MATLAP : Manga Automatic Translation & Language Assistance Program

Steven Lee, Thodori Kapouranis

May 2022

Contents

1 Abstract

This project uses recent advancements in object detection in conjunction with annotated comic book datasets to create a comic reader app intended for advanced foreign language learners. Many intermediate/advanced learners of eastern Asian languages, Chinese and Japanese in particular, opt to use comic books as enjoyable learning material. Using object detection techniques, this project implements automatic text bubble detection which allows for text mining of comic books and integrates it with existing popular open source flashcard applications. This allows for easy and efficient studying from comic books, where it would otherwise require having multiple applications and dictionaries open. The project is served as a local desktop application with slow CPU performance, although it is easily extendable to a client and remote ML GPU server model where performance would increase immensely.

2 Background Information

2.1 Growth of Interest in (Asian) Languages

In recent decades with the expansion of the internet there has been a large increase of popularity surrounding foreign media and foreign languages. Amongst the fastest growing languages are East Asian languages, especially amongst younger groups of people. Many universities, such as University of Dallas have noted that registration for Japanese has increased 260% from 2017 to 2021 [2]. Overall language registration also grew by 76%, indicating the growing demand amongst younger generations.

Many have taken opportunity from the spike of interest to create resources for language learners, but due to the natural difficulty of making habits stick, language learning resources are heavily saturated towards beginners. One of the hardest aspects of learning a new language is deciding when and how to wean off beginner friendly material and begin utilizing native materials. A large problem with native materials is that they are dense with words a learner may not know, which makes studying and learning from these both frustrating and slow. They are intimidating and may be the quitting point for many learners.

Comic books are often one of the first native materials learners of Japanese want to jump into due to the young learner demographic. Comic books are stored in image formats so text is not selectable and makes studying unknown words and structures further frustrating. Our application aims at softening the difficult curve of jumping into these materials by making the process more streamline and enjoyable.

