Handcrafted Feature Detection and Extraction

Pablo Mesejo

pmesejo@go.ugr.es

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE GRANADA

DaSCI

Instituto Andaluz de Investigación en **Data Science** and **Computational Intelligence**

Readings

- ***Filters as Templates***
  - Forsyth and Ponce (2012), Chapter 4.5 & 4.6
- ***Hough Transform***
  - Forsyth and Ponce (2012), Chapter 10.1
  - Szeliski (2022), Chapter 7.4.2
- ***Texture***
  - Forsyth and Ponce (2012), Chapter 6.1 & 6.2
  - Sonka, Hlavac and Boyle (2015), Chapter 15.1

Template Matching

Main question

How can we find a part of one image that matches another?

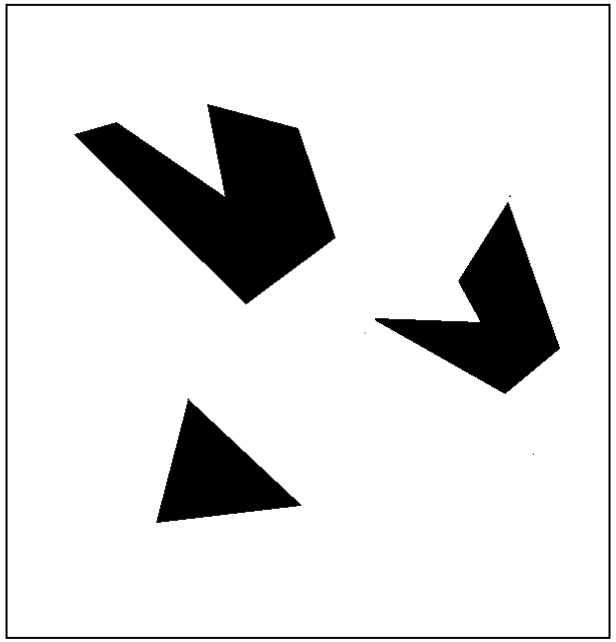
or

How can we find instances of a pattern in an image?

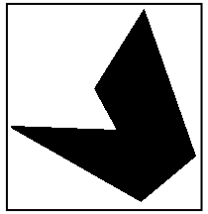
Main question

Key Idea: Use the pattern as a template

Template matching



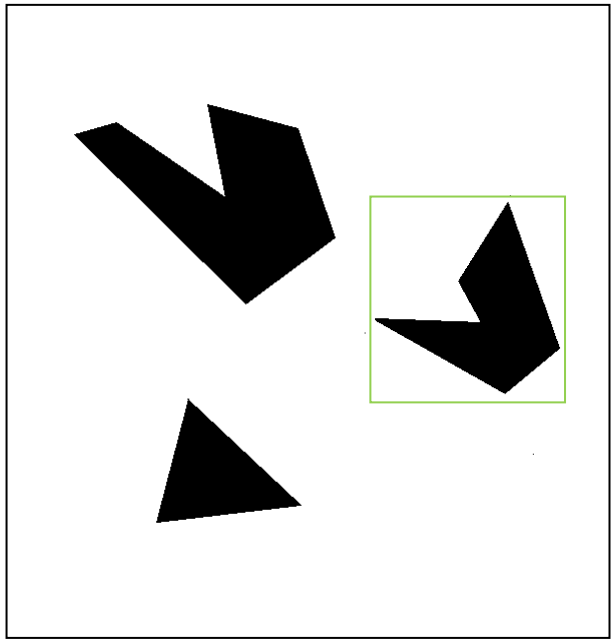
Scene



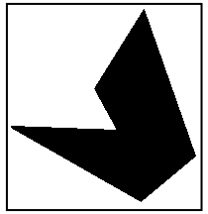
Template (mask)

A toy example

Template matching



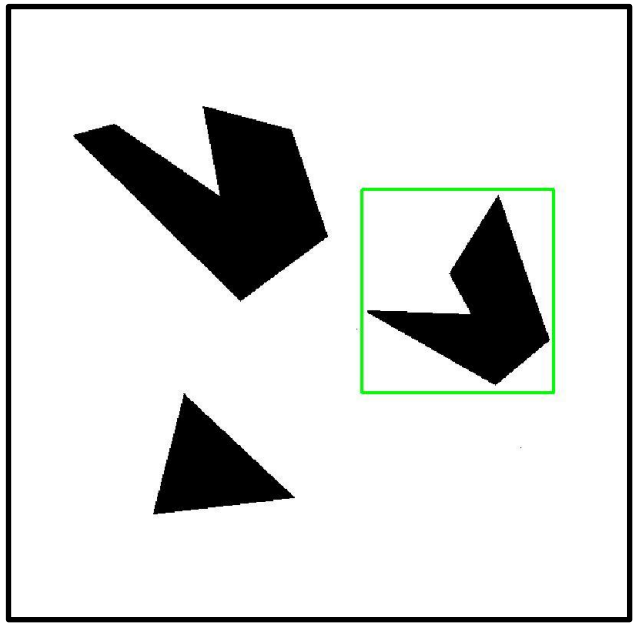
Detected Template



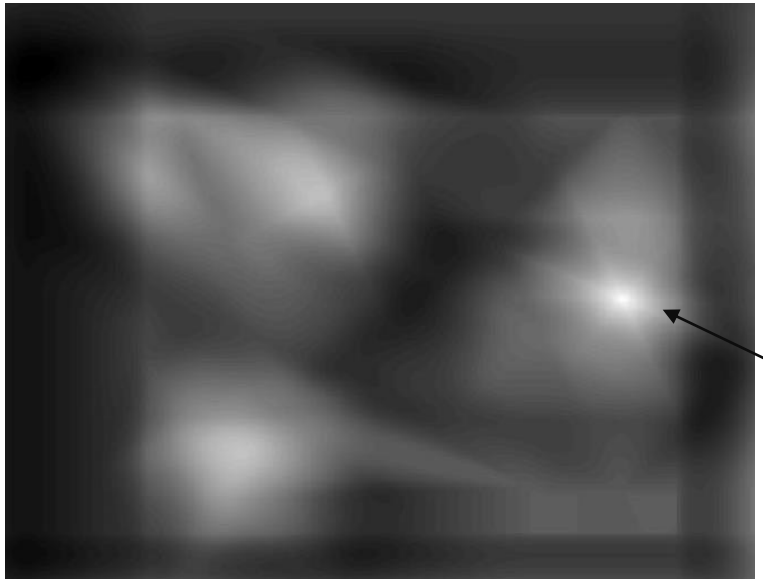
Template (mask)

A toy example

Template matching



Detected template

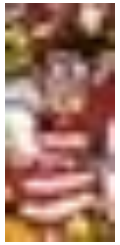


Correlation map

Brightest area is region of best match. Note: if we use a “difference map”, the darkest area would be the region of best match.

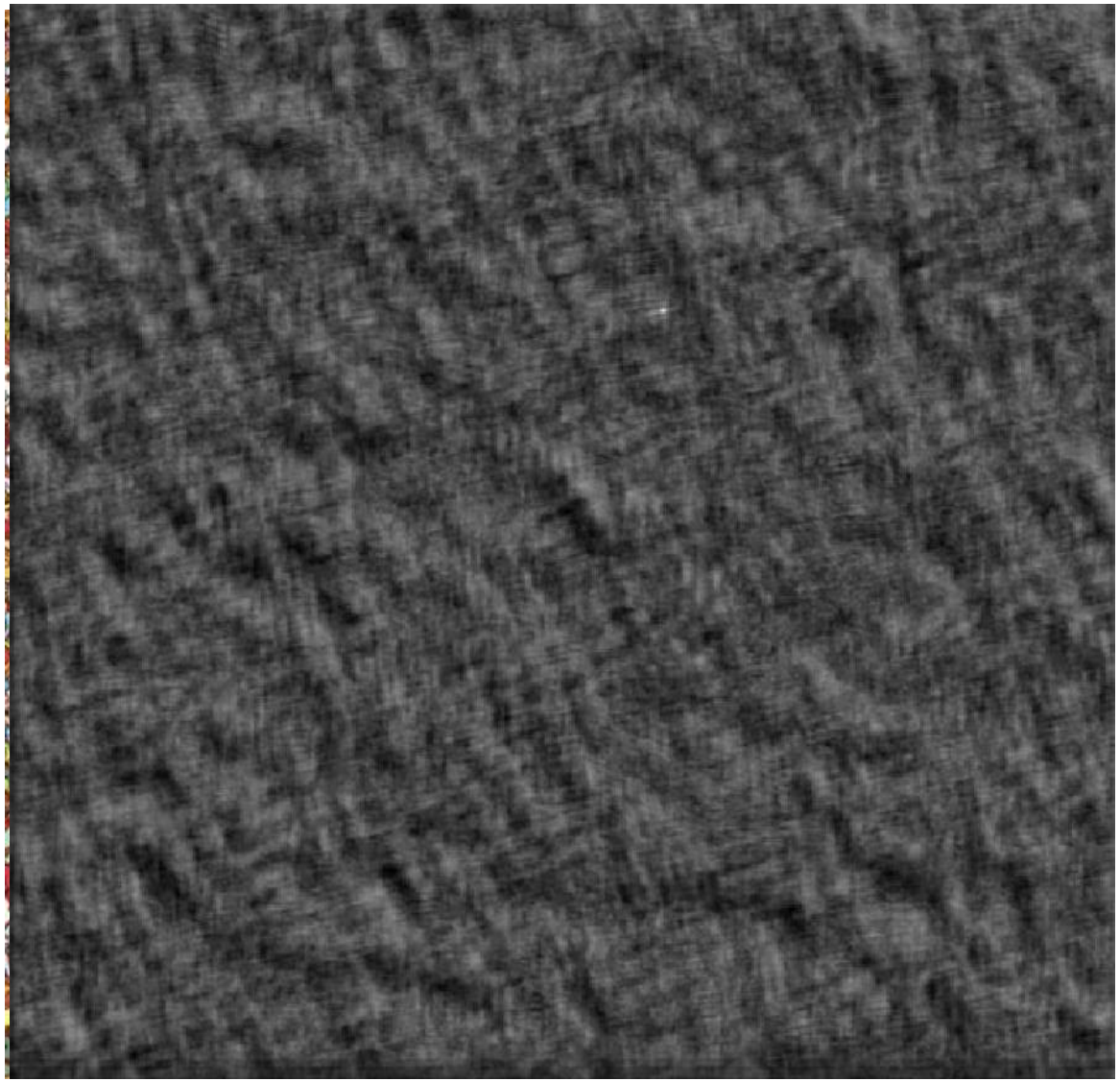
Where’s Waldo?

Scene

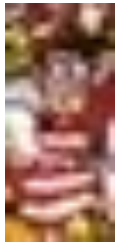


Template

Where’s Waldo?



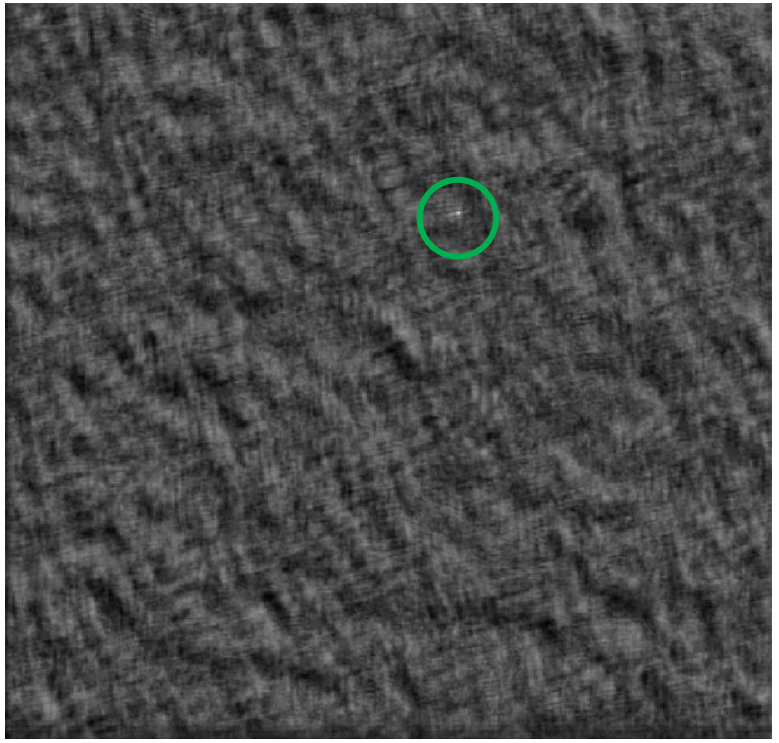
Scene



Template

Where’s Waldo?

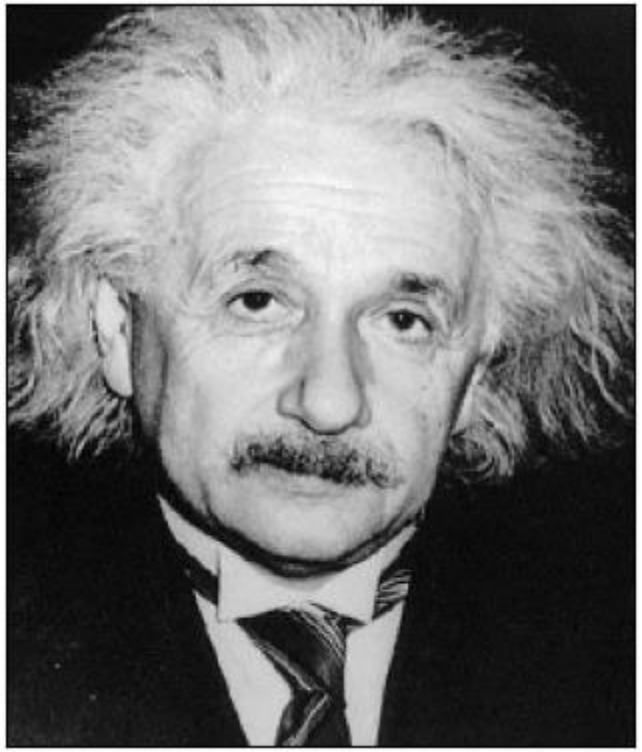
Detected template



Correlation map

Template Matching

Goal: Find the template  in the image



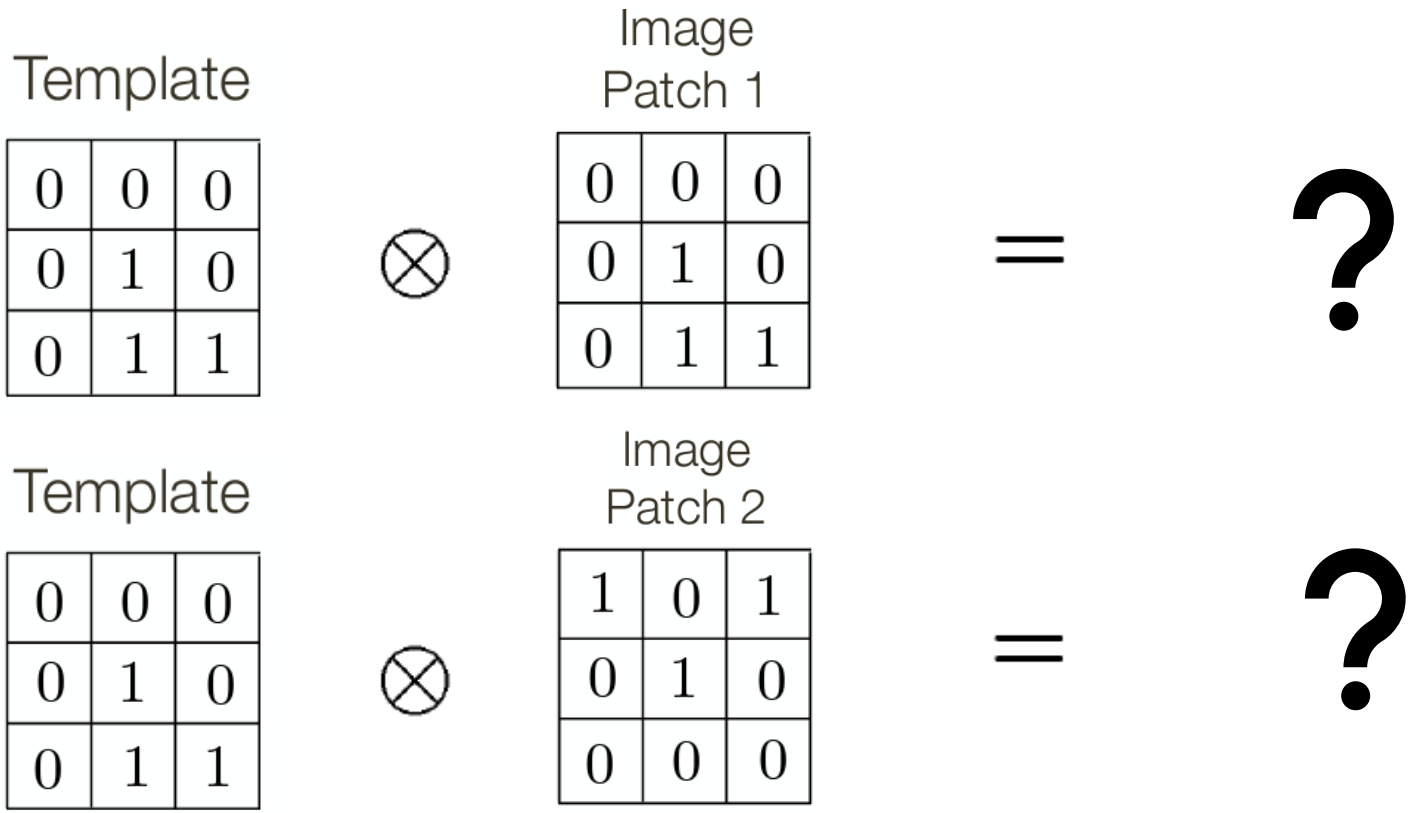
Main challenge:

What is a good similarity metric between two image patches?

As usual, no single and universal answer!

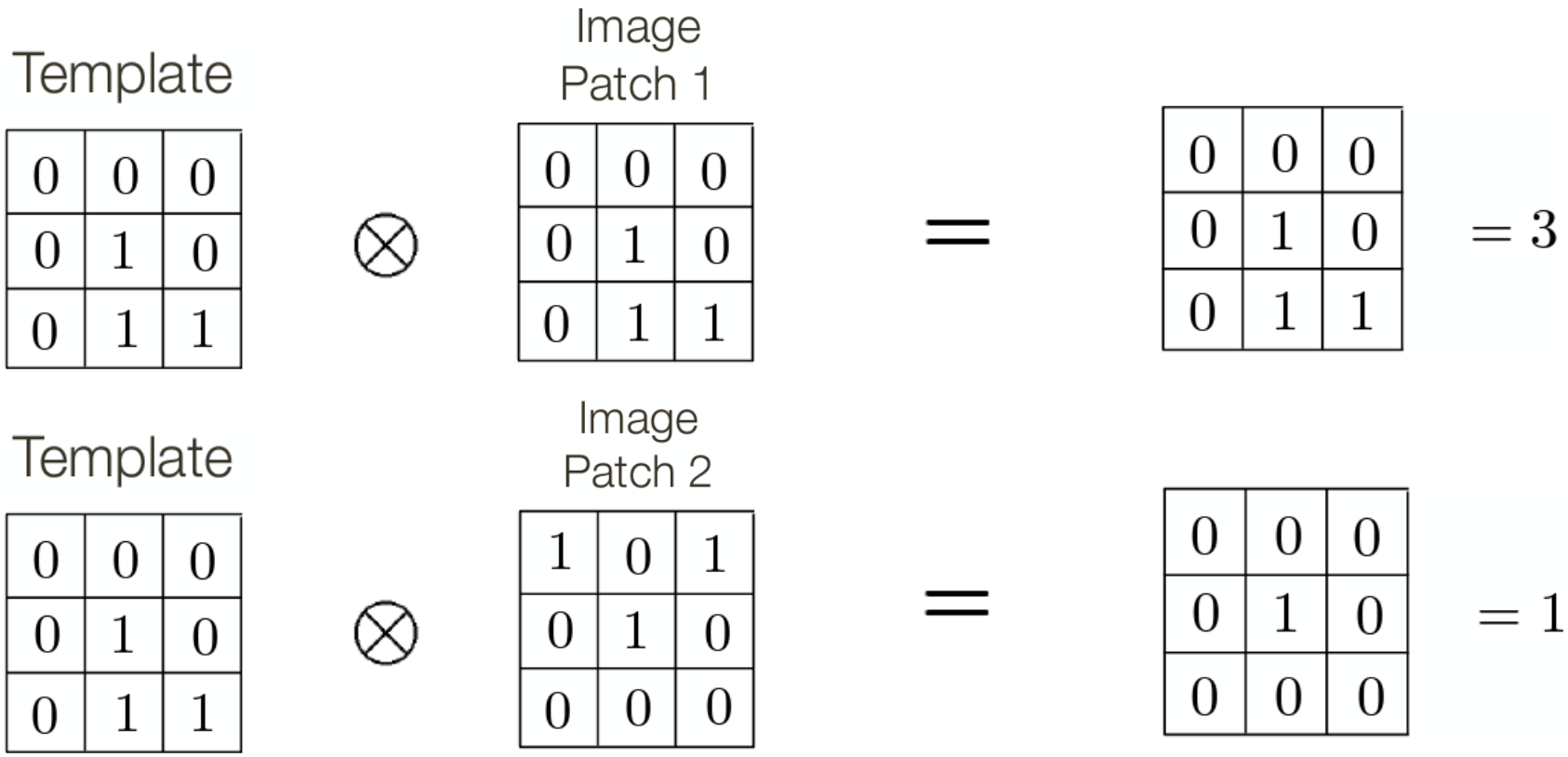
Template Matching

- **Solution 1**: “Cross-correlation measures the **similarity** between two signals/sequences”. We want to maximize cross-correlation.



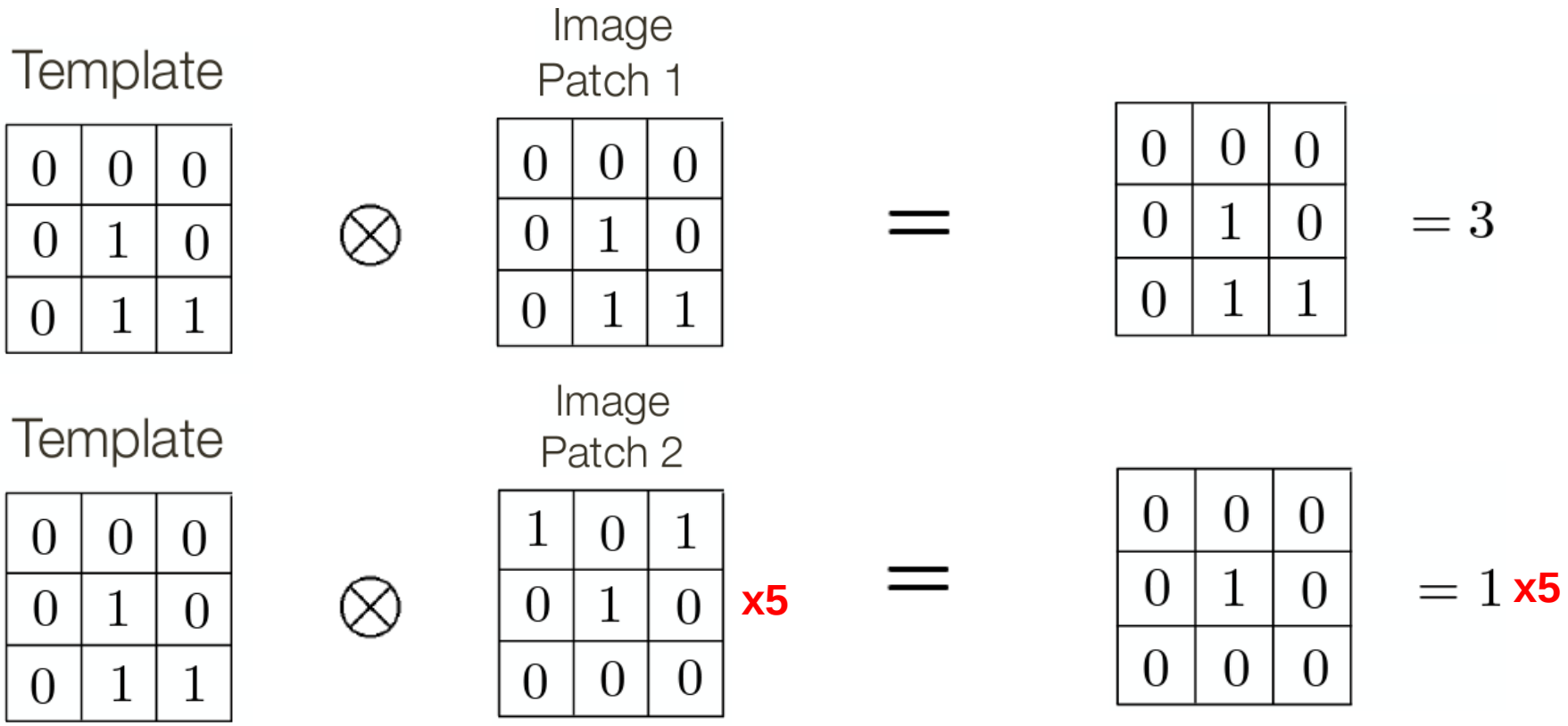
Template Matching

- **Solution 1**: “Cross-correlation measures the **similarity** between two signals/sequences”. We want to maximize cross-correlation.



