Pyramids

Lecture-7
Contents

• Gaussian and Laplacian Pyramids
  – Reduce
  – Expand
• Applications of Laplacian pyramids
  – Image compression
  – Image composting
• Optical flow using Pyramids
  – interpolation
Pyramids

• Very useful for representing images.
• Pyramid is built by using multiple copies of image.
• Each level in the pyramid is 1/4 of the size of previous level.
• The lowest level is of the highest resolution.
• The highest level is of the lowest resolution.
Pyramid

Level 1: 1x1

Level 2: 2x2

Level 3: 4x4

Level 4: 8x8

Level 10: 512x512
The Laplacian Pyramid as a Compact Image Code

PETER J. BURT, MEMBER, IEEE, AND EDWARD H. ADelson

Abstract—We describe a technique for image encoding in which local operators of many scales but identical shape are used as the basic functions. The representation differs from established techniques in that the code elements are localized in spatial frequency as well as in space.

Pixel-to-pixel correlations are first removed by subtracting a low-pass filtered copy of the image from itself. The result is a set of difference images, or error images, which have low correlation and entropy, and the low-pass filtered image may be represented at a reduced sample density. Further data compression is achieved by quantizing the difference images. These steps are then repeated to compress the low-pass image. Iteration of the process at appropriately expanded scales generates a pyramid data structure. The encoding process is equivalent to sampling the image with Laplacian operators of many scales. Thus, the code tends to enhance salient image features. A further advantage of the present code is that it is well suited for many image analysis tasks as well as for image compression. Fast algorithms are described for coding and decoding.

INTRODUCTION

A COMMON characteristic of images is that neighboring points are highly correlated. To represent the image directly in terms of the point values is therefore inefficient: most of the encoded information is redundant. The first task in designing an efficient, compressed code is to find a representation which, in effect, decorrelates the image pixels. This has been achieved through predictive and transform techniques (cf. [7], [10] for recent reviews).

In predictive coding, pixels are encoded sequentially in a raster format. However, prior to encoding each pixel, its value is predicted from previously coded pixels in the same and preceding raster lines. The predicted pixel value, which represents redundant information, is subtracted from the actual pixel value, and only the difference, or prediction error, is encoded. Since only previously encoded pixels are used in predicting each pixel's value, this process is said to be causal.

Restriction to causal prediction facilitates decoding: to decode a given pixel, its predicted value is recomputed from already decoded neighboring pixels, and added to the stored prediction error.

Noncausal prediction, based on a symmetric neighborhood centered at each pixel, should yield more accurate prediction and, hence, greater data compression. However, this approach does not permit simple sequential coding. Noncausal approaches to image coding typically involve image transforms, or the solution to large sets of simultaneous equations. Rather than encoding pixels sequentially, such techniques encode them all at once, or by blocks.

Both predictive and transform techniques have advantages. The former is relatively simple to implement and is readily adapted to local image characteristics. The latter generally provides greater data compression, but at the expense of considerably greater computation.

Here we shall describe a new technique for removing image correlation which combines features of predictive and transform methods. The technique is noncausal, yet computations are relatively simple and local.

The predicted value for each pixel is computed as a local weighted average, using a unimodal Gaussian-like (or related trilinear) weighting function centered on the pixel itself. The predicted values for all pixels are first obtained by convolving this weighting function with the image. The result is a low-pass filtered image which is then subtracted from the original.

Let \( g(x) \) be the original image, and \( b(x) \) be the result of applying an appropriate low-pass filter to \( g(x) \). The prediction error \( L(x) = g(x) - b(x) \) is then given by

\[
L(x) = g(x) - b(x)
\]

Rather than encode \( g(x) \), we encode \( L(x) \) and \( b(x) \). This results in a far lower compression because \( L(x) \) is largely decorrelated, and so may be represented pixel by pixel with many fewer bits than \( g(x) \); and \( b(x) \) is low-pass filtered, and so may be encoded at a reduced sample rate.

