Computer Vision

MoSIG M2

James L. Crowley

Fall Semester 5 November 2020

Lesson 4

Invariant Image Description with Receptive Fields

Lesson Outline:

Glossary of Symbols and Mathematical Notation

Convolution of *f(n)* with *g(x)*. $f(x) = (f * g)(x) = \sum_{n=-\infty}^{\infty} f(x-n)g(n,\sigma)$

Normalized 1-D Gaussian Function $g(x,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$

Gaussian Scale Parameter σ

Gaussian Scale Property $g(x,\sigma_1) * g(x,\sigma_2) = g(x,\sqrt{\sigma_1^2 + \sigma_2^2})$

Normalized 2-D Gaussian Function $g(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$

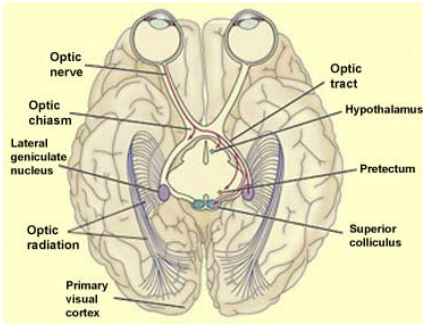
2D Gaussian Separability Property $g(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} * \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}$

The Laplacian of the Gaussian $\nabla^2 g(x,y,\sigma) = g_{xx}(x,y,\sigma) + g_{yy}(x,y,\sigma) = \frac{\partial g(x,y,\sigma)}{\partial \sigma}$

Finite impulse response (FIR) digital Gaussian filter $G(n,\sigma) = \frac{1}{B} W_N(n) \cdot e^{-\frac{(n\Delta x)^2}{2\sigma^2}}$

1. The Mammalian Visual Cortex

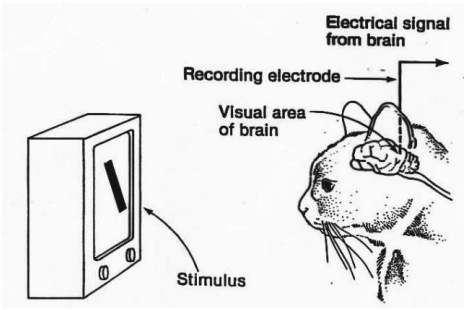
The Visual Cortex of mammals is composed of multiple layers of retinotopic maps.



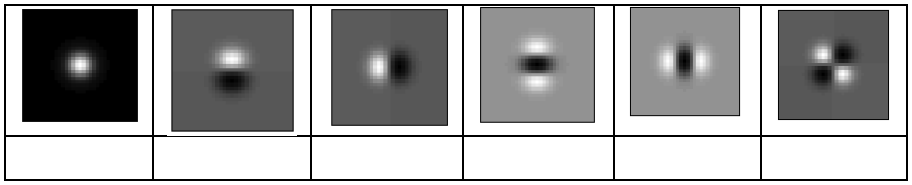
Each map is an image of the retina projected onto (convolved with) a receptive field at different scales and different orientations.

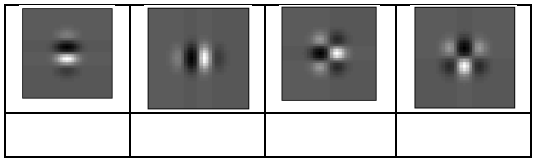
Receptive Fields in the Visual Cortex

In 1968, Hubel and Wiesel probed the visual cortex of a cat with electrodes and found layers of cells that responded to local patterns of stimulation. The discovered that the visual cortex is composed of a series of layers. Each layer is a map of the retina filtered by a "receptive field" that respond to a certain pattern over a narrow range of sizes and orientations.

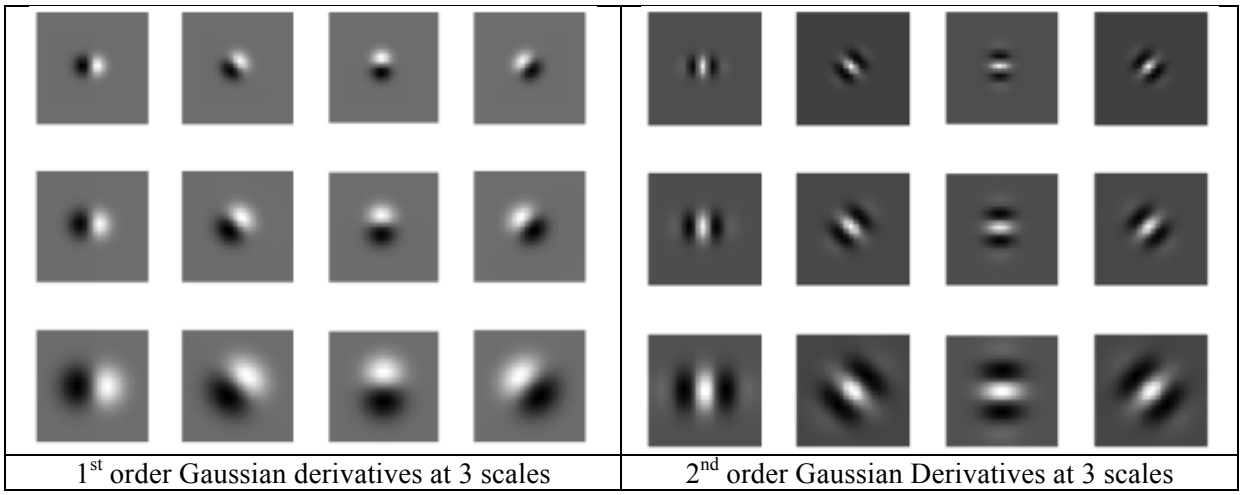


The patterns at the lowest level look like these:





Each layer is a specific pattern at a specific orientation and scale.



1st order Gaussian derivatives at 3 scales	2nd order Gaussian Derivatives at 3 scales

This figure shows first and second order Gaussian derivative features from 3 scales, computed using sums and differences of adjacent samples in a half-octave Gaussian pyramid.

The first level receptive fields were found to be local filters that pass a narrow range of spatial frequencies. The first layer receptive fields were found to be modeled by Gabor functions (Gaussian modulated by a Cosine plus an imaginary Sin).

The first layer receptive fields can also be modeled multi-scale derivatives of Gaussians functions. These two representations are very similar. However, Gabor functions can be very expensive to convolve with an image. Multi-scale Gaussian derivatives can convolved very efficiently because of a number mathematical properties that we will discuss below.

As they moved up the visual cortex, Hubel and Weisel found that these patters were combined to form more complex patterns, such as corners, bars, crosses, etc. These were named "complex" receptive field, and are similar to receptive field patterns learned by convolutional neural networks.

It is possible to learn receptive fields for local image description using back propagation. This is fine for tuning a system to a specific problem domain. However, the general case of vision in the real world, this can require a very large set of training data and computing time.

Researchers in several different communities have found that as the variety of training data increases, the learned receptive fields converge to Gaussian Derivative Functions. This result has been demonstrated by researchers in both machine learning and computer vision, since the 1990's.

2. Gaussian Functions and Gaussian Filters

In the following, we use lower case, $g(x,\sigma)$, for the Gaussian function, and uppercase case, $G(x,\sigma)$, for Finite Impulse Response Gaussian Filters.

2.1. 2-D Finite Impulse Response Gaussian Filters

Gaussian functions

The Gaussian Function: $g(x,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$ (x can be real or integer)

Discrete (sampled) Convolution with a function: $(f * g)(x) = \sum_{n=-\infty}^{\infty} f(x-n)g(n,\sigma)$

Scale Property: $g(x,\sigma) * g(x,\sigma) = g(x,\sqrt{2}\sigma)$

More Generally: $g(x,\sigma_1) * g(x,\sigma_2) = g(x,\sqrt{\sigma_1^2 + \sigma_2^2})$

And in 2-D $g(x,y,\sigma_1) * g(x,y,\sigma_2) = g(x,y,\sqrt{\sigma_1^2 + \sigma_2^2})$

The 2D Gaussian function is: $g(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$

This function is separable: $g(x,y,\sigma) = g(x,\sigma) * g(y,\sigma)$

And thus $p * g(x,y,\sigma) = \left(p * g(x,\sigma)\right) * g(y,\sigma)$

(a 2-D convolution can be computed as two O(N) 1-D convolutions).

