

XỬ LÝ TỆP PDF ĐƠN GIẢN VỚI PYPDF

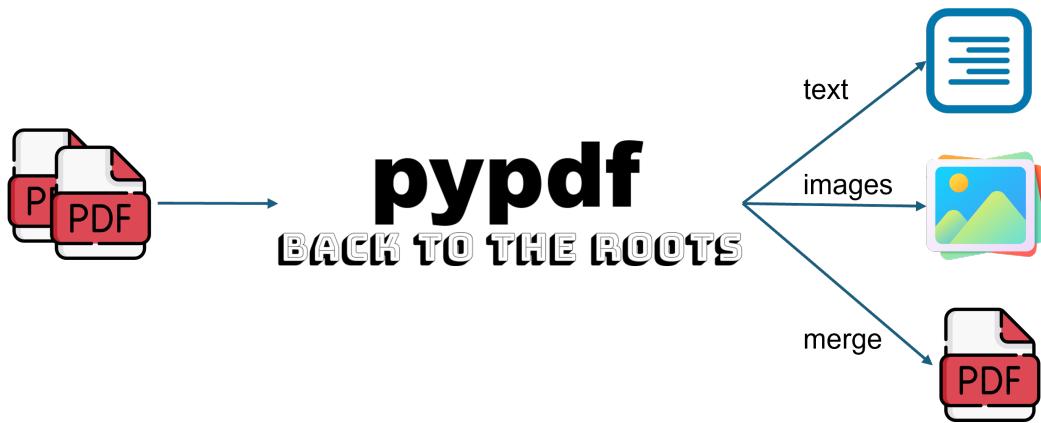
Dinh-Tiem Nguyen và Quang-Vinh Dinh

1 Mở đầu

Làm sao để trích xuất nội dung văn bản và hình ảnh trong file pdf? Làm thế nào để ghép nhiều file pdf thành một file duy nhất? Trong bài viết này, chúng ta sẽ trả lời những câu hỏi đó bằng cách sử dụng thư viện pypdf-một thư viện xử lý file pdf hiệu quả.

Yêu cầu:

- Máy tính đã cài đặt Python ≥ 3.7
- Biết lập trình Python cơ bản.



2 Hướng dẫn cài đặt và sử dụng pypdf

Để cài đặt pypdf ta sử dụng lệnh sau:

```
1 pip install pypdf
```

2.1 Trích xuất văn bản từ pdf

Pypdf cho phép ta trích xuất nội dung văn bản từ một hoặc nhiều trang trong tập tin PDF. Điều này rất hữu ích khi ta cần trích xuất thông tin hoặc dữ liệu văn bản từ tài liệu PDF để sử dụng trong các ứng dụng khác.

Ví dụ chương trình dưới đây, chúng ta khai báo sử dụng thư viện pypdf và sử dụng PdfReader để đọc file `yolov9.pdf` sau đó lưu kết quả vào biến reader. Tiếp theo ta sử dụng reader.pages, phương thức này trả về một danh sách các trang trong tập tin PDF đã được đọc bằng PdfReader. Mỗi phần tử trong danh sách này đại diện cho một trang trong tập tin PDF và có thể được truy cập bằng cách sử dụng chỉ mục hoặc vòng lặp. Ví dụ, để truy cập trang thứ hai trong tập tin PDF, ta có thể sử dụng `reader.pages[1]`, vì chỉ mục trong Python bắt đầu từ 0. Điều này sẽ trả về một đối tượng đại diện cho trang thứ hai trong tài liệu PDF. Để truy cập vào nội dung của một trang cụ thể, ta có thể sử dụng phương thức `extract_text()`, như trong ví dụ.

```

1 from pypdf import PdfReader
2
3 # Đọc file PDF
4 reader = PdfReader("yolov9.pdf")
5
6 # Lấy số trang
7 num_pages = len(reader.pages)
8
9 # Lấy nội dung trang đầu tiên
10 page_1 = reader.pages[0]
11 page_1_txt = page_1.extract_text()
12
13 # Lấy nội dung toàn bộ các trang
14 pages_txt = ""
15 for i in range(num_pages):
16     page = reader.pages[i]
17     pages_txt += page.extract_text() + "\n"