Further data compression is achieved by iterating this process. The relabeled image \( g(x) \) is itself low-pass filtered to yield \( L_1(x) \), and a second error image is obtained: \( L_2(x) = g(x) - b(x) \). By repeating these steps several times we obtain a sequence of two-dimensional arrays \( L_1(x), L_2(x), \ldots, L_n(x) \). In our implementation each is smaller than its predecessor by a scale factor of \( 1/2 \) due to reduced sample density. If we now imagine these arrays stacked one above another, the result is a tapering pyramid data structure. The values at each node in the pyramid represent the difference between two Gaussian-like or related functions convolved with the original image. The difference between these two functions is similar to the Laplacian operators commonly used in image enhancement [13]. Thus, we refer to the new compressed image representation as the Laplacian pyramid code.

The encoding scheme outlined above will be practical only if required filtering computations can be performed with an efficient algorithm. A suitable fast algorithm has recently been developed [2] and will be described in the next section.
Gaussian Pyramids (reduce)

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

Level \( l \)

\[ g_l = REDUCE[g_{l-1}] \]
Convolution

\[
h(x, y) = f(x-1, y+1) * g(-1, 1) + f(x, y+1) * g(0, 1) + f(x+1, y+1) * g(1, 1) +
           f(x-1, y) * g(-1, 0) + f(x, y) * g(0, 0) + f(x+1, y) * g(1, 0) +
           f(x-1, y-1) * g(-1, -1) + f(x, y-1) * g(0, -1) + f(x+1, y-1) * g(1, -1)
\]
Reduce (1D)

\[ g_l(i) = \sum_{m=-2}^{2} \hat{w}(m) g_{l-1}(2i+m) \]

\[ g_l(2) = \hat{w}(-2)g_{l-1}(4-2) + \hat{w}(-1)g_{l-1}(4-1) + \hat{w}(0)g_{l-1}(4) + \hat{w}(1)g_{l-1}(4+1) + \hat{w}(2)g_{l-1}(4+2) \]

\[ g_l(2) = \hat{w}(-2)g_{l-1}(2) + \hat{w}(-1)g_{l-1} \hat{w}(3) + \hat{w}(0)g_{l-1}(4) + \hat{w}(1)g_{l-1}(5) + \hat{w}(2)g_{l-1}(6) \]
Reduce

Gaussian Pyramid

\[ g_0 = \text{IMAGE} \]

\[ g_1 = \text{REDUCE}[g_{L-1}] \]
Gaussian Pyramids (expand)

\[
g_{l,n}(i, j) = \sum_{p=-2}^{2} \sum_{q=-2}^{2} w(p, q) g_{l,n-1}\left(\frac{i-p}{2}, \frac{j-q}{2}\right)
\]

\[
g_{l,n} = \text{EXPAND}[g_{l,n-1}]
\]
Expand (1D)

\[ g_{l,n}(i) = \sum_{p=-2}^{2} \hat{w}(p) g_{l,n-1}\left(\frac{i-p}{2}\right) \]

\[ g_{l,n}(4) = \hat{w}(-2) g_{l,n-1}\left(\frac{4+2}{2}\right) + \hat{w}(-1) g_{l,n-1}\left(\frac{4+1}{2}\right) + \]

\[ \hat{w}(0) g_{l,n-1}\left(\frac{4}{2}\right) + \hat{w}(1) g_{l,n-1}\left(\frac{4-1}{2}\right) + \hat{w}(2) g_{l,n-1}\left(\frac{4-2}{2}\right) \]

\[ g_{l,n}(4) = \hat{w}(-2) g_{l,n-1}(3) + \hat{w}(0) g_{l,n-1}(2) + \hat{w}(2) g_{l,n-1}(1) \]

\[ g_{l,n}(4) = g_{l,n-1}(3)c + g_{l,n-1}(2)a + g_{l,n-1}(1)b \]
Expand (1D)

\[ g_{l,n}(i) = \sum_{p=-2}^{2} \hat{w}(p) g_{l,n-1}\left(\frac{i-p}{2}\right) \]

\[ g_{l,n}(3) = \hat{w}(-2) g_{l,n-1}\left(\frac{3+2}{2}\right) + \hat{w}(-1) g_{l,n-1}\left(\frac{3+1}{2}\right) + \]

\[ \hat{w}(0) g_{l,n-1}\left(\frac{3}{2}\right) + \hat{w}(1) g_{l,n-1}\left(\frac{3-1}{1}\right) + \hat{w}(2) g_{l,n-1}\left(\frac{3-2}{2}\right) \]

\[ g_{l,n}(3) = \hat{w}(-1) g_{l,n-1}(2) + \hat{w}(1) g_{l,n-1}(1) \]

\[ g_{l,n}(3) = bg_{l,n-1}(2) + bg_{l,n-1} \]
Expand

\[ g_{l,n}(4) = g_{l,n-1}(3)c + g_{l,n-1}(2)a + + g_{l,n-1}(1)b \]

