IntroML
ML. 7.3. Convolutional neural nets

DataLab CSIC

Objectives and schedule

Introduce key concepts about convolutional neural networks (convolutional networks, CNNs, Convnets)

CASI 18, Goodfellow et al 9, Chollet and Allaire 5

Case by Nacho Villanueva (ICMAT, UCM) on forecasting eolic energy production

http://srdas.github.io/DLBook/ConvNets.html

https://www.youtube.com/watch?v=BFdMrDOx_CM

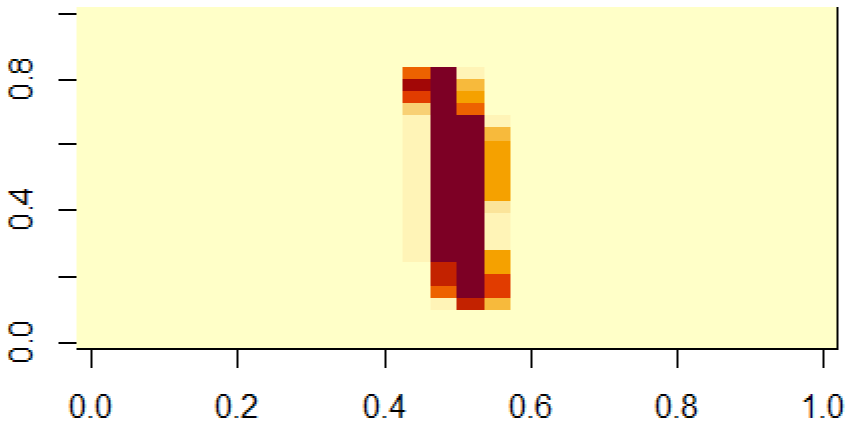
Labs

- MNIST
  - logreg (with Keras notation as nn), improved by rescaling
  - fully connected network
    - With L2 regularization
    - With dropout
  - Model maintenance (store and reload the model for predictions)
  - Random forest
  - CNN (with 2 epochs, version for Google Collab)

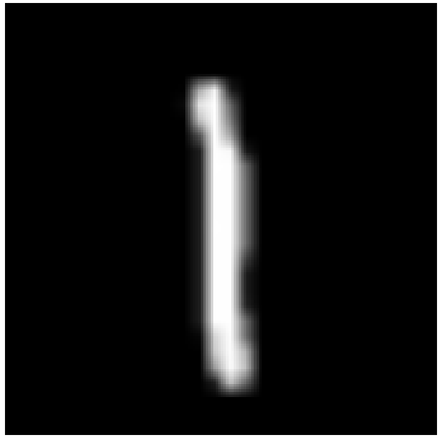
We start to need HPC

- Drago, Artemisa @CSIC
- Lovelace @ICMAT

MNIST data looks like this
28x28 matrix with entries from 0 to 255



When rasterised



Conv NNs. Motivation

Motivation

- Fully connected NNs can approximate any function….
- But training can be super slow and may require lots of data
- In some domains, lots to be gained through specific architectures
- E.g., in vision, convolutional neural nets
    - 1-d conv-nets for time series (but better competitors)
    - ***2-d conv-nets for image recognition et al (eg ADS)***
    - 3-d conv-nets for video et al

History

- Hubel and Wiesel (60's) visual perception of cats
- Fukushima (late 70's) introduces neocognitron (without training algo)
- Le Cun et al (90's) 'neocognitron+backprop' leads to convnets to handle MNIST dataset. Le Net 5
- Alexnet (2012) (1.3 M images to recognise 1000 objects)
- VGGNet (2014), Googlenet (2015) win the Imagenet competition leading to explosion of interest

Towards convnets

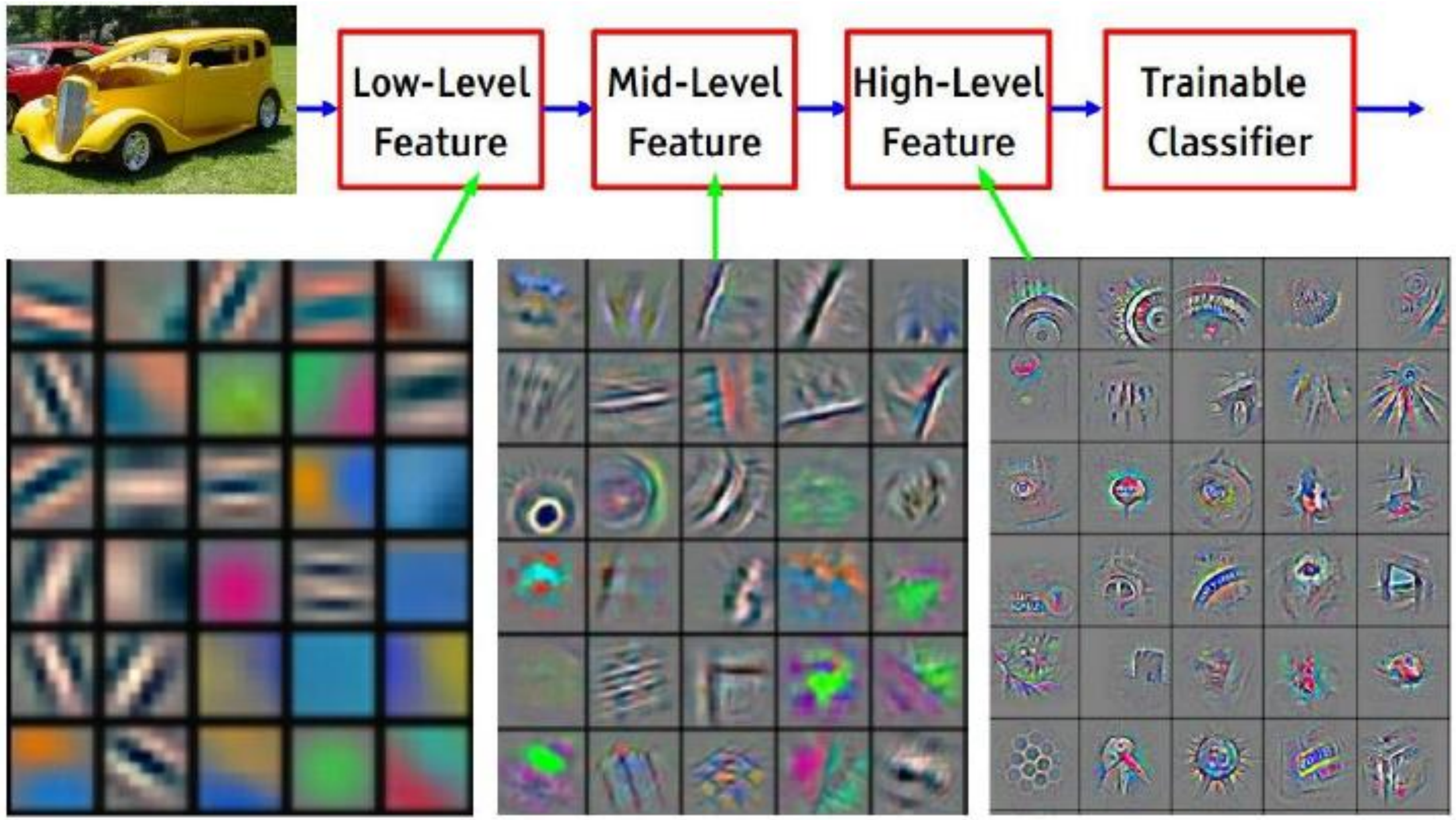
Before

- Objects to be classified ----> Extract features ‘manually’ ----> Trainable classifier

