Multi-Resolution Processing
CS 430
©Denbigh Starkey

Background

The original idea of multi-resolution processing of images is that different resolution views of an image contain different types of information. Consider, for example, a 1024×1024 aerial image of a farm. At lower resolution, say 128×128, one will be able to get information on the field structure and buildings. By going to 256×256 one will be able to get far more details on the buildings and should begin to be able to see information like which fields contain animals. By 512×512 one can see individual animals and will get some idea of how fields are being managed, and by 1024×1024 one will get the maximum detail including, say, the ability to identify individual animals and to determine variations in a grain crop in a field.

This basic idea has led to multi-resolution processing, and in particular to wavelets, which can be used for image analysis (including replacing Fourier transforms) and compression (and hence also transmission).

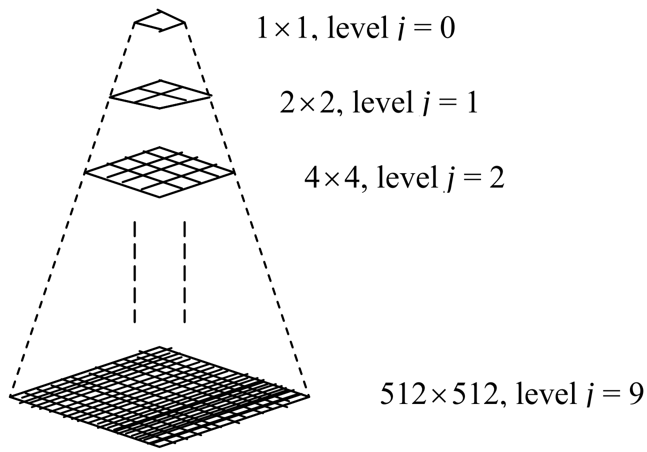
I'll assume throughout these notes that the image is $N \times N$, where N is a power of 2, and so

$$N = 2^J, \text{ where } J = \log_2(N).$$

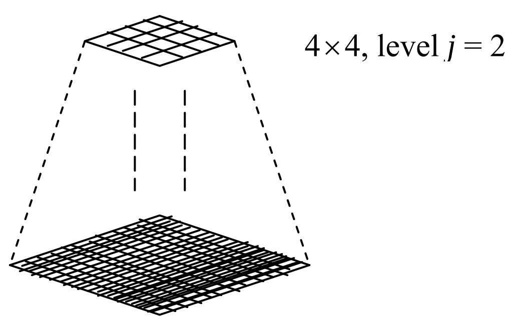
I'll use N and J throughout the notes, without further explanation, for these image values. I'll also switch as needed, and so, for example, for an image with dimensions 1024×1024 I'll just assume that I can say $N = 1024$ and $J = 10$.

Image Pyramids

Multi-resolution analysis is based on the concept of image pyramids. Say that we have a 512×512 input image. A full pyramid for this image will consist of ten images, with resolutions 1×1, 2×2, 4×4, …, 512×512. I.e., we have levels j = 0 through j = 9, where each level has resolution $2^j \times 2^j$.



In general the information high up in the pyramid tends to be useless, because, for example, a 1×1 or a 2×2 image typically doesn't contain much useful information about the scene. So the usual situation will be that we'll have a *P*-pyramid with $P + 1$ layers from level $J - P$ on top down to level J on the bottom. E.g., if $P = 7$ in the example above then the *P*-pyramid will have eight layers from level 2 through level 9.

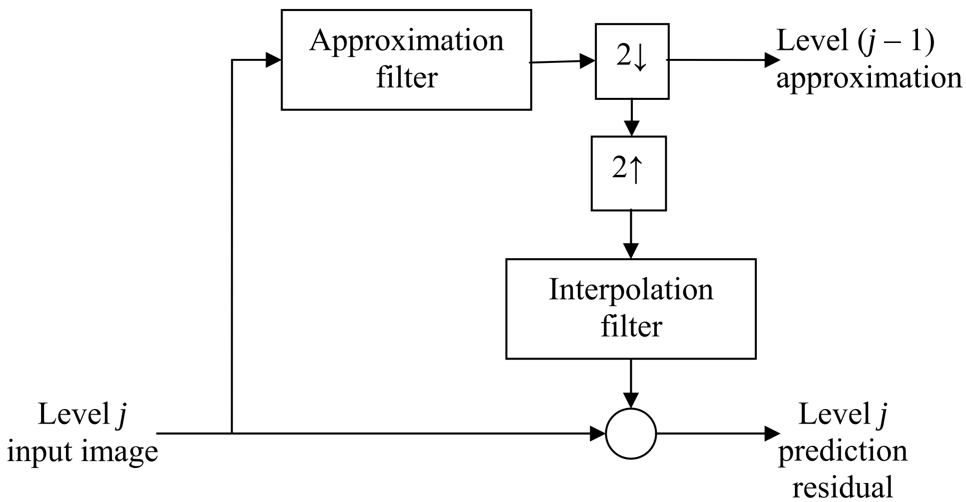


If we were to save all of these pyramids this would appear to require a lot more storage, but it cannot increase the storage requirements by as much as a third. To see this, sum the pixels at each level, from the bottom, giving:

$$N^2 + \tfrac{1}{4}N^2 + (\tfrac{1}{4})^2 N^2 + \ldots + (\tfrac{1}{4})^P N^2, \text{ since there are } P + 1 \text{ levels.}$$

$$= N^2(1 + \tfrac{1}{4} + (\tfrac{1}{4})^2 + .. + (\tfrac{1}{4})^P) = N^2 \frac{1 - (\frac{1}{4})^{(P+1)}}{1 - \frac{1}{4}} = \tfrac{4}{3} N^2 (1 - (\tfrac{1}{4})^{(P+1)}) < \tfrac{4}{3} N^2.$$

