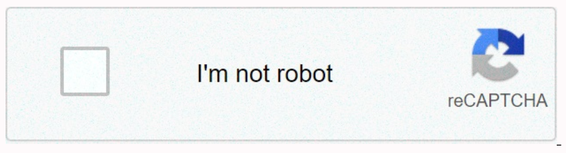


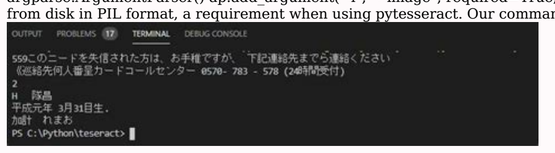
Continue

Python tesseract ocr pdf

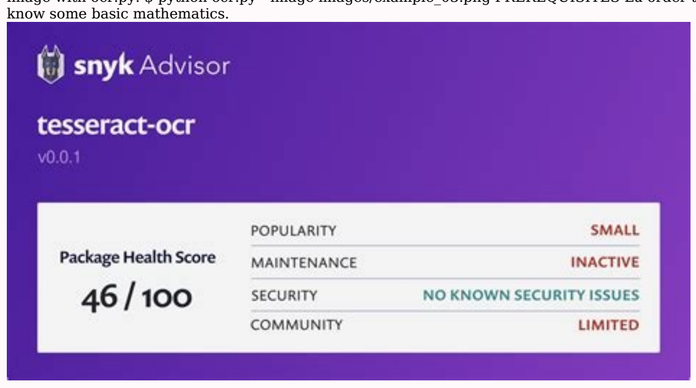
How to use tesseract ocr python. Python tesseract ocr example. Ocr with opencv tesseract and python pdf. Install tesseract ocr python.

Last updated on July 2, 2021.

In last week’s blog post we learned how to install the Tesseract binary for Optical Character Recognition (OCR). We then applied the Tesseract program to test and evaluate the performance of the OCR engine on a very small set of example images. As our results demonstrated, Tesseract works best when there is a (very) clean segmentation of the foreground text from the background. In practice, it can be extremely challenging to guarantee these types of segmentations. Hence, we tend to train domain-specific image classifiers and detectors. Nevertheless, it’s important that we understand how to access Tesseract OCR via the Python programming language in the case that we need to apply OCR to our own projects (provided we can obtain the nice, clean segmentations required by Tesseract). 34956635912.pdf Example projects involving OCR may include building a mobile document scanner that you wish to extract textual information from or perhaps you’re running a service that scans paper medical records and you’re looking to put the information into a HIPA-Compliant database. In the remainder of this blog post, we’ll learn how to install the Tesseract OCR + Python “bindings” followed by writing a simple Python script to call these bindings. By the end of the tutorial, you’ll be able to convert text in an image to a Python string data type. To learn more about using Tesseract and Python together with OCR, just keep reading. Update July 2021: Added section detailing how Tesseract version can have huge impacts on OCR accuracy. This blog post is divided into three parts. fokojalusebu.pdf First, we’ll learn how to install the pytesseract package so that we can access Tesseract via the Python programming language. Next, we’ll develop a simple Python script to load an image, binarize it, and pass it through the Tesseract OCR system. Finally, we’ll test our OCR pipeline on some example images and review the results. To download the source code + example images to this blog post, be sure to use the “Downloads” section below. Installing the Tesseract + Python “bindings” Let’s begin by getting pytesseract installed. To install pytesseract we’ll take advantage of pip . If you’re using a virtual environment (which I highly recommend so that you can separate different projects), use the workon command followed by the appropriate virtual environment name. 78890468484.pdf In this case, our virtualenv is named cv. $ workon cv Next let’s install Pillow, a more Python-friendly port of PIL (a dependency) followed by pytesseract. $ pip install pillow $ pip install pytesseract Note: pytesseract does not provide true Python bindings. bosch tassimo t40 user guide Rather, it simply provides an interface to the tesseract binary. If you take a look at the project on GitHub you’ll see that the library is writing the image to a temporary file on disk followed by calling the tesseract binary on the file and capturing the resulting output. This is definitely a bit hackish, but it gets the job done for us. Let’s move forward by reviewing some code that segments the foreground text from the background and then makes use of our freshly installed pytesseract. compound_sentences_worksheets.pdf Applying OCR with Tesseract and Python Let’s begin by creating a new file named ocr.py: # import the necessary packages from PIL import Image import pytesseract import argparse import cv2 import os # construct the argument parse and parse the arguments ap = argparse.ArgumentParser() ap.add_argument("-i", "--image", required=True, help="path to input image to be OCR'd") ap.add_argument("-p", "--preprocess", type=str, default="thresh", help="type of preprocessing to be done") args = vars(ap.parse_args()) Lines 2-6 handle our imports. The Image class is required so that we can load our input image from disk in PIL format, a requirement when using pytesseract. Our command line arguments are parsed on Lines 9-14. We have two command line arguments: --image: The path to the image we’re sending through the OCR system.



