Subject: Re: Majority Filter
Posted by Jeremy Bailin on Fri, 08 Mar 2013 01:46:53 GMT
View Forum Message <> Reply to Message

On 3/7/13 2:18 PM, Coyote wrote:
> Folks,
>
> Has anyone ever written a majority image filter in IDL? I wish to filter out small incorrectly classified pixels floating in a sea of correctly classified pixels in an image, before processing the image further. These pixels have classification numbers, not data values. I can imagine doing it in a FOR loop (or two), but these images are large and I have a lot of them. :-(
>
> Cheers,
>
> David
>

Here's a quick stab at it.

It does have a for loop, but it's just a loop through the classification values rather than a loop through pixels. You could always try using the double-histogram-trick if that's a problem. Also, it doesn't worry about edge effects (although it almost always gets the right answer even on the edge because of how IDL deals with out-of-bounds indexing in these cases), and it only worries about the majority-of-pixel criterion and not the contiguity criterion.

The basic idea is that it uses histogram reverse indices to collect pixels of a given value, then compares each pixel in the reverse index list to all of the pixels that it would be compared to for deciding the majority.

```
;+
; NAME:
;    JBMAJORITYFILTER
;
; USAGE:
;    Result = JBMAJORITYFILTER(Data)
;
; INPUTS:
;    Data:   2D array to be filtered
;
; OUTPUTS:
;    Result will be the input Data with values replaced if 2 or more of
the neighboring
;    4 pixels have identical values. If the neighbors are 2-and-2, then
the higher value
```

```
;    will be used.
;
; RESTRICTIONS:
;    Does not apply the contiguity criterion of the true majority
filter. Does not
;    correctly deal with edge effects (but usually gets the right answer
anyways).
;
; MODIFICATION HISTORY:
;   Written by: Jeremy Bailin
;   7 March 2013
;
;-
function jbmajorityfilter, data

newdata = data   ; this will get its values replaced

valuehist = histogram(data, omin=om, reverse_indices=ri)
nvalues = n_elements(valuehist)

; loop through the values and deal with each one independently
for vi=0l,nvalues-1 do if valuehist[vi] gt 0 then begin
   val = vi+om   ; the actual value for this histogram bin - add the
minimum back in

   ; build a list of all neighbours-of-neighbours of each pixel. in
other words, for the
   ; "up" version, it's a list of all the pixels that this one will be
compared with
   ; to determine whether to replace the "up" pixel

   pix_with_this_val = ri[ri[vi]:ri[vi+1]-1]
   xy_with_this_val = array_indices(data, pix_with_this_val)  ; turn
into x,y pairs

   ; order of this: top, left, right
   ; these are now valuehist[vi] x 3 arrays of the x and y pixels of all
   ; neighbours-of-top-neighbours
   ; NOTE THAT WE HAVE NOT DEALT WITH EDGE EFFECTS HERE!!
   up_nofn_x = [xy_with_this_val[0,*], xy_with_this_val[0,*]-1,
xy_with_this_val[0,*]+1]
   up_nofn_y = [xy_with_this_val[1,*]-2, xy_with_this_val[1,*]-1,
xy_with_this_val[1,*]-1]
   ; check for trailing-dimension-truncation issues
   if valuehist[vi] eq 1 then begin
     up_nofn_x = reform(up_nofn_x, 1, 3)
     up_nofn_y = reform(up_nofn_y, 1, 3)
   endif
```

```
   ; also figure out what the coordinates of the up neighbour are
   up_n_x = xy_with_this_val[0,*]
   up_n_y = xy_with_this_val[1,*]-1

   ; check if this is a majority replacement for the up neighbour - if so,
   ; we need at least one neighbour to agree with this value
   up_majority = where(total(/int, data[up_nofn_x, up_nofn_y] eq val, 1)
ge 1, nreplace)

   ; and replace them in the new array
   if nreplace gt 0 then newdata[up_n_x[up_majority],
up_n_y[up_majority]] = val

   ; do the same for the down, left, and right neighbours
   left_nofn_x = [xy_with_this_val[0,*]-2, xy_with_this_val[0,*]-1,
xy_with_this_val[0,*]-1]
   left_nofn_y = [xy_with_this_val[1,*], xy_with_this_val[1,*]-1,
xy_with_this_val[1,*]+1]
   if valuehist[vi] eq 1 then begin
     left_nofn_x = reform(left_nofn_x, 1, 3)
     left_nofn_y = reform(left_nofn_y, 1, 3)
   endif
   left_n_x = xy_with_this_val[0,*]-1
   left_n_y = xy_with_this_val[1,*]
   left_majority = where(total(/int, data[left_nofn_x, left_nofn_y] eq
val, 1) ge 1, nreplace)
   if nreplace gt 0 then newdata[left_n_x[left_majority],
left_n_y[left_majority]] = val

   right_nofn_x = [xy_with_this_val[0,*]+2, xy_with_this_val[0,*]+1,
xy_with_this_val[0,*]+1]
   right_nofn_y = [xy_with_this_val[1,*], xy_with_this_val[1,*]-1,
xy_with_this_val[1,*]+1]
   if valuehist[vi] eq 1 then begin
     right_nofn_x = reform(right_nofn_x, 1, 3)
     right_nofn_y = reform(right_nofn_y, 1, 3)
   endif
   right_n_x = xy_with_this_val[0,*]+1
   right_n_y = xy_with_this_val[1,*]
   right_majority = where(total(/int, data[right_nofn_x, right_nofn_y]
eq val, 1) ge 1, nreplace)
   if nreplace gt 0 then newdata[right_n_x[right_majority],
right_n_y[right_majority]] = val

   down_nofn_x = [xy_with_this_val[0,*], xy_with_this_val[0,*]-1,
xy_with_this_val[0,*]+1]
   down_nofn_y = [xy_with_this_val[1,*]+2, xy_with_this_val[1,*]+1,
xy_with_this_val[1,*]+1]
```

```
   if valuehist[vi] eq 1 then begin
     down_nofn_x = reform(down_nofn_x, 1, 3)
     down_nofn_y = reform(down_nofn_y, 1, 3)
   endif
   down_n_x = xy_with_this_val[0,*]
   down_n_y = xy_with_this_val[1,*]+1
   down_majority = where(total(/int, data[down_nofn_x, down_nofn_y] eq
val, 1) ge 1, nreplace)
   if nreplace gt 0 then newdata[down_n_x[down_majority],
down_n_y[down_majority]] = val

endif

return, newdata

end
```
