Linear Filters and Sampling

Goal: Mathematical foundations for digital image analysis, representation and transformation.

Outline:

- **Sampling Continuous Signals**
- **Linear Filters and Convolution**

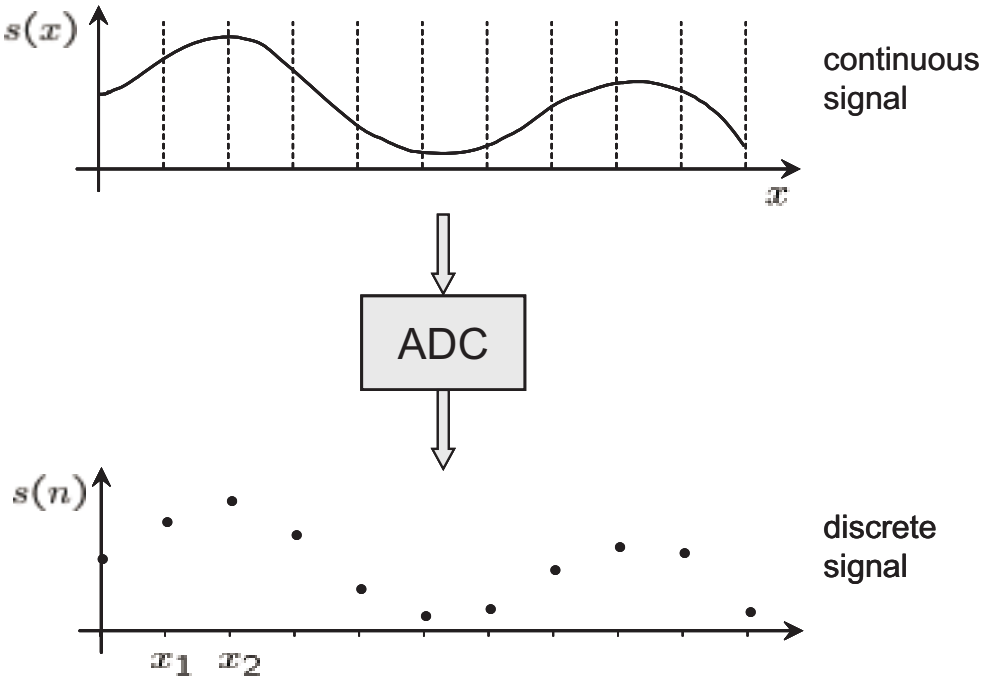
Readings: Chapters 1 and 7 of the text.

Matlab Tutorials: linSysTutorial.m, samplingTutorial.m and imageTutorial.m.

Sampling

Approximate continuous signals with a discrete sequence of *samples* from the continuous signal, taken at regularly spaced intervals.

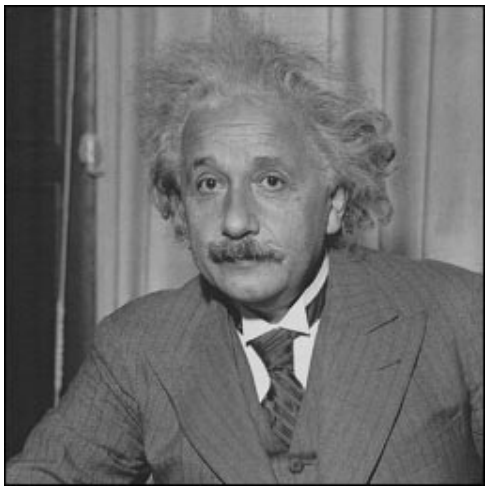
Useful approximations require that the coninuous signal be sufficiently smooth relative to the sampling interval so that one can approximately reconstruct the continuous signal (we'll discuss the Fourier sampling theorem and interpolation in greater detail later).