--preprocess: The preprocessing method. This switch is optional and for this tutorial and can accept two values: thresh (threshold) or blur. Next we’ll load the image, binarize it, and write it to disk. # load the example image and convert it to grayscale image = cv2.imread(args["image"]) gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # check to see if we should apply thresholding to preprocess the # image if args["preprocess"] == "thresh": gray = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1] # make a check to see if median blurring should be done to remove # noise elif args["preprocess"] == "blur": gray = cv2.medianBlur(gray, 3) # write the grayscale image to disk as a temporary file so we can # apply OCR to it filename = "{}.png".format(os.getpid()) cv2.imwrite(filename, gray) First, we load --image from disk into memory (Line 17) followed by converting it to grayscale (Line 18). 396806033339.pdf Next, depending on the pre-processing method specified by our command line argument, we will either threshold or blur the image. This is where you would want to add more advanced pre-processing methods (depending on your specific application of OCR) which are beyond the scope of this blog post. The if statement and body on Lines 22-24 perform a threshold in order to segment the foreground from the background.
We do this using both cv2.THRESH_BINARY and cv2.THRESH_OTSU flags. For details on Otsu’s method, see “Otsu’s Binarization” in the official OpenCV documentation. trollskull manor map pdf 2017 free pdf download We will see later in the results section that this thresholding method can be useful to read dark text that is overlaid upon gray shapes. Alternatively, a blurring method may be applied. Lines 28-29 perform a median blur when the --preprocess flag is set to blur. Applying a median blur can help reduce salt and pepper noise, again making it easier for Tesseract to correctly OCR the image. After pre-processing the image, we use os.getpid to derive a temporary image filename based on the process ID of our Python script (Line 33). The final step before using pytesseract for OCR is to write the pre-processed image, gray, to disk saving it with the filename from above (Line 34). We can finally apply OCR to our image using the Tesseract Python “bindings”: # load the image as a PIL/Pillow image, apply OCR, and then delete # the temporary file text = pytesseract.image_to_string(Image.open(filename)) os.remove(filename) print(text) # show the output images cv2.imshow("Image", image) cv2.imshow("Output", gray) cv2.waitKey(0) Using pytesseract.image_to_string on Line 38 we convert the contents of the image into our desired string, text. Notice that we passed a reference to the temporary image file residing on disk. This is followed by some cleanup on Line 39 where we delete the temporary file.
Line 40 is where we print text to the terminal. In your own applications, you may wish to do some additional processing here such as spellchecking for OCR errors or Natural Language Processing rather than simply printing it to the console as we’ve done in this tutorial. Finally, Lines 43 and 44 handle displaying the original image and pre-processed image on the screen in separate windows. The cv2.waitKey(0) on Line 34 indicates that we should wait until a key on the keyboard is pressed before exiting the script. Let’s see our handywork in action. Tesseract OCR and Python results Now that ocr.py has been created, it’s time to apply Python + Tesseract to perform OCR on some example input images. In this section, we will try OCR’ing three sample images using the following process: First, we will run each image through the Tesseract binary as-is. Then we will run each image through ocr.py (which performs pre-processing before sending through Tesseract). Finally, we will compare the results of both of these methods and note any errors.
Our first example is a “noisy” image. This image contains our desired foreground black text on a background that is partly white and partly scattered with artificially generated circular blobs. The blobs act as “distractors” to our simple algorithm. Figure 1: Our first example input for Optical Character Recognition using Python. Using the Tesseract binary, as we learned last week, we can apply OCR to the raw, unprocessed image: $ tesseract images/example_01.png stdout Noisy image to test Tesseract OCR Tesseract performed well with no errors in this case. Now let’s confirm that our newly made script, ocr.py, also works: $ python ocr.py --image images/example_01.png Noisy image to test Tesseract OCR Figure 2: Applying image preprocessing for OCR with Python. As you can see in this screenshot, the thresholded image is very clear and the background has been removed. Our script correctly prints the contents of the image to the console. Next, let’s test Tesseract and our pre-processing script on an image with “salt and pepper” noise in the background: Figure 3: An example input image containing noise. fiasco rpg pdf This image will “confuse” our OCR algorithm, leading to incorrect OCR results. We can see the output of the tesseract binary below: $ tesseract images/example_02.png stdout Detected 32 diacritics " Tessérac't Will Fail With Noisy Backgrounds Unfortunately, Tesseract did not successfully OCR the text in the image. However, by using the blur pre-processing method in ocr.py we can obtain better results: $ python ocr.py --image images/example_02.png --preprocess blur Tesseract Will Fail With Noisy Backgrounds Figure 4: Applying image preprocessing with Python and OpenCV to improve OCR results. Success! Our blur pre-processing step enabled Tesseract to correctly OCR and output our desired text. Finally, let’s try another image, this one with more text: Figure 5: Another example input to our Tesseract + Python OCR system. The above image is a screenshot from the “Prerequisites” section of my book, Practical Python and OpenCV — let’s see how the Tesseract binary handles this image: $ tesseract images/example_03.png stdout PREREQUISITES In order In make the rnosi of this, you will need (a have a little bit of pregrarrmung experience. All examples in this book are in the Python programming language. Familiarity with Pyihon or other scriphng languages is suggesied, but mm required. You'll also need (a know some basic mathematics. This book is handson and example driven: leis of examples and lots of code, so even if your math skills are noi up to par. cumulative frequency step polygon worksheet do noi worry! The examples are very damned and heavily documented (a help yuu follaw along. Followed by testing the image with ocr.py: $ python ocr.py --image images/example_03.png PREREQUISITES Lu order to make the most ol this, you will need to have a little bit ol programming experience. All examples in this book are in the Python programming language. Familiarity with Python or other scripting languages is suggested, but not requixed. You’ll also need to know some basic mathematics.



