Notes and approaches to OCR

Rohan Alexander, John Tang, Diego Mamanche Castellanos

08 August 2020

Introduction

A crucial aspect of the data science workflow is gathering data. It involves the tools and methods used to collect information in an established systematic fashion and identify the variables being measured. It also describes the methods used to obtain the data. Although the data collection component of research remains the same regardless of the field of study, the methods used vary among those fields of study as they have different interests. Moreover, the diversity of data types adds complexity to the process of gathering data. Structured data is organized in a highly regular manner or a pre-defined data model where the regularities apply to all the data in a particular dataset. Some examples are tables and relations. Semi-structured data contains this same information, but instead of having regular structures applied to all items in the dataset, the data might be interpreted with structural information. It can be supplied as tags e.g. name = "Bob" but also other markers to separate semantic elements and enforce hierarchies of records and fields within the data. Finally, Unstructured data, such as texts or images, holds information with no explicit structured data, such as tags. However, these tags may be assigned using manual or automatic techniques, converting the unstructured data to semi-structured data $(RobertM.Losee, 2005)$[1]. This diversity and complexity of data structures complicate, even more, the data gathering process as each data type may require a specific gathering approach(es) when needed.

The Portable Document Format (PDF) is an example of unstructured data created by the company Adobe in the 90s. It is widely used for documents. According to the $ISO32000-2:2017$[2], the latest edition of the PDF Reference (2.0), PDF enables users to exchange and view electronic documents easily and reliably, independent of the environment in which they were created or the environment in which they are viewed or printed. It is intended to fulfill the following requirements:

- preservation of document fidelity independent of the device, platform, and software,
- merging of content from diverse sources — Web sites, word processing and spreadsheet programs, scanned documents, photos, and graphics — into one self-contained document while maintaining the integrity of all original source documents,
- an extensible metadata model at the document and object level,
- collaborative editing of documents from multiple locations or platforms,
- digital signatures to certify authenticity,
- security and permissions to allow the creator to retain control of the document and associated rights,
- accessibility of content to those with disabilities,
- extraction and reuse of content for use with other file formats and applications, and
- electronic forms to gather and/or represent data within business systems.

To mention some of the features incorporated in this version we have:

- 12.10, "Geospatial features";
- 13.7, "Rich media" annotations;
- 14.7.4, "Namespaces" for tagged PDF;

- 14.9.6, “Pronunciation hints”;
- 14.12, “Document parts”;
- 14.13, “Associated files”;
- Support for PRC (see 13.6, “3D Artwork”);
- Support for UTF-8.

However, these advantages mean that the data in PDFs cannot be used for quantitative statistical analysis. The reason is that it needs measurable and verifiable data. But PDF documents, as it was mentioned previously, can have different objects, some of them non-static objects compressed in the same document. This situation makes it difficult to extract valuable information from the paper.

Sometimes just by copying and pasting the information directly from the PDF is possible when it contains simple texts or regular tables. In that case, there is no barrier in the process of extracting data. But other times it is not as easy if it is an image, a survey form, or complex shapes containing relevant information. Furthermore, there is a linguistic component that adds more complexity. For instance, it is well known that non-Latin characters such as Japanese, Chinese, Arabic, among others, generate several difficulties in different stages of the text mining process. It is because those characters have complicated structures, a huge number of categories, and resemblance among characters, font unevenness, or writing styles. In those cases, the need for reliable tools or methods with the ability to elicit data becomes extremely important.

Optical character recognition (OCR), a process that transforms a bitmapped image of printed or handwritten text into text code, and thereby making it machine-readable, has been widely used for scientists trying to capture the images of characters and texts back in the 50s. First, by mechanical and optical means of rotating disks and photomultiplier flying spot scanner with a cathode ray tube lens, followed by photocells and arrays of them. The OCR process was slow, and one line of characters could be digitized at a time. Nowadays, in OCR, once a printed or handwritten text has been captured optically by a scanner or some other optical means, the digital image goes through the following stages of a computer recognition system $(Cheriet, Mohamed; Kharma, 2007)$[3]:

- The preprocessing stage that enhances the quality of the input image and locates the data of interest.
- The feature extraction stage that captures the distinctive characteristics of the digitized characters for recognition.
- The classification stage that processes the feature vectors to identify the characters and words.

In this paper we review various OCR options for researchers who need to gather data from PDFs in which the text must be extracted to be reused, edited, or reformatted; the text should be available for full-text information retrieval; the text is to be coded in HTML or SGML; the text should be available to adaptive equipment for the visually impaired; the file size is of concern (in terms of storage or bandwidth to transmit); or the resources are available to perform OCR and correct the output.

When they consider a primer on what OCR options exist these days depending on your technical ability and then how they perform for an easy example and then how they perform for a hard example (the Japanese extract) and finally some suggestions for what is needed in the future....

Our paper represents...

The remainder of this paper is structured as follows...

Data

Test Samples

Japanese dictionaries…[Rohan]

For the purpose of this study, one page from the letter a in the Japanese dictionary was taken to assess the selected OCR tools. This image contains 3293 characters. Words in English were counted as one character for this analysis. MOreover, the sample text was divided into four images as it is illustrated in figure XX. These new images were assessed across all OCR tools.

Figure XX

This example is in a vertical form, meaning that it should be read it from right to left and from the top to the bottom. This is important since it is expected to obtain from each OCR tool the correct sequence of characters. FOr instance, from section 1 in figure xx, the first five characters from the first line are [ ]

OCR options

The following list contains some of the OCR APIs and libraries available that will be evaluated in this study:

OCR Tool	Company	Type	Monthly Price Aug/2020	Tech Requierements
Azure Cognitive Services (OCR)	Microsoft	API	Free Web Con- tainer: 5000/Free; S1 Web Con- tainer: 0- 1M/$1.50 per 1,000 trans, 1M- 5M/$1 per 1,000 trans, 5M- 10M/$0.65 per 1,000 trans, 10M- 100M/$0.65 per 1,000 trans, +100M/$0.65 per 1,000 trans	

OCR Tool	Company	Type	Monthly Price Aug/2020	Tech Requierements
Amazon Rekognition	Amazon	API	First 1 million images: $0.001 per image Next 9 million images $0.0008 per image Next 90 million images $0.0006 per image Over 100 million images $0.0004 per image	
PyTesseract (Wrapper Google's OCR Engine)		Python Library	Free	
Standard: $0.002 per trans Kraken (Linux OS)		Python Library	Free	
Google Vision (for Text recognition)	Google	API	1000/Free, 5 mill/$1.5, >5mill/$0.6	
Amazon Textract	Amazon	API	First 1 Million pages: $0.015 - $0.05 per page, Over 1 Million pages: $0.01 - $0.04 per page	

OCR Tool	Company	Type	Monthly Price Aug/2020	Tech Requierements
Kindai-OCR	Result of n2i Project	Python Library	Free	

Azure Corgnitive Services - Computer Vision OCR

Among the Microsoft Azure's portfolio, Azure Cognitive Services offers Computer Vision, an API that processes images and returns information based on the visual features the user is interested in. Those services comprise object detection of an image, visual features tagging, image categorization, Optical Character Recognition (OCR), among *others*[4]. The OCR API processes an image and returns the language of the document, orientation, regions, lines, and finally, the characters.