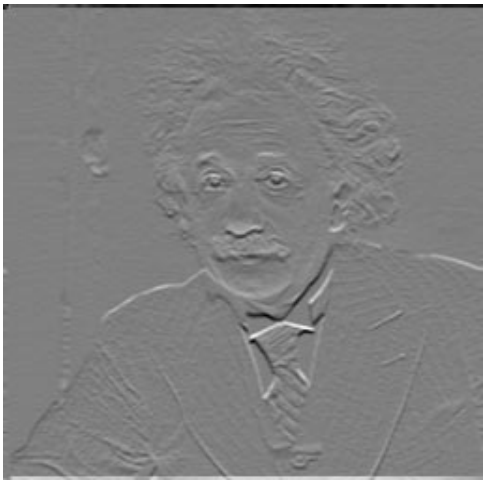


We will often express a 1D discrete signal, $s(n)$, for $n = 0...N-1$, as real-valued vector, $\vec{\mathbf{s}} \in \mathcal{R}^N$.

Introduction to Linear Filters

A filter transforms one signal into another, often to enhance certain properties (e.g., edges), remove noise, or compute signal statistics.





A transformation T, is linear iff, for inputs $s_i(n)$, responses $r_i(n) = T[s_i(n)]$, and scalars a and b, T satisfies **superposition:**

$$T[as_1(n) + bs_2(n)] = a\,T[s_1(n)] + b\,T[s_2(n)] \quad \forall a, b \in \mathcal{C}$$

In 1D, a linear filter can be represented by a matrix, A, and its response $\vec{\mathbf{r}}$ to input $\vec{\mathbf{s}}$ is given by matrix multiplication:

$$\vec{\mathbf{r}} \;=\; A\,\vec{\mathbf{s}}$$

The m^{th} element of $\vec{\mathbf{r}}$ is the inner product of the m^{th} row of A and $\vec{\mathbf{s}}$.

Shift Invariance

Often we want to apply the same operation to every point in an image (e.g., smoothing). If T is shift-invariant, then $\forall m \in \mathcal{I}$

$$r(n) = T\left[s(n)\right] \qquad \text{iff} \qquad r(n-m) = T\left[s(n-m)\right]$$

Linear, shift-invariant filters can be expressed as Toeplitz matrices (i.e., constant along diagonals), so each row is equal to the row above, but shifted right by one.

- E.g.: Let's smooth a signal by computing a weighted average of each input sample and its two neighbours with weights 0.25, 0.5, and 0.25 (i.e., with a sliding window). The corresponding matrix has the form:

$$A = \frac{1}{4}\begin{bmatrix} \ldots & 0 & 1 & 2 & 1 & 0 & \ldots & & \\ & \ldots & 0 & 1 & 2 & 1 & 0 & \ldots & \\ & & \ldots & 0 & 1 & 2 & 1 & 0 & \ldots \end{bmatrix}$$

Local filters compute responses using only small neighborhoods of pixels from the input, like the smoothing filter. For 1D signals this produces a banded Toeplitz matrix. The width of nonzero entries in a row is called the filter's *support*.

Boundary Conditions

With finite length signals we need to handle boundaries carefully.

1. Shift-invariance is preserved if we assume periodic signals and cyclical shifts. For local filters this introduces nonzero entries in the upper-right and lower-left corners of the matrix. E.g.:

$$\frac{1}{4}\begin{bmatrix} 2 & 1 & 0 & \dots & & & 1 \\ 1 & 2 & 1 & 0 & \dots & & \\ & \ddots & \ddots & \ddots & & & \\ & & & & \dots & 0 & 1 & 2 & 1 \\ 1 & & & & \dots & 0 & 1 & 2 & \end{bmatrix}$$

2. We could instead assume that the input is always zero beyond its endpoints. In practice, the number of zeros we use to *pad* the input depends on the filter's support width. If the support is M samples, then we need $M-1$ zeros on each end.

