7. FROM SHALLOW TO DEEP NEURAL NETWORKS

Stephan Robert-Nicoud
HEIG-VD/HES-SO

Credit: Andres Perez-Uribe

BY NC SA

Objectives

- ❑ Understand how we got from shallow Neural Networks to Deep Neural Networks
- ❑ Better understand how do CNNs work
- ❑ Understand some practical considerations and tricks to put CNNs in practice

Shallow Neural Networks

- The "Universal approximation" theorem stated that neural networks with a single hidden layer can represent a wide variety of interesting functions when given appropriate parameters.

  Cybenko., G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314

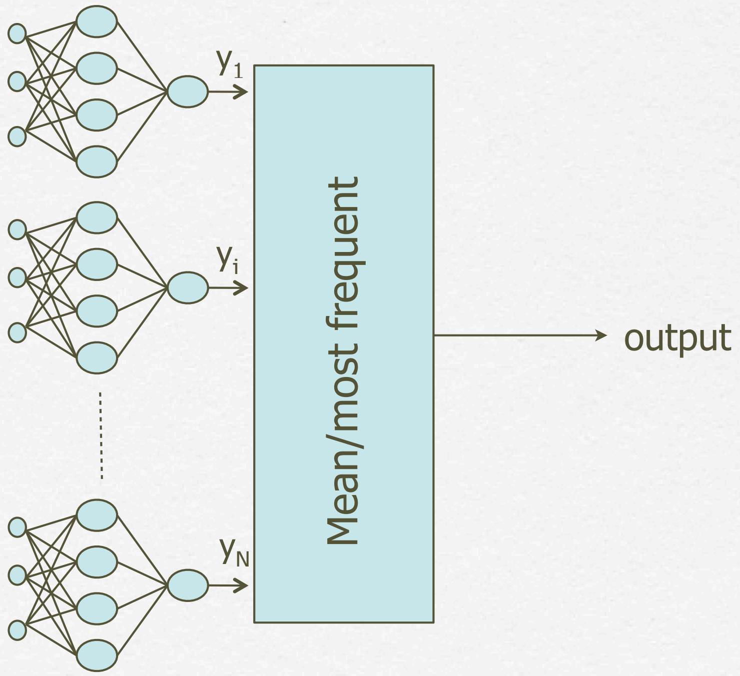
- Before 2006: training a deep feedforward neural network yielded worse results (both in training and in test error), then shallow networks (with 1 or 2 hidden layers) were preferred

“Wide but shallow” neural networks

- The availability of increasing computational capabilities allowed engineers to train multiple shallow models while searching for better generalization performances.
- The result is the development of “ensemble models”. For instance, models composed of multiple neural networks as follows:
  - multiple instances of the same neural network model trained on the same database, or
  - multiple instances of the same neural network model but trained using different subsets of the training dataset, or
  - multiple variations of shallow neural networks (e.g., having different architectures).

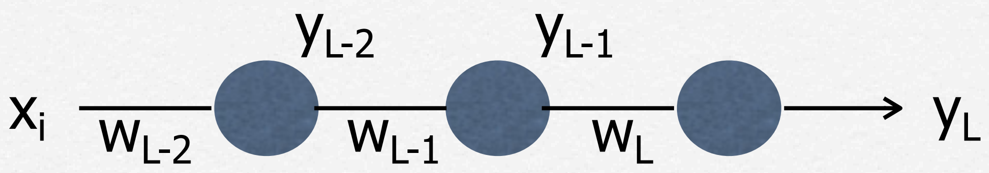
Ensemble neural networks

Ensemble averaging is the process of creating multiple models and combining them to produce a desired output, as opposed to creating just one model. Frequently an ensemble of models performs better than any individual model, because the various errors of the models "average out."



Vanishing gradients problem (1)

$E = 1/2\Sigma(t-y_L)^2$



$Y_L = f_L(\Sigma W_L Y_{L-1}+b_L) = f_L(a_L)$

$Y_{L-1} = f_{L-1}(\Sigma W_{L-1} Y_{L-2}+b_{L-1})$

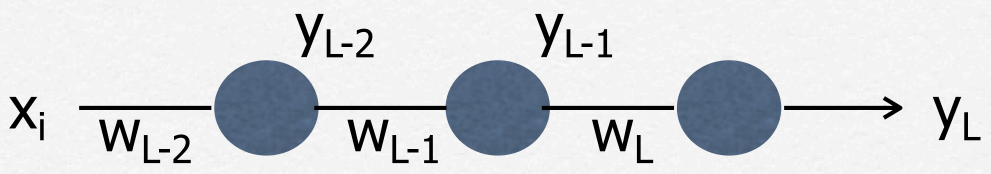
$Y_{L-2} = f_{L-2}(\Sigma W_{L-2}\, x_i + b_{L-2})$

$\partial E/\partial W_{L-2} = \partial E/\partial y_L\ \partial y_L/\partial y_{L-1}\ \partial y_{L-1}/\partial y_{L-2}\ \partial y_{L-2}/\partial W_{L-2}$

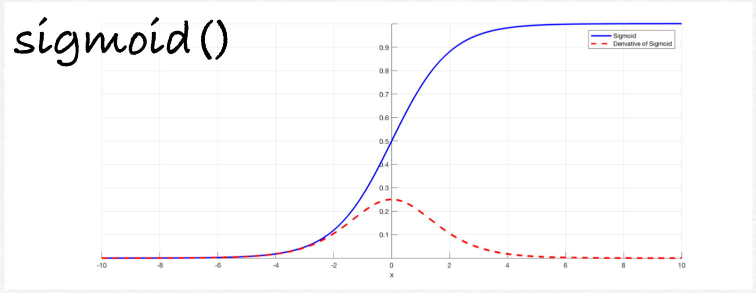
$\partial E/\partial W_{L-2} = \partial E/\partial y_L\ f_L'()\ \partial a_L/\partial y_{L-1}\ f_{L-1}'()\ \partial a_{L-1}/\partial y_{L-2}\ f_{L-2}'()\ \partial a_{L-2}/\partial W_{L-2}$

