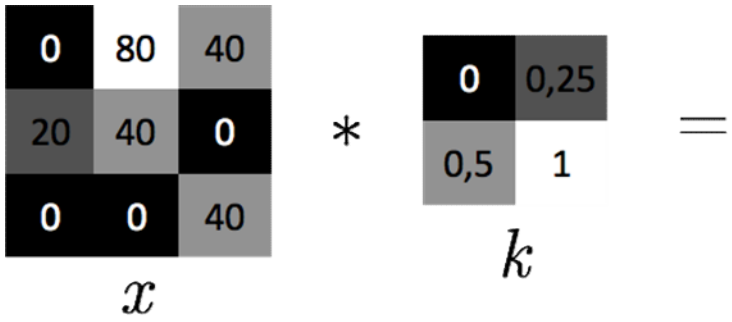
can add bias

Discrete Convolution

- Discrete convolution between **one** kernel (filter) and **one** channel image (matrix; 2-d tensor)
- This is not quite convolution; instead, it is correlation

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

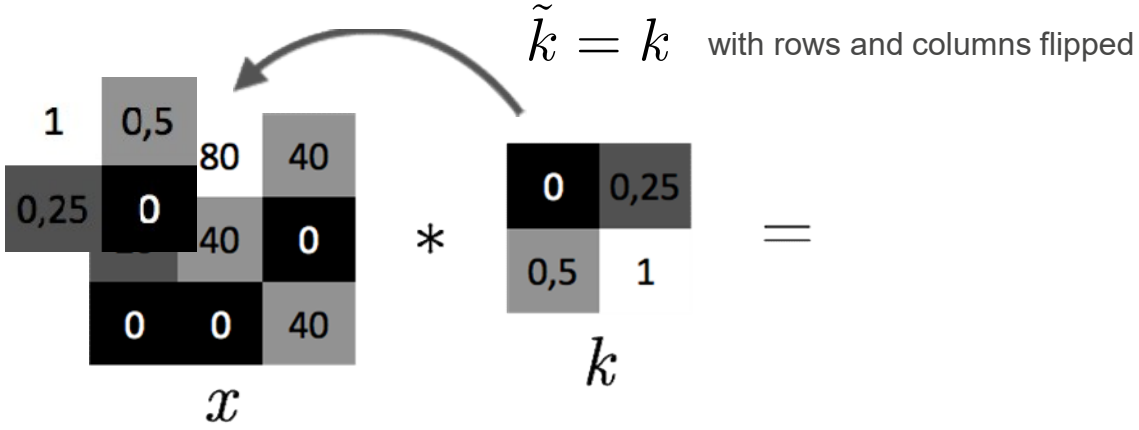
- Example:



Discrete Convolution

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

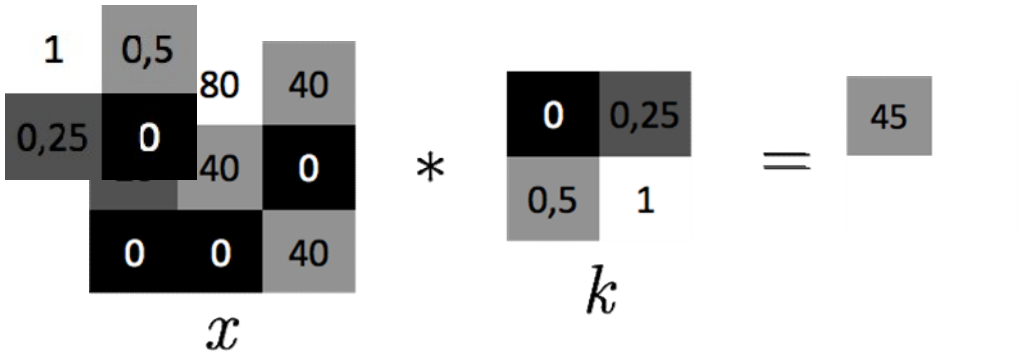
- Example:



Discrete Convolution

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

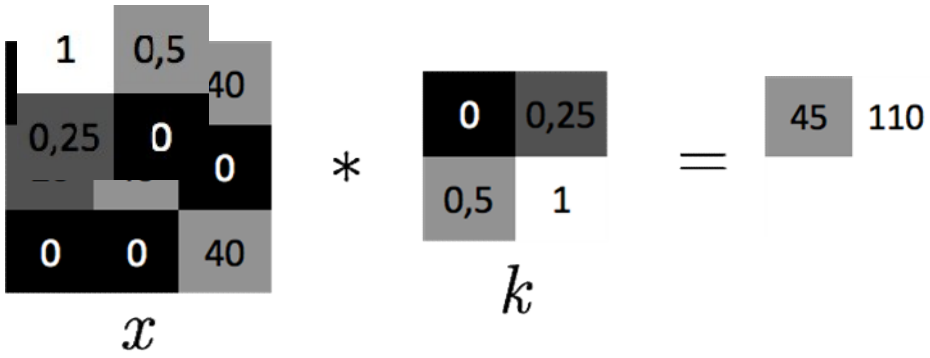
- Example: 1 x 0 + 0.5 x 80 + 0.25 x 20 + 0 x 40 = 45



Discrete Convolution

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

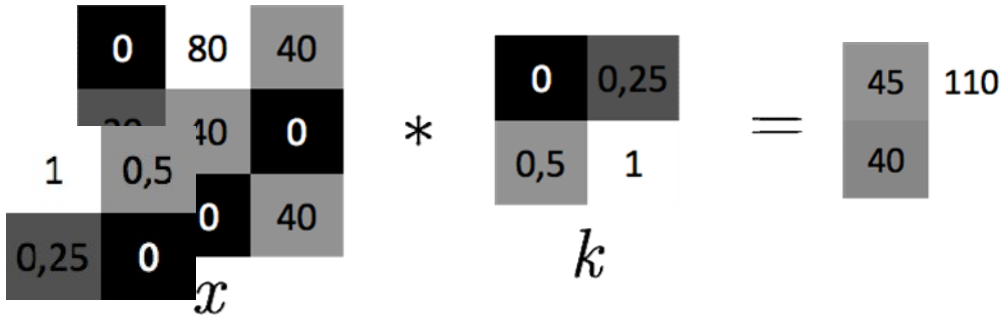
- Example: 1 x 80 + 0.5 x 40 + 0.25 x 40 + 0 x 0 = 110



Discrete Convolution

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

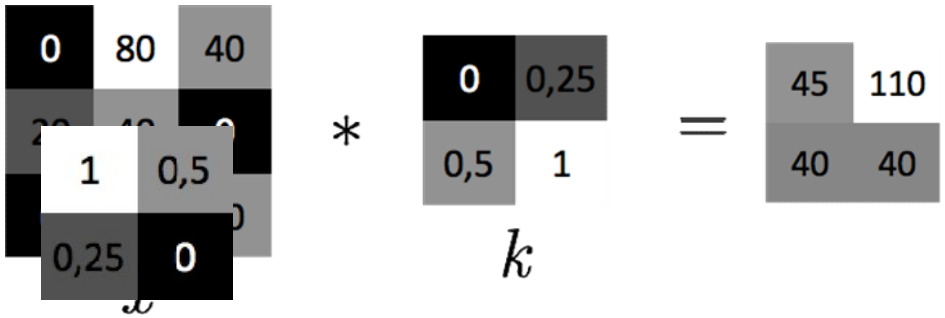
- Example: 1 x 20 + 0.5 x 40 + 0.25 x 0 + 0 x 0 = 40



Discrete Convolution

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

- 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 $k_{ij} = W_{ij}$ from the connection matrix W_{ij}
- applying the correlation $x_i * k_{ij}$
- **We abuse the terminology and notation and refer and use the same notation for convolution and correlation when there is no ambiguity.**

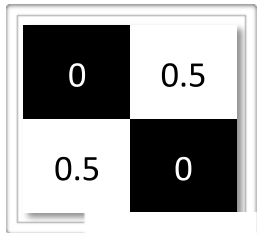
  - This is equivalent to computing the discrete correlation of x_i with W_{ij}
  - Discrete convolution in general form (for f^{th} output filter (kernel), for c^{th} input channel)
  - k is a 4-d Tensor which is convolved to the 3-d Tensor input x

