Image Processing with Python Programming

Atsushi Matsubara, Seiko Yamano

Production Division

1 Introduction

Image processing extracts the (potential) features from two-dimensional information, such as camera images or spatial-temporal series (reflected waves per scanning point) obtained from mobile observation devices using Photoshop skills (e.g., compositing, combining, transforming) and analytical approaches. Hence, the image processing has been widely used in scientific-specialized fields, evaluates, classifies or diagnoses information-related targets. Furthermore, Python, one of development languages, can achieve relatively simple and speedy implementations of analytical programs for the image processing, because the language has sufficient libraries for the processing.

This paper discusses programming skills for the image processing using Python. We specifically introduce four well-known and useful methods for the processing: background subtraction, edge detection, color separation /combination and face recognition. In addition, we install OpenCV into the Python development environment. OpenCV, one of the open-source libraries, can reduce program sizes for image processing because it replaces the image analysis algorithms with the special functions and the combination of simple codes[1)]. OpenCV thus has friendly popularized as a library for object-oriented programming languages such as Python, Java, Visual Basic and C# as well as C++.

We build the analytical programs of these four methods using Python (ver.3.9.12) on Spyder (ver.5.15) with a platform called Anaconda3[2)]. We adopt photos obtained from a webcam (equipped on PC or connected via USB port) to image information.

2 Image Processing

2.1 Background Subtraction

Background subtraction, an image processing method to extract information (target) from current data which is not included in previous data, has been applied to detect moving objects or people. Data (Frame information) obtained in the prior while-loop processing are often assigned to previous data, although the timing to obtain the previous data depends on the situation of background and the behavior of extracted information.

Code 1 shows a sample program for this processing. Implementation of cv2 module using import statement draws the features of OpenCV. VideoCapture class, which recognizes and opens a device (webcam) for capturing images, must set a distinctive identification index (ID) automatically assigned to the device when calling and driving its own constructor. The instance, the object (named 'cap') created for the class, obtains image information using read () function. Eventually, we store the returns of this function (image information) at a specific memory location assigned a variable named 'frame'

The processing of the background subtraction has the following procedure: first, createBackgroundSubtractorMOG () function belonging to bgsegm namespace creates an instance of BackgroundSubtractor class; next, apply () function of the instance obtains a processed image (foreground mask image); finally, the algorithm stores the image at a specific memory location – we assign the memory address to a variable 'mask'.

For a presentation (display) of the result, we horizontally put original and processed images using hstack () function in numpy module. However, we must pay close attention to the following in the parallel presentation for these two images: (1) the data structure of the processed image is 2D with width and height, whereas that of the original image is 3D

including color information as well as width and height (the former image is stored in a 2D array, whereas the later is stored in a 3D array); (2) we need to fit the dimension sizes of two data structures to display these images together. These noticed points are due to the fact that the image processing needs to convert the original image to the grayscale image of two-dimensional structuring.

```
import cv2
import numpy as np
import copy as cpy

cap= cv2.VideoCapture(0)
fps=int(cap.get(cv2.CAP_PROP_FPS))
t=int(1000/fps)
bk=cv2.bgsegm.createBackgroundSubtractorMOG()
while True:
    ret,frame=cap.read()

    frm=cpy.copy(frame)
    mask=bk.apply(frm)
    threeDimg=np.stack([mask,mask,mask], axis=2)
    img=np.hstack((frame,threeDimg))
    cv2.imshow('    ',img)

    key=cv2.waitKey(t)
    if key==ord('z'):
        print("key=",key)
        break
cap.release()
cv2.destroyAllWindows()
```

Code1. Background Subtraction

Therefore, we implement a procedure to increment the dimensions of the data structure for the processed images (i.e., 2D→3D) using stack () function in numpy module. The stack () function concatenates or stacks several data-storing arrays to create the higher dimensional array (expanding memory area), and an axis mode of the function determines where to stack the arrays. We specifically set the following policy: (1) creating a 3D array stacking multiple 2D arrays; (2) preparing three processed images (2D array: the variable 'mask') because the number of the third elements in the 3D array storing an original image is three, and equals to that of color component: 'R', 'G', 'B'; (3) setting the axis mode to 2 to stack the three 2D arrays, aiming at the position of the third element in the 3D array.

