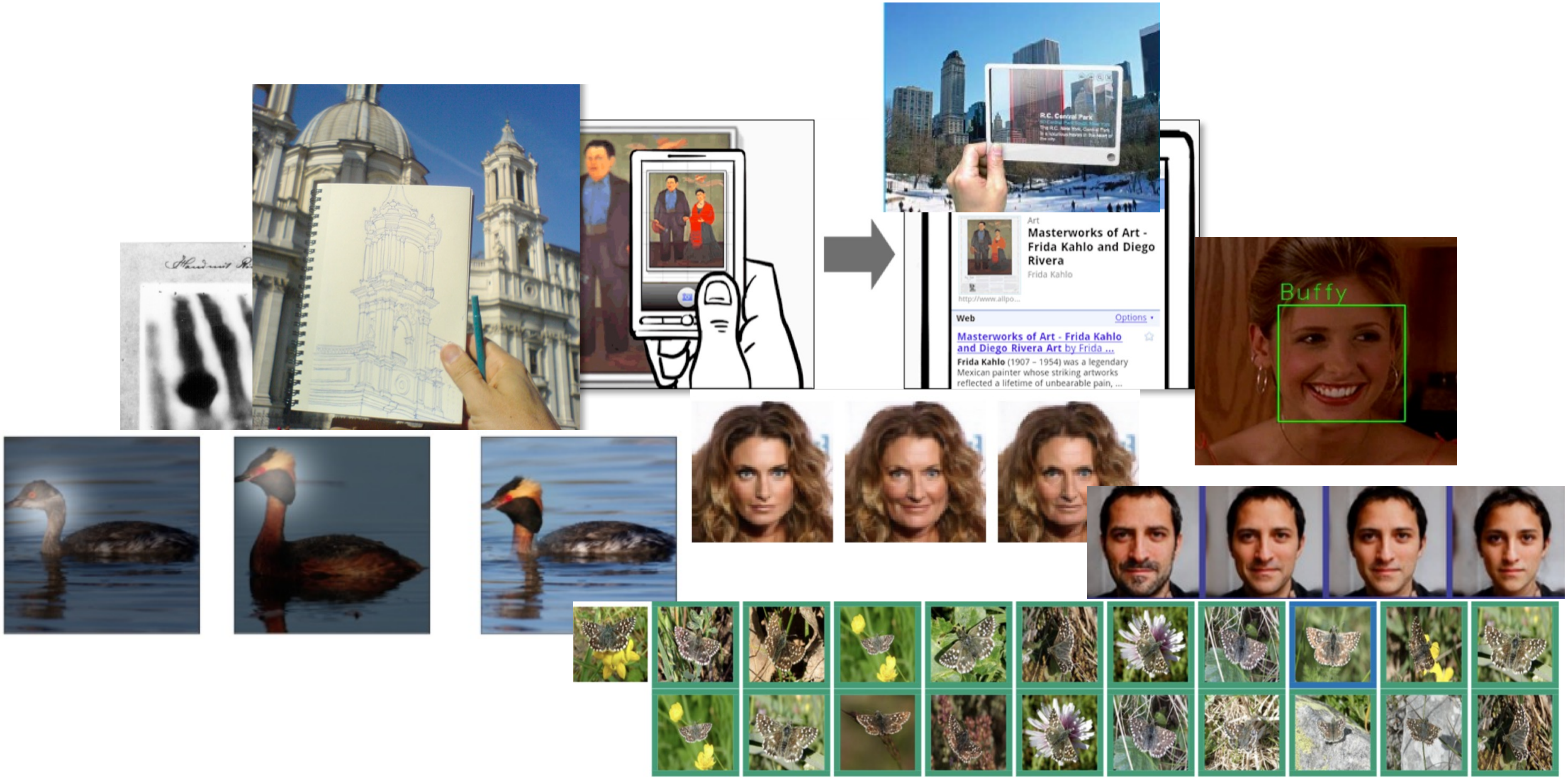


COURS RDFIA deep Image

Matthieu Cord
Sorbonne University

Course Outline

1. Computer Vision and ML basics: Visual (local) feature detection and description, Bag of Word Image representation, Linear classification (SVM)
2. Introduction to Neural Networks (NNs)
3. Machine Learning basics (2): Risk, Classification, Datasets, benchmarks and evaluation
4. **Neural Nets for Image Classification**
5. Vision Transformers
6. Transfer learning and domain adaptation
7. Segmentation and Detection
8. Generative models with GANs
9. Generative models with diffusion
10. Large VL models: CLIP, StableDiffusion, Flamingo
11. Control (to be checked) -- Explainable AI, Fairness

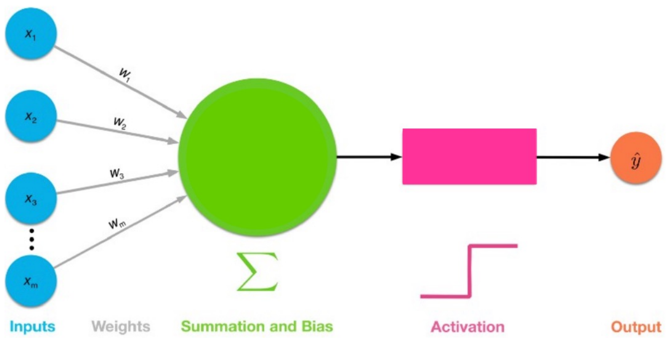
12/13 Bayesian deep learning

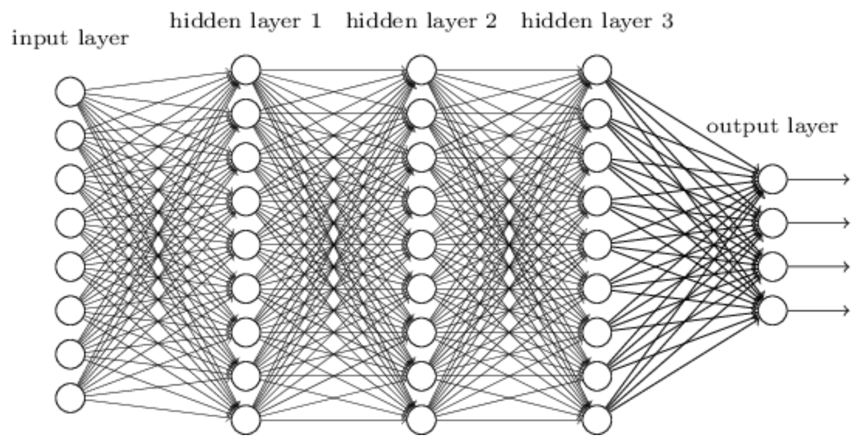
14 Robustness

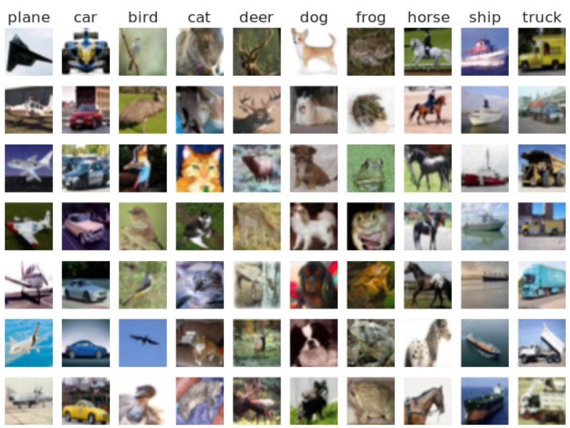
Outline
Convolutional Nets for visual classification

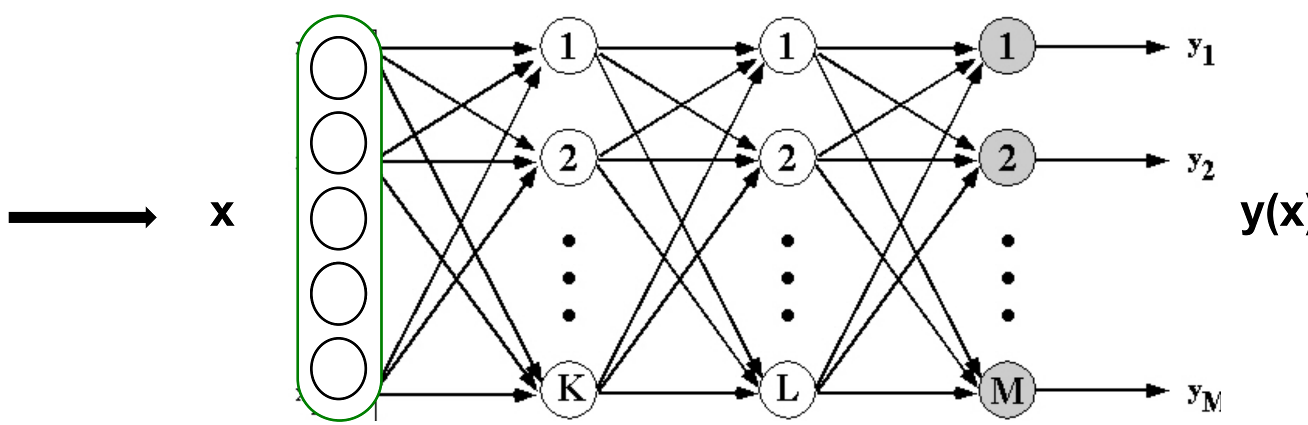
1. **Recap MLP**
2. Convolutional Neural Networks

Recap MLP

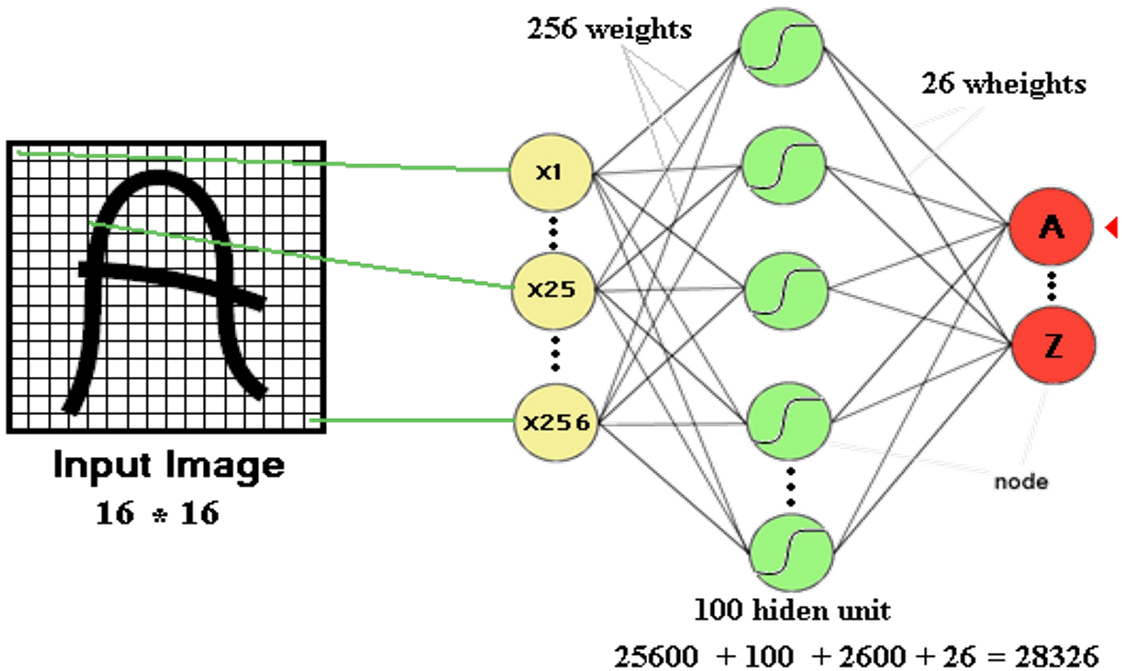

x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
Σ
$\hat{y}$
Inputs
Weights
Summation and Bias
Activation
Output

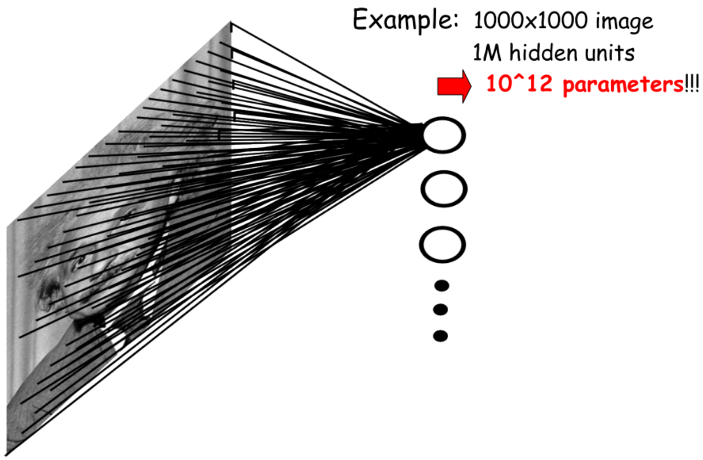

input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer


plane
car
bird
cat
deer
dog
frog
horse
ship
truck


x
1
2
K
1
2
L
1
2
M
y_1
y_2
y_M
y(x)

MLP example: brute force connection





First Pb: Scalability

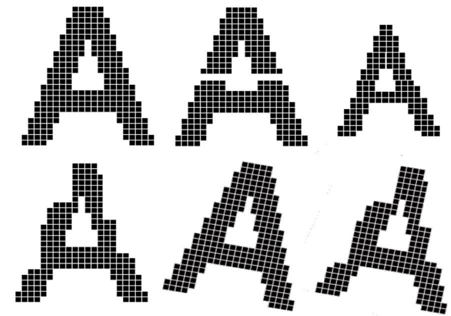
Large images => extremely large number of trainable parameters

MLP example: brute force connection

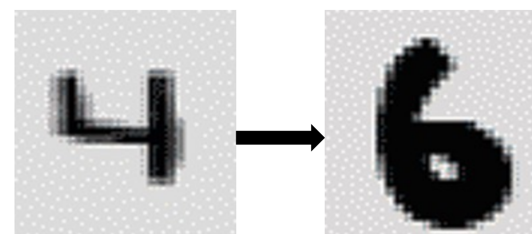
2d Pb: *Stability* of the representation

Expectation:

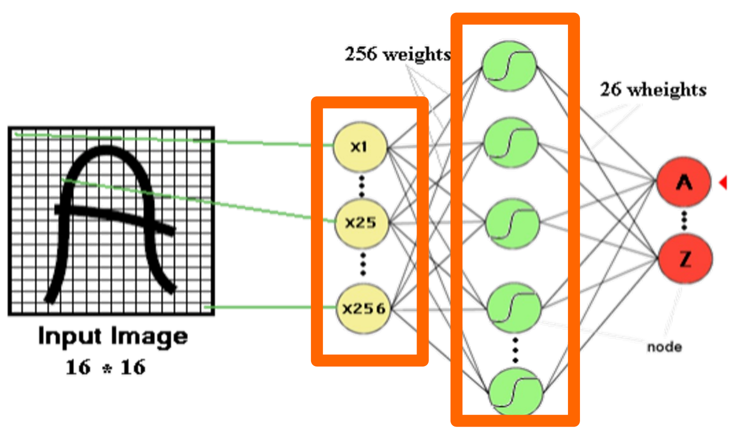
- *Small deformation in the input space*
  => similar representations



- *Large (or unexpected) transfo in the input space*
  => very dissimilar representations

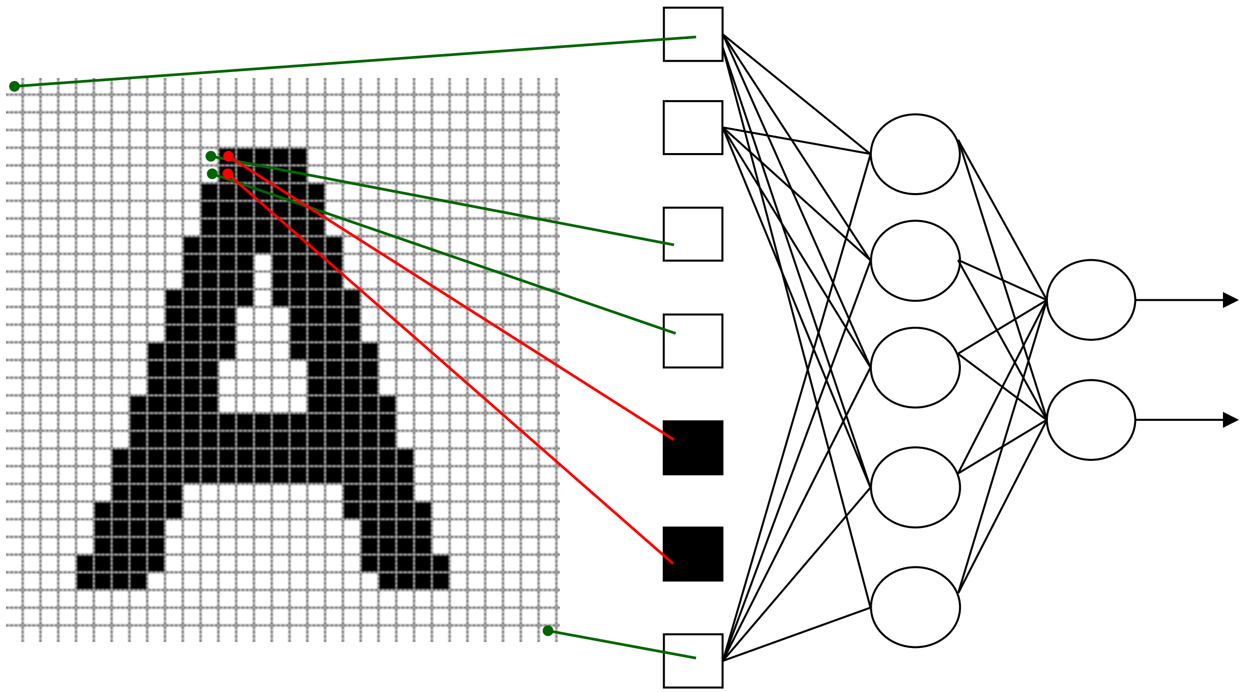


Representations:



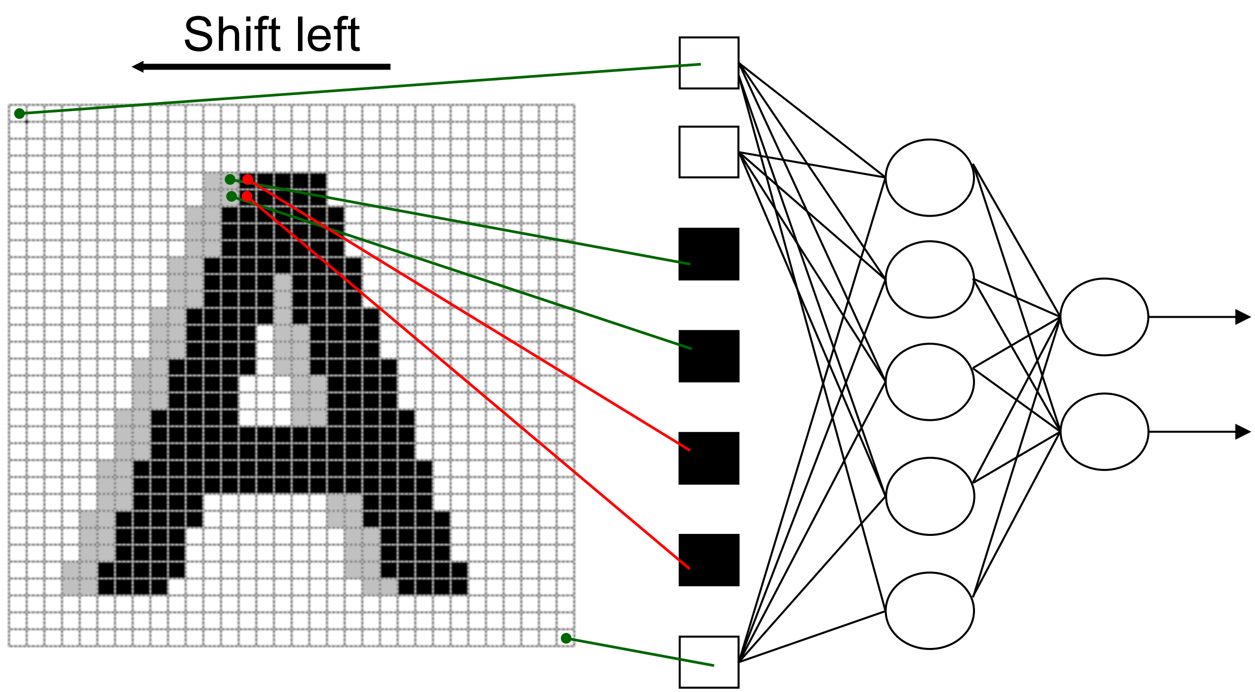
MLP example: brute force connection

Stability: Invariance/Robustness to (local) shifting, scaling, and other forms of (small) distortions?

