POSSIBILITIES OF PYTHON BASED EMOTION RECOGNITION

Primož Podržaj and Boris Kuster

Faculty of Mechanical Engineering, University of Ljubljana,
Askerceva 4, 1000 Ljubljana, Slovenia

ABSTRACT

Vision is probably the most important sense for human beings. As a consequence, our way of behaviour and thinking is also often based on visual information. When trying to perform complex information especially in situations where humans are involved, it is of great benefit if some information can be obtained from images. This is the field of image processing and computer vision. There are various libraries available for these tasks. Probably the best known one is OpenCV. It can also be used in Python programming language. Simple and more complex image processing algorithms are already available in the library. One of the more complex ones is face detection. In this paper it shown how face detection can be executed within Python with OpenCV library. This is the first step needed in emotion recognition. When face is detected, we can determine the emotional state of the subject using a special purpose library.

KEYWORDS

Image processing, Python, OpenCV, face detection, emotion recognition.

1. INTRODUCTION

Of the five senses (vision, hearing, smell, taste, and touch) vision is undoubtedly the one that man has come to depend upon above all others, and indeed the one that provides most of the data he receives [1]. Actually almost all animal species use eyes in fact evolution has invented the eye many times over [2]. The main reason for this is that eyes are very effective sensors for recognition, navigation, obstacle avoidance and manipulation. Artificial sensors that mimic the function of an eye are cameras. Computer vision is inspired by the capabilities of the human vision system and could be defined as the automatic analysis of images and videos by computers in order to gain some understanding of the world [3]. An important part of computer vision is image processing, which means transforming an image in some way.

In this paper some Python libraries that make it possible to perform basic image processing tasks will be introduced. OpenCV, which is the most important one will be presented in some detail. It enables some not so basic image processing (we could say it already enters computer vision). One example is face detection. This is the first step in reaching the main topic of the paper, which is emotion recognition.

Natarajan Meghanathan et al. (Eds) : SPPR, SCAI, CSIA, WiMoA, ICCSEA, InWeS, NECO, GridCom - 2018
pp. 31–40, 2018. © CS & IT-CSCP 2018 DOI : 10.5121/csit.2018.81704

2. IMAGE ACQUISITION AND MANIPULATION

In order to analyze any visual information, we must first get it into an appropriate form. A typical setup for obtaining in image is shown in Fig. 1. The image acquisition process starts with an illumination source, where the light rays are coming from. The scene element is the object under observation. The rays that come from the illuminations source are either reflected or absorbed by the object. Then the imaging system (typically optical lens) collects the incoming light an focuses it on the imaging plane. This is the plane where the sensor should be located. The sensor actually measures the amount of energy received at a specific location. If we want to get a color image, we would have to use filters for each of the three primary colors ((R)ed, (G)reen and (B)lue). As this is too cumbersome, the sensor elements can be arranged in the so called Bayer pattern as shown in Fig. 2. So, for each element we know only one of the colors (note that due to the sensitivity of the human eye there are twice as many green pixels than red or blue).

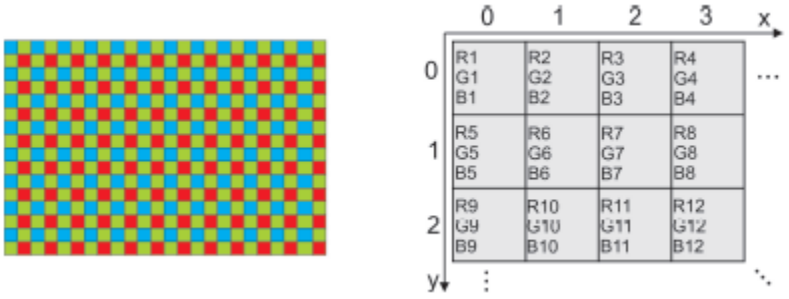

Fig. 2. Bayer pattern (left) and mathematical representation of digital image (right) [5]

