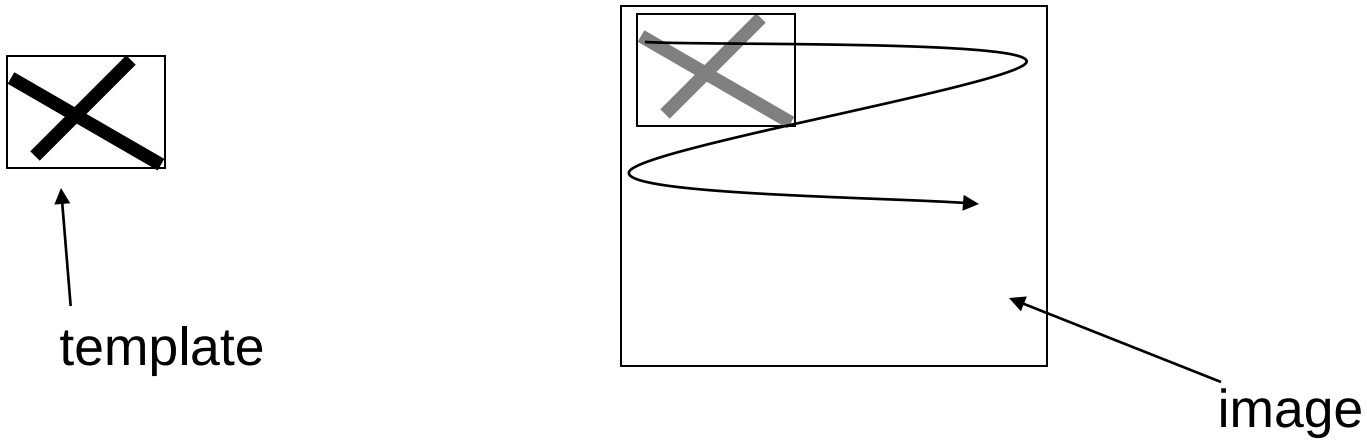
Measures of similarity

- An approach to detect some previously defined object within a given image is to perform *template matching*
- A *template* is a sub-image representing the “ideal”pattern that is sought in the image
- Involves the translation of the *template* to every possible position of the image, and the evaluation of a the level of *match* between the template and the image in that position
- *Global* versus *local* template



- For each position of the template in the image we evaluate a *similarity measure* between the template and the image
- Possible measures are *summation difference* or *cross-correlation*
- These measures are not only used for template matching but also to estimate the level of similarity between two signals

1. Euclidean distance

- Estimate the similarity between *g* and *t* in *m,n*:

$$E(m,n) = \sqrt{\sum_i \sum_j \left[g(i,j) - t(i-m, j-n) \right]^2}$$

The sum is computed for all points for which (i-*m,j-n)* is a valid coordinate of *t*, in other words for all points in the template image

- To find the *best match* we look for the smallest value in *E(m,n)*

- From an intuitive point of view we are computing the distance between two *HxW* dimensional points, where *H,W* is the size of the template
- Since we are not interested in the exact value of the distance (we search for the minimum in *E*) we can avoid computing the square root (Sum of Squared Differences):

$$SSD(m,n) = E^2(m,n) = \sum_i \sum_j \left[g(i,j) - t(i-m, j-n) \right]^2$$

- Or:

$$S(m,n) = \sum_i \sum_j \left| g(i,j) - t(i-m, j-n) \right|$$

$$E^2(m,n) = \sum_i \sum_j \left[g(i,j)^2 + t(i-m, j-n)^2 - 2g(i,j)t(i-m, j-n) \right]$$

assume this constant

this is constant

2. Cross-correlation

$$R(m,n) = \sum_i \sum_j \left[g(i,j)t(i-m, j-n) \right]$$

t and *g* are similar when *R(i,j)* is **large**

Problems with cross-correlation:

-False matches if the image energy $\sum\sum g(i,j)^2$ varies with the position

-Values of R depends on the size of the template

-Not invariant to illumination change

Consider the correlation with a constant pattern, of gray value *v*

a	b	c
d	e	f
g	h	i

v	v	v
v	v	v
v	v	v

R=v*(a+b+c+…+i)

Now consider the correlation with a constant image twice as bright

a	b	c
d	e	f
g	h	i

2v	2v	2v
2v	2v	2v
2v	2v	2v

R=2*v*(a+b+c+…+i)>v*(a+b+c+…i)