In python we need several libraries for calling the API.

```
import os
import sys
import requests
# If you are using a Jupyter notebook, uncomment the following line.
#%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from PIL import Image
from io import BytesIO
import matplotlib
from os import path
import ast
```

Then, we need to create headers and parameters necessary for calling the API. The library **requests** is commonly used in python to do that. The method **post** is needed as the API uses the HTTP method POST.

```
#Create the URL for the API call. The endpoint corresponds to the Azure's
# endpoint of the account.
ocr_url = endpoint + "vision/v3.0/ocr"

#Create the header using the Azure's subscription key.
headers = {'Ocp-Apim-Subscription-Key': subscription_key}

#In params language': 'ja' for japanese only. 'unk' means self detection
params = {'language': 'unk', 'detectOrientation': 'true'}
data = {'url': image_url}

#Call and save the response using the method post from the library request
response = requests.post(ocr_url, headers=headers, params=params, json=data)
```

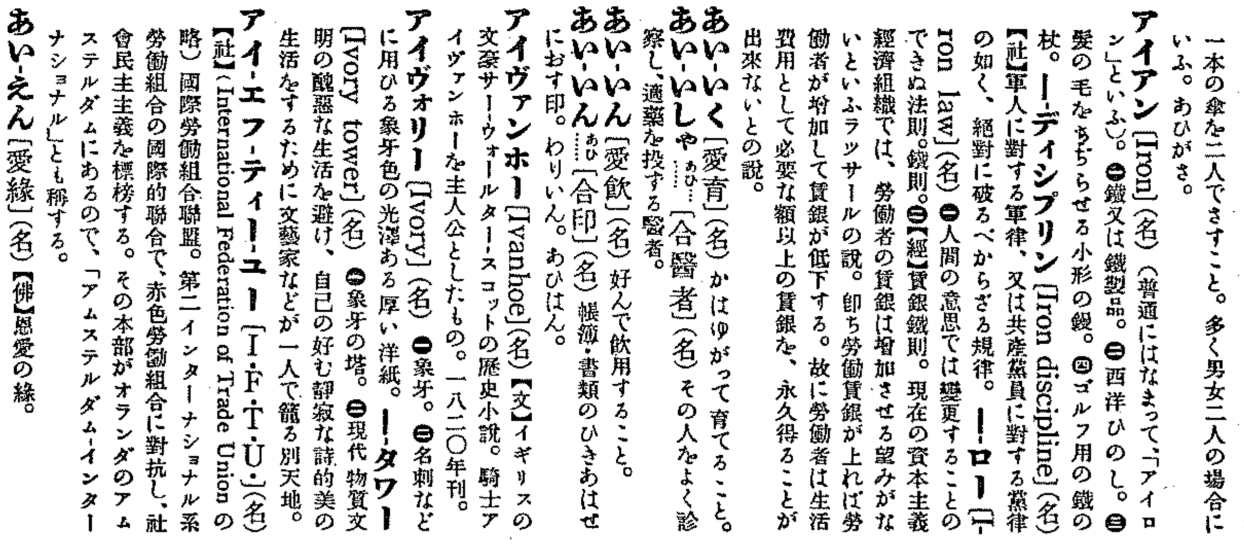
Here we have an example of the response:

```
{'language': 'ja',
 'textAngle': 0.0,
```

```
'orientation': 'Up',
'regions': [{'boundingBox': '31,30,1521,2721',
  'lines': [{'boundingBox': '730,30,44,2718',
    'words': [{'boundingBox': '735,30,34,34', 'text': ' '},
     {'boundingBox': '736,71,34,26', 'text': ' '},
     {'boundingBox': '749,100,6,11', 'text': ' '},
     {'boundingBox': '735,116,34,27', 'text': ' '},
     {'boundingBox': '736,149,33,33', 'text': ' '},
```


Language and Orientation

Figure 1 presents a sample that corresponds to the section1 of the image containing Japanese text, but it also contains English text in some fragments of the text. The API offers the possibility to do self-detection, which in this case is helpful, but also the opportunity to specify the language in advance. Moreover, the orientation can also be self-detected or not.



一本の傘を二人でさすこと。多く男女二人の場合にいふ。あひがさ。

アイアン [Iron] (名) (普通にはなまって「アイロン」といふ)。㊀鐵又は鐵製品。㊁西洋ひのし。㊂髪の毛をちぢらせる小形の鏝。㊃ゴルフ用の鐵の杖。**|-ディシプリン** [Iron discipline] (名) 【社】軍人に對する軍律、又は共産黨員に對する黨律の如く、絶對に破るべからざる規律。**|-ロー** [Iron law] (名) ㊀人間の意思では變更することのできぬ法則。鐵則。㊁【經】賃銀鐵則。現在の資本主義經濟組織では、勞働者の賃銀は增加させる望みがないといふラッサールの說。卽ち勞働賃銀が上れば勞働者が增加して賃銀が低下する。故に勞働者は生活費用として必要な額以上の賃銀を、永久得ることが出來ないとの說。

あい-いく [愛育] (名) かはゆがって育てること。

あい-いしゃ あひ… [合醫者] (名) その人をよく診察し、適藥を投する醫者。

あい-いん [愛飮] (名) 好んで飮用すること。

あい-いん あひ… [合印] (名) 帳簿・書類のひきあはせにおす印。わりいん。あひはん。

アイヴァンホー [Ivanhoe] (名) 【文】イギリスの文豪サー・ウォールター・スコットの歷史小說。騎士アイヴァンホーを主人公としたもの。一八二〇年刊。

アイヴォリー [Ivory] (名) ㊀象牙。㊁名刺などに用ひる象牙色の光澤ある厚い洋紙。**|-タワー** [Ivory tower] (名) ㊀象牙の塔。㊁現代物質文明の醜惡な生活を避け、自己の好む靜寂な詩的美の生活をするために文藝家などが一人で籠る別天地。

アイ・エフ・ティー・ユー [I・F・T・U・] (名) 【社】(International Federation of Trade Unionの略) 國際勞働組合聯盟。第二インターナショナル系勞働組合の國際的聯合で、赤色勞働組合に對抗し、社會民主主義を標榜する。その本部がオランダのアムステルダムにあるので、「アムステルダム・インターナショナル」とも稱する。

あい-えん [愛緣] (名) 【佛】戀愛の緣。

Figure 1

Regions

The blue line figure 2 denotes the segment from the image that includes the characters. In this example, there is only one blue line because the document is only text. In other cases, it may contain images requiring more than one region.

