Homework Assignment #3 - SOLUTIONS

Posted: Thursday, 03 October, 2013

WRITTEN EXERCISES

1) Compute by hand the spatial filtering of the following signal by the following mask/kernel:

Signal:

1	3	4	2	1	3	2	1	3	1

Mask/Kernel:

1	1	1

You only need to produce an output signal that is as long as the input signal. How do you handle the boundaries of the finite-length signal?

There are many ways to handle the boundaries of a finite-length signal. The most common ways are to extend the signal either by (i) padding with zeros, (ii) replicating the boundary value, or (iii) symmetrically mirroring the values near the boundary. Note that when our mask/kernel has only 3 elements, then to get the output value at the boundary pixel, we need to extend our signal by just one pixel (on each side). Because of this replicating and mirroring are the same. I've shown the extended pixels in blue and those pixels in the result that were affected by these extended pixels in gray. If you selected a different approach to handling the boundaries then you might have different results for these gray pixels.

Zero-padded Input:

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

Zero-padded Result:

4	8	9	7	6	6	6	6	5	4

Replicated/Mirrored Input:

1	1	3	4	2	1	3	2	1	3	1	1

Replicated/Mirrored Result:

5	8	9	7	6	6	6	6	5	5

2) Compute by hand the spatial filtering of the following signal by the following mask:

Signal:

1	3	4	2	1	3	2	1	3	1

Mask/Kernel:

1	2	1

As before, you only need to produce an output signal that is as long as the input signal.

The extended input arrays are the same as above in (1). Here are the respective outputs.

Zero-padded Result:

5	11	13	9	7	9	8	7	8	5

Replicated/Mirrored Result:

6	11	13	9	7	9	8	7	8	6

3) Suppose the following signal was put into a system with the following impulse response. What would the result be?

Signal:

1	3	4	2	1	3	2	1	3	1

Impulse response:

1	2	3

Unlike the previous two exercises, treat the signal as an infinite one with samples before/after these as zeroes. Your results should thus have 12 elements. Remember to flip the kernel appropriately depending on which method you use to calculate the convolution.

As before, I'll show the extended pixels in blue, and while the signal is infinite (infinitely padded with zeros) we'll just show the ones that are relevant. Also, I will show two outputs, one that would result from the kernel or "impulse response" as is and the other which results from rotating (flipping) the kernel. I will accept either answer depending on whether or not you rotated the kernel. I'll be talking more about impulses and convolution in the next lecture (on Friday).

Padded Input:

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

Convolution without rotating the kernel:

3	11	19	17	11	13	13	10	13	10	5	1

Convolution with rotating the kernel:

1	5	13	19	17	11	11	14	11	10	11	3

4) Compute by hand the spatial filtering of the following image and mask:

Image:

1	3	4	4	1
1	2	5	6	3
1	3	2	3	2
2	2	1	2	1
1	3	2	1	1

Mask:

1	1	1
1	2	1
1	1	1

Your result should be 5x5. State how you handle the boundaries of the image.

As before, I'll show the extended pixels in blue, on the left is the zeros-padded case, on the right, the repeated or mirror case. Inputs are above and results are below (with gray indicating those pixels impacted by the boundary extension).

0	0	0	0	0	0	0
0	1	3	4	4	1	0
0	1	2	5	6	3	0
0	1	3	2	3	2	0
0	2	2	1	2	1	0
0	1	3	2	1	1	0
0	0	0	0	0	0	0

1	1	3	4	4	1	1
1	1	3	4	4	1	1
1	1	2	5	6	3	3
1	1	3	2	3	2	2
2	2	2	1	2	1	1
1	1	3	2	1	1	1
1	1	3	2	1	1	1

8	19	28	27	15
12	24	37	36	22
12	22	28	28	19
14	19	20	17	11
9	14	13	9	6

15	27	39	36	25
15	24	37	36	28
16	22	28	28	25
18	19	20	17	15
17	20	19	13	11

The following exercise is from the Gonzales and Woods (G&W) textbook:

5) G&W 3.19b

a) Develop a procedure for computing the median of an *n* x *n* neighborhood [for reference]

b) Propose a technique for updating the median as the center of the neighborhood is moved from pixel to pixel

The idea here is to take advantage of the reusable information as you slide the median-filtering window across the image. Can you avoid having to sort/select again from scratch each time?

There are several ways of doing this. The key is that you use an approach which doesn't sort the entire neighborhood "from scratch" each time. For an *n x n* neighborhood (n^2 pixels), the cost of computing the median is the cost of sorting the n^2 pixels and then pulling out the value in the middle of the sorted list (or if *n* is even the average of the middle two). I'm not going to explicitly require a complexity analysis, but it was desirable and I will provide an informal one. The computational complexity of sorting n^2 elements is $O(n^2 \lg n^2) = O(n^2 \lg n)$. Pulling out the middle value (or values if n is even) is $O(1)$, so the complexity we want to beat is $(n^2 \lg n)$. For those not familiar with "$\lg x$", it means the base-2 logarithm "$\log_2 x$". The basic idea is that we'll have our prior neighborhood of n^2 elements and as we move to the next pixel, we'll remove elements in the n-element column that was in the prior neighborhood but not in our new one and add the elements of the n-element column that's in the new one but not in the old one. In this way we're keeping the $n(n-1)$ elements that are in common between the two neighborhoods and just touching $2n$ elements to change the old neighborhood into the new one. The key is doing this so that we can then get our new neighborhood sorted without sorting it from scratch.

Two approaches that I came up with are as follows.

The first approach uses a self-balancing binary search tree (wikipedia page for reference) which for a tree with k elements has $O(\lg k)$ for deleting an existing element and $O(\lg k)$ for inserting a new one. With a data structure that has such efficient deletion and insertion, we can remove the n elements from the old neighborhood and add the n elements from the new one, each in $O(n \lg n^2) = O(n \lg n)$. The slower part in this approach is then walking the tree to find the median element which takes time proportional to the number of elements or $O(n^2)$. Thus the total time is $O(n^2 + n \lg n) = O(n^2)$.

The second approach uses a linked list and stores the neighborhood in sorted order (we assume the prior neighborhood was already sorted). If we take the n elements to remove and sort those (which takes $n \lg n$ time) and do the same with those to insert, we can remove those n elements (walking though the linked list and removing them as their encountered) in $O(n^2)$ time. We can similarly add the n new elements in their correct locations in a total $O(n^2)$ time. Then finding the median is just indexing the middle element (or two) which is a constant time operation. Thus the total time is again $O(n \lg n + n^2) = O(n^2)$.