Bilinear Interpolation Upsampling on CUDA

Project Phase-I

Bilgin AKSOY

Informatics Enstitute

12th Nov 2017

Problem

- A deep network for object segmentation.
- This network has 5 two-by-two max-pooling layer, each of which down-samples its input by factor two.

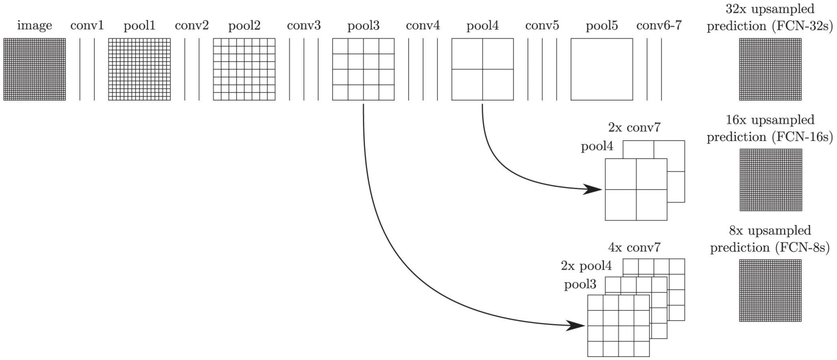
image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)

Figure 1 : Network Architecture-Long et al.[1]

Problem (cont'd)

- After pool3, the activation size have been reduced by factor $8 = 2^3$.
- After pool4, the activation size have been reduced by factor $16 = 2^4$.
- After pool5, the activation size have been reduced by factor $32 = 2^5$.

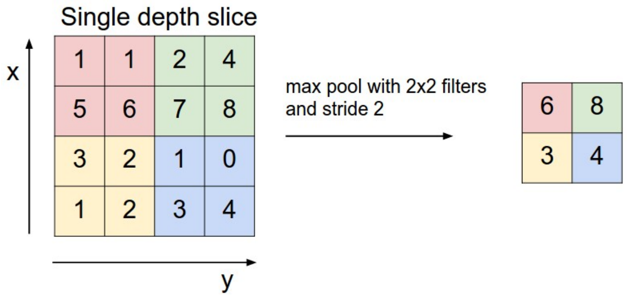

Figure 2 : Max-Pooling - Image from `http://cs231n.github.io/convolutional-networks`

Problem (cont'd)

- At the end of the final layer, sum three activations and calculate the error(loss) and back-propagate the error to be able to minimize the error.

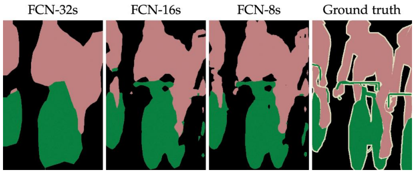

Figure 3 : Activations of Pooling Layers-Long et al.[1]

Problem (cont'd)

- I will implement a bilinear interpolation upsampling kernel and my kernel will not only upsample the activations but also produce the final result by summing the activations.
- This kernel should also work on Python environment (by using Cython / PyCUDA).

Literature Survey

- **Bilinear Interpolation:** Bilinear interpolation is used to know values at random position from the weighted average of the four closest pixels to the specified input coordinates, and assigns that value to the output coordinates.[2] (1)

Literature Survey (cont'd)

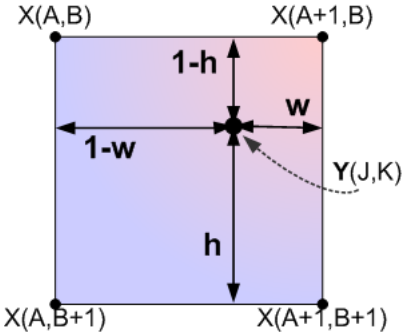

Figure 4 : Image from `https://www.giassa.net/?page_id=240`[3]

$$Y(J,K) = (1-W)(1-H)X(A,B) + (W)(1-H)X(A+1,B) + (1-W)(H)X(A,B+1) + (W)(H)X(A+1,B+1) \quad (1)$$

Literature Survey (cont'd)

- Qingshuang et al.[4] proposed "GPGPU-based parallel algorithm can take full advantage of the GPU's parallel computing capabilities, and can achieve about 40 times speedup."

Table 1 : The Time-Consuming Of Serial And Parallel Bilinear Spatial Interpolation Of Different Grid Size [4]

Grid Size	CPU (sec)	GPU (sec)
40x40	1.70	0.11
100x100	10.34	0.37
200x200	41.43	1.21
300x300	93.13	2.36
500x500	240.02	5.70
800x800	630.92	14.40
1200x1200	1432.33	32.41

- Akgün and Gevrekci have been proposed "a massively multi-threaded implementation of super-resolution image formation on the NVIDIA CUDA architecture." [5] "Super-resolution reconstruction begins with an initial estimate image which is typically chosen as a bilinearly up-scaled version of the original low resolution frame."

Table 2 : Bilinear filtering code analysis [5]

Thread numbers (x,y)	(32,32)
Shared memory bank conflict	N/A
Global memory BW efficiency	100
Register usage	17
Occupancy	0,877
Total execution time per frame	161 microsec

Literature Survey (cont'd)(Real Life Examples)

OpenCV CPU

```
void  cv::resize (
InputArray src, OutputArray dst, Size dsize,
   ↪ double fx=0, double fy=0, int interpolation
   ↪ =INTER_LINEAR
)
```

OpenCV GPU

```
void cv::cuda::resize (
InputArray src, OutputArray dst, Size dsize,
   ↪ double fx = 0, double fy = 0, int
   ↪ interpolation = INTER_LINEAR, Stream &
   ↪ stream=Stream::Null()
)
```

Literature Survey (cont'd)(Real Life Examples)

Intel Integrated Performance Primitives(CPU Optimizations)

```
IppStatus ippiResizeLinear_<mod>(const Ipp<
  ↪ datatype>* pSrc, Ipp32s srcStep, Ipp<
  ↪ datatype>* pDst, Ipp32s dstStep, IppiPoint
  ↪ dstOffset, IppiSize dstSize, IppiBorderType
  ↪  border, const Ipp<datatype>* pBorderValue,
  ↪  const IppiResizeSpec_32f* pSpec, Ipp8u*
  ↪ pBuffer)
```

- Just for Intel CPU. Not free.(249$for Academic Licence)
- OpenCV 3.x has native support for IPP-ICV library which is a special subset of Intel IPP functions for image processing and computer vision.

Literature Survey (cont'd)(Real Life Examples)

Linux Command Line Tool - Imagemagick (CPU Only, OpenCL For Some Algorithms-No information avout Bilinear Upsampling)

```
$ convert input.jpg -scale 200% -interpolate
    ↪ bilinear output.jpg
```

References

[1] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015.

[2] S. Fadnavis, “Image interpolation techniques in digital image processing: An overview,” *International Journal of Engineering Research and Applications*, vol. 4, no. 10, pp. 70–73, 2014.

[3] “Bilinear interpolation.” `https://www.giassa.net/?page_id=240`.

[4] W. Qingshuang, W. Qiang, and C. Xiangfu, “Parallel bilinear spatial interpolation algorithm based on GPGPU.,” *Journal of Theoretical & Applied Information Technology*, vol. 48, no. 3, 2013.

[5] T. Akgün and M. Gevrekci, “Accelerating super-resolution reconstruction using gpu by cuda,” in *Advances in Computational Science, Engineering and Information Technology*, pp. 47–58, Springer, 2013.