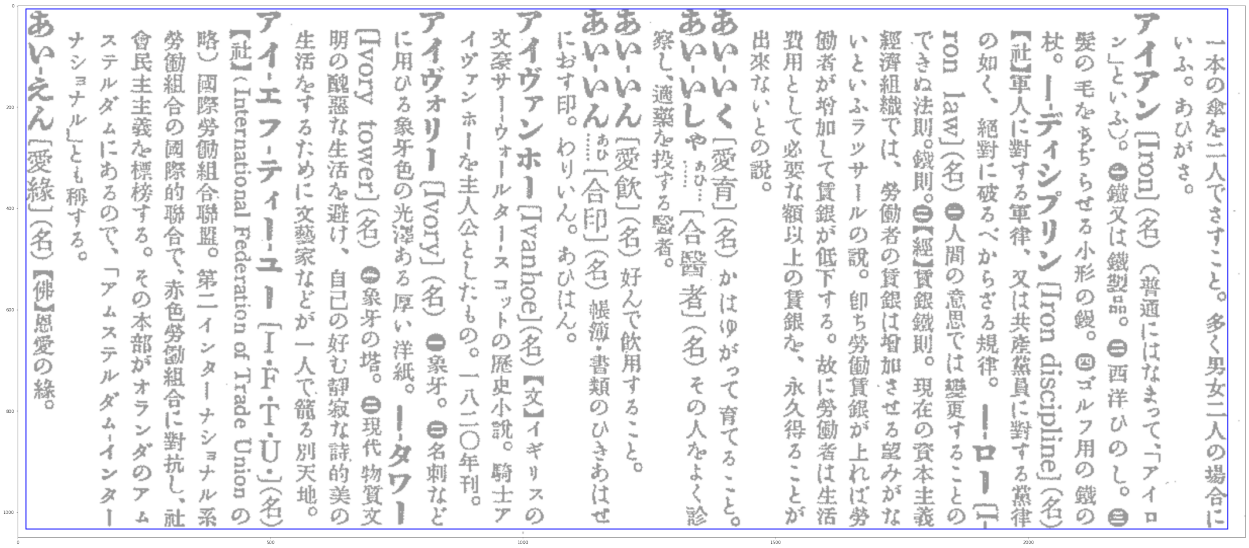


Figure 2

Lines

Once the region is defined, the API provides in figure 3 (red rectangles) the subsets of characters of the region called Lines. From the example, we can see that some characters are not included in any of the lines. Those characters are not recognized by the API.

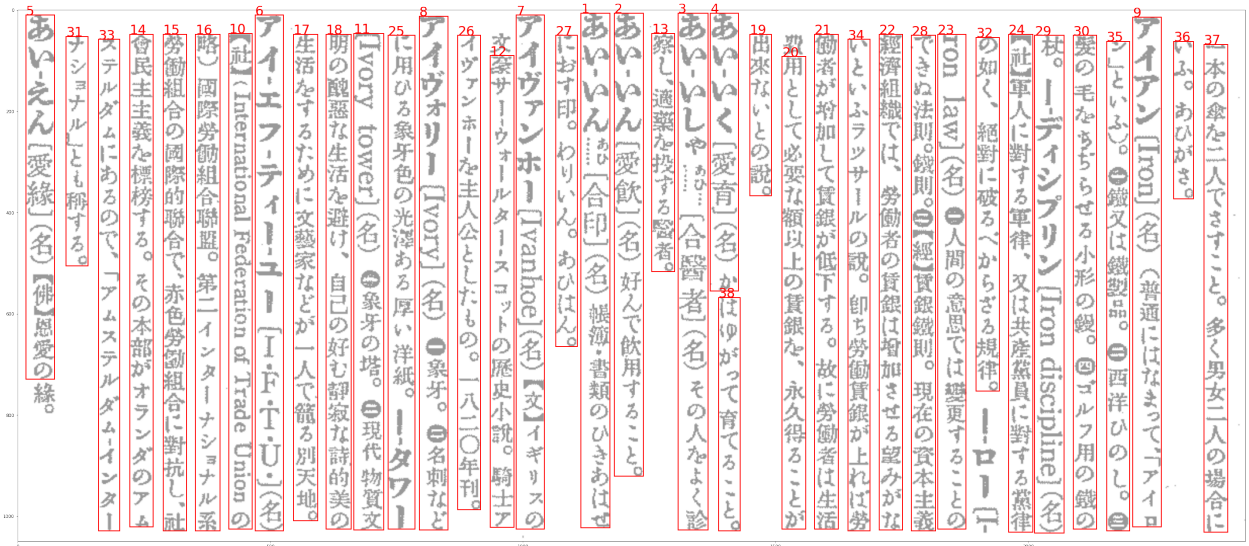


Figure 3

The Character Boundaries

After the lines are identified, the API provides in figure 4 each recognized character bounding boxes. Those without a yellow square were not recognized by the API in this example.

Figure 4

Characters

Once the bounding boxes are located, each character takes its corresponding place, as it is shown in figure 5.

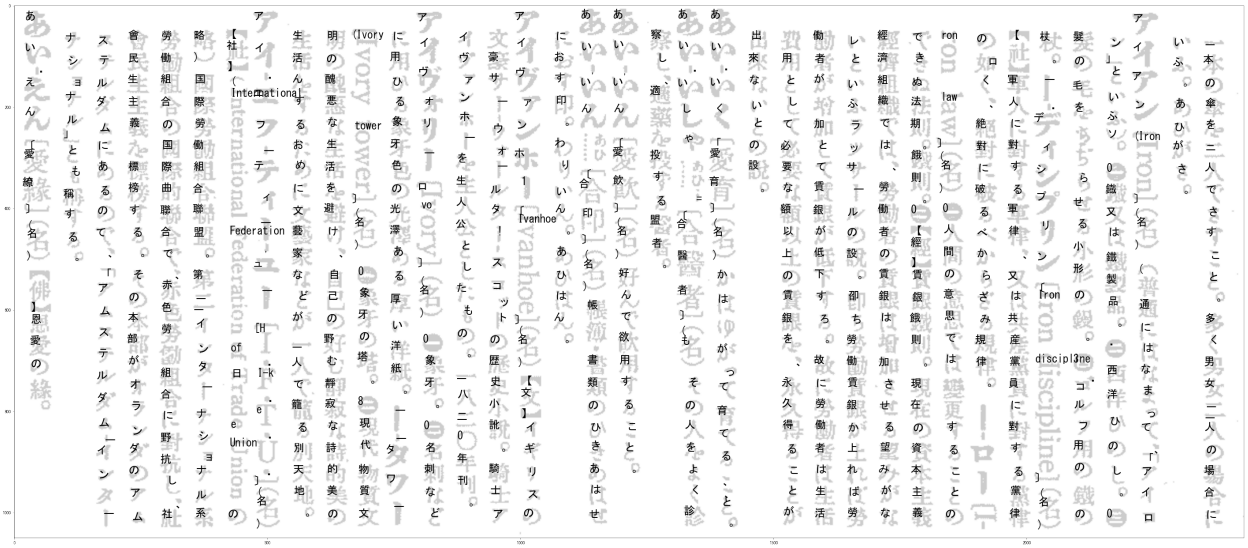


Figure 5

Advantages and Disadvantages

Advantages The API is easy to use after reading the documentation. It offers an example of how to use it in different programming languages. It also offers the possibility to self-detect languages and the orientation of the document. The response is clear, and extract the information from the JSON file is not complicated.

