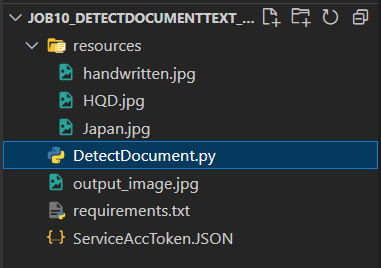
# **CHƯƠNG 7: CẮT HÌNH ẢNH VÀ PHÁT HIỆN WEBSITE**

## 7.1. Phát hiện gợi ý cắt ảnh

### 7.1.1. Mục đích đoạn mã

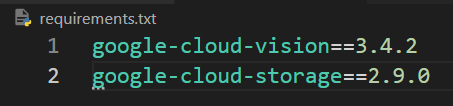
* Phát hiện văn bản là một chức năng phát hiện văn bản dày đặc (bao gồm cả văn bản viết tay) của một hình ảnh bằng cách thực hiện nhận dạng ký tự quang học.

### 7.1.2. Cấu trúc thư mục



### 7.1.3. Cài đặt các gói cần thiết

* Trong thiết bị đầu cuối, hãy điều hướng đến thư mục có chứa mã liên quan
* Chuẩn bị một file requirements.txt có chứa thông tin các gói cài đặt cần thiết như hình bên dưới. Sau đó chạy lệnh **pip install -r requirements.txt** trên terminal.



### 7.1.4. Hướng dẫn chi tiết

#### 7.1.4.1. Nhập thư viện

* **os**: Được sử dụng để thiết lập biến môi trường để xác định tệp JSON chứa thông tin xác thực dự án của bạn.
* **Enum**: Được sử dụng để định nghĩa các loại đặc trưng (feature) của văn bản.
* **google.cloud.vision**: Thư viện của Google Cloud Vision API cho việc xử lý hình ảnh và phân tích văn bản.
* **PIL**: Python Imaging Library, được sử dụng để vẽ hình ảnh và hộp xung quanh các phần của văn bản trích xuất.

#### 7.1.4.2. Xác thực API

| os.environ['GOOGLE\_APPLICATION\_CREDENTIALS'] = 'ServiceAccToken.JSON' |
| --- |

* Đoạn mã này sử dụng một tệp JSON chứa thông tin xác thực để gọi API của Google Cloud Vision.

#### 7.1.4.3. Khởi tạo client của Vision API

| client = vision.ImageAnnotatorClient() |
| --- |

* Đoạn mã này tạo một đối tượng client để tương tác với Google Cloud Vision API.

#### 7.1.4.4. Định nghĩa một Enum FeatureType để biểu thị loại đặc trưng bạn muốn trích xuất (trang, khối, đoạn văn, từ hoặc ký tự).

| class FeatureType(Enum):  PAGE = 1  BLOCK = 2  PARA = 3  WORD = 4  SYMBOL = 5 |
| --- |

#### 7.1.4.5. Định nghĩa hàm draw\_boxes để vẽ viền hình chữ nhật xung quanh đối tượng được trích xuất từ hình ảnh

| def draw\_boxes(image, bounds, color):  draw = ImageDraw.Draw(image)  for bound in bounds:  draw.polygon(  [  bound.vertices[0].x, bound.vertices[0].y,  bound.vertices[1].x, bound.vertices[1].y,  bound.vertices[2].x, bound.vertices[2].y,  bound.vertices[3].x, bound.vertices[3].y,  ],  None,  color,  )  return image |
| --- |

1. **draw\_boxes(image, bounds, color)**: Đây là khai báo của hàm, và nó nhận ba đối số đầu vào: image, bounds, và color. Trong đó:

* **image**: Đối số này là hình ảnh (được biểu diễn bằng đối tượng hình ảnh từ thư viện PIL) dùng để vẽ các hộp.
* **bounds**: Đây là danh sách các hộp cần được vẽ. Mỗi hộp trong danh sách bounds được biểu diễn bằng một đối tượng chứa thông tin về các tọa độ (4 đỉnh) của hộp trong không gian hình ảnh. Mỗi đỉnh được biểu diễn bằng tọa độ x và y.
* **color**: Đối số này là màu sẽ được sử dụng để vẽ các hộp.