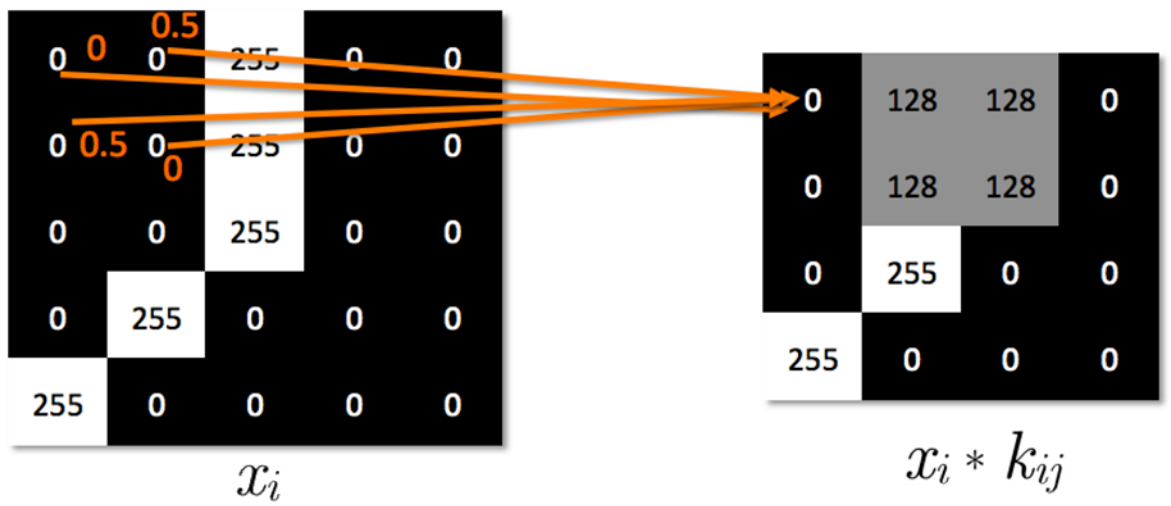
$$(x * k)_{fij} = \sum_{c} \sum_{p,q} x_{c,i+p,j+q} . k_{c,p,q,f}$$

Example

- Illustration:

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

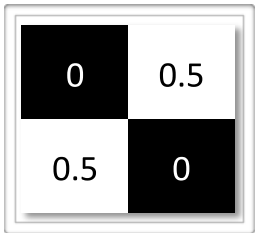


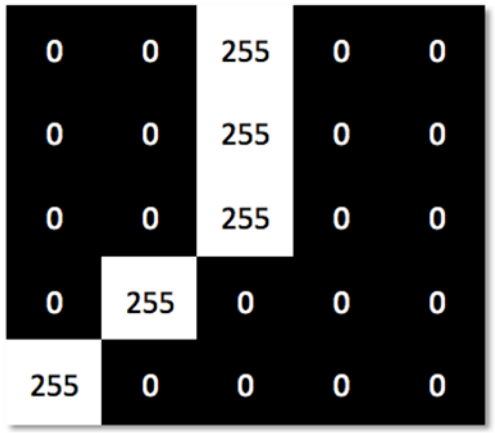


Example

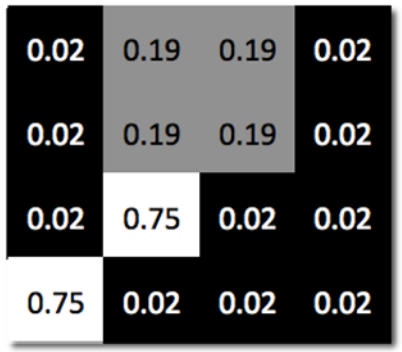
• With a non-linearity, we get a detector of a feature at any position in the image:

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





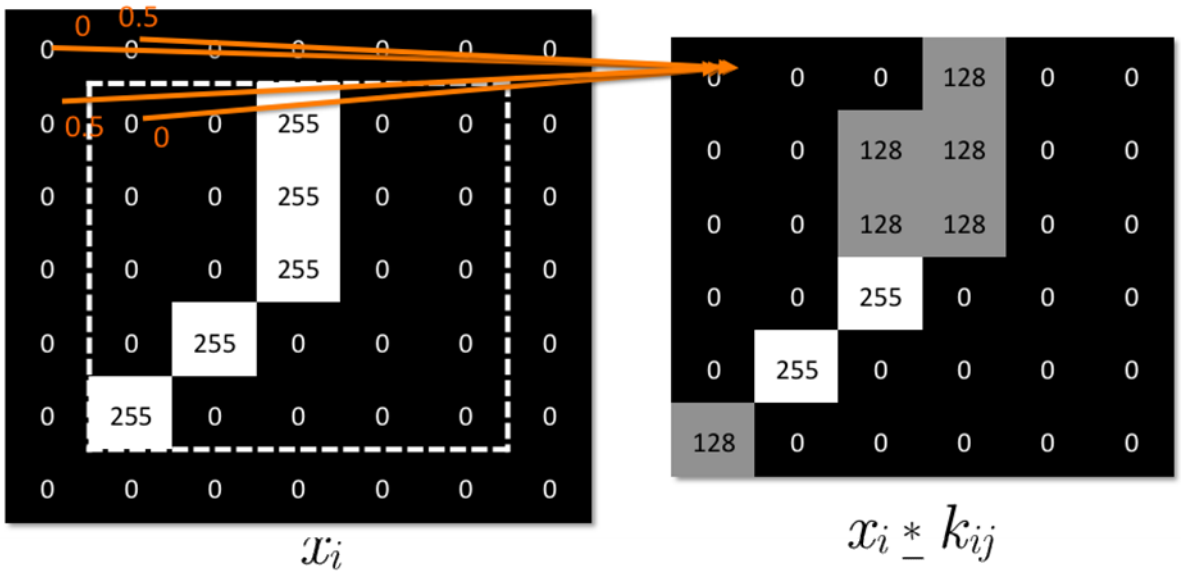
x_i



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

Example

- Can use ‘‘zero padding’’ to allow going over the borders (*)

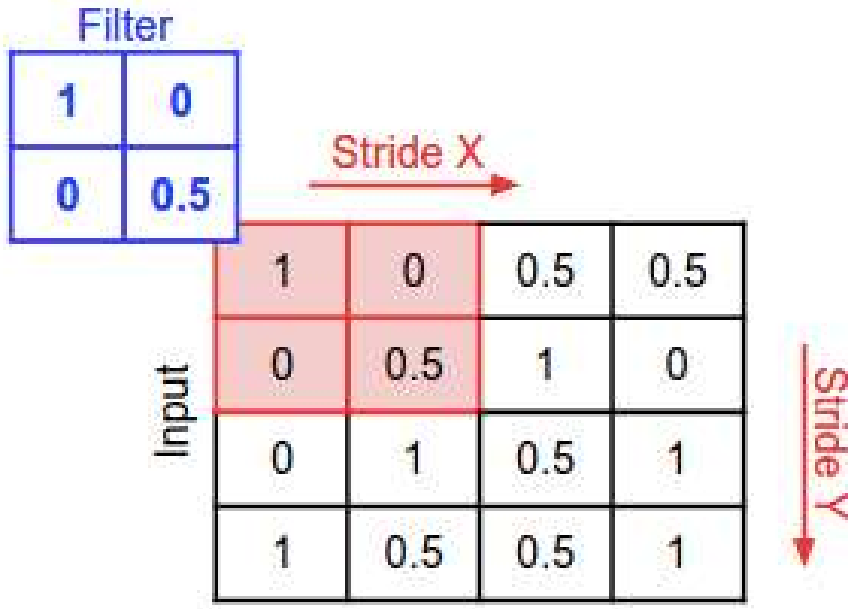


CNN Convolution Parameters

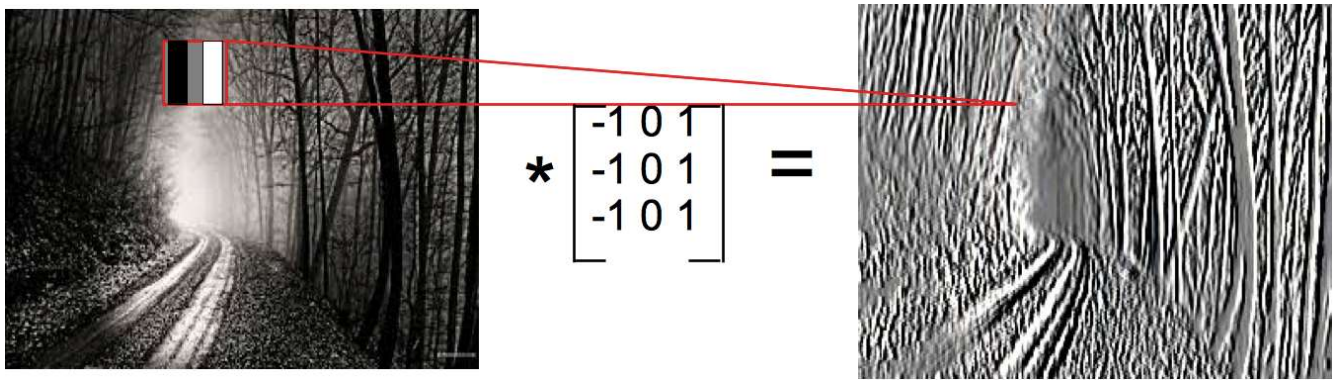
CNN — Parameters

- **Filters**: Represents the amount of filters in a CL.
- **Kernel Size**: Defines the dimensions of the filters.
- **Stride**: Sets the size of the filter shift step.
- **Padding**: defines whether or not there is entry zeroing, influencing the output dimensions:

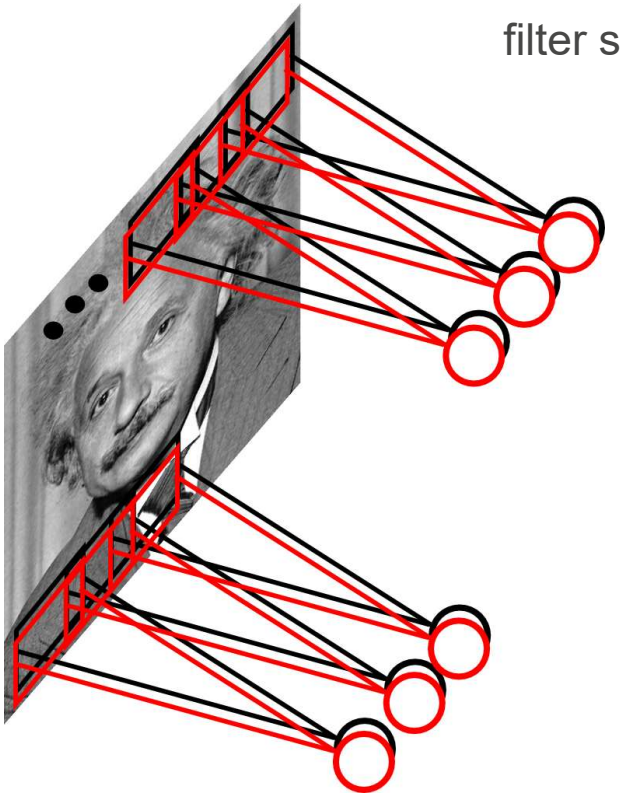
CNN Convolution Parameters



Example



Multiple Feature Maps



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

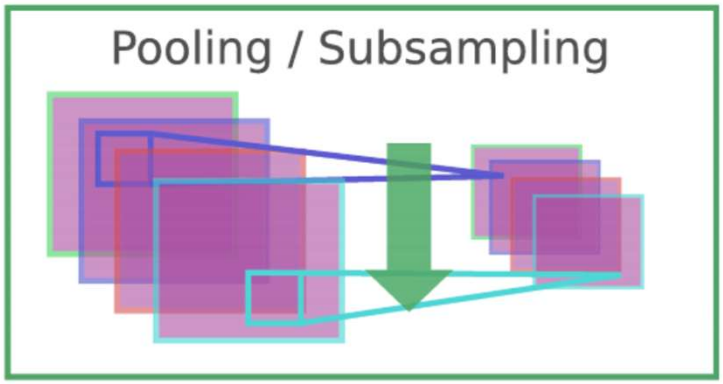
Convolutional Neural Networks

- 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)

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



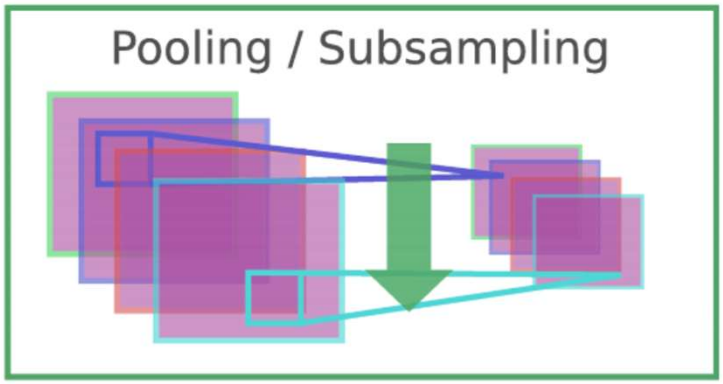
Jarret et al. 2009

- 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

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

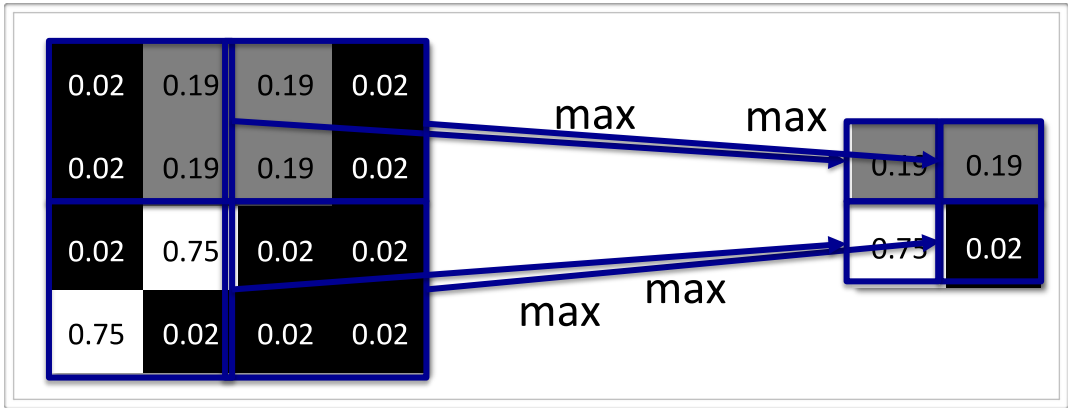


Jarret et al. 2009

- 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

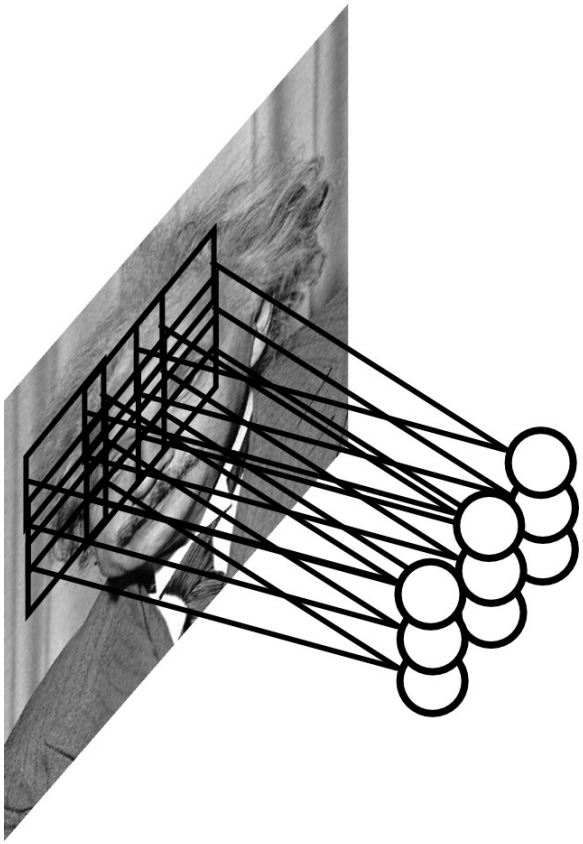
Example: Pooling

- Illustration of pooling/subsampling operation



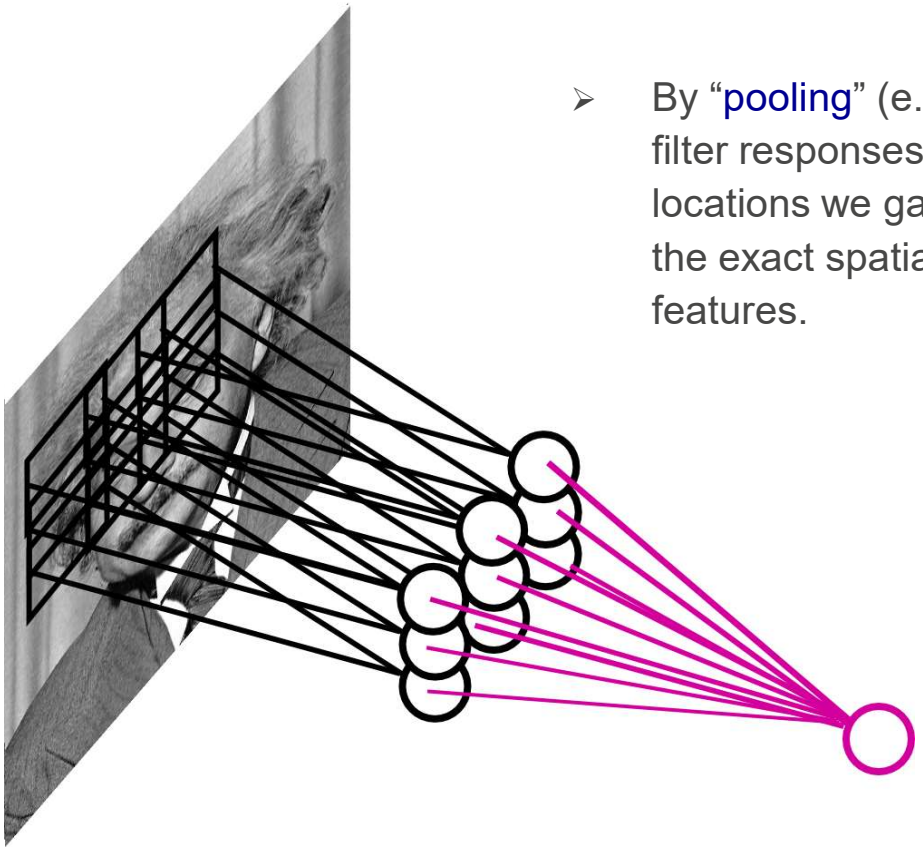
- Why pooling?
  - Introduces invariance to local translations
  - Reduces the number of hidden units in hidden layer

Example: Pooling



- can we make the detection robust to the exact location of the eye?

