PenPal: A Handwritten Note Transcriber

Iris Yang

Figure 1.

Abstract

This is a handwritten notes transcription interface that helps users create a typed transcript version of their handwritten notes. The system takes image input of a user's handwritten notes, extracts text and any sketches contained on the note page, and outputs a typed transcript of the handwriting as well as any accompanying sketches. The model utilizes OpenCV to process the image input and Google Cloud Vision's Optical Character Recognition (OCR) to extract the text transcript of the notes.

This system has an accuracy rate of 95% ± 4% and a text confidence rate of 85% ± 5%. It works well for input that has clear boundaries between text and sketches, and is robust to accurately extracting notes with a variety of formats.

Introduction

In an era where efficiency and productivity are highly valued, and time is becoming increasingly precious, students across the nation seek to optimize their intake of knowledge while they are in school. Note-taking is an integral part of the learning process, and many studies have shown that handwritten notes are extremely beneficial to retaining new information. The process of physically writing down notes offers the advantage of higher recall and comprehension of material, as well as being able to draw any diagrams with ease. However, many students might also enjoy the convenience of having typed versions of theri notes--the advantages being having options to back up notes, access on different platforms, and use a simple Ctrl-F to find any desired information.

This system seeks to resolve this problem. With PenPal, users can reap all the benefits of handwriting notes, while being able to easily convert any handwritten content to a typed transcript. Additionally, the system will extract any areas that it determines to be a sketch, and output a cropped version of the original image.

Overview

Below is a high-level overview of the PenPal interface

1. The user is able to upload any number of note pages needed, then submit the images to be processed for transcription and sketch extraction.

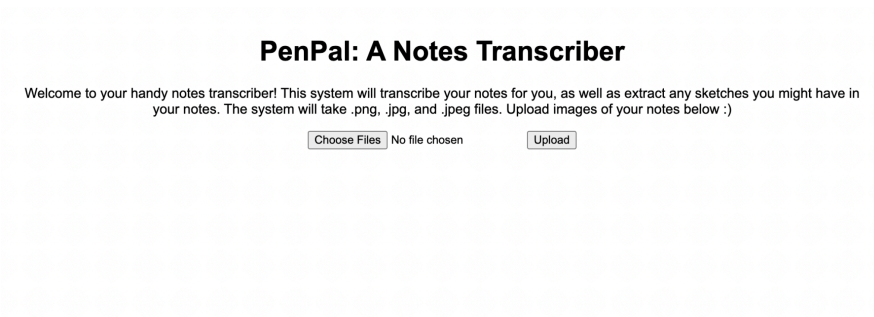


Figure 2: Home Page

2. The system will take the image inputs and run extraction on each image.

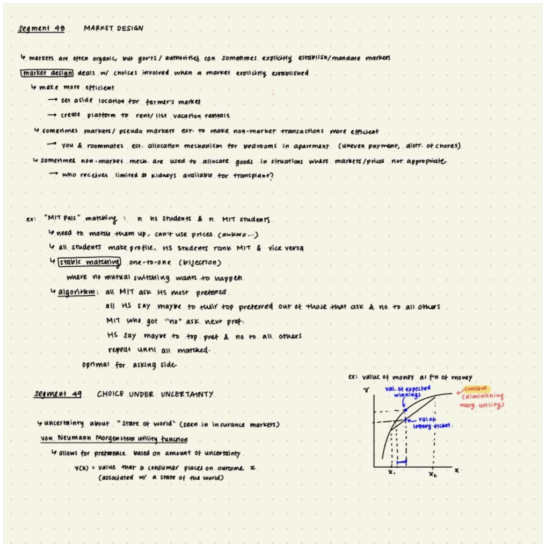


Figure 3: Example Note Page 1

Segment 46 INSURANCE

insurance markets

↳ insurance: pay small premium initially (or over time) before "state of world" revealed
→ receive no payment in positive state of world. but large settlement in negative state.
↳ where does information asymmetry come in?
→ purchaser can know they make the neg. state more likely
so insurance more valuable (insurer wouldn't know)
→ purchaser could engage in actions that make neg state more likely (insurer might not observe these)

↳ as a result of this, some regulation needed to keep things running (disclosure laws, etc.)
ex: auto, health, long term care insurance
↳ some markets don't need regulation → extended warranties on appliances
higher insurance.