2.2. Finite Impulse Response Gaussian Digital Filters

The Gaussian function can be used to construct a finite impulse response (FIR) digital filter. This requires (1) sampling the function and (2) limiting its duration (windowing) and (3) renormalizing its gain.

The finite impulse response (FIR) digital Gaussian filter is: $G(n,\sigma) = \frac{1}{B} W_N(n) \cdot e^{-\frac{(n\Delta x)^2}{2\sigma^2}}$

The function Δx is a sample distance (or sample rate). For simplicity, we can use Δx=1, but other values are possible.

The term B is a normalization factor assures that the "gain" of the filter is 1.

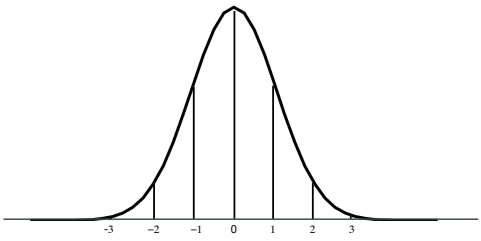
$$B = \sum_{n=-R}^{R} e^{-\frac{n^2}{2\sigma^2}}$$

The normalisation factor, B, will be slightly less than $\sqrt{2\pi}\sigma$ because of the window function truncates the tails of the Gaussian.

The function $W_N(n)$ is called a window function, and serves to limit the spatial extent (window size) of the Gaussian. We will use a rectangular function as a window:

$$W_N(n) = \begin{cases} 1 & \text{for } -R \le n \le R \\ 0 & \text{otherwise} \end{cases}$$

Where R is a "radius". Typically R should be $R \geq 3\sigma$ for reasons developed in the previous lecture. Setting $\Delta x=1$ and R = 3 gives a filter that looks like this:



The 2-D Gaussian filter

The 2D Finite Impulse Response (FIR) Gaussian filter is $G(x,y,\sigma) = \frac{1}{B} W_N(x,y) \cdot e^{-\frac{(x^2+y^2)}{2\sigma^2}}$

Where *x* and *y* are integers.

We limit the spatial extent of the Gaussian with a Window function $W_N(x,y)$

$$W_N(x,y) = \begin{cases} 1 & \text{for } -R \le x \le R \text{ and } -R \le y \le R \\ 0 & \text{otherwise} \end{cases}$$

The finite window, $W_N(x,y)$ has $N^2 = (2R+1)^2$ coefficients

Where R is a "radius. Typically R should be $R \geq 3\sigma$.

A normalization factor assures that the "Gain" (sum of the coefficients) is 1.

$$B = \sum_{x=-R}^{R} \sum_{y=-R}^{R} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \le 2\pi\sigma$$

2.3. Binomial Filters

The binomial series is the series $b_n(m)$ of coefficients of the polynomial:

$$(x+y)^n = \sum_{m=0}^{n} b_n(m) x^{n-m} y^m$$

The coefficients can be computed as an auto-convolution $b_n(m) = (1, 1)^{*n}$
These are the coefficients of Pascal's Triangle.

The variable n, the number of "auto-convolutions" represents the "row" of the filter.
The value μ is the center of gravity of the filter. The value σ is the standard deviation

$b_n(m) = (1, 1)^{*n}$: n convolutions of (1, 1)

Gain (sum) is: $s_n = \sum_{m=0}^{n} b_n(m) = 2^n$

Center of gravity is $\mu_n = \frac{1}{s_n} \sum_{m=0}^{n} b_n(m) \cdot m = \frac{n}{2}$

The variance is: $\sigma_n^2 = \frac{1}{s_n} \sum_{m=0}^{n} b_n(m) \cdot (m-\mu)^2 = \frac{n}{4}$

This can be used to compute the center and variance (scale) for each filter:

n	sum = 2^n	μ= n/2	σ^2 = n/4	$\sigma = \sqrt{n/2}$	Coefficients
0	1	0	0	0	1
1	2	0.5	0.25		1 1
2	4	1	0/5		1 2 1
3	8	1.5	0.75		1 3 3 1
4	16	2	1	1	1 4 6 4 1
5	32	2.5	1.25		1 5 10 10 5 1
6	64	3	1.5		1 6 15 20 15 1
7	128	3.5	1.75		1 7 21 35 35 21 7 1
8	256	4	2	$\sqrt{2}$	1 8 29 56 70 56 29 8 1

These coefficients provide a family of low pass filters with remarkable properties. They are the best integer coefficient approximation for a Gaussian filter, and are thus inexpensive to use on an embedded system without floating point hardware.

Three filters, in particular, are interesting: $b_2(m)$=(1, 2, 1), $b_4(m)$=(1, 4, 6, 4, 1) and $b_8(m)$=(1, 8, 29, 56, 70, 56, 29, 8, 1). The filter $b_2(m)$ is used in the Sobel edge detector, invented by Irwin Sobel in his 1964 Stanford Doctoral thesis. This detector was made famous by the classic text book on Pattern Recognition and Image Processing by R. Duda and P. Hart published in 1972.

The Sobel edge detector is composed of two image filters.

$$m_r(i,j) = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad m_c(i,j) = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$m_c(i,j)$: detects contrast in column direction. $m_r(i,j)$: detects contrast in row direction
When used as a low pass filter it is common to divide by the sum of the coefficients 4.

The filter $b_4(m)$=(1, 4, 6, 4, 1) = $b_2(m) * b_2(m)$ provides an integer approximation for Gaussian Filter with σ= 1 using only 5 integer coefficients. Note that the sum of the coefficients is 2^4=16

The 2-D version of this filter, $b_4(i,j)=b_4(i)*b_4(j)$, is often used as the σ= 1 filter to compute a Gaussian Pyramid. For example, OpenCV contains a Gaussian Smoothing function that uses this filter.

$b_4(i,j) = (1/256)$

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

The $\sigma=\sqrt{2}$ version is $b_8(i,j)=b_4(i,j)*b_4(i,j)$. The impulse response for $b_8(i,j)$ is 9x9 but can be computed as convolution of 5x5 filters. We can even compute $b_8(i,j)$ as

$b_8(i,j)=b_4(i)*b_4(i)*b_4(j)*b_4(j)$ for a cost of 4x5=20 operations
or
$b_8(i,j)=b_2(i)*b_2(i)*b_2(i)*b_2(i)*b_4(j)*b_4(j)*b_4(j)*b_4(j)$ for a cost of 8x3=24 operations.

This is an approximation for an FIR Gaussian, where the approximation error decreases with increasing sigma.

3. Scale Invariant Image Pyramids

At the end of our last lecture we saw a recursive algorithm for computing a full octave pyramid with a scale invariant impulse response. In this lecture I would like to develop more fully the notion of scale invariance, and introduce the notion of a scale invariant half-octave pyramid.

As before, let, *P(x,y),* be an image array of size *RxC* pixels, where *(x, y)* are integers.

To compute the scale space representation of the image we must sample scale space in x, y and in σ. As we have seen above, σ is generally sampled over an exponential range.

$$\sigma_k = \sigma_0 \Delta\sigma^k$$

Where $\sigma_0 \geq 1$. We will sometimes use $\sigma_0 = 1$ for simplicity.

The smallest value of for σ_k is σ_0, because for k=0 as $\Delta\sigma^0 = 1$.

The largest value of scale is determined by the image size, say

$$\sigma_{k_{\max}} = \max(R,C)$$

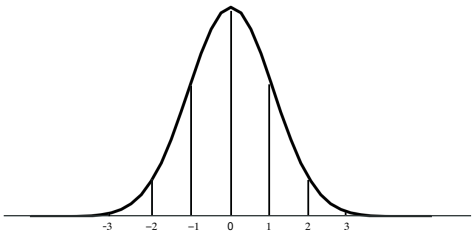
For a full octave pyramid, we can have maximum of $K_{max} = Log_2\{\max(R,C)\}$ levels.