We get higher correlation, with the same template…

- Solution: normalize the intensity
  - subtract the mean of both signals
  - divide by std. deviation

$$g = \frac{g - \overline{g}}{\sqrt{\sum \left(g - \overline{g}\right)^2}}, \qquad \hat{t} = \frac{t - \bar{t}}{\sqrt{\sum \left(t - \bar{t}\right)^2}}$$

The *normalized cross-correlation* is defined as:

$$NCC(m,n) = \frac{\sum_{i,j}\left[g(i,j) - \overline{g}_{m,n}\right]\left[t(i-m, j-n) - \bar{t}\right]}{\sqrt{\sum_{i,j}\left[g - \overline{g}_{m,n}\right]^2 \sum_{i,j}\left[t(i-m, j-n) - \bar{t}\right]^2}} \in [-1,1]$$

$\bar{t}$ mean value of t, $\overline{g}_{m,n}$ mean value of g in the region under t

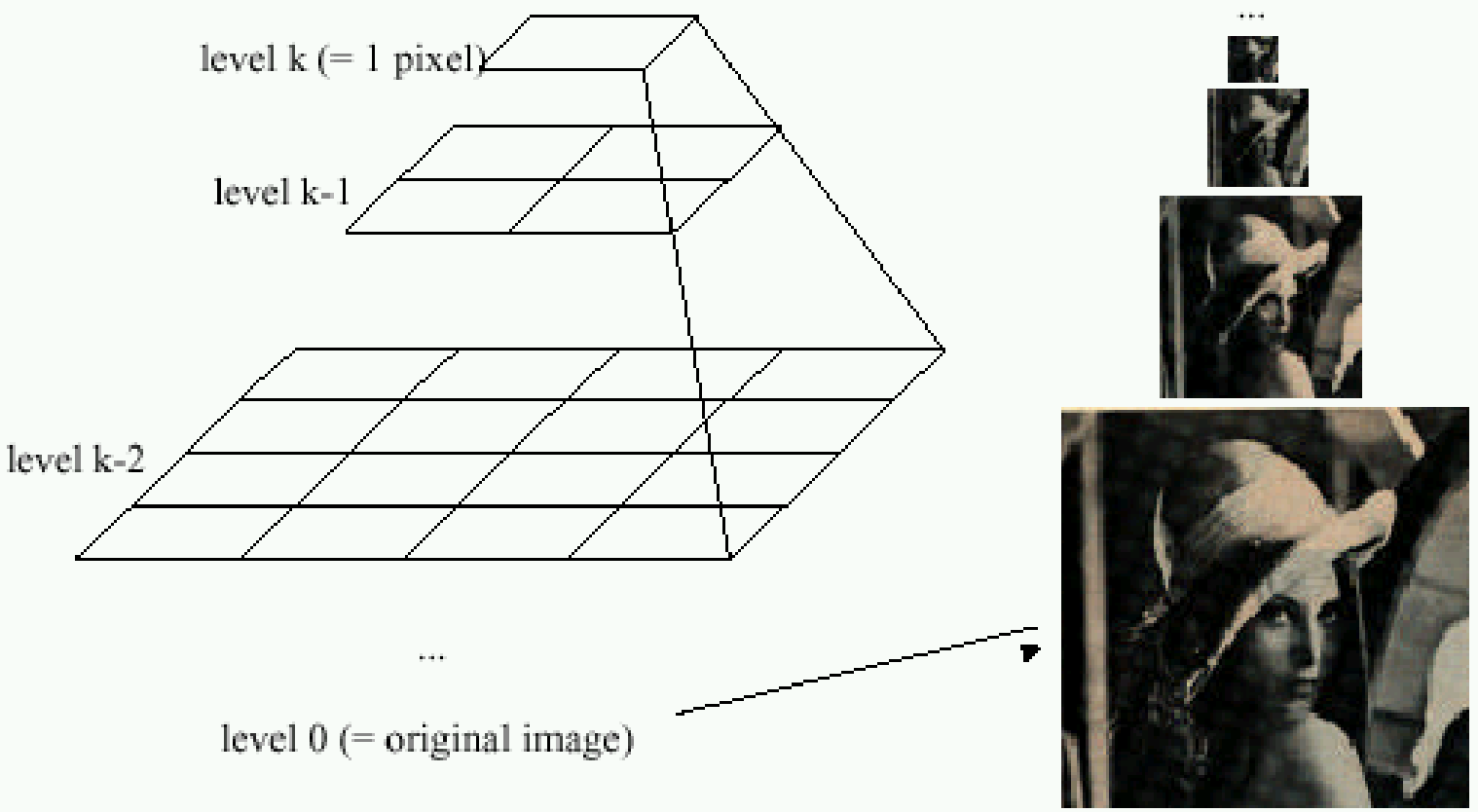
Problems with template matching

- The template represents the object as we expect to find it in the image
- The object can indeed be scaled or rotated
- This technique requires a separate template for each scale and orientation
- Template matching become thus too expensive, especially for large templates

