Vision API
Product overview
Features list
Try it!

Quickstarts
All Quickstarts
Set up the Vision API
Using client libraries
Using the command line
Using API explorer

Samples
All Vision API code samples
All code samples for all products

How-to Guides
All How-to guides
Before you begin
Optical character recognition (OCR)
Detect crop hints
Detect faces
Detect image properties
Detect labels
Detect landmarks
Detect logos
Detect multiple objects
Detect explicit content (SafeSearch)
Detect Web entities and pages
Batch feature detection
Using Vision with Spring framework
Base64 encode
Restricted access

Tutorials
All tutorials
Text detection (OCR) tutorial
Translating and speaking text from a photo
Face detection tutorial
Label detection tutorial
Document Text tutorial
Web detection tutorial
Crop hints tutorial
Search community tutorials

Security logging
Cloud audit logs

Document Text Tutorial

Send feedback

Table of contents

Audience
Prerequisites
Annotating an image using Document Text OCR
Complete code listing
A closer look at the code
•••

Audience

The goal of this tutorial is to help you develop applications using Google Cloud Vision API Document Text Detection. It assumes you are familiar with basic programming constructs and techniques, but even if you are a beginning programmer, you should be able to follow along and run this tutorial without difficulty, then use the Cloud Vision API reference documentation to create basic applications.

Prerequisites

- Set up a Cloud Vision API project in the Google Cloud Console.
- Set up your environment for using Application Default Credentials.

Python

- Install Python.
- Install pip.
- Install the Google Cloud Client Library and the Python Imaging Library.

Annotating an image using Document Text OCR

This tutorial walks you through a basic Vision API application that makes a `DOCUMENT_TEXT_DETECTION` *request*, then processes the `fullTextAnnotation` *response*.

> ★ Note that both standard **`TEXT_DETECTION`** and **`DOCUMENT_TEXT_DETECTION`** return **`fullTextAnnotations`**, as described below. However, there is no input character limit on the premium **`DOCUMENT_TEXT_DETECTION`** feature. Also, if both **`TEXT_DETECTION`** and **`DOCUMENT_TEXT_DETECTION`** are specified in a Cloud Vision request, **`DOCUMENT_TEXT_DETECTION`** will take precedence.

A `fullTextAnnotation` is a structured hierarchical response for the UTF-8 text extracted from the image, organized as Pages→Blocks→Paragraphs→Words→Symbols:

- `Page` is a collection of blocks, plus meta-information about the page: sizes, resolutions (X resolution and Y resolution may differ).
- `Block` represents one "logical" element of the page—for example, an area covered by text, or a picture or separator between columns. The text and table blocks contain the main information needed to extract the text.
- `Paragraph` is a structural unit of text representing an ordered sequence of words. By default, words are considered to be separated by word breaks.
- `Word` is the smallest unit of text. It is represented as an array of Symbols.
- `Symbol` represents a character or a punctuation mark.

The `fullTextAnnotation` also can provide URLs to Web images that partially or fully match the image in the request.

> ★ The previous **`textAnnotations`** OCR output will continue to be supported, and is available in the JSON Response as **`textAnnotations`**.

Complete code listing

As you read the code, we recommend that you follow along by referring to the Cloud Vision API Python reference.

samples/snippets/document_text/doctext.py View on GitHub

```
import argparse
from enum import Enum
import io

from google.cloud import vision
from PIL import Image, ImageDraw


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


def draw_boxes(image, bounds, color):
    """Draw a border around the image using the hints in the vector list."""
    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):
    """Returns document bounds given an image."""
    client = vision.ImageAnnotatorClient()

    bounds = []

    with io.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

    # Collect specified feature bounds by enumerating all document features
    for page in document.pages:
        for block in page.blocks:
            for paragraph in block.paragraphs:
                for word in paragraph.words:
                    for symbol in word.symbols:
                        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)

    # The list `bounds` contains the coordinates of the bounding boxes.
    return bounds


def render_doc_text(filein, fileout):
    image = Image.open(filein)
    bounds = get_document_bounds(filein, FeatureType.BLOCK)
    draw_boxes(image, bounds, 'blue')
    bounds = get_document_bounds(filein, FeatureType.PARA)
    draw_boxes(image, bounds, 'red')
    bounds = get_document_bounds(filein, FeatureType.WORD)
    draw_boxes(image, bounds, 'yellow')

    if fileout != 0:
        image.save(fileout)
    else:
        image.show()


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('detect_file', help='The image for text detection.')
    parser.add_argument('-out_file', help='Optional output file', default=0)
    args = parser.parse_args()

    render_doc_text(args.detect_file, args.out_file)
```

This simple application performs the following tasks:

- Imports the libraries necessary to run the application
- Takes three arguments passes it to the `main()` function:
  - `image_file` — the input image file to be annotated
  - `output_file` —the output filename into which Cloud Vision will generate an output image with polyboxes drawn
- Creates an `ImageAnnotatorClient` instance to interact with the service
- Sends the request and returns a response
- Creates an output image with boxes drawn around the text

A closer look at the code

Importing libraries

samples/snippets/document_text/doctext.py View on GitHub

```
import argparse
from enum import Enum
import io

from google.cloud import vision
from PIL import Image, ImageDraw
```

We import standard libraries:

- `argparse` to allow the application to accept input file names as arguments
- `enum` for the `FeatureType` enumeration
- `io` for File I/O

Other imports:

- The `ImageAnnotatorClient` class within the `google.cloud.vision` library for accessing the Vision API.
- The `types` module within the `google.cloud.vision` library for constructing requests.
- The `Image` and `ImageDraw` libraries from the `PIL` library are used to create the output image with boxes drawn on the input image.

