Optical Character Recognition is an old, but still challenging problem that involves the detection and recognition of text from unstructured data, including images and PDF documents. It has cool applications in banking, e-commerce and content moderation in social media.

But as with everything topic in data science, there is a huge amount of resources when trying to learn how to solve the OCR task. This is why I am writing this tutorial, which can help you on getting started.

In this article, I am going to show some Python libraries that can allow you to fastly extract text from images without struggling too much. The explanation of the libraries is followed by a practical example. The dataset used is taken from [Kaggle](https://www.kaggle.com/datasets/robikscube/textocr-text-extraction-from-images-dataset?select=annot.csv). To simplify the concepts, I am just using an image of the film Rush.

**Table of contents:**

1. **pytesseract**
2. **EasyOCR**
3. **Keras-OCR**
4. **TrOCR**
5. **docTR**

**1. pytesseract**

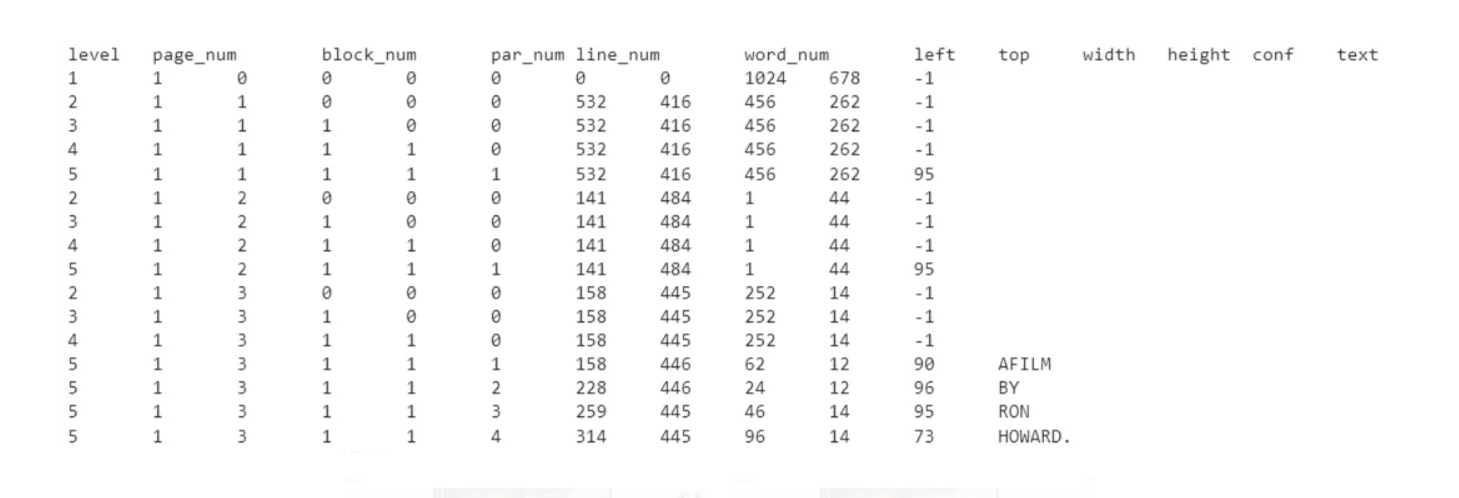
It is one of the most popular Python libraries for optical character recognition. It uses [Google’s Tesseract-OCR Engine](https://github.com/tesseract-ocr/tesseract) to extract text from images. There are multiple languages supported. Check [here](https://tesseract-ocr.github.io/tessdoc/Data-Files-in-different-versions.html) if you want to see if your language is supported. You just need a few lines of code to convert the image into text:

# installation  
!sudo apt install tesseract-ocr  
!pip install pytesseract  
  
import pytesseract  
from pytesseract import Output  
from PIL import Image  
import cv2  
  
img\_path1 = '00b5b88720f35a22.jpg'  
text = pytesseract.image\_to\_string(img\_path1,lang='eng')  
print(text)

We can also try the bounding box coordinates for each item detected from the image.

