ECE 5973: Lecture 5
Image pyramids

Samuel Cheng

School of ECE
University of Oklahoma

Spring, 2019

Table of Contents

1. Introduction
2. Gaussian pyramid
3. Laplacian pyramids
4. Applications

Reference

532 IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. COM-31, NO. 4, APRIL 1983

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 serve as the basis 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 the image itself. The result is a net data compression since the difference, or error, image has low variance and entropy, and the low-pass filtered image may represented at reduced sample density. Further data compression is achieved by quantizing the difference image. 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 pixels are highly correlated. To represent the image directly in terms of the pixel 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 through transform techniques (cf. [9], [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 trimodal) 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 $g_0(ij)$ be the original image, and $g_1(ij)$ be the result of applying an appropriate low-pass filter to g_0. The prediction error $L_0(ij)$ is then given by

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

Rather than encode g_0, we encode L_0 and g_1. This results in a net data compression because a) L_0 is largly decorrelated, and so may be represented pixel by pixel with many fewer bits than g_0, and b) g_1 is low-pass filtered, and so may be encoded at a reduced sample rate.

Further data compression is achieved by iterating this process. The reduced image g_1 is itself low-pass filtered to yield g_2 and a second error image is obtained: $L_2(ij)=g_1(ij)-g_2(ij)$. By repeating these steps several times we obtain a sequence of two-dimensional arrays $L_0, L_1, L_2, \ldots, L_n$. 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 value at each node in the pyramid represents 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 proposed compressed image representation as the Laplacian-pyramid code.

The coding 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.

Paper approved by the Editor for Signal Processing and Communication Electronics of the IEEE Communications Society for publication after presentation in part at the Conference on Pattern Recognition and Image Processing, Dallas, TX, 1981. Manuscript received April 12, 1982; revised July 21, 1982. This work was supported in part by the National Science Foundation under Grant MCS-79-23422 and by the National Institutes of Health under Postdoctoral Training Grant EY07003.

P. J. Burt is with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12181.

E. H. Adelson is with the RCA David Sarnoff Research Center, Princeton, NJ 08540.

cited: 7343

From Wikipedia

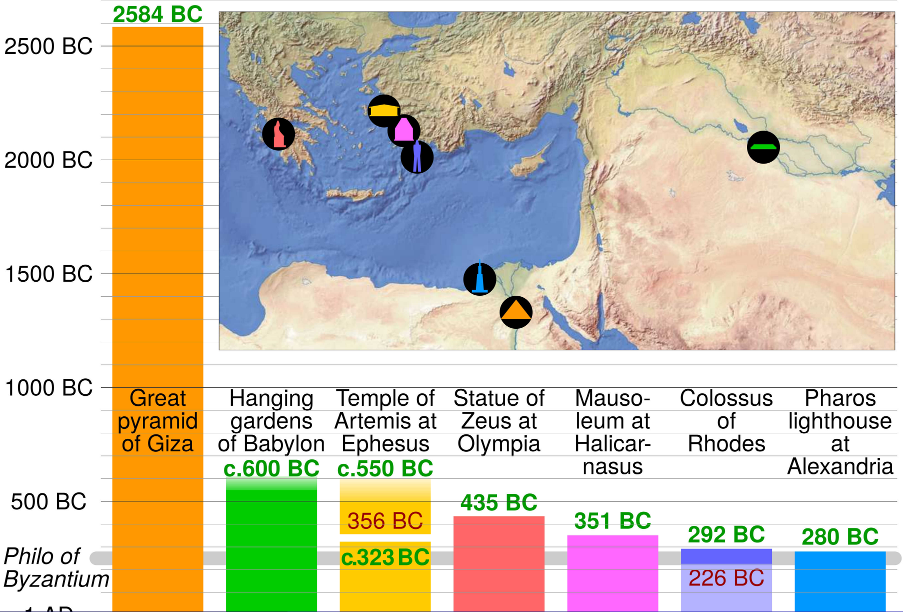
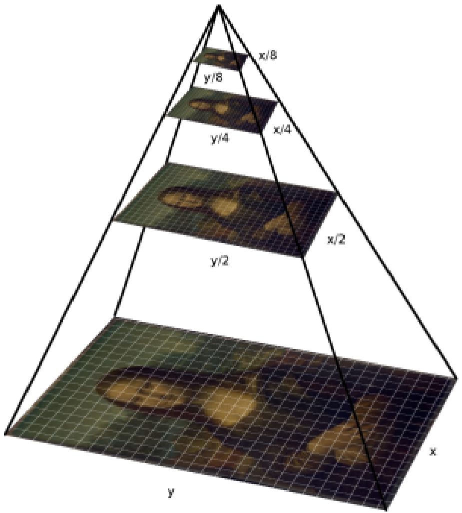


