THE UNIVERSITY OF BRITISH COLUMBIA

# CPSC 425: Computer Vision

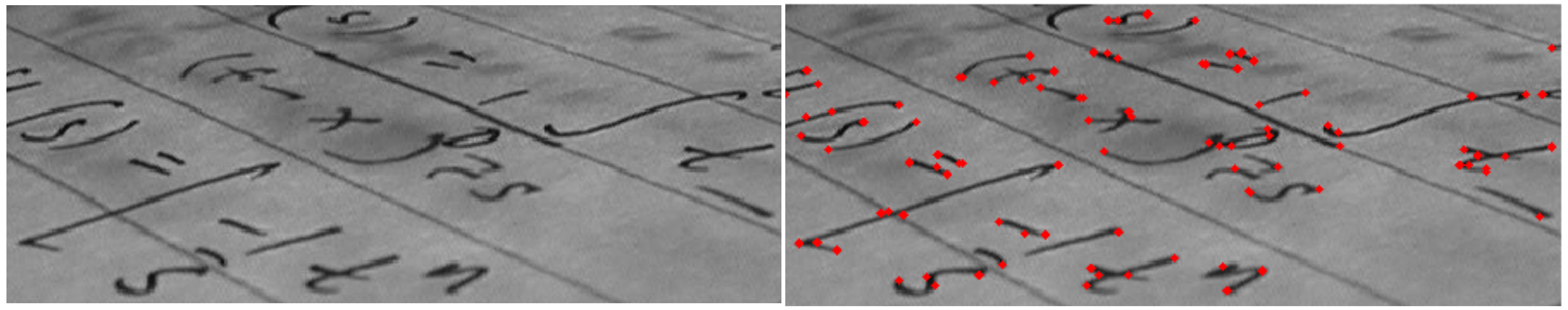


**Image Credit**: https://en.wikipedia.org/wiki/Corner_detection

## **Lecture 11:** Corner Detection (cont.)

( unless otherwise stated slides are taken or adopted from **Bob Woodham, Jim Little** and **Fred Tung** )

# **Menu** for Today (**October 10, 2024**)

## Topics:

- Harris **Corner** Detector (review)
- **Blob** Detection
- Searching over **Scale**
- **Texture** Synthesis & Analysis

## Readings:

- **Today's** Lecture: Forsyth & Ponce (2nd ed.) 5.3, 6.1, 6.3, 3.1-3.3

## Reminders:

- **Assignment 2**: Face Detection in a Scaled Representation is due **today**
- **Assignment 3:** Texture Synthesis is out **next Wednesday**
- (practice) **Quiz 1** and **Quiz 2** are out; **Quiz 3** will be out Monday

# **Menu** for Today (**October 10, 2024**)

## Topics:

- Harris **Corner** Detector (review)
- **Blob** Detection
- Searching over **Scale**
- **Texture** Synthesis & Analysis

## Readings:

- **Today's** Lecture: Forsyth & Ponce (2nd ed.) 5.3, 6.1, 6.3, 3.1-3.3

## Reminders:

- Study questions for **Midterm** will be up on Canvas over the weekend
- **Extra office hours** next week (Friday)
- **Review lecture** next **Thursday**

# Today's "**fun**" Example: Texture Camouflage

https://en.wikipedia.org/wiki/File:Camouflage.jpg

# Today's "**fun**" Example: Texture Camouflage

Cuttlefish on gravel seabed

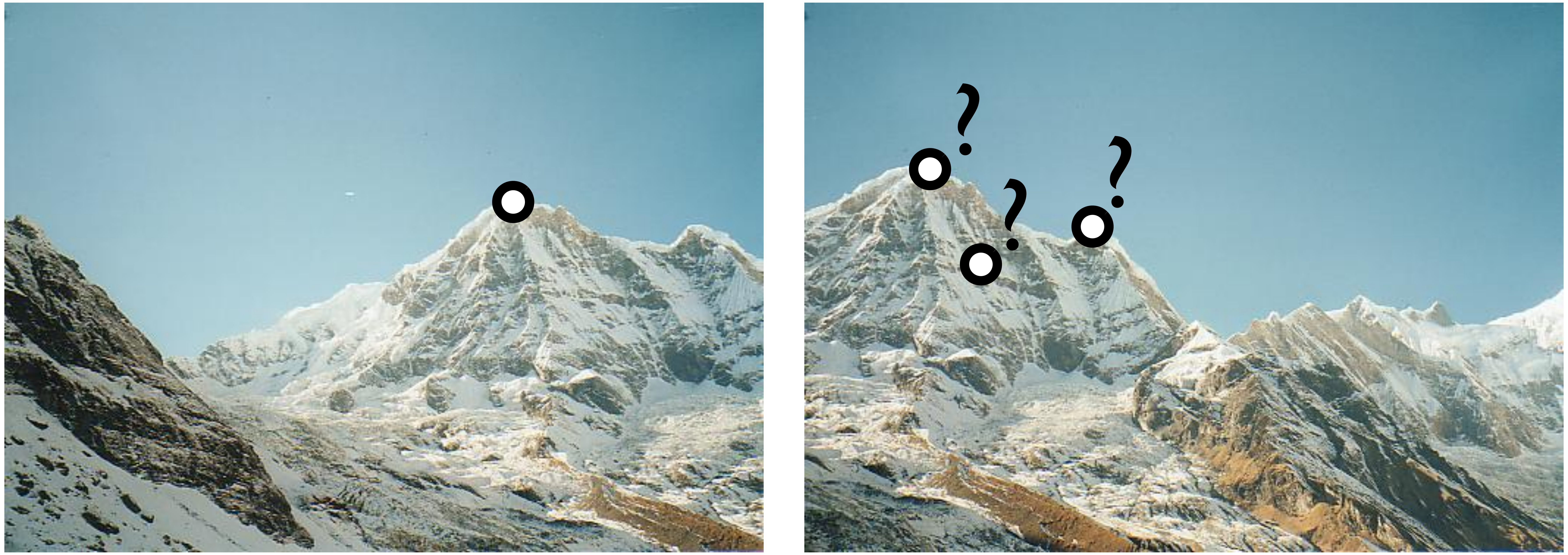
Seconds later…

http://www.marinet.org.uk/campaign-article/an-illustrated-guide-to-uk-marine-animals

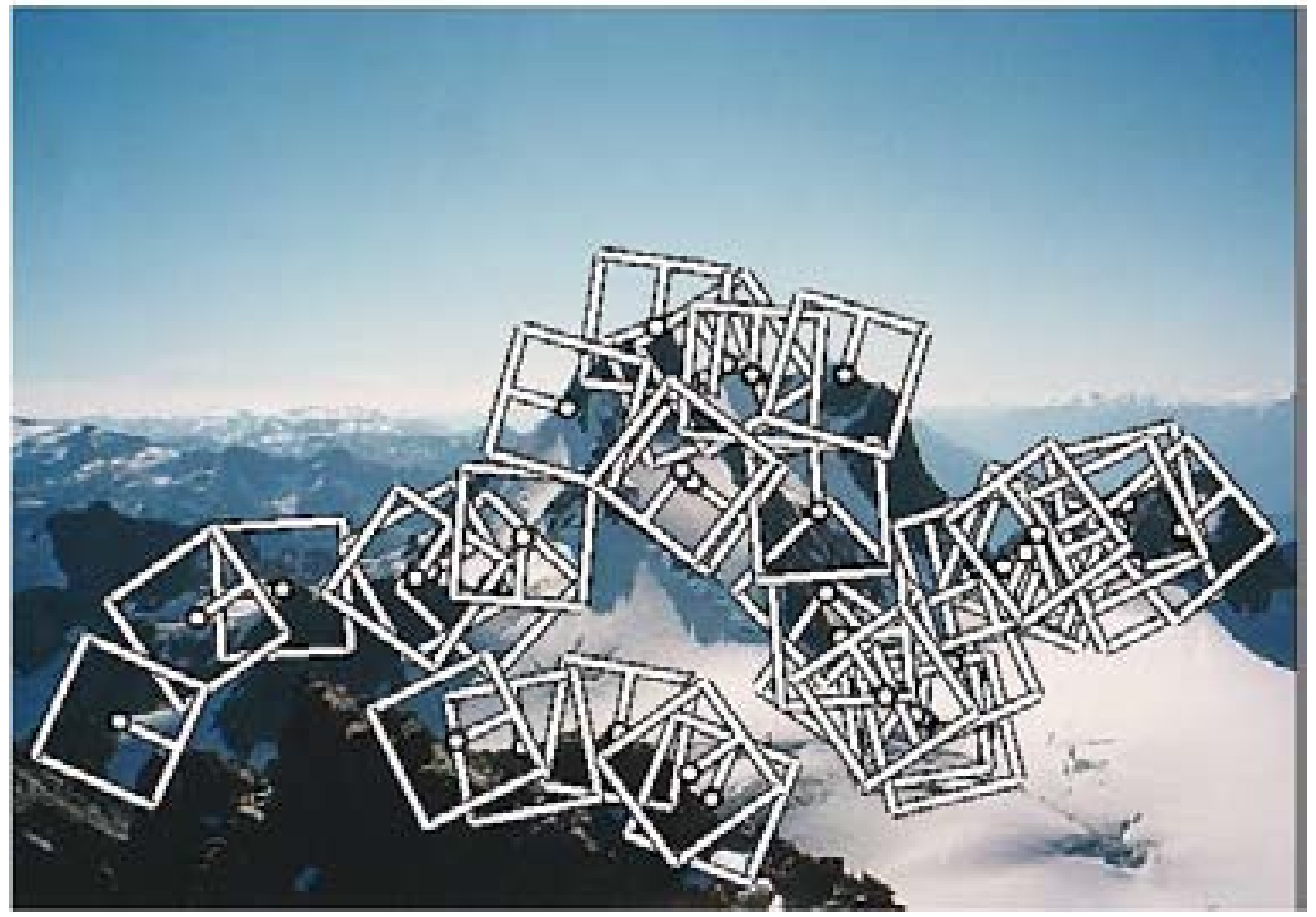
# **Lecture** 10: Re-cap (**Correspondence** Problem)

A basic problem in Computer Vision is to establish matches (correspondences) between images

This has **many** applications: rigid/non-rigid tracking, object recognition, image registration, structure from motion, stereo...



# **Lecture** 10: Re-cap (Feature **Detectors** [last time and today])



Corners/Blobs


Regions

Edges

Straight Lines

# **Lecture** 10: Re-cap (Feature **Descriptors** [later — after midterm])

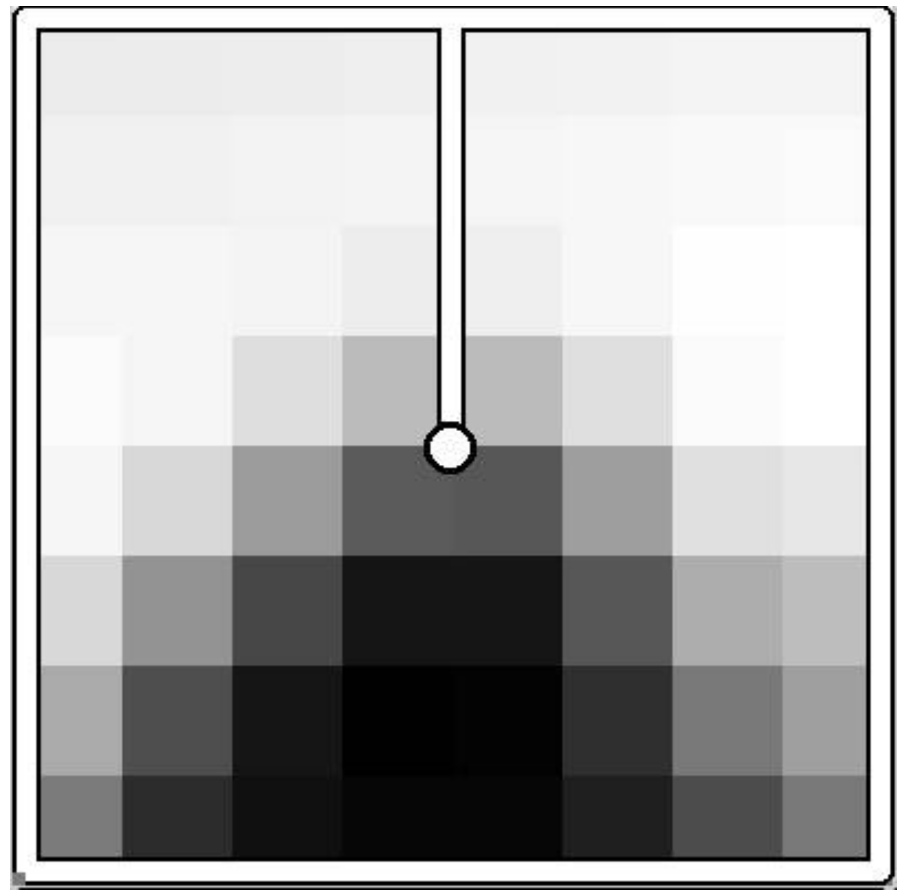
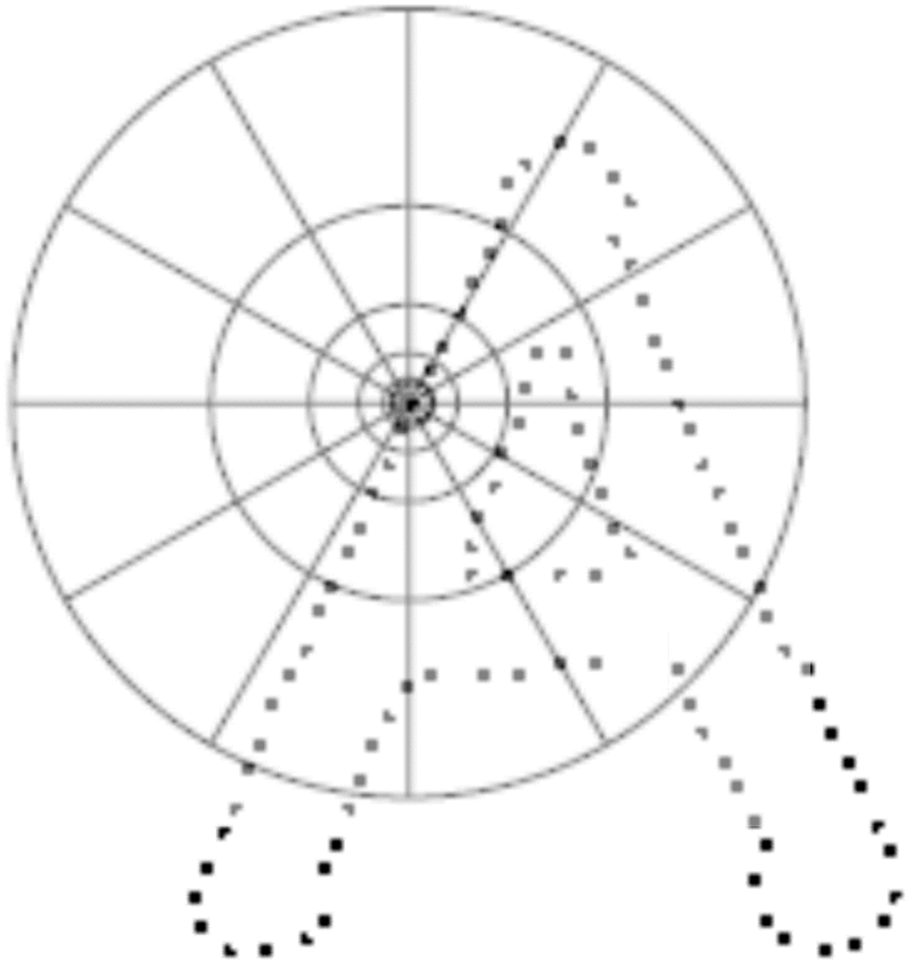
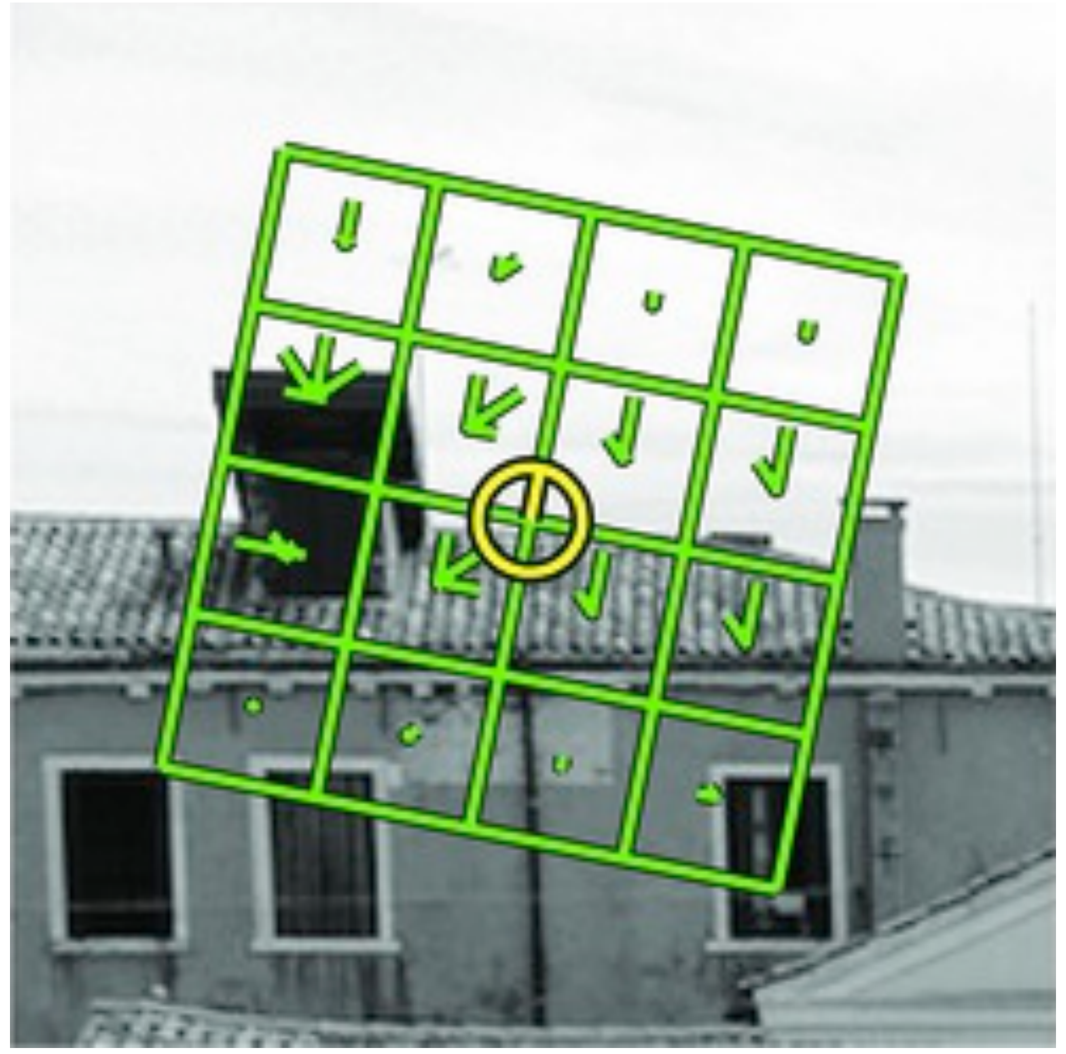


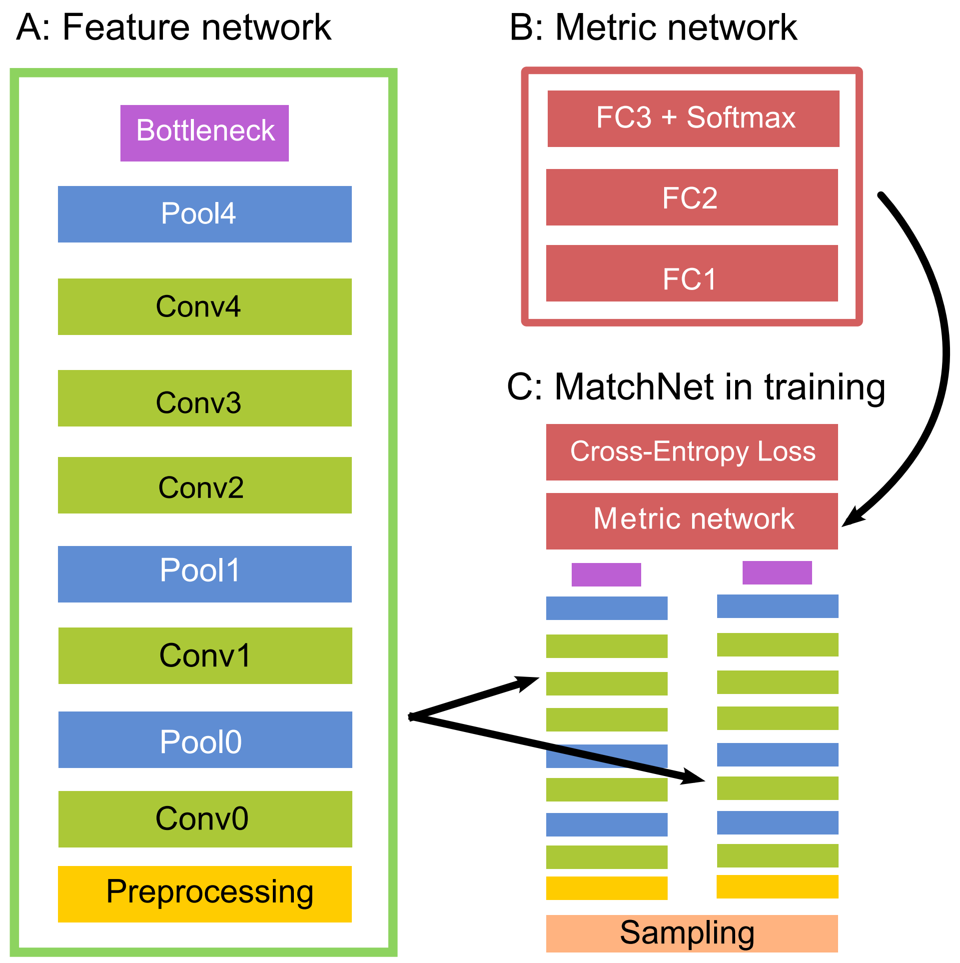
Image Patch



Shape Context



SIFT




Learned Descriptors

# **Lecture** 10: Re-cap (General **Setup**)

Use **small neighborhoods** of pixels to do **feature detection** — find locations in image that we MAY be able to match (sometimes this will also come with an estimate of the scale or canonical orientation of the feature)

# **Lecture** 10: Re-cap (General **Setup**)

Use **small neighborhoods** of pixels to do **feature detection** — find locations in image that we MAY be able to match (sometimes this will also come with an estimate of the scale or canonical orientation of the feature)

# **Lecture** 10: Re-cap (General **Setup**)

Use **small neighborhoods** of pixels to do **feature detection** — find locations in image that we MAY be able to match (sometimes this will also come with an estimate of the scale or canonical orientation of the feature)

# **Lecture** 10: Re-cap (General **Setup**)

Use **small neighborhoods** of pixels to do **feature detection** — find locations in image that we MAY be able to match (sometimes this will also come with an estimate of the scale or canonical orientation of the feature)

Use (typically larger neighborhoods) around the feature detections to characterize the region well, using a **feature descriptor**, in order to do matching (the scale and orientation, if available, will impact the region of descriptor)

# **Lecture** 10: Re-cap (General **Setup**)

Use **small neighborhoods** of pixels to do **feature detection** — find locations in image that we MAY be able to match (sometimes this will also come with an estimate of the scale or canonical orientation of the feature)

Use (typically larger neighborhoods) around the feature detections to characterize the region well, using a **feature descriptor**, in order to do matching (the scale and orientation, if available, will impact the region of descriptor)

# **Lecture** 10: Recap (What is a **Good Feature**?)

**Local**: features are local, robust to occlusion and clutter

**Accurate**: precise localization

**Robust**: noise, blur, compression, etc. do not have a big impact on the feature.

**Distinctive**: individual features can be easily matched

**Efficient**: close to real-time performance

# **Lecture** 10: Recap (What is a **Good Feature**?)

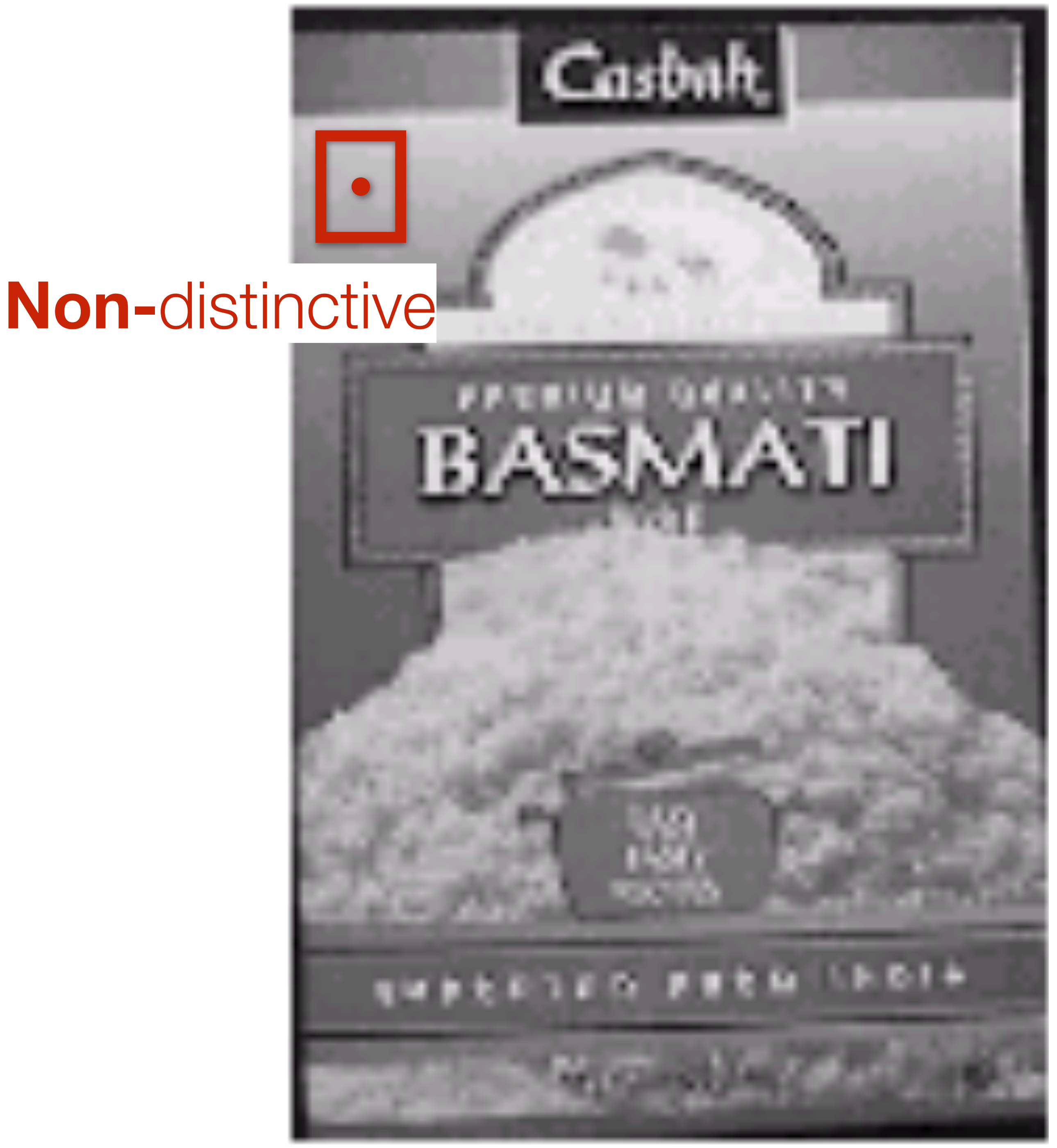
**Local**: features are local, robust to occlusion and clutter

**Accurate**: precise localization

**Robust**: noise, blur, compression, etc. do not have a big impact on the feature.

**Distinctive**: individual features can be easily matched

**Efficient**: close to real-time performance



# **Lecture** 10: Recap (What is a **Good Feature**?)

**Local**: features are local, robust to occlusion and clutter

**Accurate**: precise localization

**Robust**: noise, blur, compression, etc. do not have a big impact on the feature.

**Distinctive**: individual features can be easily matched

**Efficient**: close to real-time performance

# **Lecture** 10: Recap (What is a **Good Feature**?)

**Local**: features are local, robust to occlusion and clutter

**Accurate**: precise localization

**Robust**: noise, blur, compression, etc. do not have a big impact on the feature.

**Distinctive**: individual features can be easily matched

**Efficient**: close to real-time performance

# **Lecture** 10: Re-cap (Harris Corner Detection)

1. Compute image gradients over small region

2. Compute the covariance matrix

3. Compute eigenvectors and eigenvalues

4. Use threshold on eigenvalues to detect corners

$$I_x = \frac{\partial I}{\partial x}$$

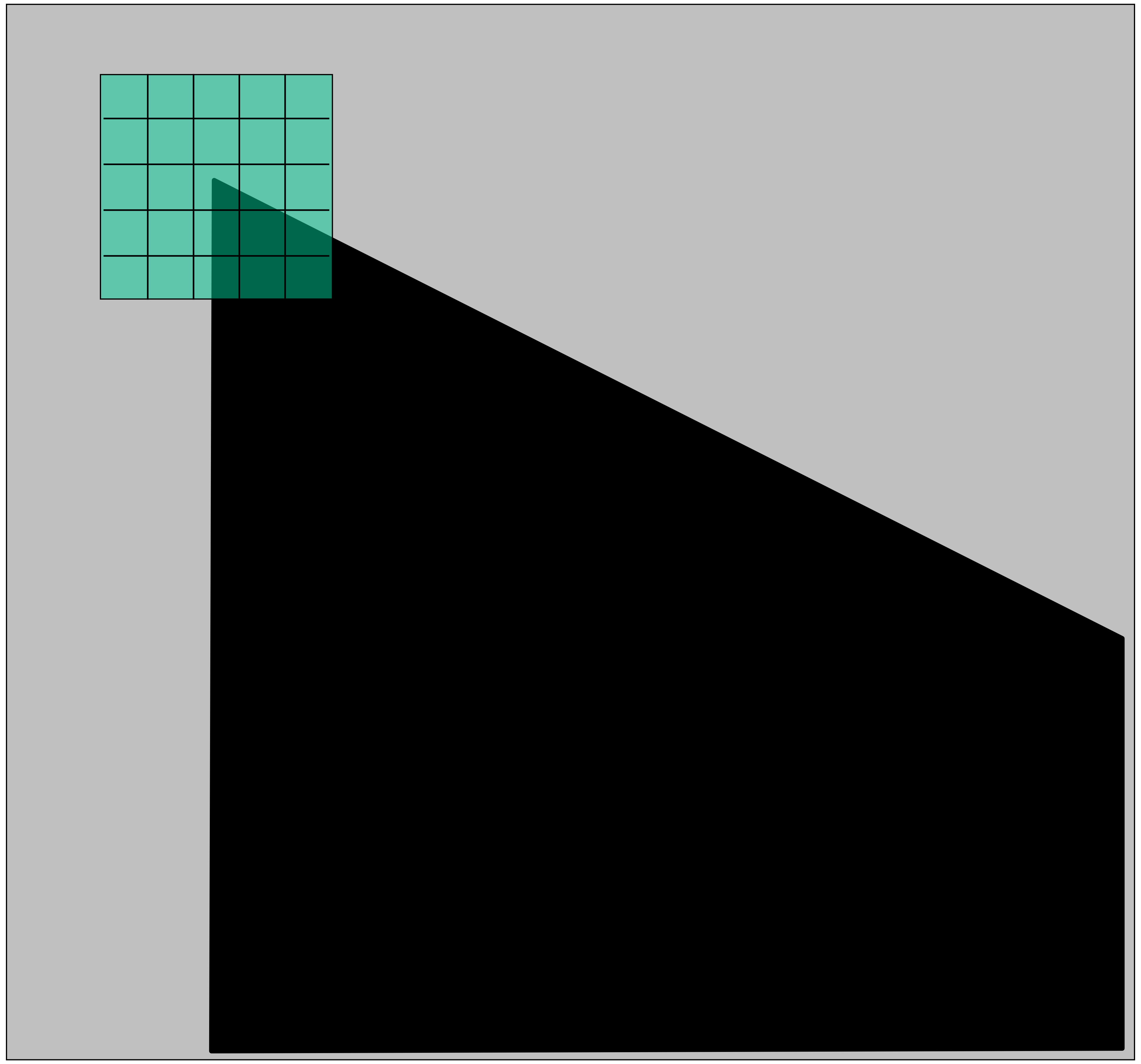
$$I_y = \frac{\partial I}{\partial y}$$

$$\begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

**Slide Adopted**: Ioannis (Yannis) Gkioulekas (CMU)

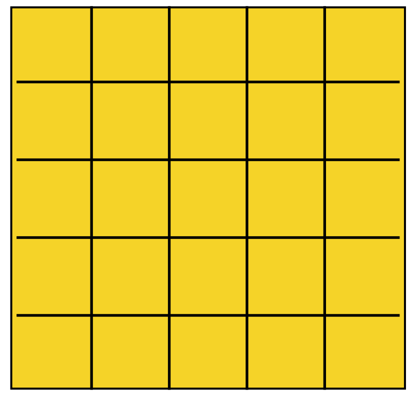
# **Lecture** 10: Re-cap (compute image gradients at patch)

(not just a single pixel)



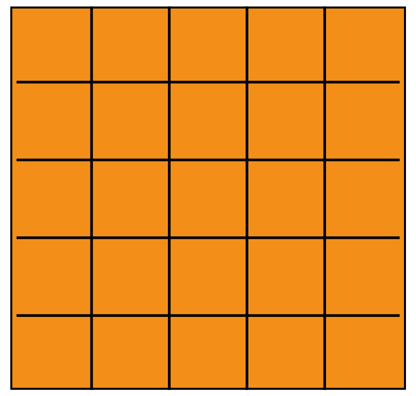
array of x gradients

$$I_x = \frac{\partial I}{\partial x}$$



array of y gradients

$$I_y = \frac{\partial I}{\partial y}$$



**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Lecture** 10: Re-cap (compute the covariance matrix)

**Sum** over small region around the corner

**Gradient** with respect to x, times gradient with respect to y

$$C = \begin{bmatrix} \sum\limits_{p \in P} I_x I_x & \sum\limits_{p \in P} I_x I_y \\ \sum\limits_{p \in P} I_y I_x & \sum\limits_{p \in P} I_y I_y \end{bmatrix}$$

Matrix is **symmetric**

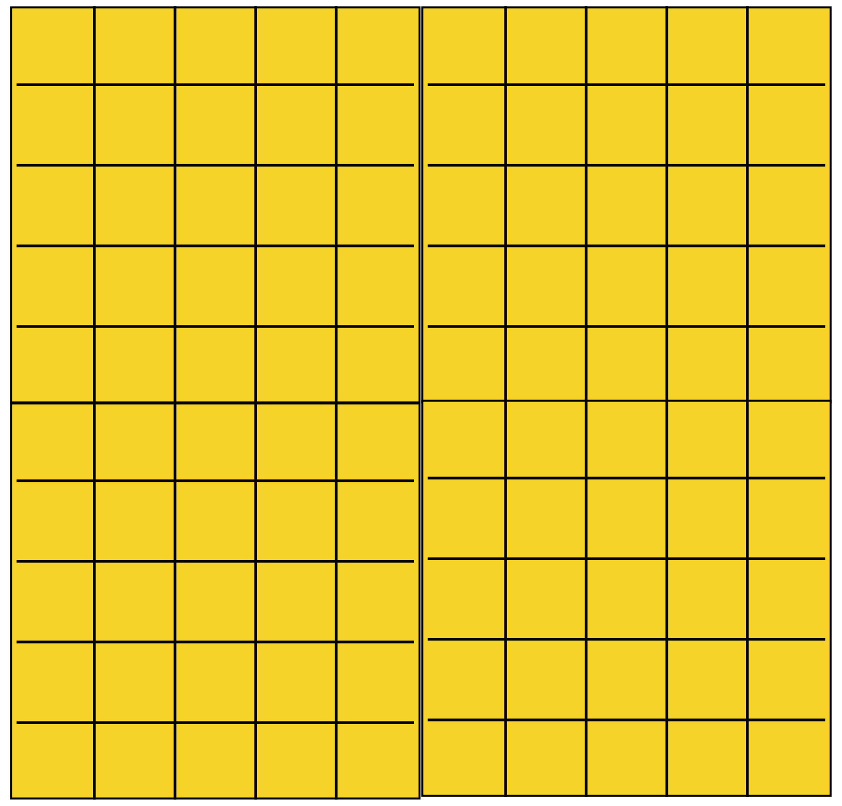
# Computing Covariance Matrix **Efficiently**

$$C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

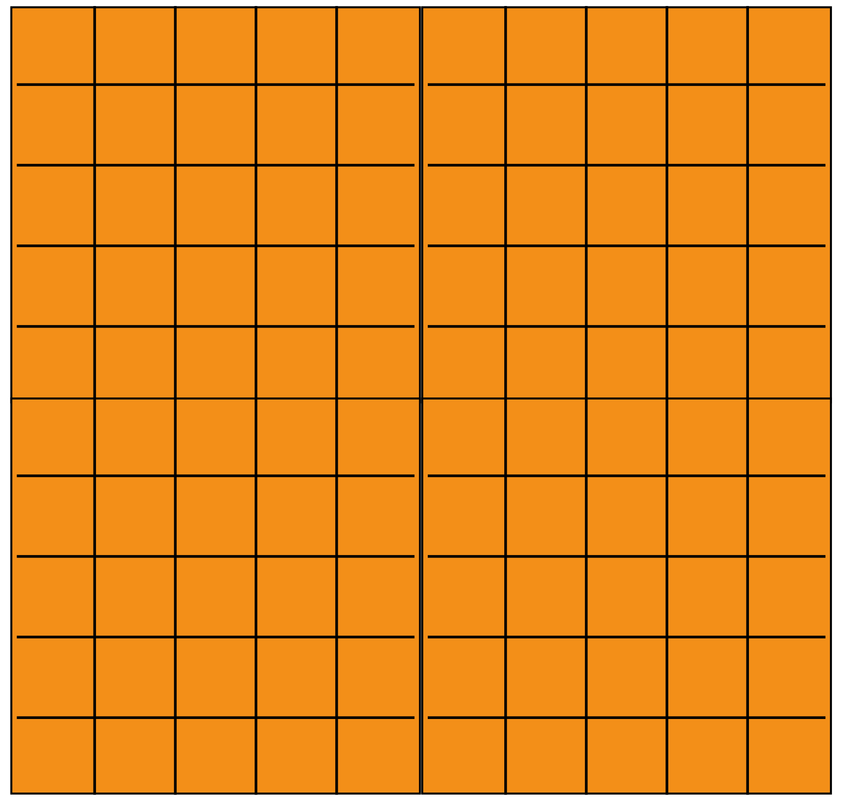
# Computing Covariance Matrix **Efficiently**

$$C = \begin{bmatrix} \sum\limits_{p \in P} I_x I_x & \sum\limits_{p \in P} I_x I_y \\ \sum\limits_{p \in P} I_y I_x & \sum\limits_{p \in P} I_y I_y \end{bmatrix}$$

$$I_x = \frac{\partial I}{\partial x}$$



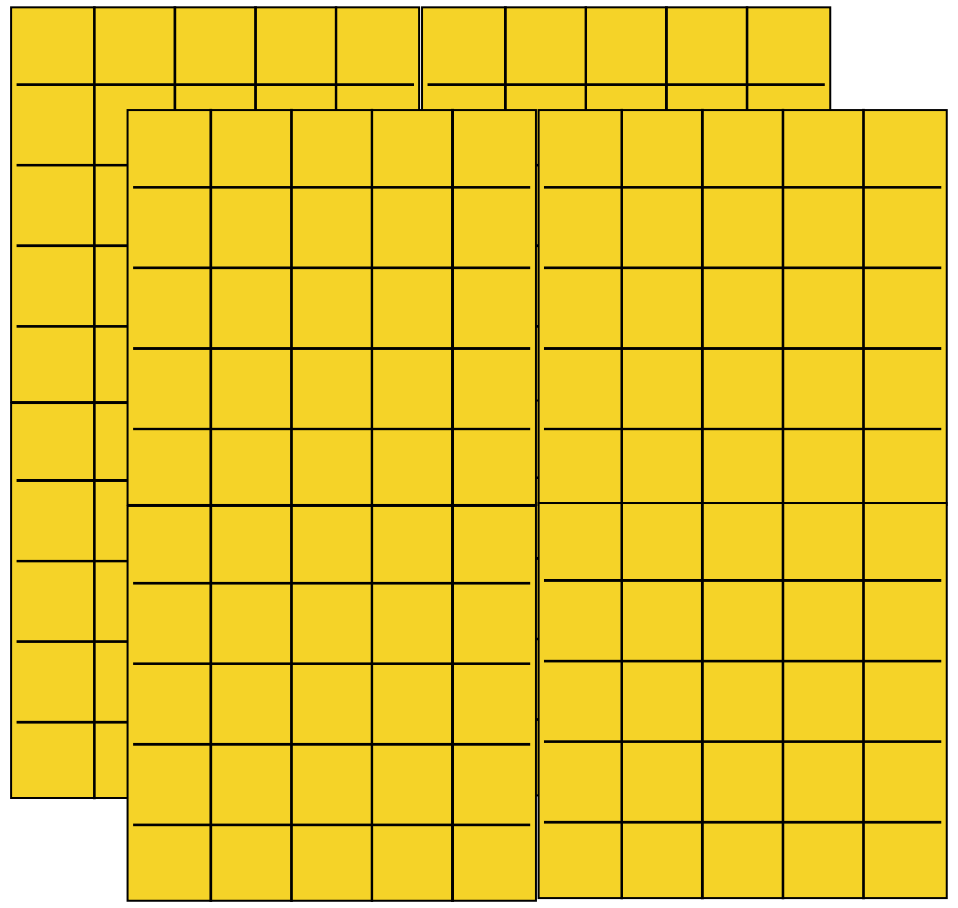
$$I_y = \frac{\partial I}{\partial y}$$



# Computing Covariance Matrix **Efficiently**

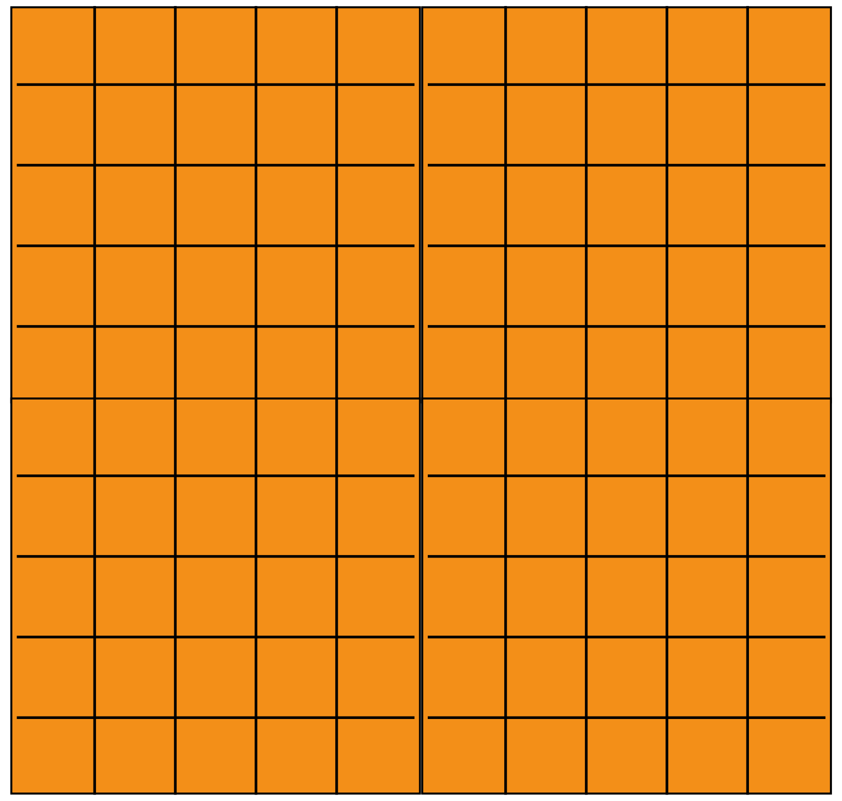
$$C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

$$I_x = \frac{\partial I}{\partial x}$$



$$I_x = \frac{\partial I}{\partial x}$$

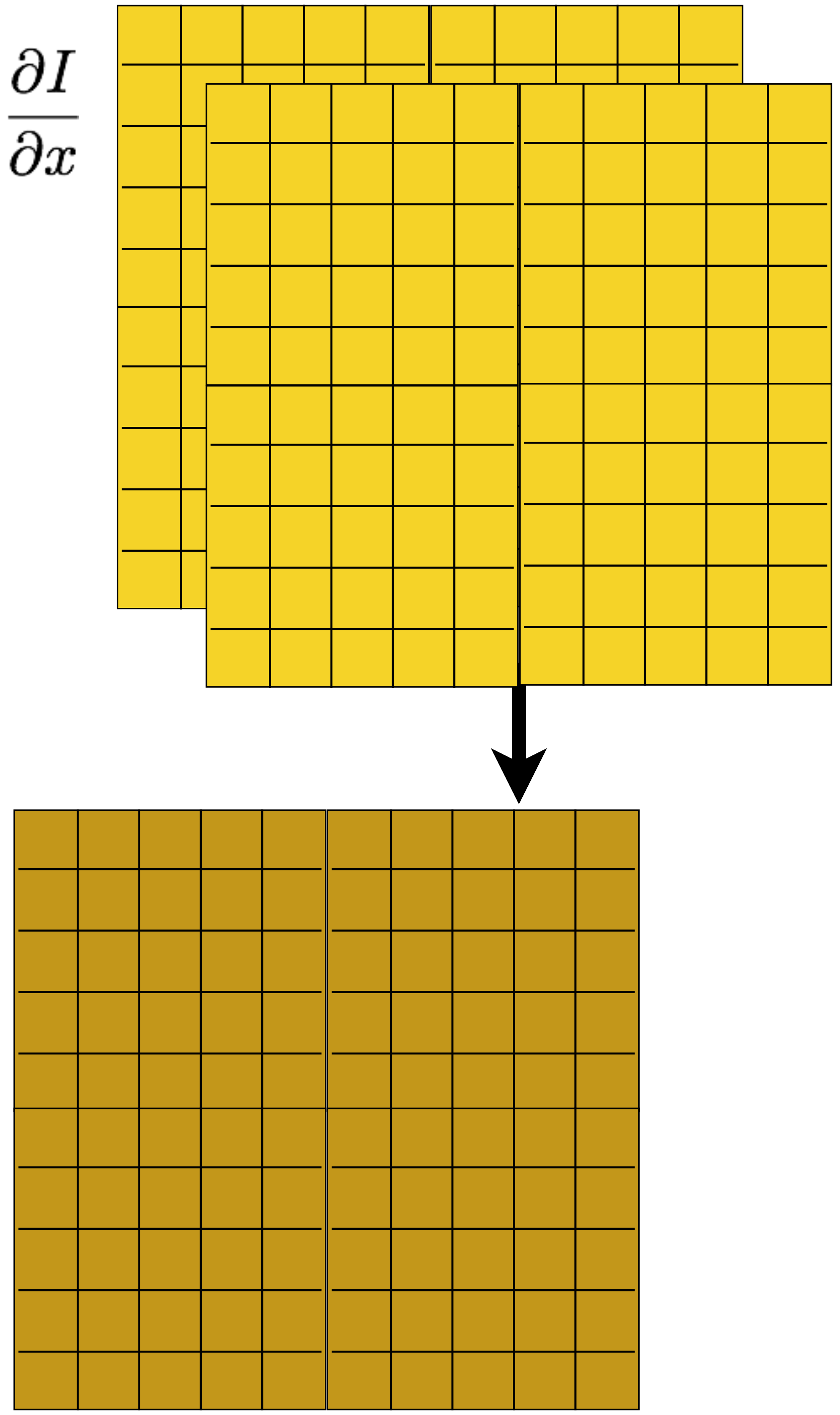
$$I_y = \frac{\partial I}{\partial y}$$



# Computing Covariance Matrix **Efficiently**

$$C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

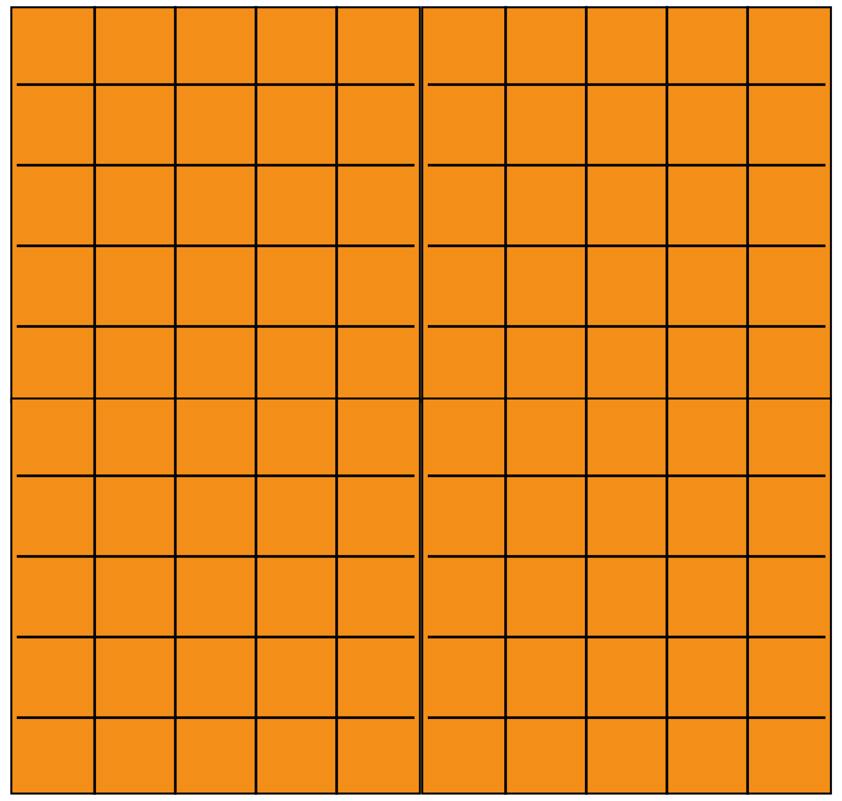
$I_x = \frac{\partial I}{\partial x}$



$I_x = \frac{\partial I}{\partial x}$

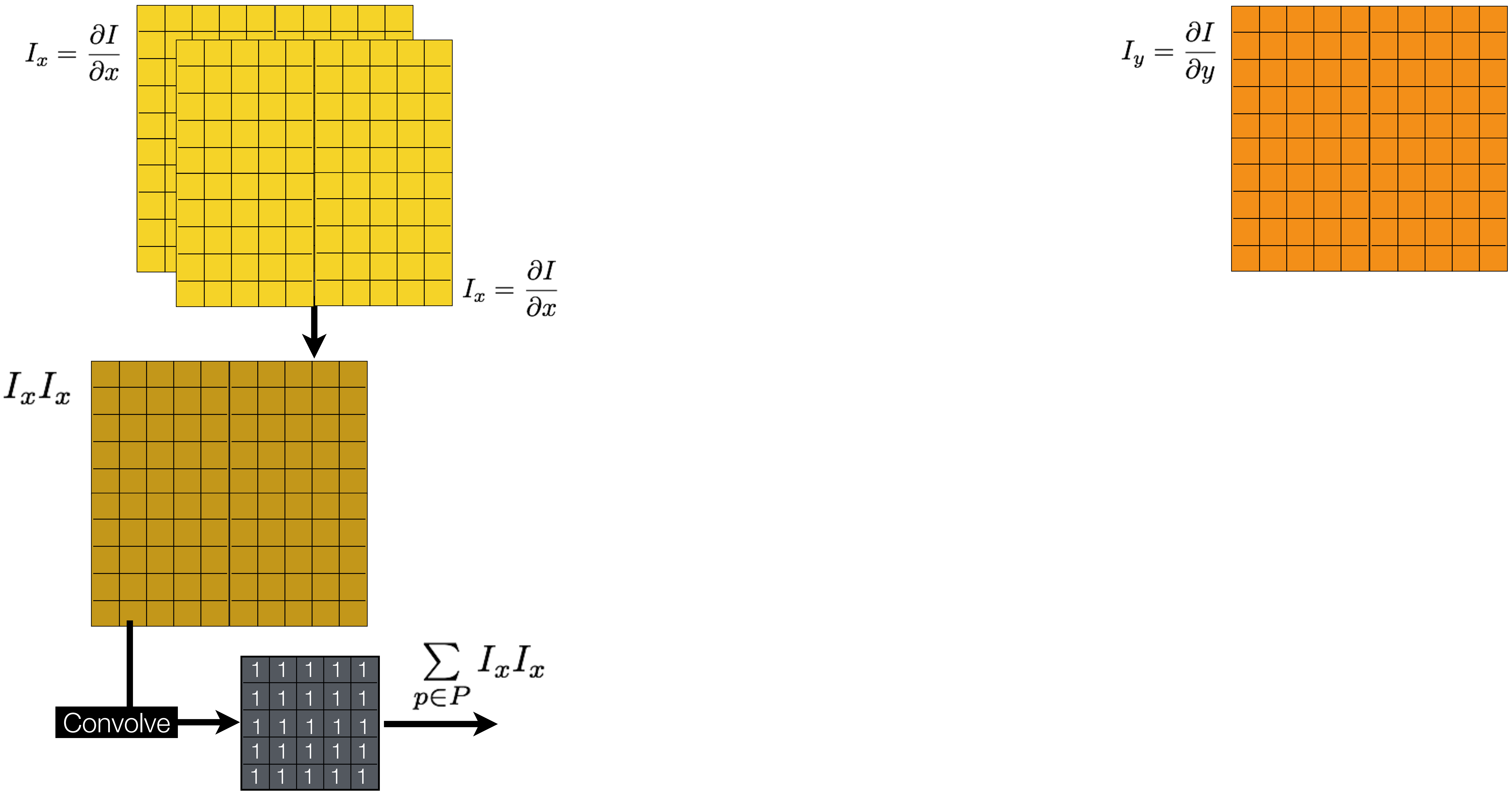
$I_x I_x$

$I_y = \frac{\partial I}{\partial y}$



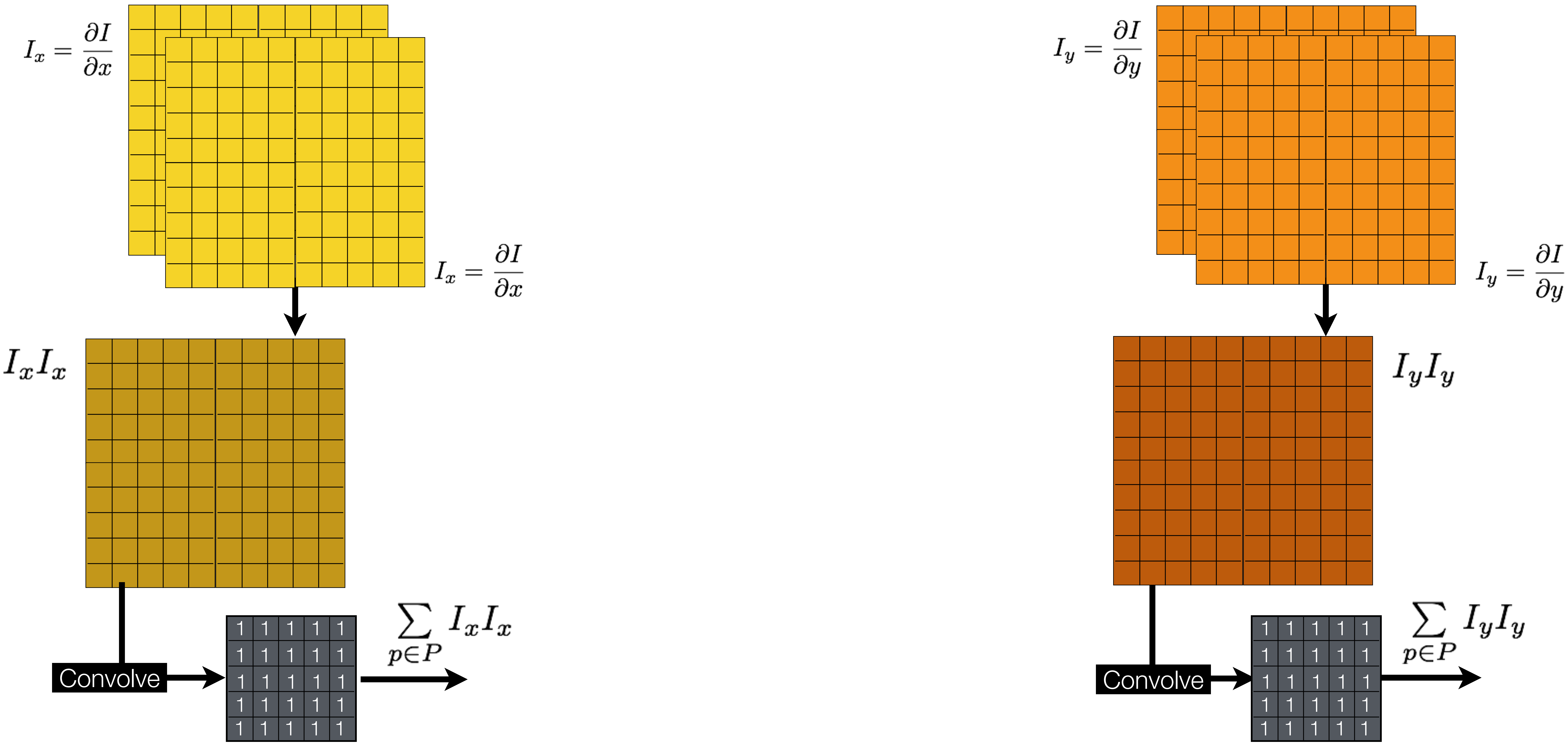
# Computing Covariance Matrix **Efficiently**

$$C = \begin{bmatrix} \sum\limits_{p \in P} I_x I_x & \sum\limits_{p \in P} I_x I_y \\ \sum\limits_{p \in P} I_y I_x & \sum\limits_{p \in P} I_y I_y \end{bmatrix}$$



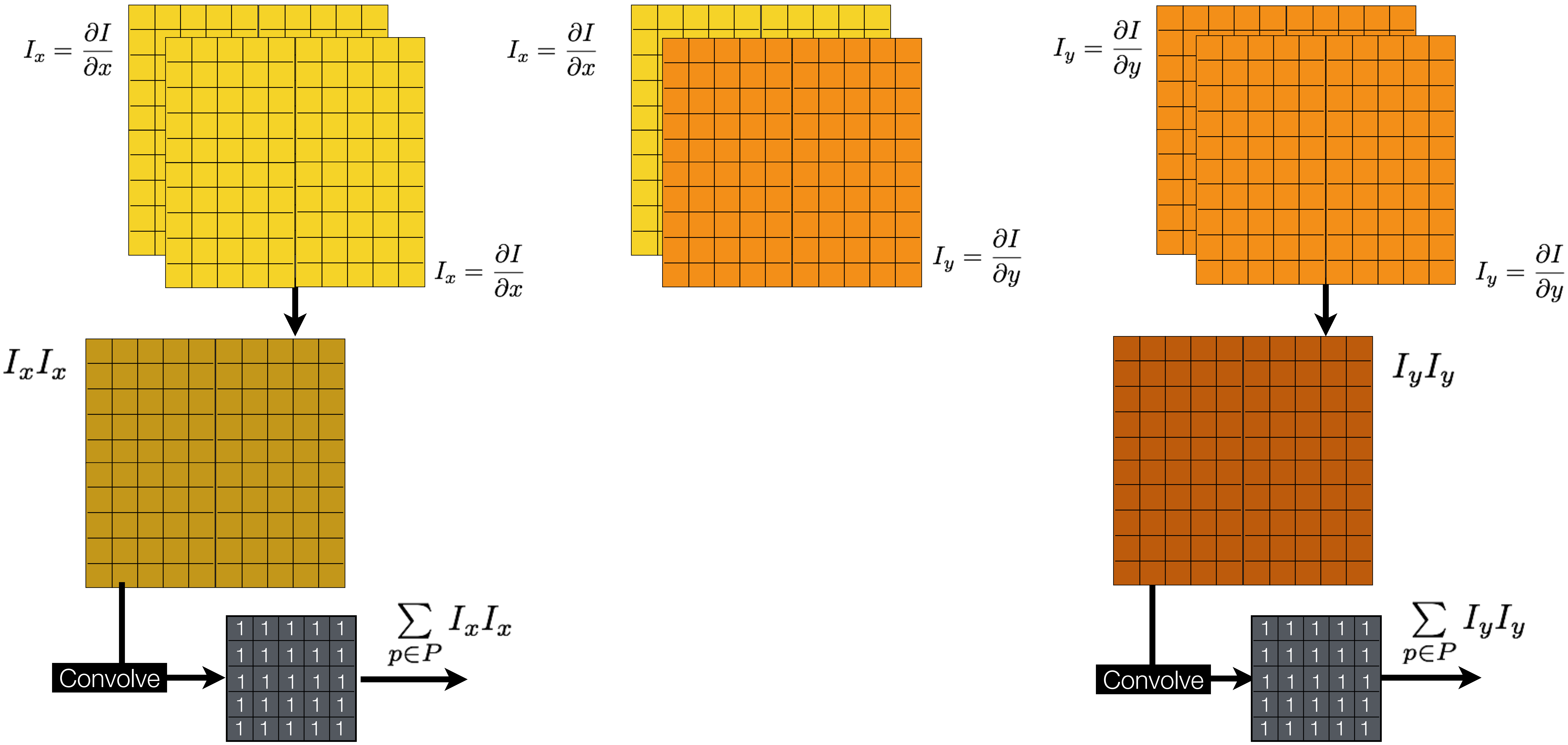
# Computing Covariance Matrix **Efficiently**

$$C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$



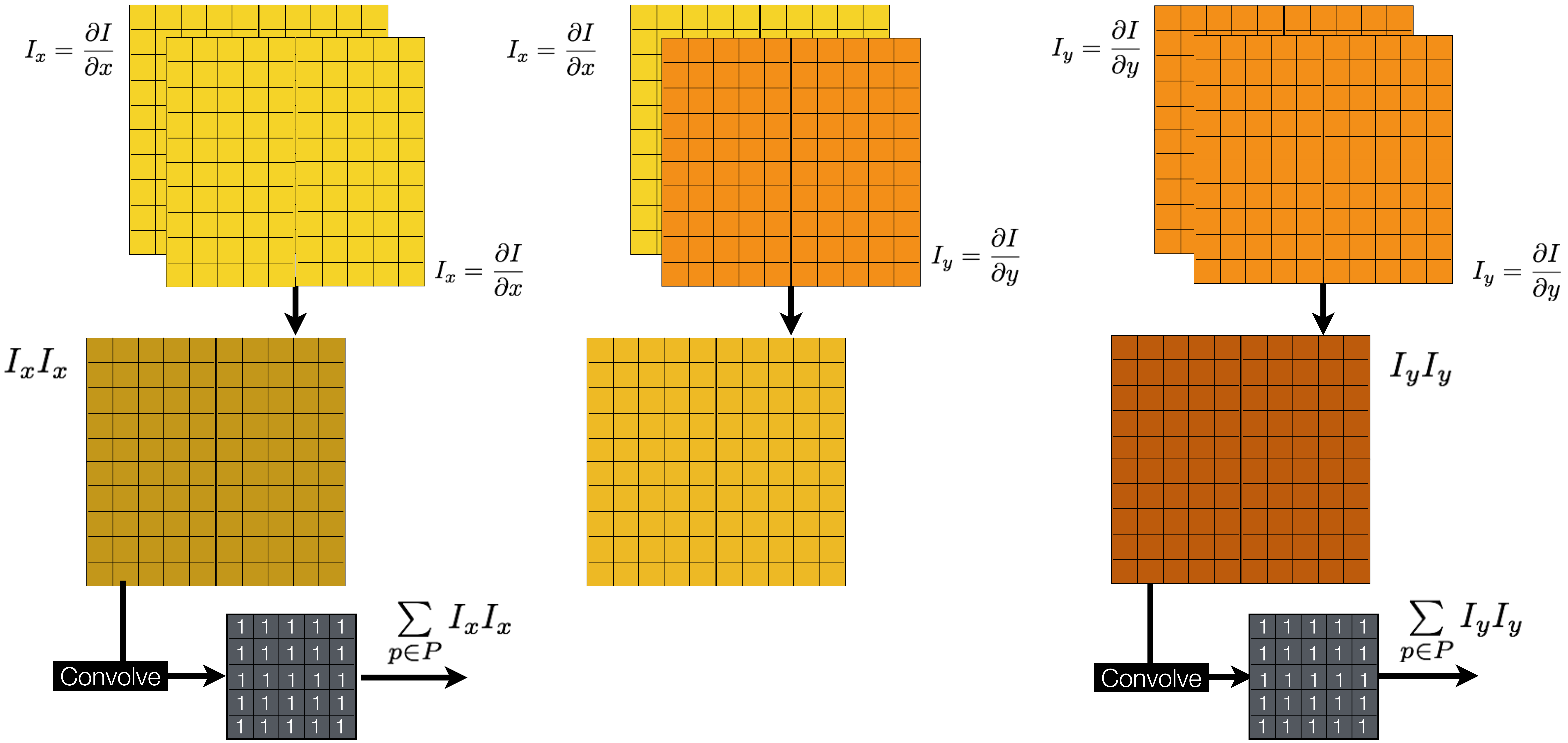
# Computing Covariance Matrix **Efficiently**

$$C = \begin{bmatrix} \sum_{p\in P} I_x I_x & \sum_{p\in P} I_x I_y \\ \sum_{p\in P} I_y I_x & \sum_{p\in P} I_y I_y \end{bmatrix}$$



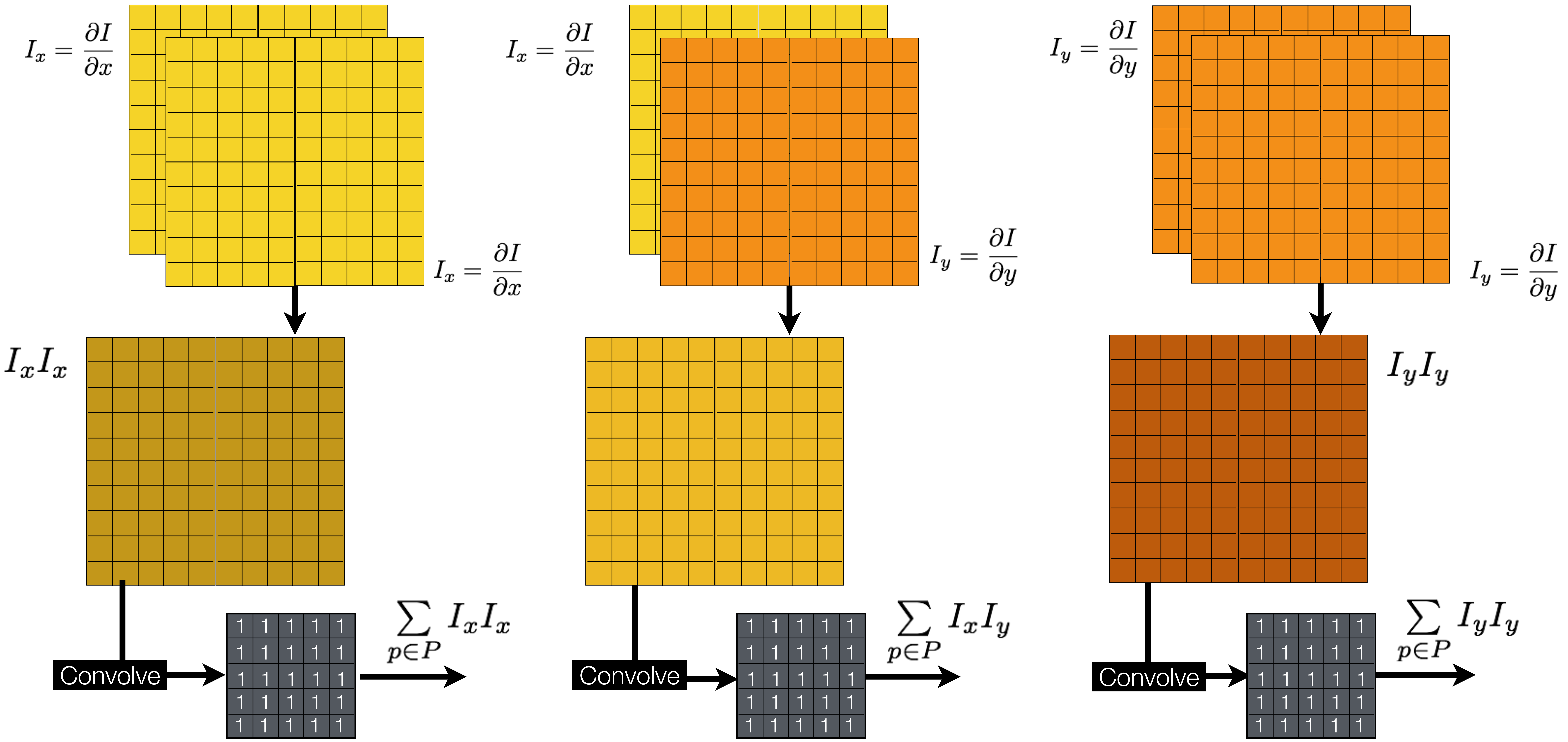
# Computing Covariance Matrix **Efficiently**

$$C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$



# Computing Covariance Matrix **Efficiently**

$$C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$



# **Lecture** 10: Re-cap

It can be shown that since every C is symmetric:

$$C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

# **Lecture** 10: Re-cap (computing eigenvalues and eigenvectors)

eigenvalue

$$Ce = \lambda e$$

eigenvector

$$(C - \lambda I)e = 0$$

| | |
|---|---|
| 1. Compute the determinant of (returns a polynomial) | $C - \lambda I$ |
| 2. Find the roots of polynomial (returns eigenvalues) | $\det(C - \lambda I) = 0$ |
| 3. For each eigenvalue, solve (returns eigenvectors) | $(C - \lambda I)e = 0$ |

