Image Processing with Python

Desert Py Meetup
26 February 2014

Sarah E. Braden
Overview

- **Pillow**
  - Pillow is a fork of PIL, the Python Imaging Library
  - http://python-imaging.github.io/

- **OpenCV**
  - [http://opencv.org/](http://opencv.org/)

- **Files for this presentation**
  - [https://github.com/desertpy/presentations](https://github.com/desertpy/presentations)
Why Pillow?

The **Python Imaging Library**, (PIL) is the library for image manipulation, however...
Why Pillow?

... PIL’s last release was in 2009
Why Pillow?

- easier to install
- supports Python 3
- active development
- actually works*

* Non-Pillow PIL really is frustrating

http://python-imaging.github.io/
What can Pillow do for me?

- Automatically generate thumbnails
- Apply image filters (auto-enhance)
- Apply watermarks (alpha layers)
- Extract images from animated gifs
- Extract image metadata
- Draw text for annotations (and shapes)
- Basically script things that you might do in Photoshop or GIMP for large numbers of images, in Python

Modules:

- ImageOps
- ImageMath
- ImageFilter
- ImageEnhance
- ImageStat
Pillow Setup

Pillow’s prerequisites:
https://pypi.python.org/pypi/Pillow/2.1.0#platform-specific-instructions

Warning!
Since some (most?) of Pillow's features require external libraries, prerequisites can be a little tricky

After installing prereqs:
$ pip install Pillow
Documentation

Pillow documentation:  http://pillow.readthedocs.org/en/latest/about.html


from PIL import Image

im = Image.open(infile)
im.show()

print(infile, im.format, "%dx%d" % im.size, im.mode)

Important note: Opening an image file is a fast operation, independent of file size and compression. Pillow will read the file header and doesn’t decode or load raster data unless it has to.
Basic Methods

geometric transforms:

\[
\begin{align*}
\text{out} &= \text{im}.\text{resize}((128, 128)) \\
\text{out} &= \text{im}.\text{rotate}(45) \ # \text{ degrees counter-clockwise} \\
\text{out} &= \text{im}.\text{transpose}(\text{Image.FLIP_LEFT_RIGHT}) \\
\text{out} &= \text{im}.\text{transpose}(\text{Image.FLIP_TOP_BOTTOM})
\end{align*}
\]

crop:

\[
\begin{align*}
\text{box} &= (100, 100, 400, 400) \ # (\text{left, upper, right, lower}) \\
\text{region} &= \text{im}.\text{crop}(\text{box})
\end{align*}
\]
Practical Things: Make thumbnails

```python
from PIL import Image

def autothumb(infile, outfile, size, format):
    if infile != outfile:
        try:
            im = Image.open(infile)
            im.thumbnail(size)
            im.save(outfile, format)
        except IOError:
            print("cannot create thumbnail for", infile)
```
Practical Things: Apply filters

The **ImageFilter** module contains a number of pre-defined enhancement filters that can be used with the `filter()` method:

- BLUR
- CONTOUR
- DETAIL
- EDGE_ENHANCE
- EDGE_ENHANCE_MORE
- EMBOSS
- FIND_EDGES
- SMOOTH
- SMOOTH_MORE
- SHARPEN

```python
from PIL import Image, ImageFilter

out1 = im.filter(ImageFilter.BLUR)
out2 = im.filter(ImageFilter.GaussianBlur(radius=20))
```
Practical Things: Apply filters

UnsharpMask($radius=2$, $percent=150$, $threshold=3$)

- radius – size of the area
- percent – % contrast change allowed in area
- threshold – minimum brightness change needed before filter takes effect

Unsharp masks basically apply a Gaussian blur to a copy of the original image and compare it to the original. If the difference is greater than a threshold setting, the images are basically subtracted.

Kernel($size$, $kernel$, $scale=None$, $offset=0$)

- size – Kernel size, given as (width, height)
- kernel – a sequence containing kernel weights
- scale – the result for each pixel is divided by this value. Default = sum of the kernel weights
- offset – this value is added to the result, after it has been divided by the scale factor
from PIL import Image

baseim = Image.open(imgfile)
logoim = Image.open(watermark)  # transparent image
baseim.paste(logoim, (baseim.size[0]-logoim.size[0],
baseim.size[1]-logoim.size[1]), logoim)
baseim.save('new.png', "PNG")

# Important thing is the 3rd argument of the paste function. Specifies your PNG as alpha layer so that you avoid a black background.
What is OpenCV?

