Image Filtering (Frequency Domain)

Dr. Sander Ali Khowaja,
Assistant Professor, Department of Telecommunication Engineering
Faculty of Engineering and Technology, University of Sindh, Pakistan
Senior Member, IEEE – Member, ACM
https://sander-ali.github.io

Computer Vision & Image Processing

Some AI related news

Augmented reality headset enables users to see hidden objects

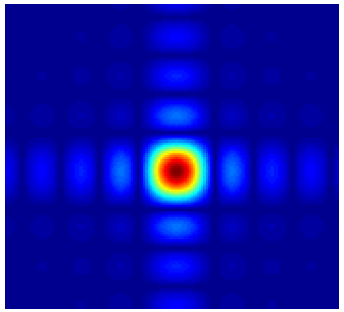
The device could help workers locate objects for fulfilling e-commerce orders or identify parts for assembling products.

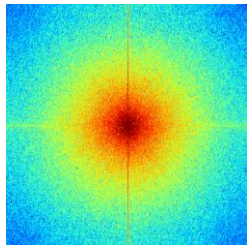
Watch Video

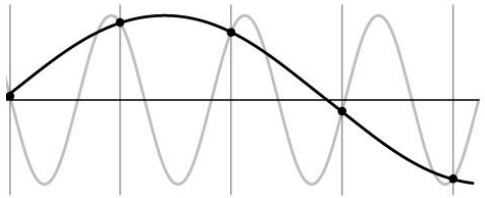
Adam Zewe | MIT News Office
February 27, 2023

Previous Class

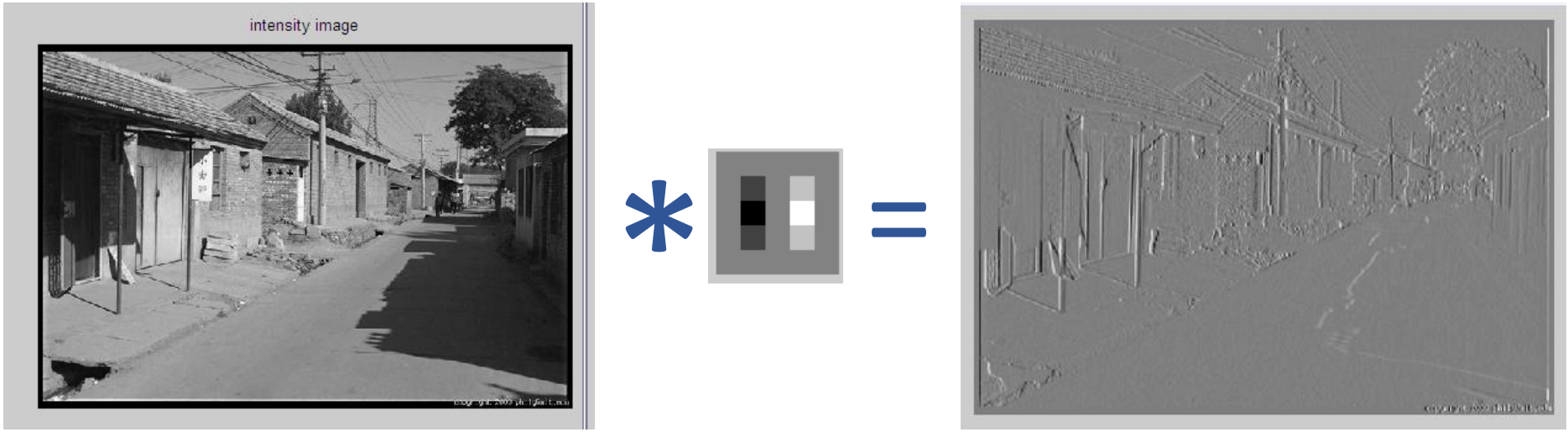
- Sometimes it makes sense to think of images and filtering in the frequency domain
  - Fourier analysis
- Can be faster to filter using FFT for large images (N logN vs. N^2 for auto-correlation)
- Images are mostly smooth
  - Basis for compression
- Remember to low-pass before sampling







Spatial domain



Frequency domain

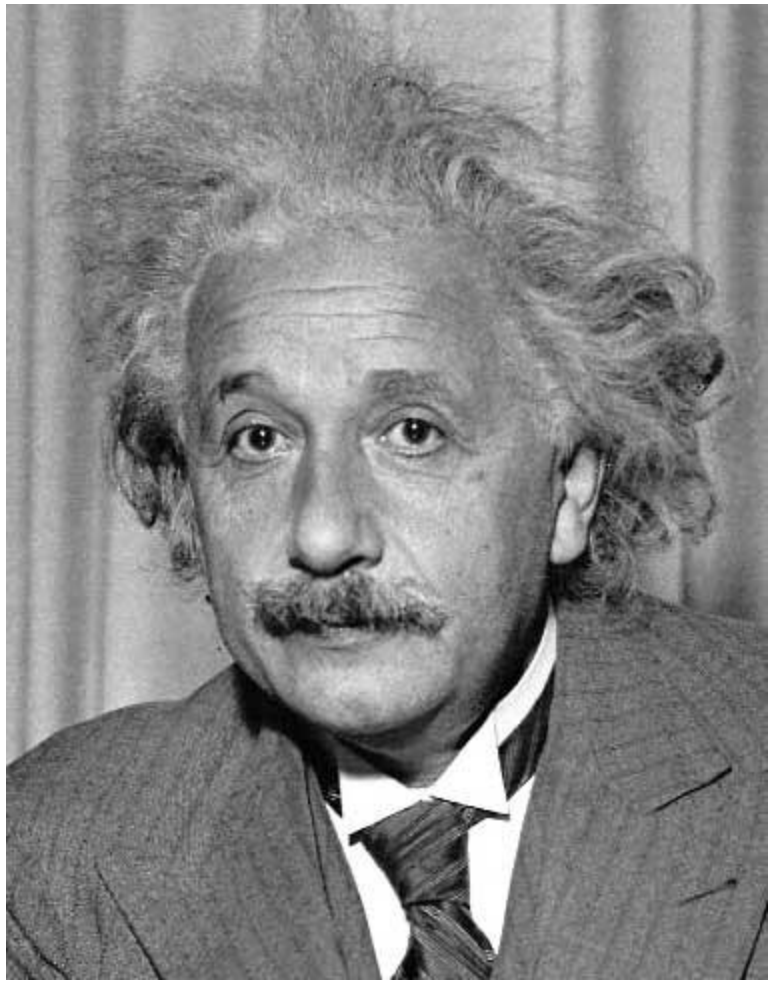
Today's Class

- Template matching
- Image Pyramids
- Compression

Template Matching

- Goal: find  in image

- Main challenge: What is a good similarity or distance measure between two patches? D(,)
  - Correlation
  - Zero-mean correlation
  - Sum Square Difference
  - Normalized Cross Correlation

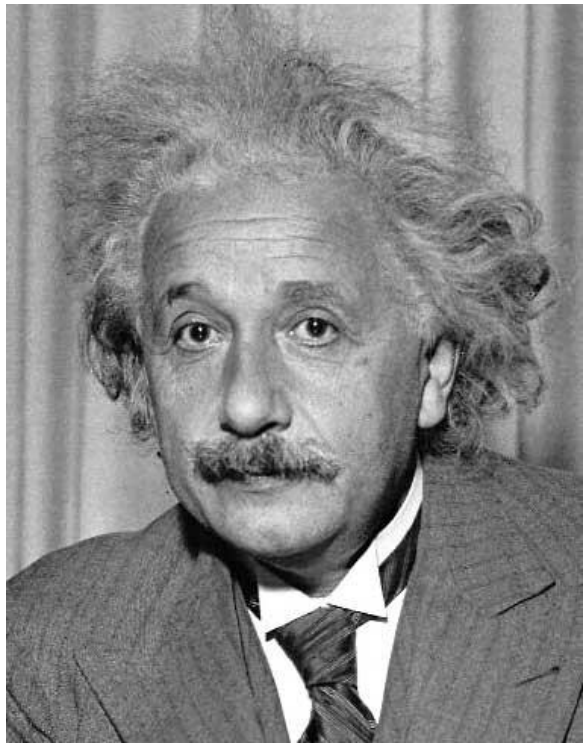


Matching with Filters

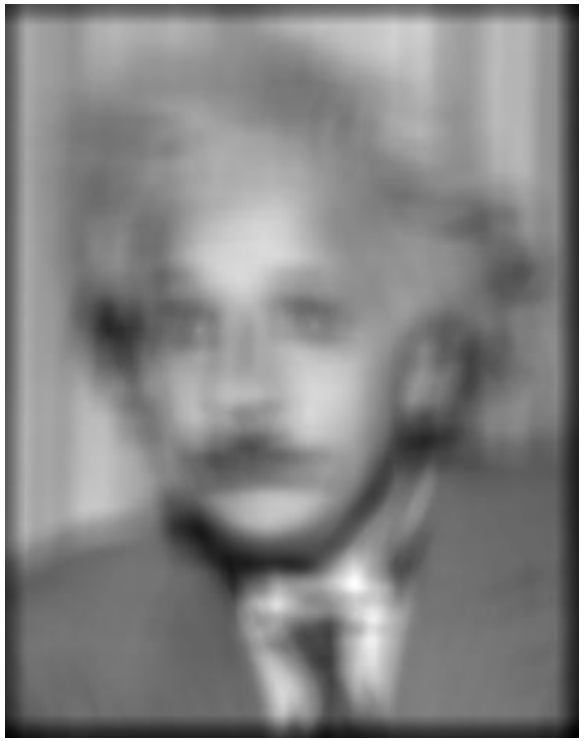
- Goal: find  in image
- Method 0: filter the image with eye patch

$$h[m,n] = \sum_{k,l} g[k,l]\ f[m+k, n+l]$$

g = filter f = image



Input



Filtered Image

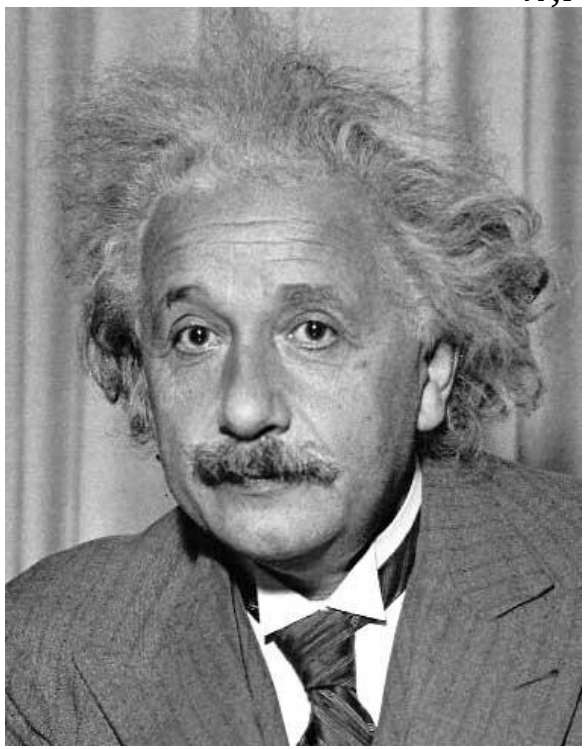
What went wrong?

Matching with Filters

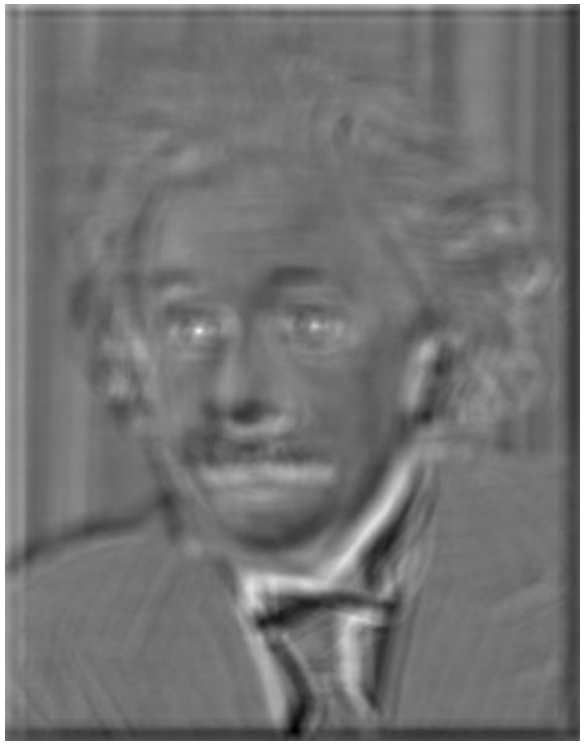
- Goal: find  in image
- Method 1: filter the image with zero-mean eye

$$h[m,n] = \sum_{k,l} (g[k,l] - \bar{g})\,(f[m+k, n+l])$$

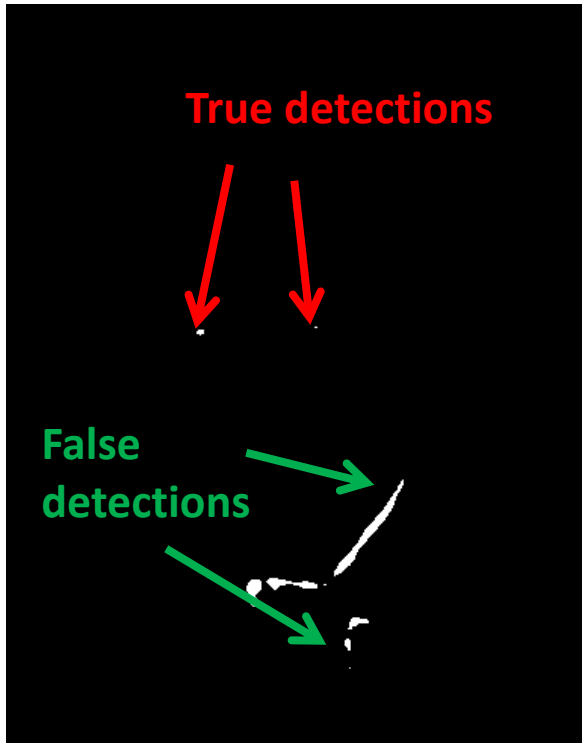
mean of template g



Input



Filtered Image (scaled)