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Lecture** 10: Re-cap (interpreting eigenvalues)

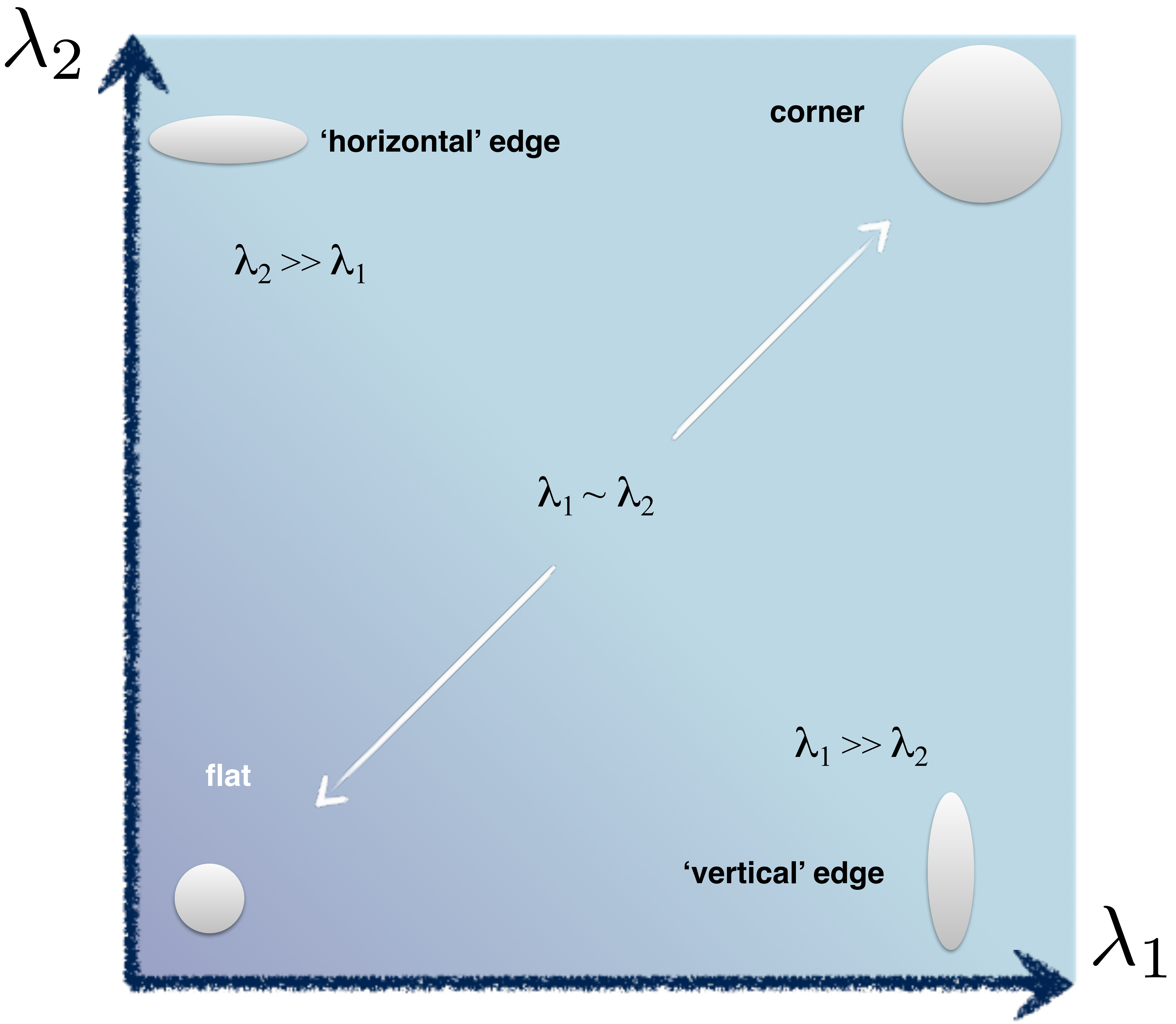
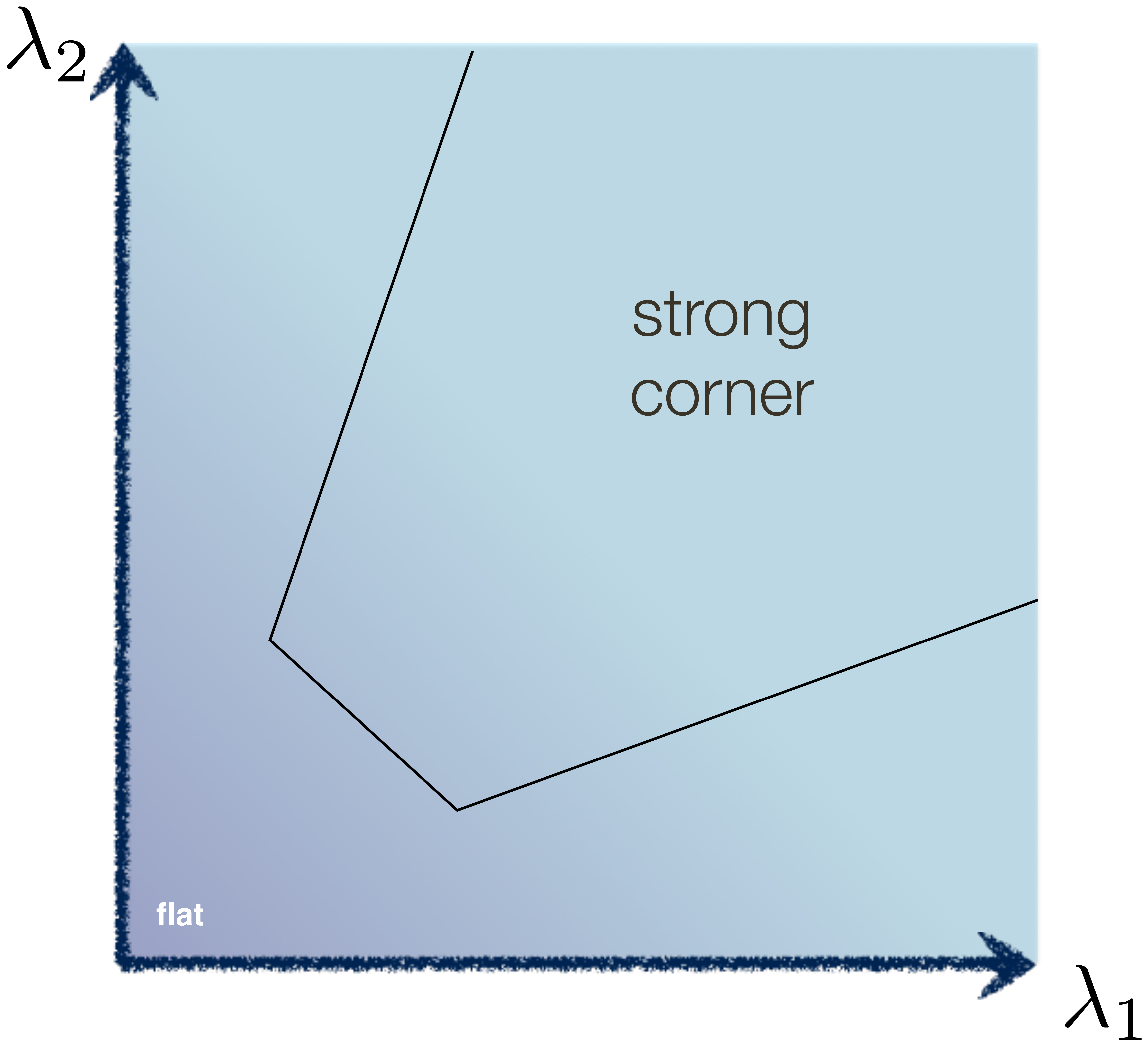



**Image Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Lecture** 10: Re-cap (**Threshold** on Eigenvalues to **Detect Corners)**




Think of a function to score 'cornerness'

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Lecture** 10: Re-cap (**Threshold** on Eigenvalues to **Detect Corners)**

Harris & Stephens (1988)

$$\det(C) - \kappa \text{trace}^2(C)$$

Kanade & Tomasi (1994)

$$\min(\lambda_1, \lambda_2)$$

Nobel (1998)

$$\frac{\det(C)}{\text{trace}(C) + \epsilon}$$

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Example**: Harris Corner Detection

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |

# **Example**: Harris Corner Detection

Lets compute a measure of “corner-ness” for the green pixel:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |

# **Example**: Harris Corner Detection

Lets compute a measure of “corner-ness” for the green pixel:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |

# **Example**: Harris Corner Detection

Lets compute a measure of “corner-ness” for the green pixel:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |

$$I_x = \frac{\partial I}{\partial x}$$

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | |
| -1 | 1 | 0 | 0 | -1 | 1 | |
| -1 | 0 | 0 | 0 | 1 | 0 | |
| -1 | 0 | 0 | 0 | 1 | 0 | |
| 0 | -1 | 0 | 0 | 1 | 0 | |
| 0 | -1 | 0 | 0 | 1 | 0 | |
| 0 | -1 | 0 | 0 | 1 | 0 | |
| 0 | -1 | 0 | 0 | 1 | 0 | |

# **Example**: Harris Corner Detection

Lets compute a measure of “corner-ness” for the green pixel:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |

$$I_x = \frac{\partial I}{\partial x}$$

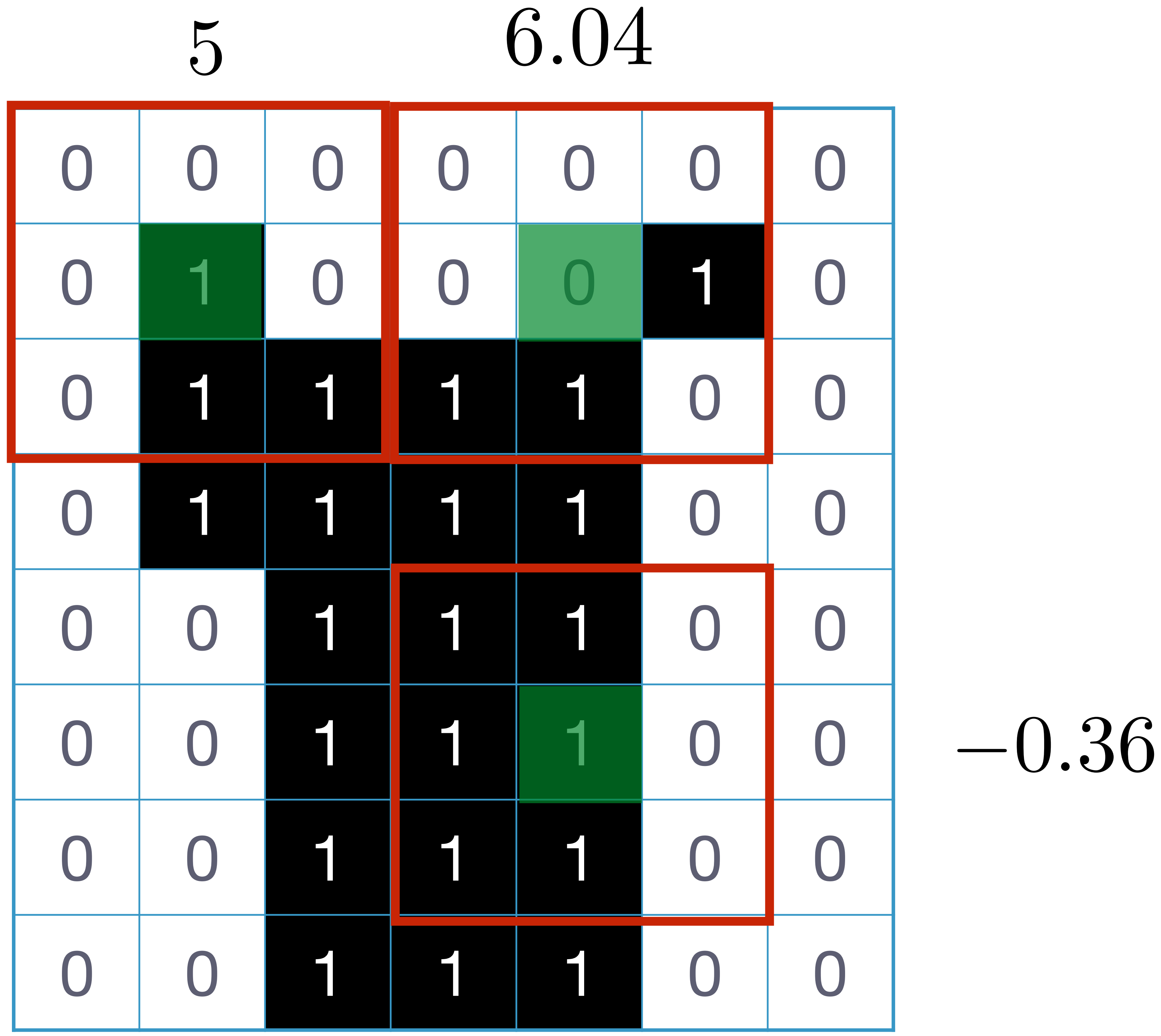
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | |
| -1 | 1 | 0 | 0 | -1 | 1 | |
| -1 | 0 | 0 | 0 | 1 | 0 | |
| -1 | 0 | 0 | 0 | 1 | 0 | |
| 0 | -1 | 0 | 0 | 1 | 0 | |
| 0 | -1 | 0 | 0 | 1 | 0 | |
| 0 | -1 | 0 | 0 | 1 | 0 | |
| 0 | -1 | 0 | 0 | 1 | 0 | |

$$I_y = \frac{\partial I}{\partial y}$$

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | -1 | 0 | 0 | 0 | -1 | 0 |
| 0 | 0 | -1 | -1 | -1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 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 | 0 |
| | | | | | | |

# **Example**: Harris Corner Detection

Lets compute a measure of “corner-ness” for the green pixel:



# **Harris** Corner Detection Review

— Filter image with **Gaussian**

— Compute magnitude of the x and y **gradients** at each pixel

— Construct C in a window around each pixel

— Harris uses a **Gaussian window**

— Solve for product of the λ's

— If λ's both are big (product reaches local maximum above threshold) then we have a corner

— Harris also checks that ratio of λs is not too high

Harris & Stephens (1988)

$$\det(C) - \kappa \operatorname{trace}^2(C)$$

# Compute the **Covariance Matrix**

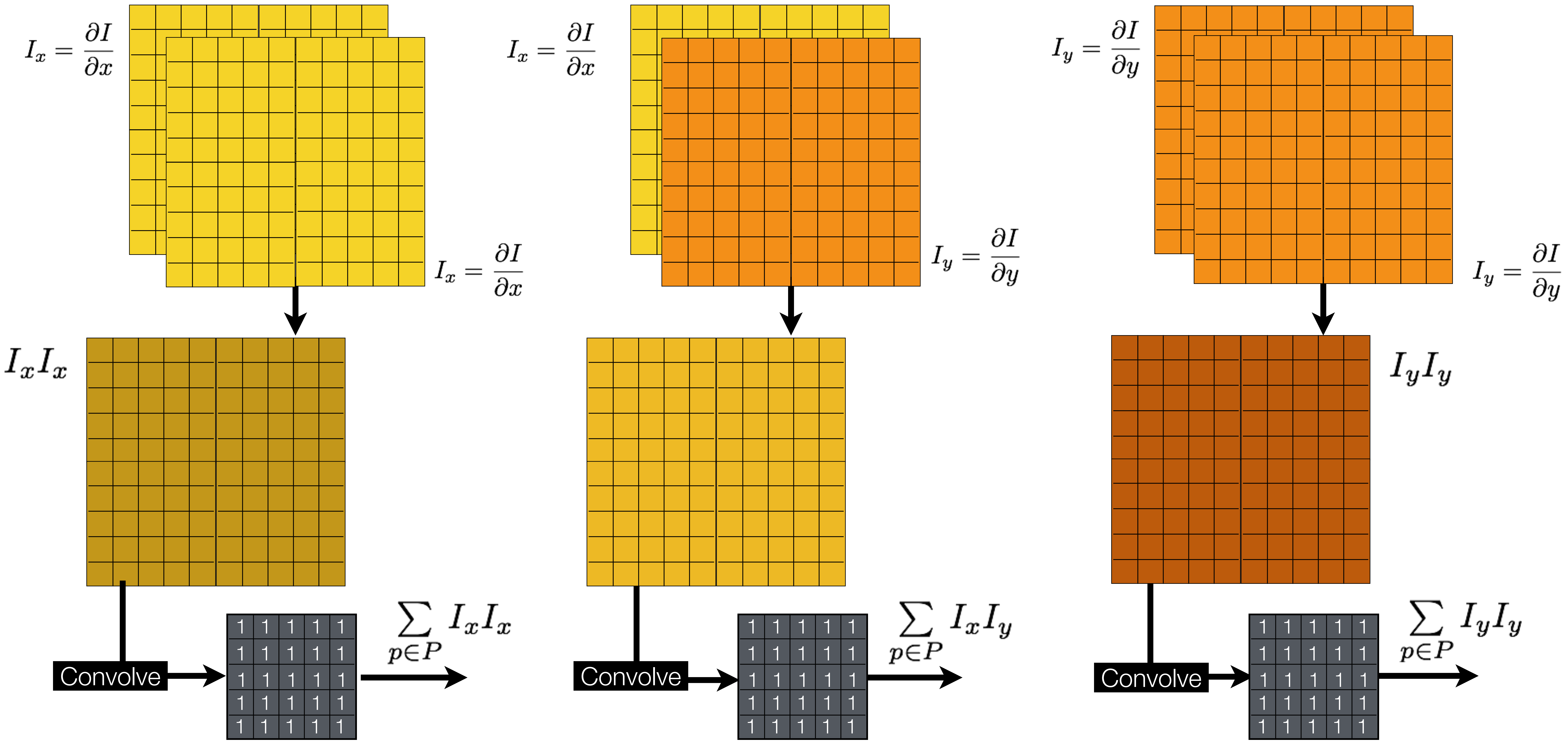
**Sum** can be implemented as an (unnormalized) box filter with

$$C = \begin{bmatrix} \sum\limits_{p \in P} I_x I_x & \sum\limits_{p \in P} I_x I_y \\ \sum\limits_{p \in P} I_y I_x & \sum\limits_{p \in P} I_y I_y \end{bmatrix}$$

Harris uses a **Gaussian** weighting instead

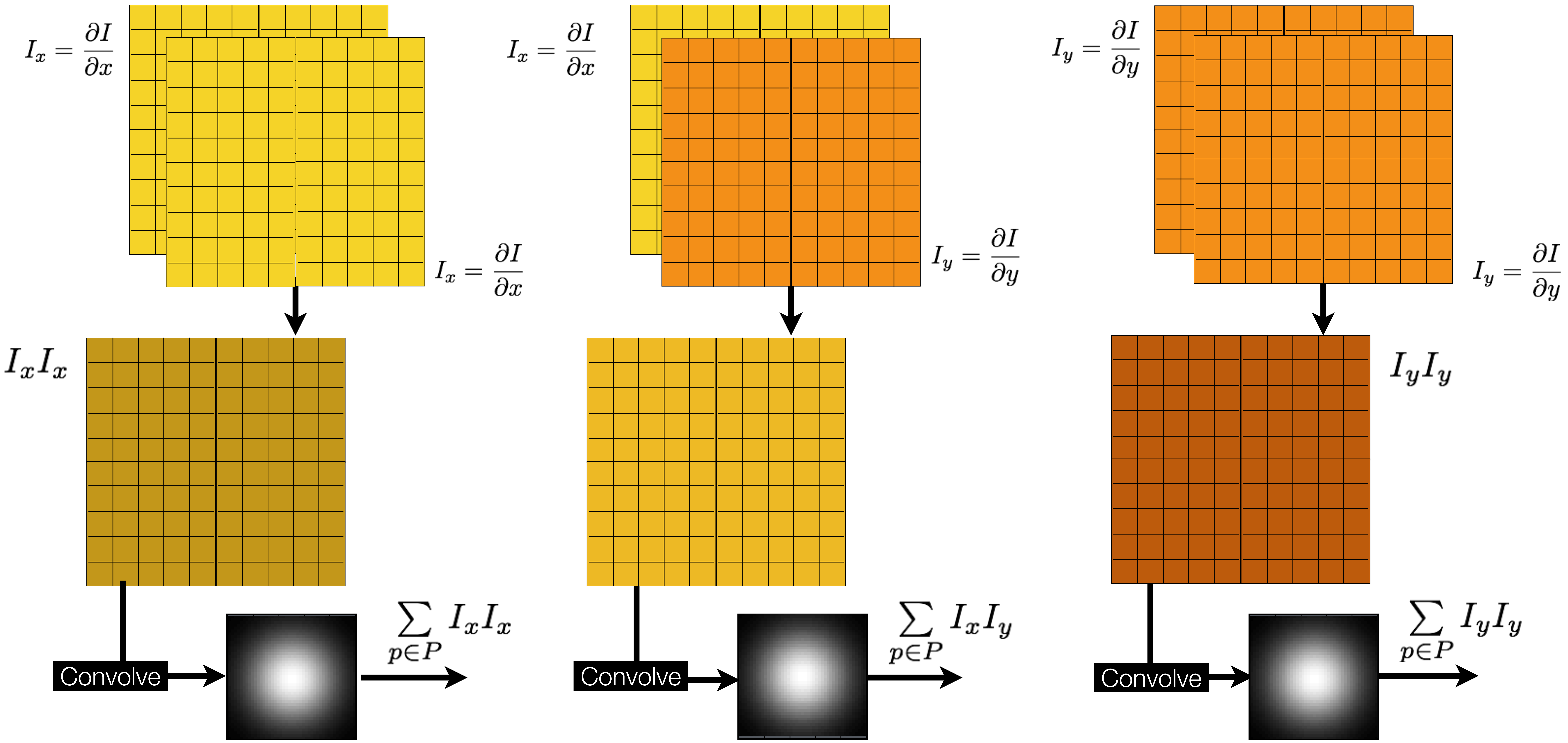
# Computing Covariance Matrix **Efficiently**

$$C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

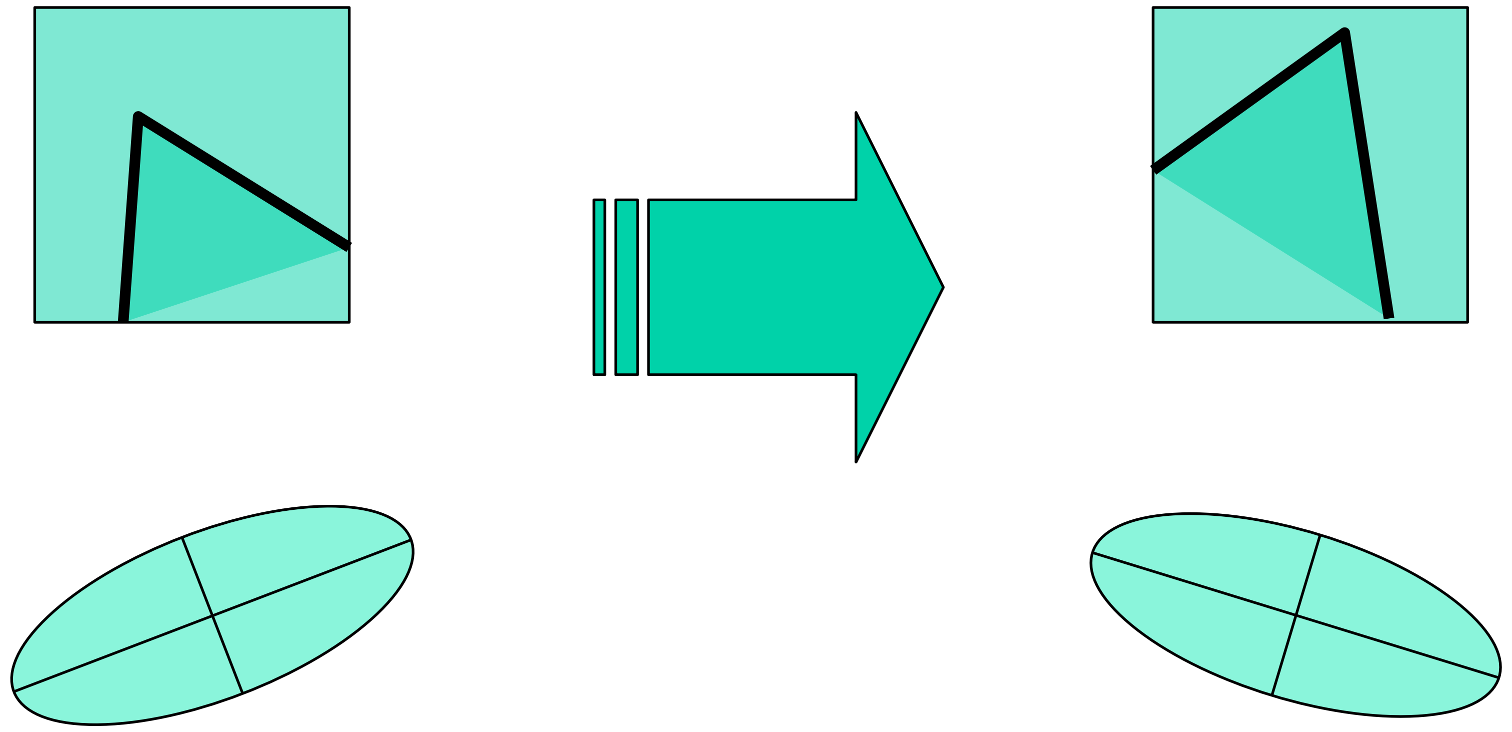


# Computing Covariance Matrix **Efficiently**

$$C = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$



# **Properties**: Rotational Invariance



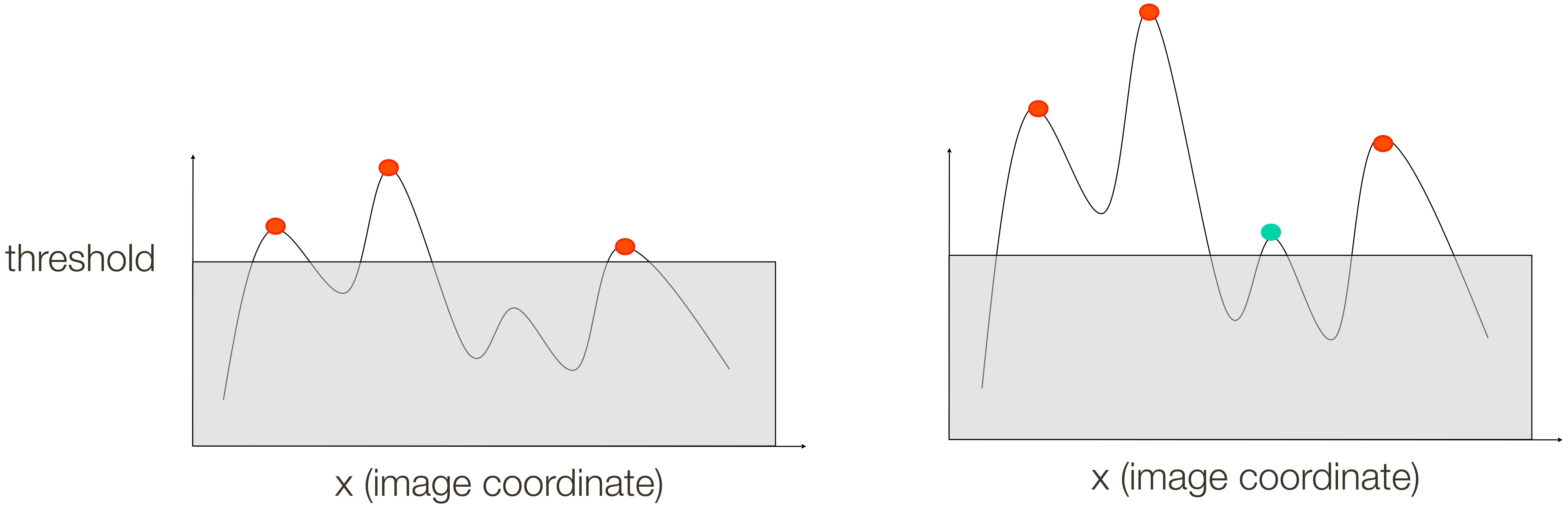
Ellipse rotates but its shape (**eigenvalues**) remains the same

**Corner response is invariant to image rotation**

# **Properties**: (partial) Invariance to Intensity Shifts and Scaling

Only derivatives are used -> Invariance to intensity shifts

Intensity scale could effect performance




**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Properties**: NOT Invariant to Scale Changes


**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Example** 2: Wagon Wheel (Harris Results)

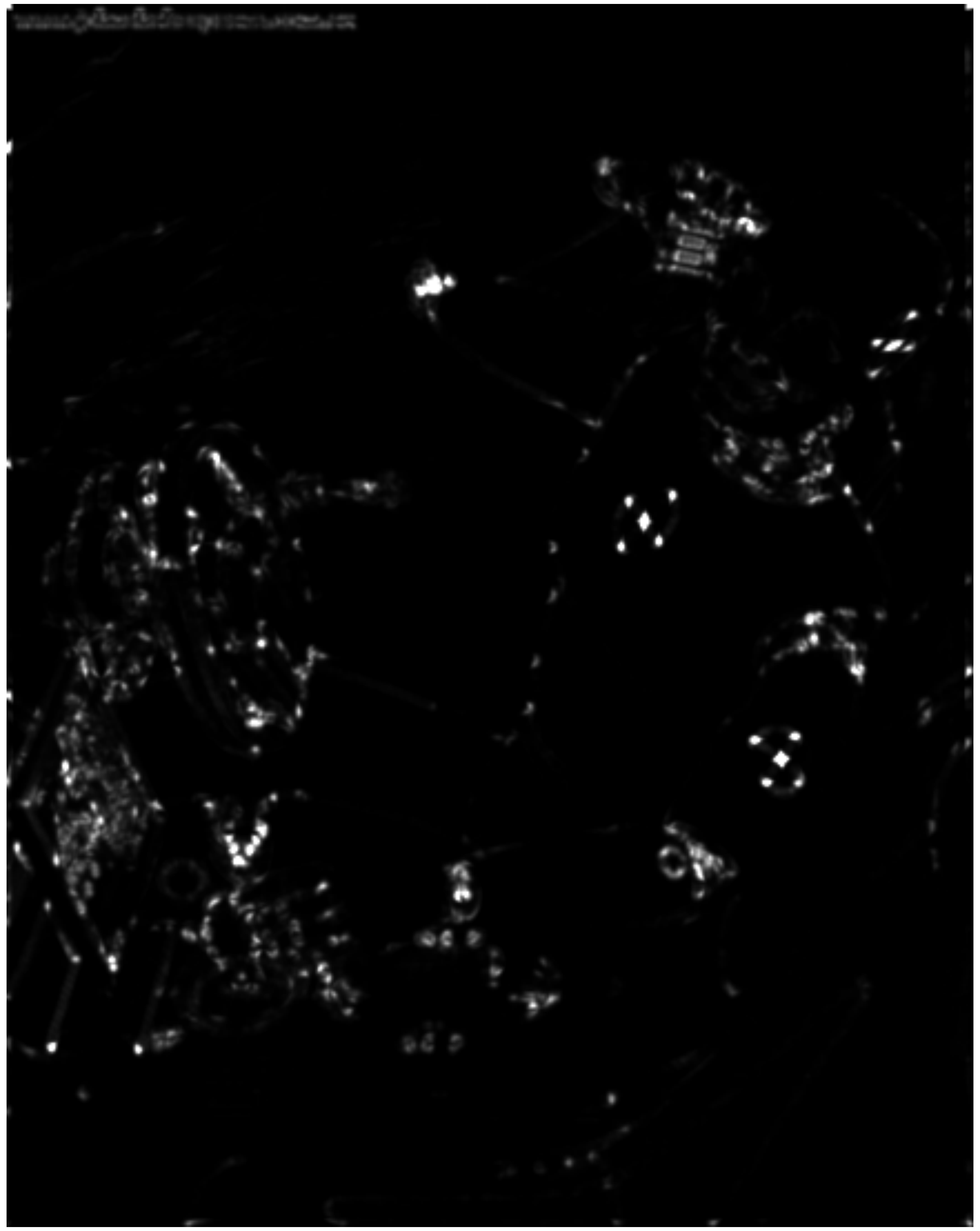
$\sigma = 1$ (219 points)

$\sigma = 2$ (155 points)

$\sigma = 3$ (110 points)

$\sigma = 4$ (87 points)

# **Example** 3: Crash Test Dummy (Harris Result)



corner response image


$\sigma = 1$ (175 points)

**Original Image Credit**: John Shakespeare, Sydney Morning Herald

# **Example** 2: Wagon Wheel (Harris Results)

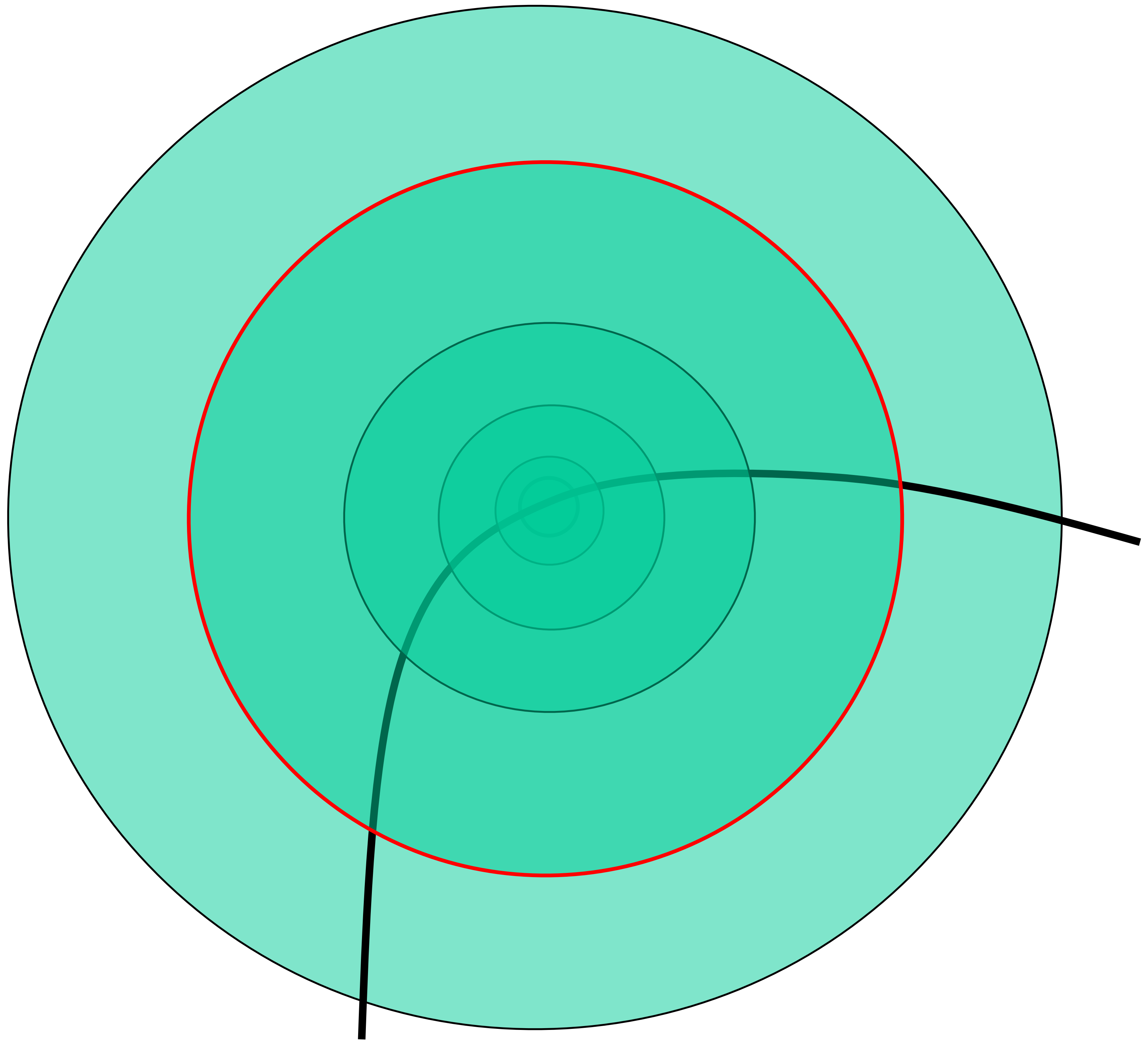
$\sigma = 1$ (219 points)

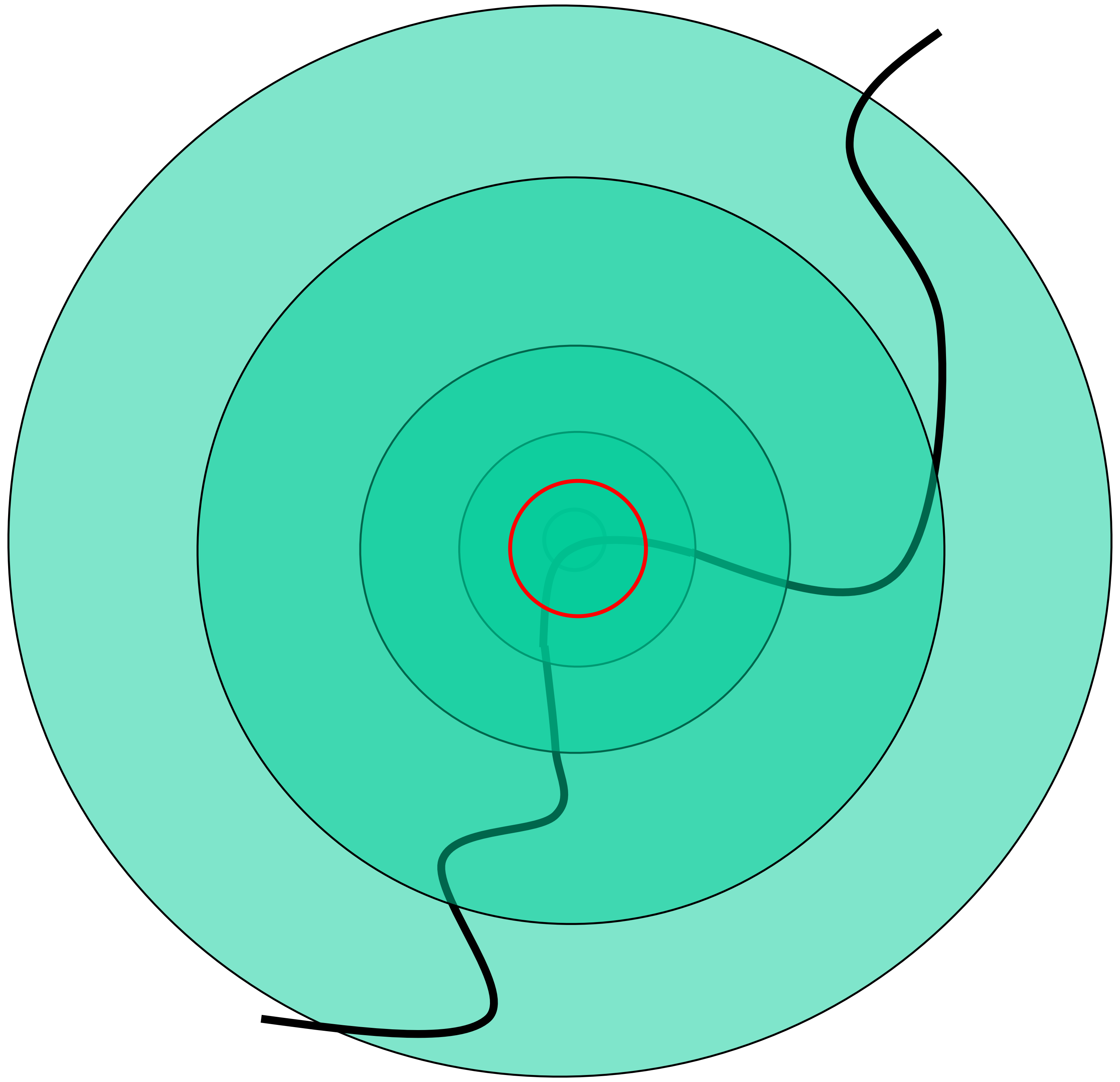
$\sigma = 2$ (155 points)

$\sigma = 3$ (110 points)

$\sigma = 4$ (87 points)

# Intuitively ...

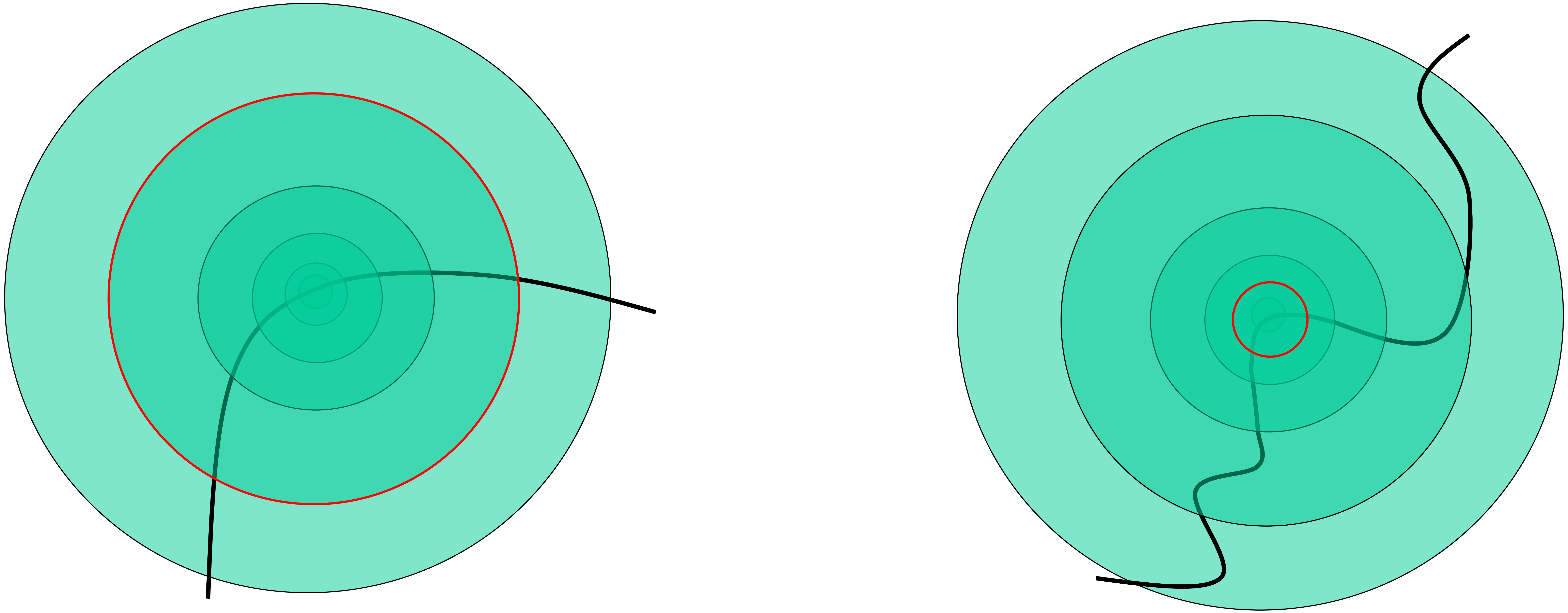




**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Intuitively …

Find local maxima in both **position** and **scale**



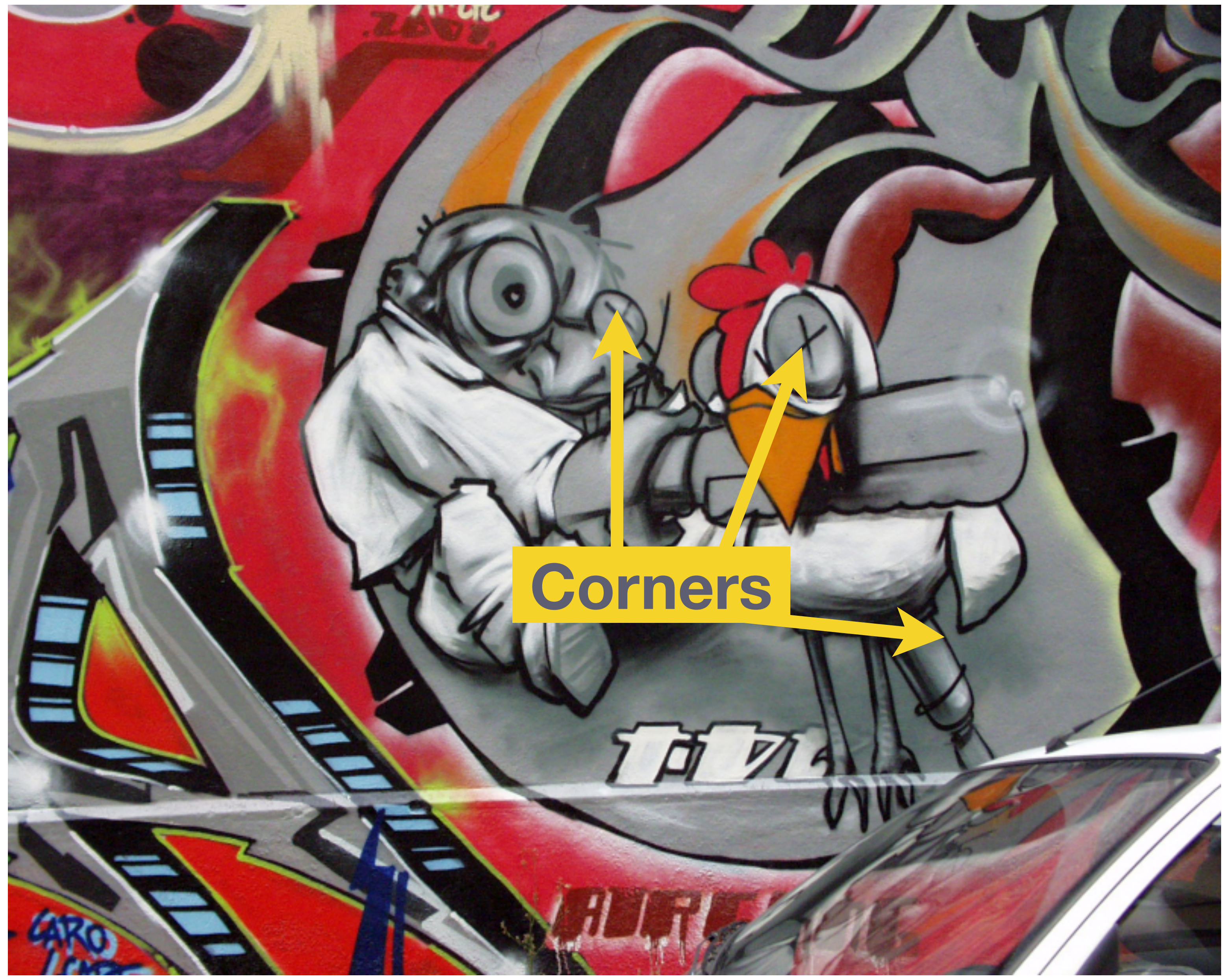
**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Blobs** features

**Blobs** are circular regions in the image

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

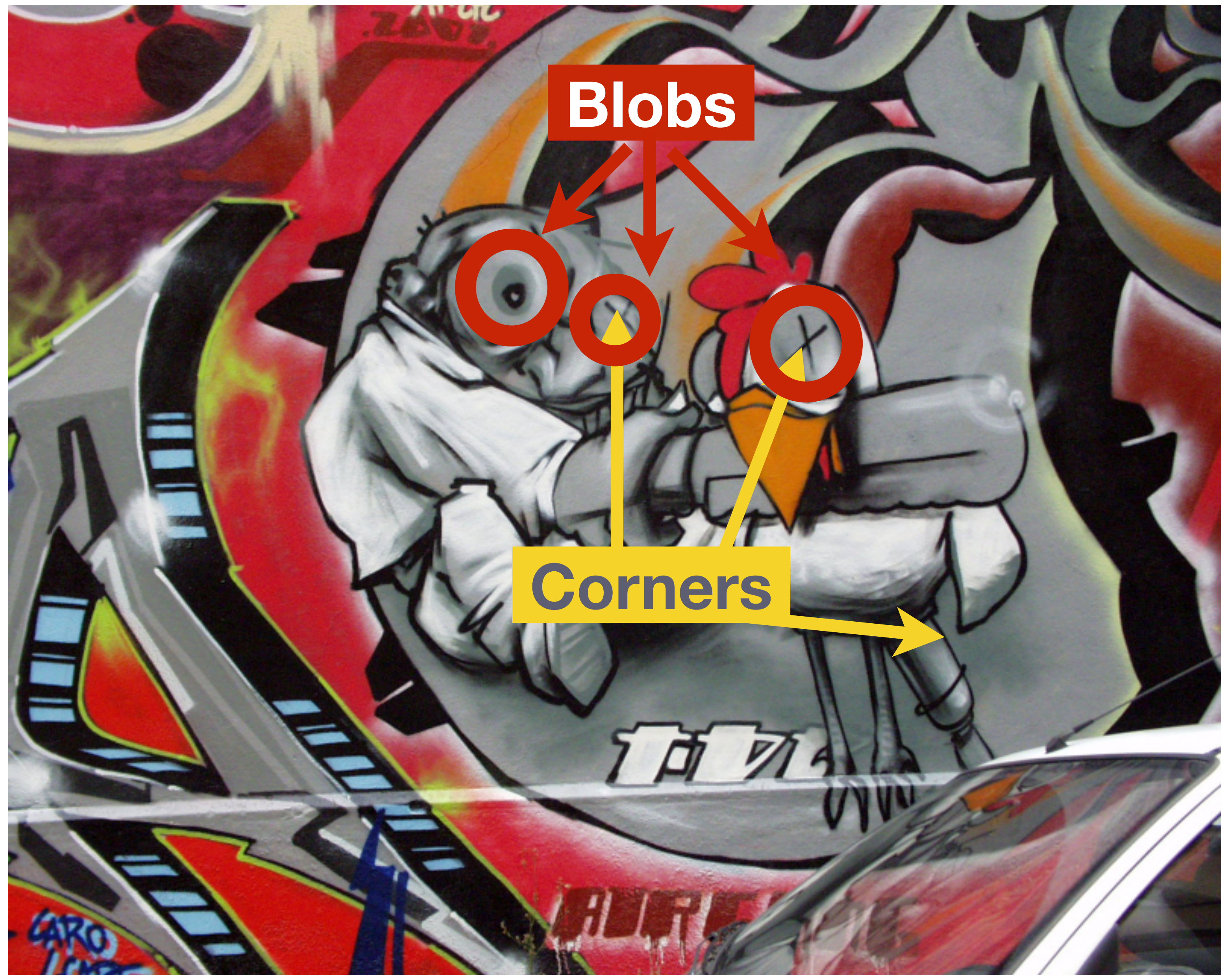
# **Blobs** features




**Blobs** are circular regions in the image

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

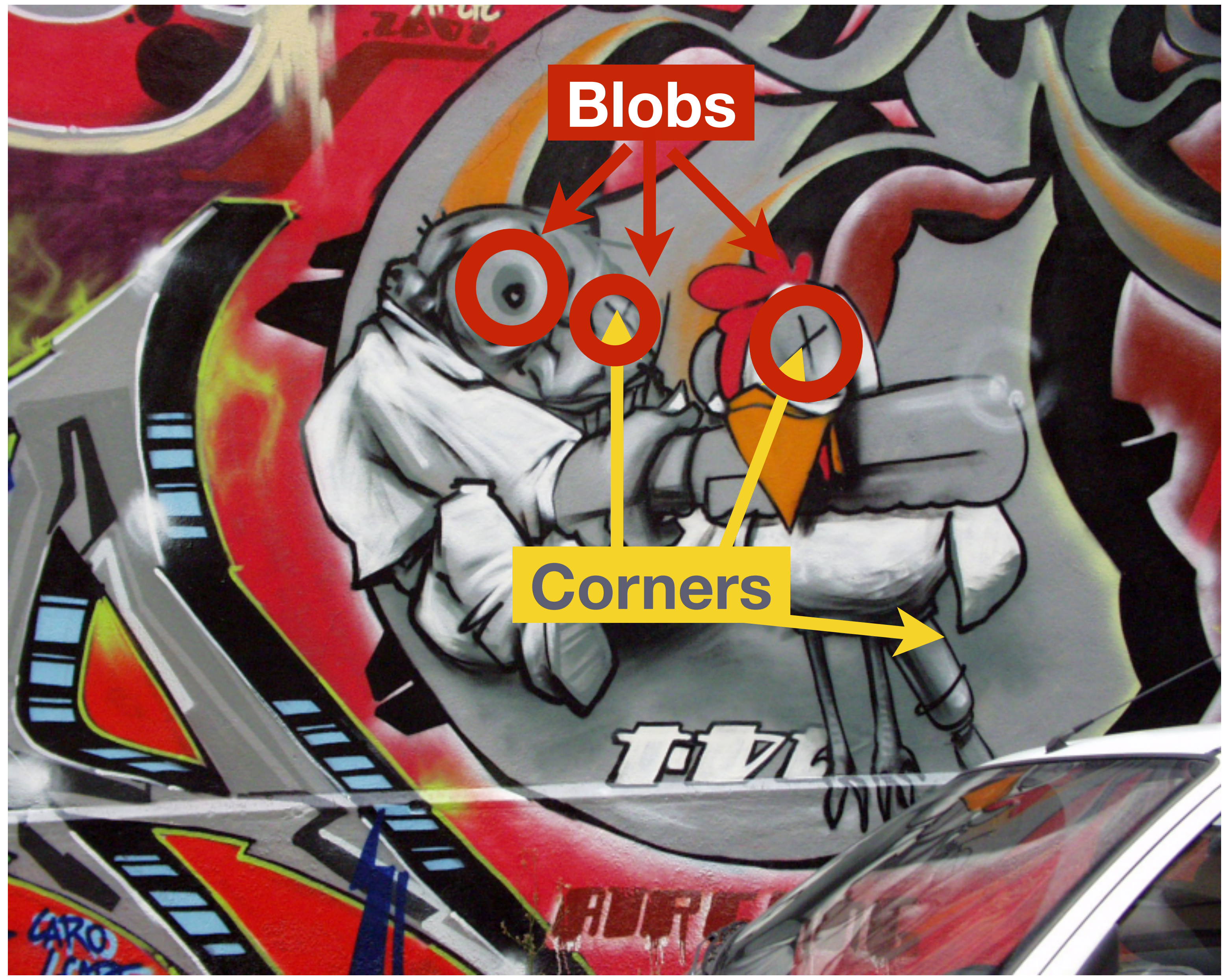
# **Blobs** features




**Blobs** are circular regions in the image

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

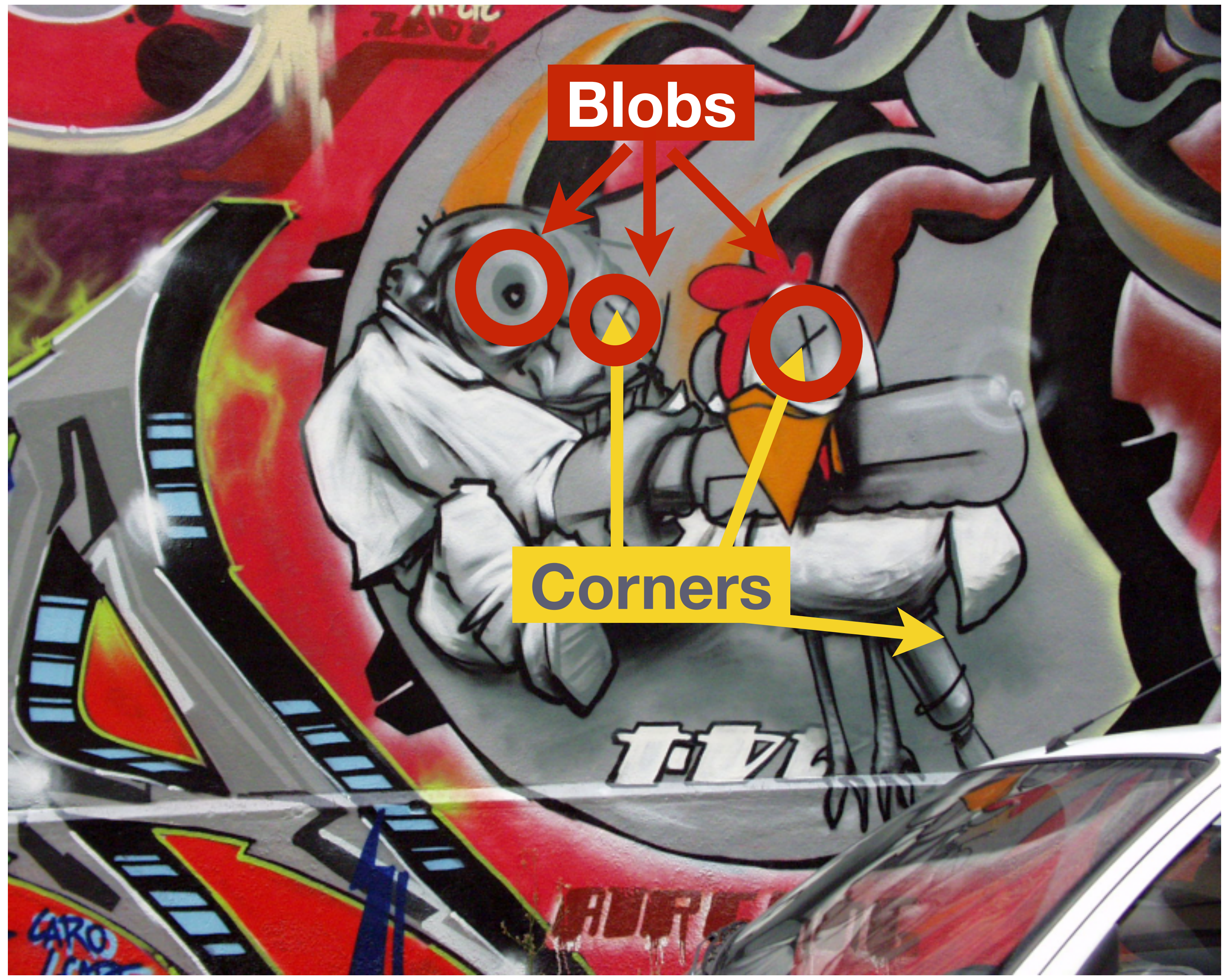
# **Blobs** features



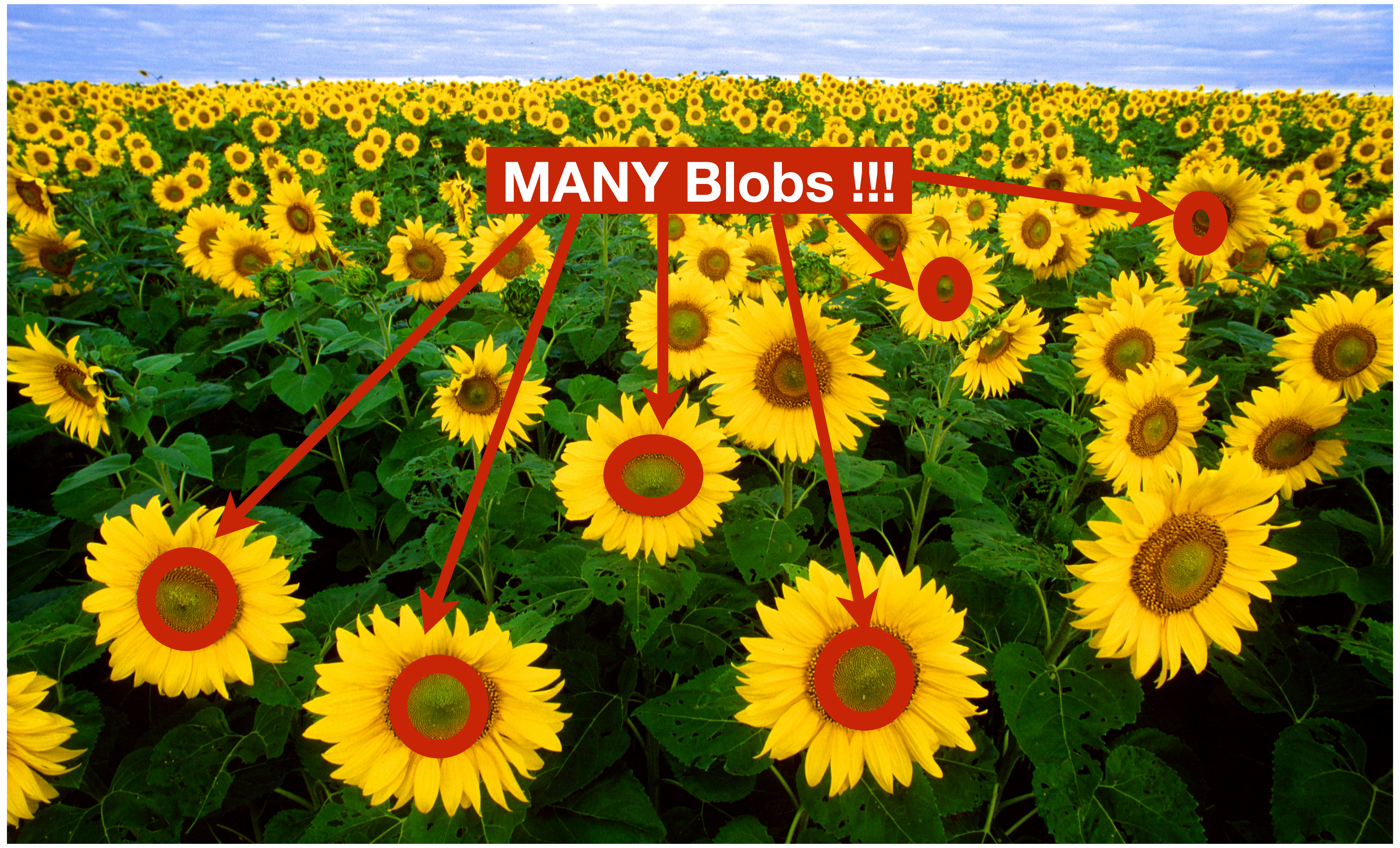

**Blobs** are circular regions in the image

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Blobs** features



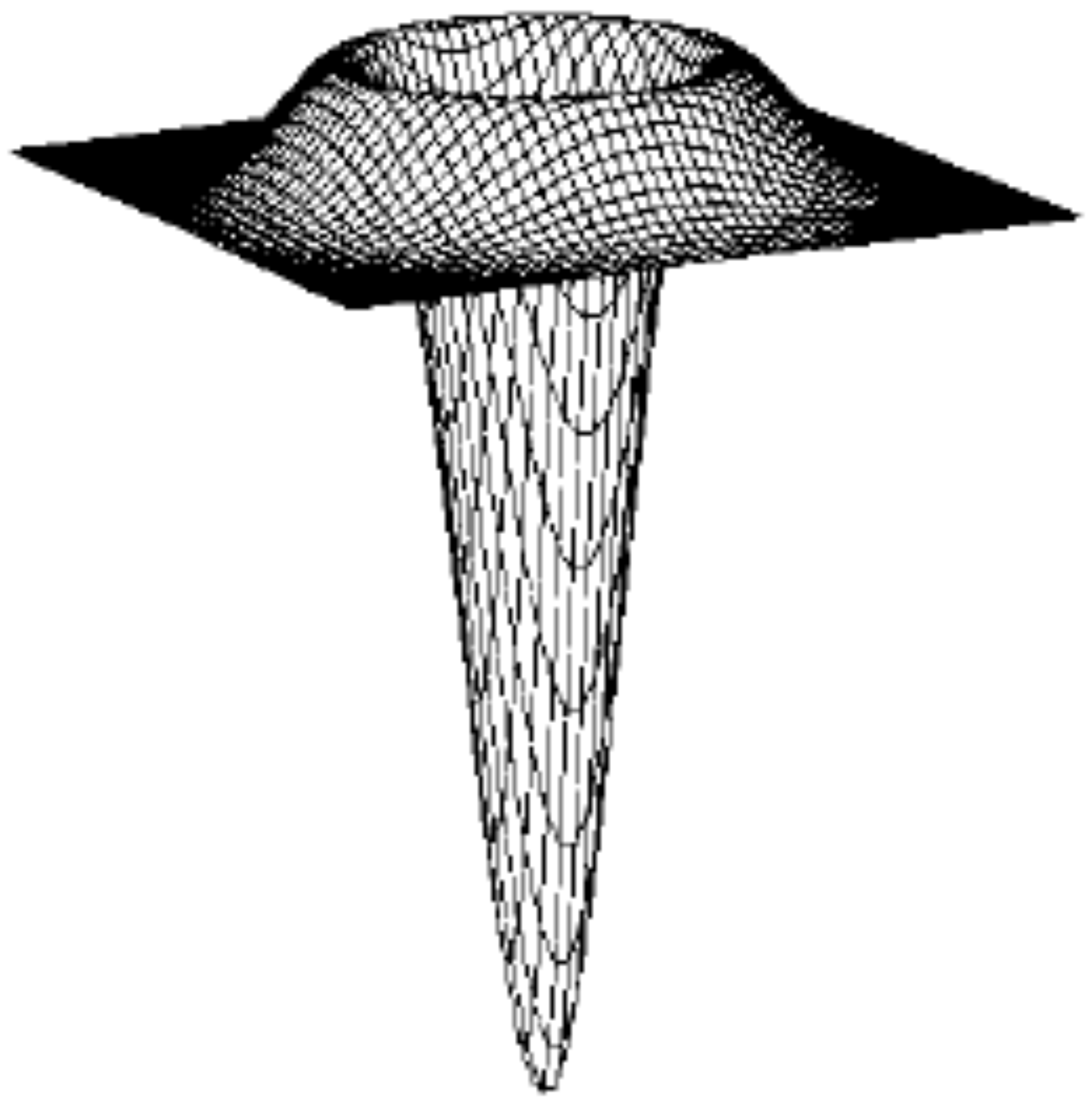

**Blobs** are circular regions in the image




**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Recall**: Marr / Hildreth **Laplacian of Gaussian**

Here's a 3D plot of the Laplacian of the Gaussian ( $\nabla^2 G$ )



. . . with its characteristic "Mexican hat" shape

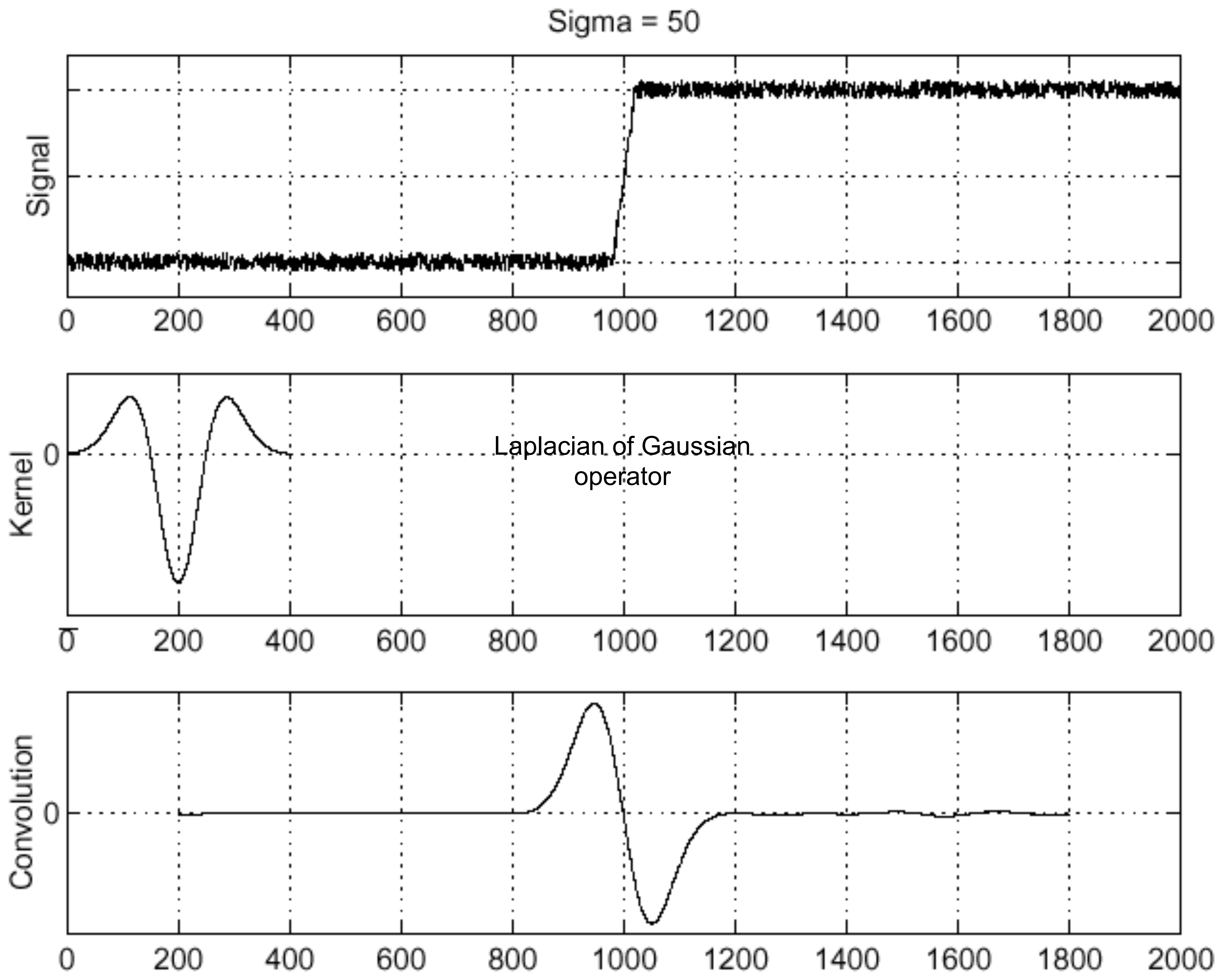
# **Recall**: Marr / Hildreth **Laplacian of Gaussian**

Lets consider a row of pixels in an image:

$I(X, 245)$

$\nabla^2 G$

$\nabla^2 G \otimes I(X, Y)$

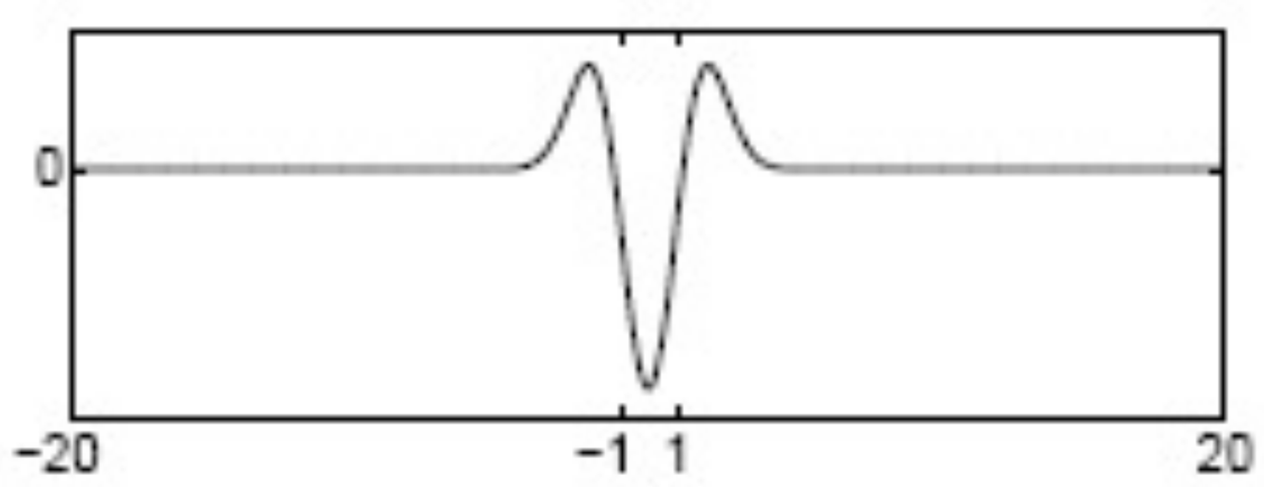


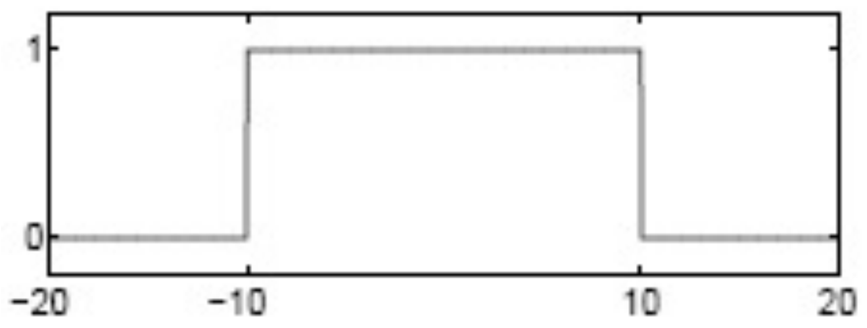

Where is the edge?

