Subject: Re: efficient kernel or masking algorithm ?
Posted by John-David T. Smith on Wed, 29 Nov 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Richard Tyc wrote:
>
> WOW, I need to look at these equations over about a dozen times to see what
> is going on ?
>
> I have been struggling with the variance of an nxn window of data, INCLUDING
> central pixel
>
> ;mean of the neighboring pixels (including central)
> mean=smooth(arr,n)
> ;square deviation from that mean
> sqdev=(arr-mean)^2
> ;variance of an nxn window of data, INCLUDING central pixel
> var=(smooth(sqdev,n)*n^2-sqdev)/(n^2-1)
>

Almost right. Try:

var=smooth(sqdev,n)*n^2/(n^2-1)

but this still won't yield exactly what you're after, but maybe you're
after the wrong thing ;)

What this computes is a smoothed box variance, not a true box variance,
since the mean you are using changes over the box (instead of
subtracting the mean value at the central pixel from each in the box, we
subtract the box mean value at *that* pixel). Usually, this type of
variance is a more robust estimator, e.g. for excluding outlier pixels,
etc. (in which case you probably should exclude the central pixel after
all to avoid the chicken and egg problem with small box sizes). If you
really want the true variance, you're probably stuck with for loops,
preferrably done in C and linked to IDL.

This reminds me of a few things I've been thinking about IDL recently.
Why shouldn't *all* of these smooth type operations be trivially
feasible in IDL. Certainly, the underlying code required is simple.
Why can't we just say:

a=smooth(b,n,/VARIANCE)

to get a true box variance, or

a=smooth(b,n,/MAX)

to get the box max. Possibilities:
*MEAN (the current default)
*TOTAL (a trivial scaling of mean),
*VARIANCE
*MEDIAN (currently performed by the median function, in a addition to its normal duties. To see why this is strange, consider that total() doesn't have an optional "width" to perform neighborhood filtering).
*MIN
*MAX
*MODE
*SKEW
etc.

To be consistent, these should all operate natively on the input data type (float, byte, long, etc. -- like smooth() and convol() do, but like median() does not!), and should apply consistent edge conditions activated by keywords. These seem like simple enough additions, and would require much reduced chicanery.

While I'm on the gripe train, why shouldn't we be able to consistently perform operations along any dimension of an array we like with relevant IDL routines. E.g., we can total along a single dimension. All due respect to Craig's CMAPPLY function, but some of these things should be much faster. Resorting to summed logarithms for multiplication is not entirely dubious, but why shouldn't we be able to say:

col_max=max(array,2,POS=mp)

and have mp be a list of max positions, indexed into the array, and rapidly computed? While we're at it, why not

col_med=median(array,2,POS=mp)

IDL is an array based language, but it conveniently forgets this fact on occassion. Certainly there are compatibility difficulties to overcome to better earn this title, but that shouldn't impede progress.

JD

--
J.D. Smith | WORK: (607) 255-6263
Cornell Dept. of Astronomy | (607) 255-5842
304 Space Sciences Bldg. | FAX: (607) 255-5875
Ithaca, NY 14853 |

Subject: Re: efficient kernel or masking algorithm ?

Posted by Richard Tyc on Wed, 29 Nov 2000 08:00:00 GMT
View Forum Message <> Reply to Message

WOW, I need to look at these equations over about a dozen times to see what is going on ?

I have been struggling with the variance of an nxn window of data, INCLUDING central pixel

```
;mean of the neighboring pixels (including central)
mean=smooth(arr,n)
;square deviation from that mean
sqdev=(arr-mean)^2
;variance of an nxn window of data, INCLUDING central pixel
var=(smooth(sqdev,n)*n^2-sqdev)/(n^2-1)
```

This doesn't seem correct with test samples ? (Only difference is mean and division by n^2-1 ??)

