IMAGE PROCESSING CAPABILITIES OF PYTHON

[1]PRIMOŽ PODRŽAJ, [2]SAMO SIMONČIČ

Faculty of Mechanical Engineering, University of Ljubljana
E-mail: [1]primoz.podrzaj@fs.uni-lj.si, [2]samo.simoncic@fs.uni-lj.si

Abstract - Python is becoming increasingly popular programing language. Some sources even claim it is the fastest growing major programming language. As it is completely free, it is very popular in academic environment. In this paper its capabilities in the area of image processing, image analysis and computer vision in general are discussed. The libraries which facilitate the associated tasks are described. The execution of some of the most common algorithms in the field is also demonstrated.

Index Terms - Image Processing, Computer Vision, Python.

I. INTRODUCTION

Python is a high-level general-purpose programming language created by Guido van Rossum in 1991. It has a design philosophy that puts emphasis on code readability. It supports multiple programming paradigms including object-oriented, imperative, functional and procedural and has a large standard and comprehensive library. The first release was followed by Python 2.0 in 2000 and Python 3.0 in 2008. At the time of writing this paper the latest version is Python 3.7.

Python is a good choice for all the researchers in the scientific community because it is [2]:

- free and open source
- a scripting language, meaning that it is interpreted
- a modern language (object oriented, exception handling, dynamic typing etc.)
- concise, easy to read and quick to learn
- full of freely available libraries, in particular scientific ones (linear algebra,visualization tools, plotting, image analysis, differential equations solving,symbolic computations, statistics etc.)
- useful in a wider setting: scientific computing, scripting, web sites, text parsing,etc.
- widely used in industrial applications

In comparison with other programming languages such as C/C++, Java, and Fortran, Python is a higher-level language. The computation time is therefore typically a little longer, but it is much easier to program in. In the case of C and Fortran, wrappers are also available. PHP and Ruby on the other side are high-level languages as well. Ruby can be compared to Python but lacks scientific libraries. PHP on the other hand is a more web-oriented language.

Python can also be compared to Matlab, which has a really extensive scientific library. It is however not open source and free.Scilab and Octave are open source environments similar to Matlab. Their language features are however inferior to the ones available in Python.

People in general tend tothink that complex problems demand complex processes in order to produce complexsolutions. Python was developed with exactly the opposite philosophy. It has an extremely flat learning curve and developmentprocess for software engineers [3]. It is used for system administration tasks, by NASA both for development and as a scripting language in several of its systems, Industrial Light & Magic uses Python in its production of special effects for large-budget feature films, Yahoo! uses it(among other things) to manage its discussion groups and Google has used it to implement many componentsof its web crawler and search engine [4].

As Python is also a language that is easy to learn and both powerful and convenient from the start [5], we might soon be asking, who is not using it.

As already mentioned Python has an extensive set of libraries which can be imported into a project in order to perform specific tasks. The ones that really should be mentioned in any scientific paper dealing with mathematics are NumPy and SciPy. NumPy is a library which provides support for large, multi-dimensional arrays. As images are in fact large two (greyscale) or three (color) dimensional matrices, this library is essential in all image processing tasks. It should also be emphasized that many other libraries (not limited to image processing) use NumPy array representation. SciPy is a library build on the NumPy array object and contains modules for signal and image processing, linear algebra, fast Fourier transform, etc. The last library mentioned in this introductory section is Matplotlib. As the name suggests this library is a plotting library. Although it is used a lot in all areas of science, Image processing relies heavily on it.

In this paper the libraries and their capabilities in the area of image processing, image analysis and computer vision in general are discussed. The execution of some of the most common algorithms in the field is also demonstrated together with the resulting images.

II. IMAGE PROCESSING LIBRARIES

There are several Python libraries related to Image processing and Computer vision. The ones that will be presented in this paper are:

- **PIL/Pillow**

This library is mainly appropriate for simple image manipulations (rotation, resizing, etc.) and very basic image analysis (histogram for example)

- **SimpleCV**

It's a library intended (as the name suggests) to be a simplified version of OpenCV. It doesn't offer all the possibilities of OpenCV, but it is easier to learn and use.

- **OpenCV**

It is by far the most capable and most commonly used computer vision library. It is written in C/C++, but Python bindings are added during the installation. It also gives emphasis on real time image processing.
Among the ones, which will not be presented it might be worth mentioning Ilastik. It is a simple, user-friendly tool for interactive image classification, segmentation and analysis.