The process at this point actually involves the creation of two pyramids, an *approximation pyramid* and a *prediction residual* pyramid. The process is described in Gonzalez and Woods with the following diagram:



This will be used repeatedly, for $j = J$ down to $j = J - P + 1$ to get a P-pyramid of approximations from $J - 1$ to $J - P$, and a prediction residuals pyramid from J to $J - P$.

First, some explanation of the notation is needed.

The approximation filter and 2↓ form a downsampler or subsampler which returns an image with half the resolution in each dimension, which is the $j - 1$ approximation image shown. This can be created in a number of ways, of which the most common are:

- Sampling, where one of four pixels in the input image is used to provide the pixel in the reduced resolution image.
- Using the mean, or possibly median, of four pixels to provide the pixel in the reduced resolution image.
- Using a lowpass Gaussian filter.

The quality of the process is in the inverse order of the options shown above.

The 2↑ and interpolation filter take the reduced image and build an image with the original resolution, usually by supersampling. I.e., if we are using supersampling then one pixel in the reduced image becomes four in the new image. The original image and the supersampled image are then compared to come up with a difference image, called the prediction residual image, which can be used to recreate the original image.

E.g., say that $J = 2$, and we have the 4×4 three-bit image shown below:

6	7	5	3
6	5	4	3
3	3	2	5
2	0	1	3

Using a mean approximation filter with rounding will give two approximation images,

6	4
2	3

4

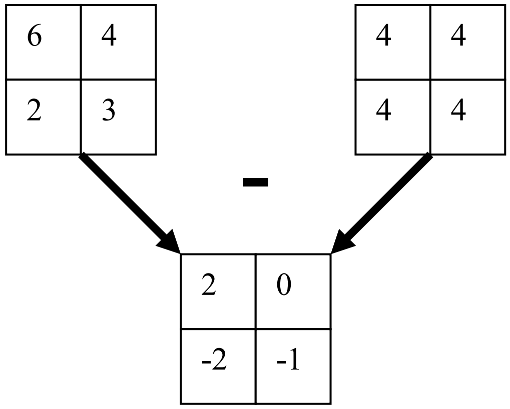
If we use supersampling and the original images these give two prediction residual images

6	7	5	3
6	5	4	3
3	3	2	5
2	0	1	3

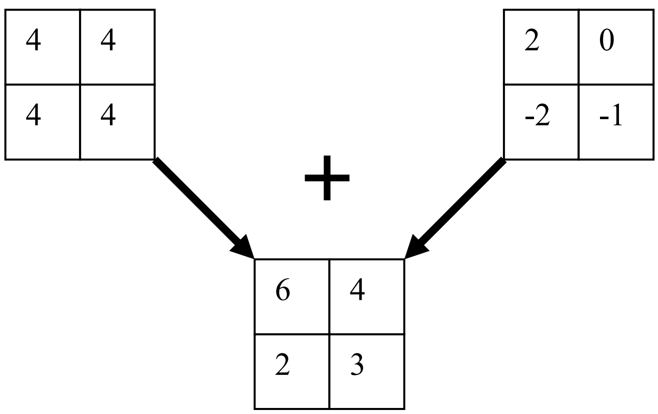
-

6	6	4	4
6	6	4	4
2	2	3	3
2	2	3	3

0	1	1	-1
0	-1	0	-1
1	1	-1	2
0	-2	-2	0



Now with the image on top of the approximation pyramid we can use the prediction residual images to rebuild the rest of the approximation tree. In this case we start with a single 4. Supersample it and add the 2×2 prediction residual which gives:



Repeat this for the next level:

6	6	4	4
6	6	4	4
2	2	3	3
2	2	3	3

+

0	1	1	-1
0	-1	0	-1
1	1	-1	2
0	-2	-2	0

6	7	5	3
6	5	4	3
3	3	2	5
2	0	1	3

The gain here is that compression will be much better on the prediction residual images, because in areas of the image that have relatively constant intensity they will have values that are all at, or close to, zero. So by just saving the top image in a *P*-pyramid and the pyramid of residuals we can, with frequency-based encoding of the pixel values, achieve significant compression.

This shows up well in Figure 7.3 of Gonzalez and Woods, which shows the processing of a picture of a philodendron in a vase with three approximation images created using a lowpass Gaussian filter, and shows the corresponding prediction residual images. The figure also shows the histogram for the original image and for the prediction residual images. The original image has a histogram that is so wide and flat that it will be difficult to compress effectively. On the other hand the histogram for the prediction residual images is tightly concentrated around zero, and so by assigning codes whose length is shortest closer to the center of the histogram we could achieve significant compression.