# CSE 252C: Computer Vision III

**Lecturer: Serge Belongie**
**Scribe: Catherine Wah**

# LECTURE 5
# Basics of Image Processing

## 5.1. Introduction

Digital image processing is a broad field, with applications ranging from medical image analysis (*e.g.* radiology) to compression (jpeg, png) to consumer electronics and software (photo retouching). In this lecture, we focus on aspects of image processing with particular relevance to object recognition:

- linear filtering
- the frequency domain
- efficient implementation

We'll focus on the continuous case, for ease of exposition; peculiarities of the discrete case are addressed as needed.

[1]Department of Computer Science and Engineering, University of California, San Diego.

August 10, 2009

## 5.2. Fourier transforms

### 5.2.1. The frequency domain

We start with a motivating example we're all familiar with: blurring. How does one operationalize blurring, in a computational sense? Most commonly, this is done via *linear filtering* (convolution) with a Gaussian.

How does this come about? The key to understanding linear filtering, whether for blurring or for more general filtering (as we'll see in the case of texture analysis), is to look at the image in the *frequency domain*. We present a very brief overview, since it is a substantial topic of its own. Broadly speaking, the frequency domain can refer to so-called "wavelet analysis" as well as Fourier analysis; we'll consider both, with emphasis on the latter.

For simplicity, let's consider the 1D case, which we can think of as a single row/column of an image. Ignoring boundary conditions for the time being, since images are finite, Fourier analysis allows us to express a function $f(x)$ as a superposition of sinusoids:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega x} dw. \tag{5.1}$$

In the other direction:

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-j\omega x} dx, \tag{5.2}$$

with $e^{j\omega x} = \cos(\omega x) + j \sin(\omega x)$, $j = \sqrt{-1}$, and we write

$$f(x) \overset{\mathcal{F}}{\leftrightarrow} F(\omega)$$

to denote a *Fourier transform pair*: the lowercase $f$ is a function of a spatial variable, and the uppercase $F$ is a function of spatial frequency.

We consider an example. Suppose the image is a constant value of 1; then, by inspection, $F(\omega) = 2\pi\delta(\omega)$. This only lets through $e^{j\omega x}$ for $\omega = 0$, which is equal to 1.

Now suppose the image is a cosine: $f(x) = \cos(\omega_o x)$. Then $F(w) = \pi\delta(\omega - \omega_o) + \pi\delta(\omega + \omega_o)$, since $\cos\alpha = \frac{1}{2}(e^{j\alpha} + e^{-j\alpha})$, and so on for all suitably well-behaved functions (including all the ones we'll consider in this class). (What would this be for $\sin(\omega_o x)$?)

The take-home message is that the Fourier transform is a recipe of the sinusoids, consisting of both amplitude and phase, that when linearly combined produce a desired function. Many analytical cases are tabulated; in general, one uses the fast Fourier transform (FFT) for generic (discrete) functions (not covered in this class). An important analytical case is the Gaussian, which has the special property of being its own FT (left as an

exercise for the reader):

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/(2\sigma^2)} \overset{\mathcal{F}}{\leftrightarrow} F(\omega) = \sqrt{2\pi\sigma^2} e^{-\omega^2\sigma^2/2} \tag{5.3}$$

figures Note the inverse relationship between width in space and spatial frequency.

### 5.2.2. Gaussians as filters

Now let's return to the blurring operation. Intuitively, blurring an image results in loss of high freqency detail. In the frequency domain, we can effect this as a product between the *spectrum* (FT) of an image and (for example) a Gaussian:

figure Here the Gaussian acts as a low-pass filter.

What is the corresponding operation in the spatial doman? The celebrated "convolution theorem" (proof left as an exercise) tells us:

$$F(\omega)H(\omega) \overset{\mathcal{F}}{\leftrightarrow} f(x) * h(x), \tag{5.4}$$

where

$$f(x) * h(x) = \int_{-\infty}^{\infty} f(x')h(x - x')dx'. \tag{5.5}$$

The $x$ is called a "kernel" (in a different sense than lecture 3, but they are still related). This can be visualized as a sliding inner product over the image– a continuous version of the cross-correlation operation we saw in lecture 2 (convolution has a sign change, so the kernel is flipped).

