MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CENG 403
Introduction to Deep Learning

Week 9a

Sinan Kalkan

Previously on CENG403

Weight Initialization

- Zero weights
  - Wrong!
  - Leads to updating weights by the same amounts for every input
- Initialize the weights randomly to small values:
  - Sample from a small range, e.g., Normal(0, 0.01)
  - Don’t initialize too small
- The bias may be initialized to zero
  - For ReLU units, this may be a small number like 0.01.

Note: None of these provide guarantees of convergence.
Moreover, there is no guarantee that one of these will always be better.

Weight Initialization

Previously on CENG403

Solution:

Get rid of n in $\text{Var}(s) = (n\ \text{Var}(\text{w}))\text{Var}(\text{x})$

- How?
  - Scale the initial weights by $\sqrt{n}$
  - Why? Because: $\text{Var}(aX) = a^2\text{Var}(X)$

- Standard Initialization (top plots in Figure 6 & 7):

$$w_i \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$$

which yields $n\ \text{Var}(w) = \frac{1}{3}$

because variance of $U[-r, r]$ is $\frac{r^2}{3}$ (see [1])

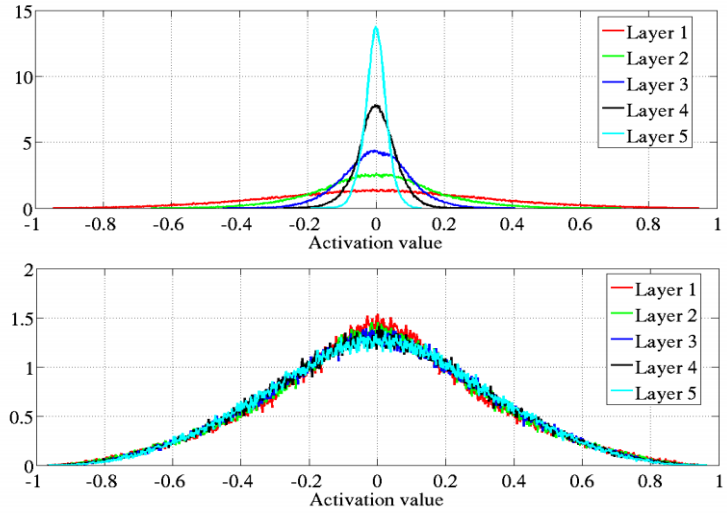


Figure 6: *Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.*

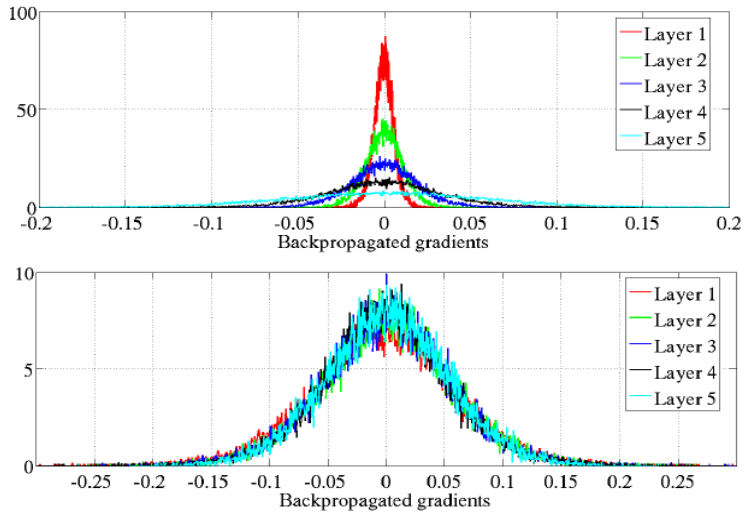


Figure 7: *Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.*

Figures: Glorot & Bengio, "Understanding the difficulty of training deep feedforward neural networks", 2010.

Xavier initialization for symmetric activation functions (Glorot & Bengio):

$$w_i \sim N\left(0, \frac{\sqrt{2}}{\sqrt{n_{in} + n_{out}}}\right)$$

With Uniform distribution:

$$w_i \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\right]$$

[1] https://proofwiki.org/wiki/Variance_of_Continuous_Uniform_Distribution

Previously on ENG403

Alternative: Batch Normalization

- Normalization is differentiable
  - So, make it part of the model (not only at the beginning)
  - I.e., perform normalization during every step of processing
- More robust to initialization
- Shown to also regularize the network in some cases (dropping the need for dropout)
- Issue: How to normalize at test time?
  1. Store means and variances during training, or
  2. Calculate mean & variance over your test data
- PyTorch: use model.eval() in test time.

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

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

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

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

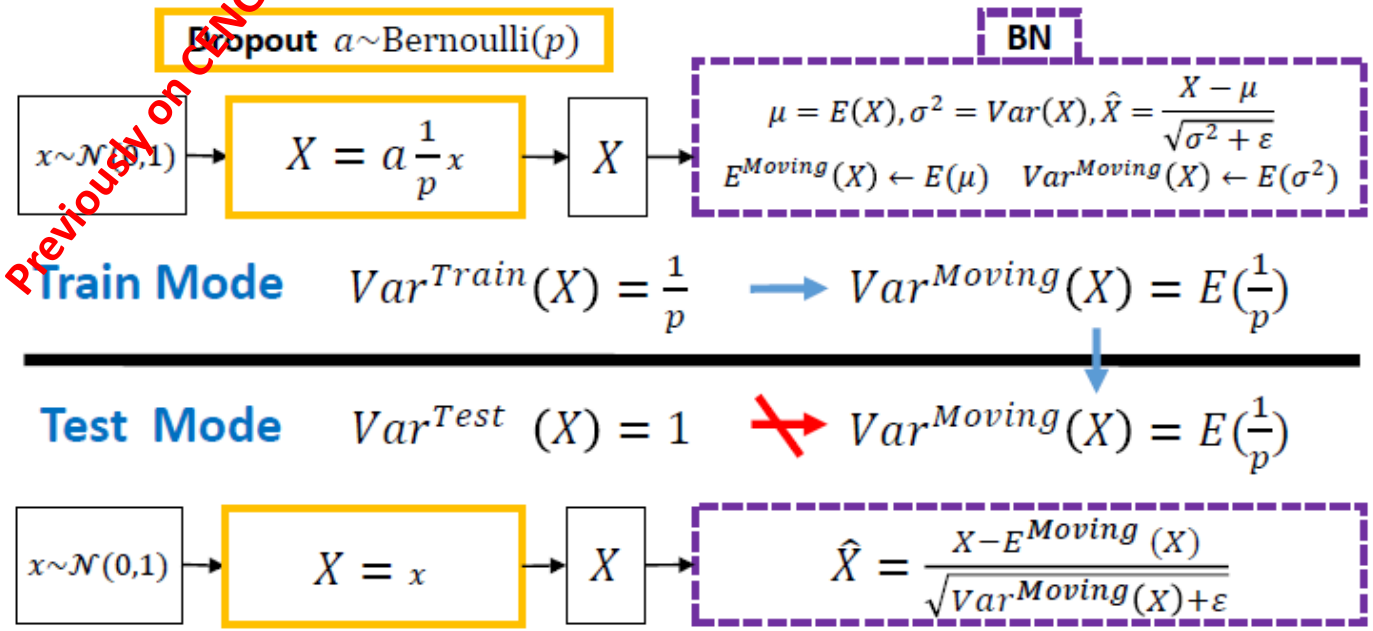
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

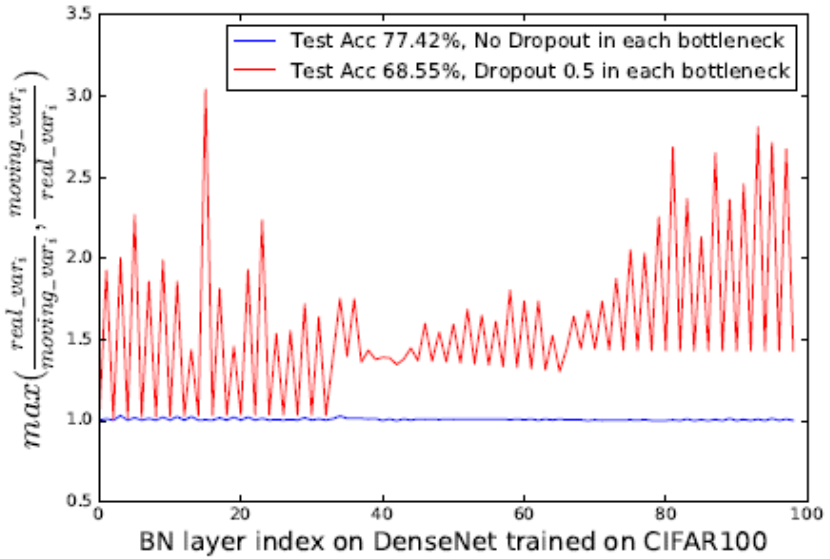
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Ioffe & Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", 2015.

Previously on CENG403





Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift

Xiang Li [1] Shuo Chen [1] Xiaolin Hu [2] Jian Yang [1]

2018

Since we get a clear knowledge about the disharmony between Dropout and BN, we can easily develop several approaches to combine them together, to see whether an extra improvement could be obtained. In this section, we introduce two possible solutions in modifying Dropout. One is to avoid the scaling on feature-map before every BN layer, by only applying Dropout after the last BN block. Another is to slightly modify the formula of Dropout and make it less sensitive to variance, which can alleviate the shift problem and stabilize the numerical behaviors.

Today

- Convolutional Neural Networks (CNNs)
  - Motivation for CNNs
  - MLPs vs CNNs
  - Operations in CNNs
  - Types of convolution in CNNs

Convolutional Neural Networks: MOTIVATION

Disadvantages of MLPs: Dimensionality

- The number of parameters in an MLP is high for practical problems
  - e.g., for grayscale images with 1000x1000 resolution, a fully-connected layer with 1000 neurons requires 10^9 parameters.
- The number of parameters in an MLP increases quadratically with an increase in input dimensionality
- For example, for a fully-connected layer with n_{in} input neurons and n_{out} output neurons:
  - Number of parameters: $n_{in} \times n_{out}$
  - Assuming proportional decrease in layer size, e.g. $n_{out} = n_{in}/10$, gives: $n_{in} \times n_{out} = n_{in}^2/10$
  - Increasing n_{in} by d yields a change of $\mathcal{O}(d^2)$.
- This is a problem because:
  - More parameters => larger model size & more computational complexity.
  - More parameters => more data for good generalization.
- Teaser for CNNs:
  - Input size does not affect model size (in general)

Disadvantages of MLPs: Curse of Dimensionality

- For conventional ML methods:
  - The number of required samples for obtaining small error increases exponentially with input dimensions

- For deep networks:
  - This does not seem to be an issue for deep networks (see e.g. Poggio & Liao, 2018).

