INTRODUCTION

Our project is OCR for Japanese manga! We take manga text bubbles, segment characters into subimages, classify each subimage into machine-readable characters, and combine characters into sentences. We can even translate entire sentences to English with a simple API call. The user will be able to highlight the text boxes themselves to reduce the complexity of our project.

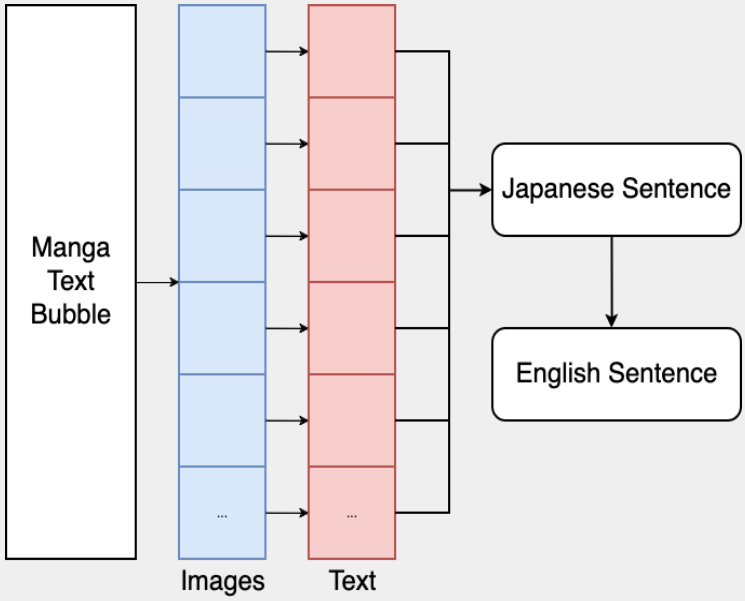


Fig 1. Our proposed implementation

LITERATURE REVIEW

Optical character recognition (OCR) is a field of ML that focuses on discerning information about characters encoded in a non-traditional format (ex. not UTF-8), such as in images, by using pattern and feature discrimination and classification [1]. OCR has a number of classification methods, including artificial neural networks (feedforward networks, recurrent neural networks, convolutional neural networks), Kernel methods (support vector machines, Kernel Fisher discriminant analysis, Kernel principal component analysis), statistical methods (logistic regression, linear discriminant analysis, hidden Markov models, K nearest neighbors, decision trees), template matching techniques, and structural pattern recognition (chain code histogram) which typically depend on large datasets with enough character variety to allow for “a meaningful comparison” [1]. The

dataset being considered for the problem of OCR for Japanese manga is Manga109, which contains sets of manga pages and accompanying annotations for elements within, such as characters' names and, more importantly, the text for the speech bubbles [2]. Manga109 has been used to train an AI model to recognize and retrieve a manga based on a user's sketch of a character or panel contained within by using an objectness-based edge orientation histogram, approximate nearest neighbor search methods, image processing techniques, and more [3].

Binarization is a main step in the processing part of OCR, where an image is turned into a binary image that contains only black and white pixels. This is done through finding a threshold that will make the pixels with an intensity higher than it turn to white pixels and the other pixels turn to black removing much of the noise that comes in images. Following this step is the deskewing step in which the new binary image is projected horizontally creating a histogram. This histogram is then rotated at different angles to calculate the variance and then find the angle with the greatest variance, after identifying that angle the image is then deskewed by rotating the image by that angle [4]. Morphology is a technique that analyzes structures within an image and this is used to isolate characters and determine what direction each line of text is meant to be interpreted from then it removes any parts that are not a part of the characters and rearranges the new simplified characters back into the text lines in the correct direction [5].

Tesseract OCR Engine is an optical character recognition system that integrates advanced techniques in connected component analysis, line and word segmentation, and adaptive classification delivering high-accuracy text recognition results [6].

