Handwritten Digit and Signal Recognition for Children

Yiwei Huang
The Chinese University of Hong Kong, Shenzhen
120090800@link.cuhk.edu.cn
Boshi Xu
The Chinese University of Hong Kong, Shenzhen
boshixu@link.cuhk.edu.cn
Yuhang Huang
The Chinese University of Hong Kong, Shenzhen
122040082@link.cuhk.edu.cn

Abstract

In the digital age, deep learning and other digital technologies have rapidly matured, with resulting applications quickly entering the market and reducing costs. Early childhood math education often involves time-consuming tasks, psychological pressure, and material waste. Automatically correcting children's handwritten calculations can significantly reduce the mental and physical stress on parents and teachers, decrease paper waste, and protect the environment. Our project aims to develop a simple, user-friendly handwritten recognition and calculation system. Utilizing datasets from the internet and self-collected children's handwriting, we preprocess images through multiple OpenCV methods including Gaussian blurring, edge detection, and Contour Recognition. We then employ Convolutional Neural network (CNN) to classify and verify handwritten symbols and numbers. The front-end, built with React, captures and processes user input, integrates with a Python backend for image analysis, and outputs the results, streamlining the educational process and to achieve the expected goals.

1 Introduction

In the modern age, with rising labor costs and the pervasive shift towards digitalization, the realm of early childhood education is also undergoing transformation. As a response to these trends, there is a growing necessity to streamline educational processes, particularly in areas such as homework correction, which traditionally consumes significant time for both parents and teachers. Recognizing these challenges, our goal is to develop an innovative program capable of autonomously evaluating simple handwritten calculations provided by children. By harnessing the power of digital tools, we aim to alleviate the burden on educators and parents, allowing them to focus more on guiding and nurturing young learners rather than spending valuable time on routine tasks like checking basic arithmetic exercises.

In order to realise the recognition and checking of handwritten computational equation, it can be roughly divided into three steps, the first is to input the handwritten picture through the front-end, the second is to pre-process the input picture and split each individual number and symbol into separate pictures, and the last is to recognise each character and check whether the answer is correct or not, and then output the result. The preprocessing method used for most handwritten digit recognition is contour recognition to obtain the border of the digit. Although contour recognition can correctly recognise the border of the digit, it fails to recognise some special symbols such as the division sign and the equals sign. The contours of these two symbols are not consecutive, so multiple contours are recognised and multiple borders are obtained. This is not the expected result, so we have to find a

way to capture non-contiguous characters such as the division sign and the equals sign with only one border.

In this paper, we propose that the distance between the centres of the contours can be used to determine whether the two contours should be selected with only one border. The key to this algorithm is to find the distance between the centre of the previous contour and the centre of the next contour in left-to-right order then compare this distance with the minimum value derived from our tests, if this distance is greater than the minimum value then the next contour is considered to be a separate contour as well to be added to the contour array, and if this distance is less than the minimum value then it is assumed that this contour and the previous contour belong to the same character, and the next contour is not stored in the contour array. With this algorithm we can successfully get the division and equal sign with only one border

In conclusion, the main contribution of this work is that using the distance between the centres of the contours to determine whether the two contours should be selected with only one border. https://github.com/xh2002/EIE4512_FinalProject/tree/master

1.1 Related Work

Research on recognizing numerical symbols and processing calculation results has been extensive, focusing primarily on improving accuracy and efficiency. Various methods, including convolutional neural networks (CNNs), support vector machines (SVMs), and recurrent neural networks (RNNs), have been employed to address the complexities of handwritten mathematical symbol recognition (He, 2020). These studies typically utilize datasets like MNIST, CROHME, and others for training and validation (Sakshi, 2022).

Despite significant advancements, current methods face several limitations. The computational cost of real-time processing can be high, and most existing solutions do not focus on specific user groups, such as children, whose handwriting may differ significantly from adults. Thus, the lack of sufficient training data limits the generalizability of these models.

Our approach differs by focusing specifically on the handwriting of young children, a demographic often overlooked in existing research. We incorporate preprocessing techniques like Gaussian blurring and edge detection tailored to children's handwriting. Our system also employs a simplified user interface to facilitate ease of use in educational settings. By using a customized dataset of children's handwritten digits and symbols, our model is trained to handle the unique challenges presented by this user group, thus improving both accuracy and usability in practical applications.