→ gov't intervention in these markets dependent on risk aversion & difficulty in predicting the future by customers.

Segment 47 PRINCIPAL-AGENT MODELS

↳ informational asymmetries cause problems → market failures/inefficiencies.

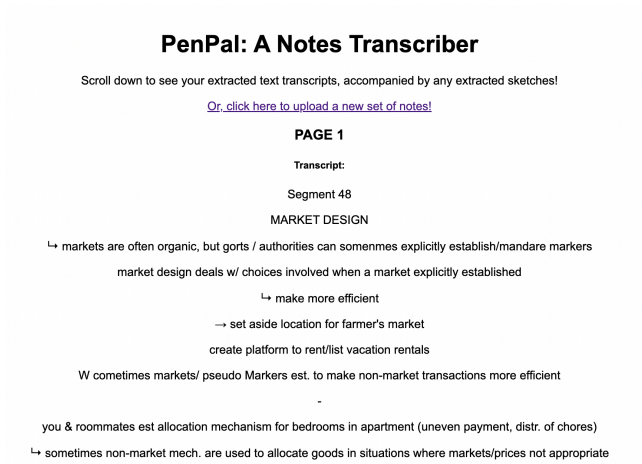
principal-agent problem (agency problem)

ex: soccer balls made up of black pentagons & white hexagons
manufacturer needs to cut hexs & pentagons.
but regular hexagons tile plane but pentagons don't! (excess material)
→ try & find more efficient way to tile pentagons
→ try & diffuse new info/ tech & observe diffusion
① manufacture new tech
② deliver to randomly selected firms & ensure them how to use it
③ firms w/ new tech immediately decreased costs & increasing profits
↓
but... those stopped using new tech.
b/c gives slightly harder to use for workers
→ profits increased
→ wages stagnant ↓
→ so workers lied to managers so they didn't have to use it any more

~ essentially: multiple agents (managers & workers) w/ diff information & diff incentives
it exists w/ superior info to make a market/process more efficient aren't true goals benefitting from making a market or process more efficient ⟹ efficiency may never be realized!

Figure 4: Example Note Page 2

3. The system outputs the text transcript, as well as any corresponding images (below is part of the output for Page 1)



PenPal: A Notes Transcriber

Scroll down to see your extracted text transcripts, accompanied by any extracted sketches!

Or, click here to upload a new set of notes!

PAGE 1

Transcript:

Segment 48

MARKET DESIGN

↳ markets are often organic, but gorts / authorities can somenmes explicitly establish/mandare markers

market design deals w/ choices involved when a market explicitly established

↳ make more efficient

→ set aside location for farmer's market

create platform to rent/list vacation rentals

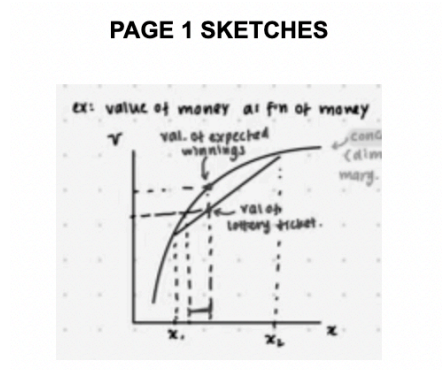
W cometimes markets/ pseudo Markers est. to make non-market transactions more efficient

-

you & roommates est allocation mechanism for bedrooms in apartment (uneven payment, distr. of chores)

↳ sometimes non-market mech. are used to allocate goods in situations where markets/prices not appropriate