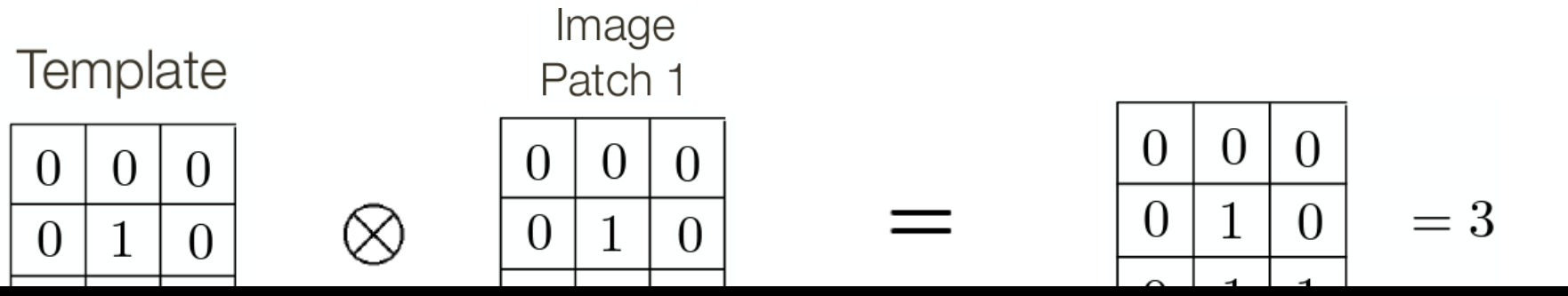
Template Matching

- So far so good, but… what if the image is brighter?

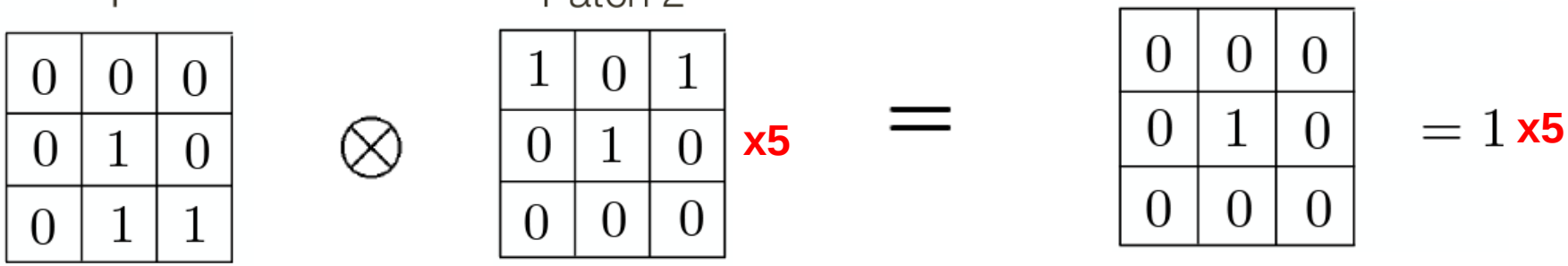


Template Matching

- So far so good, but… what if the image is brighter?



The dot product may be larger simply because the image region is brighter. We need to normalize the result in some way.

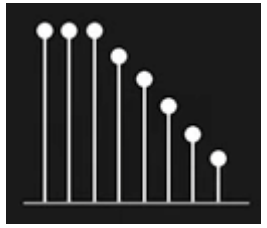


Slide: Leonid Sigal

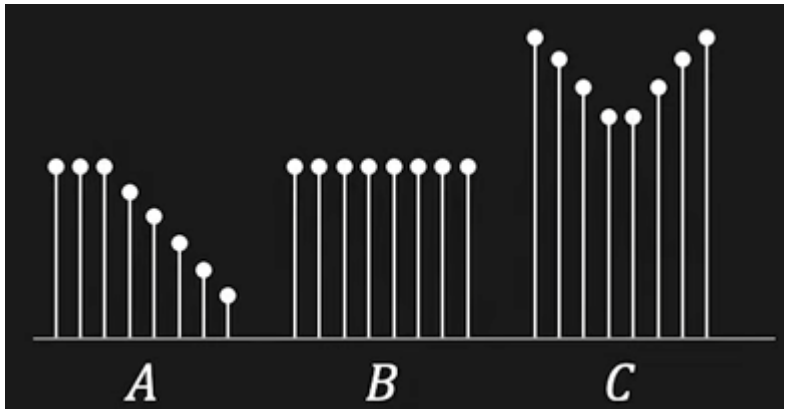
Template Matching

- Let’s see a 1D example

Template



Different signals



A should be the perfect match (so, ideally, we should get maximum correlation)

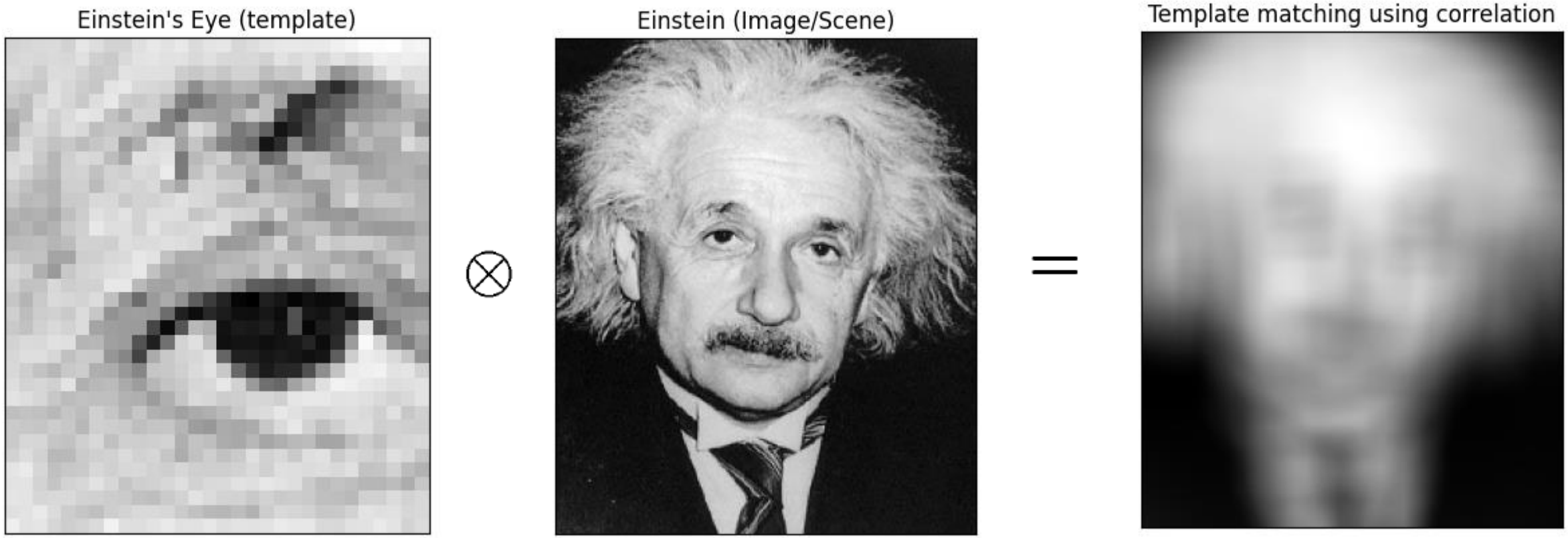
Where (A, B or C) correlation is going to be the highest??

🤔

Slide: Shree K. Nayar

Template Matching

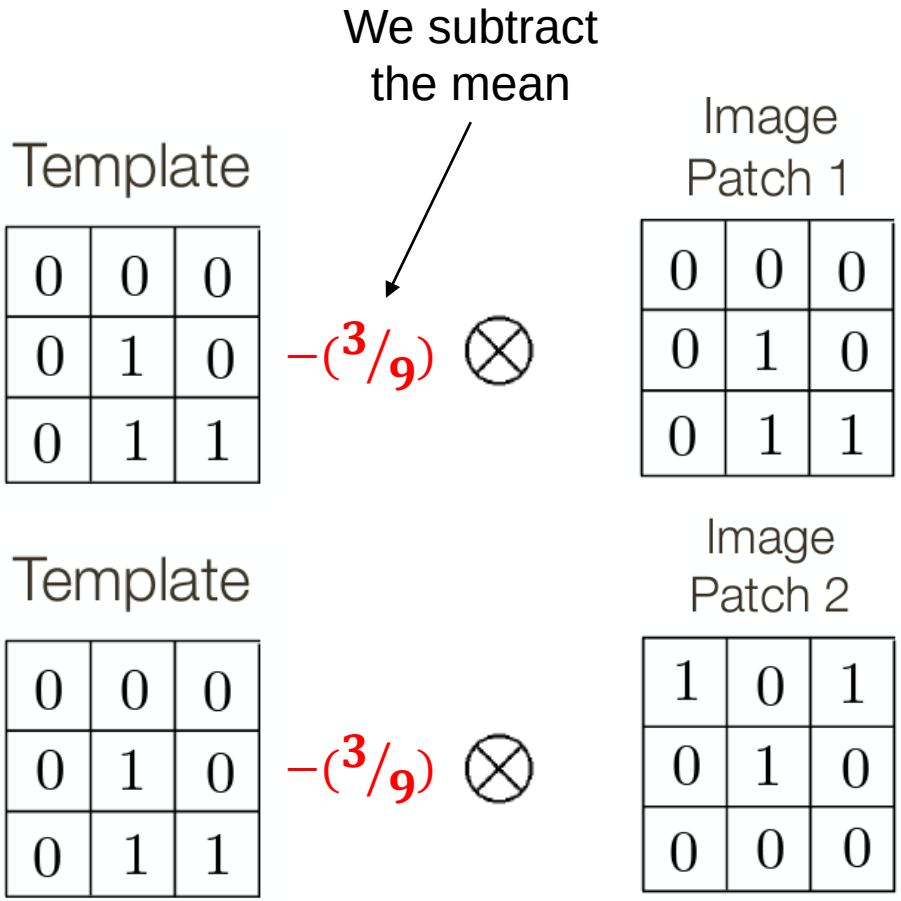
- Let’s see a 2D example



It doesn’t seem very useful for finding patterns… 😔

Template Matching

- **Solution 2**: filter the image with the template (eye patch) with zero mean (i.e. zero mean normalization of the template)



Template Matching

- **Solution 2**: filter the image with the template (eye patch) with zero mean (i.e. zero mean normalization of the template)

Template

-1/3	-1/3	-1/3
-1/3	2/3	-1/3
-1/3	2/3	2/3

$\otimes$

Image Patch 1

0	0	0
0	1	0
0	1	1

Template

-1/3	-1/3	-1/3
-1/3	2/3	-1/3
-1/3	2/3	2/3

$\otimes$

Image Patch 2

1	0	1
0	1	0
0	0	0

Template Matching

- **Solution 2**: filter the image with the template (eye patch) with zero mean (i.e. zero mean normalization of the template)

Template

-1/3	-1/3	-1/3
-1/3	2/3	-1/3
-1/3	2/3	2/3

$\otimes$

Image Patch 1

0	0	0
0	1	0
0	1	1

$=$

0	0	0
0	2/3	0
0	2/3	2/3

$=$ **2**

Template

-1/3	-1/3	-1/3
-1/3	2/3	-1/3
-1/3	2/3	2/3

$\otimes$

Image Patch 2

1	0	1
0	1	0
0	0	0

$=$

-1/3	0	-1/3
0	2/3	0
0	0	0

$=$ **0**

Template Matching

- **Solution 2**: filter the image with the template (eye patch) with zero mean (i.e. zero mean normalization of the template)

Template

-1/3	-1/3	-1/3
-1/3	2/3	-1/3
-1/3	2/3	2/3

$\otimes$

Image Patch 1

0	0	0
0	1	0
0	1	1

$=$

0	0	0
0	2/3	0
0	2/3	2/3

$=$ **2**

Template

-1/3	-1/3	-1/3
-1/3	2/3	-1/3
-1/3	2/3	2/3

$\otimes$

Image Patch 2

1	0	1
0	1	0
0	0	0

x5 $=$

-5/3	0	-5/3
0	10/3	0
0	0	0

$=$ **0**

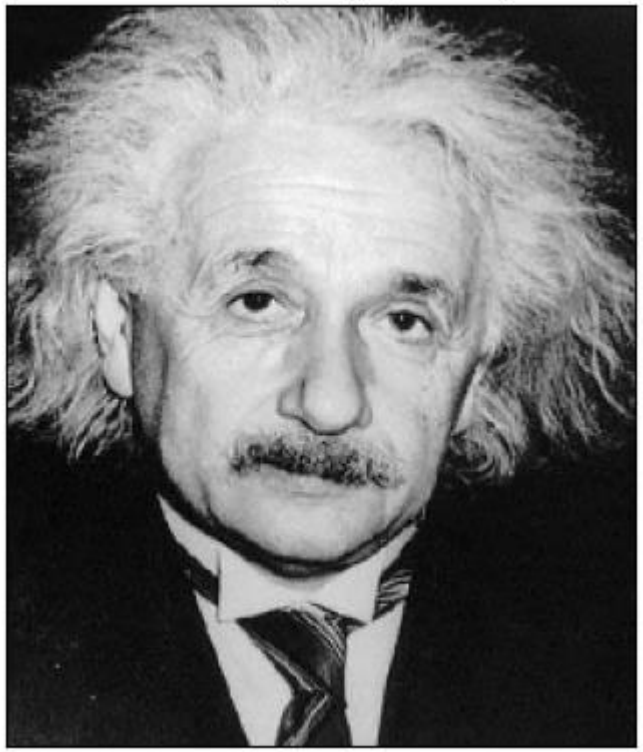
Template Matching

- Let’s see a 2D example

Zero-mean eye (template)

⊗

Einstein (Image/Scene)



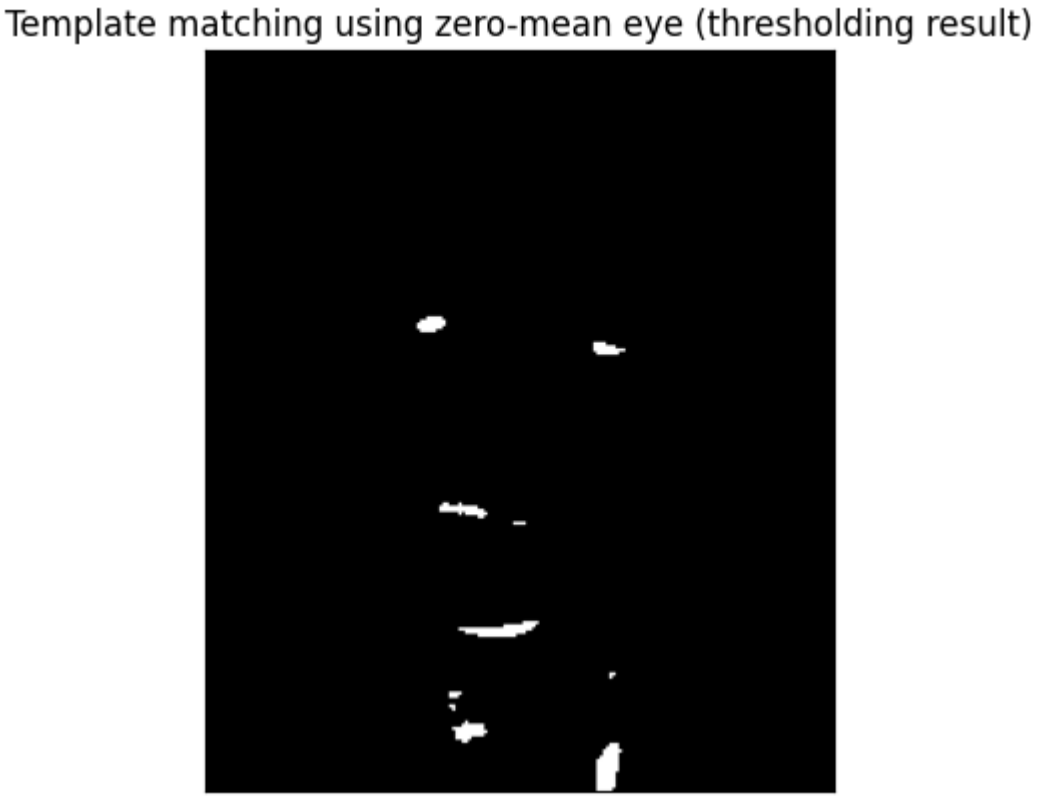
=

Template matching using zero-mean eye

Hey!! This looks different! 😎 **But… <u>it seems quite sensitive to high contrast areas</u>**

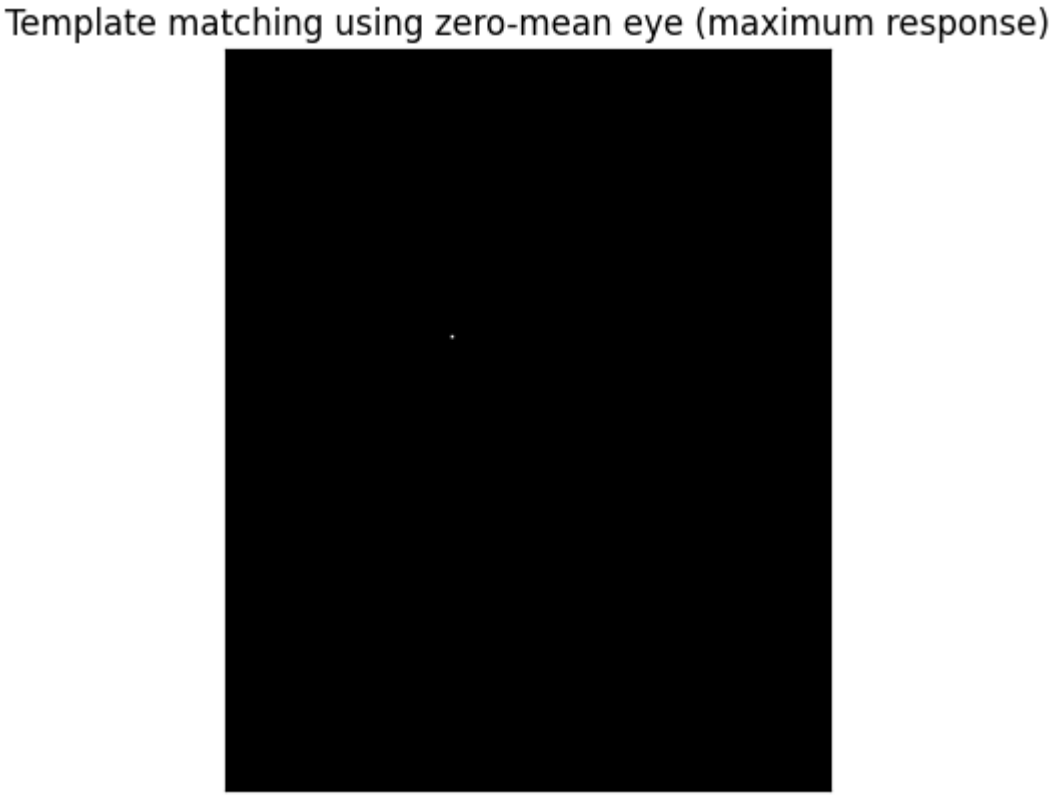
Template Matching

- Depending on the threshold we use, we can have many false detections (note: this is not exclusive of this posible solution)



Template Matching

- Depending on the threshold we use, we can have many false detections (note: this is not exclusive of this posible solution)



Template Matching

- **Solution 3**: Minimize sum of squared differences (SSD)

$$\sum \left(\begin{array}{c} \text{Template} \\ \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 1 & 1 \\ \hline \end{array} \end{array} - \begin{array}{c} \text{Image} \\ \text{Patch 1} \\ \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 1 & 1 \\ \hline \end{array} \end{array} \right)^2 = \sum \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} = \mathbf{0}$$

$$\sum \left(\begin{array}{c} \text{Template} \\ \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 1 & 1 \\ \hline \end{array} \end{array} - \begin{array}{c} \text{Image} \\ \text{Patch 2} \\ \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \end{array} \right)^2 = \sum \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline 0 & 0 & 0 \\ \hline 0 & 1 & 1 \\ \hline \end{array} = \mathbf{4}$$

We slide the template over the image/scene and, at each point, we can **find the difference between the template and the image in the overlapping region**.

Template Matching

- **Solution 3**: Minimize sum of squared differences (SSD)

$$\sum \left(\text{Template} - \text{Image Patch 1} \right)^2 = \sum \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = 0$$

Template:
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$

Image Patch 1:
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$

$$\sum \left(\text{Template} - \text{Image Patch 2} \right)^2 = \sum \begin{bmatrix} 25 & 0 & 25 \\ 0 & 16 & 0 \\ 0 & 1 & 1 \end{bmatrix} = 68$$

Template:
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$

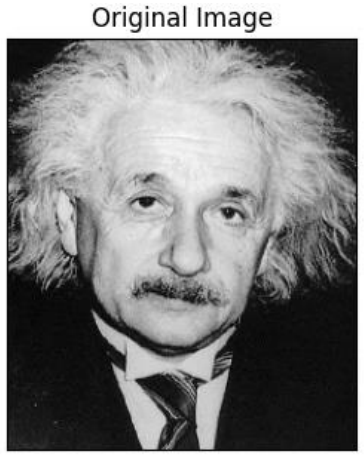
Image Patch 2:
$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ x5

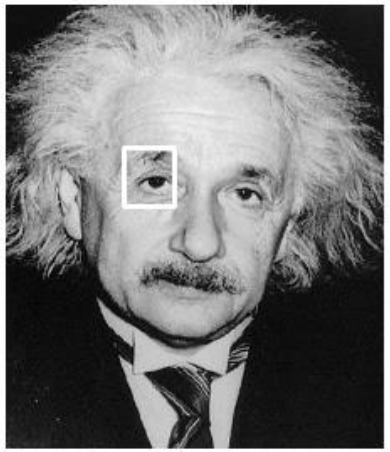
We slide the template over the image/scene and, at each point, we can **find the difference between the template and the image in the overlapping region**.

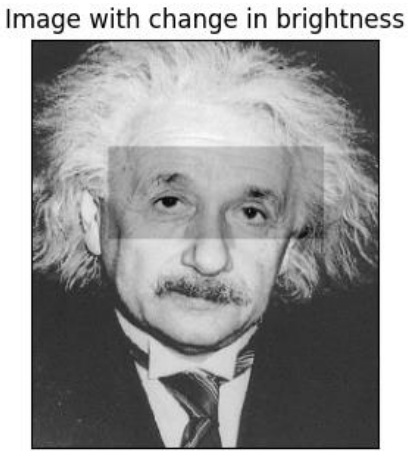
Template Matching

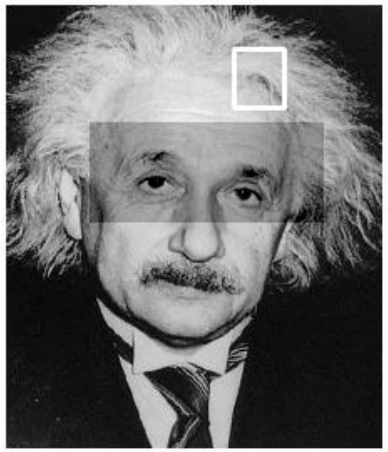
- **Solution 3**: Minimize sum of squared differences (SSD)

Problem: not robust to changes in local brightness!









Note1: we display 1-SSD
Note2: in the right column we locate the bounding box centered in the coordinates of minimum SSD

Template Matching

- $SSD[i,j] = \underbrace{\sum_{u=-k}^{k}\sum_{v=-k}^{k}(H[u,v]-F[i+u,j+v])^2}_{\text{Sum of squared differences (SSD) between image } F \text{ and template } H} =$

$$= \sum_{u=-k}^{k}\sum_{v=-k}^{k}(H[u,v]^2 + F[i+u,j+v]^2 - \underbrace{2H[u,v]F[i+u,j+v]}_{\text{The minimization of SSD implies the maximization of this last term (which is the cross-correlation between H and F)}})$$

Reminder about cross-correlation:

$$\sum_{u=-k}^{k}\sum_{v=-k}^{k} H[u,v]F[i+u,j+v] = H \otimes F$$

Template Image

Note: there are normalized versions of SSD. See Szeliski (2022), Ch. 9.1

Template Matching

- **Solution 4**: Normalized Cross-Correlation (NCC)

Template mean

Image patch mean

$$\mathrm{NCC}[i,j] = \frac{\sum_{u=-k}^{k}\sum_{v=-k}^{k}(H[u,v]-\bar{H})(F[i+u,j+v]-\bar{F})}{\sqrt{\underbrace{\sum_{u=-k}^{k}\sum_{v=-k}^{k}(H[u,v]-\bar{H})^2}_{\text{Energy of the (normalized) template}}\underbrace{\sum_{u=-k}^{k}\sum_{v=-k}^{k}(F[i+u,j+v]-\bar{F})^2}_{\text{Energy of the (normalized) image patch}}}}$$

By doing this, we make cross-correlation insensitive to changes in brightness!

