Breaking Microsoft's CAPTCHA

Colin Hong Bokil Lopez-Pineda Karthik Rajendran Adrià Recasens

May 2015

Abstract

In this work we analyze the weaknesses of the CAPTCHA scheme used by Microsoft in their registration website. Furthermore, we propose some alternatives to mitigate the weaknesses found. In the first part of this project we built a CAPTCHA data set and used two different methods to automatically predict the CAPTCHA content, achieving a 57% success rate by using state-of-the-art techniques in computer vision. Finally, we propose some methods to overcome the vulnerabilities found. After a review on existing methods, we propose two alternatives: visual question-answer challenges and motion-based CAPTCHAs.

1 Introduction

Distinguishing humans from computers have been an important topic since the birth of Internet. Many systems become vulnerable to a wide range of attacks when computers are able to automatically act as humans. Thus, systems to verify whether the agent submitting the petition is a human or a computer have been evolving during the last decades. In this project, we will focus on one of the main well-known and widely used proposal: CAPTCHA.

CAPTCHA stands for "**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part". It was proposed by Louis Von Ahn et al. [1] in 2003, and since then has been widely used in a wide range of places. Since then, multiple variations [2] have been proposed but always keeping the same basic idea: to show to the user images that humans are able to recognize while computers are not. The use of a wide variance of shapes, transformations and deformations to write the letters make the CAPTCHA recognition a complex problem for computers while humans are easily able to recognize the characters.

However, during the last three years, the computer vision state-of-the-art has been improving constantly due to the emergence of a new actor: the Convolutional Neural Networks (CNNs), commonly named as deep learning. Deep learning techniques were proposed some time ago, but the lack of large data sets to be trained on hurt their performance. Recently, the computer vision community started to present larger data sets [3–5], which allowed these techniques to be redefined and optimized. Thus, the state-of-the-art in all the computer vision problems has been improving during the last few years. In particular, text recognition systems based on deep learning achieve outstanding performance, being almost perfect at recognizing a wide variations of characters.

Given this recent breakthrough, the ability of computers to recognize CAPTCHA sequences has significantly increased, weakening the security of many services. In this project, we study the security of the CAPTCHA system used by Microsoft on its account creation website. We show that it is feasible with actual technology to automatically overpass the human-verification system used by Microsoft.

Some proposals [6, 7] have been made to secure CAPTCHA schemes against text recognition systems. The substitution of text with more complex visual inputs such as images or motion-based text makes the recognition task harder for computer and still easy for humans. To close the loop, we propose some possible improvements to the Microsoft CAPTCHA to avoid the attacks presented in section 3 and section 4. In particular, we use a recent problem introduced to the computer vision community, Visual Question-Answering [8], to propose a stronger CAPTCHA scheme, hard to be solved automatically.

Furthermore, we also propose a motion CAPTCHA scheme, where it is shown to be hard for attackers to get a clear crop of the target characters.

To sum up, in this project we analyze the security of the CAPTCHA scheme used by Microsoft, showing clear weaknesses that could be potentially exploited to create a large amount of accounts automatically. In section 3 we use a simple method based on the creation of templates. This method shows some weaknesses of the system, achieving an accuracy of 5.56%. Furthermore, in section 4, we propose a deep learning-based method which is able to achieve a success rate of 57.05%. Finally, in section 5 we propose some improvements to the actual CAPTCHA system, in order to make it secure against the attacks proposed in section 3 and 4.

2 Related work

Multiple proposals have been done since the birth of CAPTCHA to automatically pass the challenge and, therefore, gain automatic access to multiple services such as account creation. Even though the challenge is primarily visual, a wide range of attacks have been proposed, using weaknesses from both the visual scheme or the protocol running behind it.

In this section, we will detail the main related work on attacking CAPTCHAs. We will divide the related work in two separated parts: the attacks using vision-based techniques and the attacks using weaknesses on other parts of the pipeline such as the HTTP protocol or the audio system provided for vision-impaired users.

2.1 Computer vision-based attacks

Many approaches have been proposed to attack CAPTCHA systems using visual recognition techniques [7, 9–14]. As it has been presented in the introduction, the state-of-the-art methods are mainly based in deep learning techniques. However, some previous successful attempts have been done to break CAPTCHA systems. In this section, we want to review all these different techniques.

Shape has been deeply studied in the object recognition context. [9] uses some of these ideas to attack Gimpy and EZ-Gimpy, two visual CAPTCHA schemes. In particular, they use the shape features descriptors introduced in [15] to build a matching system between a large database and every new query example. By comparing the shape features, they are able to find the best matching example in the data set and, thus, predict the characters in the CAPTCHA image. Using these approach they achieve a 92% success rate on EZ-Gimpy Images, and an accuracy of 33% on Gimpy images. Our first approach is similar to [9] in spirit; although they have a more complex representation of the CAPTCHA image, our method also matches a given sample with examples in a database.

Much effort has been historically focused on the CAPTCHA segmentation problem. Although the single CAPTCHA character classification accuracy is high for many recent machine learning classification techniques, a previous character-by-character segmentation needs to be done in order for these techniques to be applicable to the CAPTCHA problem. [10] presents a full segmentation scheme for an old Microsoft CAPTCHA, already deprecated. The scheme in [10] attempts to automatically segment character by character and remove the noise present in the final segmentation. After the segmentation step, a regular state-of-the-art classifier is used, achieving a success of 61% in Microsoft CAPTCHA and 60% in Yahoo! CAPTCHA.

After the raise of the deep learning techniques applied to the computer vision problems, some of these proposals have been applied to CAPTCHA classification. Recent work in digit recognition for street-view images [11] has been applied to the CAPTCHA framework with great success. [11] models the probability of a sequence by factorizing it and training a Convolutional Neural Network to estimate these probability factors. It is important to note that by modeling the problem as sequence prediction and not character prediction, they avoid the segmentation step. By using this technique, [11] achieves a 99.8% on reCAPTCHA [2]. Our CNN-based system shares some characteristic with this

method. It uses a Convolutional Neural Network for classification and, furthermore, it somehow avoids complex segmentation procedures by using the power of the CNN.

