Machine Learning for Japanese Kanji

Supervised by Dr. Cedric Spire & Dr. Sulthana Belgum

Isaac Temidayo Awokoya
220074391

Research Dissertation presented for the degree of
Masters of Science (MSc) Data Analytics

Department of Engineering and Physical Sciences
Aston University
United Kingdom
November 2023.

Abstract

In recent years, machine learning has emerged as a powerful tool for various applications in the field of natural language processing and character recognition. This paper presents an in-depth exploration of the application of machine learning techniques for Japanese Kanji detection. Kanji which is an integral part of the Japanese writing system, poses unique challenges due to its complexity and the vast number of characters involved. The ability to accurately detect and classify Kanji characters has significant implications for a wide range of applications, including optical character recognition (OCR), language processing, Image-text detection and conversion and information retrieval. This study investigates the effectiveness of the machine learning algorithm, Convolutional neural networks (CNNs)with conjoining help frpm previous work which included neural machine translation (NMT), recurrent neural networks (RNNs), and support vector machines (SVMs), in the task of Kanji detection. Various pre-processing techniques such as loading, grayscale conversion, thresholding, contour detection, bounding boxes, and extraction of characters were utilised to enhance the model's performance. Additionally, a dataset comprising a diverse set of extracted handwritten Kanji characters was used for training and validation. The primary research aim and objective is to explore the potential of machine learning, deep learning, and artificial intelligence in facilitating the detection of Japanese Kanji characters. Furthermore, we delve into the challenges of handling class imbalance issues, where some Kanji characters are significantly more common than others, and propose strategies to mitigate this problem. The experimental results demonstrate the potential of machine learning in accurately detecting Japanese Kanji characters, achieving high precision and recall rates. The findings have practical implications for improving the accuracy and efficiency of OCR systems for Japanese text and contribute to the broader field of character recognition. In conclusion, this paper provides valuable insights into the application of machine learning for Japanese Kanji detection. The research contributes to the growing body of knowledge in character recognition and paves the way for enhanced Kanji-related applications in the realms of language processing, document analysis, and information retrieval.

Acknowledgement

I'd like to convey my heartfelt appreciation to everyone who has been there for me during my journey. Firstly, I want to express my gratitude to God for blessing me with good health and a stable mental state, which enabled me to successfully complete this project. My family deserves special thanks for their unwavering love and support; their belief in me and constant encouragement played a pivotal role in my achievements.

I am also deeply appreciative of my research advisor, Dr. Cedric Spire and co supervisor, Dr. Sulthana Belgum, for their invaluable guidance, patience, and expertise. Their mentorship has been of immense significance to me, and I am thankful for the opportunities they've provided. Additionally, I want to extend my thanks to my colleagues, including Beverly Muskwe, Eniola Badejo, Daniel Oseni, Tobi Ogunleye, for their unwavering professional and personal support and encouragement. Their constant inspiration has meant a lot me.

I'm grateful to the dedicated staff and faculty of the Department of Engineering and Physical Science at Aston University. Their commitment to excellence has made my academic journey truly fulfilling. To all of you, I extend my heartfelt thanks for your support and encouragement. My goals would not have been achievable without your presence in my life.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

1. ML - Machine Learning
2. AI - Articicial Intelligence
3. NLP - Natural Language Processing
4. SVM - Support Vector Machine

5. RNN - Reccurent Neural Network
6. CNN - Convolutional Neural Network
7. KNN - K-Nearest Neighbor
8. GP - Gaussian Process
9. OCR - Optical Character Recognition
10. NMT - Neural Machine Translation
11. CV - Computer Vision
12. BoW - Bag of Words
13. PIL - Python Imaging Library
14. RELU - Rectified Linear Unit

Keywords:

Machine learning, Optical Character Recognition (OCR), Neural Machine Translation (NMT), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Support Vector Machines (SVMs)

1 Introduction

1.1 Machine Learning

In recent years, the field of Machine Learning has seen a surge in developments and applications, particularly in the realm of language processing and recognition. Machine Learning (ML), being a branch of Artificial Intelligence is dedicated to the utilization of real world data to recreate and imitate learning, the way humans learn. This being a fundamental pillar in the age of computing that has given rise to multiple AI software that automates real world scenarios in computational and economical society with the aim of constant improvement in the field.
A category of ML that will be the focus of this research is called Natural Language Processing (NLP), which is also a branch of AI and Machine Learning that grants computers the understanding and processing ability of Naturally occurring languages the way humans understand them.
According to the University of Stanford, *"The field of natural language processing began in the 1940s, after World War II. At this time, people recognized the importance of translation from one language to another and hoped to create a machine that could do this sort of translation automatically."*. [17] Dr Noam Chomsky, in 1958 was the first to notice the early irregularities of early NLP modelling when grammatically incorrect and incoherent sentences were recognized with as much accuracy with grammatically correct and coherent sentences.[17] His displeasure gave a rise to further researching to improve the model to what we have today.
Modern NLP is utilized in several aspects of life and several fields of study, for instance, the use of Speech Recognition and Text-to-Speech in mobile and computer devices which converts orally spoken words to text and text to computer generated Audio, respectively, and Machine Translation, which is utilized to translate text from one language to another using Machine Learning, for instance Japanese to English.

1.2 Japanese Kanji

Japanese is a complex language with a mixture of hiragana, katakana, and kanji characters.
Japanese Kanji (漢字) [15]identifies as the character script used in Japanese Language, it was adopted from Chinese Characters in Hanzi, in the 5^{th} century and has a vast number of individual characters with multiple readings and meanings with the script of Japanese for Kanji refered to as logographic kanji[2] [8]

Japanese is spoken majorly by people native to Japan. According to worlddata.info it is native to approximately 124 Million people predominantly live in Japan, while other native speakers who migrated to the United States and Brazil.[18]

Figure 1: Image of Japanese Kanji [13]

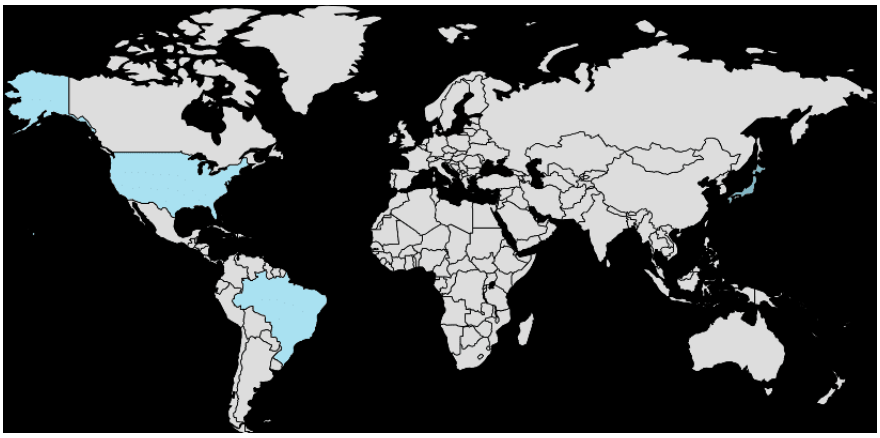

Figure 2: Image of Native Japanese Speakers around the World [18]

The Kanji character is difficult to process and read due to its complexity, only a certain percentage of native speakers can read and write kanji due to the vast number of characters and the different methods of pronunciation and context of characters. According to Akira Katakami, a professional Japanese Translator, says *"Kanji involves quite a bit of memorization and a whole lot of dedication. There are about 50,000 Kanji characters. However, very few native Japanese speakers actually know or use anywhere near this many. At the end of their compulsory education, a child will have learned about 2200 characters, which is approximately the number of Jōyō kanji – the most commonly used characters in everyday life. A native speaker who grew up in the country may know about 3500 to 4000, while a fairly well-read academic might know about 5000."* [19]

1.3 Aims and Objectives

The primary aim and objective of this research is to explore the potential of machine learning, deep learning, and artificial intelligence in facilitating the detection of Japanese Kanji characters. By delving deep into the existing literature and analyzing various tools and techniques currently being employed in this field, this research aims

Table 1: Language Distribution by Country [18]

Country	Region	Official language	Distribution	Total
Japan	East Asia	yes	99.1%	123,999,000
United States of America	North America	no	0.2%	667,000
Brazil	South America	no	0.2%	431,000
Guam	Micronesia	no	0.5%	3,000

to carve out a pathway for future developments in Kanji detection systems. Further improvement on the Machine Learning model, which can simply be defined as a program that can identify and solve problems with the aid of learning data, would be incorporated to develop translation of the Kanji Characters to English.

1.4 Research Scope

This research will focus on understanding the deep learning algorithms and neural network structures that are pivotal in recognizing and translating Kanji characters. It will also explore the potential applications of these technologies in various domains including computer vision, natural language processing, and speech recognition.

2 Literature Review

There are limited articles relating to Machine Learning for Japanese Kanji Detection but there are multiple articles that give insight on how this project can be achieved. This literature review focuses on the tools and methods being used for Machine Learning for Japanese Kanji and how the data can be collated, preprocessed and extracted, how the data will be parsed through and exactly how and which deep learning algorithm can be utilized to accurately detect and translate the Kanji characters to English.

2.1 Deep Learning tools for Kanji Detection

Dzmitry Bahdanau et all discusses how Neural Machine Translation (NMT) uses deep learning encoders to read in source material database (the Kanji characters) into vectors of usual fixed length, then utilizes decoders to output the translated phrase or word from the encoded vector. These two coders, which are usually described as encoder-decoder system are always being trained together to optimize the accuracy of getting a correct translation. [1]
Charlie Tsai also discusses convolutional neural networks (CNN) and how it is utilized for recognizing handwritten Japanese, his work focuses on n the classification of the type of script, character recognition within each type of script, and char-

acter recognition across all three types of scripts. This is also a method of deep learning that consists of layers that transform an input 3-dimensional volume into another output 3-dimensional volume through a differentiable function. Since the database being utilized involves images, CNN transforms the original image through each layer in the architectures to produce a class score. Since the input consists of images, the neurons in a convolutional layer of a CNN have an activation "volume" with width, height, and depth dimensions [2]
Y. Bengio et al published an article as well on Deep learning which provides an overview of deep learning, a subfield of machine learning that focuses on training artificial neural networks with multiple layers to learn hierarchical representations of data. Applications of deep learning in various domains, including computer vision, natural language processing, and speech recognition was also highlighted in the article with possibilities on what deep learning can accomplish in each respective field. [12]

2.2 Optical Recognition tools for Kanji Detection

The Data sets being used involve images of hand written Kanji which can be read in using Optical Character Recognition (OCR), it as a prominent tool of Computer Vision for text recognition. OCR software singles out letters on the image, puts them into words and then puts the words into sentences, thus enabling access to and editing of the original content. It also eliminates the need for manual data entry. [3]
MD Anwar Hossain also discusses OCR with the use of Template Matching, which is a technique used in finding areas of images that match templates. it is a high level machine visibility technique that destines the parts of an image that match a predefined template. Template Matching involves determining the similarities between a given pattern and windows of the same size in an image and identifying the window that produces the highest similarities measure. It works by comparing derived image features of the image and the template for each possible displacement.
Yago Diez et all discuss on computer vision and deep learning tools for the automatic processing of Wasan documents with blob detection, a computer vision technique was utilized to detect the 20 strokes of the kanji characters being used as the data. This technique isolates the Kanji on the document and with the use of Convolutional Neural Networks(CNN) as their deep learning technique, the detection of the 'íma' character was proven successful, The Wasan document was of low quality and the Computer Vision (CV) technique was able to isolate the kanji and the CNN model was able to detect the required character 'ima'. [21]

2.3 Translation Techniques for Kanji Detection

David Allen and Kathy Conklin, authors of Cross-linguistic similarity norms for Japanese–English translation equivalents[6], discuss on formal and semantic overlap that occurs between two languages which is a major underlying factor in bilingual language processing. They discuss how "Japanese (first language; L1)–English (second language; L2)" [6] can vary in terms of cross-linguistic overlapping, which simply means words translated to Japanese from English do not necessarily equate to the same word when the English word is translated to Japanese.[6] This provides insight for the training and validation data moving forward, their article provides information for the label for the data that will be used for the Machine Learning module. This label is in charge of providing context to the dataset for the Machine Learning model to understand.

Chooi-Ling Goh et all discusses about using existing dictionaries, Japanese-English and English-Chinese to build Japanese-Chinese dictionary in their research paper Building a Japanese-Chinese Dictionary using Kanji/Hanzi Conversion[28], which is done with Machine converters, which has significant accuracy of 77% which can be said is bolstered by the similarities between Kanji and Hanzi, where Japanese Kanji was derived from.[28] Insight is provided in this paper and it gives more understanding to data labelling for the Machine Learning model to have the appropriate Kanji character for the extracted image data and not Hanzi Characters. Even though both share similarities, Kanji characters are the main focus and Hanzi characters can mislead the model and inhibit appropriate learning.

2.4 Literature Review Summary

Throughout the Literature review, these researchers have provided a foundation on how Deep Learning methodologies can provide solutions to the difficult task of getting accurate readings of Kanji characters and non English characters alike, from images and handwritten Kanji.

Charlie Tsai's paper for example, provides insight on how Convolutional Neural Networks can accurately handwritten Japanese by classifying the type of scripts and identifying between each script, namely Kanji, Hirigana and Katakana. Charlie Tsai's incorporation of Deep Learning and Convolutional Neural Networks provides motivation that the CNN modell is the best model to utilize to reach this projects aim and objective.

MD Anwar Hossain also provides insight on how Optical Character Recognition plays a massive part in dealing with training image data, as the computer needs to visualize the image the same way human beings do for this to work. Open CV also provides access to manipulate image data in any virtual environment such as JupyterLab for example, which is a virtual environment for Python programming Language.

David Allen and Chooi-L ing Goh also give an appropriate base approach for the model to attempt Kanji Translation.
The objective of this research is to detect Japanese Kanji with the aid of Machine Learning and attempt to translate the detected Kanji to its English translation. This project can be a stepping stone towards more accurate understanding of Kanji and bridge the gap between Native Japanese speakers and their willingness to understand and write Kanji.

3 Theory

This section discusses the terminologies involved for this research project and how it was utilized to achieve the final objective

3.1 Deep Learning

Deep learning which is under the umbrella of machine learning that utilizes neural networks with multiple layers to automatically learn hierarchical representations from data.[16]
It is a machine learning technique that teaches computers to do what comes naturally to humans
Deep learning is a sub field of machine learning that focuses on training artificial neural networks with multiple layers to learn hierarchical representations of data. It is inspired by the structure and functioning of the human brain, particularly its interconnected network of neurons.[16] [24]
The Neural Networks are constructed with several hidden layers between the input layers and output layers. They each consist of a set of artificial neurons, that process and transform the input data. The layers are interconnected by weighted connections, and each neuron applies a non-linear activation function to its input before passing it to the next layer.[4]
The depth of the network, achieved by having multiple hidden layers, enables deep learning models to learn intricate representations of data. Each layer learns to extract higher-level features from the previous layer's output. By progressively combining and transforming these features, the network can learn complex patterns and relationships in the data.
A good use of deep learning is its ability to learn features from raw data on its own. This thereby stops the need for manual feature engineering. Deep learning models can discover meaningful representations and hierarchies of features by iterative adjustments to the connection weights during the training process. This makes deep learning particularly powerful in tasks such as image and speech recognition, natural language processing, and many other areas where complex patterns need to be learned from large amounts of data.[29]

The reason as to why Deep Learning is being utilized for this research proposal is due to the models vast learning experience and capabilities. It has been determined from the technique makeup as the best Machine Learning tool that can be used for supervised training on labelled image data.

3.2 Machine Learning

Machine learning refers to the use of algorithms and statistical models to enable computer systems to automatically learn and improve from experience without being explicitly programmed.[10] It involves the development and application of computational techniques that allow systems to recognize and detect Japanese Kanji characters based on patterns and features extracted from data.

3.3 Artificial Intelligence

Artificial intelligence (AI) refers to the broader field of computer science that aims to develop intelligent systems capable of performing tasks that typically require human intelligence. AI encompasses the design and development of algorithms, models, and techniques that enable machines to exhibit intelligent behavior and make decisions based on input data. Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems [9]

3.4 Neural Networks

Neural networks are computational models inspired by the structure and function of biological neural networks. These networks consist of interconnected nodes (neurons) that process and transmit information. Recurrent Neural Networks (RNNs), where connections between nodes can create a cycle, allowing output from some nodes to affect subsequent input to the same nodes have been explored for sequence-based Kanji recognition tasks, taking into account the inherent temporal dependencies in handwritten or stroke-based Kanji writing. A neural network takes an input vector of p variables X = (X1, X2,...,Xp) and builds a nonlinear function f(X) to predict the response Y . [5]

An artificial neural network is a model of computation inspired by the structure of neural networks in the brain. In simplified models of the brain, it consists of a large number of basic computing devices (neurons) that are connected to each other in a complex communication network, through which the brain is able to carry out highly complex computations. Artificial neural networks are formal computation constructs that are modeled after this computation paradigm. [5]

The idea behind neural networks is that many neurons can be joined together by communication links to carry out complex computations

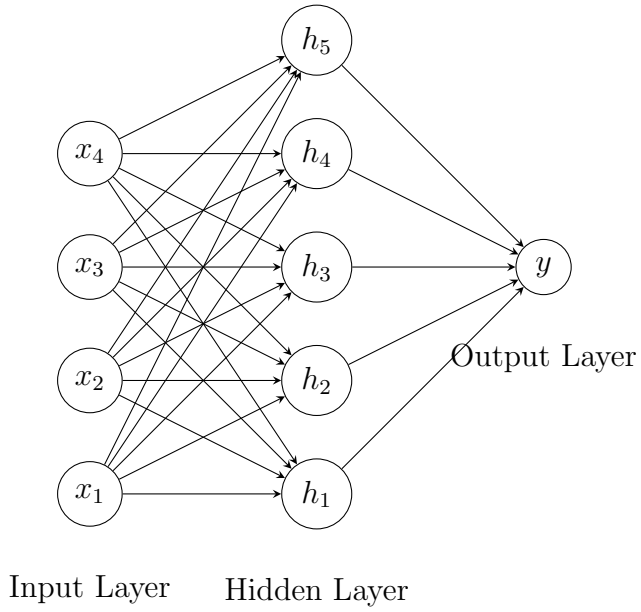

Figure 3: Simple Neural Network Diagram

3.5 Convolutional Neural Network (CNN)

3.5.1 Training and Optimization.

A custom CNN architecture was employed, consisting of:

- **Convolutional Layers:** These layers, equipped with multiple filters, extract hierarchically complex features from the input images. Activation functions introduce non-linearity, enabling the model to learn intricate patterns.
- **Pooling Layers:** Pooling operations reduce the spatial dimensions of the feature maps, thereby reducing computational demands and preventing overfitting.
- **Dropout:** Dropout layers were incorporated to prevent co-adaptation of neurons, further mitigating overfitting risks.
- **Fully Connected Layers:** Dense layers at the end of the network facilitate the final decision-making process, determining which character the input image corresponds to.
- **Output Layer:** The final layer employs a softmax activation function to yield probabilities for each potential character class.

3.6 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a subfield of artificial intelligence that focuses on enabling machines to understand, interpret, and generate human language. This understanding could range from simple tasks (like identifying the language of the text) to complex ones (like sentiment analysis, machine translation, and summarization). Components of NLP include Tokenization which is the conversion sequential characters in text to sequential tokens (words, subwords, or characters). It's the foundational step in text processing. [25] and Bag-of-Words (BoW)which represents a document as a vector where each dimension corresponds to a unique word in the document, and its value represents the frequency or presence (binary) of that word.

Given a document d and vocabulary V of size N, the BoW representation $\mathbf{v}$ is given by:

$$\mathbf{v} = [f_1, f_2, \ldots, f_N] \tag{1}$$

where f_i is the frequency of the i^{th} word in d.

Word Embeddings Word embeddings are dense vector representations of words where semantically similar words are close in the embedding space. Techniques like Word2Vec, GloVe, and FastText, which are all Tensorflow packages are used to generate these embeddings.[**?**]

The core idea of Word2Vec (in the Skip-Gram variant) is to maximize the probability of observing context words given a center word. This can be represented as:

$$\max \prod_{w_c \in V} \prod_{w_o \in \text{context}(w_c)} P(w_o|w_c) \tag{2}$$

Recurrent Neural Networks (RNNs) RNNs are used to process sequences, which makle then perfect for NLP projects. They maintain a hidden state that captures information from processed tokens.[26]
An input sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$, the hidden states h_t and output y_t at each time step t are computed as:

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h)yt \tag{3}$$

$$= W_{hy}h_t + b_y \tag{4}$$

where W_{hh}, W_{xh}, and W_{hy} are weight matrices, and b_h and b_y are biases.

Transformers and Attention Mechanisms Transformers, which utilize attention mechanisms, are currently state-of-the-art for many Natural Language Processing (NLP) tasks. Attention allows the model to focus on different parts of the input

text differently, enabling it to capture long-range dependencies.[27]

A query Q, key K, and value V, the attention output can be seen as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{5}$$

whith d_k being the dimensionality of the key.

4 Data

Natural Language Processing using Machine Learning requires image or audio data to process.
The data was obtained by writing down Kanji characters on a plain sheet of paper to avoid detection ambiguity from the OCR.
Two OCR Packages were utilized for data, CV2 and Tesseract. CV2 is the prebuilt OpenCV python package [20] utilized in reading in the image into the Jupyter Lab environment.
Cv2 is mainly utilized for the easy manipulation of image and video data.
Tesseract is the second OCR in use and the function of this Python package is to convert the image kanji character to text to enable easy labelling for the image data.
The Image Data utilized in training the model are 3 handwritten Kanji Characters namely (火), which translates to "Fire" in English, (土) with the English Translation "Earth"and lastly, (本), which translates to "Book". These Kanji Labels were selected due to their simplicity to write and simplicity to read and translate, especially for this developing research project being conducted. The Table below gives a descriptive overview of the Data:

Kanji Character	(火)	(本)	(土)
English Translation	Fire	Book	Earth
Number of Image Data	18	13	20
Image Extension	.jpg	.jpg	.jpg

Table 2: Image Kanji data description

From Table 2, The image data extracted is seen to be 51 in total, compromised of 3 labels which interpret to Fire, Earth and Book. The Extracted images are stored as .jpg (Joint Photographic Experts Group) [30] image formats and this was chosen for its efficient image compressing qualities and versatility across different operating systems and webpages.

Data labelling, which is the attaching of Kanji text to relevant extracted k anji image was achieved through an online Kanji dictionary known as Kanshudo. This provided accurate descriptions and labelling that matched the extracted image data and provided the accurate Kanji translations for each character .
This image provides context on the online dictionary. It also provided a step by step tutorial on how to properly write Kanji Characters which was the technique carried out when creating the Training data for the Machine Learning Model.

This Figure 5 visualises the step by step procedure taken to prepare the dataset for Machine Learning model evaluation. The steps are explained below:

Figure 4: Online Kanji Dictionary Kanshudo

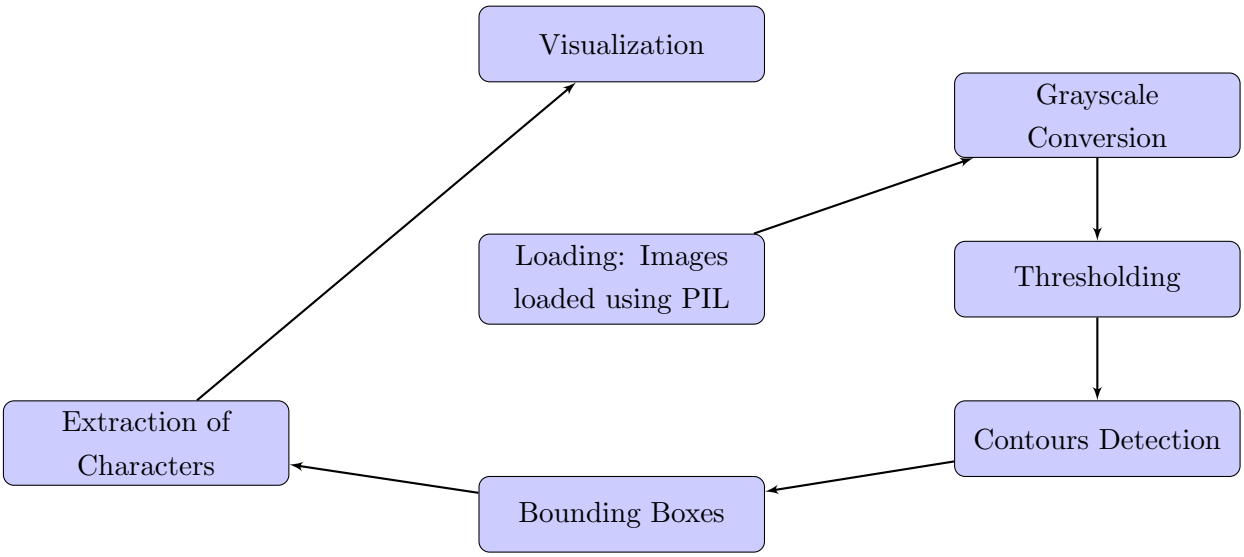

Figure 5: Data Preparation Flowchart

4.1 Data Preprocessing

1. **Loading:** The images were initially loaded using the Python Imaging Library (PIL) for further processing.

2. **Grayscale Conversion:** Each image was converted to grayscale to reduce the dimensionality of the data, which simplifies the learning process and reduces computation time.

3. **Thresholding:** Otsu's thresholding method was applied to convert the grayscale images into binary format, highlighting the Kanji characters against the background.
   Otsu's Thresholding technique can be defined as a technique that "determines an optimal global threshold value from the image histogram", according to the cv2 documentation. [31]

4. **Contours Detection:** Using the binary images, contours were detected to

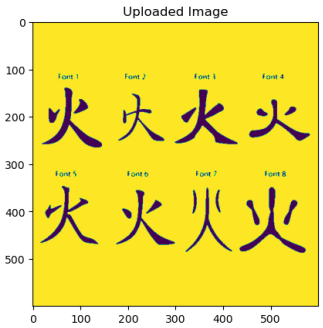

Figure 6: Example Image of Data

identify distinct characters or components in the image. This step is crucial for segmenting individual Kanji characters, especially when an image contains multiple characters.

5. **Bounding Boxes:** For each detected contour, a bounding box was computed. These bounding boxes serve as regions of interest, encapsulating individual Kanji characters. This segmentation allows for the extraction and individual processing of each character from the larger image.

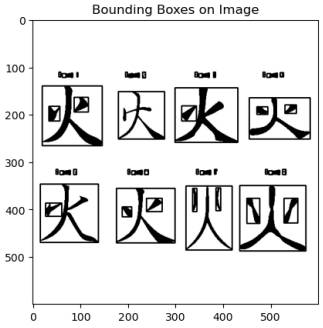

Figure 7: Example Image of Bounding Boxes

The main interest here, is from the bounding box that surround the whole Kanji, not just the boxes containing small strokes that are apart of the Kanji. This is essential as the small strokes would lead to poor information processing by the model if trained on its own, which hampers the Training and Validation Accuracy.
The bounding box detection was achieved using cv2 package in Python.

6. **Extraction of Characters:** Based on the computed bounding boxes, each Kanji character was cropped from the original image, resulting in a dataset of individual character images.

   It is ensured the whole Kanji is extracted, not including the smaller bounding boxes detected earlier by the OpenCV tool.

Figure 8: Example Image of Extracted Data

4.2 Data Organization:

Each Kanji character extracted was saved to ensuring uniqueness and to avoid any data loss or overwriting. Then the characters were written in a structured directory for easy access and training. Additionally, a CSV file, `newlabels.csv`, was maintained to keep track of each character image and its corresponding label, facilitating supervised learning.

Figure 9 and 10 visualise the newlabels.csv file and shows adequately that all extracted Kanji are stored uniqely and sequentially.

[537]:

	Character Name	Label	English Translation
0	character_1.jpg	火	Fire
1	character_2.jpg	火	Fire
2	character_3.jpg	火	Fire
3	character_4.jpg	火	Fire
4	character_5.jpg	火	Fire
5	character_6.jpg	火	Fire
6	character_7.jpg	火	Fire
7	character_8.jpg	火	Fire
8	character_108.jpg	火	Fire
9	character_109.jpg	火	Fire

Figure 9: newlabels.csv entries I

22	character_122.jpg	土	Earth
23	character_123.jpg	土	Earth
24	character_124.jpg	土	Earth
25	character_125.jpg	土	Earth
26	character_126.jpg	土	Earth
27	character_127.jpg	土	Earth
28	character_128.jpg	土	Earth
29	character_129.jpg	土	Earth
30	character_130.jpg	土	Earth
31	character_131.jpg	土	Earth
32	character_132.jpg	本	Book
33	character_133.jpg	本	Book
34	character_134.jpg	本	Book
35	character_135.jpg	本	Book

Figure 10: newlabels.csv entries II

The following table also gives a description of the newlabel.csv file, which indicate there are no null values:

Property	**Value**
Dataframe Class	pandas.core.frame.DataFrame
Index Range	0 to 52
Number of Entries	53
Columns	
Character Name	53 non-null object
Label	53 non-null object
English Translation	53 non-null object
Data types: object(3)	
Memory usage: 1.4+ KB	

Table 3: newlabel.csv Information

Table 3 provides the metadata about a newlabels.csv. The dataframe has 53 entries with three columns detailing the character's extracted image identifier, the Kanji label and its English translation.

A Descriptive Visualisation of the Image Data used for The ML module can be seen below. It features:

1. The Image weight & height distribution of the extracted characters 11

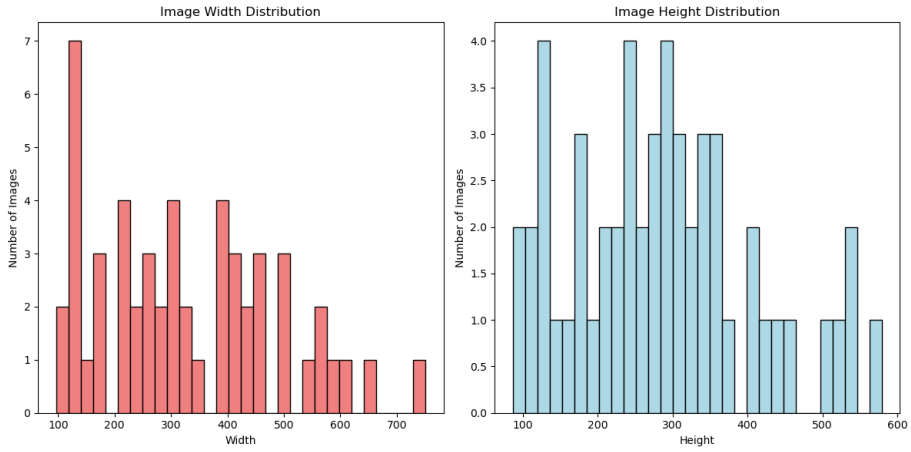

Figure 11: Image Distribution of Extracted characters

2. The Class Distribution of Kanji labels 12

3. The Class Distribution of the English Translations 13

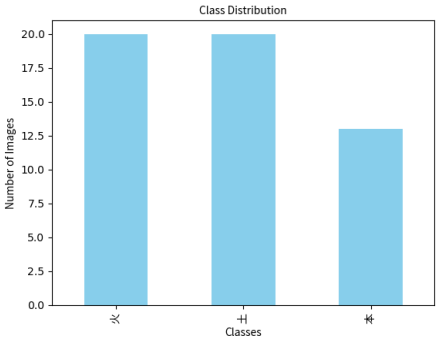

Figure 12: Distribution of Extracted Character Labels

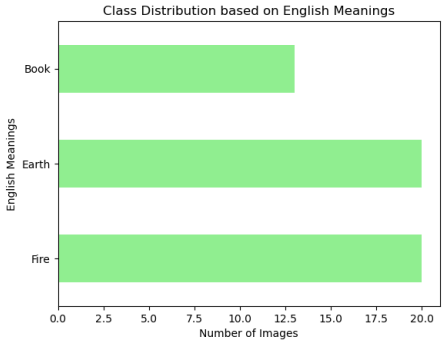

Figure 13: Distribution of English Translations for Labels

The Dataset used for this research is quite small, due to the immense complexity of labelling each extracted character from extensive Kanji image data repositories like ETL9G and Kuzushiji-Kanji which have over 10,000 different images. This research focuses on utilizing an ML model that will be able to detect test data given to it and also accurately deploy its English Translation.

As stated earlier in Section 4, the Image Data utilized in training the model are 3 handwritten Kanji Characters namely (火), which translates to "Fire" in English, (土) with the English Translation "Earth"and lastly, (本), which translates to "Book". These Kanji Labels were selected due to their simplicity to write and

simplicity to read and translate, especially for this developing research project being conducted.
The datasets used are linked to this GitHub Repository.

5 Methodology

Machine Learning for Japanese Kanji Detection aims to utilize the power of Machine Learning to isolate Kanji from any image data, E.g Handwritten Kanji, printed image Kanji or Kanji from documents. This can be achieved by delving into the concept of Machine Learning.

5.1 Machine Learning for Japanese Kanji Detection

Machine learning is a sub field of artificial intelligence, which is broadly defined as the capability of a machine to imitate intelligent human behavior. Artificial intelligence systems are used to perform complex tasks in a way that is similar to how humans solve problems.[14]
It is subdivided into 2: Supervised Learning and Unsupervised Learning.

5.1.1 Supervised machine learning

Supervised machine learning builds a model that makes predictions based on evidence in the presence of uncertainty. A supervised learning algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions for the response to new data. [7]

Supervised learning use classification and regression techniques to develop and train machine learning models. Classification techniques are used for predicting discrete responses. i.e categorizing tasks into true or false. Classification models classify input data into categories. Typical applications include medical imaging, speech recognition, and credit scoring, while Regression techniques predict continuous responses i.e measuring quantities of matter. [7]
Supervised Learning is only suitable for training on labelled data. Labelled data is dataset that has its defined attributed for the ML model to learn from and make corrections during the training process. It provides information and context to the originally ambiguous training set. Without Labelled data, the supervised model will have little understanding of what it is supposed to learn from the data.
In supervised learning, we have a dataset consisting of input-output pairs (x_i, y_i) for $i = 1, \ldots, N$, where N is the number of data points. The goal is to learn a function f from the input space to the output space such that $f(x_i) \approx y_i$ for all i.

Given a training set

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \tag{6}$$

where $x_i \in \mathcal{X}$ (input space) and $y_i \in \mathcal{Y}$ (output space), the goal is to find a function $f : \mathcal{X} \to \mathcal{Y}$ that minimizes the empirical risk:

$$R(f) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)) \tag{7}$$

where $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ is a loss function that measures the discrepancy between the true output y_i and the predicted output $f(x_i)$.

5.1.2 Unsupervised learning

Unsupervised learning determines hidden patterns and intrinsic structures found in data. It is used to map inferences from datasets consisting of input data without labeled responses.
Clustering is the most common unsupervised learning technique. It is used for exploratory data analysis to find hidden patterns or groupings in data. Applications for cluster analysis include gene sequence analysis, market research, and object recognition.[7]

Given a set of data points $\{x_1, \ldots, x_N\}$ where $x_i \in \mathcal{X}$, the goal is to partition the dataset into K clusters such that data points in the same cluster are more similar to each other than to those in other clusters. One common approach, the k-means algorithm, aims to minimize the following objective:

$$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2 \tag{8}$$

where C_k represents the k-th cluster and μ_k is the centroid of that cluster.
After reviewing the functions and uses of Suoervised Machine Learning and Unsupervised Machine Learning, it is deduced that unsupervised ML is not the suitable training method to be carried out for this research because an Unsupervised Model cannot use clustering techniques to process and classify the extracted Kanji characters with dependable accuracy since there is no labelling of data to aid learning.
The methodology employed for this research includes the utilization the Supervised Machine Learning technique namely Convolutional Neural Networks (CNN) because it is the best method to use for image classification due to its complexity.
The Deep Learning technique was selected due to the fact it pertains towards the research specifications.

5.2 CNN Model Architecture

Convolutional Neural Networks (CNNs) are a subset of deep learning that is primarily utilized for image processing. Their architecture is inspired by the human

visual cortex, where individual neurons respond to stimuli only in restricted regions of the visual field.[22]
The architecture of the CNN in regards to the research involvees the utilization of Convolutional Layers, which is utilized for feature detection across the input data. In the data given to the model, the Convolutional layer will detect the stroke pattern and texture then store into the feature map, a kernel is utilized for this. This can be calculated as

$$(IF)(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j)F(x-i, y-j) \tag{9}$$

Where:

$(IF)(x,y)$: This is the result of the convolution operation between I and F at the position (x,y).

i and j: These are the variables of summation. For 2D signals or images, they iterate over the rows and columns, respectively.

$I(i,j)$: This denotes the value of the function (or signal or image) I at the position (i,j).

$F(x-i, y-j)$: Represents the value of the function F offset by i in the x-direction and j in the y-direction.

$\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty}$: The bounds of $-\infty$ to ∞ indicate an infinite summation. However, for discrete signals like images, the summation is typically over the dimensions of the image or the region of interest.

Both I and F are finite matrices, so the bounds of the summation are restricted to their dimensions. The mathematical annotation can be represented below in this Matrix Filter Figure 14:

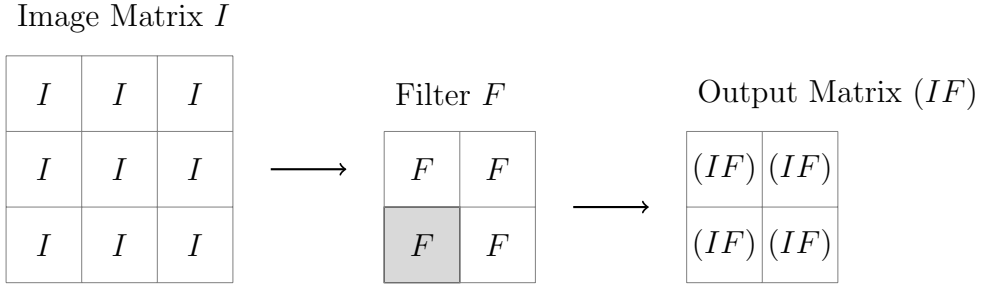

Figure 14: CNN Matrix

After convolution, the resultant feature maps go through an activation function, which is usually the Rectified Linear Unit (ReLU). RELU *"is an activation function that introduces the property of non-linearity to a deep learning model and solves the vanishing gradients issue".* [23]

$$f(x) = \max(0, x) \tag{10}$$

Pooling layers are then introduced to reduce the spatial dimensions of the data, using max pooling operation.

$$\text{max\_pool}(M) = \max(M_{i:i+1,j:j+1}) \tag{11}$$

Where i and j iterate over the rows and columns of M in steps determined by the pool size.
Then the Fully Connected Layer connect every neuron in one layer to every neuron in the next layer, which is placed at the end of the network to produce the final outputs.

5.2.1 Training

CNNs are trained using back propagation which involves adjusting their weights to minimize the difference between the predicted output and the actual target values. This involves iterative optimization techniques such as gradient descent.

- **Optimizer:** The Adam Moment Optimization (Adam optimizer), which is a subset of stochastic gradient descent approach that involves the use of AdaGrad and RMSProp, was utilized for its adaptive learning rate capability, ensuring faster convergence.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{12}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{13}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{14}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{15}$$

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{16}$$

Where g_t is the gradient at time step t, m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the

gradients respectively, β_1 and β_2 are exponential decay rates for these moment estimates, α is the learning rate, and ϵ is a small scalar to prevent division by zero.

- **Loss Function:** Categorical cross-entropy loss was utilized, given the fact there are 3 classes in use.

$$L(y, \hat{y}) = -\sum_{c=1}^{C} y_c \log(\hat{y_c}) \tag{17}$$

Where y_c is the true label (1 if the sample belongs to class c and 0 otherwise) and $\hat{y_c}$ is the predicted probability of the sample belonging to class c.

- **Metrics:** Accuracy was the primary metric monitored during training to assess the model's performance. The accuracy is calculated as

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{18}$$

- **Epochs and Batch Size:** Training was conducted for 100 epochs to ensure adequate learning from the training data while processing a batch of images in each iteration.

5.2.2 Evaluation and Testing

The Model's Learning was tested using test data that was not utilized during the learning process to assess the model's performance. This procedure is carried out to determine how well the model works for unseen data. This determines if the CNN model can be tried and tested for real world use.

The Figures 15, 16 & 17 below visualizes the convolutional neural network architecture, The architecture with their feature dimensions and also visualises the filters in the first convolutional layer of the model.

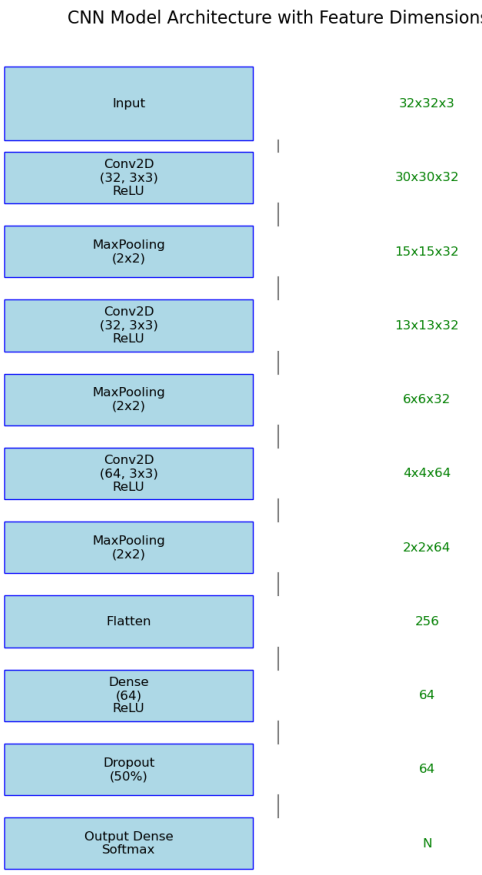

Figure 15: Image of CNN model and feature Dimensions

Figure 15 shows the input image starts as a 32x32x3 matrix (corresponding to a 32x32 pixel image with 3 color channels: RGB). As the input data goes through convolutional layers, the spatial dimensions (height and width) decrease, due to the convolutions (especially when no padding is used) and max pooling operations. The depth increases with more convolutional filters (4x4x64). The flatten layer reshapes the 3D feature matrix into a 1D vector, which is then fed into the dense layers. The dropout layer doesn't change the feature dimensions; it merely sets some neuron activations to zero to prevent overfitting. The final layer's output will have a dimension of N, where N is the number of classes the model is trained to classify. Figure 16 visualises a grid of plots, each representing a different feature map from the first convolutional layer. These feature maps highlight different aspects of the input image that the filters are responding to This means some filters respond to edges of the input Kanji while other filters respond to texture of the input Kanji, or to the specific shapes or stroke patterns of the kanji.
Where:

- **Input Layers:** (I_1, I_2, I_3)
- **Conv2D Layer Nodes:** $(C1_1, C1_2, C1_3, C1_4)$

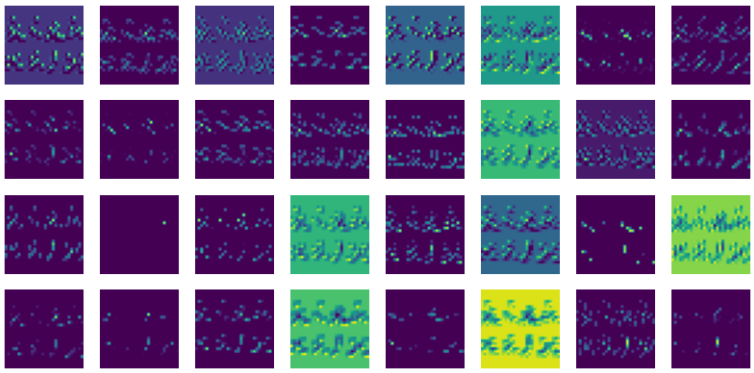

Figure 16: Image of first CNN filter

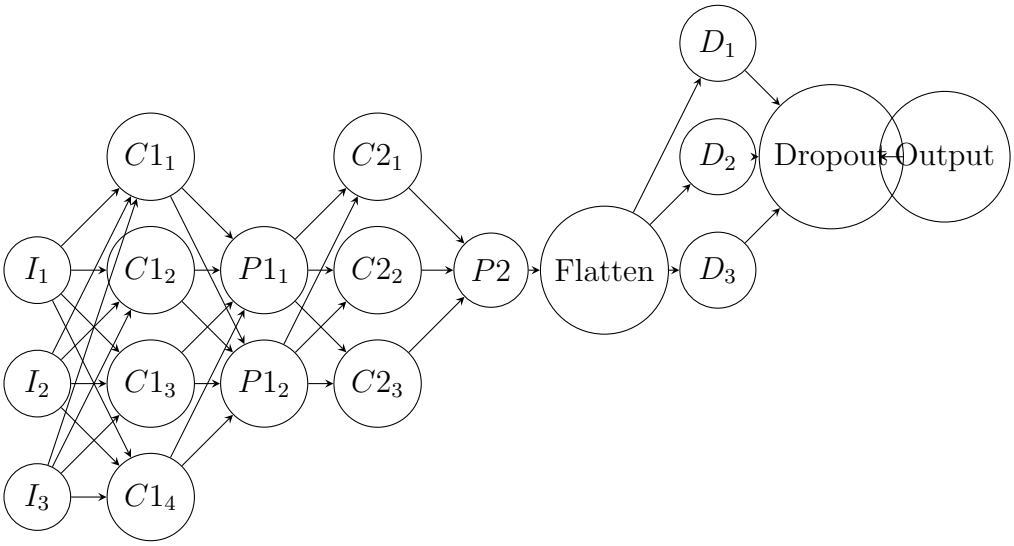

Figure 17: CNN Model Architecture

- **MaxPooling Layer Nodes:** ($P1_1$, $P1_2$)
- **Second Conv2D Layer Nodes:** ($C2_1$, $C2_2$, $C2_3$)
- **Second MaxPooling Layer Node:** ($P2$)
- **Flatten Node:** It signifies the flattening operation that reshapes the 3D feature map into a 1D vector, preparing it for the subsequent dense layers.
- **Dense Layer Nodes:** (D_1, D_2, D_3)
- **Dropout Node:** A regularization layer that intermittently sets a fraction of the neuron activations to zero during training to mitigate over fitting.
- **Output Node:** Represents the final output layer of the CNN.

5.3 Hyperparameter Optimization.

The use of Hyperparameter optimization in Machine Learning is aimed to achieve the best possible outcome and most accurate performance on the validation data.

The Configuration is outside the Convolutional Neural Network model and its value can not be determined from data.[32] Given two points x and x', the kernel function $k(x, x')$ defines the covariance between the function values at these points as follows:

$$\mathrm{Cov}(f(x), f(x')) = k(x, x') \tag{19}$$

There are four major methods used to perform Hyperparameter Optimization, namely:

1. Manual Search
2. Random Search
3. Grid Search
4. Bayesian Optimization

Bayesian Optimization was the technique utilized as the Hyperparameter Optimization. This can be defined as probabilistic model-based approach for finding the minimum of a function that is expensive to evaluate. [33] [34] Bayesian Optimization involves creating an external model that approximates the function that requires minimization. T his step can be carried out using the Gaussian process(GP) which defines a prior over functions and can be updated to a posterior given data. [35][36] Given two points x and x', the kernel function $k(x, x')$ defines the covariance between the function values at these points as follows:

$$\text{Cov}(f(x), f(x')) = k(x, x') \tag{20}$$

This covariance is a measure of the similarity between the function values at x and x' as modeled by the Gaussian process.
Then the GP utilizes an acquisition function to determine the next sample[37], which can be denoted as:

$$EI(x) = \mathbb{E}[\max(0, f_{\min} - f(x))] \tag{21}$$

After observing the data pair $(x, f(x))$, the Gaussian Process (GP) posterior is updated. The new posterior mean $\mu(x)$ and covariance $k(x, x')$ are computed using the kernel function and the observed data, as follows:

$$\mu(x) = \text{updated mean based on observations,}$$
$$k(x, x') = \text{updated covariance function.}$$

This finetuning procedure is ultimately done to improve the CNN model.

5.4 Advantage of CNN

Convolutional Neural Networks are one of the best supervised Learning models that can be used when working with image data.
CNNs basically do not require manual feature engineering because they learn features though surplus iterations. The model's technicality makes it very useful for surplus data, whilw still producing high quality results. [38]

6 Results

In the research project "Machine Learning for Japanese Kanji Detection", a Convolutional Neural Network (CNN) model was proposed to accurately detect Japanese Kanji characters from images and subsequently provide their English translations. The model architecture was designed to efficiently process and recognize the intricate Kanji patterns. The hyperparameters were optimized using Bayesian Optimization, The model consists of multiple convolutional layers with batch normalization, followed by max-pooling layers to reduce the spatial dimensions. Dropouts were strategically added after each pooling layer to prevent overfitting.
After extracting the features from the convolutional layers, the output was flattened and passed through the dense layers with batch normalization and dropout for further regularization. The final layer utilizes the softmax activation function to predict the probability distribution over the different Kanji classes.
The model is a very enhanced ML algorithm which can be determined as overkill, due to the limited data the CNN model's training and validation set.
The model was trained on 100 epochs on the extracted labeled Kanji characters. HyperParameter OPtimization proved to be overkill as training terminated at 17/100 epochs and 23/50 epochs. This is attributed to the minimal data used for the CNN Model. Fallback to the model without the added hyperparameters was utilized to identify the Kanji. This was done over 100 epochs with 85.71% accuracy.

6.1 Training Results:

The model was trained over 100 epochs on labeled Kanji images. The training exhibited a consistent improvement in accuracy over the epochs, and by the end of the 100th epoch, the model achieved a near-perfect training accuracy of 100%. The validation accuracy, an indicator of the model's generalization capability, reached a commendable 85.71%. Given the complexity of Kanji characters and the limited data provided, this validation accuracy underscores the model's robustness.

```
Epoch 92/100
2/2 [==============================] - 0s 25ms/step - loss: 1.4735e-04 - accuracy: 1.0000 - val_loss: 0.2946 - val_accuracy: 0.8571
Epoch 93/100
2/2 [==============================] - 0s 26ms/step - loss: 9.4545e-05 - accuracy: 1.0000 - val_loss: 0.3145 - val_accuracy: 0.8571
Epoch 94/100
2/2 [==============================] - 0s 24ms/step - loss: 2.1606e-04 - accuracy: 1.0000 - val_loss: 0.3330 - val_accuracy: 0.8571
Epoch 95/100
2/2 [==============================] - 0s 31ms/step - loss: 2.1590e-04 - accuracy: 1.0000 - val_loss: 0.3486 - val_accuracy: 0.8571
Epoch 96/100
2/2 [==============================] - 0s 31ms/step - loss: 5.5380e-04 - accuracy: 1.0000 - val_loss: 0.3730 - val_accuracy: 0.8571
Epoch 97/100
2/2 [==============================] - 0s 31ms/step - loss: 2.5339e-04 - accuracy: 1.0000 - val_loss: 0.3985 - val_accuracy: 0.8571
Epoch 98/100
2/2 [==============================] - 0s 25ms/step - loss: 4.9896e-05 - accuracy: 1.0000 - val_loss: 0.4175 - val_accuracy: 0.8571
Epoch 99/100
2/2 [==============================] - 0s 26ms/step - loss: 1.2895e-04 - accuracy: 1.0000 - val_loss: 0.4364 - val_accuracy: 0.8571
Epoch 100/100
2/2 [==============================] - 0s 26ms/step - loss: 2.7256e-04 - accuracy: 1.0000 - val_loss: 0.4588 - val_accuracy: 0.8571
```

```
final_val_accuracy = history.history['val_accuracy'][-1]
print(f"Final validation accuracy: {final_val_accuracy * 100:.2f}%")
```

```
Final validation accuracy: 85.71%
```

Figure 18: Image of CNN Model Accuracy

6.2 Model Evaluation on New Data:

To further assess the model's performance, new, unseen Kanji images are used as test data on the model to evaluate its training, which is without finetuned hyperparameters. The model displayed satisfactory results, correctly predicting the Kanji character (火) from both `resized_image` and `closing_image`, which translates to "Fire" in English, with 100% accuracy. Additionally, when presented with the image `063.gif`, the model accurately identified the Kanji character as (土), which translates to "Earth" in English.

New Test data was given to the model so it can detect the Kanji Character, Process it and provide its English Translation. The Test data image is Provided below:

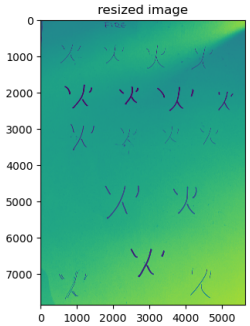

Figure 19: First Test data for Model Translation

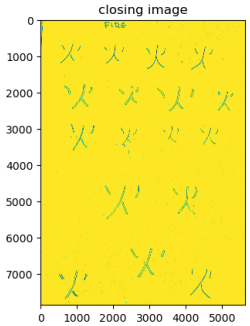

Figure 20: Second Test data for Model Translation

The extracted Kanji Characters from Figure 19 & 20

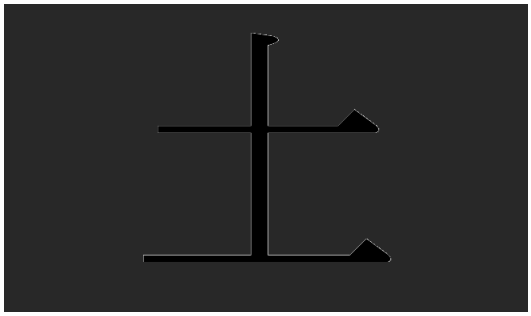

Figure 21: Third Test data for Model Translation

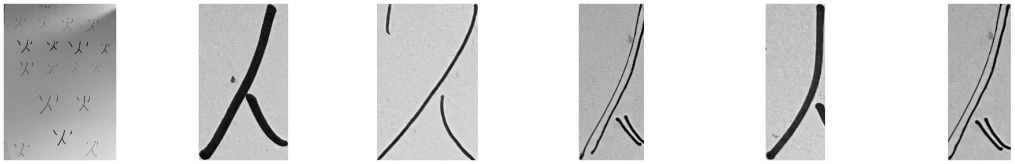

Figure 22: Extracted Test 1

The result of the test validation is shown below:
From Figure 24, it can be determined the CNN model has trained well enough to the initial test data given to it. This is determined due to the ambiguous nature of the extracted Kanji Characters from both Resized_Image and Closing_Image. The extracted image is not the full, complete Kanji, unlike the image extracts used in the training data, however, the CNN Model has adapted to the the extracted data and can recognize the stroke pattern of that Kanji label, which is one of the aims of the research, to accurately detect Kanji.

Figure 23: Extracted Test 2

```
closing_predicted_translation = label_to_translation[closing_predicted_label]

print(f"Prediction for resized_image: {resized_predicted_label} (English: {resized_predicted_translation})")
print(f"Prediction for closing_image: {closing_predicted_label} (English: {closing_predicted_translation})")
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 10ms/step
Prediction for resized_image: 火 (English: Fire)
Prediction for closing_image: 火 (English: Fire)
```

```
[503]: # Given true labels and predicted labels
true_labels = ["火", "火"]
predicted_labels = [class_labels[resized_label], class_labels[closing_label]]

correct_predictions = sum([true == pred for true, pred in zip(true_labels, predicted_labels)])
accuracy = correct_predictions / len(true_labels) * 100

print(f"Accuracy: {accuracy:.2f}%")
```

```
Accuracy: 100.00%
```

Figure 24: Image Result of Translated Test Data

7 Discussion

The results visualises the accuracy of the Convolutional Neural Network model for Japanese Kanji detection. Achieving an Accuracy of 85.71% indicates the model has trained well on the training data, Which composes of 51 extracted images, 3 Classes for Fire, Earth and Book which have 18, 20 and 13 image data respectively, but there is still more improvement that can happen to the model. However, achieving a 100% accuracy on unseen test data demonstrates the model's capability to generalize well beyond the training dataset, but this can also be attributed to the small sample size the model trained and tested with. The hyperparameter optimization performed on the Model was another indication that the model might be slightly iadvanced for the data being used and simpler models such as Support Vector Machines (SVM) or K-Nearest Neighbors Algorithm (k-NN) will have interesting metrics.
Generally CNN is used for training very large repositories and training just 51 extracted images can lead to overfitting on the data by the model.
Model robustness can be achieved if a larger sample size is used to train the model, which means more written kanji, by different people under different lighting conditions such as capturing the handwritten Kanji in a dim room, or different physical conditions, which can include the paper which the Kanji is being written on might not be plain but have horizontal lines and margins or the writers Write the Kanji Character with their less dominant hand. These conditions will create adversity with the data the model's performance on it will determine if the Model is truly robust and has taken to the training adequately. More data should have been utilized for this research but limitations were encountered with labelling data from larger repos-

itories like ETL9G data, this data will need to be labelled with the Kanji Character and its Translation before any machine Learning can be done on it. This task is very time consuming and extensive for a single researcher.
Another Limitation encountered were certain Python Packages such as keras proved difficult to utilize in the Virtual Environment due to there being constant mismatches between Keras and Sklearn, which inhibited the desired approach to utilize GridSearchCV for Cross Validation on the data.
However, the success in translating detected Kanji to English provides a practical application of the model, bridging the language gap and aiding in real-world scenarios like translation services, educational tools, and more.

8 Conclusion

Japanese Kanji is an intricate language Character which dates back all the way to the 5th Century from Chinese Language characters. The character intricacy makes it difficult for native speakers to accurately read and write Japanese Kanji and most optical read in tools cannot fully process Kanji after morphing from Image to Text with sustainable Accuracy. The research projects aims to be a stepping stone to achieve this.
Extensive research was carried out on previous papers and documents that have similar objectives to this research paper, this was done to tap into the vast knowledge of previous scholars to help bring this project to life. Convolutional Neural Network was the Machine Learning Technique used to develop this project on the extracted,m hand written image data, which albeit limited due to the sample size, satisfactory accuracy levels were drawn from the results of the model which indicates this project can be the first phase of achieving the objectives fully.
Limitations were encountered, due to the small sample size in data and it is noted that a different ML model can be used on the training data to determine if those models, such as SVM and K-NN and drawbacks gotten from virtual machine environment packaging mismatches. In conclusion, the proposed CNN model has proven to be a reliable tool for Japanese Kanji detection and translation, with promising results and model robustness that can be further enhanced with a more extensive, labelled dataset and a working Hypereparamater Optimization for model refinements.

9 Future Work

Future work on this research is required, at this current stage this is a complex model working on simple data. For the model to yield better accuracy, more data will be required and each data will require labelling and English translation for each individual character. This research will also run smoothly with more manpower, as

labelling over 10,000 images single handily is time consuming and strenuous. More Model types can be created and tested, like the utilization of Recurrent Neural Networks (RNNs) or Transformers and comparing to the CNN model. This research can also be deployed into a web application which will give the model a more User Friendly, User Experience.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio, "Published as a conference paper at ICLR 2015 NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE," 2015.

[2] C. Tsai, "Recognizing Handwritten Japanese Characters Using Deep Convolutional Neural Networks" [Online]. Available: http://cs231n.stanford.edu/reports/2016/pdfs/262_Report.pdf

[3] What Is Optical Character Recognition (OCR)? by IBM, 5 Jan. 2022, www.ibm.com/cloud/blog/optical-character-recognition.

[4] K. Yasar, "What Is an Artificial Neural Network (ANN)?," SearchEnterpriseAI, 2023. https://www.techtarget.com/searchenterpriseai/definition/neural-network

[5] Shai Shalev-Shwartz and Shai Ben-David, Understanding machine learning : from theory to algorithms. Delhi: Cambridge University Press, 2015.

[6] [21]D. Allen and K. Conklin, "Cross-linguistic similarity norms for Japanese–English translation equivalents," Behavior Research Methods, vol. 46, no. 2, pp. 540–563, Sep. 2013, doi: https://doi.org/10.3758/s13428-013-0389-z.

[7] "What Is Machine Learning? — How It Works, Techniques & Applications," Mathworks.com, 2020. Available: https://uk.mathworks.com/discovery/machine-learning.html

[8] I. Awokoya, "Research Proposal for Japanese Kanji Detection," 2023.

[9] E. Burns, "What is Artificial Intelligence (AI)? - AI Definition and How it Works," SearchEnterpriseAI, Feb. 2022. Available: https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence#: :text=Artificial -intelligence-20is-the-simulation

[10] https://gurucul.com/blog/what-is-machine-learning#: :text=The-definition-of-machine-learning,a-subset-artificial-intelligence.

[11] F. Akkoyun, A. Erçetin, S. Güçlüer, A. Ozcelik, I. Arpaci, and S. Gucluer, "A Multi-Flow Production Line for Sorting of Eggs Using Image Processing Characterization and micro machinability of Mg alloys produced through powder metallurgy method View project Journal of Characterization View project A Multi-Flow Production Line for Sorting of Eggs Using Image Processing," 2022, doi: https://doi.org/10.3390/s23010117.

[12] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, Massachusetts: The Mit Press, 2016.

[13] http://plus.google.com/+riizhushiteru, "Japanese Language Characters (Kanji, Hiragana and Katakana)," Learn Japanese Language Online. [Online]. Available: https://japaneselearningonline.blogspot.com/2015/03/japanese-languagecharacters-kanji.html

[14] S. Brown, "Machine learning, explained," MIT Sloan, Apr. 21, 2021. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained#: :text=Machine-20learning-is-a-subfield-of-artificial-intelligence

[15] "The Difficulties of Kanji— 'Kanji Art' Your name in Japanese kanji," kanjiart.net. [Online]. Available: https://kanjiart.net/column/the-difficulties-of-kanji/. [Accessed: May 04, 2023]

[16] IBM, "What Is Deep Learning? — IBM," www.ibm.com, 2023. httPs://www.ibm.com/topics/deep-learning#: :text=Deep learning

[17] Stanford University, "NLP - Overview," cs.stanford.edu. https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/nlp/overview_history.html#: :text=NLP-D-overview&text=The-field-of-natural-language

[18] "Japanese - Worldwide Distribution," Worlddata.info. https://www.worlddata.info/languages/japanese.php#: :text=The-Japanese-language-(native-name)

[19] A. Katakami, "The Difficulties of Kanji — 'Kanji Art' Your name in Japanese kanji," kanjiart.net, Dec. 18, 2020. https://kanjiart.net/column/the-difficulties-of-kanji#: :text=Another-reason-why-people-may (accessed Oct. 30, 2023).

[20] Python Software Foundation, "opencv-python," PyPI, Nov. 21, 2019. https://pypi.org/project/opencv-python/

[21] Y. Diez, T. Suzuki, M. Vila, and K. Waki, "Computer Vision and Deep Learning Tools for the Automatic Processing of Wasan Documents," Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods, 2019, doi: https://doi.org/10.5220/0007555607570765.

[22] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way — Saturn Cloud Blog," saturncloud.io, Dec. 15, 2018. https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

[23] B. Krishnamurty, "ReLU Activation Function Explained — Built in," builtin.com, Oct. 28, 2022. https://builtin.com/machine-learning/relu-activation-function

[24] J. Brownlee, "What Is Deep Learning?," Machine Learning Mastery, Sep. 22, 2016. https://machinelearningmastery.com/what-is-deep-learning/

[25] Deepgram, "What Is Tokenization in NLP & Machine Learning?," Deepgram, Sep. 29, 2023. https://deepgram.com/ai-glossary/tokenization (accessed Nov. 01, 2023).

[26] J. Nabi, "Recurrent Neural Networks (RNNs)," Medium, Jul. 21, 2019. https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85

[27] I. Neranjana, "Transformers and the Future of AI: How This Architecture is Shaping the Next Generation of...," Medium, May 08, 2023. https://medium.com/@ishараner96/transformers-and-the-future-of-ai-how-this-architecture-is-shaping-the-next-generation-of-fdea0d5fc71c (accessed Nov. 03, 2023).

[28] C.-L. Goh, Masayuki Asahara, and Y. Matsumoto, "Building a Japanese-Chinese Dictionary Using Kanji/Hanzi Conversion," Lecture Notes in Computer Science, vol. 3651, pp. 670–681, Jan. 2005, doi: https://doi.org/10.1007/11562214_59.

[29] Saroha, "50+ Key Terms/ Topics in Deep Learning [Complete DL revision]," OpenGenus IQ: Computing Expertise & Legacy, Feb. 26, 2023. https://iq.opengenus.org/key-terms-in-deep-learning/

[30] JPEG, "JPEG - about JPEG," jpeg.org, 2023. https://jpeg.org/about.html

[31] OpenCV, "OpenCV: Image Thresholding," docs.opencv.org, Nov. 2023. https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

[32] J. Brownlee, "What is the Difference Between a Parameter and a Hyperparameter?," Machine Learning Mastery, Jul. 25, 2017. https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/

[33] M. Diessner et al., "Investigating Bayesian optimization for expensive-to-evaluate black box functions: Application in fluid dynamics," Frontiers in Applied Mathematics and Statistics, vol. 8, no. 1076296, Dec. 2022, doi: https://doi.org/10.3389/fams.2022.1076296.

[34] H. Rawlani, "Hyperparameter tuning with Keras and Ray Tune," Medium, Oct. 12, 2020. https://towardsdatascience.com/hyperparameter-tuning-with-keras-and-ray-tune-1353e6586fda

[35] J. Brownlee, "How to Implement Bayesian Optimization from Scratch in Python," Machine Learning Mastery, Oct. 08, 2019. https://machinelearningmastery.com/what-is-bayesian-optimization/

[36] P. Frazier, "A Tutorial on Bayesian Optimization," 2018. Available: https://arxiv.org/pdf/1807.02811.pdf

[37] M. Krasser, "Bayesian Optimization - Martin Krasser's Blog," krasserm.github.io, Mar. 21, 2018. http://krasserm.github.io/2018/03/21/bayesian-optimization/

[38] Datagen, "Convolutional Neural Network: Benefits, Types, and Applications," Datagen, 2022. https://datagen.tech/guides/computer-vision/cnn-convolutional-neural-network/

10 APPENDIX

```
#!/usr/bin/env python
# coding: utf-8

# In[138]:


cd downloads


# # Input Packages

# In[ ]:


from PIL import Image
import matplotlib.pyplot as plt
import cv2
import numpy as np
import os
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Activation, Dropout, Flatten,
    Dense
from sklearn.model_selection import train_test_split
import matplotlib.patches as patches
import tensorflow as tf


# # Dataset 1

# In[6]:


# Load and display the provided image
img_path = "exampledata7.jpg"
img = Image.open(img_path)

# Display the image
plt.imshow(img)
plt.axis('on')  # Display axes for clarity
plt.title("Uploaded Image")
plt.show()


# # Bounding box of data
```

```
47
48 # In[7]:
49
50
51 # Since the image is already in grayscale, we can proceed with
       thresholding
52 _, thresh = cv2.threshold(np.array(img), 0, 255, cv2.
       THRESH_BINARY_INV + cv2.THRESH_OTSU)
53
54 # Find contours in the thresholded image
55 contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.
       CHAIN_APPROX_SIMPLE)
56
57 # Filter out small contours based on area to avoid noise
58 filtered_contours = [cnt for cnt in contours if cv2.contourArea(cnt
       ) > 100]
59
60 # Extract bounding boxes from the contours
61 bboxes = [cv2.boundingRect(cnt) for cnt in filtered_contours]
62
63 # Draw the bounding boxes on the original image
64 img_with_bboxes = np.array(img).copy()
65 for x, y, w, h in bboxes:
66     cv2.rectangle(img_with_bboxes, (x, y), (x + w, y + h), (255, 0,
       0), 2)
67
68 # Display the image with bounding boxes
69 plt.imshow(img_with_bboxes, cmap='gray')
70 plt.axis('on')
71 plt.title("Bounding Boxes on Image")
72 plt.show()
73
74 bboxes
75
76
77 # # Ectracted image from data
78
79 # In[27]:
80
81
82 # Extract individual character images based on the bounding boxes
83 cropped_images = []
84 for x, y, w, h in bboxes:
85     cropped = img.crop((x, y, x + w, y + h))
86     cropped_images.append(cropped)
87
88 # Display the cropped images
89 fig, axes = plt.subplots(1, len(cropped_images), figsize=(20, 3))
90 for ax, cropped_img in zip(axes, cropped_images):
```

```
    ax.imshow(cropped_img, cmap='gray')
    ax.axis('off')

plt.tight_layout()
plt.show()


# In[13]:


# Sort bounding boxes based on area
sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3],
    reverse=True)

# Set a threshold for the number of Kanji you expect to extract
num_kanji = 8

# Extract the bounding boxes with the largest areas up to the
    threshold
largest_bboxes = sorted_bboxes[:num_kanji]

cropped_images = []
for x, y, w, h in largest_bboxes:
    cropped = img.crop((x, y, x + w, y + h))
    cropped_images.append(cropped)

# Display the cropped images
fig, axes = plt.subplots(1, len(cropped_images), figsize=(20, 3))
for ax, cropped_img in zip(axes, cropped_images):
    ax.imshow(cropped_img, cmap='gray')
    ax.axis('off')

plt.tight_layout()
plt.show()


# In[14]:


# Sort bounding boxes based on area
sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3],
    reverse=True)

# Set a threshold for the number of Kanji you expect to extract
num_kanji = 8

# Extract the bounding boxes with the largest areas up to the
    threshold
largest_bboxes = sorted_bboxes[:num_kanji]
```

```
136
137 cropped_images = []
138 for x, y, w, h in largest_bboxes:
139     cropped = img.crop((x, y, x + w, y + h))
140     cropped_images.append(cropped)
141
142
143 # In[15]:
144
145
146 # Directory to save the cropped images
147 save_dir = "/Users/macbookpro/Downloads"
148 os.makedirs(save_dir, exist_ok=True)
149
150 # Save each cropped image to the directory
151 saved_paths = []
152 for idx, cropped_img in enumerate(cropped_images, start=1):
153     save_path = os.path.join(save_dir, f"character_{idx}.jpg")
154     cropped_img.save(save_path)
155     saved_paths.append(save_path)
156
157 saved_paths
158
159
160 # In[10]:
161
162
163 # Extract individual character images based on the bounding boxes
164 #cropped_images = []
165 #for x, y, w, h in bboxes:
166 #    cropped = img.crop((x, y, x + w, y + h))
167 #    cropped_images.append(cropped)
168
169
170 # In[11]:
171
172
173 #import os
174
175 # Directory to save the cropped images
176 #save_dir = "/Users/macbookpro/Downloads"
177 #os.makedirs(save_dir, exist_ok=True)
178
179 # Save each cropped image to the directory
180 #saved_paths = []
181 #for idx, cropped_img in enumerate(cropped_images, start=1):
182 #    save_path = os.path.join(save_dir, f"character_{idx}.jpg")
183 #    cropped_img.save(save_path)
184 #    saved_paths.append(save_path)
```

```
185
186 #saved_paths
187
188
189 # In[21]:
190
191
192 # Create a DataFrame
193 data = {
194     'Character Name': [os.path.basename(path) for path in
    saved_paths],  # Extract the filename from the path
195     'Label': ['   '] * len(saved_paths),  # The label for all is

196     'English Translation': ['fire'] * len(saved_paths)  # The
    translation for all is fire
197 }
198
199 df = pd.DataFrame(data)
200
201 # Save the DataFrame to a CSV file
202 csv_path = os.path.join(save_dir, "newlabels.csv")
203 df.to_csv(csv_path, index=False)
204
205
206 # # Importing CSV file for the labelling
207
208 # In[22]:
209
210
211 import pandas as pd
212
213 # Load the labels from the CSV file
214 labels_df = pd.read_csv('newlabels.csv')
215 labels_df
216
217
218 # In[23]:
219
220
221 labels_df.describe()
222
223
224 # # CNN MODEL
225
226 # In[24]:
227
228
229 # Load the labels from the CSV file
230 labels_df = pd.read_csv('newlabels.csv')
```

```
231
232 # Split the data into training and validation sets
233 train_df, val_df = train_test_split(labels_df, test_size=0.2,
    random_state=220074391)
234
235 # Data Augmentation
236 train_datagen = ImageDataGenerator(
237     rescale=1./255,
238     rotation_range=20,
239     width_shift_range=0.2,
240     height_shift_range=0.2,
241     shear_range=0.2,
242     zoom_range=0.2,
243     horizontal_flip=True,
244     fill_mode='nearest'
245 )
246
247 val_datagen = ImageDataGenerator(rescale=1./255)
248
249 # Data Generators
250 train_generator = train_datagen.flow_from_dataframe(
251     dataframe=train_df,
252     directory='/Users/macbookpro/Downloads/',
253     x_col="Character Name",  # Changed column name to "Character
    Name"
254     y_col="Label",           # Changed column name to "Label"
255     target_size=(32, 32),
256     class_mode="categorical",
257     batch_size=5
258 )
259
260 val_generator = val_datagen.flow_from_dataframe(
261     dataframe=val_df,
262     directory='/Users/macbookpro/Downloads/',
263     x_col="Character Name",  # Changed column name to "Character
    Name"
264     y_col="Label",           # Changed column name to "Label"
265     target_size=(32, 32),
266     class_mode="categorical",
267     batch_size=5
268 )
269
270 # Define the CNN model
271 model = Sequential()
272 model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3)))
273 model.add(Activation('relu'))
274 model.add(MaxPooling2D(pool_size=(2, 2)))
275
276 model.add(Conv2D(32, (3, 3)))
```

```
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_generator.class_indices)))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Train the model
model.fit(train_generator,
          epochs=50,
          validation_data=val_generator)

# Save the model
model.save('kanji_recognition_model.h5')


# In[25]:


loss, accuracy = model.evaluate(val_generator)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")


# In[26]:


import matplotlib.pyplot as plt
import matplotlib.patches as patches

def visualize_cnn_layers():
    # Define the figure
    fig, ax = plt.subplots(figsize=(12, 8))

    # Layer names and their descriptions
    layers = [
        "Input\n(32x32x3)",
        "Conv2D\n(32, 3x3)\nReLU",
        "MaxPooling\n(2x2)",
```

```
        "Conv2D\n(32, 3x3)\nReLU",
        "MaxPooling\n(2x2)",
        "Conv2D\n(64, 3x3)\nReLU",
        "MaxPooling\n(2x2)",
        "Flatten",
        "Dense\n(64)\nReLU",
        "Dropout\n(50%)",
        "Output Dense\nSoftmax"
    ]

    # Define the positions and sizes for each layer's rectangle
    positions = [(i, 0.5) for i in range(len(layers))]
    sizes = [(1.5, 1) for _ in range(len(layers))]
    sizes[0] = (1.5, 1.5)  # Input layer is slightly bigger

    # Create the rectangles and texts
    for (x, y), (width, height), layer in zip(positions, sizes,
    layers):
        rect = patches.Rectangle((x, y), width, height, linewidth
    =1, edgecolor='r', facecolor='none')
        ax.add_patch(rect)
        ax.text(x + width/2, y + height/2, layer, ha='center', va='
    center', fontsize=12)

    # Adjust the plot
    ax.set_xlim(0, len(layers) + 1)
    ax.set_ylim(0, 2.5)
    ax.axis("off")
    plt.title("CNN Model Architecture", fontsize=16)

    plt.show()

# Call the function to visualize the layers
visualize_cnn_layers()


# In[27]:


import matplotlib.pyplot as plt
import matplotlib.patches as patches

def visualize_cnn_features():
    # Define the figure
    fig, ax = plt.subplots(figsize=(10, 15))

    # Layer names and feature matrix dimensions
    layers = [
        "Input",
```

```
        "Conv2D\n(32, 3x3)\nReLU",
        "MaxPooling\n(2x2)",
        "Conv2D\n(32, 3x3)\nReLU",
        "MaxPooling\n(2x2)",
        "Conv2D\n(64, 3x3)\nReLU",
        "MaxPooling\n(2x2)",
        "Flatten",
        "Dense\n(64)\nReLU",
        "Dropout\n(50%)",
        "Output Dense\nSoftmax"
    ]
    feature_dims = [
        "32x32x3",
        "30x30x32",
        "15x15x32",
        "13x13x32",
        "6x6x32",
        "4x4x64",
        "2x2x64",
        "256",
        "64",
        "64",
        "N"
    ]

    # Define positions and sizes for each layer's rectangle
    layer_positions = [(1.5, i) for i in range(len(layers), 0, -1)]
    feature_positions = [(4.5, i) for i in range(len(layers), 0,
-1)]
    sizes = [(2.5, 0.7) for _ in layers]
    sizes[0] = (2.5, 1)  # Input layer is slightly bigger

    # Draw rectangles and text for layers and feature dimensions
    for (lx, ly), (fx, fy), (width, height), layer, feature_dim in
zip(layer_positions, feature_positions, sizes, layers,
feature_dims):
        rect = patches.Rectangle((lx - width/2, ly - height/2),
width, height, linewidth=1, edgecolor='b', facecolor='lightblue'
)
        ax.add_patch(rect)
        ax.text(lx, ly, layer, ha='center', va='center', fontsize
=12)
        ax.text(fx, fy, feature_dim, ha='center', va='center',
fontsize=12, color='green')

    # Add connection lines between layers
    for i in range(len(layers) - 1):
        plt.plot([3, 3], [layer_positions[i][1] - sizes[i][1]/2,
layer_positions[i+1][1] + sizes[i+1][1]/2], color='gray')
```

```
413
414     # Adjust the plot
415     ax.set_xlim(0, 6)
416     ax.set_ylim(0, len(layers) + 1)
417     ax.axis("off")
418     plt.title("CNN Model Architecture with Feature Dimensions",
        fontsize=16)
419
420     plt.show()
421
422 # Now you can call this function
423 visualize_cnn_features()
424
425
426 # In[548]:
427
428
429 import numpy as np
430 import matplotlib.pyplot as plt
431
432 # Generate a random 32x32x3 matrix (values between 0 and 1)
433 matrix = np.random.rand(32, 32, 3)
434
435 # Visualize the matrix as an image
436 plt.imshow(matrix)
437 plt.axis('off')  # To hide the axis
438 plt.title("Visualization of 32x32x3 Matrix")
439 plt.show()
440
441
442 # In[30]:
443
444
445 import tensorflow as tf
446
447
448 # # Testing Data
449
450 # In[31]:
451
452
453 from tensorflow.keras.preprocessing.image import load_img,
        img_to_array
454
455 def predict_image(image_path, model, train_generator):
456     # Load and preprocess the image
457     image = load_img(image_path, target_size=(32, 32), color_mode="
        grayscale")
458     image_array = img_to_array(image)
```

```
    image_array = np.repeat(image_array, 3, axis=-1)  # Replicate the grayscale channel 3 times to make it RGB-like
    image_array = image_array / 255.0  # Normalize
    image_array = np.expand_dims(image_array, axis=0)  # Add batch dimension

    # Predict using the model
    predictions = model.predict(image_array)
    predicted_class = np.argmax(predictions[0])

    # Get the class indices from the training generator
    label_map = train_generator.class_indices
    predicted_label = list(label_map.keys())[list(label_map.values()).index(predicted_class)]

    return predicted_label

# Load the trained model (assuming you have already trained and saved it)
model = tf.keras.models.load_model('kanji_recognition_model.h5')

# Predict for the uploaded images
resized_image_path = '/Users/macbookpro/Downloads/resized_image.jpg'
closing_image_path = '/Users/macbookpro/Downloads/closing_image.jpg'

predicted_label_resized = predict_image(resized_image_path, model, train_generator)
predicted_label_closing = predict_image(closing_image_path, model, train_generator)

print(f"Prediction for resized_image: {predicted_label_resized}")
print(f"Prediction for closing_image: {predicted_label_closing}")


# In[32]:


# True labels for the test images (assuming both are "   ")
true_labels = ["   ", "   "]

# Model predictions for the test images
predicted_labels = [predicted_label_resized, predicted_label_closing]

# Compute accuracy
correct_predictions = sum([true == pred for true, pred in zip(true_labels, predicted_labels)])
```

```
accuracy = correct_predictions / len(true_labels) * 100

print(f"Accuracy: {accuracy:.2f}%")


# # Improving the Model by adding more Data

# In[33]:


from PIL import Image
import matplotlib.pyplot as plt

# Load and display the provided image
img_path = "exampledata10.jpg"
img = Image.open(img_path)

# Display the image
plt.imshow(img)
plt.axis('on')  # Display axes for clarity
plt.title("Uploaded Image")
plt.show()


# In[37]:


import cv2
import numpy as np
from matplotlib import pyplot as plt

# Assuming 'img' is already defined and loaded
# Since the image is already in grayscale, we can proceed with thresholding
_, thresh = cv2.threshold(np.array(img), 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

# Find contours in the thresholded image
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Extract bounding boxes from the contours
bboxes = [cv2.boundingRect(cnt) for cnt in contours]

# Find the bounding box with the largest area (assuming it's the full Kanji character)
largest_bbox = max(bboxes, key=lambda bbox: bbox[2] * bbox[3])

# Draw the largest bounding box on the original image
```

```
543 img_with_bbox = np.array(img).copy()
544 x, y, w, h = largest_bbox
545 cv2.rectangle(img_with_bbox, (x, y), (x + w, y + h), (255, 0, 0),
    2)
546
547 # Display the image with bounding box
548 plt.imshow(img_with_bbox, cmap='gray')
549 plt.axis('on')
550 plt.title("Bounding Box on Image")
551 plt.show()
552
553 largest_bbox
554
555
556 # # visualizing the bounding box
557
558 # In[38]:
559
560
561 import cv2
562 import numpy as np
563 import matplotlib.pyplot as plt
564 from PIL import Image
565
566 def visualize_bounding_boxes(image_path):
567     # Load the image using PIL
568     img = Image.open(image_path)
569
570     # Convert the image to binary using Otsu's thresholding
571     _, thresh = cv2.threshold(np.array(img), 0, 255, cv2.
    THRESH_BINARY_INV + cv2.THRESH_OTSU)
572
573     # Find contours in the thresholded image
574     contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)
575
576     # Extract bounding boxes from the contours
577     bboxes = [cv2.boundingRect(cnt) for cnt in contours]
578
579     # Find the bounding box with the largest area (assuming it's
    the full Kanji character)
580     largest_bbox = max(bboxes, key=lambda bbox: bbox[2] * bbox[3])
581
582     # Draw the largest bounding box on the original image
583     img_with_bbox = np.array(img).copy()
584     x, y, w, h = largest_bbox
585     cv2.rectangle(img_with_bbox, (x, y), (x + w, y + h), (0, 255,
    0), 2)
586
```

```
    # Display the image with bounding box
    plt.imshow(img_with_bbox, cmap='gray')
    plt.axis('on')
    plt.title("Bounding Box on Image")
    plt.show()

    return largest_bbox

# Usage example:
image_path = 'exampledata10.jpg'
bounding_box = visualize_bounding_boxes(image_path)
print(bounding_box)


# In[525]:


import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

def visualize_bounding_boxes(image_path):
    # Load the image using PIL
    img = Image.open(image_path)

    # Convert the image to binary using Otsu's thresholding
    _, thresh = cv2.threshold(np.array(img), 0, 255, cv2.
    THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # Find contours in the thresholded image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)

    # Extract bounding boxes from the contours
    bboxes = [cv2.boundingRect(cnt) for cnt in contours]

    # Draw all the bounding boxes on the original image
    img_with_bbox = np.array(img).copy()
    for bbox in bboxes:
        x, y, w, h = bbox
        cv2.rectangle(img_with_bbox, (x, y), (x + w, y + h), (0,
    255, 0), 2)

    # Display the image with bounding boxes
    plt.imshow(img_with_bbox, cmap='gray')
    plt.axis('on')
    plt.title("Bounding Boxes on Image")
    plt.show()
```

```
633
634     return bboxes
635
636 # Usage example:
637 image_path = 'exampledata7.jpg'
638 bounding_boxes = visualize_bounding_boxes(image_path)
639 print(bounding_boxes)
640
641
642 # In[41]:
643
644
645 # Sort the bounding boxes based on their area (from largest to
    smallest)
646 sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3],
    reverse=True)
647
648 # Extract the top 'n' bounding boxes
649 num_kanji = 12  # Adjust this based on the number of Kanji
    characters you expect
650 selected_bboxes = sorted_bboxes[:num_kanji]
651
652 # Extract individual Kanji character images based on the selected
    bounding boxes
653 cropped_images = [img.crop((x, y, x + w, y + h)) for x, y, w, h in
    selected_bboxes]
654
655 # Display the extracted Kanji characters
656 fig, axes = plt.subplots(1, len(cropped_images), figsize=(20, 3))
657 for ax, cropped_img in zip(axes, cropped_images):
658     ax.imshow(cropped_img, cmap='gray')
659     ax.axis('off')
660
661 plt.tight_layout()
662 plt.show()
663
664
665 # In[53]:
666
667
668 import pandas as pd
669
670 # Sort the bounding boxes based on their area (from largest to
    smallest)
671 sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3],
    reverse=True)
672
673 # Number of Kanji characters you expect to extract
674 num_kanji = 12
```

```
675
676 # Extract the top 'n' Kanji character images based on the selected
    bounding boxes
677 selected_bboxes = sorted_bboxes[:num_kanji]
678 cropped_images = [img.crop((x, y, x + w, y + h)) for x, y, w, h in
    selected_bboxes]
679
680 # Directory to save the cropped images
681 save_directory = "/Users/macbookpro/Downloads"
682 os.makedirs(save_directory, exist_ok=True)
683
684 # Check the existing files in the directory to ensure unique
    filenames
685 existing_files = os.listdir(save_directory)
686 existing_counts = [int(f.split('_')[1].split('.')[0]) for f in
    existing_files if "character_" in f]
687 start_idx = max(existing_counts, default=-1) + 1
688
689 # List to store new entries for the CSV
690 new_entries = []
691
692 # Save each cropped image to the directory and add their info to
    new_entries
693 for idx, cropped_img in enumerate(cropped_images, start=start_idx):
694     filename = f"character_{idx}.jpg"
695     cropped_img.save(f"{save_directory}/{filename}")
696
697     # Append to new_entries
698     new_entries.append({"Character Name": filename, "Label": "    ",
    "English Translation": "Fire"})
699
700 # Load the existing CSV or create a new one if not present
701 if os.path.exists('newlabels.csv'):
702     df = pd.read_csv('newlabels.csv')
703 else:
704     df = pd.DataFrame(columns=["Character Name", "Label", "English
    Translation"])
705
706 # Append the new entries to the dataframe and save it back
707 df_new = pd.DataFrame(new_entries)
708 df = pd.concat([df, df_new], ignore_index=True)
709 df.to_csv('newlabels.csv', index=False)
710
711
712 # In[50]:
713
714
715 # Extract individual character images based on the bounding boxes
716 #cropped_images = []
```

```
#for x, y, w, h in bboxes:
#    cropped = img.crop((x, y, x + w, y + h))
#    cropped_images.append(cropped)


# In[51]:


#save_directory = "/Users/macbookpro/Downloads"
# Check the existing files in the directory to ensure unique filenames
#existing_files = os.listdir(save_directory)
#existing_counts = [int(f.split('_')[1].split('.')[0]) for f in existing_files if "character_" in f]
#start_idx = max(existing_counts, default=-1) + 1

# Save the cropped images with unique filenames
#for idx, cropped_img in enumerate(cropped_images, start=start_idx):
#    filename = f"character_{idx}.jpg"
#    cropped_img.save(f"{save_directory}/{filename}")


# In[537]:


df.head(10)


# In[530]:


import pandas as pd

# Load the CSV file into a DataFrame
df = pd.read_csv('newlabels.csv')

# Check for null values in the entire DataFrame
null_values = df.isnull()

# Print the number of null values for each column
print(null_values.sum())

# If you want to display rows containing any null value:
rows_with_nan = df[df.isnull().any(axis=1)]
print(rows_with_nan)


# In[541]:
```

```
763
764
765 df.info()
766
767
768 # # Model Retraining
769
770 # In[67]:
771
772
773 import tensorflow as tf
774 from tensorflow.keras.preprocessing.image import ImageDataGenerator
775
776 # Path to the directory containing the images
777 data_directory = "/Users/macbookpro/Downloads"
778
779 # Initialize the data generators
780 datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
        # Rescale and set aside 20% of data for validation
781
782 # Training data generator
783 train_generator = datagen.flow_from_dataframe(
784     dataframe=pd.read_csv('/Users/macbookpro/Downloads/newlabels.
    csv'),
785     directory=data_directory,
786     x_col="Character Name",
787     y_col="Label",
788     target_size=(32, 32),
789     color_mode="rgb",
790     class_mode="categorical",
791     batch_size=32,
792     subset="training"
793 )
794
795 # Validation data generator
796 validation_generator = datagen.flow_from_dataframe(
797     dataframe=pd.read_csv('/Users/macbookpro/Downloads/newlabels.
    csv'),
798     directory=data_directory,
799     x_col="Character Name",
800     y_col="Label",
801     target_size=(32, 32),
802     color_mode="rgb",
803     class_mode="categorical",
804     batch_size=32,
805     subset="validation"
806 )
807
808
```

```
# In[68]:


model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu',
    input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(len(train_generator.class_indices),
    activation='softmax')  # Number of classes
])

model.compile(optimizer='adam', loss='categorical_crossentropy',
    metrics=['accuracy'])


# In[69]:


history = model.fit(
    train_generator,
    epochs=50,
    validation_data=validation_generator
)


# In[70]:


model.save('Retrained_Kanji_Model.h5')

#loaded_model = tf.keras.models.load_model('
    updated_kanji_recognition_model.keras')


# In[71]:


loss, accuracy = model.evaluate(val_generator)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")


# In[ ]:


# Load the Model
```

```
# loaded_model = tf.keras.models.load_model('Retrained_Kanji_Model.h5')


# In[ ]:




# # Adding two (2) new Kanji to the Training Data

# In[72]:


from PIL import Image
import matplotlib.pyplot as plt

# Load and display the provided image
img_path = "exampledata20.jpg"
img = Image.open(img_path)

# Display the image
plt.imshow(img)
plt.axis('on')  # Display axes for clarity
plt.title("Uploaded Image")
plt.show()


# In[73]:


import cv2
import numpy as np
from matplotlib import pyplot as plt

# Assuming 'img' is already defined and loaded
# Since the image is already in grayscale, we can proceed with thresholding
_, thresh = cv2.threshold(np.array(img), 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

# Find contours in the thresholded image
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Extract bounding boxes from the contours
bboxes = [cv2.boundingRect(cnt) for cnt in contours]

```

```
# Find the bounding box with the largest area (assuming it's the full Kanji character)
largest_bbox = max(bboxes, key=lambda bbox: bbox[2] * bbox[3])

# Draw the largest bounding box on the original image
img_with_bbox = np.array(img).copy()
x, y, w, h = largest_bbox
cv2.rectangle(img_with_bbox, (x, y), (x + w, y + h), (255, 0, 0), 2)

# Display the image with bounding box
plt.imshow(img_with_bbox, cmap='gray')
plt.axis('on')
plt.title("Bounding Box on Image")
plt.show()

largest_bbox


# In[75]:


import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

def visualize_bounding_boxes(image_path):
    # Load the image using PIL
    img = Image.open(image_path)

    # Convert the image to binary using Otsu's thresholding
    _, thresh = cv2.threshold(np.array(img), 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # Find contours in the thresholded image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Extract bounding boxes from the contours
    bboxes = [cv2.boundingRect(cnt) for cnt in contours]

    # Find the bounding box with the largest area (assuming it's the full Kanji character)
    largest_bbox = max(bboxes, key=lambda bbox: bbox[2] * bbox[3])

    # Draw the largest bounding box on the original image
    img_with_bbox = np.array(img).copy()
    x, y, w, h = largest_bbox
```

```
    cv2.rectangle(img_with_bbox, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Display the image with bounding box
    plt.imshow(img_with_bbox, cmap='gray')
    plt.axis('on')
    plt.title("Bounding Box on Image")
    plt.show()

    return largest_bbox

# Usage example:
image_path = 'exampledata20.jpg'
bounding_box = visualize_bounding_boxes(image_path)
print(bounding_box)


# In[76]:


# Sort the bounding boxes based on their area (from largest to smallest)
sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3], reverse=True)

# Extract the top 'n' bounding boxes
num_kanji = 12  # Adjust this based on the number of Kanji characters you expect
selected_bboxes = sorted_bboxes[:num_kanji]

# Extract individual Kanji character images based on the selected bounding boxes
cropped_images = [img.crop((x, y, x + w, y + h)) for x, y, w, h in selected_bboxes]

# Display the extracted Kanji characters
fig, axes = plt.subplots(1, len(cropped_images), figsize=(20, 3))
for ax, cropped_img in zip(axes, cropped_images):
    ax.imshow(cropped_img, cmap='gray')
    ax.axis('off')

plt.tight_layout()
plt.show()


# In[77]:


import pandas as pd
```

```
986
987 # Sort the bounding boxes based on their area (from largest to
    smallest)
988 sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3],
    reverse=True)
989
990 # Number of Kanji characters you expect to extract
991 num_kanji = 12
992
993 # Extract the top 'n' Kanji character images based on the selected
    bounding boxes
994 selected_bboxes = sorted_bboxes[:num_kanji]
995 cropped_images = [img.crop((x, y, x + w, y + h)) for x, y, w, h in
    selected_bboxes]
996
997 # Directory to save the cropped images
998 save_directory = "/Users/macbookpro/Downloads"
999 os.makedirs(save_directory, exist_ok=True)
1000
1001 # Check the existing files in the directory to ensure unique
    filenames
1002 existing_files = os.listdir(save_directory)
1003 existing_counts = [int(f.split('_')[1].split('.')[0]) for f in
    existing_files if "character_" in f]
1004 start_idx = max(existing_counts, default=-1) + 1
1005
1006 # List to store new entries for the CSV
1007 new_entries = []
1008
1009 # Save each cropped image to the directory and add their info to
    new_entries
1010 for idx, cropped_img in enumerate(cropped_images, start=start_idx):
1011     filename = f"character_{idx}.jpg"
1012     cropped_img.save(f"{save_directory}/{filename}")
1013
1014     # Append to new_entries
1015     new_entries.append({"Character Name": filename, "Label": "    ",
     "English Translation": "Earth"})
1016
1017 # Load the existing CSV or create a new one if not present
1018 if os.path.exists('newlabels.csv'):
1019     df = pd.read_csv('newlabels.csv')
1020 else:
1021     df = pd.DataFrame(columns=["Character Name", "Label", "English
    Translation"])
1022
1023 # Append the new entries to the dataframe and save it back
1024 df_new = pd.DataFrame(new_entries)
1025 df = pd.concat([df, df_new], ignore_index=True)
```

```
df.to_csv('newlabels.csv', index=False)


# In[78]:


df


# In[79]:


from PIL import Image
import matplotlib.pyplot as plt

# Load and display the provided image
img_path = "exampledata21.jpg"
img = Image.open(img_path)

# Display the image
plt.imshow(img)
plt.axis('on')  # Display axes for clarity
plt.title("Uploaded Image")
plt.show()


# In[80]:


import cv2
import numpy as np
from matplotlib import pyplot as plt

# Assuming 'img' is already defined and loaded
# Since the image is already in grayscale, we can proceed with
    thresholding
_, thresh = cv2.threshold(np.array(img), 0, 255, cv2.
    THRESH_BINARY_INV + cv2.THRESH_OTSU)

# Find contours in the thresholded image
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)

# Extract bounding boxes from the contours
bboxes = [cv2.boundingRect(cnt) for cnt in contours]

# Find the bounding box with the largest area (assuming it's the
    full Kanji character)
largest_bbox = max(bboxes, key=lambda bbox: bbox[2] * bbox[3])
```

```
1071
1072 # Draw the largest bounding box on the original image
1073 img_with_bbox = np.array(img).copy()
1074 x, y, w, h = largest_bbox
1075 cv2.rectangle(img_with_bbox, (x, y), (x + w, y + h), (255, 0, 0),
     2)
1076
1077 # Display the image with bounding box
1078 plt.imshow(img_with_bbox, cmap='gray')
1079 plt.axis('on')
1080 plt.title("Bounding Box on Image")
1081 plt.show()
1082
1083 largest_bbox
1084
1085
1086 # In[83]:
1087
1088
1089 import cv2
1090 import numpy as np
1091 import matplotlib.pyplot as plt
1092 from PIL import Image
1093
1094 def visualize_bounding_boxes(image_path):
1095     # Load the image using PIL
1096     img = Image.open(image_path)
1097
1098     # Convert the image to binary using Otsu's thresholding
1099     _, thresh = cv2.threshold(np.array(img), 0, 255, cv2.
     THRESH_BINARY_INV + cv2.THRESH_OTSU)
1100
1101     # Find contours in the thresholded image
1102     contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.
     CHAIN_APPROX_SIMPLE)
1103
1104     # Extract bounding boxes from the contours
1105     bboxes = [cv2.boundingRect(cnt) for cnt in contours]
1106
1107     # Find the bounding box with the largest area (assuming it's
     the full Kanji character)
1108     largest_bbox = max(bboxes, key=lambda bbox: bbox[2] * bbox[3])
1109
1110     # Draw the largest bounding box on the original image
1111     img_with_bbox = np.array(img).copy()
1112     x, y, w, h = largest_bbox
1113     cv2.rectangle(img_with_bbox, (x, y), (x + w, y + h), (0, 255,
     0), 2)
1114
```

```
    # Display the image with bounding box
    plt.imshow(img_with_bbox, cmap='gray')
    plt.axis('on')
    plt.title("Bounding Box on Image")
    plt.show()

    return largest_bbox

# Usage example:
image_path = 'exampledata21.jpg'
bounding_box = visualize_bounding_boxes(image_path)
print(bounding_box)


# In[82]:


# Sort the bounding boxes based on their area (from largest to
    smallest)
sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3],
    reverse=True)

# Extract the top 'n' bounding boxes
num_kanji = 13  # Adjust this based on the number of Kanji
    characters you expect
selected_bboxes = sorted_bboxes[:num_kanji]

# Extract individual Kanji character images based on the selected
    bounding boxes
cropped_images = [img.crop((x, y, x + w, y + h)) for x, y, w, h in
    selected_bboxes]

# Display the extracted Kanji characters
fig, axes = plt.subplots(1, len(cropped_images), figsize=(20, 3))
for ax, cropped_img in zip(axes, cropped_images):
    ax.imshow(cropped_img, cmap='gray')
    ax.axis('off')

plt.tight_layout()
plt.show()


# In[84]:


import pandas as pd

# Sort the bounding boxes based on their area (from largest to
    smallest)
```

```
sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3], reverse=True)

# Number of Kanji characters you expect to extract
num_kanji = 13

# Extract the top 'n' Kanji character images based on the selected bounding boxes
selected_bboxes = sorted_bboxes[:num_kanji]
cropped_images = [img.crop((x, y, x + w, y + h)) for x, y, w, h in selected_bboxes]

# Directory to save the cropped images
save_directory = "/Users/macbookpro/Downloads"
os.makedirs(save_directory, exist_ok=True)

# Check the existing files in the directory to ensure unique filenames
existing_files = os.listdir(save_directory)
existing_counts = [int(f.split('_')[1].split('.')[0]) for f in existing_files if "character_" in f]
start_idx = max(existing_counts, default=-1) + 1

# List to store new entries for the CSV
new_entries = []

# Save each cropped image to the directory and add their info to new_entries
for idx, cropped_img in enumerate(cropped_images, start=start_idx):
    filename = f"character_{idx}.jpg"
    cropped_img.save(f"{save_directory}/{filename}")

    # Append to new_entries
    new_entries.append({"Character Name": filename, "Label": "   ", "English Translation": "Book"})

# Load the existing CSV or create a new one if not present
if os.path.exists('newlabels.csv'):
    df = pd.read_csv('newlabels.csv')
else:
    df = pd.DataFrame(columns=["Character Name", "Label", "English Translation"])

# Append the new entries to the dataframe and save it back
df_new = pd.DataFrame(new_entries)
df = pd.concat([df, df_new], ignore_index=True)
df.to_csv('newlabels.csv', index=False)


```

```
# In[140]:


from PIL import Image
import matplotlib.pyplot as plt

# Load and display the provided image
img_path = "exampledata22.jpg"
img = Image.open(img_path)

# Display the image
plt.imshow(img)
plt.axis('on')  # Display axes for clarity
plt.title("Uploaded Image")
plt.show()


# In[141]:


import cv2
import numpy as np
from matplotlib import pyplot as plt

# Assuming 'img' is already defined and loaded
# Since the image is already in grayscale, we can proceed with thresholding
_, thresh = cv2.threshold(np.array(img), 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

# Find contours in the thresholded image
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Extract bounding boxes from the contours
bboxes = [cv2.boundingRect(cnt) for cnt in contours]

# Find the bounding box with the largest area (assuming it's the full Kanji character)
largest_bbox = max(bboxes, key=lambda bbox: bbox[2] * bbox[3])

# Draw the largest bounding box on the original image
img_with_bbox = np.array(img).copy()
x, y, w, h = largest_bbox
cv2.rectangle(img_with_bbox, (x, y), (x + w, y + h), (255, 0, 0), 2)

# Display the image with bounding box
plt.imshow(img_with_bbox, cmap='gray')
```

```
plt.axis('on')
plt.title("Bounding Box on Image")
plt.show()

largest_bbox


# In[146]:


# Sort the bounding boxes based on their area (from largest to smallest)
sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3], reverse=True)

# Extract the top 'n' bounding boxes
num_kanji = 8  # Adjust this based on the number of Kanji characters you expect
selected_bboxes = sorted_bboxes[:num_kanji]

# Extract individual Kanji character images based on the selected bounding boxes
cropped_images = [img.crop((x, y, x + w, y + h)) for x, y, w, h in selected_bboxes]

# Display the extracted Kanji characters
fig, axes = plt.subplots(1, len(cropped_images), figsize=(20, 3))
for ax, cropped_img in zip(axes, cropped_images):
    ax.imshow(cropped_img, cmap='gray')
    ax.axis('off')

plt.tight_layout()
plt.show()


# # Extracting Test data 1

# In[571]:


from PIL import Image
import matplotlib.pyplot as plt

# Load and display the provided image
img_path = "resized_image.jpg"
img = Image.open(img_path)

# Display the image
plt.imshow(img)
```

```
plt.axis('on')  # Display axes for clarity
plt.title("resized image")
plt.show()


# In[563]:


# Sort bounding boxes based on area
sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3],
    reverse=True)

# Set a threshold for the number of Kanji you expect to extract
num_kanji = 6

# Extract the bounding boxes with the largest areas up to the
    threshold
largest_bboxes = sorted_bboxes[:num_kanji]

cropped_images = []
for x, y, w, h in largest_bboxes:
    cropped = img.crop((x, y, x + w, y + h))
    cropped_images.append(cropped)

# Display the cropped images
fig, axes = plt.subplots(1, len(cropped_images), figsize=(20, 3))
for ax, cropped_img in zip(axes, cropped_images):
    ax.imshow(cropped_img, cmap='gray')
    ax.axis('off')

plt.tight_layout()
plt.show()


# In[572]:


from PIL import Image
import matplotlib.pyplot as plt

# Load and display the provided image
img_path = "closing_image.jpg"
img = Image.open(img_path)

# Display the image
plt.imshow(img)
plt.axis('on')  # Display axes for clarity
plt.title("closing image")
plt.show()
```

```
1334
1335
1336 # In[569]:
1337
1338
1339 # Sort bounding boxes based on area
1340 sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3],
        reverse=True)
1341
1342 # Set a threshold for the number of Kanji you expect to extract
1343 num_kanji = 16
1344
1345 # Extract the bounding boxes with the largest areas up to the
        threshold
1346 largest_bboxes = sorted_bboxes[:num_kanji]
1347
1348 cropped_images = []
1349 for x, y, w, h in largest_bboxes:
1350     cropped = img.crop((x, y, x + w, y + h))
1351     cropped_images.append(cropped)
1352
1353 # Display the cropped images
1354 fig, axes = plt.subplots(1, len(cropped_images), figsize=(20, 3))
1355 for ax, cropped_img in zip(axes, cropped_images):
1356     ax.imshow(cropped_img, cmap='gray')
1357     ax.axis('off')
1358
1359 plt.tight_layout()
1360 plt.show()
1361
1362
1363 # In[ ]:
1364
1365
1366
1367
1368
1369 # In[ ]:
1370
1371
1372
1373
1374
1375 # In[ ]:
1376
1377
1378
1379
1380
```

```
# In[ ]:



# In[ ]:



# In[ ]:



# In[ ]:



# In[ ]:



# In[147]:


import pandas as pd

# Sort the bounding boxes based on their area (from largest to smallest)
sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[2] * bbox[3], reverse=True)

# Number of Kanji characters you expect to extract
num_kanji = 8

# Extract the top 'n' Kanji character images based on the selected bounding boxes
selected_bboxes = sorted_bboxes[:num_kanji]
cropped_images = [img.crop((x, y, x + w, y + h)) for x, y, w, h in selected_bboxes]

```

```
# Directory to save the cropped images
save_directory = "/Users/macbookpro/Downloads"
os.makedirs(save_directory, exist_ok=True)

# Check the existing files in the directory to ensure unique filenames
existing_files = os.listdir(save_directory)
existing_counts = [int(f.split('_')[1].split('.')[0]) for f in existing_files if "character_" in f]
start_idx = max(existing_counts, default=-1) + 1

# List to store new entries for the CSV
new_entries = []

# Save each cropped image to the directory and add their info to new_entries
for idx, cropped_img in enumerate(cropped_images, start=start_idx):
    filename = f"character_{idx}.jpg"
    cropped_img.save(f"{save_directory}/{filename}")

    # Append to new_entries
    new_entries.append({"Character Name": filename, "Label": "   ", "English Translation": "Earth"})

# Load the existing CSV or create a new one if not present
if os.path.exists('newlabels.csv'):
    df = pd.read_csv('newlabels.csv')
else:
    df = pd.DataFrame(columns=["Character Name", "Label", "English Translation"])

# Append the new entries to the dataframe and save it back
df_new = pd.DataFrame(new_entries)
df = pd.concat([df, df_new], ignore_index=True)
df.to_csv('newlabels.csv', index=False)


# In[550]:


df


# In[172]:


# Assuming you have labels for the 8 unique Kanji characters
# For demonstration purposes, let's use placeholder labels.
# Replace these with the correct labels for your characters.
```

```
1470
1471
1472 #kanji_labels = ["   ", "   ", "   ", "   ", "   ", "   ", "   ", "
       "]
1473 #english_translations = ["Go", "New", "Talk", "Love", "Woman", "
     Heaven", "Determined", "Window"]
1474
1475 # Load the existing CSV or create a new one if not present
1476 #if os.path.exists('newlabels.csv'):
1477 #    df = pd.read_csv('newlabels.csv')
1478 #else:
1479 #    df = pd.DataFrame(columns=["Character Name", "Label", "English
      Translation"])
1480
1481 # Iterate over the saved files and assign labels
1482 #for idx, file_path in enumerate(saved_files):
1483  #   char_name = os.path.basename(file_path)
1484  #   label = kanji_labels[idx % 8] # Since there are 8 unique
     characters
1485   #  translation = english_translations[idx % 8]
1486   #  new_entry = {"Character Name": char_name, "Label": label, "
     English Translation": translation}
1487   #  df.loc[len(df)] = new_entry
1488
1489 # Save the updated CSV
1490 #df.to_csv('newlabels.csv', index=False)
1491
1492
1493 # In[581]:
1494
1495
1496 import tensorflow as tf
1497 from tensorflow.keras.preprocessing.image import ImageDataGenerator
1498 import numpy as np
1499 # Set seeds for reproducibility
1500 np.random.seed(220074391)
1501 tf.random.set_seed(220074381)
1502
1503
1504 # Path to the directory containing the images
1505 data_directory = "/Users/macbookpro/Downloads"
1506
1507 # Initialize the data generators
1508 datagen = ImageDataGenerator(rescale=1./255, validation_split=0.15)
       # Rescale and set aside 20% of data for validation
1509
1510 # Training data generator
1511 train_generator = datagen.flow_from_dataframe(
1512     dataframe=pd.read_csv('/Users/macbookpro/Downloads/newlabels.
```

```
    csv'),
1513     directory=data_directory,
1514     x_col="Character Name",
1515     y_col="Label",
1516     target_size=(32, 32),
1517     color_mode="rgb",
1518     class_mode="categorical",
1519     batch_size=32,
1520     subset="training"
1521 )
1522
1523 # Validation data generator
1524 validation_generator = datagen.flow_from_dataframe(
1525     dataframe=pd.read_csv('/Users/macbookpro/Downloads/newlabels.
    csv'),
1526     directory=data_directory,
1527     x_col="Character Name",
1528     y_col="Label",
1529     target_size=(32, 32),
1530     color_mode="rgb",
1531     class_mode="categorical",
1532     batch_size=32,
1533     subset="validation"
1534 )
1535
1536
1537 # In[582]:
1538
1539
1540 import pandas as pd
1541
1542 # Load the CSV to determine the number of unique classes
1543 df = pd.read_csv('/Users/macbookpro/Downloads/newlabels.csv')
1544 num_classes = df['Label'].nunique()
1545
1546 def create_complex_model():
1547     model = tf.keras.models.Sequential([
1548         tf.keras.layers.Conv2D(64, (3,3), activation='relu',
    padding='same', input_shape=(32, 32, 3)),
1549         tf.keras.layers.BatchNormalization(),
1550         tf.keras.layers.Conv2D(64, (3,3), activation='relu',
    padding='same'),
1551         tf.keras.layers.BatchNormalization(),
1552         tf.keras.layers.MaxPooling2D(2, 2),
1553         tf.keras.layers.Dropout(0.3),
1554
1555         tf.keras.layers.Conv2D(128, (3,3), activation='relu',
    padding='same'),
1556         tf.keras.layers.BatchNormalization(),
```

```
        tf.keras.layers.Conv2D(128, (3,3), activation='relu',
    padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPooling2D(2, 2),
        tf.keras.layers.Dropout(0.3),

        tf.keras.layers.Conv2D(256, (3,3), activation='relu',
    padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(256, (3,3), activation='relu',
    padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPooling2D(2, 2),
        tf.keras.layers.Dropout(0.3),

        tf.keras.layers.Flatten(),

        tf.keras.layers.Dense(1024, activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.5),

        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.5),

        tf.keras.layers.Dense(len(train_generator.class_indices),
    activation='softmax')  # Number of classes
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy'
    , metrics=['accuracy'])
    return model


# In[583]:


model = create_enhanced_model()


# In[584]:


history = model.fit(
    train_generator,
    epochs=100,  # Number of epochs
    validation_data=validation_generator
)


```

```
# In[585]:


final_val_accuracy = history.history['val_accuracy'][-1]
print(f"Final validation accuracy: {final_val_accuracy * 100:.2f}%")


# In[502]:


from tensorflow.keras.preprocessing import image
import numpy as np

def preprocess_image(img_path, target_size=(32, 32)):
    """Preprocess the image for prediction."""
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = img_array / 255.0  # Normalize to [0,1]
    return np.expand_dims(img_array, axis=0)

# Preprocess the provided images
resized_image = preprocess_image(resized_image_path)
closing_image = preprocess_image(closing_image_path)

# Make predictions
resized_prediction = model.predict(resized_image)
closing_prediction = model.predict(closing_image)

# Decode predictions to get class labels
resized_label = np.argmax(resized_prediction, axis=1)[0]
closing_label = np.argmax(closing_prediction, axis=1)[0]

# Get the class labels from the train generator
class_labels = list(train_generator.class_indices.keys())

print(f"Prediction for resized_image: {class_labels[resized_label]}")
print(f"Prediction for closing_image: {class_labels[closing_label]}")


# In[549]:


from tensorflow.keras.preprocessing import image
import numpy as np
import pandas as pd

```

```
def preprocess_image(img_path, target_size=(32, 32)):
    """Preprocess the image for prediction."""
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = img_array / 255.0  # Normalize to [0,1]
    return np.expand_dims(img_array, axis=0)

# Load the CSV to get the mapping between labels and English
    translations
df = pd.read_csv('/Users/macbookpro/Downloads/newlabels.csv')
label_to_translation = dict(zip(df['Label'], df['English
    Translation']))

# Preprocess the provided images
resized_image = preprocess_image(resized_image_path)
closing_image = preprocess_image(closing_image_path)

# Make predictions
resized_prediction = model.predict(resized_image)
closing_prediction = model.predict(closing_image)

# Decode predictions to get class labels
resized_label_index = np.argmax(resized_prediction, axis=1)[0]
closing_label_index = np.argmax(closing_prediction, axis=1)[0]

# Get the class labels from the train generator
class_labels = list(train_generator.class_indices.keys())

# Fetch the predicted label and its corresponding English
    translation
resized_predicted_label = class_labels[resized_label_index]
closing_predicted_label = class_labels[closing_label_index]

resized_predicted_translation = label_to_translation[
    resized_predicted_label]
closing_predicted_translation = label_to_translation[
    closing_predicted_label]

print(f"Prediction for resized_image: {resized_predicted_label} (
    English: {resized_predicted_translation})")
print(f"Prediction for closing_image: {closing_predicted_label} (
    English: {closing_predicted_translation})")


# In[503]:


# Given true labels and predicted labels
true_labels = ["   ", "   "]
```

```
predicted_labels = [class_labels[resized_label], class_labels[
    closing_label]]

correct_predictions = sum([true == pred for true, pred in zip(
    true_labels, predicted_labels)])
accuracy = correct_predictions / len(true_labels) * 100

print(f"Accuracy: {accuracy:.2f}%")


# In[505]:


from tensorflow.keras.preprocessing import image
import numpy as np
import pandas as pd

# Load the CSV to get the mapping between labels and English
    translations
df = pd.read_csv('newlabels.csv')
label_to_translation = dict(zip(df['Label'], df['English
    Translation']))

# Path to the image
img_path = '063.gif'

# Load and preprocess the image
img = image.load_img(img_path, target_size=(32, 32))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0) / 255.0

# Predict the class
prediction = model.predict(img_array)
predicted_class = np.argmax(prediction, axis=1)[0]

# Fetch the predicted label and its corresponding English
    translation
predicted_label = class_labels[predicted_class]
predicted_translation = label_to_translation[predicted_label]

print(f"Prediction for 063.gif: {predicted_label} (English: {
    predicted_translation})")


# In[508]:


# Instantiate the model
model_instance = create_complex_model()
```

```
1732
1733 # Save the model
1734 model_instance.save('complex_model.h5')
1735
1736
1737 # # Bayesian Optimization on model
1738
1739 # In[553]:
1740
1741
1742 import tensorflow as tf
1743 from tensorflow.keras.preprocessing.image import ImageDataGenerator
1744 from tensorflow.keras.callbacks import EarlyStopping,
        ModelCheckpoint
1745 import numpy as np
1746 import pandas as pd
1747
1748 # Set seeds for reproducibility
1749 np.random.seed(220074391)
1750 tf.random.set_seed(220074381)
1751
1752 data_directory = "/Users/macbookpro/Downloads"
1753 datagen = ImageDataGenerator(rescale=1./255, validation_split=0.15)
1754
1755 train_generator = datagen.flow_from_dataframe(
1756     dataframe=pd.read_csv('/Users/macbookpro/Downloads/newlabels.
        csv'),
1757     directory=data_directory,
1758     x_col="Character Name",
1759     y_col="Label",
1760     target_size=(32, 32),
1761     color_mode="rgb",
1762     class_mode="categorical",
1763     batch_size=32,
1764     subset="training"
1765 )
1766
1767 validation_generator = datagen.flow_from_dataframe(
1768     dataframe=pd.read_csv('/Users/macbookpro/Downloads/newlabels.
        csv'),
1769     directory=data_directory,
1770     x_col="Character Name",
1771     y_col="Label",
1772     target_size=(32, 32),
1773     color_mode="rgb",
1774     class_mode="categorical",
1775     batch_size=32,
1776     subset="validation"
1777 )
```

```
1778
1779 def create_complex_model(lr=1e-3, dropout_rate=0.3, num_filters
     =[64, 128, 256]):
1780     model = tf.keras.models.Sequential([
1781         tf.keras.layers.Conv2D(num_filters[0], (3,3), activation='
     relu', padding='same', input_shape=(32, 32, 3)),
1782         tf.keras.layers.BatchNormalization(),
1783         tf.keras.layers.Conv2D(num_filters[0], (3,3), activation='
     relu', padding='same'),
1784         tf.keras.layers.BatchNormalization(),
1785         tf.keras.layers.MaxPooling2D(2, 2),
1786         tf.keras.layers.Dropout(dropout_rate),
1787
1788         tf.keras.layers.Conv2D(num_filters[1], (3,3), activation='
     relu', padding='same'),
1789         tf.keras.layers.BatchNormalization(),
1790         tf.keras.layers.Conv2D(num_filters[1], (3,3), activation='
     relu', padding='same'),
1791         tf.keras.layers.BatchNormalization(),
1792         tf.keras.layers.MaxPooling2D(2, 2),
1793         tf.keras.layers.Dropout(dropout_rate),
1794
1795         tf.keras.layers.Conv2D(num_filters[2], (3,3), activation='
     relu', padding='same'),
1796         tf.keras.layers.BatchNormalization(),
1797         tf.keras.layers.Conv2D(num_filters[2], (3,3), activation='
     relu', padding='same'),
1798         tf.keras.layers.BatchNormalization(),
1799         tf.keras.layers.MaxPooling2D(2, 2),
1800         tf.keras.layers.Dropout(dropout_rate),
1801
1802         tf.keras.layers.Flatten(),
1803         tf.keras.layers.Dense(1024, activation='relu'),
1804         tf.keras.layers.BatchNormalization(),
1805         tf.keras.layers.Dropout(0.5),
1806
1807         tf.keras.layers.Dense(512, activation='relu'),
1808         tf.keras.layers.BatchNormalization(),
1809         tf.keras.layers.Dropout(0.5),
1810
1811         tf.keras.layers.Dense(len(train_generator.class_indices),
     activation='softmax')
1812     ])
1813     model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=
     lr), loss='categorical_crossentropy', metrics=['accuracy'])
1814     return model
1815
1816 # Callbacks
1817 early_stopping = EarlyStopping(monitor='val_loss', patience=10,
```

```
    restore_best_weights=True)
model_checkpoint = ModelCheckpoint("best_model.h5", monitor='
    val_loss', save_best_only=True)

model = create_complex_model()
history = model.fit(
    train_generator,
    epochs=100,
    validation_data=validation_generator,
    callbacks=[early_stopping, model_checkpoint]
)


# In[515]:


# Replace 'fire' with 'Fire' in the entire DataFrame
df.replace('fire', 'Fire', inplace=True)

# If you specifically want to replace only in a certain column, say
    'English', you can do:
# df['English'].replace('fire', 'Fire', inplace=True)

# Save the updated DataFrame back to the CSV if needed
df.to_csv('/Users/macbookpro/Downloads/newlabels.csv', index=False)


# In[518]:


import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

# Load the CSV
df = pd.read_csv('/Users/macbookpro/Downloads/newlabels.csv')

# Class distribution
class_counts = df['Label'].value_counts()

# Define font properties using the provided font path
font_path = '/Users/macbookpro/Downloads/NotoSansJP-Regular.ttf'
font_properties = FontProperties(fname=font_path)

# Plot
ax = class_counts.plot(kind='bar', color='skyblue')
plt.title('Class Distribution', fontproperties=font_properties)
plt.xlabel('Classes', fontproperties=font_properties)
plt.ylabel('Number of Images', fontproperties=font_properties)
```

```
1864
1865 # Set tick labels with the font properties
1866 ax.set_xticklabels(class_counts.index, fontproperties=
     font_properties)
1867 plt.show()
1868
1869
1870 # In[542]:
1871
1872
1873 # Class distribution based on English meanings
1874 english_counts = df['English Translation'].value_counts()
1875
1876 # Plot
1877 english_counts.plot(kind='barh', color='lightgreen')
1878 plt.title('Class Distribution based on English Meanings')
1879 plt.xlabel('Number of Images')
1880 plt.ylabel('English Meanings')
1881 plt.show()
1882
1883
1884 # In[513]:
1885
1886
1887 from PIL import Image
1888
1889 # Extract image sizes
1890 image_sizes = []
1891
1892 for img_name in df['Character Name']:
1893     img_path = os.path.join(data_directory, img_name)
1894     with Image.open(img_path) as img:
1895         width, height = img.size
1896         image_sizes.append((width, height))
1897
1898 # Convert to DataFrame for easier plotting
1899 sizes_df = pd.DataFrame(image_sizes, columns=['Width', 'Height'])
1900
1901 # Plot
1902 plt.figure(figsize=(12, 6))
1903
1904 # Width distribution
1905 plt.subplot(1, 2, 1)
1906 plt.hist(sizes_df['Width'], bins=30, color='lightcoral', edgecolor=
     'black')
1907 plt.title('Image Width Distribution')
1908 plt.xlabel('Width')
1909 plt.ylabel('Number of Images')
1910
```

```
# Height distribution
plt.subplot(1, 2, 2)
plt.hist(sizes_df['Height'], bins=30, color='lightblue', edgecolor='black')
plt.title('Image Height Distribution')
plt.xlabel('Height')
plt.ylabel('Number of Images')

plt.tight_layout()
plt.show()


# In[547]:


import numpy as np
import matplotlib.pyplot as plt

# Load an image (you can replace this with any image from your dataset)
img = image.load_img("exampledata7.jpg", target_size=(32, 32))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)
img_tensor /= 255.  # Model input should be normalized to [0,1]

# Extract model layer outputs
layer_outputs = [layer.output for layer in model.layers if "conv" in layer.name]  # Assuming "conv" in the name of convolutional layers

# Create a model for feature map extraction
activation_model = tf.keras.models.Model(inputs=model.input, outputs=layer_outputs)

# Get feature maps
activations = activation_model.predict(img_tensor)

# Visualize the feature maps of the first convolutional layer
first_layer_activation = activations[0]
plt.figure(figsize=(10, 10))
for i in range(first_layer_activation.shape[-1]):
    plt.subplot(8, 8, i + 1)  # Assuming 64 filters, adjust if different
    plt.imshow(first_layer_activation[0, :, :, i], cmap='viridis')
    plt.axis('off')
plt.show()


# In[ ]:
```

```
1954
1955
1956
1957
1958
1959 # In[ ]:
```