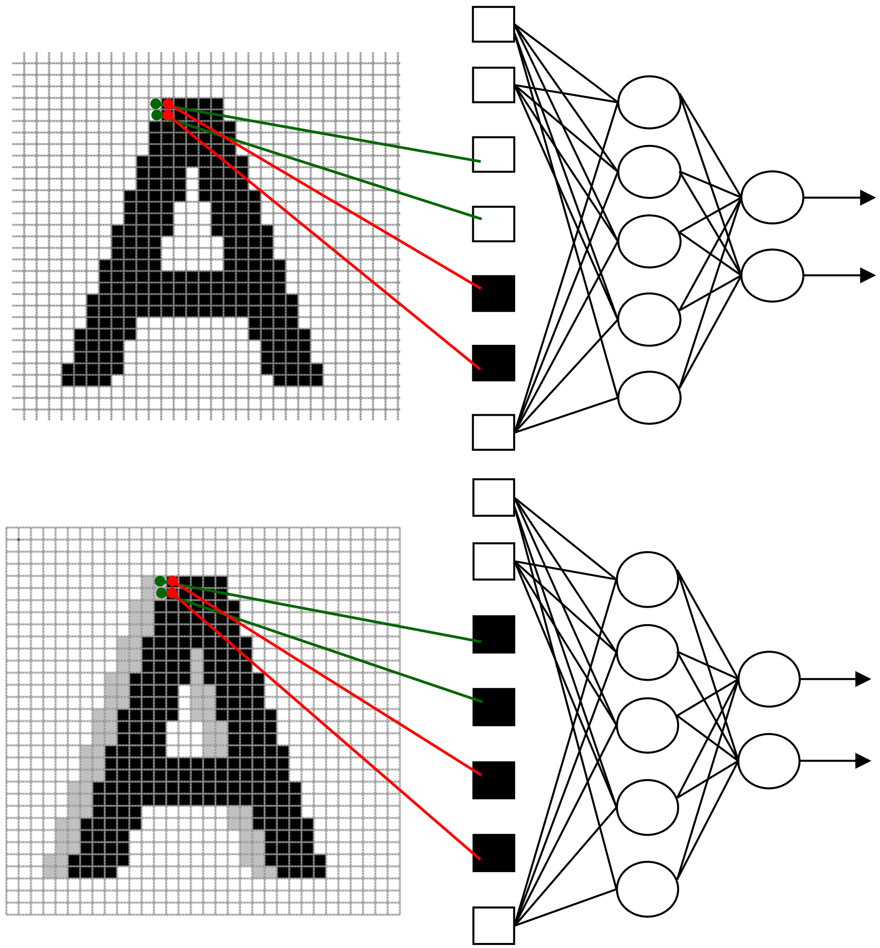


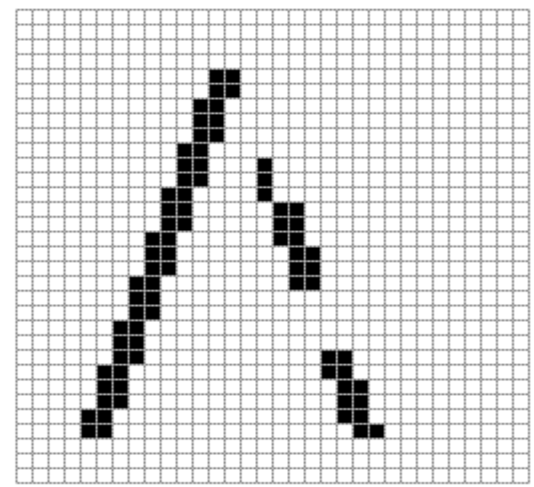
MLP example: brute force connection

Little or no invariance to shifting, scaling, and other forms of distortion



MLP example: brute force connection





154 input change
from 2 shift left
77 : black to white
77 : white to black

@LeCun

MLP example: brute force connection

Scaling and other forms of distortions => same pb

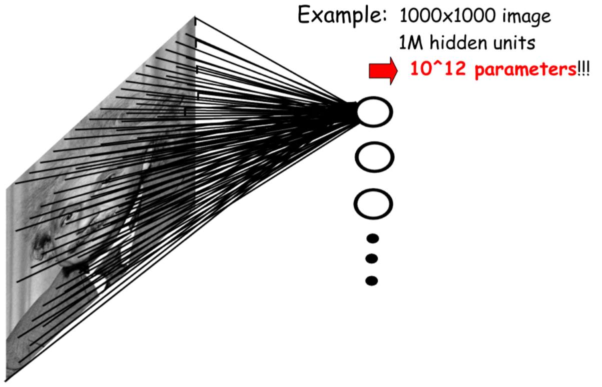
Conclusion of MLP on raw data

Brute force connection of images as input of MLP NOT a good idea

- No Invariance/Robustness of the representation because topology of the input data completely ignored
- Nb of weights grows largely with the size of the input image

How keep spatial topology?

How to limit the weight number?



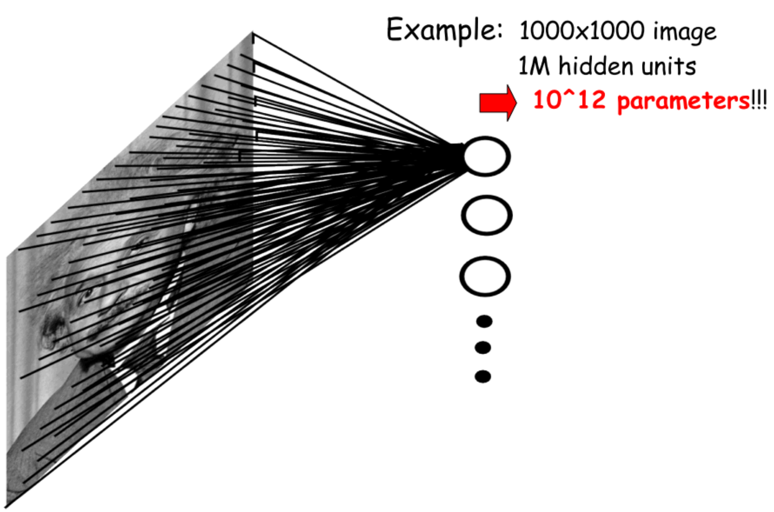
Outline
Convolutional Nets for visual classification

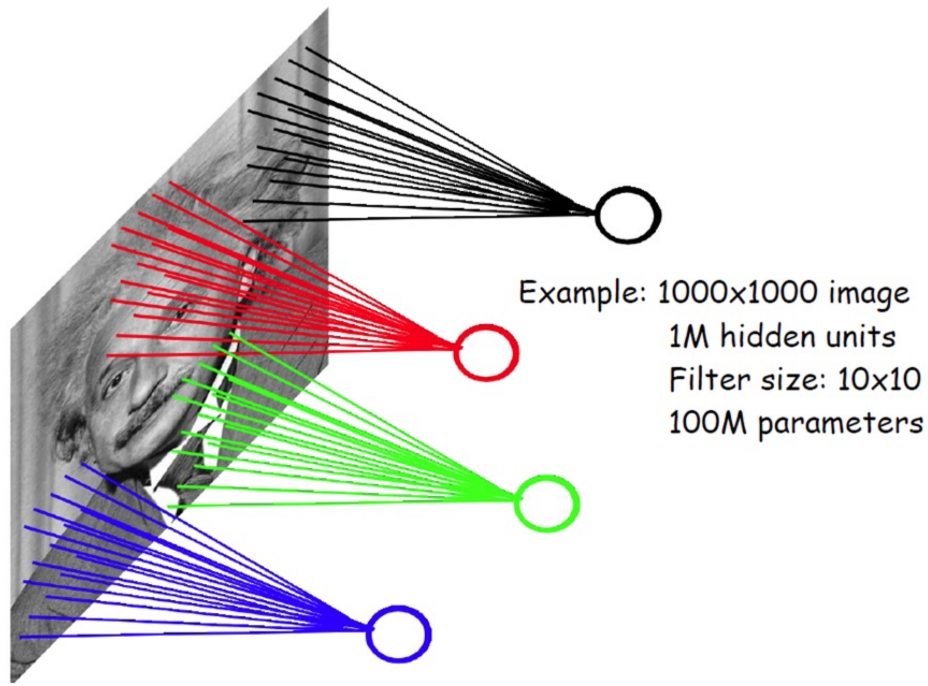
1. Recap MLP
2. **Convolutional Neural Networks**

How to limit the weight numbers?

1/ Locally connected neural networks

- **Sparse connectivity**: a hidden unit is only connected to a local patch (weights connected to the patch are called filter or kernel)
- Inspired by biological systems, where a cell is sensitive to a small sub-region of the input space, called a receptive field. Many cells are tiled to cover the entire visual field

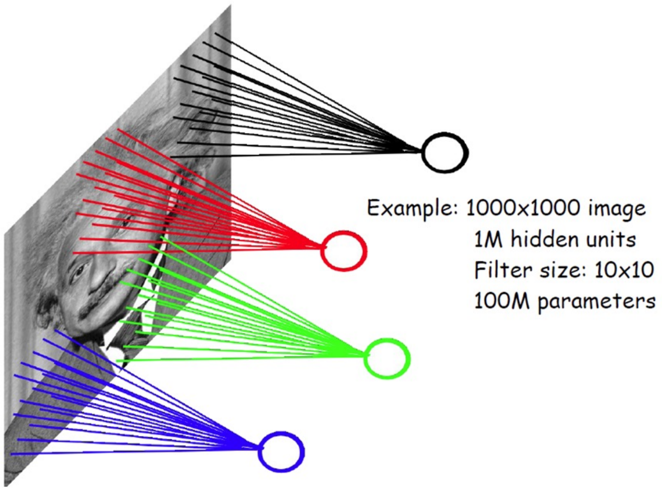


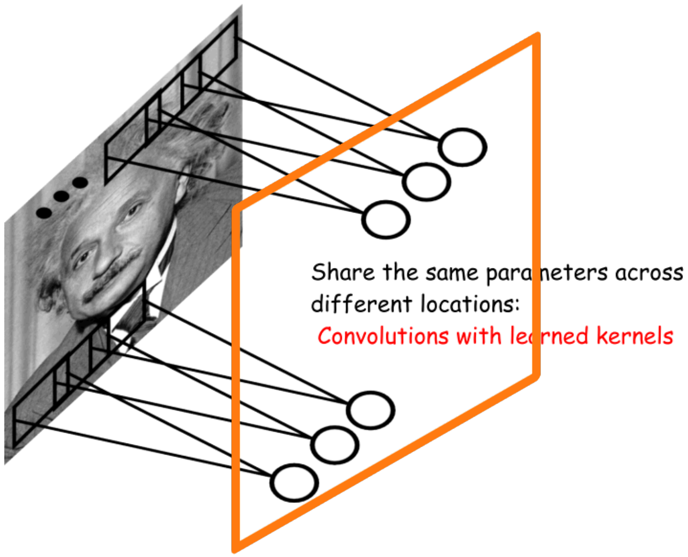


How to limit the weight numbers?

2/ Shared Weights

- Hidden nodes at different locations share the same weights
    - greatly reduces the number of parameters to learn
- Keep spatial information in a 2D feature map (hidden layer map)





⇒ Computing responses at hidden nodes equivalent to convoluting input image with a linear filter (learned)

⇒ A learned filter as a feature detector

Recap (1D/2D) convolution

1D discrete convolution of input signal x[n], with filter impulse response h[n], and output y[n]:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

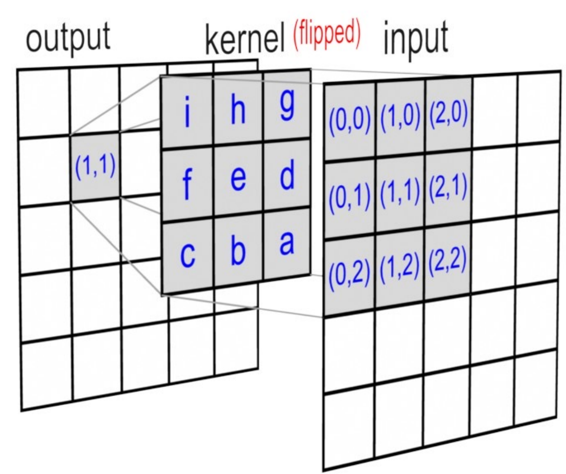
2D discrete convolution of input signal x[m,n], with filter impulse response h[m,n] (*kernel*), and output y[m,n]:

$$y[m,n] = x[m,n] * h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j]$$

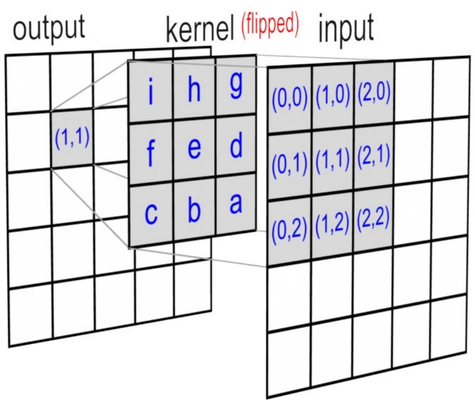
Example with impulse response (kernel) 3x3, and it's values are a, b, c, d,... :
(0,0) located in the center of the kernel

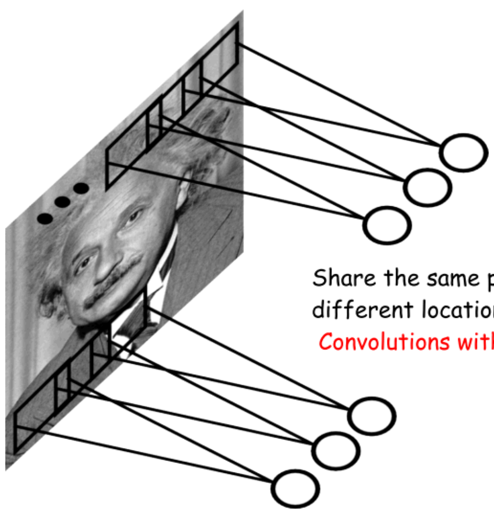
n \ m	-1	0	1
-1	a	b	c
0	d	e	f
1	g	h	i

$$\begin{aligned} y[1,1] &= \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[1-i, 1-j] \\ &= \quad x[0,0] \cdot h[1,1] \quad + x[1,0] \cdot h[0,1] \quad + x[2,0] \cdot h[-1,1] \\ &\quad + x[0,1] \cdot h[1,0] \quad + x[1,1] \cdot h[0,0] \quad + x[2,1] \cdot h[-1,0] \\ &\quad + x[0,2] \cdot h[1,-1] + x[1,2] \cdot h[0,-1] + x[2,2] \cdot h[-1,-1] \end{aligned}$$

