Detection and face recognition

1st Seliman David
CTI (of 1306B)
Faculty of Automatic Control and Computer Engineering
Iasi, Romania

2nd Arsene Stefan
CTI (of 1307B/)
Faculty of Automatic Control and Computer Engineering
Iasi, Romania

I. Introduction

Python is a widely-used, high-level programming language created by Guido van Rossum in 1991. It prioritizes code readability and supports various programming paradigms, including object-oriented, functional, imperative and procedural. It has a large standard library and multiple versions have been released, with Python 2.0 in 2000 and Python 3.0 in 2008. Python is a good choice for all the researchers in the scientific community because it is [1]:
• 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
• abundance of open-source libraries, particularly those for scientific computing, such as libraries for linear algebra, visualization, plotting, image analysis, solving differential equations, symbolic computation, and statistics.
• useful in a wider setting: scientific computing, scripting, web sites, text parsing, etc.
• widely used in industrial applications
Python is a higher-level programming language compared to languages like C/C++, Java, and Fortran. It is easier to program in, but may have slightly longer computation time. Wrappers are available for C and Fortran. Other high-level languages like PHP and Ruby exist, but Ruby lacks the scientific libraries that Python has, and PHP is primarily used for web development. Python is similar to Matlab in its extensive scientific library, but Matlab is not open-source and free. Scilab and Octave are open-source alternatives to Matlab, but their language features are not as advanced as those of Python. Python has a simple and easy-to-learn approach to solving complex problems, with a flat learning curve and development process for software engineers. [4]. Python is used in various industries and applications, such as system administration tasks, by NASA for development and scripting in multiple systems, special effects production in large-budget films by Industrial Light Magic, management of discussion groups by Yahoo, and implementation of web crawlers and search engines by Google.[3]. Python is easy to learn, powerful, and convenient, making it a widely adopted language. It also has a large collection of libraries that can be imported for specific tasks, such as NumPy and SciPy for scientific computing, particularly in mathematics. NumPy is a library that supports large, multi-dimensional arrays, which are commonly used in image processing tasks[2]. NumPy and SciPy are essential libraries for any scientific paper dealing with mathematics. NumPy provides support for large, multi-dimensional arrays, which are useful in image processing tasks as images are often represented as large 2D (greyscale) or 3D (color) matrices. Many other libraries also use NumPy array representation. SciPy is built on top of NumPy and contains modules for signal and image processing, linear algebra, and fast Fourier transform. Matplotlib is also an important library for visualization and plotting. Matplotlib is widely used in various scientific fields, but it is particularly important in image processing. This paper discusses the capabilities of libraries such as NumPy, SciPy, and Matplotlib in the field of image processing, analysis, and computer vision. It also demonstrates the execution of some common algorithms in the field and provides the resulting images.

II. Python's image processing libraries

This paper presents several Python libraries related to image processing and computer vision, such as:
• PIL/Pillow This library is mainly suitable for basic image manipulations (such as rotation and resizing) and simple image analysis tasks (such as creating a histogram).
• SimpleCV There are multiple Python libraries available for image processing and computer vision, such as PIL/Pillow, SimpleCV, OpenCV, and Ilastik. PIL/Pillow is suitable for basic image manipulation and simple image analysis. SimpleCV is a simplified version of OpenCV, making it easier to learn and use, while OpenCV is a powerful, widely used library written in C/C++ with added Python bindings, and it focuses on real-time image processing. Ilastik is a user-friendly tool for interactive image classification, segmentation, and analysis. SimpleCV is a library that aims to be a simplified version of OpenCV. It offers fewer capabilities compared to OpenCV but is easier to learn and use.
• OpenCV OpenCV is a highly capable and widely used computer vision library. It is developed in C/C++, but it also comes with Python bindings. OpenCV is particularly strong in real-time image processing. Another notable library in the field is Ilastik, which is a user-friendly tool for interactive image classification, segmentation, and analysis.

A. Python Imaging Library

The Python Imaging Library (PIL), originally developed by Fredrik Lundh[5], is a library for image processing that is a little outdated as its last release was in 2009. Its successor, which supports Python 3, is called Pillow[6] and as a result, both libraries cannot be installed at the same time. At the time of writing, the latest version of Pillow is 5.1.0. A basic example of using the library is to display an image, which can be done as follows:

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

Pillow is a powerful library that can extract a lot of information from an image and allows for the execution of various standard image manipulation procedures, such as:

• 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 these capabilities will be demonstrated in the following code. For example, an image can be easily rotated by a specific angle (in this case 45 degrees) and then saved:

$rotated_image = im.rotate(45)$
$rotated_image.save('rotated.jpg')$

A color image can also be split into different components (red, green and blue).

