Convolutional Neural Networks

Giacomo Boracchi,
DEIB, Politecnico di Milano
October, 22th, 2020

giacomo.boracchi@polimi.it
https://boracchi.faculty.polimi.it/

To recap

Yesterday we have seen what happens after training a linear classifier (or a NN having no hidden layers) on images

Unroll the image column-wise

-8.1	...	2.7	9.5	...	-9.0	-5.4	...	4.8
9.0	...	5.4	4.8	...	1.2	9.5	...	-8.0
1.2	...	9.5	-8.0	...	8.1	-2.7	...	9.5

W

$*$

23
...
21
34
...
12
34
...
23

$\boldsymbol{x_i}$ $\in \mathbb{R}^N$

$+$

-2
32
-1

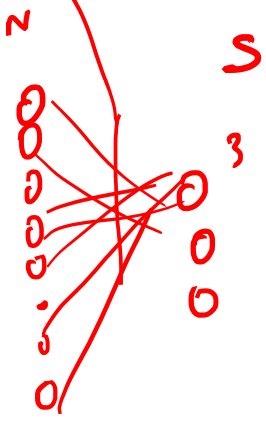
$\boldsymbol{b}$

$=$

-4	s_1 dog score
22	s_2 cat score
33	s_3 rabbit score

$\mathcal{K}(\boldsymbol{x_i}; W, \boldsymbol{b})$

$s = Wx + b$



To recap

Yesterday we have seen what happens after training a linear classifier (or a NN having no hidden layers) on images

Unfold the image column-wise

Training

-8.1	...	2.7	9.5	...	-9.0	-5.4	...	4.8
9.0	...	5.4	4.8	...	1.2	9.5	...	-8.0
1.2	...	9.5	-8.0	...	8.1	-2.7	...	9.5

W

*

23
...
21
34
...
12
34
...
23

$\boldsymbol{x_i}$

\+

-2
32
-1

$\boldsymbol{b}$

=

-4	s_1 dog score
22	s_2 cat score
33	s_3 rabbit score

$\mathcal{K}(\boldsymbol{x_i}; W, \boldsymbol{b})$

Fold each row back to an image

CS231n: Convolutional Neural Networks for Visual Recognition
http://cs231n.github.io/

The Class Score

The classification score is then computed as the correlation between each input image and the «template» of the coresponding class

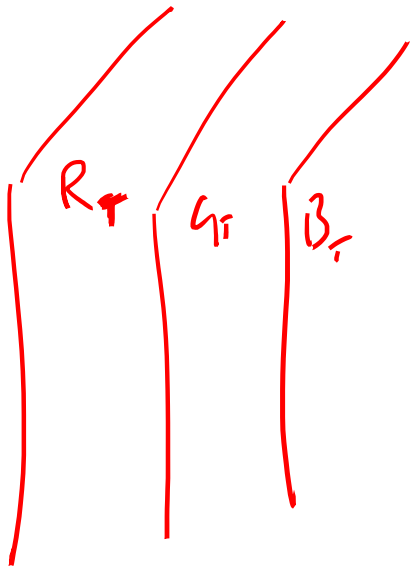
$$\sum_{(x,y)\in U} w_1(x,y) * I(r+x, c+y)$$

Correlation between two RGB images

The image and the filter have the same size

I

$\circledast$



T_1

$$(R_I * R_T)(0,0) +$$
$$(G_I * G_T)(0,0) +$$
$$(R_I * G_T)(0,0)$$

$w_1 \cdot x$

$$R_I \circledast R_T + G_I \circledast G_T + B_I * B_T =$$
$$\sum_{x,y} R_I(x,y) \cdot R_T(x,y) + \dots$$

The Feature Extraction Perspective

The Feature Extraction Perspective

Images can not be directly fed to a classifier

We need some intermediate step to:

- Extract meaningful information (to our understanding)
- Reduce data-dimension

We need to extract features:

- The better our features, the better the classifier

The Feature Extraction Perspective

Input image

$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction Algorithm

"expert driven"

$\mathbf{x} \in \mathbb{R}^d$

$(d \ll r_1 \times c_1)$

Classifier

NN

"wheel"

$y \in \Lambda$

The Feature Extraction Perspective

Input image

$I_2 \in \mathbb{R}^{r_2 \times c_2}$

Feature Extraction Algorithm

$\mathbf{x} \in \mathbb{R}^d$

$(d \ll r_2 \times c_2)$

Classifier

"castle"

$y \in \Lambda$

Hand-Crafted Features

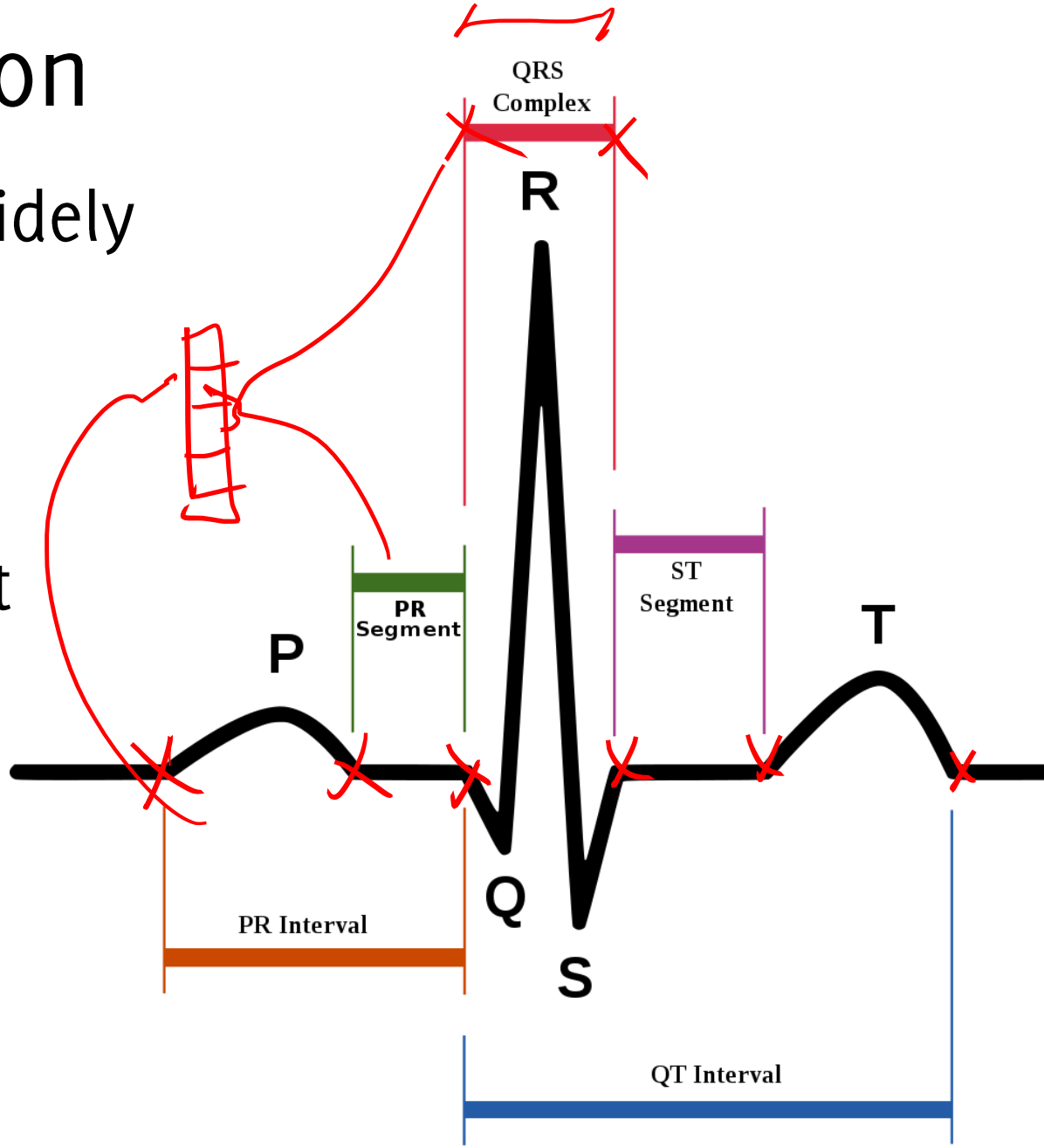
Example: ECG classification

Heartbeats morphology has been widely investigated

Doctors know which patterns are meaningful for classifying each beat

Features are extracted from landmarks indicated by doctors:

e.g. QT distance, RR distance…



Created by Agateller (Anthony Atkielski), Public Domain, https://commons.wikimedia.org/w/index.php?curid=1560893

Hand Crafted Featues

Engineers:

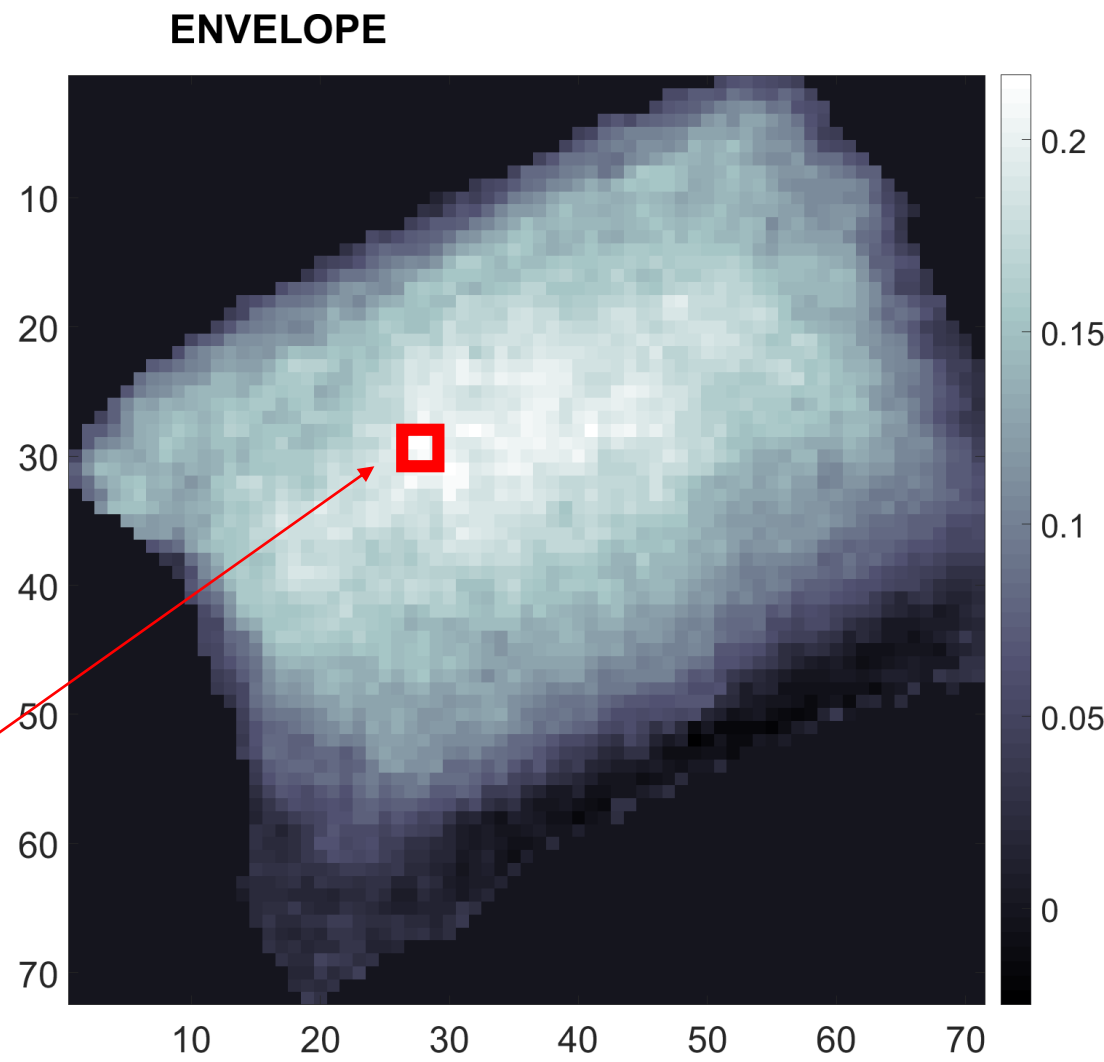
- know what's meaningful in an image (e.g. a specific color/shape, the area, the size)
- can implement algorithms to map these information in a feature vector

Feature Extraction Algorithm

An Illustrative Example: Parcel Classification

Images acquired from a RGB-D sensor:

- No color information provided
- A few pixels report depth measurements
- Images of 3 classes
  - ENVELOPE
  - PARCEL
  - DOUBLE

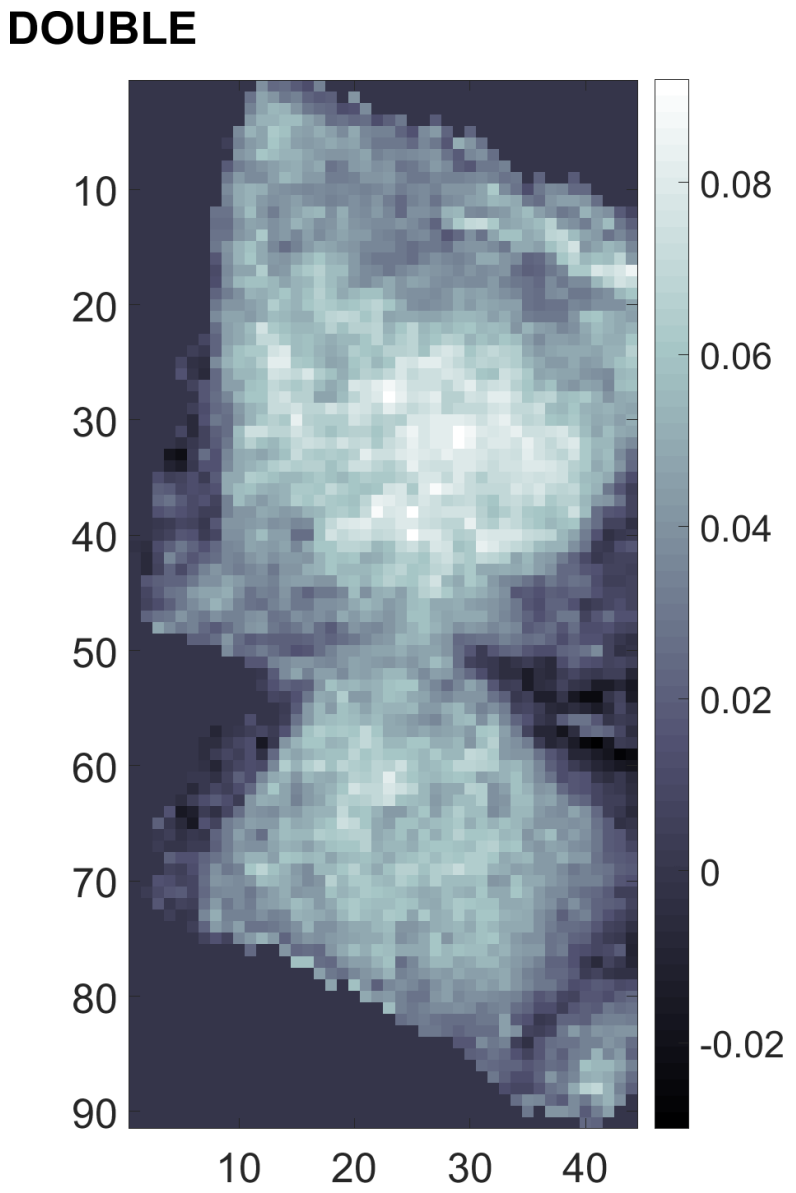


Envelop height at that pixel

An Illustrative Example: Parcel Classification

Images acquired from a RGB-D sensor:

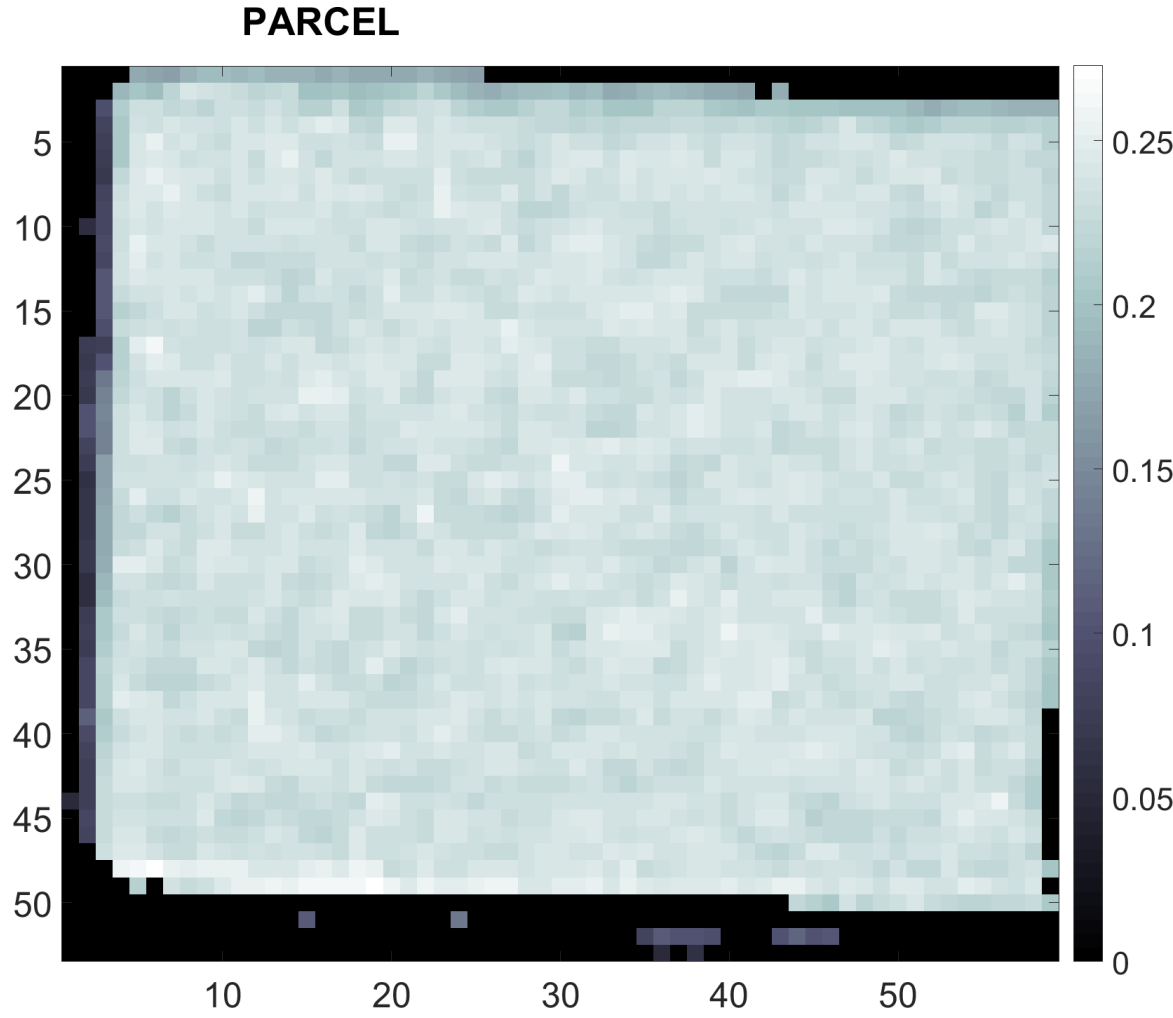
- No color information provided
- A few pixels report depth measurements
- Images of 3 classes
  - ENVELOPE
  - PARCEL
  - DOUBLE



An Illustrative Example: Parcel Classification

Images acquired from a RGB-D sensor:

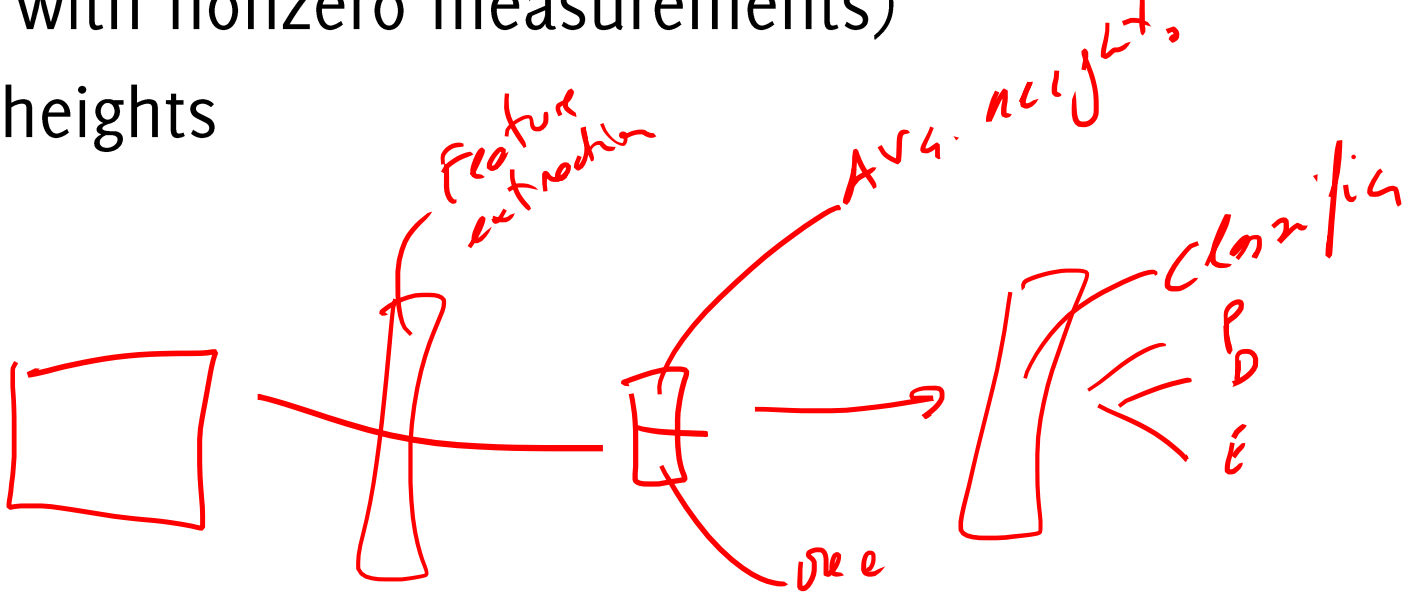
- No color information provided
- A few pixels report depth measurements
- Images of 3 classes
  - ENVELOPE
  - PARCEL
  - DOUBLE



Example of Hand-Crafted Features

Example of features:

- Average height
- Area (coverage with nonzero measurements)
- Distribution of heights
- Perimeter
- Diagonals



The Training Set

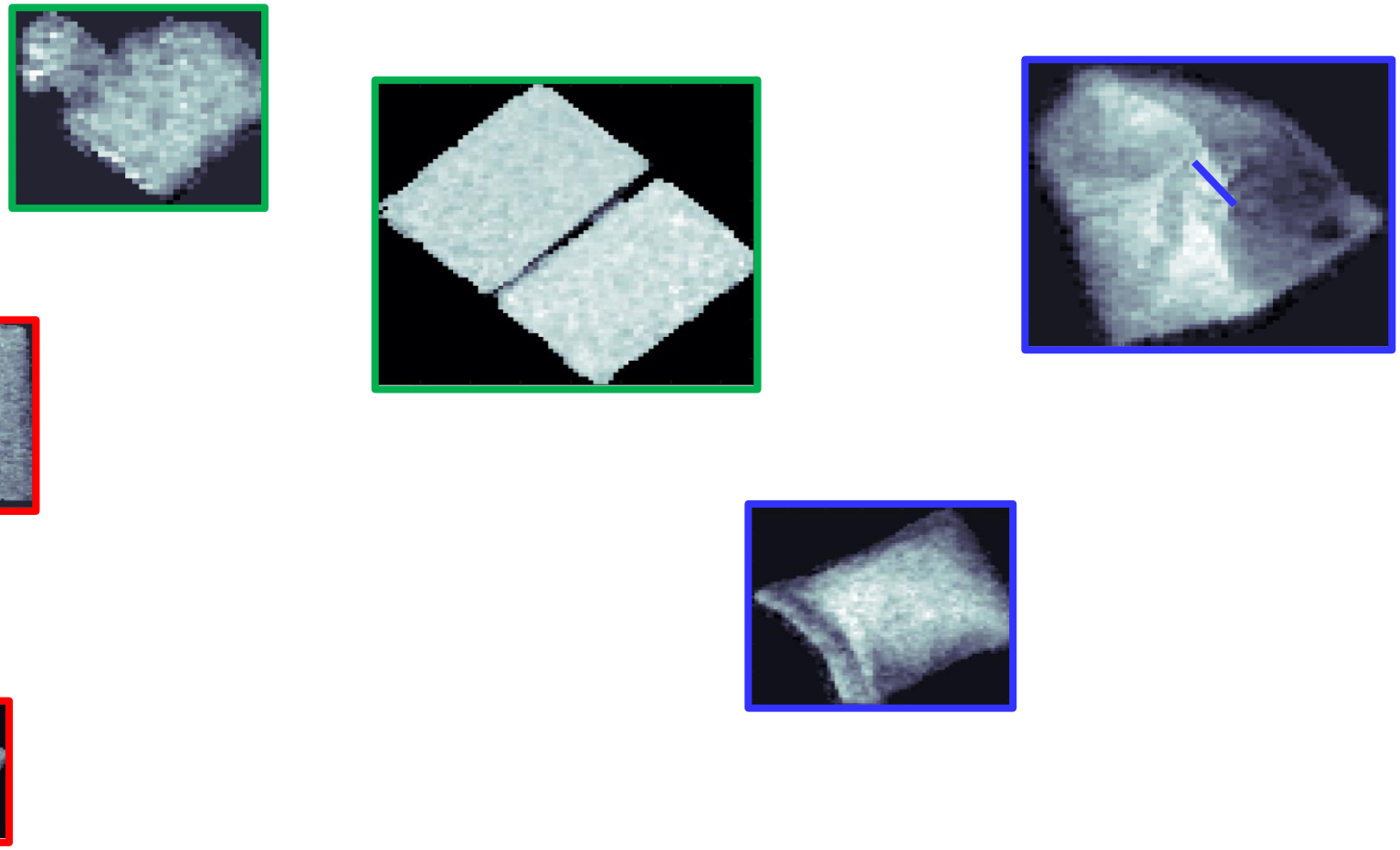
The training set is a set of annotated examples

$$TR = \{(I, l)_i, i = 1, \ldots, N\}$$

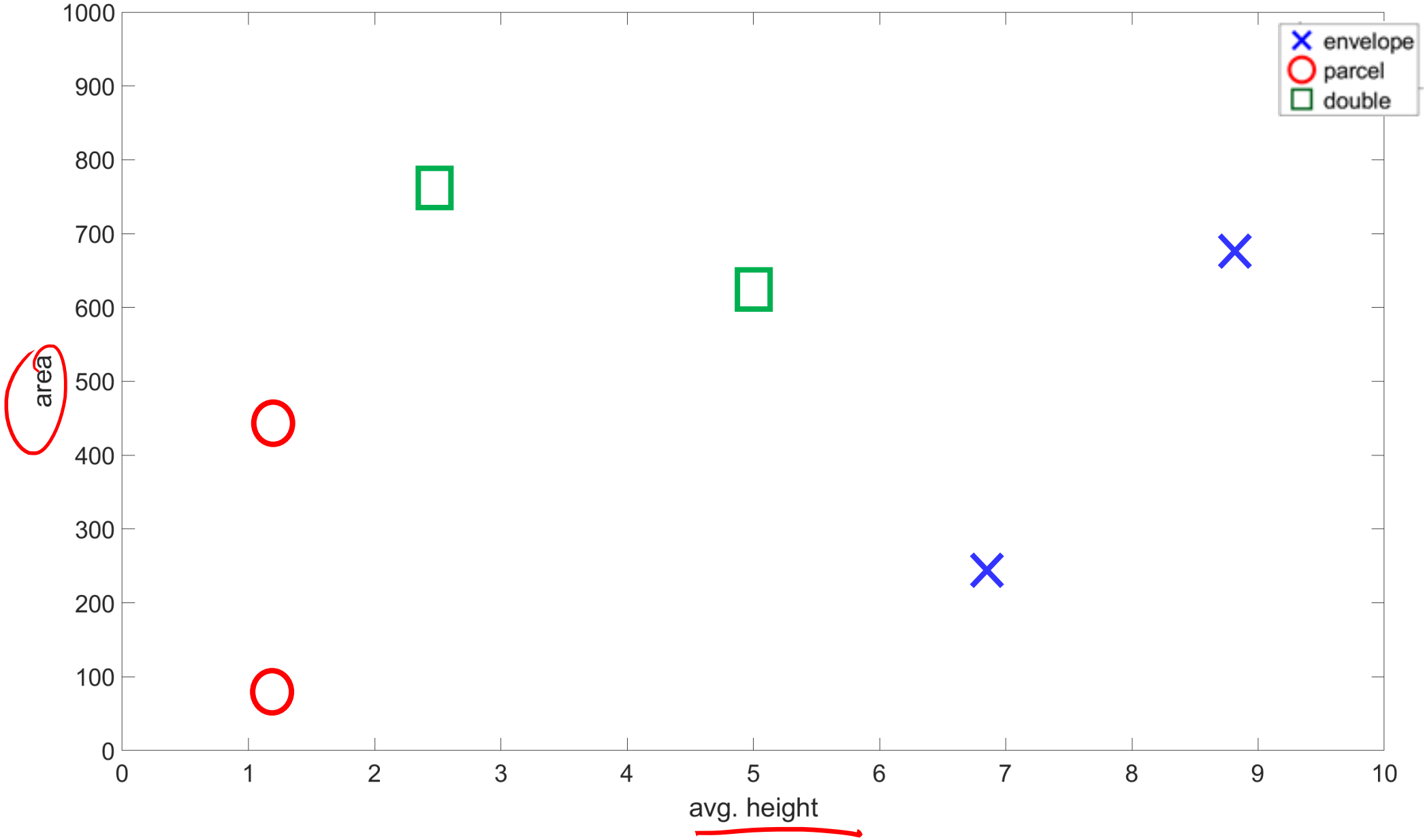
Each couple $(I, l)_i$ corresponds to:

- an image I_i
- the corresponding label l_i

The Training Set: images + labels

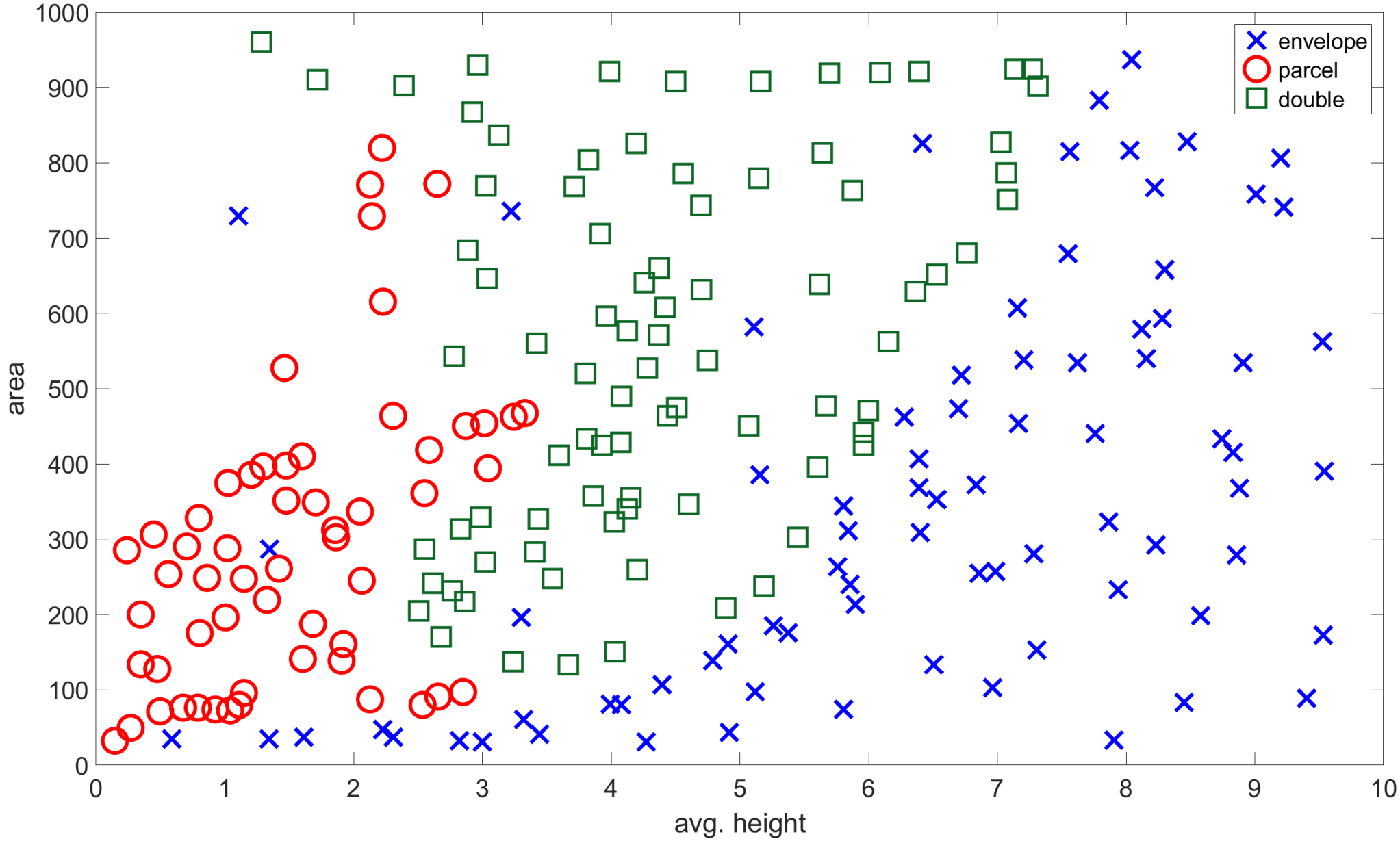


The Training Set: images + labels



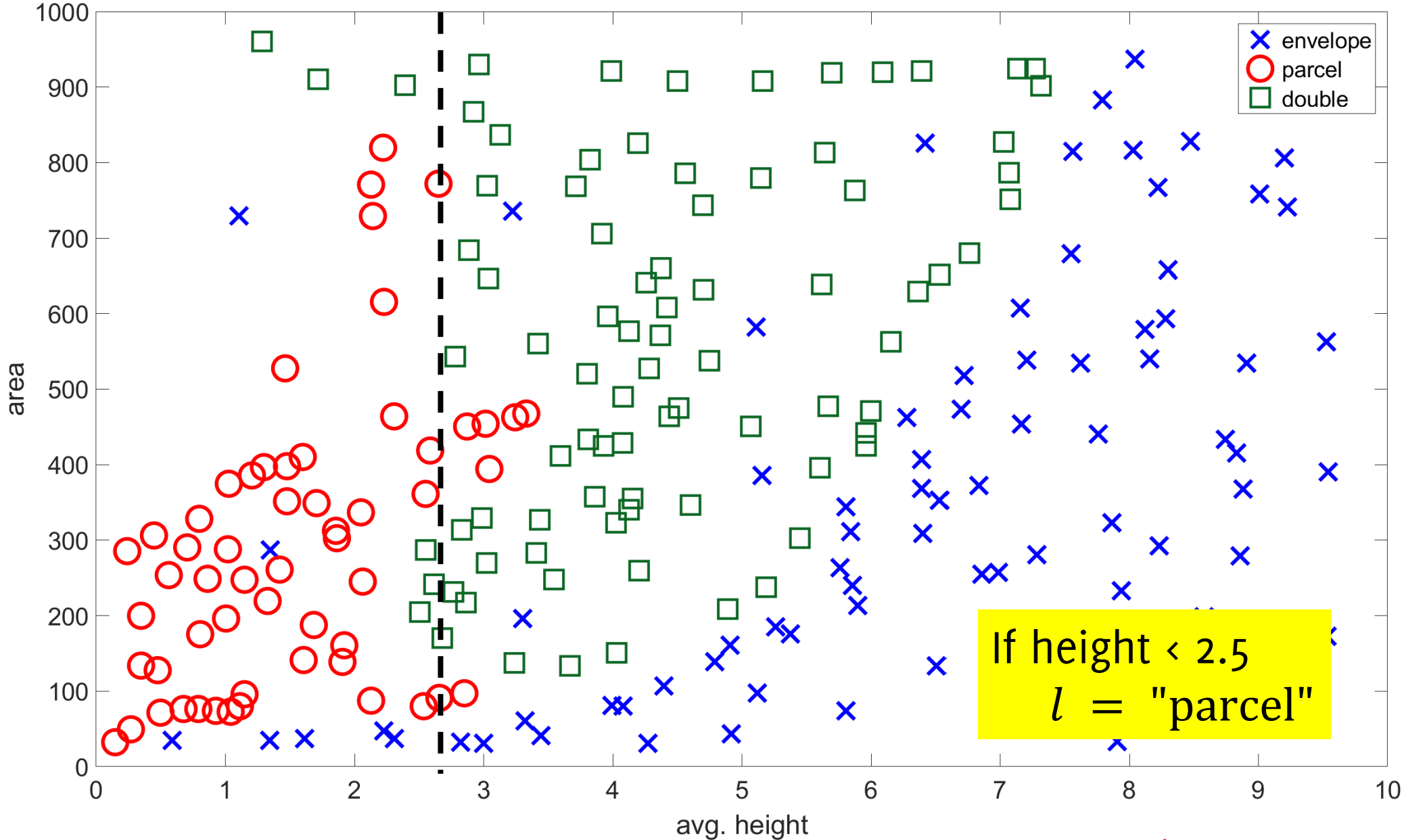
G. Boracchi

The Training Set



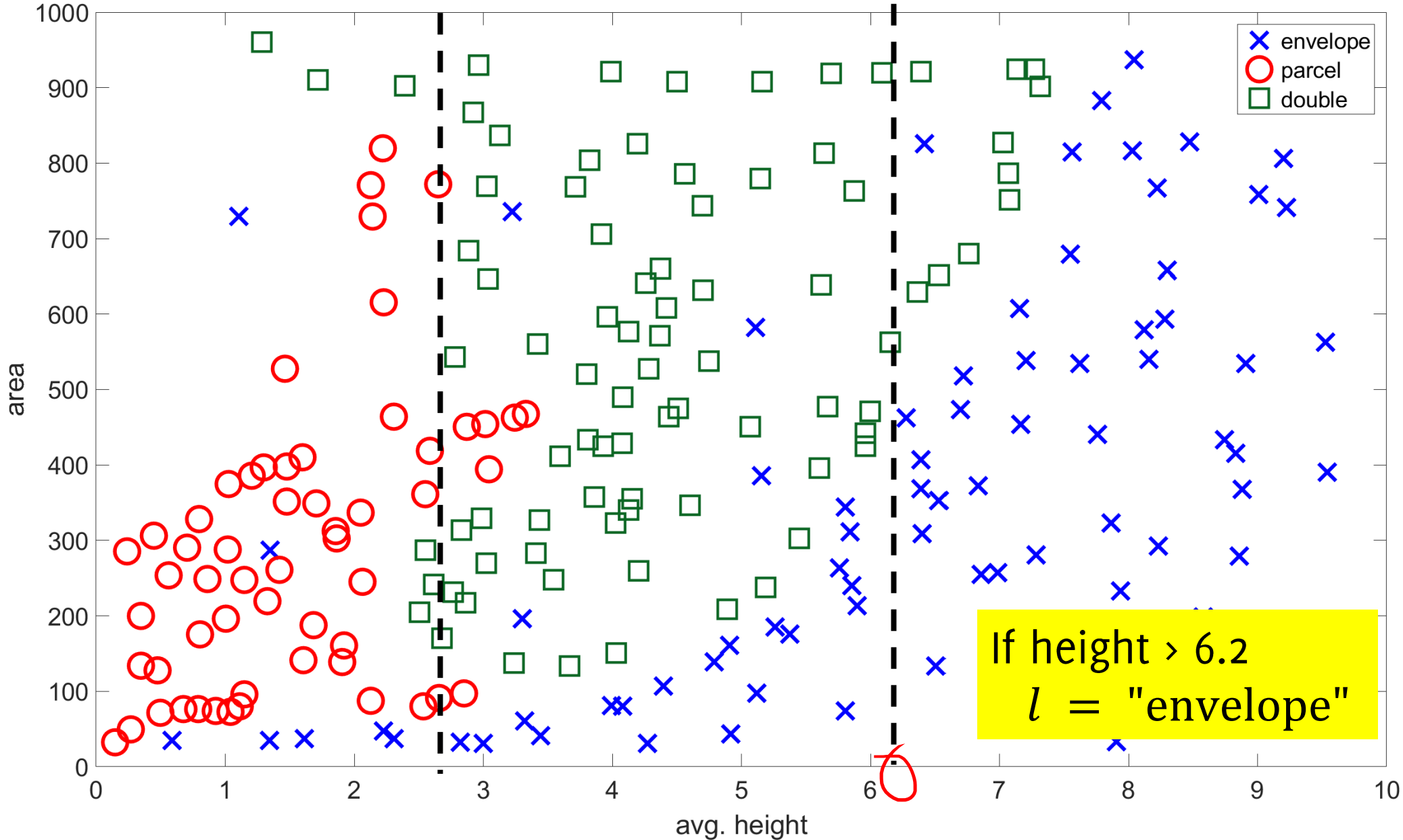
G. Boracchi

Training Set



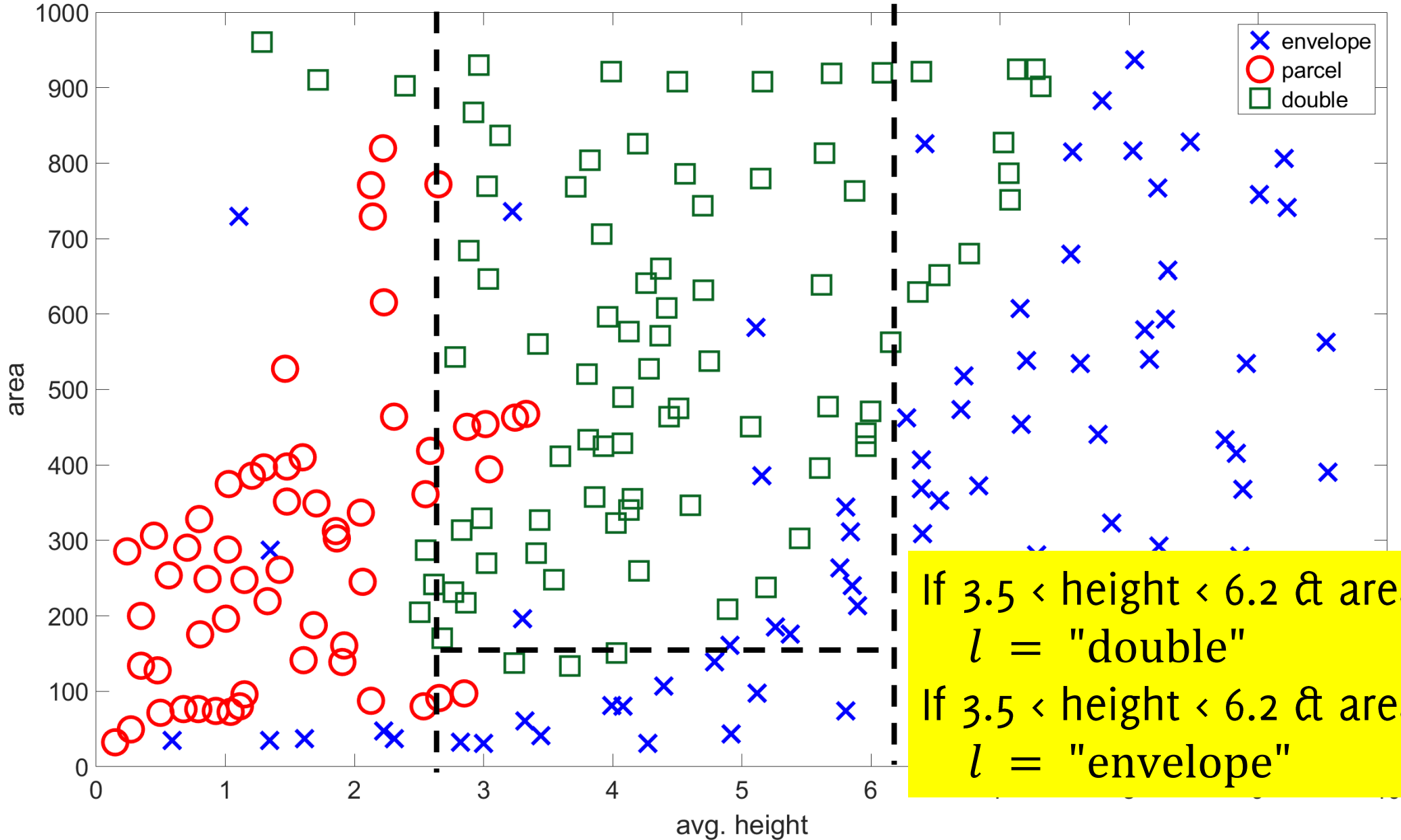
G. Boracchi

Training Set



G. Boracchi

Training Set

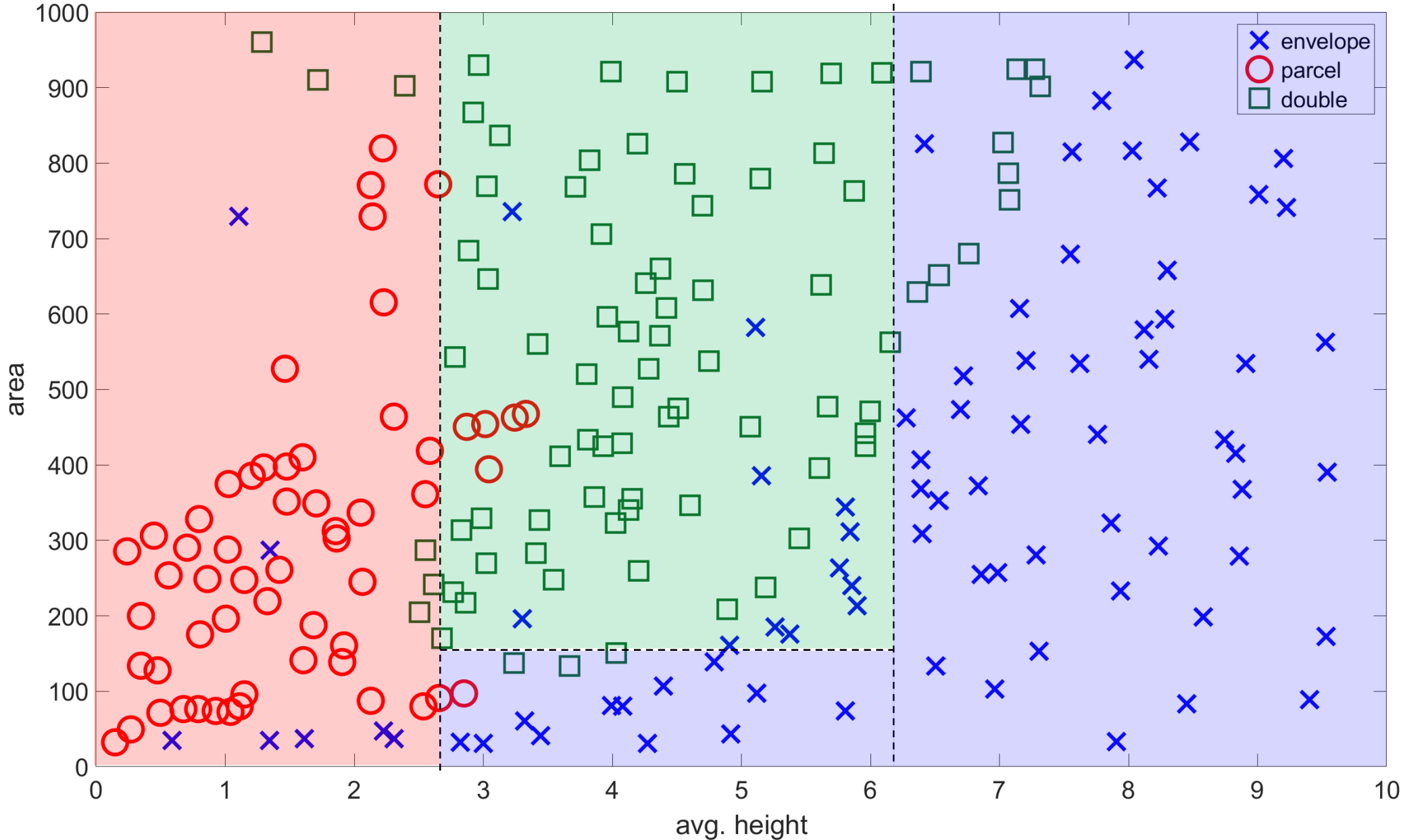


If $3.5 < \text{height} < 6.2$ & $\text{area} > 200$
l = "double"

If $3.5 < \text{height} < 6.2$ & $\text{area} < 200$
l = "envelope"

G. Boracchi

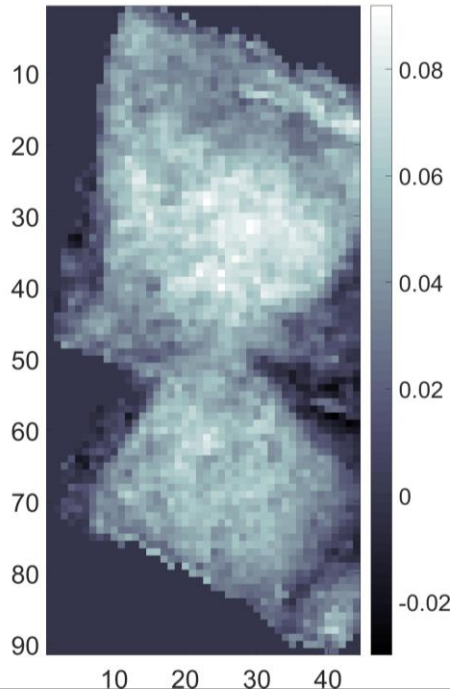
Classifier output



G. Boracchi

A tree classifying image features

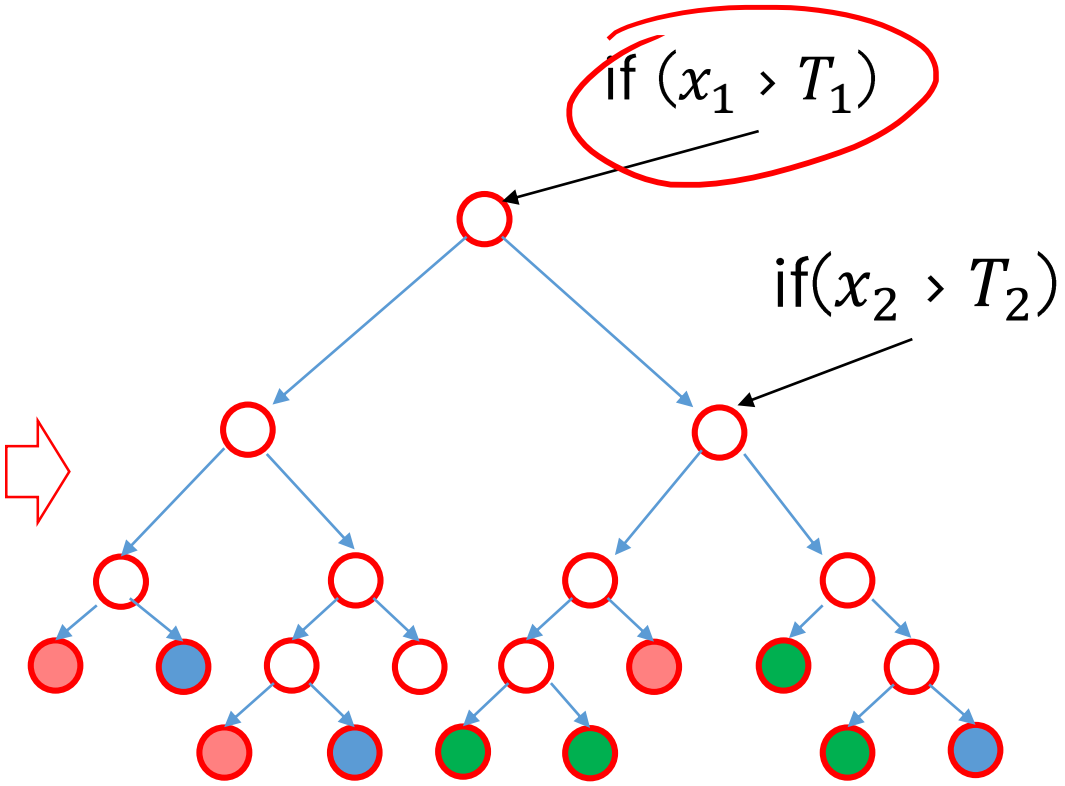
Input image



$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction Algorithm

$\mathbf{x} \in \mathbb{R}^2$



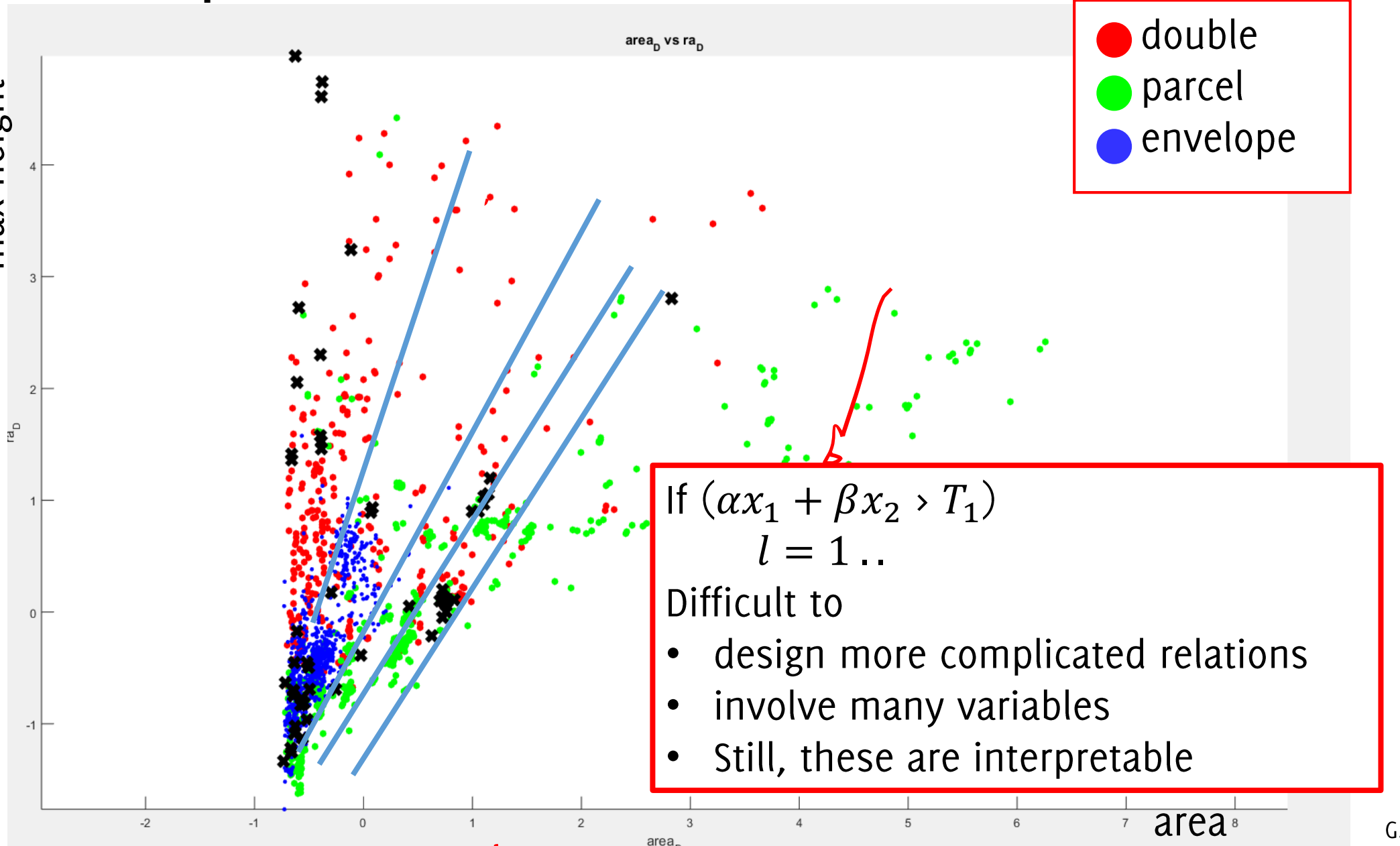
"double"

"envelope"

"parcel"

G. Boracchi

These separation lines are difficult to design



If $(\alpha x_1 + \beta x_2 \rangle T_1)$

$l = 1..$

Difficult to

- design more complicated relations
- involve many variables
- Still, these are interpretable

G. Boracchi

Limitations of Rule Based Classifier

It is difficult to grasp what are meaningful dependencies over multiple variables (it is also impossible to visualize these)

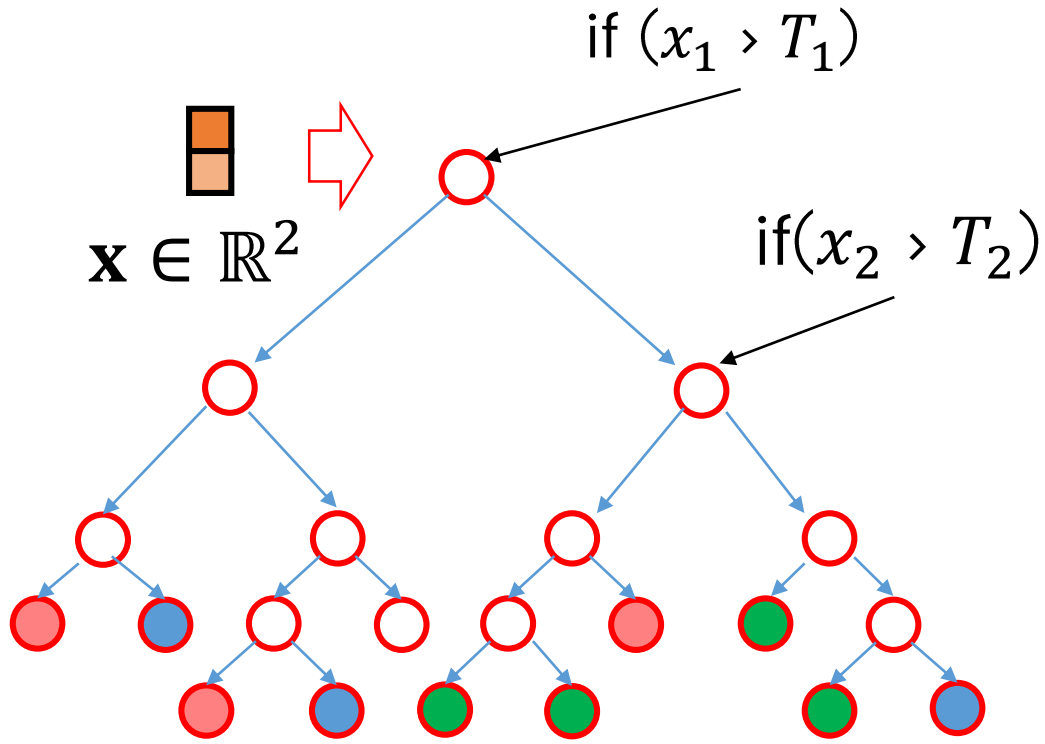
Let's resort to a **data-driven model** for the only task of separating feature vectors in different classes.

How can a classifier achieve better performance?