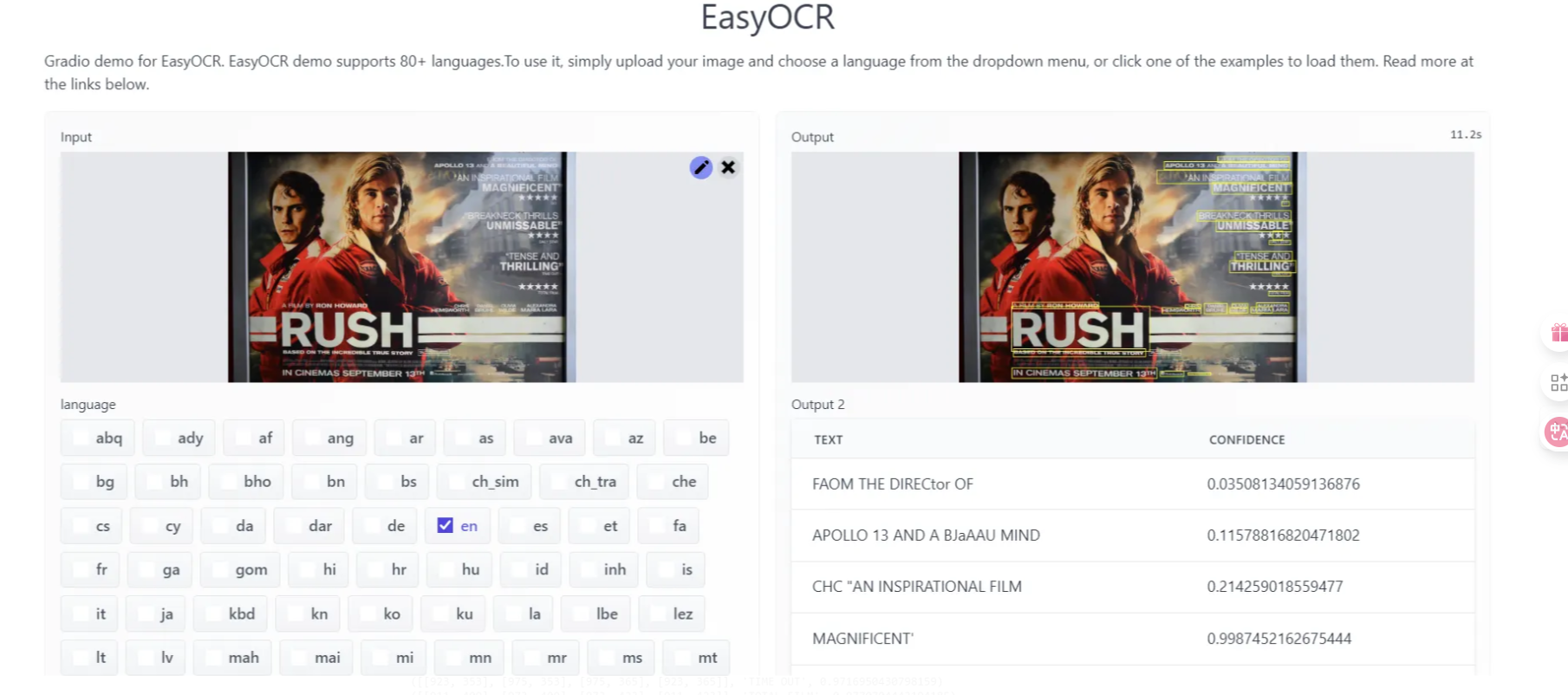
# boxes around character  
print(pytesseract.image\_to\_boxes(img\_path1))

As you can notice, it estimates the bounding box for each character, not each word! In case, we want to extract the box for each word, there is another method that should be used instead of image\_to\_boxes:



The result returned is not so perfect. For example, it interpreted “AFILM” as a unique word. Moreover, it didn’t detect and recognize all the words from the input image.