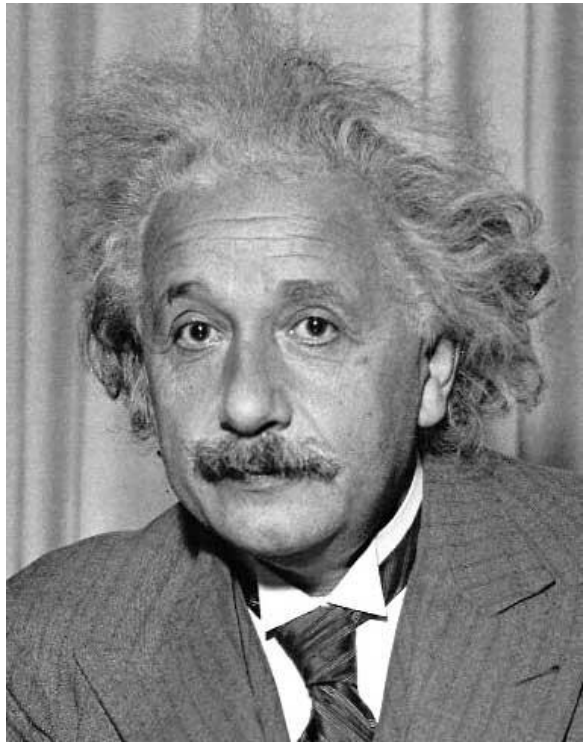


Thresholded Image

Matching with Filters

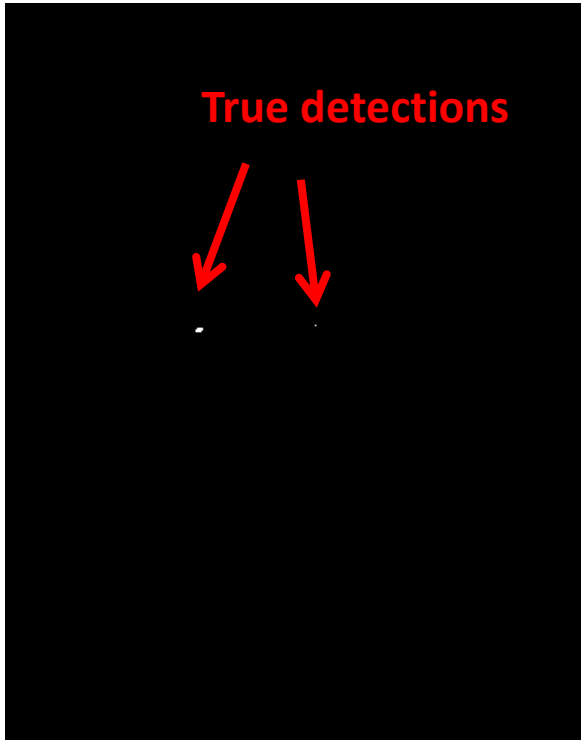
- Goal: find  in image
- Method 2: Sum of squared differences (SSD)

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k,n+l])^2$$



Input

1- sqrt(SSD)



Thresholded Image

Matching with Filters

Can SSD be implemented with linear filters?

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k, n+l])^2$$

$$h[m,n] = \sum_{k,l} (g[k,l])^2 - 2\sum_{k,l} g[k,l] f[m+k, n+l] + \sum_{k,l} (f[m+k, n+l])^2$$

Constant

Filtering with g

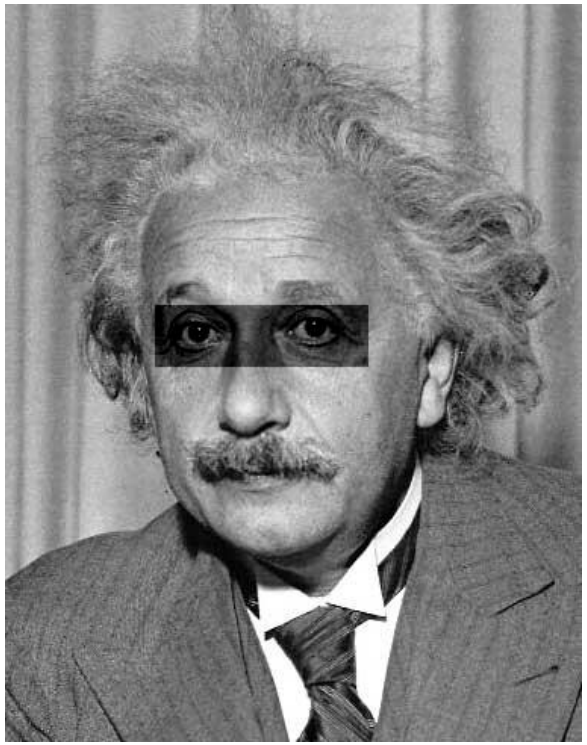
Filtering with box filter

Matching with SSD

- Goal: find  in image
- Method 2: SSD

What's the potential downside of SSD?

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k,n+l])^2$$



Input

1- sqrt(SSD)

Template Matching

Maximize:

$$R_{tf}[i,j] = \sum_m \sum_n f[m,n]t[m-i,n-j] = t \otimes f$$

(Cross-Correlation)

How do we locate the template in the image?

Minimize:

$$E[i,j] = \sum_m \sum_n (f[m,n] - t[m-i,n-j])^2$$

Sum of Squared Differences (SSD)

$$E[i,j] = \sum_m \sum_n (f^2[m,n] + t^2[m-i,n-j] - 2f[m,n]t[m-i,n-j])$$

Maximize

Convolution vs Correlation (Template Matching)

Convolution:

$$g[i,j] = \sum_m \sum_n f[m,n]t[i-m, j-n] = t * f$$

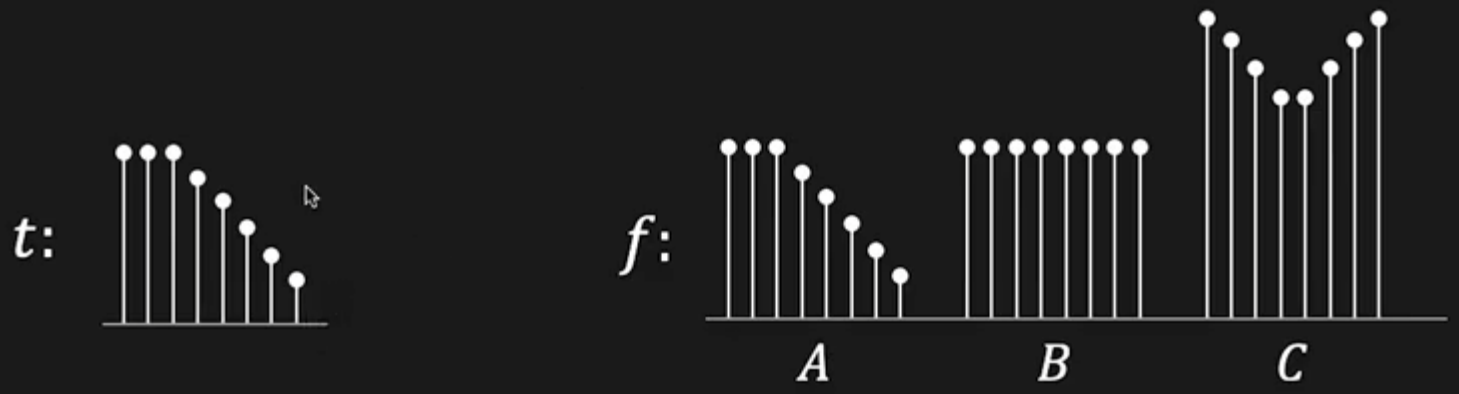
Correlation:

$$R_{tf}[i,j] = \sum_m \sum_n f[m,n]t[m-i, n-j] = t \otimes f$$

No Flipping in Correlation

Problem with Cross-Correlation

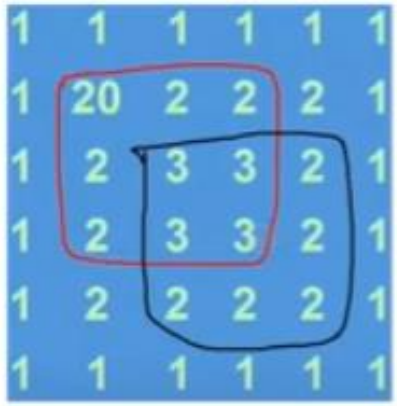
$$R_{tf}[i,j] = \sum_m \sum_n f[m,n]t[m-i,n-j] = t \otimes f$$



$$R_{tf}(C) > R_{tf}(B) > R_{tf}(A)$$

We need $R_{tf}(A)$ to be the maximum!

Cross Correlation Example



47	54	56	20	18	12
54	87	94	40	34	21
56	94	107	54	43	24
20	40	54	56	44	24
18	34	43	44	37	22
12	21	24	24	22	15

```
clc
clear all
close all
warning off
a=[1 1 1 1 1 1;
   1 20 2 2 2 1;
   1 2 3 3 2 1;
   1 2 3 3 2 1;
   1 2 2 2 2 1;
   1 1 1 1 1 1];
w1=[3 3 2;
    3 3 2;
    2 2 2];
newA = zeros(size(a)+2);
newA(2:end-1,2:end-1)=a;
```

```
a=newA;
[row col]=size(a);
a1=zeros(row,col);
for x=2:1:row-1
    for y=2:1:col-1
        a1(x,y)=w1(1)*a(x-1,y-1)+w1(2)*a(x-1,y)+w1(3)*...
            a(x-1,y+1)+w1(4)*a(x,y-1)+w1(5)*a(x,y)+w1(6)*...
            a(x,y+1)+w1(7)*a(x+1,y-1)+w1(8)*a(x+1,y)+w1(9)*...
            a(x+1,y+1);
    end
end
aa1=a1(2:end-1,2:end-1);
```

Matching with Filters

- Goal: find  in image
- Method 3: Normalized cross-correlation

mean template

mean image patch

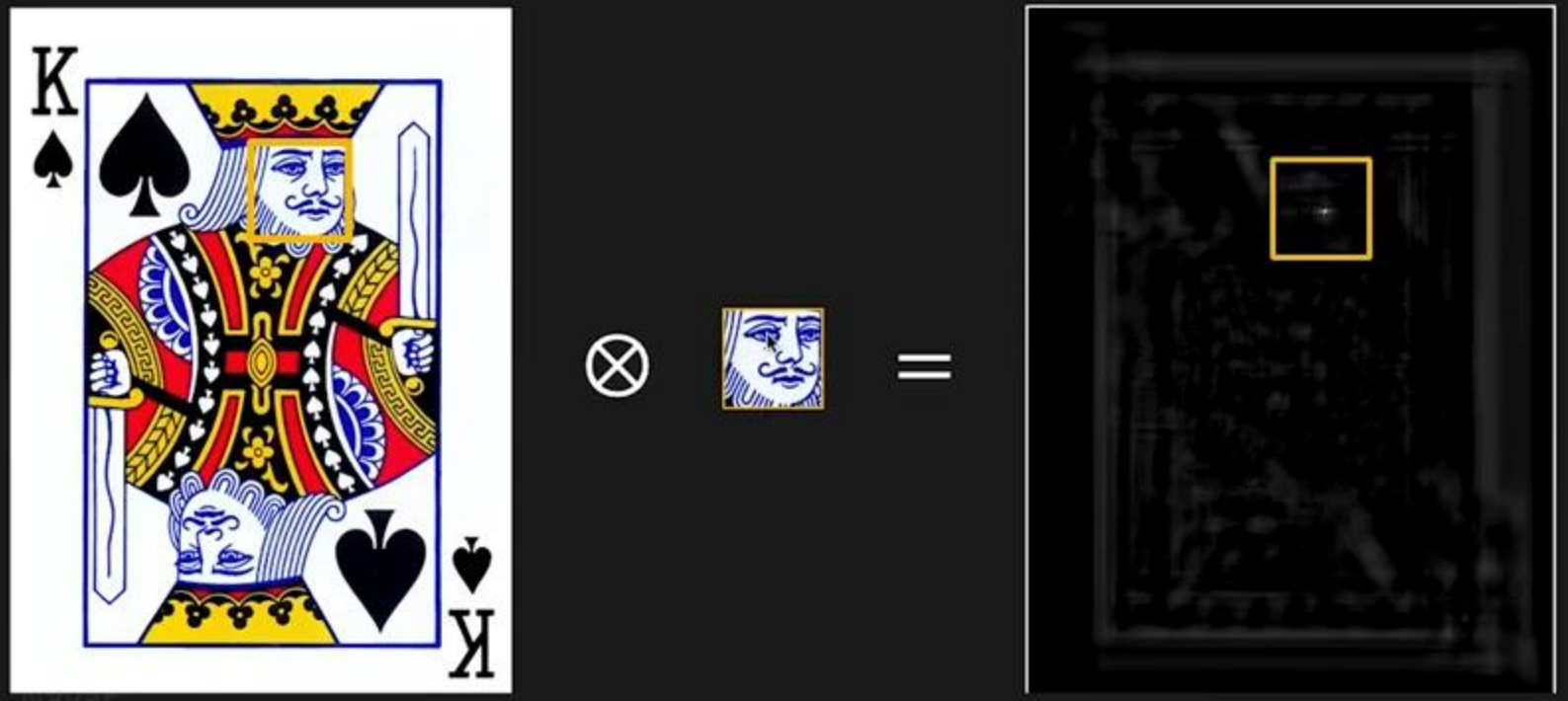
$$h[m,n] = \frac{\sum_{k,l}(g[k,l]-\bar{g})(f[m+k,n+l]-\bar{f}_{m,n})}{\left(\sum_{k,l}(g[k,l]-\bar{g})^2 \sum_{k,l}(f[m+k,n+l]-\bar{f}_{m,n})^2\right)^{0.5}}$$

Matlab: `normxcorr2(template, im)`

Normalized Cross Correlation

Account for energy differences

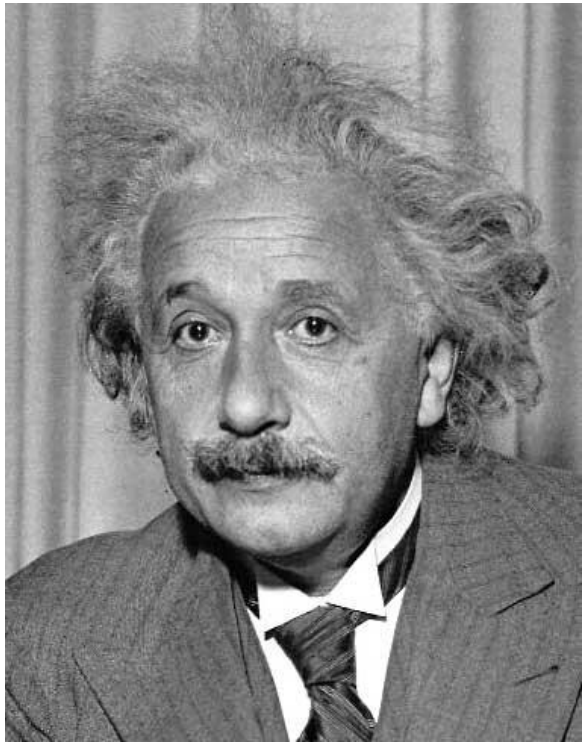
$$N_{tf}[i,j] = \frac{\sum_m \sum_n f[m,n]t[m-i,n-j]}{\sqrt{\sum_m \sum_n f^2[m,n]} \sqrt{\sum_m \sum_n t^2[m-i,n-j]}}$$