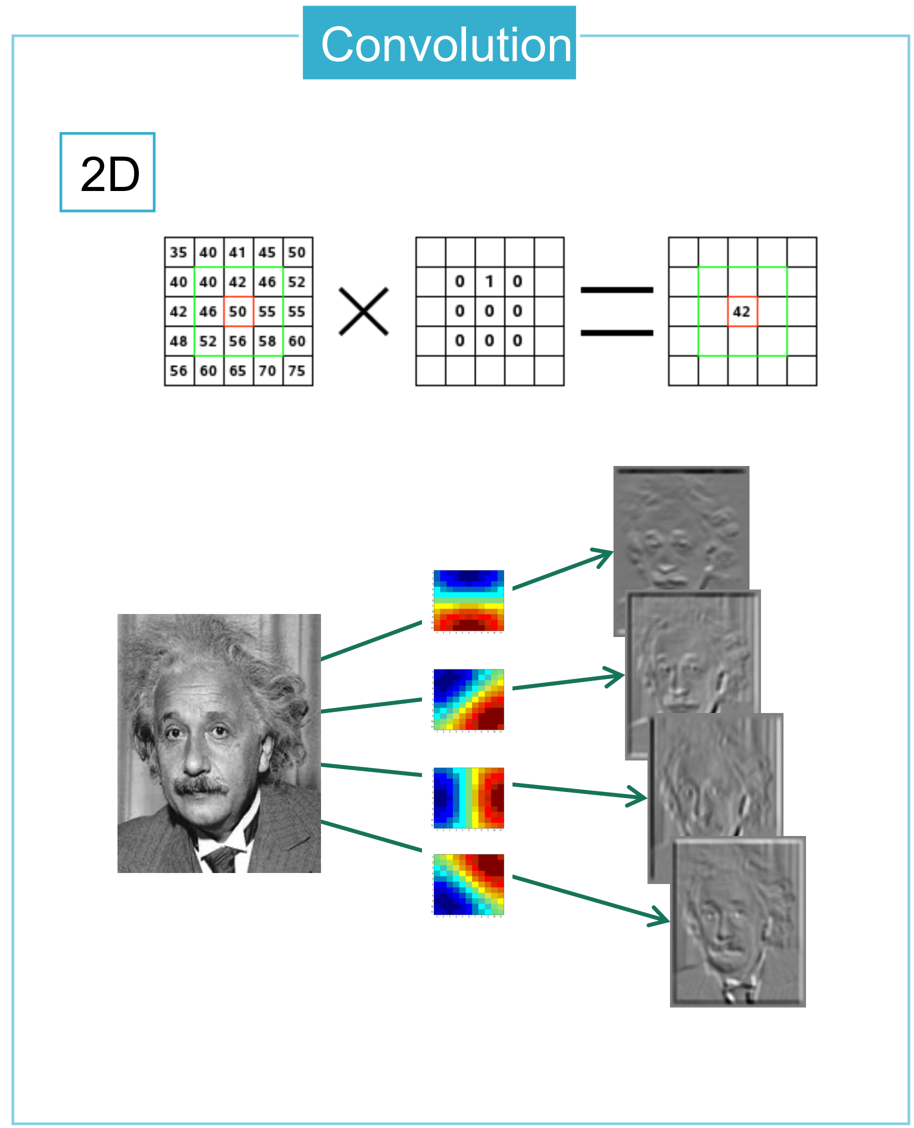


Ex. of convolution operator



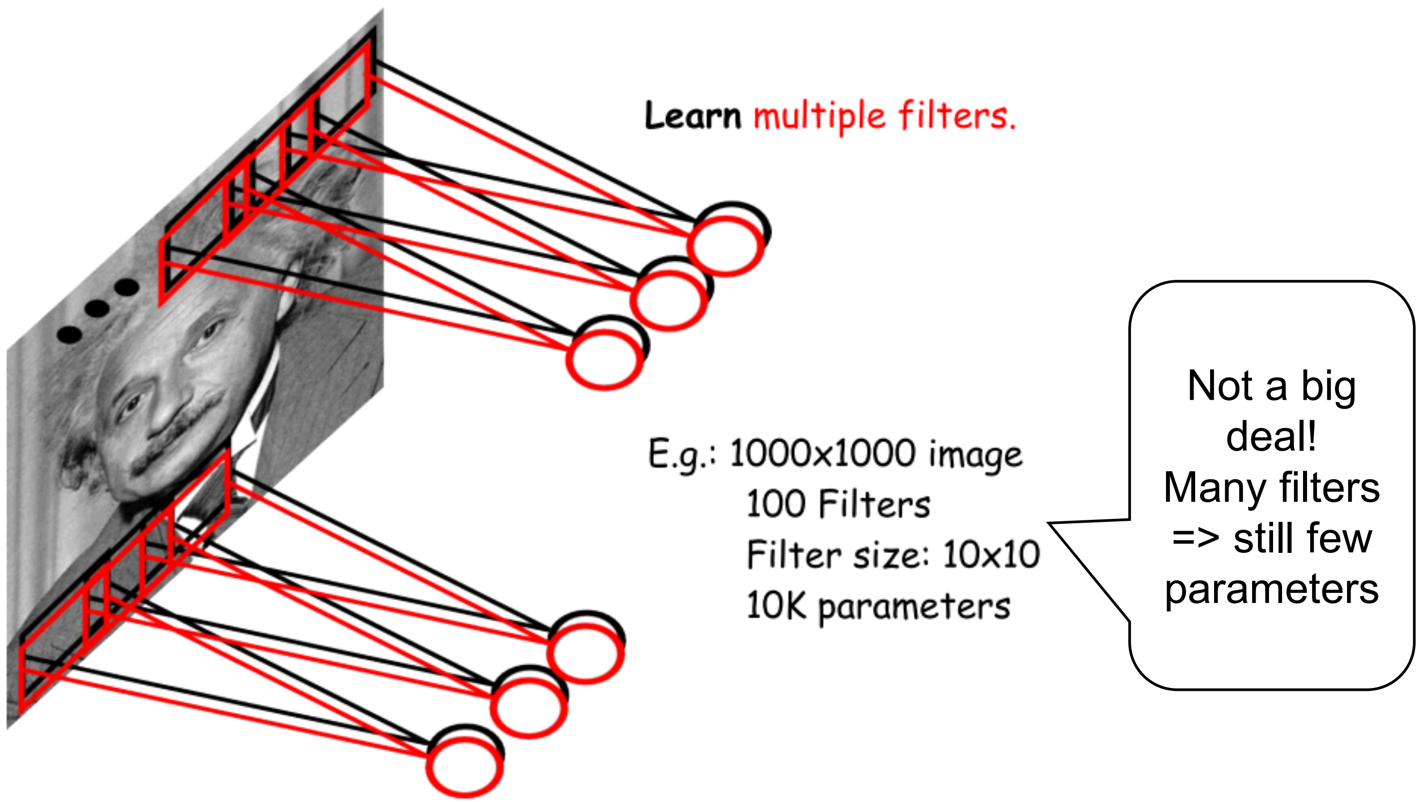


Share the same parameters across different locations:
Convolutions with learned kernels



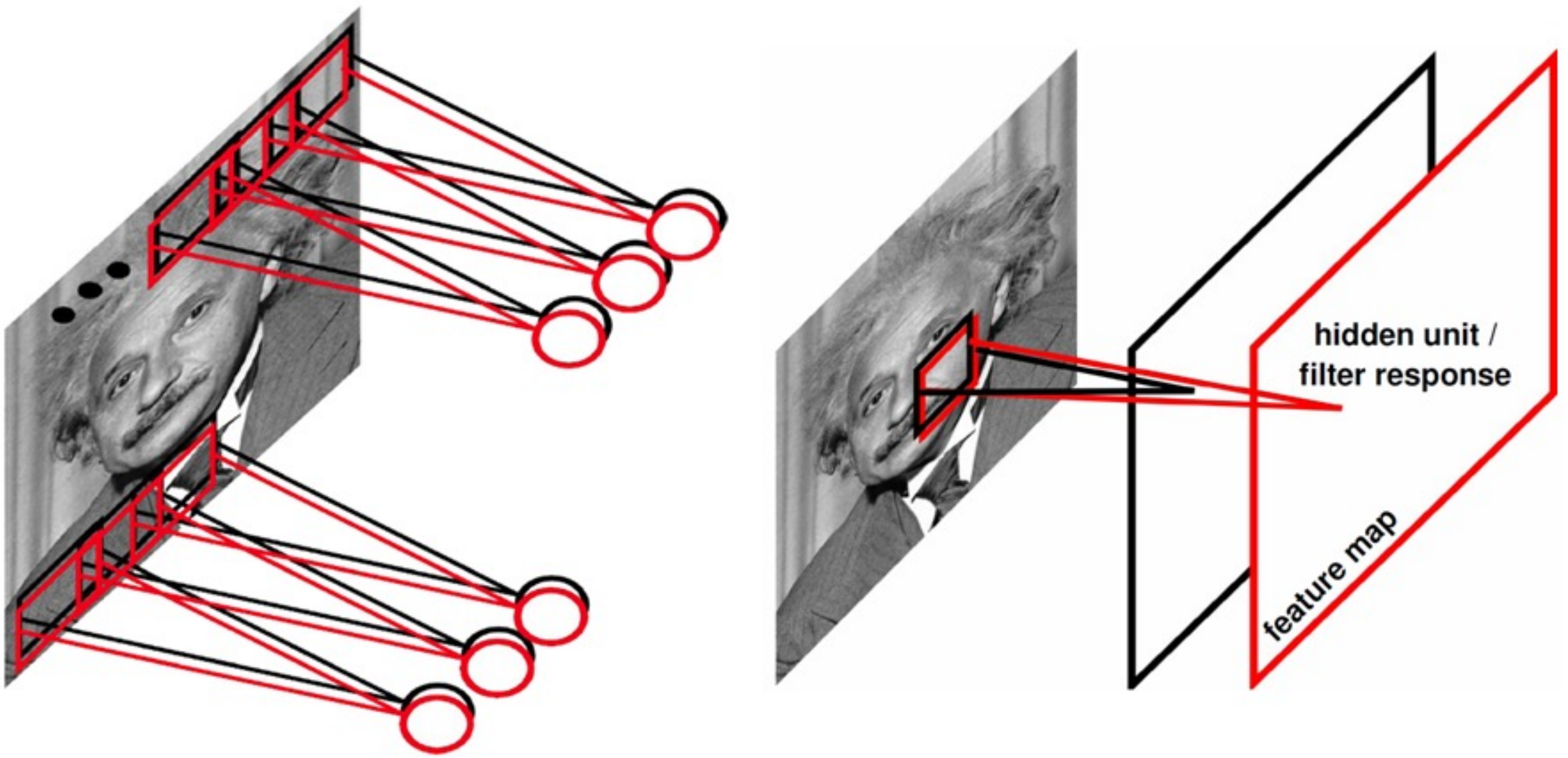
From one to many filters

1 filter => 1 feature map (corresponding to 1 visual pattern)
To detect spatial distributions of multiple visual patterns: Multiple filters
M filters => M feature maps!   Get richer description



From one to many filters

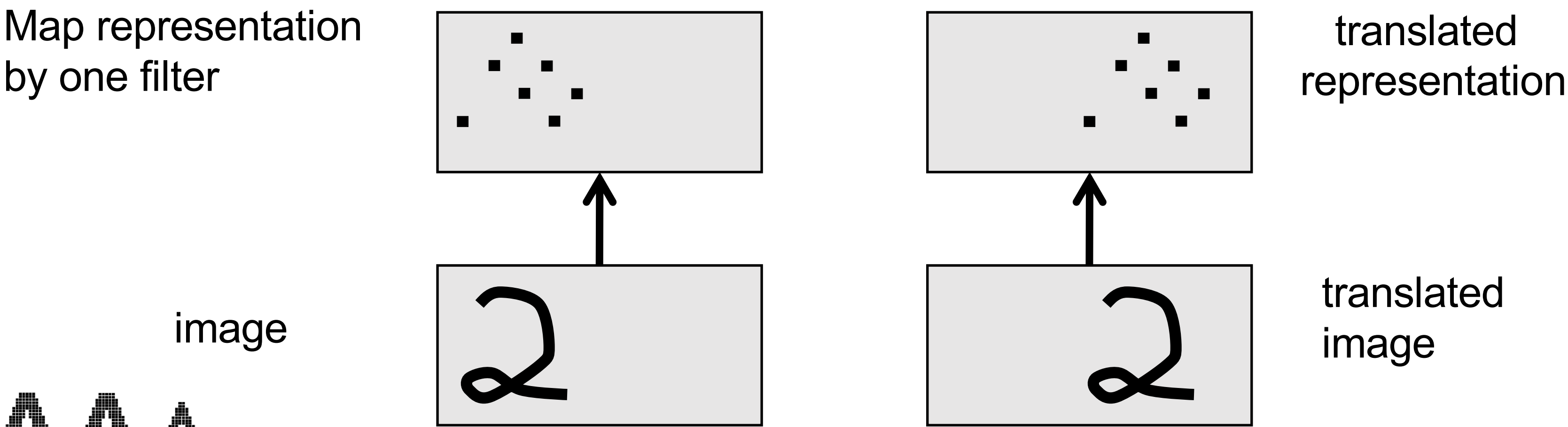
M filters => M feature maps

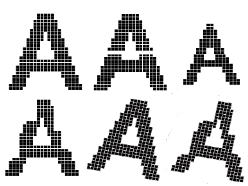


Rq: not many weights but many neurons! => memory issues will appear

What does replicating the feature detectors achieve?

- Equivariant activities (Hinton Ex): Replicated features do not make the neural activities invariant to translation. The activities are equivariant.





⇒How to get invariance to 2D spatial transformation of the input?

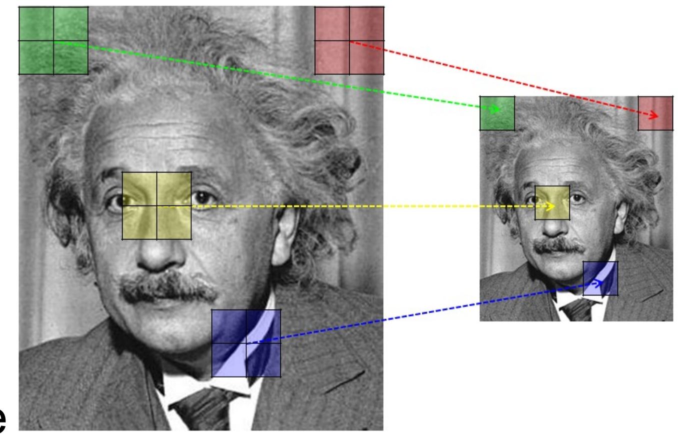
Getting (more) local Invariance

(local) spatial **POOLING** of the outputs of replicated feature detectors:

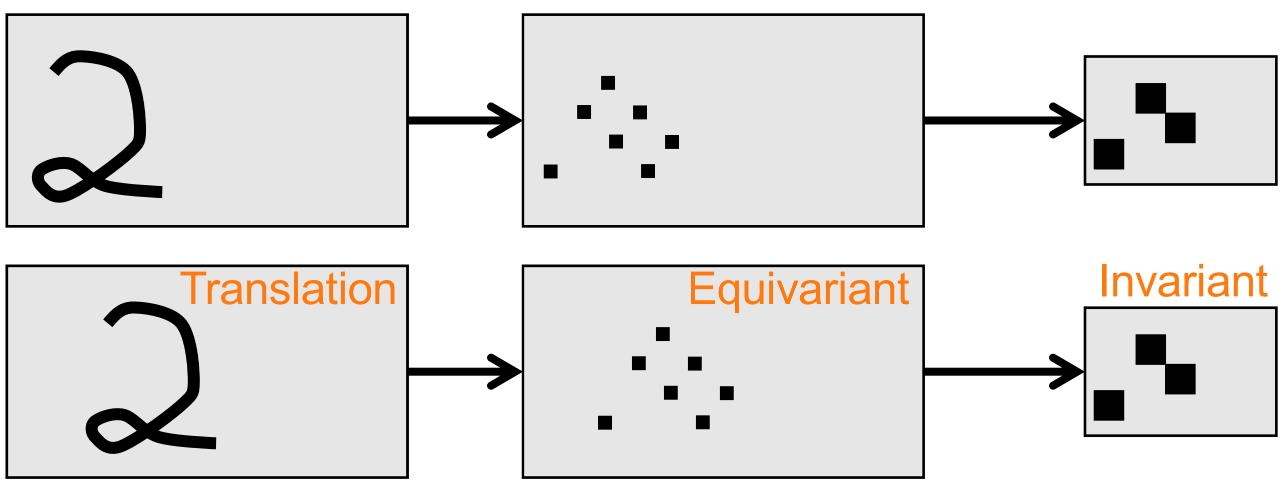
- Averaging neighboring replicated detectors to give a single output to the next level
- Max pooling: Taking the maximum in a neighboring

Get a small amount of translational invariance at each level

Reducing the number of inputs to the next layer of feature extraction

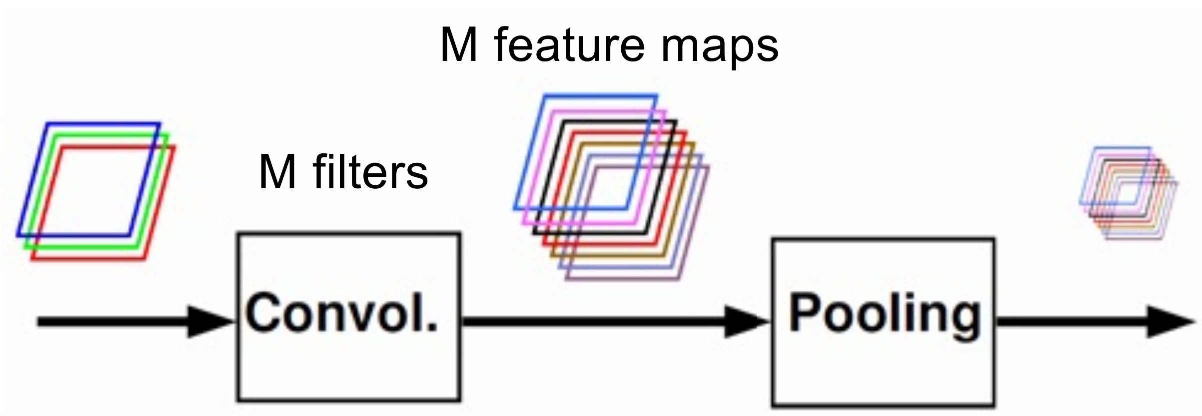


$$y_{ij} = \frac{1}{4}\left(x_{2i,2j} + x_{2i+1,2j} + x_{2i,2j+1} + x_{2i+1,2j+1}\right)$$



=> Stability OK (at least for local shift) for Convolutional Net!

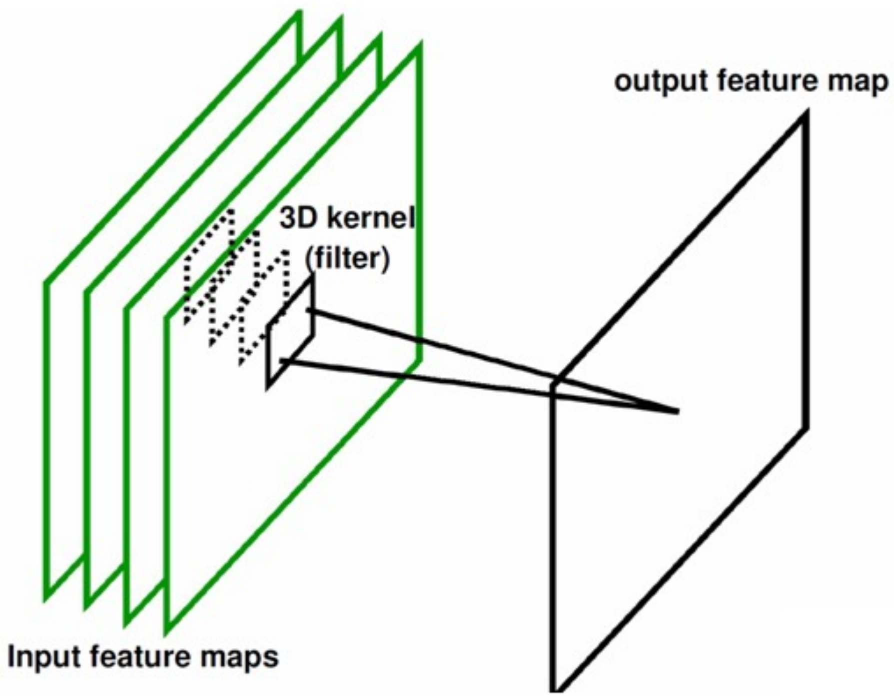
To sum up:

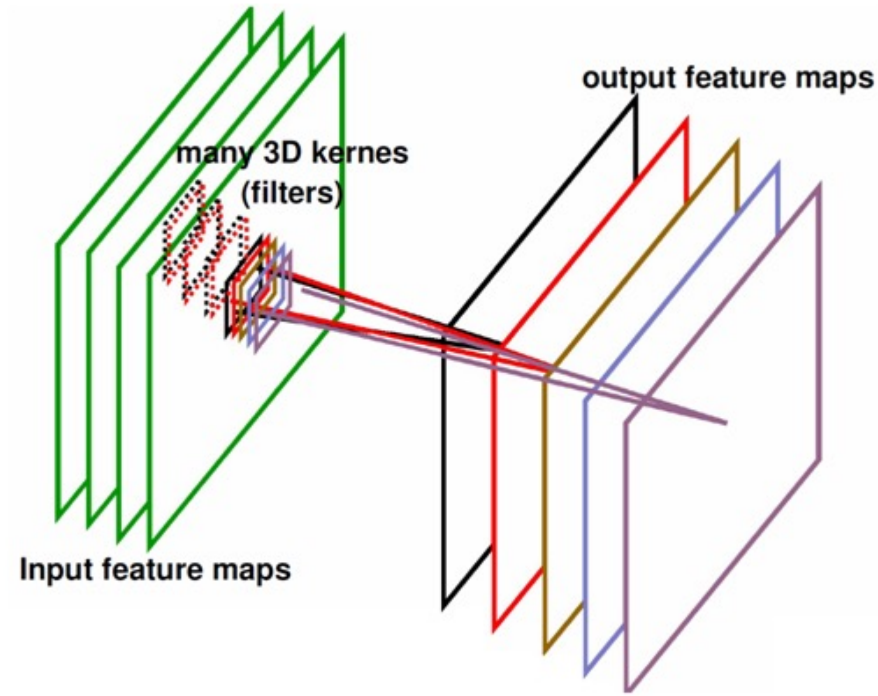


Color images: 3D kernels for filtering

mxnxd parameters per filter

Idem for any layer i to layer i+1

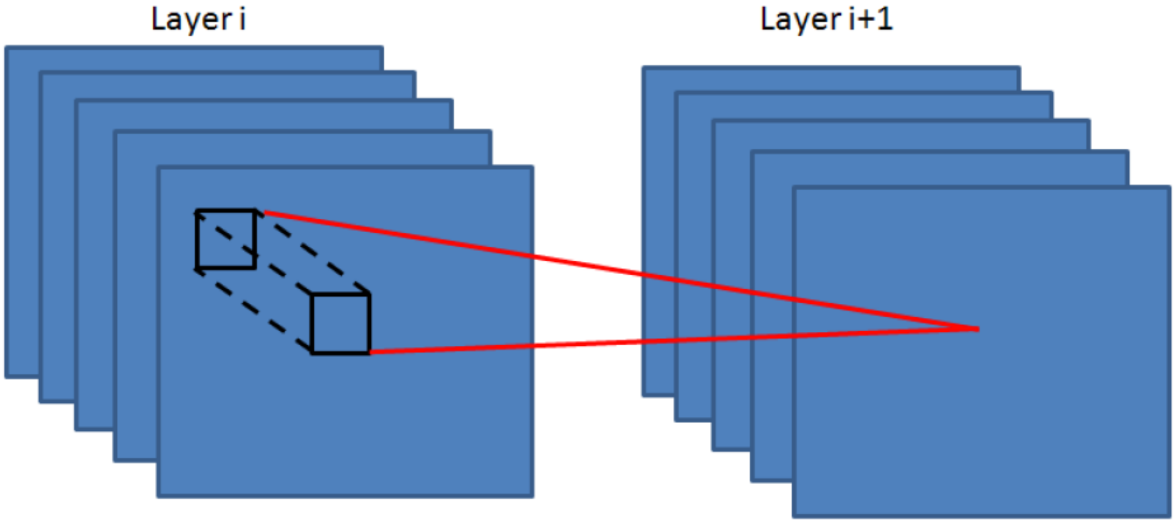




LCN: Local Contrast Normalization

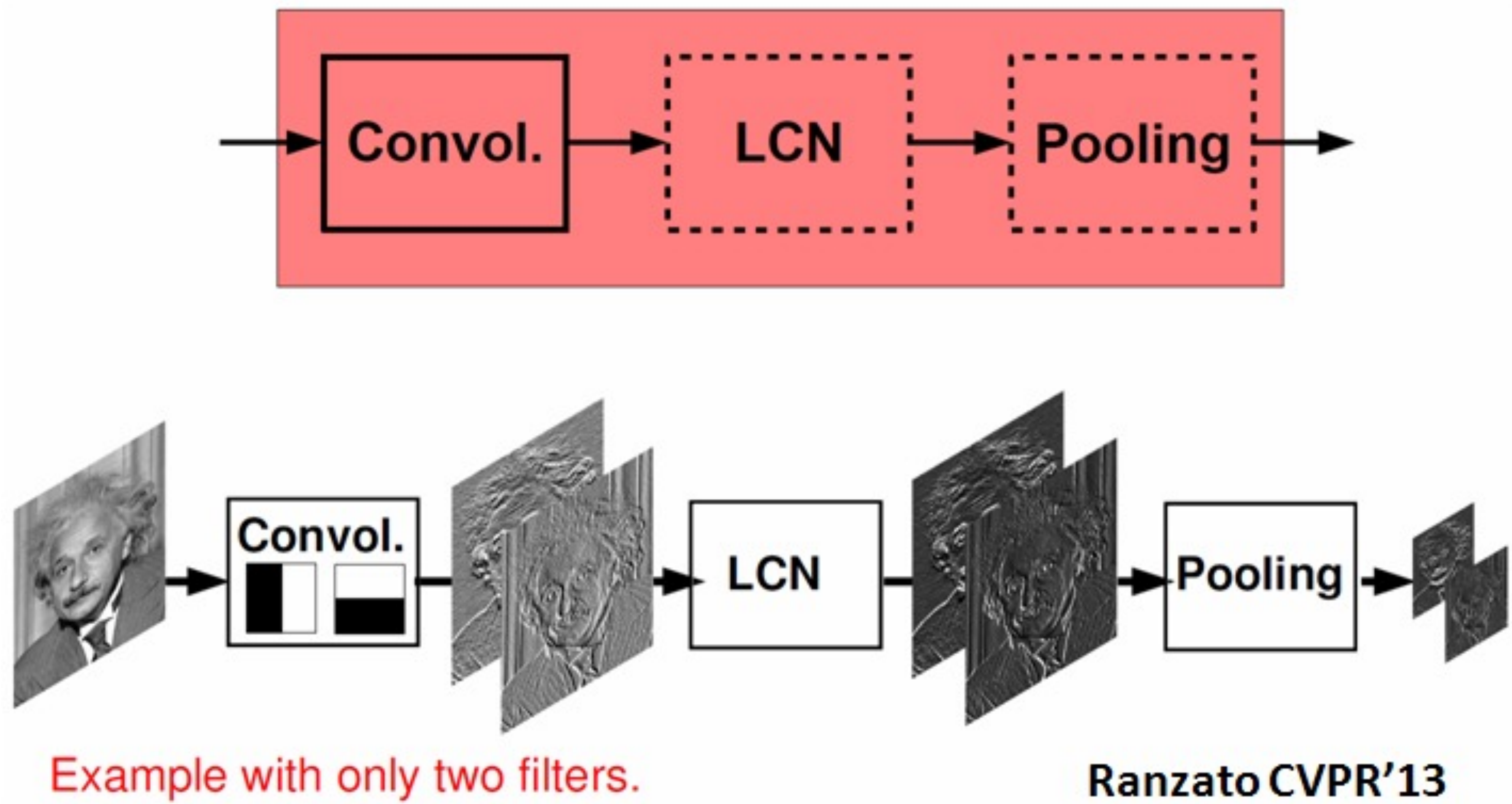
Normalization within a neighborhood along both spatial and feature dimensions

$$h_{i+1,x,y,k} = \frac{h_{i,x,y,k} - m_{i,N(x,y,k)}}{\sigma_{i,N(x,y,k)}}$$

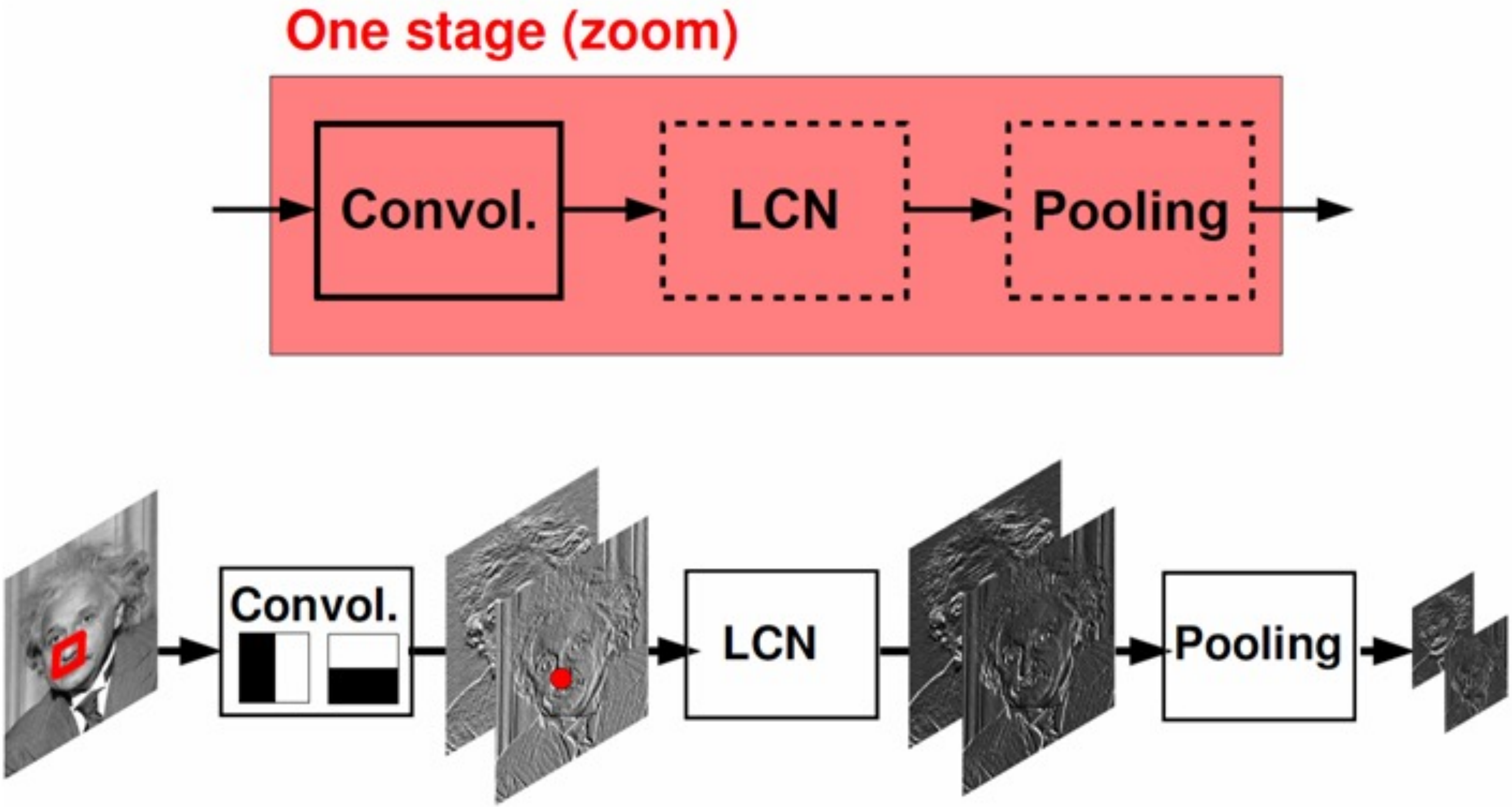


=> Very important for training large nets to carefully consider normalization within mini-batchs [S. Ioffe, C. Szegedy 2015]

1stage of convolutional neural networks



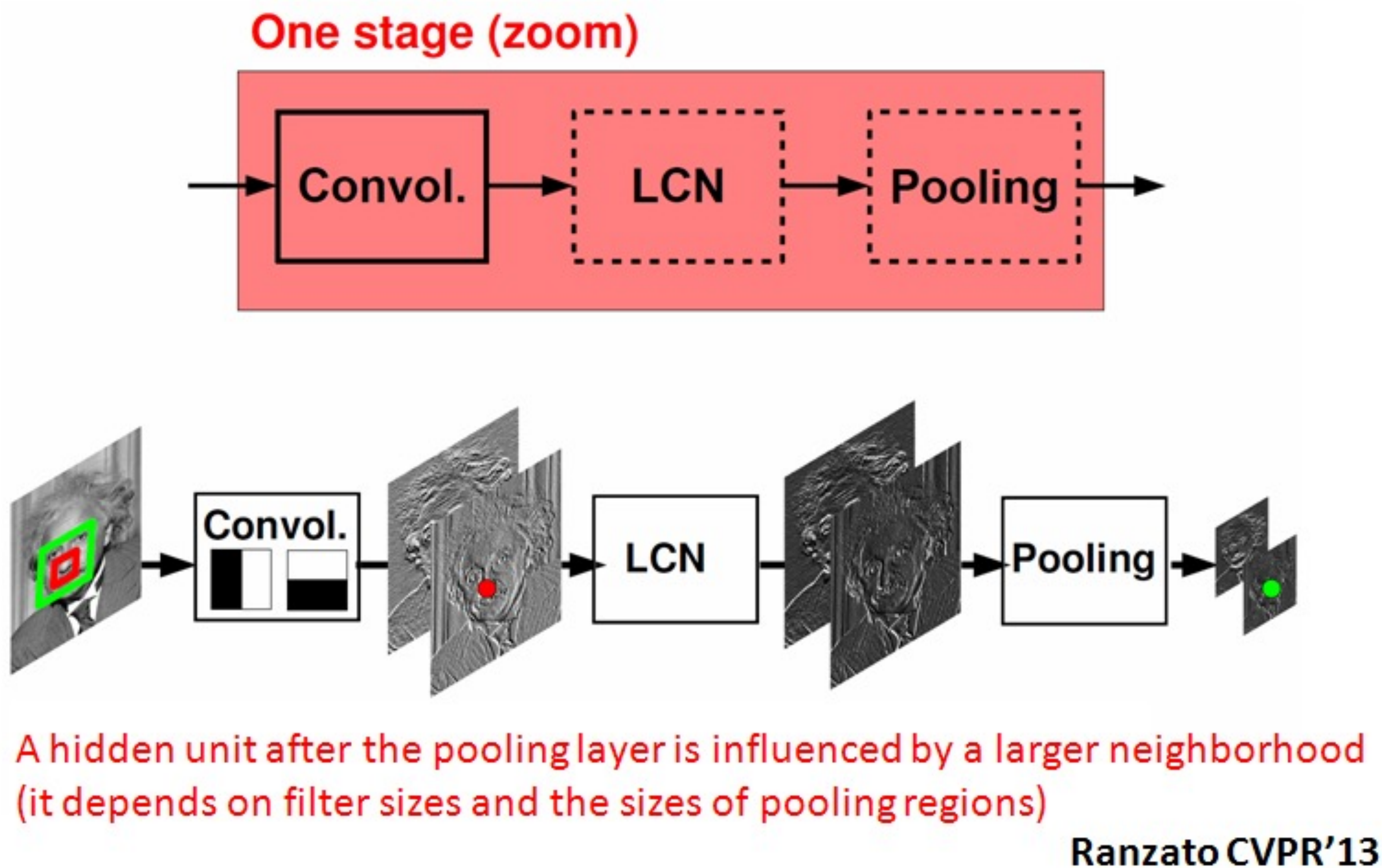
1stage of convolutional neural networks



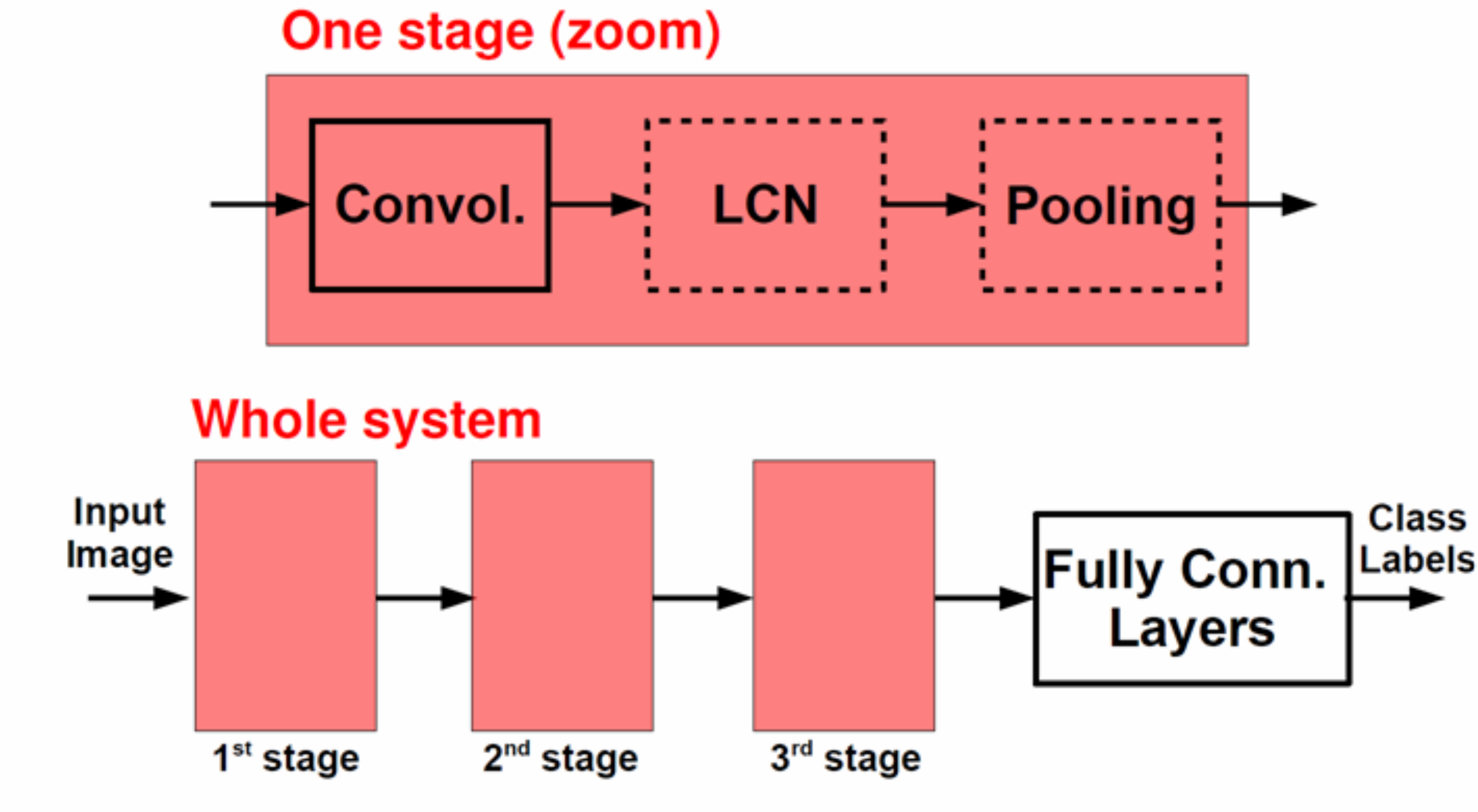
A hidden unit in the first hidden layer is influenced by a small neighborhood (equal to size of filter).