NCC varies between -1 and 1:

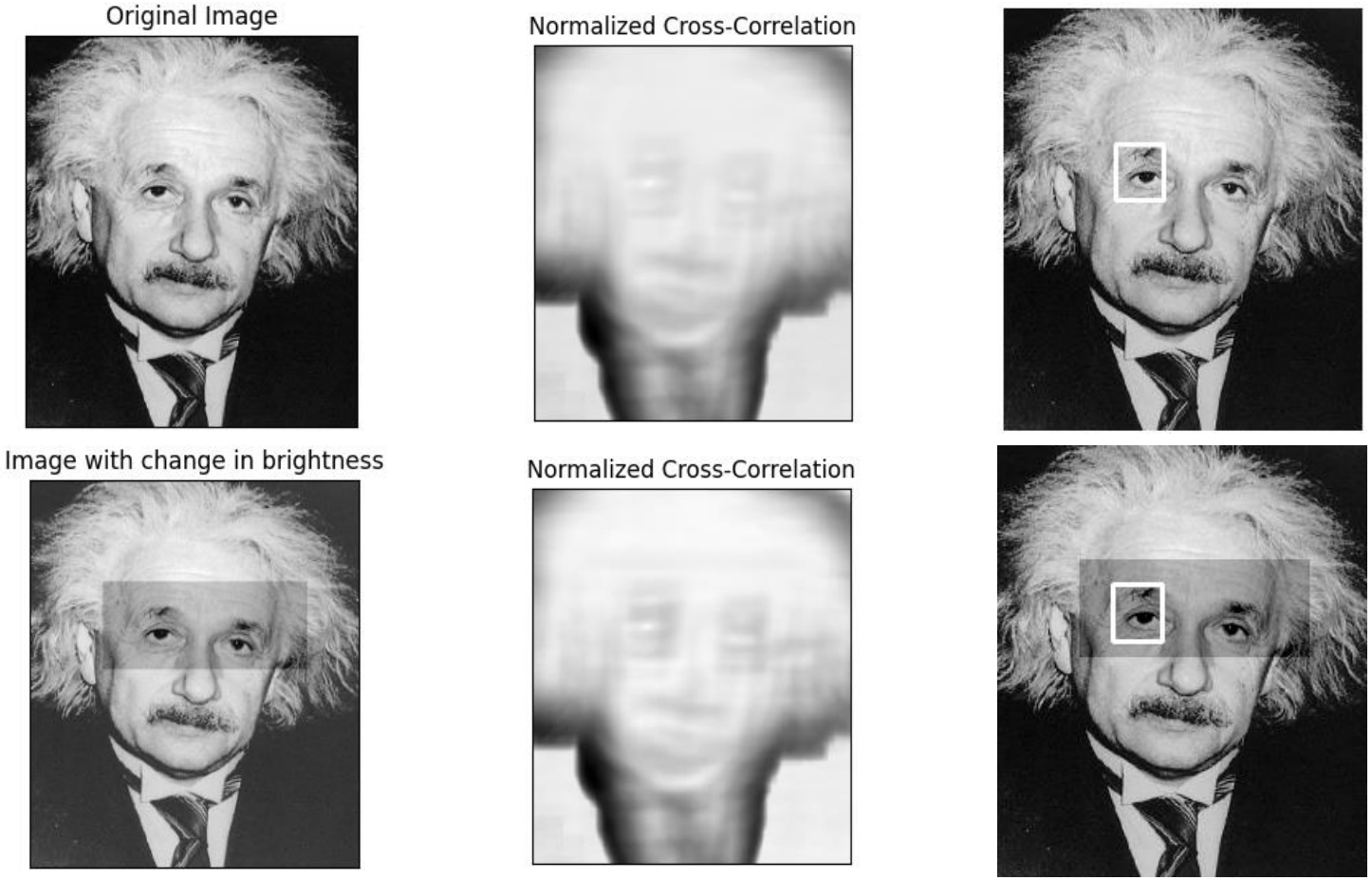
- $NCC = 1$: template and image are identical
- $NCC = -1$: image region is a contrast-reversed version of the filter kernel
- NCC could be squared if contrast reversal doesn't matter.

Note: cross-correlation is a dot product, and it has geometric interpretation

$$\cos\theta = \frac{a \cdot b}{|a||b|} = \frac{a \cdot b}{\sqrt{(a \cdot a)(b \cdot b)}}$$

Template Matching

- **Solution 4**: Normalized Cross-Correlation (NCC)



NCC is the slowest among the possible solutions presented here,
but it’s invariant to local average intensity and constrast!

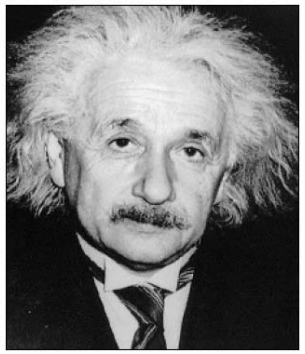
Important Insight

- filters look like the pattern they are intended to find
- filters find patterns they look like

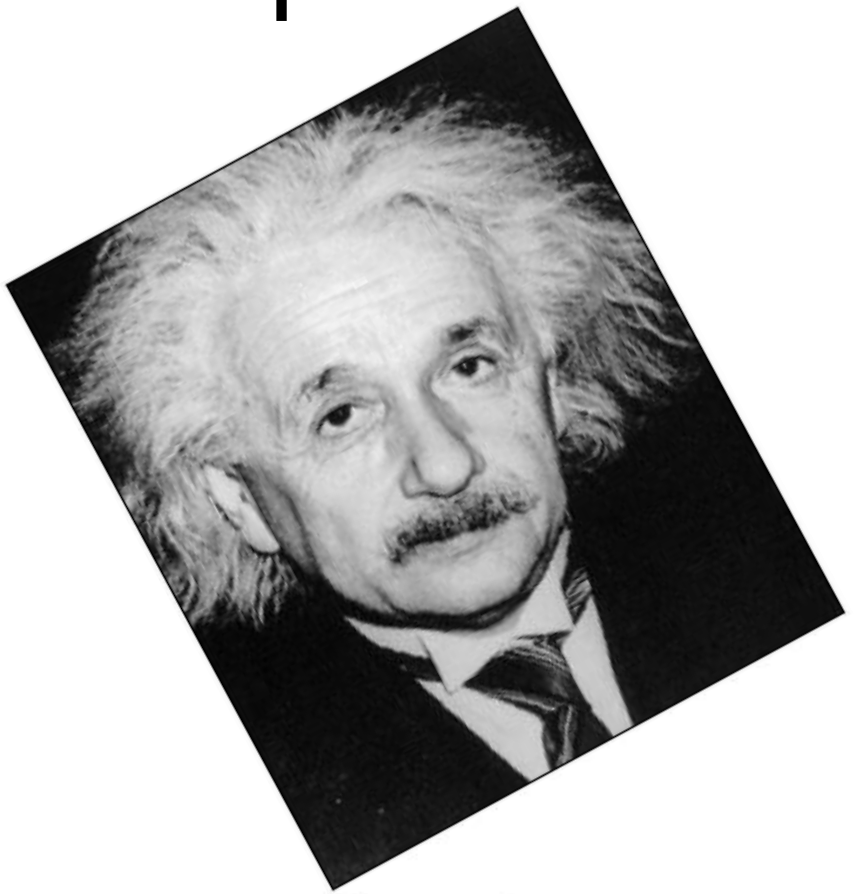
Linear filtering is sometimes referred to as template matching

"**Filters** offer a natural mechanism for finding simple patterns because they **respond most strongly to pattern elements that look like them**. For example, smoothed derivative filters are intended to give a strong response at a point where the derivative is large. At these points, **the kernel of the filter looks like the effect it is intended to detect**."
(Forsyth & Ponce, 2012)

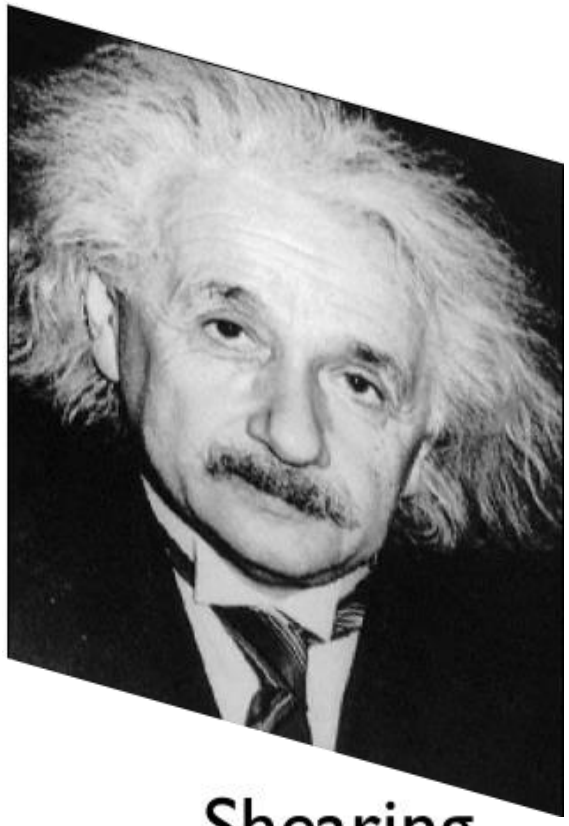
When might template matching fail?



scale



Rotation



Shearing

Or… partial occlusions, different perspective, motion/blur,…

How to make template matching robust to changes in 2D spatial scale?

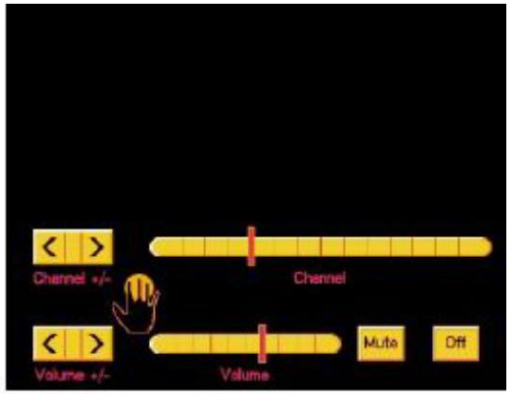
- Key ideas:
  - **Build a scaled representation** (Gaussian Pyramid) **of the image**
  - **Find template matches at all scales**
    - template size constant, image scale varies
    - finding objects whose size is unknown

Template Matching (technical note)

- https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html

- Template match modes in OpenCV: https://docs.opencv.org/4.x/df/dfb/group__imgproc__object.html#ga3a7850640f1fe1f58fe91a2d7583695d

See a practical application of template matching using normalized cross-correlation in Forsyth&Ponce (2012, Ch. 4.6):
finding hands to control television



Hough Transform

Hough Transform

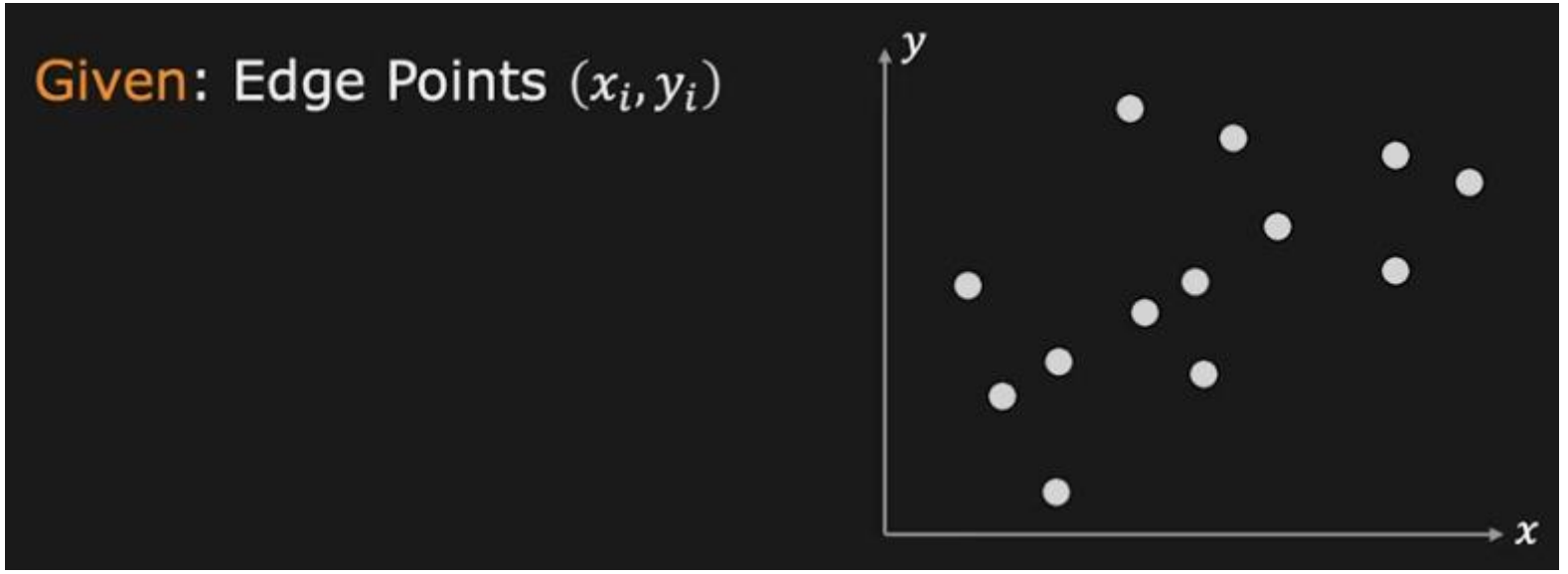
- Another way of detecting (simple) patterns/features/shapes
  - In this case, not directly based on convolution/correlation
  - Useful in boundary detection: which edges in an image actually correspond to the boundary you are looking for?

- Invented by Richard Duda and Peter Hart* (1972), who called it "generalized Hough transform“ after the Paul Hough’s 1962 patent.
  - Duda, R. O., & Hart, P. E. (1972). Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, *15*(1), 11-15.

* One of the authors of the A* search algorithm.

Note: these slides are based on Shree Nayar’s notes.

Line Detection

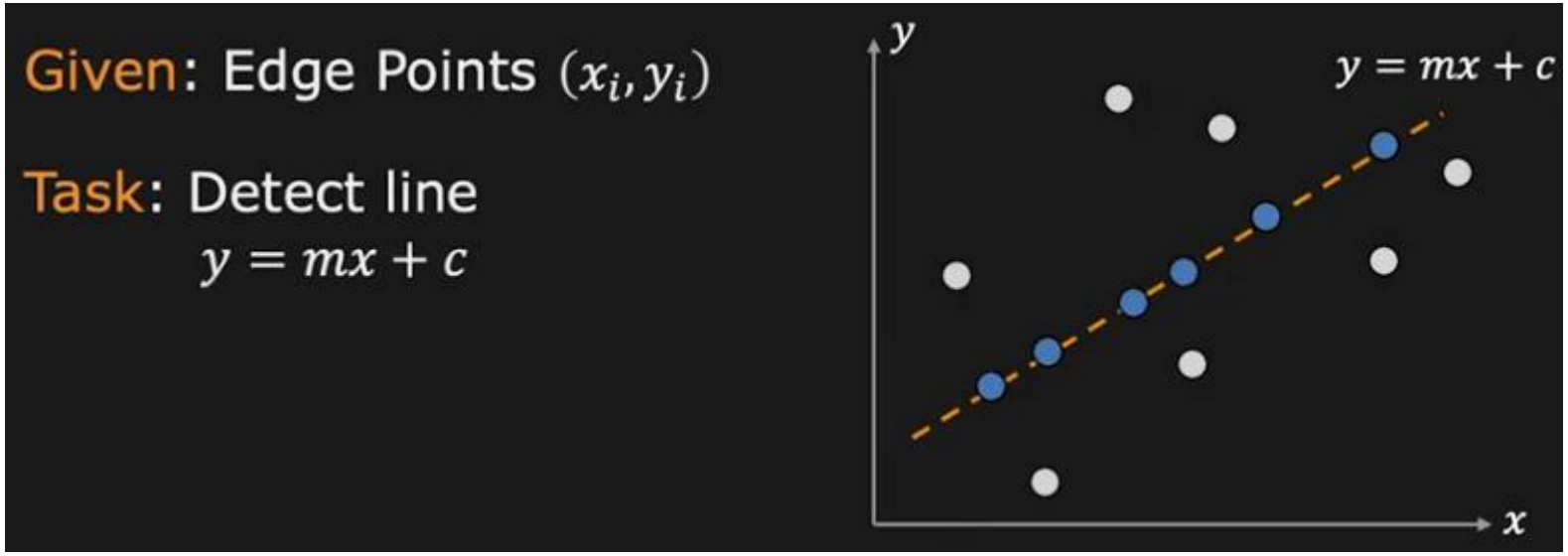


We have a bunch of points in an image (for instance, from an edge map).

We assume there is noise and we deal with incomplete data.

Which points do actually belong to a line?? This is a model fitting problem!

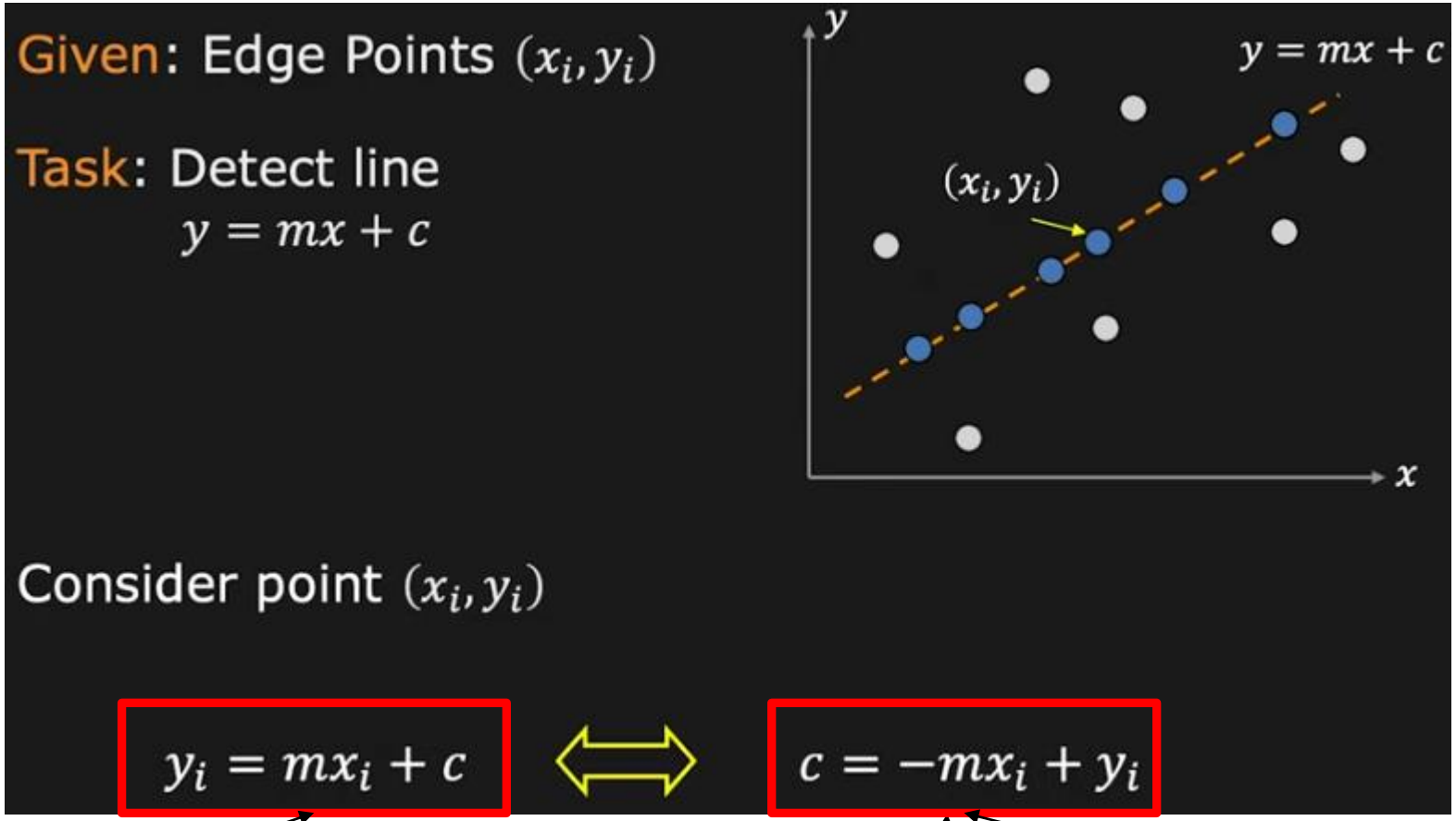
Line Detection



m: slope
c: intercept

We want to find the line embedded in this cluster of points.

Line Detection



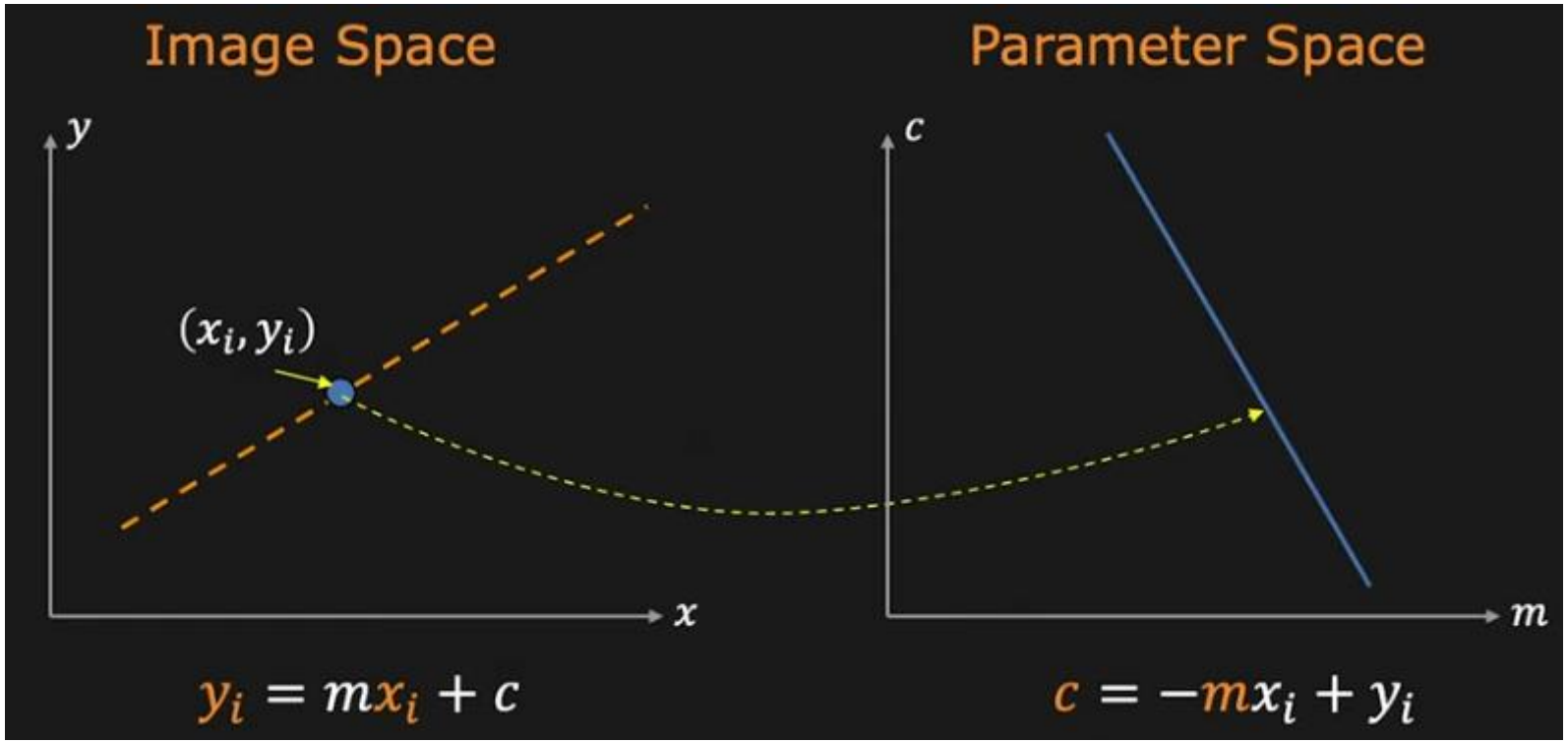
m: slope
c: intercept

Image Space

Parameter Space

This is also a straight line equation. We can rewrite/transform the image space equation in this way.

