Using Optical Character Recognition in an Applied Research Workflow

Rohan Alexander* John Tang† Diego Mamanche Castellanos‡

14 December 2020

Abstract

We discuss the use of Optical Character Recognition (OCR) as a initial step in an applied research workflow. OCR is a way of converting images, for instance in a PDF file, into text and numbers that can then be analysed. We review a variety of methods and their relative merit. The choice between these approaches turns on the scale of the OCR required, the level of funding that is available, and the user's technical skills. We provide a hueristic to guide this decision and worked examples. Finally, we place this within the context of an illustrative applied research workflow detailing its relationship with other aspects of the workflow.

1 Introduction

A crucial step in the data analysis workflow is gathering data. This step involves tools and methods to collect raw data in an systematic way. In this paper we are interested in gathering data from Portable Document Format (PDF) files, in particular, when these PDFs were created by scans, and hence must be processed before the data they contain can be prepared, cleaned, and analysed. Such PDF files are generally unstructured data, meaning that the data are not tagged nor organised in a regular way.[1] While tags in the form of a text replacement for each character image could be assigned manually, typically a statistical model is used for reasons of efficiency and reproducibility, and this process is called Optical Character Recognition (OCR). We evaluate several pre-trained commercial OCR options accessed through proprietary APIs: 1) Abbyy, 2) Amazon Rekognition, 3) Amazon Textract, 4) Azure Cognitive Services, and 5) Google Vision; additionally several pre-trained and trainable OCR options that a user runs themselves: 6) Kindai OCR, 7) Kraken, and finally 8) Tesseract. We find… [**TBD**].

PDF files were developed in 1993 by the technology company Adobe and are useful for documents because they are meant to display in a consistent way independent of the environment that created them or the environment in which they are being viewed. A PDF viewed on an iPhone should look the same as on a Android phone, as on a Linux desktop. One feature of PDFs is that they can include a variety of objects, for instance, text, photos, figures, etc. However, this variety can limit the capacity of PDFs to be used directly as data inputs for statistical analysis. The data first needs to be extracted from the PDF.

Sometimes it is possible to copy and paste the data from the PDF. This is more likely when the PDF only contains simple text or regular tables. In particular, if the PDF has been created by an application such as

*University of Toronto, rohan.alexander@utoronto.ca.

†University of Melbourne.

‡University of Toronto.

[1]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. 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 (Losee 2006). 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.

Table 1: OCR options

OCR Tool	Company	Type	Price (December 2020)	Tech requirem
Abbyy	Abbyy	Proprietory API	NA	NA
Amazon Rekognition	Amazon	Proprietory API	NA	NA
Amazon Textract	Amazon	Proprietory API	NA	NA
Azure Cognitive Services	Microsoft	Proprietory API	NA	NA
Google Vision	Google	Proprietory API	NA	NA
Kindai OCR	n2i Project	Open source pre-trained OCR model	NA	NA
Kraken	Open source	Open source OCR engine	NA	NA
Tesseract	Open source	Open source OCR engine	NA	NA

Microsoft Word, or another document or form creation system, then often the text data can be extracted in this way because they are actually stored as text within the PDF. But it is not as easy if the text has been stored as an image which is then part of the PDF. This may be the case for PDFs produced through scans or photos of physical documents, and some older document preparation software. Furthermore, there is a linguistic component that adds more complexity (Kanagarathinam et al. 2019). For instance, non-Latin characters such as Japanese, Chinese, Arabic, among others, generate several difficulties in different stages of the text mining process. Not only do these languages have complicated structures and a large number of categories (for instance, compare the 26 characters in the English alphabet with the thousands of characters in Japanese kanji). But often many characters are similar and fonts may have a large effect. In those cases, the need for reliable approaches to replace an image containing many characters with text of those characters becomes important.

Optical character recognition (OCR) is a process that transforms a image of text into actual text. Although there may not be much difference to a human reading a PDF before and after OCR, the main difference is that the PDF is now machine-readable (Cheriet et al. 2007). OCR has been used to parse images of characters since the 1950s, initially using physical processes. Such approaches were understandably slow and have been replaced by a computer-based recognition system involving the following steps (Cheriet et al. 2007, 4):

- Pre-processing to both: increase the quality of the input image, and to focus on the data of interest.
- Feature extraction to identify the distinctive features of the characters to be recognized.
- The actual classification stage, in which those features are used to identify the characters.

In this paper we review various OCR options for data analysts who need to gather their data from PDFs where the characters of interest for analysis must first be transformed from image to text. The paper goes through the process of using various options to do this OCR and the tradeoffs that must be made. We consider a few different options in terms of the image that is to be transformed. We start with a one page extract of modern English, a one-page table of numbers, one page of modern Japanese, one page from an English book from the 1800s, and a page from a Japanese dictionary from the early 1900s.

We find… **[TBD]**.

Our paper is similar to Rychlik et al. (2020) which focuses on Pashto, Farsi, and Traditional Chinese. Additional useful surveys include Lombardi and Marinai (2020) and Reul et al. (2019).

The remainder of this paper is structured as follows… **[TBD]**. Finally we make some suggestions for what is needed in the future.

2 Background

In this section we introduce the OCR options that we consider, their essential features, and cost. There are **eight [UPDATE]** OCR options evaluated in this study (Table 1).

The gold-standard in terms of quality is a manual approach. For instance, a team of research assistants who were paid to inspect each page, inputting the correct element and checking the inputs of others will likely provide the best recognition. While this option is only possible for smaller datasets and is not reproducible, sharing high-quality datasets created in this way is important for evaluating the output of automated approaches.

