Convolution

Adrian F. Clark <alien@essex.ac.uk>
CSEE, University of Essex

From points to regions

To date, we have looked at

- histograms
- mean and s.d.
- contrast stretching

All of these treat each pixel separately, ignoring the pixels elsewhere in the image

We shall now move from this to looking at pixels and their surrounding regions

Emphasizing small grey level differences

It would be good sometimes if we could emphasize differences in pixel value

If we multiply all pixels by a big number, it doesn't really help; we'd do better if we multiplied the dark pixel by a big number and all surrounding ones by small ones

This is **convolution**

We multiply the following mask over every 3×3 region of the image

```
-1  -1  -1
-1   8  -1
-1  -1  -1
```

and add up the products

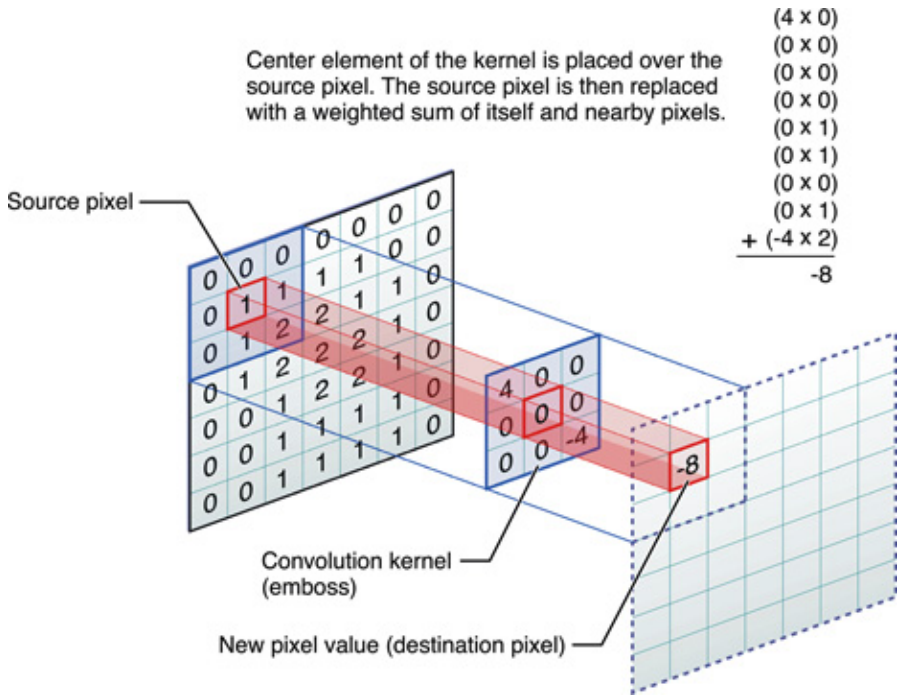
Why these values? Over a uniform region of the image — where all pixels have the same value v — we end up with

$$8v - 8v = 0$$

meaning the response is zero

This is known as the *Laplacian* mask

The convolution process



Note that you cannot overwrite the pixels of an image with the values you calculate because they would then be used when processing pixels later in the image

Doing a convolution by hand

It is easy, though sometimes time-consuming, do work out a convolution by hand

The module's Moodle site has a PDF file which has 50 randomly-generated example images, and a corresponding document giving the answers — I encourage you to make sure you can do this

An important question is what you should do at the edges of images

Although it doesn't always look good in practice, the underlying theory (Fourier theory and DFTs for the initiated) says you should really wrap the image back from the opposite edge of the image

People rarely use masks with an even dimension along the axes because it's difficult to figure out which pixel is its centre

Convolution always has to be done from one image into another

5	4	2	1	3	2	4
1	2	3	5	3	1	4
3	1	5	5	4	5	1
4	5	4	2	0	2	3
5	5	2	5	1	4	3
0	4	2	4	4	0	2
5	1	1	5	1	3	1

IMAGE (SEED: 0)

0	1	0
1	1	1
0	1	0

MASK

	11	17				
.						

RESULT

2
1
4 5 4

"cyclic wrap-around"

At top left, we have

$\cancel{5\times0} + 4\times1 + \cancel{2\times0} + 1\times1 + 2\times1 + 3\times1 + \cancel{3\times0} + 1\times1 + \cancel{5\times0}$

$= 4 + 1 + 2 + 3 + 1 = 11$ if my arithmetic is correct

$2\times1 + 2\times1 + 3\times1 + 5\times1 + 5\times1 = 17$

Blur masks

The Laplacian mask emphasizes isolated spots

What happens if we use the masks

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \qquad \begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$$

The 3×3 mask *blurs* the image a little, while the 5×5 version blurs it more

The easiest way is to understand this is to think of an image which is zero apart from a single pixel

$$\begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

and think how many places the 3×3 blur mask produces a non-zero response

Do we always use the sum after multiplying?