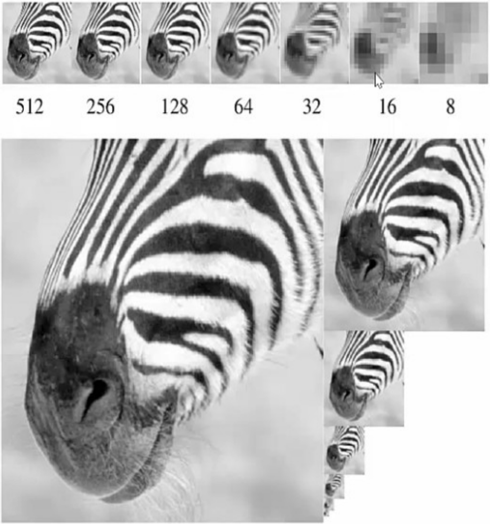
Image pyramid

- Very useful for representing images
- Pyramid is built as multiple resolution approximations of a same image
- Each level in the pyramid is 1/4 of the size of the previous level
- Lowest level has highest resolution
- Highest level has lowest resolution



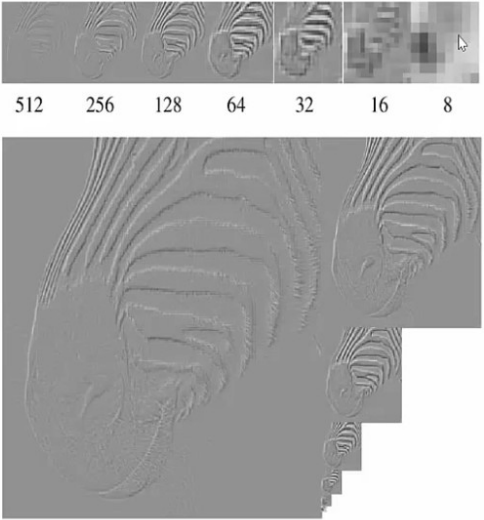
https://www.pyimagesearch.com/2015/03/16/image-pyramids-with-python-and-opencv/

Gaussian pyramid



Source: Forsyth

Laplacian pyramid



Source: Forsyth

Things to learn today

- Gaussian and Laplacian pyramid
  - Reduce
  - Expand
- Applications of Laplacian pyramid
  - Image compression
  - Image composting

Reduce operation

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

$$\underbrace{g_l(i,j)}_{l\text{-level}} = \sum_{m=-2}^{2}\sum_{n=-2}^{2} \tilde{w}(m,n)\underbrace{g_{l-1}(2i-m, 2j-n)}_{l-1\text{-level}}$$

Reduce operation

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

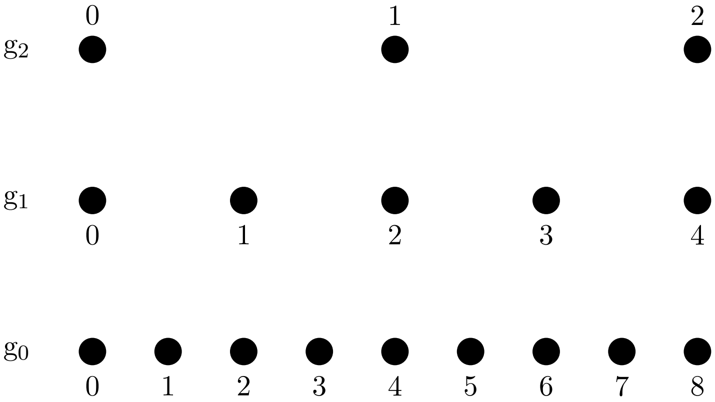
$$\underbrace{g_l(i,j)}_{l\text{-level}} = \sum_{m=-2}^{2}\sum_{n=-2}^{2} \tilde{w}(m,n)\underbrace{g_{l-1}(2i-m, 2j-n)}_{l-1\text{-level}}$$

Remark

Note that it is different from convolution that we skip every one sample per dimension for the input