Thanks JD

Rich

J.D. Smith <jdsmith@astro.cornell.edu> wrote in message news:3A25758E.A83B10CA@astro.cornell.edu...
>
> Oh my this is a common topic lately. See my recent posts in a thread
> with title "Array Manipulations". Here's the good stuff:
>
> ; the nxn window total
> total=smooth(arr,n)*n^2
> ; the nxn window total not including central pixel
> neighbors=smooth(arr,n)*n^2-arr
> ; the mean of the neighboring pixels (excluding central)
> neighmean=(smooth(arr,n)*n^2-arr)/(n^2-1)
> ; the square deviation from that mean
> sqdev=(arr-neighmean)^2
> ; the variance of an nxn window of data, excluding central pixel
> imvar=(smooth(sqdev,n)*n^2-sqdev)/(n^2-2)
>
> Take a look at the "EDGE*" keywords too, if you care about what happens
> near the borders.
>
> JD
>
> --
> J.D. Smith | WORK: (607) 255-6263
> Cornell Dept. of Astronomy | (607) 255-5842

> 304 Space Sciences Bldg. | FAX: (607) 255-5875
> Ithaca, NY 14853 |

Subject: Re: efficient kernel or masking algorithm ?
Posted by John-David T. Smith on Wed, 29 Nov 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Richard Tyc wrote:
>
> I need to apply a smoothing type kernel across an image, and calculate the
> standard deviation of the pixels masked by this kernel.
>
> ie. lets say I have a 128x128 image. I apply a 3x3 kernel (or simply a
> mask) starting at [0:2,0:2] and use these pixels to find the standard
> deviation for the center pixel [1,1] based on its surrounding pixels, then
> advance the kernel etc deriving a std deviation image essentially.
> I can see myself doing this 'C' like with for loops but does something exist
> for IDL to do it better or more efficiently ?
>
> Rich

Oh my this is a common topic lately. See my recent posts in a thread
with title "Array Manipulations". Here's the good stuff:

```
; the nxn window total
total=smooth(arr,n)*n^2
; the nxn window total not including central pixel
neighbors=smooth(arr,n)*n^2-arr
; the mean of the neighboring pixels (excluding central)
neighmean=(smooth(arr,n)*n^2-arr)/(n^2-1)
; the square deviation from that mean
sqdev=(arr-neighmean)^2
; the variance of an nxn window of data, excluding central pixel
imvar=(smooth(sqdev,n)*n^2-sqdev)/(n^2-2)
```

Take a look at the "EDGE*" keywords too, if you care about what happens
near the borders.

JD

--
J.D. Smith | WORK: (607) 255-6263
Cornell Dept. of Astronomy | (607) 255-5842
304 Space Sciences Bldg. | FAX: (607) 255-5875
Ithaca, NY 14853 |

Subject: Re: efficient kernel or masking algorithm ?
Posted by Struan Gray on Thu, 30 Nov 2000 08:00:00 GMT
View Forum Message <> Reply to Message

J.D. Smith, jdsmith@astro.cornell.edu writes:

> If you really want the true variance, you're
> probably stuck with for loops,
> preferrably done in C and linked to IDL.

There was a thread back in August called 'Standard Deviation' where we discussed this a bit. The best i could come up with was to use the SHIFT function, which for a 3x3 kernal looks like this:

```
;*******************************

function smg_imageSD, image

  fimage = float(image)
  localmean = smooth(fimage, 3)
  sum = (fimage - localmean)^2
  sum = temporary(sum) + $
    shift((fimage - shift(localmean, 1, 1))^2,-1,-1)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean, 0, 1))^2, 0,-1)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean,-1, 1))^2, 1,-1)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean, 1, 0))^2,-1, 0)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean,-1, 0))^2, 1, 0)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean, 1,-1))^2,-1, 1)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean, 0,-1))^2, 0, 1)
  sum = temporary(sum) + $
    shift((fimage - shift(localmean,-1,-1))^2, 1, 1)

  sum = sqrt(temporary(sum)/8)

  dims = size(sum, /dim)
  sum[0,*] = 0.0
  sum[*,0] = 0.0
  sum[dims(0)-1,*] = 0.0
  sum[*,dims(1)-1] = 0.0

  return, sum
end
```

```
;*******************************
```

This handles the edges crudely, but a mixture of padding the original image and using the right keywords when doing the initial smooth allows more complex behaviours.

Obviously, it would be possible to generalise the shifting and final sqrt for other kernal sizes, and to add all sorts of checking for different data types.

It's reasonably fast.

Struan
