16/10/2024

Image Classification

Giacomo Boracchi,
giacomo.boracchi@polimi.it

Artificial Neural Networks and Deep Learning

Politecnico di Milano
https://boracchi.faculty.polimi.it/

Lecture Outline

- Computer Vision and Digital Images
- Basics of Image Filtering
- Training a Linear Classifier on Images
- The challenges of Image Classification

What is Computer Vision About

Visual Recognition Problems

Computer Vision

DEF:

An interdisciplinary scientific field that deals with **how computers** can be made to **gain high-level understanding from digital images or videos**

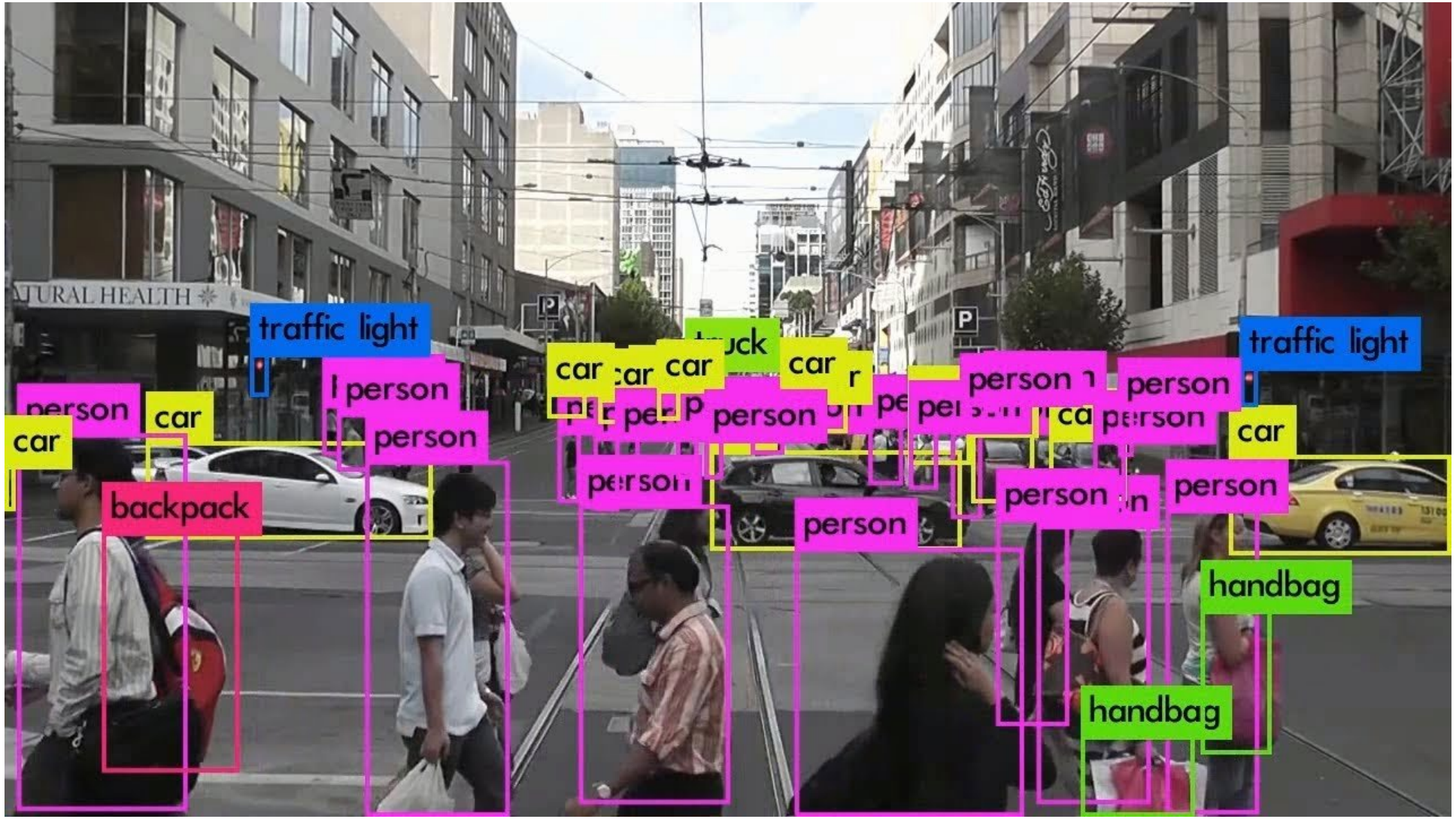
Computer Vision

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 : A first task in computer vision.



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

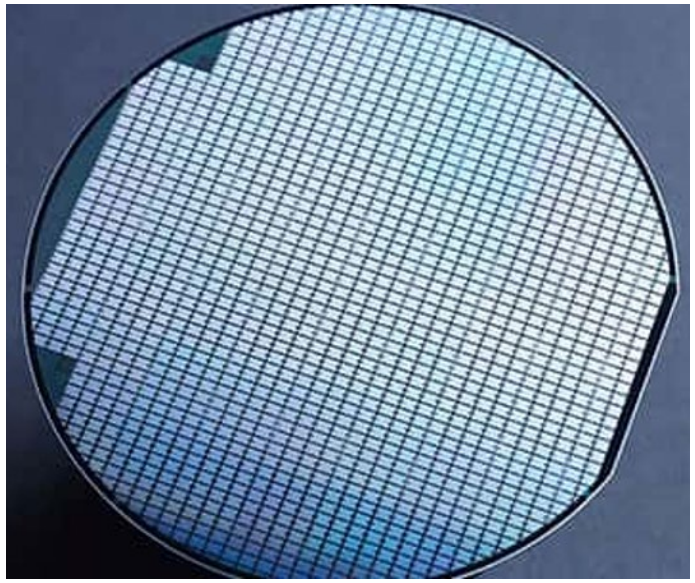
Pose Estimation : A second task in computer vision.

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).

Quality Inspection

A third task in computer vision

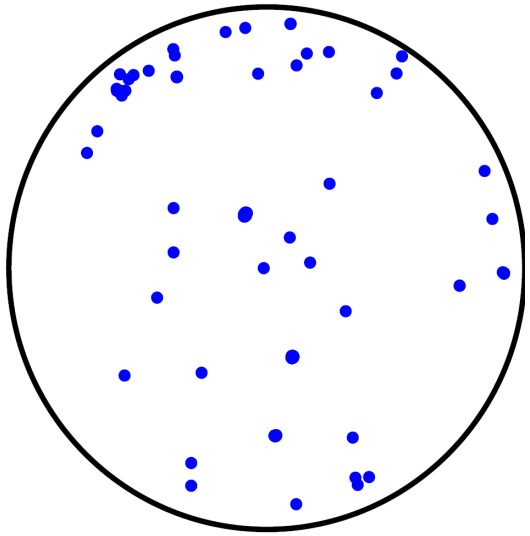
Wafer Defect Map (WDM)

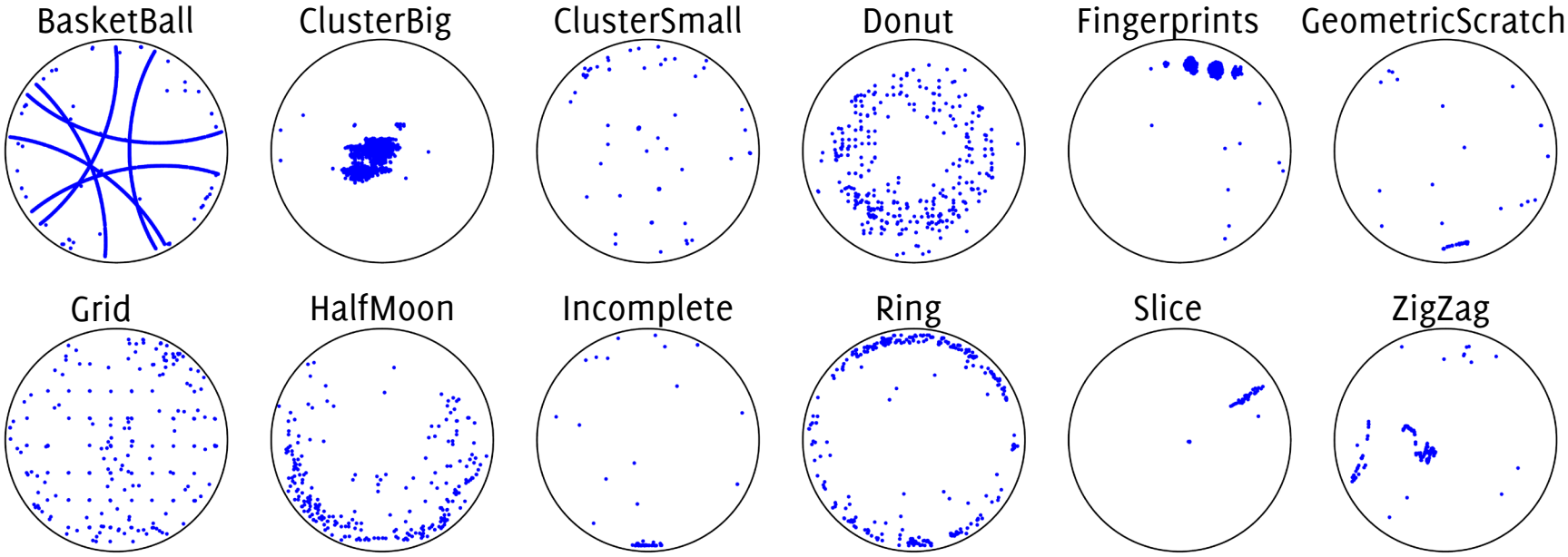


Silicon Wafer

Inspection Tool

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

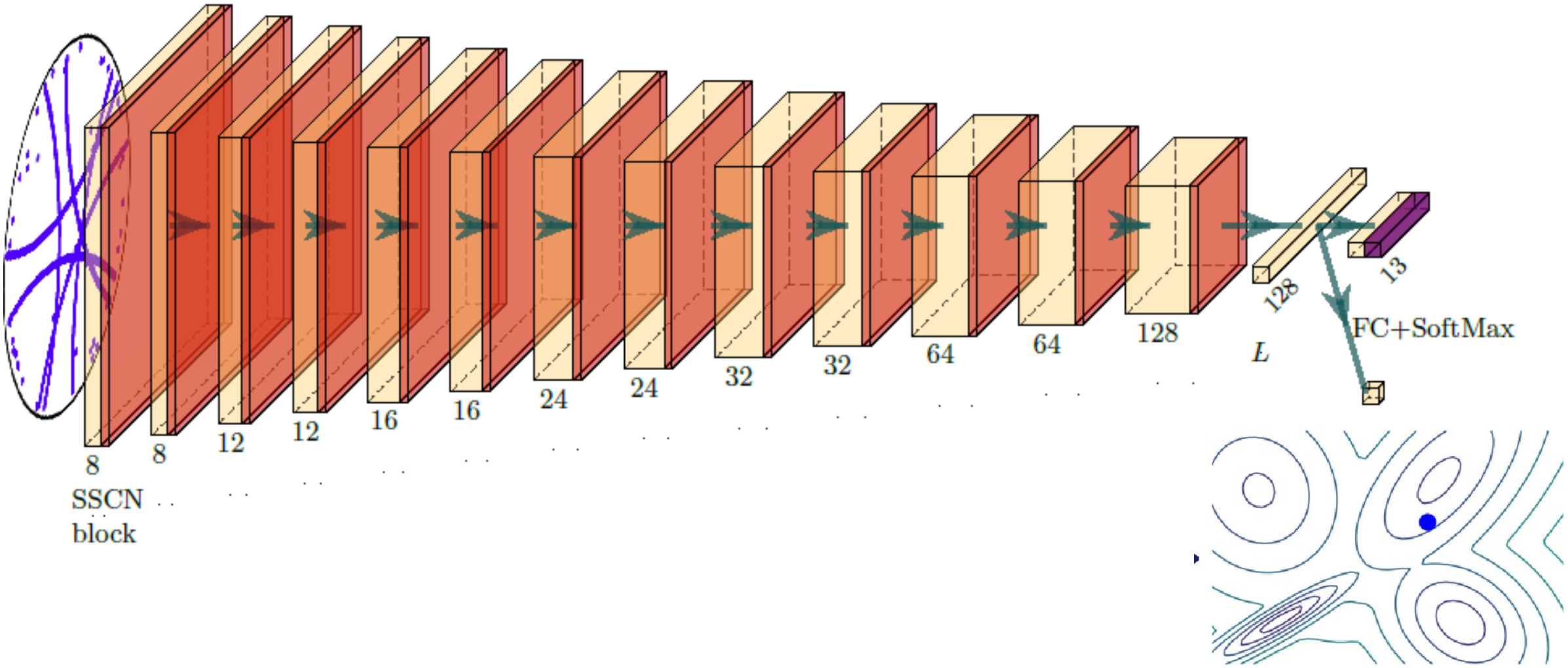




L. Frittoli, D. Carrera, B. Rossi, P. Fragneto *"Deep Open-Set Recognition for Silicon Wafer Production Monitoring"* Pattern Recognition 2022

Quality Inspection : A third task in computer vision (again).

Perfoms both Classification and Novel-class detection



L. Frittoli, D. Carrera, B. Rossi, P. Fragneto *"Deep Open-Set Recognition for Silicon Wafer Production Monitoring"* Pattern Recognition 2022

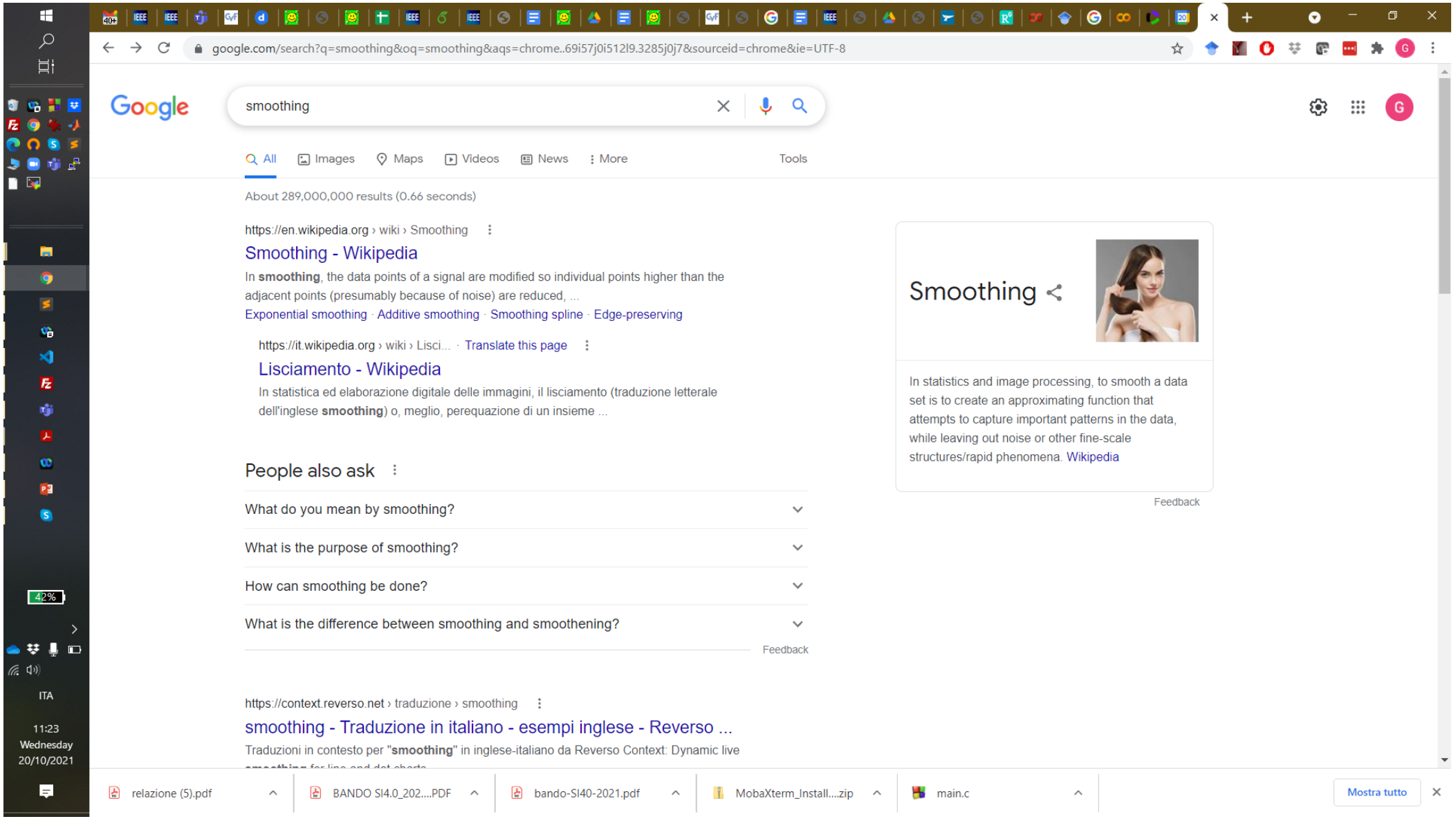
Image Captioning : A fourth task in computer vision.

"little girl is eating piece of cake."

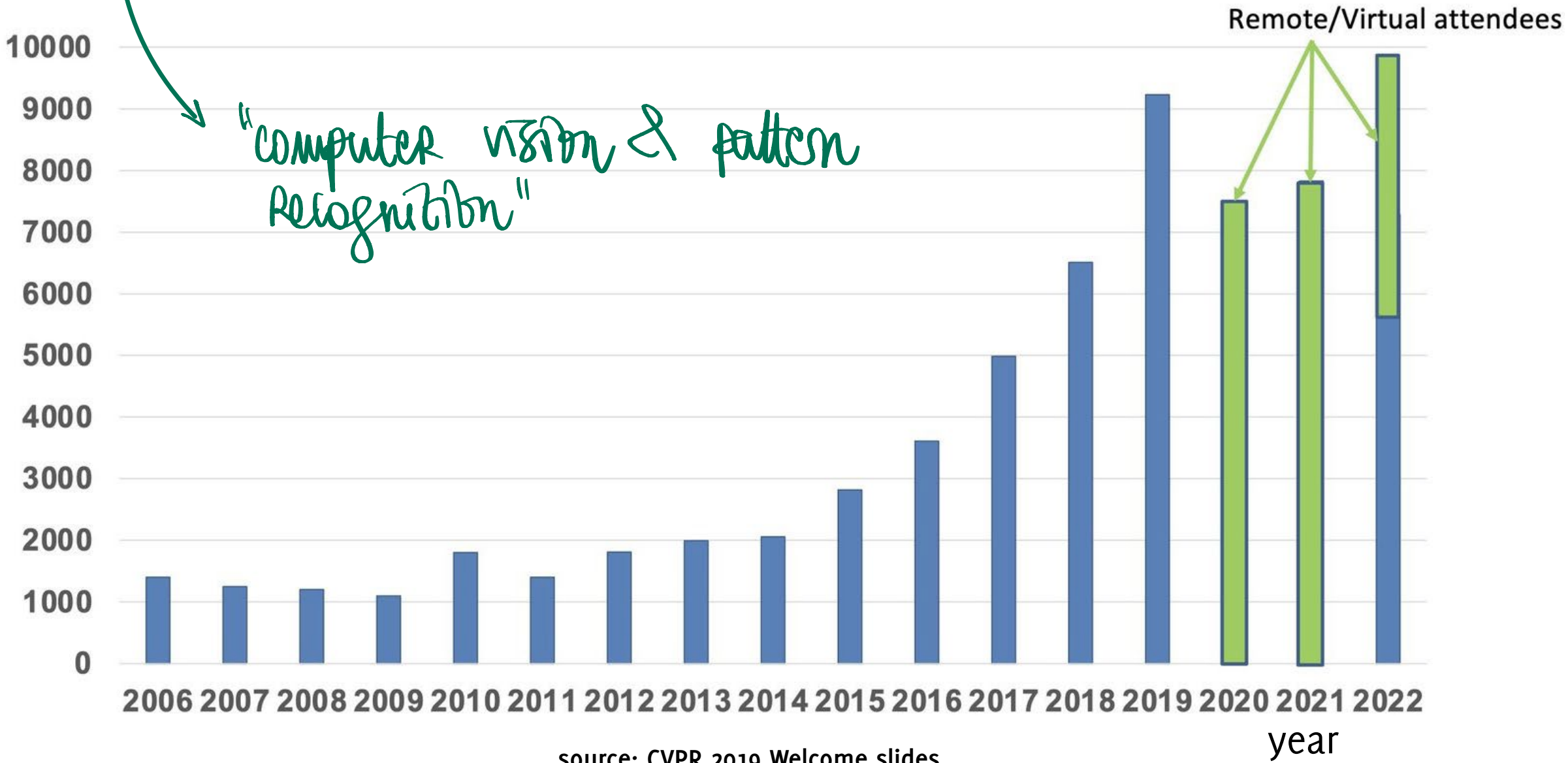
"black cat is sitting on top of suitcase."