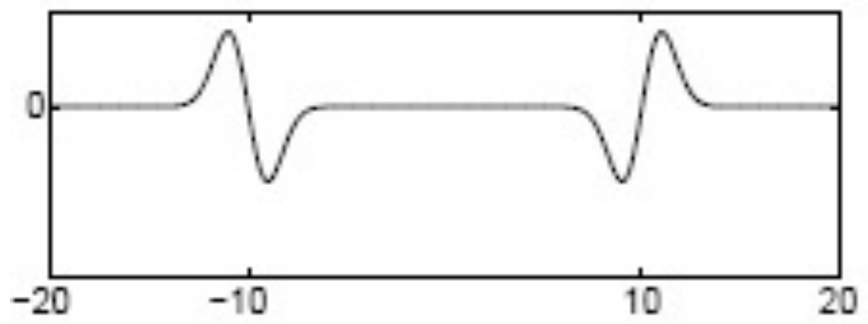
Zero-crossings of bottom graph

# Formally …

Laplacian filter



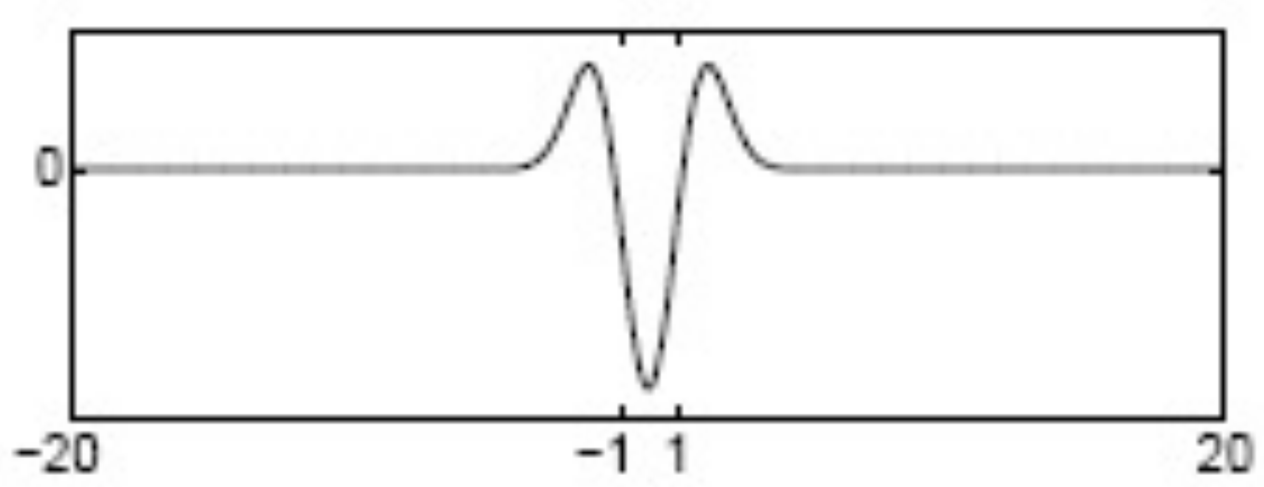



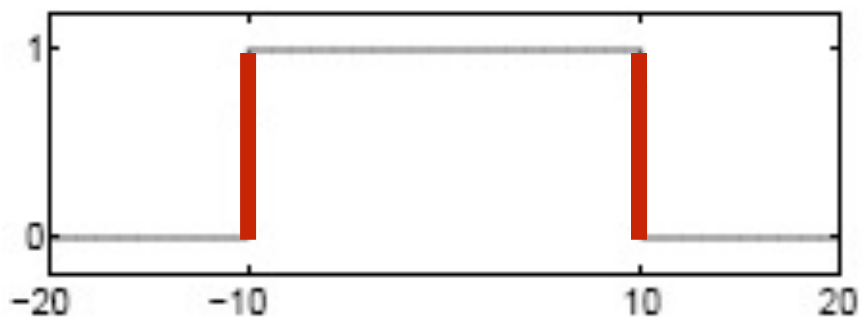



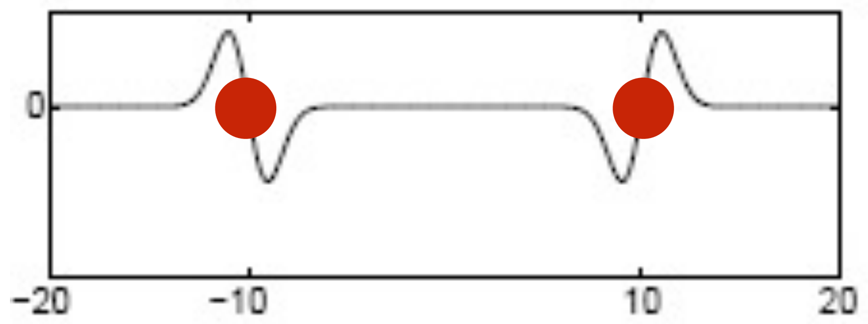

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Formally …

Laplacian filter



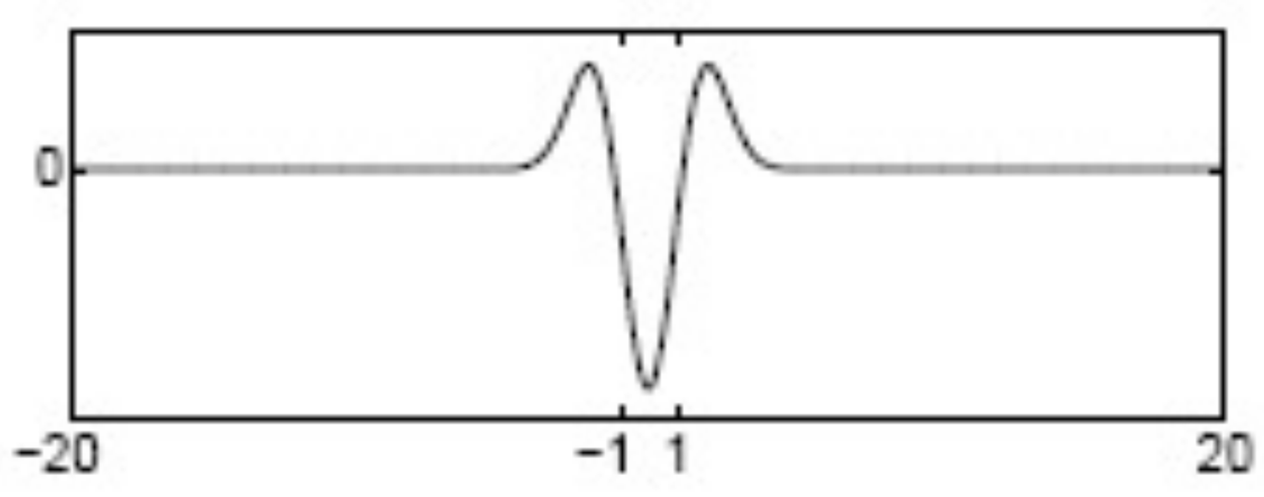



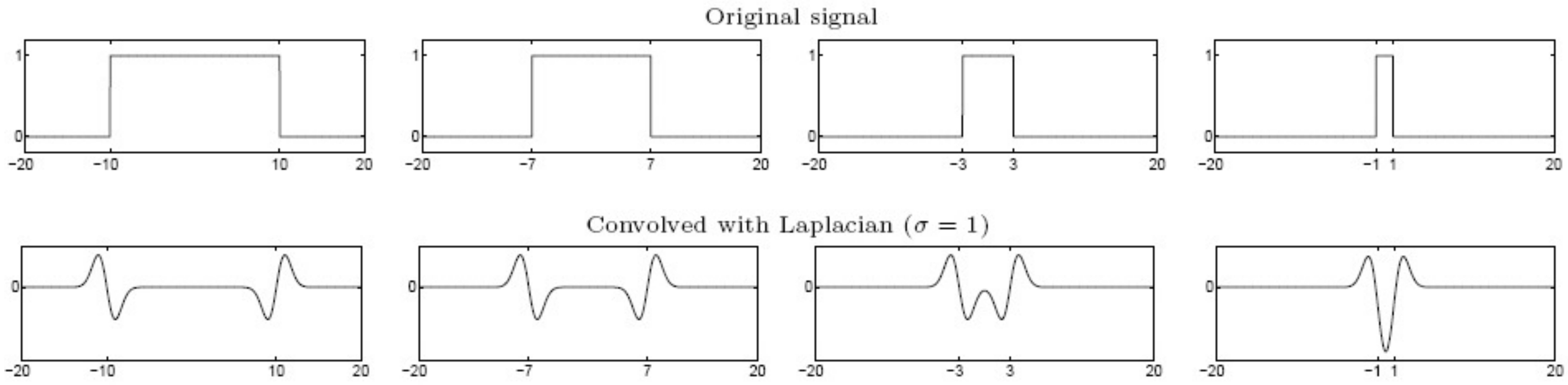




**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Formally …

Laplacian filter

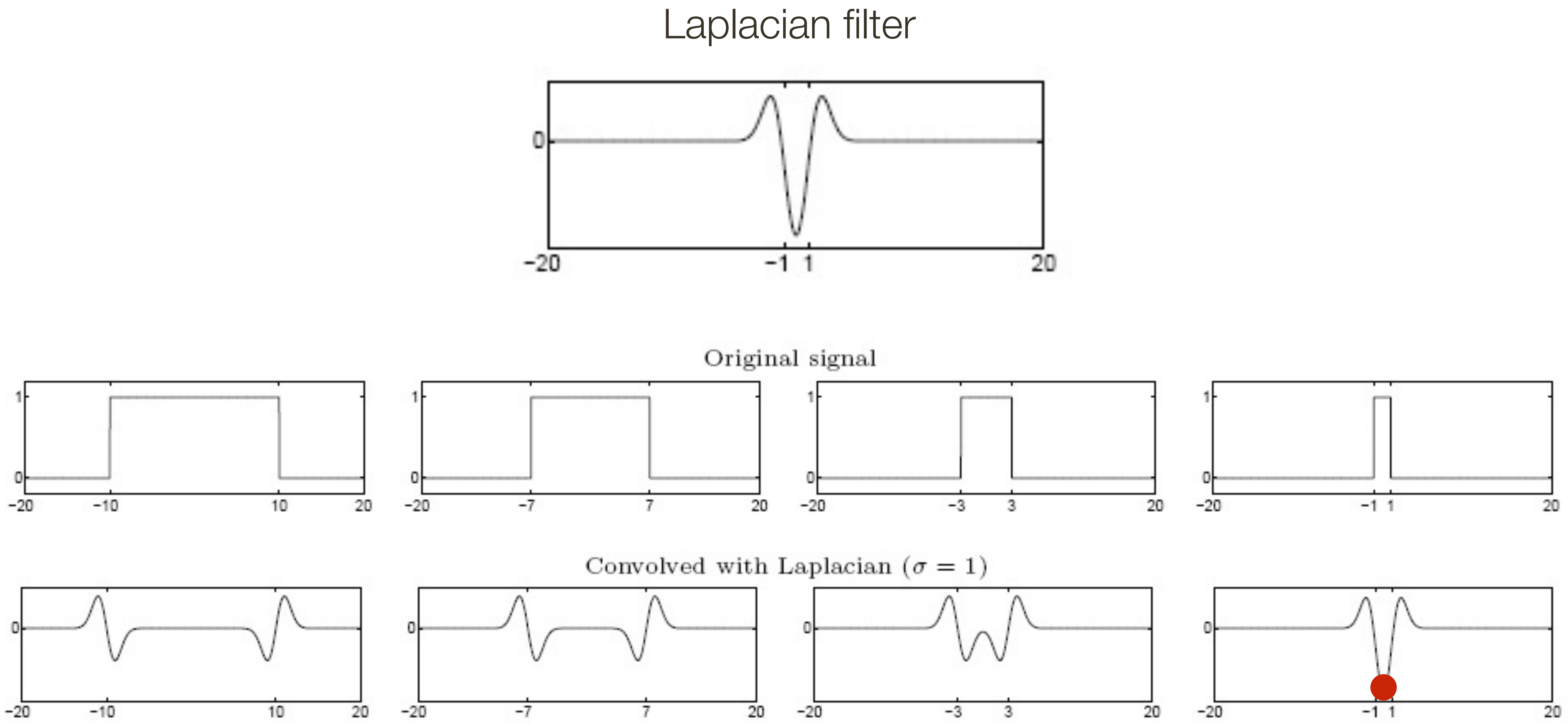



Highest response when the signal has the same **characteristic scale** as the filter

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Formally …



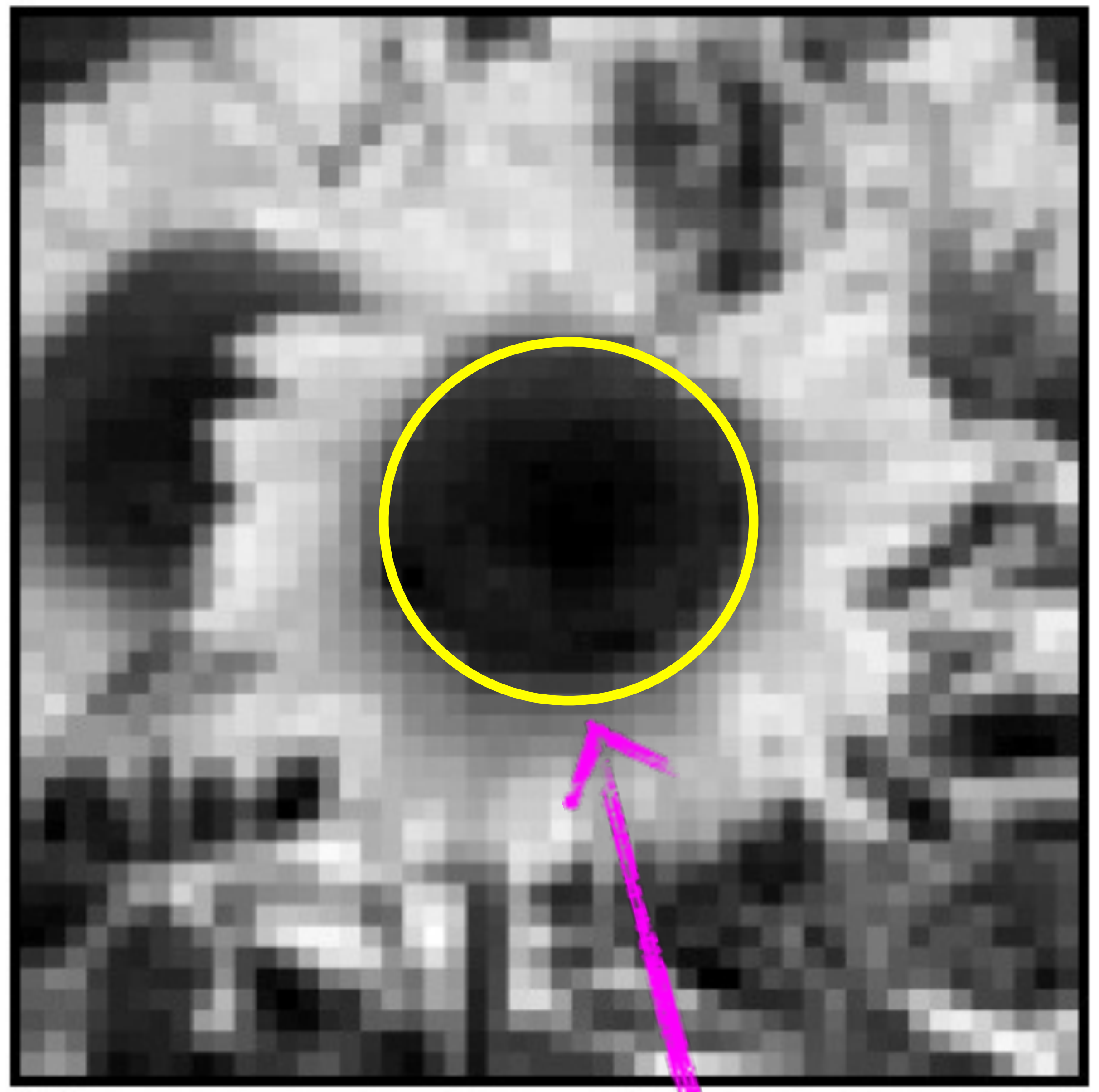

Highest response when the signal has the same **characteristic scale** as the filter

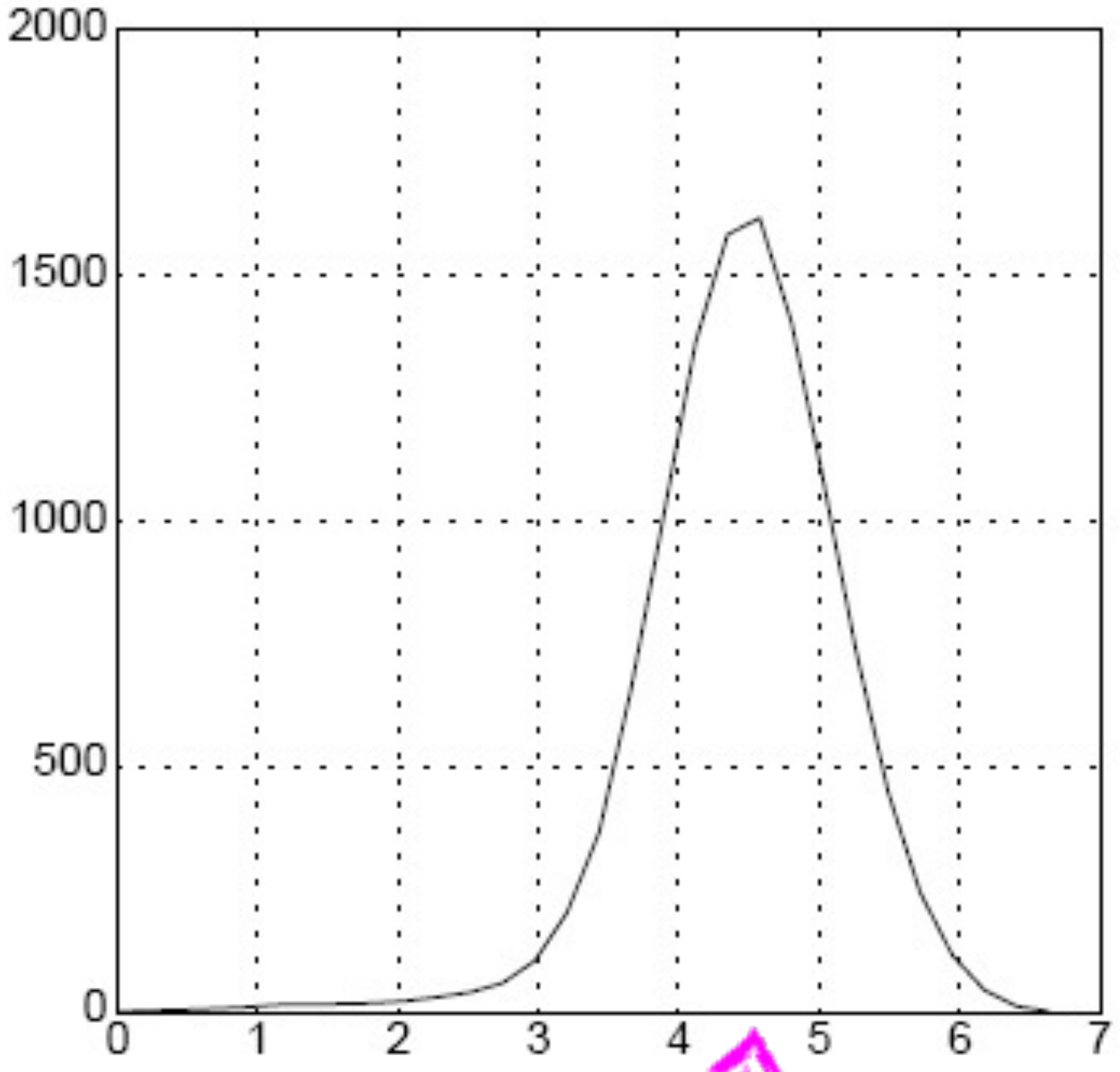
**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Characteristic** Scale

characteristic scale - the scale that produces peak filter response



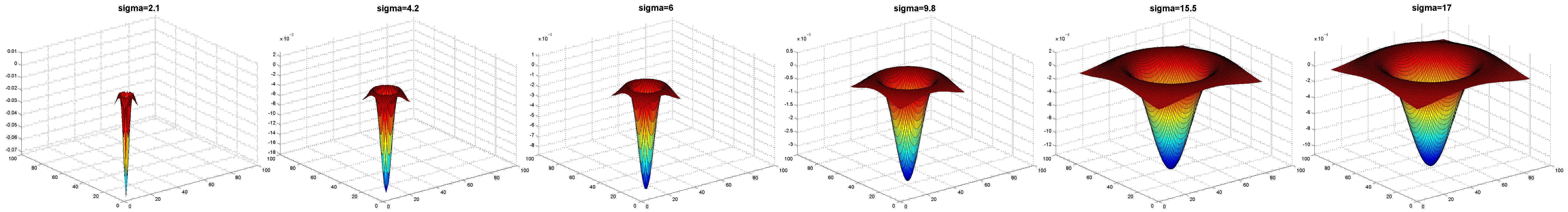



characteristic scale

we need to search for characteristic scales

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

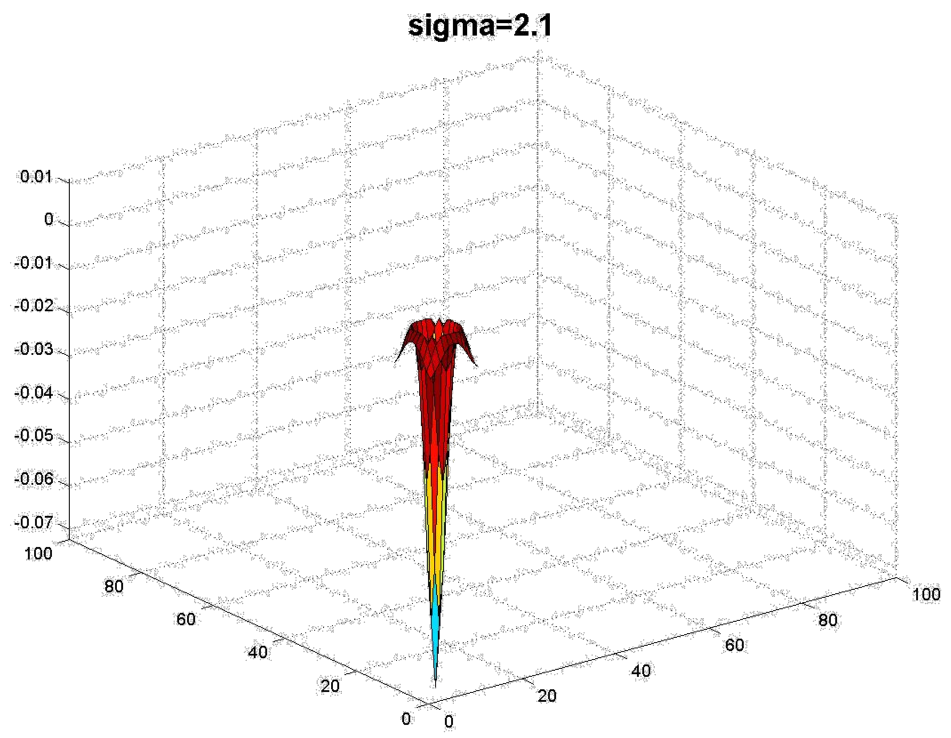
# Applying **Laplacian** Filter at Different **Scales**




Full size

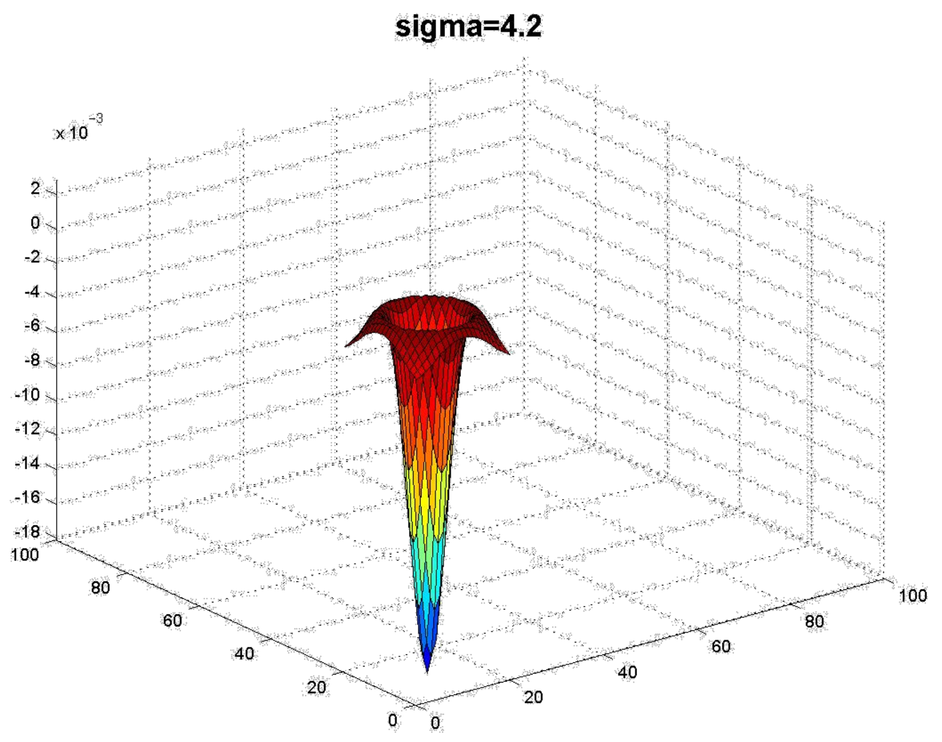
**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

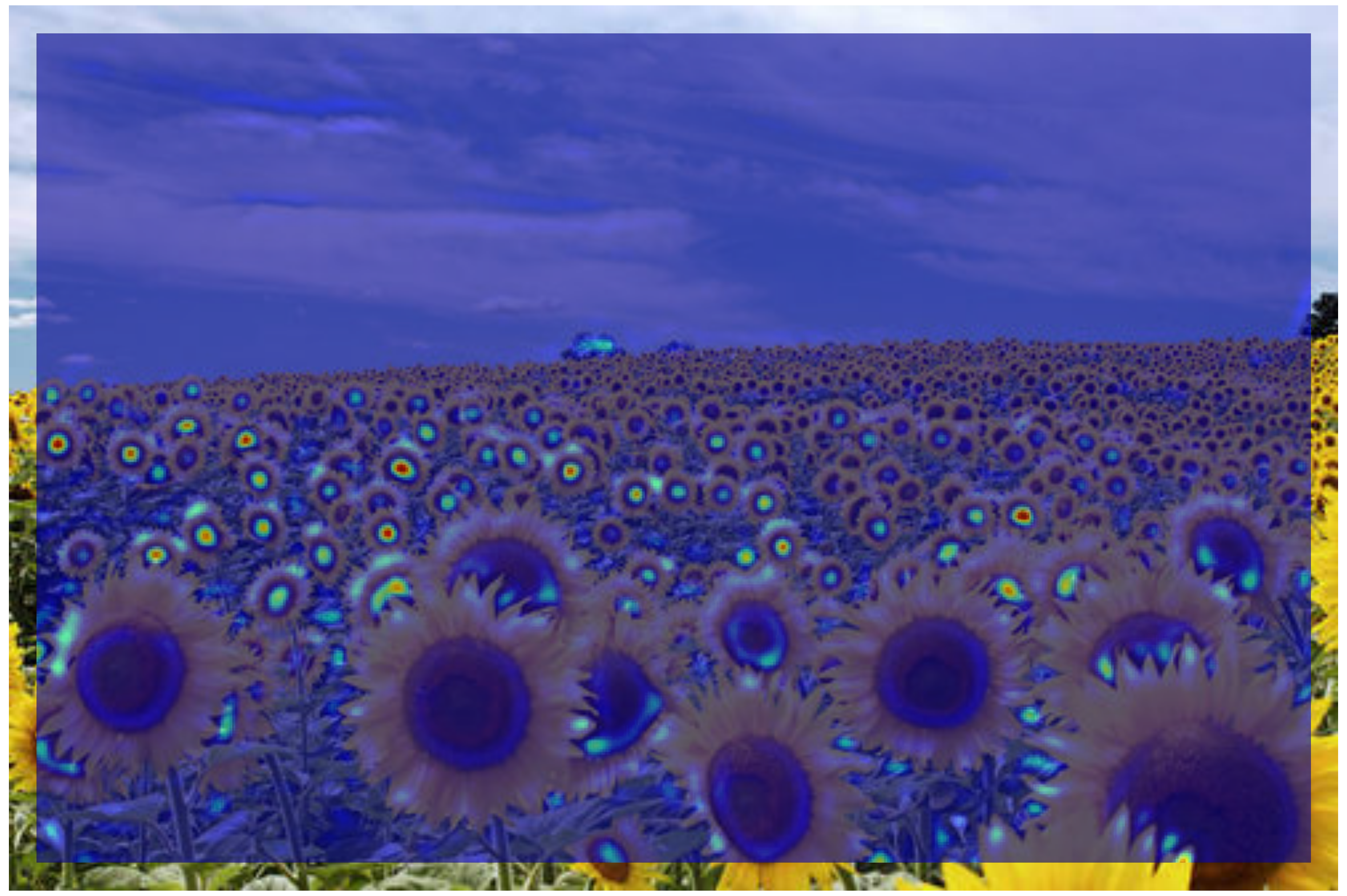
# Applying **Laplacian** Filter at Different **Scales**




**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

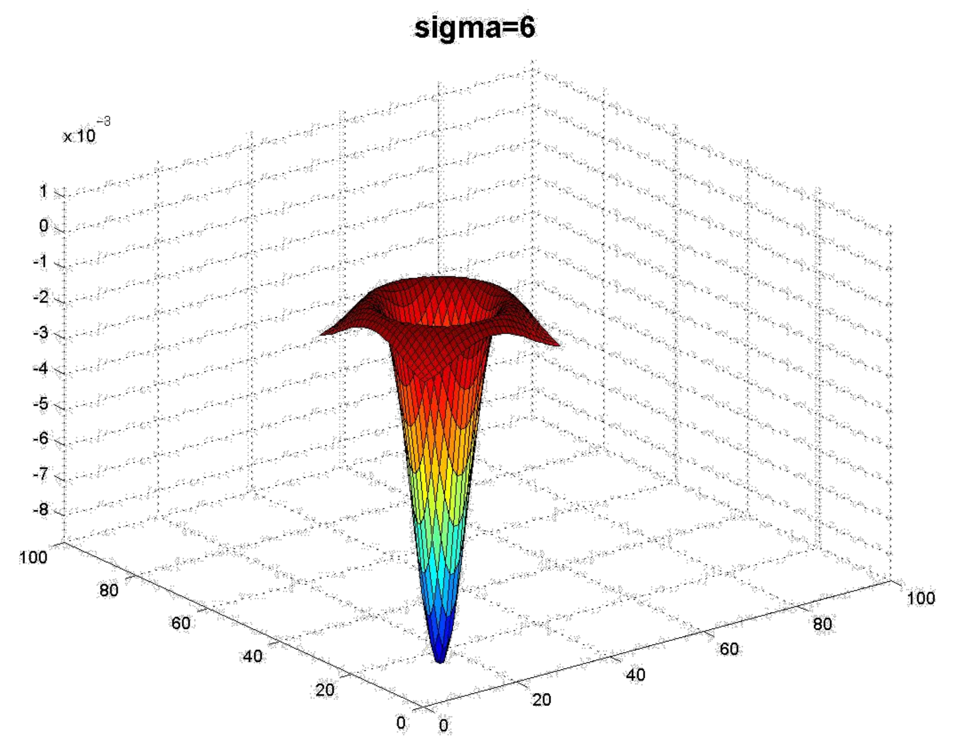
# Applying **Laplacian** Filter at Different **Scales**



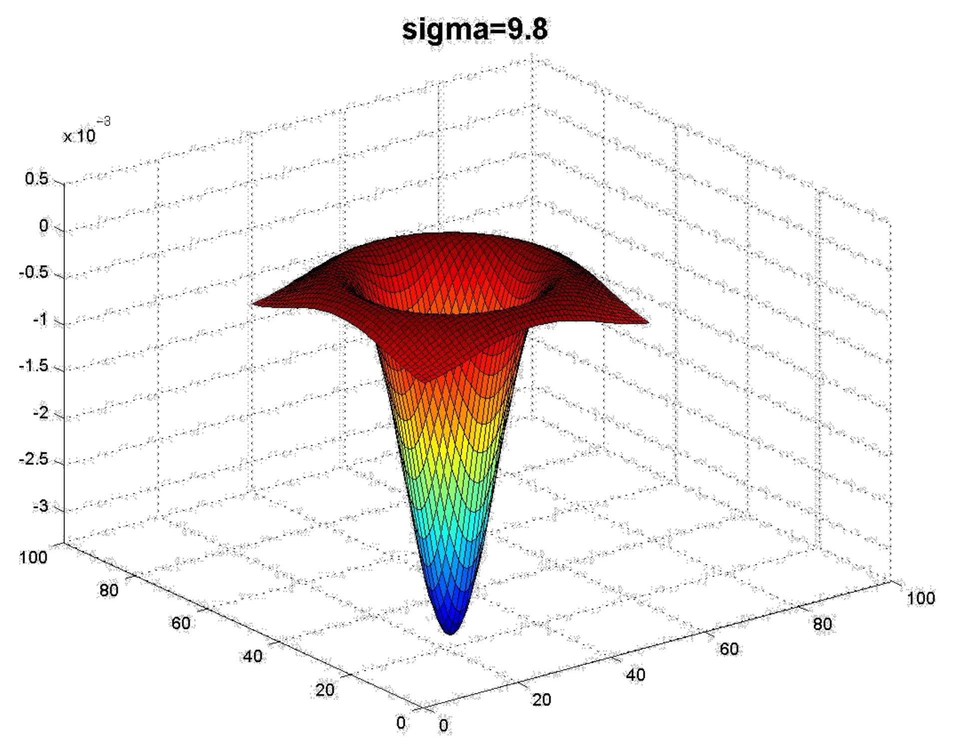
**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Applying **Laplacian** Filter at Different **Scales**



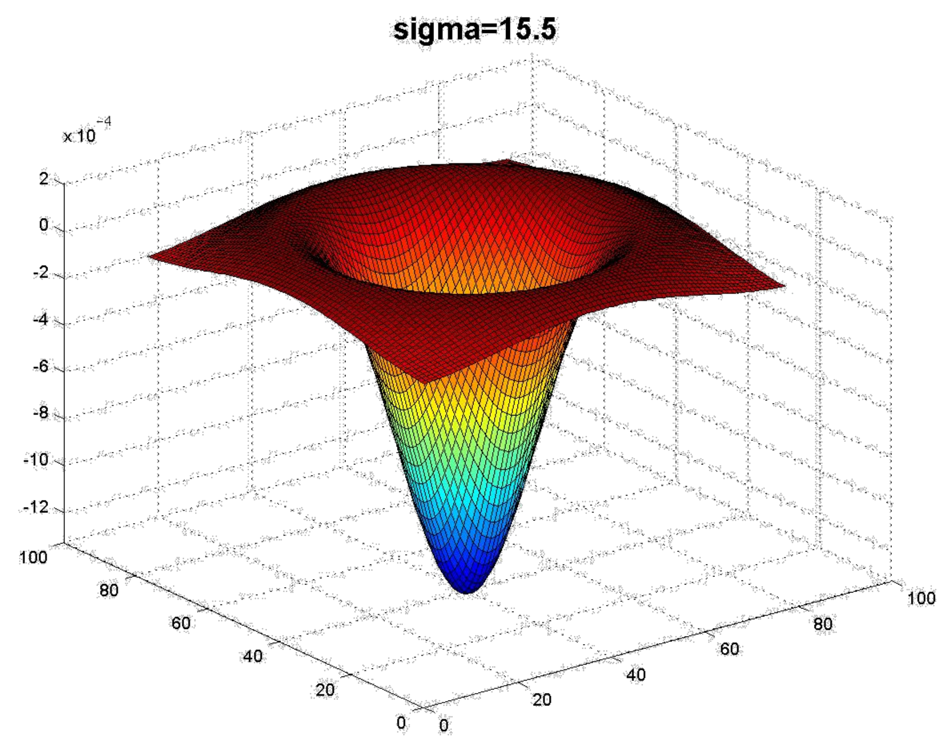

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

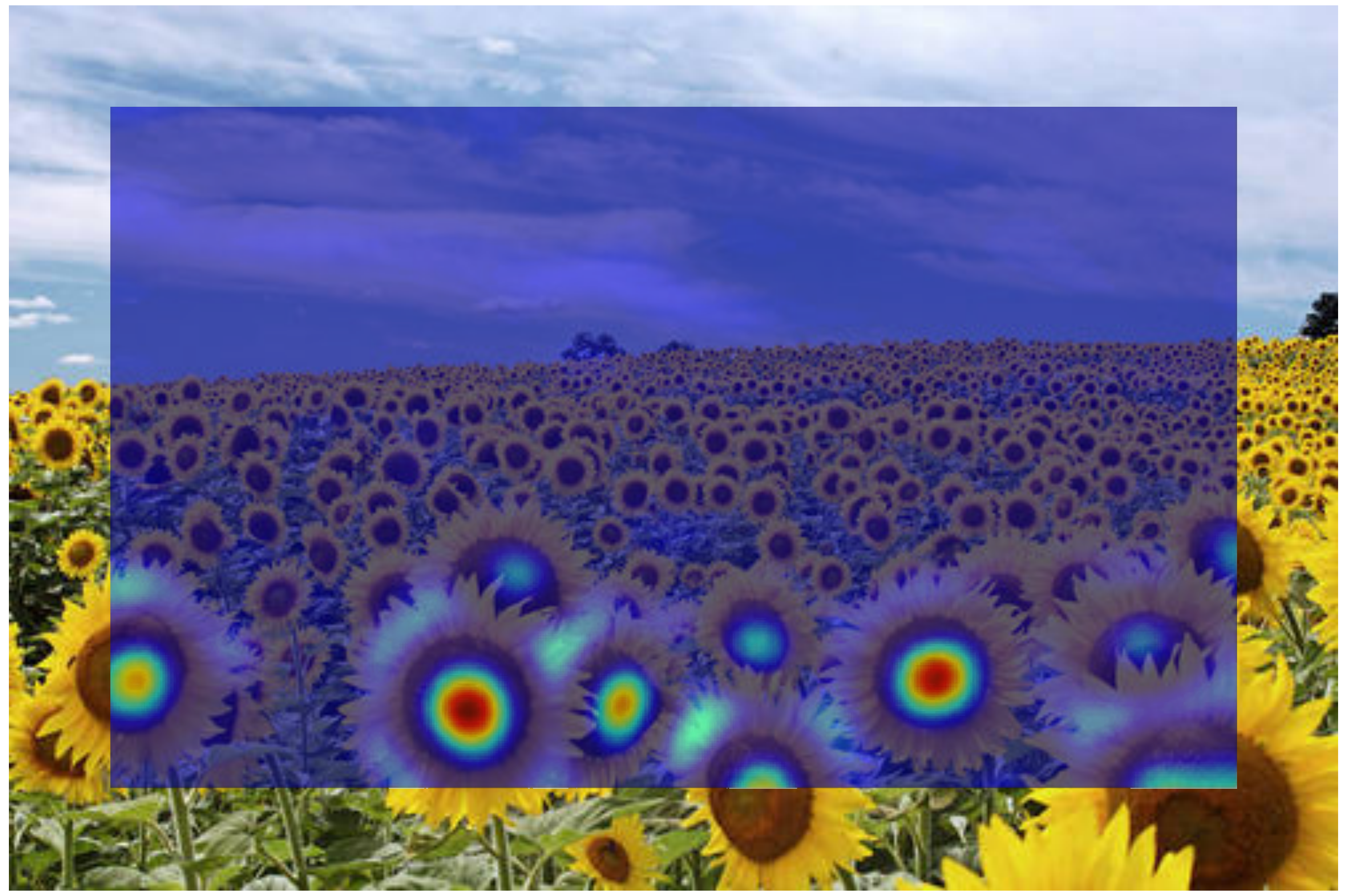
# Applying **Laplacian** Filter at Different **Scales**




**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

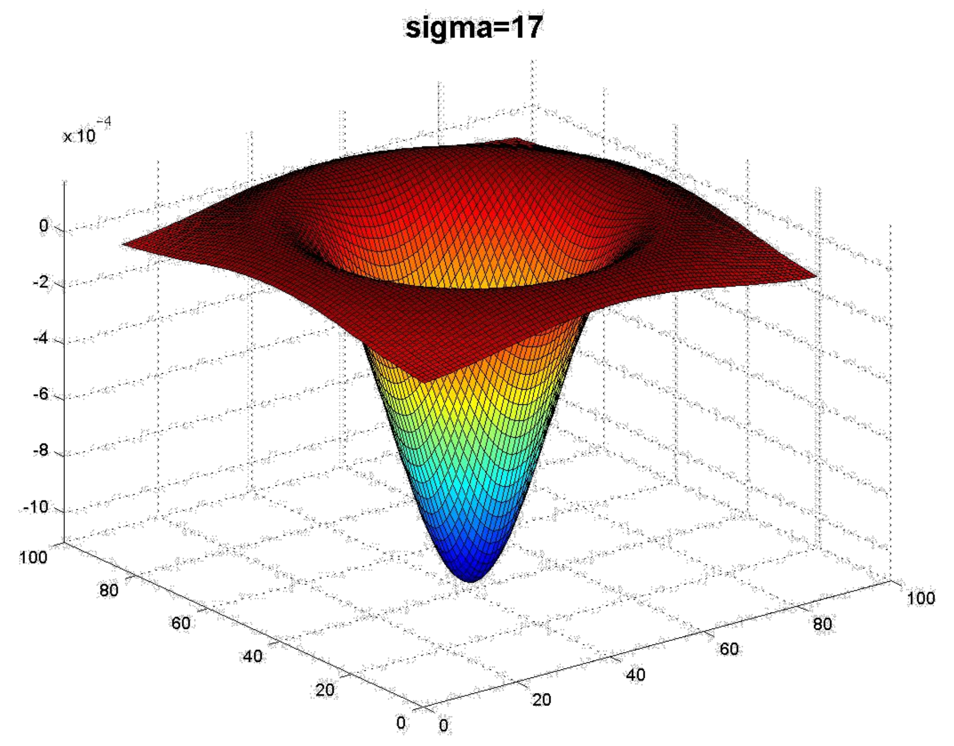
# Applying **Laplacian** Filter at Different **Scales**



**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Applying **Laplacian** Filter at Different **Scales**




**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

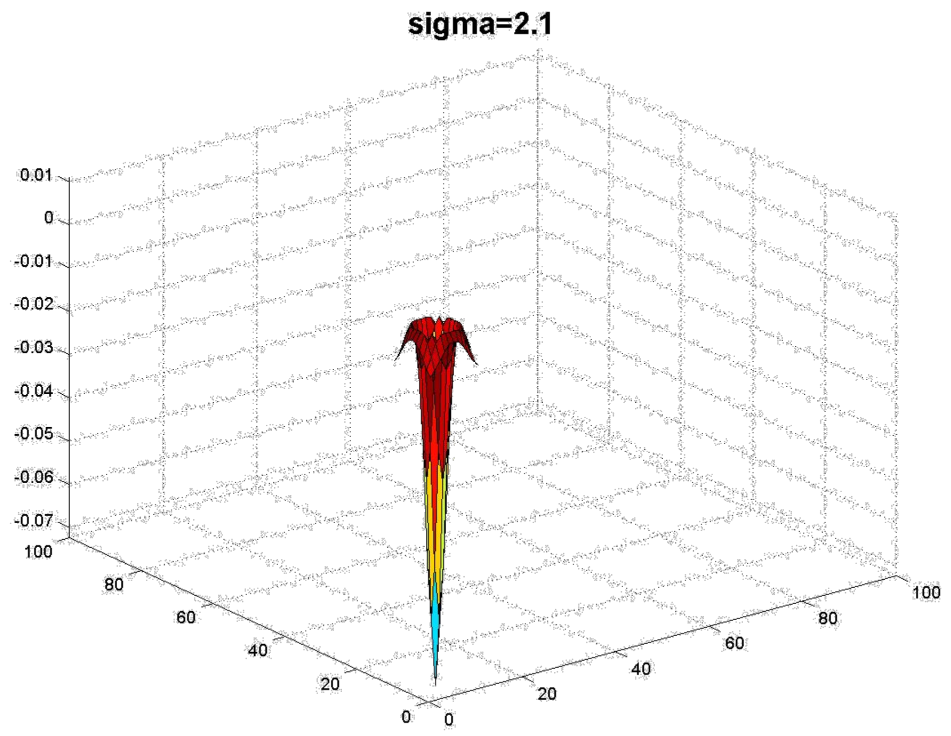
# Applying **Laplacian** Filter at Different **Scales**

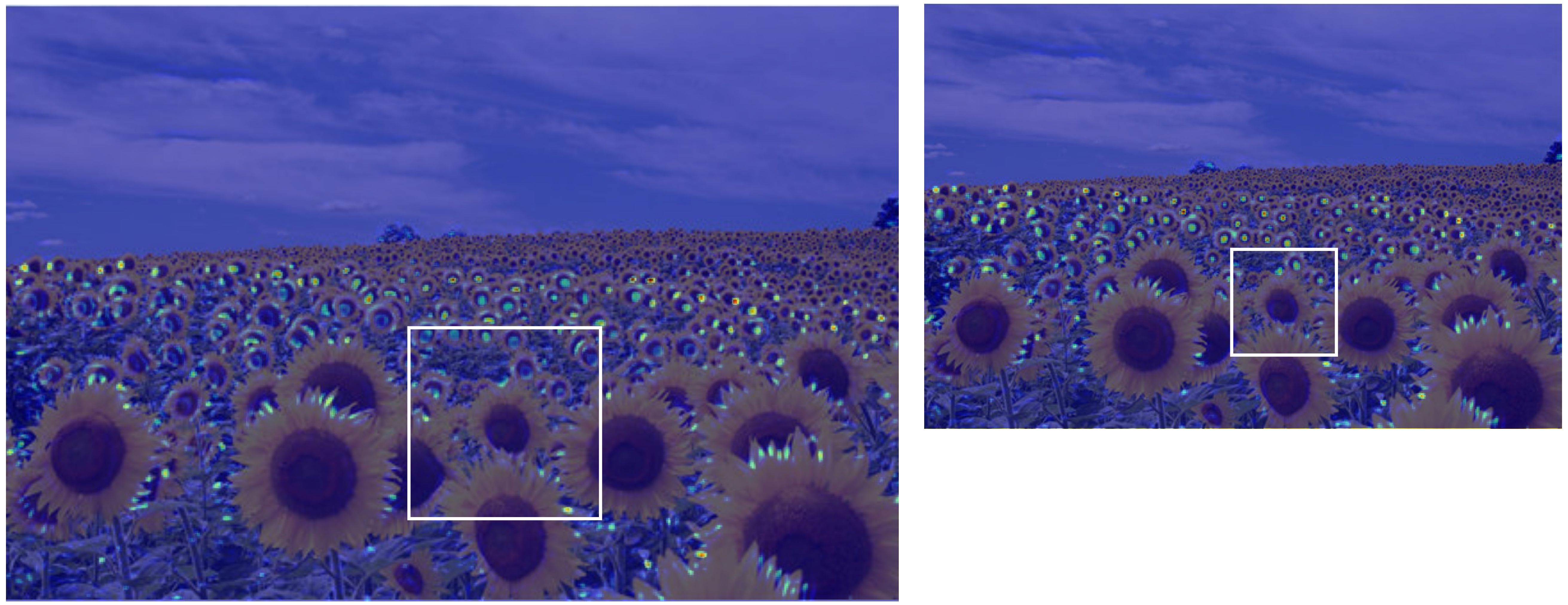
Full size

3/4 size

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

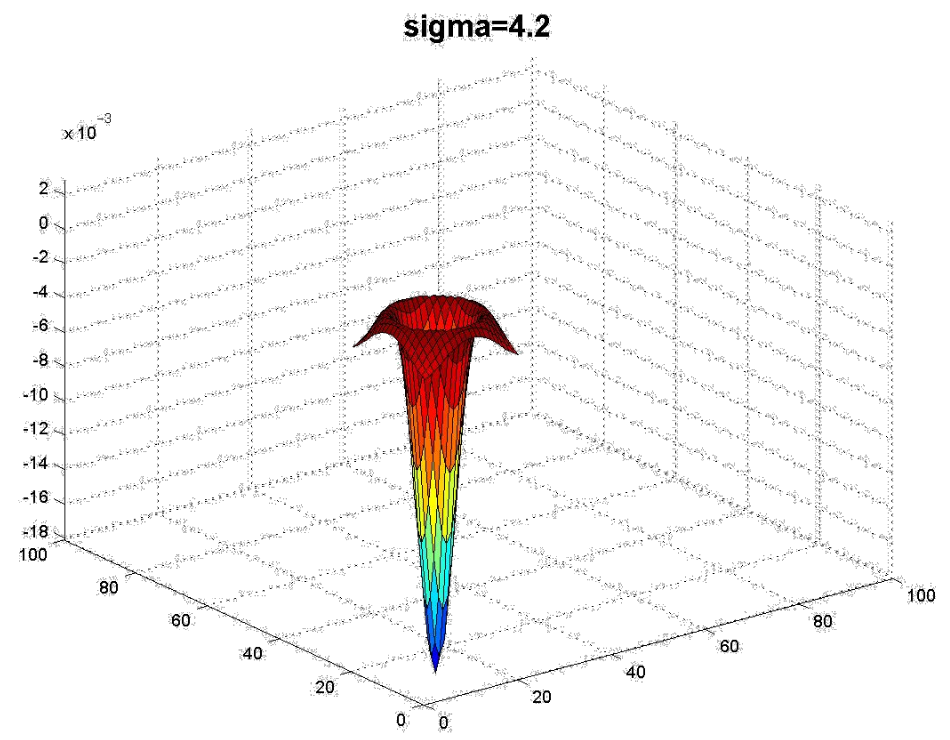
# Applying **Laplacian** Filter at Different **Scales**

