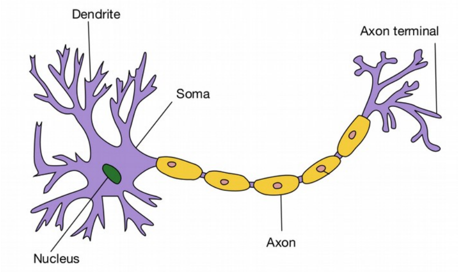
3. Deep Learning

Laerte Sodré Jr.
IAG – Universidade de São Paulo

IX La Plata International School (LAPIS) on Astronomy and Geophysics
S-PLUS: The Universe in True Colors
17-21 February 2020, La Plata, Argentina

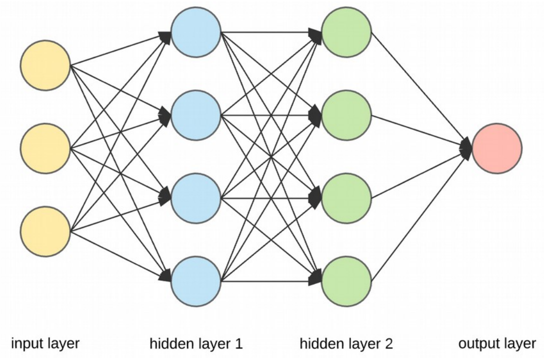
what are artificial neural networks (ANN)?

- type of information processing loosely inspired by the human brain
- structure- large number of connected processing units: artificial neurons
- an ANN learns from the data: the “inteligence” of the net is in the weights between connections



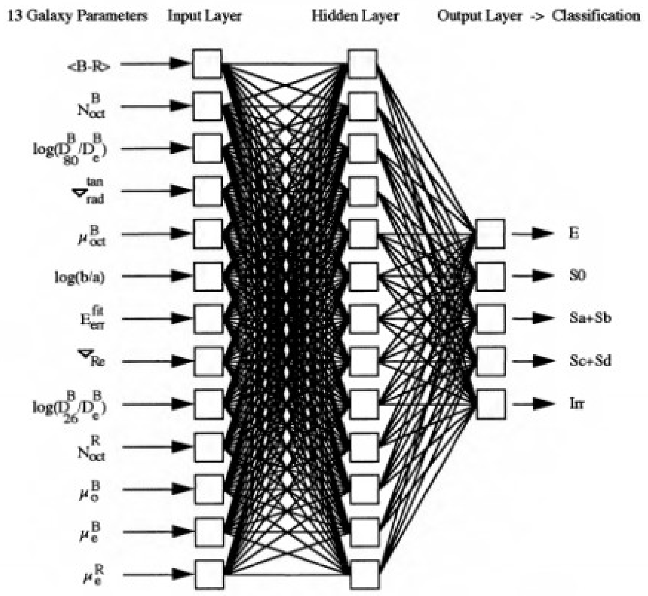
advantages:

- non-linearity: able to model complex data
- fault tolerant (robust), due to the distributed nature of the information
- massively parallel processing



what are artificial neural networks (ANN)?

- a NN learns a function: y=f(x)



Storrie-Lombardi et al. (1993)

architecture types:

- single layer: shallow net
- multiple layers: deep nets
- feed-forward
- recurrent
- convolutional

learning:

- supervised (perceptron)
- unsupervised (Kohonen)
- reinforcement (self-driven cars)

units:

- Sigmoid
- ReLU
- linear

activation units

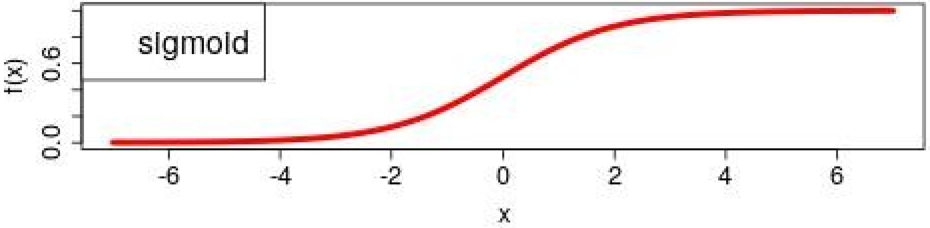
- **activation function: computes the output of a unit from its inputs**

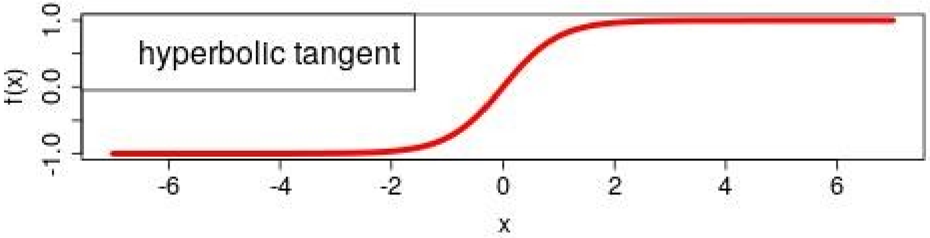
- **sigmoid**
  f(x) = 1/[1+ exp(-x)]

