Computer Vision: COMP6206

Introduction: Technology

John Carter (jnc)
Mark Nixon (msn)
Introduction to Technique

• Images
• Basic tools
  – Matlab
  – Julia
  – C++
  – Python
• Programming
• Internet Resources
Images and Formats

- Don’t use compressed images i.e. JPEG
- PNG are ok
- matlab, python, Julia can read all these
- C/C++ use stb_image.h at
  - https://github.com/nothings/stb
External Tools: Picture Viewers

- Lots of Picture Editors
- Depends on computing platform
- Best to have a simple Picture Viewer
- **IrfanViewer for Windows.**
  - www.irfanview.com
Basic Tools - MATLAB

- Supports Mark’s book
- Relatively easy to use
- More algorithmic less associated with maths.
- Image Toolbox – Mark and I have had bad experiences.
- ISS provision in School and University
- Code from WWW often broken.
- Matlab has a big problem.
Octave

I = loadimage ("default.img");
S = conv2 (I, ones (5, 5) / 25, "same");
[Dx, Dy] = gradient (S);

• Octave is almost 100% compatible with MATLAB
• Its free, no licences
• Images can be displayed.
  • Grey, RGB, multi-band.

• http://www.gnu.org/software/octave/
• https://octave.sourceforge.io/packages.php
Octave 4.0
Java

- Neither machine code or interpreter.
- Pixel loops can be slow, depends on JIT
- Portable, web friendly
- Native code extensions (not web friendly)
- Libraries for imaging and graphics
- Low level code
- OpenCV support
Java Script

• Ndarry package
• Leverage WebGL for GPGPU

• Java Script can be almost as fast as C++
  – See Emscripten
  – www examples
Basic Tools C/C++

- OpenCV, CImg
- OpenCV is probably better.
- Lots of other libraries etc
  - All different (and incompatible)
- Programmer must do all the work
- Highly efficient code
- Low productivity.
Basic Tools - Python

- Not supported by book
- My standard tool - https://www.continuum.io/
- Anaconda
  - Freely available, open source, cross platform
  - Extension package for image handling
  - Vector array processing
  - Easy to extend in C/C++
- Book – ‘Programming Computer Vision with Python’
- Link to copy on module notes page.
Python Tools

- numpy
- OpenCV
- Python Image Library
- SciPy
- ndimage
- imageio
- iPython, Jupyter
- Notebook Interface
- matplotlib
Image Processing with Python

```python
import imageio, numpy, scipy

a = imageio.imread('lena.jpg')

s = scipy.ndimage.sobel(a)

imageio.imwrite('edge.png', s)
```
Anaconda Distribution

• **Continuum Analytics**, http://continuum.io/
• Anaconda distribution of **Python**.
  – **Windows 7/8** (32/64 bit), **OSX** (64 bit), **Linux** (64 bit)
• Everything you need.
• Accelerate Package – **multi-core**, **gpu** and **jit**.
• **Academic licence** for accelerated version
Julia

• Newish language
  – Optimised for numerical calculation
  – http://julialang.org

• Interpreter + JIT

• Beats Python

• Beats C++, little overhead

• Supported in Jupyter (iPython notebook)
using NPZ, PyPlot
x = npzread("table_3f0595.npy")
p = imread("frame.0001.maskpp2.png");
q = zeros(Int16, 240,286,600);
function f(x::Array{Int16, 4}, p::Array{Float32, 2}, q::Array{Int16, 3})
    for i::Int = 87:201
        for j::Int = 1:196
            for k::Int = 81:521
                u::Int16 = x[i, j, k, 1]
                v::Int16 = x[i, j, k, 2]
                h::Float64 = p[u, v]
                if h > 0.0
                    q[i, j, k] = 1.
                end
            end
        end
    end
end
F(x, p, q) -> 250 ms
I don’t like lots of ‘end’
Julia in the big world

- Can call python modules directly
- Native arrays map to numpy.
- Matlab interface
- C++ interface
- OpenCV interface.
Julia Performance
Tools: Summary

• High level
  – New language
  – Slow for pixel operations
  – High productivity
  – Good comprehension

• Low level
  – Familiarity
  – Low productivity
  – High performance
  – Difficult to understand

• Best solution is to mix and match.
• High level code calling low level procedures for performance.
Vector Arithmetic

• Python, MATLAB and Octave are interpreted languages.
  