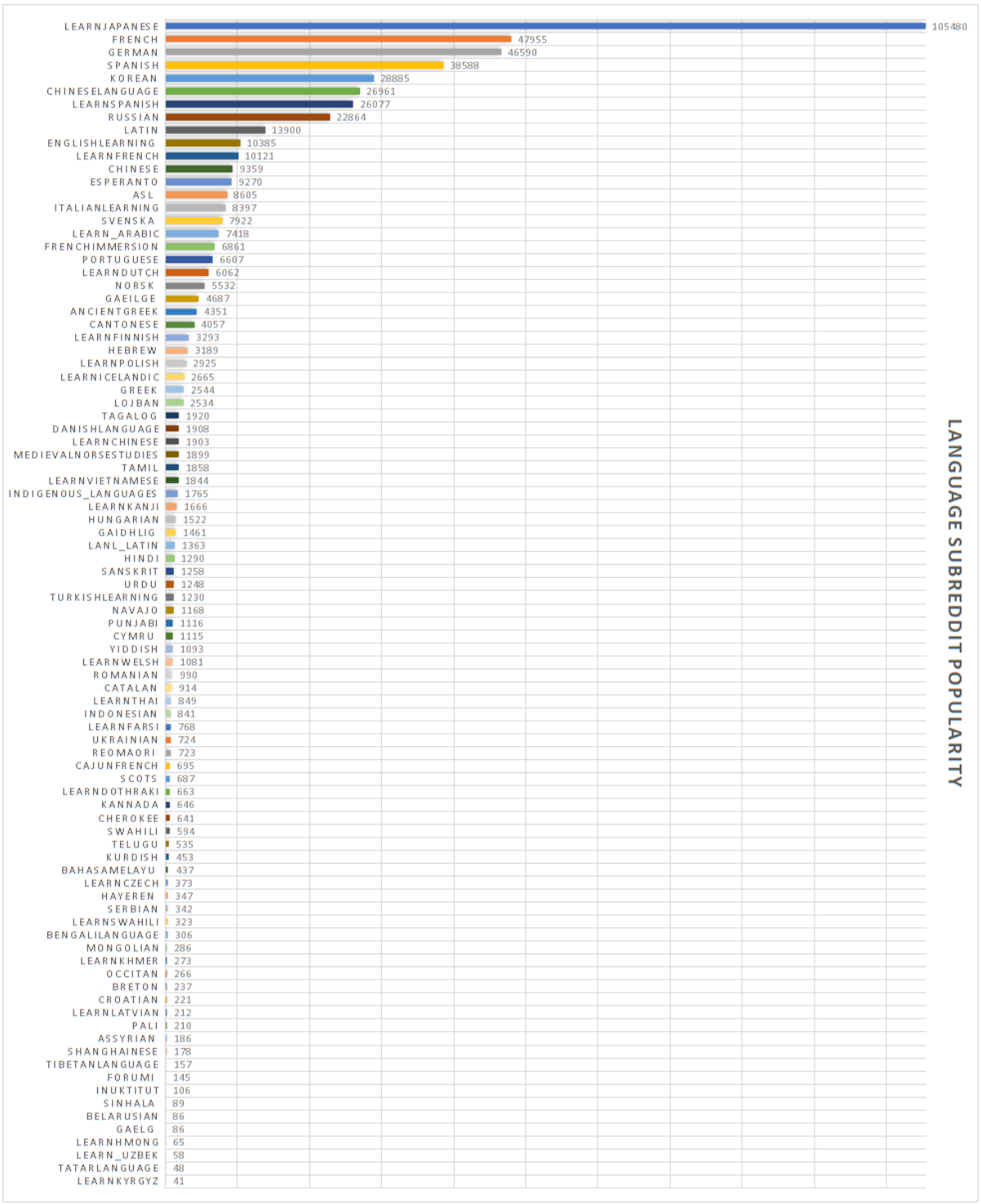

Figure 1: Language Subreddit Popularity (2017). r/LearnJapanese has now expanded five fold in 2021, largely due to the Anime and Manga mediums.

2.2 Japanese Language

2.2.1 Morphology

The morphology of the Japanese language makes it especially difficult for native English and other Indo-European language speakers. The Japanese Language has three writing systems, Hiragana, Katakana, and Kanji. Hiragana and Katakana are similar to the English alphabet, they are the syllabary alphabet of the language that composes words. Kanji is the adopted and transformed version of Chinese symbolic characters, Hanzi. Each kanji has multiple readings and meanings. The Japanese Ministry of Education has deemed 2,136 Kanji to be the basis of high school level literacy (Jōyō Kanji). All three written forms are used in Japanese sentences, adding considerable complexity for language learners.

Figure 2: Japanese Writing Systems

It is a difficult task to learn unknown kanji without having any selectable text because the pronunciation is not apparent. This makes typing it to search in a dictionary a challenging and time consuming task. There exists applications that infer kanji from the user drawing it out. These aren't always accurate, and they are time consuming. For words that are made of multiple kanji where the user knows some but not all, regex searches with wildcards can be used to search the missing kanji. Physical dictionaries are so inefficient that they are very rarely in use anymore.

Japanese sentences do not have whitespace to separate words. This makes it difficult to discern words from one another for learners and Natural Language Processing (NLP) tasks. Since white-spaces cannot be used, large dictionaries are needed to assist NLP packages with splitting sentences into their smallest word components, called tokenization.

Native Japanese material occasionally has the pronunciation above kanji written in smaller font hiragana, called 'furigana'. This is used for difficult words that is outside the scope of regular high school education, or in materials for younger audiences. Comic books, for example, appeal to younger audiences, so they often have furigana. For Optical Character Recognition (OCR) tasks, furigana next to kanji can be interpreted as sentences of their own or add noise to the kanji they're attached to. This issue is detrimental to the accuracy of OCR if furigana is not taken into account. The simplest solution to this is to preprocessing the images before feeding into an OCR model to remove the furigana noise.

Figure 3: Furigana attached to kanji.

2.2.2 Grammar

Japanese follows a Subject Object Verb (SOV) grammar structure. English and nearly all Indo-European languages on the other hand, follow an SVO grammar structure. This makes machine translation between these languages fairly difficult as the entire structure of the sentence must be shuffled around to be logical in the target language.

Japanese additionally is a context rich language. Very often, the subject and/or topic of sentences is completely dropped and must be inferred by the speakers in the conversation. This adds considerable difficulty in machine translation, and often requires a model with memory to make sense of what is being talked about. In Linguistics, languages that permit this omittance are called 'null-subject languages'.

Here is an example of a sentence with both an omitted subject and topic:

発売日　なんだけど、　迷っている。
hatsubaihi nandakedo, mayotteiru.
Release day, but hesitating.

What is being released, who the person hesitating is, and what they are hesitating to do are not made clear. Despite this, it is a completely normal colloquial sentence in modern Japanese. Comic books tend to have fairly colloquial speech which follows the trend of being less verbose and therefore omits information often. As language learners, it adds a layer of difficulty of reading between the lines that beginner friendly material does not have.

To showcase how much is omitted, this is the sentence if it were as verbose as proper English:

今日　は　発売日　なんだけど、私　は　それ　を　買おうか　どうか　迷っている。
kyō wa hatsubaihi nandakedo, watashi wa sore o kaōka dōka mayotteiru.
Today is the release day, but I am hesitating whether or not to buy it.

2.3 Popular Language Learning Methods

With the development of software and mobile phones, language learning has largely shifted from a rote learning approach to space repetition approach. In the late 1960s, professors of psychology Thomas K. Landauer and Robert A. Bjork ran experiments to explore human memory and studying methods. They tested a space repetition system, where one studies a piece of information in a set schedule with increasing intervals if they manage to correctly recall that information. If they fail to do so, the progress is reset. For example, the days

between reviewing a word again can be 1, 1, 2, 3, 5, 8, and onwards if they are correctly recalled each time. If a user cannot recall the word, it goes back to the beginning. Research showed this method being very efficient for memorizing information long term, and proved useful for students and people suffering from memory problems. In the 21st century the popularity of these spaced repetition systems (SRS) has exploded within language learning communities. Anki is the most popular open source flashcard program that implements these learning systems. It is easily customizable and open source plugins are written to make studying certain subjects easier, which this project aims to utilize.

