My own OCR-program in Python

Asked 14 years, 11 months ago Modified 4 years, 11 months ago Viewed 23k times

27

I am still a beginner but I want to write a character-recognition-program. This program isn't ready yet. And I edited a lot, therefor the comments may not match exactly. I will use the 8-connectivity for the connected component labeling.

```
from PIL import Image
import numpy as np

im = Image.open("D:\\Python26\\PYTHON-PROGRAMME\\bild_schrift.jpg")

w,h = im.size
w = int(w)
h = int(h)

#2D-Array for area
area = []
for x in range(w):
    area.append([])
    for y in range(h):
        area[x].append(2) #number 0 is white, number 1 is black

#2D-Array for letter
letter = []
for x in range(50):
    letter.append([])
    for y in range(50):
        letter[x].append(0)

#2D-Array for label
label = []
for x in range(50):
    label.append([])
    for y in range(50):
        label[x].append(0)

#image to number conversion
pix = im.load()
threshold = 200
for x in range(w):
    for y in range(h):
        aaa = pix[x, y]
        bbb = aaa[0] + aaa[1] + aaa[2] #total value
        if bbb<=threshold:
            area[x][y] = 1
        if bbb>threshold:
            area[x][y] = 0
np.set_printoptions(threshold='nan', linewidth=10)

#matrix transponation
ccc = np.array(area)
area = ccc.T #better solution?

#find all black pixel and set temporary label numbers
```

```
i=1
for x in range(40): # width (later)
    for y in range(40): # heigth (later)
        if area[x][y]==1:
            letter[x][y]=1
            label[x][y]=i
            i += 1

#connected components labeling
for x in range(40): # width (later)
    for y in range(40): # heigth (later)
        if area[x][y]==1:
            label[x][y]=i
            #if pixel has neighbour:
            if area[x][y+1]==1:
                #pixel and neighbour get the lowest label
                pass # tomorrows work
            if area[x+1][y]==1:
                #pixel and neighbour get the lowest label
                pass # tomorrows work
            #should i also compare pixel and left neighbour?

#find width of the letter
#find height of the letter
#find the middle of the letter
#middle = [width/2][height/2] #?
#divide letter into 30 parts --> 5 x 6 array

#model letter
#letter A-Z, a-z, 0-9 (maybe more)

#compare each of the 30 parts of the letter with all model letters
#make a weighting

#print(letter)

im.save("D:\\Python26\\PYTHON-PROGRAMME\\bild2.jpg")
print('done')
```

python arrays artificial-intelligence ocr

Share

Improve this question

Follow

edited Jan 3, 2010 at 4:28

asked Jan 1, 2010 at 23:14

kame

21.9k ● 35 ● 110 ● 168

Hm ... the devil is in the details. For this to work well, I think you need to have many different fonts loaded. My hunch is that OCR programs cycle through various fonts until they find the one they like. Obviously, there are many papers published on the subject. Why do you want to implement this as one of your first Python tasks? – Hamish Grubijan Jan 1, 2010 at 23:32

1 More clarifications: all is well if your code is black-and-white. However, what if some letters / words are in grey? You want something like the Gimp's "select region by color given threshold" operation. I personally would start by computing the darkness distribution - average darkness + std of the image. I would then start at a "white" spot, and keep on selecting white,

until I identify islands of non-white - those are the potential letters. By the way, you need no randomness - a breadth first search can help you locate all black pixels as well ... the trick is in locating the islands. – Hamish Grubijan Jan 1, 2010 at 23:43

1 My naive approach would be: a) find an iland, b) encircle it, c) remember it's original location in the test, d) remove it from the image (color the remaining area white) and append it to a list of mini-images to process ... that is a way to start. I personally would read up on the existing methods because linear algebra and statistics, etc. might be packing some very powerful things for ya. – Hamish Grubijan Jan 1, 2010 at 23:46

1 Right ... you just described Breadth First Search. Look it up. I recommend that over DFS, because you can stop after N pixels and have a ball rather than spaghetti (not that it matters that much) - because that would be too big for a letter. – Hamish Grubijan Jan 2, 2010 at 0:38