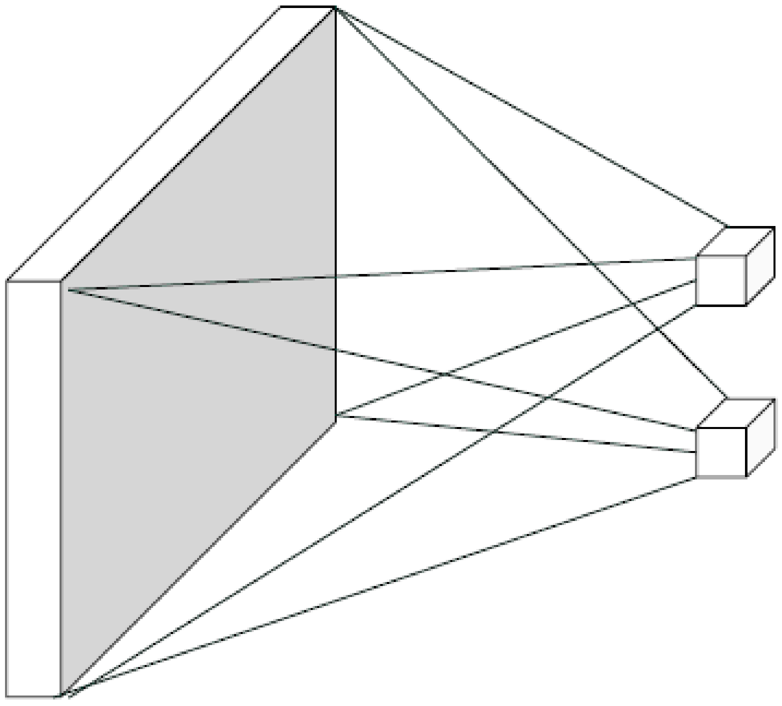
Today

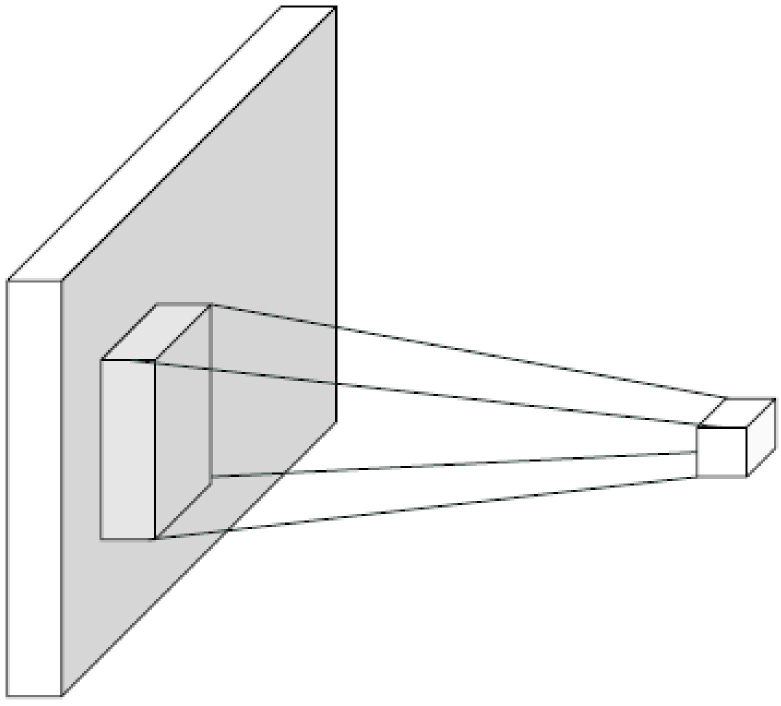
- Objects to be classified ----> Trainable feature extractor ----> Trainable classifier (even integrated)

CNNs. Typical hierarchical extraction of features



From fully connected DNs to convolutional (deep) nets



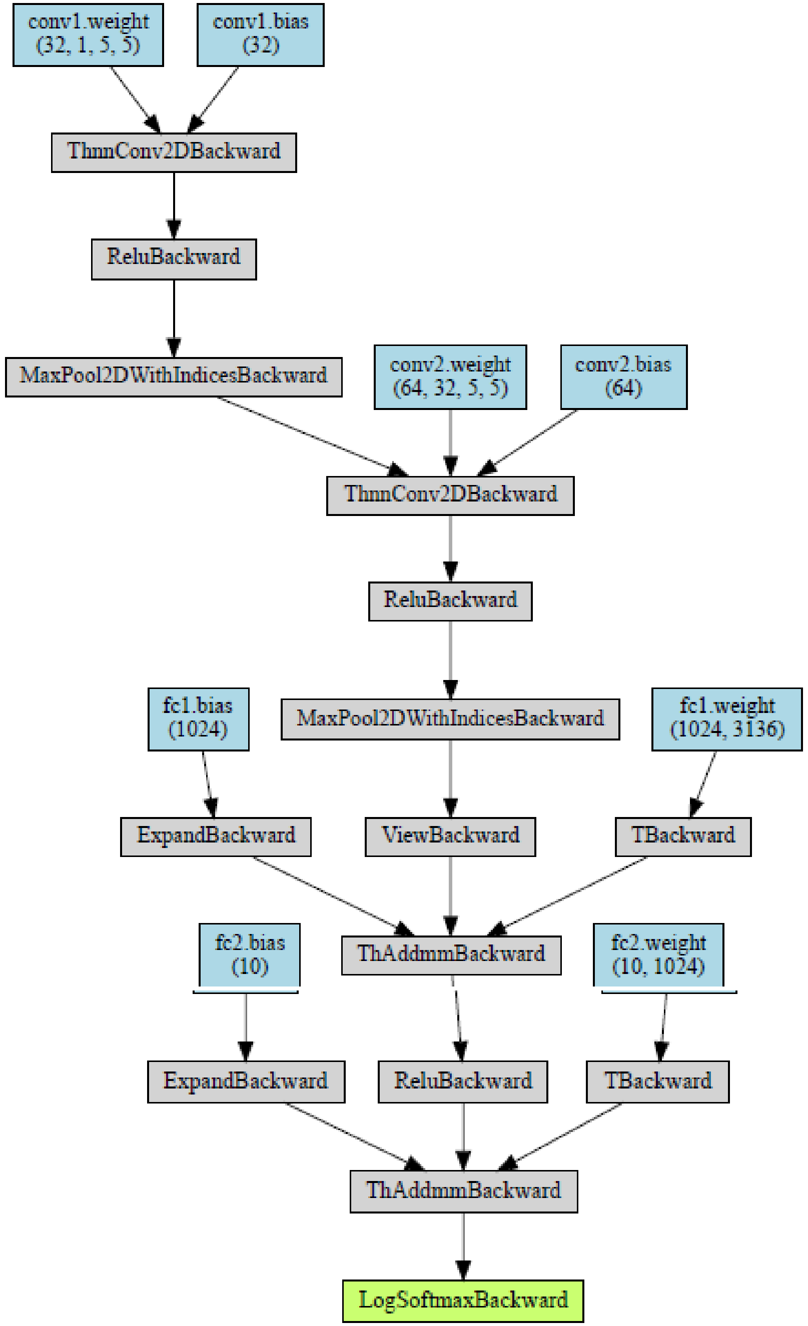


Towards convolutional networks

- Three core concepts
    - Convolution
    - RELU
    - MaxPool

- Applications
    - Image recognition (e.g., security)
    - Video analysis (e.g., ADS)
    - Sound analysis

    - Pharma discovery
    - ….. And others



Core concepts

Convolution

CNNs: Neural nets which include convolution in at least one of the layers

Convolution. Linear operator. For measurement x(t), weighted w(a) (a, past time)

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

x, input; w, kernel

Convolution

In DL: Input. Multidimensional matrix (tensor) (eg, piece of an image)

Kernel. Multidimensional matrix (tensor)

Convolution in more than one dimension (eg I bidimensional image, K bidimensional kernel)

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

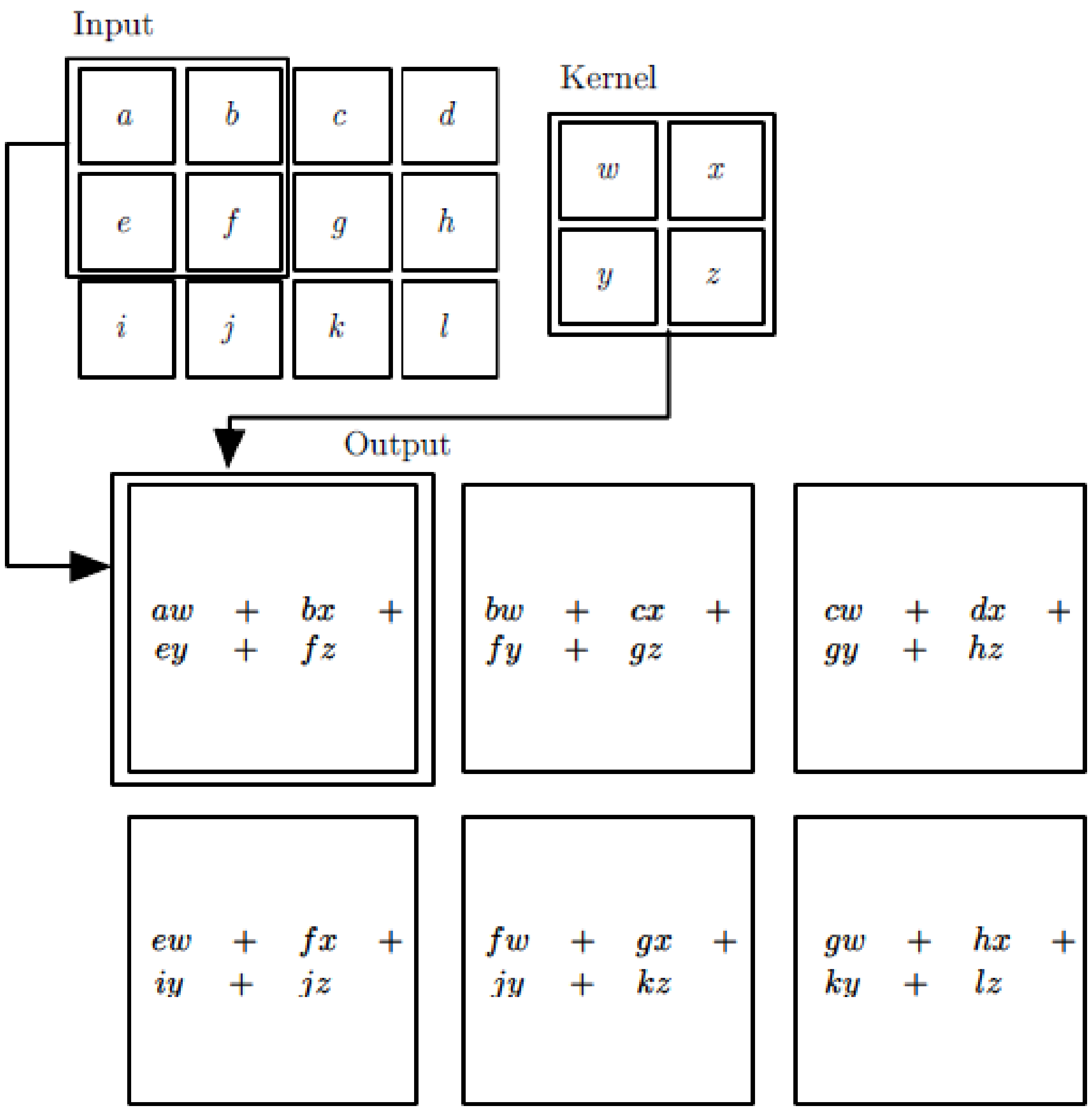
Multiplication by a circulating matrix, frequently sparse (many zeros)