```

Abstract

Today's deep learning methods focus on how to design the most appropriate objective functions so that the prediction results of the model can be closest to the ground truth. Meanwhile, an appropriate architecture that can facilitate acquisition of enough information for prediction has to be designed. Existing methods ignore a fact that when input data undergoes layer-by-layer feature extraction and spatial transformation, large amount of information will be lost. This paper will delve into the important issues of data loss when data is transmitted through deep networks, namely information bottleneck and reversible functions. We proposed the concept of programmable gradient information (PGI) to cope with the various changes required by deep networks to achieve multiple objectives. PGI can provide complete input information for the target task to calculate objective function, so that reliable gradient information can be obtained to update network weights. In addition, a new lightweight network architecture - Generalized Efficient Layer Aggregation Network (GELAN), based on gradient path planning is designed. GELAN's architecture confirms that PGI has gained superior results on lightweight models. We verified the proposed GELAN and PGI on MS COCO dataset based object detection. The results show that GELAN only uses conventional convolution operators to achieve better parameter utilization than the state-of-the-art methods developed based on depth-wise convolution. PGI can be used for variety of models from lightweight to large. It can be used to obtain complete information, so that train-from-scratch models can achieve better results than state-of-the-art models pre-trained using large datasets, the comparison results are shown in Figure 1. The source codes are at: <https://github.com/WongKinYiu/yolov9>.


 PYPDF →
Abstract

Today's deep learning methods focus on how to design the most appropriate objective functions so that the prediction results of the model can be closest to the ground truth. Meanwhile, an appropriate architecture that can facilitate acquisition of enough information for prediction has to be designed. Existing methods ignore a fact that when input data undergoes layer-by-layer feature extraction and spatial transformation, large amount of information will be lost. This paper will delve into the important issues of data loss when data is transmitted through deep networks, namely information bottleneck and reversible functions. We proposed the concept of programmable gradient information (PGI) to cope with the various changes required by deep networks to achieve multiple objectives. PGI can provide complete input information for the target task to calculate objective function, so that reliable gradient information can be obtained to update network weights. In addition, a new lightweight network architecture - Generalized Efficient Layer Aggregation Network (GELAN), based on gradient path planning is designed. GELAN's architecture confirms that PGI has gained superior results on lightweight models. We verified the proposed GELAN and PGI on MS COCO dataset based object detection. The results show that GELAN only uses conventional convolution operators to achieve better parameter utilization than the state-of-the-art methods developed based on depth-wise convolution. PGI can be used for variety of models from lightweight to large. It can be used to obtain complete information, so that train-from-scratch models can achieve better results than state-of-the-art models pre-trained using large datasets, the comparison results are shown in Figure 1. The source codes are at: <https://github.com/WongKinYiu/yolov9> .

Hình 1: Trích xuất văn bản từ file PDF

2.2 Trích xuất hình ảnh từ pdf

Trích xuất hình ảnh từ các tập tin PDF là quá trình lấy các hình ảnh từ các trang trong tài liệu PDF và lưu chúng thành các tập tin hình ảnh độc lập, chẳng hạn như JPEG hoặc PNG. Việc này thường được thực hiện để xử lý và sử dụng hình ảnh từ tài liệu PDF trong các ứng dụng khác nhau, như xây dựng bộ sưu tập hình ảnh, phân tích hình ảnh...