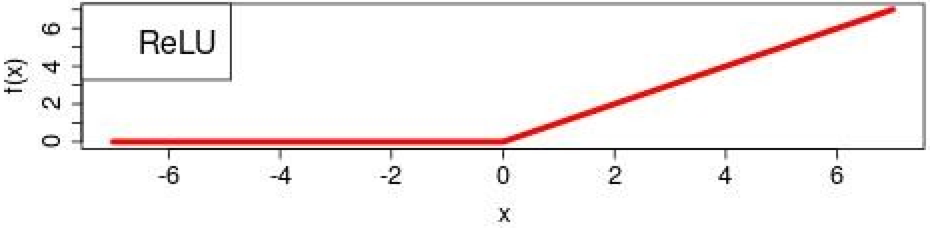
- **hyperbolic tangent**
  f(x) = [exp(x)-exp(-x)]/[exp(x)+exp(-x)]

- **ReLU (Rectified Linear Unit)**
  f(x) = max(0,x)

- **linear:**
  f(x)=a+bx







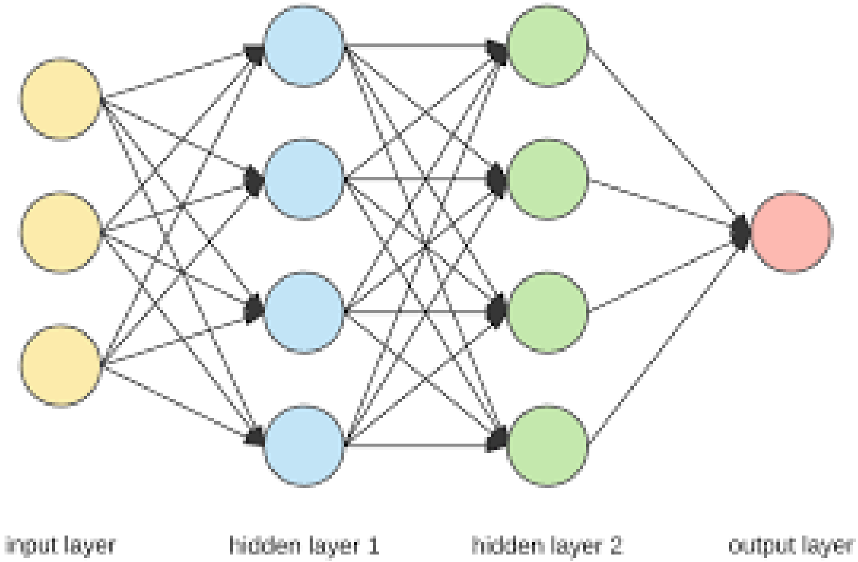
the multilayer perceptron

architecture:

- **input layer**
- **one or more hidden layers**
- **output layer**

- **one layer is *fully connected* to the next**

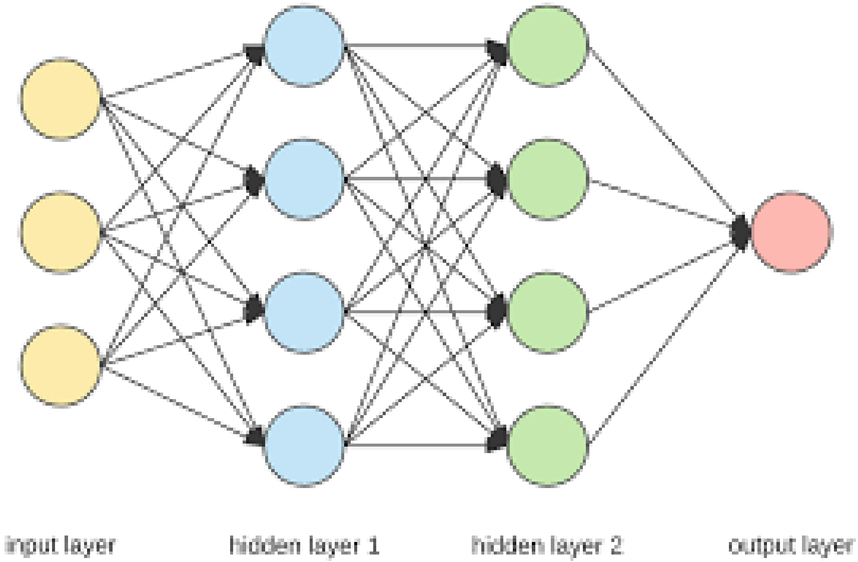
- **inference: forward pass**
  the net computes the output of each neuron



the multilayer perceptron

the universality theorem

- any continuous real function can be realized with a neural network with a single layer of sufficient capacity



deep learning

- deep: many hidden layers
- in general is easier to learn a function with many hidden layers

