DEPARTMENT OF COMPUTER SCIENCE

Hand-Written Chinese Character Error Detection

A web application for assisting Chinese character learning

Shuye Liu

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering.

Wednesday 27th May, 2020

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of BSc in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Shuye Liu, Wednesday 27th May, 2020

Project Mitigation Plan

Event/Issue	Potential/actual Impact on project	Action(s) taken to mitigate impact on project outcomes	Remaining impact
Unable to collect more hand-written stroke samples from people during the lockdown	I had less samples to train the stroke type classification model (a Convolutional Neural Network model), which affected it's accuracy.	Generated more samples based on the collected samples through compressing vertically, horizontally and rotating individual samples.	Because I generated more samples from the existing one, it could potentially lead to the model overfitting to the dataset. And the model performance on some nonstandard stroke shapes is not desirable.
Unable to conduct face-to-face user study on a wider range of people.	Small user samples means less feedback and less evidence to verify the effectiveness of the application.	Conducted online survey on 5 participants through audio or video calls. Sent online questionnaires to collect user feedback.	It's likely that a comprehensive understanding of the effectiveness and the usability of the application was not achieved, given the very few feedback from users

Acknowledgement

I would like to thank Prof. Ian Nabney for his dedicated support and guidance throughout the project, especially during this extraordinary time period. I would also like to thank all my friends who helped with sample collection, testing and user studies.

Abstract

Written Chinese uses logograms to represent a word or a phrase, which is distinct from alphabetic languages such as English and German etc. Therefore, there is no alphabet in which the language can be fully expressed. Each character is comprised of different radicals and strokes with various shapes and orientations, while having their own meaning. Chinese learners often struggle to write characters correctly given the significant differences between their native language and Chinese [6]. The common issues are missing strokes, strokes in wrong positions and out-of-shape strokes. These sometimes could result in a wrong character, or a completely different character having a different meaning. In this project, I will propose a method of detecting errors in hand-written Chinese characters, which incorporates a machine learning technique – Convolutional Neural Networks. The method will be tested out by implementing a web application which detects errors based on the proposed method. The application will aim to point out the missing or incorrectly written strokes while users are writing and provide feedback on how to improve.

Contents

Chapter 1

Introduction

This chapter will explain my motivation for building this project and analyse the need for such a tool to help Chinese learners practise writing characters, as well as the goals I want to achieve.

1.1 Current Tools For Aiding Character Learning

There are a number of character learning tools currently available. However, the majority of these applications focus mainly on recognising characters instead of teaching users how to write them. The most common learning methods that they provide are: associating a character with its meaning, or associating a character with a word that it makes up (Figure 1.1). Some others would allow users to build up a character by the individual strokes. Beginners tend to memorise a character as a whole, instead of its individual components [1]. This could mean that when writing down the character, details might be lost. Therefore, such applications or games aid visual recognition of characters but do not necessarily help writing characters.

Figure 1.1: Example from a Chinese learning app – Chineasy.

There are also a few applications that encourage users to write characters. They do so mostly by providing grey-coloured templates for users to write on. Judging from the position of the stroke written (whether it is out of the grey stroke), they will then fill up the grey stroke with a dark colour if it is correct (Figure 1.2). This way of learning indeed involves writing actual characters instead of just looking at them. However, what they are writing is not shown on screen using these applications. Instead, only the

template stroke is shown when they have written a correct stroke. Users are not able to see instant visual feedback on what they are writing or, how well they are writing. It could be that they are writing the correct strokes but when put together they might not satisfy the spatial relationships of components and relationships with other strokes. This is because the apps allow too much latitude on where the stroke can be drawn and allow too much tolerance on different sizes of strokes. Written strokes are considered correct as long as they are in the right position and are similar to the stroke intended.

Figure 1.2: Example from a Chinese learning app – ZIHOP.

Only a few applications allow users to write. They judge the correctness of writing based on the full character that they write. 'Stroke Order' is an iOS application that provides an error detection engine for hand-written characters. It is not clear how the detection mechanism is implemented and the detection result is not satisfactory. It is able to detect if a character is written correctly or not in general, but even if a stroke is misplaced, it will still give positive feedback (Figure 1.3). On top of this, it only indicates if a character is correct or incorrect, without detailed feedback pointing out where strokes or their order are incorrect. [2] proposed a way of detecting errors by Attributed Relational Graph(ARG). It defined all possible spatial relationships between two strokes and formed ARGs based on the template and user-written character. By using ARG matching algorithms, it was possible to detect errors and point out specific area where errors occur. The proposed method worked well on detecting errors in terms of stroke relationships. It was able to detect a number of types of error. However, it is purely based on the geometry of the character. Therefore, it does not have much tolerance for small inaccuracies. For example, if two strokes that are supposed to connect have a small distance between them, even if the distance is not noticeable immediately, it will be considered wrong. In addition, the detector is only checked with a small dataset of simple characters. More complex characters have not been tested.

1.2 Challenges Faced by Chinese Learners in Writing Characters

The Foreign Service Institute (FSI) of America published a list of learning hours of different languages for native English speakers [4]. The list approximates how many hours a native English speaker needs to learn a certain language in order to reach a 'General Professional Proficiency in Speaking and Reading' and categorised languages into 4 groups. Chinese is listed in category 4 - Languages which are exceptionally difficult for native English speakers.

For learners whose native language is written in an alphabet-based system, Chinese characters can be significantly different from what they are used to [6]. Potentially such learners need to rethink how languages work and accept the concept of using logograms to write a language.

Chinese learners often write characters with reference to templates (correct characters), but this can

Figure 1.3: Example from a Chinese learning app – Stroke Order.

be problematic because such templates do not indicate the correct stroke order. Especially nowadays with digital input methods using Pinyin (transforming pronunciations into characters), more and more Chinese users cannot write characters correctly without them [5]. Not writing in the correct stroke order could result in out-of-shape characters.

Components in a Chinese character often have spatial relationships. This is not easy to pick up by foreign Chinese learners [7]. For example, '起 (rise)' which follows Surrounded-From-Bottom-Left structure (⿺), may be incorrectly written as '走己' (Left-Right structure ⿰). This is due to a lack of understanding of the spatial features of components.

Sometimes the difference between Chinese characters are insignificant, yet they convey completely different meanings, such as 己 (oneself), 已 (already) and 巳 (year of the Snake). In addition, variation of characters used in different countries also poses a challenge. For example, 每 (every) in Japanese Kanji is written as 每 (every) in Chinese. Such small differences need exceptional care to distinguish.

The combination of strokes or components in a character is never random. They follow the basic combination of connect, intersect and detach [15]. Components have designated spatial relationships within a character. Writing must follow stroke orders with basic principles. All of these require learners to make effort to practice, more importantly, to get used to a new way of writing.

1.3 My solution

This project aims to provide an error detector with which Chinese learners can practise writing characters and get feedback on their writings as they write. The error detector should be able to:

1. Provide a canvas for users to write on.
2. Report on wrong stroke order.
3. Report on incorrect stroke (incorrect shape, position, etc.).
4. Report on incorrect spatial relationship of strokes.

Given the strengths and weaknesses of the applications currently available, I propose a method to build an error detection application using Convolutional Neural Networks to examine the correctness of a stroke.

A template of Kai Shu writing will be used to verify its size and position. Finally, based on its position and spatial relationship with other strokes, it can then be determined whether or not it is correct in the character. A brief flow diagram of the system is depicted in Figure 1.4.

Note

The user study conducted to verify the performance of the final application satisfies the ethics requirements as specified by ethics application 97842. Details can be found in Appendix C.

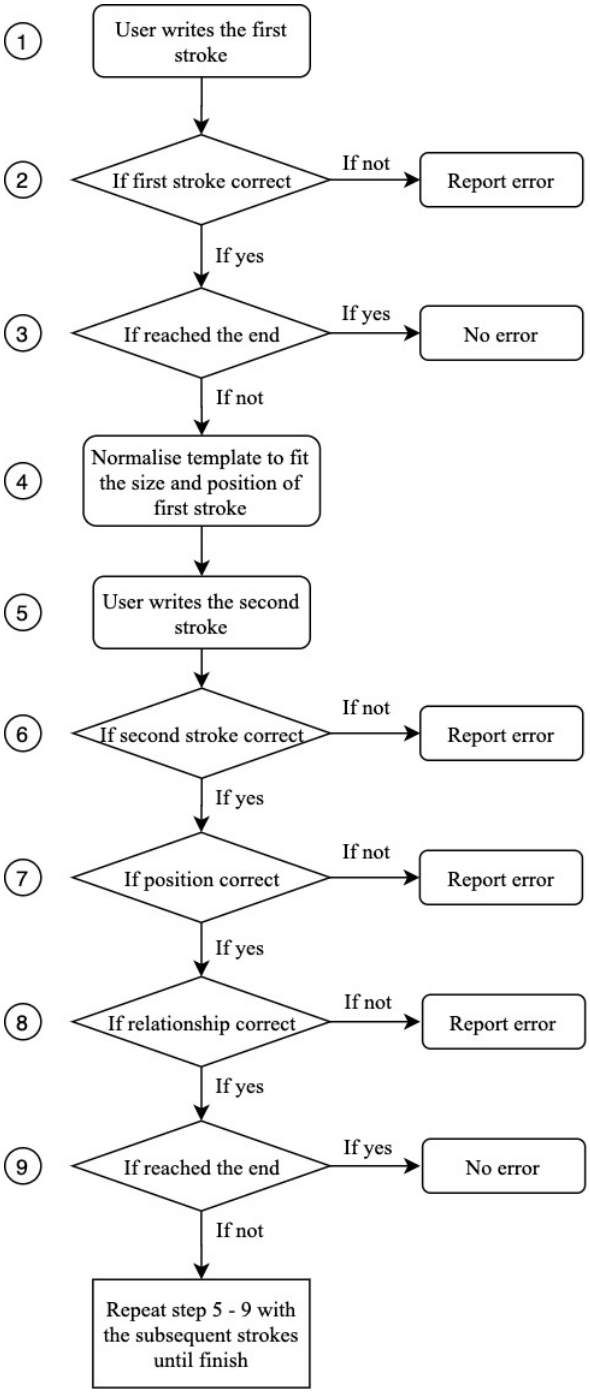


Figure 1.4: Flow diagram of the detection system.

Chapter 2

Contextual Background

In this chapter, I will explain the relevant background information about this project, as well as the technologies or tools used in the project.

2.1 A Brief Introduction To Chinese Characters

Chinese characters (Han Zi 汉字) are the written form of the Chinese language. It is one of the earliest forms of written language in the world which dates back to as early as late Shang dynasty (1250-1050BC) [8]. It is believed that the Chinese characters originated from religious rituals [8], where the Shang king would perform divine ceremonies to communicate with his ancestors regarding topics such as weather forecasting, crop harvesting and religious scarifies etc. The answers were then recorded on oracle bones or bronze vessels, also known as the oracle bone script.

Like many other languages, Chinese Characters evolved through different stages over thousands of years of history. Different styles of writing emerged in this process. Seven major styles of writing are identified in Chinese calligraphy today, with each linked to an era in Chinese History (Figure 2.1). Among all the writing styles, Kai Shu is the most widely used in modern Chinese handwriting. Therefore our error detection will be based on Kai Shu, taking Kai Shu writing as the standard writing.

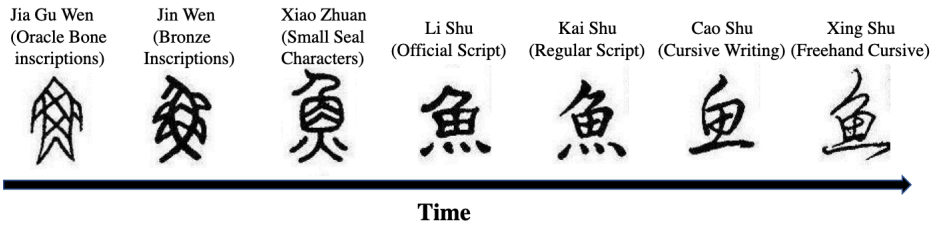


Figure 2.1: Different styles for 鱼 (fish).

On top of the different styles of writing, there are different forms of Chinese characters. In the 1950s, P.R.China initiated plans to decrease illiteracy by simplifying a part of the Chinese characters system, which are called the Simplified Characters and are now commonly used in Mainland China, Singapore and Malaysia. The unsimplified ones are called Traditional Characters, which are used in regions such as Taiwan, Hong Kong SAR and Macao SAR [9].

There are many Chinese characters, but the exact number is not clear as it depends on the way of counting. The *Kangxi Dictionary* from Qing dynasty recorded more than 47,000 characters [11]. *Hanyu Dazidian (Great Compendium of Chinese Characters)* published in 2010 includes 60,370 characters. However, *Table of General Standard Chinese Characters* [10] published by the State Council of the

P.R.China contains 8,104 characters, of which 6,500 characters are listed as common, while only 3,500 characters are listed as frequent. A great number of characters used in history are archaic and have become deprecated, and are no longer in use. A study in 2005 analysed a corpus of more than 700 million Chinese characters, most of which were collected from news media, TV broadcasts and publications. Results showed that 934 characters are enough to cover 90% of all text material, and 2,314 characters can cover 99% [12]. It is reasonable to say that a general understanding of Chinese texts can be achieved by the knowledge of around 2,500 characters. Therefore, in this project I'll be looking at 2,500 simplified characters which are the most widely used and studied.

2.1.1 Structures of Chinese Characters

Each Chinese character consists of one or more components, which in general divides characters into two groups - single character (**独体字**) and compound character (**合体字**) [3].

Single characters refer to characters that contain only one independent component, instead of two or more separable components. Single characters are often pictograms or ideograms, which evolved through drawings and iconic illustrations. Therefore, each of them is an inseparable entity, such as 日 (sun), 月 (moon), 上 (upward) and 二 (two).

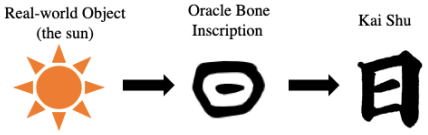


Figure 2.2: Example of a pictogram 日 (sun).

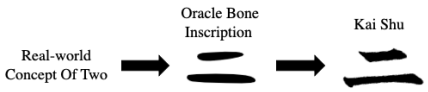


Figure 2.3: Example of an ideogram 二 (two).

Compound characters refer to characters that are comprised of two or more components. They make up the majority of the Chinese characters. In general, 12 types of structure are recognised in compound characters, as shown in Table 2.1.

Structure	Examples	Note
⿰	外 (outside) 叶 (leaf)	Left-Right
⿱	昏 (dawn) 星 (star)	Top-Bottom
⿲	树 (tree) 谢 (thank)	Left-Middle-Right
⿳	蓝 (blue) 鼻 (nose)	Top-Middle-Bottom
⿵	匀 (even) 可 (can)	Surrounded From Upper Right
⿸	床 (bed) 店 (shop)	Surrounded From Upper Left
⿺	过 (pass) 建 (build)	Surrounded From Bottom Left
⿷	问 (ask) 同 (same)	Surrounded From Above
⿹	凶 (terrible) 函 (letter)	Surrounded From below
⿶	医 (cure) 匣 (box)	Surrounded From Left
⿴	图 (image) 国 (nation)	Surrounded Fully
⿻	坐 (sit) 噩 (shocking)	Overlaid

Table 2.1: Compound character structures.

Single characters only take up a small proportion in all characters that are now being actively used. However, they are essential to the character system, because most single characters serve as the components which make up compound characters. Therefore, they are core to the Chinese writing system, because they are key to character composition. For example, 口 (mouth) , as one of the single characters, is used in more than 700 compound characters, such as 吐 (spit), 吹 (blow) and 吞 (shallow) etc, which in general are associated with mouth movement. Good understanding and memory of single characters could potentially alleviate burdens when memorising compound characters.

2.1.2 Writing Chinese Characters

As mentioned in the previous section, each Chinese character contains one or more components. Components cannot be subdivided into smaller components, but components can be further broken into strokes. A stroke (Bi Hua 笔画) is a line (cursive or straight) or dot formed between a pen-down and a pen-lift [13]. In other words, strokes constitute components. Components constitute characters. Thus, strokes are the fundamental building blocks of Chinese characters.

According to *Chinese Character Turning Stroke Standard of GB 13000.1 Character Set* published by the State Language Commission of P.R.China in 2001 [14], strokes are classified into 32 types, of which 6 are basic strokes, 25 are complex strokes, and remaining one is a special stroke for Kai Shu printing (乀) (Table 2.2). This standard only applies to Chinese character writing, which excludes some Japanese-made characters (also known as Kanji). For the sake of consistency, we will use the standard mentioned above.

Three types of relationship between individual strokes can be found in a Chinese character. They either intersect, connect, or detach [15]. Examples of stroke relationship can be found in Figure 2.4.

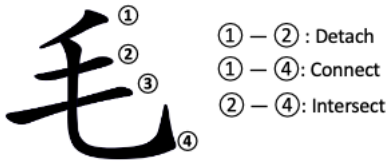


Figure 2.4: Example of stroke relationships in 毛 (fur).

Stroke order refers to the order in which strokes are written to form a character. The following rules regarding stroke orders for Kai Shu apply to writing characters in general [16]. An example can be found in Figure 2.5.

- Heng before shu, pie before na
- Top to bottom, left to right
- Outside to inside
- Middle before left and right
- Finish inside before enclosing

Figure 2.5: Example of the stroke order of 和 (harmony).

Stroke orders are an important part in Chinese character writing. Correct stroke orders help learners understand the structure of characters, keeping the characters in the correct shape. For example, in the character 圃 (garden), 冂 should be written first, 甫 comes after, and finally 一 will close up the bottom (Figure 2.6). However, if 甫 is written before 冂, when surrounding 甫 with 冂, the size is decided by the how big 甫 is. If 甫 is written bigger, so will the entire character. Having 冂 decided by how big 甫 is in the middle would likely result in an out-of-shape character (Figure 2.7).

In addition, using a fixed stroke order can help memorising the character as well as improving efficiency when writing [17]. Last but not the least, studying the correct stroke orders aids learning new characters in the future as the stroke order of same component remains the same in different characters.

Figure 2.6: 圃 when written with the correct stroke order.

Figure 2.7: 圃 when written with an incorrect stroke order.

