Images and Transformations

Images by Pawan Sinha

1

Reading

Forsyth & Ponce, chapter 7

2

What is an image?

We can think of an **image** as a function, *f*, from R^2 to R:

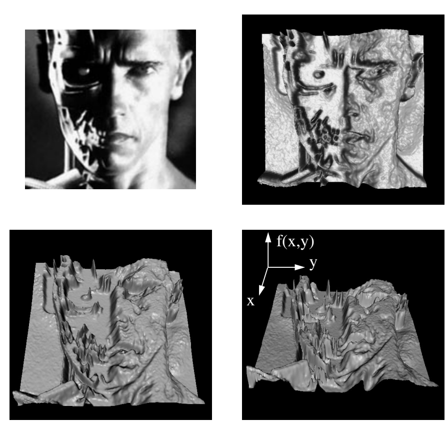
- $f(x, y)$ gives the **intensity** at position (x, y)
- Realistically, we expect the image only to be defined over a rectangle, with a finite range:
  - $f: [a,b] \times [c,d] \rightarrow [0,1]$

A color image is just three functions pasted together. We can write this as a "vector-valued" function:

$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

3

Images as functions



4

What is a digital image?

In computer vision we usually operate on **digital** (**discrete**) images:

- **Sample** the 2D space on a regular grid
- **Quantize** each sample (round to nearest integer)

If our samples are Δ apart, we can write this as:

$f[i, j]$ = Quantize{ $f(i\,\Delta, j\,\Delta)$ }

The image can now be represented as a matrix of integer values

j → ; i ↓

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

Image transformations

An **image processing** operation typically defines a new image g in terms of an existing image f.

We can transform either the domain or the range of f.

Range transformation:

$$g(x,y) = t(f(x,y))$$

What's kinds of operations can this perform?

Image processing

Some operations preserve the range but change the domain of f :

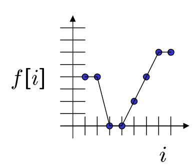
$$g(x,y) = f(t_x(x,y), t_y(x,y))$$

What kinds of operations can this perform?

Many image transforms operate both on the domain *and* the range

Linear image transforms

Let's start with a 1-D image (a "signal"): $f[i]$



$$f[i] = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

A very general and useful class of transforms are the **linear transforms** of f, defined by a matrix M

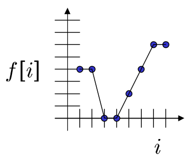
$$\begin{bmatrix} * & * & \dots & * \\ * & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \dots & * \end{bmatrix} \begin{bmatrix} * \\ * \\ \vdots \\ * \end{bmatrix} = \begin{bmatrix} * \\ * \\ \vdots \\ * \end{bmatrix}$$

$M[i,j]$ $f[i]$ $g[i]$

$$g[i] = \sum_{j=1} M[i,j] f[j]$$

Linear image transforms

Let's start with a 1-D image (a "signal"): $f[i]$

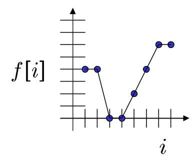


$$\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix} \quad f[i] \longrightarrow$$

$$\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0 \end{bmatrix} \quad f[i] \longrightarrow$$

Linear image transforms

Let's start with a 1-D image (a "signal"): $f[i]$



$$\begin{bmatrix} 0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0 \end{bmatrix} \quad f[i] \longrightarrow$$

$$\frac{1}{2}\begin{bmatrix} 2&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 0&1&1&0&0&0&0&0\\ 0&0&1&1&0&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&0&1&1&0&0\\ 0&0&0&0&0&1&1&0\\ 0&0&0&0&0&0&1&1 \end{bmatrix} \quad f[i] \longrightarrow$$

Linear image transforms

Another example is the discrete Fourier transform

$$F[s] = \frac{1}{N}\sum_{k=0}^{N-1} f(k)e^{-ik2\pi s/N}$$

Each row of **M** is a sinusoid (complex-valued)

The frequency increases with the row number

Linear shift-invariant filters

$$\begin{bmatrix} * & * & 0&0&0&0&0&0\\ a&b&c&0&0&0&0&0\\ 0&a&b&c&0&0&0&0\\ 0&0&a&b&c&0&0&0\\ 0&0&0&a&b&c&0&0\\ 0&0&0&0&a&b&c&0\\ 0&0&0&0&0&a&b&c\\ 0&0&0&0&0&0&*&* \end{bmatrix}$$

This pattern is very common
- same entries in each row
- all non-zero entries near the diagonal

It is known as a **linear shift-invariant filter** and is represented by a **kernel** (or **mask**) h:

$$h[i] = [a \quad b \quad c]$$

and can be written (for kernel of size 2k+1) as:

$$g[i] = \sum_{u=-k}^{-k} h[u]f[i+u]$$

The above allows negative filter indices. When you implement need to use: h[u+k] instead of h[u]

2D linear transforms

We can do the same thing for 2D images by concatenating all of the rows into one long vector:

$$\hat{f}[i] = f[\lfloor i/m \rfloor, i\%m]$$

$$\begin{bmatrix} * & * & \dots & * \\ * & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \dots & * \end{bmatrix} \begin{bmatrix} * \\ * \\ \vdots \\ * \end{bmatrix} = \begin{bmatrix} * \\ * \\ \vdots \\ * \end{bmatrix}$$

$M[i,j]$ $\hat{f}[i]$ $\hat{g}[i]$

2D filtering

A 2D image f[i,j] can be filtered by a 2D kernel h[u,v] to produce an output image g[i,j]:

$$g[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] f[i+u, j+v]$$

This is called a **cross-correlation** operation and written:

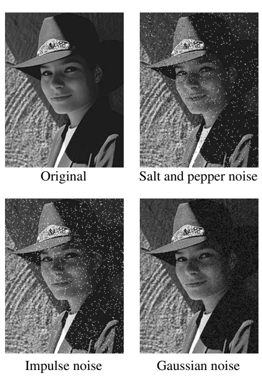
$$g = h \otimes f$$

h is called the “filter,” “kernel,” or “mask.”

