Báo cáo bài tập lớn môn Xử lý ảnh

Nhận diện chữ monospace trong ảnh chụp

Nhóm 18* - lớp INT3404 1

Học kỳ 1, 2020

Thành viên trong nhóm:

- Nguyễn Cẩm Tú - 18020058
- Hoàng Minh Đức Anh - 18020003

Link repo code: https://github.com/tunc2112/uet-img-processing

Nhóm đăng ký thuyết trình

Mục lục

1 Mở đầu

Trong những năm gần đây, với sự gia tăng ngày càng mạnh số lượng học sinh, sinh viên học lập trình, các hội nhóm, diễn đàn lập trình ngày càng nhiều. Mỗi ngày, có hàng chục, hàng trăm lượt hỏi của người học lập trình trên khắp các hội nhóm, diễn đàn lớn nhỏ, mà phần lớn trong số này là các câu hỏi về gỡ lỗi code. Để người khác có thể giúp đỡ, người hỏi cần cung cấp source code và error log (nếu có), tuy nhiên có rất nhiều người hỏi không cung cấp source code dưới dạng text, thay vào đó là hình chụp màn hình soạn thảo code của mình. Hình chụp có thể được chụp bằng chức năng chụp màn hình có sẵn do phần mềm trên máy tính hỗ trợ, thậm chí có thể được chụp qua camera điện thoại hoặc máy ảnh. Do vậy, để giảm thiểu lượng hình ảnh chụp source code cũng như giúp đỡ những người khác không mất công gõ lại code, nhóm chúng tôi lựa chọn đề tài áp dụng một số thư viện OCR hỗ trợ cho ngôn ngữ Python để chuyển đổi ảnh chụp sang dạng text.

Các tool chuyển đổi online (như www.onlineocr.net/) không đảm bảo toàn vẹn nội dung source code, không kể sai sót khi không nhận diện đúng kí tự, kết quả còn làm mất kí tự tab đầu dòng. Mục tiêu của chúng tôi là cố gắng khắc phục các nhược điểm trên, sau quá trình chuyển đổi, đoạn văn bản sẽ có đủ kí tự, nhất là kí tự whitespace nhất có thể.

2 Phương pháp tiếp cận

Khi soạn thảo source code, người ta thường soạn thảo trên nền IDE hoặc một text editor nào đó. Giao diện của một IDE thường có:

- Khu vực soạn thảo
- Project view
- Error view
- Menubar

Các IDE thường có viền lớn ngăn cách giữa các phần (ví dụ: left side và editor side, editor side và error log). Thông thường, khu vực soạn thảo có thể có cả line count. Những dữ liệu dạng text từ những vùng không phải khu vực soạn thảo sẽ làm "nhiễu" thông tin, làm sai lệch dữ liệu ta mong muốn thu được. Như vậy, việc cắt ảnh để chỉ giữ lại khu vực soạn thảo là cần thiết.

2.1 Các bước thực hiện

1. Lọc nhiễu ảnh.
2. Từ 1 ảnh lớn, cắt bỏ những phần không nằm thuộc khu vực soạn thảo (menubar, terminal panel,...):
    - Dùng (Canny và) Hough Line Transform để xác định các đường thẳng là đường viền giữa các phần.
    - Từ những đường thẳng thu được, thực hiện cắt ảnh. Ta cần phải lựa chọn đường thẳng nào áp dụng vào việc cắt để tránh mất mát thông tin.
3. Áp dụng thư viện opencv và 1 thư viện OCR, nhận dạng chữ và trích xuất dưới dạng text.
4. So sánh độ chính xác của đoạn text kết quả so với output mẫu.

2.2 Yêu cầu hệ thống tối thiểu

- python3.6+
- Thư viện: opencv, numpy, Pillow, scipy.
- pytesseract
- pyocr
- tesserocr

3 thư viện OCR trên đây không chỉ dùng thư viện opencv để đọc ảnh mà còn dùng thư viện Pillow.

3 Thực hiện chi tiết

3.1 Thu thập dữ liệu

Tập dữ liệu ban đầu là ảnh chụp màn hình các loại, bao gồm 122 ảnh đặt trong thư mục img_src, bao gồm:

- 109 ảnh chụp qua chức năng PrintScreen (gọi là ảnh loại (1)), mã số ảnh từ 001 đến 114.
- 13 ảnh chụp qua máy ảnh ngoài (gọi là ảnh loại (2)), mã số ảnh từ 201 đến 213.