Residual Network, or ResNet, is a deep learning architecture in which the weight layers learn residual functions with reference to the layer inputs. It has won the ILSVRC (ImageNet Large Scale Visual Recognition Challenge) 2015 in image classification, localization, and detection. One of the reasons why it won was because it managed to tackle several issues that the plain networks had. Several issues that the plain networks had were when its network was deeper or the layers increased, gradients would explode or vanish. This occurs during backpropagation where it multiplies the number n of these small (vanish) or large (explode) numbers to compute gradients of the front layers in a n-layer network. The way that

ResNet handled this problem is by using the skip shortcut connection that allows it to fit the input from the previous layer to the next layer without having to modify the input. Even if a vanishing or exploding gradient takes place, it always has the identity x to transfer back to earlier layers [7]. Shortcut connections simply perform identity mapping and their outputs are added to the outputs of the stacked layers. They add neither extra parameter nor computational complexity [8]. This would be a great character classification model for our OCR implementation because of how it prevents gradients from exploding or vanishing. This allows the character classification learning to become more efficient, be able to extract relevant features, and better generalize to new characters. Being able to handle variability in characters is crucial because it must distinguish between two characters that look almost identical but aren't the same. If that problem was not addressed, the character classification accuracy wouldn't improve that much and would have a higher validation error.

Vision Transformer (ViT) is a model that applies Transformer architecture to image recognition tasks, facilitating the integrating of global information through self-attention mechanisms. Its high computational efficiency makes ViT a great model to use for OCR where both accuracy and processing speed are critical. Furthermore, when combined with Convolutional Neural Networks (CNNs) such as ResNet to form a hybrid architecture, the resulting hybrid model achieves a higher accuracy than using either ViT or ResNet independently [9]

DATASETS

Manga109: A dataset of 109 manga which are annotated with face, body, character, and (importantly for us) text.

CC-100: A collection of datasets for constructed sentences of multiple different languages, scraped from the internet, including Japanese which we will use to create samples. This allow class frequency for character classification be naturalistic.

Fonts: We downloaded 63 free fonts which are most likely to resemble fonts in Japanese manga.

PROBLEM DEFINITION

PROBLEM

The problem is to extract text from manga in the browser.

MOTIVATION

Learning Japanese is very hard (source: Noah and Ryan). One of the best ways for second language acquisition is through immersion and to pair this with daily rote memorization (typically through something like Anki). One of the best ways to study Japanese naturally is with a tool called Yomitan, a browser extension equipped with a Japanese-to-Japanese dictionary, a Japanese-to-English dictionary, and support for automatic Anki card generation. While it's rather easy to find free Japanese media online, it's very hard to extract text so that learners can use Yomitan. Our project aims to convert Manga images to plaintext such that this tool can be used in the browser.

METHODS

Our working implementation is with Google's OCR Eng
bubbles from the Manga109 dataset then use Tesserac
readable text.

DATA PREPROCESSING

Manga speech bubbles are quite noisy. The pages ma
manga could be included in our text bubble images. Pr
the most salient information (the characters) from the i
pipeline which does following:

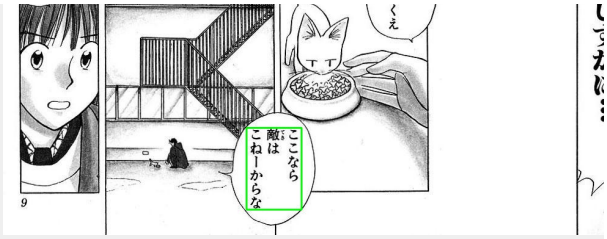


Image Source: "Hisoka Returns!" from the Manga1
Annotated textbox is highlig

Scale the image up: When the image is too small, futu
For example, when performing morphological operatio
1x1 kernel literally doesn't do anything. The problem is
We could mimic this behavior by doubling the size of th
Furthermore, a lot of the text in Manga109 is very sma
dramatically. We invert the color as well because open
background for morphology.