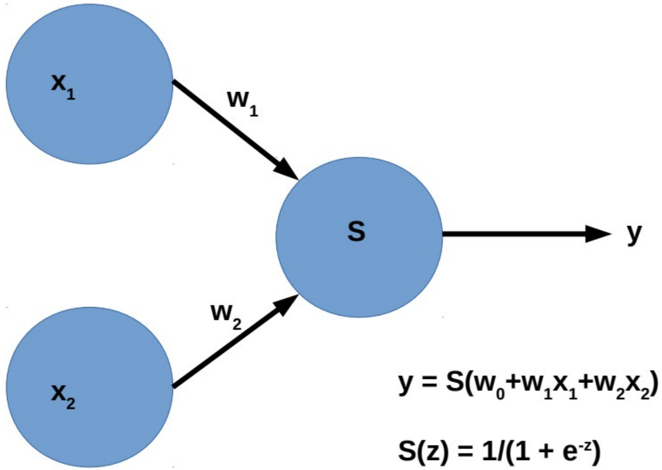
learning: back-propagation

type of gradient descent:

- update the weights starting with the last layer
- propagates the error to the previous layer
- update the weights of this layer and repeat the procedure up to the input layer

example: logistic regression

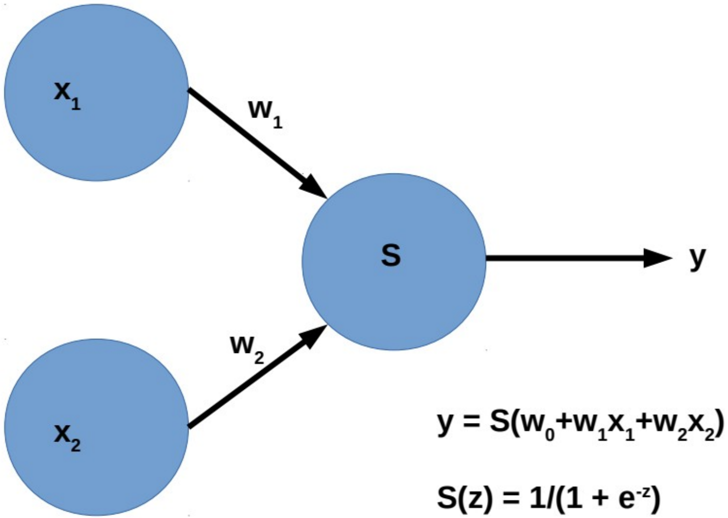
- x: input
- the net is trained to estimate targets t
- y: prob(y=1|x)
- activation: sigmoid



$y = S(w_0+w_1x_1+w_2x_2)$

$S(z) = 1/(1 + e^{-z})$

learning: back-propagation



$y = S(w_0+w_1x_1+w_2x_2)$

$S(z) = 1/(1 + e^{-z})$

Example: logistic regression

sigmoid: $S(x) = 1/(1+e^{-x})$

derivative: $S' = S(x)[1-S(x)]$

output: $y = S(z)$
$z = w_0+w_1x_1+w_2x_2$

cost function: $E = ½ (t-y)^2$

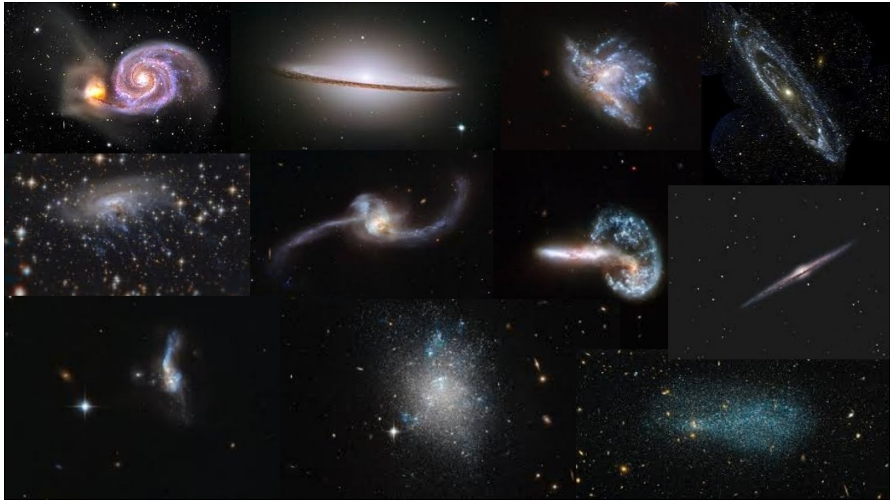
chain rule: $dE/dw_k = dE/dy\ dy/dS\ dS/dw_k$
$dE/dw_k = [-(t-y)]\ [y(1-y)]\ x_k$

weight update: $\Delta w_k = \alpha(t-y)y(1-y)\ x_k$

convolutional neural networks

Why, compared to a human, is difficult for an algorithm to identify images?

