Image Classification

Giacomo Boracchi,
DEIB, Politecnico di Milano
October, 20th, 2021

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

Materials

https://boracchi.faculty.polimi.it/teaching/AN2DL.htm

What is image classification (and computer vision) about

Computer Vision

An interdisciplinary scientific field that deals with **how computers** can be made to **gain high-level**

understanding from digital images or videos

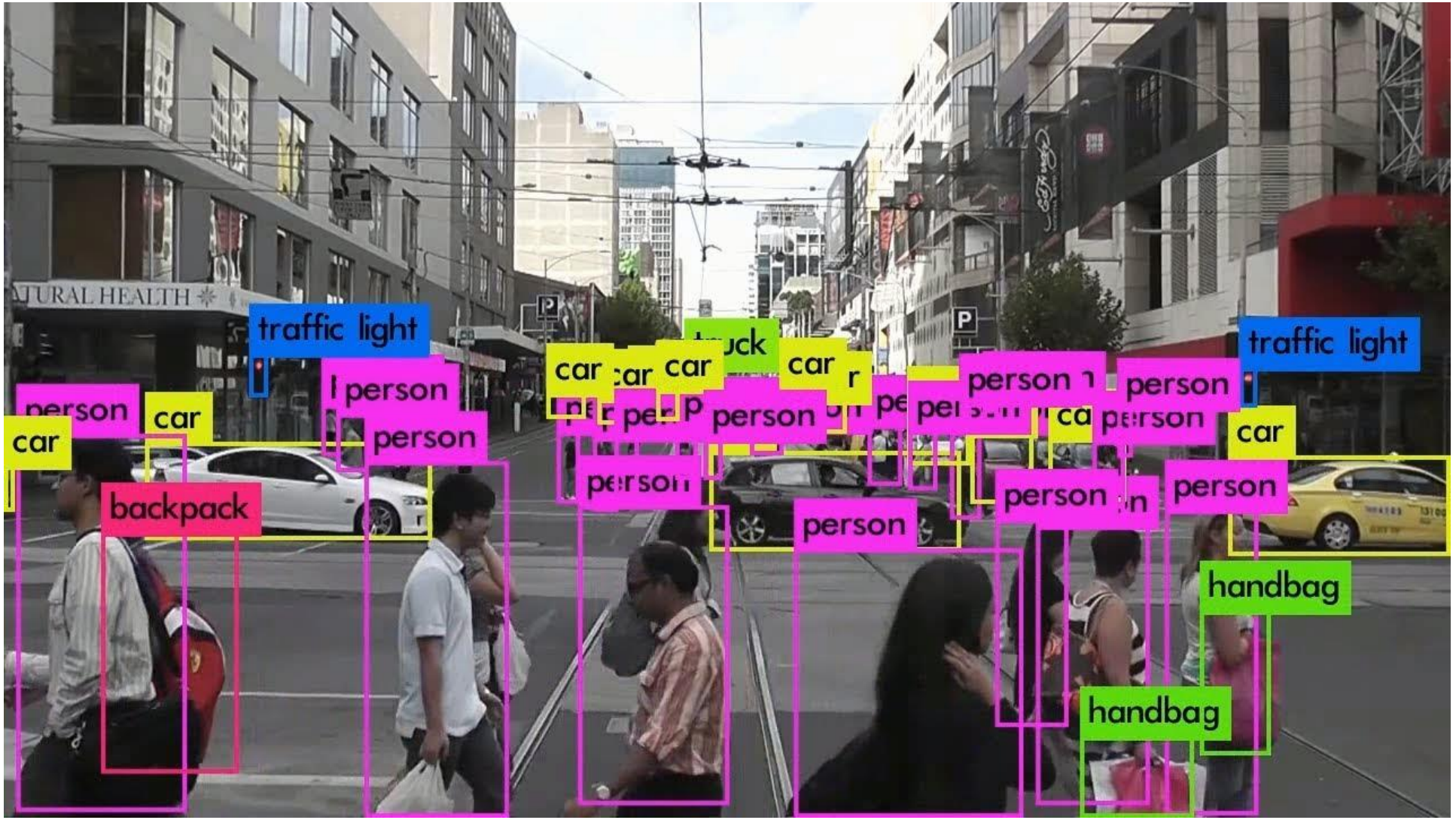
Computer Vision

An interdisciplinary scientific field that deals with **how computers** can be made to **gain high-level**

understanding from digital images or videos

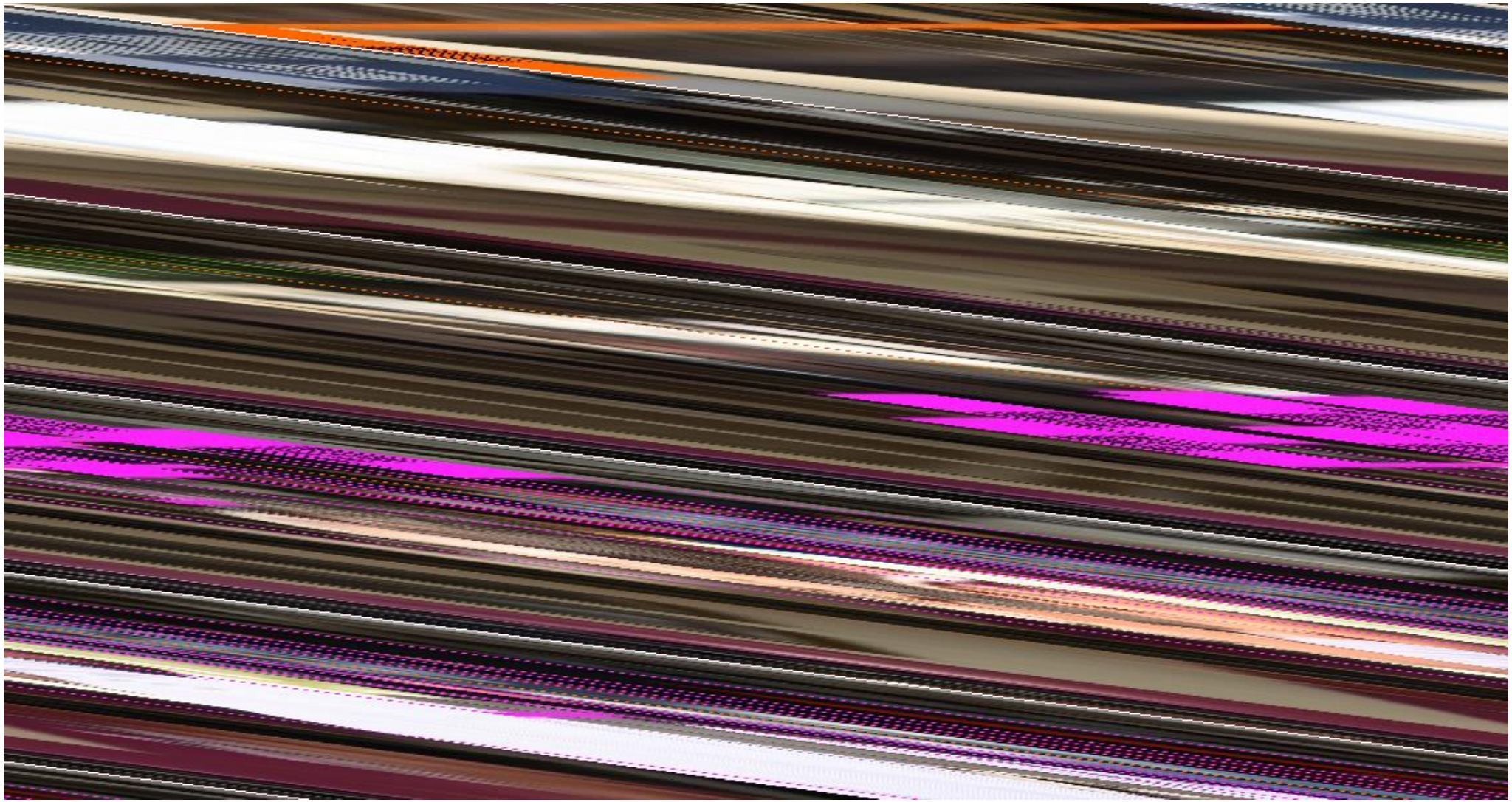
… which has grown incredibly fast in the last years

Object Detection



Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

Object Detection

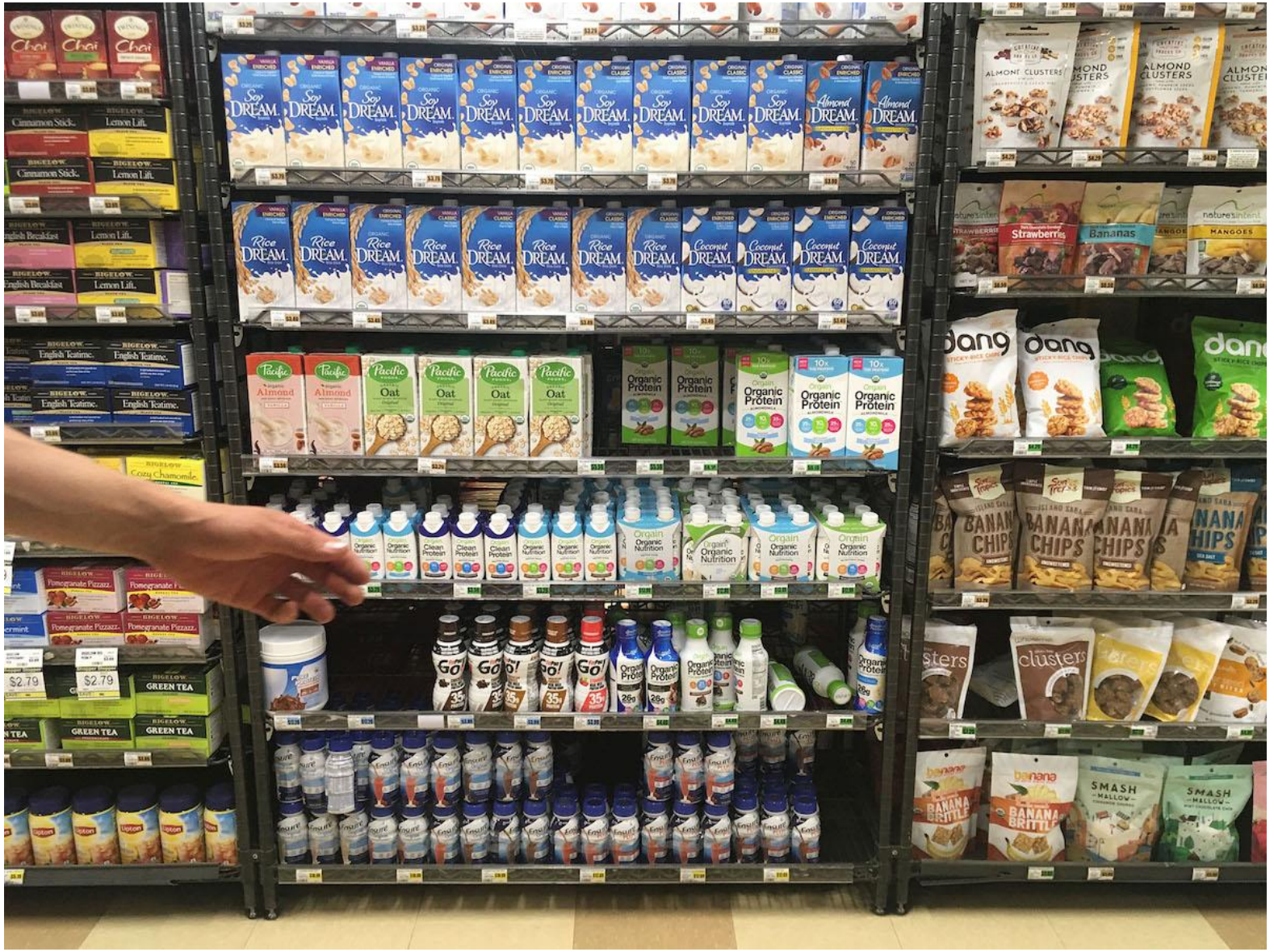


Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

Pose Estimation

Cao, Z., Simon, T., Wei, S. E., & Sheikh, Y. (2017). Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7291-7299).

Automatic Shelf Analysis



http:focal.systems

Automatic Shelf Analysis

http:focal.systems

Our Solution

Our Solution

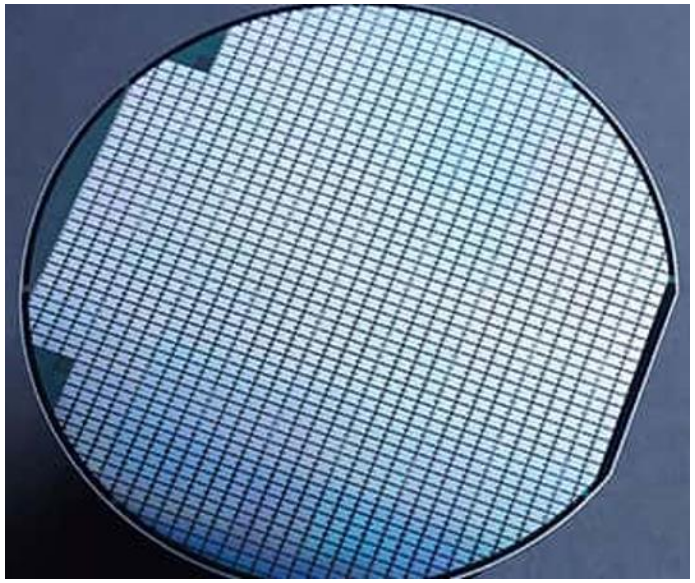
T&O ITALIA

Our Solution

T&O ITALIA

Quality Inspection

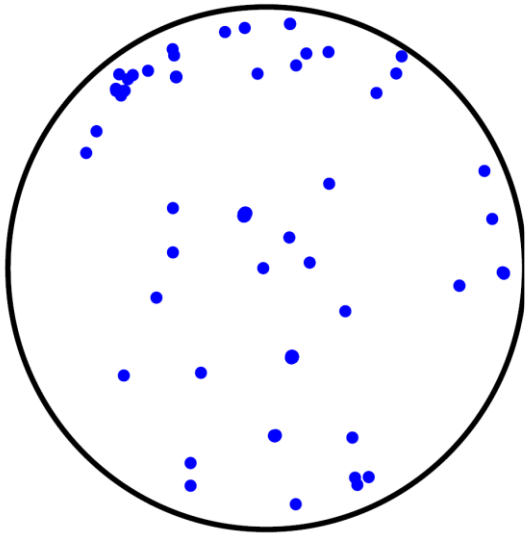
Wafer Defect Map (WDM)

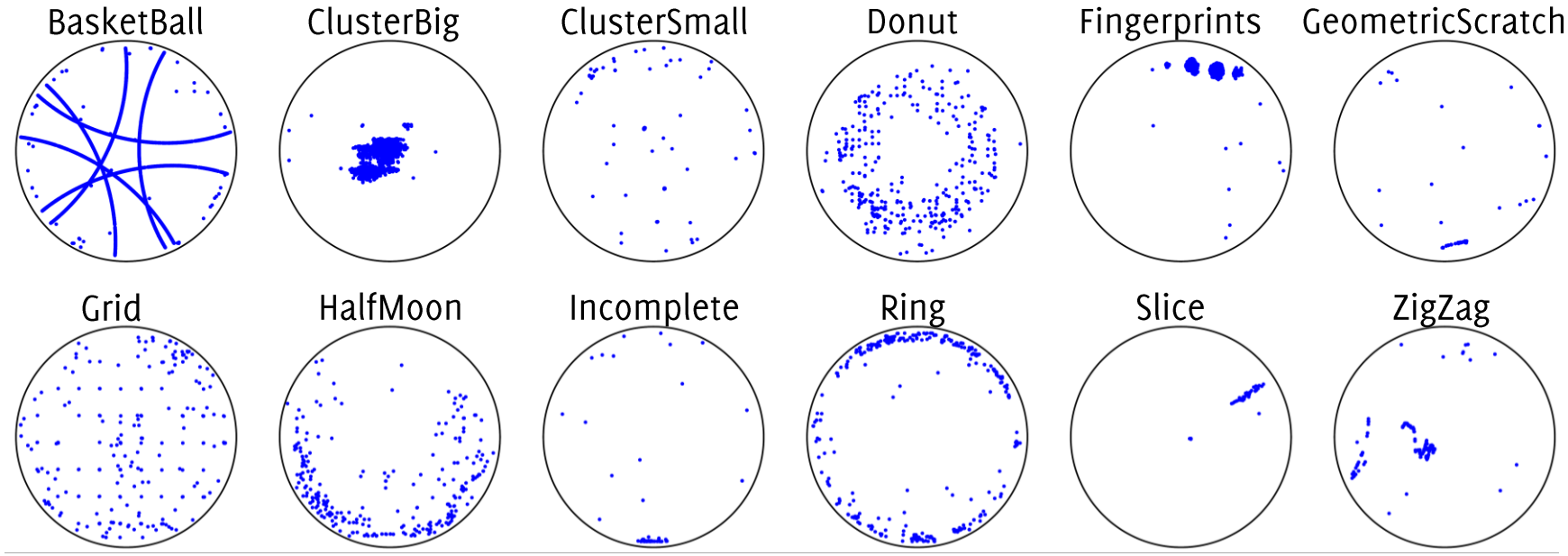


Silicon Wafer

Inspection Tool

X	Y
1863	709
1346	3067
2858	17095
3392	3508
...	...
282	6532
892	18888
4427	9873

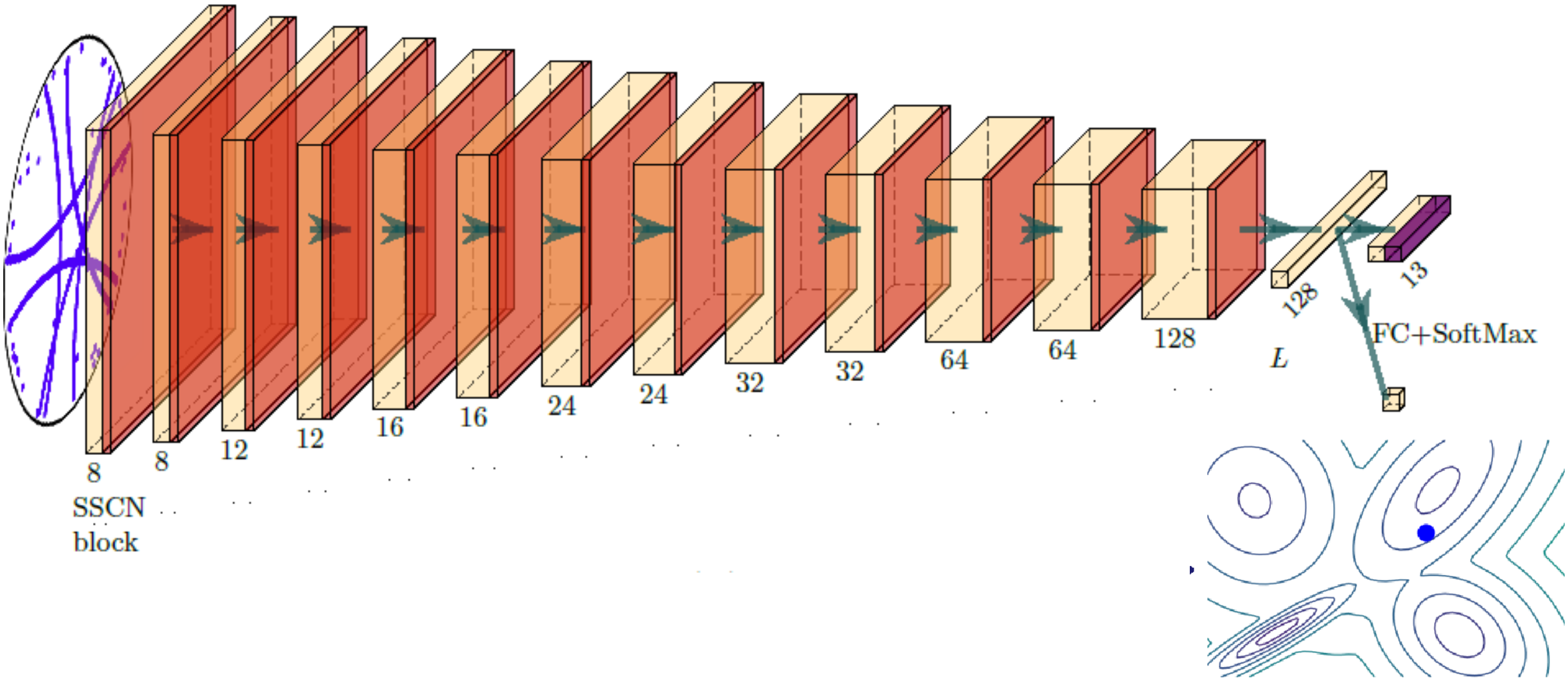




Di Bella, Carrera, Rossi, Fragneto, Boracchi *Wafer Defect Map Classification Using Sparse Convolutional Neural Networks* ICIAP19

Our SSCN architecture

Perfoms both Classification and Novel-class detection



L. Frittoli, D. Carrera, B. Rossi, P. Fragneto *"Deep Open-Set Recognition for Silicon Wafer Production Monitoring"* under review

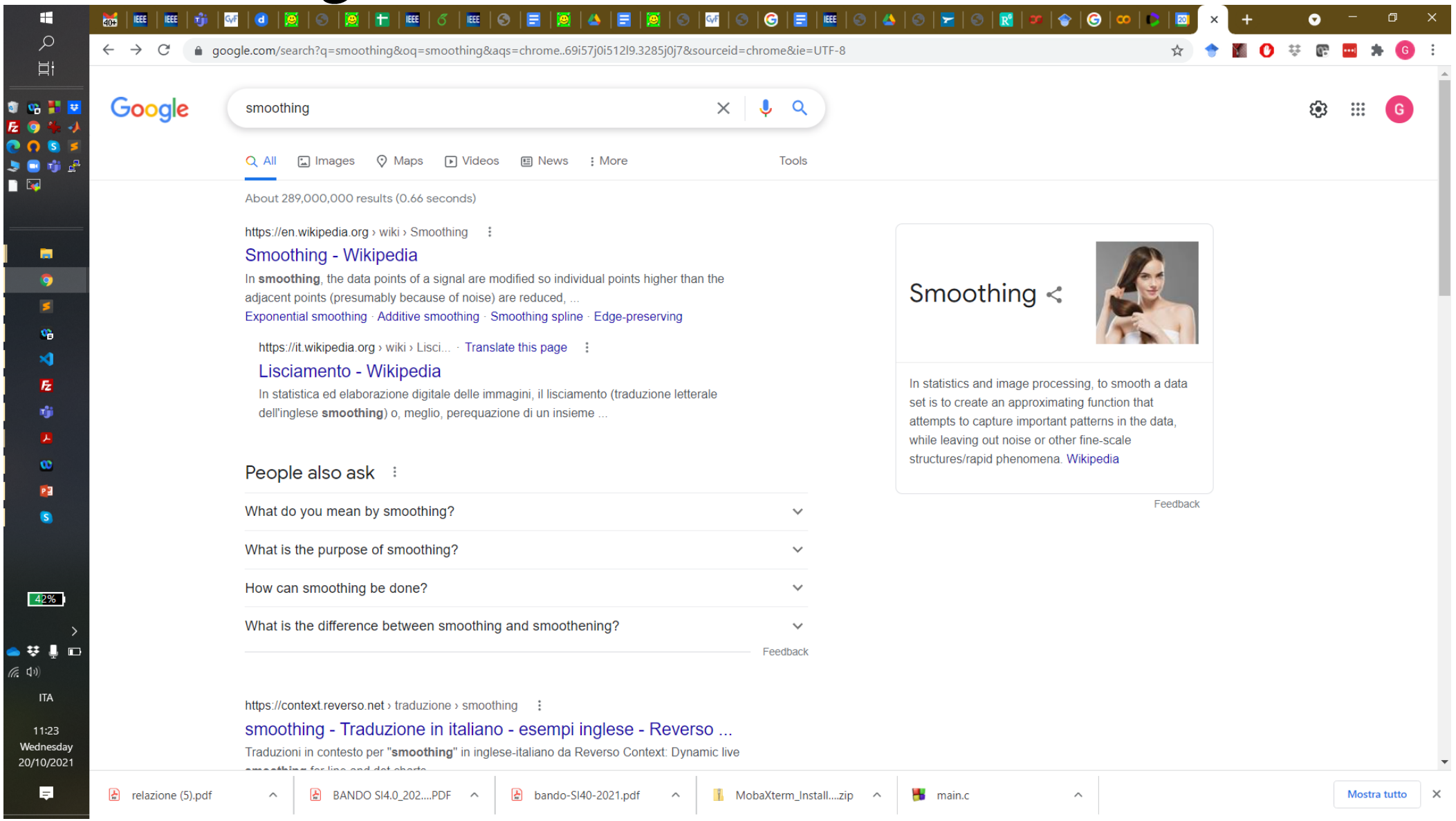
Image Captioning

"little girl is eating piece of cake."