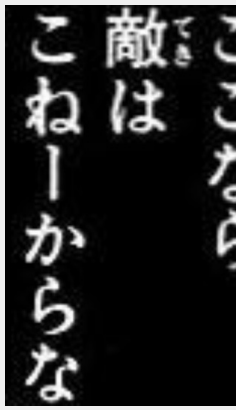
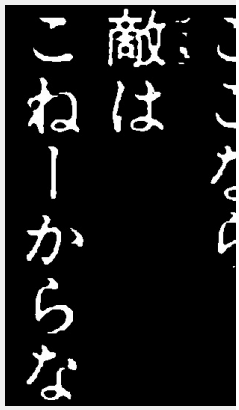


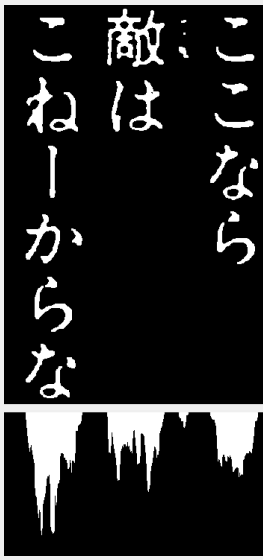
Image thresholding: Operating on a binary black and w
image. This makes detecting characteristics of the ima



Morphology: We apply morphology onto the images to
the text bubbles (either originally there, or as a result fr
performance of our models. We apply an opening morp
isolate the characters and to remove stray pixels betwe

Character segmentation: We segment the text bubble
character. This reduces the complexity of inferencing r
making histograms for black pixel frequency along colu
given range of columns indicates those columns conta
that line of characters, we generate a horizontal histog

most characters can fit in the same box space, we find
height to determine where characters fit. This helps wh
characters like こ.



Deskewing: We rotate the image a little bit then calcula
frequencies. A cool fact about this ("I have a truly marv
which this margin is too narrow to contain"), the histogr
properly aligned. We have implemented this but have r
implementation. For starters, we cannot find any exam
verify its nonexistence since there are 147887 textboxe
importantly, it hurts performance in cases where the te
Example:

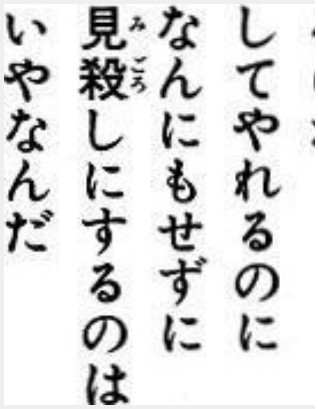
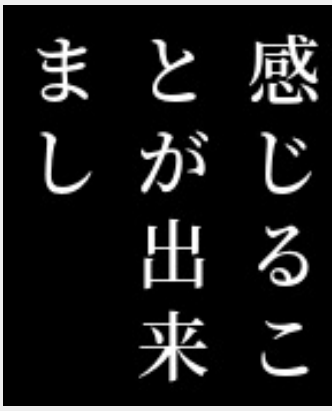


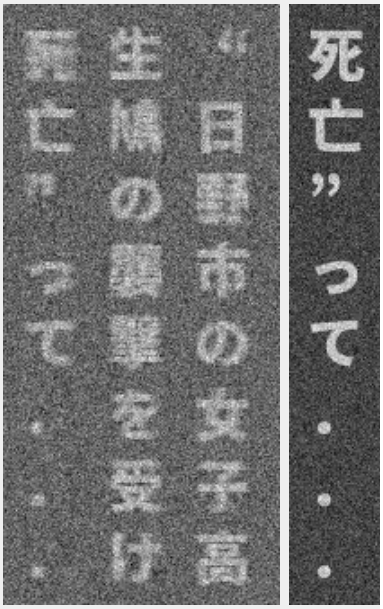
Image Source: "Hisoka Returns!" from the Manga1