Disadvantages Even though the API offers the possibility to self-detect the language, It changes the order in which the characters should be in the output. In figures 6 we can see that the first line provided by the self-detection approach is located in the middle. This situation does not allow a correct interpretation of the

text because it does not follow the reading pattern. In this case, it is a vertical orientation, which means we have to read it from right to left.

Figure 6

When the parameter language is stated as Japanese since the beginning, as we can see in figure 7, it recognizes the first line correctly.

Figure 7

In general, self-detection and the case with the language parameter as the Japanese do not generate the boundaries correctly. In Figures 6 and 7, we can observe that both cross the line, which is not correct for in this document. There is a line delimiting the boundaries of the text, but it has been ignored by both API calls. This situation makes it difficult for researchers to organize the information correctly as the order of the characters matter.

Google Vision OCR

Google Vision API can integrate vision detection features, including image labeling, face and landmark detection, optical character recognition (OCR), and tagging of explicit content. The text recognition process detects text in images and recognizes the text contained therein. Once detected, the recognizer then determines the actual text in each block and segments it into lines and words.

To evaluate this OCR option, we need the following python libraries.

```
from os import path
from PIL import Image
from PIL import ImageFont
from PIL import ImageDraw
from PIL import ImageEnhance
import base64 #Convert an image into base64 form for API requests
import requests #API request - GET/POST
import json
import ast
```

Same as the Azure API, we need to create the headers and parameters necessary for calling the API. We also need the python library **requests** to make the API call. The method **post** is needed as the API uses the HTTP method POST.

The following example illustrates how to state the headers and parameter, but also create two functions. The first one, called encode_image converts an image into base64, an encoding process designed to carry data stored in binary formats across channels that only reliably support text content. It can embed image files or other binary assets inside textual assets. The second one makes an HTTP POST request that calls Google's Vision API.

```
#Create the URL for the API call. The endpoint corresponds to the Azure's
# endpoint of the account.
url = "https://vision.googleapis.com/v1/images:annotate"

querystring = {"key": google_vision_api_key }
headers = {'Content-Type': "application/json"}

def encode_image(image_path):
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read())

def image_request(image_path):

    payload = '{\"requests\":[{\"image\":{\"content\":\"'+
    encode_image(image_path).decode('utf-8')+
    '"},\"features\":[{\"type\":\"TEXT_DETECTION\"}]}]}'

    response = requests.request("POST", url, data=payload, headers=headers, params=querystring)
```

```
    return ast.literal_eval(response.text)
```

This is an example of the response:

```
{'responses': [{'textAnnotations': [{'locale': 'ja',
     'description': '-]\n\n              -  [ ]()           ]()    ............ ) \n-[~Q)
     ]()\n 0    \n   p®( ]()     \n\nV\n
          \n      \n',
     'boundingPoly': {'vertices': [{'x': 26, 'y': 21},
       {'x': 1556, 'y': 21},
       {'x': 1556, 'y': 2748},
       {'x': 26, 'y': 2748}]}},
    {'description': '-',
     'boundingPoly': {'vertices': [{'x': 1227, 'y': 686},
       {'x': 1227, 'y': 677},
       {'x': 1262, 'y': 677},
       {'x': 1262, 'y': 686}]}},


         \n'}}]}
          {'description': ' ',
     'boundingPoly': {'vertices': [{'x': 785, 'y': 880},
       {'x': 814, 'y': 880},
       {'x': 814, 'y': 914},
       {'x': 785, 'y': 914}]}},
    {'description': '(',
     'boundingPoly': {'vertices': [{'x': 775, 'y': 916},
       {'x': 814, 'y': 916},
       {'x': 814, 'y': 940},
       {'x': 775, 'y': 940}]}},
    ...],
```

From the previous example, we can observe that the response returns a list of nested dictionaries. The first nested dictionary called **textAnnotations** consists of a list of nested dictionaries. Each one of those has a description (character) and boundingPoly (boundaries of the characters).

BoundingPoly

Each boundingPoly consist of vertices in which the symbol is located.

Region Te first object in the nested dictionary contains all identified characters in the image as well as the boundaries that comprises the location of those characters. This object can resemble the idea of the region provided in the Azure's API.

Figure 8 shows the region identified by the API.

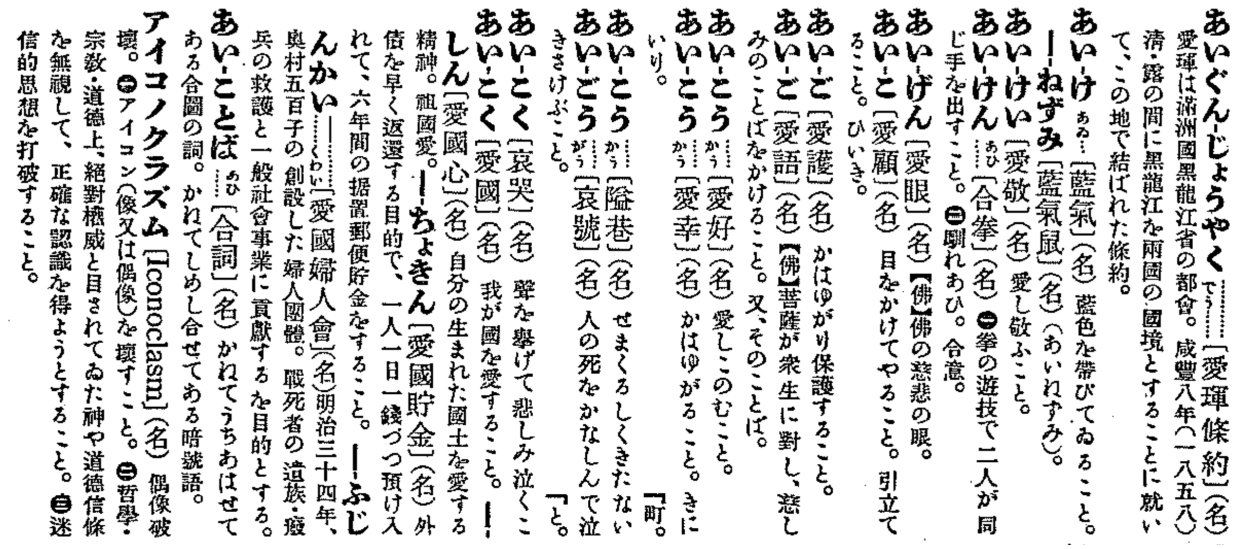


Figure 8

The Character Boundaries The remaining objects enclose the boundaries of the characters. As it is shown in figure 9, each boundingPoly node can include one or more characters.

