

1. Abstract

A Captcha is a type of challenge-response test used in computing as an attempt to ensure that the response is generated by a person. Because other computers are supposedly unable to solve the Captcha, any user entering a correct solution is presumed to be human. [Wikipedia: Captcha]

Our project aims to crack the Captcha system design, that is, to be able to analyze and recognize the Captcha challenge image and return a text answer to the question.

2. Introduction

A Captcha, as Abstract section described, is used to test and tell who is sitting in front of the client computer, a human or a computer? Its system design makes some computer scientists think of the famous Turing test, but in a reverse way: The traditional Turing test is typically administered by a human and targeted to a machine. But Captcha, in contrast, is administered by a machine and targeted to a human. Therefore, it is not hard to imagine that the word, "CAPTCHA", is an acronym based on the word "capture" and standing for "Completely Automated Public Turing test to tell Computers and Humans Apart", coined by Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford (all of Carnegie Mellon University). [Wikipedia: Captcha]

CAPTCHAs are often used in attempts to prevent automated software from performing actions which degrade the quality of service of a given system [Wikipedia: Captcha], as mentioned in the first slogan of Google reCaptcha service: "Stop Spam". For instance, the e-mail spam can be done in a completely automated way, from e-mail account registration to spam e-mail sending. Now, many famous e-mail service providers have adopted Captcha systems to prevent automatic account registration, thus prevent e-mail spam effectively. It is worth noting that many online file sharing services have also adopted Captcha system, together with countdown timers, to prevent massive automatic file downloading.

![Figure 1. Typical reCaptcha Challenge. Image Size is all the same: 300x57](reCaptcha Official Website)

In many Captcha systems, the Google reCaptcha is one of most interesting one. The prefix "re" may mean "re-design" of the Captcha system, as the system now serves not only the role of "Stop Spam", but also the role of "Read Books." "Reading Books" means helping digitize the text of books. This idea was originated with Guatemalan computer scientist Luis von Ahn, aided by a MacArthur Fellowship. As an early CAPTCHA developer, he realized "he had unwittingly created a system that was frittering away, in ten-second increments, millions of hours of a most precious resource: human brain cycles." [Wikipedia: reCaptcha]

Although CAPTCHA and the successor reCaptcha is for the benefit of online service provider and old written texts digitalization, some Captcha systems are designed in an annoying way that people cannot solve and thus unable to access the service.(link [4], [5], and [6] provides some bad and funny Captcha examples) Other kinds of annoying Captcha systems, for example, by putting many CAPTCHA challenges during every step of user's operations no matter which steps should be done by a human or not. These annoying things may kill a website's conversion rate[7], which is proportional to the number of works done by website visitors[8]. From an unknown source, an online file sharing host has gotten rid of Captcha system due to the newly adopted one caused losses of many visitors.

Interestingly, there is a game called "clickclickclick", which also uses Captcha systems to prevent computer program automatically "clicking." The game itself is not interesting at all, but the issues behind the scene are quite interesting, such as network nationalism, network social interaction, and the one related to us, Captcha security system. The following links provides some information about the game "clickclickclick":

http://mmdays.com/2007/06/20/clickclickclick

Our main motivation for this project is trying to achieve automatic behavior of some annoying Captcha systems. For example, to automatically download files from online file sharing hosts without human intervention, or to play the game "clickclickclick" completely automatically.

In addition, some Captcha design researchers also cited many papers related to Captcha cracking, and proposed various guidelines on Captcha design. It seems that cracking Captcha do have some research value.

3. Related Work

Our project idea is originated from 2011 February IEEE computer magazine [Yan and El Ahmad 2011]. This article demonstrated various techniques to crack Captcha systems, and it inspires us to figure out our project.

At first, we brainstormed various topic, such as distributed video codec, accelerate machine learning algorithm, porting PS2-emulator: PSCX2, phenomenon simulation, game, graphic topics, cryptography attack, and so on. Captcha breaking/cracking is a
topic that combines graphic topics and cryptography attack. And the techniques that described in the magazine are quite easy to implement. Therefore, we think that this project is a good choice.

