UCLouvain

École polytechnique de Louvain

Unlocking the ancient history of Japan, hidden behind kuzushiji

Participating in the Kuzushiji Recognition Competition on Kaggle

Author: **Vladislav KUCHERIAVYKH**
Supervisor: **Charles PECHEUR**
Readers: **Samy BETTAIEB, Sébastien JODOGNE**
Academic year 2023–2024
Master [120] in Computer Science

Abstract

Kuzushiji is the cursive writing style of pre-modern Japanese. Millions of books and historical documents written in kuzushiji remain undeciphered. To address this challenge, we developed a two-stage optical character recognition (OCR) pipeline. The first stage, a text detector based on the Co-DETR architecture, locates and classifies individual characters into broad categories. The second stage employs a Mixture-of-Experts approach, utilizing specialized SVTR models to recognize characters within each category. The pipeline was trained on the Kuzushiji Recognition dataset from Kaggle. Evaluation on the test set demonstrates promising results, with high F_1-scores in text detection, character categorization, and character recognition. The model's ability to handle diverse challenges, such as character variations, faded text, and complex layouts, is showcased through qualitative analysis. This research contributes to the ongoing efforts to unlock the valuable historical and cultural knowledge contained within old Japanese texts.

Contents

Chapter 1

Introduction

Historical documents offer us a glimpse into the past, allowing us to understand the culture, norms, and values of bygone eras and reflect on our own. Japan's unique historical path, particularly its period of relative isolation during the Edo period (from 1603 to 1868), fostered a rich cultural landscape and a literary tradition that rivals any in the world. Yet, the modernization of the Japanese language following the Meiji Restoration (1868) left a vast trove of knowledge locked in a script most modern Japanese can no longer read: *Kuzushiji*.

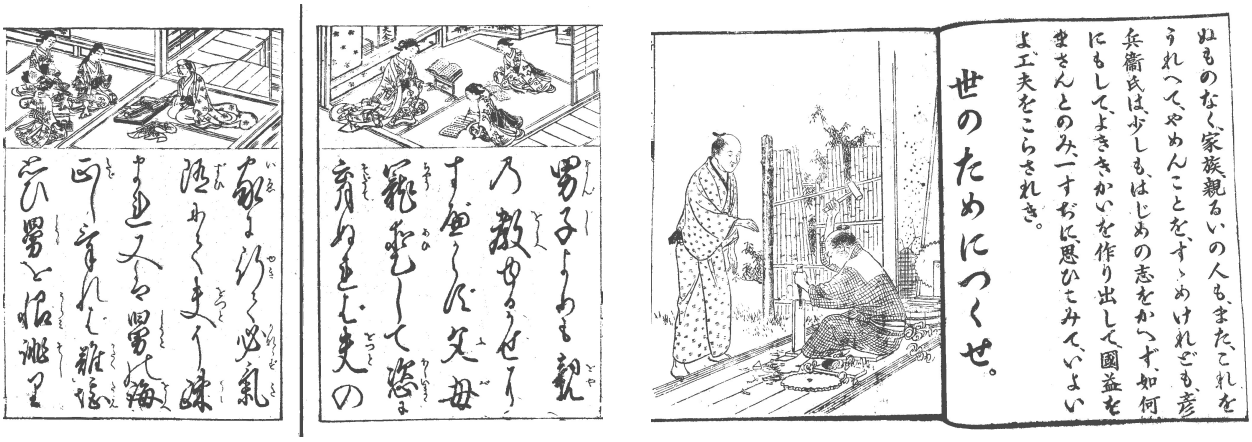

Figure 1.1: The difference between a text printed in 1772 (left) and one printed in 1900 (right). Adapted from [1, p. 2]

It is estimated that there are over 3 million books written or published before 1867 and a billion historical documents preserved nationwide in Japan [1]. Unlocking these texts could reveal untold stories and answer countless questions about Japan's past: from daily life and customs to historical events and artistic traditions. Imagine deciphering diaries detailing the eruption of Mount Fuji, or uncovering old traditional Japanese recipes passed down through generations. The world of samurai and ninjas, often romanticized in modern portrayals, could be explored in authentic detail.

The intricate and varied nature of kuzushiji, however, presents a significant challenge to deciphering these texts. This is where the power of deep learning becomes relevant. Advanced algorithms and neural networks, trained on vast amounts of data, can learn to recognize the nuanced patterns and variations in kuzushiji, potentially unlocking this hidden history on an unprecedented scale.

In 2019, a Kaggle machine learning competition was organized to foster collaboration and innovation in

recognizing and transcribing kuzushi text. This Master's thesis leverages resources and datasets made available through the competition to advance kuzushiji decipherment.

1.1 Objectives of the Thesis

The objectives of the thesis are analyzing the problem, becoming acquainted with the field, and developing a solution with a novel approach, using deep learning techniques. Achieving top-tier performance is not a primary objective.

1.2 Roadmap

After this introduction, Chapter 2 will provide background information on Kaggle competitions, the Kuzushiji Recognition competition, and previous solutions in the field, including details about convolutional neural networks (CNNs). The thesis then explores the competition dataset and the challenges inherent to kuzushiji in Chapter 3. The model architecture is detailed in Chapter 4, and the training procedure in Chapter 5. After which, an evaluation of the results obtained is proposed in Chapter 6. The thesis will conclude with a summary of the work accomplished and suggestions for future research.

1.3 Remarks

Where relevant, Japanese terms will be transliterated into Latin script using the revised Hepburn romanization system [2].

The expressions "kuzushi characters" and "kuzushiji", with "ji" meaning "character" will be used interchangeably.

Automated grammar tools and a Large Language Model (LLM) were used to revise writing style and consistency, and to develop plotting code for the graphs.

Chapter 2

Background

2.1 What are Kaggle and Kaggle Competitions?

Kaggle is an online platform for data scientists, machine learning engineers and researchers. It is most well-known for organizing and hosting machine learning competitions where users around the globe are invited to tackle a data science problem and develop statistical models whose performance can outperform that of the other contestants. In addition to the competitions that form the heart of the platform, Kaggle also provides other related tools and services like a repository of public data sets formatted for computer processing, mini-lessons and interactive in-browser notebook environments with access to free GPUs and TPUs (Tensor Processing Units).

2.2 Background of the Competition

The Kuzushiji Recognition competition [3] was launched on July 19, 2019 and closed on October 15, 2019. The Center for Open Data in the Humanities (CODH) of the Research Organization of Information and Systems in Data Science (ROIS-DS), based in Tōkyō, sponsored this competition. The five best teams were awarded a $3,000 research grant each.

The hosts who organized the competition were Tarin Klanuwat (CODH), Mikel Bober-Irizar (University of Cambridge), Alex Lamb (Mila[1]), and Asanobu Kitamoto (CODH).

They set up the competition because they wanted to collect algorithm ideas for future kuzushiji recognition models.

2.3 Previous Solutions

Although this is a niche domain, there already exists multiple solutions targeting this exact problem. These solutions can be categorized into three categories: solutions developed before the launch of the competition (pre-competition), solutions developed as submissions for the competition while it was active (in-competition) and solutions developed after the end of the competition (post-competition). A detailed survey on deep learning-based kuzushiji recognition had already been conducted by Ueki Kazuya, et al. (2020) in [4]. It will be frequently referenced throughout this section.

2.3.1 Pre-Competition

Prior to the Kaggle competition, several Japanese researchers had already explored automated kuzushiji recognition using various approaches:

[1] Mila is an AI and ML research institute from the Université de Montréal

Template Matching (Yamada Shōji et al., 2001 [5]) This method retrieves character patterns from a digital dictionary of historical characters and matches them to target images using a technique they called "matching by overlay." The width of the target image is normalized to match the width of a candidate pattern from the dictionary. Then, they use masking to limit the matching range to avoid issues with adjacent characters. The matching process continues until the distance between the two patterns falls below a certain threshold, indicating a match.

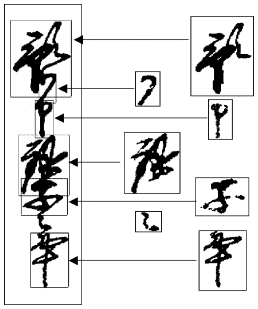

Figure 2.1: Example of pattern matching with the source text on the left and closest matching patterns on the right. Retrieved from [5, p. 15]

Modular Neural Networks (Horiuchi et al., 2011 [6]) This approach involves dividing the overall task into smaller, more manageable subtasks, each handled by a separate neural network module. These modules are then combined to form a complete system.

Self-Organizing Maps (Katō et al., 2014 [7]) This type of unsupervised learning algorithm helps to visualize and cluster high-dimensional data. In this case, it was used to group similar kuzushi characters based on their visual features.

Neocognitron (Hayasaka et al., 2015 [8]) Inspired by the structure of the human visual system, this hierarchical neural network architecture is designed for pattern recognition tasks such as identifying kuzushi characters.

Convolutional Neural Networks (Ueda et al., 2018 [9]) Convolutional neural networks (CNNs) are a class of deep learning models highly effective for image recognition tasks. Their ability to learn spatial hierarchies of features makes them well-suited for identifying complex patterns in kuzushi characters.

End-to-End Method with Attention Mechanism (Hu et al., 2019 [10]) This approach involves training a single model to directly map input images to output text sequences, with an attention mechanism that allows the model to focus on different parts of the input at different times.

The CODH Kuzushiji dataset [11], released in 2016 and extended in 2019, played a pivotal role in accelerating research in this area. This dataset, also used in the Kaggle competition, provided ample training data in a computer-readable format and a standardized benchmark for evaluating different models and techniques. The CODH team also developed KuroNet [12], a deep learning model based on a U-Net architecture, which is a type of CNN often used for image segmentation tasks. KuroNet, specifically designed for kuzushiji recognition, jointly predicts the location and identity of all characters within a page of text.

2.3.2 More on Convolutional Neural Networks

Before moving to the section discussing the competition submissions, it is helpful to clarify what CNNs are, since they play an important role in many of the submissions.

Convolutional neural networks (CNNs) are a type of artificial neural network specifically designed for processing grid-like data, such as images. CNNs use filters to identify patterns in the image, gradually learning to recognize more complex features like edges, corners, and eventually whole objects (Figure 2.2).

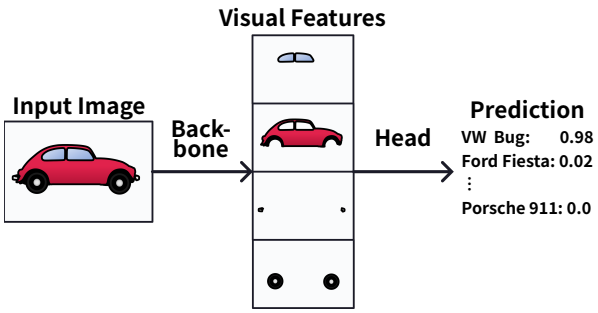

Figure 2.2: Simplified, intuitive overview of the visual features extraction. Adapted from [13]

These CNNs are typically used in the backbone of a backbone-neck-head architecture (Figure 2.3).

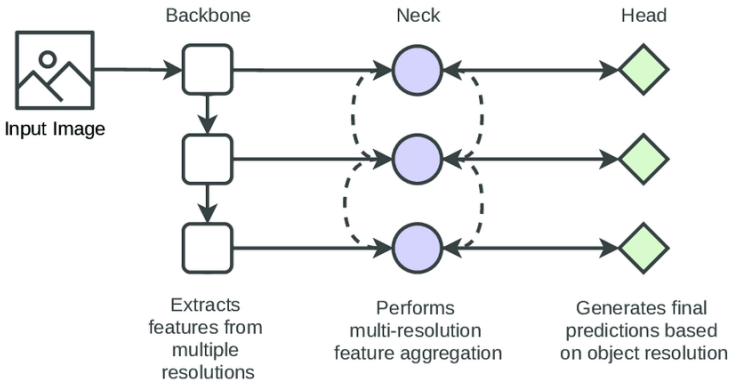

Figure 2.3: Diagram of a general backbone-neck-head architecture in computer vision. Retrieved from [14]

To clarify, the backbone is the core of the model, responsible for extracting features from the input image. It can be thought of as a series of filters that gradually transform the raw image into a set of feature maps. Each layer in the backbone extracts more abstract features, going from low-level (e.g. edges, corners) to high-level (e.g. object parts).

The neck optionally sits between the backbone and the head. Its role is to refine and combine the feature maps extracted by the backbone. Its primary focus is on addressing scale variation and aggregating information from different levels of the backbone. It does not necessarily create new representations of specific objects like wheels on a car. Without the neck's feature fusion capabilities, the model might struggle to detect objects at different scales. Small objects could be missed due to the lack of low-level features, and large objects might lack fine-grained details due to the absence of high-level features.

Finally, the head is the task-specific component that sits on top of the neck. It takes the refined feature maps from the neck and makes the final predictions. For example, for object detection, the head would predict bounding boxes around objects and their class labels while for image segmentation, the head would predict a mask for each object, classifying each pixel in the image.

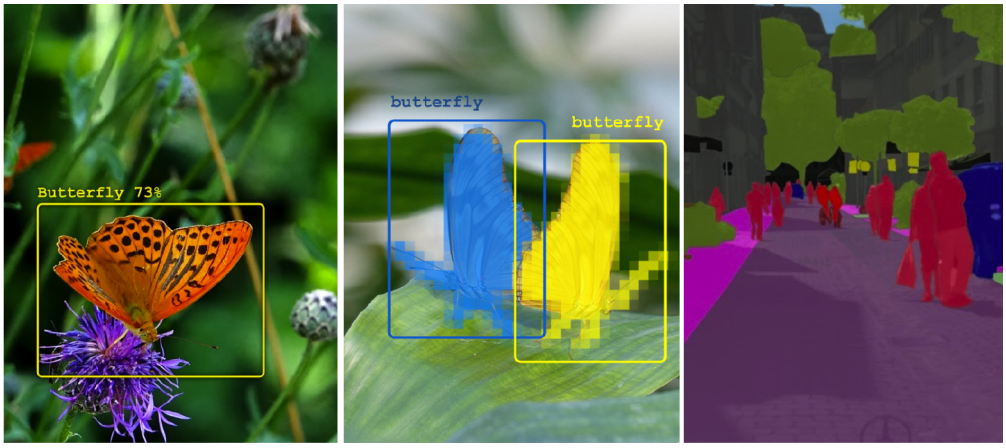

Figure 2.4: From left to right: object detection, instance segmentation, and whole-scene semantic segmentation. Images from Arthropods and Cityscapes datasets. Example retrieved from [15, p. 131]

2.3.3 In-Competition

This brings us to the Kaggle competition. The competition attracted 293 teams, and nine of the top fifteen teams (rankings 1, 3, 4, 5, 7, 8, 9, 13, and 15) shared their approaches. The following section is a summary of the teams' write-ups [2] and an analysis by Ueki et al. [4].

A common theme was the use of a two-step process, first identifying individual characters within the text and then classifying each character. This process heavily relied on convolutional neural networks.

Popular detection architectures included Faster R-CNN [16], a region-based CNN that proposes regions of interest (ROIs), which are areas in the image potentially containing objects. Faster R-CNN then refines these ROIs into accurate bounding boxes for character detection. Another popular approach was CenterNet [17], which directly predicts the center point and size of each object, providing a more efficient alternative to region-based methods. ResNet-based models were also frequently used for classification. ResNet [18] is a deep CNN whose main idea is its residual skip connections that allow gradients to flow more easily during training, enabling the training of much deeper networks and alleviating the vanishing gradient problem[3].

To improve model generalizability and address the challenge of character variations across different historical documents, teams employed various data augmentation techniques (artificially creating variations of existing data) and preprocessing methods to enhance model robustness and character visibility. Combining the predictions from multiple models (ensembling) was a strategy adopted by several teams to enhance overall accuracy.

The winning team achieved a F_1-score of 95% using a Cascade R-CNN [19] with HRNet (High-Resolution Net) [20] as the backbone network. They employed multi-scale training and testing strategies. As Ueki et al. point out [4], "the team was able to achieve a high accuracy while maintaining greater simplicity than the methods used by the other teams because the latest techniques were applied". This goes to show that even if it is common for object detection tasks to deal with only 80 classes or so, as is the case for the famous MS COCO dataset [21] used in object detection benchmarks, it is entirely possible to achieve competitive results despite a number of classes to detect in the order of thousands.

Other successful approaches included the third-place team's use of Faster R-CNN for detection, a combination of ResNet and ResNeXt [22] (a variant of ResNet with grouped convolutions) for classification, and an additional model for post-processing to filter out false positives. The fourth-place team stood out by employing a unique line-based processing method, utilizing a Hybrid Task Cascade (HTC) [23] architecture, which effectively combines instance segmentation with detection features for joint multi-stage processing. This was coupled with a Recurrent Convolutional Neural Network (RCNN) [24], a type of neural network designed for sequential data, which they used for line recognition. The fifth-place team achieved a comparable F_1-score of 94% with a modified single-stage CenterNet model, showcasing the effectiveness of this alternative architecture.

Other teams experimented with various techniques, including pseudo labeling[4], language models, and different backbones for their detection and classification models. Some teams found that deeper networks were not necessarily beneficial, while others reported limited success with language models — this is explored further in the section 3.3.8, discussing the scarcity of pre-modern Japanese text data.

Overall, the competition showcased a variety of effective approaches for recognizing kuzushi text. The winning solutions leveraged strong backbones, multi-scale training, and ensembling techniques. Data augmentation and preprocessing methods played an important role in all teams' approaches.

[2] See the last column of the final Kaggle leaderboard (https://www.kaggle.com/competitions/kuzushiji-recognition/leaderboard) for links towards the write-up of each of the mentioned teams

[3] The vanishing gradient problem arises during the training of deep neural networks when gradients (error derivatives used in gradient-based learning methods and backpropagation), used to update the weights, become extremely small as they get propagated through layer after layer, and thus prevent effective learning.

[4] Pseudo labeling is a semi-supervised learning technique where predictions from a model on unlabeled data (typically with high confidence) are treated as ground-truth labels and added to the training dataset. This effectively increases the amount of labeled training data, and lets the model get exposure to the test set data.

2.3.4 Post-competition

After the competition, a few commercial products, aimed at the general public, were developed including "miwo" [25], and "fuminoha" [26]. Both are mobile applications that send the user's image to a server running an inference model. Fuminoha is developed by the TOPPAN company while miwo is developed by the same group who organized the Kaggle competition. Backend information for fuminoha is unavailable but it is said that miwo was initially running the KuroNet model but after collecting ideas from the competition, the ROIS-DS Center for Open Data in the Humanities (CODH) developed a new AI kuzushiji recognition system called RURI, which replaced KuroNet.

According to the developers,

> In comparison to the previous AI kuzushiji recognition model, RURI's AI object detection technology has been optimized for kuzushiji recognition, and the performance has been improved for the cases when the color or the pattern of the background is complex. In terms of the target of kuzushiji recognition, the performance is best for woodblock-printed[5] books in the Edo period, but it can also be used for old hand-written documents.

(Source "miwo: App for AI Kuzushiji Recognition"[25])

For high-speed text conversion of a huge number of images on a server, there also exists the open-source NDL Koten OCR program [27].

The model consists of three main components:

Layout Recognition Model This model identifies areas within an image that contain text to be converted. It uses the Cascade Mask R-CNN architecture (an extension of Cascade R-CNN that adds a mask head for instance segmentation) and is trained on datasets of pre-modern materials.

Text Recognition Model This model takes the identified text areas and converts them into text data. It is based on Microsoft's TrOCR model [28] but with a custom decoder trained on pre-modern Japanese text data. The training data for this model is created through a complex process involving the artificial generation of images, mapping to transcribed text data, and iterative training to improve accuracy.

Reading Order Adjustment Model This model ensures the correct reading order of the extracted text using layout position coordinates.

These three models work together to first identify the text regions within an image, convert those regions into text data, and then determine the correct reading order. The output is text data with positional coordinates, suitable for further analysis.

[5] Japanese woodblock printing is a traditional relief printing technique where entire pages of text and images are carved into a single woodblock, inked, and then pressed onto paper, allowing for mass production of books, art, and other printed materials, without the use of movable type blocks.

Chapter 3

About the Data

3.1 Data Collection

The kuzushiji dataset is "created by the National Institute of Japanese Literature (NIJL), and is curated by the Center for Open Data in the Humanities (CODH). In 2014, NIJL and other institutes begun a national project to digitize about 300,000 old Japanese books"[1]. Among those, 44 woodblock-printed books from the Edo period were transcribed by experts in classical Japanese literature. The transcription includes information on the bounding box of every individual character, as well as the Unicode code point of the closest modern Japanese character. In many cases, it is the exact equivalent but due to various reforms to the writing script across the years, the appearance of characters evolved.

The discrepancy between the ground truth and the annotation is not due to a technical limitation. The Unicode standard includes code points for archaic variations, like 氣 that was simplified to 気 or 國 that turned into 国. However, due to inconsistencies in the usage of the traditional form and simplified form, *nearly* every label was converted to the new modern form, even if the annotated image clearly shows a depiction of the old traditional form. Nonetheless, some inconsistencies still remain in the data as is the case for depictions of the 縣 character sometimes getting labeled as 縣 (old form) or 県 (new form) despite the source images being essentially identical.

Additionally, it is important to note that no sequence or ordering information is saved. The order in which the labels appear in the data files brings no information since it is irrelevant of reading order or geometric position within the image. The layout of old Japanese documents can get quite complex so it is challenging to sequence or otherwise group characters together in meaningful blocks.

3.2 Statistics

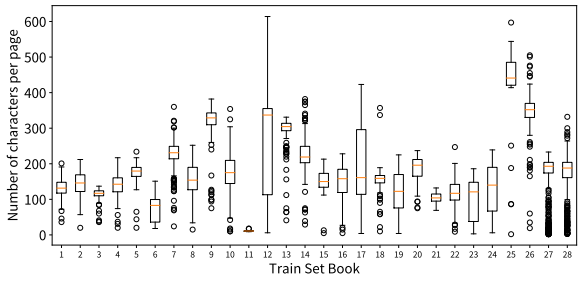

Figure 3.1: Character density per page for books of the training set

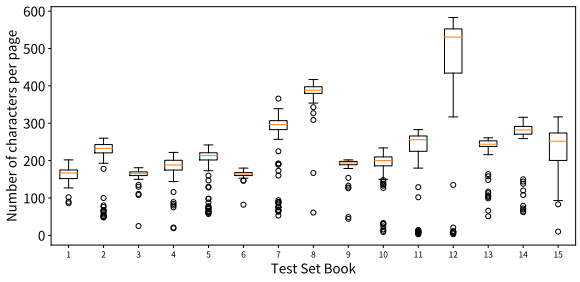

Figure 3.2: Character density per page for books of the test set

The Kaggle competition's training set consists of 3605 JPEG images with an average size of 2096×3254 pixels (std. ±362×509), containing a total of 683,464 annotated characters. The public test set comprises 1730 JPEG images with an average size of 2204×3400 pixels (std. ±418×626), totaling 390,897 characters.

Figures 3.1 and 3.2 illustrate the character density distribution per book. The training set includes diverse books from various genres, resulting in high variability in character density among books. The test set exhibits less variation in character density compared to the training set. In any case, we can expect to see hundreds of characters per page. Notably, the presence of numerous outliers across books suggests that character density can vary significantly within individual books. The densest image, found in the 12th book of the training set, contains 614 characters.

3.3 Challenges in Kuzushiji Recognition

Several properties inherent to books and manuscripts written in kuzushiji present challenges to the automatic processing and recognition of old Japanese text.

3.3.1 Large Number of Character Types

Japanese has one of the most complex writing system in the world with four writing scripts in common use:

1. Kanji: Chinese characters representing morphemes or words; used for nouns, verbs, adjectives, and content words. Example: 日本語
2. Hiragana: Curvy syllabic characters representing phonetic sounds; used for native Japanese words, verb endings, and grammatical elements. Example: にほんご
3. Katakana: Angular syllabic characters; used for foreign loanwords, technical terms, and emphasis. Example: ニホンゴ
4. Rōmaji: Latin alphabet letters; used for beginners, typing, transliterations, company names, and acronyms. Example: nihongo

Alphabetic and syllabic writing systems contain a limited amount of glyphs because there is a hard limit on the amount of individually distinguishable sounds that a human mouth can produce. While logographic writing systems do not have a direct correlation between characters and sounds. Instead, they represent entire words, morphemes,[1] or concepts with individual glyphs, allowing for a potentially vast number of characters.

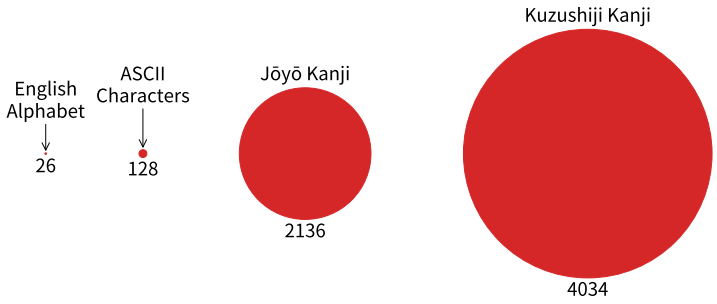

Figure 3.3: Comparison of the number of characters in the English alphabet, in ASCII, and in the jōyō kanji list to the number of kanji in the kuzushiji dataset

In the case of Japanese kanji, there are multiple ways one can count them. We will retain that the list of "jōyō kanji" (literally "commonly used kanji"), taught in compulsory education, contains 2,136 kanji. For comparison, the input domain covered by the train set labels encompasses 4,034 kanji (including variations). Add to this, all the syllabic characters, the punctuation symbols and various typographic symbols and the

[1] A morpheme is defined as the smallest linguistic unit within a word that can carry a meaning, such as "un-", "break", and "-able" in the word "unbreakable" (The American Heritage® Dictionary of the English Language, 5th Edition)

total count reaches about 4,208 possible characters in the training set. Interestingly, there are 574 other characters that never appear in the annotated data but *may* appear in the test set. If the dataset were to encompass older periods of Japanese history, this count would increase even more.

3.3.2 Long Tail Distribution

Not only is the number of character types very large but their frequency distribution follows a long-tail distribution that makes it very challenging for machine learning models to effectively learn and generalize from the data. In a long-tail distribution, a few characters occur frequently, while many others occur infrequently, resulting in imbalanced training data. This imbalance can lead to difficulties in accurately recognizing and classifying less common characters, which are essential for achieving high performance in kuzushiji recognition tasks. Furthermore, when used as a tool to assist a human reading an old text, the most valuable contribution a kuzushiji recognizer can make is help the reader decipher rare kanji. A human reader would quickly learn to recognize very frequent characters while rarer characters are a lot more challenging.

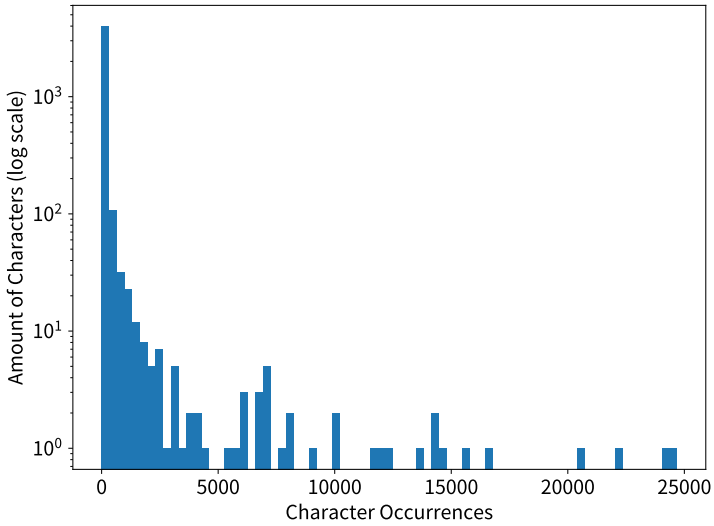

Figure 3.4: Histogram of character frequencies on a logarithmic scale

Figure 3.4 illustrates the long-tail distribution of the character types in the training data set. On the left side, we can see that an extremely large amount of characters (y-scale) occur very rarely in the text. Moving rightwards alongside the axis of character occurrences, we see that the amount of characters that have a high occurrence count sharply falls, until we are left with only a handful of characters occurring over 15,000 times in the corpus.

A more detailed overview of the most frequent characters is given in Figure 3.5. In a dataset where 18.7% of the character types occur only once and 51.9% of the character types occur strictly less than 10 times, notice how the two most common character types both occur almost 25,000 times. The top 4 most common characters make up, together, 13.6% of the entire corpus. Moreover, all the top frequent characters are hiragana.

3.3.3 Similarity between Characters

There is a considerable amount of characters that look similar and would be very hard to tell apart without additional context. Examples can be found even in contemporary Japanese as in 一 (ichi; meaning one), and ー (symbol to lengthen the sound of the preceding vowel) or 口 (kuchi; meaning mouth) and ロ (ro; no meaning) or also 卜 (to; no meaning) and 卜 (uranai; meaning fortune telling). Aside from those visually identical pairs, there are also many more examples of characters with subtle differences such as 天 (yō;

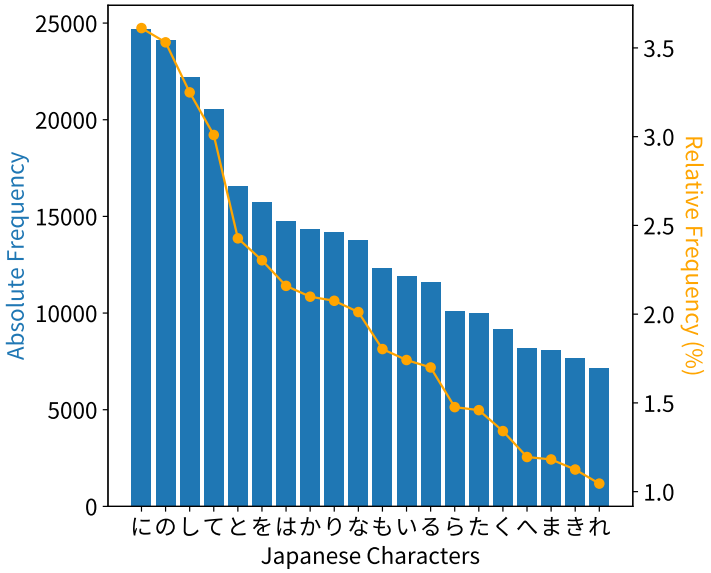

Figure 3.5: Top 20 Japanese character frequencies

meaning early death, calamity) and 天 (ten; meaning sky, heavens) or 宇 (uchi; meaning space) and 字 (ji; meaning character). What's more, the handwritten cursive nature of kuzushiji amplifies this problem and spreads the issue to new sets of characters that had been rendered more easily distinguishable with the move towards modern standardized print. A notable example of this phenomenon is the triplet く (ku), て (te), 〳 (kana vertical repetition mark), highlighted in red in Figure 3.6.

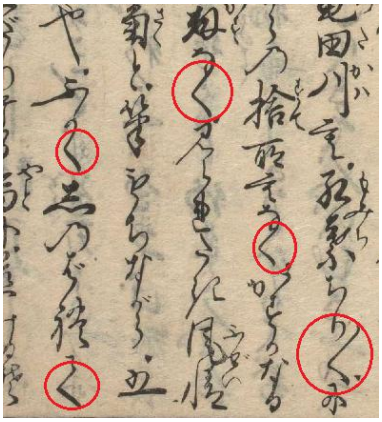

Figure 3.6: The red circles show 3 types of characters : く, て, and 〳

This challenge is particularly problematic for approaches based on the recognition of isolated, cropped individual characters.

3.3.4 Connected and Overlapping Characters

The recognition of connected cursive text has always been more challenging than the recognition of disconnected block letter text, regardless of the language. Modeling the almost infinitely many variations of human handwriting is much harder than modeling constrained, consistent typefaces with clear separation and spacing between characters.

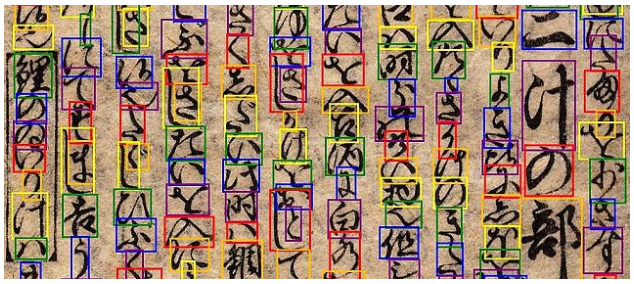

Figure 3.7: Overlapping bounding boxes

Not only are the characters connected between each other but creative placement of characters of varying sizes cause considerable overlap between the bounding boxes of nearby characters, as illustrated in Figure 3.7. As a consequence, it is often impossible to crop a certain character without also including bits and pieces from neighboring characters; this is extra noise that a character-wise recognizer would need to learn to ignore.

Degree of connectedness and overlap also significantly varies from one page to another, or even within the same page; Consider Figure 3.8 to appreciate the contrast between two vastly different degrees of connectedness between the characters.

3.3.5 Various Layouts

As discussed in the previous section, creative placement of characters gives birth to overlapping characters, but the phenomenon can be generalized to layouts as a whole. Although text is normally arranged into columns, read vertically from top to bottom, right to left, this is not always the case.

Take for example a page from a cooking book written during the Edo period, as shown in Figure 3.9. The page is broken down into an irregular grid of rectangle blocks for which the reading order is not necessarily clear. The layout of the text differs from one block to another and, in one case, there is even text wrapping around an illustration of a squid.

Figure 3.8: Examples of connected (top) and disconnected (bottom) handwriting styles in Japanese calligraphy.

3.3.6 Illustrations Mixed with Text

Aside from recipe books, it is also possible to come across lavishly illustrated books like "*The Colors of Spring: The Plum Calendar*" [29], where the text is engraved onto a complex scene. An example taken from the book is given in Figure 3.10.

Pay attention to the angled text inscribed in the fan depicted in the top left corner of the page. A naïve sequencing algorithm might assume that the X and Y coordinates (assuming that the origin 0,0 is at the top left corner of the image) of the next character in a column of text have to be both lower in coordinates compared to the coordinates of the previous character. However, as we can see, it is entirely possible that this rule is respected at the beginning of a "paragraph" and then broken at the end of the paragraph. This example also highlights the importance of rotational invariance for the model. That is to say, that the model should be able to detect and recognize a character even if it is rotated around its center.

Figure 3.9: A page from a recipe book with complex irregular grid layout

Figure 3.10: Illustration of a woman holding a shamisen, with surrounding text

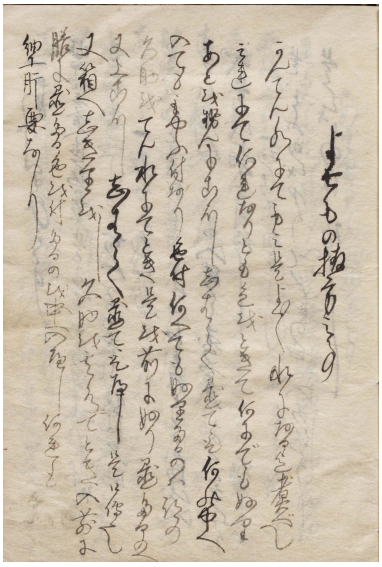

Figure 3.11: Variations in ink application and fading of some characters

3.3.7 Faded Characters

Whether it be through degradation over time or simply through the use of a light brush stroke, some characters look faded and almost erased and thus harder to read. Figure 3.11 is one of the most pathological cases. Furthermore, there are many cases where the characters from the reverse side bleed through the paper and become faintly visible, as illustrated in Figure 3.13. Discerning between characters on the correct side but erased through time and characters coming from an irrelevant page can be challenging.

3.3.8 Scarcity of Pre-Modern Japanese Text Data

Utilizing a language model to refine the predictions of a text spotter model is a common framework in Optical Character Recognititon (OCR) but it presupposes the existence of a digitized corpus of text in the target language that one can use to train said language model. All instances of pre-modern Japanese are stored in analog media and only a tiny portion has been transcribed digitally. Furthermore, during the transition from classical Chinese to modern Japanese, some genres like political essays and historical books stayed closer to Chinese grammar and word usage while artistic works of fiction embraced more openly the change and wrote using totally different norms and character choices. In these transitory times, it was also possible to come across books that mixed three very different writing conventions (Figure 3.12).

Few-shot out-of-domain transfer learning leveraging contemporary Japanese corpora might be a promising lead to mitigate the lack of training data but this is outside the scope of this Master's thesis.

3.3.9 Extreme Simplification of Certain Characters

By virtue of the handwritten nature of kuzushi text, we are bound to come across sometimes extreme simplifications of certain characters for the sake of handwriting speed. Unrecognizable levels of simplification are a common occurrence in old Japanese documents. The most pathological case is that of the 御 kanji, typically used as a honorific prefix. Page 319 of the "Dictionary of Kuzushiji Examples"[31] (Figure 3.14) shows a step-by-step decomposition of the kanji in question from its original, standard 12-strokes form into a mere single stroke, more akin to a comma than a fusion of 彳 and 卸.

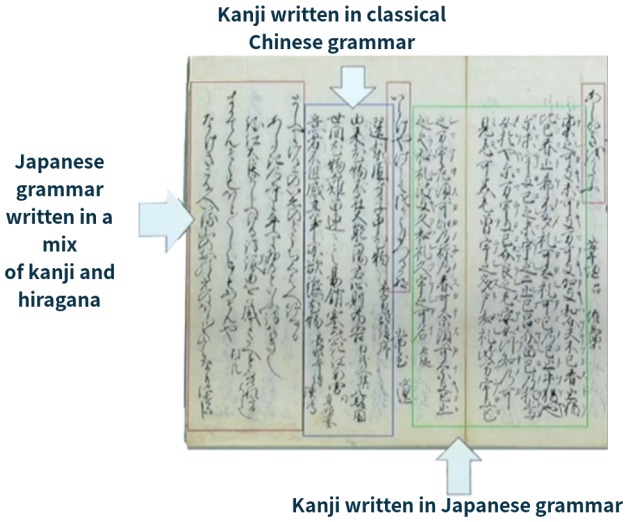

Figure 3.12: Showcase of three different writing conventions within the same book. (Image adapted from a presentation given by Tarin Clanuwat [30])

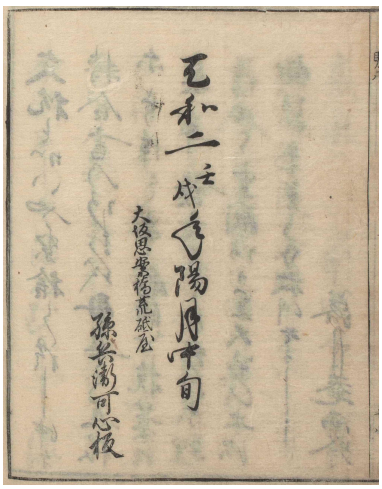

Figure 3.13: Characters from the reverse side bleed through the paper and become faintly visible

Figure 3.14: Progressive simplification of the 御 kanji from right to left, in eight steps. Adapted from [31, p. 319]

3.3.10 Furigana

Small text written to the side of kanji is called “furigana” and it serves as a reading aid to help the reader determine the correct pronunciation for a given word written in kanji. Those reading aids are out-of-scope for the task and must be ignored.

3.4 Summary

Kuzushiji recognition presents a complex challenge due to the unique characteristics of the writing system and the historical context of the texts. The sheer number of character types, many of which are rare and visually similar, coupled with the cursive and interconnected nature of the script, make accurate recognition difficult. The variability in layouts, the presence of illustrations, faded characters, and the scarcity of digitized pre-modern Japanese data further exacerbate these challenges.

Chapter 4

Model Architecture

To address these challenges, we propose a model architecture that incorporates seven different models in total. The process is split up in three big phases: text detection, coarse-grained categorization and fine-grained character recognition by Mixture-of-Experts.

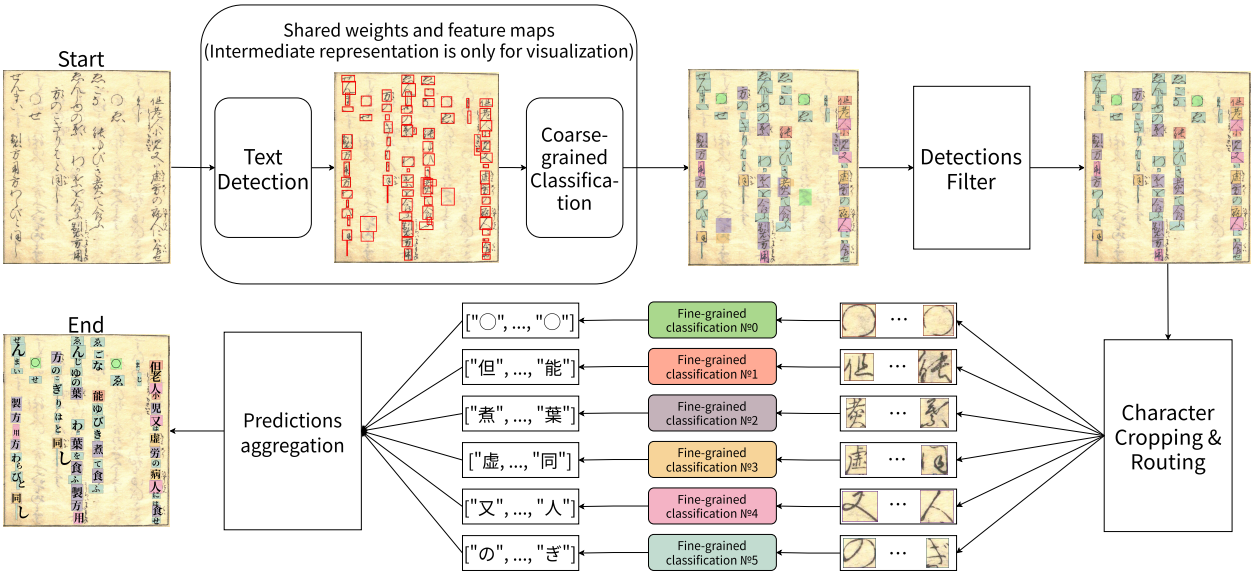

Figure 4.1: Schematic overview of the inference pipeline

The initial model processes an input image to identify and enclose individual characters within bounding boxes, concurrently classifying each character into one of six predefined categories. To enhance accuracy, detections with low confidence or substantial overlap with others are discarded.

Subsequently, the model's output is utilized to crop individual characters and direct them to the most suitable specialized character recognition model, chosen based on the broad character class identified in the crop.

Finally, predictions from all specialized models are aggregated, and the previously dispersed crop prediction data is grouped by the corresponding source image. After mapping all input images to their predicted bounding boxes and characters, a final post-processing step (absent from the diagram) is applied to transform the bounding box coordinates from `x,y,h,w` (top-left corner coordinates, height, and width) to a single center point, ensuring compliance with the Kaggle competition submission format.

With the high-level overview out of the way, let us now proceed to an explanation of the reasoning behind those six coarse classes before decomposing and analyzing each model in more technical details.

4.1 Coarse-Grained Categorization Systems

To divide the problem, we came up with the idea of training different "expert" models, specializing in the recognition of characters belonging to one kind of category. A similar idea had been advanced for future work by Tarin Clanuwat, et al. in [12], saying that "if we could somehow group or identify related kanji characters, then we could share more parameters between them and perhaps generalize much better."

There exists many ways to group and classify Japanese characters. For our case, we needed a classification system based on graphical features of the character since that is all the model is fed with. Fortunately, many such systems have been devised; some old, and some new. In this section, we shall briefly present three of the most popular kanji classification systems, compare them and evaluate their suitability for our task.

4.1.1 Kangxi Radical System

Almost all graphically organized monolingual Japanese dictionaries use "radicals" to group and index Japanese kanji. Although complex-looking and vast in numbers, all kanji are compositions of a restricted set of the same graphical components. The radical of a kanji is the graphical component that is used to index the kanji. It is typically the semantic indicator (as opposed to the phonetic indicator) and is most often on the left side of the kanji but not always.

Figure 4.2: Examples of radicals and their usage in kanji

Various dictionaries came up with various sets of radicals but we will retain our attention on the *Kangxi* radical system whose radical set was first introduced in Mei Yingzuo's "Zihui", a dictionary published in 1615, but only gained widespread traction in the more famous Kangxi Dictionary of 1716.

This system contains 214 radicals, where many of those radicals actually have multiple forms depending on where they are positioned in the kanji. For example, the radical 人, as in 仄か, can also show up as 亻, as in 仏. It can also morph into 𠆢, as seen in 今. Not only are the radicals large in number, but they are also not straightforward to identify. Certain kanji are composed exclusively of graphical components present in the radical set but, by definition, a kanji can only have one radical. A model would need to learn that the kanji 杉, although composed of 木 and 彡, is actually indexed by 木. While the similar looking 彩 which is also composed of 木 on the left and 彡 on the right is not indexed by 木; its radical is 彡.

Furthermore, there are multiple instances in the training data where the image shows the old form of a kanji while the label annotation uses the new form. This can pose a big problem when the radical of a kanji changes after the kanji simplification reform, as illustrated in table 4.1.

Old Form	Old Radical	New Form	New Radical
兩	入	両	一
會	日	会	人

Table 4.1: Examples of kanji undergoing a radical change alongside a graphical change. Retrieved from [32]

In conclusion, radical systems like Kangxi are unsuitable for our task because training hundreds of different expert models would be prohibitive and detecting which exact graphical component of a kanji is *the* radical can be challenging.

4.1.2 Stroke Count

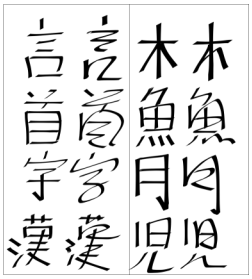

Figure 4.3: Handwritten Japanese characters showing block-style characters on the left and light cursive on the right. Retrieved from https://www.sljfaq.org/afaq/shotai-handwritten.html

A second system of classification, often used in combination with the prior radical system to sort entries belonging to the same radical, is based on counting the number of strokes in a kanji. Simple looking kanji like 女 have a low stroke count while complex-looking kanji like 鬱 are written with many strokes.

This is unsuitable for kuzushiji because the amount of strokes between block characters and cursive characters is not the same. Characters written in regular block style have a fixed amount of strokes, regardless of the writer, while cursive writing offers infinite variety in what strokes get merged together or simplified.

4.1.3 System of Kanji Indexing by Patterns (SKIP)

Finally, let us look at the System of Kanji Indexing by Patterns, abbreviated SKIP, developed by Jack Halpern for *The Kodansha Kanji Learner's Dictionary*, aimed at English learners of Japanese.

As the name implies, kanji are organized based on patterns. There are four major patterns describing the geometrical shape of the overall kanji:

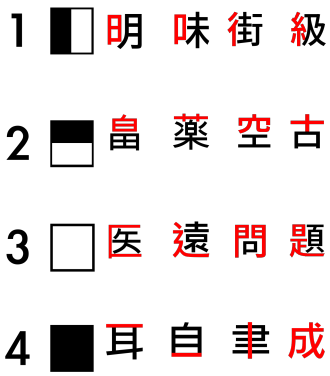

Figure 4.4: Color-coded representative examples for each pattern.

1. Left-right
2. Up-down
3. Enclosure
4. Solid/standalone

The exact graphical components used are irrelevant. What matters is the formation, the pattern in which they are put together. The first three patterns are the most straightforward. The fourth pattern technically contains subtypes (one subtype for top line, another for bottom line, a third for through line and all the rest,

as illustrated in order in Figure 4.4) but we will not pay attention to those subtypes and classify all kanji that are not in any of the first 3 categories into the fallback category №4.

Due to the tendency of connecting strokes belonging to different components in cursive, there might not be a clear space between the left and right parts of kanji of type 1 or the up and down parts of a kanji of type 2 but the overall horizontally or vertically stacked nature of the kanji is still perceivable.

Most importantly, this option lets us train only a handful of specialized kanji recognizer models. To make sure that this is a good idea, it is helpful to take a look at the partitioning statistics that this kind of classification gives for kanji in our training dataset.

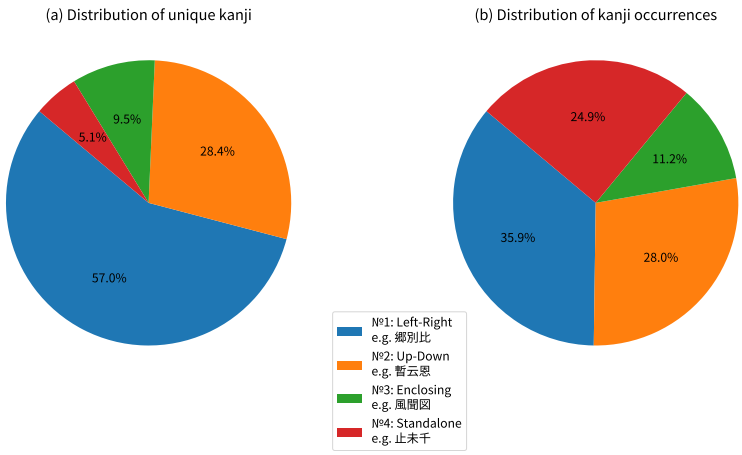

Figure 4.5: Distribution of the *kanji* in the training set among the 4 SKIP patterns

A majority of the unique kanji types fall under the vertical-split pattern, with members of the horizontal-split pattern also representing a considerable percentage of the unique kanji types. The remaining two patterns account for less than 10% each. However, linking this information with the partitioning statistics of kanji occurrences offers compelling insights. For instance, correctly classifying 24.9% of the observed kanji under the fourth pattern reduces the search space to only 5.1% of unique character types. In comparison, the reduction in uncertainty resulting from the knowledge that a kanji adheres to the left-right pattern may not appear significant but this is still a 43% reduction of the search space.

Considering these factors, we adopted the System of Kanji Indexing by Patterns (SKIP), with a few extensions, as our coarse classification system. In its current form, SKIP only categorizes kanji while our datasets also feature hiragana, katakana, and various typographic and punctuation symbols. To include these characters, two new pseudo-patterns were added to the SKIP: pattern 0 for typographic symbols, and pattern 5 for kana (i.e. hiragana and katakana combined).

Additionally, while the full SKIP data mappings are available under a Creative Commons license, the latest version lacks some rarer historical kanji found in Edo-period texts. These were manually classified into the most suitable SKIP pattern.

4.2 Text Detector

The first model in the pipeline is the text detector, which is also in charge of predicting the SKIP category of the detected characters. In that sense, the initial kuzushi text detection and recognition problem is modeled as an object detection task with six classes of objects, according to the extended SKIP explained earlier. Although, the task is fundamentally a handwritten OCR problem, the training labels and expected output are more in line with object detection tasks. OCR models made to recognize handwriting typically expect the image of a full line of text in input and output a string with no positional information. While object detection models are trained on pairs of bounding boxes and object classes and output the coordinates of the box bounding a detected object as well as what kind of object it is.

Consequently, we set out to fine-tune an object detection model that proved to be the state-of-the-art in the vibrant field of deep learning object detection, on the kuzushiji dataset. At the time of writing, the best performing model architecture on the MS COCO (Microsoft Common Objects in Context) [21] and LVIS (Large Vocabulary Instance Segmentation) [33] benchmark datasets is "DETRs with Collaborative Hybrid Assignments Training" [34], also called "Co-DETR".

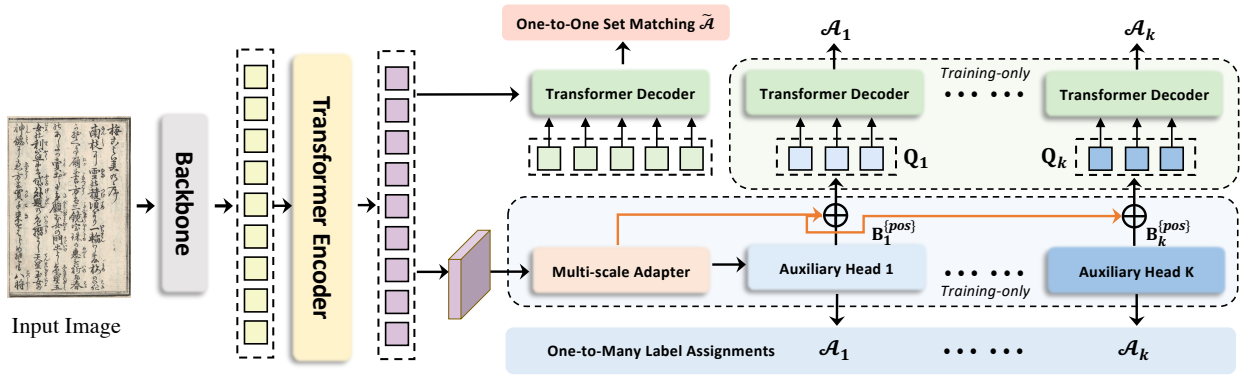

Figure 4.6: Diagram of the Co-DETR architecture. Adapted from [34]

Modern deep-learning object detection models are typically composed of a backbone, a neck and one or more prediction heads. This is how Co-DETR is also composed.

In the Co-DETR variant used for this thesis, the backbone is the large version of a Swin transformer [35], the neck is a channel mapper used to merge the channels of backbone features with convolution and there are three separate heads: a DETR head present during training and inference and two auxiliary heads only present during training that also predict bounding boxes and object classes.

4.2.1 Preface on Transformers and Attention Mechanisms

Before delving into the object detection architecture piece-by-piece, we first establish the foundation by introducing transformers and attention mechanisms, concepts pioneered by Vaswani et al. (2017) in their seminal work "Attention is All You Need" [36].

Convolutional Neural Networks (CNNs) have long been the dominant tool in computer vision tasks like image classification and object detection, as is evidenced by their popularity in previous solutions (see Section 2.3). While CNNs excel at detecting local patterns through their convolutional layers, the responsibility of integrating these local features into a global understanding often falls on the fully connected layers (the head of the network). They receive the extracted local features and learn to combine them to make a prediction about the entire image. While they can learn some relationships between features, this process is often limited due to the flattening of the spatial information.

This limitation becomes particularly problematic in scenarios where long-range dependencies are crucial for comprehension. In the recognition of kuzushiji, for example, characters are often ambiguous and require contextual information from surrounding characters or words for disambiguation. Furthermore, old documents may suffer from degradations that necessitate utilizing information from other parts of the document to reconstruct damaged characters. In both cases, the ability to capture long-range dependencies is vital for accurate and robust recognition.

To illustrate, consider the car part decomposition example from Section 2.3.2. While CNNs effectively identify individual components (wheels, headlights, etc.), their understanding of spatial relationships between these components (e.g., the wheel is below the door, the headlight is next to the grille) might be less precise. Transformers, on the other hand, can directly model these relationships thanks to their attention mechanism, resulting in a more integrated understanding of the object's structure. This mechanism allows transformers to weigh the importance of each element relative to all others, thus capturing both local and global context.

The intuition being to focus the attention on the most relevant areas for deciphering the character. This means paying less attention to irrelevant parts of the image such as the texture or tears of the page, and extraneous strokes and scribbles.

In the context of computer vision, transformers operate as follows:

Patching The image is divided into smaller patches, analogous to words in a sentence. This is where the Swin transformer differentiates itself from other transformers developed for computer vision, as we will see later.

Embedding Each patch is converted into a vector residing in multi-dimensional space representing its features and spatial position.

Attention Mechanism This mechanism enables the model to assess the relevance of each patch in relation to all other patches, determining which parts of the image are most crucial for understanding the overall scene.

Learning and Prediction Through multiple layers of attention and processing, the transformer learns to capture complex relationships within the image, leading to improved predictions for various tasks.

Each patch is associated with three vectors: a Query (what the patch is looking for), a Key (what the patch represents), and a Value (the information the patch carries).

To illustrate how the Query, Key, and Value vectors might work for kuzushiji recognition, imagine that a transformer is trying to recognize a specific character in a kuzushi text, let's say the character for "mountain" (山).

The Query vector for the patch representing 山 might encode questions like: "Am I a single stroke or multiple strokes?", "What is my overall shape (triangular, square, etc.)?", "Are there any prominent hooks or curves?".

The Key vectors for all other patches in the text (including the patches within the "mountain" character itself) would encode information like "I am a horizontal stroke at the top.", or "I am a diagonal stroke pointing downwards.", or "I am a small hook at the end of a stroke."

The Value vectors for all patches would contain detailed information about their visual features, such as the angle of the stroke, the thickness of the stroke, the position relative to other strokes, etc.

The distinction between Key and Value vectors can be a bit subtle in the attention mechanism, but the idea is that the Query is used to compare against the Keys to find relevant patches. Once a good match is found between a Query and a Key, the transformer reads the actual content of the embedded patch (the Value) to update the internal representation of the "mountain" character. To give an analogy, it is like searching for a book in a library on a specific topic (the Query). Instead of opening and reading every book in the library to find an answer to the query, only the book title and summaries (the Key) are checked to evaluate its relevance to the query. Once a relevant book is identified, we can open the book and read the content to gain a better understanding of the topic we had in mind (updating the Value vector).

In practice, as is often the case in machine learning, everything is expressed in terms of numeric vectors and matrices so the similarity between a Query and a Key is computed by a dot-product of the two vectors. The similarity scores thus obtained are then normalized (usually with a softmax function) to create attention weights. Since these attention weights are computed between each pair of patches in the image, we talk about "self-attention" and the full matrix of attention weights, the attention matrix, has as many rows and columns as there are patches.

With this in mind, we can now turn our attention to the modules of Co-DETR.

4.2.2 Backbone

In the first stage, input images are converted into feature maps via the backbone. As the name implies, the Swin Transformer uses a transformer with an attention mechanism under the hood. Since transformers only work on sequences of tokens, the 2D image must be flattened into a sequence before it can be fed into the transformer.

The first idea one might have to flatten an image would be to create a sequence where each token corresponds to a single pixel of the image but even a rather low resolution image of 256×256 would produce a sequence of 65,536 tokens. This is already quite a lot to hold in memory but it gets worse when you remember that the shape of the self-attention matrix is the square of the number of tokens, which would amount to 4,294,967,296 matrix cells in our small example. Furthermore, that would be akin to segmenting text into letter tokens, which has proven to be unoptimal in NLP. If the tokens are too small and do not encode sufficient meaning, a lot of layers are needed before meaningful units of information can emerge.

Consequently, models like the Visual Transformer (ViT) [37] divide the image into 16×16 patches. This is perfectly viable for image classification tasks but in dense vision tasks like object detection, it might cause poor performance on the detection of small objects (of which we have many in the context of text on book pages) due to the low-resolution feature maps.

To overcome these issues, Z. Liu et al. proposed the general-purpose Swin Transformer backbone. To quote the original paper, "Swin Transformer constructs a hierarchical representation by starting from small-sized patches and gradually merging neighboring patches in deeper Transformer layers."

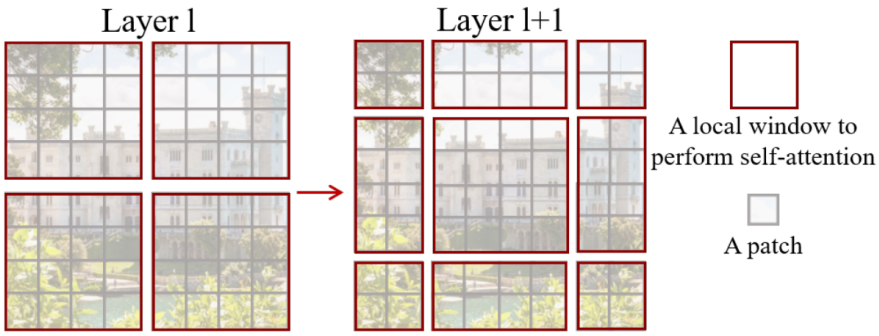

Figure 4.7: "An illustration of the shifted window approach for computing self-attention in the [...] Swin Transformer architecture." [35]

The embeddings of the image tokens/patches are updated through a self-attention mechanism limited to patches in a local window, as illustrated in Figure 4.7, instead of applying self-attention to all the patches of the image. However, this is not enough to model longer-range dependencies. For example, imagine that the features associated to the 立 upper part of 音 are in one local window but the features associated to the 日 bottom graphical component of the same kanji are in the domain of another local window. This **S**hifted **Win**dow partitioning approach, designed to ensure a connection across windows, lends the method its name. Transformer blocks with shifted window partitionings are put in succession and that way, the self-attention computation in the new windows crosses the boundaries of the windows of the previous partitioning scheme, thus providing connections among them.

4.2.3 Neck

The backbone produces multiple feature maps, each focusing on different aspects of the input. The neck gathers these maps and combines them, like collecting information from different sources to get a complete picture. Since the feature maps can be quite large and complex, the neck uses a 1x1 convolution that condenses the information, making it easier to process. By reducing the dimensionality and combining features, the neck creates a more compact and informative representation that is better suited for the head to make accurate predictions.

In the context of the particular architecture used for the thesis, we followed the official implementation of Co-DETR by ensuring that the convolution is followed by a group normalization with 32 groups and that the number of out channels is limited to 256. Group normalization is often preferred over other normalization methods (like batch normalization) in models like DETR and Co-DETR because these models work with small batch sizes — as a matter of fact, the model is so big that we had to reduce the batch size to only 2

images processed at once. Group normalization is less sensitive to small batch sizes and can lead to better performance in such cases.

4.2.4 Heads

The heads form the most complex part of the Co-DETR model. There is one main head and K parallel auxiliary heads present only during training. Features from previous layers go through a transformer encoder and the outputs of the encoder are then passed to the various heads.

The main head is a DETR head with a one-to-one label assignment scheme, while the auxiliary heads employ a one-to-many label assignment scheme. This creates a situation where multiple heads "collaborate" together to infuse more meaningful information into the output features of the transformer encoder, while using two types of assignment schemes during training, hence the name "DETRs with Collaborative Hybrid Assignments Training".

This raises the question: what are one-to-one and one-to-many label assignment schemes? The answer is that this denotes how the ground truth labels get matched with the predictions of the head. To cite the Co-DETR authors, "for one-to-many label assignment in object detection, multiple box candidates can be assigned to the same ground truth box as positive samples in the training phase." Conversely, one-to-one set matching is an idea first brought up in the original paper introducing DETR [38]. In this scheme, each of the N predicted objects is paired with the most similar ground truth object. Since the number of objects may differ between images, a special "Background/No Object/Ø" class is added to ensure a bipartite matching where all predictions have a corresponding label. This creates a consistent N-length label vector for each image.

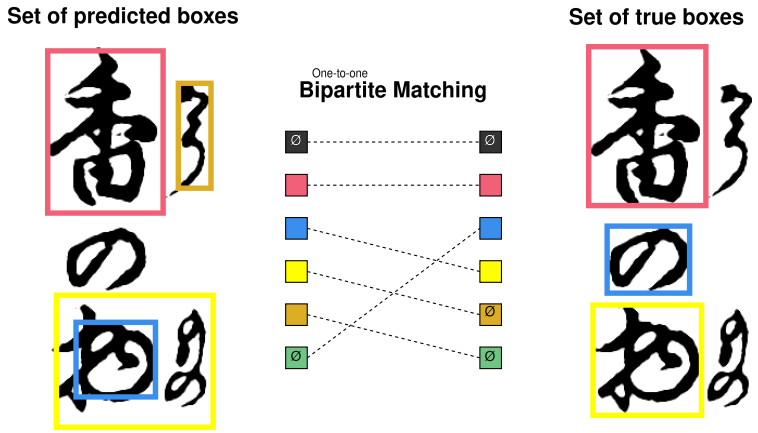

Figure 4.8: During training, bipartite matching uniquely assigns predictions with ground truth boxes. Dissimilar associations (e.g. a box matched to background or two matched boxes centered on the same point but with different sizes) are penalized.

In the paper, Zong et al. found that using two auxiliary heads ($K = 2$) maximizes accuracy. Additional heads lead to inconsistent feature learning and decreased performance due to mixed signals.

The model also uses positive bounding box coordinates from auxiliary prediction heads to create additional sets of positive queries for the transformer decoder. These queries are specifically designed to train the auxiliary decoders on positive samples. Importantly, the auxiliary decoders share parameters with the layers of the main DETR decoder, meaning that learning in the auxiliary branches directly improves the main decoder.

By generating and processing these extra positive queries, the auxiliary branches enhance the diversity, robustness, stability, and consistency of the training signals. This is particularly beneficial for addressing the instability inherent in the one-to-one bipartite matching used in the main head.

The bipartite matching dynamically assigns each ground truth box to a specific query during training. Since these assignments can shift across iterations, the same query might be linked to different ground truth boxes over time, leading to instability in the learning process. The auxiliary branches mitigate this issue by providing a more stable source of positive query-ground truth pairs through their one-to-many label assignments.

Thanks to this transformer architecture, we can make use of self-attention to capture close- and long-range dependencies as well as focus on the features most important for certain classifications while ignoring the noise.

At the final stage of the architecture, there are some simple linear layers taking in the multi-dimensional embeddings of the decoder to give a final prediction on the bounding box of all the objects found in the image as well as a probability distribution over possible classes for this object.

4.2.5 Predictions Filter

After obtaining bounding box and SKIP classification predictions alongside their confidence scores, the predictions go through a post-processing phase. Post-processing of the Co-DETR model predictions involves the removal of low-quality predictions. A confidence threshold of 30% was determined experimentally to optimally balance false positives and false negatives (Figure 4.9).

(a) Confidence > 0% (b) Confidence > 30% (c) Confidence > 60% (d) Confidence > 90%

Figure 4.9: Comparison of confidence thresholds

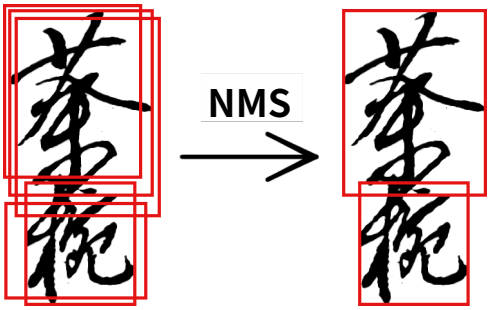

Figure 4.10: Applying Non-Max Suppression (NMS) on an example

Additionally, non-maximum suppression (NMS) was employed to address redundant bounding boxes, a phenomenon observed when the model was uncertain about SKIP assignments or segmentation choices (Figure 4.10). NMS prioritizes the highest-confidence bounding box and eliminates those with lower confidence and significant overlap, as quantified by the Intersection-over-Union (IoU) metric (Figure 4.11). Due to the prevalence of true overlapping boxes in kuzushiji, a moderately high IoU threshold of 55% was selected for our post-processing pipeline.

4.3 Specialized Character Classifiers

After text detection and SKIP classification, the next step is to accurately recognize the specific characters in the image. To utilize the information from the SKIP classification, a Mixture-of-Experts (MoE) approach was implemented.

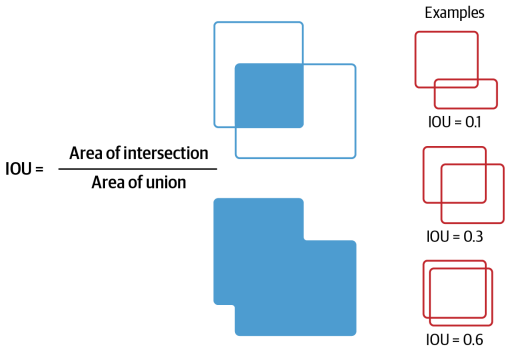

Figure 4.11: Explanation of the Intersection-over-Union (IoU) metric. Retrieved from [15, p. 137]

4.3.1 Mixture-of-Experts

In our simplified MoE variant, multiple deep learning models are trained independently, each specializing in a particular subdomain of the problem. While in some MoE variants, the actual subdomain that each expert will specialize in is a learnable parameter, in this case, the extended SKIP classification system serves as an inductive prior, guiding the selection of the appropriate expert model for each cropped image based on its detected SKIP pattern. This conditional computation property allows for faster inference as only the relevant parts of the network are activated for each example. This stands in contrast to ensemble methods involving routing the same image data to multiple different experts and pooling the results together.

Each of the six character recognizers shares the same architecture but differs in their training data and output domain. This specialization allows each model to focus on recognizing characters within its specific subdomain, leading to improved accuracy and efficiency.

4.3.2 SVTR

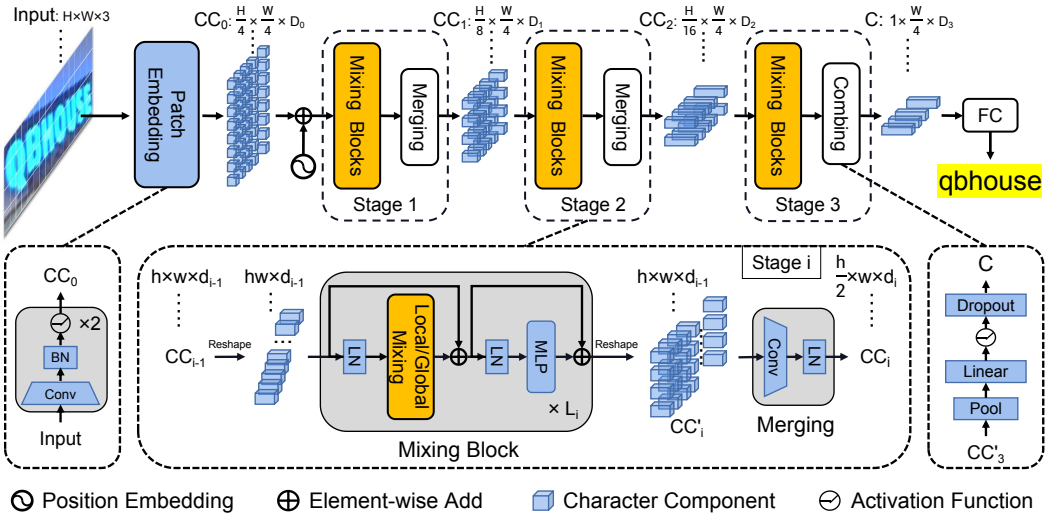

Figure 4.12: SVTR diagram. Retrieved from [39]

The architecture in use is a fine-tuned Single Visual model for scene Text Recognition, abbreviated SVTR [39]. As the name implies, SVTR is a deep learning model designed for the task of scene text recognition, a sub-branch of OCR. Its main appeal is its single visual model that does away with the hybrid architecture, typically seen in OCR contexts. Hybrid OCR architectures require a visual model for feature extraction and a sequence model for text transcription. However, kuzushiji reading order is unavailable and digitized

corpora of pre-contemporary Japanese are not sufficient in number to train a well-performing language model. Hence the interest in a single visual model that remains nonetheless specialized in the recognition of text.

SVTR operates by first dividing the input image into small patches. These patches are then embedded into a higher-dimensional space using a series of convolutions. This process is similar to how Vision Transformers (ViT) [37] break down images into patches.

The embedded patches are then passed through a series of mixing blocks. Each mixing block consists of two types of mixers: local and global. The mixing blocks essentially combine and transform information from different parts of the image using self-attention to create a richer and more informative representation of the underlying text in the image. The difference between the two types of mixers resides in the scope of the self-attention. Local mixing blocks focus on self-attention within a small, fixed-size window of patches, similar to a convolutional operation. Global mixing blocks, on the other hand, perform self-attention across all patches in the image, capturing long-range dependencies.

Unlike the previously explained Swin Transformer, SVTR mixing blocks do not use shifting windows. They always operate on the same fixed-size window. Since the model deals with very small image crops containing only one element of interest in our case, the resolution can be lowered and global self-attention becomes more affordable.

Nonetheless, the spatial resolution is not constant throughout the network. It gets reduced after each stage of mixing. In the middle stages, a convolution operation is used to downsample the height of the feature map while keeping the width constant. In the final stage, the height dimension is pooled to 1, and a fully connected layer is used to combine the features.

Finally, a linear layer at the end predicts the character sequence. Although SVTR is capable of predicting long sequences of characters, we limited its sequence length to 1 since we do not have information on the correct reading order of the individually detected characters in the page.

Thanks to this architecture, the authors achieved competitive accuracy on English and Chinese scene text recognition benchmarks.

4.4 Summary of Chosen Model Architecture

In summary, a fine-tuned Co-DETR model locates each character within the input image. A tightly-cropped image of each character is then forwarded to the most appropriate SVTR model, specialized in recognizing characters of the corresponding SKIP class. Then, results are aggregated back together.

It is important to note that the SVTR models are limited to recognizing characters within their trained classes and cannot identify errors made by the Co-DETR model. Consequently, the overall performance of this two-stage pipeline is constrained by the accuracy of its least precise component.

Chapter 5

Training Procedure

5.1 Software

The training procedure was implemented in Python using the OpenMMLab ecosystem of software libraries for deep learning. In particular, MMDetection version 2.25.3 for PyTorch version 1.11.0 on CUDA 11.3 was used for training Co-DETR, and MMOCR version 1.0.1 for PyTorch version 2.0.0 on CUDA 11.8 was used for training SVTR. Unfortunately, they require different incompatible versions of Pytorch and MMCV (a common dependency for OpenMMLab libraries focused on computer vision) which complicates the project setup and installation.

5.2 Hardware

Hardware resources used for training and inference involve a plethora of various GPU providers in the cloud, and a virtual machine granted by the INGI department, albeit without a GPU. Nvidia T4 and Nvidia P100 GPUs freely available on Kaggle under a weekly quota of 30h of compute per week were extensively used. When that was not enough, we rented GPUs on the vast.ai online marketplace. This included a RTX4080S, a RTX4090, a RTX3060Ti, a GTX1070, a Titan V, and most importantly, an A40 GPU. We chose the small variant of SVTR so any GPU with 8 to 16 gigabytes of GPU RAM was sufficient but the best-performing variant of Co-DETR is very large, with the checkpoint weighing 2.7GB on disk. Even after batch size and resolution reduction, training Co-DETR with a Swin-Large backbone takes up 47GB of GPU RAM, hence the importance of a GPU like the A40 with over 48GB of GPU RAM. Inference on a single image only consumes around 8GB of GPU RAM however.

5.3 Train-Val-Test Data Split

We maintained the same train-test split in the dataset as in the Kaggle competition. The train set was further split, with 80% of the samples used for fitting the network's weights and 20% held out for validation after each epoch to facilitate early stopping. While it was possible to hold out pages from one book as the validation set, like some past competitors had done, and obtain better approximations of the model's true performance, we opted for a simple shuffled 80–20 split. This choice aimed to expose the model to as many writing styles as possible.

However, during the initial exploration to assess the viability of a considered approach, a simplified subset of the data was used to expedite the process. To achieve this, samples were taken from a single book with a consistent text layout that is free of illustrations. To relax the problem further, rare characters appearing only once or twice were removed. In addition, the regions of these rare characters were covered in the source images with a solid rectangle filled with the background color. This reduced the total number of classes from 4208 to 1120, and the total number of images from 3605 to 217.

5.4 Loading and Preprocessing Pipeline

In our data loading and preprocessing pipeline, data was initially loaded from annotations in textual format and JPEG files stored locally on the machine.

To enhance the robustness of the models, the training set underwent augmentation, unlike the validation and test sets. Data augmentation is a whole art in and of itself in the domain of computer vision, but we limited ourselves to the basics, without exploring kuzushiji-specific hand-crafted augmentation techniques due to time constraints.

The training data for the SKIP detector underwent only horizontal flipping and random cropping. On the other hand, the SVTR models had to classify a lot more than 6 classes so this is where we focused more on data augmentation to increase the representation of rarer characters. Specifically, we followed the CVPR 2020 paper "*Learn to Augment: Joint Data Augmentation and Network Optimization for Text Recognition*"[40] to randomly apply general geometric augmentations to the image like distortion, stretching, and perspective transforms. On top of that, random cropping, Gaussian blur, additive Gaussian noise, color inversal, and image content and color jittering[1] were also applied here.

Following this, multi-scale resizing was applied to the images. In the case of the SKIP detector, the height was variable while keeping the aspect ratio and the width fixed at 1024 pixels[2]. For SVTR, images were fundamentally much smaller so the crops could be resized down to 256×64 pixels without loss of essential information for character recognition. Coupled with resizing, the pixel values were also normalized.

5.5 Optimizer, Batch Sizes, and Epochs

The training process involved using the AdamW optimizer, a batch size of 2 for Co-DETR and 128 for SVTR, and a total training duration of 15 epochs for Co-DETR and 60 epochs for each SVTR. The performance of the various SVTR models stagnated or even dipped a little on the validation set in high epochs around 40, so we stopped training "early" at 60 epochs because it was unlikely that the model would improve considerably with just further training. Whereas the evaluation performance of the Co-DETR was on an upwards trajectory and would have probably improved more with further training. However, the model is huge and we could not afford spending more time and money to train it for very long.

5.6 Pre-Trained Weights

In all the cases, training started from a set of pre-trained weights. In other words, we fine-tuned existing models on our data instead of training from scratch.

The starting weights for the SKIP detector were taken from a Co-DETR model pre-trained on the Objects 365 dataset [41], a large-scale object detection benchmark with 365 categories, 2 million images and 30 million bounding boxes. The model was later fine-tuned on COCO for 16 epochs. The model features 900 trained object queries in its decoder and uses a Swin-Large backbone which was itself pre-trained on ImageNet22K [42], a large-scale image classification dataset with over 14 million images across 21,841 (≈22K) categories.

As briefly mentioned previously, the base for the character recognizers is the small variant of SVTR. Unfortunately, MMOCR only offers a SVTR-S pre-trained on synthetic English scene-text datasets (MJSynth also known as Synth90K [43], and SynthText [44]). A model trained on the Chinese Scene Dataset [45], as done in the SVTR paper, would have been preferable to transfer learning to Japanese, but it was certainly better than nothing.

That said, this pre-trained SVTR checkpoint was only used once to initialize the weights of the first character expert. Subsequent models loaded the checkpoint of previous trained kuzushiji recognizers. After all, a model

[1] Color jittering: Randomly adjusts an image's colors as expressed by its brightness, contrast, saturation and hue. Image content jittering: Randomly shifts the image's content within its frame.

[2] The original Co-DETR configuration sets the width at a fixed 2048 pixels but this was too big in memory for the GPU we used

fitted to recognize kanji in left-right SKIP pattern №1 is a much better base for a model whose task is to recognize kanji in up-down SKIP pattern №2, and so forth.

5.7 Training Challenges

Overall, managing the technical aspects of running and training the model in the cloud was the most time-consuming task of this work. There are many interconnected components and dependencies in the software toolchain.

Chapter 6

Results Analysis

This chapter presents the results of the two-stage OCR pipeline, detailing the performance of the SKIP detector (Co-DETR) for character detection and the individual SVTR models for character recognition. The evaluation focuses on both quantitative metrics and qualitative observations, offering insights into the strengths and limitations of the chosen approach.

6.1 Text Detection Performance

To assess the effectiveness of our text detector on historical Japanese documents, we employed a comprehensive approach involving quantitative metrics, qualitative analysis of representative samples, and error analysis.

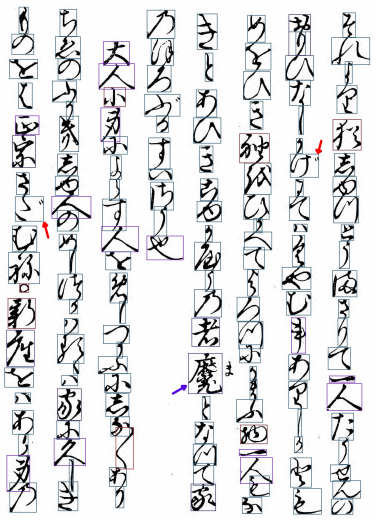

Figure 6.1: Perfect text detection on a well-preserved document with high contrast and connected overlapping characters

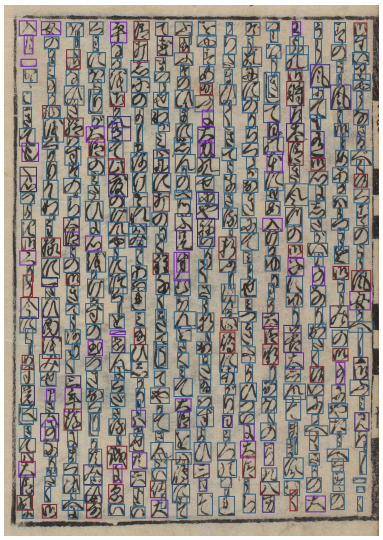

Figure 6.2: Many crowded predictions on a page with very dense text

6.1.1 Quantitative Evaluation

We used the official Kuzushiji Recognition competition metric. To score a true positive, center point coordinates of predicted character regions must be within the ground truth bounding box — the predicted

character itself must also match with the ground truth character but we shall focus our attention on pure text detection only for the time being. As long as the point is *anywhere* within the ground truth box, it is registered as a correct text detection. A predicted point that is closer to the true center of the bounding box does not give a better score than one that is closer to the edges but still within the box.

For that metric, our model demonstrated exceptional text detection capabilities, achieving a high F_1-score of **98.48%**. The precision of 96.65% indicates a low rate of false positive detections (i.e., predicting text where none exists), while the recall of 99.32% highlights the model's ability to capture most ground truth text regions. In text detection tasks, prioritizing recall is often beneficial, as false positives are generally less detrimental than missing true characters. A minor amount of noise in the form of extra character predictions is typically acceptable in exchange for greater sensitivity.

However, the center point-based metric alone does not assess the tightness of predicted bounding boxes. To evaluate this aspect, we utilized the Intersection-over-Union (IoU) metric, defining true positives as predicted bounding boxes exceeding a specified IoU threshold with a ground truth box. False positives include predictions not meeting this threshold with any ground truth box or overlapping a ground truth box already associated with another prediction. False negatives encompass unmatched ground truth boxes.

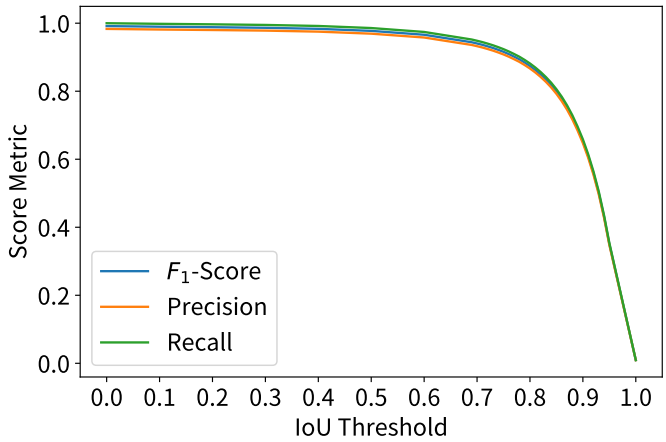

Figure 6.3: F_1-score, precision and recall metrics for text detection on the test set over multiple IoU thresholds

Figure 6.3 illustrates the application of this evaluation approach. The model's performance remains high until an IoU threshold of 85%, after which a sharp decline is observed. Notably, the model maintains a precision of 91% at a 75% IoU threshold, indicating robust detection. Moreover, as detailed in Section 5.4, the downstream character recognition models are trained on randomly cropped data, mitigating the need for perfectly tight bounding boxes.

6.1.2 Qualitative Analysis

To complement the quantitative metrics, we visually examined representative samples from the dataset, showcasing the detector's performance across varying difficulty levels.

In Figure 6.1, the detector adeptly adjusts bounding boxes to capture nuanced details such as the "diacritics" on letters ざ and げ (red arrows), while accurately disregarding reading aids, like the ま written besides the 魔 kanji (blue arrow). Figure 6.6 showcases the model's ability to differentiate between faded text and bleed-through from the reverse side, focusing solely on the relevant characters.

The model also excels in handling dense text, as demonstrated in Figure 6.2. Despite the page containing 534 characters, the detector successfully identifies 531, illustrating its efficacy in complex scenarios.

Furthermore, the model demonstrates proficiency in detecting text written on top of illustrations (Figure 6.5), but struggles with scene text embedded within images (Figure 6.7). This is likely attributable to

Figure 6.5: Illustration with small overlaid text blocks that were detected

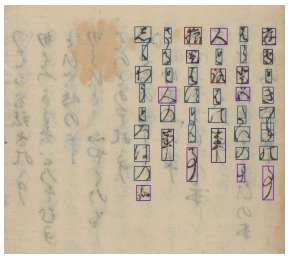

Figure 6.6: Stained document with characters from the reverse side bleeding through

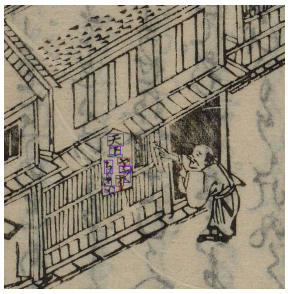

Figure 6.7: Small scene text that the model was not able to detect at all (ground truth bounding boxes are depicted)

the inconsistent and sparse annotation of scene text in the training data, coupled with the small size and distortion often seen in such text.

Another more recurrent problem is that some intricate character sequences still get marked with superfluous and incorrectly sized bounding boxes (Figure 6.4). This problem can be alleviated by post-processing the results. Our current post-processing pipeline involves pruning out bounding boxes with a confidence score below 30% and applying a Non-Max Suppression algorithm with a 55% IoU threshold. Those specific threshold values were hand-picked based on manual experiments. Possibly better thresholds values may be found by using systematic and automated hyper-parameter tuning. Another area for improvement could be the NMS algorithm. Currently, it only does pairwise comparisons but it might be interesting to check if there does not exist two or more bounding boxes nested within a bigger bounding box that have higher confidence, as is the case for the two first characters of the second sequence in Figure 6.4.

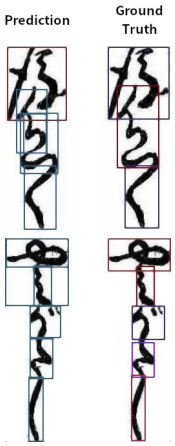

Figure 6.4: Comparison of incorrectly segmented character sequences (left) with the ground truth (right)

6.1.3 Summary

This analysis highlights the model's overall robustness and accuracy in text detection within diverse contexts. However, it also identifies areas for potential improvement, particularly in relation to scene text detection and spurious bounding boxes.

6.2 SKIP Detection Performance

This section presents the performance analysis of our SKIP classifier, which, for reminder, categorizes Japanese characters into six broad groups: punctuation, kanji in left-right formation, kanji in up-down

formation, kanji with an enclosing part, standalone kanji, and kana. As before, the evaluation leverages quantitative metrics, confusion matrices, and error analysis to provide insights into the model's strengths and weaknesses.

The model's overall accuracy on located test set characters is 94.15%. The weighted average F_1-score is **94.12%** with equal precision and recall at 94.15%.

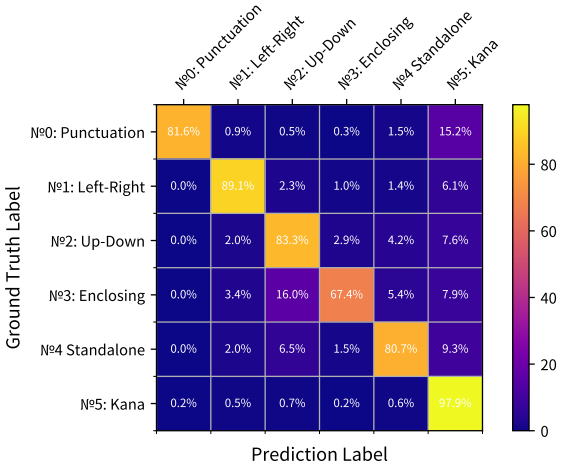

Figure 6.8: Row-normalized confusion matrix for SKIP classification

Figure 6.8 presents a normalized confusion matrix for the SKIP classification. The matrix demonstrates a strong diagonal, indicating a high degree of accuracy, with minimal off-diagonal misclassification errors. The most prevalent confusion arises between patterns №2 and №3, with 16% of kanji featuring an enclosing component being erroneously classified as kanji split into upper and lower parts.

A second similarly frequent type of misclassification error is confusing punctuation for kana. Specifically, 15.2% of typographic symbols are misidentified as kana. The main culprit for this type of error is 〈, a typographic symbol used to repeat the previous kana in text. The visual similarity between 〈 and the kana characters く (ku) and つ (te) was previously mentioned in the Section 3.3.3 *Similarity between Characters*.

The next biggest misclassification errors are in the kana column. Accuracy is very high as the model correctly classifies 97.7% of kana, but this is also the SKIP pattern with the most false positives. Conversely, the model is most conservative when predicting punctuation symbols. A parallel can be made to the frequency of those two categories: kana is by far the most frequent category in the dataset and punctuation naturally takes the last place in frequency.

Figure 6.9 illustrates the specific characters within the SKIP category that are most frequently involved in misclassifications. That way,we can confirm that 〈 is indeed the typographic symbol predominantly misidentified as a kana. Similarly, く is the kana character with the highest frequency of misclassification into the punctuation category. Additionally, this visualization reveals a cluster of four enclosing kanji that are prone to being incorrectly categorized as belonging to pattern №2. It is those characters that were hiding behind the 16% figure we analyzed earlier. Obviously, the four characters displayed in each cell represent only a subset of those involved in misclassifications, with others omitted for the sake of conciseness.

Interestingly, there are instances where there is no need to omit anything as there are less than four characters contributing to misclassifications within a specific scenario. For example, ㊥ is the sole typographic symbol misclassified as an enclosing kanji. 代 and 郎 are the only vertically-split kanji falsely categorized as punctuation. Furthermore, the test set reveals no instances of enclosing kanji being misclassified as typographic symbols.

A final observation we can make is that for each row in the confusion matrix, at least one of the most frequently correctly classified characters (as depicted in the diagonal) is also found off-diagonal, indicating

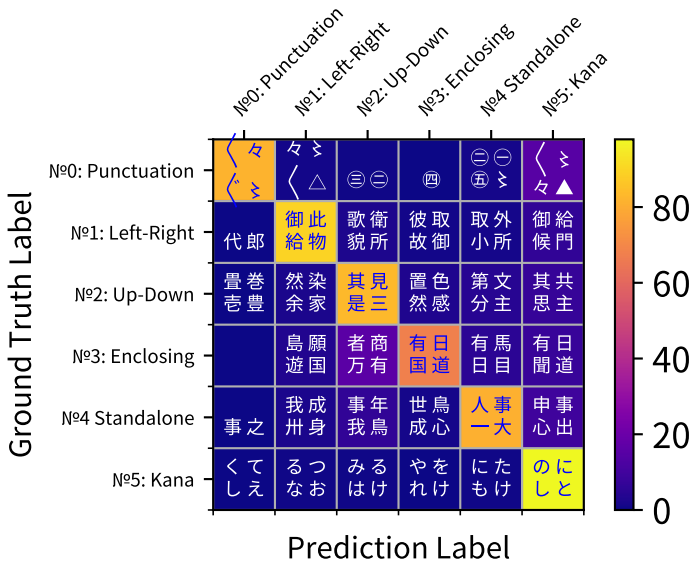

Figure 6.9: Confusion matrix showing the most common characters for each classification scenario

that even characters with ample training examples can still be subject to misclassification. This underscores the inherent challenges in accurately classifying characters within this complex domain.

6.3 Character Recognition Performance

To isolate the effects of the character recognition, we assume, in this section, that all text is perfectly detected and classified into the correct SKIP category. Additionally, all reported measures were done on the test set. With this setup, the mixture-of-experts achieves overall a F_1-score of **94.13%** with a precision of 94.35% and a recall of 94.27%.

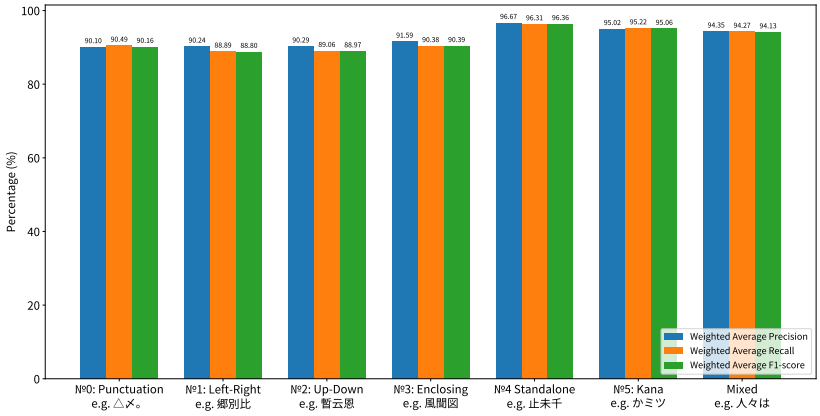

Figure 6.10: Aggregate performance metrics by SVTR model

This high score indicates a strong agreement between the model's predictions and the ground truth labels. In contrast, the *macro* average F_1-score is only 59.66%. Unlike the previously cited numbers, macro average metrics are not weighted by the character frequency. This stark difference suggests that the models have

become proficient at recognizing very frequent characters but struggle on the classification of rare characters, which aligns with our expectations.

This class imbalance can make it difficult to assess the model's performance on these rare classes, as even a few misclassifications can significantly impact the metrics. For instance, 卿 in SKIP category № 1 has high precision but low recall, suggesting that the model is conservative in predicting this character that it has only seen about twenty times in thousands of book pages.

As for the variability in the performance across models, we can see on Figure 6.10 that all models attain similar levels of performance within their domain of expertise. There is not one category that stands out from the others in terms of classification difficulty, despite an uneven distribution of character types among the SKIP categories, as illustrated in Figure 6.11. Please note that "Mixed" in Figure 6.10 refers to using a mixture of the 6 different SVTR models according to the correct SKIP pattern for a kuzushi character. It is not a 7th SVTR model trained on the totality of the input domain.

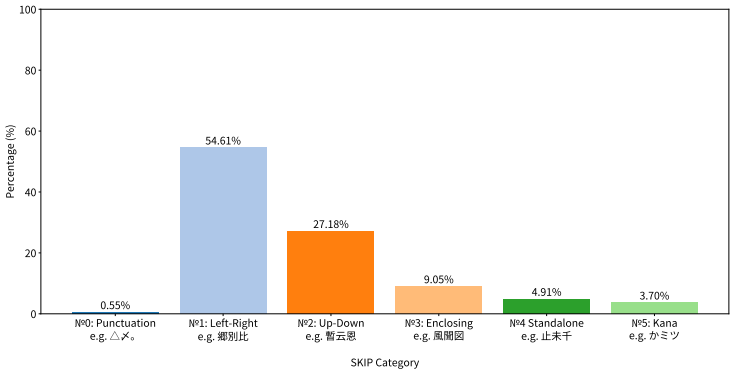

Figure 6.11: Distribution of unique character types among SKIP categories

To improve the performance of the models further, it is interesting to find the pain points: what characters do the models confuse and get wrong.

The first reflex would be to look at the confusion matrix to answer this question but, unlike the previous section, there are thousands of character types so it is unreadable. We need to pinpoint the predictions that tend to be wrong most of the time, as that would be indicative of a character for which the model has failed to learn its discriminative features properly.

To resurface those problematic characters, a custom metric based on sigmoid-weighted regularization was developed:

$$E_{\text{rate}}(c) \times \frac{1}{1+\exp\left(-\alpha \times \left(E_{\text{counts}}(c) - \frac{1}{N}\sum_{i=0}^{N-1} E_{\text{counts}}(c)_i\right)\right)} = E_{\text{rate}}(c) \times \sigma\left(\alpha \times \left(E_{\text{counts}}(c) - \overline{E_{\text{counts}}(c)}\right)\right)$$

Where $N \in \mathbb{N}$ is the number of distinct character types belonging to the SKIP category, $E_{rate}(c) \in \mathbb{R}$ is the error rate for character c, $E_{counts}(c) \in \mathbb{N}^{1\times N}$ is a one-dimensional vector containing the pair-wise error count between character c and every other character in the category, and $\sigma(x)$ is the sigmoid function. The hyperparameter α controls the steepness of the sigmoid curve. A higher α will make the metric more sensitive to smaller differences in error frequencies. Experimentation revealed that $\alpha = 0.5$ provided the most interesting results.

The intent is to find characters with a high error rate weighted by the frequency with which the error is commited while penalizing very frequent characters. Indeed, the problem with a simple error rate metric is that misclassified rare characters that appear only once would have an error rate of 100% and thus appear at the top of the listing but misclassified obscure characters are not what degrade the overall performance the

most. Thus, births the desire to weight those error rates by the frequency but then even a tiny error rate is sufficient to propel extremely frequent characters to the top of the list of confusable characters. Consequently, a regularization term was necessary. The sigmoid function scales the raw error frequency difference in a way that amplifies the importance of moderately frequent errors while downplaying both very rare errors and errors on very common character types.

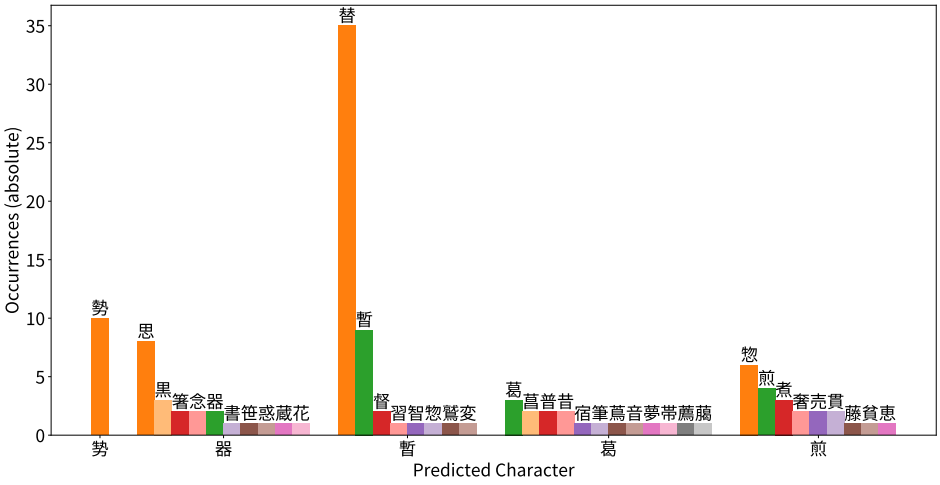

Figure 6.12: Top 5 Error-Prone Characters in Category №2 $\frac{\text{top}}{\text{bottom}}$

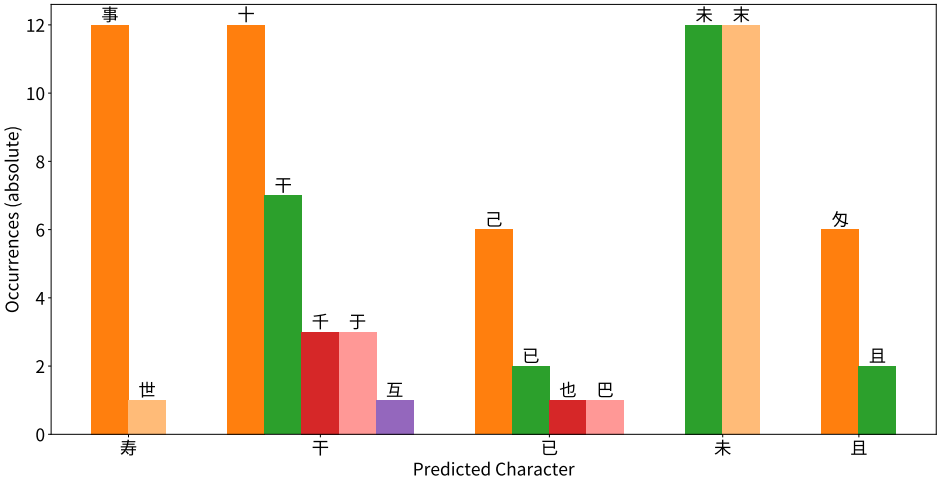

Figure 6.13: Top 5 Error-Prone Characters in Category №4 standalone

The graphs 6.12, 6.13, 6.14, 6.16, and A.1, A.2 in the appendix illustrate the distribution of ground truth characters for the top five error-prone characters for each SKIP category, as measured by our custom metric. The x-axis displays the predicted characters, while the y-axis shows the absolute frequency of occurrence in the test set. Green bars refer to correct predictions while other bars in the group refer to prediction errors, with the actual ground truth character shown above the bar.

Analysis of the error distributions reveals that even for human readers, differentiating between certain character pairs can be challenging due to their subtle visual distinctions, as illustrated by 勢 and 勢 in Figure 6.12 as well as 未 and 未 in Figure 6.13. In the first example, the top left graphical component undergoes a minor alteration from 生 to 垄, while in the second example, the length of the top two strokes is the distinguishing

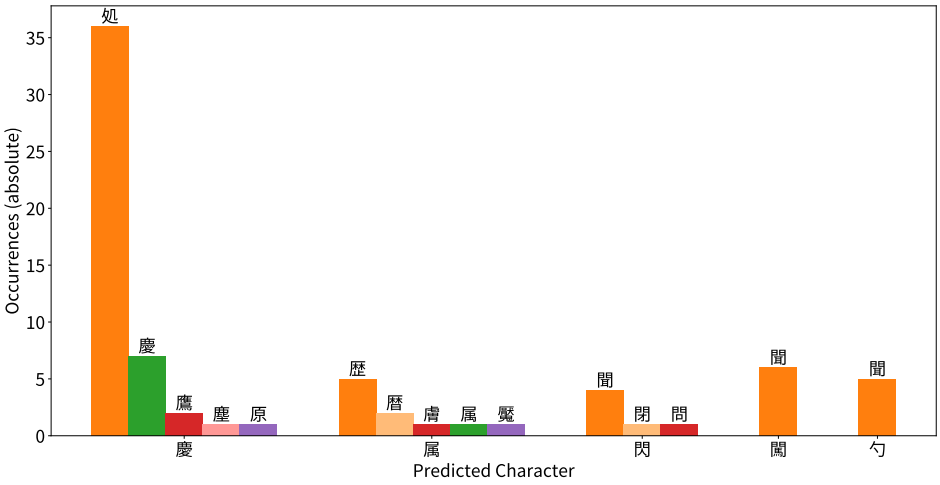

Figure 6.14: Top 5 Error-Prone Characters in Category №3 enclosing

feature. The presence of numerous visually similar kanji in Japanese poses a significant obstacle to accurate recognition, as discerning such fine details proves difficult.

Furthermore, while some character pairs may appear readily distinguishable in the computer typeface employed in the graphs, their kuzushiji counterparts present a greater challenge. For instance, 勺 and 聞, featured in Figure 6.14, exhibit a clear visual difference in block typefaces. However, an alternative handwritten form of 聞 exists that closely resembles 勺 (Figure 6.15).

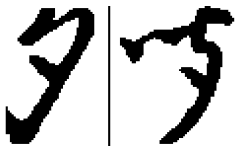

Figure 6.15: Visual comparison of 勺 (left) and 聞 (right)

Aside from that, there is an interesting observation to be made for the Figure 6.16. The test set features circled ideographs. Those are ㊀, ㊁, ㊂, ㊃, and ㊄ which mean the same as ①, ②, ③, ④, and ⑤. The problem is that those circled ideographs never appear in the training set. The output of all SVTR models is technically limited to characters they have seen during training at least once. In the final layer of the model, there are only as many neurons as there are characters in the dictionary configured for the model. Even if one extended the dictionary to include those circled ideographs, in the absence of training samples, the model would have no idea what those characters actually look like. Consequently, the test set performance is naturally degraded due to the presence of out-of-domain characters. Nonetheless, the model classifies them all to the closest available class which is the circle ○. If anything, that shows the generalization ability of the model to recognize circles even if there is an object inside. On a related note, none of the sentence-ending periods in the test set are annotated unfortunately, so this cannot be evaluated without going through the time-consuming task of manually annotating them.

6.3.1 Summary

In the end, the Japanese character classifiers demonstrate promising performance, particularly for common characters, without relying on any surrounding context to guide their decisions. However, there is room for improvement in handling rare and visually similar characters.

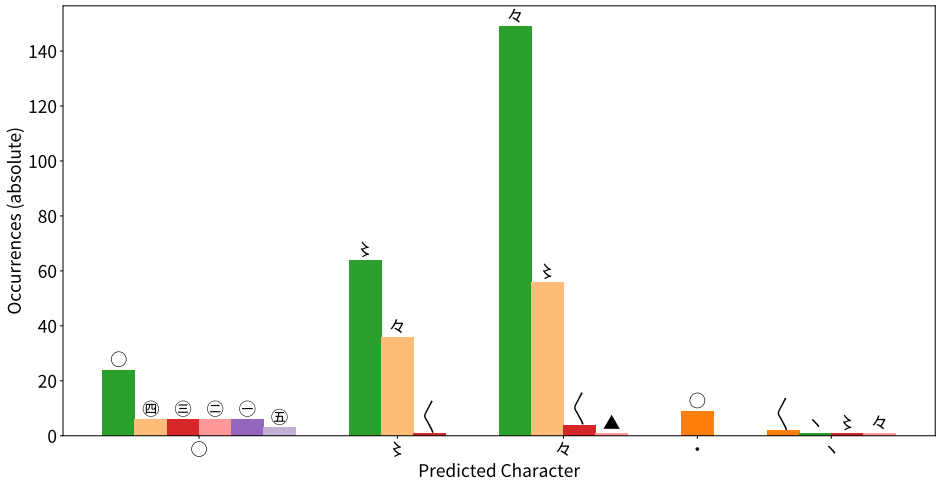

Figure 6.16: Top 5 Error-Prone Characters in Category №0 punctuation

6.4 End-to-End Performance

Text Detection	SKIP Classification	Fine Character Classification
98.48%	94.12%	94.13%

Table 6.1: F_1-scores of separate components on the test set

The F_1-score of the full end-to-end OCR pipeline on the public test set is **87.28%**, with a precision of 86.55% and a higher recall of 88.02%.

This kind of score is similar to the performance achieved by groups that ranked approximately in the top 25 of the competition when it was still running. For reference, a F_1-score of $\approx 91\%$ was necessary to enter the top 10 and the winners had achieved 95%.

The analyses show that the classification components of the pipeline are the bottleneck. This echoes the experience of past competitors, who also observed better performance on the text detection task compared to the kuzushiji classification task.

Chapter 7

Future Work

Building upon the promising results achieved in this study, there are several avenues for further research and improvement in the domain of kuzushi character recognition:

Contextual Modeling Cropping characters for individual recognition may discard valuable contextual information. Exploring models that incorporate surrounding characters or utilize sliding window approaches could enhance accuracy, particularly for ambiguous characters.

Advanced Post-processing While non-maximum suppression has proven effective, investigating other post-processing techniques like language models could further refine the output and correct errors. Leveraging linguistic context, such as n-grams or word-level probabilities, could aid in resolving uncertain character sequences.

Multi-Modal Integration Integrating additional modalities, such as semantic segmentation or layout analysis, could provide complementary information and improve recognition accuracy, especially for complex documents with mixed content.

Previous research on predicting character order in Japanese historical documents, as presented in [46], provides a valuable foundation for further investigation in this field.

Domain-Specific Data Augmentation Augmenting the training dataset with synthetic kuzushi characters could enhance model robustness to variations in handwriting and provide increased exposure to rare characters. Leveraging the compositional nature of kanji, images of infrequent characters could be generated by combining components from more common kanji. For example, the rare kanji 翳 can be decomposed into 医殳羽, for which we have way more samples. However, the cursive nature of kuzushiji poses challenges to this approach, necessitating careful consideration of stroke connections and deformations.

Previous research in synthetic data generation for Chinese characters, such as "Recurrent Net Dreams Up Fake Chinese Characters in Vector Format with TensorFlow" [47] and "A Book from the Sky 天书 Exploring the Latent Space of Chinese Handwriting" [48], offer valuable insights into radical interpolation and linguistic algebra techniques. Additionally, the domain transfer approach described in "Deep Learning for Classical Japanese Literature" [1] presents relevant methodologies for adapting models trained on modern kanji to kuzushiji. Further exploration of these avenues could prove fruitful in generating realistic and diverse synthetic kuzushi characters, thereby improving the overall performance of recognition models.

Chapter 8

Conclusion

This thesis has delved into the intricate world of kuzushiji recognition, exploring its historical significance, the challenges it presents, and the potential of deep learning to unlock the wealth of knowledge hidden within these ancient texts. Through a comprehensive analysis of the kuzushiji dataset and the development of a two-stage OCR pipeline by training deep learning models in the cloud, we have demonstrated the effectiveness of combining a Co-DETR object detector with specialized SVTR character classifiers.

Our approach has achieved promising results, with high F_1-scores in text detection, SKIP classification, and character recognition. The qualitative analysis further highlighted the model's ability to handle diverse challenges, such as character variations, faded text, and complex layouts. However, the evaluation also revealed areas for improvement, particularly in the recognition of rare and visually similar characters, as well as the handling of out-of-domain characters and scene text. The obtained results do not surpass the current state-of-the-art, but this is expected given our limited experience, and the project's constraints in terms of time and resources.

The development of accurate and efficient kuzushiji recognition systems holds immense potential for preserving cultural heritage, facilitating historical research, and making ancient Japanese texts accessible to a wider audience. This Master's thesis has been an enriching experience that married interest in the Japanese language with knowledge in computer science.

Bibliography

[1] Tarin Clanuwat et al. “Deep learning for classical Japanese literature.” In: *Proceedings of 2018 Workshop on Machine Learning for Creativity and Design (Thirty-second Conference on Neural Information Processing Systems)*. Vol. 3. Dec. 2018.

[2] James Curtis Hepburn. 改正増補和英英和語林集成 / *A Japanese-English and English-Japanese Dictionary*. Tokyo: Z. P. Maruya & Co. Limited (丸善商社), 1886.

[3] Tarin Clanuwat et al. *Kuzushiji Recognition*. 2019. URL: https://kaggle.com/competitions/kuzushiji-recognition.

[4] Kazuya Ueki and Tomoka Kojima. “Survey on deep learning-based Kuzushiji recognition.” In: *Pattern Recognition. ICPR International Workshops and Challenges: Virtual Event, January 10-15, 2021, Proceedings, Part VII*. Springer. 2021, pp. 97–111.

[5] Shōji Yamada et al. 古文書翻刻支援システム開発（ＨＣＲ）プロジェクト報告（２） / *Historical Character Recognition (HCR) Project Report (2)*. Japanese. Tech. rep. 51(2001-CH-050). May 25, 2001, pp. 9–16. URL: http://id.nii.ac.jp/1001/00055198.

[6] Tadashi Horiuchi and Satoru Kato. “A Study on Japanese Historical Character Recognition Using Modular Neural Networks.” In: *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*. 2009, pp. 1507–1510. DOI: 10.1109/ICICIC.2009.57.

[7] Satoru Katō and Ryōta Asano. “SOM テンプレートを用いた古文書文字認識に関する研究 | A study on historical character recognition by using SOM template.” Japanese. In: 日本知能情報ファジィ学会ファジィシステムシンポジウム講演論文集 30 (2014), pp. 242–245. DOI: 10.14864/fss.30.0_242.

[8] Taichi Hayasaka, Wataru Ōhno, and Katō Yumie. “ネオコグニトロンによる日本語の歴史的典籍におけるくずし字の認識 | Recognition of obsolete script in pre-modern Japanese texts by Neocognitron.” Japanese. In: 豊田工業高等専門学校研究紀要 48 (2016), pp. 5–12. DOI: 10.20692/toyotakosenkiyo.KJ00010158360.

[9] K. Ueda, M. Sonogashira, and M. Iiyama. “Old Japanese character recognition by convolutional neural net and character aspect ratio.” Japanese. In: *ELCAS Journal* 3 (2018), pp. 88–90.

[10] Xiaoran Hu, Mariko Inamoto, and Akihiko Konagaya. “Recognition of Kuzushi-Ji with deep learning method a case study of Kiritsubo chapter in the Tale of Genji.” In: *Proceedings of the Annual Conference of JSAI 33rd (2019)*. The Japanese Society for Artificial Intelligence. 2019, 4H2E501–4H2E501.

[11] Tarin Clanuwat et al. “文字データの分析：機械学習によるくずし字認識の可能性とそのインパクト | Analysis of Character Data: Potential and Impact of Kuzushiji Recognition by Machine Learning.” Japanese. In: *Journal of Institute of Electronics, Information, and Communication Engineers (IEICE)* 102.6 (June 2019), pp. 563–568. DOI: 10.20676/00000349.

[12] Tarin Clanuwat, Alex Lamb, and Asanobu Kitamoto. “KuroNet: Pre-Modern Japanese Kuzushiji Character Recognition with Deep Learning.” In: *2019 International Conference on Document Analysis and Recognition (ICDAR)*. Sept. 2019, pp. 607–614. DOI: 10.1109/ICDAR.2019.00103.

[13] Ryan Holbrook. *Computer Vision: Build convolutional neural networks with TensorFlow and Keras*. Kaggle Learn Course. 2021. URL: https://www.kaggle.com/learn/computer-vision (visited on 05/27/2024).

[14] Faris Kateb et al. “FruitDet: Attentive Feature Aggregation for Real-Time Fruit Detection in Orchards.” In: *Agronomy* 11 (Nov. 2021), p. 2440. DOI: 10.3390/agronomy11122440.

[15] Valliappa Lakshmanan, Martin Görner, and Ryan Gillard. *Practical Machine Learning for Computer Vision*. 1st ed. http://oreilly.com. Sebastopol, CA: O’Reilly Media, July 2021.

[16] Shaoqing Ren et al. “Faster R-CNN: Towards real-time object detection with region proposal networks.” In: *Advances in neural information processing systems* 28 (2015).
[17] Kaiwen Duan et al. “Centernet: Keypoint triplets for object detection.” In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6569–6578.
[18] Kaiming He et al. “Deep residual learning for image recognition.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
[19] Zhaowei Cai and Nuno Vasconcelos. “Cascade R-CNN: Delving into high quality object detection.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 6154–6162.
[20] Jingdong Wang et al. “Deep High-Resolution Representation Learning for Visual Recognition.” In: *CoRR* abs/1908.07919 (2019). arXiv: 1908.07919. URL: http://arxiv.org/abs/1908.07919.
[21] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context.” In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.
[22] Saining Xie et al. “Aggregated Residual Transformations for Deep Neural Networks.” In: *CoRR* abs/1611.05431 (2016). arXiv: 1611.05431. URL: http://arxiv.org/abs/1611.05431.
[23] Kai Chen et al. “Hybrid Task Cascade for Instance Segmentation.” In: *CoRR* abs/1901.07518 (2019). arXiv: 1901.07518. URL: http://arxiv.org/abs/1901.07518.
[24] Ming Liang and Xiaolin Hu. “Recurrent convolutional neural network for object recognition.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3367–3375.
[25] *miwo: App for AI Kuzushiji Recognition*. Online. URL: http://codh.rois.ac.jp/miwo/index.html.en (visited on 05/23/2024).
[26] ふみのは：古文書解読とくずし字資料の利活用サービス / *fuminoha: Services for deciphering ancient documents and utilizing kuzushiji materials*. Japanese. Online. URL: https://www.toppan.com/ja/joho/fuminoha/ (visited on 05/23/2024).
[27] National Diet Library Lab. *Experimental OCR conversion of rare books and old materials*. Online. 2022. URL: https://lab.ndl.go.jp/data_set/r4_kotenocr_en/ (visited on 05/30/2024).
[28] Minghao Li et al. “TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models.” In: *AAAI 2023*. Feb. 2023. URL: https://www.microsoft.com/en-us/research/publication/trocr-transformer-based-optical-character-recognition-with-pre-trained-models/.
[29] Tamenaga Shunsui. 春色梅児誉美 / *The Colors of Spring: The Plum Calendar*. Japanese. 1833.
[30] Tarin Clanuwat. *Kuzushiji Reconnition[sic] Competition | Tarin Clanuwat | Kaggle Days*. Japanese. Timestamp 20:00. Jan. 27, 2020. URL: https://www.youtube.com/watch?v=cGpIVyV96Hg (visited on 02/23/2024).
[31] Kōta Kodama. くずし字用例辞典 / *Dictionary of Kuzushiji Examples*. Japanese. Fukyūban, shinsō shohan. Tōkyō: Tōkyōdō Shuppan, 1993.
[32] Kristen Dexter. *How to Find The Kanji Radical*. Online. Nov. 18, 2014. URL: https://www.tofugu.com/japanese/how-to-find-the-kanji-radical/#missing-radicals (visited on 05/08/2024).
[33] Agrim Gupta, Piotr Dollar, and Ross Girshick. “LVIS: A Dataset for Large Vocabulary Instance Segmentation.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019.
[34] Zhuofan Zong, Guanglu Song, and Yu Liu. *DETRs with Collaborative Hybrid Assignments Training*. 2022. arXiv: 2211.12860.
[35] Ze Liu et al. “Swin transformer: Hierarchical vision transformer using shifted windows.” In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 10012–10022.
[36] Ashish Vaswani et al. “Attention is all you need.” In: *Advances in neural information processing systems* 30 (2017).
[37] Alexey Dosovitskiy et al. “An image is worth 16x16 words: Transformers for image recognition at scale.” In: *arXiv preprint arXiv:2010.11929* (2020).
[38] Nicolas Carion et al. “End-to-end object detection with transformers.” In: *European conference on computer vision*. Springer. 2020, pp. 213–229.
[39] Yongkun Du et al. “SVTR: Scene Text Recognition with a Single Visual Model.” In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*. Ed. by Lud De Raedt. Main Track. International Joint Conferences on Artificial Intelligence Organization, July 2022, pp. 884–890. DOI: 10.24963/ijcai.2022/124. URL: https://doi.org/10.24963/ijcai.2022/124.

[40] Canjie Luo et al. “Learn to augment: Joint data augmentation and network optimization for text recognition.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 13746–13755.
[41] Shuai Shao et al. “Objects365: A large-scale, high-quality dataset for object detection.” In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 8430–8439.
[42] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database.” In: *CVPR09*. 2009.
[43] Max Jaderberg et al. “Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition.” In: *Workshop on Deep Learning, NIPS*. 2014.
[44] Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. “Synthetic data for text localisation in natural images.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2315–2324.
[45] Haiyang Yu et al. “Benchmarking Chinese text recognition: Datasets, baselines, and an empirical study.” In: *arXiv preprint arXiv:2112.15093* (2021).
[46] Tarin Clanuwat et al. “Predicting the Ordering of Characters in Japanese Historical Documents.” In: *CoRR* abs/2106.06786 (2021). arXiv: 2106.06786. URL: https://arxiv.org/abs/2106.06786.
[47] David Ha. “Recurrent Net Dreams Up Fake Chinese Characters in Vector Format with TensorFlow.” In: *blog.otoro.net* (2015). URL: https://blog.otoro.net/2015/12/28/recurrent-net-dreams-up-fake-chinese-characters-in-vector-format-with-tensorflow/ (visited on 05/30/2024).
[48] Gene Kogan. *A Book from the Sky* 天书. Online. Exploring the Latent Space of Chinese Handwriting. Dec. 2015. URL: https://genekogan.com/works/a-book-from-the-sky2/ (visited on 05/30/2024).

Appendix A

Supplementary Graphs on Error-Prone Characters

Supplementary graphs, related to the analysis conducted in Section 6.3, provides distributions of ground truth characters for the top five error-prone characters, as identified during recognition by SVTR models for SKIP categories №1 and №5. Due to the high frequency of occurrences within the kana category, Figure A.2 uses a logarithmic scale, unlike all other figures presenting similar data.

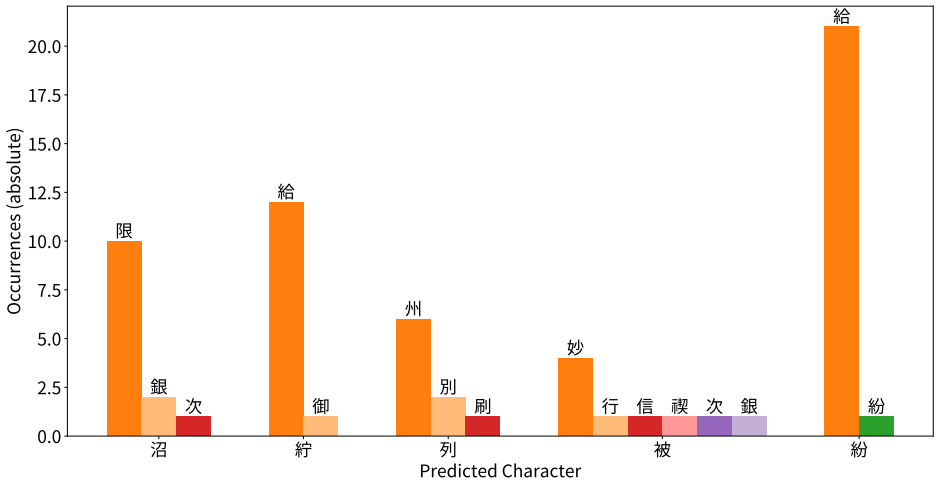

Figure A.1: Top 5 Error-Prone Characters in Category №1 left|right

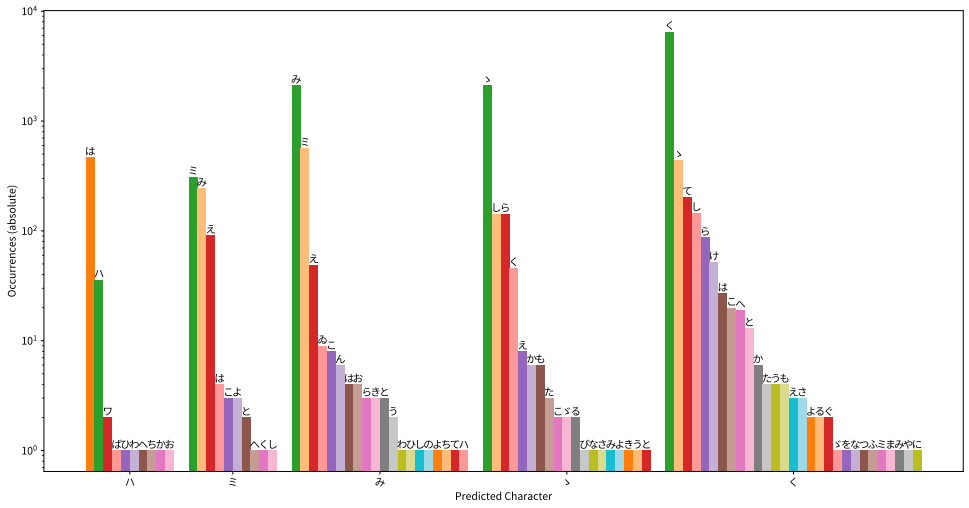

Figure A.2: Top 5 Error-Prone Characters in Category №5 kana, on a logarithmic scale

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**