Line Detection

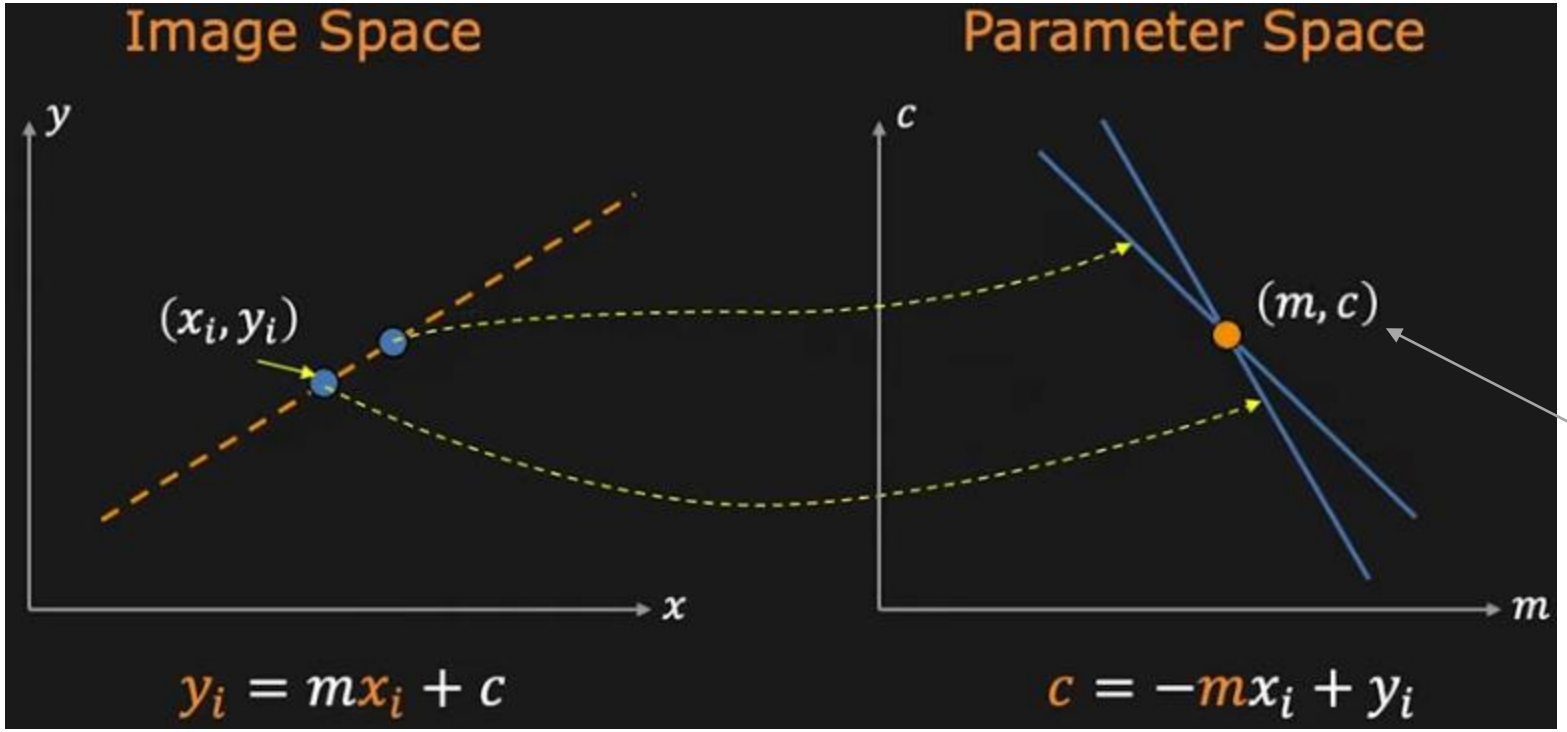


The blue line represents all the straight lines (*c* and *m* values) that pass through (x_i, y_i) in image space

(x_i, y_i) gives a line in parameter space.

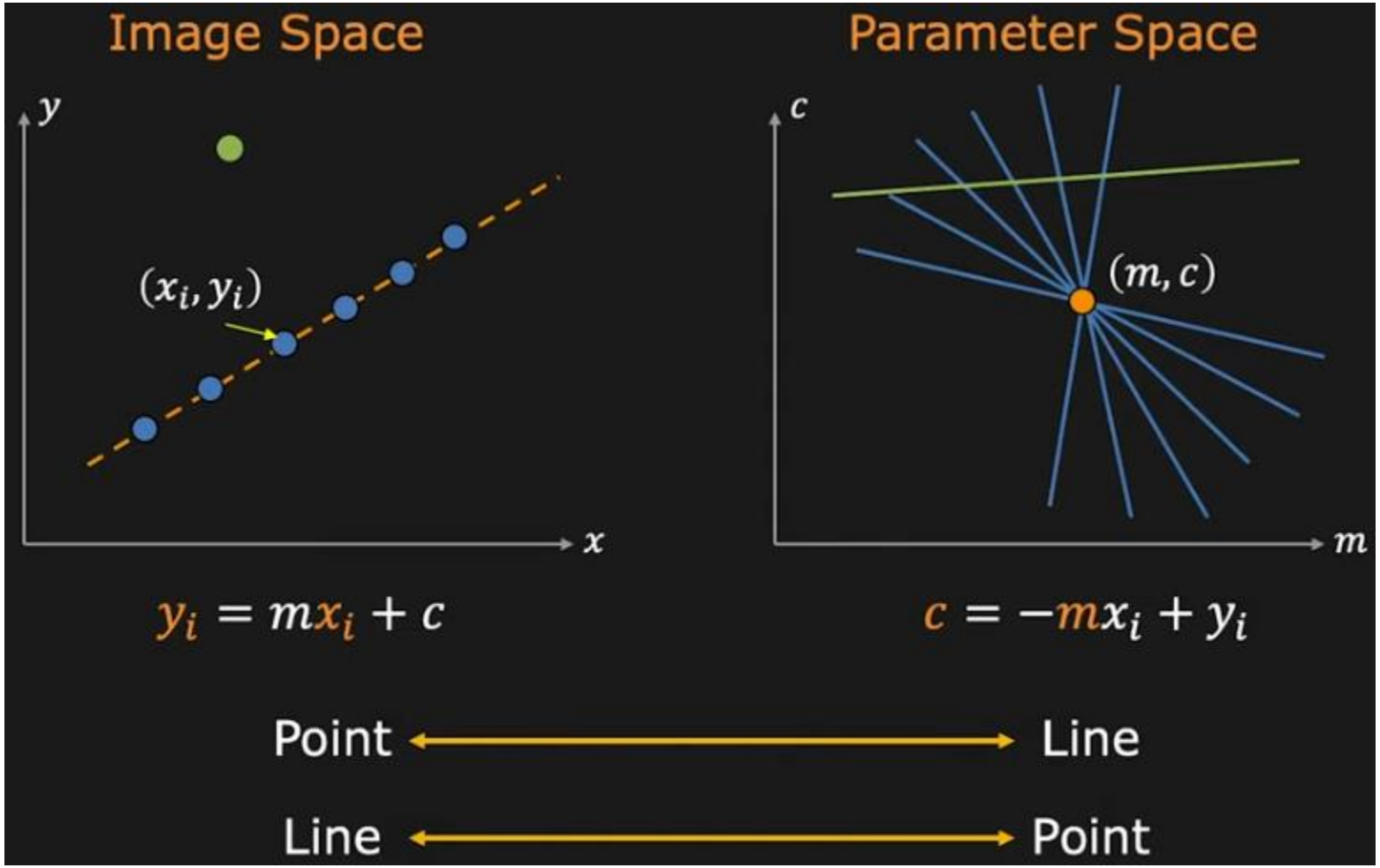
We can look at the problem in these two spaces.

Line Detection



All points belonging to the same line in image space, intersect at the same point in parameter space.

Line Detection

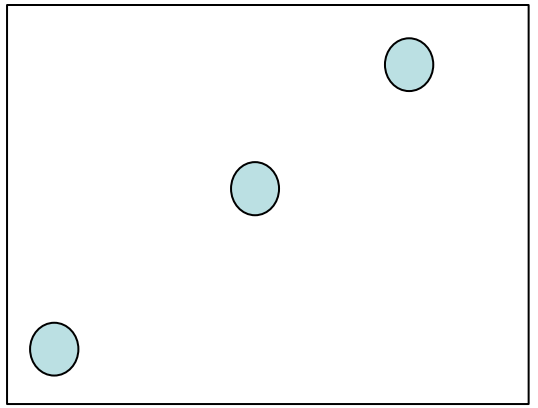


Hough Transform for Line Detection

- Step 1: quantize parameter space (m, c)
- Step 2: create accumulator array $A(m, c)$
- Step 3: set $A(m, c) = 0$ for all (m, c)
- Step 4: for each edge point (x_i, y_i),

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$



$A(m, c)$

c

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

m

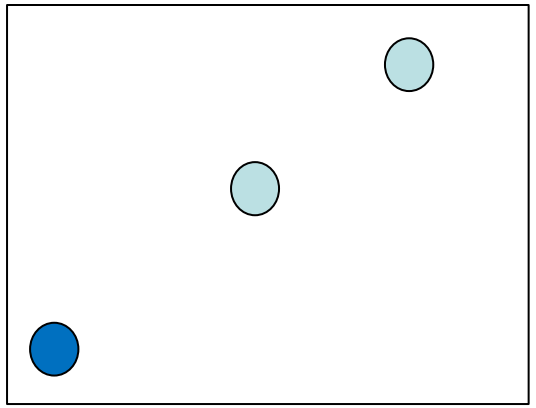
We quantize the parameter space by creating this accumulator array with a resolution that is appropriate for the problem.

Hough Transform for Line Detection

- Step 1: quantize parameter space (m, c)
- Step 2: create accumulator array $A(m, c)$
- Step 3: set $A(m, c) = 0$ for all (m, c)
- Step 4: for each edge point (x_i, y_i),

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$



$A(m, c)$

c					
	1	0	0	0	0
	0	1	0	0	0
	0	0	1	0	0
	0	0	0	1	0
	0	0	0	0	1

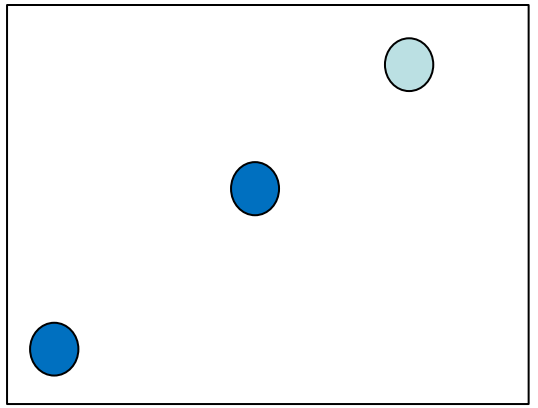
m

Hough Transform for Line Detection

- Step 1: quantize parameter space (m, c)
- Step 2: create accumulator array $A(m, c)$
- Step 3: set $A(m, c) = 0$ for all (m, c)
- Step 4: for each edge point (x_i, y_i),

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$



$A(m, c)$

c					
	1	0	0	0	1
	0	1	0	1	0
	0	0	2	0	0
	0	1	0	1	0
	1	0	0	0	1

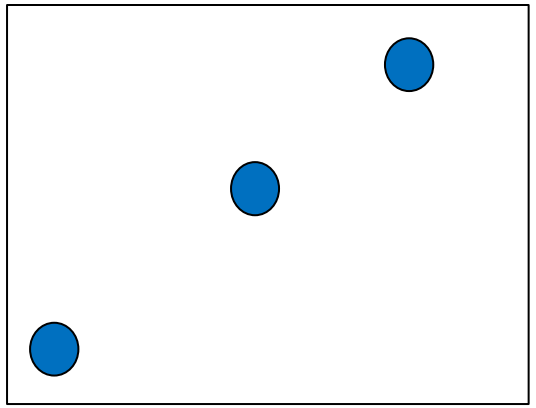
m

Hough Transform for Line Detection

- Step 1: quantize parameter space (m, c)
- Step 2: create accumulator array $A(m, c)$
- Step 3: set $A(m, c) = 0$ for all (m, c)
- Step 4: for each edge point (x_i, y_i),

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$



$A(m, c)$

c					
	1	0	0	0	1
	0	1	0	1	0
	1	1	3	1	1
	0	1	0	1	0
	1	0	0	0	1

m

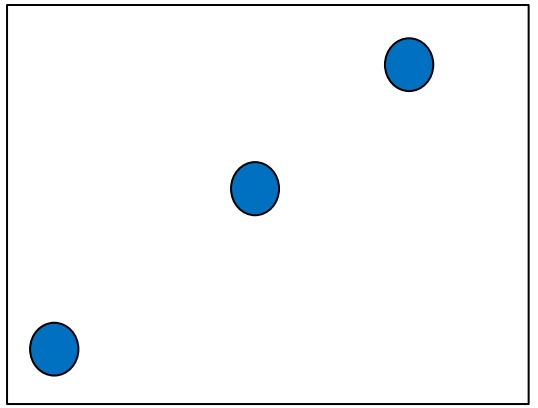
Hough Transform for Line Detection

- Step 1: quantize parameter space (m, c)
- Step 2: create accumulator array $A(m, c)$
- Step 3: set $A(m, c) = 0$ for all (m, c)
- Step 4: for each edge point (x_i, y_i),

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$

- Step 5: find local maxima in $A(x_i, y_i)$



$A(m, c)$

c					
	1	0	0	0	1
	0	1	0	1	0
	1	1	3	1	1
	0	1	0	1	0
	1	0	0	0	1

m

Hough Transform for Line Detection

- Step 1: quantize parameter space (m, c)
- Step 2: create accumulator array $A(m, c)$
- Step 3: set $A(m, c) = 0$ for all (m, c)
- Step 4: for each edge point (x_i, y_i),

$$A(m, c) = A(m, c) + 1$$

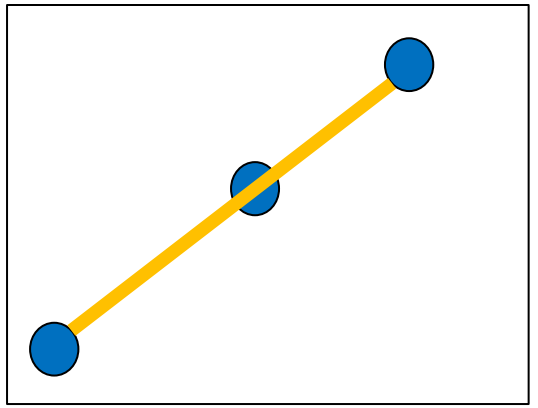
if (m, c) lies on the line: $c = -mx_i + y_i$

- Step 5: find local maxima in $A(x_i, y_i)$

This is like a **voting scheme** that allows you to identify lines!

The point of intersection in parameter space has the coefficients of the line we're looking for.

Image



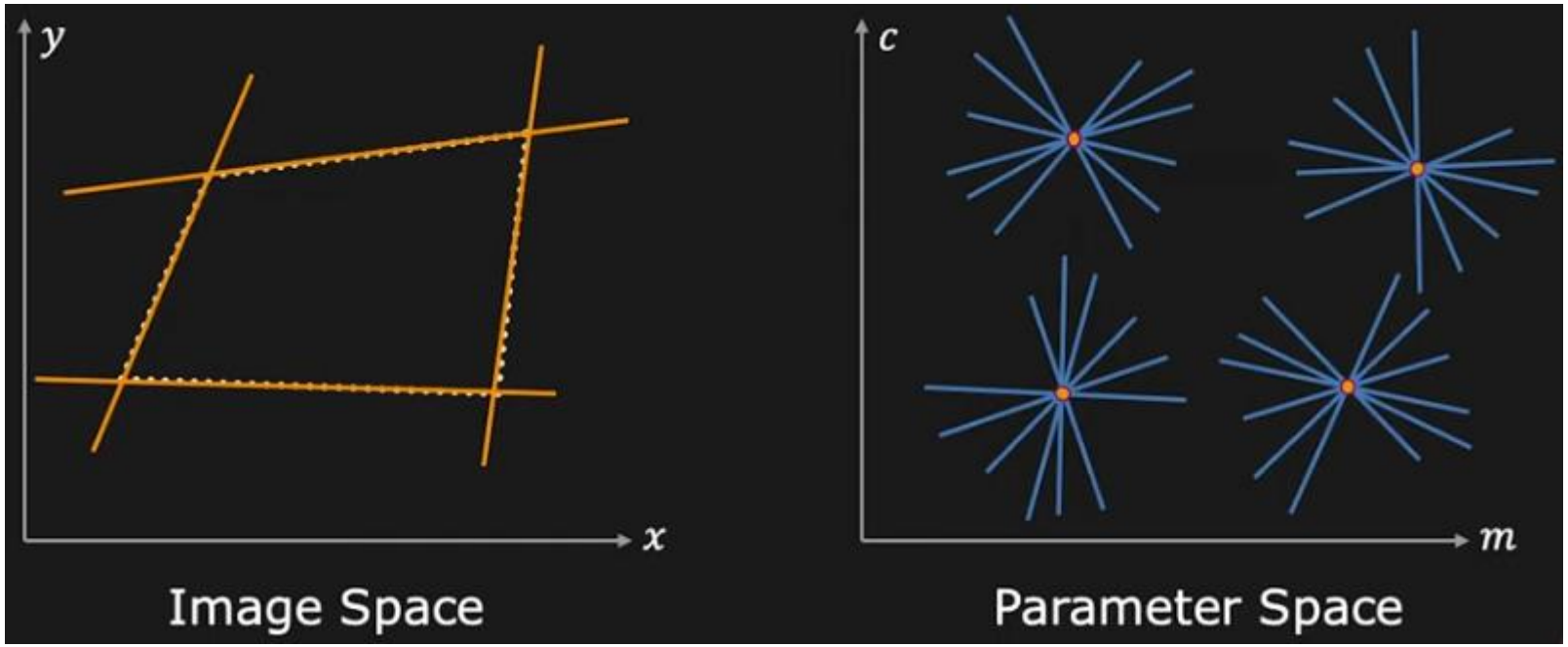
$A(m, c)$

c

1	0	0	0	1
0	1	0	1	0
1	1	3	1	1
0	1	0	1	0
1	0	0	0	1

m

Multiple Line Detection



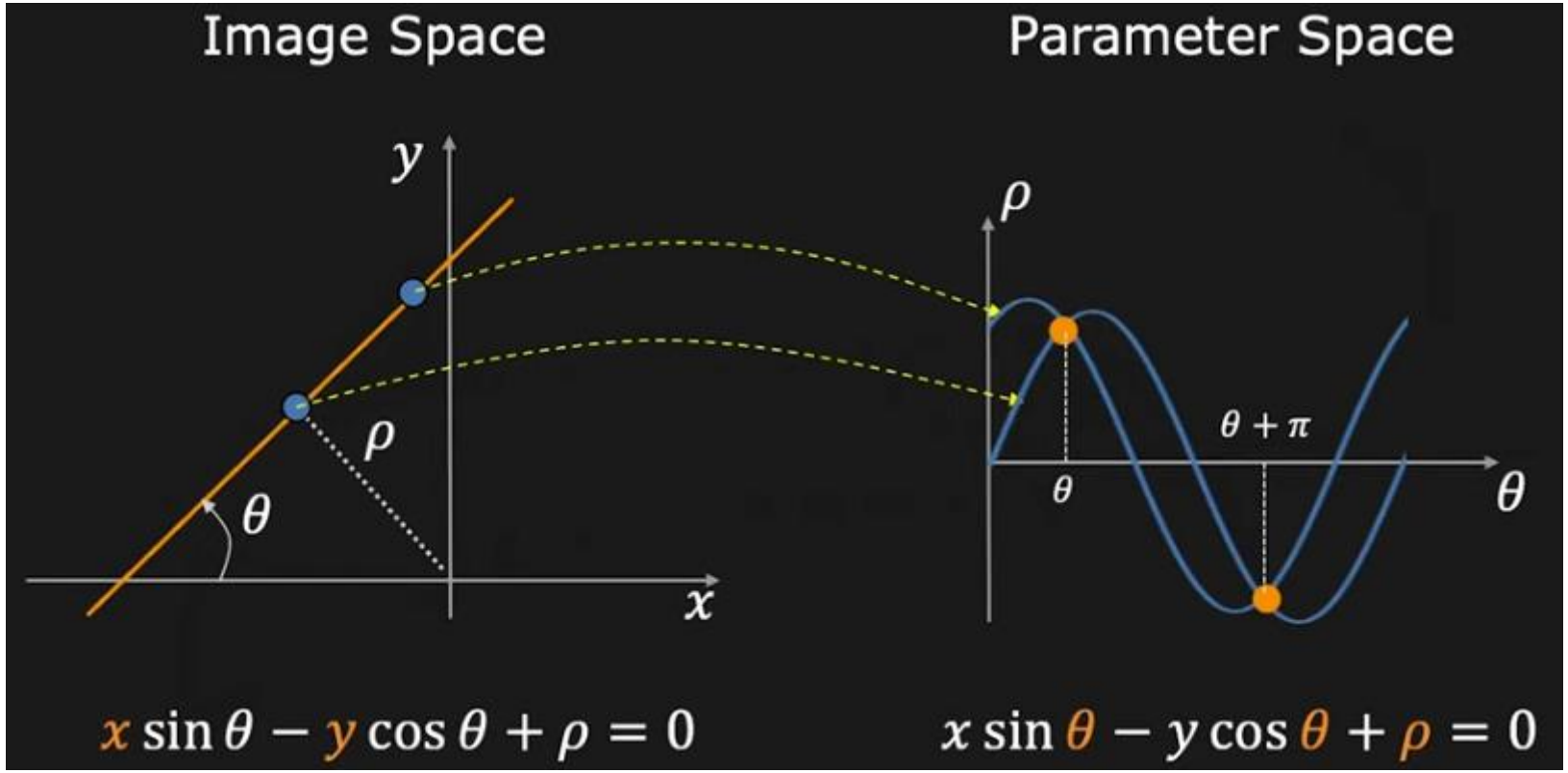
Each one of the line segments in image space will give an intersection in parameter space!

Better Parametrization

- Issue:
  - slope of the line $-\infty \leq m \leq \infty$
    - Large accumulator
    - More memory and computation

- Solution:
  - use $xsin\theta - ycos\theta + \rho = 0$
    - Orientation θ is finite: $0 \leq \theta < \pi$
    - Distance ρ is finite (distance of the line from the image origin)

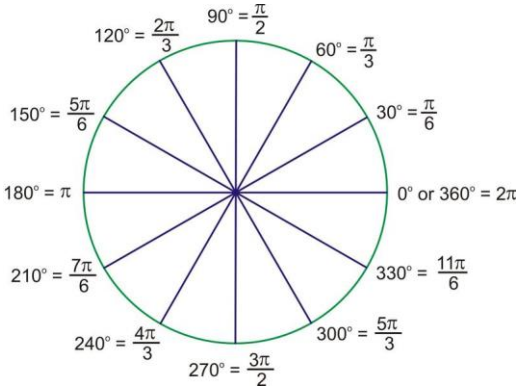
Better Parametrization



The intersection of these sinusoids yields the parameters of the straight line on which the two image points lie.

We see that we get two intersections here, where one of them corresponds to θ and the other one corresponds to $\theta + \pi$, which is essentially the same straight line.

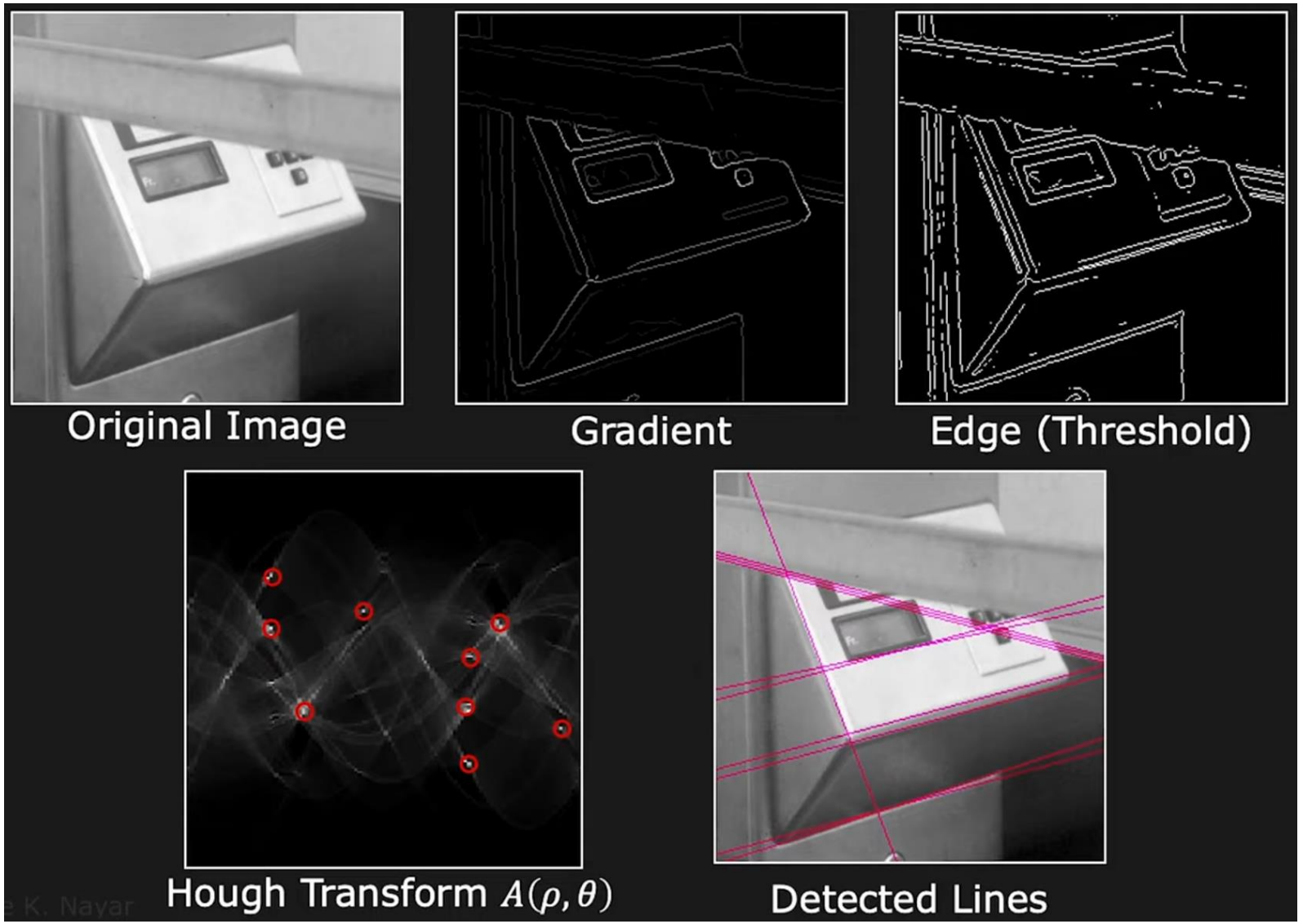
With our new line parameterization, **the point is going to map to a sinusoid in parameter space!!!** In this case, we vote in this sinusoid space!