Insensitive to changes in Brightness

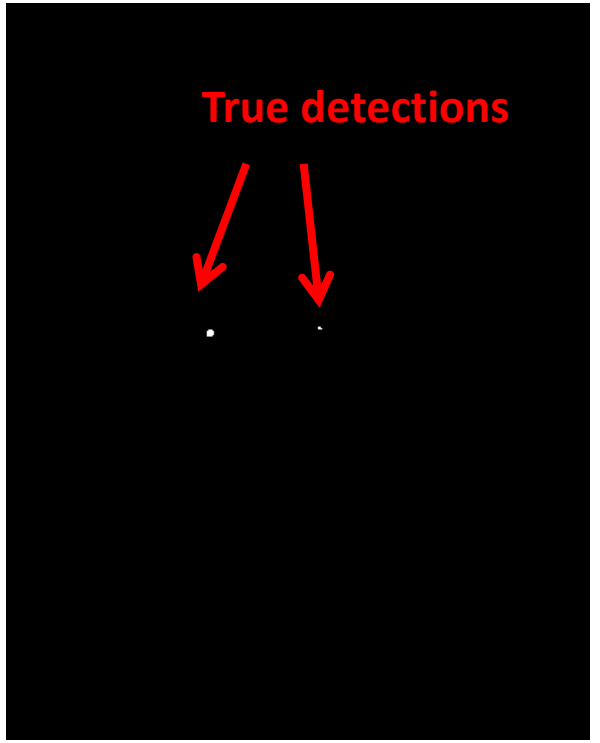
Matching with Filters

- Goal: find  in image
- Method 3: Normalized cross-correlation



Input

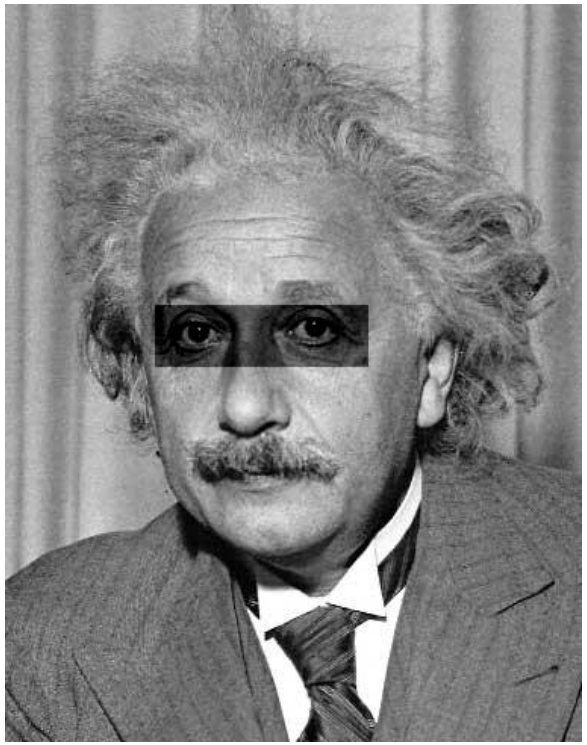
Normalized X-Correlation



Thresholded Image

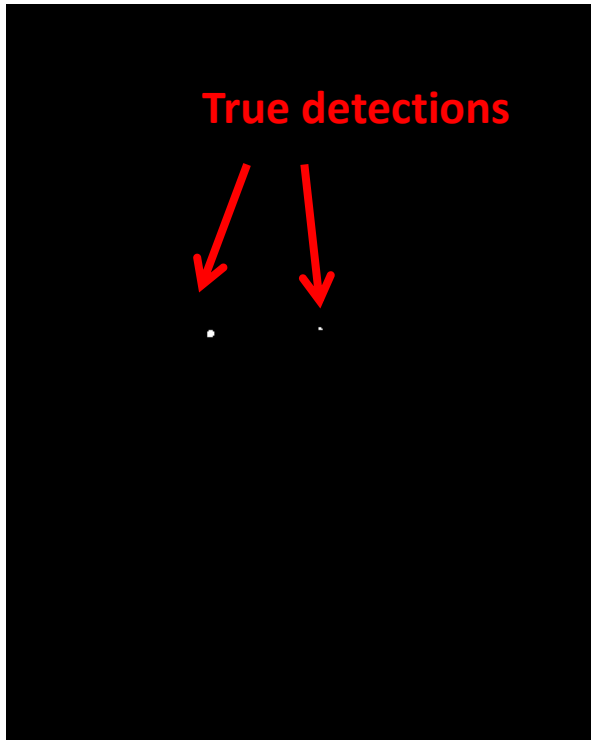
Matching with Filters

- Goal: find  in image
- Method 3: Normalized cross-correlation



Input

Normalized X-Correlation



Thresholded Image

What is the best method to use then?

A: Depends

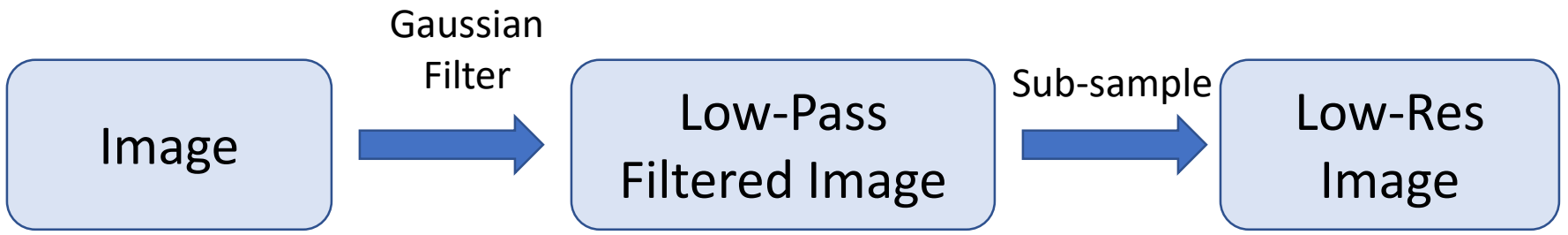
- Zero-mean filter: fastest but not a great matcher
- SSD: next fastest, sensitive to overall intensity
- Normalized cross-correlation: slowest, invariant to local average intensity and contrast

What about scaling

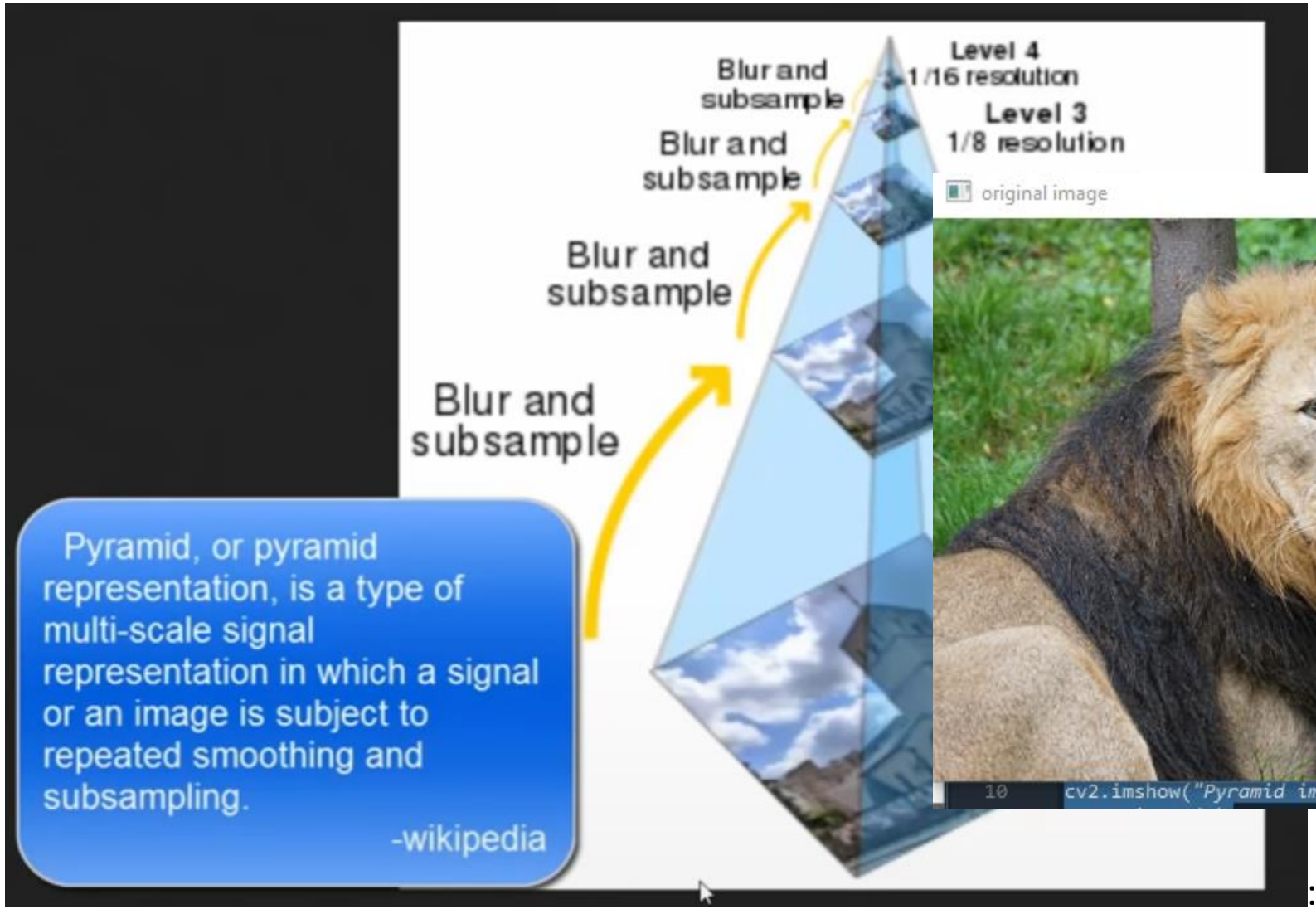
Q: What if we want to find larger or smaller eyes?