The **impulse response** of a digital filter is the result of processing an input signal composed of a single impulse (dirac), with all other samples set to zero. For convolution with an FIR filter, this would return the coefficients of the filter. For a sequence of filtering operations, this is shows the equivalent composite filter.

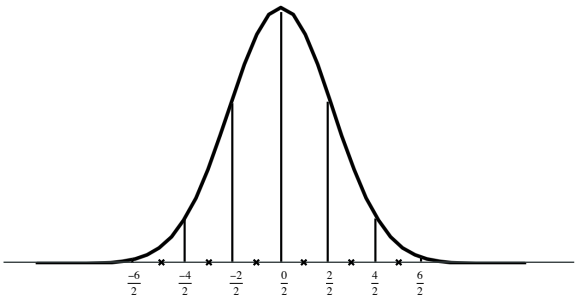
Scale invariance means that the impulse response is identical at every level of the pyramid. This is assured by having the sample rate (Δx_k, Δy_k) by directly proportional to the scale variable, σ_k.

For example, if we convolve a Gaussian filter $P(x,y,1) = P(i,j) * G(x,y,\sigma = 1)$

We obtain an impulse response of

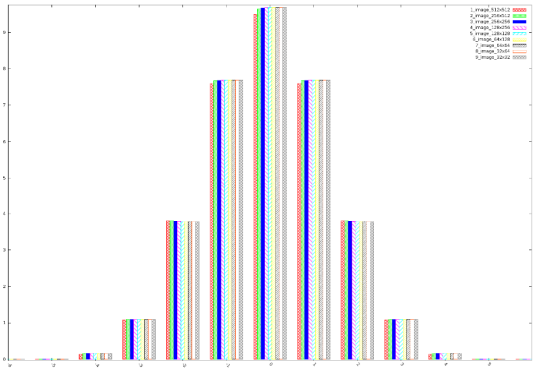


If we convolve our impulse with Gaussian filter with σ=2, and then resample with $\Delta x = 2$ we obtain the same result



Thus we can say that the impulse response is "invariant" with σ.

For a scale invariant pyramid the impulse responses should look like this



(This was taken from a half-octave pyramid algorithm using root-2 sampling, but the principle is the same.)

Last lecture we introduced a 2D sample operator that computes every Δx_k pixel of every Δy_k row:

$$P(i,j,k) = S_2\{P(x,y,k)\} = P(2i,2j,k)$$

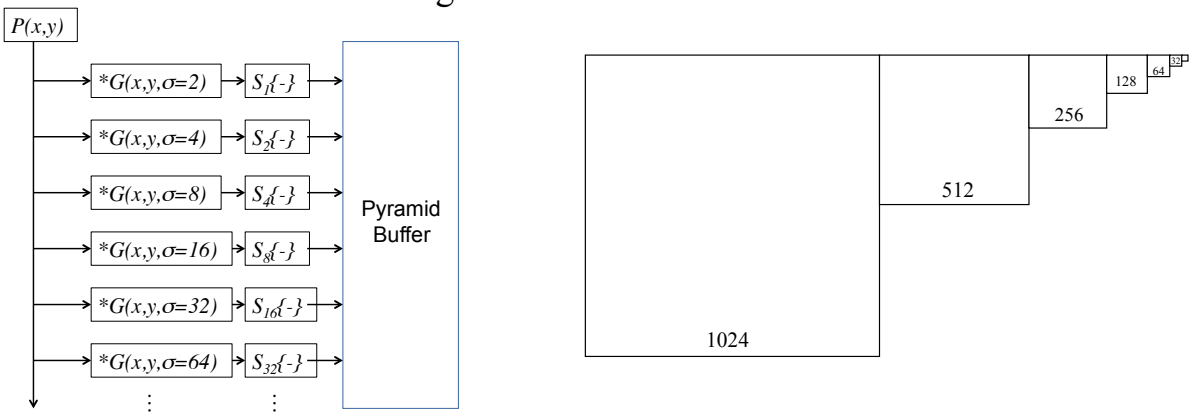
The variables x, y are the pixel coordinates in the original image. The variables i,j are the sample position for level k.

in general $P(i,j,k) = S_{\Delta x_k}\{P(x,y,k)\} = P(i\Delta x_k, j\Delta x_k, k)$

For a scale invariant impulse response, the spatial sample rates Δx_k and Δy_k must be proportional to σ_k. To avoiding aliasing, we require that $\Delta x_k = \Delta y_k \leq \sigma_k$.
Using a value of $\sigma_0 = 1$ results in a small but acceptable amount of aliasing.

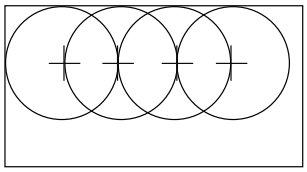
Using $\sigma_0 = 2$ and $\Delta x_k = \Delta y_k = \sigma_k / 2$ minimizes aliasing and can improve recognition rates for many applications.

With the above values the direct algorithm for this would look like this:



The result is a collection of images of decreasing size with a total of 1.5 x 1024^2 samples. (1+1/4+1/8+...)≈1.5

The cost for computing this pyramid by brute force appears very high. However, the math for this is somewhat misleading. The convolution equation suggests that an inner product with an N_k x N_k filter at EVERY pixel in the image where $N_k=6\sigma_k+1$. However, the inner product is followed by resampling, meaning that most of the resulting samples are not used!



It is possible to program the convolution as a sequence of inner-products with a step size of $\Delta x_k=\Delta y_k = \sigma_k/2$. Consider the convolution equation for a large Gaussian ($\sigma_k>>1$) with an image $P(i,k)$.

$$P(x,y,k) = P(x,y) * G(x,y,\sigma_k) = \sum_{i=-R_k}^{R_k} \sum_{j=-R_k}^{R_k} P(x-i, y-j) G(i,j,\sigma_k)$$

The coordinates (x,y) are the original pixel coordinates. This formula can be evaluated at step sizes of $\Delta x_k=\Delta y_k = \sigma_k/2$. As the filter grows in size, the number of inner products decreases proportionally, and the computational cost stays constant!

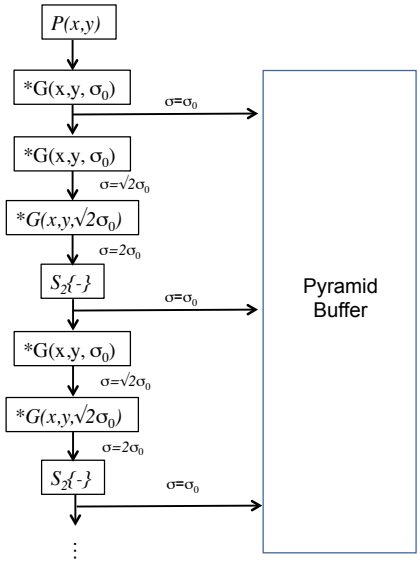
We can express this as a sampled discrete convolution operator ($*_{\Delta x}$).
This says to apply the convolution every Δx samples:
For example, for i from 1 to $C/\Delta x_k$, and j from 1 to $R/\Delta x_k$

$$P(i,j,k) = P(x,y) *_{\Delta x} G(x,y,\sigma_k) = \sum_{m=-R_k}^{R_k} \sum_{n=-R_k}^{R_k} P(i\Delta x_k - m, j\Delta x_k - n) G(m,n,\sigma_k)$$

Pyramid samples are at discrete positions *(i, j, k)*
The image pixel coordinates for any sample *P(i, j, k)* are $x=i\Delta x_k$ and $y=j\Delta x_k$

3.1. Scale Invariant Recursive Pyramid Algorithm

In the last lecture we saw a recursive algorithm for computing a full octave pyramid. Scale Invariance requires an initial convolution with a Gaussian at $\sigma=\sigma_0$, and then compute each successive level with a Gaussians of $\sigma=\sigma_0$ and $\sigma=\sqrt{2}\sigma_0$. This is the pyramid used in the SIFT image descriptor.

