CAPTCHAs Recognition Using Neural Networks

Benjamin Hinton
University of California-Berkeley
306 Stanley Hall, Berkeley, CA, 94720
bhinton@berkeley.edu

Jun Song
University of California-Berkeley
253 Cory Hall, Berkeley, CA, 94720
juns123@berkeley.edu

Michael Tong
University of California-Berkeley
253 Cory Hall, Berkeley, CA, 94720
mrtong96@berkeley.edu

Abstract

CAPTCHA (Completely Automated Public Turing test to Tell Computers and Humans Apart) images have been used for years to help separate actual human users from computer-controlled counterparts online. If software could successfully answer CAPTCHA prompts, it would indicate bots could successfully bypass this security process and that it needs revision. We propose a project that would attempt to use computer vision to successfully answer CAPTCHA prompts. This information would be valuable to any company with an online presence and especially useful to IT security auditors. Such knowledge could potentially find alternative human authentication methods that are more difficult to automate. Our group created a convolutional net inspired by MNIST convolutional nets with the goal of correctly guessing the contents of a CAPTCHA image. After designing segmentation algorithms, creating and training a deep convolutional net, we were able to accurately guess the contents of a CAPTCHA with 74% accuracy on validation set. Our work shows that computer vision can be used to bypass an integral feature of the internet.

1. Introduction

CAPTCHA (Completely Automated Public Turing test to Tell Computers and Humans Apart) images have been used for years to help separate actual human users from computer-controlled counterparts online. CAPTCHA images typically contain a string of letters and/or numbers that are distorted through skew, discoloration, occlusions, or other means [1]. Samples of CAPTCHAs are shown in figure 1 and can be found readily online.

Ensuring CAPTCHAs are only readable by humans is hugely important to data and computer security. CAPTCHAs are often used as gateways on websites to ensure a human is authorizing an action, and is used in online banking, account creation, and other online interactions [1]. Creating bots that that can guess the answer to a CAPTCHA would make these sites less secure because bots could automate actions to create spam accounts, simulate human activity on forums, and . Therefore, it is worthwhile to examine whether machine learning is capable of correctly replicating the answers to CAPTCHA images.

Previous work has attempted to solve CAPTCHA images using various methods. Malik, et al produced a CAPTCHA guessing algorithm that first prunes the image to retrieve a set of likely word locations and then match that set to a matching procedure [1]. They and other groups describe two possible algorithms used: first a method that splits up the characters of a word and attempts to guess each letter, and a second method that attempts to find an entire word immediately [1, 9]. Both methods were shown to be effectively implementable in many settings [2, 3, 4, 5, 6]. Other groups have used neural networks and adversarial networks to set up a system to guess CAPTCHA images [2, 4, 6]. While neural networks are popular, other groups have also used optical character recognition (OCR) without the use of neural networks [8].

Our group hypothesized that it would be possible and efficient to create a neural network that would be able to guess CAPTCHA images correctly by using a previously established method to split a CAPTCHA word into its individual letters and guess each letter individually [9]. By showing CAPTCHAs can be solved routinely with computer vision, our work shows that an essential security element of the internet can be easily circumvented.

2. Our Approach

2.1. Preprocessing

Because we decided to segment the CAPTCHA images into each individual character and solve for each character individually, we determined several preprocessing steps that were important to carry out for either the purpose of saving memory, reducing noise, or aiding in segmentation of the characters.

2.1.1 Adding Constrast

First, we added constrast to our CAPTCHA images by transferring the RGB images to grey scale images. Adding constrast improves the preformance of classification, saves memory for image storing and increases computational speed for further neural network training. Images from before and after the conversion are shown in figures 1 and 2.

Figure 1. Orignal CAPTCHA Image

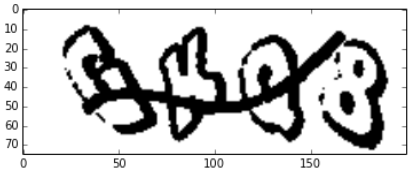


Figure 2. Constrast CAPTCHA Image

2.1.2 Line Removal

The major noise of CAPTCHAs generated by **wsy-captcha** is the horizontal line across all characters. This horizonal line has similar font weight as the characters and it makes all characters connected. Because of the noise, a flood-filling segmentation algorithm will fail to segment correctly and return a single component containing all characters instead. Further, histogram segmentation algorithms will be unable to find minima points as possible segmentation locations. Pattern matching would also have a poor performance because of the random line across each character. To make the CAPTCHA image segmentable, we design a **line locating and tracking algorithm** and successfullly remove the horizonal line.

