Introduction to Image Processing with SciPy and NumPy

Anil C R
cranil@ee.iisc.ernet.in

Department of Electrical Engineering
Indian Institute of Science

September 19, 2010

Outline

Outline

What are Images?

Continious domain, Continious range

$$f : \mathbb{R}^2 \rightarrow \mathbb{R}(\mathbb{R}^3)$$

Discrete domain, Continious range

$$f : \mathbb{Z}^2 \rightarrow \mathbb{R}(\mathbb{R}^3)$$

Discrete domain, Discrete range

$$f : \mathbb{Z}^2 \rightarrow \mathbb{Z}(\mathbb{R}^3)$$

Finite domain, Continious range

$$f : \mathbb{Z}_m \times \mathbb{Z}_n \rightarrow \mathbb{R}(\mathbb{R}^3)$$

What are Images?

Continious domain, Continious range

$$f : \mathbb{R}^2 \rightarrow \mathbb{R}(\mathbb{R}^3)$$

Discrete domain, Continious range

$$f : \mathbb{Z}^2 \rightarrow \mathbb{R}(\mathbb{R}^3)$$

Discrete domain, Discrete range

$$f : \mathbb{Z}^2 \rightarrow \mathbb{Z}(\mathbb{R}^3)$$

Finite domain, Continious range

$$f : \mathbb{Z}_m \times \mathbb{Z}_n \rightarrow \mathbb{R}(\mathbb{R}^3)$$

Using Matrices to Represent Images

- f as an element of $\mathbb{R}^{m \times n}(\mathbb{R}^{m \times n \times k})$
- $\Rightarrow$ Linear Algebra
- $\Rightarrow$ LAPACK, BLAS, etc
- $\Rightarrow$ FORTRAN, C, etc
- $\Rightarrow$ Super Hard
- $\Rightarrow$ MATLAB
- $\Rightarrow$ Super Expensive
- $\Rightarrow$ SciPy + NumPy, GNU Octave, Scilab, etc
- PyCon 2010
- $\Rightarrow$ SciPy + NumPy

Outline

NumPy

- Numerical Processing
- Started off as *numecric* written in 1995 by Jim Huguni et al.
- Numeric was slow for large arrays and was rewritten for large arrays as *Numarray*
- Travis Oliphant, in 2005 merged them both into *NumPy*

SciPy

- Libraries for scientific computing
- Linear Algebra
- Statistics
- Signal and Image processing
- Optimization
- ODE Solvers

Outline

1. Introduction
   - Image Processing
   - What are SciPy and NumPy?
2. Some Theory
   - Filters
   - The Fourier Transform
3. Doing the Stuff in Python
4. Demo(s)

Filters

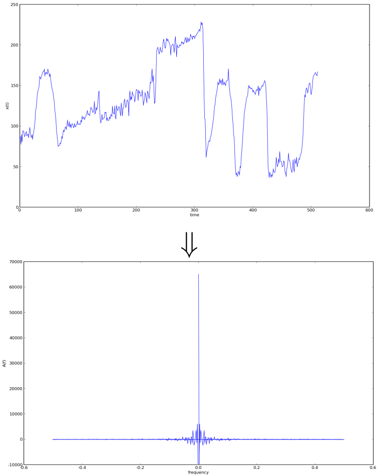
- Keep what you want and throw away the rest
- Studying filters is the most important part in Image Processing
- Classified into *linear* and *non-linear* filters

Linear Filters

- Given images $f_1, f_2, \ldots, f_n$ a filter H is called linear if $H(\alpha_1 f_1 + \alpha_2 f_2 + \cdots + \alpha_n f_n) =$
$$\alpha_1 H(f_1) + \alpha_2 H(f_2) + \cdots + \alpha_n H(f_n)$$
- Linearity can be useful in fast computation

Outline

Time and Frequency Domains



Fourier Transform

Continious FT

$$F(\omega_x x, \omega_y y) = \int\int_{\mathbb{R}^2} f(x, y) exp(-i\omega_x x - i\omega_y y) dxdy$$

Discrete FT

$$F(\omega_x x, \omega_y y) = \sum_0^M \sum_0^N f(x, y) exp(-i\{\frac{2\pi\omega_x}{M} x - \frac{2\pi\omega_y}{N} y\})$$

Notation: F is the FT of f, also $F = \mathcal{F}\{f\}$

Convolution

Continious

$$(f \circledast g)(x, y) = \iint_{\mathbb{R}^2} f(x', y')g(x - x', y - y')dxdy$$

Discrete

$$(f \circledast g)(x, y) = \sum\sum_{\mathbb{Z}^2} f(x, y)g(x - x', y - y')$$

Theorem

(a) $\mathcal{F}\{f \circledast g\} = FG$

(b) $\mathcal{F}\{fg\} = F \circledast G$

Any linear filter can be written as a convolution.

