piltesseract Documentation

Release 0.0.1

Christopher Digirolamo

April 04, 2016

Contents

Table of Contents:

CHAPTER 1

Installation

Installing PILtesseract is very simple, but before installing it, you should make sure you install the requirements.

1.1 Tesseract-OCR

PILtesseract call the Tesseract-OCR command line tool, the tool must be installed and on your PATH variable before using PILtesseract.

Make sure you install version 3.03 or higher.

- Install either:
  - from source
  - from binaries
    - **Windows users may have to download from third-party installers like UB Mannheim**
- Ensure that the tesseract binary folder is on your PATH.

1.2 Pillow

The fork of the Python Image Library ("PIL")

- Pillow
  - `$ pip install Pillow`

Note The Pillow library installs as PIL, so you import it like: `import PIL`

1.3 Six

Library for python 2 and 3 compatibility

- Six
  - `$ pip install six`

1.4 PILtesseract

Finally you can now install this library.

- Installing from pip: - `$ pip install piltesseract`

CHAPTER 2

The tesseractwrapper Module

This module contains all of the tesseract wrapping and image-to-text code.

tesseractwrapper.**TESSERACT_DIR**
str

The default path to the tesseract install directory.

tesseractwrapper.**DEFAULT_FORMAT**
str

The default image format to send to tesseract if an image doesn't not have a declared format. Otherwise, try to use the former format if we can.

2.1 Main Function

tesseractwrapper.**get_text_from_image**(*image*, *tesseract_dir_path=u''*, *stderr=None*, *psm=3*, *lang=u'eng'*, *tessdata_dir_path=None*, *user_words_path=None*, *user_patterns_path=None*, *config_name=None*, ***config_variables*)

Uses tesseract to get text from an image.

Outside of image, tesseract_dir_path, and stderr, the arguments mirror the official command line's usage. A list of the command line options can be found here: https://tesseract-ocr.googlecode.com/svn/trunk/doc/tesseract.1.html

Parameters

- **image** (*Image.Image or str*) – The image to find text from or a path to that image.
- **tesseract_dir_path** (*Optional[str]*) – The path to the directory with the tesseract binary. Defaults to "", which works if the binary is on the environmental PATH variable.
- **stderr** (*Optional[file]*) – The file like object (implements *write*) the tesseract stderr stream will write to. Defaults to None. You can set it to sys.stdin to see all output easily.
- **psm** (*Optional[int]*) – Page Segmentation Mode. Limits Tesseracts layout analysis (see the Tesseract docs). Default is 3, full analysis.
- **lang** (*Optional[str]*) – The language to use. Default is 'eng' for English.
- **tessdata_dir_path** (*Optional[str]*) – The path to the tessdata directory.
- **user_words_path** (*Optional[str]*) – The path to user words file.
- **user_patterns_path** (*Optional[str]*) – The path to the user patterns file.

- **config_name** (*Optional[str]*) – The name of a config file.
- ****config_variables** – The config variables for tesseract. A list of config variables can be found here: http://www.sk-spell.sk.cx/tesseract-ocr-parameters-in-302-version

Returns The parsed text.

Return type str

Raises subprocess.CalledProcessError – If the tesseract exit status is not a success.

Examples

Examples assume "image" is a picture of the text "ABC123". See piltesseract tests for working code.

```
>>> get_text_from_image(image)
'ABC123'
>>> get_text_from_image(image, psm=10)  #single character psm
'A'
```

You can use tesseract's default configs or your own:

```
>>> get_text_from_image(image, config_name='digits')
'13123'
```

Without a config file, you can set config variables using optional keywords:

```
>>> text = get_text_from_image(
        image,
        tessedit_char_whitelist='1'
        tessedit_ocr_engine_mode=1,  #cube mode enum found in Tesseract-OCR docs
        )
'1  11 '
```

2.2 Advanced Functions

tesseractwrapper.**get_tesseract_process**(*commands*, *tesseract_dir_path=u''*, *stdin=-1*, *stdout=-1*, *stderr=-1*)

Popen and return tesseract command line utility.