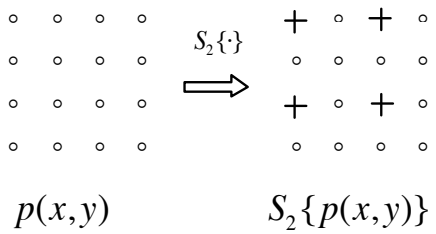


This is often computed with $\sigma_0=1$. The Gaussian filters with $\sigma_0=1$ can be implemented with a 7x7 filter and $\sigma=\sqrt{2}$ is implemented with a 9x9 filter. Thus the cost for the first level is 2x7=14 operations. The cost for each successive level is 2x7 +2x9 = 32 operations (multiplies and adds) per pixel. The total number of multiplies and adds is 14 + 32 (1 + 1/4 + 1/16 +...) = 64 operations (multiplies and adds) per pixel, and the total number of samples in the pyramid is NxNx(1 + 1/4 + 1/16 +...) or approximately 1.5 x N x N. This is easily computed at video rate for most computers.

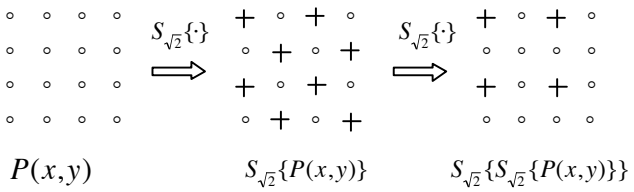
However, a sample step size of $\Delta\sigma=2$ is very large. The same algorithm can be used to provide a step size of $\Delta\sigma=\sqrt{2}\sigma$ using $\sqrt{2}$ sampling.

3.2. Square root of 2 sampling.

Last week we saw the sample operator $S_2\{\cdot\}$.



Each sample is Δx=2 pixels from the nearest neighbor. It is also possible to sample a 2D grid with a step size of Δx=√2. Applying such sampling recursively a second time results in a step size of 2.



We can use this to compute a Half Octave ($\Delta\sigma=\sqrt{2}\sigma$) Pyramid.

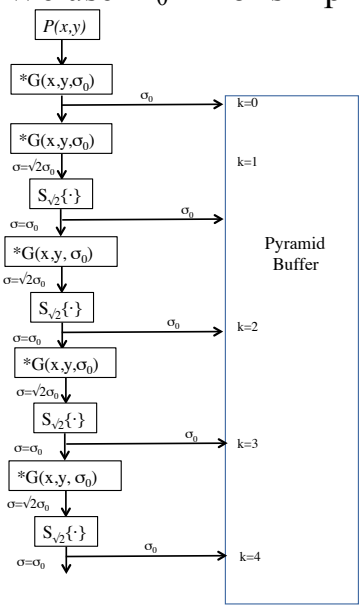
When programming a √2 sampling, one encounters a practical detail of how to address the samples on the odd (√2 sampled) levels. The simplest solution is to simply use the same adresses as the corresponding lower even level, leaving every other sample to be zero. Every other pixel of each row is left to be zero.

For a pyramid, for example, for odd k, $\Delta x_k=\Delta x_{k-1}$. For even k, $\Delta x_k=2^{k/2}$

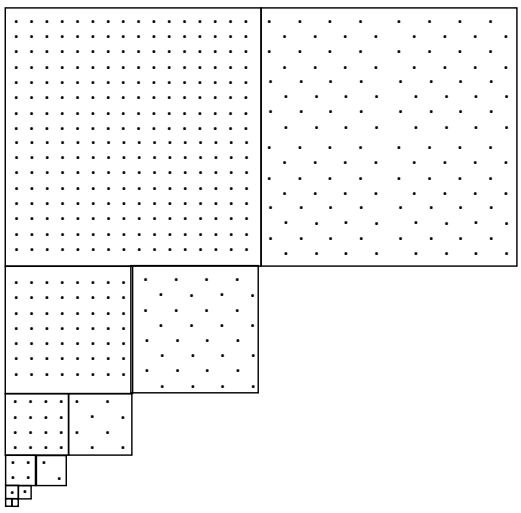
When memory is important, more complicated address schemes are possible.

3.3. Scale Invariant Half-Octave Recursive Pyramid Algorithm

The following diagram shose the algorithm for the scale-invariant half-octave pyramid. We use $\sigma_0=1$ for simplicity.



Half-Octave Pyramid Algorithm



Half-Octave Pyramid Data Structure

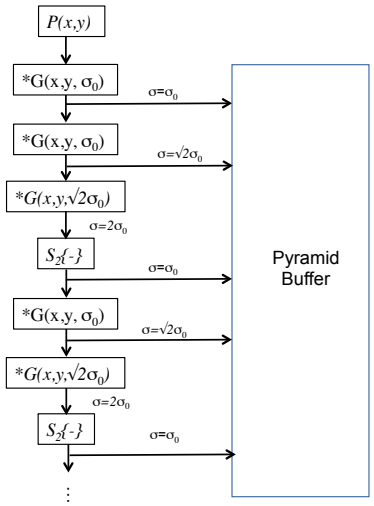
A half octave pyramid uses √2 sampling. For each even level of the pyramid the $S_{\sqrt{2}}\{\cdot\}$ sampling operation takes a filtered image at $\sigma=\sqrt{2}$ and expresses it on a sample grid of where $\sigma=1$. However, for the even levels, the sample distance is along the diagonals.

Thus all levels of the pyramid have an impulse response equal to a Gaussian of σ=1. For the odd levels, the distance is measured along the diagonals, and the Gaussian impulse response must be extracted in the diagonal direction to see the invariance.

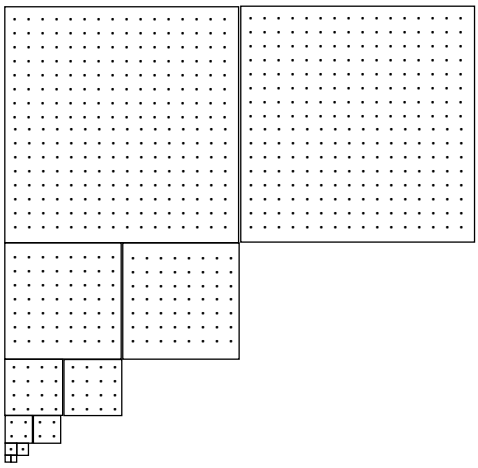
For a half octave pyramid, the number of samples is an RxC image, the total number of samples P = $R \cdot C \cdot (1+1/2+1/4+1/8+...) = 2 \cdot R \cdot C$. The number of filter operations is the same as for the scale invariant full octave pyramid: 64 operations per pixel!

The half-octave pyramid provides a sufficiently dense sampling of scale to enable some interesting image invariants to be easily extracted.

However, the algorithms for addressing root 2 sample grids are very complicated. Thus for simplicity, this is typically computed with an algorithm like this:



Half-Octave Pyramid Algorithm



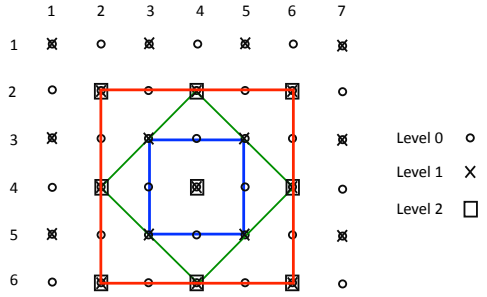
Half-Octave Pyramid Data Structure

This is the structure used in the SIFT detector, for example.

3.4. Multi-Resolution Region of Interests

The multi-resolution equivalent to a 2D Region of Interest (ROI), is a 3D volume of pyramid samples. We can refer to this as a multi-resolution region of interest (MS-ROI). Each multi-resolution region of interest has a center sample, *P(i, j, k)*, surrounded by adjacent samples within some radius of Δi, Δj and Δk.

For example consider a 3x3x3 Multi-resolution ROI from a half octave pyramid.



This ROI is centered on an odd level (k=1) where the 3x3 neighborhood for this pixel is a diamond shape shown in green. The 3x3 neighborhood at level k=0 is a square shown in blue. The 3x3 neighborhood at level k=2 is shown in red.

