COMS 6998 Computational Photography Spring 2009

BREAKING AN IMAGE BASED CAPTCHA

FINAL REPORT

Michele Merler (mm3233)
Jacquilene Jacob (jj2442)

INTRODUCTION

A CAPTCHA is "Completely Automated Public Turing test to tell Computers and Humans Apart". An image of distorted letters is dynamically generated. Since the letters are a part of an image and not text, it is difficult for a spam bot or other computer program to read. A human, in fact, has little trouble reading the letters in a captcha image. Using a captcha test on a website is a great way to ensure, for instance, that a person and not a spambot is filling out a web form. For example, humans can read distorted text as the one shown in Figure 1, but current computer programs cannot.

Captcha is a so called win-win solution, in that if a bot cannot break it, it provides security, but if it is automatically broken, that means that a difficult task in computer vision or related areas has been solved.

The problem with current visual text based captcha systems is that most of them have proven to be either not robust enough (they have been broken) or they are too complicated or annoying to read even for humans.

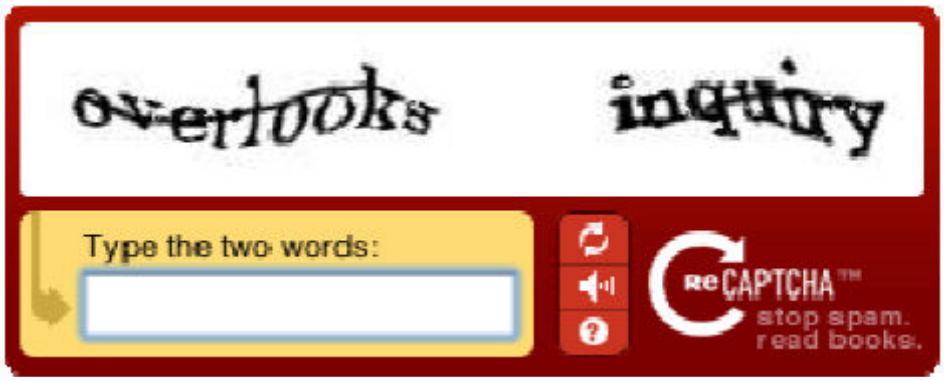

Figure 1 : Example of text based captcha. Source reCaptcha[1]

VidoopCAPTCHA

VidoopCAPTCHA[2] is a verification solution that uses images of objects, animals, people or landscapes, instead of distorted text, to distinguish a human from a computer program. By verifying that users are human, the site and users are protected against malicious bot attacks.

[1] http://recaptcha.net/

[2] http://vidoop.com/captcha/

VidoopCAPTCHA is more intuitive for the user compared to the more traditional text based CAPTCHAs. It then presents itself as the solution to the current captcha problems.

As shown in Figure 2, a Vidoop challenge image consists in a combination of pictures representing different categories. Each picture is associated with a letter which is embedded in it. In order to pass the challenge, the user is asked to report the letters corresponding to a list of required categories. The robustness of the approach relies in the fact that object recognition is a straightforward and fast to solve task for humans, whereas for a computer it is a fundamentally hard problem. In fact, it has represented for many years and still represents a topic of active research in computer vision. What the authors underestimate, though, is that since a bot can try to access a service thousands of times in a day, recognition rates which are considered quite low by the object recognition community (50% or 60%), still would allow automatic attacks to services protected by the image captcha to be considered fully successful.

Figure 2 : Example of an image challenge from VidoopCatcha

OBJECTIVE

The core idea of the project consists in trying to break an image based captcha, and in particular VidoopCAPTCHA, following in the line of work initiated by Mori and Malik[3].
The objective of this system is to show that image based captchas, and in particular the vidoop one, are not as secure as their authors claim. This automatically leads to insecurity for the different applications using the image captchas. We chose this idea in order to show our concerns in today's world where the security methods developed to preserve confidentiality in online systems, of which image based captcha represent of the latest developments, are not only insecure but are prone to attacks by hackers with high success rates.

ALGORITHM

The proposed algorithm consists in the following parts (visualized in Figure 3):

- Test image preprocessing: isolate the single pictures within the challenge image and extract their corresponding
characters regions with some simple image processing
- Image category recognizer: classify the images according to the categories required by the test
- Character recognizer: extract the characters corresponding to the images classified as being part of the required categories

Data acquisition

We wrote a Perl script to download 200 Vidoop challenges from their website. The images can be found from our project page, together with .txt files containing the correspondent categories required by the challenge and manually annotated ground truth letters that actually solve the test, and the results of the split and letter detection algorithm,.