2 The Proposed Algorithm

2.1 Image Preprocessing

First, the input colour image (BGR format) is converted to a grayscale image using the cv2.cvtColor function. A grayscale image contains only luminance information, which helps reduce the amount of computation for subsequent processing. Then, the grayscale image is Gaussian blurred using the cv2.GaussianBlur function. It removes high-frequency noise and details from the image and makes the image smooth. The blurred image is immediately followed by edge detection using the cv2.Canny function. Next, the cv2.dilate function is used to perform the dilation operation on the Canny edge-detected image. Finally, the cv2.erode function is used to perform the erosion operation on the expanded image. The erode operation shrinks the edges in the image inward, which helps in removing small and unimportant details. The main purpose of these operations is to extract and enhance the edge information in the image and to remove noise and unnecessary details from the image, making the edges clearer and more prominent, thus improving the accuracy and efficiency of subsequent processing.

2.2 Image Splitting

Contour detection is performed using the cv2.findContours function, which retrieves the external contours from the preprocessed image. The contours are then filtered based on their area and distance from each other.

The detailed operation of filtering is to use the sorted() function to sort the contours extracted from the image in order from left to right, creat the variable prev rect to store the border of the previous contour and then we use cv2.boundingRect() function to get the border of the first contour as the initialisation data for the prev rect (because we default that the first contour must be a number). Creating the filtered contours array to store the filtered contours and initialise the first element as the first contour. Next, iterate through all the contours. (skip the first contour because the first contour is a number by default), use cv2.boundingRect() function to get the origin coordinates of the contour's borders, as well as the length and width so that we can calculate the contour's centre coordinates, and then using np.linalg.norm(np.array(centre) - np.array(prev centre)) to calculate the distance from the centre of the previous contour to the centre of this contour, then compare this distance with distance threshold (a suitable threshold we have tested many times), if it is less than distance threshold then skip the contour and continue traversing the next contour, if it is greater than distance threshold then add the current contour to filtered contours and update prev rect to the bounding box of the current contour.

Finally we traverse the filtered contours contour array. Still using cv2.boundingRect to get the contour's border coordinates x,y as well as the width w and height h. Using this data to intercept the image directly from the original image. The formula for intercepting the image is img[y-50:y+h+50, x-20:x+w+20]. Border in y-axis is expanded more than x-axis because we need to make sure that the division sign and the equals sign are captured in full.

A Convolutional Neural Network (CNN) is a type of deep learning model particularly well-suited for processing image data. It can automatically extract features from images and use these features for classification, detection, and other tasks. In this context, the CNN algorithm is used for character recognition by training a model to recognize characters in input images. In our program, we arrange the data collected ourselves, training a CNN model, and using this model to recognize the characters in images.

1. Training

(1) Load and Data Preprocessing

First, image data and their corresponding labels are loaded as figure 1 shown.

```
Saved 2840 images to D:\Python_code\Output\add.npy
Saved 2629 images to D:\Python_code\Output\divide.npy
Saved 2775 images to D:\Python_code\Output\eight.npy
Saved 2827 images to D:\Python_code\Output\five.npy
Saved 3180 images to D:\Python_code\Output\four.npy
Saved 3160 images to D:\Python_code\Output\multiply.npy
Saved 3212 images to D:\Python_code\Output\nine.npy
Saved 3631 images to D:\Python_code\Output\one.npy
Saved 3246 images to D:\Python_code\Output\seven.npy
Saved 2984 images to D:\Python_code\Output\six.npy
Saved 3452 images to D:\Python_code\Output\subtract.npy
Saved 2586 images to D:\Python_code\Output\three.npy
Saved 3828 images to D:\Python_code\Output\two.npy
Saved 2399 images to D:\Python_code\Output\zero.npy
```

Figure 1: image data and their corresponding labels

The data is then split into training and testing sets. The image data is normalized by scaling pixel values to a range of 0 to 1. The normalization formula is:

$$\mathrm{x}_{norm} = \frac{\mathrm{x}}{255}$$

Next, the image data is reshaped to match the input shape required by the CNN.