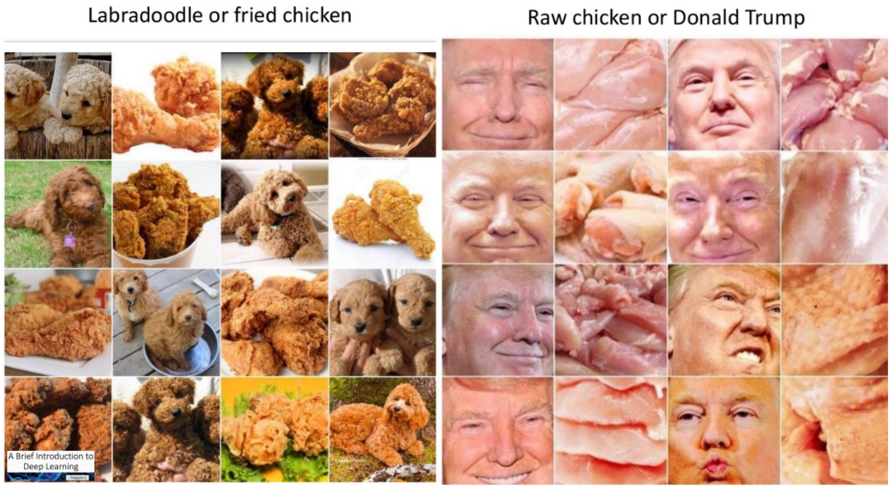
- large variation of images of the same type of object
- segmentation: which pixels are of a certain object?
- invariances: easy for us to recognize them
- “deformations”: galaxy morphology, calligraphy



convolutional neural networks

Why, compared to a human, is difficult for an algorithm to identify images?

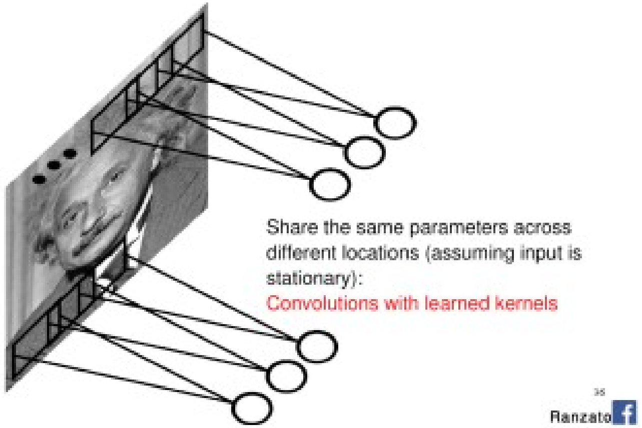
- large variation of images of the same type of object
- segmentation: which pixels are of a certain object?
- invariances: easy for us to recognize them
- “deformations”: galaxy morphology, calligraphy



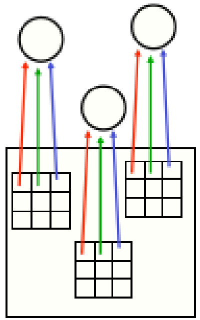
convolutional neural networks

LeCun, 1998

- locally connected layers
- multiple copies of ‘detectors’ or ‘filters’ at different positions
- convolutional layers: each hidden unity connects to a small region of the image
- each layer contains multiple filters



The red connections all have the same weight.



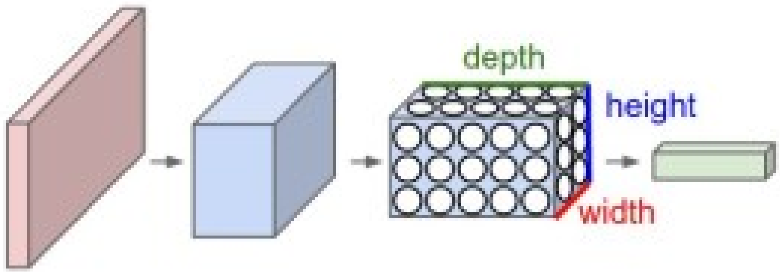
convolutional neural networks

hyperparameters:

- number of filters: depth of output volume
- stride: separation between the filters (controls the size of the output volume)
- filter sizes: *w x h*

pooling:

- each convolutional layer is followed by a *pooling layer*
- they extract the maximum (or mean) value of a set of filters