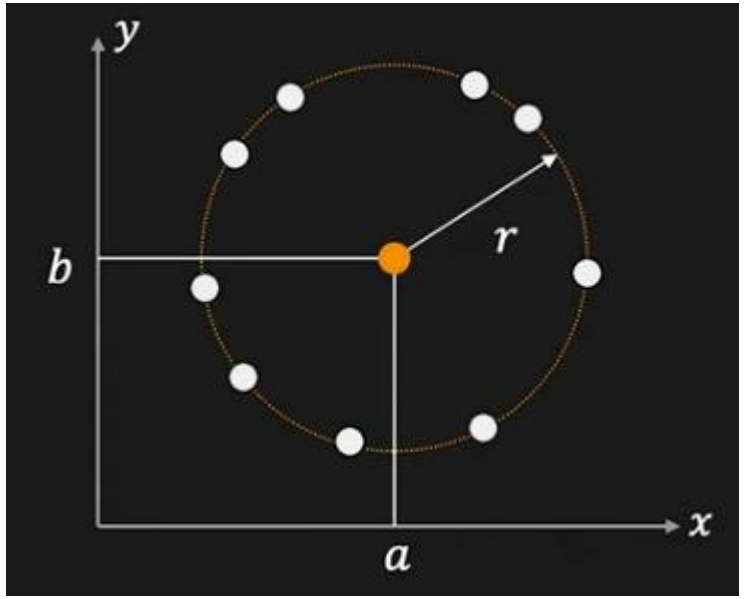
Issues

- How big should the accumulator cells be?
  - very low-resolution accumulator array (large cells) → many different lines could fall within the same cell
  - too small → we may not have any cell that gets a high enough number of votes to be declared a maximum
- How many lines do we actually have in the image?
  - Need of a peak-finding algorithm to count the peaks in the accumulator array
- Handling inaccurate edge locations
  - Lines are not expected to be perfectly straight → Increment small patch of cells in accumulator rather than single cell (more resilient to noise)

Some line detection results

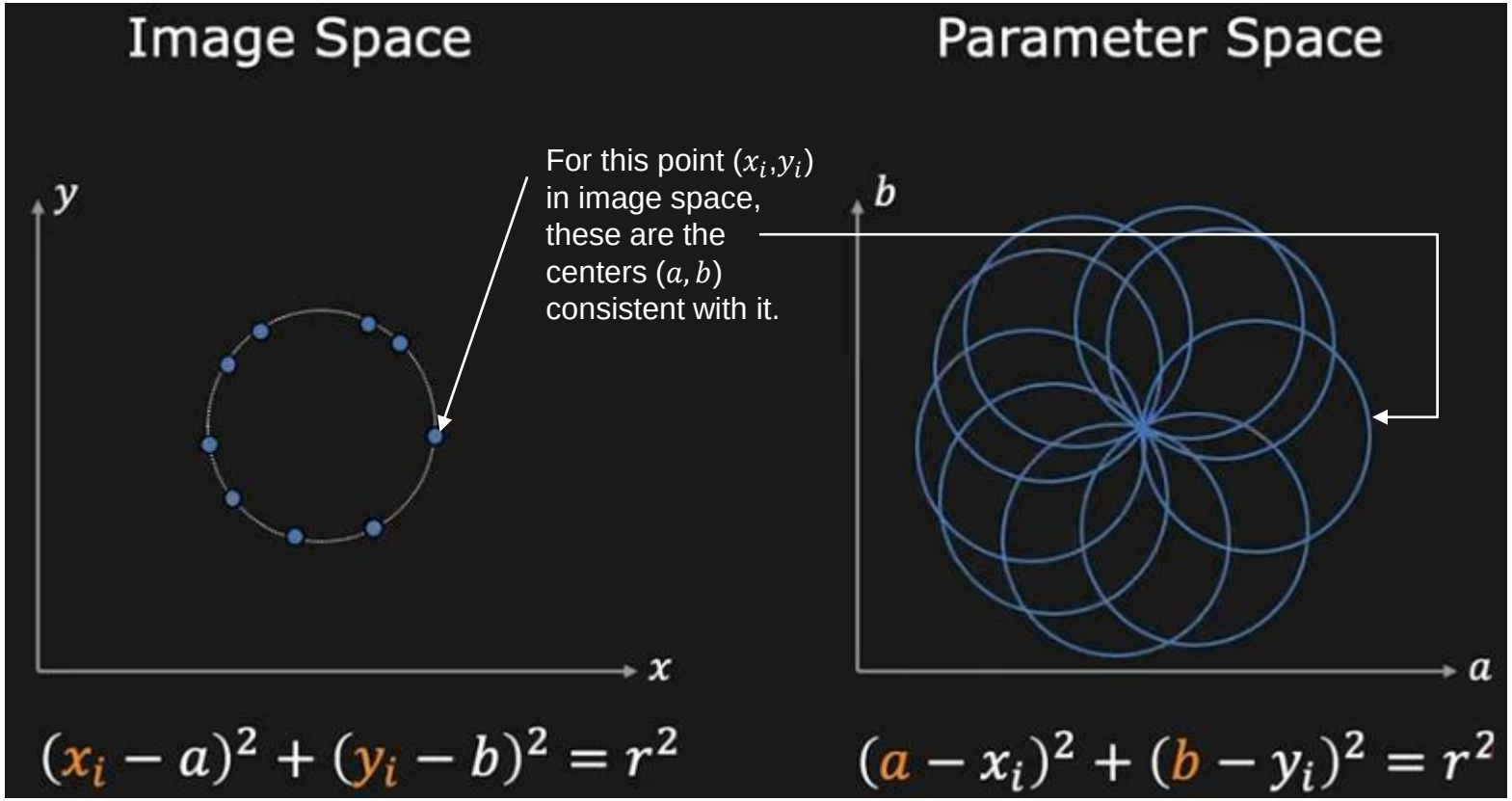


Circle Detection



We need a parametrization suitable for the shape/feature we want to detect. To detect circles, (a, b) corresponds to the center of the circle, while r is the radius.

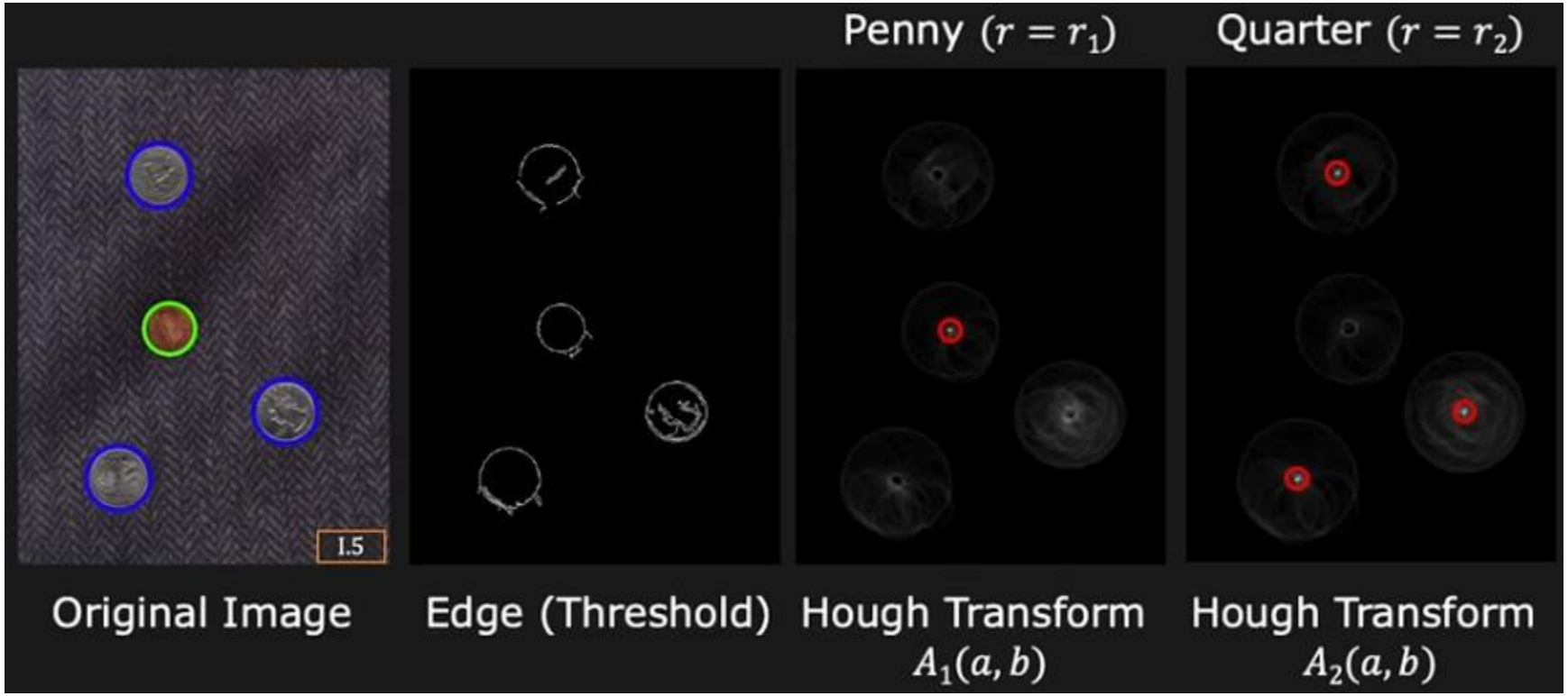
In this example, to make the problem easier, we asume that r is known.



One point from image space maps to a circle in parameter space.

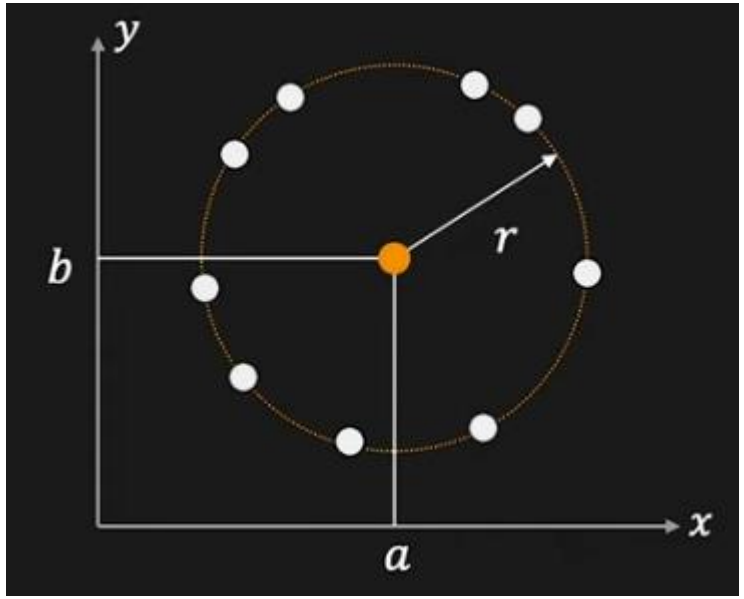
Again, all circles in parameter space intersect at one point (a, b values that pass through all points belonging to the circle in image space).

Circle Detection

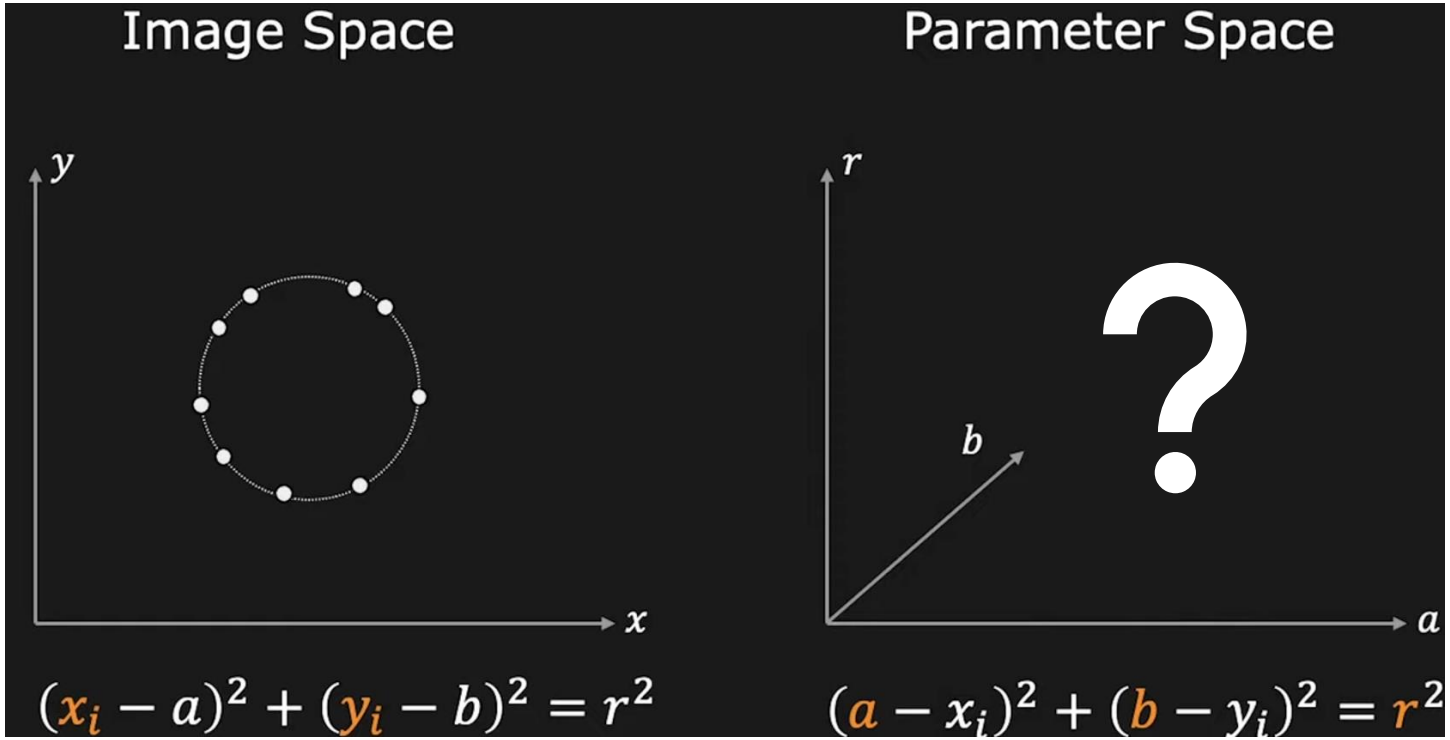


Example: Let's imagine an automatic process where we want to locate coins whose diameter is known (given that we start from a properly calibrated system).

Circle Detection



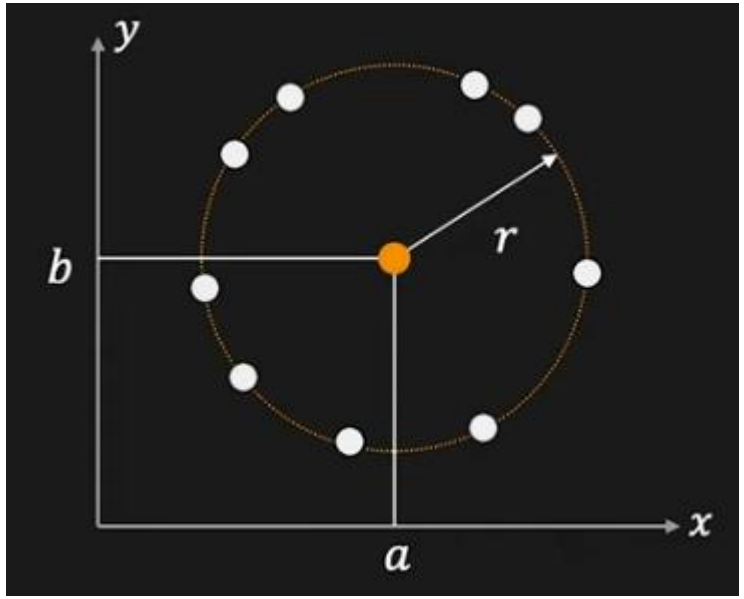
If radius r is also unknown.



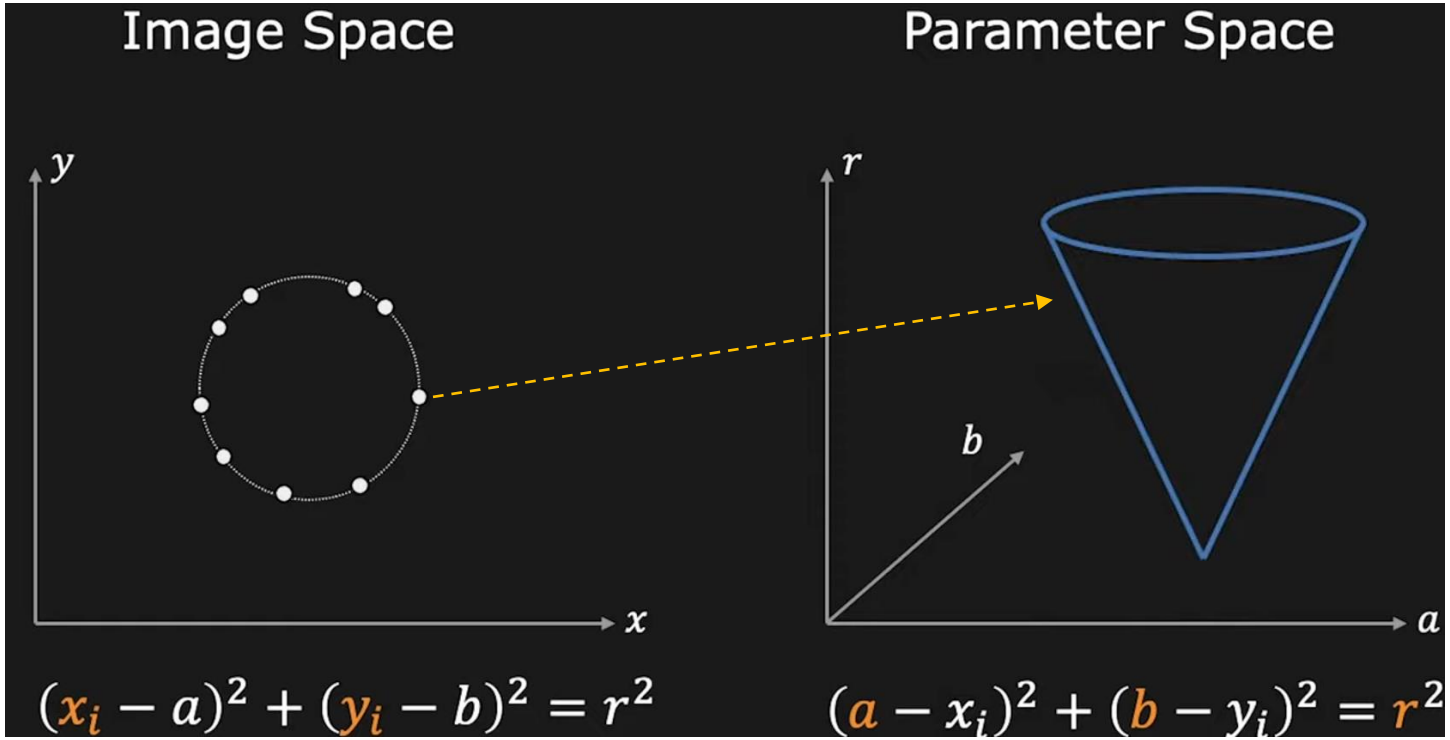
For any given (x_i, y_i), what is the shape we'll have in the parameter space???

🤔

Circle Detection



If radius r is also unknown.



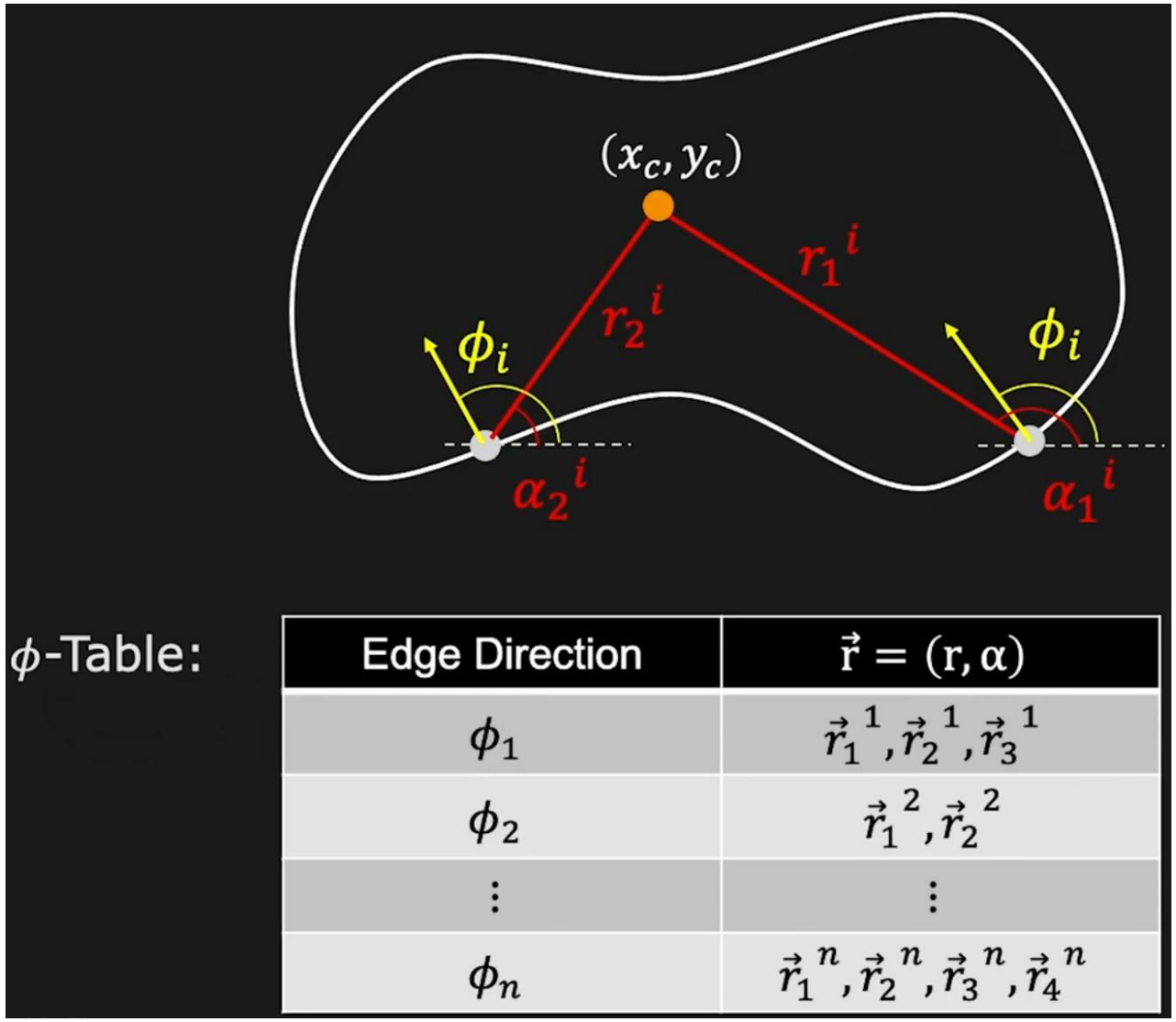
Accumulator Array: $A(a, b, r)$. If the dimensionality of the parameter space increases, Hough Transform becomes impractical.

For each point, now we need to vote along a cone. Why? Because we can go from a small radius to a large one, while keeping the same center.

Generalized Hough Transform

- We have seen how to use the Hough Transform to detect very simple shapes (i.e. that can be described by a small number of parameters), but…
  - **How can we find more complex shapes that cannot be expressed using an equation???**

Generalized Hough Transform



ϕ-Table:

Edge Direction	$\vec{r} = (r, \alpha)$
ϕ_1	$\vec{r}_1^{\,1}, \vec{r}_2^{\,1}, \vec{r}_3^{\,1}$
ϕ_2	$\vec{r}_1^{\,2}, \vec{r}_2^{\,2}$
⋮	⋮
ϕ_n	$\vec{r}_1^{\,n}, \vec{r}_2^{\,n}, \vec{r}_3^{\,n}, \vec{r}_4^{\,n}$

1) We define a reference point (x_c, y_c). Our accumulator array has (x_c, y_c). If we have a pair (x_c, y_c) with many votes, we'd have found our shape.

2) We parametrize our shape using the result of edge detection and edge direction information.

Edge direction:	$\phi_i \qquad 0 \leq \phi_i < 2\pi$
Edge location:	$\vec{r}_k^{\,i} = (r_k^{\,i}, \alpha_k^{\,i})$

$\vec{r}_k^i$: k^{th} point with orientation ϕ_i, composed of r_k^i length (from the edge point to the reference point) and α_k^i angle with respect to the horizontal axis.

3) We create the ϕ-Table, and this would be the Hough model associated with this object (we'll use it to find the object in the image).