III. PIL/PILLOW

Python Imaging Library (PIL) is a library originaly written by Fredrik Lundh. As PIL's last release dates back to 2009 it is a little outdated [6]. It's successor that also supports Python 3 is called Pillow [7]. Consequently not both of them can be installed at the same time. At the time of wrting the paper, the last version of Pillow is 5.1.0. The most trivial program (showing an image) can be written as follows:

```
from PIL import Image
image = Image.open('/path/to/the/image/Lena.jpg')
image.show()
```

The resulting image is shown in Fig. 1.



Fig. 1: "Lena" image - often used in Image Processing

Pillow is able to extract a lot of information from an image and makes it possible to execute several standard procedures for image manipulation, including [yxc]:

- per-pixel manipulations,
- masking and transparency handling,
- image filtering, such as blurring, contouring, smoothing, or edge finding,
- image enhancing, such as sharpening, adjusting brightness, contrast or color

Some of them will be demonstrated. Image can for example be easily rotated for specific angle (in our case 45^{o}) and then save by the following code:

```
rotated_image = image.rotate(45)
rotated_image.save('rotated_image.jpg')
```

The resulting image is shown in Fig. 2.A color image can also be split into different components (red, green and blue).

```
r, g, b = image.split()
r.show()
```

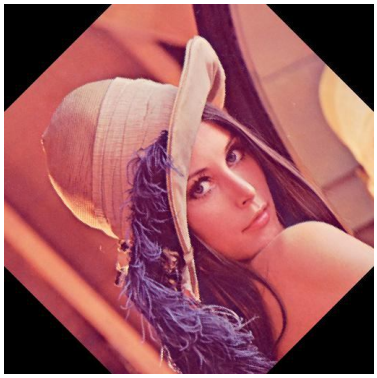


Fig. 2: Rotated version of the image shown in Fig. 1

The red component of the image shown in Fig. 1 can be seen in Fig. 3.

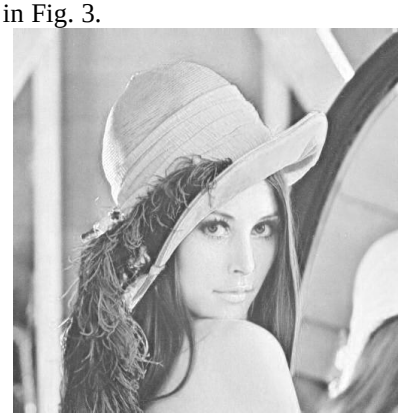


Fig. 3: Red component in RGB representation of the image shown in Fig. 1

Image can also easily be sharpened or blurred. In this case it is however important that we also import the ImageFilter library. The code needed is shown below.

```
from PIL import ImageFilter
sharpened_image = image.filter(ImageFilter.SHARPEN)
blurred_image = image.filter(ImageFilter.BLUR)
```

Both sharpened and blurred version of the image

shown in Fig. 1 are shown in Fig. 4.



Fig. 4: Sharpened (left) and blurred (right) version of the image shown in Fig. 1

Image can for example also be easily cropped with the following command

```
cropped_image = image.crop((100, 100, 400, 400))
```

The resulting image is shown in Fig. 5. It is also very easy to



Fig. 5: Cropped version of the image shown in Fig. 1

add a watermark to an image. This can be done by the following sequence of commands:

```
from PIL import ImageDraw, ImageFont
watermark_image = Image.open('lena-color.jpg')
iDraw = ImageDraw.Draw(watermark_image)
text = "Lena"
font = ImageFont.truetype("arial.ttf", size=40)
iDraw.text((10, 10), text, font=font)
```

The result is shown in Fig. 6.



Fig. 6: Watermarked version of the image from Fig. 1

It was demonstrated that PIL/Pillow is very easy if only basic image processing task are needed. For more detailed analysis and computer vision SimpleCV and OpenCV are more appropriate.

IV. SIMPLE CV

SimpleCV is an interface for Open Source machine vision libraries in Python. As the name suggests it is a simplified version of OpenCV. It is relatively easy to learn and use, as it is concise, has readable interface for cameras, and makes it possible to execute image manipulation, feature extraction, and format conversion. At the time of writing it is however still limited to Python2. As now the majority of Python users have already transferred to Python3, SimpleCV won't be described in detail. Below a simple program can be seen.

```
from SimpleCV import Image
img = Image('lena.jpg')
logo.show()
```

As everything included in SimpleCV can be done in OpenCV, which is still developed, emphasis will be put on OpenCV.

V. OPEN CV

OpenCV is an open source computer vision library availablewritten in C and C++ which runs under Linux, Windows, Mac OS X, iOS,and Android. Interfaces are available for Python, Java, Ruby, Matlab, and other languages.