$r, g, b = im.split()$
$r.show()$

An image can also be easily sharpened or blurred. However, in this case, it is important to also import the ImageFilter library. The code needed is shown below.

from PIL import ImageFilter
$sharp = im.filter(ImageFilter.SHARPEN)$
$blur = im.filter(ImageFilter.BLUR)$

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

$cropped_im = im.crop((100, 100, 400, 400))$

It has been demonstrated that PIL/Pillow is very easy to use for basic image processing tasks. For more advanced analysis and computer vision tasks, libraries like SimpleCV and OpenCV are more suitable.

B. OpenCV

OpenCV is an open-source computer vision library written in C and C++ that is compatible with Linux, Windows, Mac OS X, iOS, and Android. It also has interfaces available for Python, Java, Ruby, Matlab, and other languages. A very basic program for showing an image can be written as follows:

$importnumpyasnp$
$importcv2$
$img = cv2.imread('lena - color.jpg')$
$cv2.imshow('image', img)$
$cv2.waitKey(0)$
$cv2.destroyAllWindows()$ As OpenCV is currently the most suitable library for computer vision, it will be used in the remaining part of the paper.

C. Face Detection

OpenCV allows us to perform even more complex tasks relatively easily. For example, it has routines for detecting faces (and eyes) in an image. The following sequence of commands demonstrates this capability.

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

The result is shown in Figure 1. For the image in Figure 1, the algorithm works perfectly. However, if a more complex image is used, the result (especially for eyes) is not as good, as seen in Figure 2. The algorithm uses Haar feature-based cascade classifiers, which was proposed as an effective object detection method by Paul Viola and Michael Jones [9]. The number of these features can be huge, but most of them are irrelevant. A good feature for example could be the fact that the region of the eyes is typically darker than the region of the nose and cheeks. Another good feature could be based on the fact that the eyes are usually darker than the bridge of the nose. As more such features are added, the reliability of the algorithm increases. However, misclassifications can still occur. It should also be noted that the reliability decreases with a decrease in the number of pixels in the face area.

D. Face Recognition

Face recognition is a research area in which images of faces are grouped into sets that belong to a single individual. An example of this is Facebook. In the past, Facebook was able to detect faces (as previously mentioned), but the user had to manually tag the person by clicking on the image and specifying their name. Now, Facebook can automatically tag everyone in the image using face recognition algorithms. In Python, this task can be accomplished using pre-trained convolutional neural networks and OpenCV. The program relies heavily on various libraries such as modules paths, face recognition, argparse, pickle and os which have to be imported into the Python project. Initially, some images of the person to be recognized must be collected. This can be done manually

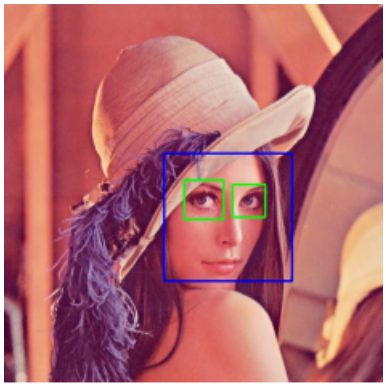

Fig. 1. Example of face detection

Fig. 2. Example of multiple face detection

or by using Microsoft's Bing API for searching. Ideally, the dataset should contain at least 30 images of each person and no other persons should be present in the images used for training. Figure 3 shows two sample images, one for each person. The network architecture used for face recognition is based on the ResNet-34 neural network [10]. However, the Python's face recognition library has fewer layers and the number of filters is reduced by half. The network was trained on a dataset of approximately 3 million images, primarily the VGG dataset [11] and the scrubs dataset [12].

The algorithm is composed of four steps:

1.Finding all the faces:

In the first step, the face detection algorithm described in the

Fig. 3. Examples of persons to detect

Fig. 4. Original image and its histogram of gradients

Fig. 5. Face with landmarks

previous section [9] can be used, which is the most commonly applied method. However, the face recognition library uses a more advanced Histogram of Oriented Gradients (HOG) method [13]. Color images must first be converted to grayscale images. Then, for each pixel in the image, we look at the direction in which the image is getting darker, resulting in a matrix of gradients (see Figure). This matrix is largely independent of the brightness variations in the original image. However, it is too large to manipulate, so submatrices of 16x16 size are formed. Then, the predominant direction for each submatrix is found.