Our pre-implement survey helps us identify common workflow and algorithm to attack Captcha. For example, the workflow often as follows: pre-processing → segmentation → feature extraction → character recognition. [Chandvalle et al. 2009]

For the segmentation part, there are three techniques possible. They are Human Visual System Segmentation [Lin et al. 2008], Color-filling Segmentation [Yan and El Ahmad 2008], and Distortion Estimation Techniques [Moy et al. 2004].

In character recognition, we surveyed a good optical character recognition introductory paper [Mori et al. 1992] that describe “template matching” was the early approach of OCR. And three machine learning/pattern recognition approaches to break visual Captcha. Tidwell and Shadoan used feed-forward neural nets and the self-organizing maps to recognize characters [Tidwell and Shadoan 2008]. Jeff and El Ahmad analyzed many feature patterns of Captchas and used naive pattern recognition to summarize each features’ pattern [Yan and El Ahmad 2007]. Chellapilla and Simard used the same method as Tidwell and Shadoan, neural network to solve Captcha, and they found that once the segmentation problem is solved, solving the Captcha becomes a pure recognition problem, and it can trivially be solved using machine learning [Chellapilla and Simard 2004].

4. Preprocessing (CPU implementation)

Our project aims to crack reCaptcha, therefore, our preprocessing steps are designed specifically to deal with reCaptcha system.

By reCaptcha system design, the Captcha challenge is often consist of two words to be recognized, one for human verification, the other for text digitization.[20] This is confirmed by us, since reCaptcha is a free web service, anyone can get such a service given that the one has a gmail account.

After we trying to solve some of the reCaptcha challenge, we got an observation that the word for human verification has distortion algorithm applied. This also coincides that the official reCaptcha explanation: they use a word already hard for OCR, and distort it more. But it raises a question, how they got correct answers for the words already hard for OCR? A simple inference showed that they must hold some initial answers to the challenge words. And our deeper inference showed that this system can be designed to be self-sufficiency.

We inferred that reCaptcha may already have some initial recognized words, they use them to create distorted words for human verification. Combining with the words that wished to be digitalized by human recognition, a reCaptcha challenge is generated. If the challenge's human verification word is answered correctly, the word for digitization is assumed answered correctly by reCaptcha system. The system then gives the new unknown word image to a number of other people to determine, with higher confidence, whether the original answer was correct. As the number of answers to the unknown new words rises, the system eventually will narrow down possible answers for the new unknown words, and categorized as “recognized words.” [reCaptcha Official Website]

The distortion algorithm seems interesting, they use human's “stereopsis” to generate distorted characters, quite a fashion way, coincident with recent 3D stereoscopic visual effect techniques. By overlapping a character and introduce occlusion, a distorted word is generated.

**Figure 2.** Recent reCaptcha Challenge (2011 June) [reCaptcha Official Website]

Summarize the above observations, we implemented two simple algorithms for preprocessing steps:

1. **bisetReCaptcha:**
   - Separate the two-worded reCaptcha challenge into individual CAPTCHA words to crack.
2. **overlapTemplate:**
   - Generate overlapped and occluded character pattern for later template matching.

Both algorithms are implemented in CPU version, with the aid of Open Computer Vision (OpenCV) library, version 2.2.

4.1 Preprocessing - bisectReCaptcha

From the observation of reCaptcha challenge, we found that it is easy to bisect it into two separated Captcha words because the prominent white areas naturally separate these two words. Most often, the reCaptcha challenge image can be divided by 4 vertical cut (5 sections: left margin, 1st captcha word, separation blank, 2nd Captcha word, right margin). With a vertical line scan through to determine whether the entire line has black filled area or not, we can bisect the reCaptcha challenge image.

**Algorithm 1: reCaptcha Image Bisection**

| BISECT_RECAPTCHA(ImageIn reCaptcha, ImageOut leftCaptcha, ImageOut rightCaptcha) |
|------|------|------|
| MEDIAN_FILTER_3X3 (reCaptcha) | // to get rid of some noise |
| BINARIZE_IMAGE(reCaptcha, threshold=240) | |
| VERTICAL_CURVE_STATE_MACHINE(reCaptcha, leftCaptcha, rightCaptcha) | |

![reCaptcha Image Bisection](reCaptcha_image.png)
Figure 3. Bisect reCaptcha Image Process Visualization. The grayRegion shows binarized reCaptcha Image, The regionMap shows the five region classification, and the red region is a viable bisection cut region.