Vanishing gradients problem (2)

$E = 1/2\Sigma(t-y_L)^2$



$$\partial E/\partial w_{L-2} = \partial E/\partial y_L \; f_L'() \; \partial a_L/\partial y_{L-1} \; f_{L-1}'() \; \partial a_{L-1}/\partial y_{L-2} \; f_{L-2}'() \; \partial a_{L-2}/\partial w_{L-2}$$



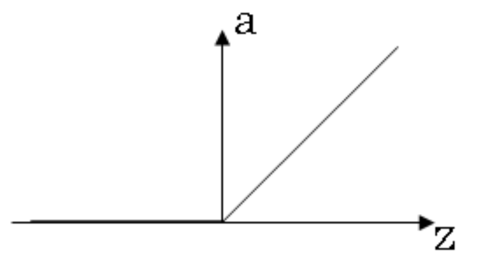
$0 < f'() < 0{,}25$

$\max(f_L'() * f_{L-1}'() * f_{L-2}'()) = 0.25^3 \simeq 0.016$

The gradient becomes very small and does not provide adequate information about how to adapt the weights in the first layers

New activation functions

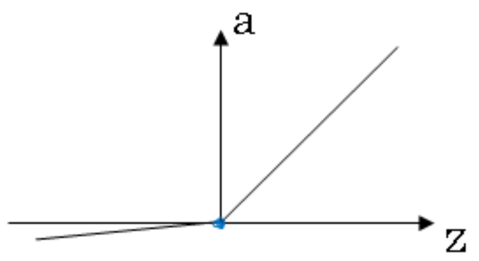
ReLU and Leaky ReLU



ReLU

$g(z) = \max(0, z)$

$$\begin{aligned} g'(z) &= 0 \text{ if } z < 0 \\ &= 1 \text{ if } z > 0 \\ &= \text{undefined at } z = 0 \end{aligned}$$



Leaky ReLU

$g(z) = \max(0.01z, z)$

$$\begin{aligned} g'(z) &= 0.01 \text{ if } z < 0 \\ &= 1 \text{ if } z > 0 \\ &= \text{undefined at } z = 0 \end{aligned}$$

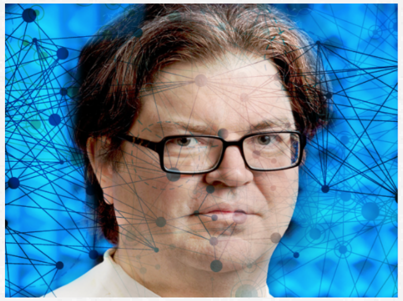
ReLU: Rectified Learning Unit

Leaky ReLU function

- Fei Fei Li and Andrej Karpathy recommend to never use sigmoids! (slides from 01/2015)

Deep Neural Networks

Geoffrey Hinton


Yann LeCun

- This technique allows the best (current) performance in image processing benchmarks (and many other problems)
- Key people: G. Hinton (U of Toronto & Google), Y. LeCun (NYU & Facebook), Y. Bengio (U of Montreal), J. Schmidhuber (IDSIA & USI, Switzerland)
- Pioneering ideas were published in the late 1980's and early 1990's.

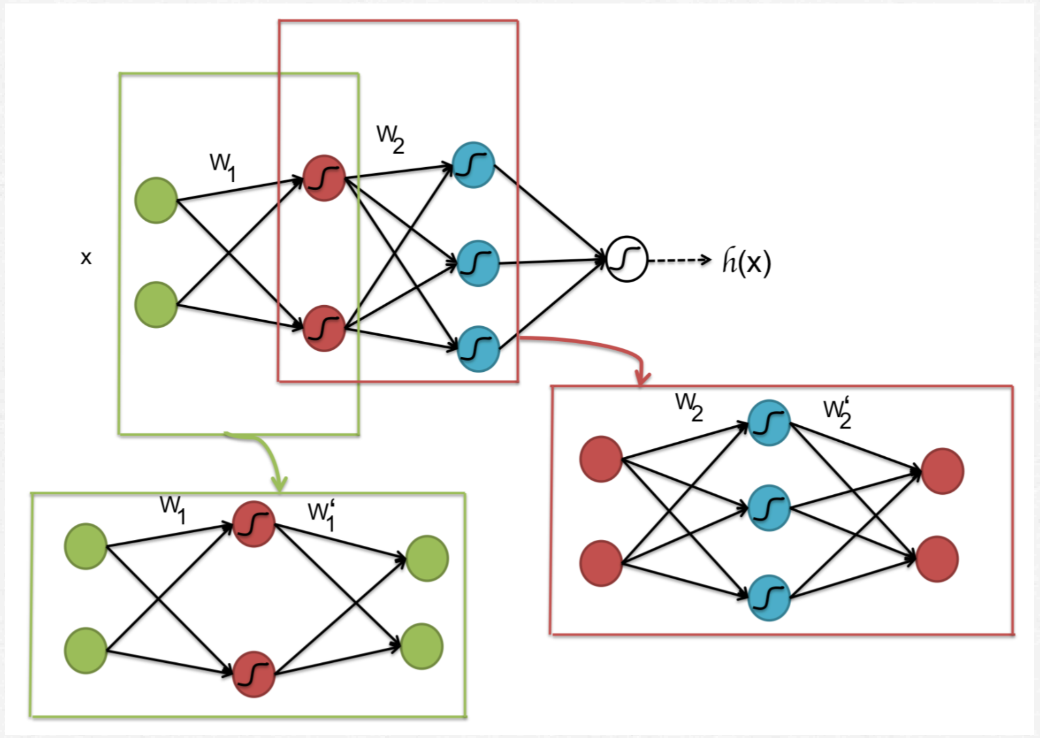
Yoshua Bengio

Juergen Schmidhuber

Deep networks: (first) two flavors

- Deep Belief Networks: consist on multiple layers of stochastic neural networks called Restricted Boltzmann Machines (RBM). They have bidirectional and symmetric connections between neurons. They recursively learn layers of feature detectors in an unsupervised way. Not so successful after all…
- Convolutional Neural Networks (CNN): consist of multiple layers of convolutions, or spatial filters that extract features, subsampling layers and fully connected layers. Many state-of-the-art solutions are based on these models.