The process of obtaining the values for other pixels is called demosaicing. There are various algorithms for this task, which of course differ in execution time. The most simple one just infers the missing color components from the nearest pixel with that color. After demosaicing we are left with a three dimensional matrix. It is however better to speak of a two dimensional matrix, where each element has three components as shown in the left image of Fig. 2. This is also a mathematical representation of an image and also represents the starting point for any image manipulation. In general image manipulation can be defined as any mathematical operation that transforms the original two dimensional image into another one. There are many possible divisions of these opeartions. In this short introduction we will divide them into two groups:

1. Point processing operations
2. Neighborhood processing operations

Point processing is an operation where the intensity value (in monochromatic image) or intensity values (color image) in the transformed image i_t depend only on the intensity value (values) of the corresponding pixel in the original image i_o.

$$i_t(x, y) = f(i_o(x, y)) \tag{1}$$

Although, being a very simple operation, it has some applications. A typical one is full-scale histogram stretch which is easily the most common linear point operation [6]. Another important operation is for example the conversion of a color image to a grayscale (monochromatic) one using the following equation [7]

$$i_t(x,y) = 0.299 \cdot i_{oR}(x,y) + 0.587 \cdot i_{oG}(x,y) + 0.114 \cdot i_{oB}(x,y) \quad (2)$$

Neighborhood processing is an operation where the intensity of a specific pixel in the original image depends on the intensity of more than pixel in the original image. Typically the corresponding pixel and some pixels in the neighborhood (that's where the operation gets its name from). A typical operation of this kind is filtering used to remove or at least decrease the noise in an image. One possible (very simple) formula for such an operation (using only four neighboring pixels) can be written as follows:

$$i_t(x,y) = \frac{i_o(x,y-1) + i_o(x-1,y) + i_o(x,y) + i_o(x+1,y) + i_o(x+1,y+1)}{5} \quad (3)$$

It probably doesn't need to be stressed out that neighborhood processing is much more powerful. As a consequence almost all the applications need that kind of operations.

3. PYTHON PROGRAMMING LANGUAGE

3.1 Some Basic Information

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 [8]:

- 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 to think that complex problems demand complex processes in order to produce complex solutions. Python was developed with exactly the opposite philosophy. It has an extremely at learning curve and development process for software engineers [9]. 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 components of its web crawler and search engine [10]. As Python is also a language that is easy to learn and both powerful and convenient from the start [11], 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.

3.2 Basic Image Processing Libraries

There are several Python libraries related to Image processing and Computer vision.
The most important ones are however:

- 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. By far the most important library is however the OpenCV library. Some of its capabilities will be demonstrated.

3.3 OpenCV library

OpenCV is an open source computer vision library available written 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 (see Fig. 3) 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. 3.

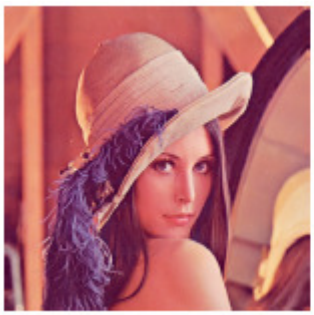

Fig. 3. Lena color image

Fig. 4. Binary image

```
gray_image = cv2.cvtColor (img, cv2.COLOR_BGR2GRAY)
ret, binary_image = \
       cv2.threshold (gray_image, 127,255,cv2 .THRESH_BINARY)
cv2.imshow( 'image', binary_image)
```

One of the important image processing tasks is edge detection. In OpenCV this task can be performed using a simple command:

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

If the source image is the one shown in Fig. 3, we get the resulting image as shown in Fig. 5. 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.

4. COMPUTER VISION TASKS RELATED TO EMOTION RECOGNITION

Emotion recognition is quite a complex task to achieve. Basically it is divided into two steps (face detection and emotion recognition).