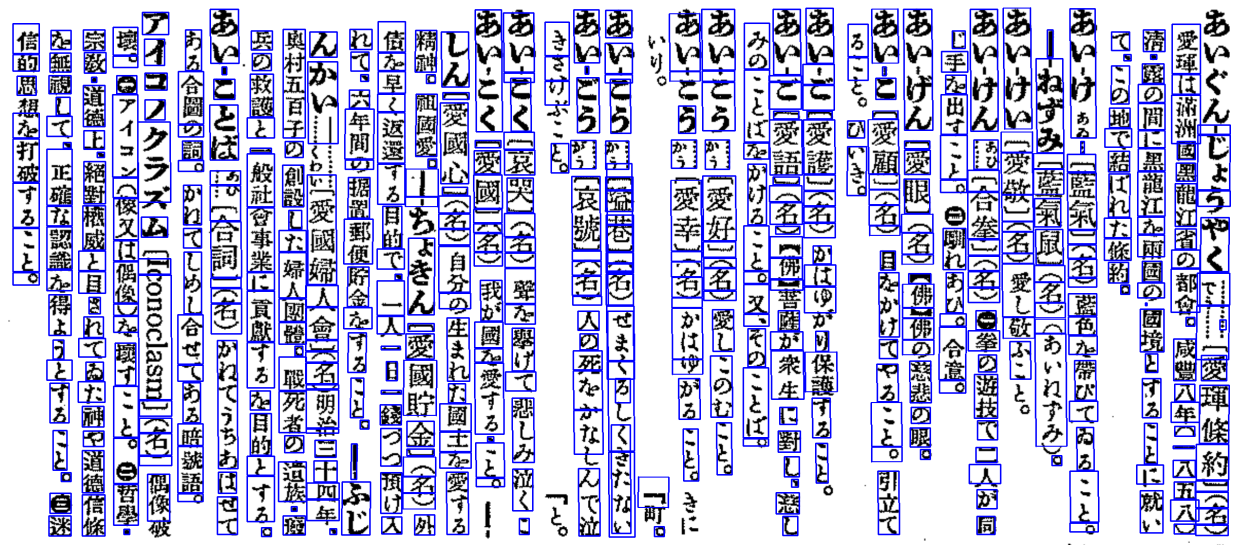


Figure 9

Characters

Once the bounding boxes are located, each character takes its corresponding place, as it is shown in figure 10. **Pending find a way to plot the text in vertical form to make it readable.**

Figure 10

languageHints

Google Vision API offers the advantage of including in the request language hints. It consists of a list of languages the user knows in advance the image contains. For this study, we run two versions, one without

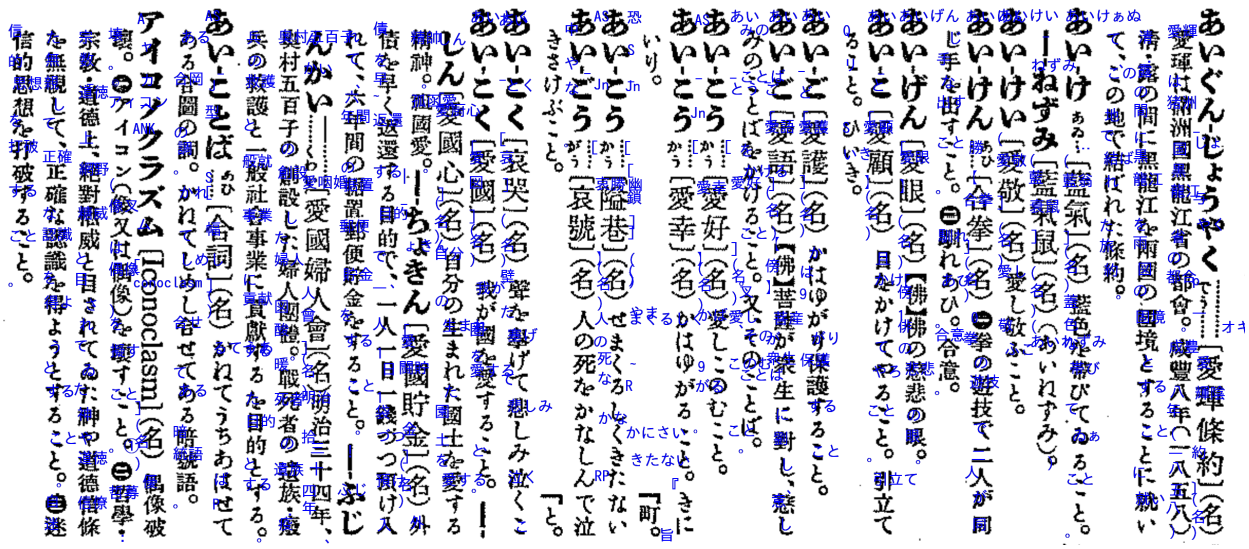


Figure 1: image

hints and the other one with a list of two hints, such as Japanese and English, as the image contains some words in English.

Advantages and Disadvantages

Advantages Same as the Azure API, this is also easy to use after reading the documentation. It also offers several examples in different programming languages using the Google libraries for Python. It gives the possibility to include languages in advanced as hints, and identifies the orientation automatically. The response contains an object with all identified characters in the correct order.

Moreover, the API gives the opportunity to send the image in base64 format or using an internal (google storage) or external URL.

Disadvantages

PyTesseract - Python Library

Python-Tesseract is an optical character recognition (OCR) library for python. It is a wrapper for Google's Tesseract-OCR Engine. PyTesseract can read all image types such as png, jpeg, gif, tiff, BMP, among others. It is widely used to process everything from scanned documents. This inbuilt library of python performs the OCR techniques on the image to produce the digitized output text in a readable and editable form.

To use this library, we need to install Tesseract. For this study, we use version 4. IT is necessary to call the .exe file to use Tesseract with PyTesseract. See the following example.

```
#Call .exe location for Windows
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
```

Along with the PyTesseract library, the following libraries are used.

```
from PIL import Image
from PIL import ImageOps
import pytesseract
from os import path
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
```

PyTesseract offers several methods with different outputs when processing an image to extract text. According to Pypi.org, the uses of those methods are:

- **image_to_string:** Returns the result of a Tesseract OCR run on the image to string
- **image_to_boxes** Returns result containing recognized characters and their box boundaries
- **image_to_data:** Returns result containing box boundaries, confidences, and other information. Requires Tesseract 3.05+. For more information, please check the Tesseract TSV documentation
- **image_to_osd:** Returns result containing information about orientation and script detection.
- **image_to_alto_xml:** Returns result in the form of Tesseract's ALTO XML format.
- **run_and_get_output:** Returns the raw output from Tesseract OCR. Gives a bit more control over the parameters that are sent to tesseract.