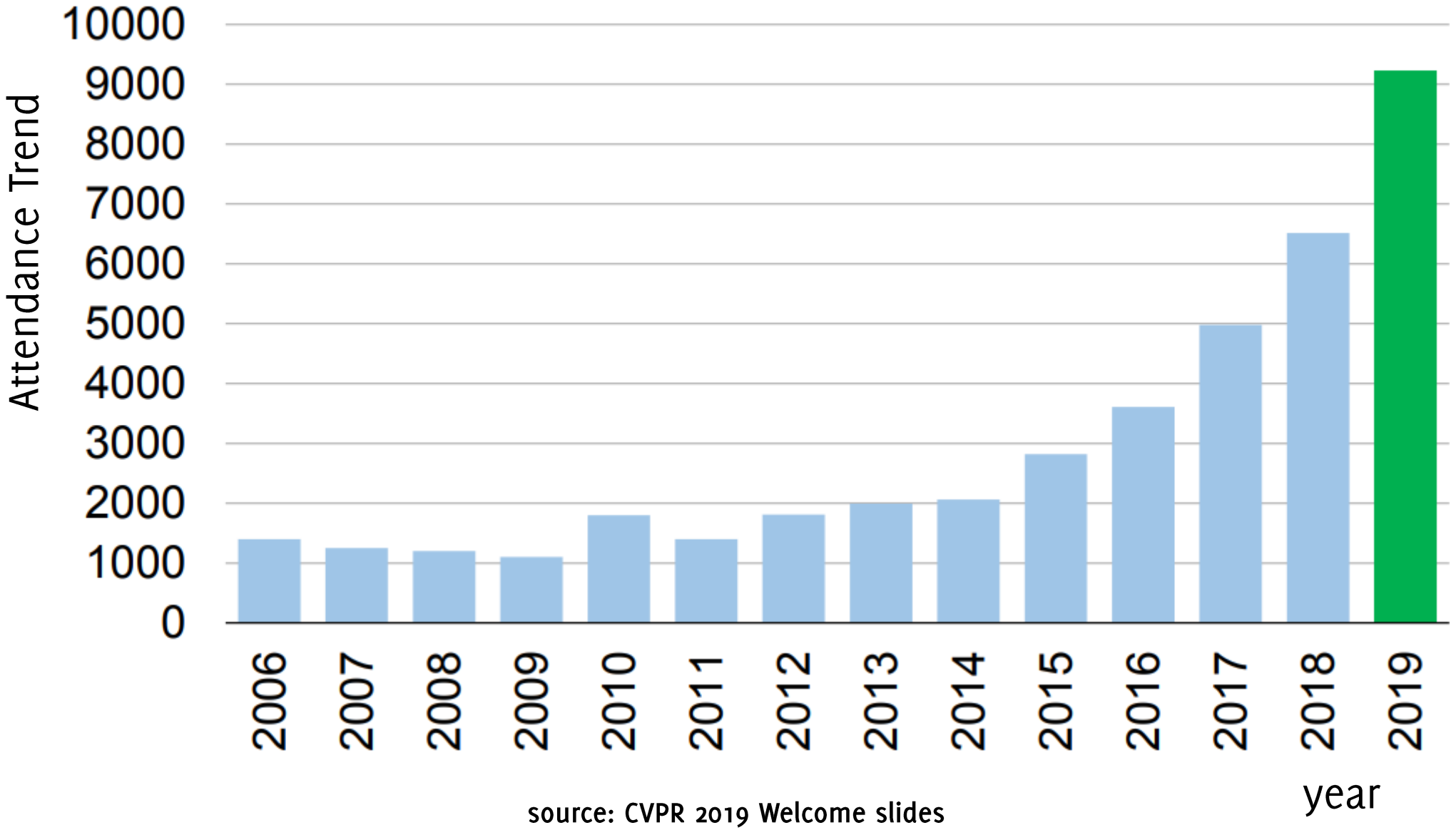
"black cat is sitting on top of suitcase."

Andrej Karpathy, Li Fei-Fei "Deep Visual-Semantic Alignments for Generating Image Descriptions" CVPR 2015

Even though sometimes it fails

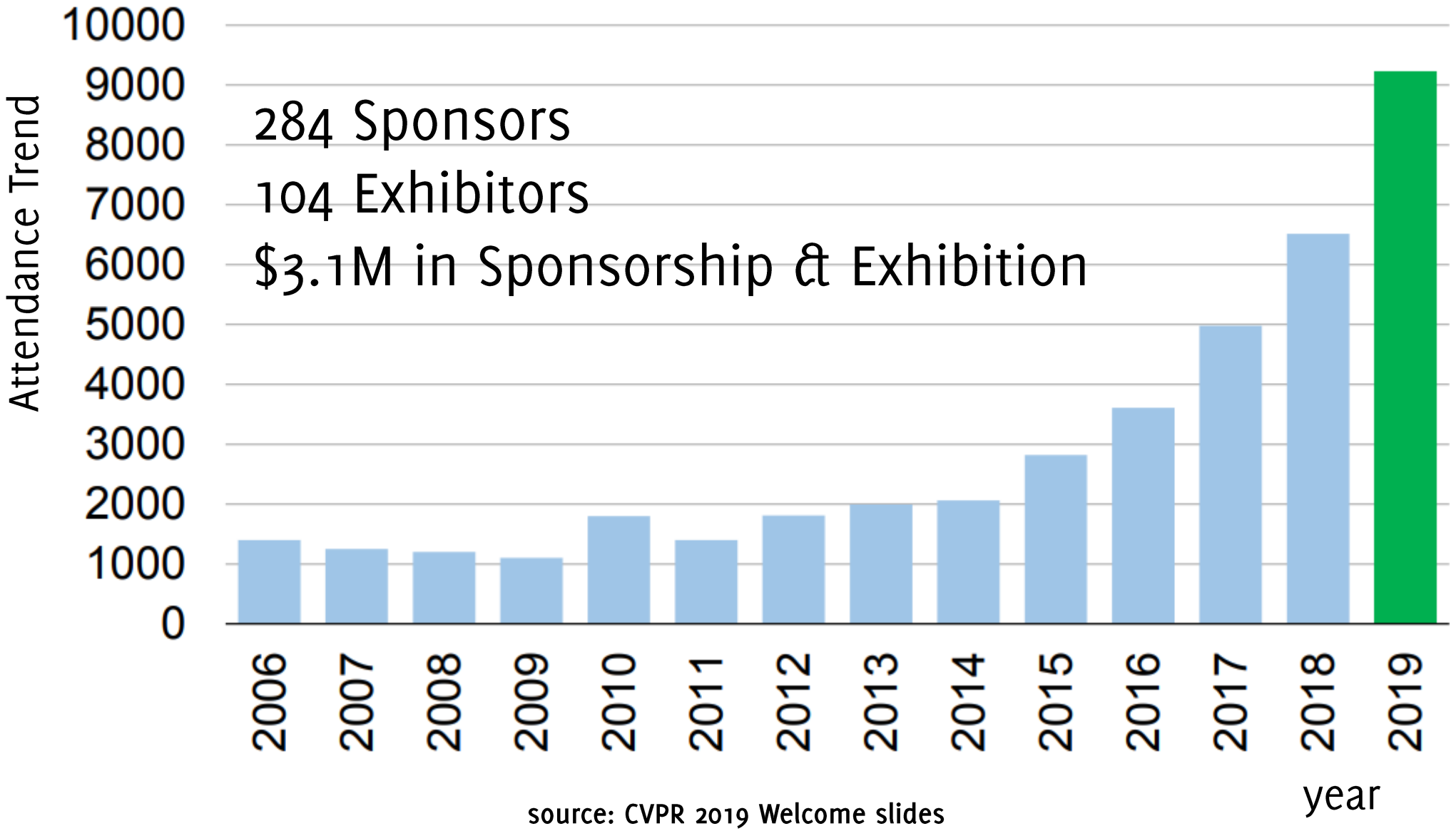


CVPR



source: CVPR 2019 Welcome slides

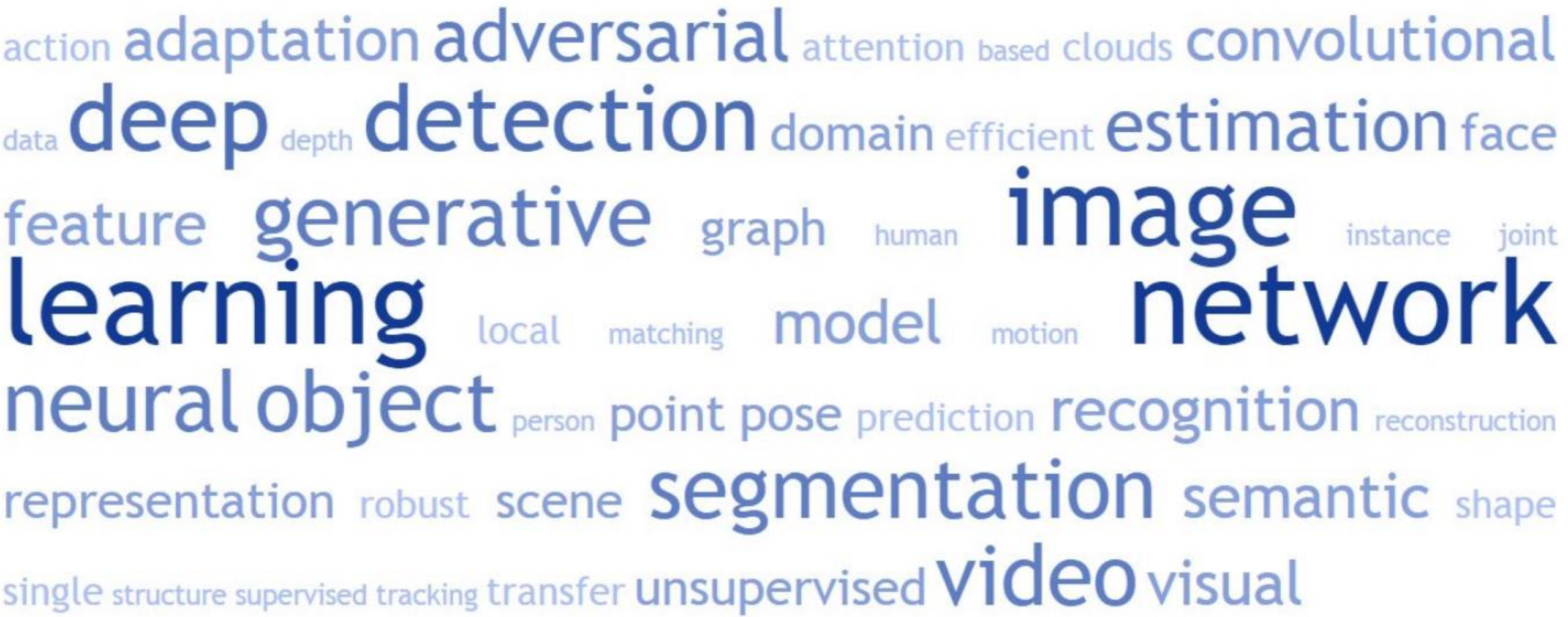
CVPR



source: CVPR 2019 Welcome slides

1,294 papers in CVPR'19 (25.2% acceptance rate)

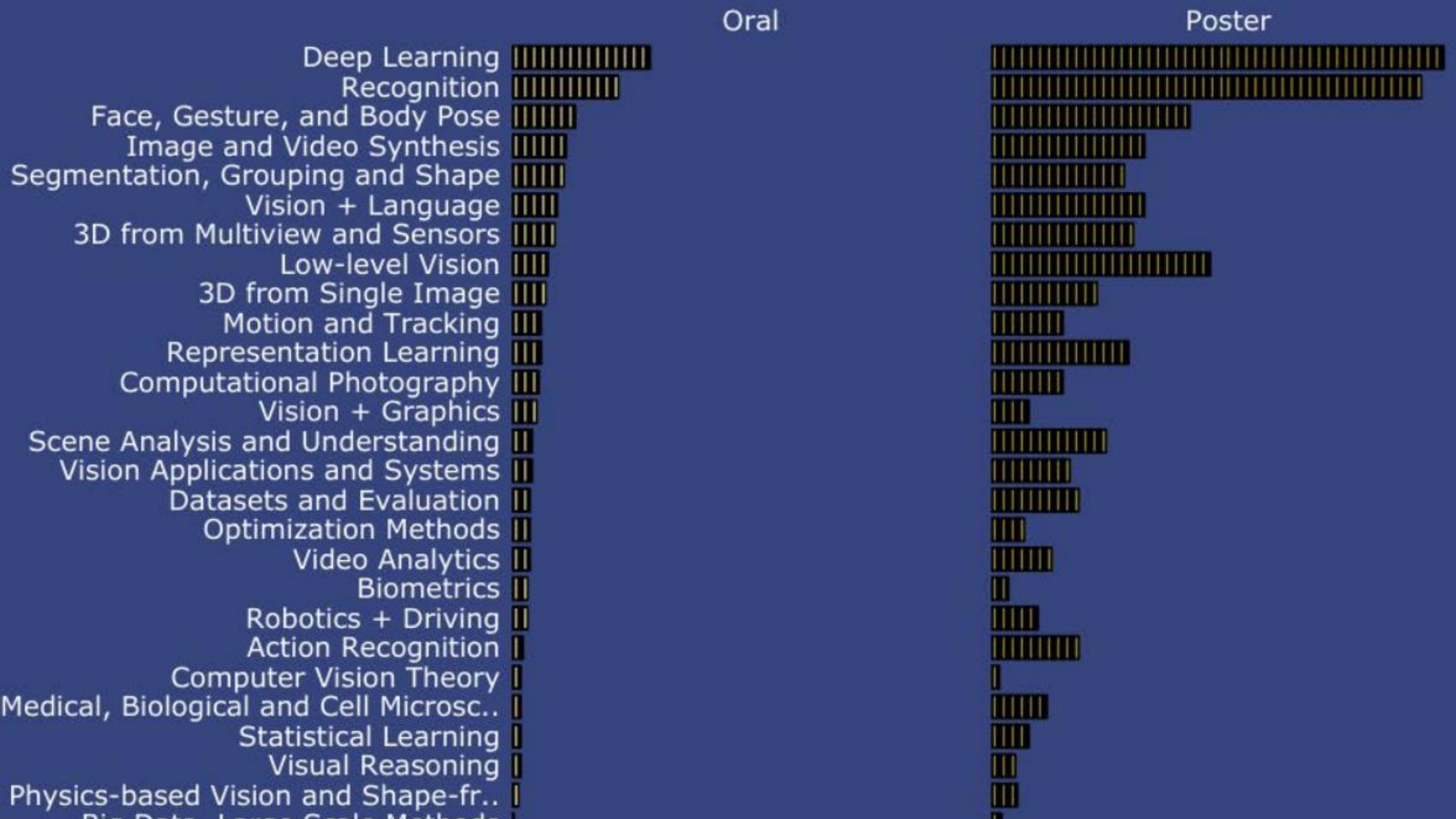
Acceptance rate is roughly even across topics



Lately, connection with ML

There has seen a dramatic change in CV:

- Once, most of techniques and algorithms **build upon a mathematical/statistical** description of **images**
- Nowadays**, machine-learning** methods are much more popular


Oral
Poster
Deep Learning
Recognition
Face, Gesture, and Body Pose
Image and Video Synthesis
Segmentation, Grouping and Shape
Vision + Language
3D from Multiview and Sensors
Low-level Vision
3D from Single Image
Motion and Tracking
Representation Learning
Computational Photography
Vision + Graphics
Scene Analysis and Understanding
Vision Applications and Systems
Datasets and Evaluation
Optimization Methods
Video Analytics
Biometrics
Robotics + Driving
Action Recognition
Computer Vision Theory
Medical, Biological and Cell Microsc..
Statistical Learning
Visual Reasoning
Physics-based Vision and Shape-fr..

Images

Photo Credits: Andrea Sanfilippo

RGB Images

$I \in \mathbb{R}^{R \times C \times 3}$

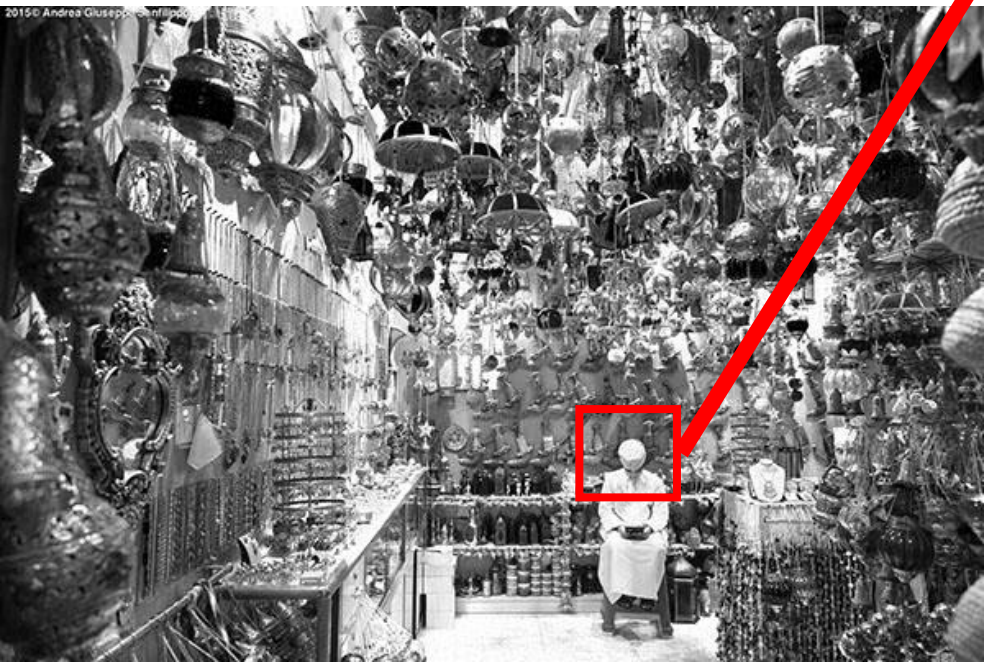
$R \in \mathbb{R}^{R \times C}$

$G \in \mathbb{R}^{R \times C}$

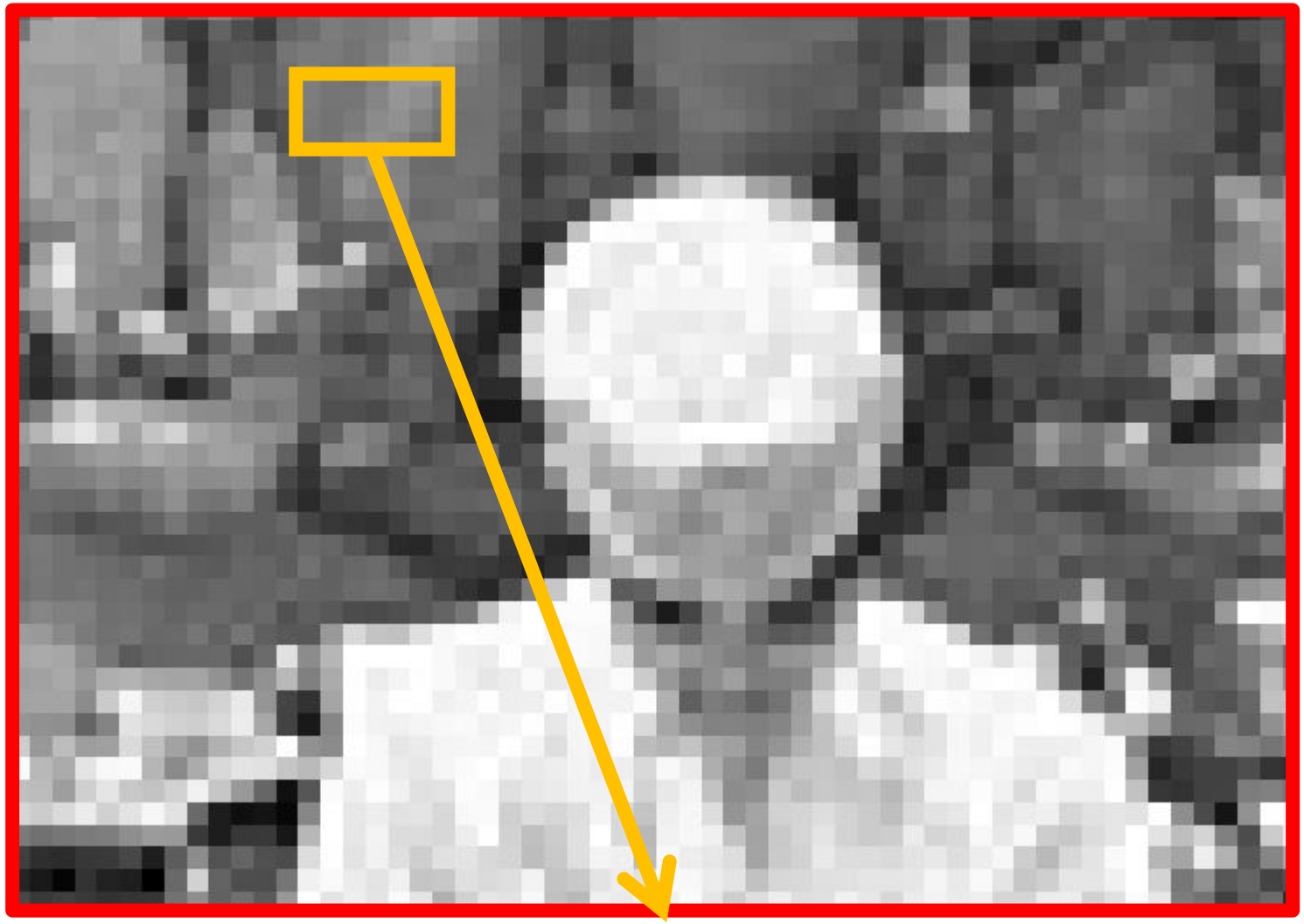
$B \in \mathbb{R}^{R \times C}$

RGB Images

Images are saved by encoding each color information in 8 bits. So images are rescaled and casted in [0,255]



$$R \in \mathbb{R}^{R \times C}$$



123	122	134	121	132
122	121	125	132	124
119	127	137	119	139

RGB Images

[0, 205, 155]

[15, 17, 19]

[230,234, 233]

[253, 5, 6]

This is an RGB triplet
R = 253; G = 2; B = 6

[106,124,138]

The input of our classifier!

Three dimensional arrays $I \in \mathbb{R}^{R \times C \times 3}$

```
from skimage.io import imread

# Read the image
I = imread('bazar.jpg')

# Extract the color channels of the image
R = I[:, :, 0]
G = I[:, :, 1]
B = I[:, :, 2]
```

When loaded in memory, image sizes are much larger than on the disk where images are typically compressed (e.g. in jpeg format)

Videos

Higher dimensional images

Videos are sequences of images (frames)

If a frame is

$$I \in \mathbb{R}^{R \times C \times 3}$$

a video of T frames is

$$V \in \mathbb{R}^{R \times C \times 3 \times T}$$

```
print(V.shape)
  (144, 180, 3, 30)
```

In this example: $R = 144, C = 180$, thus these 5 color frames contains: 388.800 values in [0,255], thus in principle, 388 KB

Dimension Increases very quickly

Without compression: 1Byte per color per pixel

1 frame in full HD: $R = 1080, C = 1920 \approx 6MB$

1 sec in full HD (24fps) $\approx 150MB$

Fortunately, visual data are very redundant, thus compressible

This has to be taken into account when you design a Machine learning algorithm to be used on images

- e.g. while training a neural network these information are not compressed!