Fast Fourier Transform (FFT)

- Computing the Discrete Fourier Transform takes $O(n^2m^2)$ for an $m \times n$ image
- FFT Computes the same in $O(n \log nm \log m)$

Interactive Python

- Install NumPy
- Install SciPy
- Install Matplotlib
- Install IPython

Running IPython

```
$ ipython -pylab
```

Fast Fourier Transform (FFT)

FFT in NumPy

```
In[1]: from scipy import lena
In[2]: f = lena()
In[3]: from numpy.fft import fft2 #
unnecessary if you invoke ipython with -pylab
In[4]: F = fft2(f)
In[5]: imshow(real(F))
```

Fast Fourier Transform (FFT)

FFT in NumPy

```
In[1]: from scipy import lena
In[2]: f = lena()
In[3]: from numpy.fft import fft2 #
unnecessary if you invoke ipython with -pylab
In[4]: F = fft2(f)
In[5]: imshow(real(F))
```

Fast Fourier Transform (FFT)

FFT in NumPy

```
In[1]: from scipy import lena
In[2]: f = lena()
In[3]: from numpy.fft import fft2 #
unnecessary if you invoke ipython with -pylab
In[4]: F = fft2(f)
In[5]: imshow(real(F))
```

Fast Fourier Transform (FFT)

FFT in NumPy

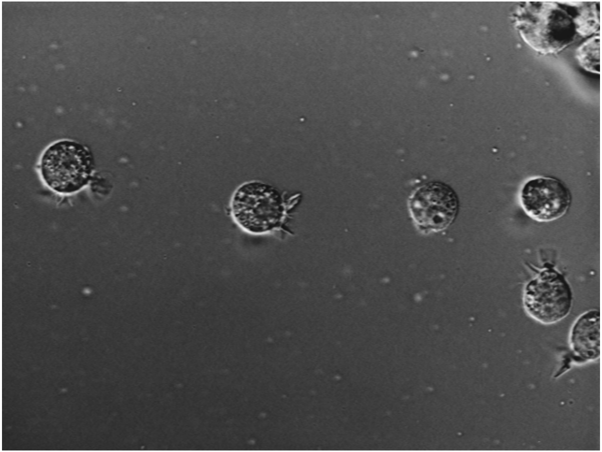
```
In[1]: from scipy import lena
In[2]: f = lena()
In[3]: from numpy.fft import fft2 #
unnecessary if you invoke ipython with -pylab
In[4]: F = fft2(f)
In[5]: imshow(real(F))
```

Fast Fourier Transform (FFT)

FFT in NumPy

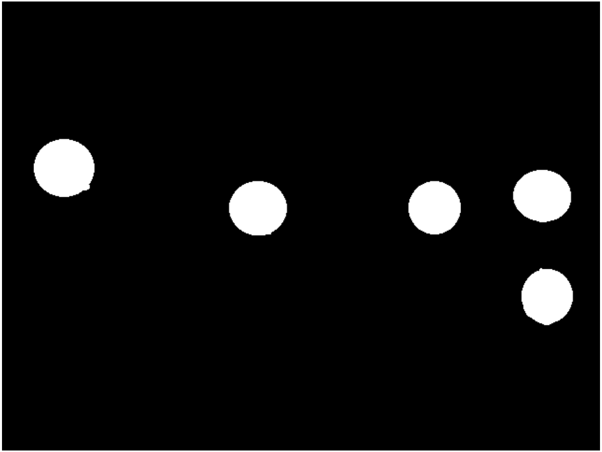
```
In[1]: from scipy import lena
In[2]: f = lena()
In[3]: from numpy.fft import fft2 #
unnecessary if you invoke ipython with -pylab
In[4]: F = fft2(f)
In[5]: imshow(real(F))
```

Demo: Cells



Input Image

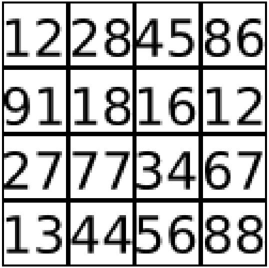
Demo: Cells



Output

Demo: Cells
Procedure

Consider the image:



Demo: Cells

Procedure Cont'd

Find the variance of the neighborhood of each pixel, store them as a 2D array

12	28	45	86
91	18	16	12
27	77	34	67
13	44	56	88

Demo: Cells

Procedure Cont'd

Find the variance of the neighborhood of each pixel, store them as a 2D array

12	28	45	86
91	18	16	12
27	77	34	67
13	44	56	88

12	28	45	86
91	18	16	12
27	77	34	67
13	44	56	88

Demo: Cells

Procedure Cont'd

Find the variance of the neighborhood of each pixel, store them as a 2D array

12	28	45	86
91	18	16	12
27	77	34	67
13	44	56	88

12	28	45	86
91	18	16	12
27	77	34	67
13	44	56	88

12	28	45	86
91	18	16	12
27	77	34	67
13	44	56	88

Demo: Cells

Procedure Cont'd

Variance map (V)

$V > \mathbb{E}\{V\}$

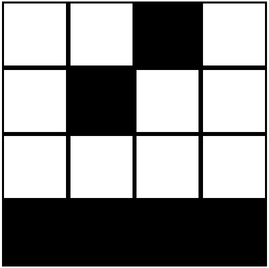
996	744	654	875
920	686	689	700
870	670	695	750
571	426	344	380

Demo: Cells

Procedure Cont'd

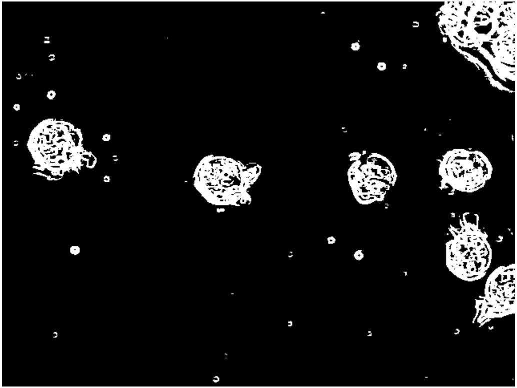
Variance map (V)

$V > \mathbb{E}\{V\}$



Demo: Cells
Procedure Cont'd

Algorithm on the cell image:



Q and A