No — we shall look at a few alternatives

An obvious thing to use is the mean, especially with a blur mask, as that keeps the range of values in the output similar to those in the input

Another interesting statistic to use is the *median*: we sort all the values into order and choose the middle one

The median is much less affected by a value widely different from the surroundings, which is usually a consequence of noise, so *median filtering* is a good way of performing noise reduction

Other masks

The two masks H and V

$$\begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix} \qquad \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$

produce a strong response at horizontal and vertical edges respectively

The sign of the response shows whether the edge is light-to-dark or dark-to-light

If we calculate

$$\sqrt{V^2 + H^2}$$

it will tend to find edges at any orientation

Sobel's edge detector

This uses two similar masks

$$\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix} \qquad \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

combined in the same way

It can run at video rate on modest hardware

How much computation is involved in Sobel?

A naive implementation would involve 9 multiplications and 9 additions per pixel

- don't multiply by zero, just ignore those locations
- don't multiply by 1
- don't multiply by 2, either add or shift left

Together, these reduce the operation count to about 16 additions per pixel

Using other statistics

We have looked at

- different mask coefficients
- different statistics: sum/mean, median

Are there others?

- *mode*: we don't use this much
- *minimum* and *maximum*: these turn out to be useful

Using the maximum

If we use a 3×3 blur mask and take the maximum at each location

```
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 * * * * * * 0
0 0 0 0 0 0 1 1 1 1 0 0          0 0 0 0 0 * 1 1 1 1 * 0
0 0 0 0 0 0 1 1 1 1 0 0    ->    0 0 0 0 0 * 1 1 1 1 * 0
0 0 0 0 0 0 1 1 1 1 0 0          0 0 0 0 0 * 1 1 1 1 * 0
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 * * * * * * 0
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0 0 0 0 0
```

The pixels indicated by * turn from 0 to 1

This increases the size of the light region by 1 pixel all the way round

I call this an *expand* but some other authors call it a *shrink*

Using the minimum

It's easy to see that taking the minimum has the opposite effect:

```
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0          0 0 0 0 0 0 * * * * 0 0
0 0 0 0 0 0 1 1 1 1 0 0    ->    0 0 0 0 0 0 * 1 1 * 0 0
0 0 0 0 0 0 1 1 1 1 0 0          0 0 0 0 0 0 * * * * 0 0
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0 0 0 0 0
```

The pixels indicated by * turn from 1 to 0

I call this a *shrink* but some authors call it an *expand*

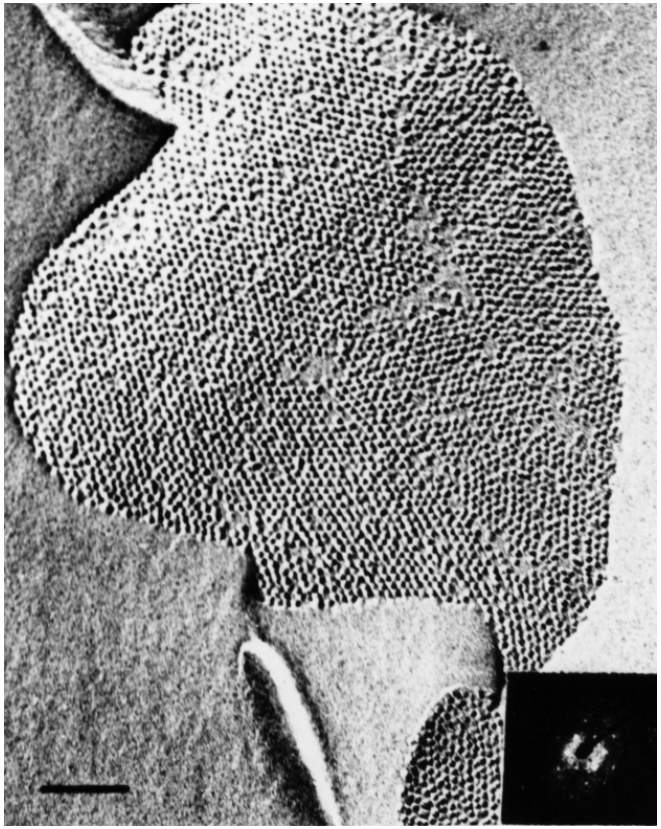
Boundary detection

If you subtract the output of a *shrink* from the original image, you're left with the boundary pixels

```
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0          0 0 0 0 0 0 1 1 1 1 0 0
0 0 0 0 0 0 1 1 1 1 0 0    ->    0 0 0 0 0 0 1 0 0 1 0 0
0 0 0 0 0 0 1 1 1 1 0 0          0 0 0 0 0 0 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0 0 0 0 0
```

Designing custom ("matched") filters

Figure 4.5 of the lecture notes shows a cell with "knobbles" on its wall which appear to be lit from above left



The mask

$$\begin{array}{ccc} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{array}$$

will show up the "knobbles"

Running this mask over the image and thresholding allows the locations of the "knobbles" to be found

Template matching

Although not convolution, this is a reasonable place in our journey into computer vision to consider it

If we know how something looks, how do we find it in an image?