4.2 Preprocessing - overlapTemplate

As we observed the security of reCaptcha lies in these overlapped and occluded pattern, we should exploit the way to deal with it. Given a y-offset and a pattern, we can generate an overlapped pattern.

Algorithm 2: overlapped pattern generation

\[
\text{OVERLAP\_TEMPLATE}(\text{ImageIn pattern, Integer yOffset, ImageOut overlapped\_pattern})
\]

if \( y\text{-offset} > 0 \)
\[
\text{abs\_yOffset} = -\text{yOffset}
\]
else
\[
\text{abs\_yOffset} = \text{yOffset}
\]
if \( \text{yOffset} > 0 \)
\[
\text{upperone\_yStart} = 0
\]
\[
\text{occluded\_yStart} = \text{abs\_yOffset}
\]
else
\[
\text{upperone\_yStart} = \text{abs\_yOffset}
\]
\[
\text{occluded\_yStart} = 0
\]

\[
// \text{copy occluded 1st}
\text{overlapped}(\text{ROI(0, occluded\_yStart, templateCols, template Rows)})=\text{pattern}
\]

\[
// \text{generate 2nd original upperone copy mask}
\text{MEDIAN\_FILTER}_{3x3}(\text{pattern})
\text{templateWhiteArea} = \text{pattern} > 240 \quad \text{matlab-style vector/matrix}
\text{comparision, this will generate a binary image}
\text{copyMask} = \text{all\_ones(same size as pattern)}
\text{copyMask}(\text{ROI}(0, \text{occluded\_yStart}, \text{templateCols}, \text{template Rows} - \text{upperone\_yStart} - \text{upperone\_yStart})) = \text{templateWhiteArea}(\text{ROI}(0, \text{upperone\_yStart}, \text{templateCols}, \text{template Rows} - \text{upperone\_yStart}))
\]

\[
// \text{copy upperone 2nd}
\text{overlapped}(\text{ROI}(0, \text{upperone\_yStart}, \text{templateCols}, \text{template Rows})) = (\text{pattern} \oplus \text{copyMask})
\]

Figure 4. Overlap Template Result

5. Template Matching (GPU implementation)

The core method we used to recognize each character is the old template matching algorithm. Initially, we don't have any clue on how to recognize characters in the Captcha image. But later we heard that a very basic algorithm in pattern recognition was "template matching." And further paper survey [Mori et al. 1992] also introduce this algorithm. Therefore, we decided to implement it as our core recognition algorithm.

We referenced the wikipedia implementation of template matching. [Wikipedia:Template_matching] In essential, this is a brute-force search and matching method. Pattern will try to superimposing every pixel in Captcha Image and compute the sum of absolute difference (SAD) at current top-left location.

The total complexity is \((C_{\text{cols}} - T_{\text{cols}}) \times (C_{\text{rows}} - T_{\text{rows}}) \times (T_{\text{cols}})\), a very time-consuming method.

(C: Captcha, T: template)

Our initial CPU implementation performance is about 12 seconds.

We take a threshold to filter bad matches, instead of only taking the global maxima position of a single pattern. We call this "candidate matched position," and will be taken to further post-processing (Chap.6) in order to refine and get better combination of answers.

But template matching is a natural parallelizable algorithm, therefore it is very suitable to run on GPU. Our efforts to accelerate this algorithm result in 2 versions of implementation.

1. GPUkernel_singleCaptchaPatternPair
2. GPUkernel_linkedPatterns

Notice that SAD must be normalized, as the size of pattern affect the score a lot.

5.1 GPUkernel_singleCaptchaPatternPair

A simple idea is direct porting to GPU and test if become faster. That is, the kernel takes a single pair of Captcha image and a character pattern, then outputs a distance map recording each comparable position's matching SAD.

In this version, we discovered that the Captcha image is always the same size: 300x57x1 gray-level image. And after bisection, Captcha images can fit entirely into shared memory (at least 16KB), this motivate us to exploit shared memory for speedup.

