Subject: Re: faster convol on local subsets?
Posted by Yngvar Larsen on Mon, 05 Dec 2011 10:54:10 GMT
View Forum Message <> Reply to Message

On Dec 5, 1:37 am, Andre <note....@gmail.com> wrote:
> Hello experts,
>
> Maybe somebody has an easy solution for this?
> I have a 2D array (img) and want the filter response from kernels that vary according to the image position. In a second array (loc, same dimensions as img) I have the information which kernel should be used at each pixel. My current approach is to first convolve the full image with the j-th kernel and take the response only at the positions with the current j indexed in the loc array:
>
> for j=0, n do begin
> kernel=kernel_store[*,*,j]
> response_temp = convol(img, kernel, /edge_zero, /NAN)
> index=where(loc eq j)
> if (index[0] gt -1)then response[index]=response_temp[index]
> endfor
>
> I works fine, but it is relatively slow and I wonder if there is a smarter (faster) to apply only the convolutions that are really needed?
>
> Thanks in advance for any help!

Yes, it seems like IDL does not implement 2D convolution very
efficiently. I found out that a straight forward implementation by
zeropadding to a power-of-2 length followed by multiplication in the
FFT domain is much faster unless the convolution kernel is very small.
Something like this (when /EDGE_ZERO and /NORMALIZE is set, some more
work for other EDGE_* keywords):

```
sizeA = size(array, /dimensions)
sizeB = size(kernel, /dimensions)

dim1 = sizeA[0] + sizeB[0] - 1
dim2 = sizeA[1] + sizeB[1] - 1

s1 = 2L^ceil(alog(dim1)/alog(2))
s2 = 2L^ceil(alog(dim2)/alog(2))

A = dcomplexarr(s1, s2)
B = dcomplexarr(s1, s2)

A[0,0] = array
B[0,0] = kernel
```

```
  convol = fft(fft(A)*fft(B), /inverse)*s1*s2
  convol = convol[sizeB[0]/2:sizeB[0]/2+sizeA[0]-1, $
                  sizeB[1]/2:sizeB[1]/2+sizeA[1]-1]

  convol = double(convol)/total(abs(kernel))
```

--
Yngvar