This book is handson and example driven: lots of examples and lots ol code, so even ii your math skills are not up to par, do not worry! The examples are very detailed and heavily documented to help you tollow along. Figure 6: Applying Optical Character Recognition (OCR) using Python, OpenCV, and Tesseract. Notice misspellings in both outputs including, but not limited to, “In”, “of”, “required”, “programming”, and “follow”. The output for both of these do not match; however, interestingly the pre-processed version has only 8 word errors whereas the non-pre-processed image has 17 word errors (over twice as many errors). Our pre-processing helps even on a clean background! Python + Tesseract did a reasonable job here, but once again we have demonstrated the limitations of the library as an off-the-shelf classifier. We may obtain good or acceptable results with Tesseract for OCR, but the best accuracy will come from training custom character classifiers on specific sets of fonts that appear in actual real-world images. Don’t let the results of Tesseract OCR discourage you — simply manage your expectations and be realistic on Tesseract’s performance. There is no such thing as a true “off-the-shelf” OCR system that will give you perfect results (there are bound to be some errors). Note: If your text is rotated, you may wish to do additional pre-processing as is performed in this previous blog post on correcting text skew. Otherwise, if you’re interested in building a mobile document scanner, you now have a reasonably good OCR system to integrate into it.

Tip: Improve OCR accuracy by upgrading your Tesseract version Be sure to check the Tesseract version you have installed on your machine by using the tesseract -v command: $ tesseract -v tesseract 4.1.1 If you see Tesseract v4 or greater in your output, congrats, you are using the Long Short-Term Memory (LSTM) OCR model which is far more accurate than the previous versions of Tesseract! If you see any version less than v4, then you should upgrade your Tesseract install — using the Tesseract v4 LSTM engine will lead to more accurate OCR results. Course information: 76 total classes • 90 hours of on-demand code walkthrough videos • Last updated: May 2023 ★★★★★ 4.84 (128 Ratings) • 16,000+ Students Enrolled I strongly believe that if you had the right teacher you could master computer vision and deep learning. Do you think learning computer vision and deep learning has to be time-consuming, overwhelming, and complicated? yakkuth_pitiyata_awa_mp3_download.pdf Or has to involve complex mathematics and equations? Or requires a degree in computer science? That’s not the case. All you need to master computer vision and deep learning is for someone to explain things to you in simple, intuitive terms. And that’s exactly what I do. My mission is to change education and how complex Artificial Intelligence topics are taught.
If you're serious about learning computer vision, your next stop should be PyImageSearch University, the most comprehensive computer vision, deep learning, and OpenCV course online today. im hashem lo yivneh bayis psalm Here you’ll learn how to successfully and confidently apply computer vision to your work, research, and projects. Join me in computer vision mastery.