2.4 Natural Language Processing

2.4.1 Optical Character Recognition (OCR)

OCR is a way to detect characters from a given image and outputs the corresponding characters as output. The basic OCR translate text documents or simple text images to text code which can then be processed to output the characters. Our usage of OCR is to implement it on the text-boxes to idenitify the characters and store those data for our application.

2.4.2 Tokenization

In NLP the task of tokenization is splitting a sentence into separate tokens before they are sent to another stage of processing. Most commonly, the sentence is split on each unique word or into the smallest linguistic pieces holding information, called "morphemes". Due to the lack of whitespace in Japanese, words are typically split up using a large dictionary corpus.

2.5 Computer Vision

2.5.1 Single Shot Detection (SSD)

The Single Shot Multi Detection model was first introduced in 2015 to provide a solution for real time object detection [9]. It is commonly used in scenarios where computation speed is necessary, such as in autonomous vehicles where computation has to be nearly real-time. It is a simple model to train and integrate, but it compromises on accuracy when compared to other models. The first attempts at object detection in comic books used an SSD model.

2.5.2 Region Based Convolution Neural Networks (R-CNN)

Convolution Neural Networks (CNN) are a class of deep learning models used commonly in computer vision. Region based CNN's (RCNN) build upon CNNs to output the bounding boxes for regions where the model predicts certain object classes exist [3]. RCNNs have been continuously improving in both runtime speed and prediction capabilities. The next iteration, Fast R-CNN, cut down on the model training time nearly 10 fold and evaluation time more than 20 fold [4]. A year later, Faster R-CNN continued to cut down on training time [11]. The Mask R-CNN model is the latest iteration, building off of Faster R-CNN but is able to predict more complex bounding region shapes other than a rectangle [6].

2.6 Machine Learning Research in the Comic Domain

Due to the sudden rise in datasets in the past few years, there has been little previous work done on comic book object detection. Most tasks involving comic books previously were regarding super scaling images and to fix noisy artifacts from low resolution pages. Recently there have also been an increase in interest in researching automatic translation of comic books using a variety of methods. Currently the most effective translation method involve using object detection models to detect information from the comic panels in order to aid translation.

2.6.1 Manga109 Dataset

A comprehensive annotated manga data set called Manga109 is available for use upon request. It is the most comprehensive annotated Manga dataset to date, featuring annotation regions for comic panels, character faces, character bodies and text boxes [10]. Ogawa et al (2018) applied Single Shot Detection (SSD-300) to create a object detection model to detect the regions annotated. The paper set the baseline metrics for object detection tasks in the realm of comic books.

