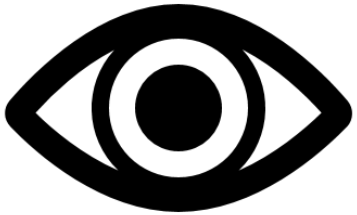
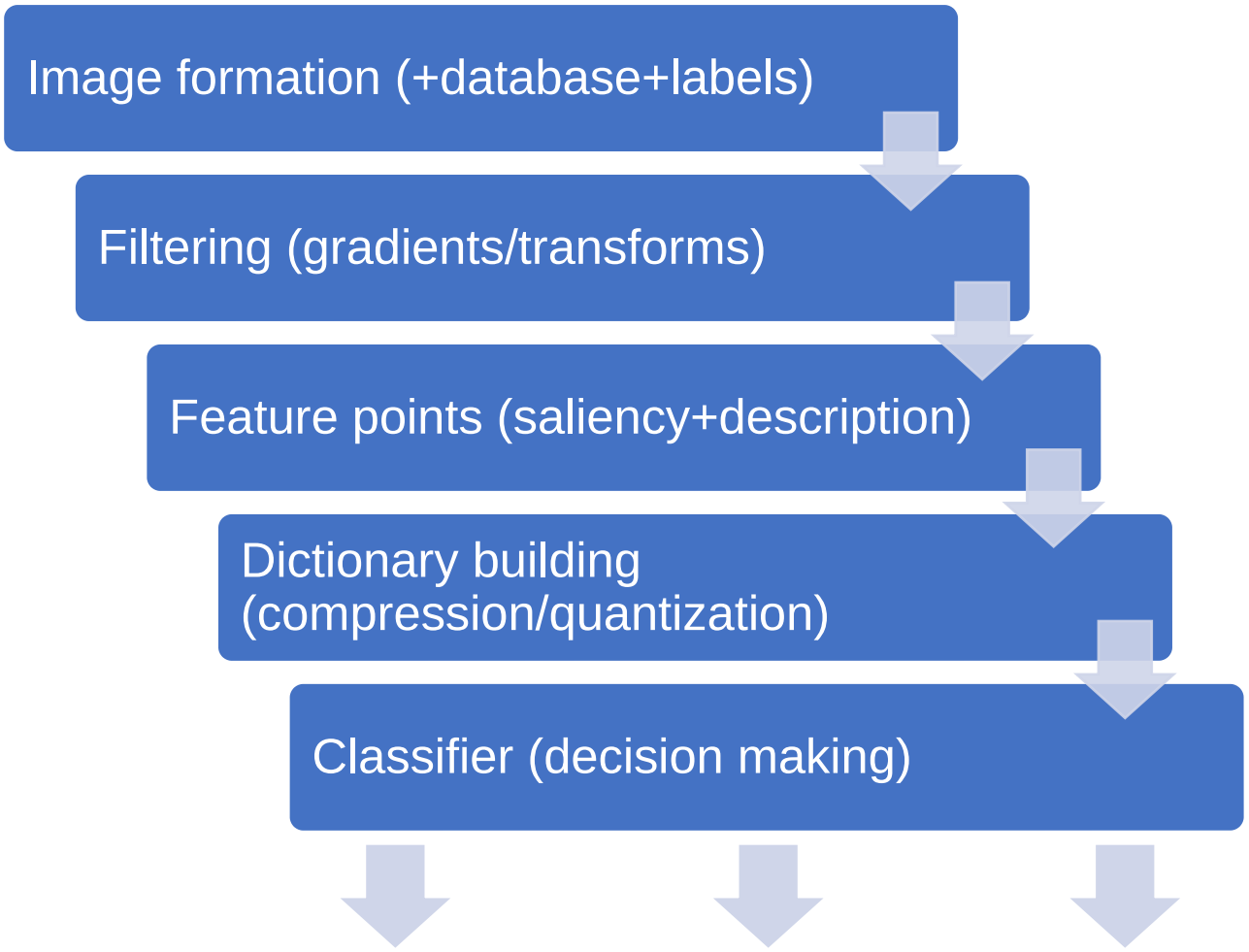


Advanced Image Processing
Neural Networks: CNN

SYDE 671

What do we have so far?



Captured+manual.

Hand designed.

Hand designed.

Hand designed.

Learned.

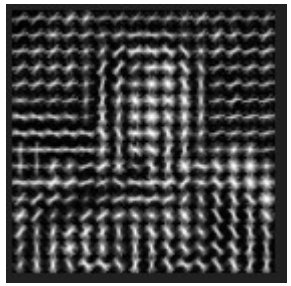
Recognition: Classification Object Detection Segmentation

What do we have so far?

Best performing visions systems have commonality:

Hand designed features

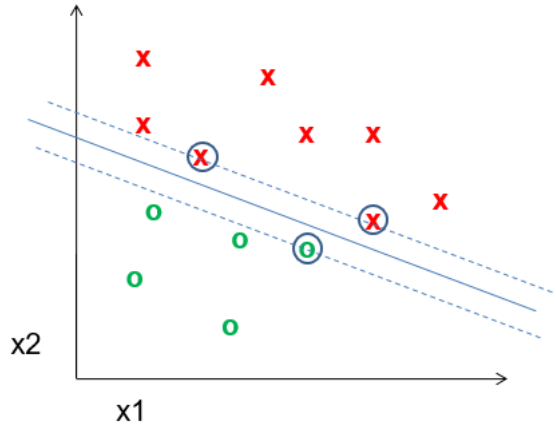
- Gradients + non-linear operations (exponentiation, clamping, binning)
- Features in combination (parts-based models)
- Multi-scale representations



Machine learning from databases

Linear classifiers (SVM)

- Some non-linear kernel tricks



But it’s still not that good…

PASCAL VOC = ~75% 20-class

ImageNet = ~75% 1000-class, top 5

ImageNet (human) = ~95%

Problems:

- Lossy features
- Lossy quantization
- ‘Imperfect’ classifier

But it’s still not that good…

- PASCAL VOC = ~75%
- ImageNet = ~75%; human performance = ~95%

How to solve?

- *Features: More principled modeling?*
  We *know* why the world looks (it’s physics!);
  Let’s build better physically-meaningful models.
- *Quantization: More data and more compute?*
  It’s just an interpolation problem; let’s represent the space with fewer data approximations.
- *Classifier: …*

The limits of learning?

Previous claim (ML-supervised.pdf):

It is more important to have
more or better labeled data than to use
a different supervised learning technique.

“The Unreasonable Effectiveness of Data” - Norvig

No free lunch theorem for ML

Wolpert (1996): https://ieeexplore.ieee.org/document/585893

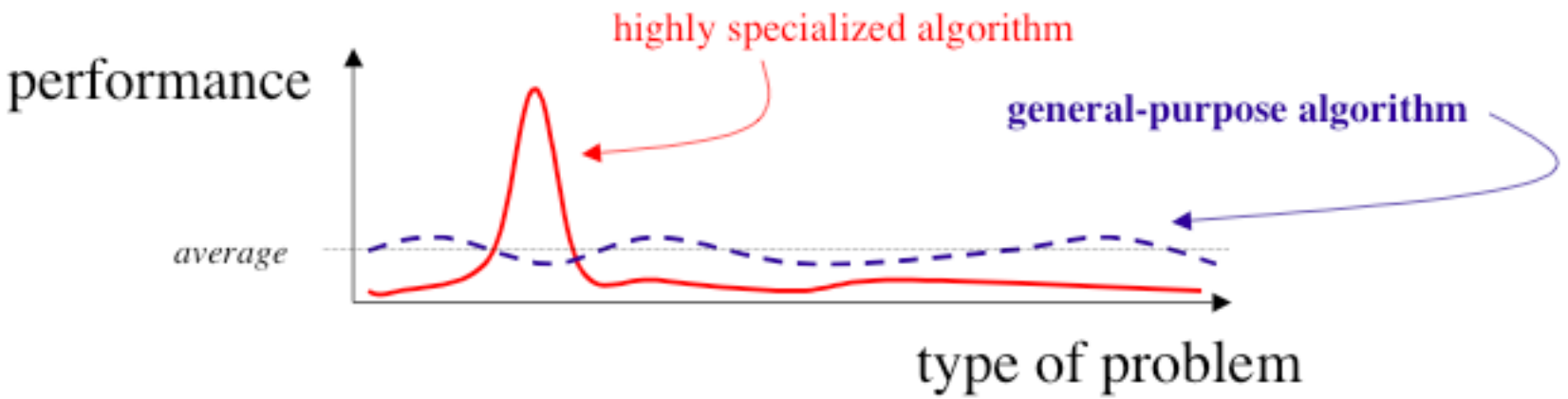
‘No free lunch’ for supervised learning:

“In a noise-free scenario where the loss function is the misclassification rate, if one is interested in off-training-set error, then there are no *a priori* distinctions between learning algorithms.”

-> Averaged over all possible datasets, no learning algorithm is better than any other.

OK, well, let’s give up. Class over.

Not so fast!



We can build a classifier which better matches the characteristics of the problem!

Or….high performance is possible by making more assumptions.

But…didn’t we just do that?

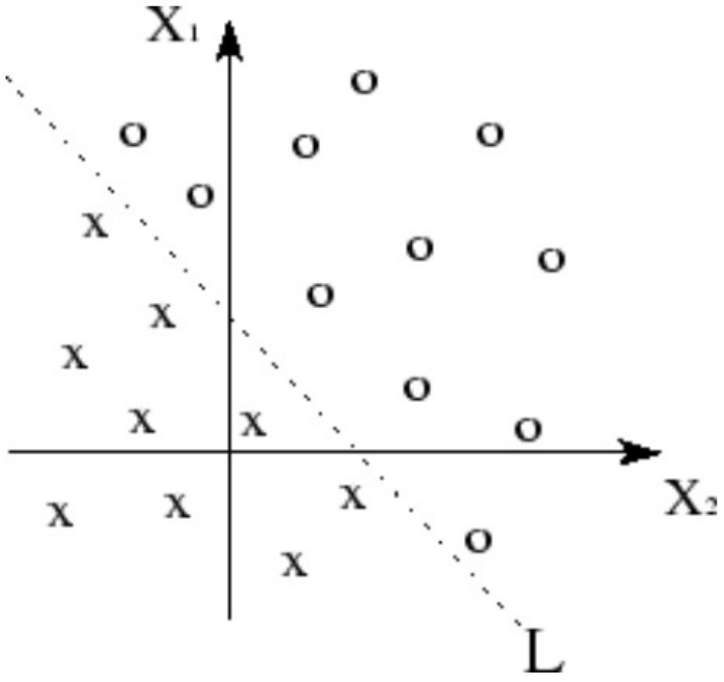
- PASCAL VOC = ~75%
- ImageNet = ~75%; human performance = ~95%

We used intuition and understanding of how we think vision works, but it still has limitations.

Why?

Linear spaces - separability

- + kernel trick to transform space.



Linearly separable data
+ linear classifer = good.

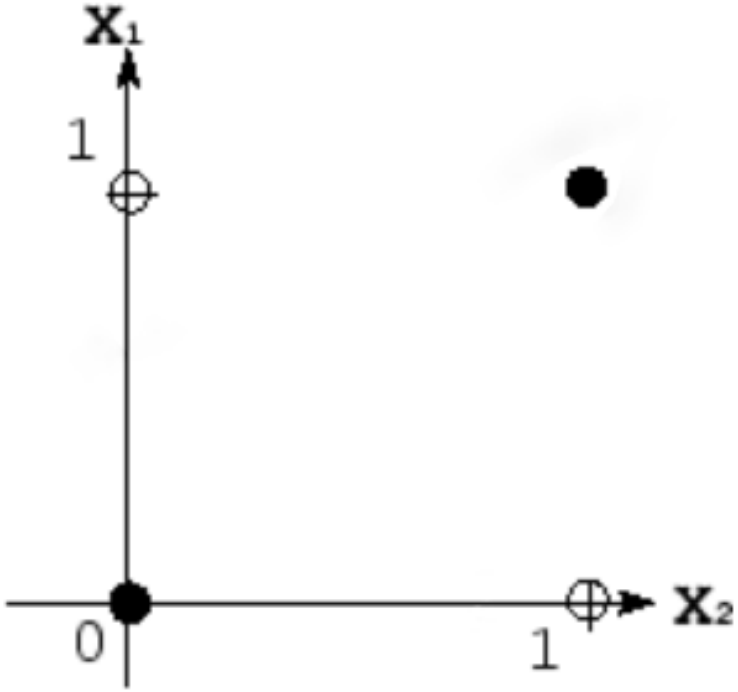
Non-linear spaces - separability

Take XOR – exclusive OR

E.G., human face has two eyes XOR sunglasses

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = X_1 \oplus X_2$

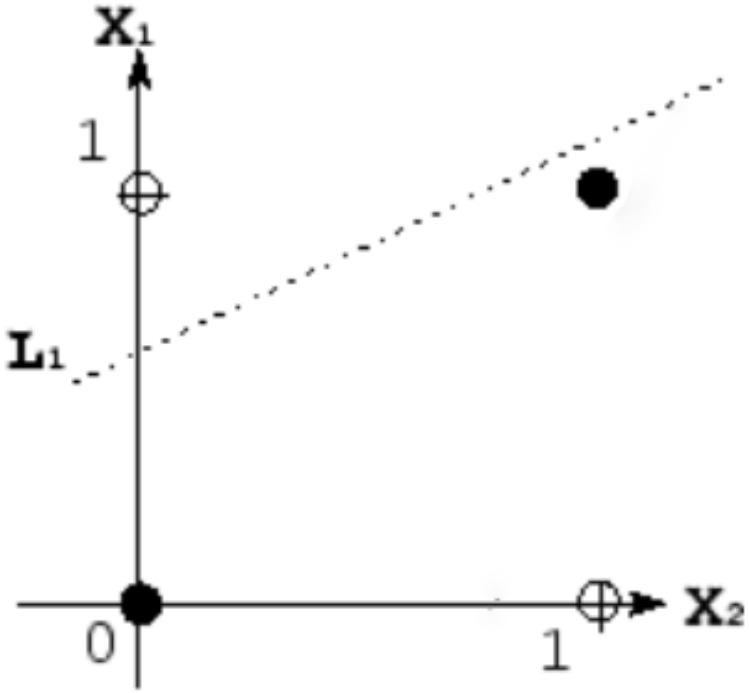


Non-linear spaces - non-separability

Linear functions are insufficient on their own.

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

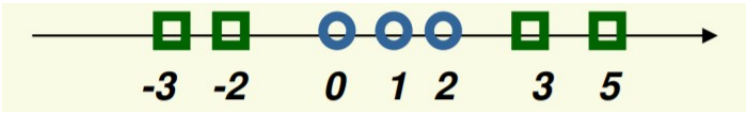
$Y = X_1 \oplus X_2$



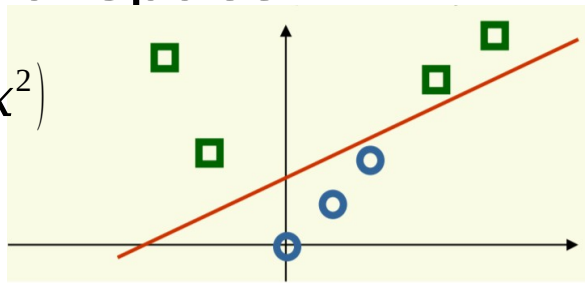
Kawaguch

We already learned a couple of solutions

- Kernel trick in SVM: non-seperable data is more likely to be seperable when non-linearly mapped to a higher dimensional space

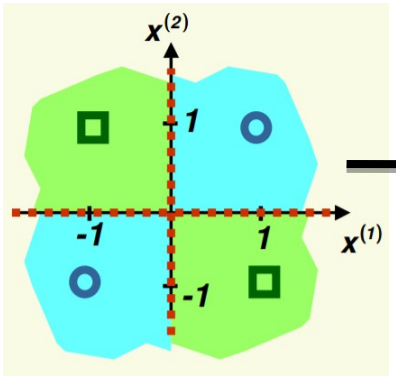


$$\varnothing(x)=(x,x^2)$$

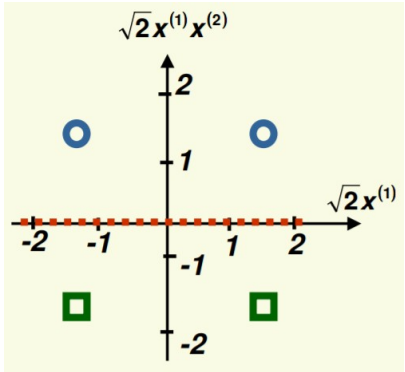


Back to XOR problem:

SVM+kernel trick

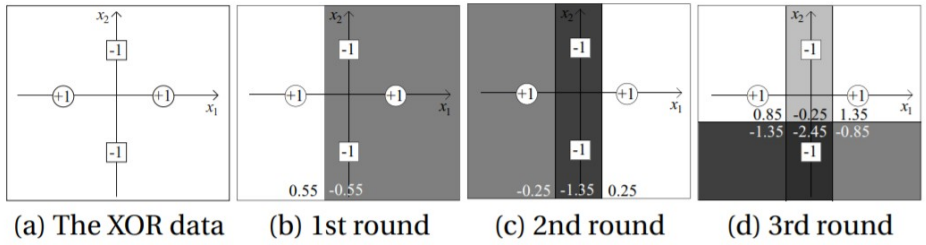


nonlinear decision boundary



linear decision boundary

Boosting



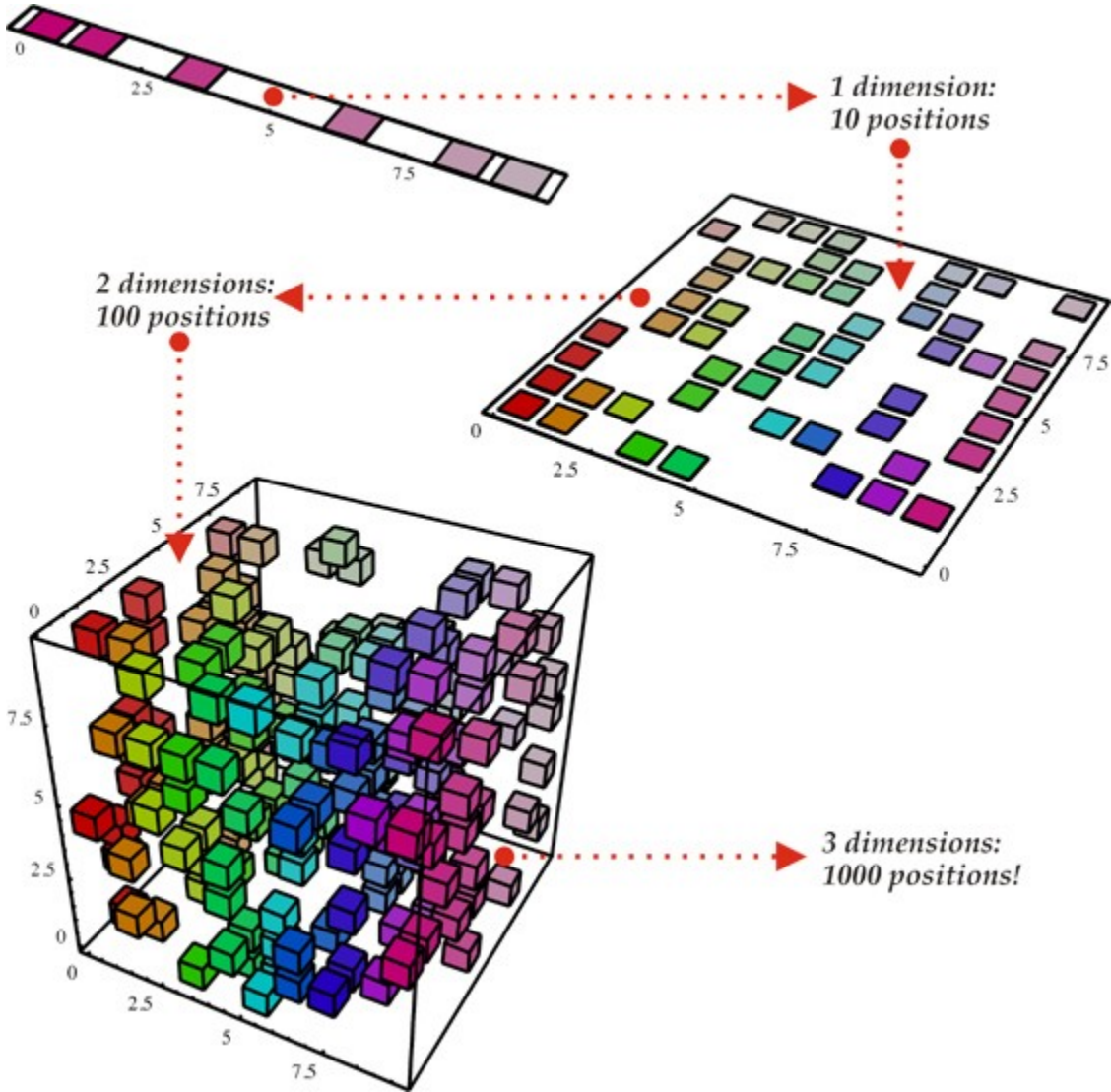
https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/emfa-ch2.pdf

Problems: Curse of Dimensionality

Every feature that we add requires us to learn the useful regions in a much larger volume.

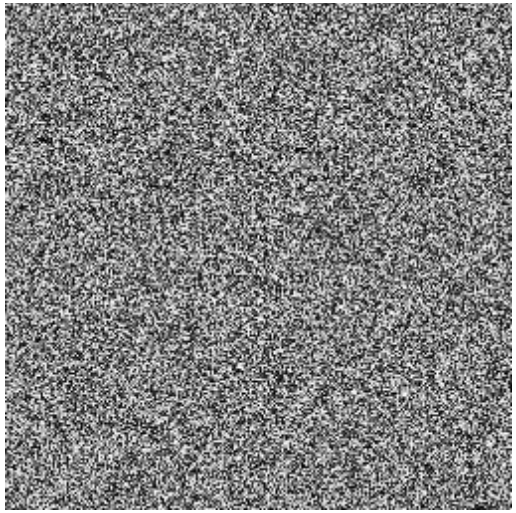
d binary variables =

O(2^d) combinations



Curse of Dimensionality

```
>> I = rand(256,256);
>> imshow(I);
```



@ 8bit = 256 values ^ 65,536

Not all regions of this high-dimensional space are meaningful.

Local constancy / smoothness of feature space

All existing learning algorithms we have seen assume **smoothness** or **local constancy.**

-> New example will be near existing examples

-> Each region in feature space requires an example

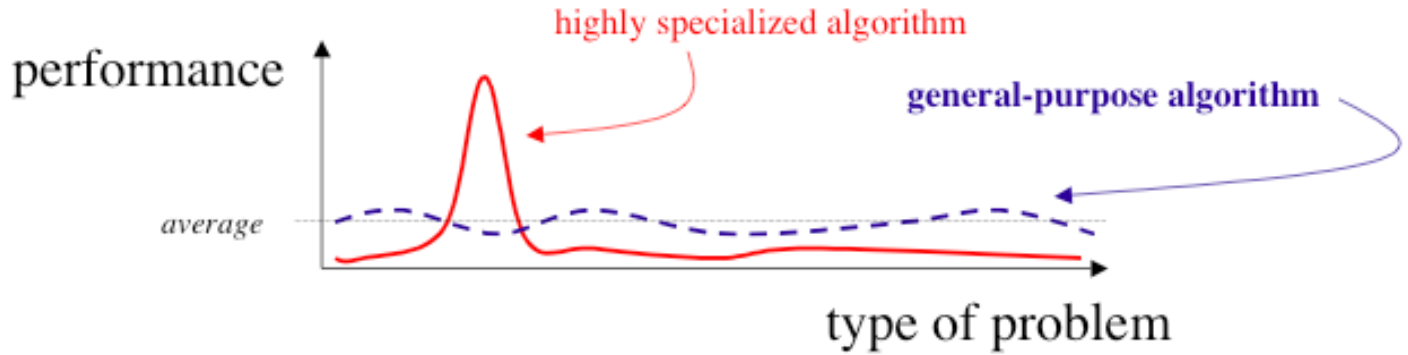
-> Cannot generalize beyond examples

Extreme example: k-NN classifier (k=1). The number of regions cannot be more than the number of examples.

How to try and represent this high-dimensional space in a way which maximizes generalization?

More specialization?

- PASCAL VOC = ~75%
- ImageNet = ~75%; human performance = ~95%



Is there a way to make our system better suited to the problem?

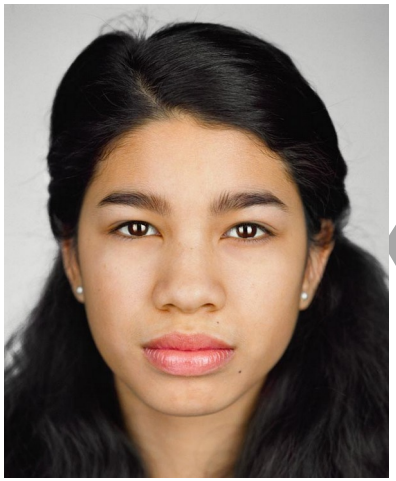
Goal: Build a classifier which is more powerful at representing complex functions *and* more suited to the learning problem.

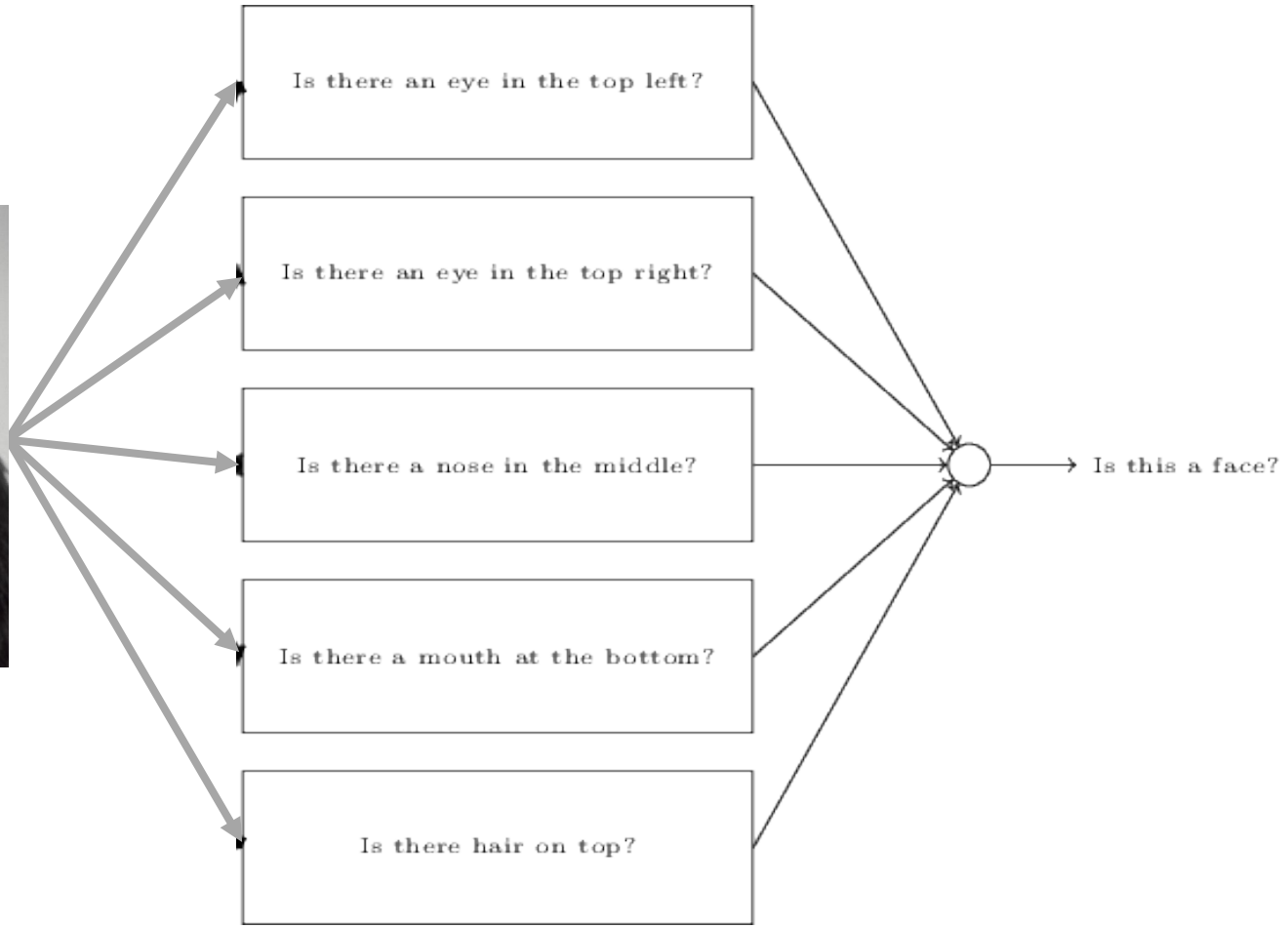
What does this mean?

1. Assume that the *underlying data generating function* relies on a composition of factors in a hierarchy.

Dependencies between regions in feature space
= factor composition

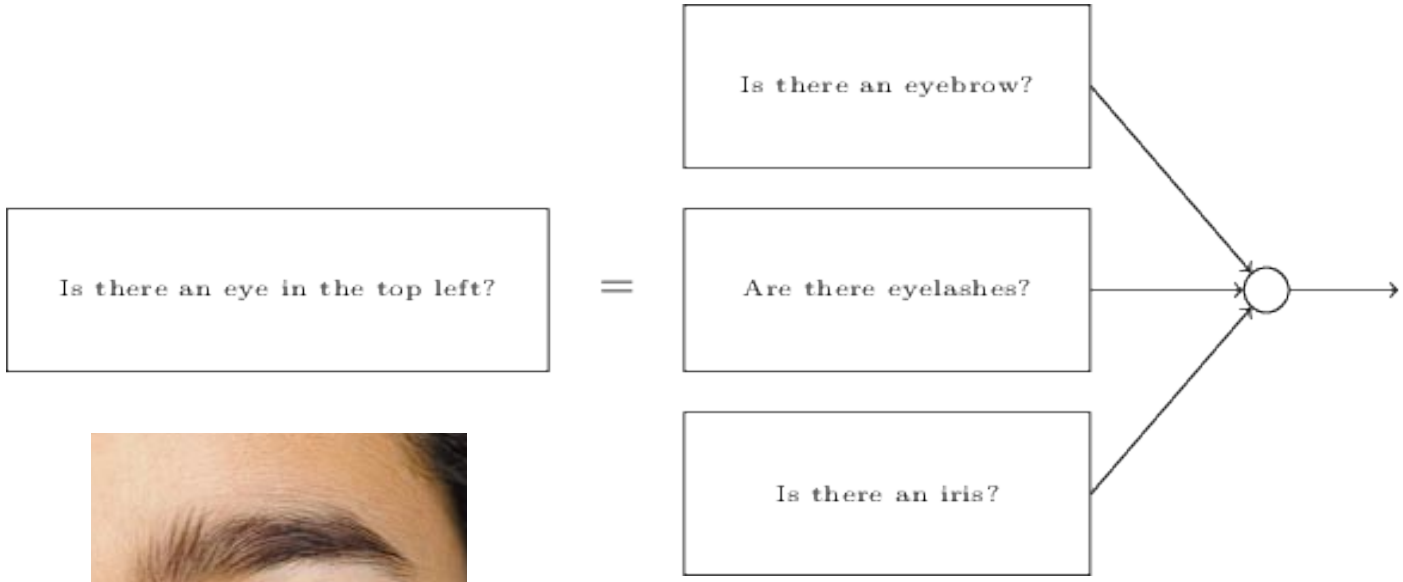
Example





Nielsen, National Geographic

Example



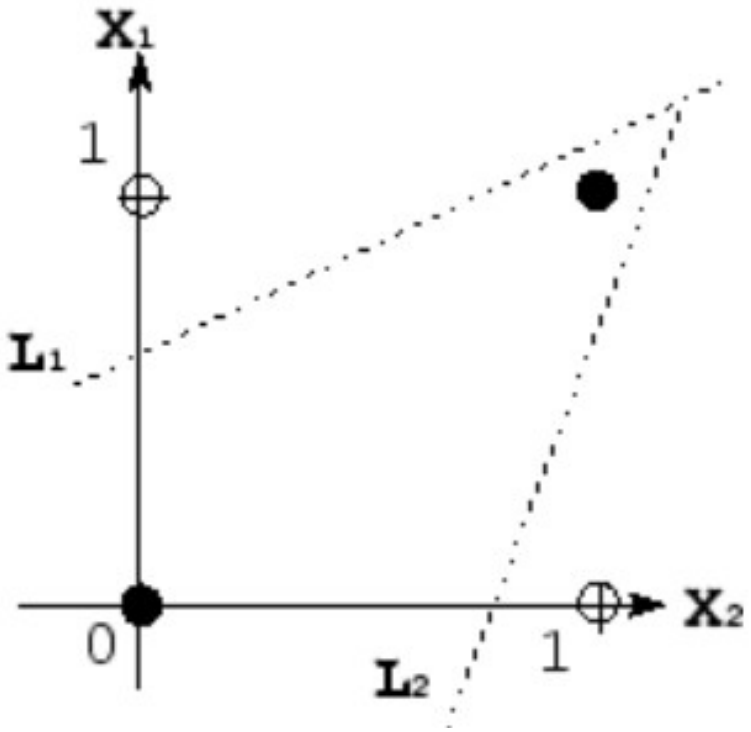
Nielsen, National Geographic

Non-linear spaces - separability

Composition of linear functions can represent more complex functions.

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = X_1 \oplus X_2$



Goal: Build a classifier which is more powerful at representing complex functions *and* more suited to the learning problem.