Convolution

Multiplication by a circulating matrix, frequently sparse (many zeros)

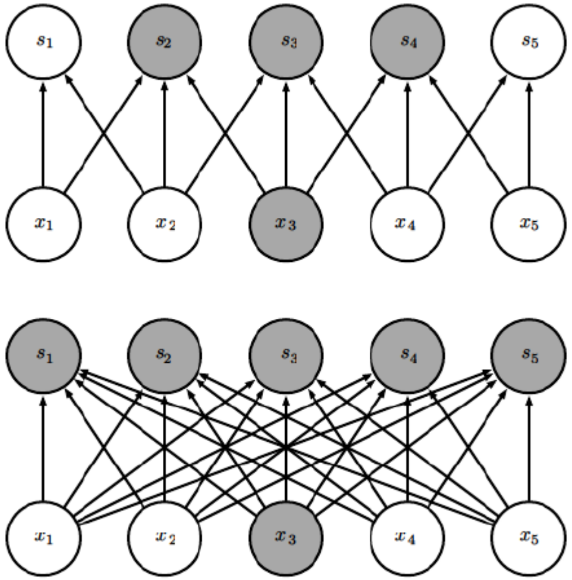
The kernel weights (circulating matrix) w,x,y,z are to be learnt

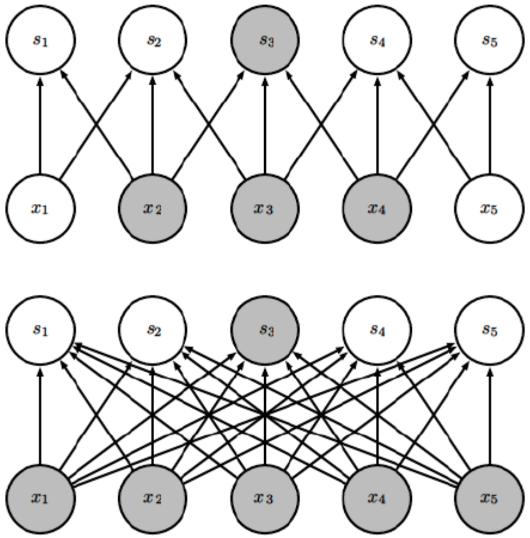
Several matrices are to be learnt at each convolutional layer

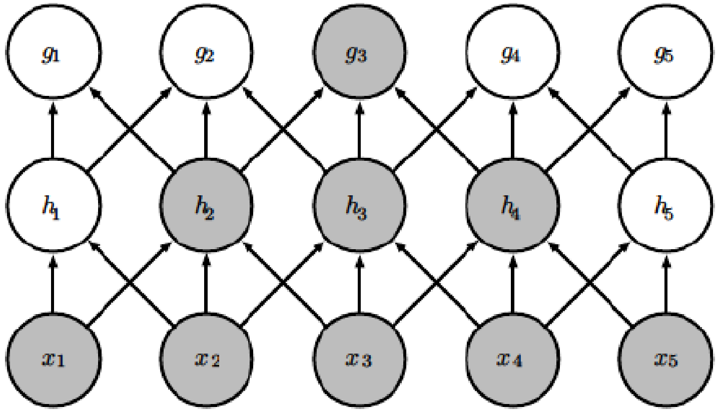


Convolution

- Property 1. Sparse (less dense) interactions (less storage, fewer operations)