Higher dimensional images

Multispectral imaging and remote sensing

Giacomo Boracchi

Higher dimensional Images

These images are stacking multiple layers as color-planes

Multi-spectral or Hyper-spectral images:

each band covers a certain wavelength that is meaningful for interpretation soil in areal images

- 0.45-0.52 µm Blue-Green
- 0.52-0.60 µm Green
- 0.63-0.69 µm Red
- 0.76-0.90 µm Near IR
- 1.55-1.75 µm Mid-IR
- 10.40-12.50 µm Thermal IR
- 2.08-2.35 µm Mid-IR

Classification of multispectral images: infer the soil coverage from the vaues in the different spectral bands

In hyperspectral images the number of bands becomes larger and you get a whole energy spectrum in each pixel

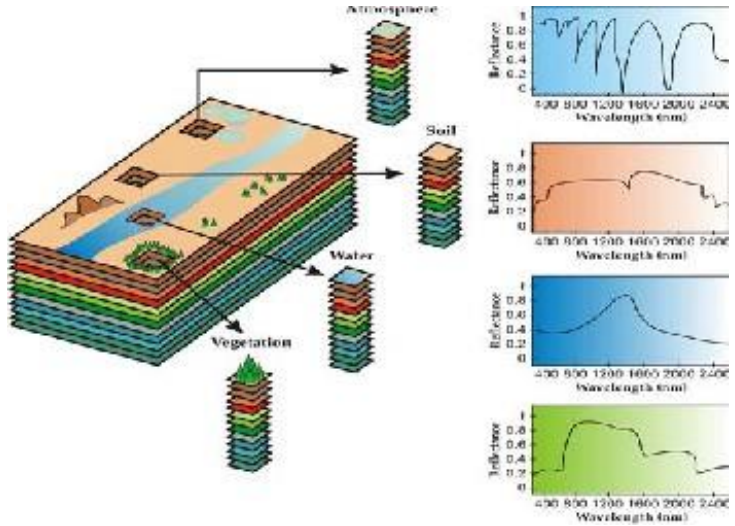
Higher dimensional Images

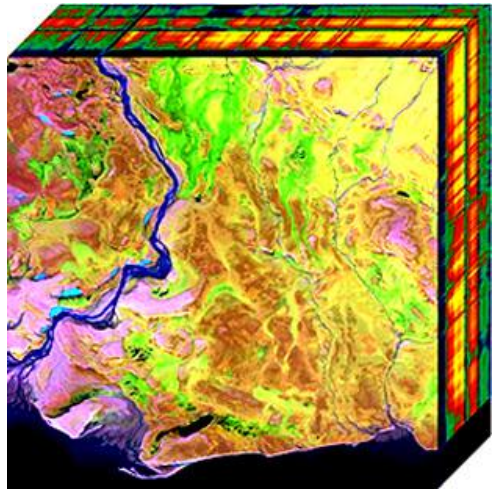
These images are stacking multiple layers as color-planes

Multi-spectral or Hyper-spectral images:

each band covers a certain wavelength that is meaningful for interpretation soil in areal images

- 0.45-0.52 µm Blue-Green
- 0.52-0.60 µm Green
- 0.63-0.69 µm Red
- 0.76-0.90 µm Near IR
- 1.55-1.75 µm Mid-IR
- 10.40-12.50 µm Thermal IR
- 2.08-2.35 µm Mid-IR





In hyperspectral images the number of bands becomes larger and you get a whole energy spectrum in each pixel

Band 1

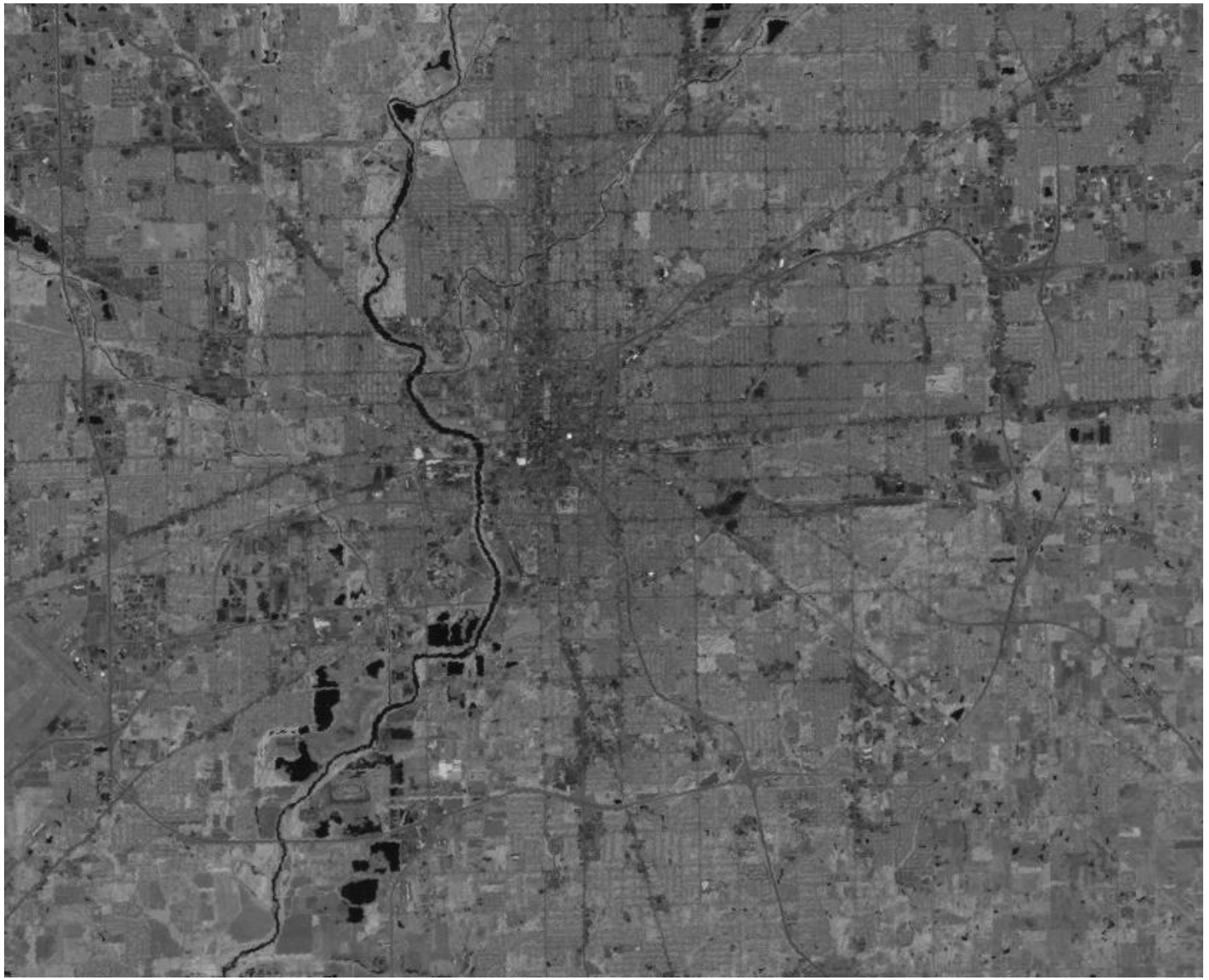
Giacomo Boracchi

Band 2

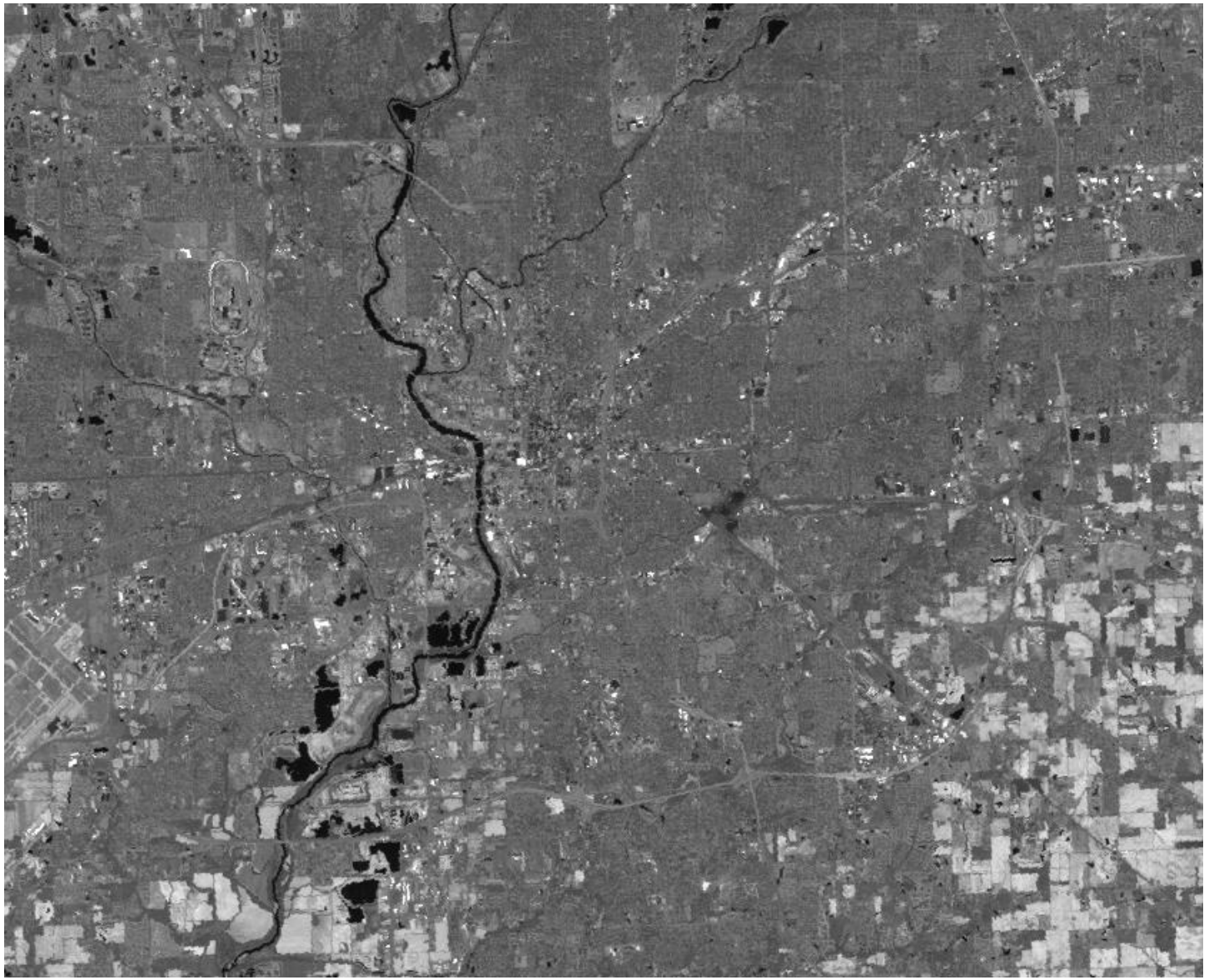
Giacomo Boracchi

Band 3

Band 4



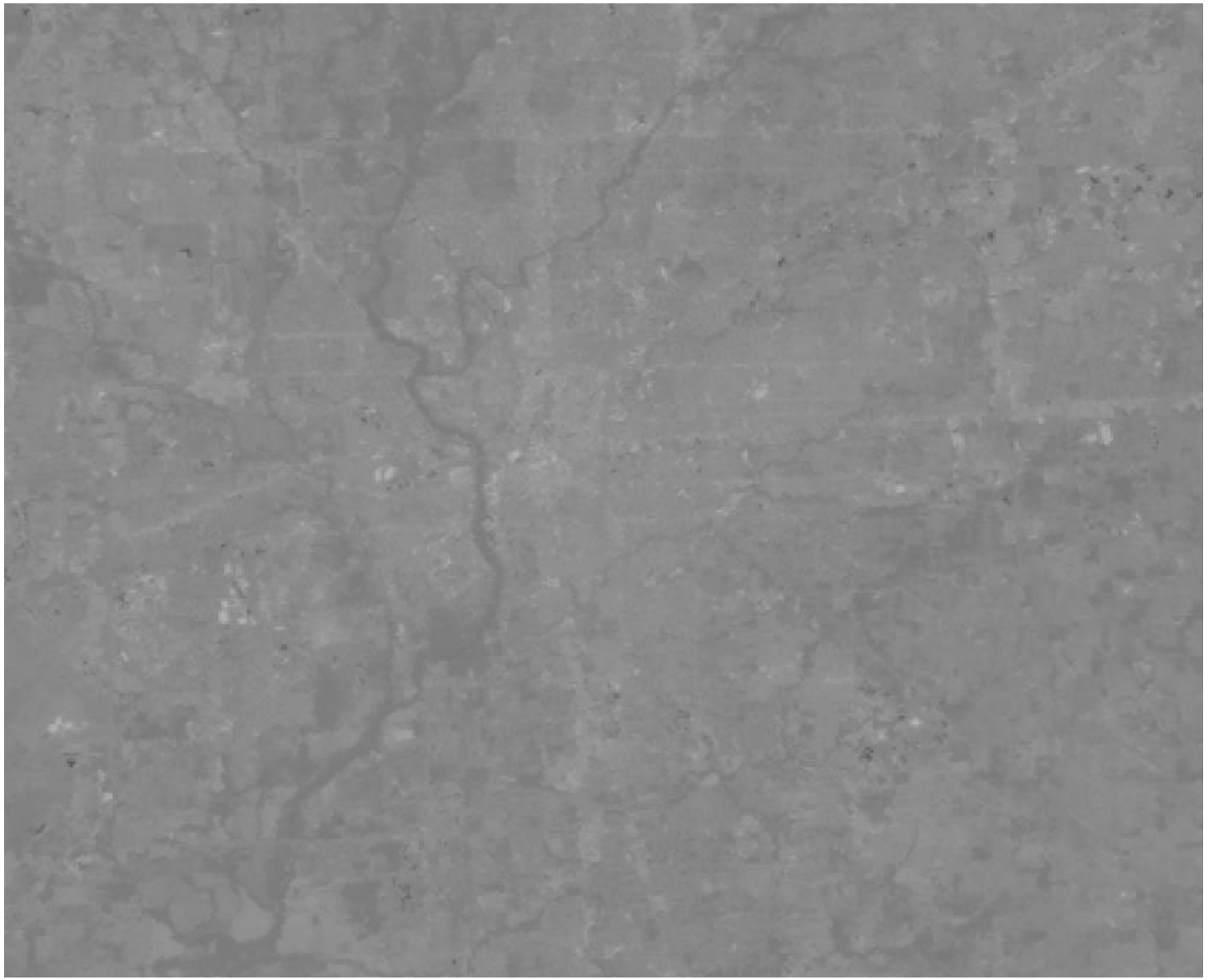
Band 5



Band 6

Giacomo Boracchi

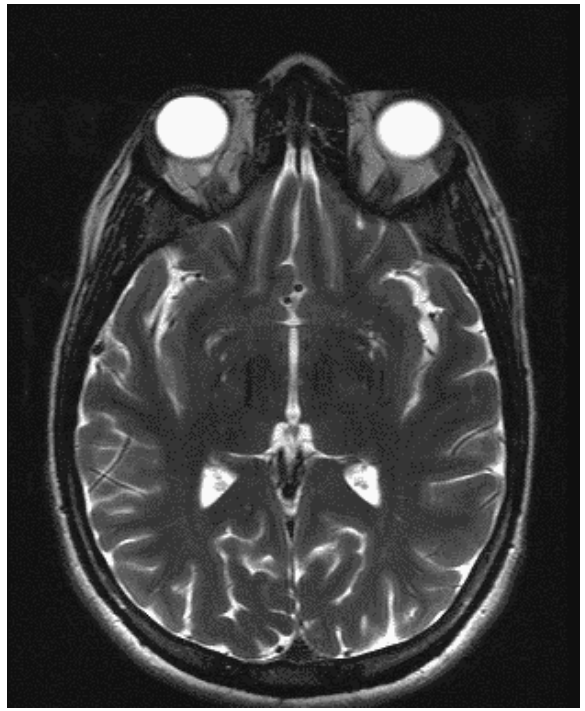
Band 7

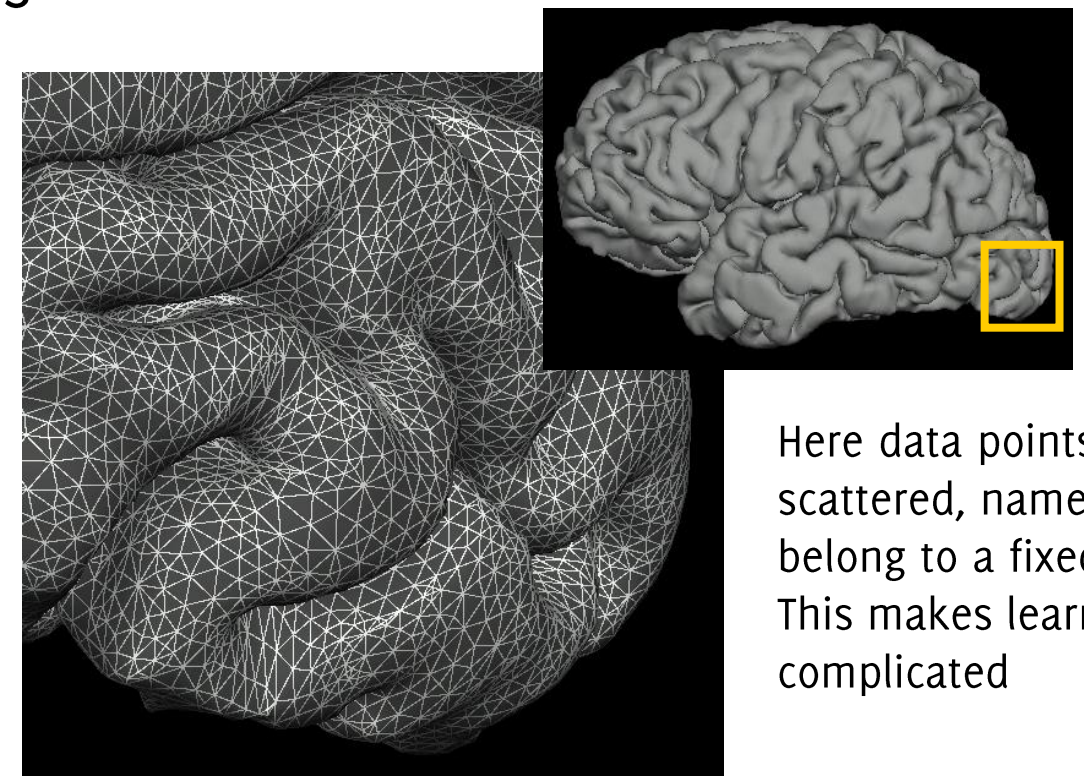


MRI and TAC

(Structural) MRI provide a 3D stack of grayscale images that are analyzing the body at different depth levels

These can be used to perform 3D reconstruction of bones, organs and membranes, e.g. cortical sufaces





Here data points are scattered, namely they do not belong to a fixed grid.
This makes learning more complicated

Local (Spatial) Transformations: Correlation

Most important image processing operations for classification problems

Giacomo Boracchi

Local (Spatial) Transformation

In general, these can be written as

$$G(r,c) = T_U[I](r,c)$$

Where

- I is the input image to be transformed
- G is the output
- T_U: $\mathbb{R}^3 \to \mathbb{R}^3$ or T_U: $\mathbb{R}^3 \to \mathbb{R}$ is a function transforming the image
- U is a neighbourhood, identifies a region of the image that will concur in the output definition

T operates on I "around" U

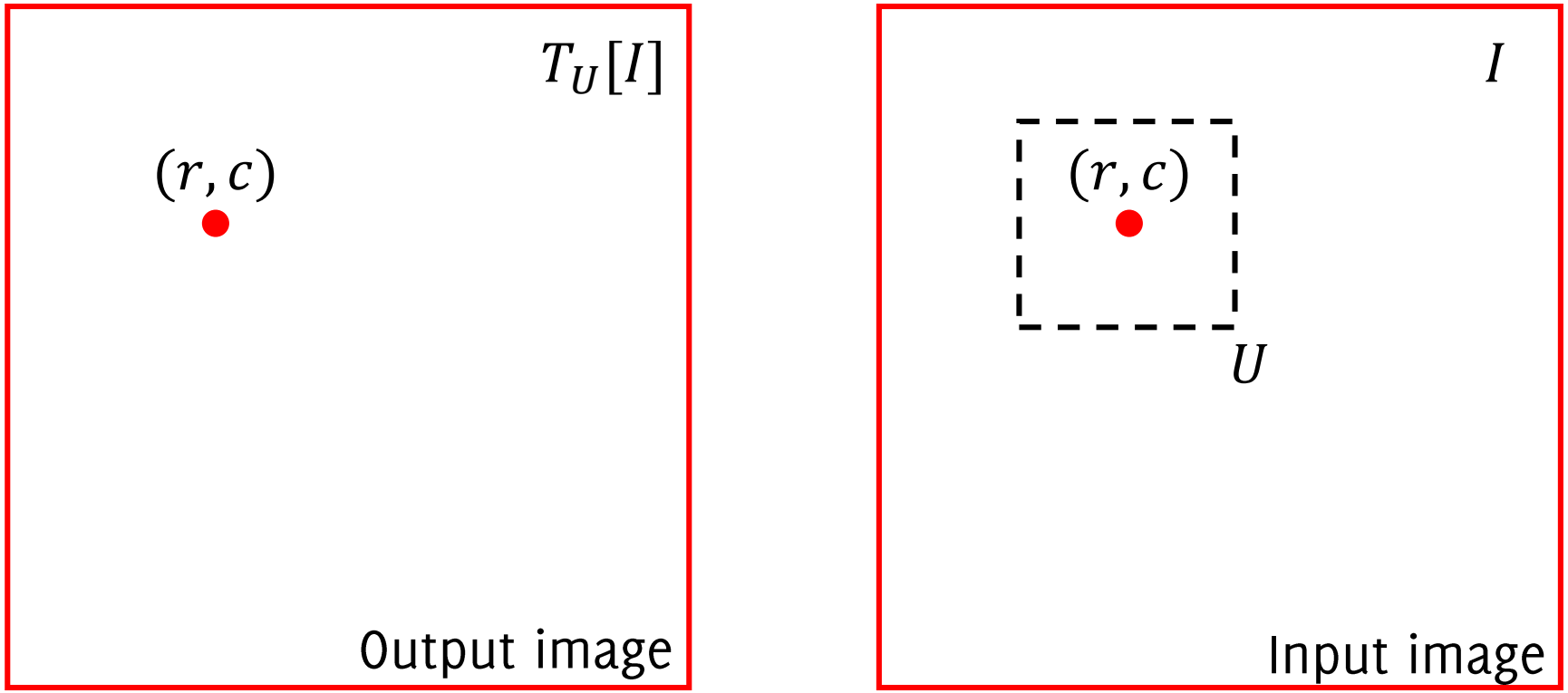
The output at pixel (r,c) i.e., $T_U[I](r,c)$ is defined by all pixel

$$\{I(r+u, c+v), \qquad (u,v) \in U\}$$

Such that U is described by a set of «displacements»

Spatial Filters

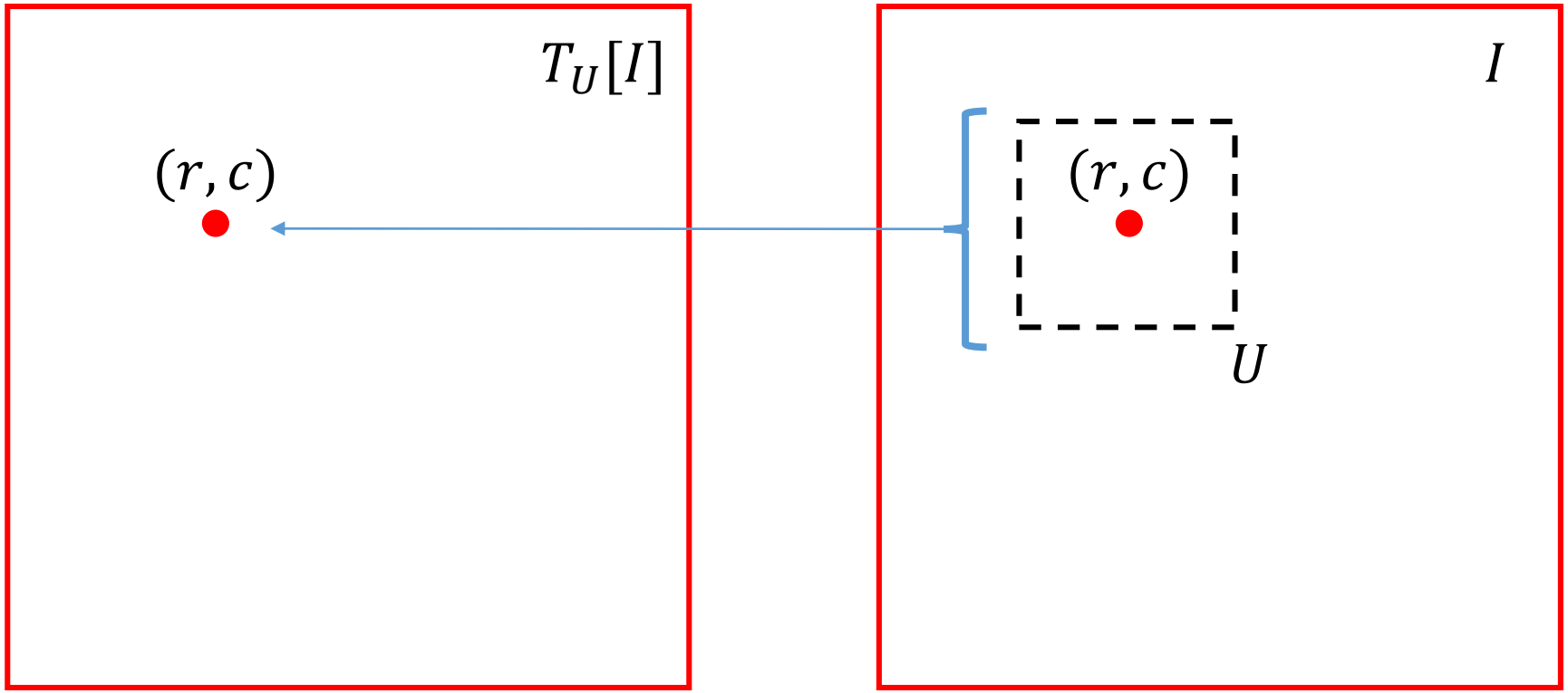
The dashed square represents $\{I(r+u, c+v), \quad (u,v) \in U\}$