Figure 1: Example of an Asirra CAPTCHA

Finally, some computer vision techniques have also been applied to image-based CAPTCHAS. As it will be detailed in Section 5, multiple modalities of CAPTCHAS have been proposed, substituting the text recognition task by object recognition or another more complex task. However, state-of-the-art classification algorithms achieve high accuracy in some of these tasks. [7] presents an attack to the Asirra CAPTCHA [16], a CAPTCHA scheme based on the human ability on distinguishing cats from dogs. By using classification methods techniques such as Support Vector Machines, [7] is able to achieve a success rate of 10.3% in the Asirra CAPTCHA.

2.2 Alternative attacks

This section presents some of the numerous attacks on CAPTCHA done using techniques such as external human translation or speech recognition. In contrast to the techniques based upon computer vision principles, these target other components of the security system and do not involve breaking the CAPTCHA directly.

The CAPTCHA Re-Riding Attack [17] uses vulnerabilities found in the HTTP protocol to bypass the security check. Attacks on the HTTP protocol can be classified as client-side and server-side. When clients are trusted with CAPTCHA verification and storage, they also gain the ability to access solutions and generate CAPTCHAs of their own choice which leads to a chosen CAPTCHA text attack, an example of a client-side attack. When static CAPTCHA identifiers are used by the server to map CAPTCHAs, the CAPTCHAs can be downloaded, solved and a rainbow table can be created containing the static identifiers and corresponding solutions, this is a example of a server-side attack.

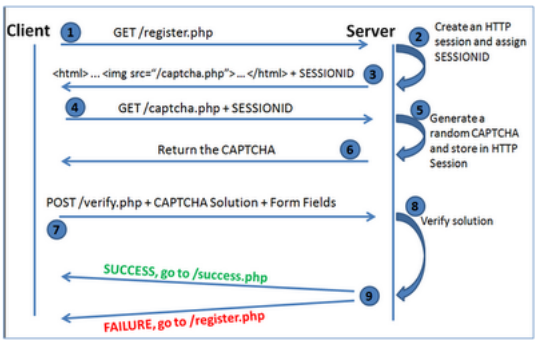


Figure 2: Example of a secure CAPTCHA system implementation as described in [17]

Audio CAPTCHAs were introduced to cater to visually impaired people who weren't able to access standard CAPTCHAs based on visual-perception tasks, but their security was never tested. A recent attempt [18] to break audio CAPTCHAs by a group of

students from Carnegie Mellon University yielded a success rate of 71%. An audio CAPTCHA consists of several speakers reading a character at random intervals. The sequence must be correctly identified to pass the CAPTCHA. The student group used several machine learning techniques such as mel-frequency cepstral coefficients (MFCC), perceptual linear prediction (PLP), and relative spectral transform-PLP (RAS TA-PLP) to extract features from the CAPTCHA audio. The audio files were converted to fixed size labeled segments and then used for training. Some of the classifiers used include AdaBoost, Support Vector Machines and k-nearest neighbor. An audio CAPTCHA possessing multiple speakers and background noise was concluded to be strong but not unbreakable.

An attack which makes use of a human translation service is Podec [19], which is reportedly the first program able to bypass CAPTCHA targeted at Android devices. It passes the CAPTCHAs encountered to an online human translation service in real time. The program signs the user for paid premium services and bypasses the Advice on Charge system, which requests the user for payment authorization. Not much is known about the implementation since the developers used code obfuscation, encryption and garbage classes. The whole application relies on the user downloading the application and granting privileges to it.

3 Template-based CAPTCHA classification method

Although multiple and complex proposals have been done for image classifications, in this section we attempt to solve the CAPTCHA classification problem with a simple approach. We believe this first attempt is useful in order to understand the insights of the problem and, thus, be able to correctly define a more complex strategy as the one presented in section 4.

The template method used a brute-force comparison to classify each character in series. "Brute-force" in this case describes the pixel-by-pixel comparison of the test character to each character in the training set. The final character classification simply defaults to the highest-match character.

This method has an individual character success rate of 60%, with an overall CAPTCHA success rate of 5.56%. This is ample success for the CAPTCHA to be considered "broken", given the weak limitation of the scheme against retries. Time analysis of this method is not a consideration, as the comparisons are trivial. In fact, the program is limited by the speed of the debugging "print" statements.

It should be added that this success was achieved with a database of only 144 distinct character images across 23 different characters. While a larger database does not necessarily translate to a large boost on performance ("5" vs. "6" vs. "S"), more samples would certainly help to reduce noise.

Finally, it is important to note that the CAPTCHA scheme only uses 23 characters from a potential pool of 62 (uppercase letters, lowercase letters and digits). This reduction on the number of categories reduces the dimensionality of the problem and its complexity.

3.1 Framework

We used PIL (Python Image Library) + numpy to process and manipulate images on a pixel level.

3.2 Data Preparation

The original CAPTCHA is unfit for processing as-is; the characters are rotated at random angles across CAPTCHAs (consist angles within a CAPTCHA). This presents two problems for would-be immediate processing. First, any vertical crop of a character would contain significant portions of another character, depending on the angle of rotation. Second, the data set would have to be significantly bigger to account for the different angles of rotation.