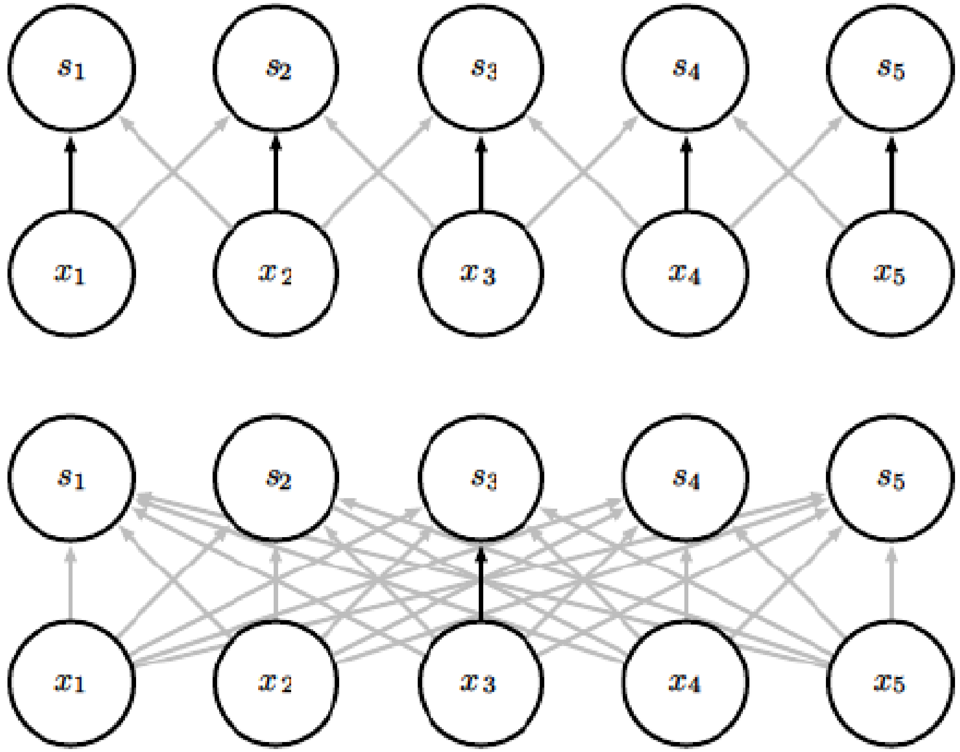






Convolution

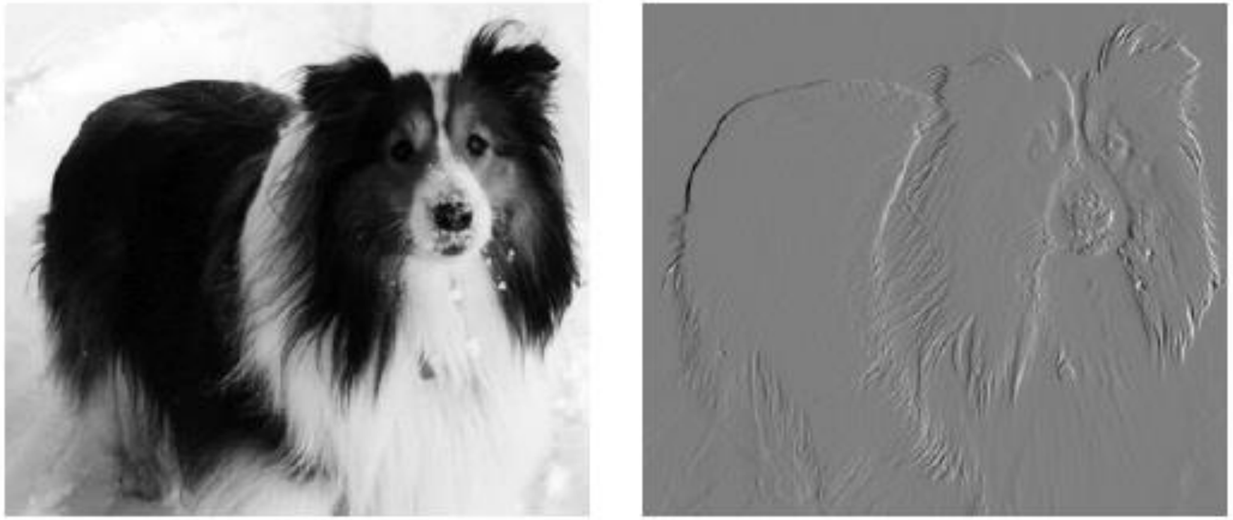
- Property 2. Parameter sharing

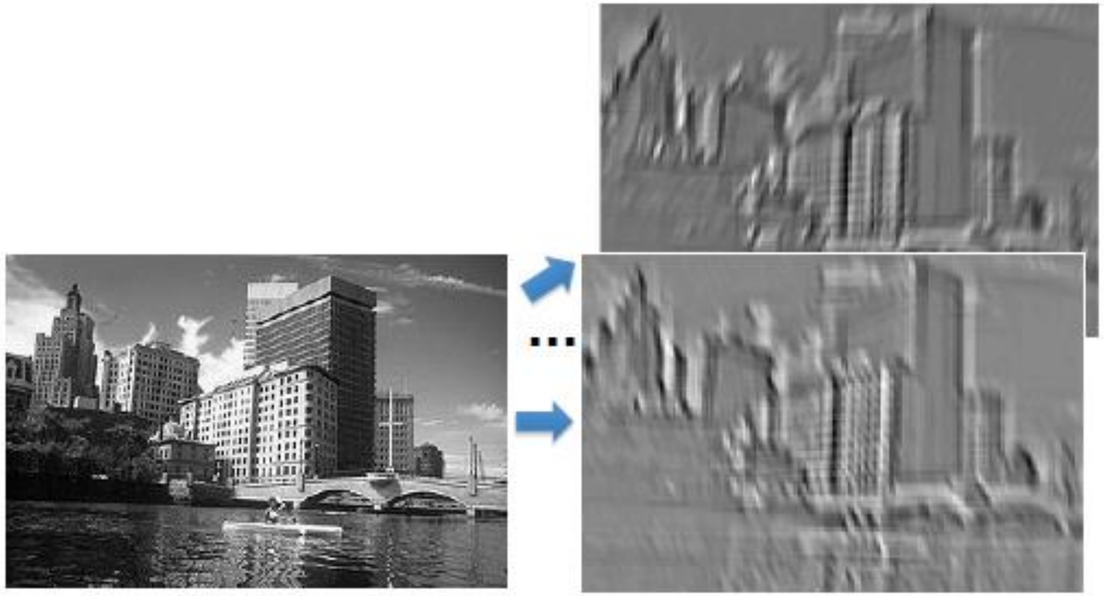


Convolution

Low density+sharing -> Edge detection, feature extraction

e.g. Remove to each pixel that on its left

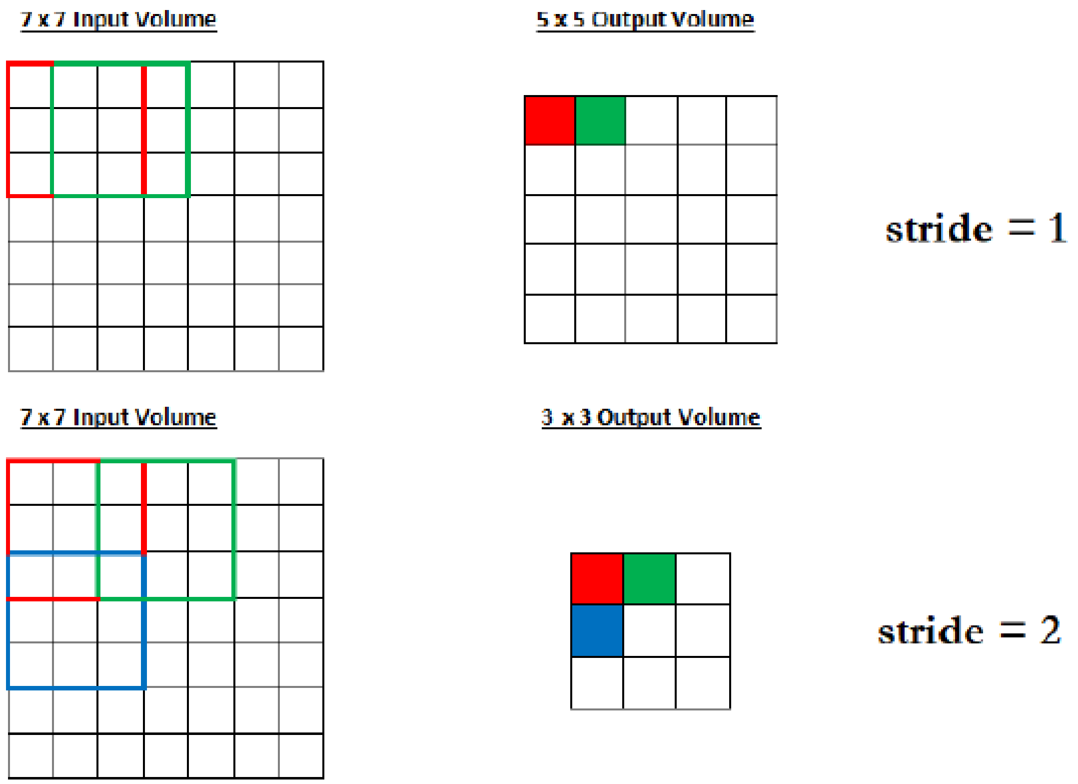




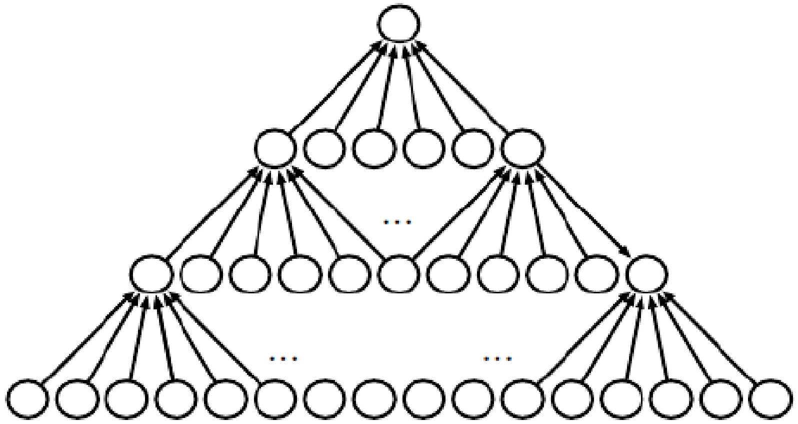
- Property 3. Equivariance against traslations

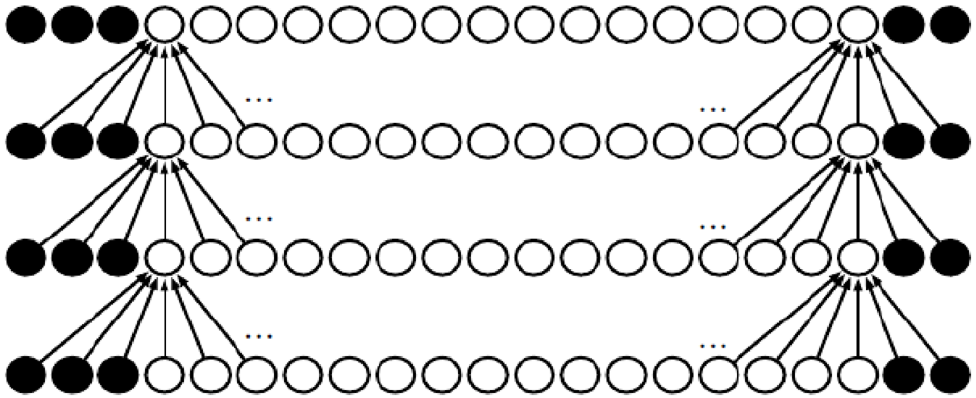
Convolution stride

- Stride of the kernels sliding window (bigger stride, less cost, less detail)



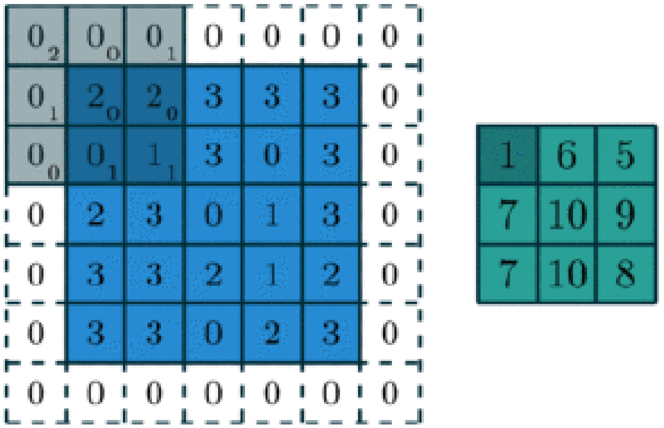
Zero padding





- What if we apply a **5x5** filter to a **32x32** image?
- The resulting image is **28x28** !!
- As we pile convolutional layers, the representation size gets reduced (info loss, specially in first layers)
- Few convo layers
- Mitigate by padding with zero's the image edge