Abbyy

Amazon Rekognition is a proprietary

Amazon Textract

Azure Cognitive Services

Google Vision

Kindai OCR (Le et al. 2019) - https://github.com/ducanh841988/Kindai-OCR - is a pre-trained model that uses an attention based encoder-decoder approach specifically designed for Japanese historical documents. It is part of a larger research program on Japanese character recognition including Horiuchi and Kato (2009) and Nguyen et al. (2017). This research program is complementary to research on historical Hanja (Korean) documents, for instance Kim et al. (2004), and historical Khmer (Cambodia) documents, for instance Valy, Verleysen, and Chhun (2020). Le, Clanuwat, and Kitamoto (2019) provides broader consideration of issues with the recognition of historical Japanese, and related, characters including 'noised, damaged characters, and background', as well as the possibility that characters may be 'written in cursive and are connected'. The Center for Open Data in the Humanities - http://codh.rois.ac.jp - provides a large number of training datasets, and runs the n2i Project from which Kindai came out of.

OCRopus - https://github.com/ocropus/ocropy - is an open-source OCR engine initially developed in 2007 as 'a Google-sponsored project' (Breuel 2007). Kraken - https://github.com/mittagessen/kraken - is an 'extensively rewritten fork of the OCRopus system' by Benjamin Kiessling (Kiessling 2019). Martínek, Lenc, and Král (2020) use Kraken to create 'an efficient domain-dependent OCR system that focuses on historical German documents by picking the best tools and approaches available'. Similarly, Romanov et al. (2017) use Kraken to achieve 'accuracy rates for classical Arabic-script texts in the high nineties'.

Tesseract - https://github.com/tesseract-ocr/tesseract - is an open-source OCR engine initially developed 'at HP between 1985 and 1995, shelved for 10 years, open-sources in 2006 and now developed mostly at Google' (Smith 2013). It has had a variety of versions, with the latest being Tesseract 4 which added a 'neural net (LSTM) based OCR engine' ("Tesseract Manual Page" 2018). Tesseract is a widely used OCR engine with a variety of implementations available including the default usage of command line, Python through the pytesseract package [**ADD CITE**] and R through the tesseract package (Ooms 2019) for which a useful starter example is Rodrigues (2019).

3 Data

3.1 Modern English

TBD.

3.2 Table of numbers

TBD.

3.3 Modern Japanese

TBD.

3.4 English book from the 1800s

TBD.

3.5 Japanese dictionaries from the early 1900s

[JOHN: Add a para about the origin of this dictionary.]

Here we focus on a page focused on 'a', which is in hiragana or in katakana Figure 1.

This page has several features of interest. Firstly, it is in vertical form and should be read from right to left. This means that the first five characters that we expect from an OCR output are: . The page is divided into four rows of content. An OCR workflow will need to take this into account during the pre-processing stage. In total, this page contains 3,293 characters, and while a majority of them are Japanese, there are still 99 English characters. Finally, there are also important nuances in terms of punctuation. For instance, certain types of brackets - [**ADD AN EXAMPLE**] - contain the origin of the word.

4 Results

4.1 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, boundaries, 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}
```

Figure 1: A page from an early 1900s Japanese dictionary.