Therefore, a certain amount of pre-processing is required. Below is the high-level code that prepares a CAPTCHA.

```
def prep_data(filename, randnum):
    print "Starting to prepare data"
    filepath = '../../test/original/'+filename
    print "Opening original file at:" + filepath
    im = Image.open(filepath)

    captcha = Captcha(im)
    captcha.filter_image_red()
    captcha.find_original_corners()
    captcha.rotate_image()
    captcha.reset_minmax()
    captcha.find_rotated_sides()
    captcha.recreate_rectangle()
    captcha.filter_image_black()

    captcha.save_image(randnum)
    print "Finished preparing data"
    #captcha.im.show()
    return captcha.im
```

Figure 3: High-level captcha prep script

- **filter_image_red()**: The original CAPTCHA is almost uniform in color. However, there are slight discrepancies in RGB values, possibly due to JPEG compression. This function assigns all the non-white pixels to be red. This simplifies further processing, as we only need to deal with two pixel categories, red or white. Red is used instead of black to distinguish the text pixels from the border pixels in recreate_rectangle().

- **find_original_corners()**: This function finds the max_x, min_x, max_y, and min_y points of the CAPTCHA. These will be used during rotation.

- **rotate_image()**: The CAPTCHA is either oriented with the left corner at max_y, or the right corner at max_y. Simply querying whether the x component of max_y is greater than or less than the x center of the CAPTCHA is enough to determine the orientation. The angle of rotation is determined by finding the Δy and Δx between min/max x and min y, and applying atan().

  *note: a side effect of rotation is a new image frame that looks like:

Figure 4: Post-Rotation Image

- **reset_minmax()**: Exactly as the function's name implies. This resets the min x/y and max x/y values to placeholders (100000 and -1 respectively).

- **find_rotated_sides()**: Similar to find_original_corners(), this function does not store points, just rather min and max values for x and y. These will be used in recreate_rectangle().

- **recreate_rectangle()**: Unfortunately, simply cropping the image does not work. In rare cases, usually involving the letter "d" or "p" or "y", as these can have irregular "stems" that cause a corner of the bounding box to be outside the image itself.

  The workaround is to simply create a new rgb data array[], populate it with all valid pixels from the original CAPTCHA, populate the rest with white space, and then convert it into an image format and save it.

- **filter_image_black**(): Assigns all the non-white pixels to be black. Black is used for human visibility only.

Below is a comparison between an input CAPTCHA and the final prepared CAPTCHA:

Figure 5: Captcha before processing

Figure 6: Captcha after processing

3.3 Pixelation

The pixel-to-pixel comparison function requires both character images to be the same width/height. Simple experiments determined a 16 pixel by 16 pixel image was sufficient to retain important characteristics. It was important to descale the image, because for each additional row/column k, it increased the time complexity by $(2k + 1)n$, where n is the number of data set character images.

The actual pixelation was done by dividing the original character image into the 256 cells, and determining the average value of each cell (1=black, 0=white). If the average value of each cell was greater than 0.5, then the script set the corresponding pixel in the pixelated image to black, and vice versa.

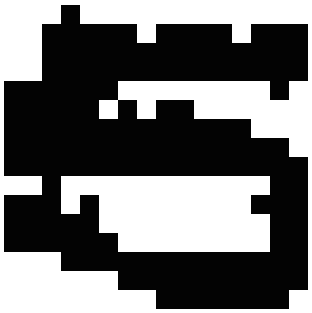


Figure 7: Pixelated "5" character

3.4 Data Set Generation

As mentioned in the overview, the methodology uses direct pixel-by-pixel comparisons of the test character and all training set characters to classify. This thus assumes that the data set characters are properly generated.

Guidelines for the Data Set:

- All widths must be exact, with no white pixels near the left or right border. This is due to the pixelation process, which would disproportionally represent one edge pixel as 1/16 of the entire width.
- The height can be arbitrarily determined. The cropping function sets the top and bottom of the character image as the top black and bottom black pixel. Thus any white space will be eliminated. However, if another character bleeds into the target character, it is important to set the height at the top and bottom edges of the target character only.

Unfortunately, the width specification prevented automatic data set generation. All images must be captured by hand, classified in the name of the file, and then put into a loading directory. We then wrote a simple script to iterate over all files in the loading directory, generate the 16 pixel descaled image, and sort it into the correct directory.

Our method reached the success rate of 5.56% with a data set of only 120 characters. This meant each character had only about five or six "perfect" images with which to compare.

3.5 Brute Force Script

First, the data is processed as described above in data processing. The resulting image is similar to the one shown in Figure 6.

3.5.1 Border Iteration

As the script will automatically crop the top and bottom excess from a test character image, the hardest part of classification is choosing the correct left and right borders, from which to generate a pixelated image.

The script gets the width of the CAPTCHA from the min x and max x values, and divides by six to get the average character width. However, many characters such as "V" or "y" are under the average character width, while characters such as "W", "G", and "K" often exceed the average character width.

A pixel by pixel comparison of "W" might incorrectly classify it as "V" if the borders are not selected correctly. Therefore, we iterate over nine possible widths, [avg width-4, avg width+4].

3.5.2 Comparison

For each width i, we descale the corresponding character image to 16×16 pixels.

We then compare the generated 16×16 pixels test image to each data set image. For the comparison, we simply iterate over the 256 (x,y) pixel pairs, (testIm(x,y), dataSetIm(x,y)). If either the test image or data set image has a black pixel at (x,y), we check for a pixel color match. This eliminates false positive matches for blank space.

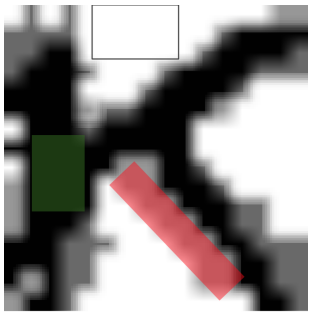


Figure 8: Visual Example of pixel-by-pixel CAPTCHA comparison.

In this image, the green box represents correct pairs, the red box represents incorrect pairs, and the white box is not included in the heuristic.

3.5.3 Classification

The classification is simply:

For each width i → chose the highest percentage match from all data set characters.
For each character → chose the highest percentage match from all widths.

3.5.4 Classification: Series vs. Parallel

The character width offset from the average character width compounds. In other words, if the CAPTCHA begins with a skinny letter, such as "y", then every letter after "y" will be slightly offset left, until enough wide characters offset the error.