A very simple program used just to show an image (the same as in Fig. 1) can be written as follows:

```
import numpy as np
import cv2
img = cv2.imread('lena-color.jpg')
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

It is also very easy to form a binary image with a certain predefined threshold. The following commands give a binary image (threshold is set to 127) for the image shown in Fig. 1.



Fig. 7: Binary image (threshold 127) of the image shown in Fig. 1

Edges in an image can be obtained by a simple command:

```
edges = cv2.Canny(gray_image,100,200)
```

The resulting image is shown in Fig. 8.



Fig. 8: Edges obtained by the Canny edge detector for the image shown in Fig. 1 (

The parameters 100 and 200 define the limiting values of the intensity gradient. Pixels below the lower value are non-edge pixels. Pixels above the upper value are edge pixels. Pixels in between are edge pixels if they are connected to the pixels with the intensity gradient above the upper limiting value. Of course they can be set arbitrarily.

OpenCV enables us to do even more complex tasks relatively easy. There are for example routines, which detect face(s) (eyes in an image). The following sequence of commands does just that.

```
face_cascade=cv2.CascadeClassifier('C:\\Users\\...\\haarcascade_frontalface_default.xml')
eye_cascade=cv2.CascadeClassifier('C:\\Users\\...\\haarcascade_eye.xml')
img = cv2.imread('lena-color.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
roi_gray = gray[y:y+h, x:x+w]
roi_color = img[y:y+h, x:x+w]
 eyes = eye_cascade.detectMultiScale(roi_gray)
for (ex,ey,ew,eh) in eyes:
cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The result is shown in Fig. 9.

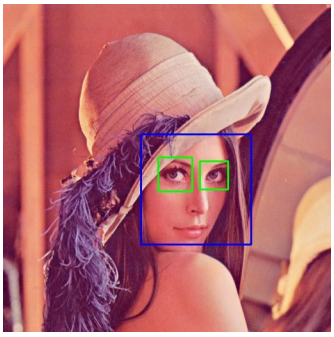


Fig. 9: Image of Lena with face and eyes marked

For the image in Fig. 1 algorithm works perfectly. If however a more complex image is used, the result (especially for eyes) is not so good. See for example Fig. 10.



Fig. 10: Face detected in an image

If needed, it is also very easy to control certain parameters in the image with a slider. The following sequence of commands for example makes it possible to interactively determine the color of the rectangle.

```
def nothing(x):
    pass
# Create a black image, a window
img = np.zeros((300,512,3), np.uint8)
cv2.namedWindow('image')
# create trackbars for color change
cv2.createTrackbar('R','image',0,255,nothing)
cv2.createTrackbar('G','image',0,255,nothing)
cv2.createTrackbar('B','image',0,255,nothing)
# create switch for ON/OFF functionality
switch = '0 : OFF \n1 : ON'
cv2.createTrackbar(switch, 'image',0,1,nothing)
while(1):
    cv2.imshow('image',img)
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break
    # get current positions of four trackbars
    r = cv2.getTrackbarPos('R','image')
    g = cv2.getTrackbarPos('G','image')
    b = cv2.getTrackbarPos('B','image')
    s = cv2.getTrackbarPos(switch,'image')
    if s == 0:
img[:] = 0
    else:
img[:] = [b,g,r]
cv2.destroyAllWindows()
```

This program can of course easily be modified for specific application

CONCLUSION

It can be concluded, that due to SimpleCV not being developed anymore, the main choice is between Pillow

and OpenCV. If only basic image processing is needed, Pillow might be better. In Image analysis and Computer vision, OpenCv is however still the best package to choose.

REFERENCES

[1] https://en.wikipedia.org/wiki/Python_(programming_language)

[2] C.Fuhrer, J. E. Solem, and O.Verdier, "Scientific Computing with Python 3,"Packt Publishing Ltd, 2016.

[3] S. Nagar, "Introduction to Python: For Scientists and Engineers," Bookmuft, 2016.

[4] M. L. Hetland, "Beginning Python: from novice to professional, 3rd Ed.,"Apress, 2017.

[5] R. V. Hattem, "Mastering Python: master the art of writing beautiful and powerful Python by using all of the features that Python 3.5 offers," Packt Publishing, 2016.

[6] http://www.pythonware.com/products/pil/

[7] http://python-pillow.org/

[8] https://en.wikipedia.org/wiki/Python_Imaging_Library

[9] A.Kaehler, andG.Bradski, Learning OpenCV: computer vision in C++ with the OpenCV library, 2nd Ed.," O'Reilly,2016.

★★★