```
#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': ' '},
```


4.1.1 Language and Orientation

Figure xx presents a sample that corresponds to the section 1 of the image containing Japanese text, but it also includes 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.

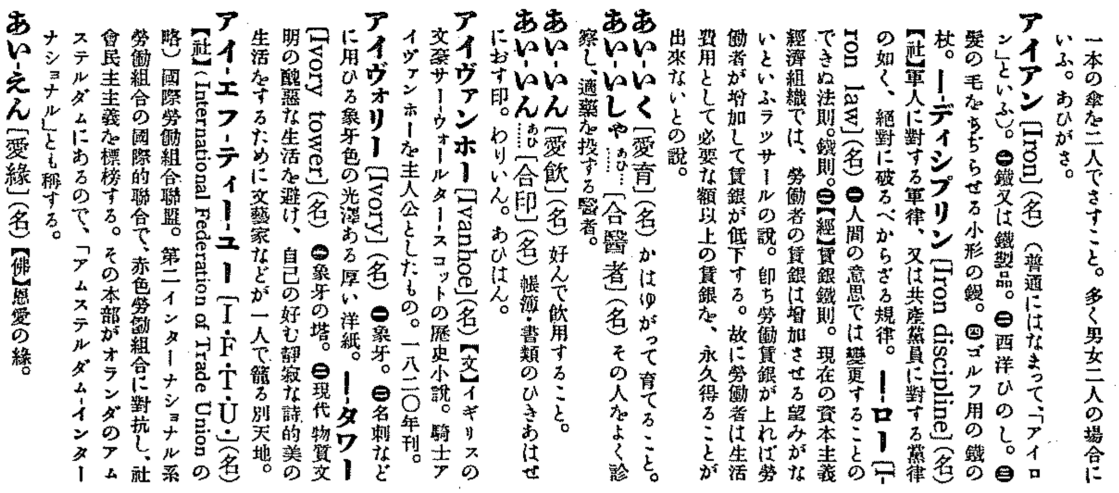

Figure 2: UPDATE CAPTION

4.1.2 Regions

The blue line figure xx 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.

4.1.3 Lines

Once the region is defined, the API provides in figure xx (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.

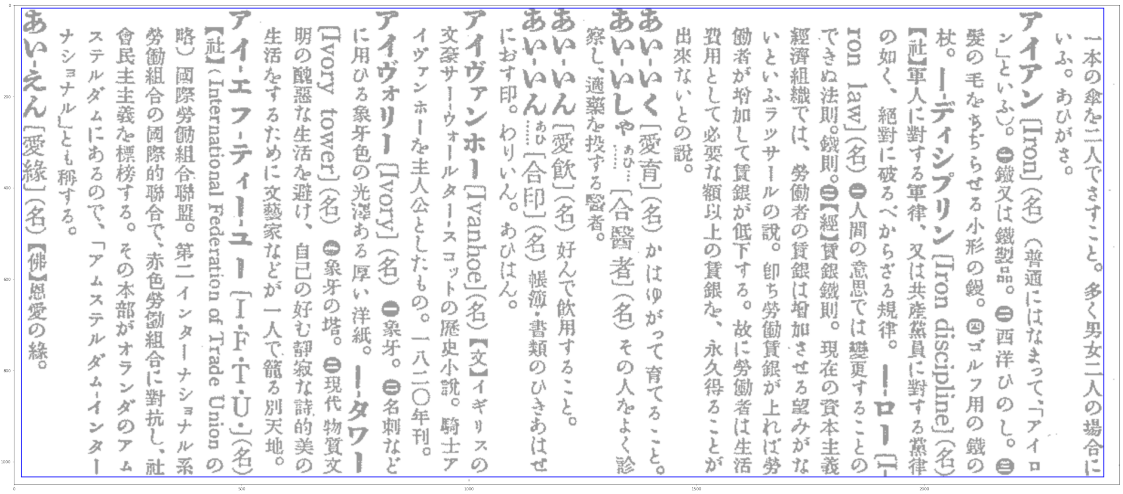

Figure 3: UPDATE CAPTION

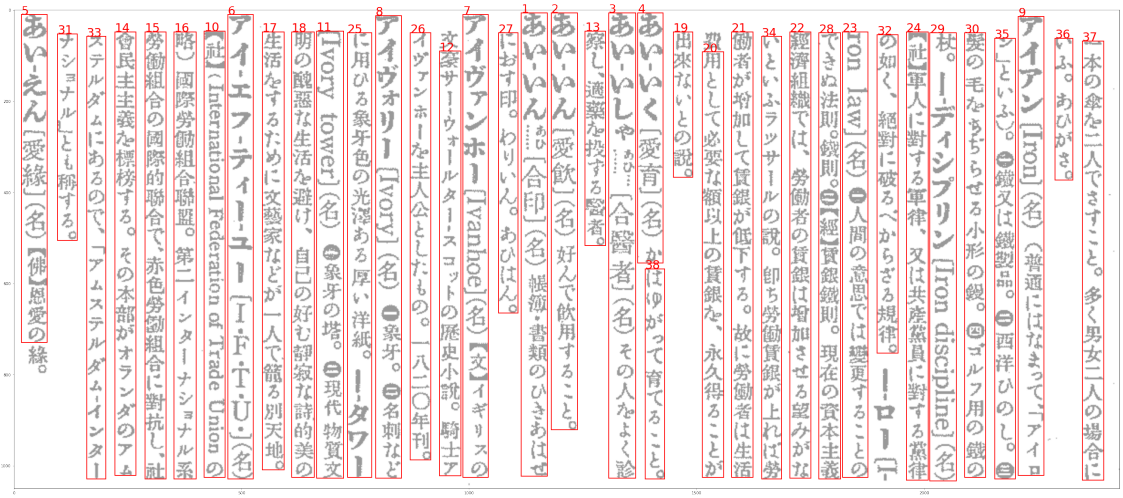

Figure 4: UPDATE CAPTION

4.1.4 The Character Boundaries

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

Figure 5: UPDATE CAPTION

4.1.5 Characters

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

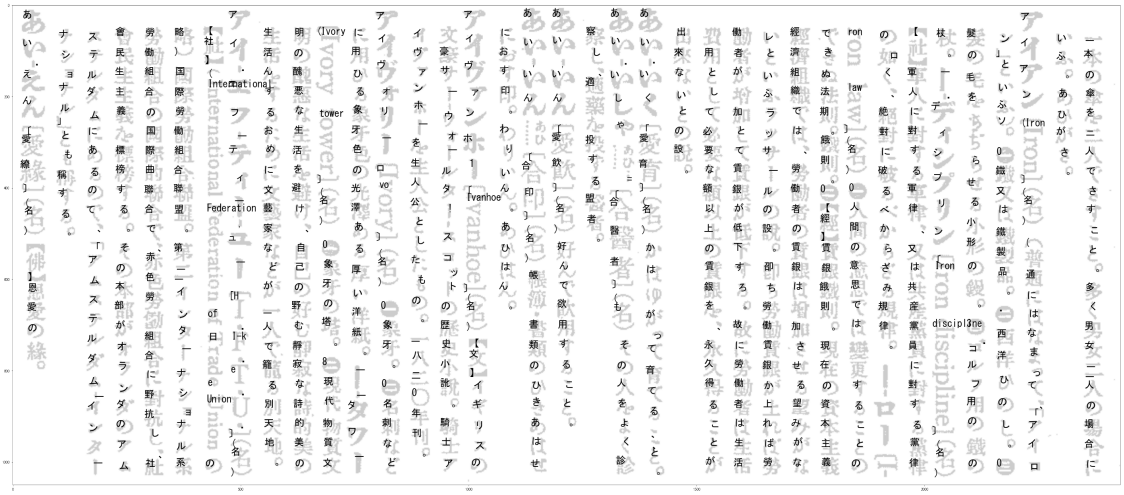

Figure 6: UPDATE CAPTION

4.1.6 Advantages and Disadvantages

4.1.6.1 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.

4.1.6.2 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 figure xx 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.

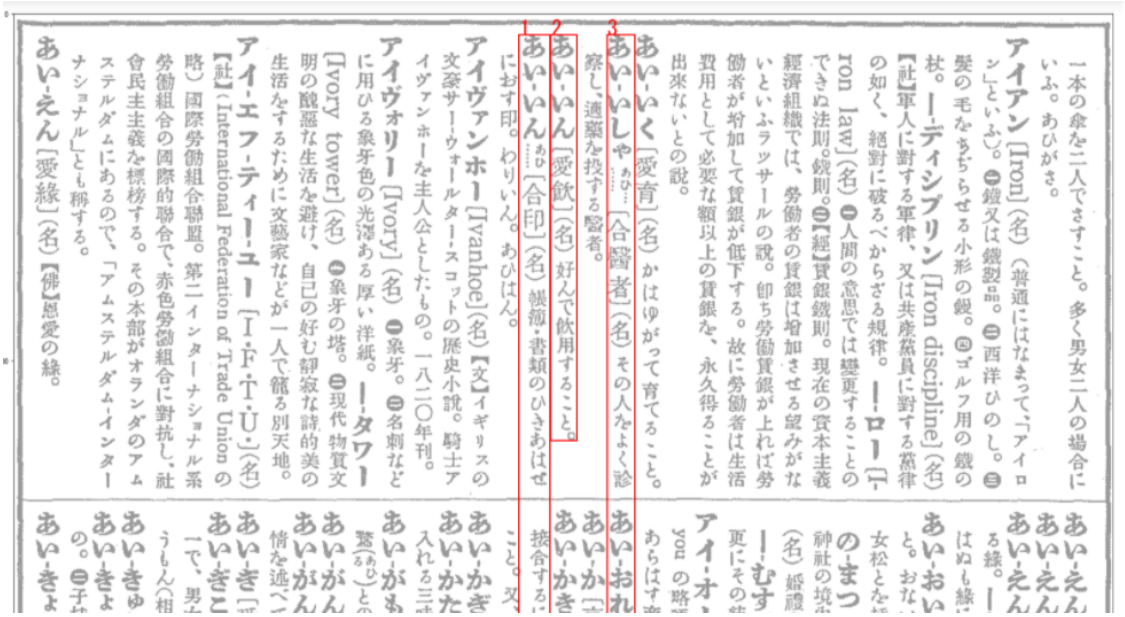

Figure 7: UPDATE CAPTION

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

4.2 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 format, an encoding process designed to

Figure 8: UPDATE CAPTION