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

Even though sometimes it fails



CVPR Attendance Trend (as of June 20, 2022)



source: CVPR 2019 Welcome slides

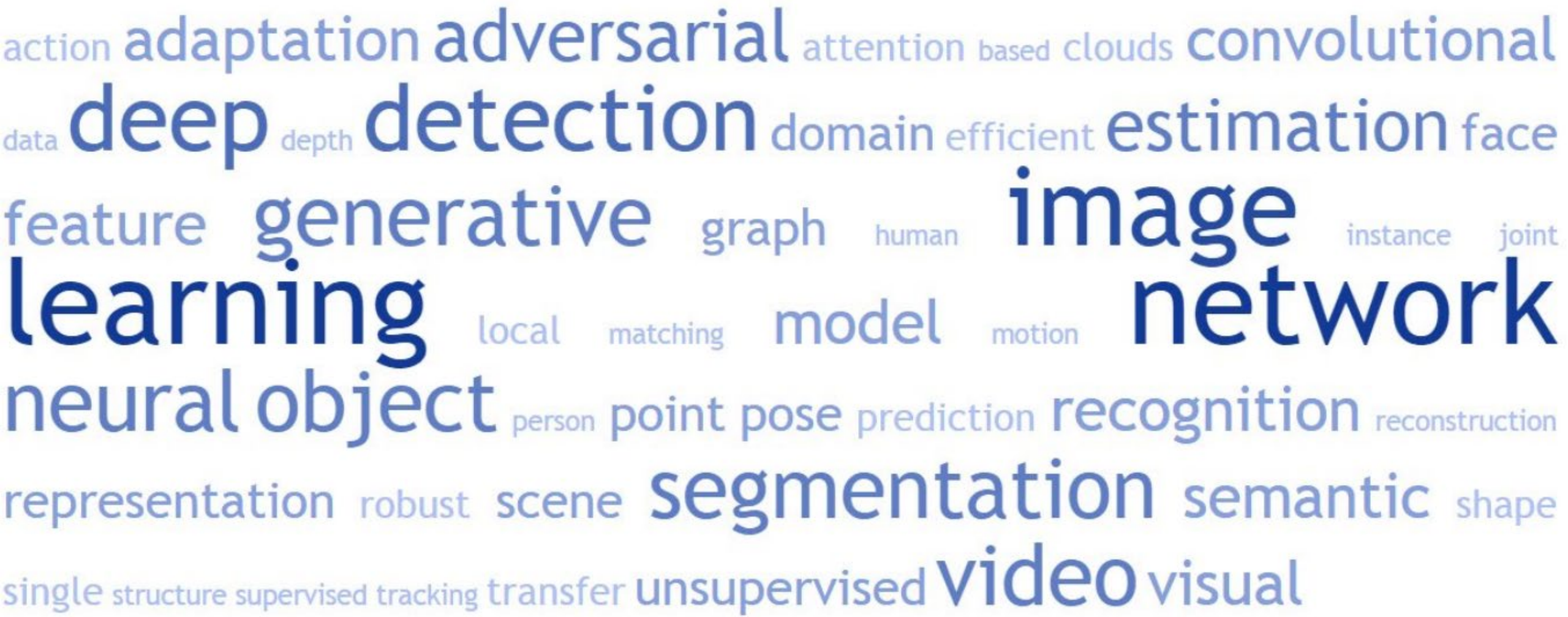
Lately, connection with ML

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

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

Acceptance rate is roughly even across topics



CVPR 2022

MENU Select Primary Subject Area		# papers	Oral	Poster
1	Recognition: detection, categorization, retrieval	177	25	152
2	Image and video synthesis and generation	157	26	131
3	3D from multi-view and sensors	137	26	111
4	Low-level vision	110	19	91
5	Vision + language	105	20	85
6	Segmentation, grouping and shape analysis	99	16	83
7	Transfer/ low-shot/ long-tail learning	86	15	71
8	Deep learning architectures and techniques	85	20	65
9	Self-& semi-& meta- & unsupervised learning	84	7	77
10	Video analysis and understanding	77	15	62
11	Pose estimation and tracking	62	14	48
12	Representation learning	61	11	50
13	3D from single images	60	10	50
14	Scene analysis and understanding	56	9	47
15	Face and gestures	54	7	47
16	Computational photography	53	10	43
17	Motion and tracking	53	8	45
18	Adversarial attack and defense	52	10	42
19	Datasets and evaluation	52	7	45
20	Machine learning	41	7	34
21	Action and event recognition	40	8	32
22	Efficient learning and inferences	40	3	37
23	Medical, biological and cell microscopy	37	5	32

Today…

We will start fresh from the simplest visual recognition problem:

Image Classification

Then we will move to the most advanced ones…

But first, let’s have a look at images

Digital Images

Photo Credits: Andrea Sanfilippo

RGB Images

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

Color image

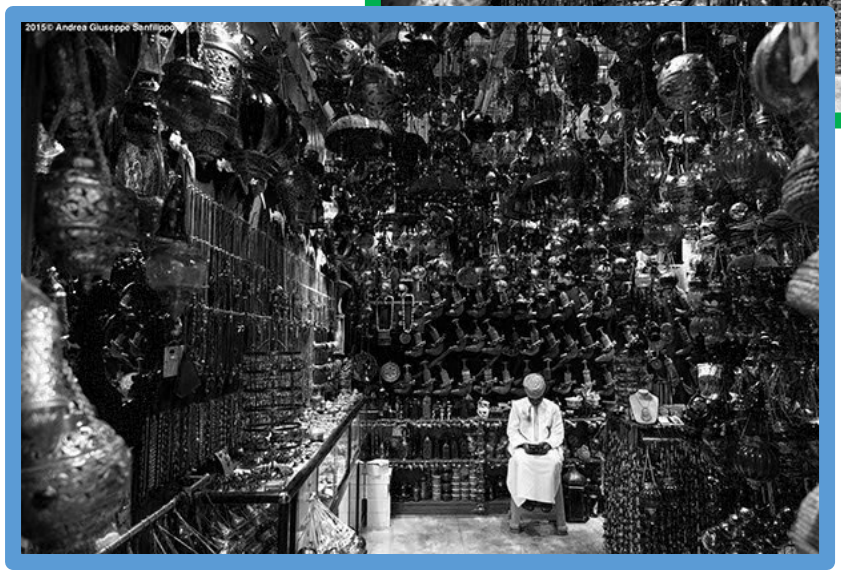
Red pixels.

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

green pixels.

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

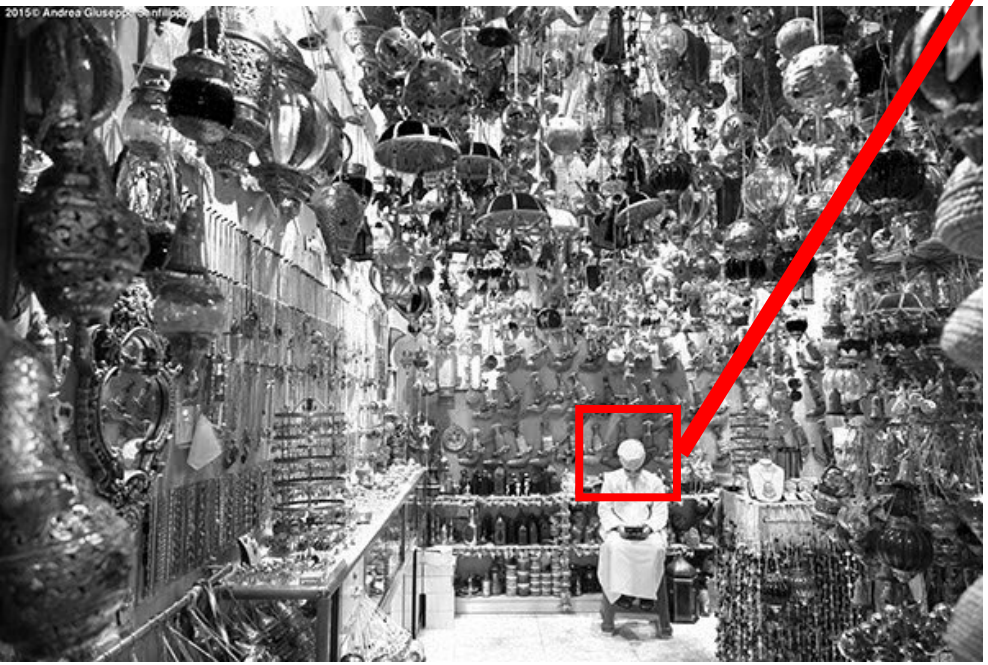
blue pixels.



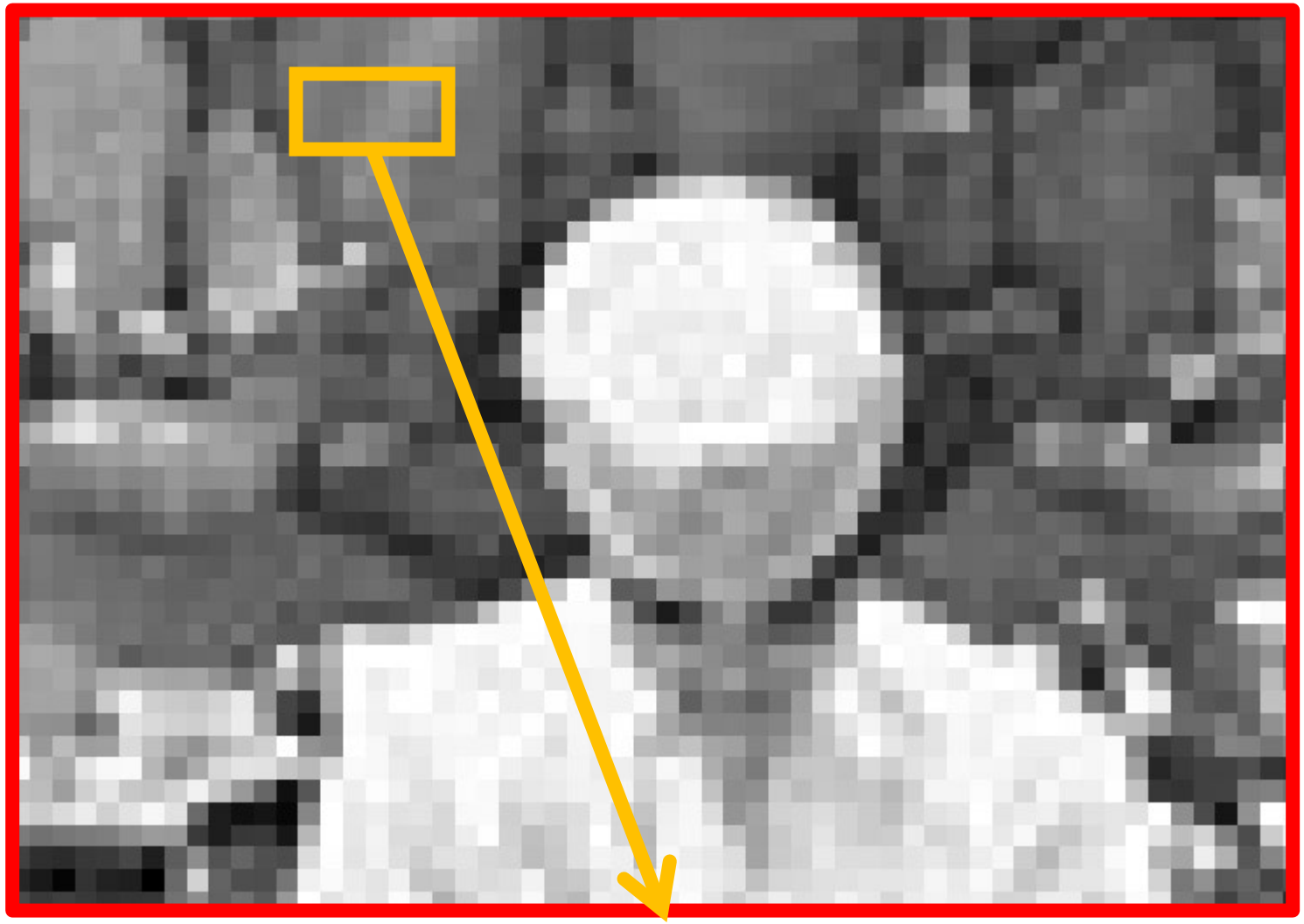
$B \in \mathbb{R}^{R \times C}$

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 Neural Network!

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

```python
from skimage.io import imread

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

# Extract the color channels

R = I[:, :, 0]
G = I[:, :, 1]
B = I[:, :, 2]
```

Python

```matlab
% Matlab
R = I(:, :, 1)
G = I(:, :, 2)
B = I(:, :, 3)
```

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: $R = 1080, C = 1920 \approx 6MB$ $\times 24\,s^{-1} \times 1s$

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** for images or vides

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

Local (Spatial) Transformations: Correlation

Most important image processing operations for classification problems

Local (Spatial) Transformation

These transformations mix all the pixels locally "around" the neighborhood U of a given pixel:

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

@ pixel (r,c).

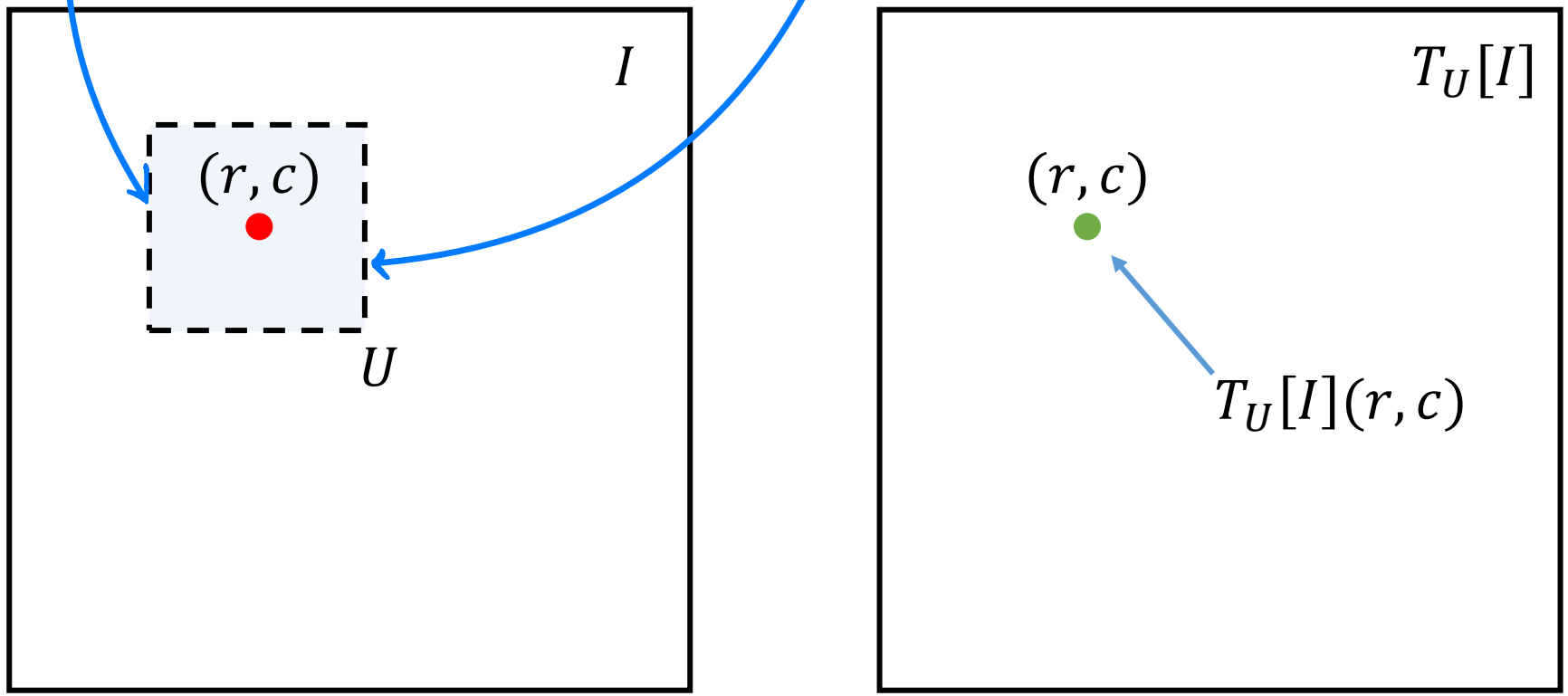
Where

- I is the input image
- G is the output
- U is a neighbourhood, identifies a region defining the output
- T_U: $\mathbb{R}^3 \to \mathbb{R}^3$ or T_U: $\mathbb{R}^3 \to \mathbb{R}$ is a spatial transf. function

The output at pixel (r,c) i.e., $T_U[I]\,(r,c)$ is defined by all the intensity values: $\{I(u,v),\ \ (u-r, v-c) \in U\}$