(2) Build the CNN Model

A sequential CNN model is constructed, consisting of the following layers:

a. Convolutional Layer: Applies a convolution operation to the input. The formula is:

$$Conv2D(x) = ReLU\left(\sum_{i=1}^{k} w_i x_i + b\right)$$

b. Activation Function: Uses the ReLU (Rectified Linear Unit) activation function. The formula is:

$$ReLU(x) = \max(0, x)$$

c. MaxPooling Layer: Reduces the spatial dimensions of the output. The formula is:

$$MaxPooling(x) = \max(x)$$

d. Dropout Layer: Randomly sets a fraction of input units to 0 to prevent overfitting. The formula is:

$$Dropout(x) = x \cdot mask, where mask \sim Bernoulli(p)$$

e. Flatten Layer: Converts the 3D data to 1D.

f. Dense Layer: Fully connected layer. The formula is:

$$Dense(x) = ReLU(W \cdot x + b)$$

g. Output Layer with Softmax: Converts the output to a probability distribution over classes. The formula is:

$$Softmax\left(x_i\right) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

(3) Compile and Train the Model:

The model is compiled using categorical crossentropy loss, the Adam optimizer, and accuracy metrics. The model is then trained with the training data for a specified number of epochs and batch size. The training parameters are:

Batch Size: 32

Epochs: 10

Finally, the trained model is saved, this process is shown as figure 2.

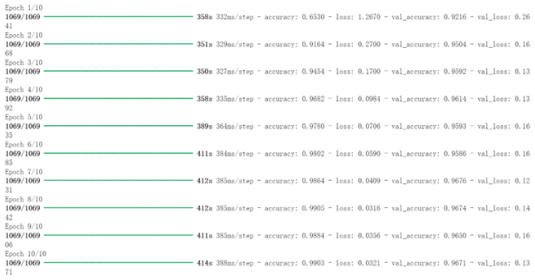


Figure 2: trained model is saved

2. Character Recognition (CNN_recognition.py)

(1) Load the Trained Model:

The trained model is loaded from the saved file.

(2) Preprocess Input Image:

The input image is loaded and resized. The image data is then normalized using the formula:

$$x_{norm} = \frac{x}{255}$$

The image data dimensions are expanded to match the input shape required by the model.

(3) Predict Character:

The trained model is used to predict the character in the input image as shown in figure 3. The predicted index is mapped to the corresponding character, and the result will save as a picture shown below in figure 4.