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 dictionaries. Each one of those has a description (character) and boundingPoly (boundaries of the characters).

4.2.1 BoundingPoly

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

4.2.1.1 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 xx shows the region identified by the API.

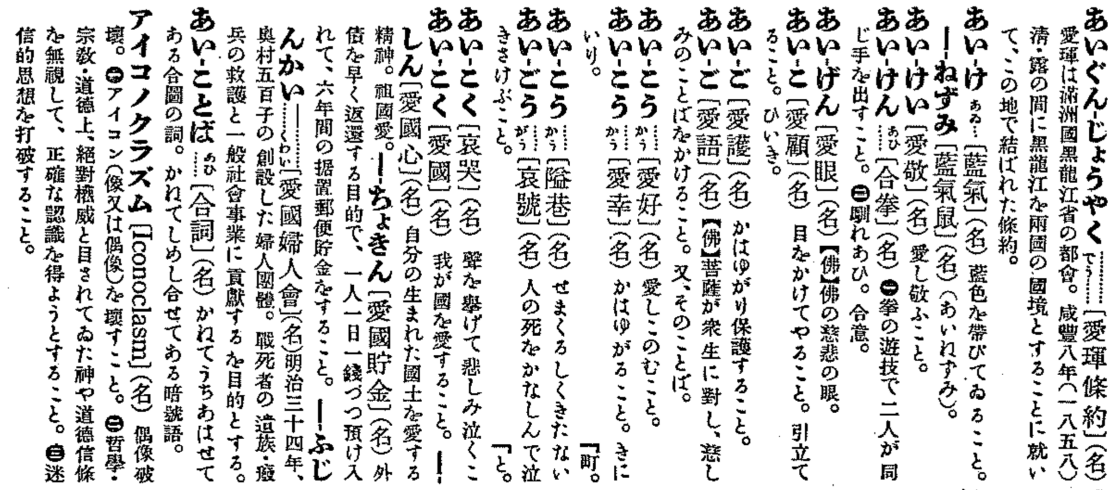

Figure 9: UPDATE CAPTION

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

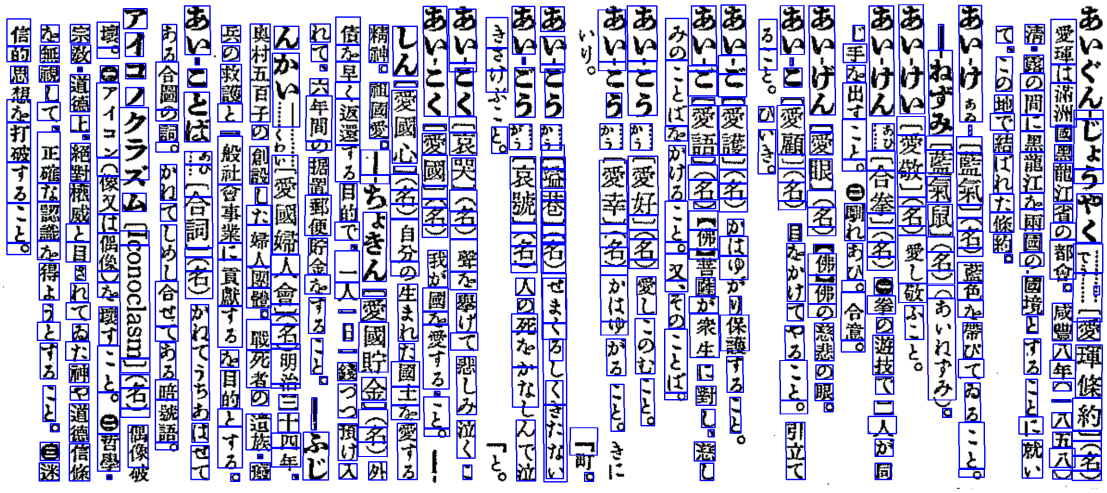

Figure 10: UPDATE CAPTION

4.2.2 Characters

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

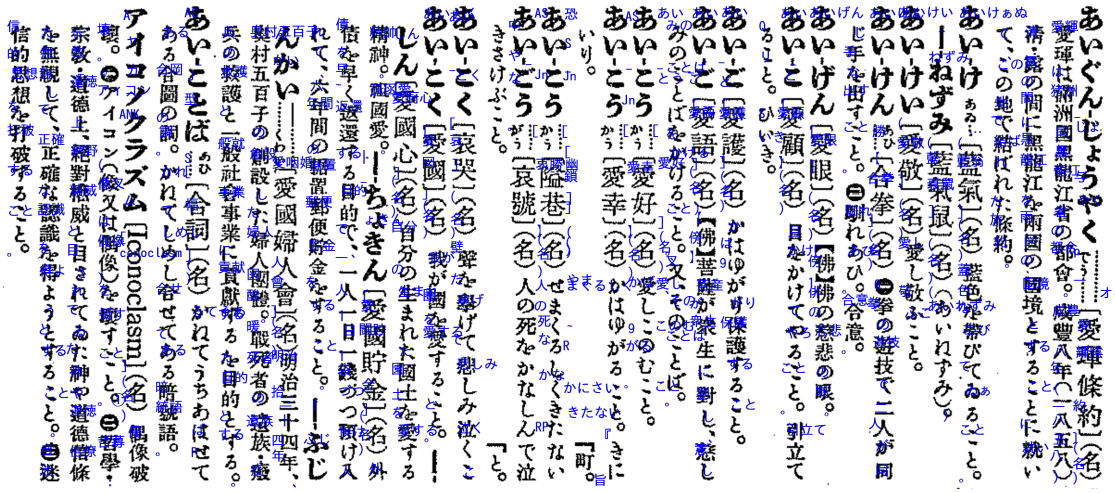

Figure 11: UPDATE CAPTION

4.2.3 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 hints and the other one with a list of two hints, such as Japanese and English, as the image contains some words in English. Both versions returned the same output.

4.2.4 Advantages and Disadvantages

4.2.4.1 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.

4.2.4.2 Disadvantages Unlike Azure Cognitive Services, which returns each recognized character individually, Google Vision groups characters in some cases. This situation adds some degree of complexity when processing the response from the API. [**Not sure if it is a disadvantage in real life**]

4.3 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 python library 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 xx.

