StackAbuse

PyTesseract: Simple Python Optical Character Recognition

Robley Gori

Introduction

Humans can understand the contents of an image simply by looking. We perceive the text on the image as text and can read it.

Computers don't work the same way. They need something more concrete, organized in a way they can understand.

This is where *Optical Character Recognition* (OCR) kicks in. Whether it's recognition of car plates from a camera, or hand-written documents that should be converted into a digital copy, this technique is very useful. While it's not always perfect, it's very convenient and makes it a lot easier and faster for some people to do their jobs.

In this article, we will delve into the depth of Optical Character Recognition and its application areas. We will also build a simple script in Python that will help us detect characters from images and expose this through a Flask application for a more convenient interaction medium.

What is Optical Character Recognition?

Optical Character Recognition involves the detection of text content on images and translation of the images to *encoded text* that the computer can easily understand. An

image containing text is scanned and analyzed in order to identify the characters in it. Upon identification, the character is converted to machine-encoded text.

How is it really achieved? To us, text on an image is easily discernible and we are able to detect characters and read the text, but to a computer, it is all a series of dots.

The image is first scanned and the text and graphics elements are converted into a bitmap, which is essentially a matrix of black and white dots. The image is then pre-processed where the brightness and contrast are adjusted to enhance the accuracy of the process.

The image is now split into zones identifying the areas of interest such as where the images or text are and this helps kickoff the extraction process. The areas containing text can now be broken down further into lines and words and characters and now the software is able to match the characters through comparison and various detection algorithms. The final result is the text in the image that we're given.

The process may not be 100% accurate and might need human intervention to correct some elements that were not scanned correctly. Error correction can also be achieved using a dictionary or even *Natural Language Processing* (NLP).

The output can now be converted to other mediums such as word documents, PDFs, or even audio content through text-to-speech technologies.

Uses of OCR

Previously, digitization of documents was achieved by manually typing the text on the computer. Through OCR, this process is made easier as the document can be scanned, processed and the text extracted and stored in an editable form such as a word document.

If you have a document scanner on your phone, such as Adobe Scan, you have probably encountered OCR technology in use.

Airports can also use OCR to automate the process of passport recognition and extraction of information from them.

Other uses of OCR include automation of data entry processes, detection, and recognition of car number plates.

What we'll Use

For this OCR project, we will use the Python-Tesseract, or simply *PyTesseract*, library which is a wrapper for Google's Tesseract-OCR Engine.

I chose this because it is completely open-source and being developed and maintained by the giant that is Google. Follow these instructions to install Tesseract on your machine, since PyTesseract depends on it.

We will also use the Flask web framework to create our simple OCR server where we can take pictures via the webcam or upload photos for character recognition purposes.

We are also going to use Pipenv since it also handles the virtual-environment setup and requirements management.

Besides those, we'll also use the Pillow library which is a fork of the *Python Imaging Library* (PIL) to handle the opening and manipulation of images in many formats in Python.

In this post, we'll concentrate on *PyTesseract* although there are other Python libraries that can help you extract text from images such as:

- Textract: which can extract data from PDFs but is a heavy package.
- Pyocr: offers more detection options such as sentences, digits, or words.

Setup

Start by installing *Pipenv* using the following command via Pip (In case you need to set it up, refer to this).

```
$ pip install pipenv
```

Create the project directory and initiate the project by running the following command:

```
$ mkdir ocr_server && cd ocr_server && pipenv install --three
```

We can now activate our virtual environment and start installing our dependencies:

```
$ pipenv shell
$ pipenv install pytesseract Pillow
```

In case you'll not be using Pipenv, you can always use the Pip and Virtual Environment approach. Follow the official documentation to help you get started with Pip and Virtual Environment:

Note: In that case, instead of `pipenv install Pillow`, the command will be `pip install Pillow`.

Implementation

We are going to implement this project in 2 phases. In the first, we'll create the script, and in the next we'll build a Flask application to act as an interface.

OCR Script

With the setup complete, we can now create a simple function that takes an image and returns the text detected in the image - this will be the core of our project:

```
try:
    from PIL import Image
except ImportError:
    import Image
```

```
import pytesseract

def ocr_core(filename):
    """
    This function will handle the core OCR processing of images.
    """
    text = pytesseract.image_to_string(Image.open(filename))  # We'll use Pillow'
    return text

print(ocr_core('images/ocr_example_1.png'))
```

The function is quite straightforward, in the first 5 lines we import `Image` from the `Pillow` library and our `PyTesseract` library.

We then create and `ocr_core` function that takes in a file name and returns the text contained in the image.