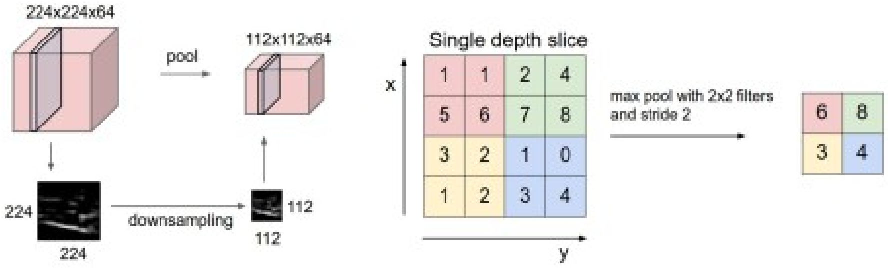
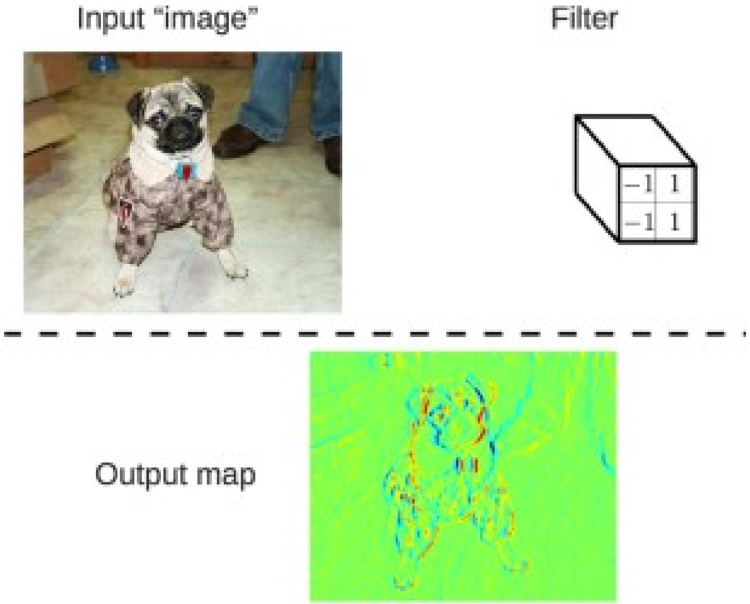


Figure : **Left:** Pooling, **right:** max pooling example

filters

if the filter is [-1,1], you get a vertical edge detector:



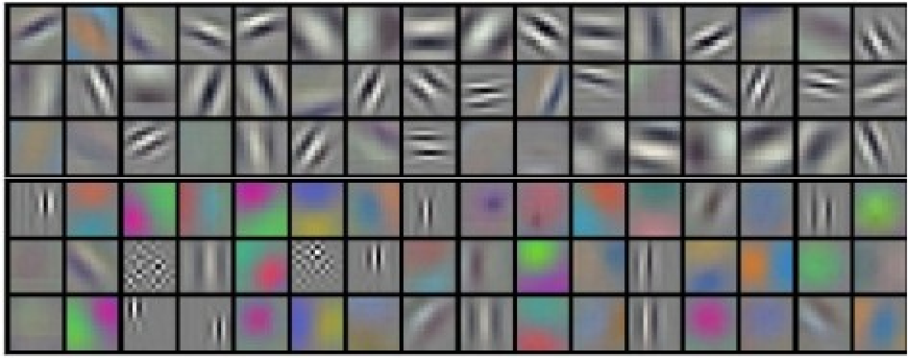
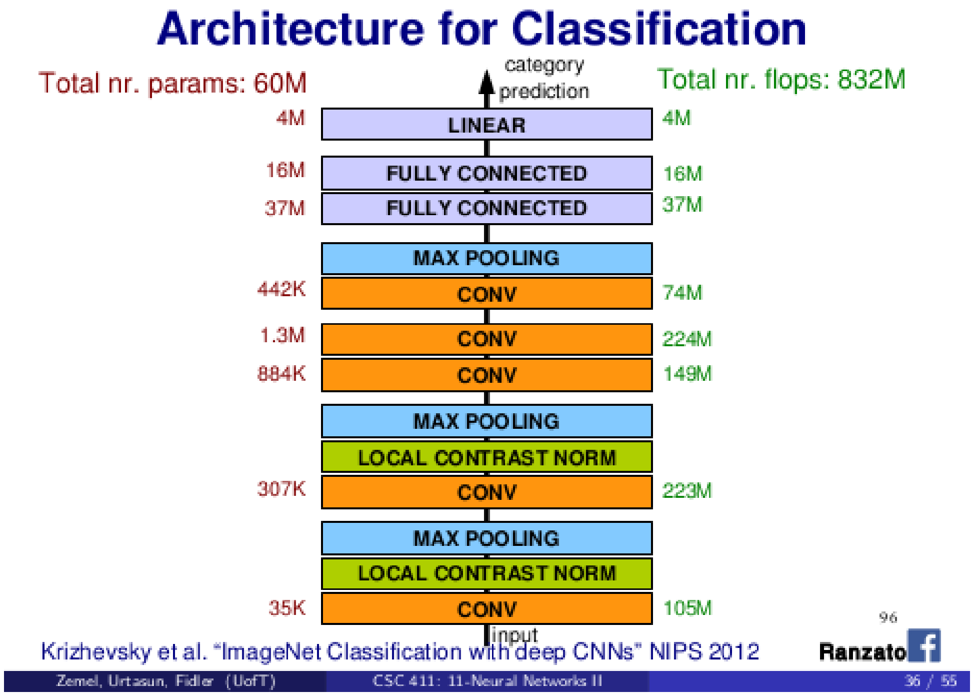


