

[AI & Machine Learning Products](https://cloud.google.com/products/machine-learning/) (<https://cloud.google.com/products/machine-learning/>)

[Cloud Vision API](https://cloud.google.com/vision/) (<https://cloud.google.com/vision/>)

[Documentation](https://cloud.google.com/vision/docs/) (<https://cloud.google.com/vision/docs/>) [Guides](#)

# Document Text Tutorial

## 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](https://cloud.google.com/vision/reference/rest/) (<https://cloud.google.com/vision/reference/rest/>) to create basic applications.

## Prerequisites

- [Set up a Cloud Vision API project](https://cloud.google.com/vision/docs/getting-started#set_up_a_google_cloud_vision_api_project) ([https://cloud.google.com/vision/docs/getting-started#set\\_up\\_a\\_google\\_cloud\\_vision\\_api\\_project](https://cloud.google.com/vision/docs/getting-started#set_up_a_google_cloud_vision_api_project)) in the Google Cloud Console.
- Set up your environment for using [Application Default Credentials](https://cloud.google.com/vision/docs/common/auth#authenticating_with_application_default_credentials) ([https://cloud.google.com/vision/docs/common/auth#authenticating\\_with\\_application\\_default\\_credentials](https://cloud.google.com/vision/docs/common/auth#authenticating_with_application_default_credentials))

### PYTHON

- Install [Python](https://www.python.org/) (<https://www.python.org/>).
- [Install pip](https://pip.pypa.io/en/latest/installing/) (<https://pip.pypa.io/en/latest/installing/>).
- Install the [Google Cloud Client Library](https://cloud.google.com/vision/docs/reference/libraries#installing_the_client_library) ([https://cloud.google.com/vision/docs/reference/libraries#installing\\_the\\_client\\_library](https://cloud.google.com/vision/docs/reference/libraries#installing_the_client_library)) and the [Python Imaging Library](http://python-pillow.github.io/) (<http://python-pillow.github.io/>).

## 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](https://googleapis.github.io/google-cloud-python/latest/vision/index.html) (<https://googleapis.github.io/google-cloud-python/latest/vision/index.html>).

```
vision/cloud-client/document_text/doctext.py  
(https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/vision/cloud-client/document\_text/doctext.py)
```

```
FORM/PYTHON-DOCS-SAMPLES/BLOB/MASTER/VISION/CLOUD-CLIENT/DOCUMENT_TEXT/DOCTEXT.PY
```



```
import argparse
from enum import Enum
import io

from google.cloud import vision
from google.cloud.vision import types
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 = types.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

[vision/cloud-client/document\\_text/doctext.py](https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/vision/cloud-client/document_text/doctext.py)

([https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/vision/cloud-client/document\\_text/doctext.py](https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/vision/cloud-client/document_text/doctext.py))

FORM/PYTHON-DOCS-SAMPLES/BLOB/MASTER/VISION/CLOUD-CLIENT/DOCUMENT\_TEXT/DOCTEXT.PY)

```
import argparse
from enum import Enum
import io

from google.cloud import vision
from google.cloud.vision import types
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

[vision/cloud-client/document\\_text/doctext.py](https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/vision/cloud-client/document_text/doctext.py)

([https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/vision/cloud-client/document\\_text/doctext.py](https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/vision/cloud-client/document_text/doctext.py))

```
FORM/PYTHON-DOCS-SAMPLES/BLOB/MASTER/VISION/CLOUD-CLIENT/DOCUMENT_TEXT/DOCTEXT.PY
```

```
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](https://cloud.google.com/vision/docs/common/auth#adc) (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](https://cloud.google.com/vision/docs/common/auth#set_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](https://cloud.google.com/vision/reference/rest) (https://cloud.google.com/vision/reference/rest) for complete information on the structure of a request.

```
vision/cloud-client/document_text/doctext.py
```

```
(https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/vision/cloud-client/document_text/doctext.py)
```

```
FORM/PYTHON-DOCS-SAMPLES/BLOB/MASTER/VISION/CLOUD-CLIENT/DOCUMENT_TEXT/DOCTEXT.PY)
```

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

bounds = []

with io.open(image_file, 'rb') as image_file:
    content = image_file.read()

image = types.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

(<https://cloud.google.com/vision/reference/rest/v1/images/annotate#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 (<https://cloud.google.com/vision/reference/rest/v1/images/annotate#TextAnnotation>), and collect the boundaries for the specified document feature.

## Running the application

To run the application, you can download this receipt.jpg file (<https://cloud.google.com/vision/docs/images/receipt.jpg>) (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`).



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.





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

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](https://creativecommons.org/licenses/by/4.0/) (https://creativecommons.org/licenses/by/4.0/), and code samples are licensed under the [Apache 2.0 License](https://www.apache.org/licenses/LICENSE-2.0) (https://www.apache.org/licenses/LICENSE-2.0). For details, see our [Site Policies](https://developers.google.com/terms/site-policies) (https://developers.google.com/terms/site-policies). Java is a registered trademark of Oracle and/or its affiliates.

Last updated January 10, 2020.