Figure 5: Snippet of Text Transcript



PAGE 1 SKETCHES

Figure 6: Extracted Sketch from Page 1

System Mechanics

Architecture Overview

1. System takes images (.png, .jpeg, .jpg) as inputs.
2. Images are processed to be higher contrast, which reduces noice, allowing the content of the notes to be more accurately extracted.
3. Send processed images through Google Cloud Vision API for text extraction. In return, system receives a full text transcription, as well as the bounding boxes for any text on the image.
4. System binarizes the image and runs a clustering algorithm on the ink pixels in the image. The clustering algorithm is dependent on a clustering threshold, which corresponds to a lower bound on the "distance" between clusters. This clustering threshold is determined based on the overall ink density of the page. The higher the ink density, the lower the clustering threshold (more clusters will form), and vice versa. Using the clustering algorithm, we can calculate the bounding boxes of each cluster to get bounding boxes for all areas with ink on the image.
5. The system then identifies which areas are considered sketches by cross-referencing he text bounding boxes (which include ONLY text), and the ink bounding boxes (which include both text and sketches).
    - if a high percentage of an ink cluster is contained within text bounding boxes, we can safely categorize these clusters as text.
    - otherwise, we should categorize the cluster as a sketch.
6. The system also calculates the average confidence score of all text on the page, and use it as a guideline to to identify any sketches that may have been misidentifeid as text. If the confidence score is much lower than the average, the system identifies it as a sketch.
7. Iterate through all identified sketch blocks and merge it with any text that has sufficient overlap with the sketch block. This allows the system to combine any smaller labels for sketches with the sketch itself.
8. Delete any ink blocks that are not a part of text, and are smaller than a certain threshold based on the average word size of the text. This is because ink marks are likely not to be full sketches, and more likely to be ink strokes made by mistake.
9. Output text transcript and extracted sketches

Below is an architecture diagram that summarizes the main steps of extraction.