**Gaussian Pyramid**

\[ g_{l,1} = \text{EXPAND}[g_{1} \ ] \]
$g_{l,n}(3) = bg_{l,n-1}(2) + bg_{l,n-1}$

Gaussian Pyramid

$g_{1,1} = \text{EXPAND}[g_1]$
Convolution Mask

\[ [w(-2), w(-1), w(0), w(1), w(2)] \]
Convolution Mask

- Separable

\[ w(m, n) = \hat{w}(m)\hat{w}(n) \]

- Symmetric

\[ \hat{w}(i) = \hat{w}(-i) \]

[\[c, b, a, b, c\]]
Convolution Mask

• The sum of mask should be 1.

\[ a + 2b + 2c = 1 \]

• All nodes at a given level must contribute the same total weight to the nodes at the next higher level.

\[ a + 2c = 2b \]
\[ a + 2c = 2b \]
\[ a + 2b + 2c = 1 \]

\[ b = \frac{1}{4} \]

\[ c = \frac{1}{2} (2b - a) \]
\[ c = \frac{1}{4} - \frac{a}{2} \]
Convolution Mask

\[ \hat{w}(0) = a \]

\[ \hat{w}(-1) = \hat{w}(1) = \frac{1}{4} \]

\[ \hat{w}(-2) = \hat{w}(2) = \frac{1}{4} - \frac{a}{2} \]

\(a=.4\) GAUSSIAN, \(a=.5\) TRINGULAR
Triangular

a=.4 GAUSSIAN, a=.5 TRINGULAR
Approximate Gaussian

\[ a = 0.4 \text{ GAUSSIAN, } a = 0.5 \text{ TRINGULAR} \]
\( \hat{w}(0) = a \)

\( \hat{w}(-1) = \hat{w}(1) = \frac{1}{4} \)

\( \hat{w}(-2) = \hat{w}(2) = \frac{1}{4} - \frac{a}{2} \)
Gaussian

\[ g(x) = e^{\frac{-x^2}{2\sigma^2}} \]

<table>
<thead>
<tr>
<th>x</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>g(x)</td>
<td>.011</td>
<td>.13</td>
<td>.6</td>
<td>1</td>
<td>.6</td>
<td>.13</td>
<td>.011</td>
</tr>
</tbody>
</table>
Separability
Algorithm

• Apply 1-D mask to alternate pixels along each row of image.
• Apply 1-D mask to each pixel along alternate columns of resultant image from previous step.
Gaussian Pyramid
Laplacian Pyramids
Pierre-Simon Laplace (1749–1827)

- Mathematician and astronomer
- French Newton
  - Laplacian
  - Laplace Transform
  - Celestial mechanics
  - Spherical harmonics
- On 15 March 1788 at the age of thirty-nine, Laplace married Marie-Charlotte de Courty de Romanges, a pretty eighteen-and-a-half-year-old girl from a good family in Besançon.
Laplacian Pyramids

- Similar to edge detected images.
- Most pixels are zero.
- Can be used for image compression.

\[
L_1 = g_1 - \text{EXPAND}[g_2]
\]

\[
L_2 = g_2 - \text{EXPAND}[g_3]
\]

\[
L_3 = g_3 - \text{EXPAND}[g_4]
\]
Fig. 5. First four levels of the Gaussian and Laplacian pyramid. Gaussian images, upper row, were obtained by expanding pyramid arrays (Fig. 4) through Gaussian interpolation. Each level of the Laplacian pyramid is the difference between the corresponding and next higher levels of the Gaussian pyramid.
Coding using Laplacian Pyramid

• Compute Gaussian pyramid

\[ g_1, g_2, g_3, g_4 \]

