Subject: Re: faster convol on local subsets?
Posted by ben.bighair on Mon, 05 Dec 2011 01:19:13 GMT
View Forum Message <> Reply to Message

On Dec 4, 7:37 pm, 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!

Hi,

Since the value of loc doesn't change, you might try precomputing the values of index in loc for each step, j. That would save a two full scans of you loc array at each iteration (one for loc eq j and one for WHERE). Instead, use HISTOGRAM with its REVERSE_INDICES to get them sorted and tagged. Then for each iteration step use David Fanning's REVERSEINDICES to grab the correct indices in loc.

So, it might look like...

```
locHist = HISTOGRAM(loc, min = 0, REVERSE_INDICES = ri)

for j=0, n do begin
    kernel=kernel_store[*,*,j]
    response_temp = convol(img, kernel, /edge_zero, /NAN)
    index = REVERSEINDICES(ri, j, COUNT =
count)
    if (count GT 0)then response[index]=response_temp[index]
endfor
```

Cheers,

Ben

http://www.idlcoyote.com/programs/reverseindices.pro

---

Subject: Re: faster convol on local subsets?
Posted by Jeremy Bailin on Mon, 05 Dec 2011 03:02:43 GMT
View Forum Message <> Reply to Message

On 12/4/11 8:19 PM, ben.bighair wrote:
> On Dec 4, 7:37 pm, 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!
>
> Hi,
>
> Since the value of loc doesn't change, you might try precomputing the
> values of index in loc for each step, j. That would save a two full
> scans of you loc array at each iteration (one for loc eq j and one for
> WHERE). Instead, use HISTOGRAM with its REVERSE_INDICES to get them
> sorted and tagged. Then for each iteration step use David Fanning's
> REVERSEINDICES to grab the correct indices in loc.
>
> So, it might look like...
>
> locHist = HISTOGRAM(loc, min = 0, REVERSE_INDICES = ri)
>
> for j=0, n do begin

> kernel=kernel_store[*,*,j]
> response_temp = convol(img, kernel, /edge_zero, /NAN)
> index = REVERSEINDICES(ri, j, COUNT =
> count)
> if (count GT 0)then response[index]=response_temp[index]
> endfor
>
> Cheers,
> Ben
>
>
>
> http://www.idlcoyote.com/programs/reverseindices.pro
>

You should probably also move the check on whether any points use that kernel to before you bother doing the convolution... so I'd modify that to:

```
lochist = histogram(loc, min=0, reverse_indices=ri)

for j=0,n-1 do if lochist[j] gt 0 then begin
  kernel = kernel_store[*,*,j]
  response_temp = convol(img, kernel, /edge_zero, /nan)
  index = ri[ri[j]:ri[j+1]-1]
  response[index] = response_temp[index]
endif
```

-Jeremy.

---

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

---

## Subject: Re: faster convol on local subsets?
## Posted by Andre on Mon, 05 Dec 2011 23:17:48 GMT

View Forum Message <> Reply to Message

On Dec 5, 11:54 am, Yngvar Larsen <larsen.yng...@gmail.com> wrote:
> 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

Thanks for all the suggestions so far.
I tried it with the changes that Jeremy suggested but for some reason
it runs even a little bit slower than the original version.
On a 2300x2900 array the original loop runs for 322.46600s while with
REVERSEINDICES it needs 394.51800s (even when precomputing the kernel
outside the loop). My guess is that it takes more time because calling
the routine in each loop is expensive (http://ross.iasfbo.inaf.it/IDL/
Robishaw/idlfast.html).

I did not yet find time to check the implementation that Yngvar
suggested but tried http://idlastro.gsfc.nasa.gov/ftp/pro/image/convolve.pro
which also implements convolution in the Fourier domain. Still its
slower than the native IDL convolution. According to a comment in
their code IDL 8.1 has a CONVOL_FFT() which seems worth a further try
after I got the update.

Last I also tried to convolve at each position only with desired
kernel. The code looks more or less like this

```
m=half_kernel_size
nc= number of columns of the input
nr = number of rows of the input

for i=m, nc - m-1 do begin
  for j=m, nr - m-1 do begin
        patch=img[i-m:i+m, j-m:j+m]
        kernel=kernel_store[*,*, (max_loc[i,j])]
        temp = convol(patch, kernel])
        response[i,j] = temp[m, m]
  endfor
endfor
```

As expected the convolution runs much quicker than on the full image
but the large number of loops eats up all that speed gains and in the
end its even 409.56500s for the same array as before.
...to be continued...

---

Subject: Re: faster convol on local subsets?
Posted by Yngvar Larsen on Tue, 06 Dec 2011 14:23:07 GMT
View Forum Message <> Reply to Message

On Dec 6, 12:17 am, Andre <note....@gmail.com> wrote:

> I did not yet find time to check the implementation that Yngvar
> suggested but tried http://idlastro.gsfc.nasa.gov/ftp/pro/image/convolve.pro
> which also implements convolution in the Fourier domain. Still its
> slower than the native IDL convolution.

This is not my experience. I typically got a speedup by a factor of
10-100 for some applications where the kernels are quite large.

> According to a comment in
> their code IDL 8.1 has a CONVOL_FFT() which seems worth a further try
> after I got the update.

I didn't know that. Thanks for the tip!

> Last I also tried to convolve at each position only with desired
> kernel. The code looks more or less like this
>
> m=half_kernel_size
> nc= number of columns of the input
> nr = number of rows of the input
>
> for i=m, nc - m-1 do begin
>  for j=m, nr - m-1 do begin
>  patch=img[i-m:i+m, j-m:j+m]
>  kernel=kernel_store[*,*, (max_loc[i,j])]
>  temp = convol(patch, kernel])
>  response[i,j] = temp[m, m]
>  endfor
> endfor

You are calculating the 2D convolution of PATCH and KERNEL, and then
picking out only one element. You could try to calculate this element
by hand, which should be a linear operation.

What is the typical dimension of KERNEL_STORE?

--
Yngvar