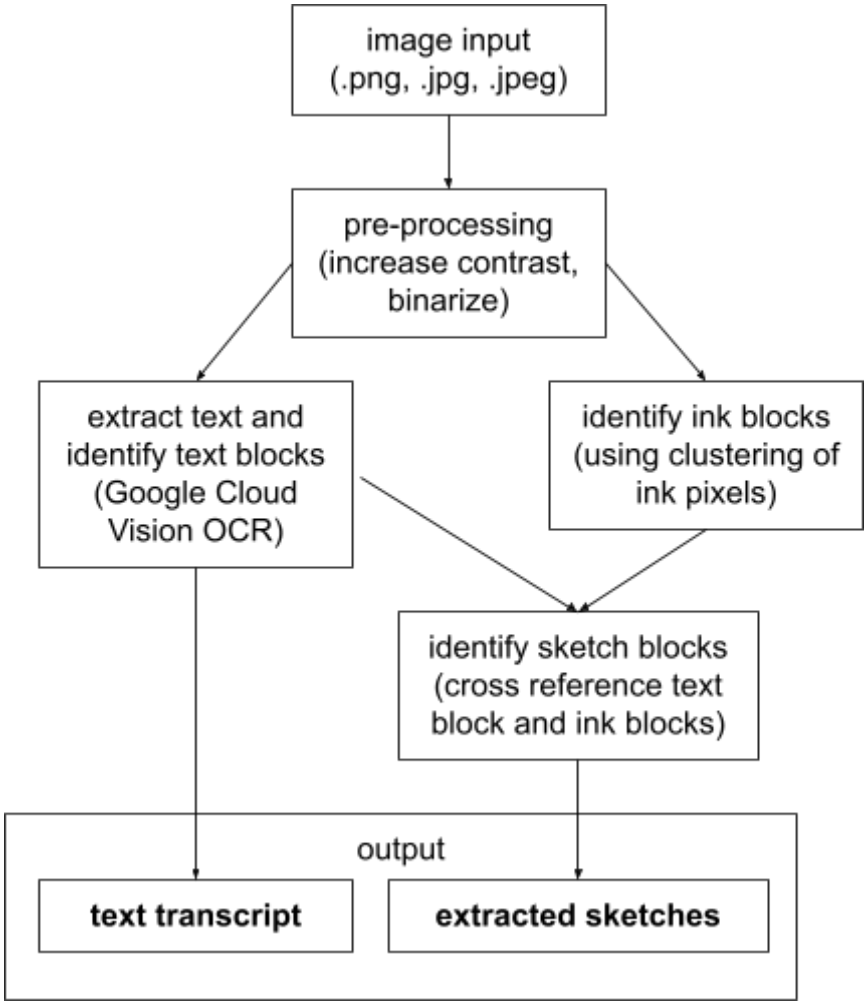


Figure 7: Architecture Diagram

Performance

The system is robust to different text and sketch orientations. This is extremely important when extracting handwritten notes, because the system is built to try and maintain the freedom of the note-taker when taking handwritten notes. Because of the way it is implemented, the format and order of the text and sketches on the page does not pose an issue to the extraction process. The image below shows the extracted bounding boxes for both sketches and text on the page.

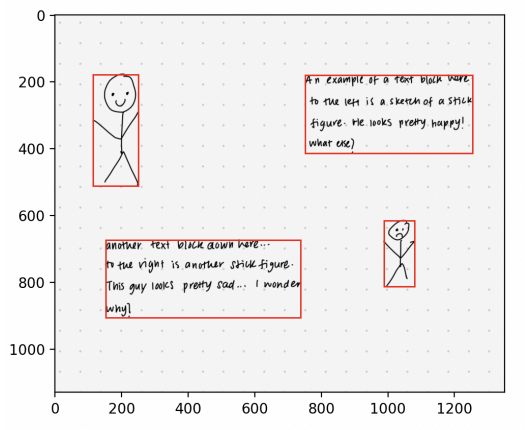


Figure 8: Example with Alternating Formatting

The system is also relatively robust in terms of sketch extraction. It can handle cases where the text is close to the sketch as a result of the cross-referencing of ink and text blocks. Below is an example of this case. The first image depicts the text bounding boxes, the second image depicts the ink bounding boxes, and the third image depicts the final cross-referenced bounding boxes. The fourth image is the extracted sketch. While there is some text included in the resulting extracted sketch, this is overall a non-issue because the system is built to prioritize extracting the *whole* sketch. This is because the output sketch would still be useful even if it had extraneous text included, but would be useless if the sketch was not extracted in its entirety.