Therefore, instead of centering each character image at .5*avg char width, 1.5*avg char width, etc..., we simply classify each character in series, using the previous best match width as the left border.

3.6 Results

This method has an individual character success rate of 60%, with an overall CAPTCHA success rate of 5.56%. This is ample success for the CAPTCHA to be considered "broken". The density pixelation function actually solved the two main CAPTCHA distortions: hollow letters and letter rips. The brute force method also worked on an incredibly small data set, with an average of 5.21 data images per character. The primary weakness of the brute force method is a high error rate on similar characters. Without more advanced classification techniques, characters such as "S", "5" and "6" will always have high error rates.

4 CNN-based CAPTCHA classification method

Results presented in section 3 show clear weaknesses on the CAPTCHA system used by Microsoft. First and foremost, the character shape variance is not enough to avoid learning templates of the characters and predicting the CAPTCHA content by pixel-level comparison. Furthermore, the dimensionality of the problem is strongly reduced by only using a small subset of characters of the full alphabet. Finally, the constant color makes easy the comparison among different examples.

However, further exploration can be done in order to improve classification accuracy. The technique used in section 3 requires of a precise segmentation of characters for the training. Furthermore, only pixel-level comparison is used, without taking into account typical image transformation such as rotations or deformations. In this section we follow a different approach to solve the same problem as section 3: we use a Convolutional Neural Network (CNN) to classify character by character the CAPTCHA sequence. We are able to achieve better results while using a simpler segmentation technique: the power of the Convolutional Neural Network is able to overcome the problems derived from the segmentation.

4.1 Convolutional Neural Networks applied to text recognition

Convolutional Neural Networks have driven research in computer vision for the last three years. Since the publication of [3], in 2012, the state-of-the-art of the field has been constantly changing due to the advances in the use of these techniques. However, during the last decades, it had been already shown that Neural Networks were a good solution for text classification challenges. [20] showed, already in 1990, that Neural Networks were able to correctly classify text. Furthermore, with the rapid growth of the deep learning world, text recognition systems have even improved performance, dropping error rates for handwritten digit data sets for significant scales [11].

Although the details of the deep learning techniques fall outside the scope of this project, we believe they are useful and deserved a quick review of some of its main principles to better understand our work. In this section we will present the main layers

used in the typical Convolutional Neural Networks, and how there are combined to build classifications systems.

Convolutional Neural Networks are typically composition of multiple layers chained one after the other. The main novelty from classical Neural Networks is the number of layers used: unlike traditional shallow Neural Networks, actual Convolutional Neural Network use a significant number of layers. Although many different layers are used, we will focus on detailing the two main layers typically used.

- **Convolutional layer**: The convolutional layer convolves the input image with a set of filters, which produces a multidimensional output. At the end of the layer, a non-linearity is applied. Traditionally $f(x) = (1+e^{-x})^{-1}$ was used, but [3] proposed $f(x) = \max(0, x)$ which presents some advantages when training the model. It is important to note that convolutional layers keep some spacial relation from input to output.

- **Fully connected layer**: The fully connected layer linearly combines the previous layer and applies a non-linearity to its output. The non-linearity used is the same as the convolutional layer. Fully connected layers are used for dimensionality reduction, to map spacial information to categorical information.

Finally, we are going to detail the structure used by one of the historically used networks for text recognition, LeNet [21]. LeNet was initially proposed by Yann LeCunn on 1989 and has been evolving during the years until the currently used version, Lenet-5.

In figure 9 we present the structure of the LeNet-5 network. It consists of three convolutional layers, followed by two fully connected layers at the end. This network has high performance on the digit recognition task.

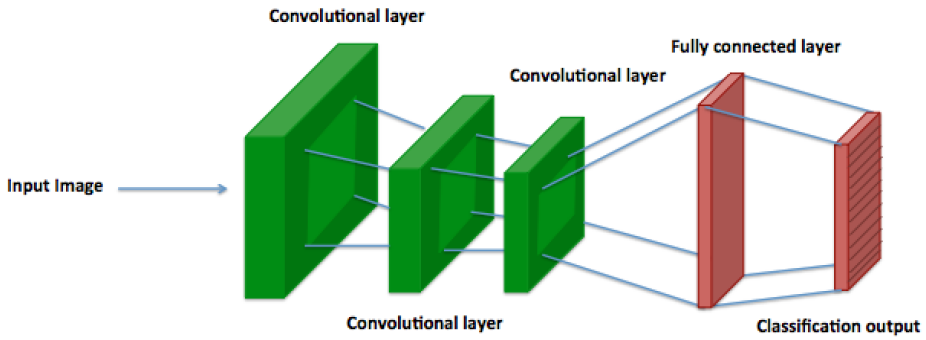


Figure 9: LeNet structure

For our project, we have used a similar structure to LeNet-5. We added one extra layer to create a deeper structure, but the main layout of the network remains the same.

Finally, it is important to note that multiple proposals are already available to do CAPTCHA detection using deep learning. [11] presents a digit classification system for street-view images which can also be applied to CAPTCHA recognition and reports 99.8% success on reCAPTCHA [2]. [11] proposes a more complex system, by using a graphical model to model the probability distribution of the sequence and a convolutional neural network to estimate this probability.

4.2 CAPTCHA data set

One of the key factors to explain the growth of deep learning is the availability for large amounts of data. During the last five years, large data sets [4, 5, 22] have been released providing to the deep learning systems with enough data to train. However, in our context, there was no previous data available for this particular CAPTCHA scheme,

therefore the first step was to build our own large data set. Unlike in section 3, to successfully train a convolutional neural network we need a significant amount of data.

This is why we decided to build a separate data set of characters, using automatic segmentation instead of the human annotation technique used in previous sections. Due to the technical requirement coming from the deep network for a large amount of data, we decided to avoid human segmentation, which would slow down the process and limit our ability to collect data. Furthermore, we show that the inaccuracies produced by the automatic segmentation are not problematic for the Convolutional Neural Network, which is able to successfully deal with them. Finally, we present an evaluation strategy which helps to overcome the potential problems of automatic segmentation.