For this analysis, we use the method image_to_data as this offers the boundaries and the recognized characters in a handy form so we can understand the process. Furthermore, in PyTesseract, it is needed to specify the language in advance. Since PyTesseract is a wrapper of Google's Tesseract-OCR Engine, Tesseract-OCR must have installed all required languages. In this case, it is Japanese, which is not included in its out-of-the-box version. To include a new language, adding the **.traineddata** file into the Tesseract-OCR/tessdata folder is sufficient. However, since the text in the image is vertical Japanese. It is necessary to include two files: jpn.traineddata and jpn_vert.traineddata. Once the files are located correctly, the following code extracts the text from the image.

```
img1 = Image.open(r'~\inputs\00-sample-pages_1.png')
image_to_data = pytesseract.image_to_data(img1, lang = 'jpn_vert')
```

The output from the previous code converted into a pandas dataframe is shown in figure 11.

```
]:  1 result.head(8)
```

]:

	level	page_num	block_num	par_num	line_num	word_num	left	top	width	height	conf	text
0	1	1	0	0	0	0	0	0	1588	2776	-1	NaN
1	2	1	1	0	0	0	898	30	654	657	-1	NaN
2	3	1	1	1	0	0	898	30	654	657	-1	NaN
3	4	1	1	1	1	0	1522	70	30	616	-1	NaN
4	5	1	1	1	1	1	1522	70	28	44	13	本
5	5	1	1	1	1	2	1525	123	26	35	92	の
6	5	1	1	1	1	3	1525	170	25	20	87	傘
7	5	1	1	1	1	4	1525	199	26	20	78	を

Figure 11

From figure 11, we can observe that the output in the column **level** generates five levels. Those are pages, blocks, paragraphs, lines, and words (characters). Each one is subsets of the previous one.

Page

For this example, the page resembles the concept of region, which is the largest area of the image enclosing all recognized characters. Figure 12 illustrates the page area.

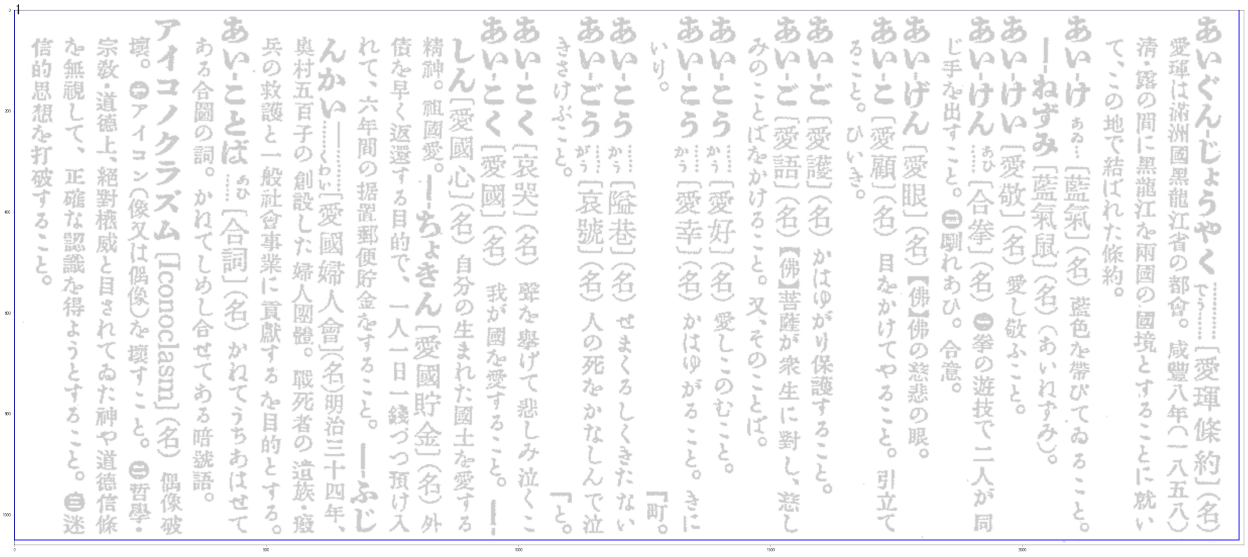


Figure 12

Blocks

The blocks are subsets of the page area that consist of small groups of recognized characters. In figure 13, we can see that most of the blocks are well ordered. In terms of correctness of the boundaries, we can see that some of them overlap several characters that are not recognized by the library.

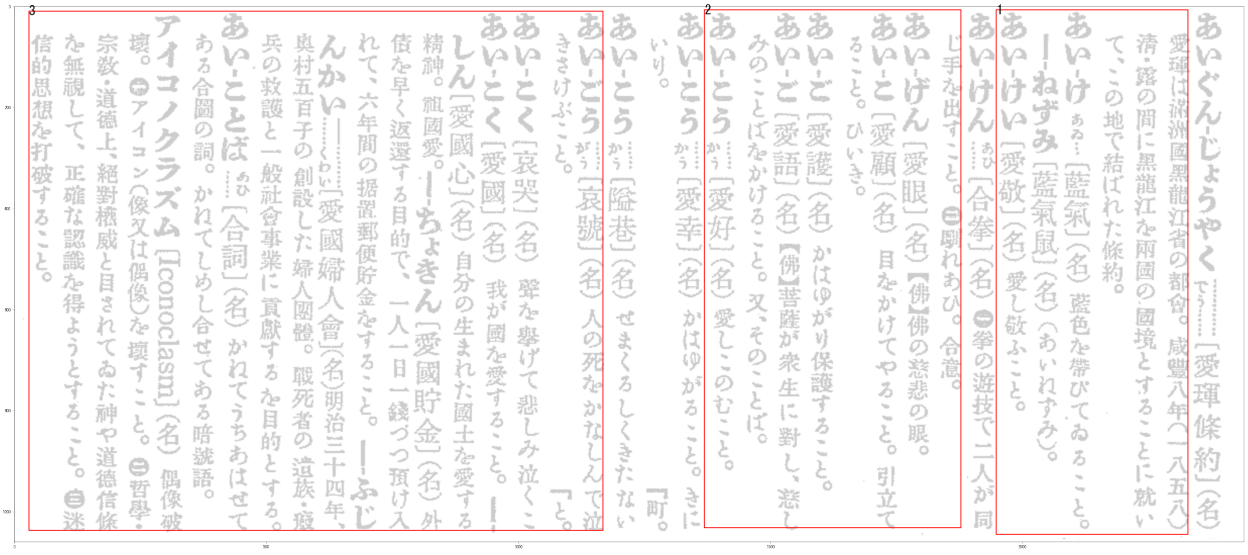


Figure 13

Paragraphs

Similar to the blocks, the paragraphs are subsets of blocks. These follow the same idea of subareas containing a set of characters. In this case, for example, we can look at paragraphs five and six, which are subsets of block five. It is interesting to see how the only paragraph (21) of block 12 is a smaller area compared to the block. This situation happens in several other blocks. See figure 14 for a better understanding.