4.1 Face Detection

The task of face detection is (as the words suggest) to find the face(s) (sometimes also 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 . imwrite ( ' face_lena. jpg ', img )
cv2 . waitKey ( 0 )
cv2 . destroyAl lWindows ( )
```

The result is shown in Fig. 6. For the image in Fig. 6 algorithm works perfectly.

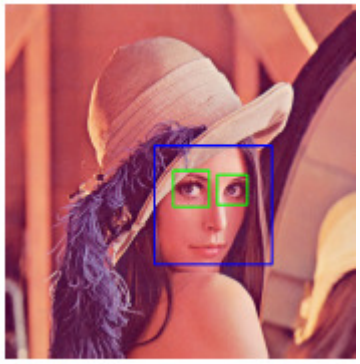

Fig. 6. An example of face detection

If however a more complex image is used, the result (especially for eyes) is not so good. See for example Fig. 7. The algorithm itself applies the so called Haar feature-based cascade classifiers. It was proposed as an effective object detection method by Paul Viola and Michael Jones [12]. The number of these features can be enormous. But most of them are irrelevant. A good feature is for example the fact that the region of the eyes is usually darker than the region of the nose and cheeks. A second good feature could for example be based upon the fact that the eyes are usually darker than the bridge of the nose. With the increasing number of such features, we can increase the reliability of the algorithm. Misclassifications are of course always a possibility. It should also be noted, that the reliability decreases with the decreasing amount of pixels in the face area.

Fig. 7. An example of face detection for many people being on the image

5. EMOTION RECOGNITION

Emotion recognition is very difficult computer vision task. In essence, what we expect from this task is to group faces in an image into one of the following seven emotions: angry, disgust, fear, happy, sad, surprise, neutral. The principle behind the algorithm is the so called deep learning shown schematically in Fig. 8. The input signal are the faces detected. After they are detected (as described above), they are cropped out of the image and fed into a convolutional neural network. Convolutional networks are a neural network architecture particularly well suited to processing images. A "kernel" is slid over the image and multiplied with its pixel values. A kernel's weights are learned using backpropagation. In effect, kernels find patterns in the image regardless of their position in an image, which allows the convolutional neural network to be spatially invariant.

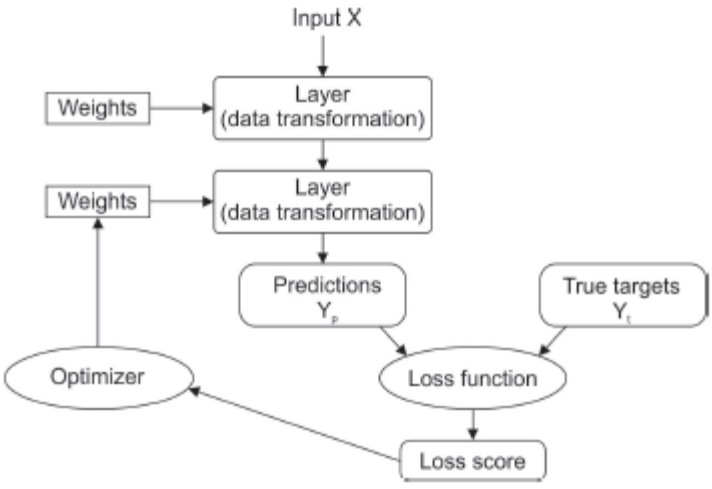

Fig. 8. Schematic representation of deep learning concept [13]

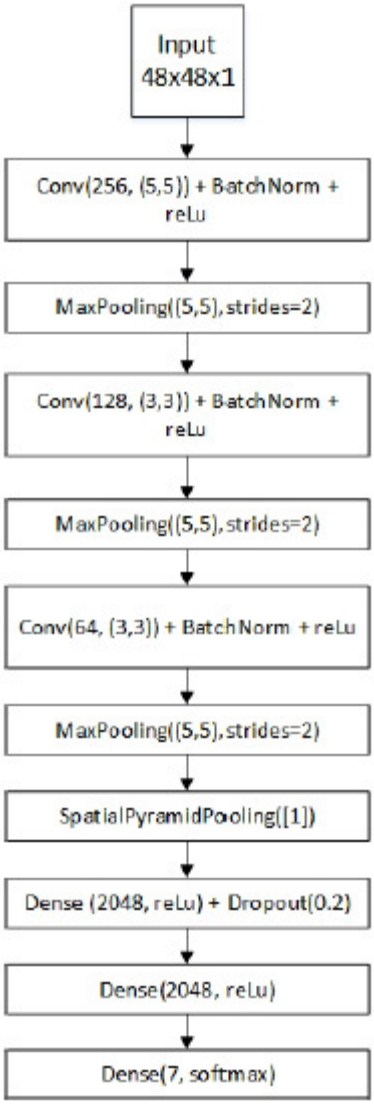

Fig. 9. Convolutional Neural Network used in the paper

Our particular implementation of the CNN is shown schematically in Fig. 9. It consists of three convolutional layers, each followed by a Batch Normalization layer, Rectified Linear Unit (relu) activation layer and max pooling, to reduce the representation size. After this, a Spatial Pyramid Pooling layer enables us to input pictures of arbitrary input shape. Some regularization is applied to the convolutional kernels to reduce the possibility of over-fitting. The CNN layers extract features from the images, which are then fed into a multi layer perceptron neural network, with the layers containing 2048, 2048 and 7 neurons, respectively. Some Dropout (0.2) is applied after the first dense layer, to combat over-fitting. The output layer contains 7 neurons and a softmax activation function, since we classify faces into one of the above mentioned seven emotions. We use the Adam optimizer with the default learning rate of 0.01. In order to train the network, the Facial Emotion Recognition dataset from Kaggle has been used. It contains around 30000 grayscale pictures of 48x48 dimension, each containing a centered face. The faces emotions are labeled.

