Subject: Re: efficient kernel or masking algorithm ? UPDATE
Posted by Martin Downing on Sun, 25 Feb 2001 18:44:43 GMT
View Forum Message <> Reply to Message

From thread: http://cow.physics.wisc.edu/~craigm/idl/archive/msg03957.htm l
a.. Date: Wed, 29 Nov 2000 16:30:54 -0500
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

I was wandering through new Craig's IDL archive site (which is brilliant by
the way) and came across this question asking for an efficient way of
calculating the loacal standard deviation in an array. It seemed to me that
the thread had not reached a full solution so perhaps some of you might be
interested in this method which is very fast. It is based on the crafty
formula for variance:
  variance = (sum of the squares)/n + (square of the sums)/n*n

[apologies if this is going over old ground !]

```
function IMAGE_VARIANCE , image, halfWidth, MEAN=av_im,  $
       NEIGHBOURHOOD=NEIGHBOURHOOD,$
       POPULATION_ESTIMATE=POPULATION_ESTIMATE
;+
; NAME:
; IMAGE_VARIANCE
;
; PURPOSE:
;   This function calculates the local-neighbourhood statistical variance.
I.e. for each array element a the variance
;   of the neighbourhood of +- halfwidth is calculated.
;  The routine avoids any loops and so is fast and "should" work for any
dimension of array
;
; CATEGORY:
; Image Processing
;
```

```
; CALLING SEQUENCE:
;
; Result = IMAGE_VARIANCE(Image, HalfWidth)
;
; INPUTS:
; Image : the array of which we calculate the variance. Can be any dimension
;   HalfWidth:  the half width of the NEIGHBOURHOOD, indicates we are
looking at a
;        neigborhood +/- N from the pixel in each dimension
;
; OPTIONAL INPUTS:
; Parm2: Describe optional inputs here. If you don't have any, just
;  delete this section.
;
; KEYWORD PARAMETERS:
;
; NEIGHBOURHOOD: calculate for the NEIGHBOURHOOD only not the central pixel.
;
; POPULATION_ESTIMATE: return the population estimate of variance, not the
sample variance
;
; OUTPUT:
;  returns an array of same dimensions as input array in which each pixel
represents the local variance centred at that position
;
; OPTIONAL OUTPUTS:
; MEAN_IM: set to array of local area mean, same dimensionality as input.
;
; RESTRICTIONS:
;    Edges are dealt with by replicating border pixels this is likely to
give an underestimate of variance in these regions
;
; PROCEDURE:
;  Based on the formula for variance:
;  var = (sum of the squares)/n + (square of the sums)/n*n
;
; EXAMPLE:
; Example of simple statistical-based filter for removing spike-noise
;
;     var_im =  image_variance(image,  5, mean=mean_im, /neigh)
;     zim = (image-mim)/sqrt(var_im)
;     ids = where(zim gt 3, count)
;     if count gt 0 then image[ids] = mean_im[ids]
;
; MODIFICATION HISTORY:
;  Written by: Martin Downing, 30th September 2000
; m.downing@abdn.ac.uk
;-
```

```
 ; full mask size as accepted by SMOOTH()
 n = halfWidth*2+1

 ; this keyword to SMOOTH() is always set
 EDGE_TRUNCATE= 1
 ; sample size
    m = n^2
 ; temporary double image copy to prevent overflow
 im = double(image)
 ; calc average
 av_im = smooth(im, n, EDGE_TRUNCATE=EDGE_TRUNCATE)
 ; calc squares image
 sq_im = temporary(im)^2
 ; average squares
 asq_im = smooth(sq_im, n, EDGE_TRUNCATE=EDGE_TRUNCATE)

 if  keyword_set(NEIGHBOURHOOD) then begin
  ; remove centre pixel from estimate
  ; calc neighbourhood average (removing centre pixel)
  av_im = (av_im*m - image)/(m-1)
  ; calc neighbourhood average of squares (removing centre pixel)
  asq_im = (asq_im*m - temporary(sq_im))/(m-1)
  ; adjust sample size
   m = m-1
 endif

 var_im =  temporary(asq_im) - (av_im^2)
 if keyword_set(POPULATION_ESTIMATE) then begin
  var_im = var_im *( double(m)/(m-1))
 endif

 return, var_im

end
```

Martin Downing,
Clinical Research Physicist,
Orthopaedic RSA Research Centre,
Woodend Hospital,
Aberdeen, AB15 6LS.
m.downing@abdn.ac.uk

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

Martin Downing,
Clinical Research Physicist,
Orthopaedic RSA Research Centre,
Woodend Hospital,
Aberdeen, AB15 6LS.

m.downing@abdn.ac.uk