```
]:  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 12: UPDATE CAPTION

From figure xx, 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 a subset of the previous one.

4.3.1 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.

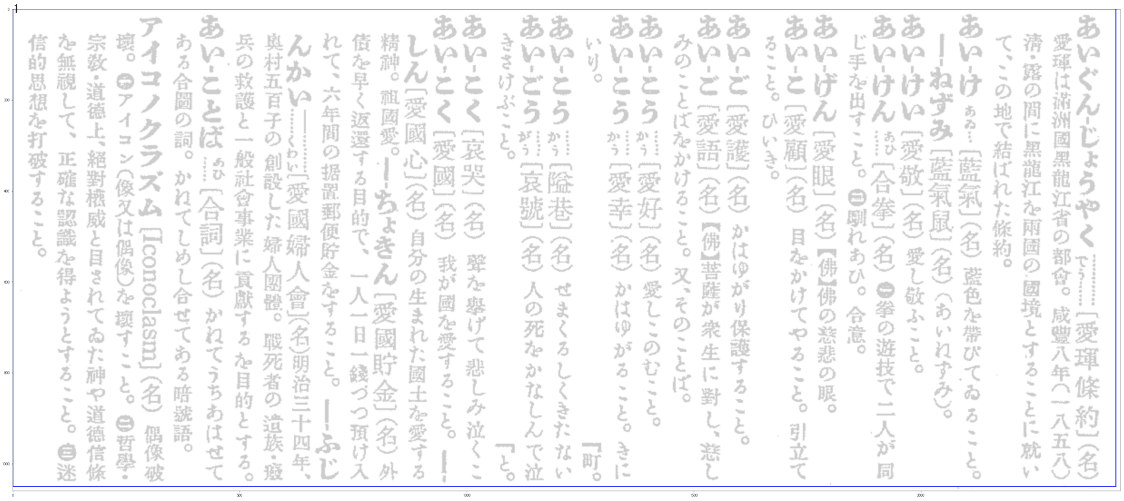

Figure 13: UPDATE CAPTION

4.3.2 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.

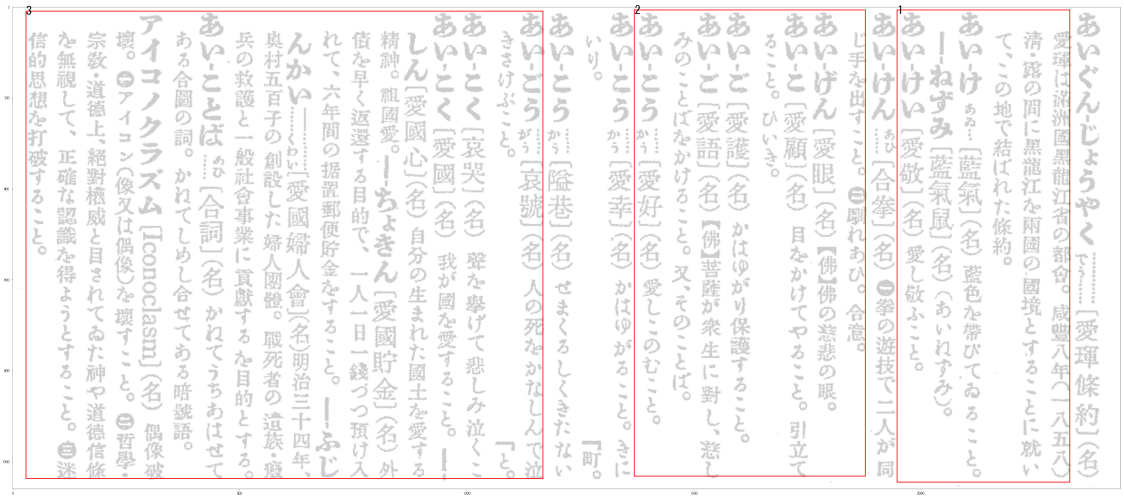

Figure 14: UPDATE CAPTION

4.3.3 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 three and four, which are subsets of block four (figure xx). It is interesting to see how the only paragraph four of block four is a very small area compared to its parent block. This situation happens in several other blocks. See figure xx for a better understanding.