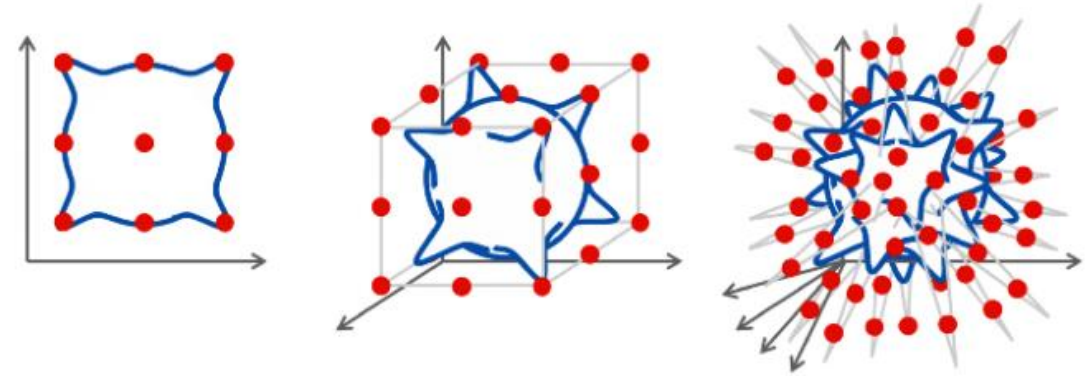


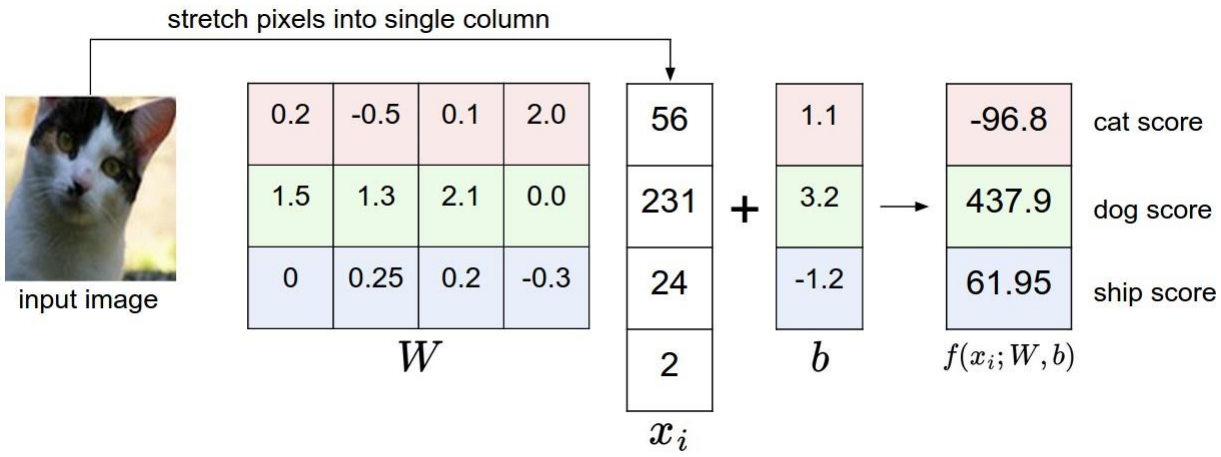
Illustration of the curse of dimensionality: in order to approximate a Lipschitz-continuous function composed of Gaussian kernels placed in the quadrants of a d-dimensional unit hypercube (blue) with error ε, one requires $\mathcal{O}(1/\varepsilon^d)$ samples (red points).

Figure: https://towardsdatascience.com/geometric-foundations-of-deep-learning-94cdd45b451d

Poggio, T., & Liao, Q. (2018). Theory I: Deep networks and the curse of dimensionality. Bulletin of the Polish Academy of Sciences: Technical Sciences, (6).

Disadvantages of MLPs: Equivariance & Invariance

- Vectorizing an image breaks patterns in consecutive pixels.
  - Shifting one pixel means a whole new vector
  - Makes learning more difficult
  - Requires more data to generalize



Shifted Image
(e.g. by one pixel)

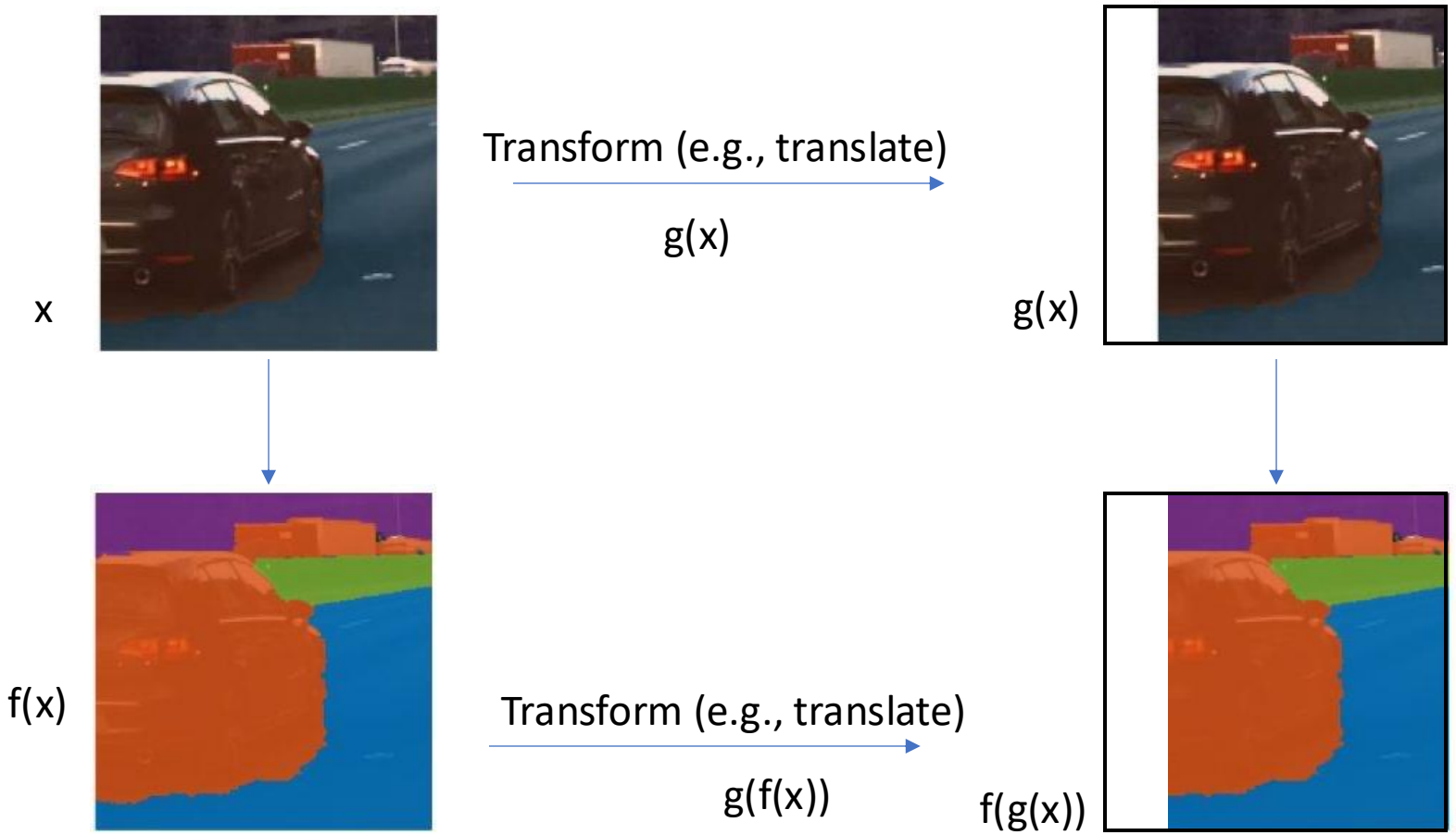
Prediction scores will change!

Figure: http://cs231n.github.io/linear-classify/

Input Image & Its Segmentation:
https://www.mathworks.com/discovery/image-segmentation.html

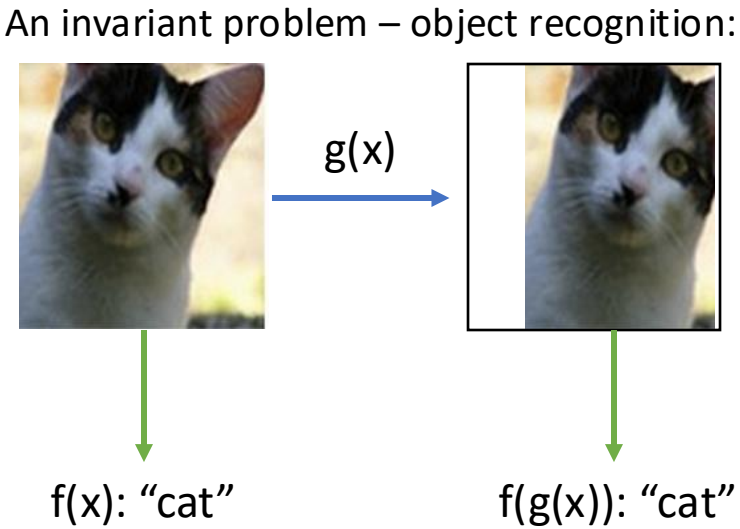
Equivariance vs. Invariance

- Equivariant problem: image segmentation: f(g(x)) = g(f(x))



Equivariance vs. Invariance

- Invariant problem: object recognition.
  - f(g(x)) = f(x)

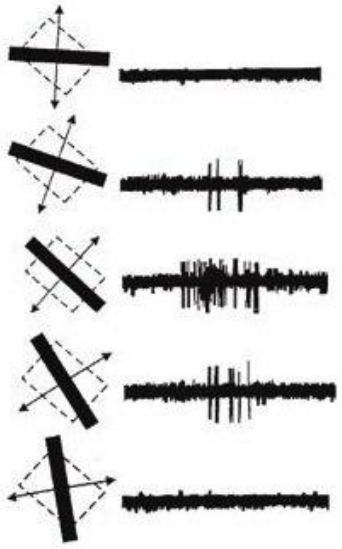


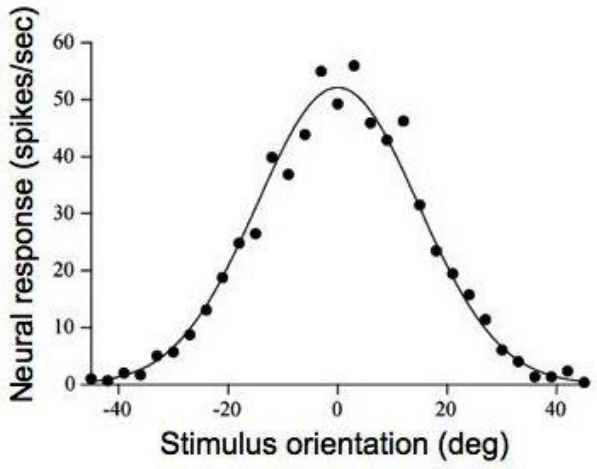
- Teaser for CNNs:
  - Pooling provides invariance, convolution provides equivariance. To a certain degree.

An Alternative to MLPs

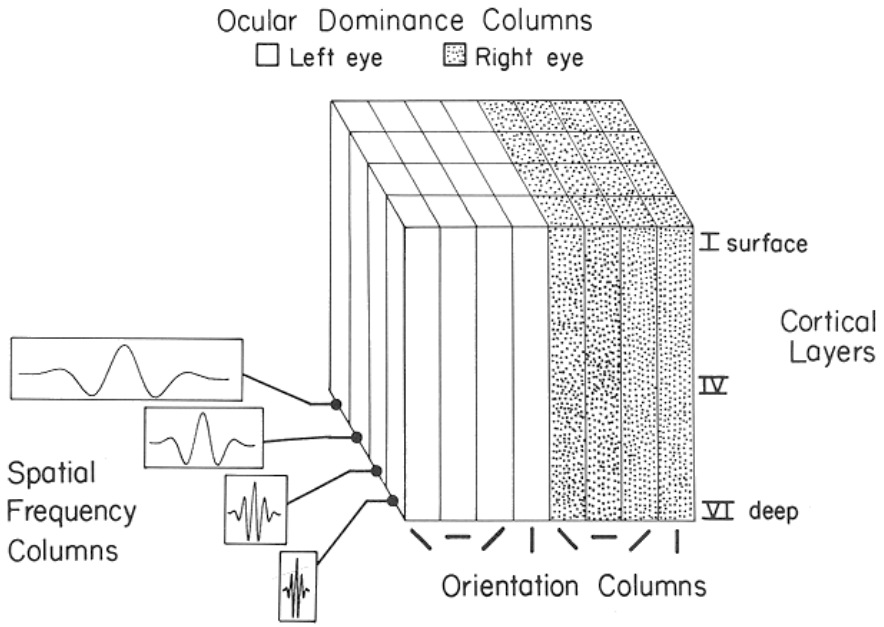
Solution (inspiration):

- Hubel & Wiesel (1960s): Brain neurons are not fully connected. They have local receptive fields