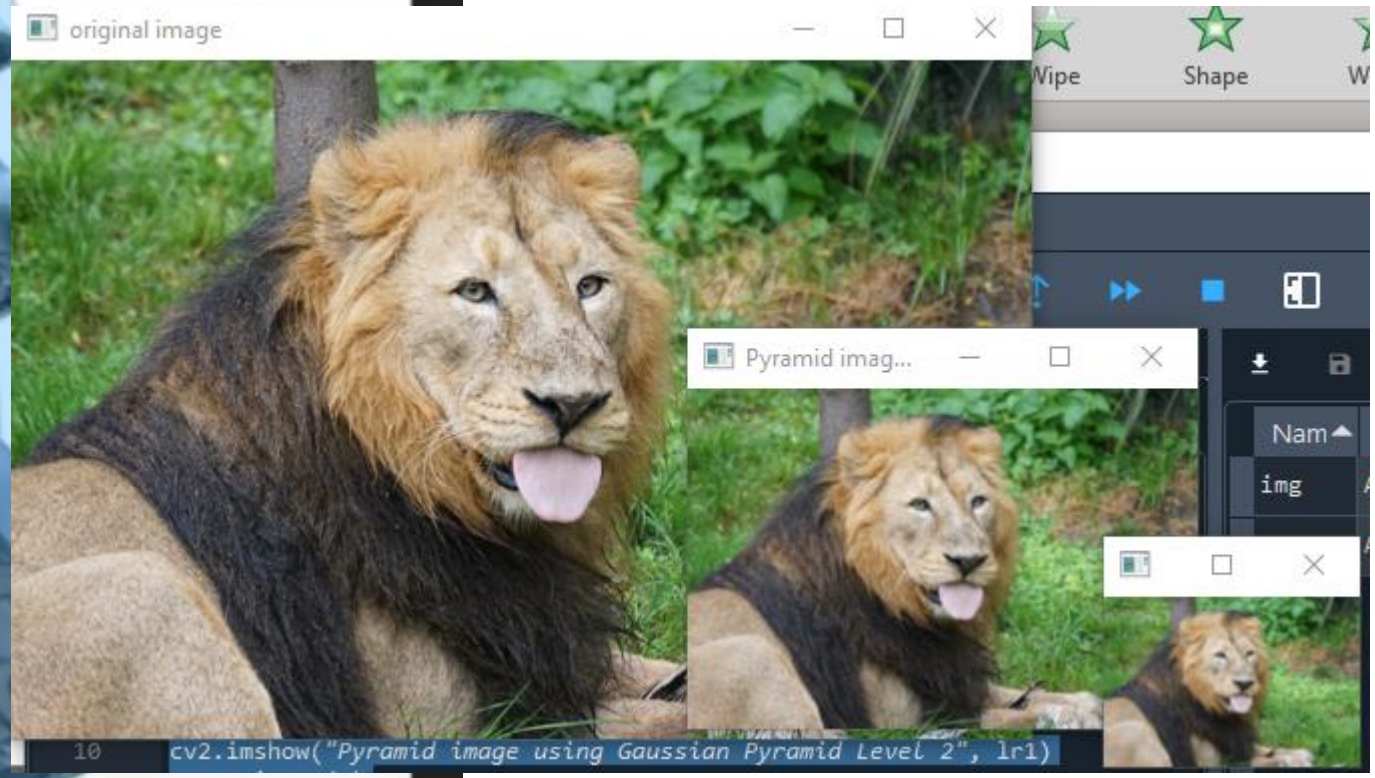
A: Image Pyramid

Review of Sampling



Gaussian Pyramid



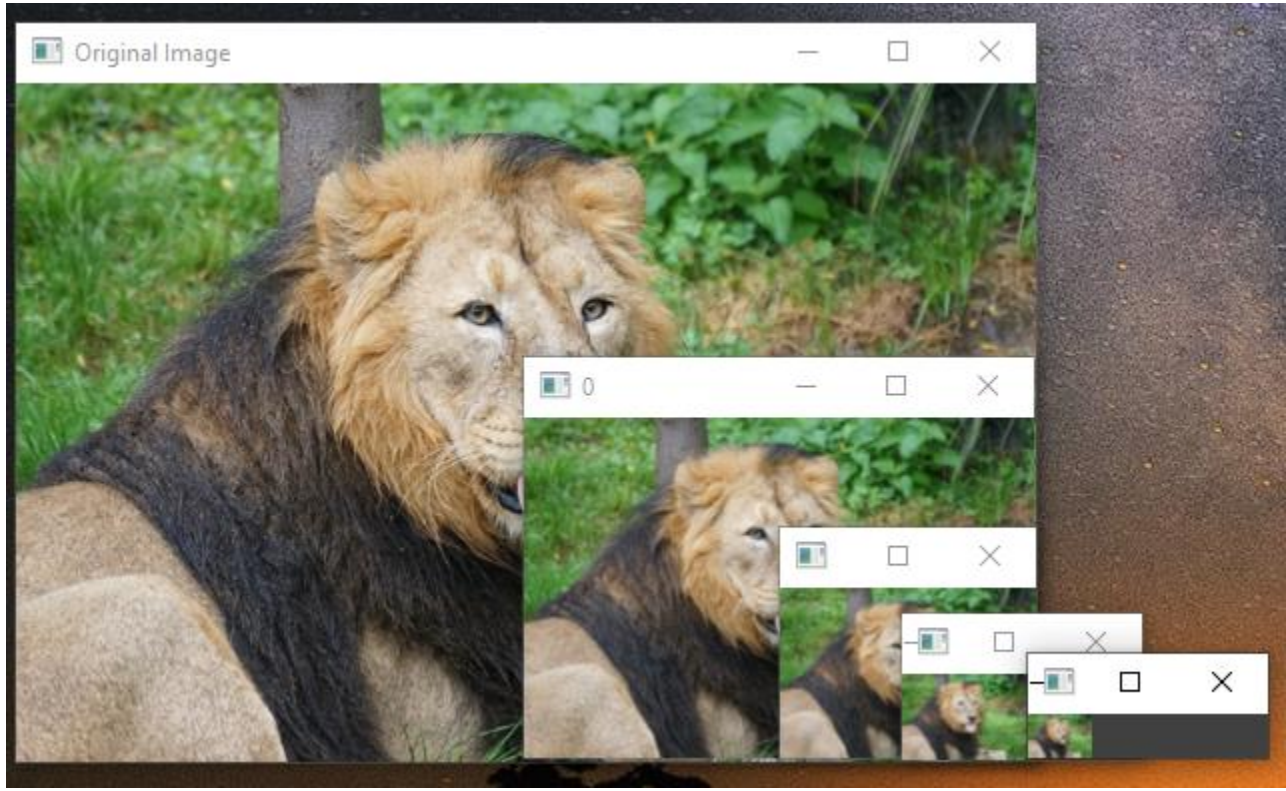


: Forsyth

Gaussian Pyramid OpenCV

```python
import cv2
import numpy as np

img = cv2.imread("0809x4.png")
lr = cv2.pyrDown(img)
lr1 = cv2.pyrDown(lr)
hr1 = cv2.pyrUp(lr1)
hr = cv2.pyrUp(hr1)

cv2.imshow("original image", img)
cv2.imshow("Pyramid image using Gaussian Pyramid", lr)
cv2.imshow("Pyramid image using Gaussian Pyramid Level 2", lr1)
cv2.imshow("Pyramid image using Gaussian Pyramid", hr1)
cv2.imshow("Pyramid image using Gaussian Pyramid Level 2", hr)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```python
import cv2
import numpy as np

img = cv2.imread("0809x4.png")
layer = img.copy()
gp = [layer]

for i in range(6):
    layer = cv2.pyrDown(layer)
    gp.append(layer)
    cv2.imshow(str(i), layer)

cv2.imshow("Original Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Template matching with Gaussian Pyramid

Input: Image, Template

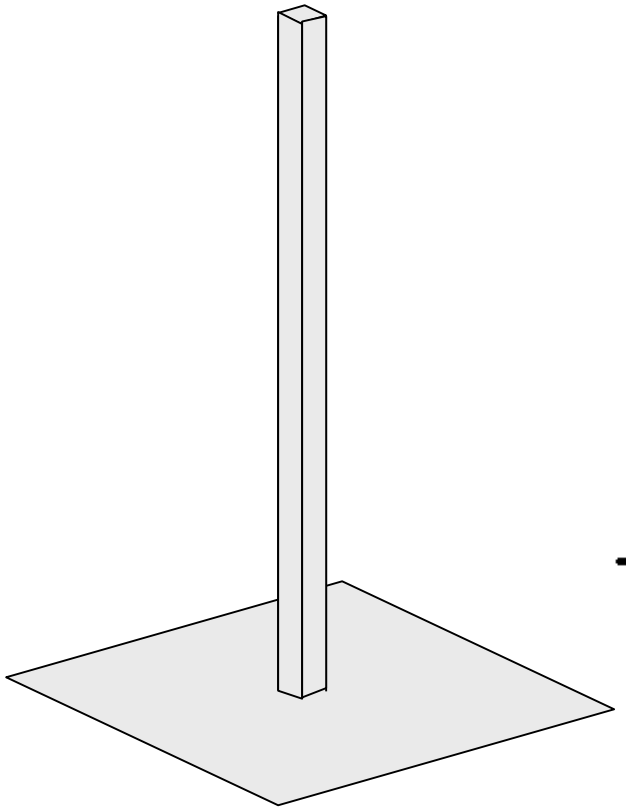
1. Match template at current scale

2. Downsample image
   In practice, scale step of 1.1 to 1.2

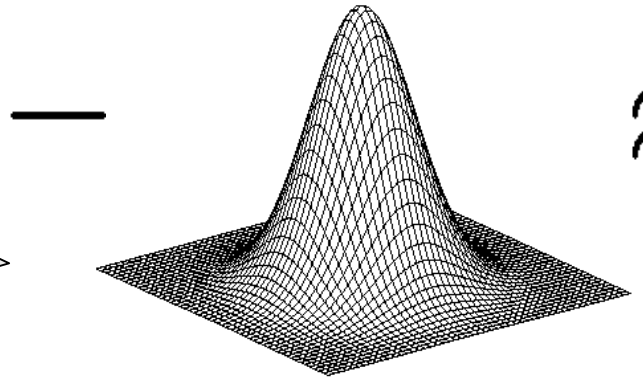
3. Repeat 1-2 until image is very small

4. Take responses above some threshold, perhaps with non-maxima suppression

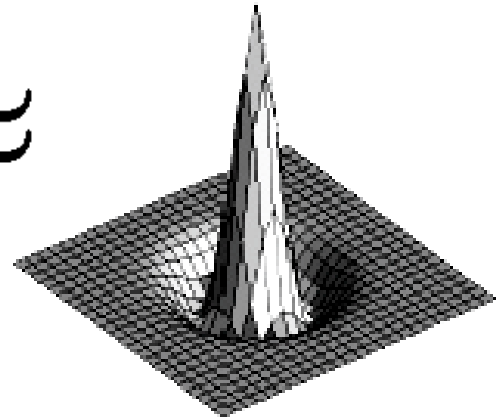
Laplacian Filter



unit impulse



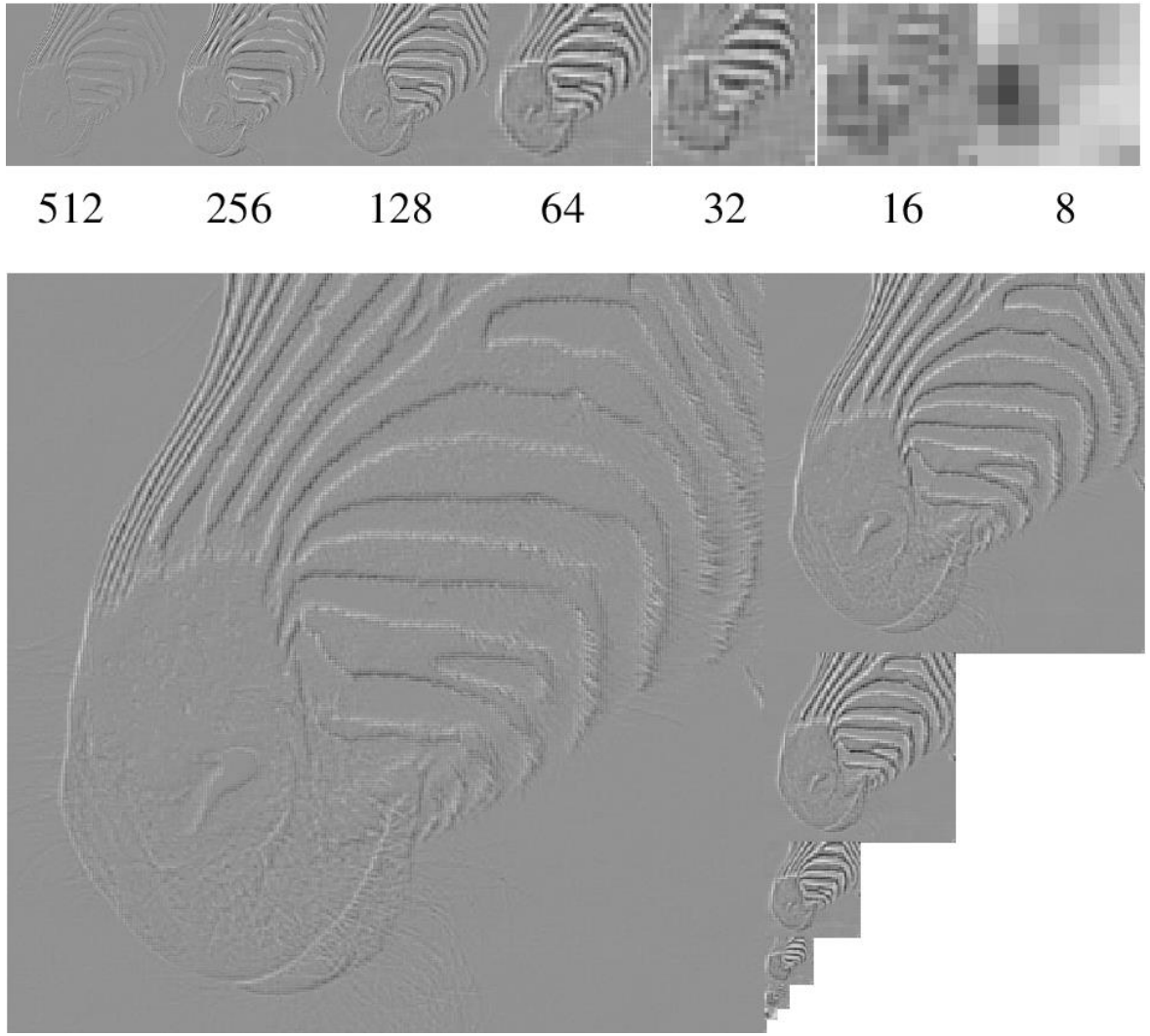
Gaussian



Laplacian of Gaussian

Source: Lazebnik

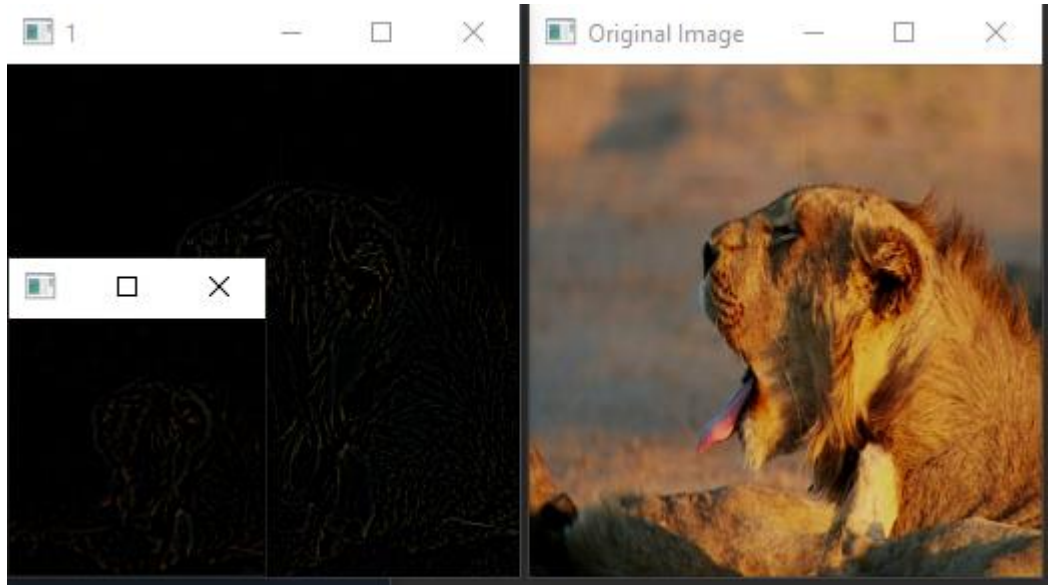
Laplacian Pyramid



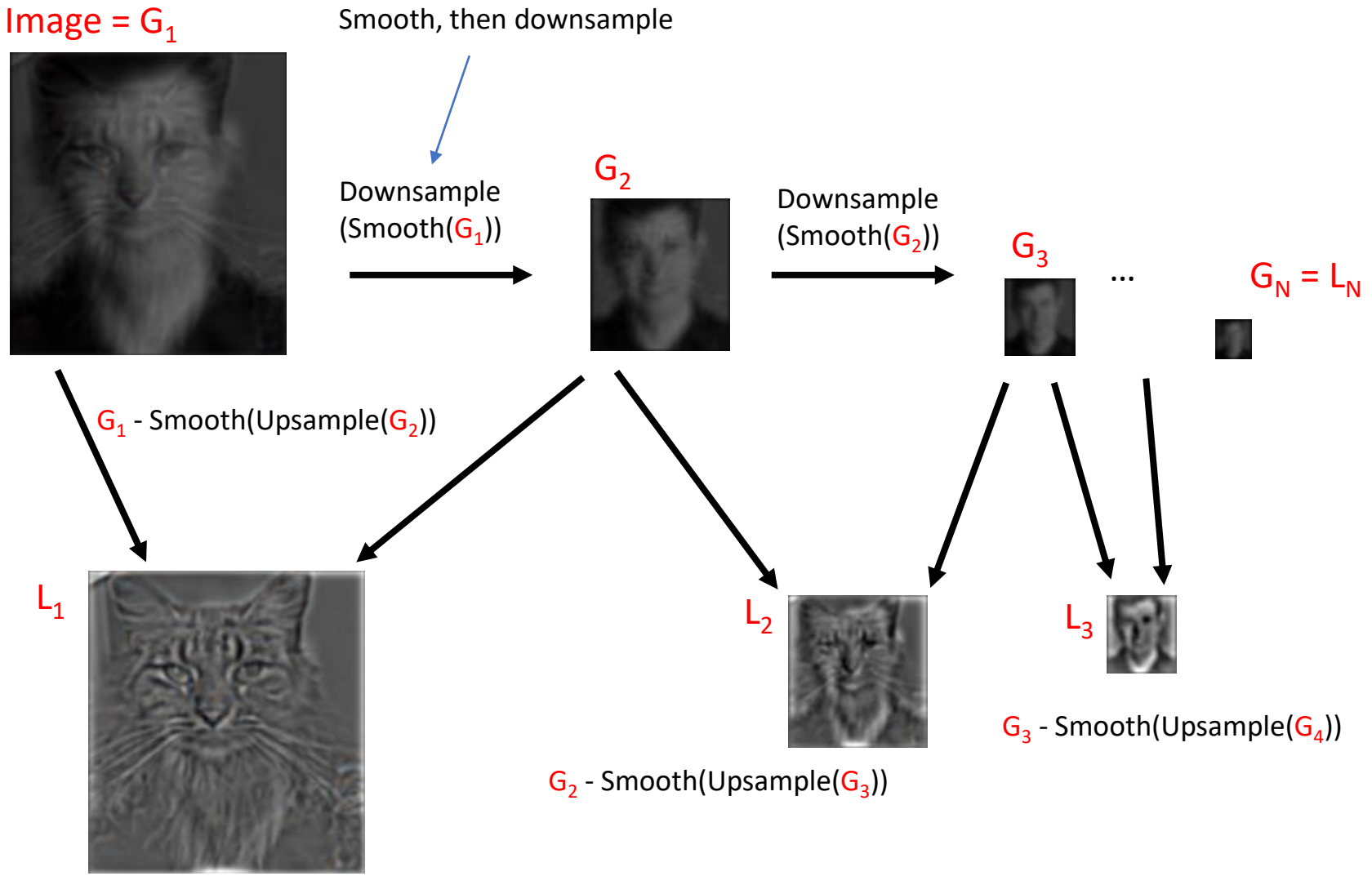
Source: Forsyth

Python Example (Laplacian Pyramid)