1 True, in theory DFS and BFS should compute the same thing. I jut like BFS better in this case because it can also compute levels for you - can help you "peel the onion". – Hamish Grubijan Jan 2, 2010 at 3:37

4 Answers

Sorted by: Highest score (default)

34

OCR is not an easy task indeed. That's why text CAPTCHAs still work :)

To talk only about the letter extraction and not the pattern recognition, the technique you are using to separate the letters is called **Connected Component Labeling**. Since you are asking for a more efficient way to do this, try to implement the two-pass algorithm that's described in this article. Another description can be found in the article Blob extraction.

EDIT: Here's the implementation for the algorithm that I have suggested:

```
import sys
from PIL import Image, ImageDraw

class Region():
    def __init__(self, x, y):
        self._pixels = [(x, y)]
        self._min_x = x
        self._max_x = x
        self._min_y = y
        self._max_y = y

    def add(self, x, y):
        self._pixels.append((x, y))
        self._min_x = min(self._min_x, x)
        self._max_x = max(self._max_x, x)
        self._min_y = min(self._min_y, y)
        self._max_y = max(self._max_y, y)

    def box(self):
        return [(self._min_x, self._min_y), (self._max_x, self._max_y)]

def find_regions(im):
    width, height  = im.size
    regions = {}
    pixel_region = [[0 for y in range(height)] for x in range(width)]
    equivalences = {}
    n_regions = 0
    #first pass. find regions.
    for x in xrange(width):
        for y in xrange(height):
            #look for a black pixel
            if im.getpixel((x, y)) == (0, 0, 0, 255): #BLACK
                # get the region number from north or west
                # or create new region
                region_n = pixel_region[x-1][y] if x > 0 else 0
                region_w = pixel_region[x][y-1] if y > 0 else 0

                max_region = max(region_n, region_w)

                if max_region > 0:
                    #a neighbour already has a region
                    #new region is the smallest > 0
                    new_region = min(filter(lambda i: i > 0, (region_n,
region_w)))
                    #update equivalences
                    if max_region > new_region:
                        if max_region in equivalences:
                            equivalences[max_region].add(new_region)
                        else:
                            equivalences[max_region] = set((new_region, ))
                else:
                    n_regions += 1
                    new_region = n_regions

                pixel_region[x][y] = new_region

    #Scan image again, assigning all equivalent regions the same region value.
    for x in xrange(width):
        for y in xrange(height):
                r = pixel_region[x][y]
                if r > 0:
                    while r in equivalences:
                        r = min(equivalences[r])

                    if not r in regions:
                        regions[r] = Region(x, y)
                    else:
                        regions[r].add(x, y)

    return list(regions.itervalues())

def main():
    im = Image.open(r"c:\users\personal\py\ocr\test.png")
    regions = find_regions(im)
    draw = ImageDraw.Draw(im)
    for r in regions:
        draw.rectangle(r.box(), outline=(255, 0, 0))
    del draw
    #im.show()
    output = file("output.png", "wb")
    im.save(output)
    output.close()
```

```
if __name__ == "__main__":
    main()
```

It's not 100% perfect, but since you are doing this only for learning purposes, it may be a good starting point. With the bounding box of each character you can now use a neural network as others have suggested here.

Share

Improve this answer

Follow

edited Dec 29, 2019 at 10:59

Cœur
38.6k ● 26 ● 202 ● 276

answered Jan 2, 2010 at 8:34

jbochi
29.5k ● 16 ● 77 ● 90

Hello jbochi. I had the Connected Component Labeling idea before you wrote to me. I will use it in my newer version. – kame Jan 2, 2010 at 22:16

I made a mistake. I watched line for line. I should watch better the whole letter at first and then go to the next letter. like you described before. :) – kame Jan 3, 2010 at 0:32

1 but why north and west pixel (when considering 4-connectivity) and not south and west pixel? i start in the upper left corner and go from left to right. – kame Jan 3, 2010 at 0:40

@kame, You should always test the pixels that you have already tested before. Let's say you found a black pixel at (x=5, y=5). If (4, 5) or (5, 4) are black too, you should give (5, 5) the same region/letter number. If they are both white, create a new letter number. Let me know if you need any help to implement this. – jbochi Jan 3, 2010 at 11:25

I have to prepare for my exam, but I will continue soon – kame Jan 5, 2010 at 12:07

7

OCR is very, very hard. Even with computer-generated characters, it's quite challenging if you don't know the font and font size in advance. Even if you're matching characters exactly, I would not call it a "beginning" programming project; it's quite subtle.