1. **draw = ImageDraw.Draw(image)**: Dòng này tạo một đối tượng ImageDraw từ thư viện PIL và gán cho biến draw. Đối tượng ImageDraw này cho phép bạn thực hiện các thao tác vẽ trên hình ảnh.
2. Tiếp theo, hàm sử dụng vòng lặp for để duyệt qua từng hộp trong danh sách bounds:

**for bound in bounds:**: Dòng này duyệt qua mỗi hộp trong danh sách bounds và gán hộp hiện tại cho biến bound.

1. Bên trong vòng lặp, hàm sử dụng **draw.polygon()** để vẽ hộp cho mỗi bound:

**draw.polygon(...)**: Đây là phương thức của đối tượng draw (đã tạo ở bước #2) để vẽ một đa giác (hình bao gồm nhiều đỉnh) trên hình ảnh. Phương thức này nhận hai đối số:

Đối số đầu tiên là một danh sách gồm tọa độ của các đỉnh của đa giác. Trong trường hợp này, danh sách này được tạo bằng cách liệt kê tọa độ của 4 đỉnh của bound:

bound.vertices[0].x, bound.vertices[0].y: Đây là tọa độ (x, y) của đỉnh thứ nhất của hộp.

bound.vertices[1].x, bound.vertices[1].y: Đây là tọa độ (x, y) của đỉnh thứ hai của hộp.

bound.vertices[2].x, bound.vertices[2].y: Đây là tọa độ (x, y) của đỉnh thứ ba của hộp.

bound.vertices[3].x, bound.vertices[3].y: Đây là tọa độ (x, y) của đỉnh thứ tư của hộp.

Đối số thứ hai là **None**, nghĩa là không có fill color cho đa giác, chỉ có đường viền.

Đối số thứ ba là **color**, đó là màu sẽ được sử dụng để vẽ đa giác, và nó được truyền từ đối số của hàm.

Cuối cùng, hàm trả về image sau khi đã vẽ các hộp. Bây giờ, hình ảnh image đã được cập nhật với các hộp xung quanh các đối tượng trích xuất và có thể được hiển thị hoặc lưu trữ bằng cách sử dụng các phương thức của thư viện PIL như .show() hoặc .save().

#### 7.1.4.6. Định nghĩa hàm get\_document\_bounds

| def get\_document\_bounds(image\_file, feature):  bounds = []  with open(image\_file, "rb") as image\_file:  content = image\_file.read()  image = vision.Image(content=content)  response = client.document\_text\_detection(image=image)  document = response.full\_text\_annotation    for page in document.pages:  for block in page.blocks:  print(f'\nBlock confidence: {format(block.confidence)}')  for paragraph in block.paragraphs:  print(f'\nParagraph confidence: {format(paragraph.confidence)}')  for word in paragraph.words:  word\_text = ''.join([  symbol.text for symbol in word.symbols  ])  print(f'\n Word text: {word\_text} (confidence: {format(word.confidence)})')  for symbol in word.symbols:  print(f'\n Symbol: {symbol.text} (confidence: {format(symbol.confidence)})')  if feature == FeatureType.SYMBOL:  bounds.append(symbol.bounding\_box)  if feature == FeatureType.WORD:  bounds.append(word.bounding\_box)  if feature == FeatureType.PARA:  bounds.append(paragraph.bounding\_box)  if feature == FeatureType.BLOCK:  bounds.append(block.bounding\_box)  if feature == FeatureType.PAGE:  bounds.append(page.bounding\_box)  print(bounds)  return bounds |
| --- |

Hàm này dùng để trích xuất thông tin từ hình ảnh và xác định các hộp xung quanh đối tượng văn bản trên hình ảnh.