Hubel & Wiesel, 1968



Model of Striate Module in Cats

http://fourier.eng.hmc.edu/e180/lectures/retina/node1.html

An Alternative to MLPs

Solution (inspiration):

- Hubel & Wiesel: Brain neurons are not fully connected. They have local receptive fields

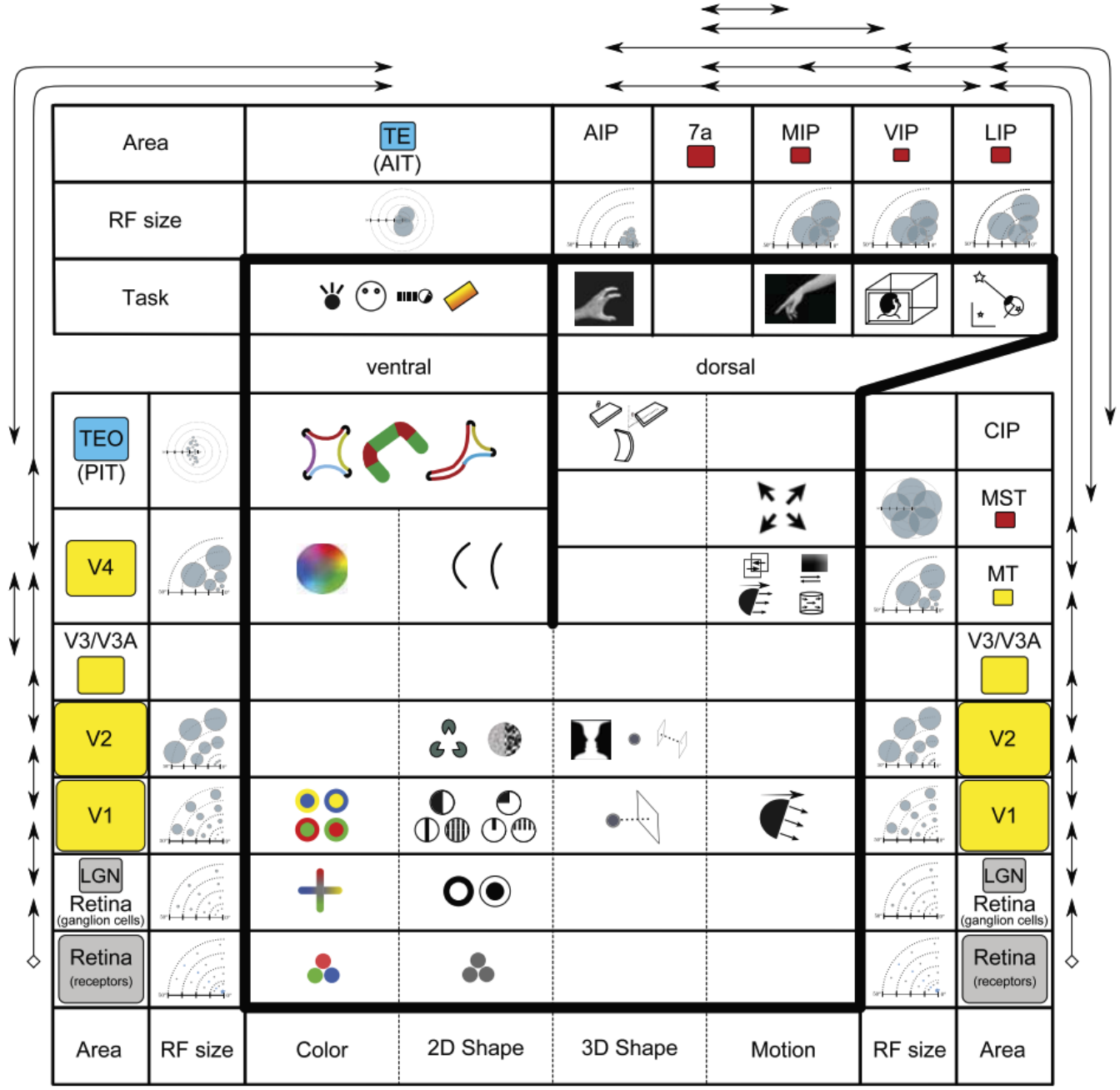


Figure: N. Krueger, P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, A. J. Rodriguez-Sanchez, L. Wiskott, "Deep Hierarchies in the Primate Visual Cortex: What Can We Learn For Computer Vision?", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 2013.

An Alternative to MLPs

Solution: Neocognitron (Fukushima, 1979):

A neural network model unaffected by shift in position, applied to Japanese handwritten character recognition.

- S (simple) cells: local feature extraction.
- C (complex) cells: provide tolerance to deformation, e.g. shift.
- Self-organization-based learning method.

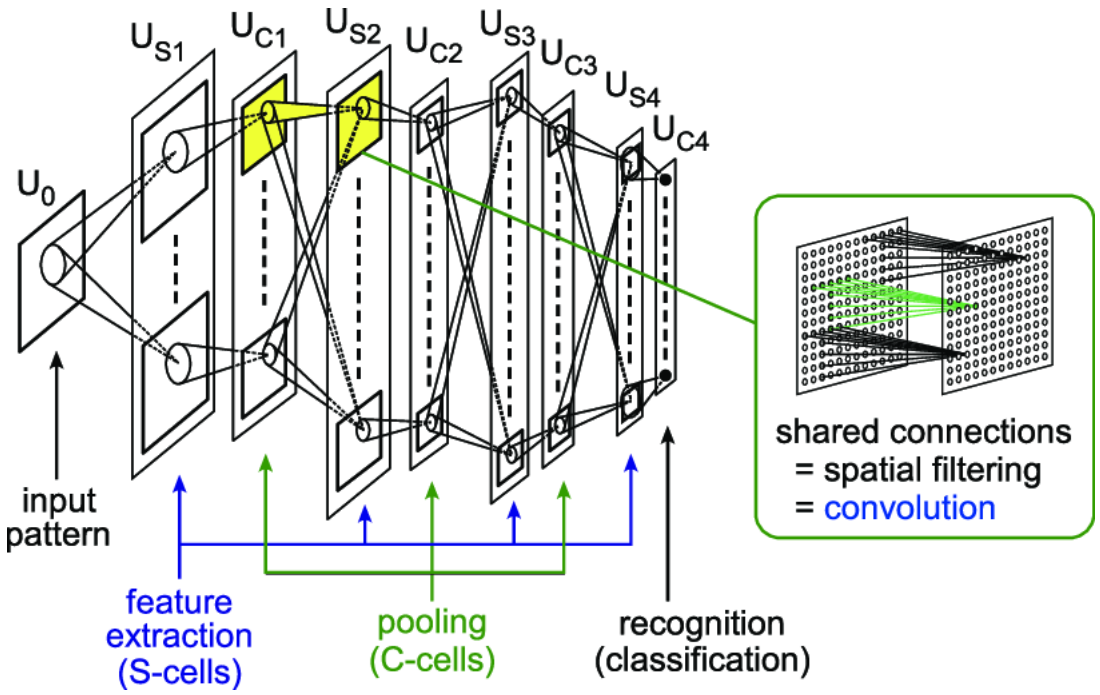


Figure: Fukushima (2019), Recent advances in the deep CNN neocognitron.

An Alternative to MLPs

Solution:

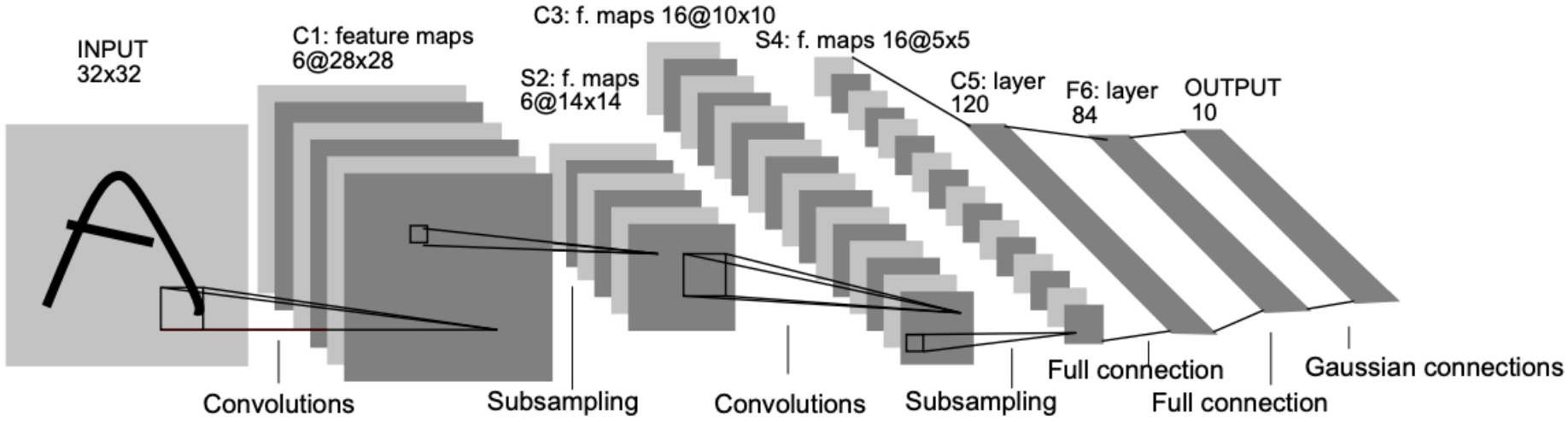
Neocognitron's self-organized learning method (Fukushima, 2019):

"For training intermediate layers of the neocognitron, the learning rule called AiS (Add-if-Silent) is used. Under the AiS rule, a new cell is generated and added to the network if all postsynaptic cells are silent in spite of non-silent presynaptic cells. The generated cell learns the activity of the presynaptic cells in one-shot. Once a cell is generated, its input connections do not change any more. Thus the training process is very simple and does not require time-consuming repetitive calculation."

An Alternative to MLPs

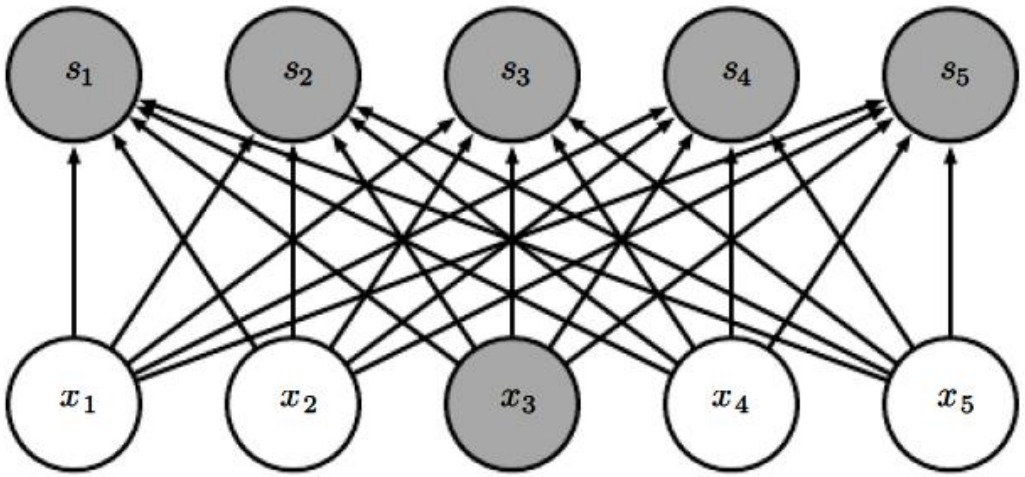
Solution: Convolutional Neural Networks (Lecun, 1998)

- Gradient descent
- Weights shared
- Document recognition



Lecun, 1998

CNNs: Underlying Principle



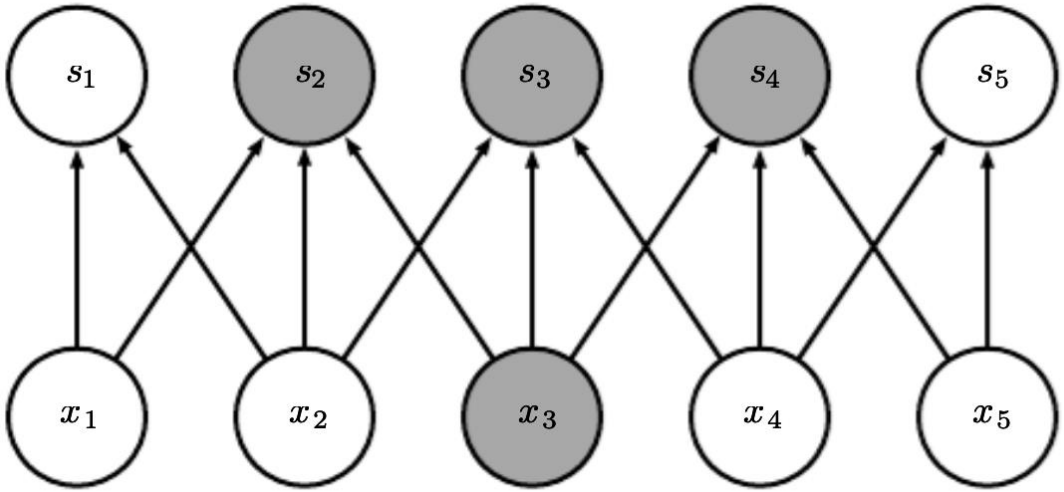


Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

CNNs vs. MLPs: Dimensionality

- A fully-connected network has too many parameters
  - On CIFAR-10:
    - Images have size 32x32x3 ➔ one neuron in hidden layer has 3072 weights!
  - With images of size 1024x1024x3 ➔ one neuron in hidden layer has 3,145,728 weights!
  - This explodes quickly if you increase the number of neurons & layers.
- Alternative: enforce local connectivity!