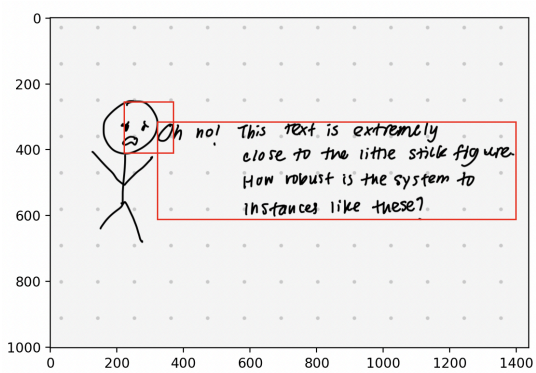


Figure 9: Text Bounding Boxes

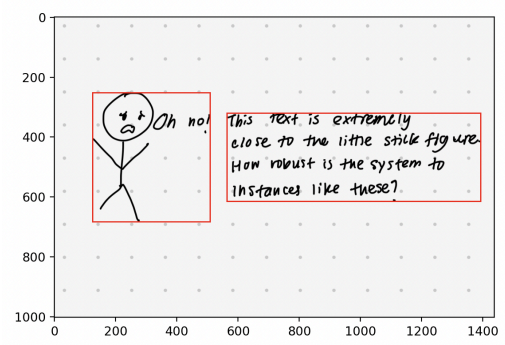


Figure 10: Ink Cluster Bounding Boxes

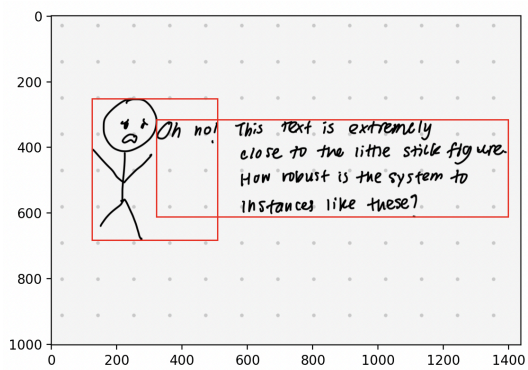


Figure 11: Final Sketch and Text Boxes

Figure 12: Final Extracted Sketch

An interesting failure that I encountered was the issue of text and sketch extraction in note-pages that were more ink-dense. Because the pages were more ink-heavy, large blocks of text and sketches were being clustered together with a constant clustering threshold. In the images below, we can see that my value of 50 for the clustering threshold yielded six clusters. However, the note page actually contains many more sketches than the clustering indicated. The extracted sketches were comprehensive, as in they extracted the entirety of the sketch, but also extracted a lot of the accompanying text.

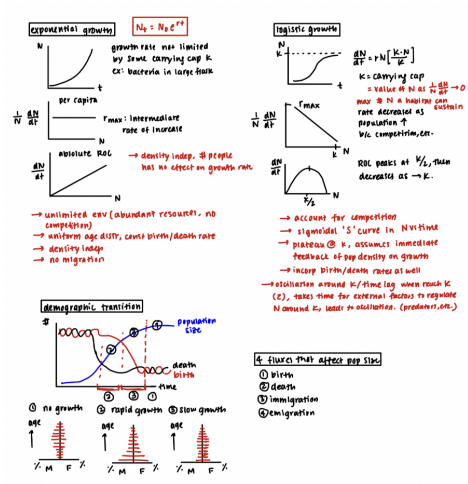


Figure 13: Dense Note Page

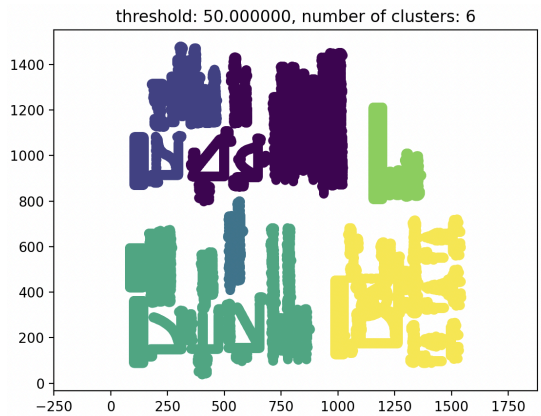


Figure 14: Clustering with Threshold = 50

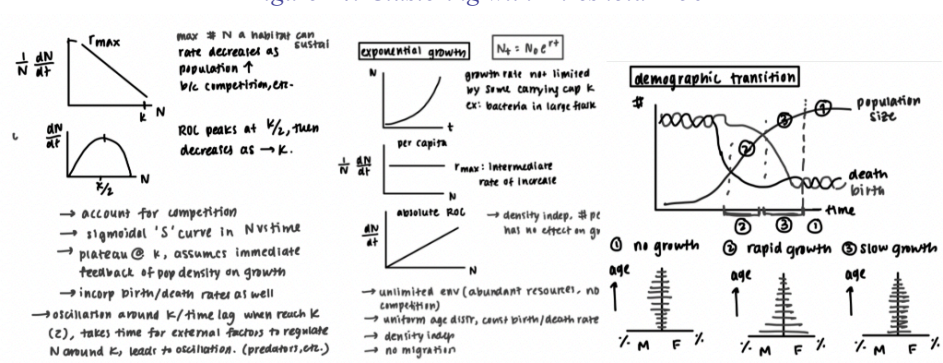


Figure 15: Extracted Sketches (Lots of Extra Text)

As a result, I decided to implement a variable clustering threshold that was dependent on the overall ink density of the note page. The denser the ink, the lower the clustering threshold (i.e., shorter distance between clusters, which means more clusters). Below is the result of the new implementation of ink clustering. There are many more clusters, and the extracted sketches are still comprehensive, but much more localized, and don't have extraneous text included.