A trick to train deep neural networks

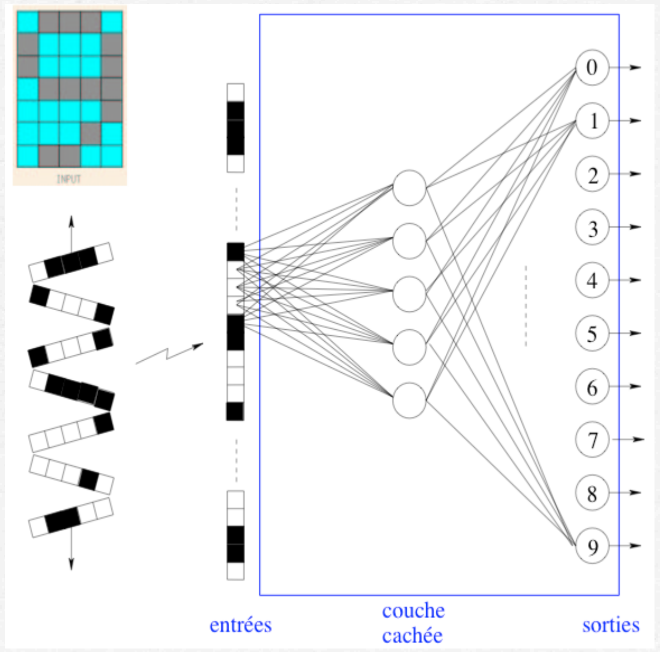


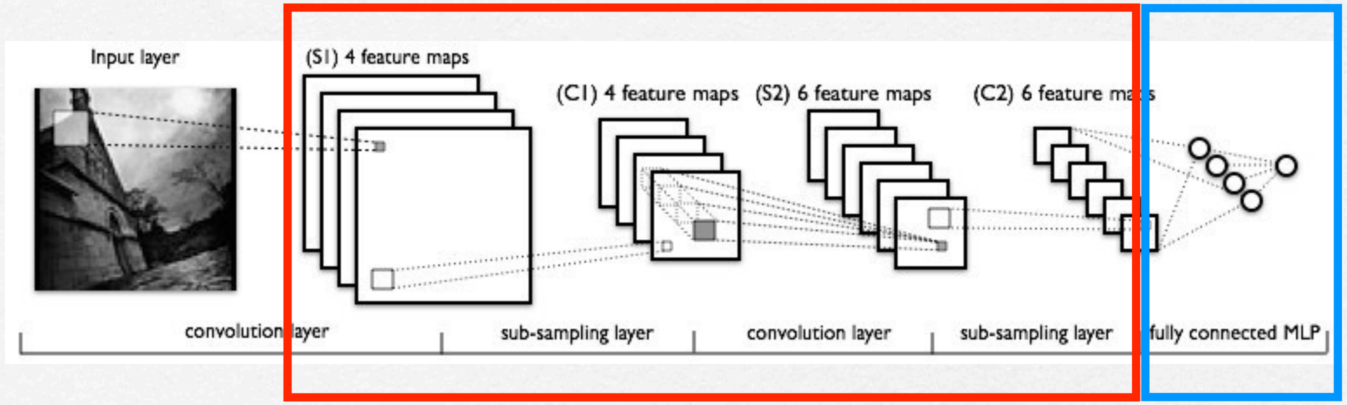
- We train the weights between the first two layers using a so-called auto-encoder architecture where we attempt to reproduce the inputs at the output.
- We fix those weights and repeat the process to train the weights of the next layer and so on…
- We finalize with a fine-tuning Backpropagation step

From Quoc V. Le, Google Brain, "A Tutorial on Deep Learning. Part 2."

CNN: from shallow to deep

flattened
input image



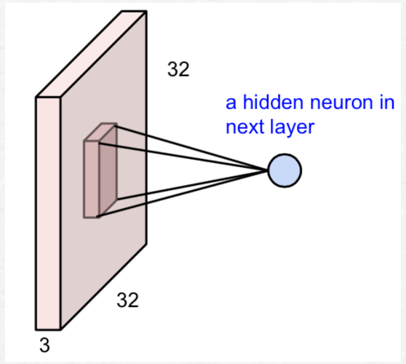


feature extraction classification

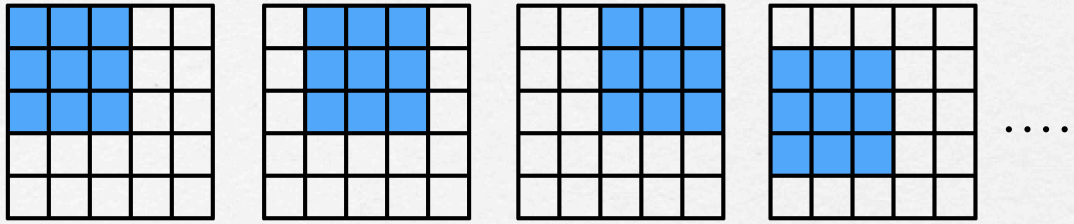
shallow feed-forward
neural network

deep convolutional
neural network

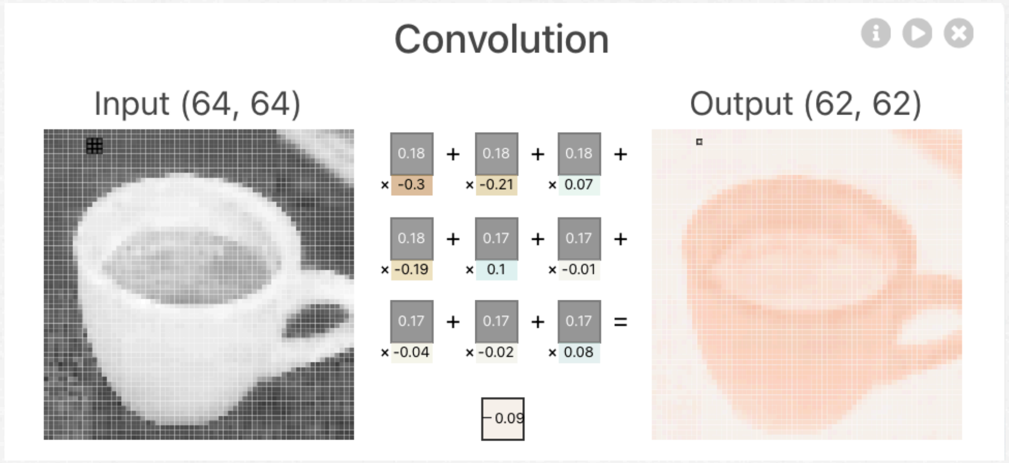
1 - Spatial processing (convolutions) & weight sharing