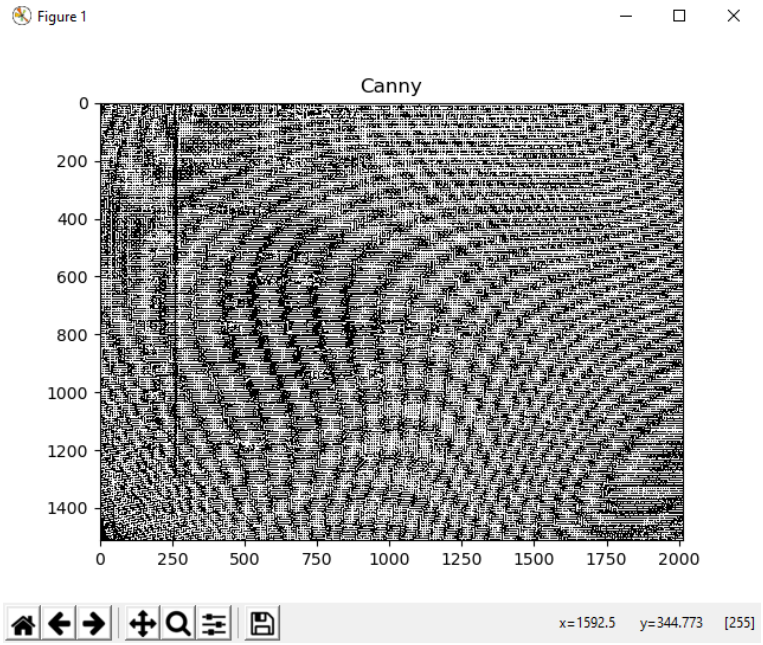
Định dạng tên của file ảnh là "src<mã số ảnh>.(png|jpg|jpeg)".

Từ dữ liệu được lấy ra từ các ảnh, tạo output mẫu là các file text được đặt trong thư mục expected_output, định dạng tên file text là "expected<mã số ảnh>.txt". Output được sinh ra được đặt trong thư mục output và có định dạng tên là "output_<tên thư viện>_<mã số ảnh>.txt".

3.2 Lọc nhiễu ảnh và cắt ảnh

Ta sử dụng Gaussian Filter để lọc nhiễu ảnh. Bên cạnh hàm GaussianBlur của opencv, thư viện scipy cũng hỗ trợ Gaussian Filter qua hàm scipy.ndimage.gaussian_filter.

Mỗi loại ảnh cần xử lý theo cách khác nhau, vì chúng đến từ các nguồn khác nhau. Với ảnh loại (1), ta không cần lọc nhiễu vì chất lượng hình rất tốt, từng pixel rõ ràng, nếu vẫn thực hiện lọc nhiễu thì Canny sẽ không phát hiện được cạnh nữa. Tuy nhiên, với ảnh loại (2), nếu không lọc nhiễu, Canny sẽ lọc cạnh dựa trên những điểm nhiễu:



Hình 1: Lỗi khi áp dụng Canny nhưng không lọc nhiễu, áp dụng với ảnh src201.jpeg

Bên cạnh đó, mỗi loại hình ảnh cần các giá trị min_threshold và max_threshold khác nhau khi áp dụng Canny:

- Ảnh loại (1): min_threshold = 50, max_threshold = 200
- Ảnh loại (2): min_threshold = 10, max_threshold = 15

3.3 Source code sử dụng các thư viện OCR

3.3.1 Sử dụng thư viện pytesseract

```
def ocr_pytesseract(img_id):
    img_filename = './' + get_image_filename(img_id)
    img = cv2.imread(img_filename, cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = cv2.bilateralFilter(gray, 11, 17, 17)

    original = pytesseract.image_to_string(gray, config='')
    write_output(img, original)
```

3.3.2 Sử dụng thư viện pyocr

```
def ocr_pyocr(img_id):
    img_filename = './' + get_image_filename(img_id)
    img = cv2.imread(img_filename, cv2.IMREAD_COLOR)
    tools = pyocr.get_available_tools()
    if len(tools) == 0:
        print("No OCR tool found")
        sys.exit(1)

    tool = tools[0]
    langs = tool.get_available_languages()
    lang = langs[0]
    txt = tool.image_to_string(
        Image.open(img_filename),
        lang=lang,
        builder=pyocr.tesseract.builders.TextBuilder()
    )
    write_output(img_id, txt)
```

3.3.3 Sử dụng thư viện tesserocr

```
def ocr_tesseract(img_id):
    img_filename = './' + get_image_filename(img_id)
    txt = tesserocr.image_to_text(Image.open(img_filename))
    write_output(img_id, txt)
```

3.3.4 So sánh mức độ trùng nhau giữa 2 đoạn text

Sử dụng thư viện difflib được hỗ trợ sẵn bởi ngôn ngữ Python:

```
    file1_content = file1.read()
    file2_content = file2.read()

    s = difflib.SequenceMatcher(None, file1_content, file2_content)
    return s.ratio()



def summary(max_img_id=214):
    overall = {"pytesseract": [], "pyocr": [], "tesserocr": []}
    for img_id in [1, 3, 4, 5, 6, 7]:
        if get_image_filename(img_id) is not None:
            for libname in ["pytesseract", "pyocr", "tesserocr"]:
                ratio = compare_output(libname, img_id)
```