SYNTHETIC DATA GENERATION

CC-100 contains a large amount of data of Japanese t
while this is certainly helpful for what we need to do, it
hand. To train our models, what we need are images a
image). For example, if we have the character “夜” fror
we also want the image containing the character. For s
something similar, but more representative of how mar
need both sample sentences and sample images that
example, given the text “とを肌で感じることが出来ま
well, like so:



To accomplish this, the plaintext data from CC-100 was
images and sentence images. For the character image
rendered into an image using a random Japanese text
random range between 1 to 16 was used to determine
and these characters were then rendered into an imag
(start top-left, write to bottom-right). For the Tesseract i
perfect, with character segmentation being extremely e
noise present. As such, SVD image compression and/o
on the images, creating results that look like the follow



Left: Gaussian noise. Right: Compressed with SV
generator.

TESSERACT MODEL

We run the text bubble through our preprocessing pipe
each character. Our preprocessed character subimage
we invert the subimages once more since Tesseract ex
found online which were generated by training on verti
configured Tesseract to increase performance on singl
just parameters to the model). Tesseract generates a b
the text. We only accepted text which had over 30% co
characters and then combine all characters in reading-
a given text bubble.

```
def get_params():
    """
    Returns:
        (string): Configuration for Tesser
    """
    params = ""
    params += "--psm 5 "
    params += "--oem 3 "
    params += r"--tessdata-dir ../lib/weig
    params += r"--loglevel DEBUG "
    configParams = []

    def configParam(param, val):
        return "-c " + param + "=" + val
```

RESNET MODEL

We made a ResNet-18 model that takes in grayscale i
grayscale images of single characters and it outputs w
google colab and jupyter notebook to train our model d
from google colab and debugging on jupyter notebook.
day to fully train the data, while on colab it took approx
data, we used cuda to test the model with the Manga-1
complete the testing.

Vision Encoder Decoder Transformer

Our third model is a Vision Encoder Decoder Transforr
approach are through the transformers python packag
Image Transformer (DeiT) [10] as the encoder and a p
decoder. Both of these are with default configurations.
synthetic sentences--around 100k of them. This finetur
Nvidia GTX1070. Training was handled through the Se
package. Weights, metrics, and visualizations were up
though these weren't very interesting. We only realized
finished training and did not implement any features to

After finetuning our model, we tested our model on the
preprocessing or segmentation when testing our mode
operate as a de-facto preprocessor.

Our hyperparameter tuning strategy is nonexistent (as
scary!). Jokes aside, this model was bottlenecked by a
compute or time to perform hyperparameter tuning. We
from the original DeiT and BERT papers for our specifi

RESULTS

Metrics for string comparisons are non-trivial, especially when strings can be of different length. The metric we used is Levenshtein distance per predicted character. A Levenshtein distance is the number of operations (insert, delete, replace) to convert string A to string B. Here is an example on how to interpret this:

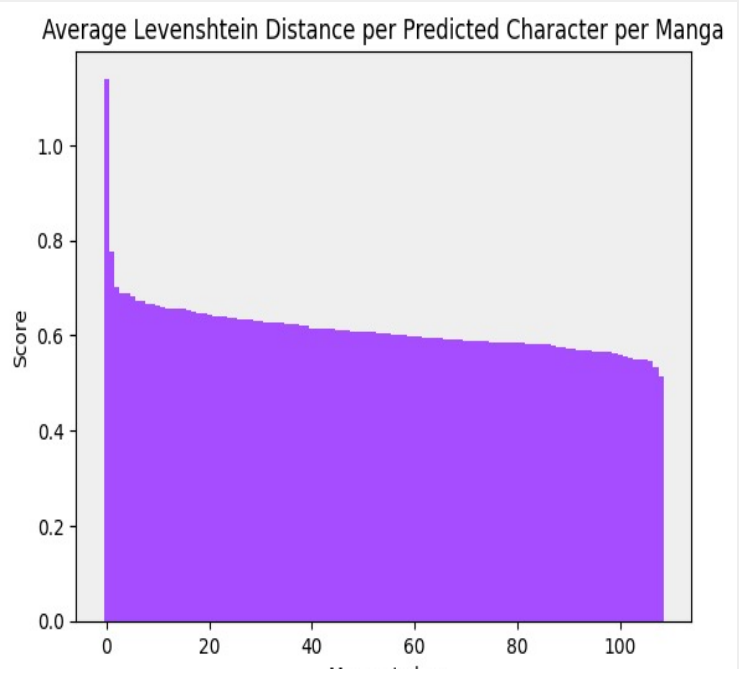
Predicted Text	Actual Text	Levenshtein Distance per Character
"abc"	"abd"	0.33
"ab"	"abcd"	1.00