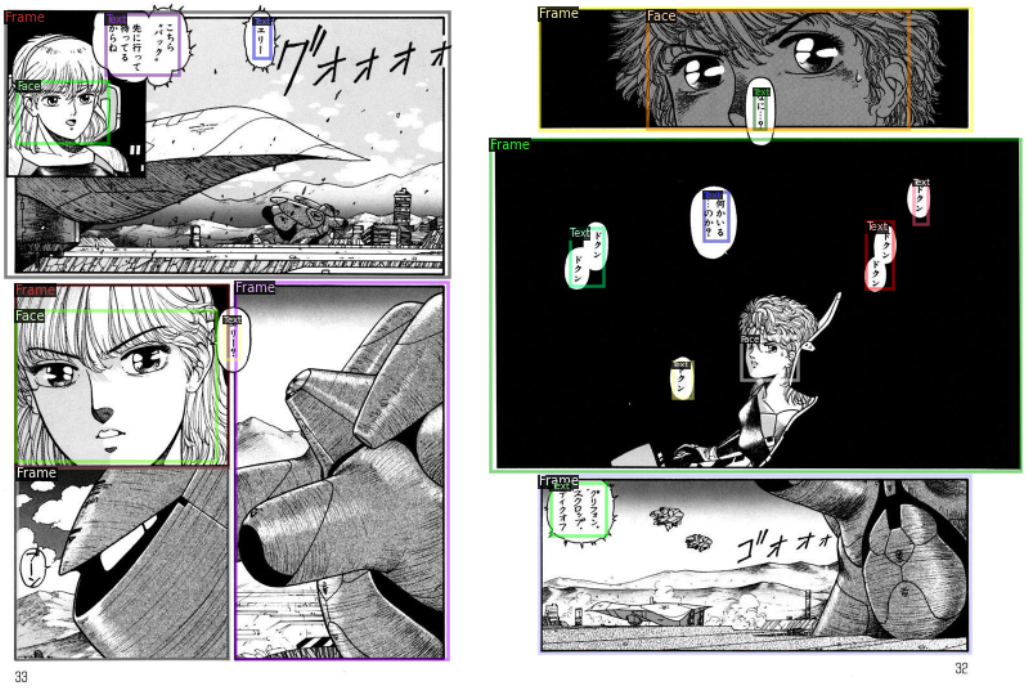

Figure 4: Sample page with annotations from the Manga109 dataset

2.6.2 Towards Fully Automated Manga Translation

Ryota et. al (2021) continued exploring the task of information extraction from Manga [7]. Ryota et. al utilized a Faster R-CNN model with a ResNet101 backbone to achieve 11.3% higher text region detection accuracy (95.4%) at the cost of runtime speed. In addition to increased inference performance, the paper introduces novel methods of contextual text translation. By grouping together text bubbles that occur within the same frame, using facial features of the characters present in the frame, and determining visible gender of the characters, they were able to achieve better machine translation performance. Because Japanese often omits the subject of a sentence, using visual information to infer omitted subjects and topics is a future avenue of research. Finally, they also propose automatic text region cleaning and automatic typesetting, paving a future path to complete automatic comic book translation.

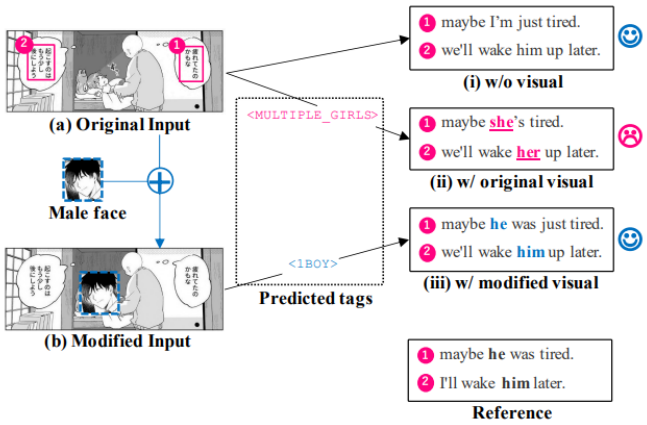

Figure 5: Sample of how contextual translations can change based on inferred character gender)

Method	input image size	backbone	text		frame	
			AP	AP_{50}	AP	AP_{50}
SSD-fork (Ogawa et al. 2018)	300	VGG	n/a	84.1	n/a	96.9
Faster R-CNN	500	ResNet-101	65.0	92.5	91.6	97.5
	800	ResNet-101	69.3	94.4	92.5	97.6
	1170	ResNet-101	**71.2**	94.9	92.5	97.7
	1170	ResNet-50	70.9	94.8	90.7	97.5
	1170	ResNet-101-FPN	70.3	94.4	92.5	97.7
	1170	ResNeXt-101	70.4	94.5	**92.9**	**98.5**
RetinaNet	1170	ResNet-101	70.6	**95.4**	89.8	98.3

Figure 6: Text and frame detection performance on the Manga109 dataset [7]

2.7 Previous applications

2.7.1 Manual OCR

Current OCR manga reading applications exist but they are all manual and require user input. This means that a user would have to drag a box with their mouse or fingers to select the area to perform OCR on. The deciphered plain text would then be presented to them and do whatever they wish with it. These systems are all manual, so for the program to know a sentence exists means the user must select a box over it. With only manual OCR, a lot of potential future features are blocked because the program has no means of operating on sentences that are not manually selected by the user.

2.7.2 Automatic Flashcard Exports

Some manual OCR manga readers allow the users to export the selected vocabulary to flashcard applications. This feature is immensely useful for the end user because it simplifies the process of searching for word definitions and making the card. With automatic region detection the task of adding cards can become even faster as everything has been ran through OCR beforehand. Additionally since Manga109 has annotated panel regions, it becomes possible to instantly export the panel that the sentence comes from as well. This would be useful for sentences that do not make sense without some context, which are plentiful in Japanese.

3 Concept of Operations

Using the information presented in the previous section, our project implements these operations for end users:

— Allows user to upload their own folder of manga images.

— Automatically detects text and panel regions.

— Performs vertical Japanese OCR in the region of interest.

— Automatically splits tokens and attatches their definitions.

— The user is able to hover over text regions and select tokens from a popup to see the definition of the selected word.

— On the click of a button the user is able to export the token, definition, sentence, and panel region to an external flashcard application.