Local (Spatial) Filters

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

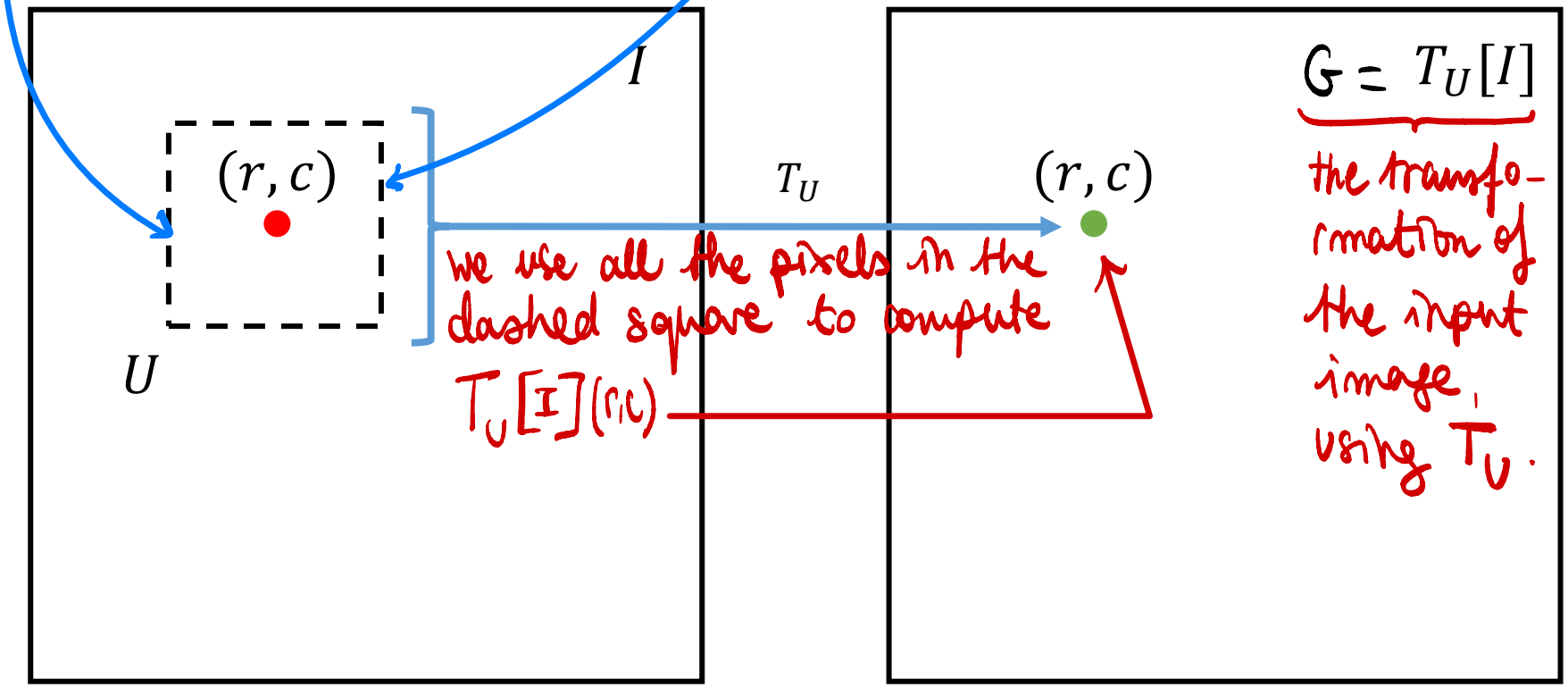


Input Image

Output Image

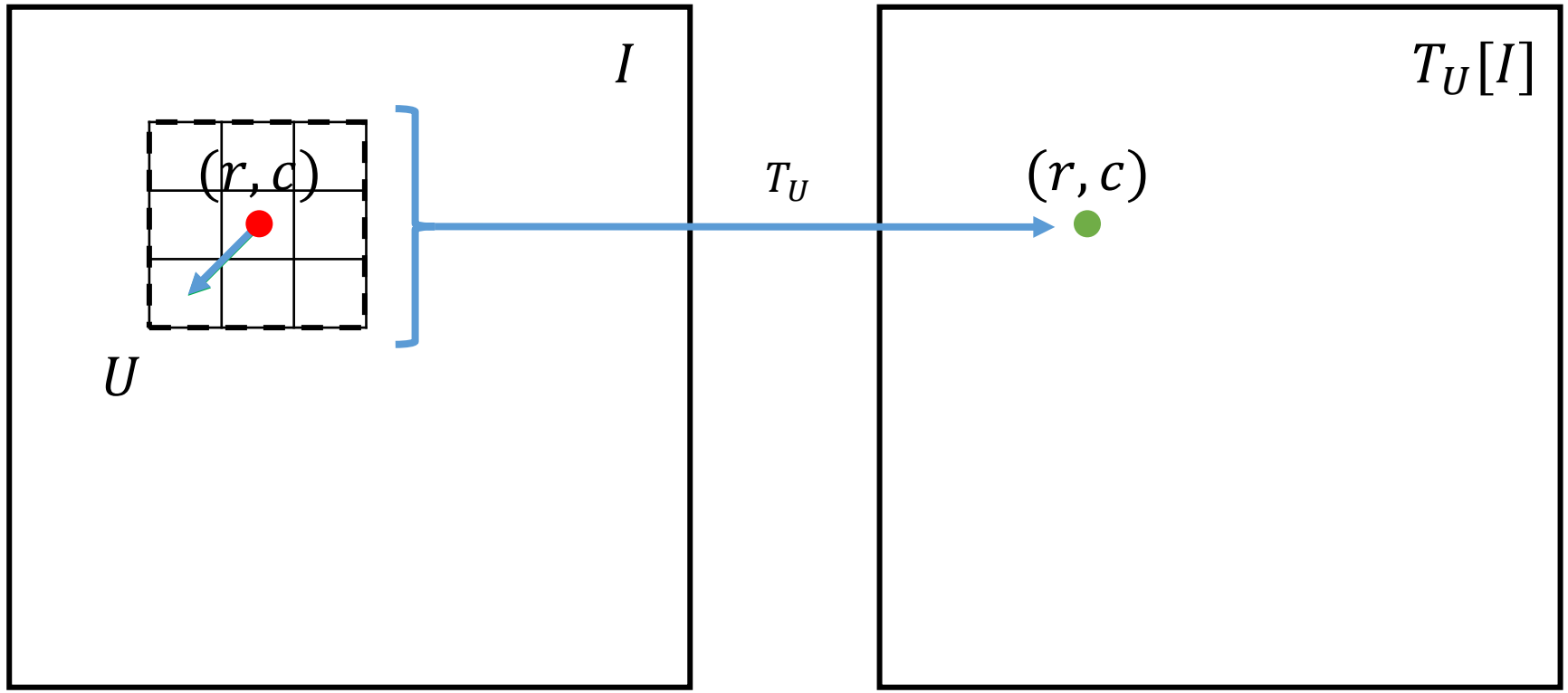
Local (Spatial) Filters

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



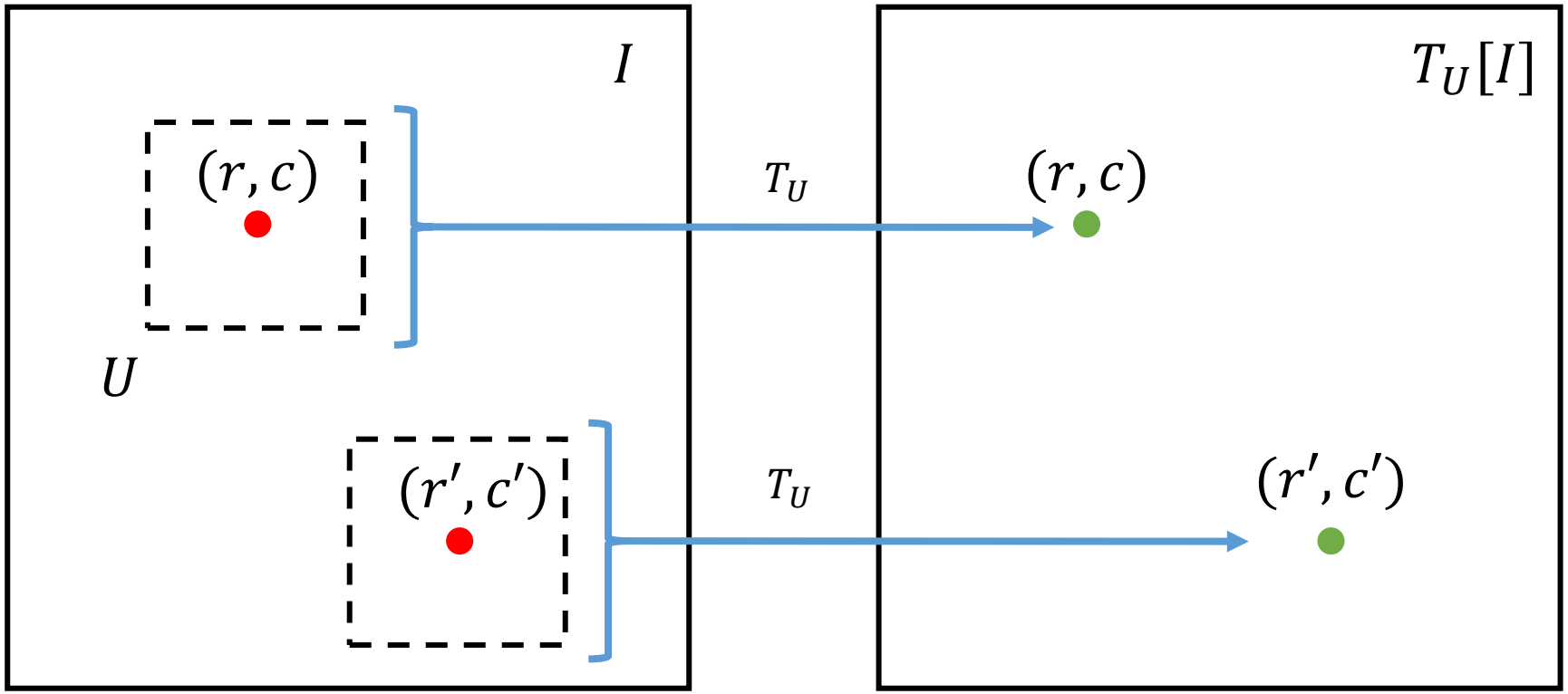
Local (Spatial) Filters

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



And (u,v) has to be interpreted as a "displacement vector" w.r.t. the neighborhood center (r,c), e.g., $(u,v) \in \{(1,-1),(1,0),(1,-1)\ldots\}$

Local (Spatial) Filters



- **Space invariant transformations** are repeated for each pixel (don't depend on r, c)
- T can be either linear or nonlinear

Local Linear Filters

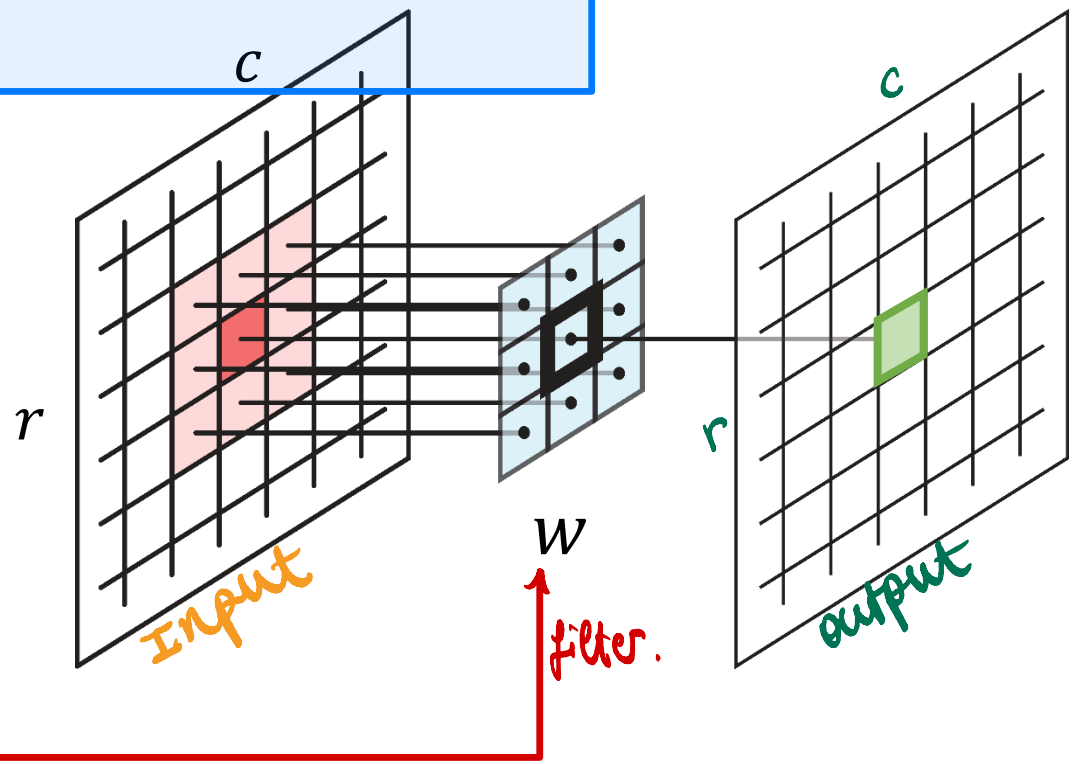
Linear Transformation: Linearity implies that the output $T[I](r,c)$ is a linear combination of the pixels in U:

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

Considering *some* *weights* $\{w_i\}$

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

The filter h entirely defines this operation



Local Linear Filters

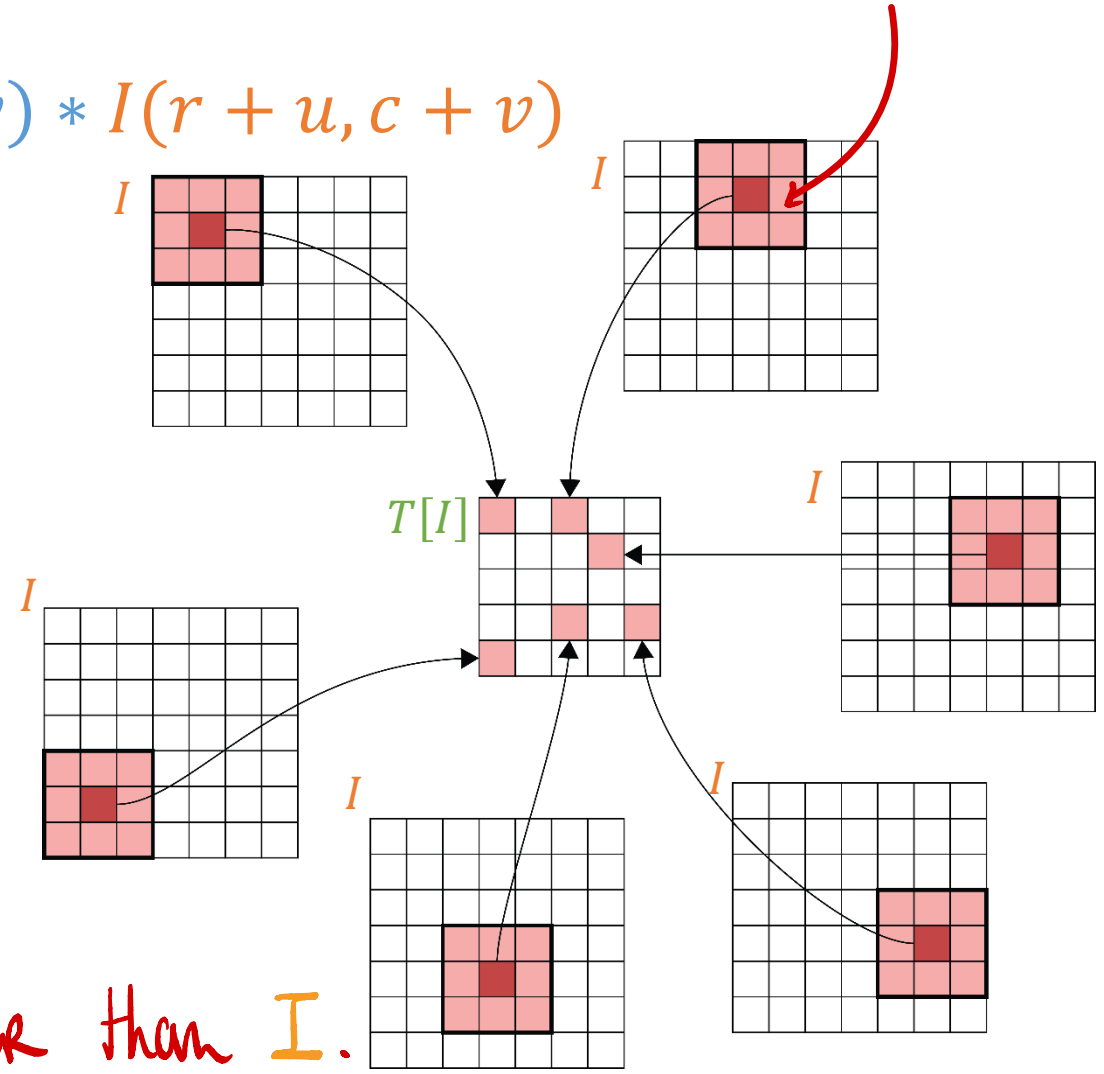
Linear Transformation: the filter weights can be assoicated to a matrix $\boldsymbol{w}$

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

$\boldsymbol{w}$

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

This operation is repeated for each pixel in the input image



As you can see, T[I] is smaller than I.

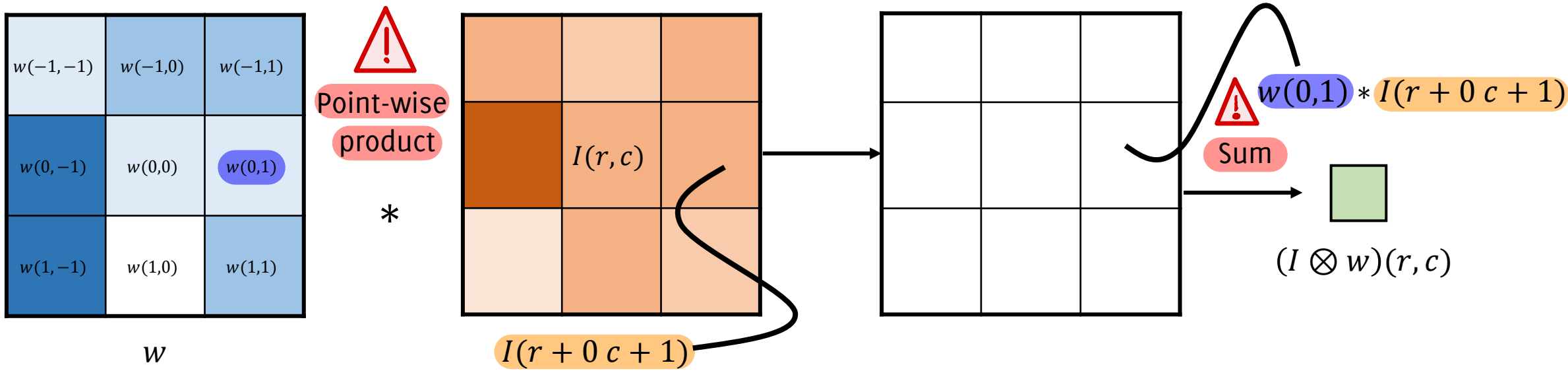
Correlation

The **correlation** among a filter $w = \{w_{ij}\}$ and an image is defined as

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

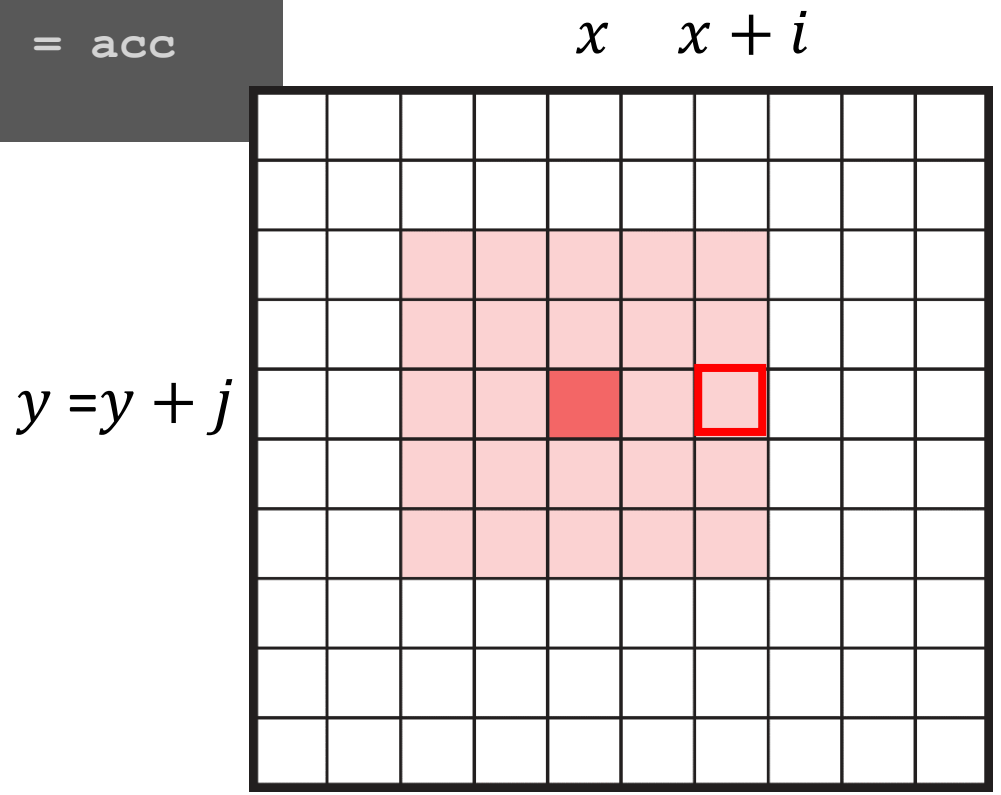
"It's the sum in a square window of the point-wise product"

where the filter h is of size $(2L+1) \times (2L+1)$ and contains the weights defined before as w. The filter w is also sometimes called "kernel"



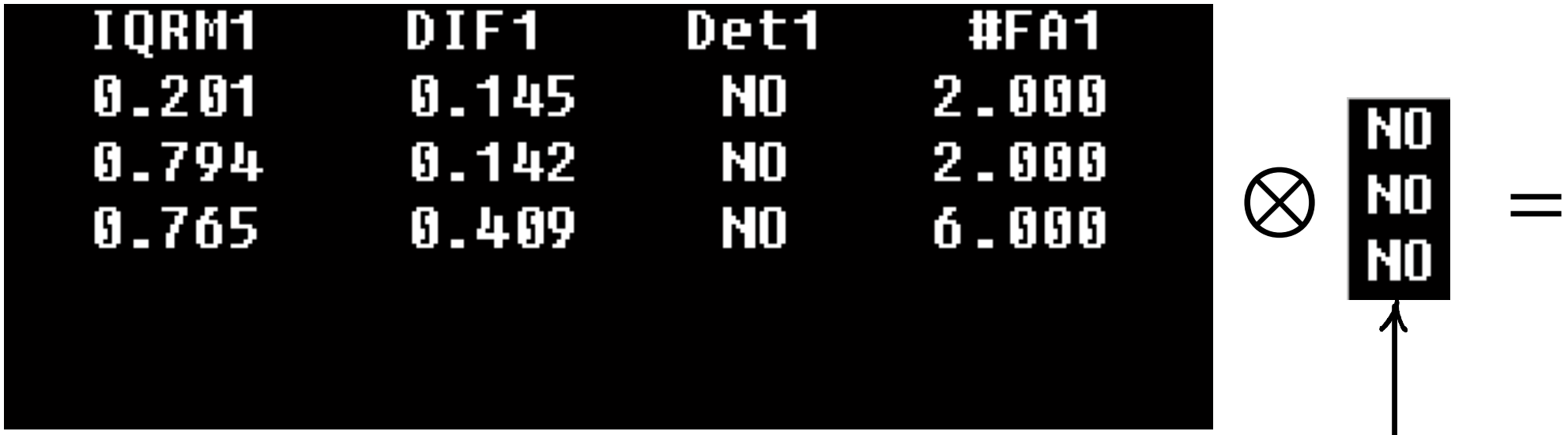
Correlation

```
acc = 0;
for i in np.arange(template_height):
        for j in np.arange(template_width):
                acc = acc + image[y + i, x + j]*template[i,j]
image[x+ template_height//2, y + template_width//2] = acc
```



Correlation for BINARY target matching

I w



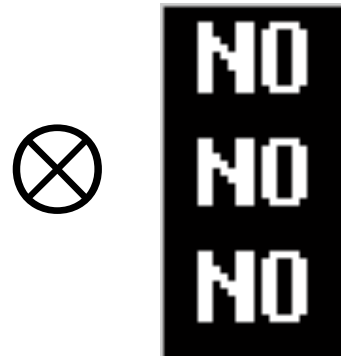
Easy to understand with binary images

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

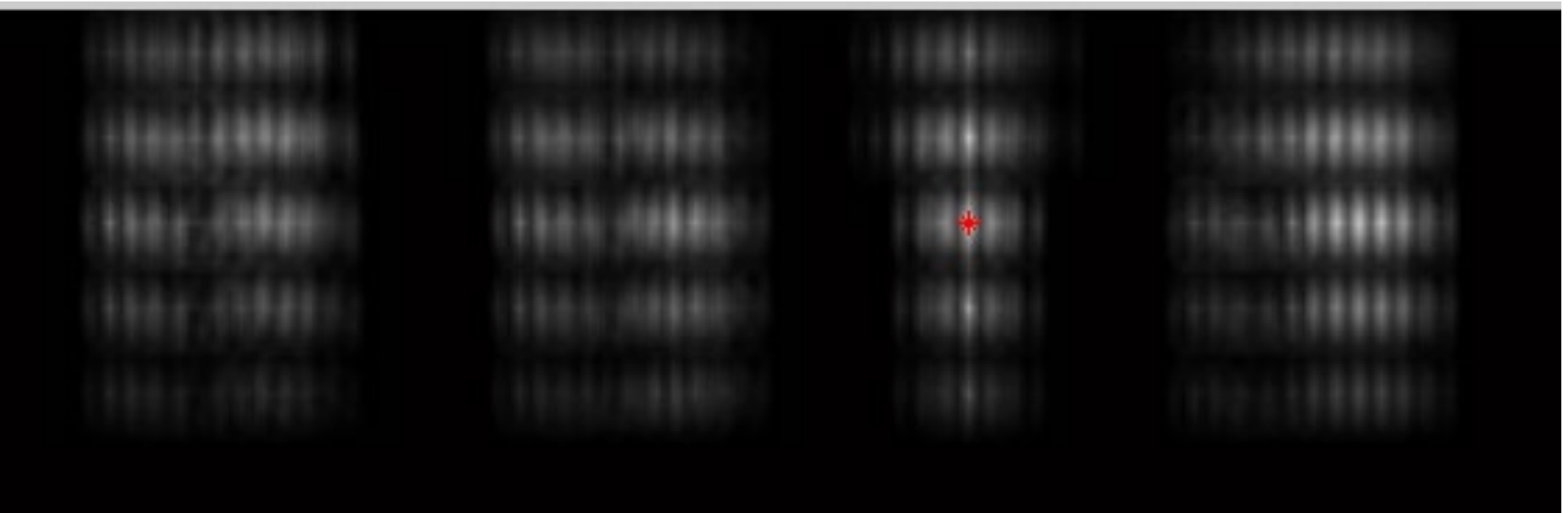
NOI	QRM1	DIF1	Det1	#FA1
NOI	.201	0.145	NO	2.000
NOI	.794	0.142	NO	2.000
	0.765	0.409	NO	6.000



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



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

Because it's the part of the image that is like the template!
→ But as we sum everything, it's not perfect.