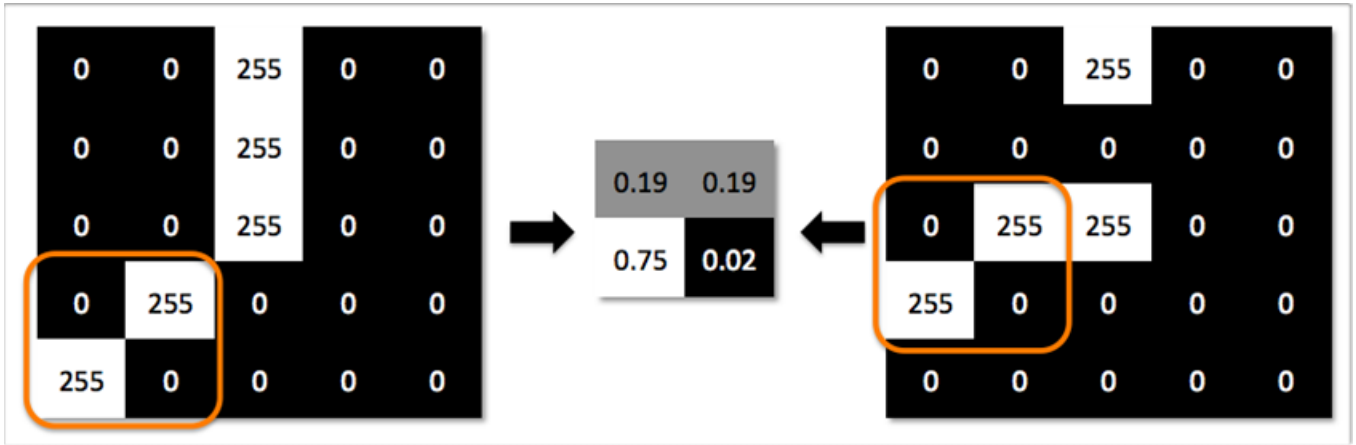
Example: Pooling



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

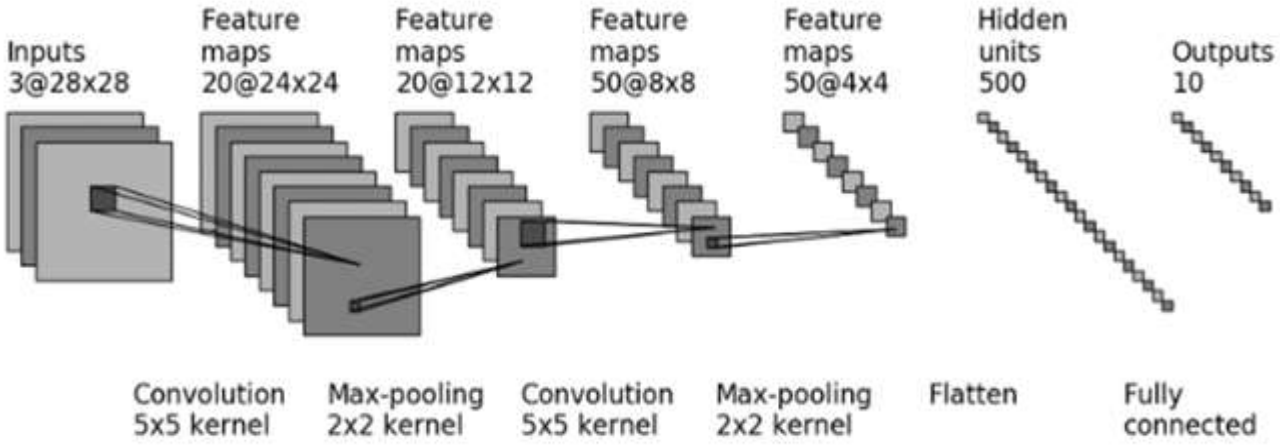
Translation Invariance

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



Convolutional Network

- Convolutional neural network alternates between the convolutional and pooling layers



Structure of LeNet-5.

From Yann LeCun's slides

Convolutional 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

Gradient of Pooling Layer

- Let l be the loss function
  - For max pooling operation $y_{ijk} = \max_{p,q} x_{i,j+p,k+q}$, the gradient for x_{ijk} is

$$\nabla_{x_{ijk}} l = 0, \text{ except for } \nabla_{x_{i,j+p',k+q'}} l = \nabla_{y_{ijk}} l$$

  - where p', q' = argmax $x_{i,j+p,k+q}$
  - In other words, only the "winning" units in layer x get the gradient from the pooled layer
  - For the average operation $y_{ijk} = \frac{1}{m^2} \sum_{p,q} x_{i,j+p,k+q}$, the gradient for x_{ijk} is

$$\nabla_x l = \frac{1}{m^2} \text{upsample}(\nabla_y l)$$

where you should calculate upsample(.) as an excerise.

Gradient of Convolutional Layer

The goal is to compute the gradient of the loss function w.r.t. to the weights of the filters in layer h^u and input X^{u-1} given the gradient in the layer h^u.

Remember $h^u = g(w^u * X^{u-1})$, where g is some nonlinear operator (nonlinear activation, pooling,..)

Assume you have computed the gradient of loss function, l, up to the **current hidden convolutional layer $\boldsymbol{h^u}$**, i.e., $\partial_{h^u_{ijk}} \triangleq \nabla^u_{h_{ijk}} l = \frac{\partial l}{\partial h^u_{ijk}}$.

Here h stands for "hidden" layer and has been introduced for the ease of notation. Also, index i denotes the channel number of the current hidden layer h_u. That is, $i = 1,2,\dots,C_u$ and $u = 1,2,\dots L$, where C_u denotes the number of channels in layer u and L is the total number of layers.

For simplicity drop the channel index and the layer number. So, $\partial_{h_{ij}} \triangleq \frac{\partial l}{\partial h_{ij}}$.

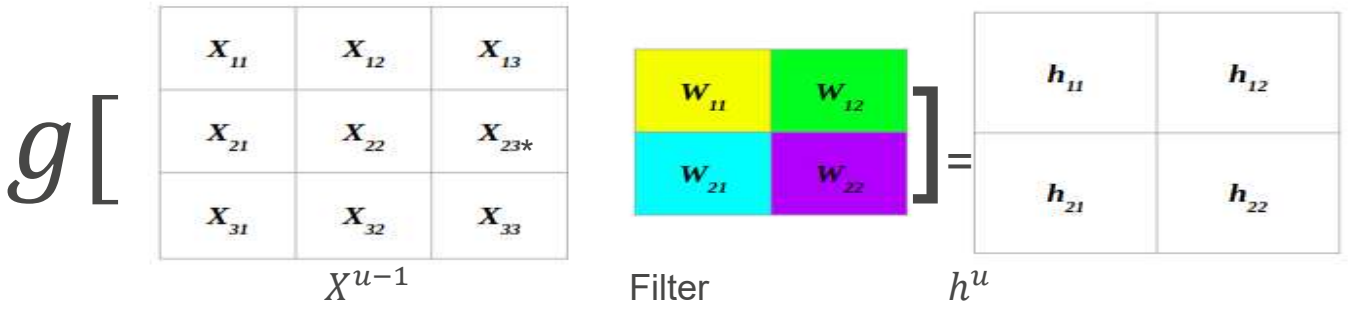
All the computations should be done for all the channels and for all the convolutional layers.

Similarly define the gradient w.r.t. to the filter coefficients as $\partial_{w_{ij}} \triangleq \frac{\partial l}{\partial w_{ij}}$

Gradient of Convolutional Layer -- Continue

Now, we establish the gradient operation visually [1].