We thus store the processed image as the novel image in a memory area for a variable 'threeDimg'.

Python establishes the call by reference for two array variables when connecting them with '=' operator. We therefore temporarily stock the copy of the original images in a memory area for a variable 'frm'. We furthermore control the while loop and the display of the images using waitKey() function. The function keeps the display of the images for given time or until pushing a certain key. We set the time to the inverse of the webcam's frame rate (= 30 fps), and use imshow () function for displaying.

Figure 1 shows the original and processed images, where the right side image shows that the white area represents the motion.

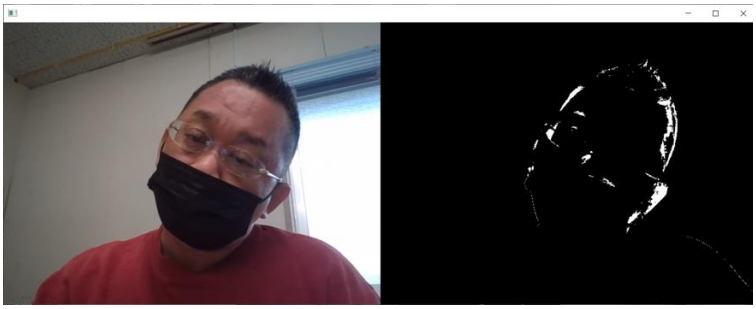


Figure1. An example for the results of Background Subtraction
(left: original, right: processed)

2.2 Edge Detection

Edge Detection focuses on the contrast of image information transformed to grayscale, and can extract the outline of an object. Canny method, a well-known and fundamental detection, uses the gradients of the neighboring pixel values of image information after smoothing it.

Code 2 shows a sample program for edge detection using Canny method. The variables ('min_val' and 'max_val'), the second and third arguments in Canny () function, mean the minimum and maximum thresholds for the detection respectively, which are compared to the difference (d) of the neighboring pixel values. The detection depends on the following Boolean logic: (1) 'true' if $d \geq \max\_val$; (2) 'false' if $\min\_val \geq d$; (3) in the case of $\min\_val < d < \max\_val$, 'true' if a certain d is adjacent to the differences (d') recognized as 'true', 'false' otherwise. We set these thresholds whenever capturing an image. We specifically calculate the center point (the variable 'mad') of grayscale

```
import cv2
import numpy as np
import copy as cpy

cap= cv2.VideoCapture(0)
fps=int(cap.get(cv2.CAP_PROP_FPS))
t=int(1000/fps)
sig=1/3

while True:
    ret,frame=cap.read()
    frm=cpy.copy(frame)
    frm_g= cv2.cvtColor(frm, cv2.COLOR_BGR2GRAY)

    med=np.median(frm_g)
    min_val=int((1-sig)*med)
    max_val=int((1+sig)*med)

    edge=cv2.Canny(frm_g,min_val,max_val)

    threeDimg=np.stack([edge,edge,edge], axis=2)
    img=np.hstack((frame,threeDimg))
    cv2.imshow('      ',img)

    key=cv2.waitKey(t)
    if key==ord('z'):
        print("key=",key)
        break
cap.release()
cv2.destroyAllWindows()
```

Code2. Edge Detection

image's pixel values using median () function in numpy module, and then set the variable 'max_val' to the value that adds the variable 'mad' to the ratio (sig×mad), whereas the 'min_val' to the value that subtracts the raito from the variable 'mad'.

Figure 2 shows a sample detection when the minimum and maximum thresholds are 68 and 136, respectively.