Stroke	name	Stroke	name
一	Héng 横	𠃌	Héng Gōu 横钩
丨	Shù 竖	𠃍	Héng Zhé Gōu 横折钩
丿	Piě 撇	𠄎	Héng Zhé Tí 横折提
乀	Nà 捺	𠃎	Héng Zhé Wān 横折弯
丶	Diǎn 点	𠃋	Héng Zhé Zhé 横折折
𠂆	Héng Zhé 横折	𠃏	Héng Xié Gōu 横斜钩
𠂇	Héng Piě 横撇	𠃑	Héng Zhé Wān Gōu 横折弯钩
𠂉	Tí 提	𠃒	Héng Piě Wān Gōu 横撇弯钩
𠃊	Héng Zhé Zhé Piě 横折折撇	𠃓	Shù Zhé Piě 竖折撇
𠃋	Héng Zhé Zhé Zhé 横折折折	𠃌	Shù Zhé Zhé 竖折折
𠃍	Héng Zhé Zhé Zhé Gōu 横折折折钩	𠃍	Shù Zhé Zhé Gōu 竖折折钩
𠃎	Shù Tí 竖提	𠃎	Piě Diǎn 撇点
𠃏	Shù Zhé 竖折	𠃏	Piě Zhé 撇折
𠃐	Shù Gōu 竖钩	𠃐	Xié Gōu 斜钩
𠃑	Shù Wān 竖弯	𠃑	Wān Gōu 弯钩
𠃒	Shù Wān Gōu 竖弯钩	𠃒	Wò Gōu 卧钩

Table 2.2: List of strokes defined by GB 13000.1 Character Set.

Stroke orders are a set of rules developed over time, aiming at maintaining the consistency of writing and writing characters in correct shapes. However, the stroke order of a character may not be universal across different countries. Different countries or regions where Chinese characters are used may have their own standardisation of stroke orders. They remain consistent in the majority of characters, with a few differences. In this project, I will adopt the standard published by the State Language Commission of P.R.China in 1997 [18].

2.2 Convolutional Neural Network(CNN)

2.2.1 2D Convolution

Convolution is a specialised type of linear operation used for feature extraction [20]. In digital image processing, a convolution is often performed between a convolution matrix (kernel) and an image, as shown in Figure 2.8. In the example, a 3×3 kernel m is applied on a 3×3 part of the image f. A convolution is performed by taking pair-wise multiplication between the two corresponding values in the image and the kernel. Adding all the results of multiplication gives the final value. The result of a convolution (i.e. an integer) will be the new pixel value for the centre pixel. This operation is performed on every pixel in the image by sliding the kernel through the image one pixel at a time. Using kernels as masks, convolution on images can achieve effects such as blurring, sharpening, embossing and edge detection.

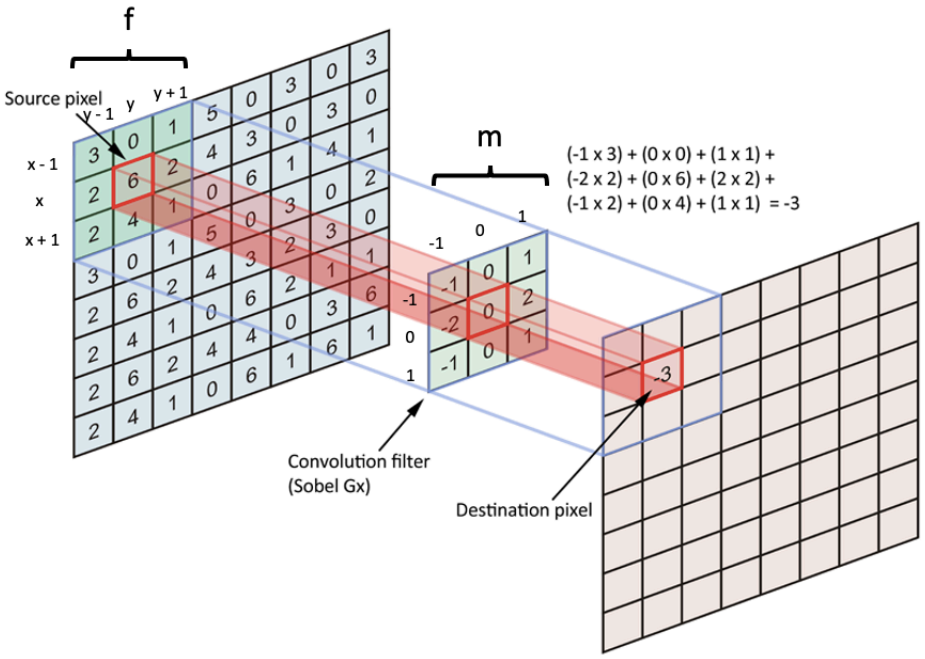


Figure 2.8: Example of convolution operation. [21]

2.2.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a variation of a Neural Network. The structure of the network is inspired by animal visual cortex [19], where a single neuron responds to a particular area of the visual field, and a collection of neurons are able to cover the entire image. CNNs have a wide range of applications, mostly in image analysis, i.e. image classification.

CNNs have a similar structure to a Fully-Connected Neural Network, with differences in the representation and function of each layer. The core ideas behind CNNs include local perceptive field, weight sharing and sub-sampling. The visual process of a CNN is shown in Figure 2.11.

Different Layers in CNN

1. Input layer

   The input layer pre-processes the data input so that it is standardised. For example, because CNN uses gradient descent optimisation, images pixel values need to be normalised from 0 - 255 to 0 - 1. This will help to improve the efficiency and performance of the network.

2. Convolution layer

   The convolution layer is the core layer which provides feature extraction through applying convolution operations using kernels [20]. Unlike the Sobel operator which uses filters with fixed parameters to extract edge information through convolution, the parameters in the convolution layer of CNN is learnable [23]. The parameters are updated by back-propagation to better extract features. The output will be the input for the next layer.

3. Activation layer

   An activation function is applied to outputs of the convolution layer. The purpose of an activation layer is to give the outputs more non-linearity. This allows the model to classify more complicated data. The commonly used ones include the sigmoid function and ReLU (Rectified Linear Unit) [23].

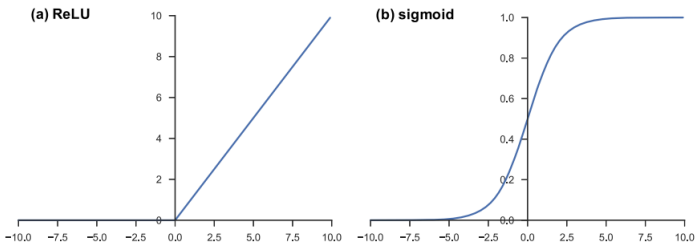


Figure 2.9: Activation functions: (a) ReLU, (b) sigmoid. [20]

4. Pooling layer

   The pooling layer sub-samples the output from the convolution layer to reduce the image size while preserving the important features of the image. This also helps to reduce the computation if it is to be followed by another convolution layer. Max-pooling is commonly used for CNNs. Max pooling divides input images into small patches, outputs the maximum value in each patch, and discards all the other values.

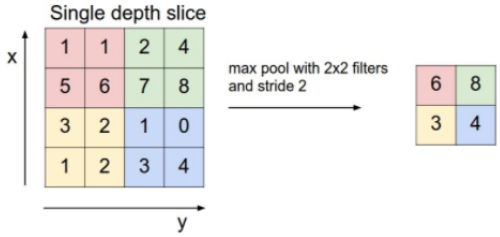


Figure 2.10: Max pooling. [24]

5. Fully-connected layer

   A fully-connected layer takes the high-level feature map from the previous layer and classifies the image into labels. It flattens the outputs from the high-level feature map and feeds them into a fully-connected layer where weights are applied to them. An activation function applied to the multi-class classification task is a softmax function which normalises output real values from the last fully connected layer to target class probabilities, where each value ranges between 0 and 1 and all values sum to 1 [20]. After passing through the activation function, they will pass forward to the output layer, where each neuron will represent each class.

Loss Function

A loss function, sometimes referred to as a cost function, is used to measure the performance of a neural network model. It maps a set of parameter values for the network onto a scalar value which measures how much the predictions differ from the actual output [26]. Loss is inversely proportional to the performance of the model. In other words, the lower the loss, the better and more accurate the model. A commonly used loss function for multi-class classification is cross entropy [20], which is used to estimate the distance between class label distribution and predicted distribution [26].

Gradient Descent

Gradient descent is an algorithm used to optimise the values of parameters of a function that minimises the loss function. It iteratively updates the learnable parameters such as weights and kernels so that the loss function can be minimised. The gradient of loss function decides in which direction the function

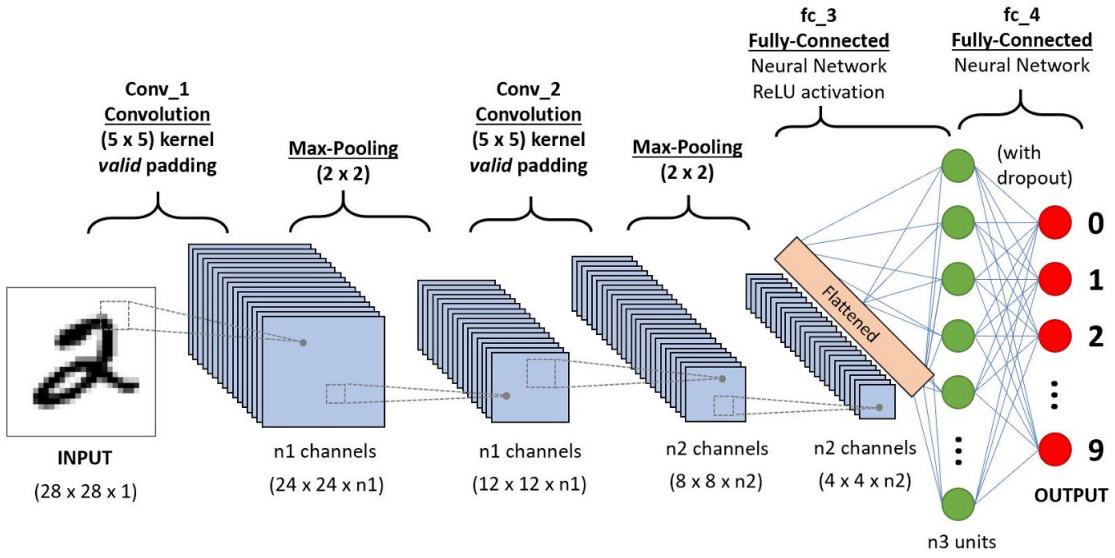


Figure 2.11: The process of a CNN. [22]

will have the biggest rate of increase. A single update of a parameter can be expressed (Equation 2.1) [20], where w stands for each learnable parameter, α stands for a learning rate, and L stands for a loss function.

$$w := w - \alpha \times \frac{\partial L}{\partial w}. \tag{2.1}$$

Therefore the negative direction of gradient is what the learnable parameters will be updated in, so that they go 'downhill'. This is when an important hyper-parameter comes in - learning rate. Learning rate decides how much the parameters will be updated by. Learning rate is one of the important parameters that need to be set before training. If learning rate is too small, the model can take a very long time to reach the minimum of loss. If it is too large, it may overshoot the minimum and may not converge (Figure 2.12).

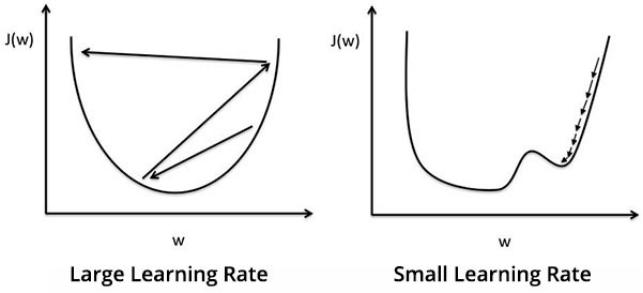


Figure 2.12: Gradient descent: Large learning rate vs Small learning rate. [27]

2.3 TensorFlow & Keras

TensorFlow [29] is an open-source Python library developed by Google Brain team for processing numerical computation. It provides machine learning models and algorithms which make feeding data, training models and predicting result faster and easier to implement.

Keras [31] is a high-level API for building neural networks which is built on top of Tensorflow and provides even more simplified implementation, allowing users to build neural networks without having to

study low-level details of mathematical techniques and optimisation methods in TensorFlow.

2.4 Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG) is an XML-based graphics format used to display a variety of graphics on the web and in other environments, including animation. SVG images are defined in an XML text file, in which they are described not in terms of pixel positions and values but instead, as shapes, numbers and coordinates. This means that SVG images can be scaled, compressed without losing resolution (Figure 2.13).

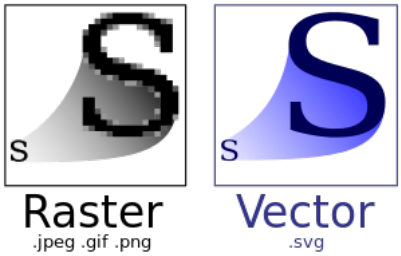


Figure 2.13: SVG vs Bitmap image.

SVG shapes can be drawn with paths. Paths can be used to create lines, curves etc. Combining simple lines and curves can form complicated shapes in SVG. Paths are defined by path commands in SVG. There are 6 types of commands:

- Move To: `M, m`
- Line To: `L, l, H, h, V, v`
- Cubic Bézier Curve: `C, c, S, s`
- Quadratic Bézier Curve: `Q, q, T, t`
- Elliptical Arc Curve: `A, a`
- Close Path: `Z, z`

`M` moves the current point to the coordinate (x,y), serving as the starting position of a line or a curve. `L` draws a line from the current point to the end point specified by two parameters: x and y coordinates of the end point. `z` closes the path by connecting the last point and the initial point. Using these simple commands, we can draw an example illustrating how SVG shapes are formed by paths:

```
<svg width="350.0px" height="350.0px" xmlns="http://www.w3.org/2000/svg">
    <path fill="none" stroke="red"
    d=" M 100.00 200.00 L 200.00 200.00 L 200.00 100.00 L 100.00 100.00 z">
</svg>
```

The resulting SVG shape of the code snippet above is a square drawn with red lines (Figure 2.14).

To draw curves, we can use cubic or quadratic Bézier Curves. Cubic Béziers take in two control points for each point. Therefore, to create a cubic Bézier, three sets of coordinates need to be specified:

```
C x1 y1, x2 y2, x y
```

`C` is the command for cubic curves. `x,y` is the end point of the curve. `x1, y1` is the control point of the starting point and `x2, y2` is the control point of the end point. The control points essentially define the slope of the line starting at each point. The Bézier function will then creates a smooth curve that goes

Figure 2.14: SVG path example.

alone the slope established at the beginning of the line, and the slope at the other end. We can draw an example to show how curves are defined.

```
<svg width="350.0px" height="350.0px" xmlns="http://www.w3.org/2000/svg">
    <path d="M 10 110 C 20 140, 40 140, 50 110 stroke="black" fill="transparent"/>
</svg>
```

The resulting shape is shown in Figure 2.15. The red dots and lines are slopes formed by joining end points and control points together. They are for illustrative purpose only. The black curve is the shape formed by the command.

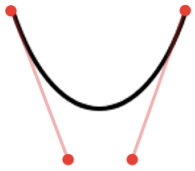


Figure 2.15: SVG cubic curve example.

2.5 HTML & CSS & JavaScript

2.5.1 HTML

HTML is short for 'HyperText Markup Language'. It is the fundamental building block for the Web. It defines the structure and content of the web pages. HTML consists of a variety of elements which give semantics and formatting to different parts of the documents. Elements are enclosed in tags, which open with `<elementName>` and end with `</elementName>`. The following code example describes an HTML document containing a button element in the body.

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <button>I am a button</button>
    </body>
</html>
```

This HTML document if displayed in a browser, will only display a button in the top-left corner of the web page as shown in Figure 2.16.

Figure 2.16: HTML example.

2.5.2 CSS

CSS (Cascading Style Sheets) are used to style or format the web page by selecting elements in HTML documents and giving them different attributes. CSS defines the presentation of a web page, i.e. where elements should be displayed, what colour should they have etc. CSS uses selectors to point to the HTML element that we wish to style and a declaration block to give attributes to properties. An example is shown in Figure 2.17. Taking the previous example, we can give a style to the button, such as a blue

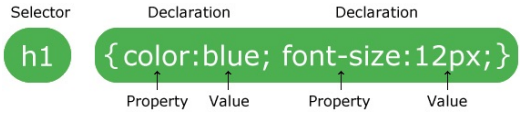


Figure 2.17: Selector example. [25]

colour.

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            button {
              background-color: blue;
            }
        </style>
    </head>
    <body>
        <button>I am a button</button>
    </body>
</html>
```

The resulting web page will display a blue button as shown in Figure 2.18.

Figure 2.18: CSS example.

2.5.3 JavaScript

JavaScript is a high-level programming language which gives a web page behaviours. It allows us to have an interactive experience with web pages. The Object-Oriented nature of JavaScript is similar to Java, but it is a light-weighted and just-in-time compiled programming language. Together with HTML and CSS, they form the core technologies in the World Wide Web we see today.

A typical example of using JavaScript is responding to events. An HTML event is something that happens to an element or something the user does (i.e. a mouse click). Taking the previous example, we can add an event to the button when it is clicked with a cursor.