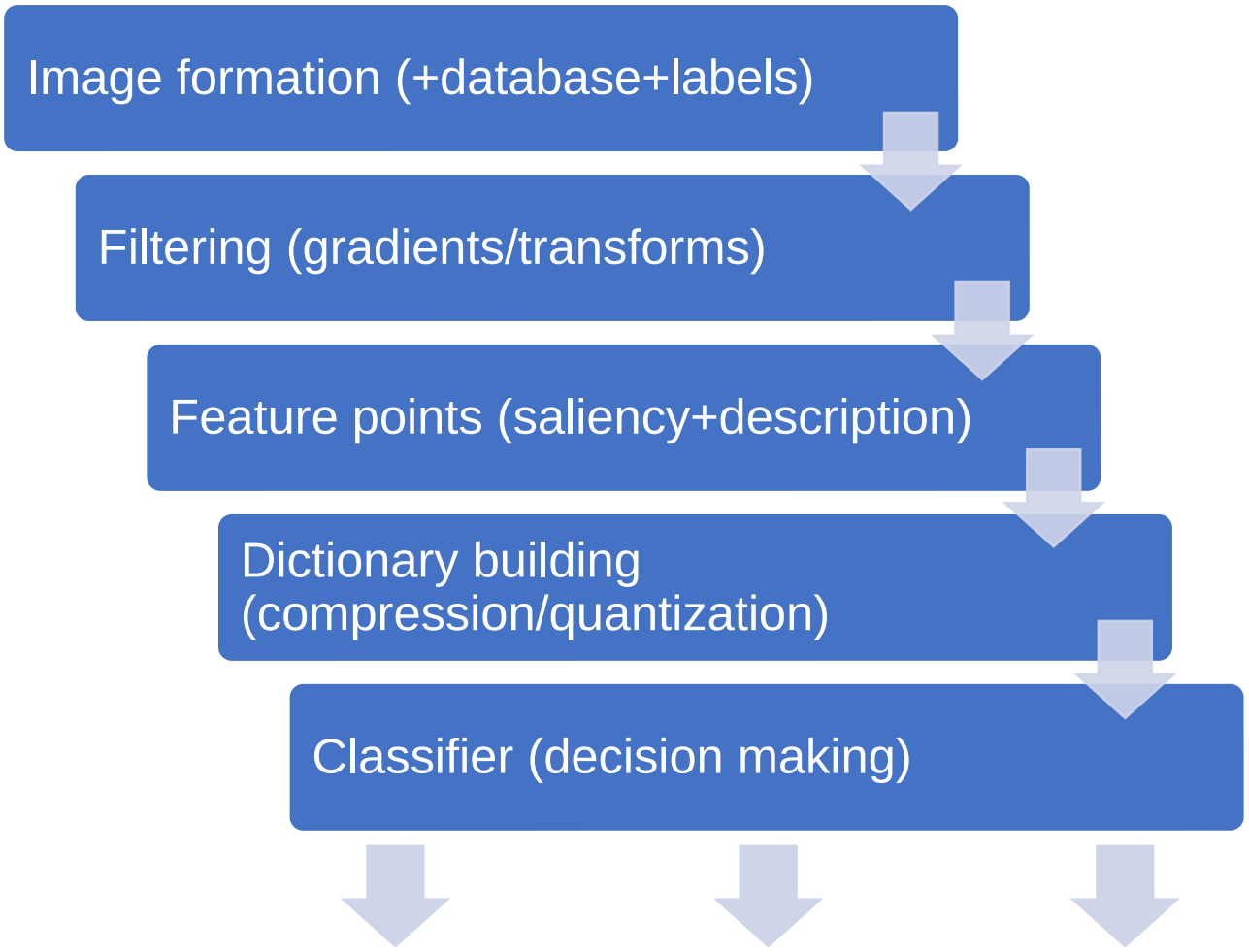
What does this mean?

1. Assume that the *underlying data generating function* relies on a composition of factors in a hierarchy.

2. Learn a feature representation specific to the dataset.

10k/100k features + factor composition
= sophisticated representation.

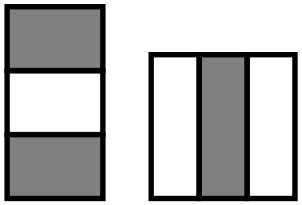
Reminder: Traditional ML



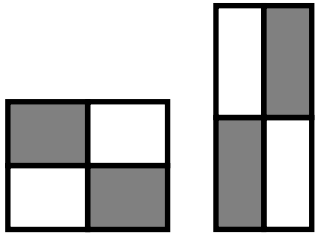
Reminder: Viola Jones Face Detector

Combine *thousands* of 'weak classifiers'

Two-rectangle features

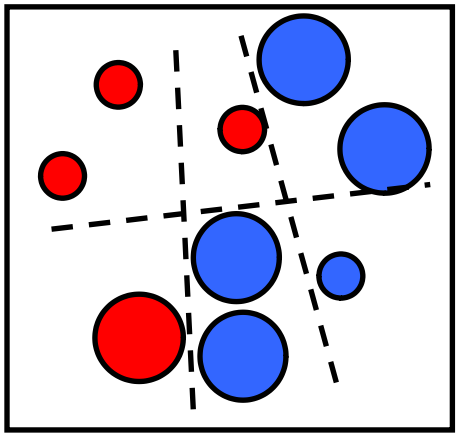


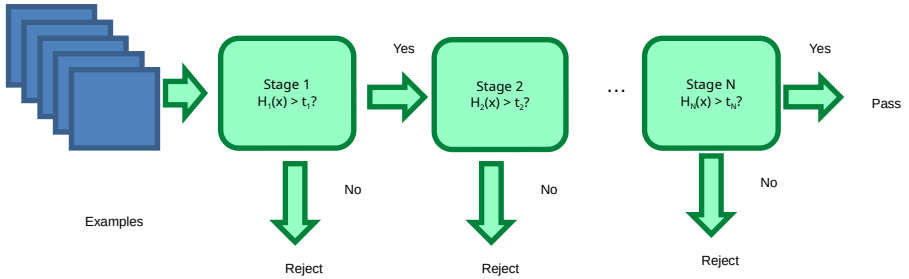
Three-rectangle features



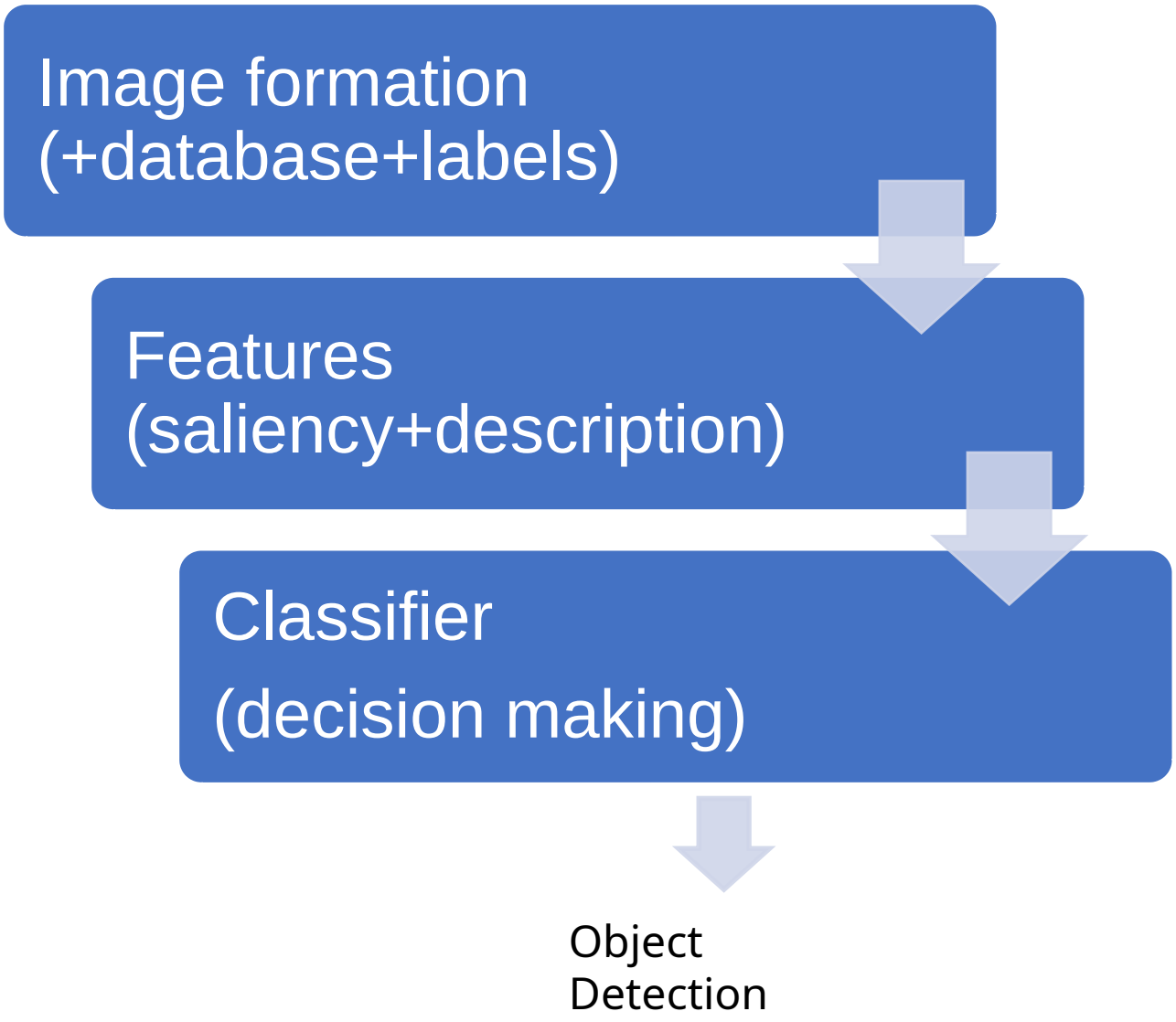
Etc.

Learn how to combine in cascade with boosting





Reminder: Viola Jones Face detector

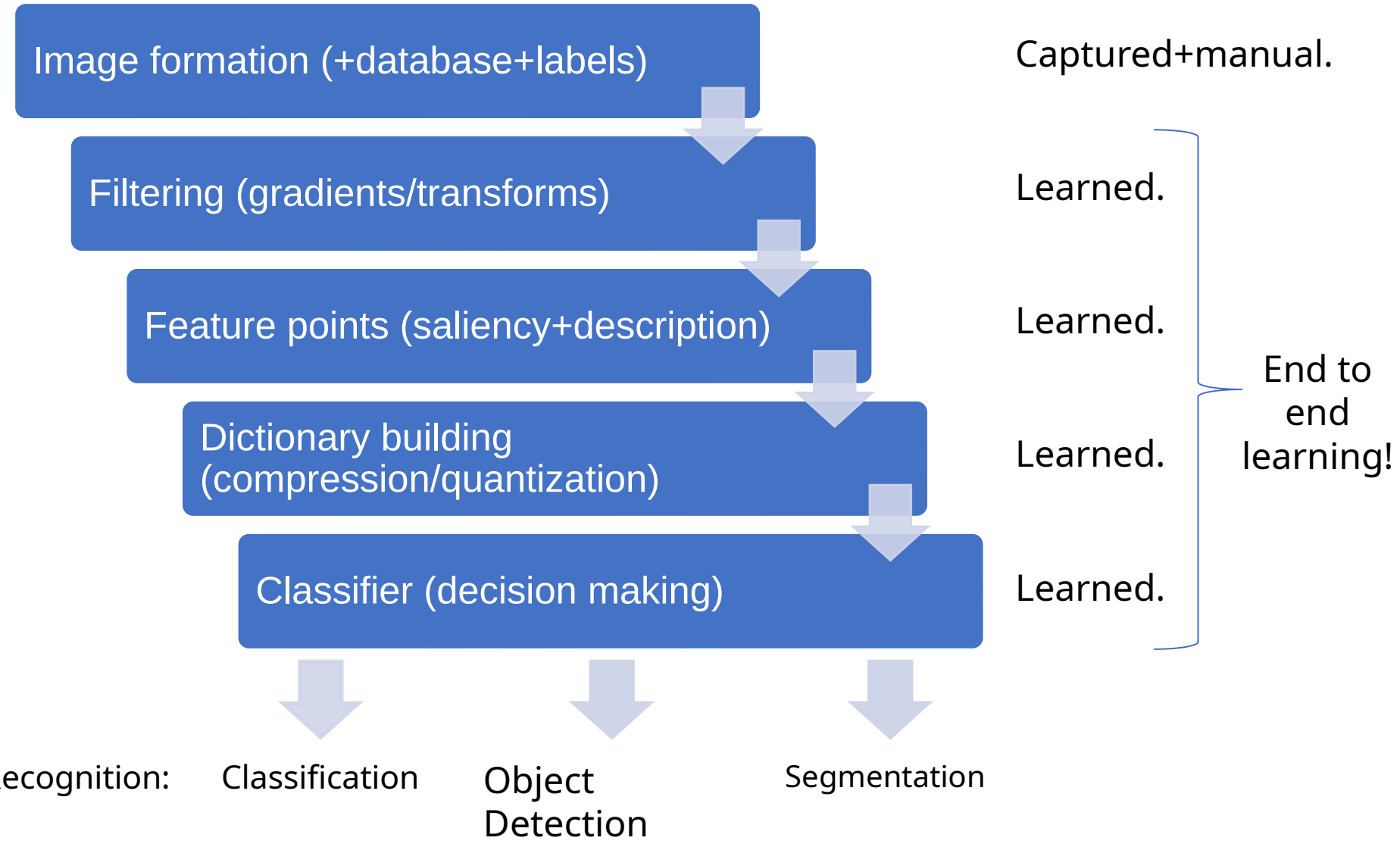


Captured+manual.

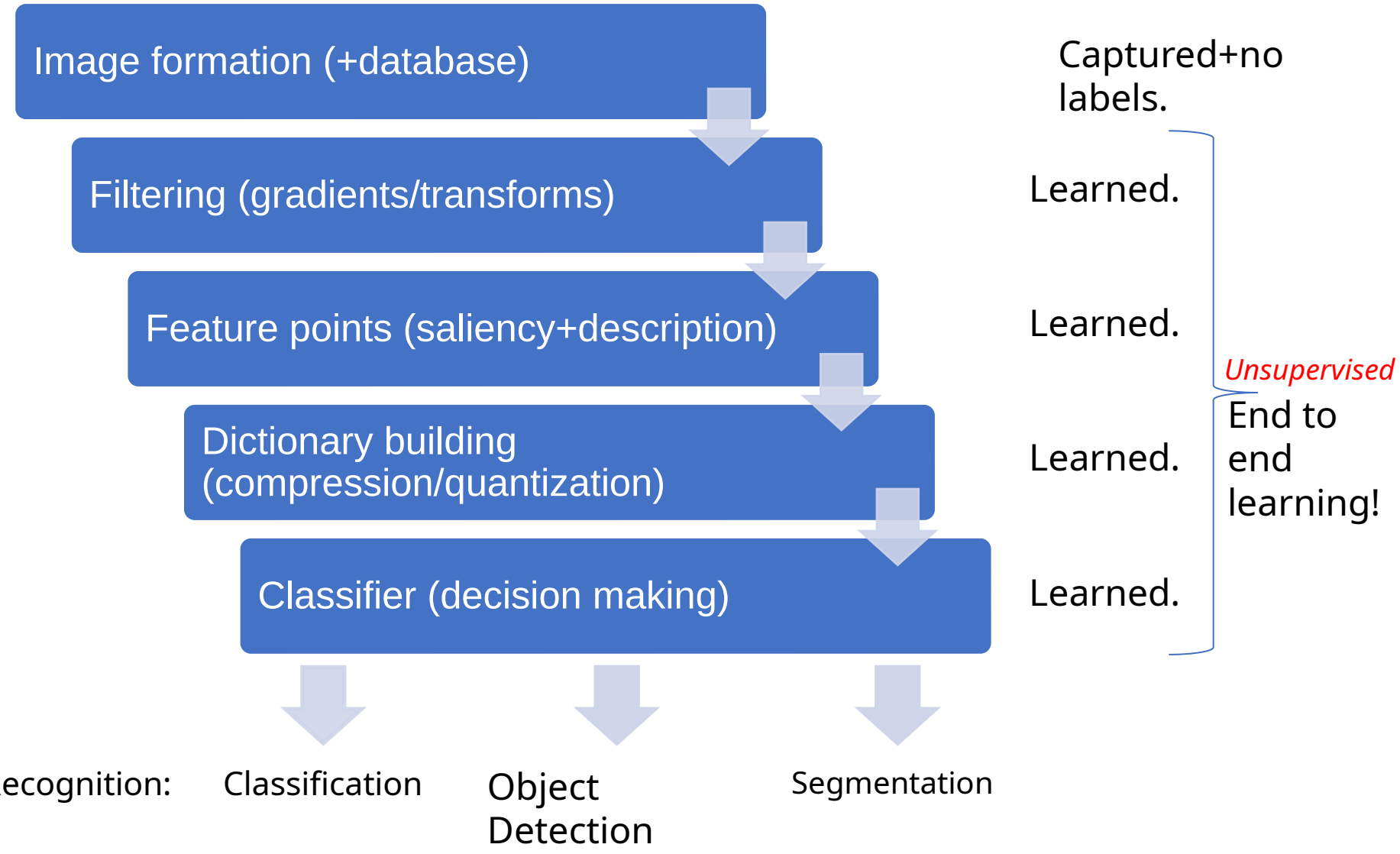
Specified space, but selected automatically.

Learned combination.

Wouldn’t it be great if we could…



Well if we can do that, then what about…

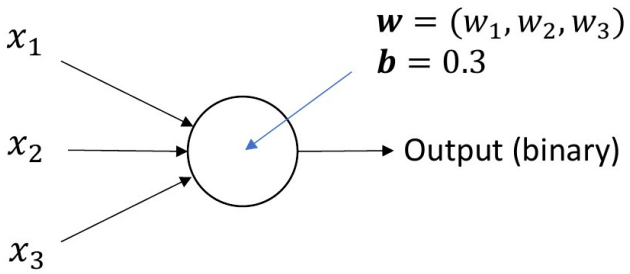


Neural Networks

Neural Networks

Basic building block for composition is a *perceptron* (Rosenblatt c.1960)

Linear classifier – vector of weights *w* and a 'bias' *b*



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

$$w \cdot x \equiv \sum_j w_j x_j$$

Binary classifying an image

Each pixel of the image would be an input.

So, for a 28 x 28 image, we vectorize:

$$\mathbf{x} = 1 \times 784$$

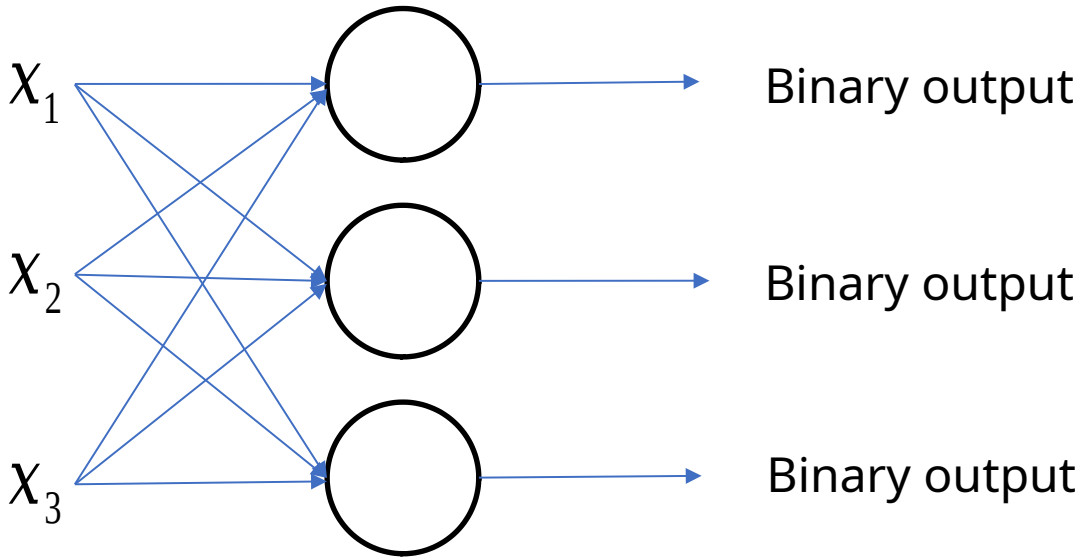
w is a vector of weights for each pixel, 784 x 1

b is a scalar bias per perceptron

Result = $\mathbf{xw} + b$ -> $(1 \times 784) \times (784 \times 1) + b = (1 \times 1) + b$

Neural Networks - multiclass

Add more perceptrons



Multi-class classifying an image

Each pixel of the image would be an input.

So, for a 28 x 28 image, we vectorize.

x = 1 x 784

W is a matrix of weights for each pixel/each perceptron

W = 10 x 784 (10-class classification)

b is a bias *per perceptron* (vector of biases); (1 x 10)

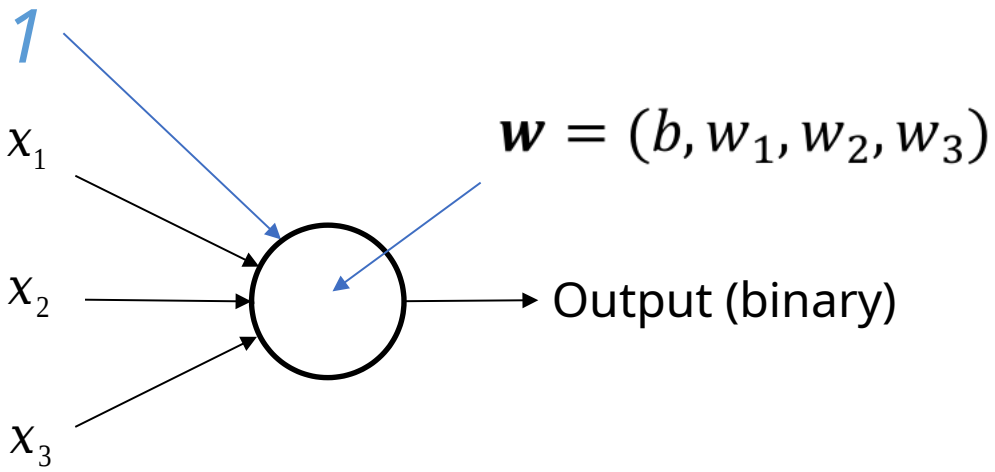
Result = **xW** + **b** -> (1x784) x (784 x 10) + **b**

-> (1 x 10) + (1 x 10) = output vector

Bias convenience

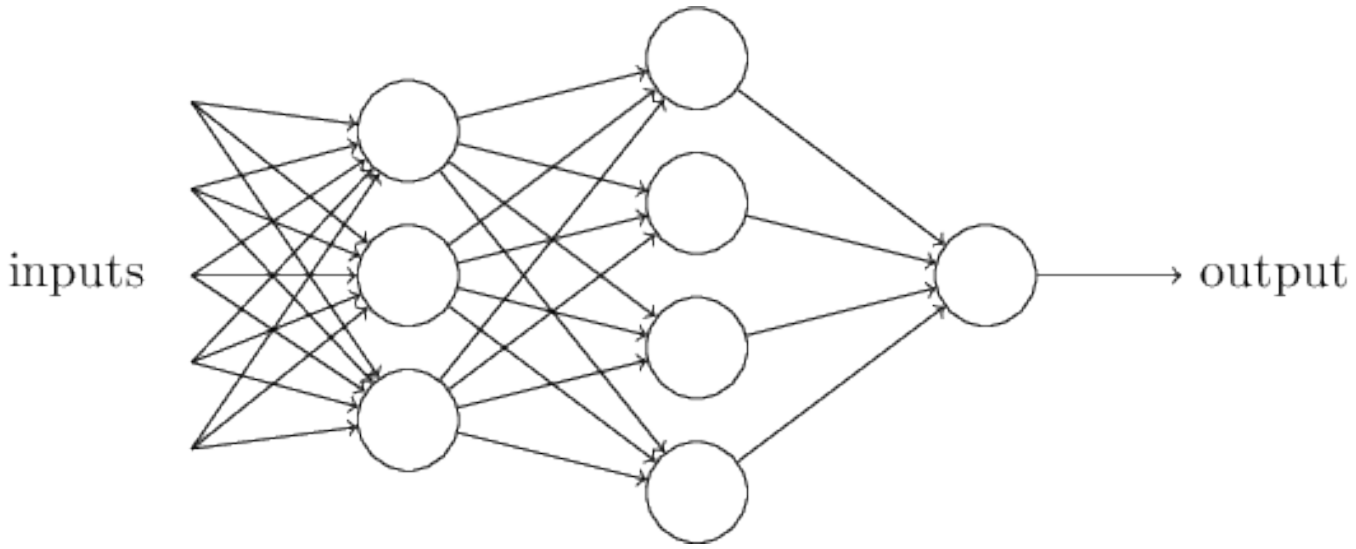
Let's turn this operation into a multiplication only:

- Create a 'fake' feature with value 1 to represent the bias
- Add an extra weight that can vary



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x \leq 0 \\ 1 & \text{if } w \cdot x > 0 \end{cases} \qquad w \cdot x \equiv \sum_j w_j x_j$$

Composition

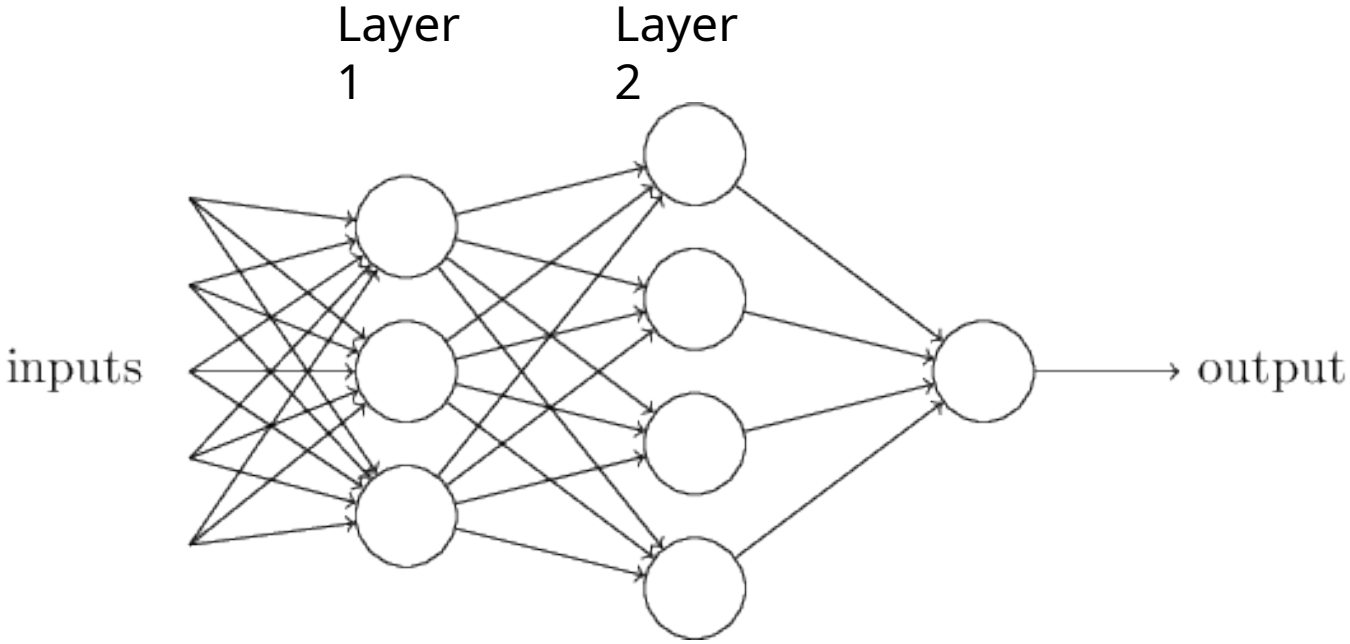


Attempt to represent complex functions as compositions of smaller functions.

Outputs from one perception are fed into inputs of another perceptron.

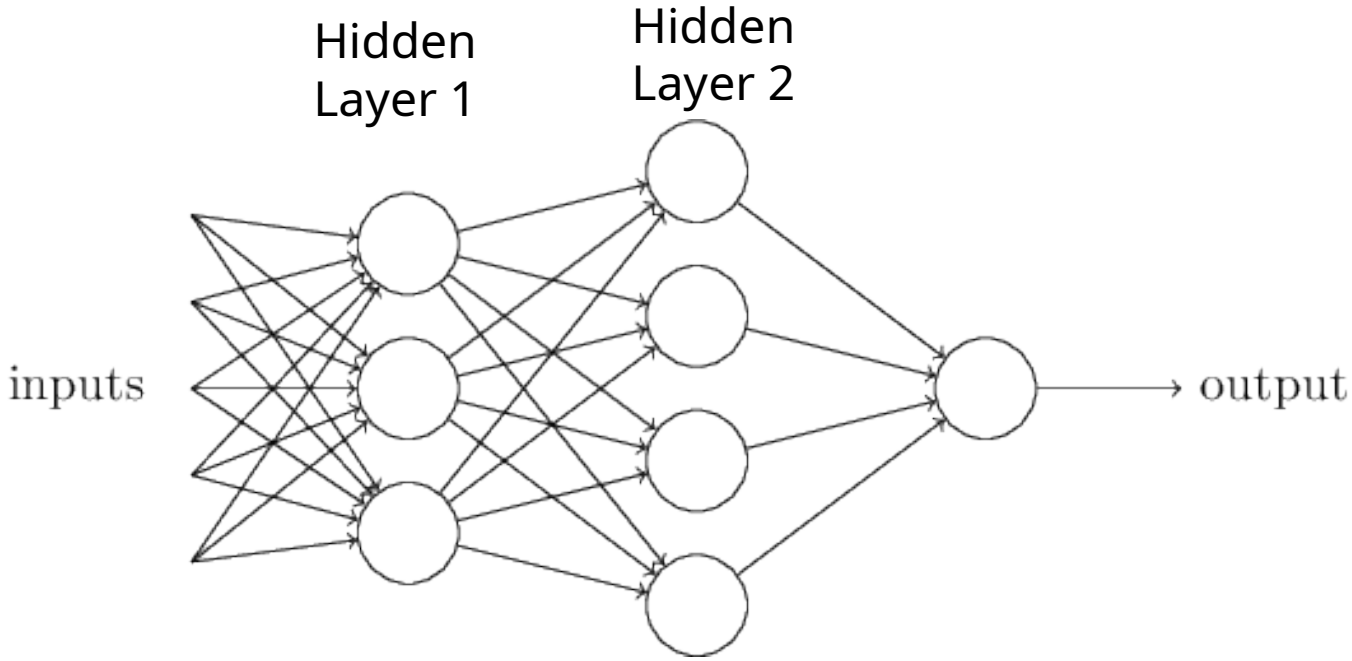
Nielse

Composition



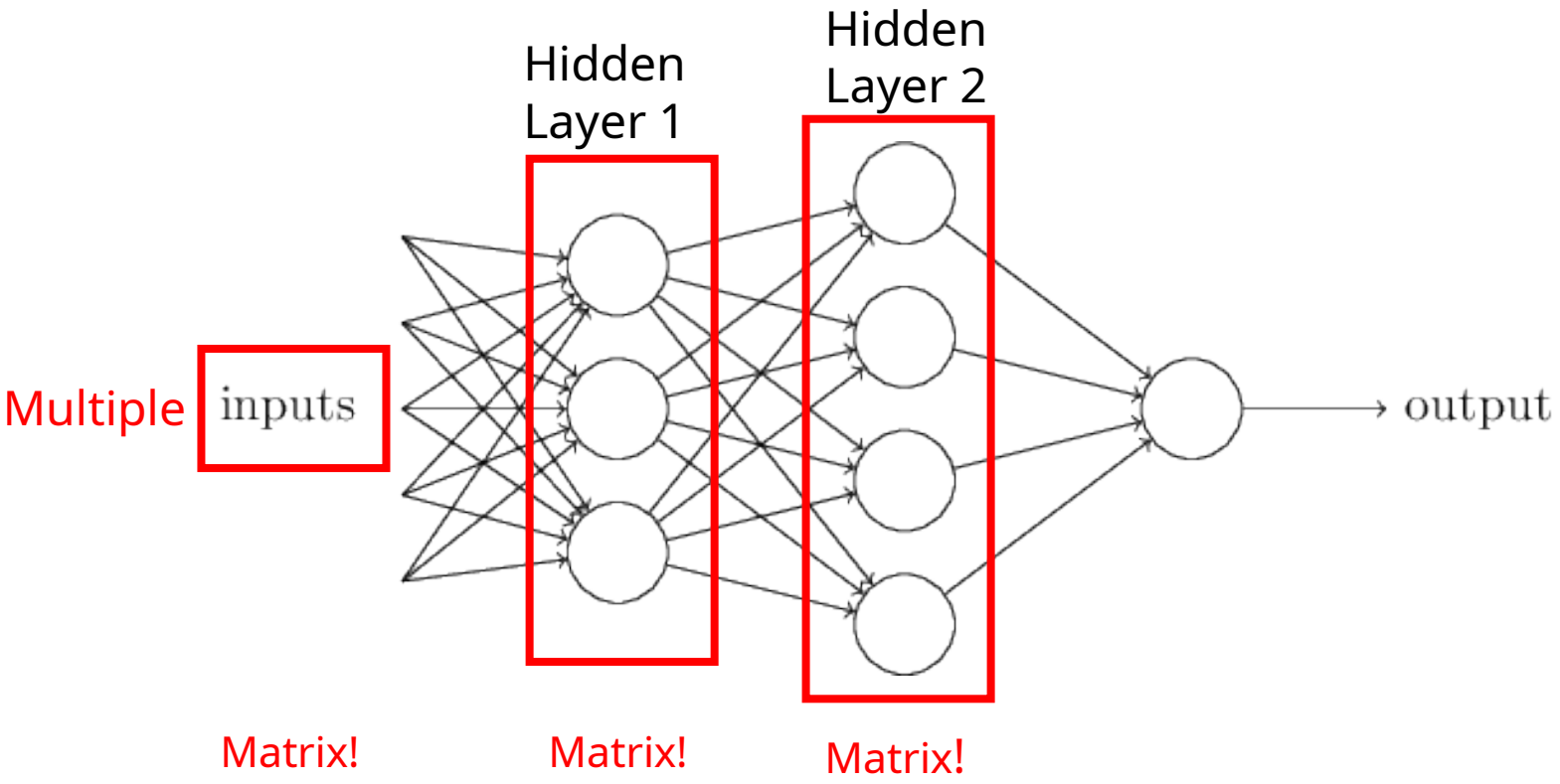
Sets of layers and the connections (weights) between them define the *network architecture.*

Composition



Layers that are in between the input and the output are called *hidden layers*, because we are going to *learn* their weights via an optimization process.

Composition



It's all just matrix multiplication!