A tree classifying image features

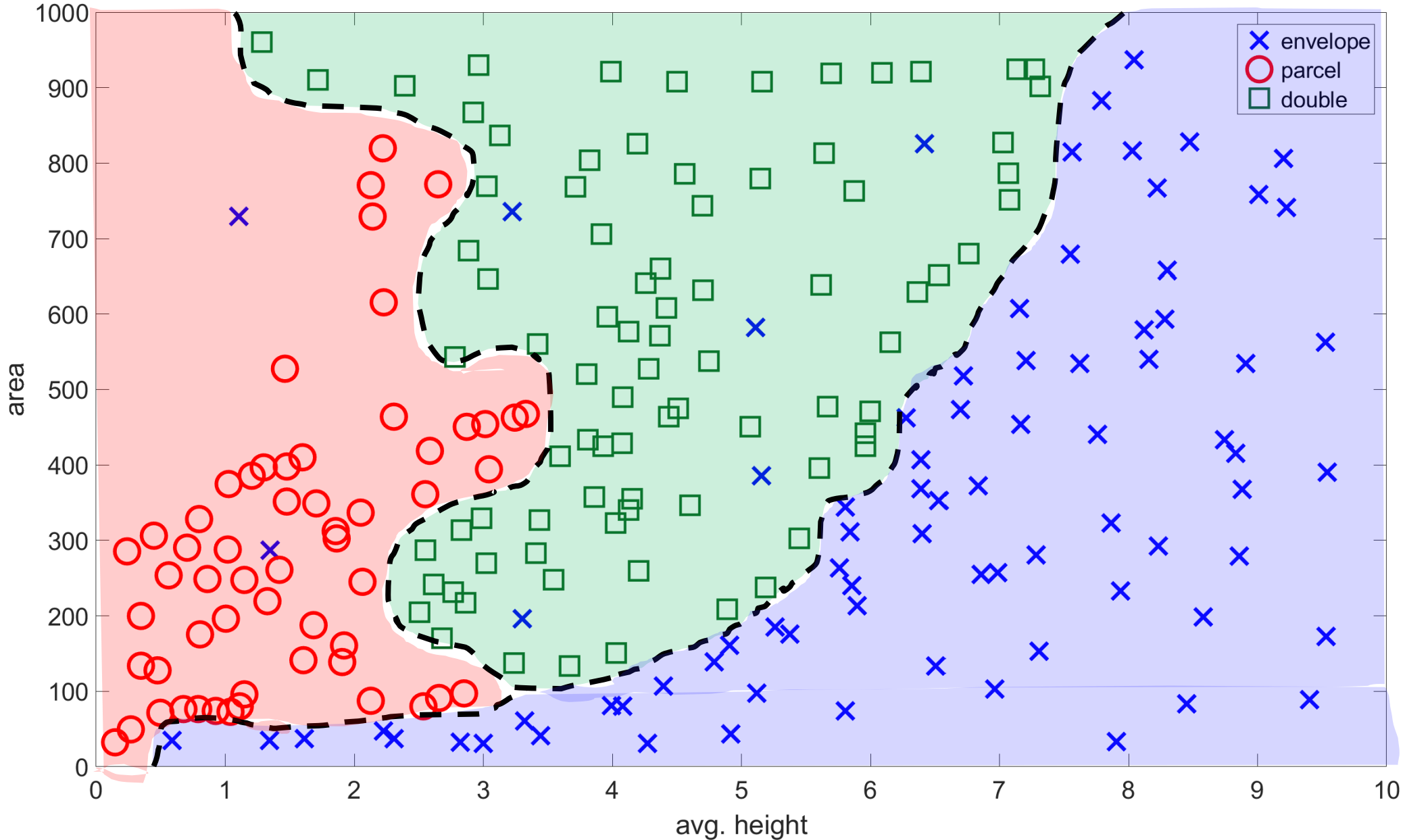
The classifier has a few patameters:

- The **splitting criteria**
- The **splitting thresholds** $\boldsymbol{T_i}$



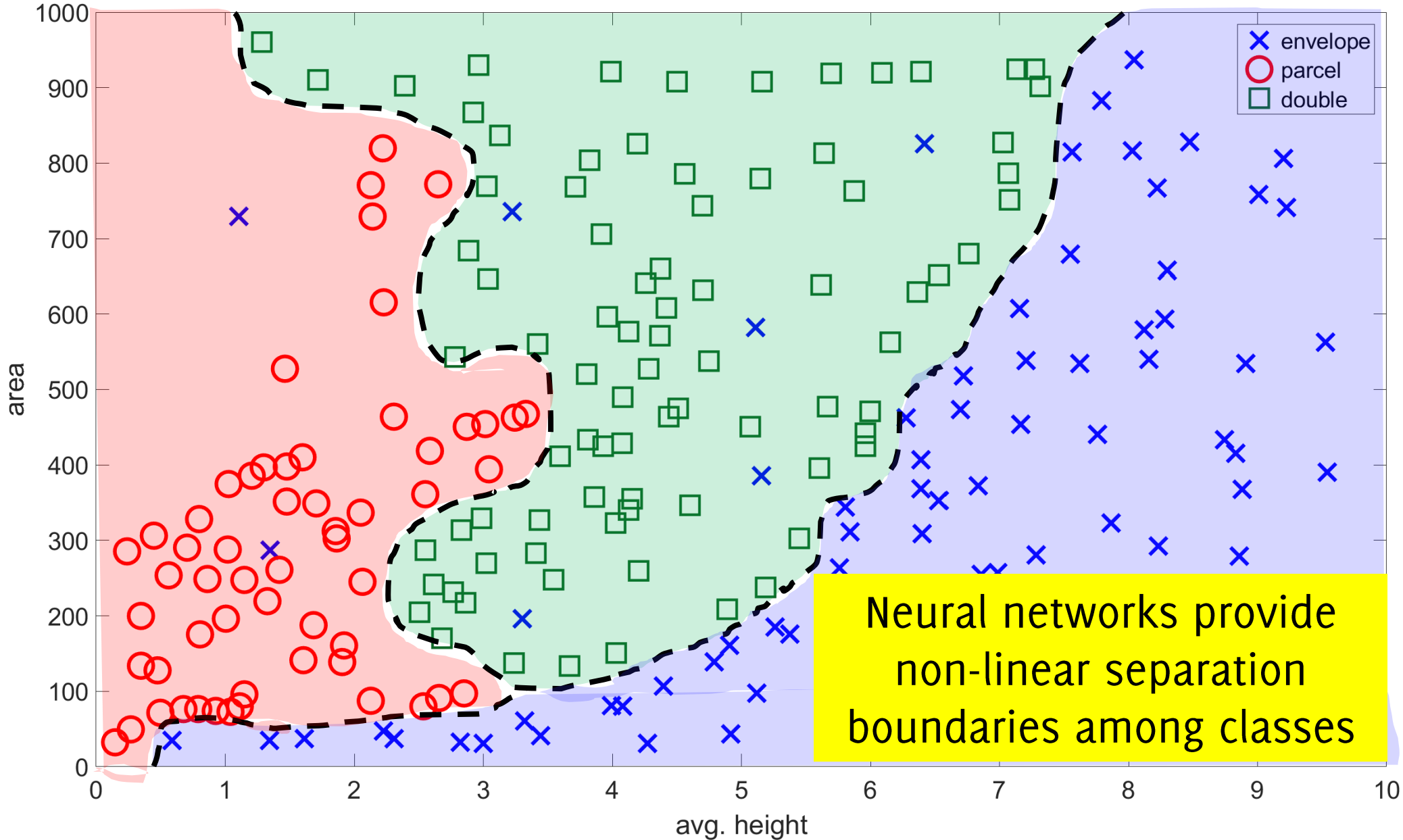
"double" "envelope" "parcel"

Are there better classifiers?



G. Boracchi

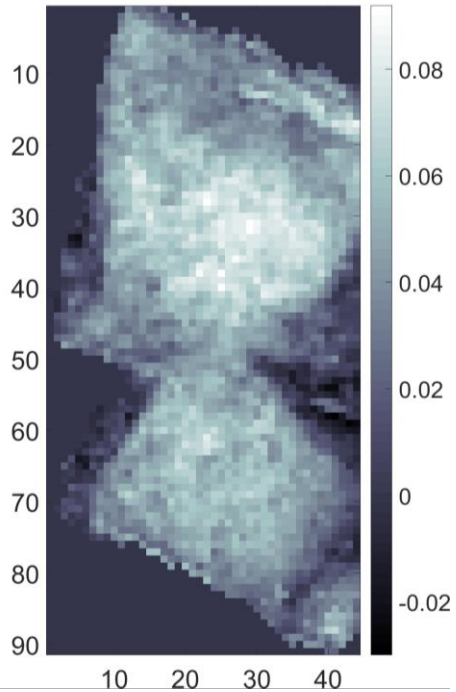
Are there better classifiers?



G. Boracchi

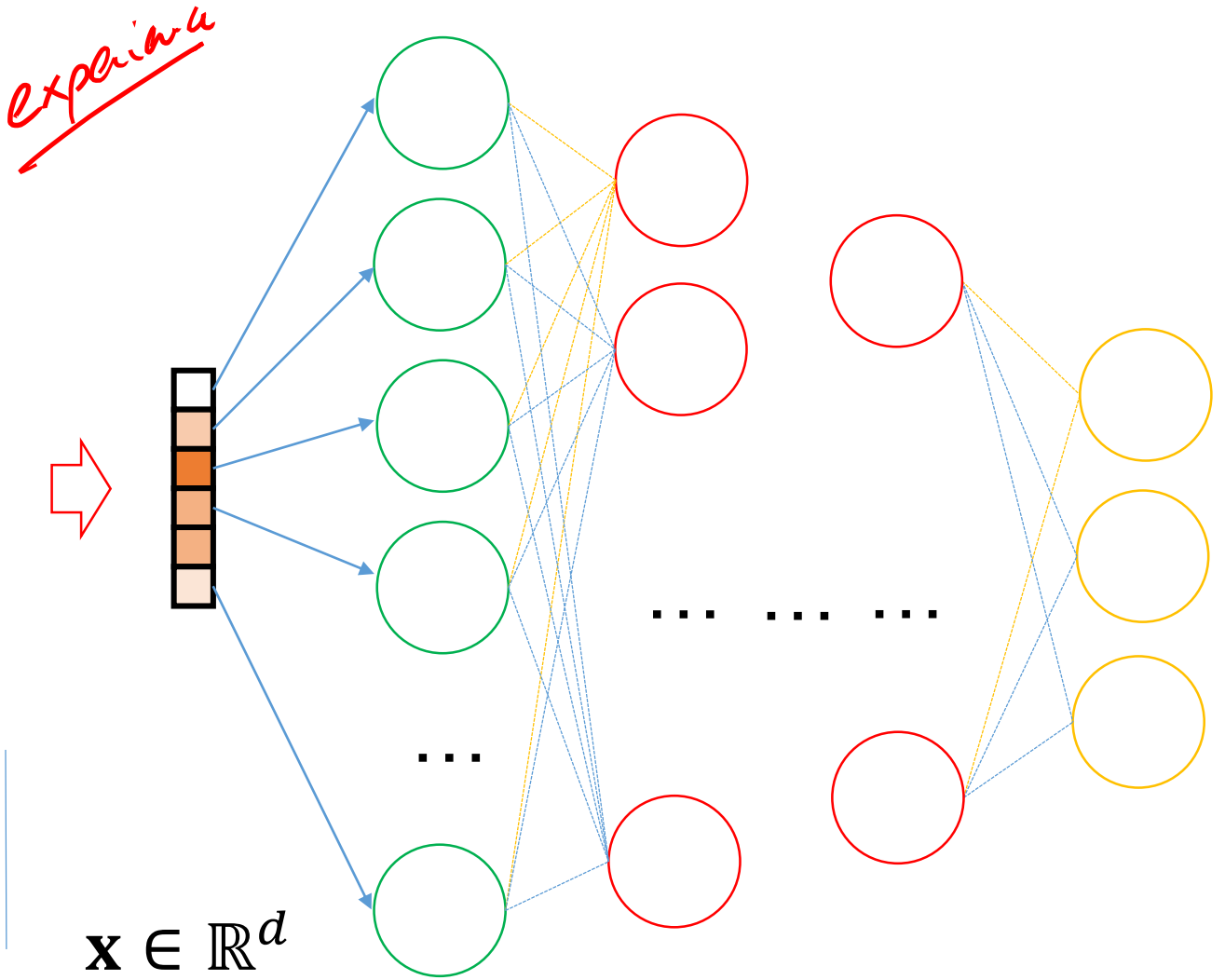
Neural Networks

Input image



$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction Algorithm



$\mathbf{x} \in \mathbb{R}^d$

input layer

Hidden layer(s)

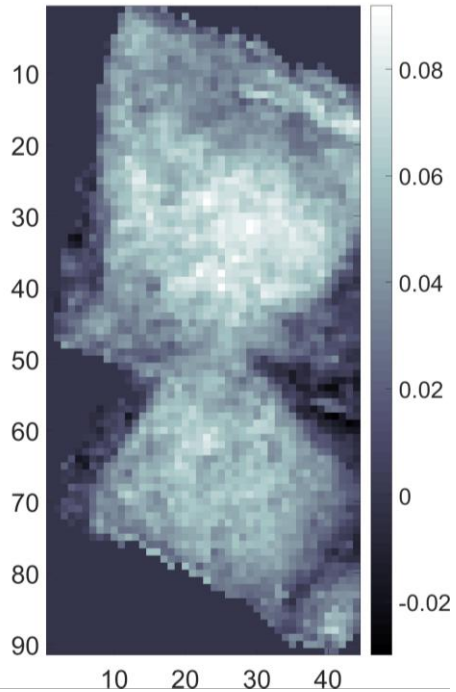
Output Layer

G. Boracchi

Neural Networks

Input layer: Same size of the feature vector

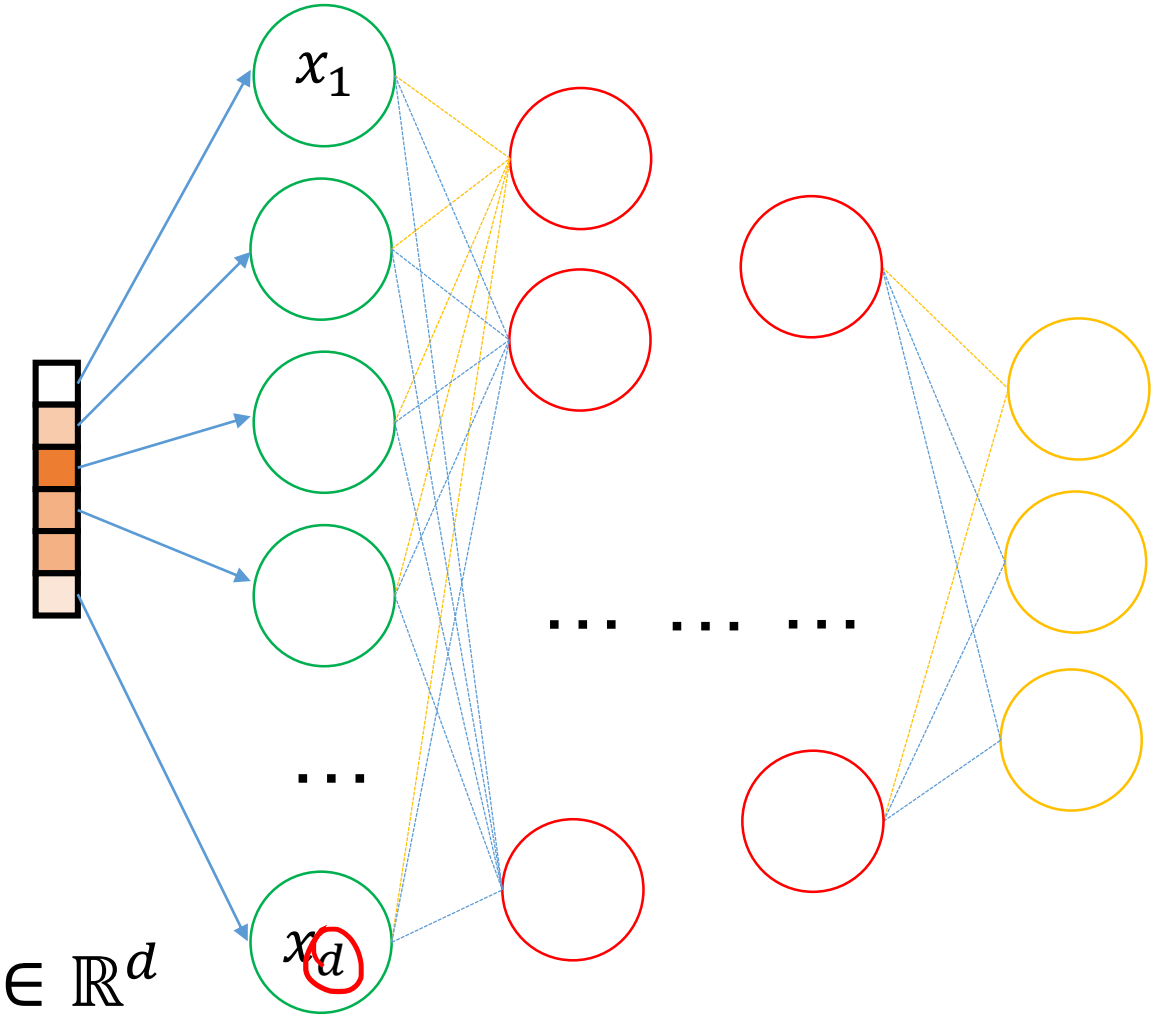
Input image



$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction Algorithm

$\mathbf{x} \in \mathbb{R}^d$



input layer

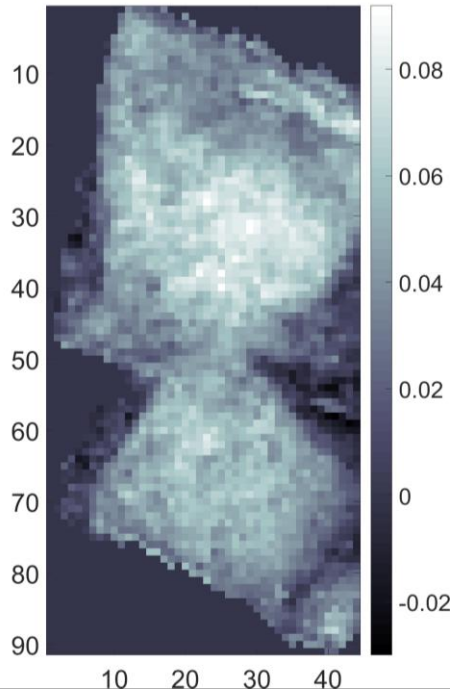
Hidden layer(s)

Output Layer

Neural Networks

Output layer: Same size as the number of classes

Input image



$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction Algorithm

$\mathbf{x} \in \mathbb{R}^d$

$P(l = \text{"doub."} | \boldsymbol{x})$

$P(l = \text{"env."} | \boldsymbol{x})$

$P(l = \text{"parc."} | \boldsymbol{x})$

input layer

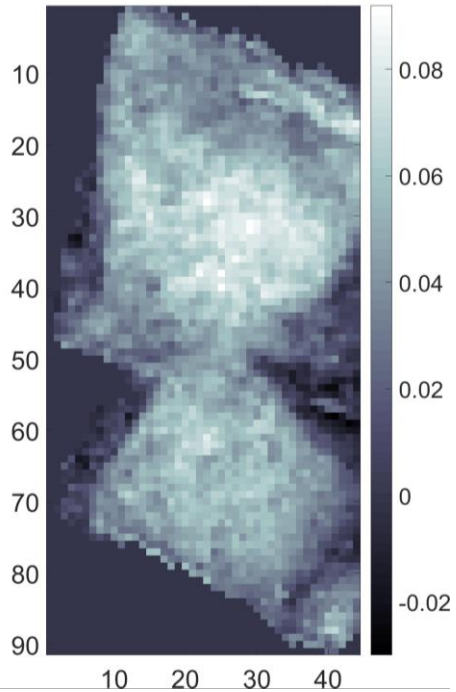
Hidden layer(s)

Output Layer

Neural Networks

Hidden layers: arbitrary size

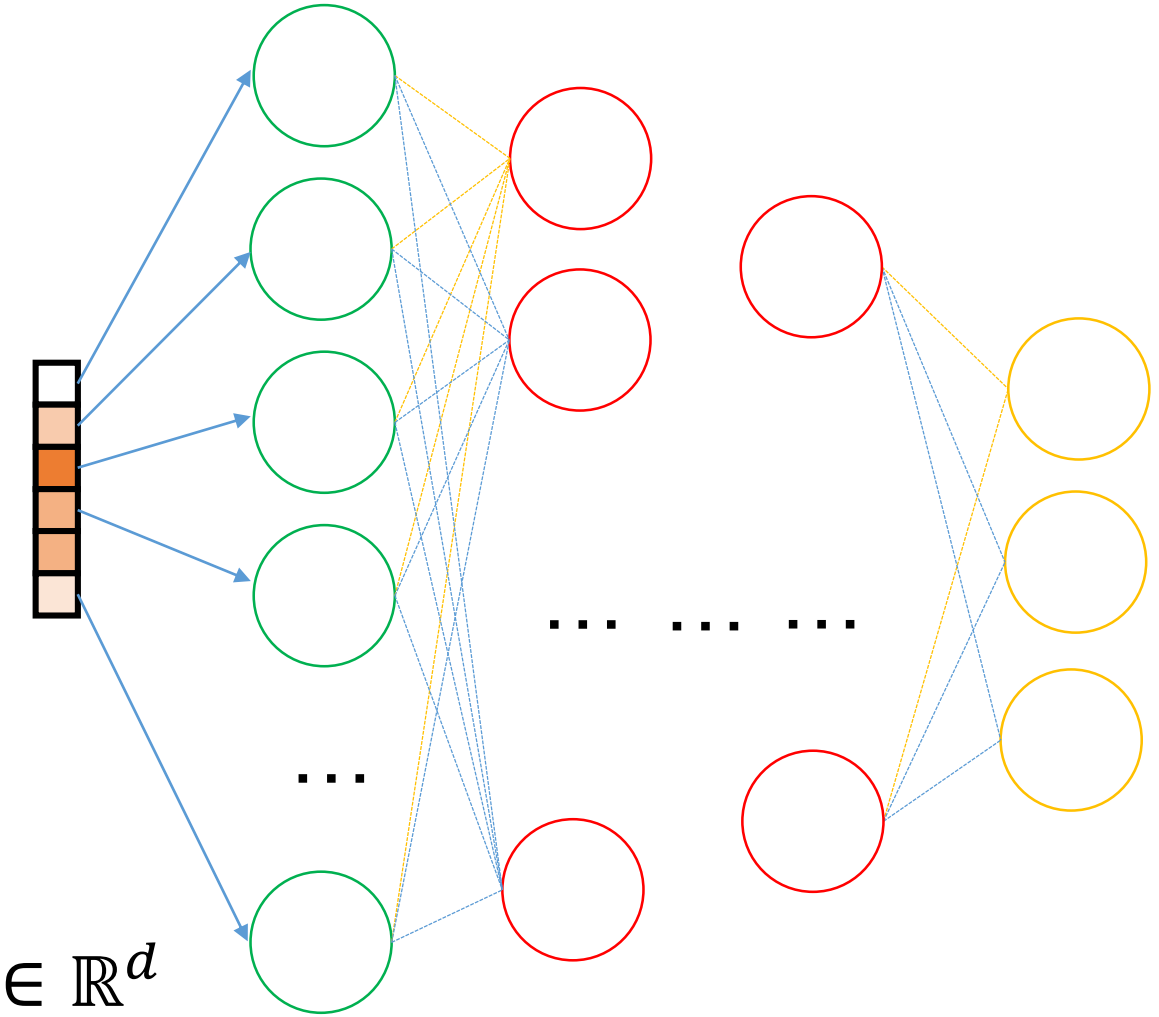
Input image



$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction Algorithm

$\mathbf{x} \in \mathbb{R}^d$



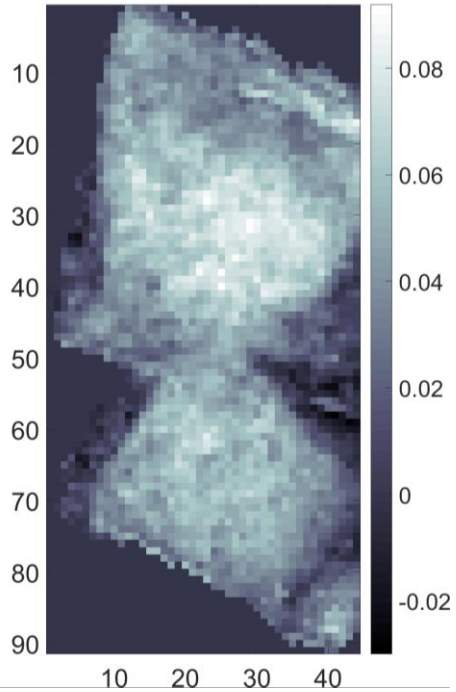
input layer

Hidden layer(s)

Output Layer

Neural Networks

Input image



$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction Algorithm

$\mathbf{x} \in \mathbb{R}^d$

$w^1{}_{1,1}$

$w^1{}_{1,2}$

$o_i = \tanh\left(\sum_{j=1:d} w^1{}_{i,j}\, x_j + b_i\right)$

x_1

x_d

o_1

$\boldsymbol{o} \in \mathbb{R}^p$

input layer

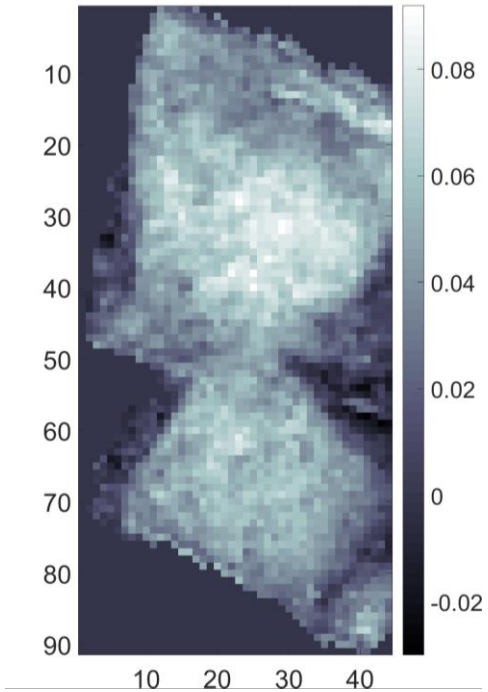
Hidden layer(s)

Output Layer

G. Boracchi

Image Classification by Hand Crafted Features

Input image



$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction Algorithm

Feature vector

Hand Crafted

Data Driven

G. Boracchi

Hand Crafted Featues, pros:

- **Exploit a priori** / expert **information**
- Features are **interpretable** (you might understand why they are not working)
- You can **adjust features** to improve your performance
- **Limited amount of training data** needed
- You can give more relevance to some features

Hand Crafted Featues, cons:

- Requires a lot of **design/programming efforts**
- **Not viable** in many **visual recognition** tasks (e.g. on natural images) which are easily performed by humans
- **Risk of overfitting** the training set used in the design
- **Not very general and "portable"**

Data-Driven Features

… the advent of deep learning

Data-Driven Features

Input image

$I_1 \in \mathbb{R}^{r_1 \times c_1}$

Feature Extraction

Data Driven

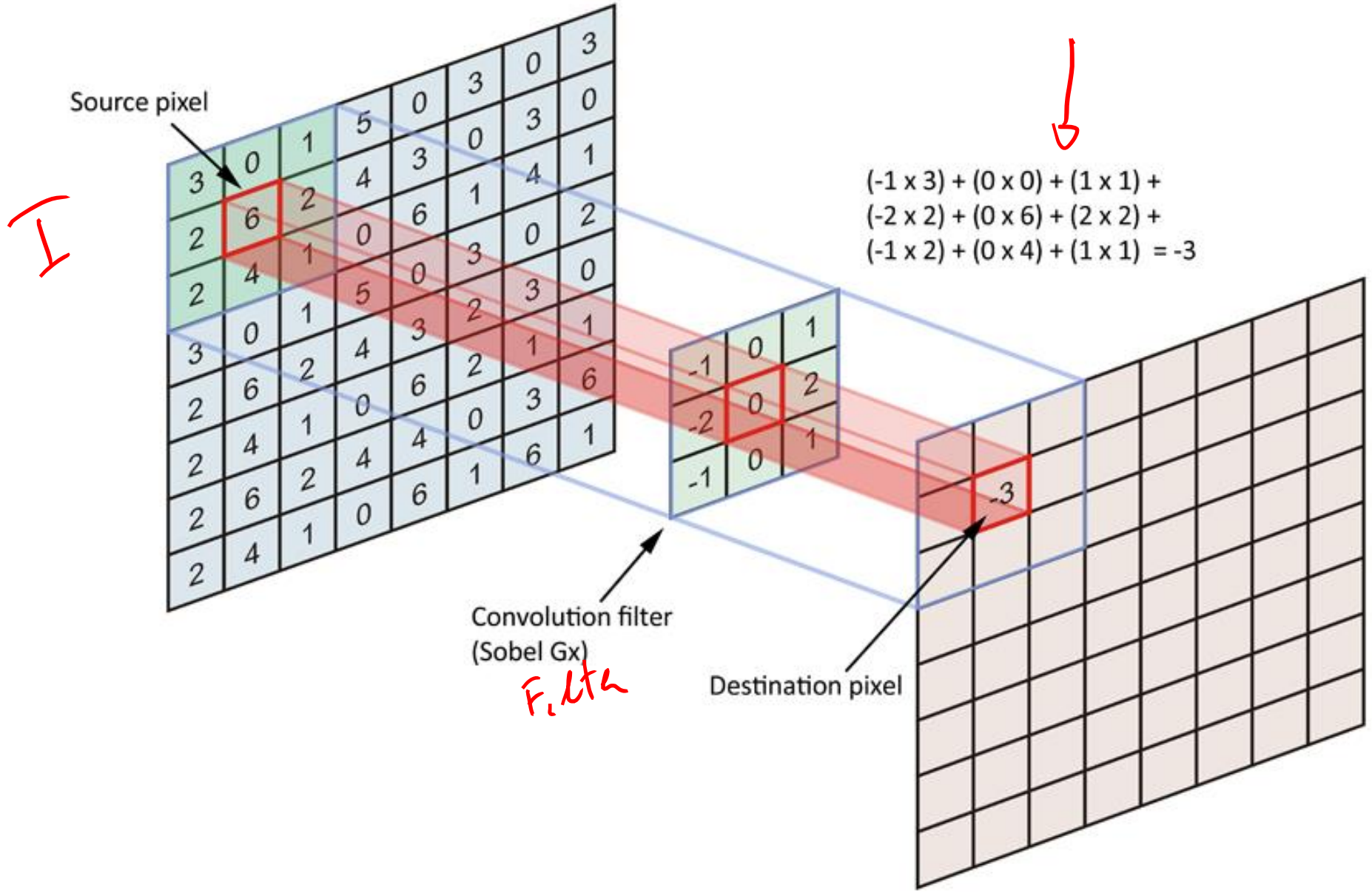
Data Driven

Setting up the stage

Convolutional neural networks

G. Boracchi

What is a convolution?