Running the application

samples/snippets/document_text/doctext.py View on GitHub

```
parser = argparse.ArgumentParser()
parser.add_argument('detect_file', help='The image for text detection.')
parser.add_argument('-out_file', help='Optional output file', default=0)
args = parser.parse_args()
```

```
render_doc_text(args.detect_file, args.out_file)
```

Here, we simply parse the passed-in arguments and pass it to the `render_doc_text()` function.

Authenticating to the API

Before communicating with the Vision API service, you must authenticate your service using previously acquired credentials. Within an application, the simplest way to obtain credentials is to use Application Default Credentials (ADC). By default, the Cloud client library will attempt to obtain credentials from the `GOOGLE_APPLICATION_CREDENTIALS` environment variable, which should be set to point to your service account's JSON key file (see Setting Up a Service Account for more information).

Making the API request and reading text bounds from the response

Now that our Vision API service is ready, we can access the service by calling the `document_text_detection` method of the `ImageAnnotatorClient` instance.

The client library encapsulates the details for requests and responses to the API. See the Vision API Reference for complete information on the structure of a request.

samples/snippets/document_text/doctext.py

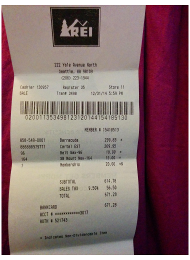
View on GitHub

```
"""Returns document bounds given an image."""
client = vision.ImageAnnotatorClient()

bounds = []

with io.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

# Collect specified feature bounds by enumerating all document features
for page in document.pages:
    for block in page.blocks:
        for paragraph in block.paragraphs:
            for word in paragraph.words:
                for symbol in word.symbols:
                    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)

# The list `bounds` contains the coordinates of the bounding boxes.
```

After the client library has handled the request, our response will contain an AnnotateImageResponse, which consists of a list of Image Annotation results, one for each image sent in the request. Because we sent only one image in the request, we walk through the full TextAnnotation, and collect the boundaries for the specified document feature.

Running the application

To run the application, you can download this `receipt.jpg` file (you may need to right-click the link), then pass the location where you downloaded the file on on your local machine to the tutorial application (`doctext.py`).



222 Yale Avenue North
Seattle, WA 98109
(206) 223-1944

Cashier 130957 Register 35 Store 11
SALE Tran# 3498 12/31/14 5:59 PM

02001135349812312014415418513 0

MEMBER # 15418513

858-549-0001 Barracuda 299.83 *
886888979771 Cartel EST 269.95
96 Belt Wax-96 10.00 *
164 SB Mount New-164 15.00 *
1 Membership 20.00 *N

SUBTOTAL 614.78
SALES TAX 9.50% 56.50
TOTAL 671.28

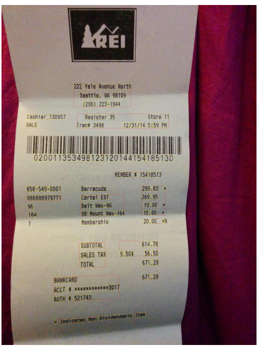
BANKCARD 671.28
ACCT # ************3017
AUTH # 521743

* Indicates Non-Dividendable Item

Here is the Python command, followed by the Text Annotation output images.

```
$ python doctext.py receipt.jpg -out_file out.jpg
```

The following image shows words in yellow boxes and sentences in red.



222 Yale Avenue North
Seattle, WA 98109
(206) 223-1944

Cashier 130957 Register 35 Store 11
SALE Tran# 3498 12/31/14 5:59 PM

020011353498123120144154185130

MEMBER # 15418513

858-549-0001 Barracuda 299.83 *
886888979771 Cartel EST 269.95
96 Belt Wax-96 10.00 *
164 SB Mount New-164 15.00 *
1 Membership 20.00 *N

SUBTOTAL 614.78
SALES TAX 9.50% 56.50
TOTAL 671.28

BANKCARD 671.28
ACCT # ************3017
AUTH # 521743

* Indicates Non-Dividendable Item

Congratulations! You've performed Text Detection using Google Cloud Vision Full Text Annotations!

Rate and review

Send feedback

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 4.0 License, and code samples are licensed under the Apache 2.0 License. For details, see the Google Developers Site Policies. Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2021-06-08 UTC.

Why Google
Choosing Google Cloud
Trust and security
Open cloud
Multicloud
Global infrastructure
Sustainability
Customers and case studies
Analyst reports
Whitepapers

Products and pricing
GCP pricing
Google Workspace pricing
Maps Platform pricing
See all products

Solutions
Infrastructure modernization
Databases
Application modernization
Smart analytics
Artificial Intelligence
Security
Productivity & work transformation
Industry solutions
DevOps solutions
Small business solutions
See all solutions

Resources
GCP documentation
GCP quickstarts
Google Cloud Marketplace
Google Workspace Marketplace
Support
Code samples
Tutorials
Training
Certifications
Google Developers
Google Cloud for Startups
System status
Release Notes

Engage
Contact sales
Find a Partner
Become a Partner
Blog
Events
Podcast
Developer Center
Press center
Google Cloud on YouTube
Google Cloud Tech on YouTube
Google Workspace on YouTube
Follow on Twitter
Join User Research
We're hiring. Join Google Cloud!

About Google | Privacy | Site terms | Google Cloud terms
Carbon neutral since 2007
Sign up for the Google Cloud newsletter
Subscribe
English