Note: the object should appear in the image with the same orientation and scale. If we want to handle scale (s) and rotation (θ), we should include them in the accumulator array: $A(x_c, y_c, s, \theta)$

Generalized Hough Transform

- Create accumulator array $A(x_c, y_c)$
- Set $A(x_c, y_c) = 0$ for all (x_c, y_c)
- For each edge point (x_i, y_i, ϕ_i),

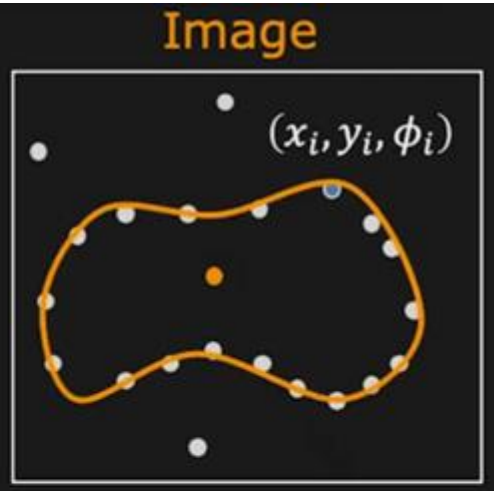
  For each entry $\phi_i \rightarrow \vec{r}_k{}^i$ in ϕ – table,

  $$x_c = x_i \pm r_k{}^i \cos(\alpha_k{}^i)$$

  $$y_c = y_i \pm r_k{}^i \sin(\alpha_k{}^i)$$

  $$A(x_c, y_c) = A(x_c, y_c) + 1$$

- Find local maxima in $A(x_c, y_c)$



$A(x_c, y_c)$

x_c					
	0	0	0	0	0
	0	2	0	1	0
	0	0	4	1	0
	0	2	0	0	0
	0	0	0	1	0

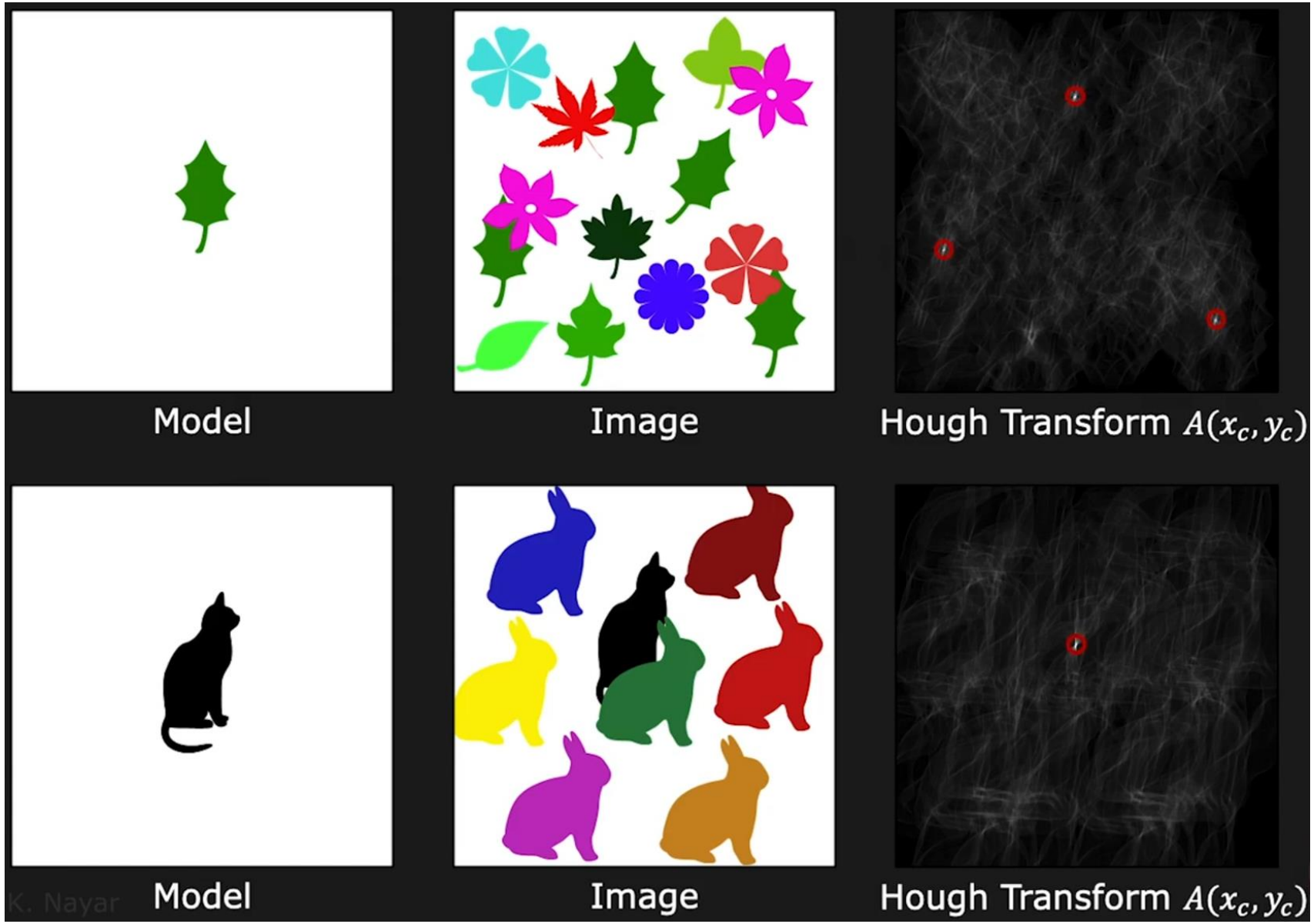
y_c

4) We create the accumulator array from the reference point given, and we initialize it to 0.

5) For each edge point, we go to the ϕ-Table and recover the (r, α) pairs associated with the ϕ_i of the edge point. Then, we compute the “expected” reference point (x_c, y_c), and we increase the accumulator in that position.

6) We find the local maxima in the accumulator, and we can recover the position of the shape we’re looking for because we have the reference point.

Some Results



Some conclusions about the Hough Transform

- It works on disconnected edges, and it's relatively insensitive to occlusion and noise

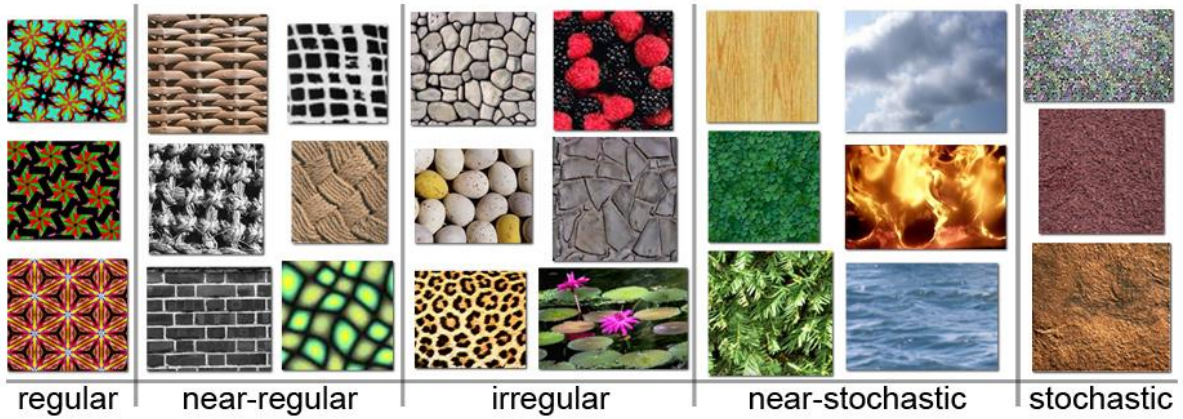
- It's very effective for simple parametric shapes (lines, circles, etc.)

- It can work with complex shapes (Generalized Hough Transform) but, if you increase the dimensionality of your parameter space, it can require a huge amount of memory and can be very computationally expensive!

Textures

What is texture?

- **Texture is detail** in an image that is at a scale **too small to be resolved into its constituent elements and** at a scale **large enough to be apparent** in the spatial distribution of image measurements.
- Textures are also thought of as **patterns composed of repeated instances** of one (or more) identifiable elements, called ***textons***.
  - e.g. bricks in a wall, spots on a cheetah



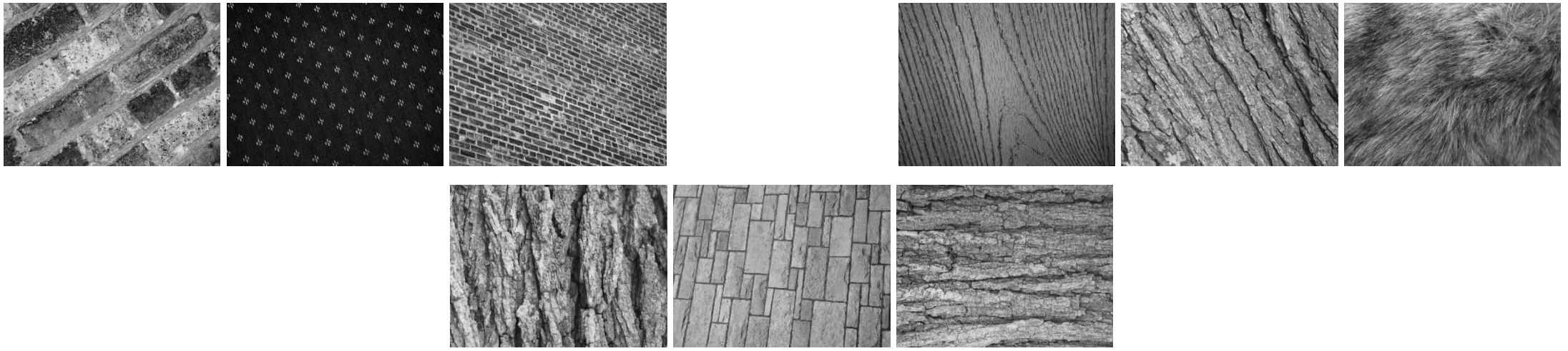
Source

Important insight: we need a region to have a texture.

Note: most of these slides are based on those by Leonid Sigal, Trevor Darrell and Derek Hoiem

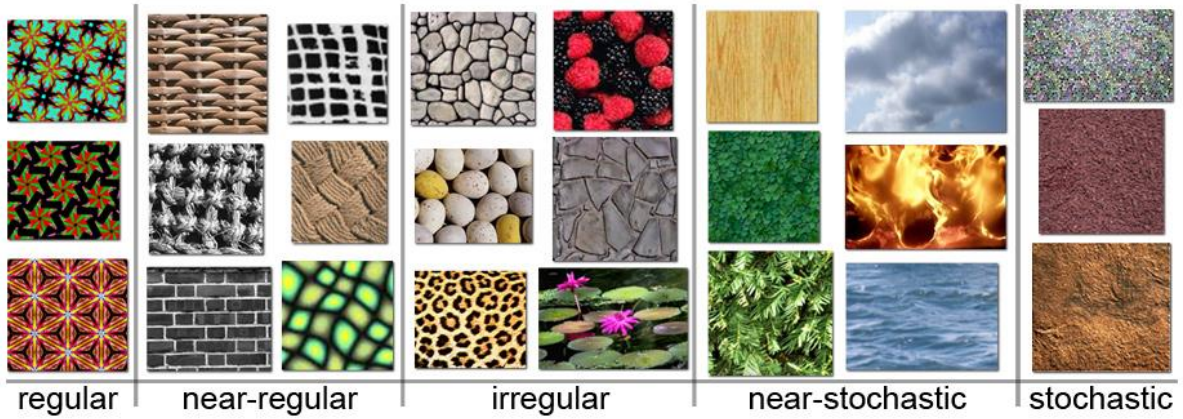
What is texture?

- Texture can be a **strong cue to identify objects** if they have distinctive material properties.
- Texture can be a **strong cue to an object's shape** based on the deformation of the texture from point to point.
  - Estimating surface orientation or shape from texture is known as "shape from texture"



What is texture?

- Two main questions here:
  - Texture **synthesis**: how do we generate new examples of a texture? For instance, see https://una-dinosauria.github.io/efros-and-leung-js/ (https://people.eecs.berkeley.edu/~efros/research/EfrosLeung.html) for an interactive demo using image inpainting (fill-in regions).
  - Texture **analysis**: how do we represent texture? We'll focus on this.

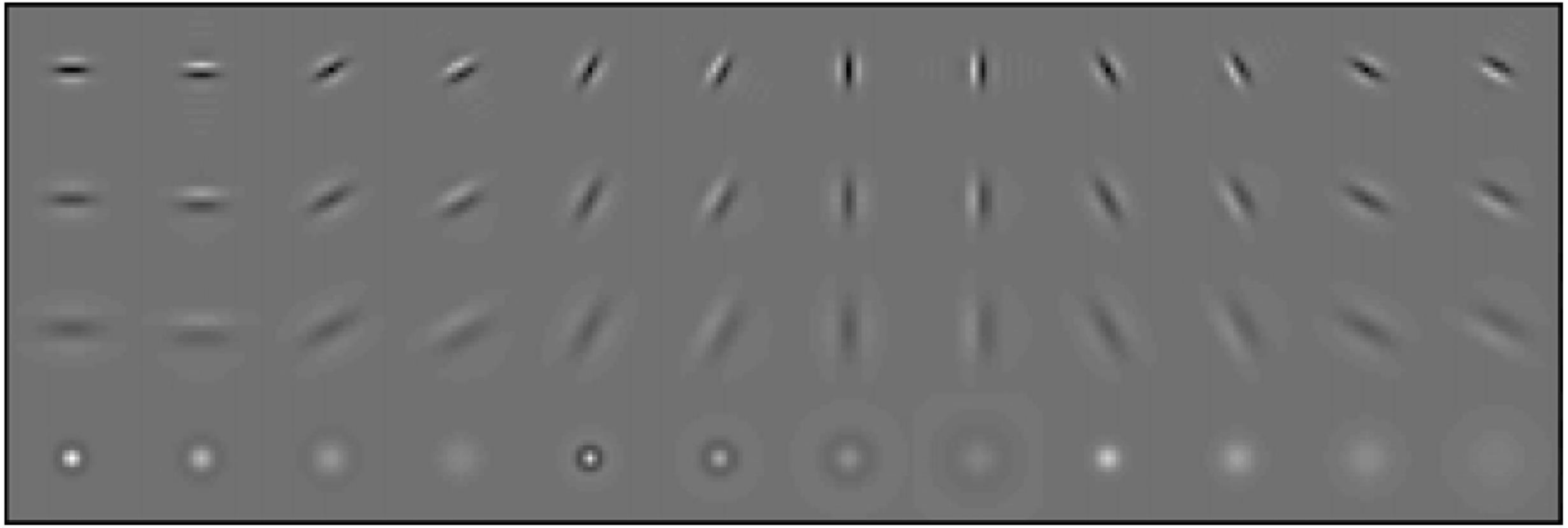


Source

Texture Representation

- **Observation**: Textures are made up of generic sub-elements, repeated over a region with similar statistical properties
- **Idea**: Find the sub-elements with filters, then represent each pixel in the image with a summary of the pattern in the local region
- **Question**: What filters should we use?
- **Answer**: Human vision suggests spots and oriented edge filters with different orientations and scales
- **Question**: How do we “summarize”?
- **Answer**: Compute the mean or maximum of each filter response over the region. Other statistics can also be useful.

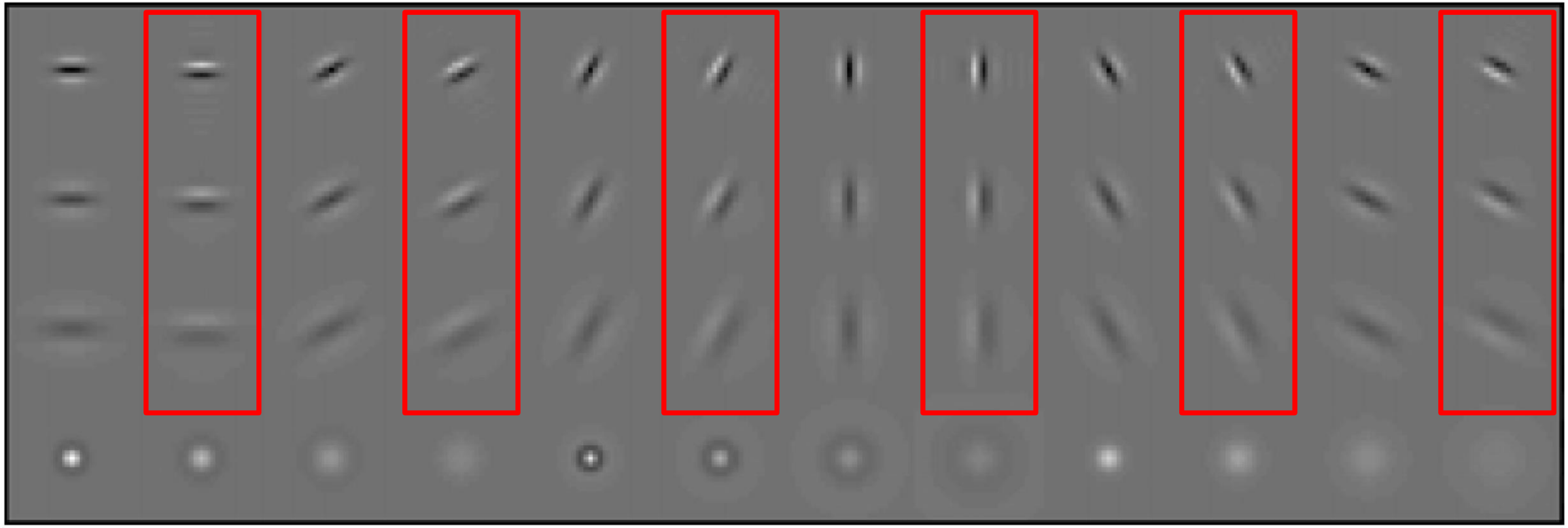
Texture Representation



The **filter bank** used in Leung, Thomas, and Jitendra Malik. "Representing and recognizing the visual appearance of materials using three-dimensional textons." International journal of computer vision 43 (2001): 29-44. Total of **48 filters**

Texture Representation

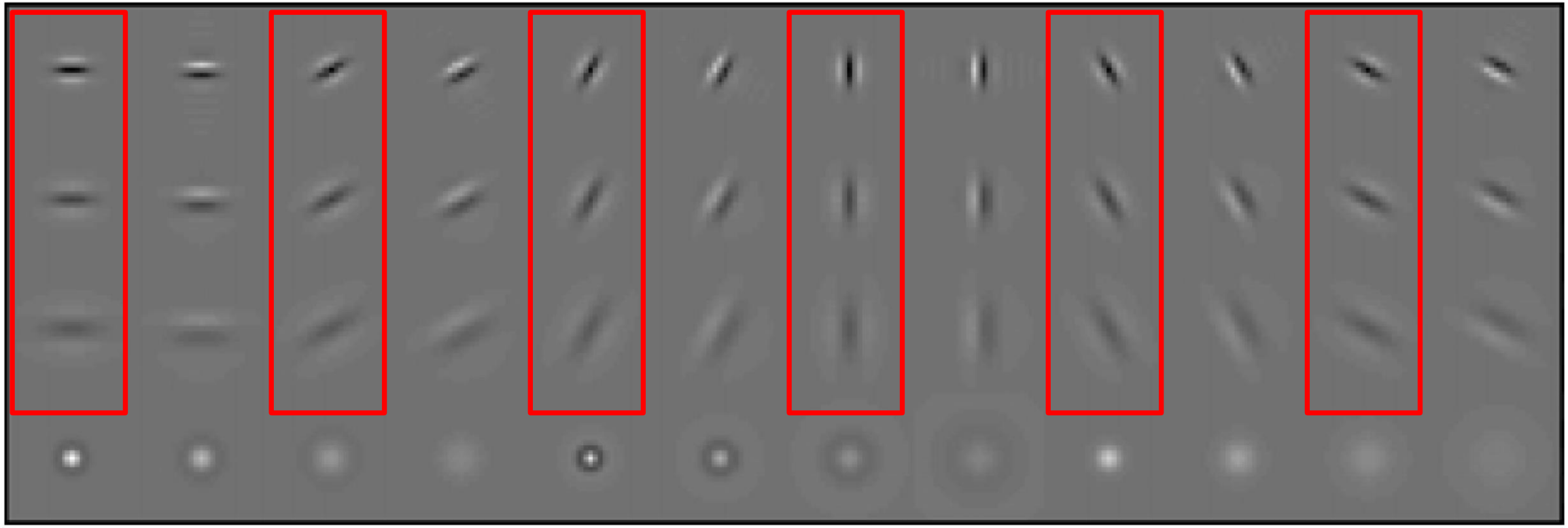
First derivative of Gaussian at 6 orientations and 3 scales



The **filter bank** used in Leung, Thomas, and Jitendra Malik. "Representing and recognizing the visual appearance of materials using three-dimensional textons." International journal of computer vision 43 (2001): 29-44. Total of **48 filters**

Texture Representation

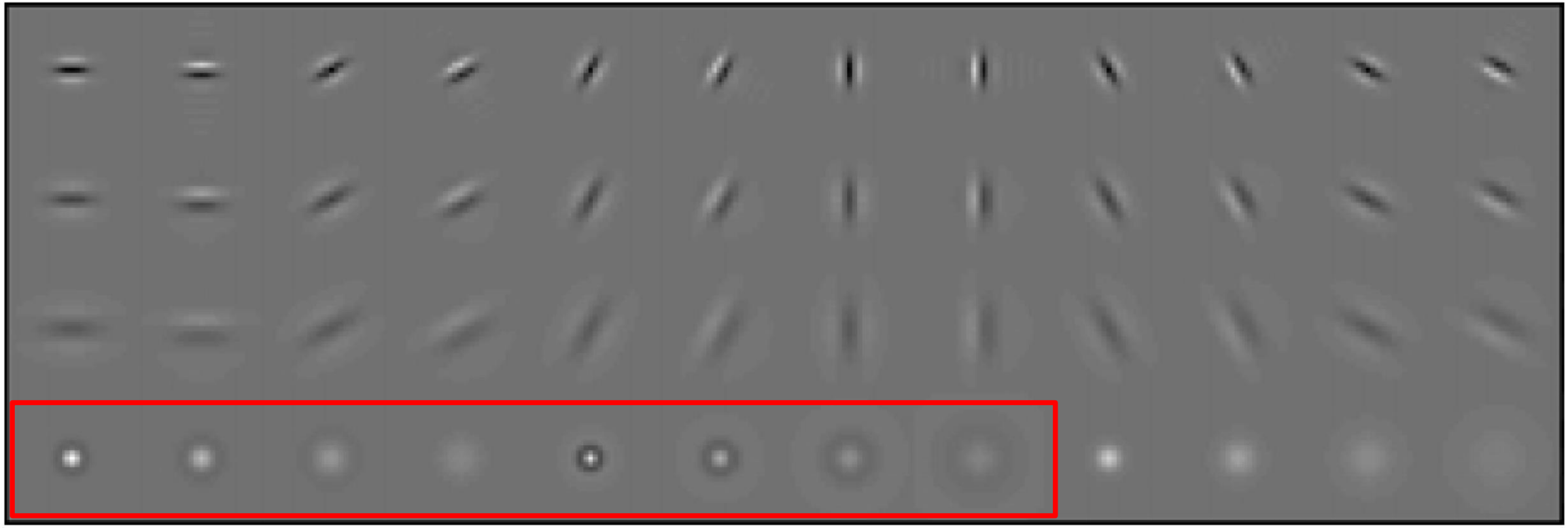
Second derivative of Gaussian at 6 orientations and 3 scales



The **filter bank** used in Leung, Thomas, and Jitendra Malik. "Representing and recognizing the visual appearance of materials using three-dimensional textons." International journal of computer vision 43 (2001): 29-44. Total of **48 filters**

Texture Representation

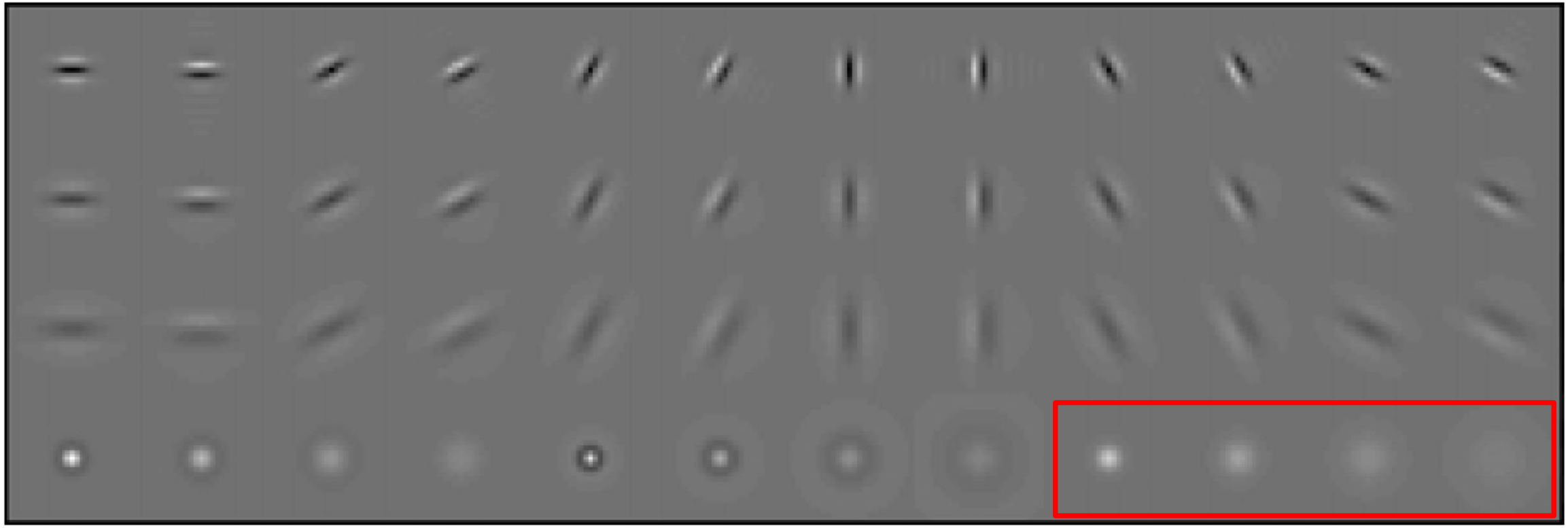
Laplacian of Gaussian filters at different scales