2. Posing and Projecting Faces

This step addresses the issue that faces in an image may be facing a different direction and not straight into the camera. There are several solutions to this problem. Python's library uses the approach of 68 landmarks that are present on any face [14]. An example of such an image is shown in Figure 5. Then, a machine learning algorithm is trained to locate these 68 landmarks on any face. After that, the face is transformed using affine transformations so that the eyes and mouth are centered as best as possible.

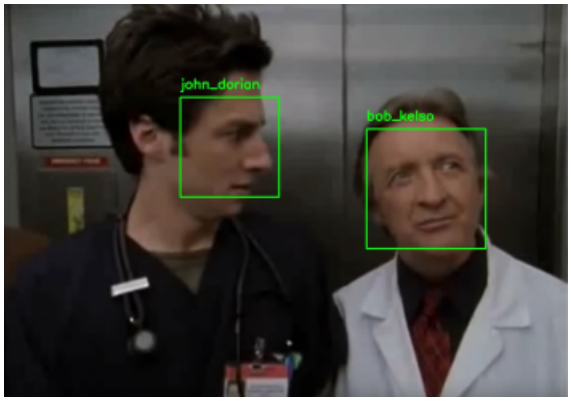

Fig. 6. Example of the face detection algorithm

3.Encoding Faces
TThe Deep Convolutional Neural Network is trained to generate 128 measurements for each face. The training process uses three images at a time (an image of a known person, another image of the same person, and an image of a different person) [15]. This step requires a large dataset and a significant amount of computational power. However, it only needs to be executed once. Pretrained neural networks are also available on the internet.
4. Finding the persons name from the encoding
The final step is relatively simple. The face being analyzed is compared to the faces in the database. Python's library uses a Support Vector Machine (SVM) for this purpose. In principle, any other classification algorithm could be used.
The performance of the algorithm is demonstrated on the image shown in Figure 6. It can be seen that it works well, despite the fact that both Bob Kelso and John Dorian are not looking straight into the camera. In the case of John Dorian, the head is positioned almost perpendicular to the camera axis. It should also be noted that the whole image is much darker than an average one, yet the algorithm was able to recognize both of them perfectly. Another example is shown in Figure 7. In this case, there is no problem with brightness. John Dorian is again looking in the direction of Bob Kelso and his emotion is expressed by his facial expression. In the case of Bob Kelso, we can see that he is smoking a pipe and also has an uncommon facial expression. Despite this, the algorithm was able to correctly recognize both of them.

III. Conclusion

The paper is divided into two parts. The first part provides a brief overview of the most commonly used Python libraries for image processing and computer vision. The second part focuses on the OpenCV library. In this part, the libraries for face detection and face recognition are described and analyzed. Face detection and face recognition are areas of active research because they enable better interaction between computer systems or robots and humans. The Python's face recognition library is shown to be a fast and reliable tool for face detection and recognition. As Python is a high-level programming language, the library is well suited for use as a face detection (recognition) procedure in a larger project without the need for a detailed understanding of the underlying algorithms. Therefore, it has a promising future. In the future, it would be interesting to explore the capabilities of Python and its libraries for emotion detection. This field is a highly relevant topic in human-machine interface research. The results of this research can be used to greatly improve the social aspects of robots or software packages that can adapt to the user, providing reliable feedback in human-machine interactions.

References

[1] C. Fuhrer, J. E. Solem, and O. Verdier, Scientific Computing with Python 3, Packt Publishing Ltd, 2016. (references)
[2] S. Nagar, Introduction to Python: For Scientists and Engineers, Bookmuft, 2016.
[3] M. L. Hetland, Beginning Python: from novice to professional, 3rd Ed., Apress, 2017.
[4] 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.
[5]] http://www.pythonware.com/products/pil/
[6] http://python-pillow.org/
[7] https://en.wikipedia.org/wiki/Python Imaging Library
[8] A. Kaehler, and G. Bradski, Learning OpenCV: computer vision in C++ with the OpenCV library, 2nd Ed.," O'Reilly, 2016.
[9] P. Viola, and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, pp. I-511-I-518, 2001.
[10] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition," In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778, 2016.
[11] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," In British Machine Vision Conference, vol. 1, no. 3, p. 6, September, 2015.
[12] H. W. Ng, and S. Winkler, "A data-driven approach to cleaning large face datasets," IEEE International Conference on Image Processing (ICIP), pp. 343- 347, 2014.
[13] N. Dalal, and B. Trigs, "Histograms of oriented gradients for human detection," IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, pp. 886-893, 2005.
[14] V. Kazemi, and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1867-1874, 2014
[15] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 815-823, 2015.