• This means that loops over pixels are very slow.
  – Use Julia, C++
  – Vector operations
  – Convolution
    operator/function

for var = pixels
  operate on var
endfor
Vector Code in Python
X Sobel Gradient

```python
import numpy, imageio

B = imageio.imread('lena.png')  # load picture
C = B - 128                    # Subtracts 128 from every pixel
C[0:10, 30:40] = 0             # sets a block in the middle to 0
D = array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])

E = conv(A, D)                 # direct convolution
E = invfft( fft(A) * fft(D) )  # in Fourier Domain
```
Vector Code in Python
X Sobel Gradient

```python
import numpy, imageio

B = imageio.imread('lena.png')  # load picture
C = B - 128  # Subtracts 128 from every pixel
C[0:10, 30:40] = 0  # sets a block in the middle to 0

D = array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])

E = conv(A, D)  # direct convolution
E = invfft(fft(A) * fft(D))  # in Fourier Domain
```
Vector Code in Python
X Sobel Gradient

```
import numpy, imageio

B = imageio.imread('lena.png')  # load picture
C = B - 128  # Subtracts 128 from every pixel
C[0:10, 30:40] = 0  # sets a block in the middle to 0
D = array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])

E = conv(A, D)  # direct convolution
E = invfft(fft(A) * fft(D))  # in Fourier Domain
E = 2*A[1:-1,:]- (A[:-2,:]+A[2:,:])  # Vector Sum
```
import numpy, imageio

B = imageio.imread('lena.png')  # load picture

C = B - 128  # Subtracts 128 from every pixel

C[0:10, 30:40] = 0  # sets a block in the middle to 0

D = array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])

E = conv(A, D)  # direct convolution

E = invfft(fft(A) * fft(D))  # in Fourier Domain

E = 2*A[1:-1,:]-(A[:-2,:]+A[2:,:])  # Vector Sum
numpy

- Adds **multi-dimensional** array structure to python
- Many basic **operations**
- Convolution, **fft**, **arithmetic**, **matrix** (inverse, svd, eigen ..)
- *Easy* to **extend** with simple C interface.
- Easy to use with OpenCV
- OpenCV has python bindings, numpy as image format.
SciPy

- Advanced maths functions based on numpy.
- statistics, optimisation, integration, linear algebra, signal processing
- Image processing `scipy.ndimage`
  - correlation/convolution, fourier, morphology, interpolation, measurement
Scikits

• Python, numpy and scipy packages

• Scikit-image – imaging functions
• Scikit-learn - machine learning
Jupyter, matplotlib

• **Extension** Python Shell.
• **Notebook interface** like Matematica.
  – Windows, Mac and Linux.
• **Spyder** gui tool a bit like Matlab.
• Matlab equivalent **graphics** (with **latex**)
  – numpy, pil, scipy, OpenCV gives an open source/cross platform environment

  - \( a = \text{numpy.arange}(0., 30., 0.1) \)
  - \( \text{matplotlib.plot}(a, \sin(a) \times \exp(-4\times a)) \)
  - \( \text{matplotlib.title}('\$\sin(x)\times e^{-4x}\$') \)
  - \( \text{matplotlib.xlabel}('\text{Time}') \)

• Works with Julia.
$\sin(x) e^{-4x}$
iPython displays images

```python
l = imread('lena.png')
imshow(l)
```
In [1]: %pylab

   Using matplotlib backend: Qt4Agg
   Populating the interactive namespace from numpy and matplotlib