Gaussian Pyramid (Burt and Adelson 1983)

- Image pyramid is a collection of representations of an image
- Typically each layer of the pyramid is half the width and half the height of the previous layer
- If we stack the layers on top of each other, we get a pyramid:
  - lowest level, highest resolution
  - highest level, lowest resolution

Idea: Represent NxN image as a "pyramid" of
1x1, 2x2, 4x4,…, 2^kx2^k images (assuming N=2^k)

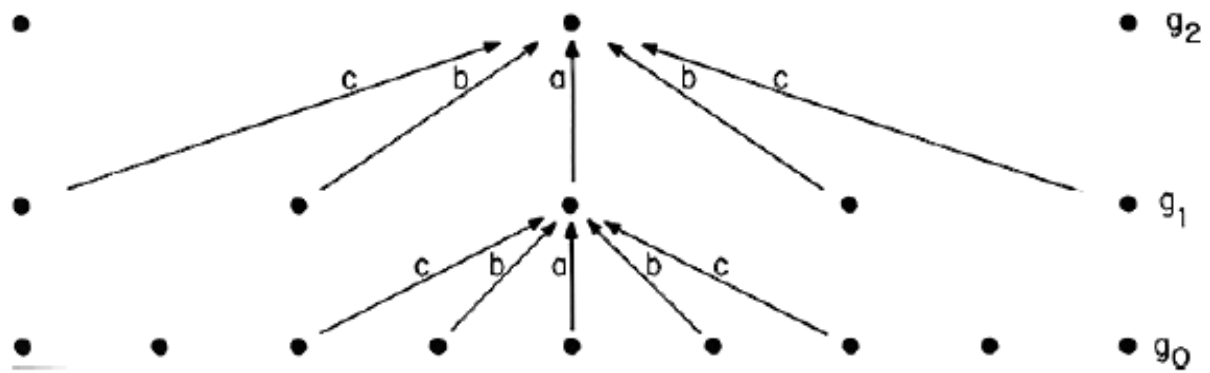


- In a Gaussian pyramid, each layer is smoothed by a symmetric Gaussian kernel, and resampled to get the next layer
- The smallest image is the most heavily smoothed

Reduction

$$g_l(i, j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m, n) g_{l-1}(2i + m, 2j + n).$$

GAUSSIAN PYRAMID



g_0 = IMAGE

g_L = REDUCE $[g_{L-1}]$

- Usually the mask is Gaussian-like function
- Burt & Adelson derivation: for simplicity consider *w* a separable function, set of weights:

$$w(m,n) = \hat{w}(m)\hat{w}(n)$$

$$\sum_{m=-2}^{2} \hat{w}(m) = 1$$ (1) Normalization

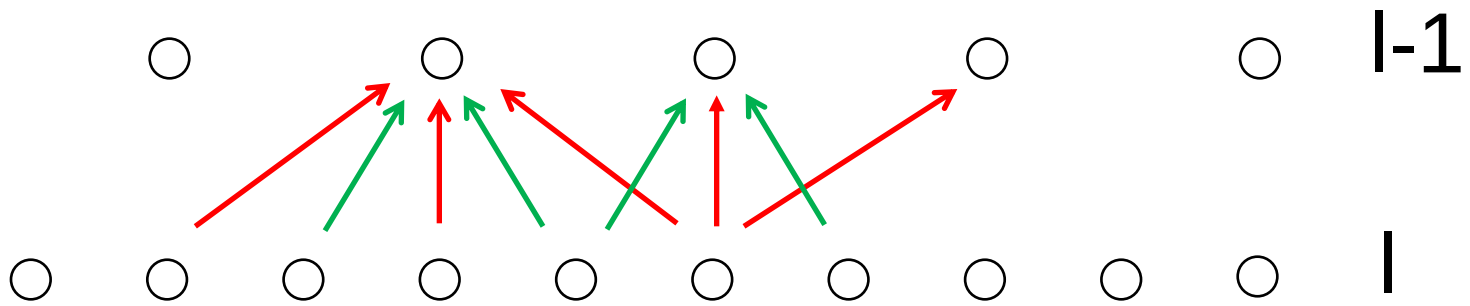
$$\hat{w}(i) = \hat{w}(-i) \quad i = 0,1,2$$ (2) Symmetry