For this ROI to exist, the pyramid must have a sample at position P(i,j,k+1). This requires that at level k, this ROI exists only when i and j have even values. For example, this neighborhood is shown centered as pixel (4, 4, 1). A simpler way to see this is to consider the center pixel at level k+1 as the address. For any sample in the pyramid at position P(i,j,k), when $k \geq 2$ there exists such a neighborhood centered at pixel P(i,j,k–1).

The 3x3x3 ROI has $3^3 = 27$ samples. It is also possible to compute multi-resolution ROIs at 5x5x5 (with 125 Samples) and 7x7x7 (with 343 samples).

Multi-resolution ROIs can be used as input windows to convolutional neural networks. In this way, a CNN trained with data over a narrow range of scales can be used over all scales. This can result in a substantial reduction in the required amount of training data, and a substantial reduction in the number of neural units requierd for eeuivalent performance.

4. Gaussian Receptive Fields

Gaussian Scale Space has a number of very interesting invariant and equivariant properties. Differential operators can be used to detect key-points (interest points) in scale space. These can be used to define invariant structures that represent appearance independent of scale, position or image plane rotation. These properties are detected with Gaussian derivatives, also known as Gaussian Receptive Fields.

For a variety of reasons, derivatives of the Gaussian function emerge as general local image features. Chief among these are invariance to affine transformations. In addition, using a well-defined mathematical function for receptive fields makes it possible to predict and explain system behavior, AND to avoid the very high cost of learning general functions. Gaussian Receptive Fields also make possible a highly efficient computation for the lower levels of a deep network for image recognition.

4.1. Gaussian Image Derivatives

Gaussian derivatives provide general-purpose receptive fields for image descriptions. These descriptions can be made invariant to scale, orientation, and illumination, and robust to changes in viewing direction and illumination color.

A derivative can be shown to be form of convolution. As a result, derivatives are commutative with convolution, and the derivative of the convolution with a function is equal to convolution with the derivative of a function.

$$f_x(x) = \frac{\partial (f * g)(x)}{\partial x} = \left(f * \frac{\partial g(x)}{\partial x} \right)(x) = \sum_{n=-\infty}^{\infty} f(x-n) g_x(n)$$

We can use this to compute image derivatives with Gaussian receptive fields.

For a 2D Gaussian function of $g(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$

The 1st derivative in x is: $g_x(x,y,\sigma) = -\frac{x}{\sigma^2} \cdot \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$

The 2D FIR Gaussian Derivative filter is: $G_x(x,y,\sigma) = -\frac{1}{B} W_N(x,y) \frac{x}{\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$

As before, B is a normalization factor assures that the "gain" of the filter is 1, and $W_N(n)$ is a window function, that limits the spatial extent to N=2R+1 coefficients, and R should be $R \geq 3\sigma$. Note that the sum of the coefficients of the derivative should be 0. The normalization factor is computed from the truncated Gaussian

$$B = \sum_{x=-R}^{R} \sum_{y=-R}^{R} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Gaussian Derivatives are used to compute derivatives of images

For images

$$P_x(x,y,\sigma) = P(x,y) * G_x(x,y,\sigma) \quad P_y(x,y,\sigma) = P(x,y) * G_y(x,y,\sigma)$$
$$P_{xx}(x,y,\sigma) = P(x,y) * G_{xx}(x,y,\sigma) \quad P_{yy}(x,y,\sigma) = P(x,y) * G_{yy}(x,y,\sigma)$$

Using a Gaussian Pyramid

$$P_x(i,j,k) = P(i,j,k) * G_x(x,y,\sigma) \quad P_y(i,j,k) = P(i,j,k) * G_y(x,y,\sigma)$$
$$P_{xx}(i,j,k) = P(i,j,k) * G_{xx}(x,y,\sigma) \quad P_{yy}(i,j,k) = P(i,j,k) * G_{yy}(x,y,\sigma)$$

A typical scale value in this case is σ=1

4.2. Steerability of Gaussian Derivatives.

It is possible to synthesize an oriented derivative at any point as a weighted sum of derivatives in perpendicular directions. The weights are given by sine and cosine functions. The weights are given by sine and cosine functions.

$$g_x^{\theta}(x,y,\sigma) = \cos(\theta) \cdot g_x(x,y,\sigma) + \sin(\theta) \cdot g_y(x,y,\sigma)$$

Higher order derivatives can also be steered.

Thus we can use Gaussian image derivatives along the row and column direction to compute the derivative at any orientation.

Thus:

1st order $P_x^{\theta}(i,j,k) = Cos(\theta)P_x(i,j,k) + Sin(\theta)P_y(i,j,k)$

2nd order $p_{xx}^{\theta}(i,j,k) = Cos(\theta)^2 p_{xx}(i,j,k) + 2Cos(\theta)Sin(\theta)p_{xy}(i,j,k) + Sin(\theta)^2 p_{yy}(i,j,k)$

3rd order

$$p_{xxx}^{\theta}(i,j,k) = Cos(\theta)^3 p_{xxx}(i,j,k) + 3 \cdot Cos(\theta)^2 Sin(\theta)p_{xxy}(i,j,k) + 3 \cdot Cos(\theta)Sin(\theta)^2 p_{xyy}(i,j,k) + Sin(\theta)^3 p_{yyy}(i,j,k)$$

By steering the derivatives to the local orientation, we obtain an "invariant" measure of local contrast. We can also "steer" in scale to obtain invariance to size.

Note, we can NOT steer the mixed derivatives, i.e $g_{xy}(x,y,\sigma)$

4.3. Intrinsic Orientation

For each pixel, one can calculate the orientation of maximal gradient. This orientation is equivariant with rotation. One can use this as an "intrinsic" orientation to normalize the receptive fields at any point in the image.

Local orientation: $$\theta_i(x,y,\sigma) = Tan^{-1}(\frac{P_y(x,y,\sigma)}{P_x(x,y,\sigma)})$$

Or with a Pyramid: $$\theta_i(i,j,k) = Tan^{-1}(\frac{P_y(i,j,k)}{P_x(i,j,k)})$$

Note that local orientation depends on σ!

4.4. Computing Image Derivatives with a Gaussian Pyramid

It is possible to use the Gaussian Pyramid to compute image derivatives at scale using sums and differences. These are very similar to the receptive fields observed in the visual cortex of mammals.

Let *P(x,y)* be an image and *P(i, j, k)* be a Gaussian pyramid of the image.
We can obtain a fast approximation for Gaussian derivatives by sum and difference of the samples.

$$P_x(i,j,k) \approx P(i+1,j,k) - P(i-1,j,k) = P(i,j,k) * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

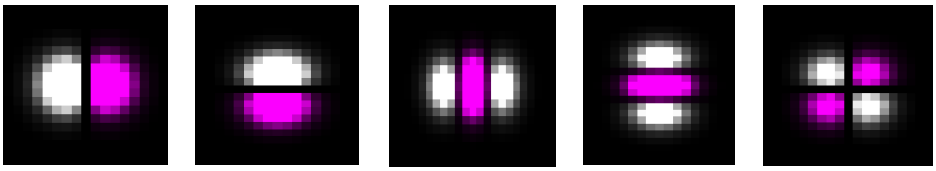
$$P_y(i,j,k) \approx P(i,j+1,k) - P(i,j-1,k) = P(i,j,k) * \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

$$P_{xx}(i,j,k) \approx P(i+1,j,k) - 2P(i,j,k) + P(i-1,j,k) = P(i,j,k) * \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$$

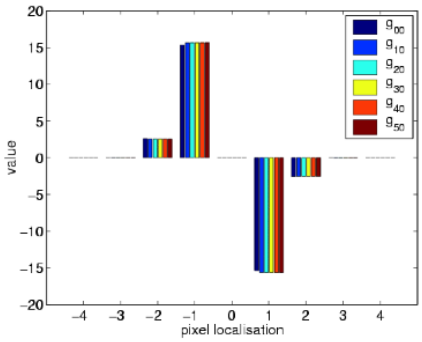
$$P_{yy}(i,j,k) \approx P(i,j+1,k) - 2P(i,j,k) + P(i,j-1,k) = P(i,j,k) * \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

$$P_{xy}(i,j,k) \approx P(i+1,j+1,k) - P(i-1,j+1,k) - P(i+1,j-1,k) + P(i-1,j-1,k) = P(i,j,k) * \begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix}$$