The maximum is here

However…

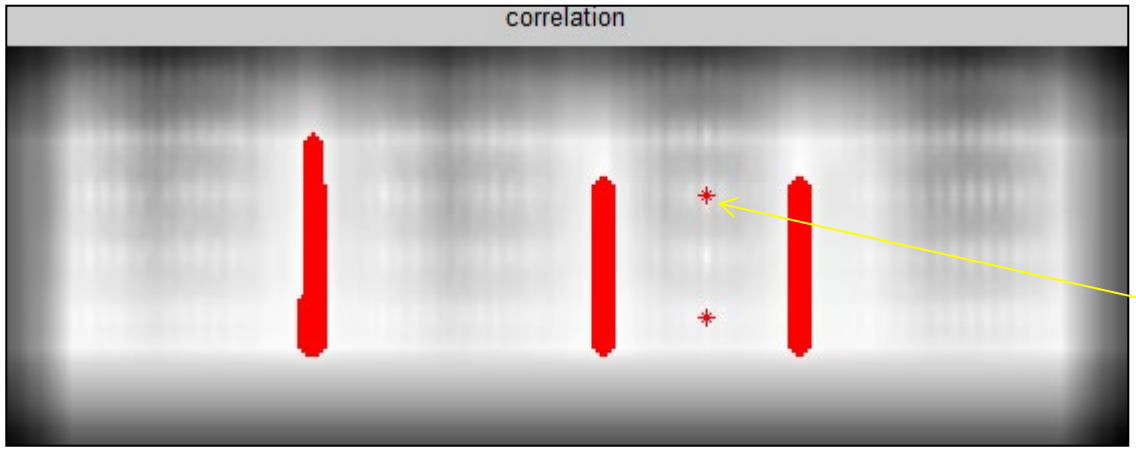
y, original image

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)

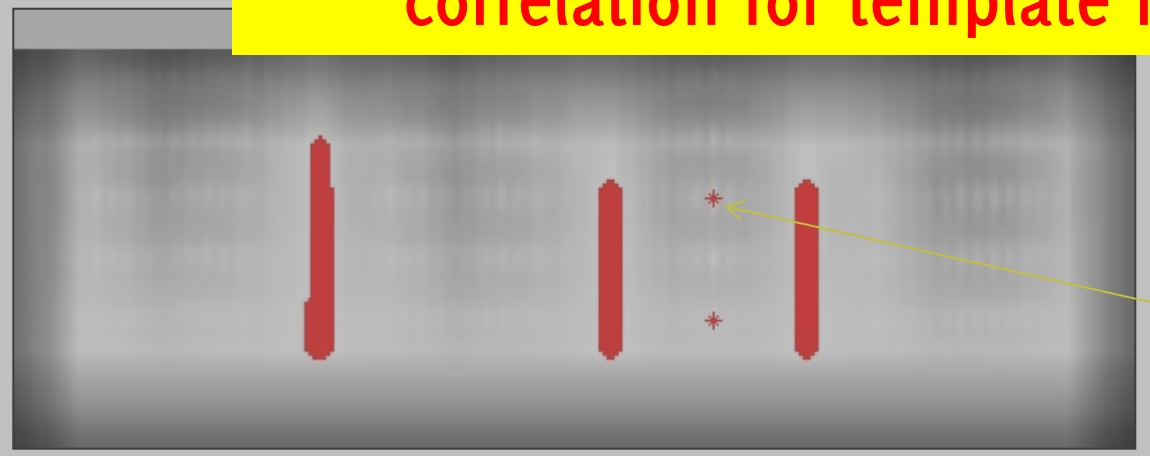
However...

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



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)

Correlation

The correlation formula holds even when inputs are grayscale images

grayscale:

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

And even when these are RGB images

RGB:

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

The Image Classification Problem

Image Classification

$\Lambda = \{\text{"wheel"}, \text{"cars"}, \ldots, \text{"castle"}, \text{"baboon"}\}$

x

⇨ "wheel"

{ Image = input
Label = output

x

⇨ "castle"

Image Classification

$\Lambda = \{\text{"wheel"}, \text{"cars"}, ..., \text{"castle"}, \text{"baboon"}\}$

x

⇨ "wheel" 65%, "tyre" 30%..

Another type of classification: this one with probabilities.

x

⇨ "castle" 55%, "tower" 43%..

Image Classification, the problem

Assign to an input image $I \in \mathbb{R}^{R \times C \times 3}$:

- a label y from a fixed set of categories Λ

The image classification problem.

The classifier is a function f_θ

$$f_\theta : I \to f_\theta(I) \in \Lambda$$

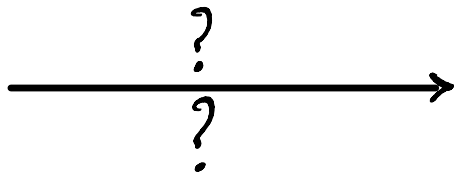
$$\mathbb{R}^{R \times C \times 3} \longrightarrow \Lambda$$

Linear Classifier

The basic building block for deep architectures

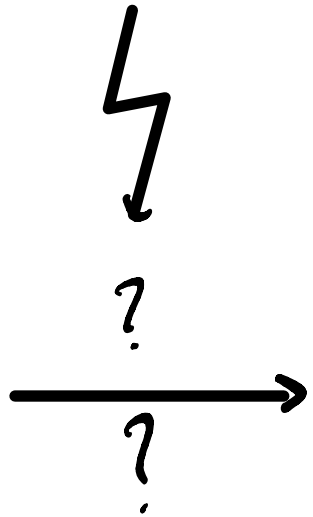
How to feed images to NN?

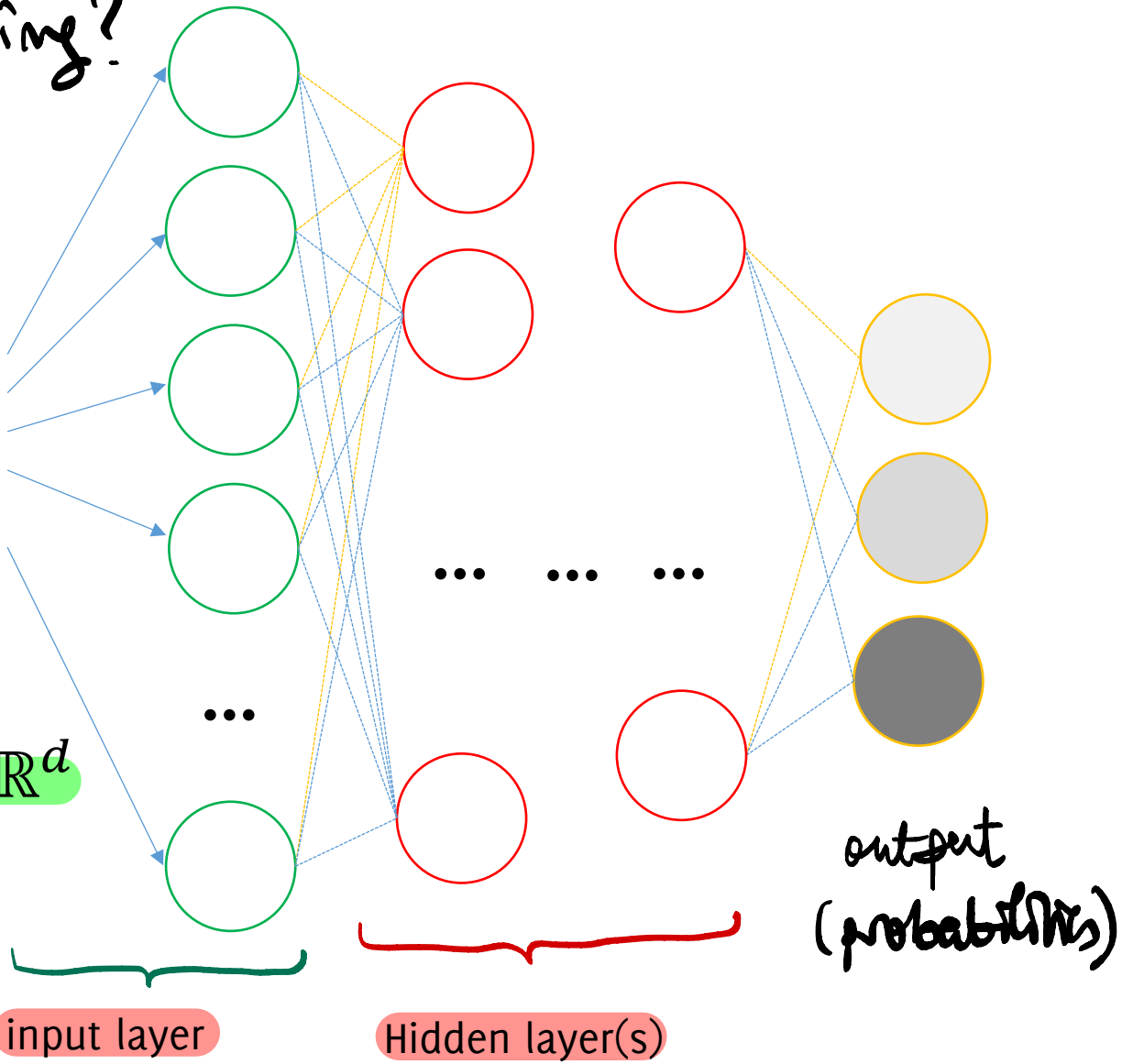
$I \in \mathbb{R}^{R \times C \times 3}$



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

how to "send" the img?

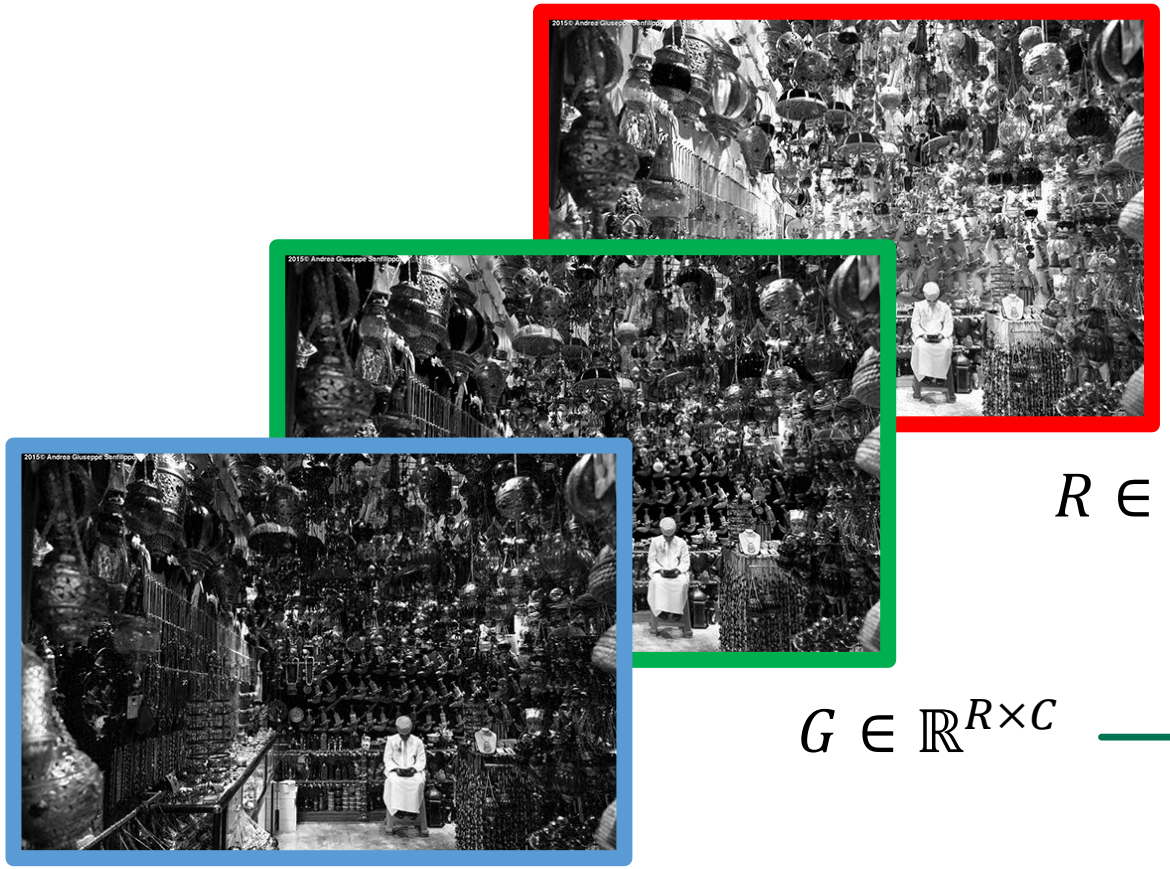




Column-wise unfolding

Flat the images : make them vectors.

Colors recall the color plane where images are from



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

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

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

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

R

G

B

RxC

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

$d = R \times C \times 3$

What we get.

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

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

32×32×3 (R×C×3)