$$\hat{w}(0) = a$$

$$\hat{w}(-1) = \hat{w}(1) = b$$

$$\hat{w}(-2) = \hat{w}(2) = c$$

- (3) Equal contribution: all nodes should contribute equally to the next layer



$$a + 2c = 2b$$

- (1)+(2)+(3) are satisfied when:

$$\hat{w}(0) = a$$
$$\hat{w}(-1) = \hat{w}(1) = 1/4$$
$$\hat{w}(-2) = \hat{w}(2) = 1/4 - a/2$$

- *a* is a free parameter that determines the shape of the weighting function

Equivalent weighting functions

- Iterative pyramid generation is equivalent to convolving the original image with a set of equivalent weighting functions:

$$g_l = h_l \oplus g_0$$

or

$$g_l(i,j) = \sum_{m=-M_l}^{Ml} \sum_{n=-Ml}^{Ml} h_l(m,n) g_0(i2^l + m, j2^l + n)$$

…where Ml, the size of the equivalent weighting function, doubles from one level to the next

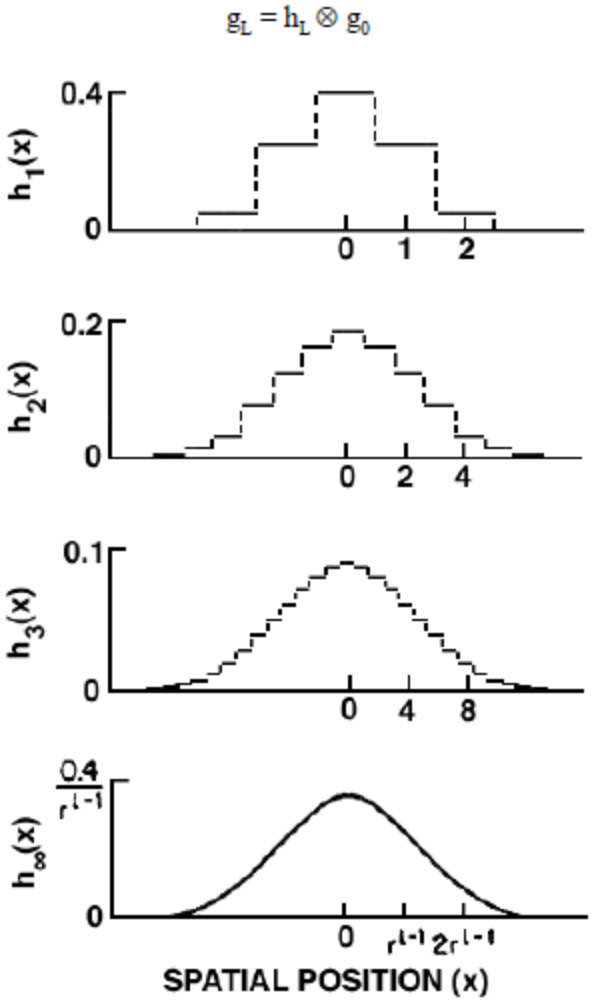


Fig. 2. The equivalent weighting functions $h_l(x)$ for nodes in levels 1, 2, 3, and infinity of the Gaussian pyramid. Note that axis scales have been adjusted by factors of 2 to aid comparison Here the parameter a of the generating kernel is 0.4, and the resulting equivalent weighting functions closely resemble the Gaussian probability density functions.