3.3.5 Thu thập kết quả chạy của source code OCR với từng test case

```
def write_output(libname, img_id, text):
    filename = "./output/output_{1}_{0:0>3}.txt".format(img_id, libname)
    with open(filename, "w") as f:
        f.write(text)

def summary(max_img_id):
    overall = {"pytesseract": [], "pyocr": [], "tesserocr": []}
    for img_id in range(1, max_img_id+1):
```

```
        if get_image_filename(img_id) is not None:
            for libname in ["pytesseract", "pyocr", "tesserocr"]:
                ratio = compare_output(libname, img_id)
                print(img_id, libname, round(ratio*100, 2))
                overall[libname].append(ratio)

print("OVERALL")
for libname in ["pytesseract", "pyocr", "tesserocr"]:
    s = sum(overall[libname])
    n = len(overall[libname])
    print(libname, round(100*s/n, 2))
```

4 Đánh giá kết quả

Bảng dưới đây ghi lại mức độ giống nhau của kết quả thu được sau quá trình OCR với output mẫu khi không cắt ảnh và chỉ thực hiện lọc nhiễu ảnh của 1 số mẫu.

	Thư viện		
Mã số	**pytesseract**	**pyocr**	**tesserocr**
001	65.33%	65.44%	65.39%
003	76.14%	75.35%	75.28%
004	45.26%	45.45%	45.66%
005	75.24%	77.67%	74.53%
006	31.01%	29.79%	43.09%
007	74.14%	71.55%	71.25%
Tổng hợp	61.21%	60.87%	62.63%

(Bảng đầy đủ được ghi ở file báo cáo riêng)

Một số kết luận khác ta có thể rút ra được:

- Trên hệ điều hành Windows, thư viện pytesseract chạy rất chậm trên ảnh jpeg.
- Ảnh dark theme chạy ít chính xác hơn ảnh light theme.
- 1 số ảnh không detect được: 90, 97, 104, 105, 106, 107, 108.

5 Kết luận

Trong số 3 thư viện trên, ta có thể sử dụng thư viện tesserocr vì độ chính xác tốt nhất, mặc dù độ chính xác chưa thực sự cao. Ta chưa xét đến yếu tố thời gian, vì thời gian chạy của mỗi thư viện phụ thuộc vào hệ điều hành và định dạng ảnh đưa vào.

Công việc tốn thời gian nhất là cắt ảnh. Hough Transform phát hiện đường thẳng khá tốt, nhưng lựa chọn dựa vào đường thẳng nào để cắt là một bài toán rất khó. Nếu ta cắt ảnh vừa đúng, kết quả của quá trình OCR sẽ chính xác hơn.

Toàn bộ font chữ của các ảnh đầu vào đều là monospace, tập kí tự khác nhau khá giới hạn, các kí tự có độ rộng bằng nhau. Tuy nhiên trái với suy nghĩ ban đầu của tác giả, bài toán nhận dạng đã trở nên khó. Trong các yếu tố ảnh hưởng đến kết quả của quá trình OCR, độ tương phản, kích thước và chất lượng của ảnh, độ lớn của chữ là 4 yếu tố đầu tiên. Tập ảnh thu thập từ các nguồn trong thực tế, không đảm bảo tốt 4 yếu tố trên nên kết quả có sự sai lệch nhiều.

Project này tiếp cận bài toán theo xử lý ảnh, chỉ sử dụng các kỹ thuật cơ bản của xử lý ảnh và chưa sử dụng các công nghệ cao cấp hơn. Thay vì giải quyết một cách bán tự động như code trong project này, một giải pháp tự động hơn sẽ làm tăng hiệu quả giải quyết bài toán. Ví dụ:

- Học cách nhận diện đường cần cắt theo màu sắc và độ dày.
- Dựa vào độ rộng của 1 số kí tự, tính khoảng cách từ đầu dòng đến lề để tính số dấu cách đầu dòng cần thiết.

Trong tương lai, nếu áp dụng Deep Learning vào OCR, chất lượng của các OCR engine sẽ được cải thiện mà có thể bỏ qua các yếu tố về ảnh (độ tương phản, kích thước, chất lượng của ảnh và kích thước của chữ).

6 Tài liệu tham khảo

- *Awesome OCR*. https://github.com/kba/awesome-ocr#python
- AnandhJagadeesan (2020, February 19). *Text Detection and Extraction using OpenCV and OCR*. GeeksforGeeks. https://www.geeksforgeeks.org/text-detection-and-extraction-using-opencv-and-ocr

- Tim Chin (2019, January 23). *Using Image Processing to Detect Text*. Medium. https://medium.com/@theclassytim/using-image-processing-to-detect-text-8be34c677c11
- *Image denoising by FFT*. http://scipy-lectures.org/intro/scipy/auto_examples/solutions/plot_fft_image_denoise.html
- *7 Tips to Improve OCR Accuracy (Why is my OCR so poor?)*. YouTube. https://www.youtube.com/watch?v=KS1gd5yUmKo