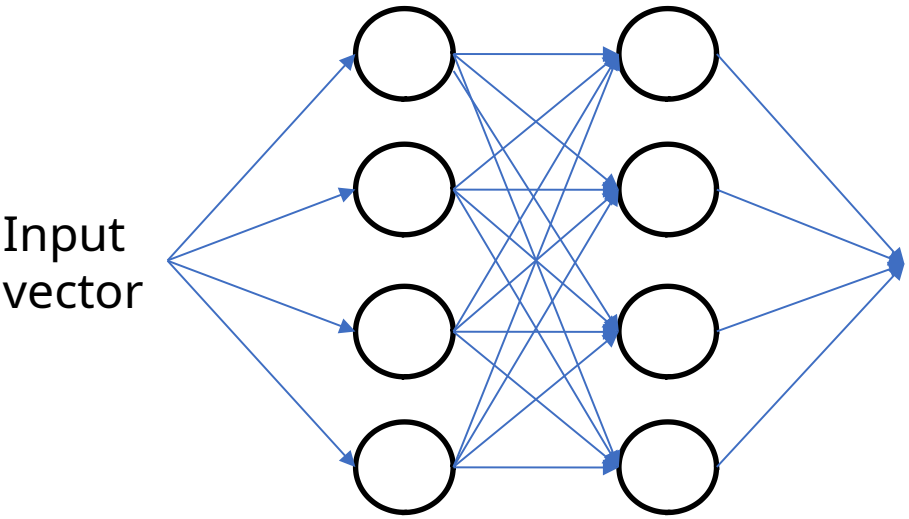
GPUs -> special hardware for fast/large matrix multiplication.

Problem 1 with all linear functions

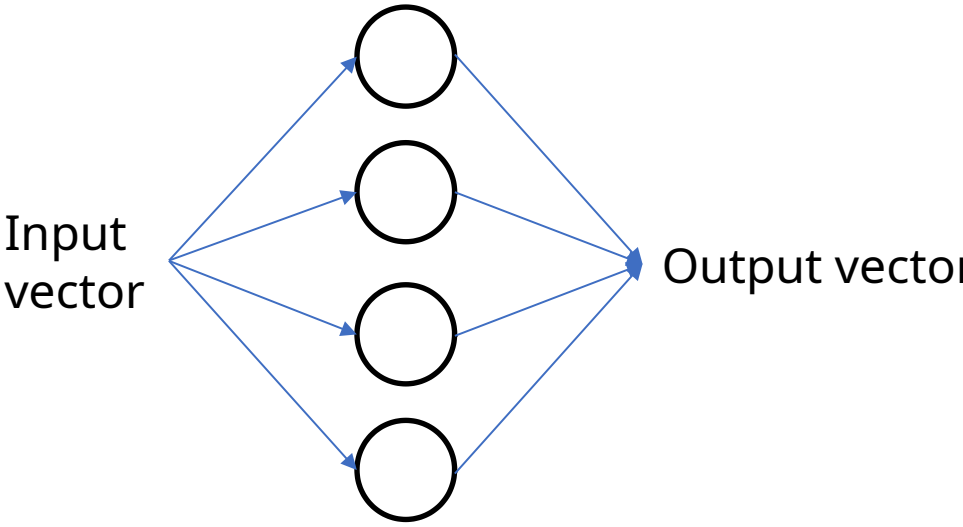
We have formed chains of linear functions.

We know that linear functions can be reduced

- g = f(h(x))



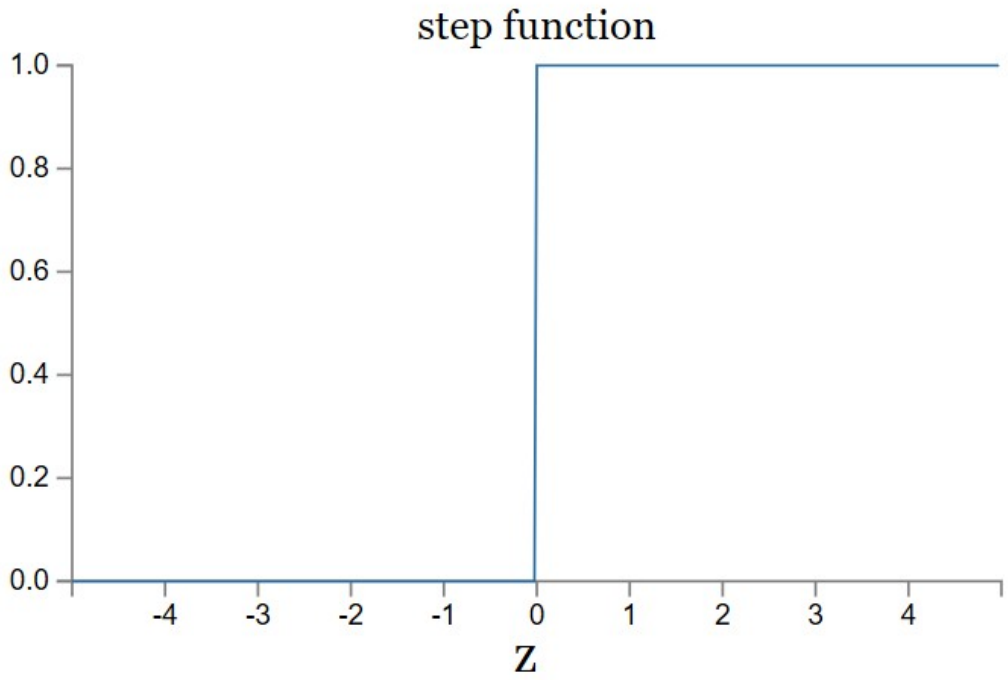
Our composition of functions is really just a single function : (



Problem 2 with all linear functions

Linear classifiers: small change in input can cause large change in binary output
= problem for composition of functions

Activation function

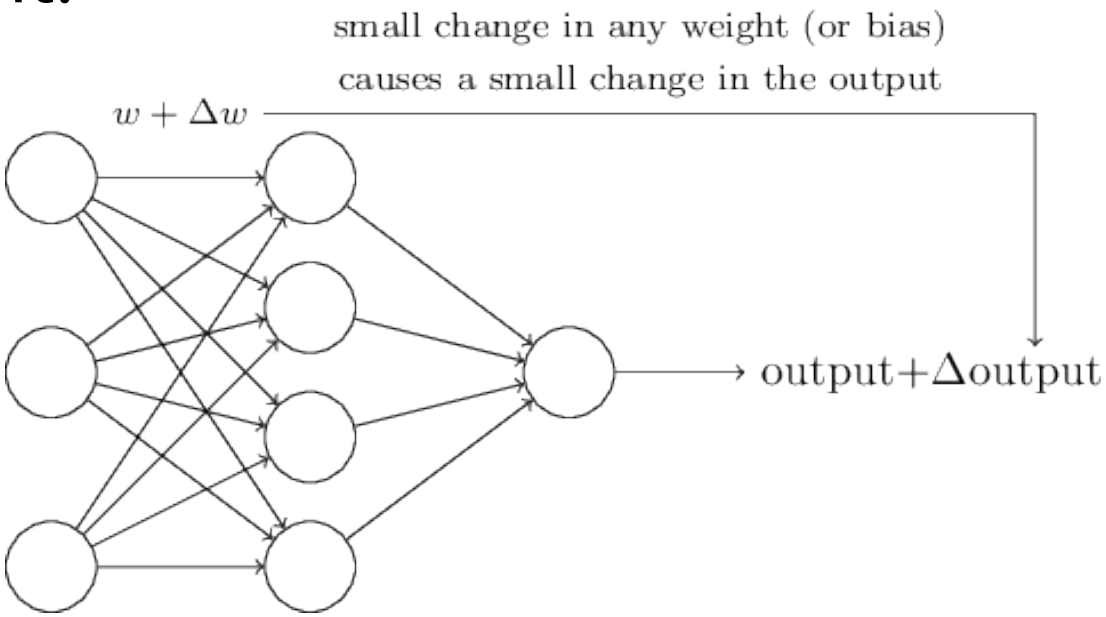


Nielse

Problem 2 with all linear functions

Linear classifiers: small change in input can cause large change in binary output.

We want:



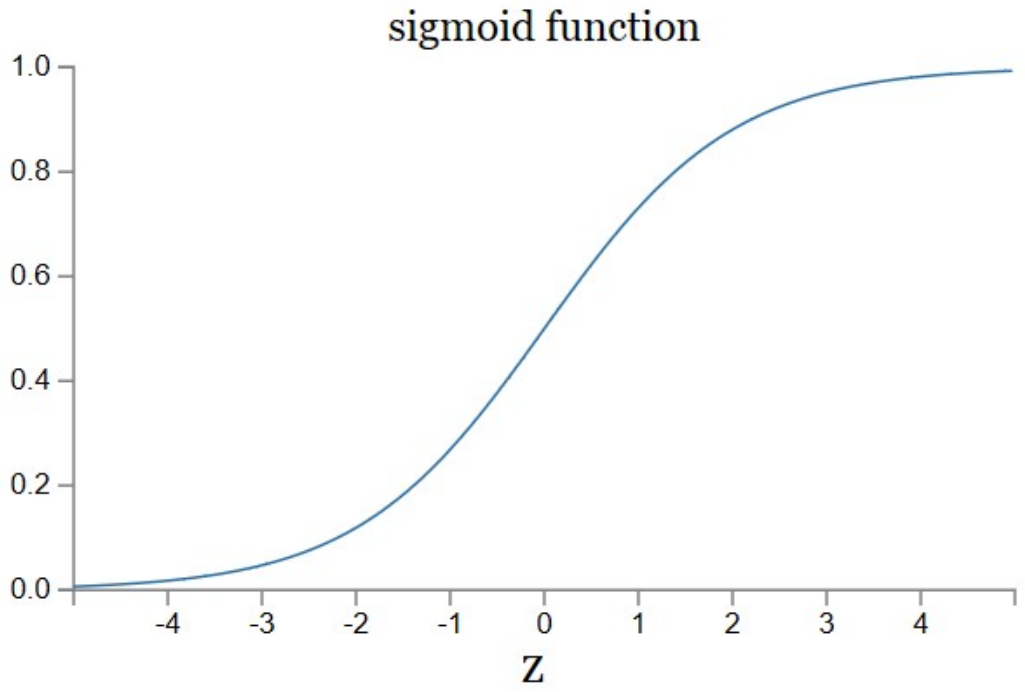
Nielse

Let’s introduce non-linearities

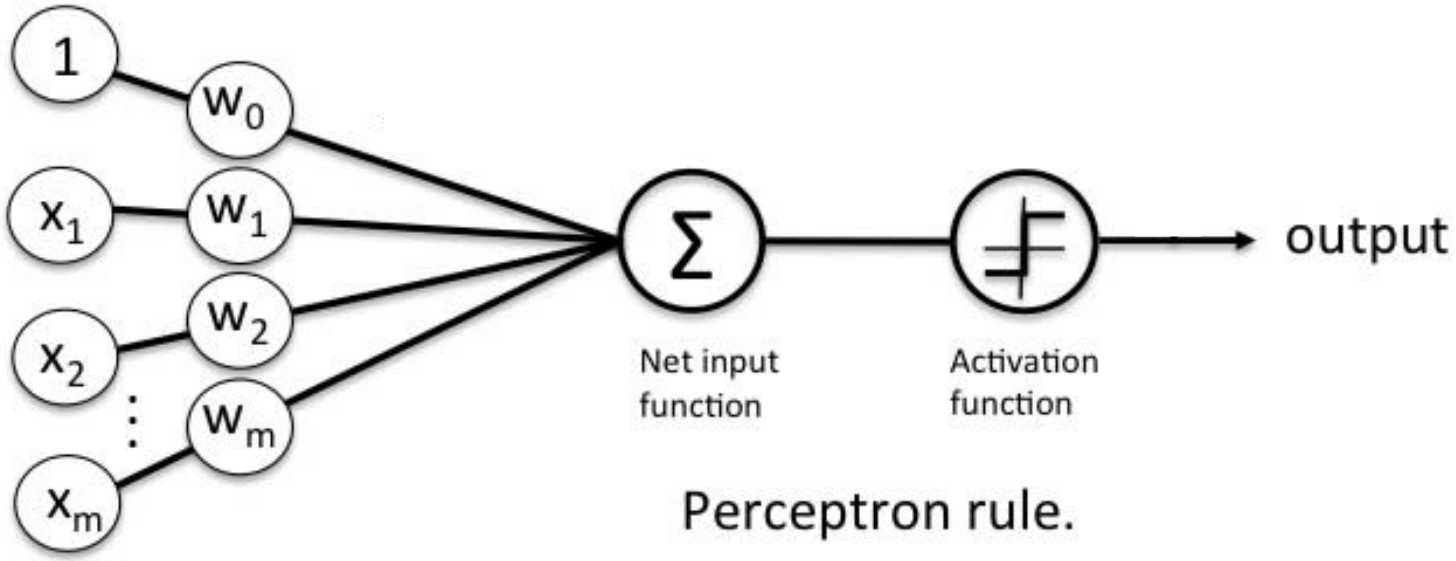
We’re going to introduce non-linear functions to transform the features.

$$\sigma(w \cdot x + b)$$

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}.$$

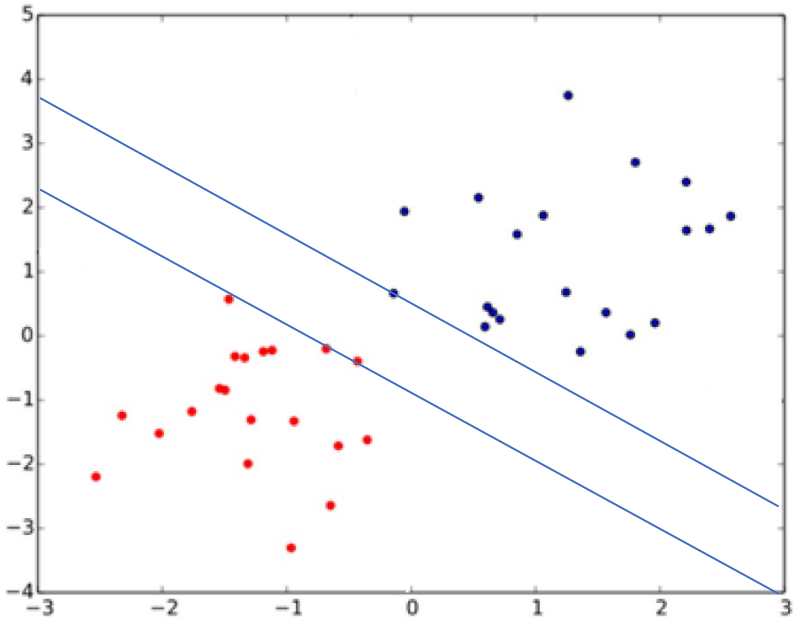


Perceptron model with activation function



aside: What is the relationship between SVMs and perceptrons?

SVMs attempt to learn the support vectors which maximizes the margin between classes.



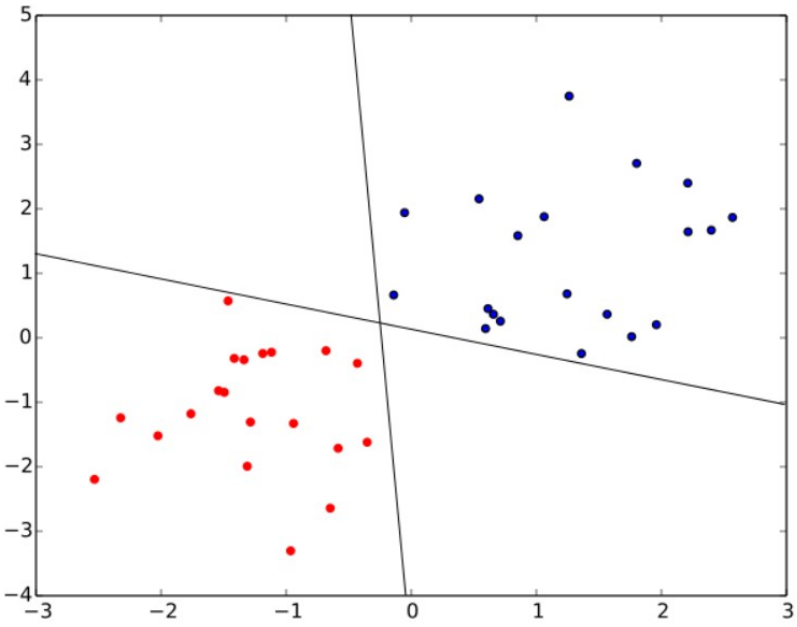
aside: What is the relationship between SVMs and perceptrons?

SVMs attempt to learn the support vectors which maximizes the margin between classes.

A perceptron does not.
Both of these perceptron classifiers are equivalent.

The 'Perceptron of optimal stability' is used in SVM:

Perceptron
+ optimal stability
+ kernel trick
= *foundations of SVM*



Perceptron model

- Use is grounded in theory
  - Universal approximation theorem (Goodfellow 6.4.1)

- Can represent a NAND circuit, from which any binary function can be built by compositions of NANDs.

=> With enough parameters, it can approximate any function.

Universal Approximation Theorem

A single-layer of perceptrons can learn any univariate function:

- So long as it is differentiable (continuous functions)
- To some approximation;
  More perceptrons = a better approximation
- Only for inputs within a specific range!

Visual proof (Michael Nielson):

http://neuralnetworksanddeeplearning.com/chap4.html

If a single-layer network can learn any function…
…given enough parameters…

…then why do we go deeper?

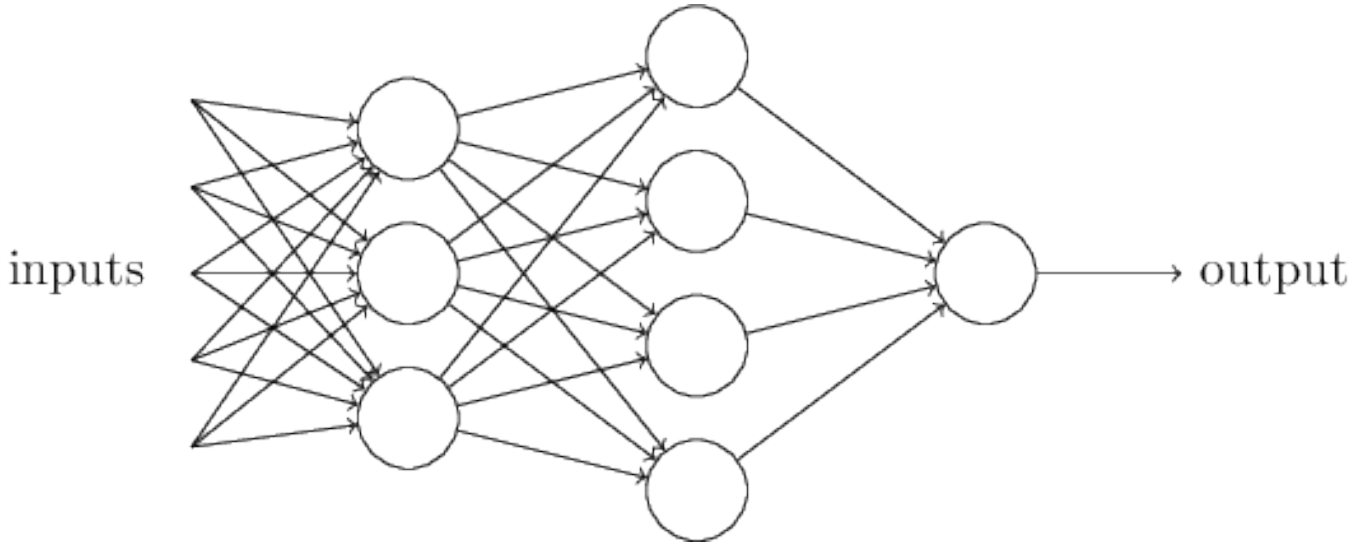
Intuitively, composition is efficient because it allows *reuse*.

Empirically, deep networks do a better job than shallow networks at learning such hierarchies of knowledge.

A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.

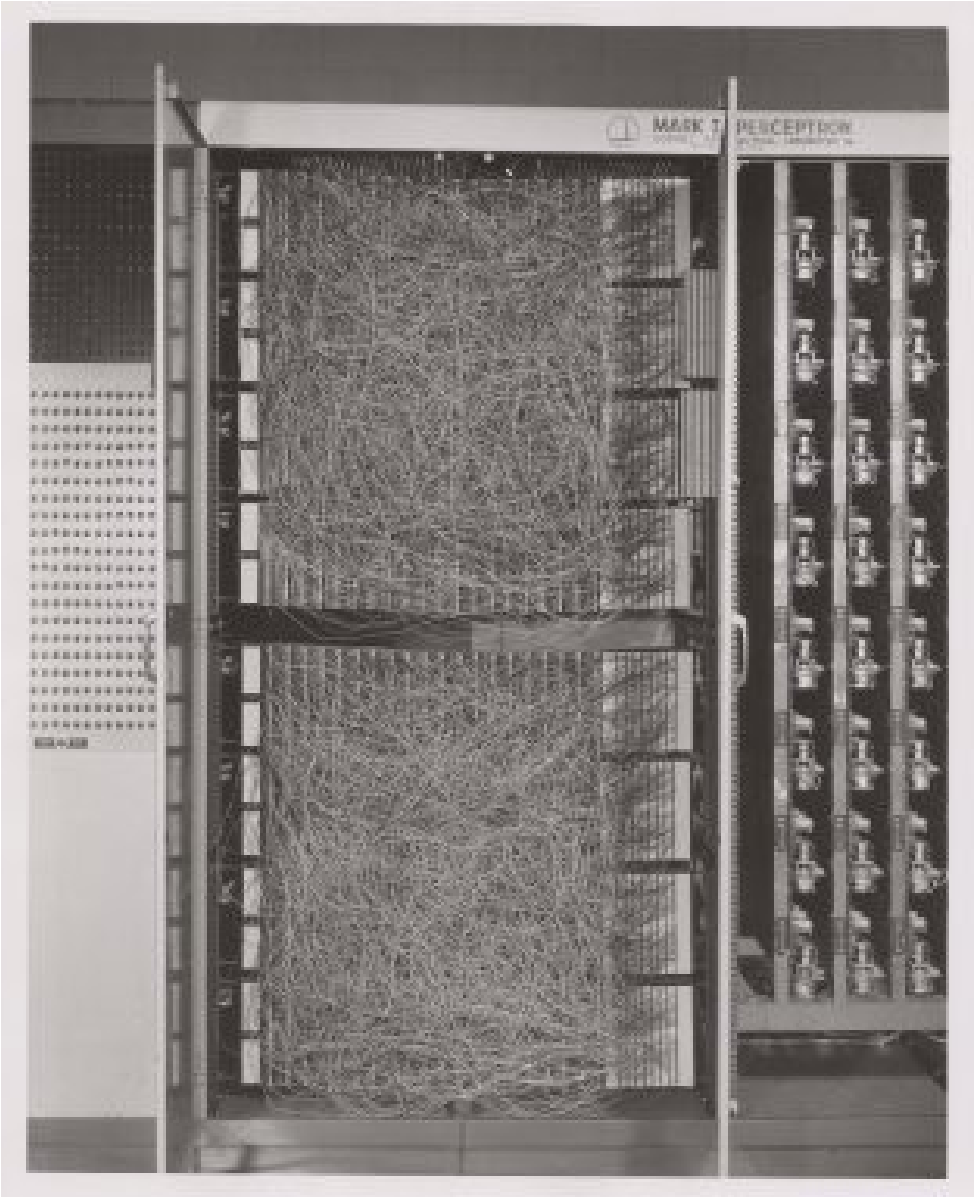
Multi-layer perceptron (MLP)

- …is a '*fully connected*' neural network with non-linear activation functions.



- '*Feed-forward*' neural network

Nielson



Mark 1 Perceptron c.1957

20x20 pixel camera feed

In a 1958 *The New York Times* reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

Wikipedia

We saw earlier step and sigmoid activation functions. Are there other activation functions?

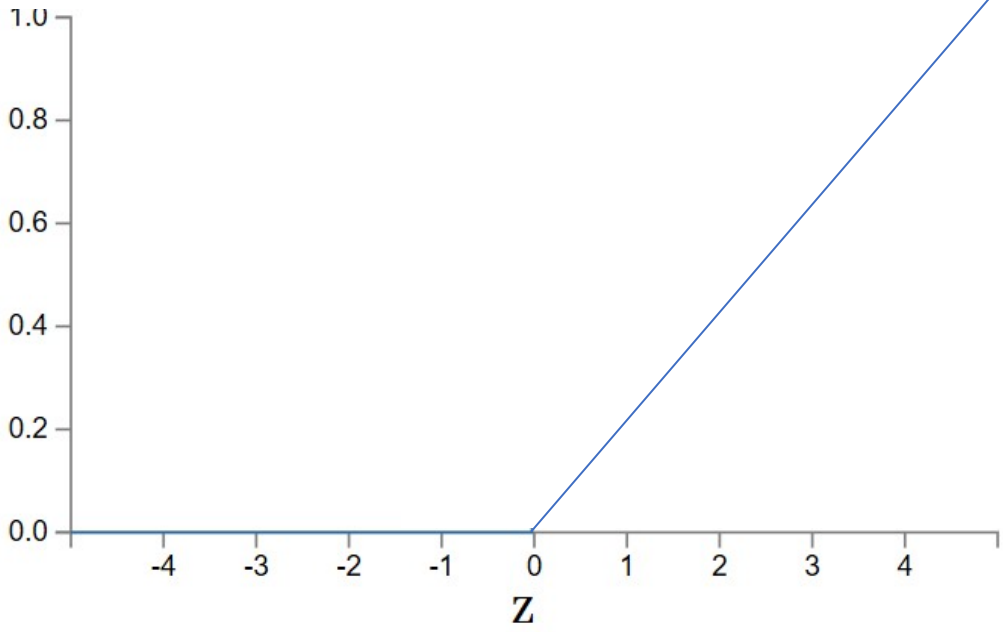
Yes, many: Linear - ReLU- Heaviside - Logistic (sigmoid) - hyperbolic tan (tanh) - Leaky ReLU, ... (a dozen more)

As long as:

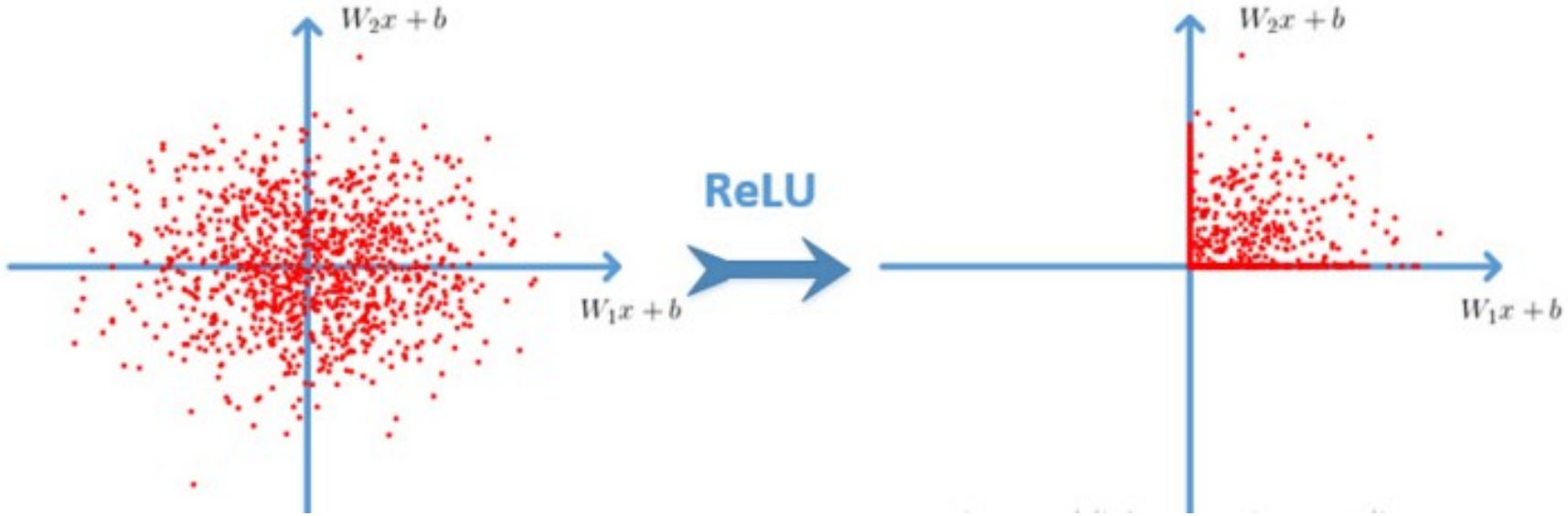
- Activation function s(z) is well-defined as z -> -∞ and z -> ∞
- These limits are different
- Monotone
- ideally Differentiable(not always the case, ex: ReLU not differentiable at 0)

Introducing non-linearities with activation functions Rectified Linear Unit

- ReLU $f(x) = \max(0, x)$



Software implementations of neural network training usually return one of the one-sided derivatives rather than reporting that the derivative is not defined or raising an error.

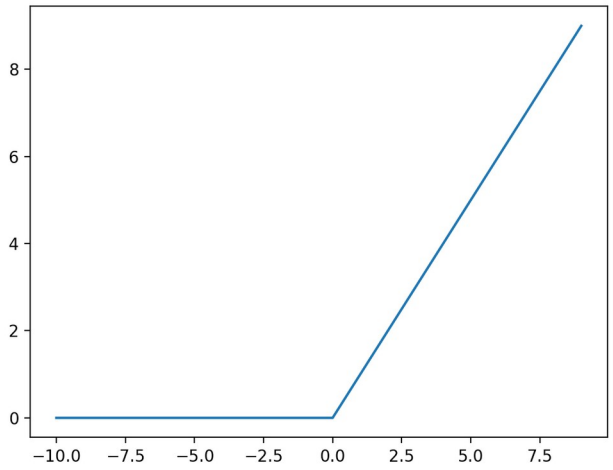

$W_2x + b$
$W_1x + b$
ReLU
$W_2x + b$
$W_1x + b$

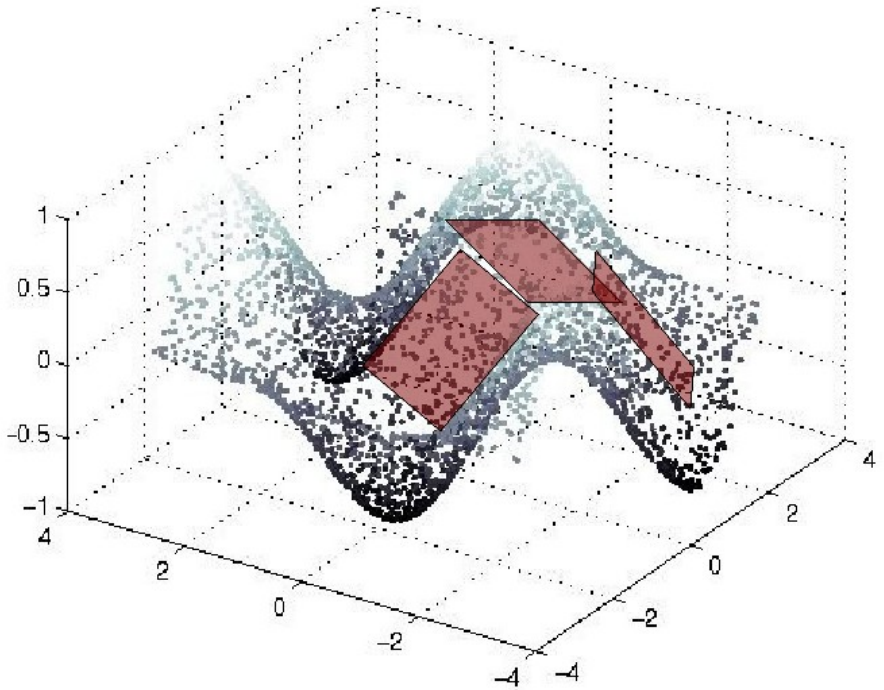
Rectified Linear Unit

Question: What do ReLU layers accomplish?

Answer: Piece-wise linear tiling: mapping is locally linear.