In [2]: l = imread('lena_std.png')

In [3]: l.shape

Out[3]: (512, 512, 3)

In [4]: %matplotlib inline

In [5]: imshow(l)

Out[5]: <matplotlib.image.AxesImage at 0x75c39f0>
iPython (ii)
C++, python and OpenCV
http://opencv.org

- C/C++/Python/Java Library
- Open Source
- Supported by Intel
- Peer review of algorithms
- Easy to use.
- Don’t bother with books talking about V1
- The API changed going from V1 to V2, now V3.1x
- Windows, Mac, Linux, iOS and Android.
C++ API is easy.

```c++
int main( int argc, char** argv )
{
    Mat image, gray, bl,;
    image = imread("Lena.jpg", 1);
    cvtColor(image, gray, CV_BGR2GRAY); // rgh->gray
    namedWindow("Blured Image", 1);// Create a window
    blur(gray, bl, Size(3,3)); // blur: grey->bl
    imshow("Blured Image", bl);
    waitKey(0);// Wait for a key stroke
}
```
C++ API is easy.

```c
int main( int argc, char** argv )
{
    Mat image, gray, bl,;
    image = imread("Lena.jpg", 1);
    cvtColor(image, gray, CV_BGR2GRAY); // rgh->gray
    namedWindow("Blured Image", 1); // Create a window
    blur(gray, bl, Size(3,3)); // blur: grey->bl
    imshow("Blured Image", bl);
    waitKey(0); // Wait for a key stroke
}
```
C++ API is easy.

```cpp
int main( int argc, char** argv )
{
    Mat image, gray, bl,;
    image = imread("Lena.jpg", 1);
    cvtColor(image, gray, CV_BGR2GRAY); // rgh->gray
    namedWindow("Blured Image", 1);// Create a window
    blur(gray, bl, Size(3,3)); // blur: grey->bl
    imshow("Blured Image", bl);
    waitKey(0); // Wait for a key stroke
}
```
C++ API is easy.

```c++
int main( int argc, char** argv )
{
    Mat image, gray, bl,;
    image = imread("Lena.jpg", 1);
    cvtColor(image, gray, CV_BGR2GRAY); // rgh->gray
    namedWindow("Blured Image", 1); // Create a window
    blur(gray, bl, Size(3,3)); // blur: grey->bl
    imshow("Blured Image", bl);
    waitKey(0); // Wait for a key stroke
}
```
C++ API is easy.

```cpp
int main( int argc, char** argv )
{
    Mat image, gray, bl,;
    image = imread("Lena.jpg", 1);
    cvtColor(image, gray, CV_BGR2GRAY);  // rgh->gray
    namedWindow("Blured Image", 1);     // Create a window
    blur(gray, bl, Size(3,3));          // blur: grey->bl
    imshow("Blured Image", bl);
    waitKey(0);                         // Wait for a key stroke
}
```
Image I/O Libraries

• Don’t write your own

• libpng, libjpeg (avoid)
• zlib (used for compressing stuff) (avoid)
• FreeImage, SDL Image, DevIL (avoid)

• Try stb_image.c a standalone solution for simple jpg and png.
  – https://github.com/nothings/stb

• Use OpenCV.
from Tkinter import *
import sys, Image, ImageTk

class UI(Label):
    def __init__(self, master, im):
        self.image = ImageTk.PhotoImage(im)
        Label.__init__(self, master, image=self.image, bd=0)

root = Tk()
root.title(sys.argv[1])
im = Image.open(sys.argv[1])
UI(root, im).pack()
root.mainloop()
Usefull tools

• geo-gebra
  – Interactive geometry toll
  – https://www.geogebra.org/

• wxMaxima
  – Symbolic maths.
  – http://andrejv.github.io/wxmaxima/
Geo Gebra
wxMaxima

```
--> a: sin(x)

(%i1) wxplot2d([a: sin(x)], [x, -5, 5])$
```

![Plot of sin(x)](image)