**2. EasyOCR**

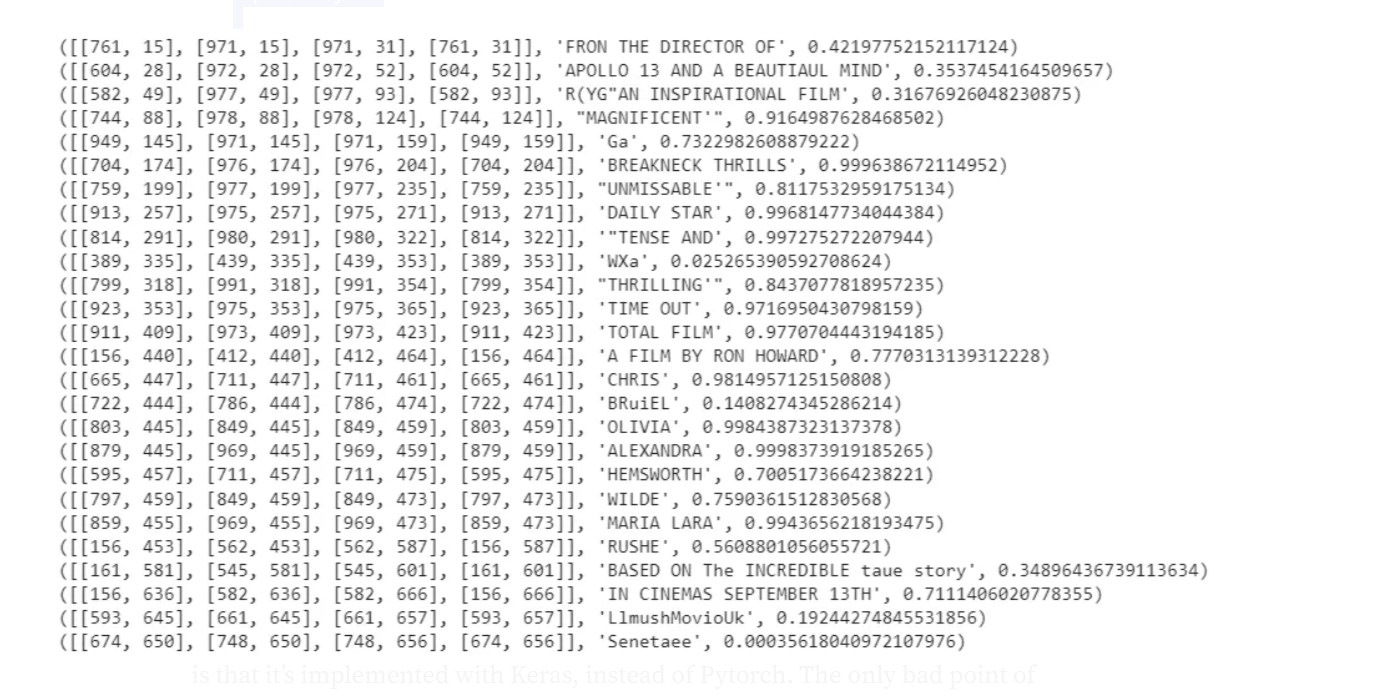


It’s the turn of another open-source Python library: EasyOCR. Similarly to pytesseract, it supports [80+ languages](https://www.jaided.ai/easyocr/). You can try it fastly and easily without writing any code from a web demo. It uses the [CRAFT](https://arxiv.org/abs/1904.01941) algorithm to detect the text and the [CRNN](https://arxiv.org/abs/1507.05717) as recognition model. Moreover, these models are implemented using Pytorch.

If you work on Google Colab, I recommend you set up the GPU, which helps speed up this framework.

These are the following code lines to exploit this tool:

# installation  
!pip install easyocr  
  
import easyocr  
  
reader = easyocr.Reader(['en'])  
extract\_info = reader.readtext(img\_path1)  
  
for el in extract\_info:  
 print(el)



Without any effort, we have detected and recognized the text using EasyOCR. The results are much better compared to pytesseract. For each text detected, we also have the bounding box and the confidence level.

**3. Keras-OCR**

Keras-OCR is another open-source library specialized in optical character recognition. As EasyOCR, it exploits the CRAFT detection model and the CRNN recognition model for solving the task. The difference from EasyOCR is that it’s implemented with Keras, instead of Pytorch. The only bad point of Keras-OCR is that it ignores non-English language. Magically, we can see that we have a much clearer and more precise results.

# installation  
!pip install keras-ocr -q  
import keras\_ocr  
pipeline = keras\_ocr.pipeline.Pipeline()  
extract\_info = pipeline.recognize([img\_path1])  
print(extract\_info[0][0])

**4. TrOCR**

TrOCR is a generative image model, based on transformers, that detect the text from the images. It is composed of an **encoder** and a **decoder**: TrOCR uses a pre-trained image transformer as an encoder and a pre-trained text transformer as a decoder. For additional details, take a look at the [paper](https://arxiv.org/abs/2109.10282). There is also good documentation of the library on [Hugging Face’s platform](https://huggingface.co/docs/transformers/model_doc/trocr).

First, we load the pre-trained models:

# installation  
!pip install transformers  
  
from transformers import TrOCRProcessor, VisionEncoderDecoderModel  
from PIL import Image  
  
model\_version = "microsoft/trocr-base-printed"  
processor = TrOCRProcessor.from\_pretrained(model\_version)  
model = VisionEncoderDecoderModel.from\_pretrained(model\_version)

Before passing the image, we need to resize and normalize it. Once the image has been transformed, we can extract the text using the .generate() method.

image = Image.open(img\_path1).convert("RGB")  
pixel\_values = processor(image, return\_tensors="pt").pixel\_values  
generated\_ids = model.generate(pixel\_values)  
extract\_text = processor.batch\_decode(generated\_ids, skip\_special\_tokens=True)[0]  
print('output: ',extract\_text)  
# output: 2.50

Different from the previous libraries, it returns a meaningless number. Why? TrOCR just includes the recognition model, not the detection model. For solving the OCR task, there is the need to first detect the objects within the image and, then, extract the text from the input. Since it just focuses on the last step, it doesn’t reach good performances.