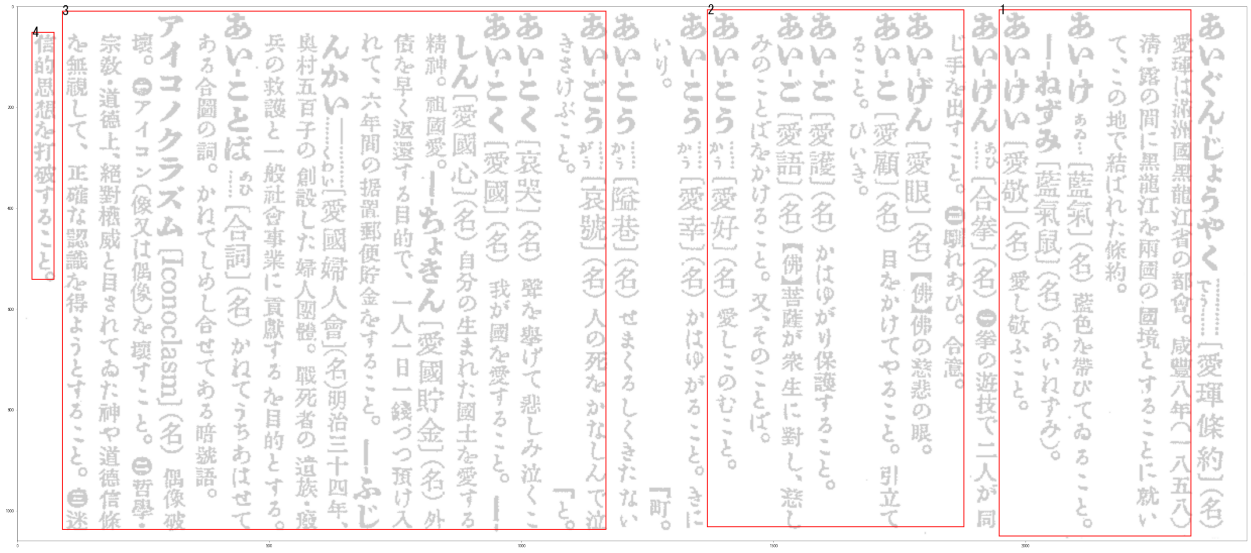


Figure 14

Lines

Once the paragraphs are defined, the library provides in figure 15, the last subsets of characters called Lines. From the example, we can see that most of the characters are located in one of the lines. However, several others are not. Some of those cases are curious since there is no clear reason why those characters are not included. For instance, between the lines 65 and 66, two clear lines are not recognized by the library.

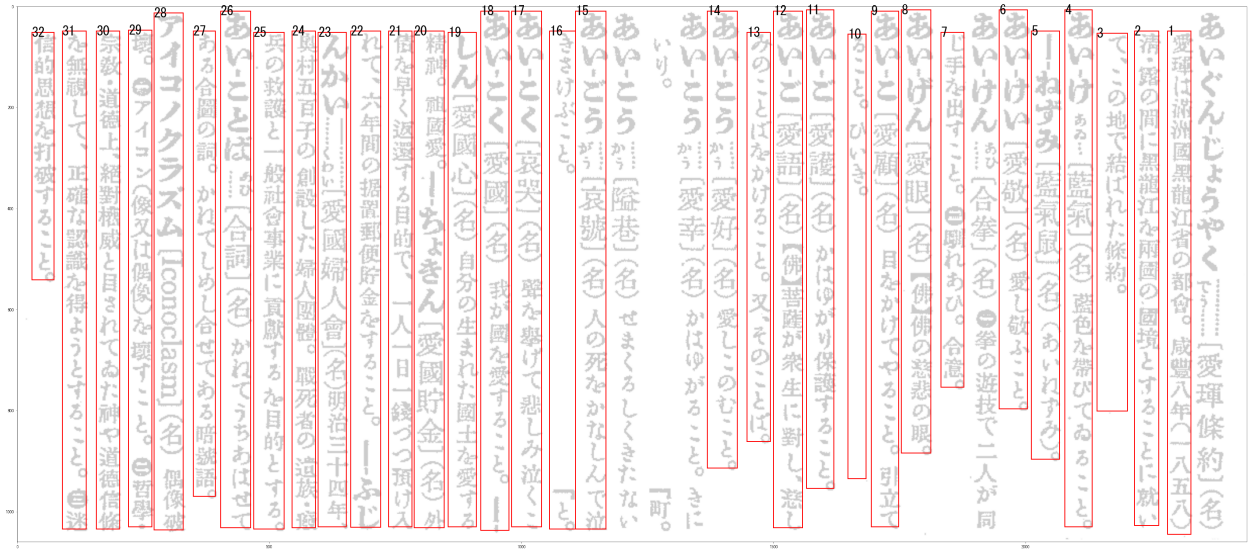


Figure 15

Character Boundaries

These boundaries enclose the characters. Figure 16 shows that a character boundary can include one or more characters. However, some character boundaries overlap with other ones. Moreover, the same as the previous areas, several characters have been skipped with no apparent reason.

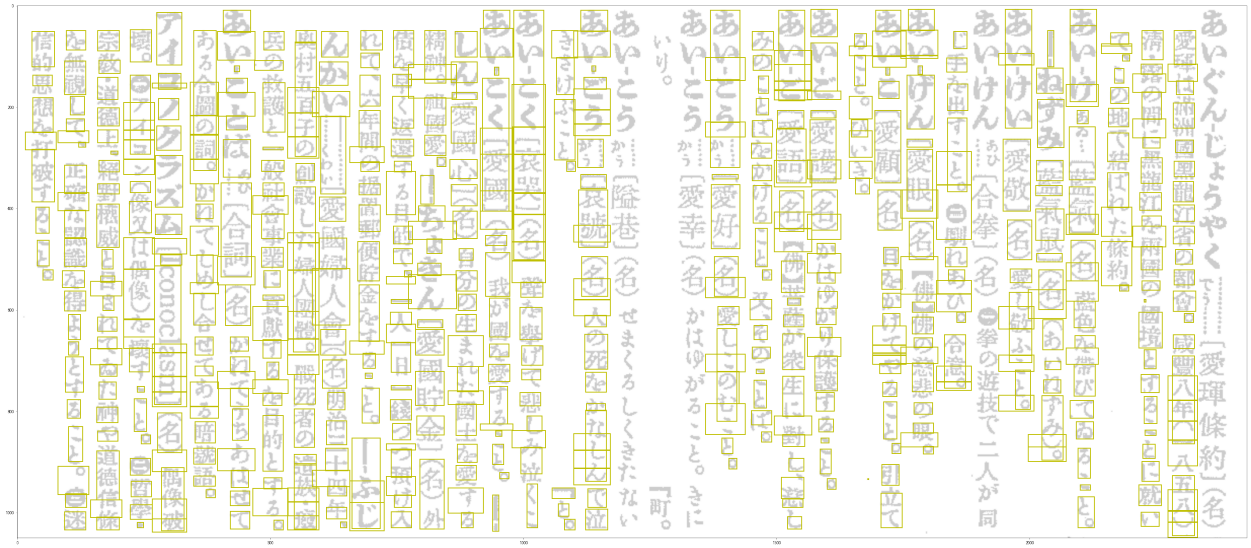


Figure 16

Characters

In terms of character-level recognition, we can observe that many characters are not recognized. Interestingly those characters are in a correct block, paragraph, line, and character boundary, but it was skipped. For example, in the first line and the first character boundary of this line, there are two words, but one of them is not recognized. Figure 17 illustrates this case.