Using the Gaussian for smoothing has many desirable properties, *e.g.* it doesn't introduce "ringing" in filtered signals (as would occur with a box-shaped filter in the frequency domain). We use this operation to eliminate noise and as preprocessing for subsampling.

More interestingly, Gaussians serve as a template upon which several interesting filters are constructed, *e.g.* by differentiating and/or linearly combining them or modulating them by sinusoids. The result can be as simple as a blurred gradient in 2D:

figures (in Matlab: conv2.m, fspecial.m for Gaussian, gradient.m)
or as complex oriented bandpass filters, such as Gabor functions (Gaussian times sine or cosine), Derivative of Gaussian (DoG), etc.

We provide an example of a filter bank (Fig. **??**) and its effect on an image (Fig. **??**). Such filters are sensitve to spots and oriented bars and edges, in a manner reminiscent of so-called "simple cells" in the striate cortex of the human visual system. These filters are very important for tasks such as edge detection, interest point detection and description, and texture analysis.

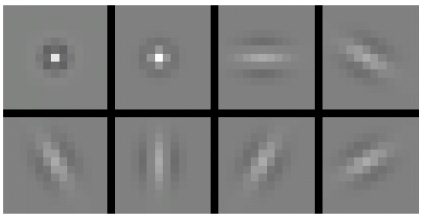

**Figure 1.** A set of 8 filters in a filter bank.

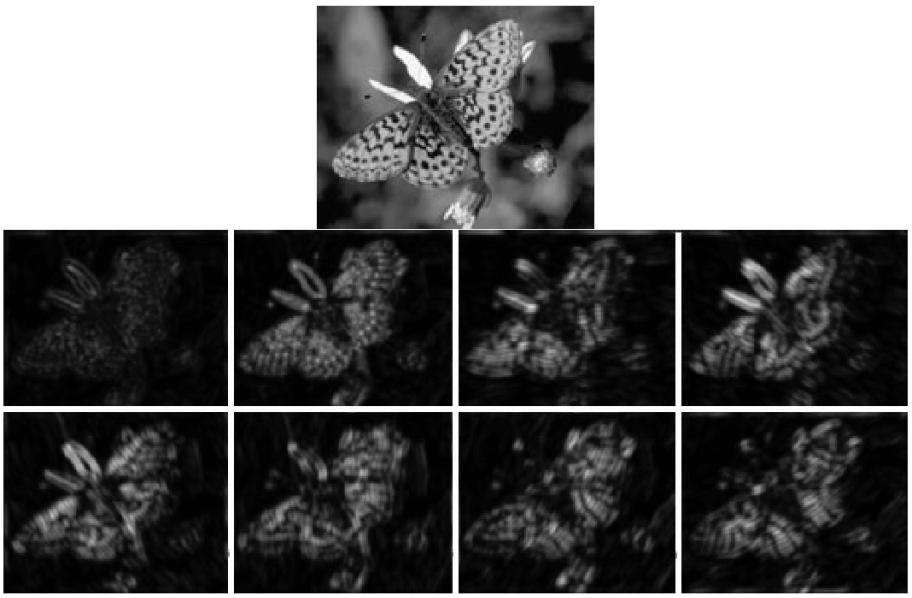

**Figure 2.** Effects of a filter bank on an image.

### 5.2.3. Implementation

One concern is computational cost (whether in spatial or frequency domain)– doing all these sliding inner products is expensive (or doing a lot of FFTs and products). What can be done about this? One strategy is to (a) exploit redundancy in the filters, and another is to (b) use filters that are cheaper to apply.