We discovered that only 26 categories are used in the challenges. Their distribution can be observed in the graph of Figure 4.

We also wrote another Perl script to download images from Flickr for every category, in order to use them as training data. In this context, we decided to download 500 images per concept, a number large enough to train a fairly robust classifier, but small enough to prevent too many

[3] http://www.cs.sfu.ca/~mori/research/gimpy/

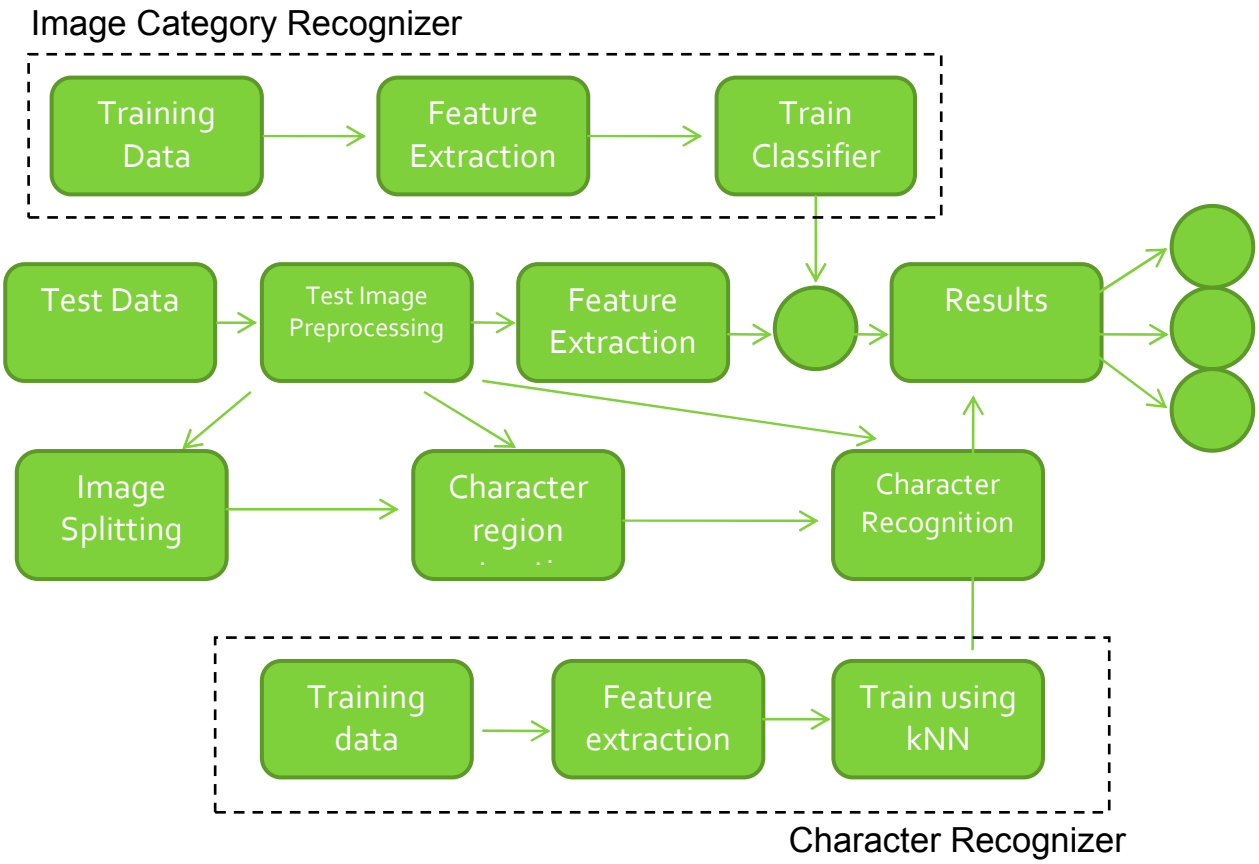

Figure 3 : Process flow of the algorithm

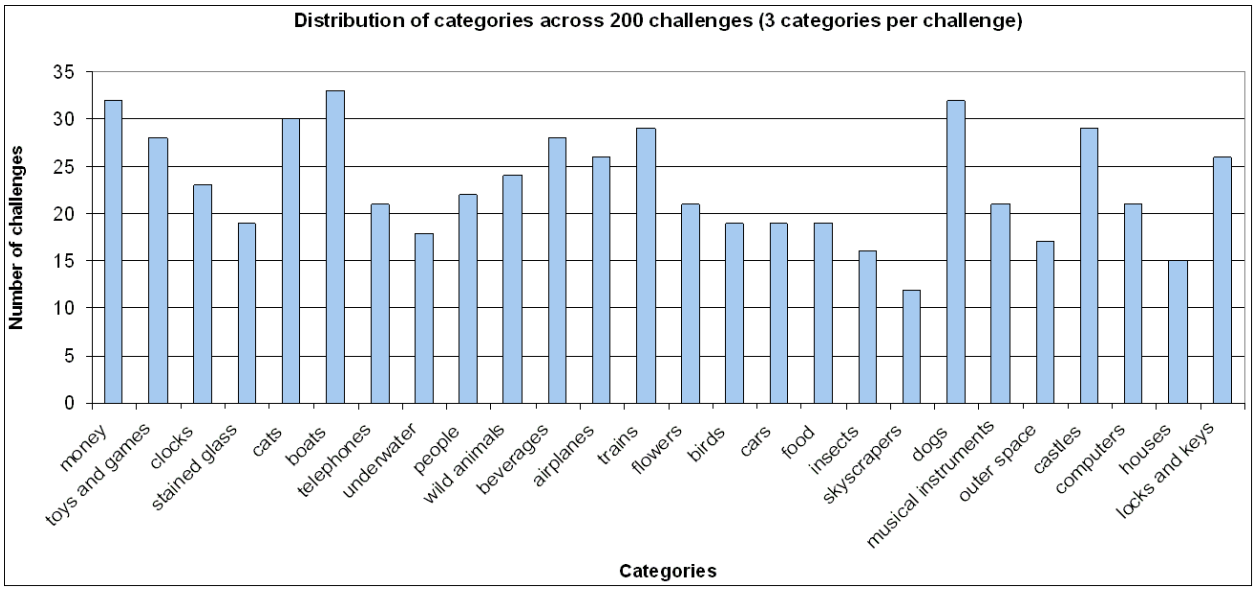

Figure 4 : Distribution of 26 categories across 600 requests in 200 Vidoop challenges

noisy examples to be in the training set (even though we still found noisy images in the training set). In fact, downloading images from Flickr allows to automatically obtain a large scale of data, but many examples might not be relevant to the given query. Flickr's query system relies on users tagging or other text labeling of the images, rather than on their actual content, therefore mislabeling by users can lead to errors, which increase as we proceed to lower rankings in the returned list of results.

Test images preprocessing

The goal of this step is to split each challenge image into the correct subimages, and then localize and extract the circular region containing the character within each subimage. The split algorithm we use is based on localizing vertical and horizontal lines containing the maximum number of edges in the edge image obtained by applying a Laplacian of Gaussian filter to the original challenge image. Once the image had been split into the subimages, a generalized Hough transform (we found the code here[4]) is computed on each subimage to detect circular regions. The circular region which is detected in most of the subimages in approximately the same position and with same radius is kept to be the character's region. Finally, the rectangle with equal sides of length l = r*sqrt(2) inscribed in the localized circle of radius r is the final character region, which is then rescaled to a size of 27x27 pixels to match the training data and thresholded into a binary representation. The algorithm, while being simple and a little bit as hoc, is quite effective. In fact, it splits and segments subimages and text regions with 100% accuracy. Figure 5 presents an example of the processing chain for Challenge1.

Character Classification

We built a 1-Nearest Neighbor classifier (we used code form Piotr Dollar's ToolBox[5]) from a training data consisting of 64 27x27 binary images per capital letter, generated with the GD library. Each image contains a capital letter printed with one out of 20 commonly used fonts (Arial, Times New Roman, Courier, etc.), also with bold and italic variations. Figure 6 contains some examples for the latter "A".

[4] http://www.mathworks.com/matlabcentral/fileexchange/9168
[5] http://vision.ucsd.edu/~pdollar/toolbox/doc/

As features representing each image letter we used the simple binary vectors built by concatenating image lines one after each other, thus obtaining a 1x729 feature vector per image, in a similar fashion to what is done for the MNIST[6] handwritten digits dataset.

The 1-NN classifier proved to be quite effective on the text regions (12*200 = 2400) extracted from the Vidoop challenges, achieving 96% accuracy.