- Open source computer vision library in C++
- Includes a machine learning library to support computer vision applications
- OpenCV-Python is the Python API of OpenCV
- Large user base = good documentation and excellent online tutorials and help
- Huge library, super powerful
OpenCV Fun Facts

- In 2005, OpenCV was used on Stanley, the vehicle who won 2005 DARPA Grand Challenge [1]
- You can solve sudoku puzzles with OpenCV [2]
- OpenCV is under a BSD license [3]
- OpenCV is being used to program new robots like the PR2 [4]
- Shoot squirrels [5]

[3] opencv.org
Install OpenCV for Python

$ sudo apt-get install python-opencv

Documentation: http://docs.opencv.org/modules/core/doc/doc/intro.html
Weird things

- Color image loaded by OpenCV is in BGR mode
- But Matplotlib displays in RGB mode
- So color images will not be displayed correctly in Matplotlib if image is read with OpenCV
Fourier Transforms on 2D images

Use Numpy or Opencv

Center of the image represents the lower frequencies
More examples of 2D Fourier Transforms

Images from: http://cns-alumni.bu.edu/~slehar/fourier/fourier.html
import cv2
import numpy as np
from matplotlib import pyplot as plt

# zero frequency will be in the corners
# to bring zero frequency to the center, shift the result by N/2
# in both the directions: use the function np.fft.fftshift()

def cv_fft(img):
    """
    OpenCV provides the functions cv2.dft() and cv2.idft()
    It returns the same result as numpy, but with two channels.
    First channel will have the real part of the result and
    second channel will have the imaginary part of the result.
    The input image should be converted to np.float32 first.
    """
    dft = cv2.dft(np.float32(img), flags = cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
    a = dft_shift[:,:,:,0]
    b = dft_shift[:,:,:,1]
    magnitude_spectrum = 20*np.log(cv2.magnitude(a, b))
    return magnitude_spectrum
Mask and Inverse FFT

def create_mask(img):
    rows, cols = img.shape
    crow, ccol = rows/2, cols/2

    # create a mask first, center square is 1, remaining all zeros
    mask = np.zeros((rows, cols, 2), np.uint8)

    return mask

def apply_mask(mask, img):
    # apply mask and inverse DFT
    dft = cv2.dft(np.float32(img), flags = cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
    fshift = dft_shift*mask
    f_ishift = np.fft.ifftshift(fshift)
    img_back = cv2.idft(f_ishift)
    img_back = cv2.magnitude(img_back[:,:,0], img_back[:,:,1])

    return img_back
High Pass Filter Using FFT

Hey look it’s edge-detection!
Questions?

Post questions to the DesertPy G+ page!

Sarah Braden Twitter: @ifmoonwascookie
Alternatives to Pillow?

- **PythonMagickWand** is an object-oriented Python interface to MagickWand based on ctypes. 21 January 2009.
- **PythonMagick** is an object-oriented Python interface to ImageMagick. Last build 22 January 2014.
- **Wand** is a ctypes-based ImageMagick binding library for Python. Last release 17 June 2013.
Pillow Setup

Some (most?) of Pillow's features require external libraries.

- **libjpeg** provides JPEG functionality.
  - Pillow has been tested with libjpeg versions **6b**, **8**, and **9**
- **zlib** provides access to compressed PNGs
- **libtiff** provides group4 tiff functionality
  - Pillow has been tested with libtiff versions **3.x** and **4.0**
- **libfreetype** provides type related services
- **littlecms** provides color management
- **libwebp** provides the Webp format.
  - Pillow has been tested with version **0.1.3**, which does not read transparent webp files.
    - Version **0.3.0** supports transparency.

Also: you may want to: sudo apt-get install imagemagick

Prerequisites are installed with on **Ubuntu 12.04 LTS** with

```bash
$ sudo apt-get install libtiff4-dev libjpeg8-dev zlib1g-dev libfreetype6-dev liblcms1-dev libwebp-dev
```