For (a), one can exploit one or more of separability, steerability, and scalability (Adelson and Freeman, 1991). Consider an example: a 2D Gaussian is exactly separable into $h(x, y) = h(x)h(y)$, or two 1D convolutions. More generally, SVD provides components of multi-pass separability (Perona, 1995).

Let's look at another example. The oriented Gaussian derivative along direction $\theta$ is exactly steerable:

$$g_\theta(x, y) = \frac{\partial g}{\partial x} \cos\theta + \frac{\partial g}{\partial y} \sin\theta \tag{5.6}$$

as shown by (Lindenberg, *J. of Applied Statistics*'94) for this and higher orders. Again, SVD extends this to general case (approximately).

What has been more popular in recent years is the other option (b). In particular, a 2001 paper by Viola and Jones on face detection popularized the use of Haar-like wavelets, which also kind of look like spots, bars, and edges, but are very boxy and consist entirely of $\{-1, 0, +1\}$.

## 5.3. Haar wavelets

Haar wavelets are an alternative orthogonal basis to the Fourier basis, with a theoretical disadvantage of not being differentiable. For applications such as fast object detection, this is not a problem.

For example, four Haar wavelets on a 4-pixel (1D) image are given by:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 0 & -1 \end{bmatrix}.$$

"Haar-like" wavelets are formed by computing differences of boxes of varying size (Fig. **??**).

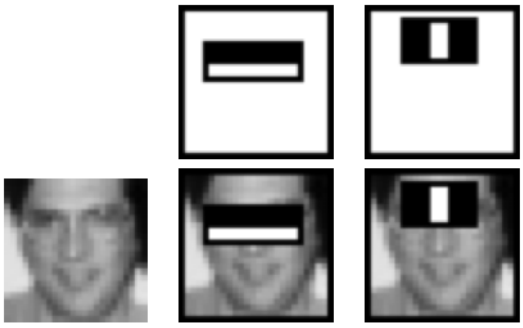

**Figure 3.** An example of Haar-like wavelets. The surrounding zeros are important here for encoding localization together with spatial frequency ( a benefit of wavelets).

### 5.3.1. Integral images

The inner product with the image can be computed extremely efficiently using *integral images*. Integral images are an instance of a classic computer science technique known as memoization, which makes clever use of a lookup table for accelerating certain kinds of computation.

The integral image $J$ is defined as:

$$J(x,y) = \sum_{x'=0}^{x} \sum_{y'=0}^{y} I(x',y'). \tag{5.7}$$

Now suppose we want to evaluate the inner product of the image with a Haar-like wavelet that looks like:

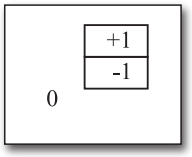


**Figure 4.** A Haar-like wavelet.

This has two rectangles whose sums we need to compute. For any rectangle (Fig. **??**), we need only to reference the $J$ array four times to compute the sum of the shaded rectangle:

$$F(x,y,w,h) = J(x,y) + J(x+w,y+h) - J(x,y+h) - J(x+w,y). \tag{5.8}$$

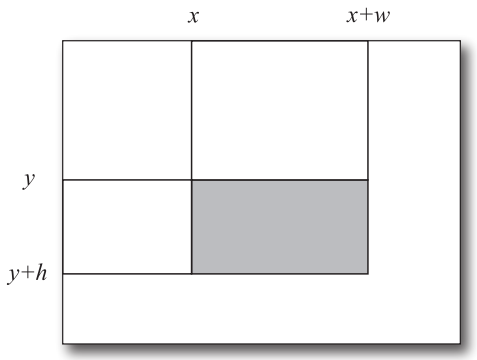


**Figure 5.** The integral image trick references four computations to find the sum of the shaded rectangle.

This trick allows for rapid (frame rate) computation of image features in applications such as Adaboost-based face detection.

In summary, while these features may not have the physiological relevance that, say, Gabors have, they have proven to be highly effective in many practical object detection tasks.