```python
import cv2
import numpy as np

img = cv2.imread("09011.png")
layer = img.copy()
gp = [layer]

for i in range(6):
    layer = cv2.pyrDown(layer)
    gp.append(layer)
    #cv2.imshow(str(i), layer)

layer = gp[5]
cv2.imshow('Upper Level gaussian pyramid', layer)

lp = [layer]

for i in range(5, 0, -1):
    gaussian_extend = cv2.pyrUp(gp[i])
    laplacian = cv2.subtract(gp[i-1],gaussian_extend)
    cv2.imshow(str(i),laplacian)


cv2.imshow("Original Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



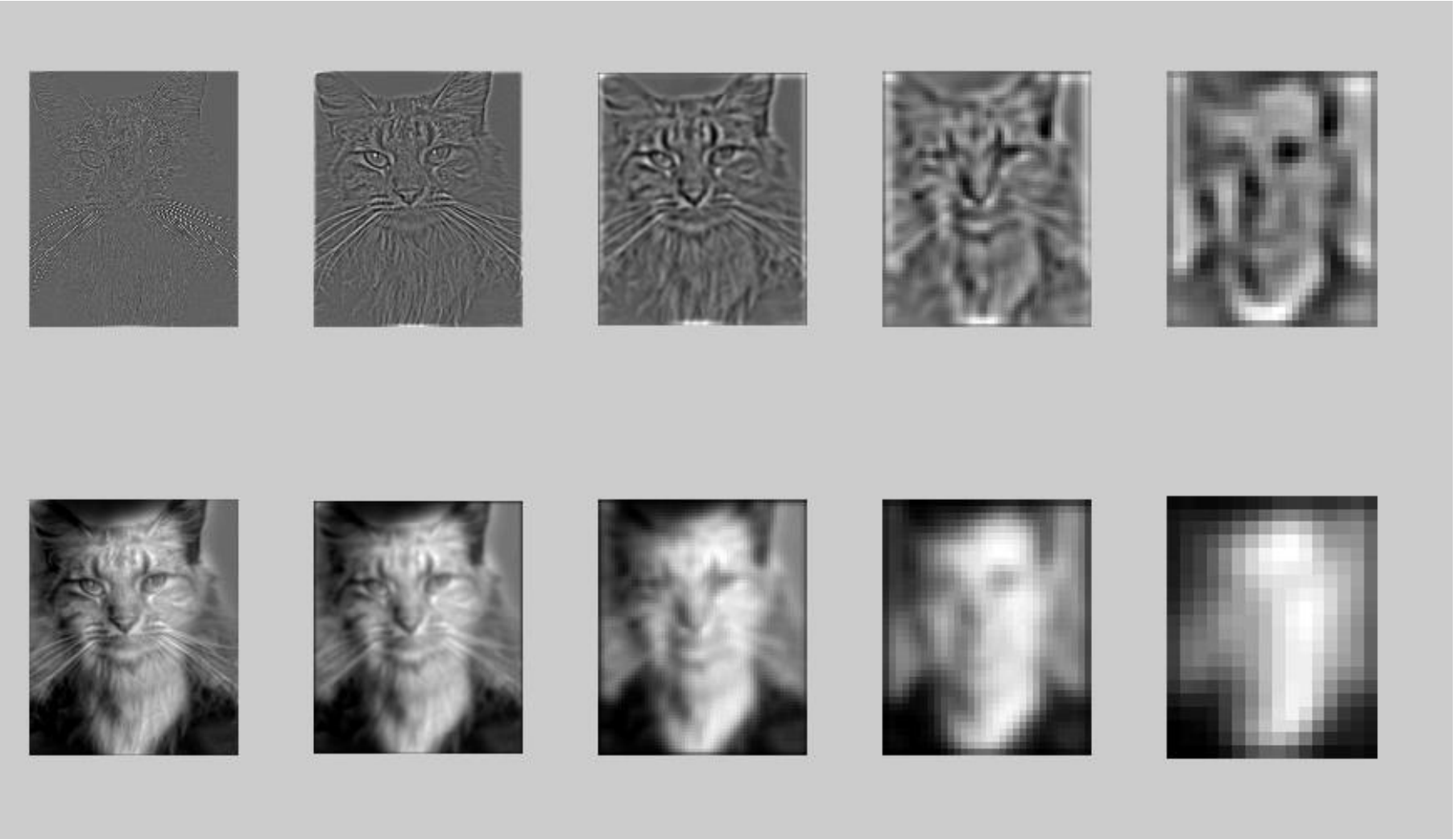
Creating the Gaussian/Laplacian Pyramid



- Use same filter for smoothing in each step (e.g., Gaussian with $\sigma = 2$)
- Downsample/upsample with "nearest" interpolation

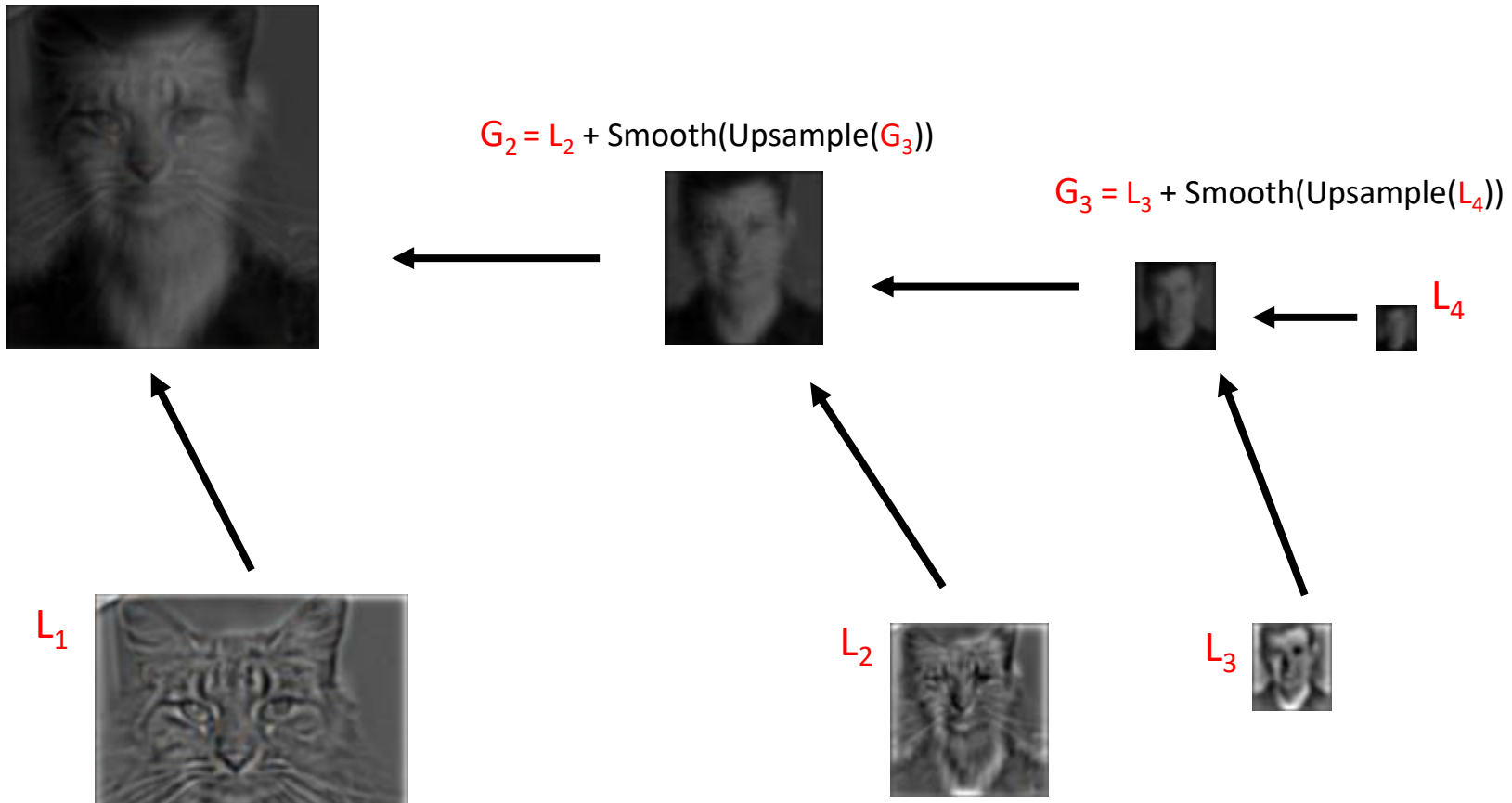
Hybrid image in Laplacian Pyramids

High frequency → Low frequency



Reconstructing Image from Laplacian Pyramid

Image = L_1 + Smooth(Upsample(G_2))



- Use same filter for smoothing as in desconstruction
- Upsample with “nearest” interpolation
- Reconstruction will be lossless

Major Uses of Image Pyramid

- Object detection
  - Scale search
  - Features
- Detecting stable interest points
- Course-to-fine registration
- Compression


LEGEND
IN THE MAKING


LEGEND
IN THE MAKING


LEGEND
IN THE MAKING

Template matching in Python

```python
# Python program to illustrate
# template matching
import cv2
import numpy as np

# Read the main image
img_rgb = cv2.imread('Image Path').

# Convert it to grayscale
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)

# Read the template
template = cv2.imread('template', 0)

# Store width and height of template in w and h
w, h = template.shape[::-1]

# Perform match operations.
res = cv2.matchTemplate(img_gray, template, cv2.TM_CCOEFF_NORMED)

# Specify a threshold
threshold = 0.8

# Store the coordinates of matched area in a numpy array
loc = np.where(res >= threshold)

# Draw a rectangle around the matched region.
for pt in zip(*loc[::-1]):
    cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0, 255, 255), 2)

# Show the final image with the matched area.
cv2.imshow('Detected', img_rgb)
```

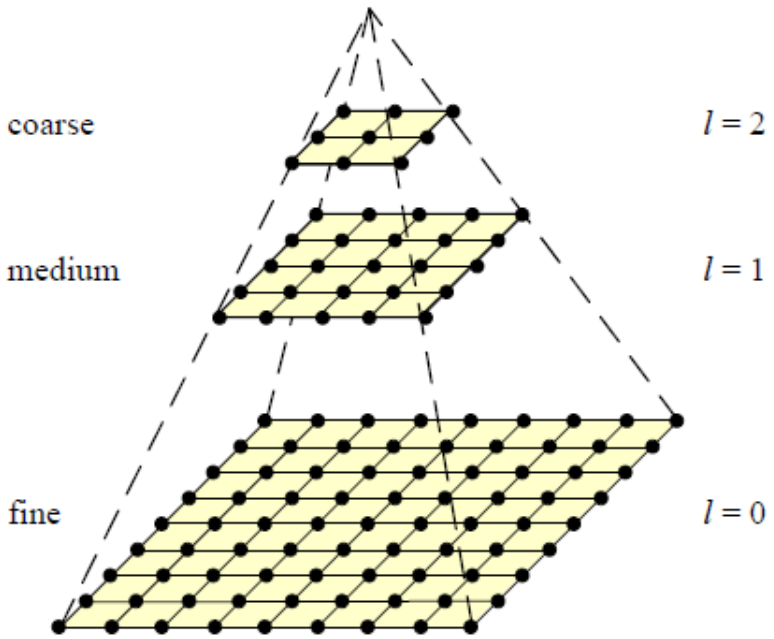
```python
# Python program to illustrate
# template matching
import cv2
import numpy as np

# Read the main image
img_rgb = cv2.imread('Image Path').

# Convert it to grayscale
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)

# Read the template
template = cv2.imread('template', 0)

# Store width and height of template in w and h
w, h = template.shape[::-1]

# Resize the image according to scale and
# keeping track of ratio of resizing
resize = imutils.resize(img_gray, width=int(shape[0]), height=int(img_gray.shape[1]*scale)

# If resize image is smaller than that of template
# break the loop
# Detect edges in the resized, grayscale image and apply template
# Matching to find the template in image edged
# If we have found a new maximum correlation value, update
# the found variable if
# found = null/maxVal > found][0]
if resized.shape[0] < h or resized.shape[1] < w:
        break
found=(maxVal, maxLoc, r)

# Unpack the found variables and compute(x,y) coordinates
# of the bounding box
(__, maxLoc, r)=found
(startX, startY)=(int(maxLoc[0]*r), int maxLoc[1]*r)
(endX, endY)=(int((maxLoc[0]+tw)*r), int(maxLoc[1]+tH)*r)

# Draw a bounding box around the detected result and display the image
cv2.rectangle(image, (startX, startY), (endX, endY), (0, 0, 255), 2)
cv2.imshow("Image", image)
cv2.waitKey(0)
```