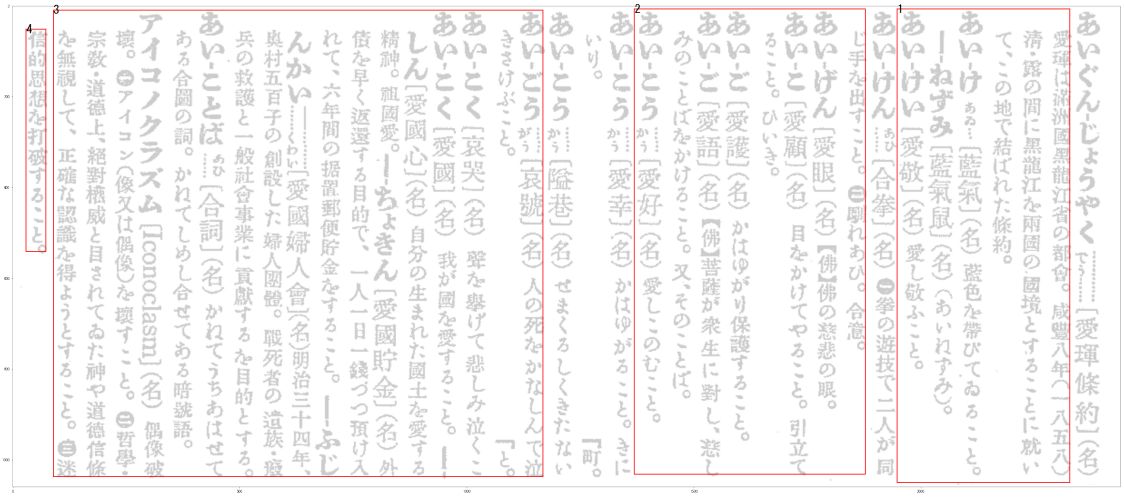

Figure 15: UPDATE CAPTION

4.3.4 Lines

Once the paragraphs are defined, the library provides in figure xx, 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. For some cases there is no clear reason why those characters are not included. For instance, between the lines 14 and 15, two clear lines are not recognized by the library.

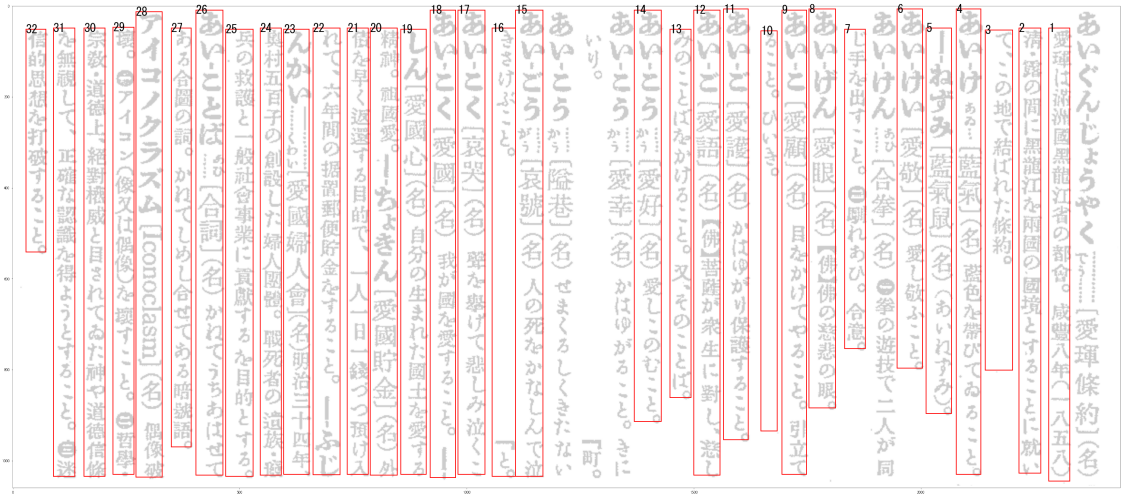

Figure 16: UPDATE CAPTION

4.3.5 Character Boundaries

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

Figure 17: UPDATE CAPTION

4.3.6 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 of recognized characters and the first character boundary of this line, there are two words, but one of them is not recognized. Figure xx illustrates this case.

4.3.7 Advantages and Disadvantages

4.3.7.1 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

Figure 18: UPDATE CAPTION

blocks, paragraphs, lines, boundaries, and finally the character level.

4.3.7.2 Disadvantages Some lines were completely ignored by Pyteseract, even thought it was clear enough to be recognized by the tool. Moreover, same as Google Vision, this library groups more than one character sometimes.

4.4 Kindai-OCR

According to the Center for Open Data in the Humanities, Kindai-OCR is an OCR system for modern Japanese documents. It was built under the N2I project that study state-of-the-art statistical models for OCR, and work on the development of infrastructure and providing open access to data. The datasets were constructed by the University of Tokyo Center for Education and Research and the National Institute of Japanese Language and are used for OCR machine learning. These datasets are commonly known as the Modern Magazine Datasets.

The Kindai-OCR code was developed in Python and released as an open source code in August 2020 on Github. The Github folder contains all the necessary code and instructions to assess the OCR tool. [**not sure if it is necessary to mention the modification in the file test.py I did, in order to run the model using GPUs.**]

After running the Kindai-OCR for our samples, the system generated two main outputs. They are explained as follows:

4.4.1 File result.xml

This file is presented in an Extensible Markup Language (XML) format containing a tree with the identified lines, as well as the corresponding characters. An example of the xml file is shown as follows:

```
<?xml version='1.0' encoding='Shift_JIS'?>
<paper xmlns="http://codh.rois.ac.jp/modern-magazine/"><page dpi="100" file="00-sample_pages_3_sec_3.png
```

After transforming this file into a dataframe form, we found that 238 lines with their corresponding characters, were generated. In figure XX we can see that the result.xml file provides information such as the corresponding level (paper, page, or line) in the tag column, the identified characters in the text column, the associated image the file column, among other attributes.