Fig. 10. Results of the program

Some examples of images being fed to our program are shown in Fig. 10. Not all of the images are classified correctly. An example of such a case is shown in Fig. 11. The accuracy of the network on test set is around 50%, while the best result on Kaggle leaderboard achieved around 70% accuracy. We suspect a dataset of higher resolution face images would increase the recognition ability of the network.

Fig. 11. An example of a wrong emotion recognition

6. CONCLUSION

The concept of deep learning has probably very bright future ahead. Some people even say that it might be the concept which will finally rival human intelligence in many areas. In this paper the deep learning concept has been used to asses the possibilities of emotion recognition in images.

At first some very basic information about image acquisition and its mathematical representation is given. This is followed by the introduction of python and its comparison with some other popular programming languages. As the topic of the paper is image processing, related libraries are given. OpenCV, being most popular is described in some detail. Face detection and emotion recognition are outlined as well. The results show that further research is needed in order to increase the accuracy of the approach.

REFERENCES

[1] E. R. Davies, Computer and machine vision: theory, algorithms, practicalities. Academic Press, 2012.

[2] P. Corke, Robotics, Vision and Control: Fundamental Algorithms In MATLAB, 2nd Ed..Springer, 2017.

[3] K. Dawson-Howe, A practical introduction to computer vision with opencv. John Wiley & Sons, 2014.

[4] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 3rd Ed.. Pearson Education, 2008.

[5] T. B. Moeslund, Introduction to Video and Image Processing - Building Real Systems and Applications. Springer, 2008.

[6] A. C. Bovik, The essential guide to image processing. Academic Press, 2009.

[7] G. Blanchet and M. Charbit, Digital signal and image processing using MATLAB, Vol. 1 - Fundamentals. ISTE, 2014.

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

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

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

[11] 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.

[12] P. Viola, and M. Jones, Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, pp. I-511-I-518, 2001.

[13] F. Chollet, Deep learning with Python. Manning Publications Co., 2017.

AUTHORS

Primož Podržaj received his PhD from the University of Ljubljana where he is an Associate Professor in the field of Control Theory. He is the head of the Laboratory for Process Automation. His research interest includes machine vision, image processing and artificial intelligence.

Boris Kuster received his bachelor's degree in mechanical engineering from the University of Ljubljana, where he is now studying for a master's degree in mechatronics. His primary interests are machine learning, artificial intelligence and robotics.