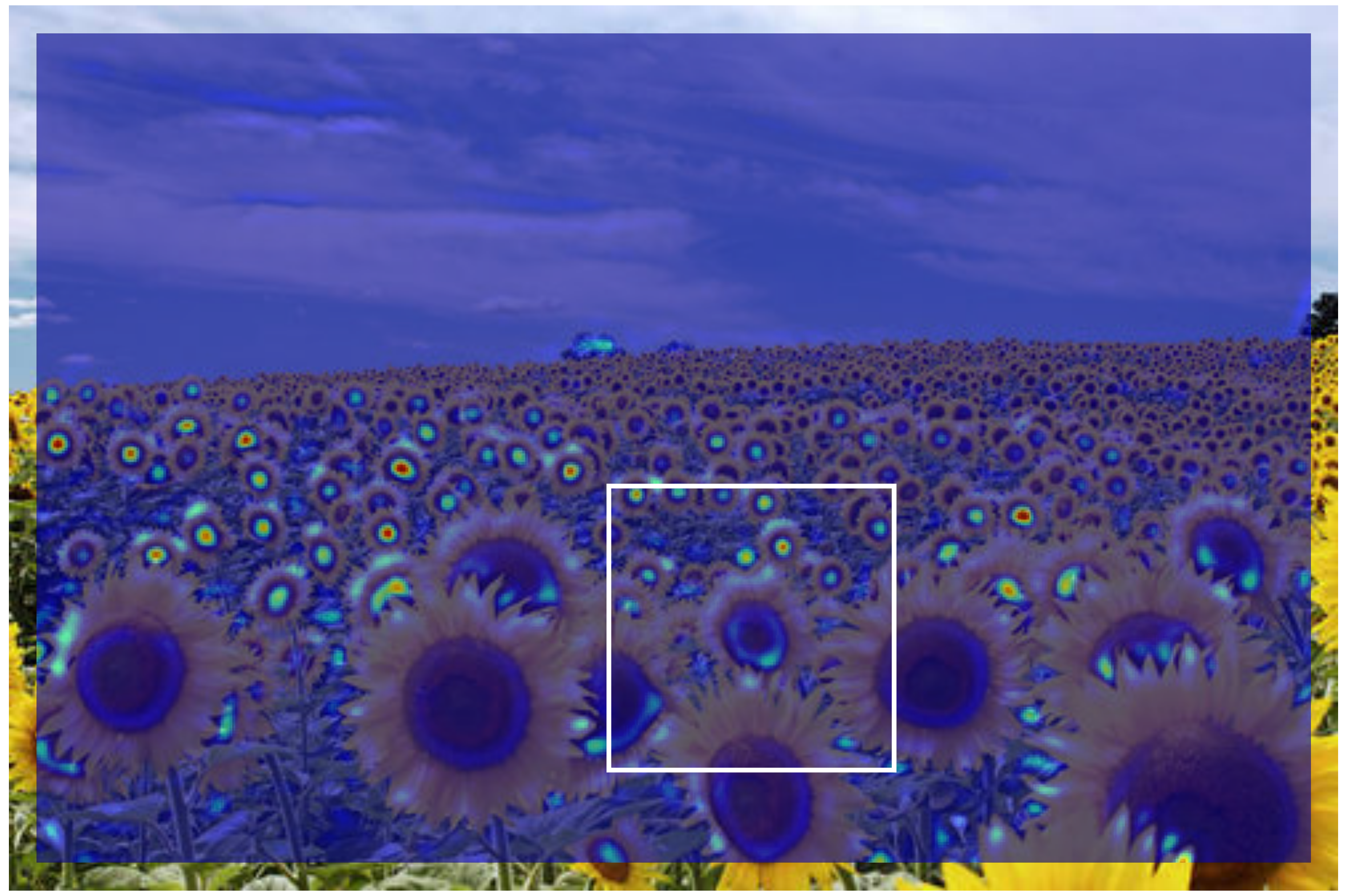


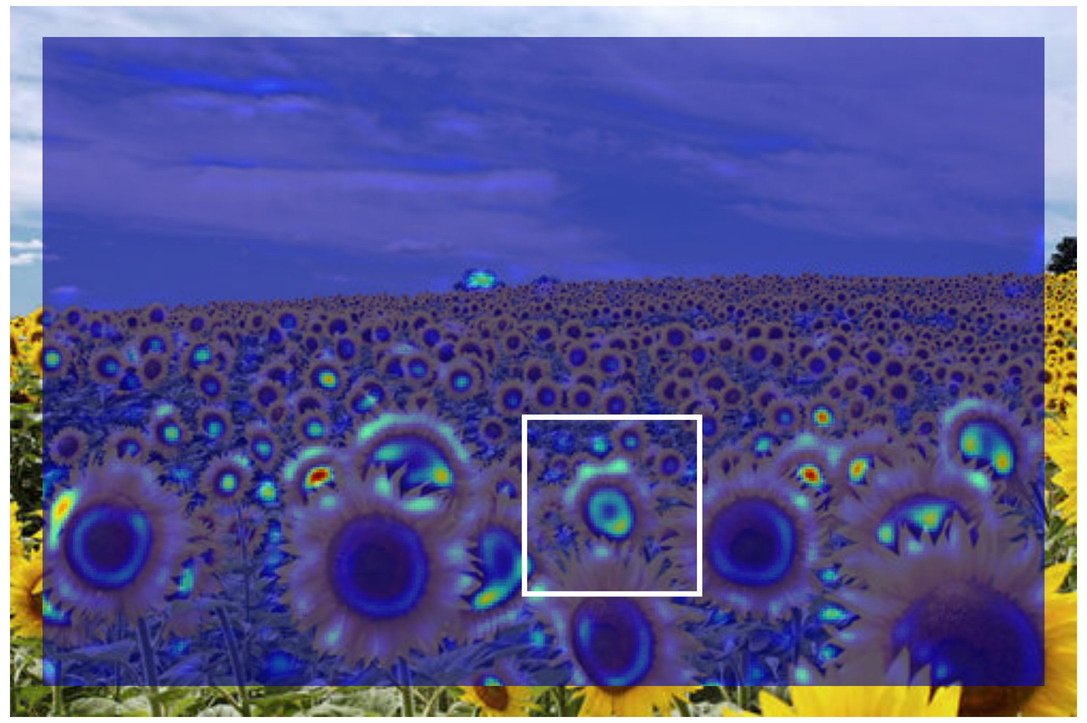



**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Applying **Laplacian** Filter at Different **Scales**

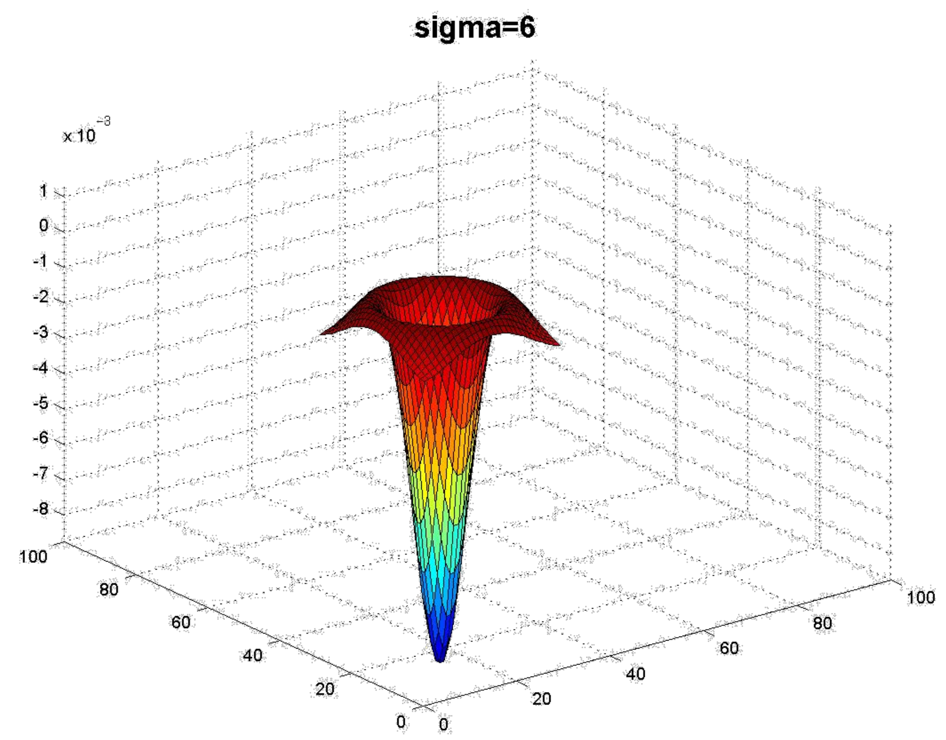


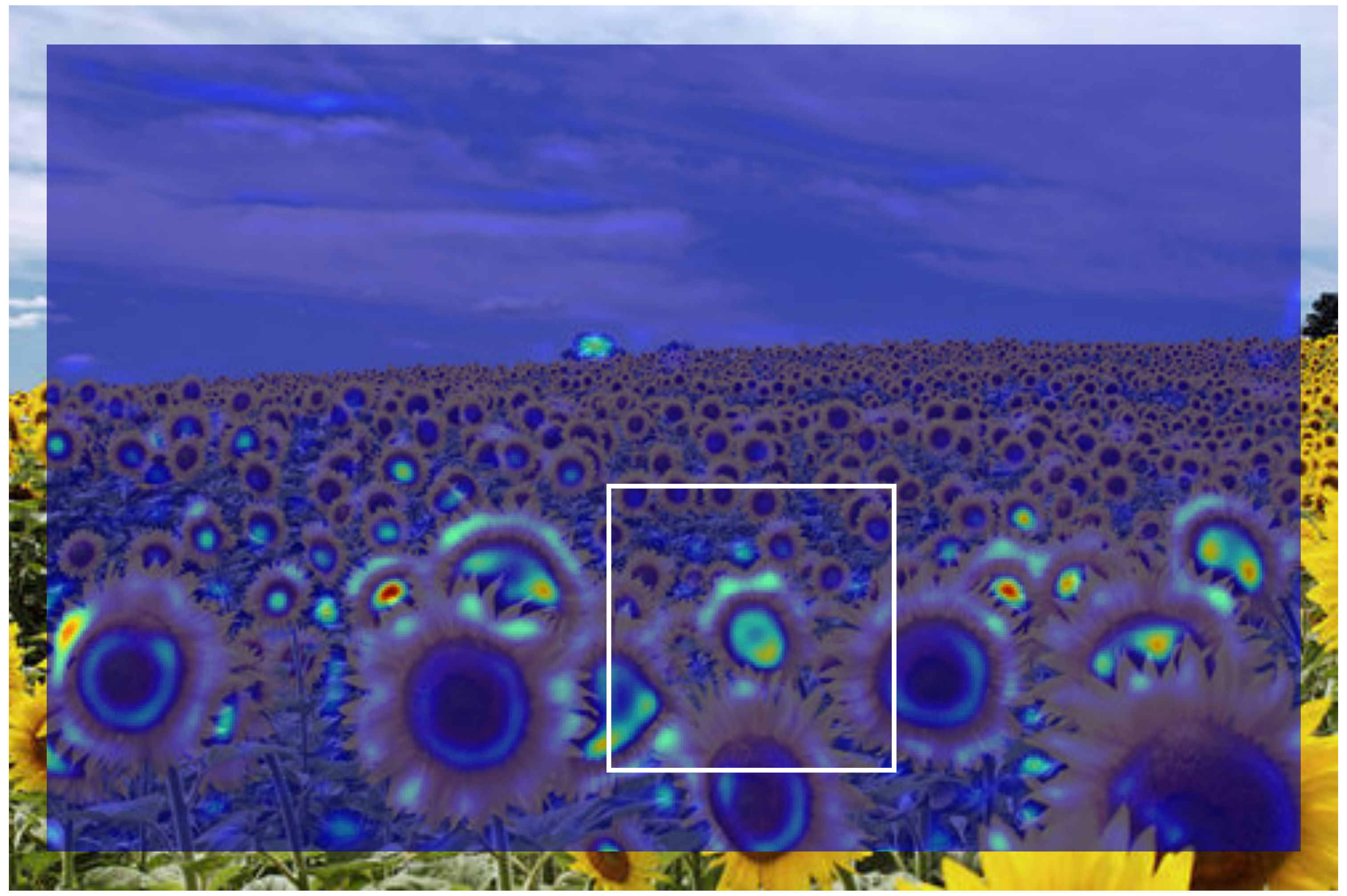



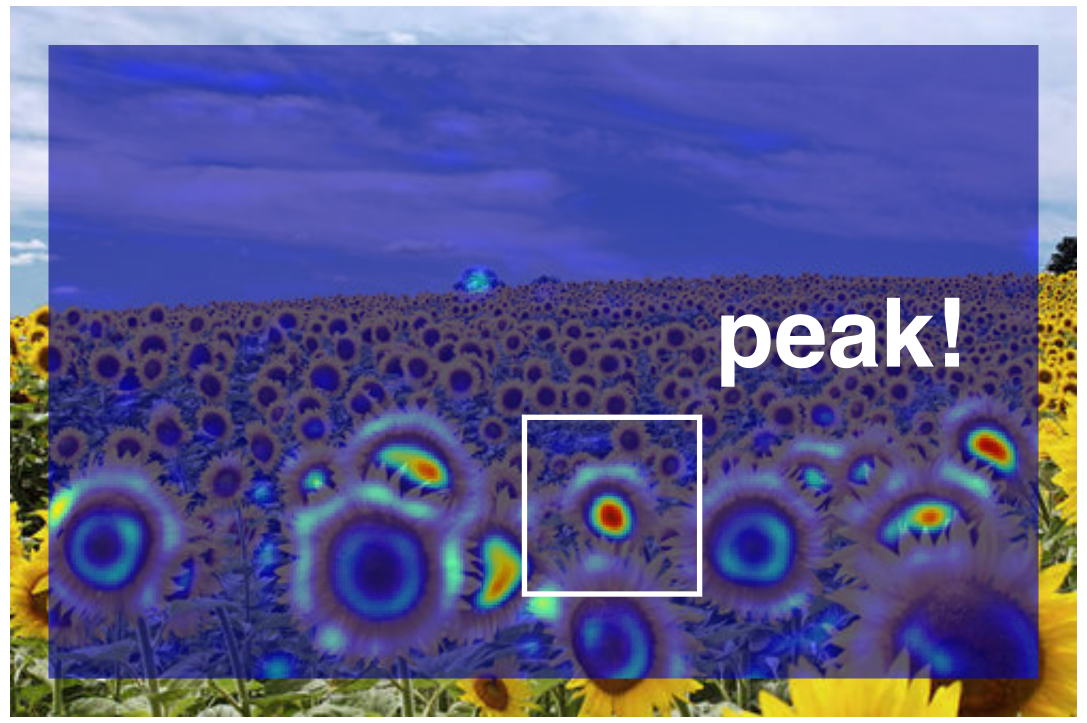


**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Applying **Laplacian** Filter at Different **Scales**

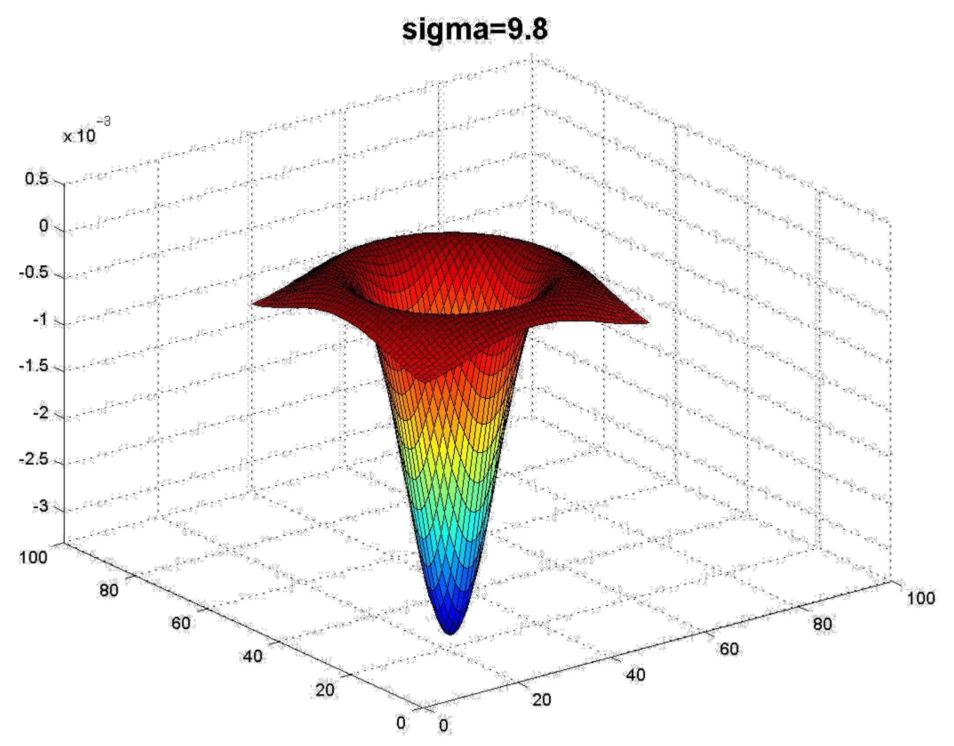


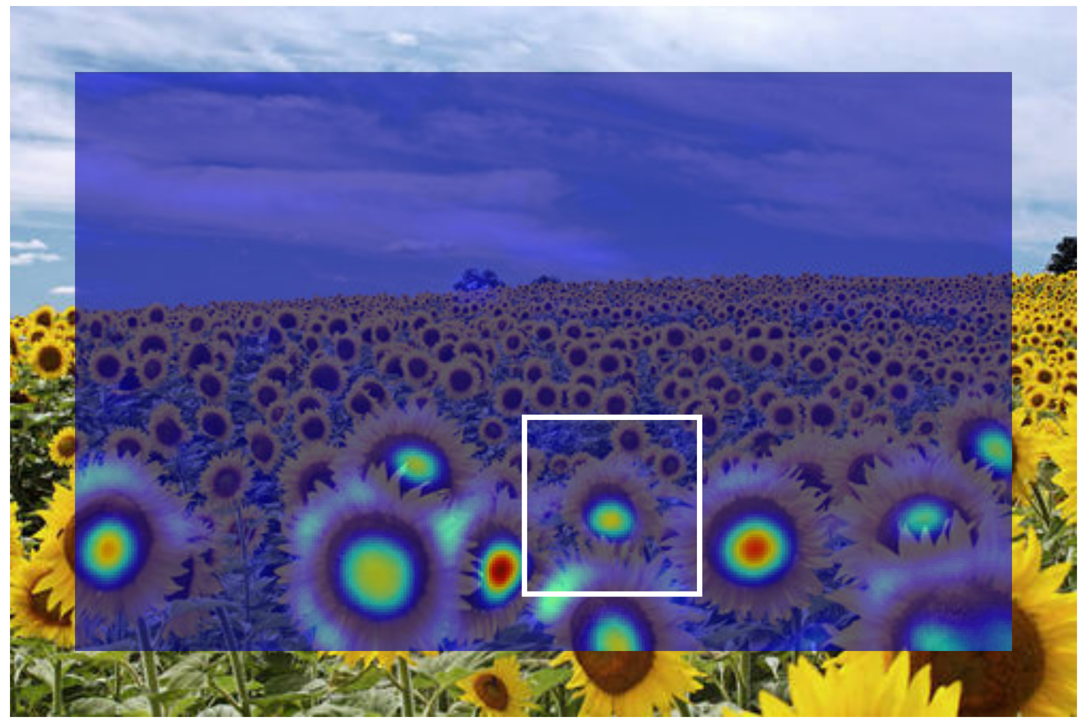



**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Applying **Laplacian** Filter at Different **Scales**

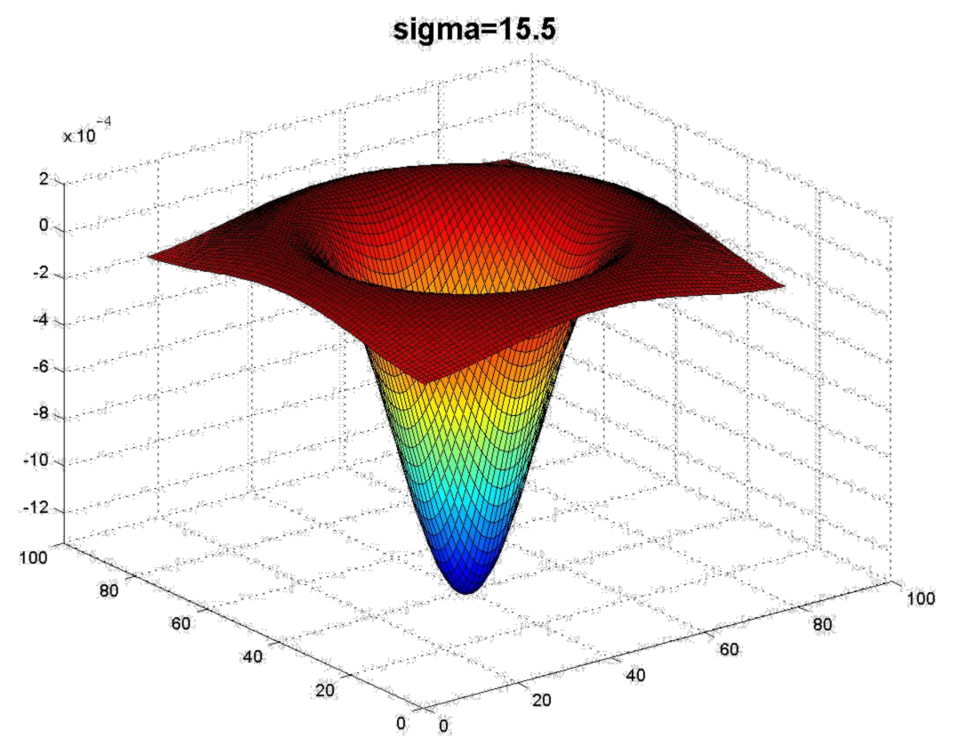


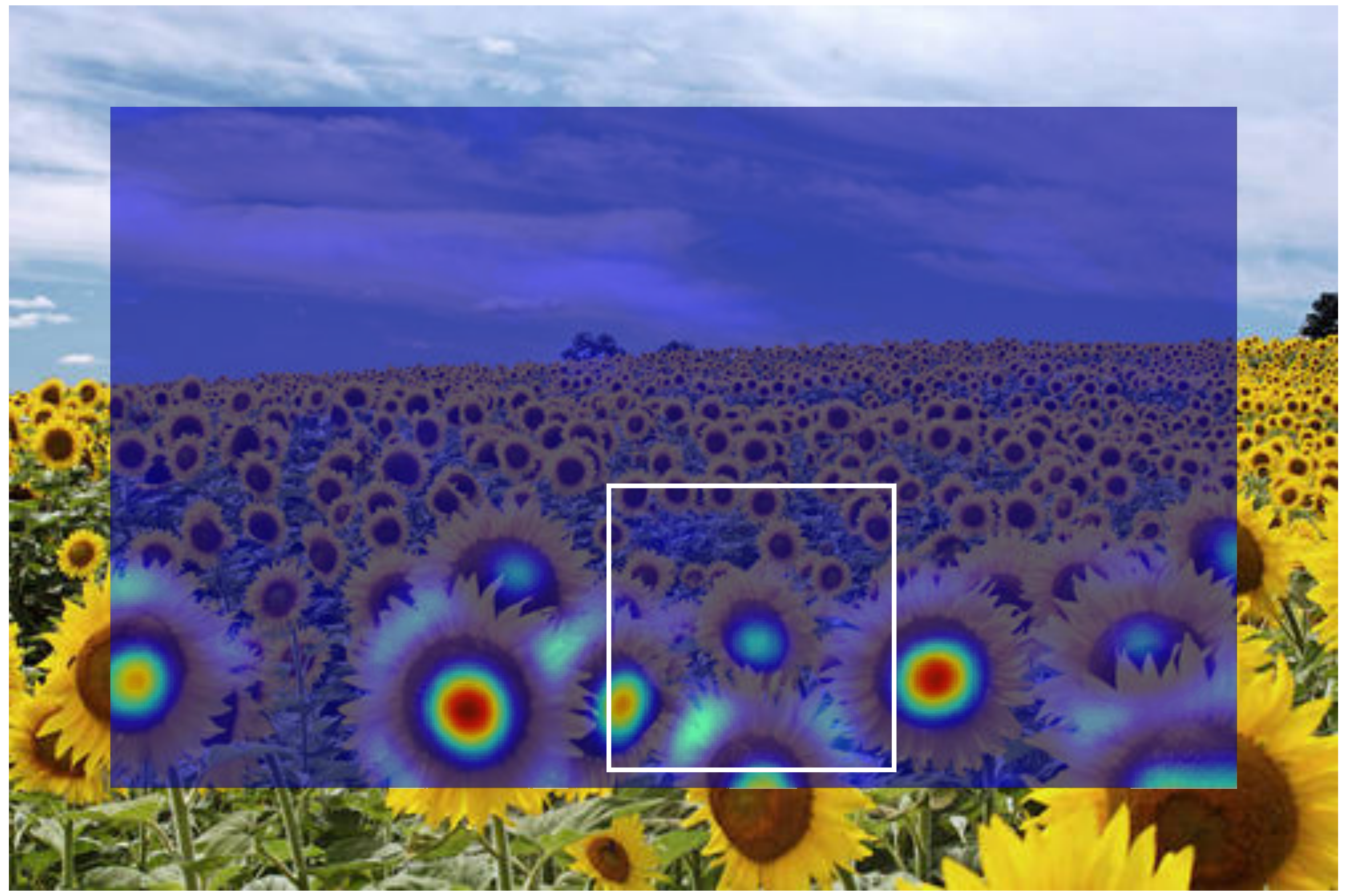


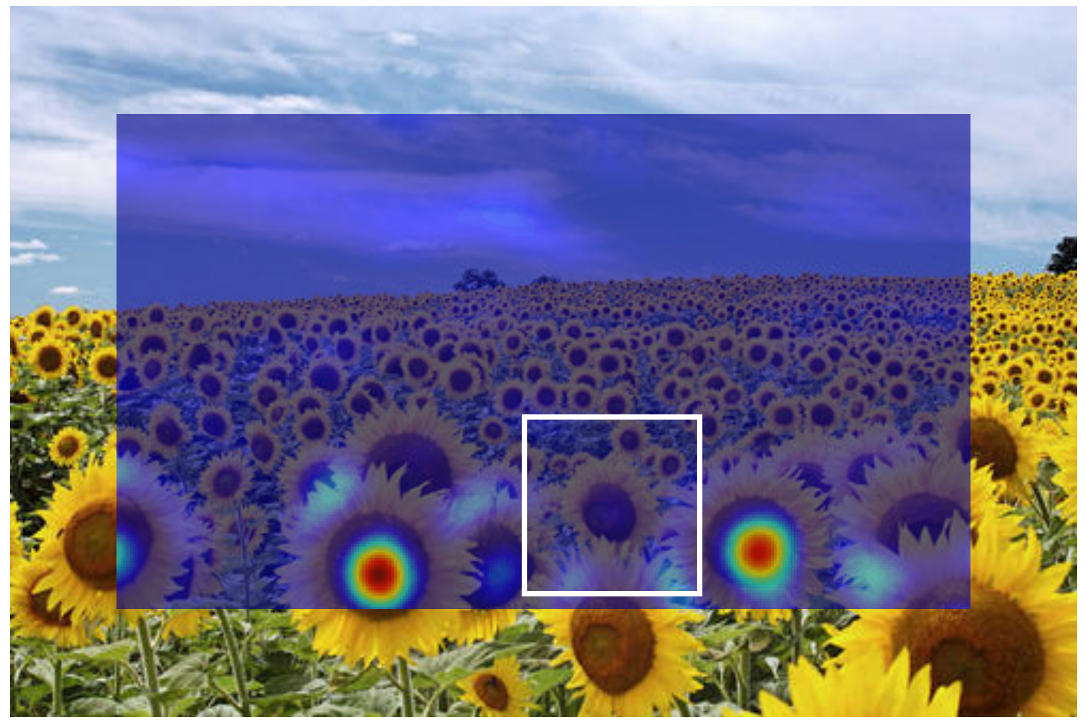


**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Applying **Laplacian** Filter at Different **Scales**

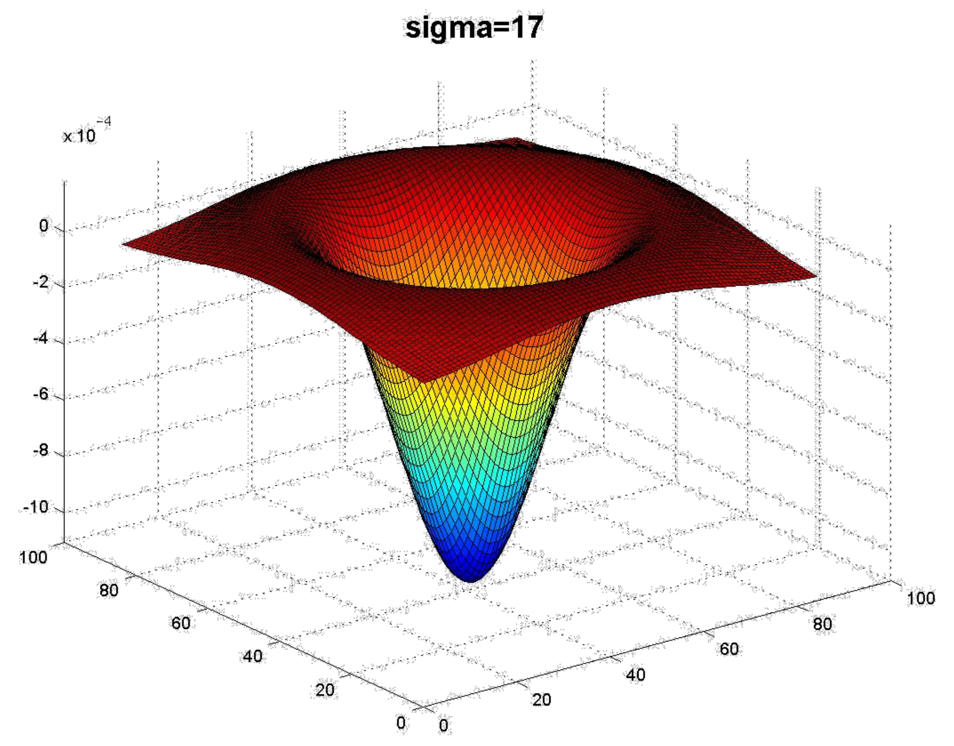


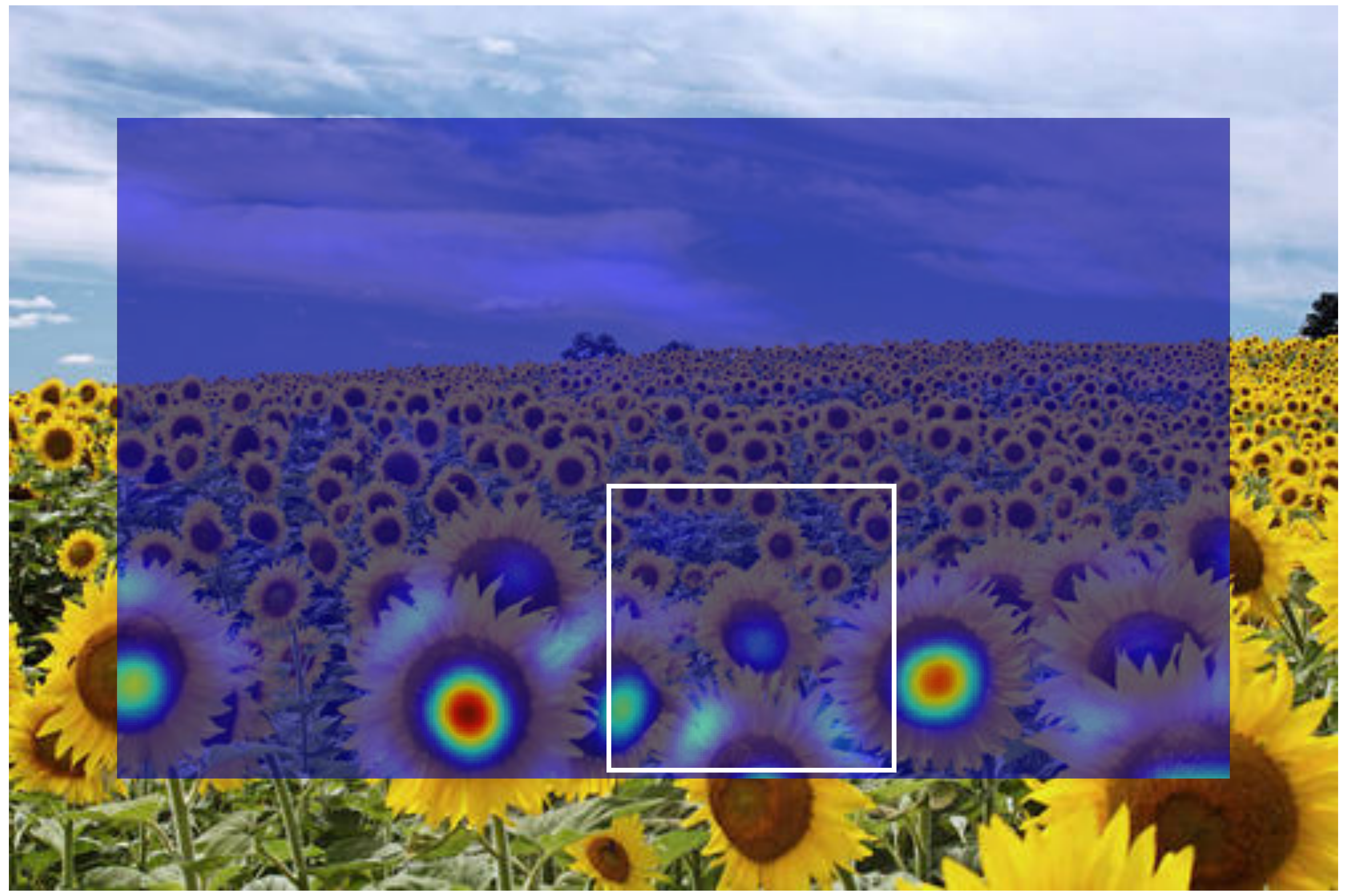



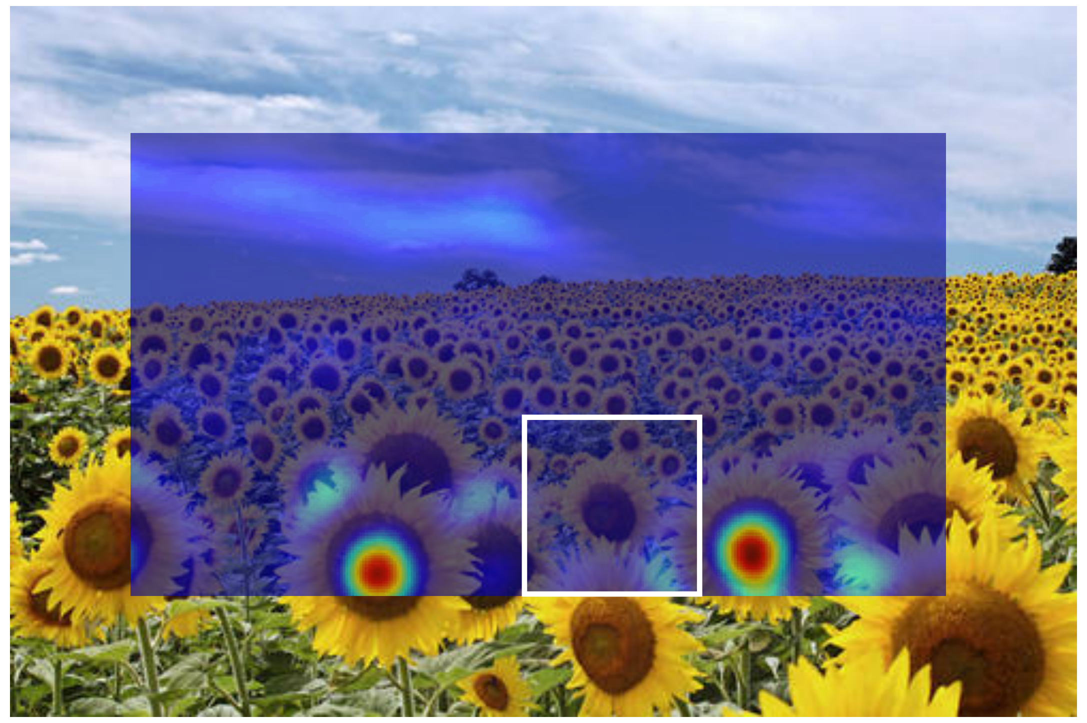


**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Applying **Laplacian** Filter at Different **Scales**





**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Applying **Laplacian** Filter at Different **Scales**

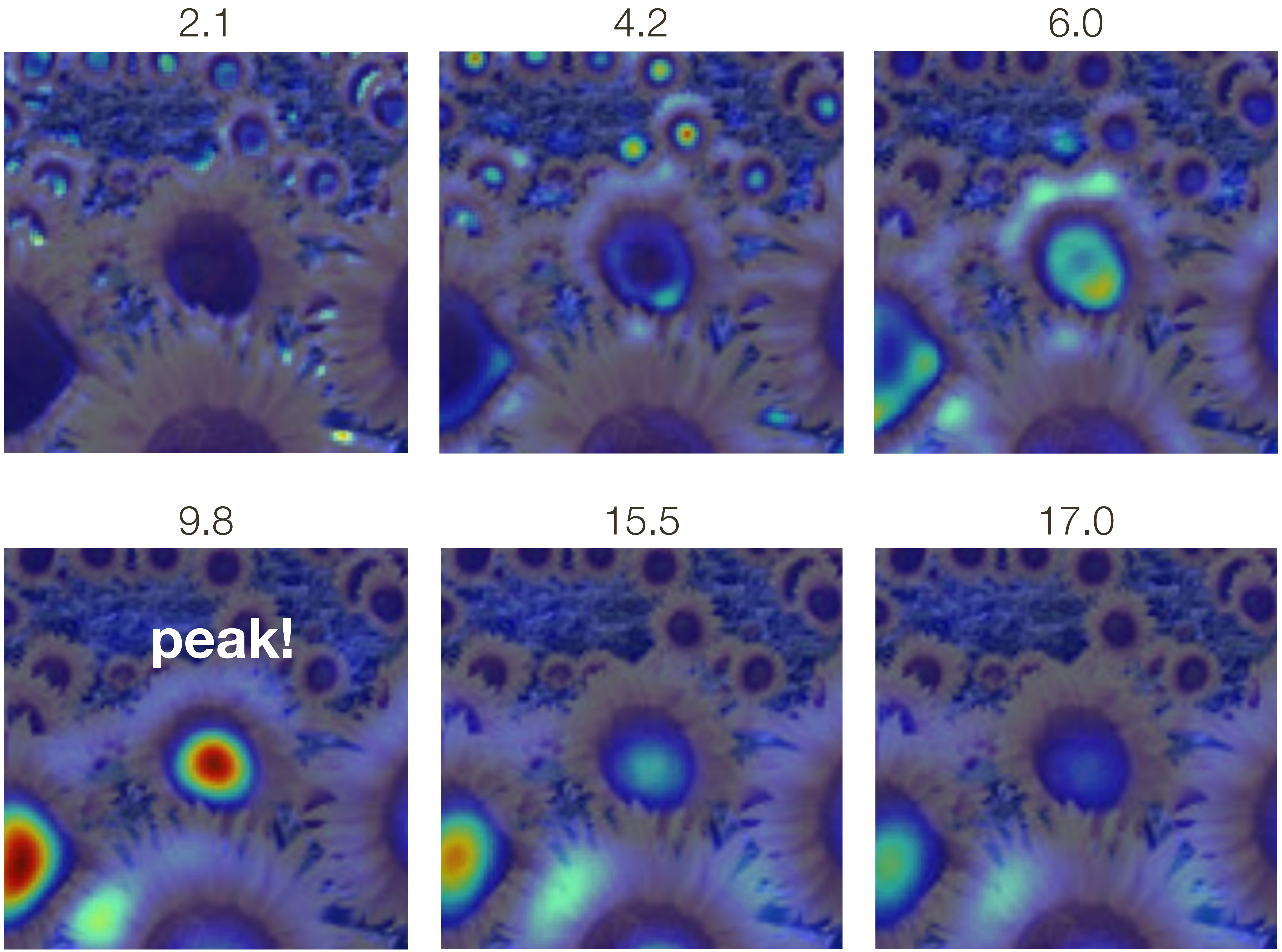
Full size

3/4 size

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

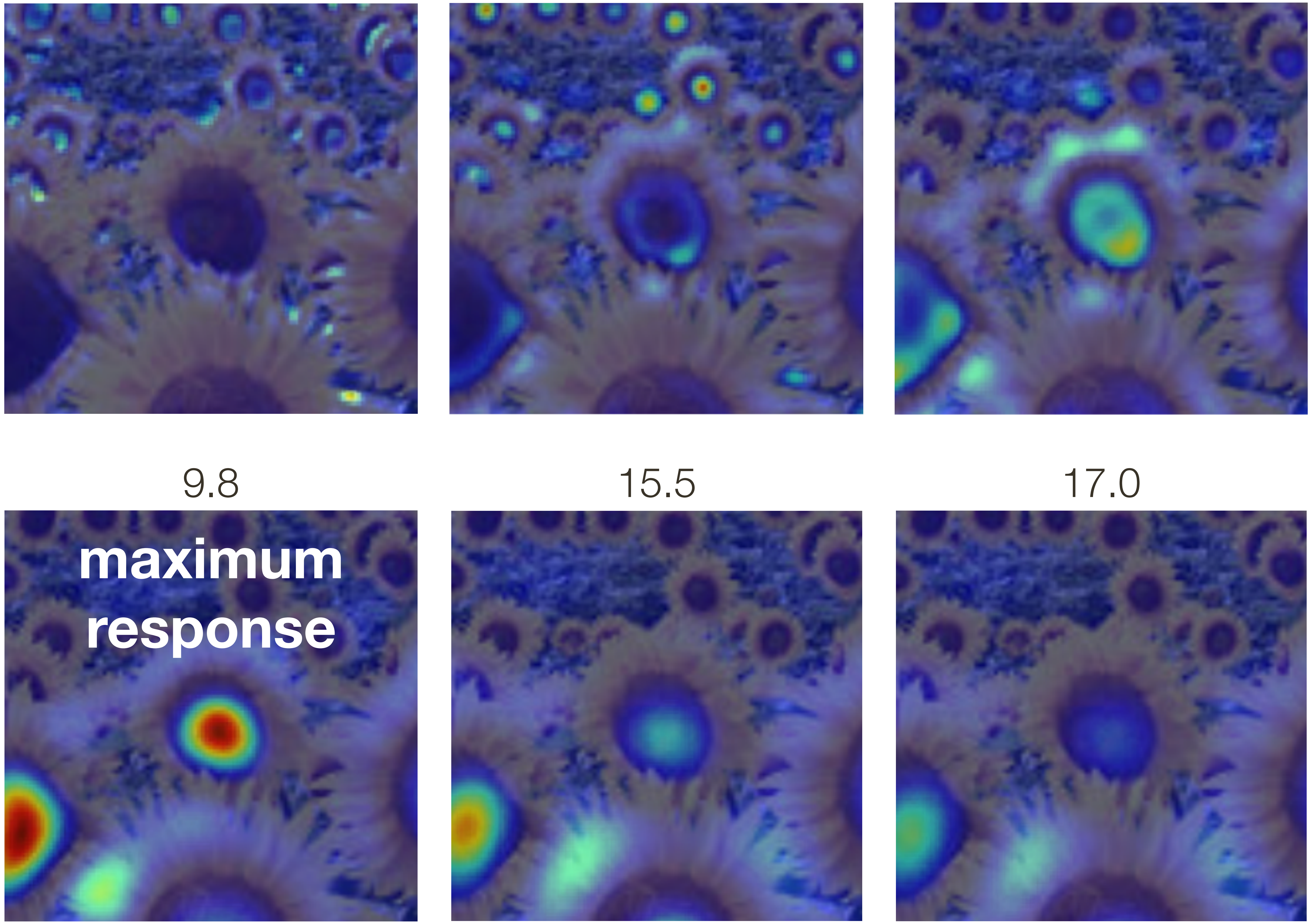
# Applying **Laplacian** Filter at Different **Scales**

Full size

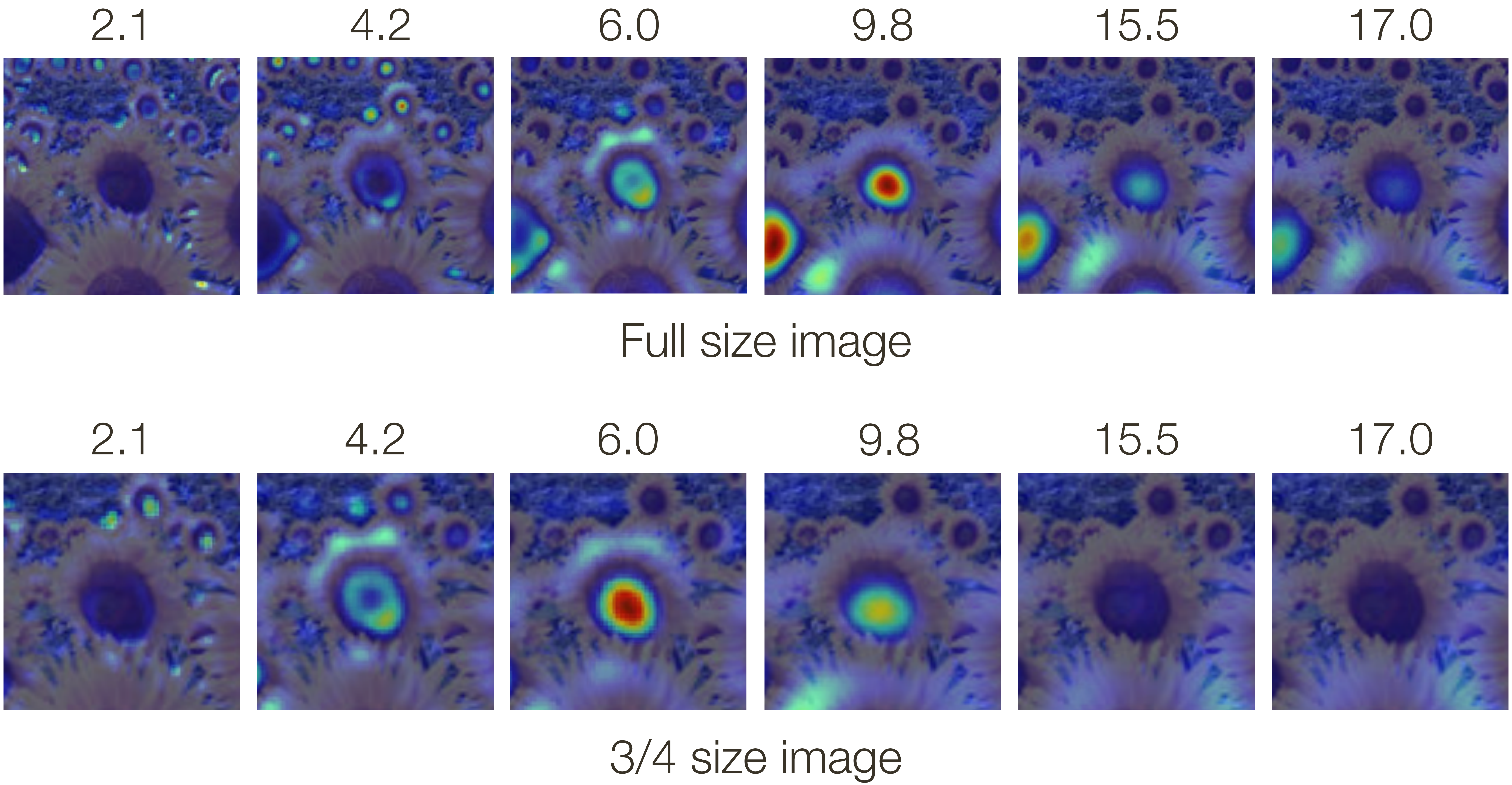


# Applying **Laplacian** Filter at Different **Scales**

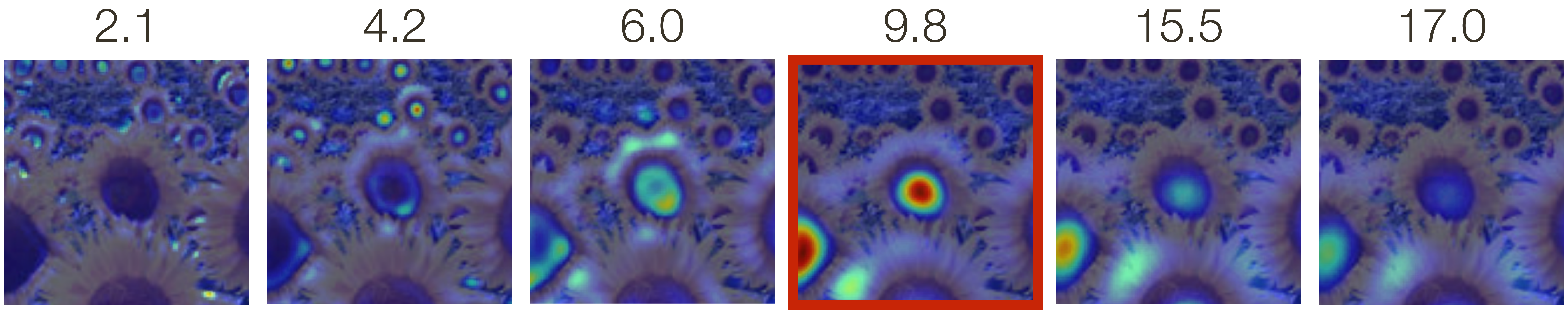
Full size



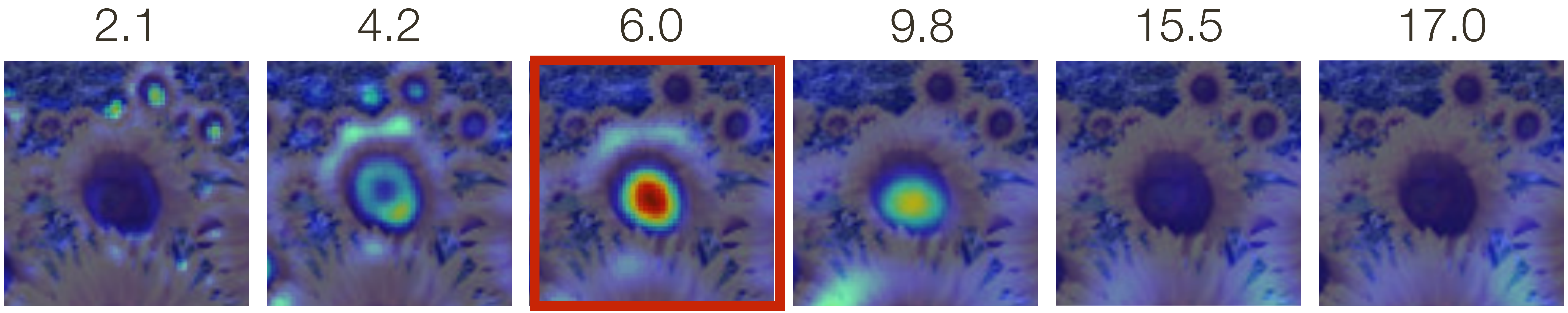
# Optimal **Scale**



# Optimal **Scale**



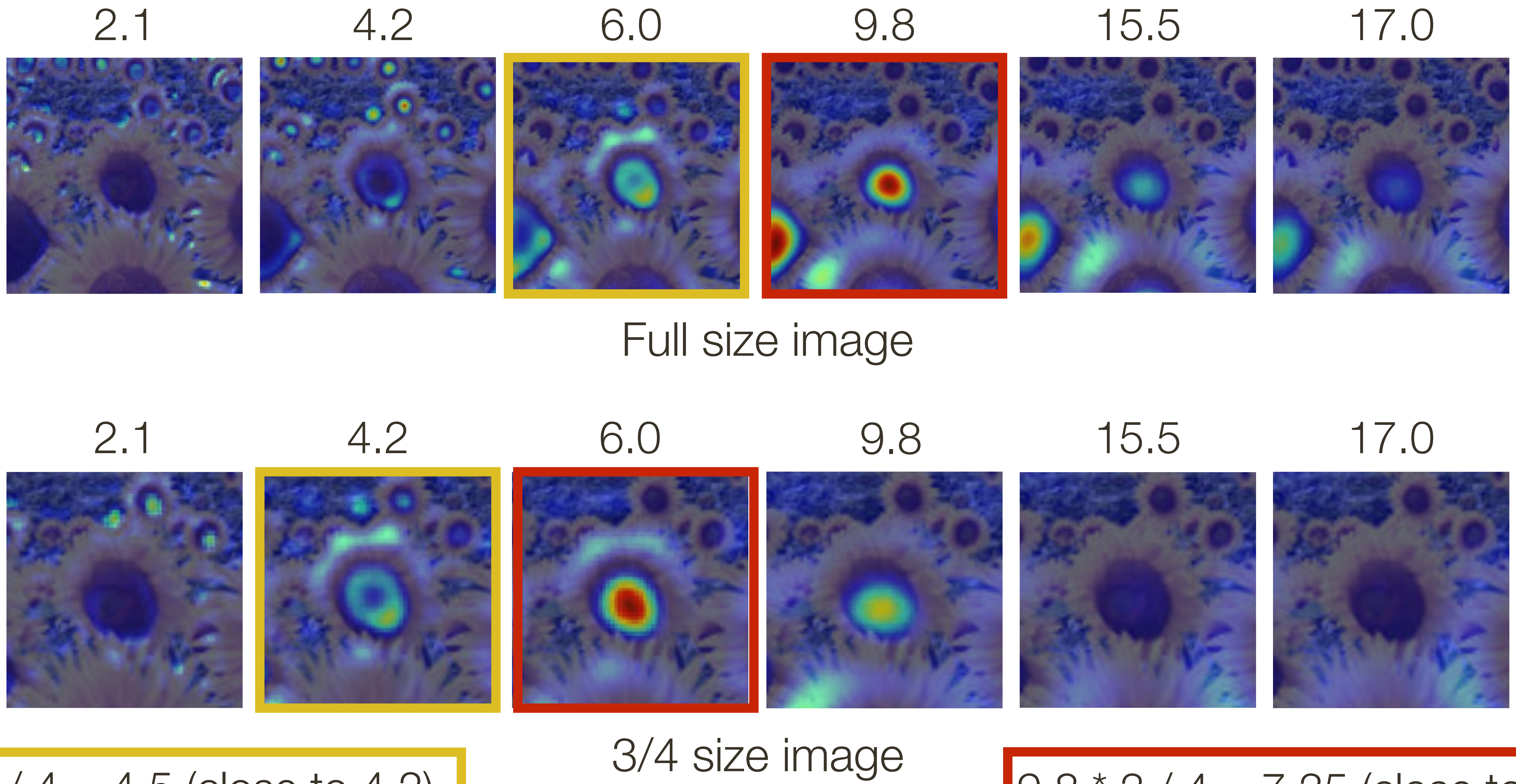

Full size image




3/4 size image

9.8 * 3 / 4 = 7.35 (close to 6.0)

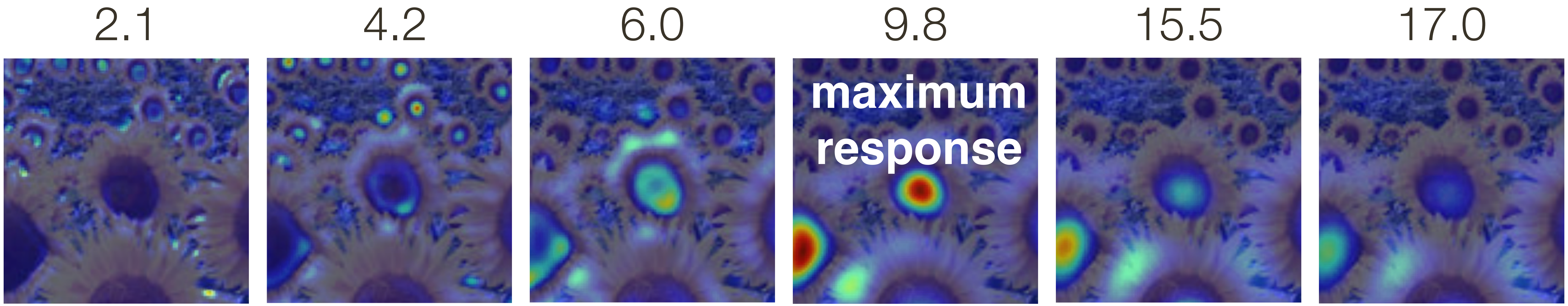
# Optimal **Scale**



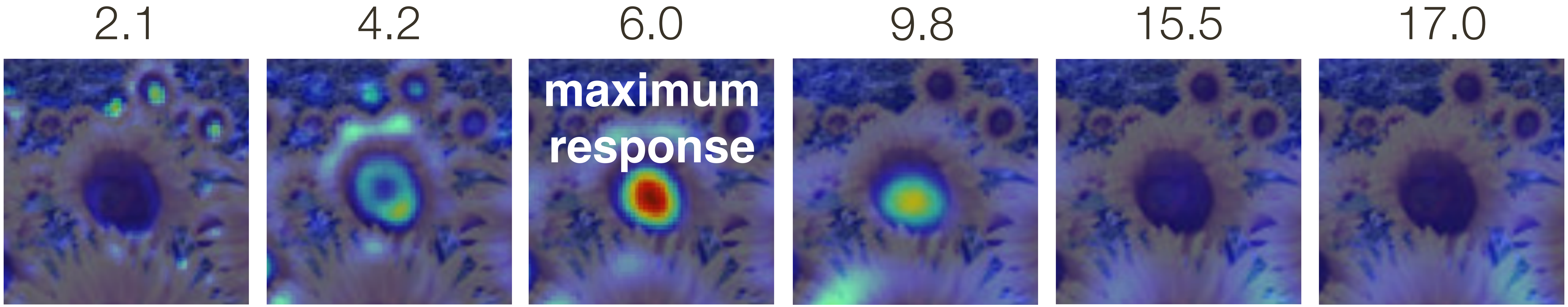

6 * 3 / 4 = 4.5 (close to 4.2)

9.8 * 3 / 4 = 7.35 (close to 6.0)

# Optimal **Scale**




Full size image

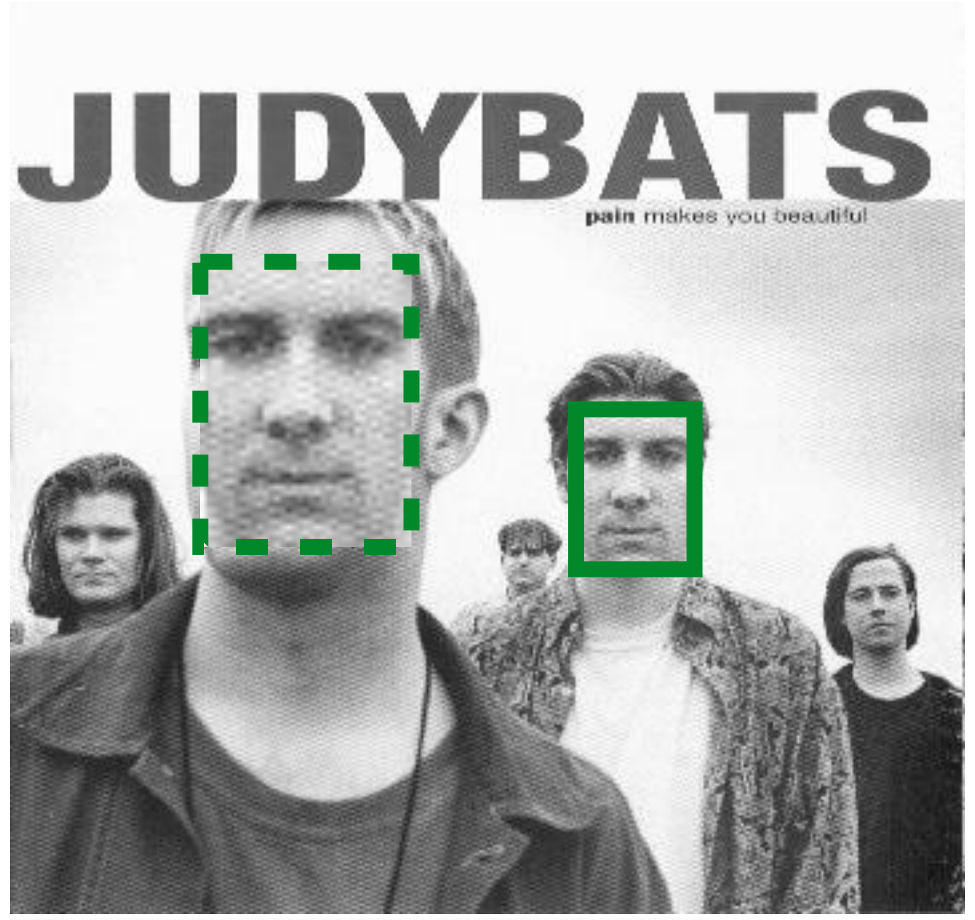



3/4 size image

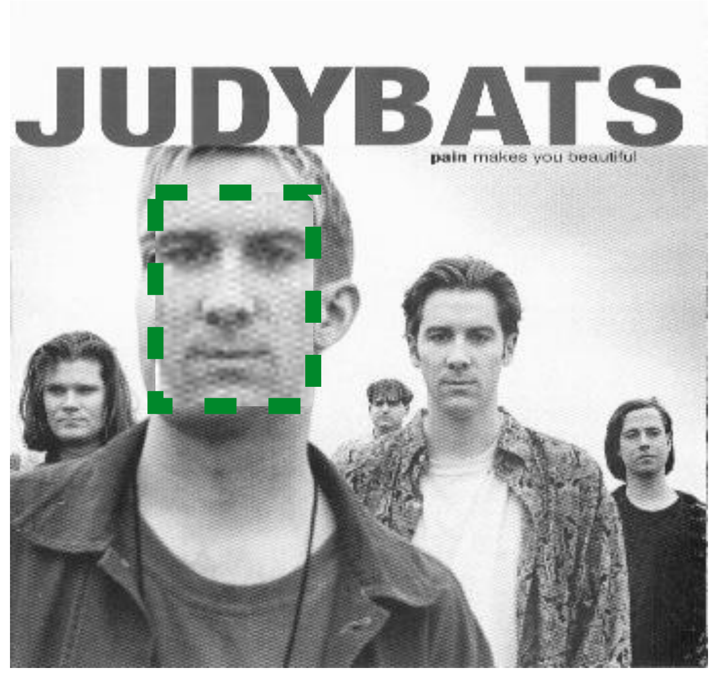
# **Recall: Template** matching

**Level** | **Image** Pyramid (s) | **Template**

0

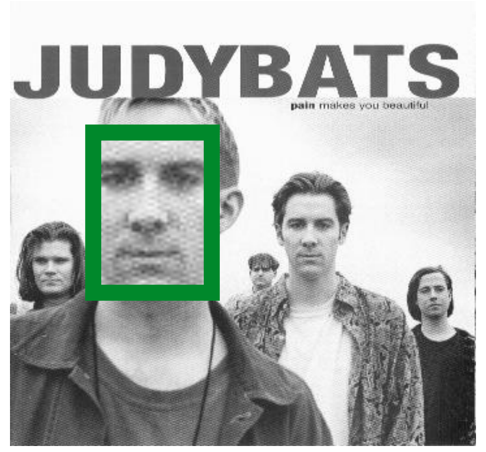



1



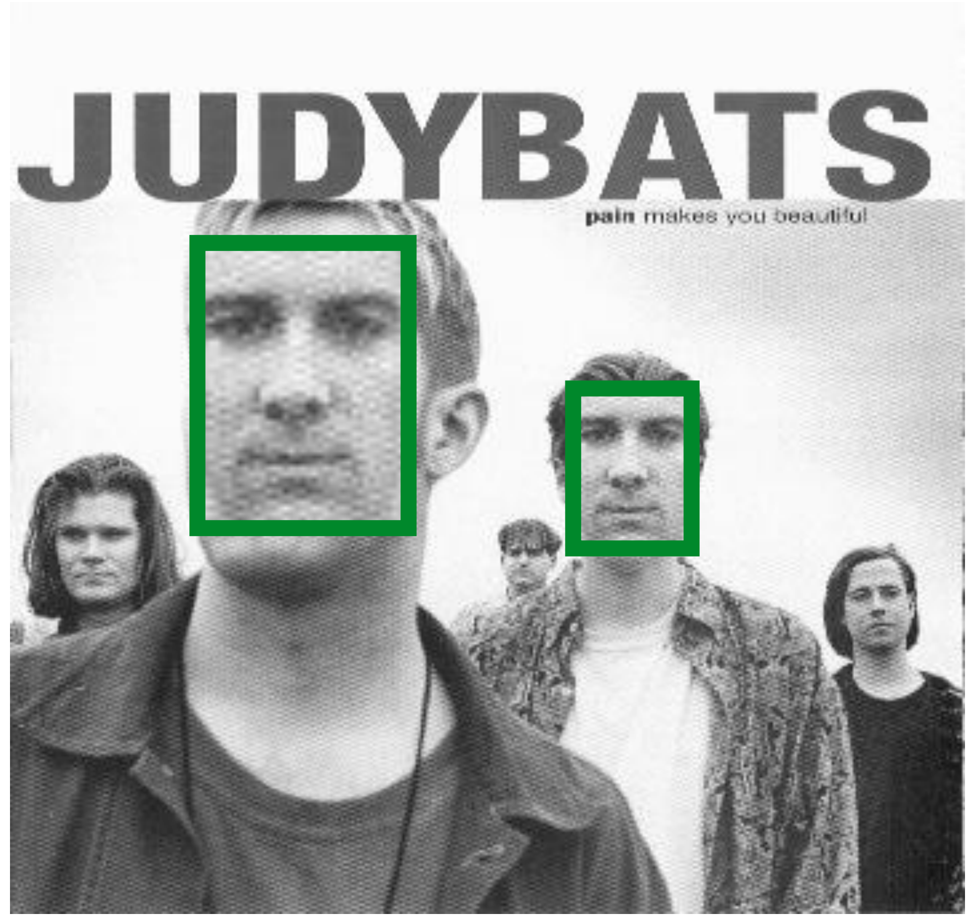

...

L




**Template** Pyramid (1/s) | **Image**

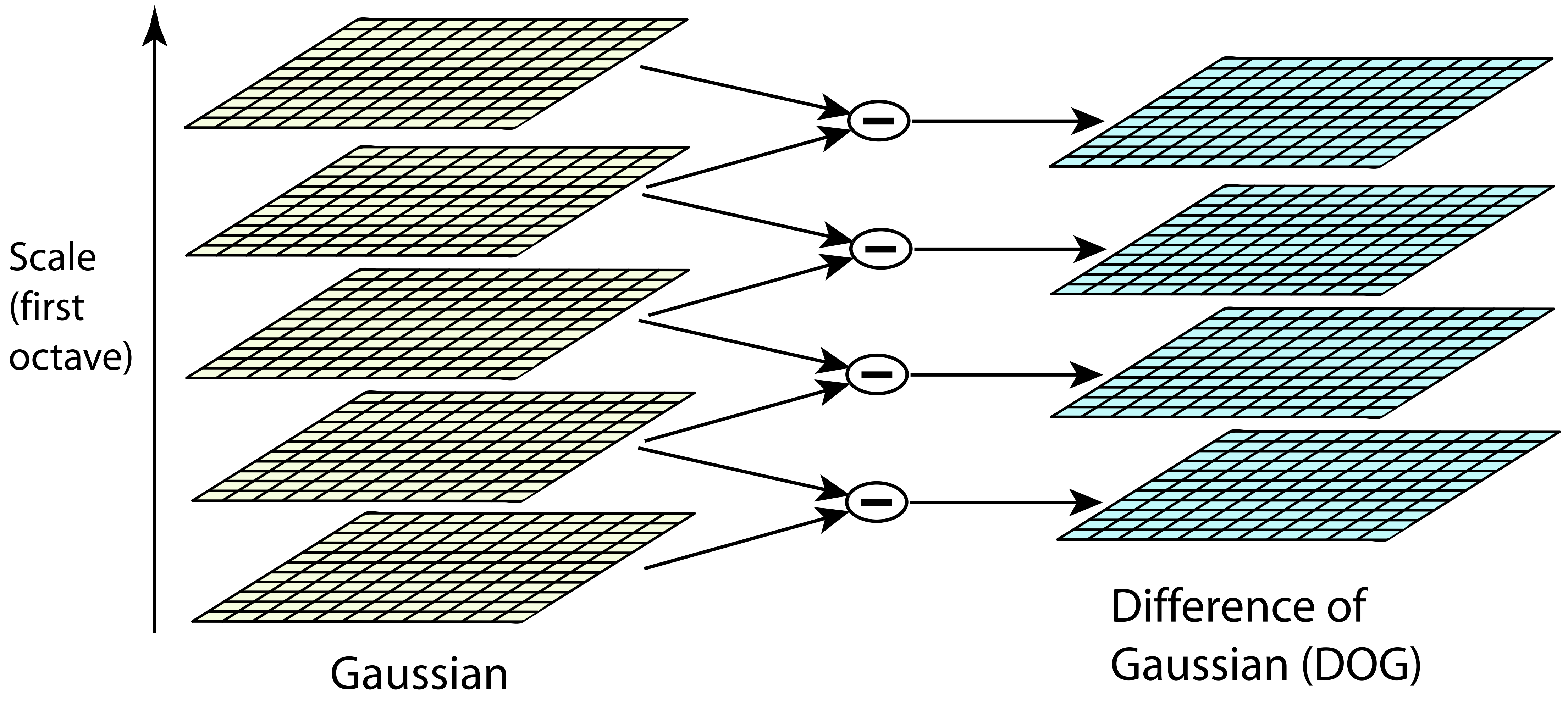
...

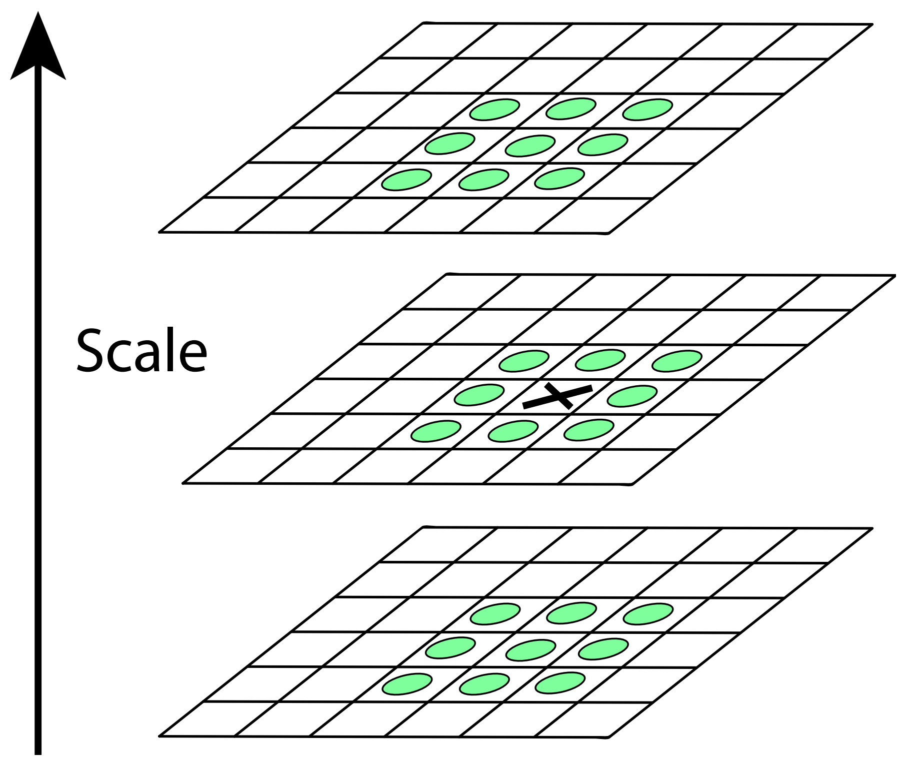



Both allow **search over scale**

# **Scale** Selection

A DOG (Laplacian) Pyramid is formed with multiple scales per ocatve

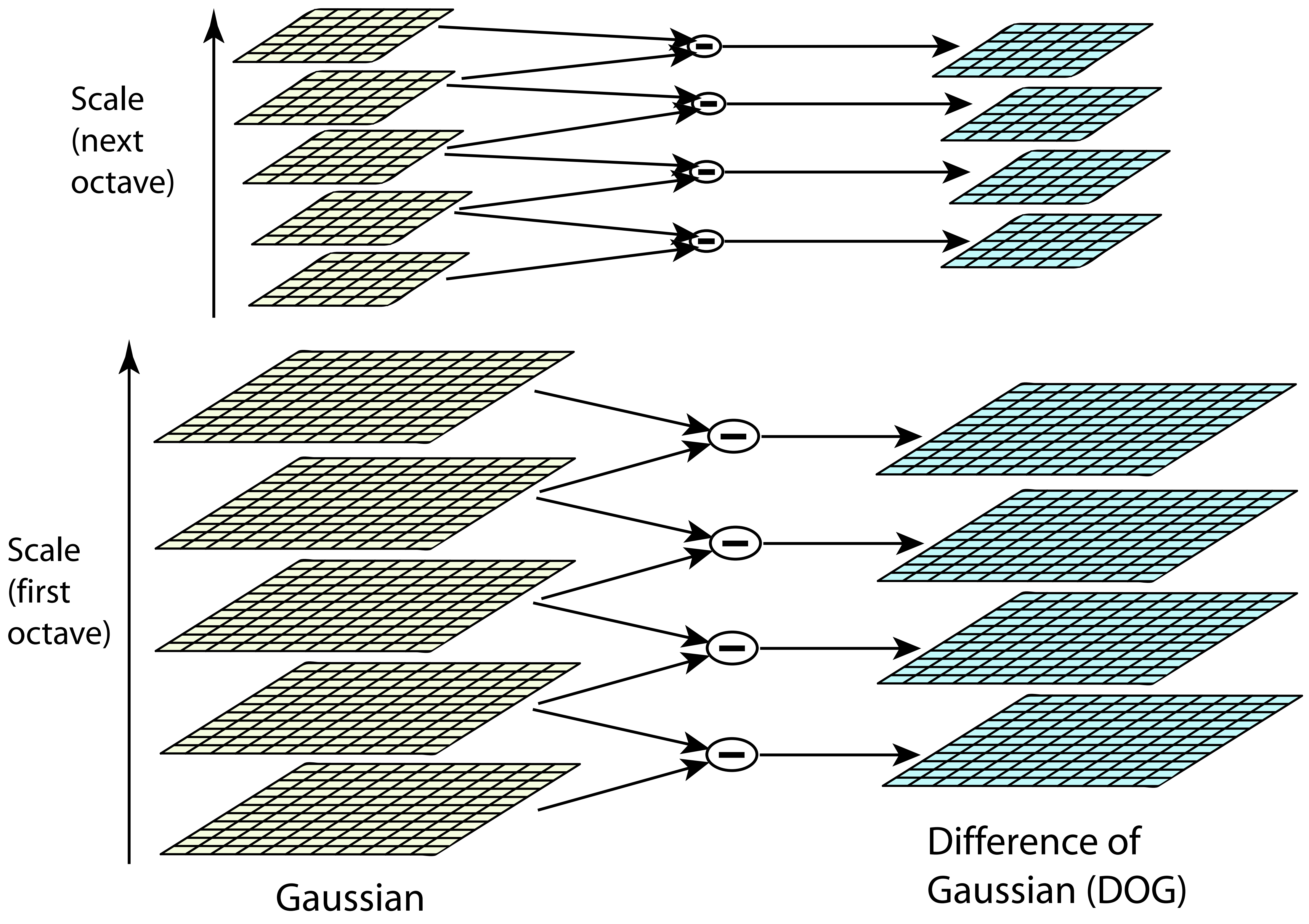


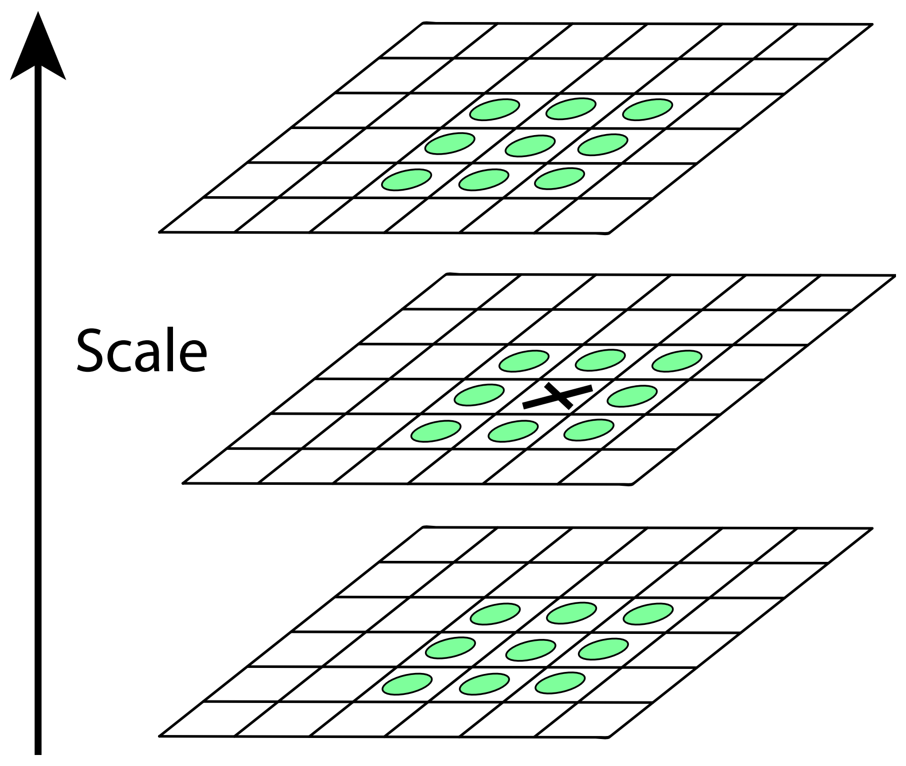




Detections are local maxima in a 3x3x3 scale-space window

# **Scale** Selection

A DOG (Laplacian) Pyramid is formed with multiple scales per ocatve

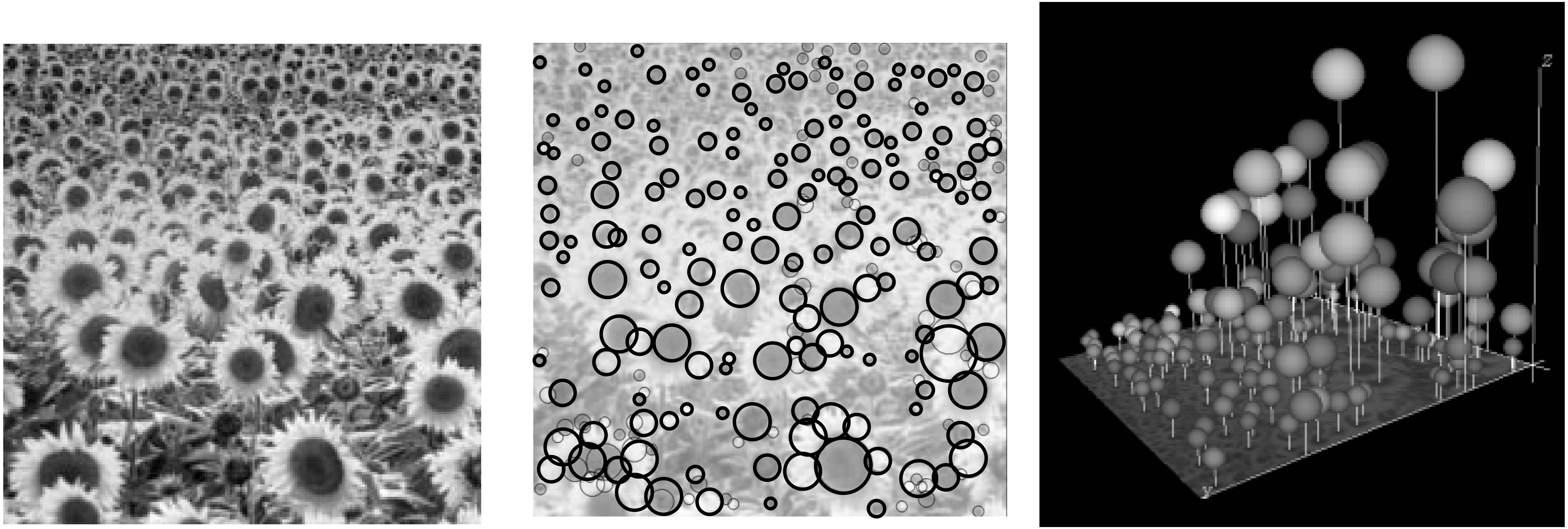



Detections are local maxima in a 3x3x3 scale-space window

# **Scale** Selection

Maximising the DOG function in scale as well as space performs scale selection



[ T. Lindeberg ]