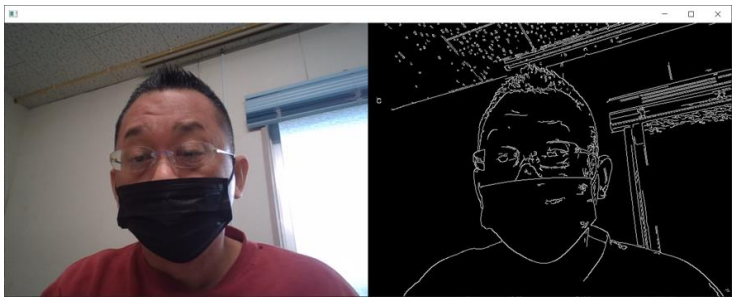


Figure2. An example for the results of Edge Detection (left: original, right: processed)

2.3 Color Separation and Combination

A color image information contains three primary colors of light (Red, Green, Blue), each of which is represented by an 8-bit integer (=2^8). Separation into these color components allows us to obtain knowledge about their content rates and combinations in the image information. In addition, the image combination of the modified or replaced version of these color components allows us to provide a novel image.

Code 3 shows a sample program for color separation and combination: the former uses split () function, the latter does merge () function.

The split () function in cv2 module enables us to obtain the three primary color components of 'R', 'G', 'B' from an image information, which are stored in 3D array (tuple type) named 'img_BGR'. The 3D array has color information in its first element ('B', 'G', 'R' order).

The merge () function achieves the editing and the restructuring of an image. We set the arguments of the function assigned to non-target colored components to zero values, in order to create a target colored image. For example, we set the arguments for 'B' and 'G' to zeros in the case of a colored image based on 'R'.

The processed results display the original and the blue-

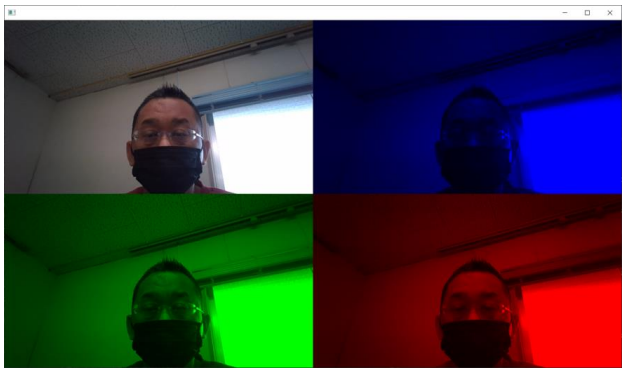
```
import cv2
import numpy as np
import copy as cpy

cap= cv2.VideoCapture(0)
fps=int(cap.get(cv2.CAP_PROP_FPS))
t=int(1000/fps)
width=int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height=int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
while True:
    ret,frame=cap.read()
    frm=cpy.copy(frame)
    img_BGR=cv2.split(frm)
    zeros=np.zeros((height,width),frm.dtype)
    Bule=cv2.merge((img_BGR[0],zeros,zeros))
    Green=cv2.merge((zeros,img_BGR[1],zeros))
    Red=cv2.merge((zeros,zeros,img_BGR[2]))

    img1=np.hstack((frame,Bule))
    img2=np.hstack((Green,Red))
    img=cv2.vconcat([img1,img2])
    cv2.imshow('      ',img)

    key=cv2.waitKey(t)
    if key==ord('z'):
        print("key=",key)
        break
cap.release()
cv2.destroyAllWindows()
```

Code3. Color Separation and Combination

colored images horizontally at the top of a window, whereas the green-colored and the red-colored images are similarly at the bottom, using appropriate hstack () and cv2.vconcat () functions.

Figure 3 shows the processed results: the left side displays the original and three color-separated images, the right side displays the newly combined image exchanged 'R' for 'B' of the original image at the bottom. We use the merge () function for the exchange as follows:

cv2.merge((img_BGR[2], img_BGR[1], img_BGR[0]))



(a) separated into the three color components

(b) combination(bottom)

Figure3. An example for the results

2.4 Face Recognition

Face Recognition generally uses a cascade classifier. The cascade classifier, which has several small-scale and serial-connected classifiers, operates as the following procedure: (1) if the determination result for recognition from the first classifier is 'true', the classifier sends the image to the second classifier, otherwise it stops the processing; (2) the face recognition succeeds when obtaining determinations of 'true' in all classifiers.