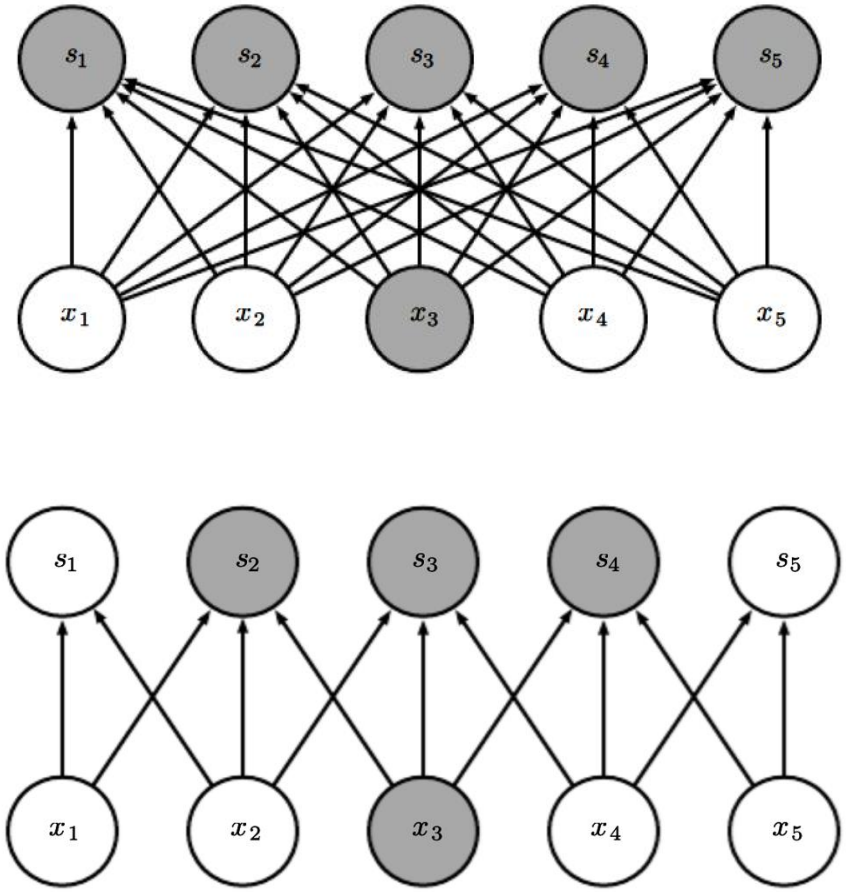
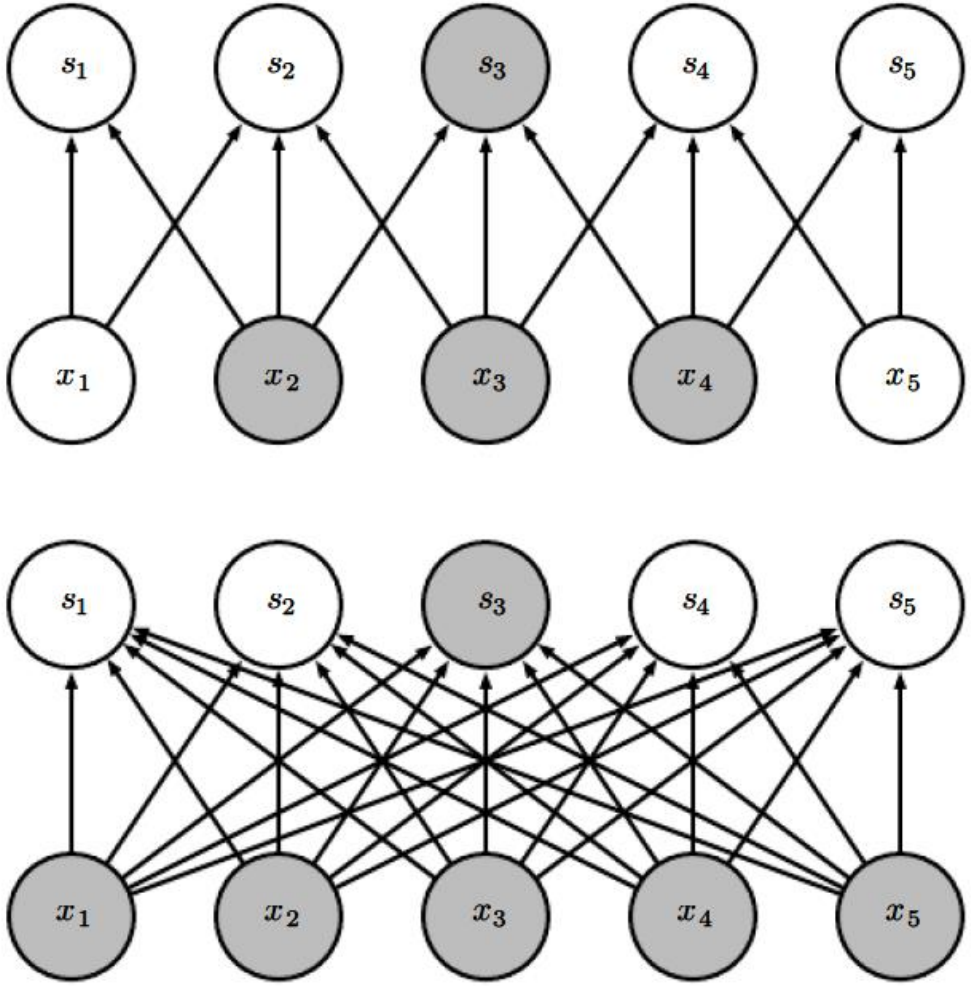


Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

CNNs vs. MLPs: Dimensionality



When things go deep, an output may depend on all or most of the input:

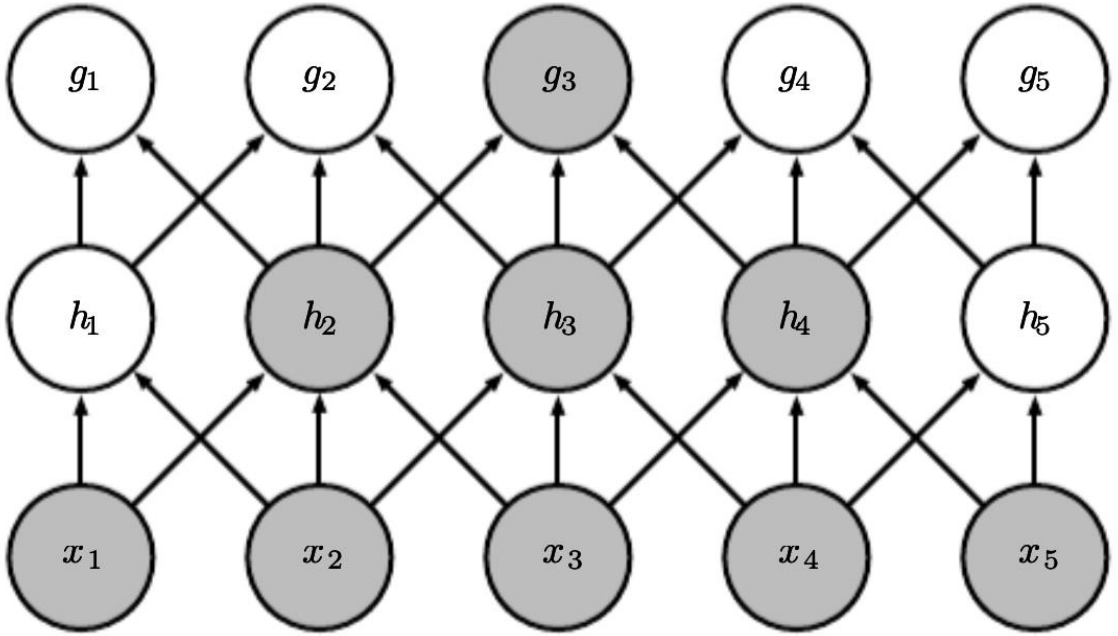


Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

How Many Samples are Needed to Learn a Convolutional Neural Network?

Simon S. Du*[1], Yining Wang*[1], Xiyu Zhai[2], Sivaraman Balakrishnan[3], Ruslan Salakhutdinov[1], and Aarti Singh[1]

[1]Machine Learning Department, Carnegie Mellon University
[2]University of Cambridge
[3]Department of Statistics, Carnegie Mellon University

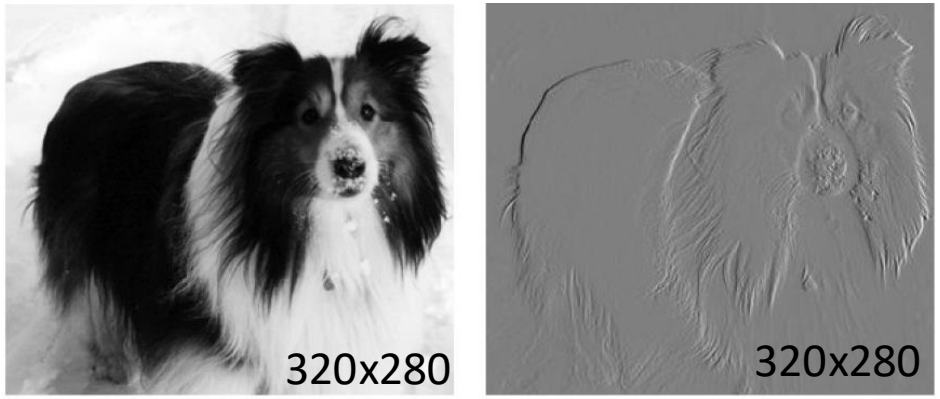
May 22, 2018

Abstract

A widespread folklore for explaining the success of convolutional neural network (CNN) is that CNN is a more compact representation than the fully connected neural network (FNN) and thus requires fewer samples for learning. We initiate the study of rigorously characterizing the sample complexity of learning convolutional neural networks. We show that for learning an m-dimensional convolutional filter with linear activation acting on a d-dimensional input, the sample complexity of achieving population prediction error of ϵ is $\widetilde{O}(m/\epsilon^2)$[1], whereas its FNN counterpart needs at least $\Omega(d/\epsilon^2)$ samples. Since $m \ll d$, this result demonstrates the advantage of using CNN. We further consider the sample complexity of learning a one-hidden-layer CNN with linear activation where both the m-dimensional convolutional filter and the r-dimensional output weights are unknown. For this model, we show the sample complexity is $\widetilde{O}\left((m+r)/\epsilon^2\right)$ when the ratio between the stride size and the filter size is a constant. For both models, we also present lower bounds showing our sample complexities are tight up to logarithmic factors. Our main tools for deriving these results are localized empirical process and a new lemma characterizing the convolutional structure. We believe these tools may inspire further developments in understanding CNN.