airplane

automobile

bird

cat

deer

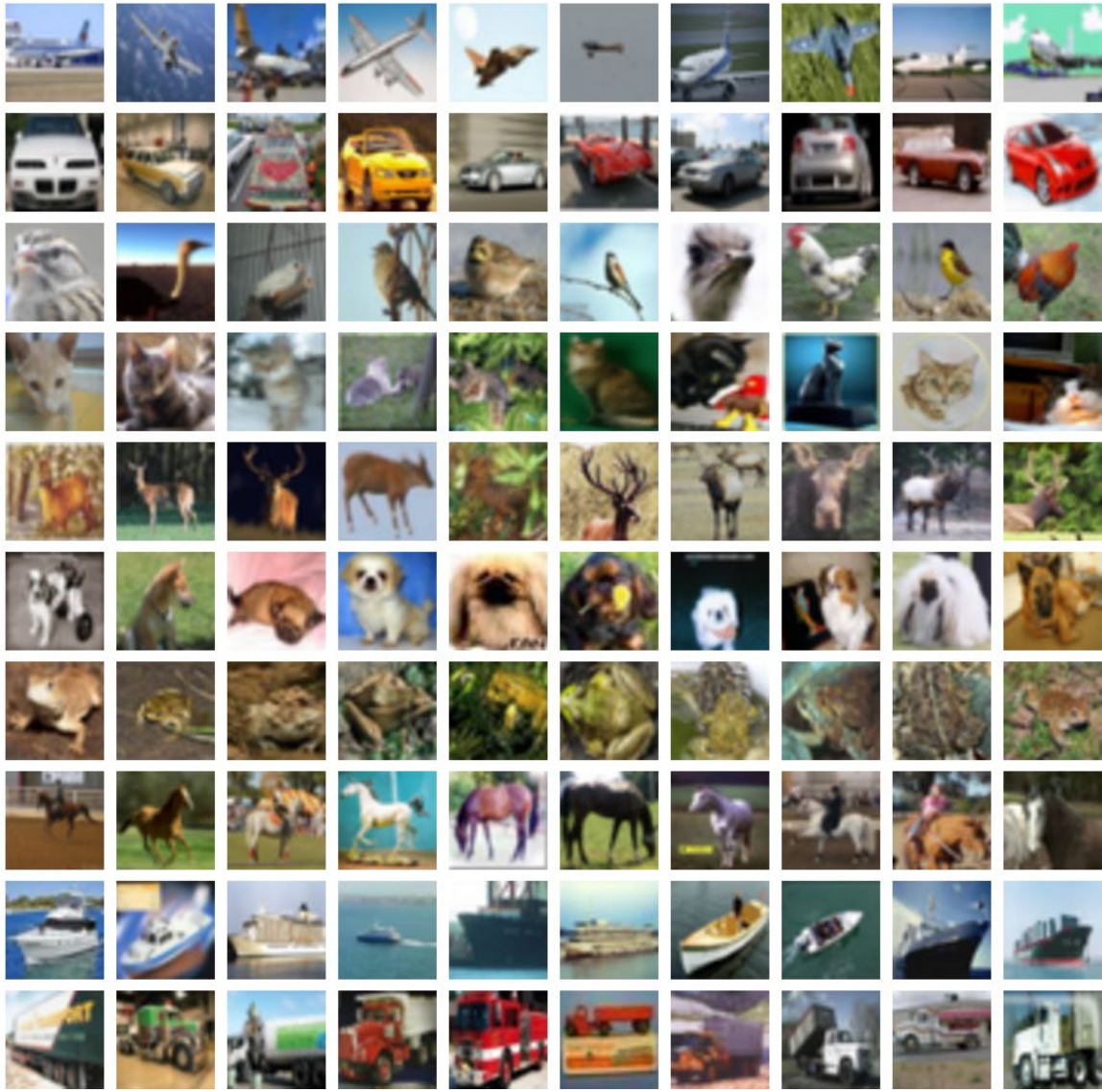
dog

frog

horse

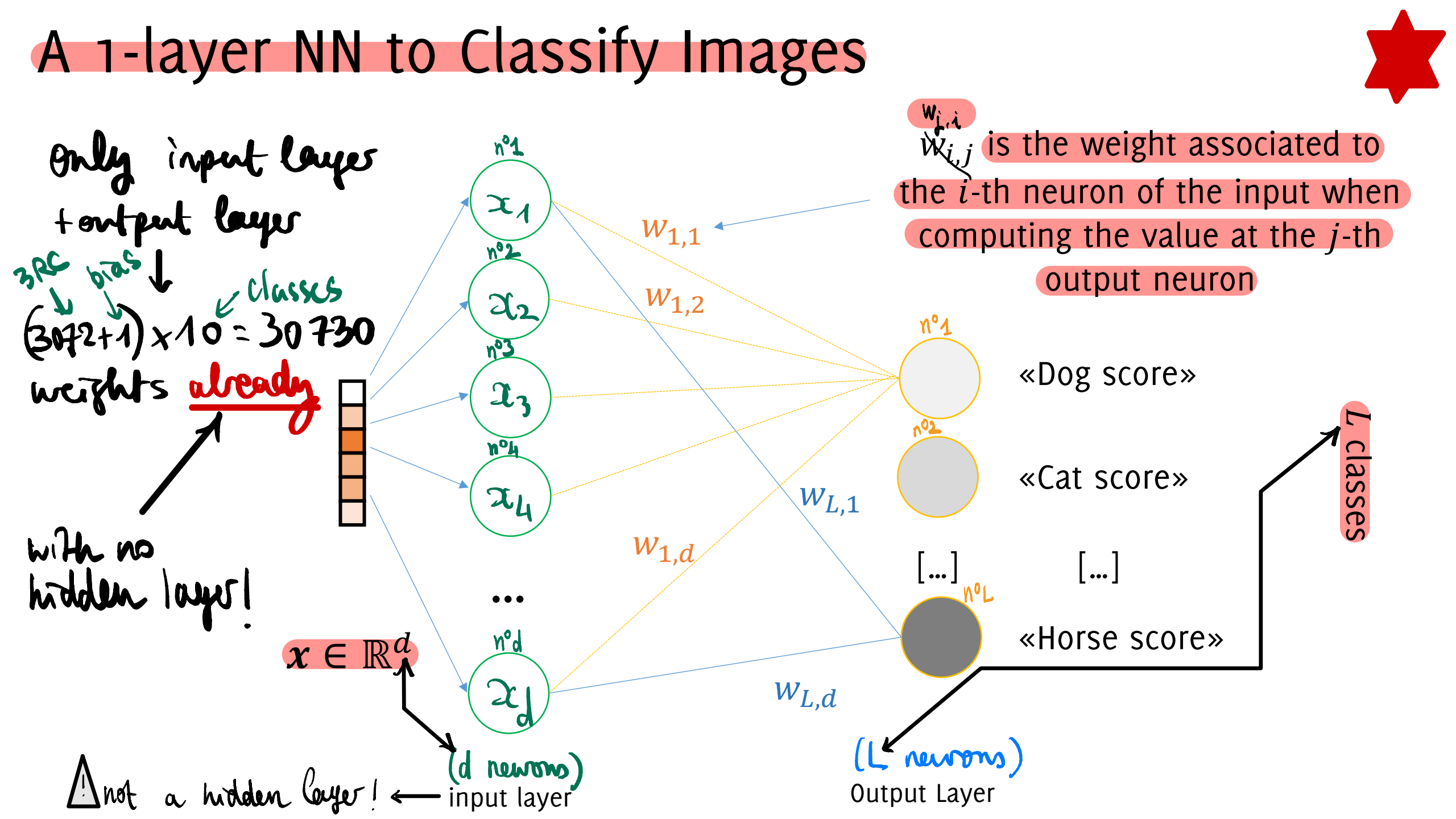
ship

truck

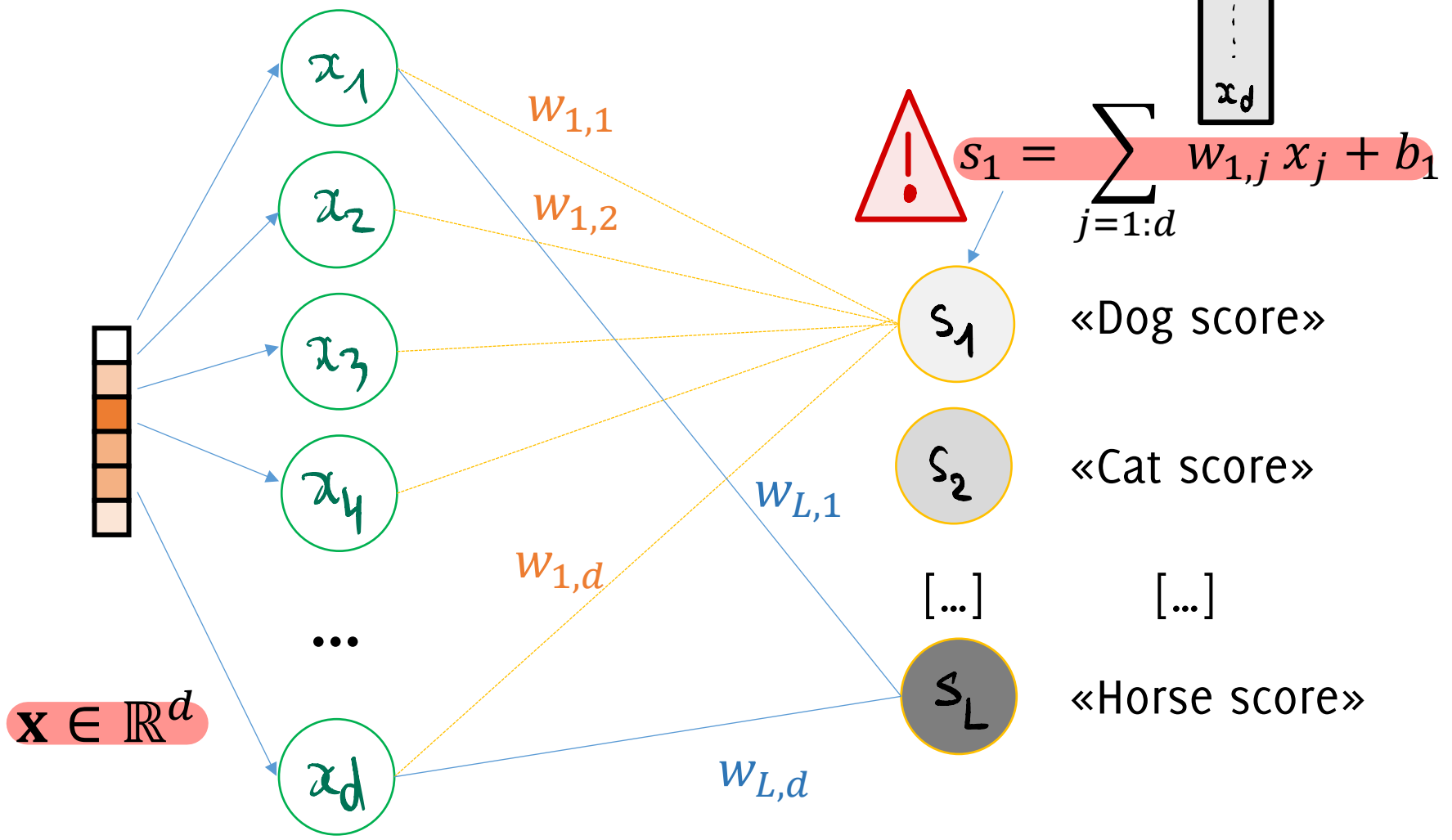


Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

A 1-layer NN to Classify Images



A 1-layer NN to Classify Images



$$\begin{bmatrix} w_{11} & \cdots & w_{1d} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} + b_1 = s_1$$