	tag	attributes	text	dpi	file	number	width	height	x	y
0	{http://codh.rois.ac.jp/modern-magazine/}paper	{}	None							
1	{http://codh.rois.ac.jp/modern-magazine/}page	{'dpi': '100', 'file': '00-sample_pages_3_sec_...	None	100	00-sample_pages_3_sec_3.png	1	2430	1040		
2	{http://codh.rois.ac.jp/modern-magazine/}line	{'height': '905', 'width': '72', 'x': '332', '...	あいくるし 愛嬌事、愛くるし（形、	100	00-sample_pages_3_sec_3.png	1	72	905	332	0
3	{http://codh.rois.ac.jp/modern-magazine/}line	{'height': '1026', 'width': '76', 'x': '466', ...	あいくち 匕首（合口。名。あれせも。ものよ	100	00-sample_pages_3_sec_3.png	1	76	1026	466	0
4	{http://codh.rois.ac.jp/modern-magazine/}line	{'height': '1026', 'width': '69', 'x': '592', ...	あいくすりし（（きき」名 その人の為に信身あ	100	00-sample_pages_3_sec_3.png	1	69	1026	592	0

Figure 19: UPDATE CAPTION

4.4.2 New version of the image

Kindai-OCR generated a new version of the image containing the identified lines and characters. An example is shown as follows in figure xx.

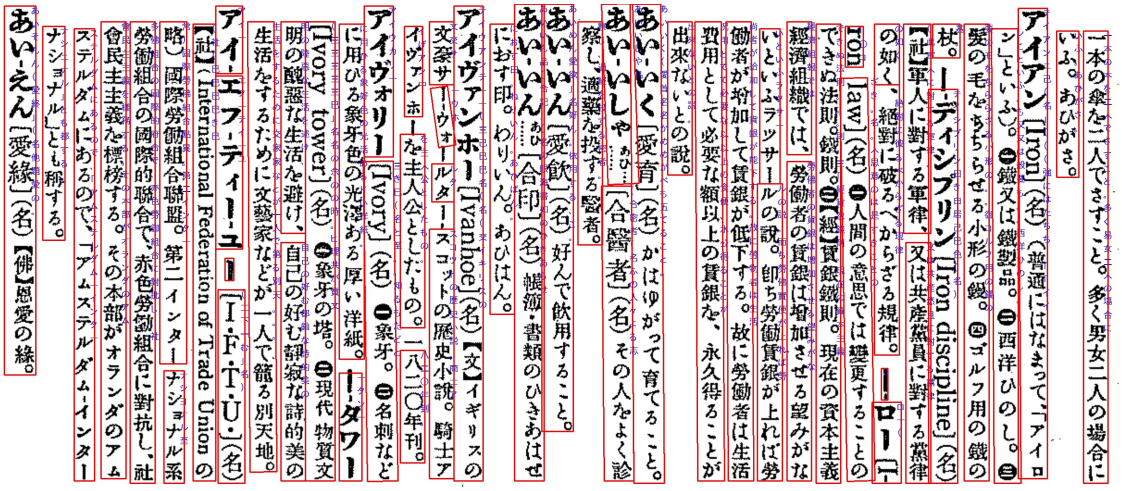

Figure 20: UPDATE CAPTION

4.4.3 Advantages and Disadvantages

4.4.3.1 Advantages Same as libraries such as Pytesseract, one of The most important advantages is that it is a free tool. Moreover, the recognition process is very easy. To obtain results, just by putting the images in the test folder and running the file called test.py on Python is enough to obtain the results.

4.4.3.2 Disadvantages At the time of this assessment, the use of GPUs is required which is a limitation to take into account. For this study, we used Colab from Google, which offers this capability. Moreover, Kindai-OCR returns the characters grouped by lines. They are not generated individually.

4.5 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 correspond(s) 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 **sec1__line1.gt.txt**.

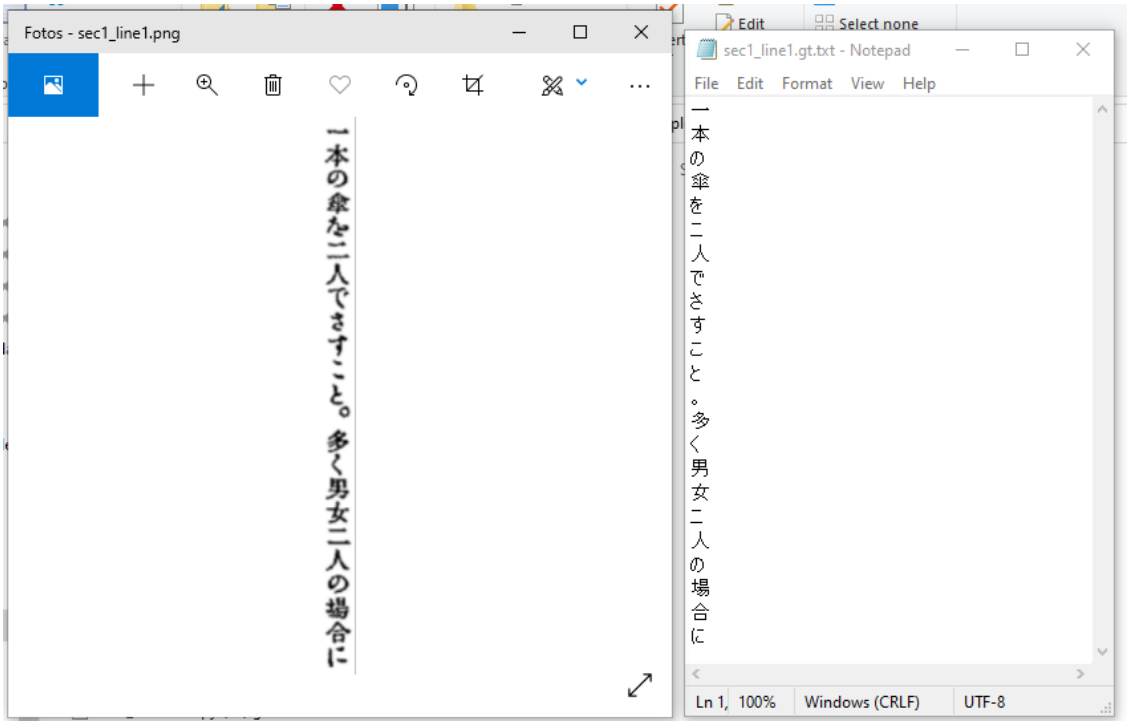

Figure 21: UPDATE CAPTION

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.