If you want to recognize scanned, or handwritten characters, that's even harder - you'll need to use advanced math, algorithms, and machine learning. There are quite a few books and thousands of articles written about this topic, so you don't need to reinvent the wheel.

I admire your effort, but I don't think you've gotten far enough to hit any of the actual difficulties yet. So far you're just randomly exploring pixels and copying them from one array to another. You haven't actually done any comparison yet, and I'm not sure the purpose of your "random walk".

- Why random? Writing correct randomized algorithms is quite difficult. I would recommend starting with a deterministic algorithm first.
- Why are you copying from one array to another? Why not just compare directly?

When you get the comparison, you'll have to deal with the fact that the image is not exactly the same as the "prototype", and it's not clear how you'll deal with that.

Based on the code you've written so far, though, I have an idea for you: try writing a program that finds its way through a "maze" in an image. The input would be the image, plus the start pixel and the goal pixel. The output is a path through the maze from the start to the goal. This is a much easier problem than OCR - solving mazes is something that computers are great for - but it's still fun and challenging.

Share Improve this answer Follow

answered Jan 2, 2010 at 7:52
dmazzoni
13.2k ● 4 ● 41 ● 34

Hello dmazzoni. In the newer version i don't use randomness. Now I will use DFS or BFS. / Copying from one array to an other? Because I want to compare the letter with the model-letters. / I didn't tell how I want to do the comparison, but I have a plan ;) The thing with the maze is also interesting, but I will do it with OCR despite the warnings. :) – kame Jan 2, 2010 at 22:10

5

Most OCR algorithms these days are based on neural network algorithms. Hopfield networks are a good place to start. Based on the Hopfield Model available here in C, I built a very basic image recognition algorithm in python similar to what you describe. I've posted the full source here. It's a toy project and not suitable for real OCR, but can get you started in the right direction.

> The Hopfield model is used as an autoassociative memory to ***store and recall a set of bitmap images***. Images are stored by calculating a corresponding weight matrix. Thereafter, starting from an arbitrary configuration, the memory will settle on exactly that stored image, which is nearest to the starting configuration in terms of Hamming distance. ***Thus given an incomplete or corrupted version of a stored image, the network is able to recall the corresponding original image.***

A Java applet to toy with an example can be found here; the network is trained with example inputs for the digits 0-9. Draw in the box on the right, click test and see the results from the network.

Don't let the mathematical notation intimidate you, the algorithms are straightforward once you get to source code.

Share

Improve this answer

Follow

edited Jan 3, 2010 at 3:36

answered Jan 3, 2010 at 3:29

J.J.

5,069 ● 2 ● 30 ● 26

I'm more intimidated by the messy python code linked to than the mathematical notation. May I suggest cleaning it up if you intend on having it as part of your answer. – Zoran Pavlovic Dec 16, 2012 at 12:03

4

OCR is very, very difficult! What approach to use to attempt OCR will be based on what you are trying to accomplish (hand writing recongnition, computer generated text reading, etc.)

However, to get you started, read up on Neural Networks and OCR. Here are a few jump-right-in articles on the subject:

http://www.codeproject.com/KB/cs/neural_network_ocr.aspx

http://www.codeproject.com/KB/dotnet/simple_ocr.aspx

Use your favorite search engine to find information.

Have fun!

Share

Improve this answer

Follow

edited Jan 2, 2010 at 8:17

answered Jan 2, 2010 at 8:03

tgiphil

1,252 ● 10 ● 22