2D Correlation

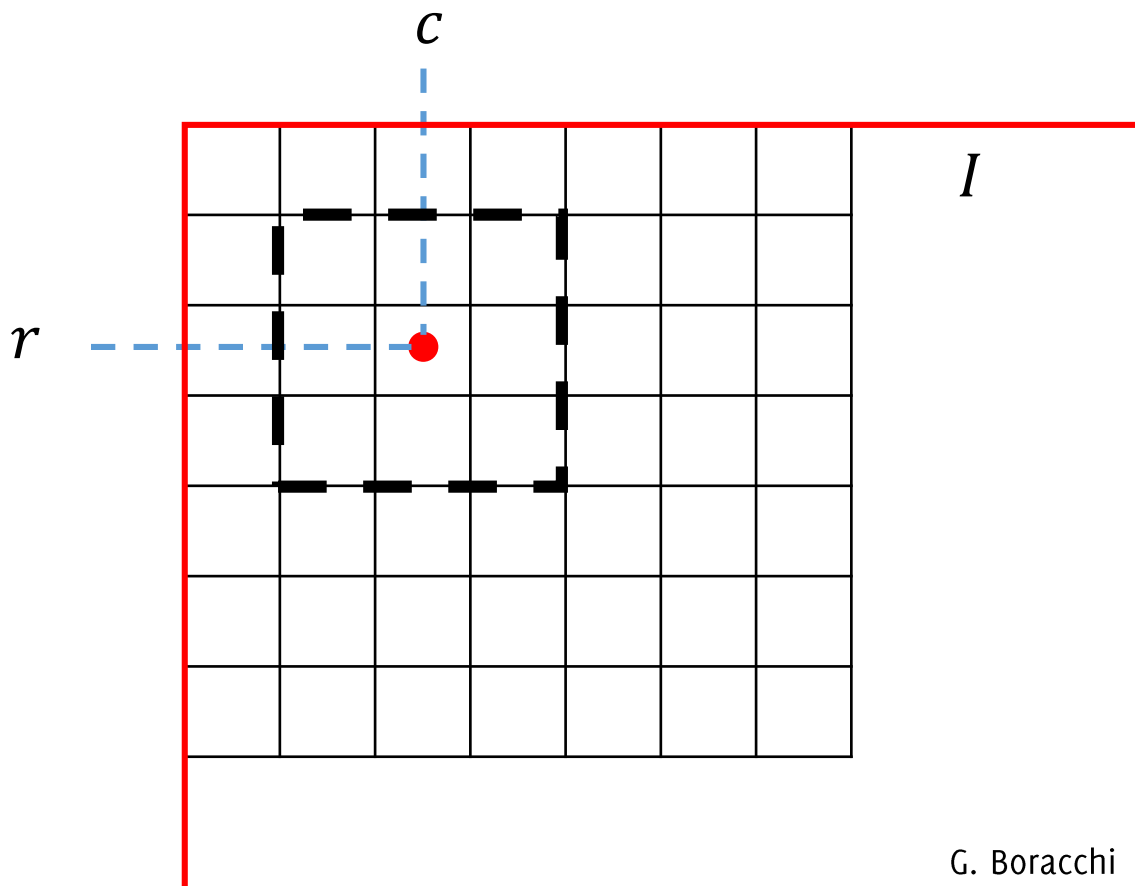
Convolution is a linear transformation. Linearity implies that

$$T[I](r,c) = \sum_{(x,y)\in U} w(x,y) * I(r+x, c+y)$$

filter

U

$w(-1,-1)$	$w(-1,0)$	$w(-1,1)$
$w(0,-1)$	$w(0,0)$	$w(0,1)$
$w(1,-11)$	$w(1,0)$	$w(1,1)$



We can consider weights as **a filter** h

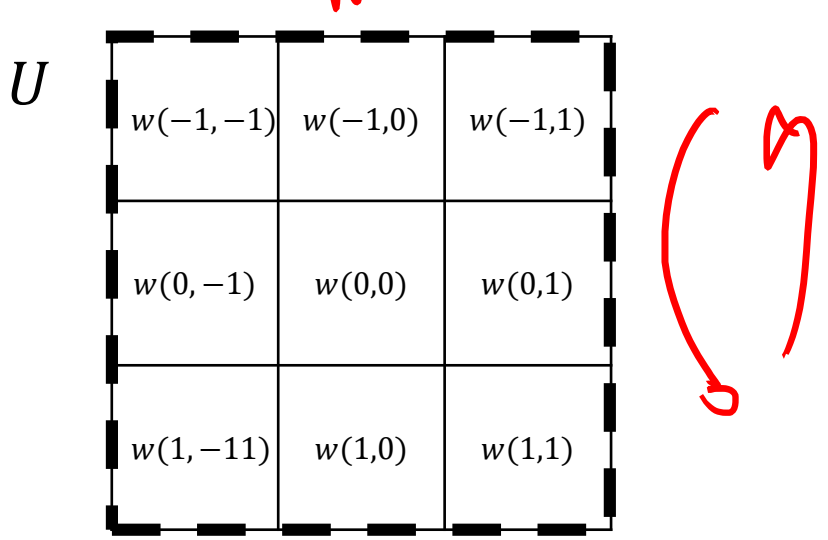
The filter h **entirely defines convolution**

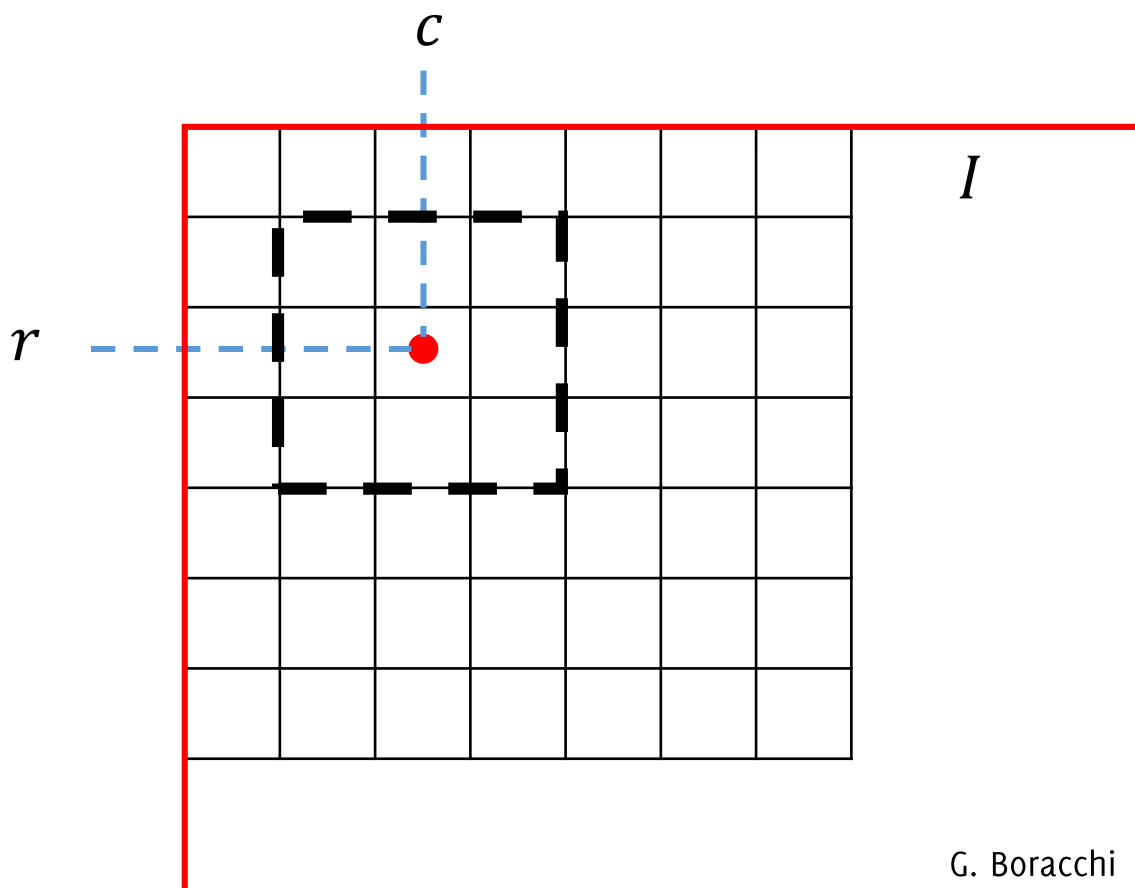
Convolution operates the same in each pixel

2D Convolution

Convolution is a linear transformation. Linearity implies that

$$T[I](r,c) = \sum_{(x,y)\in U} w(x,y) * I(r-x, c-y)$$





We can consider weights as **a filter** h

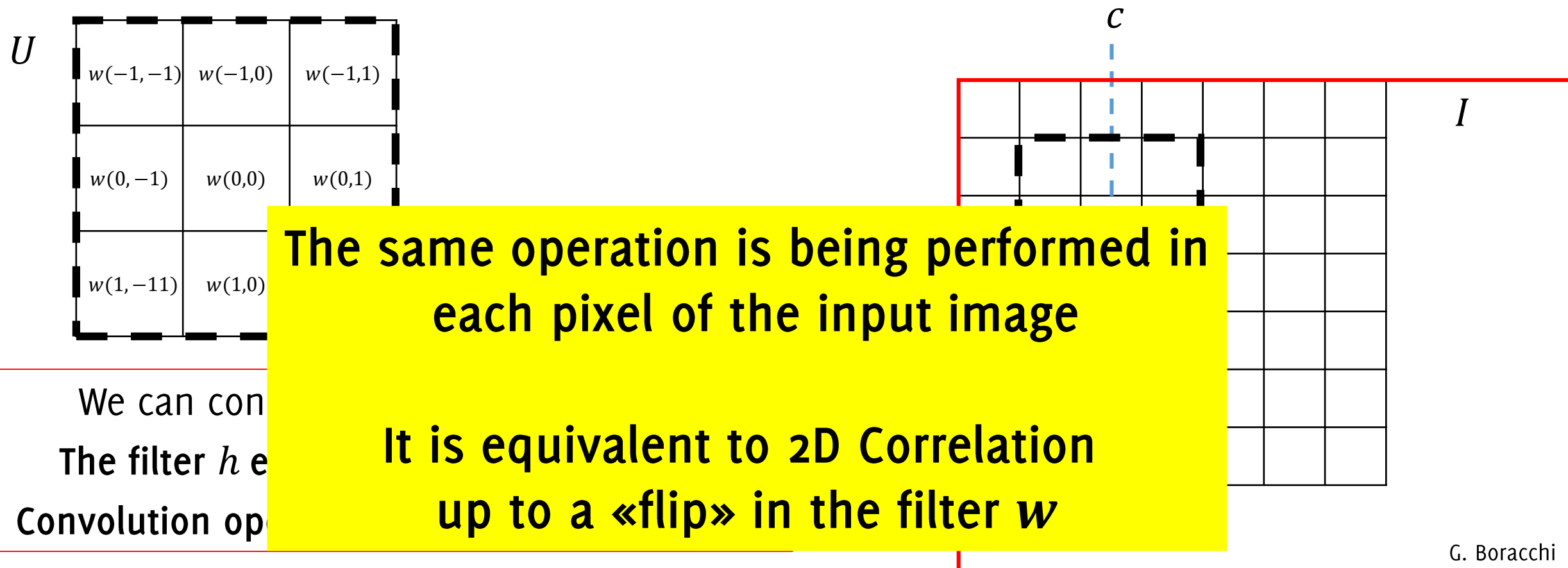
The filter h entirely defines convolution

Convolution operates the same in each pixel

2D Convolution

Convolution is a linear transformation. Linearity implies that

$$T[I](r,c) = \sum_{(x,y)\in U} w(x,y) * I(r-x, c-y)$$



We can con

The filter h e

Convolution op

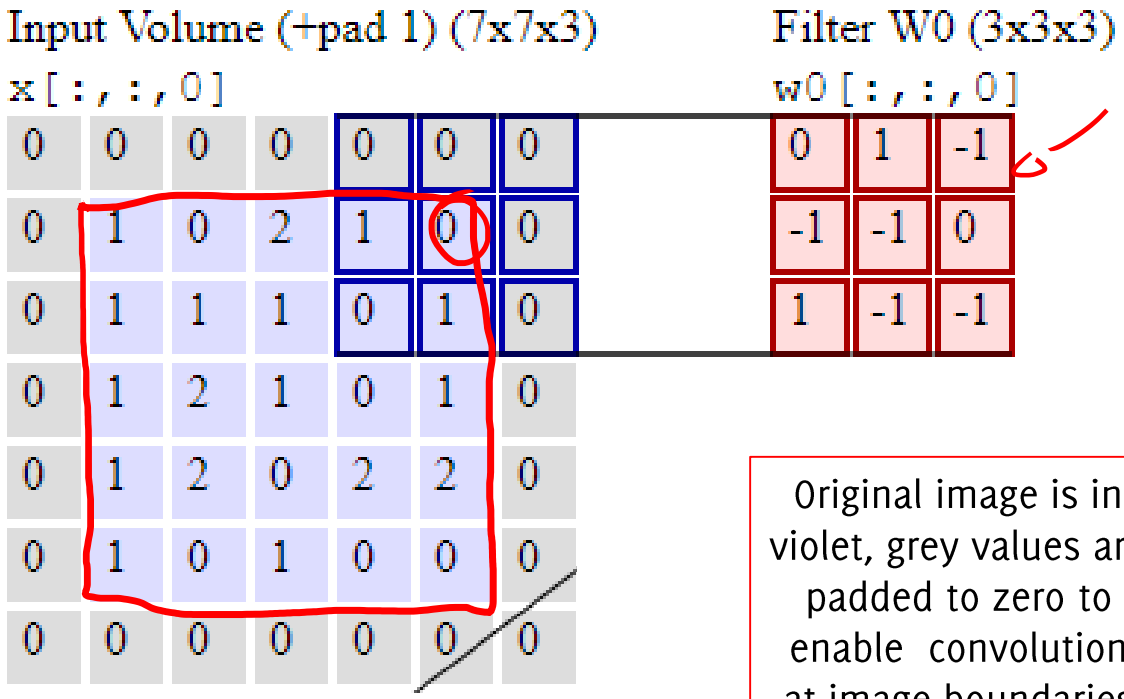
The same operation is being performed in each pixel of the input image

It is equivalent to 2D Correlation up to a «flip» in the filter w

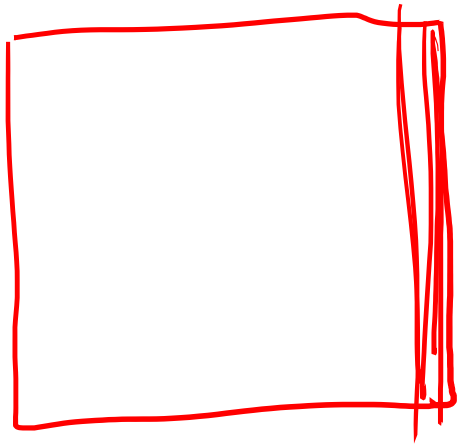
Convolution: Padding

How to define convolution output close to image boundaries?

Padding with zero is the most frequent option, as this does not change the output size. However, no padding or symmetric padding are also viable options

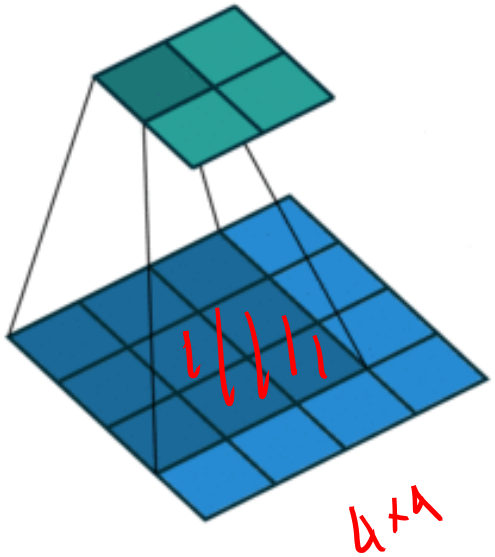


Original image is in violet, grey values are padded to zero to enable convolution at image boundaries

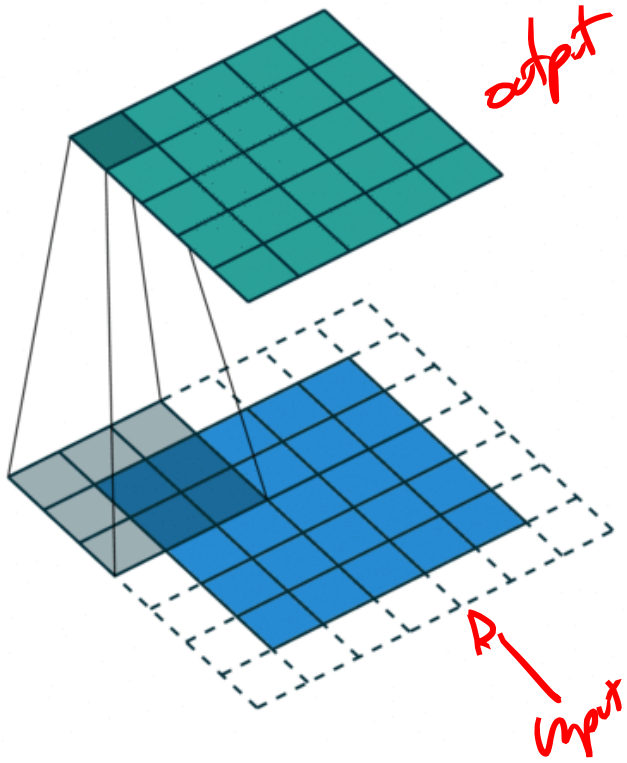


Convolution Animation

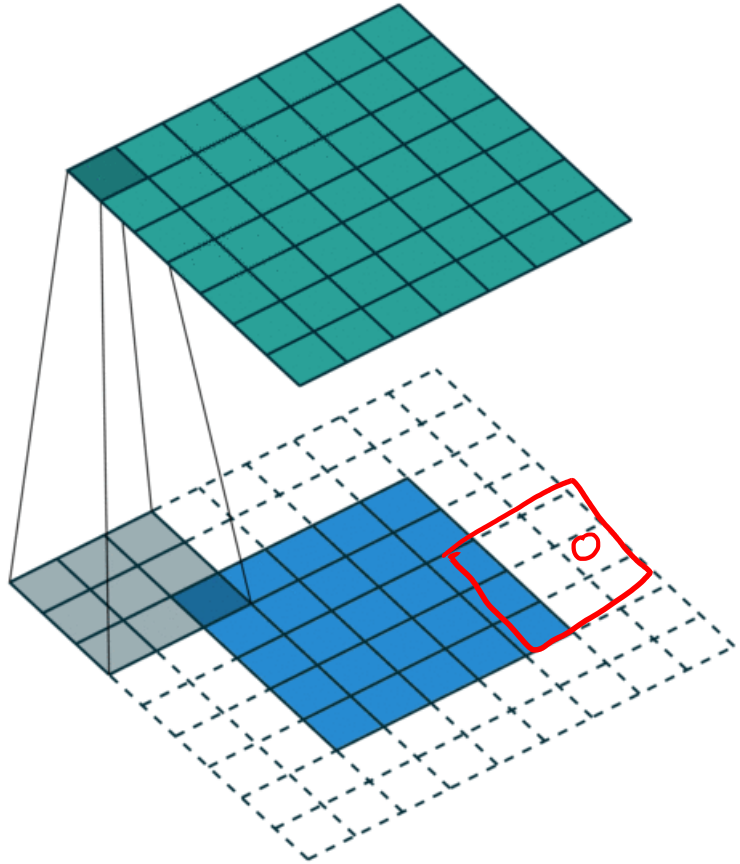
N.B.: Blue maps are inputs, and cyan maps are outputs.



No padding
«valid»



Half padding
«same»



full padding
«full»

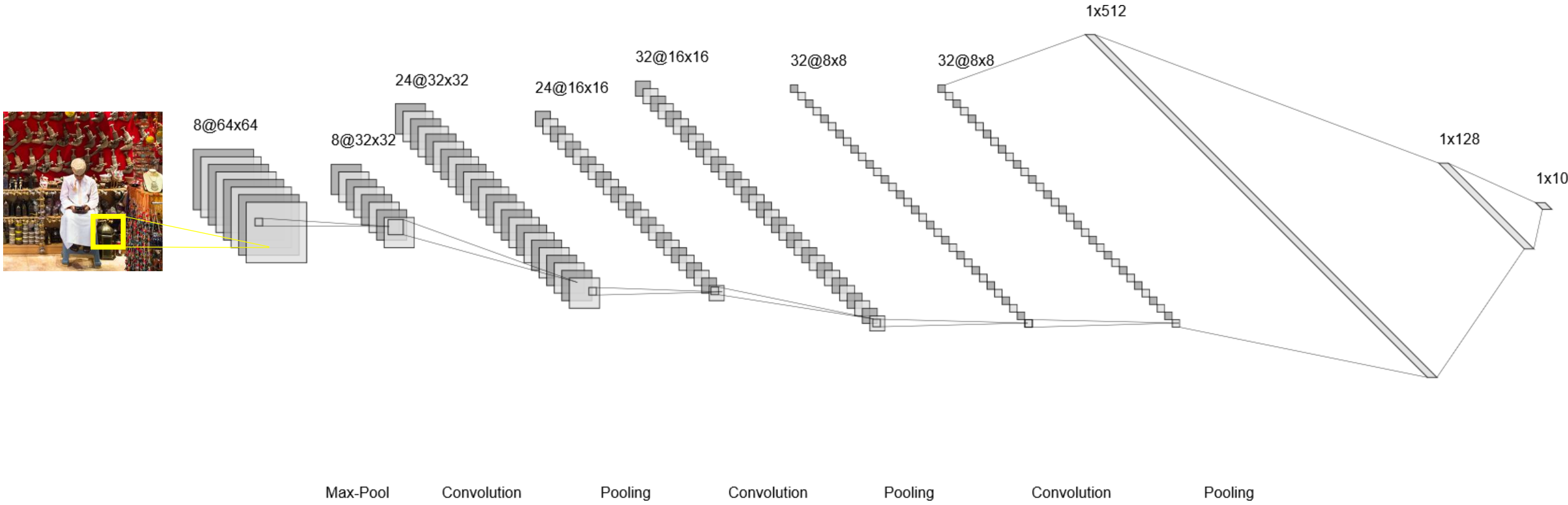
Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolutional Neural Networks

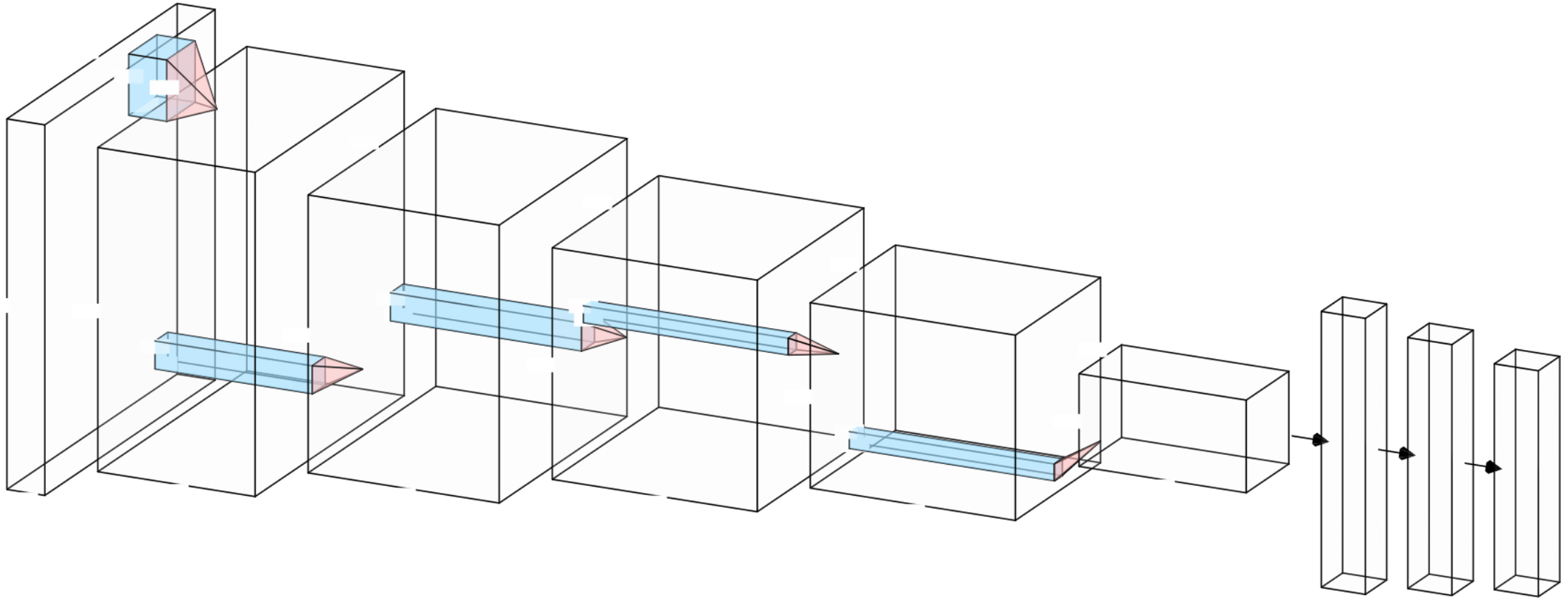
CNNs

G. Boracchi

The typical architecture of a CNN



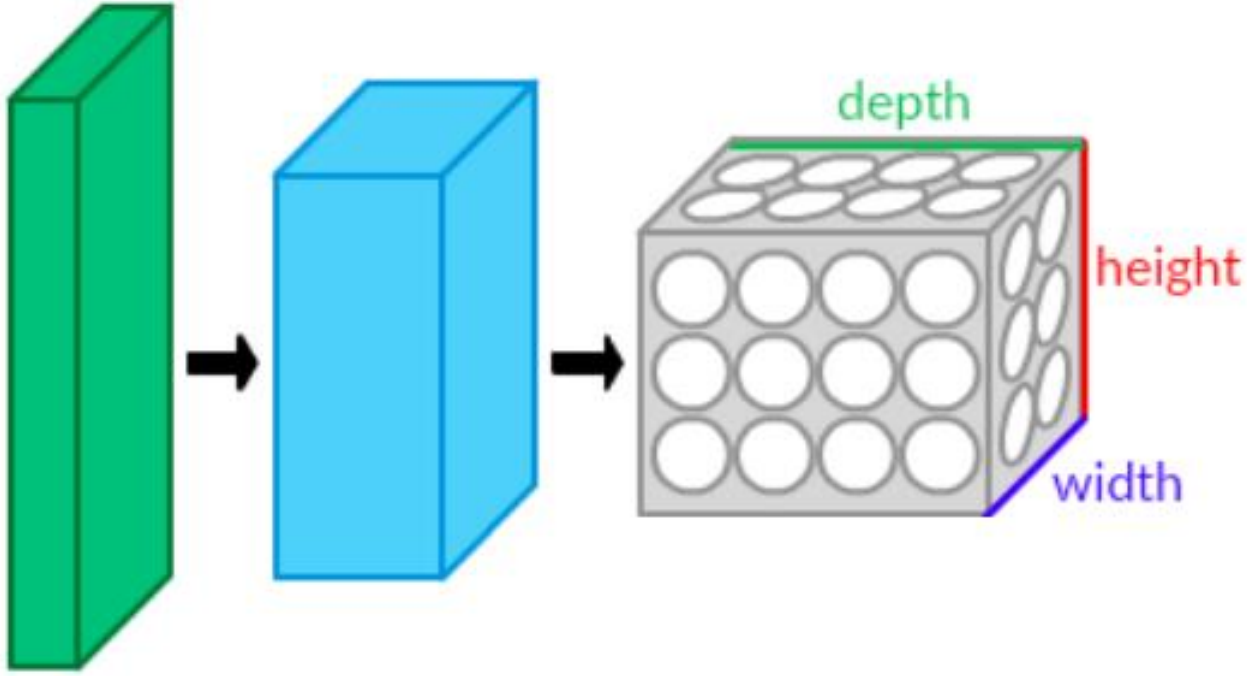
The typical architecture of a CNN



Convolutional Neural Networks (CNN)

CNN are typically made of blocks that include:

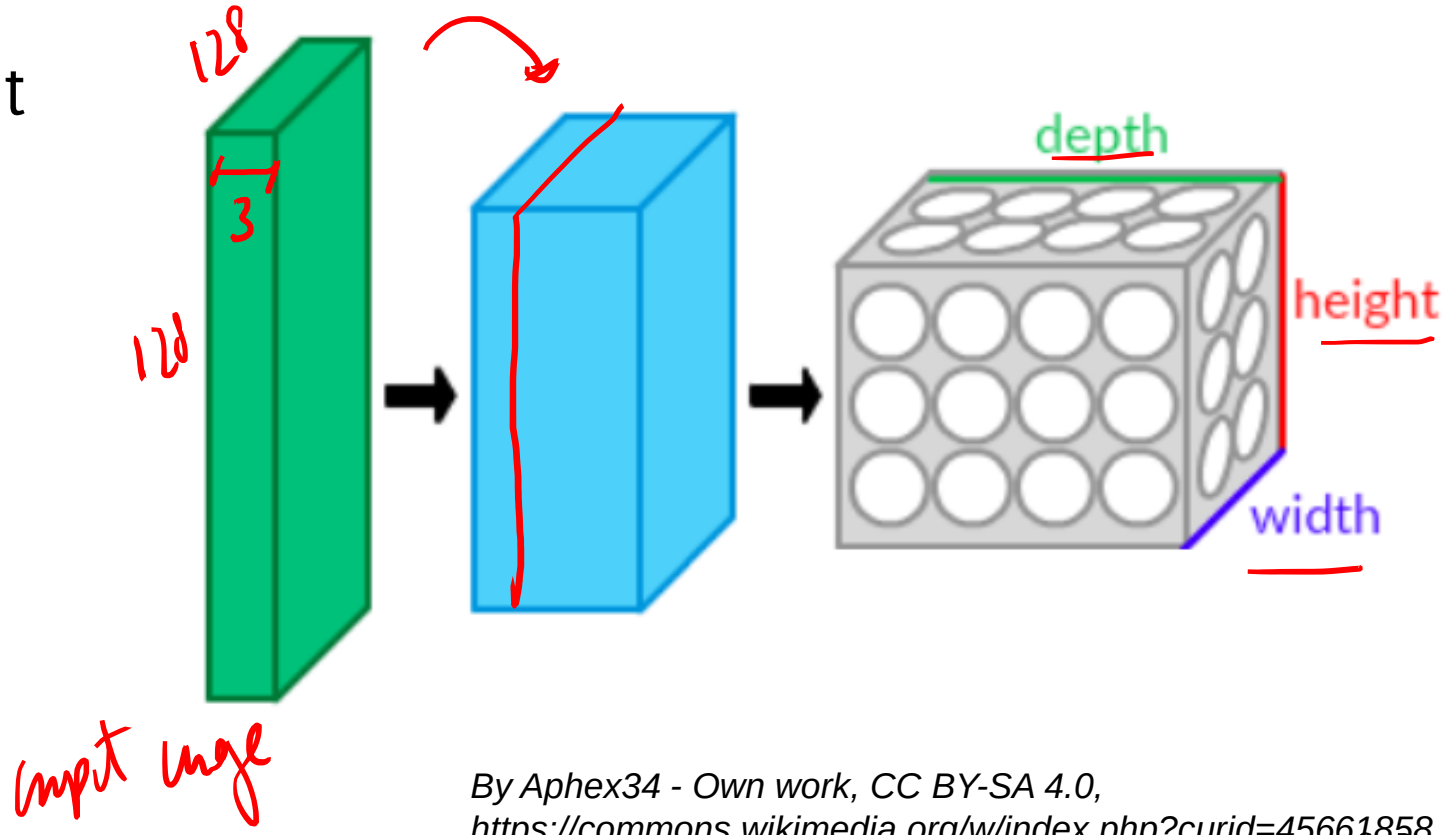
- Convolutional layers
- Nonlinearities (activation functions)
- Maxpooling



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45661858

Convolutional Neural Networks (CNN)

- An image passing through a CNN is transformed in a sequence of volumes.
- As the depth increases, the height and width of the volume decreases
- Each layer takes as input and returns a volume



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45661858

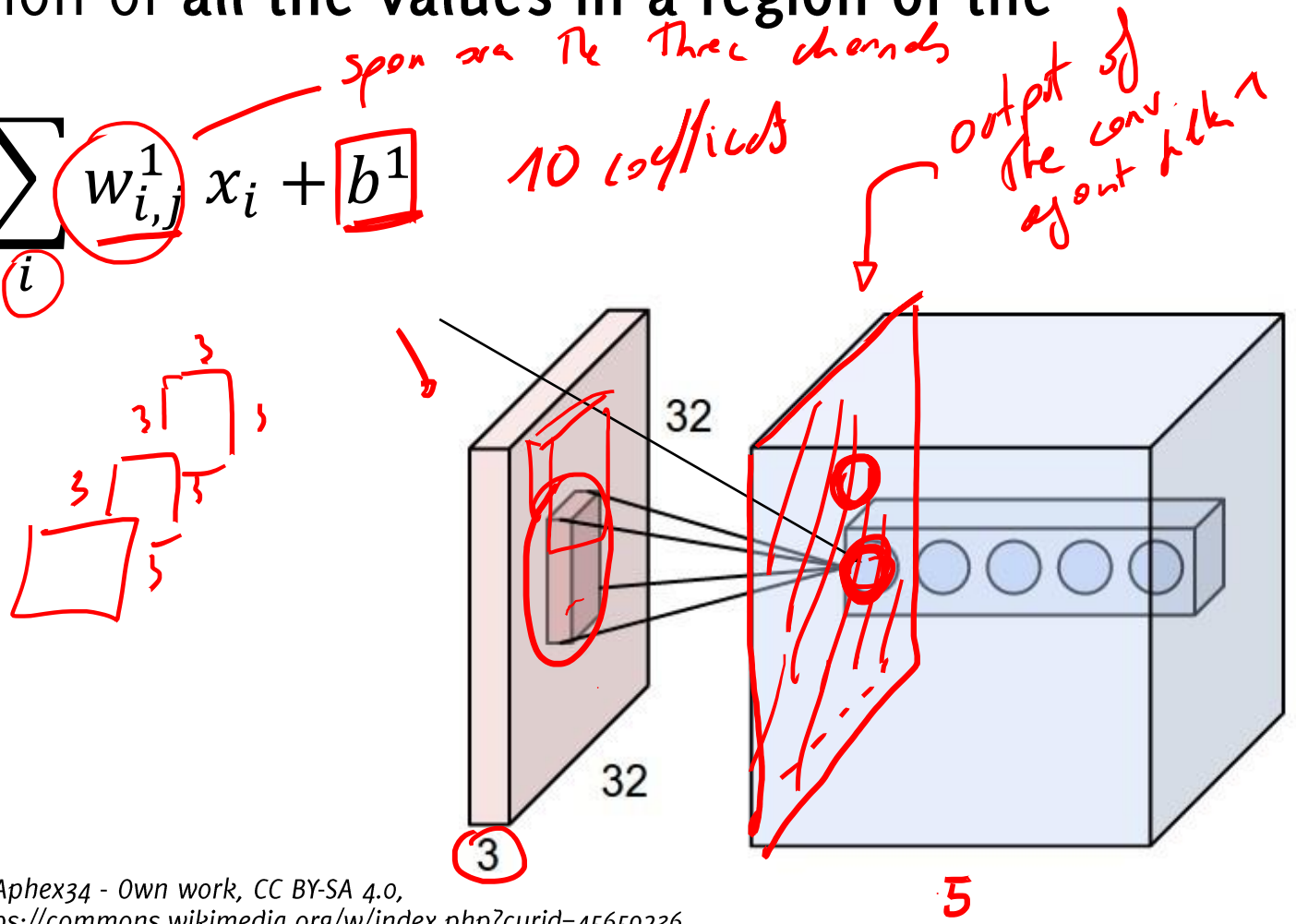
Convolutional Layers

Convolutional Layers

Convolutional layers "mix" all the input components

The output is a linear combination of **all the values in a region of the input**

$$y = \sum_i w_{i,j}^1 x_i + b^1$$



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

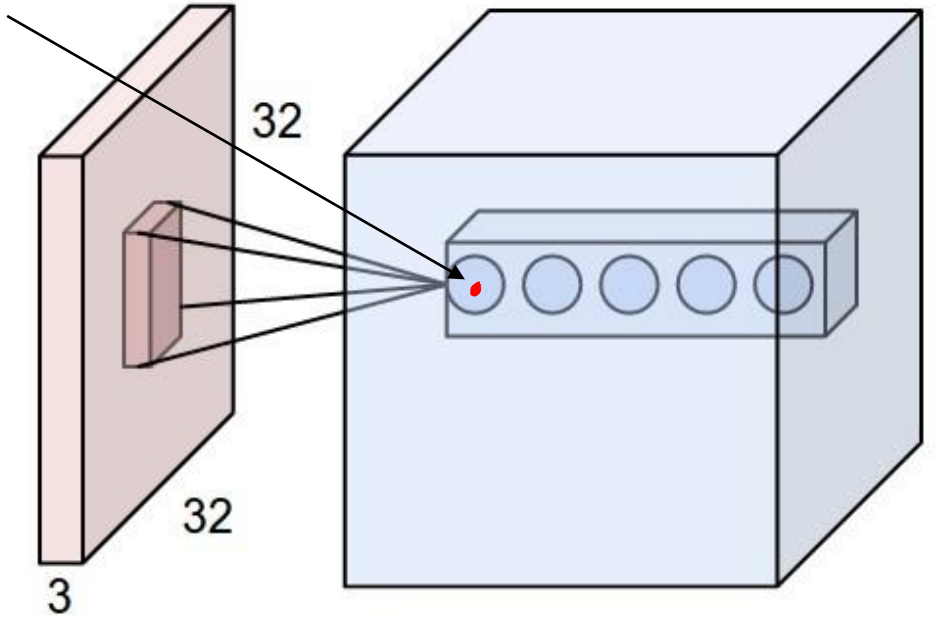
Convolutional Layers

Convolutional layers "mix" all the input components

The output is a linear combination of all the values in a region of the input

$$y = \sum_i w_{i,j}^1 x_i + b^1$$

The parameters of this layer are called filters.



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers

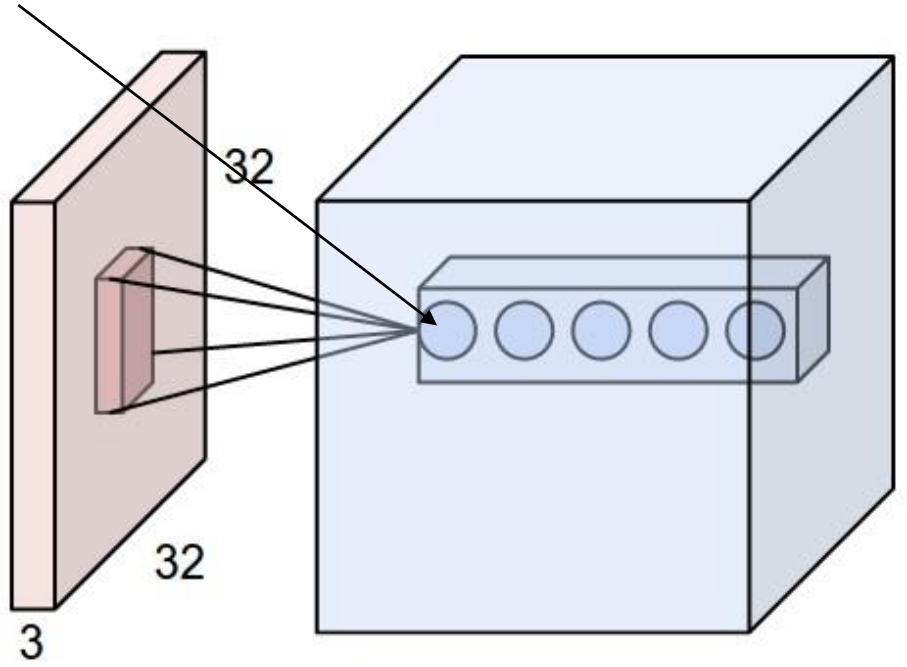
Convolutional layers "mix" all the input components

The output is a linear combination of all the values in a region of the input

$$y = \sum_i w_{i,j}^1 x_i + b^1$$

The parameters of this layer are called filters.

The same filter is used through the whole spatial extent of the input



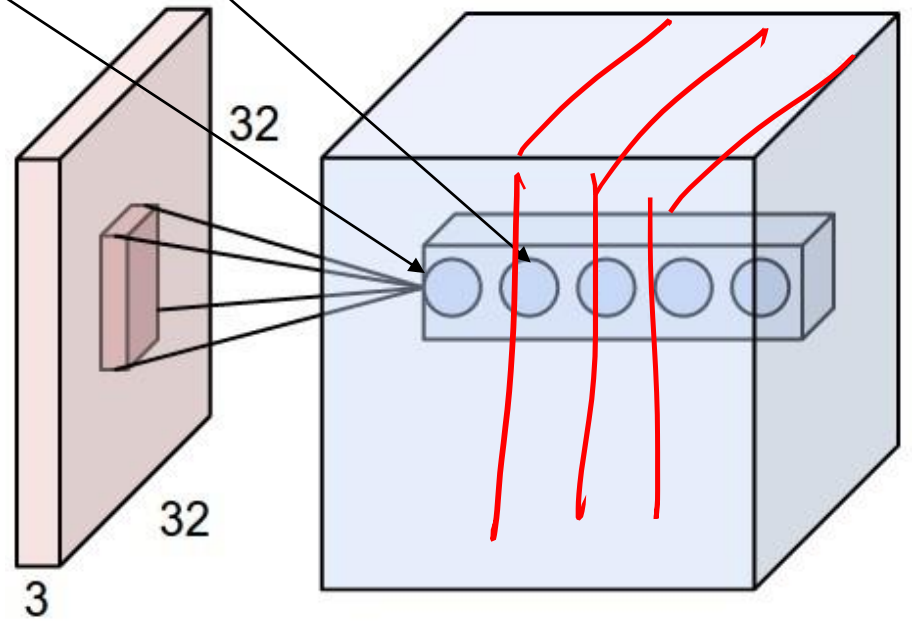
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers

Different filters yield different layers in the output

$$\sum_i w_{i,j}^1 \, x_i + b^1$$

$$\sum_i w_{i,j}^2 \, x_i + b^2$$

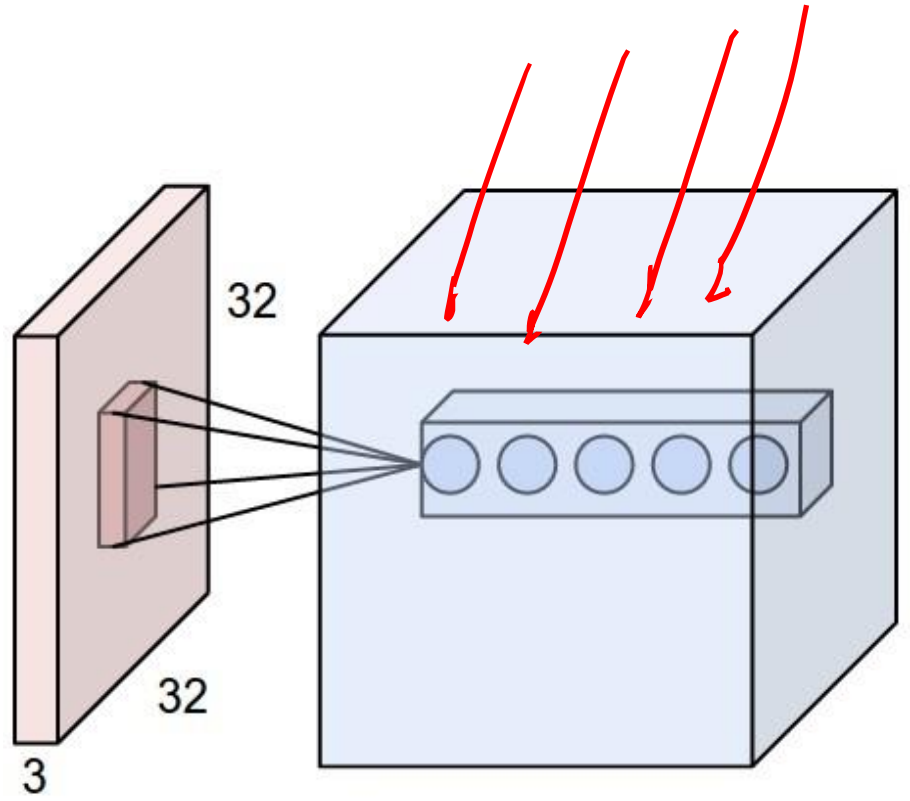


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers, recap

Convolutional layers "mix" all the input components

- The **output** is also called **volume** or **activation maps**
- Each **filter yields a different slice** of the output volume
- **Each filter has depth equal to the depth of the input volume**

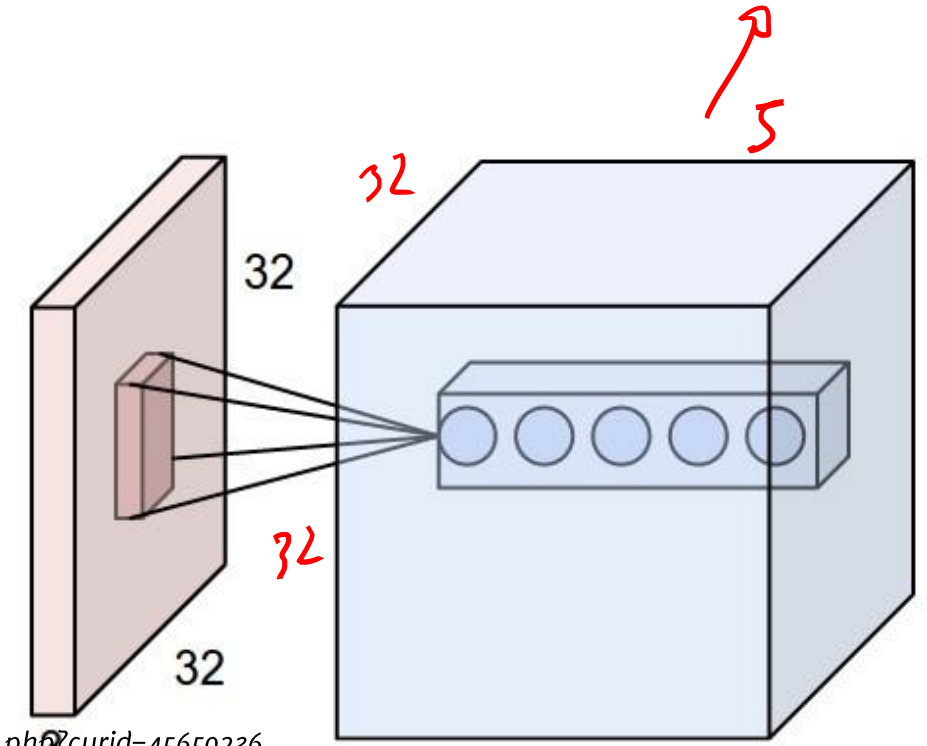


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

Convolutional Layers, remarks:

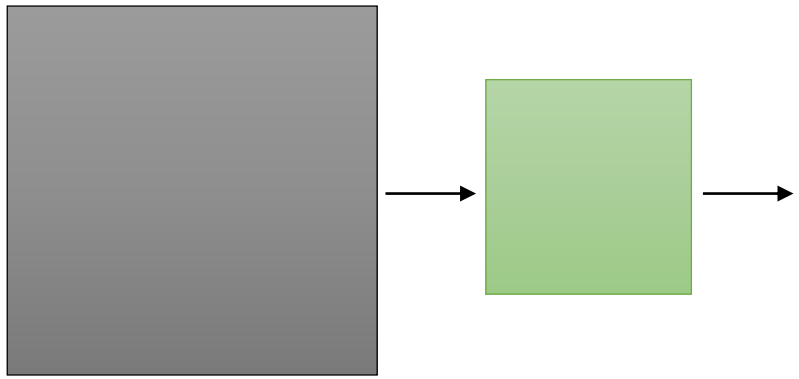
- Convolutional Layers are described by a set of **filters**
- **Filters represent the weights of these linear combination.**
- Filters have **very small spatial extent and large depth extent**,
- The **filter depth** is typically not specified, because it corresponds to the **number of layers of the input volume**

The output of the convolution
against a filter becomes a slice
in the volume feed to the next layer