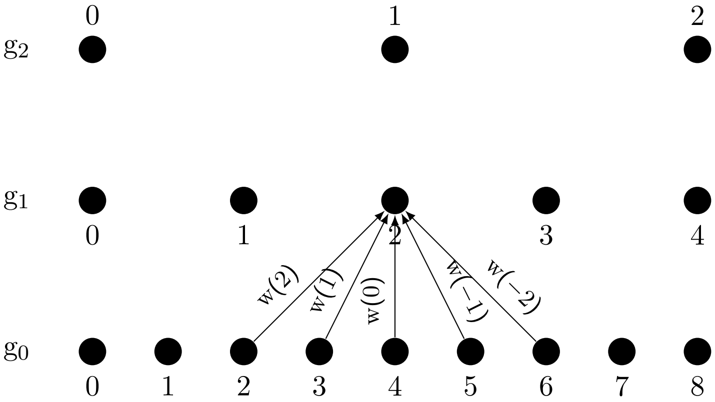
1-D case

$$g_l(i) = \sum_{m=-2}^{2} w(m) g_{l-1}(2i - m)$$



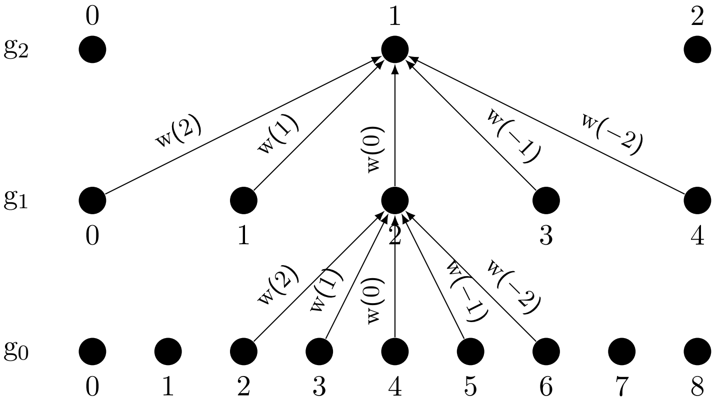
1-D case

$$g_l(i) = \sum_{m=-2}^{2} w(m) g_{l-1}(2i - m)$$



1-D case

$$g_l(i) = \sum_{m=-2}^{2} w(m) g_{l-1}(2i - m)$$



Expand operation

$$\hat{g}_l = \mathrm{EXPAND}[\hat{g}_{l-1}]$$

$$\hat{g}_l(i,j) = \sum_{n=-2}^{2} \sum_{m=-2}^{2} \tilde{w}(m,n)\hat{g}_{l-1}\left(\frac{i-n}{2}, \frac{j-m}{2}\right)$$

Remark

EXPAND is approximately the inverse operation of REDUCE

1-D case

$$\hat{g}_l(i) = \sum_{m=-2}^{2} w(m)\hat{g}_{l-1}\left(\frac{i-m}{2}\right)$$

1-D case

$$\hat{g}_l(i) = \sum_{m=-2}^{2} w(m)\hat{g}_{l-1}\left(\frac{i-m}{2}\right)$$

$$\begin{aligned}\hat{g}_l(2) = {} & w(-2)\hat{g}_{l-1}\left(\frac{2+2}{2}\right) + w(-1)\hat{g}_{l-1}\left(\frac{2+1}{2}\right) \\ & + w(0)\hat{g}_{l-1}\left(\frac{2}{2}\right) + w(1)\hat{g}_{l-1}\left(\frac{2-1}{2}\right) + w(2)\hat{g}_{l-1}\left(\frac{2-2}{2}\right)\end{aligned}$$

1-D case

$$\hat{g}_l(i) = \sum_{m=-2}^{2} w(m)\hat{g}_{l-1}\left(\frac{i-m}{2}\right)$$

$$\hat{g}_l(2) = w(-2)\hat{g}_{l-1}\left(\frac{2+2}{2}\right) + \cancel{w(-1)\hat{g}_{l-1}\left(\tfrac{2+1}{2}\right)} + w(0)\hat{g}_{l-1}\left(\frac{2}{2}\right) + \cancel{w(1)\hat{g}_{l-1}\left(\tfrac{2-1}{2}\right)} + w(2)\hat{g}_{l-1}\left(\frac{2-2}{2}\right)$$

1-D case

$$\hat{g}_l(i) = \sum_{m=-2}^{2} w(m)\hat{g}_{l-1}\left(\frac{i-m}{2}\right)$$

$$\begin{aligned}\hat{g}_l(2) &= w(-2)\hat{g}_{l-1}\left(\frac{2+2}{2}\right) + \cancel{w(-1)\hat{g}_{l-1}\left(\tfrac{2+1}{2}\right)} \\ &\quad + w(0)\hat{g}_{l-1}\left(\frac{2}{2}\right) + \cancel{w(1)\hat{g}_{l-1}\left(\tfrac{2-1}{2}\right)} + w(2)\hat{g}_{l-1}\left(\frac{2-2}{2}\right) \\ &= w(-2)\hat{g}_{l-1}(2) + w(0)\hat{g}_{l-1}(1) + w(2)\hat{g}_{l-1}(0)\end{aligned}$$