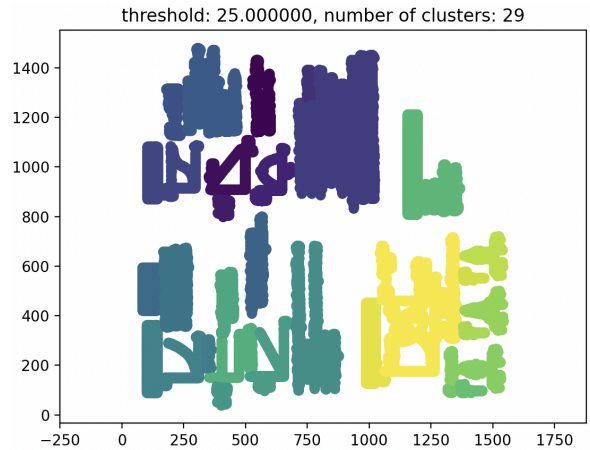


Figure 16: Clustering with Variable Threshold = 25

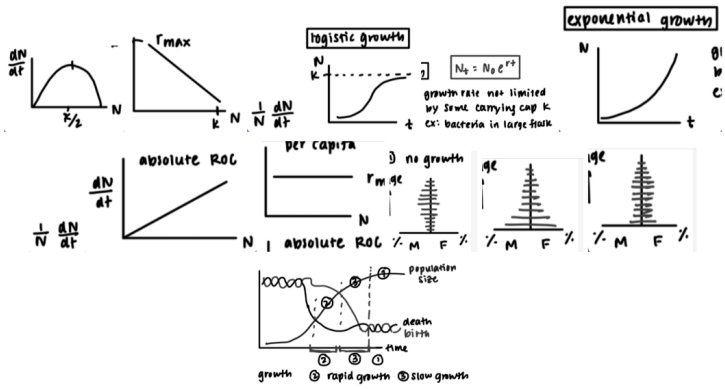


Figure 17: Compiled Extracted Sketches with New Threshold

User Study

For my user studies, I initially had potential users describe what they would want out of a system like PenPal. One of the main suggestions that was repeated was that the users wanted to be able to upload multiple images of their notes at once, so the system could process multiple pages in one go. There was also a lot of emphasis put on having accurate and comprehensive sketch extraction. It was expressed that the sketches would not be useful if they did not include all parts of a drawing. Finally, the users wanted a handwriting transcription service that was accurate.

I thought the emphasis on comprehensive sketch extraction was interesting, since it was not something I had considered before implementing the system. When I was implementing the system, I took this into consideration and used processing techniques that prioritized sketch bounding boxes over text bounding boxes to ensure that the entirety of all sketches was extracted.

I had users use my interface from front to back, and report any feedback they had on the usability and effectiveness of the system. Overall, the users were satisfied with the system, and said that the system performed the job as expected. One piece of feedback that I received was to create an interface to be able to preview the images before fully uploading. This was an aspect of the interface that I hadn't considered before as I was mainly focused on implementing the system and making it work. However, from a user standpoint, it makes sense that it would allow for a smoother user experience.

Some note pages took a longer time than expected to extract, which dissatisfied some users. However, this would easily be preventable if I had access to a machine with higher processing power.

The text extraction was overall not as accurate as the users would have liked, especially given the variability in certain users' handwriting. Depending on the penmanship, different users expressed different levels of satisfaction with respect to the handwriting recognition. Creating a more accurate transcription model would be a next step for PenPal, as I could implement functionalities such as a version of autocorrect in order to refine any extracted text from user input.

Performance

While Google Cloud Vision's OCR has an accuracy rate of 99.8% for typed texts, it is much more unreliable with handwritten text. Using samples of handwriting, my testing yielded an accuracy rate of 95% ± 4% and a text confidence rate of 85% ± 5%. The large error margins is due to the extreme variability of handwriting--not only from user to user, but also for each individual user. The other main factor that affects the overall accuracy rate is the ordering of different text on the page. In the case that the transcript outputs text blocks in an order that is different from the "true" value, it is labeled as inaccurate, even if the text is correct. Overall, the system appears to be relatively robust, and generally accurate.