We also experimented with BLEU score, a popular metric for string-to-string comparisons. We found this metric is too volatile for our model since our model underguesses characters. BLEU score is calculated by comparing multiple n-grams, but since the predicted text is commonly shorter than the actual text and the textboxes don't contain much text, our BLEU score commonly shortcuts to 0. Example:

Predicted Text	Actual Text	BLEU Score
決めた 芸術だ	よし！決めた！！芸術だ！！	0.00

The only thing meaningful text Tesseract did not predict was "よし". In English, the differences between these two strings are "It's decided it's art" and "Yeah! It's decided!! It's art!!", so a BLEU score of 0.0 seems too harsh.

We ran our model on all manga in the Manga109 dataset:



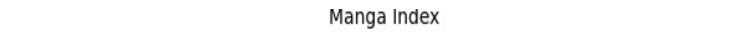


Fig 2. Tesseract model results on Manga109 dataset. Mean: 0.614. Standard Deviation: 0.064.

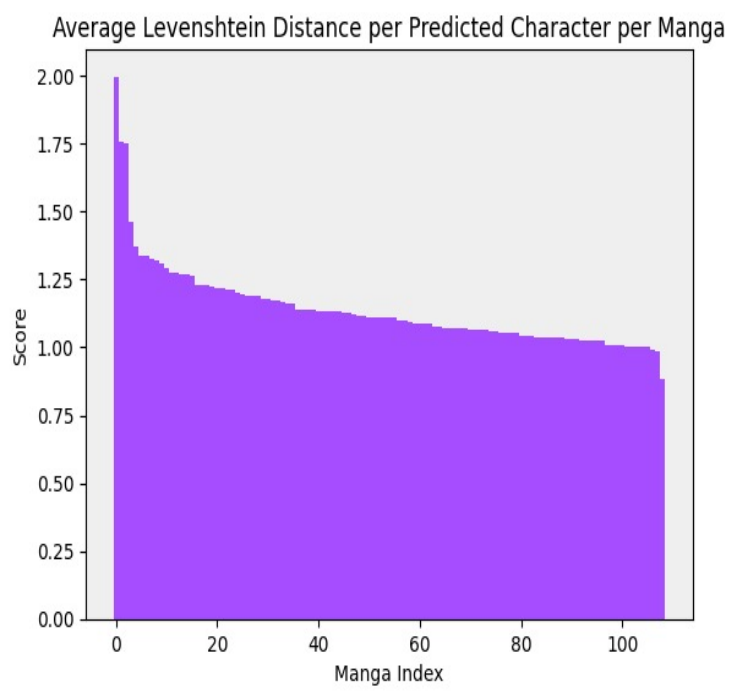


Fig 3. ResNet model results on Manga109 dataset. Mean: 1.140. Standard Deviation: 0.154.

Notably, the ResNet achieved an accuracy of 98.0% when training on character subimages with a loss of 0.1451 after one epoch.