Let's see how the script fares with a simple image containing some text:

And upon running the piece of code, we're greeted with this:

```
(ocr_project-WEQqYocZ) robley at Mufasa in ~/Desktop/code/python/ocr_project git:(master) ✗
$ python3 ocr_core.py

A Python Approach to Character
Recognition
```

Our simple OCR script works! Obviously, this was somewhat easy since this is digital text, perfect and precise, unlike handwriting. There's a lot more that we can do with the PyTesseract library, but more on this later in the post.

Let's integrate this script into a Flask application first, to make it easier to upload images and perform character recognition operations.

Flask Web Interface

Our script can be used via the command line, but a Flask application would make it more user-friendly and versatile. For instance, we can upload photos via the website and get the extracted text displayed on the website or we can capture photos via the web camera and perform character recognition on them.

If you are unfamiliar with the Flask framework, this is a good tutorial to get you up to speed and going.

Let's start by installing the Flask package:

```
$ pipenv install Flask
```

Now, let's define a basic route:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home_page():
    return "Hello World!"

if __name__ == '__main__':
    app.run()
```

Save the file and run:

```
$ python3 app.py
```

If you open your browser and head on to `127.0.0.1:5000` or `localhost:5000` you should see "Hello World!" on the page. This means our Flask app is ready for the next steps.

We'll now create a `templates` folder to host our HTML files. Let's go ahead and create a simple `index.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index</title>
  </head>
  <body>
    Hello World.
  </body>
</html>
```

Let us also tweak our `app.py` to render our new template:

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/')
def home_page():
    return render_template('index.html')

if __name__ == '__main__':
    app.run()
```

Notice we have now imported `render_template` and used it to render the HTML file. If you restart your Flask app, you should still see "Hello World!" on the home page.

That's enough on the Flask crash course, let us now integrate our OCR script on the web application.

First, we'll add functionality to upload images to our Flask app and pass them to the `ocr_core` function that we wrote above. We will then render the image beside the extracted text on our web app as a result:

```
import os
from flask import Flask, render_template, request

# import our OCR function
from ocr_core import ocr_core

# define a folder to store and later serve the images
UPLOAD_FOLDER = '/static/uploads/'

# allow files of a specific type
ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg'])

app = Flask(__name__)

# function to check the file extension
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

# route and function to handle the home page
@app.route('/')
def home_page():
    return render_template('index.html')

# route and function to handle the upload page
@app.route('/upload', methods=['GET', 'POST'])
def upload_page():
    if request.method == 'POST':
        # check if there is a file in the request
        if 'file' not in request.files:
            return render_template('upload.html', msg='No file selected')
        file = request.files['file']
        # if no file is selected
        if file.filename == '':
            return render_template('upload.html', msg='No file selected')
```

```
        if file and allowed_file(file.filename):

            # call the OCR function on it
            extracted_text = ocr_core(file)

            # extract the text and display it
            return render_template('upload.html',
                                   msg='Successfully processed',
                                   extracted_text=extracted_text,
                                   img_src=UPLOAD_FOLDER + file.filename)
    elif request.method == 'GET':
        return render_template('upload.html')