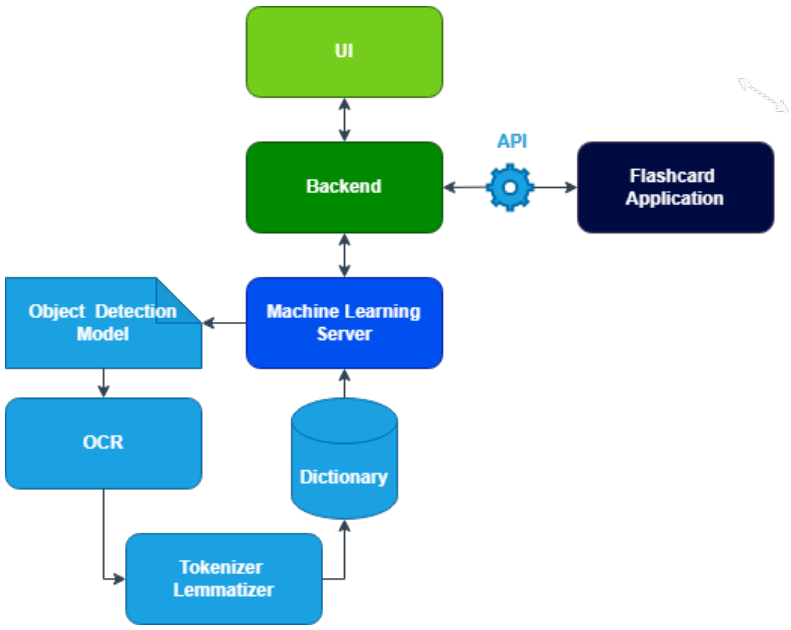

Figure 7: Overview of the project operation

Further details into each of these systems will be introduced in the following sections.

4 Machine Learning Server

4.1 Object Detection Model

4.1.1 Detectronv2

Our project decided to use Detectronv2 as the object detection library/pytorch wrapper. Detectronv2 library was developed by Facebook to easily provide state of the art object detection models. Detectronv2 utilizes the latest mask R-CNN model. Mask R-CNN is not usable by our dataset as there is no segmentation data for the objects, each bounding region is simply a rectangle. Because of that, the dataset is instead trained on a faster RCNN model which is the same model used in Ryota et al [7].

4.1.2 COCO format

To use Detectronv2, we would require to put our data in a COCO dataset format (Common Object in COntext). The Manga109 dataset is in an XML format, so preprocessing of every single annotation file into the correct format was performed.

Since the Manga109 dataset comes in multiple folders for each of the manga images and their associated annotation files, they must first be concatenated into one large folder and with one large annotation file. Since annotation files reference the image file names, careful attention had to be made to ensure that each image file's name was unique. This was accomplished by adding the comic's name as prefix to each image file's name.

```
<pages>
  <page index="0" width="1654" height="1170"/>
  <page index="1" width="1654" height="1170"/>
  <page index="2" width="1654" height="1170">
    <frame id="00000000" xmin="83" ymin="86" xmax="751" ymax="1090"/>
    <text id="00000001" xmin="550" ymin="660" xmax="583" ymax="696">あ</text>
    <body id="00000002" xmin="178" ymin="660" xmax="548" ymax="965" character="00000003"/>
    <face id="00000004" xmin="406" ymin="684" xmax="456" ymax="764" character="00000003"/>
  </page>
  <page index="3" width="1654" height="1170">
    <text id="00000005" xmin="601" ymin="291" xmax="685" ymax="402">キャーッ</text>
    <body id="00000006" xmin="1229" ymin="709" xmax="1352" ymax="875" character="00000003"/>
    <text id="00000007" xmin="1155" ymin="595" xmax="1239" ymax="686">はやく逃げないとまきぞえくっちゃう</text>
    <body id="00000008" xmin="959" ymin="820" xmax="1172" ymax="1089" character="00000003"/>
    <frame id="00000009" xmin="899" ymin="585" xmax="1170" ymax="1085"/>
    <face id="0000000a" xmin="989" ymin="890" xmax="1072" ymax="941" character="00000003"/>
    <text id="0000000b" xmin="925" ymin="773" xmax="943" ymax="803">え？</text>
    <frame id="0000000c" xmin="2" ymin="0" xmax="826" ymax="513"/>
    <face id="0000000d" xmin="341" ymin="615" xmax="453" ymax="700" character="00000003"/>
    <frame id="0000000e" xmin="72" ymin="516" xmax="743" ymax="1101"/>
    <face id="0000000f" xmin="1113" ymin="882" xmax="1162" ymax="940" character="00000010"/>
    <body id="00000011" xmin="499" ymin="399" xmax="646" ymax="501" character="00000010"/>
    <body id="00000012" xmin="1202" ymin="363" xmax="1307" ymax="545" character="00000003"/>
    <text id="00000013" xmin="64" ymin="646" xmax="150" ymax="736">今の時代は人を見たら敵と思えなのよっ</text>
    <frame id="00000014" xmin="906" ymin="95" xmax="1575" ymax="576"/>
    <text id="00000015" xmin="1104" ymin="746" xmax="1174" ymax="830">そっちはあぶないよ……</text>
    <text id="00000016" xmin="624" ymin="524" xmax="697" ymax="601">あービックリしたっ</text>
    <text id="00000017" xmin="232" ymin="515" xmax="307" ymax="584">またあの男だわっ</text>
    <text id="00000018" xmin="1071" ymin="348" xmax="1112" ymax="455">オートマトンたちが来るわ</text>
    <body id="00000019" xmin="273" ymin="541" xmax="741" ymax="1093" character="00000003"/>
    <text id="0000001a" xmin="1452" ymin="154" xmax="1471" ymax="203">やばっ</text>
    <text id="0000001b" xmin="166" ymin="557" xmax="255" ymax="697">ああいうのはきっと痴漢か変態ねっ！</text>
    <text id="0000001c" xmin="1420" ymin="1009" xmax="1465" ymax="1051">あ</text>
    <frame id="0000001d" xmin="1167" ymin="588" xmax="1580" ymax="1090"/>
    <text id="0000001e" xmin="529" ymin="327" xmax="574" ymax="411">ちょっと待ってよ</text>
  </page>
```