1-D case

$$\hat{g}_l(i) = \sum_{m=-2}^{2} w(m)\hat{g}_{l-1}\left(\frac{i-m}{2}\right)$$

$$\begin{aligned}\hat{g}_l(2) &= w(-2)\hat{g}_{l-1}\left(\frac{2+2}{2}\right) + \cancel{w(-1)\hat{g}_{l-1}\left(\tfrac{2+1}{2}\right)} \\ &\quad + w(0)\hat{g}_{l-1}\left(\frac{2}{2}\right) + \cancel{w(1)\hat{g}_{l-1}\left(\tfrac{2-1}{2}\right)} + w(2)\hat{g}_{l-1}\left(\frac{2-2}{2}\right) \\ &= w(-2)\hat{g}_{l-1}(2) + w(0)\hat{g}_{l-1}(1) + w(2)\hat{g}_{l-1}(0)\end{aligned}$$

$$\begin{aligned}\hat{g}_l(5) &= w(-2)\hat{g}_{l-1}\left(\frac{5+2}{2}\right) + w(-1)\hat{g}_{l-1}\left(\frac{5+1}{2}\right) + w(0)\hat{g}_{l-1}\left(\frac{5}{2}\right) \\ &\quad + w(1)\hat{g}_{l-1}\left(\frac{5-1}{2}\right) + w(2)\hat{g}_{l-1}\left(\frac{5-2}{2}\right)\end{aligned}$$

1-D case

$$\hat{g}_l(i) = \sum_{m=-2}^{2} w(m)\hat{g}_{l-1}\left(\frac{i-m}{2}\right)$$

$$\begin{aligned}
\hat{g}_l(2) &= w(-2)\hat{g}_{l-1}\left(\frac{2+2}{2}\right) + \cancel{w(-1)\hat{g}_{l-1}\left(\tfrac{2+1}{2}\right)} \\
&\quad + w(0)\hat{g}_{l-1}\left(\frac{2}{2}\right) + \cancel{w(1)\hat{g}_{l-1}\left(\tfrac{2-1}{2}\right)} + w(2)\hat{g}_{l-1}\left(\frac{2-2}{2}\right) \\
&= w(-2)\hat{g}_{l-1}(2) + w(0)\hat{g}_{l-1}(1) + w(2)\hat{g}_{l-1}(0)
\end{aligned}$$

$$\begin{aligned}
\hat{g}_l(5) &= \cancel{w(-2)\hat{g}_{l-1}\left(\tfrac{5+2}{2}\right)} + w(-1)\hat{g}_{l-1}\left(\frac{5+1}{2}\right) + \cancel{w(0)\hat{g}_{l-1}\left(\tfrac{5}{2}\right)} \\
&\quad + w(1)\hat{g}_{l-1}\left(\frac{5-1}{2}\right) + \cancel{w(2)\hat{g}_{l-1}\left(\tfrac{5-2}{2}\right)}
\end{aligned}$$

1-D case

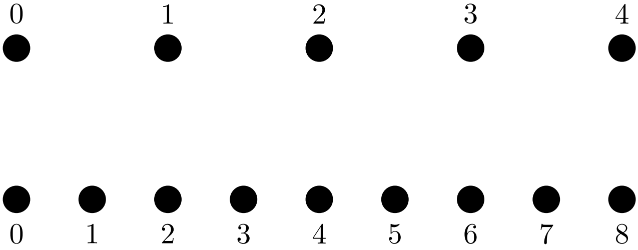
$$\hat{g}_l(i) = \sum_{m=-2}^{2} w(m)\hat{g}_{l-1}\left(\frac{i-m}{2}\right)$$

$$\begin{aligned}\hat{g}_l(2) &= w(-2)\hat{g}_{l-1}\left(\frac{2+2}{2}\right) + \cancel{w(-1)\hat{g}_{l-1}\left(\tfrac{2+1}{2}\right)} \\ &\quad + w(0)\hat{g}_{l-1}\left(\frac{2}{2}\right) + \cancel{w(1)\hat{g}_{l-1}\left(\tfrac{2-1}{2}\right)} + w(2)\hat{g}_{l-1}\left(\frac{2-2}{2}\right) \\ &= w(-2)\hat{g}_{l-1}(2) + w(0)\hat{g}_{l-1}(1) + w(2)\hat{g}_{l-1}(0)\end{aligned}$$

$$\begin{aligned}\hat{g}_l(5) &= \cancel{w(-2)\hat{g}_{l-1}\left(\tfrac{5+2}{2}\right)} + w(-1)\hat{g}_{l-1}\left(\frac{5+1}{2}\right) + \cancel{w(0)\hat{g}_{l-1}\left(\tfrac{5}{2}\right)} \\ &\quad + w(1)\hat{g}_{l-1}\left(\frac{5-1}{2}\right) + \cancel{w(2)\hat{g}_{l-1}\left(\tfrac{5-2}{2}\right)} \\ &= w(-1)\hat{g}_{l-1}(3) + w(1)\hat{g}_{l-1}(2)\end{aligned}$$