Pyramid Application: Coarse to Fine Image Registration

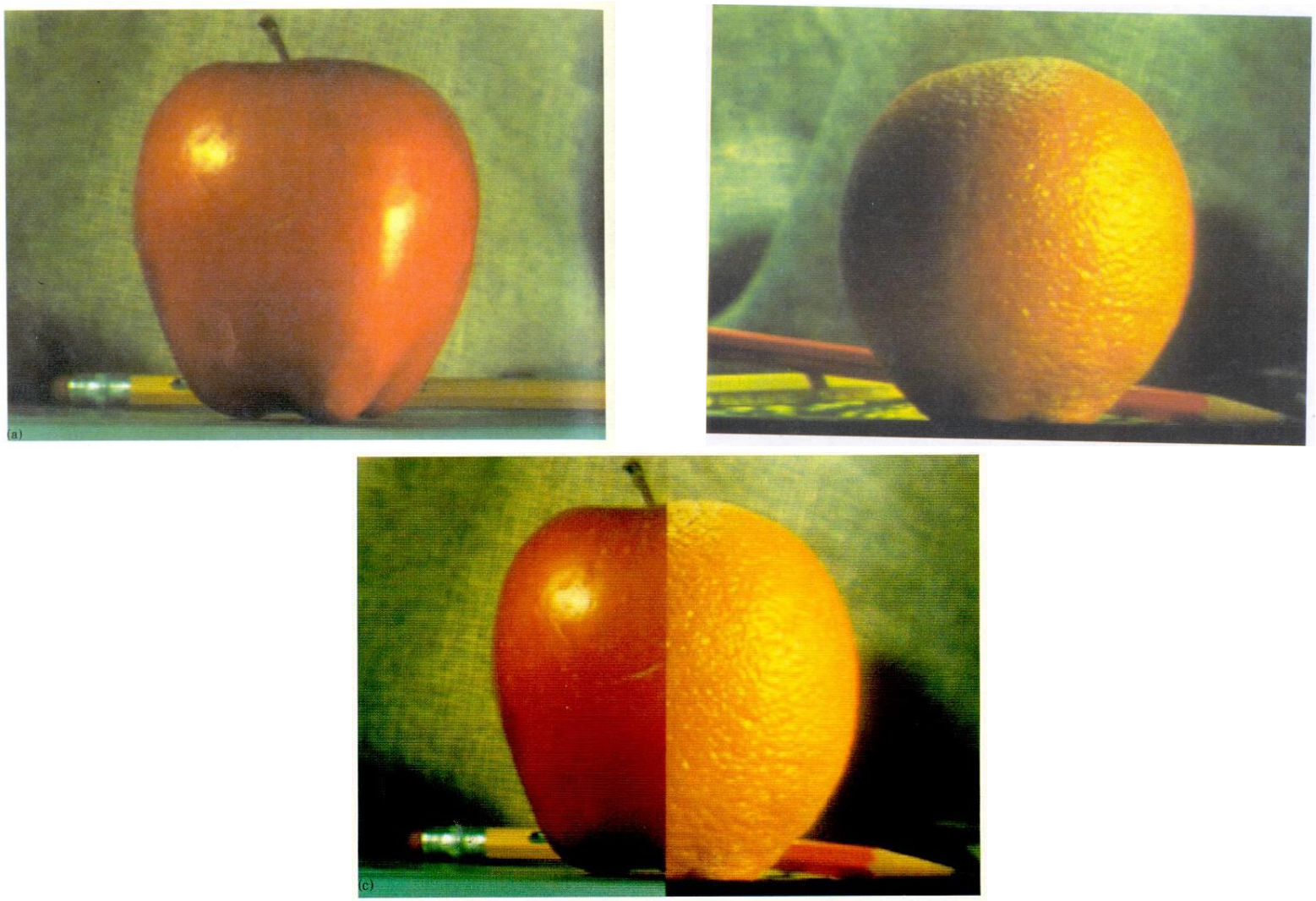
1. Compute Gaussian pyramid
2. Align with coarse pyramid
3. Successively align with finer pyramids
   - Search smaller range

Why is this faster?

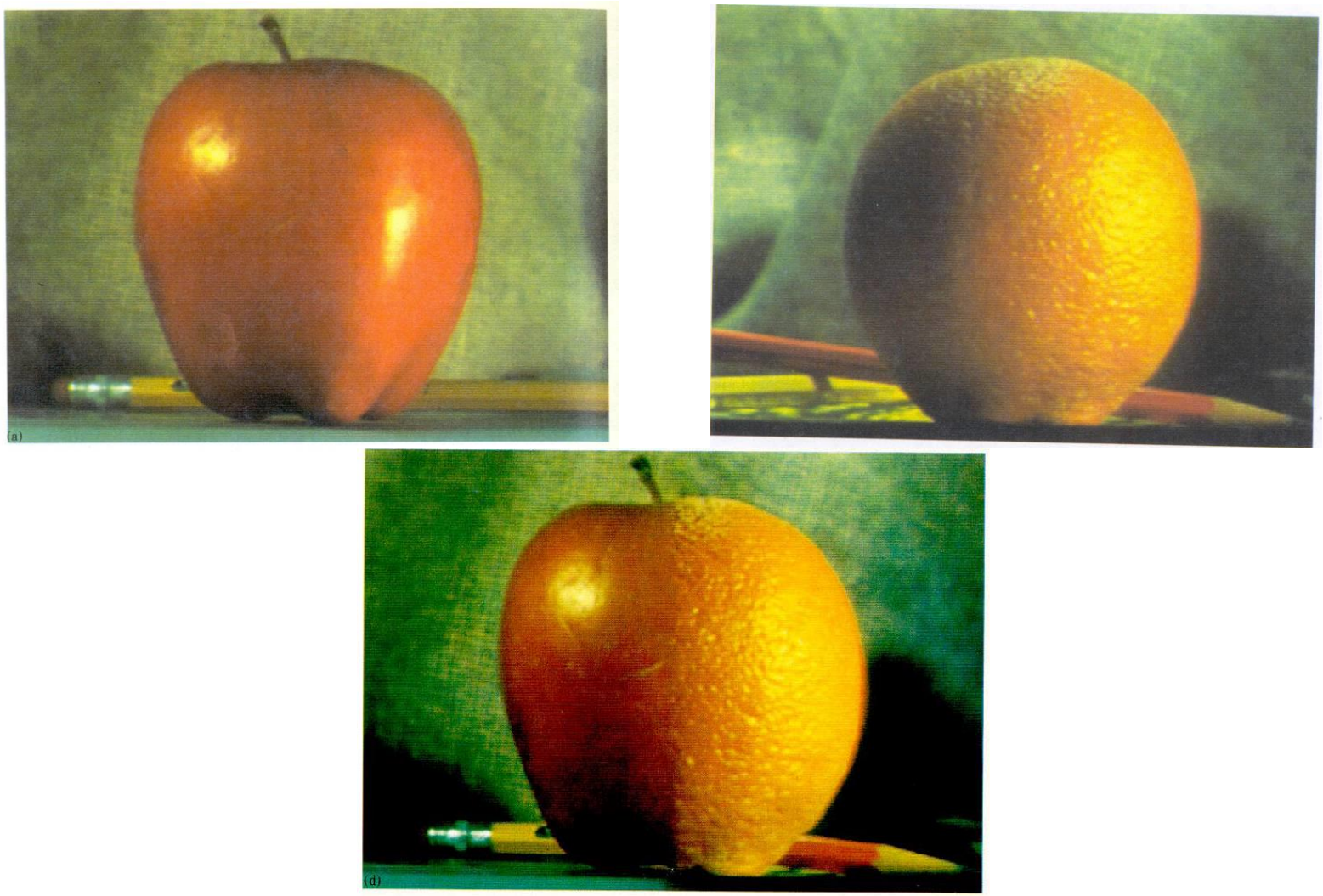
Are we guaranteed to get the same result?



Applications: Pyramid Blending

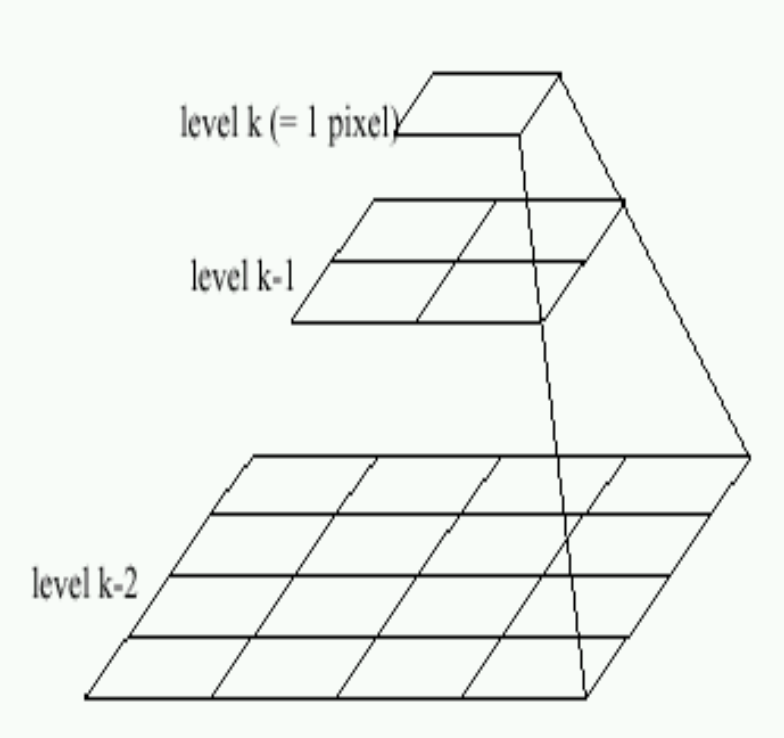


Applications: Pyramid Blending

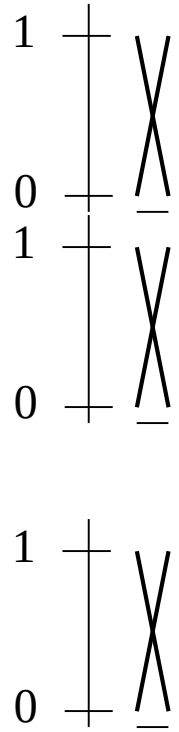


Pyramid Blending

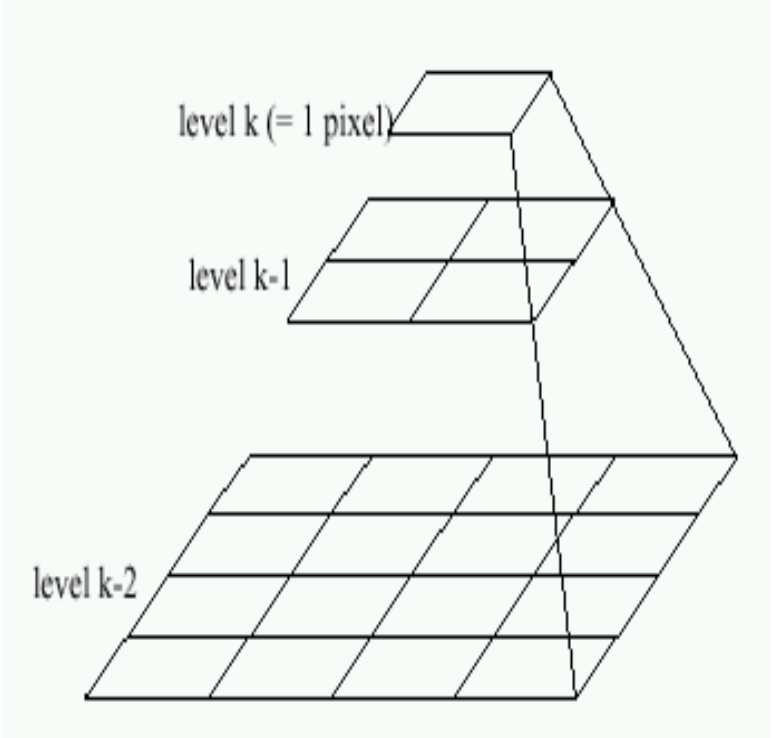
- At low frequencies, blend slowly
- At high frequencies, blend quickly