- The location of the output does not change
- The operation is repeated for each pixel
- T can be either linear or nonlinear

Spatial Filters

The dashed square represents $\{I(r+u, c+v), \quad (u,v) \in U\}$

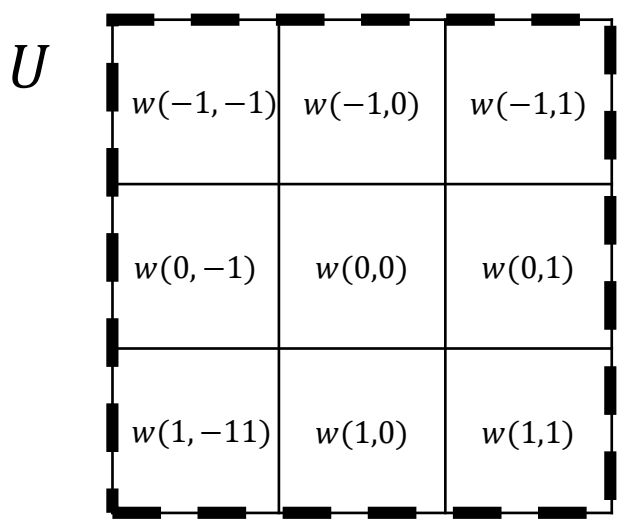


- The location of the output does not change
- The operation is repeated for each pixel
- T can be either linear or nonlinear

Linear Spatial Filters

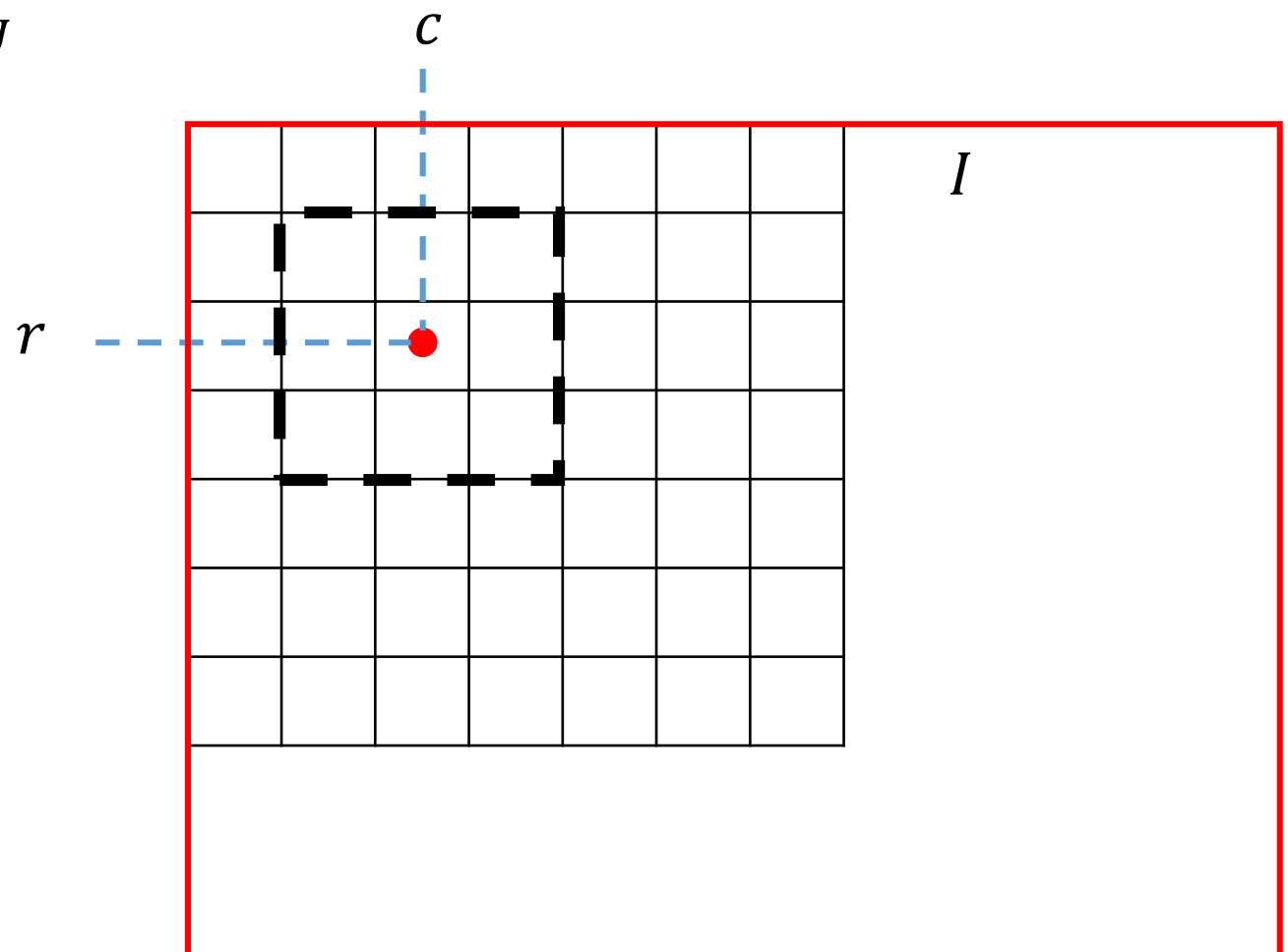
Linear Transformation: Linearity implies that

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



We can **consider weights** as an image, or **a filter** w

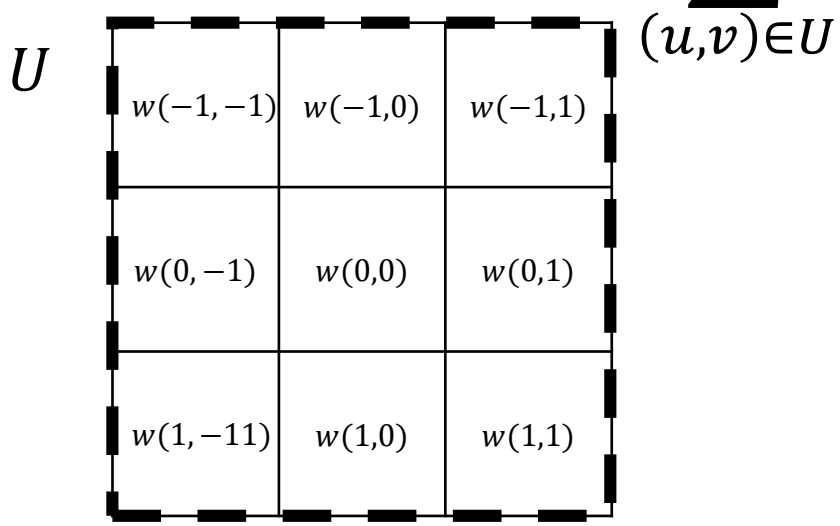
The filter w entirely defines this operation



Linear Spatial Filters

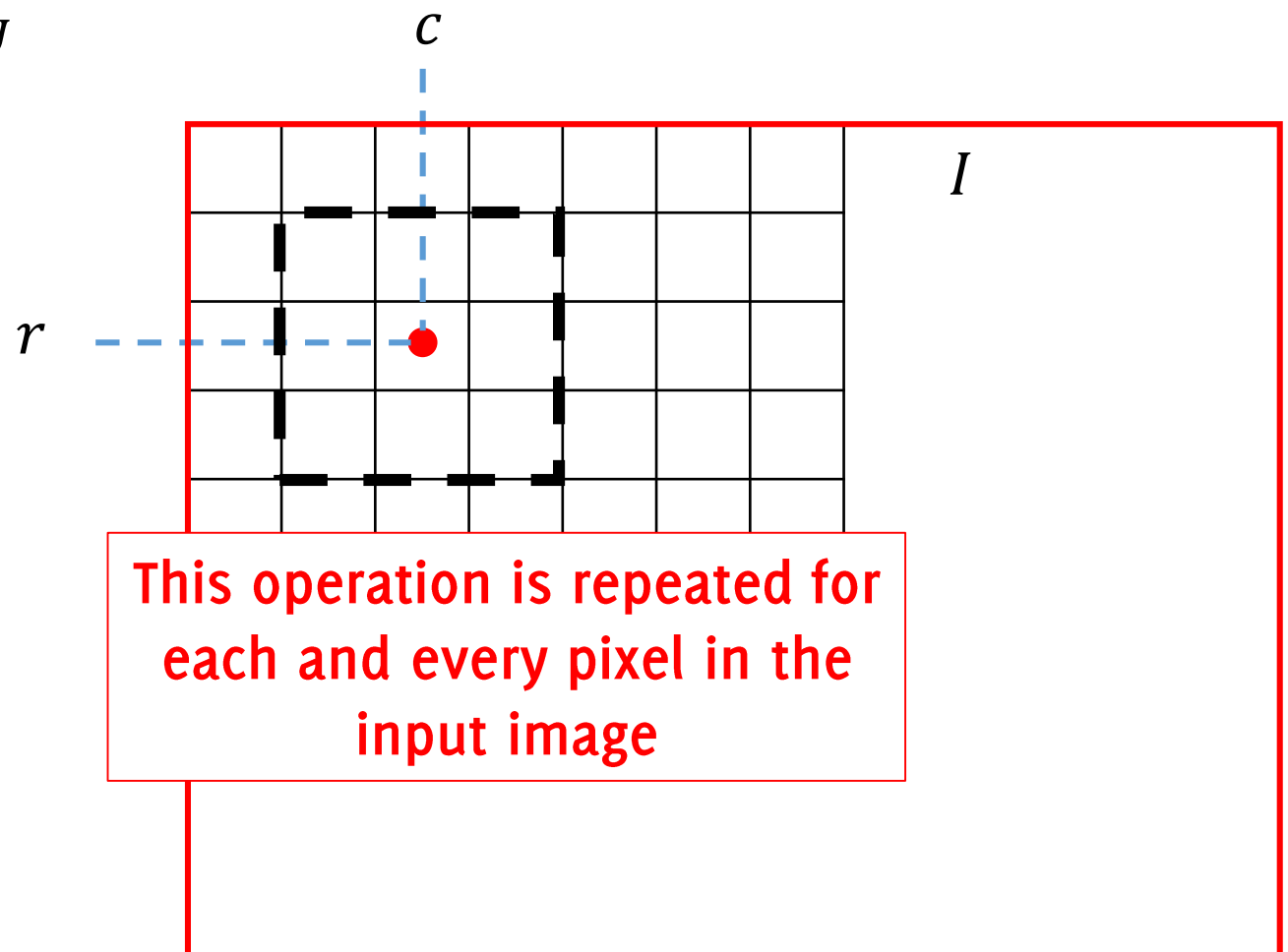
Linear Transformation: Linearity implies that

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

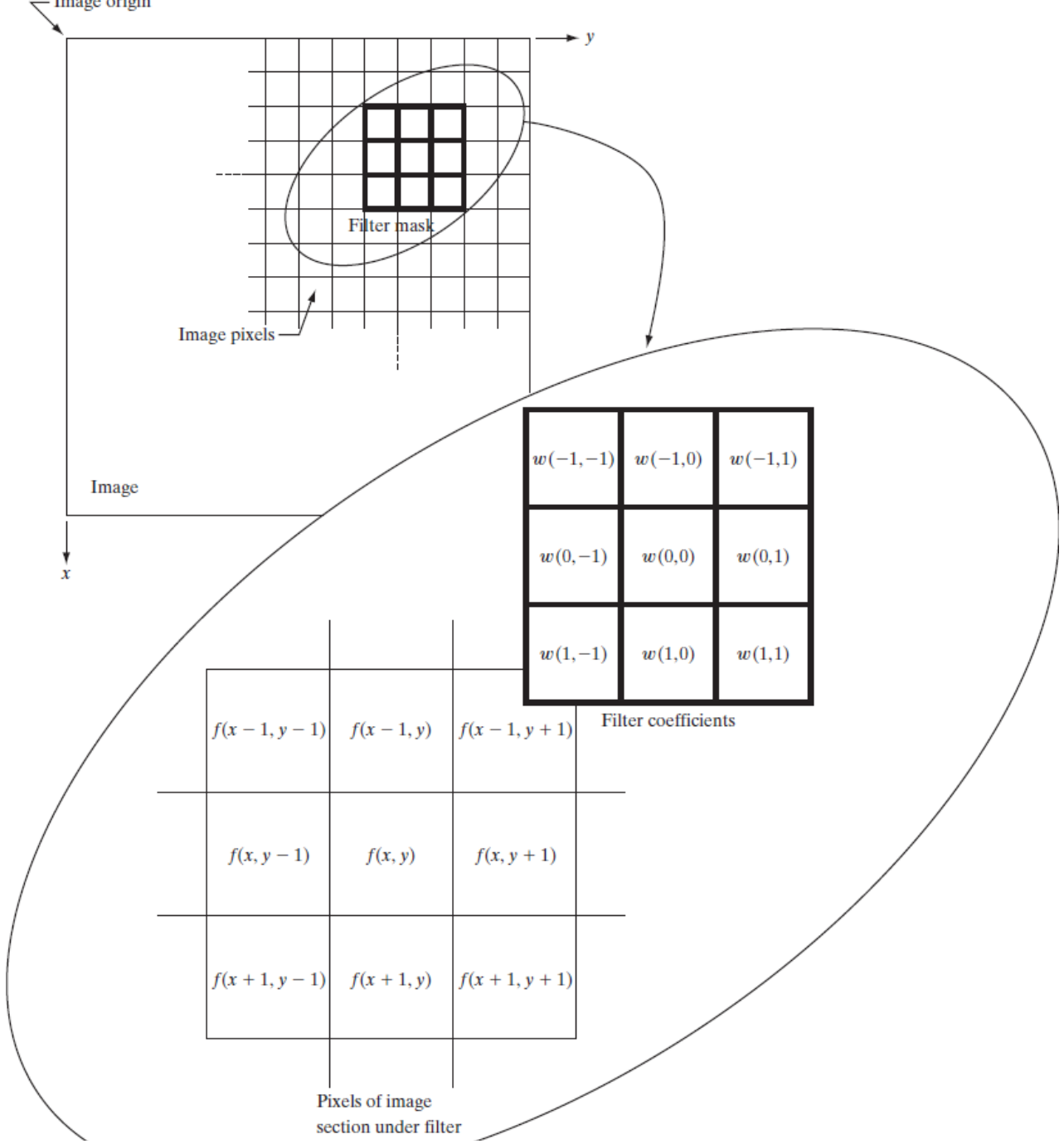


We can consider weights as an image, or **a filter** h

The filter h entirely defines this operation



This operation is repeated for each and every pixel in the input image



Gonzalez and Woods «Digital image Processing», Prentice Hall;, 3° edition

Correlation

The **correlation** among a filter h and an image is defined as

$$(I \otimes w) = \sum_{u=-L}^{L} \sum_{v=-L}^{L} w(u,v) * I(r+u, c+v)$$

where the filter w is of size $(2L+1) \times (2L+1)$

Correlation

The **correlation** among a filter h and an image is defined as

$$(I \otimes w) = \sum_{u=-L}^{L} \sum_{v=-L}^{L} w(u,v) * I(r+u, c+v)$$

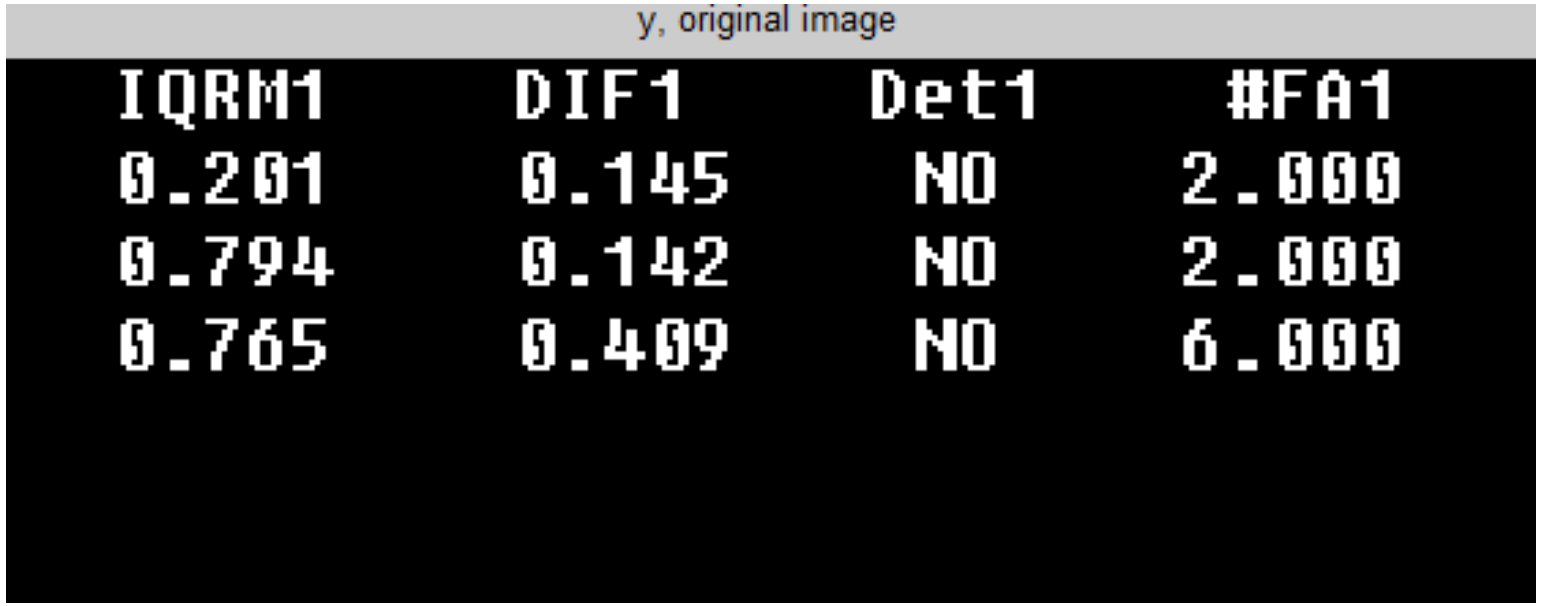
where the filter w is of size $(2L+1) \times (2L+1)$

```
from scipy import signal


# correlation
T_corr = signal.correlate(I, w)
```

Correlation for BINARY target matching

I w



$\otimes$

$=$

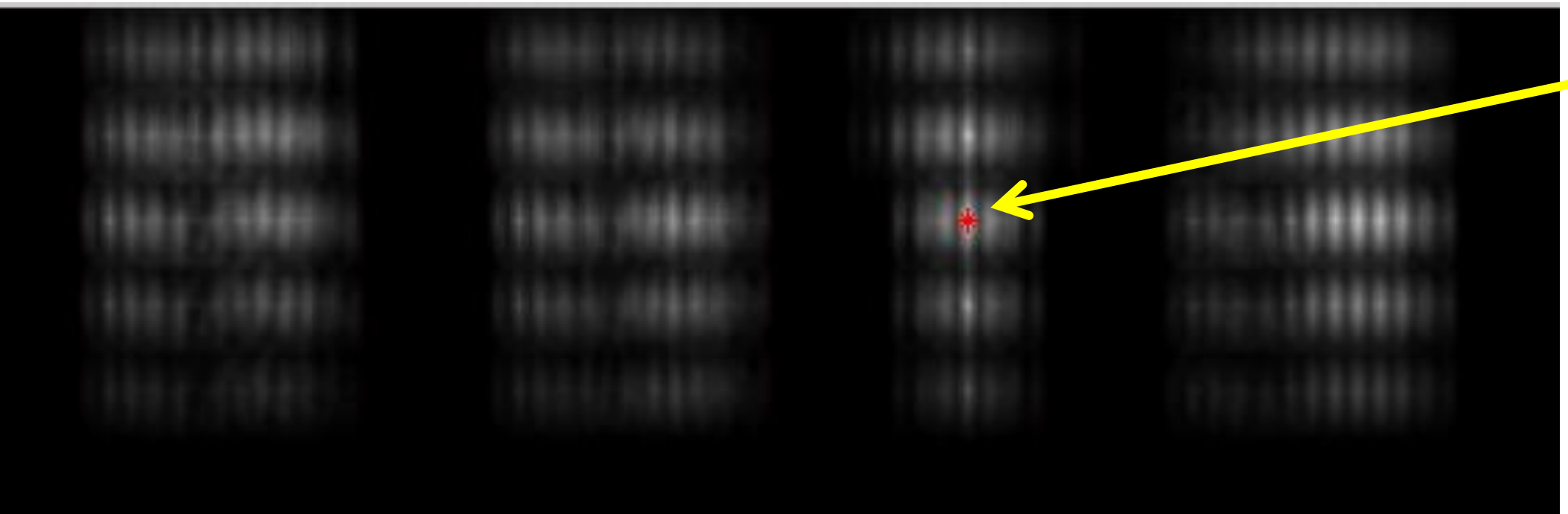
Easy to understand with binary images

Target used as a filter

IQRM1	DIF1	Det1	#FA1
0.201	0.145	NO	2.000
0.794	0.142	NO	2.000
0.765	0.409	NO	6.000

⊗

NO
NO
NO



The maximum is here

However…

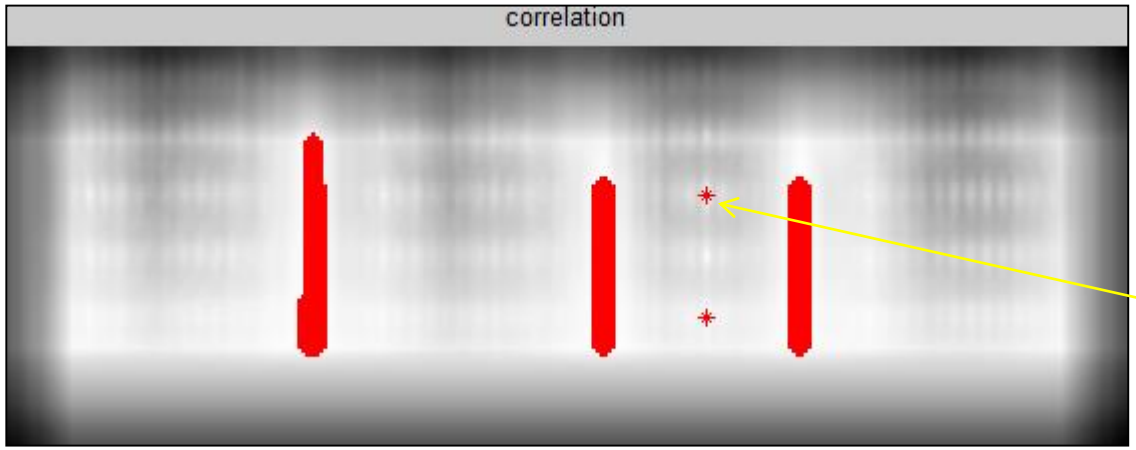
y, original image

IQRM1	DIF1	Det1	#FA1
0.201	0.145	NO	2.000
0.794	0.142	NO	2.000
0.765	0.409	NO	6.000

* template

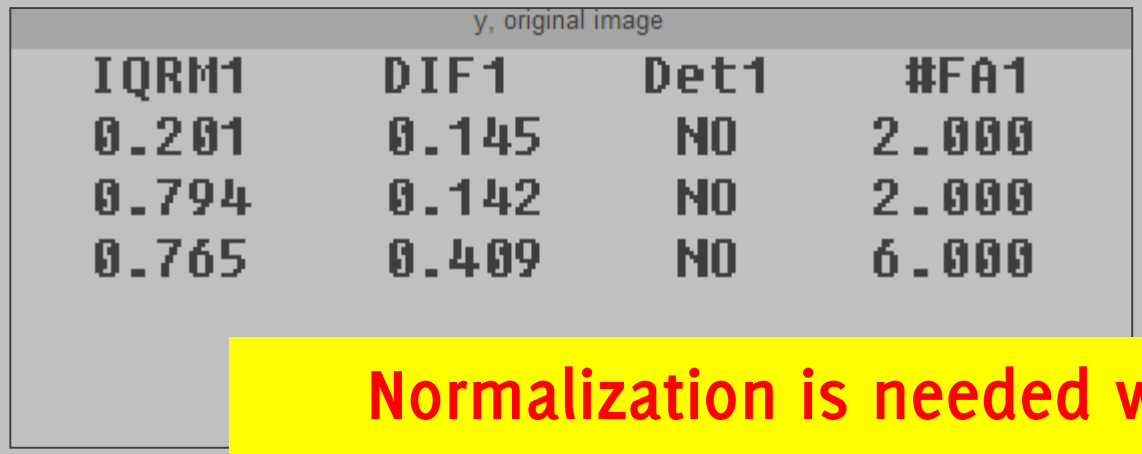
NO
NO
NO

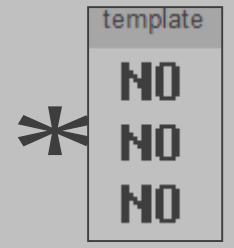
=



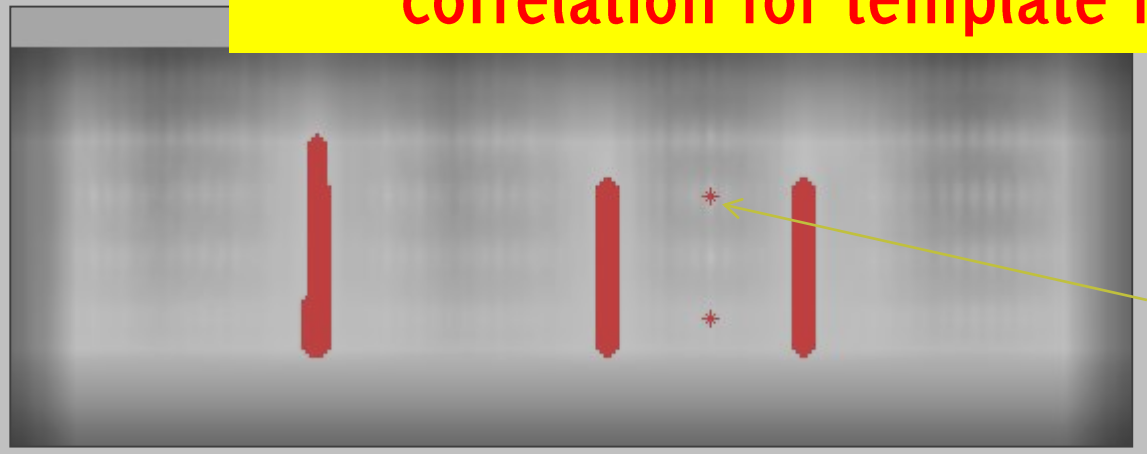
Each point in a white area is as big as the template achieve the maxium value (togheter with the perfect match)

However…





Normalization is needed when using correlation for template matching!