The **line locating and tracking algorithm** has two phases: locating and tracking. In the locating phase, we locate a vertical cross section of the horizonal line by generating a line segments list containing line segments of each column of the CAPTCHA images. Then we loop through the above list and pick an optimal column that contains only one line segment which width is closest to the font weight. With high probability, the single line segment in the optimal column we find above is a cross section of the horizontal line.

In the tracking phase, we take the cross section from locating phase as input and track the entire horizontal noise line.

Tracking Phase

Input: a constrast CAPTCHA image img, a vertical cross section C_j of the horizontal line that contains the upper and lower pixels of the cross section and assume the cross section is in j-th column

Calculate the middle index of the cross section in j-th column:

$m_j = avg(C_j[0], C_j[1])$

$m = m_j$

For i-th column from j-th column to leftmost column: start from $img[i, m]$, move $0.5 *$ font weight + threshhold in both up and down directions and get cross section of the horizontal line in i-th column C_i.

$m_i = avg(C_i[0], C_i[1])$

$m = m_i$

Remove C_i from the image

For i-th column from j-th column to rightmost column:

Do the same

end

A sample image after removing line using line locating and tracking algorithm:

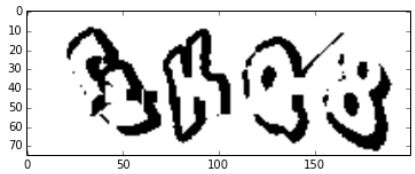


Figure 3. CAPTCHA Image after Line Removal

Our image is now ready for character segmentation.

2.1.3 Segmentation

The goal of segmentation is to identify the location of the characters and extract them the image. The most commonly used algorithms include Pixel Counting Approach and Histogram Approach [9]. In this section, we discuss these methodologies to segment characters and how we apply and combine them to get our final segmentation algorithm.

Pixel Counting Approach: In the original Pixel Counting Approach, the line separation procedure consists of scanning the image row by row. If the image is binarized, white pixels represent blank space between text lines whereas black pixels represent the actual text. In the first method, the line separation on a binarized image is obtained by setting a threshhold value for the number of white pixel rows between two address lines. Two lines are separated if the number of white pixel rows between them is greater than the threshhold value. However, if the first line consists letters like y, g and the second line consists letters like f, d. Due to the overlapping, the first approach will fail to provide accurate results [9]. The second method is performed under the assumption that the space between two lines is constant. The text image can be segmented at regular intervals if the space between two consecutive lines are treated as a constant. This method successfully addresses the problem of overlapping characters in the first method [9].

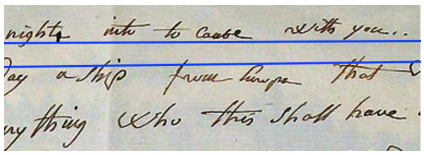


Figure 4. Original Pixel Counting Approach Example

In our application, we modify the original pixel counting approach in two ways. In our experiment, the CAPTCHAs have a single line of $1-4$ characters. Instead of finding horizontal separation for lines by scanning row by row in the original pixel counting approach, we scan the CAPTCHAs column by column to find vertical separations to separate characters. As can be seen in the below figure, the white spaces between each character in our experiment vary hugely and cases exist where a single character has white spaces within itself. As a result, setting threshhold for blank space between character (method 1 above) or assuming the blank space has constant length (method 2 above) work poorly for our CAPTCHAs. However, we find that in our CAPTCHA images, the number of black pixels is within a certain range $400 - 800$ for most characters. So if we find vertical separation lines by counting black pixels column by column and setting the average pixels in a character as threshhold, we can get a rough separation result, which can be further improved by applying histogram segmentation.

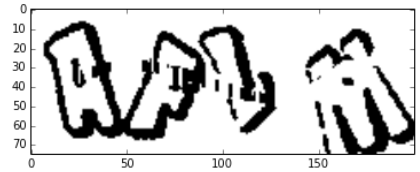


Figure 5. CAPTCHA showing the varying spaces between characters

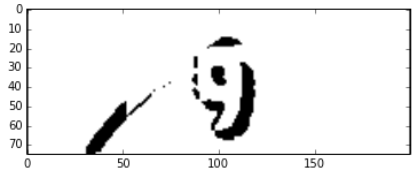


Figure 6. CAPTCHA showing the white spaces within a single character

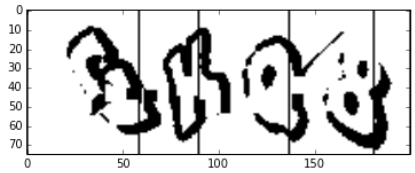


Figure 7. Result of segmentation lines after applying pixel counting segmentation

Histogram Approcach: Histogram approach is a method to automatically identify and segment the text line regions of a handwritten document. The feature extraction or binarization step is applied to the input image. Then, a Y histogram projection is obtained to detect the possible lines and an X histogram is used to segment words and characters.