• Compute Laplacian pyramid

\[ L_1 = g_1 - \text{EXPAND}[g_2] \]
\[ L_2 = g_2 - \text{EXPAND}[g_3] \]
\[ L_3 = g_3 - \text{EXPAND}[g_4] \]
\[ L_4 = g_4 \]

• Code Laplacian pyramid
Decoding using Laplacian pyramid

• Decode Laplacian pyramid.
• Compute Gaussian pyramid from Laplacian pyramid.

\[
\begin{align*}
g_4 &= L_4 \\
g_3 &= \text{EXPAND}[g_4] + L_3 \\
g_2 &= \text{EXPAND}[g_3] + L_2 \\
g_1 &= \text{EXPAND}[g_2] + L_1
\end{align*}
\]

• \( g_1 \) is reconstructed image.
Laplacian Pyramid

Fig. 5. First four levels of the Gaussian and Laplacian pyramids. Gaussian images, upper row, were obtained by expanding pyramid arrays (Fig. 4) through Gaussian interpolation. Each level of the Laplacian pyramid is the difference between the corresponding and next higher levels of the Gaussian pyramid.
Image Compression (Entropy)

Bits per pixel

7.6
Image Compression

1.58

.73
Combining Apple & Orange
Combining Apple & Orange
Algorithm

• Generate Laplacian pyramid Lo of orange image.
• Generate Laplacian pyramid La of apple image.
• Generate Laplacian pyramid Lc by
  – copying left half of nodes at each level from apple and
  – right half of nodes from orange pyramids.
• Reconstruct combined image from Lc.
Reading Material

• http://ww-bcs.mit.edu/people/adelson/papers.html

• Fundamental of Computer Vision, Section 4.5.
Lucas Kanade with Pyramids

- Compute ‘simple’ LK optical flow at highest level
- At level $i$
  - Take flow $u_{i-1}$, $v_{i-1}$ from level $i-1$
  - bilinear interpolate it to create $u_i^*$, $v_i^*$ matrices of twice resolution for level $i$
  - multiply $u_i^*$, $v_i^*$ by 2
  - compute $f_t, f_x, f_y$ using masks centered at $(x,y)$ and $(x+u_i^*, y+v_i^*)$
  - Apply LK to get $u_i'(x, y)$, $v_i'(x, y)$ (the correction in flow)
  - Add corrections $u_i'$, $v_i'$, i.e. $u_i = u_i^* + u_i'$, $v_i = v_i^* + v_i'$. 
Pyramids

\[ u_i = u_i^* + u_i', v_i = v_i^* + v_i' \]
Interpolation

\[ u = 1 \]
\[ u^* = 3 \]
\[ v = 1 \]
\[ v^* = 3 \]
1-D Interpolation

\[ y = mx + c \]
\[ f(x) = mx + c \]
2-D Interpolation

\[ f(x, y) = a_1 + a_2 x + a_3 y + a_4 xy \]

Bilinear

\[ X(3,6) \quad X(4,6) \]

\[ X(3,5) \quad X(4,5) \]
Bi-linear Interpolation

Four nearest points of \((x,y)\) are:

\[(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)\]

\[(3,5), (4,5), (3,6), (4,6)\]

\[x = \text{int}(x)\]

\[y = \text{int}(y)\]

\[x = x + 1\]

\[y = y + 1\]
Bi-linear Interpolation

\[ f(x, y) = \varepsilon_x \varepsilon_y f(x, y) + \varepsilon_x \varepsilon_y f(x, y) + \varepsilon_x \varepsilon_y f(x, y) + \varepsilon_x \varepsilon_y f(x, y) \]

Homework

\[ \varepsilon_x = x - x \]
\[ \varepsilon_y = y - y \]

\[ \varepsilon_x = x - x = 4 - 3.2 = .8 \]
\[ \varepsilon_y = y - y = 6 - 5.6 = .4 \]

\[ \varepsilon_x = x - x = 3.2 - 2 = .2 \]
\[ \varepsilon_y = y - y = 5.6 - 5 = .6 \]
Lucas-Kanade without pyramids

Fails in areas of large motion
Lucas-Kanade with Pyramids