Each point in a white area is as big as the template achieve the maxium value (togheter with the perfect match)

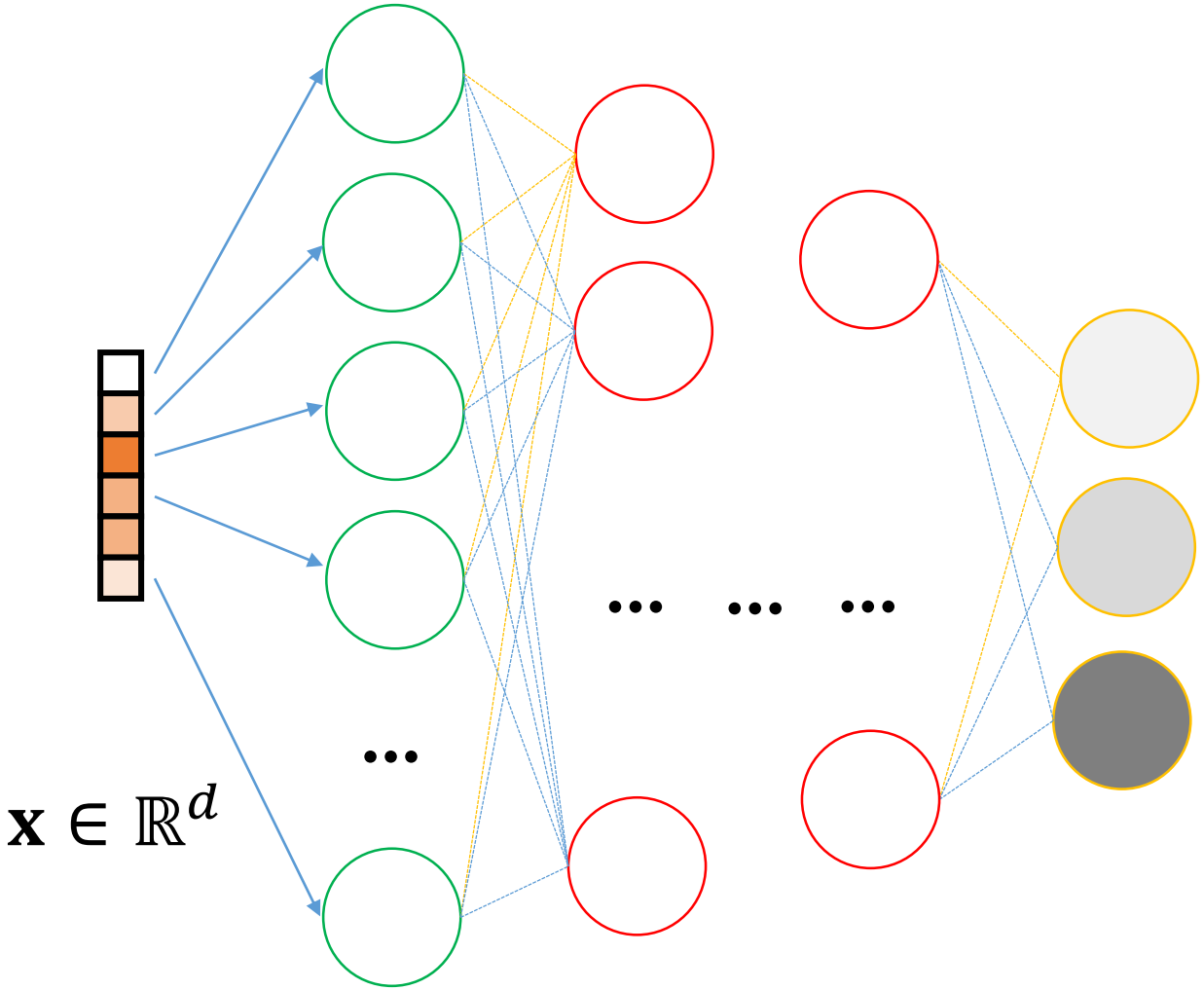
Linear Classifier

the basic building block for deep architectures

Giacomo Boracchi

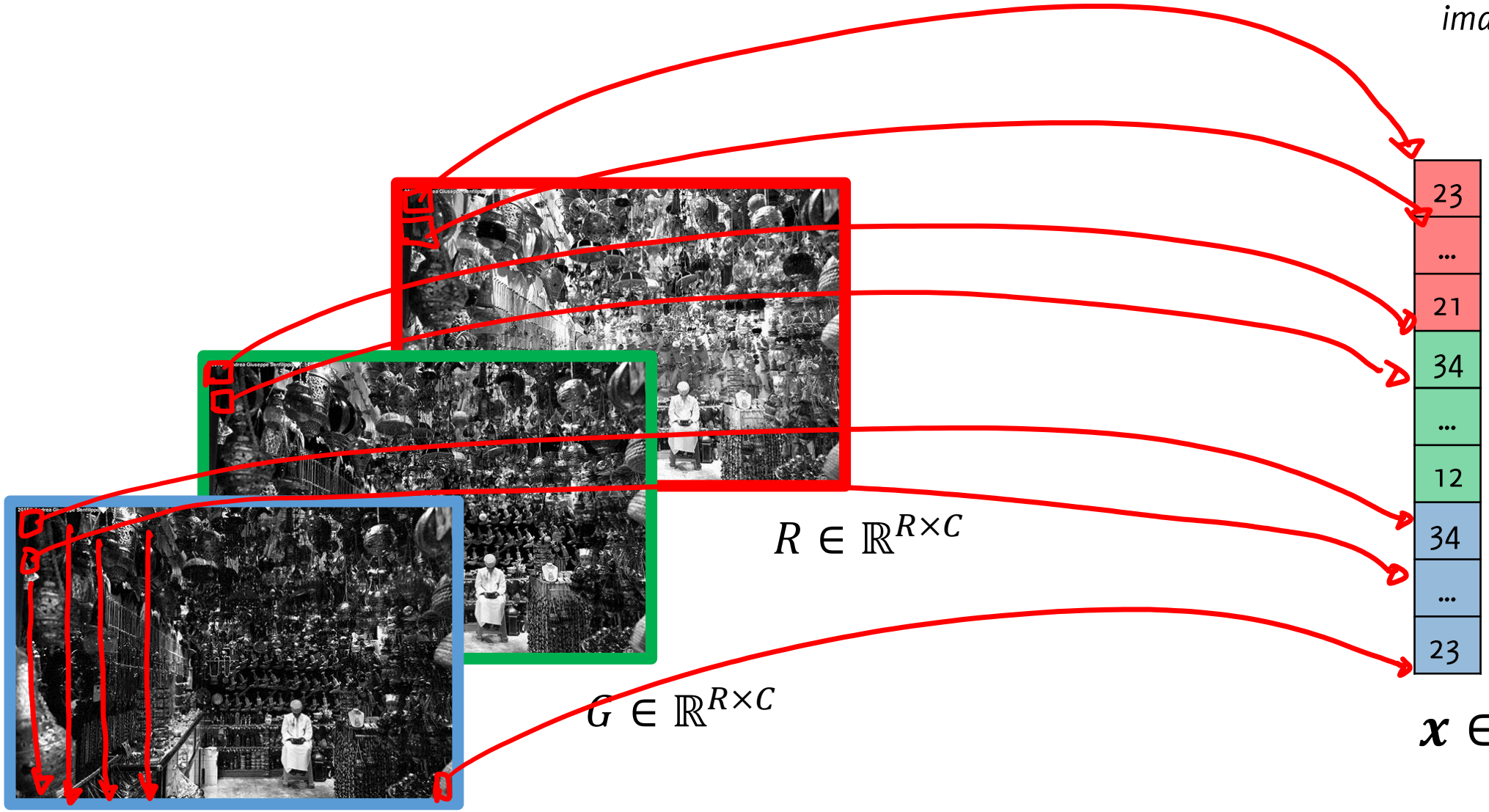
How to feed images to NN?

$I \in \mathbb{R}^{R \times C \times 3}$



Column-wise unfolding

Colors recall the color plane where images are from



$R \in \mathbb{R}^{R \times C}$

$G \in \mathbb{R}^{R \times C}$

$B \in \mathbb{R}^{R \times C}$

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

$d = R \times C \times 3$

Classification over the CIFAR-10 dataset

The CIFAR-10 dataset contains 60000 images:

Each image is 32x32 RGB

Images are in 10 classes

6000 images per class

$$\boldsymbol{x} \in \mathbb{R}^d, d = 3072$$

airplane

automobile

bird

cat

deer

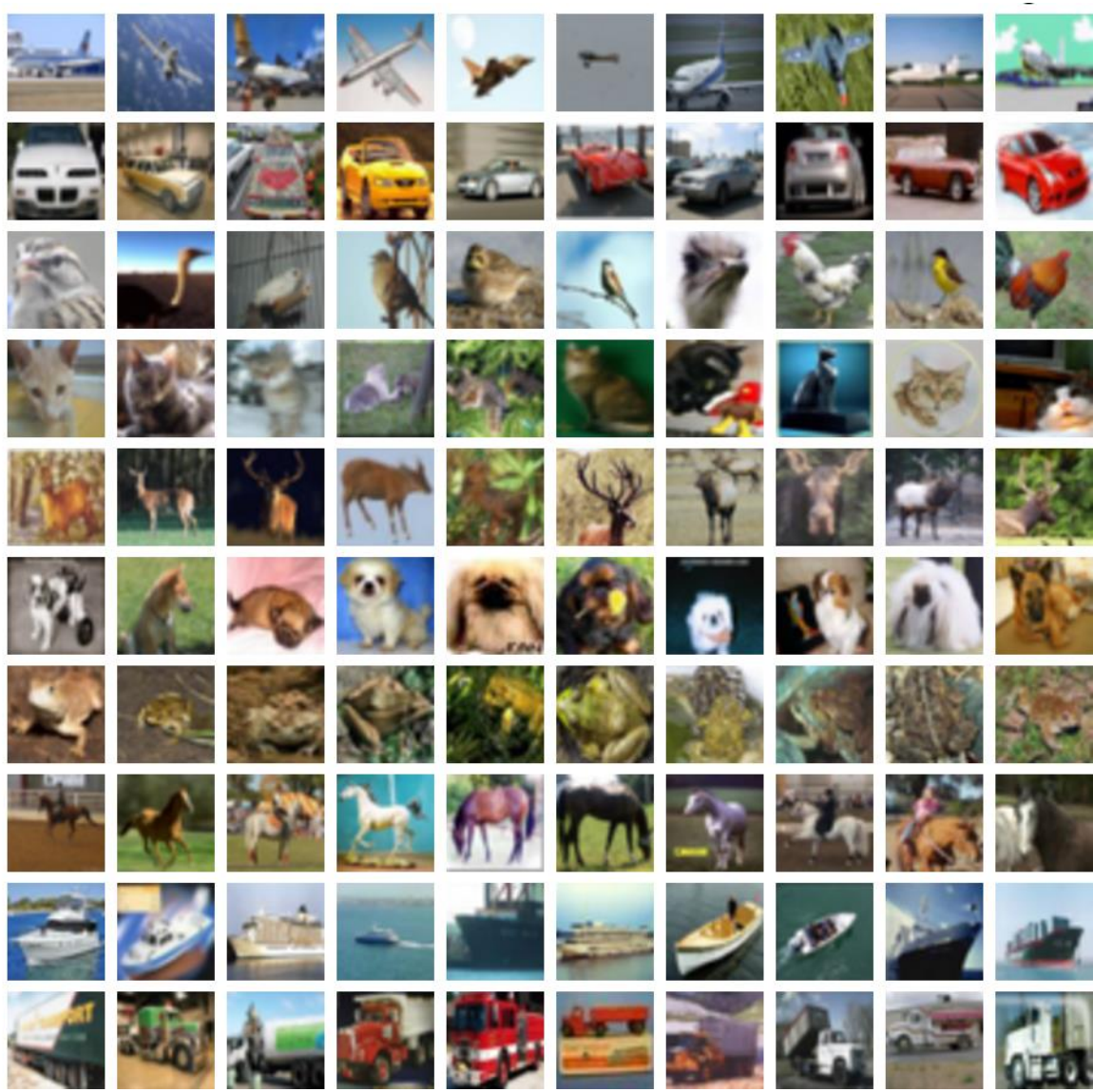
dog

frog

horse

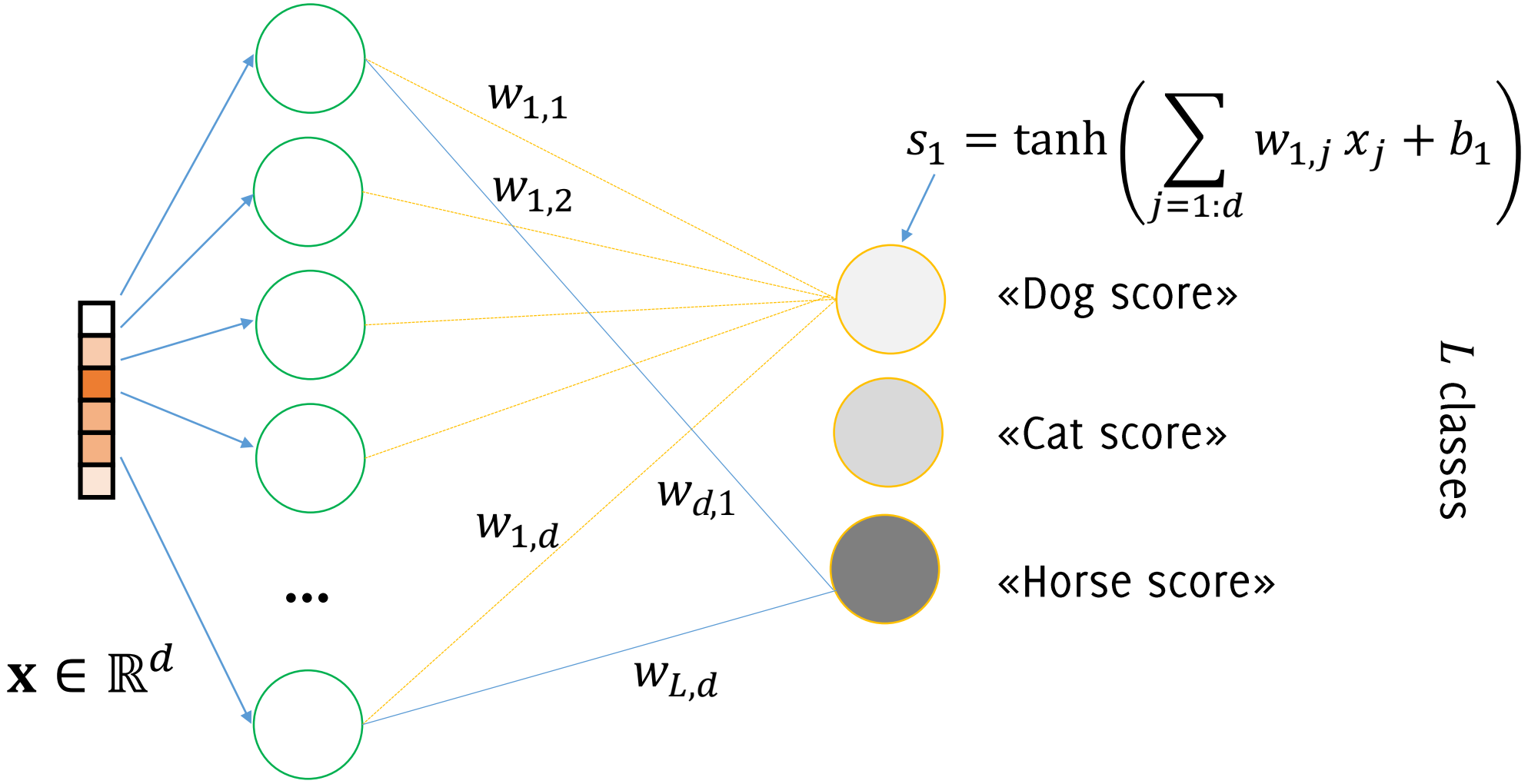
ship

truck



Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

A 1-layer NN to Classify Images



model.summary();

```
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 32, 32, 3)]       0
_________________________________________________________________
Flatten (Flatten)            (None, 3072)              0
_________________________________________________________________
Output (Dense)               (None, 10)                30730
=================================================================
Total params: 30,730
Trainable params: 30,730
Non-trainable params: 0
```

Why don’t we take a larger network

Dimensionality prevents us from using in a straightforward manner deep NN as those seen so far.

Let’s take a network with an hidden layer having half of the neurons of the input layer.

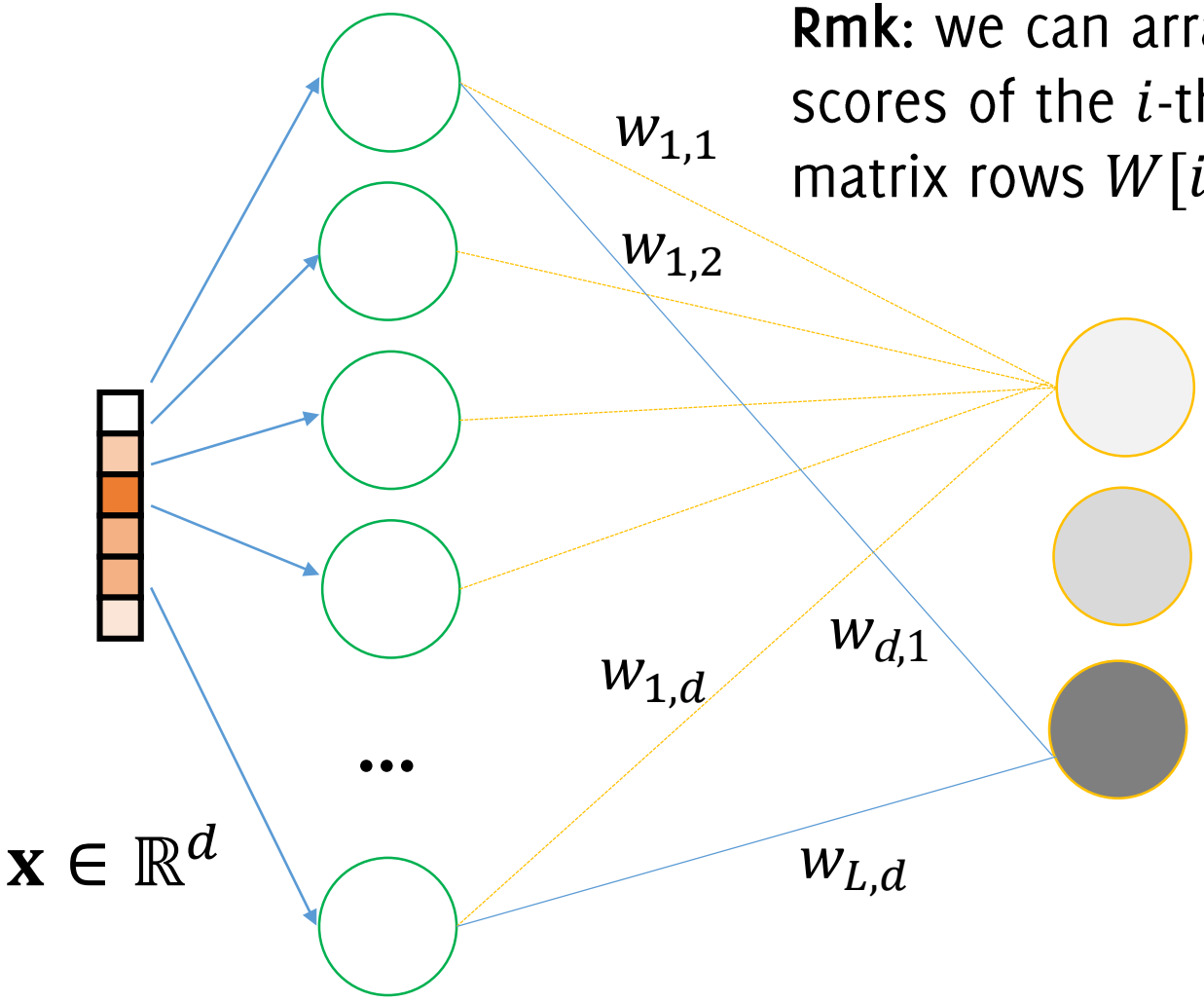
On CIFAR 10 images, the number of neurons would be:

- 3072 first layer
- 1536 second layer
- 10 output layer

1,536 * 3,072 + 1,536 = **4,720,128 parameters (!)**

10* 1,536 + 10 = 15,370 parameters

A 1-layer NN to Classify Images



Rmk: we can arrange weights in a matrix $W \in \mathbb{R}^{L \times d}$, then the scores of the i-th class is given by inner product between the matrix rows $W[i,:]$ and $\boldsymbol{x}$. Scores then becomes:

$$s_i = W[i,:] * \boldsymbol{x} + b_i$$

«Dog score»

«Cat score»

«Horse score»

L classes

Rmk: nonlinearity is not needed here since there are no layers following

Rmk: we can also ignore the softmax in the output since this would not change the order of the scores (would just normalize them)

A 1-layer NN to Classify Images

$$W \in \mathbb{R}^{L \times d}$$

W[1,:]	-8.1	...	2.7	9.5	...	-9.0	-5.4	...	4.8
W[2,:]	9.0	...	5.4	4.8	...	1.2	9.5	...	-8.0
W[3,:]	1.2	...	9.5	-8.0	...	8.1	-2.7	...	9.5

$*$

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

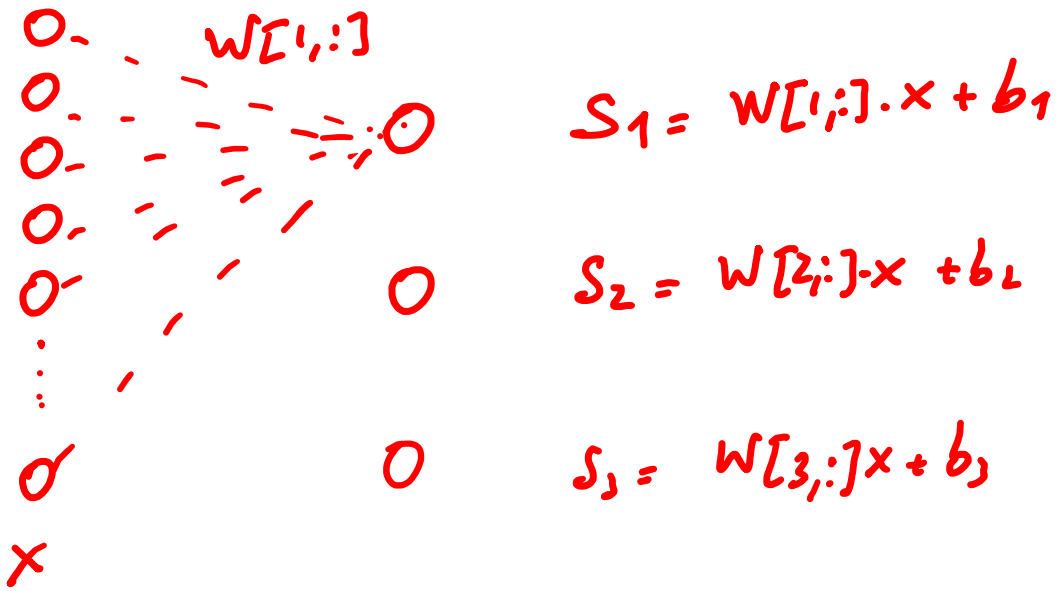
$+$

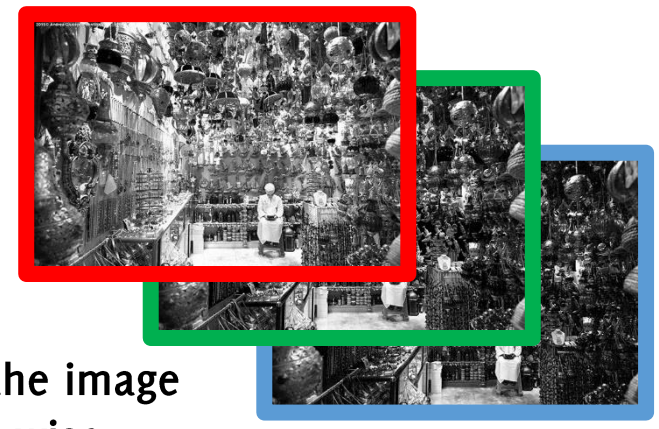
b
-2
32
-1

$=$

$\mathcal{K}(x; W, b)$	
-4	s_1 dog score
22	s_2 cat score
33	s_3 horse score

Rmk: colors indicate to which color plane in the image these weights refer to





Unroll the image column-wise

This simple layer is a linear classifier

In linear classification $\mathcal{K}$ is a linear function:

$$\mathcal{K}(\boldsymbol{x}) = W\boldsymbol{x} + b$$

where $W \in \mathbb{R}^{L \times d}$, $b \in \mathbb{R}^{L}$ are the parameters of the classifier $\mathcal{K}$.