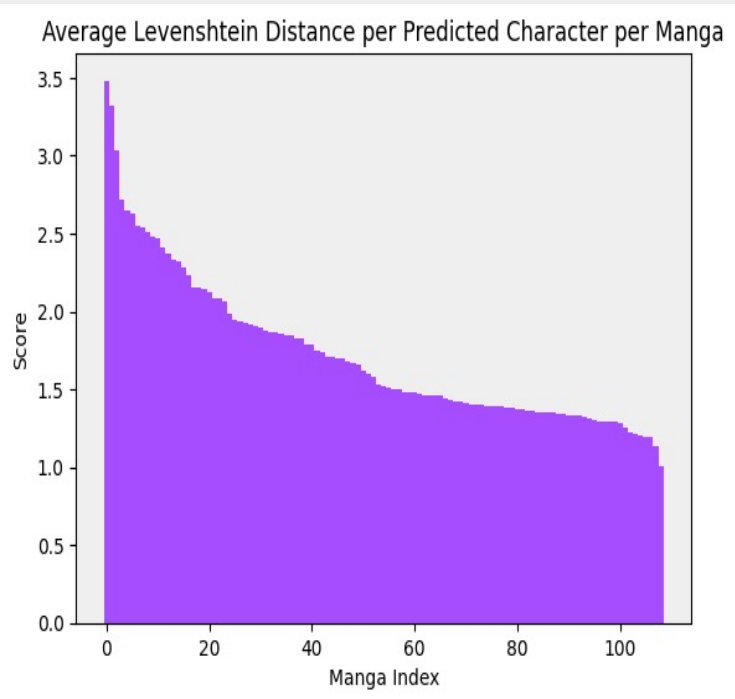


Fig 4. Vision Encoder Decoder Transformer model results on Manga109 dataset. Mean: 1.721.

Standard Deviation: 0.477.

Our loss for this vision encoder decoder transformer was logged through wand. We only did a few validation steps so we cannot, with fine granularity, see what the loss fucntion looks like. Despite that, the loss function was monotonically decreasing, so it seems the model was, in fact, learning.

Model	Mean Levenshtein Distance	Std.
Tesseract OCR	0.614	0.064
ResNet-18	1.140	0.154
Vision Encoder-Decoder Transformer	1.721	0.477

The Tesseract model had a 0.614 Levenshtein distance, which is not great, but with reference to the other two models it's the best. The ResNet had a 1.140 Levenshtein distance, which is pretty bad, but it was relatively consistent with a standard deviation of 0.154. The worst model by far was the Vision Encoder-Decoder Transformer, which had a horrendous 1.721 Mean Levenshtein Distance, with a standard Deviation of 0.477. The Tesseract model is far better than the other two models, but that is most likely due to our lack of resources in testing and working with the other two. The ResNet performed alright in comparison, and was relatively consistent across the board other than a few outliers. The Vision Encoder-Decoder was incredibly inconsistent, and will need far further improvements in order for it to be applied.

DISCUSSION

On a whole, the Levenshtein distance had a mean of 0.614, which is clearly not an ideal score, but is workable as we tune Tesseract and improve our techniques; what is less workable is the fact that one manga had a Levenshtein distance greater than one. The culprit is a manga called Joouari, and the following is a panel from the manga:

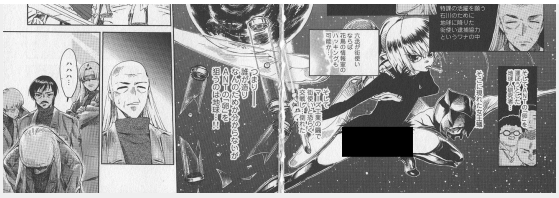


Image source: "Joouari" from Manga109 dataset. Courtesy of Ooi Masakazu.

While it may be difficult to tell:

These three boxes are written differently than the other text boxes on the page, with text being written top-left to bottom-right. Previously, we specified that our training data would follow the standard paradigm of vertical writing present in manga, but ultimately, this does not account for left-to-right writing, and we did not account for this in the Tesseract implementation either. Upon further inspection, these sorts of text boxes were found throughout the manga, giving probable explanation as to why a Levenshtein distance over 1 was obtained. This manga also has very dense character blocks in its text boxes, many white-on-black text boxes (the expected is black-on-white), and other elements that could also contribute to this error.