This is a linear classifier

$$s_1 = \sum_{j=1:d} w_{1,j}\, x_j + b_1$$

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
```

↑ # of slides 49 & 50

30 730 = (3072+1) × 10

Why don't we take a larger network?

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

As an example:

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

Note that those images are very small: RGB 32x32...

On CIFAR10 images, the number of neurons would be:

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

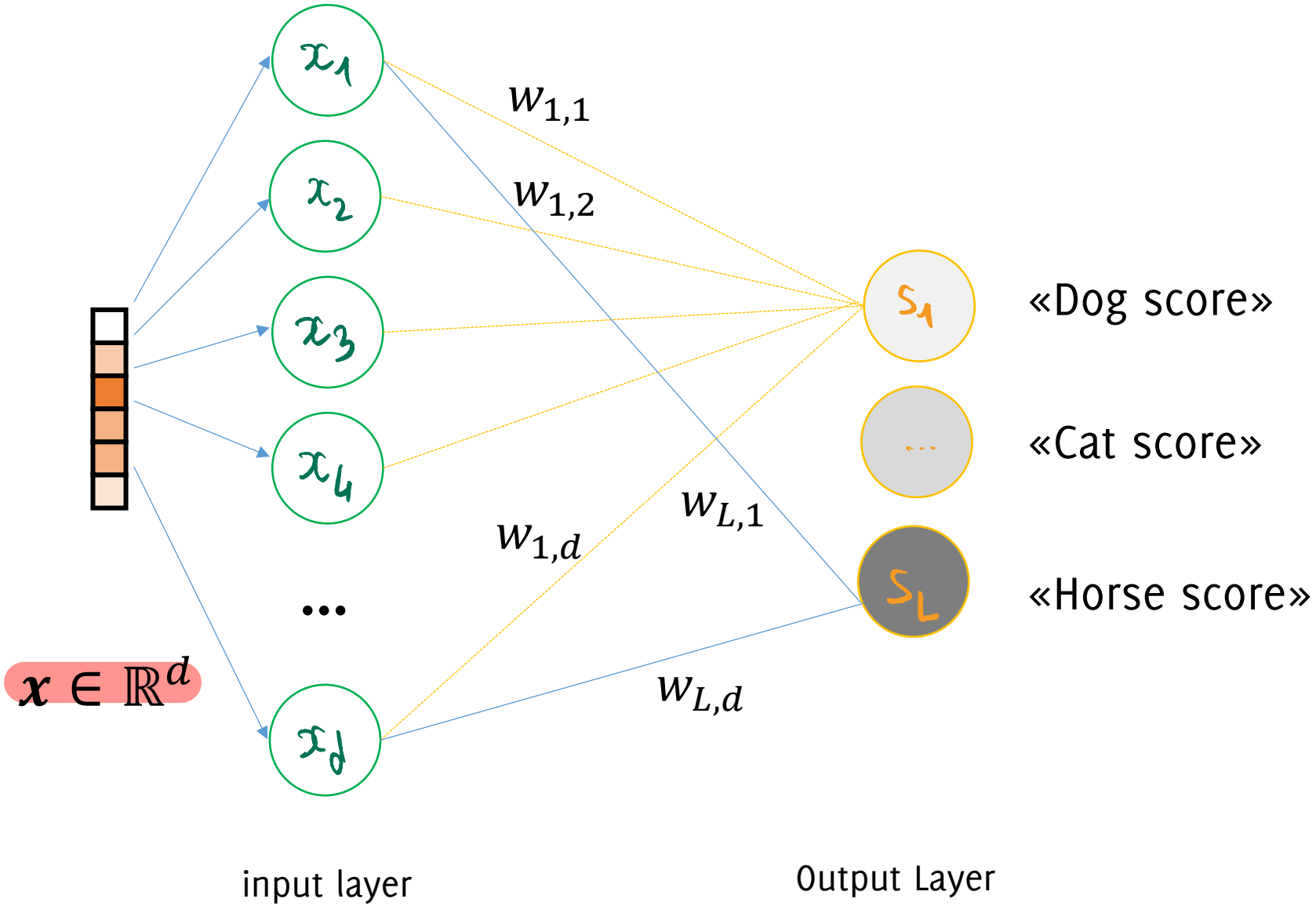
1,536 * 3,072 + 1,536 = **4,720,128 parameters (!)**

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

PROBLEM!

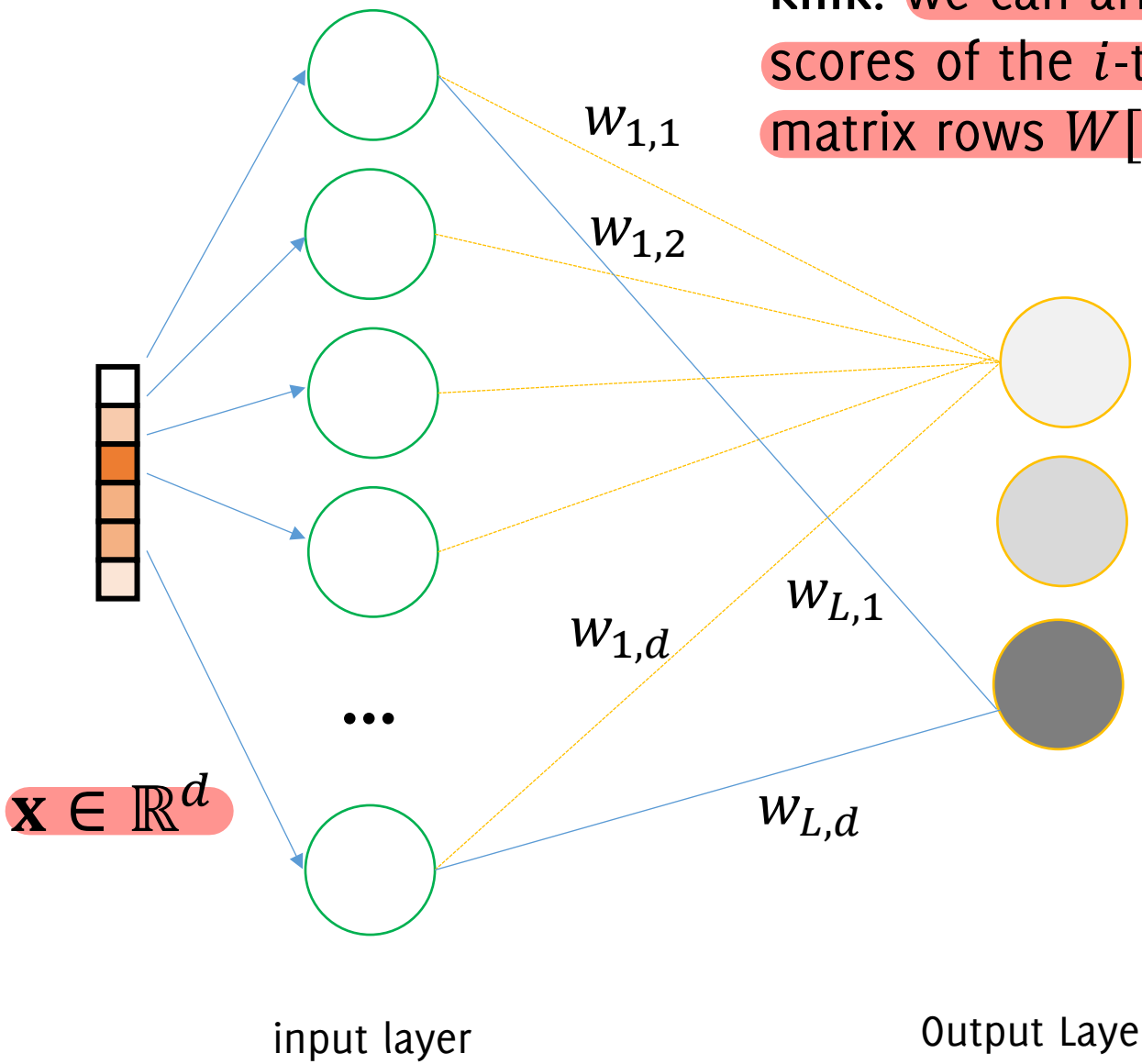
A 1-layer NN to Classify Images

We stick with it for the moment ...



we can train this very simple network.

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:

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

LINEAR CLASSIFIER

«Dog score»

«Cat score»

«Horse score»

L classes

$W =$ $\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ \vdots & & & \vdots \\ w_{L1} & \cdots & & w_{Ld} \end{bmatrix}$

No need a softmax here: just take the biggest score s_i (probability).

It's completely a linear classifier. I mean, if we had hidden layers of the same type we would get:

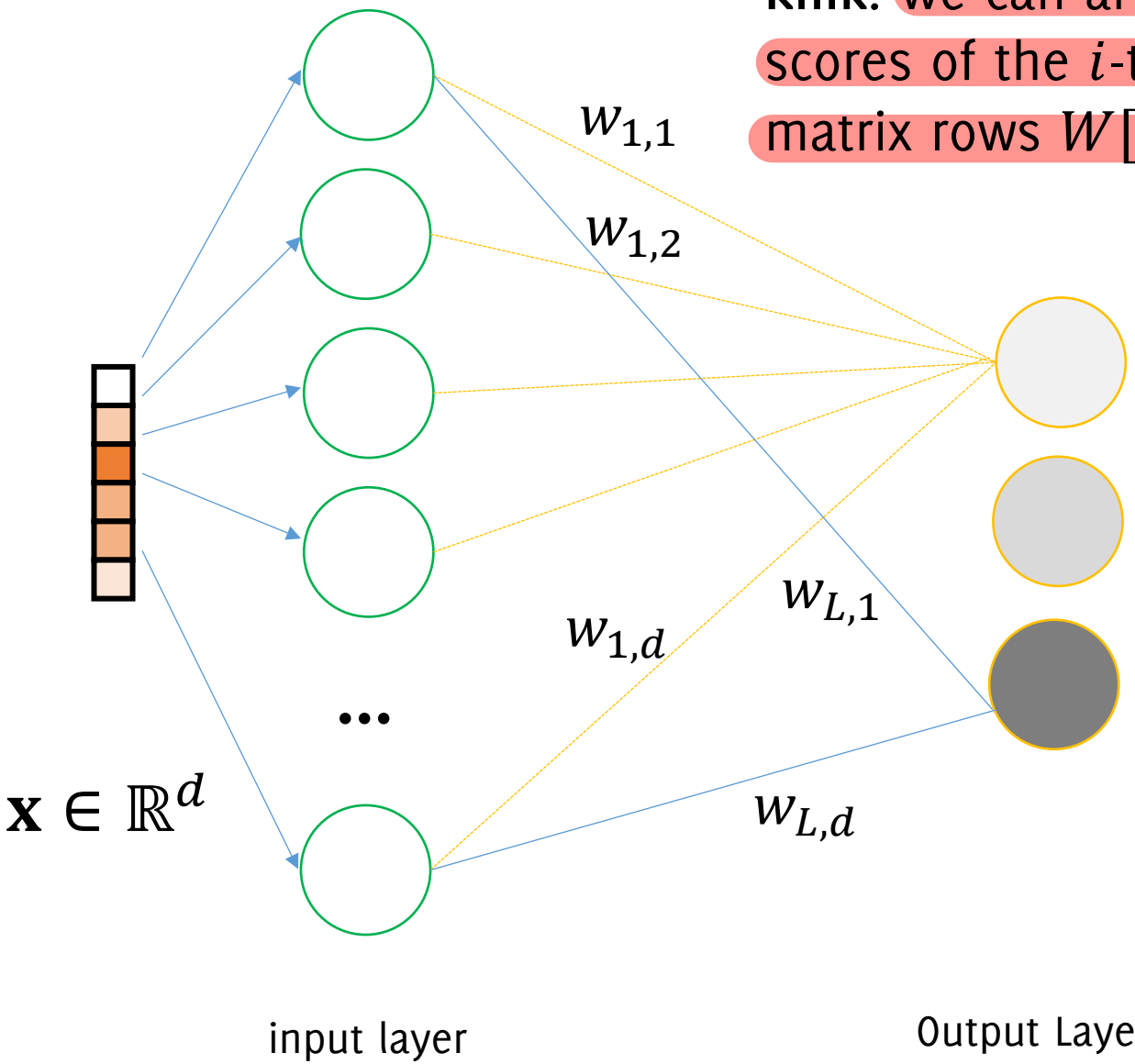
$$W^{(3)}\left(W^{(2)}\left(W^{(1)}\underline{x}+b^{(1)}\right)+b^{(2)}\right)+b^{(3)} \ldots \}$$ we can simplify

everything into the form "$Wx+b$".

⟶ Linear! That's why we need non linearity ⟶ otherwise, everything simplifies.

⟶ That's why we use in practice "$\tanh(W^{(1)}\underline{x}+b^{(1)})$" for ex.

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»

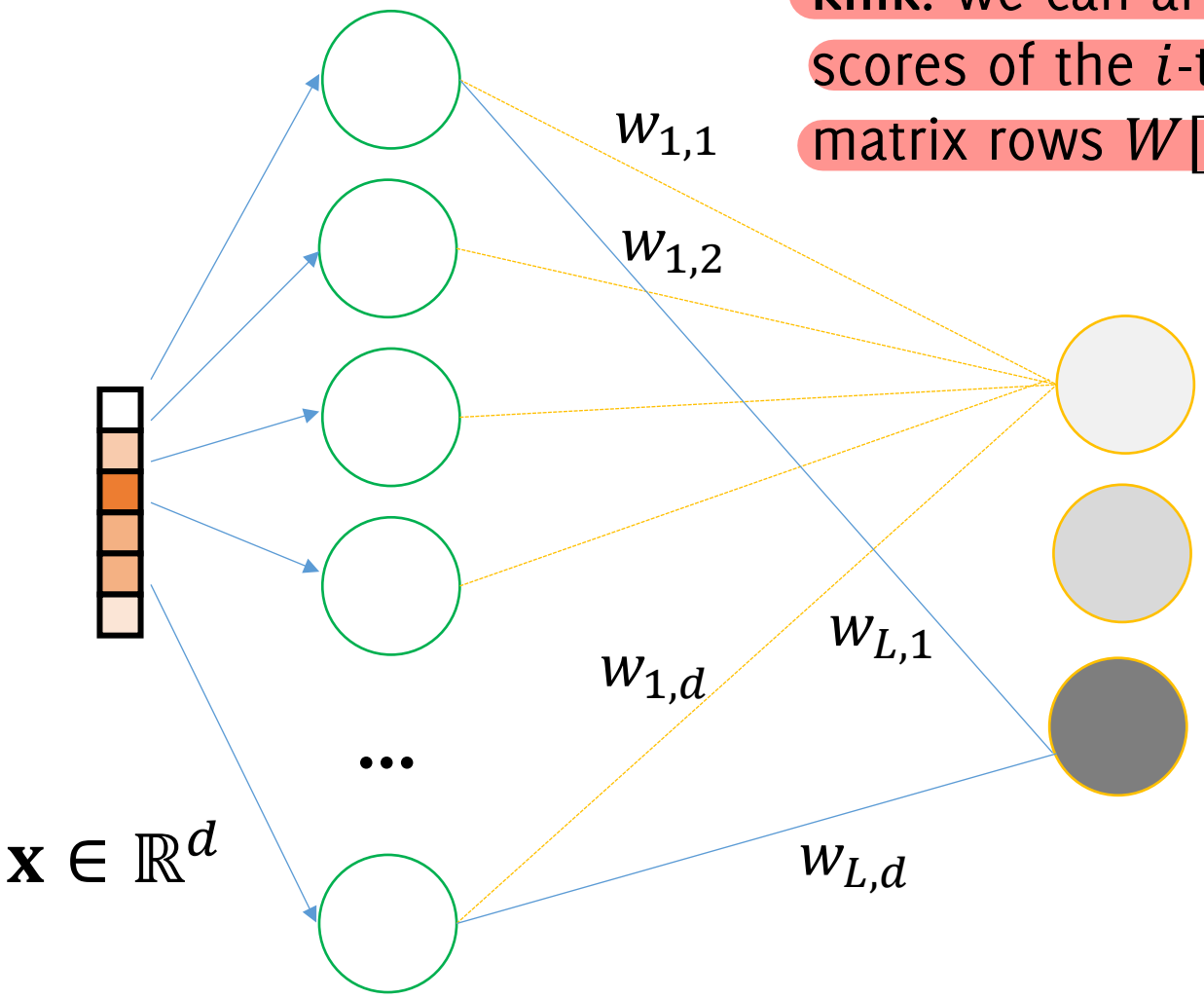
«Cat score»

«Horse score»

L classes

Rmk: nonlinearity is not needed here since there are no other layers after this

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 other layers after this

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

A 1-layer NN to Classify Images

VISUALISATION

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

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

-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

produces s_1.

produces s_2.

produces s_3.

$*$

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

$\boldsymbol{x}$

$+$

-2
32
-1

$\boldsymbol{b}$

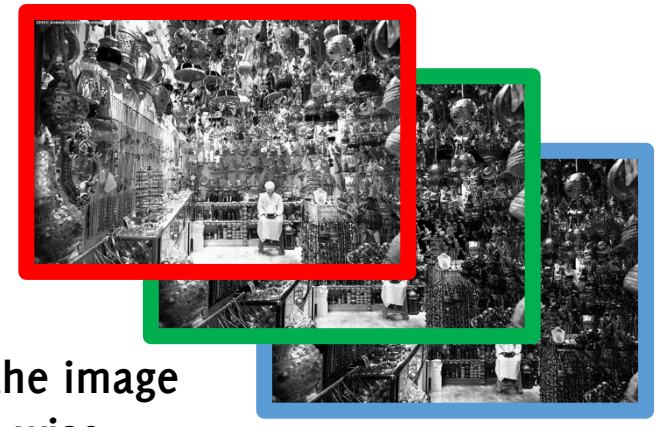
$=$

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

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

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

weights associated to red.



Unroll the image column-wise

Linear Classifiers for Images

An image classifier can be seen as a function that maps an image I to a confidence scores for each of the L class:

$$\mathcal{K}\colon \mathbb{R}^d \to \mathbb{R}^L$$

where $\mathcal{K}(I)$ is a L −dimensional vector and the i −th component

$$s_i = [\mathcal{K}(I)]_i$$

contains a score of how likely I belongs to class i.

A **good classifier associates to the largest score to the correct class!**

This simple layer is a linear classifier

In linear classification $\mathcal{K}$ is a linear function to map the unfolded image $\boldsymbol{x} \in \mathbb{R}^d$:

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

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 vector.

d

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 horse score

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

Unroll the image column-wise

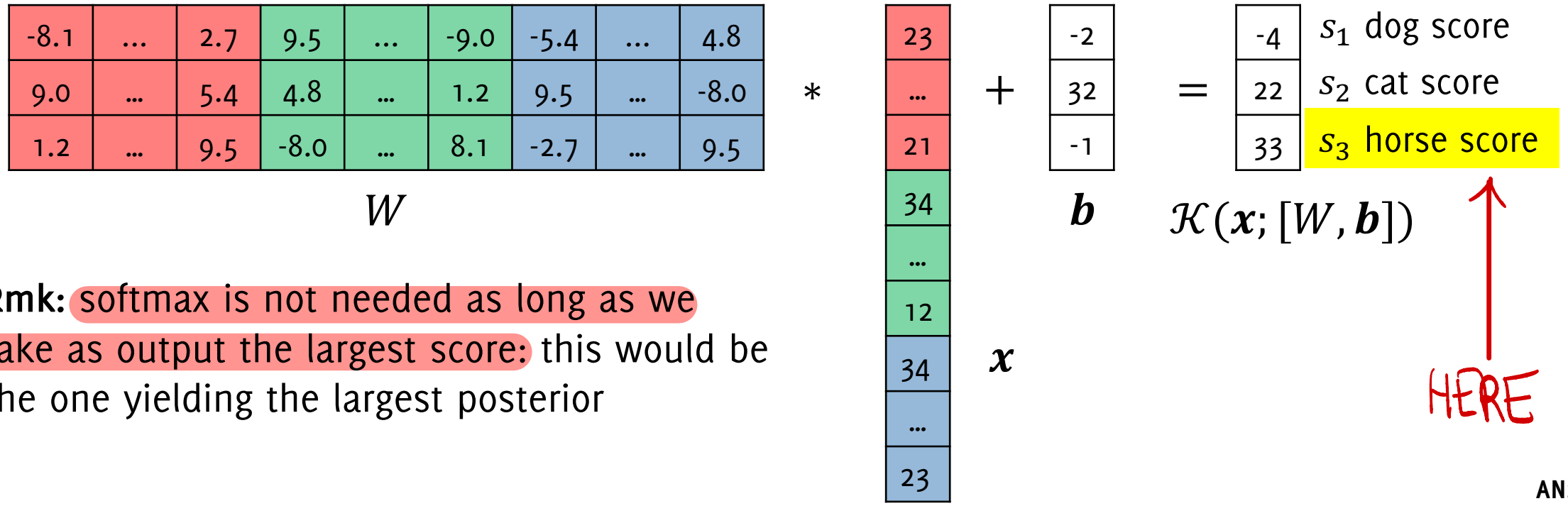
This simple layer is a linear classifier

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