W are referred to as the weights, b the bias.

L

-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

$\boldsymbol{x}$

$+$

-2
32
-1

$\boldsymbol{b}$

$=$

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

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

Unroll the image column-wise

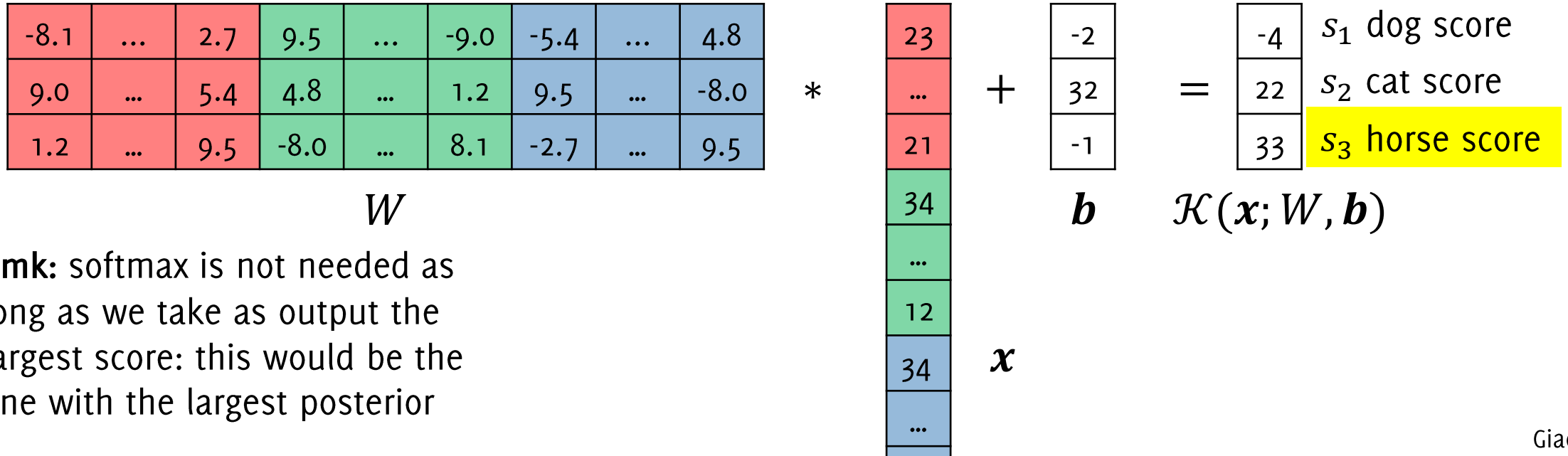
This simple layer is a linear classifier

The classifier assign to an input image the class corresponding to the largest score

$$\hat{y}_j = \underset{i=1,..,L}{\operatorname{argmax}} \left[s_j\right]_i$$

being $\left[\boldsymbol{s}_j\right]_i$ the i −th component of the vector

$$\mathcal{K}(\boldsymbol{x}) = W\boldsymbol{x} + \boldsymbol{b}$$



Rmk: softmax is not needed as long as we take as output the largest score: this would be the one with the largest posterior

Parameters of a Linear Classifier

The **score of a class is the weighted sum** of all the image pixels.
Weights are actually the classifier parameters.

The weights are:

-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

-2
32
-1

$\boldsymbol{b}$

and indicate which are the most important pixels / colors

Training the Linear Classifier

Training a Classifier

Given a training set TR and a loss function, define the parameters that minimize the loss function over the whole TR

In case of linear classifier

$$[W, b] = \underset{W \in \mathbb{R}^{L \times d},\ b \in \mathbb{R}^{L}}{\operatorname{argmin}} \sum_{(\boldsymbol{x}_i, y_i) \in TR} \mathcal{L}(\boldsymbol{x}, y_i)$$

Solving this minimization problem provides the weights of our classifier

Loss Function

Loss function: a function $\mathcal{L}$ that measures our unhappiness with the score assigned to training images

The loss $\mathcal{L}$ will be high on a training image that is not correctly classifier, low otherwise.

Loss Function Minimization

Loss function can be minimized by specific algorithms (see Matteo' classes)

The loss function has to be typically regularized to achieve a unique solution satisfying some desired property

$$[W, b] = \underset{W \in \mathbb{R}^{L \times d},\ b \in \mathbb{R}^{L}}{\operatorname{argmin}} \sum_{(\boldsymbol{x}_i, y_i) \in TR} \mathcal{L}(\boldsymbol{x}, y_i) + \lambda\, \mathcal{R}(W, b)$$

being $\lambda > 0$ a parameter balancing the two terms

… Once Trained

The classifier is expected to provide to the correct class a score that is larger than that assigned to the incorrect classes.

The training data is used to learn the parameters $W, \boldsymbol{b}$

Once the training is completed, it is possible to discard the whole training set and only keep the learned parameters.

-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

-2
32
-1

$\boldsymbol{b}$

Geometric Interpretation of a Linear Classifier

$W[i,:]$ is a d −dimensional vector containing the weights of the score function for the i −th class.

Computing the score function for the i −th class corresponds to computing the inner product

$$W[i,:] * \boldsymbol{x}$$

Thus, the neural network is like computing the inner products against L different weights vectors and selecting the one yielding the largest score (up to bias correction)

Rmk: these "inner product classifiers" operate independently, and the output of the j-th row is not influenced weights at a different row

Rmk: this would not be the case if the network had hidden layer that would mix the outputs of intermediate layers

Geometric Interpretation of a Linear Classifier

In Python notation:

In Python $*$ denotes the element-wise product, here I mean the inner product of vectors:

$$\text{np.inner}(W[i,:], \boldsymbol{x})$$

Geometric Interpretation

Interpret each image as a point in $\mathbb{R}^d$.

Each classifier is a weighted sum of pixels, which corresponds to a linear function in $\mathbb{R}^d$

In $\mathbb{R}^2$ these would be

$$f([x_1, x_2]) = w_1 x_1 + w_2 x_2 + b$$

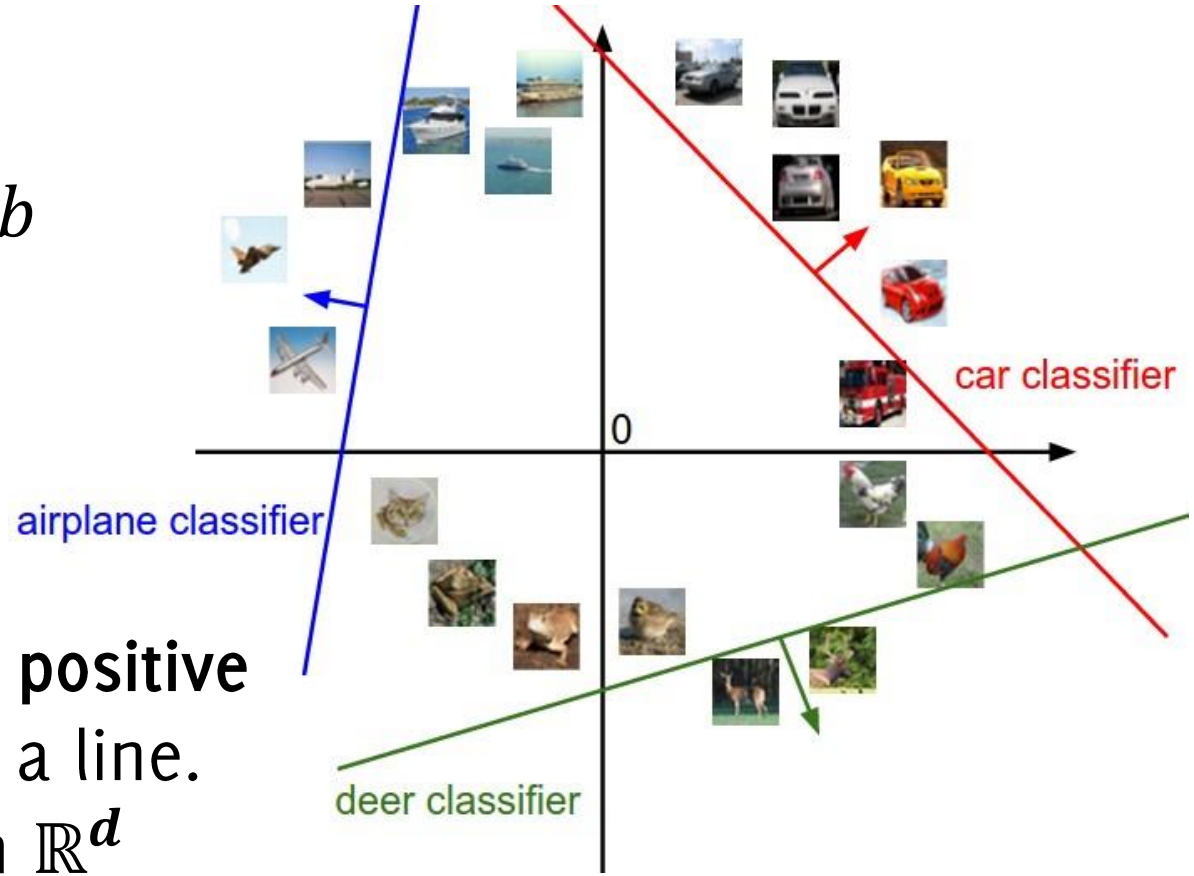
Then, points $[x_1, x_2]$ yielding

$$f([x_1, x_2]) = 0$$

would be lines.

Thus, in $\mathbb{R}^2$ the **region that separates positive from negative scores for each class is** a line.

This region becomes **an hyperplane in** $\mathbb{R}^d$



Template Matching Interpretation

In Python notation:

- $W[i,:]$ is a d −dimensional vector containing the weights of the score function for the i −th class
- Computing the score function for the i −th class corresponds to computing the inner product

$$W[i,:] * \boldsymbol{x}$$

Then, $W[i,:]$ **can be seen as a template used in matching (the output of correlation in the central pixel)**

The template $W(i,:)$ is learned to match at best images belonging to the i −th class

Let's have a look at these templates

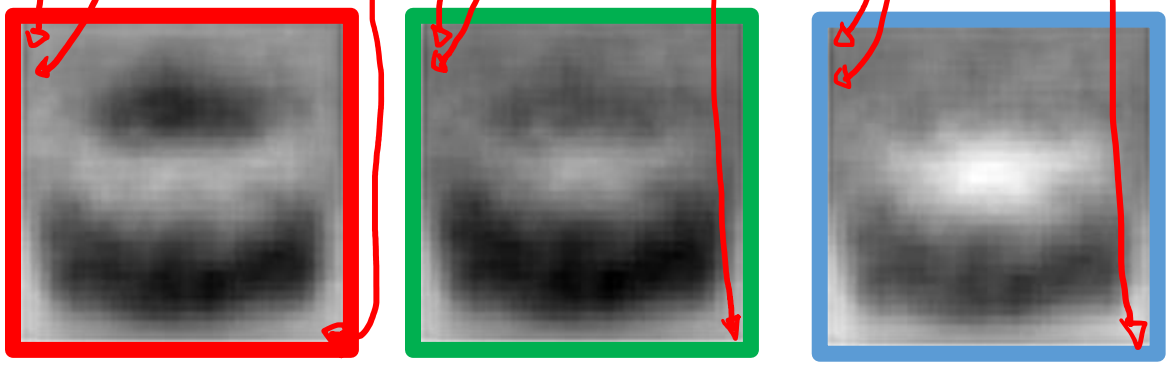
Bring the classifier weights back to images

W

-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

$d = R \times C \times 3$

$W[i,:] \in \mathbb{R}^d$, car classifier



$R \in \mathbb{R}^{R \times C}$ $G \in \mathbb{R}^{R \times C}$ $B \in \mathbb{R}^{R \times C}$

car template in $\mathbb{R}^{R \times C \times 3}$

Templates Learned on the CIFAR-10 dataset

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

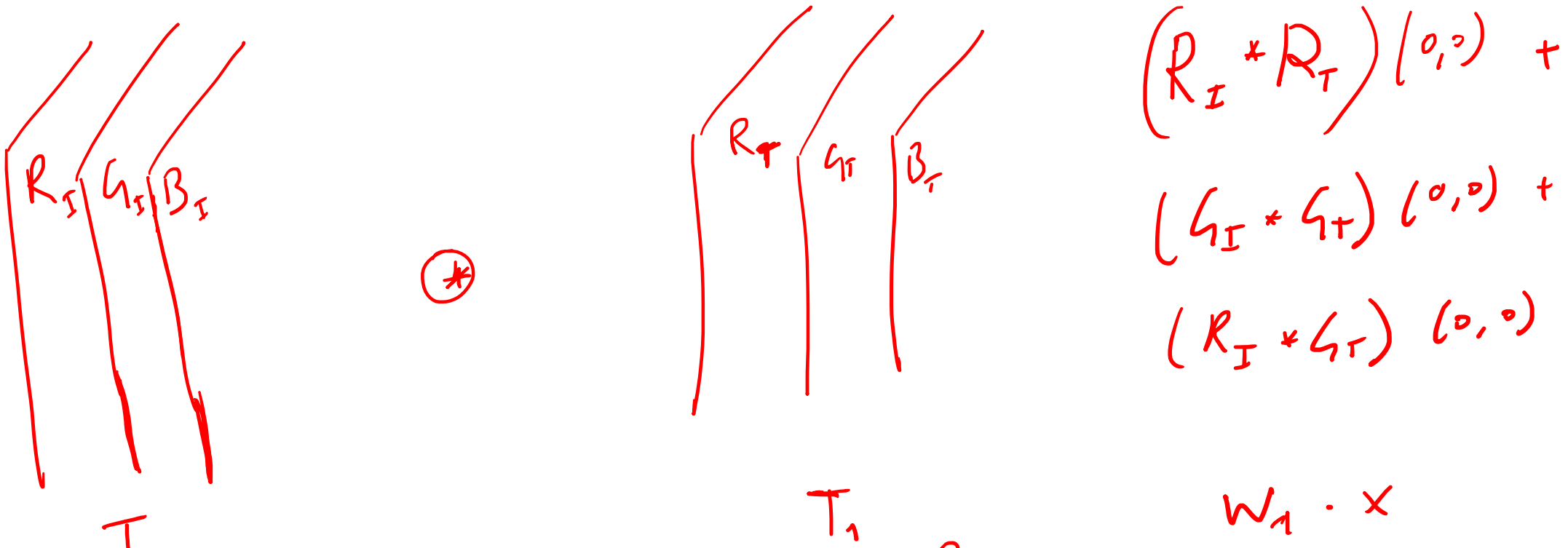
The Class Score

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

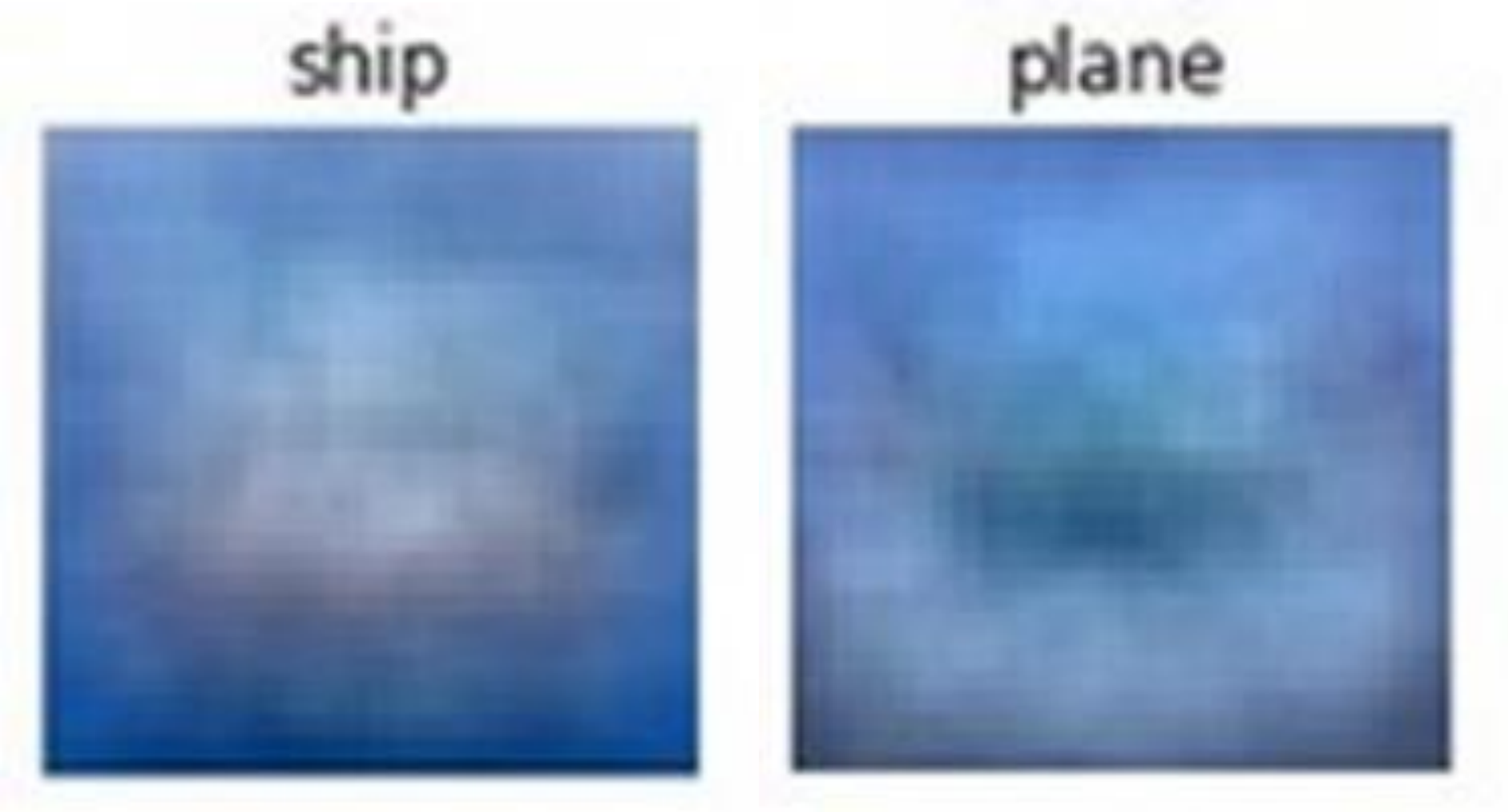
$$(I \otimes T_1)(0,0) = \sum_{(x,y)\in U} T_1(x,y) * I(x,y)$$