The **filter bank** used in Leung, Thomas, and Jitendra Malik. "Representing and recognizing the visual appearance of materials using three-dimensional textons." International journal of computer vision 43 (2001): 29-44. Total of **48 filters**

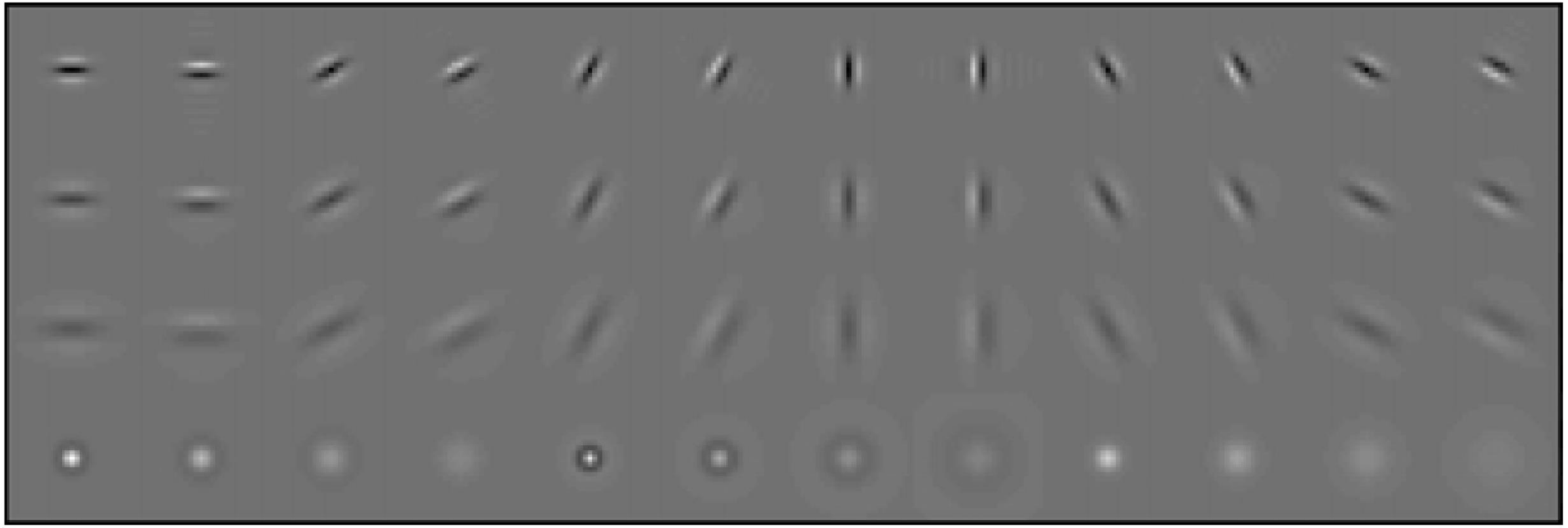
Texture Representation

Gaussian filters at different scales



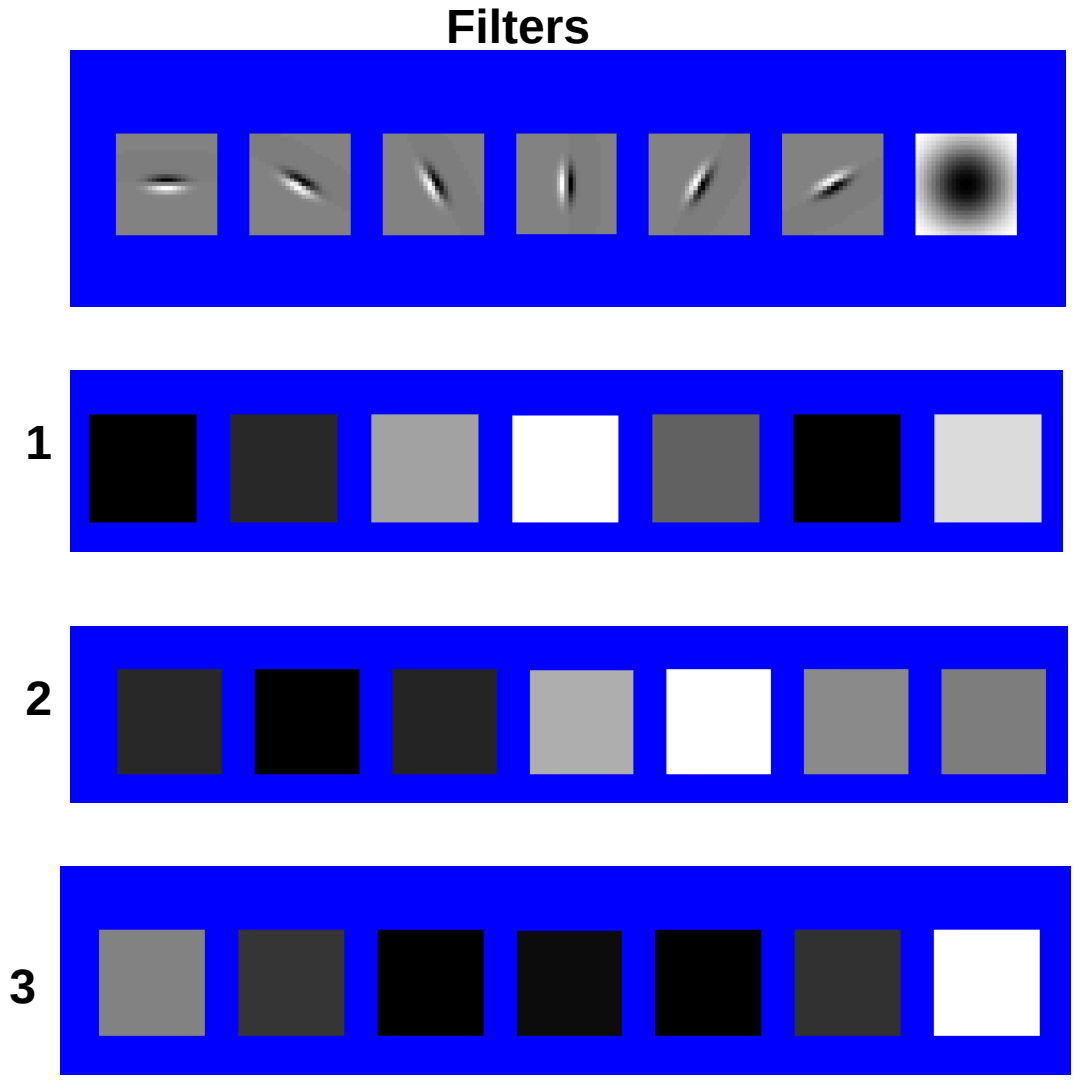
The **filter bank** used in Leung, Thomas, and Jitendra Malik. "Representing and recognizing the visual appearance of materials using three-dimensional textons." International journal of computer vision 43 (2001): 29-44. Total of **48 filters**

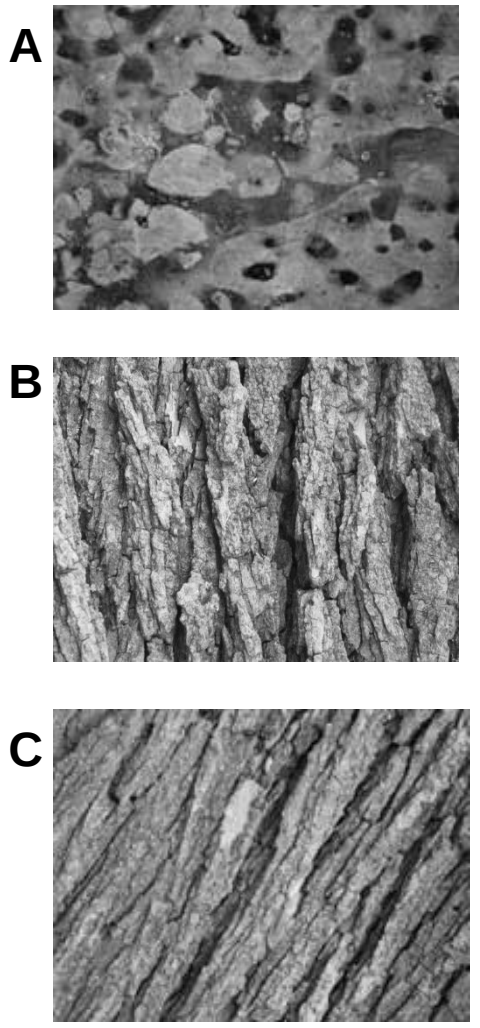
Texture Representation



Result: 48-channel “image”

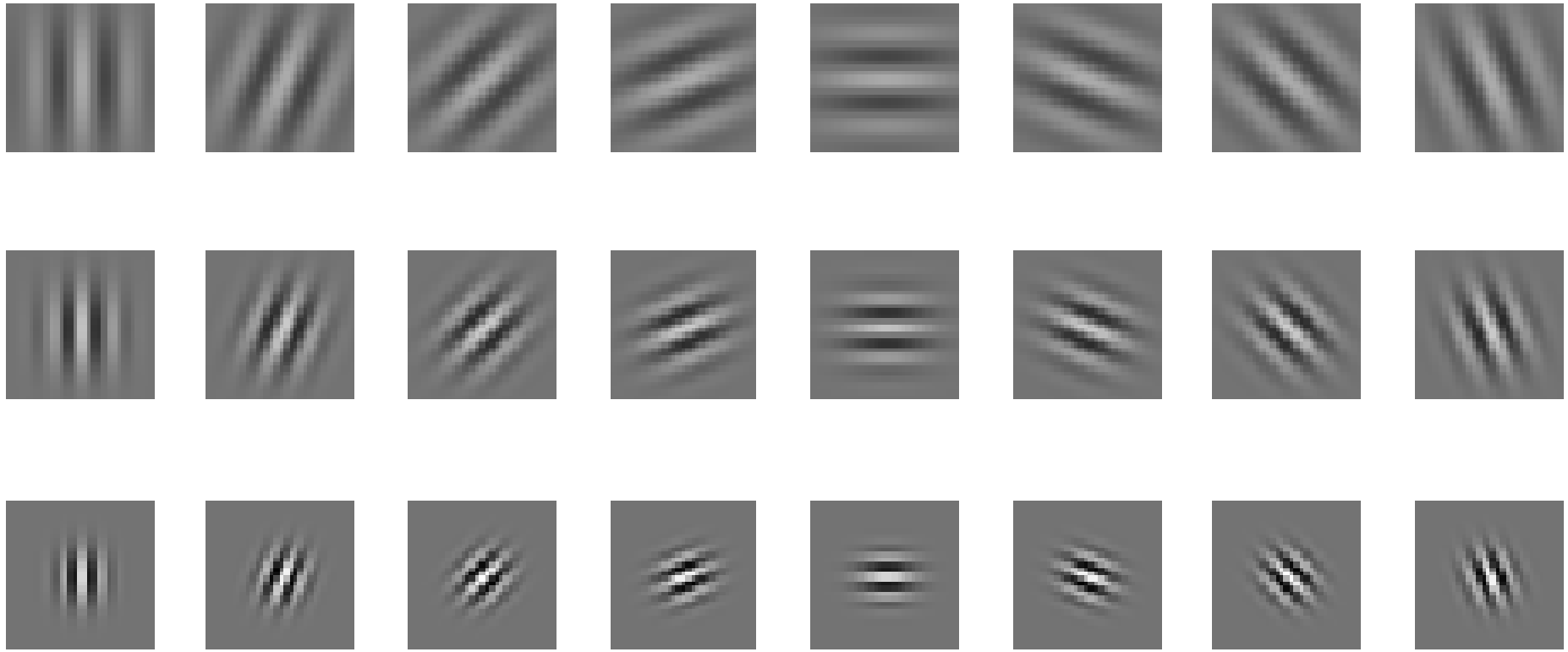
Can you match the texture to the response?





Texture Representation

There are other filter banks like the **Gabor filter bank** for texture analysis



Example of Gabor filter bank for 3 scales and 8 orientations

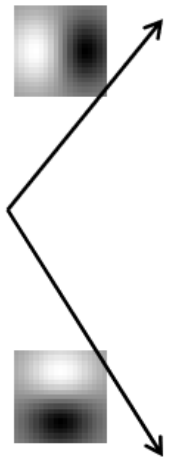
Grigorescu, Simona E., Nicolai Petkov, and Peter Kruizinga. "Comparison of texture features based on Gabor filters." *IEEE Transactions on Image processing* 11.10 (2002): 1160-1167.

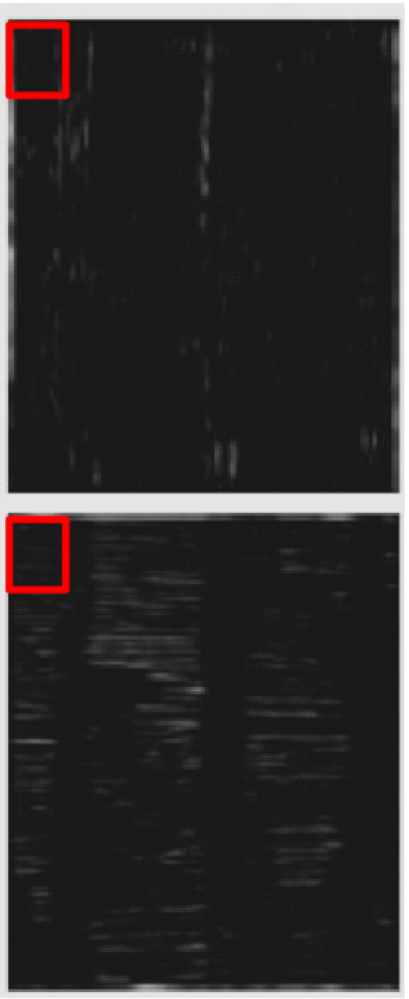
Texture Representation

- Textures are made up of repeated local patterns, so:
  - Find the patterns
    - Use filters that look like patterns (spots, bars, raw patches…)
    - Consider magnitude of response
  - Describe the response statistics within each local window
    - Mean, standard deviation,…
    - Histogram of “prototypical” feature occurrences

Texture Representation: example

original image





derivative filter responses, squared

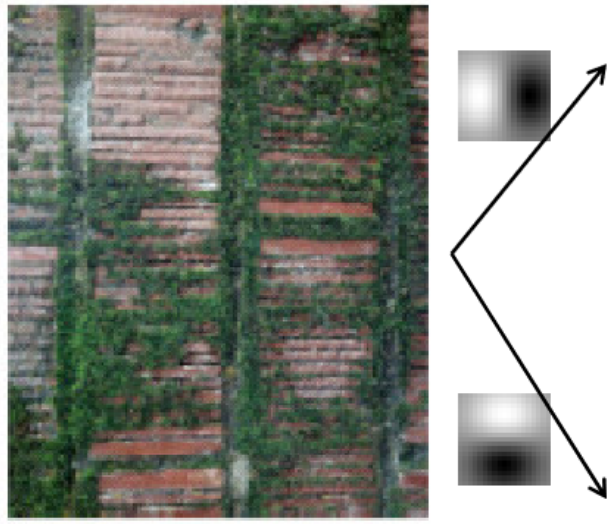
	mean d/dx value	mean d/dy value
Win. #1	4	10

statistics to summarize patterns in small windows

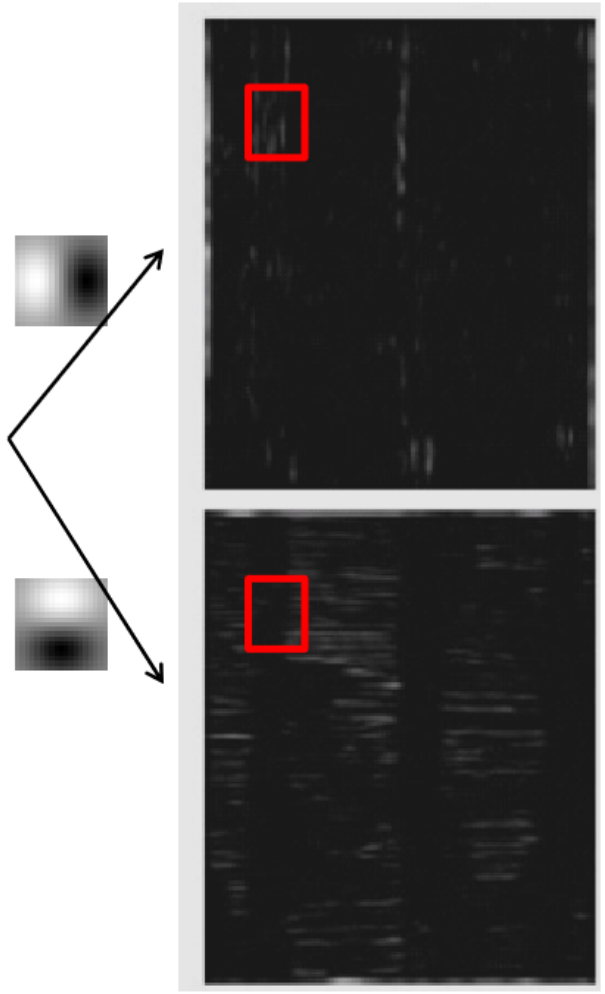
For simplicity, this example just uses 2 filters!

Slide: Trevor Darrell

Texture Representation: example



original image



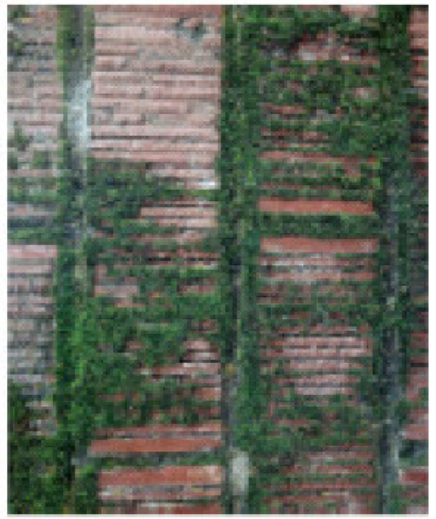
derivative filter responses, squared

	mean d/dx value	mean d/dy value
Win. #1	4	10
Win.#2	18	7	

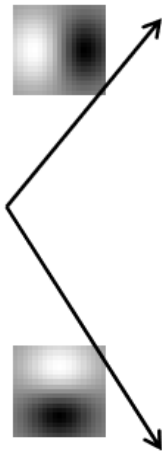
statistics to summarize patterns in small windows

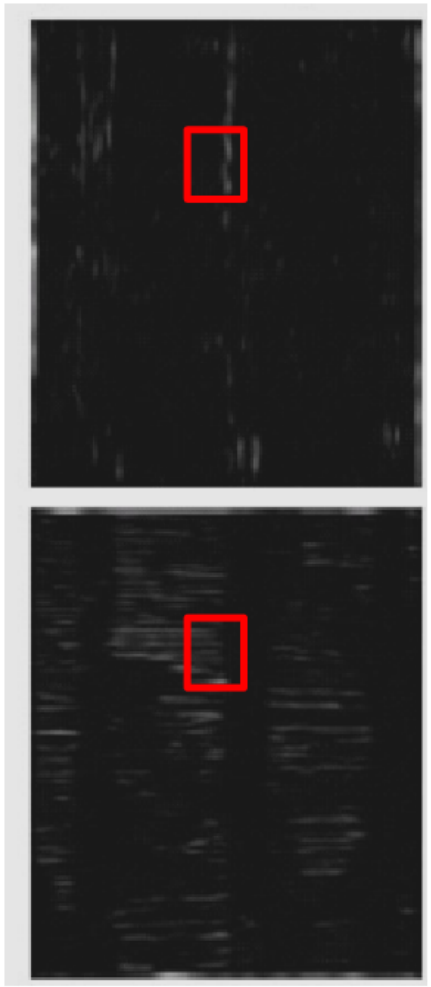
Slide: Trevor Darrell

Texture Representation: example



original image





derivative filter responses, squared

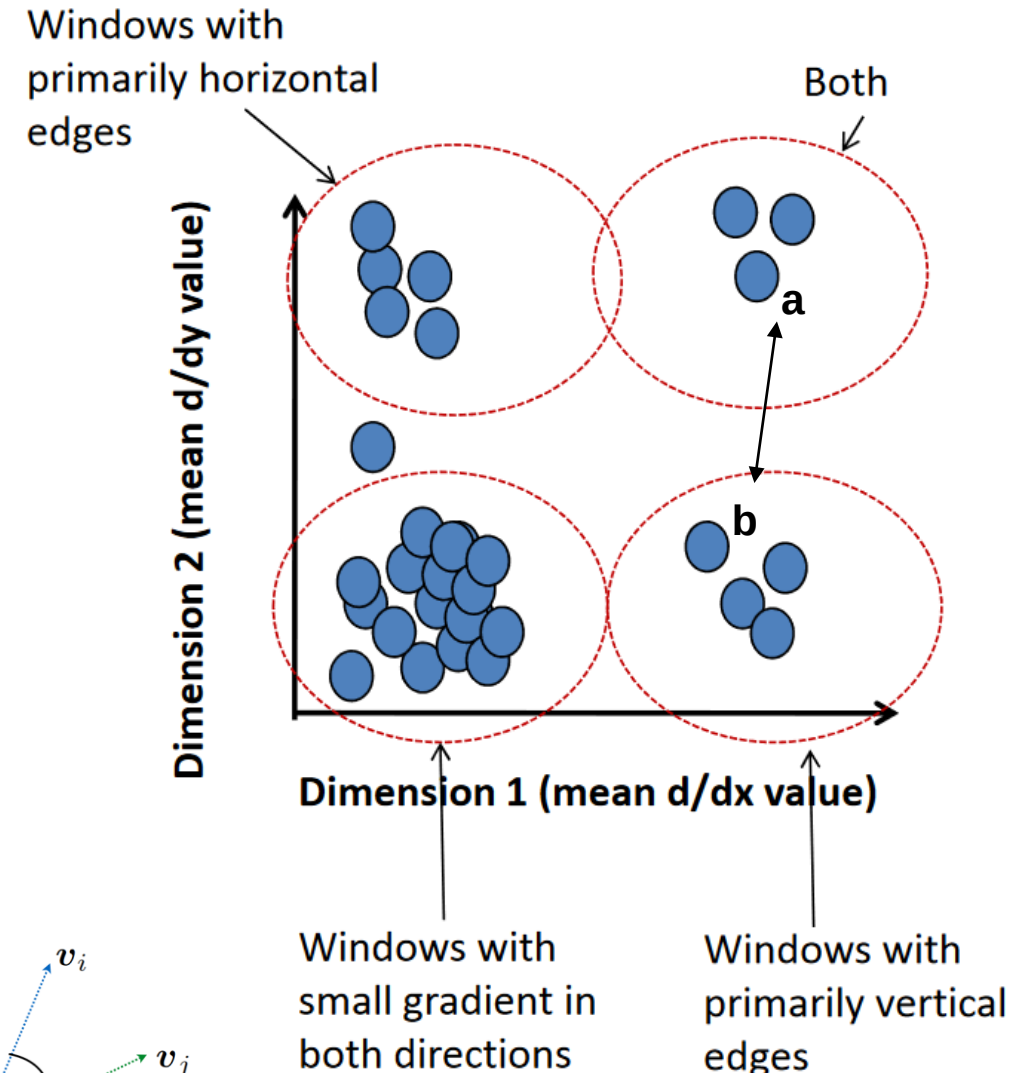
	mean d/dx value	mean d/dy value
Win. #1	4	10
Win.#2	18	7
⋮		
Win.#9	20	20

⋮

statistics to summarize patterns in small windows

Slide: Trevor Darrell

Texture Representation: example



	mean d/dx value	mean d/dy value
Win. #1	4	10
Win.#2	18	7
⋮		
Win.#9	20	20

⋮

statistics to summarize patterns in small windows

$$D(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

Distance reveals how dissimilar texture from window **a** is from texture in window **b**.