# **Difference of Gaussian** blobs in 2020

Harris & Stephens (1988)

$$\det(C) - \kappa \text{trace}^2(C)$$

Kanade & Tomasi (1994)

$$\min(\lambda_1, \lambda_2)$$

Nobel (1998)

$$\frac{\det(C)}{\text{trace}(C) + \epsilon}$$

# **Difference of Gaussian** blobs in 2020

Harris & Stephens (1988)

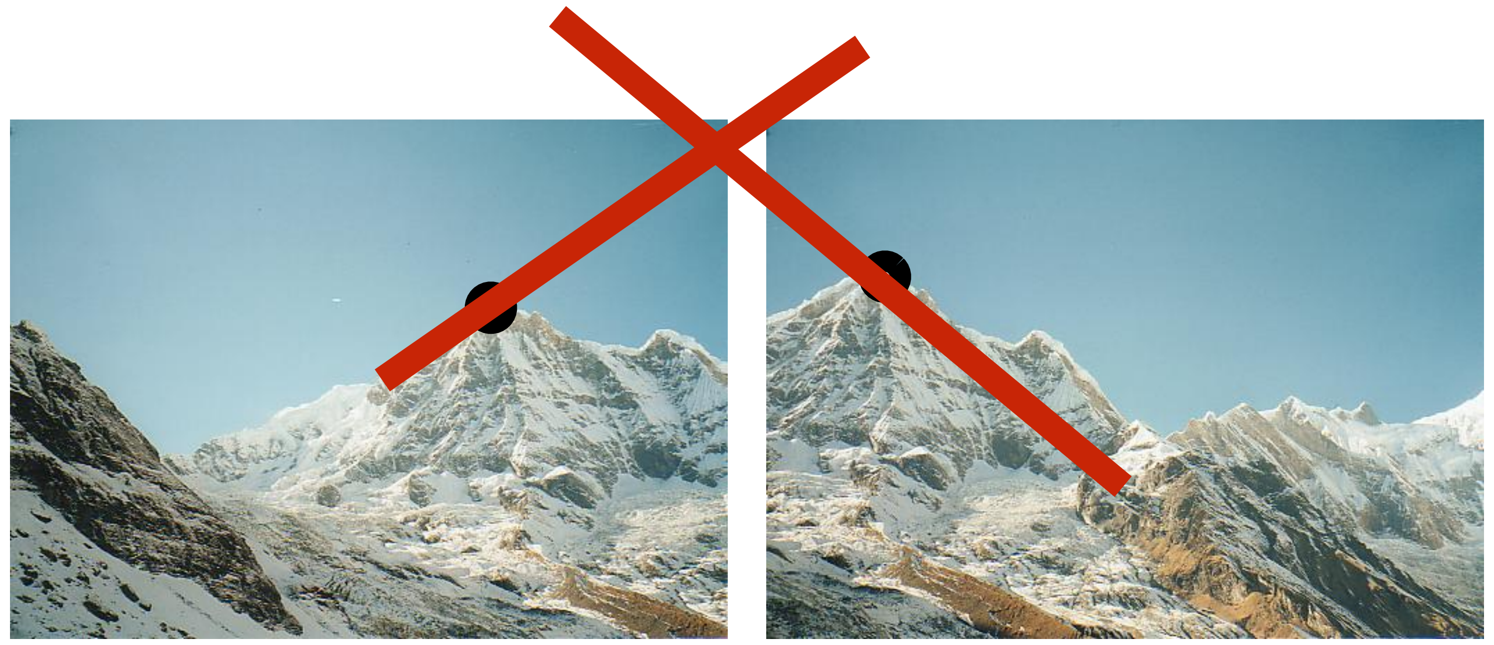
$$\det(C) - \kappa \operatorname{trace}^2(C)$$

Kanade & Tomasi (1994)

$$\min(\lambda_1, \lambda_2)$$

Nobel (1998)

$$\frac{\det(C)}{\operatorname{trace}(C) + \epsilon}$$



# **Difference of Gaussian** blobs in 2020

Harris & Stephens (1988)

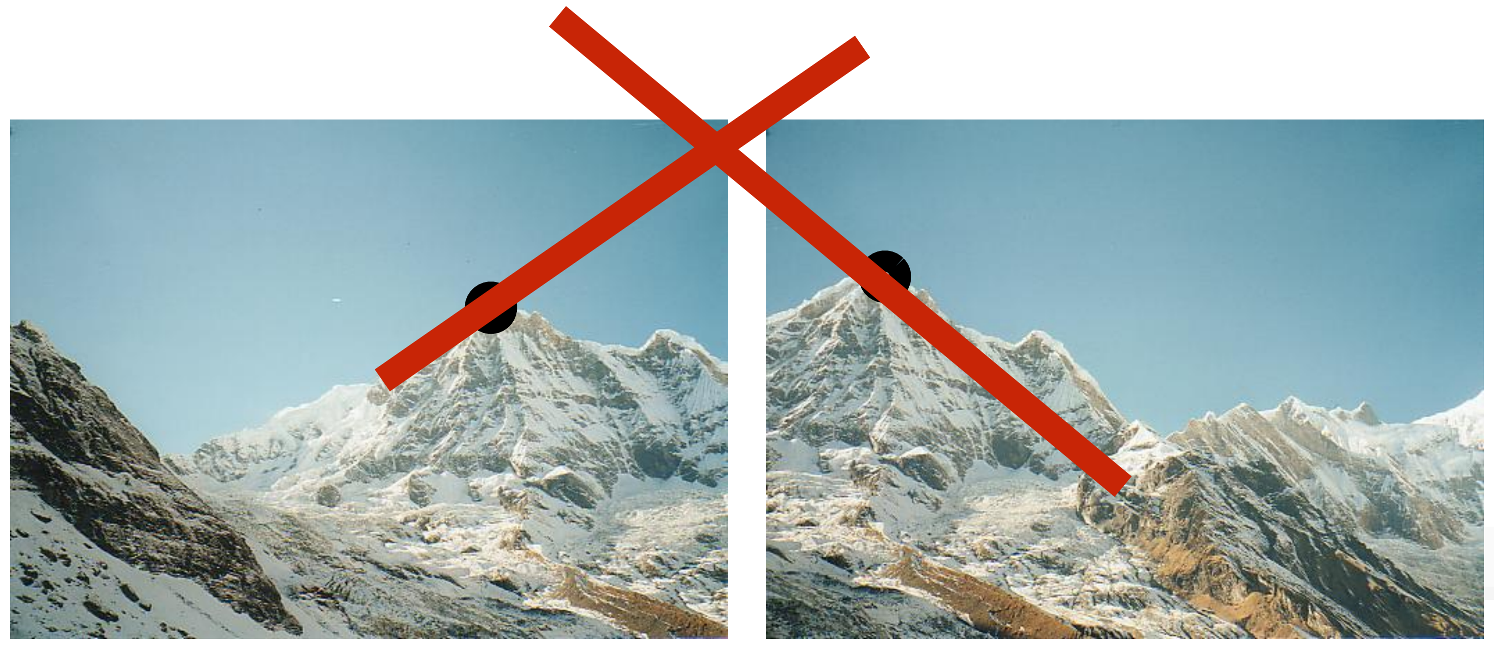
$$\det(C) - \kappa \operatorname{trace}^2(C)$$

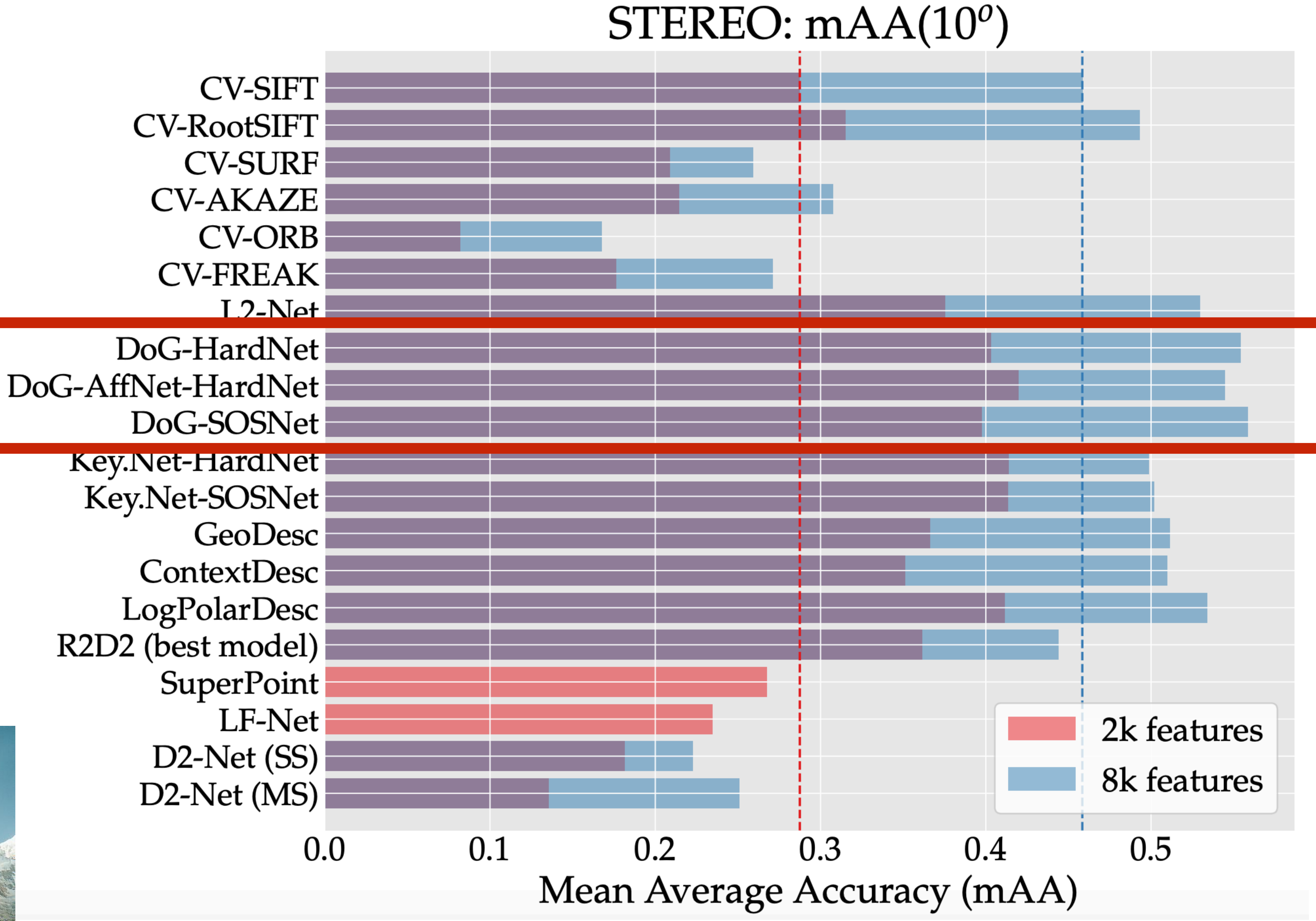
Kanade & Tomasi (1994)

$$\min(\lambda_1, \lambda_2)$$

Nobel (1998)

$$\frac{\det(C)}{\operatorname{trace}(C) + \epsilon}$$



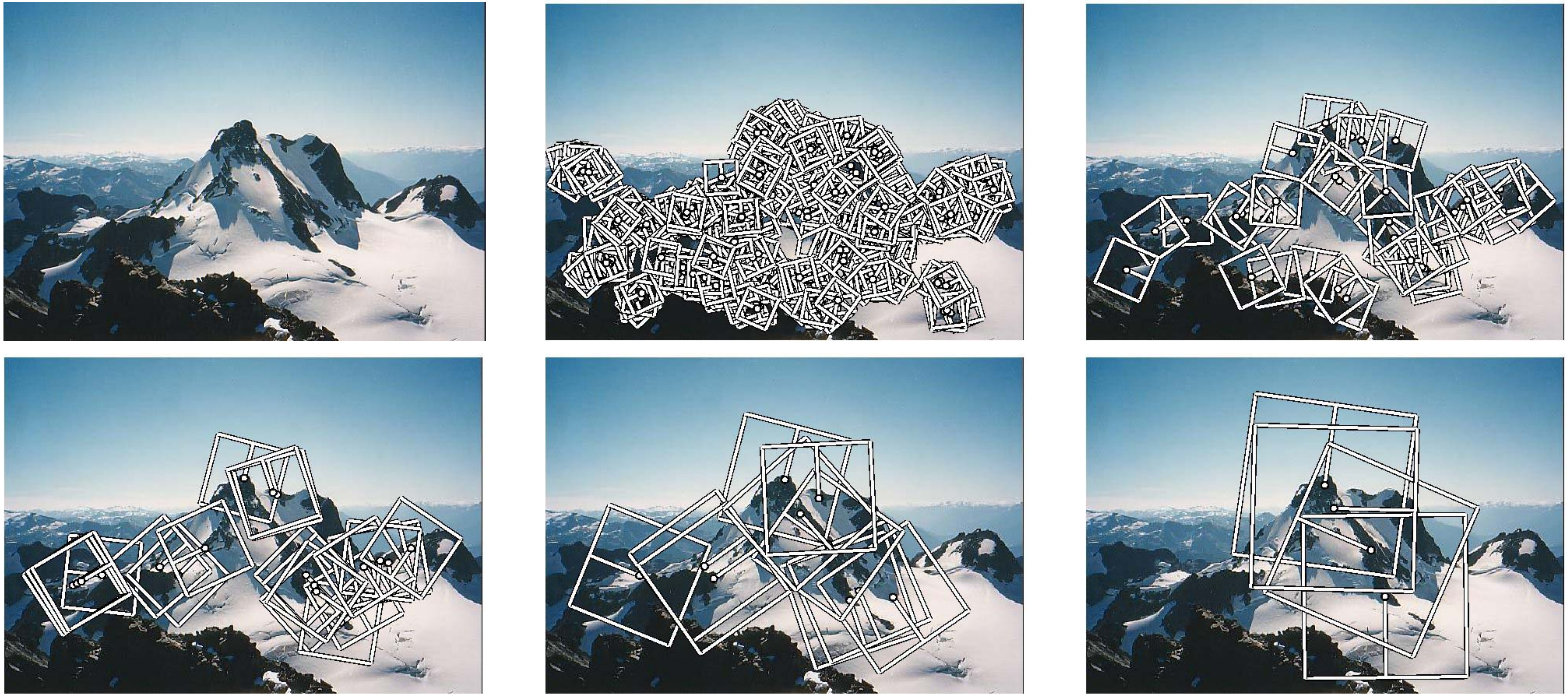


# Implementation

```
For each level of the Gaussian pyramid
compute feature response (e.g. Harris, Laplacian)

For each level of the Gaussian pyramid
    if local maximum and cross-scale
    save scale and location of feature (x, y, s)
```

# **Multi-Scale** Harris Corners



# Re-cap

Summary of what we have seen so far:

| Representation | Results in | Approach | Technique |
|---|---|---|---|
| intensity | dense | template matching | (normalized) correlation |
| edge | relatively sparse | derivatives | Sobel, LoG, Canny |
| corner | sparse | locally distinct features | Harris (and variants) |
| blob | sparse | locally distinct features | LoG |

# Re-cap

Summary of what we have seen so far:

| Representation | Results in | Approach | Technique |
|---|---|---|---|
| intensity | dense | **template matching** | **(normalized) correlation** |
| edge | relatively sparse | derivatives | Sobel, LoG, Canny |
| corner | sparse | locally distinct features | Harris (and variants) |
| blob | sparse | locally distinct features | LoG |

# Re-cap

Summary of what we have seen so far:

| Representation | Results in | Approach | Technique |
|---|---|---|---|
| intensity | dense | template matching | (normalized) correlation |
| edge | relatively sparse | derivatives | Sobel, LoG, Canny |
| **corner** | **sparse** | **locally distinct features** | **Harris (and variants)** |
| **blob** | **sparse** | **locally distinct features** | **LoG** |

# Re-cap

Summary of what we have seen so far:

| Representation | Results in | Approach | Technique |
|---|---|---|---|
| intensity | dense | template matching | (normalized) correlation |
| edge | relatively sparse | derivatives | Sobel, **LoG**, Canny |
| corner | sparse | locally distinct features | Harris (and variants) |
| blob | sparse | locally distinct features | **LoG** |

# Course **Re-cap**

Course Beginning

Course End

# Course **Re-cap**

**Brittle**
(failure in many conditions)

**Robustness**

**Robust**
(works with noise, complex images, clutter)

Course Beginning

Course End

# Course **Re-cap**

Course Beginning

**Brittle**
(failure in many conditions)

Robustness

**Robust**
(works with noise, complex images, clutter)

**Global**
(templates)

**Local**
(edges, corners, blobs, patches)

Image Representations

**Compositional**
(local + flexible global)

Course End

# Course **Re-cap**

Course Beginning

**Brittle**
(failure in many conditions)

**Robustness**

**Robust**
(works with noise, complex images, clutter)

**Global**
(templates)

**Local**
(edges, corners, blobs, patches)

**Image Representations**

**Compositional**
(local + flexible global)

**Hand** defined
(filters, thresholds)

**Statistical**
(means, covariances, histograms)

**Method of Obtaining Image Representations**

**Learned**
(SVMs, Neural Networks)

Course End

# Summary

**Edges** are useful image features for many applications, but suffer from the aperture problem

**Canny** Edge detector combines edge filtering with linking and hysteresis steps

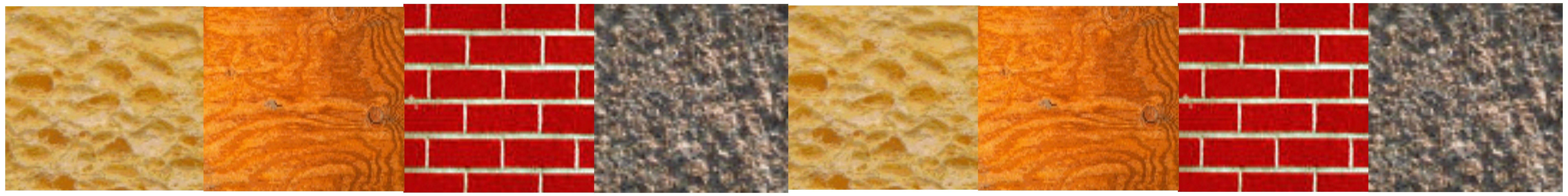
**Corners / Interest Points** have 2D structure and are useful for correspondence

**Harris** corners are minima of a local SSD function

**DoG** maxima can be reliably located in scale-space and are useful as interest points

THE UNIVERSITY OF BRITISH COLUMBIA

# CPSC 425: Computer Vision



**Lecture 11:** Texture

( unless otherwise stated slides are taken or adopted from **Bob Woodham, Jim Little** and **Fred Tung** )

# Texture

What is **texture**?

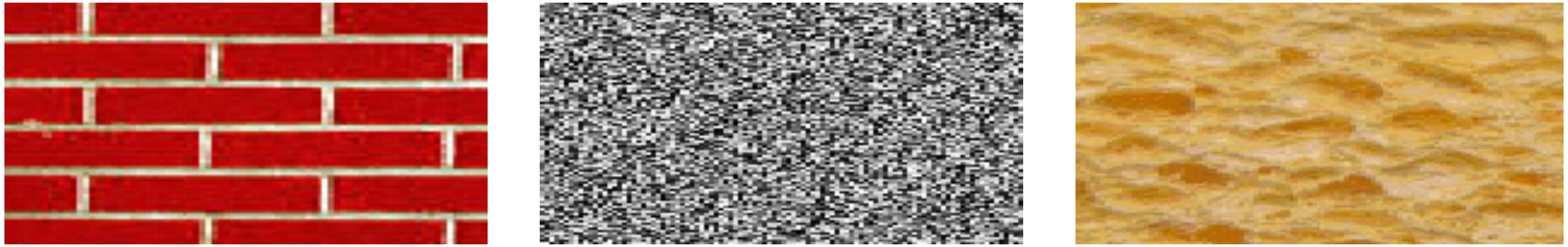


**Figure Credit**: Alexei Efros and Thomas Leung

Texture is widespread, easy to recognize, but hard to define

Views of large numbers of small objects are often considered textures

— e.g. grass, foliage, pebbles, hair

Patterned surface markings are considered textures

— e.g. patterns on wood

# Definition of **Texture**

(Functional) **Definition**:

**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

# Definition of **Texture**

(Functional) **Definition**:

**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

Sometimes, textures are 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

# Uses of **Texture**

Texture can be a strong cue to **object identity** if the object has 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**"

# Texture

We will look at two main questions:

1. How do we represent texture?
   → Texture **analysis**

2. How do we generate new examples of a texture?
   → Texture **synthesis**

We begin with texture synthesis to set up **Assignment 3**

# Texture **Synthesis**

# Texture **Synthesis**

Why might we want to synthesize texture?

# Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

# Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

— Art directors might want to remove telephone wires. Restorers might want to remove scratches or marks.

# Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

— Art directors might want to remove telephone wires. Restorers might want to remove scratches or marks.

— We need to find something to put in place of the pixels that were removed

# Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

— Art directors might want to remove telephone wires. Restorers might want to remove scratches or marks.

— We need to find something to put in place of the pixels that were removed

— We synthesize regions of texture that fit in and look convincing

# Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

— Art directors might want to remove telephone wires. Restorers might want to remove scratches or marks.

— We need to find something to put in place of the pixels that were removed

— We synthesize regions of texture that fit in and look convincing

2. To produce large quantities of texture for computer graphics

# Texture **Synthesis**

Why might we want to synthesize texture?

1. To fill holes in images (**inpainting**)

— Art directors might want to remove telephone wires. Restorers might want to remove scratches or marks.

— We need to find something to put in place of the pixels that were removed

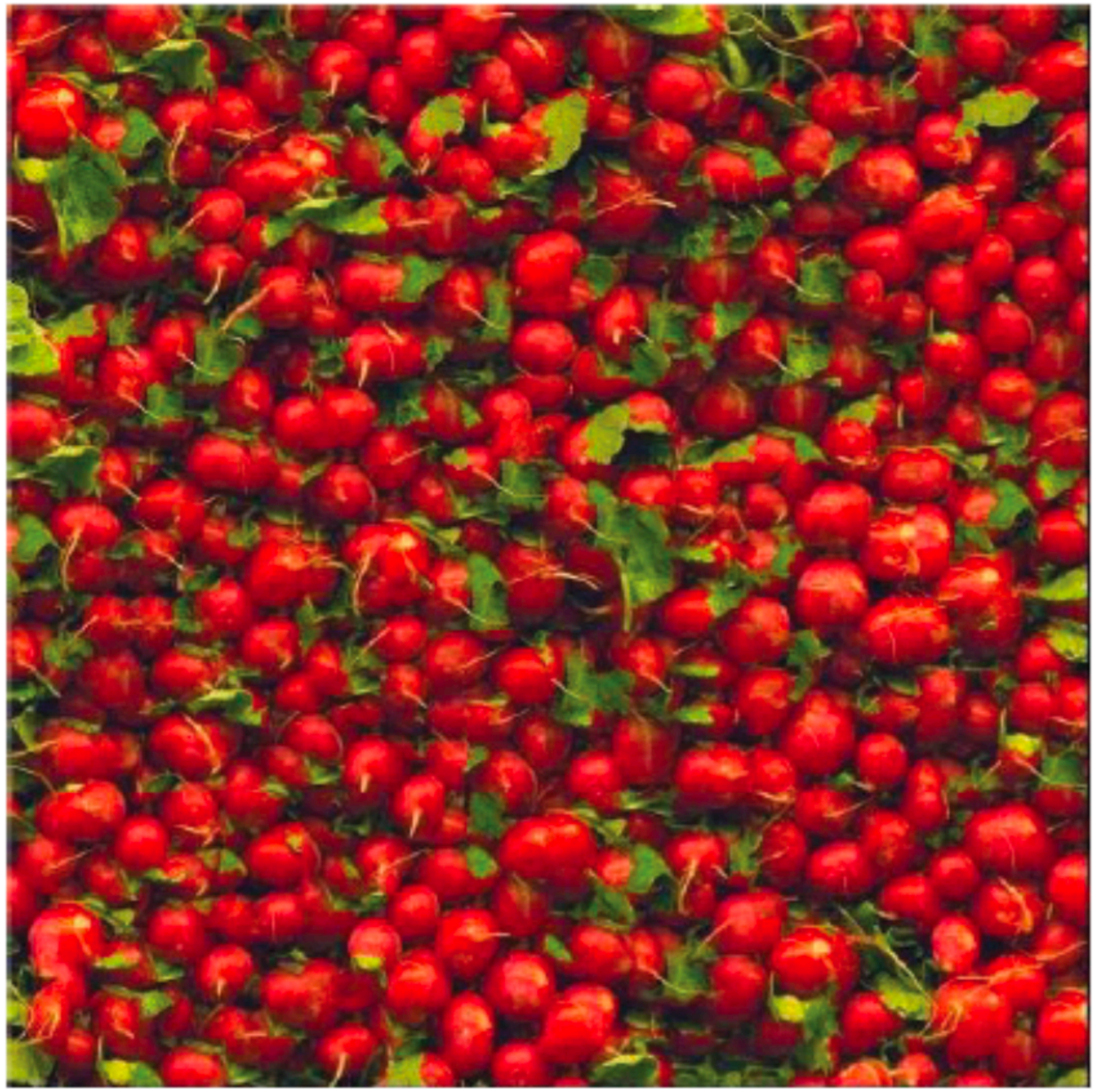
— We synthesize regions of texture that fit in and look convincing

2. To produce large quantities of texture for computer graphics

— Good textures make object models look more realistic

# Texture **Synthesis**

radishes



lots more radishes

Szeliski, Fig. 10.49

# Texture **Synthesis**

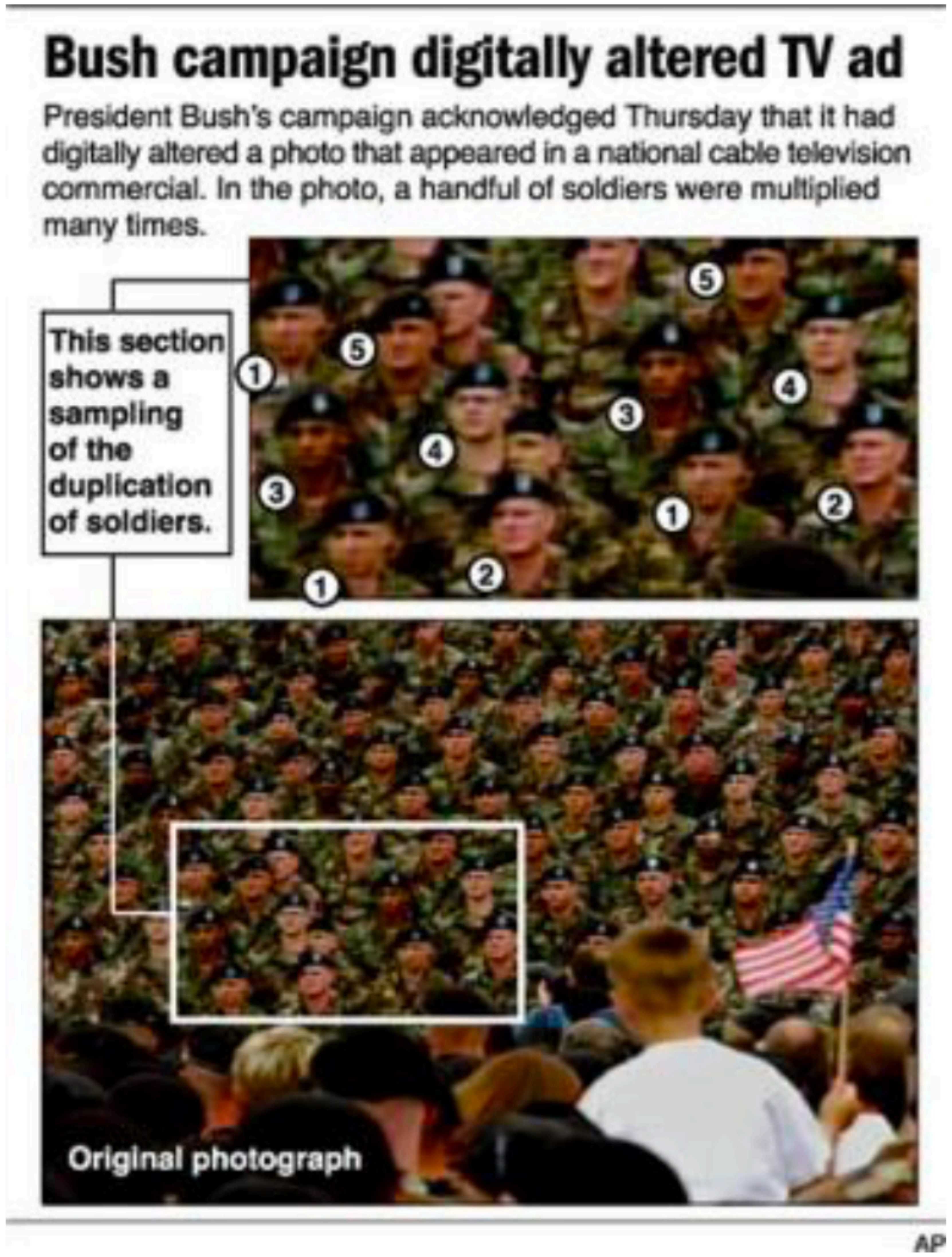



**Photo Credit**: Associated Pres

# Texture **Synthesis**

Cover of "The Economist," June 19, 2010


**Photo Credit** (right): Reuters/Larry Downing

# **Assignment** 3 Preview: Texture Synthesis

**Task**: Make donkey vanish

# **Assignment** 3 Preview: Texture Synthesis

**Task**: Make donkey vanish

**Method**: Fill-in regions using texture from the white box

# **Assignment** 3 Preview: Texture Synthesis

**Task**: Make donkey vanish

**Method**: Fill-in regions using texture from the white box

# **Texture** Synthesis

**Objective**: Generate new examples of a texture. We take a “data-driven" approach

**Idea**: Use an image of the texture as the source of a probability model

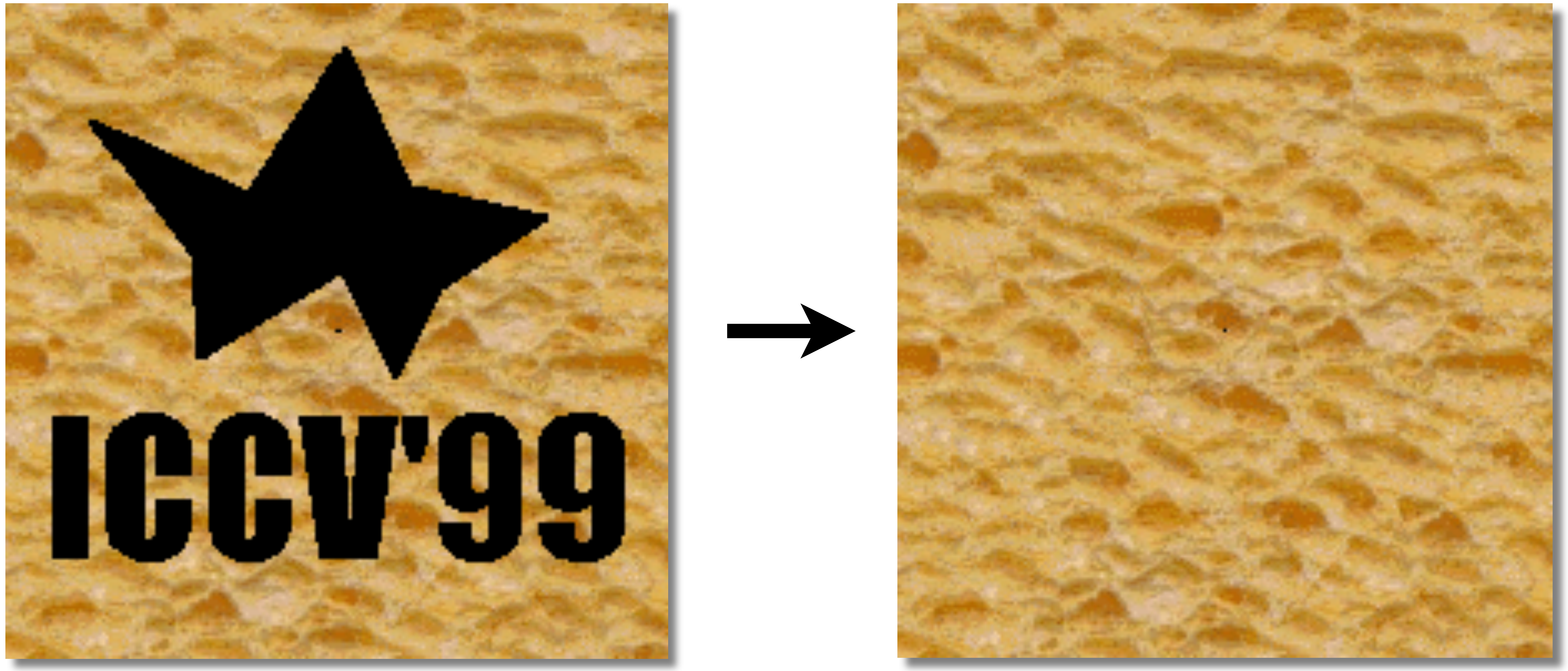
— Draw samples directly from the actual texture

— Can account for more types of structure

— Very simple to implement

— Success depends on choosing a correct “distance”

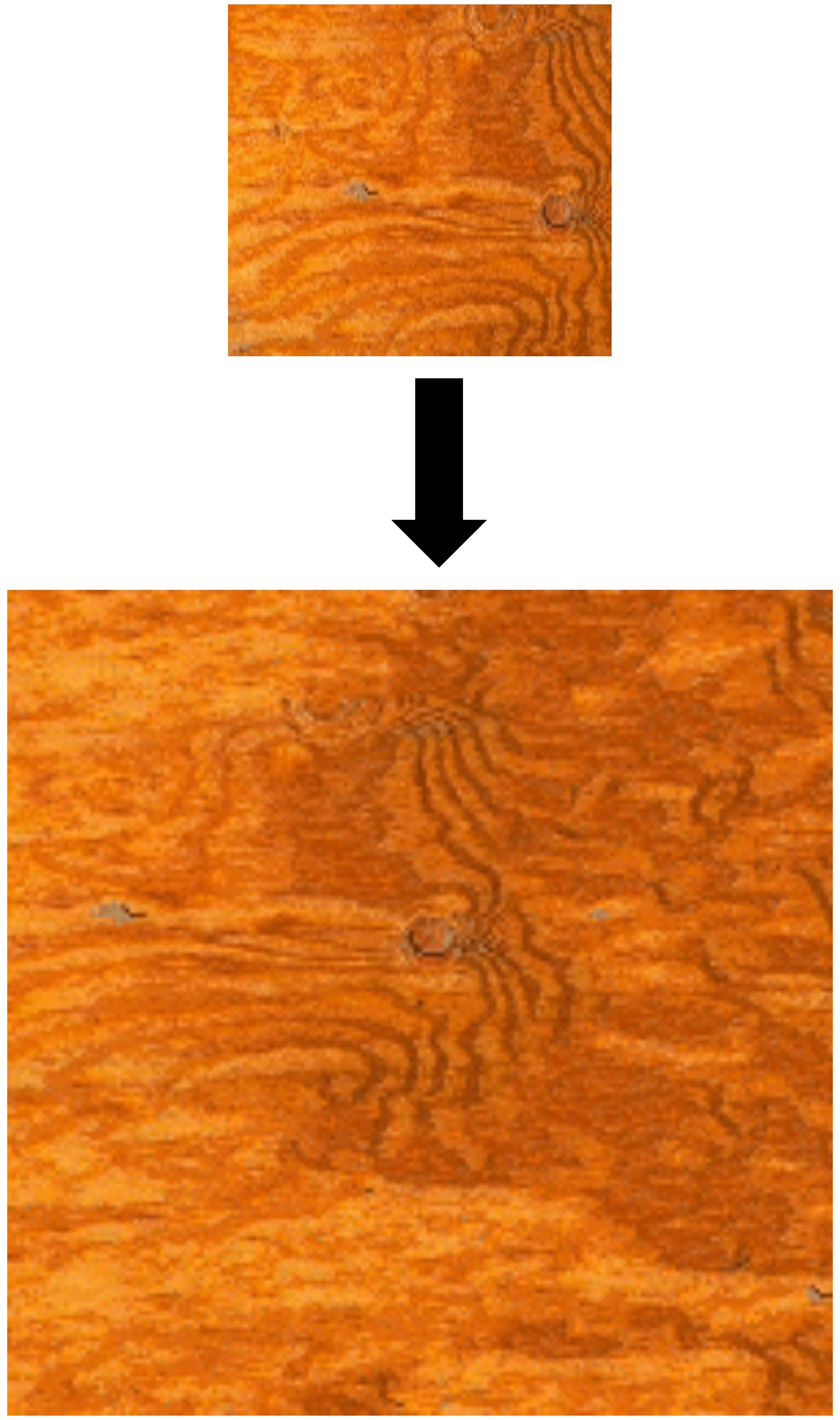
# **Texture** Synthesis by Non-parametric Sampling



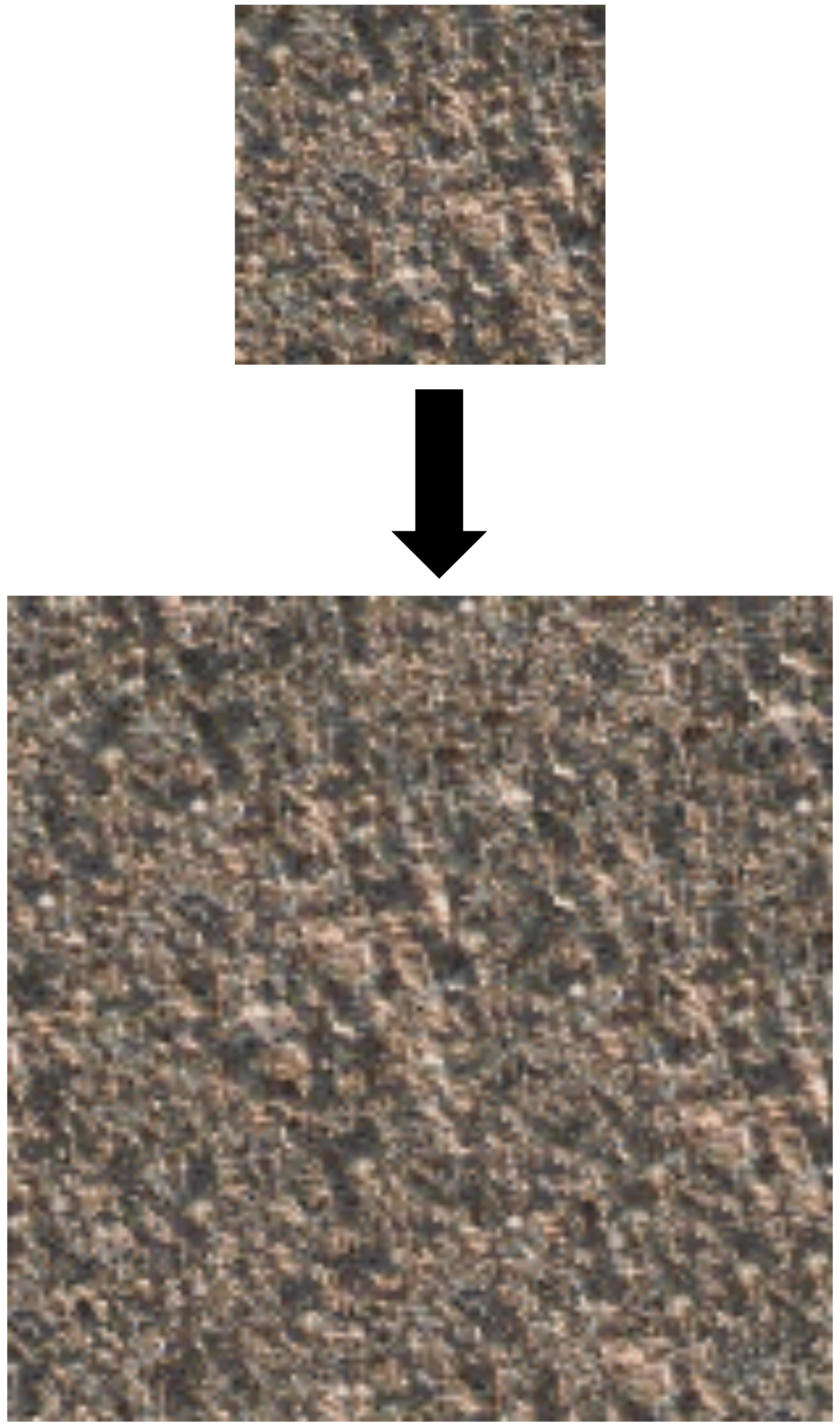

Alexei Efros and Thomas Leung
UC Berkeley

**Slide Credit:** http://graphics.cs.cmu.edu/people/efros/research/NPS/efros-iccv99.ppt

# **Efros** and Leung

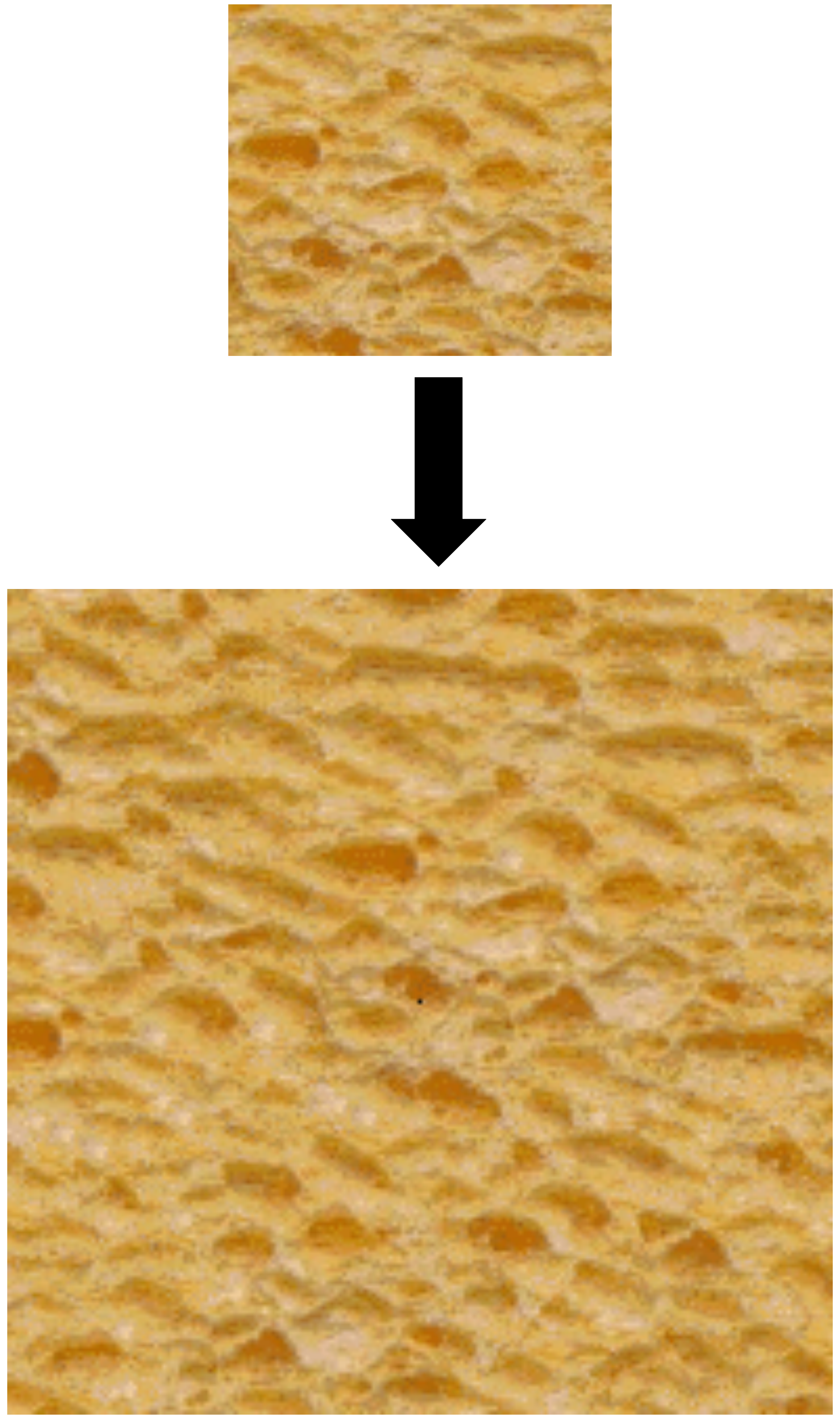


wood

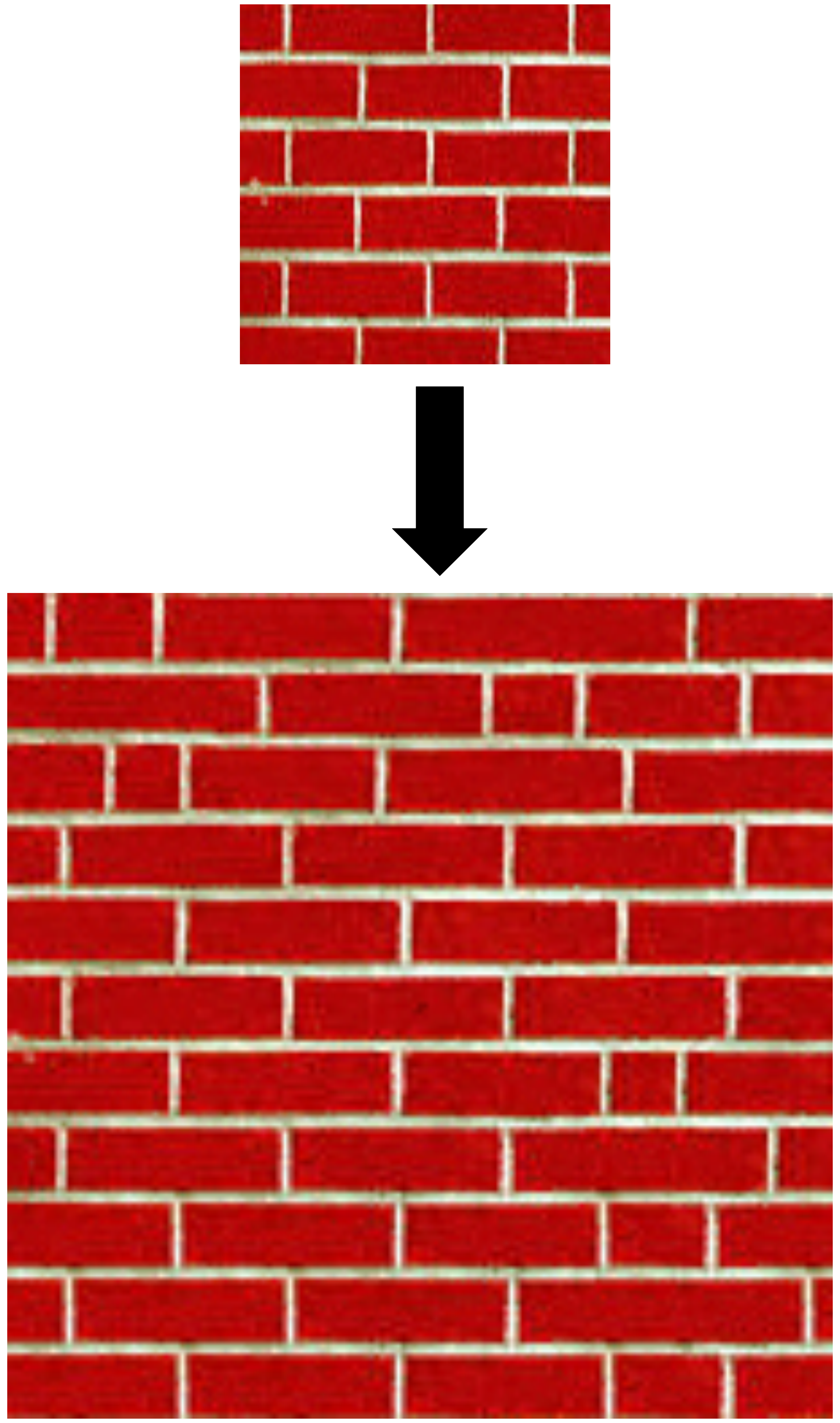


granite

# **Efros** and Leung

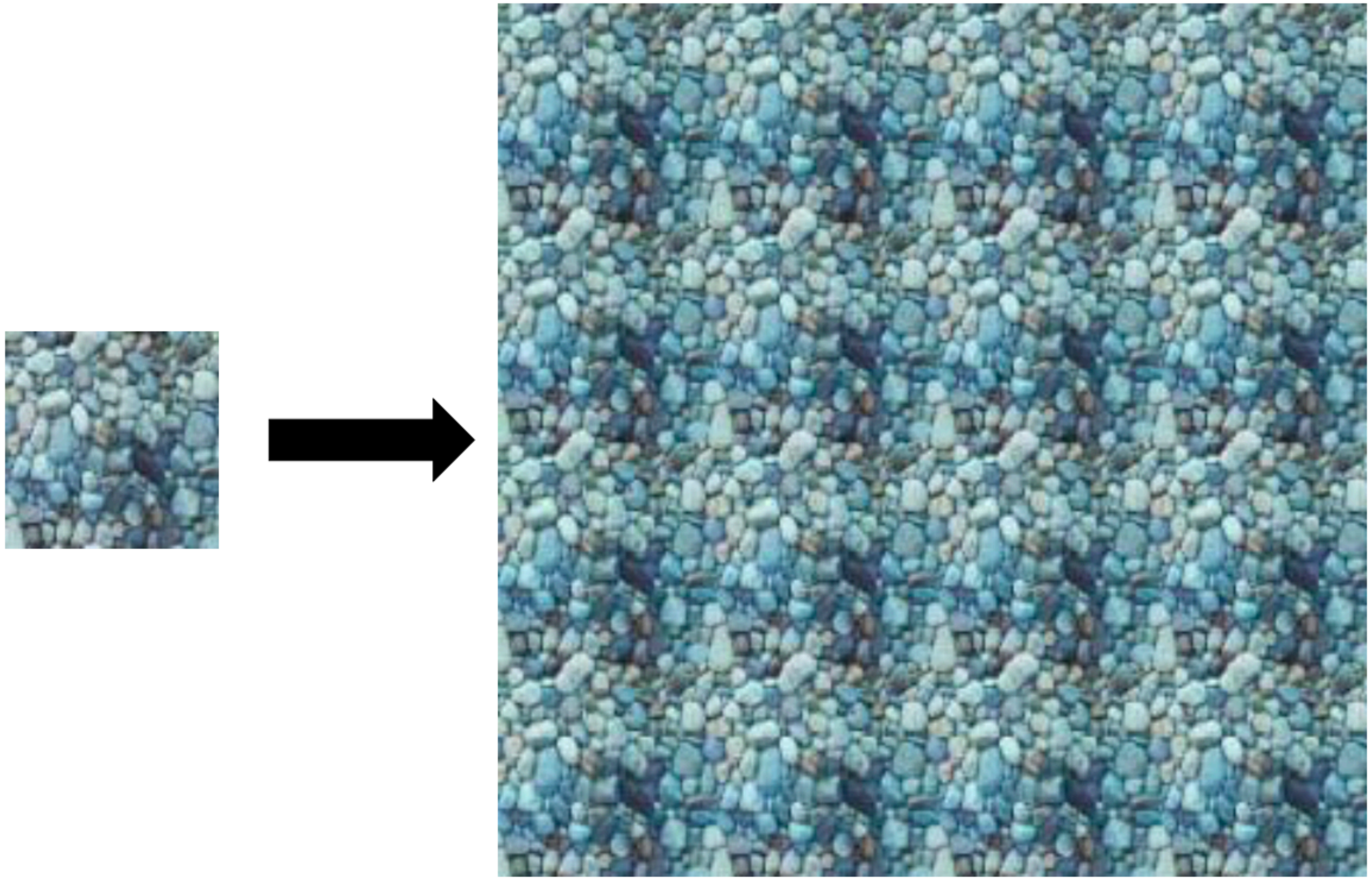


white bread

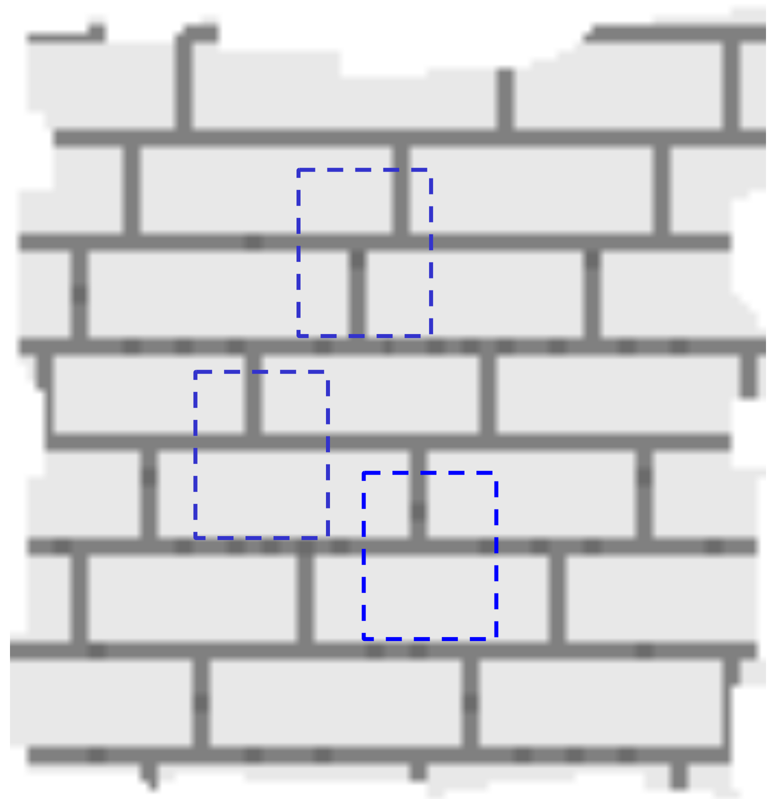


brick wall

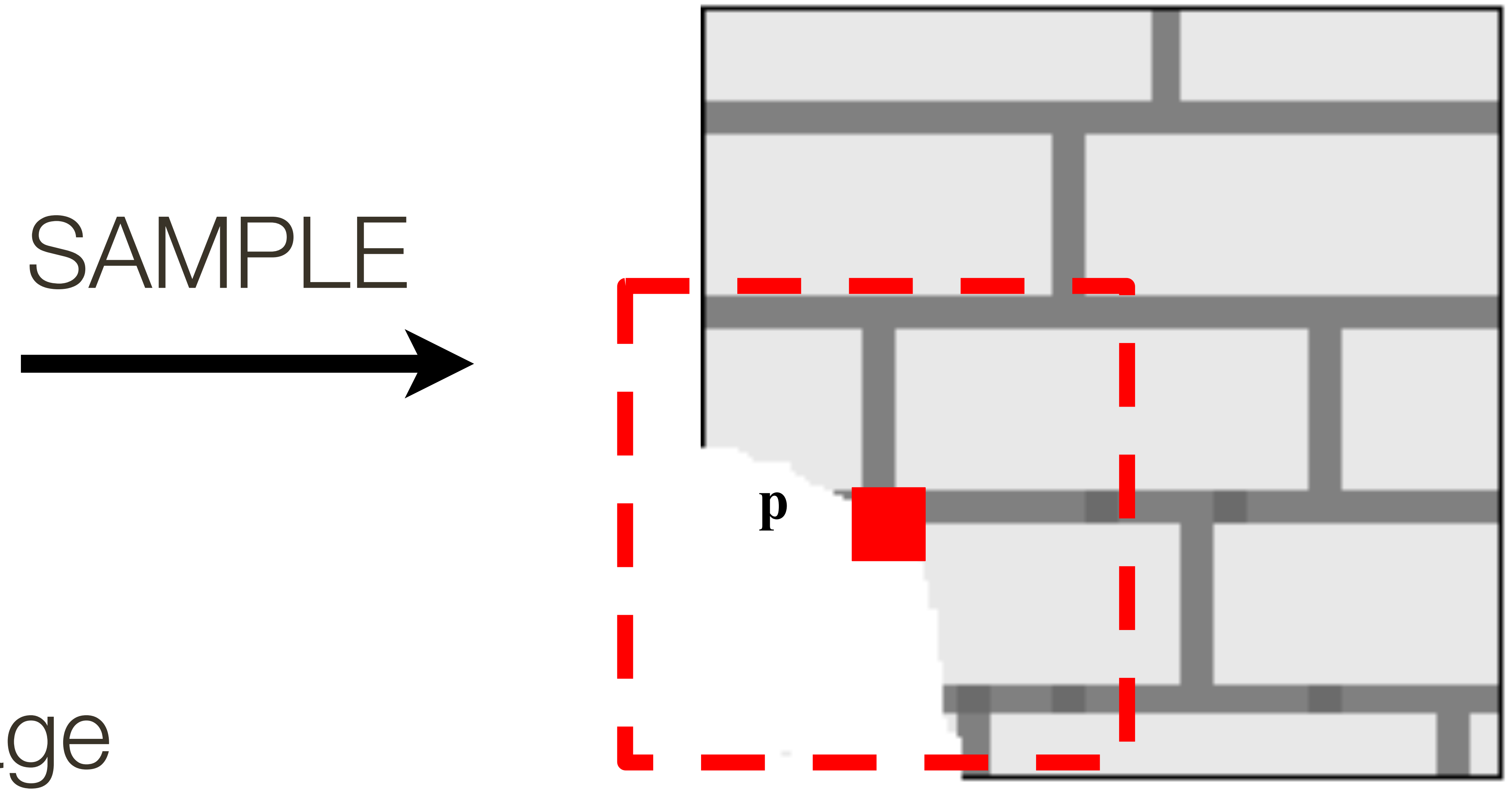
# Like **Copying**, But not Just Repetition



# **Efros** and Leung: Synthesizing One Pixel

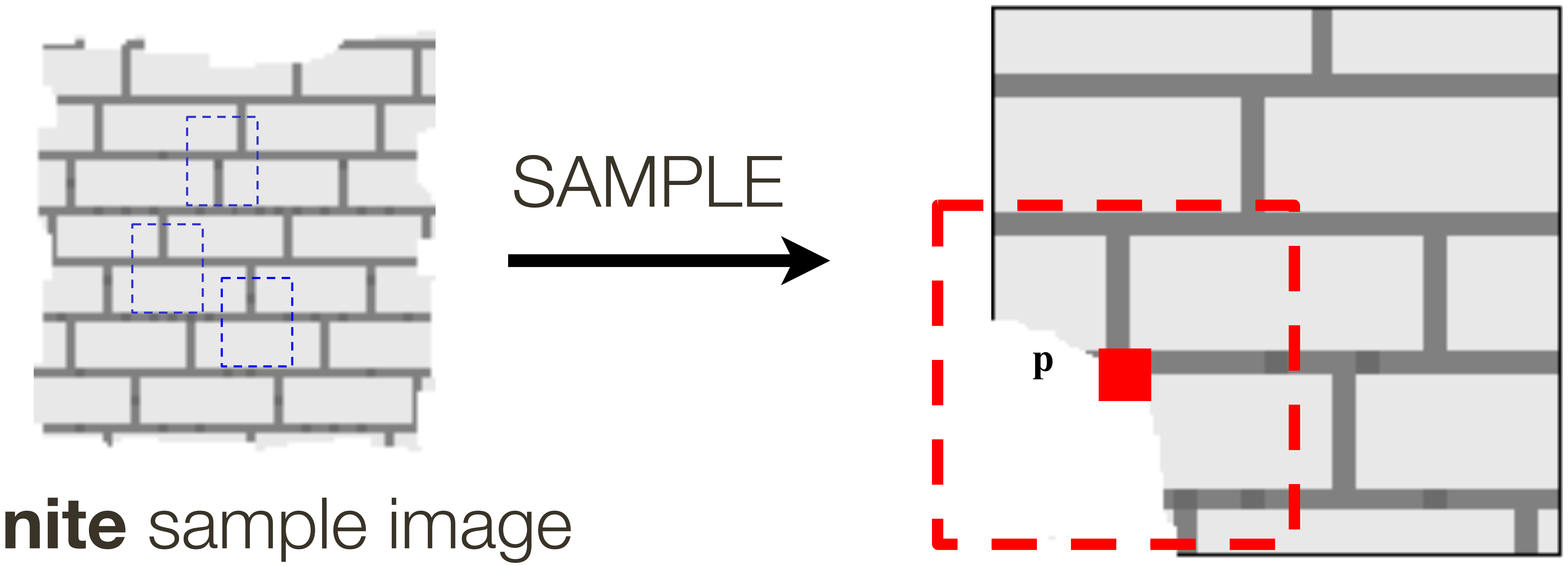


**Infinite** sample image




— What is **conditional** probability distribution of *p*, given the neighbourhood window?

# **Efros** and Leung: Synthesizing One Pixel

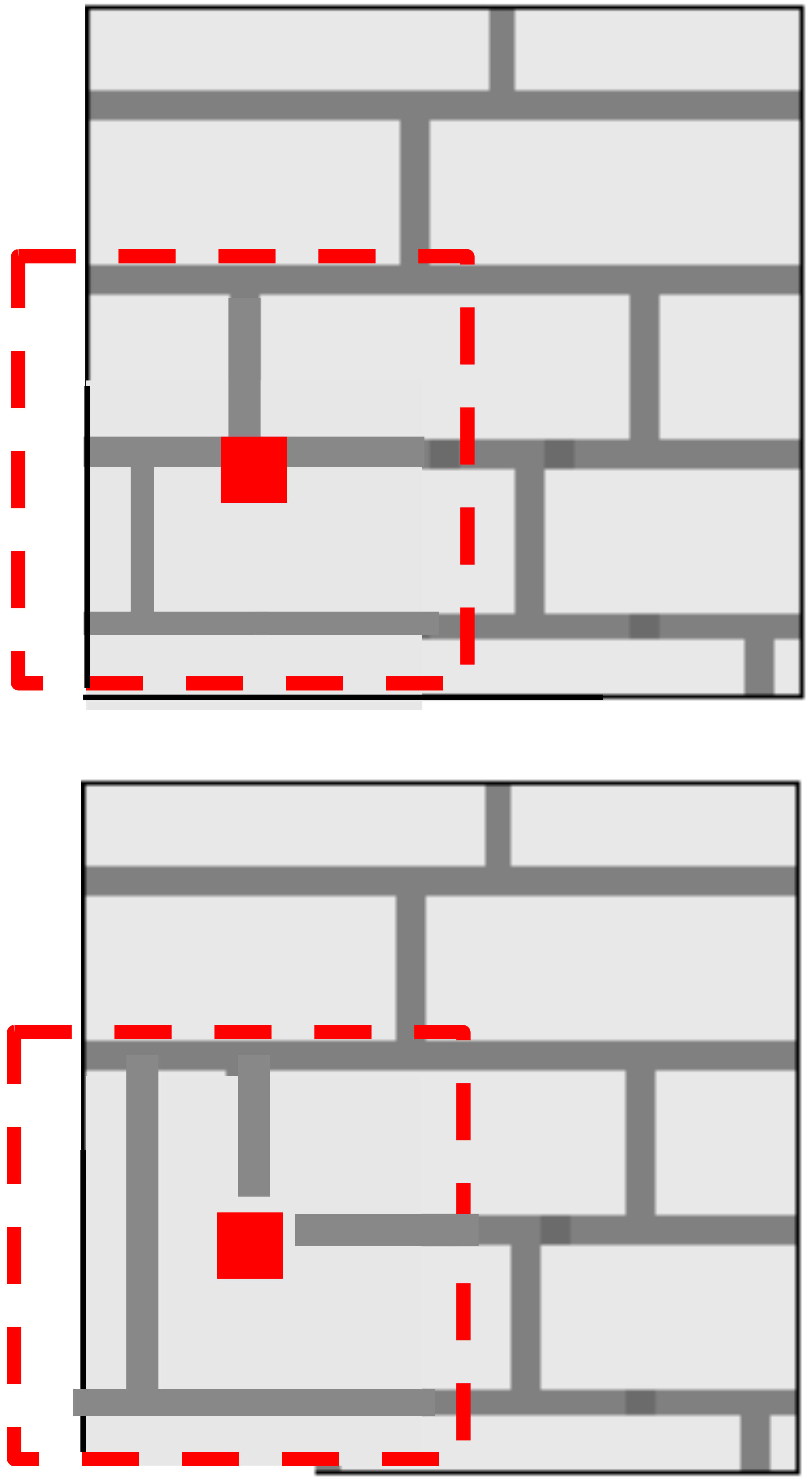


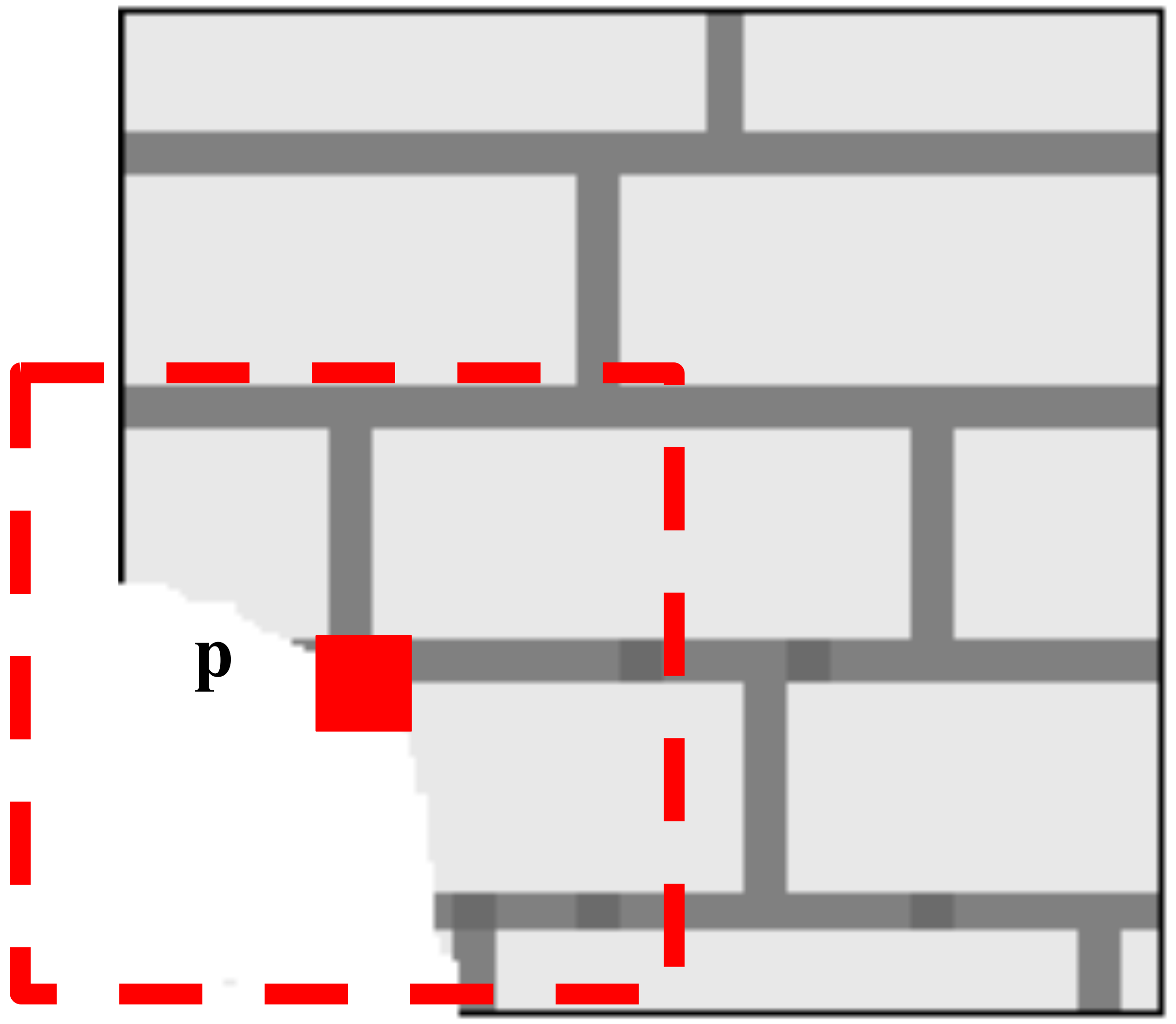

**Infinite** sample image

— What is **conditional** probability distribution of *p*, given the neighbourhood window?