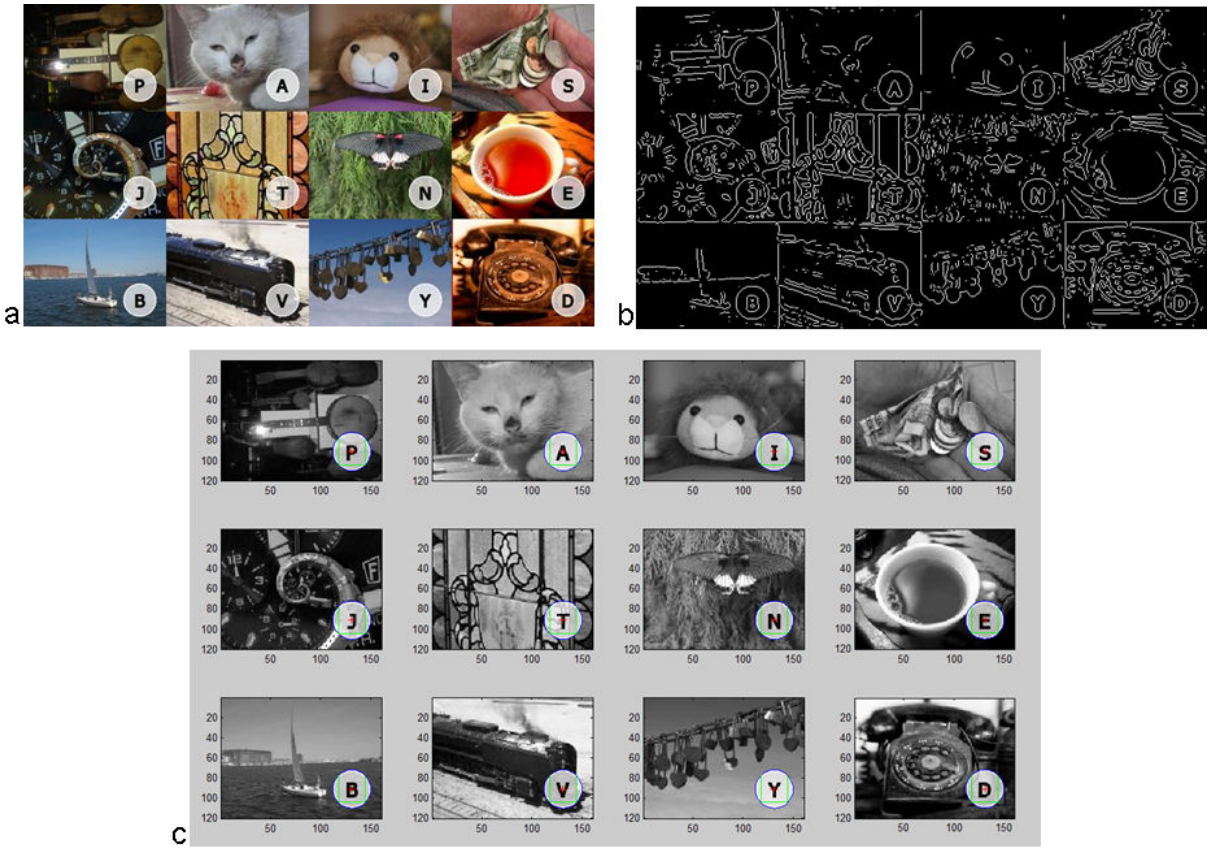

Figure 5 : Preprocessing chain: a) original test image, b) LoG based egde image, c) split and circle detection result.

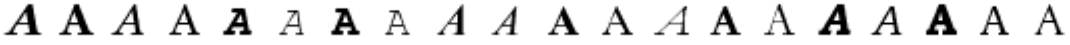

Figure 6 : Examples of training data for 1-NN character classifier

[6] http://yann.lecun.com/exdb/mnist/

Features Extraction and Image Category Classifiers

We used and tested 4 types of features to build our image category classifiers:

- Edge Histograms (6x8 regions)
- Color Moments (RGB, 3x3 regions)
- Color Histograms (32+32 bins in CbCr)
- GIST features (314 dims. vectors)