To make it work well, it would be better to crop specific portions of the image using a bounding box, like this:

crp\_image = image.crop((750, 3.4, 970, 33.94))  
display(crp\_image)

Then, we try to apply again the model:

pixel\_values = processor(crp\_image, return\_tensors="pt").pixel\_values  
generated\_ids = model.generate(pixel\_values)  
extract\_text = processor.batch\_decode(generated\_ids, skip\_special\_tokens=True)[0]  
print(extract\_text)

This is much better! This operation can be repeated for every word/phrase contained within the image.

**5. docTR**

Finally, we are covering the last Python package for text detection and recognition from documents: docTR. It can interpret the document as a PDF or an image and, then, pass it to the two stage-approach. In docTR, there is the text detection model ([DBNet](https://mindee.github.io/doctr/latest/modules/models.html#doctr-models-detection) or [LinkNet](https://mindee.github.io/doctr/latest/modules/models.html#doctr-models-detection)) followed by the [CRNN](https://mindee.github.io/doctr/latest/modules/models.html#doctr-models-recognition) model for text recognition. This library requires both Pytorch and Tensorflow installed since the implementation is done with both these deep learning frameworks.

! pip install python-doctr  
# for TensorFlow  
! pip install "python-doctr[tf]"  
# for PyTorch  
! pip install "python-doctr[torch]"

After, we import the relevant libraries for using docTR and load the model, which is a two-step approach. Indeed, we need to specify the DBNet with ResNet-50 Backbone and CRNN with a VGG-16 backbone for text detection and text recognition:

from doctr.io import DocumentFile  
from doctr.models import ocr\_predictor  
model = ocr\_predictor(det\_arch = 'db\_resnet50',  
 reco\_arch = 'crnn\_vgg16\_bn',  
 pretrained = True  
 )

Then, we can finally read the file, use the pre-trained model and export the output as a nested dictionary:

# read file  
img = DocumentFile.from\_images(img\_path1)  
  
# use pre-trained model  
result = model(img)  
  
# export the result as a nested dict  
extract\_info = result.export()

! pip install python-doctr  
# for TensorFlow  
! pip install "python-doctr[tf]"  
# for PyTorch  
! pip install "python-doctr[torch]"

After, we import the relevant libraries for using docTR and load the model, which is a two-step approach. Indeed, we need to specify the DBNet with ResNet-50 Backbone and CRNN with a VGG-16 backbone for text detection and text recognition:

from doctr.io import DocumentFile  
from doctr.models import ocr\_predictor  
model = ocr\_predictor(det\_arch = 'db\_resnet50',  
 reco\_arch = 'crnn\_vgg16\_bn',  
 pretrained = True  
 )

Then, we can finally read the file, use the pre-trained model and export the output as a nested dictionary:

# read file  
img = DocumentFile.from\_images(img\_path1)  
  
# use pre-trained model  
result = model(img)  
  
# export the result as a nested dict  
extract\_info = result.export()

This is the very long output:

{'pages': [{'page\_idx': 0, 'dimensions': (678, 1024), 'orientation': {'value': None, 'confidence': None},...

For better visualization, it’s better to a double for loop and takes only the information we are interested in:

for obj1 in extract\_info['pages'][0]["blocks"]:  
 for obj2 in obj1["lines"]:  
 for obj3 in obj2["words"]:  
 print("{}: {}".format(obj3["geometry"],obj3["value"]))

Zoom image will be displayed

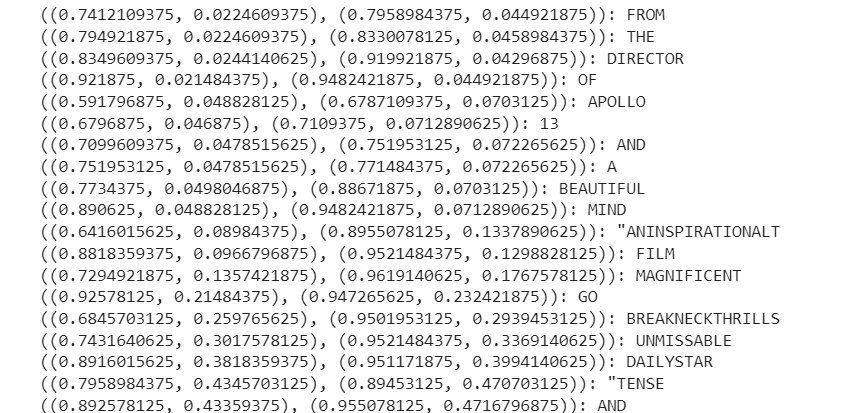


Illustration by Author

That’s great! docTR is another good option to extract valuable information from images or PDFs.