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

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

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



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

The 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 / colours

Do we need nonlinear layers?

Each layer in a NN can be seen as matrix multiplication (+ bias).

$$\mathbf{s} = W\boldsymbol{x} + \boldsymbol{b}$$

If we stack 3 layers without activations:

$$\mathbf{s} = \big((W_1\boldsymbol{x} + \boldsymbol{b}_1)\mathrm{W}_2 + \boldsymbol{b}_2\big)W_3 + b_3$$

This is equivalent to

$$\mathbf{s} = \widetilde{\boldsymbol{W}}\boldsymbol{x} + \widetilde{\boldsymbol{b}}$$

for some $\widetilde{\boldsymbol{W}} \in \mathbb{R}^{L\times d}, \widetilde{\boldsymbol{b}} \in \mathbb{R}^{L}$… thus stacking many layers w/o nonlinear activations is pointless as it gets to a linear classifier.

This is a further confirmation why it becomes pointless to stack many layers without including a nonlinear activations…

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

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

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 gradient descent and all its variants (see Prof. Matteucci classes) ⟵ First lectures.

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

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

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

REGULARISATION.

… Once Trained

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

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

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

$\boldsymbol{W}[\boldsymbol{i},:]$ 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** (and summing the bias)

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

Thus, the NN computes the inner products against L different weights vectors, and selects 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 by 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:

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

Python syntax.

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 $\mathbb{R}^d$

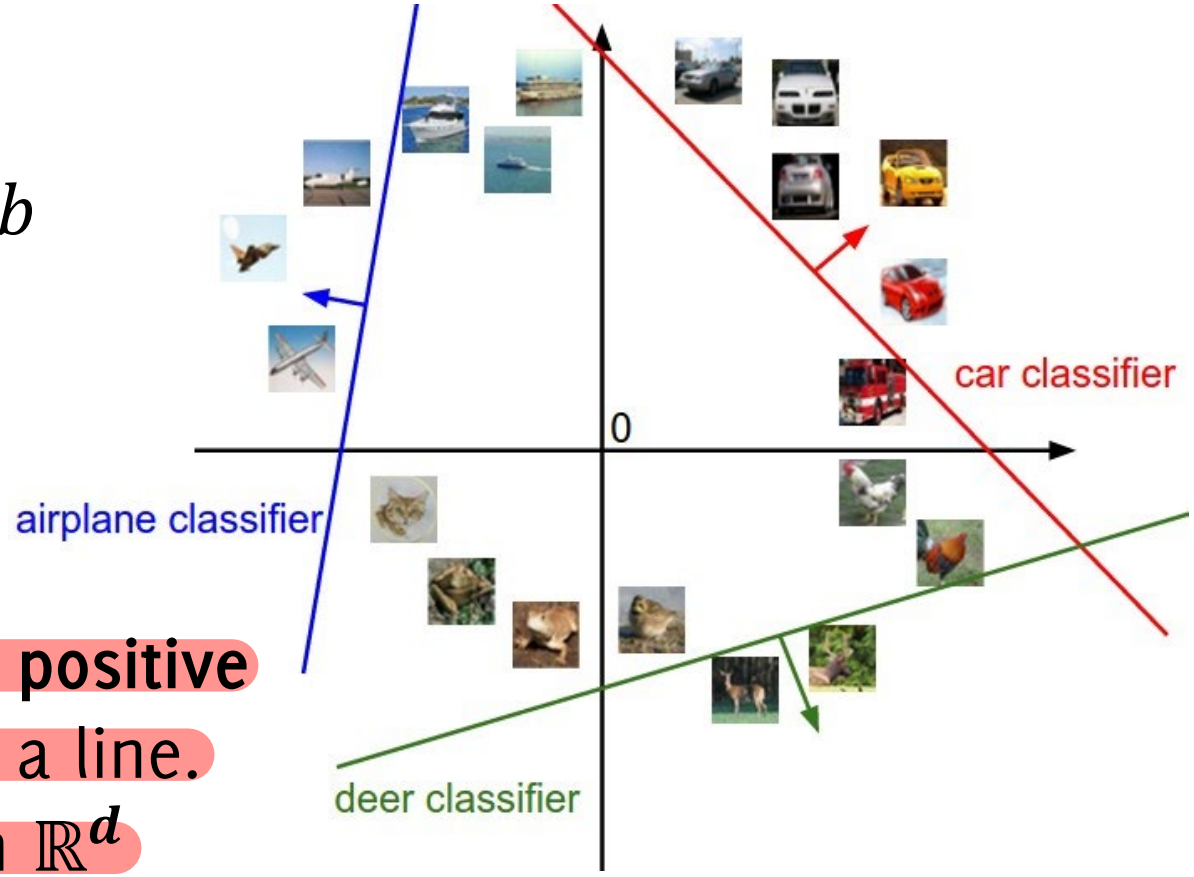
In $\mathbb{R}^2$ these would be

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

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

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

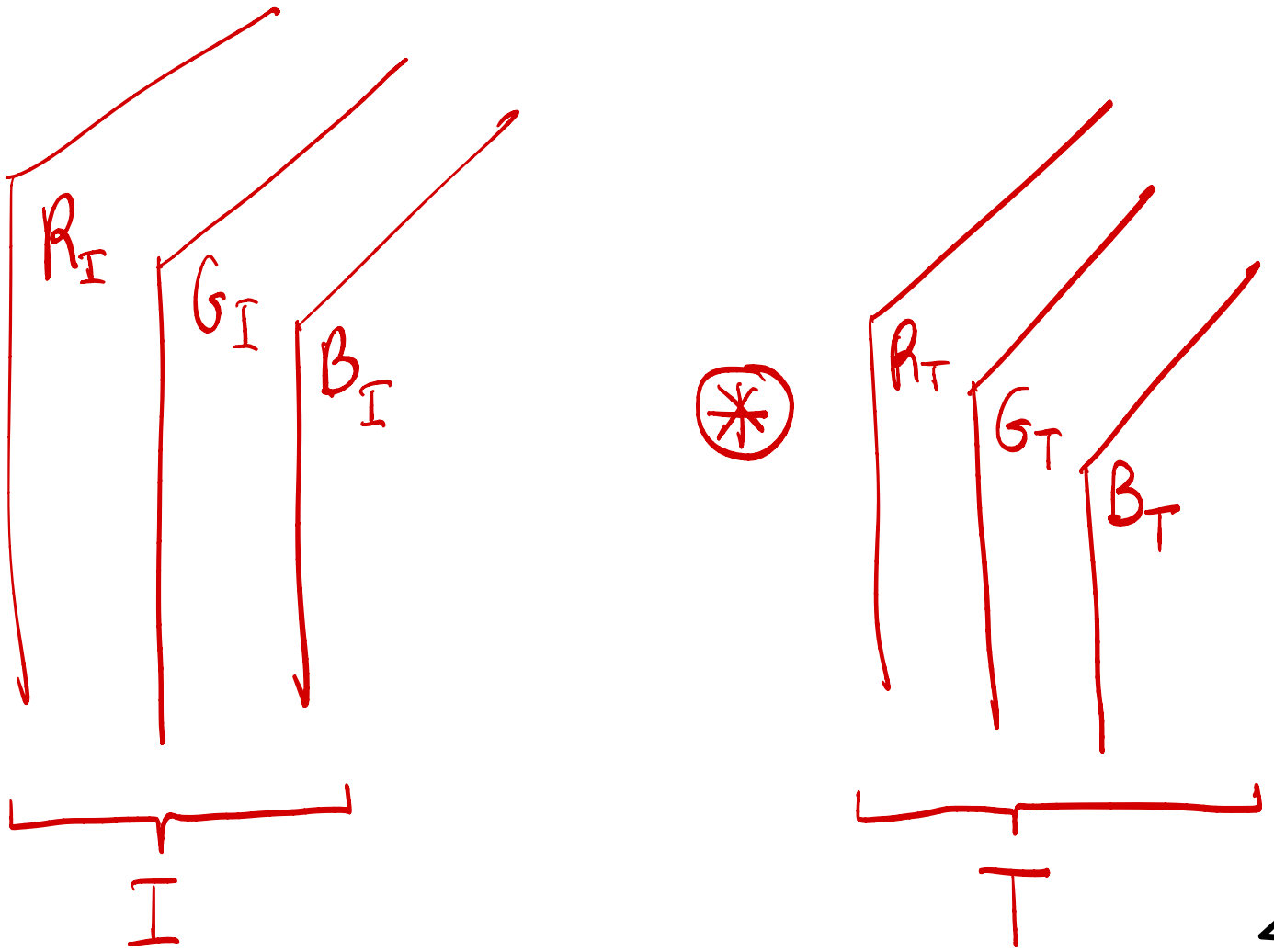
Then, $W[i,:]$ **can be seen as a template used in matching (the output of correlation in the central pixel)**

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

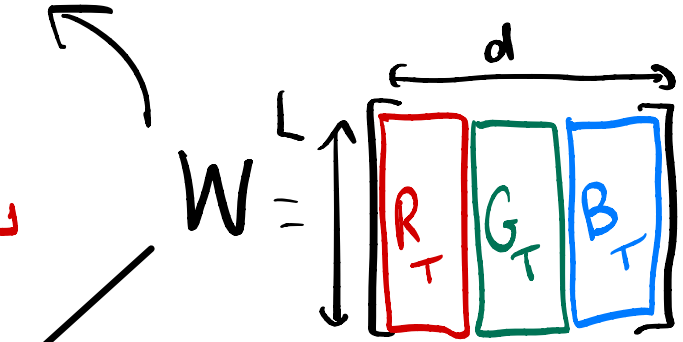
Let's have a look at these templates

Correlation between two RGB images

The image and the filter have the same size



$$R_I \circledast R_T + G_I \circledast G_T + B_I \circledast B_T = \sum_{(x,y)} \left[R_I(x,y) \cdot R_T(x,y) + G_I(x,y) \cdot G_T(x,y) + B_I(x,y) \cdot B_T(x,y) \right]$$



We can interpret is as a RGB img

Bring the classifier weights back to images TEMPLATES

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 (one row)

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

RGB

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

An example of the img "that is learnt".

Templates Learned on the CIFAR-10 dataset

Sum the 3 correlations:

Red channel against red channel,
G. against G,
B. against B,
And you $\sum$.

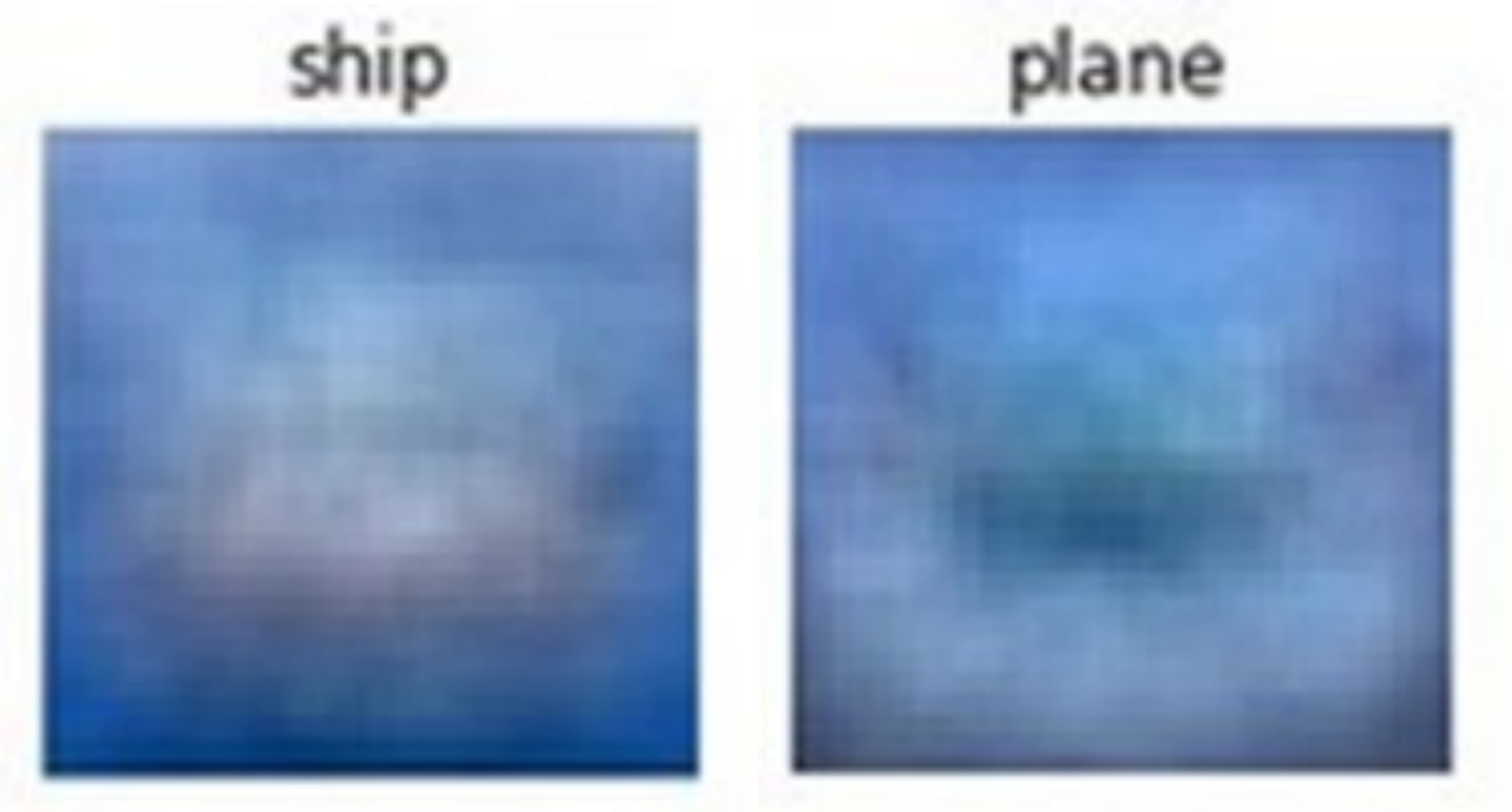
What this NN is doing: it takes an input & computes the correlations against all the 10 templates, and it takes the maximum.

The Class Score

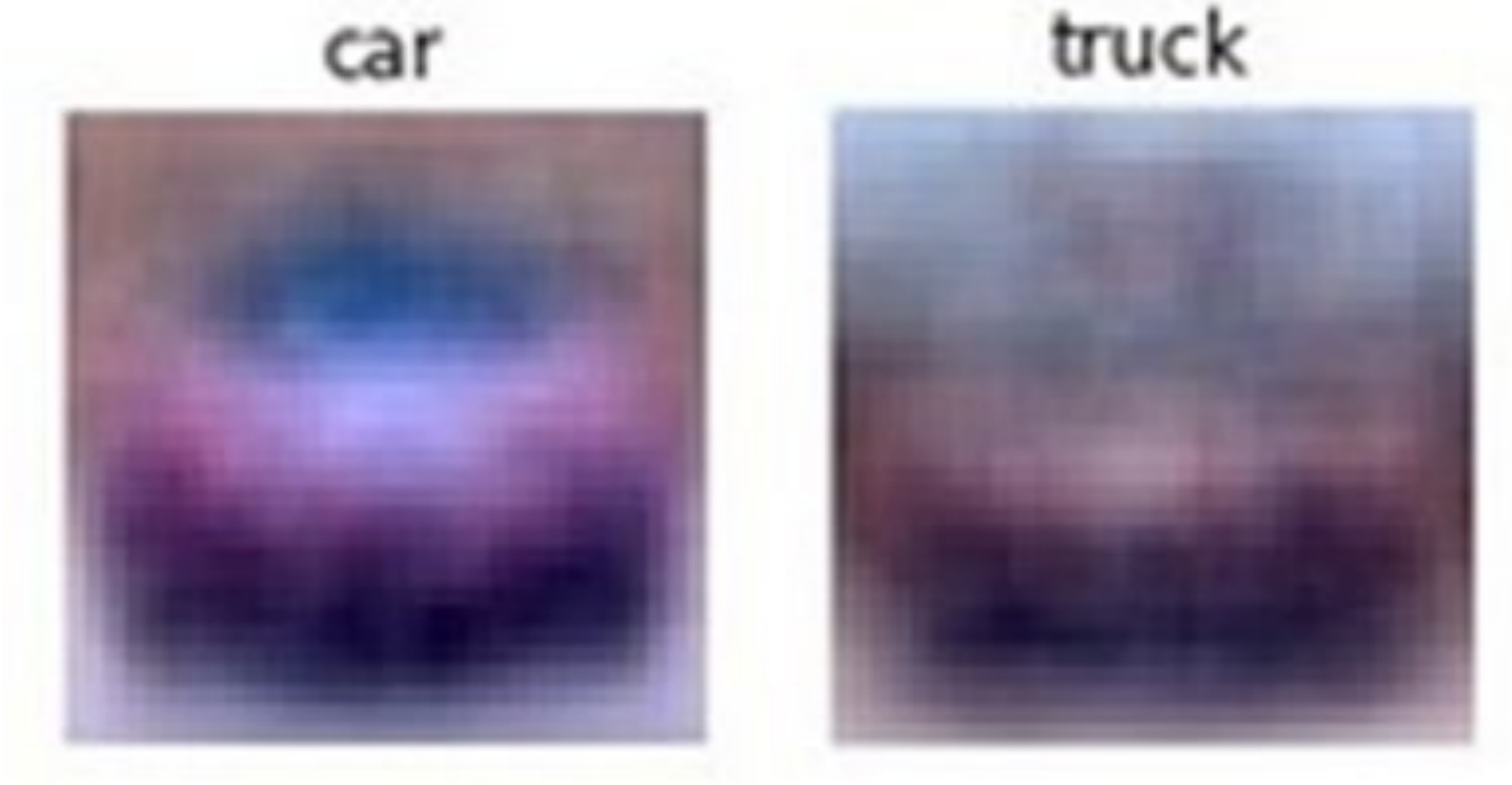