Input 5x5, kernel 3x3, Stride 2

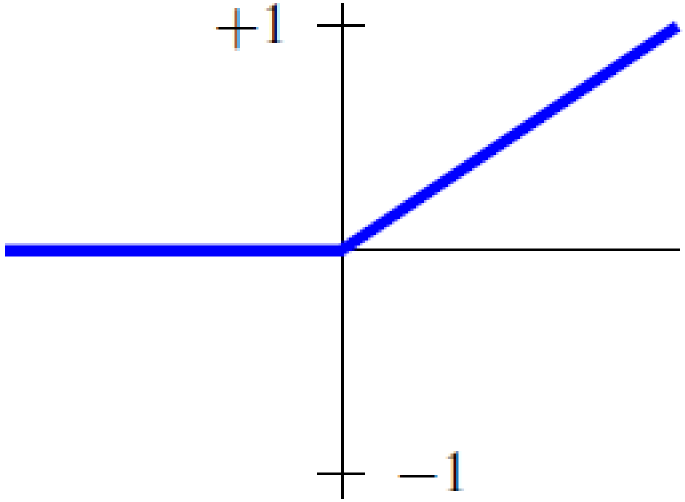


Parameters defining convolution

- Stride
- Kernel size (usually square) 3x3, 5x5
- Depth (number of kernels)
- Padding

Rectified linear unit. RELU

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$



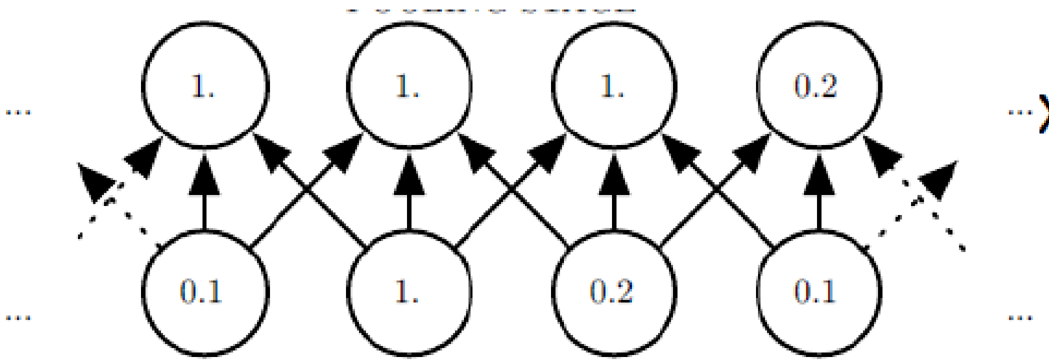
Lineal rectificada (ReLU)

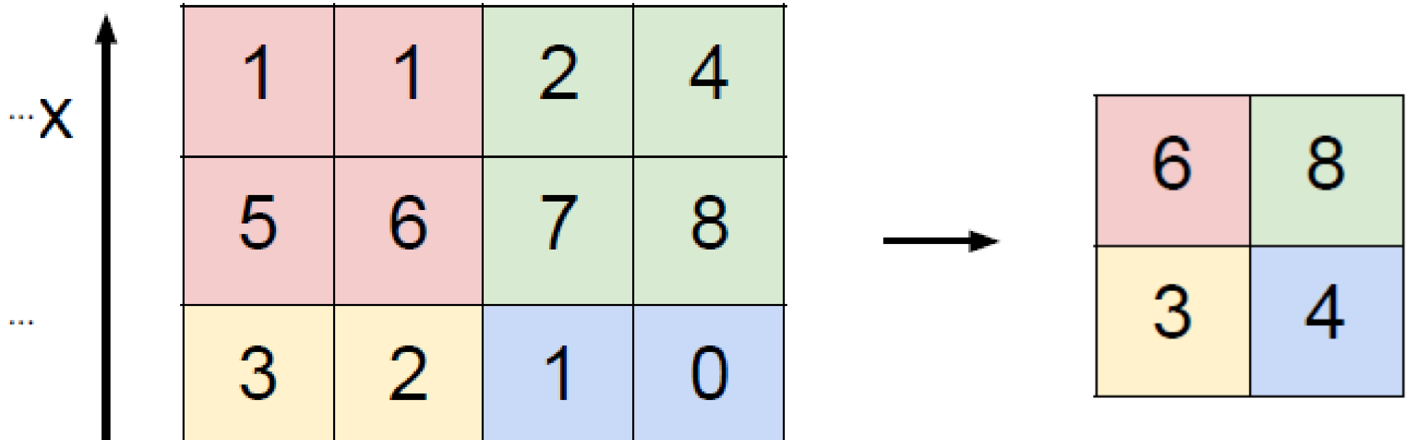
- More efficient computationally than other nonlinear functions (e.g. derivative simple, derivative??)
- Alleviating *vanishing gradient*

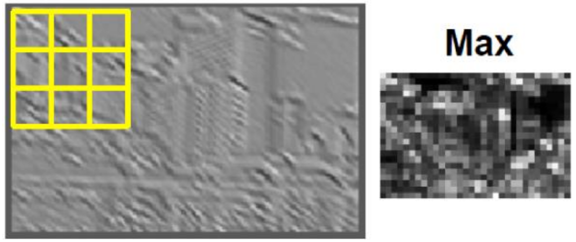
MaxPooling

Pooling. Replaces output in a position by outputs in adjacent positions. Reduces number of pars. Is a feature present?