Ranzato CVPR'13

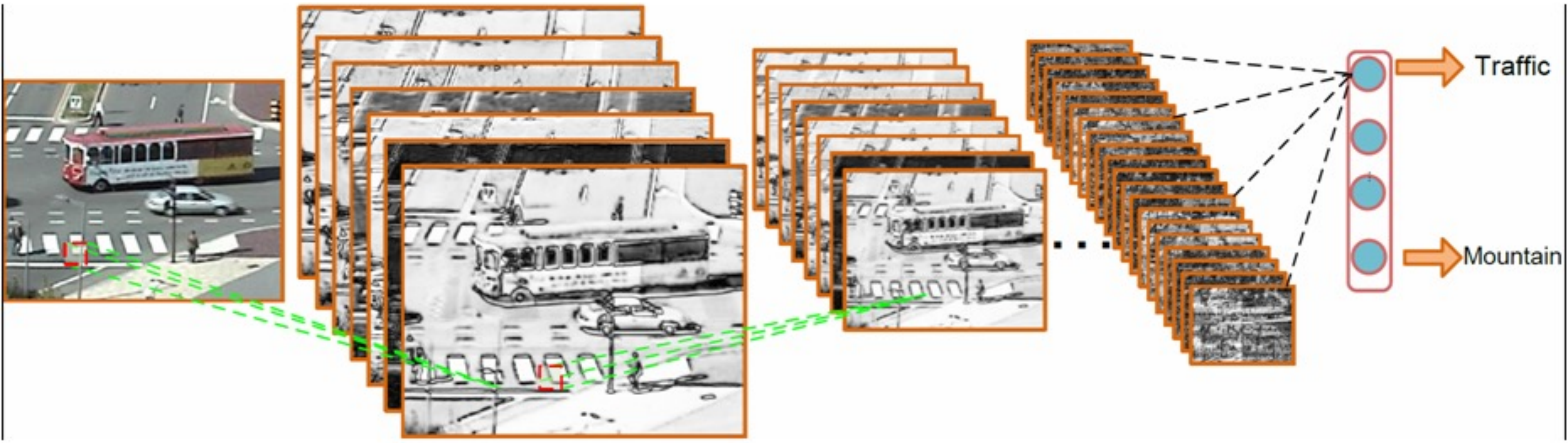
1stage of convolutional neural networks

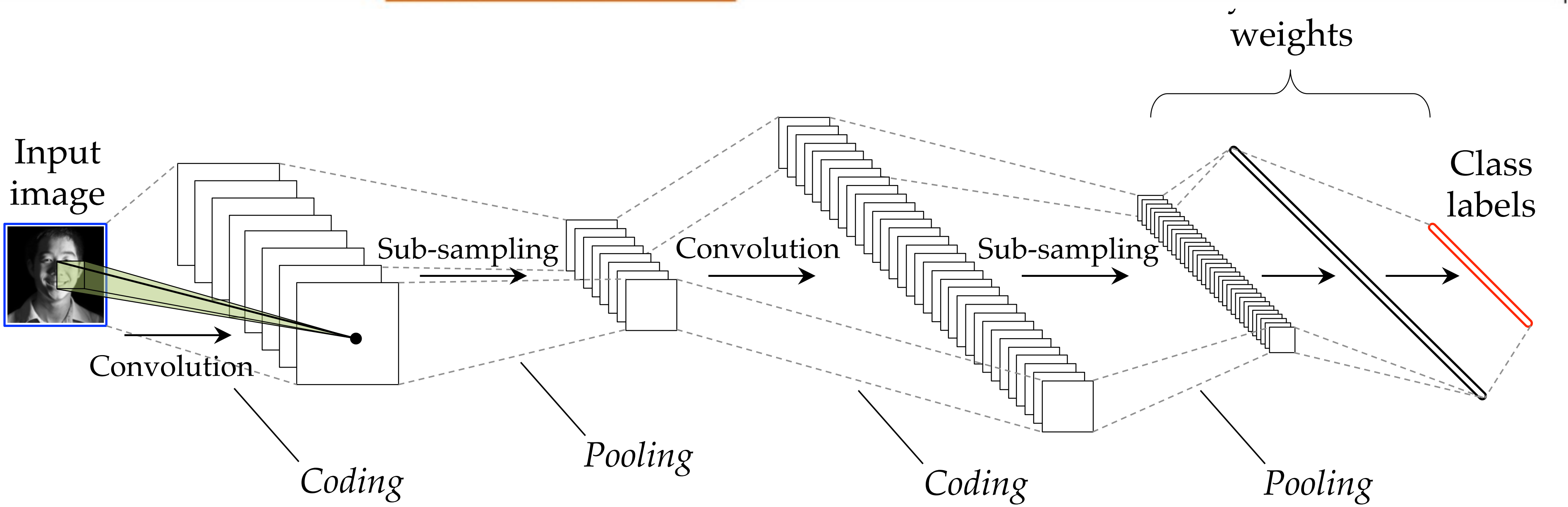


Full ConvNet architecture



To sum up: Full ConvNet architecture





To sum up: Full ConvNet architecture

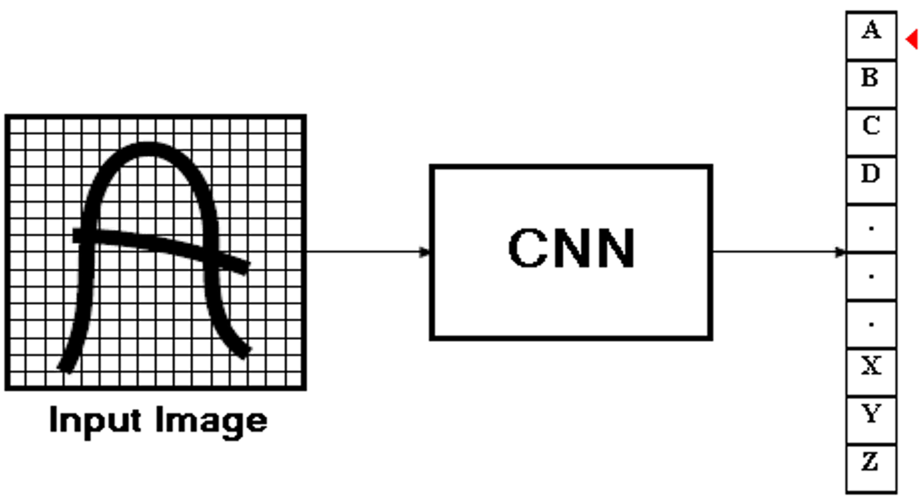
ConvNet (CNN): feed-forward network with

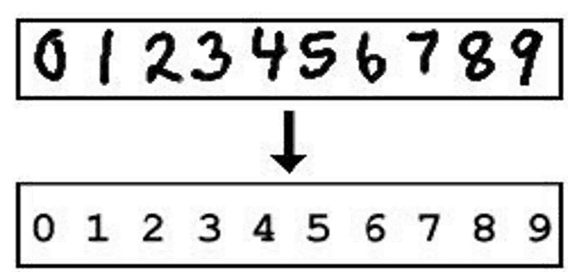
-- ability to extract topological properties from image

-- designed to recognize visual patterns

Working directly from pixel images with (no/minimal) preprocessing

Trained with back-propagation





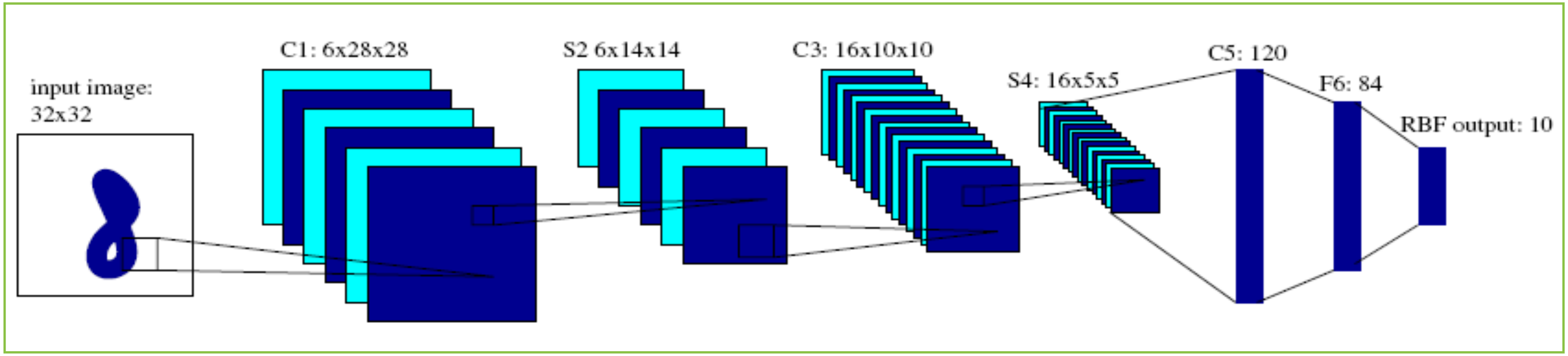
Outline
Convolutional Nets for visual classification

1. Recap MLP
2. Convolutional Neural Networks
3. **Examples: LeNet5, AlexNet**

Example: LeNet5

Introduced by Y. LeCun

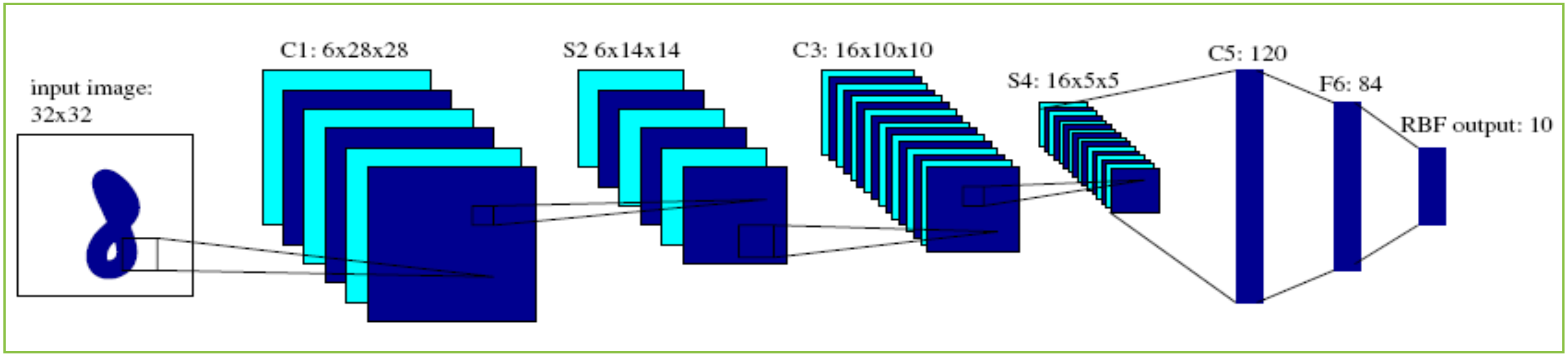
Raw image of 32 × 32 pixels as input



Example: LeNet5

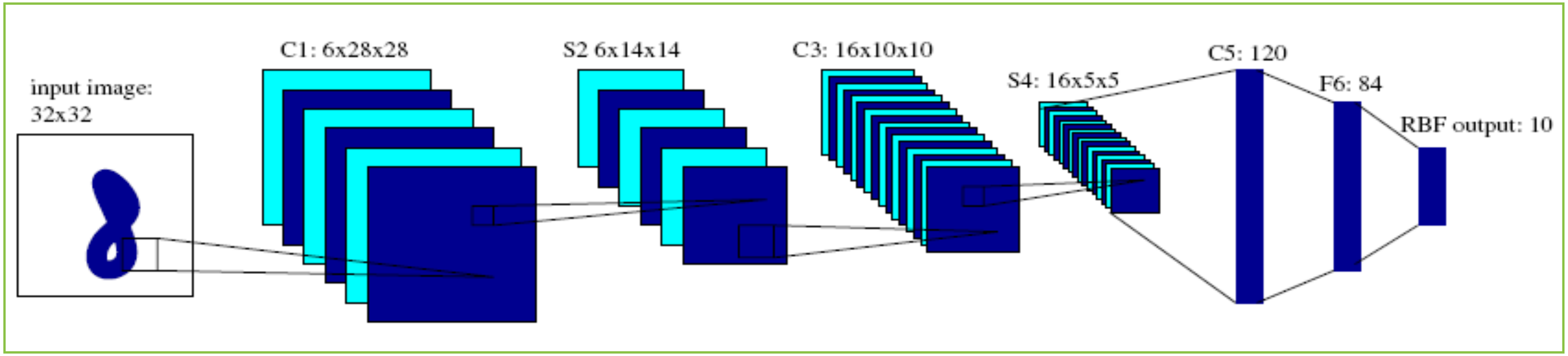
- C1,C3,C5 : Convolutional layer
- 5 × 5 Convolution matrix
- S2 , S4 : Subsampling layer = Pooling+stride s=2

  => Subsampling by factor 2
- F6 : Fully connected layer

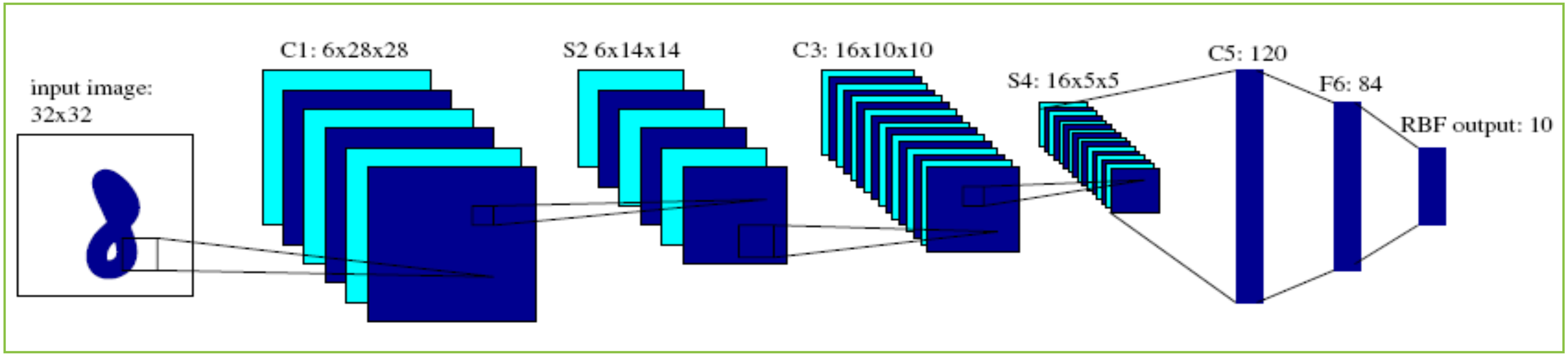


LeNet5

All the units of the layers up to F6 have a sigmoidal activation function

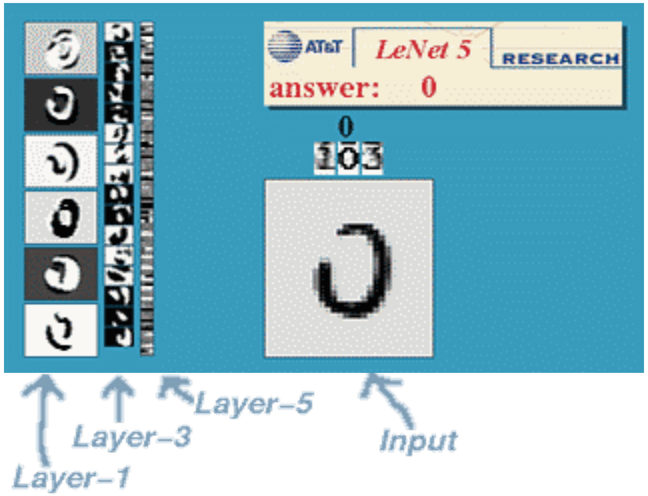


LeNet5

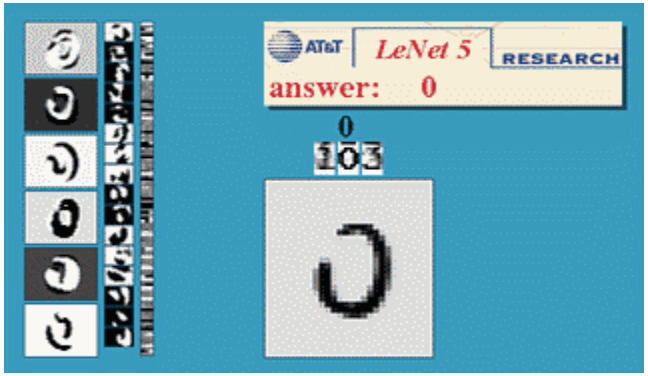


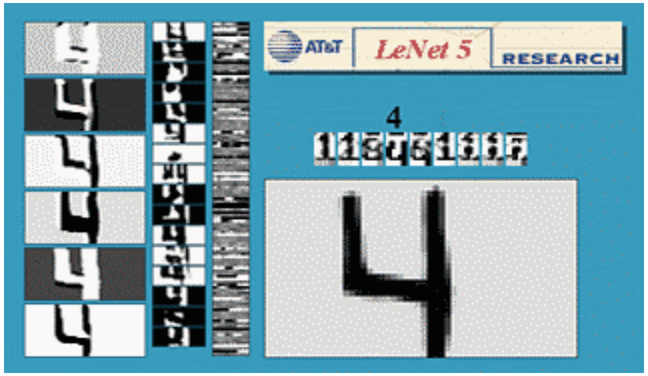
About 187,000 connections

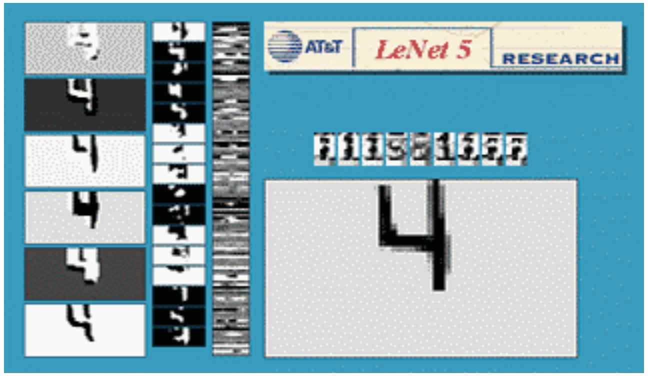
About 14,000 trainable weights

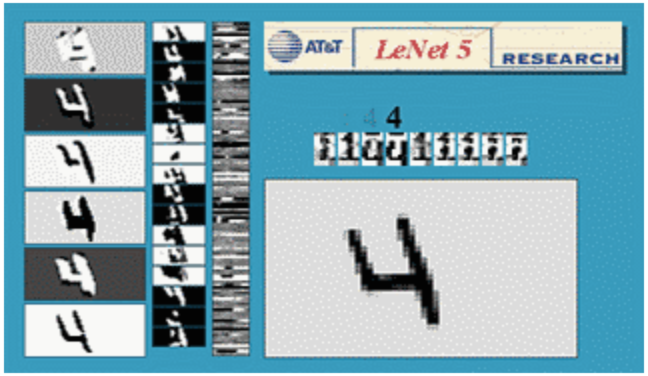


LeNet5 (@LeCun)