Edge histograms were computed by simply applying a LoG filter to the grayscale image, split the edge image into 6x8 regions and then counting the number of edges in each region.
Color moments were 1x9 vectors computed as in Assignment 2, with mean, variance and skewness of 9 subregions of the image computed for each of the RGB channels.
Color histograms were computed as 32 bins histograms in the Cb and Cr channels from YCbCr and concatenated into a 1x64 feature vector. The chrominance plane CbCr was used to partially avoid dependence from illumination changes and instead extract pure color information.
GIST features are richer descriptors combining bith color and intensity information, computed as responses to filters applied at different orientations and scales, and to different subregions of the image itself. We used the implementation from Antonio Torralba at MIT[7] with default settings, which produced 1x314 vectors.
For each type of feature and for each category, we trained an SVM classifier from the ~500 positive examples downloaded from Flickr for that category and ~500 negative examples randomly selected from the other 25 categories.
Figure 7 presents the image category recognition performances on the 200 test challenges of the SVM classifiers built on the 4 feature types. The best performing (as expected, since it's a richer descriptor) is GIST, which recognizes 1 out of the 3 requested images on 79 challenges, 2 out 3 on 54 challenges and all 3 requested images on 7 challenges, for a total of 79 + 54*2 + 7*3 = 208 correctly recognized images out of the 600 (200*3) requested. GIST has therefore a classification accuracy of 34.7%. The worst performing is the Color Histogram based classifier, with accuracy of 14.8%.

[7] http://people.csail.mit.edu/torralba/LabelMeToolbox/

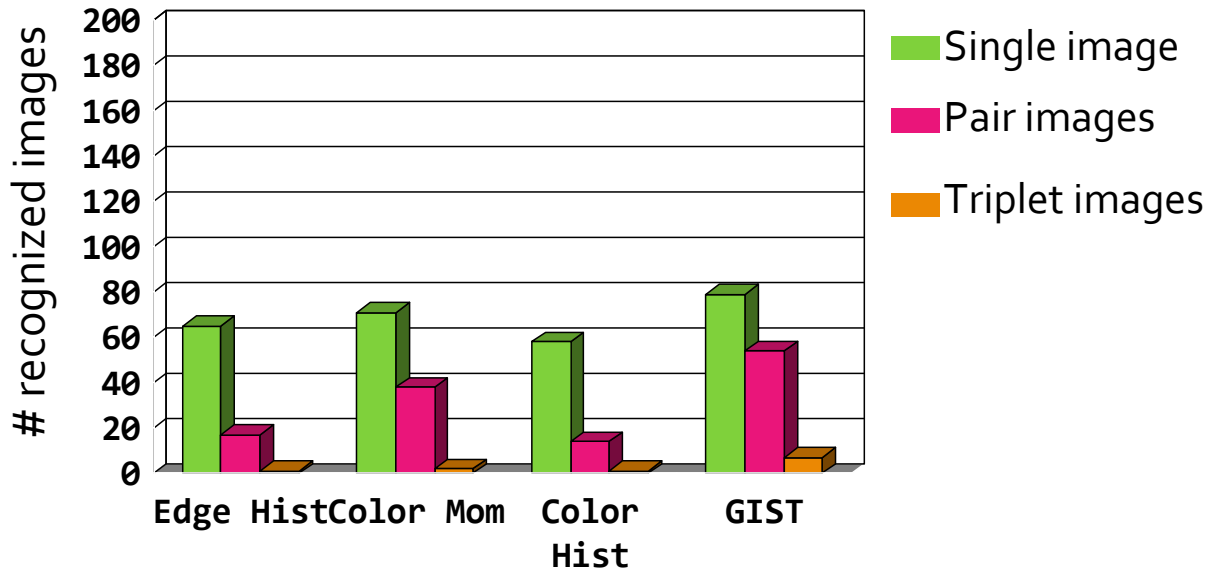

Figure 7 : Image category recognition results

Final Results

Final challenge break performances are reported in Figure 8, where again GIST based classification allows the system to break 6 (not 7 as Figure 7 suggests, because of an error of the character recognizer) challenges out of the 200 tested. The Captcha breaking rate of our system is therefore 3%. This results, which at first glance may not result quite exciting, is acceptable enough to consider the Image based Captcha proposed by Vidoop not secure, for 2 reasons.

The first reason is that if we compare our performances with random choices, which are in the order of $1/12^3$, we see that our method performs definitely better than a random brute force baseline.

In second instance, if we consider that when using a GIST based SVM category recognizer, the average processing time per challenge is 12 seconds, we can see we are able to break 9 image captchas per hour, and 216 in a day. If we further consider that our implementation could be made more efficient by using a faster programming language than Matlab (for example C++), it is clear that the performances are more then sufficient to consider the captcha broken, or at least not safe.