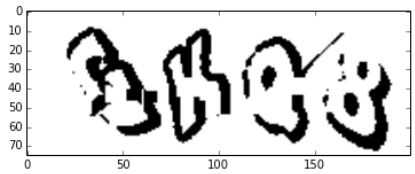


Figure 8. Sample CAPTCHA

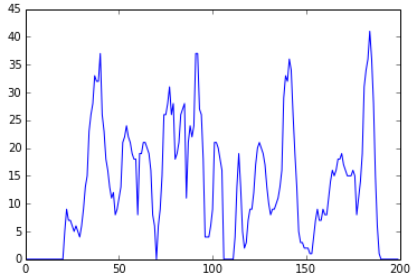
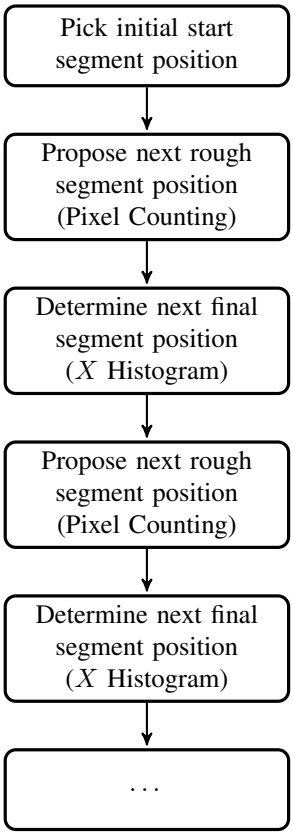


Figure 9. X-histogram of the sample CAPTCHA

In our experiment, after we find the rough segmentation positions using pixel counting approach, we apply X histogram approach to determine the final segmentation positions. Specifically, we propose a rough next segmentation position by counting the total black pixels from the previous segmention position. Then the number of pixels at each column that is in the window centered at the rough next segmentation position is calculated and compared. The minima

of this value are chosen as the final next segmentation position.

Pixel counting approach and histogram approach were applied in an alternating way as in the following graph:



Pixel Counting with Histogram Segmentation

Input: a CAPTCHA image after line removal, c the average number of black pixels in each character, d the maxima possible distance between neighbor segmentation positions.

Output: a list of segmentation positions lst

Define the initial segmentation position:

$s_0 = 0$

Define the latest segmentation position:

$s = s_0$

$lst = [s_0]$

While black pixel count from s to rightmost column $\geq 2 \times c$

total pixel numbers: $count$ = 0

For i-th column from s to $s + d$:

$count = count + num\text{-}pixel(i\text{-th column})$

If $count \geq c$:

$s_{approx} = i$, **break**

Search for an optimal column s_{opt} from $s_{approx} - \epsilon_0$ to $s_{approx} + \epsilon_1$ that has the minumum number of black pixels.

Assign next segmentation position: $s = s_{opt}$

$lst = [lst, s]$

end

We perform the above algorithm from left to right and right to left and get two sets of solutions. Then, we choose the segmentation solution with smaller variance in the array of pixel counts in each segment. Sample outputs of our segmentation algorithm are shown in figures 10, 11, and 12:

Figure 10. Segmented CAPTCHA Image

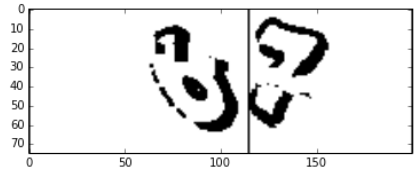


Figure 11. Segmented CAPTCHA Image of 2 Characters

Figure 12. Segmented CAPTCHA Image of 3 Characters

2.1.4 Segmentation Performance

We tested the segmentation algorithm on 100 images with different numbers of characters. The algorithm has a success rate of 89%.

2.2. Classification: Convolutional Neural Network

2.2.1 Overview

Our net takes in the segmented images from the segmentation and crops the image into 75x75 pixel sections then attempts to classify the cropped images.Since most of the crops are not 75x75 pixels, we pad or crop the image to keep it 75x75 pixels and center it.

2.2.2 Architecture

Our architecture is loosely based off of Alexnet [10]. The layers are as follows:

layer name	layer type	nonlinearity
conv1	5x5x8 convolutional layer	RELU
pool1	2x2 max pooling layer	none
conv2	5x5x16 convolutional layer	RELU
pool2	2x2 max pooling layer	none
conv3	3x3x32 convolutional layer	RELU
pool3	2x2 max pooling layer	none
fc1	1024 fully connected layer	RELU
fc2	1024x36 fully connected layer	RELU

2.2.3 Training

For training, we used the package wsy_captcha to generate 10^4 sample CAPTCHA images.