1-D case

$$\hat{g}_l(i) = \sum_{m=-2}^{2} w(m)\hat{g}_{l-1}\left(\frac{i-m}{2}\right)$$

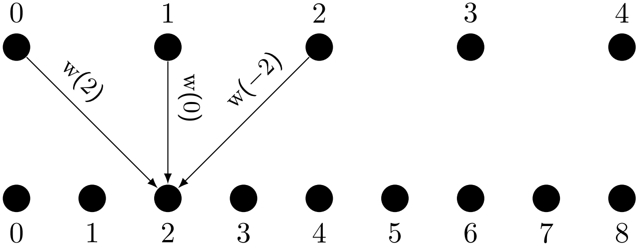


Remark

Note that since we have discard half of the coefficients during expansion, we should double the weights $w(m)$ comparing to those during reduction

1-D case

$$\hat{g}_l(i) = \sum_{m=-2}^{2} w(m)\hat{g}_{l-1}\left(\frac{i-m}{2}\right)$$

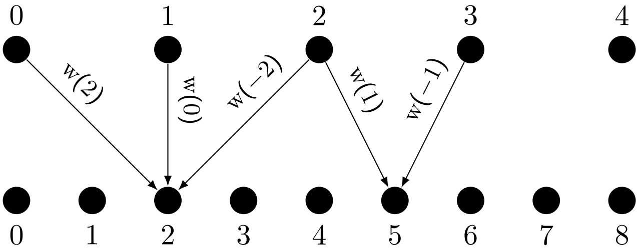


Remark

Note that since we have discard half of the coefficients during expansion, we should double the weights w(m) comparing to those during reduction

1-D case

$$\hat{g}_l(i) = \sum_{m=-2}^{2} w(m)\hat{g}_{l-1}\left(\frac{i-m}{2}\right)$$



Remark

Note that since we have discard half of the coefficients during expansion, we should double the weights w(m) comparing to those during reduction

Design properties of convolution mask

Separable

- That is, $\tilde{w}(m,n) = w(m)w(n)$
- We can save computational cost (from N^2 to $2N$)

Design properties of convolution mask

Separable

- That is, $\tilde{w}(m,n) = w(m)w(n)$
- We can save computational cost (from N^2 to $2N$)

Symmetric

- That is, $w(i) = w(-i)$
- $[w(-2), w(-1), w(0), w(1), w(2)] = [c, b, a, b, c]$

Design properties of convolution mask

Separable
- That is, $\tilde{w}(m, n) = w(m)w(n)$
- We can save computational cost (from N^2 to $2N$)

Symmetric
- That is, $w(i) = w(-i)$
- $[w(-2), w(-1), w(0), w(1), w(2)] = [c, b, a, b, c]$

Normalized
- Sum of mask should be 1
- Thus, $a + 2b + 2c = 1$

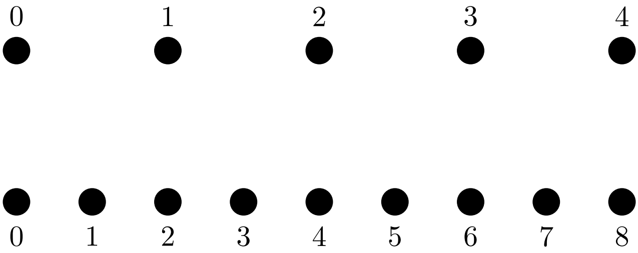
Design properties of convolution mask

Separable
- That is, $\tilde{w}(m, n) = w(m)w(n)$
- We can save computational cost (from N^2 to $2N$)

Symmetric
- That is, $w(i) = w(-i)$
- $[w(-2), w(-1), w(0), w(1), w(2)] = [c, b, a, b, c]$

Normalized
- Sum of mask should be 1
- Thus, $a + 2b + 2c = 1$

"Unbiased"
- All nodes at a given level should contribute the same to nodes at the next level



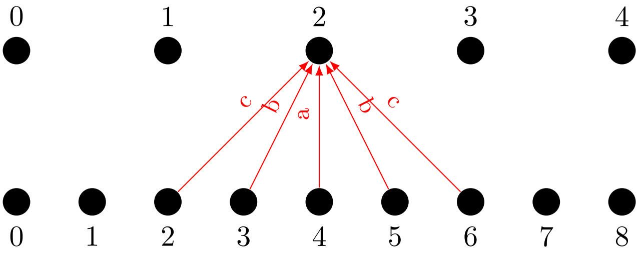
Design properties of convolution mask

Separable
- That is, $\tilde{w}(m,n) = w(m)w(n)$
- We can save computational cost (from N^2 to $2N$)