Something that could be implemented as an interesting next step would be a version of autocorrect built into the system. This would allow the system to refine any extracted text from user input, and overall achieve a higher accuracy rate on the text extraction for more satisfactory results. This would overall allow for cleaner, more legible output, and reduce variability in accuracy rates both across different users as well as for individual users.

Tools and Packages

<u>Google Cloud Vision API</u>

1. The Google Cloud Vision API's Optical Character Recognition (OCR) is written in multiple languages, including Java, Python, Node.js, C#, and Ruby. It can be called from any programming language that supports HTTP requests, such as Java, JavaScript, C#, etc. In this case, I am using Python.
2. The input to the Google Cloud Vision API's OCR consists of images containing printed or handwritten text. These images can be in various formats, including .png, .jpg, and .jpeg. The content of the image should ideally be clear, legible, and contrasted for the best results.
3. The output of the Google Cloud Vision API's OCR is a JSON response that contains the extracted text from the input image, along with additional information such as the bounding boxes of detected text regions and confidence scores for each detected word. Specifically, PenPal uses the "FullTextAnnotation" of the response. The output is an extremely nested JSON file that has layers of values including "page," "paragraph," and "word." Below is a screenshot of a function I wrote to print the full structure of the full text annotation response.

```python
def print_structure(response):
    for page in response['pages']:
        print("### NEW PAGE ###")
        print("width:", page['width'])
        print('height', page['height'])
        ### average page confidence!
        print('confidence', page['confidence'])
        for block in page['blocks']:
            print("### NEW BLOCK ###")
            print('bounding box:', block['boundingBox'])
            print('block type', block['blockType'])
            print('confidence', block['confidence'])
            for paragraph in block['paragraphs']:
                print("### NEW PARAGRAPH ###")
                print('bounding box:', paragraph['boundingBox'])
                print('confidence', paragraph['confidence'])
                paragraph_text = ''
                for word in paragraph['words']:
                    text = ''.join([symbol['text'] for symbol in word['symbols']])
                    paragraph_text += " " + text
                    print("---> WORD =", text)
                    # print("bounding box", word['boundingBox'])
                    # print('confidence', word['confidence'])
                print("PARAGRAPH TEXT:", paragraph_text)
```

Figure 19: print_structure( ) Function

4. No, the Google Cloud Vision API is not open source.
5. Caveats in the description of the Google Cloud Vision API's OCR include the potential for inaccuracies in text recognition, especially with handwritten or low-quality images. Additionally, the API may have limitations on the number of requests or the size of input images for free usage plans.
6. In order to use the Google Cloud Vision API's OCR, you need access to the Google Cloud Platform, as well as authentication credentials. The API can be accessed using client libraries and HTTP requests, using any of the languages that support HTTP requests. In this case, I am using Python, which is supported by my OS.
7. The Google Cloud Vision API is actively developed and maintained by Google. Updates and improvements to the API are regularly released, and any issues or bugs are continuously being addressed.
8. The Google Cloud Vision API uses machine learning models trained on large datasets of images containing text. The specific training data for the model is not available to the public.

<u>Flask</u>

1. Flask is a micro web framework written in Python and callable from any programming language that supports HTTP requests. It is specifically designed to be used with Python, which is what this system uses.
2. The Flask server takes HTTP requests such as GET or POST. The content of the input depends on the intended functionality.
3. Flask provides output in the form of HTTP responses. The content of the output also depends on the intended functionality.
4. Flask is open source, and released under the BSD license.
5. Flask is a very minimalistic web framework, which means that not all web app functionalities may be able to be implemented with just Flask. This means that it may be necessary to install additional libraries to integrate functions such as authentication. This also poses an obstacle to scalability. However, for the purposes of PenPal, this is not necessary.
6. Flask does not require any additional tools or packages to run, other than being able to run Python (which is available for MacOS).

7. Flask is actively developed and maintained by a community of contributors. Updates and bug fixes are regularly released, and the Flask GitHub repository shows recent activity, which indicates that it is still being updated.
8. Flask does not rely on machine learning or trained models.