if __name__ == '__main__':
    app.run()
```

As we can see in our `upload_page()` function, we will receive the image via *POST* and render the upload HTML if the request is *GET*.

We check whether the user has really uploaded a file and use the function `allowed_file()` to check if the file is of an acceptable type.

Upon verifying that the image is of the required type, we then pass it to the character recognition script we created earlier.

The function detects the text in the image and returns it. Finally, as a response to the image upload, we render the detected text alongside the image for the user to see the results.

The `upload.html` file will handle the posting of the image and rendering of the result by the help of the Jinja templating engine, which ships with Flask by default:

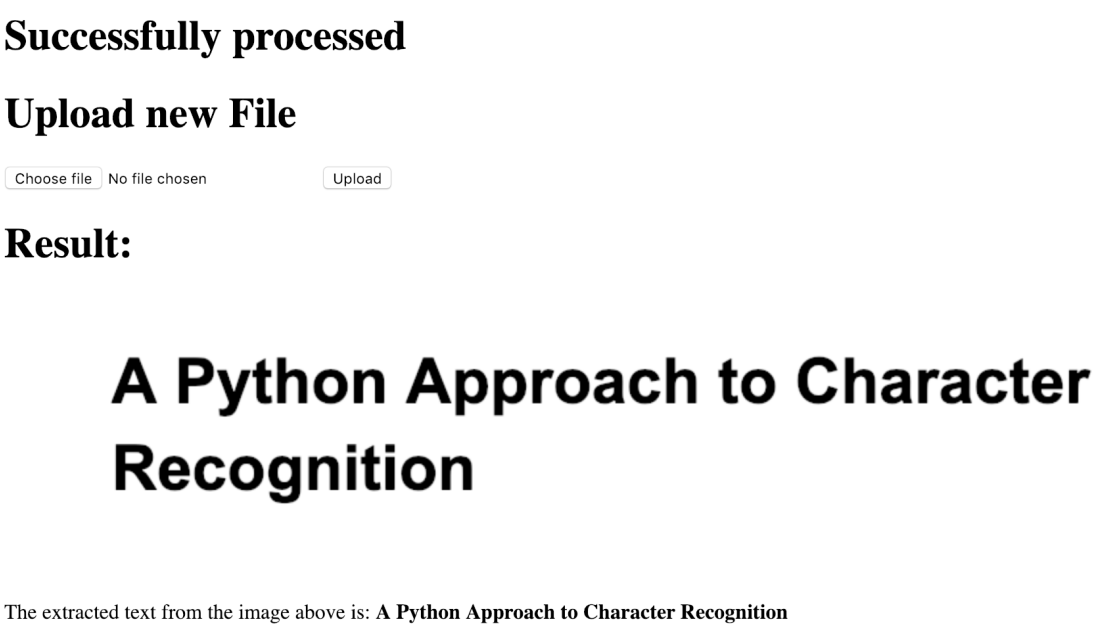
```
<!DOCTYPE html>
<html>
 <head>
   <title>Upload Image</title>
 </head>
 <body>

   {% if msg %}
   <h1>{{ msg }}</h1>
   {% endif %}
```

```
    <h1>Upload new File</h1>

    <form method=post enctype=multipart/form-data>
      <p><input type=file name=file>
         <input type=submit value=Upload>
    </form>

    <h1>Result:</h1>
    {% if img_src %}
      <img src="{{ img_src }}">
    {% endif %}

    {% if extracted_text %}
      <p> The extracted text from the image above is: <b> {{ extracted_text }} </b

    {% else %}
      The extracted text will be displayed here
    {% endif %}
  </body>
</html>
```

Jinja templating allows us to display text in specific scenarios through the `{% if %} {% endif %}` tags. We can also pass messages from our Flask app to be displayed on the webpage within the `{{ }}` tags. We use a form to upload the image to our Flask app.

The result is:

Now, if we go ahead and upload our image from earlier:



Yes! Our Flask application has been able to integrate the OCR functionality and display the text on the browser. This makes it easier to process images instead of running commands on the CLI every time we have a new image to process.

Let's attach some more images to further explore the limits of our simple OCR script since it will not work in all situations.

For example, let's try extracting text from the following image and the result has been highlighted on the image:

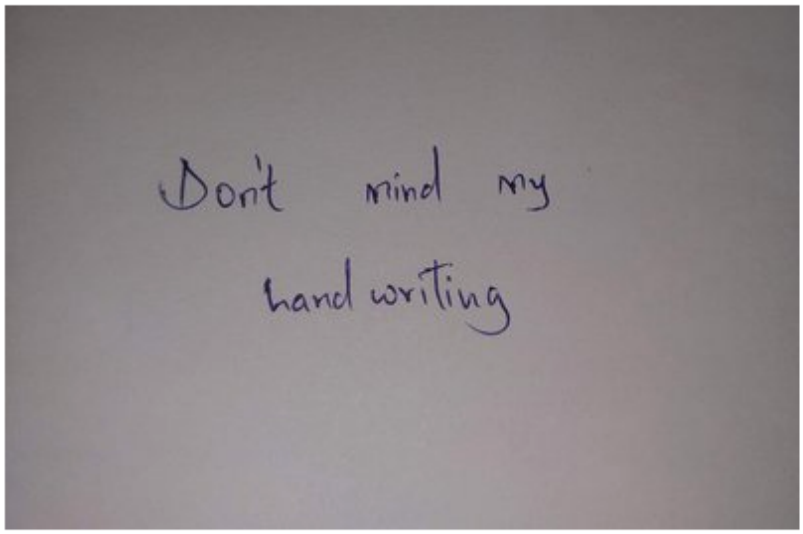
Result:

The extracted text from the image above is: **7 TU et r= i Uae Lt) Me TE**

This is evidence that OCR is not always 100% accurate and may need human intervention from time to time.

I also tested the OCR script against my handwriting to see how it would perform, and this is the result:

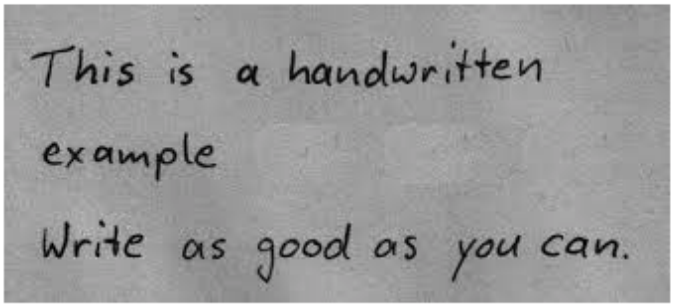
Result:



The extracted text from the image above is: **ad oviling**

As you can see, it cannot quite extract text from my handwriting as it did with other images we've seen before. I decided to give it another try, this time with an image from this source, and these were the results:

Result:



The extracted text from the image above is: **This is a handwritten example Write as geoal as you can.**

The character recognition on this image is much better than the one where I used my own handwriting. As you can see the lines in the downloaded image are thicker and there's better contrast between the text and the background and this could be the reason for the poor detection on my handwriting.

This is an area to explore further, you can get handwritten notes from friends or colleagues and see how well the script will be able to detect characters. You can even get posters to events and try scanning them for text, the possibilities are plenty.

Other PyTesseract Options

Python-Tesseract has more options you can explore. For example, you can specify the language by using a `lang` flag:

```
pytesseract.image_to_string(Image.open(filename), lang='fra')
```

This is the result of scanning an image without the `lang` flag:

Result:

The extracted text from the image above is: **Cae ea ao LOCALE SEULEMENT, EN CAS DE FEU SIGNALER 911**

And now with the `lang` flag:

Result:

The extracted text from the image above is: **CETTE ALARME EST LOCALE SEULEMENT, EN CAS DE FEU SIGNALER 911**

The framework is also optimized to detect languages better as seen in the screenshots. (Image source).

Without the `lang` flag, the script missed some French words, but after introducing the flag it was able to detect all the French content. Translation is not possible but this is still impressive. Tesseract's official documentation includes the supported languages in this section.

Orientation and script detection is also among the capabilities of PyTesseract and this aids in the detection of the fonts used and orientation of the text on the given image. If we may refer to the handwritten image we downloaded earlier:

```
print(pytesseract.image_to_osd(Image.open('downloaded_handwritten.png')))
```

```
Page number: 0
Orientation in degrees: 0
Rotate: 0
Orientation confidence: 0.32
Script: Arabic
Script confidence: 4.44
```

There was no page number information on the image so this was not detected. The Tesseract engine is able to extract information about the orientation of the text in the image and rotation. The orientation confidence is a figure of the surety of the engine about the orientation detected to act as a guide and to also show that it is not always 100% accurate. The script section denotes the writing system used in the text and this is also followed by the confidence marker.

If we were after the recognized characters and their box boundaries, PyTesseract achieves this through `pytesseract.image_to_boxes(Image.open('downloaded_handwritten.png'))`.

These are some of the capabilities of PyTesseract among others such as conversion of the extracted text into a searchable PDF or HOCR output.

What we haven't Done

We have accomplished a lot in this post, but there is still more to do to refine our project and prepare it for the real world. First, we can add style to our website and make it more

appealing to the end user by using CSS. We can also add the option to upload and scan multiple images at once and displaying all their output at once. Wouldn't this make it more convenient to scan multiple documents?

