# Image Kernel Convolution

Nicholas J. Tustison and James C. Gee

January 19, 2009

Penn Image Computing and Science Laboratory
University of Pennsylvania

**Abstract**

Convolution is a basic operation in image analysis and the Insight Toolkit provides a well-framed mechanism for such operations (see Chapter 11 of the ITK Software Guide [1]). The classes associated with this contribution are meant to simplify convolution operations for the user who wishes to convolve a given image with an arbitrary image kernel. This entails the use of the neighborhood iterator coupled with a image kernel neighborhood operator which we provide. A couple of examples are provided to illustrate its use.

convolution

## 1 Introduction

Given the foundational nature of convolution operations for any type of image analysis research, we forego an introduction to the concept of convolution and refer the interested reader to the extensive literature on the subject or invite the interested reader to seek guidance at the informational pantheon of Google and Wikipedia.

## 2 Usage

### 2.1 Averaging Example

The first example we give is that of a simple averaging operation involving a $5x5$ kernel and the brain slice image given in Figure 1. We first specify the image type and the kernel image type.

```
9  int itkConvolutionImageFilterTest( unsigned int argc, char *argv[] )
10 {
11   typedef float                                PixelType;
12   typedef itk::Image<PixelType, ImageDimension>  ImageType;
```

We then allocate and fill the kernel image with the appropriate values.

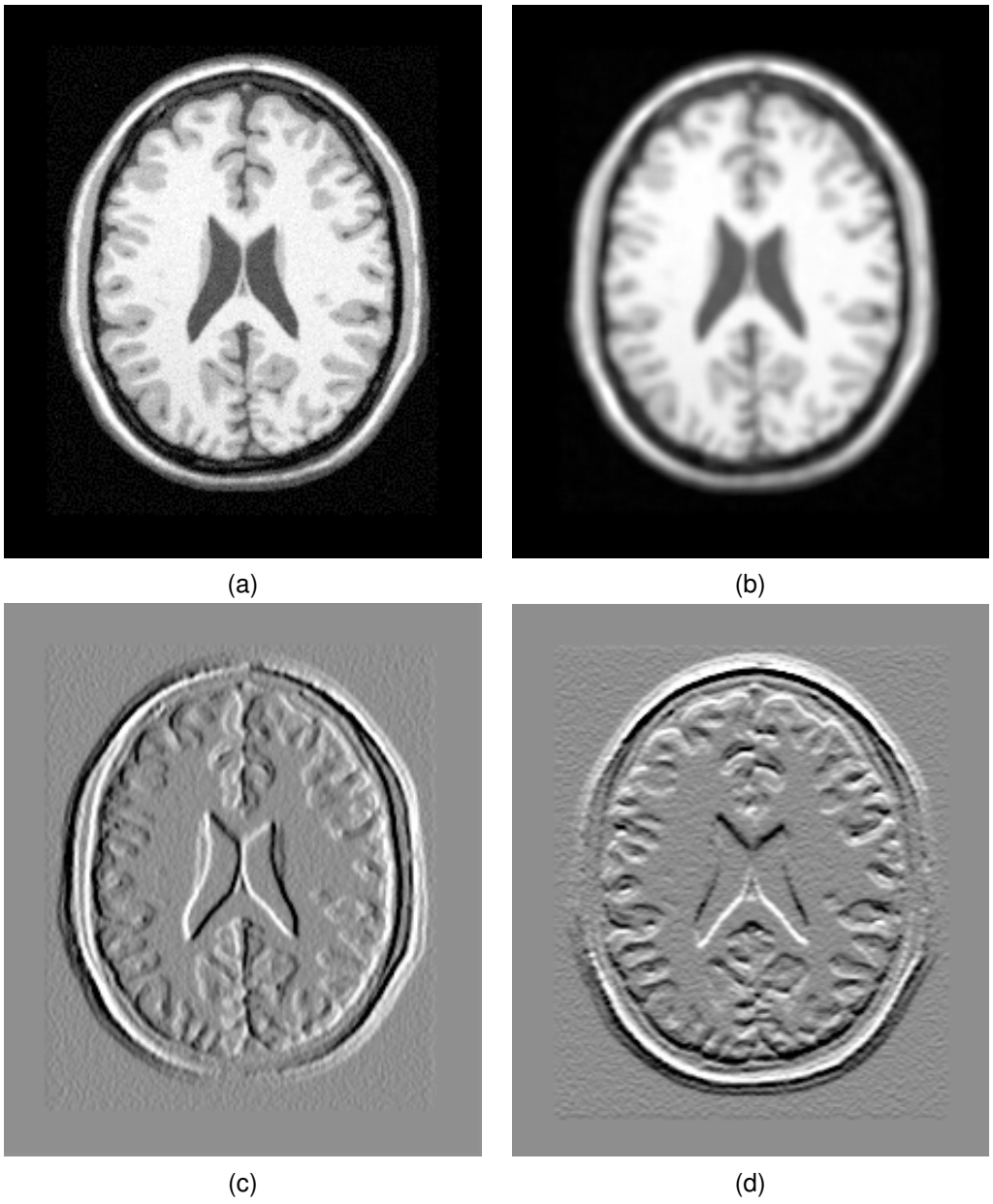


Figure 1: Sample results: (a) original image, (b) average image ($5 \times 5$ filter), (c) sobel filter in the $x$-direction, and (d) sobel filter in the $y$-direction.

```
19  typename ReaderType::Pointer reader2 = ReaderType::New();
20  reader2->SetFileName( argv[3] );
21  reader2->Update();
22
23  typedef itk::ConvolutionImageFilter<ImageType> ConvolutionFilterType;
24  typename ConvolutionFilterType::Pointer convoluter
```

The `ConvolutionImageFilter` is derived from the `ImageToImageFilter` class so we set the first image as the input. We also specify the `ImageKernel`.

```
26    convoluter->SetInput( reader1->GetOutput() );
27    convoluter->SetImageKernelInput( reader2->GetOutput() );
28
29    if( argc > 5 && static_cast<bool>( atoi( argv[5] ) ) )
30      {
31      convoluter->NormalizeOn();
```

## 2.2 Sobel Filter Example

Sobel filtering is basic to evaluating discrete approximations to the image derivative. Sobel filters are given as

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

or its rotational variants. Instead of specifying the filter as in Chapter 11 of [1], we just fill in the `KernelImage` with the appropriate values ($x$-direction).

```
55      {
56      std::cout << "Usage: " << argv[0] << "imageDimension"
57        << " inputImage kernelImage outputImage [normalizeImage]" << std::endl;
58      return EXIT_FAILURE;
59      }
60
61    switch( atoi( argv[1] ) )
62      {
63      case 2:
64        return itkConvolutionImageFilterTest<2>(
65          argc, argv );
66        break;
67      case 3:
68        return itkConvolutionImageFilterTest<3>(
69         argc, argv );
70        break;
71      default:
72        std::cerr << "Unsupported dimension" << std::endl;
73        return EXIT_FAILURE;
74      }
75  }
```

The output images are given in Figure 1.

# References

[1] Luis Ibanez, Will Schroeder, Lydia Ng, and Josh Cates. *The ITK Software Guide*. Insight Software Consortium, 2 edition, November 2005. (document), 2.2