Figure 17

Advantages and Disadvantages

Advantages The most important advantage is that this is a free library. It is also a wrapper from Google's Tesseract Engine. Moreover, the way it presents the covered areas are very detailed. It offers blocks, paragraphs, lines, boundaries, and finally the character level.

Disadvantages

Kindai-OCR

Kraken - Python Library

Kraken is a python library created by Benjamin Kiessling from Université PSL in France and Leipzig University. This library is a combination of a fork (independent development built on an existing one) from the Ocropus package and CLSTM neural network library. Its purpose was to improve accuracy and performance rates compared with Ocropus. It has been tested with languages such as Arabic, Persian, Syriac Polytonic Greek, Latin, among other. However, there is no information about japanese-trained models.

To assess the performance of this library, it is necessary to train a model for japanese texts. To do that, Kraken offers a function called **train**. This function expects a series of images that according to Kraken´s website, these images must be high quality scans, preferably color or grayscale, and at least 300dpi. Images in PDF format are not allowed. Each image must be accompanied with a text file with extension .gt.txt. This file will contain the character(s) that corresponds to the image with the same name. For instance in figure xx we see the image file is called **sec1__line1.png**, the pair-wise text element for this image is **sec__line.gt.txt**.

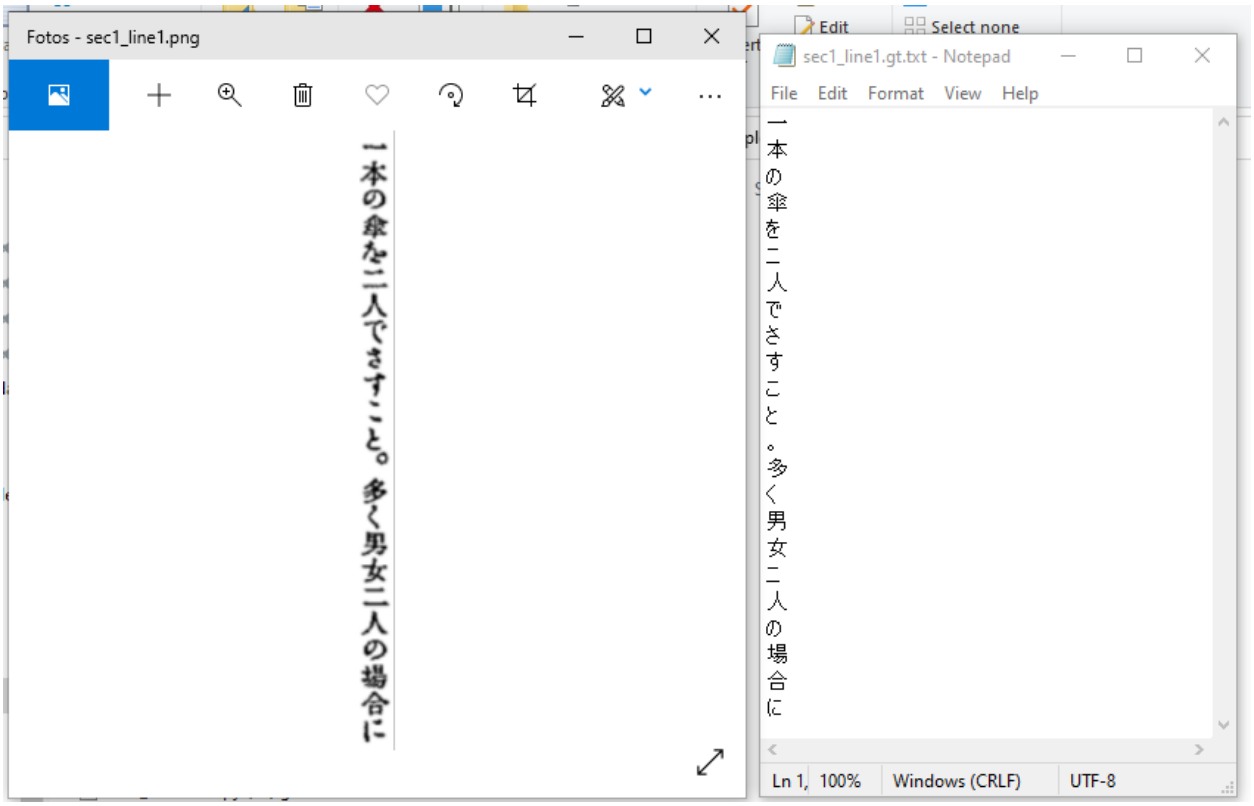


Figure 18

After the training stage, Kraken analyses the layout and extracts text lines from an input image for later processing by the recognition. A step by step process is provided by Kraken´s website (http://kraken.re/index.html) to recognize text from an image using the model we want.

Training process challenges

The training process itself is very friendly. However, the data preparation is the most important part as it generates the samples we feed into the model. Other studies pointed out that for languages such as Arabic, at least 800 lines are required to train a descent model. For the Japanese text of this study we trained 36 lines in which several characters appear more than once. The result after the training process was not good, indicating that the need of greater sample data might be needed which is not in the scope of this study.

Other limitations

In order to use this library it is required to have either Linux or Mac. In this study, we use Google Colab which is offers a Linux environment to assess the library.

Amazon Textract

According to Amazon's website, Amazon Textract can detect Latin-script characters from the standard English alphabet and ASCII symbols.

https://aws.amazon.com/textract/faqs/

Amazon Rekognition

According to Amazon's website, Amazon Rekognition supports text in most Latin scripts and numbers. Text detection recognizes up to 50 sequences of characters per the image or video frame and lists them as words and lines.

https://aws.amazon.com/rekognition/faqs/

Evaluation

For each option, we rank them based on two scales: results and difficulty. We need to work out a definition for each of these.

Diego's comment: From what I learned so far, it could be easiness (an API is more straightforward than a library), number of characters correctly recognized, flexibility (the opposite of easiness), and cost. Moreover, some APIs such as Google's groups more than one character, whereas Azure returns only one character per boundary.

In terms of flexibility…

Sequence of the characters…

Output given…xml, json, flat file, csv…

NUmber of characters well recognized…(Accuracy)

The main aspect of the end product will be a graph with two axis, and dots for where each service is positioned, coloured or faceted by the test.

Discussion

Appendix

References

1: https://www.researchgate.net/publication/267465115_An_Overview_and_Applications_of_Optical_Character_Recognition

2: https://books-scholarsportal-info.myaccess.library.utoronto.ca/en/read?id=/ebooks/ebooks2/wiley/2011-12-13/1/9780470176535

3:

4: https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home

5: https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/quickstarts/python-print-text

6: https://cloud.google.com/vision/docs/ocr

7: https://pypi.org/project/pytesseract/

8: https://github.com/tesseract-ocr/tessdata

9: https://github.com/UB-Mannheim/tesseract/wiki