iLMS Captcha Breaker by JavaScript Injection

1st Tsung-Wei Lin
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan

2nd Yi-Hsuan Pan
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan

***Abstract*—Several tools have been employed for number recognition; however, no tools has accomplished both high efficiency and high accuracy without installing pre-trained model. This study aims to design a delicate algorithm for recognizing the captcha on iLMS. Specifically, the method is composed with three parts, artificial noise removal, partition, and number recognition. Color filtering and fix position partition are used by artificial noise removal and partition, respectively. Number recognition is done by pattern matching. The result shows a drastically improvement on the accuracy in comparison with the Ocrad's, and the computation takes less than 0.1 second. On this basis, we provide users a JavaScript for injection, and the captcha on iLMS would be inserted automatically through this method.**

***Index Terms*—number recognition, decaptcha**

I. Introduction

iLMS [1] has introduced the captcha since December, 2020, which is a confusing decision to us. The captcha used by iLMS is obviously easy-to-break by some well trained models such as Tesseract [2], so it has no effect on preventing web crawlers. The extra process, yet, is supremely annoying for normal users. As the extra captcha does not serve its purpose, we try to build a fast, reliable iLMS captcha breaker that is easy to install on any browser.

In order to perform the captcha recognition on browsers with acceptable delay and error rate, developing a method without pre-trained deep learning models is necessary. The usage of pre-trained models leads to longer loading time or complicated installing packages for different browsers. However, the accuracy by deep learning is usually higher than those by any existing OCR based on feature extraction methods. Balancing between efficiency and accuracy, a specific method for the certain website is needed. Therefore, we have to design a new method for breaking iLMS captcha in a fast and precise way.

Fig. 1. Examples of iLMS captcha.

II. Methodology

Imitating the stages of common OCR process, we divided the task into mainly 3 steps: Artificial Noise Removal, Partition, Number Recognition. Once the captcha image and dependencies are loaded, the image will go through these 3 steps, and the result can be placed into the user input node. Following are the explanation of steps list above.

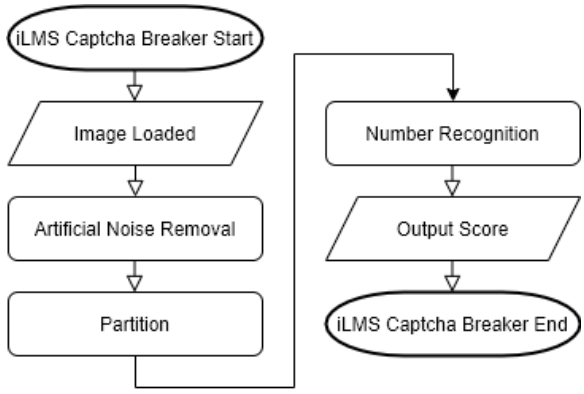

Fig. 2. Process of iLMS Captcha Breaker.

A. Artificial Noise Removal

A color filter is used to reduce the artificial noise of the image, and to generate a clear binary mask for the following steps. The low-pass filter is deprecated, even though the artificial noise seems like the same color from far away. The reason is that the dark blue pixels consist of low value on both red and green, so filtering out pixels out of range can provide an acceptable result. Moreover, using only color filter takes less time and provides a result with higher valid resolution. Considering the benefit, we use the 'inRange' function provided by OpenCV [3] to build our color filter, and the result will be a binary mask.

Fig. 3. Examples of noise removal results.

B. Partition

We partition the mask into 4 target images by some fixed positions for performance concern. On account of the fact that digits in the different target images only shift within 3 pixels, the influence of the shifting pixels is hand to the recognition to overcome. The performance benefit of not computing any seam and the resolvable disadvantage contribute to this decision.

Fig. 4. Digit shifts by some pixels.