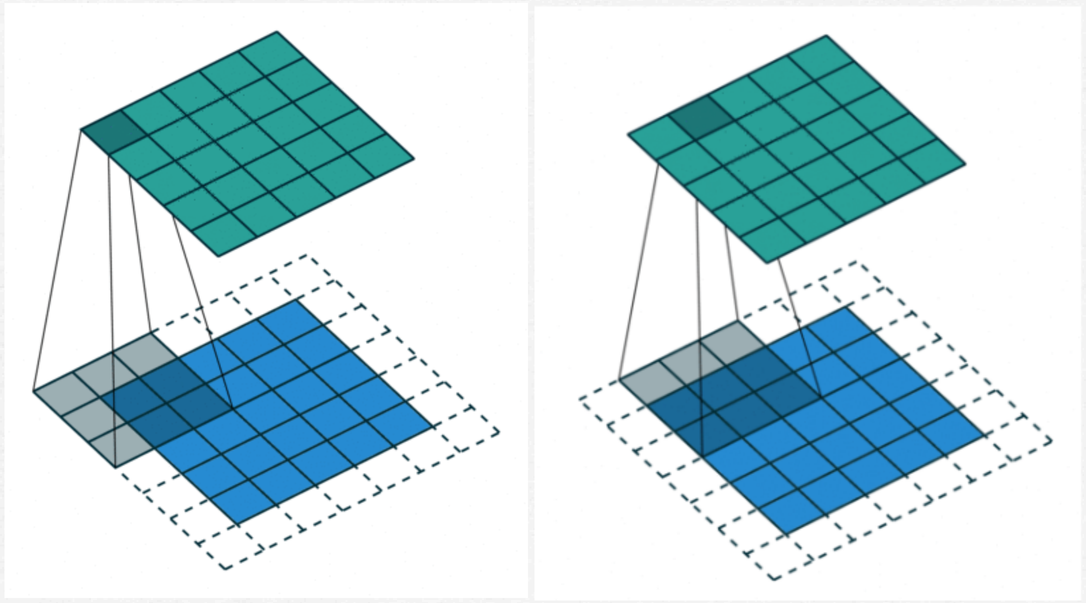
Each hidden neuron sequentially processes small regions of the input image until processing the complete input image using the same weights

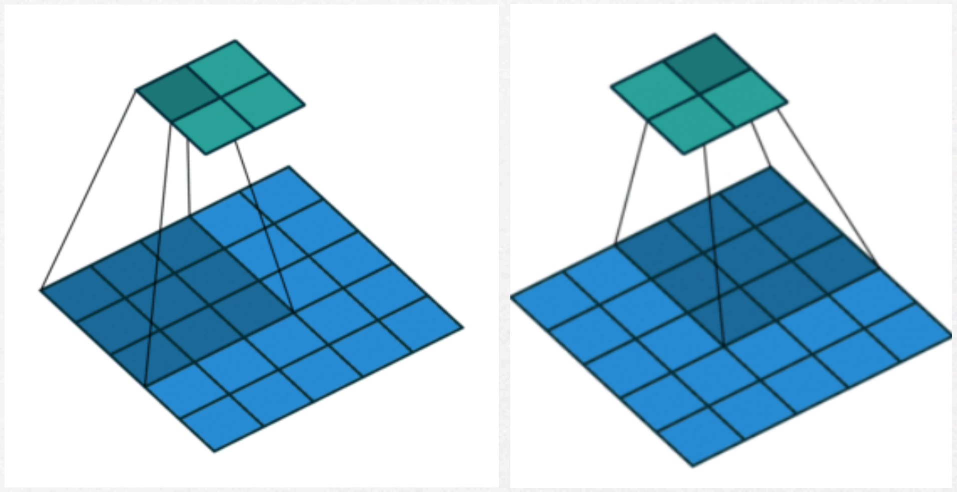


The output of each neuron is also an image:

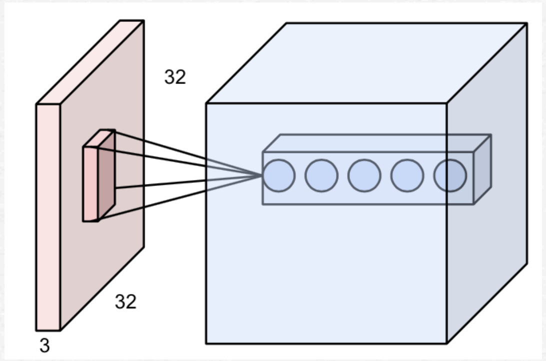


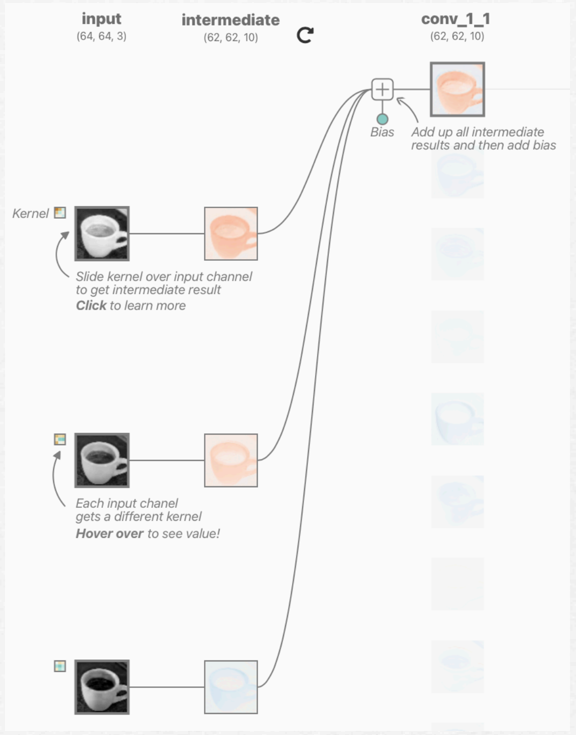
Stride and padding in a convolutional layer