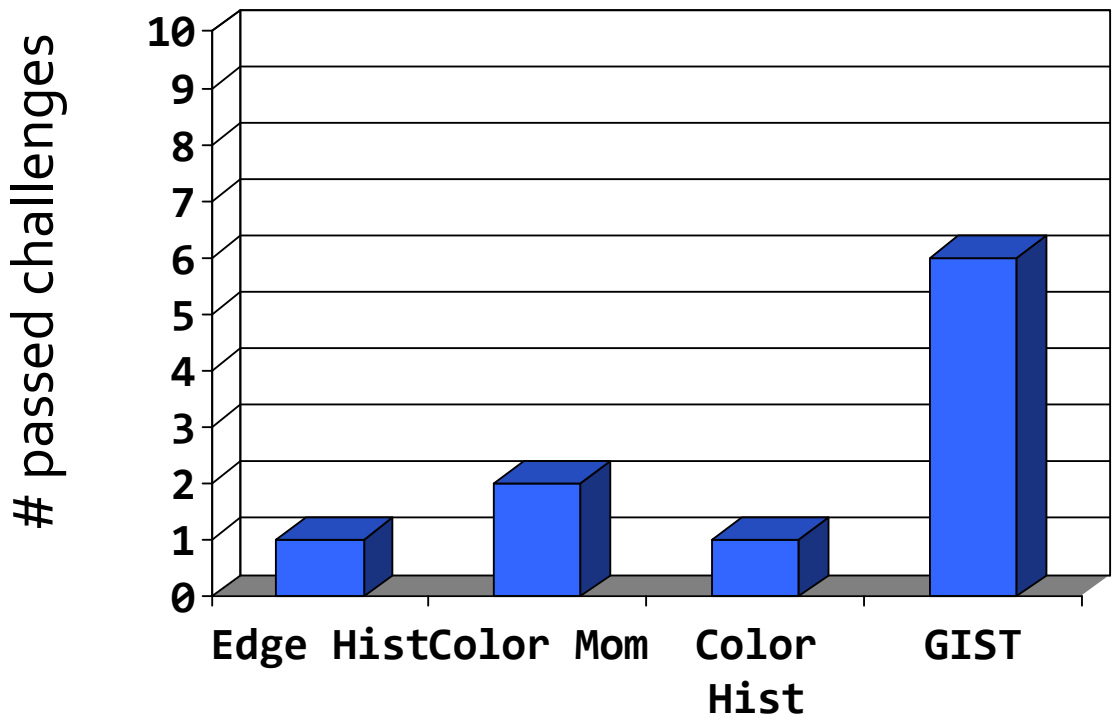

Figure 8 : Test breaking results

Conclusions and Future Directions

We proposed and implemented an automatic system able to break an Image based Captcha, specifically the on proposed by Vidoop. The experimental results show that our system is able to break 9 challenges in one hour, and could be potentially improved to be even more efficient and effective.

Therefore, the Vidoo image based captcha is not secure, as its authors claim.

There are a few things that time constrains did not allow us to test, and which would be interesting to try and could improve our system:

- Try other features (SIFT + codebook)
- Obtain cleaner training data (performances suggest poor training data)
- Improve speed and efficiency using more powerful programming languages
- Test online version of Captcha breaker

As a final remark, we had a lot of fun working on this problem. Even though the performances are not explosively good, we feel that we managed to prove our point. Both of us spent a good deal of effort in building the system, and were gratified by seeing the results. We could experiment and try different algorithms that maybe we would not have considered or used if not challenged by the class project. Of course, we realize that the main nature of our work has been "destructive", in the sense that we just showed that other people's work are not robust, but we also think that thanks to this experience and to recognizing the weaknesses of existing systems, we might be able to come up with a more robust Captcha system.

Code

Data acquisition

- flickrDownload.pl
- vidoopDownload.pl
- testConceptStats.m

Test image preprocessing

- processTestImages.m
- imresizecrop.m
- CircularHough_Grd.m
- DrawCircle.m

Features extraction and image category classification

- training.m
- extractGistFeatures.m
- createGabor.m
- LMgist.m
- extractColorMoments.m
- extractEdgeHistogram.m
- prepareData.m
- svm-predict.exe
- svm-train.exe

Character classification

- characterTrain.m
- clfKnn.m
- clfKnnDist.m
- clfKnnFwd.m

- clfKnnTrain.m
- pdist2.m

Testing

- testChallenge.m