Opens and returns a tesseract process to the tesseract command line utility. Uses Popen to open a process and pipes to tesseract.

Parameters

- **commands** (*List[str]*) – The command line strings passed into the tesseract binary. Do not include the binary name or path in this variable.
- **tesseract_dir_path** (*Optional[str]*) – The path to the directory with the tesseract binary. Defaults to "", which works if the binary is on the environmental PATH variable.

Returns The open subprocess pipe.

Return type subprocess.Popen

tesseractwrapper.**test_tesseract_path_version**(*tesseract_dir_path=u''*)

Tests that the correct version tesseract is installed and on the path.

Use this function to ensure that either tesseract is on a default or specified path. The function raises ImportErrors if tesseract does not work or is not the right version. The function is silent if everything passes.

Parameters **tesseract_dir_path** (*Optional[str]*) – The path to the directory with the tesseract binary. Defaults to "", which works if the binary is on the environmental PATH variable.

Raises ImportError – If the tesseract requirement is not met.

CHAPTER 3

Advanced Example

In this example we will parse pypi version numbering from images.
We will do some manual image preprocessing using PIL.

You will need to have tesseract on your PATH and already done the `pip install` for both `piltesseract` and `requests`.

```
# This cell is for imports and helper functions.

import copy

# For python 2/3 compatability.
try:
    from StringIO import StringIO as BytesIO
except ImportError:
    from io import BytesIO

from PIL import Image, ImageFilter
from piltesseract import get_text_from_image
import requests


def get_image_from_url(url):
    """Gets an image from a url string.

    Args:
        url (str): The url to the image.

    Returns:
        Image: The image downloaded from the url.

    """

    response = requests.get(url)
    image = Image.open(BytesIO(response.content))
    return image


def scale_image_from_width(image, new_width):
    """Rescales an image based on a new width.

    Args:
```

```
        image (PIL.Image): The image to scale.
        new_width (int): The new width to scale to.

    Returns:
        PIL.Image: The new scaled image.

    """
    if new_width == image.width:
         return copy.copy(image)
    width_percent = new_width / float(image.width)
    new_height = int(image.height * width_percent)
    new_size = (new_width, new_height)
    image = image.resize(new_size, Image.ANTIALIAS)
    return image
```

First, we download the pypi image that contains the version.

```
url = u'https://img.shields.io/pypi/v/piltesseract.png?branch=master'
pypi_image = get_image_from_url(url)
pypi_image
```

Now we crop out the information we do not care about.

```
margin_crop = 1
left = 33
upper = margin_crop
right = pypi_image.width - margin_crop
lower = pypi_image.height - margin_crop
crop_box = (left, upper, right, lower)

version_image = pypi_image.crop(box=crop_box)
version_image
```

If we simply get the text at this point, the result will not be very accurate.
The size is smaller than desired and the white on orange does not help.

```
text = get_text_from_image(version_image)
text
```

```
'van:'
```

Because we know versions are numbers + periods and a "v", we can use a tesseract white list, the results are more accurate.

```
white_list = 'v0123456789.'
text = get_text_from_image(version_image,
                           tessedit_char_whitelist=white_list)
text
```

```
'v002'
```

Although we can do better by manually changing the image. We should scale and smooth the image.

```
width = 100
preprocessed_image = scale_image_from_width(version_image, width)
preprocessed_image = preprocessed_image.filter(ImageFilter.SMOOTH_MORE)
preprocessed_image
```


```
text = get_text_from_image(preprocessed_image)
text
```

```
'v0.0.2'
```

The new result is accurate! We can add on the white list for good measure and reliability.

```
white_list = 'v0123456789.'
text = get_text_from_image(preprocessed_image,
                           tessedit_char_whitelist=white_list)
text
```

```
'v0.0.2'
```