Figure 8: Manga109 annotated format

```
{
    "id": 92,
    "image_id": 6,
    "category_id": 2,
    "bbox": [
        217,
        341,
        284,
        392
    ],
    "area": 3417,
    "segmentation": [],
    "iscrowd": 0
},
{
    "id": 93,
    "image_id": 6,
    "category_id": 0,
    "bbox": [
        943,
        665,
        1003,
        708
    ],
    "area": 2580,
    "segmentation": [],
    "iscrowd": 0
},
{
    "id": 94,
    "image_id": 6,
    "category_id": 1,
    "bbox": [
        904,
        553,
        1567,
        1094
    ],
    "area": 358683,
    "segmentation": [],
    "iscrowd": 0
},
```


Figure 9: COCO annotated format

Additionally, Detectronv2 requires the bounding regions to be in X, Y, Width, Height (XYWH) format. Manga109 bounding regions come in Xmin, Ymin, Xmax, Ymax (XYXY) format. The documentation was unclear about this and we initially set the COCO metadata to allow for XYXY format since that is what we provided. It turned out to be an issue and we ended up converting everything to XYWH format to train it properly.

4.1.3 Training Results

Figure 10: Training results of the panel and text detection model.

4.1.4 Exporting

Because Detectronv2 is hefty, unconventional, and only officially supports Linux systems, making users download it to run the model locally does not follow the projects local desktop application philosophy. Due to this, we also decided to go with the CPU version of the model as that would be available for all clients. Detectronv2 is essentially a library and wrapper

for pytorch, so it is possible to export models trained with Detectronv2 to other formats to loosen dependencies. Detectronv2 offers functions to easily perform this exportation, but we found it not trivial at all. Only one exportation method, Caffe2Tracer was able to run.

Tracing is the act of feeding an input to the model and following the computations a model performs to rebuild the model in another platform. Because Detectronv2 does some pre-processing transforms on the input images before feeding into the model, we had to look through the source code and remake these transforms in order to trace the model to Caffe2. Exporting the model to Caffe2 removed the Detectronv2 dependency.

4.1.5 Deployment

With the Detectronv2 dependency removed, the model is now freely able to run with just pytorch dependency. Since the frontend is written in Javascript using Electron, we decided to make a standalone python Flask server to host the machine learning computations. The server receives an image in base64 format and performs the region proposals, tokenization, lemmatization, dictionary lookup, and translation. The results are sent back in JSON format for the frontend to freely parse.

4.2 OCR

4.2.1 Implementing OCR

OCR is an important factor of our project and it is something that is at the root of the project. When the images go through the model, it outputs the text and panel region on each page. Then, we can apply OCR on the text region to extract the text before moving forward to other processing steps.

For our OCR we used Pytesseract and with a vertical Japanese dataset [5]. A manga layout may have horizontal text but the actual text bubbles which are our regions of interest all contain vertical text only. We could not use a standard horizontal dataset due to the orientation of the detection.

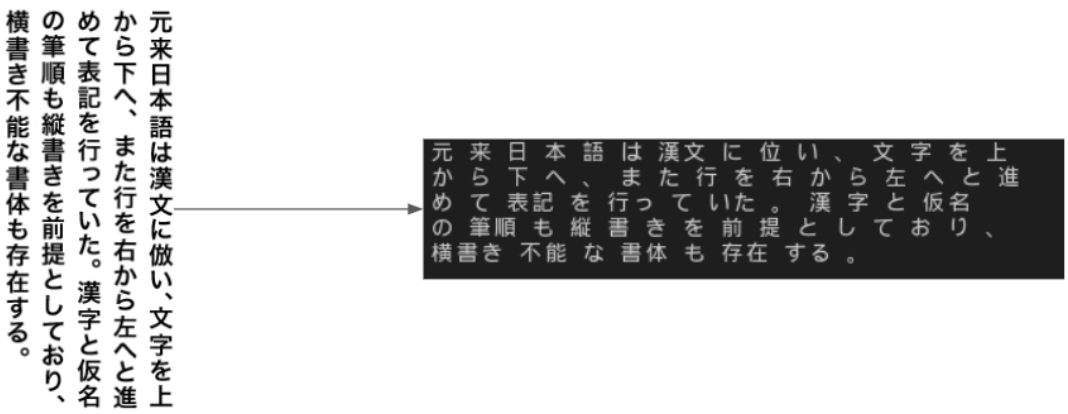

Figure 11: On the left is a vertical text (mimicking a highlight text that a user would do) and on the right side is output of the OCR model