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <button onclick="alert('Hello world!')" >I am a button</button>
    </body>
</html>
```

When the code snippet above is run in a browser, an alert from the browser will come out when the button is clicked (Figure 2.19).

Figure 2.19: JavaScript example

2.6 Flask Server

Flask is a micro-web framework written in Python. It requires little to no dependencies to external libraries. It creates a Python back-end which can connect to the web front-end through the HTTP protocols. As a light-weight framework, it is a quick way for developers to get started with building a server for a web application.

A minimal example [31]:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

First the Flask class needs to be imported. An instance of the Flask class will be the Web Server Gateway Interface application. Next, an instance of the class is created. `__name__` is a built-in variable in Python which tells the name of the current module (i.e. if it is imported). A `route()` decorator informs Flask which URL should trigger the function. This function, which is given a name, will be used to generate URLs for this particular function. The function `hello_world` will return the message 'Hello, World!' and display in the user's browser.

To run the application, the file should be saved and given a name such as `hello.py`. Running the following command in terminal under the directory which contains `hello.py`:

```
$ export FLASK_APP=hello.py
$ flask run
```

A URL will prompt up, suggesting that the server is running on this URL:

```
* Running on http://127.0.0.1:5000/
```

The webpage associated with this URL will display: Hello, World!

2.7 Intersection Over Union

Intersection Over Union (IOU) is an evaluation metric often used to evaluate the performance of object detection in machine learning or image processing techniques. Typically, when detecting an object on an image, a bounding box will be produced to point out where the detector thinks the target object is on the image. This bounding box is called a 'predicted bounding box'. The manually labelled bounding box which describes the precise location and size of the object is called a 'ground-truth bounding box'. An example is shown in Figure 2.20, where the green bounding box is the ground-truth bounding box which represents the precise size and location of the car. The red bounding box is the predicted bounding box, which is the prediction made by the detector and may not be accurate.

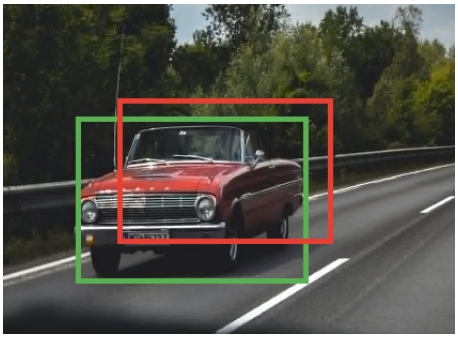


Figure 2.20: Ground-truth bounding box (green) and predicted bounding box (red). Image source: [33]

IOU measures how well the predicted bounding box matches the ground-truth bounding box, both in terms of size and location. It is calculated by:

$$\text{IOU} = \frac{\text{Area Of Overlap}}{\text{Area Of Union}}. \tag{2.2}$$

$$\text{Area Of Union} = \text{Area Of Ground-Truth Box} + \text{Area Of Predicted Box} - \text{Area Of Overlap}. \tag{2.3}$$

The score of IOU is a number between 0 and 1. The higher the score, the better the two boxes match. Figure 2.21 shows some example of IOU scores.

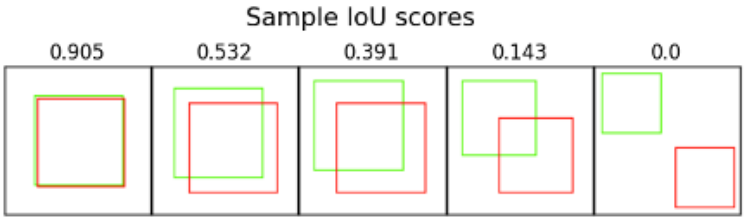


Figure 2.21: Example of IOU scores. [34]

Chapter 3

Implementation Planning

This chapter will provide a plan on what should be implemented and how to implement them in this project.

Based on problem analysis and background knowledge of Chinese characters, we have learned that we need to encourage learners to write actual characters which they can see, instead of only showing the grey strokes (Figure 1.2). For a Chinese character to be correct, it has to follow the correct stroke order. Each stroke must be of the correct type and be located in the correct position as well as have the correct relationships with other strokes. Therefore, the plan is to have an application which can complete the following objectives:

1. Provide an interface where users can select the character they wish to practice

   The interface should clarify the purpose of the application and how to use the application. It should also allow users to input the character they wish to practise.

2. Create a canvas for users to write on

   After receiving the character, the application should create a drawing canvas where users can write the character stroke by stroke. The drawing interface should also allow users to undo strokes and clear canvas.

3. Classify the user stroke input to determine if it is correct

   A Convolutional Neural Network will be used to classify each stroke drawn onto the canvas. If the classification result of user input does not match what is defined in the database, the application should report the error.

4. Determine if a stroke is of the correct size and is put in the correct position

   When each stroke is drawn, the application should check the size and position of the stroke against the correct stroke defined in the database. If the size or position do not match what is correct, the application should report error and provide feedback.

5. Determine if a stroke has correct spatial relationships with every other stroke

   Another Convolutional Neural Network will be applied to classify stroke relationships. The application should extract every pair of strokes and feed them into a trained model to decide what relationship the two strokes have. All pairs of strokes should conform to the correct relationships defined in the database. The application should report any pair of strokes that does not match the correct relationship.

6. Provide visual guidance to user when requested

The application should be able to provide hints for each stroke or hints for the full character on the canvas if requested by the user. The size and position of the hint shown on the canvas should be adjusted based on the first stroke of user input, because the first stroke will normally decide where the whole character will be positioned and how big the overall character should be.

7. Report error and feedback to users

   A feedback box should be provided below the canvas to provide instant feedback after each stroke has been completed and analysed by the application.

Chapter 4

Implementation

This chapter is an expanded explanation on the implementation planning, outlining the process of the implementation and how each step is achieved in detail.

4.1 Platform Selection

Platform selection is an important part of this project, as it decides how users will interact with the application and whether the application can best achieve its purpose. The aim of this application is to enable users to write the characters on a screen and get feedback based on their writing. Therefore, the way users input the character is crucial to this project. Using a mouse to write on a personal computer is not an ideal way for practising Chinese hand-writing, as it does not simulate writing with our hands and the mouse can sometimes be difficult to control. Therefore, developing the application as a piece of computer software is not an ideal option.

A mobile application is what comes to mind next. Using touch screens on mobile devices, users can write with fingers or capacitive stylus pens which better simulate the scenario of writing with pen and paper. Nowadays, the majority of mobile applications are developed on either Android or iOS. Given the limited time on this project, I could only choose one platform to develop the application for. Choosing one platform means that the application won't be accessible for users using other mobile operating systems. This application is light-weight, as it provides specific functionality only - detecting writing errors and providing feedback. Having users download a piece of software for a light-weight application and excluding users from different operating systems is unnecessary, especially when cross-platform implementation is available.

I finally settled on developing a web application, for the following reasons:

- All mobile devices can access the application as long as there is a browser and an internet connection
- Users can also access the application from browsers on PCs or laptops
- Users don't need to download anything, making it easy to access
- Development on a single web application will suffice, because most browsers across different operating systems support the same HTML standards

At the same time, developing a web application poses a disadvantage, which is that users can only access the application when there is an internet connection.

This issue is a trade-off between developing a mobile application and a web application. In a mobile application the programme for implementing the functionality and all the files which assist the programme can be downloaded onto the device, making offline use possible. A web application is freely accessible on any platform but requires internet access. On this issue, I chose web application over mobile application.

As the implementation progresses, and even after the implementation has been completed, more and more characters will be added into the programme to allow a wider range of character practice. If it was a offline mobile application, constant update on the database would be required. Whereas with a web application, newly added characters will be available on the server as soon as they are ready. A URL to the web page is all the users need. In this respect together with the convenience of accessing web application, building a web application instead of a mobile one is justifiable.

4.2 Data Collection

4.2.1 Correct Template Characters

The detection mechanism requires a correct template(a correct version of writing) for each character. Each stroke drawn by users will be compared against the corresponding stroke in the template character. This means for each character, there must be a correct template which can be broken down into correct strokes. In addition, Kai Shu is widely used as the standard Chinese handwriting style, which would be most suitable for Chinese learners to start with.

To summarise, for this project I need template characters that are written in Kai Shu style and can be broken down into individual strokes.

Arphic PL KaitiM GB, a free Chinese font released by Arphic Technology under a permissive license in 1999, provides the glyphs for more than 3,000 common Chinese characters in Kai Shu style. Skishore published a project named: *Make Me a Hanzi* on Github which aims to provide free, open-source Chinese character data [37]. The project provides glyphs data derived from Arphic PL KaitiM GB, in which a JSON object stores all information about a character's glyph.

Each JSON object contains three keys and their corresponding values. The first key is the character glyph, showing which character this object is for. The second key is strokes. Stroke data of a character is represented in a list of SVG path data, with each defining a stroke of this character. The SVG paths in the list are ordered by the proper stroke order. The third key is the medians, containing a list of medians for each stroke. Medians are useful for defining the orientation in which a stroke is written.

The below code is a JSON object that defines the character 大 (big):

```
{"character":"大",
"strokes":["M 494 476 Q 542 485 795 501 Q 817 502 822 512 Q 826 525 808 540
           Q 750 580 707 569 Q 631 550 500 522 L 436 509 Q 331 490 213 469
           Q 189 465 208 447 Q 241 420 294 432 Q 357 453 431 465 L 494 476
           Z","M 487 437 Q 491 456 494 476 L 500 522 Q 510 711 528 763 Q
           534 776 523 786 Q 501 805 459 822 Q 434 832 414 825 Q 390 816
           410 796 Q 444 762 444 726 Q 445 602 436 509 L 431 465 Q 398 275
           310 179 Q 303 173 297 166 Q 251 118 148 55 Q 133 48 130 43 Q 124
           36 144 34 Q 195 34 300 104 Q 385 173 414 218 Q 444 266 480 396
           L 487 437 Z","M 480 396 Q 501 357 575 245 Q 657 124 718 56 Q 746
           22 774 22 Q 856 28 928 32 Q 959 33 959 41 Q 960 50 927 66 Q 753
           144 719 174 Q 614 267 500 419 Q 493 429 487 437 C 469 461 465
           422 480 396 Z"],
"medians":[[[210,458],[268,453],[514,503],[719,534],[770,529],[810,517]],
          [[416,810],[444,799],[482,759],[469,518],[448,394],[426,320],
          [386,231],[361,196],[307,140],[202,67],[138,41]],[[486,430],
          [500,393],[576,284],[660,182],[722,118],[774,77],[953,42]]]}
```

*Note: In the above example lines are separated for better readability. In the original data, they are in the same line. Each line defines an object.

I wrote a python programme to parse the JSON objects so that each SVG path of a stroke in the list can form an SVG image which represents the glyph of this stroke. An example in Figure 4.1 shows the individual strokes after parsing the JSON object of the character 大 and extracting each stroke to form SVG images.

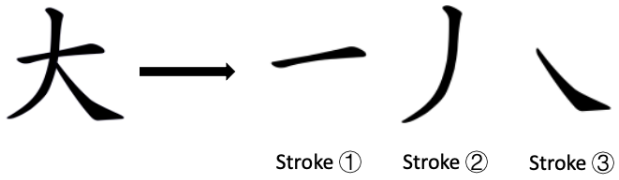


Figure 4.1: Example of parsing the JSON object of 大 (big).

All SVG images containing stroke glyphs were then transformed into BMP images, making pixel manipulation on the images possible for the detection process later. They were also transformed into PNG images, which will be used for hint display.

4.2.2 Stroke Samples

To train a Convolutional Neural Network that can classify the type of each stroke, samples for each one of the 32 hand-written stroke types will need to be collected. Unfortunately, I could not find samples that have been created by other people and shared on the internet. This meant that I had to collect the samples myself.

I collected the stroke samples from 5 Chinese native speakers, including myself. Each person contributed 20 samples of each stroke types, amounting to 100 samples for each stroke and 640 samples from each person (20 samples × 32 stroke types). The samples were collected with an iPad and a stylus pen on a drawing application named Procreate. Canvas size was set to 300 and each pen width was 12px (Figure 4.2). Each stroke within the same type was written with slight intentional variation while conforming to contributor's writing style (Figure 4.3). This is to capture as much variety as possible because a type of stroke can appear in many similar shapes in different characters. For example, 木 (wood), 林 (woods) and 森 (forest) all contain the stroke 丿 (Piě), but they appear in slightly different shapes and sizes in each character.

Figure 4.2: A sample for stroke type 乛 (Héng Zhé Zhé Zhé Gōu 横折折折钩).

Due to the variety of the same stroke type in different characters, all samples collected need to be scaled up and down, compressed horizontally and vertically and slightly rotated. This is to increase the sample size so that the model can classify as much variation as possible. I wrote a python programme that implements the scaling and variations.

Each sample was scaled by 0.6, 0.8, 1.2 and 1.4 times of the original, and each sample was horizontally scaled by 0.9, 0.8 times of the original. The same operations were then applied to the sample vertically. Finally, each sample was rotated by -5 degrees and 5 degrees. In total, there were 10 variations of each

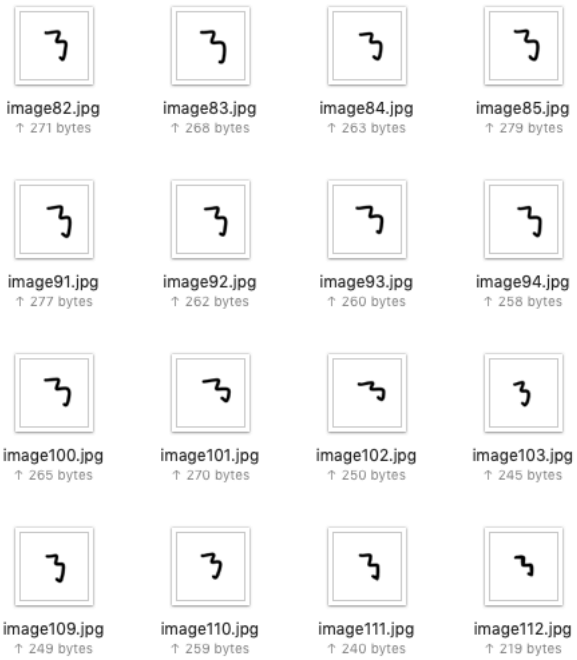


Figure 4.3: Samples of stroke type ㇆ (Héng Zhé Zhé Zhé Gōu 横折折折钩).

sample. Each stroke type had 100 samples collected from 5 people. Therefore, after variation (Figure 4.4), sample size for each stroke type grew up to 1,100 (100 samples × 10 variations + 100 samples).

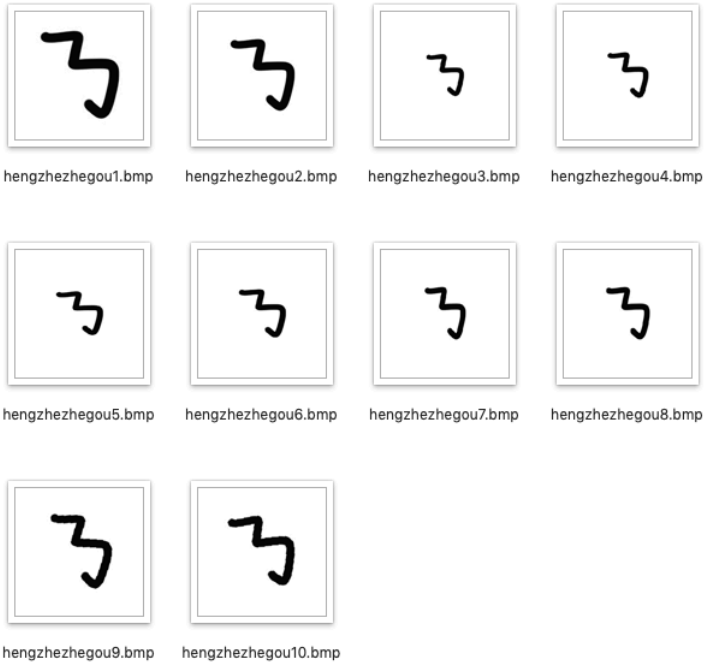


Figure 4.4: Resulting samples after varying the shape an original sample of ㇆ (Héng Zhé Zhé Zhé Gōu 横折折折钩).

4.2.3 Stroke Relationship Samples

Each pair of stroke in a character follows one type of relationship, namely detach, connect or intersect. To train a CNN model that classifies stroke relationships, stroke relationship samples also need to be

collected. Such samples are also not available on the internet, therefore need to be created by myself. Each sample should feature one of the three types of stroke relationships, while having various stroke types in them.

The samples were generated by decomposing a hand-written Chinese character into individual strokes and combining every pair of strokes to form a sample. In this way, the samples generated can simulate the real stroke pairs which will need to be classified when the application is put into use. The way this was achieved was by creating an HTML5 Canvas and a python back-end programme which allowed me to write characters on the canvas and automatically generate stroke pairs as samples. After the samples were created, I hand-labelled them individually as shown in Figure 4.5.

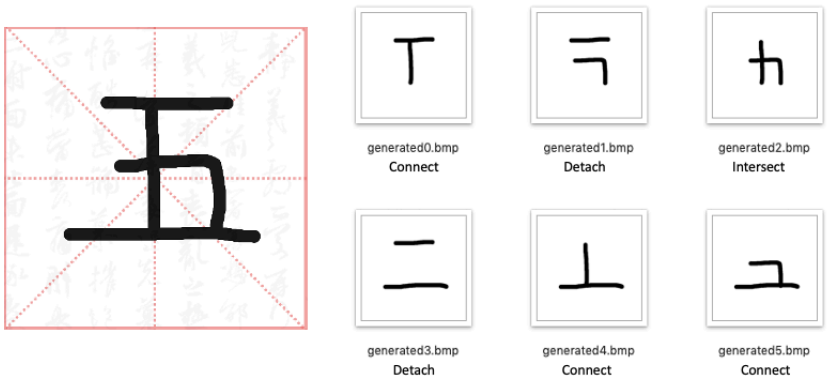


Figure 4.5: Generating relationship samples from 五.

In this way, I created around 2700 samples and labelled them individually. Each sample was then rotated by 90, 180 and 270 degrees to expand the sample size, the final dataset size is shown in Table 4.1.

Relationship type	**Sample size**
Detach	3,414
Intersect	3,700
Connect	3,625
Total	**10,739**

Table 4.1: Dataset of relationship types.

4.2.4 Correct Stroke Type Data and Correct Stroke Relationship Data

Correct stroke type data and correct stroke relationship data are used for checking if the strokes inputs from users are of the correct stroke types and if they conform to the correct stroke relationships. Each stroke written by the user will be classified by a CNN model to determine the stroke type and the result will be compared with the correct stroke type data. Each stroke written will also be paired with previously written strokes and be classified by another CNN model to determine the stroke relationship. The result will be checked against the correct stroke relationship data.

Correct Stroke Type Data

The correct stroke type data should contain the correct stroke types in the correct stroke order. I labelled each stroke type with a unique number(Table 4.2), which greatly simplified the representation of each stroke type.

No.	Stroke	name	No.	Stroke	name
0	丶	Diǎn	16	𠃌	Héng Zhé Zhé Piě
1	一	Héng	17	𠃍	Héng Zhé Zhé Zhé Gōu
2	丨	Shù	18	𠂉	Shù Tí
3	丿	Piě	19	𠃊	Shù Zhé
4	乀	Nà	20	𠃎	Shù Zhé Piě
5	乛	Héng Zhé	21	亅	Shù Gōu
6	乚	Héng Zhé Gōu	22	𠃋	Shù Wān
7	乛	Héng Gōu	23	𠃏	Shù Wān Gōu
8	𠂆	Héng Piě	24	𠃌	Shù Zhé Zhé
9	𠂇	Tí	25	𠃍	Shù Zhé Zhé Gōu
10	𠂈	Héng Zhé Tí	26	𠂊	Piě Diǎn
11	𠂉	Héng Zhé Wān	27	𠂋	Piě Zhé
12	𠂊	Héng Zhé Zhé	28	𠂌	Xié Gōu
13	𠂋	Héng Xié Gōu	29	𠂍	Wān Gōu
14	𠂌	Héng Zhé Wān Gōu	30	𠂎	Wò Gōu
15	𠂍	Héng Piě Wān Gōu	31	𠂏	Héng Zhé Zhé Zhé

Table 4.2: Table of stroke types labelled with numbers.

Using Table 4.2, the correct stroke type data can be represented by a list of integers, separate with ','. Taking the character 五 (five) shown in Figure 4.5 as an example, its correct stroke types in correct order is:

```
Héng, Shù, Héng Zhé, Héng
```

This can be represented by:

```
1, 2, 5, 1
```

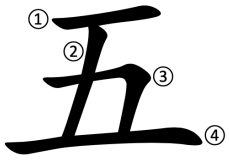


Figure 4.6: Correct stroke order of 五 (five).

Correct Stroke Relationship Data

In a character, each stroke has a unique relationship with every other stroke. In the correct stroke relationship data of a character, such relationships must be defined between all pairs of strokes in the character.

A matrix-like data structure is used to represent the stroke relationships in a character. In a matrix R which represents the stroke relationship of a character, the column size is equal to the row size, both representing the number of strokes in the character. $R_{x,y}$ is the element in the xth row and the yth column. It represents the relationship between the xth stroke and the yth stroke in the character. Each relationship type is represented by a number, as shown in Table 4.3. If $x = y$, which are the elements in the diagonal line, $R_{x,y} = 0$.

Taking 五 as an example, the correct stroke order is: Héng - Shù - Héng Zhé - Héng as Figure 4.6 shows. The correct relationship can be represented as Figure 4.7.

Relationship type	Representation
Detach	1
Connect	2
Intersect	3
Diagonal elements	0

Table 4.3: Relationship types.

$$\begin{bmatrix} 0 & 2 & 1 & 1 \\ 2 & 0 & 3 & 2 \\ 1 & 3 & 0 & 2 \\ 1 & 2 & 2 & 0 \end{bmatrix}$$

Figure 4.7: Relationship matrix of 五 (five).

Combining The Two

Correct stroke type data and correct relationship data form the information about a character which the user input will be checked against. They are put together to form a text file, separated by a new line. The file is named after the character they represent. In the text file, relationship matrix is flattened because using the correct stroke type data we can get the total stroke number and reconstruct the matrix if needed. As an example, 五.txt contains the following:

```
1,2,5,1
0,2,1,1,2,0,3,2,1,3,0,2,1,2,2,0
```

4.3 Model Training

In the error detection process, there are two steps that require classification using machine learning techniques. They are:

1. Stroke type classification: Used to classify the type of stroke that the users write on the canvas.
2. Stroke relationship classification: Used to classify the relationship between every pair of strokes that the users write on the canvas.

The machine learning technique that I adopted for classification is CNN(Convolutional Neural Network), with the reasons being:

1. Using ideas behind local receptive fields, weight sharing and sub-sampling, CNN can achieve some degree of shift and deformation invariance[28]. This means that the location of the target object is less important, which implies that the stroke types can be classified regardless of their position on the canvas.
2. CNN is computationally more efficient when it comes to classifying images, compared with its predecessor - Fully Connected Network(FNN) by extracting features using convolution, sub-sampling to reduce size of images and weight sharing to reduce number of parameters.

4.3.1 Data Pre-Processing

Both models were trained in the same way. Before training, sample images were first compressed to a smaller size (50×50). When read in, they were converted to grey-scale images. This helps to reduce the

time for training. The samples were then converted into array and paired with a label (a stroke type). This produces pre-processed training samples. An example code snippet is as follows:

```
def loadfile(files, label):
    train = []
    for file_name in files:
        image = Image.open(label + '/' + file_name)
        image = image.convert("L")
        train_data = np.array(image)
        train.append([train_data, label])
    return train
```

When all samples of different labels have been read in, they are randomly shuffled with a pre-set random seed. By randomly shuffling data samples, it ensures that changes on the model made by each sample are independent, without being biased because of the previous samples having the same label. Setting the seed of the random number generator ensures that the result of random shuffle creates the same order of samples every time, which makes the results reproducible. This is important when evaluating a model's performance. Random seed gives reproducibility to the data samples which eliminates the possibility that the changes in performance are brought in by sample orders. This helps us to better understand the effect of changes in model configurations.

4.3.2 Sample Split

The dataset is split into 3 parts, training samples, validation samples and test samples (Figure 4.8). Training samples are a set of data used to train the model. Validation samples are a set of data separate from the training samples that is used to validate our model during training. This process helps us by providing information that we can use to adjust the hyper-parameters during training by methods such as early stopping. Test samples are a set of data that the model has never seen before and is used to evaluate the performance of the model without any bias. The ratio between the training, validation and test samples is set to be $3 : 2 : 1$.

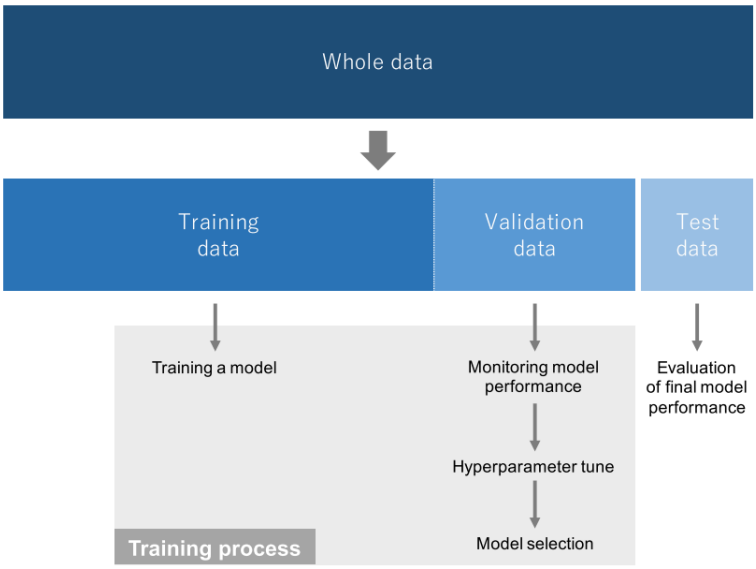


Figure 4.8: Sample split. [20]

4.3.3 Model Training

The model training method takes experience from [32], which introduced a way of constructing a CNN model for classifying hand-written digits.

The model is build with Keras in TensorFlow. The structure of model is shown in Figure 4.9. The model consists of two convolution layers, both with a 3×3 kernel and followed by a 3×3 max pooling layer. The training process runs through 20 epochs, meaning the entire training dataset is run through the neural network 20 times. The reason for this is that, as mentioned in Gradient Descent (section 2.2.2), the optimisation algorithm is an iterative process which means we need to run the dataset through the model multiple times to achieve the optimal result. The optimisation algorithm used is Adam, with Learning rate set to its default.

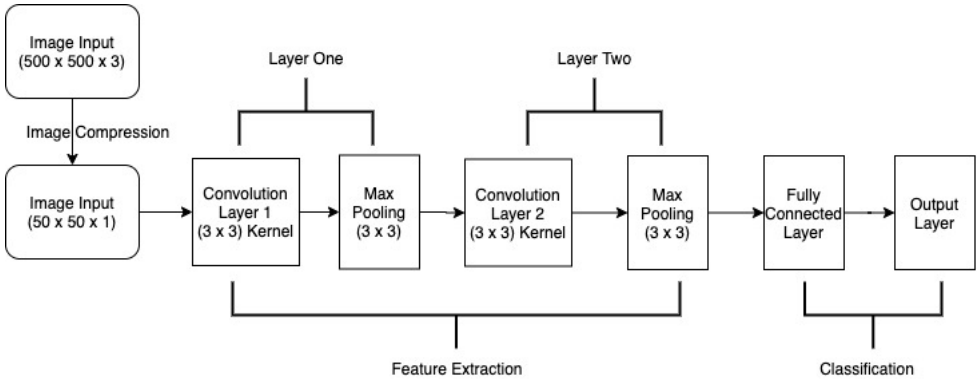


Figure 4.9: Model structure.

4.3.4 Results

After applying the CNN on the two collected datasets, models for stroke type classification and stroke relationship classification were obtained, with training results shown in Figure 4.10 and Figure 4.11 respectively.

From the figures we can see that they both show similar trends. As the model runs through more and more epochs, validation loss keeps decreasing until it reaches a stable level. Validation accuracy keeps increasing until it reaches a stable level. The test accuracy on stroke type classification and stroke relationship classification reached 0.99 and 0.95 respectively.

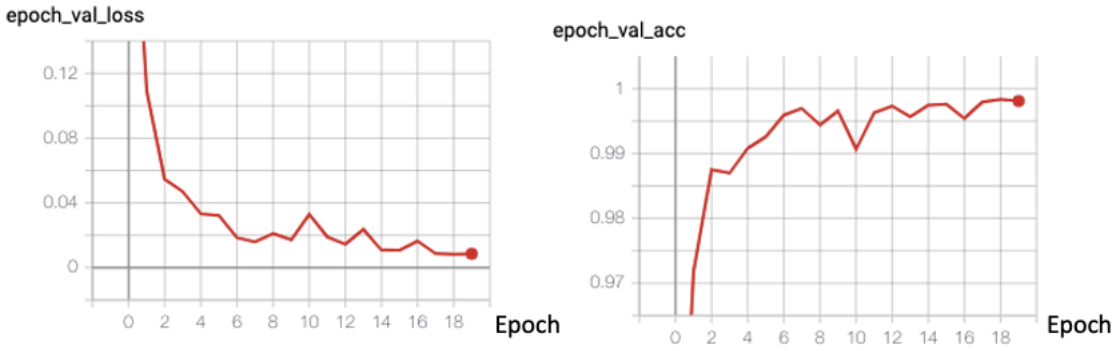


Figure 4.10: Training result of the CNN model for stroke type classification.

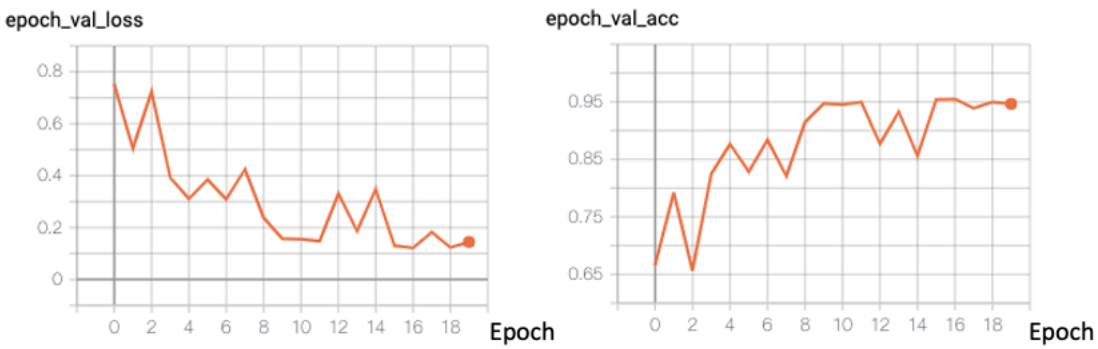


Figure 4.11: Training result of the CNN model for stroke relationship classification.

4.4 Website Back-end Development

Back-end is also known as the server side of an application. In this application, it is where the user input is received and where the error detection process takes place. Based on the detection result, feedback will be returned from the server to client(the web pages). The back-end application is implemented using Python.

4.4.1 Checking If The Character Exists

The first function that needs to be implemented is checking whether the data(character) requested by users exists in the server files. For the detection process to run on a character, we need the correct character stroke templates, correct stroke type and stroke relationship data. All template images of a character are stored in a folder named after the character, and the correct stroke type data, stroke order data are stored in a text file also named after the character. The `check_exist()` function will run through all file names to find the file name that match the user input. If failed, this would mean the character data requested have not been collected, and users will see a prompt from the web browser saying: This character is not collected in the database so far.

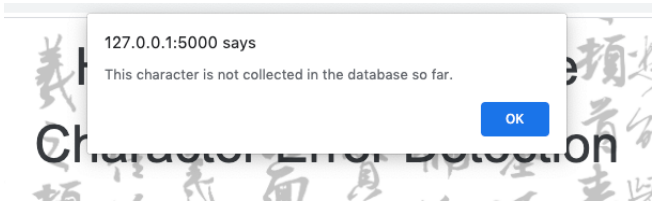


Figure 4.12: A prompt to user.

4.4.2 Loading Character Information

If the character requested does exist in the server files. The next step is to load them into the program. Correct stroke templates are loaded into a list named `correct_strokes_img` following the correct stroke order. That is to say, index 0 of the list stores the image of first stroke, and index 1 stores the second stroke etc. The correct stroke type and stroke relationship data are stored in a text file. A function named `read_stroke_info` is used to parse the text file so that the correct stroke type data is stored in a list named `correct_strokes` following the correct order, and the correct stroke relationship data is stored in a $n \times n$ numpy array named `correct_relationship` with n being the number of strokes. For example, 廾.txt would produce the following data:

```
correct_strokes: [1,2,5,1]
```

```
correct_relationship: array([[0, 2, 1, 1],
                             [2, 0, 3, 2],
                             [1, 3, 0, 2],
                             [1, 2, 2, 0]])
```

At the same time, an $n \times n$ numpy array named `stroke_relationship` which represents the current stroke relationships will be initialised, with n being the number of strokes. This is the relationship matrix that will be updated each time when the server receives user input.

4.4.3 Detecting The First Stroke

The first stroke from the user input is important to the error detection, because the first stroke will decide where the final character will be located on the canvas, and the size of the final character.

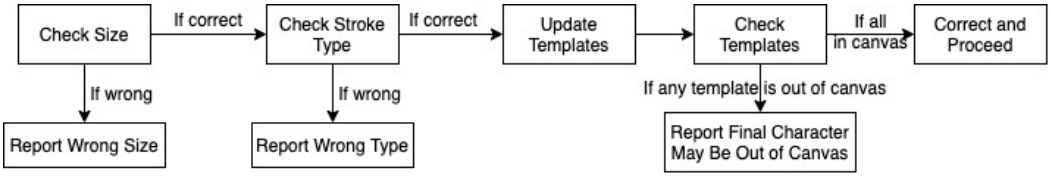


Figure 4.13: Process for detecting the first stroke.

Figure 4.13 outlines the process for detecting the first stroke. When the first stroke is received by the server, the size of it is the first to be checked. The size can neither be too big nor too small. An overly big first stroke may cause the subsequent stroke to go out of canvas (Figure 4.14). If it was too small, the final character may end up being very small, making all strokes to be crammed together when using a fixed pen width (Figure 4.15).

Figure 4.14: Out of canvas character.

Figure 4.15: Overly small character.

Size Checking

Checking the size involves using the template. The bounding box around the actual stroke on the template image decides the size of the correct stroke. The same will be applied on the stroke from the user.

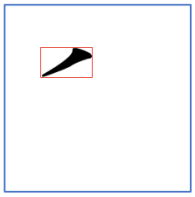


Figure 4.16: Bounding box around the correct template stroke.

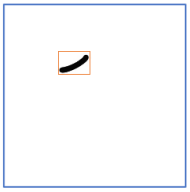


Figure 4.17: Bounding box around the user-drawn stroke.

The general idea for size checking is to take the size ratio between the user-drawn stroke box (Figure 4.17) and the correct template box(Figure 4.16), and define an upper bound and a lower bound on

the ratio. However, this method turned out to be not generalisable on all types of strokes. Taking the first stroke of character 我 as an example, the first stroke 丿 (Piě) is roughly a diagonal curve to the left. If the length of user-drawn Piě is about half the length of the correct template, the size ratio between the user-drawn box and the correct template box would actually be 1:4, meaning that the bounding box size of user-drawn Piě is four times smaller the correct template. This is clearly not true as we know the size of user-drawn Piě is half as long. Therefore, depending on the type of stroke, the size ratios between the bounding boxes are calculated differently. The ways of calculating ratios are listed in Table 4.4.

Stroke	Way of Calculating Ratio
一 Héng	take the ratio between horizontal lengths of bounding boxes
丨 Shù	take the ratio between vertical lengths of bounding boxes
丿 Piě	take the ratio between diagonal lengths of bounding boxes
乀 Nà	take the ratio between diagonal lengths of bounding boxes
㇀ Tí	take the ratio between diagonal lengths of bounding boxes
乛 Héng Gōu	take the ratio between horizontal lengths of bounding boxes
亅 Shù Gōu	take the ratio between vertical lengths of bounding boxes
others	take the ratio between bounding boxes

Table 4.4: Different ways of calculation ratios

If the ratio calculated is between 0.5 and 2.0, which means the user-drawn stroke is not smaller than half of the correct template stroke, and not bigger than twice of the correct template stroke, the size of the user-drawn stroke will be considered correct and will proceed to the next stage.

Checking The Stroke Type

Checking the stroke type will make use of the model that has been trained with different stroke type samples. The model is loaded with the `loadmodel` function imported from `tensorflow.keras.models`. Prediction can be made with `predict` function. A sample code snippet is shown below:

```
model_stroke = load_model('stroke_type_model.h5')
prediction = model_stroke.predict(stroke_image)
```

The prediction is an array of 32 numbers ranging from 0 to 1, each indicating the probability of the image being a certain class. The index which has the highest probability is the predicted class. This is obtained by using `numpy.argmax`. When given an array, it will return the index which contains the highest value. Comparing the result with the first value in `correct_stroke` will indicate if the predicted stroke type matches with the correct stroke type. If they do not match, the user will be notified that the stroke is incorrect and will be asked to redo the stroke and attempt again. If the result matches with the correct stroke type, the detection will continue.

Updating Templates

Once the first stroke is written by the user, the location and the size of the first stroke is fixed. Now the stroke templates must be updated so that the size and position of them match the first stroke. This is done by first resizing all template strokes by a factor, which is the ratio calculated above, and moving all templates by a certain distance to match the first stroke. The distance is decided by the distance between the centre points of two bounding boxes. In Figure 4.18, the red box represents the resized template stroke bounding box, and the orange box represents the user-drawn stroke bounding box. HD and VD are the horizontal and vertical displacement from centre of resized template stroke bounding box to the centre of user-drawn stroke bounding box.

All template strokes are resized individually. Resizing is achieved by using the `thumbnail` function in Pillow - an imaging library in Python. `thumbnail` will resize the image to a size specified by a tuple

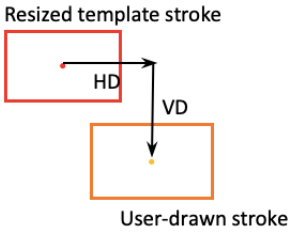


Figure 4.18: Distance between the centres of bounding boxes.

which contains the width and height of the target image. If the resized image is larger than the original image, the extra size will need to be cut off. If it is smaller, the image will be padded with white area using `fill_image` so that the size remains the same as the original image (Figure 4.19). A code snippet is shown below:

```
def scale_image(stroke_image, scaling_factor):
    image_array = np.array(stroke_image)
    img_size = image_array.shape[0]
    new_size = int(img_size * scaling_factor)
    stroke_image.thumbnail((new_size, new_size))
    image_array = np.array(stroke_image)

    if img_size > new_size :
        image_array = fill_image(image_array, 0, img_size - new_size, 0, img_size - new_size)
    if img_size < new_size :
        np.delete(image_array, list(range(img_size, new_size+1)), 1)
        np.delete(image_array, list(range(img_size, new_size+1)), 0)
    return image_array
```

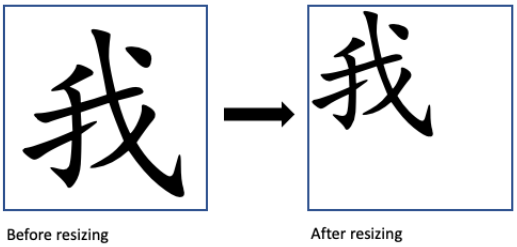


Figure 4.19: Resizing the templates (The resized template strokes are all separate images each containing a stroke. But for illustration purpose they are put on top of each other to form a character in this example).

After a stroke template has been resized, the next step is to move it to where it will match the user-drawn stroke. The distance is determined by the horizontal and vertical displacement between two centres as shown in Figure 4.18. Moving the strokes on the template image is done by moving all black pixel on the image by the displacements. Below is an example code snippet:

```
def move_stroke(img, distance):
    new_canvas = np.zeros((img.shape[0], img.shape[1]))
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i][j] = 0: # black pixel
```

```
                if (i + distance[1]) >= img.shape[1] or (j + distance[0]) >= img.shape[0]:
                    new_canvas[0][0] = -1 # the pixel after moving is out of boundary
                    return new_canvas
                else:
                    new_canvas[i + distance[1]][j + distance[0]] = img[i][j]
    return new_canvas
```

If a pixel location exceeds the boundary of the image after it is moved, process will be terminated and the user will be notified that the updated template is out of boundary.

If all template strokes land within the boundary after moving, `correct_strokes_img` will be updated with the updated templates (Figure 4.20).

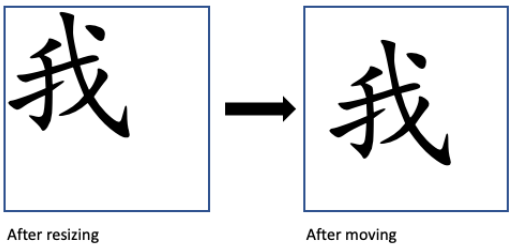


Figure 4.20: Moving the templates after resizing (The moved template strokes are all separate images each containing a stroke. But for illustration purpose they are put on top of each other to form a character in this example).

If the templates are resized and moved successfully, the detection process on the first stroke is complete, and users will be notified to proceed to the second stroke. In addition, upon finishing the first stroke, the correct character template will be shown on the canvas as a hint to guide the user where the character should be and how big or small it should be based on the first stroke.

4.4.4 Detecting Subsequent Strokes

The detection process on the subsequent strokes are similar to the first stroke, with the main difference being that stroke relationships need to be determined between current stroke and all previous strokes and the location of each stroke must match the location of the corresponding template stroke. A flow diagram which outlines the process is shown in Figure 4.21.

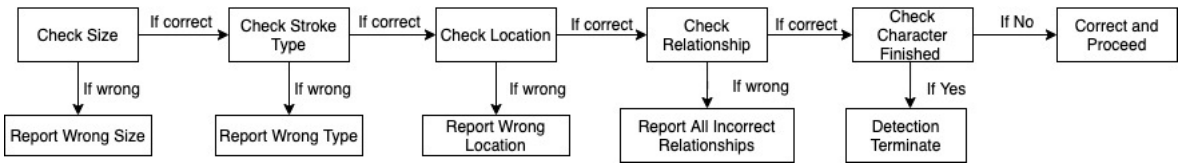


Figure 4.21: Process for detecting the subsequent strokes.

Checking Size And Stroke Type

These two steps are implemented in the same way as the detection on the first stroke, as described in section 4.4.3.

Checking Location

In section 4.4.3, template strokes have been adjusted based on the size and position of the first stroke. Now all the location of the subsequent strokes must match their corresponding updated template stroke.

In this way it can be made sure that all strokes written conform to the correct spatial position and the resulting character will not become out-of-shape.

Location checking makes use of the IOU score between the correct template stroke box and the user-drawn stroke box, with the equation listed in Equation 2.2. The threshold of IOU score is set at 0.5 to allow a reasonable margin of imperfection, which means that the stroke drawn by the user does not need to perfectly match the position and size of the template. If the IOU score is below 0.5, users will be notified and provided with feedback on how to improve.

When the user-drawn stroke fail to pass the location check, based on where its centre is in relation to the correct stroke template, different feedback will be given. Figure 4.22 shows 8 different regions where the the centre of a user-drawn stroke bounding box may land in relation to the correct template stroke bounding box in the middle. Different feedback based on where the centre lands is shown in Table 4.5.

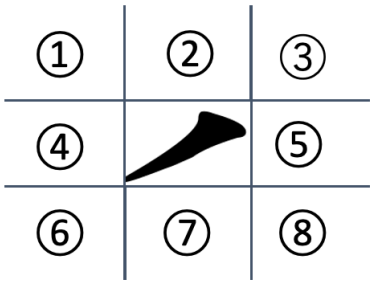


Figure 4.22: Different regions where the centre of a user-drawn stroke bounding box may land.

Region	Feedback
1	Try moving towards bottom-right corner.
2	Try moving it down a bit.
3	Try moving towards bottom-left corner.
4	Try moving towards right a bit.
5	Try moving towards left a bit.
6	Try moving towards top-right corner.
7	Try moving it up a bit.
8	Try moving it towards top-left corner.

Table 4.5: Different feedback based on the where the centre of user-drawn bounding box lands.

Users will be encouraged to attempt again until the size and location check are passed, then the detection will move onto the next stage.

Checking Relationship (CNN Model)

Each pair of stroke in a character follows a specific relationship - detach, connect or intersect. Starting from the second stroke, all strokes written need to pass the relationship test between itself and all the previously written strokes.

A stroke will be combined with all previously written strokes and each stroke pair will be fed into the stroke relationship classification model trained in section 4.3. For example, in Figure 4.23, when the user writes the second stroke in 扎 (marked in red), the stroke image will be combined with the stroke image of the first stroke to form a pair. Next, this pair of strokes will be classified by the model.

Each prediction made by the classifier will update the `stroke_relationship` array. In Figure 4.23, if classified correctly, the relationship between the strokes in this pair should be connect, which is labelled as 2. Thus, the `stroke_relationship` array should be updated from Figure 4.24 to Figure 4.25. This means that the first stroke and the second stroke have a relationship of connect.

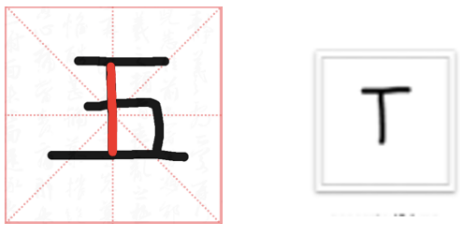


Figure 4.23: Combining strokes.

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.24: Relationship matrix of before the second stroke.

$$\begin{bmatrix} 0 & 2 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.25: Relationship matrix of after the second stroke

The resulting array will be compared with the `correct_relationship` array. Since the character has not been finished, we only need to check if the currently finished strokes match with the `correct_relationship` array. In this case, only the intersection between the first two rows and the first two columns in Figure 4.25 will be compared with `correct_relationship` array. A example code snippet is shown below:

```
def check_relationship(true_relationship, current_relation, current_stroke_num):
    result = []
    for i in range(current_stroke_num):
        for j in range(current_stroke_num):
            if current_relation[i][j] != true_relationship[i][j]:
                result.append([i, j])
    result = remove_duplicate(result)
return result
```

If a relationship between two strokes does not match with the correct relationship, the pair numbers will be coupled and appended to a `result` list. Because the relationship array is symmetrical along the diagonal line, there will be duplicated couples such as `[1, 2] and [2, 1]`. The duplicated items in the list are then removed by `remove_duplicate` function.

The pair numbers are collected to give feedback to users, indicating which pairs of strokes do not match the correct relationships.

Problem With Stroke Relationship Classification Model

After doing a few trials on classifying strokes relationship using the model with new samples from other people's writing, I noticed that while it classifies relationships reasonably correctly, there were some stroke relationships that it always failed to classify, especially in the characters that have more strokes (Figure 4.26).

The model accuracy was 0.95, which suggests that it is unlikely for the model to make incorrect predictions, which is contrary to the fact. Some plausible reasons are listed below:

1. To determine if all the strokes in a character follow the correct stroke relationships, each stroke needs to be checked with every other stroke in the character. If a character has 10 strokes, 100 stroke pairs could be formed. However, the repeated ones and the ones which have a stroke paired

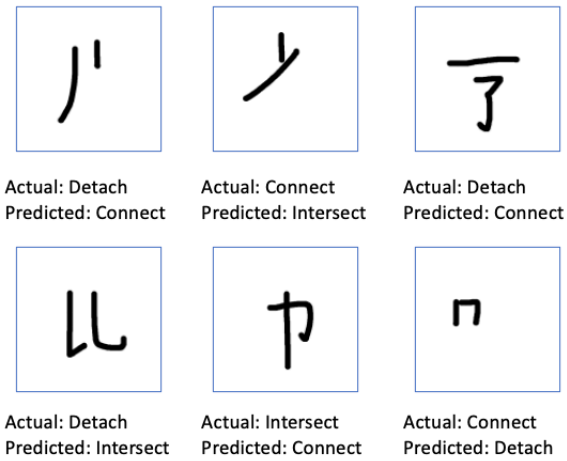


Figure 4.26: Mis-classified strokes.

with itself need to be removed. There will be 45 pairs of strokes left (100 pairs - 10 sample stroke pairs - 45 repeated pairs), whose relationship need to be determined. An accuracy of 0.95 might seem reasonably good, but the probability of the model classifying 45 pairs correctly is 0.95^{45}, which is around 0.1. It is likely that using a CNN model is not a good solution to this problem, because even if the accuracy is raised to 0.99, 0.99^{45} is around 0.63, which is still not a satisfactory result.

2. The training samples are created by decomposing just over 100 different characters, which may not contain all the possible stroke combinations. Therefore, when classifying uncommon stroke pairs, the model is unable to make correct predictions. This could be a lack of samples issue.

Checking Relationship (Second Method)

Stroke relationship checking is crucial to the detection process. Given the issues with the relationship classification model, I proposed another method for determining stroke relationship. The second method will be relying on the actual geometry of the stroke pairs. I will talk about the methods for determining detach, connect and intersect individually.

To determine if two strokes detach from each other, we can add up the corresponding pixel values in the two stroke images and determine the relationship using the resulting values. Assuming that 0 represents white pixels and 1 represents black pixels, the pixel positions where one black pixel overlap another will result in an addition of 2, this means that the two strokes overlap with each other (Figure 4.27). If after addition, there are no values of 2, we can conclude that the two strokes detach from each other (Figure 4.28).

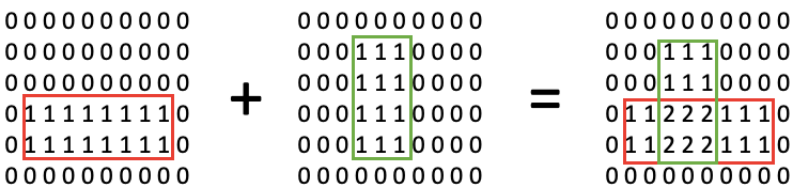


Figure 4.27: An example of two lines that overlap with each other.

The challenges lies in how to determine whether two strokes connect or intersect with each other. Judging from the area where they overlap cannot give any clue to the question. However, the area around where they overlap might suggest some evidence.

Figure 4.28: An example of two lines that detach from each other.

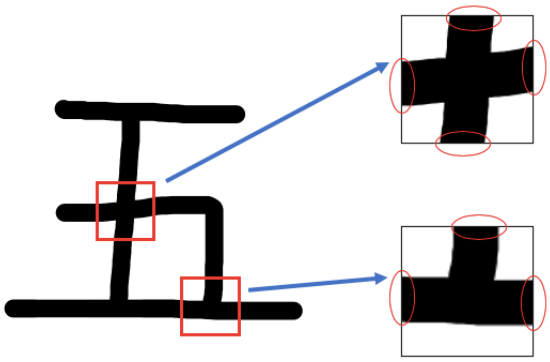


Figure 4.29: Analysing the overlapping area.

In Figure 4.29, by cropping around the intersecting area, the cropped image has black pixels on 4 sides of the boundary. Doing the same with the connecting area, we can see that the cropped image has black pixels on 3 sides. This is a plausible way of distinguishing if two overlapping strokes connect or intersect. However, not all overlapping strokes form a perpendicular crossing, because in a lot of other cases, they form diagonal crossings such as Figure 4.30. We can see that black pixels touches all four sides of the cropped image, but these two lines are connecting instead of intersecting. Therefore, I generalised it to counting black pixels that touch the boundary of the cropped image. The line width of each stroke is 20 pixels, which means a cropped image which contains an intersection will have at least 80 black pixels (20 pixels $\times$ 4 sides) on its boundary.

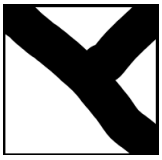


Figure 4.30: Diagonal connection.

To summarise, for detecting connection or intersection, we need to first identify the overlapping location, then crop a small area around the overlapping location. Next, by counting how many black pixels lie on the boundary of the cropped image, we can determine whether two strokes connect or intersect. If the black pixels count is over 80, we can classify it as an intersection, otherwise it is a connection.

Updating the relationship array uses the same method as mentioned in the previous sub-section.

Problem With The Second Method

Despite another attempt, the second method is still imperfect. It has some edge cases where the method will fail to determine some relationships.

Figure 4.31 and Figure 4.32 showcase two of the edges cases where the second method will not work

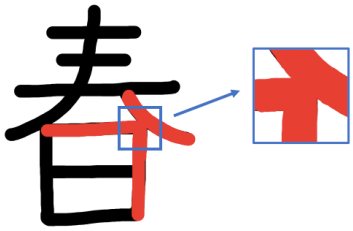


Figure 4.31: Edge case 1 (春).

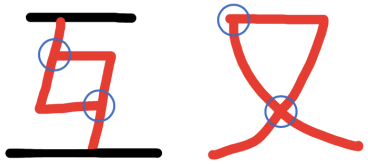


Figure 4.32: Edge case 2 (互, 又).

properly. Figure 4.31 shows the character 春 (spring), where two strokes – 丶 (Nà) and 𠃌 (Héng Zhé) are coloured in red. The relationship between the two strokes should be connect. However, looking at the cropped image around the area where they connect and recall that I assumed if cropped image has black pixels on all four sides then the relationship is intersect (section 4.4.4), the relationship between 丶 (Nà) and 𠃌 (Héng Zhé) will be determined as intersect in this case, which is incorrect. Figure 4.32 shows two characters – 互 (mutual) and 又 (again). Notice that the strokes in red in both characters have more than one place where they overlap with each other. In 互, 𠃍 (Shù Zhé) and 𠃌 (Héng Zhé) overlap with each other in two places, as circled in the image. And the same situation happens between 𠂆 (Héng Piě) and 丶 (Nà) in 又. The second method assumed that between two strokes there is at most one overlapping area, which clearly is not the case in these two characters. This will also lead to the method failing to determine their relationship.

Checking If Finished

If upon finishing the stroke, the `stroke_num` counter reaches the total stroke number, the detection process will terminate and users will be notified.

4.4.5 Providing Hints

Hints can guide users on where and what the next stroke should be. When hints are requested by the user, the correct stroke template images should be returned and displayed.

The returned images are URLs to the images instead of actual images. This will allow the images to be rendered much quicker on the canvas, because it does require the server application to transmit the individual pixel data and does not need to reconstruct the images upon receiving them. The returned value is an JSON object, an example can be found below:

```
{
"dis_x" : 50,
"dis_y" : 50,
"factor" : 0.8,
"url_1" : "http://127.0.0.1:5000/public/templates/五/image1.png"
"url_2" : "http://127.0.0.1:5000/public/templates/五/image2.png"
"url_3" : "http://127.0.0.1:5000/public/templates/五/image2.png"
...
}
```

`dis_x` and `dis_y` are the distance moved, as introduced in Updating Templates in section 4.4.3. `factor` is the ratio representing how much the template is resized by, as introduced in Size Checking in section 4.4.3. The urls each represents an stroke image. The number of URLs depends of the number strokes requested by user. If the user only requires the hint for the next stroke, only one URL will be returned, whereas if the user requires hints for the whole character, all stroke template images URLs will be returned.

4.5 Website Front-end Development

The detection will be run on a web application. As stated in Chapter 3 - Implementation Planning, the application should have an interface where users can input the characters they wish to practice and an interface where users can practice writing characters.

4.5.1 Character Input Interface

The main objectives in the character input interface are: 1. Pointing out the purpose of the application; 2. Giving intuitive instruction on how to get started; 3. Receiving character input from user and lead to the character practice interface. The completed interface is shown in Figure 4.33.

Figure 4.33: Input interface.

This HTML file is mainly comprised of a title, images, instructions, a text box for character input and a submit button. They are all placed in the centre for better readability. The short title gives the purpose of the application and the instruction simply points out what users should do to get started. The logo '查错' is a hand-written Chinese word meaning 'checking mistakes', made with a digital drawing tool.

The text box at the bottom records the user character input and the 'Go' button submits the request to server. When 'Go' button is clicked, JavaScript code will be triggered and a request will be sent to the server via HttpRequest. If the user character input is collected in the file system on the server, user will be directed to the character practice interface. Otherwise, user will be notified that the character has not been collected, as mentioned in Check If Character Exists in subsection 4.4.1.

4.5.2 Character Practice Interface

The character practice interface should provide a canvas for users to input hand-written characters. At the same time, it should allow user to make changes to their character and see clues on how to write the character if needed. Most importantly, a feedback section should be provided for users to see how to improve and should give necessary instructions. Figure 4.34 shows the completed interface.

This interface features a canvas, a feedback box and a few buttons. The canvas is created using a HTML5 Canvas element, with a standard character grid block pre-loaded onto it. The grid block is a

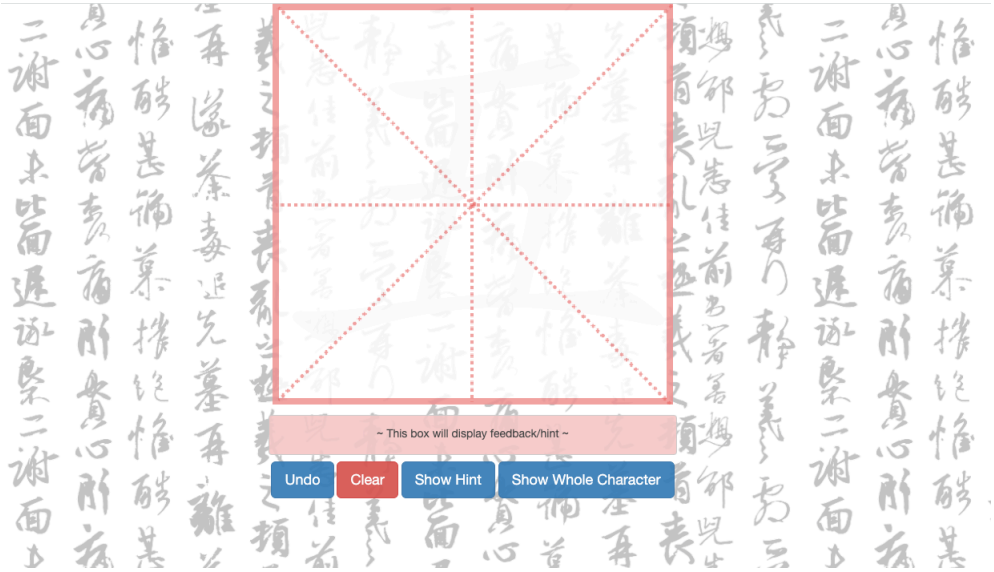


Figure 4.34: Practice interface

tool for helping learners understand the spacial relationship between different components in a character, which is commonly used among Chinese children when they first start to learn Chinese characters. From left to right, the buttons allow users to undo a stroke, clear the canvas, show the next stroke and show the entire character respectively.

Canvas Implementation

The canvas plays a key role in the detection process. It captures the individual stroke input by users and displays them. It also sends the user input to the server for detection.

The actions on the canvas are implemented in JavaScript. To capture the stroke input from users, HTML events such as `'mouseup'`, `'mousedown'` and `'mousemove'` are involved. `'mousedown'` event detects if the mouse is being pressed down and log the position as the start position where the drawing begins. When users move the mouse while holding the mouse down, `'mousemove'` event is triggered for every pixel moved across. For every single `'mousemove'` event detected, a line is drawn from the start position to where the movement ends. The start position is updated after each movement. This will generate a smooth line based on the users' writing trajectory. Finally, when `'mouseup'` event is triggered, which means the mouse is no longer pressed down, the current stroke will be marked as finished.

When the users write down the first stroke, one stroke will be recorded on the canvas. When the users down down a second stroke, two strokes will be recorded on the same canvas. This poses a challenge in the implementation, which is how to extract individual stroke image while keeping all strokes on the canvas which the users are working on.

This challenge is solved by having multiple hidden canvases behind the scene, which can be done by setting the display to be `none` using a CSS selector. When the writing interface is being loaded after the users have requested a character, the total number of strokes of this character will also be requested. This will determine the number of hidden canvases needed. When a stroke is written on the main canvas, a same stroke will also be written on a hidden canvas. The stroke images on the hidden canvases are the ones that will be sent to the server for detection. To show an example, I enabled the display of hidden canvases when writing 五 in the main canvas (Figure 4.35). From the hidden canvases, we can see the individual strokes been extracted out from the main canvas.

Upon finishing a stroke, the stroke image will be sent from the hidden canvas to the server for detection.

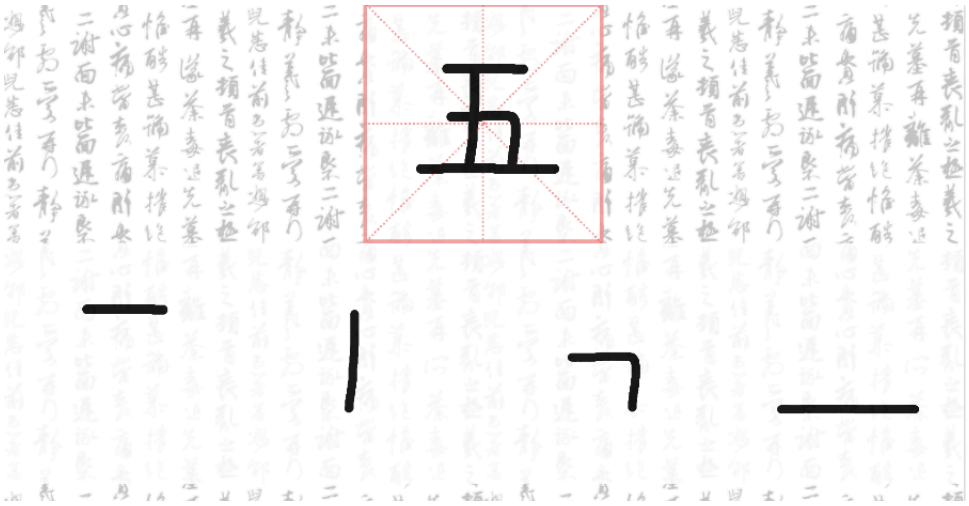


Figure 4.35: Hidden canvases

Implementation Of Undo And Clear

The 'Undo' button will reverse the writing by one stroke, which means that the current stroke will be deleted. 'Undo' is implemented by using a stack. A stack is an collection of items where items are always added on one end, and items are always removed from the same end. The end where items come in and are removed from is often referred to as 'top'. After each stroke is written by the user, an image from the main canvas will be added to the stack. The image on the top of the stack will always be displayed on the canvas. When the user request an 'undo' operation, the image on the top of the stack will be removed and the next one will be shown on the canvas, which will display the image from the previous stroke (Figure 4.36). The corresponding stroke image on the hidden canvas will be cleared subsequently.

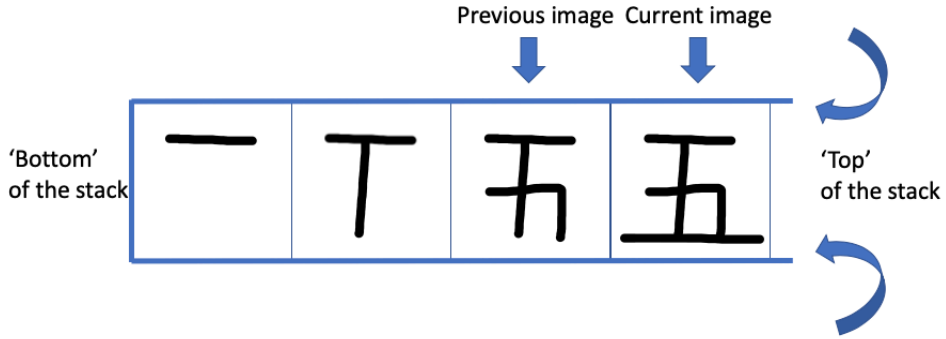


Figure 4.36: An example of the stack.

The 'Clear' button will clear everything on the canvas, meaning the whole detection process will start from the beginning. Implementing the 'Clear' operation is more straight forward. When 'Clear' operation is requested, main canvas will be emptied and all the hidden canvas will be emptied as well.

Displaying Hints

There are two buttons which allow users to request hints. 'Show Hint' button displays the hint for next stroke, whereas 'Show Full Character' displays the whole character on canvas.

The contents of return values are explained in subsection 4.4.5. The URLs returned can be used to create Image objects:

```
var hint_image = new Image();
```

```
hint_image.src = URL;
```

The HTML Canvas element provides a `drawImage` function which allows us to display images on a canvas. The function is specified by [35]:

```
void ctx.drawImage(image, dx, dy, dWidth, dHeight);
where
image is the image object,
dx, dy are the x-axis and y-axis coordinate in the destination canvas
at which to place the top-left corner of the source image,
dWidth is the width to draw the image,
dHeight is the height to draw the image.
```

Therefore, `hint_image` is drawn to the canvas by:

```
ctx.drawImage(hint_image, dis_X, dis_Y, factor * img_height, factor * img_width);
```

Since the hints are shown for indicative purpose only, they should disappear after they are viewed. To create this fade-out effect, each image is drawn multiple times using a for-loop, each with a smaller transparency value, until the hints fully disappear.

Chapter 5

Evaluation

In this chapter, the detection application will be evaluated by performing trial runs and conducting user studies. This is to illustrate to what extend has the application achieved the initial goals and is it plausible for the application to be actually deployed in real-world learning scenarios. The application is uploaded onto an online server, which can be accessed from:

`http://80.85.84.44:5000/public/test.html`

5.1 Choosing Characters For Evaluation

For a character to be detected using the process outlined in the Implementation chapter, 4 kinds of information are needed:

1. Template stroke images in BMP format, which enable size and position checking.
2. Template stroke images in PNG format, which are used to display hints on the canvas.
3. Correct stroke type data.
4. Correct stroke relationship data.

Stroke number	**Collected Characters**
3	大小干土子
4	书心从专五
5	本电鸟业出
6	在有她好会
7	我这来你没
8	宝诚使畅狗
9	是星春说将
10	被高通能起
11	眼球做
12	就然鼎
13	感新

Table 5.1: Collected Characters

Given the time frame of the project, it is impractical to collect the information needed for all 2,500 common characters. Instead, I have chosen 48 common characters among them which contain a mix of complexity and variety. The characters collected have stroke numbers ranging from 3 to 13. According to [36], among all 2500 frequent characters, 86% of them have a stroke number from 3 to 13. Therefore,

the characters collected are reasonably representative of the common characters. Table 5.1 shows a table of all collected characters and they are what the trial runs and user study will be based on.

5.2 Stroke Relationship Classification: CNN Model Vs Second Method

In chapter 4 I proposed two methods to determine the stroke relationship of two strokes in a character. Neither of them is perfectly reliable for correctly classifying the relationship between two strokes. The CNN model has high accuracy when classifying a single pair of strokes, but in a character the number of pairs that need to be classified grows exponentially as the number of stroke grows, which means lower and lower overall accuracy. Whereas in the second method, it has edge cases where sometimes a connection could be misinterpreted as an intersection etc.

Stroke relationship classification is a crucial step in the error detection process. Therefore, I have to chose between the two methods to apply in the detection process. The way to choose which one to apply to the detection is through testing. Testing will be done by writing correct characters and letting the two classification methods classify all the relationships within the characters. Since the methods will be used to classify correct characters only, ideally they should not report any error in the detection, meaning that they should classify all stroke relationships in a character correctly. But because of the weaknesses of the two methods discussed above, they will likely mis-report some errors. If a method classifies a character correctly (no error reported), we say that the method passes the character. The method which passes the most characters will be adopted. To make sure the classification works on other hand-writing styles but not just my own, I invited another Chinese speaker to help with testing. Both of us will test the 48 characters on both method, to see which method can pass the most characters.

Table 5.2 shows the results after testing with the two different methods, where a $\surd$ represents the method used can classify all stroke relationship within the character correctly and a $\times$ represents the method failed to do so. 'Total Passes' at the bottom of the tables indicates the number of characters for which the method classifies all stroke relationships correctly. From the results we can see that both methods are not able to classify the stroke relationships in all the 48 characters perfectly, but the second method passes more characters than the CNN model in both my own testing and the test helper's. Looking at the test results on CNN model, we can observe that as the table goes down (the characters get more complicated), the performance of CNN model goes down. As the number of strokes grows in a character, the stroke pairs that can be formed grows dramatically, which will greatly undermine the accuracy overall.

Overall, we can see that the second method passes more characters than the CNN model, and has good performance in complex characters. Although it occasionally fails in some edge cases, it still does a better job compared to CNN model, since edge cases are not common in characters.

Admittedly, neither of the two methods is perfect, but for the application to function, I needed to choose one. By comparing the results, I decided to use the second method over CNN model for classifying stroke relationships. This problem will need more effort in the future to be solved.

5.3 Trial Runs

All 48 characters collected will be tested on the complete application to verify the usability of the application, which means it will involve full error detection process (stroke type classification, size and location checking etc.) Specifically, these trial runs will evaluate if the implementation completed in chapter 4 has achieved its goal and if the performance is as intended. A full trial run on an example character 我 can be found in Appendix A. In this section, I will be more focused on evaluating the performance of the application and analysing the effectiveness of the detection process.

Character	CNN Model	2nd Method
大	√	√
小	√	√
干	√	√
土	√	√
子	√	√
书	×	√
心	√	√
从	×	√
专	√	√
五	√	√
本	√	√
电	√	√
鸟	×	×
业	√	√
出	√	√
在	√	×
有	√	√
她	√	√
好	√	√
会	√	√
我	√	√
这	√	√
来	√	√
你	√	√
没	√	×
宝	√	√
诚	×	√
使	√	√
畅	√	√
狗	√	√
是	√	√
星	√	√
春	√	×
说	√	√
将	√	√
被	×	×
高	√	√
通	×	√
能	√	√
起	×	√
眼	√	√
球	×	√
做	√	√
就	×	√
然	√	√
鼎	×	√
感	×	√
新	×	√
Total Passes	36/48	43/48

Character	CNN Model	2nd Method
大	√	√
小	√	√
干	√	√
土	√	√
子	√	√
书	√	√
心	√	√
从	×	√
专	√	√
五	√	√
本	√	√
电	√	√
鸟	×	√
业	√	√
出	√	√
在	√	×
有	√	√
她	√	√
好	√	×
会	√	√
我	×	√
这	√	√
来	√	√
你	√	√
没	√	×
宝	√	√
诚	√	√
使	×	√
畅	√	√
狗	×	×
是	×	√
星	√	√
春	√	×
说	×	√
将	√	√
被	×	×
高	√	√
通	×	√
能	√	√
起	×	√
眼	×	√
球	√	√
做	√	√
就	×	√
然	×	√
鼎	√	√
感	×	√
新	×	√
Total Passes	33/48	42/48

Table 5.2: Test results (left table: results from myself, right table: results from test helper)

5.3.1 Checking Stroke Size And Location

Stroke size checking works well on most characters, especially with it having specific ways of checking the size of each stroke type. Figure 5.1 and Figure 5.2 show examples of size checking on the strokes when writing the first stroke of character 是 (is). In Figure 5.1 when the vertical length of the stroke drawn by user (the opaque stroke) is shorter than half if the correct template stroke (the semi-transparent stroke), error will be reported with the feedback: 'probably too small, make it bigger? Please undo'. Whereas in Figure 5.2 the user-drawn stroke is longer than twice the length of the template stroke. An error was also reported with feedback: 'too big, make it smaller? Please undo'.

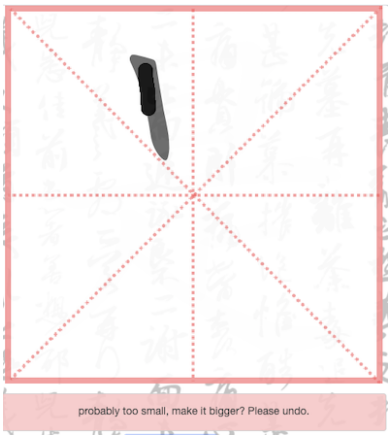


Figure 5.1: Example of size checking (too small).

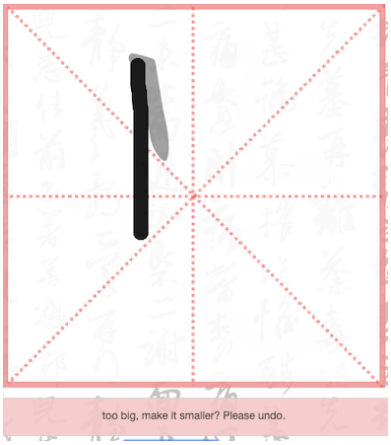


Figure 5.2: Example of size checking (too big).

Stroke location checking also does a fairly good job, it is able to give appropriate feedback when error is reported, based on the relative position to the correct stroke template. Figure 5.3 shows an example of location checking on the fifth stroke of 是, where the user-drawn stroke is below the correct template stroke, and the feedback is 'try moving it up a bit'.

Figure 5.3: Example of location checking.

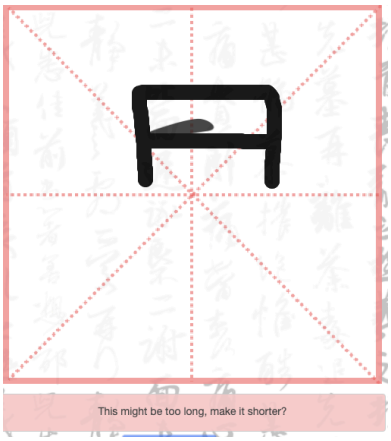


Figure 5.4: Example of size checking problem.

However, it is not without any problems. If we only look at individual strokes, the size checking is fine. But when it needs to connect with other strokes, problems sometimes occur. A typical example can be found in Figure 5.4, where the second stroke 𠃌 (Héng Zhé) in 是 is written bigger than usual but still has a big enough IOU score to pass the location check. The next stroke 一 (Héng) should connect with both the first stroke 丨 (Shù) and the second stroke 𠃌 (Héng Zhé). This is where the problem comes in. Because the second stroke is bigger than usual, the third stroke 一 (Héng) would also need to be longer than usual to reach 𠃌 (Héng Zhé), which would make it longer than the stroke template counterpart. As what we can see in Figure 5.4, the user-drawn 一 (Héng) is significantly longer than the

correct template, which leads to an error: 'This might be too long, make it shorter?'. If the third stroke is written a bit shorter it may satisfy the size checking but will fail the relationship checking, because a shorter 一 (Héng) will not reach 𠃌 (Héng Zhé). This is a dilemma situation where there is no way to satisfy both requirements. When situation like this happens, the only thing users can do is to start the second stroke again and make it smaller.

To solve this problem, I could potentially set a higher threshold for the IOU score, which means the second stroke 𠃌 (Héng Zhé) will have be more similar to the correct template in size in order to pass the check. However, changing IOU threshold can be problematic because the idea of a medium-level threshold is to allow some inaccuracy in size and location. Because landing every stroke in the exact location with the exact size is not realistic. The threshold of 0.5 is a result of multiple trials at the implementation stage which I think fits the inaccuracy people tend to make when writing. Therefore, changing the IOU threshold is not an effective option. When situation like this happens, user can only go back and make changes to previous strokes in other to proceed. This is a disadvantage of the application, but on the other hand, it can be an advantage as well. This will mean that users will need to write strokes which can more precisely match the correct stroke template in order to proceed. The resulting character will come closer to the correct template.

5.3.2 Stroke Type Classification

The CNN model for stroke type classification can classify most strokes correctly. Figure 5.5 shows the second stroke of 是, it matches the correct stroke 𠃌 (Héng Zhé), therefore no error is reported. In Figure 5.6, the second stroke was written as 𠃍 (Héng Zhé Gōu), which is pretty similar to 𠃌 (Héng Zhé), but the detector is still able to reports error, which feedback:'Wrong stroke. Correct stroke should be Hengzhe.'.

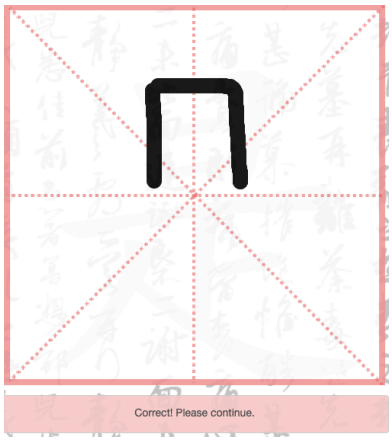


Figure 5.5: Stroke type classification (Héng Zhé).

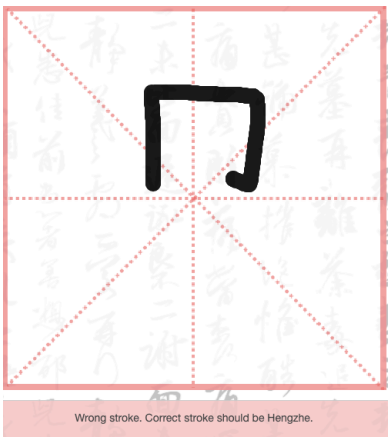


Figure 5.6: Stroke type classification (Héng Zhé Gōu).

Although the model classifies most strokes correctly, it fails to classify some strokes that are not in their standard form. For example, in 大 (big), 从 (follow) and 是 (is), they all contain the stroke 丿 (Nà). Yet the 丿 in these characters all appear in different forms. The 丿 in 大 is in its standard form, which is a cursive line from top to bottom right. The 丿 in 从 and 是 vary in shape. In 从, it is short and less cursive. In 是, it is wide and short, almost like it is lying down at the bottom. For one stroke type, many different variations can be found in different characters. Some can be very big in a character, others can be small. For example, 丿 in 木 (wood) is a lot bigger than the 丿 in 梦 (dream).

When training the model, I did take the variation of strokes in different characters into account, and further generated more samples through stretching and squishing the collected samples. Still, the samples that are used to train the model do not contain enough variation to allow classification on strokes that varies too much compared to the standard form. For a more robust model, more samples, especially

samples that contain more variation in shapes and size should be collected. The current sample size is far from enough.

5.3.3 Stroke Relationship Classification

In section 5.2, I explained the reason for choosing method two over the CNN model. I also talked about the limitation of both methods. To summarise, there has not been a robust way to classify the stroke relationships, but the current method can classify stroke relationships in most characters correctly.

Another thing that is worth talking about is the run-time for classifying relationships. Before classifying the relationship between two strokes, they are first combined into the same image. This step is done by iterating over every pixels in the images. The stroke images are all 500 $\times$ 500 in size, meaning that each combine operation needs to manipulate 250,000 pixels. In addition, as the number of strokes in a character grows, the stroke pairs that can be formed in that character grow dramatically. For a character which has 10 strokes, 45 stroke pairs can be formed. When writing down the second stroke in that character, only one pair needs to be classified. But as the process proceed, the later strokes will always need to form pairs with previous strokes. When the user writes the 10th stroke, stroke pairs will be formed with 9 previous strokes. They will all need to be classified. Therefore, as the user writes more strokes in a character, they will have to wait for longer and longer for the results to come out. Although the wait is only a few seconds, there is room for improvement.

To increase the speed I need to avoid doing pixel manipulation. A good idea will be to store all stroke images in SVG format and combine them, since SVG only stores information about how shapes are formed instead of individual pixel positions.

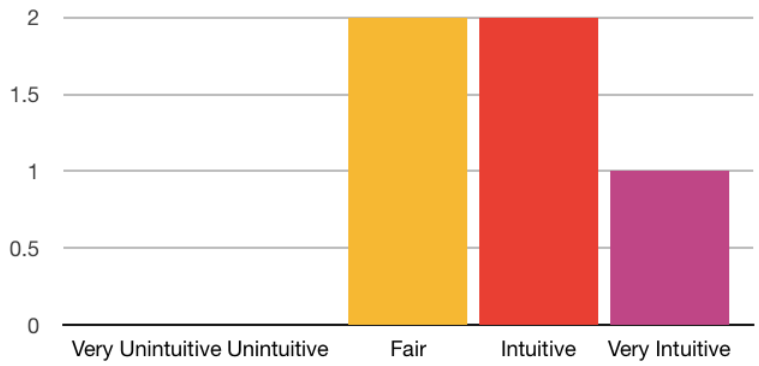
5.4 User Study

User study was conducted in order to understand the effectiveness of the application in terms of whether it helps with learning character, and understand the performance of the application from users perspective.

I invited 5 persons to participate in the study. The study was conducted remotely by arranging video or audio calls with individual participants. The participants are all Chinese learners at beginner to intermediate level from University of Bristol. The application was run on PC, laptop, or tablets, depending on participants' available devices. Participants were given a list of character from which they could choose 5 characters to test on the application. After they have tested the 5 characters, they were asked to fill out a questionnaire in which they would provide opinions on their experience and user feedback. The questionnaire can be found in Appendix B.

Questionnaire Results

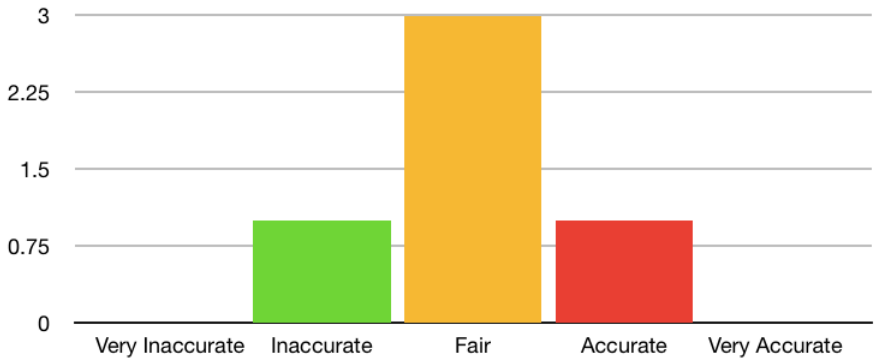
1. How intuitive do you think it is to use the application.



2. If you think it is unintuitive, please describe it (optional):

Number	Response
1	"I think the main improvement would be to automatically undo an incorrect stroke, or perhaps colour it red or something"

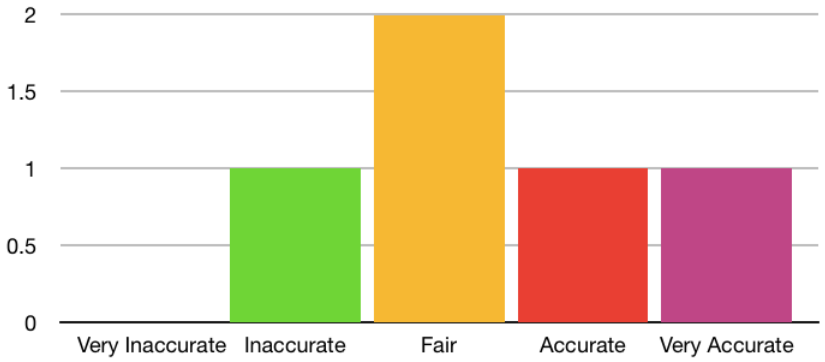
3. Overall, how accurate do you think it is in terms of stroke detection? (Example of inaccurate: When you think a stroke that you write is clearly correct but the detection result tells you otherwise.)



4. If you think stroke detection is inaccurate, please describe it (optional):

Number	Response
1	"Stroke intersection detection is a bit janky."
2	"I think it works reasonably well but is perhaps overfitting to the dataset (which would be difficult to overcome without a massive dataset)"
3	"Sometimes is quite fiddly, especially when strokes are small."

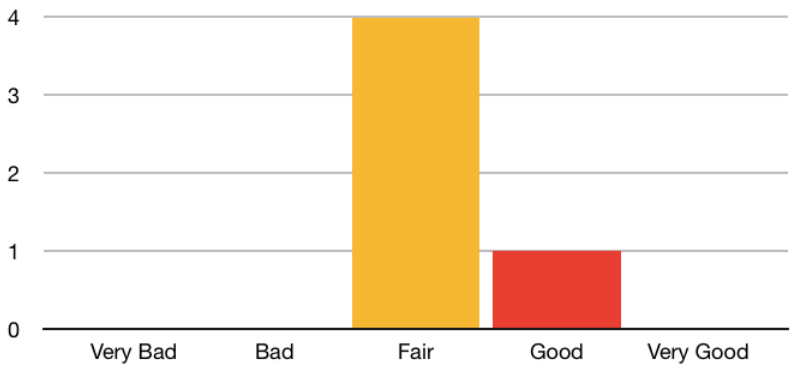
5. Overall, how accurate do you think it is in terms of position/size detection? (Example of inaccurate: When you think a stroke that you write is clearly in the right place and of the right size but the detection result tells you otherwise.)



6. If you think the position/size detection is inaccurate, please describe it (optional):

Number	Response
1	"Sometimes a bit weird, but overall okay."
2	"The location detection is somewhat too strict, otherwise it is okay."

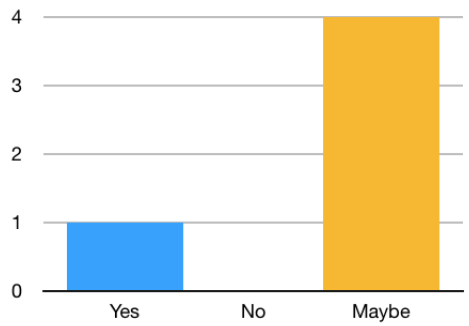
7. What is your overall experience of this detection application?



8. Can you describe your overall experience?

Number	Response
1	"It looks like a very good app! Obviously there's not a lot there, but with the time you had that's completely understandable. UI is nice but basic, bu"
2	"It is quite hard to complete a challenge I mean write a character correctly in this app."
3	"The concept is very nice, there are some problems and it does take a while to check each stroke but the foundation is there and it can work quite well."
4	"It's impressive how it can detect the exact stroke and is able to correctly gauge whether the stroke is too big, small, and whether it intersects with other strokes etc. Overall, I think with a few tweaks here and there, this application and framework could be built on to provide a robust solution for practicing and writing Chinese characters. That said, the framework is there for doing so, it's impressive how well it performs given such a non-trivial problem."
5	"Although it is quite strict about how strokes should look like and where they should be placed, it does make the final characters look nicer. Maybe it's a good thing that they are strict."

9. Would you use it for future character practices?



10. If you would not use it for future practices, can you describe why? (optional)

Number	Response
1	"Mainly because there aren't too many character there at the moment, but that's again just due to the time constraints."
2	"With some improvements I think I might use it, the long time to check each stroke makes it somewhat impractical at the moment"
3	"If the detection speed can be improved I will be happy to use it."

Analysis

From the answers for question 1 and 2, we can see that overall, the feedback on the intuitiveness of the application has been positive. Most participants did not ask for detailed instructions before starting and can navigate themselves to write characters on the canvas. One participant pointed out in the comment that they accidentally click on clear instead of undo. For this, I have changed the 'Clear' button to be red.