A 3x3 convolution applied to an image using zero-padding and a stride of 1 (the default value). We conserve the size of the input image.


A 3x3 convolution applied to an image without padding and a stride of 2. We reduce the size of the image for the subsequent layers.

2 - Multiple convolutions with different kernels to detect multiple features



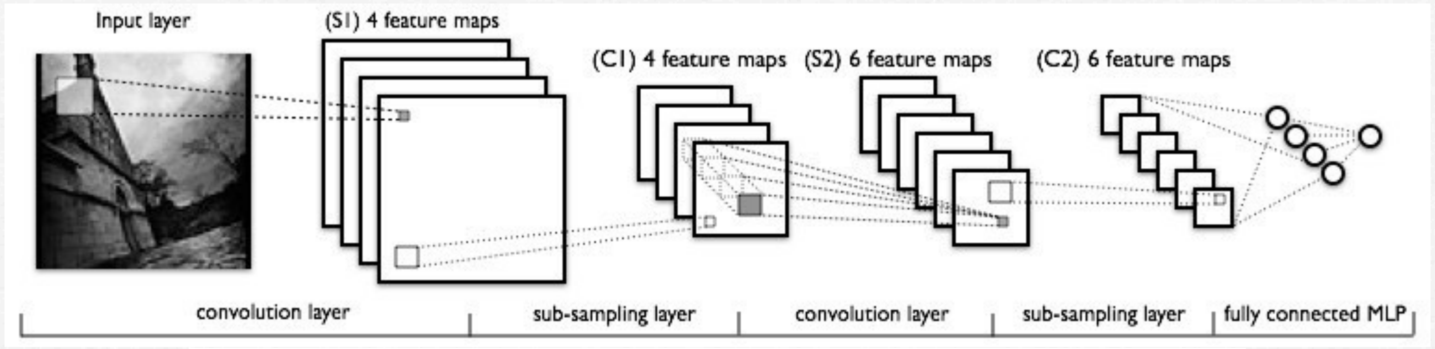

input
(64, 64, 3)
intermediate
(62, 62, 10)
conv_1_1
(62, 62, 10)
Bias
Add up all intermediate results and then add bias
Kernel
Slide kernel over input channel to get intermediate result
Click to learn more
Each input chanel gets a different kernel
Hover over to see value!

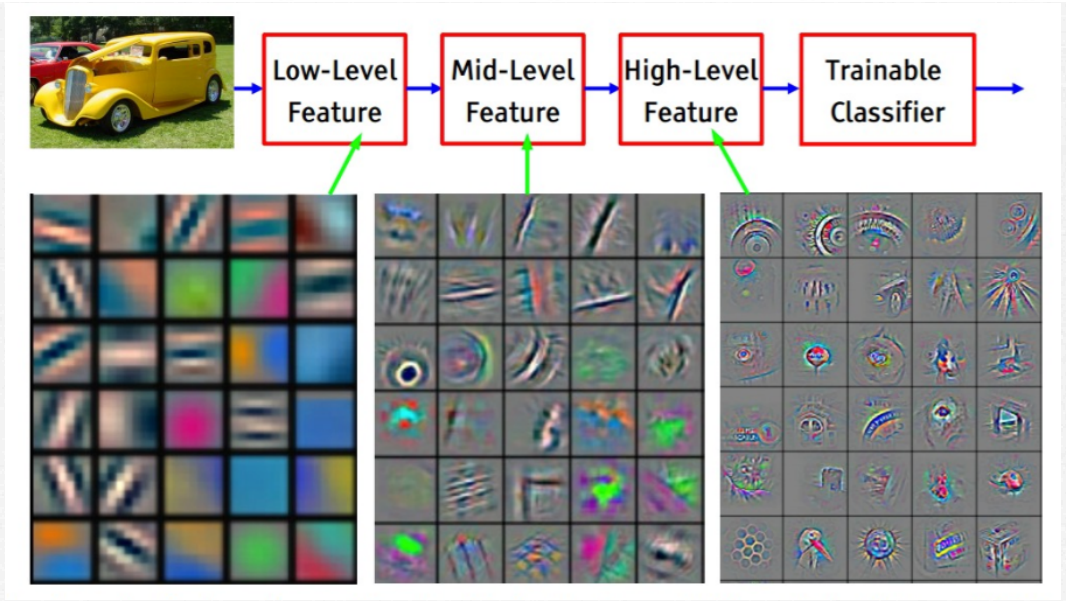
Each "neuron" applies a particular filter or kernel to the complete input image to detect a given feature. The outputs of each neuron is a new image.

When processing multidimensional images (e.g., RGB or hyperspectral), each neuron adds up the outputs of the filters applied on each channel.