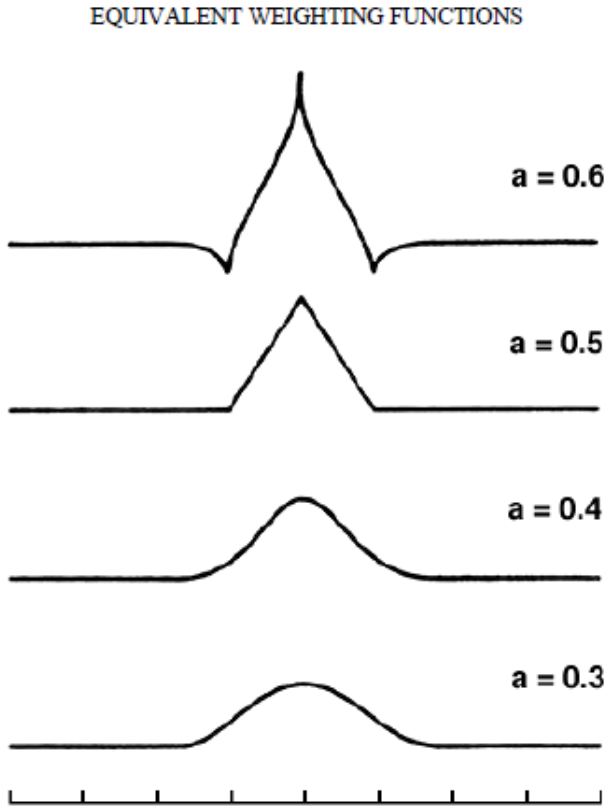


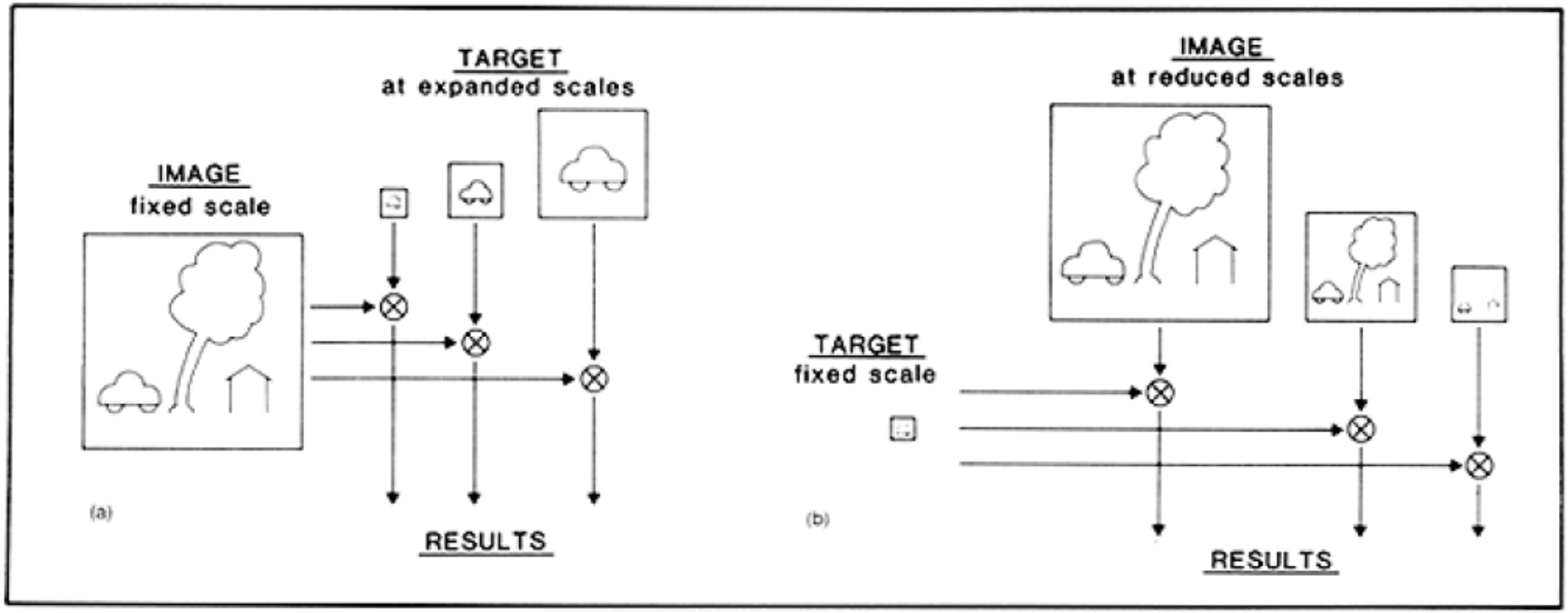
Fig. 3. The shape of the equivalent weighting function depends on the choice of parameter a. For $a = 0.5$, the function is triangular; for $a = 0.4$ it is Gaussian-like, and for $a = 0.3$ it is broader than Gaussian. For $a = 0.6$ the function is trimodal.