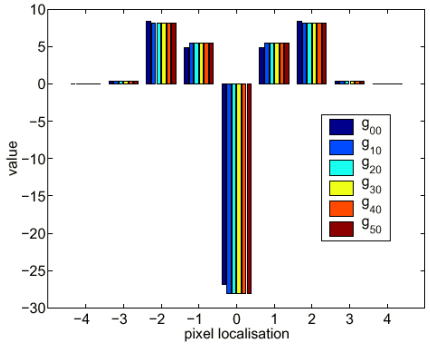
The following are some numerically evaluated examples published in the Scale Space Conference of 2003. [Crowley-Riff 2003]



$G_x(i,j,k)$ $G_y(i,j,k)$ $G_{xx}(i,j,k)$ $G_{yy}(i,j,k)$ $G_{xy}(i,j,k)$



Impulse Response for $G_x(i,j,k)$



Impulse Response for $G_{xx}(i,j,k)$

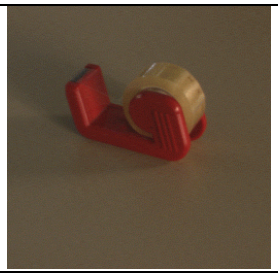
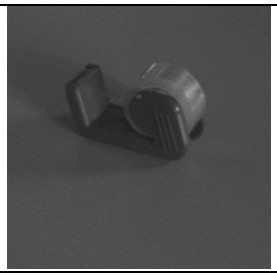
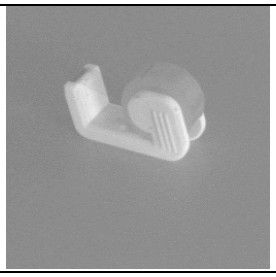
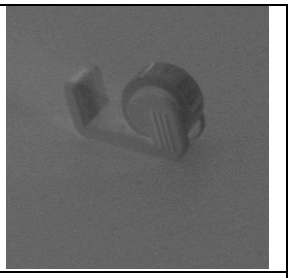
Impulse response for Gaussian derivatives for pyramid levels k=0,1,2,3,4,5.

4.5. Color Opponent Receptive Fields

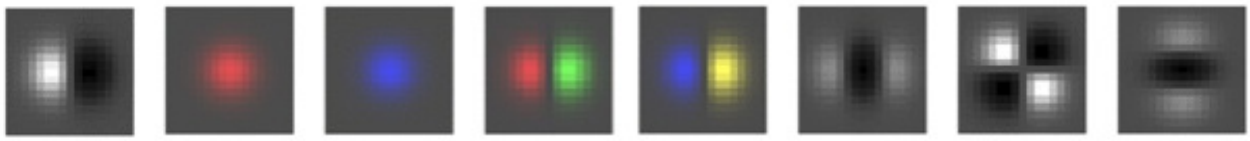
Gaussian receptive fields can be used with color images by projecting the RGB color vector to a luminance-chrominance represenation:

$$(R, G, B) \Rightarrow (L, C_1, C_2) \qquad \begin{pmatrix} L \\ C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} 0.33 & 0.33 & 0.33 \\ -0.5 & -0.5 & 1 \\ 0.5 & -0.5 & 0 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Gaussian derivatives in this space provide color opponent receptive fields.

			
RGB	B-W	R-G	R+G-B

Color opponent space can be used to build color opponent receptive fields.
We then compute 3 pyramids: $L(i, j, k)$, $C_1(i, j, k)$, and $C_2(i, j, k)$,



Examples of color opponent receptive fields.

For example, the C_2=R-B channel is very close to as skin color detector. Skin colored blobs such as hands and faces will result in strong peaks a Laplacian Pyramid computed from C_2.

The color opponent receptive fields can be used to provide general feature vectors for appearance.

5. Invariant and Equivariant Properties in Scale Space

5.1. The Laplacian of the Gaussian

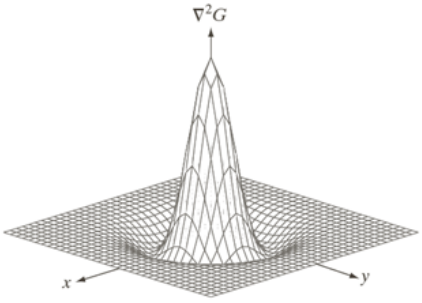
The Laplacian of the Gaussian an important derivative operator for invariant image description. The Laplacian of the Gaussian is the sum of the second derivatives.

$$\nabla^2 g(x,y,\sigma) = g_{xx}(x,y,\sigma) + g_{yy}(x,y,\sigma)$$

The equivalent digital filter is $\nabla^2 G(x,y,\sigma) = G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma)$

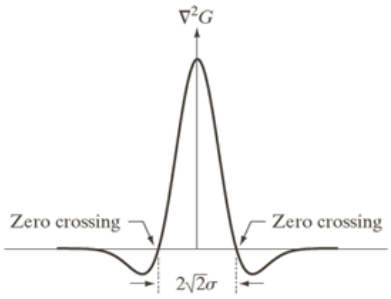
In human vision, the 2D Laplacian of the Gaussian is used to detect the natural scale of entities and thus to drive attention in the LGN. The central region acts as a kind of "spot detector" for entities of a particular scale.

For computer vision, the Laplacian of the Gaussian provides a Scale-invariant feature detector, used in the popular SIFT operator (Scale Invariant Feature Transform).



2D Plot of $\nabla^2 g(x,y,\sigma)$

Image of $\nabla^2 g(x,y,\sigma)$



1-D Cross section $\nabla^2 g(x,y,\sigma)$

The Difference of Gaussian (DoG) operator

The Gaussian is the unique solution to the diffusion equation, and as a consequence:

$$\nabla^2 g(x,y,\sigma) = g_{xx}(x,y,\sigma) + g_{yy}(x,y,\sigma) = \frac{\partial g(x,y,\sigma)}{\partial \sigma}$$

Thus, we can approximate an FIR Laplacian filter as a Difference of Gaussian filters:

$$\nabla^2 G(x,y,\sigma) \approx \left(G(x,y,\sigma_1) - G(x,y,\sigma_2)\right)$$

This is sometimes called a DoG (Difference of Gaussian) operator and can be computed very easily from a Gaussian Pyramid as a difference of levels.

It is common to use: $\nabla^2 G(x,y,\sigma) \approx G(x,y,\sqrt{2}\sigma) - G(x,y,\sigma)$

But note that from the scale property: $G(x,y,\sqrt{2}\sigma) \approx G(x,y,\sigma) * G(x,y,\sigma)$

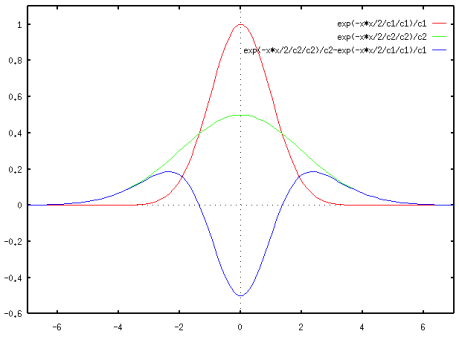
so that $\nabla^2 G(x,y,\sigma) \approx G(x,y,\sigma) * G(x,y,\sigma) - G(x,y,\sigma)$

We can compute the Laplacian of an image recursively:

$$\nabla^2 P(x,y,\sigma) = P * \left(\nabla^2 G(x,y,\sigma)\right) \approx \left(P * G(x,y,\sigma)\right) * G(x,y,\sigma) - P * G(x,y,\sigma)$$

The following illustrates this with a 1D Gaussian:

$$\nabla^2 G(x,\sigma) \approx G(x,\sqrt{2}\sigma) - G(x,\sigma)$$



In 1D the Laplacian (Blue) is the difference of a larger Gaussian (green) from a smaller Gaussian (red). Both Gaussians are normalized to sum to 1.0.