The subsequent layer to the activation layer is a learned (weighted) combination of the activation layer output, allowing for this:





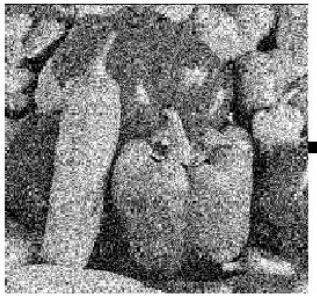
Supervised Learning: Examples

Classification

"dog"

classification

Denoising



regression

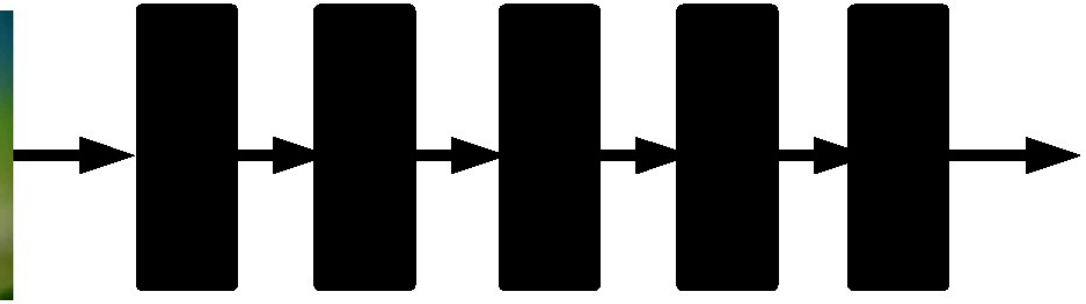
OCR

"2 3 4 5"

structured prediction

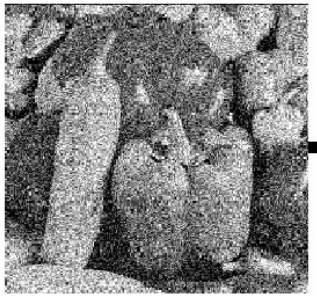
Supervised Deep Learning

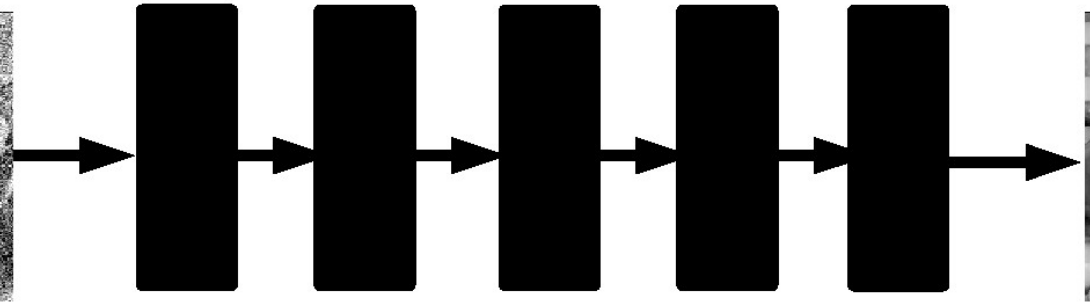
Classification



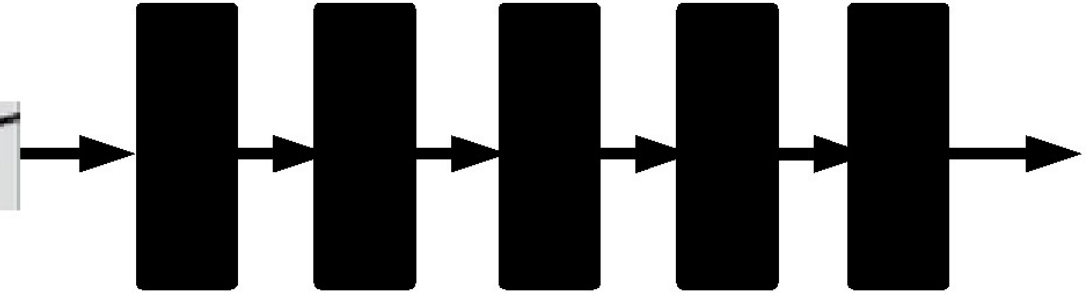
"dog"

Denoising





OCR



"2 3 4 5"

Ranzato

Goals

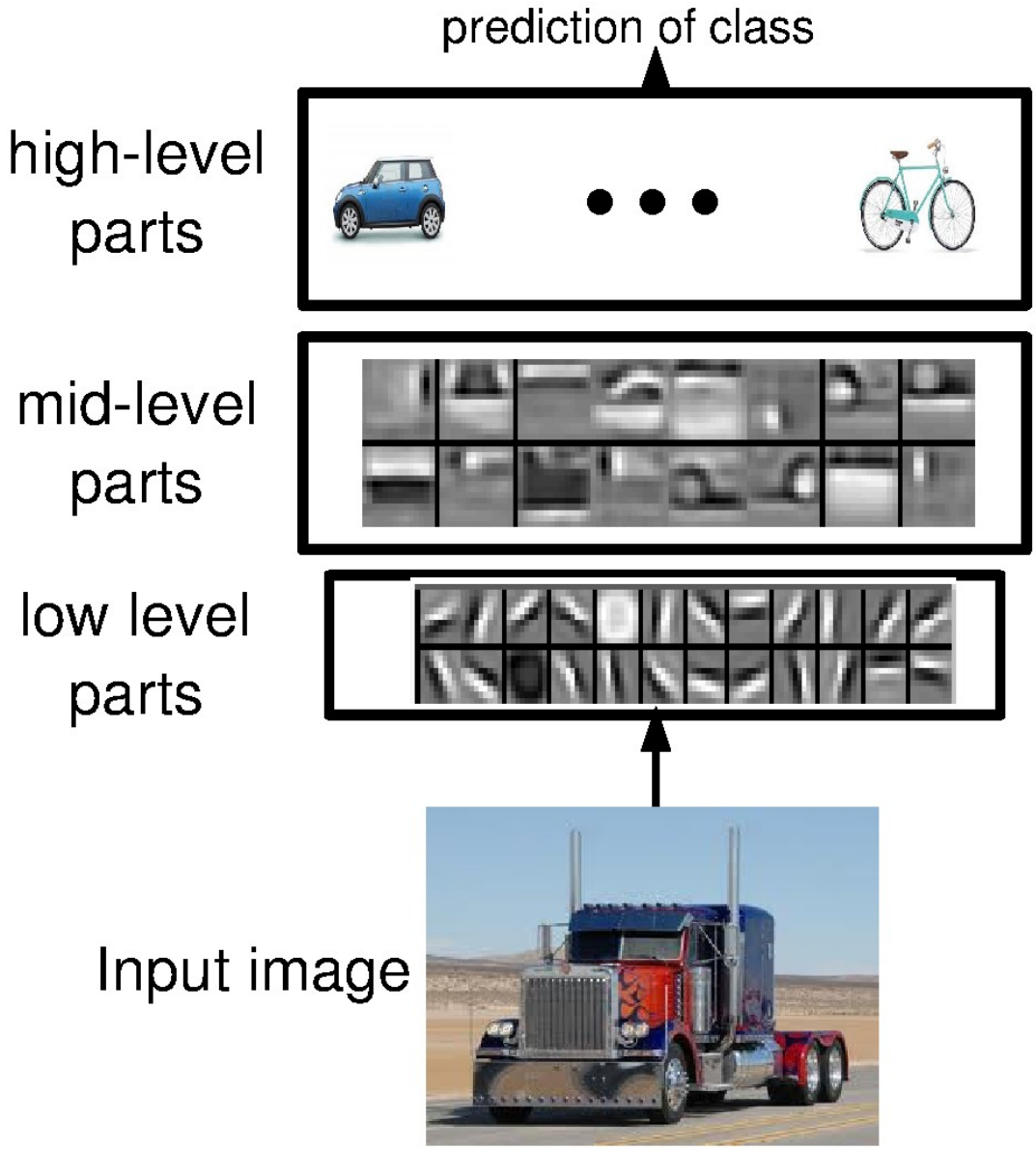
Build a classifier which is more powerful at representing complex functions *and* more suited to the learning problem.

What does this mean?

1. Assume that the *underlying data generating function* relies on a composition of factors.✓

2. Learn a feature representation that is specific to the dataset.

How the factor composition is taking place?



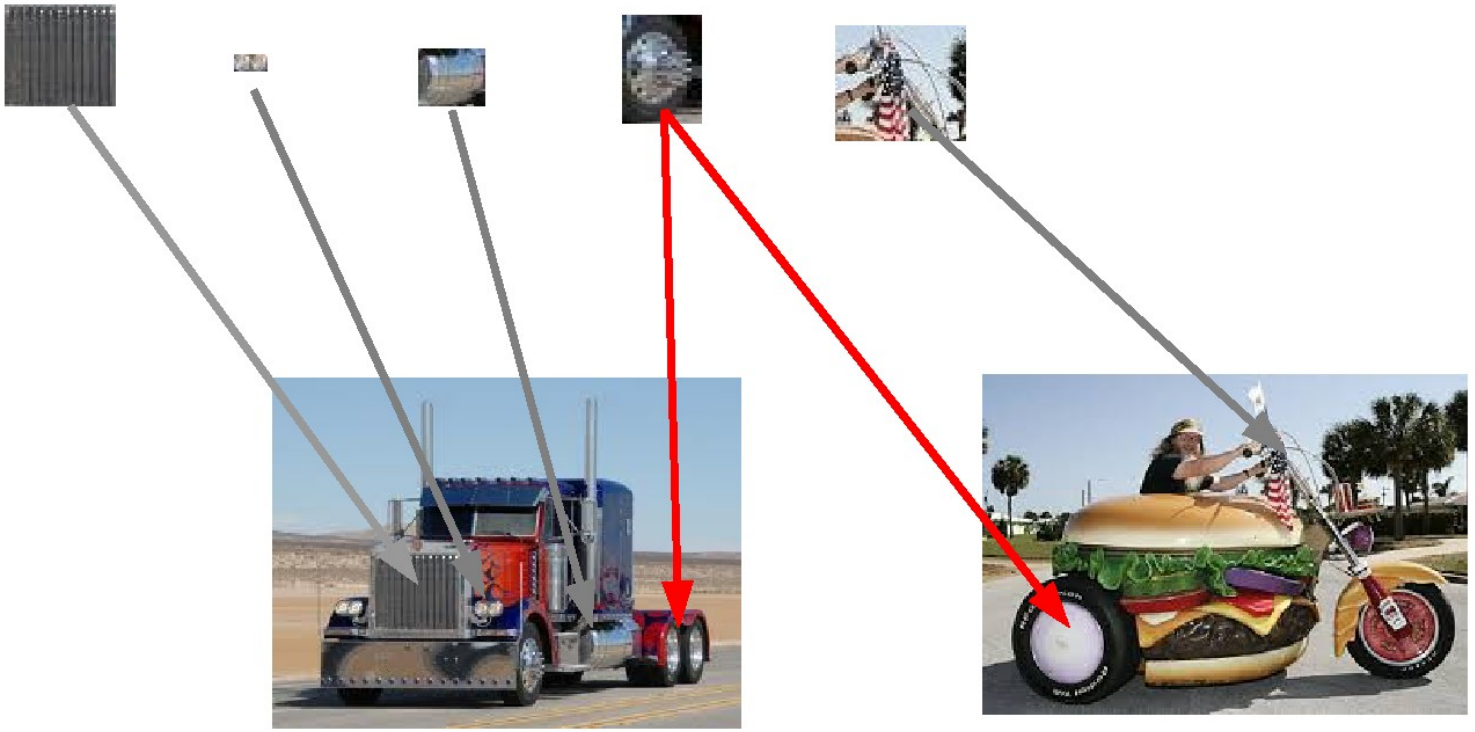
- distributed representations
- feature sharing
- compositionality

Lee et al. “Convolutional DBN's ...” ICML 2009

How the factor composition is taking place?

[1 1 0 0 0 1 0 **1** 0 0 0 0 1 1 0 1…] motorbike

[0 0 1 0 0 0 0 **1** 0 0 1 1 0 0 1 0 …] truck



Interpretation

Question: What does a hidden unit do?

Answer: It can be thought of as a classifier or feature detector.

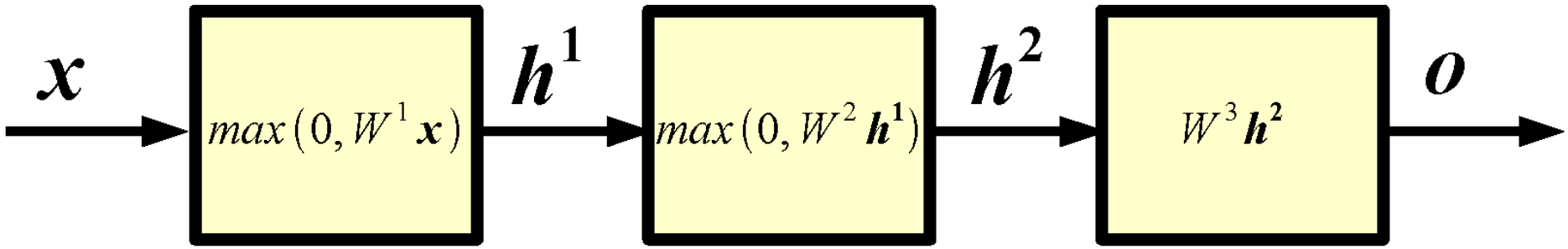
Question: How many layers? How many hidden units?

Answer: Cross-validation or hyper-parameter search methods are the answer. In general, the wider and the deeper the network the more complicated the mapping.

Question: How do I set the weight matrices?

Answer: Weight matrices and biases are learned.
First, we need to define a measure of quality of the current mapping.
Then, we need to define a procedure to adjust the parameters.

Neural Networks: example



x input

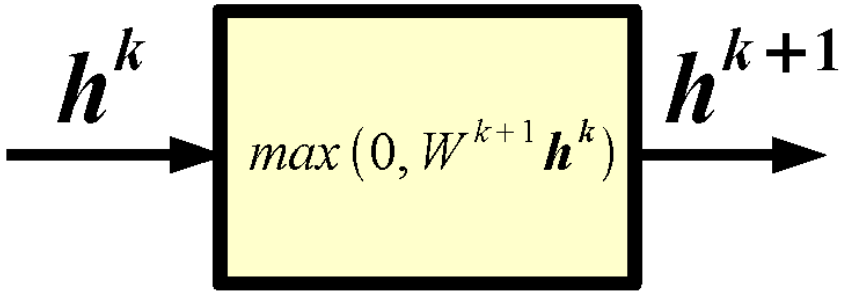
h^1 1-st layer hidden units

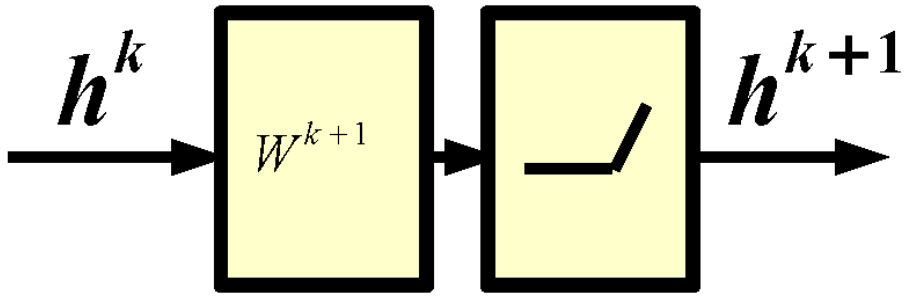
h^2 2-nd layer hidden units

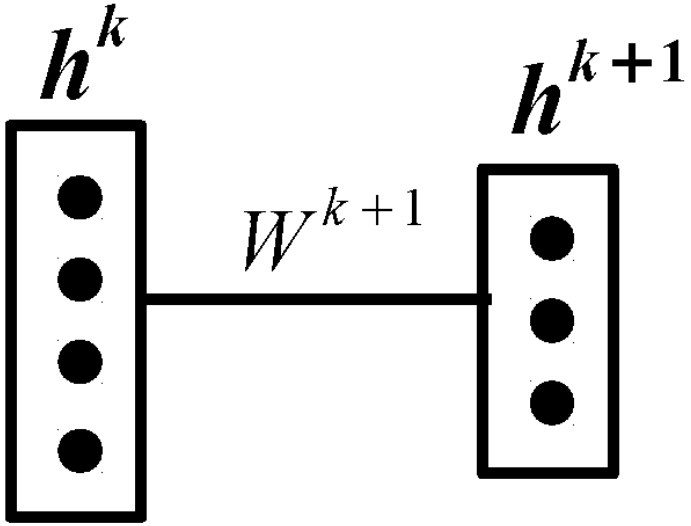
o output

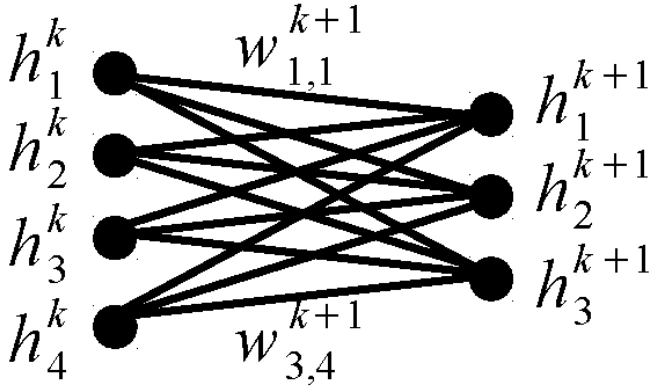
Example of a 2 hidden layer neural network (or 4 layer network, counting also input and output).

Alternative Graphical Representation







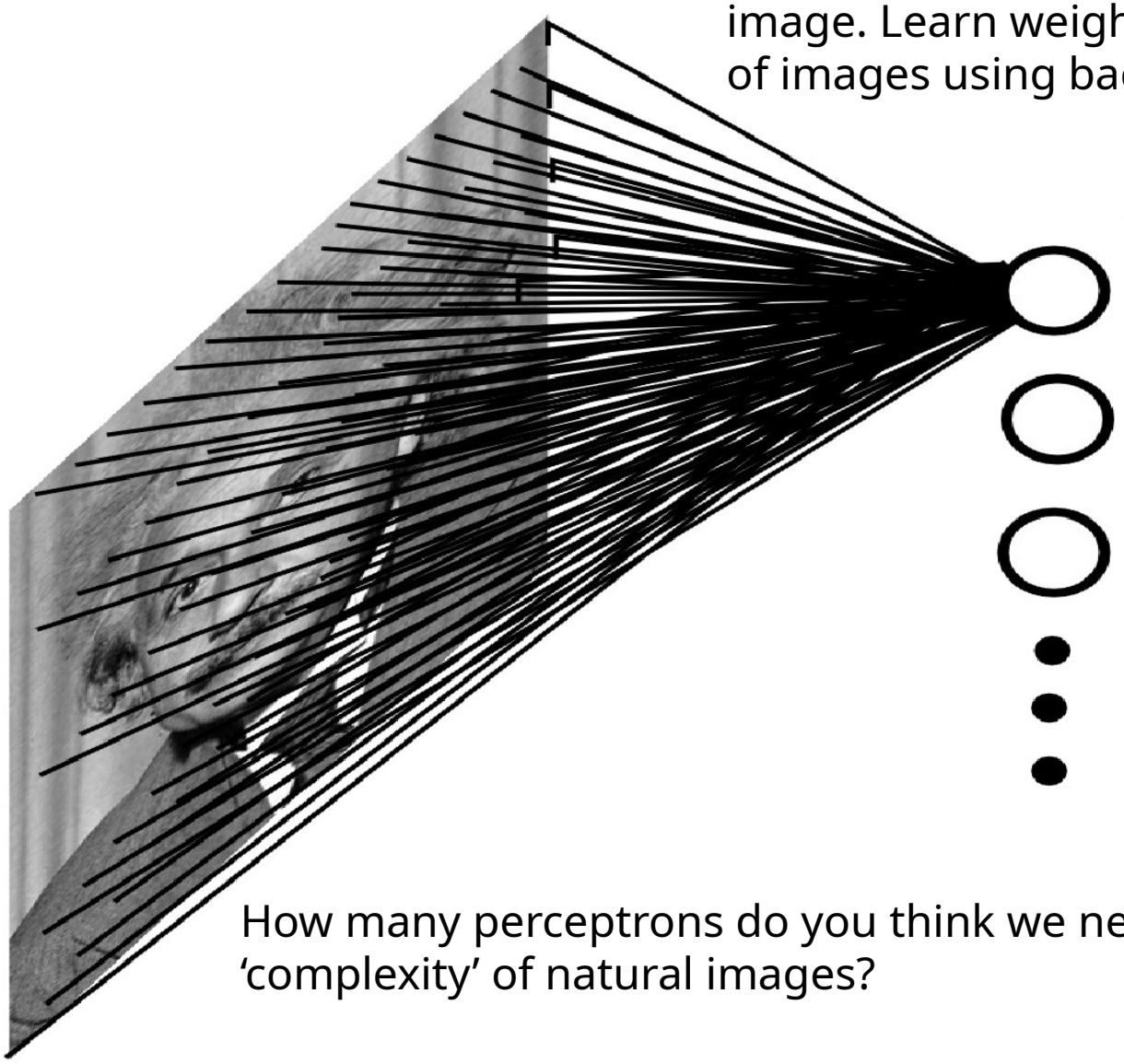


Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples

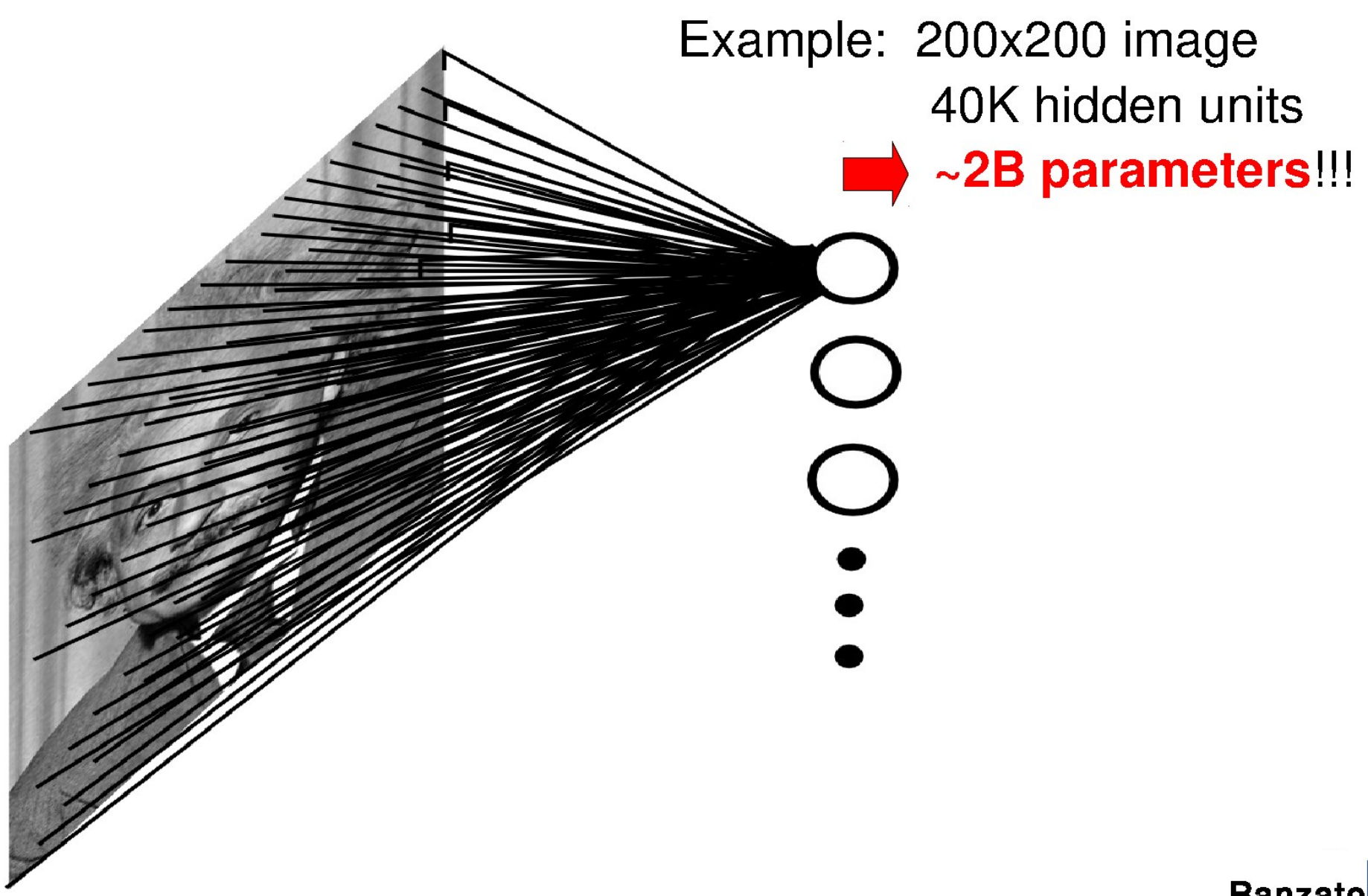
Images as input to neural networks

Going to learn a neural network to classify an image. Learn weights from a large database of images using back propagation.

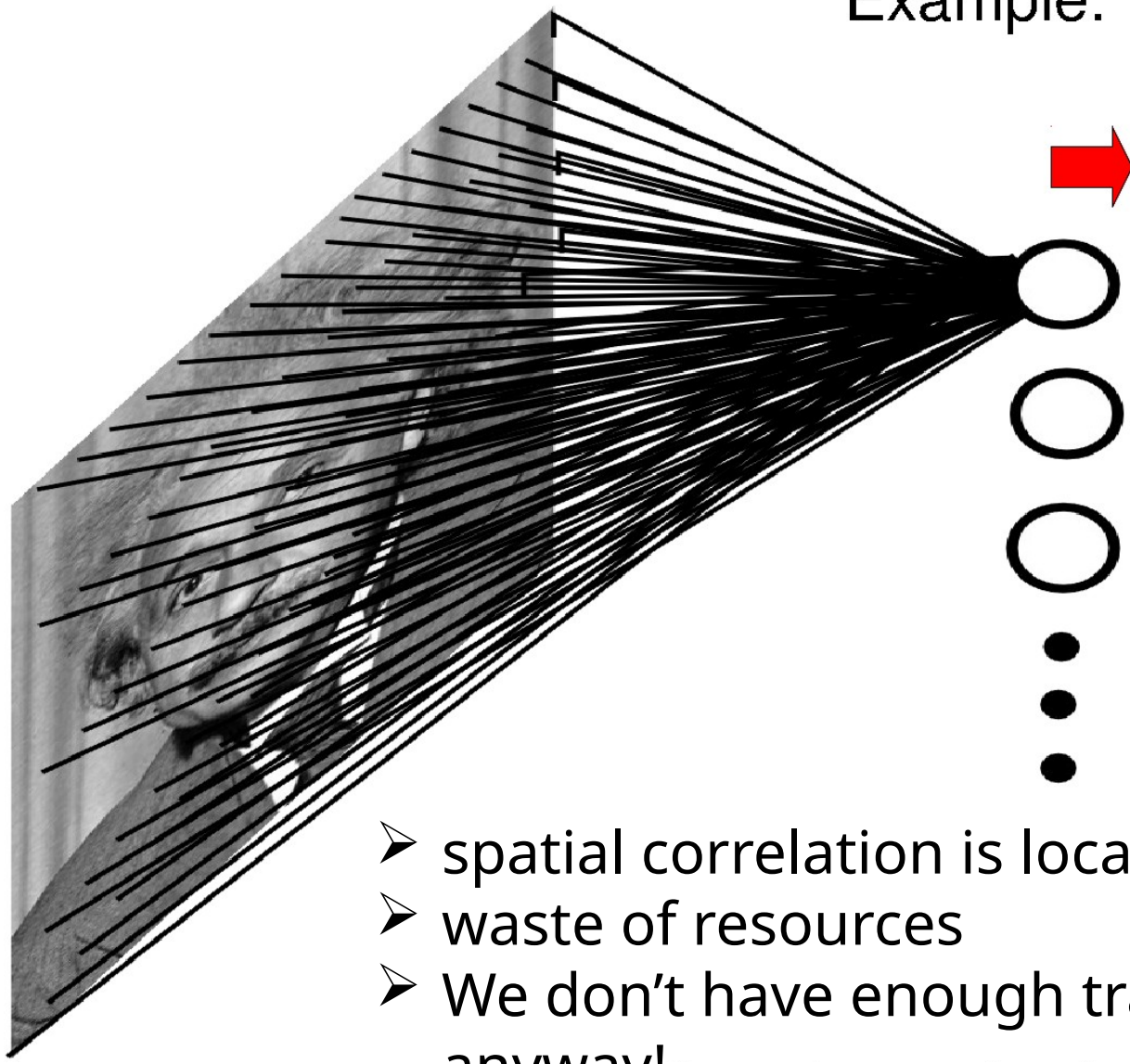


How many perceptrons do you think we need to represent the ‘complexity’ of natural images?

Images as input to neural networks



Images as input to neural networks



Example: 200x200 image

40K hidden units

~2B parameters!!!

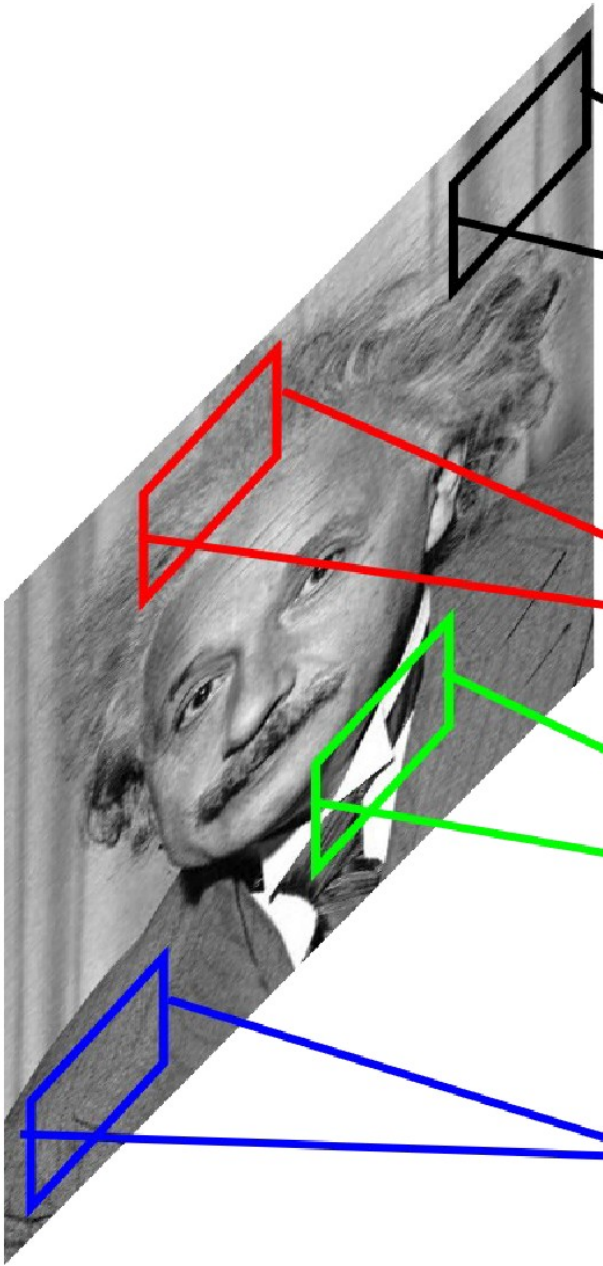
- spatial correlation is local
- waste of resources
- We don't have enough training samples anyway!

Ranzato

Motivation

- Sparse interactions – *receptive fields*
  - Assume that in an image, we care about ‘local neighborhoods’ only for a given neural network layer.
  - Composition of layers will expand local -> global.

Filter/Kernel/Receptive field: input patch which the hidden unit is connected to.



STATIONARITY? Statistics is similar at different locations

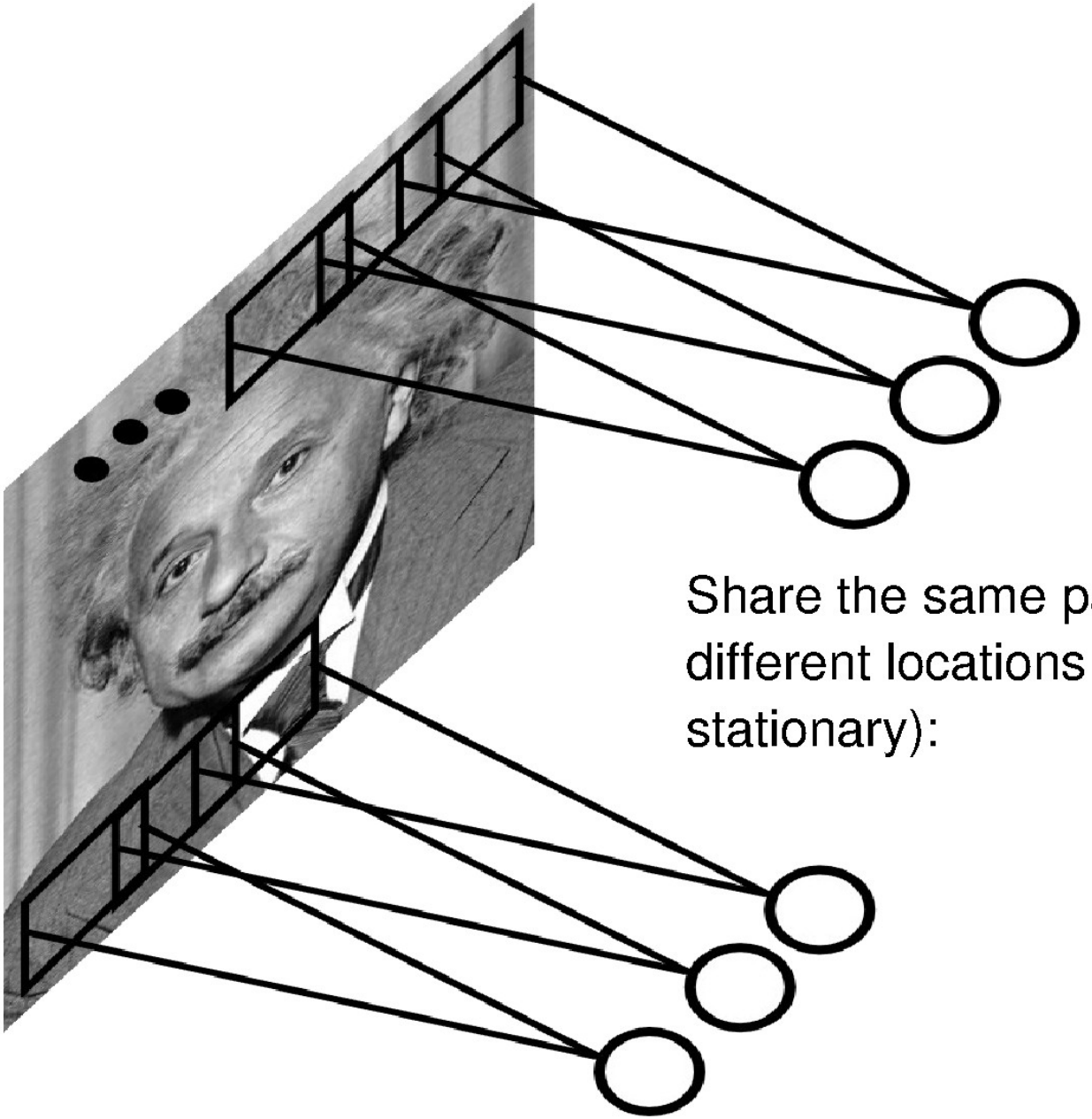
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Motivation

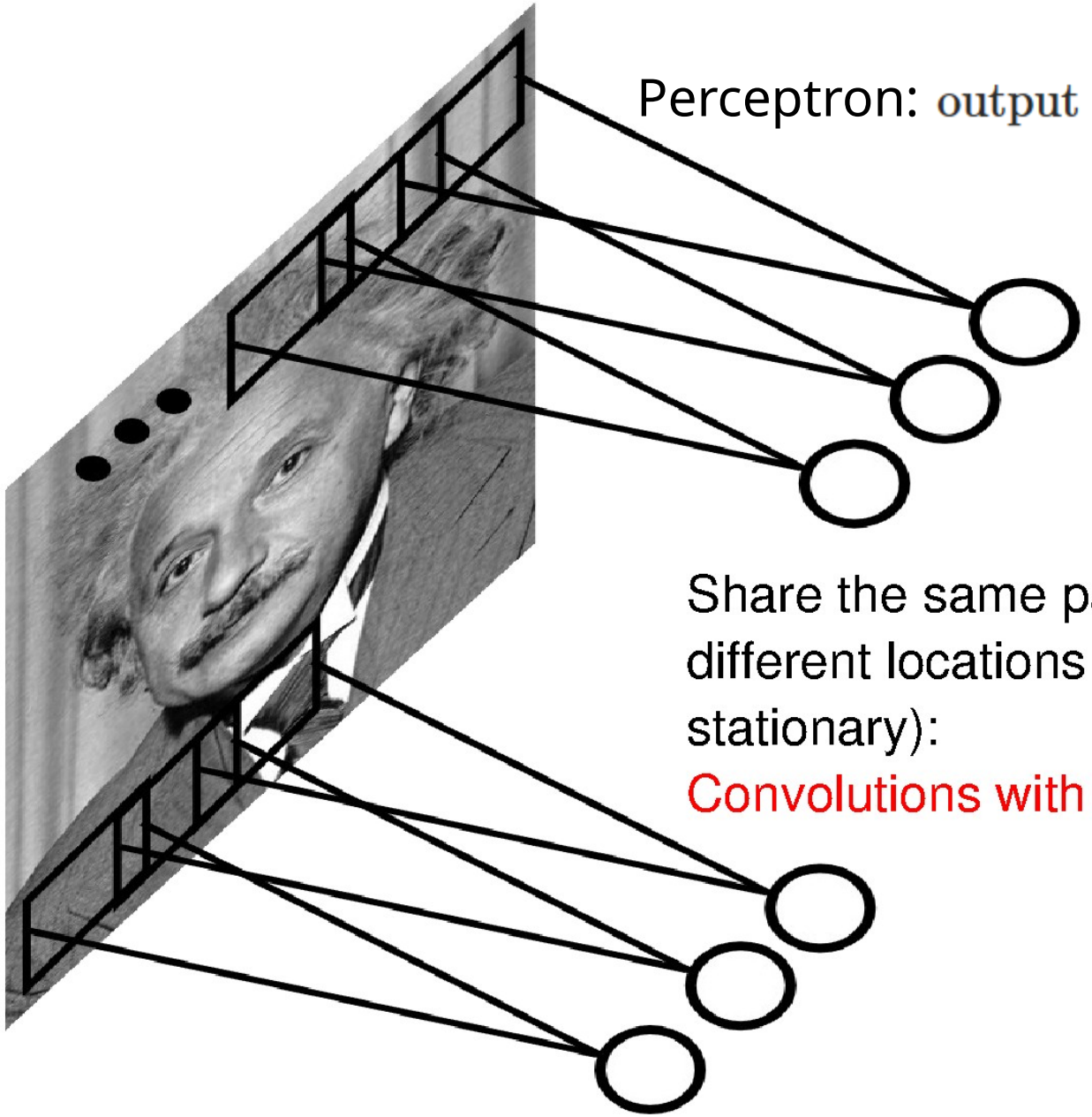
- Sparse interactions – *receptive fields*
  - Assume that in an image, we care about ‘local neighborhoods’ only for a given neural network layer.
  - Composition of layers will expand local -> global.

- Parameter sharing
  - ‘Tied weights’ – use same weights for more than one perceptron in the neural network.
  - Leads to *equivariant representation*
    - If input changes (e.g., translates), then output changes similarly



Share the same parameters across different locations (assuming input is stationary):

Convolutional Layer



Perceptron: $\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$

$w \cdot x \equiv \sum_j w_j x_j$

This is convolution!

Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

Filtering reminder:
Correlation (rotated convolution)

$f[\cdot,\cdot]$

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$I[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

Credit: S. Seitz

Wait, why isn't it called a correlation neural network?

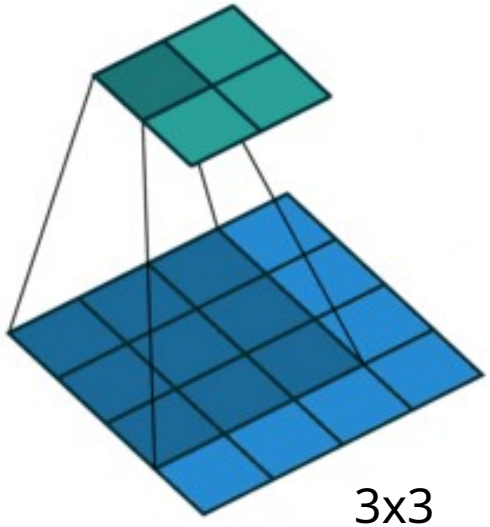
It could be.

Deep learning libraries actually implement correlation.

Correlation relates to convolution via a 180deg rotation of the kernel. When we *learn* kernels, we could easily learn them flipped.

Associative property of convolution ends up not being important to our application, so we just ignore it.

[p.323, Goodfellow]

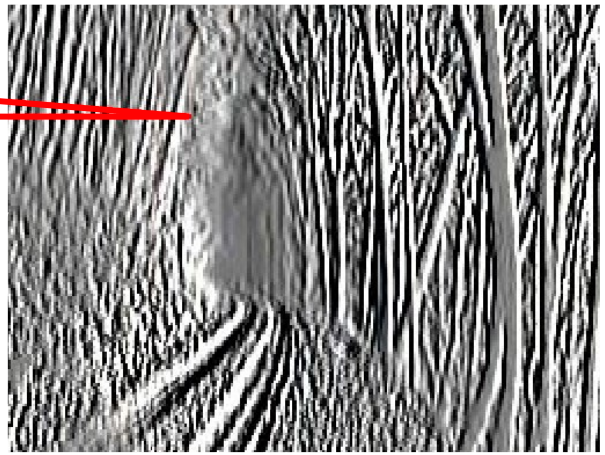


3x3
kernel

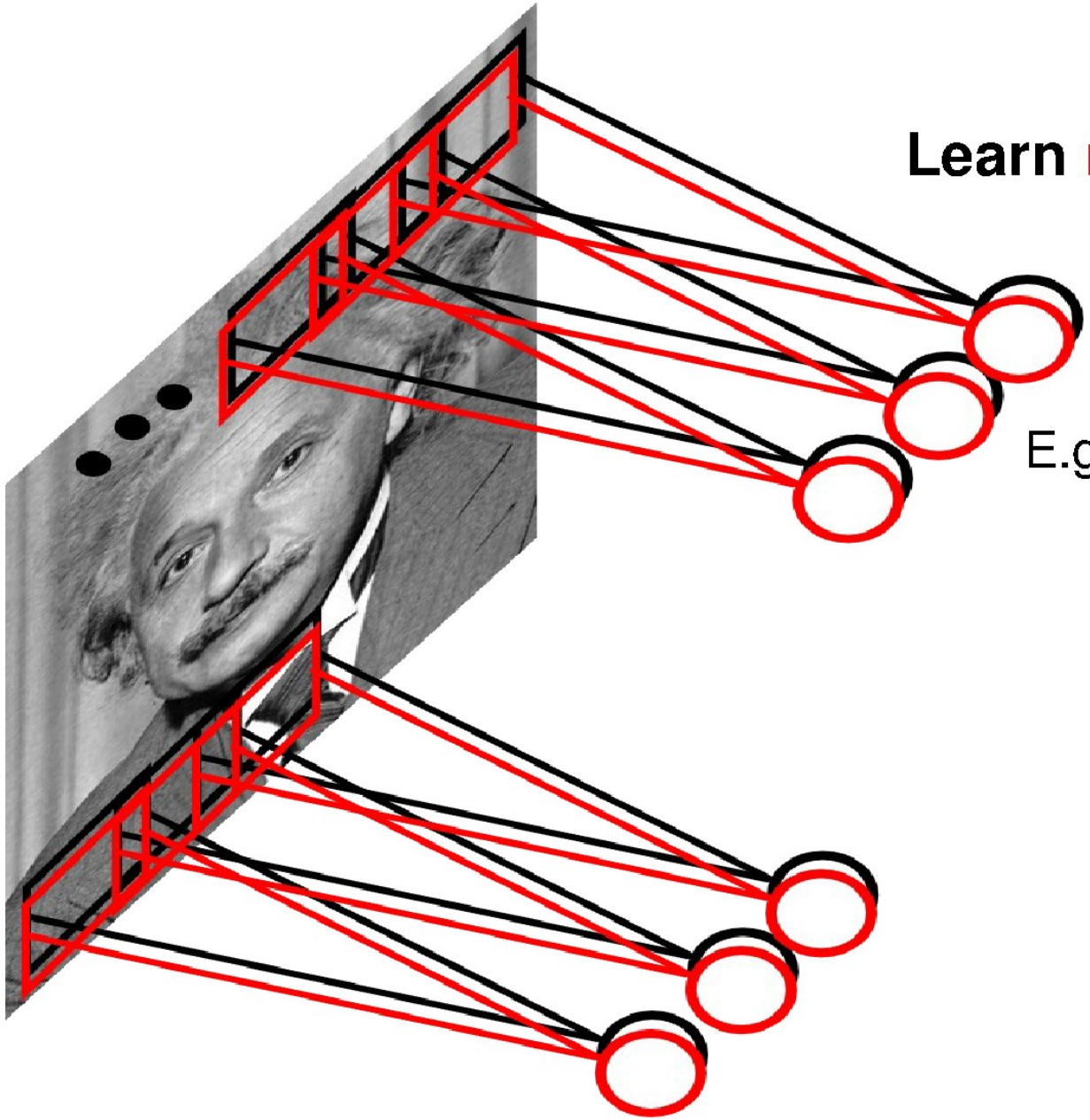
Convolution

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

Shared
weights



Convolutional Layer



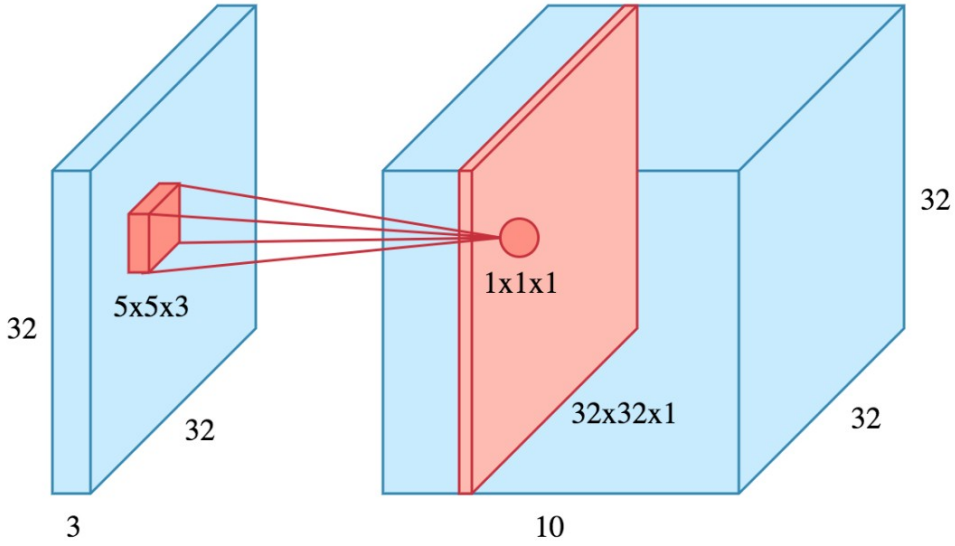
Learn multiple filters.

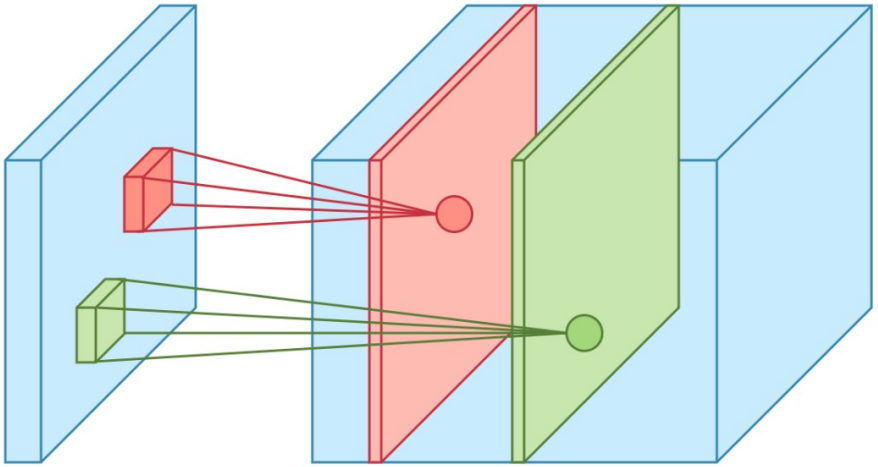
Filter = 'local' perceptron.
Also called *kernel.*

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

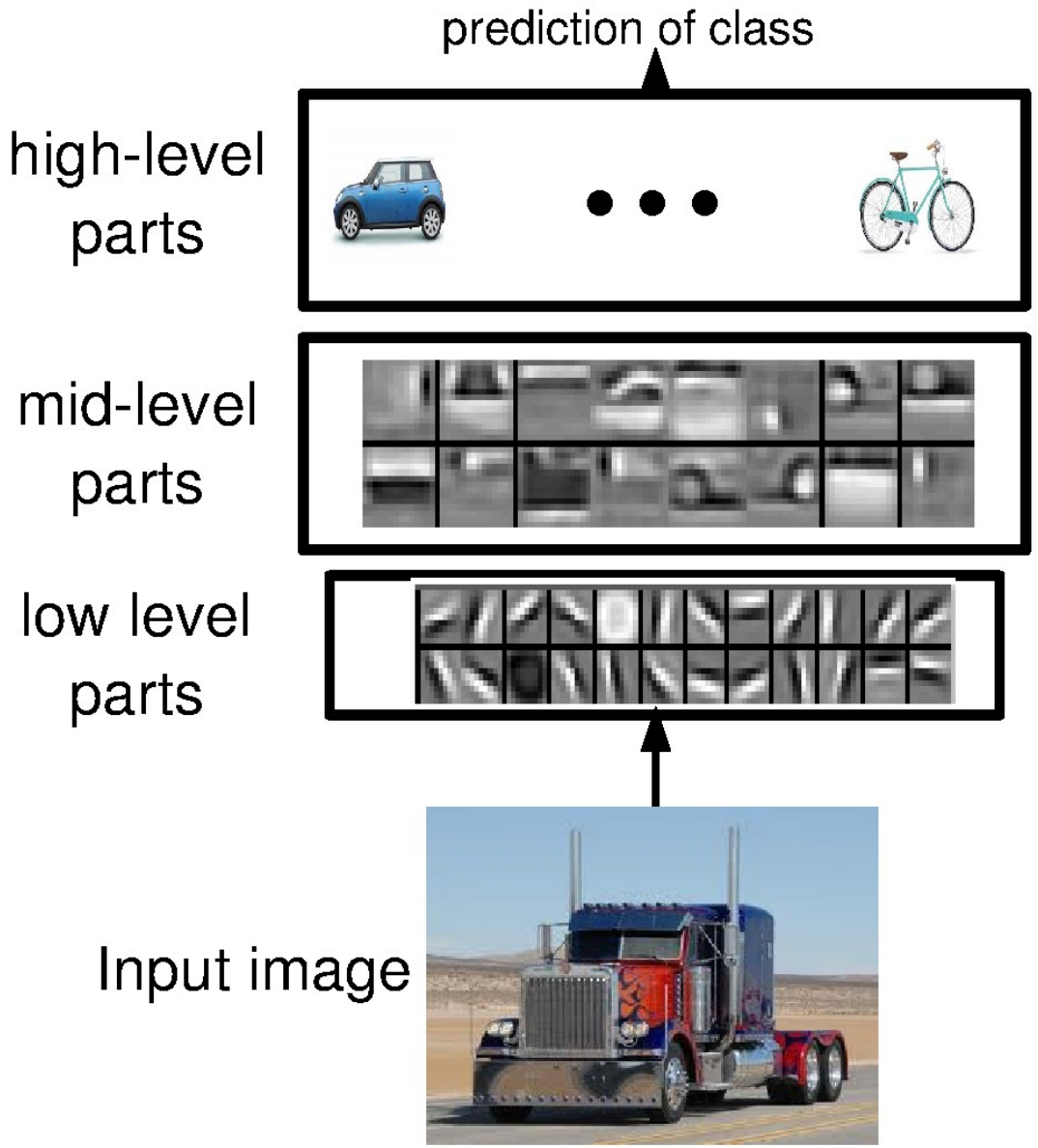
Ranzato

Learning multiple filters





Interpretation



- distributed representations
- feature sharing
- compositionality

Lee et al. “Convolutional DBN's ...” ICML 2009

Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

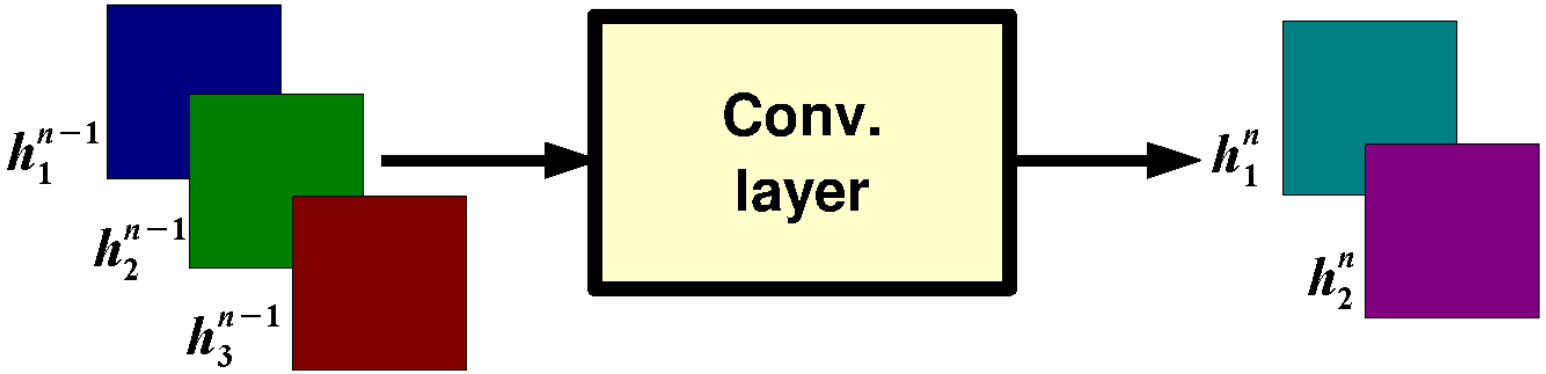
output feature map

input feature map

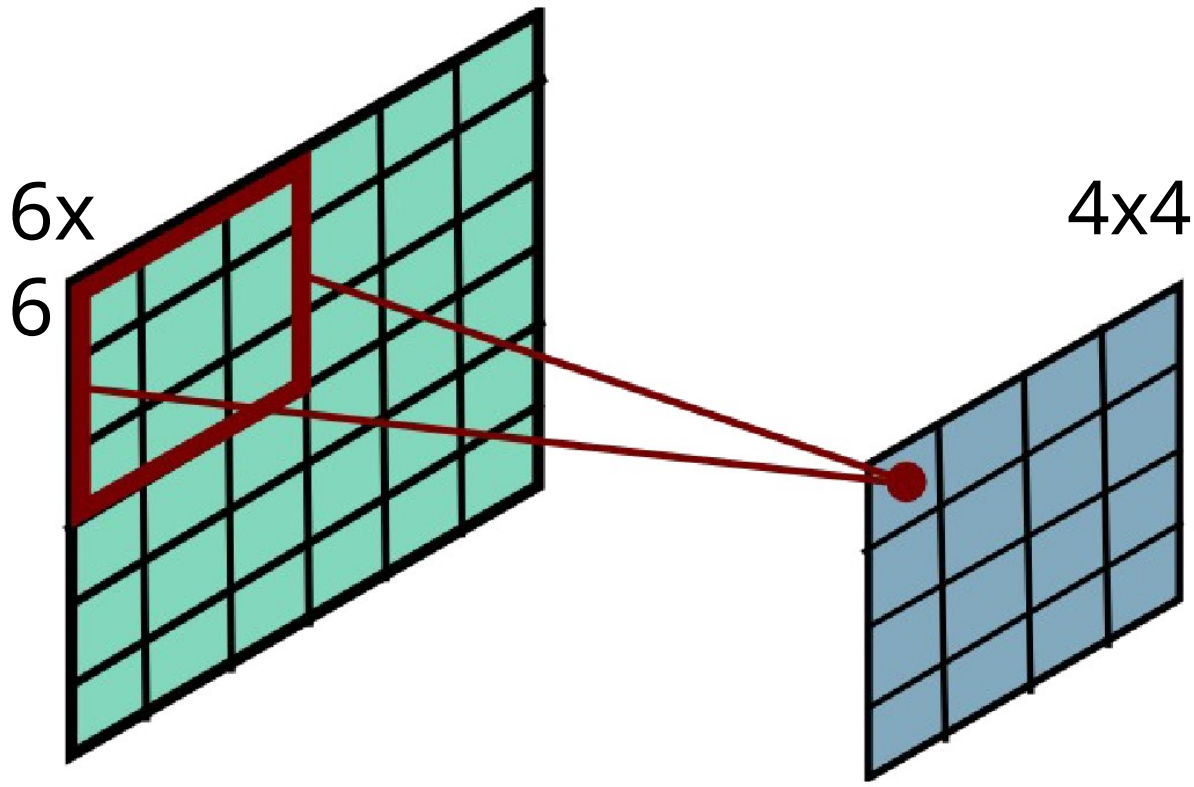
kernel

n = layer number
K = kernel size
j = # channels (input)
or # filters (depth)

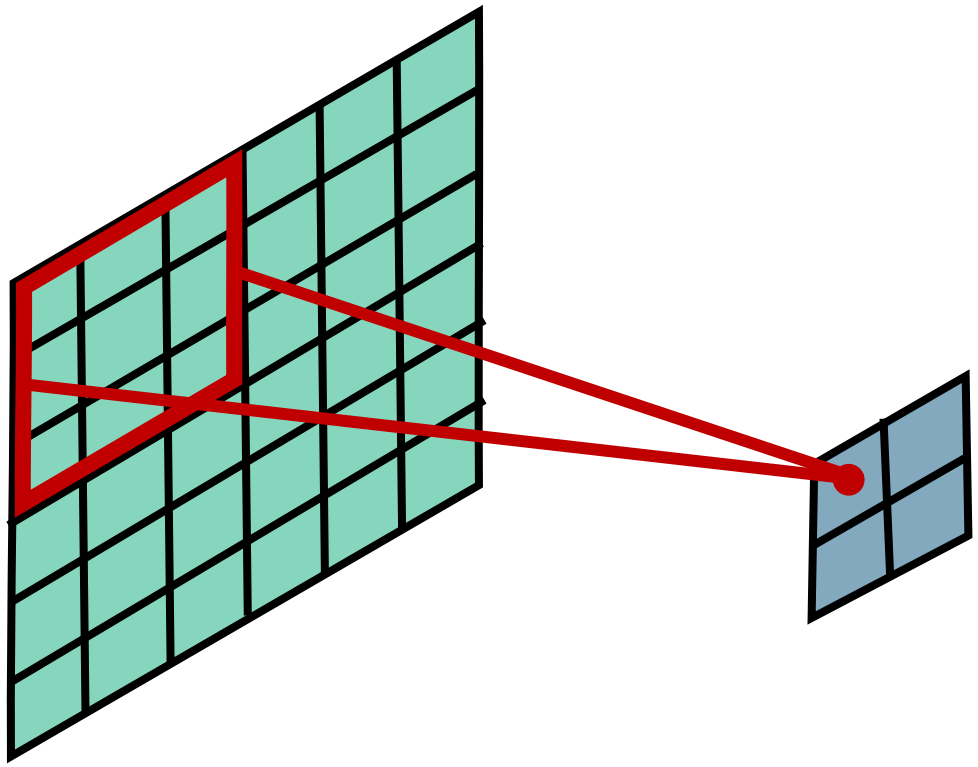
Activation function= relu



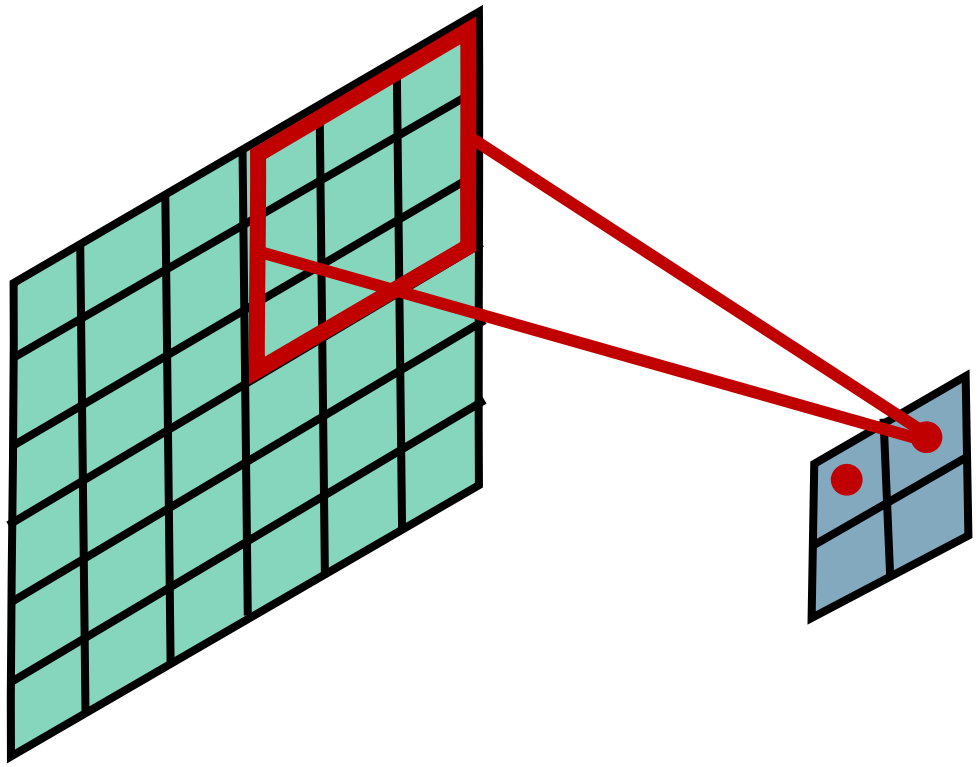
Stride = 1



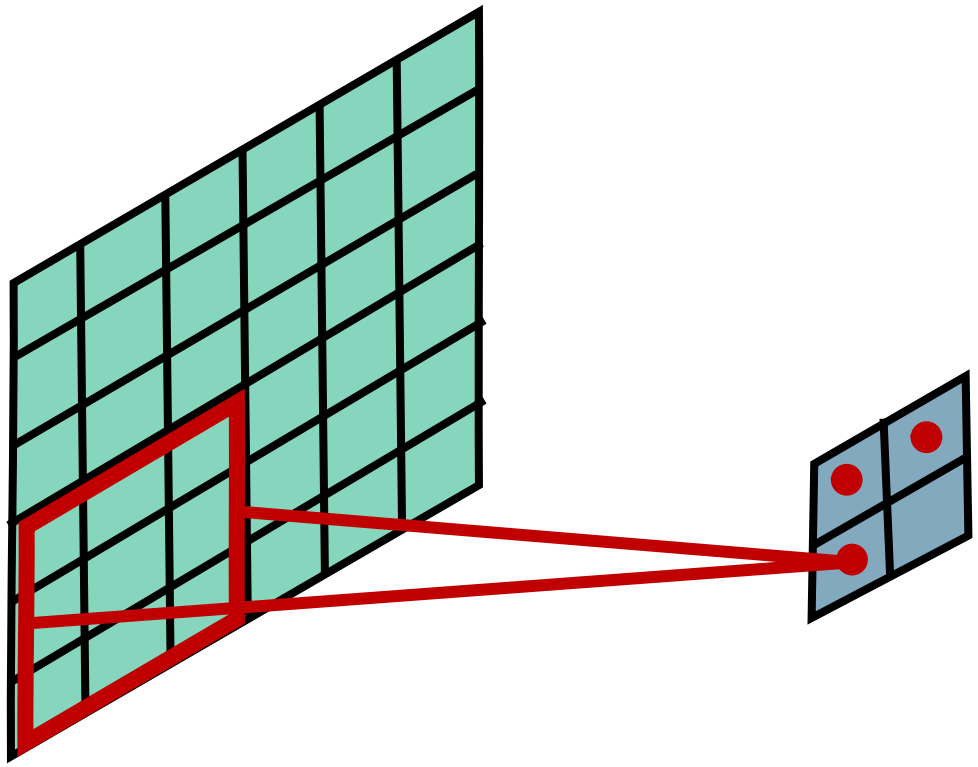
Stride = 3



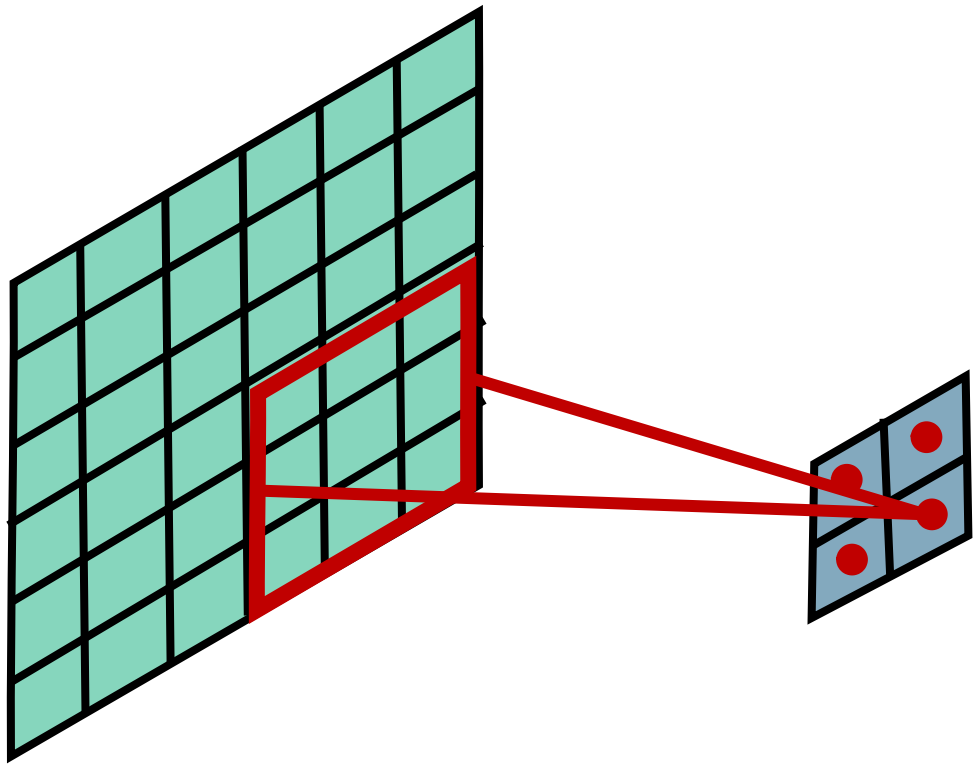
Stride = 3



Stride = 3



Stride = 3



Convolutional Layer

Question: What is the size of the output? What's the computational cost?

Answer: It is proportional to the number of filters and depends on the stride. If kernels have size KxK, input has size DxD, stride is 1, and there are M input feature maps and N output feature maps then:
- the input has size M@DxD
- the output has size N@(D-K+1)x(D-K+1)
- the kernels have MxNxKxK coefficients (which have to be learned)
- cost: M*K*K*N*(D-K+1)*(D-K+1)

Question: How many feature maps? What's the size of the filters?

Answer: Usually, there are more output feature maps than input feature maps. Convolutional layers can increase the number of hidden units by big factors (and are expensive to compute).
The size of the filters has to match the size/scale of the patterns we want to detect (task dependent).