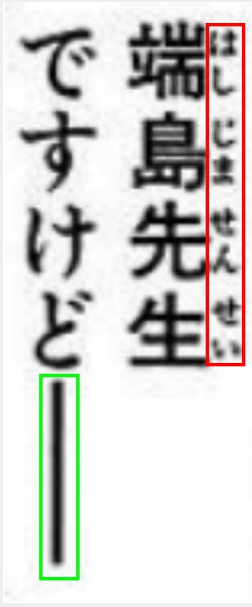
The ResNet model could’ve been a bit better with the data that was given for training. Our training results with the model had a loss of 0.1506 and an accuracy of 0.9792. The accuracy being very high may at first glance, seem like the model is performing very well. However, this also means that the model could have been overfitting due to the data not being regularized enough, so we don’t know for sure if the accuracy is correct, which is

why we tested with the Manga-109 dataset. After testing with the Manga-109 dataset, we had an average Levenshtein distance of per predicted character of 1.140 with a standard deviation of 0.154. It's an okay score but not the greatest as it suggests that the character detection is always off somewhere since it has to make at least one operation with comparing the predicted and actual characters.

The Vision Encoder-Decoder Transformer was our worst model by far. We think it the poor results stem from the following reasons: poorly regularized data, not enough data, not enough time, not enough fonts, synthetic data not being representative of manga panels, not enough compute, etc. We think this model is, nonetheless, appealing for future work. From the few validation steps we ran, as visualized through wandb, the model seemed to be learning, but we did not give it enough time.

NEXT STEPS

There are currently a lot of issues with how we are processing data, and a lot of areas where we can improve. Here is an example of a very common form of stylized writing which hurts the performance of our preprocessing pipeline.



Highlighted with green bounding box: kanji which spans multiple characters. Highlighted with red bounding box: furigana. Since Japanese kanji is not phonetic, furigana is common in Manga to show the reader how to pronounce some words. These do not add additional information and are less common in forms of media with older audiences.

Our segmentation is also far too specific. While the

histogram method is working well, it doesn’t consider anything else when deciding where lines and characters appear. All we do is check if there’s no white pixels in a column and if there isn’t there's probably a column to the left and right. This means that Furigana may or may not be counted in a column, or (incredibly rarely) a column may be counted as two. Not to mention, we don’t consider horizontal or skewed Japanese at all.

We’re going to need significant changes in our algorithm to fix these issues, most of which will be based on the methods used in “Detection of Furigana text in images” [12]. First, rather than take in raw image data, we can first extract a text mask using the Manga Image Translator (MIT) [13]. We can then use DBSCAN to group areas of text, and distinguish between Furigana and the rest of the text, by size and shape. We can also distinguish between vertical and horizontal text in this step by their height/length ratio.

Then, once we have an image without Furigana, post deciding whether it’s vertical or horizontal, we can split the characters. Since length and width of a text box is the same, we should use width for a horizontal box, and width for a vertical one, because “long” characters wouldn’t span across lines. Then if a box’s edge happens to lie on an area with text, it’s probably a long character, so we can combine it with the next box, and then ignore both, or (incredibly theoretically, beyond our scope) apply the tone of the “long” character to wording.

If we were tasked with making one model as good as possible, we would choose the vision encoder decoder transformer. With a lot of "good" data, this approach seems to be state-of-the-art in the field. This means we would have to massively scale up our synthetic data generation and probably use Georgia Tech's PACE.