1. **get\_document\_bounds(image\_file, feature)**: Đây là khai báo cyar hàm và nó nhận hai đối số đầu vào **image\_file** và **feature**. Trong đó:

* **image\_file**: Đây là đường dẫn đến tệp hình ảnh mà bạn muốn trích xuất.
* **feature**: Đây là 1 tham số dạng Enum từ lớp Feature (được định nghĩa ở mục 7.1.4.4), đại diện cho loại đặc trưng (page, block, para, word, symbol) mà bạn muốn trích xuất.

1. **bound = []** : Đây là danh sách sẽ được sử dụng để lưu trữ các hộp xung quanh các đối tượng văn bản trên hình ảnh. Ban đầu, sẽ khởi tạo rỗng
2. **with open(image\_file, “rb”) as image\_file**: Dòng này sẽ được sử dụng để mở tệp hình ảnh (**‘image\_file’**) để đọc dưới dạng dữ liệu nhị phân (“**rb**”). Nội dung được đọc sẽ được gán vào biến **content**.
3. **image = vision.Image(content=content)**: Dòng này tạo một đối tượng hình ảnh (**image**) từ nội dung đã đọc ở bước trên, sử dụng thư viện Google Cloud Vision API. Đối tượng image sẽ được sử dụng để gửi yêu cầu trích xuất thông tin văn bản đến API.
4. **response = client.document\_text\_detection(image=image)**: Gửi yêu cầu trích xuất thông tin văn bản từ hình ảnh bằng cách sử dụng đối tượng hình ảnh đã tạo ở bước trước. Kết quả từ API sẽ được lưu trữ trong biến response.
5. **document = response.full\_text\_annotation**: Trích xuất thông tin văn bản từ kết quả của API và lưu trữ nó trong biến document
6. Duyệt qua thông tin văn bản từ mức trang (**page**) đến mức khối (**block**). Mục tiêu là duyệt qua và trích xuất thông tin từng mức này)

**for page in document.pages:**: Dòng này duyệt qua từng trang (page) trong tài liệu (document). Một hình ảnh có thể có nhiều trang (ví dụ: nếu hình ảnh chứa nhiều trang văn bản).

**for block in page.blocks:**: Bên trong vòng lặp trang, hàm duyệt qua từng khối (block) trên trang hiện tại. Một khối có thể là một phần của trang văn bản, và có thể chứa đoạn văn, từ, và ký tự.

**print(f'\nBlock confidence: {format(block.confidence)}'):** Dòng này in ra mức độ tự tin (confidence) của khối hiện tại. Mức độ tự tin thể hiện độ chắc chắn của Google Cloud Vision API về khả năng nhận diện văn bản trong khối.

1. Duyệt qua thông tin văn bản từ mức đoạn văn (**paragraph**) đến mức ký tự (**symbol**). Trong phần này, hàm tiếp tục xuống từ mức khối (**block**) và duyệt qua từng mức: đoạn văn (**paragraph**), từ (**word**), ký tự (**symbol**). Mục tiêu là duyệt qua và trích xuất thông tin từ từng mức này và kiểm tra nếu mức này tương ứng với **Feature Type** mà bạn quan tâm.

**for paragraph in block.paragraphs::** Dòng này duyệt qua từng đoạn văn (paragraph) trong khối hiện tại. Một đoạn văn có thể chứa nhiều từ và câu.

**print(f'\nParagraph confidence: {format(paragraph.confidence)}')**: Dòng này in ra mức độ tự tin (confidence) của đoạn văn hiện tại.

**for word in paragraph.words:**: Bên trong vòng lặp đoạn văn, hàm duyệt qua từng từ (word) trong đoạn văn. Từ là các đơn vị cơ bản trong văn bản.