Key Ideas

A standard neural net applied to images:

- scales quadratically with the size of the input
- does not leverage stationarity

Solution:

- connect each hidden unit to a small patch of the input
- share the weight across space

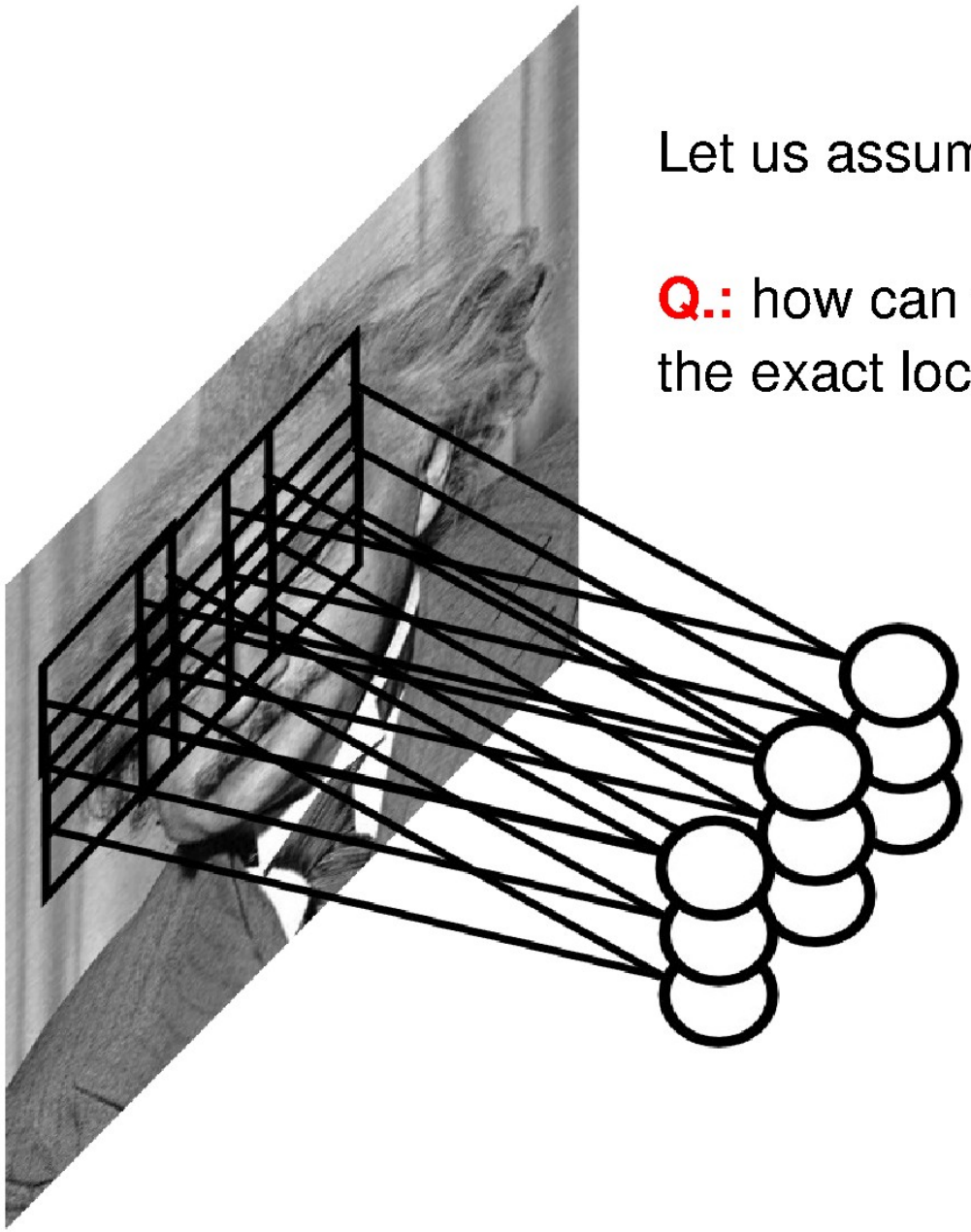
This is called: **convolutional layer.**

A network with convolutional layers is called **convolutional network.**

- add multiple filters per layer

LeCun et al. “Gradient-based learning applied to document recognition” IEEE 1998

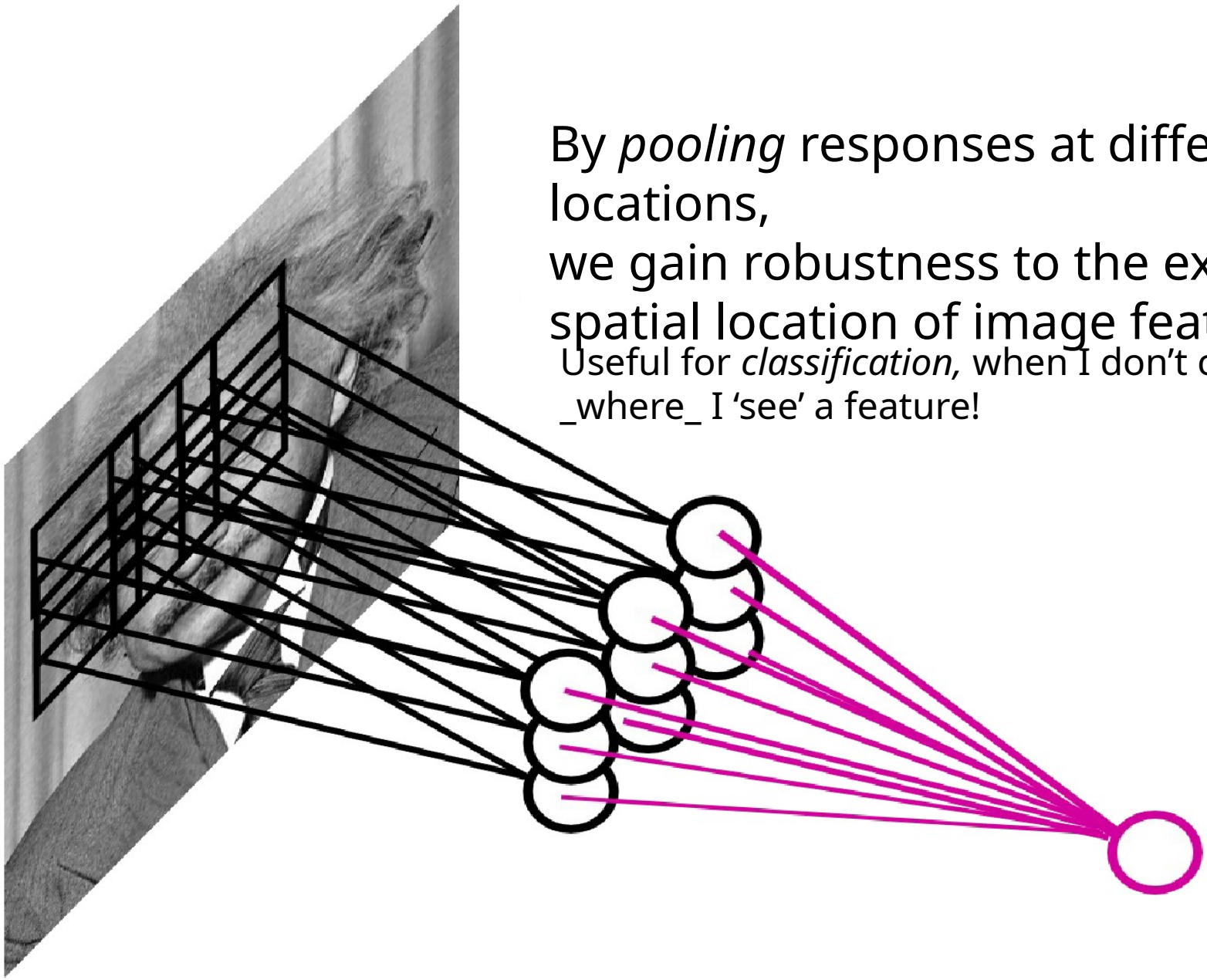
Pooling Layer



Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

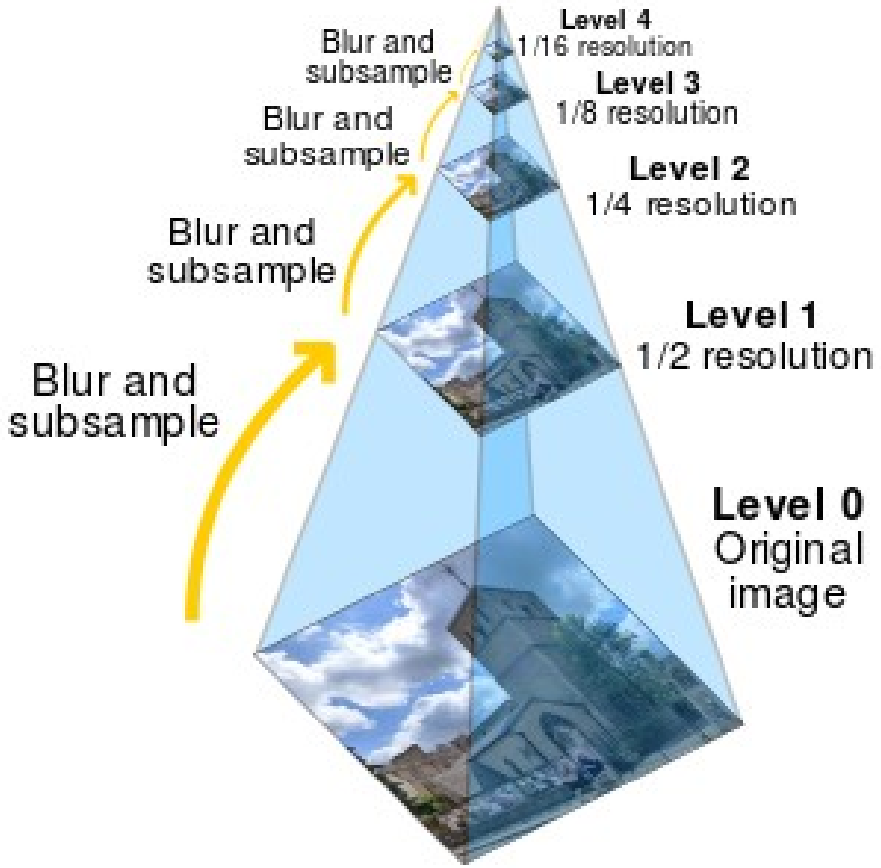
Pooling Layer



By *pooling* responses at different locations,
we gain robustness to the exact spatial location of image features.

Useful for *classification,* when I don't care about _where_ I 'see' a feature!

Pooling is similar to downsampling



…except sometimes we don’t want to blur, as other functions might be better for classification

Pooling Layer: Examples

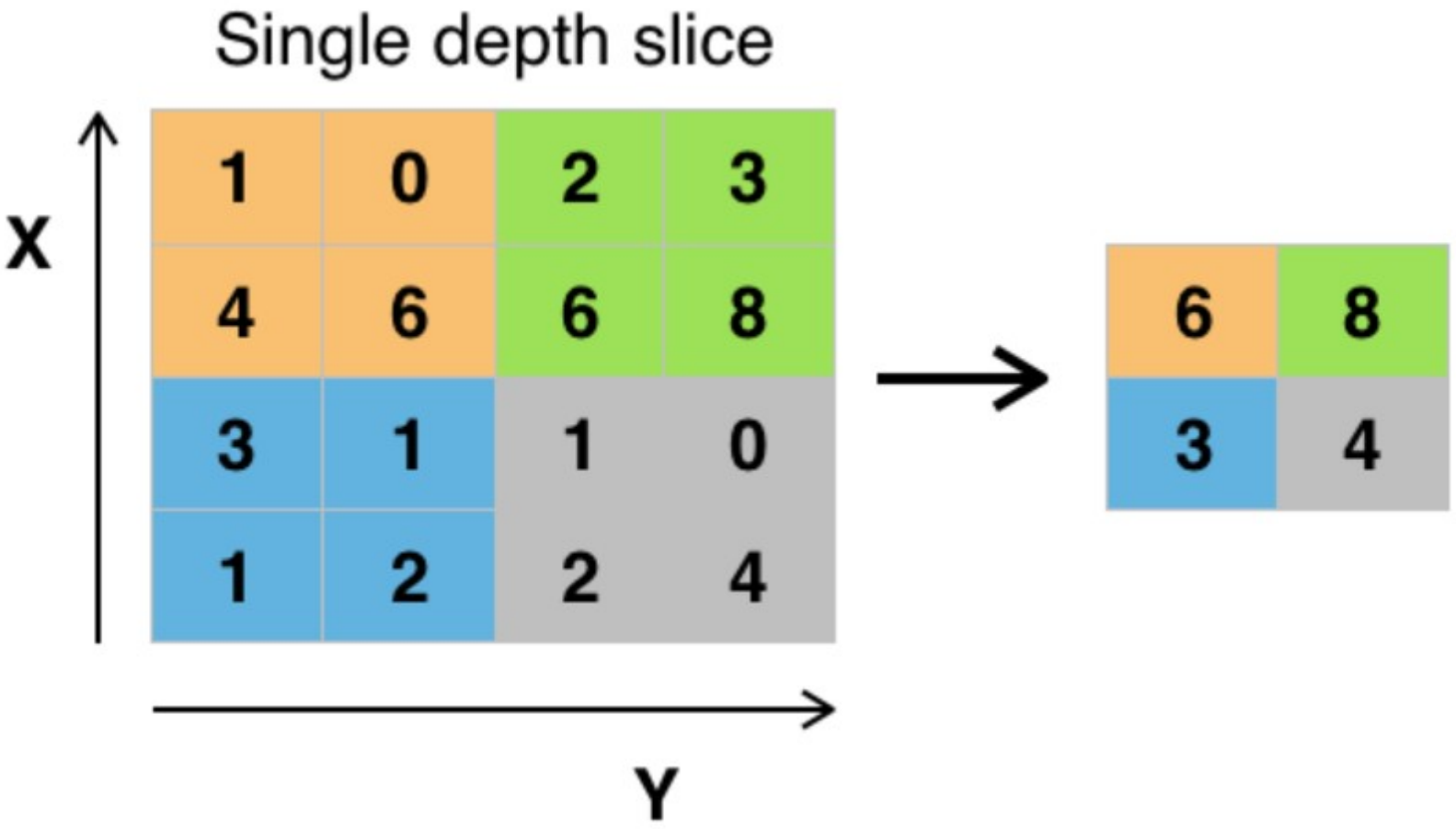
Max-pooling:

$$h_j^n(x, y) = max_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Max pooling



Wikipedi

Pooling Layer: Examples

Max-pooling:

$$h_j^n(x,y) = max_{\bar{x}\in N(x),\, \bar{y}\in N(y)} h_j^{n-1}(\bar{x},\bar{y})$$

Average-pooling:

$$h_j^n(x,y) = 1/K \sum_{\bar{x}\in N(x),\, \bar{y}\in N(y)} h_j^{n-1}(\bar{x},\bar{y})$$

L2-pooling:

$$h_j^n(x,y) = \sqrt{\sum_{\bar{x}\in N(x),\, \bar{y}\in N(y)} h_j^{n-1}(\bar{x},\bar{y})^2}$$

L2-pooling over features:

$$h_j^n(x,y) = \sqrt{\sum_{k\in N(j)} h_k^{n-1}(x,y)^2}$$

Pooling Layer

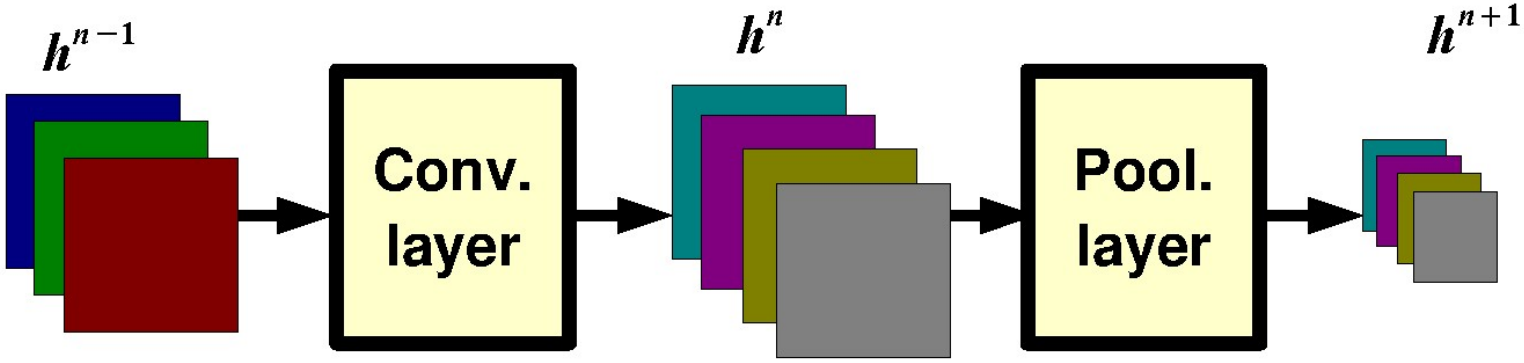
Question: What is the size of the output? What's the computational cost?

Answer: The size of the output depends on the stride between the pools. For instance, if pools do not overlap and have size KxK, and the input has size DxD with M input feature maps, then:
- output is M@(D/K)x(D/K)
- the computational cost is proportional to the size of the input (negligible compared to a convolutional layer)

Question: How should I set the size of the pools?

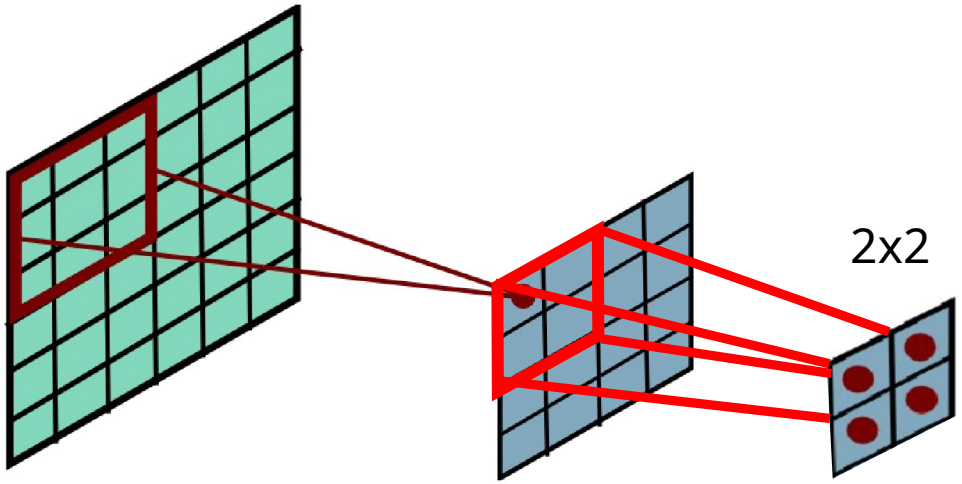
Answer: It depends on how much “invariant” or robust to distortions we want the representation to be. It is best to pool slowly (via a few stacks of conv-pooling layers).

Pooling Layer: Receptive Field Size



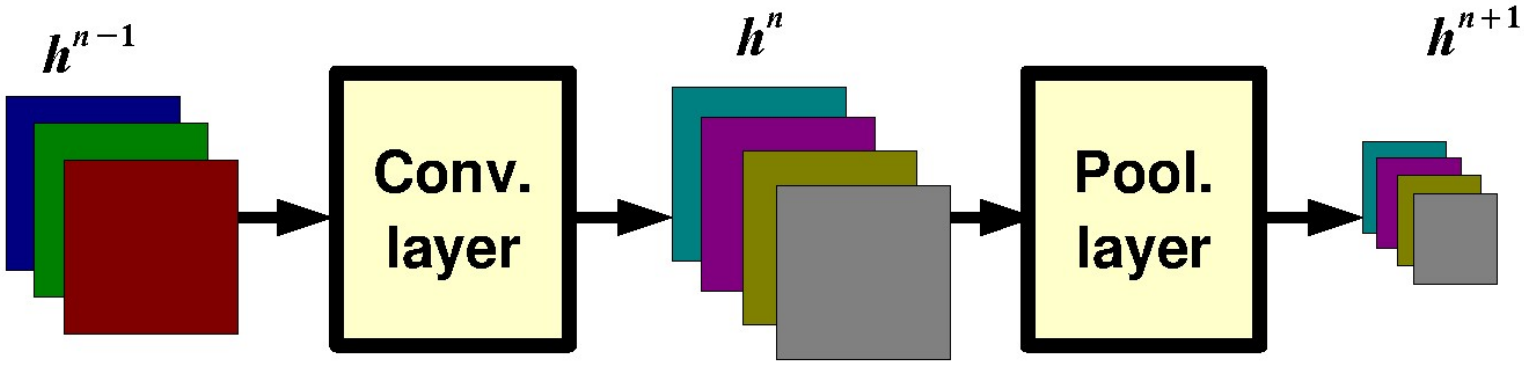
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size:

(P+k-1)x(P+k-1)



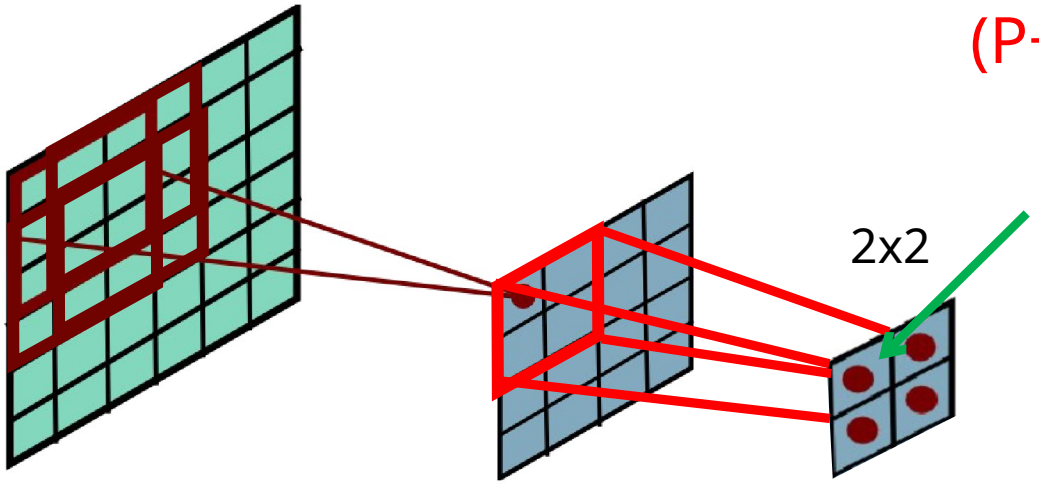
Ranzato

Pooling Layer: Receptive Field Size



If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size:

(P+k-1)x(P+k-1)



Ranzato

OK, so does pooling give us rotation invariance?

What about scale invariance?

Convolution is translation invariant (‘shift-invariant’) – we could shift the image and the kernel would give us a corresponding shift in the feature map.

But if we rotated or scaled the input, the same kernel would give a very different response.

Pooling lets us aggregate (avg) or pick from (max) responses, but the kernels themselves must be trained on and so learn to activate on scaled or rotated instances of the object.

If we max pooled over features (depth or a # kernels)...

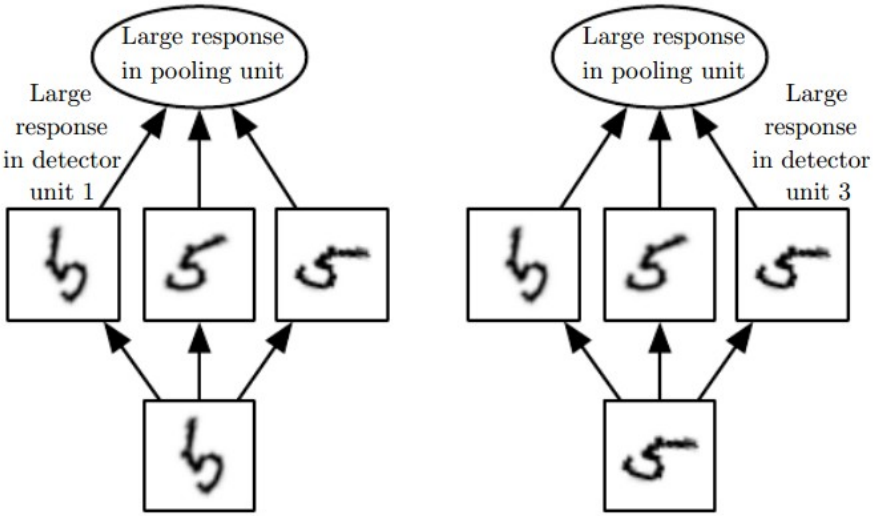
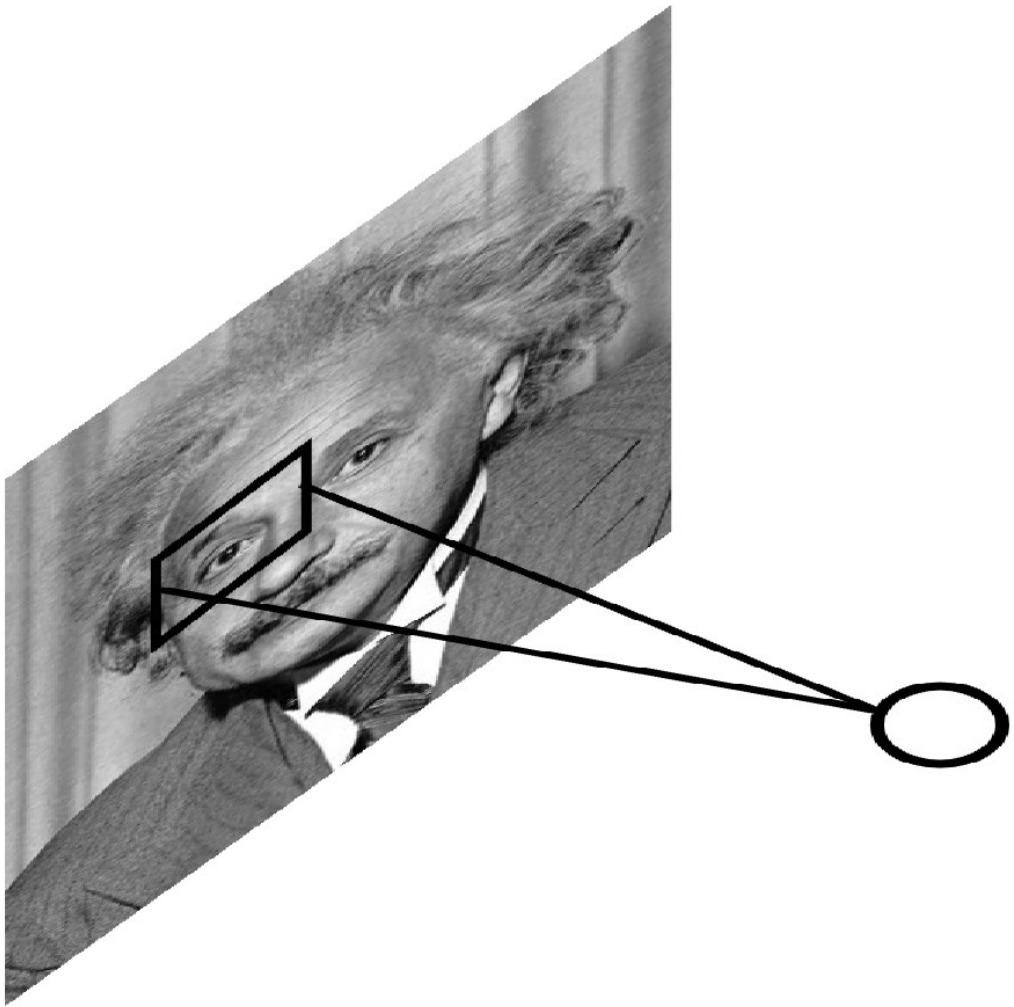


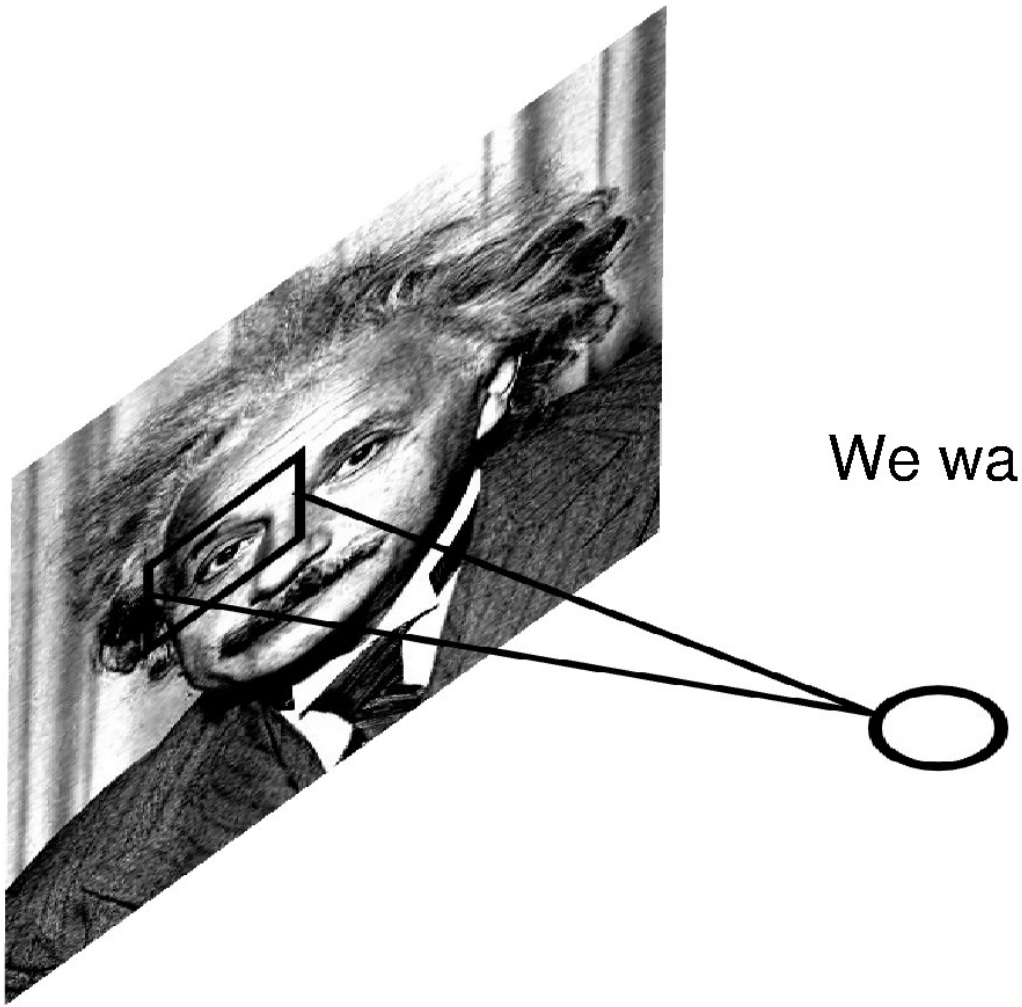
Figure 9.9: *Example of learned invariances:* A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input. Here we show how a set of three learned filters and a max pooling unit can learn to become invariant to rotation. All three filters are intended to detect a hand-written 5. Each filter attempts to match a slightly different orientation of the 5. When a 5 appears in the input, the corresponding filter will match it and cause a large activation in a detector unit. The max pooling unit then has a large activation regardless of which pooling unit was activated. We show here how the network processes two different inputs, resulting in two different detector units being activated. The effect on the pooling unit is roughly the same either way. This principle is leveraged by maxout networks (Goodfellow *et al.*, 2013a) and other convolutional networks. Max pooling over spatial positions is naturally invariant to translation; this multi-channel approach is only necessary for learning other transformations.

Fig 9.9, Goodfellow et al. [the book]

Local Contrast Normalization



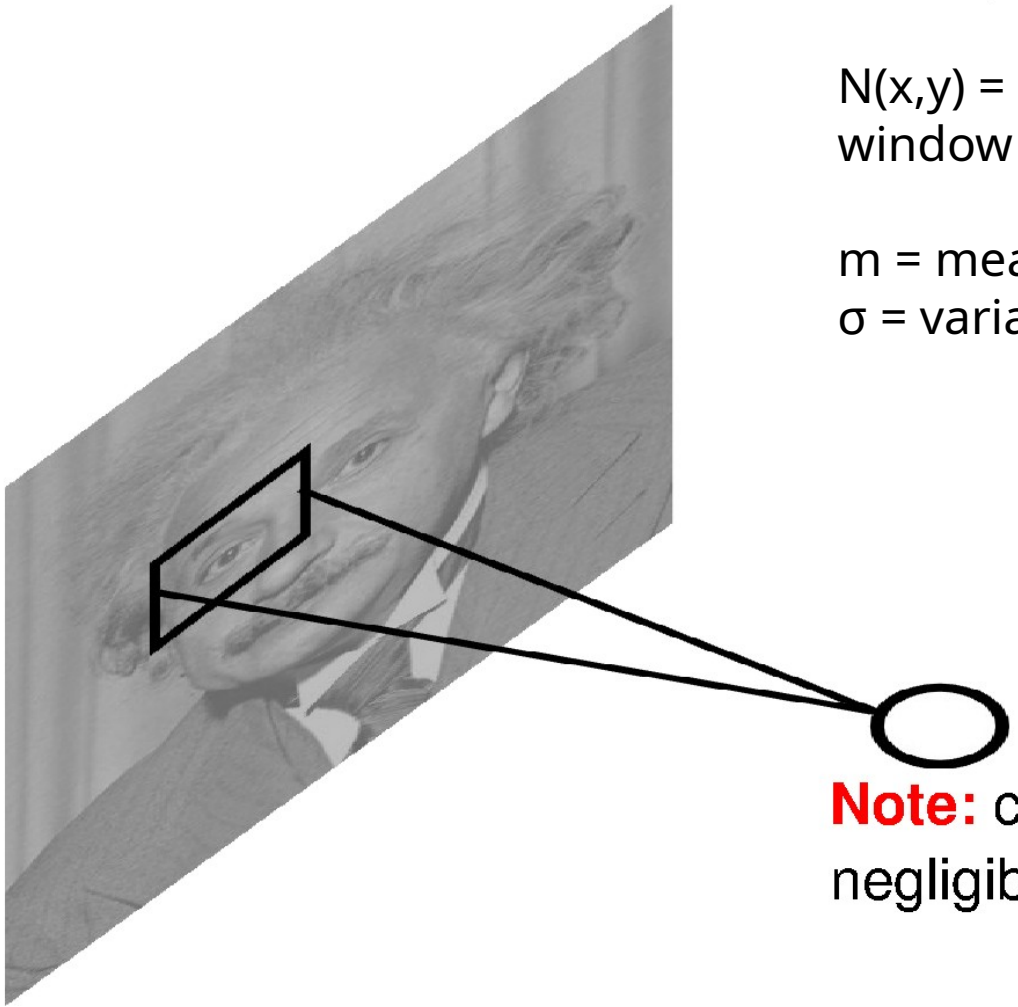
Local Contrast Normalization



We want the same response.

Local Contrast Normalization

$$h^{i+1}(x,y) = \frac{h^i(x,y) - m^i(N(x,y))}{\sigma^i(N(x,y))}$$



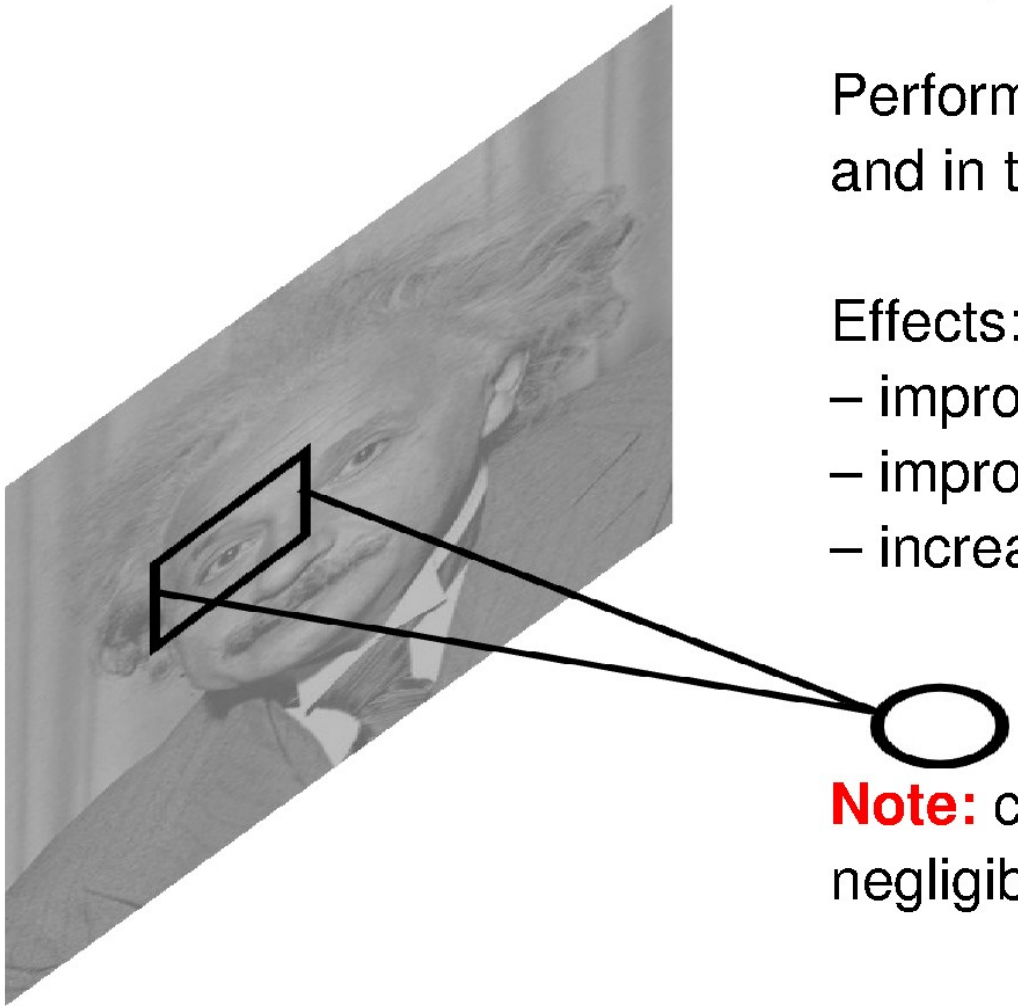
N(x,y) = model pixel values in window as a normal distribution

m = mean
σ = variance

Note: computational cost is negligible w.r.t. conv. layer.