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

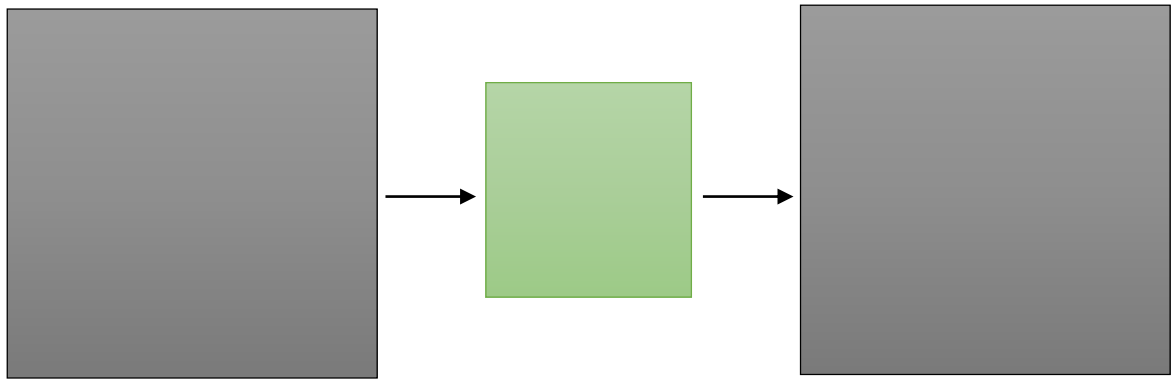
CNN Arithmetic



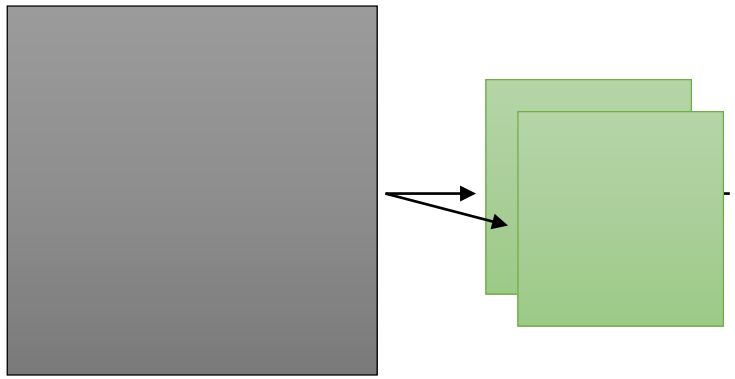
1 input map

1 filter 3x3

CNN Arithmetic



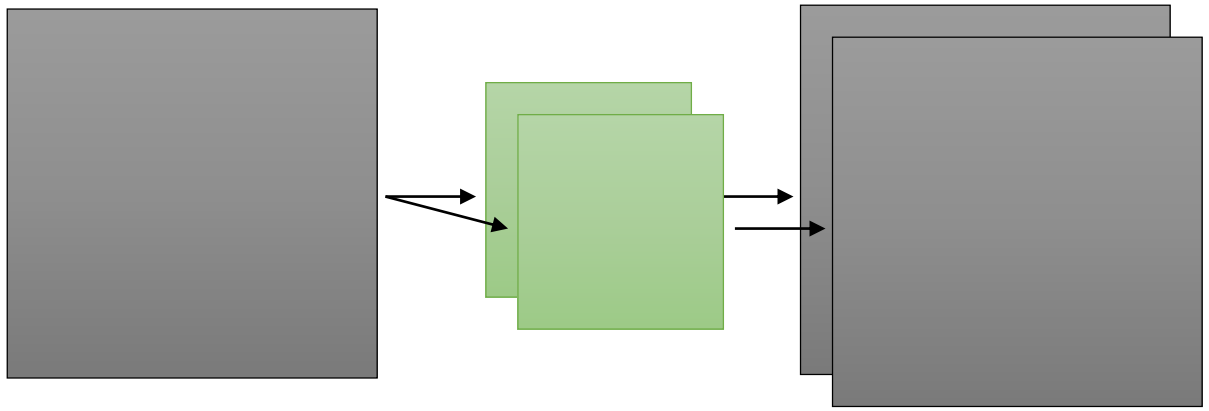
CNN Arithmetic



1 input map

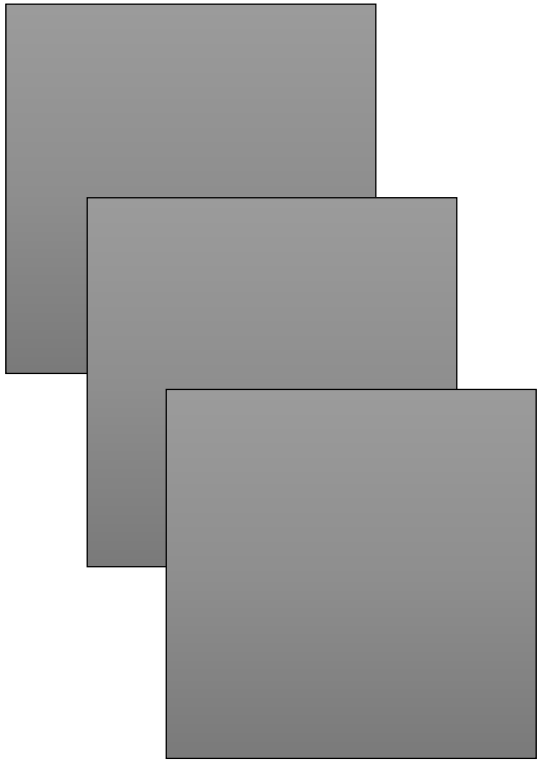
2 filters 3x3

CNN Arithmetic



1 input map 2 filters 3x3 2 output maps

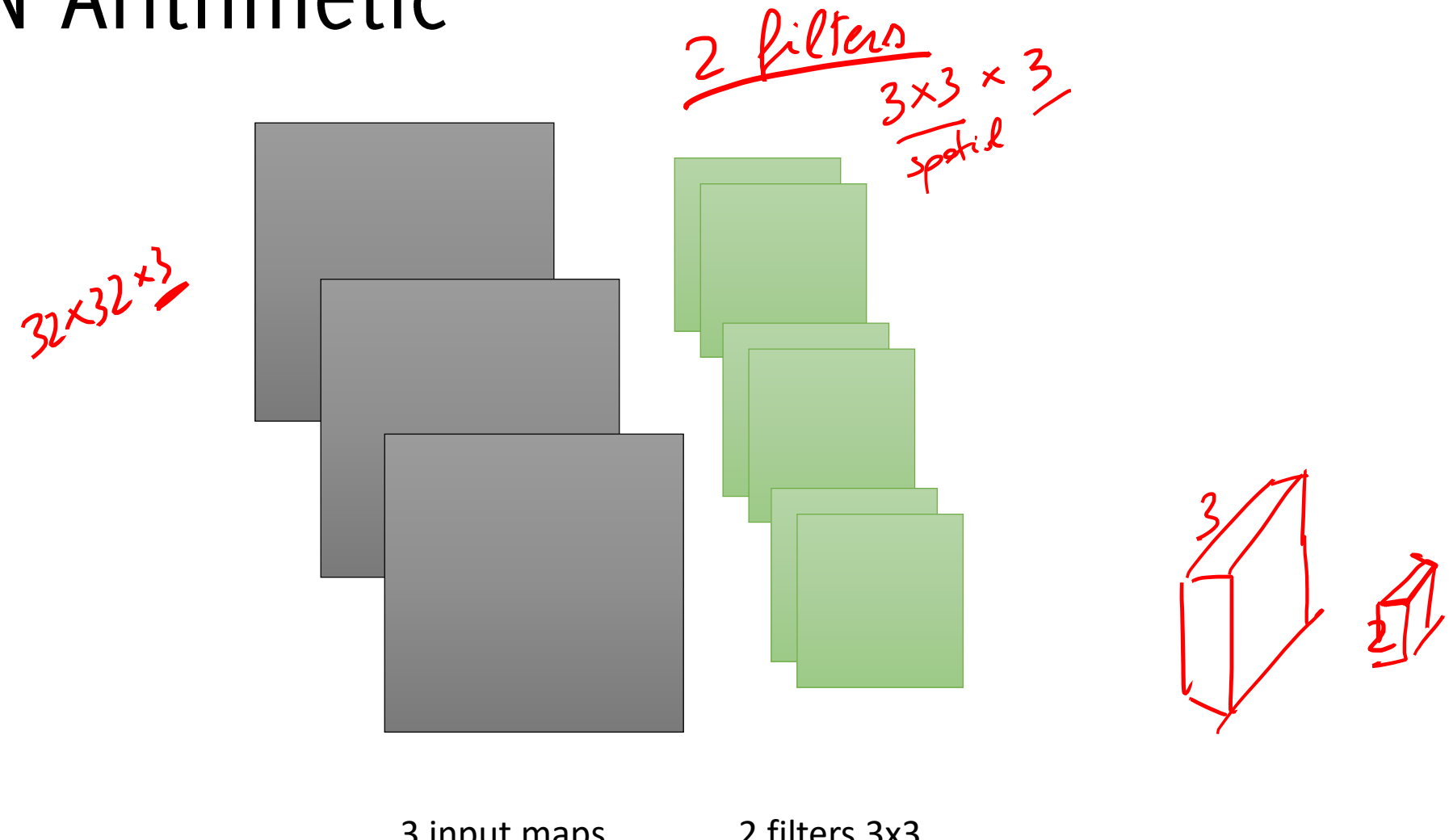
CNN Arithmetic



3 input maps

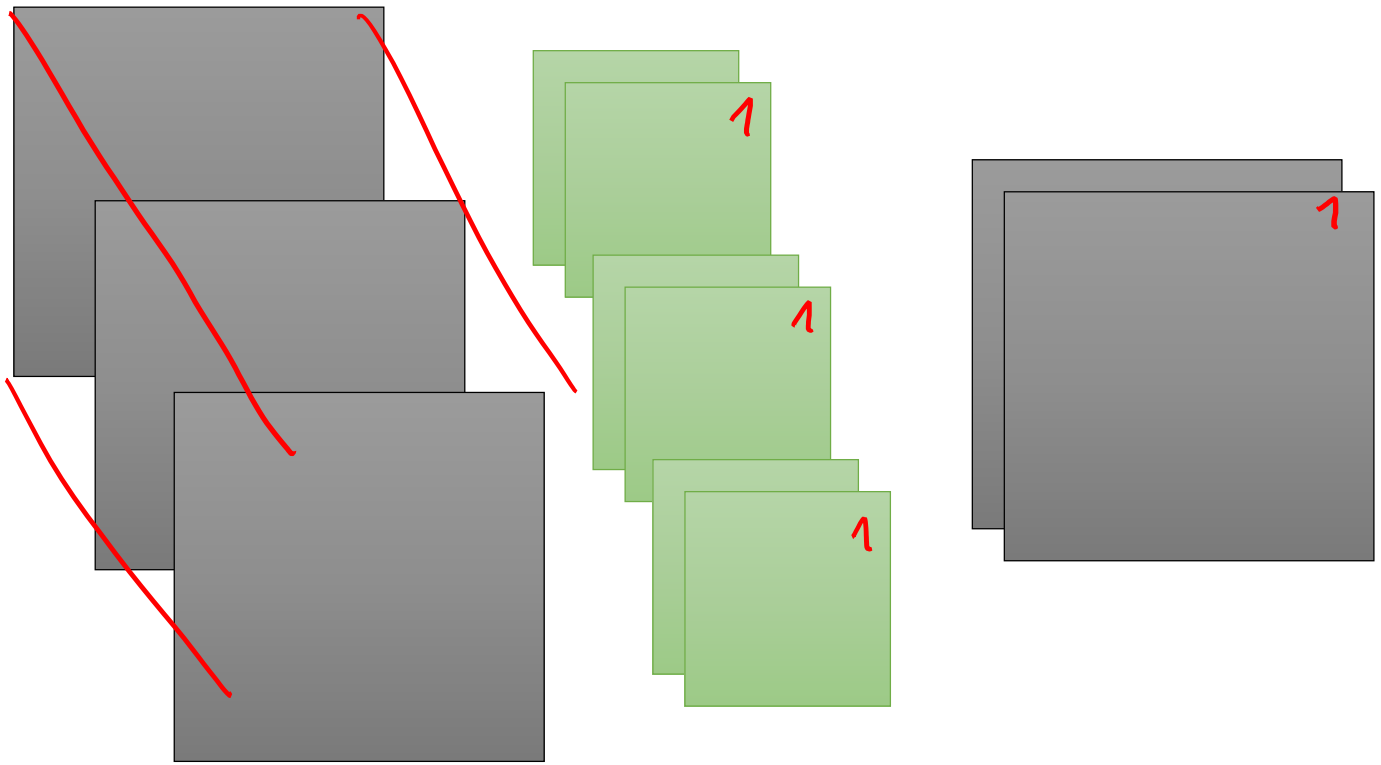
2 filters 3x3

CNN Arithmetic



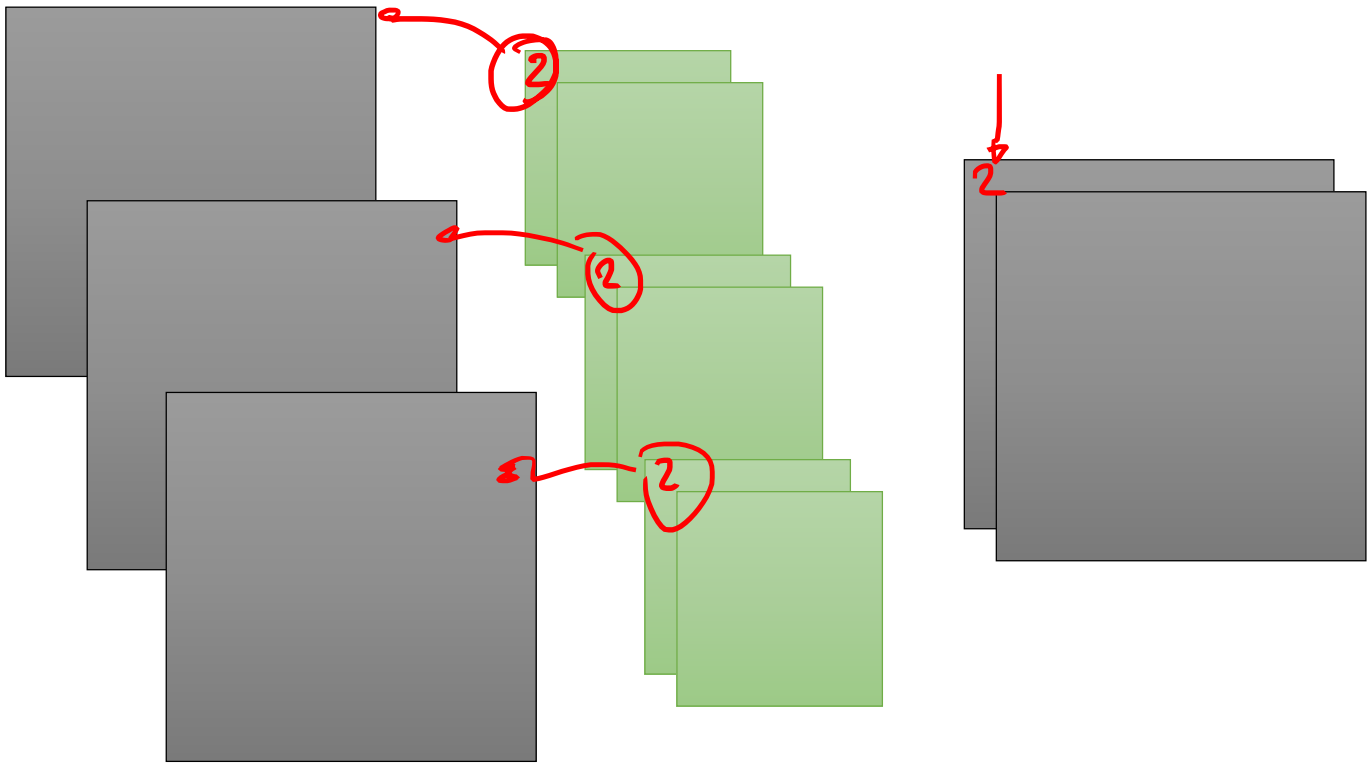
3 input maps 2 filters 3x3

CNN Arithmetic



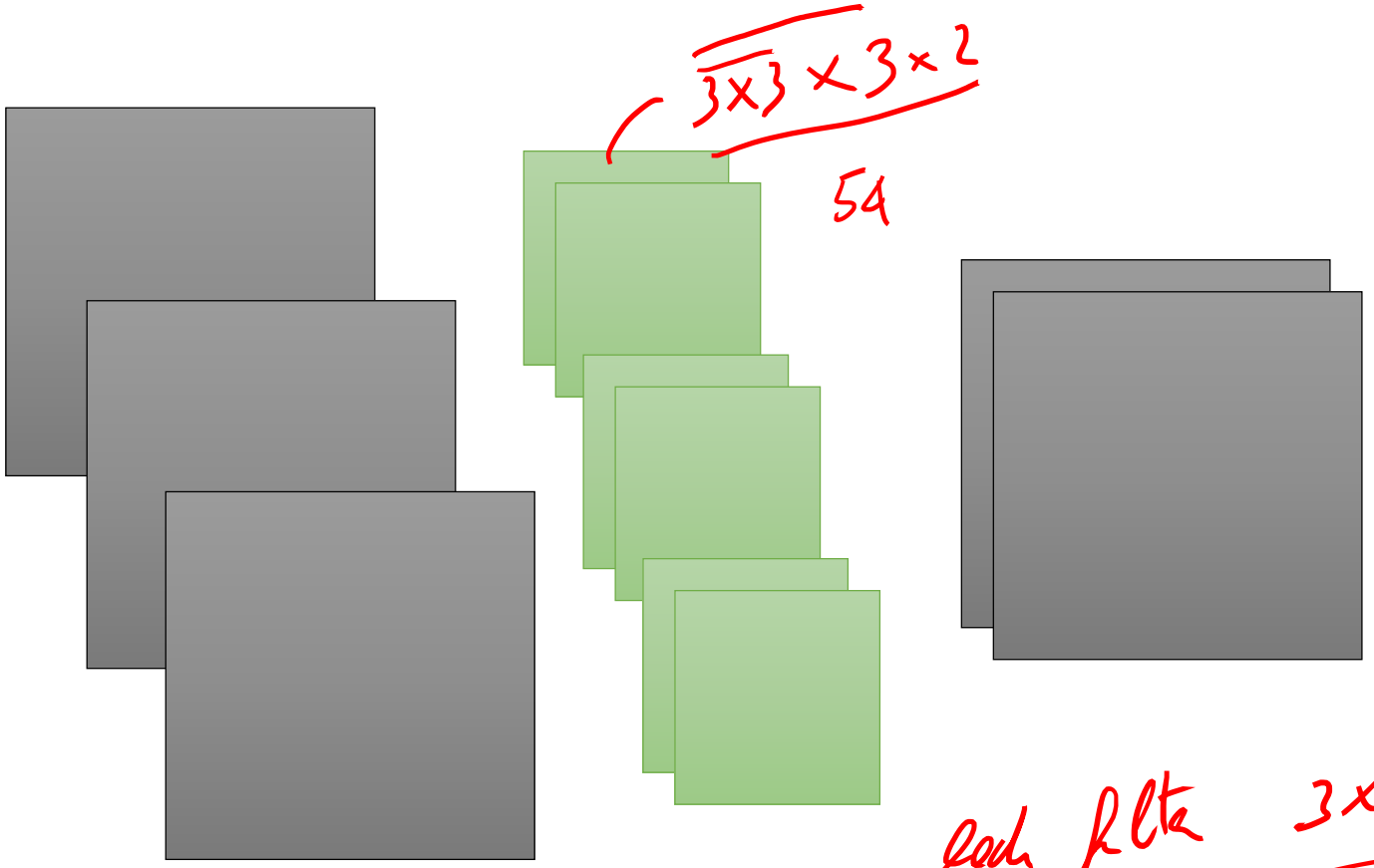
3 input maps 2 filters 3x3 2 output maps

CNN Arithmetic



3 input maps 2 filters 3x3 2 output maps

CNN Arithmetic



3 input maps

2 filters 3x3

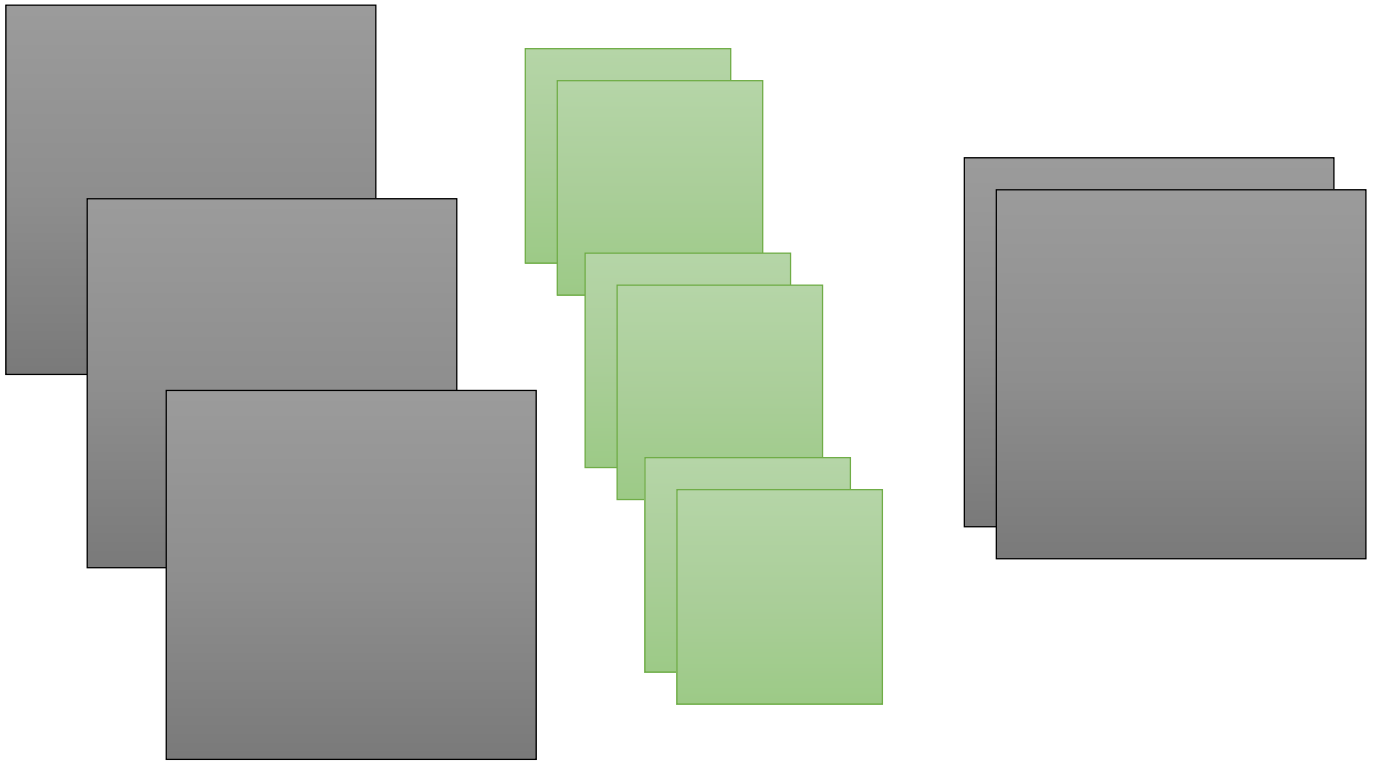
2 output maps

each filter 3x3x3 = 27 x 2 filters

54 + 2 biases

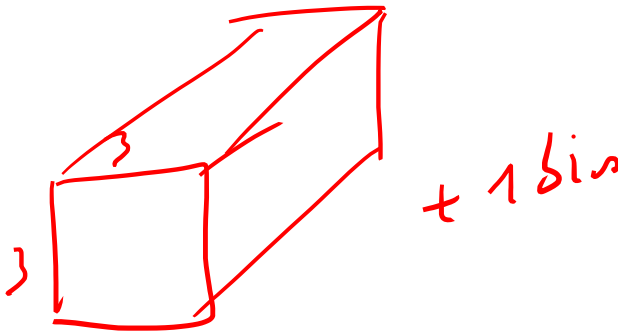
Quiz: how many parameters does this layer have?

CNN Arithmetic

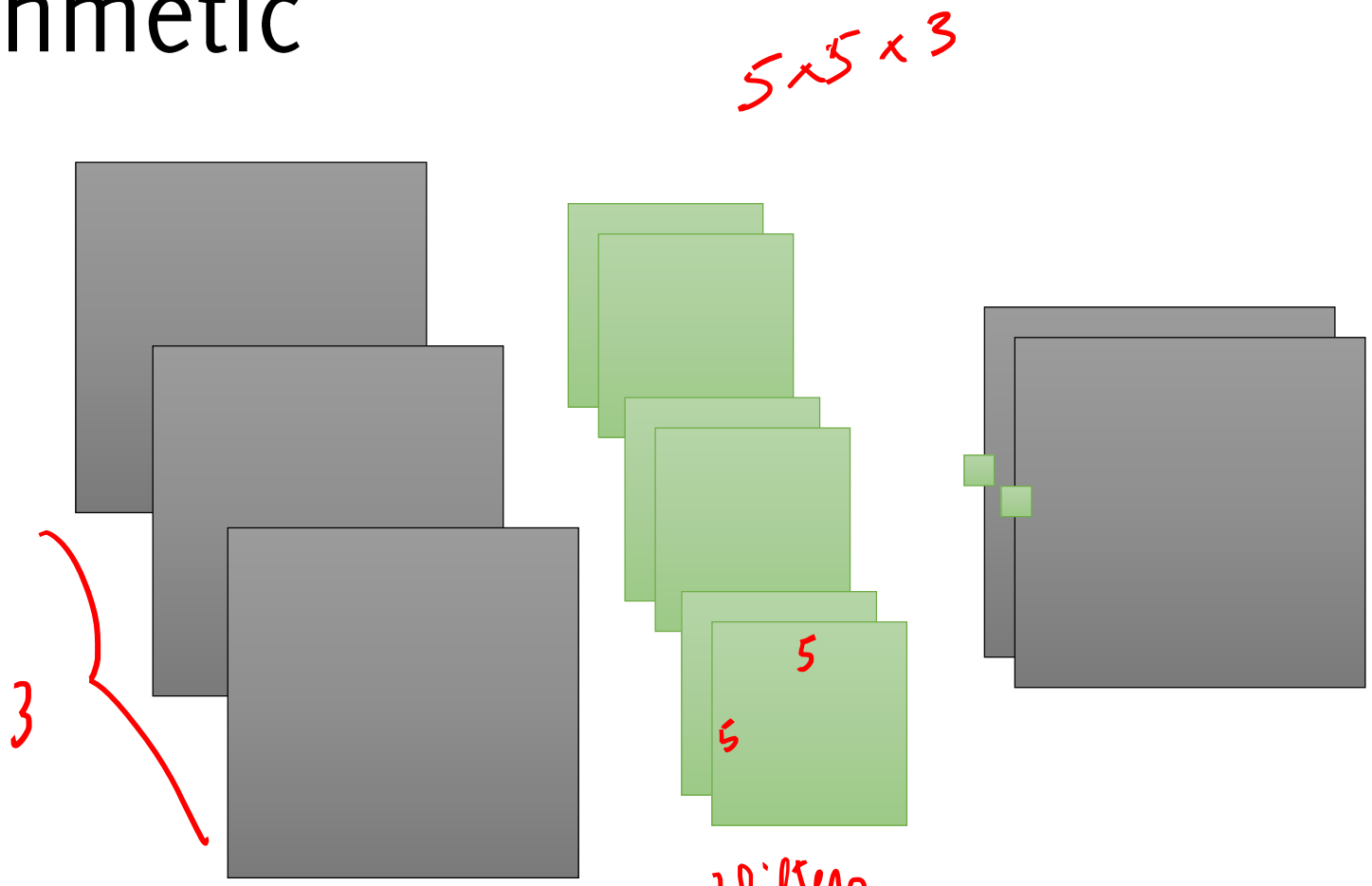


3 input maps 2 filters 3x3 2 output maps

= 54 parameters in the filters



CNN Arithmetic



3 input maps

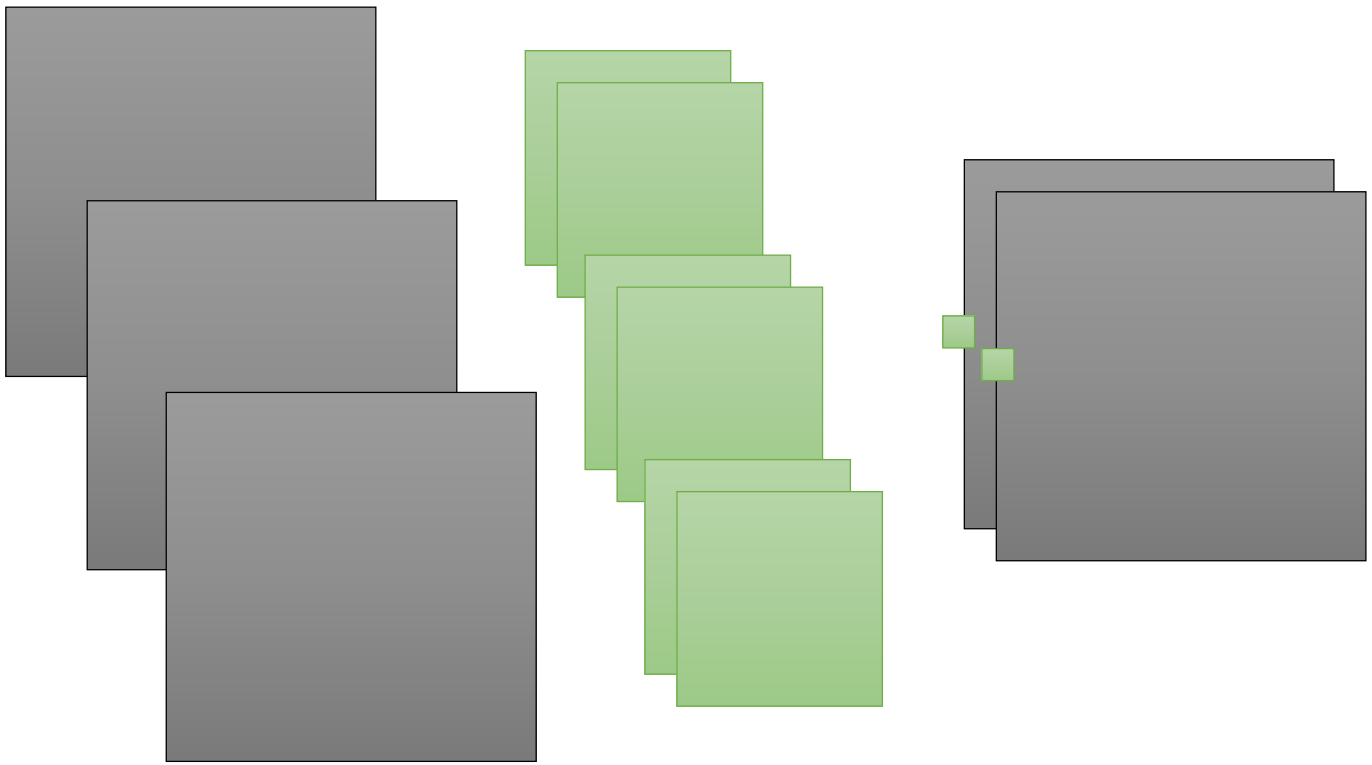
2 filters 3x3

= 54 ...

2 output maps

+ 2 biases

CNN Arithmetic



3 input maps

2 filters 3x3

= 54 ...

2 output maps

+ 2 biases

= 56 trainable parameters (weights)

Other Layers

Activation and Pooling

G. Boracchi

Activation Layers

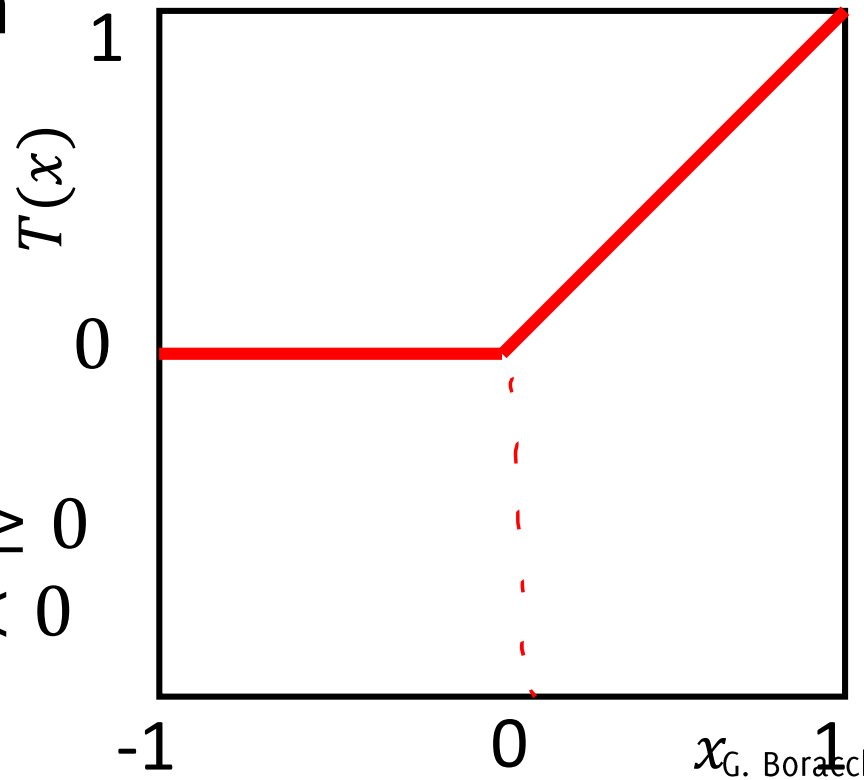
ReLu

Introduce nonlinearities in the network, otherwise the CNN might be equivalent to a linear classifier…

RELU (Rectifier Linear Units): it's a thresholding on the feature maps, i.e., a $\max(0,\cdot)$ operator.

- By far the most popular activation function in deep NN (since when it has been used in AlexNet)
- Dying neuron problem: a few neurons become insensitive to the input (vanishing grandient problem)

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

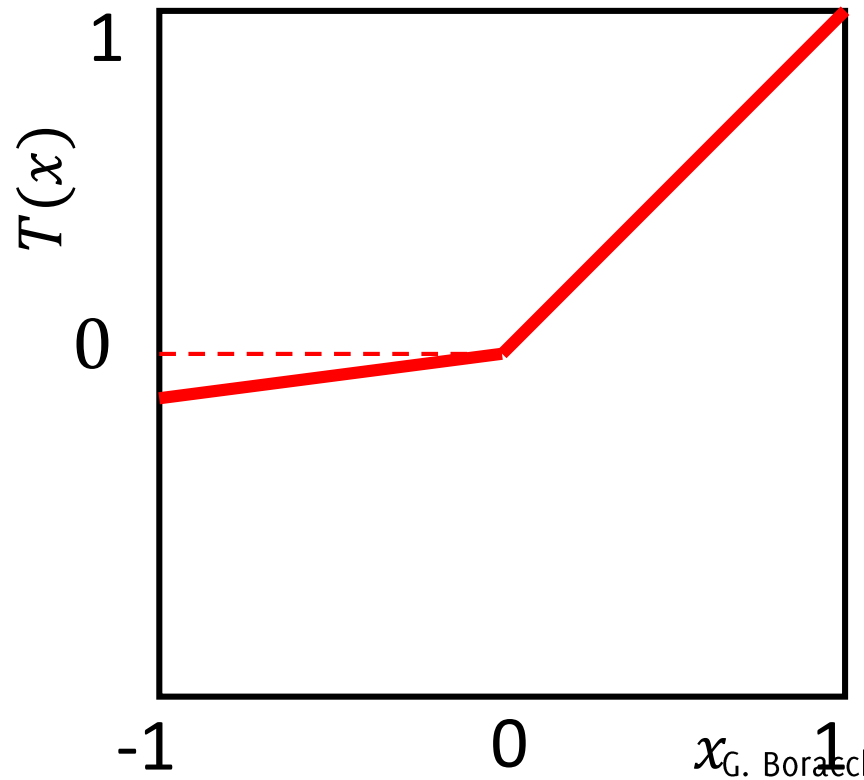


Activation Layers

Introduce nonlinearities in the network, otherwise the CNN might be equivalent to a linear classifier…

LEAKY RELU: like the relu but include a small slope for negative values

$$T(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.01 * x & \text{if } x < 0 \end{cases}$$



Activation Layers

Introduce nonlinearities in the network, otherwise the CNN might be equivalent to a linear classifier…

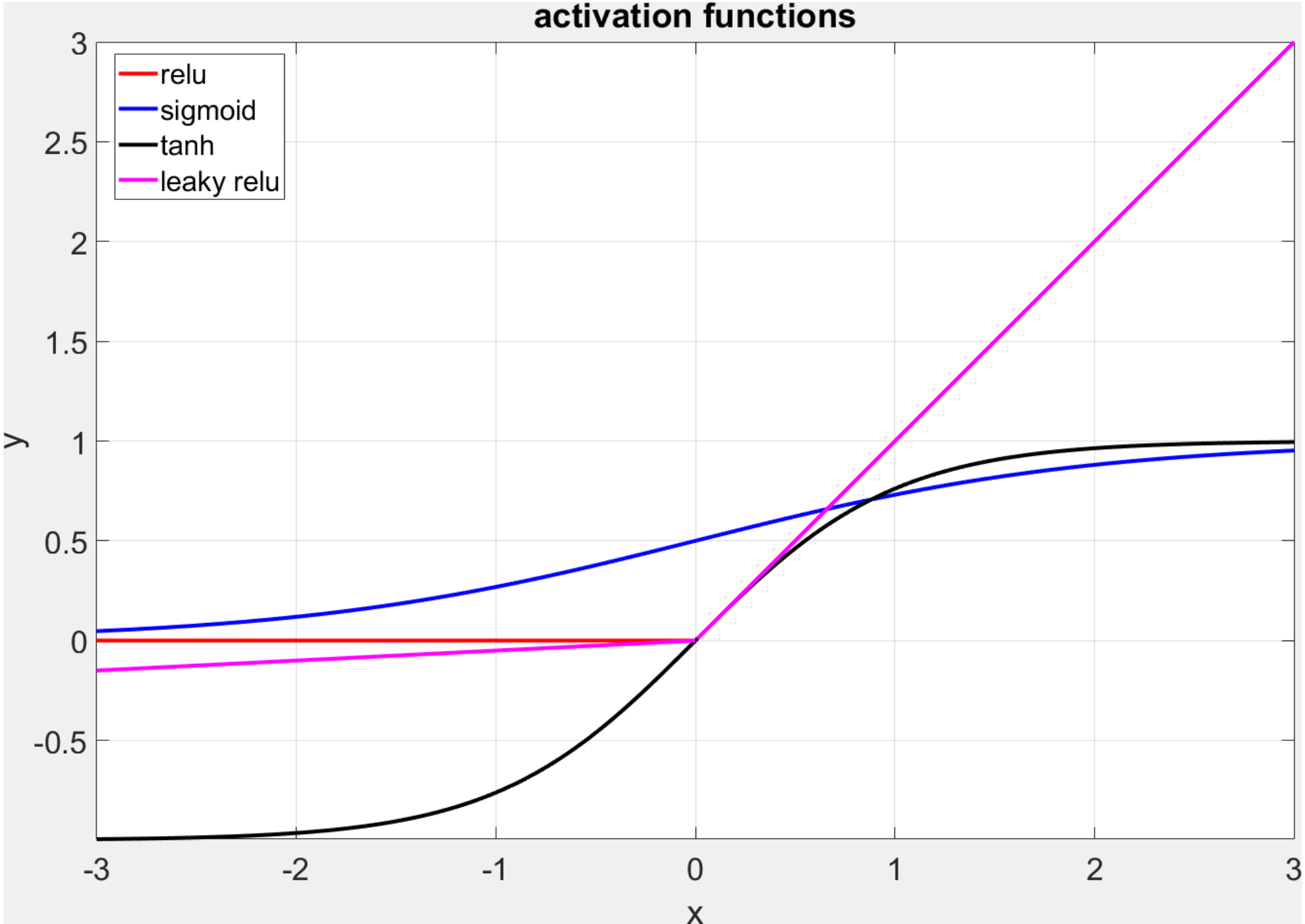
TANH (hyperbolic Tangent): has a range (-1,1), continuous and differentiable

$$T(x) = \frac{2}{1 + e^{-2x}} - 1$$

SIGMOID: has a range (0,1), continuous and differentiable

$$S(x) = \frac{1}{1 + e^{-2x}}$$

These activation functions are mostly popular **in MLP architectures**


activation functions
relu
sigmoid
tanh
leaky relu
3
2.5
2
1.5
1
0.5
0
-0.5
y
-3
-2
-1
0
1
2
3
x

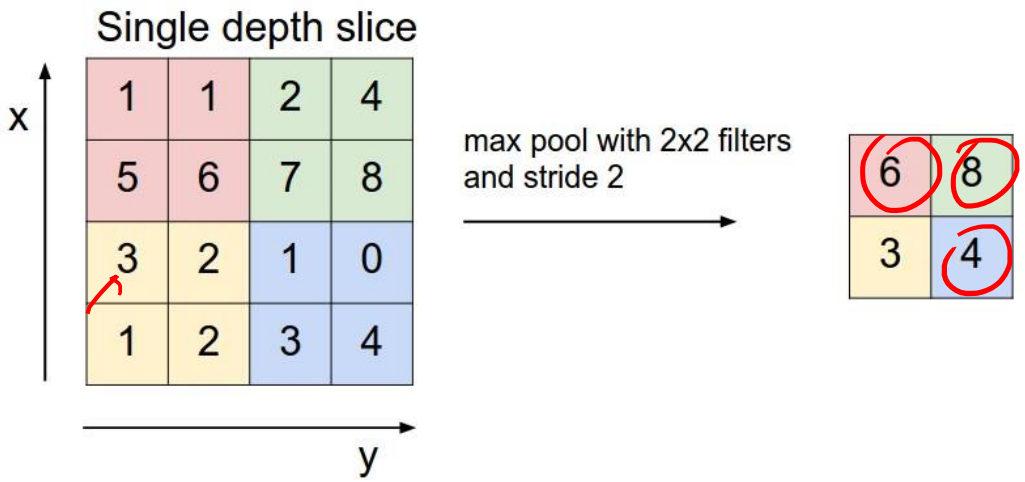
G. Boracchi

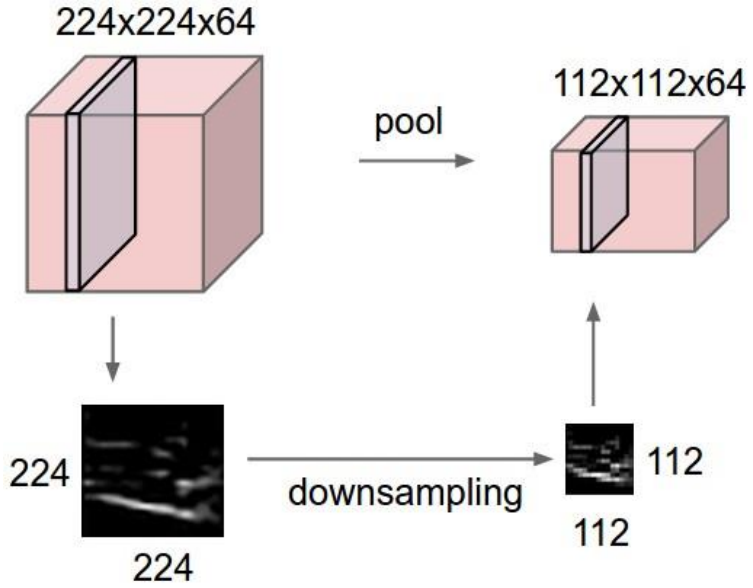
Pooling Layers

Pooling Layers reduce the **spatial** size of the volume.

