CDS540 Assignment 1 - Text Detection from Images Using OpenCV

> Student Name：YANG Yiting
>
> Student ID：5556949
>
> Demo video link：https://youtu.be/tDjD4fxvmNE
>
> GitHub link：https://github.com/guaiguai66666666/CV-.git
>
> Reference link：https://colab.research.google.com/github/r3gm/InsightSolver-Colab/blob/main/OCR_with_Pytesseract_and_OpenCV.ipynb#scrollTo=XXiu6DoRS3qs

OCR with Pytesseract and OpenCV

> 1. 安装必要的软件包
>
> - `tesseract-ocr`: Tesseract OCR引擎，用于提取图像中的文本。
> - `libtesseract-dev`: Tesseract的开发库，包含必要的头文件和库。
> - `pytesseract`: Python包装器，简化了与Tesseract的交互。`-q`选项抑制输出。
>
> 2. 将Google Drive挂载到Colab的工作目录，以便访问图像文件`CVAsg1.jpg`。
>
> 3. 读取图像并使用Pytesseract进行OCR，`custom_config`指定了Tesseract的引擎模式和页面分割模式。

```
!sudo apt install tesseract-ocr libtesseract-dev
!pip -q install pytesseract
```

```
# Mount Google Drive to the working directory of Colab
from google.colab import drive
drive.mount('/content/drive')
```

```
im1 = '/content/drive/MyDrive/Colab Notebooks/CVAsg1.jpg'
```

```
# Use OpenCV (cv2) to read images and pytesseract to perform OCR (Optical character recognition)
import cv2
import pytesseract

image = cv2.imread(im1)

# Adding custom options
custom_config = r'--oem 3 --psm 6'
pytesseract.image_to_string(im1, config=custom_config)
```

Preprocessing for Tesseract

1. 这部分定义了一系列图像预处理函数，包括：

- `get_grayscale`: 转换为灰度图像。
- `remove_noise`: 去除噪声。
- `thresholding`: 二值化处理。
- `dilate`: 膨胀处理。
- `erode`: 腐蚀处理。
- `opening`: 开运算。
- `canny`: 边缘检测。
- `deskew`: 校正倾斜。
- `match_template`: 模板匹配。

2. 对图像进行预处理，并分别进行OCR，比较不同方法的效果。使用`cv2_imshow`显示图像

```
import cv2
import numpy as np
import time

start_time = time.time()
recog_start_time = time.time()

image = cv2.imread(im1)

# get grayscale image
def get_grayscale(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# noise removal
def remove_noise(image):
    return cv2.medianBlur(image,5)

#thresholding
def thresholding(image):
    return cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]

#dilation
def dilate(image):
    kernel = np.ones((5,5),np.uint8)
    return cv2.dilate(image, kernel, iterations = 1)

#erosion
def erode(image):
    kernel = np.ones((5,5),np.uint8)
    return cv2.erode(image, kernel, iterations = 1)

#opening - erosion followed by dilation
```

```
def opening(image):
    kernel = np.ones((5,5),np.uint8)
    return cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)

#canny edge detection
def canny(image):
    return cv2.Canny(image, 100, 200)

#skew correction
def deskew(image):
    coords = np.column_stack(np.where(image > 0))
    angle = cv2.minAreaRect(coords)[-1]
    if angle < -45:
        angle = -(90 + angle)
    else:
        angle = -angle
    (h, w) = image.shape[:2]
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotated = cv2.warpAffine(image, M, (w, h), flags=cv2.INTER_CUBIC,
borderMode=cv2.BORDER_REPLICATE)
    return rotated

#template matching
def match_template(image, template):
    return cv2.matchTemplate(image, template, cv2.TM_CCOEFF_NORMED)

end_time = time.time()
print(f"程序总运行时间: {(end_time - start_time) * 1000:.2f} 毫秒")

recog_end_time = time.time()
print(f"识别用时: {(recog_end_time - recog_start_time) * 1000:.2f} 毫秒")
```

```
image = cv2.imread(im1)

gray = get_grayscale(image)
thresh = thresholding(gray)
opening = opening(gray)
canny = canny(gray)
```

```
# this is only for google colab, the original is cv2.imshow()
from google.colab.patches import cv2_imshow
```

results of OCR with preprocessing

```
img = gray
cv2_imshow(img)
# Adding custom options
custom_config = r'--oem 3 --psm 6'
# Extract text from an image
pytesseract.image_to_string(img, config=custom_config)
```

```
img = thresh
cv2_imshow(img)
# Adding custom options
custom_config = r'--oem 3 --psm 6'
pytesseract.image_to_string(img, config=custom_config)
```

```
img = opening
cv2_imshow(img)
# Adding custom options
custom_config = r'--oem 3 --psm 6'
pytesseract.image_to_string(img, config=custom_config)
```

```
img = canny
cv2_imshow(img)
# Adding custom options
custom_config = r'--oem 3 --psm 6'
pytesseract.image_to_string(img, config=custom_config)
```

keys

> 获取图像中每个文本块的详细信息，包含文本内容和置信度

```
import cv2
import pytesseract
from pytesseract import Output

import time

start_time = time.time()
recog_start_time = time.time()

img = cv2.imread(im1)

d = pytesseract.image_to_data(img, output_type=Output.DICT)
print(d.keys())

# Iterate over the rows in the dictionary and print the text and confidence
for i in range(len(d['text'])):
  print(f"Text: {d['text'][i]}, Confidence: {d['conf'][i]}")

end_time = time.time()
```

```
print(f"程序总运行时间: {(end_time - start_time) * 1000:.2f} 毫秒")

recog_end_time = time.time()
print(f"识别用时: {(recog_end_time - recog_start_time) * 1000:.2f} 毫秒")
```