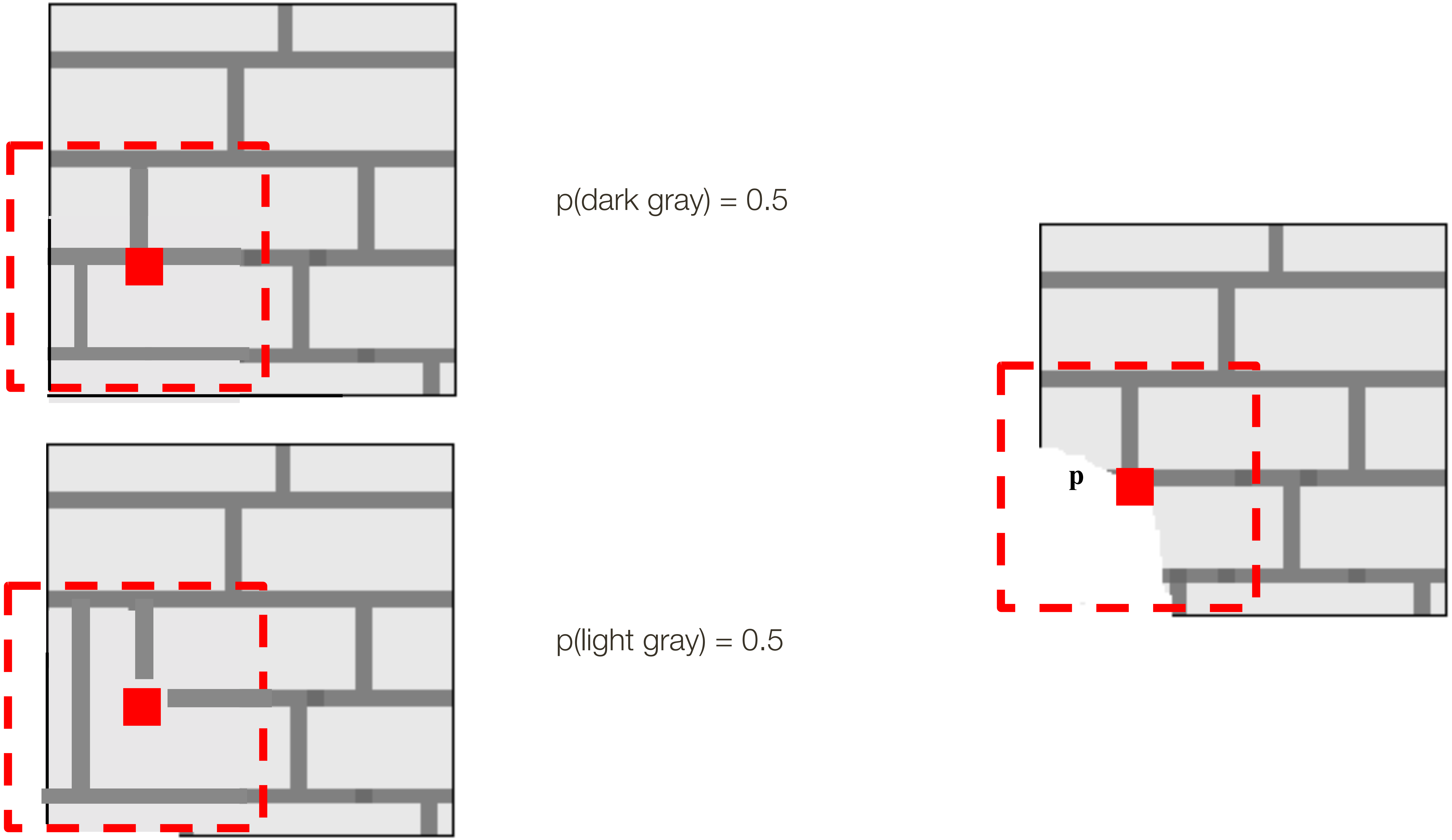
— Directly search the input image for all such neighbourhoods to produce a **histogram** for *p*

# **Efros** and Leung: Synthesizing One Pixel

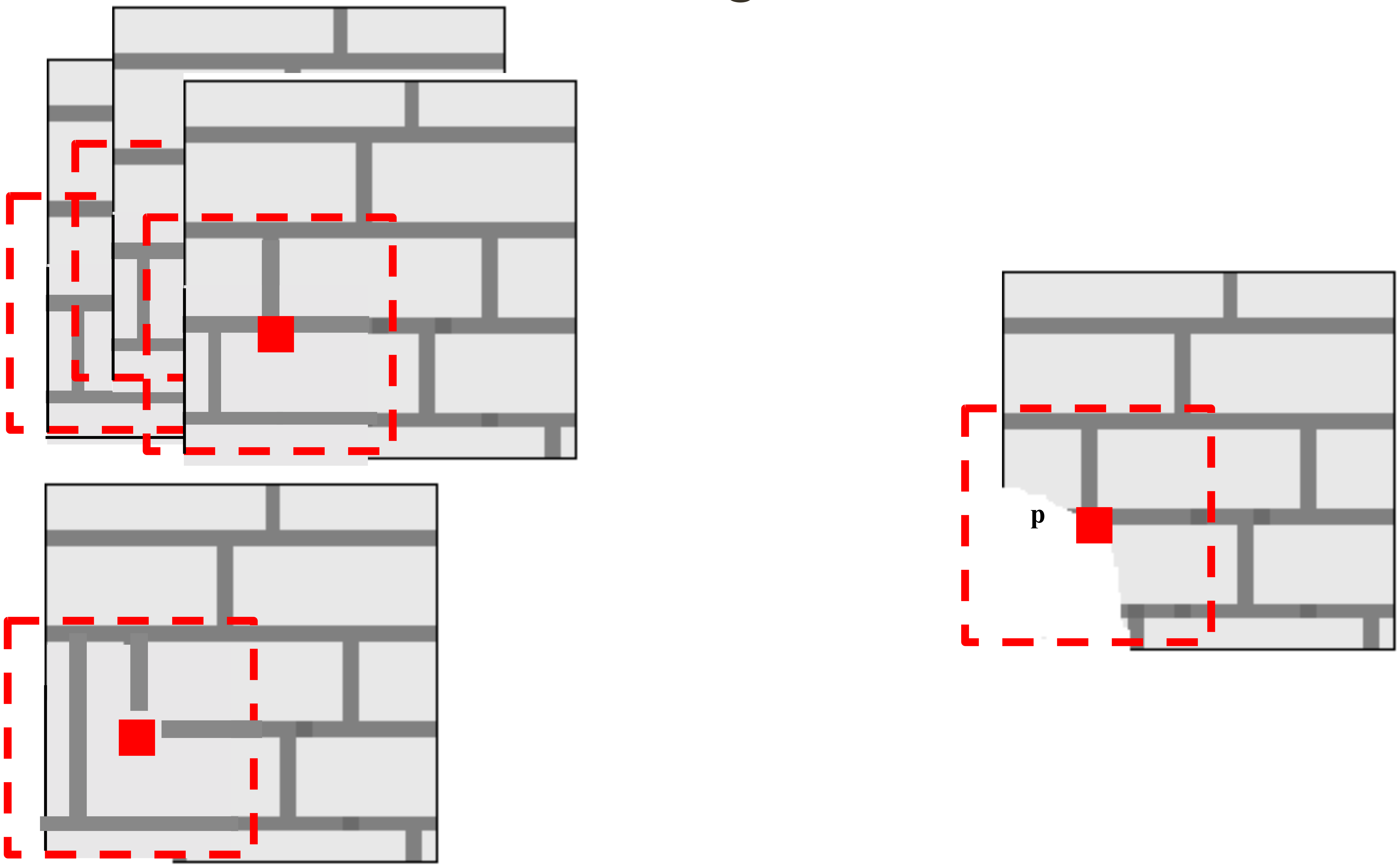




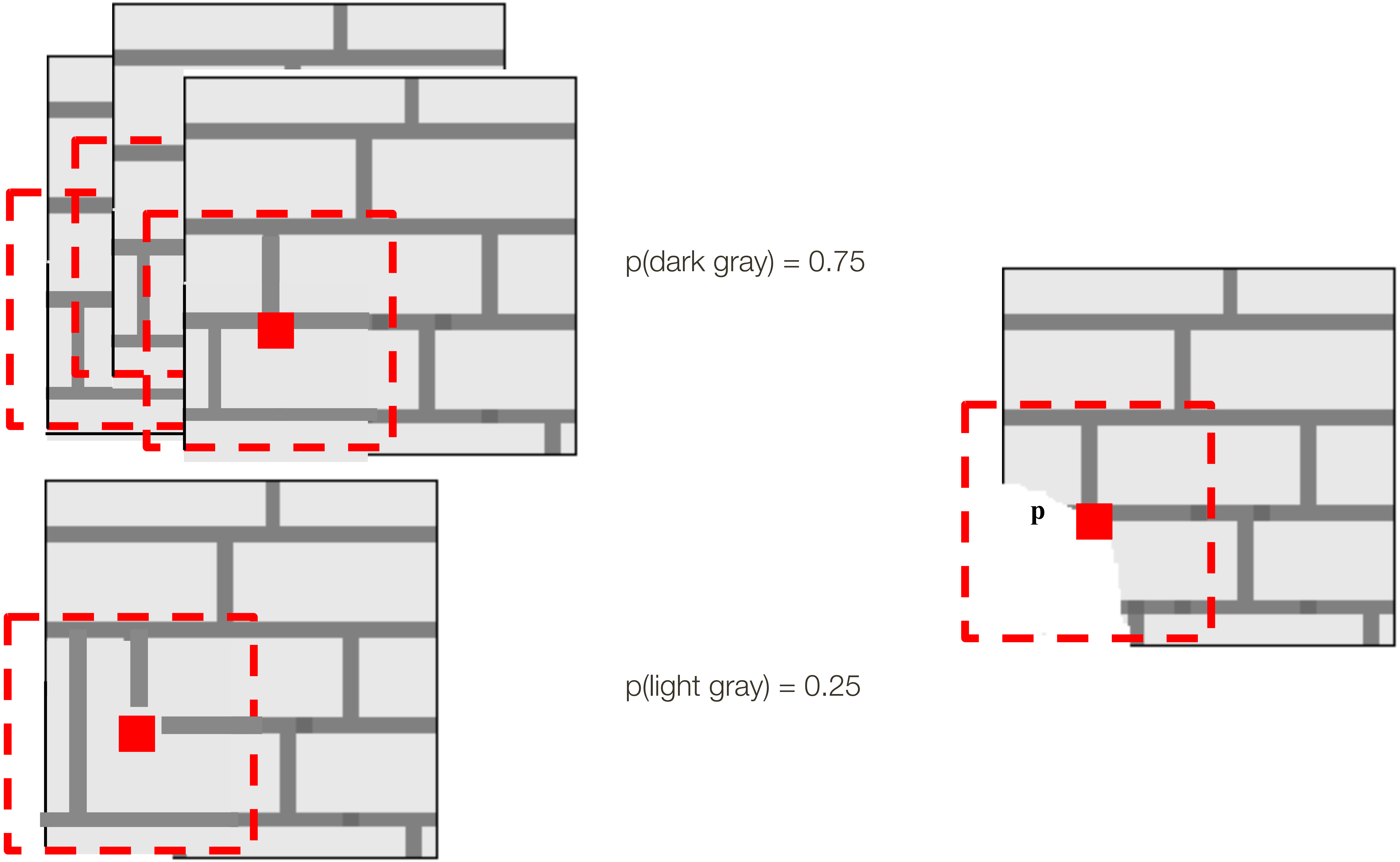
# **Efros** and Leung: Synthesizing One Pixel



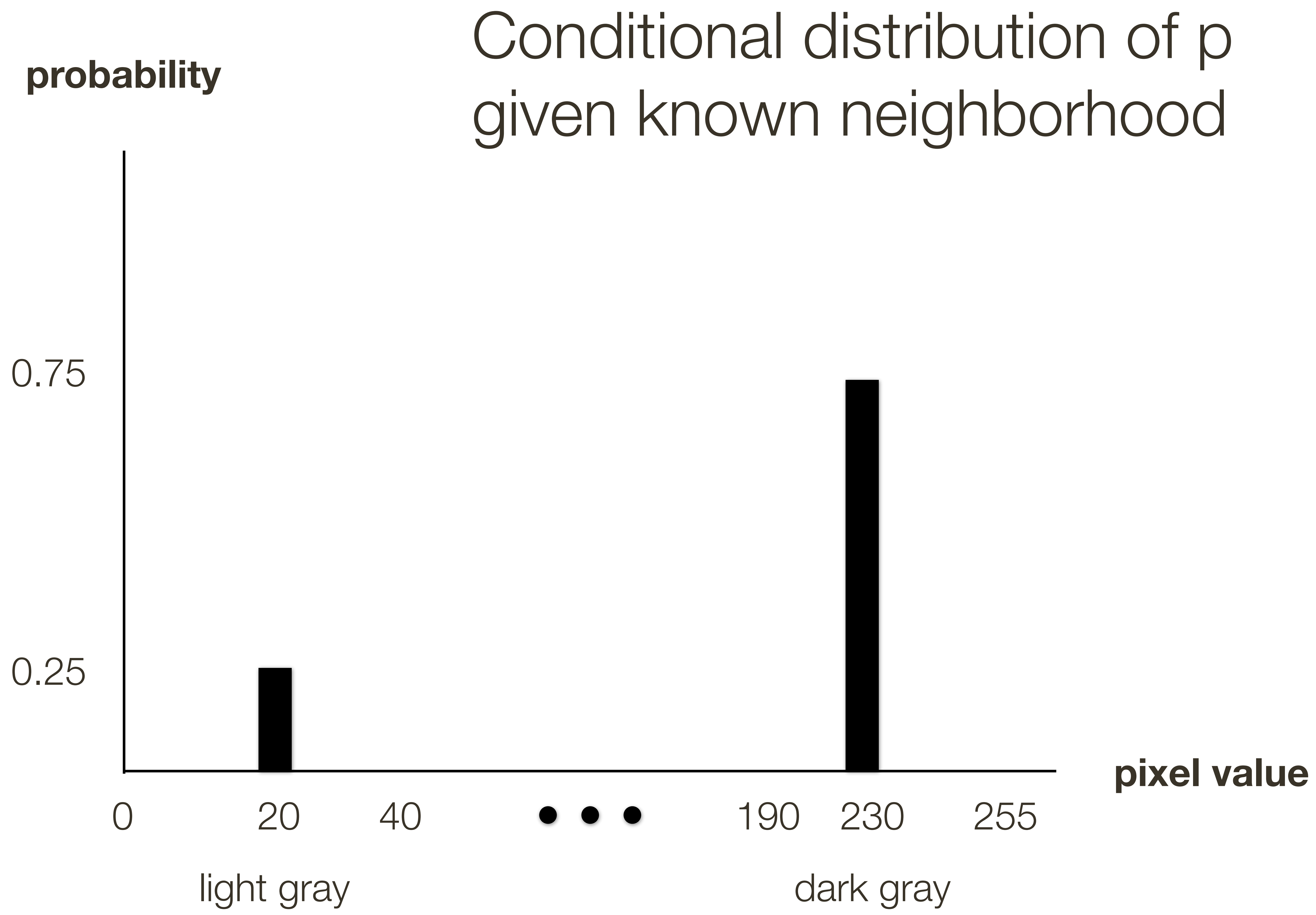
# **Efros** and Leung: Synthesizing One Pixel

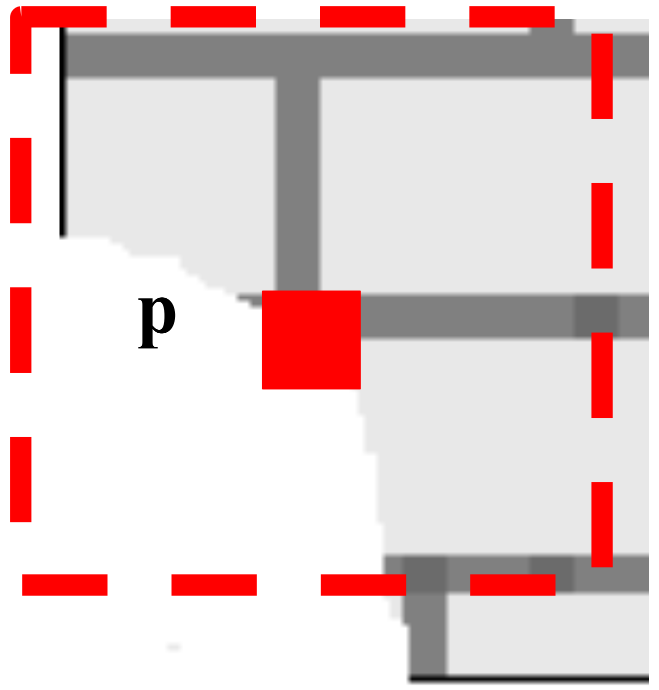


# **Efros** and Leung: Synthesizing One Pixel

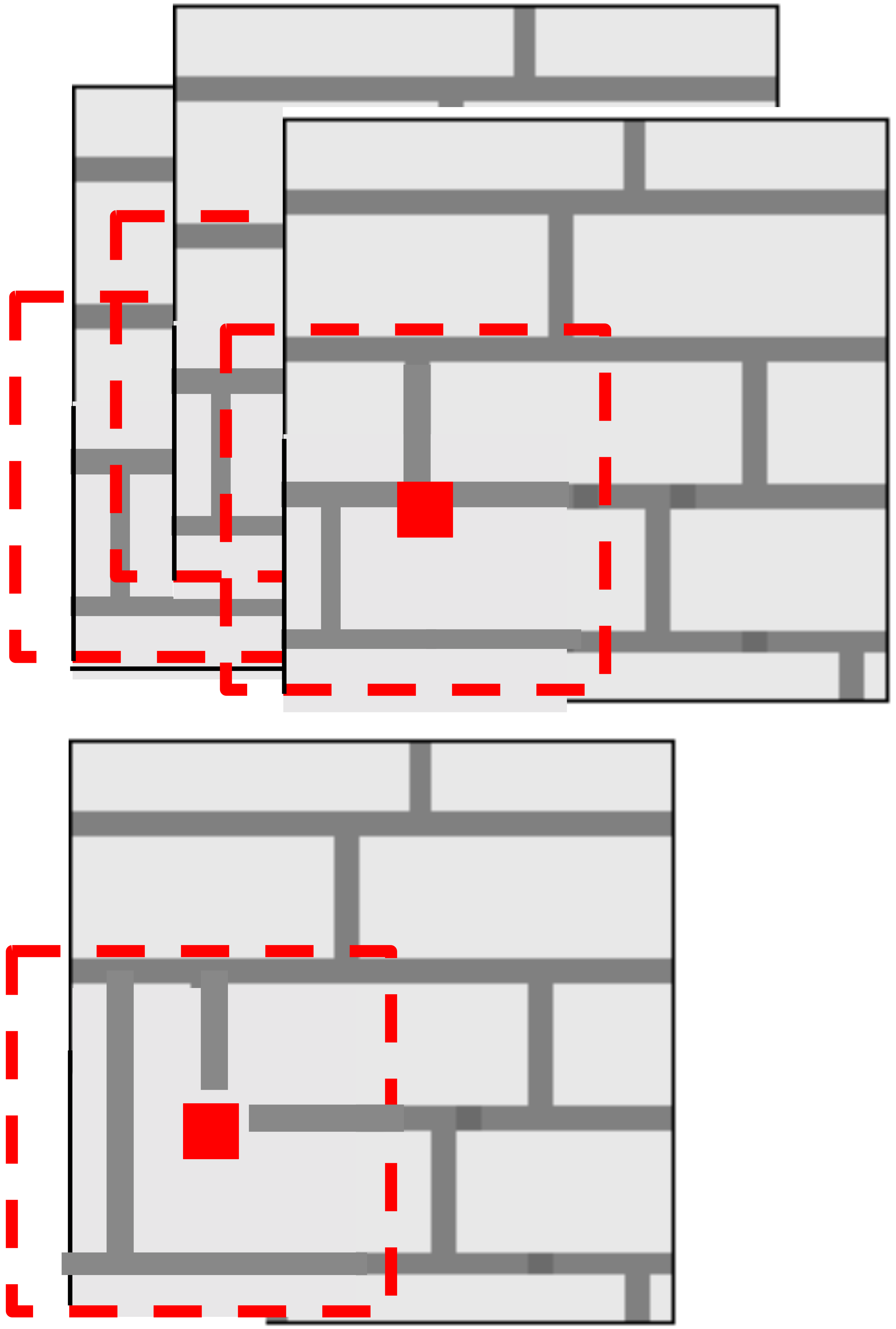


# **Efros** and Leung: Synthesizing One Pixel

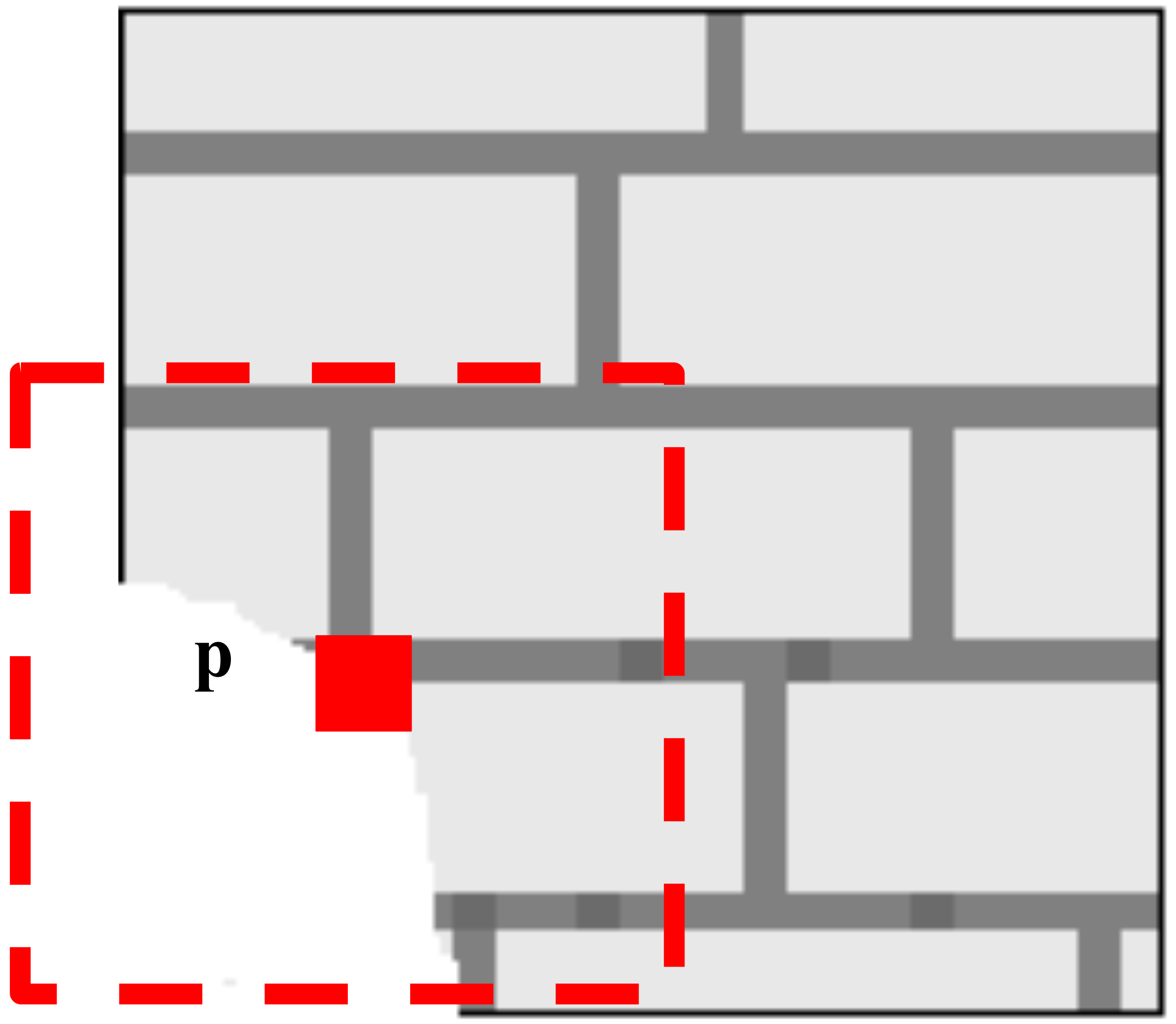


# **Efros** and Leung: Synthesizing One Pixel

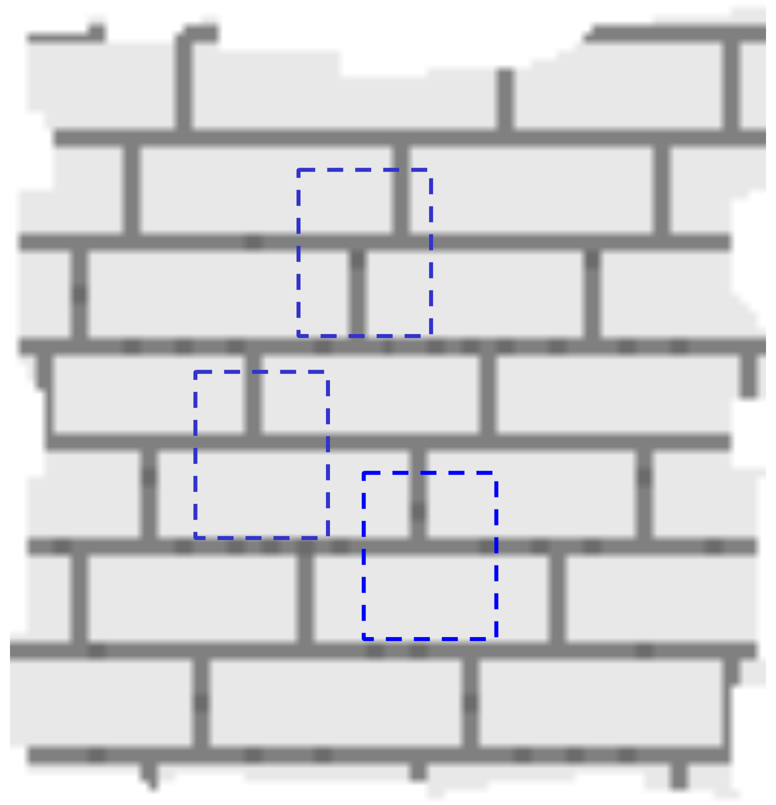


p(dark gray) = 0.75

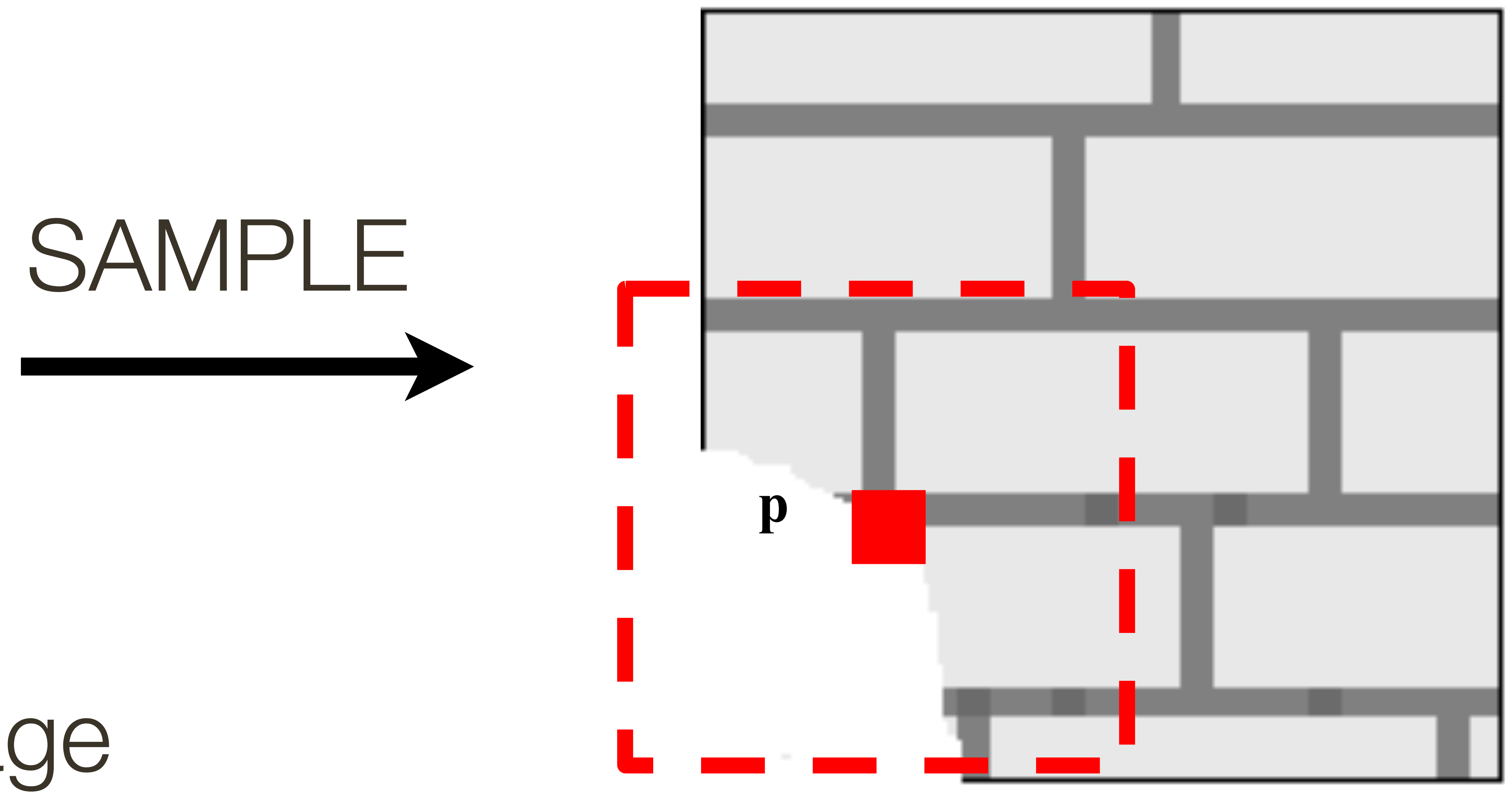



p(light gray) = 0.25

# **Efros** and Leung: Synthesizing One Pixel



**Infinite** sample image

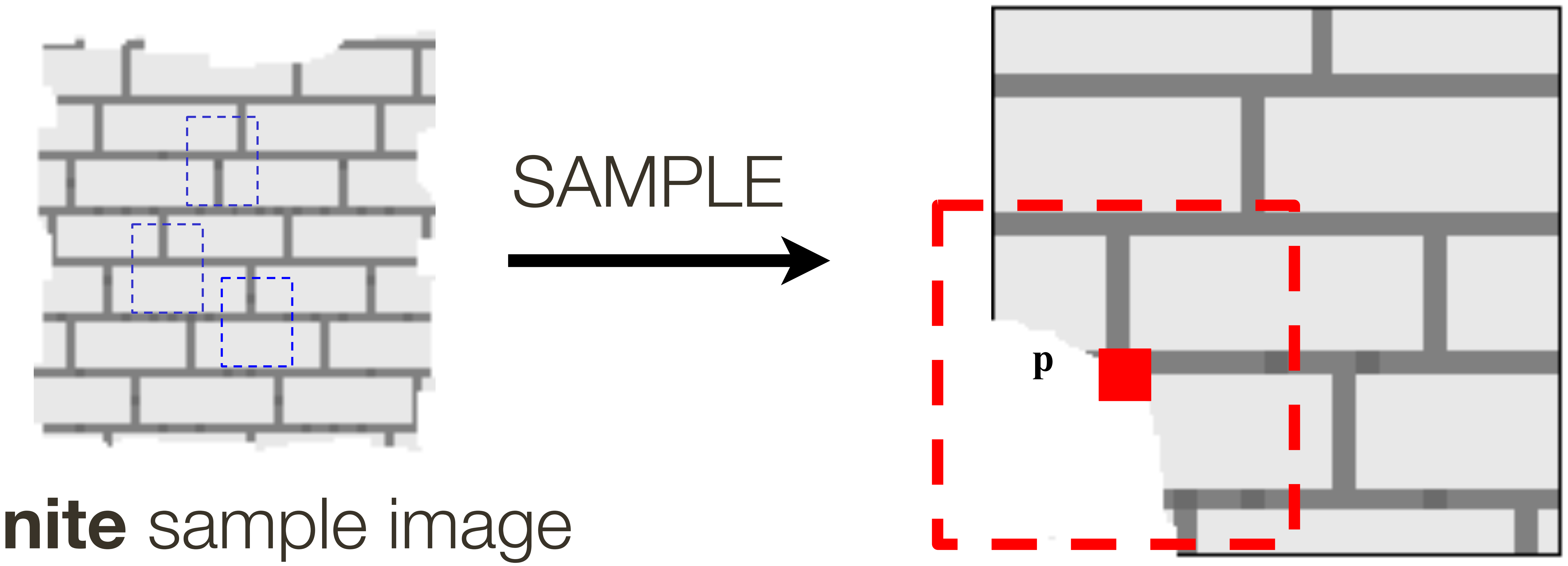



— What is **conditional** probability distribution of *p*, given the neighbourhood window?

— Directly search the input image for all such neighbourhoods to produce a **histogram** for *p*

— To **synthesize** *p*, pick one match at random

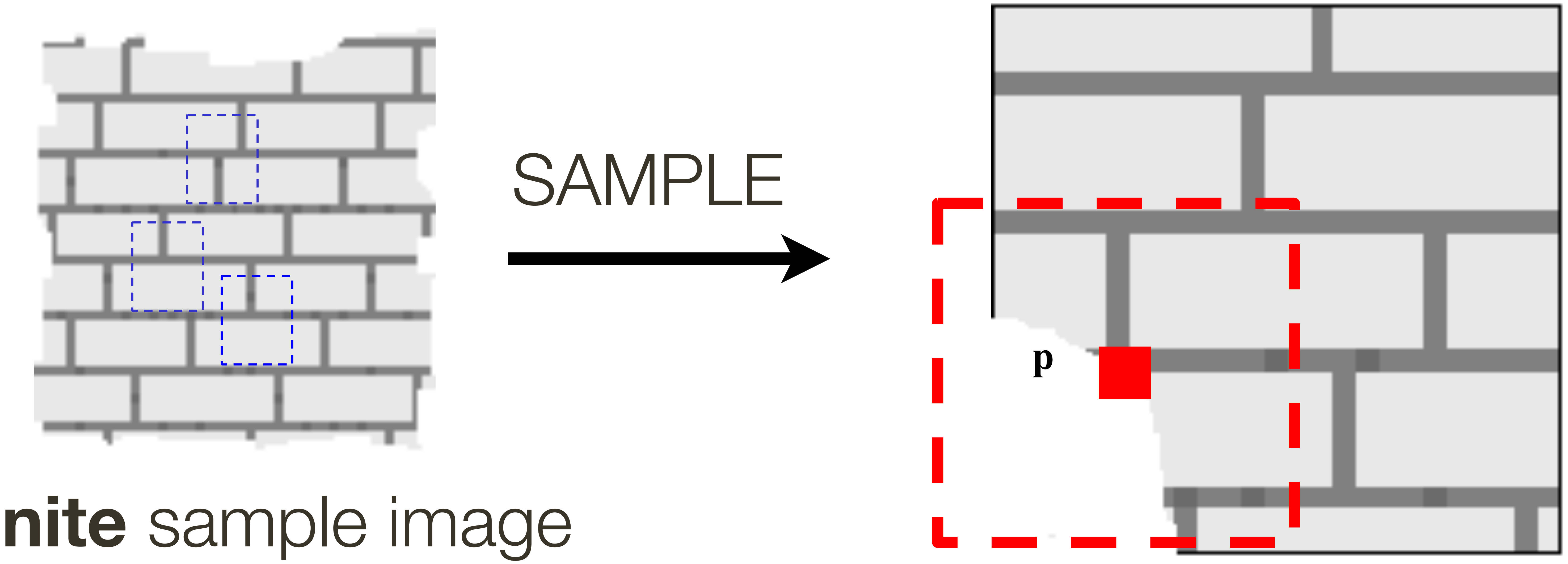
# **Efros** and Leung: Synthesizing One Pixel




**Infinite** sample image

— Since the sample image is finite, an exact neighbourhood match might not be present

# **Efros** and Leung: Synthesizing One Pixel

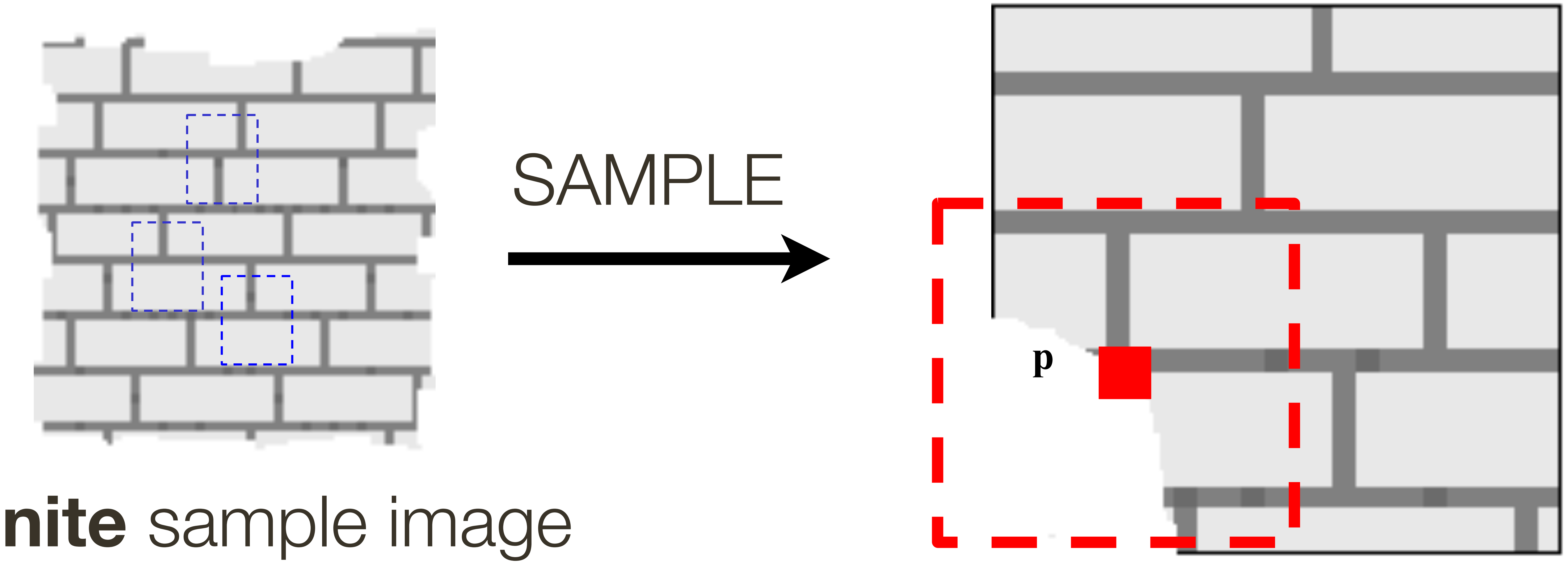



**Infinite** sample image

— Since the sample image is finite, an exact neighbourhood match might not be present

— Find the **best match** using SSD error, weighted by Gaussian to emphasize local structure, and take all samples within some distance from that match

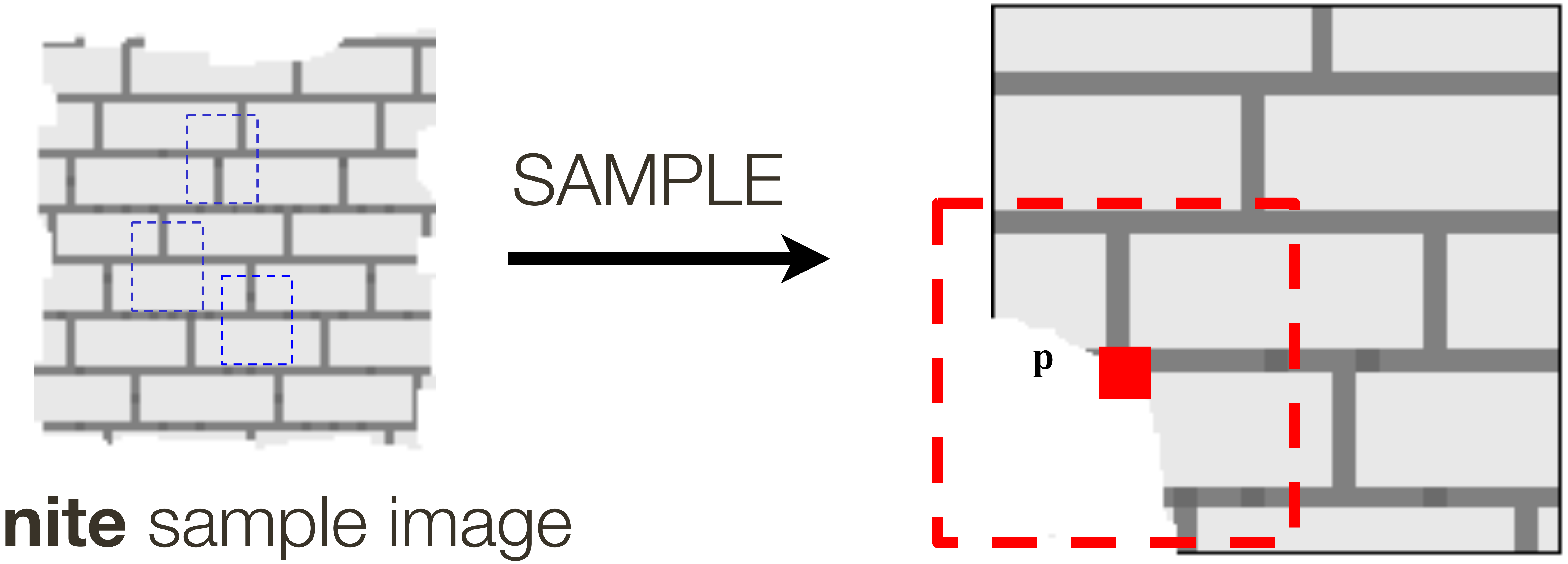
# **Efros** and Leung: Synthesizing One Pixel




**Infinite** sample image

| **Ranked List** | **Similarity (cos)** |
| --- | --- |
| x = 5, y = 17 | 0.87 |
| x = 63, y = 4 | 0.75 |
| x = 3, y = 44 | 0.72 |
| x = 123, y = 54 | 0.64 |
| x = 4, y = 57 | 0.60 |
| ⋮ | ⋮ |

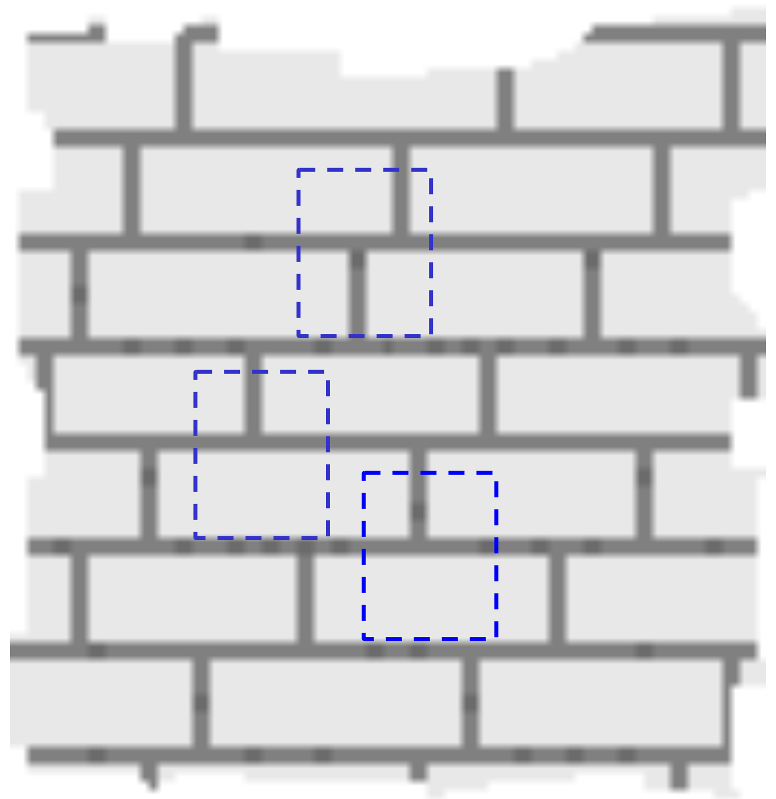
# **Efros** and Leung: Synthesizing One Pixel




**Infinite** sample image

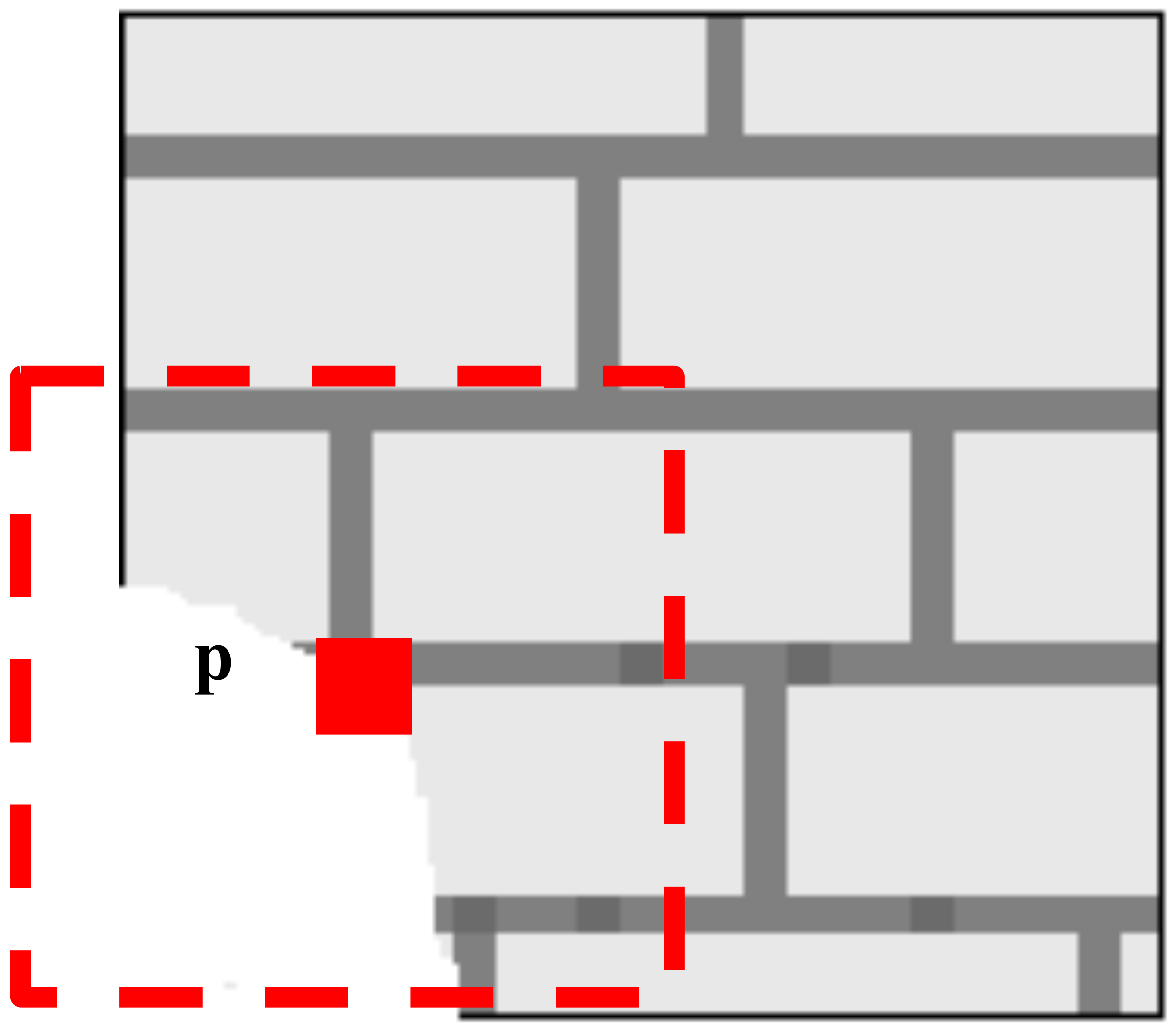
| **Ranked List** | **Similarity (cos)** | |
|---|---|---|
| x = 5, y = 17 | 0.87 | ← best match |
| x = 63, y = 4 | 0.75 | |
| x = 3, y = 44 | 0.72 | |
| x = 123, y = 54 | 0.64 | |
| x = 4, y = 57 | 0.60 | |
| ⋮ | ⋮ | |

# **Efros** and Leung: Synthesizing One Pixel



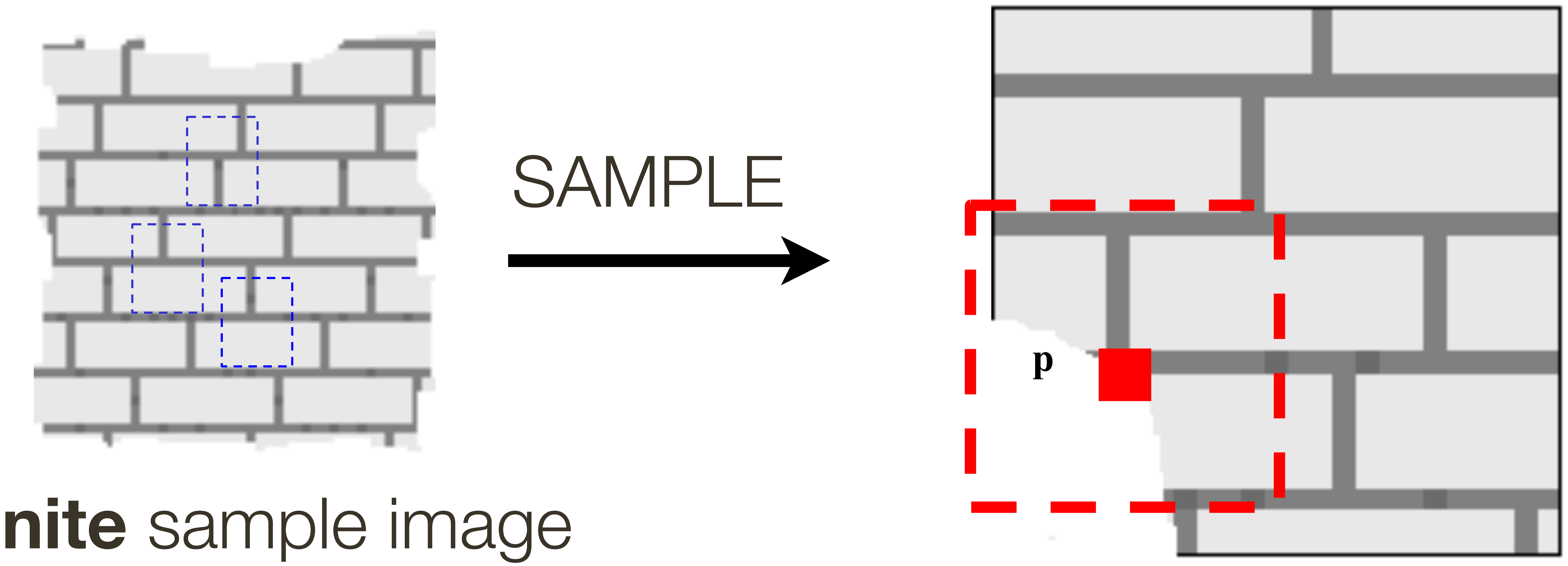
**Infinite** sample image

SAMPLE →




| Ranked List | Similarity (cos) | |
|---|---|---|
| x = 5, y = 17 | 0.87 | ← best match |
| x = 63, y = 4 | 0.75 | |
| x = 3, y = 44 | 0.72 | |
| | | threshold = best match * **0.8** = 0.696 |
| x = 123, y = 54 | 0.64 | |
| x = 4, y = 57 | 0.60 | |
| ⋮ | ⋮ | |

# **Efros** and Leung: Synthesizing One Pixel

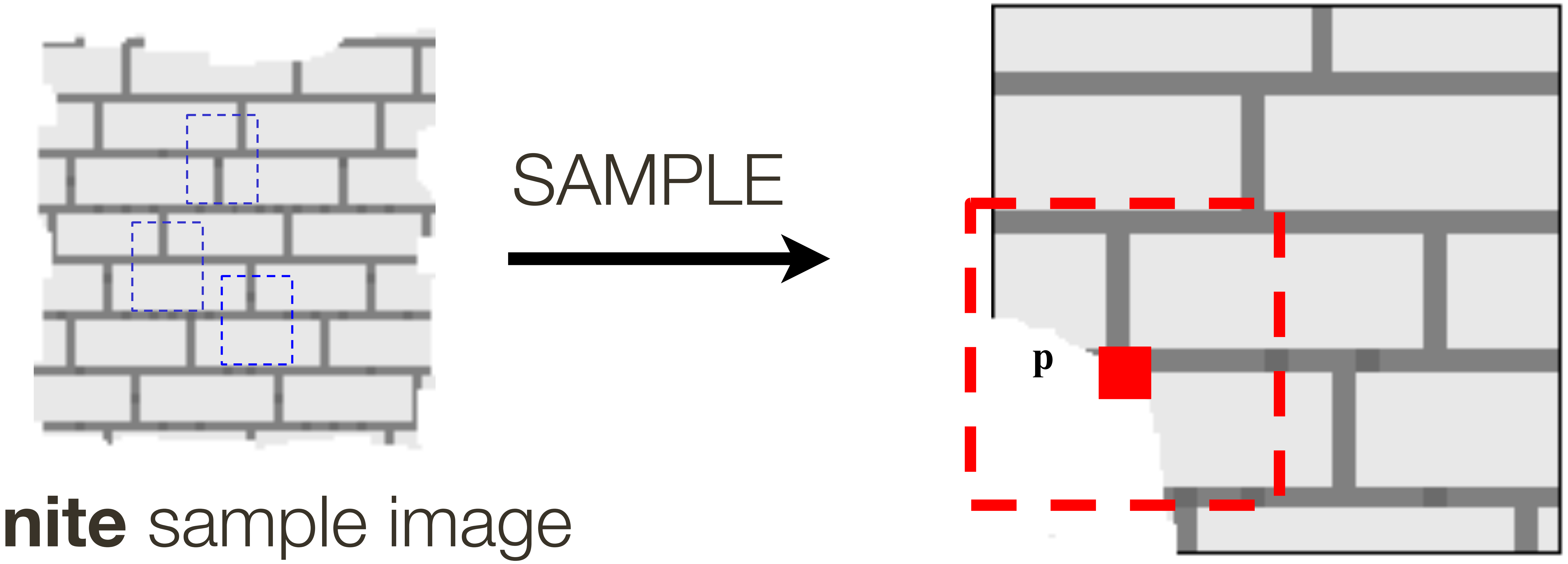



**Infinite** sample image

| **Ranked List** | **Similarity (cos)** |
|---|---|
| x = 5, y = 17 | **0.87** ← best match |
| x = 63, y = 4 | **0.75** |
| x = 3, y = 44 | **0.72** |
| x = 123, y = 54 | 0.64 |
| x = 4, y = 57 | 0.60 |
| ⋮ | ⋮ |

threshold = best match * **0.8** = 0.696

# **Efros** and Leung: Synthesizing One Pixel



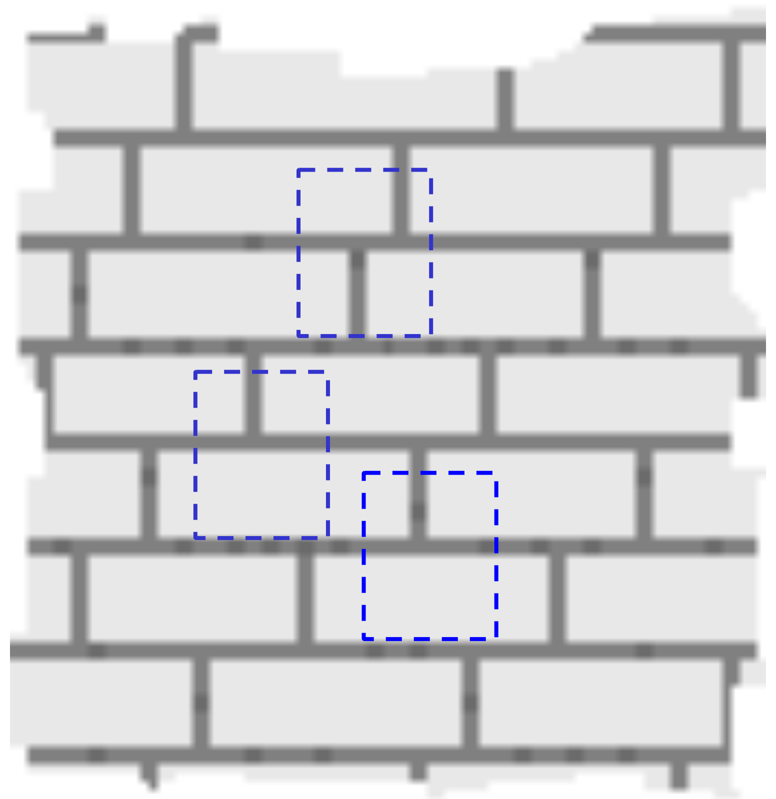

**Infinite** sample image

| **Ranked List** | **Similarity (cos)** |
| --- | --- |
| x = 5, y = 17 | **0.87** |
| x = 63, y = 4 | **0.75** |
| x = 3, y = 44 | **0.72** |
| x = 123, y = 54 | 0.64 |
| x = 4, y = 57 | 0.60 |
| ⋮ | ⋮ |

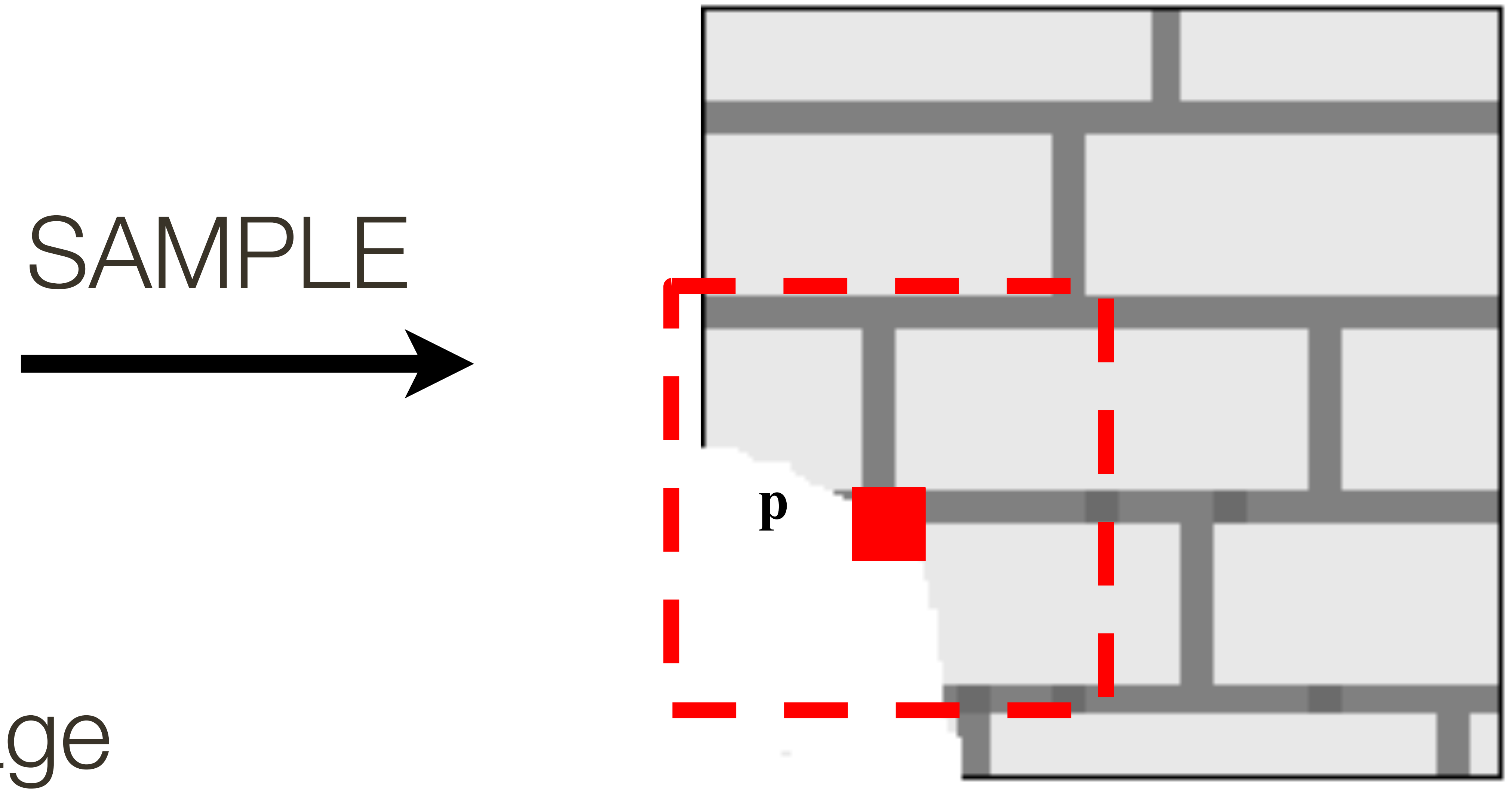
pick one at random and copy target pixel from it

threshold = best match * **0.8** = 0.696

# **Efros** and Leung: Synthesizing One Pixel



**Infinite** sample image




| **Ranked List** | **Similarity (ssd)** |
|---|---|
| x = 5, y = 17 | **0.13** |
| x = 63, y = 4 | **0.25** |
| x = 3, y = 44 | **0.28** |
| x = 123, y = 54 | 0.36 |
| x = 4, y = 57 | 0.40 |
| ⋮ | ⋮ |

pick one at random and copy target pixel from it

threshold = best match * **2.5** = 0.325

# **Efros** and Leung: Synthesizing Many Pixels

For multiple pixels, "grow" the texture in layers

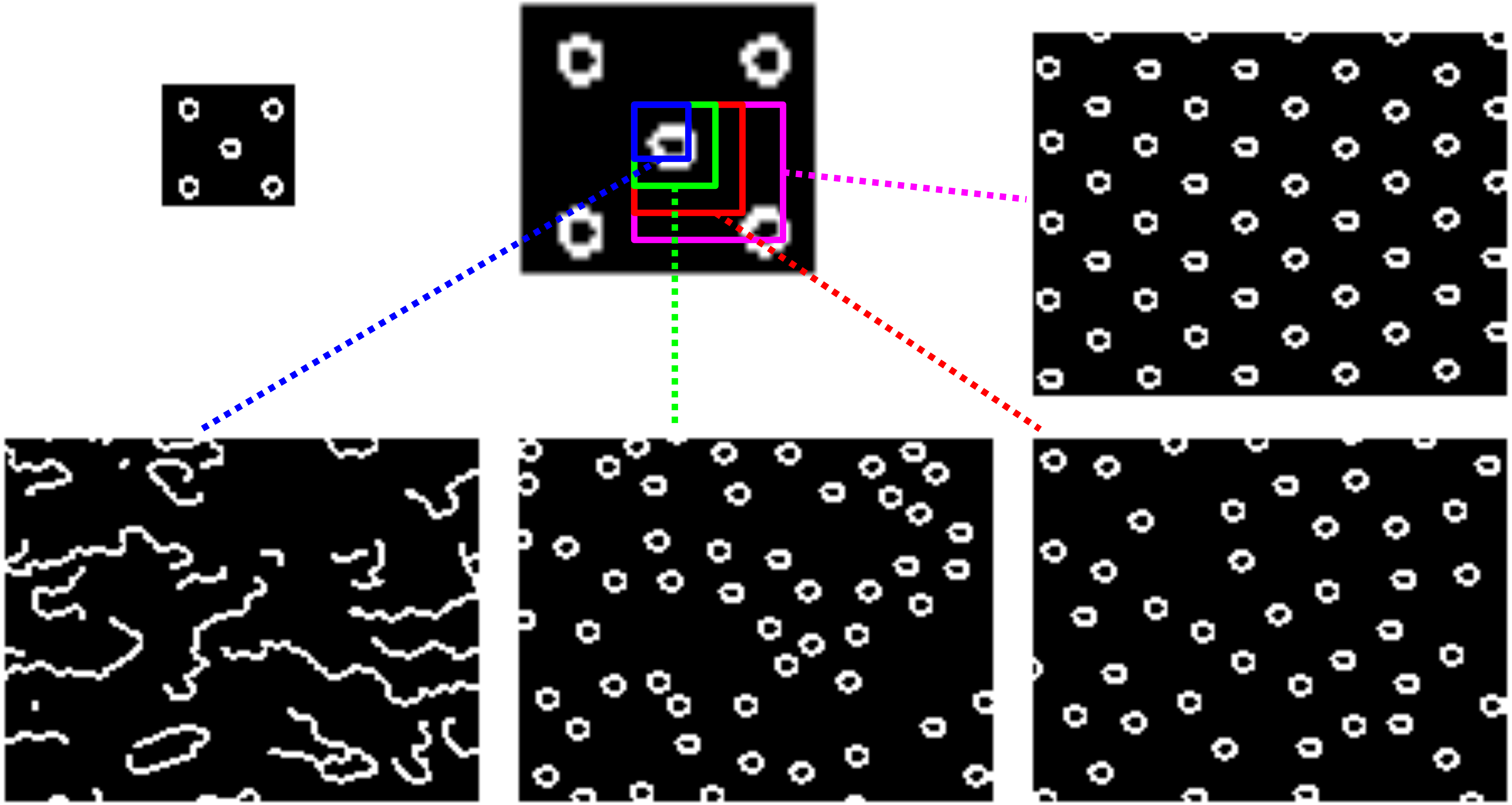
— In the case of hole-filling, start from the edges of the hole

For an interactive demo, see

https://una-dinosauria.github.io/efros-and-leung-js/

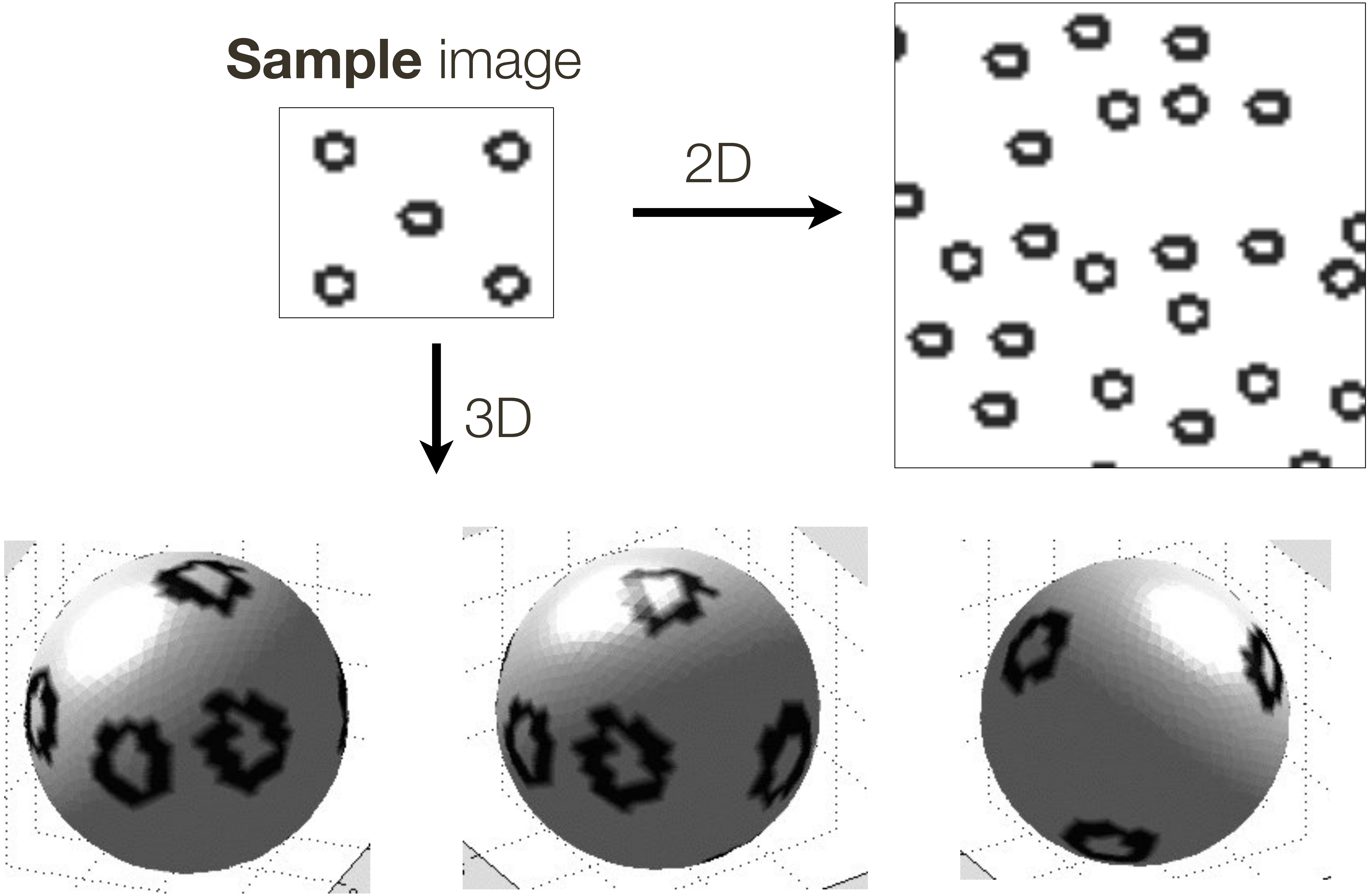
(written by Julieta Martinez, a previous CPSC 425 TA)