5.2. Computing a Scale Invariant Laplacian Pyramid

We can use DoG to compute the Laplacian of an image as:

$$\nabla^2 P(x,y,\sigma) = P * \left(\nabla^2 G(x,y,\sigma)\right) \approx P * G(x,y,\sqrt{2}\sigma) - P * G(x,y,\sigma)$$

Thus the Laplacian of the image can be computed directly from a difference of samples at the same image position for adjacent levels in a Half-Octave Gaussian Pyramid.

For each position *P(i, j, k)* in a Half-Octave Gaussian Pyramid, a Laplacian may be computed as:

$$L(i,j,k) = P(i\Delta x_k, j\Delta y_k, k) - P(i\Delta x_{k-1}, j\Delta y_{k-1}, k-1)$$

Note that it is necessary to re-align the samples to original image coordinates using Δx_k, Δy_k in order to assure that the image position of the two Gaussians is the same.

The resulting difference is assured to have an impulse response that sums to zero because both Gaussian Filters have been normalized to 1.

For computation with a √2 pyramid, the "scale" of the inner lobe is is the scale a level k-1: σ_{k-1}.

5.3. The Laplacian Profile

The Laplacian profile is the sequence of values for the Laplacian of the image over a range of scales at an image position, x,y.

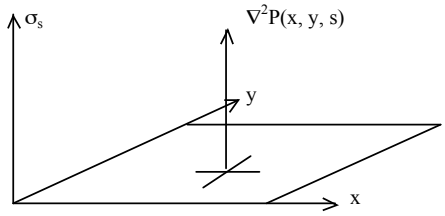
The Laplacian of the image is $\nabla^2 p(x,y,\sigma) = p * \nabla^2 g(x,y,\sigma) = p_{xx}(x,y,\sigma) + p_{yy}(x,y,\sigma)$

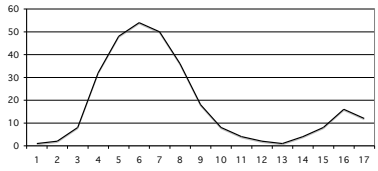
In theory, a Laplacian profile exists for every image position. However, for an image pyramid, the profile only exists for pyramid samples. For any

A sampled Laplacian Profile of the image can be computed for any

$$L(i,j,k) = P(i\Delta x_k, j\Delta y_k, k) - P(i\Delta x_{k-1}, j\Delta y_{k-1}, k-1)$$

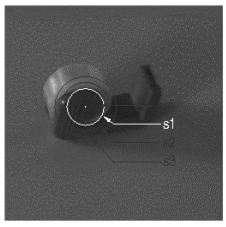
The Laplacian profile is invariant to rotation and translation and equivariant to changes in scale. Since scale is proportional to distance, the profile is equivariant to viewing distance.



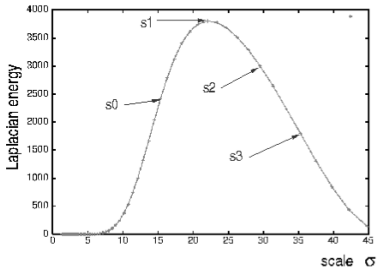


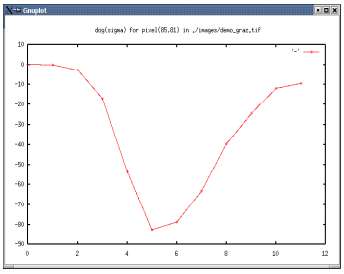
When evaluated over an exponential range of scales, the Laplacian profile is an invariant descriptor for scale.

Some examples from images (computed with a half-octave pyramid):



zero crossing of Laplacian at s1





A change in viewing distance at x, y shifts the function $\nabla^2 P(x,y,\sigma)$ in σ. The form of the profile translates in scale but remains the same. Technically this is called a "covariant".

The Laplacian profile is the basis for the Natural Interest points used in the highly popular SIFT (Scale Invariant Feature Transform). In the SIFT operator, local maxima in the Laplacian (in position and scale) serve to position the operator where there is a maximum of image signal.

The Laplacian profile can also be used to extract a very efficient operator for "time to crash" that describes the "looming reflex".

For every pixel in the image, there are one or more scales at which the Laplacian profile is a local maximum. These local maximum indicate the scale at information can be found in the image, and are referred to as the "intrinsic scales" for the pixel.

5.4. Laplacian Interest Points.

A Laplacian interest point is a local maximum within some volume in a Laplacian pyramid in space and scale.

$$(x_i, y_i, \sigma_i) = \underset{x,y,k,R}{local - \max}\{\nabla^2 P(x,y,k)\}$$

The smallest possible volume is a 3x3x3 ROI. However, because of the smoothness properties of the Laplacian, it is possible for the maximum to fall exactly between two samples, resulting in a "double-peak" of two adjacent samples with equal values. This is a common event for elongated structures. Reliability of detection can be improved by testing over a radius of 2, resulting in a 5x5x5 ROI.

Candidate local maxima can be detected using a scanning window approach over all possible 5x5x5 ROIs.

It is possible to obtain a more precise location of the peak by computing the center of gravity (moment) over a range of samples in x, y and σ.

6. Histogram of Oriented Gradients (HOG)

A local histogram of gradient orientation provides a vector of features image appearance that is relatively robust to changes in orientation and illumination.

HOG gained popularity because of its use in the SIFT feature point detector (described next). It was subsequently explored and made popular by Navneet Dalal (M2R GVR 2003) and Bill Triggs (currently at UGA Labo LJK).

The orientation of a gradient at pyramid sample (i,j,k) is:

$$\theta(i,j,k) = Tan^{-1}\left\{\frac{p_y(i,j,k)}{p_x(i,j,k)}\right\}$$

This is a number between 0 and π. We can quantize it to a value between 1 and N value by

$$a(i,j,k) = Round\left\{N \cdot \frac{\theta(i,j,k)}{\pi}\right\}$$

We can then build a local histogram for a window of size WxH, with upper left corner at i_o, j_o, k. We allocate a table of N cells: h(a). Then for each pixel *i,j* in our window:

$$\forall_{i=1}^{W} \forall_{j=1}^{H} h(a(i+i_o, j+j_o, k)) = h(a(i+i_o, j+j_o, k)) + 1$$

The result is a local feature composed of N values.
Recall that with histograms, we need around 8 samples per bin to have a low RMS error. Thus a good practice is to have N=W=H. For example N=4, W=4 and H=4. Many authors ignore this and use values such as N=8, W=4, H=4, resulting in a sparse histogram.

Remark: A fast version when N=4 replaces the inverse tangent by computing the diagonal derivatives with differences:

$$P_0(i,j,k) = P(i+1,j,k) - P(i-1,j,k)$$
$$P_{\frac{\pi}{4}}(i,j,k) = P(i+1,j+1,k) - P(i-1,j-1,k)$$
$$P_{\frac{\pi}{2}}(i,j,k) = P(i,j+1,k) - P(i,j-1,k)$$
$$P_{\frac{3\pi}{4}}(i,j,k) = P(i+1,j-1,k) - P(i-1,j+1,k)$$

To determine *a(i,j,k)* simply choose the maximum of $P_0, P_{\frac{\pi}{4}}, P_{\frac{\pi}{2}}, P_{\frac{3\pi}{4}}$

7. Scale Invariant Feature Transform (SIFT)

SIFT uses a scale invariant pyramid to compute scale invariant interest points as shown above.