Assume in forward pass, we have convolved a 3x3 input with a kernel 2X2 which outputs a 2X2 matri



We calculate this for g$(z) = z$. Please extend to the general case as an exercise

With the notation from the previous slide, we can write:

$$\partial_{w_{11}} = X_{11}\partial_{h_{11}} + X_{12}\partial_{h_{12}} + X_{21}\partial_{h_{21}} + X_{22}\partial_{h_{22}}$$
$$\partial_{w_{12}} = X_{12}\partial_{h_{11}} + X_{13}\partial_{h_{12}} + X_{22}\partial_{h_{21}} + X_{23}\partial_{h_{22}}$$
$$\partial_{w_{21}} = X_{21}\partial_{h_{11}} + X_{22}\partial_{h_{12}} + X_{31}\partial_{h_{21}} + X_{32}\partial_{h_{22}}$$
$$\partial_{w_{22}} = X_{22}\partial_{h_{11}} + X_{23}\partial_{h_{12}} + X_{32}\partial_{h_{21}} + X_{33}\partial_{h_{22}}$$

Gradient of Convolutional Layer -- Continue

- This is our old friend, discrete convolution operator:

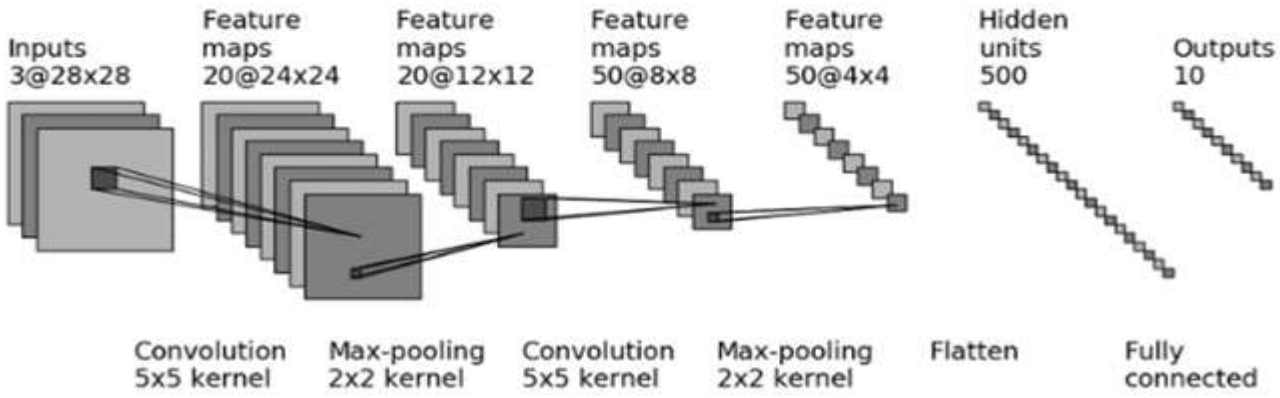
$$\partial_W = X * \partial_h$$

- Where $\partial_{w_{ij}} = \sum_{p,q} X_{i+p,j+q} \partial_{h_{ij}}$

- Similarly, we can compute the gradient of the loss w.r.t. X (input layer , or X^{u-1}) since we need these gradients in order to propagate the gradient to the layers towards input of the CNN.

Convolutional Network

• Convolutional neural network alternates between the convolutional and pooling layers



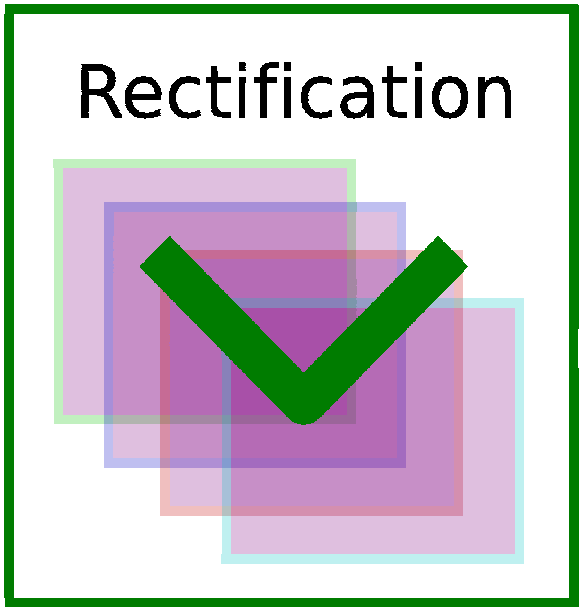
Structure of LeNet-5.

• Need to introduce other operations that can improve object recognition.

Rectification

- Rectification layer: $y_{ijk} = |x_{ijk}|$

- introduces invariance to the sign of the unit in the previous layer
- for instance, loss of information of whether an edge is black-to-white or white-to-black



Local Contrast Normalization

- Perform local contrast normalization

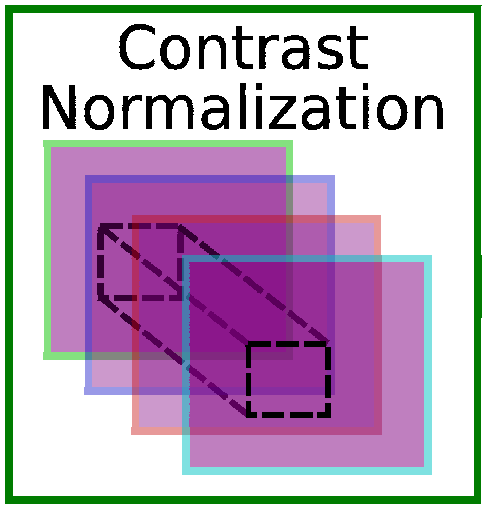
Local average

$$v_{ijk} = x_{ijk} - \sum_{ipq} \alpha_{pq} x_{i,j+p,k+q}$$

$$y_{ijk} = v_{ijk} / \max(c, \sigma_{jk})$$

Local stdev

$$\sigma_{jk} = (\sum_{ipq} \alpha_{pq} v^2_{i,j+p,k+q})^{1/2}, \quad \sum_{pq} \alpha_{pq} = 1$$

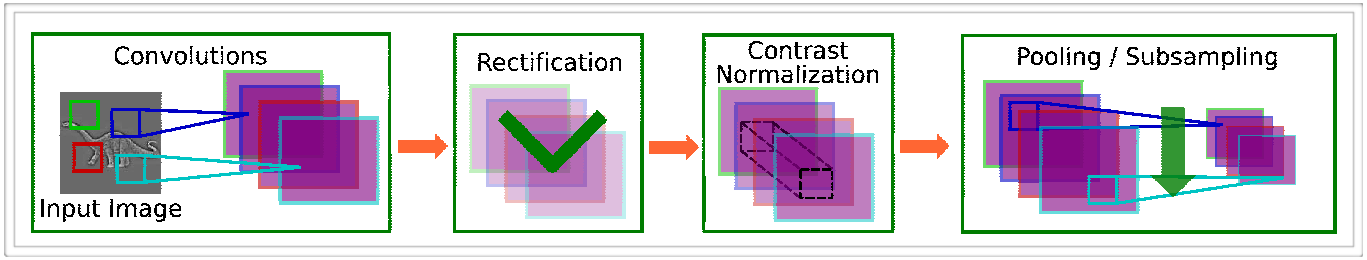


where c is a small constant to prevent division by 0 and $\alpha_{pq} \geq 0$.

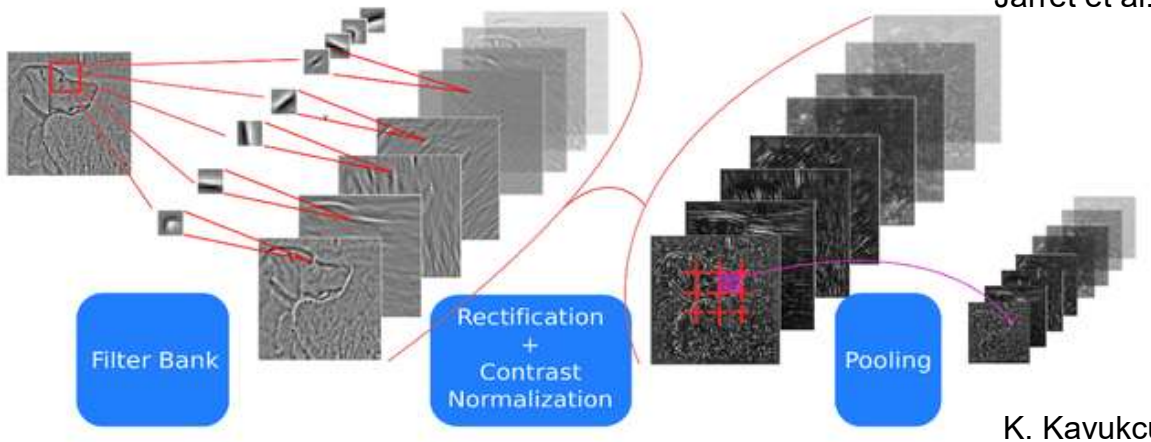
- reduces unit's activation if neighbors are also active
- creates competition between feature maps
- scales activations at each layer better for learning

Convolutional Network

• These operations are inserted after the convolutions and before the pooling



Jarret et al. 2009



K. Kavukcuoglu

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Learned linear transformation to adapt to non-linear activation function (**γ and β are trained**)

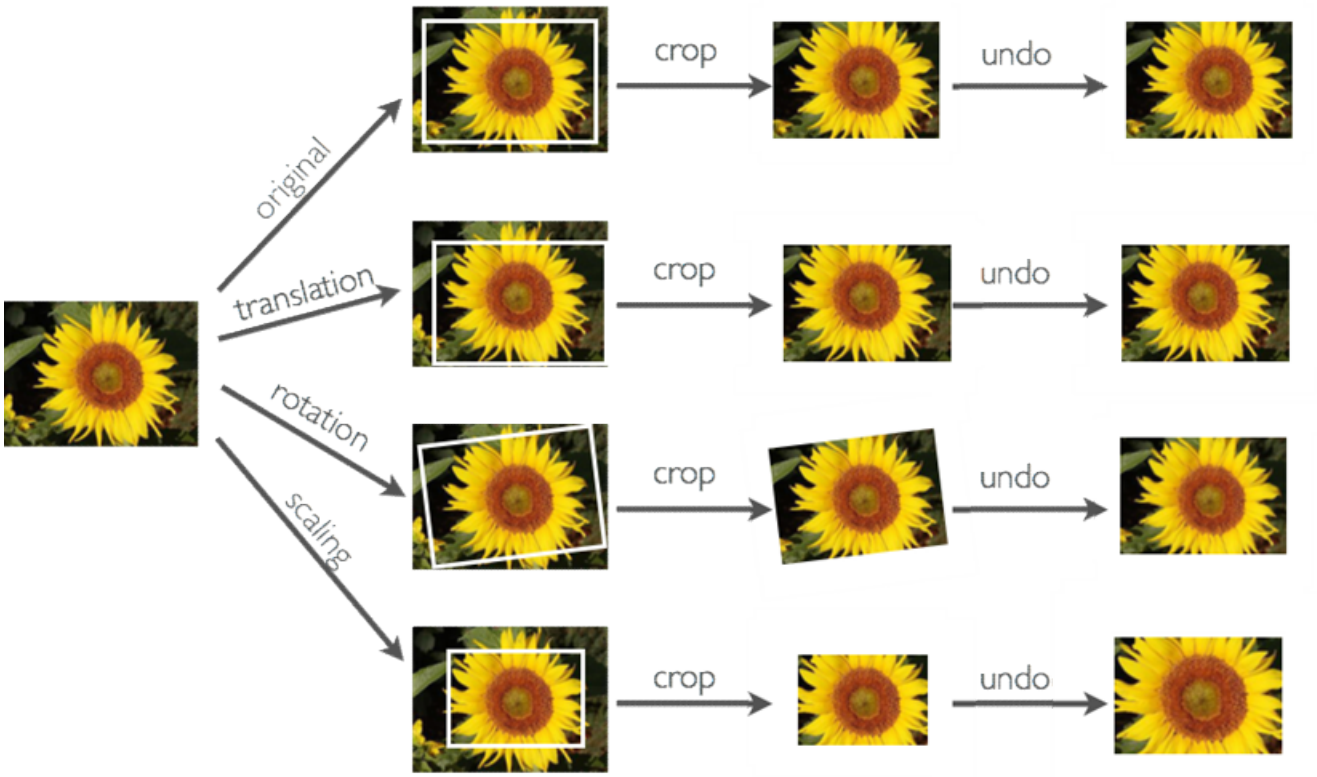
Invariance by Dataset Expansion (Augmentation)

- 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

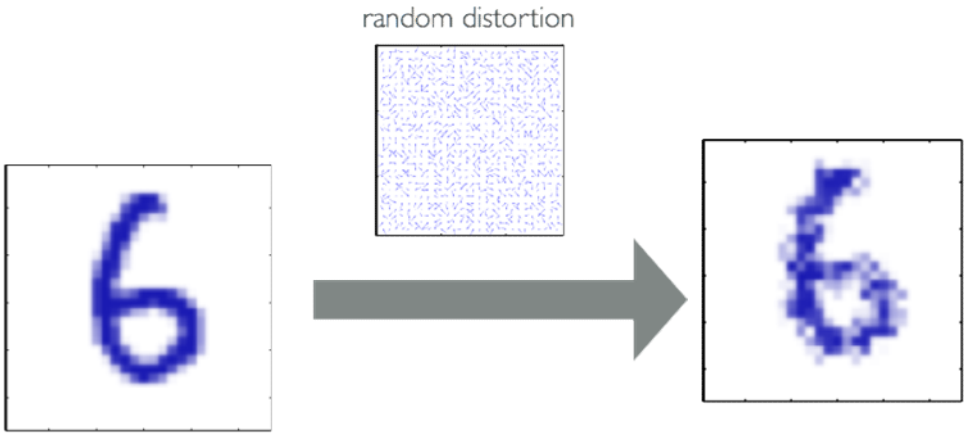
- However, it's easy to artificially generate data with such transformations
  - could use such data as additional training data
  - neural network can potentially learn to be invariant to such transformations

Generating Additional Examples



Elastic Distortions

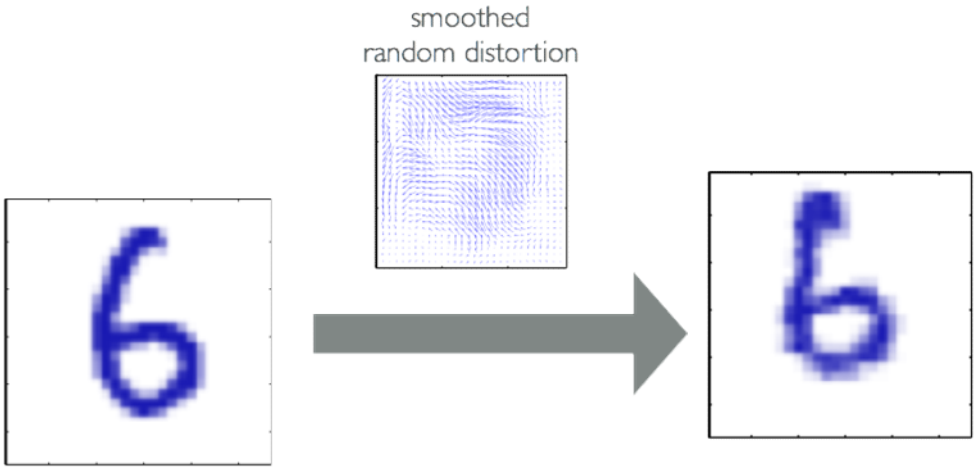
- Can add ‘‘elastic’’ deformations (useful in character recognition)
- We can do this by applying a ‘‘distortion field’’ to the image
  - a distortion field specifies where to displace each pixel value



Bishop’s book

Elastic Distortions

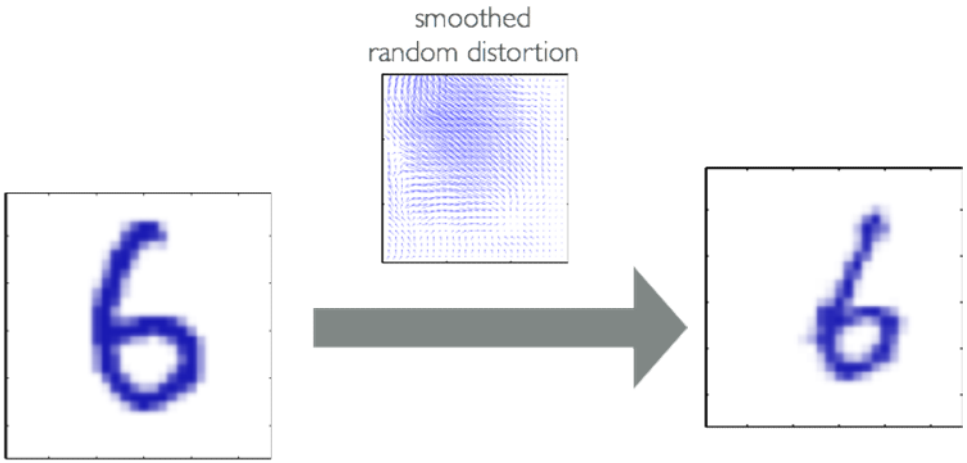
- Can add ‘‘elastic’’ deformations (useful in character recognition)
- We can do this by applying a ‘‘distortion field’’ to the image
  - a distortion field specifies where to displace each pixel value



Bishop’s book

Elastic Distortions

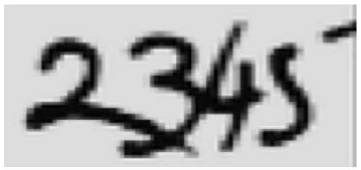
- Can add ‘‘elastic’’ deformations (useful in character recognition)
- We can do this by applying a ‘‘distortion field’’ to the image
  - a distortion field specifies where to displace each pixel value



Bishop’s book

Conv Nets: Examples

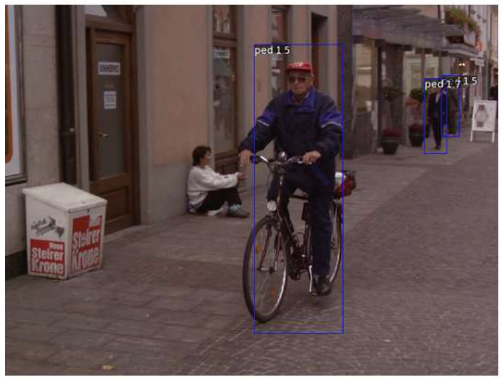
- Optical Character Recognition, House Number and Traffic Sign classification