The above allows negative filter indices. When you implement need to use: h[u+k,v+k] instead of h[u,v]

Noise

Filtering is useful for noise reduction...



Common types of noise:

- **Salt and pepper noise**: contains random occurrences of black and white pixels
- **Impulse noise:** contains random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Mean filtering

$f[x,y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[x,y]$

Mean filtering

$f[x,y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[x,y]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

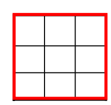
Effect of mean filters

Mean kernel

What's the kernel for a 3x3 mean filter?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$f[x,y]$



$h[u,v]$

Gaussian Filtering

A Gaussian kernel gives less weight to pixels further from the center of the window

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$f[x, y]$

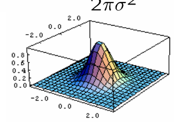
$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

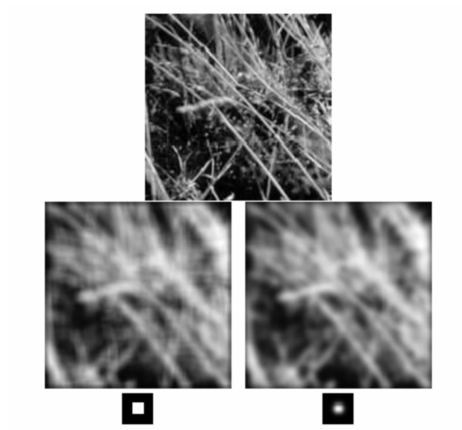
$h[u, v]$

This kernel is an approximation of a Gaussian function:

$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



Mean vs. Gaussian filtering



Convolution

A **convolution** operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

$$g[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u, v] f[i - u, j - v]$$

It is written: $g = h \star f$

Suppose H is a Gaussian or mean kernel. How does convolution differ from cross-correlation?

Convolution theorems

Let F be the 2D Fourier transform of f, H of h

Define $(h \cdot f)[x, y] = h[x, y] f[x, y]$

Convolution theorem: *Convolution* in the *spatial (image)* domain is equivalent to *multiplication* in the *frequency* (Fourier) domain.

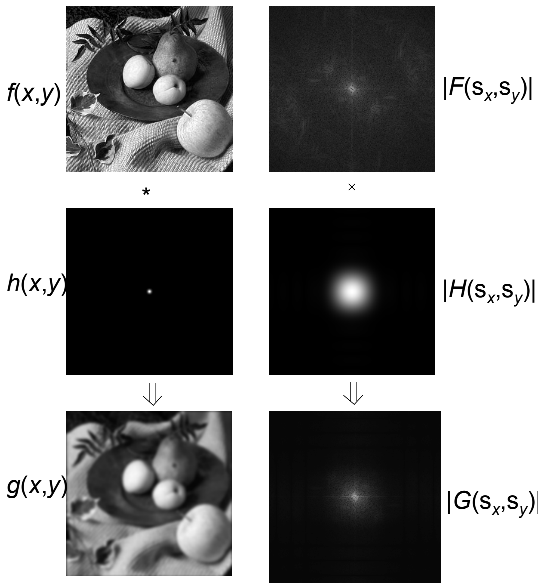
$$h \star f \leftrightarrow H \cdot F$$

Symmetric theorem: *Convolution* in the *frequency* domain is equivalent to *multiplication* in the *spatial* domain.

$$h \cdot f \leftrightarrow H \star F$$

Why is this useful?

2D convolution theorem example



25

Median filters

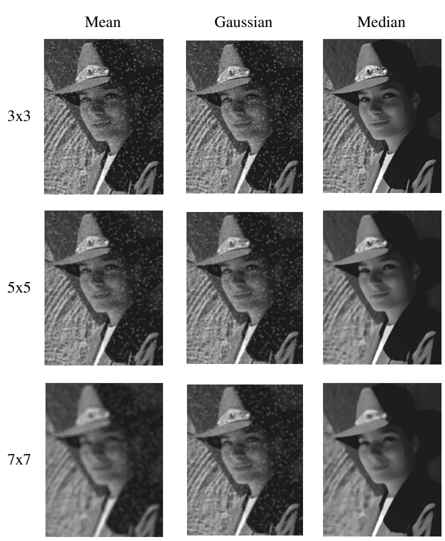
A **Median Filter** operates over a window by selecting the median intensity in the window.

What advantage does a median filter have over a mean filter?

Is a median filter a kind of convolution?

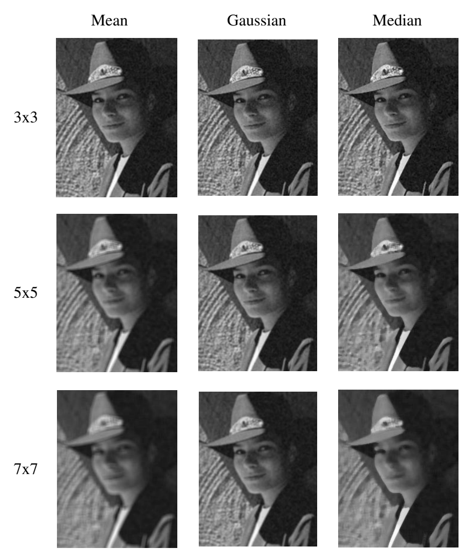
26

Comparison: salt and pepper noise



27

Comparison: Gaussian noise



28