   The response is then longer than the input by $M-1$ samples, so people often just truncate the response. The transform is then:

$$\frac{1}{4}\begin{bmatrix} 2 & 1 & 0 & \dots & & & \\ 1 & 2 & 1 & 0 & \dots & & \\ & \ddots & \ddots & \ddots & & & \\ & & & & \dots & 0 & 1 & 2 & 1 \\ & & & & & \dots & 0 & 1 & 2 \end{bmatrix}$$

3. Often it is more desirable to assume a constant signal beyond the boundary, i.e., pad the ends by repeating the two end samples. This can also be viewed as a linear shift-invariant filter.

Impulse Response

One can also characterize a linear, shift-invariant operator with its impulse response, i.e., the response to an impulse, δ:

- Kronecker delta function (discrete)

$$\delta(n) = \begin{cases} 1 & n = 0 \\ 0 & \text{otherwise} \end{cases}$$

- Dirac delta function (continuous)

$$\delta(x) = 0 \quad \forall x \neq 0, \quad \text{and} \quad \int \delta(x) f(x)\, dx = f(0)$$

  for sufficiently smooth $f(x)$

In the discrete case, multiplying A by the delta function $\delta(n)$ simply extracts the first column from A:

$$\vec{\mathbf{h}} \;=\; A\,[1, 0, \ldots, 0]^t$$

$h(n)$ is called the *impulse response*. (If we pad boundaries of the input and truncate the result, then we should move the origin to the middle of the vector so we don't get a truncated impulse response!)

- If we apply A to a shifted impulse signal, then we just get a different column of A, which is a shifted version of the impulse response. Therefore, from the impulse response one can construct the matrix; i.e., it completely defines the filter.

Convolution

The conventional way to express a linear shift-invariant filter mathematically is with the convolution operator. Let the scalar w_p denote the signal value at position p, i.e., $w_p = S(p)$. Then we write $s(n)$ as

$$s(n) \;=\; \sum_{p=-\infty}^{\infty} w_p\, \delta(n-p)$$

For a linear, shift-invariant operator T it follows that

$$\begin{aligned} T\left[s(n)\right] \;&=\; T\left[\sum_{p=-\infty}^{\infty} w_p\, \delta(n-p)\right] \\ &=\; \sum_{p=-\infty}^{\infty} w_p\, T\left[\delta(n-p)\right] \\ &=\; \sum_{p=-\infty}^{\infty} w_p\, h(n-p) \end{aligned}$$

where h is the impulse response. Thus the response $r(n) = T[s(n)]$ is just a weighted sum of shifted impulse responses, that is:

$$r(n) \;=\; \sum_{p=-\infty}^{\infty} s(p)\, h(n-p) \qquad (1)$$

Eqn (1) is called convolution, and is expressed as a binary operator (often with $*$):

$$s * h \;\equiv\; \sum_{p=-\infty}^{\infty} s(p)\, h(n-p)\ .$$

For continuous signals, $h(x)$ and $s(x)$, convolution is written as

$$s * h \;=\; \int_{-\infty}^{\infty} s(\xi)\, h(x-\xi)\, d\xi$$

Properties of Convolution

Commutativity: $s * h \;=\; h * s$

Not all matrix operations commute, but this does.

Associativity: $(h_1 * h_2) * h_3 \;=\; h_1 * (h_2 * h_3)$

This is true of all matrix multiplication.

Distributivity over Addition: $(h_1 + h_2) * h_3 \;=\; h_1 * h_3 + h_2 * h_3$

This is true of all matrix multiplication.

Local Support: Often the support of the filter is limited. If $h(m)$ is only nonzero for $-M/2 \leq m \leq M/2$, then we rewrite Eqn (1) as

$$s * h \;=\; \sum_{p=-M/2}^{M/2} s(n+p)\, h(-p)\ .$$

In words, for each signal position, n, center the reflected impulse response at position n, and then take its inner product with the image. This is a better way to implement the filter than matrix multiplication!