Left pyramid

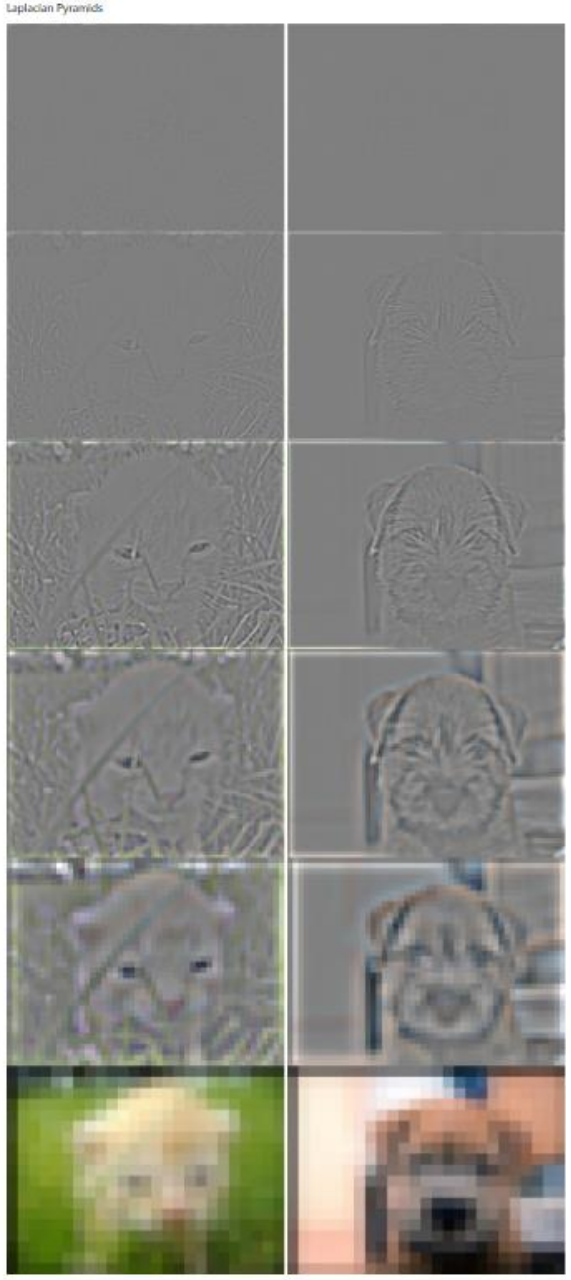


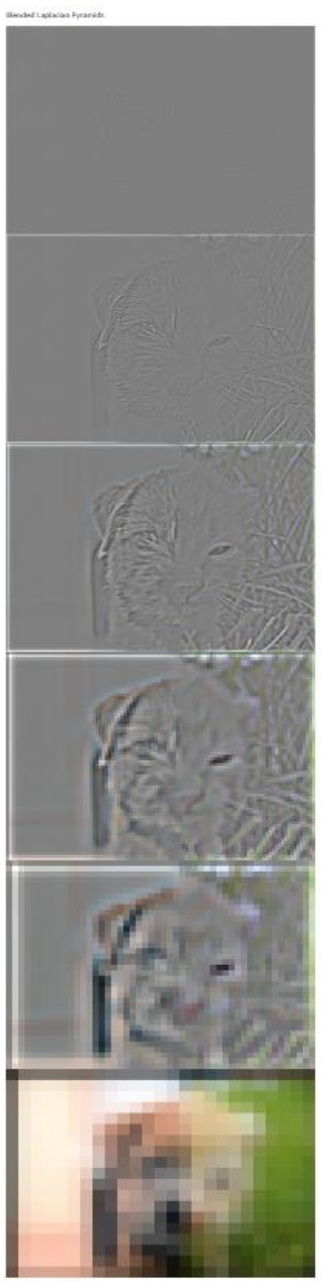
blend



Right pyramid

Pyramid Blending: Examples





Blended Output Image

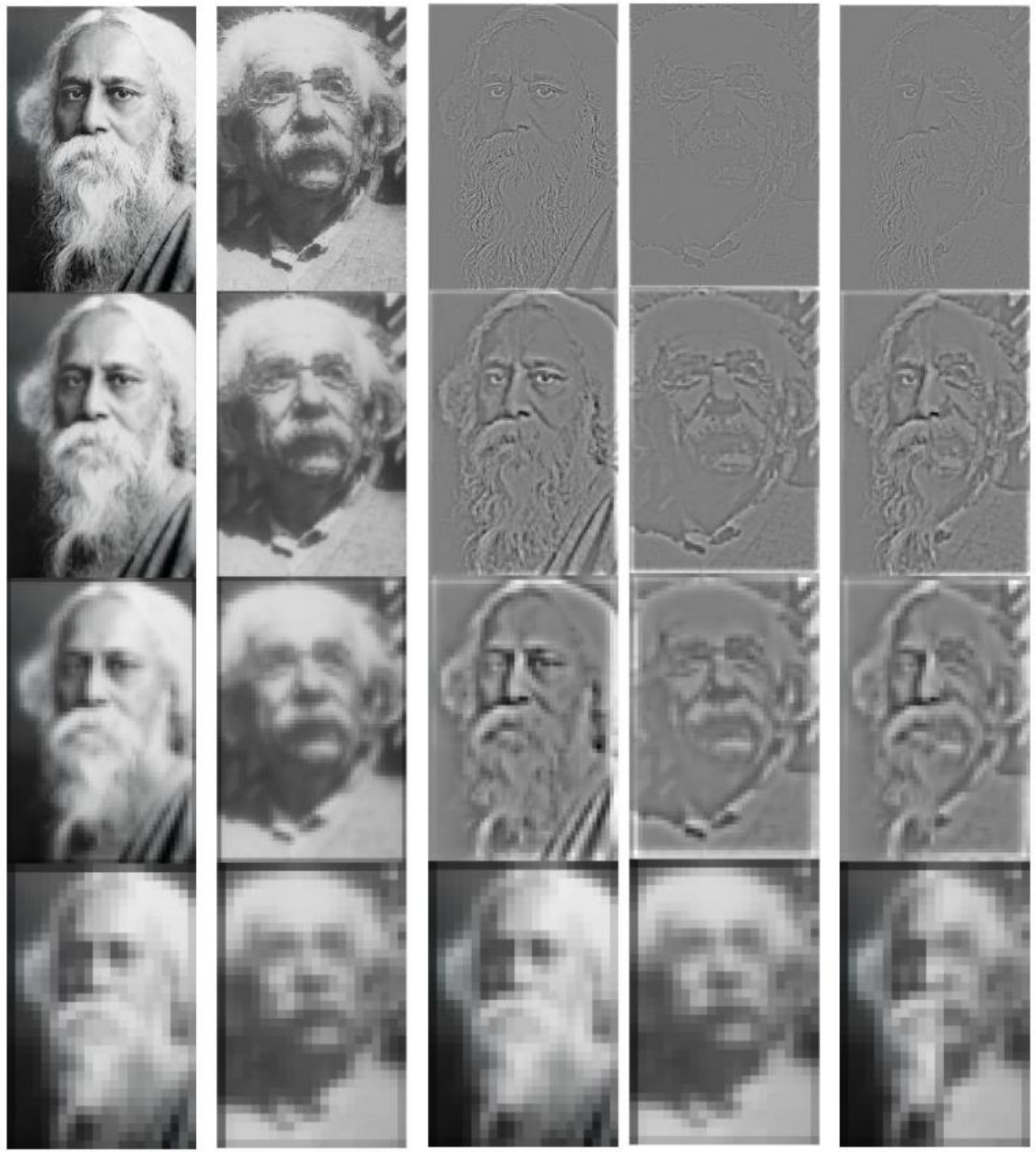
ja

Pyramid: Blending Example

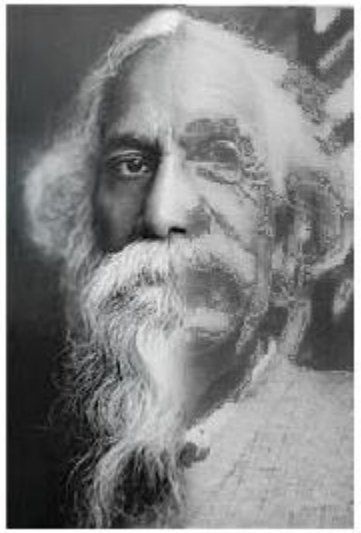
Gaussian Pyramids

Laplacian Pyramids

Blended Laplacian Pyramids



Blended Output Image

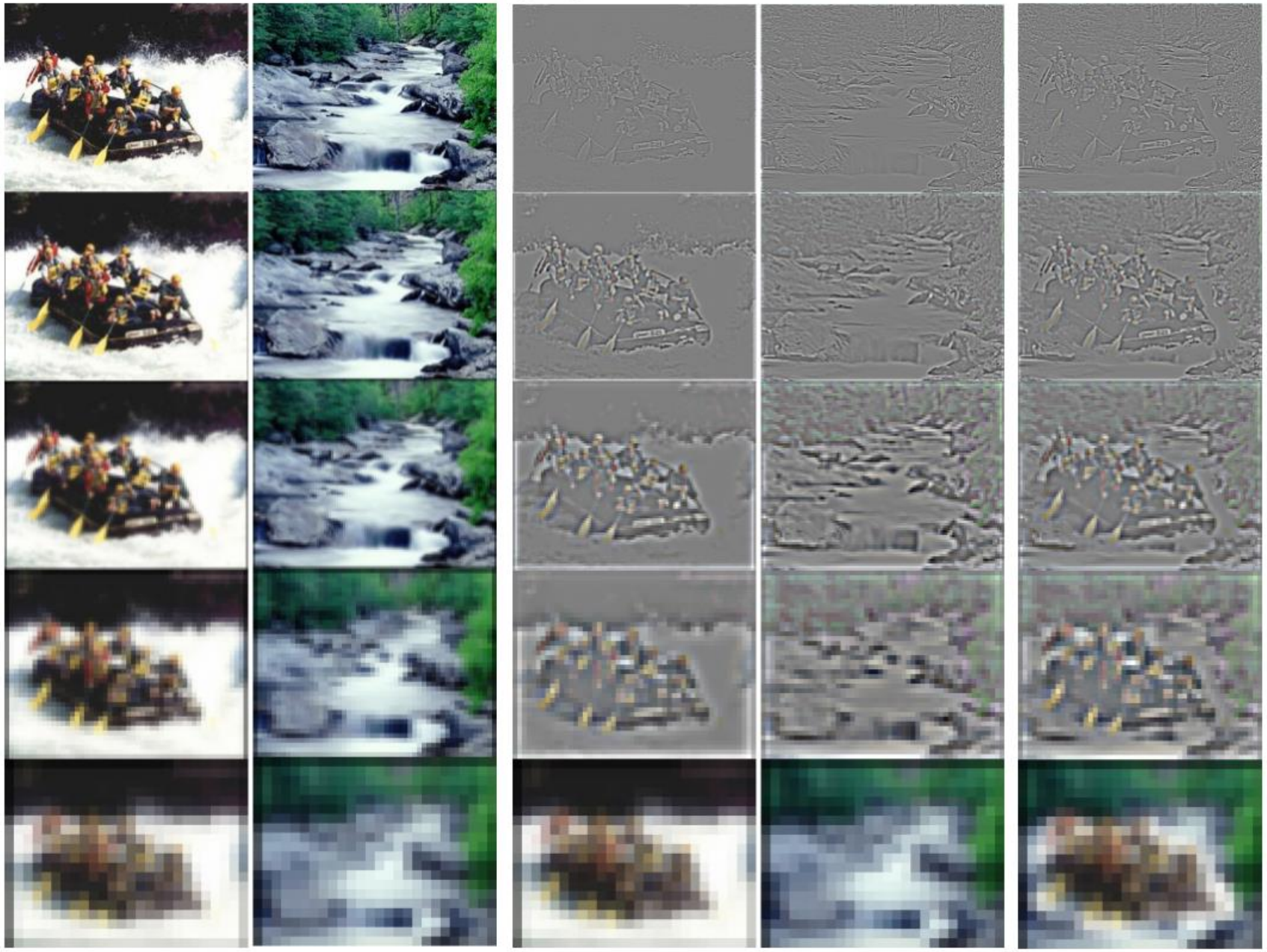


Pyramid: Blending Example

Gaussian Pyramids

Laplacian Pyramids

Blended Laplacian Pyramids

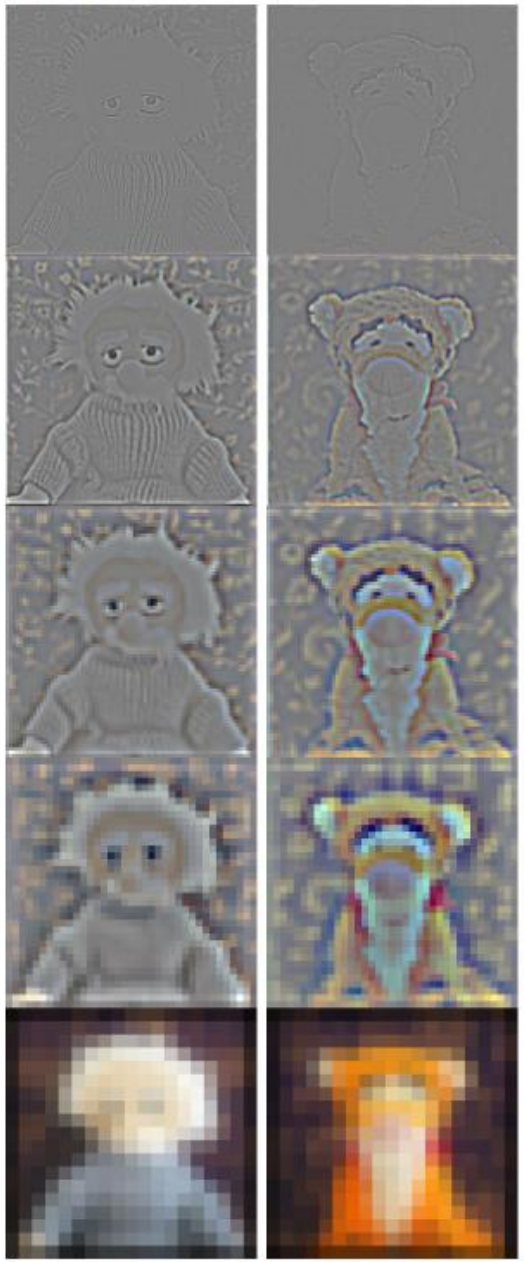


Blended Output Image

Pyramid: Blending Example

Gaussian Pyramids

Laplacian Pyramids



Blended Laplacian Pyramids

Blended Output Image

Python code for Image Blending

```python
import cv2
import numpy as np

img1 = cv2.imread('0809.png')
img2 = cv2.imread('09011.png')
print(img1.shape)
print(img2.shape)
blend_lion = np.hstack((img1[:, :256], img2[:, 256:]))
## generate Gaussian pyramid for lion1
img1_copy = img1.copy()
gp_img1 = [img1_copy]
for i in range(6):
    img1_copy = cv2.pyrDown(img1_copy)
    gp_img1.append(img1_copy)
## generate Gaussian pyramid for lion2
img2_copy = img2.copy()
gp_img2 = [img2_copy]
for i in range(6):
    img2_copy = cv2.pyrDown(img2_copy)
    gp_img2.append(img2_copy)
## generate Laplacian Pyramid for apple
img1_copy = gp_img1[5]
lp_img1 = [img1_copy]
for i in range(5, 0, -1):
    gaussian_expanded = cv2.pyrUp(gp_img1[i])
    laplacian = cv2.subtract(gp_img1[i-1], gaussian_expanded)
    lp_img1.append(laplacian)
## generate Laplacian Pyramid for orange
img2_copy = gp_img2[5]
lp_img2 = [img2_copy]
for i in range(5, 0, -1):
    gaussian_expanded = cv2.pyrUp(gp_img2[i])
    laplacian = cv2.subtract(gp_img2[i-1], gaussian_expanded)
    lp_img2.append(laplacian)
## Now add left and right halves of images in each level
img1_img2_pyramid = []
n = 0
for img1_lap, img2_lap in zip(lp_img1, lp_img2):
    n += 1
    cols, rows, ch = img1_lap.shape
    laplacian = np.hstack((img1_lap[:, 0:int(cols/2)], img2_lap[:, int(cols/2):]))
    img1_img2_pyramid.append(laplacian)
# now reconstruct
img1_img2_reconstruct = img1_img2_pyramid[0]
for i in range(1, 6):
    img1_img2_reconstruct = cv2.pyrUp(img1_img2_reconstruct)
    img1_img2_reconstruct = cv2.add(img1_img2_pyramid[i], img1_img2_reconstruct)
cv2.imshow("img1", img1)
cv2.imshow("img2", img2)
cv2.imshow("img1_img2", img1_img2_pyramid)
cv2.imshow("img1_img2_reconstruct", img1_img2_reconstruct)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Dr. Sander Ali Khowaja

MATLAB code for Image Blending