> **Important**
>
> The speed of the text detection system：程序总运行时间: 2346.69 毫秒 识别用时: 2346.84 毫秒

Getting boxes around text**

> 使用`image_to_boxes`获取文本块的边界框，并绘制在图像上

```
import cv2
import pytesseract
import time

start_time = time.time()
recog_start_time = time.time()

img = cv2.imread(im1)

h, w, c = img.shape
boxes = pytesseract.image_to_boxes(img)
for b in boxes.splitlines():
    b = b.split(' ')
    img = cv2.rectangle(img, (int(b[1]), h - int(b[2])), (int(b[3]), h - int(b[4])), (0,
255, 0), 2)

cv2_imshow( img)
cv2.waitKey(0)

end_time = time.time()
print(f"程序总运行时间: {(end_time - start_time) * 1000:.2f} 毫秒")

recog_end_time = time.time()
print(f"识别用时: {(recog_end_time - recog_start_time) * 1000:.2f} 毫秒")
```

> **Important**
>
> The speed of the text detection system：程序总运行时间: 1876.40 毫秒 识别用时: 1876.57 毫秒

Getting boxes around text words

> 只识别置信度大于60的文本块，并绘制边界框

```
import time

start_time = time.time()
```

```
recog_start_time = time.time()

img = cv2.imread(im1)

n_boxes = len(d['text'])
for i in range(n_boxes):
    if int(d['conf'][i]) > 60:
        (x, y, w, h) = (d['left'][i], d['top'][i], d['width'][i], d['height'][i])
        img = cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2_imshow(img)
cv2.waitKey(0)

end_time = time.time()
print(f"程序总运行时间: {(end_time - start_time) * 1000:.2f} 毫秒")

recog_end_time = time.time()
print(f"识别用时: {(recog_end_time - recog_start_time) * 1000:.2f} 毫秒")
```

> **Important**
>
> The speed of the text detection system：程序总运行时间: 244.72 毫秒 识别用时: 245.77 毫秒

Custom detection, only numbers

> 使用自定义配置，只识别数字

```
import time

start_time = time.time()
recog_start_time = time.time()

img = cv2.imread(im1)

custom_config = r'--oem 3 --psm 6 outputbase digits'
print(pytesseract.image_to_string(img, config=custom_config))

end_time = time.time()
print(f"程序总运行时间: {(end_time - start_time) * 1000:.2f} 毫秒")

recog_end_time = time.time()
print(f"识别用时: {(recog_end_time - recog_start_time) * 1000:.2f} 毫秒")
```

> **Important**
>
> The speed of the text detection system：程序总运行时间: 1250.81 毫秒 识别用时: 1250.94 毫秒

Blacklisting characters

> 排除数字字符，仅识别其他字符

```
custom_config = r'-c tessedit_char_blacklist=0123456789 --psm 6'
pytesseract.image_to_string(img, config=custom_config)
```

Script, run in loop for read pages

> 定义了`read_text_from_image`函数，用于对图像进行预处理和OCR，并将结果写入文件

```
import time

start_time = time.time()
recog_start_time = time.time()

img = cv2.imread(im1)

def read_text_from_image(image):
  """Reads text from an image file and outputs found text to text file"""
  # Convert the image to grayscale
  gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

  # Perform OTSU Threshold
  ret, thresh = cv2.threshold(gray_image, 0, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV)

  rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (18, 18))

  dilation = cv2.dilate(thresh, rect_kernel, iterations = 1)

  contours, hierachy = cv2.findContours(dilation, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

  image_copy = image.copy()

  for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)

    cropped = image_copy[y : y + h, x : x + w]

    file = open("results.txt", "a")

    text = pytesseract.image_to_string(cropped)

    file.write(text)
    file.write("\n")

  file.close()

end_time = time.time()
print(f"程序总运行时间: {(end_time - start_time) * 1000:.2f} 毫秒")
```

```
recog_end_time = time.time()
print(f"识别用时: {(recog_end_time - recog_start_time) * 1000:.2f} 毫秒")
```