**word\_text = ''.join([symbol.text for symbol in word.symbols]):** Dòng này kết hợp các ký tự (symbols) trong từ để tạo ra nội dung của từ (word\_text). Mỗi từ có thể chứa nhiều ký tự.

**print(f'\n Word text: {word\_text} (confidence: {format(word.confidence)})')**:Dòng này in ra nội dung của từ và mức độ tự tin (confidence) của từ hiện tại.

**for symbol in word.symbols**: Bên trong vòng lặp từ, hàm duyệt qua từng ký tự (symbol) trong từ. Ký tự là các đơn vị nhỏ nhất trong văn bản (chữ cái, số, dấu chấm, v.v.).

**print(f'\n Symbol: {symbol.text} (confidence: {format(symbol.confidence)})')**:Dòng này in ra nội dung của ký tự và mức độ tự tin (confidence) của ký tự hiện tại.

Nếu feature (loại đặc trưng) bạn quan tâm là SYMBOL, thì hàm kiểm tra nếu feature bằng FeatureType.SYMBOL, sau đó thêm hộp xung quanh ký tự hiện tại vào danh sách bounds.

Nếu feature là WORD, PARA, hoặc BLOCK, tương tự, hàm kiểm tra loại đặc trưng và thêm hộp xung quanh từ, đoạn văn, hoặc khối hiện tại vào danh sách bounds.

1. Cuối cùng, danh sách **bounds** chứa thông tin về các hộp xung quanh các đối tượng văn bản mà bạn đã quy định (dựa trên feature) và được trả về từ hàm **get\_document\_bounds**.

#### 7.1.4.7. Định nghĩa hàm render\_doc\_text (filein, fileout):

| def render\_doc\_text(filein, fileout:  image = Image.open(filein)  bounds = get\_document\_bounds(filein, FeatureType.BLOCK)  draw\_boxes(image, bounds, "blue")  if fileout != 0:  image.save(fileout)  else:  image.show() |
| --- |

Hàm render\_doc\_text được sử dụng để hiển thị hoặc lưu trữ hình ảnh gốc sau khi đã vẽ các hộp xung quanh các đối tượng văn bản trên đó

1. **image = Image.open(filein)**: Dòng này mở hình ảnh gốc từ đường dẫn tệp hình ảnh filein bằng cách sử dụng thư viện PIL (Pillow). Kết quả là một đối tượng hình ảnh (image) chứa hình ảnh từ tệp đã mở.
2. **bounds = get\_document\_bounds(filein, FeatureType.BLOCK)**: Dòng này gọi hàm **get\_document\_bounds** để trích xuất các hộp xung quanh các đối tượng văn bản từ hình ảnh (filein). Trong trường hợp này, bạn quy định **FeatureType.BLOCK** như đặc trưng mà bạn muốn trích xuất. Kết quả là danh sách bounds chứa thông tin về các hộp xung quanh các khối văn bản trên hình ảnh.
3. **draw\_boxes(image, bounds, "blue"):** Dòng này gọi hàm **draw\_boxes** để vẽ các hộp xung quanh các đối tượng văn bản lên hình ảnh image. Hộp sẽ được vẽ bằng màu xanh (**blue**).
4. **if fileout != 0:**: Dòng này kiểm tra xem biến fileout có khác 0 hay không. Nếu bạn đã chỉ định một tệp đầu ra (fileout khác 0), nghĩa là bạn muốn lưu hình ảnh đã được vẽ, tiếp tục vào phần if:
5. **image.save(fileout)**: Dòng này lưu hình ảnh đã được vẽ (với các hộp xung quanh đối tượng văn bản) vào tệp được chỉ định bởi biến fileout.
6. Nếu bạn không chỉ định tệp đầu ra (fileout là 0 hoặc không được đặt), bạn sẽ vào phần else.
7. **image.show()**: Trong phần else, nếu bạn không chỉ định tệp đầu ra, hình ảnh đã được vẽ sẽ được hiển thị trực tiếp lên màn hình. Điều này cho phép bạn xem hình ảnh với các hộp xung quanh đối tượng văn bản mà không cần lưu thành tệp.

#### 7.1.4.8. Sử dụng hàm

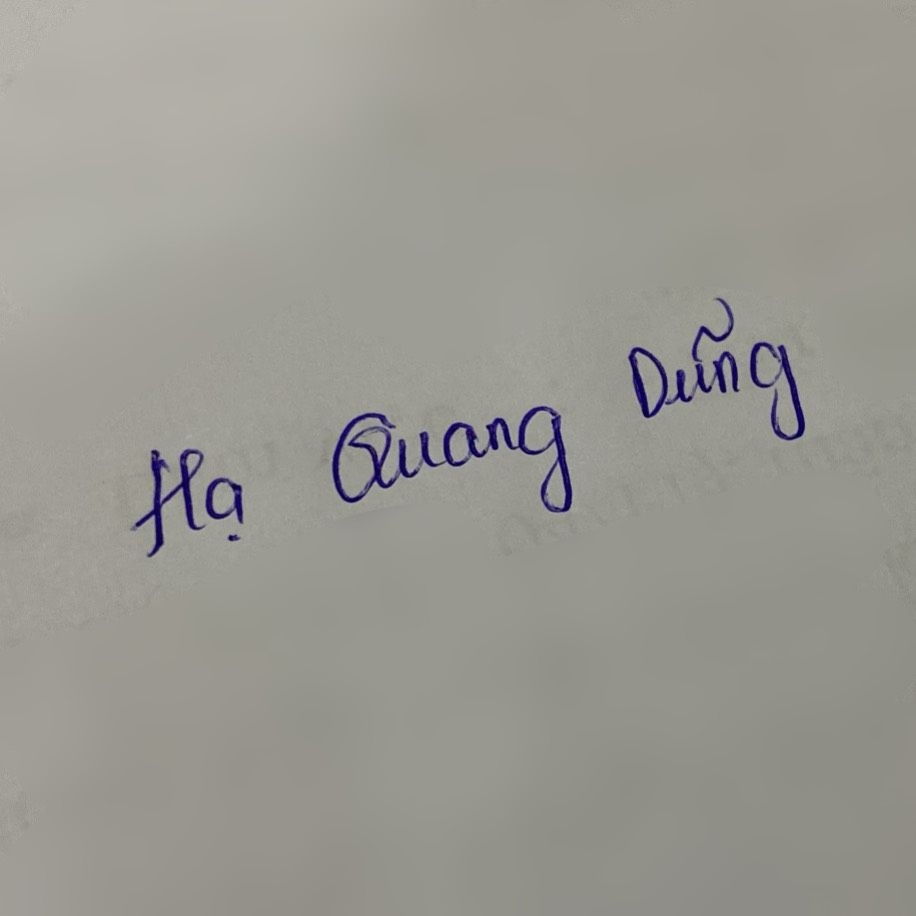
| img\_path = './resources/HQD.jpg'  # render\_doc\_text(img\_path, 'output\_image.jpg') |
| --- |

### 7.1.5. Chi tiết mã nguồn

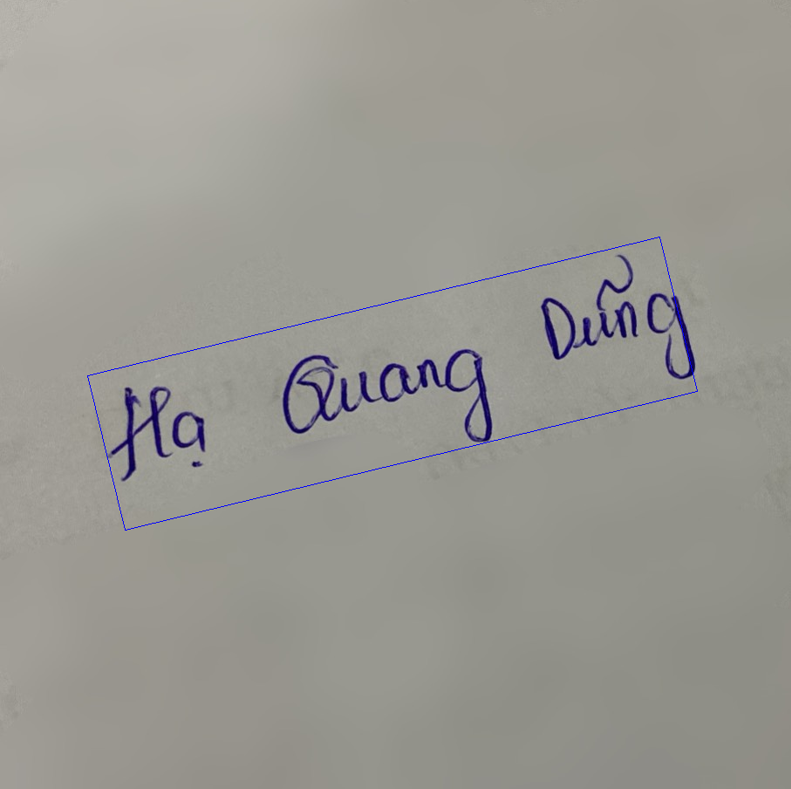
| import os  from enum import Enum  from google.cloud import vision  from PIL import Image, ImageDraw  os.environ['GOOGLE\_APPLICATION\_CREDENTIALS'] = 'ServiceAccToken.JSON'  client = vision.ImageAnnotatorClient()  class FeatureType(Enum):  PAGE = 1  BLOCK = 2  PARA = 3  WORD = 4  SYMBOL = 5  def draw\_boxes(image, bounds, color):  draw = ImageDraw.Draw(image)  for bound in bounds:  draw.polygon(  [  bound.vertices[0].x, bound.vertices[0].y,  bound.vertices[1].x, bound.vertices[1].y,  bound.vertices[2].x, bound.vertices[2].y,  bound.vertices[3].x, bound.vertices[3].y,  ],  None,  color,  )  return image  def get\_document\_bounds(image\_file, feature):  bounds = []  with open(image\_file, "rb") as image\_file:  content = image\_file.read()  image = vision.Image(content=content)  response = client.document\_text\_detection(image=image)  document = response.full\_text\_annotation    for page in document.pages:  for block in page.blocks:  print(f'\nBlock confidence: {format(block.confidence)}')  for paragraph in block.paragraphs:  print(f'\nParagraph confidence: {format(paragraph.confidence)}')  for word in paragraph.words:  word\_text = ''.join([  symbol.text for symbol in word.symbols  ])  print(f'\n Word text: {word\_text} (confidence: {format(word.confidence)})')  for symbol in word.symbols:  print(f'\n Symbol: {symbol.text} (confidence: {format(symbol.confidence)})')  if feature == FeatureType.SYMBOL:  bounds.append(symbol.bounding\_box)  if feature == FeatureType.WORD:  bounds.append(word.bounding\_box)  if feature == FeatureType.PARA:  bounds.append(paragraph.bounding\_box)  if feature == FeatureType.BLOCK:  bounds.append(block.bounding\_box)  if feature == FeatureType.PAGE:  bounds.append(page.bounding\_box)  print(bounds)  return bounds  def render\_doc\_text(filein, fileout):  image = Image.open(filein)  bounds = get\_document\_bounds(filein, FeatureType.BLOCK)  draw\_boxes(image, bounds, "blue")  if fileout != 0:  image.save(fileout)  else:  image.show()  img\_path = './resources/HQD.jpg'  render\_doc\_text(img\_path, 'output\_image.jpg') |
| --- |

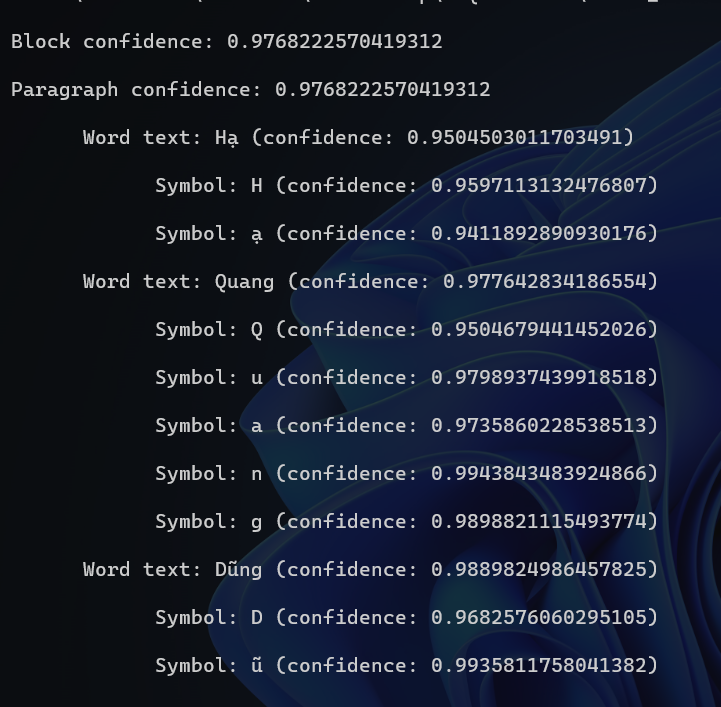
### 7.1.7. Chạy chương trình

Hình ảnh đầu vào



Kết quả nhận được



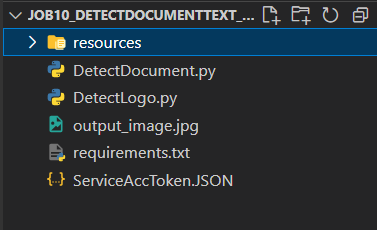


## 7.2. Phát hiện Web

### 7.2.1. Mục đích đoạn mã

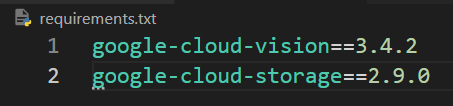
Phát hiện logo là một tính năng phát hiện logo sản phẩm phổ biến trong một hình ảnh.

### 7.2.2. Cấu trúc thư mục



### 7.2.3. Cài đặt các gói cần thiết

* Trong thiết bị đầu cuối, hãy điều hướng đến thư mục có chứa mã liên quan
* Chuẩn bị một file requirements.txt có chứa thông tin các gói cài đặt cần thiết như hình bên dưới. Sau đó chạy lệnh **pip install -r requirements.txt** trên terminal.



### 7.2.4. Hướng dẫn chi tiết

#### 7.2.4.1. Nhập thư viện

* **os** : Thư viện này được sử dụng để làm việc với hệ thống tệp và thư mục.
* **google.cloud.vision**: Thư viện này chứa các công cụ cho việc gửi hình ảnh đến Google Cloud Vision API và nhận kết quả.

#### 7.2.4.2. Xác thực API

| os.environ['GOOGLE\_APPLICATION\_CREDENTIALS'] = 'ServiceAccToken.JSON' |
| --- |

* Đoạn mã này sử dụng một tệp JSON chứa thông tin xác thực để gọi API của Google Cloud Vision.

#### 7.2.4.3. Khởi tạo client của Vision API

| client = vision.ImageAnnotatorClient() |
| --- |

* Đoạn mã này tạo một đối tượng client để tương tác với Google Cloud Vision API.

#### 7.2.4.4. Định nghĩa hàm detect\_logos

| def detect\_logos(path):  with open(path, "rb") as image\_file:  content = image\_file.read()  image = vision.Image(content=content)  response = client.logo\_detection(image=image)  logos = response.logo\_annotations  print("Logos:")  for logo in logos:  print(f' {logo.description}') |
| --- |

Hàm **detect\_logos** dùng phát hiện các biểu trưng (logos) trên một hình ảnh.

1. **def detect\_logos(path):** Hàm nhận một đối số đầu vào là path. path là đường dẫn đến tệp hình ảnh mà bạn muốn phát hiện biểu trưng trên.
2. **with open(path, "rb") as image\_file:**: Dòng này mở tệp hình ảnh tại đường dẫn path để đọc dưới dạng dữ liệu nhị phân ("rb" là chế độ đọc nhị phân). Điều này cho phép bạn đọc nội dung của hình ảnh từ tệp và lưu trữ nó trong biến content.
3. **image = vision.Image(content=content)**: Dòng này tạo một đối tượng hình ảnh (image) sử dụng nội dung hình ảnh đã được đọc. Đối tượng image sẽ được sử dụng để gửi yêu cầu phát hiện biểu trưng đến Google Cloud Vision API.
4. **response = client.logo\_detection(image=image)**: Dòng này gửi yêu cầu phát hiện biểu trưng từ hình ảnh (image) bằng cách sử dụng đối tượng Vision API đã được khởi tạo (biến client). Kết quả từ API sẽ được lưu trữ trong biến response.
5. **logos = response.logo\_annotations**: Dòng này trích xuất thông tin về biểu trưng từ kết quả phản hồi của API và lưu trữ nó trong biến logos. Biến logos sẽ chứa danh sách các biểu trưng được tìm thấy trên hình ảnh.
6. **print("Logos:")**: Dòng này in ra chuỗi "Logos:" để bắt đầu hiển thị danh sách các biểu trưng.
7. **for logo in logos:**: Dòng này bắt đầu một vòng lặp để duyệt qua từng biểu trưng trong danh sách logos.
8. **print(f' {logo.description}')**: Trong vòng lặp, hàm in ra mô tả của biểu trưng hiện tại (được lưu trong logo.description). Điều này sẽ hiển thị tên hoặc mô tả của biểu trưng.

#### 7.2.4.7. Sử dụng hàm

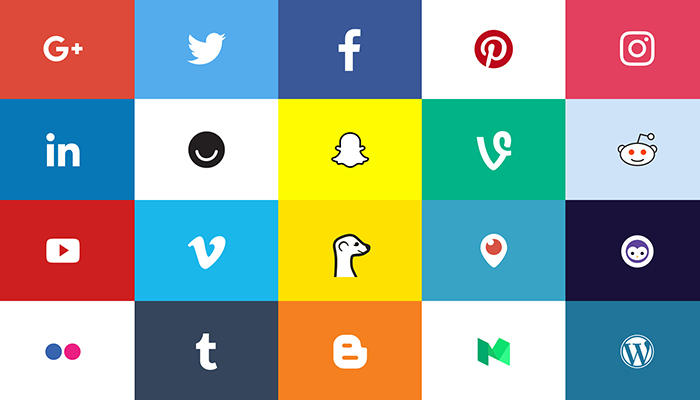
| img\_path = './resources/fb.jpg'  detect\_logos(img\_path) |
| --- |

### 7.2.5. Chi tiết mã nguồn

| import os  from google.cloud import vision  os.environ['GOOGLE\_APPLICATION\_CREDENTIALS'] = 'ServiceAccToken.JSON'  client = vision.ImageAnnotatorClient()  def detect\_logos(path):  with open(path, "rb") as image\_file:  content = image\_file.read()  image = vision.Image(content=content)  response = client.logo\_detection(image=image)  logos = response.logo\_annotations  print("Logos:")  for logo in logos:  print(f' {logo.description}')  img\_path = './resources/fb.jpg'  detect\_logos(img\_path) |
| --- |

### 7.2.7. Chạy chương trình

Hình ảnh đầu vào



Kết quả nhận được