Inverse: One way to find the inverse of a convolution operator is to create the effective Toeplitz matrix, and invert it. The inverse of a cyclic Toepliz matrix is also cyclic Toepliz, which shows that the inverse of a discrete linear shift-invariant operator, if it exists, is also linear and shift-invariant. A better way to find the inverse uses the Fourier transform.

2D Image Convolution

In 2D the convolution equation is given by

$$r(n,m) \;=\; \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} s(p,q)\, h(n-p, m-q)$$

Computational Expense: In general, 2D convolution requires $O(N^2 M^2)$ multiplications and additions where N^2 is the number of image pixels, and M^2 is the 2D support of the impulse response.

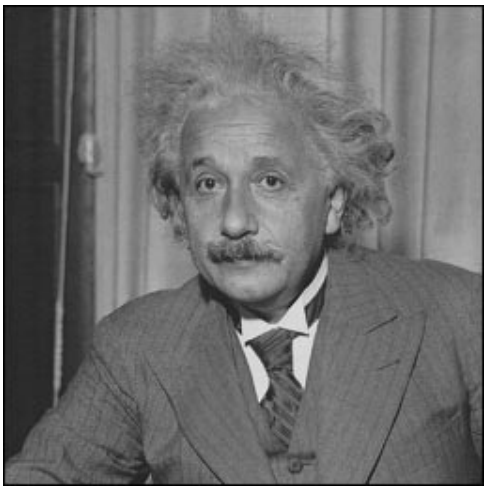
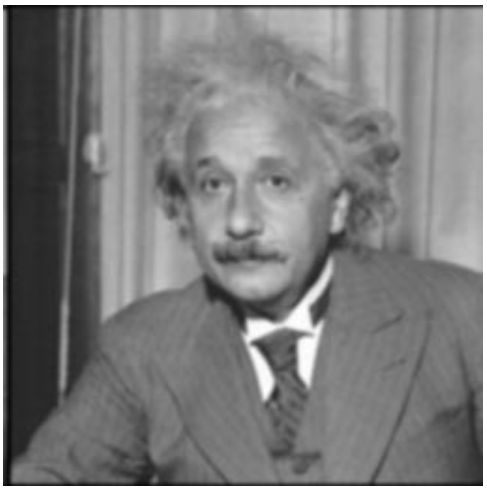
Separability: If a 2D impulse response can be expressed as $h(x,y) = h_1(x)\, h_2(y)$ for some $h_1(x)$ and some $h_2(y)$, then h is called separable. In the discrete case, the impulse response is separable if it can be written as an outer product:

$$\begin{bmatrix} \\ h[n,m] \\ \\ \end{bmatrix} \;=\; \begin{pmatrix} | \\ h_1[n] \\ | \end{pmatrix} \begin{pmatrix} — & h_2[m] & — \end{pmatrix}$$

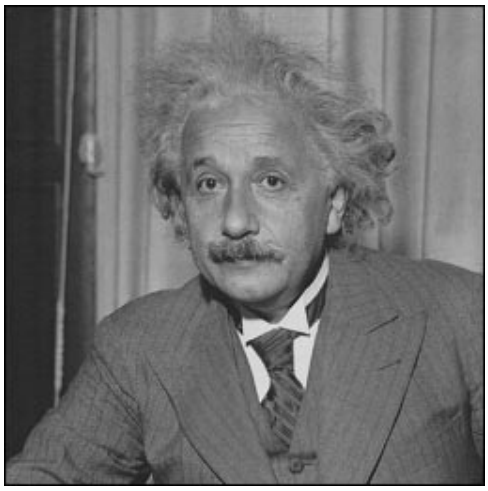
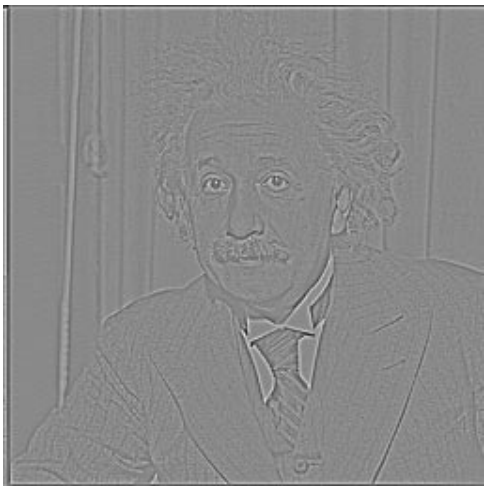
With separability, 2D convolution can be expressed as a cascade of 1D convolutions, first along the rows, and then along the columns (or along the columns and then the rows).