The classification score is then computed as the correlation between each input image and the «template» of the coresponding class

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

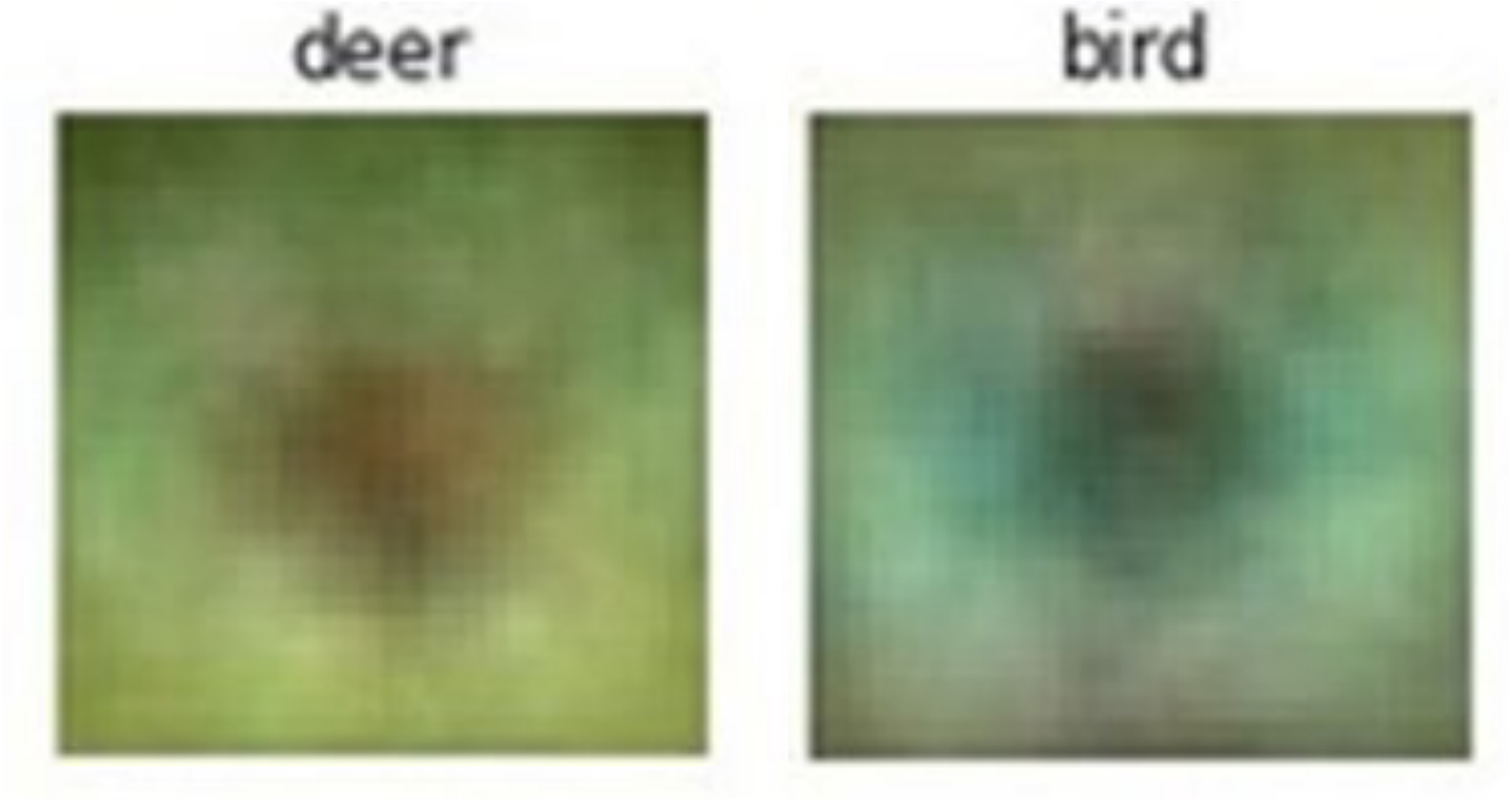
Templates Learned on the CIFAR-10 dataset



Templates Learned on the CIFAR-10 dataset



Templates Learned on the CIFAR-10 dataset



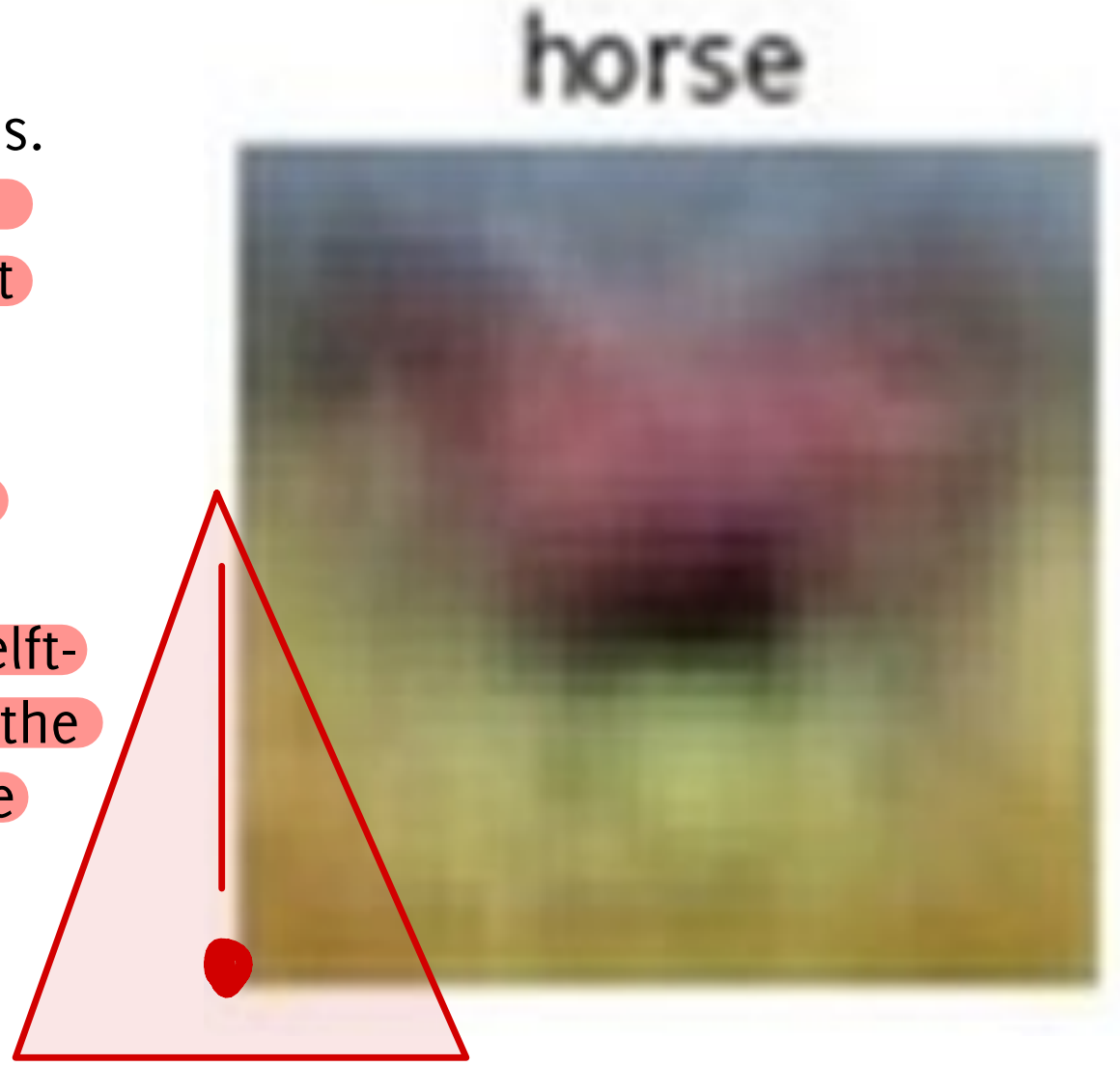
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Templates Learned on the CIFAR-10 dataset

Templates Learned on the CIFAR-10 dataset

- This is because the training set contains images of horses looking in both directions.
- If the horses were all looking at the same side, most probably the template wouldn't be two-headed.

- It is also true the other way round: if you want your network to recognize images being invariant to some transformation (lelft-right split in this case), you need to train the network with images undergoing the same transformation
- This is the principle behind image augmentation



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 green boat is blue)
- Cars are typically red
- Horses have two heads! ☺

The model was definitively too were not enough for achieving higher performance and bett

However:

- Linear Classifiers are a e 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/1kflPH3CDgnvk1JptUoCbp2LK-owoh6R3?usp=sharing

Credits Eugenio Lomurno! (visualization with clipped colors)

Is Image Classification a Challenging Problem?

Yes, it is...

First challenge: dimensionality

Images are very high-dimensional image data

CIFAR-10 dataset

The CIFAR-10 dataset contains 60000 images:

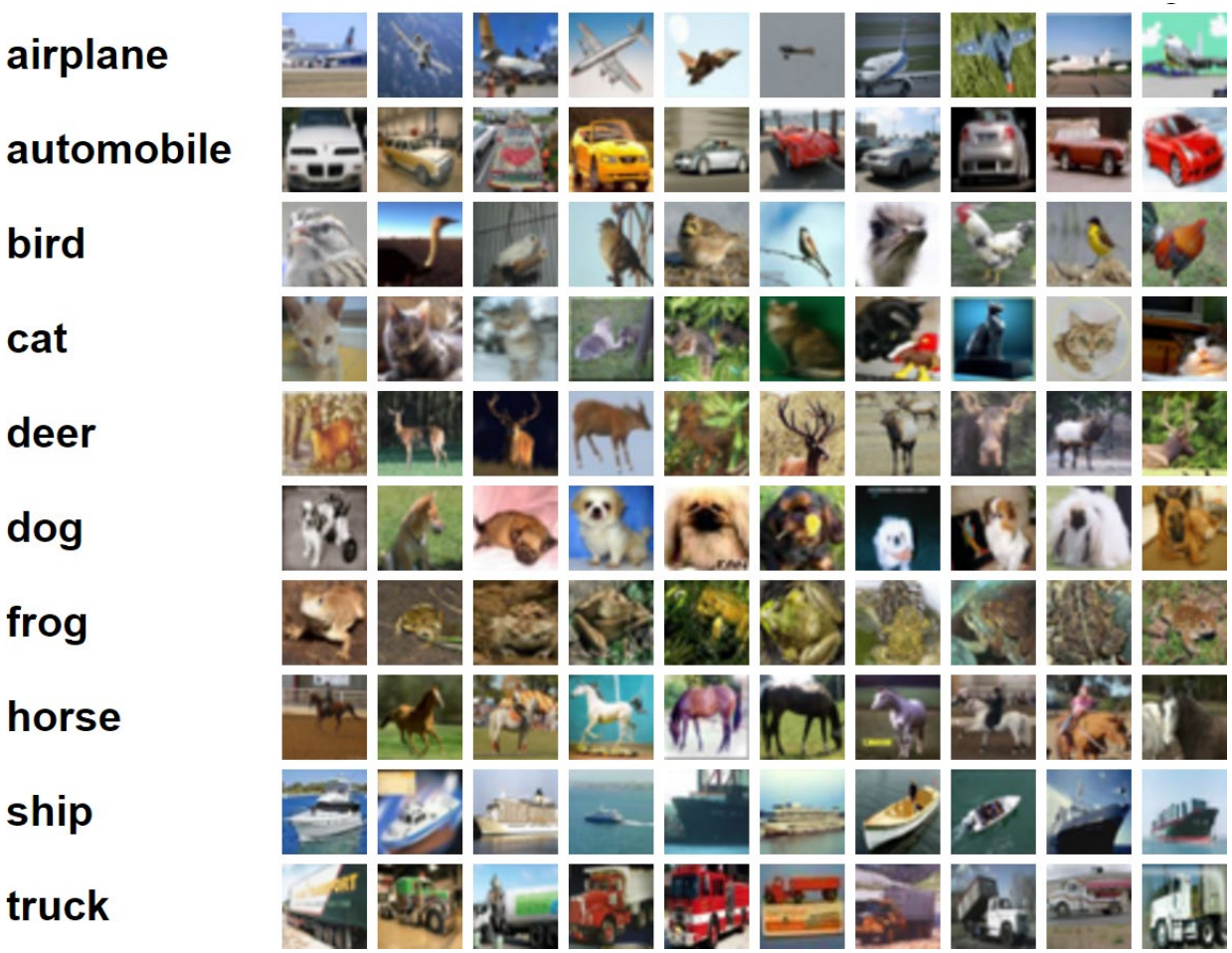
Each image is 32x32 RGB

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

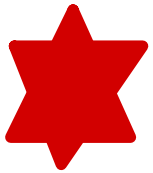
Scale variation

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

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

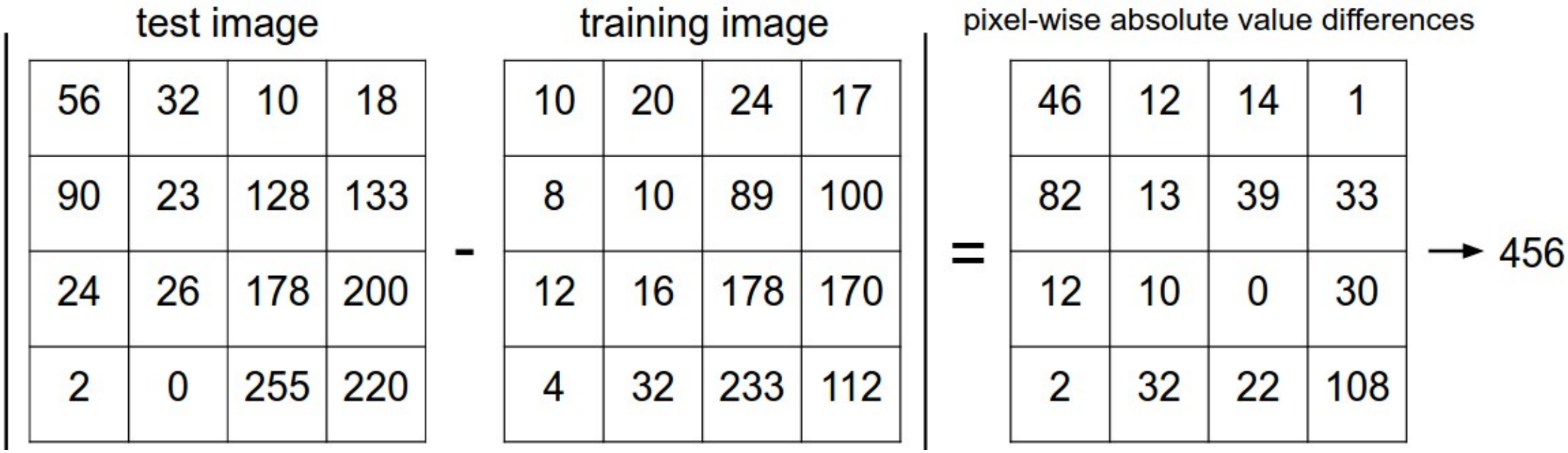
Distances are typically measured as

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

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

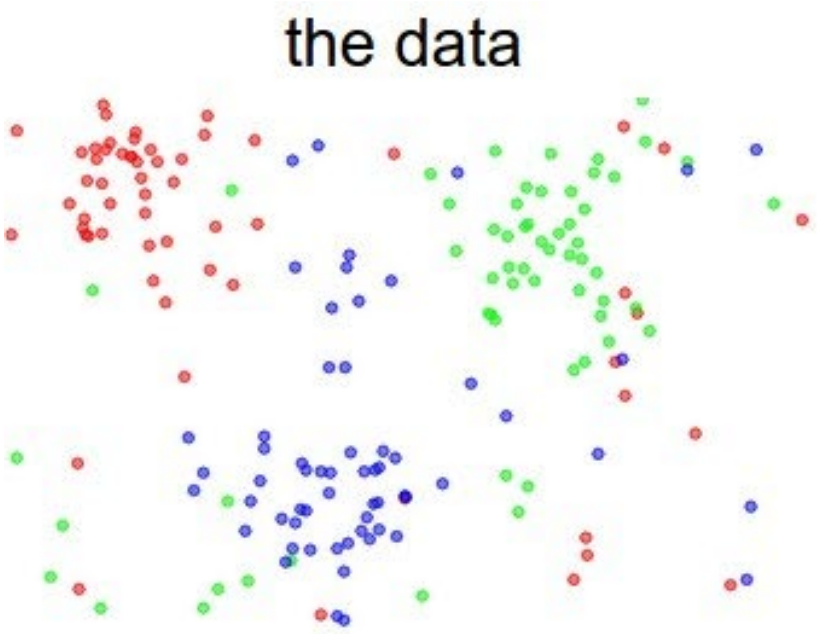
Assign to each test image, **the most frequent label among the K −closest images in the training set**

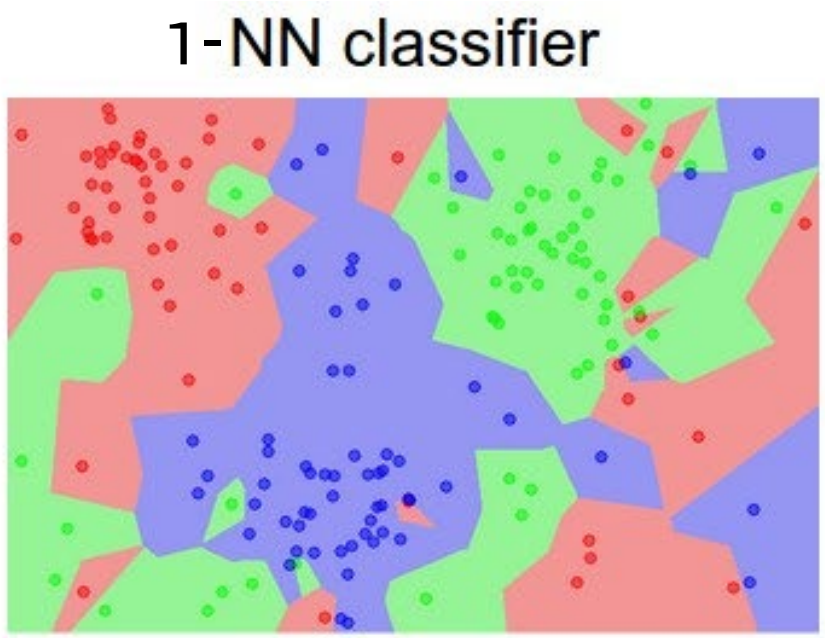
$$\hat{y}_j = y_{j^*}, \qquad \text{being } \; j^* \text{ the mode of } \mathcal{U}_K(\boldsymbol{x}_j)$$

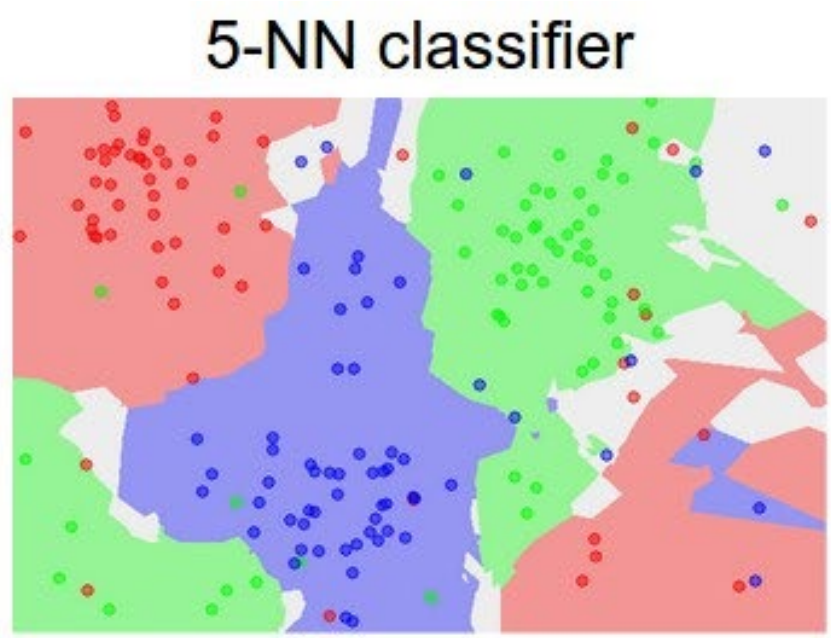
where $\mathcal{U}_K(\boldsymbol{x}_j)$ contains the K closest training images to $\boldsymbol{x}_j$

Setting the parameter K and the distance measure is an issue

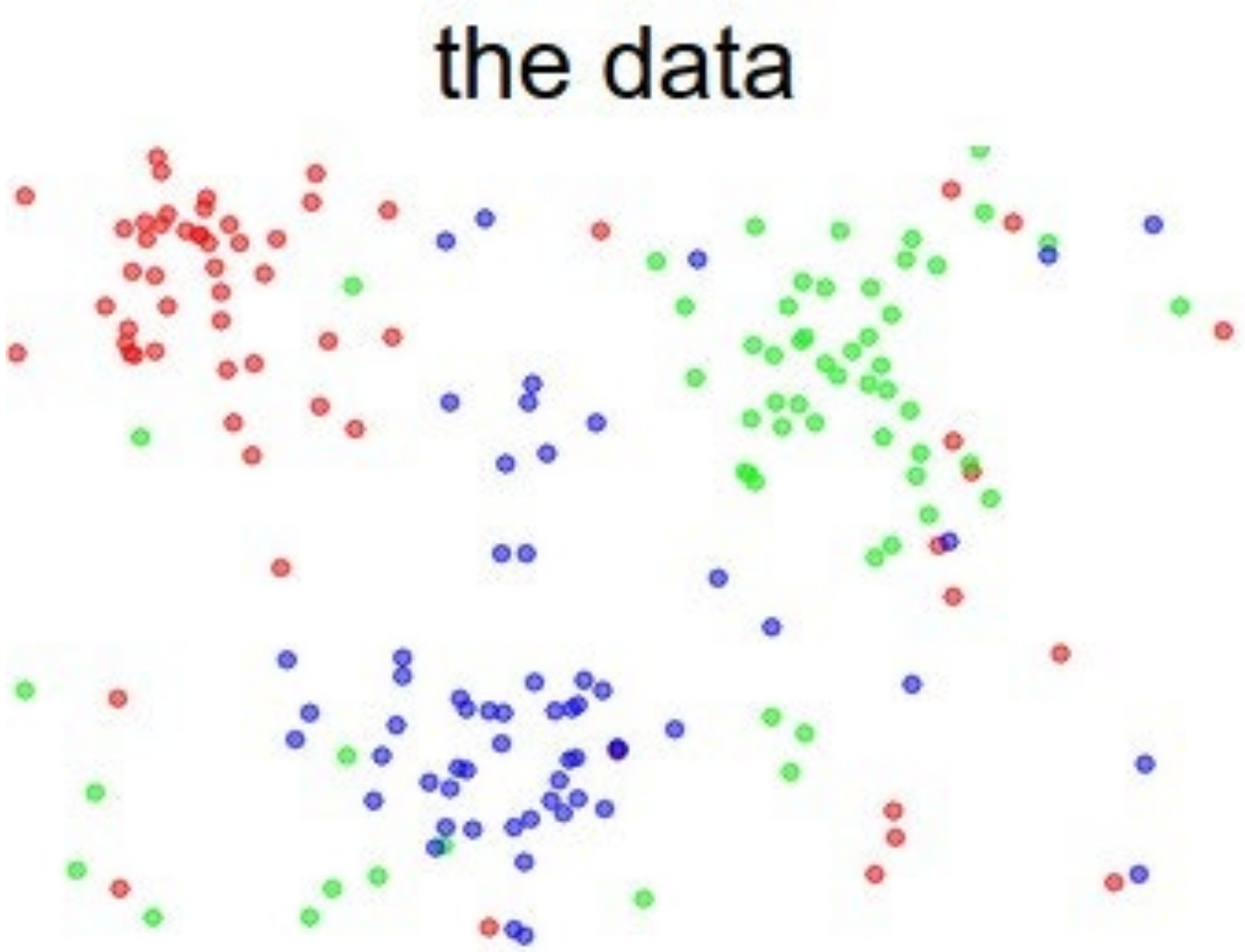
Nearest Neighborhood Classifier (k-NN) for Images





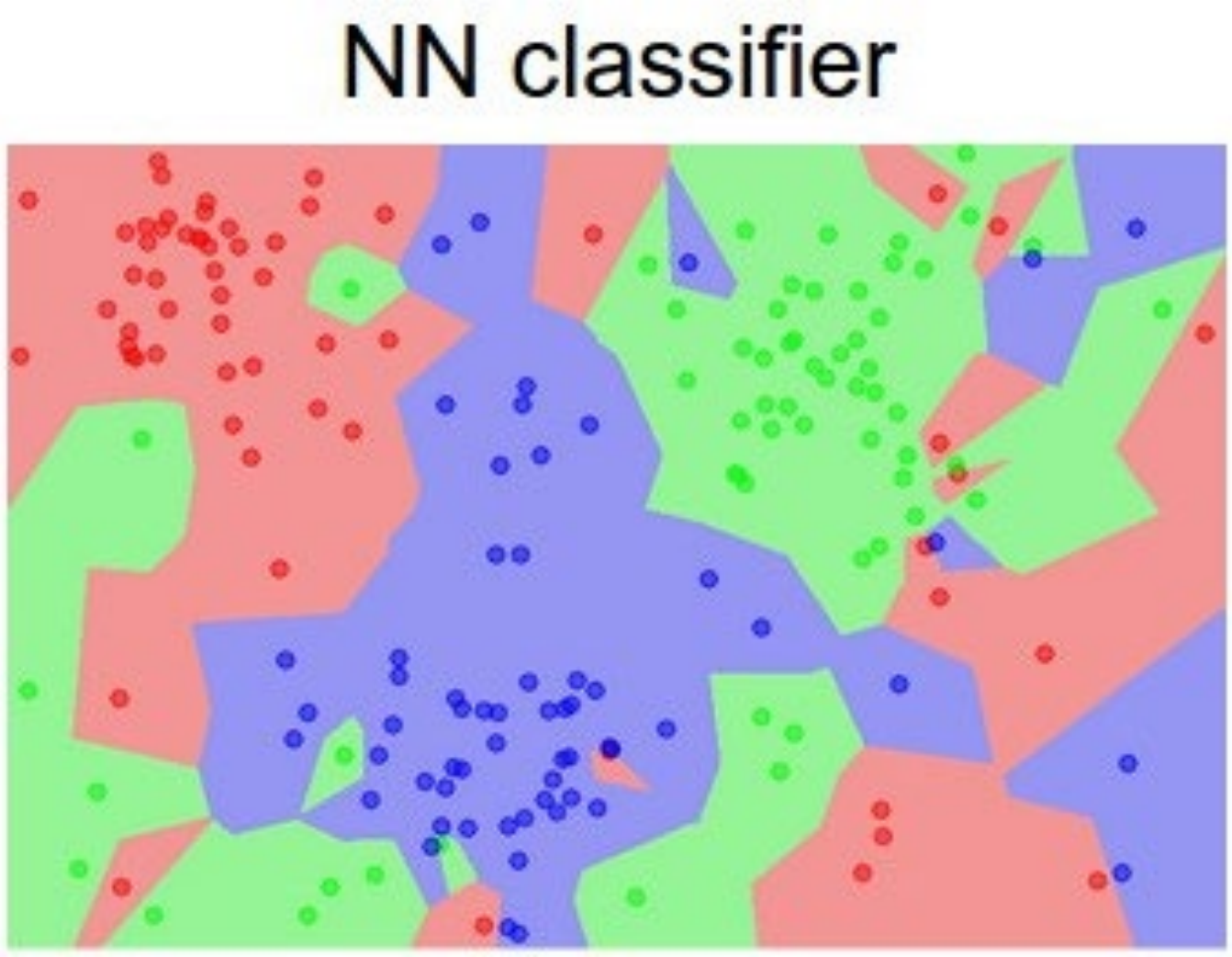


k-NN for Images



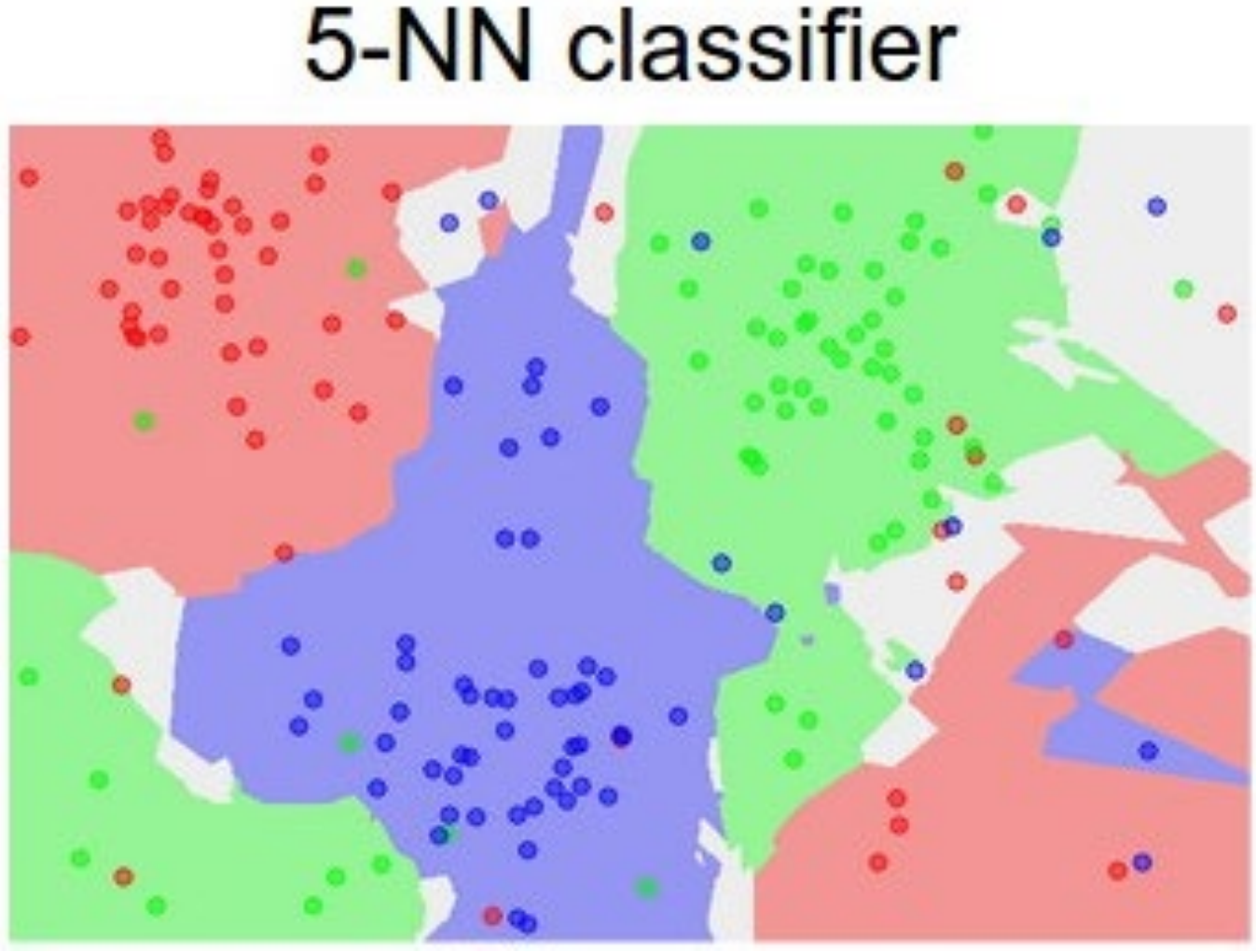
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

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

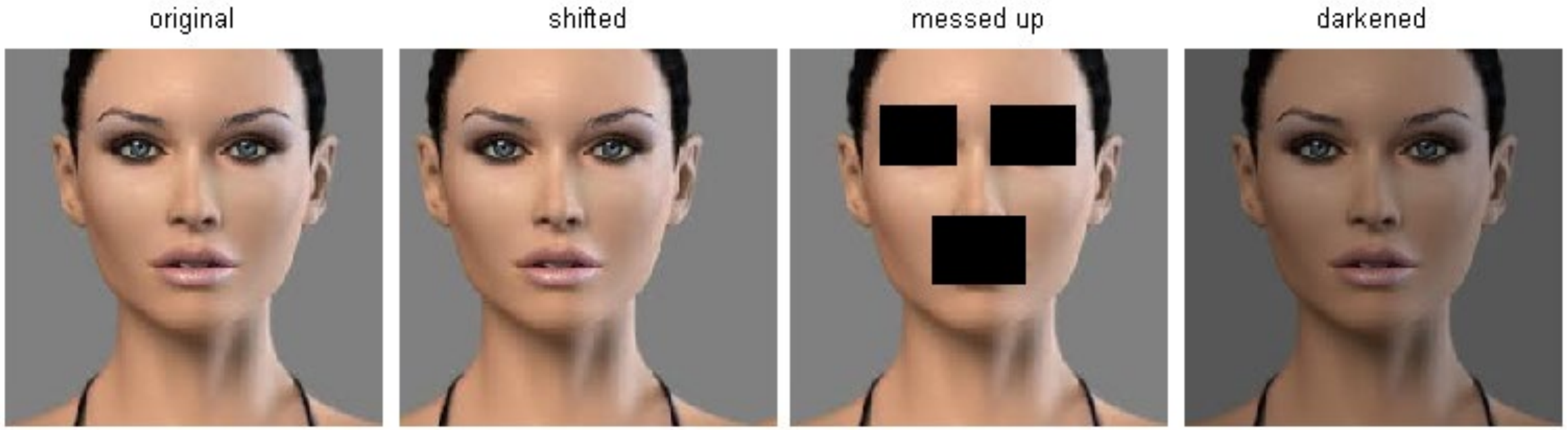
Pros:

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

Cons:

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

Perceptual Similarity vs Pixel Similarity

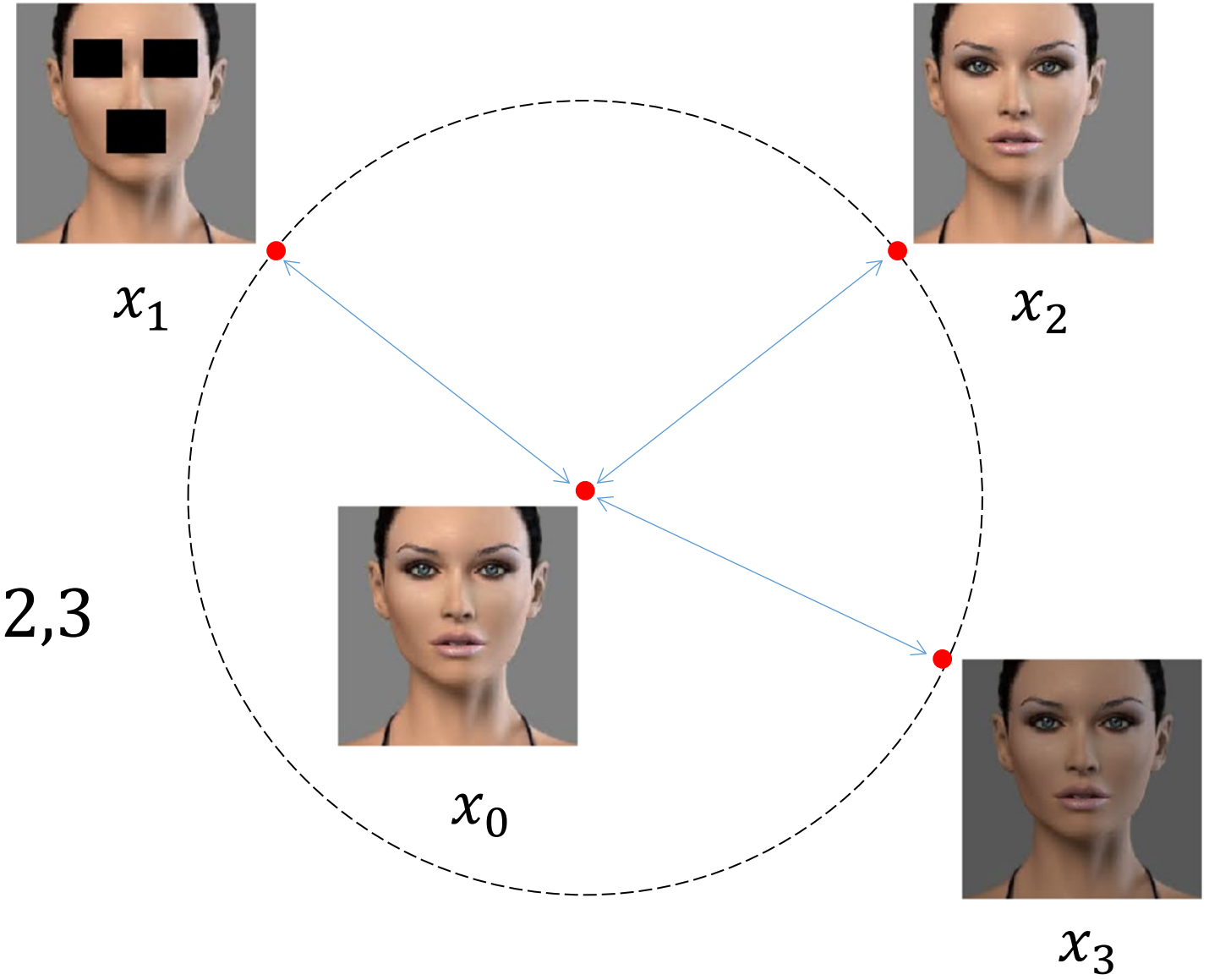


The three images have the same pixel-wise distance from the original one..

...but perceptually they are very different

Perceptual Similarity vs Pixel Similarity

$$\|x_j - x_0\|_2 \approx const\ j = 1,2,3$$



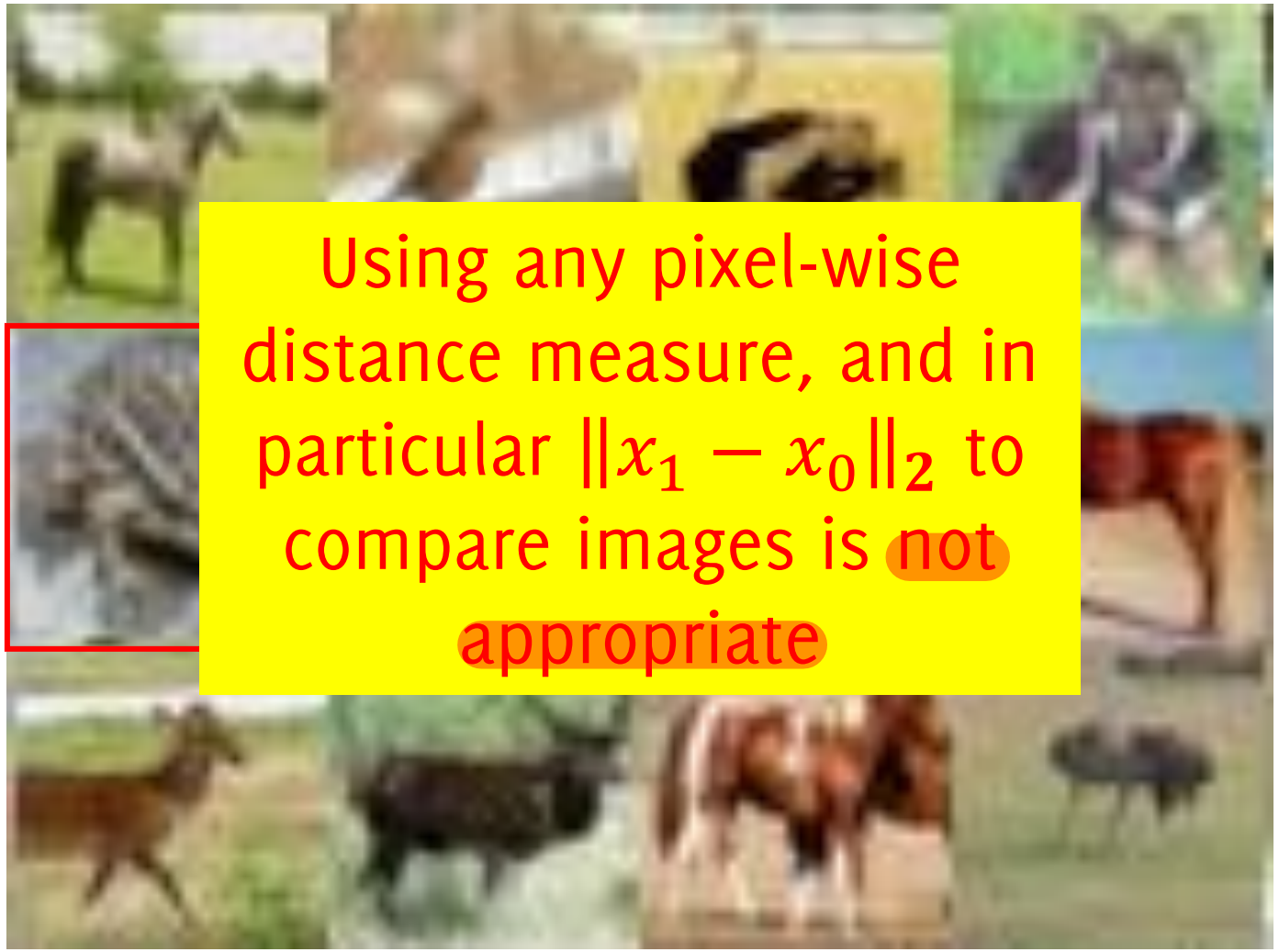
Let's see what happens on the whole CIFAR10 using t-SNE

On CIFAR10 we see exactly this problem

t-SNE: the closer, the smaller the distance.

the frogs are closer to the truck than one to each other.

On CIFAR10 we see exactly this problem



On CIFAR10 we see exactly this problem