Code 4 is a sample program using the cascade classifier.'haarcascade_frontalface_default.xml' has a learned configuration file that extracts the feature of a person's face using the cascade classifier. We thus specify the file to initial information for CascadeClassifier class.

The face recognition is achieved using the class's detectMultiScale () function, which obtains the recognition areas as a rectangular shape. The performance of the face recognition depends on minSize parameter of the function. Furthermore, we blur areas excluding those of people's face recognition using cv2.blur () function.

```
import cv2
import numpy as np
import copy as cpy
cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
cap= cv2.VideoCapture(0)
fps=int(cap.get(cv2.CAP_PROP_FPS))
t=int(1000/fps)

while True:
    ret,frame=cap.read()
    frm=cpy.copy(frame)
    frm_g= cv2.cvtColor(frm, cv2.COLOR_BGR2GRAY)
    lists=cascade.detectMultiScale(frm_g,minSize=(50,50))
    smooth=cv2.blur(frm_g, (100,100))
    threeDimg=np.stack([smooth,smooth,smooth], axis=2)
    for (x,y,w,h) in lists:
        cv2.rectangle(frm, (x,y), (x+w, y+h), (255, 0, 0), thickness=2)
        img_face=frm[y:y+h,x:x+w]
        threeDimg[y:y+h,x:x+w]=img_face

    img=np.hstack((frame,threeDimg))
    cv2.imshow('    ',img)

    key=cv2.waitKey(t)
    if key==ord('z'):
        print("key=",key)
        break
cap.release()
cv2.destroyAllWindows()
```

Code4. Face Recognition

Figure 4 shows that this processing disenables us to detect a partially hidden face (the reason for this is that a surgical

mask covers person's mouth and nose) whereas it enables us to detect even a face in a picture.

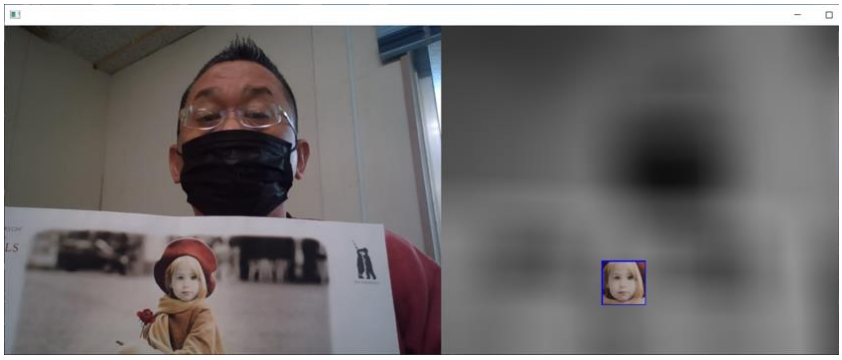


Figure4. Face Recognition (left: original, right: processed)

3 Conclusion

The tenure track system, one of the employee recruitment methods in Advanced Technology Institute, Yamaguchi University, requires the recruited staff to take short-term training in our Institute. We planned this reported programming as one of the training contents, and lectured the programming to this year's target subjects on Monday, 18 December 2023. There were two subjects but they were non-specialists in programming: one was a mechanical engineer, and the other was a chemical analysis one. We carefully explained the above four image processing techniques to them, and step by step created the programs of these methods with them. Thanks for that, we obtained their meaningful comments.

Currently, we have a plan to lecture more enriched contents for image processing to lab's students who need these skills.

References

1) OpenCV-modules, https://docs.opencv.org/4.8.0/index.html
2) Anaconda (Official Web Site), https://www.anaconda.com/