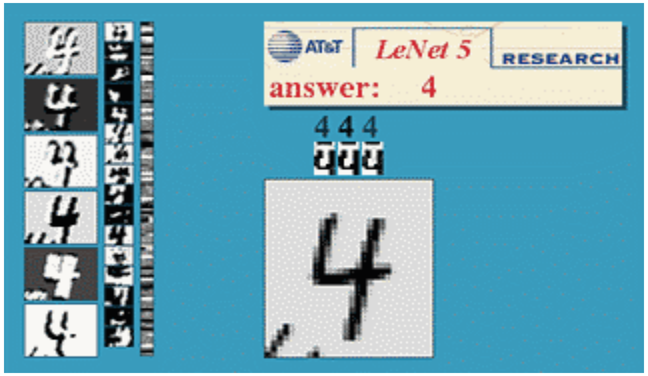


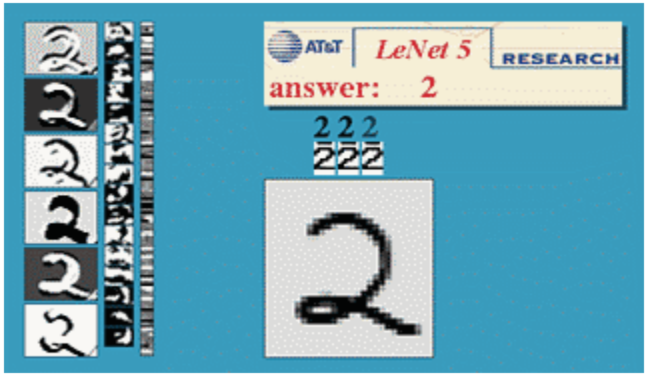


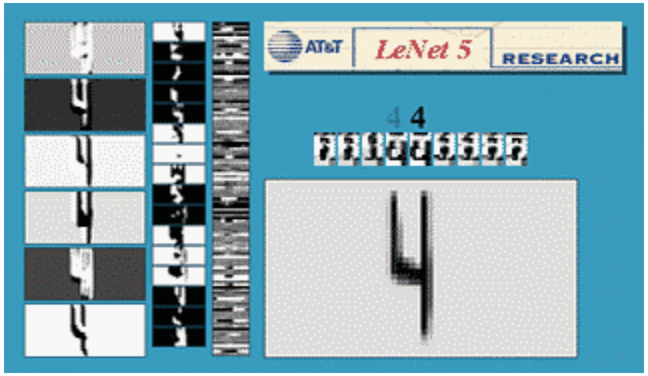


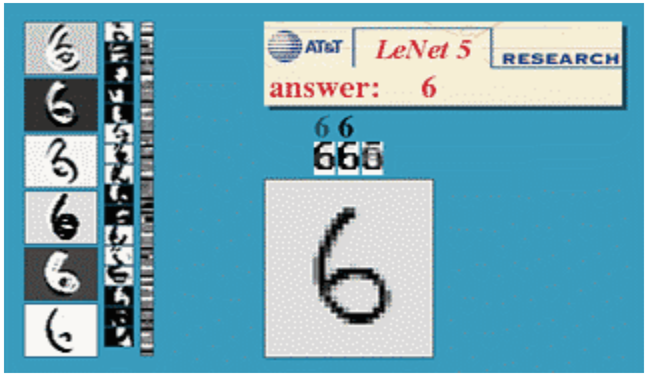


LeNet5 (@LeCun)



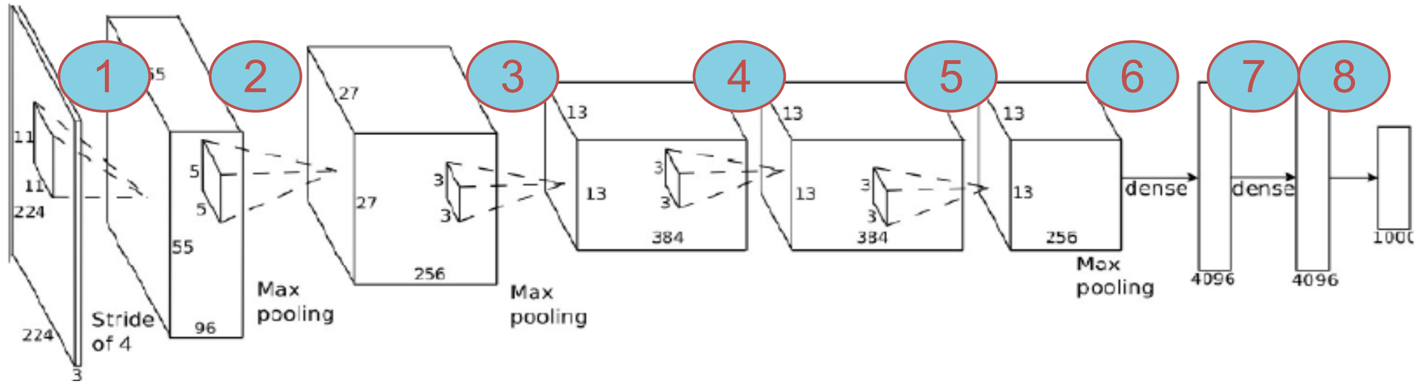




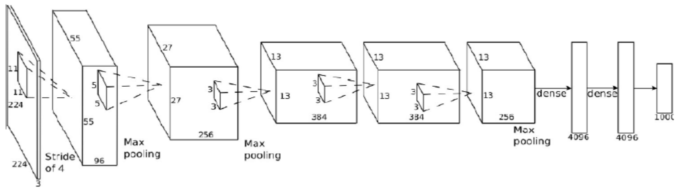


AlexNet 2012

- Same model as LeCun'98 but:
  - Bigger model (8 layers)
  - More data (10^6 vs 10^3 images)
  - GPU implementation (50x speedup over CPU)
  - Better regularization (DropOut)

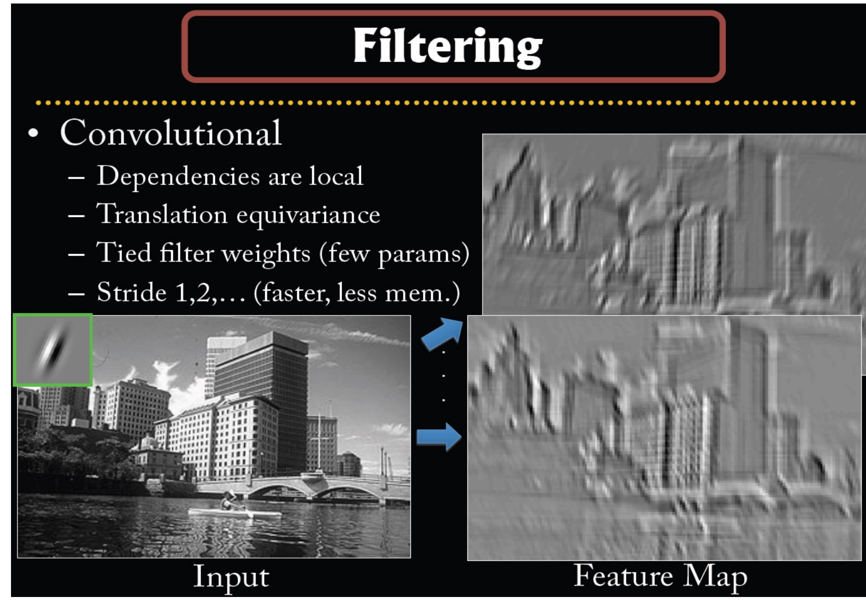


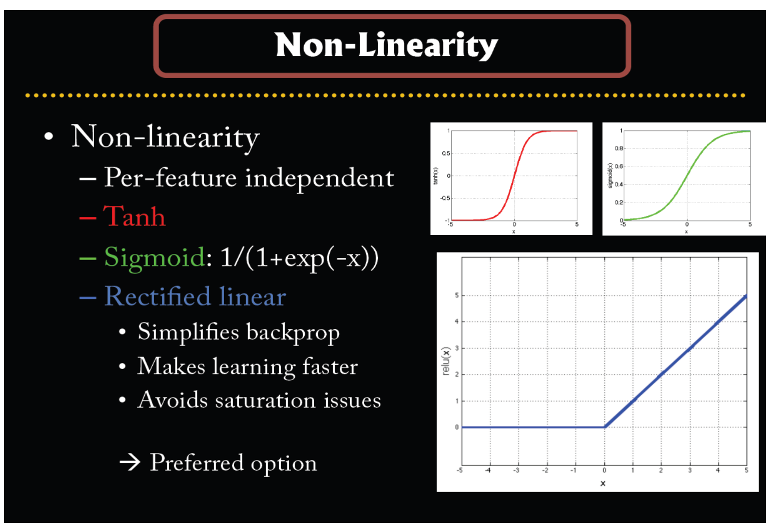
AlexNet 2012

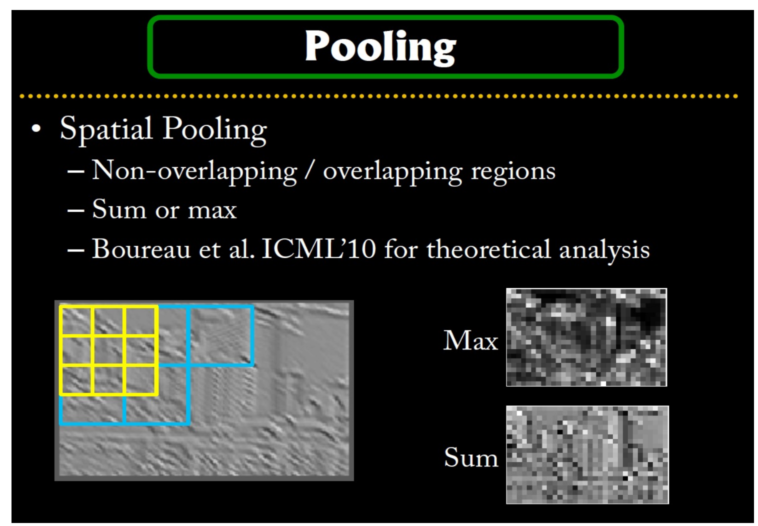


Same type of convnet with
- Filtering (convolution)
- Non-Linearity
- Pooling

8 layers but 224x224 input images => much biger model:
- 650,000 neurons
- 60,000,000 weights!



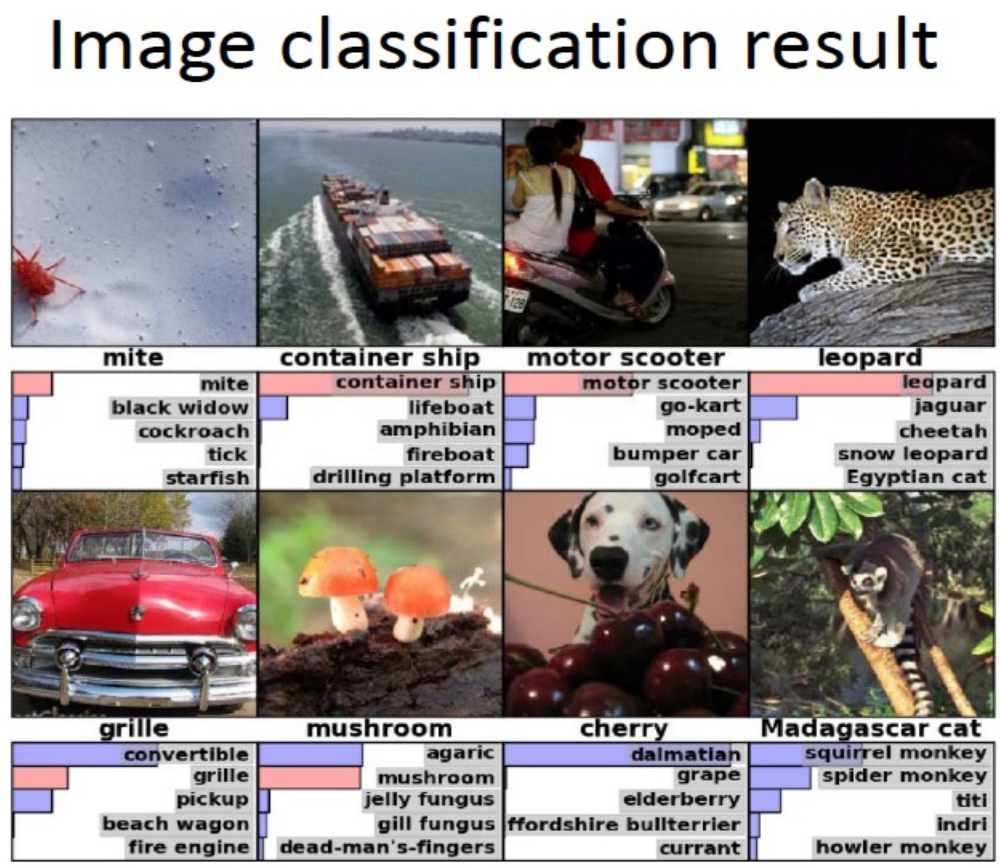




More data for supervised training

ImageNet 2012: the (deep) revolution

- 1.2 million labeled images
- 1000 classes
- Mono-class
- TOP5



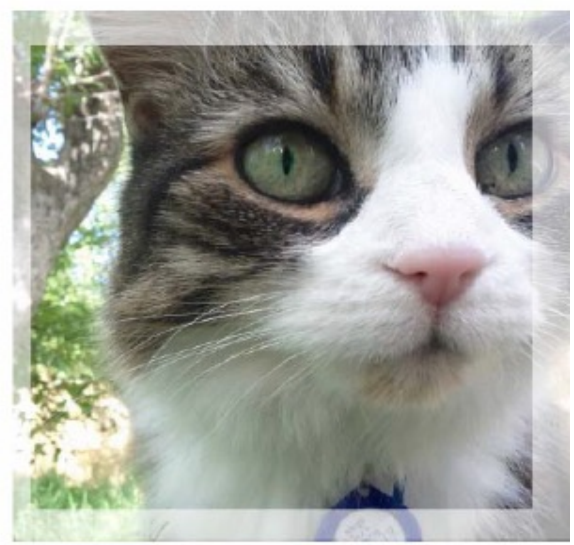
Learning the AlexNet

- Basics:
  - SGD, Backprop
  - Cross Validation
  - Grid search
- “New”
  - Huge computational resources (GPU)
  - Huge training set (1 million images)
  - Data augmentation - Pre-processing
  - Dropout
  - ReLu
  - *Contrast normalization*

Data Augmentation

lots of jittering, mirroring, and color perturbation of the original images generated on the fly to increase the size of the training set

Crop, flip,.. in train AND in test

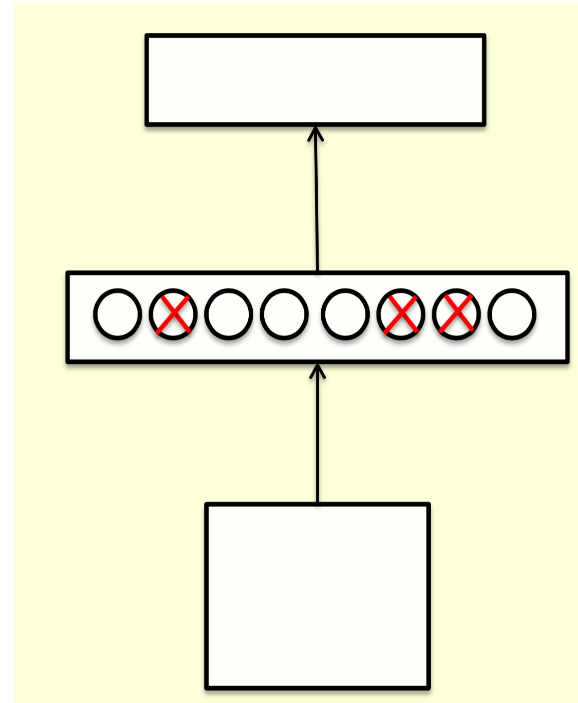


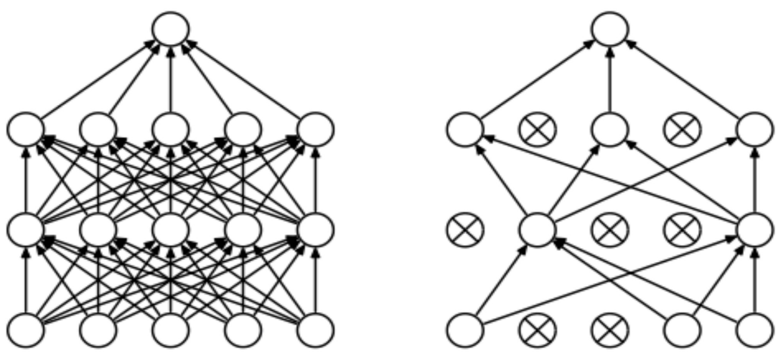
Dropout: an efficient way to average many large neural nets

For each training example, randomly omit each hidden unit with probability 0.5

Due to sharing of weights, model strongly regularized

- Pulls the weights towards what other models want.
- Better than L2 and L1 regularization that pull weights towards zero





Standard Neural Net   After applying dropout.

@Hinton, NIPS 2012

Dropout: what do we do at test time?

Option 1:

Sample many different architectures and take the geometric mean of their output distributions

Option 2: (Faster way)

Use all the hidden units

but after **halving their outgoing weights**

Rq: In case of single hidden layer, this is equivalent to the geometric mean of the predictions of all models

For multiple layers, it's a pretty good approximation and its fast

How well does dropout work?

Significantly improve generalization:

- For very deep nets, or at least when there are huge fully connected layers (eg. AlexNet first FC layer, VGG next, …)
- Less useful for fully convolutional nets

Useful to prevent feature co-adaptation (feature only helpful when other specific features present)

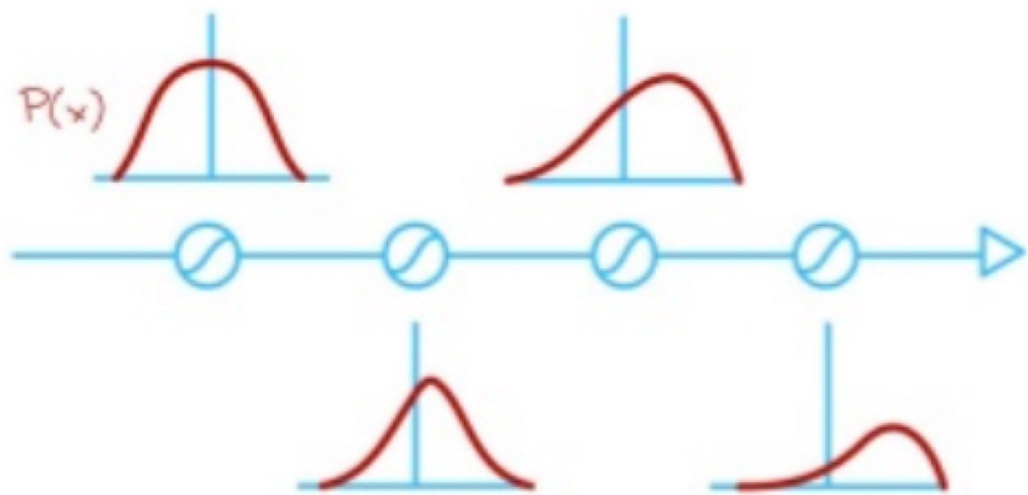
Later in course

⇒**Dropout as a Bayesian Approximation**

⇒**Representing Model Uncertainty in Deep Learning**

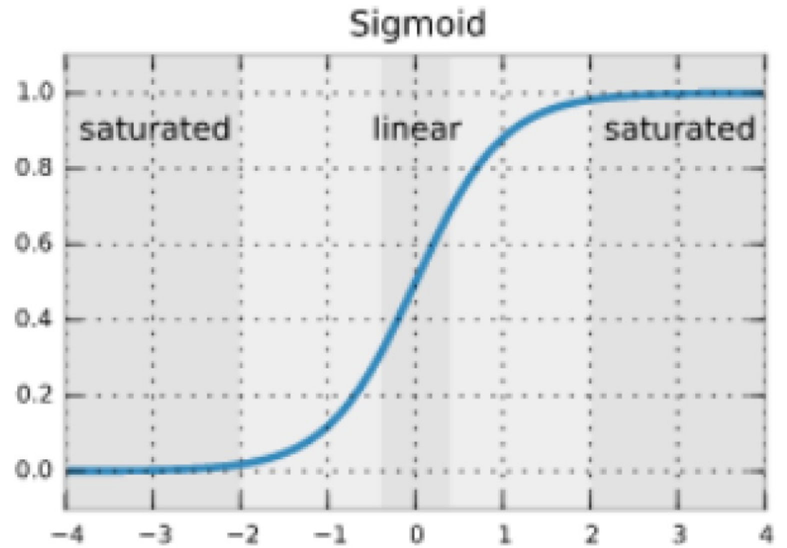
Batch Normalization (BN) [Ioffe and Szegedy, 2015]