Correlation between two RGB images

The image and the filter have the same size

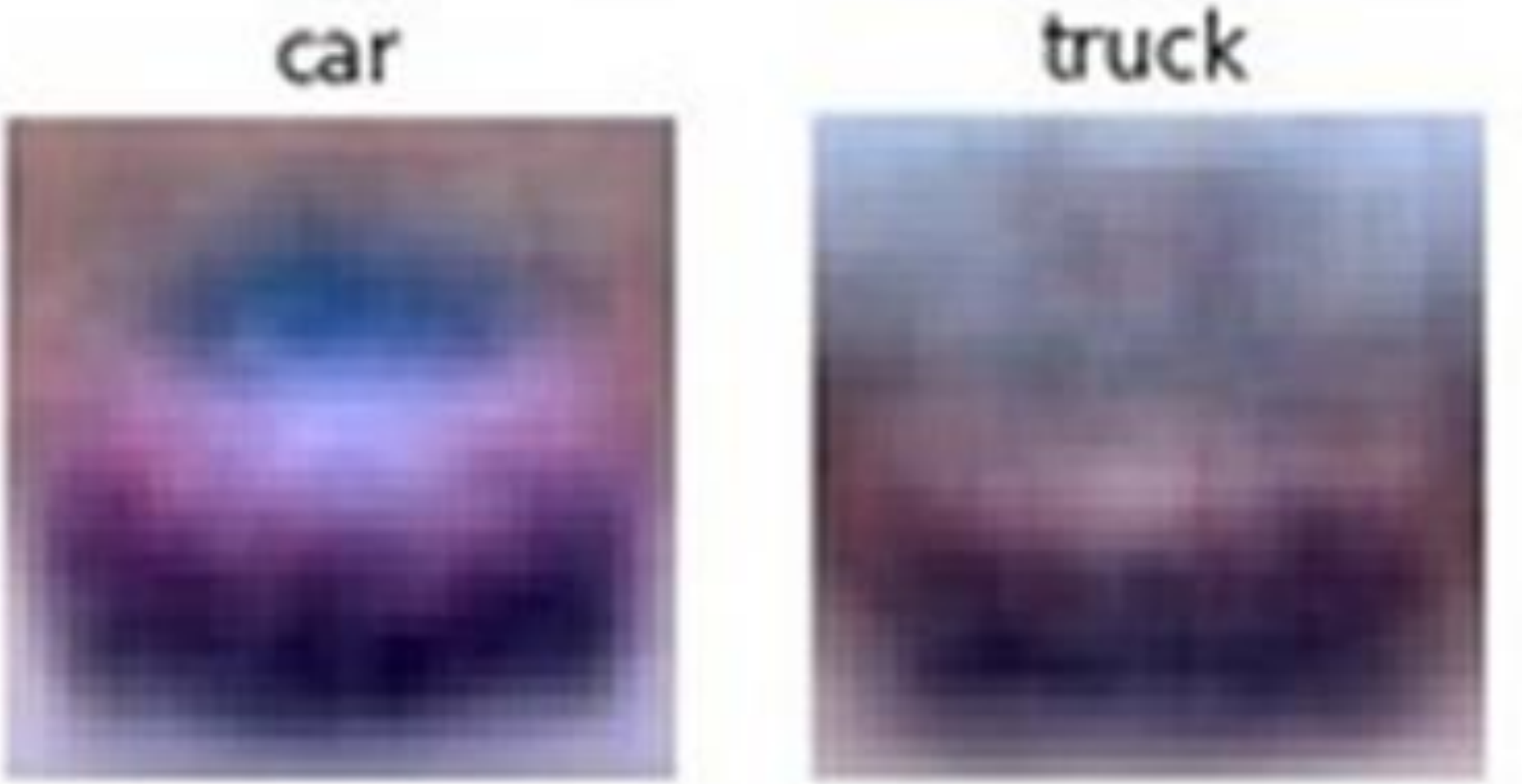


Templates Learned on the CIFAR-10 dataset



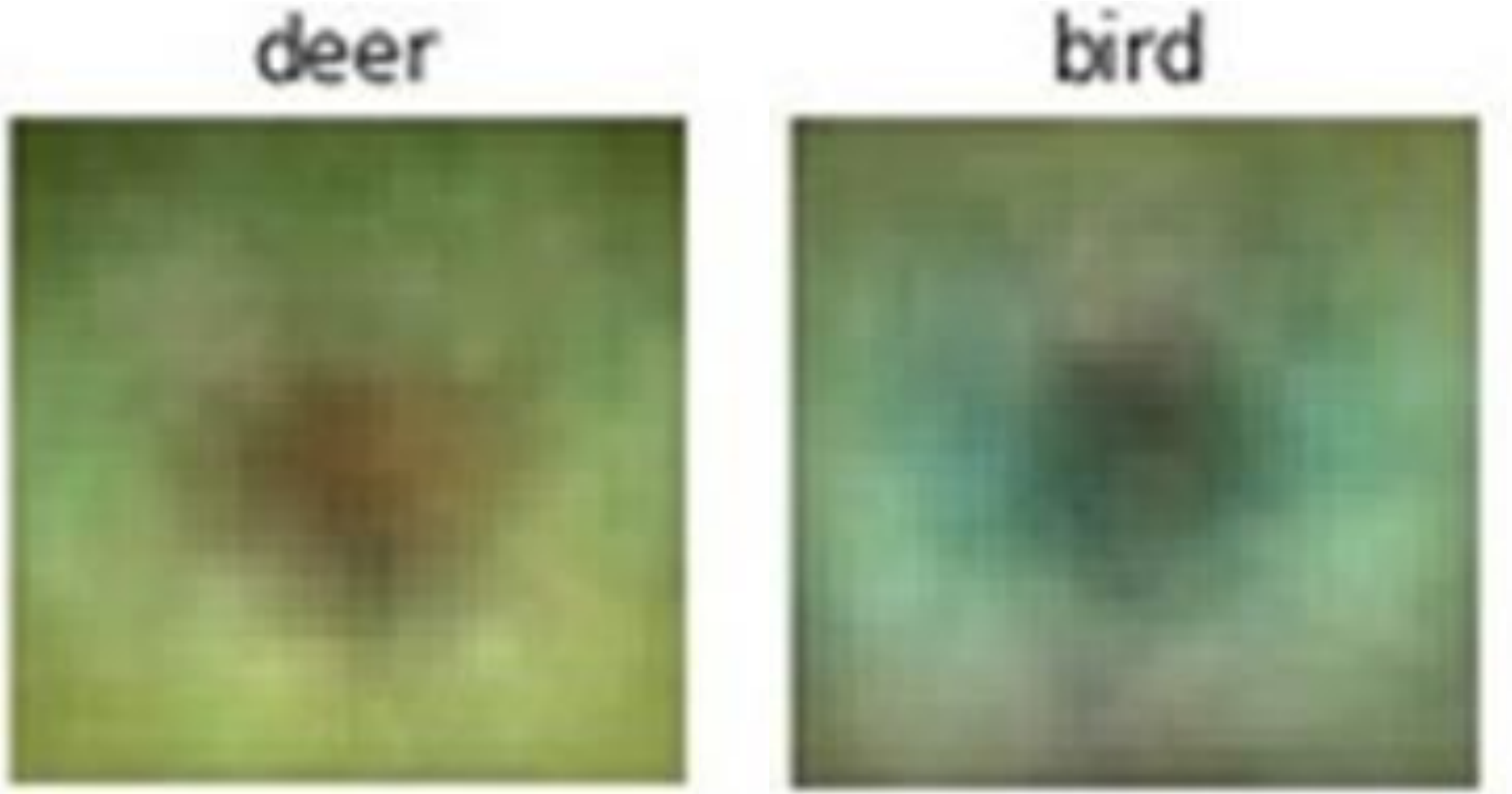
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Templates Learned on the CIFAR-10 dataset



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

Templates Learned on the CIFAR-10 dataset



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

Templates Learned on the CIFAR-10 dataset

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

Linear Classifier as a Template Matching

What has the classifier learned?

- That the background of bird and frog is green, (plane and boat is blue)
- Cars are typically red
- Horses have two heads! ☺

The model was definitively too simple / data were not enough for achieving higher performance and better templates

However:

- Linear Classifiers are among the most important layer of NN
- Such a simple model can be interpreted (with more sophisticated models you typically can’t)

Linear Classifier as a Template Matching

What has the classifier learned?

- That the background of bird and frog is ... and boat is blue)
- Cars are typically red
- Horses have two heads! ☺

The model was definitive... / data were not enough for achieving higher perfor... better templates

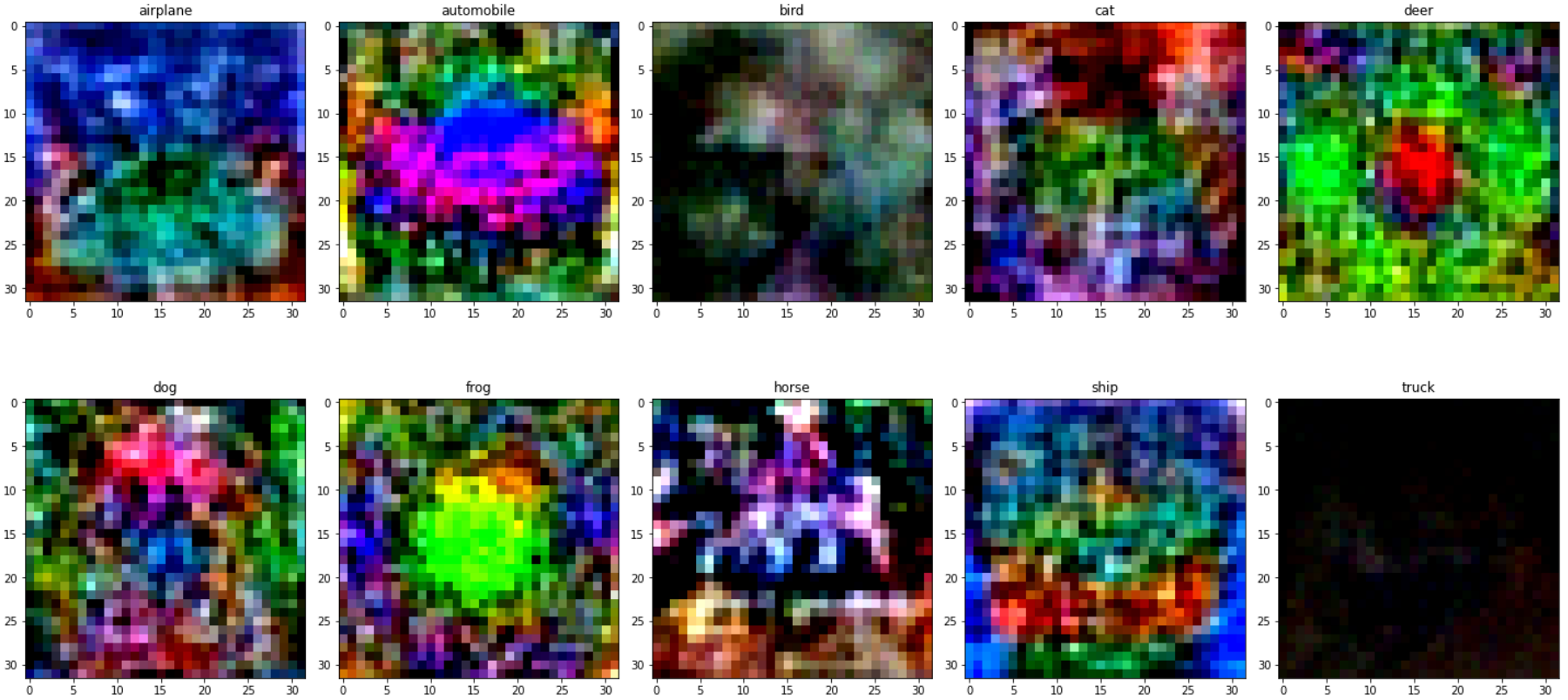
However:

- Linear Classifiers a... the most important layer of NN
- Such a simple model can be interpreted (with more sophisticated models you typically can't)

There should be a better way for handling images

Do it yourself!

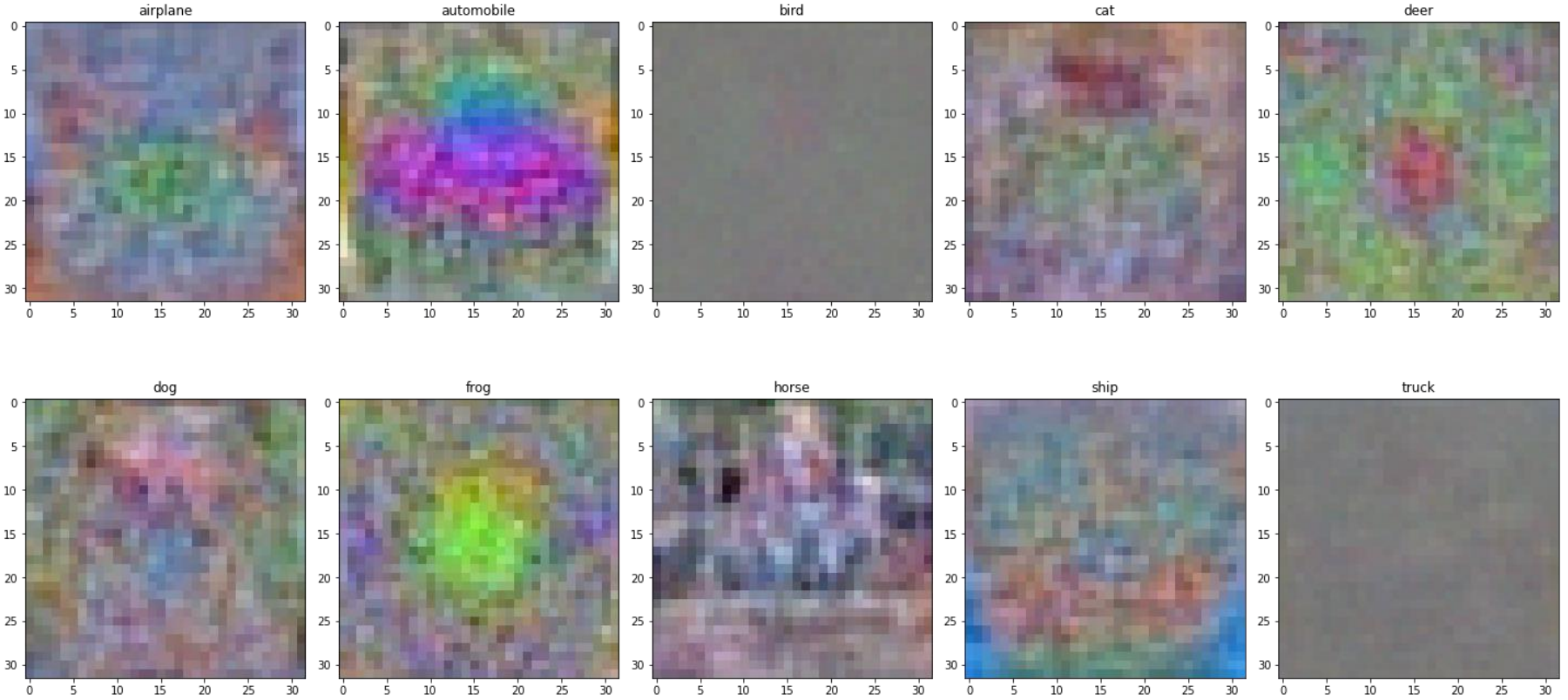
https://colab.research.google.com/drive/1MoXvvDm_eMEeHBussa2SeiLmUBy5g_7T?usp=sharing



Credits Eugenio Lomurno! (visualization with saturated colors)

Do it yourself!

https://colab.research.google.com/drive/1MoXvvDm_eMEeHBussa2SeiLmUBy5g_7T?usp=sharing



Credits Eugenio Lomurno! (visualization with clipped colors)

Is Image Classification a Challenging Problem?

Yes, it is...

Is Image Classification a Challenging Problem?

What does prevent us from

- Stacking multiple hidden layers
- Training the network with a huge amount of images

to successfully classify images?

Would this work?

Is it a challenging problem?

First challenge: dimensionality

Images are very high-dimensional image data

CIFAR-10 dataset

The CIFAR-10 dataset contains 60000 images:

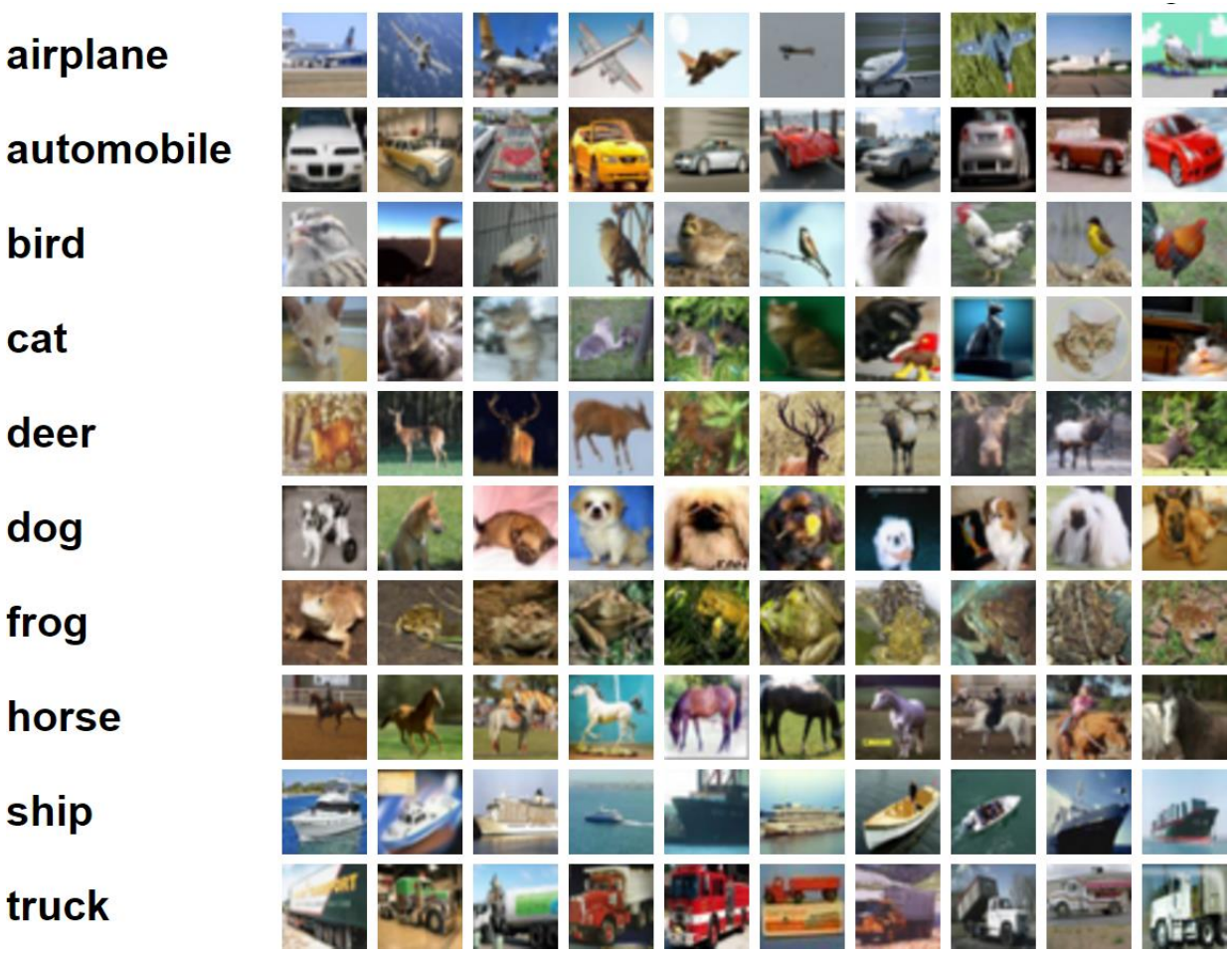
Each image is 32x32 RGB

Images are in 10 classes

6000 images per class

Extremely small images, but high-dimensional:

$$d = 32 \times 32 \times 3 = 3072$$



Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.


This resolution is by far smaller than what we are used to
$d = 3072$

Former standard repository for ML research

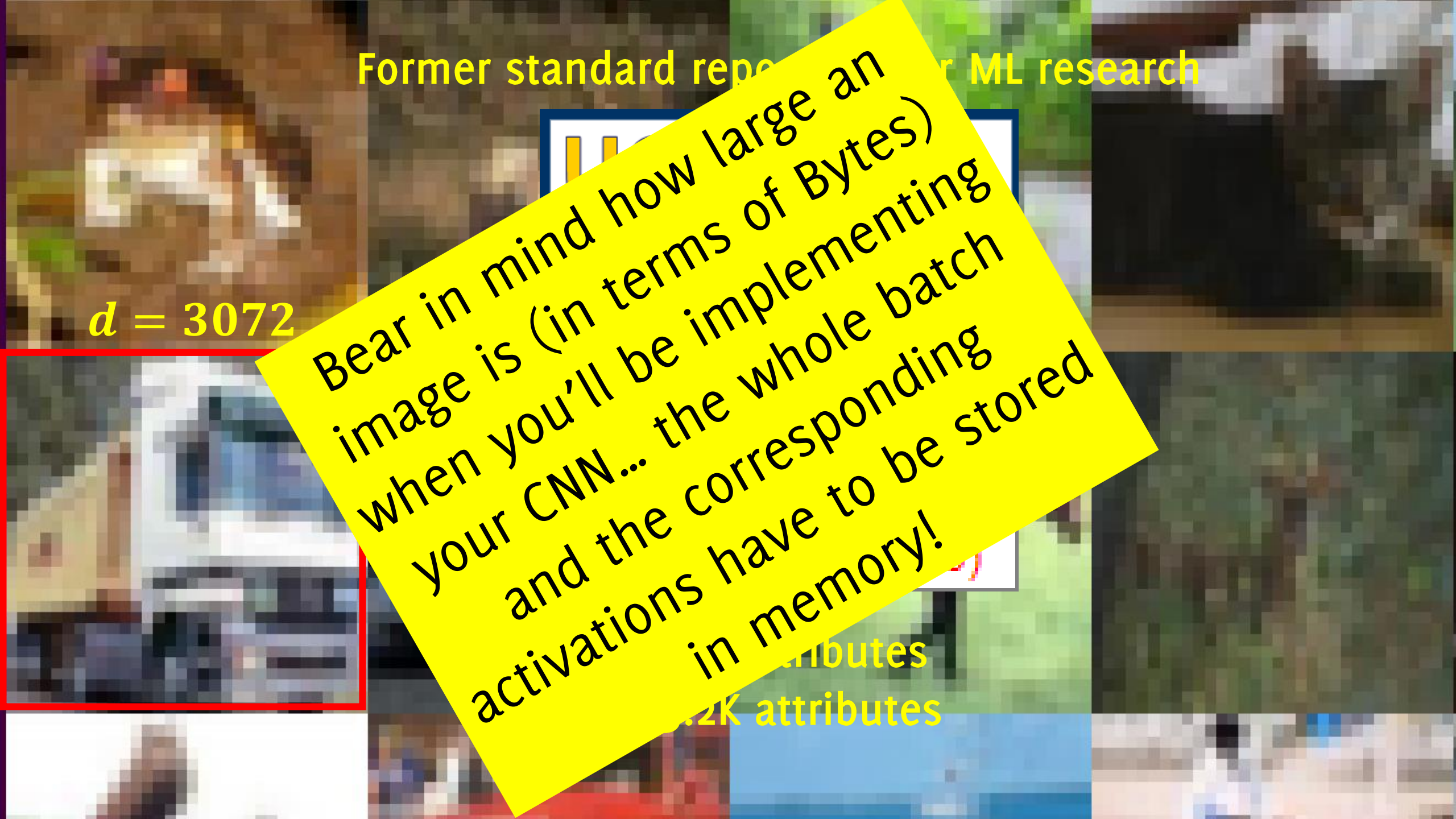
UCI Machine Learning Repository
Center for Machine Learning and Intelligent Systems

$d = 3072$

 # Attributes

Less than 10 (116)
10 to 100 (218)
Greater than 100 (86)

- 88% < 500 attributes
- 92% < 3.2K attributes


Former standard repe r ML research
$d = 3072$
Bear in mind how large an image is (in terms of Bytes) when you'll be implementing your CNN… the whole batch and the corresponding activations have to be stored in memory!
ributes
.2K attributes

Second challenge: label ambiguity

A label might not uniquely identify the image

Second challenge: label ambiguity

Man?

Beer?

Dinner?

Restaurant?

Sausages?

….