MaxPooling. Maximum of outputs in adjacent rectangle







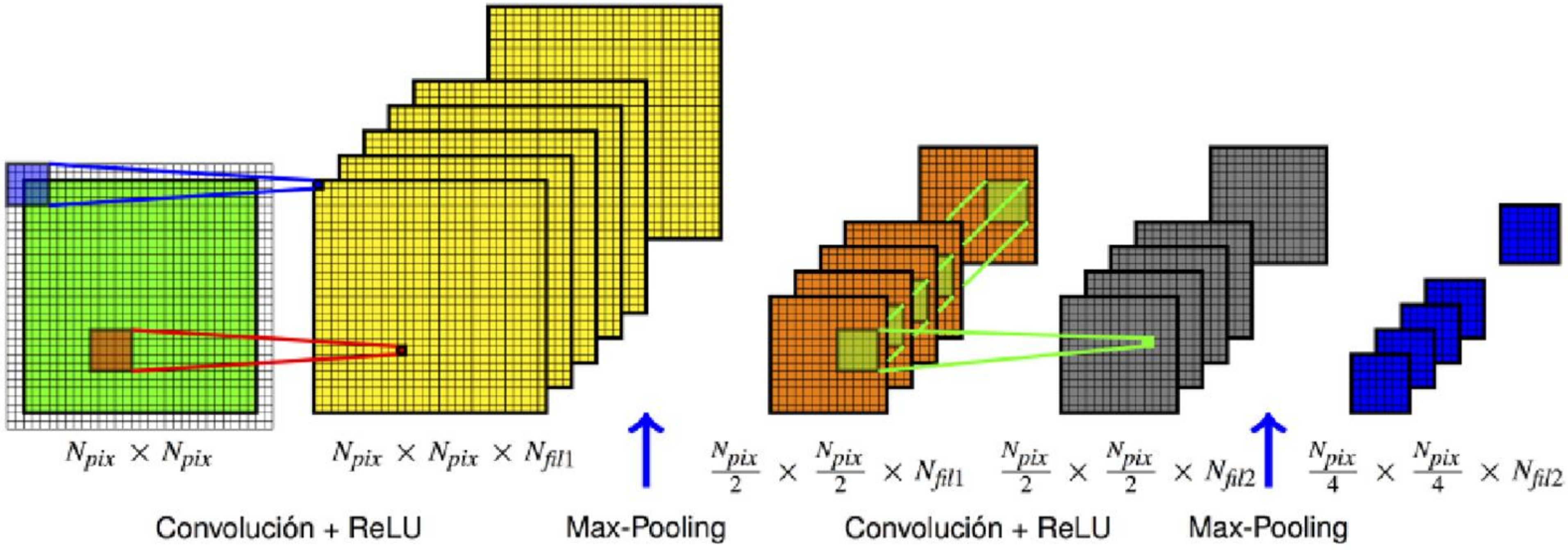
Average Pooling

CNN. Typical structures

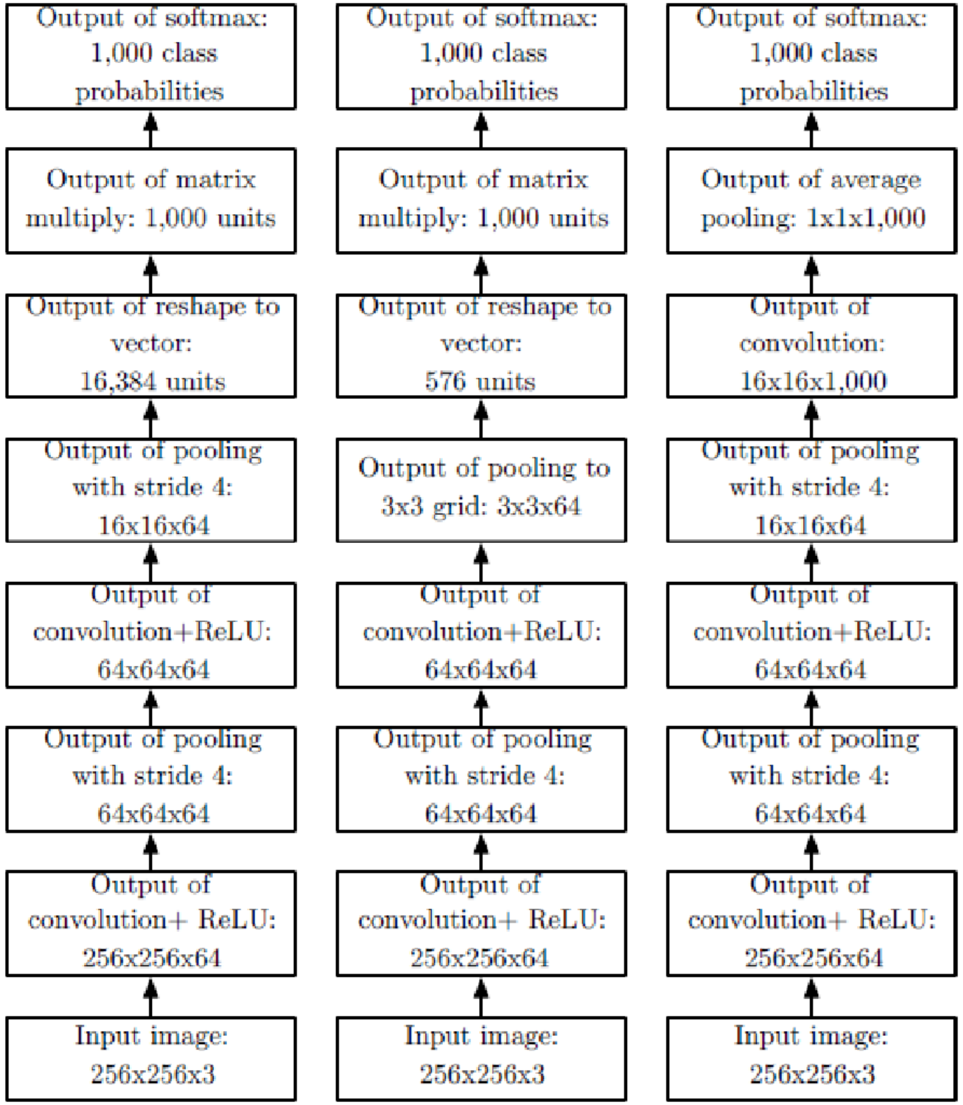
CNNs. Typical structure

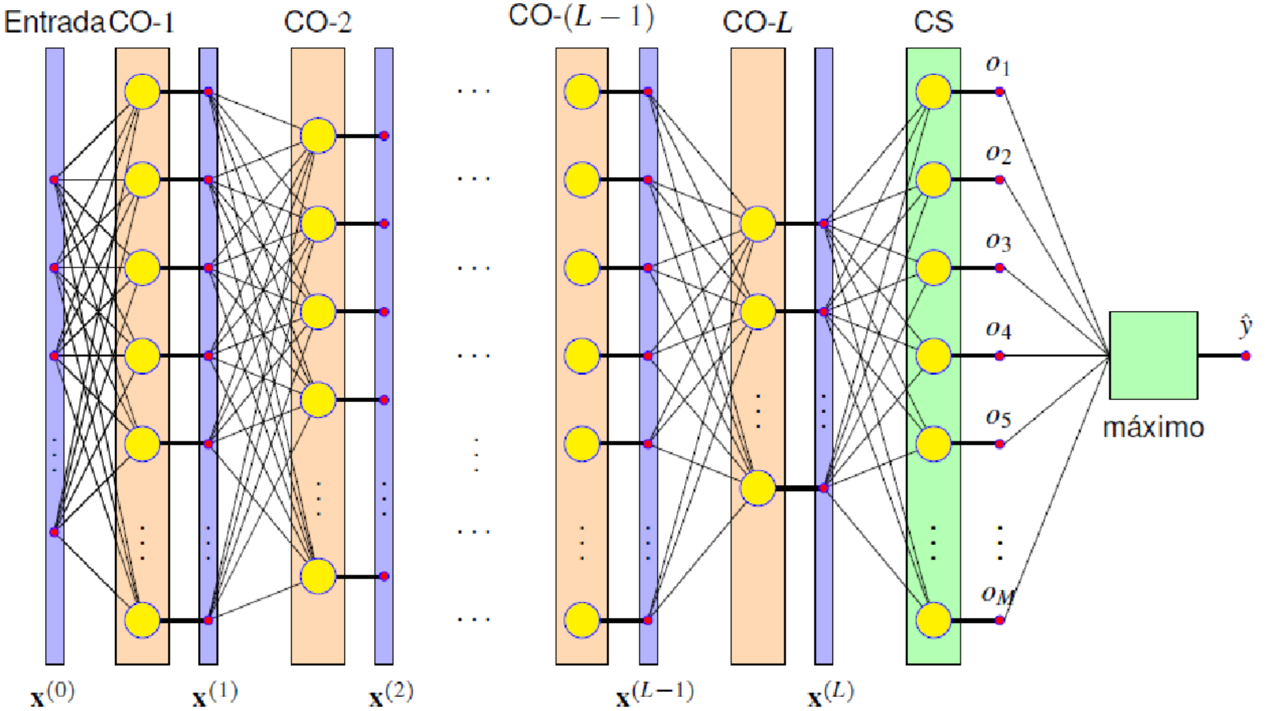
- Input to layer
- (Convolutional) Layer
  - Convolutional phase. Affine transformation
  - Detection phase. Non linearity, e.g. RELU
  - Pooling phase
- Output of layer