- Problem of covariate shift
  - Distribution of activations' ranges across the network's depth uncontrolled
  - Training sensible to initialization



- **BN Idea:** control the distribution of activations across the network's depth

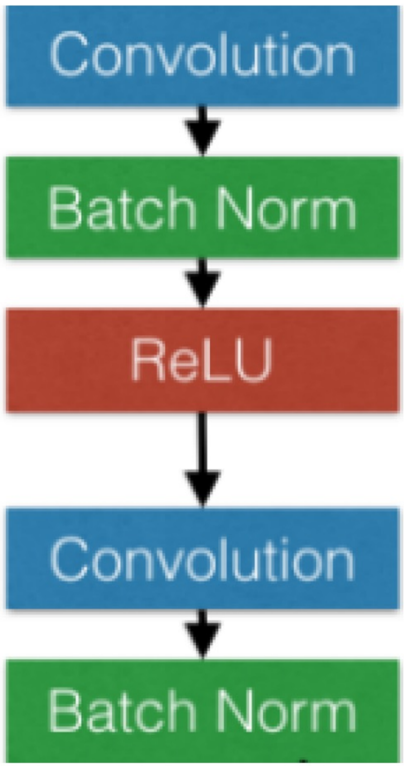
Batch Normalization (BN) [Ioffe and Szegedy, 2015]



- **Normalize input feature distribution** $\sim \mathcal{N}(0,1)$
  - Normalization across each mini-batch:
  - $\mu_B = \frac{1}{N}\sum_{i=1}^{N} x_i$
  - $\sigma_B^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_B)^2$
  - $\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B + \epsilon}$ - ϵ for numerical stability
- **Is input feature distribution** $\sim \mathcal{N}(0,1)$ **good idea?**
  - Activation may not ever "saturate", *e.g.* sigmoid or tanh
  - Keeping in linear regime: depth useless, $\sim$ global linear model

Batch Normalization (BN) [Ioffe and Szegedy, 2015]

- Scale and shift: $y_i = \gamma \hat{x}_i + \beta$, (γ, β) trained
- Apply after FC / conv and before non-linearity



$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

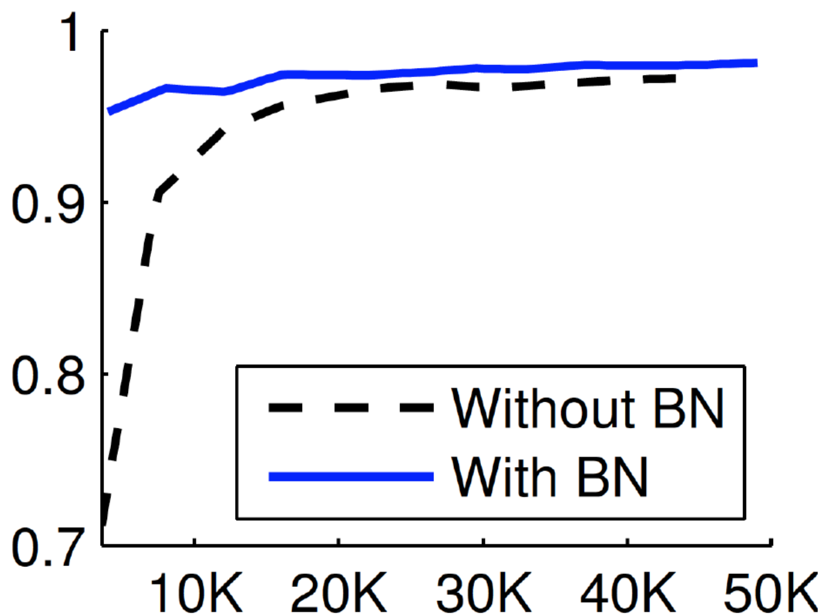
$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

- Batch Normalization differentiable

Batch Normalization (BN) [Ioffe and Szegedy, 2015]

- **Applying BN at test time?**
  $\Rightarrow$ Use train set statistics
- **BN Strengths**
  - **Faster training convergence vs covariate shift**
  - **Regularization & generalization**
    $\Rightarrow$ **better performances**
- **BN Conclusion:** especially important for very deep models, *e.g.* ResNet (see next)



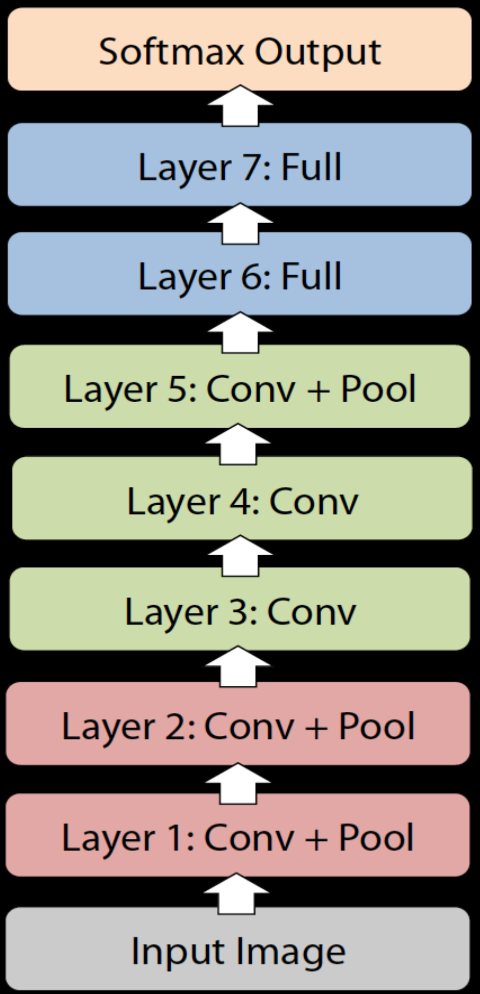
AlexNet 2012

Ablation study

1. **Number of layers**
2. ***Tapping off features at each layer***
3. **Transfo Robustness vs layers**

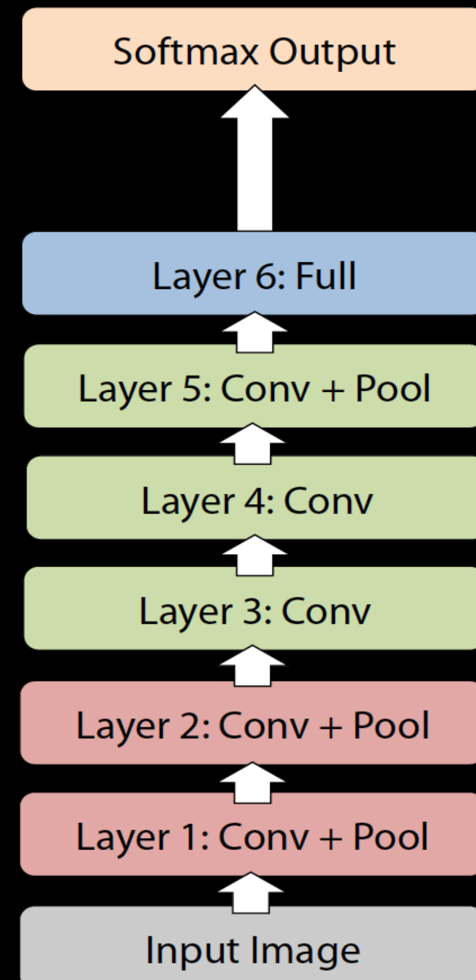
Architecture of Krizhevsky et al.

- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error
- Our reimplementation:
  18.1% top-5 error



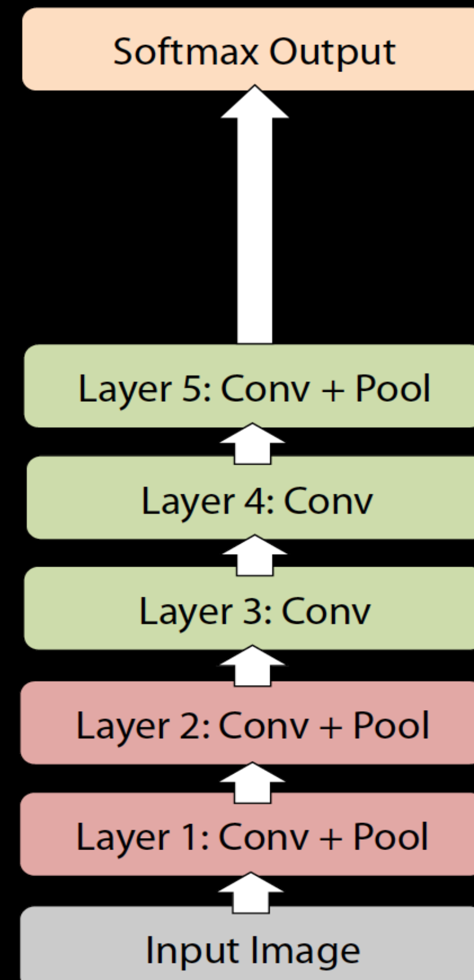
Architecture of Krizhevsky et al.

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



Architecture of Krizhevsky et al.

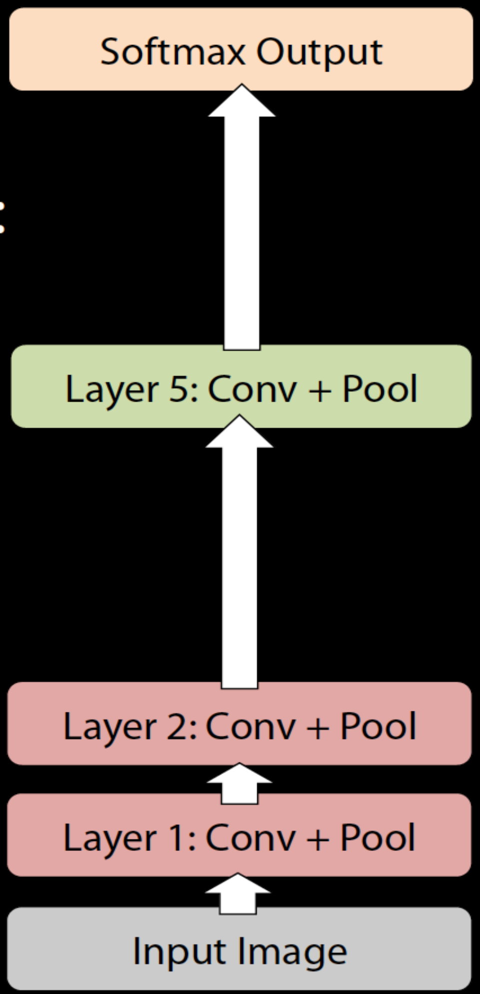
- Remove both fully connected layers
  - Layer 6 & 7
- Drop ~50 million parameters
- 5.7% drop in performance



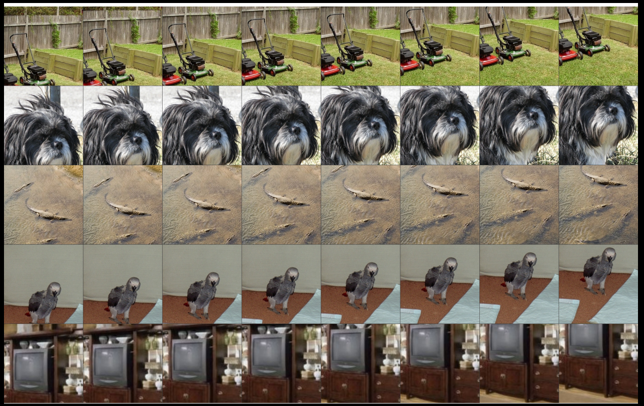
Architecture of Krizhevsky et al.

- Now try removing upper feature extractor layers & fully connected:
  - Layers 3, 4, 6 ,7
- Now only 4 layers
- 33.5% drop in performance

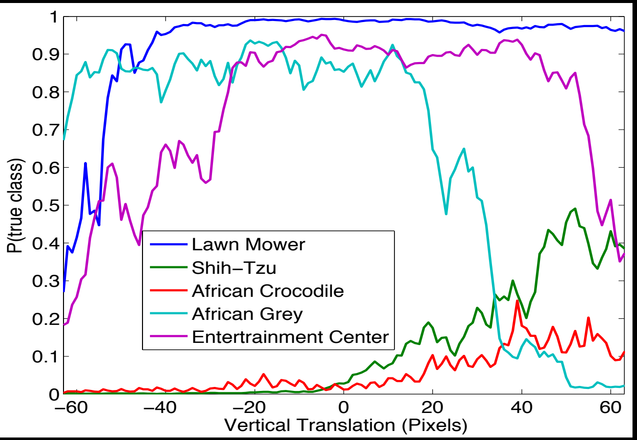
→Depth of network is key



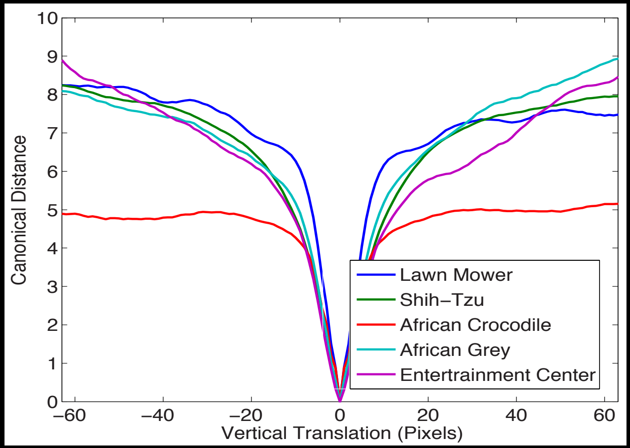
Translation (Vertical)