512 256 128 64 32 16 8

from: David Forsythhttp://www.cs.berkeley.edu/~daf/

Applications (1)

- Search over scale – objects can be represented as small image patterns; if we want to search across different *scales* we can search across the layers of the gaussian pyramid; bigger objects will be found in the coarser scale layers, smaller objects will be found in the finer scales


TARGET
at expanded scales
IMAGE
fixed scale
(a)
RESULTS
IMAGE
at reduced scales
TARGET
fixed scale
(b)
RESULTS

Applications (2)

- Spatial search: often we have a point in one image and want to find the same point in another image (example: stereo vision or motion detection). This can be achieved more efficiently if we first start to search the object in the coarser layers (using coarser template), and then refine the match by searching in the finer layers (*coarse-to-fine matching)*

Applications (3)

- Feature tracking: features (e.g. edges) found at coarse levels are associated with high-contrast image events (low contrast patches are easily lost during consequent smoothing); at fine scales there are probably many more features with lower contrast. A common strategy for improving a set of features obtained at a fine scale is to track features to coarser scales and accept only the fine scale features that are identifiable at coarser scales (*feature tracking)*

Laplacian pyramid

- The Gaussian pyramid contains low-pass version of the original image
- It is possible to obtain a band-pass version of the pyramid:

$$L_0(i,j) = g_0(i,j) - g_1(i,j)$$

- Repeat…

$$L_2(i,j) = g_1(i,j) - g_2(i,j)$$

$$\Rightarrow L_0, L_1, ..., L_n$$

- This pyramid is called “Laplacian” because each layer approximates the Laplacian operator

L_0

L_1

L_2

L_3

L_4

Fig. 4a. *The Laplacian pyramid. Each level of this band-pass pyramid represents the difference between successive levels of the Gaussian pyramid.*

Application: image coding

- Values in the layers L of the pyramid tend to be closer to zero, they can be coded with fewer bits
- Decompose image I into a set of images L0, L1, … Ln
- Transmit Ln, Ln-1, … L1, L0
- Incrementally reconstruct I whenever each layer is received