# **Randomness** Parameter



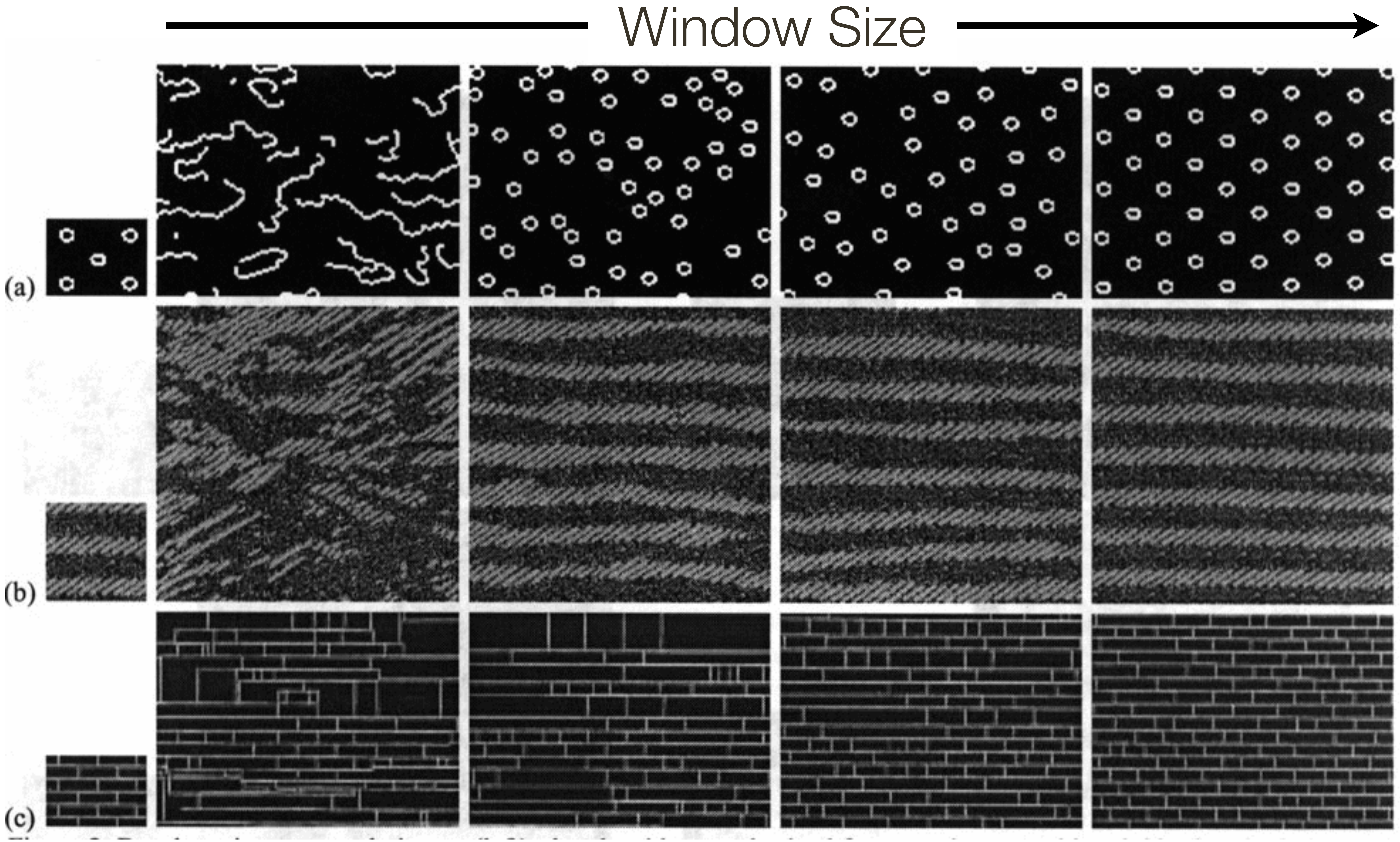
**Slide Credit:** http://graphics.cs.cmu.edu/people/efros/research/NPS/efros-iccv99.ppt

# **Texturing** a Sphere



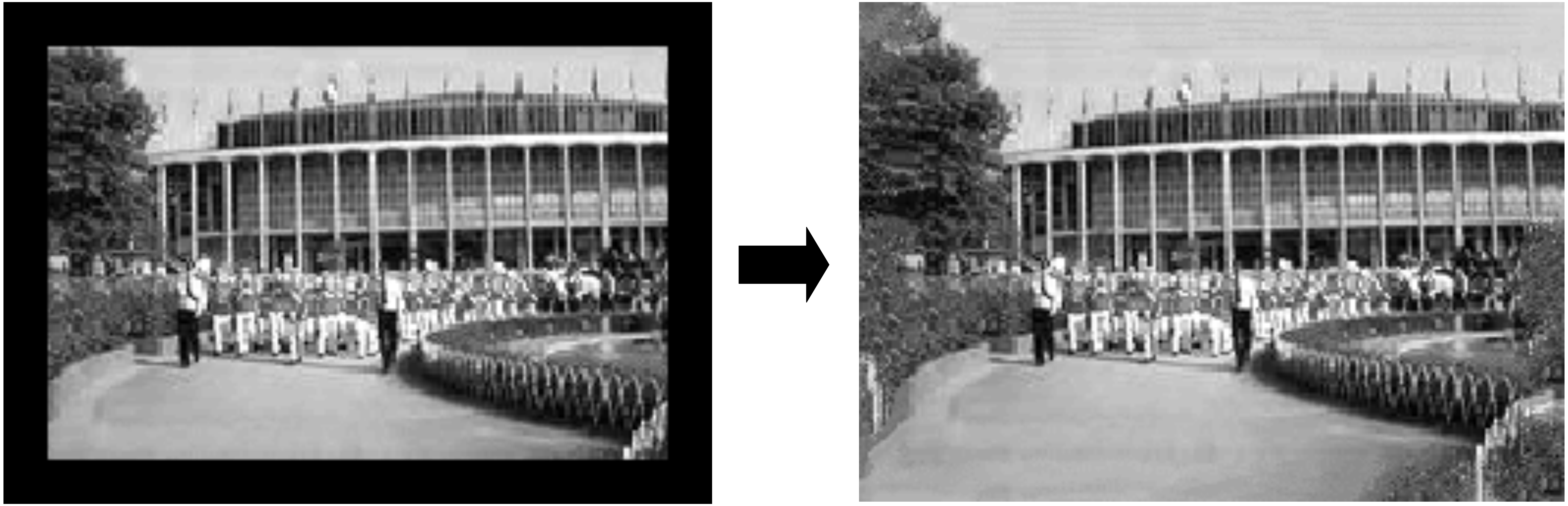

**Slide Credit:** http://graphics.cs.cmu.edu/people/efros/research/NPS/efros-iccv99.ppt

# **Efros** and Leung: More Synthesis Results




Forsyth & Ponce (2nd ed.) Figure 6.12

# **Efros** and Leung: Image Extrapolation



**Slide Credit:** http://graphics.cs.cmu.edu/people/efros/research/NPS/efros-iccv99.ppt

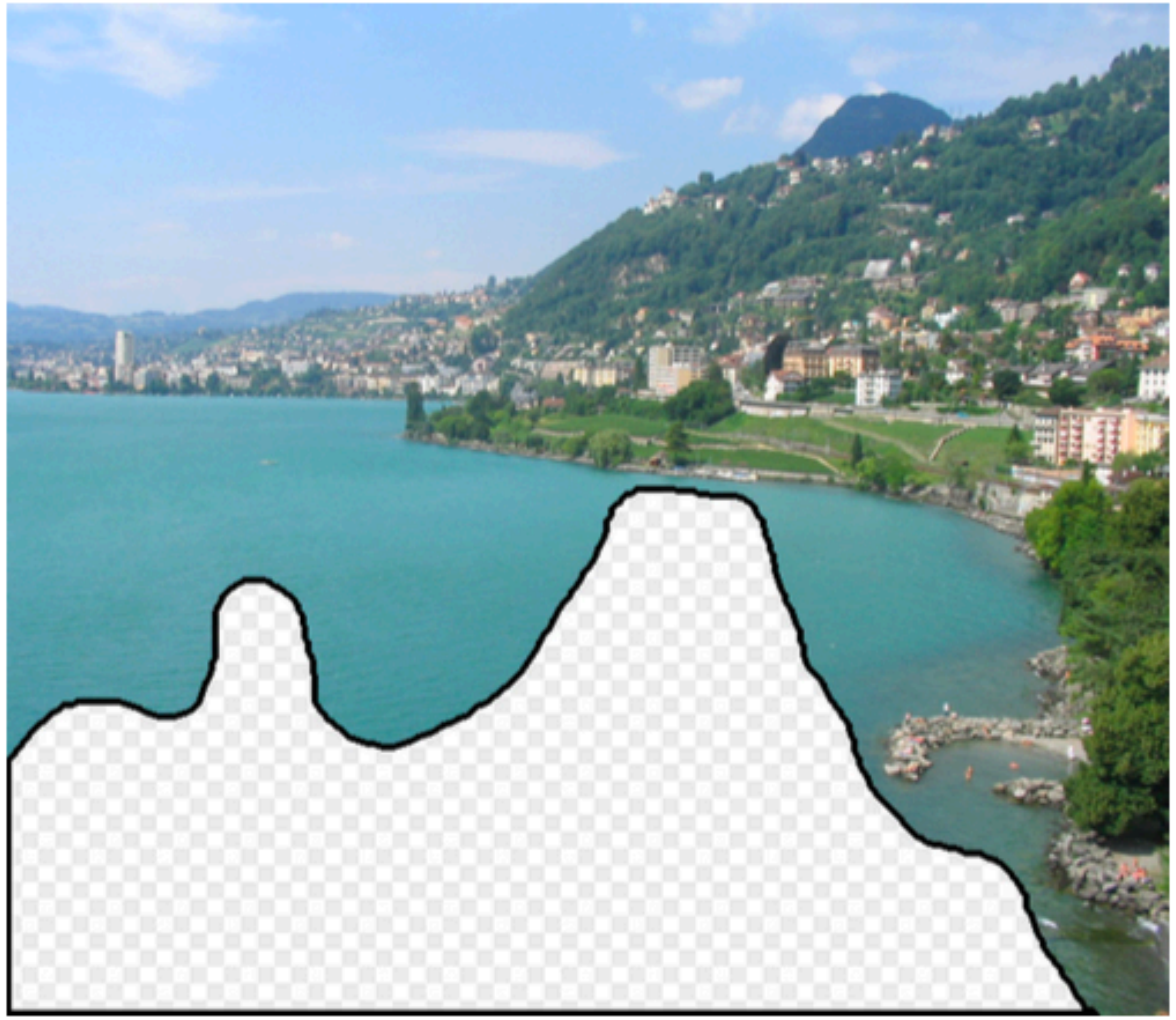
# “**Big** Data” Meets Inpainting

“**Big** Data" enables surprisingly simple non-parametric, matching-based techniques to solve complex problems in computer graphics and vision.

Suppose instead of a single image, you had a massive database of a million images. What could you do?

# “**Big** Data” Meets Inpainting

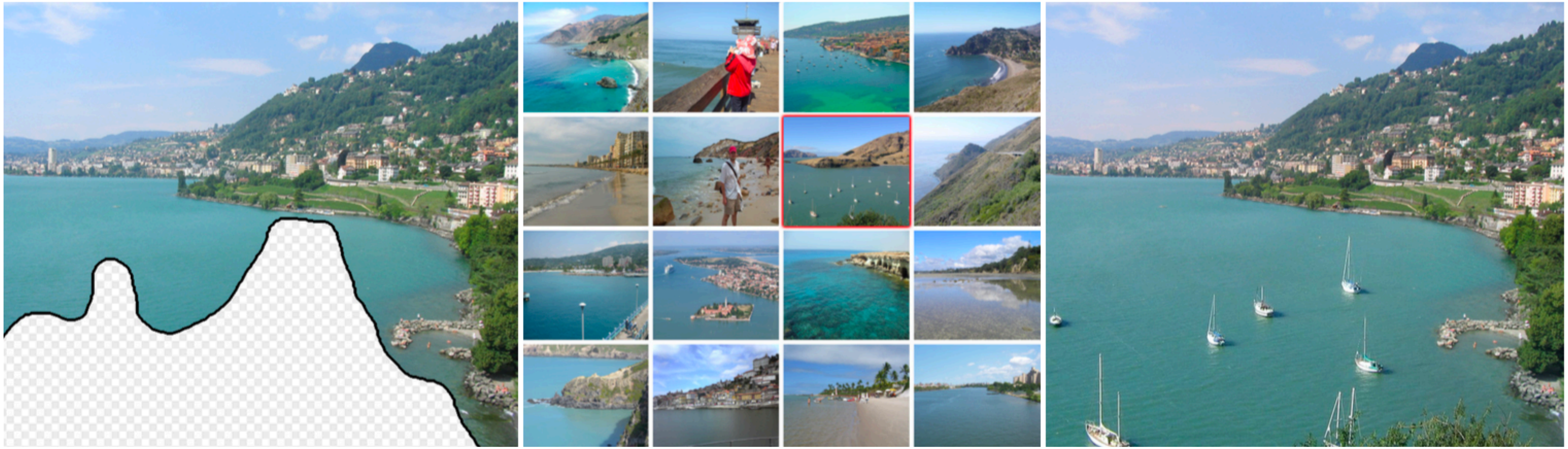
Original Image



Input

**Figure Credit**: Hays and Efros 2007

# “**Big** Data” Meets Inpainting



Input Scene Matches Output

**Figure Credit**: Hays and Efros 2007

# **Effectiveness** of "Big Data"

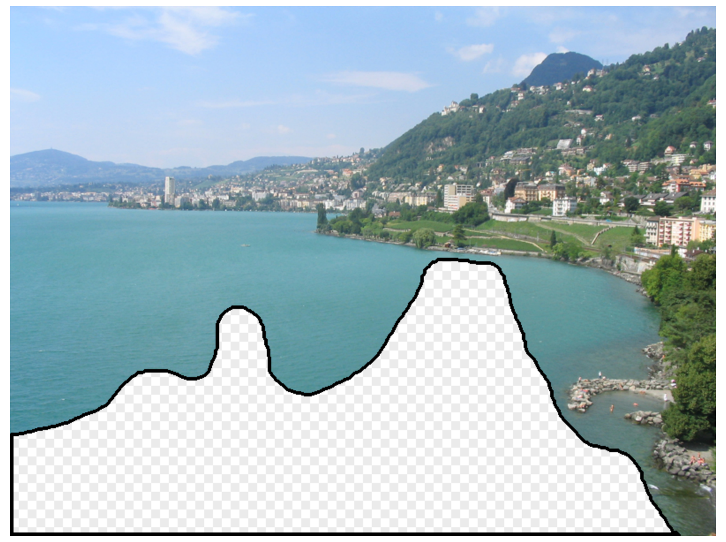
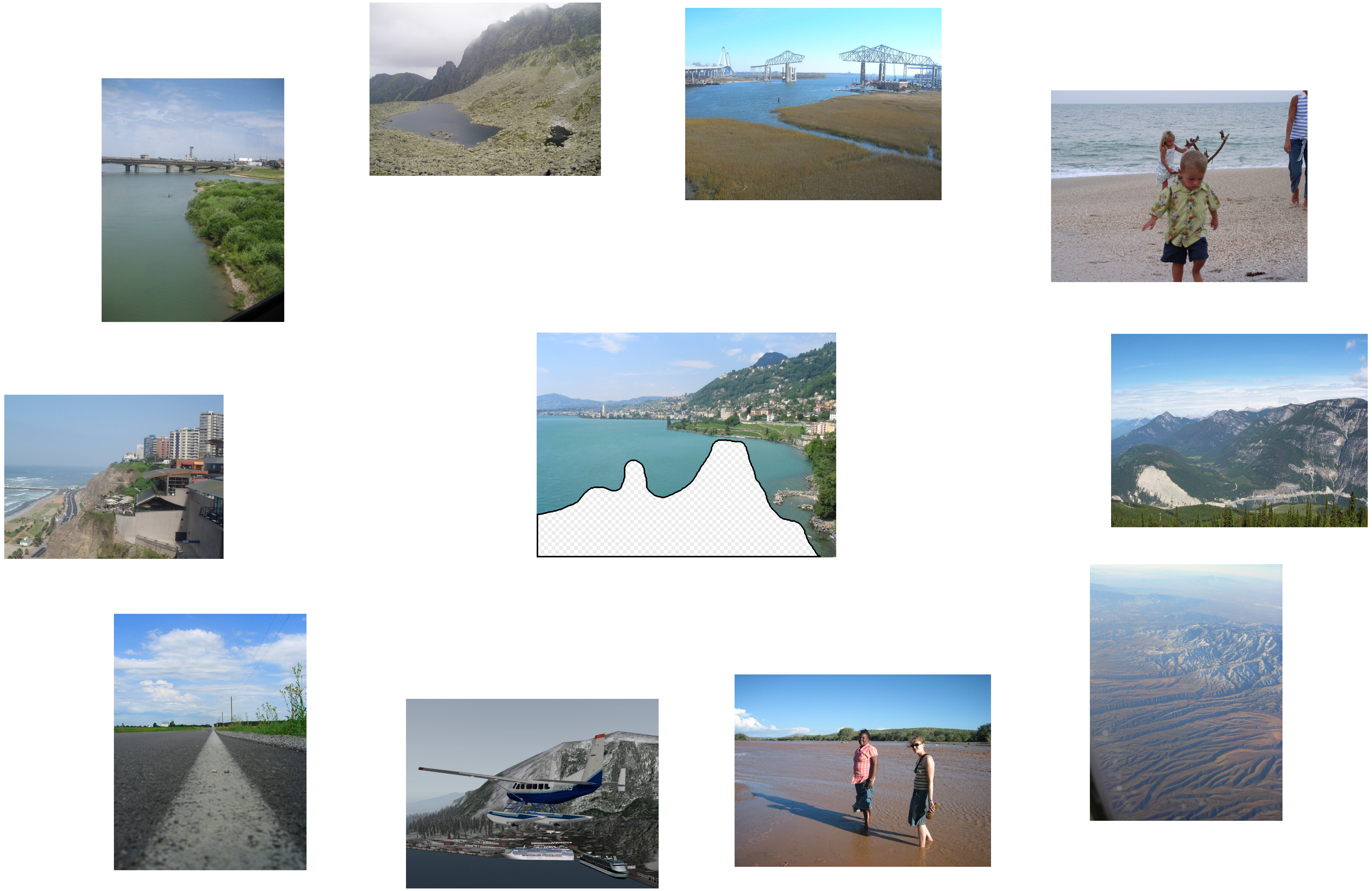


**Figure Credit**: Hays and Efros 2007

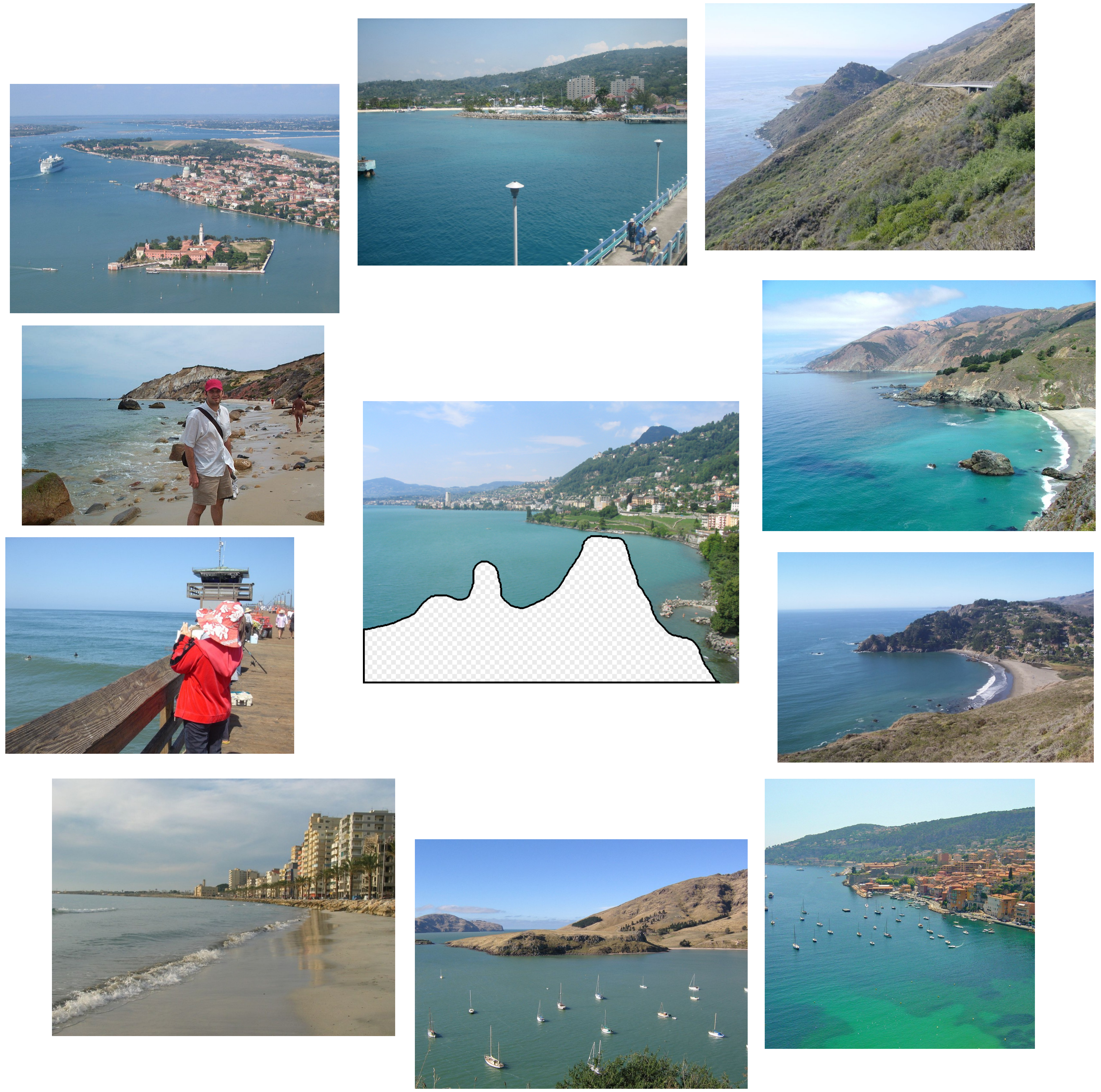
# **Effectiveness** of “Big Data”



10 nearest neighbors from a collection of 20,000 images

**Figure Credit**: Hays and Efros 2007

# **Effectiveness** of "Big Data"



10 nearest neighbors from a collection of 2 million images

**Figure Credit**: Hays and Efros 2007

# “**Big** Data” Meets Inpainting

**Figure Credit**: Hays and Efros 2007

# “**Big** Data” Meets Inpainting

**Algorithm** sketch (Hays and Efros 2007):

1. Create a short list of a few hundred “best matching" images based on global image statistics

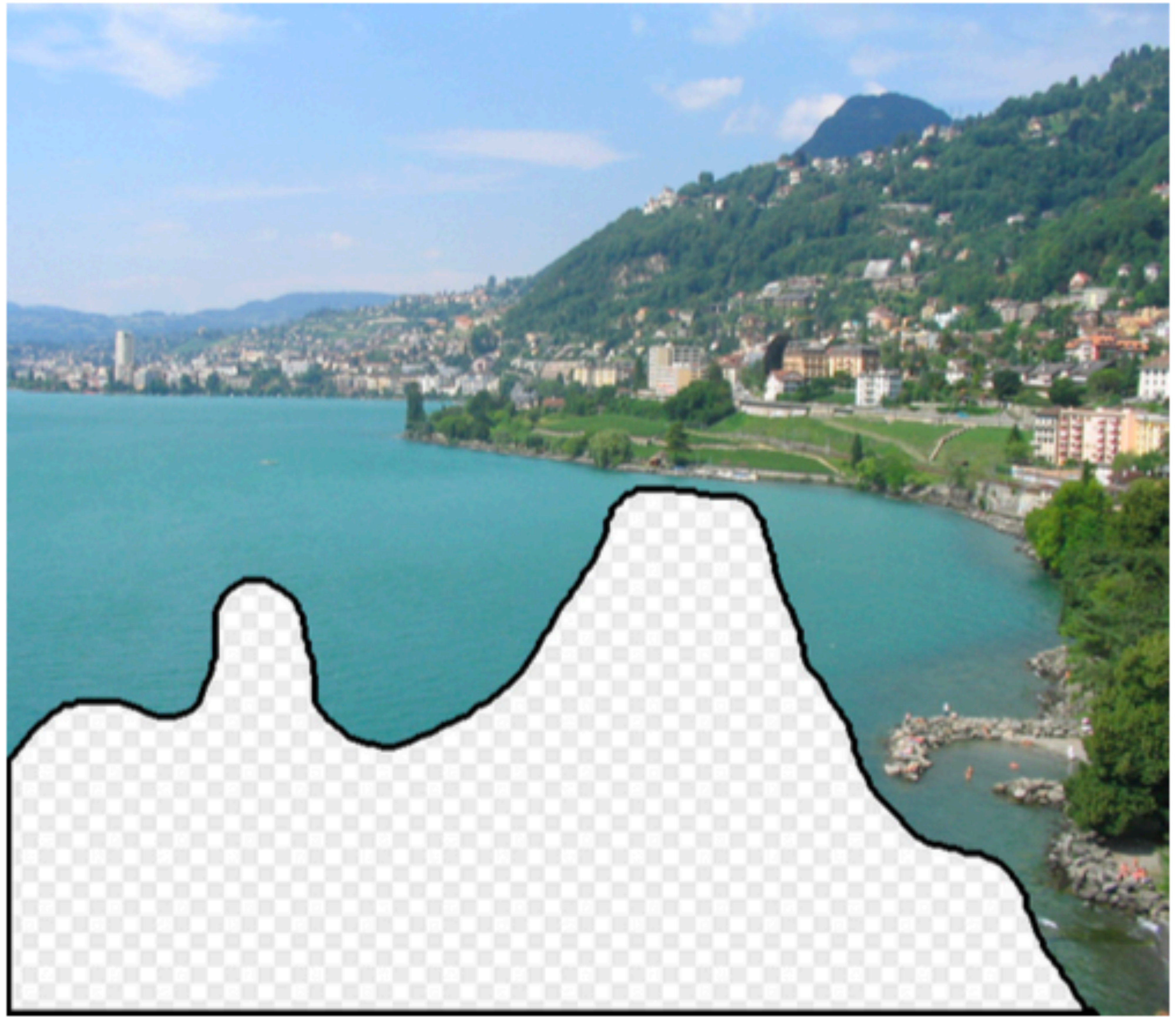
2. Find patches in the short list that match the context surrounding the image region we want to fill

3. Blend the match into the original image

Purely **data-driven**, requires no manual labeling of images

# “**Big** Data” Meets Inpainting

Original Image



Input

**Figure Credit**: Hays and Efros 2007

# “**Big** Data” Meets Inpainting

**Figure Credit**: Hays and Efros 2007

# “**Big** Data” Meets Inpainting

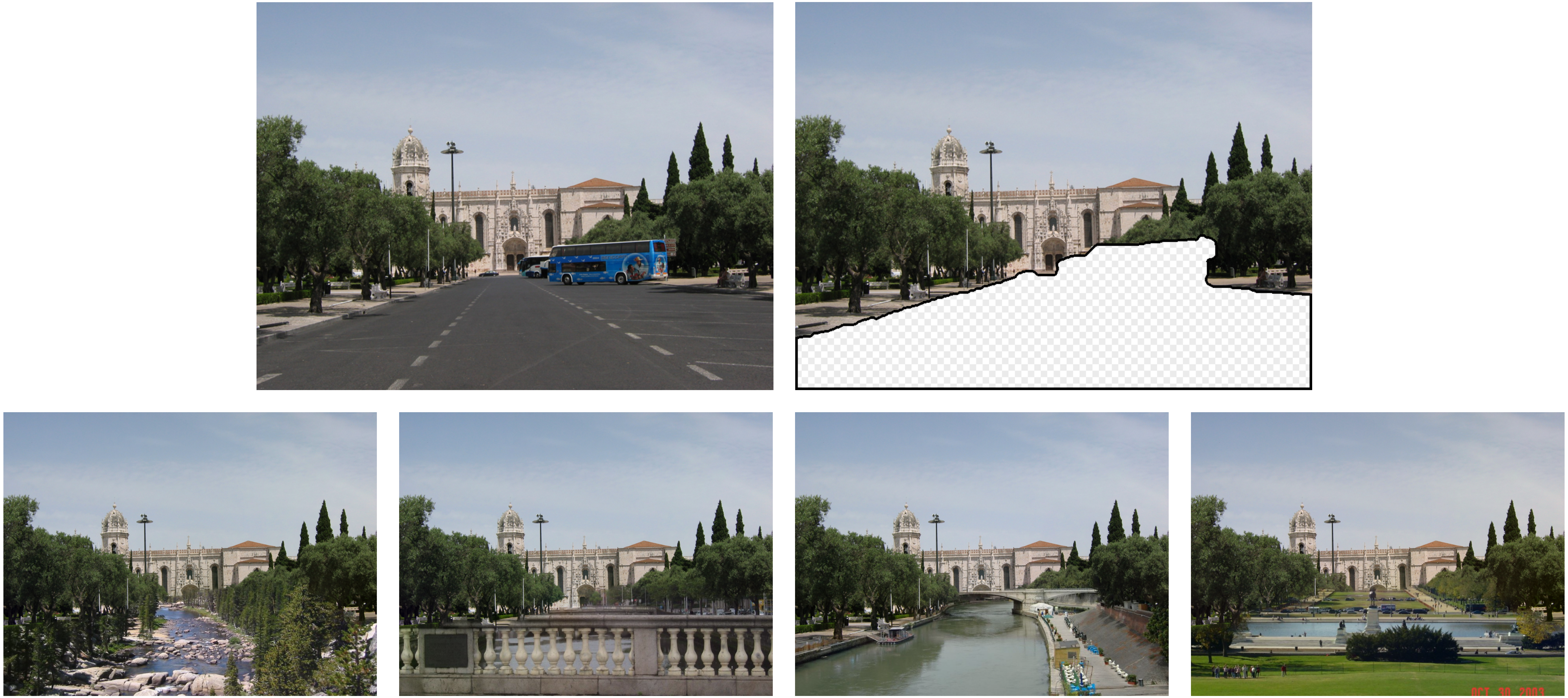


**Figure Credit**: Hays and Efros 2007

Optional subtitle

# Texture

We will look at two main questions:

1. How do we represent texture?
   → Texture **analysis**

2. How do we generate new examples of a texture?
   → Texture **synthesis**

# Texture **Segmentation**

**Question**: Is texture a property of a point or a property of a region?

# Texture **Segmentation**

**Question**: Is texture a property of a point or a property of a region?

**Answer**: We need a region to have a texture.

# Texture **Segmentation**

**Question**: Is texture a property of a point or a property of a region?

**Answer**: We need a region to have a texture.

There is a “chicken–and–egg” problem. Texture segmentation can be done by detecting boundaries between regions of the same (or similar) texture. Texture boundaries can be detected using standard edge detection techniques applied to the texture measures determined at each point

# **Recall**: Boundary Detection

**Features**:

— Raw Intensity

— Orientation Energy

— Brightness Gradient

— Color Gradient

— Texture gradient

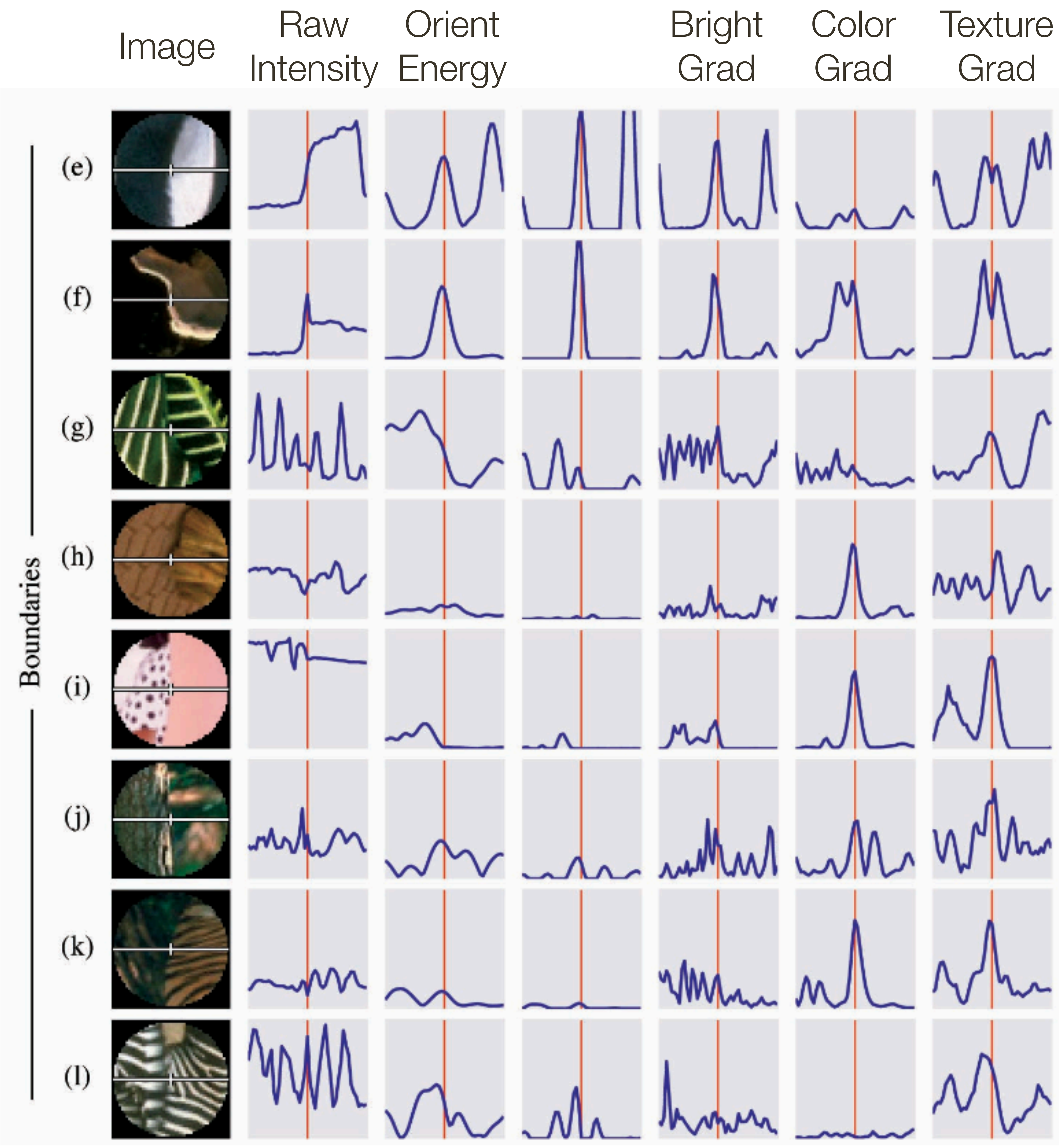



**Figure Credit**: Martin et al. 2004

# Texture **Segmentation**

**Question**: Is texture a property of a point or a property of a region?

**Answer**: We need a region to have a texture.

There is a "chicken–and–egg" problem. Texture segmentation can be done by detecting boundaries between regions of the same (or similar) texture. Texture boundaries can be detected using standard edge detection techniques applied to the texture measures determined at each point

We compromise! Typically one uses a local window to estimate texture properties and assigns those texture properties as point properties of the window's center row and column

# Texture **Representation**

**Question**: How many degrees of freedom are there to texture?

# Texture **Representation**

**Question**: How many degrees of freedom are there to texture?

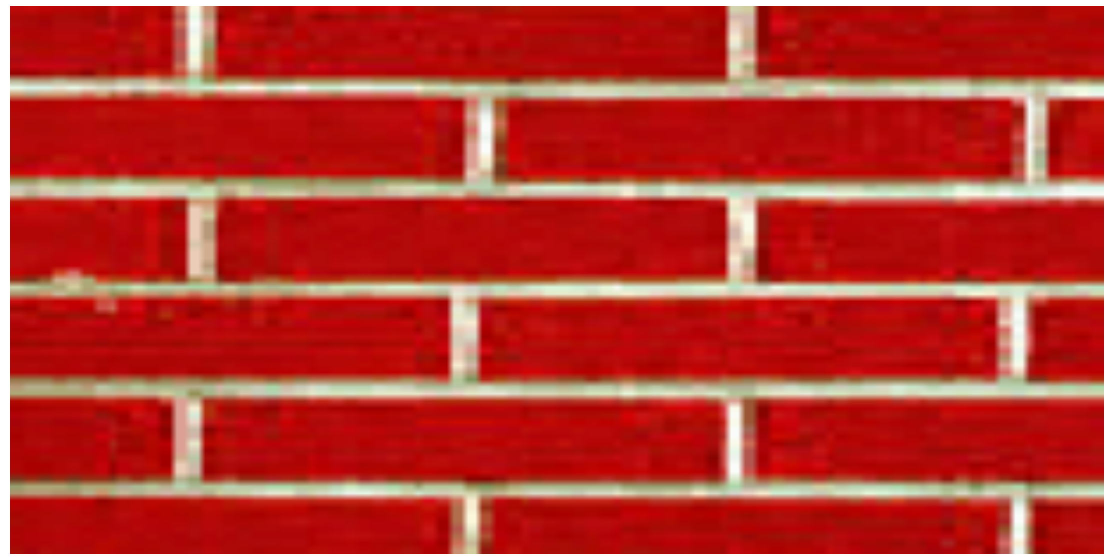
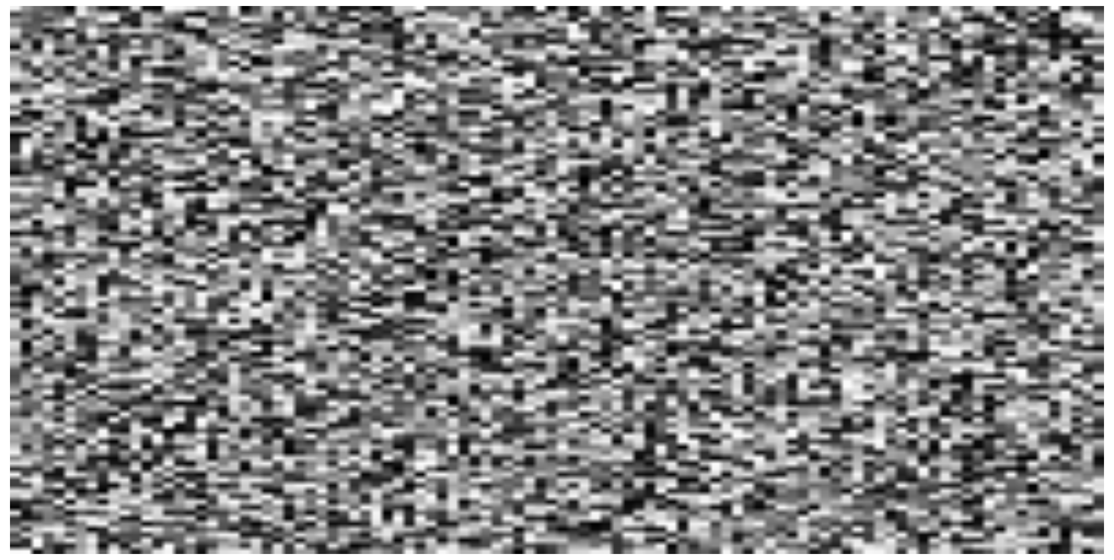
(**Mathematical**) Answer: Infinitely many

(**Perceptual Psychology**) Answer: There are perceptual constraints. But, there is no clear notion of a “texture channel” like, for example, there is for an RGB colour channel

# 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 point in the image with a summary of the pattern of sub-elements in the local region


# 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 point in the image with a summary of the pattern of sub-elements in the local region

**Question**: What filters should we use?

**Answer**: Human vision suggests spots and oriented edge filters at a variety of different orientations and scales

# Texture **Representation**

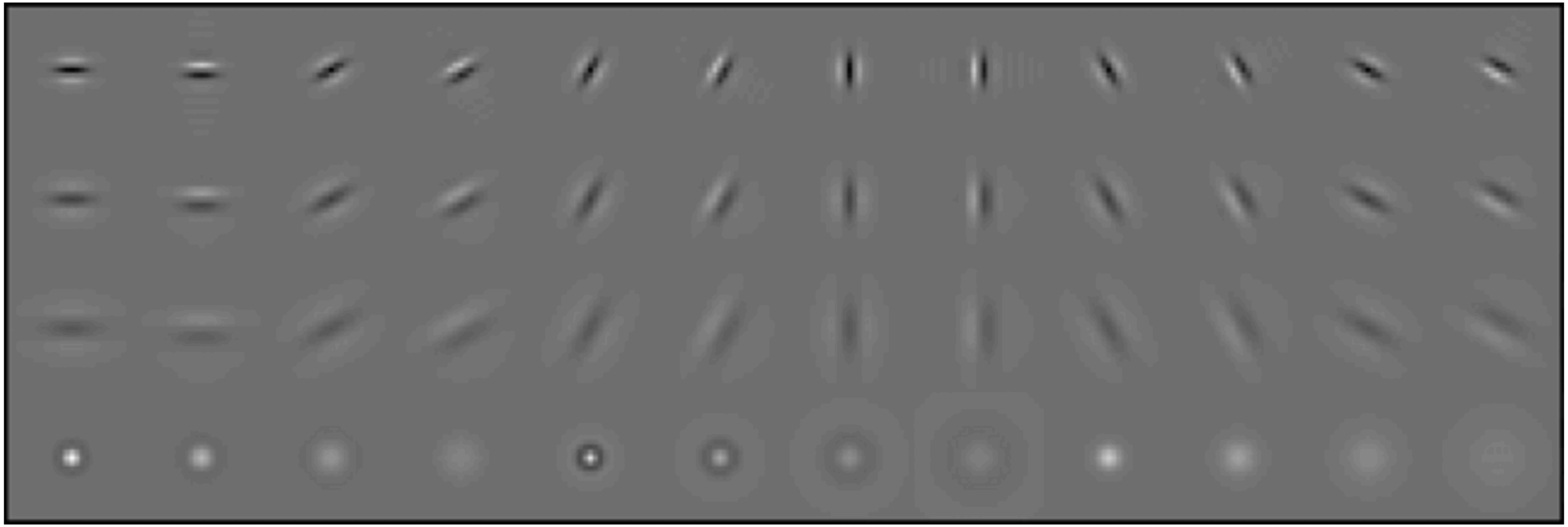


**Figure Credit**: Leung and Malik, 2001

# Texture **Representation**

First derivative of Gaussian at 6 orientations and 3 scales

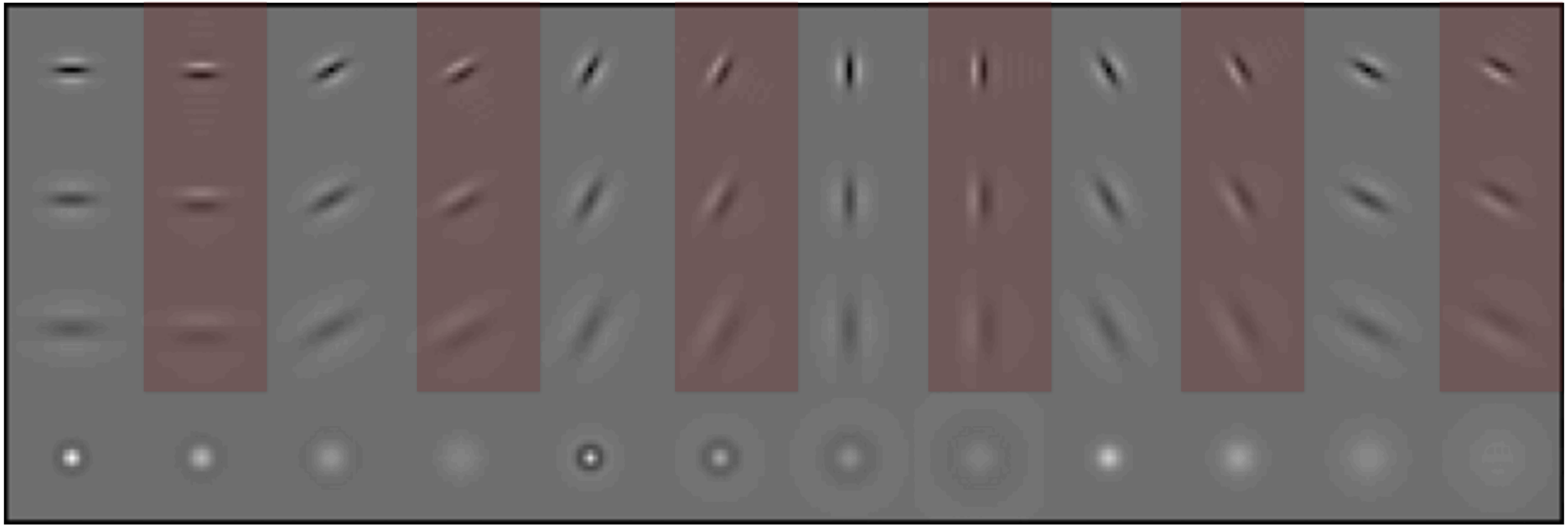


**Figure Credit**: Leung and Malik, 2001

# Texture **Representation**

Second derivative of Gaussian at 6 orientations 3 scales

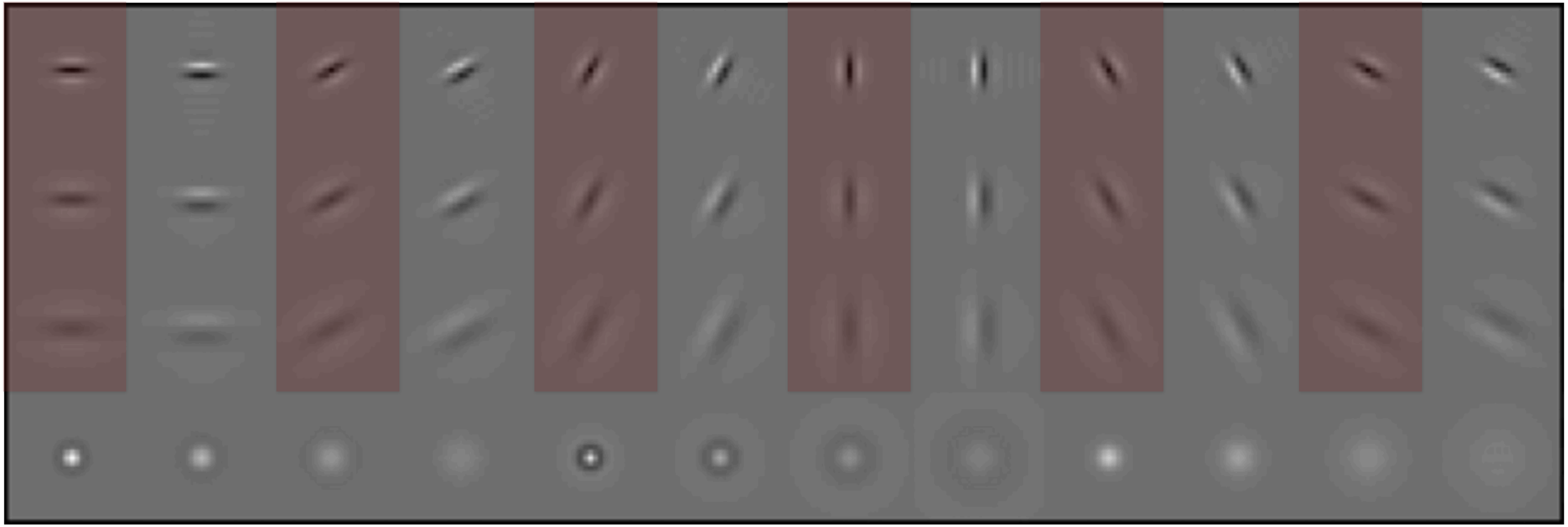


**Figure Credit**: Leung and Malik, 2001

# Texture **Representation**

Laplacian of the Gaussian filters at different scales

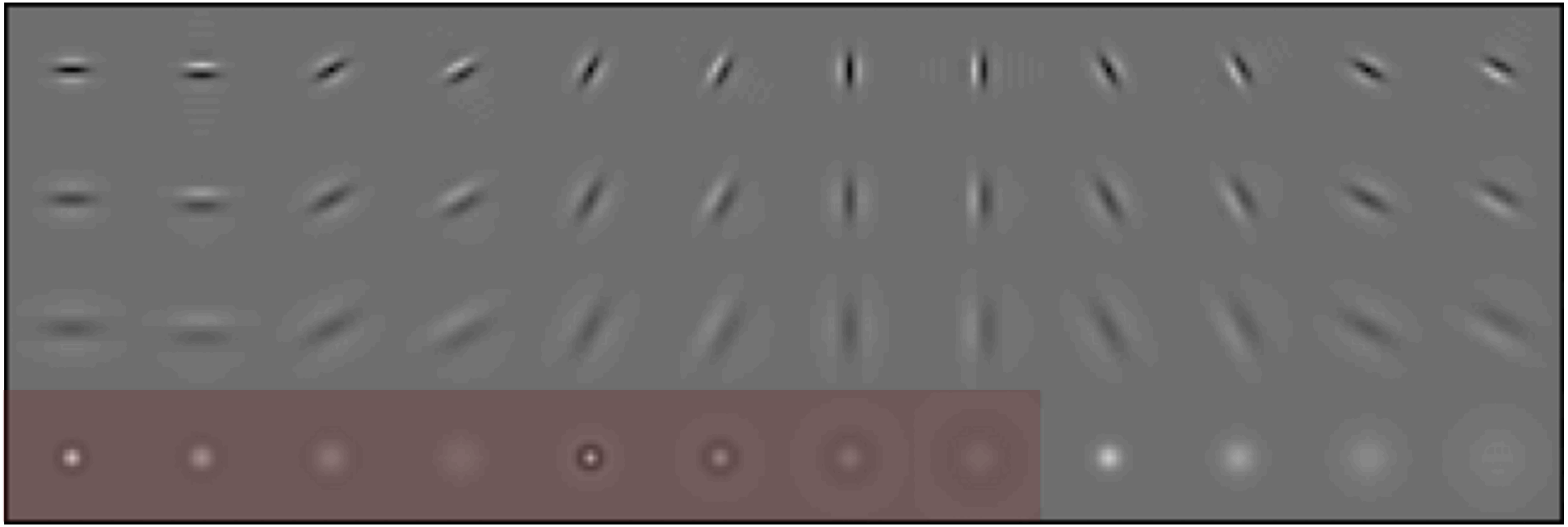


**Figure Credit**: Leung and Malik, 2001

# Texture **Representation**

Gaussian filters at different scales

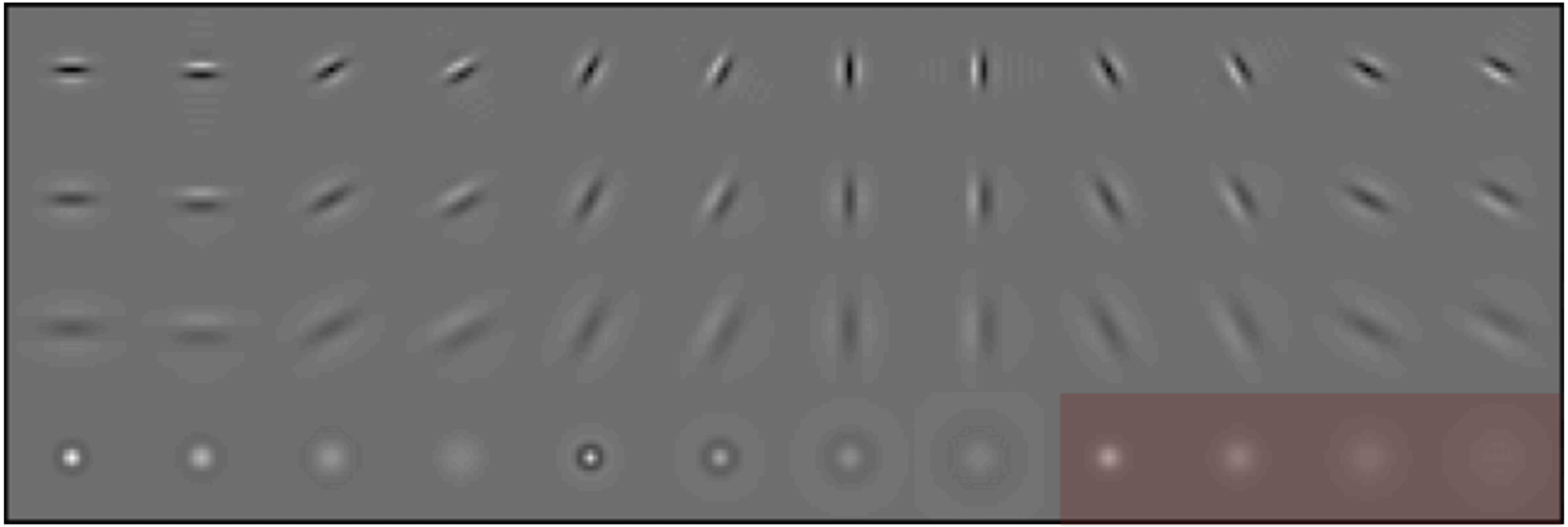
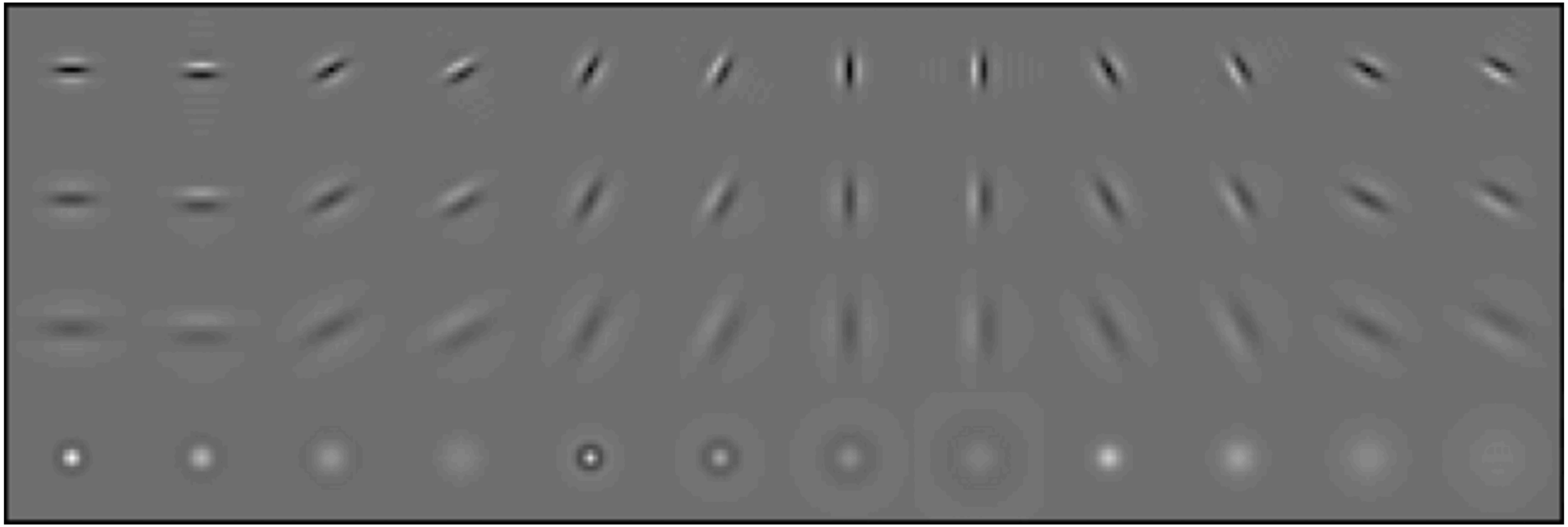


**Figure Credit**: Leung and Malik, 2001

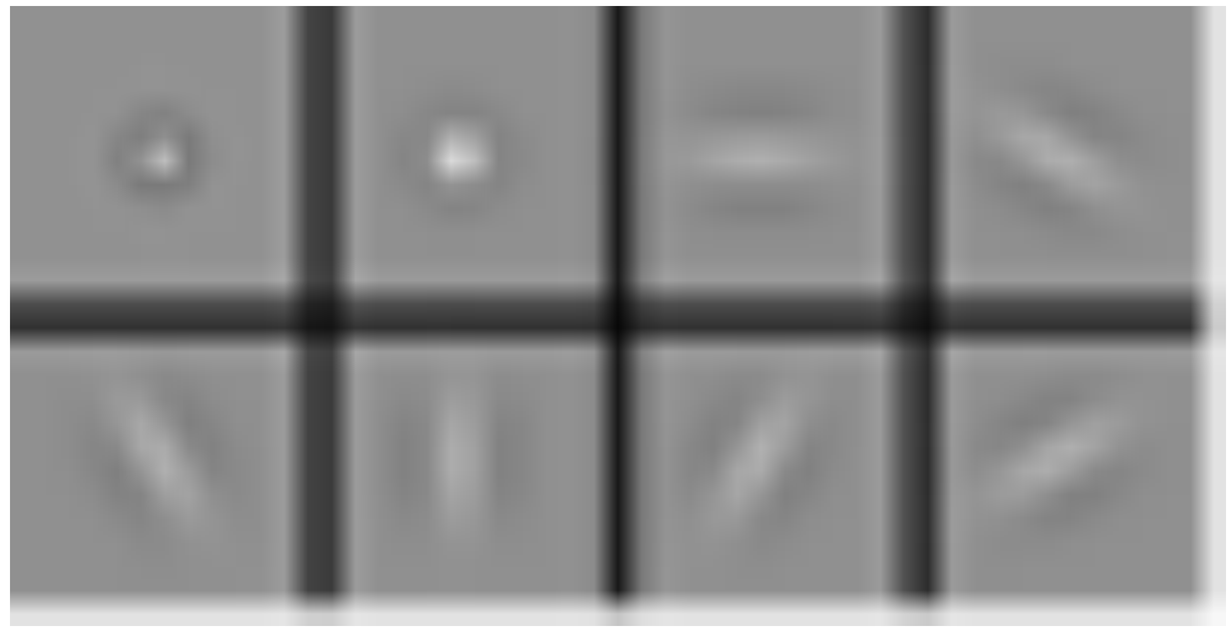
# Texture **Representation**

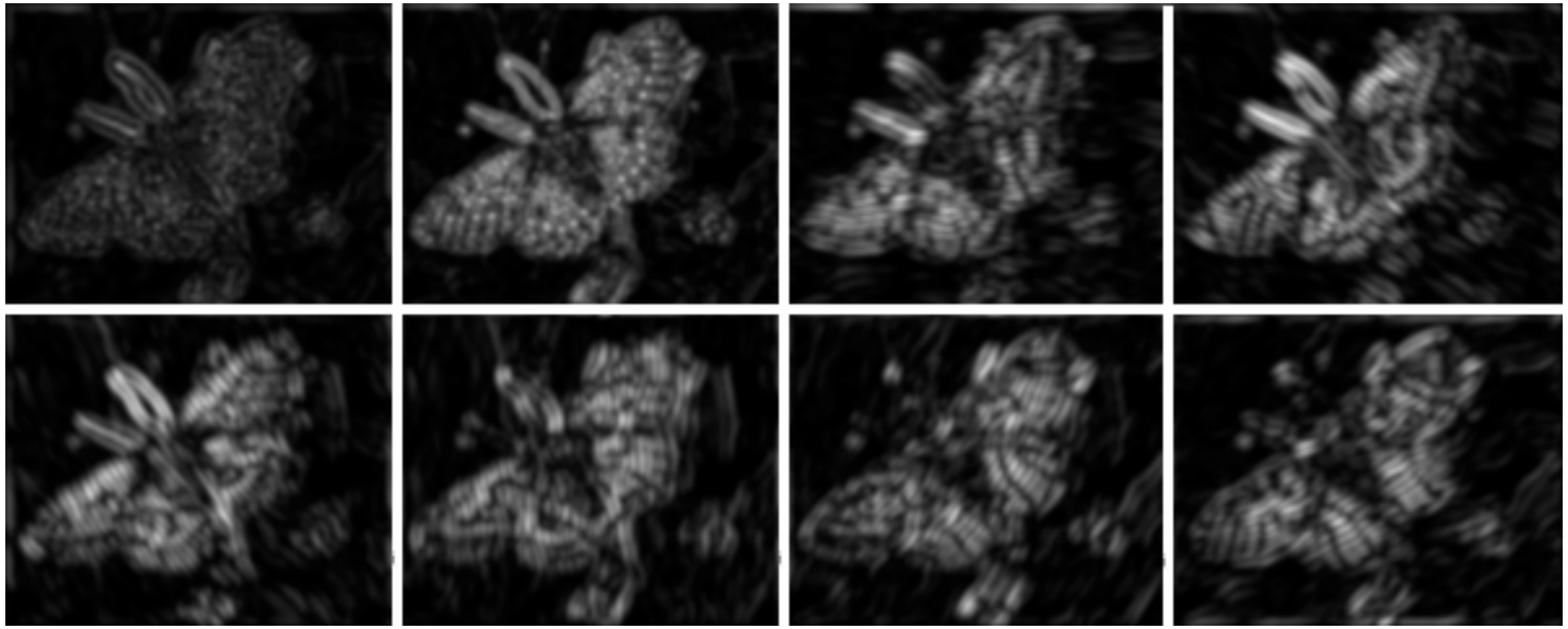


**Result**: 48-channel “image”

**Figure Credit**: Leung and Malik, 2001

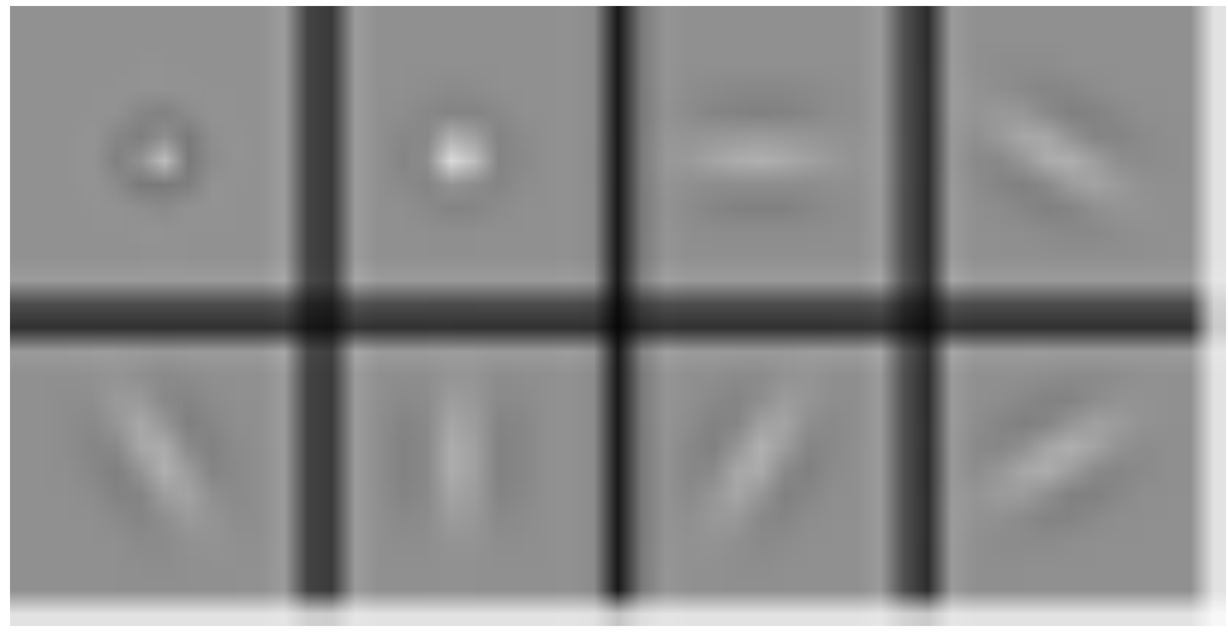
# **Spots** and Bars (Fine Scale)

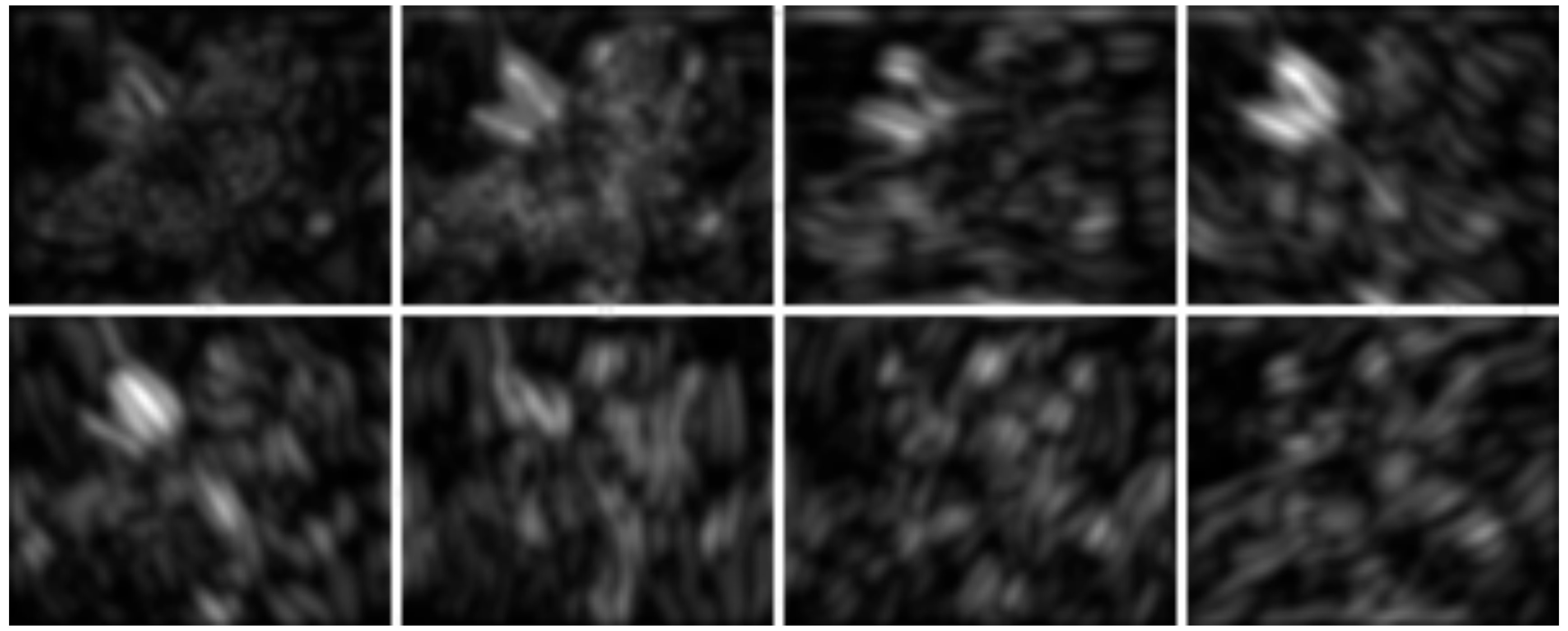




Forsyth & Ponce (1st ed.) Figures 9.3–9.4

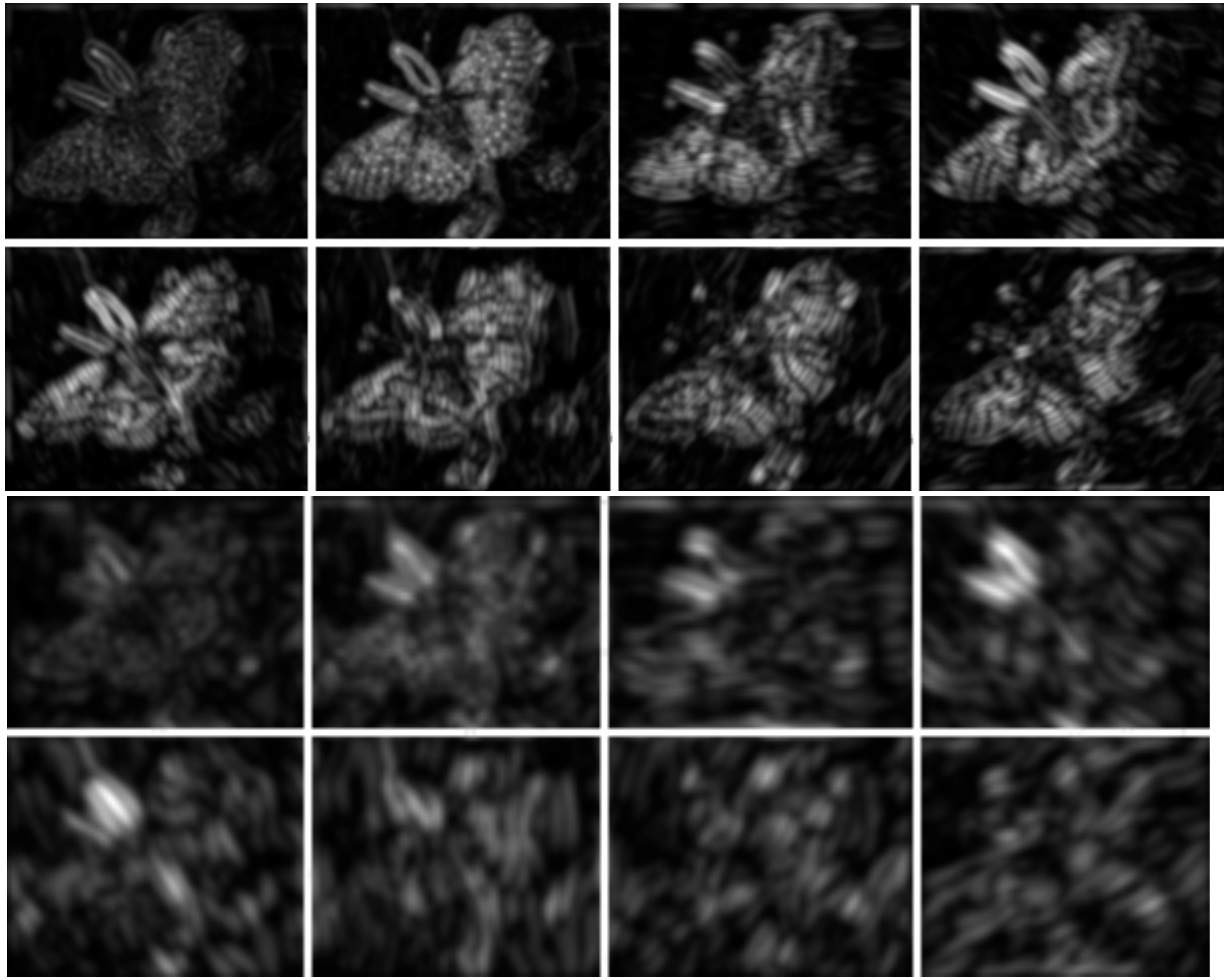
# **Spots** and Bars (Coarse Scale)