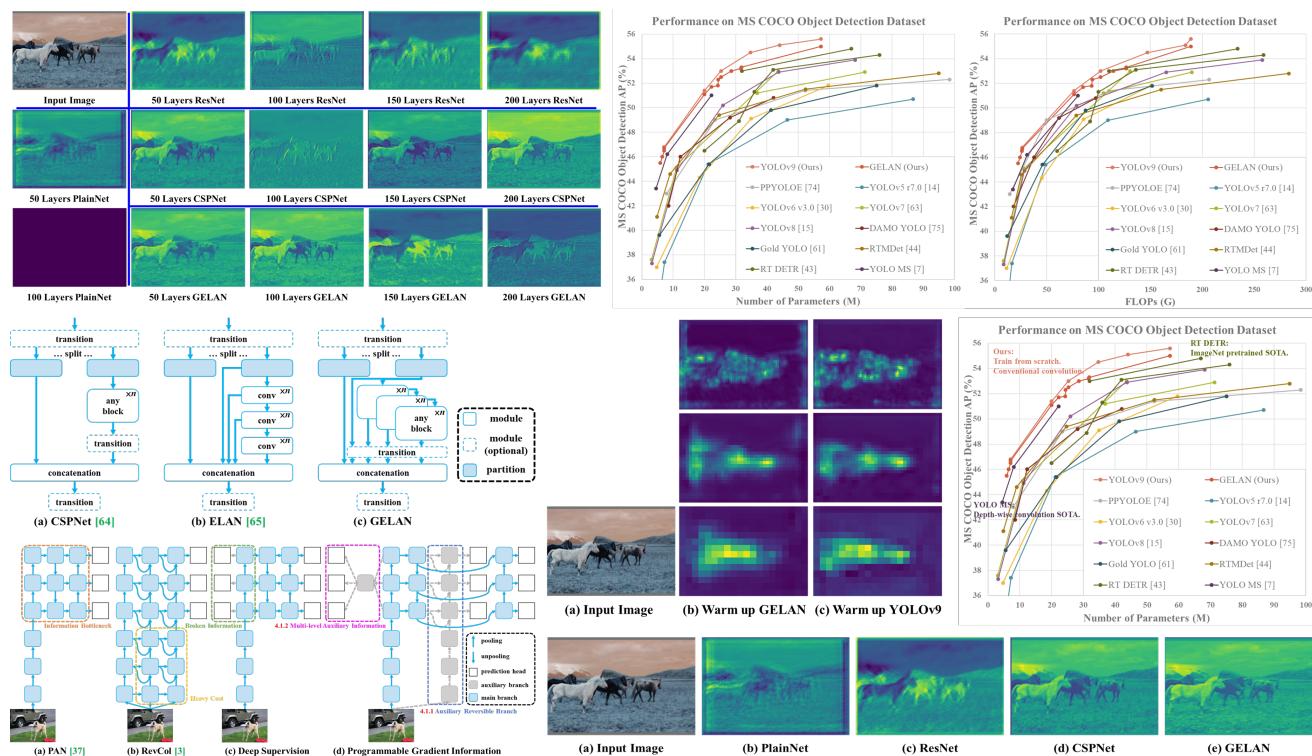
Trong pypdf cung cấp phương thức `page.images` để hỗ trợ trích xuất các hình ảnh trong tập tin pdf. Ta cùng xem ví dụ sau:

```

1 # extract images
2 from pypdf import PdfReader
3
4 reader = PdfReader("yolov9.pdf")
5 count = 0
6 for page in reader.pages:
7     for image_file_object in page.images:
8         with open(str(count) + image_file_object.name, "wb") as fp:
9             fp.write(image_file_object.data)
10            count += 1

```

Trong ví dụ trên, chúng ta đọc và sử dụng vòng lặp để duyệt qua từng trang trong file pdf, với mỗi trang chúng ta sẽ trích xuất các hình ảnh thông qua phương thức `page.images`, phương thức này sẽ trả về danh sách các hình ảnh mà nó trích xuất được từ 1 trang, sau đó chúng ta lưu lại file hình ảnh với tên hình ảnh là số thứ tự + tên hình ảnh trong file pdf ta lấy được qua phương thức `image_file_object.name`