Figure : Filters in the first convolutional layer of Krizhevsky et al

convolutional neural networks

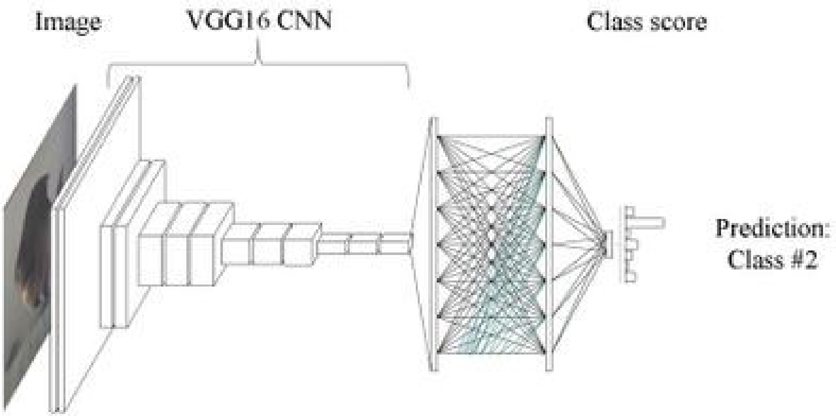
- convolutional layers are followed by a pooling layer which uses as input the output of the previous layer
- this allows the net to learn multiple filters
- end the net with one or two fully connected layers for classification or regression
- training: variant of backpropagation



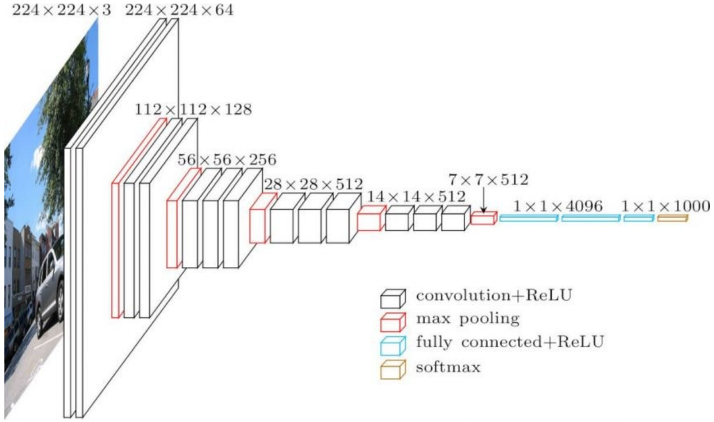
convolutional neural networks with pre-trained nets

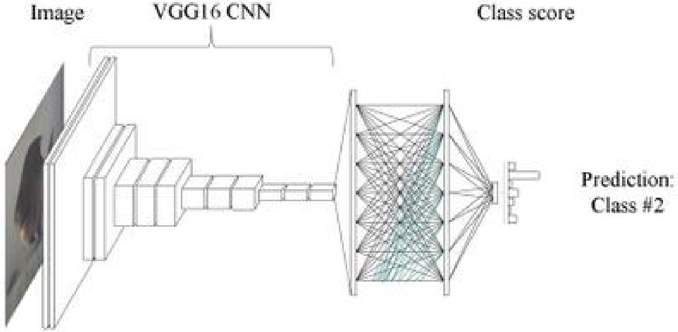
- one can train a net using a CNN previously trained in a large set of images
- example: ImageNet- database with ~14 million images classified in 1000 different classes
- VGG16: proposed by Simonyan & Zisserman and winner of the 2014 ILSVR competition
- one can use the convolutional part of a pre-trained net to feed a dense network for classification or regression

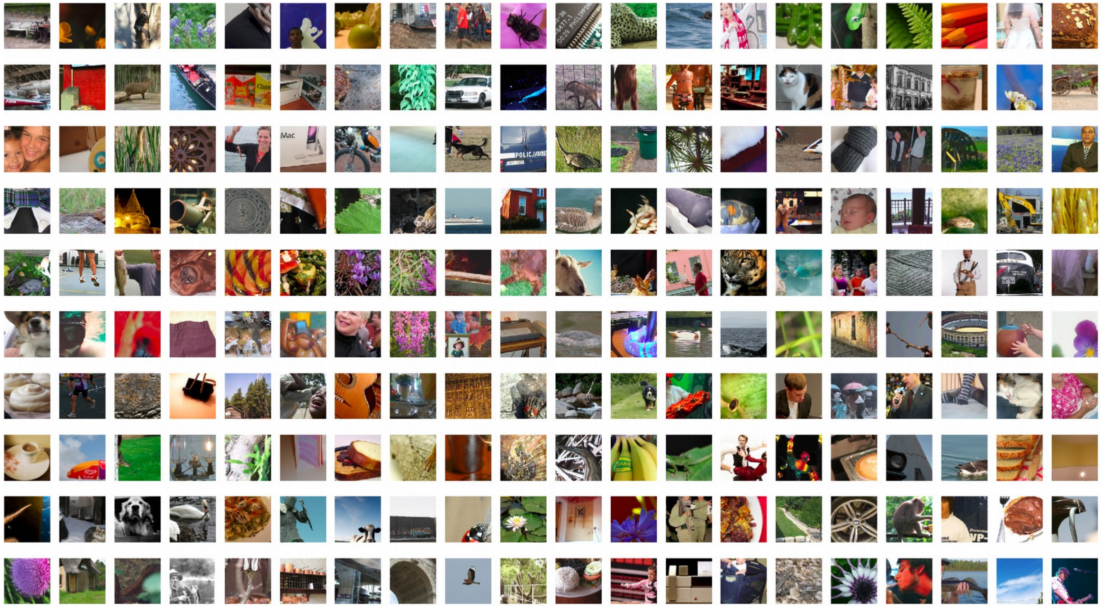
- basic idea: the filters learnt by the net may be useful for many image analysis
- after the convolutional/pooling layers we include and train a couple of fully connected layers