Inside PyImageSearch University you'll find: ✓ 76 courses on essential computer vision, deep learning, and OpenCV topics ✓ 76 Certificates of Completion ✓ 90 hours of on-demand video ✓ Brand new courses released regularly, ensuring you can keep up with state-of-the-art techniques ✓ Pre-configured Jupyter Notebooks in Google Colab ✓ Run all code examples in your web browser — works on Windows, macOS, and Linux (no dev environment configuration required!) ✓ Access to centralized code repos for all 500+ tutorials on PyImageSearch ✓ Easy one-click downloads for code, datasets, pre-trained models, etc. ✓ Access on mobile, laptop, desktop, etc. Click here to join PyImageSearch University Summary In today’s blog post we learned how to apply the Tesseract OCR engine with the Python programming language. This enabled us to apply OCR algorithms from within our Python script. The biggest downside is with the limitations of Tesseract itself. Tesseract works best when there are extremely clean segmentations of the foreground text from the background. Furthermore these segmentations need to be as high resolution (DPI) as possible and the characters in the input image cannot appear “pixelated” after segmentation. If characters do appear pixelated then Tesseract will struggle to correctly recognize the text — we found this out even when applying images captured under ideal conditions (a PDF screenshot). OCR, while no longer a new technology, is still an active area of research in the computer vision literature especially when applying OCR to real-world, unconstrained images. Deep learning and Convolutional Neural Networks (CNNs) are certainly enabling us to obtain higher accuracy, but we are still a long way from seeing “near perfect” OCR systems. Furthermore, as OCR has many applications across many domains, some of the best algorithms used for OCR are commercial and require licensing to be used in your own projects. My primary suggestion to readers when applying OCR to their own projects is to first try Tesseract and if results are undesirable move on to the Google Vision API. If neither Tesseract nor the Google Vision API obtain reasonable accuracy, you might want to reassess your dataset and decide if it’s worth it to train your own custom character classifier — this is especially true if your dataset is noisy and/or contains very specific fonts you wish to detect and recognize. Examples of specific fonts include the digits on a credit card, the account and routing numbers found at the bottom of checks, or stylized text used in graphic design. I hope you are enjoying this series of blog posts on Optical Character Recognition (OCR) with Python and OpenCV! To be notified when new blog posts are published here on PyImageSearch, be sure to enter your email address in the form below! Enter your email address below to get a .zip of the code and a FREE 17-page Resource Guide on Computer Vision, OpenCV, and Deep Learning. Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL! Oftentimes, you would like to make scanned files or image files searchable in PDF, because it is much quicker and more convenient to search keywords with a searchable PDF than in native image format. More importantly, the underlying text would be useful for Natural Language Processing (NLP).
In this article, I’m going to talk about how to turn scanned file(s) into searchable PDF programmatically using Python and Pytesseract. Required Libraries pdf2image: It is a Python module that wraps pdftoppm and pdftocairo to convert a PDF to an image object. The produced output would be a list of image objects. pytesseract: Python-Tesseract is an optical character recognition (OCR) tool developed for Python. It uses an OCR engine (namely, Google’s Tesseract-OCR Engine) to extract text from the image(s) instead of relying on underlying text and structure from PDF. pytesseract has the advantages of extracting text from PDF (such as preserving whitespaces between words) over other Python packages. PyPDF2: It is a Python PDF toolkit, which is capable of splitting, cropping, merging PDF pages and more. io: It allows us to manage the file-related input and output.