Third challenge: transformations

There are many transformations that change the image dramatically, while not its label

Changes in the Illumination Conditions






Giacomo Boracchi

Deformations

Copyright Christine Matthews

© Copyright Patrick Roper

View Point Change

… and many others

Occlusion

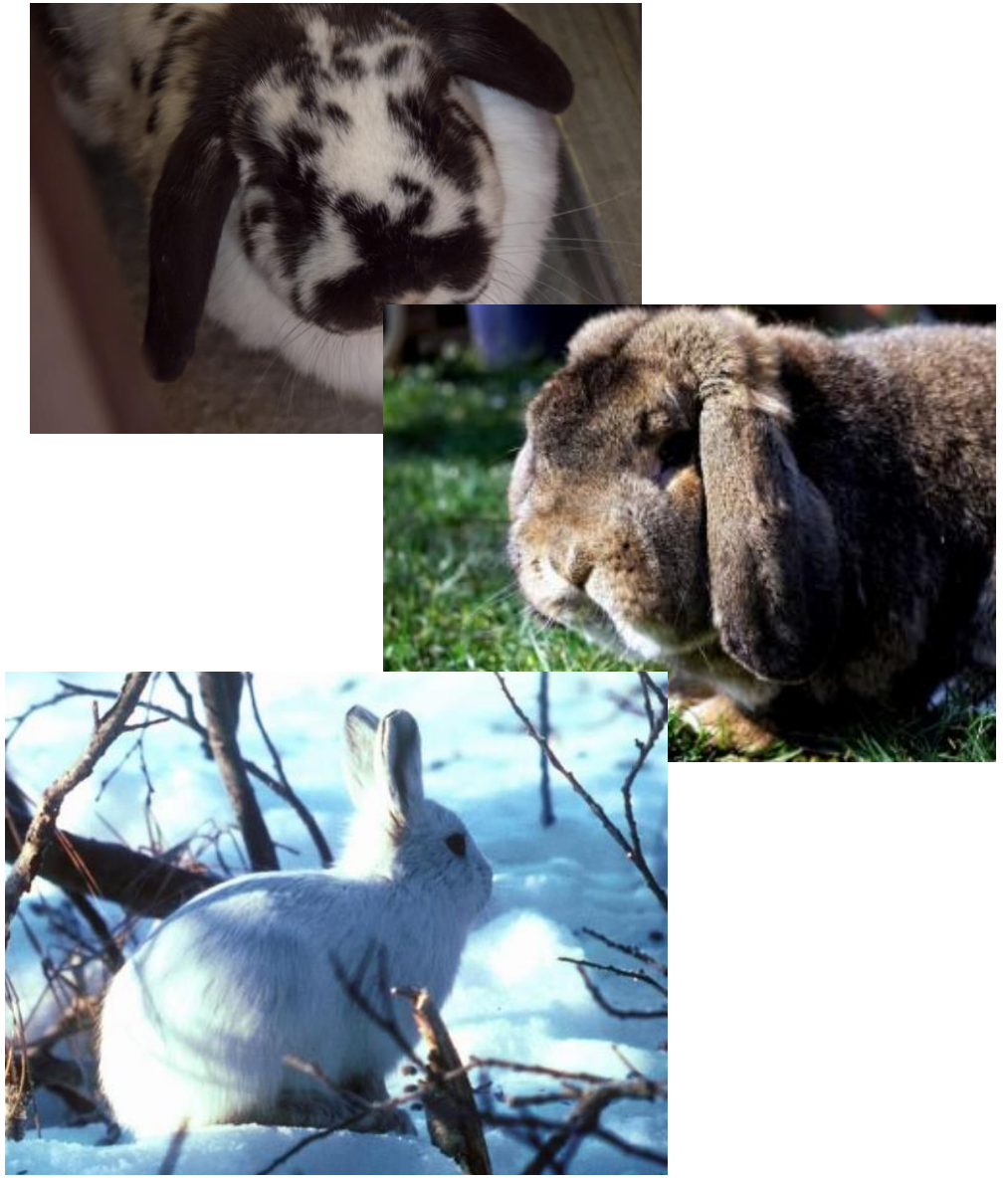
Background clutter

Scale variation

Fourth challenge: inter-class variability

Images in the same class might be dramatically different

Inter-class variability



Fifth problem: perceptual similarity

Perceptual similarity in images is not related to pixel-similarity

Nearest Neighborhood Classifiers for Images

Assign to a each test image, **the label of the closest image** in the training set

$$\hat{y}_j = y_{j^*}, \qquad \text{being} \quad j^* = \underset{i=1\ldots N}{\operatorname{argmin}}\, d(\boldsymbol{x}_j, \boldsymbol{x}_i)$$

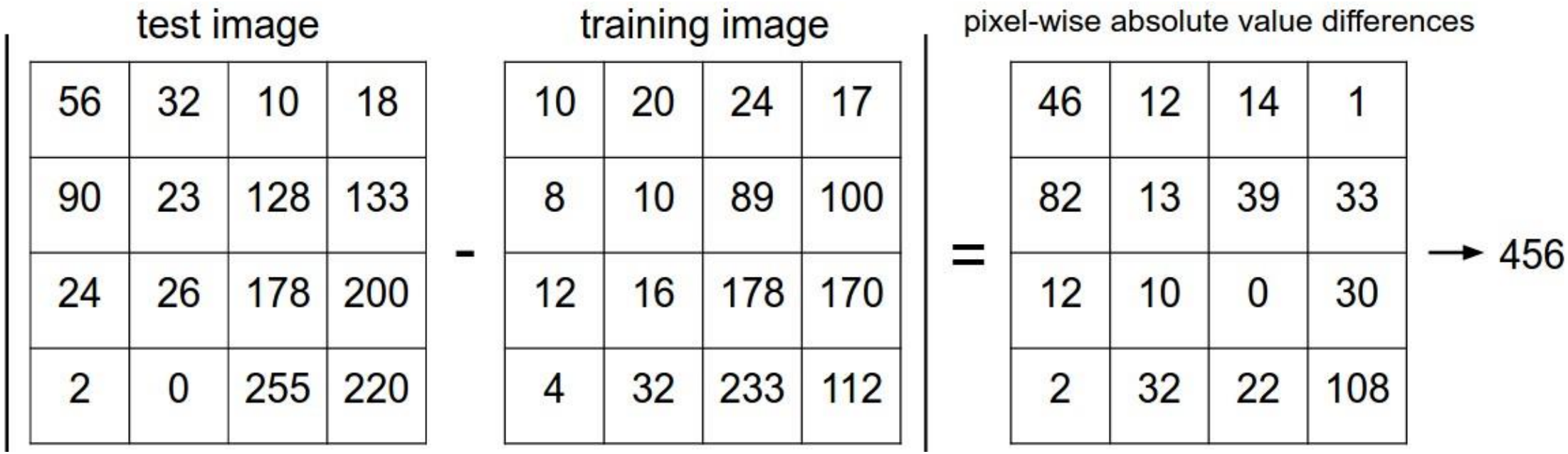
Distances are typically measured as

$$d(\boldsymbol{x}_j, \boldsymbol{x}_i) = \|\boldsymbol{x}_j - \boldsymbol{x}_i\|_2 = \sqrt{\sum_k \left([\boldsymbol{x}_j]_k - [\boldsymbol{x}_i]_k\right)^2}$$

Or

$$d(\boldsymbol{x}_j, \boldsymbol{x}_i) = |\boldsymbol{x}_j - \boldsymbol{x}_i| = \sum_k \left|[\boldsymbol{x}_j]_k - [\boldsymbol{x}_i]_k\right|$$

Pixel-wise distance among images



K-Nearest Neighborhood Classifiers for Images

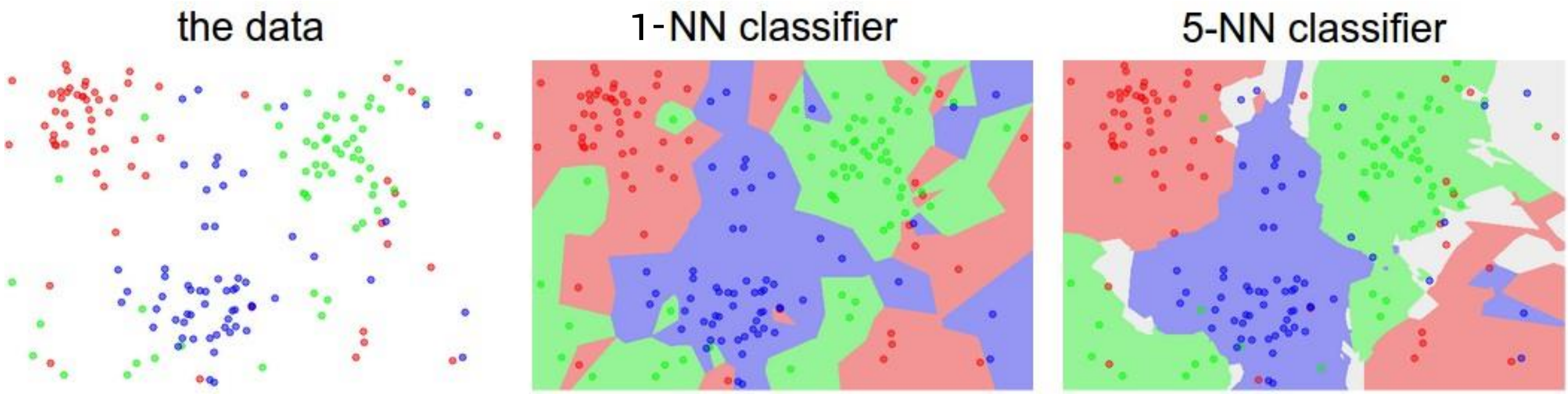
Assign to a each test image, **the most frequent label among the K −closest images in the training set**

$$\hat{y}_j = y_{j^*}, \qquad \text{being } \; j^* \text{ the mode of } \mathcal{U}_K(\boldsymbol{x}_j)$$

where $\mathcal{U}_K(\boldsymbol{x}_j)$ contains the K closest training images to $\boldsymbol{x}_j$

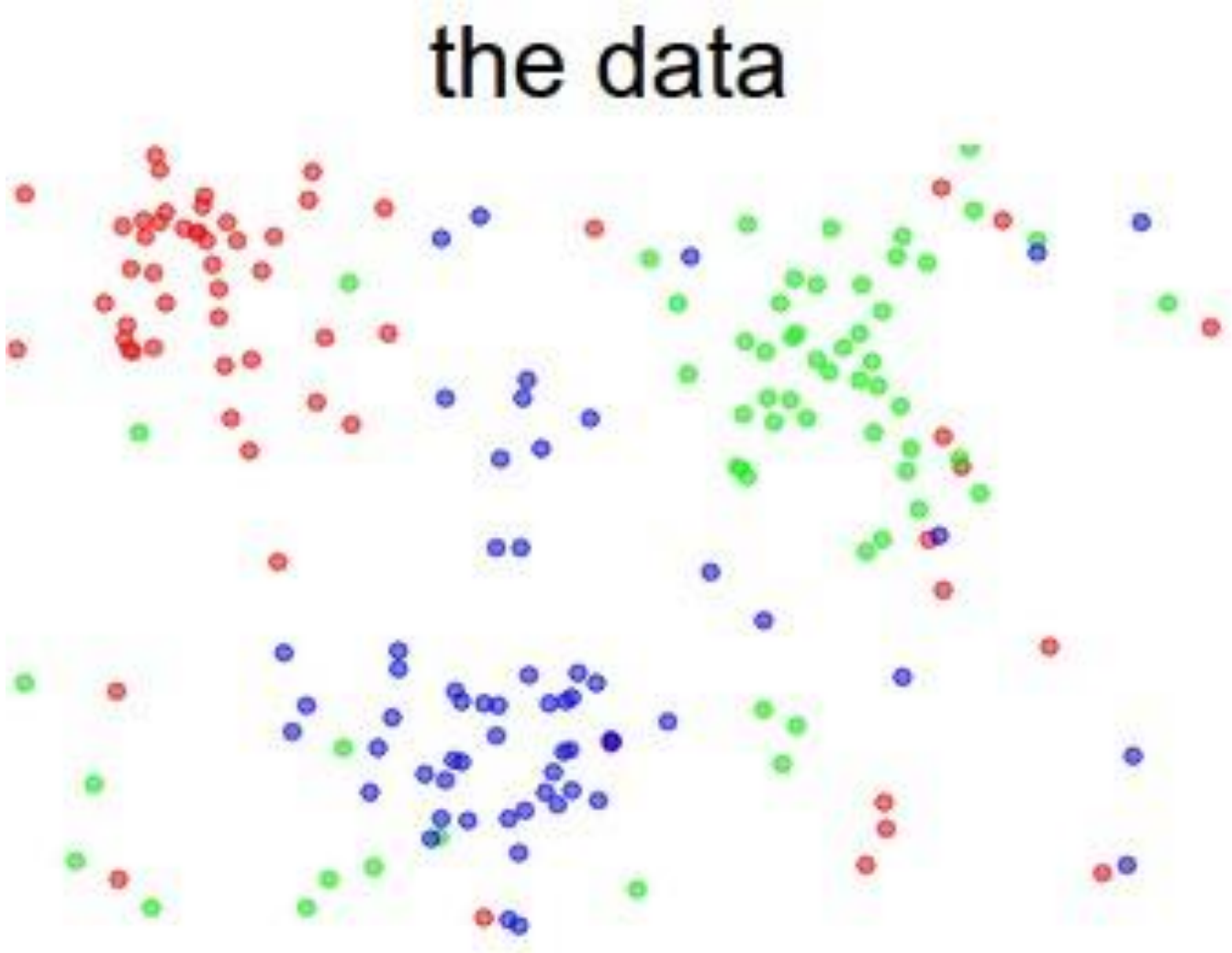
Setting the parameter K and the distance measure is an issue

Nearest Neighborhood Classifier (k-NN) for Images



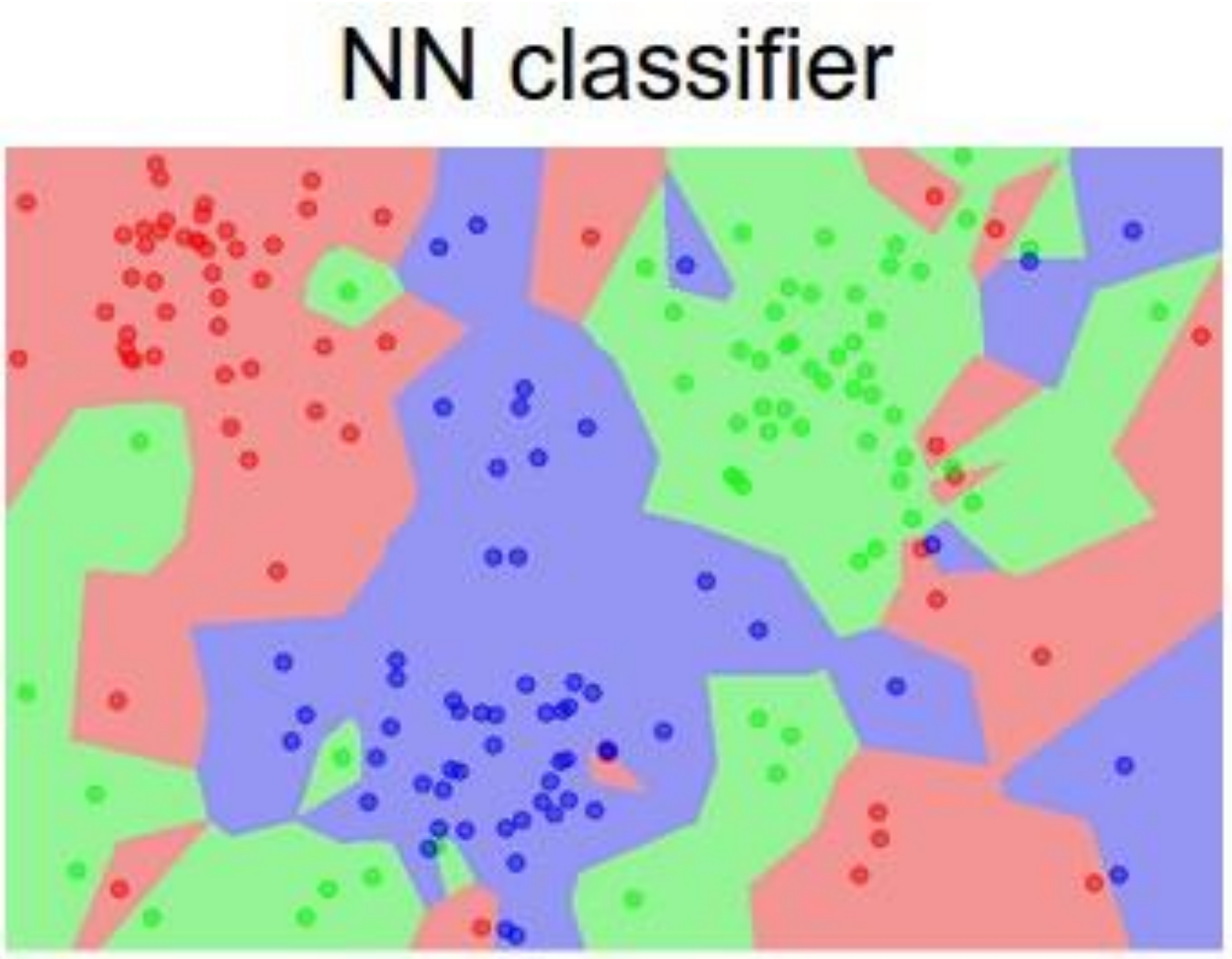
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

k-NN for Images



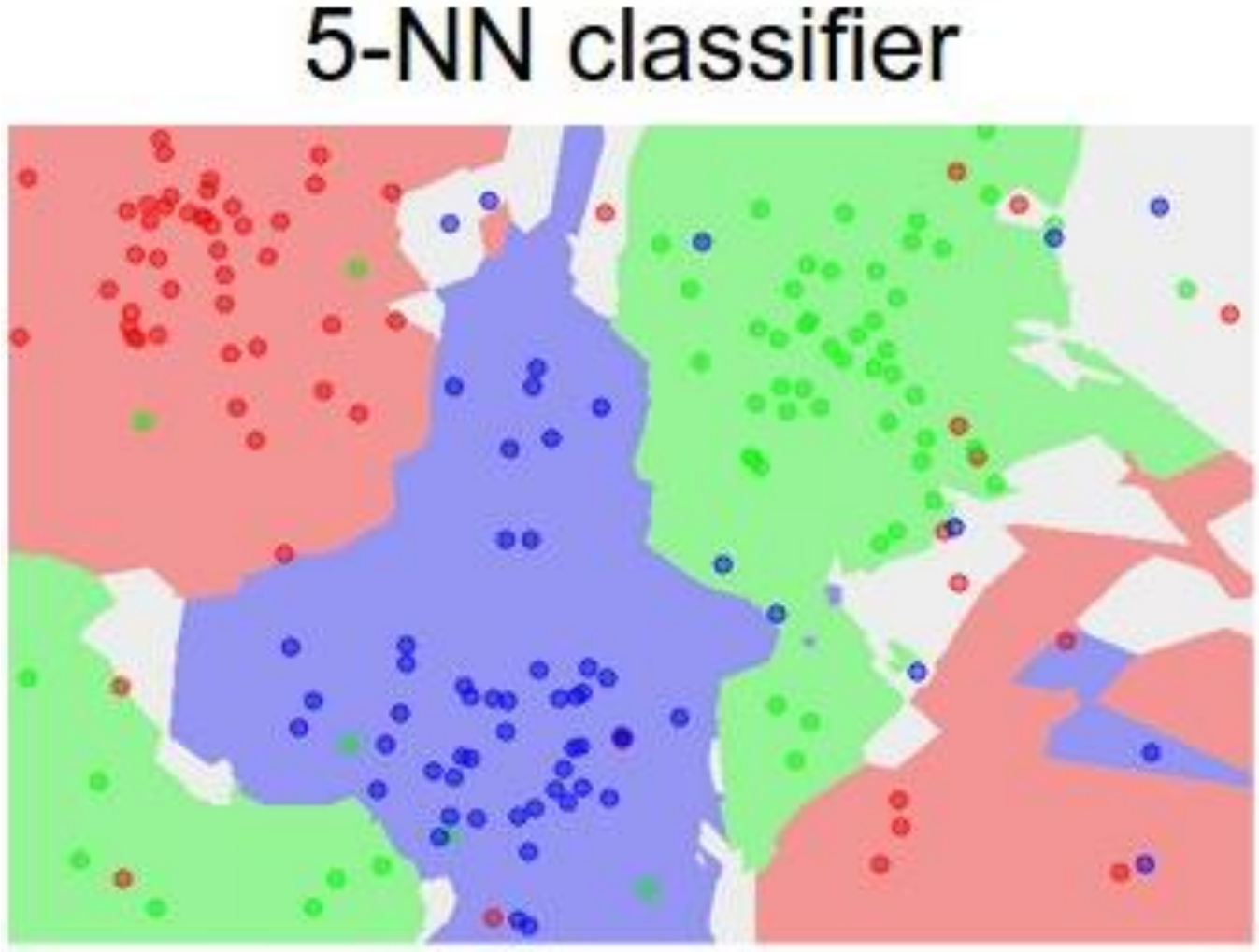
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

k-NN for Images



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

k-NN for Images



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

k-NN for Images

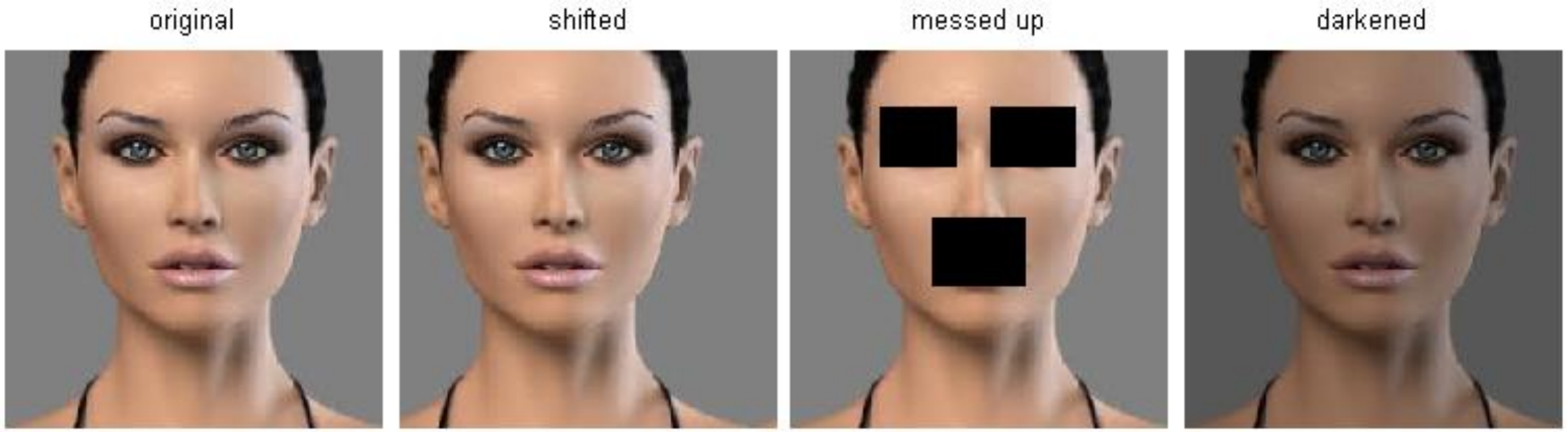
Pros:

- Easy to understand and implement
- It takes no training time

Cons:

- Computationally demanding at test time (in particular when TR is large and d is also large)
- Large training sets have to be stored in memory
- Rarely practical on images: distances on high-dimensional objects are difficult to interpret

Perceptual Similarity vs Pixel Similarity

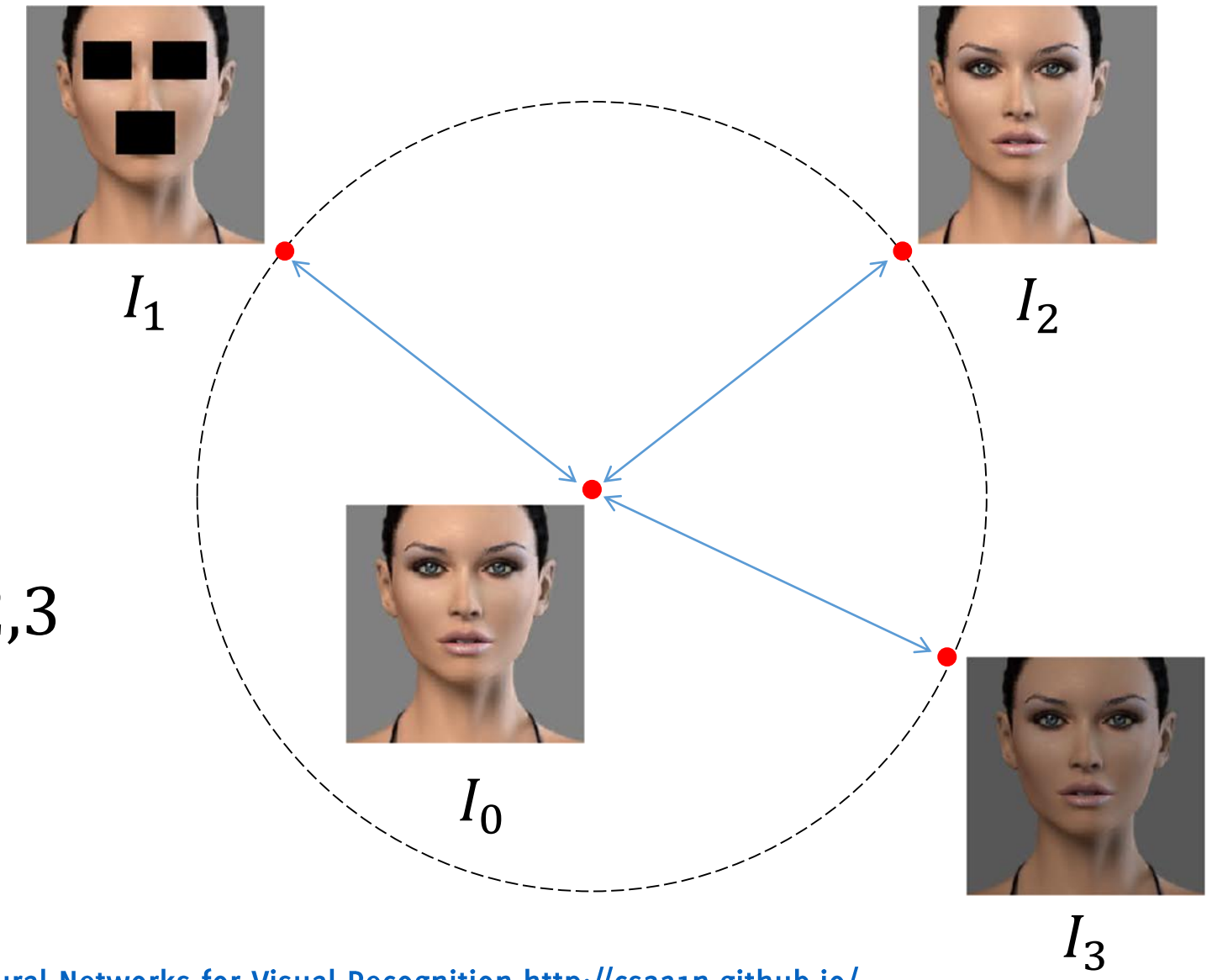


The three images have the same pixel-wise distance from the original one...

...but perceptually they are very different

Perceptual Similarity vs Pixel Similarity

$$\|I_j - I_0\|_2 \approx const\ j = 1,2,3$$



Let's see what happens on the whole CIFAR10

On CIFAR10 we see exactly this problem

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

On CIFAR10 we see exactly this problem

On CIFAR10 we see exactly this problem