4.5.1 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 is needed, which is not in the scope of this study.

5 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 offers a Linux environment to assess the library.

5.1 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/

5.2 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/

6 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, 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.

7 Discussion

8 Appendix

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

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

References

Breuel, Thomas M. 2007. "Announcing the OCRopus Open Source OCR System." https://developers.googleblog.com/2007/04/announcing-ocropus-open-source-ocr.html.

Cheriet, Mohamed, Nawwaf Kharma, Cheng-Lin Liu, and Ching Y. Suen. 2007. *Character Recognition Systems: A Guide for Students and Practitioner*. Wiley.

Horiuchi, Tadashi, and Satoru Kato. 2009. "A Study on Japanese Historical Character Recognition Using Modular Neural Networks." In *2009 Fourth International Conference on Innovative Computing, Information and Control (Icicic)*, 1507–10. https://doi.org/10.1109/ICICIC.2009.57.

Kanagarathinam, Karthick, K Ravindrakumar, R Francis, and S Ilankannan. 2019. "Steps Involved in Text Recognition and Recent Research in Ocr; a Study" 8 (May): 3095–3100.

Kiessling, Benjamin. 2019. "Kraken - an Universal Text Recognizer for the Humanities." https://dev.clariah.nl/files/dh2019/boa/0673.html.

Kim, Min-Soo, Man-Dae Jang, Hyun-Il Choi, Taik-Heon Rhee, Jin-Hyung Kim, and Hee-Kue Kwag. 2004. "Digitalizing Scheme of Handwritten Hanja Historical Documents." In *First International Workshop on Document Image Analysis for Libraries, 2004. Proceedings.*, 321–27. IEEE.

Le, Anh Duc, Tarin Clanuwat, and Asanobu Kitamoto. 2019. "A Human-Inspired Recognition System for Pre-Modern Japanese Historical Documents." *IEEE Access* 7: 84163–9. https://doi.org/10.1109/ACCESS.2019.2924449.

Le, Anh Duc, Daichi Mochihashi, Katsuya Masuda, Hideki Mima, and Nam Tuan Ly. 2019. "Recognition of Japanese Historical Text Lines by an Attention-Based Encoder-Decoder and Text Line Generation." In *Proceedings of the 5th International Workshop on Historical Document Imaging and Processing*, 37–41.

Lombardi, Francesco, and Simone Marinai. 2020. "Deep Learning for Historical Document Analysis and Recognition—a Survey." *Journal of Imaging* 6 (10): 110.

Losee, Robert M. 2006. "Browsing Mixed Structured and Unstructured Data." *Information Processing & Management* 42 (2): 440–52.

Martínek, Jiří, Ladislav Lenc, and Pavel Král. 2020. "Building an Efficient Ocr System for Historical Documents with Little Training Data." *Neural Computing and Applications* 31: 17209–27. https://doi.org/10.1007/s00521-020-04910-x.

Nguyen, Hung Tuan, Nam Tuan Ly, Kha Cong Nguyen, Cuong Tuan Nguyen, and Masaki Nakagawa. 2017. "Attempts to Recognize Anomalously Deformed Kana in Japanese Historical Documents." In *Proceedings of the 4th International Workshop on Historical Document Imaging and Processing*, 31–36. HIP2017. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3151509.3151514.

Ooms, Jeroen. 2019. *Tesseract: Open Source Ocr Engine*. https://CRAN.R-project.org/package=tesseract.

Reul, Christian, Dennis Christ, Alexander Hartelt, Nico Balbach, Maximilian Wehner, Uwe Springmann, Christoph Wick, Christine Grundig, Andreas Büttner, and Frank Puppe. 2019. "OCR4all—an Open-Source Tool Providing a (Semi-) Automatic Ocr Workflow for Historical Printings." *Applied Sciences* 9 (22): 4853.

Rodrigues, Bruno. 2019. "Historical newspaper scraping with tesseract and R." https://www.brodrigues.co/blog/2019-04-07-historical_newspaper_scraping_tesseract/#.

Romanov, Maxim, Matthew Thomas Miller, Sarah Bowen Savant, and Benjamin Kiessling. 2017. "Important New Developments in Arabographic Optical Character Recognition (Ocr)." *arXiv Preprint arXiv:1703.09550*.

Rychlik, Marek, Dwight Nwaigwe, Yan Han, and Dylan Murphy. 2020. "Development of a New Image-to-Text Conversion System for Pashto, Farsi and Traditional Chinese." http://arxiv.org/abs/2005.08650.

Smith, Ray W. 2013. “History of the Tesseract OCR engine: what worked and what didn’t.” In *Document Recognition and Retrieval Xx*, edited by Richard Zanibbi and Bertrand Coüasnon, 8658:1–12. International Society for Optics; Photonics; SPIE. https://doi.org/10.1117/12.2010051.

“Tesseract Manual Page.” 2018. https://github.com/tesseract-ocr/tesseract/blob/master/doc/tesseract.1.asc.

Valy, D., M. Verleysen, and S. Chhun. 2020. “Data Augmentation and Text Recognition on Khmer Historical Manuscripts.” In *2020 17th International Conference on Frontiers in Handwriting Recognition (Icfhr)*, 73–78. https://doi.org/10.1109/ICFHR2020.2020.00024.