C. Number Recognition

The algorithm recognizes a single digit by computing the scores to 9 standard images, and gives the result base on the scores. The shifting pixels is the primary concern of optimization in computing the scores. We define the score equation:

$$score = (coincidence - hamming\ distance) * weight \quad (1)$$

The coincidence is defined as the number of pixels that the target image coincides the standard image. The higher coincidence is, the more resemblance the two images is.

$$coincidence = standard\ image \wedge target\ image \quad (2)$$

The hamming distance can be defined, since both images are binary. The less hamming distance is, the less difference between the two images is.

$$hamming\ distance = standard\ image \oplus target\ image \quad (3)$$

The weight is defined as:

$$weight = \frac{number\ of\ black\ pixels\ of\ standard\ image}{number\ of\ pixels\ of\ standard\ image} \quad (4)$$

The weight is a critical parameter for our scoring equation. The weight solves the problem of shifting pixels and minimizes the clarity request of noise removal, since it provides the idea of probability to the scoring equation. Take the target image is 3, for example. One can expect that 3 and 8 have similar coincidence, and the score only differ by the hamming distance. However, as we mentioned above, pixels shifting may cause the hamming distance vary from ideal. Without the weight adjustment, score of 8 might be higher than 3 in this case. Therefore, adding the weight to score equation is necessary.

There are a couple methods we have tried but deprecated in the process:

- Total number of pixels
- OR - XOR: The total number of discrepant pixels.
- Column of pixels: This computes each column of pixels, try to obtain the distributed pattern of pixels.
- Cut in half: This computes the number of pixels of the left and right side to obtain the distribution of pixels.

In a nutshell, the score equation computes coincidence, hamming distance, and weight, so that the recognition can be done by selecting the highest score as result. In testing, we get the accuracy of 97% for recognizing single number.

III. Experimental Results

It is convenient for users to install our methods. Users can utilize our scripts [4] by arbitrary browser extensions that support JavaScript Injection. Once the installation is done, captcha result will be entered automatically before users can notice.

Fig. 5. Digit shifts by some pixels.

In comparison with Ocrad's [5] result, we have a pleasing result with 87% of accuracy in 100 random tests, exceed the accuracy in Ocrad by 44%. For single number recognition, we achieve 97% of accuracy in 190 random tests.

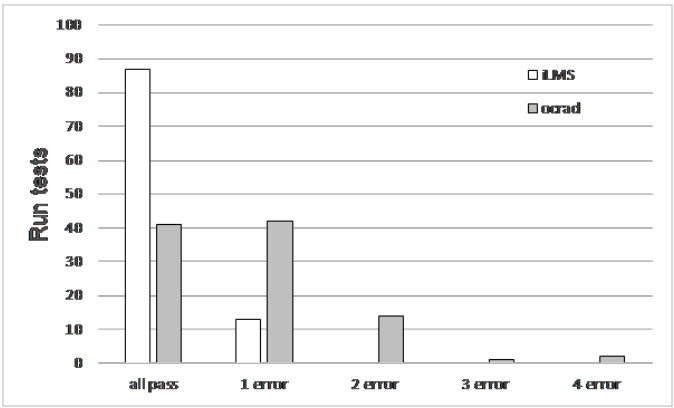

Fig. 6. Accuracy comparison with Ocrad

IV. Conclusion

In this project, we use a color filter to remove noise from raw images, and comparing digits with standard images to compute the scores, which leads to the recognition result. The color filter takes little time to complete and has clear result even in RGB color space, and our scoring computation takes few steps as well. Thus, our method can get the result as fast as the well-developed Ocrad, while double the accuracy for the iLMS captcha.

The method could apply to websites using captcha without distortion, by adjusting color filter, partition position, and standard images. The limitation comes from not using any pre-trained model, yet still possible to resolve by introducing layout analysis.

The bottleneck of speed improvement is the loading time of OpenCV. By rewriting necessary functions, this problem could be bypass relatively easy.

References

[1] NTHU iLMS Login Page Available: “lms.nthu.edu.tw/login_page.php“
[2] Tesseract Available: “github.com/tesseract-ocr/tesseract“
[3] OpenCV Available: “docs.opencv.org/4.5.0/opencv.js“
[4] iLMS-Captcha-Breaker
Available: “github.com/IanLynnEE/iLMS-Captcha-Breaker“
[5] Ocrad Availible: “cdn.rawgit.com/antimatter15/ocrad.js/master/ocrad.js“