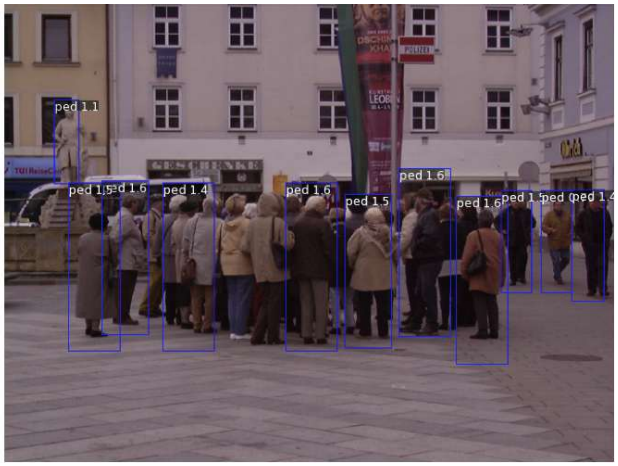


Ciresan et al. “MCDNN for image classification” CVPR 2012
Wan et al. “Regularization of neural networks using dropconnect” ICML 2013
Goodfellow et al. “Multi-digit nuber recognition from StreetView...” ICLR 2014
Jaderberg et al. “Synthetic data and ANN for natural scene text recognition” arXiv 2014

Conv Nets: Examples

- Pedestrian detection

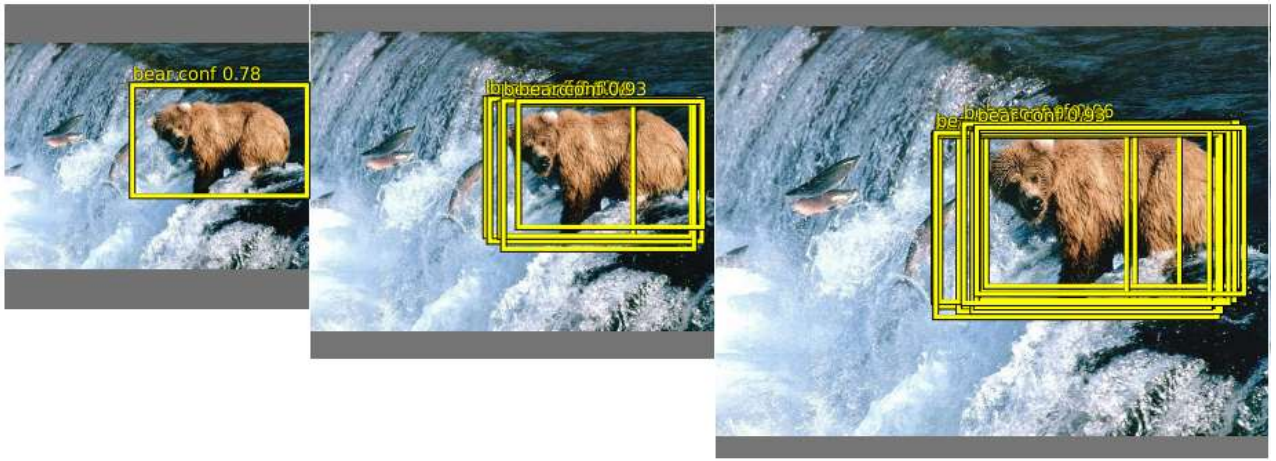




Sermanet et al. “Pedestrian detection with unsupervised multi-stage..” CVPR 2013

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

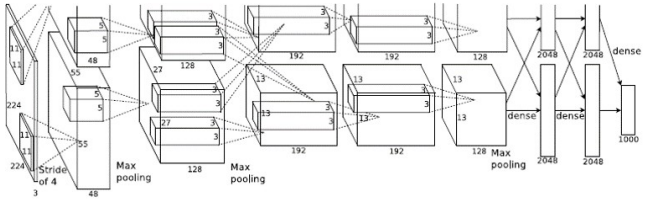
Examples of Hammer

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

Important Breakthroughs

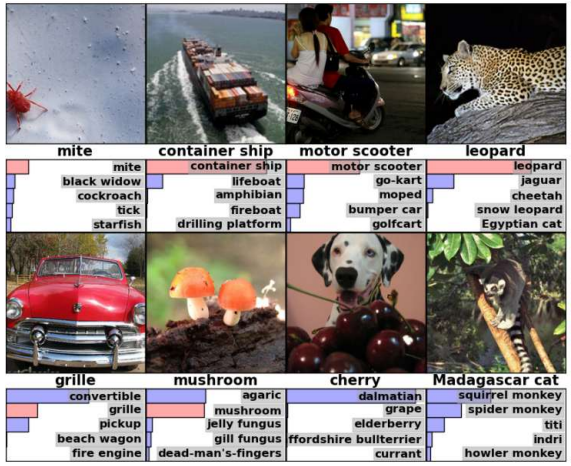
- Deep Convolutional Nets for Vision (Supervised)

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



IMAGENET

1.2 million training images
1000 classes

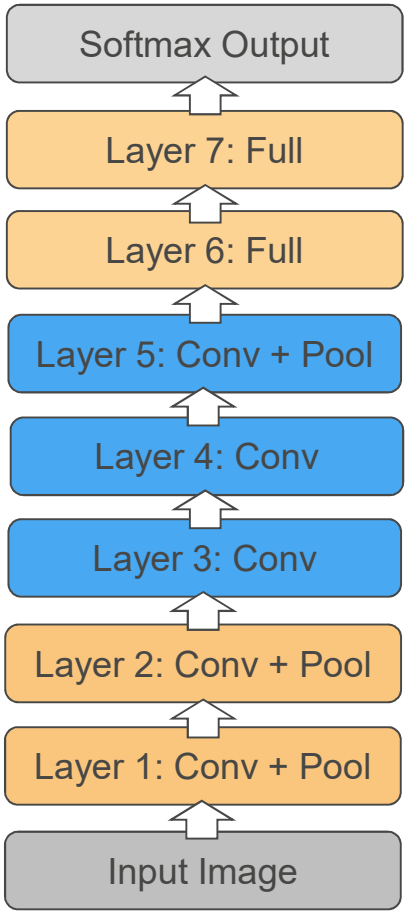


Architecture

- How can we select the right architecture:
  - Manual tuning of features is now replaced with the manual tuning of architectures
- Depth
- Width
- Parameter count

AlexNet

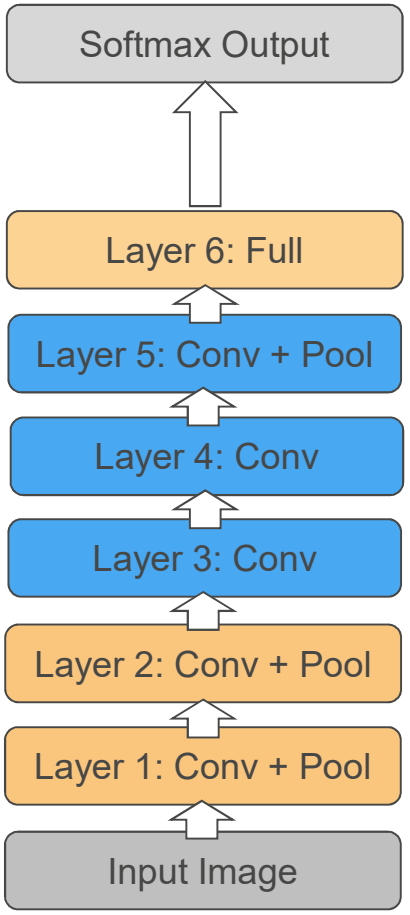
- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error



[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet

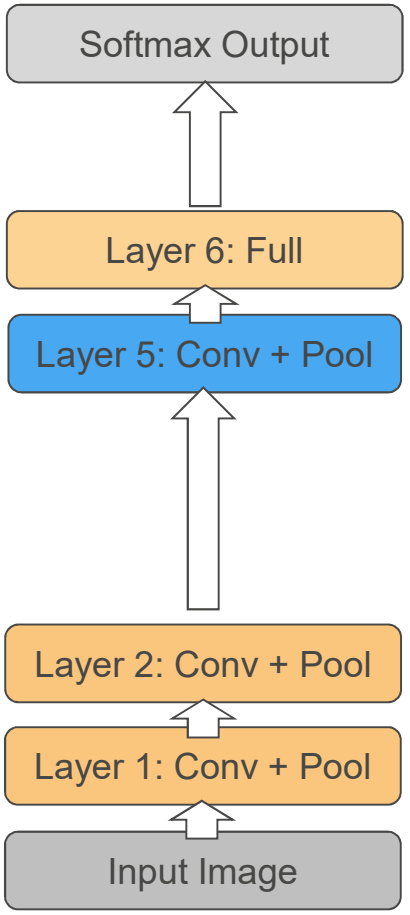
- Remove top fully connected layer 7
- Drop ~16 million parameters
- Only 1.1% drop in error!