Symmetric
- That is, $w(i) = w(-i)$
- $[w(-2), w(-1), w(0), w(1), w(2)] = [c, b, a, b, c]$

Normalized
- Sum of mask should be 1
- Thus, $a + 2b + 2c = 1$

“Unbiased”
- All nodes at a given level should contribute the same to nodes at the next level



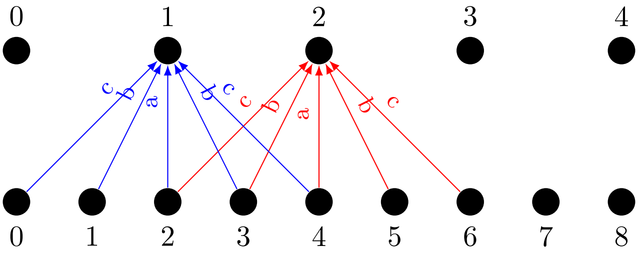
Design properties of convolution mask

Separable
- That is, $\tilde{w}(m,n) = w(m)w(n)$
- We can save computational cost (from N^2 to $2N$)

Symmetric
- That is, $w(i) = w(-i)$
- $[w(-2), w(-1), w(0), w(1), w(2)] = [c, b, a, b, c]$

Normalized
- Sum of mask should be 1
- Thus, $a + 2b + 2c = 1$

"Unbiased"
- All nodes at a given level should contribute the same to nodes at the next level



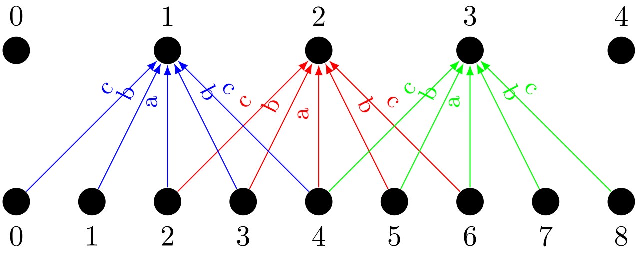
Design properties of convolution mask

Separable
- That is, $\tilde{w}(m,n) = w(m)w(n)$
- We can save computational cost (from N^2 to $2N$)

Symmetric
- That is, $w(i) = w(-i)$
- $[w(-2), w(-1), w(0), w(1), w(2)] = [c, b, a, b, c]$

Normalized
- Sum of mask should be 1
- Thus, $a + 2b + 2c = 1$

"Unbiased"
- All nodes at a given level should contribute the same to nodes at the next level



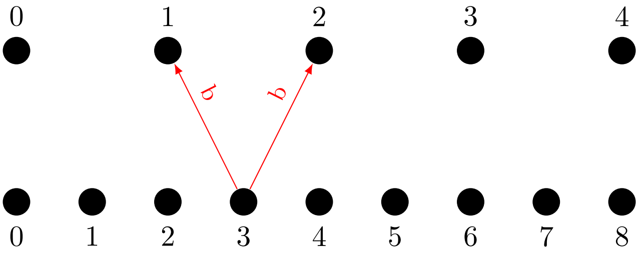
Design properties of convolution mask

Separable
- That is, $\tilde{w}(m, n) = w(m)w(n)$
- We can save computational cost (from N^2 to $2N$)

Symmetric
- That is, $w(i) = w(-i)$
- $[w(-2), w(-1), w(0), w(1), w(2)] = [c, b, a, b, c]$

Normalized
- Sum of mask should be 1
- Thus, $a + 2b + 2c = 1$

"Unbiased"
- All nodes at a given level should contribute the same to nodes at the next level



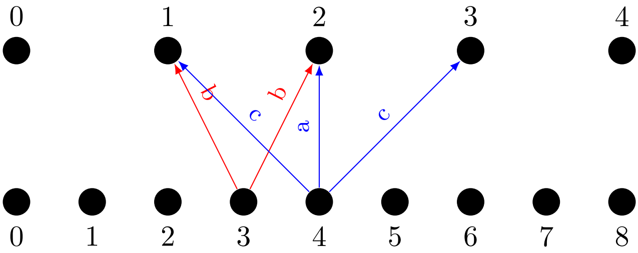
Design properties of convolution mask

Separable
- That is, $\tilde{w}(m, n) = w(m)w(n)$
- We can save computational cost (from N^2 to $2N$)

Symmetric
- That is, $w(i) = w(-i)$
- $[w(-2), w(-1), w(0), w(1), w(2)] = [c, b, a, b, c]$

Normalized
- Sum of mask should be 1
- Thus, $a + 2b + 2c = 1$

"Unbiased"
- All nodes at a given level should contribute the same to nodes at the next level