- Each 1D convolution, and hence the separable 2D filter, is $O(N^2 M)$. This is important if the filter support is more than 4 or 5 pixels.

Example: Smoothing Filters

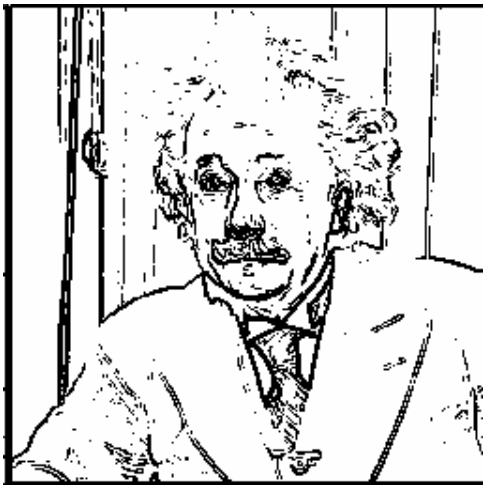
Al and a blurred version of Al are shown. The impulse response was separable, composed of the same horizontal and vertical 5-tap 1D impulse response, that is, $\frac{1}{16}(1, 4, 6, 4, 1)$.

This shows Al and the difference between Al and the blurred version of Al. The image is only non-zero where the blurred version is different from the original, i.e., where there are significant local intensity variations. The impulse response for this filter is $\delta(n) - h(n, m)$ where $h(n, m)$ is the separable blurring filter used above.

Example: Derivative Filters


Derivative filters are common in image processing. Here we use crude separable approximations to horizontal and vertical derivatives. They composed of a smoothing filter in one direction (i.e., $\frac{1}{4}(1, 2, 1)$) and a first-order central difference (i.e., $\frac{1}{2}(-1, 0, 1)$) in the other.

Sum of squared derivative responses (the squared magnitude of the image gradient at each pixel), when clipped, this gives a rough idea of where edges might be found.

Example: Down-Sampling and Up-Sampling

Down-sampling (or decimation) is the process of collapsing a signal by removing every n^{th} sample.

Up-sampling refers to the expansion of a signal by adding new samples to make it longer. One introduces n zeros in between every pair of adjacent samples in the original signal.

Both of these operators are linear.

Example: down-sampling a signal $\vec{\mathbf{s}}$ by a factor of 2 to create $\vec{\mathbf{s}}_2$.

$$\vec{\mathbf{s}}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ & & & & & & \vdots & \end{bmatrix} \vec{\mathbf{s}}$$

Example: up-sampling a signal $\vec{\mathbf{s}}$ by a factor of 2 to create $\vec{\mathbf{s}}_1$.

$$\vec{\mathbf{s}}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ & & & & & & \vdots & \end{bmatrix} \vec{\mathbf{s}}$$

Up-sampling is often a precursor to smoothing for signal interpolation.

Further Readings

Texts on Image Processing and Computer Vision

Castleman, K.R., **Digital Image Processing**, Prentice Hall, 1995

Gonzalez, R.C. and Wintz, P., **Digital Image Processing 2nd ed.**, Addison-Wesley, 1987

Rosenfeld, A. and Kak, A., **Digital Picture Processing 2nd ed.**, Academic Press, 1982

Wolberg, G., **Digital Image Warping**, IEEE Computer Society Press, 1990

Wandell, B.A., **Foundations of Vision**, Sinauer Press, 1995