CNNs vs. MLPs: Dimensionality

- Parameter sharing
  - In regular ANN, each weight is independent
- In CNN, a layer might re-apply the same convolution and therefore, share the parameters of a convolution
  - Reduces storage and learning time



- For a neuron in the next layer:
  - With ANN: 320x280x320x280 multiplications
  - With CNN: 320x280x3x3 multiplications

Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

CNNs vs. MLPs: Equivariance & Invariance

- Equivariant to translation
  - The output will be the same, just translated, since the weights are shared.

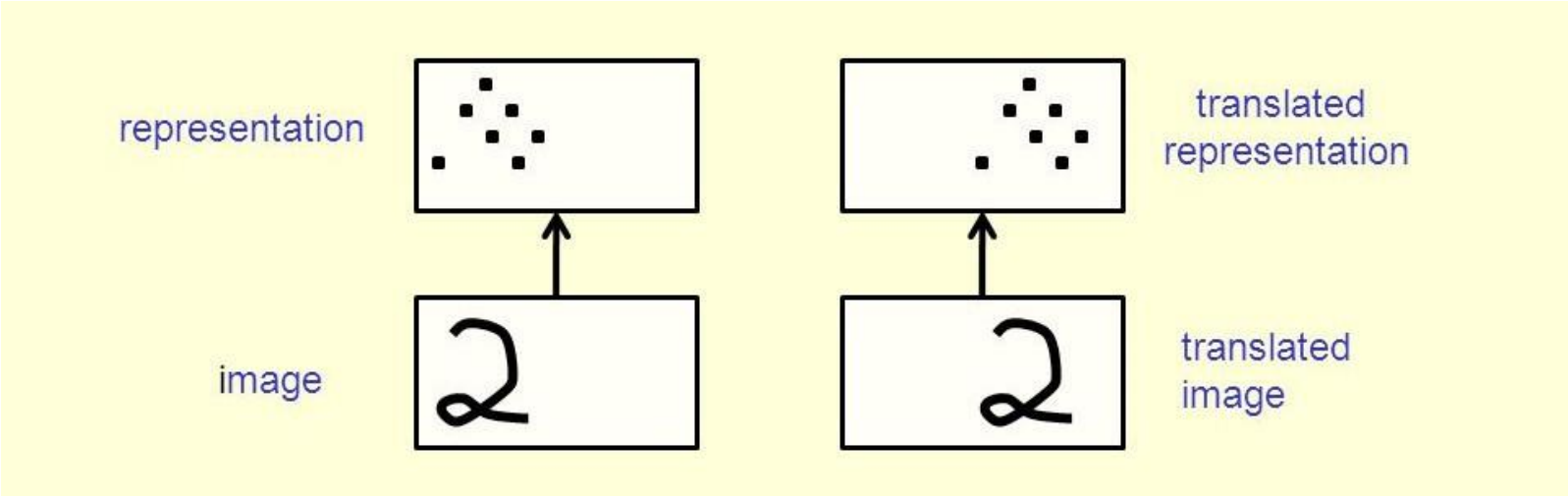
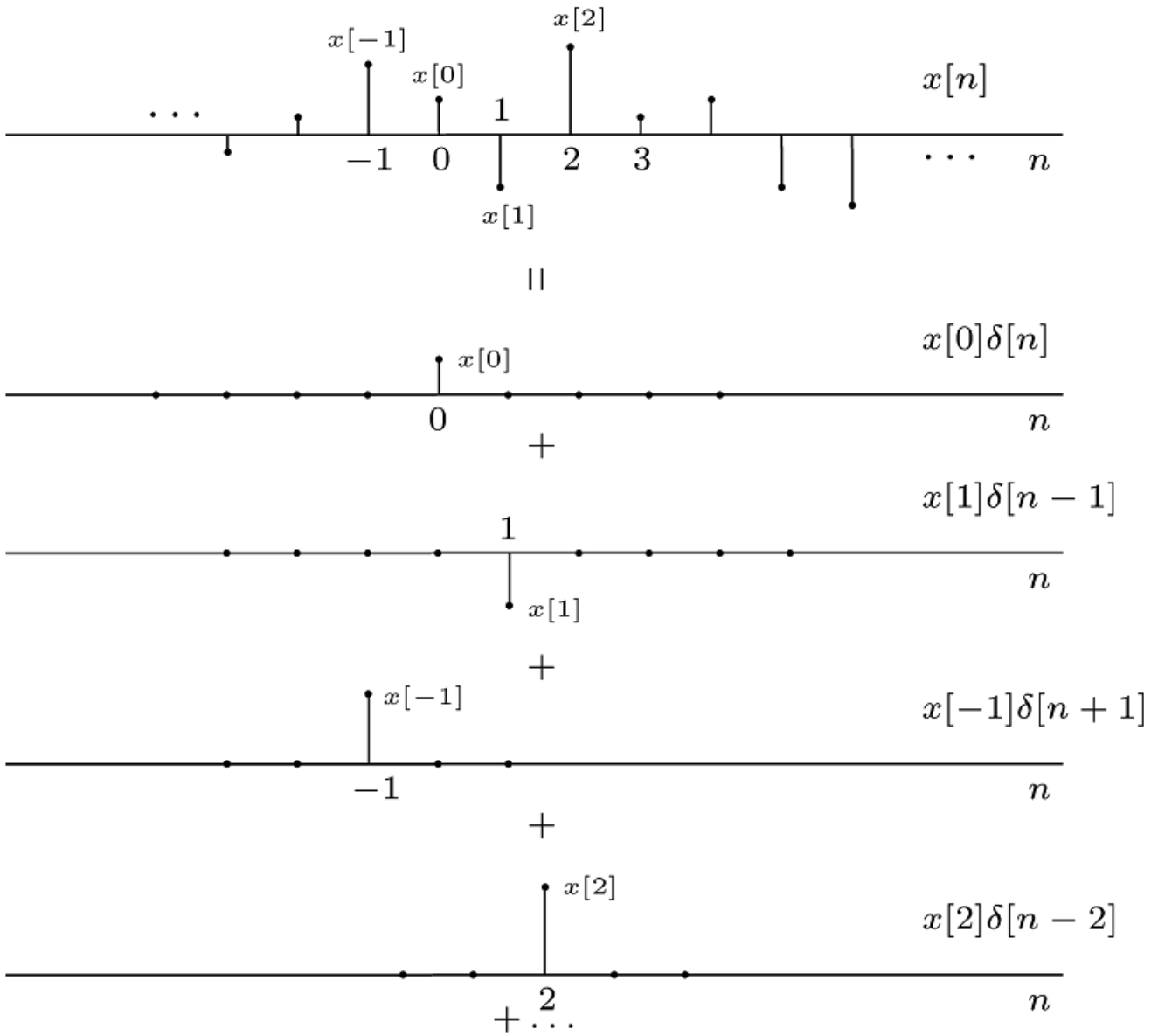


Figure: https://towardsdatascience.com/translational-invariance-vs-translational-equivariance-f9fbc8fca63a

- Not equivariant to scale or rotation.

A Crash Course on Convolution

Formulating Signals in Terms of Impulse Signal



Alan V. Oppenheim and Alan S. Willsky

Formulating Signals in Terms of Impulse Signal

$$x[n] = \ldots + x[-2]\delta[n+2] + x[-1]\delta[n+1] + x[0]\delta[n] + x[1]\delta[n-1] + \ldots$$

$$\Downarrow$$

Important to note the "-" sign

$$x[n] = \sum_{k=-\infty}^{+\infty} \underbrace{x[k]}_{\text{Coefficients}} \underbrace{\delta[n-k]}_{\text{Basic Signals}}$$

Alan V. Oppenheim and Alan S. Willsky

Unit Sample Response

- Now suppose the system is **LTI**, and define the *unit sample response* $h[n]$:

$$\delta[n] \longrightarrow h[n]$$

$$\Downarrow$$

From Time-Invariance:

$$\delta[n-k] \longrightarrow h[n-k]$$

From Linearity:

$$x[n] = \sum_{k=-\infty}^{+\infty} x[k]\,\delta[n-k] \longrightarrow y[n] = \underbrace{\sum_{k=-\infty}^{+\infty} x[k]\,h[n-k]}_{convolution\ sum} = x[n] * h[n]$$

Alan V. Oppenheim and Alan S. Willsky

Conclusion

The output of *any* DT LTI System is a convolution of the input signal with the unit-sample response, *i.e.*

$$\text{Any DT LTI} \longleftrightarrow y[n] = x[n] * h[n] = \sum_{k=-\infty}^{+\infty} x[k]\, h[n-k]$$

As a result, any DT LTI Systems are *completely characterized* by its unit sample response

Alan V. Oppenheim and Alan S. Willsky

Power of convolution

- Describe a “system” (or operation) with a very simple function (impulse response).
- Determine the output by convolving the input with the impulse response

Convolution

- Definition of continuous-time convolution

$$x(t) * h(t) = \int x(\tau)h(t-\tau)\,d\tau$$

$$h(\tau) \xrightarrow{Flip} h(-\tau) \xrightarrow{Slide} h(t-\tau) \xrightarrow{Multiply}$$

$$x(\tau)h(t-\tau) \xrightarrow{Integrate} \int_{-\infty}^{+\infty} x(\tau)h(t-\tau)d\tau$$

Alan V. Oppenheim and Alan S. Willsky

Convolution

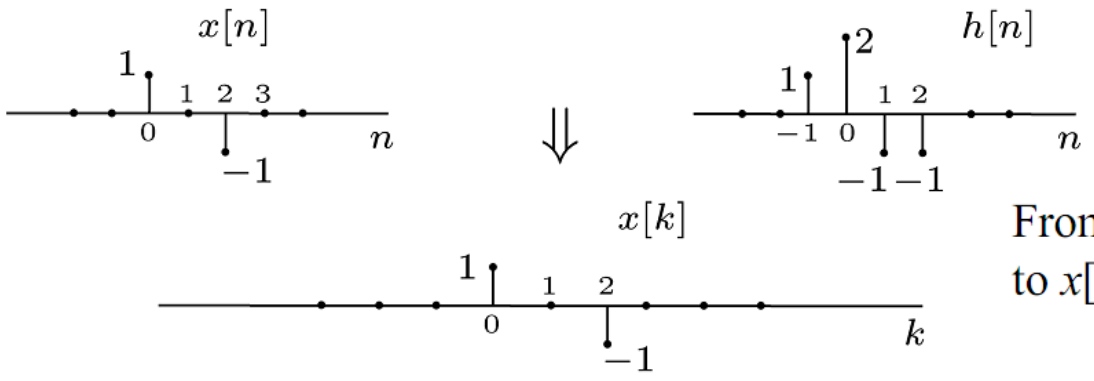
- Definition of discrete-time convolution

$$x[n] * h[n] = \sum x[k]h[n-k]$$

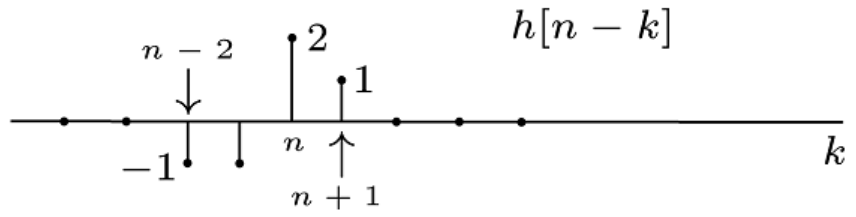
Choose the value of n and consider it fixed