4.2.2 Dealing with Furigana

One big problem with OCR on furigana is that the OCR model we used does not rigorously include furigana in the training set. Thus, any vertical text with furigana may have an undesirable output. The solution that we considered involved getting the average RGB value of vertical columns to detect regions of text lines. A text threshold would be set to determine whether or not it is a white pixel array or an array with characters. With threshold set, we would then determine the length of each subset of character column, and because furigana would be significantly smaller in width than the regular text, we would then set another threshold to determine the furigana, where we would eliminate them from the text.

However, there were some problems encountered with this method of detecting furigana:

— Threshold value may vary and even with statistical analysis, the threshold is not perfect for every manga. Scan quality effects the colors seen in text and pages.

— Sometimes the furigana text has no pixel gap between itself and the original text.

— Cases where the text-threshold cuts through a column of text because it believes that the average value of the column is too high ('high' where 255=white). This occurs due to the intrinsic left/right stroke heaviness in some characters.

Our solution is different for some cases but most are very simple at the moment. We tried grayscaling the image so that the array values are more defined. We also tried to upscale the image to get more pixels so that the OCR can detect better with larger resolution and higher quality. These method did improve some cases but overall there still needs to be work done on this part. For future work we would believe that implementing some sort of full furigana removal would work, model such as this [8] is something that can be implemented to improve the overall process.

5 Desktop Application

The user interface is designed to be interacted offline in the form of a desktop application. For this use case, we decided on using the Electronjs framework for building the desktop application, and the React framework for building the UI components of the application. Currently as the system is implemented, it requires no internet connection. Internet connection to a remote server can be used to offload the computations and storage space away from the clients computer.

5.1 NodeJS

The backend of our application is in NodeJS, as it is the default backend packaged with the ElectronJS framework described in the next section. NodeJS primary purposes is to connect the frontend applications requests to the underlying OS and the ML server. It is responsible for sending ML inference requests to the flask server as well as controlling the desktop application window controls.

Additionally, it is responsible for communicating with the external flashcard application, Anki. An Anki plugin named Anki Connect make it possible to remotely send commands to the application [1]. It hosts a local server with a REST API to do CRUD operations on cards, decks, and application settings. The NodeJS backend is responsible for making these API calls to quickly generate flashcards on user input.

5.2 Electronjs

ElectronJS is a framework that allows the creation of cross platform (Windows, OSX, Linux compatible) desktop applications using the same technology stack as a full-stack application. It uses JavaScript to render the contents of the application box and is thus therefore compatible with every JavaScript front-end framework available, such as React or Angular. React is running the local server that is hosting the application contents and ElectronJS is projecting that website over to a frameless chromium window, which acts as our desktop application.

5.2.1 IPC

In order for Electron to communicate with our frontend and the OS, an inter processor controller (IPC) service had to be set up. Essentially this is the same as signals in an operating system, where Electron sends and receives signals on ports and does the appropriate trigger functions. This is used to make frontend components have OS functionality. For example, the folder directory button sends a signal to Electron that then requests the OS to open up an folder directory browser window. JavaScript by itself cannot do folder uploads by itself because it is intended for web applications and this would pose web security risks. Similar functionality is implemented for the frontend buttons for the minimize, maximize, or close application buttons. The IPC acts as a pipeline for sending commands from frontend, to backend, to the OS.

5.2.2 React Hooks

In order for keyboard inputs to be usable by the program, hooks needed to be set up to wait on user input and asynchronously do some functionality in the frontend. This was implemented and currently only being used for flipping through pages when a comic is loaded in. The keyboard button hook implementation is generic and allows for new keyboard shortcuts to be implemented trivially. An existing open source project was heavily referenced for this implementation [12].

5.3 Reader Page

The frontend portion of the project has largely only focused on the reader page, although other common application UI features were implemented for potential stretch goals or future work that we did not get to.

When loading a new comic, the Frontend uses the IPC to tell the underlying NodeJS to send the images as base64 to the ML server for inferences. The ML server returns a JSON object about the object inference it resulted with. If it is the first time the application has requested inference on the selected manga, then it stores the JSON inference results on the local machine. Next time the user requests to load the manga, the predetermined JSON results would be used. This allows for persistence between application run sessions and near instant load times on already inferenced mangas.

The JSON information gathered from the ML server are used to dynamically create React Components that overlay over the display coordinates of the text regions. When a user hovers over these text regions, a popup appears of the plaintext observed with OCR. The user is able to hover over each individual token to select it in order to create a dialogue to export it to an existing Anki flashcard deck.