In order to build the data set, similarly as in section 3, we mined CAPTCHAS from the original Microsoft website. The full data set consist of 600 CAPTCHAs, meaning we collected a total amount of 3600 characters.

Some pre-processing was done to the data before the segmentation. Similarly as in subsection 3.4, we rotated the CAPTCHA and cropped it to avoid white space in the sides. This step is crucial for the automatic segmentation: we assume that the characters are approximately equally distributed along the image. Unlike in section 3, no color pre-processing was done, we trained the Convolutional Neural Network to deal with the variation in the data set.

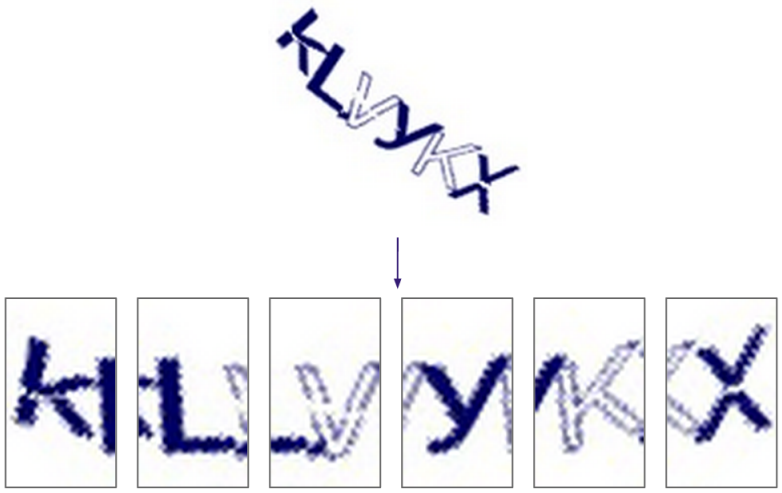


Figure 10: Segmentation example

After the pre-processing was done, we automatically segmented the resulting image and labelled the characters. The segmentation was done assuming equal distance among the characters, so the image was divided in six equally distributed segments, with a slight overlap between them. The main focus of this segmentation technique was to avoid losing part of the target character when doing the segmentation. Nevertheless, by increasing a little bit the wide of each segment we also increase the amount of overlap with other characters. However, we will show that the CNN is able to overcome this challenge and learn to distinguish between the characters and lateral noise.

An end-to-end example of the automatic segmentation process is shown in figure 10. As can be seen, the resulting characters have some pixels corresponding to the adjacent characters in the crop. As it has been said before, our network learned to deal with this style of noisy data and focus on classification of the main character in the image while avoiding the noise present at input.

Finally, we find interesting to present some statistics of our data set. Even though the potential number of categories for the CAPTCHA is 62 (26 uppercase letter, 26 lowercase letter and 10 digits), the real number of categories is only 23. Furthermore, among these 23 categories, not all of them appear with the same regularity. The difference in frequency is significant for some of the characters, which also provides extra information to the system when predicting a new character.

4.3 Method

In this section we present the main method used to classify the data set. We divided the data set in a training set (90% of the data) and a testing set (10% of the data). Furthermore, in order to generate more training data, we did not train our network using the full image but we generated large crops from the image. The random crops were large enough to keep a significant part of the character, but by generating multiple training examples per each image we also trained the network to get some invariance to scale and translation. Finally, the random crops were resized to the input size of the network: 40×40 pixels.

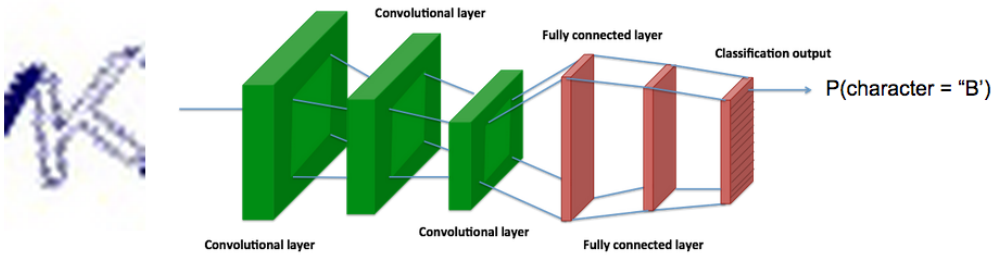


Figure 11: CNN structure

The structure of our network is presented in figure 11. It consists of three convolutional layers followed by three fully connected layers. The training was done using Caffe [23], a popular framework used to apply deep learning techniques to computer vision problems. We trained using GPU technology which significantly reduced the training time. The prediction was done by selecting the maximum value of the output vector. The output layer corresponds to the probability of the input image to be all the different characters.

Furthermore, for testing purposes, we used a cropping strategy similar as the one used for training. For each testing instance, we generated multiple crops of the input image, from which we picked the maximum of the output vector. Additionally, we considered the full input image resized to the crop size as an additional crop. Finally, we selected the maximum value among all the crops, and we set this as our prediction.

This process can be visualized in figure 12. It is easy to see that the first crop contains a significant amount of noisy pixels from the side character. However, for the second and the third crops, the amount of noisy pixels is reduced significantly. It is likely that our network assigns more probability to the last crops, given that the character appears to have less noise around it. With the final max operation, we selected the most clear prediction and use it as out final prediction for the input image. It is important to note that timing is not a problem in this scenario: given the small size of the images (40×40 pixels) and the size of the network, the evaluations through the network did not create any timing issues.