> **Important**
>
> The speed of the text detection system：程序总运行时间: 11.22 毫秒 识别用时: 11.42 毫秒

```
import time

start_time = time.time()
recog_start_time = time.time()

# Print out the text identified from the image
image = cv2.imread(im1)
read_text_from_image(image)

# OCR results
cv2_imshow(image)
f = open("results.txt", "r")
lines = f.readlines()
lines.reverse()
for line in lines:
    print(line)
f.close()

end_time = time.time()
print(f"程序总运行时间: {(end_time - start_time) * 1000:.2f} 毫秒")

recog_end_time = time.time()
print(f"识别用时: {(recog_end_time - recog_start_time) * 1000:.2f} 毫秒")
```

> **Important**
>
> The speed of the text detection system：程序总运行时间: 3271.67 毫秒 识别用时: 3271.83 毫秒

Evaluate the accuracy of the detected text

> 评估检测到的文本的准确性，可以通过比较OCR识别结果与已知的真实文本（即地面真实值）进行对比

```
import cv2
import pytesseract
from sklearn.metrics import accuracy_score
import time

start_time = time.time()
recog_start_time = time.time()

image = cv2.imread(im1)

ocr_result = pytesseract.image_to_string(image)
```

```
ground_truth = "Lingnan University Hongkong Camera Obscura Basic principle known to Mozi
(470-390 BC),Aristotle (384-322 BC) Drawing aid for artists:described by Leonardo da Vinci
(1452-1519) Source A.Efros"

# Computational accuracy
def calculate_accuracy(predicted, actual):
    predicted_chars = list(predicted)
    actual_chars = list(actual)

    # Counts the number of matching characters
    matches = sum(p == a for p, a in zip(predicted_chars, actual_chars))

    # Computational accuracy
    accuracy = matches / len(actual_chars)
    return accuracy

# Acquisition accuracy
accuracy = calculate_accuracy(ocr_result.strip(), ground_truth.strip())
print(f"OCR识别结果: {ocr_result}")
print(f"真实文本: {ground_truth}")
print(f"准确率: {accuracy:.2%}")

end_time = time.time()
print(f"程序总运行时间: {(end_time - start_time) * 1000:.2f} 毫秒")

recog_end_time = time.time()
print(f"识别用时: {(recog_end_time - recog_start_time) * 1000:.2f} 毫秒")
```

> **Important**
>
> Program run result:
>
> OCR识别结果: @ Lingnansi ie) University warenatong Camera Obscura fis dl Cheri te ce sf I5 48, + Basic principle known to Mozi (470-390 BC), Aristotle (884-322 BC) + Drawing aid for artists: described by Leonardo da Vinci (1452-1519) 'Source: A. Efros
>
> 真实文本: Lingnan University Hongkong Camera Obscura Basic principle known to Mozi (470-390 BC),Aristotle (384-322 BC) Drawing aid for artists:described by Leonardo da Vinci (1452-1519) Source A.Efros
>
> 准确率: 4.21%
>
> 程序总运行时间: 1043.55 毫秒
>
> 识别用时: 1043.63 毫秒

Problem analysis

1. **identification error**:
    - There are many character errors in the result, such as "Lingnansi" for "Lingnan", (384-322 BC) for , (884-322 BC)etc

- The presence of special characters and misspellings indicates that OCR is experiencing interference when processing text, possibly due to image quality font or background effects

2. **Low accuracy**:
    - The accuracy of only 4.21% indicates that the recognition effect is very poor. It may be caused by blurred image, low contrast, complex background and other factors.

System pros and cons discussion

Advantage

1. **Efficient text recognition**:
    - Using the combination of Tesseract and OpenCV, text can be extracted from images quickly, especially when processing clear documents with high accuracy.
2. **Flexible pre-processing options** :
    - The system provides a variety of image preprocessing functions, such as denoising binarization and edge detection, which can be adjusted according to the characteristics of different images to optimize the recognition effect
3. **Open source and easy to integrate**:
    - Both Tesseract and OpenCV are open source tools that are easy to integrate with other Python projects and have extensive community support and documentation

Disadvantage

1. **Sensitive to image quality**:
    - The system requires high quality of the input image, and blurring the image with low contrast or complex background will cause the recognition effect to decrease significantly.
2. **Character recognition limitation**:
    - For some special fonts, handwritten text, or documents with complex formats, the recognition accuracy may be insufficient, affecting practicality.

Improvement suggestion

1. **Image quality improvement**:
    - Ensure the clarity and contrast of the input image, using high-quality scanning or shooting equipment.
2. **Optimized preprocessing step**:
    - Adaptive thresholds and advanced denoising algorithms are used to ensure that images have been sufficiently processed before being passed into OCR to test different pre-processing combinations and compare their effects on recognition.
3. **Adjust OCR parameters**:
    - Fine-tune Tesseract's configuration parameters, such as Engine Mode (OEM) and Page Split Mode (PSM), to suit specific types of text.

4. **Introducing deep learning models**:
    - Consider combining deep learning models such as convolutional neural networks with traditional OCR to improve the recognition of complex text, especially when dealing with handwritten text and special fonts.
5. **Post-processing procedure**:
    - Introduce spell checking and semantic analysis to automatically correct possible spelling errors to improve the accuracy of the final output.