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

View as functions of k with n fixed



From $x[n]$ and $h[n]$ to $x[k]$ and $h[n-k]$



Alan V. Oppenheim and Alan S. Willsky

Discrete-time 2D Convolution

- For images, we need two-dimensional convolution:

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[m,n]K[i-m, j-n]$$

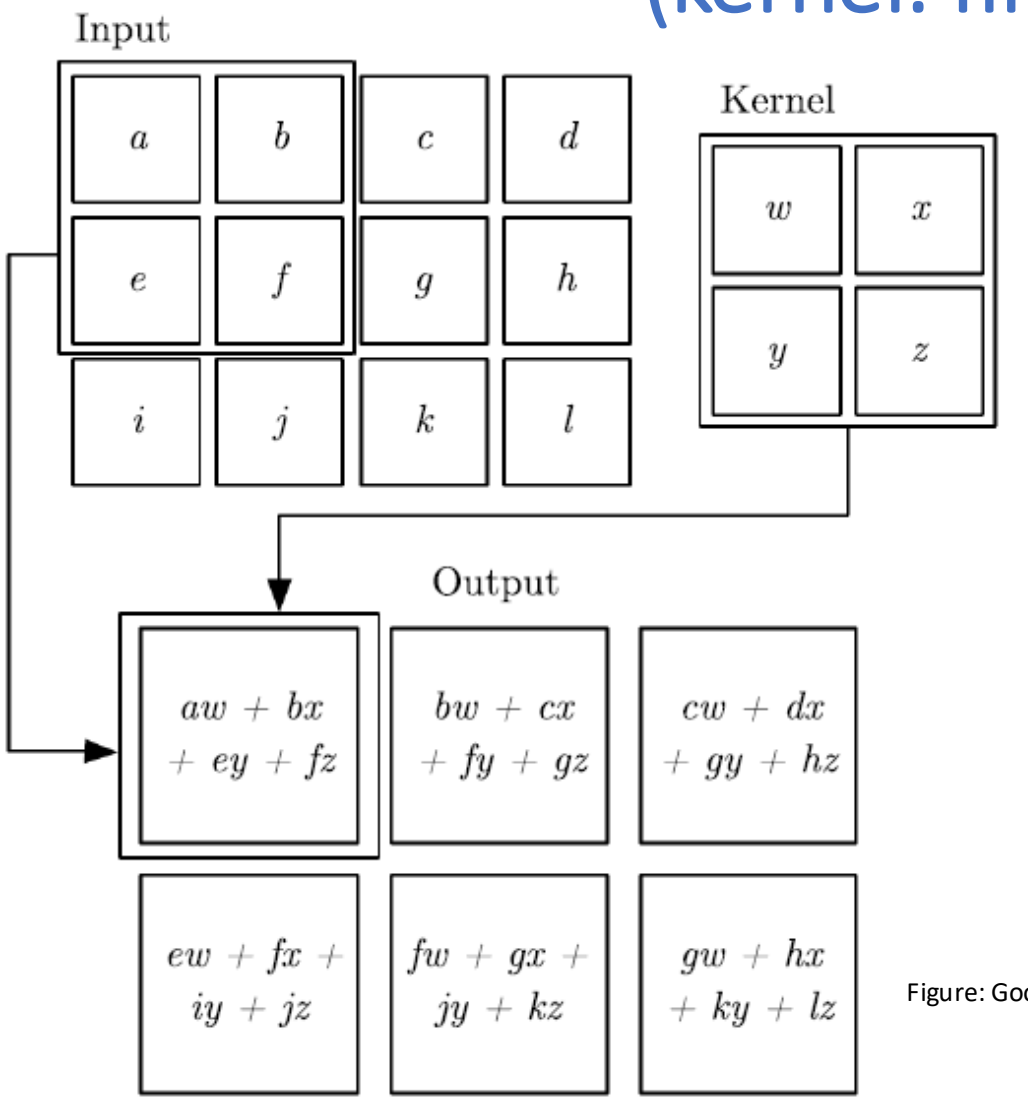
- These multi-dimensional arrays are called tensors
- We have commutative property:

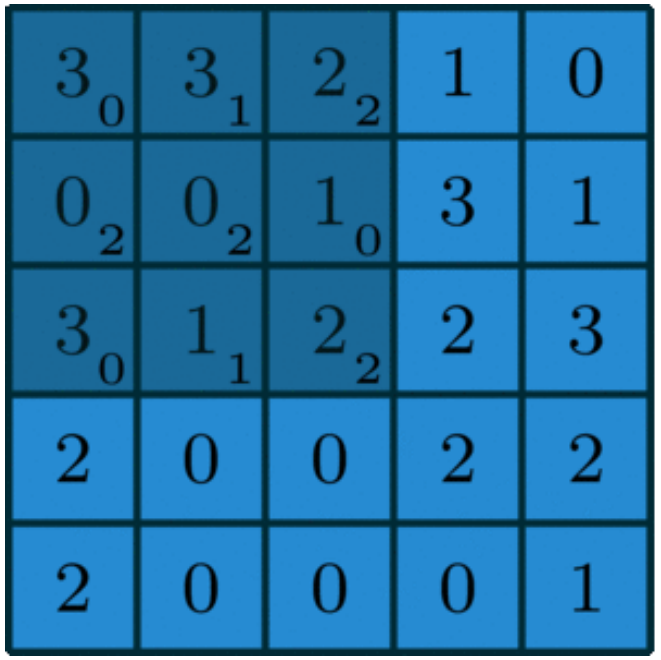
$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i-m, j-n]K[m,n]$$

- Instead of subtraction, we can also write (easy to drive by a change of variables). This is called cross-correlation:

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i+m, j+n]K[m,n]$$

Example multi-dimensional convolution
(kernel: finite impulse response)





https://github.com/vdumoulin/conv_arithmetic

Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

What can filters do?
Rectangular filter

g[m,n]

$\otimes$

h[m,n]

=

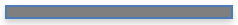
f[m,n]

Slide: A. Torralba

What can filters do?
1D filter

g[m,n]

$\otimes$



h[m,n]

=

f[m,n]

Slide: A. Torralba

What can filters do?
1D filter

g[m,n]

$\otimes$

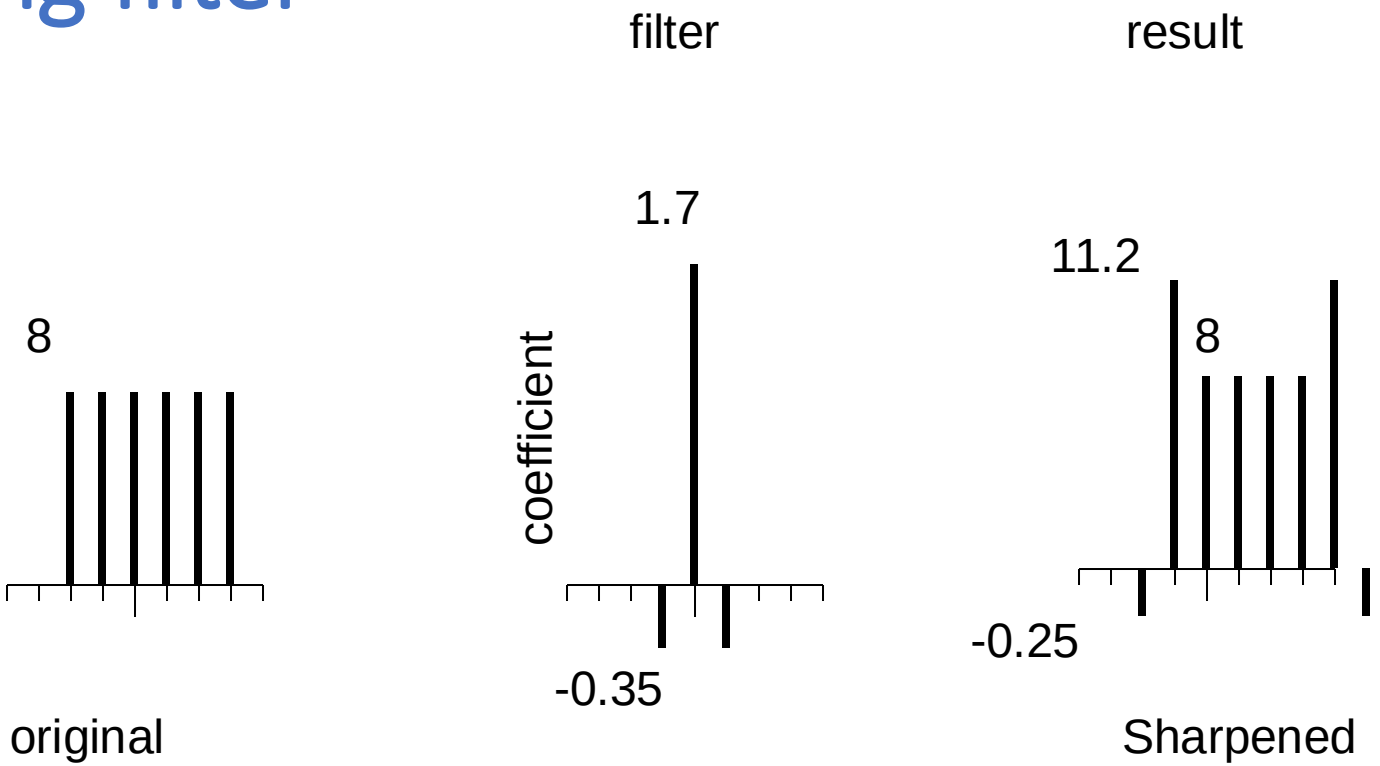
h[m,n]

=

f[m,n]