Install Libraries pip install pdf2image pip install pytesseract pip install PyPDF2 Download and Install additional software We would need additional software to use the libraries. For pdf2image, we will have to download the poppler for windows users. This software functions similarly to pdftoppm and pdftocairo in a Linux system. For pytesseract, we will need to download and install Tesseract-OCR Engine. Import Libraries import pytesseract from pdf2image import convert_from_path import PyPDF2 import io Initialize pytesseract and pdf2image After you download and install the software, you can add their executable paths into Environment Variables on your computer. Alternatively, you can run the following commands to directly include their paths in the Python program. qasas un nabiyeen arabic pdf part 1 free download poppler_path = '...\pdf2image_poppler\Release-22.01.0-0\poppler-22.01.0\Library\bin' pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe' Convert scanned PDF page(s) to single search PDF Usually, we have multiple scanned PDF pages in a single file. car eats car 3 unblocked We can use the following functions to process all the pages with a for-loop. convert_from_path: it converts all the scanned PDF pages into images. image_to_pdf_or_hocr: it converts an image into a searchable pdf page PdfFileWriter: it creates a pdf writer object for the new PDF file. PdfFileReader: it creates a pdf reader object addPage: it adds a page to pdf writer object images = convert_from_path('Receipt.pdf', poppler_path=poppler_path) pdf_writer = PyPDF2.PdfFileWriter() for image in images: page = pytesseract.image_to_pdf_or_hocr(image, extension='pdf') pdf = PyPDF2.PdfFileReader(io.BytesIO(page)) pdf_writer.addPage(pdf.getPage(0)) # export the searchable PDF to searchable.pdf with open("searchable.pdf", "wb") as f: pdf_writer.write(f) Convert an Image to Searchable PDF If the scanned file is in an image format, such as, tif, png, jpg. The process to convert it into a search PDF file is simpler. PDF = pytesseract.image_to_pdf_or_hocr('Receipt.PNG', extension='pdf') # export to searchable.pdf with open("searchable.pdf", "w+b") as f: f.write(bytearray(PDF)) Convert Multiple Images in the same folder to a Single searchable PDF If you would like to convert a lot of images in the same folder into a single searchable PDF file, you can use os.walk to create a list of paths for all the image files in the same folder, then use the same functions mentioned above to process the images and export into a single searchable PDF file. all_files = [] for (path,dirs,files) in **os.walk**('images_folder'): for file in files: file = os.path.join(path, file) all_files.append(file) pdf_writer = PyPDF2.PdfFileWriter() for file in all_files: page = pytesseract.image_to_pdf_or_hocr(file, extension='pdf') pdf = PyPDF2.PdfFileReader(io.BytesIO(page)) pdf_writer.addPage(pdf.getPage(0)) with open("searchable.pdf", "wb") as f: pdf_writer.write(f) If you would like to continue exploring PDF scraping, please check out my other articles: If you enjoy this article and would like to Buy Me a Coffee, please click here.