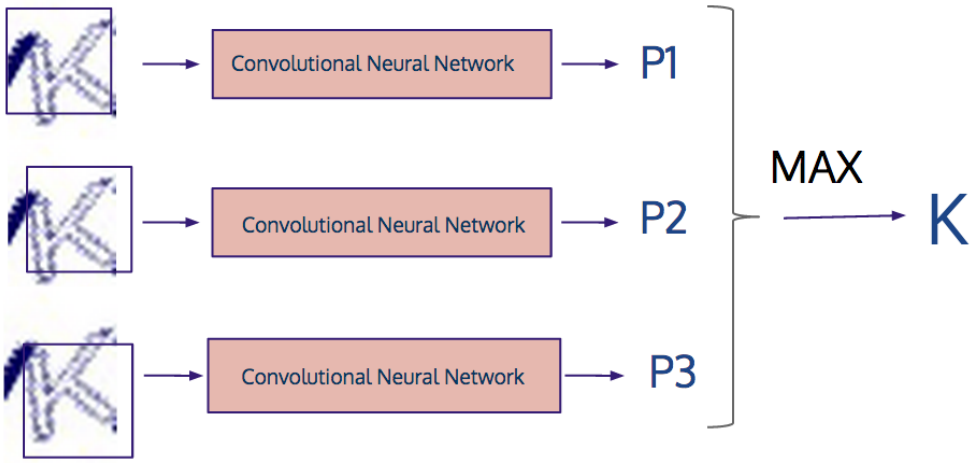


Figure 12: Prediction cropping strategy

To sum up, we used a variation of LeNet-5 trained on our CAPTCHA data set in order to automatically classify CAPTCHA sequences. To speed-up the annotation process, we used an automatic segmentation strategy where the input CAPTCHA was automatically segmented, which has a drawback of generating noisy data. However, we were able to remove the noise by using a cropping strategy in both the training phase and the testing phase.

4.4 Results

In section 3, we showed the ability of the template-matching method to classify correctly 5.56% of the given CAPTCHAs. By using a Convolutional Neural Network trained on our automatically-segmented CAPTCHA data set, we are able to increase our accuracy significantly. In table 1, we present the results for our new method.

Character accuracy	CAPTCHA success	Estimated tries to succeed
91.07%	57.05%	1.75

Table 1: Results of our Convolutional Neural Network

The CNN-based method raises the performance by an order of magnitude compared to the previous template-matching method. Thus, the number of necessary tries to get a complete correct CAPTCHA is reduced one order of magnitude: with only two tries the user is estimated to predict one valid CAPTCHA. Thus, we show a clear weakness in the Microsoft CAPTCHA, which can be completely bridged by using our method.

Even though most of the examples are correctly classified, some categories have significant confusion among them. Some interesting examples are confusions between '5' and 'S' or between '8' and '6'.

5 Improving Microsoft CAPTCHA

Thus far, we have shown two ways to attack the CAPTCHA system used by Microsoft. One utilized template-matching and pixel comparisons to solve the CAPTCHA, while the other used a Convolutional Neural Network to learn the shape of the characters and predict the solution based off of its knowledge.

Many people have investigated ways to improve CAPTCHAs [6, 7] to prevent systems such as those presented from solving them. There are many ways to increase the efficiency of CAPTCHAs. In this section we consider three of them: different colors, image recognition, and motion.

5.1 Color-based security

Different colors, such as figure 13, eliminates one of the main assumptions that we were able to make for the Microsoft CAPTCHA system by making the adversary analyze which pixel is actually important. It is commonly used by many websites today, with each website tending to create their own with their own type of complications and difficulties.

Figure 13: Different-colored CAPTCHAs

This is inherently more difficult than the Microsoft system. By combining many different backgrounds and fonts, the adversary has a much harder task of finding the letters, improving the difficulty for an algorithm, while maintaining the ease of use for the user.

5.2 Image recognition security

Images have been also used as a substitute for text in some new proposals. [7] propose a CAPTCHA scheme based on distinguishing cats and dogs in images. Even though image recognition is harder than text recognition, [16] showed how the problem was solvable by applying automatic classifiers. In the same spirit, [24] asked users to associate images with a list of words. Furthermore, [6] presented a CAPTCHA scheme based on detecting upright images. Finally, [25] used shape discovery to challenge the users.

Although the schemes presented before increase the security of the system significantly, recent computer vision techniques make it hard to rely on well-known computer vision problems such as image classification or shape discovery. In this section, we propose a new CAPTCHA scheme based on a new problem in the computer vision community: the visual question-answering [8].

Figure 14: Visual question-answering example. Figure from [8]

The problem set up is presented in figure 14. Given an image, a simple question about the image is also presented. In the computer vision community, the challenge is to train computers to automatically answer the question.

Our proposal is to show to the user three different images with one question associated. The human would need to answer the three questions correctly to pass the test. Furthermore, the question will also be encoded in a CAPTCHA-like style such as re-CAPTCHA to create more difficulties for the automated systems to correctly parse the question and generate a valid answer.

[26] presents the most accurate model for visual question-answering, but the problem set up is really restricted: all the answers in the data set have only one word and the questions are computed-generated. Thus, this model would fail applied to a more general framework with multiple-word answers and human-generated questions.

Furthermore, [8] presents a method more related to our actual task. It is important to note that in this scenario the question is always correctly parsed, while in our scenario using CAPTCHA style to present the question, a significant amount of questions would be incorrectly parsed by computers.

The models presented in [8] are evaluated in multiple situations, being the open-answer question the comparable situation to our scenario. In all the other experiments presented in [8], the system has some additional information about the answer, such as if it is binary or multiple choice. The models score with high accuracy to some of the open question types (such as the ones starting with "what sport" or "is there"). However, for some other question types the model performance is marginally low, scoring about 3% in some of them.

Our proposal is to build a data set of these low-scoring images and questions, and evaluate humans by asking these questions about the image. If the VQA model has a mean accuracy around 10% in the subset of questions we choose as our data set, the success for the three answers would be 0.1%. Furthermore, if the question text is presented in a CAPTCHA-like style, the parsing error will also impact the model performance, securing the full system even more.