3 - Hierarchical feature detection

LeNet5

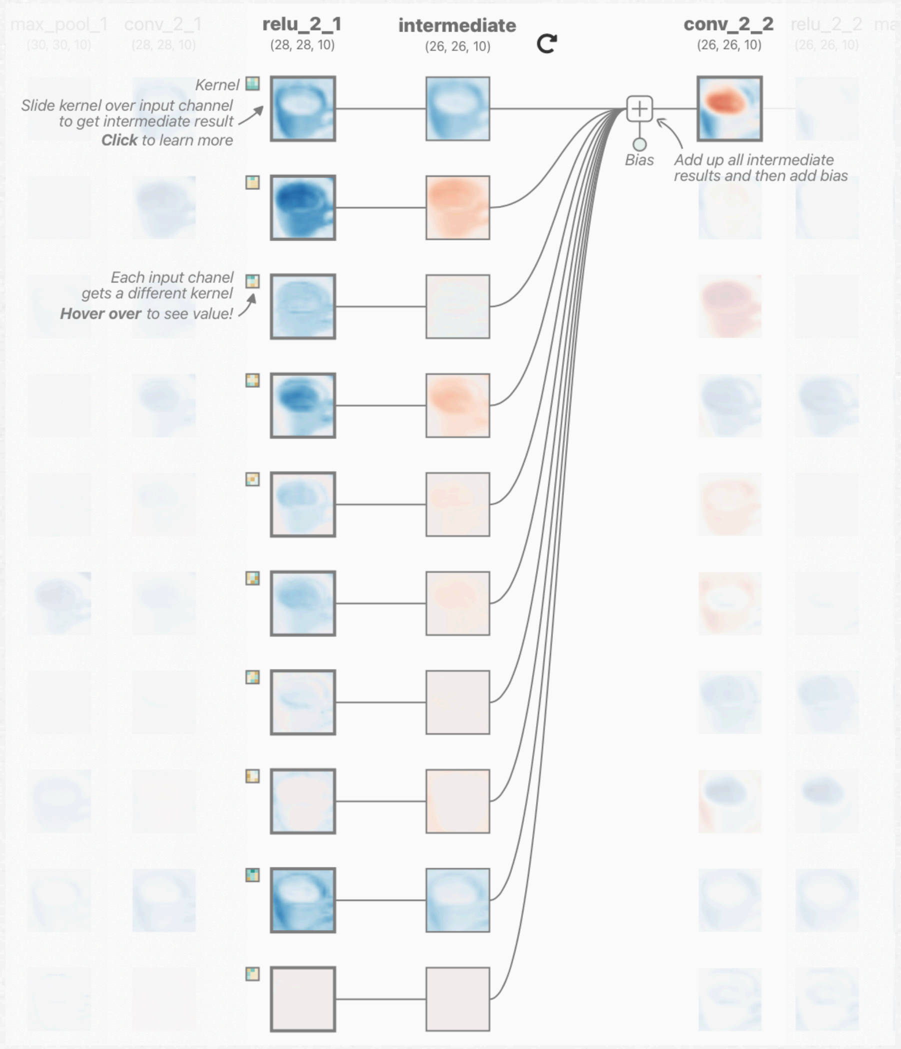




the first layer detects low-level features; the following detects combinations of low-level features, and so on...

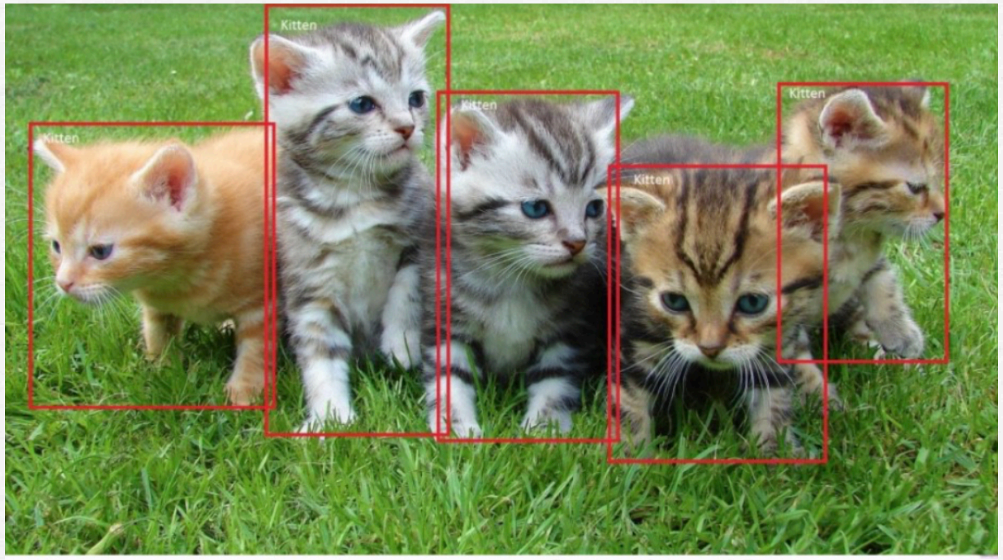
From Fei-Fei & Karpathy slides; deeplearning.com/tutorial

Convolutions in intermediate layers



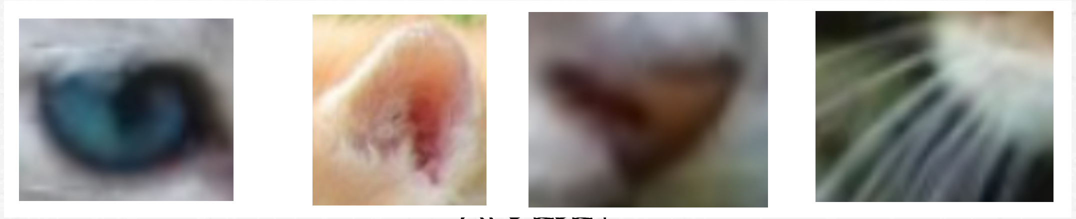
Suppose the previous layer is composed of 10 neurons (or filters). Each neuron of the subsequent layer applies a filter on each output image coming from the previous layer, it adds up the filter outputs and a bias to produce and new output image that will be passed on to the next layer.