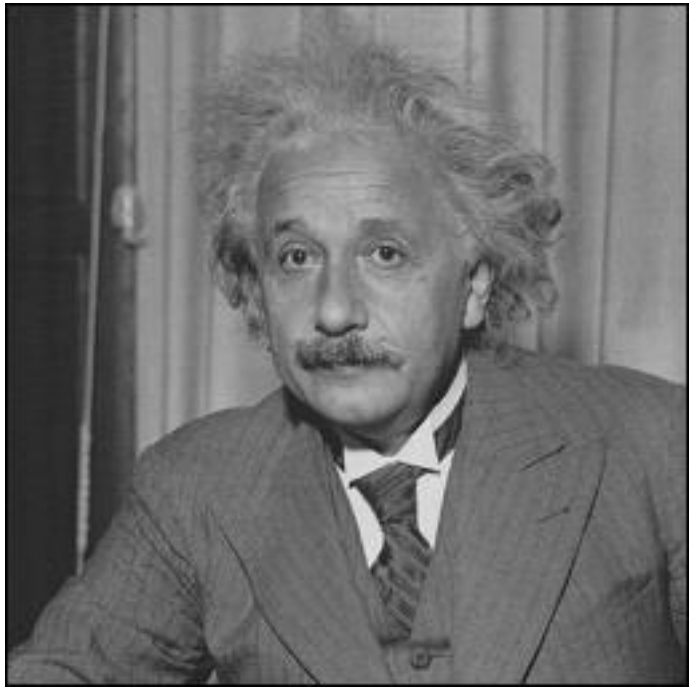
Slide: A. Torralba

What can filters do?
Sharpening filter

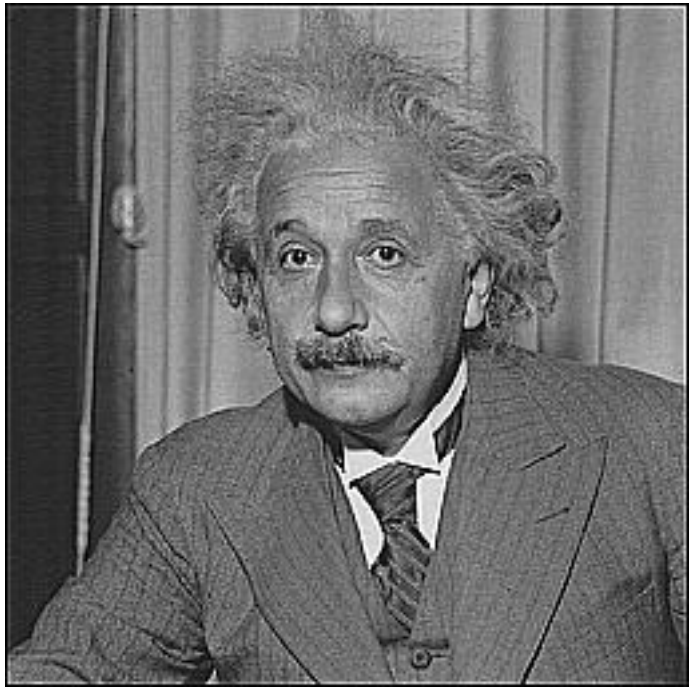


Slide: A. Torralba

What can filters do?
Sharpening filter



before

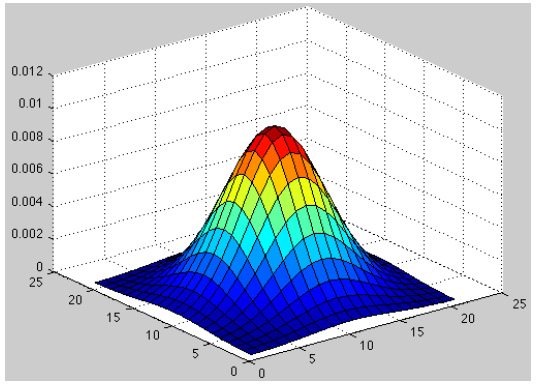


after

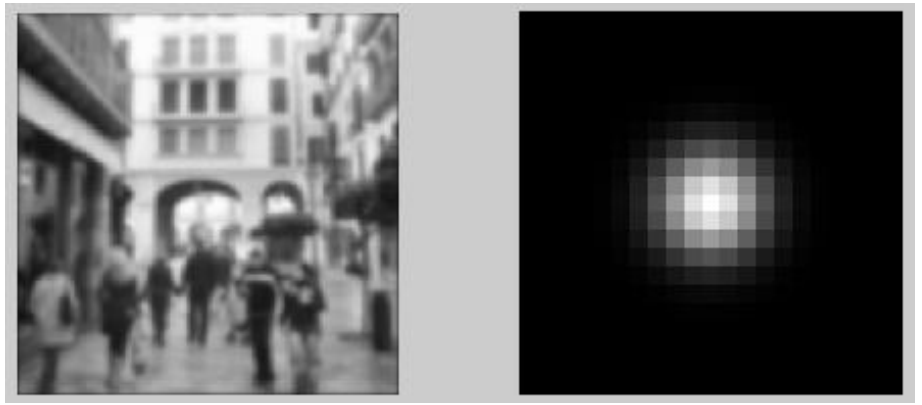
Slide: A. Torralba

What can filters do? Gaussian filter

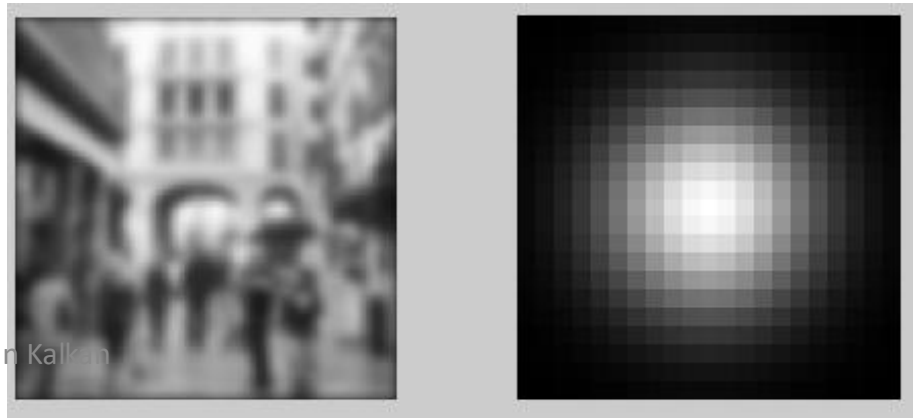
$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



σ=1



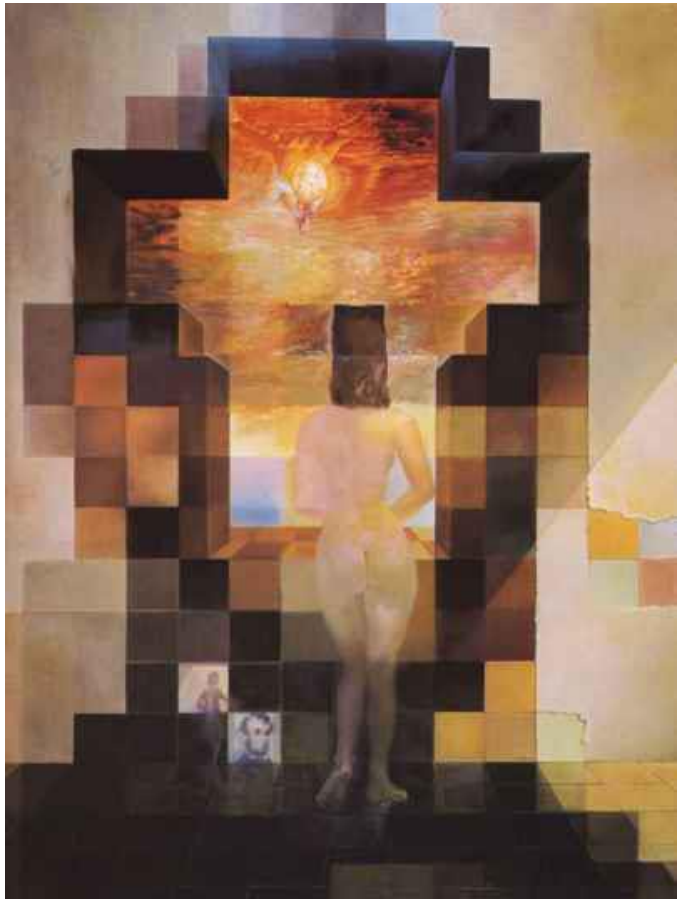
σ=2



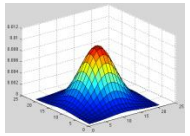
σ=4

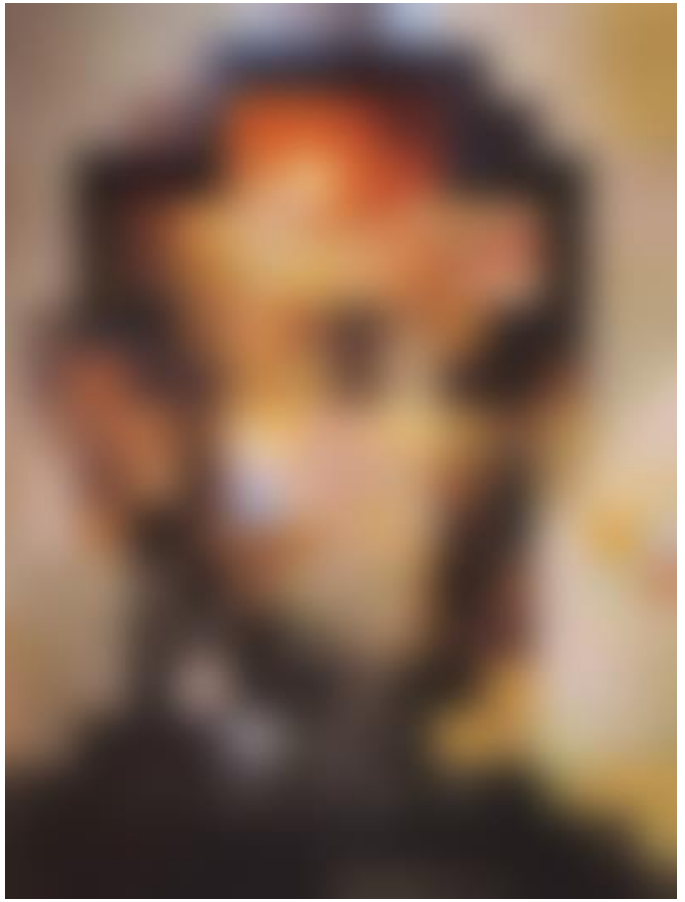
Slide: A. Torralba

Global to Local Analysis



Dali





Slide: A. Torralba

What can filters do?

[-1 1]

g[m,n]

$\otimes$

[-1, 1]

h[m,n]

=

f[m,n]

Slide: A. Torralba

What can filters do?

$[-1\ 1]^T$

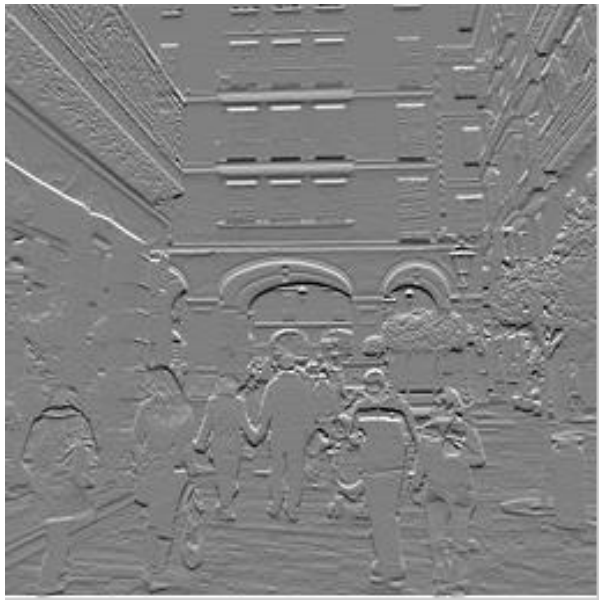
g[m,n]

$\otimes$

$[-1, 1]^T$

h[m,n]

=



f[m,n]

Slide: A. Torralba

Overview of CNN

CNN layers

- Operations in a CNN:
  - Convolution (in parallel) to produce pre-synaptic activations
  - Detector: Non-linear function
  - Pooling: A summary of a neighborhood

- Pooling of a region in a feature/activation map:
  - Max
  - Average
  - L2 norm
  - Weighted average acc. to the distance to the center
  - …