Finally, an interesting issue is how to generate the data to build the CAPTCHA scheme. [8, 26] present a question-answer data set, but using questions from a publicly available visual question-answer data set will completely break the system: attackers can just learn all the combinations of image-question-answer. Furthermore, we believe more complex questions are needed to improve the security of the system and reduce the automated answering success even more.

Our proposal is to use a mixture of sources of information. The main two sources of information are:

- **Image datasets:** The computer vision community has published many image data sets with a large amount of labeled images [3–5]. Given that image categories (such as "food","cat" or "sport") are provided, we can generate generic questions for all the images in a specific category (such as "Name one food present in the image" for all the food images).

- **Image search engines:** By querying image search engines with specific keywords such as "pizza" or "soccer", we can generate a large data set of weakly labeled images. Using the same strategy as with the categorized images, we can generate generic questions per category. The label can be verified in crowdsourcing platforms such as Amazon Mechanical Turk.

Once the images and some generic questions are generated, we want to generate more questions and to answer the already-generated questions. Our proposal is to use a crowd sourcing service such as Amazon Mechanical Turk to both answer the questions and generate new ones.

Finally, to scale up the scheme, an alternative proposal to get the valid answer for a question is to challenge each human with four images/questions. In three of them, the human is evaluated, but the answer for the fourth question is unknown by the system. Furthermore, humans should not know which is the unlabeled question. Using this

scheme, we could automatically get the answer of more questions which could be used to evaluate other humans. This strategy is similar as the one used by [2].

To sum up, we propose a CAPTCHA based on a recent problem in the computer vision community: the visual question-answering. Three images and three questions related to the images are presented to the subject who needs to answer the questions correctly to pass the test. The questions are basic given the image, therefore it should not be hard for humans to easily pass the test. To add more complexity to the system, we propose to present the question in a CAPTCHA-like style, which will difficult the automatic parsing and, thus, increase the security of the full system. Finally, it is important to note that the number of tries per user should be limited to make the system fully secure.

5.3 Motion-based CAPTCHA

Another proposal to improve the security of CAPTCHAs is to use motion in the form of GIFs. GIFs are Graphic Interchange Format, a file format that displays a sequence of images quickly, appearing to form a video. This allows for an additional layer of security because a script will either download the file or take a screen shot; a screen shot will only get a single image of the GIF, while the whole file will require more analysis.

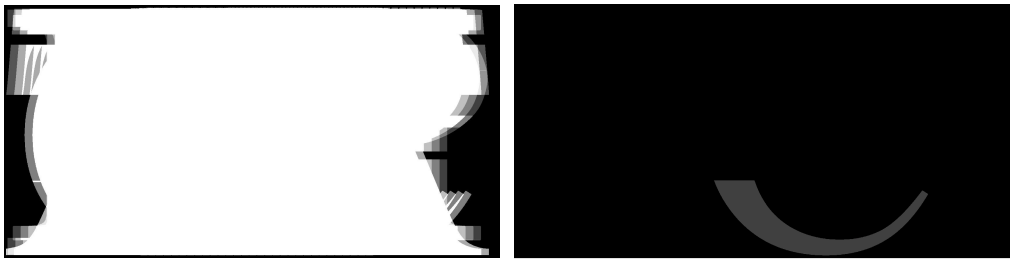
(a) Temporal mean.

(b) Single screenshot.

Figure 15: Static information for the simple motion-CAPTCHA proposal.

The proposed GIF-based CAPTCHA is deployed in this website, due to the impossibility to deploy it in the text report. Figure 15b is one frame of the GIF, while figure 15a is what they make when combined. It is obvious that each line offers a negligible amount of information as to the actual letter that is there. However, figure 15a illustrates one of the main problems with the GIF creation: temporal mean. Since the adversary will have access to the file itself, they could create a system that will get the temporal mean of the GIF. So this basic example shows the possibilities, but not the ideal solution.

To improve it, we need to create GIFs whose temporal mean has virtually no information. We found that by displaying moving letters, the temporal mean become uninformative because it adds up the signal for all the moving letters. Furthermore, using the same strategy as in figure 15b, we can also have uninformative single frames by splitting each letter in multiples frames. The resulting GIF is deployed in this website. In figure 16a we show the global temporal mean and in figure 16b we show a single frame of the CAPTCHA scheme. It is clear that neither of those provide enough information to attack the system.



(a) Temporal mean.

(b) Single screenshot.

Figure 16: Static information for the motion-based CAPTCHA proposal.

Finally, multiple attacks to motion CAPTCHA have been proposed [27]. A possible attack to our CAPTCHA proposal is the sliding window temporal mean. This is,

finding information about the characters by doing a shorter temporal mean over subsequences of the GIF. To secure our scheme against this attack, we present three main countermeasures:

1. In our scheme, each character is divided in n_{char} parts. By overlapping all the parts together, we end up with the target character. To generate a clear character, the attacker needs to do a mean of length n_{char} frames starting in T_0 where T_0 is the moment when the character starts to be generated. Our first countermeasure is to randomize the value of n_{char} and T_0 character by character. The attacker needs to try all the possible lengths of temporal means and iterate over all the possible starting points T_0, since this two values are not deterministic. The range of possible n_{char} is rather reduced, but T_0 can have a greater variation. Thus, since the total amount of necessary tries is the multiplication of these two number, we can deeply increase the number of temporal means needed and, thus, secure our system.

2. Once the attacker computed all the partial temporal means, he/she needs to automatically detect which partial temporal mean corresponds to a valid character and which are invalid partial temporal means. Our second countermeasure is to generate subsequences where the invalid temporal means can be detected as valid characters. For instance, in figure 17 we present a temporal mean generated by averaging some frames of the character "6". However, an attacker will detect a character 0 or O in the sequence, which will invalidate his attack.

Figure 17: Temporal mean for part of the character "6".