Figure 13. Samples of CAPTCHA images

Then with our generated samples, we segmented and binarized the images. This allowed us to retain most of the information in the CAPTCHA while saving drastically on memory costs and allowing us to increase our training size set.

Figure 14. Example of mapping the CAPTCHA to processed images

3. Datasets and Performance Evaluation

In our validation experiment, we generated 10^4 $4-$character CAPTCHAs using **wsy-captcha**. We separate our samples into a training and test set with ratio $8:2$. Then, we trained and tested on this data set for 10^5 steps and recorded the training and test accuracy with number of steps. The results are shown in the following figure:

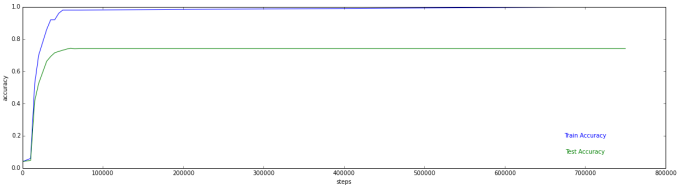


Figure 15. Training and test accuracy vs number of steps

We were able to achieve roughly 74 percent classification accuracy with our given architecture.

4. Conclusion and Future Work

In this report, we design a line locating and tracking algorithm that can denoise CAPTCHA images by removing the noise line. We also introduce a modified histogram segmentation that can work for CAPTCHA images of any length in order to solve for CAPTCHAs through the segmentation method. We then propose a 6 layer convolutional neural network to classify each segment. The system is achieving 74% test accuracy. Overall this project was a successful demonstration of how to use convolutional neural networks to read text in an adversarial setting. However, there are more ways in which the performance of text recognition can be improved.

One of the flaws in our system is how the segmentation of the image affects the later part of the classification. One way to get around this problem would be to write an end-to-end convolutional network that would encapsulate the tasks of segmenting the letters and identifying them. This would be preferred over naive approach of having d^n categories as this would take unreasonable amounts of time to train, require a huge training set, and not be as generalizable when the legnth of the CAPTCHA increases (d is the number of categories of a single character, n is the max possible length of CAPTCHA). Another benefit of this method would be it is that neural networks are much faster than our current segmentation code.

Another limitation of this system is that this devised preprocessing steps, while very important to achieve proper segmentation, would only work in the set of CAPTCHAs produced by this particular python package. For other types of CAPTCHAs, other preprocessing steps would have to be devised for segmentation to work properly or we would have to employ methods used by some of reading the entire

word into the neural network rather than just one character at a time.

Another thing we can further explore is to increase the size of training set. Since we can effectively infinitely generate new CAPTCHAS to train our model, we could drastically improve our model by increasing the set of our training set size. This would in turn allow us to increase the model capacity of our neural net as there would be less overfitting with a larger training set. The only limit to this approach would be the computational resources required to create and store these examples.

In the future, it would be interesting to train this model on a large and robust data set of CAPTCHAs and then experiment to see how much noise (in the forms of lines, occlusions, etc.) the system is able to overcome. We had to do line removal in order to do proper character segmentation, but given the wide variety of types of CAPTCHA images, it is especially pertinent to study how robust the network is to noise and certain specific types of noise (skew, rotation, color inversion)

References

[1] 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, 2003, vol. 1, pp. I134.

[2] A. Timmaraju and V. Khanna, Detecting and Recognizing Text in Natural Images using Convolutional Networks.

[3] J. Wang and P. H. How, Google Street View Character Recognition.

[4] C. Lu and K. Mohan, Recognition of Online Handwritten Mathematical Expressions Using Convolutional Neural Networks.

[5] G. Wang and J. Zhang, Recognizing Characters From Google Street View Images.

[6] V. Sundaresan and J. Lin, Recognizing Handwritten Digits and Characters.

[7] V. Shrivastava, Artificial Neural Network Based Optical Character Recognition, Signal Image Process. Int. J., vol. 3, no. 5, pp. 7380, Oct. 2012.

[8] M. Jnsson and H.-H. Bothe, OCR-Algorithm for Detection of Subtitles in Television and Cinema., in CVHI, 2007.

[9] N. Dave, Segmentation Methods for Hand Written Character Recognition, Int. J. Sig-nal Process. Image Process. Pattern Recognit., vol. 8, no. 4, pp. 155164, Apr. 2015.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, pages 11061114, 2012.