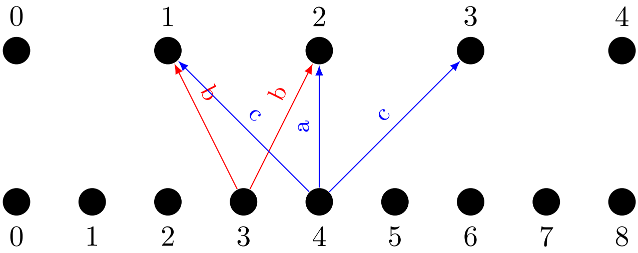
Design properties of convolution mask

Separable
- That is, $\tilde{w}(m,n) = w(m)w(n)$
- We can save computational cost (from N^2 to $2N$)

Symmetric
- That is, $w(i) = w(-i)$
- $[w(-2), w(-1), w(0), w(1), w(2)] = [c, b, a, b, c]$

Normalized
- Sum of mask should be 1
- Thus, $a + 2b + 2c = 1$

"Unbiased"
- All nodes at a given level should contribute the same to nodes at the next level ($2b = 2c + a$)



Mask parameters

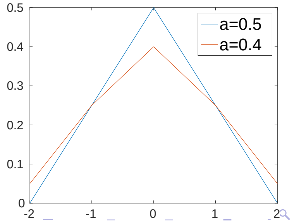
- From the previous slide,

$$\begin{cases} a + 2c = 2b \\ a + 2b + 2c = 1 \end{cases} \Rightarrow 4b = 1 \Rightarrow b = \frac{1}{4}$$

- Substitute $b = \frac{1}{4}$ back to the equations, we have

$$\begin{cases} b = \frac{1}{4} \\ c = \frac{1}{4} - \frac{a}{2} \end{cases}$$

- $a = 0.4 \approx$ Gaussian, $a = 0.5 \approx$ triangular



Laplacian pyramids

- Derive from Gaussian pyramid
  - First construct Gaussian pyramid: g_0 (original image), $g_1 = \text{REDUCE}[g_0]$, $g_2 = \text{REDUCE}[g_1]$, $\cdots$
  - One level of pyramid is difference between the level and approximation through expanding the next level

$$L_0 = g_0 - \text{EXPAND}[g_1]$$
$$L_1 = g_1 - \text{EXPAND}[g_2]$$
$$L_2 = g_2 - \text{EXPAND}[g_3]$$

- Most coefficients of the pyramid are zero $\Rightarrow$ can be used for compression

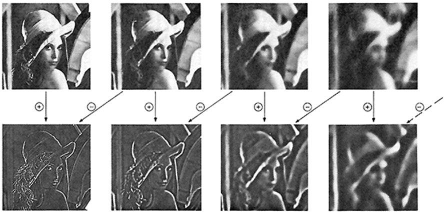


Image encoding

- Compute Gaussian pyramid

$$g_0, g_1, g_2, g_3$$

Image encoding

- Compute Gaussian pyramid

$$g_0, g_1, g_2, g_3$$

- Compute Laplacian pyramid

$$\begin{aligned} L_0 &= g_0 - \text{EXPAND}[g_1] \\ L_1 &= g_1 - \text{EXPAND}[g_2] \\ L_2 &= g_2 - \text{EXPAND}[g_3] \\ L_3 &= g_3 \end{aligned}$$

Image encoding

- Compute Gaussian pyramid

$$g_0, g_1, g_2, g_3$$

- Compute Laplacian pyramid

$$\begin{aligned} L_0 &= g_0 - \text{EXPAND}[g_1] \\ L_1 &= g_1 - \text{EXPAND}[g_2] \\ L_2 &= g_2 - \text{EXPAND}[g_3] \\ L_3 &= g_3 \end{aligned}$$

- Quantize and code Laplacian pyramid (e.g., by Huffmann coding)