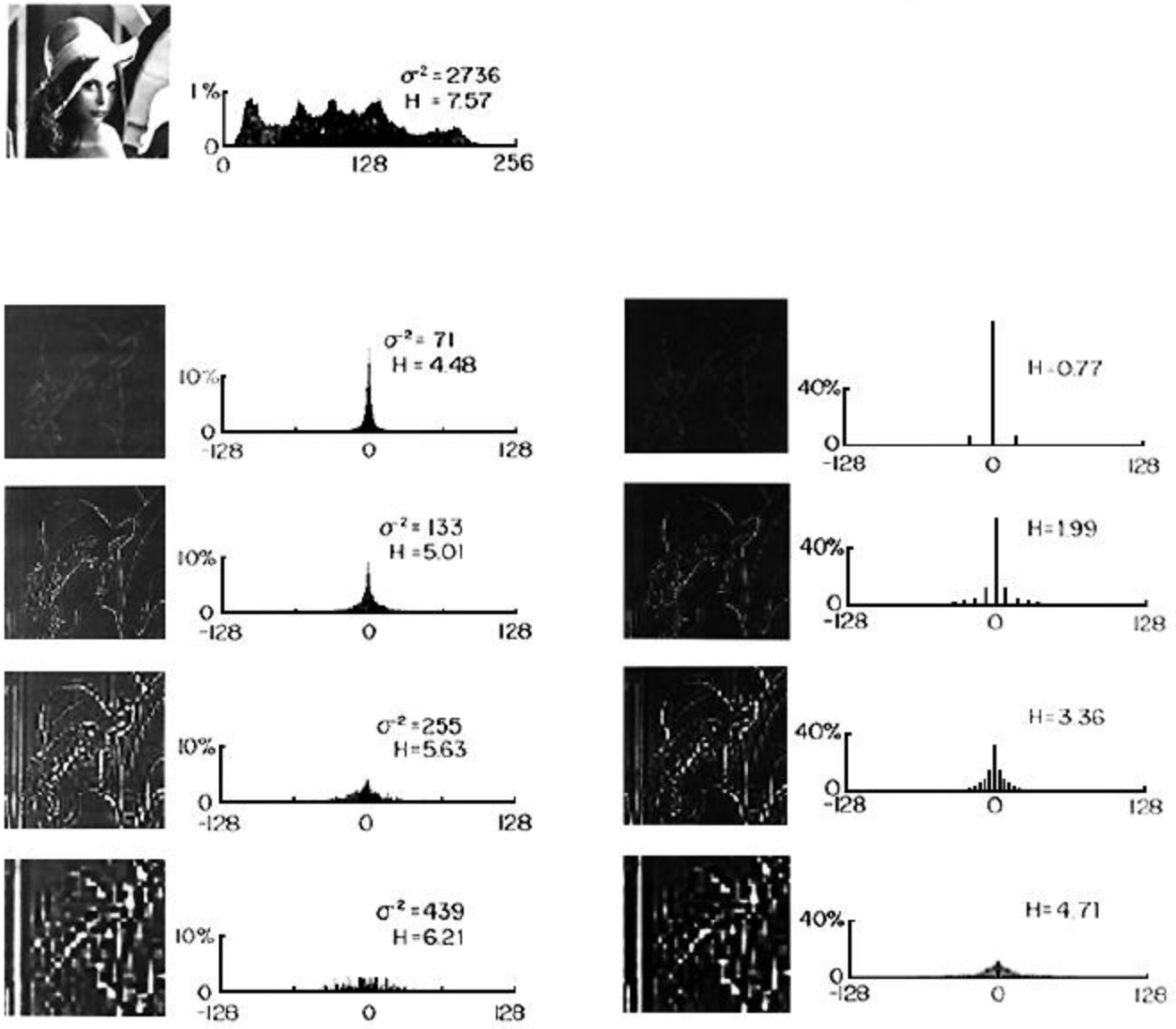


FIG 6. The distribution of pixel gray level values at various stages of the encoding process. The histogram of the original image is given in (a). (b)-(e) give histograms for levels 0-3 of the Laplacian pyramid with generating parameter a=0.6. Histograms following quantization at each level are shown in (f)-(i). Note that pixel values in the Laplacian pyramid are concentrateed near zero, permitting data compression through shortened and varable length code words. Substantial further reduction is realized through quantization (particularly at low pyramid levels) and reduced sample density (particularly at high pyramid levels).

Encoding

- Define function “EXPAND”, tansforming a MxN image into a 2Mx2N image

$$g_l = EXPAND(g_{l+1})$$

$$g_l(i,j) = 4\sum_{m=-2}^{2}\sum_{n=-2}^{2} w(m,n)\cdot g_{l+1}\left(\frac{i-m}{2},\frac{j-n}{2}\right)$$

build sequence $L_0, L_1, \ldots, L_N$

$$L_0 = g_0 - EXPAND(g_1)$$

$$L_1 = g_1 - EXPAND(g_2)$$

...