CNNs. Typical structure

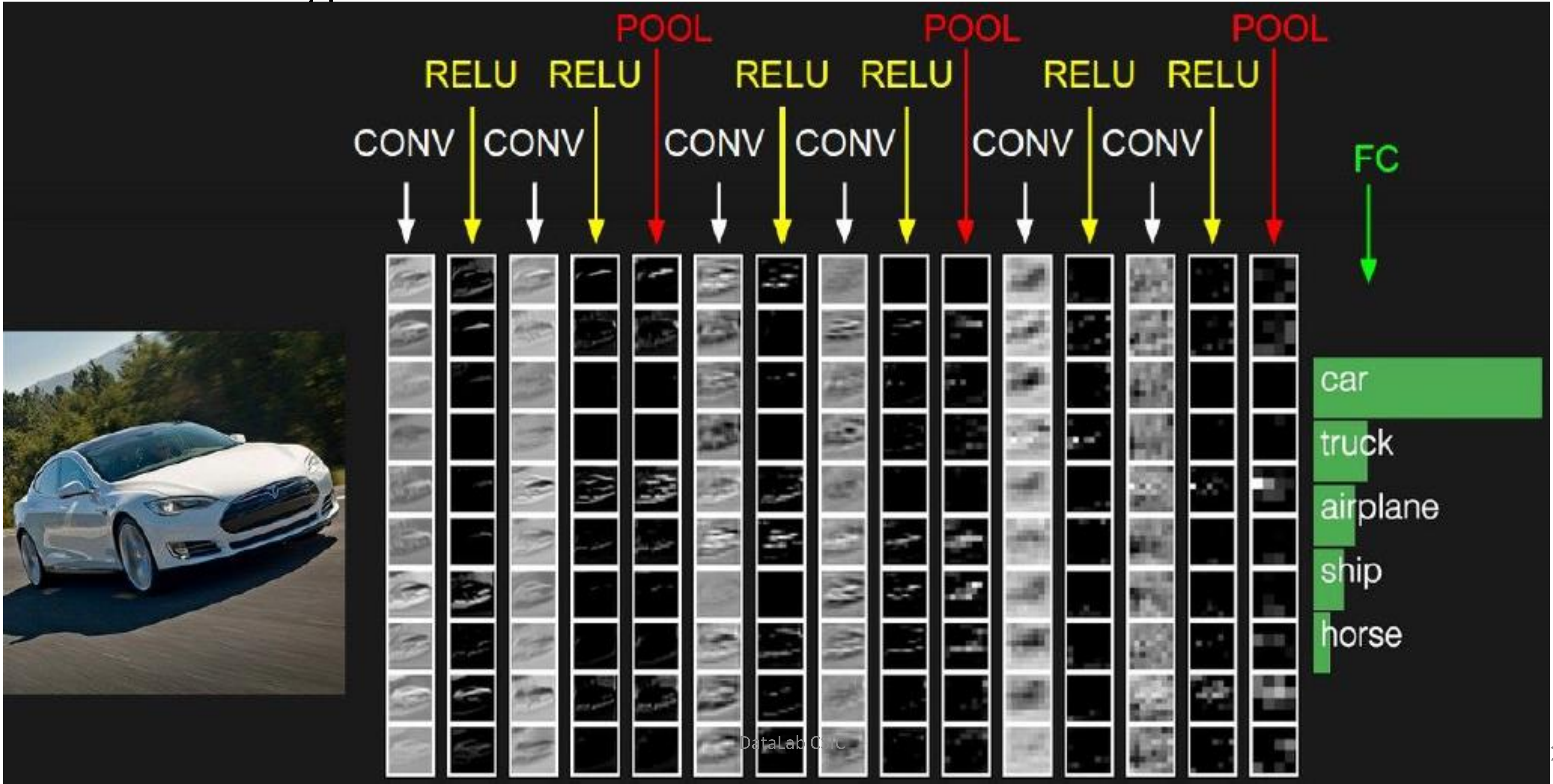


CNNs. Some typical structures





Convnets. Typical structure



The CNN in the proposed lab

MNIST. 28x28 images

1. Conv layer, 32 filters, kernel 3×3, ReLU
2. Conv layer 64 filters, kernel 3×3, ReLU
3. Max pooling 2×2 (dropout 0.25)
4. Dense 128 units, ReLU (dropout 0.5)
5. Dense 10 units (softmax)

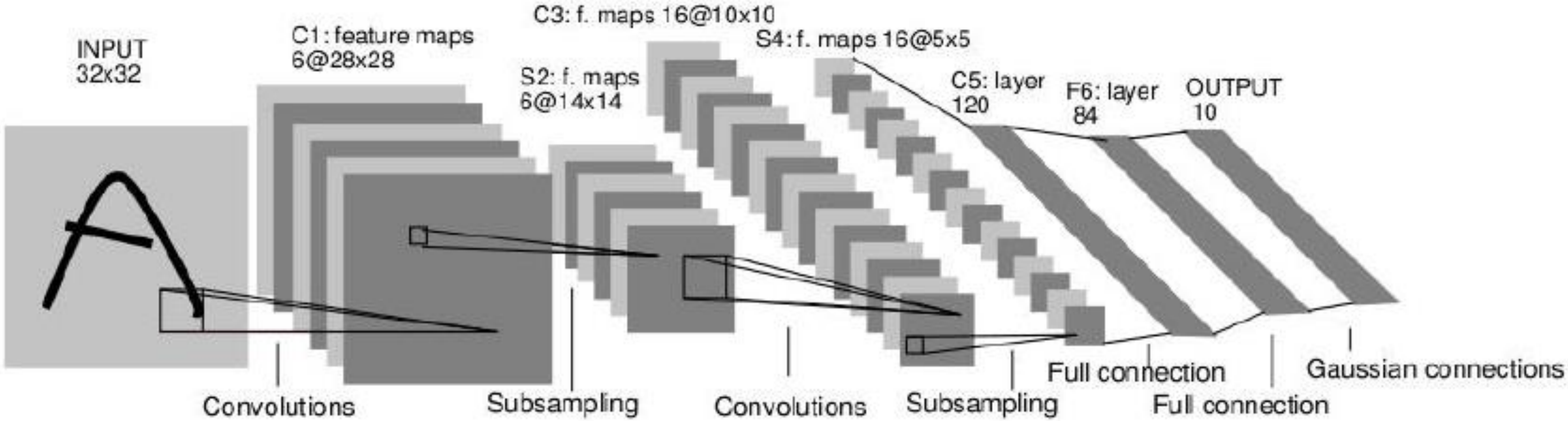
The CNN in the proposed lab. Keras

```
modelf <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu',
                input_shape = input_shape) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = num_classes, activation = 'softmax')
```

Default stride 1, no padding

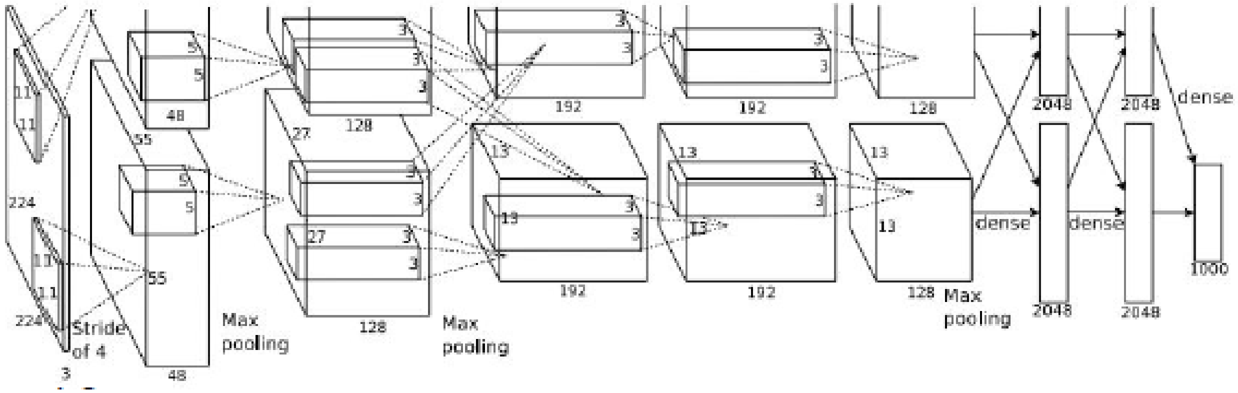
CNN. Some standards

LeNet


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

AlexNet

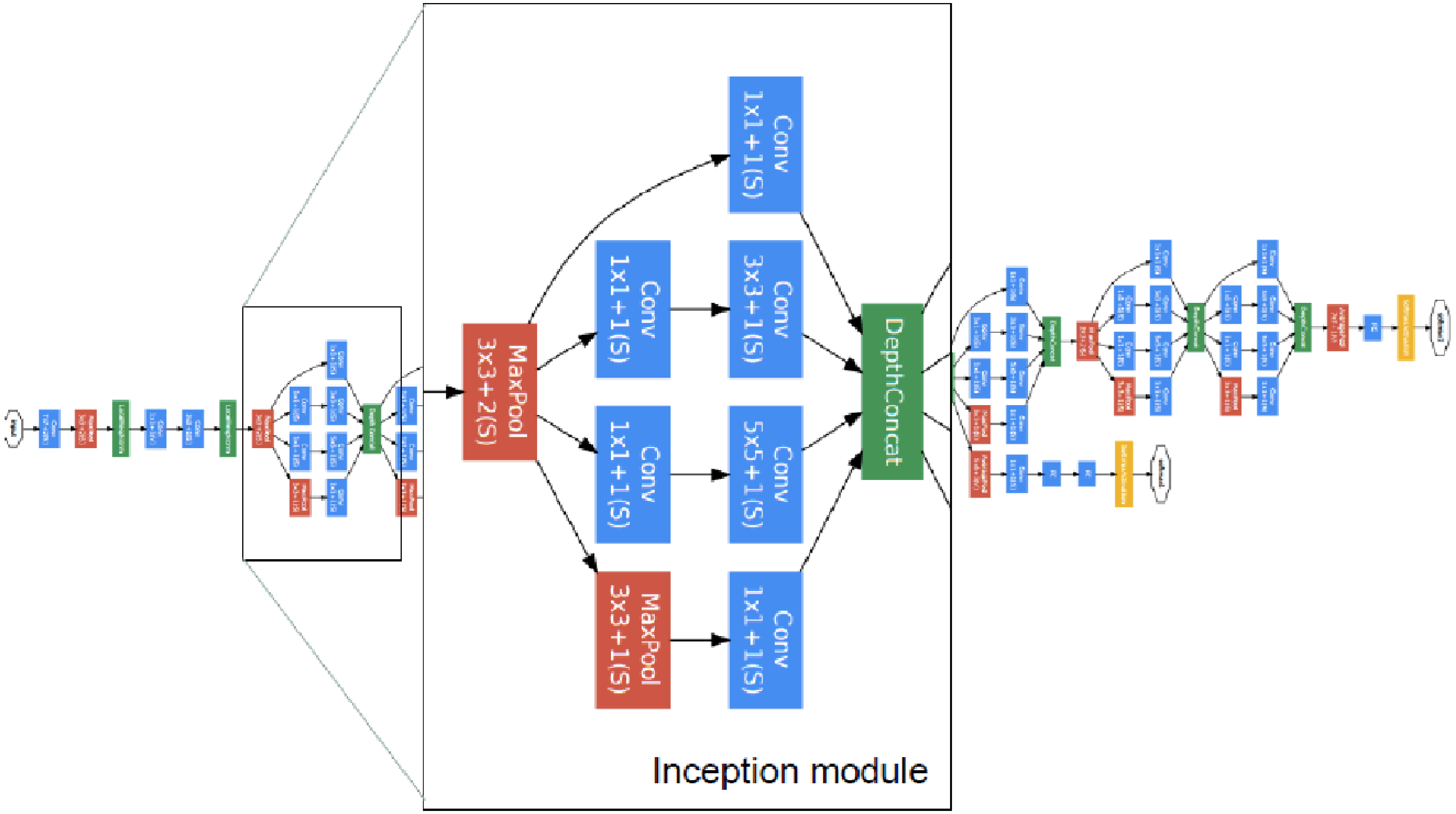
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)