Image encoding

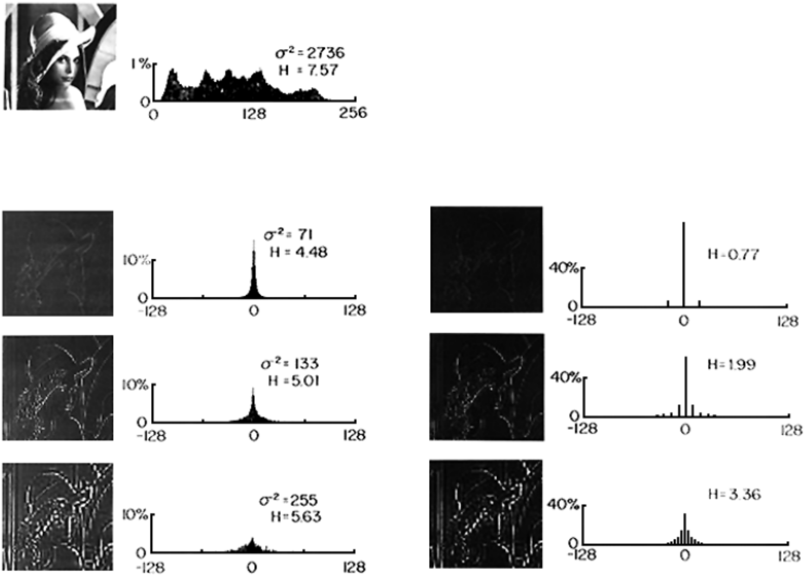


Image decoding

- Decode Laplacian pyramid

$$L_0, L_1, L_2, L_3$$

Image decoding

- Decode Laplacian pyramid

$$L_0, L_1, L_2, L_3$$

- Compute Gaussian pyramid from Laplacian pyramid

$$\begin{aligned} g_3 &= L_3 \\ g_2 &= L_2 + \text{EXPAND}[g_3] \\ g_1 &= L_1 + \text{EXPAND}[g_2] \\ g_0 &= L_0 + \text{EXPAND}[g_1] \end{aligned}$$

Image decoding

- Decode Laplacian pyramid

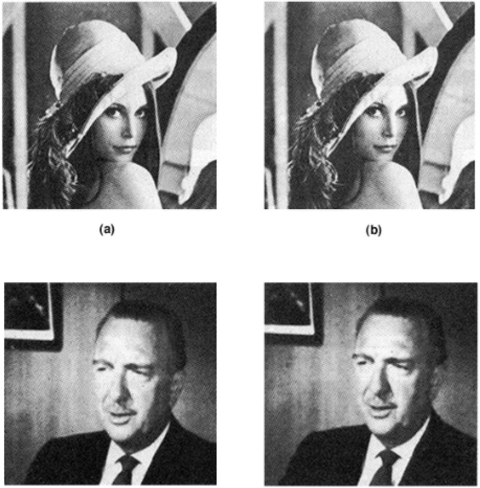
$$L_0, L_1, L_2, L_3$$

- Compute Gaussian pyramid from Laplacian pyramid

$$\begin{aligned} g_3 &= L_3 \\ g_2 &= L_2 + \mathrm{EXPAND}[g_3] \\ g_1 &= L_1 + \mathrm{EXPAND}[g_2] \\ g_0 &= L_0 + \mathrm{EXPAND}[g_1] \end{aligned}$$

- g_0 is the reconstructed image

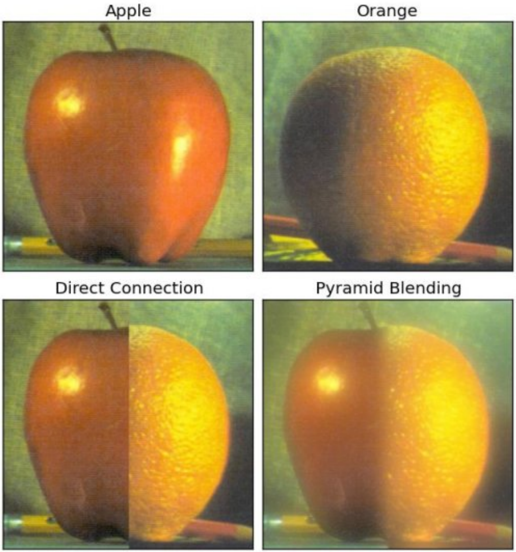
Compression result



(c) (d)

Fig 8. Examples of image data compression using the Laplacian Pyramid code. (a) and (c) give the original "Lady" and "Walter" images, while (b) and (d) give their encoded versions of the data rates are 1.58 and 0.73 bits/pixel for "Lady" and "Walter," respectively. The corresponding mean square errors were 0.88 percent and 0.43 percent, respectively.

Image composting



HW2

Algorithm for image composting

- Generate Laplacian pyramid L_o of the orange image

HW2

Algorithm for image composting

- Generate Laplacian pyramid L_o of the orange image
- Generate Laplacian pyramid L_a of the apple image

HW2

Algorithm for image composting

- Generate Laplacian pyramid L_o of the orange image
- Generate Laplacian pyramid L_a of the apple image
- Generate Laplacian pyramid L_c by
  - Copy left half of nodes at each level from apple pyramid
  - Copy right half of nodes at each level from orange pyramid

HW2

Algorithm for image composting

- Generate Laplacian pyramid L_o of the orange image
- Generate Laplacian pyramid L_a of the apple image
- Generate Laplacian pyramid L_c by
  - Copy left half of nodes at each level from apple pyramid
  - Copy right half of nodes at each level from orange pyramid
- Reconstruct combined image from pyramid L_c