The Pooling Layer operates **independently on every depth slice** of the input and **resizes it spatially, often using the MAX operation.**

Max pooling





In a 2x2 support it discards 75% of samples in a volume

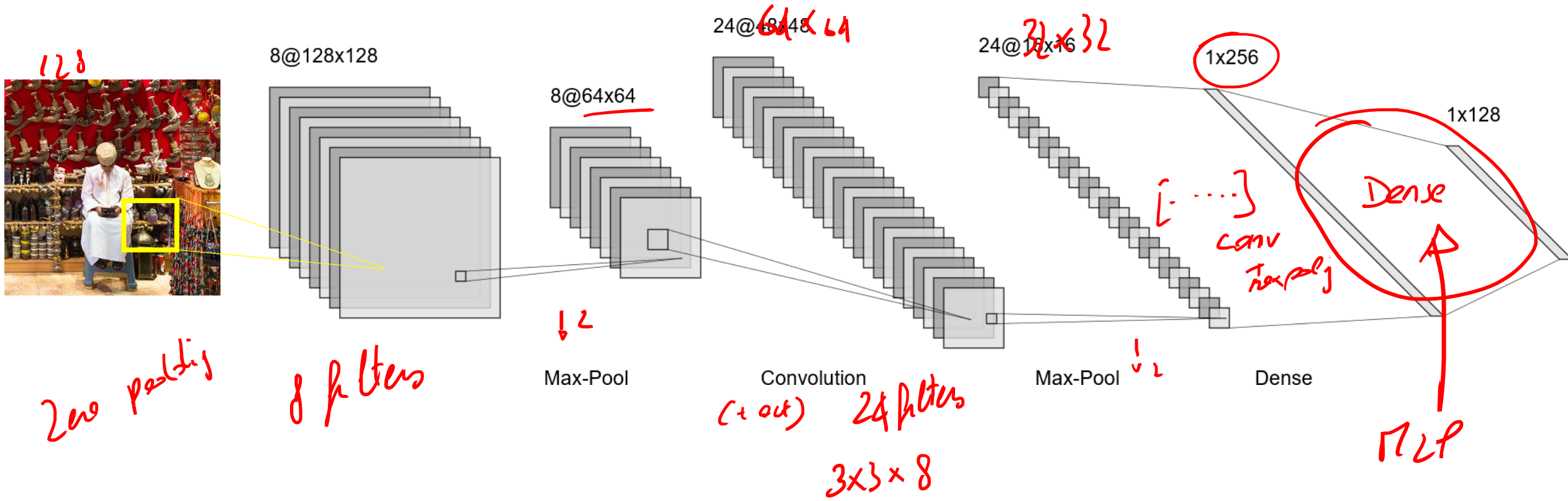
Dense Layers

Activation and Pooling

G. Boracchi

Convolutional Neural Networks (CNN)

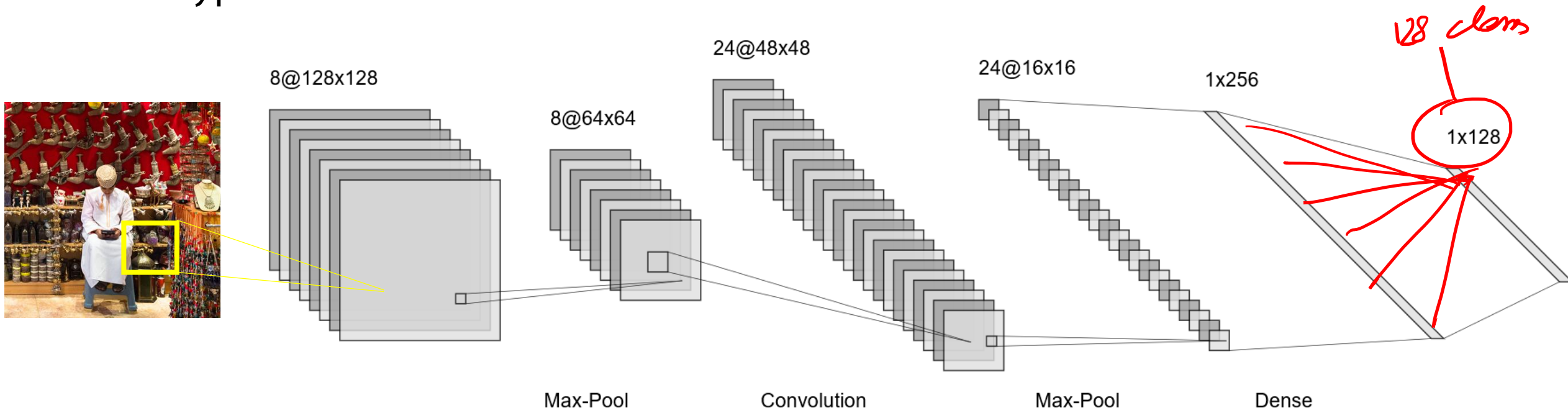
The typical architecture of a convolutional neural network



LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. *"Gradient-based learning applied to document recognition"* Proceedings of the IEEE, 1998 86(11), 2278-2324.

Convolutional Neural Networks (CNN)

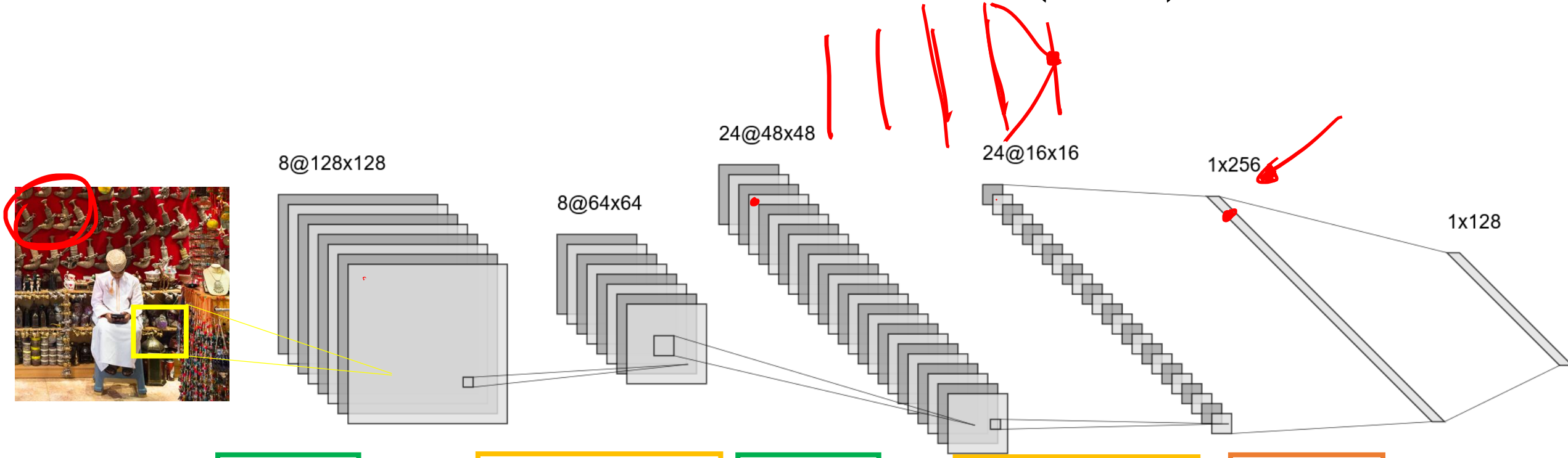
The typical architecture of a convolutional neural network



The output of the **fully connected (FC) layer** has the same size as the **number of classes**, and **provides a score** for the input image to belong to each class

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. *"Gradient-based learning applied to document recognition"* Proceedings of the IEEE, 1998 86(11), 2278-2324.

Convolutional Neural Networks (CNN)



Conv | Subsampling | Conv. | Subsampling | Dense

Convolution filters are learned for the classification task at hand

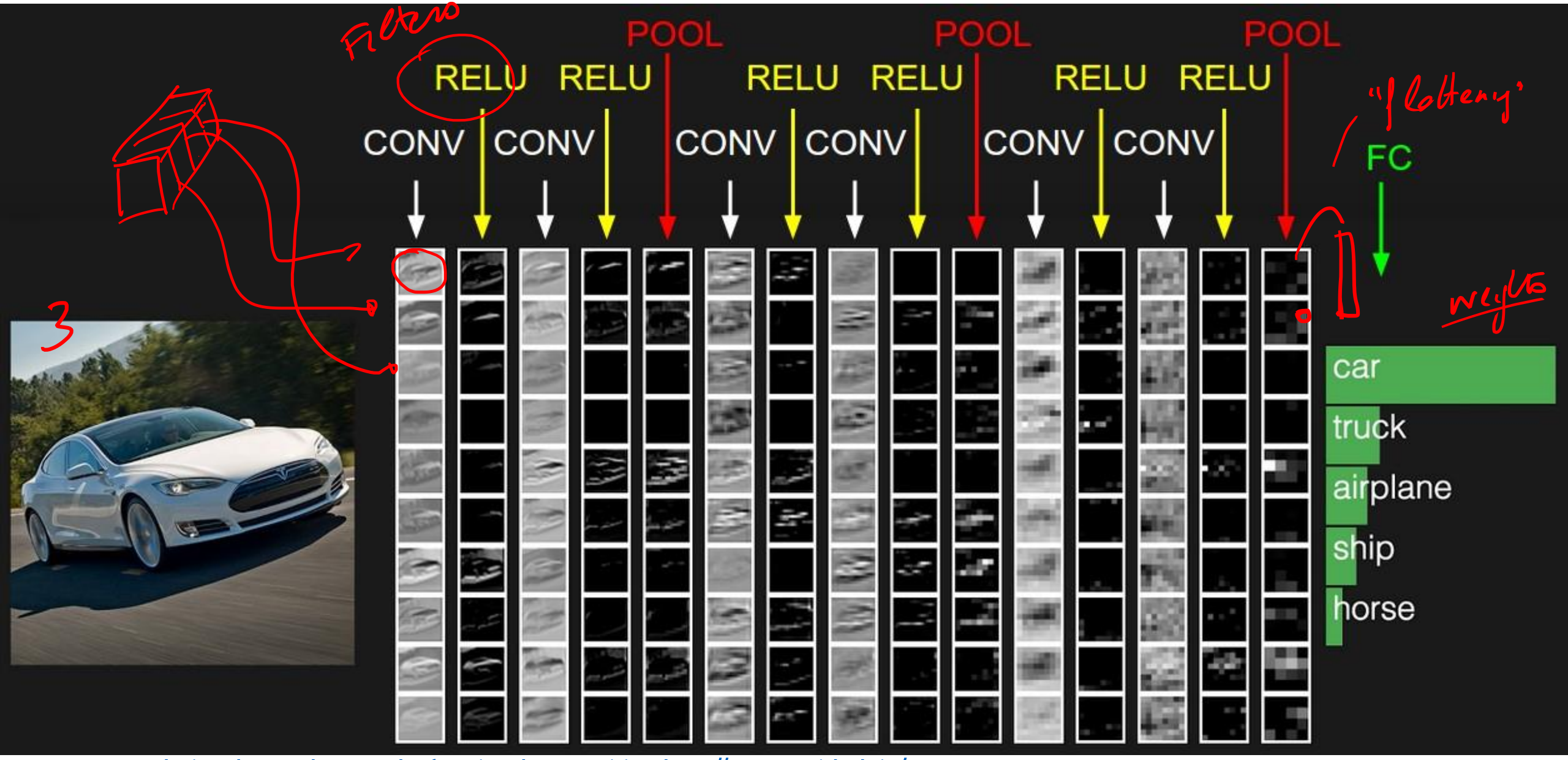
Thresholding + Downsampling (ReLu + Maxpooling)

Fully connected Neural Network providing as output class probabilities

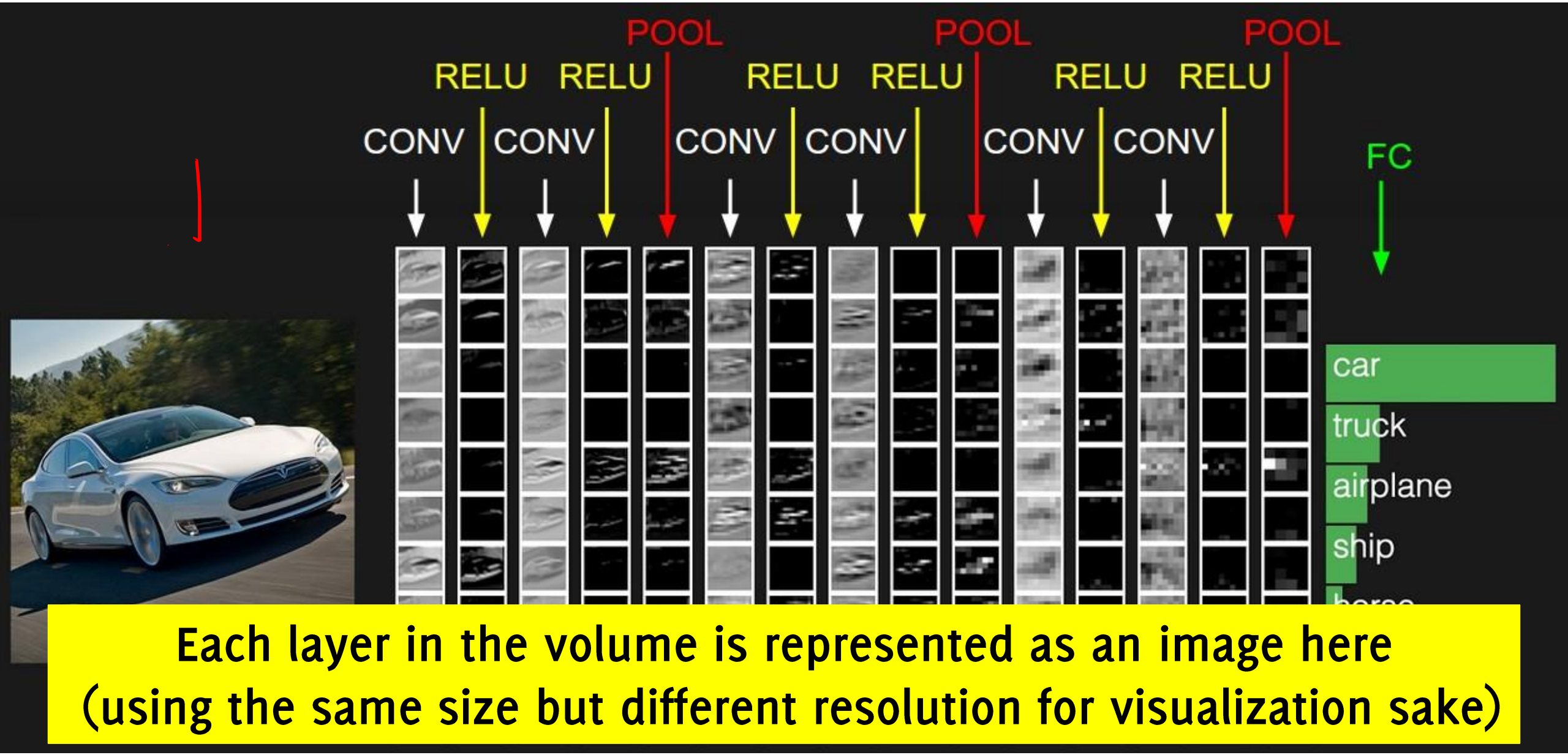
G. Boracchi

CNN «in action»

Activations in a convolutional network

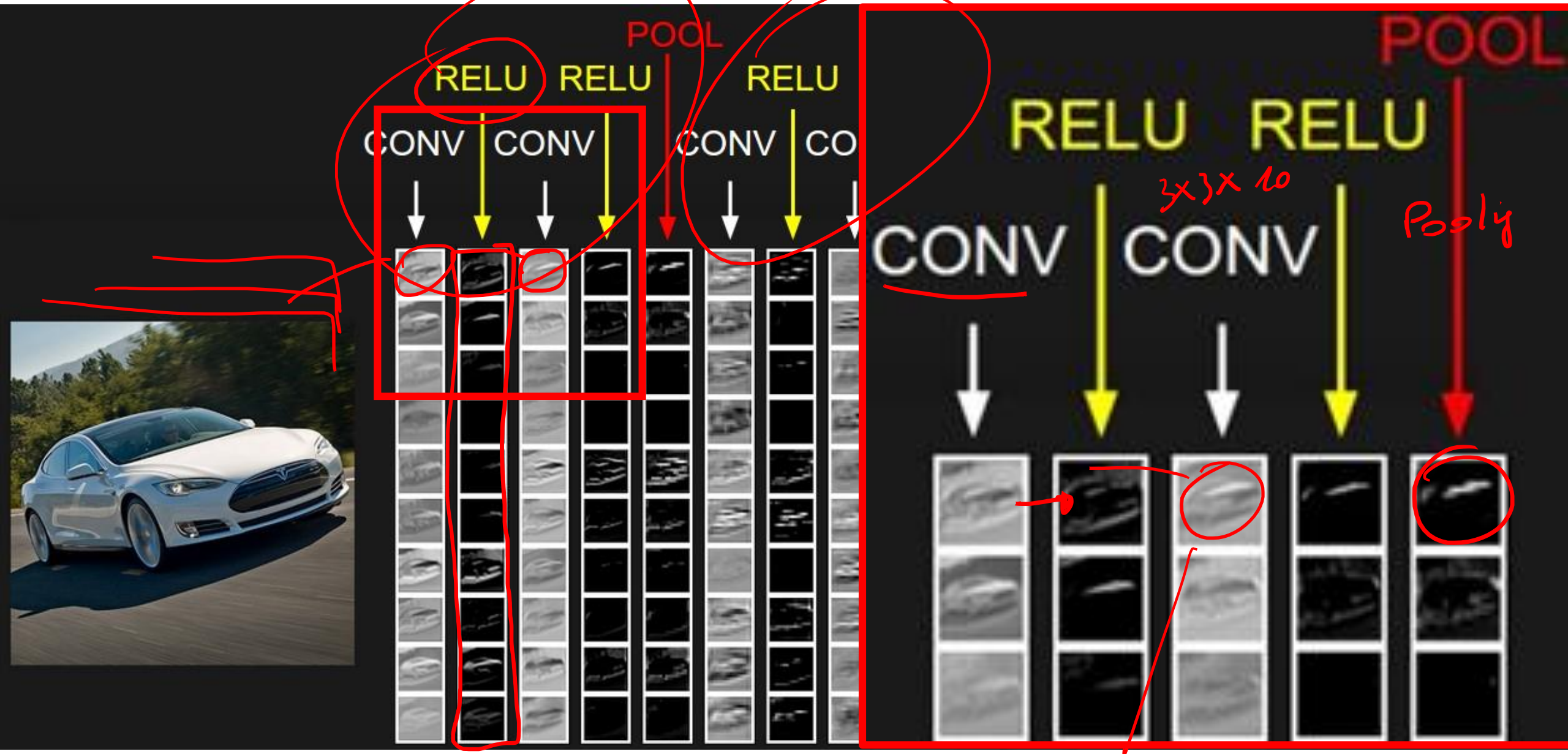


Activations in a convolutional network

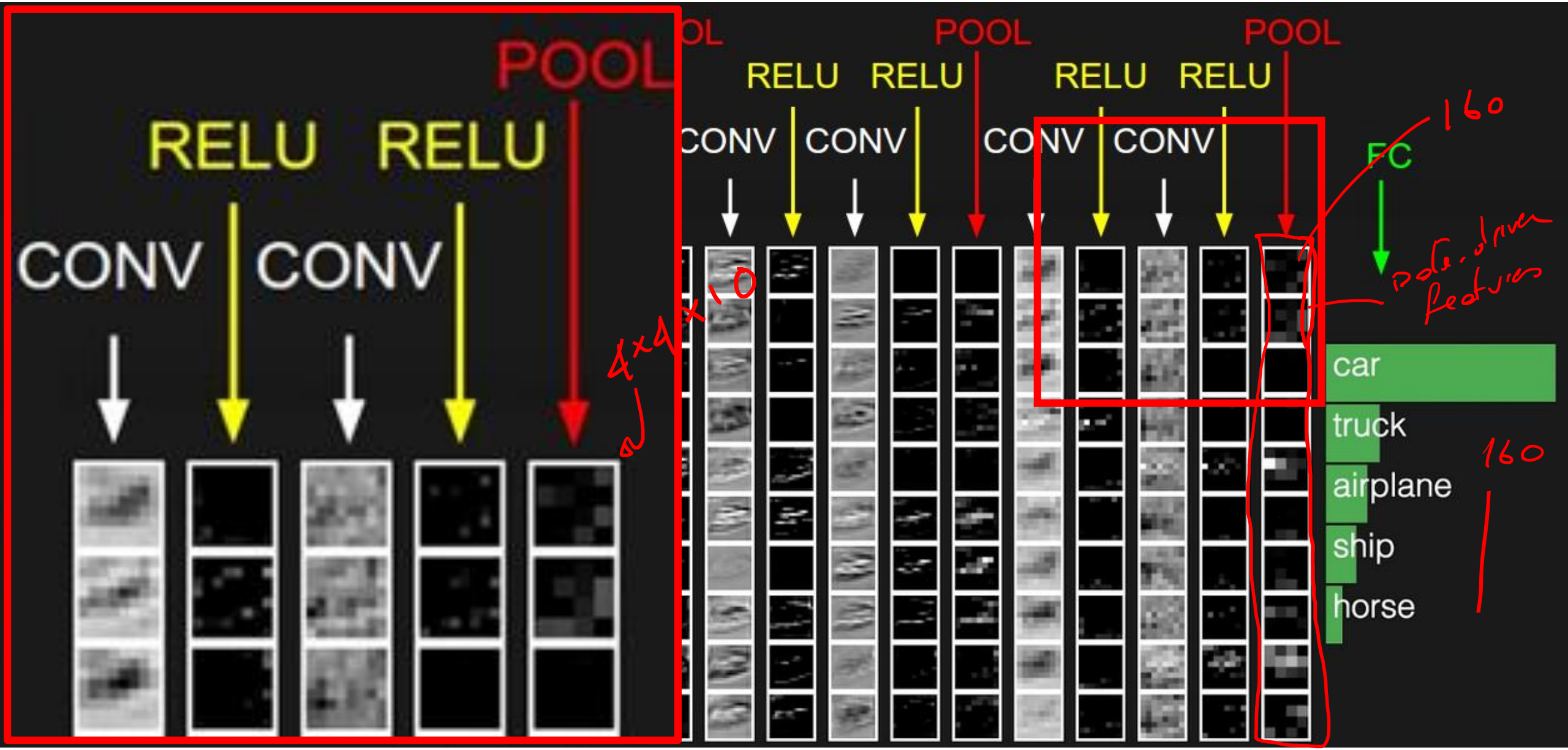


Each layer in the volume is represented as an image here
(using the same size but different resolution for visualization sake)

Activations in a convolutional network

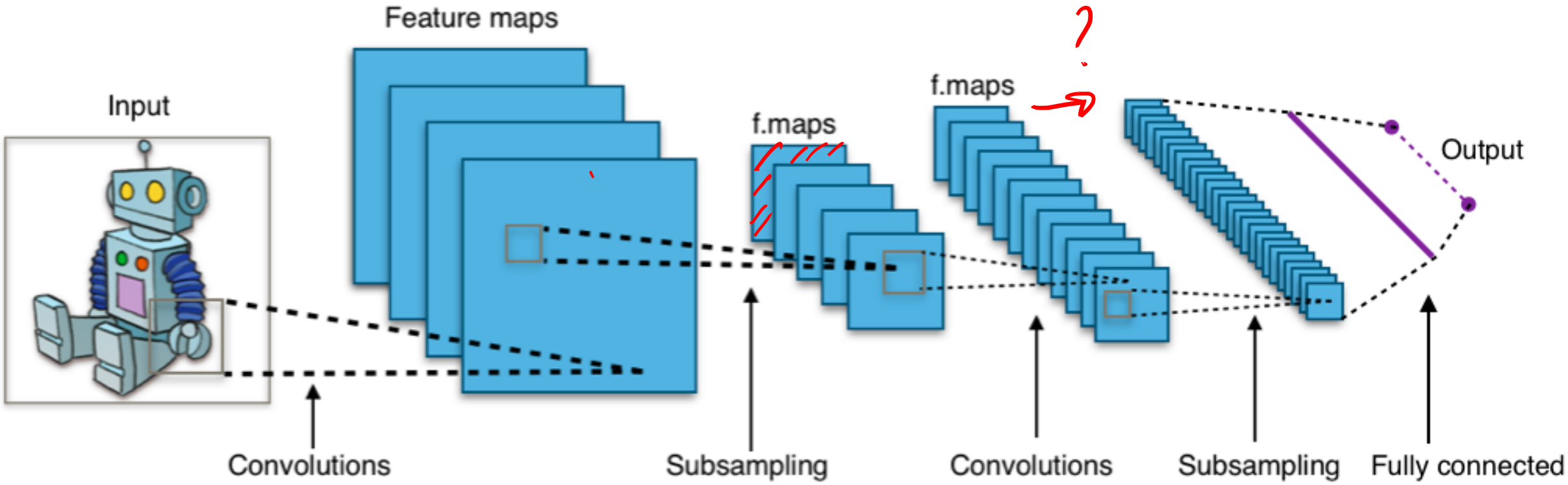


Activations in a convolutional network



Convolutional Neural Networks (CNN)

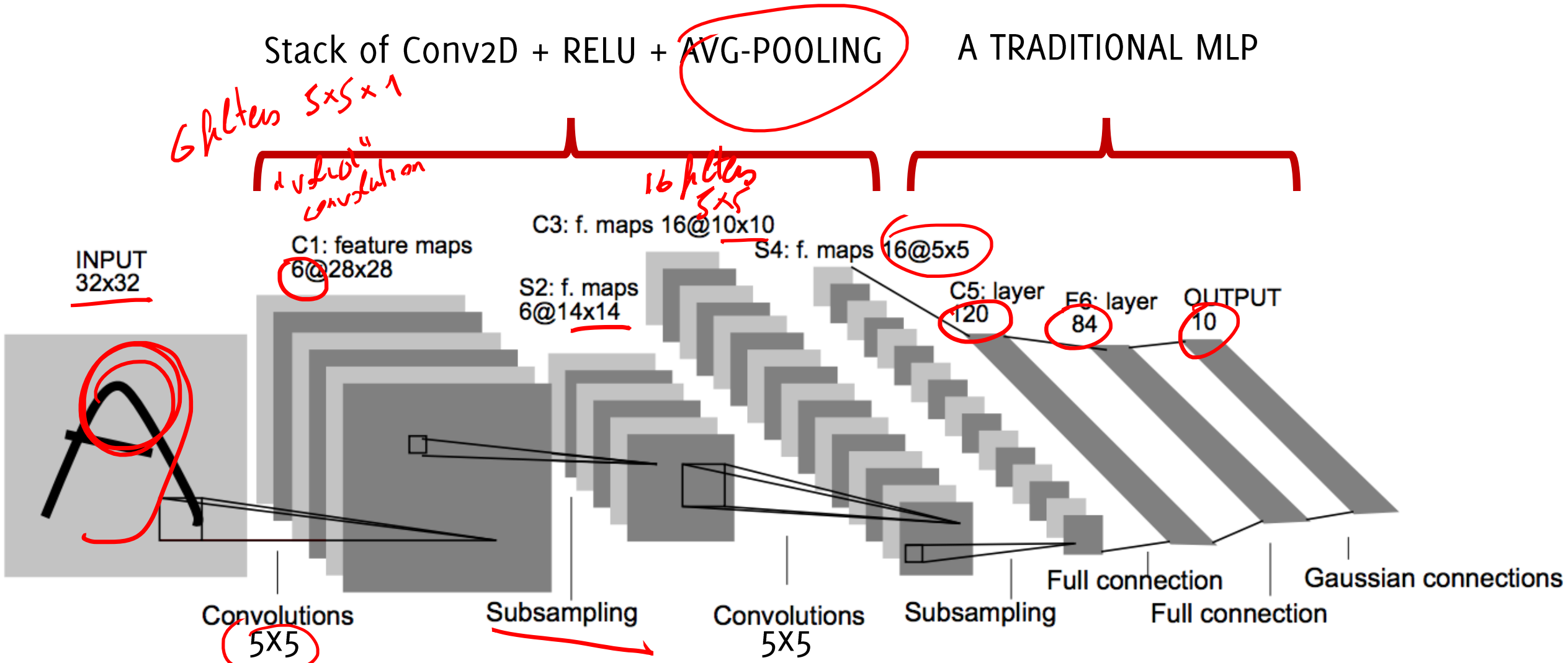
Btw, this figure contains an error.
If you are CNN-Pro, you should spot it!