The browser allows us to tap into a machine's camera and capture images, with permission from the user, of course. This can be of great help especially on mobile devices. Instead of the user having to capture and save the image then uploading it on the website, if we add the camera functionality, we can allow the user to perform the operations directly from the Flask web application. This will make the scanning process faster.

Suppose a Flask application is not what you intended to expose your OCR scanner, you can also create a CLI tool. The tool would allow you to run a command including the location of the image and then printing the output of the scanner to your terminal or sending it to a database or API. If you chose this path Docopt is a fantastic tool for building command line tools using Python.

Conclusion

Through Tesseract and the Python-Tesseract library, we have been able to scan images and extract text from them. This is Optical Character Recognition and it can be of great use in many situations.

We have built a scanner that takes an image and returns the text contained in the image and integrated it into a Flask application as the interface. This allows us to expose the functionality in a more familiar medium and in a way that can serve multiple people simultaneously.

The source code for this project is available here on Github.

#python #computer vision

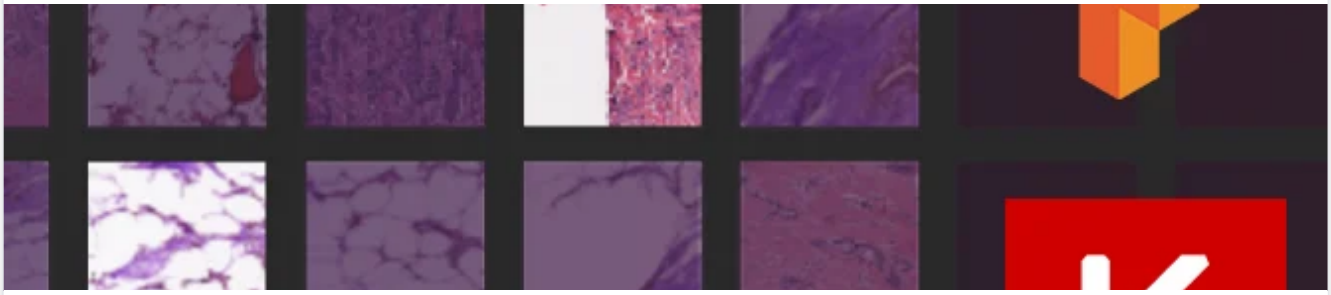
Last Updated: April 8th, 2019

You might also like...

- Object Detection with ImageAI in Python

- Course Review: Hands On Computer Vision with OpenCV & Python
- Affine Image Transformations in Python with Numpy, Pillow and OpenCV
- Object Detection with OpenCV-Python Using a Haar-Cascade Classifier

Robley Gori *Author*



Project

Breast Cancer Classification with Deep Learning - Keras and Tensorflow

#python #machine learning #tensorflow #computer vision

As Data Scientists and Machine Learning Engineers - we're exploring prospects of applying Machine Learning algorithms to various domains and extracting knowledge from data. Fast,...

Details →

Course

Data Visualization in Python with Matplotlib and Pandas

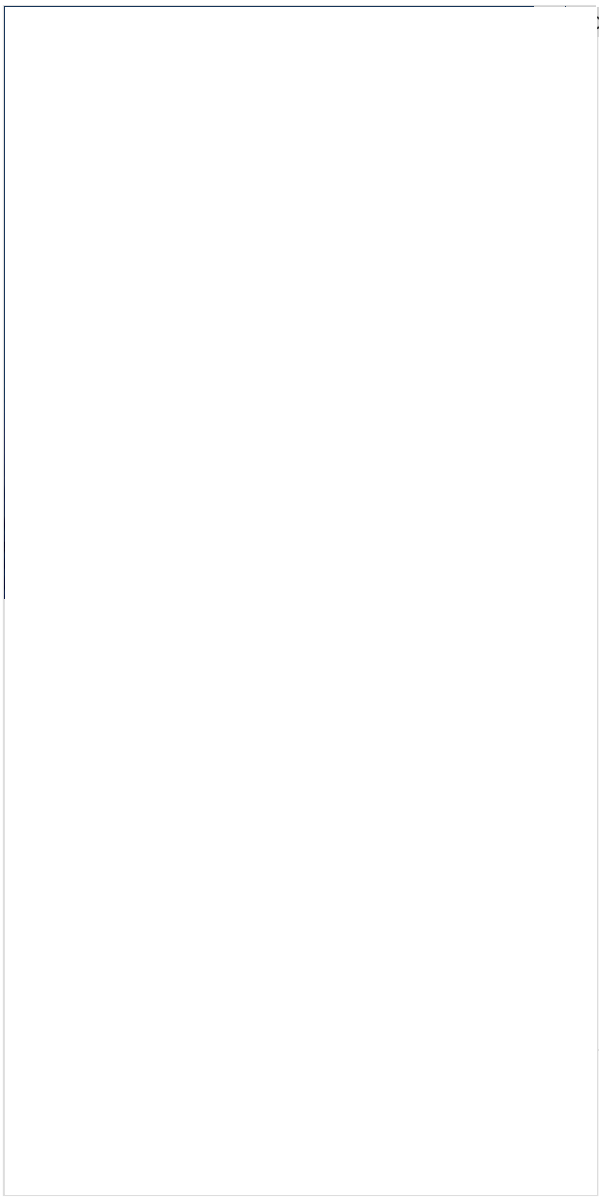
#python #pandas #matplotlib

Data Visualization in Python with Matplotlib and Pandas is a course designed to take

absolute beginners to Pandas and Matplotlib, with basic Python knowledge, and...

David Landup

Details →





Disclosure | Privacy | Terms