If we want to compare all patterns in our database, the following pseudo code will show how to compare:

\[
\text{Pseudo Code 1: using GPUkernel_singleCaptchaPatternPair}
\]

```c
for(char ch='a'; ch<='z'; ++ch) {
    for(int padid=1; (pattern=readImg())!=NULL; ++padid) {
        GPUkernel(pattern, captcha, distMap);
    }
}
```

In our experiment, the running time of this version is nearly the same as CPU version. It seems that there are some flaws in this version of GPU implementation.
5.2 GPUkernel_linkedPattern

The previous method has a serious pitfall that it must send patterns one-by-one to GPU device memory, which results in inefficient I/O operations due to frequent interrupts. Besides, it only activates very few threads for a short period of time, which does not reflect the speedup of GPU. Therefore, we decided to improve our GPU version, shown below.

We connect all patterns in our database, and transform the connected patterns into a long one-dimensional array. We bind this array onto the texture memory, in hope of accelerated by texture caching behavior. This essentially eliminates the frequent I/O operations and keeps the number of activated threads high (there are enough data to process).

Now, if we want to compare all patterns in our database, just a single call will make the job done.

This improved version do have speedup. On average, it is faster than CPU version at least 2 times. Sometimes can achieve 4~5 times in variation of the amount of data.

The following shows the final GPU implementation code:

**Pseudo Code 1:** GPUkernel_singleCaptchaPatternPair

```c
__global__ void TMkernel_linkedPattern(...) {
    // compute basic index information
    // copy captcha image into shared memory
    // do template matching
    /* T_rows x T_cols inner for loops complexity */
    /* critical step: */
    /* int diff = tex1Dfetch(patTex,patIdx) - sharedCaptcha[capIdx]; */
}
```

So far, this generates initial answer set. But kind of messy and bad since it does not consider the overall matching score of entire answer string. We can further refine our answer set using genetic algorithm optimization. It is easy to transform this problem into genetic algorithm formulation, as we can take a character as a gene and the answer string as a chromosome. But for the fitness score, as template matching pointed out, also needs to be normalized according to the length of string.

By the way, the visualization code is written in .cu file. The implementer should take care of OpenCV and CUDA linking issues. We addressed this problem by using OpenCV 2.2 (may solve some CUDA linking issues due to OpenCV 2.2 launched a GPU version branch) and CUDA VS Wizard, an open source tool that help us configure GPGPU CUDA Visual Studio projects.

And our connected patterns also include the overlapped patterns, we have a function called generateConnectedPattern, which is response for loading patterns from database, generate overlapped patterns and flatten the patterns into one-dimensional byte array.

6. Post-processing (CPU implementation)

After we got the template matching results, we can analyze and derive initial Captcha answers. First, for every template matching result, we apply a threshold to point out candidate positions. Then, we put these candidate positions into a two-dimensional array of priority queues. The row index is horizontal coordinate (x-axis) of candidate positions. The column index is the letter of candidate positions. The priority is proportional to the matching score (1.0 - "matching distance"). In other words, as the matching score increases, the priority also increases.

We also compute the variance of character patterns’ best match locations, and letter's sum of matching score. For description simplicity, we call these two value as "statistic scores". In hope of better performance, we take thresholds of these two value.

After we built the data structure of candidate positions, we begin to generate 200 initial Captcha answers (word strings). We should use the following pseudo code to describe the algorithm:

**Pseudo Code 2: Pick letters and synthesize answer word strings**

```c
for ansid = 0 to 199 // 200 initial candidate ans
    cumulated_width = 0;
    while(cumulated_width < captchaCols) {
        for each letter c
            if(ansid >= priQ[cumulated_width][c].size() ||
                letterLocVariance[c] > VarThres
                || letterSumMatchScoreOverThres[c] < Thres)
                continue;
            else {
                w = priQ[cumulated_width][c].patternCols;
                captchaAns[ansid].push_back(
                    priQ[cumulated_width][c].top());
                priQ[cumulated_width][c].dequeue();
                cumulated_width+=w > left_overlap ?
                (w = left_overlap) : w;
            }
        end // for each letter c
        if(old_cumulated_width == cumulated_width)
            ++cumulated_width;
    end // for ansid = 0 to 199
```

Figure 5. Distance Map Visualization. We use spectrum visualization method. Green or Blue means large distance, and red means small distance. Red circle indicates global maxima (best matched position)
7. Result

As a solved Captcha, the answer string should pass through the Captcha system’s answer checker. But eventually, all above Captcha breaking schemes can’t even solve a Captcha completely. That is, our breaker has 100% miss rate in our gathered test data, although we didn’t put our program into real test.