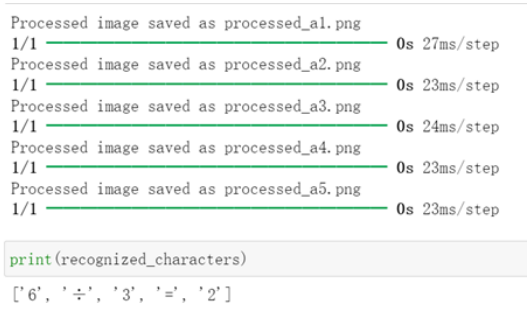


Figure 3: prediction

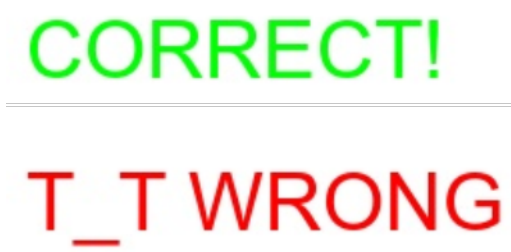


Figure 4: result

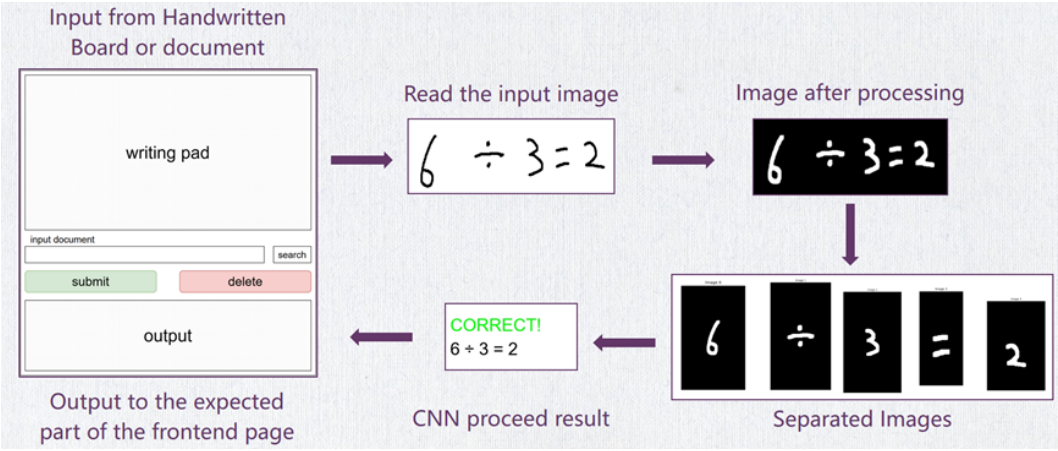


Figure 5: Pipeline

3 Experiments

3.1 Evaluation Dataset

The dataset combine with two parts, the first part of the dataset is *Mathematics Symbols Data* collected from the Internet which is related to the recognition with single digits as well as symbols. The dataset contains images of numbers from 0 to 9 and the 4 maths operator- sum, divide, subtract and multiply. The another part of dataset contains numbers of children's handwriting pictures collected by ourselves.

3.2 Accuracy

Due to the characteristics of the split analysis of this procedure, after testing, the accuracy rate decreases slightly with the increase in the number of digits, the accuracy rate for the recognition of a single digit or symbol is 0.9903, the accuracy rate for a simple calculation with 5 characters is 0.95, and when the number of characters reaches 8, the accuracy rate is 0.92.

3.3 Average time

After our 30 tests, the average running time of the whole programme was 11.535 seconds. All test samples are three digits and below, so we also conclude that the increase in characters in equation with three digits and below has a negligible increase in runtime.

3.4 Range

Because of the area limitation of the tablet and the accuracy of the recognition and considering the needs of the target group, the calculation result needs to be within three digits.

3.5 Environment

All experiments and tests were run in the following environment:

Python version: 3.11.7 | packaged by Anaconda, Inc.

pillow 10.2.0

opencv-python 4.10.0.82

keras 3.4.1

tensorflow 2.17.0rc1

cpu: AMD Ryzen 7 6800H with Radeon Graphics 3.20 GHz

4 Misc for preparing your paper

Misc for preparing your paper

For the front-end part, we want to create a simple and easy-to-use handwritten recognition and calculation user interface.

The front-end part includes:

1. Handwriting board

2. Optional file input

3. Text result output

To implement these features effectively, we adopted the following approach:

1. Developing the Front-End with React:

- We chose the React framework to build our front-end due to its efficiency in managing the user interface and its ability to handle dynamic content updates. React's component-based architecture allows us to create reusable UI components, streamlining the development process and ensuring maintainability.

2. Using Canvas for Handwriting Capture:

- To track mouse movements and capture handwriting, we utilized the HTML5 canvas element. By leveraging the canvas context, we can monitor user input in real-time and render the corresponding

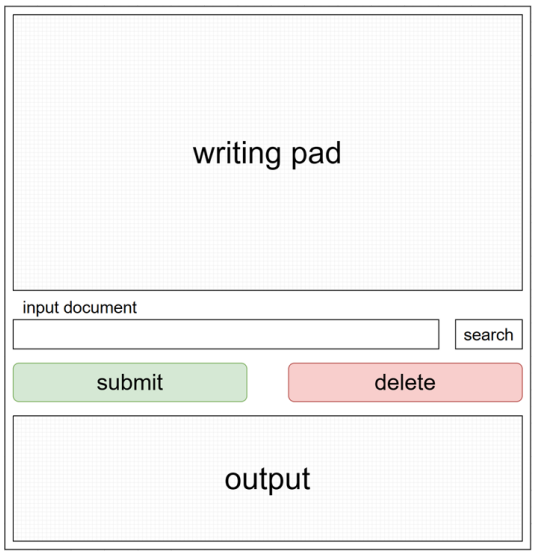


Figure 6: front-end

strokes on the canvas. This enables users to write naturally, mimicking the experience of writing on paper.

3. Implementing Image Uploads:

- For users who prefer to upload handwritten images rather than writing directly on the board, we included an input element with ‘type="file"‘. This allows users to select and upload images from their devices. The selected images are then displayed on the interface for further processing.

4. Integrating with a Python Back-End:

- To handle the image recognition and calculation, we set up a simple Python server. The server is designed to receive the uploaded images via an HTTP API. Upon receiving an image, the server invokes the recognition algorithm to process the handwriting. The result, whether it’s recognized text or a calculated value, is then sent back to the front-end and displayed in the text result output section.

This approach ensures a seamless and interactive experience for users, allowing them to either draw directly or upload images for recognition.