```
A = im2double(imread("0855x4.png"));
A = imresize(A,[256 256]);
B = im2double(imread("0880x4.png"));
B = imresize(B,[256,256]);
subplot(1,2,1)
imshow(A)
subplot(1,2,2)
imshow(B)

mask_A = zeros(256,256);
mask_A(:,1:128) = 1;

clf
imshow(mask_A)
xticks([])
yticks([])
axis on

C = (A .* mask_A) + (B .* (1 - mask_A));
clf
imshow(C)
xticks([])
yticks([])

mrp_A = multiresolutionPyramid(A);
mrp_B = multiresolutionPyramid(B);
mrp_mask_A = multiresolutionPyramid(mask_A);

lap_A = laplacianPyramid(mrp_A);
lap_B = laplacianPyramid(mrp_B);
```

```

for k = 1:length(lap_A)
    lap_blend{k} = (lap_A{k} .* mrp_mask_A{k}) + ...
        (lap_B{k} .* (1 - mrp_mask_A{k}));
end

C_blended = reconstructFromLaplacianPyramid(lap_blend);
imshow(C_blended)

x = linspace(-1,1,256);
y = x';
mask2_A = hypot(x,y) <= 0.5;

mrp_mask2_A = multiresolutionPyramid(mask2_A);

for k = 1:length(lap_A)
    lap_blend2{k} = (lap_A{k} .* mrp_mask2_A{k}) + ...
        (lap_B{k} .* (1 - mrp_mask2_A{k}));
end

C2_blended = reconstructFromLaplacianPyramid(lap_blend2);
imshow(C2_blended)
```

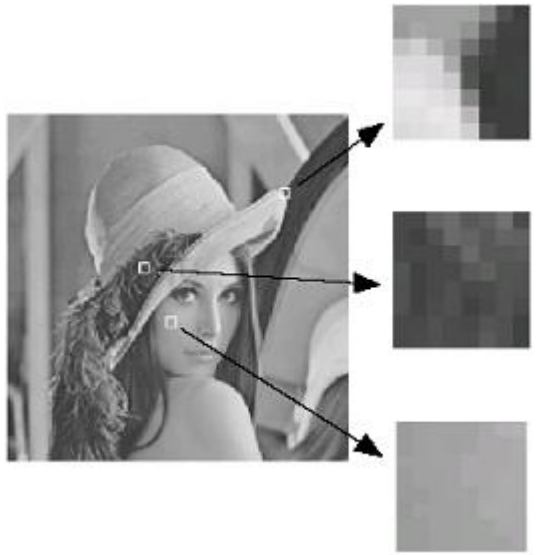
Image Representations

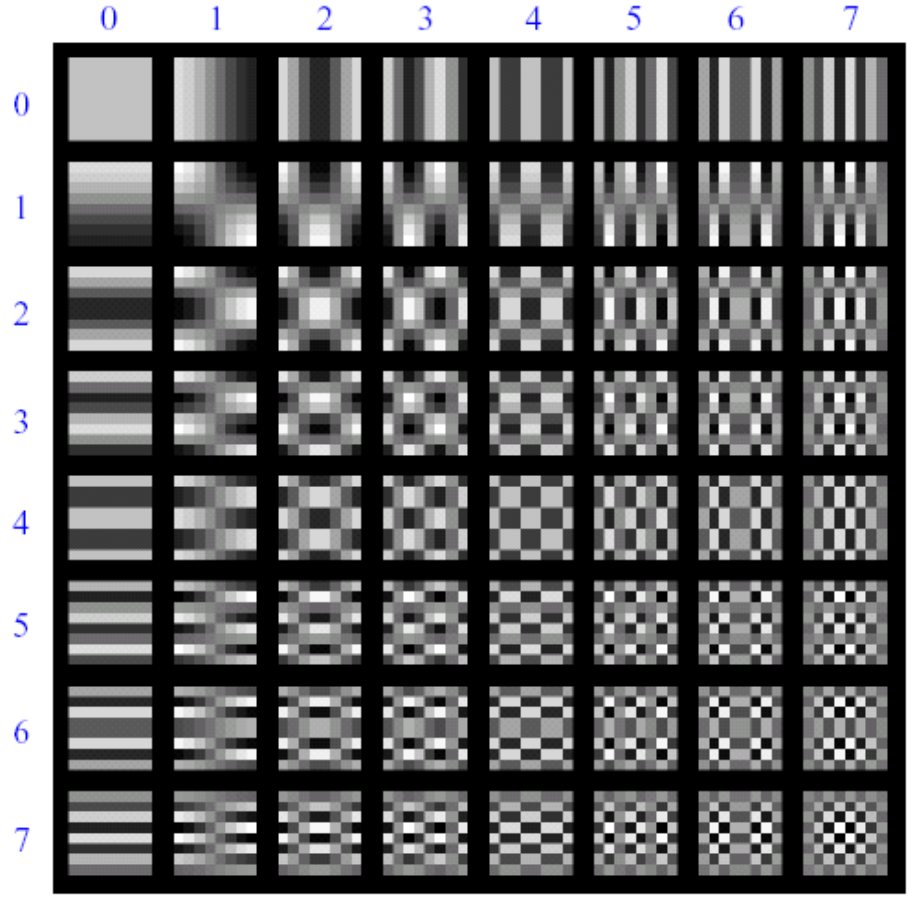
- Pixels:
  - great for spatial resolution, poor access to frequency
- Fourier transform:
  - great for frequency, not for spatial info
- Pyramids/filter banks:
  - balance between spatial and frequency information

Compression

How is it that a 4MP image (12000KB) can be compressed to 400KB without a noticeable change?

Lossy Image Compression (JPEG)



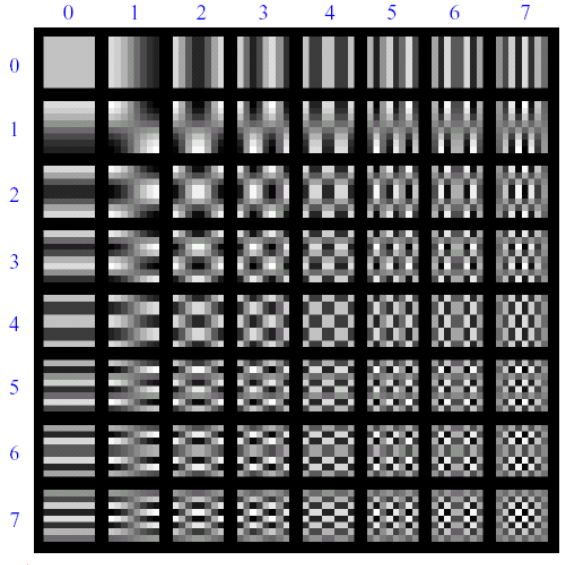


Block-based Discrete Cosine Transform (DCT)

Slides: Efros

Using DCT in JPEG

- The first coefficient $B(0,0)$ is the DC component, the average intensity
- The top-left coeffs represent low frequencies, the bottom right – high frequencies



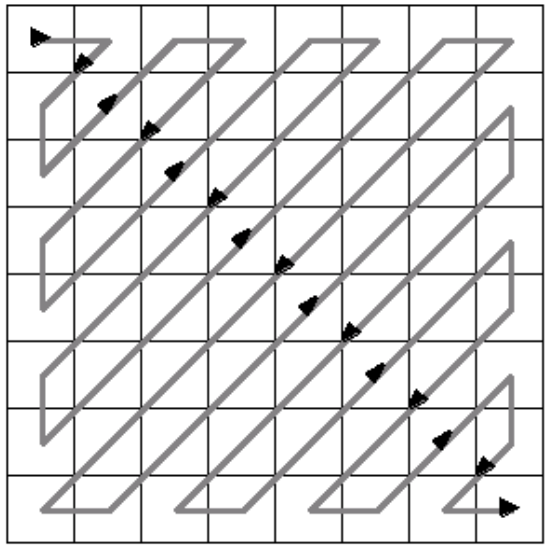


Image Compression using DCT

- Quantize
  - More coarsely for high frequencies (which also tend to have smaller values)
  - Many quantized high frequency values will be zero
- Encode
  - Can decode with inverse dct

Filter responses

$$G = \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix}$$

(u →, v ↓)

Quantized values

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Quantization table

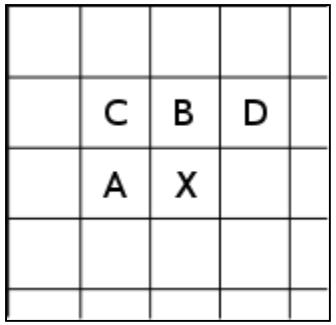
$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

JPEG Compression Summary

1. Convert image to YCrCb
2. Subsample color by factor of 2
   - People have bad resolution for color
3. Split into blocks (8x8, typically), subtract 128
4. For each block
   a. Compute DCT coefficients
   b. Coarsely quantize
      - Many high frequency components will become zero
   c. Encode (e.g., with Huffman coding)

Lossless Compression (PNG)

1. Predict that a pixel's value based on its upper-left neighborhood
2. Store difference of predicted and actual value
3. Pkzip it (DEFLATE algorithm)



Three views of Image Filtering

- Image filters in spatial domain
  - Filter is a mathematical operation on values of each patch
  - Smoothing, sharpening, measuring texture

- Image filters in the frequency domain
  - Filtering is a way to modify the frequencies of images
  - Denoising, sampling, image compression

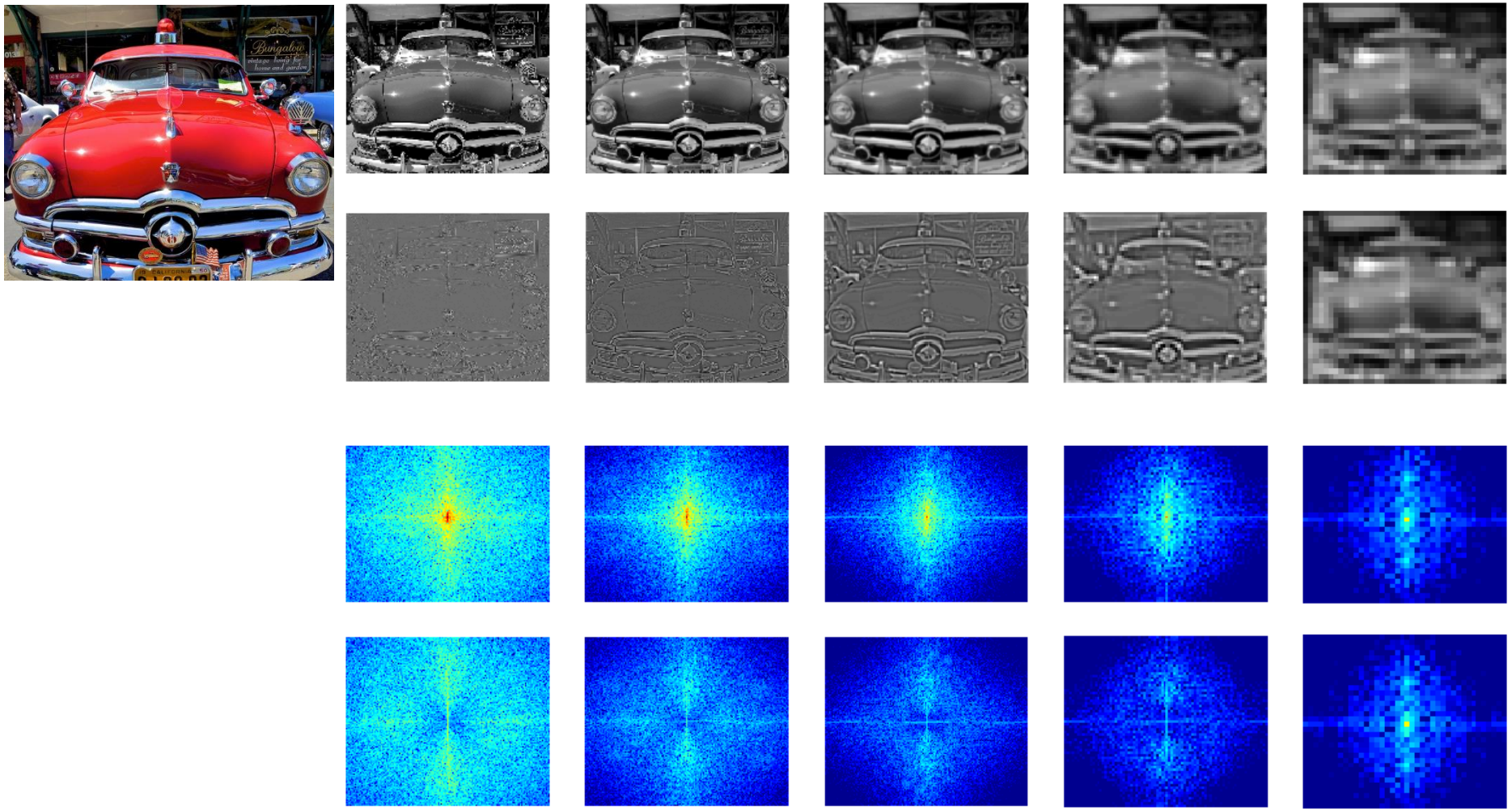
- Templates and Image Pyramids
  - Filtering is a way to match a template to the image
  - Detection, coarse-to-fine registration

Complex Engineering Task

- Hybrid image =

Low-Freq(Image A) + Hi-Freq(Image B)

Complex Engineering Task



Complex Engineering Task

Some Image Denoising Results

Things to Remember

- Template matching (SSD or Normxcorr2)
  - SSD can be done with linear filters, is sensitive to overall intensity
- Gaussian pyramid
  - Coarse-to-fine search, multi-scale detection
- Laplacian pyramid
  - More compact image representation
  - Can be used for compositing in graphics
- Compression
  - In JPEG, coarsely quantize high frequencies