$$L_{N-1} = g_{N-1} - EXPAND(g_N)$$

$$L_N = g_N$$

Decoding

from $L_N, L_{N-1}, .., L_0$

$g_N = L_N$

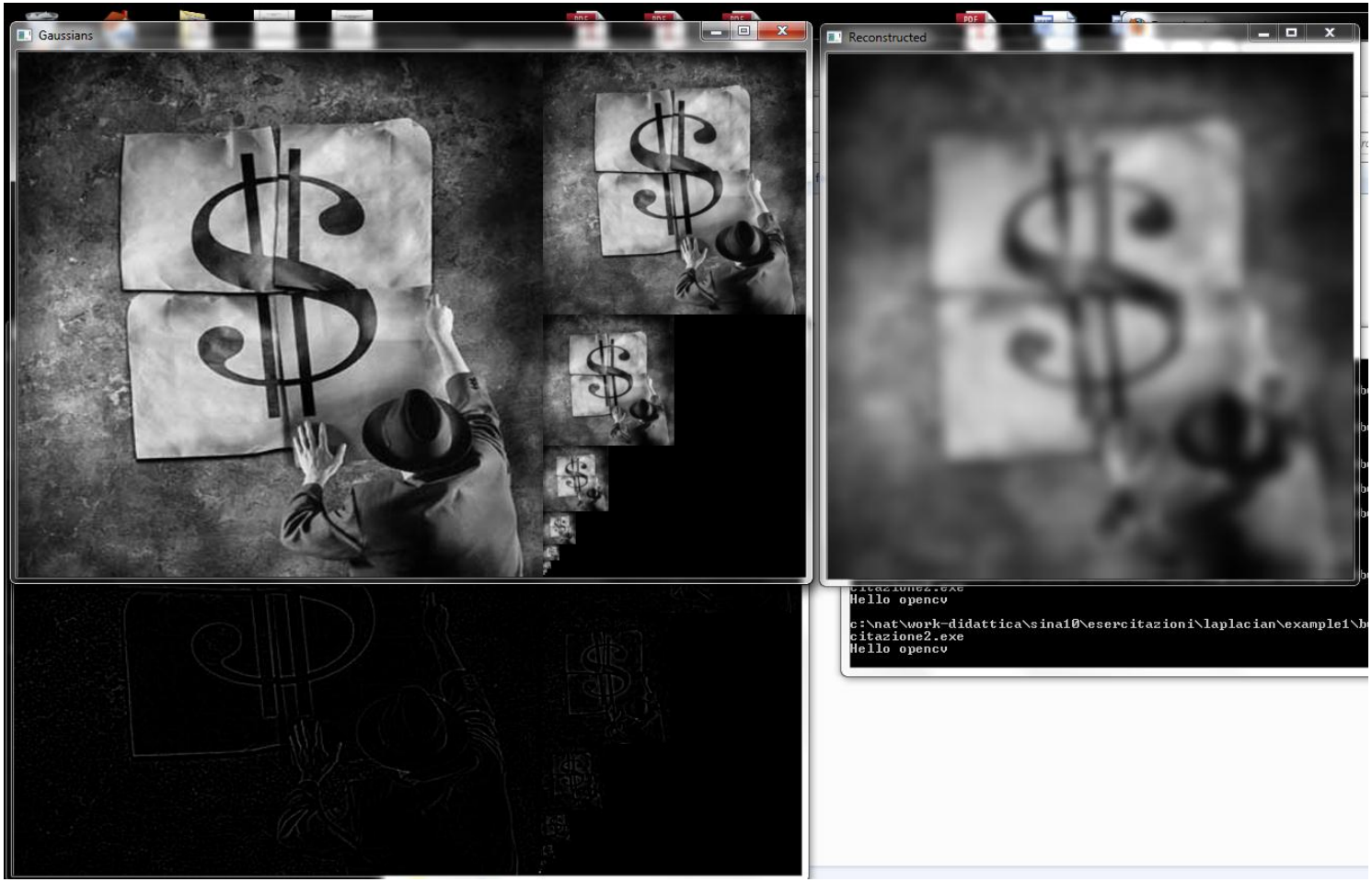
$g_{N-1} = L_{N-1} + EXPAND(g_N)$

...

$g_0 = L_0 + EXPAND(g_1)$

Exercise

- Implement image decomposition and reconstruction using the Laplacian pyramid



void cvPyrDown(const CvArr* src, CvArr* dst, int filter=CV_GAUSSIAN_5x5);

src: The source image.

dst: The destination image, should have 2x smaller width and height than the source. filterType of the filter used for convolution; only CV_GAUSSIAN_5x5 is currently supported. The function cvPyrDown performs downsampling step of Gaussian pyramid decomposition. First it convolves source image with the specified filter and then downsamples the image by rejecting even rows and columns.

void cvPyrUp(const CvArr* src, CvArr* dst, int filter=CV_GAUSSIAN_5x5);

src: The source image.

dst: The destination image, should have 2x smaller width and height than the source. filterType of the filter used for convolution; only CV_GAUSSIAN_5x5 is currently supported. The function cvPyrUp performs up-sampling step of Gaussian pyramid decomposition. First it upsamples the source image by injecting even zero rows and columns and then convolves result with the specified filter multiplied by 4 for interpolation. So the destination image is four times larger than the source image.

```
IplImage **pyrGauss;
IplImage **pyrLap;

pyrGauss=allocP(img->width, img->height, K, IPL_DEPTH_8U);
pyrLap=allocP(img->width, img->height, K, IPL_DEPTH_16S);
```

```
IplImage **allocP(int W, int H, int K, unsigned int
DEPTH)
{
    IplImage **ret=new IplImage*[K];

    ret[0]=cvCreateImage(cvSize(W, H), DEPTH, 1);

    for(int k=1;k<K;k++)
    {
        ret[k]=cvCreateImage(cvSize(ret[k-1]->width/2,
ret[k-1]->height/2), DEPTH, 1);
    }
    return ret;
}
```

```
void destroyP(IplImage **p, int K)
{
    for(int k=0; k<K; k++)
    {
        cvReleaseImage(&p[k]);
    }

    delete [] p;
}
```