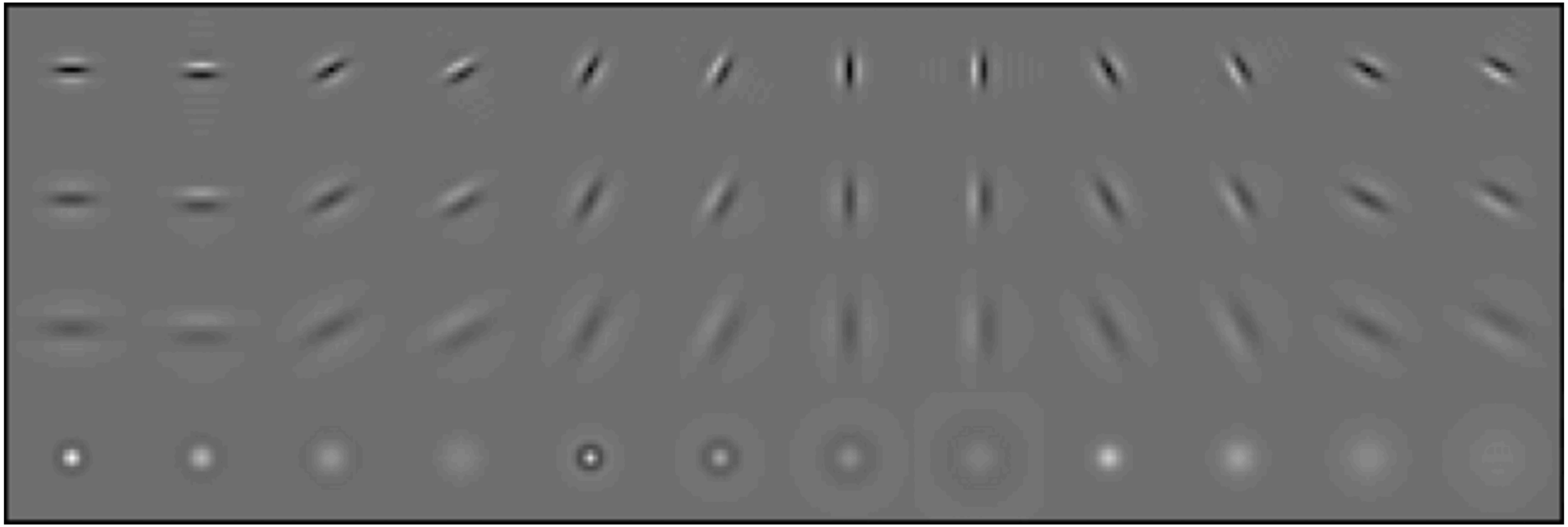
Forsyth & Ponce (1st ed.) Figures 9.3 and 9.5

# **Comparison** of Results



Forsyth & Ponce (1st ed.) Figures 9.4–9.5

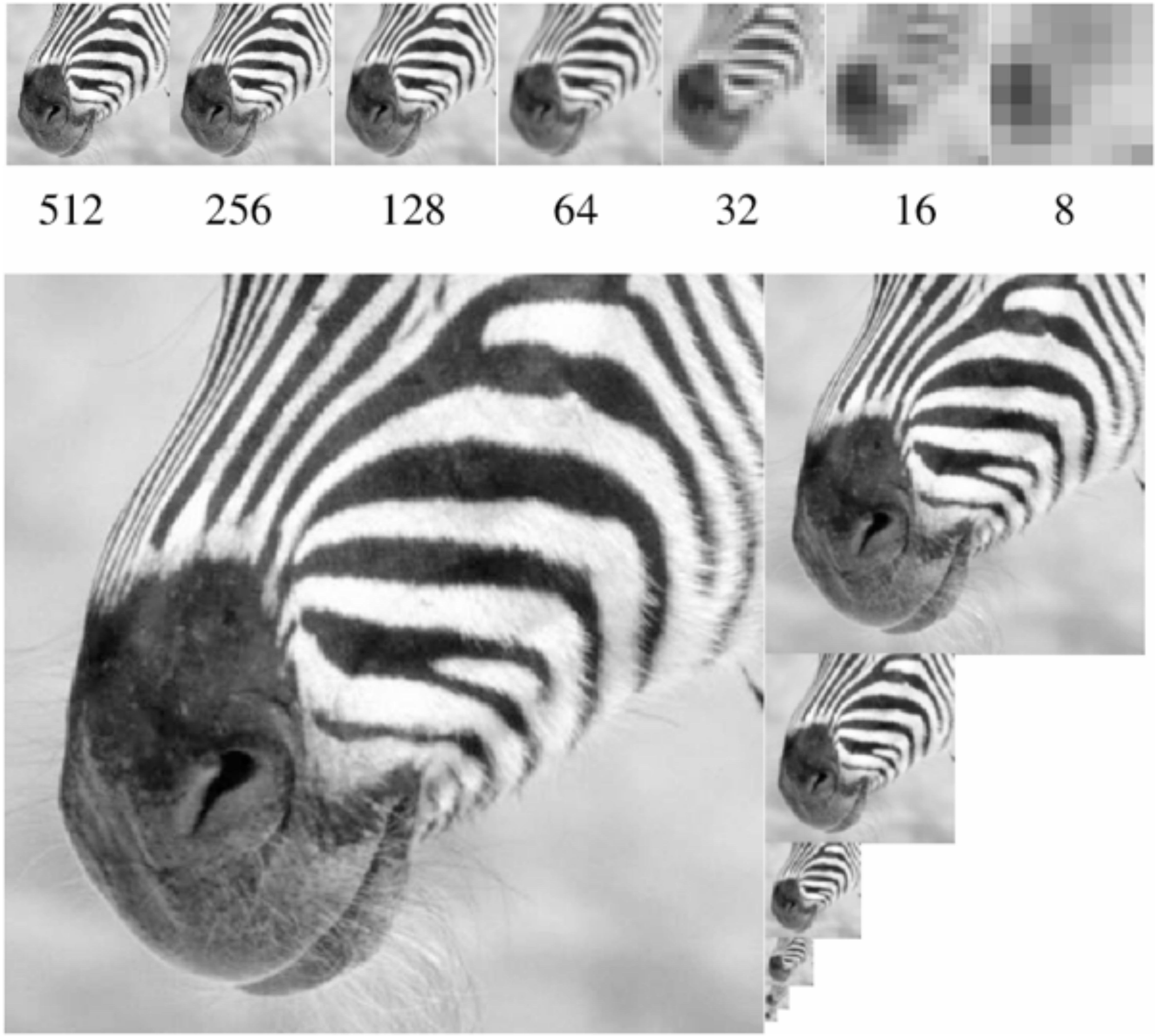
# Texture **Representation**



**Result**: 48-channel “image”

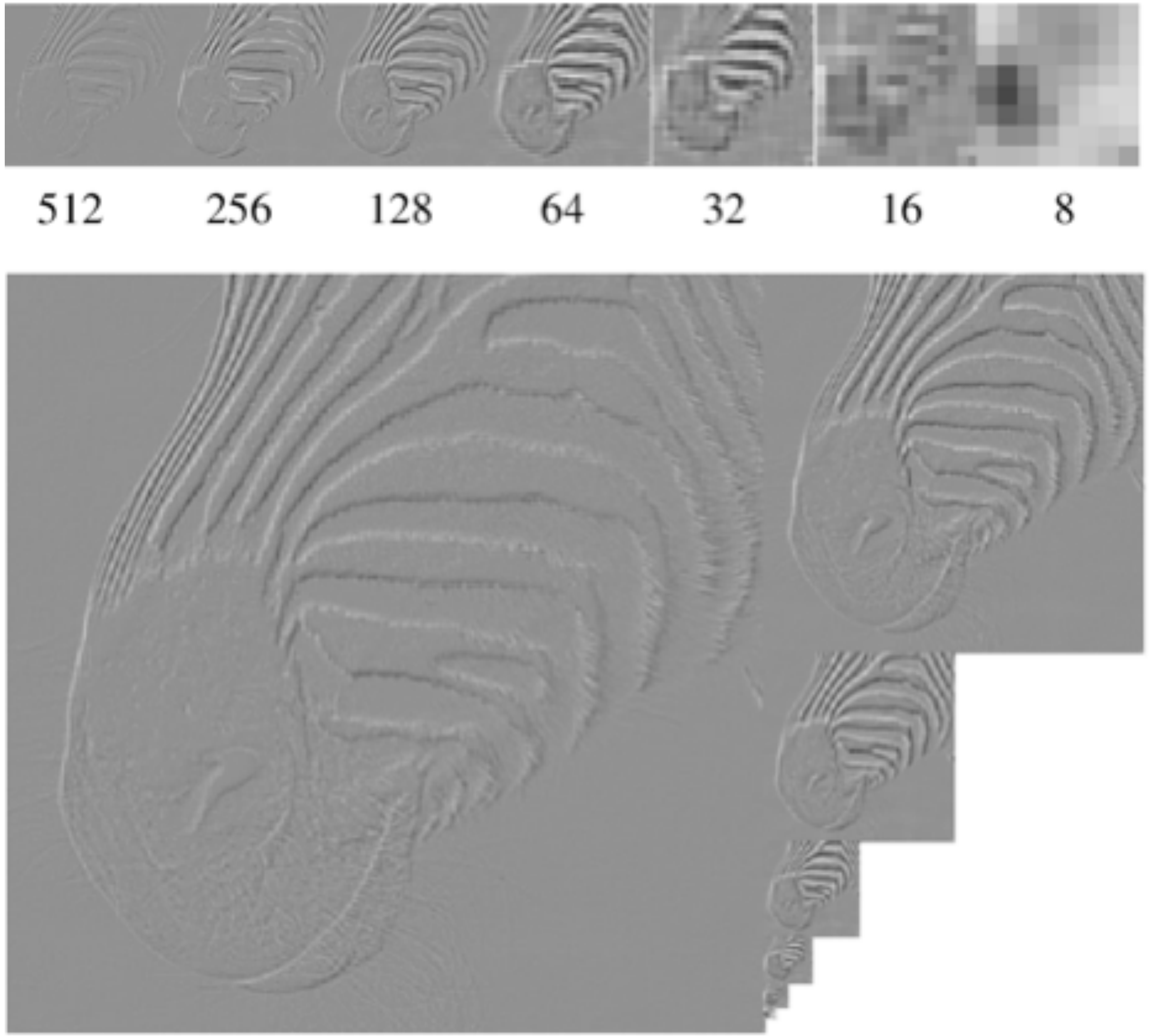
**Figure Credit**: Leung and Malik, 2001

# **Gaussian** Pyramid




Forsyth & Ponce (2nd ed.) Figure 4.17

# **Laplacian** Pyramid



# **Oriented** Pyramids

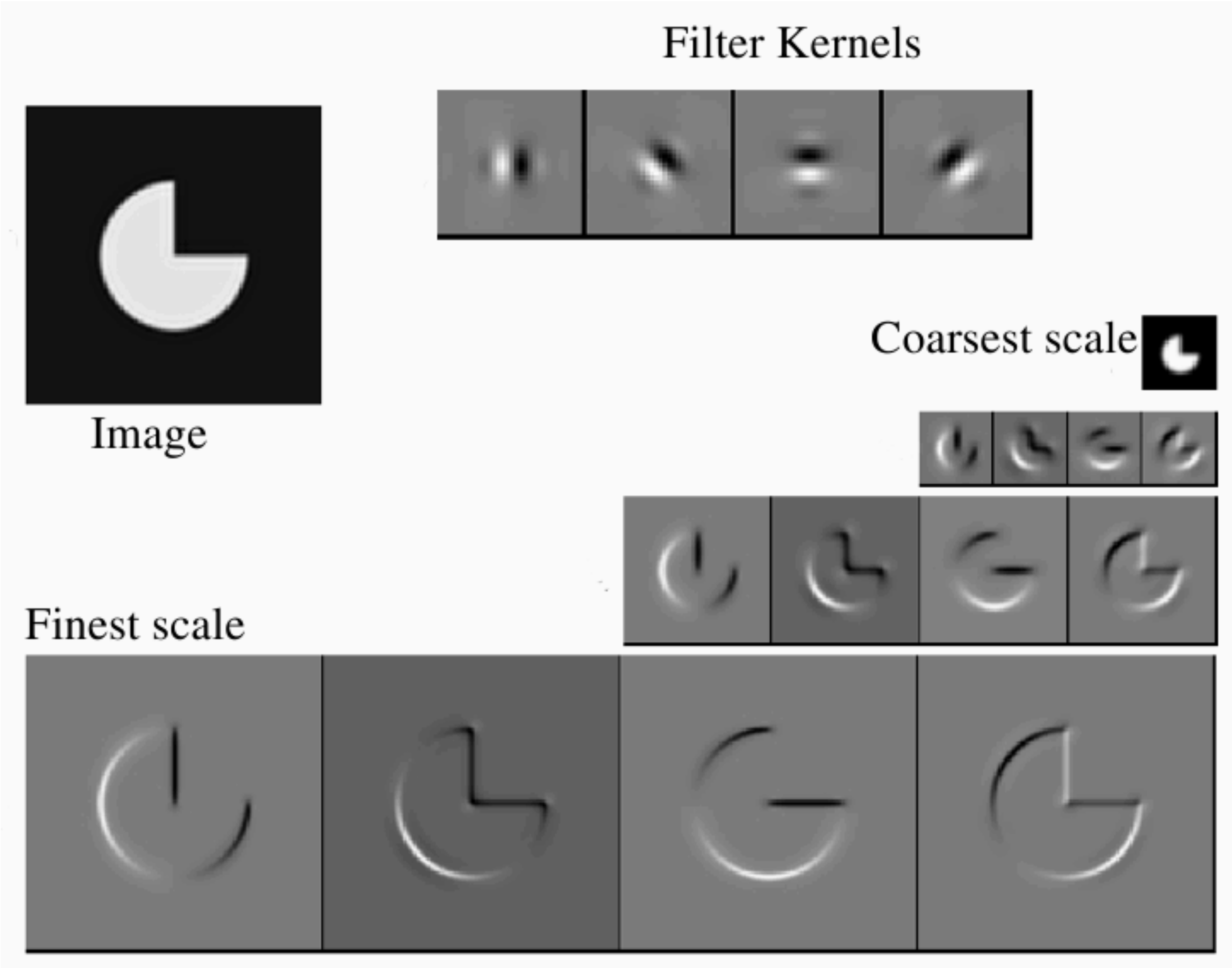
Laplacian pyramid is orientation independent

**Idea**: Apply an oriented filter at each layer

— represent image at a particular scale and orientation

— Aside: We do not study details in this course

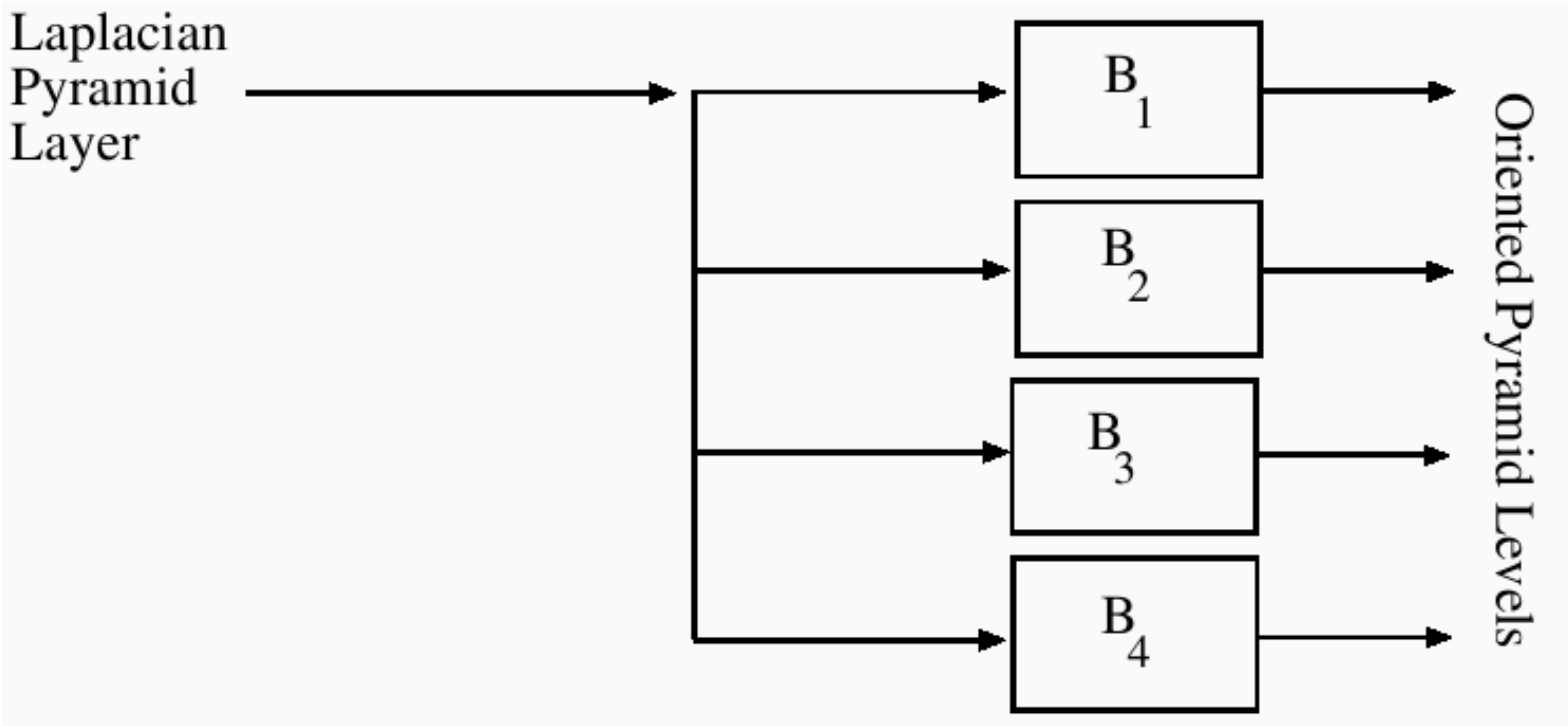
# **Oriented** Pyramids




Forsyth & Ponce (1st ed.) Figure 9.13

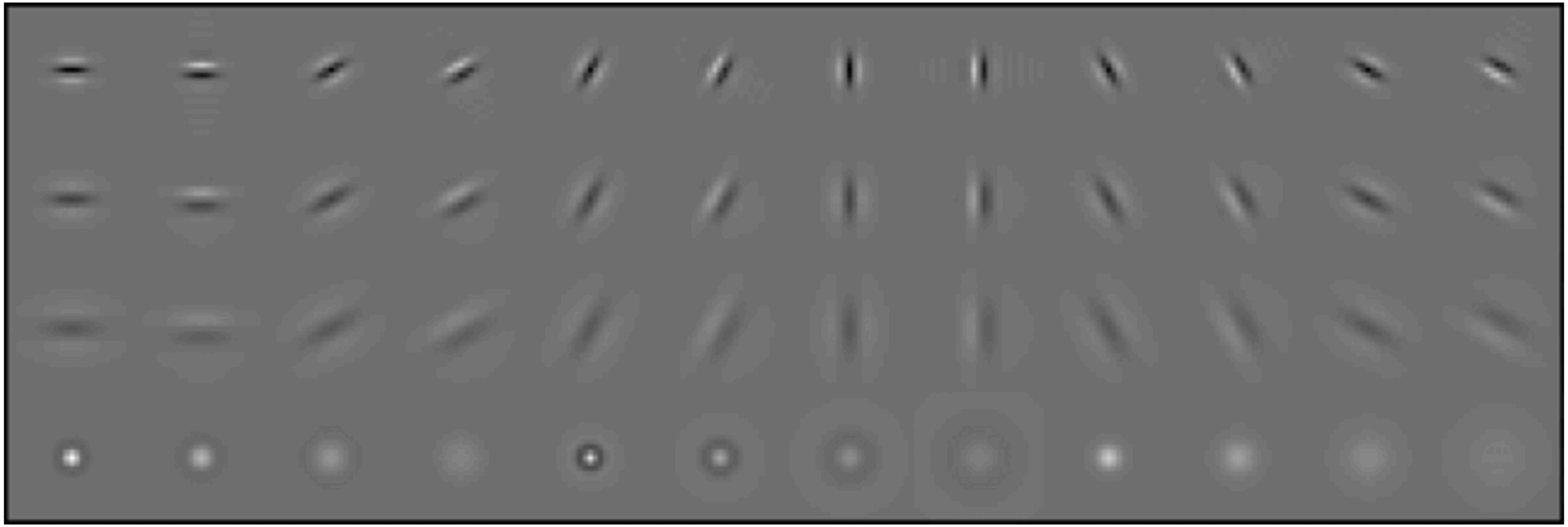
# **Oriented** Pyramids

Oriental Filters




Forsyth & Ponce (1st ed.) Figure 9.14

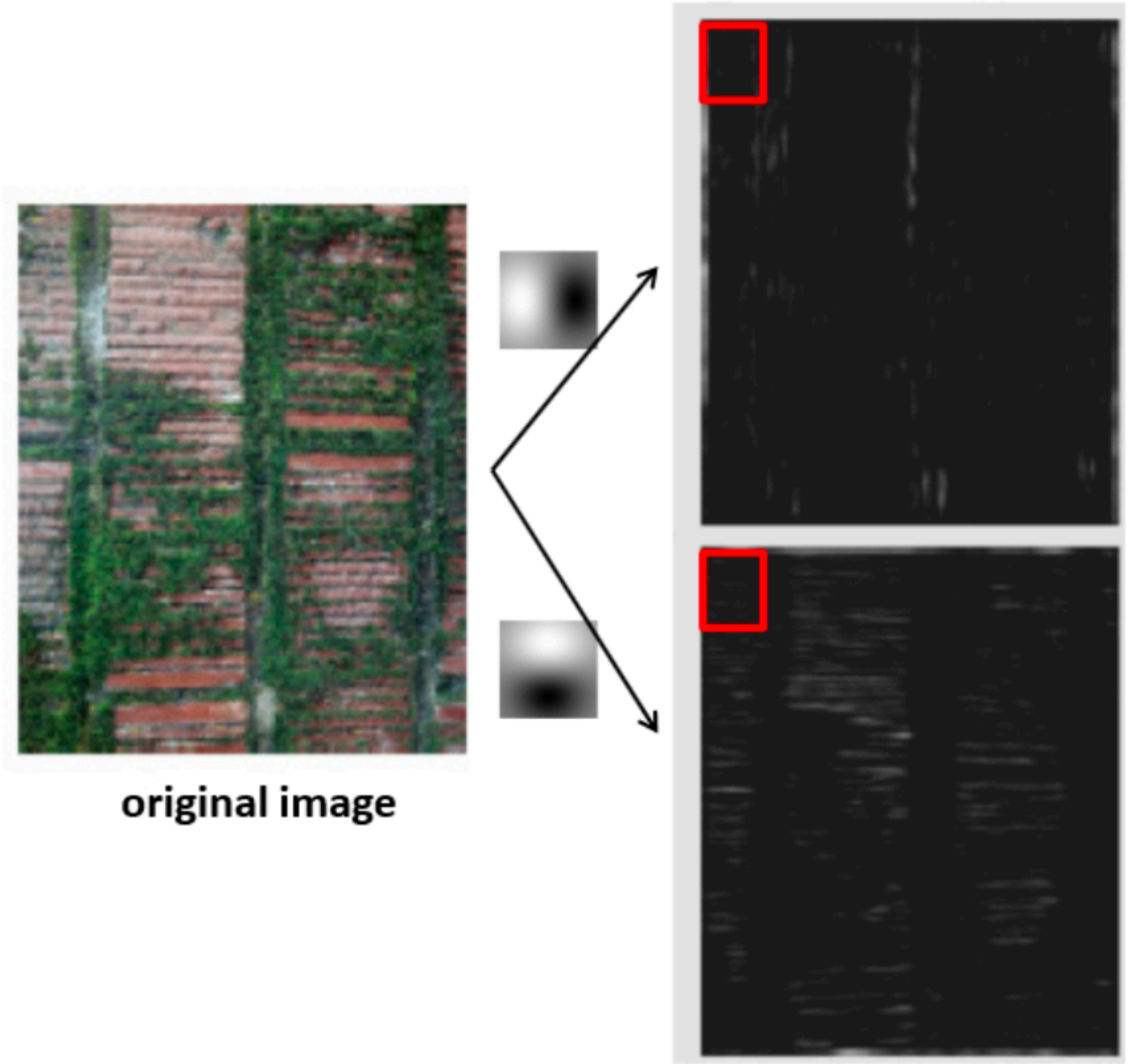
# Texture **Representation**



**Result**: 48-channel “image”

**Figure Credit**: Leung and Malik, 2001

# Texture **Representation**



original image

derivative filter responses, squared

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

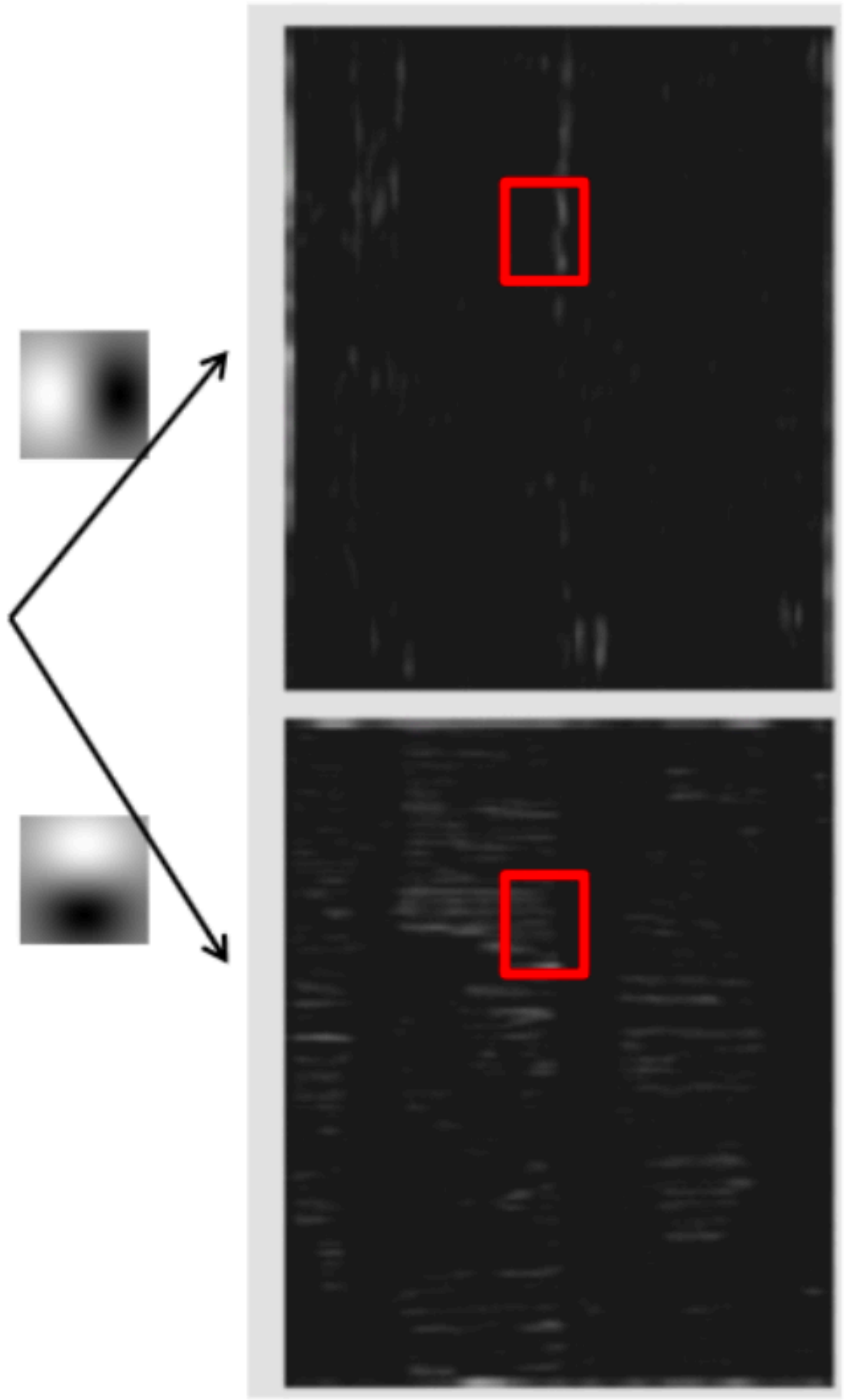
⋮

statistics to summarize patterns in small windows

**Slide Credit**: Trevor Darrell

# Texture **Representation**

original image



derivative filter responses, squared

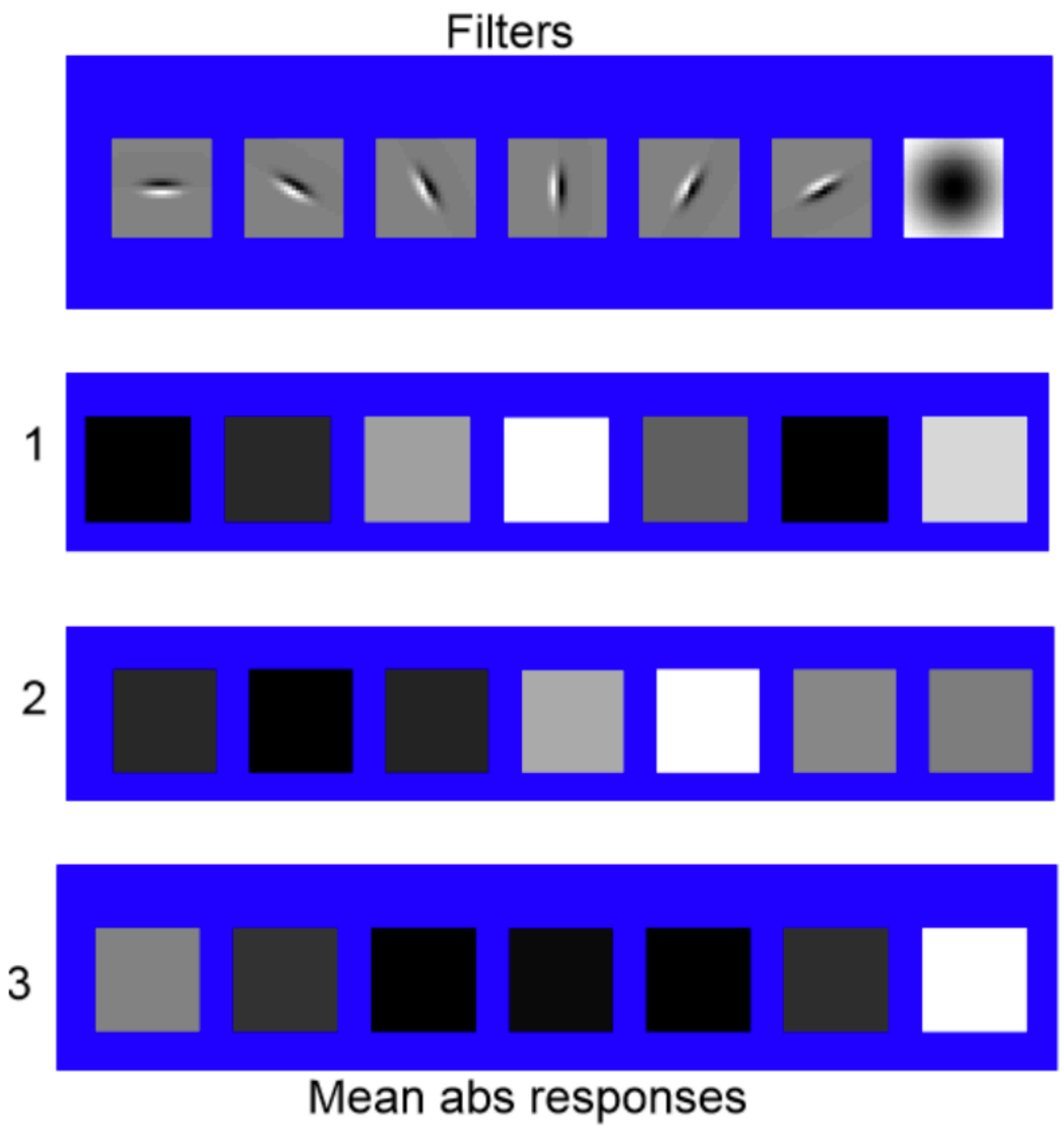
| | mean d/dx value | mean d/dy value |
|---|---|---|
| Win. #1 | 4 | 10 |
| Win.#2 | 18 | 7 |
| ⋮ | | |
| Win.#9 | 20 | 20 |
| | | |

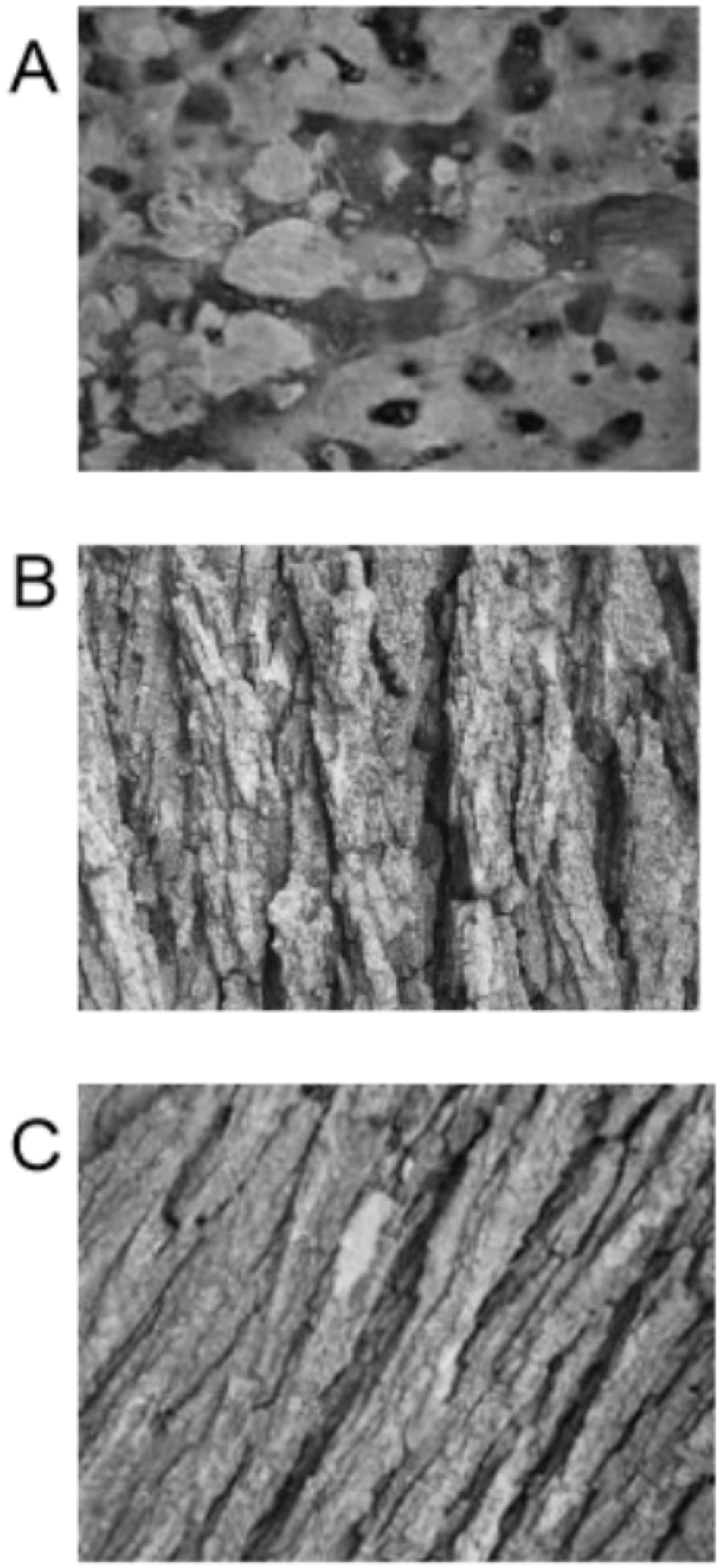
⋮

statistics to summarize patterns in small windows

**Slide Credit**: Trevor Darrell

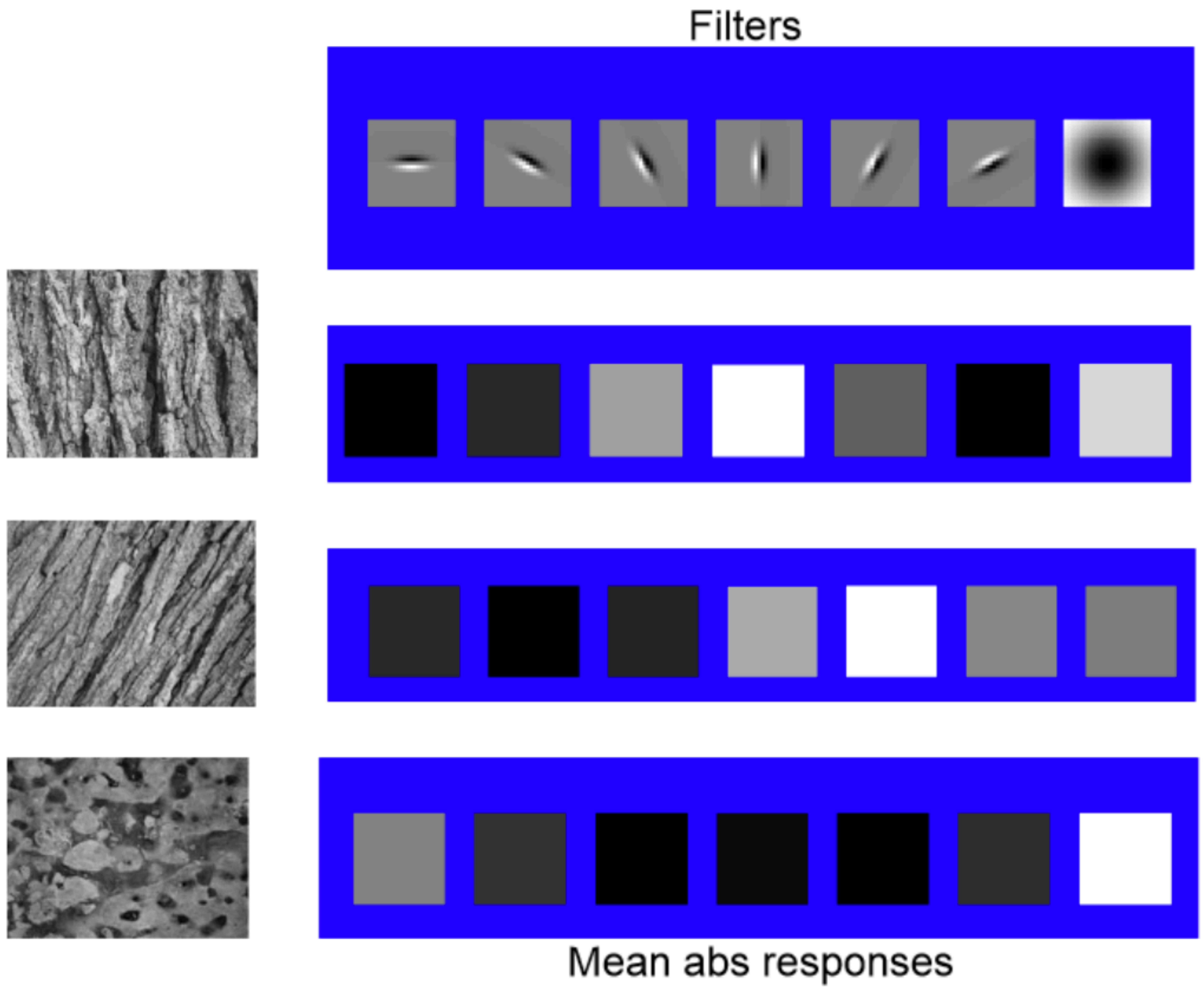
# A Short **Exercise**: Match the texture to the response



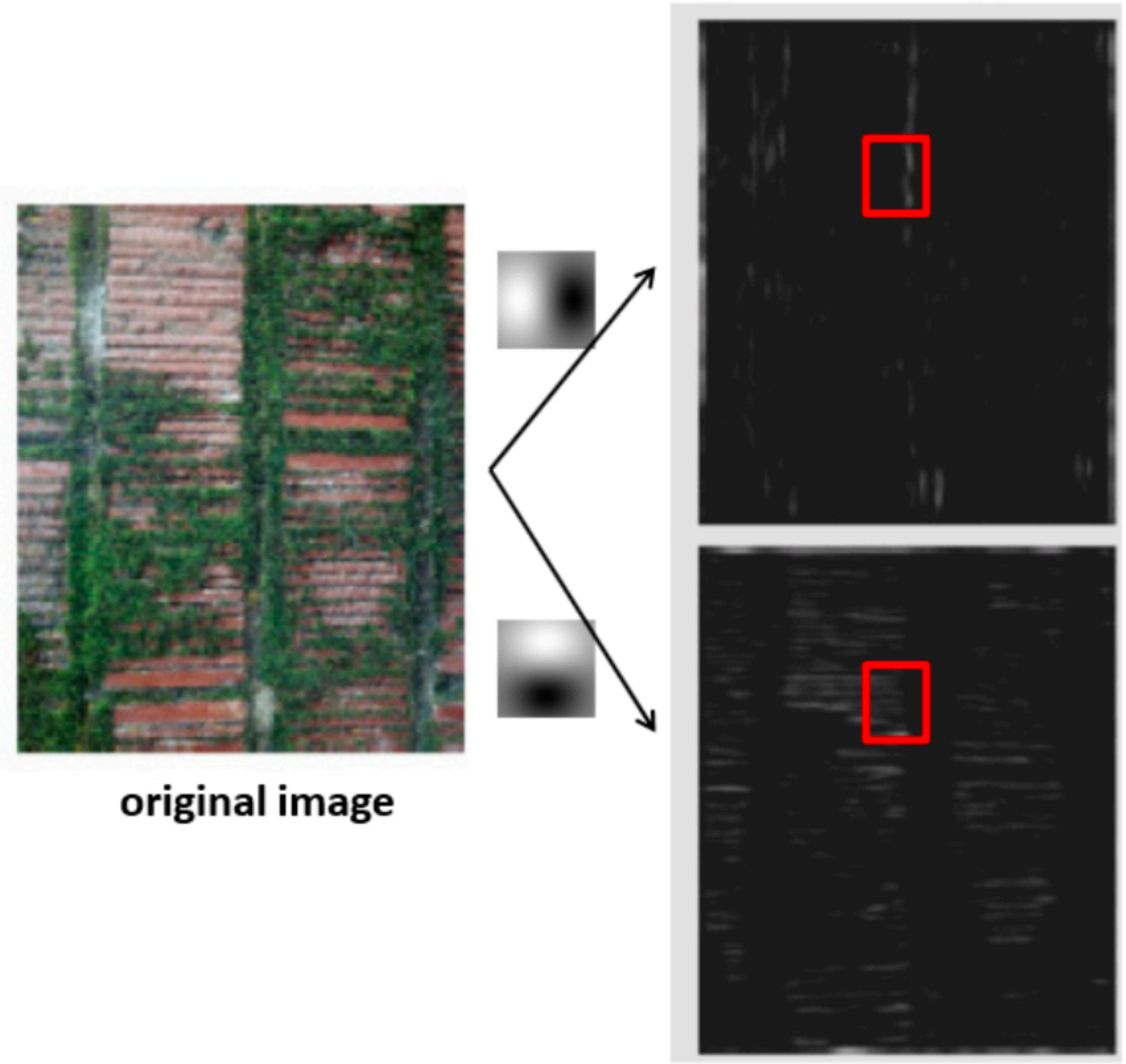

**Slide Credit**: James Hays

# A Short **Exercise**: Match the texture to the response




**Slide Credit**: James Hays

# Texture **Representation**



original image

derivative filter responses, squared

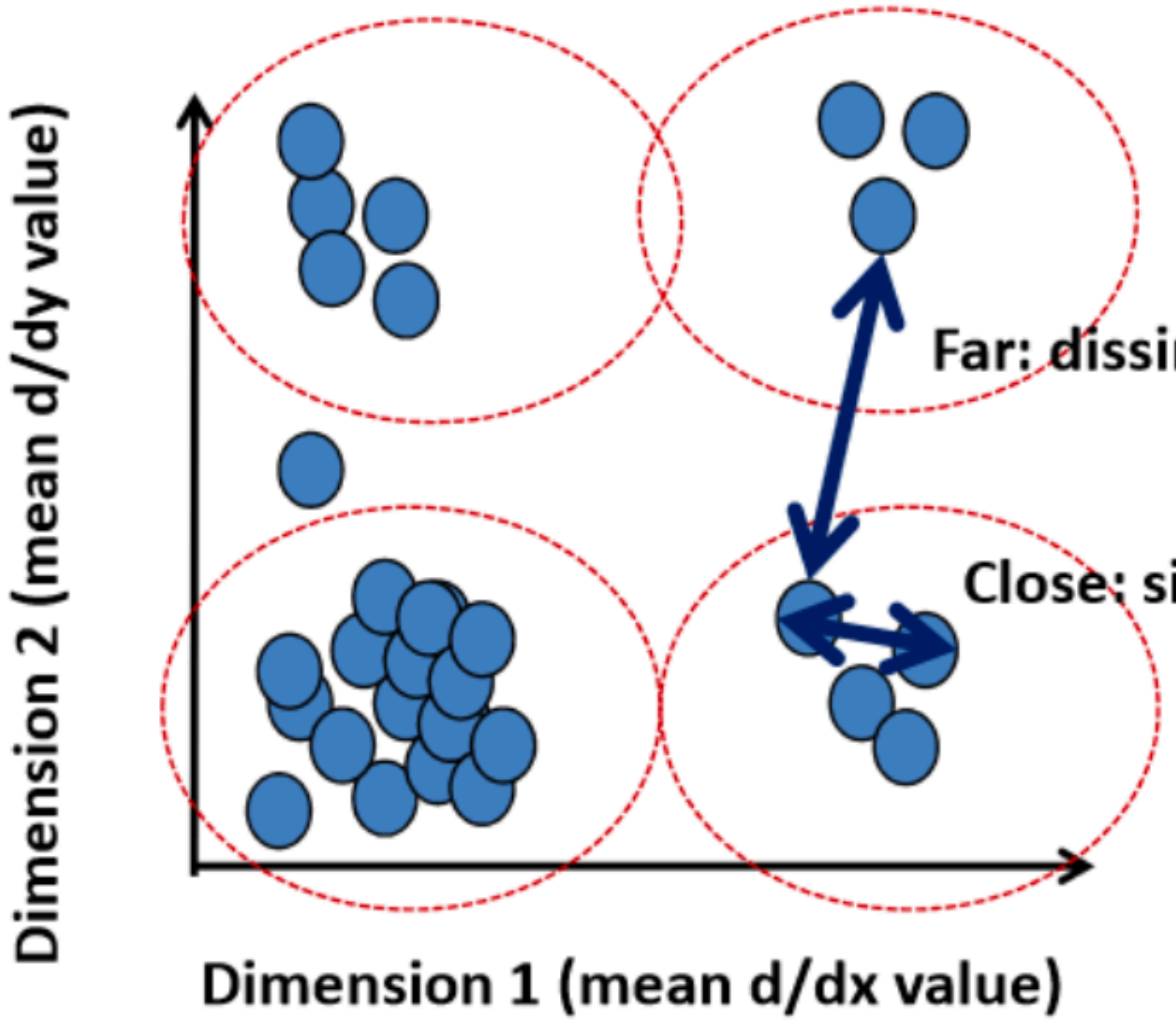
| | mean d/dx value | mean d/dy value |
|---|---|---|
| Win. #1 | 4 | 10 |
| Win.#2 | 18 | 7 |
| ⋮ | | |
| Win.#9 | 20 | 20 |
| | | |

⋮

statistics to summarize patterns in small windows

**Slide Credit**: Trevor Darrell

# Texture **Representation**



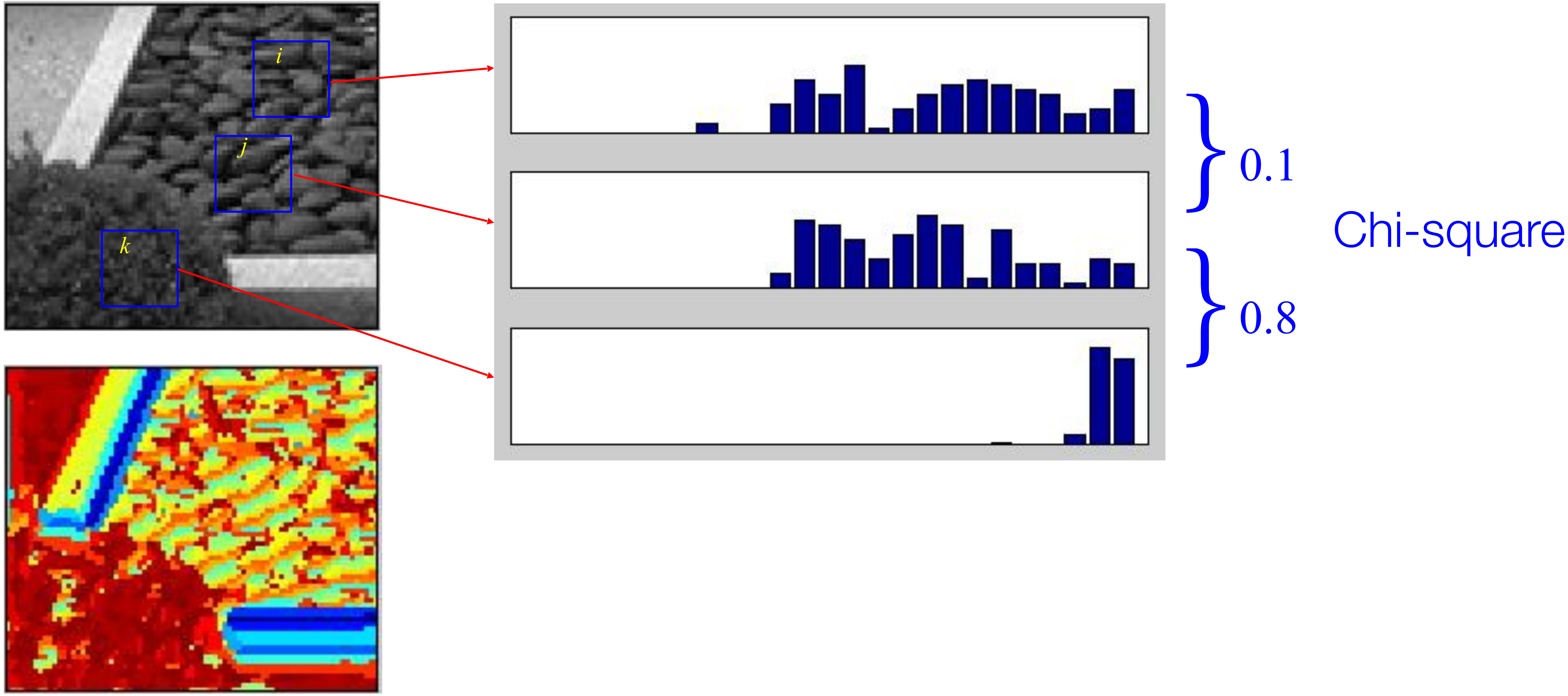

| | mean d/dx value | mean d/dy value |
|---|---|---|
| Win. #1 | 4 | 10 |
| Win.#2 | 18 | 7 |
| ⋮ | | |
| Win.#9 | 20 | 20 |
| | | |

⋮

**statistics to summarize patterns in small windows**

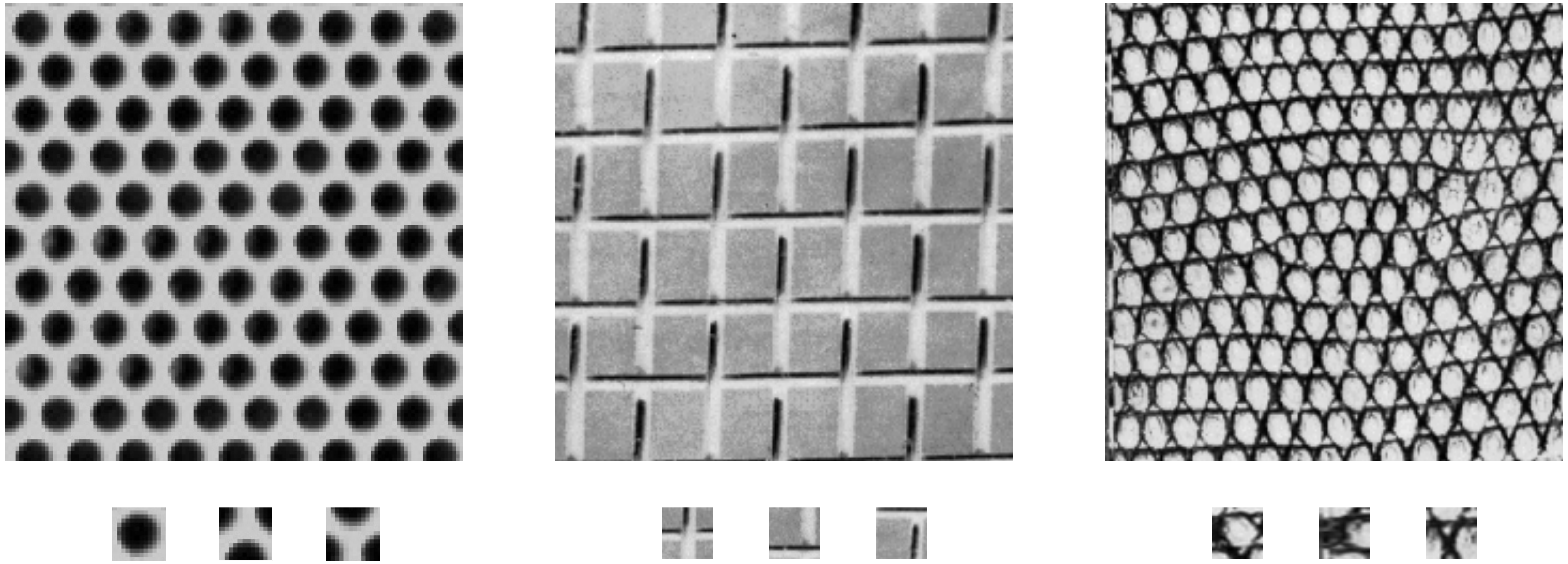
**Slide Credit**: Trevor Darrell

# Texture **Representation**



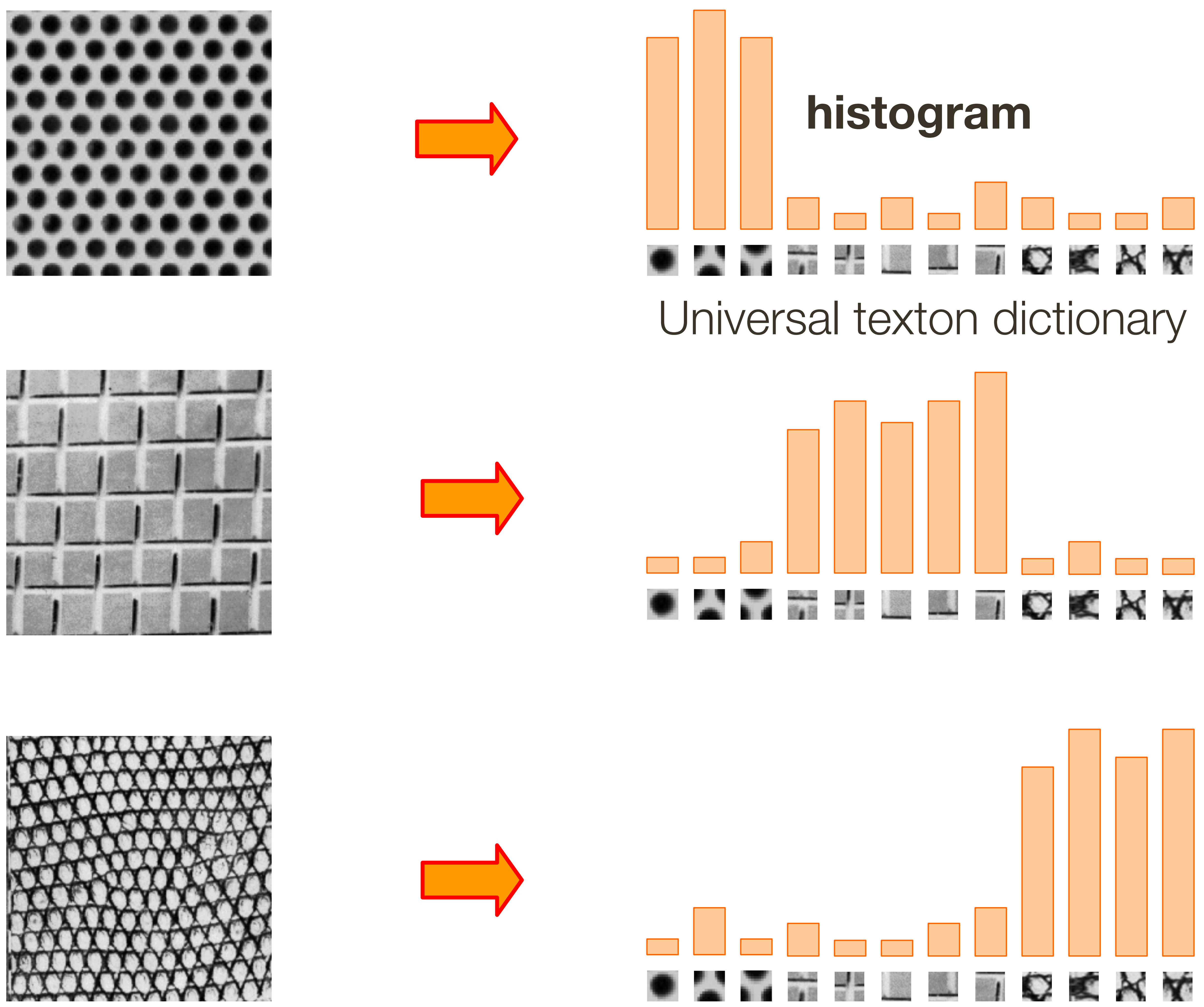
# **Texture** representation and recognition

- Texture is characterized by the repetition of basic elements or **textons**
- For stochastic textures, it is the **identity of the textons**, not their spatial arrangement, that matters

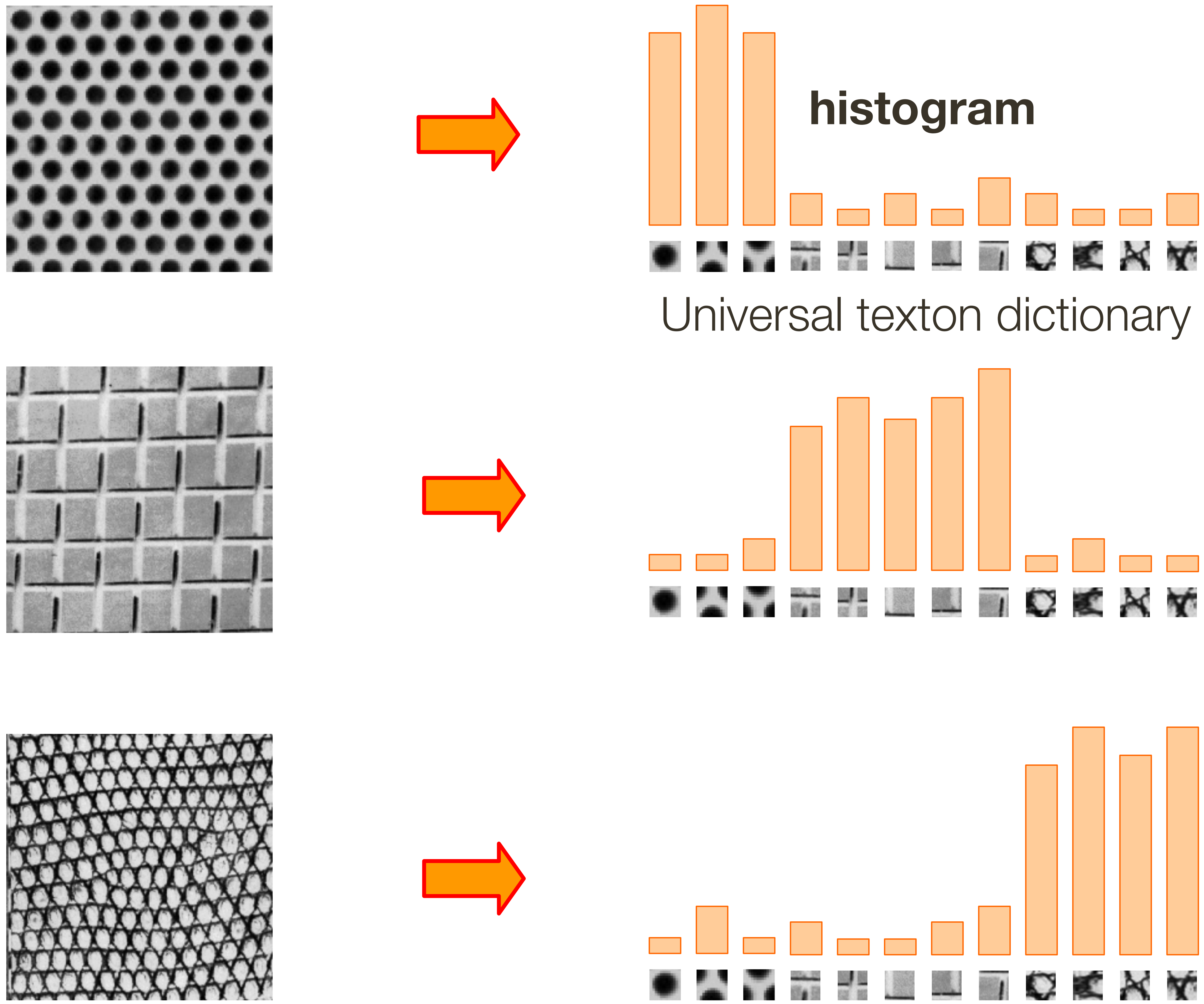


Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

# **Texture** representation and recognition



# **Texture** representation and recognition



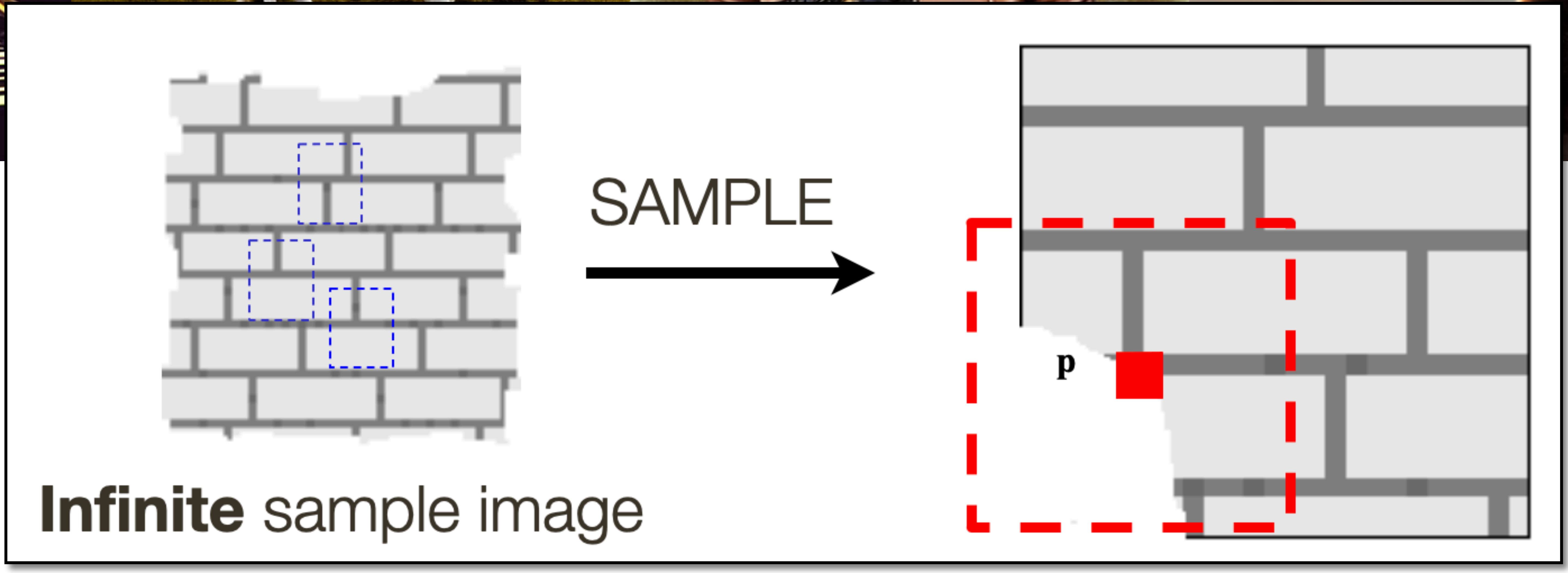

Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

# Relevant **modern** Computer Vision example

[Rombach et al., 2022] — https://github.com/CompVis/stable-diffusion

# Relevant **modern** Computer Vision example

[Rombach et al., 2022] —



# Summary

**Texture** representation is hard

— difficult to define, to analyze

— texture synthesis appears more tractable

Objective of texture **synthesis** is to generate new examples of a texture

— Efros and Leung: Draw samples directly from the texture to generate one pixel at a time. A “data-driven" approach.

Approaches to texture embed assumptions related to human perception