Local Contrast Normalization

$$h^{i+1}(x,y)=\frac{h^{i}(x,y)-m^{i}(N(x,y))}{\sigma^{i}(N(x,y))}$$



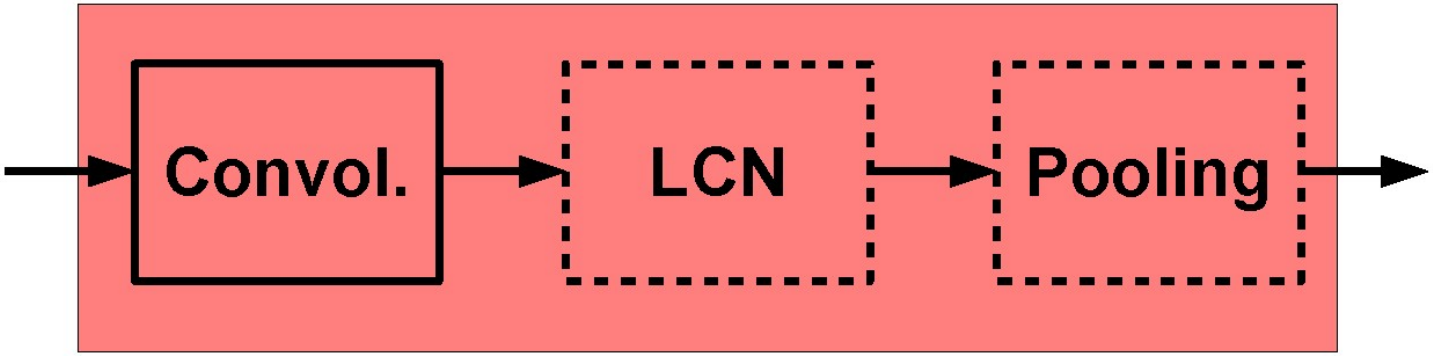
Performed also across features and in the higher layers..

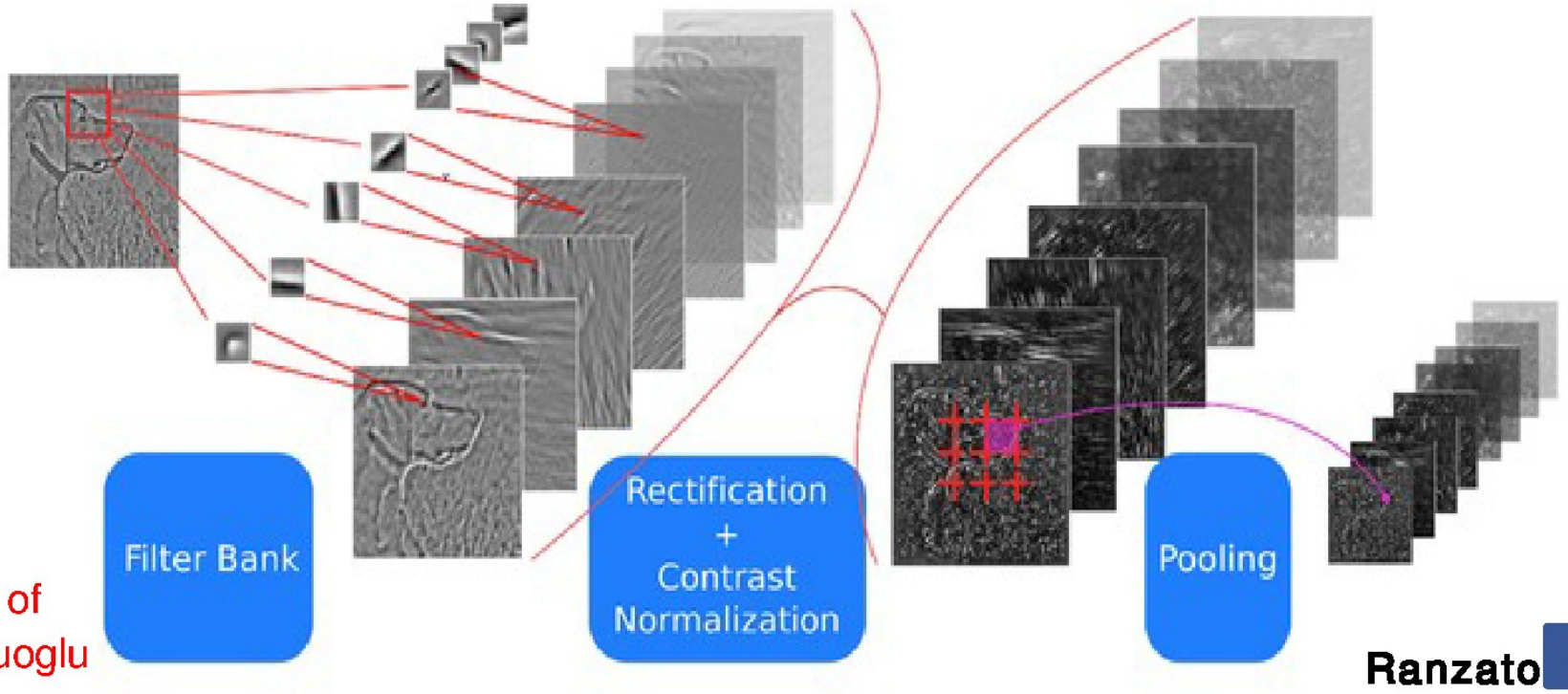
Effects:
- improves invariance
- improves optimization
- increases sparsity

Note: computational cost is negligible w.r.t. conv. layer.

ConvNets: Typical Stage

One stage (zoom)

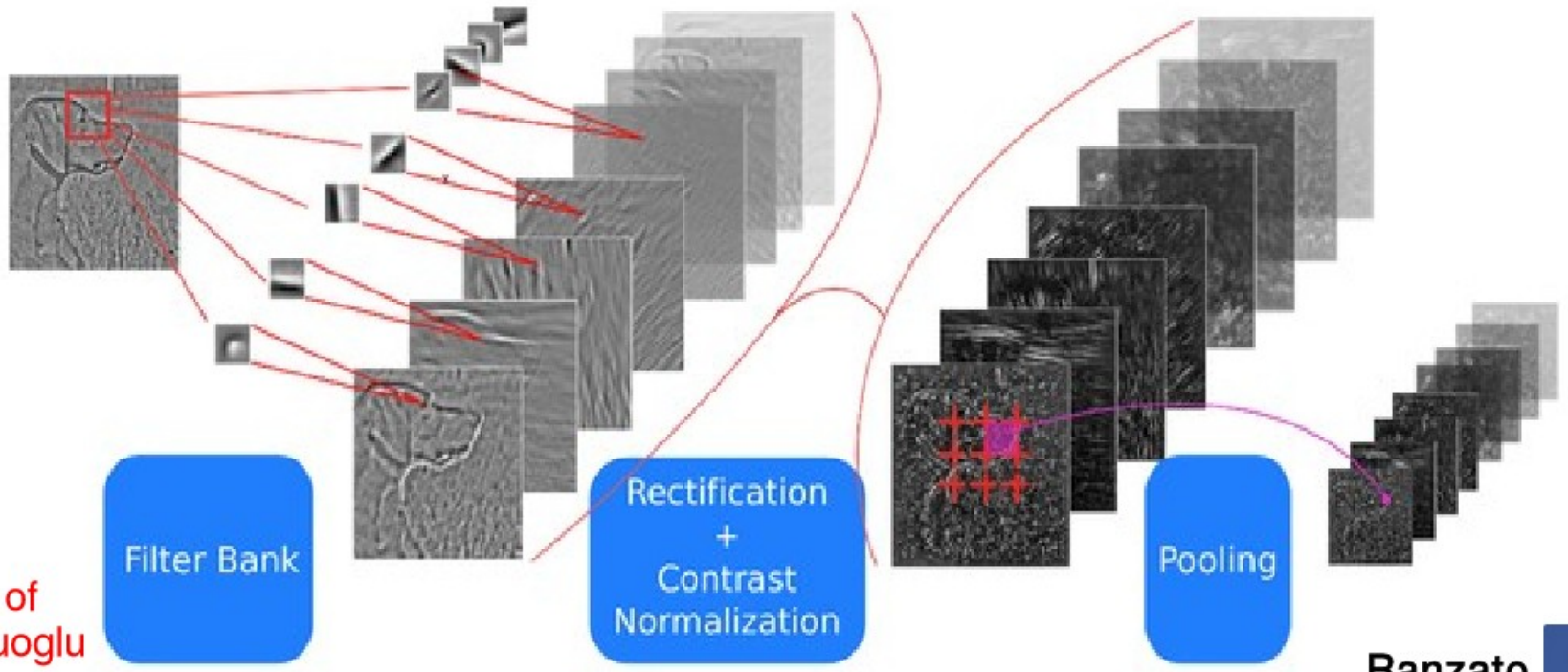




courtesy of
K. Kavukcuoglu

Note: after one stage the number of feature maps is usually increased (conv. layer) and the spatial resolution is usually decreased (stride in conv. and pooling layers). Receptive field gets bigger.

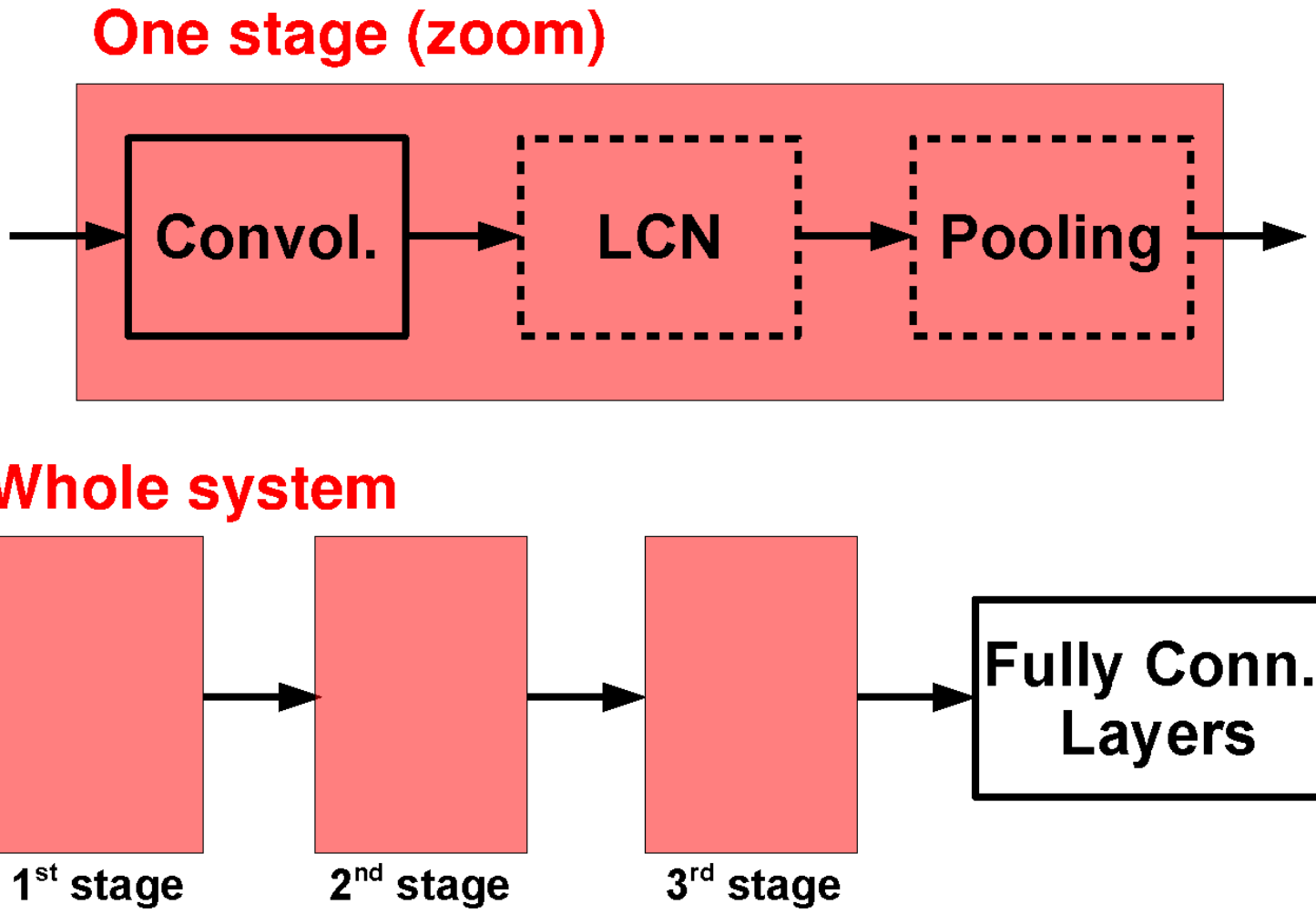
Reasons:
- gain invariance to spatial translation (pooling layer)
- increase specificity of features (approaching object specific units)



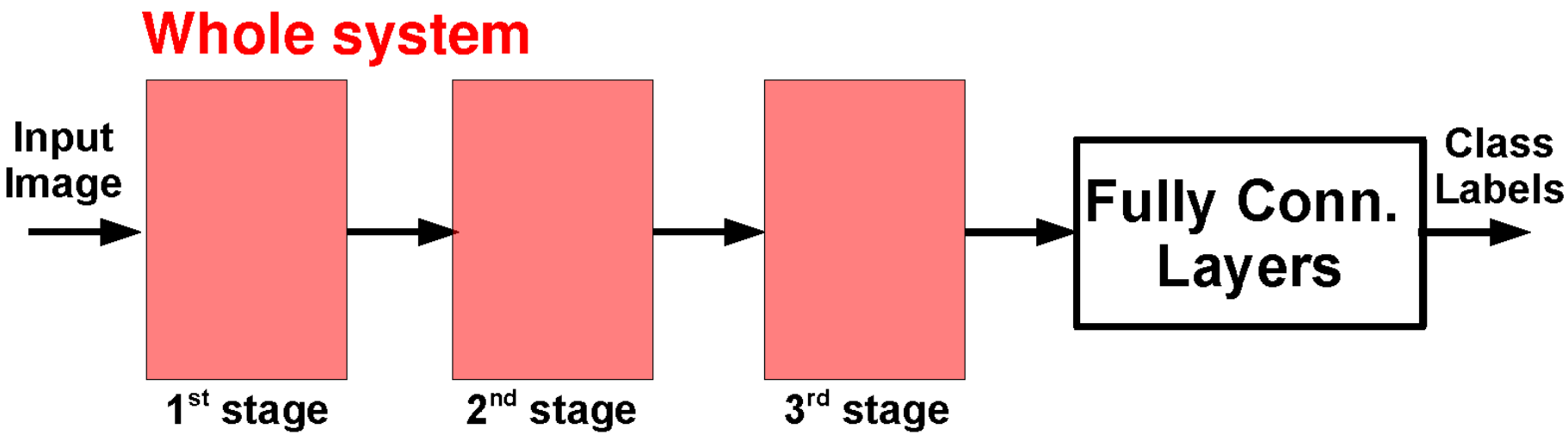
courtesy of
K. Kavukcuoglu

Ranzato

ConvNets: Typical Architecture



ConvNets: Typical Architecture



Conceptually similar to:

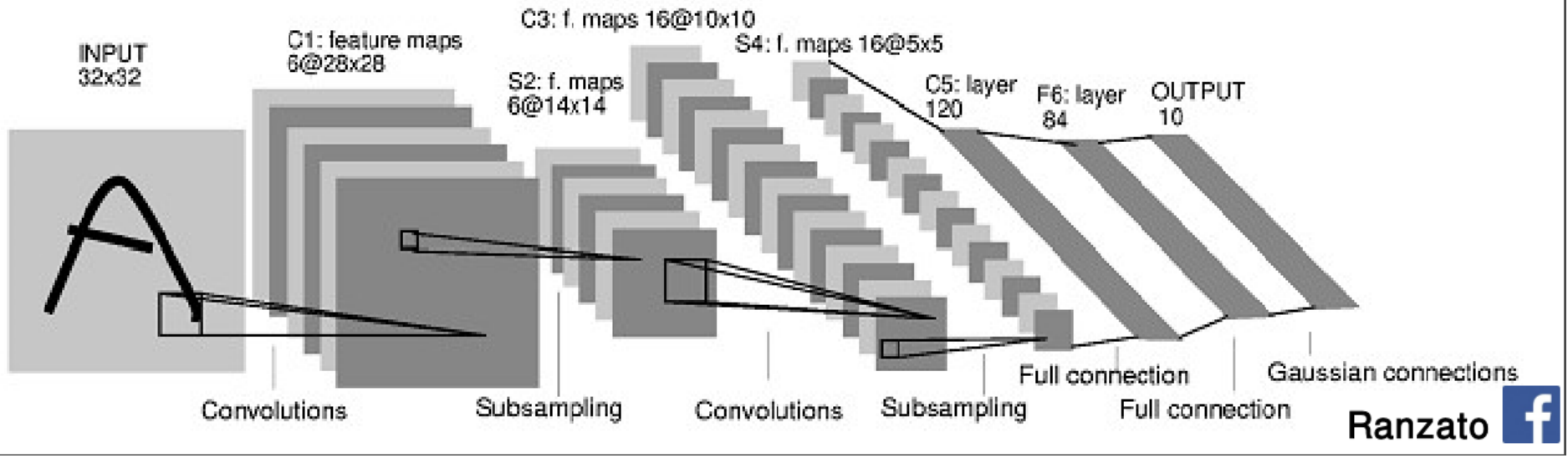
SIFT → K-Means → Pyramid Pooling → SVM

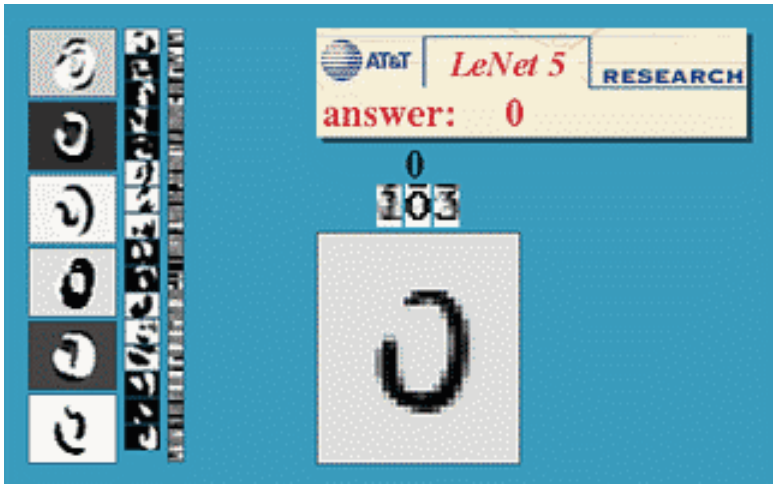
Lazebnik et al. "...Spatial Pyramid Matching..." CVPR 2006

SIFT → Fisher Vect. → Pooling → SVM

Sanchez et al. "Image classifcation with F.V.: Theory and practice" IJCV 2012

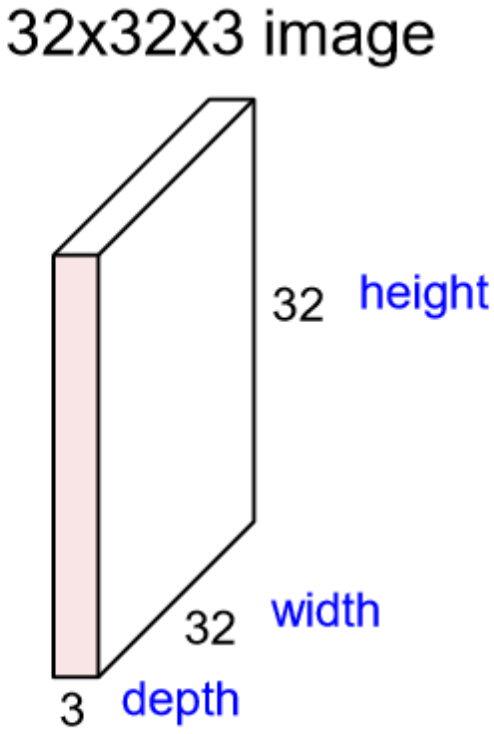
Yann LeCun's MNIST CNN architecture (1998)





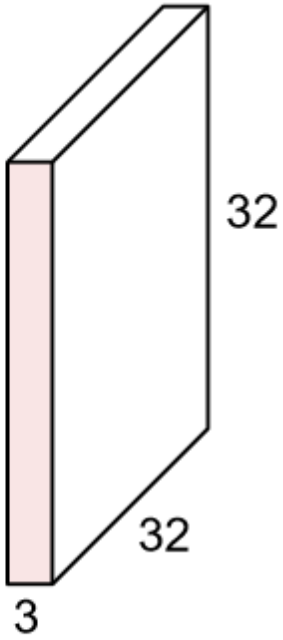
http://scs.ryerson.ca/~aharley/vis/conv/

Convolutions: More detail



Convolutions: More detail

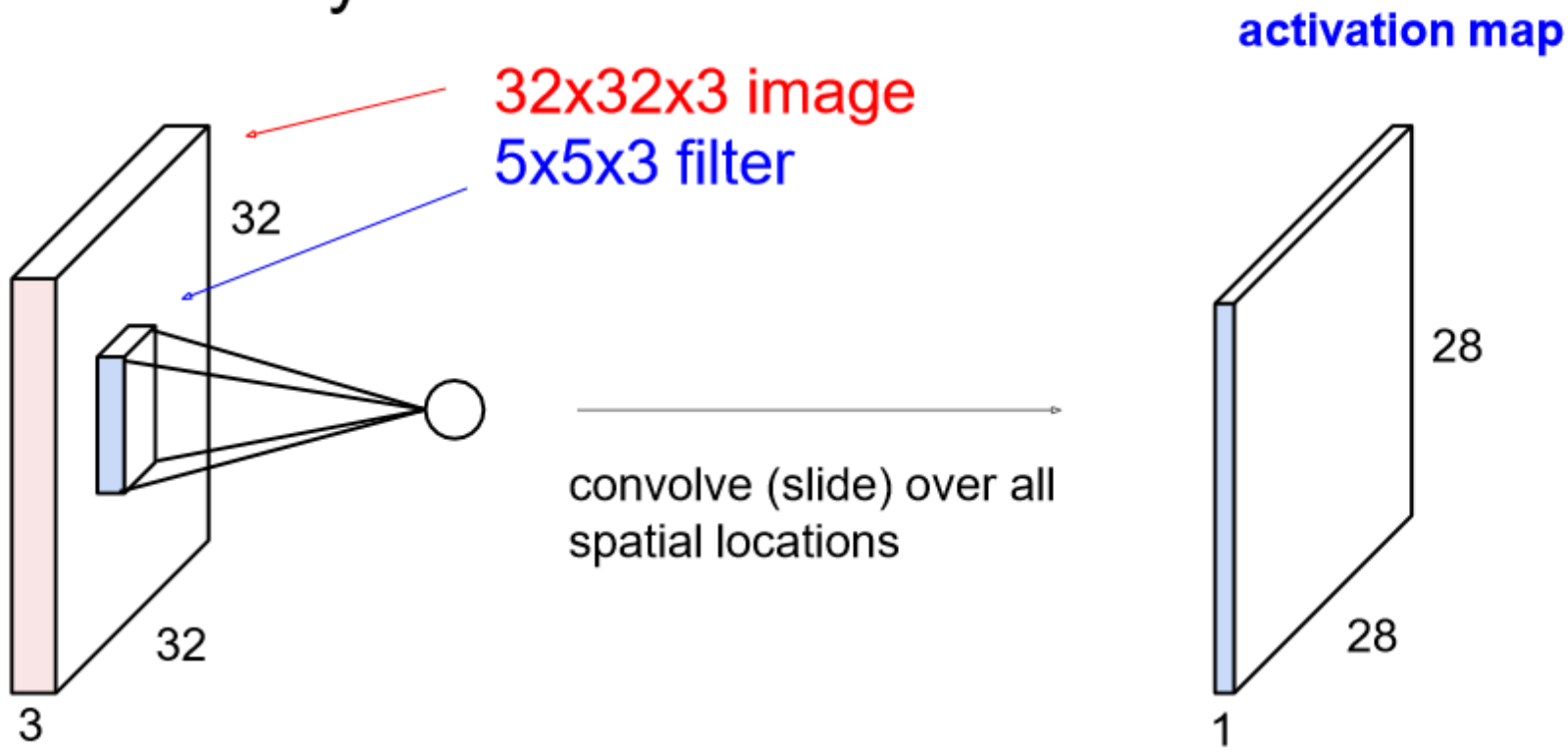
32x32x3 image



5x5x3 filter

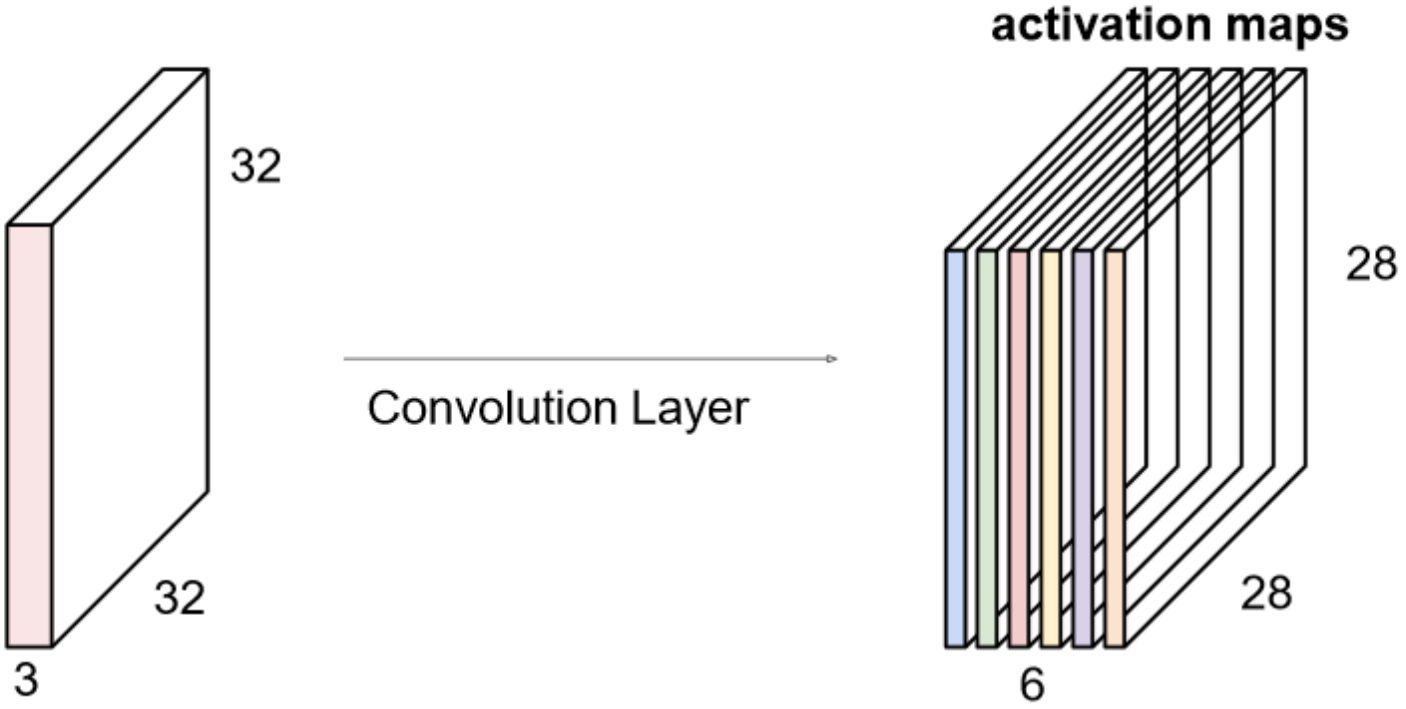
Convolutions: More detail

Convolution Layer



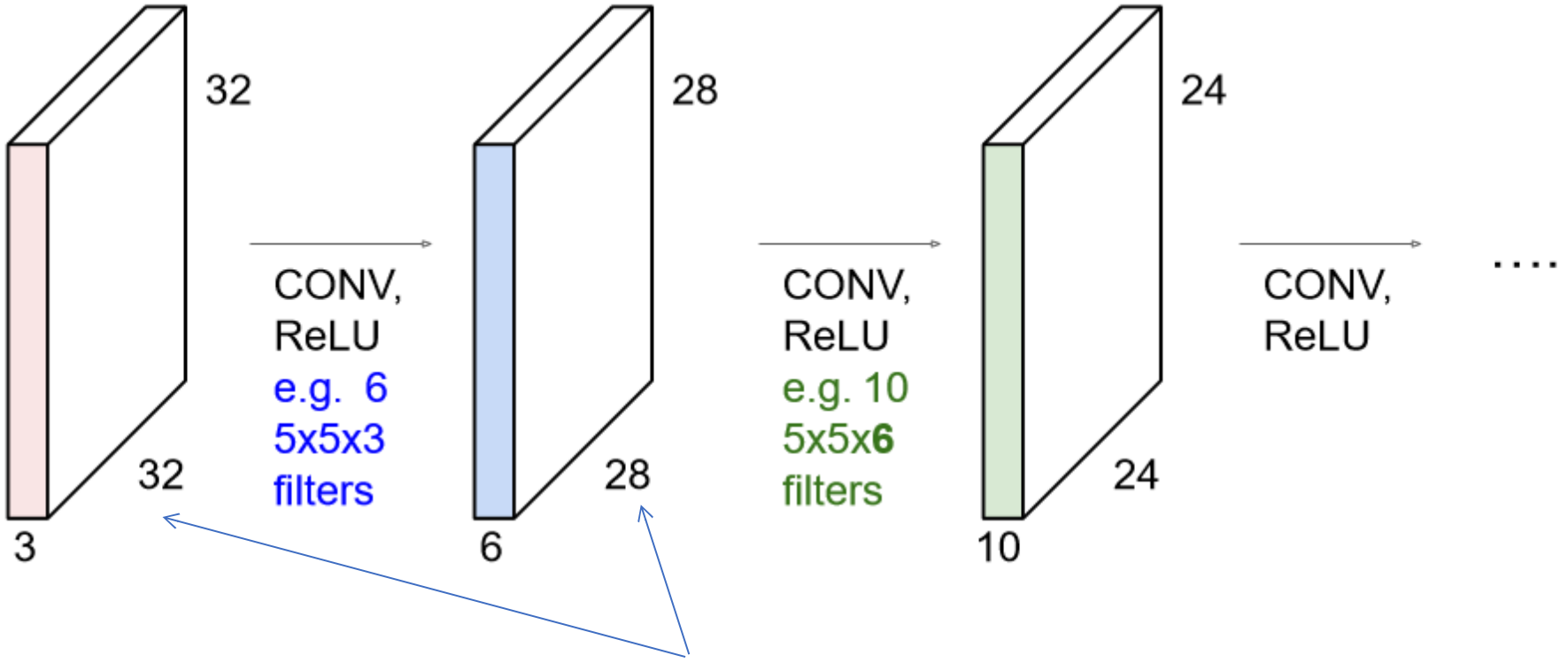
Convolutions: More detail

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



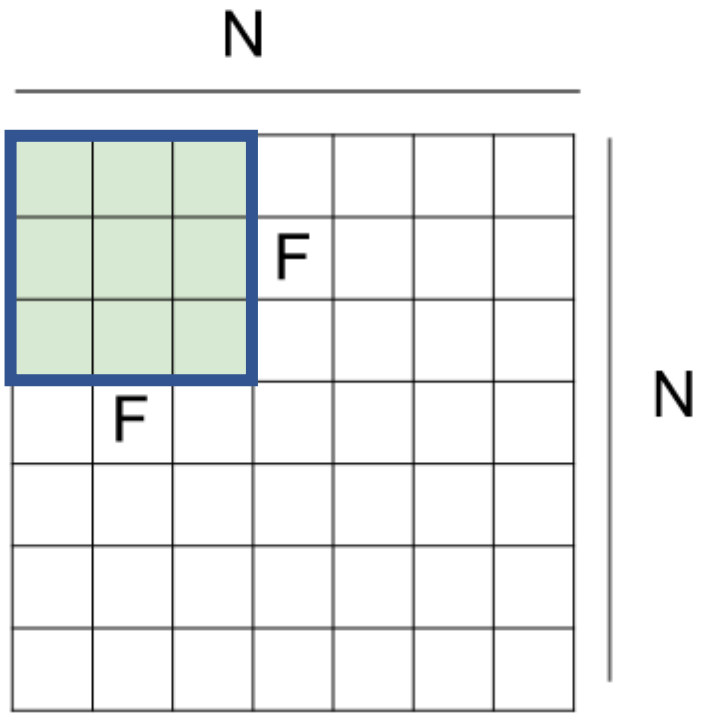
We stack these up to get a "new image" of size 28x28x6!