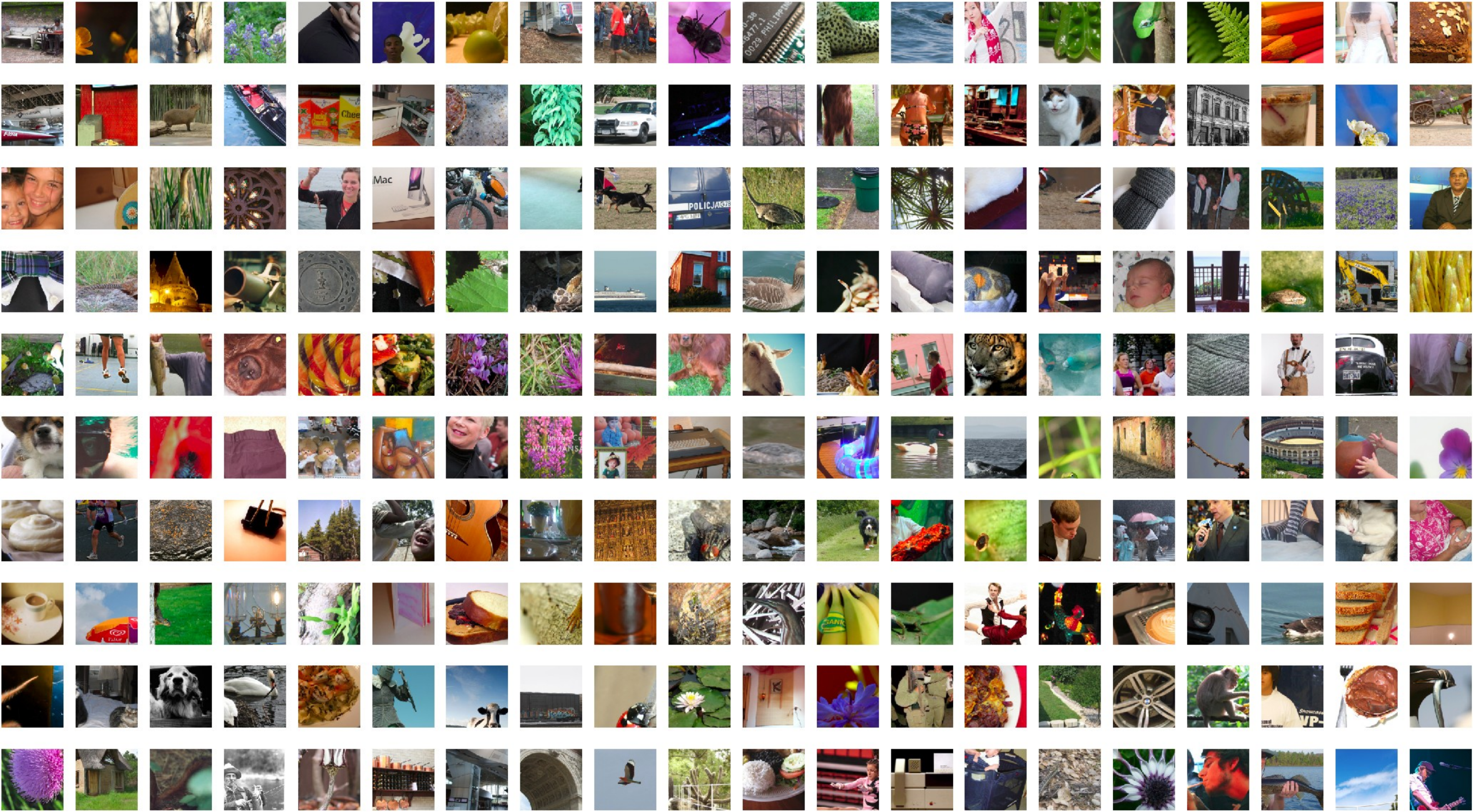


convolutional neural networks with pre-trained nets



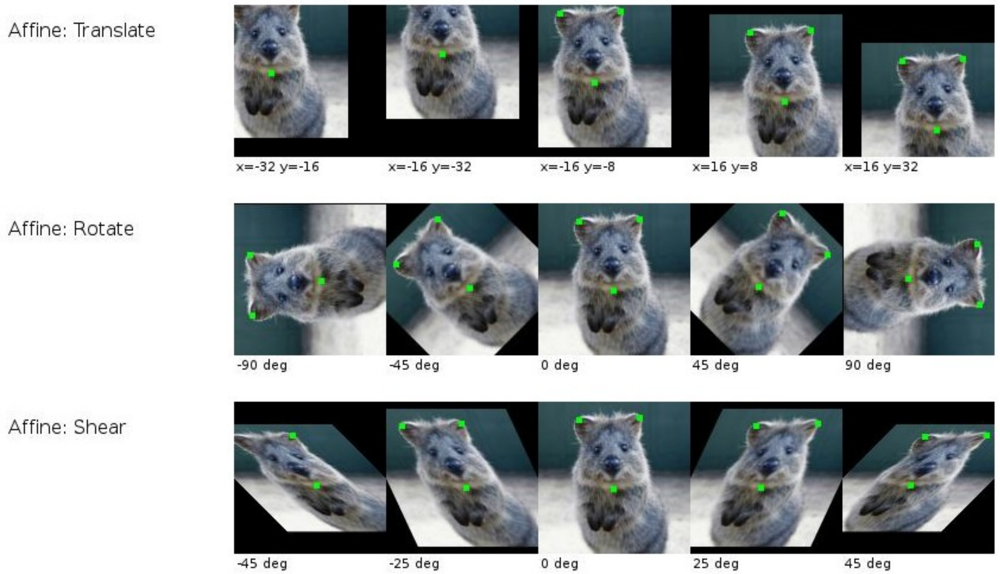





iMac
POLICJA

overfitting

- CNNs are prone to overfitting due to the large number of parameters
- two strategies to deal with overfitting: *data augmentation* and *dropout*



- data augmentation:
  - create new images through transformations from the available images during training
  - transformations: reflexion, translation, shear, etc...
- dropout
  - we remove (put equal to zero) randomly a certain number of outputs of a layer during training
  - we add a dropout layer before the dense layers