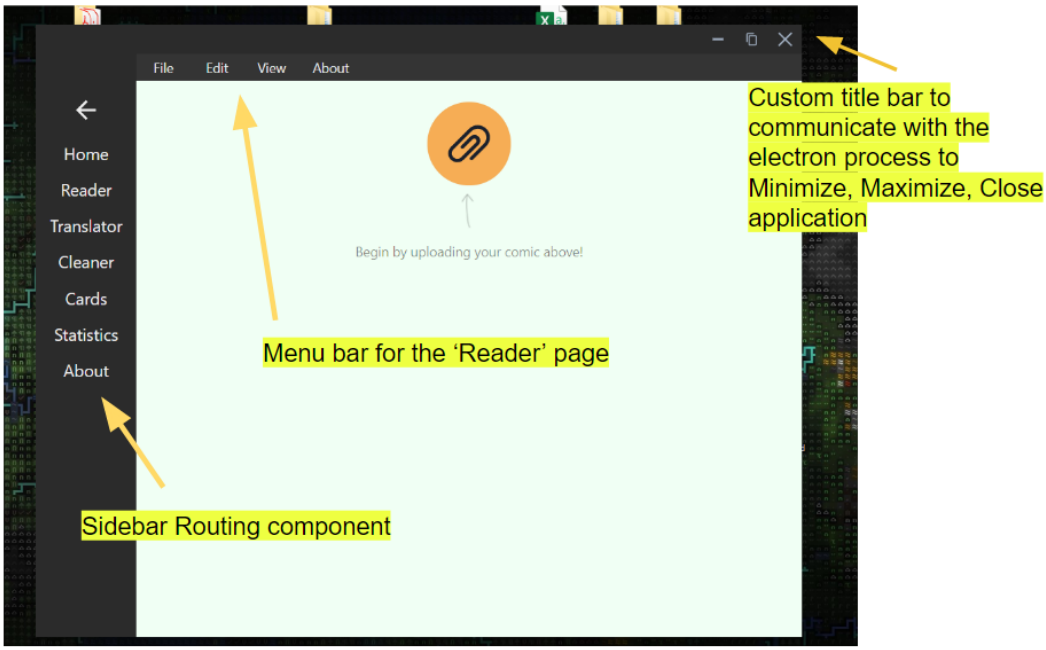

Figure 12: Basic UI navigation

Figure 13: UI when hovering over text regions

Figure 14: UI when clicking on one of the sentence tokens. (+) button for adding to Anki

Figure 15: Sample Anki card layout automatically created when clicking the add to Anki (+) button

6 Future Work

6.1 Improved Deployment

A major drawback of our implementation is that it is done on the client's computer. Object detection models, including our own, run very poorly on a CPU. To run a inference request on a AMD Ryzen 7 2700X Eight-Core Processor (3.70 GHz) takes 8 seconds. Ideally, a stand alone GPU ML server can be hosted using the same exact server code we provide to cut down the request time to less than a second. For an actual implementation the server would also need to run a job queue deployment structure, to provide multiple users with their annotated files.

6.2 Furigana Removal + Typesetting

Like we previously described, the furigana cleaning is still a problem for our application, and we think that doing more image processing would definitely help with the output of our OCR. There are some technique that this particular method uses, things such as binarizing image and then filtering it. These are work that can definitely be done beyond this current version.

If furigana removal is improved, then OCR becomes a lot more accurate. With high accuracy OCR it is possible to mask out the existing text region and typeset on top of the manga itself. This way, a user would hover over the words in the text box rather than opening a popup, which is much more intuitive and easy to use.

References

[1] 'FooSoft' Alex Yatskov. *Anki Connect*. https://github.com/FooSoft/anki-connect. 2016.

[2] *Foreign language classes speaking to interests of more comets*. URL: https://news.utdallas.edu/students-teaching/foreign-languages-rise-2021/.

[3] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. DOI: 10.48550/ARXIV.1311.2524. URL: https://arxiv.org/abs/1311.2524.

[4] Ross B. Girshick. "Fast R-CNN". In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

[5] 'zodiac3539' Gregory Choi. *Jap$_V$ert$_D$ataset*. https://github.com/zodiac3539/jpn_vert. 2018.

[6] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

[7] Ryota Hinami et al. *Towards Fully Automated Manga Translation*. 2021. arXiv: 2012.14271 [cs.CL].

[8] 'johnoneil' John O'Neil. *MangaTextDetection*. https://github.com/johnoneil/MangaTextDetection/blob/master/furigana.py. 2021.

[9] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[10] Toru Ogawa et al. *Object Detection for Comics using Manga109 Annotations*. 2018. arXiv: 1803.08670 [cs.CV].

[11] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[12] Arthur Tyukayev. *use-keyboard-shortcuts*. https://github.com/arthurtyukayev/use-keyboard-shortcut/tree/master. 2021.