Hình 2: Trích xuất các hình ảnh từ file PDF

2.3 Nối các file pdf

Merge PDF cho phép bạn tổ chức các tài liệu PDF riêng lẻ thành một tài liệu lớn hơn, giúp dễ dàng quản lý và tìm kiếm thông tin. Trong pypdf cung cấp tính năng PdfWriter để thực hiện điều này.

```

1 # Merge PDFs
2 from pypdf import PdfWriter
3
4 merger = PdfWriter()
5 for pdf in ["yolov6.pdf", "yolov7.pdf", "yolov9.pdf"]:
6     merger.append(pdf)
7
8 merger.write("merged-yolov-679.pdf")
9 merger.close()

```

Trong ví dụ trên, ta thực hiện nối ba bài báo yolov6, yolov7, yolov9 lại với nhau, đầu tiên chúng ta sẽ import PdfWriter từ thư viện pypdf, tiếp theo chúng ta tạo đối tượng merger từ PdfWriter(), đối tượng này sẽ được sử dụng để merge các tập tin PDF. Tiếp theo chúng ta duyệt qua từng file pdf và thêm chúng vào đối tượng merger qua phương thức append. Cuối cùng ta ghi dữ liệu từ đối tượng merger vào tệp tin mới có tên là merged-yolov-679.pdf và phương thức close() được gọi để đóng tập tin PDF đã được merge.

2.4 Nén file pdf

Chức năng nén file trong thư viện pypdf cho phép bạn giảm kích thước của các tập tin PDF bằng cách nén lại các luồng nội dung của mỗi trang. Việc này không chỉ giúp tiết kiệm không gian lưu trữ mà còn làm tăng tốc độ tải xuống và chia sẻ tập tin PDF.

```

1 from pypdf import PdfWriter
2
3 writer = PdfWriter(clone_from="yolov9.pdf")
4
5 for page in writer.pages:
6     page.compress_content_streams(level=8) # This is CPU intensive!
7
8 with open("out.pdf", "wb") as f:
9     writer.write(f)

```

Trong ví dụ trên, ta tạo một bản sao của tập tin PDF "yolov9.pdf", nén lại nội dung của mỗi trang trong tập tin PDF này với mức độ nén là 8, và sau đó lưu lại thành một tập tin PDF mới có tên là "out.pdf". Đầu tiên ta tạo một đối tượng PdfWriter mới gọi là writer với số clone_from chỉ định tên tập tin PDF mà chúng ta muốn tạo bản sao. Tiếp theo chúng ta lặp qua từng trang của pdf, sau đó sử dụng page.compress_content_streams(level=8) để nén, trong đó level là mức nén có giá trị từ 1 đến 9. Cuối cùng chúng ta lưu lại file có tên out.pdf.

Chúng ta có thể kiểm tra dung lượng file qua đoạn mã dưới đây:

```

1 # get size of file pdf
2 import os
3
4 file_size = os.path.getsize("yolov9.pdf") / (1024 * 1024)
5 print(f"yolo9.pdf size:{file_size}")
6
7 file_out = os.path.getsize("out.pdf") / (1024 * 1024)
8 print(f"out.pdf size:{file_out}")

```

===== Output =====
yolo9.pdf size:4.738467216491699
out.pdf size:4.718204498291016
=====

Có thể thấy dung lượng file out.pdf được nén từ file yolov9.pdf có dung lượng thấp hơn.

3 Kết Luận

Trong bài viết này, chúng ta đã tìm hiểu cách sử dụng thư viện pypdf trong Python để thực hiện hai tác vụ quan trọng là trích xuất văn bản và hình ảnh từ các tập tin PDF, cũng như thực hiện các tính năng nén, merge (gộp) các tập tin PDF lại với nhau. Qua bài viết, hy vọng có thể giúp mọi người biết thêm cách sử dụng công cụ hữu ích pypdf để xử lý dữ liệu pdf.