Feature detection for object recognition

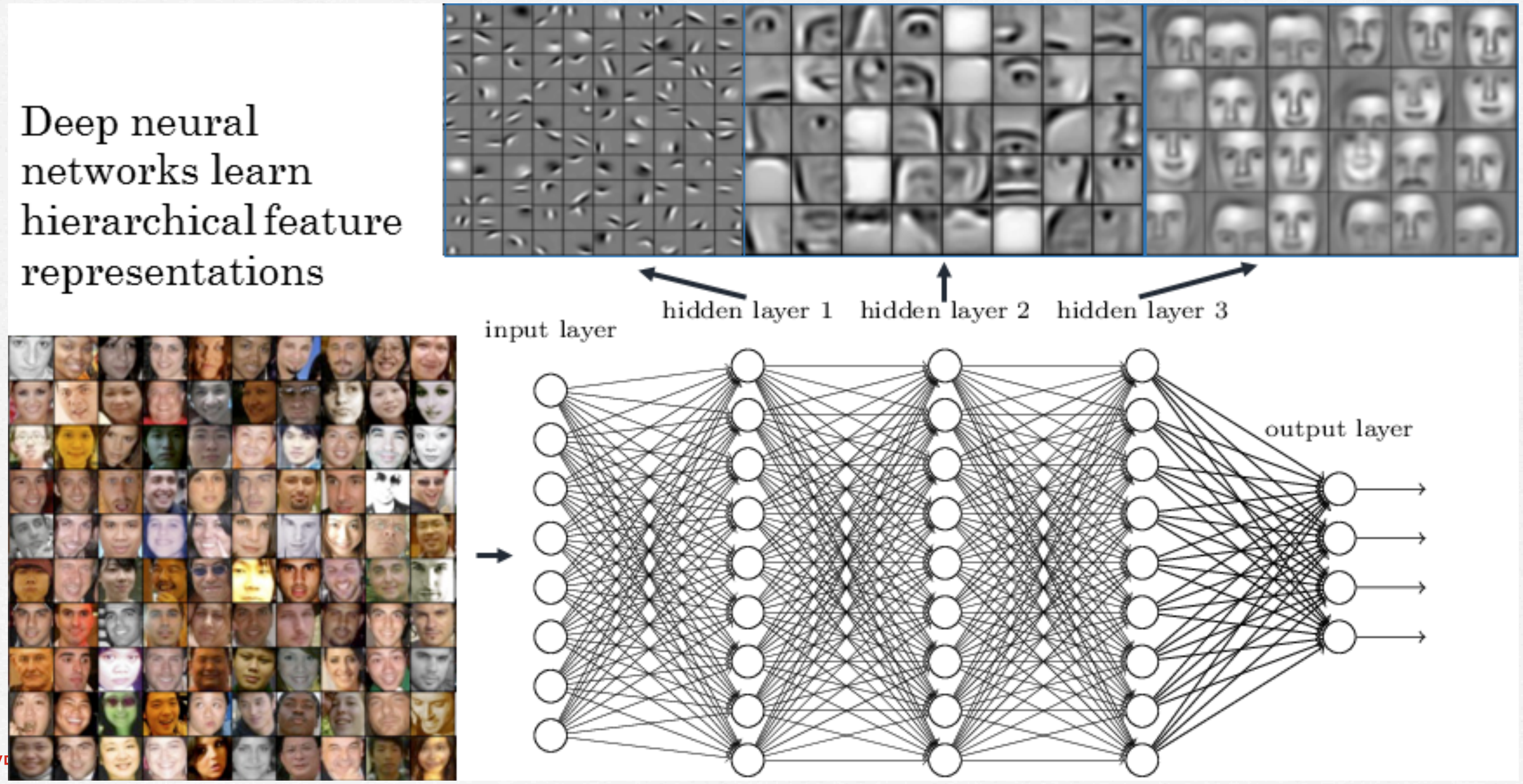


In an example of learning to recognize cats, the objective of the training of a CNN is to learn the filters that detect the features that allow the network to recognize a cat independently of its position in the image, its color, its orientation, its size, etc…

Examples of filters to be (ideally) learned are: eye, ear, nose and whiskers' detection filters:



Learning of hierarchical feature representations



4 - New activation functions

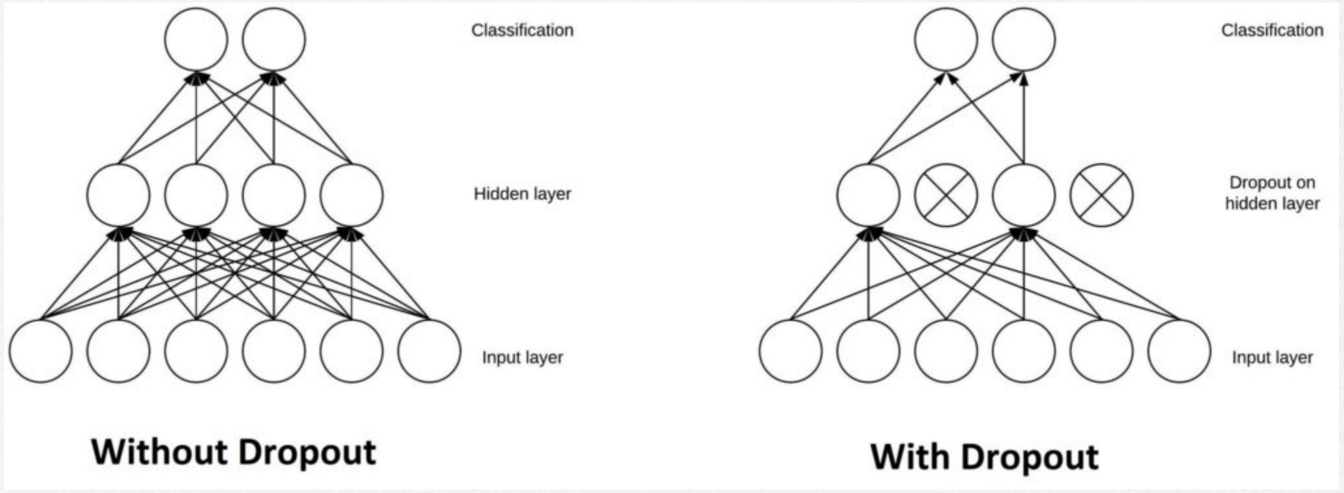
- We already mentioned that ReLU functions are preferred to avoid the vanishing of gradients.
- A softmax function is often used (in classification problems) in the output layer to compute P(target/inputs, weights, bias):

  $f(x_i) = e^{x_i} / \sum e^{x_j}$, $j = 1 \ldots K$

- The sum of $f(x_i)$ equals 1, thus the outputs represent a categorical probability distribution.