regression and classification with deep learning

output activations:

- regression: linear activation

  (or sigmoid if output in [0,1])

- classification:
  - binary: sigmoid
  - Multiclass / multiple outputs: softmax

    $y_k = e^{zk} / \Sigma_j e^{zj}$
  - In multiclass classification, for the target output one uses *1-K encoder* or *one-hot vector*:

    t = [0, 0, …, 1, 0,…,0]

Cost/loss functions:

- regression: square deviation

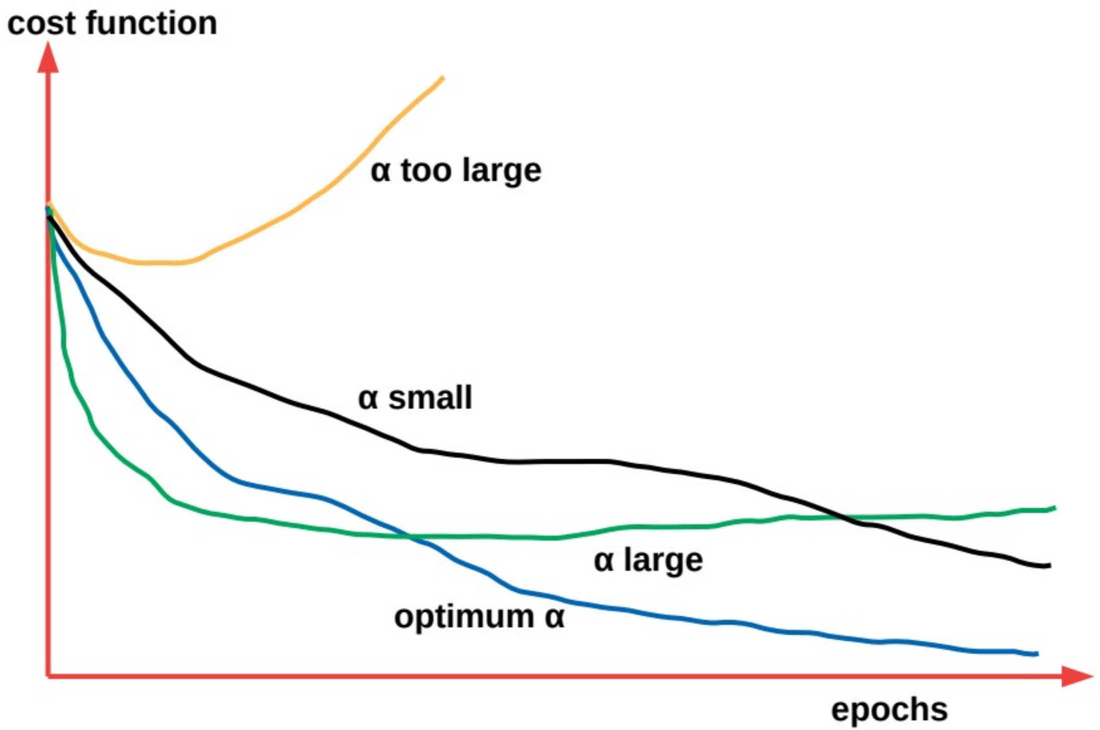
  $l(w) = \Sigma_i(t_i-y_i)^2$

- classification: cross-entropy

  $l(w) = -t_i \log(y_i)$

training

attention to the many model hyperparameters!



training

monitore the training to avoid overfitting!