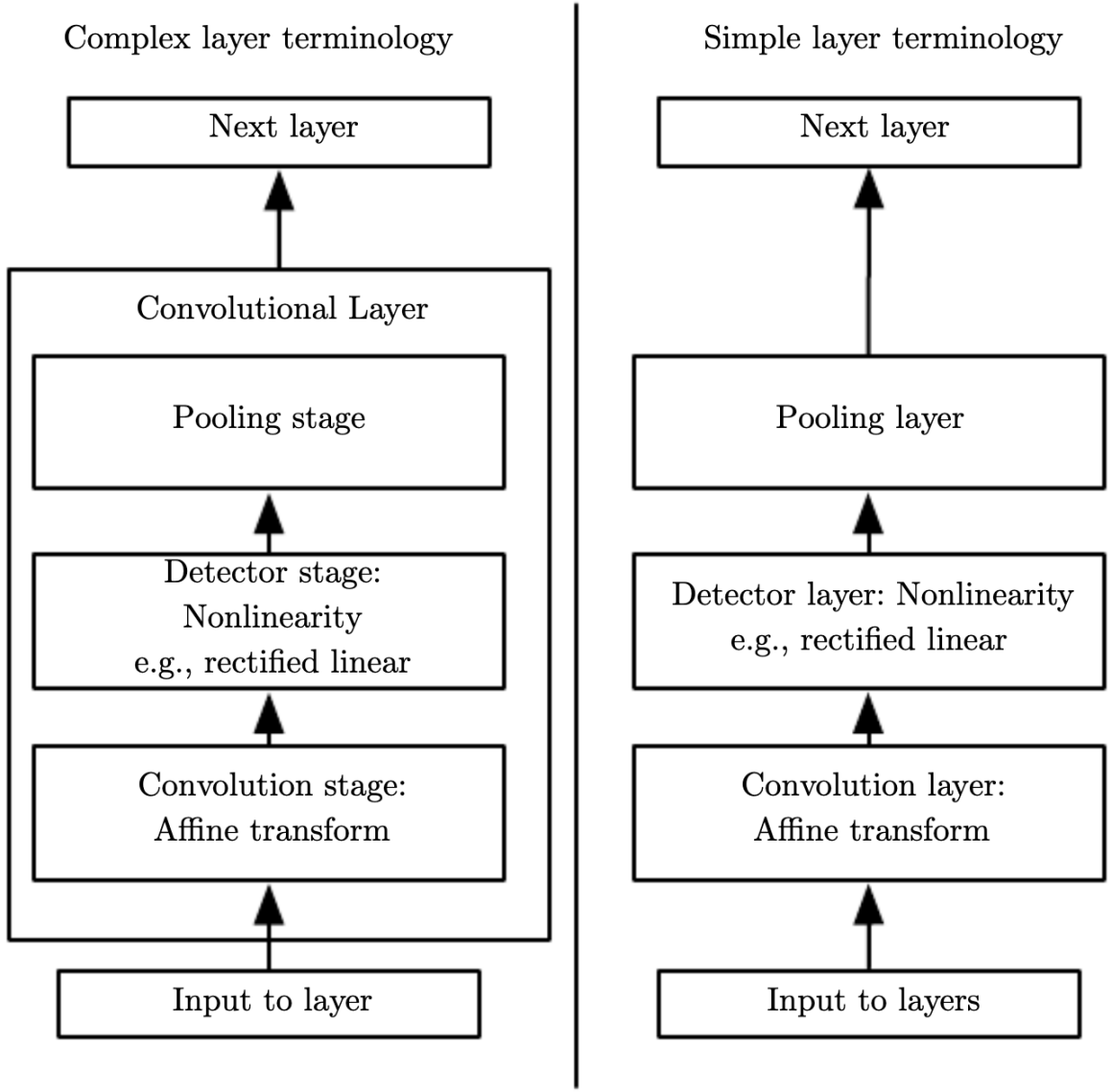
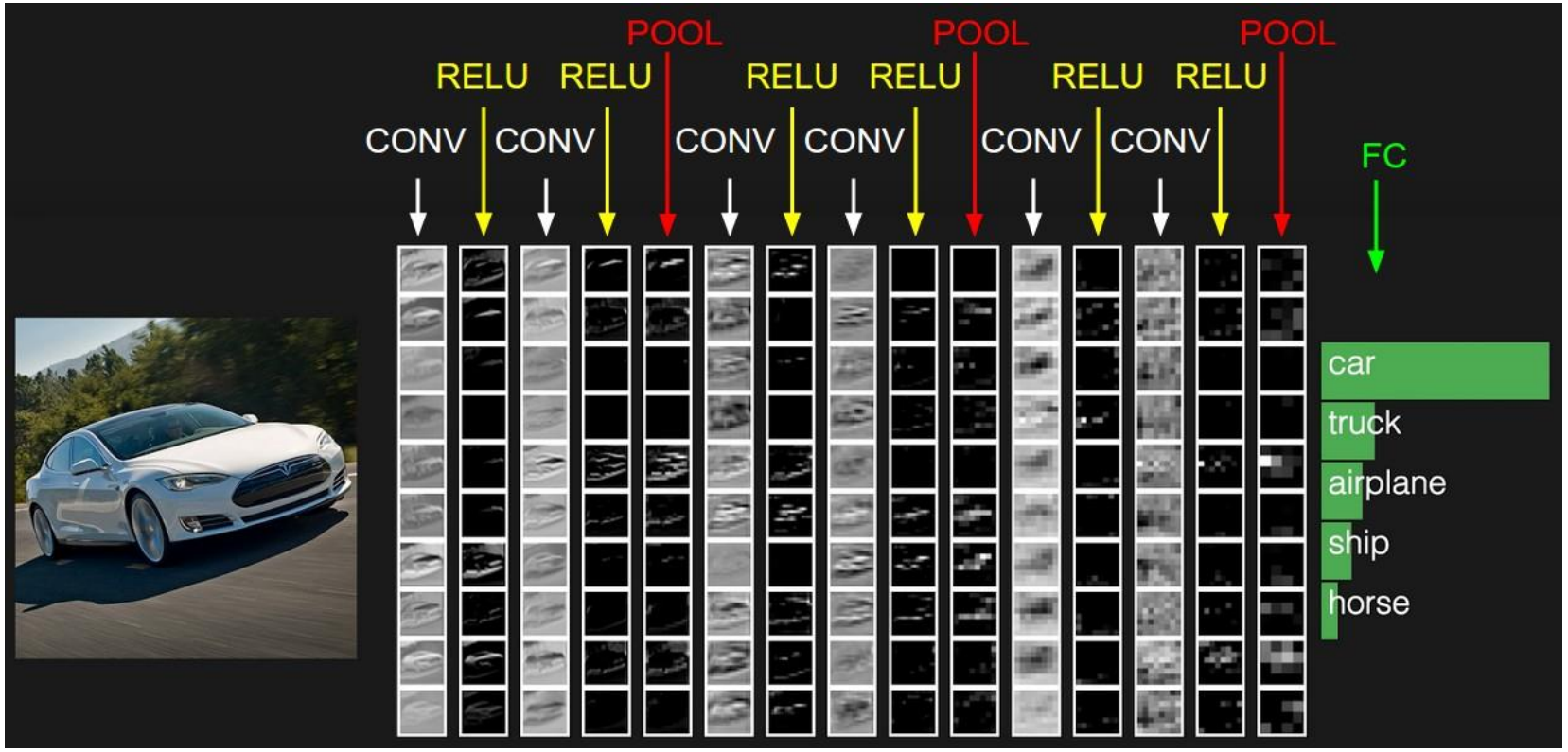


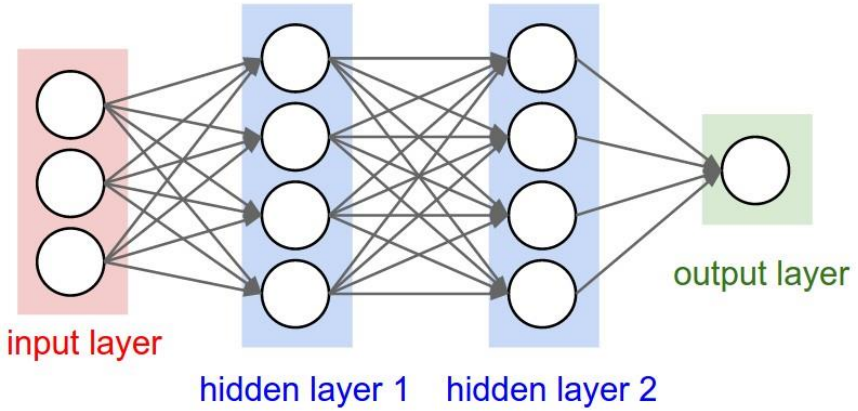
Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

An example architecture

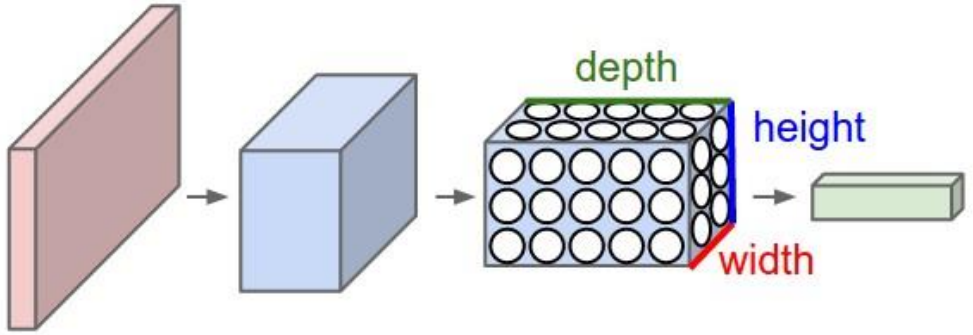


http://cs231n.github.io/convolutional-networks/

Regular ANN



CNN



http://cs231n.github.io/convolutional-networks/

OPERATIONS IN A CNN:
Convolution

Convolution in CNN

- The weights correspond to the kernel
- The weights are shared in a channel (depth slice)
- We are effectively learning filters that respond to some part/entities/visual-cues etc.

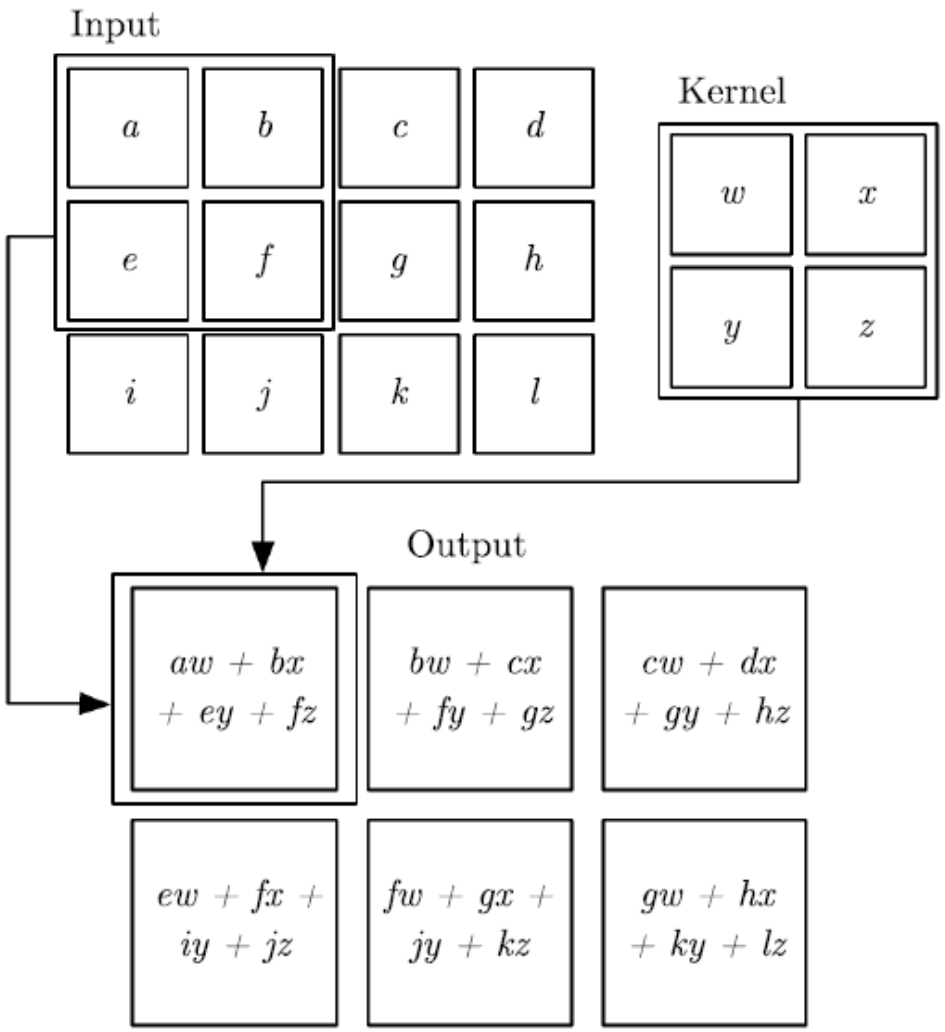


Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

Local connectivity in CNN
= Receptive fields

- Each neuron is connected to only a local neighborhood, i.e., receptive field
- The size of the receptive field ➔ another hyper-parameter.

Connectivity in CNN

- Local: The behavior of a neuron does not change other than being restricted to a subspace of the input.
- Each neuron is connected to a slice of the previous layer
- A layer is actually a volume having a certain width x height and depth (or channel)
- A neuron is connected to a subspace of width x height but to all channels (depth)
- Example: CIFAR-10
  - Input: 32 x 32 x 3 (3 for RGB channels)
  - A neuron in the next layer with receptive field size 5x5 has input from a volume of 5x5x3.