[From Rob Fergus' CIFAR 2016 tutorial]

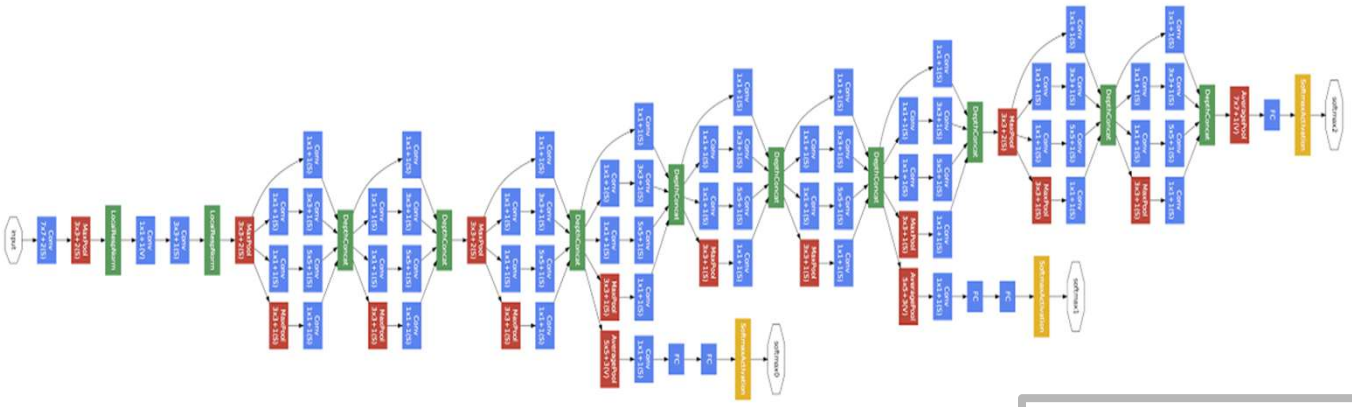
AlexNet

- Let us remove upper feature extractor layers and fully connected:
  - Layers 3,4, 6 and 7
- Drop ~50 million parameters
- **33.5 drop in error!**
- Depth of the network is the key.



[From Rob Fergus' CIFAR 2016 tutorial]

GoogLeNet



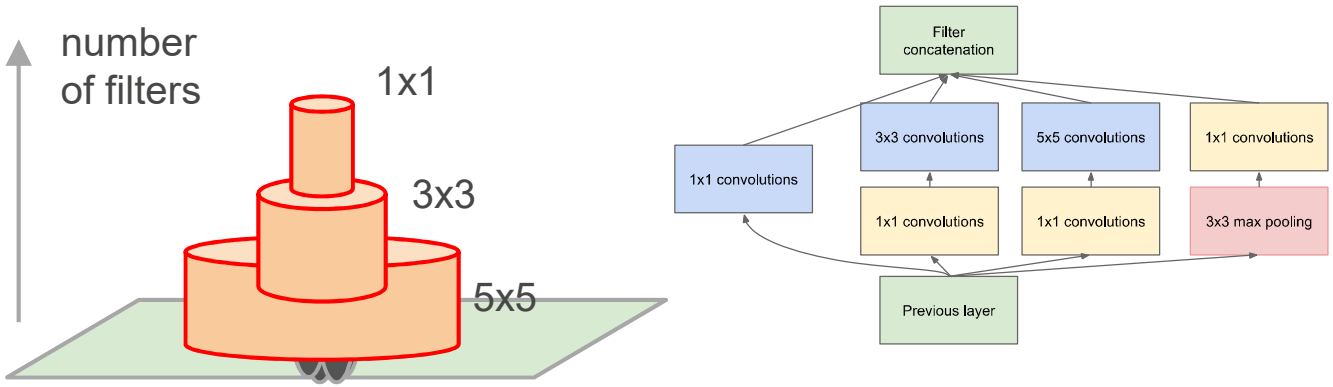
- 24 layer model that uses so-called inception module.

Convolution
Pooling
Softmax
Other

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

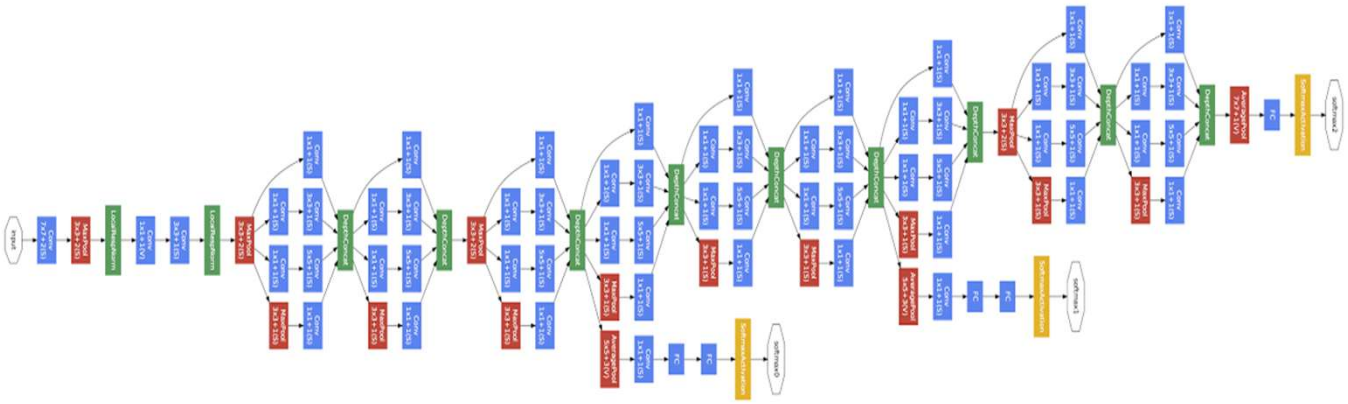
GoogLeNet

- 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

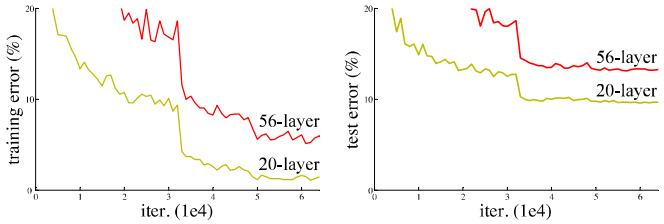


- Width of inception modules ranges from 256 filters (in early modules) to 1024 in top inception modules.
- Can remove fully connected layers on top completely
- Number of parameters is reduced to 5 million
- 6.7% top-5 validation error on Imagnet

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

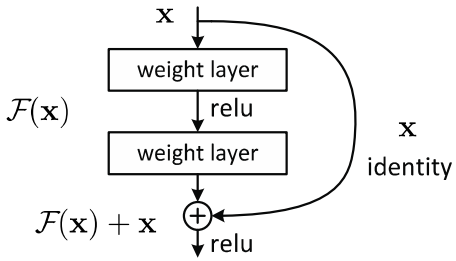
Residual Networks

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



Key idea: introduce “pass through” into each layer

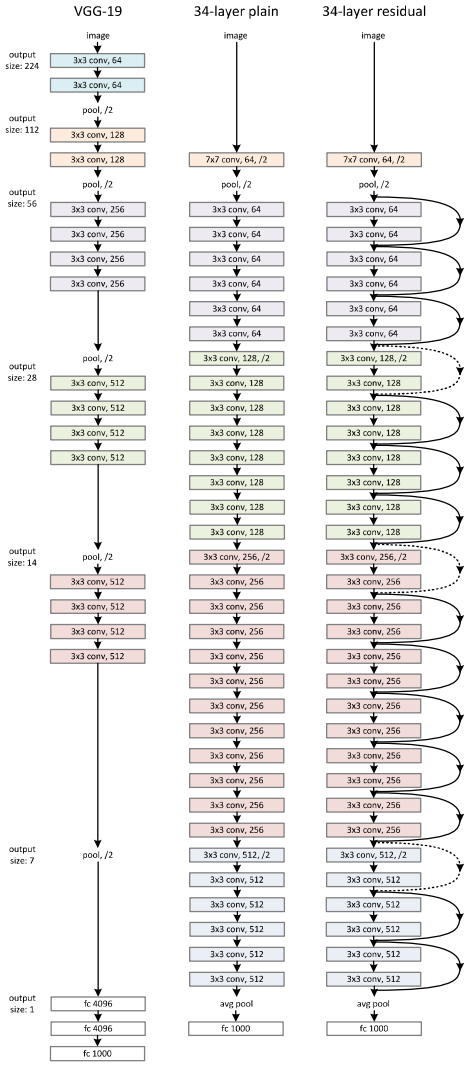
Thus only residual now needs to be learned



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]

Choosing the Architecture

- Task dependent
- Cross-validation
- [Convolution → 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]