CHAPTER 4

Recipes

4.1 Threading multiple images

Because most of the work is not completed in Python and is done via IO and the tesseract binary, we can utilize threading.

```
from multiprocessing.pool import ThreadPool
from piltesseract import get_text_from_image


thread_pool = ThreadPool(10)


def get_lines_from_images(image_list):
    """Gets text from a list of images.

    Uses threads to speed up the process

    Args:
        image_list (list[Image]): The list of images to use
            ocr on.

    Returns
        list[str]: The text from the images.

    """
    return thread_pool.map(get_text_from_image, image_list)
```

The piltesseract package is a simple Tesseract-OCR command line wrapper.

piltesseract allows quick conversion of PIL Image.Image instances to text using Tesseract-OCR.

> **Warning:** piltesseract is intended to only work with tesseract 3.03+, one awesome feature added in 3.03 is the ability to pipe images via stdin, piltesseract utilizes this feature.

Examples

```
>>> from PIL import Image
>>> from piltesseract import get_text_from_image
>>> image = Image.open('quickfox.png')
```

```
>>> get_text_from_image(image)
'The quick brown fox jumps over the lazy dog'
```

Without a config file, you can set config variables using optional keywords.

```
>>> text = get_text_from_image(
        image,
        tessedit_ocr_engine_mode=1,  # cube mode enum found in Tesseract-OCR docs
        cube_debug_level=1
        )
```

piltesseract.**get_text_from_image**(*image*, *tesseract_dir_path=u''*, *stderr=None*, *psm=3*, *lang=u'eng'*, *tessdata_dir_path=None*, *user_words_path=None*, *user_patterns_path=None*, *config_name=None*, ***config_variables*)

Uses tesseract to get text from an image.

Outside of image, tesseract_dir_path, and stderr, the arguments mirror the official command line's usage. A list of the command line options can be found here: https://tesseract-ocr.googlecode.com/svn/trunk/doc/tesseract.1.html

Parameters

- **image** (*Image.Image or str*) – The image to find text from or a path to that image.
- **tesseract_dir_path** (*Optional[str]*) – The path to the directory with the tesseract binary. Defaults to "", which works if the binary is on the environmental PATH variable.
- **stderr** (*Optional[file]*) – The file like object (implements *write*) the tesseract stderr stream will write to. Defaults to None. You can set it to sys.stdin to see all output easily.
- **psm** (*Optional[int]*) – Page Segmentation Mode. Limits Tesseracts layout analysis (see the Tesseract docs). Default is 3, full analysis.
- **lang** (*Optional[str]*) – The language to use. Default is 'eng' for English.
- **tessdata_dir_path** (*Optional[str]*) – The path to the tessdata directory.
- **user_words_path** (*Optional[str]*) – The path to user words file.
- **user_patterns_path** (*Optional[str]*) – The path to the user patterns file.
- **config_name** (*Optional[str]*) – The name of a config file.
- ****config_variables** – The config variables for tesseract. A list of config variables can be found here: http://www.sk-spell.sk.cx/tesseract-ocr-parameters-in-302-version

Returns The parsed text.

Return type str

Raises subprocess.CalledProcessError – If the tesseract exit status is not a success.

Examples

Examples assume "image" is a picture of the text "ABC123". See piltesseract tests for working code.

```
>>> get_text_from_image(image)
'ABC123'
>>> get_text_from_image(image, psm=10)  #single character psm
'A'
```

You can use tesseract's default configs or your own:

```
>>> get_text_from_image(image, config_name='digits')
'13123'
```

Without a config file, you can set config variables using optional keywords:

```
>>> text = get_text_from_image(
        image,
        tessedit_char_whitelist='1'
        tessedit_ocr_engine_mode=1,  #cube mode enum found in Tesseract-OCR docs
        )
'1  11 '
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

p

t

D

G

P

T