By Aphex34 - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=45679374

The First CNN

LeNet-5 (1998)



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

chi

The first CNN

Do not use each pixel as a separate input of a large MLP, because:

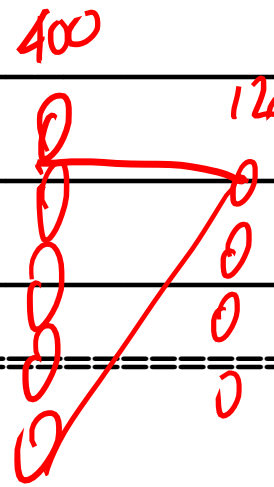
- images are highly spatially correlated,
- using individual pixel of the image as separate input features would not take advantage of these correlations.

The first convolutional layer: 6 filters 5x5

The second convolutional layer: 16 filters 5x5

model.summary()

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_1 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```



model.summary()

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         156 (6 x 5 x 5 + 6)
_________________________________________________________________
average_pooling2d_1 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416 (16 x 5 x 5 x 6 + 16)
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

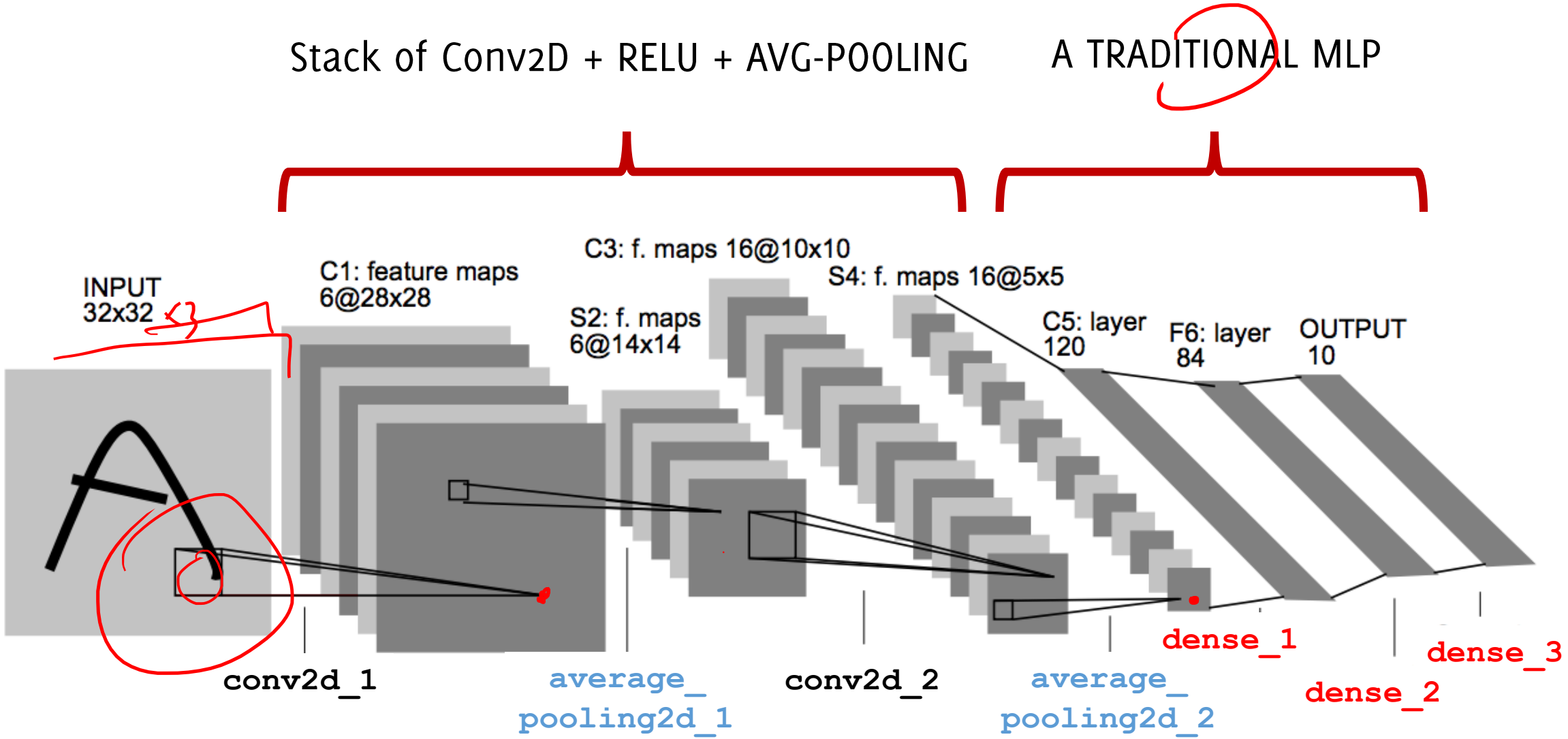
61K

Input is a grayscale image

The input is a volume having depth = 6

Most parameters are still in the MLP

LeNet-5 (1998)



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

chi

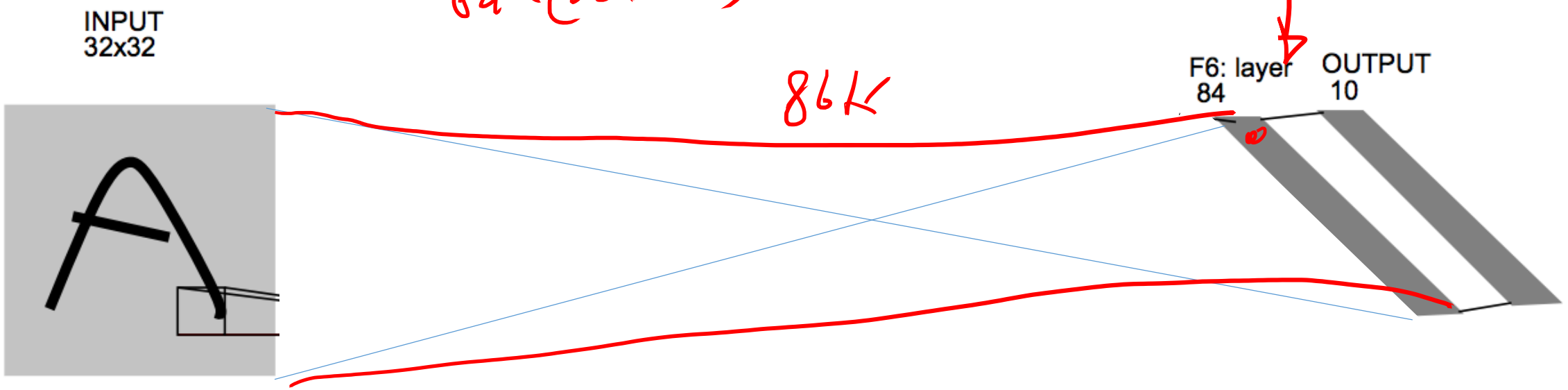
Most of parameters are in MLP

What about a MLP taking as input the whole image?

Input 32 x 32 = 1024 pixels, fed to a 84 neurons (the last FC layers of the network) -> 86950 parameters: 1024 * 84 + 84 + 84 * 10 + 10

84 x (32 x 32) + 84 +

86K



Most of parameters are in MLP

What about a MLP taking as input the whole image?

Input 32 x 32 = 1024 pixels, fed to a 84 neurons (the last FC layers of the network) -> 86950 parameters

But.. If you take an RGB input: 32 x 32 x 3,

CNN: only the nr. of parameters in the filters at the first layer increases

$$156 + 61550 \rightarrow 456 + 61550$$

$$(6 \times 5 \times 5) \rightarrow (6 \times 5 \times 5 \times 3)$$

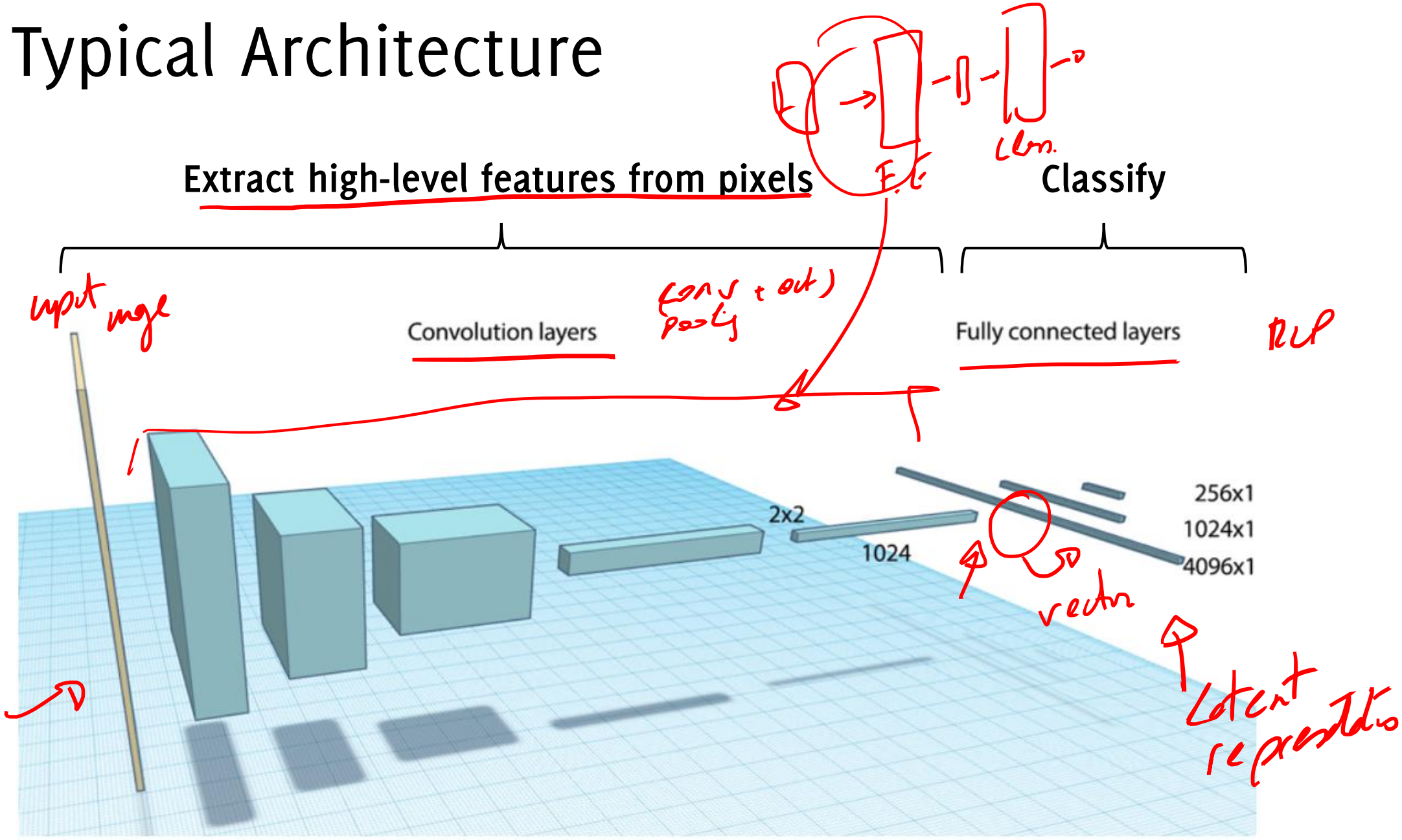
MLP: everything increases by a factor 3

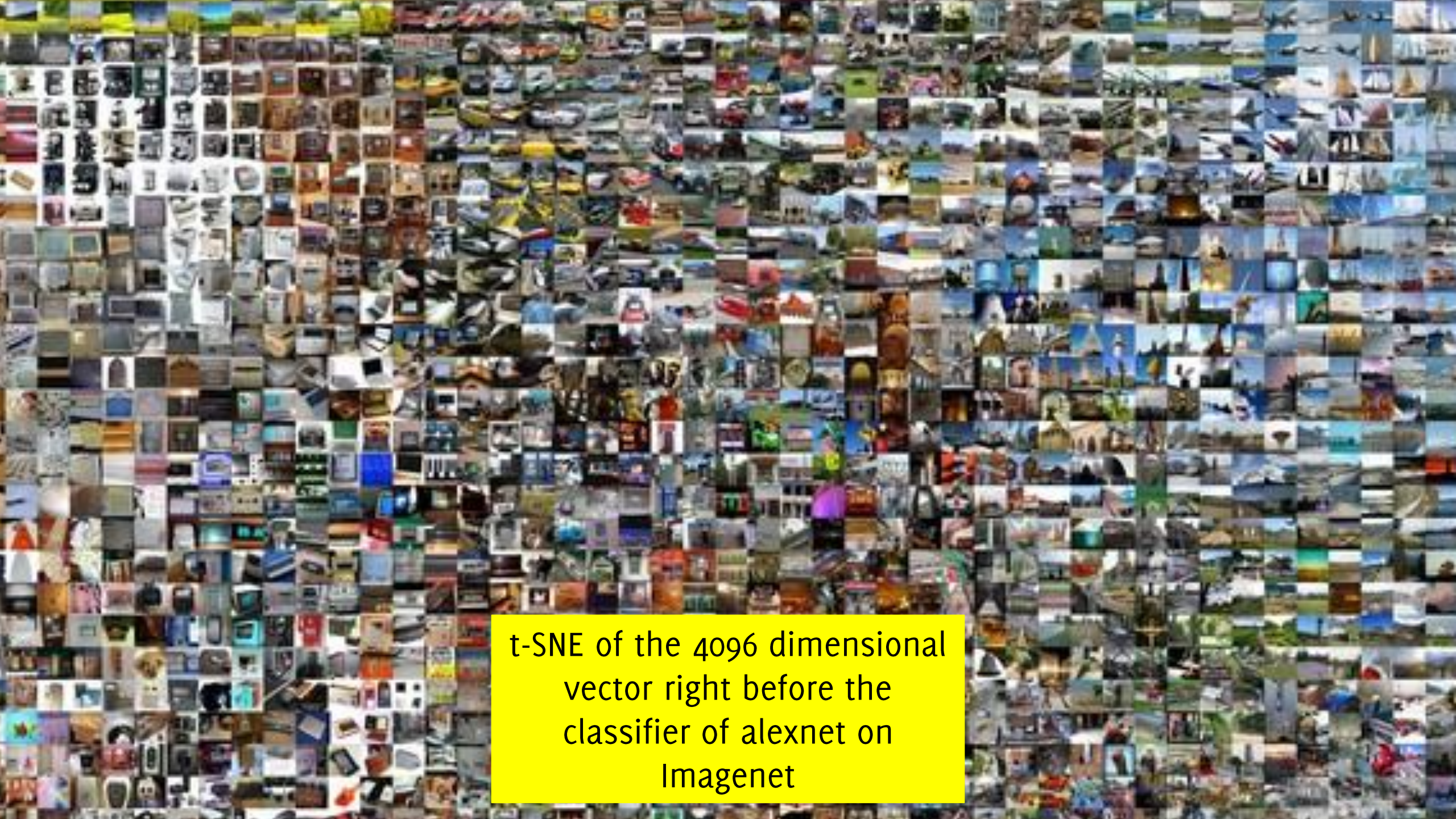
$$86950 \rightarrow 86950 \times 3$$

Latent representation in CNNs

Repeat the «t-SNE experiment» on the CIFAR dataset, using the last layer of the CNN as vectors

A Typical Architecture




t-SNE of the 4096 dimensional vector right before the classifier of alexnet on Imagenet


Distances in the latent representation of a CNN are much more meaningful than data itself

Keras

What is Keras?

An open-source library providing **high-level building blocks** for developing deep-learning models in Python

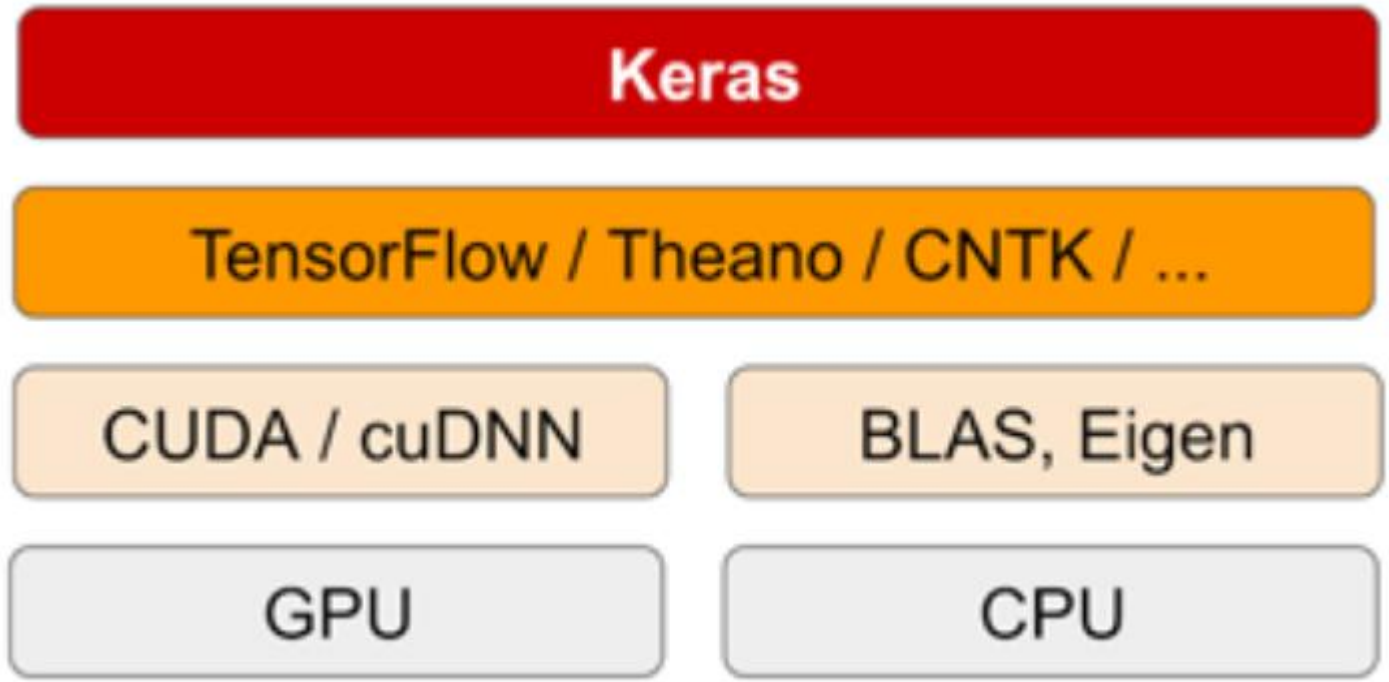
Designed to enable **fast experimentation with deep neural networks**, it focuses on being **user-friendly, modular, and extensible**

Doesn't handle low-level operations such as tensor manipulation and differentiation.

Relies on **backends** (TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML)

Enables **full access to the backend**

The software stack



Why Keras?

Pros:

Higher level → fewer lines of code

Modular backend → not tied to tensorflow

Way to go if you focus on applications

Cons:

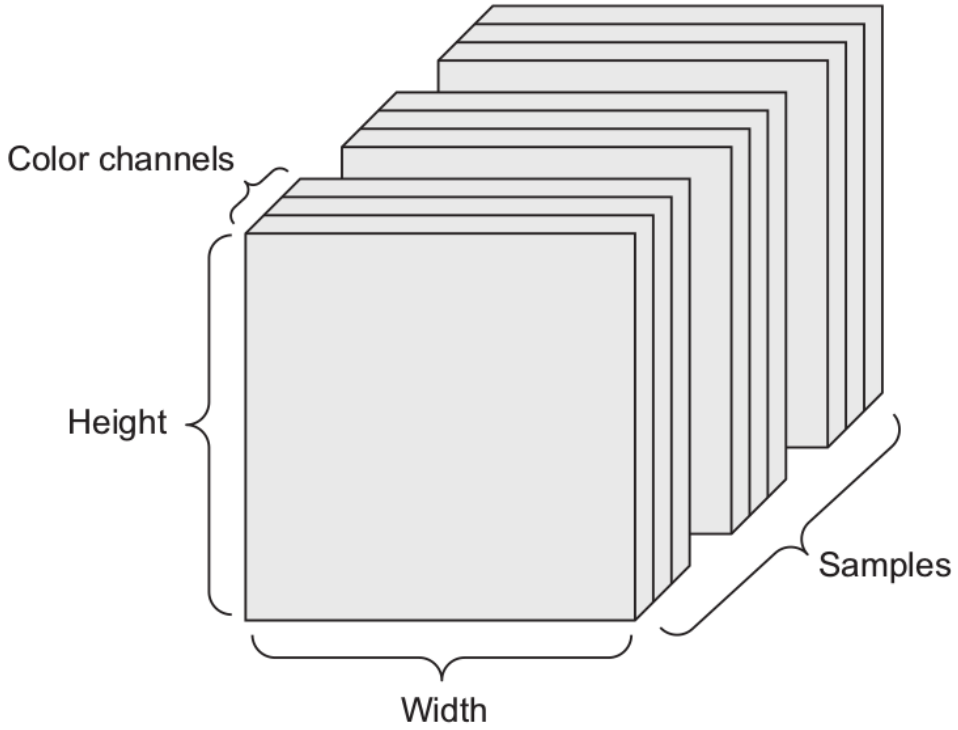
Not as flexible

Need more flexibility? Access the backend directly!

We will manipulate 4D tensors

Images are represented in 4D tensors:

Tensorflow convention: (samples, height, width, channels)



Building the Network

Convolutional Networks in Keras

```
# it is necessary to import some package
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.layers import Conv2D, MaxPooling2D

# and initialize an object from Sequential()
model = Sequential()
```

A very simple CNN

```
# Network Layers are stacked by means of the
.add() method
model.add(Conv2D(filters = 64, kernel_size=3,
activation='relu', input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```

Convolutional Layers

```
# Convolutional Layer
model.add(Conv2D(filters = 64, kernel_size=3,
activation='relu', input_shape=(28,28,1)))
# the input are meant to define:
# - The number of filters,
# - The spatial size of the filter (assumed
squared), while the depth depends on the network
structure
# - the activation layer (always include a
nonlinearity after the convolution)
# - the input size: (rows, cols, n_channels)
```

Convolutional Layers

```
# Convolutional Layer
model.add(Conv2D(filters = 64, kernel_size=3,
activation='relu', input_shape=(28,28,1)))
# This layer creates a convolution kernel that
is convolved with the layer input to produce a
tensor of outputs.
# When using this layer as the first layer in a
model, provide the keyword argument input_shape
(tuple of integers, does not include the batch
axis), e.g. input_shape=(128, 128, 3) for
128x128 RGB pictures in
data_format="channels_last".
```

Conv2D help

Arguments

filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

kernel_size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

strides: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.

padding: one of "valid" or "same" (case-insensitive). Note that "same" is slightly inconsistent across backends with strides != 1, as described here

data_format: A string, one of "channels_last" or "channels_first". The ordering of the dimensions in the inputs. "channels_last" corresponds to inputs with shape (batch, height, width, channels) while "channels_first" corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

Conv2D help

Arguments

dilation_rate: an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.

activation: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).

use_bias: Boolean, whether the layer uses a bias vector.

kernel_initializer: Initializer for the kernel weights matrix (see initializers).

bias_initializer: Initializer for the bias vector (see initializers).

kernel_regularizer: Regularizer function applied to the kernel weights matrix (see regularizer).

bias_regularizer: Regularizer function applied to the bias vector (see regularizer).

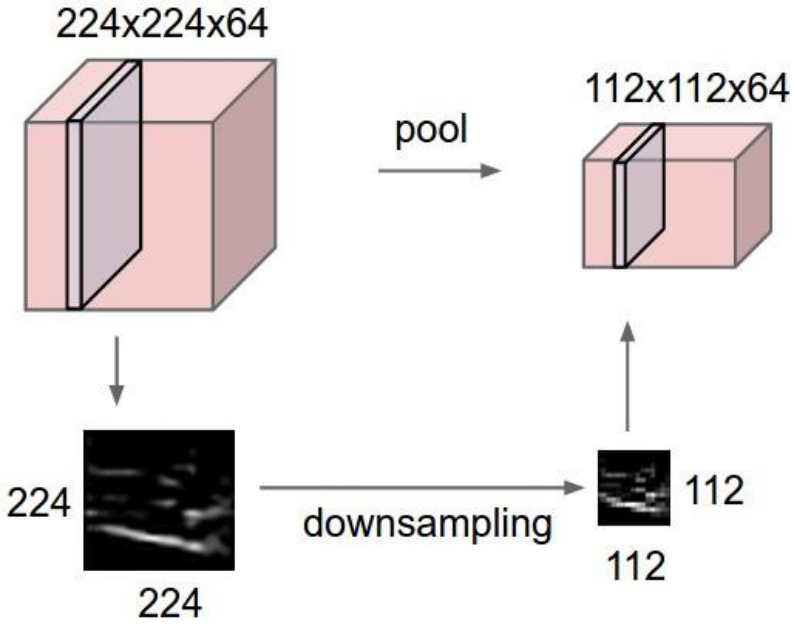
activity_regularizer: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).

kernel_constraint: Constraint function applied to the kernel matrix (see constraints).

bias_constraint: Constraint function applied to the bias vector (see constraints).

MaxPooling Layers

```
# Maxpooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))
# the only parameter here is the (spatial) size
to be reduced by the maximum operator
```



MaxPooling2D help

Arguments:

pool_size: integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.

strides: Integer, tuple of 2 integers, or None. Strides values. If None, it will default to pool_size.

padding: One of "valid" or "same" (case-insensitive).

data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

MaxPooling2D help

Input shape:

If data_format='channels_last': 4D tensor with shape: (batch_size, rows, cols, channels)

If data_format='channels_first': 4D tensor with shape: (batch_size, channels, rows, cols)

Output shape:

If data_format='channels_last': 4D tensor with shape: (batch_size, pooled_rows, pooled_cols, channels)

If data_format='channels_first': 4D tensor with shape: (batch_size, channels, pooled_rows, pooled_cols)

Fully Connected Layers

```
# at the end the activation maps are "flattened" i.e.
they move from an image to a vector (just unrolling)
model.add(Flatten())
# Dense is a Fully Connected layer in a traditional
Neural Network.
model.add(Dense(units=10, activation='softmax'))
# Implements:
# output = activation(dot(input, kernel) + bias)
```

- activation is the element-wise activation function passed as the activation argument,
- kernel is a weights matrix created by the layer,
- bias is a bias vector created by the layer

```
# "Units" defines the number of neurons
```

Visualizing the model

```
# a nice output describing the model architecture
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
flatten_3 (Flatten)          (None, 43264)             0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                432650
=================================================================
Total params: 433,290
Trainable params: 433,290
Non-trainable params: 0
```

Training the Model

Compiling the model

Then we need to compile the model using the compile method and specifying:

- **optimizer** which controls the learning rate. Adam is generally a good option as it adjusts the learning rate throughout training.
- **loss function** the most common choice for classification is ‘categorical_crossentropy’ for our loss function. The lower the better.
- **Metric** to assess model performance, ‘accuracy’ is more interpretable

```
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])
```

Training the model using

The **`fit()`** method of the model is used to train the model.

Specify at least the following inputs:

- training data (input images),
- target data (corresponding labels in categorical format),
- validation data (a pair of data, labels to be used only for computing validation performance)
- number of epochs (number of times the whole dataset is scanned during training)

```
model.fit(X_train, y_train,
validation_data=(X_test, y_test), epochs=3)
```

Training output

```
Epoch 22/100

18000/18000 [==============================] - 136s 8ms/step - loss: 0.7567
- acc: 0.6966 - val_loss: 1.9446 - val_acc: 0.4325

Epoch 23/100

18000/18000 [==============================] - 137s 8ms/step - loss: 0.7520
- acc: 0.6959 - val_loss: 1.9646 - val_acc: 0.4275

Epoch 24/100

18000/18000 [==============================] - 137s 8ms/step - loss: 0.7442
- acc: 0.7024 - val_loss: 1.9067 - val_acc: 0.4129
```

Advanced Training Options

Callbacks in Keras

A callback is a **set of functions** to be **applied at given stages** of the training procedure.

Callbacks give a **view on internal states** and statistics of the model **during training**.

You can pass **a list of callbacks** (as the keyword argument callbacks) to the .fit() method of the Sequential or Model classes.

The relevant methods of the callbacks will then be called at each stage of the training.

```
callback_list = [cb1,…,cbN]

model.fit(X_train, y_train,
validation_data=(X_test, y_test), epochs=3,
callbacks = callback_list)
```

Model Checkpoint

Training a network might take up to several hours

Checkpoints are snapshots of the state of the system to be saved in case of system failure.

When training a deep learning model, the checkpoint is the weights of the model. These weights can be used to make predictions as is, or used as the basis for ongoing training.

```
from keras.callbacks import ModelCheckpoint
```

[…]

```
cp = ModelCheckpoint(filepath,
monitor='val_loss', verbose=0,
save_best_only=False, save_weights_only=False,
mode='auto', period=1)
```

Early Stopping

The only stopping criteria when training a Deep Learning model is "reaching the required number of epochs."

However, it might be enough to train a model further, as sometimes the training error decreases but the validation error does not (overfitting)

Checkpoints are used to stop training when a monitored quantity has stopped improving.

```
from keras.callbacks import EarlyStopping
```

[…]

```
es = EarlyStopping(monitor='val_loss',
min_delta=0, patience=0, verbose=0, mode='auto',
baseline=None, restore_best_weights=False)
```

Testing the model

Predict() method

```
#returns the class probabilities for the input
image X_test
score = model.predict(X_test)
# select the class with the largest score
prediction_test = np.argmax(score, axis=1)
```

Tensorboard

When training a model it is important to monitor its progresses

Google has developed tensorboard a very useful tool for visualizing reports.

```
from keras.callbacks import TensorBoard
```

[…]

```
  tb = TensorBoard(log_dir="dirname")
```

… and add tb to the checkpoint list as well