5 - Dropout to avoid overfitting

- Dropout (Hinton et al, 2012) is a technique for training neural networks by randomly dropping units during training to prevent « overfitting ».
- If we dropout each feature detector with a probability of 0.5:

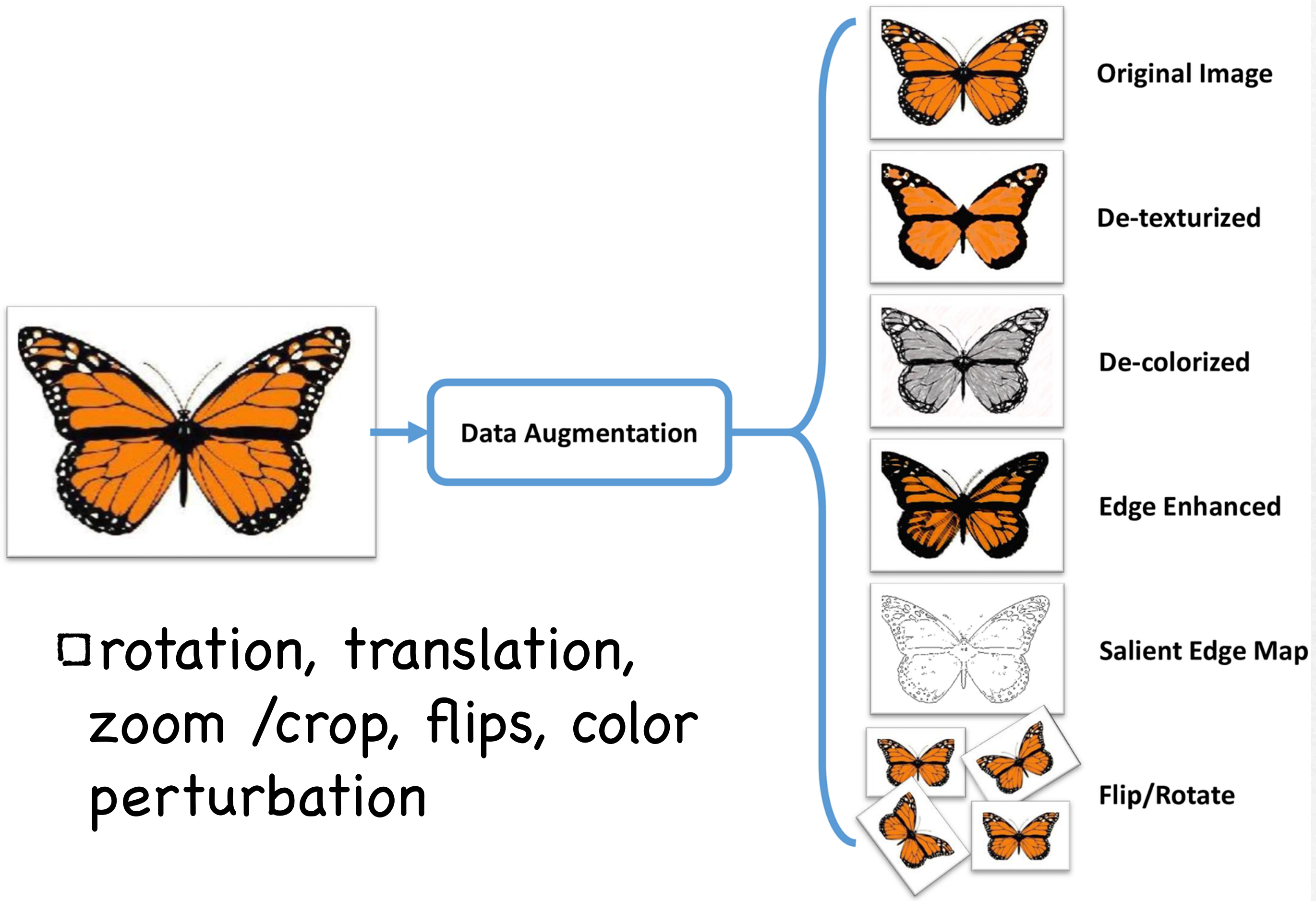


- When a neuron is dropped out, the other neurons have to learn more stuff in order to compensate the missing neuron.

Avoiding overfitting

- Steps for reducing overfitting:
  - Add more data
  - Use data augmentation
  - Use architectures that generalize well
  - Add regularization
  - Add batch regularization
  - Reduce architecture complexity

Data augmentation



- rotation, translation, zoom /crop, flips, color perturbation

from journal.pone.0183838

CNN: the problem of model selection

- Architecture :
  - How many convolutional layers ?
  - How many filters per layer ?
  - What filter sizes to use ?
  - Where to use a pooling layer ?
  - What activation function to use ?
  - What fully-connected layer configuration is required ?
- Learning algorithm:
  - Optimizer, loss function, learning rates, batch size, training epochs, regularization (L1, L2, dropout), batch normalization

CNNs making the headlines

- AI has made the front page of the NY times three times in recent times: 1) when Deep Blue beat Kasparov, 2) when Watson beat the best human Jeopardy players, and 3) when students using Deep Learning algorithms won the Kaggle's « Merck molecular activity challenge » in Nov. 2012
- Deep Learning in your browser:

  http://cs.stanford.edu/people/karpathy/convnetjs/index.html

Learnable feature hierarchies

- Image recognition:

  pixel -> edge -> texton -> motif -> part -> object

- Text processing:

  character -> word -> word group -> clause -> sentence -> story

- Speech:

  sample -> spectral band -> sound -> phoneme -> word -> …

"Anything humans can do in 0.1 sec, the right big 10-layer network can do too » in Large Scale Deep Learning by Jeff Dean (Google)

From LeCun & Ranzato