Convolutions: More detail



The size decreases in the convolution process if we don't pad

Convolutions: More detail



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

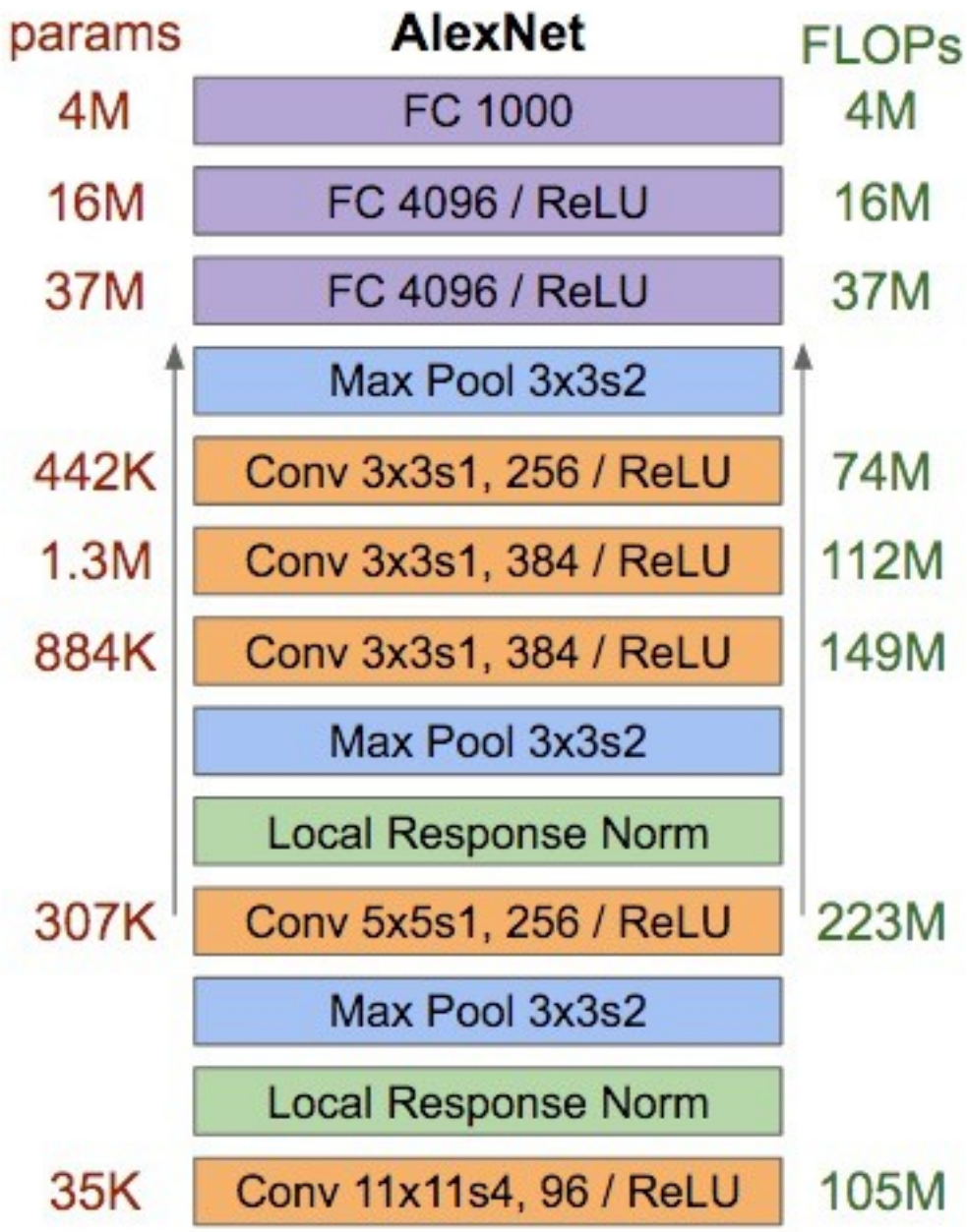
Example: N = 7, F = 3, no padding

if stride = 1, output size = (7-3)/1 + 1 = 5
if stride = 2, output size = (7-3)/2 +1 = 3
if stride = 3, output size = (7-3)/3 +1 = 2.33 :\

Reading architecture diagrams

Layers
- Kernel sizes
- Strides
- # channels
- # kernels
- Max pooling

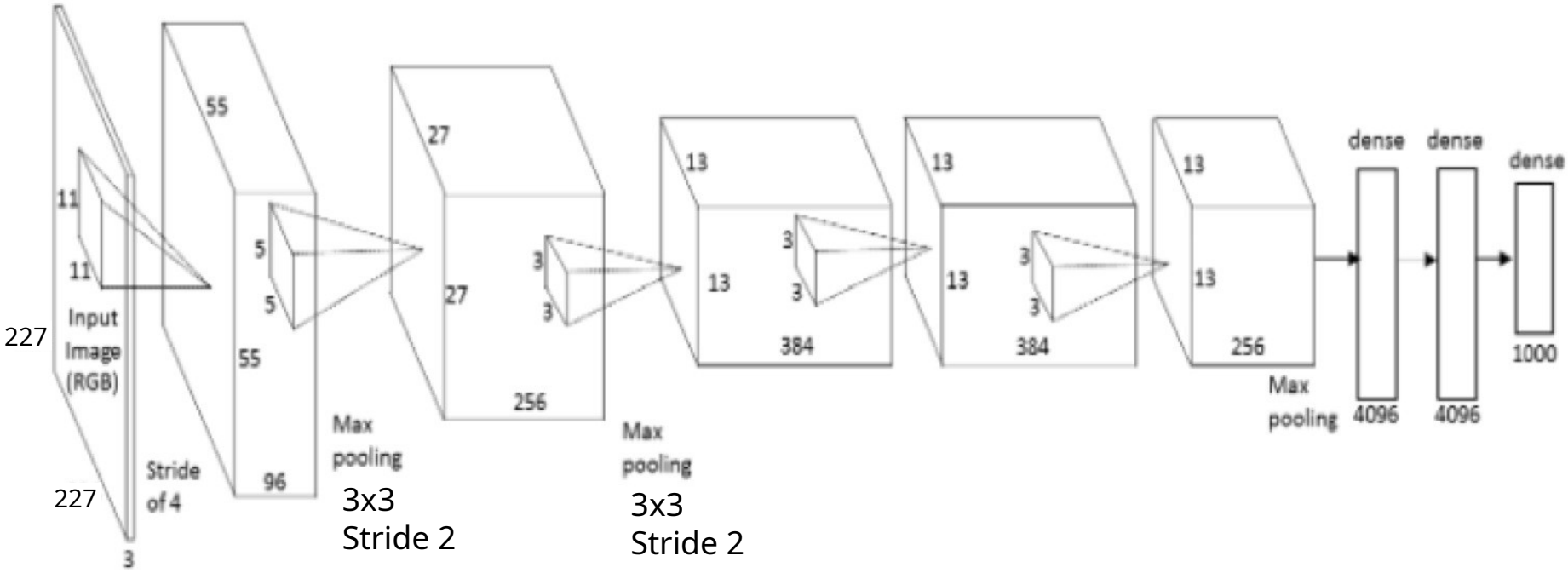
winner of ImageNet competition in 2012



AlexNet diagram (simplified)

[Krizhevsky et al. 2012]

Input size
227 x 227 x 3



Conv 1
11 x 11 x 3
Stride 4
96 filters

Conv 2
5 x 5 x 96
Stride 1
256 filters

Conv 3
3 x 3 x 256
Stride 1
384 filters

Conv 4
3 x 3 x 192
Stride 1
384 filters

Conv 4
3 x 3 x 192
Stride 1
256 filters

Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples

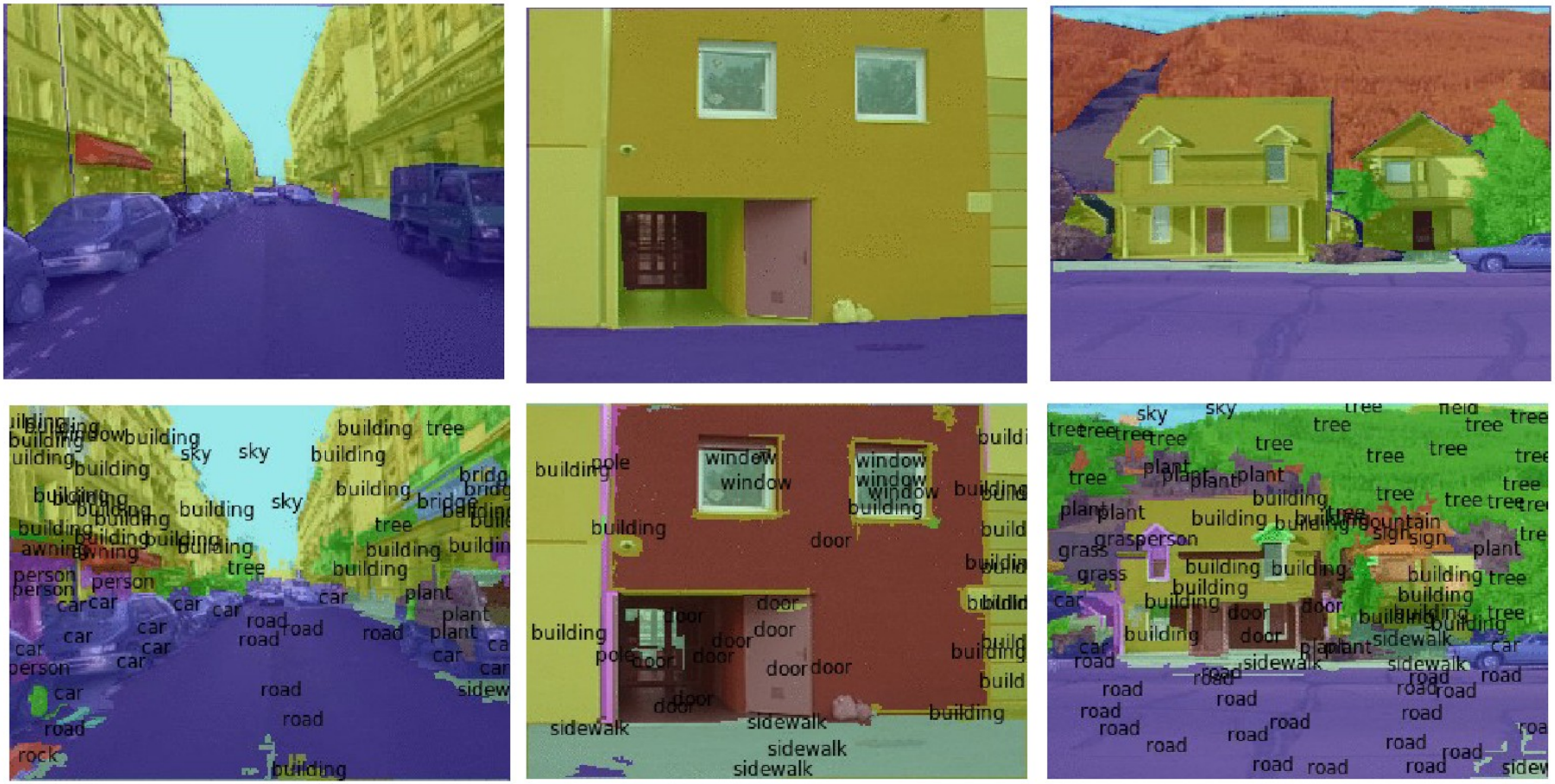
CONV NETS: EXAMPLES

- **OCR / House number & Traffic sign classification**

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

CONV NETS: EXAMPLES

- **Scene Parsing**

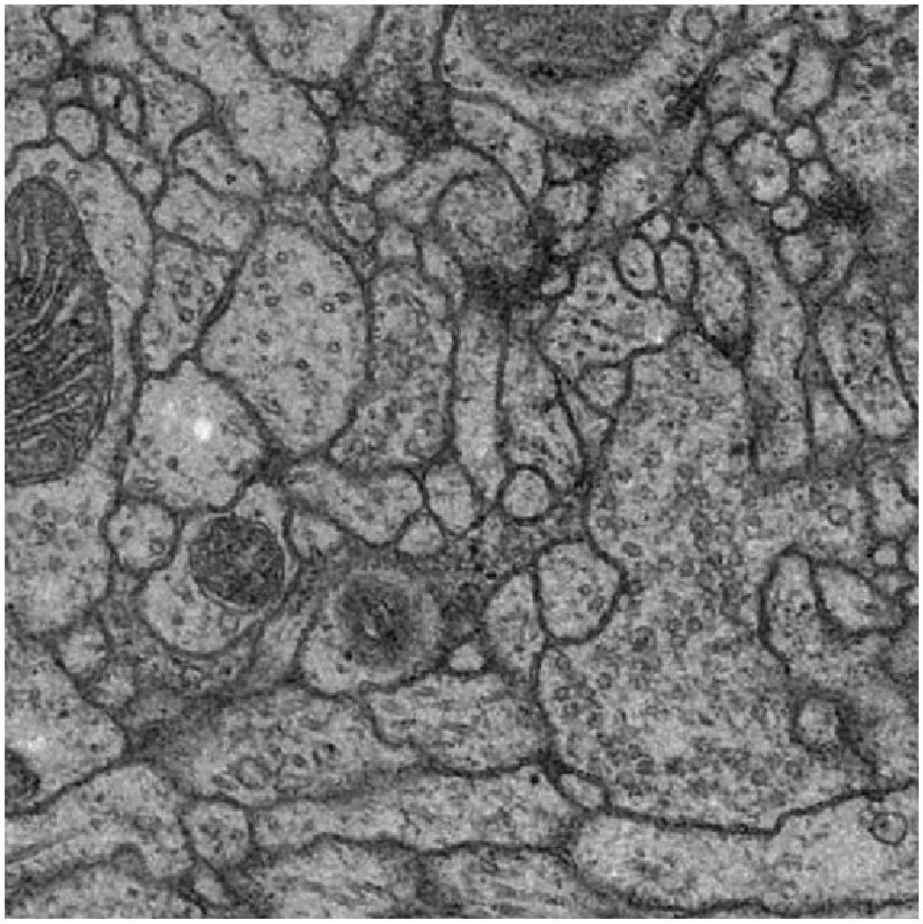


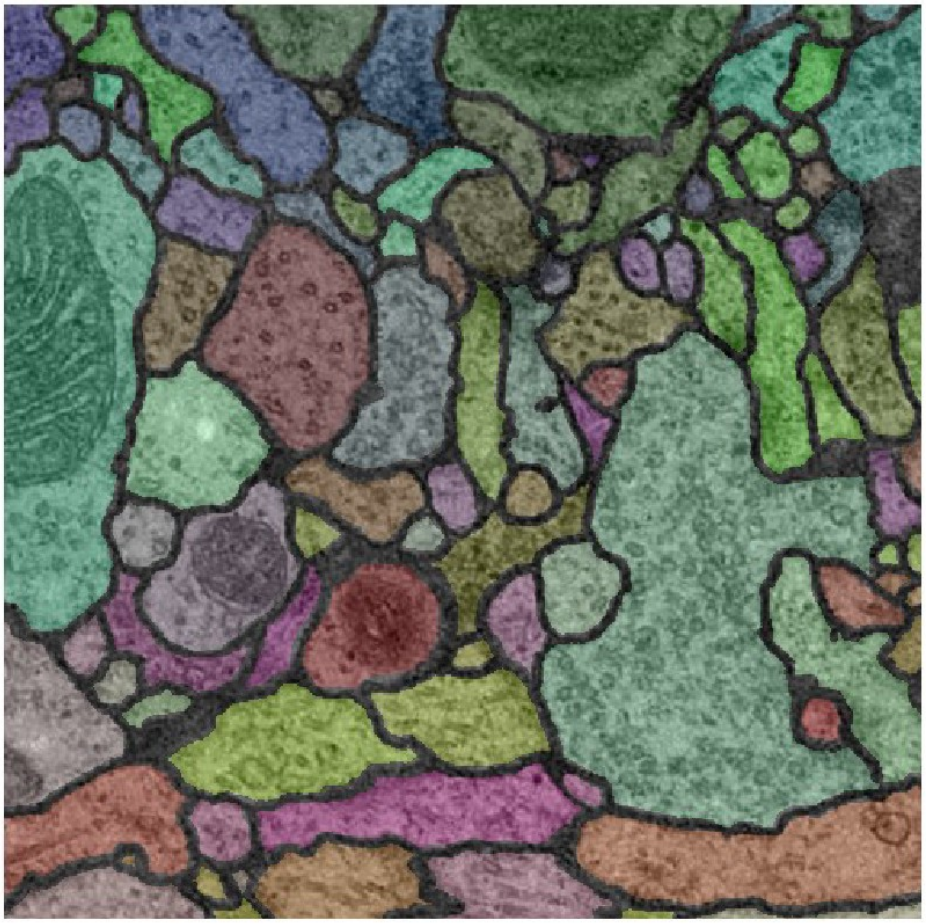
Farabet et al. “Learning hierarchical features for scene labeling” PAMI 2013

Pinheiro et al. “Recurrent CNN for scene parsing” arxiv 2013

CONV NETS: EXAMPLES

- **Segmentation 3D volumetric images**

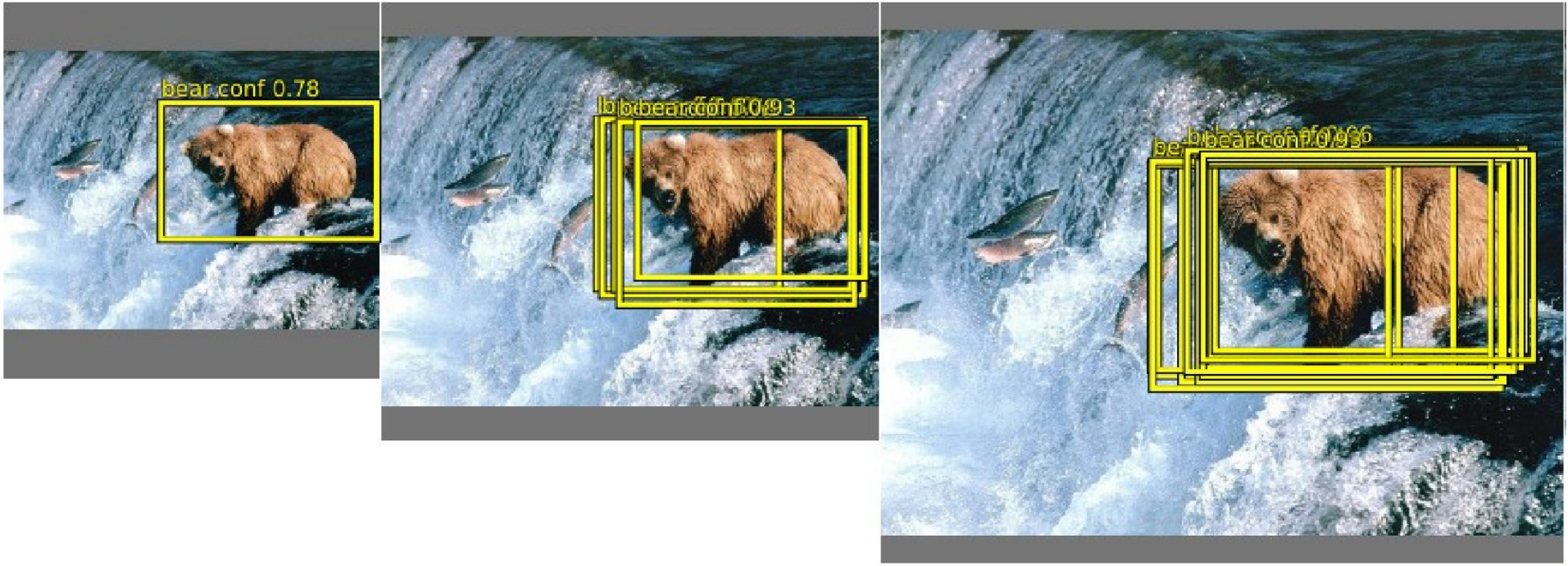




Ciresan et al. “DNN segment neuronal membranes...” NIPS 2012
Turaga et al. “Maximin learning of image segmentation” NIPS 2009

CONV NETS: EXAMPLES

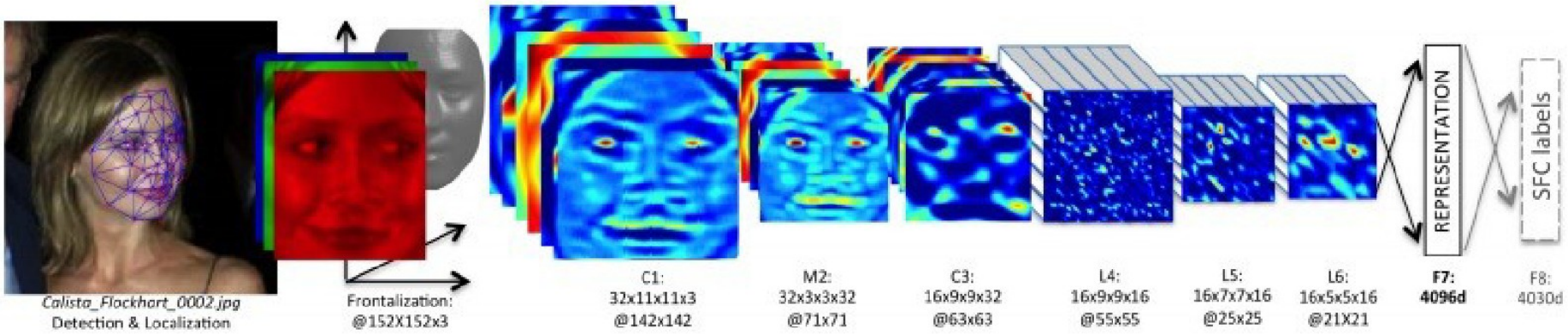
- **Object detection**



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

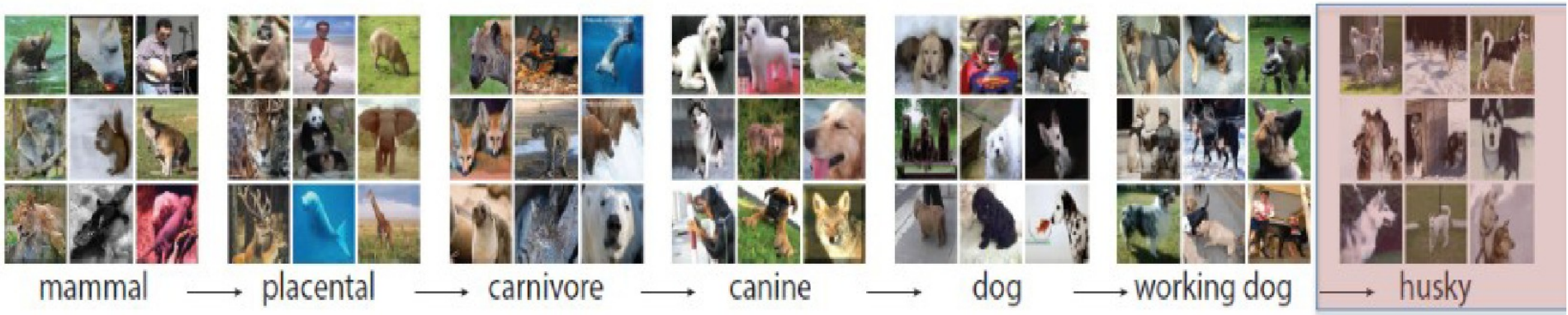
CONV NETS: EXAMPLES

- **Face Verification & Identification**



Taigman et al. "DeepFace..." CVPR 2014

Dataset: ImageNet 2012



- S: (n) Eskimo dog, husky (breed of heavy-coated Arctic sled dog)
  - *direct hypernym / inherited hypernym / sister term*
    - S: (n) working dog (any of several breeds of usually large powerful dogs bred to work as draft animals and guard and guide dogs)
      - S: (n) dog, domestic dog, Canis familiaris (a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds) *"the dog barked all night"*
        - S: (n) canine, canid (any of various fissiped mammals with nonretractile claws and typically long muzzles)
          - S: (n) carnivore (a terrestrial or aquatic flesh-eating mammal) *"terrestrial carnivores have four or five clawed digits on each limb"*
            - S: (n) placental, placental mammal, eutherian, eutherian mammal (mammals having a placenta; all mammals except monotremes and marsupials)
              - S: (n) mammal, mammalian (any warm-blooded vertebrate having the skin more or less covered with hair; young are born alive except for the small subclass of monotremes and nourished with milk)
                - S: (n) vertebrate, craniate (animals having a bony or cartilaginous skeleton with a segmented spinal column and a large brain enclosed in a skull or cranium)
                  - S: (n) chordate (any animal of the phylum Chordata having a notochord or spinal column)
                    - S: (n) animal, animate being, beast, brute, creature, fauna (a living organism characterized by voluntary movement)
                      - S: (n) organism, being (a living thing that has (or can develop) the ability to act or function independently)
                        - S: (n) living thing, animate thing (a living (or once living) entity)
                          - S: (n) whole, unit (an assemblage of parts that is regarded as a single entity) *"how big is that part compared to the whole?"; "the team is a unit"*
                            - S: (n) object, physical object (a tangible and visible entity; an entity that can cast a shadow) *"it was full of rackets, balls and other objects"*
                              - S: (n) physical entity (an entity that has physical existence)
                                - S: (n) entity (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))

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

mite

mite	
black widow	
cockroach	
tick	
starfish	

container ship

container ship	
lifeboat	
amphibian	
fireboat	
drilling platform	

motor scooter

motor scooter	
go-kart	
moped	
bumper car	
golfcart	

leopard

leopard	
jaguar	
cheetah	
snow leopard	
Egyptian cat	

grille

convertible	
grille	
pickup	
beach wagon	
fire engine	

mushroom

agaric	
mushroom	
jelly fungus	
gill fungus	
dead-man's-fingers	

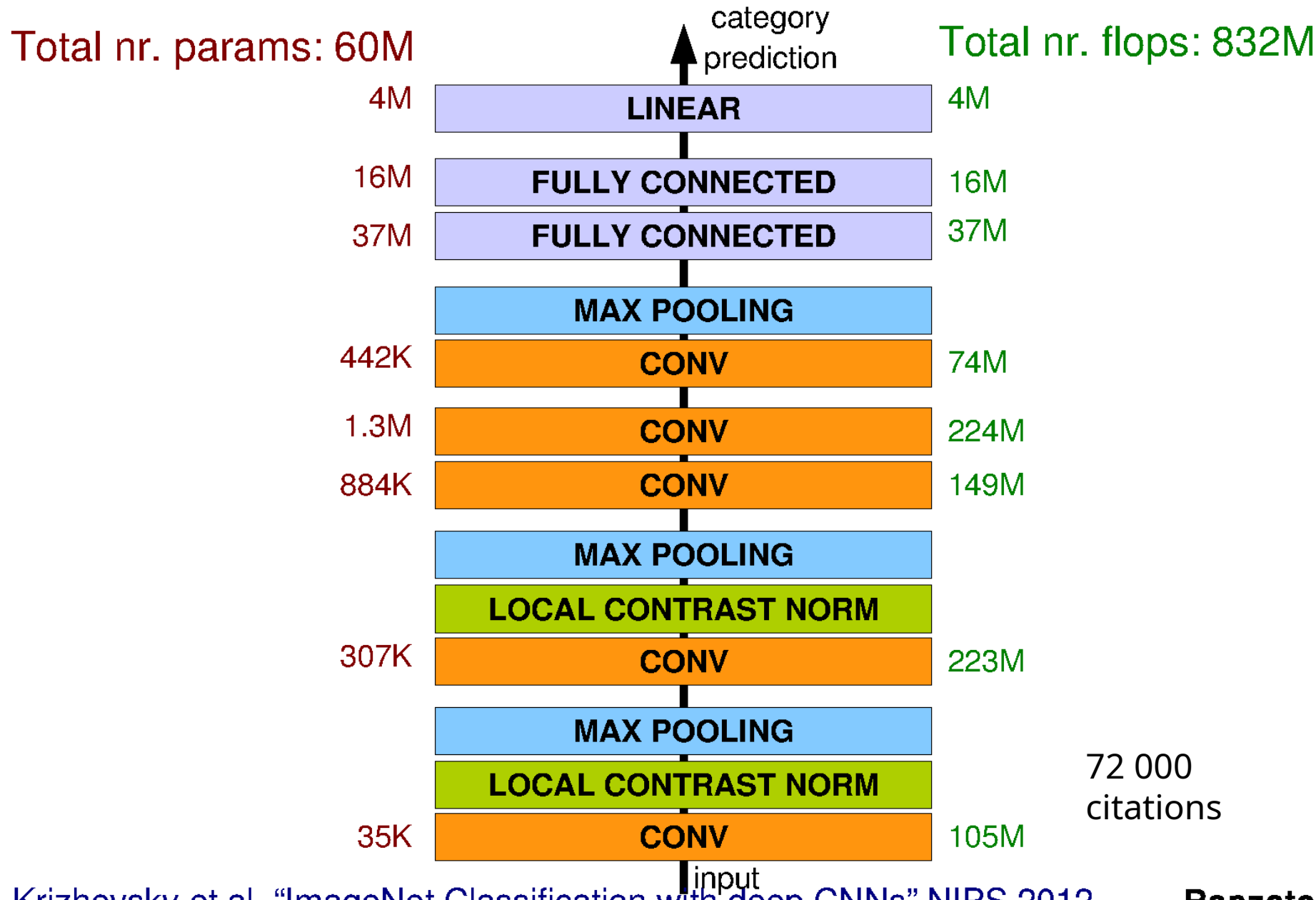
cherry

dalmatian	
grape	
elderberry	
ffordshire bullterrier	
currant	

Madagascar cat

squirrel monkey	
spider monkey	
titi	
indri	
howler monkey	

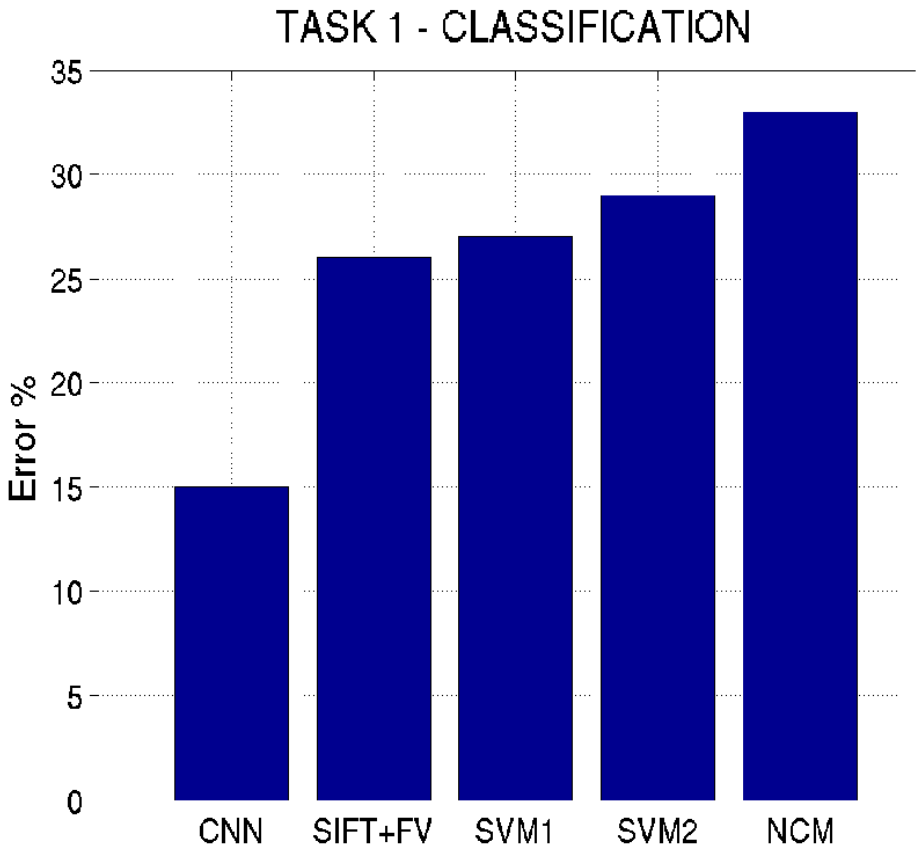
Architecture for Classification

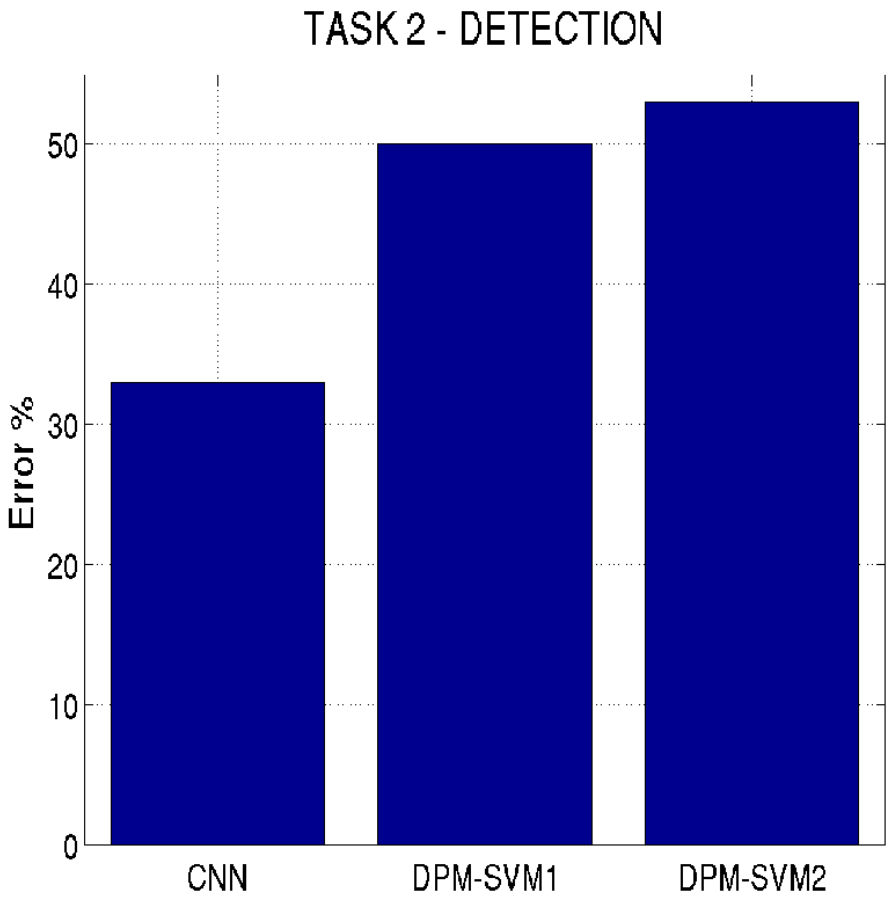


72 000 citations

Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

Results: ILSVRC 2012





Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

QUICK TEST!!

Q1

Input size: 96 x 96 x 3
Kernel size: 5 x 5 x 3
Stride: 1
Max pooling layer: 4 x 4

Output feature map size?
a) 5 x 5
b) 22 x 22
c) 23 x 23
d) 24 x 24
e) 25 x 25

Q2

Input size: 96 x 96 x 3
Kernel size: 3 x 3 x 3
Stride: 3
Max pooling layer: 8 x 8

Output feature map size?
a) 2 x 2
b) 3 x 3
c) 4 x 4
d) 5 x 5
e) 12 x 12

QUICK TEST!!

Q1

Input size: 96 x 96 x 3

Kernel size: 5 x 5 x 3

Stride: 1

Max pooling layer: 4 x 4

Output feature map size?

a) 5 x 5

b) 22 x 22

c) 23 x 23

d) 24 x 24

e) 25 x 25

the filter size is 5x5, when performing the convolution we will lose 2 pixels from each boundary -> 96 -2 -2 = 92
since the stride is 1, we will go over every single pixel of the 92x92 pixels
therefore the final convolution output is 92x92

The official equation for this is:
input_size - filter_size + stride)/stride

the max pooling layer is 4x4 -> 92/4 = 23

Final answer: 23x23

QUICK TEST!!

Q2

Input size: 96 x 96 x 3

Kernel size: 3 x 3 x 3

Stride: 3

Max pooling layer: 8 x 8

Output feature map size?

a) 2 x 2

b) 3 x 3

c) 4 x 4

d) 5 x 5

e) 12 x 12

the stride = 3 and filter size =3, so we will apply the convolution on every 3 other pixels.
This process will have
(input_size - filter_size + stride)/stride =
(96 - 3 + 3)/ 3 = 32

The ouput of conv + stride = 32x32

the max pooling layer is 8x8 -> 32/8= 4

Final answer: 4x4

More ConvNet explanations

- https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/