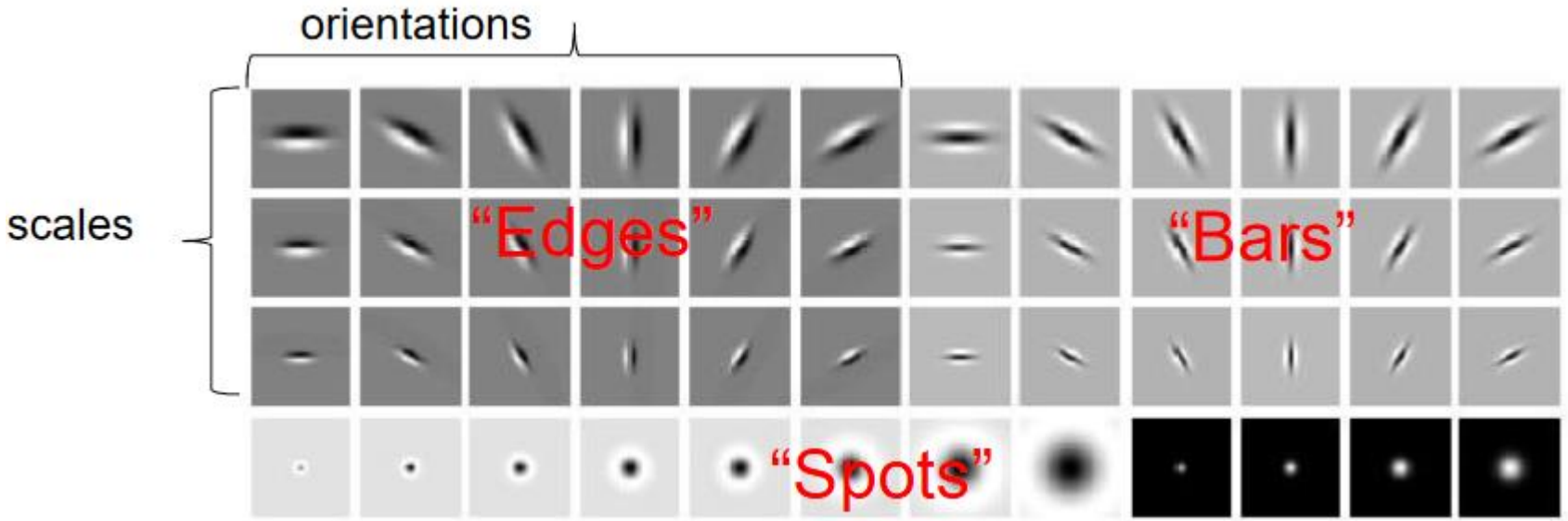
A good alternative to measure similarity is the cosine distance:

$$d(\boldsymbol{v}_i, \boldsymbol{v}_j) = \cos\theta = \frac{\boldsymbol{v}_i \cdot \boldsymbol{v}_j}{\|\boldsymbol{v}_i\|\|\boldsymbol{v}_j\|}$$

Slide: Kristen Grauman

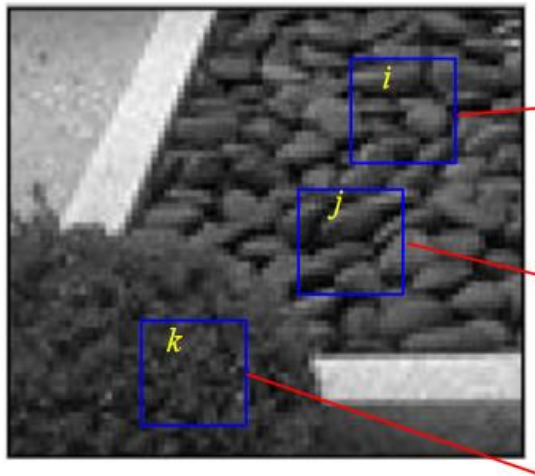
Texture Representation

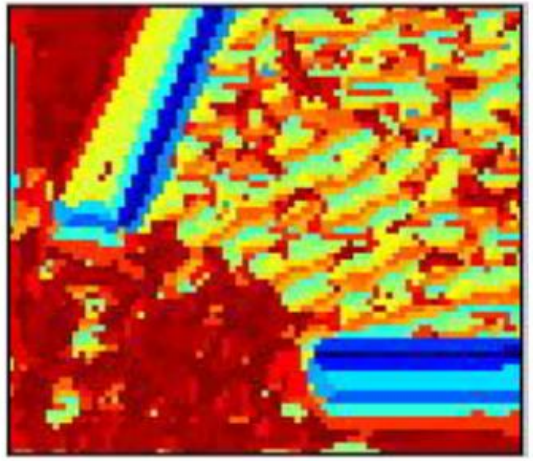
- The previous example used just two filters, but remember that we generalize this approach to apply a collection of multiples (*d*) filters: a “filter bank”.
  - Then our feature vectors will be *d*-dimensional.

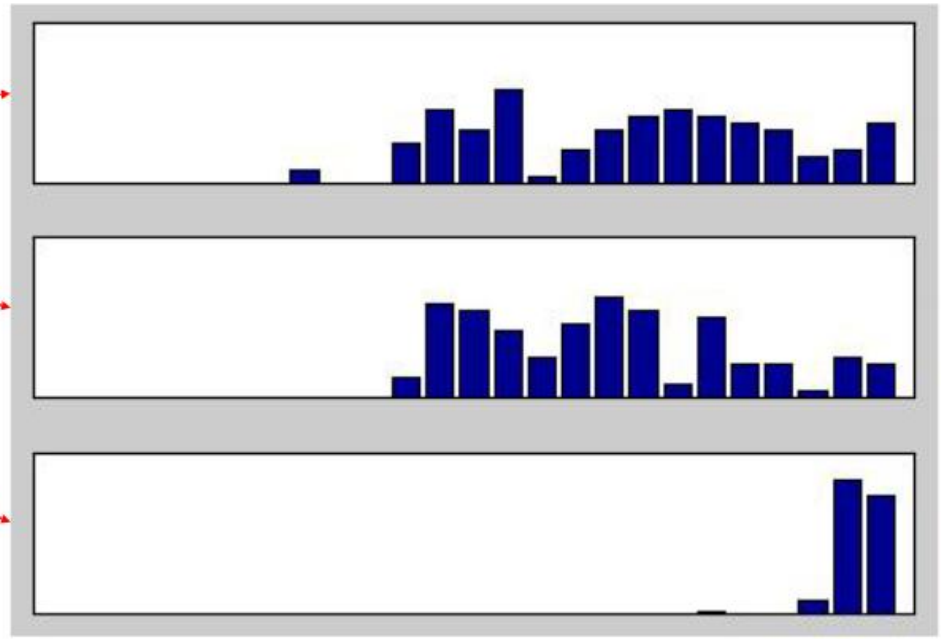


Slide: Kristen Grauman

Texture Representation







0.1

0.8

We can use a metric to measure distance from one patch to another, and see if they belong to the same texture

We can use these feature vectors to train a classifier (SVM, RF, MLP,…)

We have a *d*-dimensional feature vector for each window

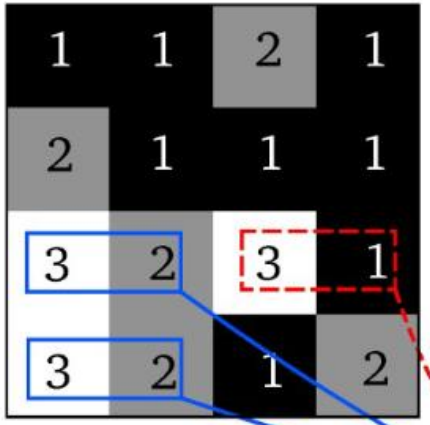
Note: this is related with the Bag-of-Words (BoW) model in computer vision that we'll see later!

Haralick Texture Features

- Haralick features are used to characterize textures.
  - Haralick, R.M., Shanmugam, K., and Dinstein, I., "Textural features for image classification." IEEE Transactions on systems, man, and cybernetics 6 (1973): 610-621.
- They are based on the Gray-Level Co-occurrence Matrix (GLCM).
  - This matrix records **how many times two gray-level pixels adjacent to each other appear**.
- Robert M. Haralick proposed 14 measures that can be extracted from the GLCM to quantify texture.

Haralick Texture Features

Image with numeric gray levels



Right neighbor GLCM

Reference pixel value (i) \ Neighbor pixel value (j)	1	2	3
1	3	2	0
2	3	0	1
3	1	2	0

Normalized GLCM p(i,j)

Reference pixel value (i) \ Neighbor pixel value (j)	1	2	3
1	0.25	0.17	0
2	0.25	0	0.08
3	0.08	0.17	0

Texture features

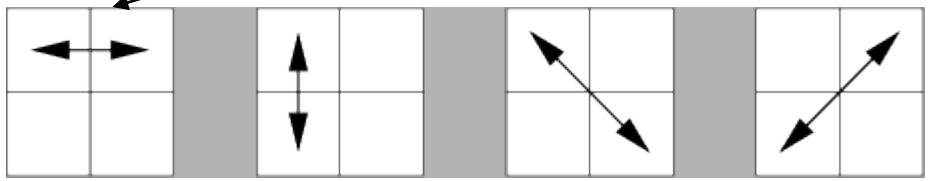
Homogeneity:

$$\sum_{ij} \frac{p(i,j)}{1+(i-j)^2}$$

Contrast:

$$\sum_{ij} p(i,j)\,(i-j)^2$$

In this example, we use a 4x4 input image (with just 3 possible gray-level values) and horizontal adjacency



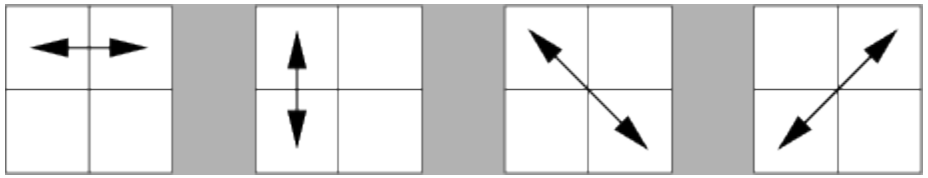
$p(i,j)$: Probability that a pixel with value i will be found adjacent to a pixel of value j

From the GLCM we compute 14 different statistics

Löfstedt, Tommy, et al. "Gray-level invariant Haralick texture features." *PloS one* 14.2 (2019): e0212110.

Haralick Texture Features

- The previous process is performed for the four directions in 2D.



- Therefore, we end with a 4x14 matrix:
  - 4 feature vectors (one row per direction)
  - each of 14-dimensionality

- Finally, we take the average of these directions to form a final feature vector of 14 elements.
  - This averaging is performed in an attempt to make the feature vector more robust to changes in rotation.

Angular Second Moment	$\sum_i \sum_j p(i,j)^2$
Contrast	$\sum_{n=0}^{N_g-1} n^2 \{\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j)\}, \lvert i-j \rvert = n$
Correlation	$\frac{\sum_i \sum_j (ij) p(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y}$ where μ_x , μ_y , σ_x , and σ_y are the means and std. deviations of p_x and p_y , the partial probability density functions
Sum of Squares: Variance	$\sum_i \sum_j (i-\mu)^2 p(i,j)$
Inverse Difference Moment	$\sum_i \sum_j \frac{1}{1+(i-j)^2} p(i,j)$
Sum Average	$\sum_{i=2}^{2N_g} i p_{x+y}(i)$ where x and y are the coordinates (row and column) of an entry in the co-occurrence matrix, and $p_{x+y}(i)$ is the probability of co-occurrence matrix coordinates summing to $x+y$
Sum Variance	$\sum_{i=2}^{2N_g} (i - f_8)^2 p_{x+y}(i)$
Sum Entropy	$-\sum_{i=2}^{2N_g} p_{x+y}(i) \log\{p_{x+y}(i)\} = f_8$
Entropy	$-\sum_i \sum_j p(i,j) log(p(i,j))$
Difference Variance	$\sum_{i=0}^{N_g-1} i^2 p_{x-y}(i)$
Difference Entropy	$-\sum_{i=0}^{N_g-1} p_{x-y}(i) \log\{p_{x-y}(i)\}$
Info. Measure of Correlation 1	$\frac{HXY - HXY1}{\max\{HX, HY\}}$
Info. Measure of Correlation 2	$(1 - \exp[-2(HXY2 - HXY)])^{\frac{1}{2}}$ where $HXY = -\sum_i \sum_j p(i,j) \log(p(i,j))$, HX , HY are the entropies of p_x and p_y , $HXY1 = -\sum_i \sum_j p(i,j) \log\{p_x(i) p_y(j)\}$ $HXY2 = -\sum_i \sum_j p_x(i) p_y(j) \log\{p_x(i) p_y(j)\}$
Max. Correlation Coeff.	Square root of the second largest eigenvalue of $\mathbf{Q}$ where $\mathbf{Q}(i,j) = \sum_k \frac{p(i,k) p(j,k)}{p_x(i) p_y(k)}$

https://murphylab.web.cmu.edu/publications/boland/boland_node26.html

Haralick Texture Features (technical note)

- Haralick texture features are implemented in the **`mahotas`** Python package

```
import mahotas
im=cv2.imread(get_image('zebra.jpg'),0)
features = mahotas.features.haralick(im).mean(axis=0)
```

This provides a 13-dimensional feature vector. The 14th feature is considered unstable and not implemented. See https://mahotas.readthedocs.io/en/latest/features.html

- **`scikit-image`** has also a GLCM texture feature implementation: https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_glcm.html

Local Binary Patterns

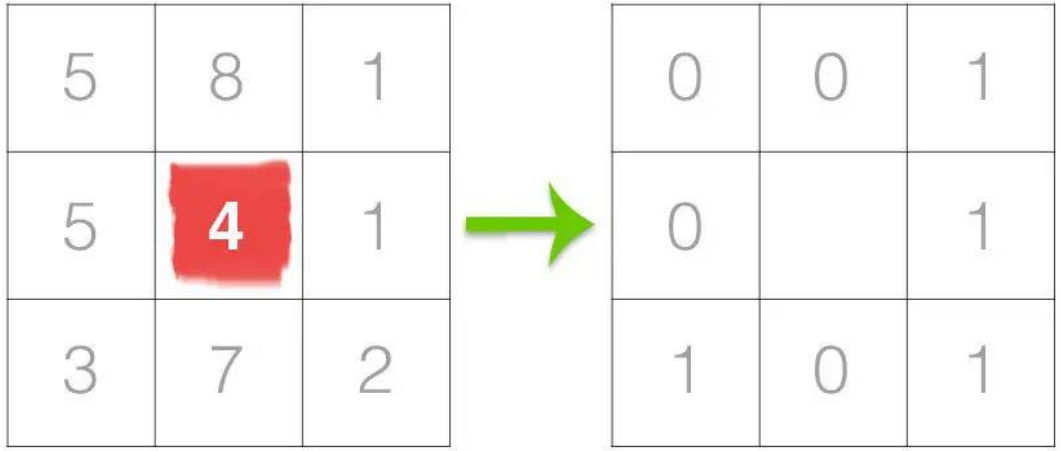
- Local Binary Patterns (LBP) are also used as texture descriptors.
  - Ojala, T., Pietikainen, M., & Maenpaa, T. (2002). *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns*. IEEE Trans. on pattern analysis and machine intelligence, 24(7), 971-987.

- Unlike Haralick features that compute a global representation of texture (based on the GLCM), LBPs instead compute a **local representation of texture**.

Visual examples of next slides taken from https://pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/

Local Binary Patterns

1st step:



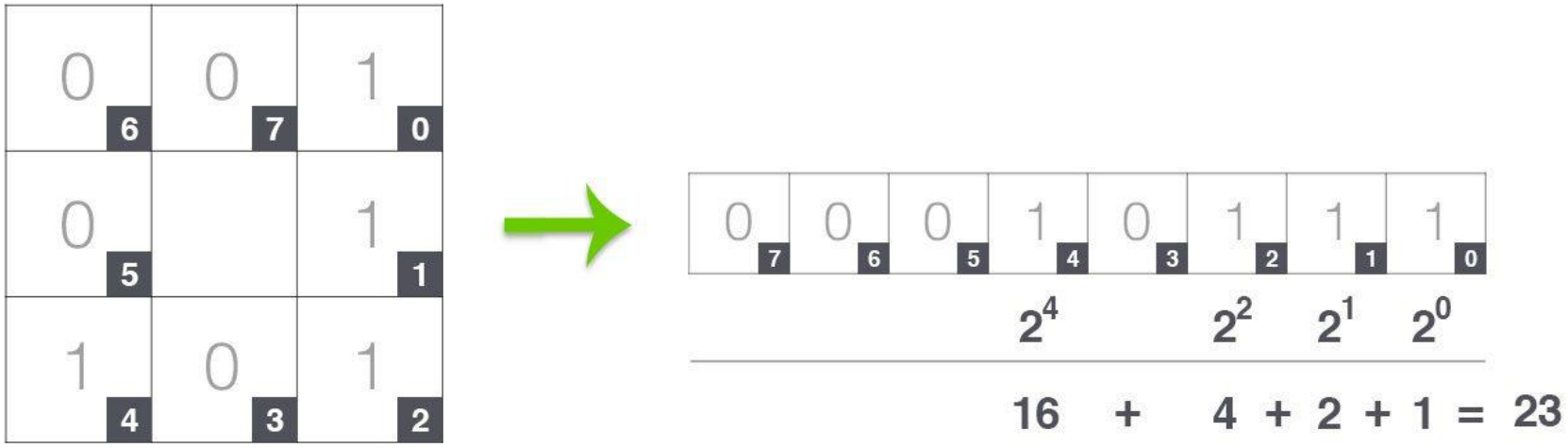
We operate in a local neighborhood (in this case, 3x3), and create a LBP following a very simple rule:

- If center pixel's value ≥ than its neighbors → we set value to 1.
- Otherwise, we set it to 0.

With 8 surrounding pixels, we have a total of $2^8 = 256$ possible combinations of LBP codes.

Local Binary Patterns

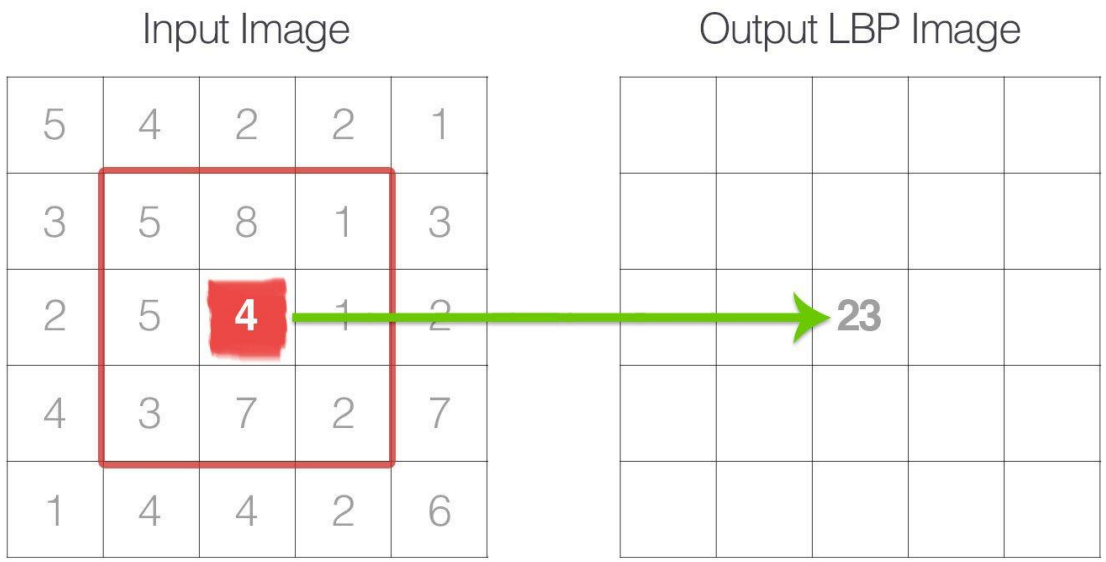
2nd step:



- We compute the LBP value for the center pixel.
  - We can start from any neighboring pixel and go clockwise or counter-clockwise, but our ordering must be kept consistent for all pixels in our image and all images in our dataset.

Local Binary Patterns

3rd step:



- This process of thresholding, accumulating binary strings, and storing the output decimal value in the LBP array is then repeated for each pixel in the input image.

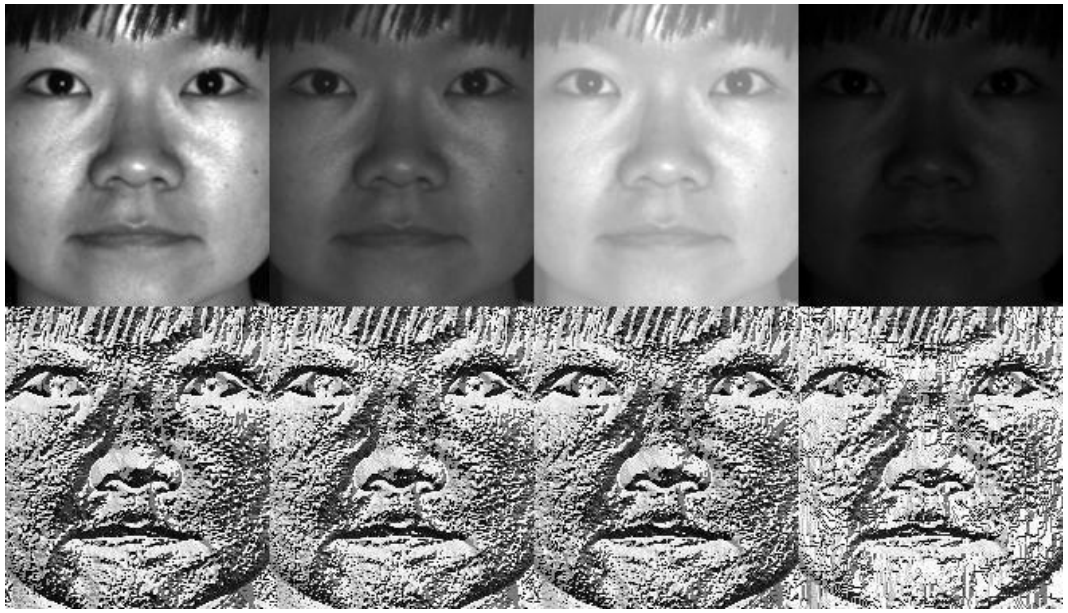
Local Binary Patterns

- These values go from 0 to 255, so we can easily visualize a full LBP 2D array:

This histogram can be our final (256-dimensional) feature vector and operate with it

Local Binary Patterns

- Main advantages:
  - robustness to gray-scale changes caused, for example, by illumination variations



Source: Ajmal Mian & Mehdi Ravanbkhsh

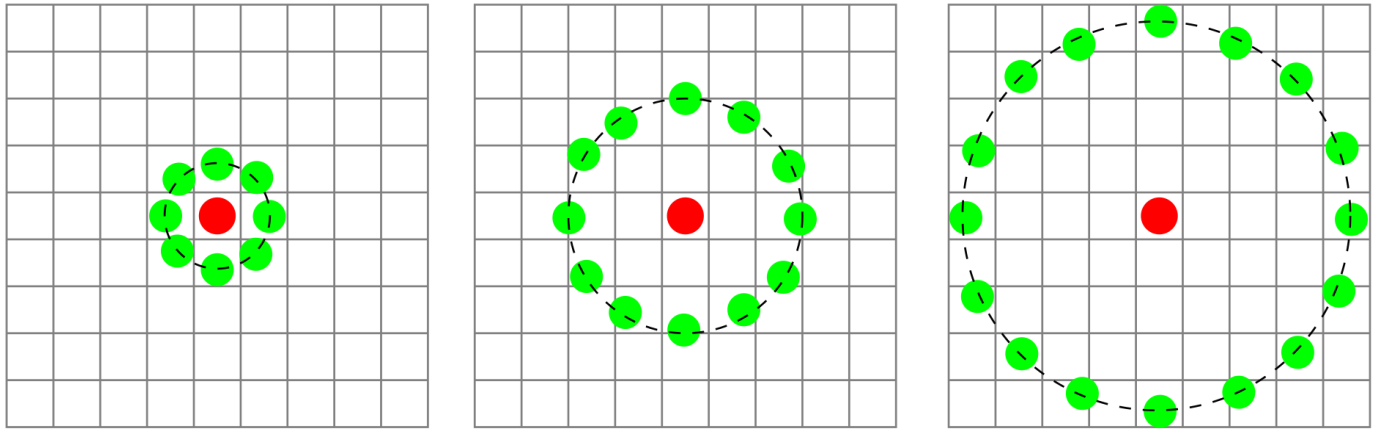
  - computational simplicity, which makes it possible to use it in real-time settings

Local Binary Patterns

- Many extensions and alternatives.
  - Bouwmans, T., Silva, C., Marghes, C., Zitouni, M. S., Bhaskar, H., & Frelicot, C. (2018). *On the role and the importance of features for background modeling and foreground detection*. Computer Science Review, 28, 26-91.
  - Liu, L., Chen, J., Fieguth, P., Zhao, G., Chellappa, R., & Pietikäinen, M. (2019). *From BoW to CNN: Two decades of texture representation for texture classification*. International Journal of Computer Vision, 127, 74-109.
  - For instance, different neighborhoods can be used to define a texture and calculate a local binary pattern (LBP)

This would allow us to capture details at varying scales. →

You can use a circular neighborhood and bilinearly interpolate values at non-integer pixel coordinates.



http://www.scholarpedia.org/article/Local_Binary_Patterns

Local Binary Patterns (technical note)

- LBPs are implemented in both the **`mahotas`** and the **`scikit-image`** Python packages:
  - https://mahotas.readthedocs.io/en/latest/lbp.html
  - https://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.local_binary_pattern

Handcrafted Feature Detection and Extraction

Pablo Mesejo

pmesejo@go.ugr.es

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE GRANADA

DaSCI

Instituto Andaluz de Investigación en **Data Science** and **Computational Intelligence**