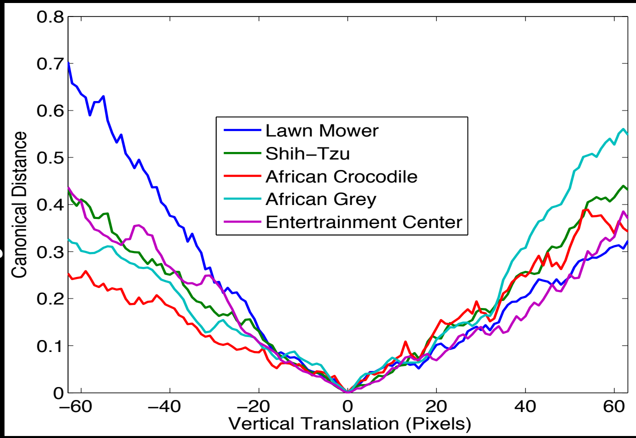
Output



Layer 1

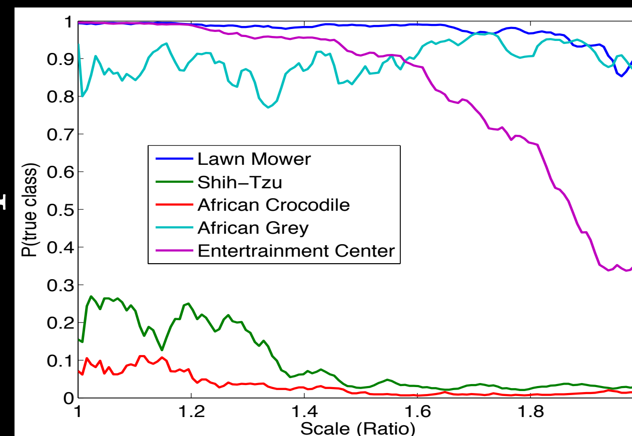


Layer 7

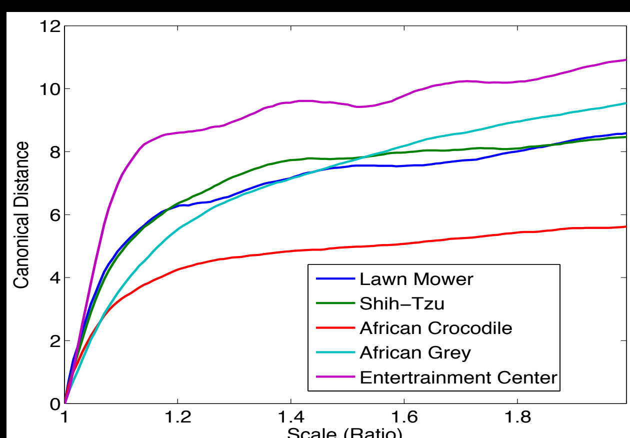


Scale Invariance

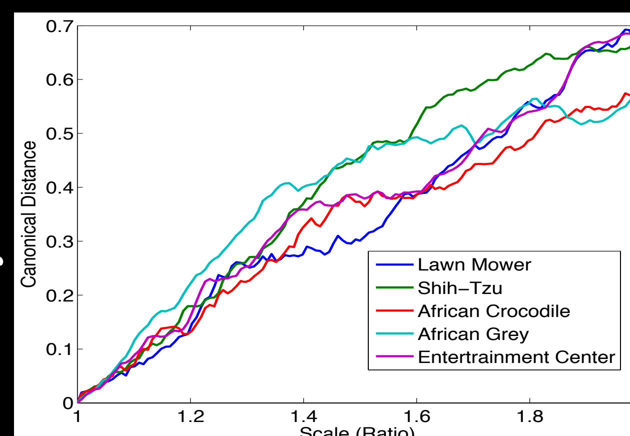
Output



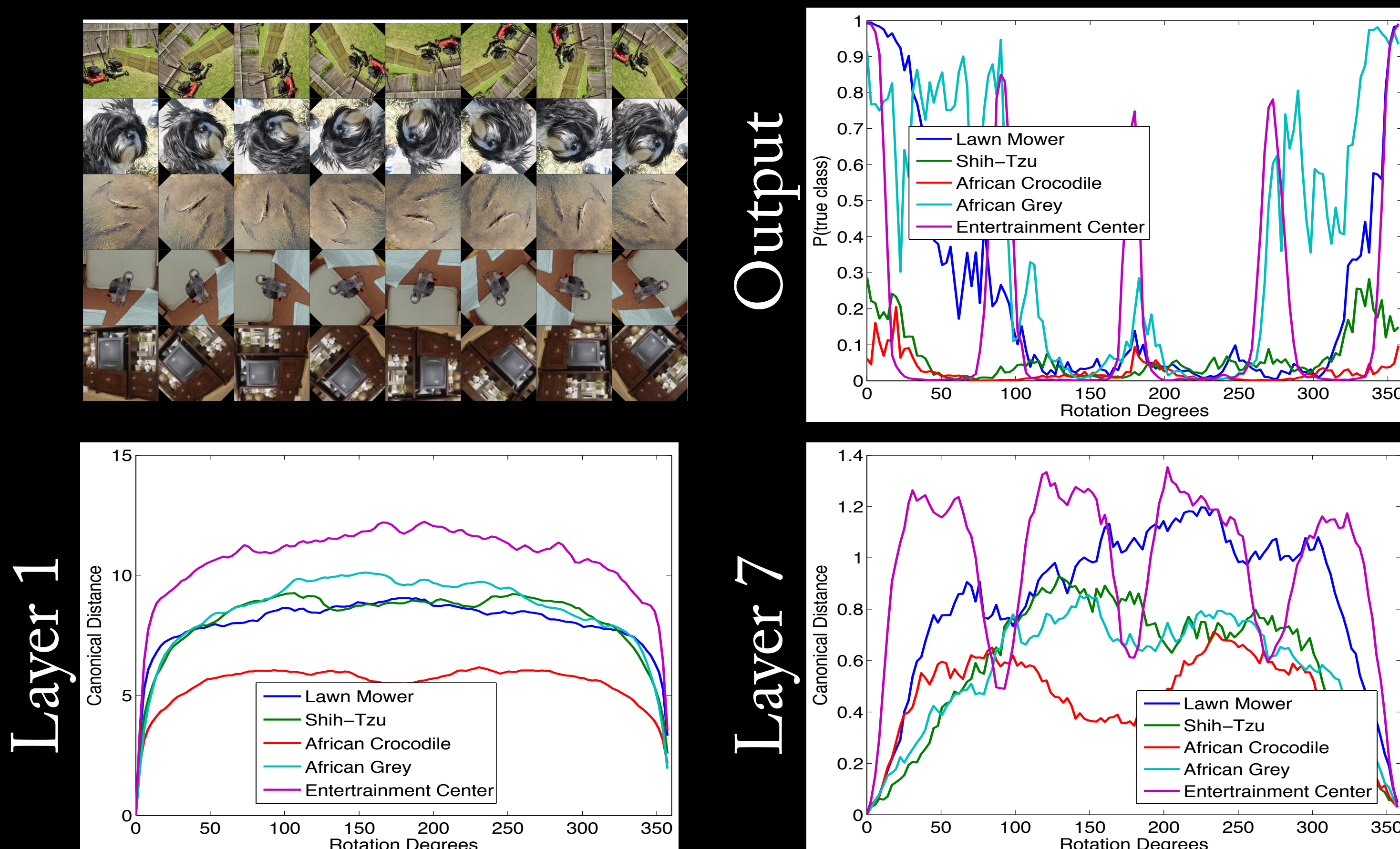
Layer 1



Layer 7

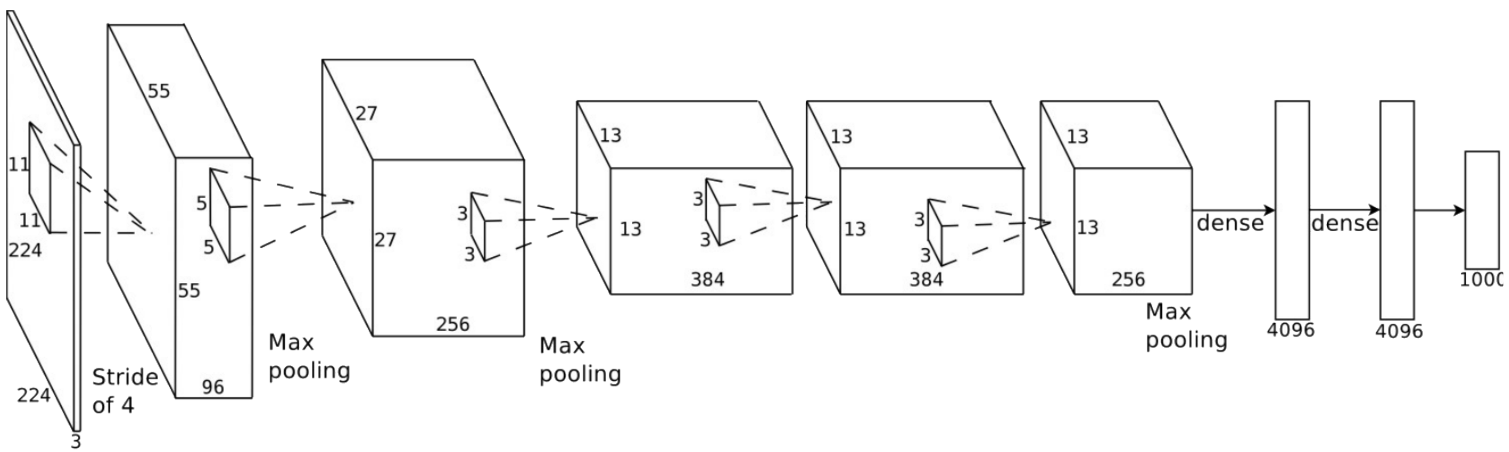


Rotation Invariance



Deep ConvNets for image classification

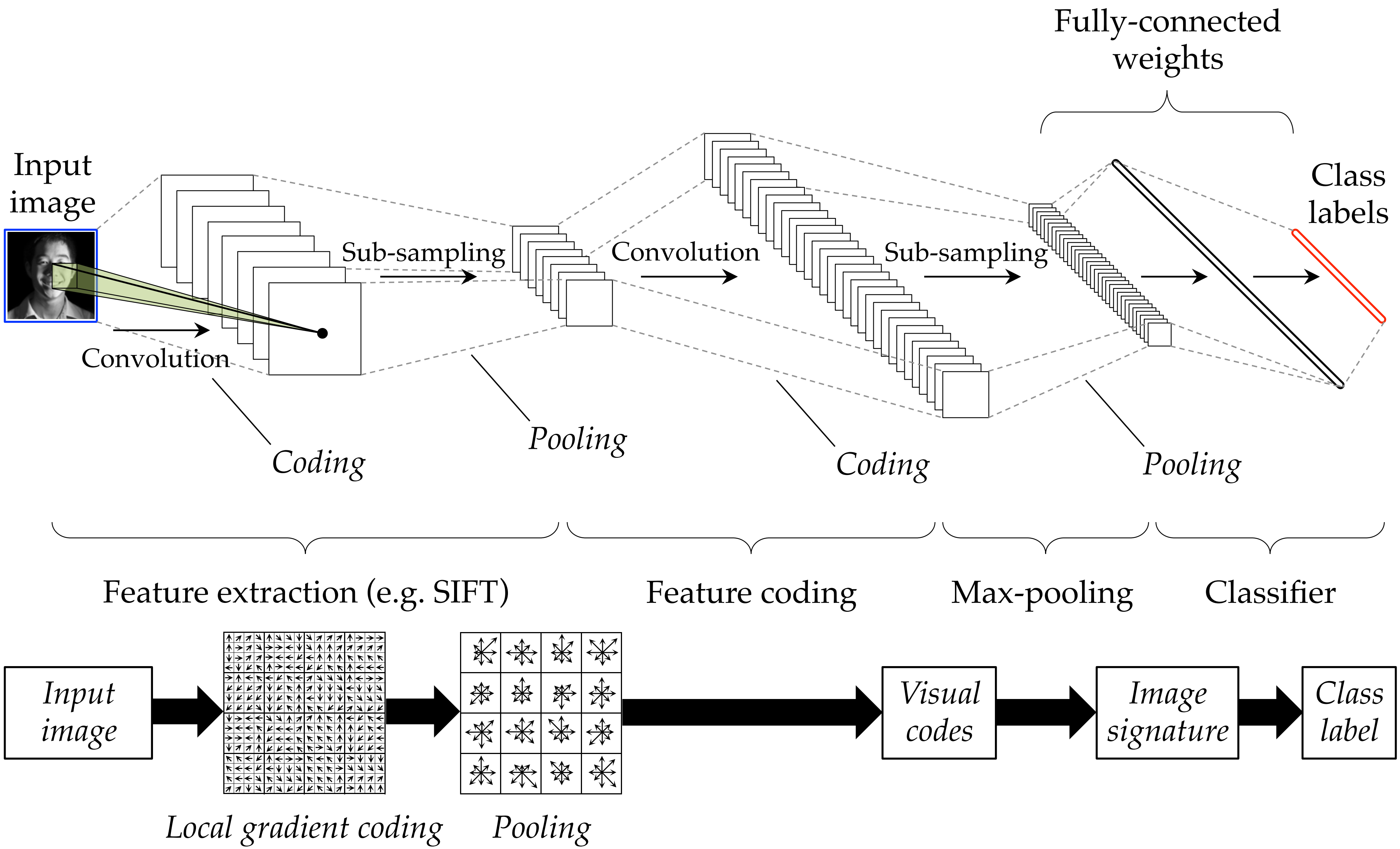
- **AlexNet** 8 layers, 62M parameters



Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton
ImageNet Classification with Deep Convolutional Neural Networks.
In *NIPS*, 2012.

Extra

Comparison BoW / deep CNN

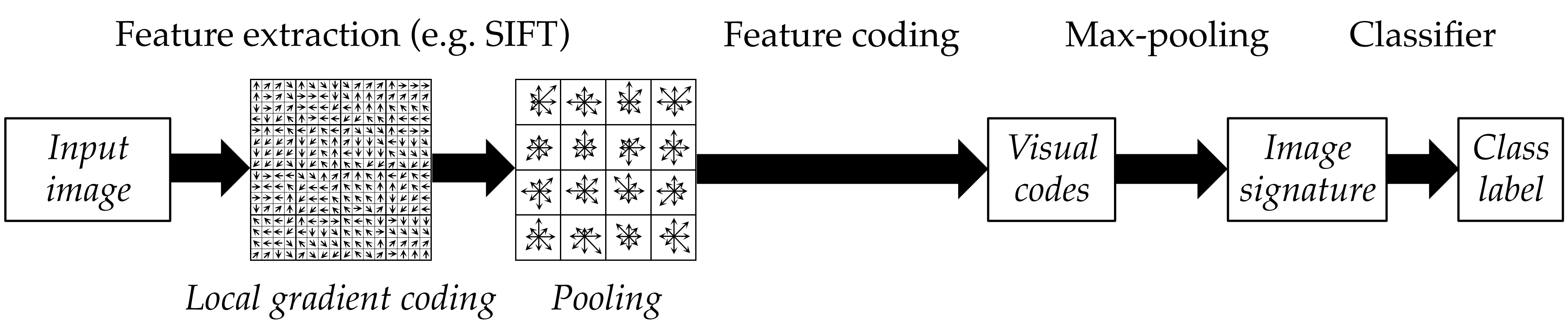


Comparison BoW / CNN deep

1. Regular grid + gradient detection (SIFT) => bank of 8 linear filters (convolution) +Winner Take All (inter-maps) => 8 maps
2. Local histogram SIFT => spatial local sum pooling inside maps on a fixed grid 4x4 => 16x8=128 (smaller) maps
3. BoW Coding = projection on M vectors (visual dico elts) => a bank of M linear filters of size 4x4x8 (=1x1x128 convolution) => M maps
4. BoW Pooling => global pooling on each map => M scalar values = 1 vector representation BoW (extension: SPM)
5. Classification (SVM) => Fully connected layers

BoW = Conv1+pooling(loc)+Conv2+pooling(global)+Fconnected

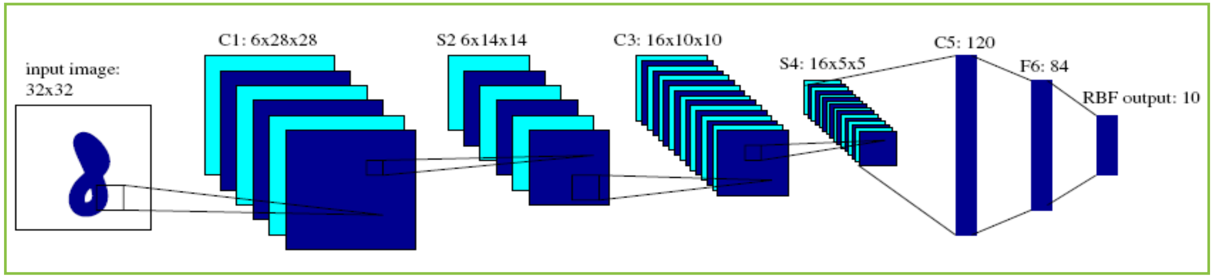
- Handcrafted+unsupervised vs. end-to-end supervision
- Light deep vs. very deep



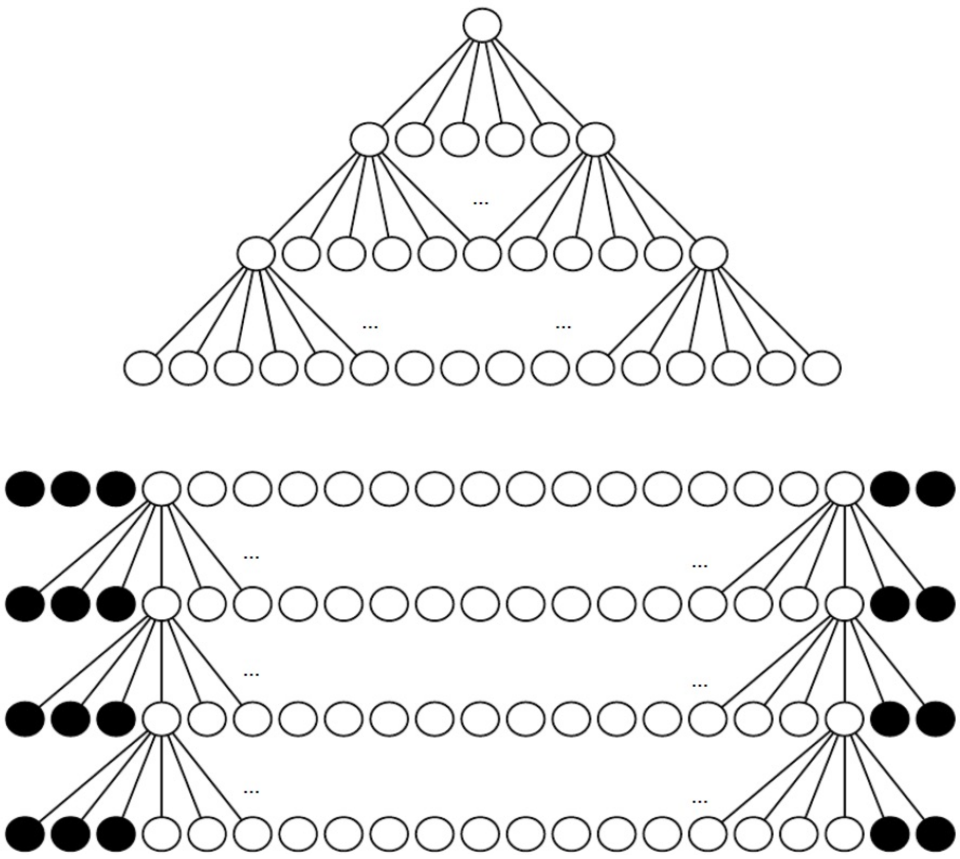
Deep vs shallow in Computer Vision

- CV work(ed) a lot on handcrafted local features
  - BoVW (Bag of Visual Words and extensions FisherVectors, BossaNova …)
  - BoVW not so shallow but not end-to-end supervised learning
- CNN: end-to-end learning on a handcrafted architecture! [Chatfield BMVC 2014]
  - Why 8 layers? why 3x3 at the 5th layer without polling? … => ad-hoc architecture

Zero-padding in convolutional neural network

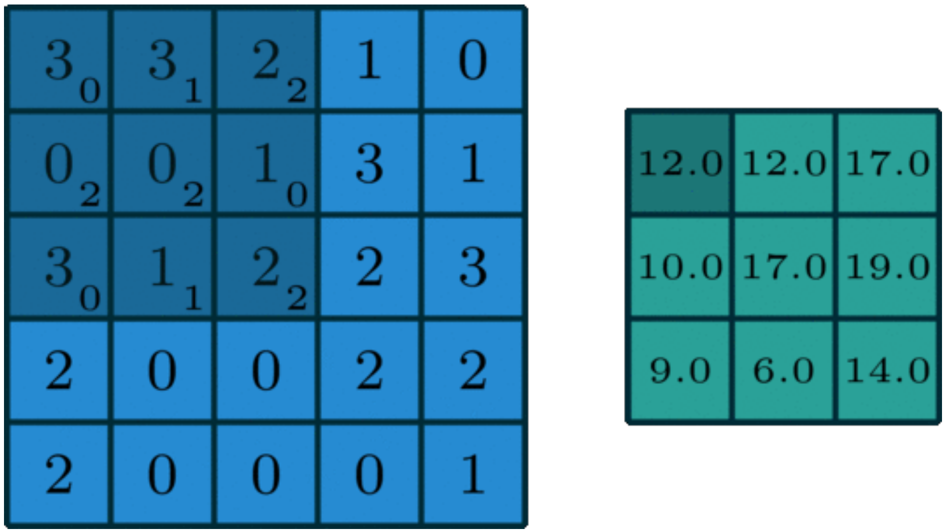


To avoid shrinking the spatial extent of the network rapidly

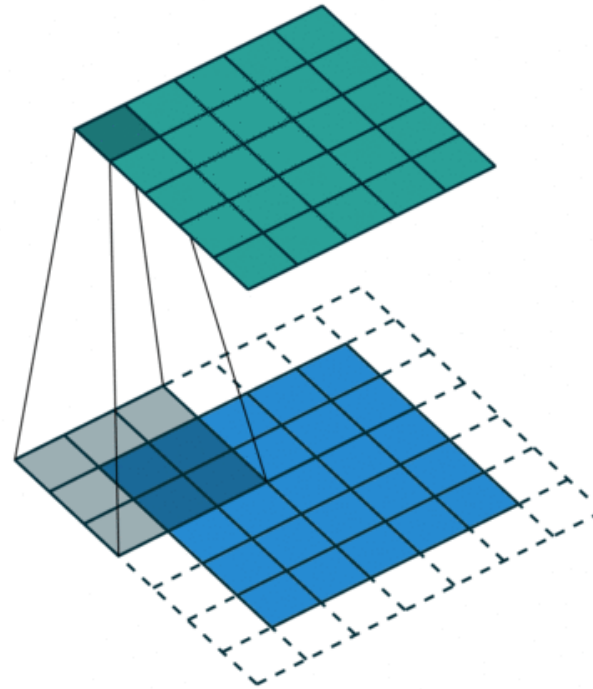


No Padding / Padding

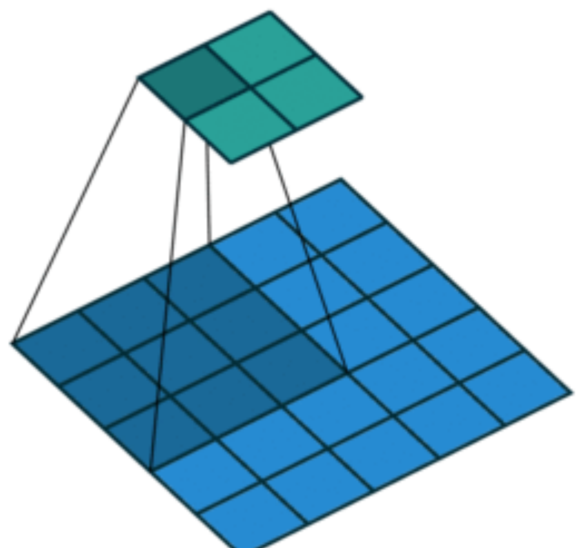
No padding 5x5 => 3x3



Padding 5x5 => 5x5



Stride 2 conv



More in this [link]()

More in getting local Invariance

Invariance to local translation (small shift) OK with pooling

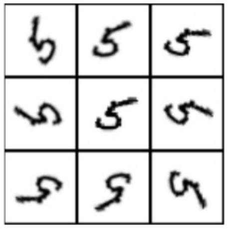
Is convolution equivariant/invariant to changes in scale or rotation?

- No such invariance with linear filters

Possible extension:

- Pooling OVER outputs of separately parameterized convolutions
- Become possible to LEARN invariance to rotation (or other)
- Example (Bengio et al. Deep Learning 2014):

By learning to have each filter be a different rotation of the “5” template + pooling over outputs => invariance to rotation of the “5”



“This is in contrast to translation invariance, which is usually achieved by hard-coding the net to pool over shifted versions of a single learned filter”