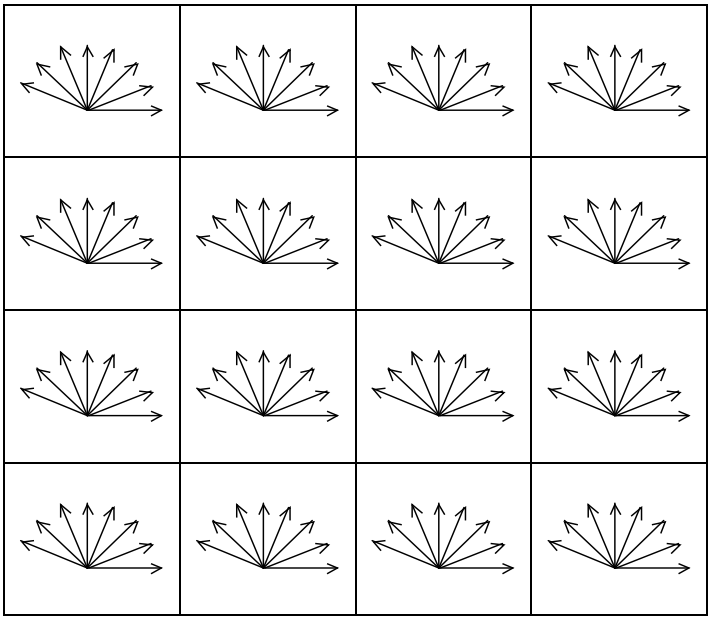
$$L_{k0} = p_1(i, j, k) - p(i, j, k)$$
$$L_{k1} = p_2(i, j, k) - p_1(i, j, k)$$
$$L_{k2} = p_3(i, j, k) - p_2(i, j, k)$$
$$L_{k3} = p_4(i, j, k) - p_3(i, j, k)$$

If $L_{k0} < L_{k1} > L_{k2}$ then the point p(i, j, k) is a natural interest point at with $\sigma = 2^{k+1/2}$
If $L_{k1} < L_{k2} > L_{k3}$ then the point p(i, j, k) is a natural interest point with $\sigma = 2^{k+1}$

For each interest point, it then computes a U x V grid of HOG detectors with N=8, W=4, H=4 at the level k
Typically U=V=4.

At level k, $\Delta i = \Delta j = 2^{k/2}$

This gives 16 x 8 = 128 features at each interest point.
This feature vector is invariant to changes in position and scale and very robust with changes in image plane rotation and illumination intensity.



Various authors experiment with other grid sizes.
For example, let the grid size be G.

G=4, W=4, H=4, N=4

16 histograms of 4 numbers =64.

8. Harris Corner Detector

Harris, Chris, and Mike Stephens. "A combined corner and edge detector." Alvey vision conference. Vol. 15. 1988.

The Harris-Stevens Corner detector is inspired from the Moravec Interest Point detector proposed in 1973 by Hans Moravec for stereo matching. Moravec used the Sum of Squared Difference (SSD) between adjacent small patches to detect interest points. In 1988, Harris and Stevens observed that this is equivalent to an auto-correlation of the image.

$$S(x,y) = \sum_{u,v} w(u,v)\big(I(u+x,v+y) - I(u,v)\big)^2$$

where $I(x,y)$ is the image,
w(x,y) is some window function, typically Gaussian.

I(u+x,v+y) can be approximated as a local Taylor Series:

$$I(u+x,v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y$$

where $I_x(x,y)$ and $I_y(x,y)$ are the local x and y derivatives

Giving $S(x,y) = \sum_{u,v} w(u,v)\big(I_x(u,v)x + I_y(u,v)y\big)^2$

Which can be written in Matrix form as: $S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix}$ where A is the "Structure Tensor"

$$A = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

With our Gaussian pyramid this is simply: $A = \begin{bmatrix} P_x^2 & P_x P_y \\ P_x P_y & P_y^2 \end{bmatrix}$

Compute the Eigenvectors of A: $\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} = R\,A\,R^T$

where λ_1 is the maximum gradient, λ_2 is the minimum gradient.

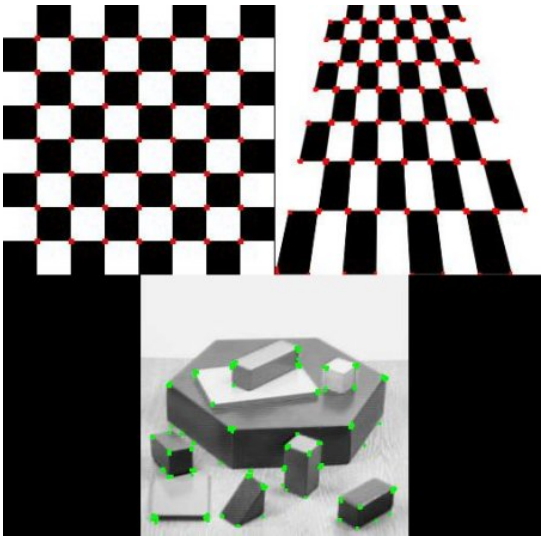
if $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$ then the point is of no interest
if $\lambda_1 \approx 0$ and $\lambda_2 >> 0$ then the point is a horizontal edge
if $\lambda_1 >> 0$ and $\lambda_2 \approx 0$ then the point is a vertical edge
if $\lambda_1 \approx \lambda_2 >> 0$ then the point is corner

To avoid computing the eigenvalues (requires a square root), we can define a measure for “corner-ness”:

$$M_c = \det(A) - \kappa \cdot Trace^2(A) = \lambda_1\lambda_2 - \kappa\left(\lambda_1 + \lambda_2\right)^2$$

where κ is a tunable sensitivity parameter.

Examples of Harris-Stevens Corners:



9. Ridge Detection.

The Eigenvalues of the Hessian provide a popular ridge detector.

The Hessian at scale s is $H(x,y,s)=\begin{pmatrix} P_{xx}(x,y,s) & P_{xy}(x,y,s) \\ P_{xy}(x,y,s) & P_{yy}(x,y,s) \end{pmatrix}$

The Eigenvalues are found by diagonalizing the Hessian.
For any point in scale space *(x, y, s)*

$$\begin{pmatrix} P_{rr} & 0 \\ 0 & P_{ss} \end{pmatrix} = R\,H\,R^T \qquad \text{where} \qquad R=\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

P_{ss} is the largest value in second derivative, while P_{rr} is the smallest.
On a ridge point, P_{rr} will be the second derivative along the ridge (close to zero) while P_{ss} will be the 2nd derivative perpendicular to the ridge.

For any 2D Matrix, the principal directions can be computed directly as

$$\cos(\theta)=\sqrt{\frac{1}{2}\left(1+\frac{P_{xx}-P_{yy}}{\sqrt{\left(P_{xx}-P_{yy}\right)^2+4P_{xy}^2}}\right)}, \qquad \sin(\theta)=\operatorname{sgn}(P_{xy})\sqrt{\frac{1}{2}\left(1-\frac{P_{xx}-P_{yy}}{\sqrt{\left(P_{xx}-P_{yy}\right)^2+4P_{xy}^2}}\right)}$$

Recall that the gradient is $\vec{\nabla}P(x,y,s)=\begin{pmatrix} P_x(x,y,s) \\ P_y(x,y,s) \end{pmatrix}=\begin{pmatrix} P*G_x(x,y,s) \\ P*G_y(x,y,s) \end{pmatrix}$

for any point *(x, y, s)*, the Gradient can be aligned with the ridge using

$$P_r=\cos(\theta)P_x-\sin(\theta)P_y$$
$$P_s=\sin(\theta)P_x+\cos(\theta)P_y$$

A positive ridge point is any point, *R(x,y,s)* that satisfies:

$P_r = 0$ and $P_{rr} \le 0$ and $|P_{rr}| \ge |P_{ss}|$

A negative ridge is any point for which

$P_r=0$ and $P_{rr} \ge 0$ and $|P_{rr}| \le |P_{ss}|$

of course, P_r will rarely be exactly zero, so we use form of approximation $P_r \approx 0$

The ridge direction at (x, y, s) is: $\cos(\theta) = \frac{P_x}{\sqrt{P_x^2 + P_y^2}}$ $\sin(\theta) = \frac{P_y}{\sqrt{P_x^2 + P_y^2}}$

A Maximal ridge is a ridge point $R(x,y,s)$ for which $\text{local} - \max_s \{\nabla^2 P(x,y,s)\}$

Examples of Maximal Ridge points: