Android Application for Kanken Preparation

John AOUSSOU
ML Engineer
personal project
September-October 2022

1 Introduction and Problem Statement

The Japan Kanji Aptitude Test (日本漢字能力検定), commonly known as "Kanken" (漢検), is designed to thoroughly test a person's knowledge of Chinese characters (Kanji) used in Japanese. While numerous applications exist to prepare for this test, they often exhibit shortcomings:

- **Limited Scope:** Many apps primarily focus on Kanji reading and writing, neglecting other question types present in the actual test, such as stroke order and count, onyomi vs kunyomi, okurigana, radical names, four-character compounds, homonyms, synonyms and antonyms.
- **Lack of Adaptive Learning:** Existing applications typically do not adapt to the user's specific weaknesses. Users often repeat entire exercise sets, including characters they have already mastered alongside those requiring more practice.
- **Format Discrepancy:** While the content might be similar, the presentation and format in many apps do not accurately replicate the experience of the official Kanken test.
- **Lenient Character Recognition:** Existing apps often employ overly **too** lenient character recognition algorithms, marking characters as correct even if they are only approximately similar to the target Kanji ('close enough'). Consequently, learners might repeatedly practice incorrect forms without being flagged.

Driven by a desire for more efficient and targeted preparation, the decision was made to develop a custom Android application addressing these issues.

2 Application Requirements

Creating such an application necessitated several key components:

- **Practice Data:** A comprehensive dataset of Kanken-related questions and characters.
- **Handwriting Recognition:** An Android module specifically for recognizing handwritten Kanji.
- **UI Design:** An interface mimicking the format and layout of the official Kanken test papers.
- **Adaptive Logic:** An algorithm to prioritize practice on Kanji characters the user struggles with.

3 Development Process

3.1 Kanken Testing Data Acquisition

The primary dataset was acquired by scraping the *Kanken Jitenon* website using `Python` scripts.

3.2 Handwritten Kanji Recognition

Developing an effective handwriting recognition module presented unique challenges:

3.2.1 Initial Assessment and Challenges

At the time of development, readily integrable Android modules for Kanji handwriting recognition were scarce. Furthermore, existing recognition systems are generally designed for input assistance (predicting the most likely character even from imperfect input), not for learning assessment. For Kanken practice, if a user writes a Kanji incorrectly or writes a non-existent character, the system should ideally *not* provide a "best guess," as this would be counterproductive to learning precise form. This necessitated the creation of a custom recognition module.

3.2.2 First Training Attempt (ETL Database)

The initial approach involved training a model using data from the *ETL Character Database*. To ensure efficiency on a mobile platform, a lightweight model, `EfficientNet`, was chosen. Training was performed using `Keras` on a personal local machine equipped with dual `RTX 2080 Ti` GPUs. The resulting model was converted to the `TensorFlow Lite` format for deployment on Android.

Unfortunately, the performance of this initial model was poor. This was attributed to the limited quantity and variety within the ETL dataset.

3.2.3 Second Training Attempt (Combined Datasets)

To address the data limitations, a much larger dataset, the Traditional Chinese Handwriting Dataset, was incorporated. While this dataset primarily features handwriting captured via computer interfaces (leading to less variety in stroke styles compared to natural handwriting), its sheer volume and the number of contributors offered significant advantages.

The ETL and Traditional Chinese datasets were combined, and the `EfficientNet` model was retrained. This yielded excellent results, with the model correctly identifying virtually all handwritten Kanji input.

An interesting and beneficial side effect emerged from using the Traditional Chinese dataset: because traditional Chinese includes many more characters than the standard Japanese Jōyō Kanji list, the model became highly sensitive to accuracy. Users needed to write the Japanese Kanji quite precisely for it to be correctly identified, reinforcing proper writing habits crucial for the Kanken test.

3.3 The Application Implementation

The Android application itself was built using modern development practices:

UI Framework: `Jetpack Compose` was used for building the user interface, allowing for a declarative and efficient UI development process.

Architecture: The `MVVM (Model-View-ViewModel)` architectural pattern was followed to ensure separation of concerns and maintainability.

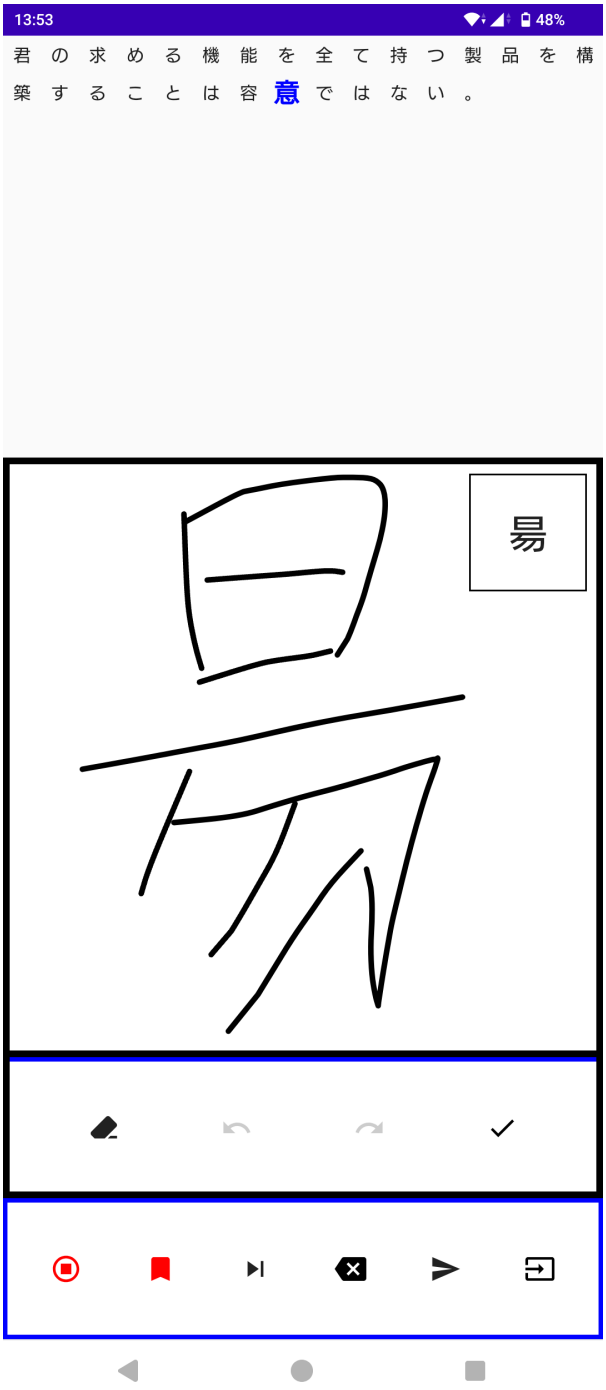
Programming Language: The application was coded entirely in `Kotlin`.

Database Management: `SQLDelight` was employed for managing the local database, which was prepopulated with the scraped Kanken data from Jitenon.

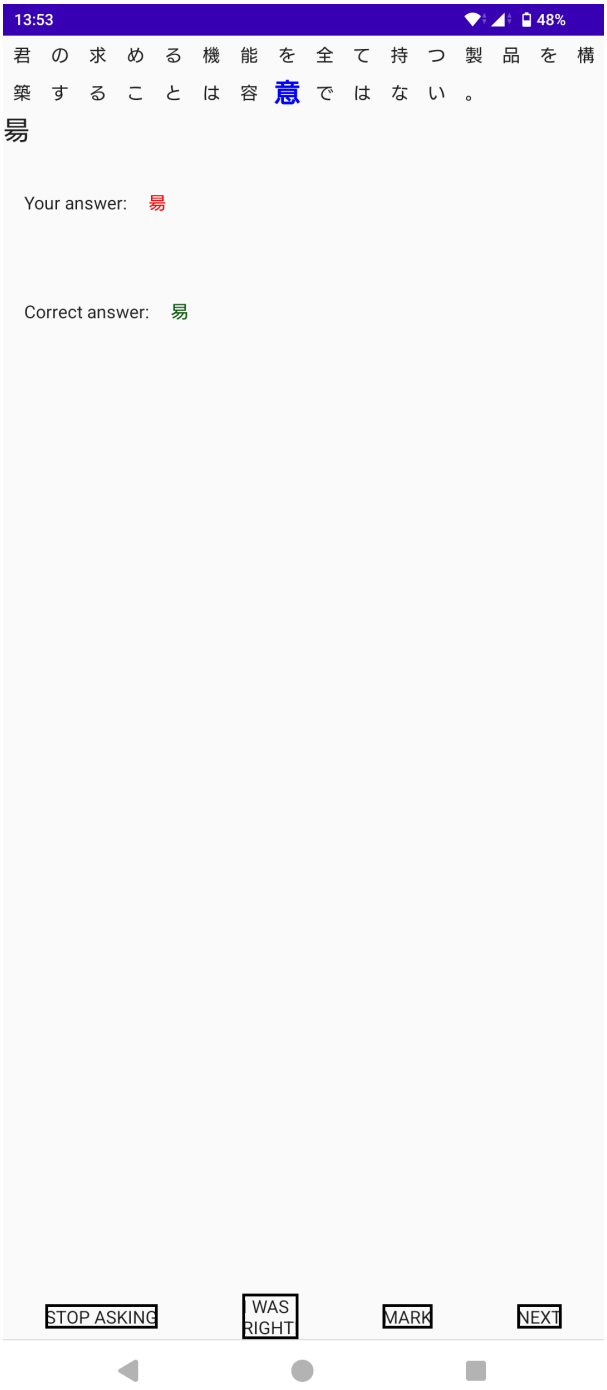
4 Result and Conclusion

The resulting application successfully served its primary purpose. Due to the use of potentially non-commercially licensed data (from scraping and datasets) and time constraints preventing the development of a full production-grade application, it was not released publicly.

However, using this custom-built application for preparation, the developer was able to successfully pass the **Kanken Level 4**. This outcome validates the effectiveness of the targeted approach, particularly the custom handwriting recognition module tailored for learning assessment and the focus on practicing difficult Kanji.



(a) User identifies incorrect Kanji.



(b) User provides incorrect replacement.

Figure 1: Example illustrating the 誤字訂正 (Kanji Correction) problem format. This task requires identifying an incorrect Kanji within a word and writing the correct homophone. The scenario depicted shows the user successfully identifying the target Kanji (a) but then writing an incorrect character as the replacement (b) —in this instance, a non-Jōyō Kanji. This highlights the application's strict character recognition, designed to help learners identify specific errors rather than accepting approximations, thus facilitating learning from 'useful mistakes'.