REFERENCES

[1] J. Memon, M. Sami, R. A. Khan, and M. Uddin, "Ha
(OCR): A Comprehensive Systematic Literature Reviev
142642–142668, 2020.<\p>

[2] Y. Matsui, K. Ito, Y. Aramaki, T. Yamasaki, K. Aizawa
Manga109 Dataset." CoRR, vol. abs/1510.04389, 201

[3] Aizawa Yamasaki Matsui Lab, “Manga109: Japanes

2024. [Online]. Available: http://www.manga109.org/en

[4] S. Reddy, “Pre-Processing in OCR!!!,” Medium, Jul.
https://towardsdatascience.com/pre-processing-in-ocr-

[5] N. K. Bjerregaard, V. Cheplygina, and S. Heinrich, D
https://arxiv.org/pdf/2207.03960 (accessed Nov. 7, 202

[6] R. Smith, “An Overview of the Tesseract OCR Engi
https://static.googleusercontent.com/media/research.g

[7] S.-H. Tsang, “Review: Resnet - winner of ILSVRC 2
detection),” Medium, https://towardsdatascience.com/r
classification-localization-detection-e39402bfa5d8 (acc

[8] K. He, J. Sun, S. Ren, and X. Zhang, Arxiv, https://a
7, 2024).<\p>

[9] A. Dosovitskiy et al., “AN IMAGE IS WORTH 16X16
IMAGE RECOGNITION AT SCALE,” arXiv.org, https://a
7, 2024<\p>

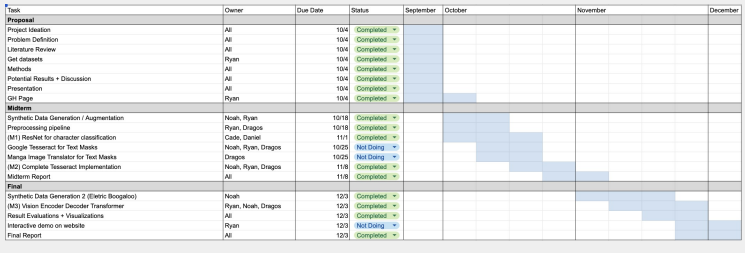
[10]Touvron, Hugo, et al. “Training Data-Efficient Image
Attention.” arXiv.Org, 2021, arxiv.org/abs/2012.12877.

[11]Devlin, Jacob, et al. “Bert: Pre-Training of Deep Bid
Understanding.” arXiv.Org, 24 May 2019, arxiv.org/abs

[12] zyddnys, “Image/Manga Translator,” GitHub, Aug.
https://github.com/zyddnys/manga-image-translator<\p

[13] Nikolaj Kjøller Bjerregaard, Veronika Cheplygina, a
Furigana text in images” arXiv.org, Jul. 2022, https://ar

GANTT CHART



Task	Owner	Due Date	Status
Proposal			
Project Ideation	All	10/4	Completed
Problem Definition	All	10/4	Completed
Literature Review	All	10/4	Completed
Get datasets	Ryan	10/4	Completed
Methods	All	10/4	Completed
Potential Results + Discussion	All	10/4	Completed
Presentation	All	10/4	Completed
GH Page	Ryan	10/4	Completed
Midterm			
Synthetic Data Generation / Augmentation	Noah, Ryan	10/18	Completed
Preprocessing pipeline	Ryan, Dragos	10/18	Completed
(M1) ResNet for character classification	Cade, Daniel	11/1	Completed
Google Tesseract for Text Masks	Noah, Ryan, Dragos	10/25	Not Doing
Manga Image Translator for Text Masks	Dragos	10/25	Not Doing
(M2) Complete Tesseract Implementation	Noah, Ryan, Dragos	11/8	Completed
Midterm Report	All	11/8	Completed
Final			
Synthetic Data Generation 2 (Elvtric Boogaloo)	Noah	12/3	Completed
(M3) Vision Encoder Decoder Transformer	Ryan, Noah, Dragos	12/3	Completed
Result Evaluations + Visualizations	All	12/3	Completed
Interactive demo on website	Ryan	12/3	Not Doing
Final Report	All	12/3	Completed

CONTRIBUTION TABLE

Name	Primary Contributions
Ryan	Website, Tesseract, Vision Encoder Decoder Transformer

Noah	Synthetic Data Generation
Dragos	Preprocessing Pipeline, Vision Encoder Decoder Transformer
Cade	ResNet
Daniel	ResNet

PRESENTATION

Fig 5. How to center a div