The following is the most closing answer strings during the run of GA post-refinement. We have the best record of only get 2 letters wrong. Still, our work is not in vein.

![Candidate Captcha answer listing. Background is terminal messages. Upper image window shows current solving Captcha image. Lower image windows shows matched template pasting.](image)

A typical genetic algorithm requires to define 4 operations, they are:
- fitness score evaluation
- breeding parents selection
- offspring generation by crossover
- random genetic mutation

There are other kinds of methodology definition of genetic algorithm [Wikipedia: Genetic algorithm]. But our refinement follows above definition, except that we do not implement mutation part.

For the fitness score evaluation, we simple add each character pattern’s matching score, and divide by the answer string length.

For the breeding parents selection, we use tournament selection. Tournament selection involves running several "tournaments" among a few individuals chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for crossover. Selection pressure is easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected [Wikipedia: Tournament selection]. In essence, just like drawing a fix number of samples, and pick the one with maximum fitness score and place into breeding pool.

For offspring generation by crossover, we examine each parent’s overlapped part of answer string letter-by-letter, and copy the letter with maximum matching score to the child answer string. This will generate an answer string which always has better fitness score among the parents’ answer string. Then we random pick a parent answer string for tail concatenation. This may lower the fitness score of child chromosome, but the impact is not noticeable.

In our GA CPU implementation, the convergence speed is quite fast. It converges within five generations on average. But unfortunately, the convergence just result in 200 same erroneous answer strings. We investigated our GA algorithm and found that there are some answer strings that are really quite closing to the answer, but there are more answer strings that are very bad....

8. Conclusion

After this project, we know that Google reCaptcha system is really a robust anti-spambot service. We even doubt that even if we could crack the reCaptcha, they can just change the distortion algorithm within a day and make our breaker broken. Well, should we face the unbelievable truth?

During this project, we googled many web pages that claim to had cracked Google reCaptcha. At that time, My partners and I had high hope in achieving good performance, but our progress was slowed down by some of GPU programming bugs. After we finally come up with Captcha answer strings, the results were just disappointing. Despite this, we fought for our GPU Captcha Breaker until the last moment, until the advent of demo time. I was very grateful to my teammates, they are really trustworthy and skillful. Just as our team name: Team robust skill guy! And special thanks to the rating committee from Microsoft Taiwan, We never thought we are deserved to have a XL T-shirt prize!!
References

http://en.wikipedia.org/wiki/CAPTCHA
http://www.google.com/recaptcha
http://en.wikipedia.org/wiki/reCAPTCHA
http://www.docstoc.com/docs/1048763/Worst-Captchas-of-All-Time
http://depressedprogrammer.wordpress.com/2008/04/20/worst-captcha-ever
http://en.wikipedia.org/wiki/Conversion_rate
http://www.seomoz.org/blog/captchas-affect-on-conversion-rates
http://mmdays.com/2007/06/20/clickclickclick


TIDWELL and SHADOAN, Machine Learning Approaches to CAPTCHA Recognition Requiring Minimal Image Processing, CS5033 2008 Fall, Machine Learning Course Final Project, instructed by Amy McGovern, University of Oklahoma (URL: www.cs.ou.edu/~amy/courses/cs5033_fall2008/Tidwell_Shadoan.pdf)


http://stackoverflow.com/questions/1435696/how-does-recaptcha-work
http://en.wikipedia.org/wiki/Template_matching
http://en.wikipedia.org/wiki/Genetic_algorithm
http://en.wikipedia.org/wiki/Tournament_selection