   By decomposing the characters in frames with this property, we increase the number of wrongly detected characters by the attacker and, thus, reduce its success probability.

3. Finally, in our toy examples, the characters are clean and do not have any noise, for simplicity reasons. By adding noise and complexity to the character shape such as done in classical CAPTCHA schemes, we can increase the detection difficulty.

The implementation of these three countermeasures will significantly reduce the probability of success. By limiting the number of tries (enough to deal with user errors), the system can be considered secure against this attack.

6 Conclusions

To sum up, this project has a two-fold purpose: to show the weaknesses of the actual Microsoft CAPTCHA system and to propose two methods to improve its security.

First, we proposed two methods to automatically classify Microsoft CAPTCHA samples. Each began with three assumptions based on the Microsoft CAPTCHA: the CAPTCHA only had a single color, only used 23 different characters, and a single font. The first method is based in a fine-grain segmentation combined with template matching. This method has a success of 5.56%, enough to compromise the security of the Microsoft system. Furthermore, we proposed a more complex method based on Convolutional Neural Network, which, using state-of-the-art recognition techniques, is able to

achieve a success rate of 57.05%, more concretely demonstrating the lack of security of the CAPTCHA scheme.

Secondly, we proposed two alternative CAPTCHA schemes: the visual question-answering challenges and motion-based CAPTCHAs. Our first proposal is based in visual question-answering, a fairly new research topic in the computer vision community. In this scheme, the user is presented some images with questions related to the images. The questions are easy enough for the human but hard for a computer to be answered. Actual models do not have enough accuracy on the answer prediction to make the system insecure. Our second proposal is based on motion CAPTCHAs. By adding motion to the characters in the CAPTCHA, we increase the complexity for attackers to capture informative frames. We discussed potential attacks such as partial temporal means and propose countermeasures to avoid them.

References

[1] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford, "Captcha: Using hard ai problems for security," in *Advances in Cryptology—EUROCRYPT 2003*, Springer, 2003, pp. 294–311.

[2] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, "Recaptcha: Human-based character recognition via web security measures," *Science*, vol. 321, no. 5895, pp. 1465–1468, 2008.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances In Neural Information Processing Systems*, pp. 1–9, 2012. arXiv: `1102.0183`.

[4] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014*, Springer, 2014, pp. 740–755.

[5] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "Labelme: A database and web-based tool for image annotation," *International journal of computer vision*, vol. 77, no. 1-3, pp. 157–173, 2008.

[6] R. Gossweiler, M. Kamvar, and S. Baluja, "What's up captcha?: A captcha based on image orientation," in *Proceedings of the 18th international conference on World wide web*, ACM, 2009, pp. 841–850.

[7] P. Golle, "Machine learning attacks against the asirra captcha," in *Proceedings of the 15th ACM conference on Computer and communications security*, ACM, 2008, pp. 535–542.

[8] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, "Vqa: Visual question answering," *ArXiv preprint arXiv:1505.00468*, 2015.

[9] G. Mori and J. Malik, "Recognizing objects in adversarial clutter: Breaking a visual captcha," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, IEEE, vol. 1, 2003, pp. I–134.

[10] J. Yan and A. S. El Ahmad, "A low-cost attack on a microsoft captcha," in *Proceedings of the 15th ACM conference on Computer and communications security*, ACM, 2008, pp. 543–554.

[11] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," *ArXiv preprint arXiv:1312.6082*, 2013.

[12] S. Li, S Shah, M Khan, S. A. Khayam, A.-R. Sadeghi, and R. Schmitz, "Breaking e-banking captchas," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ACM, 2010, pp. 171–180.

[13] V. D. Nguyen, Y.-W. Chow, and W. Susilo, "Breaking a 3d-based captcha scheme," in *Information Security and Cryptology-ICISC 2011*, Springer, 2012, pp. 391–405.

[14] ——, "Breaking an animated captcha scheme," in *Applied Cryptography and Network Security*, Springer, 2012, pp. 12–29.

[15] S. Belongie, J. Malik, and J. Puzicha, "Shape context: A new descriptor for shape matching and object recognition," in *NIPS*, vol. 2, 2000, p. 3.

[16] J. Elson, J. R. Douceur, J. Howell, and J. Saul, "Asirra: A captcha that exploits interest-aligned manual image categorization.," in *ACM Conference on Computer and Communications Security*, 2007, pp. 366–374.

[17] G. S. Kalra, "Attacking captchas for fun and profit," McAfee, 2012.

[18] S. H. L. V. A. Jennifer Tam Jiri Simsa, "B reaking audio captchas," Carnegie Mellon University, 2009.

[19] K. Lab, *Kaspersky lab discovers podec: The first trojan to trick captcha into thinking its human*, 2015. [Online]. Available: `http://www.kaspersky.com/about/news/virus/2015/Kaspersky-Lab-discovers-Podec-first-Trojan-to-trick-CAPTCHA-into-thinking-its-human`.

[20] B. B. Le Cun, J. S. Denker, D Henderson, R. E. Howard, W Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*, Citeseer, 1990.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, IEEE, 2009, pp. 248–255.

[23] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *ArXiv preprint arXiv:1408.5093*, 2014.

[24] R. Datta, J. Li, and J. Z. Wang, "Imagination: A robust image-based captcha generation system," in *Proceedings of the 13th annual ACM international conference on Multimedia*, ACM, 2005, pp. 331–334.

[25] M. Conti, C. Guarisco, and R. Spolaor, "Captchastar! a novel captcha based on interactive shape discovery," *ArXiv preprint arXiv:1503.00561*, 2015.

[26] M. Ren, R. Kiros, and R. Zemel, "Image question answering: A visual semantic embedding model and a new dataset," *ArXiv preprint arXiv:1505.02074*, 2015.

[27] V. D. Nguyen, Y.-W. Chow, and W. Susilo, "Breaking an animated captcha scheme," in *Applied Cryptography and Network Security*, Springer, 2012, pp. 12–29.