VGGNet

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

GoogleNet



MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 1x1+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
Conv 1x1+1(S)
DepthConcat
Inception module

Some historical architectures

Team	Year	Place	Error (top-5)	External data
SuperVision – Toronto (AlexNet, 7 layers)	2012	-	16.4%	no
SuperVision	2012	1st	15.3%	ImageNet 22k
Clarifai – NYU (7 layers)	2013	-	11.7%	no
Clarifai	2013	1st	11.2%	ImageNet 22k
VGG – Oxford (16 layers)	2014	2nd	7.32%	no
GoogLeNet (19 layers)	2014	1st	6.67%	no
ResNet (152 layers)	2015	1st	3.57%	
Human expert*			5.1%	

Included in Keras

- Xception
- Inception v3
- ResNet50
- VGG16
- VGG19
- MobileNet

Final comments

From fully connected NNs to CNNs.

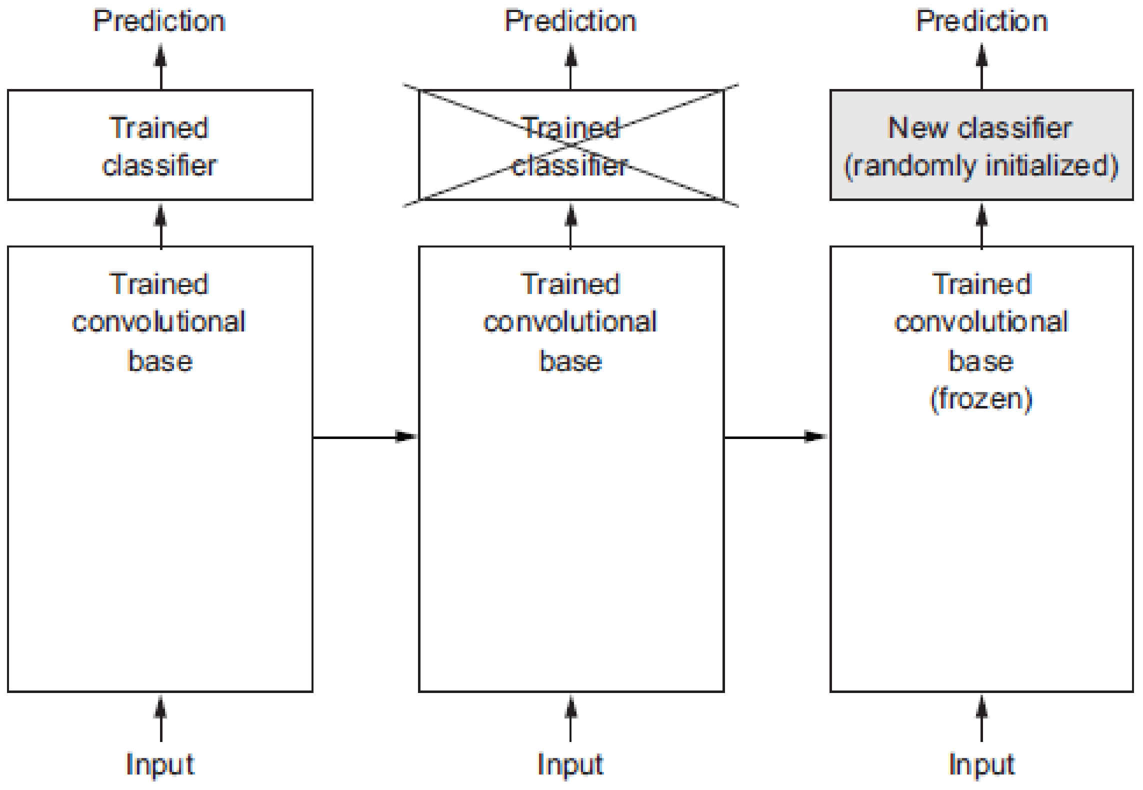
Other paradigms next (RNNs,…)

Once conceived the architecture, train by SGD+backprop (with some tiny variants to increse efficiency)

Keras

Transfer learning

- Take advantage of the architectures already trained in e.g. Keras



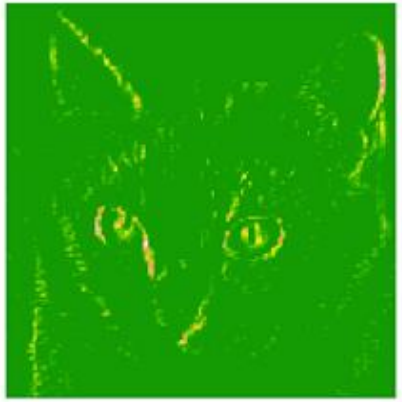

```
conv_base <- application_vgg16(
  weights = "imagenet",
  include_top = FALSE,
  input_shape = c(150, 150, 3)
)
```

Fine tuning

Data augmentation

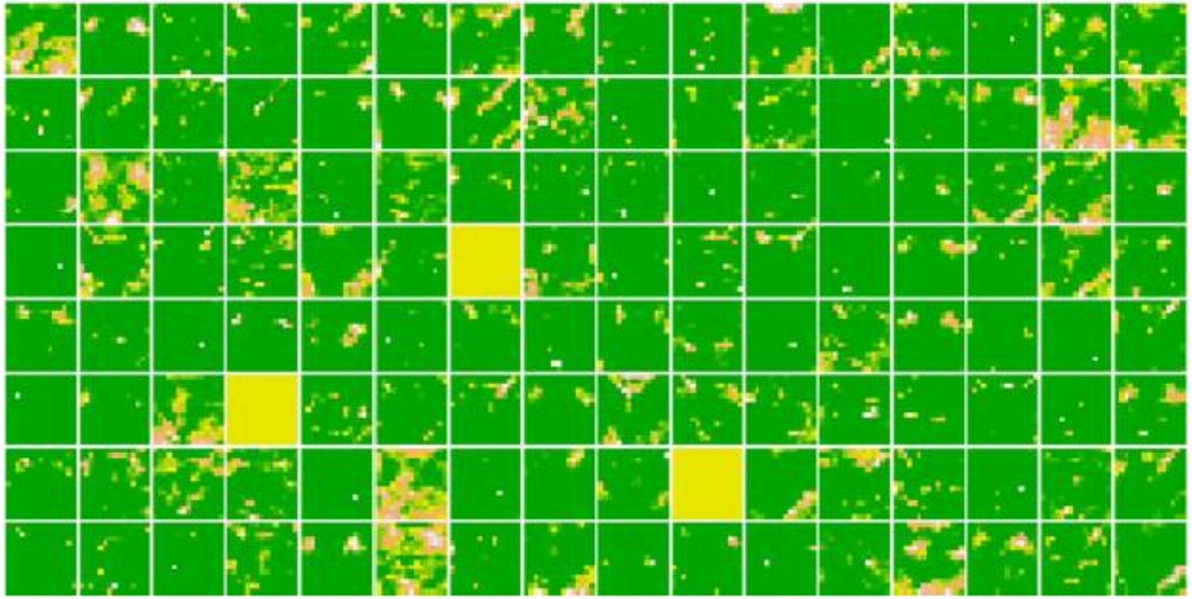
Visualization as explanation

Two channels of first layer



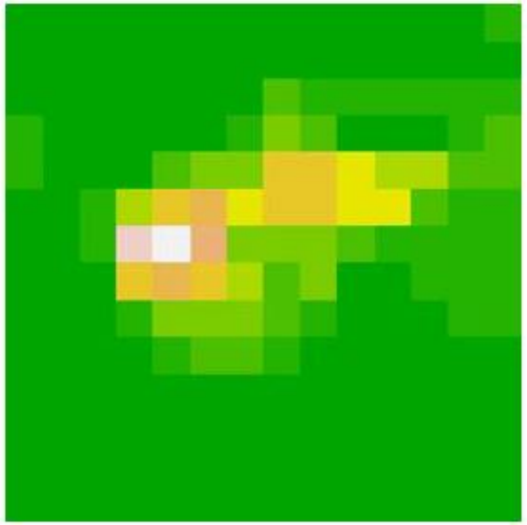
- Visualizing intermediate activations
  (intermediate convnet outputs)

All channels layer 8



Visualization as explanation

- Visualizing heatmaps of class activation (what leads a convnet to classification) (debugging, object location)



Class activation map

Adversarial machine learning

(a) Original image.

(b) Attacked image.

See you soon

introml@icmat.es

Stuff at

https://datalab-icmat.github.io/courses_stats.html