Question 3 and 4 ask about users' opinions on stroke detection, which include the stroke type detection and stroke relationship detection. We can see that most of them are not impressed by the performance. This is explained in the evaluation chapter, where I talked about the weakness of the implementation. Participants occasionally needed to attempt several times before they could get pass one stroke, even though the strokes they wrote were correct.

Question 5 and 6 are about position and size detection. The response from users suggests mixed opinions. A possible reason is that, the position and size of a stroke depends on its template stroke, which is based on Kai Shu writing. If a participant's normal writing style is closer to Kai Shu, they are more likely to find the detection accurate, because they pass detection quickly. Some other participants' writings may be correct but some strokes may not be close to the template stroke or may not be of similar sizes, which led to several further attempts.

Question 7 and 8 provide the overall user experience. We can see that most participants gave a neural rating on the application - Fair. This was expected, given the weakness in the stroke type classification model and some edge cases in the stroke relationship classification method. This suggest more work should be done in the future to improve the performance and accuracy of the application. All participants gave generous descriptions of their experience. Some appreciated the concept of the detection procedure, and some acknowledged that with more time and improvement it will become better.

Question 9 and 10 ask about users' opinions on applying the application in their future character practises. Most participants are hesitant about it, with it being sometimes inaccurate and slow. A positive side is that no one has chose 'No', and from the comments we can see that with some improvements they may consider using it in the future.

Chapter 6

Conclusion

6.1 What Was Achieved

In this application, my goal was to build an application which can detect mistakes that Chinese learners make when when writing characters and give them feedback accordingly. Linking back to chapter 1 where I stated what was expected of the final outcome, I think all of the initial goals were achieved. I built an application where users can write characters and see what they are writing. I collected hand-written samples for building both stroke type classification model and stroke relationship classification model. The application can detect errors in stroke type, stroke position and stroke relationships. It is also able to provide feedback based on the errors made. In this respect, the application fulfils all the initial goals. However, as analysed in the evaluation chapter, various problems exist in the classification models and the detection methods, and I have proposed ways to solve them where possible.

Overall, I think the application is not ready for deployment. Before it can properly help users write characters correctly, the application needs to be improved first, especially in light of the problems raised in the evaluation stage. More changes need to be made in the relationship classification and stroke type classification process.

In addition to achieving the initial goal of this project, I realised I have also achieved the ability to accomplish a project. From forming the initial idea, to doing research to verify the idea, and finally finishing it, it was a long way but a rewarding journey. Many problems were solved and many lessons were learned through researching and discussing with experienced people. Although the final outcome is not perfect, I am confident that with more time and effort, I can improve it to overcome its current weaknesses.

6.2 Future Work

An important task for future work is to gather more samples, because for stroke type classification, a more robust model is needed. This will require a larger dataset which contains strokes with various shapes and sizes, preferably from many more different people.

To improve the speed when classifying stroke relationships, instead of using BMG image and compare pixels, I should explore the use of SVG format. SVG format only contains the how a shape is drawn and the position of a shape instead of individual pixel positions of a shape, which could potentially accelerate the relationship detection process.

For the relationship classification method, I need to find another more reliable one to replace the current one, given that it has some edge cases where the classification will fail. I have two further ideas that should be tried out.

First idea is to combine the CNN model and the second method. I can use the second method to

decide whether two stroke have overlapping area. If they do not have overlapping area, it would mean that the relationship is detach. If they have overlapping area, a CNN model can be used to determine whether they connect or intersect. In this way, I can greatly reduce the number of pairs that need to be classified by the CNN model. And instead of having 3 classes in the CNN model, I only need 2, which are connect and intersect. This can also potentially increase the accuracy of classification.

My other idea is to use an image processing technique – Hough Transform. Hough Transform can be used to detect straight lines in an image. I can use this method to analyse the overlapping area of two strokes. In Figure A.15 we can see three pairs of lines, they all overlap with each other. The first two pairs intersect with each other and the last pair connect with each other. It is obvious that overlapping areas of intersections have four sides formed by straight lines while the overlapping area of connection on has one. If I can use Hough Transform to determine how many straight lines are in the cropped image of the overlapping area, I might be able to classify connection and intersection.

In addition, more characters should be collected into the application to allow a wider range of character selection.

Figure 6.1: Different overlapping areas.

6.3 What Could Have Been Done Differently

Looking back at the process of completing the project, the biggest regret has been the sample collection process. My initial idea is that I can collect samples from other people through out the project. The more samples I get the more accurate the model will become. Over the first term, I managed to collect samples from 5 people. Then I decided to use those samples to train the model. The training result was good with test accuracy at around 98%. Therefore I thought the 3200 samples together with the samples created from transformation are enough for the project. Later in the process when I started to implement the application, I realised it wasn't enough and it needed much more variations. I wanted to collect more but it was a little too late because of the virus outbreak. I should not have stopped collecting samples from other people, instead I should have been collecting samples while building the implementation. In this way maybe the application would be more accurate.

Bibliography

[1] Jiang, X., and Zhao, G, 2001. "**江新、赵果. 初级阶段外国留学生汉字学习策略的调查研究** [A Survey on the Strategies for Learning Chinese Characters among CSL Beginners]." *Language Teaching and Linguistic Studies* 4, pp.10–16.

[2] Hu, Z., Xu, Y., Huang, L. and Leung, H., 2009. A Chinese handwriting education system with automatic error detection. *JSW,* 4(2), pp.101-107.

[3] Shen, H.H., 2005. 'Introduction' in An investigation of Chinese-character learning strategies among non-native speakers of Chinese. *System*, 33(1), pp.49-68.

[4] Foreign Service Institute:*FSI' s Experience with Language Learning*[Online]. Available at: https://www.state.gov/foreign-language-training/(Accessed: 1st May 2020)

[5] Tian, F., Lv, F., Wang, J., Wang, H., Luo, W., Kam, M., Setlur, V., Dai, G. and Canny, J., 2010, April. Let's play chinese characters: mobile learning approaches via culturally inspired group games.*In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1603-1612.

[6] Wang, J. and Leland, C.H., 2011. Beginning Students' Perceptions of Effective Activities for Chinese Character Recognition. *Reading in a Foreign Language*, 23(2), pp.208-224.

[7] Shi, Z., 2000, April. Error Analysis Of Chinese Characters Written By Foreign Learners, *Chinese Character Learning*, 2(1), pp. 38-41.

[8] Kern, Martin (2010). "Early Chinese literature, beginnings through Western Han". In Owen, Stephen (ed.). *The Cambridge History of Chinese Literature, vol. 1: To 1375*. Cambridge: Cambridge University Press. pp. 1–115. ISBN 978-0-521-85558-7.

[9] Y. Gao, L. Jin and W. Yang, "An Empirical Comparative Study of Online Handwriting Chinese Character Recognition: Simplified vs. Traditional," 2013 *12th International Conference on Document Analysis and Recognition*, Washington, DC, 2013, pp. 862-866.

[10] State Language Commission of P.R. China, 2013. Table of General Standard Chinese Characters. Available from : http://www.moe.gov.cn/s78/A19/yxs_left/moe_810/s230/201306/t20130601_186002.html

[11] Sheng, P., 2017. A Comparative Study of Kangxi Dictionary and Samuel Johnson' s A Dictionary of the English Language: In honor of the 300th Anniversary of Kangxi Dictionary, *Lexicographical Studies*, 3(2), pp. 67-74.

[12] Yuming, Li., and Wei, Li. (eds) (2013). The Language Situation in China, *De Gruyter Mouton.* Available From: https://doi.org/10.1515/9781614512530 (Accessed 13 April 2020)

[13] Huang, B., Liao, X., 2007. 《**现代汉语（增订四版）**》[Modern Chinese (4th Volumn)], **高等教育出版社**, pp. 145.

[14] State Language Commission of P.R. China, 2001. Chinese Character Turning Stroke Standard of GB 13000.1 Character Set. Available from : http://www.moe.gov.cn/ewebeditor/uploadfile/2015/01/12/20150112170016626.pdf

[15] Chen, G., Zheng, Y. and Lin, L., 2007 "Computer-based Assessment for the Stroke Order of Chinese Characters Writing," *Second International Conference on Innovative Computing, Information and Control (ICICIC 2007)*, Kumamoto, pp. 160-160.

[16] Chen, C., Feng, L., 2016, **意大利汉语初学者汉字基本笔顺书写规律及其相关因素研究** [A study of the writing rule of the Chinese-character stroke order and its related factors：A study of Italian Chinese-learning beginners], **云南师范大学学报 *(*对外汉语教学与研究版*)***, pp.6-11.

[17] Li, Y., 2013, **汉字笔顺与对外汉字教学研究 – 以上海师范大学初级班留学生为例** [Chinese Character Stroke Order and Teaching Chinese Character Research – The Students of SHNU in Junior Class as an Example].

[18] State Language Commission of P.R. China, 1997. **现代汉语通用字笔顺规范** [Modern Chinese Common Characters Stroke Order Standards]. Available from: http://www.moe.gov.cn/s78/A19/yxs_left/moe_810/s230/201001/W020181228294361117660.pdf (Accessed: 15th April 2020)

[19] Hubel, D.H. and Wiesel, T.N., 1968. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1), pp.215-243.

[20] Yamashita, R., Nishio, M., Do, R.K.G. and Togashi, K., 2018. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4), pp.611-629.

[21] Matthew Stewart, 2019. Simple Introduction to Convolutional Neural Networks, *Towards data science.* Available from: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac (Accessed: 20th April 2020)

[22] Sumit Saha, 2018. A Comprehensive Guide to Convolutional Neural Networks —the ELI5 way, *Towards data science.* Avaliable from: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53(Accessed: 23rd April 2020)

[23] Kuo, C.C.J., 2016. Understanding convolutional neural networks with a mathematical model. *Journal of Visual Communication and Image Representation,* 41, pp.406-413.

[24] CS231n Convolutional Neural Networks for Visual Recognition: https://cs231n.github.io/convolutional-networks/

[25] W3school: https://www.w3schools.com/css/csssyntax.asp(Accessed: 23rd April 2020)

[26] Leif Johnson, 2015. Loss Functions. *Theanets 0.7.3 documentation*, Available from: *https://theanets.readthedocs.io/en/stable/api/losses.html*(Accessed: 4th May 2020)

[27] Saugat Bhattarai, 2018. What is gradient descent in machine learning. *A TECH BLOG* Available from: https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/(Accessed: 5th May 2020)

[28] Lawrence, S., Giles, C.L., Tsoi, A.C. and Back, A.D., 1997. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks,* 8(1), pp.98-113.

[29] TensorFlow: https://www.tensorflow.org/(Accessed: 30th April 2020)

[30] Keras documentation: https://keras.io/ (Accessed: 30th April 2020)

[31] Keras Documentation: https://flask.palletsprojects.com/en/1.1.x/quickstart/quickstart(Accessed: 10th April 2020)

[32] Sambit Mahapatra, 2018. A simple 2D CNN for MNIST digit recognition. *Towards Data Science*. Available from : https://towardsdatascience.com/a-simple-2d-cnn-for-mnist-digit-recognition-a998dbc1e79a

[33] Pexel.com : https://www.pexels.com/photo/photography-of-red-car-on-road-1172105/ (Accessed: 5th May 2020)

[34] Ronny.rest : http://ronny.rest/tutorials/module/localization$_0$01/iou/($Accessed$: 5$thMay$2020)

[35] HTML Canvas Documentation by Mozilla: https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage (Accessed: 1st May 2020)

[36] Guo, S., Piao, Z., 2006.《GB13000. 1 **字符集: 汉字字序 (笔画序) 规范》笔画数统计报告** [Stroke Number Analysis Report of *GB13000.1 Character Set: Character Order (Stroke Order) Standard*]. **现代语文: 下旬**. **语言研究**, (11), pp.39-40.

[37] Make me a Hanzi repository: https://github.com/skishore/makemeahanzi (Accessed: 14th February 2020)

Appendix A

A Full Trial Run

Figure A.1: In the character input interface, input 我 and click on Go button.

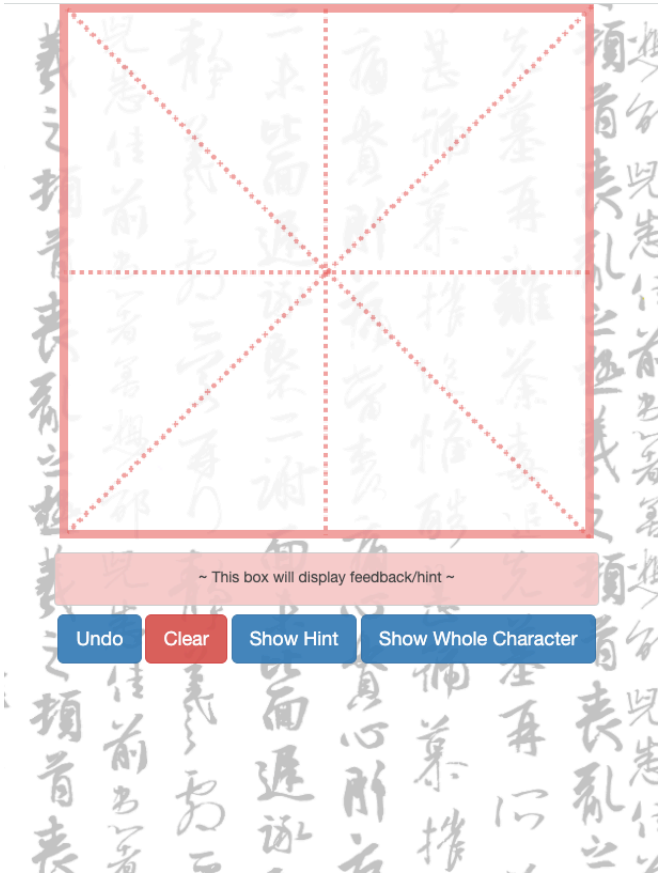


Figure A.2: This is the practise interface, where users write characters.

Figure A.3: Before starting to write, users can view the character by clicking Show Whole Character. The hint will fade away shortly.

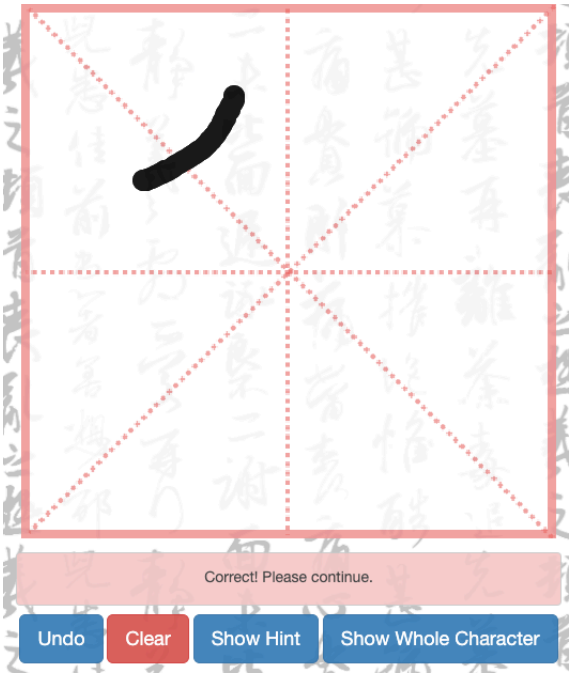


Figure A.4: The first stroke of 我 is 丿 (Piě).

Figure A.5: After the first stroke, the template character will be updated based on the size and location of the first stroke.

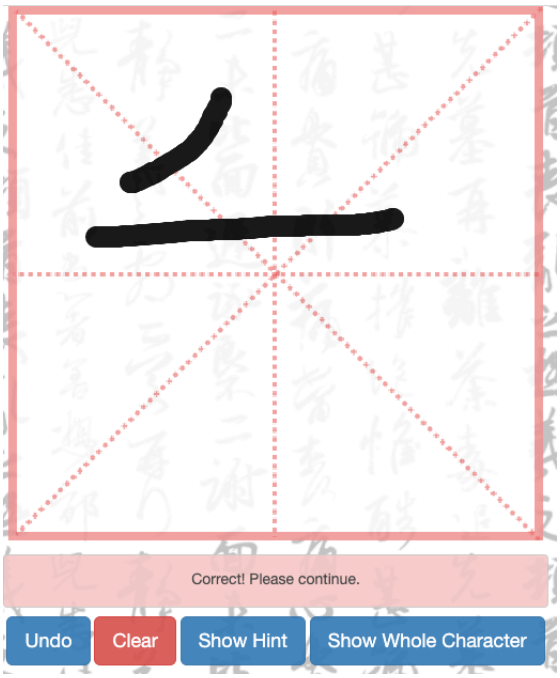


Figure A.6: The second stroke of 我 is 一 (Héng).

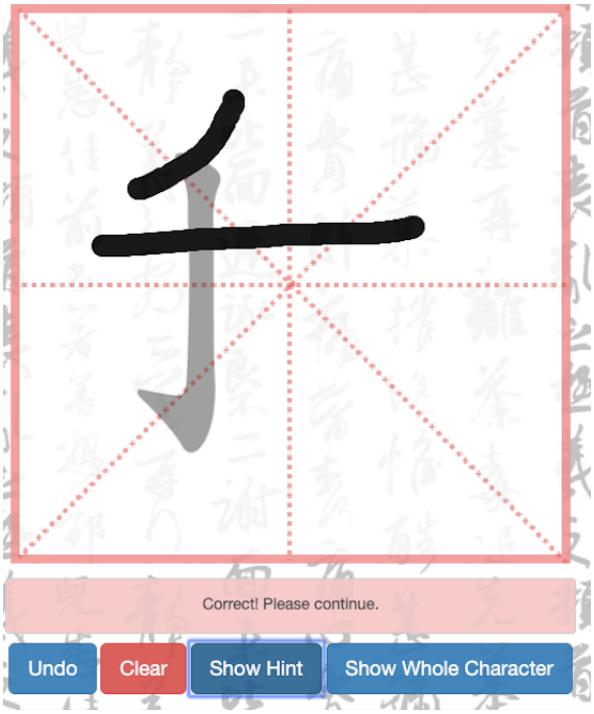


Figure A.7: If users forget how to write the third stroke, the Show Hint button will give hint on the third stroke. The hint will fade away shortly.

Figure A.8: Completing the third stroke.

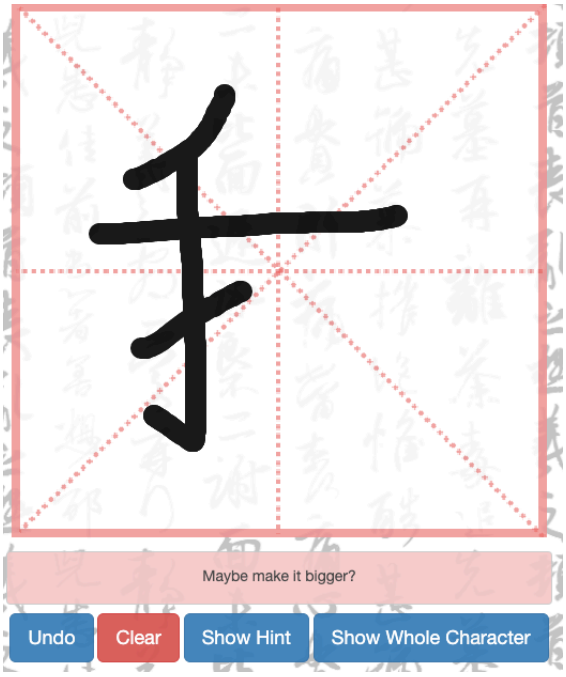


Figure A.9: Writing the fourth stroke but it seem a bit too small.

Figure A.10: Redo the fourth stroke, with a bigger 𠂉 (Tí) this time.

Figure A.11: The fifth stroke should be a 𠃌 (Xié Gōu) but was accidentally written as a 丨 (Shù). It failed to pass the stroke type detection.

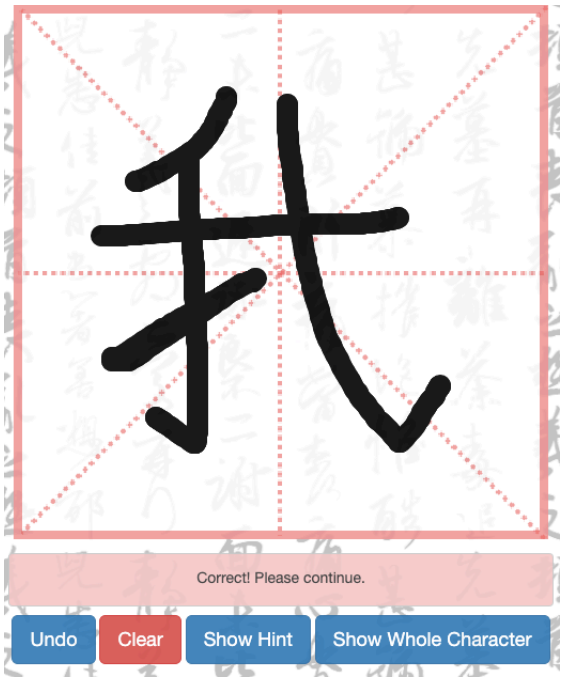


Figure A.12: Redo the fifth stroke.

Figure A.13: Write the sixth stroke 丿 (Piě).

Figure A.14: Write the seventh stroke 丶 (Diǎn). It is in a wrong position. The feedback suggests that it should move to the left.

Figure A.15: Redo the seventh stroke. The full character is now completed.

Appendix B

User Study Questionnaire

Hand-written Chinese Characters
Error Detection Application Questionnaire

The following questions are for Hand-written Chinese Characters Error Detection user study.

1. How intuitive do you think it is to use the application.
   a. Very Unintuitive
   b. Unintuitive
   c. Fair
   d. Intuitive
   e. Very Intuitive

2. If you think it is unintuitive, please describe it (optional):

   _

3. Overall, how accurate do you think it is in terms of stroke detection? (Example of inaccurate: When you think a stroke that you write is clearly correct, but the detection result tells you otherwise.)
   a. Very Inaccurate
   b. Inaccurate
   c. Fair
   d. Accurate
   e. Very Accurate

4. If you think the stroke detection is inaccurate, please describe it (optional):

   _

5. Overall, how accurate do you think it is in terms of position/size detection? (Example of inaccurate: When you think a stroke that you write is clearly in the right place and of the right size, but the detection result tells you otherwise.)
   a. Very Inaccurate
   b. Inaccurate
   c. Fair
   d. Accurate
   e. Very Accurate

6. If you think the position/size detection is inaccurate, please describe it (optional):

   _

7. What is your overall experience of this detection application?
   a. Very Bad
   b. Bad
   c. Fair
   d. Good
   e. Very Good

8. Can you describe your overall experience?

   _

9. Would you use it for future character practices?
    a. Yes
    b. No
    c. Maybe

10. If you would not use it for future practices, can you describe why? (optional)

    _

Appendix C

Experiment Permission Form

Student experiment permission form for COMS30500 and COMSM0111

Lead supervisor name: Ian Nabney

Student name(s): Shuye Liu

Student ID(s): 1710533

Project title: Hand-Written Chinese Character Error Detection

Summary of proposed experiment(s):

User study of a web application for assisting Chinese character learning using machine learning techniques.

In order for the project to fall within the scope of ethics application 97842, all answers to the following questions must be "No". More details are available in the project handbook, available on the unit Blackboard page.

Does the proposed experiment gather data from animals? Yes / No

Does the proposed experiment gather data from vulnerable populations? Yes / No

Does the proposed experiment gather sensitive information? Yes / No

Does the proposed experiment take photos or videos of people? Yes / No

Does the proposed experiment gather other data which, if lost, would allow participants to be identified? Yes / No

Does the proposed experiment gather data from people who have not given full informed consent? Yes / No

Does the proposed experiment trick or deceive participants in any way? Yes / No

Does the proposed experiment involve danger of physical or mental harm? Yes / No

I certify that the above information is accurate, and that I have read the relevant section of the project handbook.

Date: 5th May 2020 **Student signature(s): Shuye Liu**

I certify that I have discussed the methodology of the proposed experiment with the student(s) in detail, and that I am satisfied that the above information is accurate.

Date: 5th May 2020 **Lead supervisor signature: Ian Nabney**