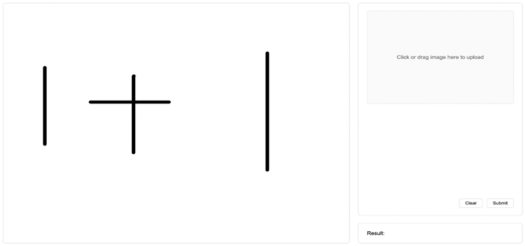


Figure 7: front-end

5 Reference

He, F., Tan, J., Bi, N. (2020). Handwritten Mathematical Expression Recognition: A Survey. In: Lu, Y., Vincent, N., Yuen, P.C., Zheng, WS., Cheriet, F., Suen, C.Y. (eds) Pattern Recognition and Artificial Intelligence. ICPRAI 2020. Lecture Notes in Computer Science(), vol 12068. Springer, Cham. https://doi.org/10.1007/978-3-030-59830-3_5

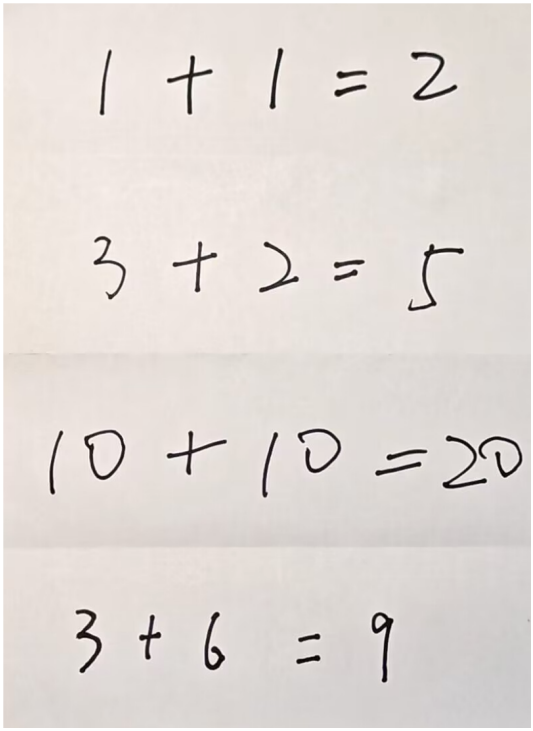
Sakshi, Sharma, C., Kukreja, V. (2022). CNN-Based Handwritten Mathematical Symbol Recognition Model. In: Tavares, J.M.R.S., Dutta, P., Dutta, S., Samanta, D. (eds) Cyber Intelligence and Information Retrieval. Lecture Notes in Networks and Systems, vol 291. Springer, Singapore. https://doi.org/10.1007/978-981-16-4284-5_35

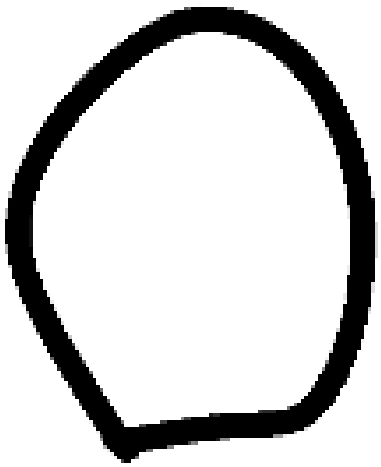
He, F., Tan, J., Bi, N. (2020). Handwritten Mathematical Expression Recognition: A Survey. In: Lu, Y., Vincent, N., Yuen, P.C., Zheng, WS., Cheriet, F., Suen, C.Y. (eds) Pattern Recognition and Artificial Intelligence. ICPRAI 2020. Lecture Notes in Computer Science(), vol 12068. Springer, Cham. https://doi.org/10.1007/978-3-030-59830-3_5

Sakshi, Sharma, C., Kukreja, V. (2022). CNN-Based Handwritten Mathematical Symbol Recognition Model. In: Tavares, J.M.R.S., Dutta, P., Dutta, S., Samanta, D. (eds) Cyber Intelligence and Information Retrieval. Lecture Notes in Networks and Systems, vol 291. Springer, Singapore. https://doi.org/10.1007/978-981-16-4284-5_35

Appendix

1. Some of the training samples.


1 + 1 = 2
3 + 2 = 5
10 + 10 = 20
3 + 6 = 9



2. Coding Part: https://github.com/xh2002/EIE4512_FinalProject/tree/master/Code