Automatic CAPTCHA Recognition

Lim Mingjie, Kenneth, kl545

I. Introduction

CAPTCHA tests are a challenge-response mechanism used as part of rate-limiting measures on websites and web services to determine whether a response was provided by a human or by a computer. A didactic example is an image containing alphanumeric characters (henceforth referred to as "text-based" CAPTCHAs) to be identified by a human, but in recent years variants utilizing audio or graphical components have also gained prominence [1], [2]. The premise of any CAPTCHA test is to require a result which is difficult for a computer to reach by adopting an algorithmic approach, but easy enough for humans that it does not require significant effort, thereby performing a "reverse" Turing test.

In this paper, we explore the effectiveness of different artificial intelligence and machine learning techniques for identifying character candidates extracted from CAPTCHA images. We start with a naive k-nearest neighbor classifier to establish a performance baseline, before moving on to characterize the performance of support vector machines and neural networks, both of which represent the current state-of-the-art for object recognition. In each case, we describe the design and structure of the technique being considered, as well as the intuition (or otherwise) for the parameters or configurations that produce optimal results.

A. Related Work

A large body of literature discusses the efficacy and security concerns of CAPTCHA tests [3], [4], the development of "segmentation-resistant" CAPTCHA images [5], [6], and the circumvention thereof [7], [8].

Bursztein et. al describe an end-to-end automated tool called "Decaptcha" that achieves between 40–70% accuracy rates on a wide variety of CAPTCHA implementations as of 2011 [9], with the exception being Google's reCaptcha because it utilizes features which involve current challenges in computer vision. However, in 2013, Goodfellow et. al from Google used deep learning and convolutional neural networks adapted from the Google Maps project to successfully circumvent reCaptcha tests 99.7% of the time [10]. While such algorithms are not easily accessible in the public domain, recognizing text-based CAPTCHAs should be considered a solved problem.

Since our attempt will no doubt be less involved, we are more interested in the performance achieved using methods similar to that which we intend to apply. Hindle et. al report between 60–90% accuracy on five different types of CAPTCHAs using a nearest neighbor approach [11]. Chellapilla et. al reported 95–99% accuracy on character recognition in CAPTCHAs using neural networks and support vector machines [12].

II. CAPTCHA Landscape

There have been multiple, non-proprietary approaches to the generation of CAPTCHA images. Regardless of the approach, the intention is to add or use features which either directly interfere with attempts to extract characters from an image, or else modify the characters to make recognition difficult using naive methods.

A. Common Libraries

Most implementations of CAPTCHA tests are drawn either from the GIMPY library [13], or from Google's reCAPTCHA project [14]. The former is a part of the original CAPTCHA generation framework proposed by Ahn, Blum and Langford at Carnegie Mellon University [15]. It is capable of generating tests at varying levels of difficulty, and is most commonly used on forums or older websites (which represent the majority of websites on the Internet). The latter presents a user with two words, each with a line drawn through all characters in the word. These words are drawn from a corpus of unrecognized text encountered when digitizing books or real-world data, further increasing the difficulty of the test.

Our work in this paper focuses on CAPTCHAs generated by the GIMPY library. We consider two levels of difficulty: (1) EZ-GIMPY, which is which represents a nominal attempt at the identification of obfuscated or deteriorated characters for which a good reference exists; and (2) GIMPY, which represents a more challenging attempt to identify malformed characters for which a reference may not exist. We ignore Google's reCAPTCHA because it utilizes features which require extensive treatment with computer vision techniques.

Since we are primarily interested in the application of artificial intelligence to *recognize* extracted characters, we do not unduly concern ourselves with the nuances of image segmentation. It is trivial to extend the work we subsequently describe to effectively handle Google's reCAPTCHA at the appropriate stage of the processing pipeline.

B. Common Techniques

The following techniques are commonly used (in any combination) by CAPTCHA generators. A good understanding of these techniques is vital for the design of algorithms that are capable of circumventing CAPTCHA tests with a high success rate.

1) Multiple Salient Features: Many algorithms often assume that the characters to be read are the most salient feature in the CAPTCHA image. This assumption is fair because a minimum amount of saliency must be present in order for humans to distinguish the characters from the background or other deterrent features. CAPTCHA implementations thus aim to increase the difficulty of the test by adding features which compete with the characters in the image for global saliency, but still remain distinguishable.

For example, some implementations apply a grid overlay in a different color or add prominent geometric shapes (of comparable scale to the characters) to the image. In the case of the former, algorithms which attempt to extract characters by thresholding the image may extract the grid instead of the characters, rendering subsequent operations invalid. In the latter, algorithms which attempt to classify said shapes as characters will produce false positives that result in an incorrect response.

2) Text Warping: Prior to recognition, extracted characters are often encoded using a feature description metric which is both rotationally invariant and scale invariant at the character level. While this may not be an explicit attempt at normalization, it decreases the dimensionality of the data and increases recognition accuracy. However, the effectiveness of such encodings can be decreased by applying affine transformations to arbitrary regions of the image. This warps the appearance of a character in a non-uniform manner and is difficult to reverse.

3) Anti-Segmentation Features: The process of extracting characters involves the reliable identification of pixel regions in the image which constitute a single character. Methods which generalize well (to a broad category of inputs) tend to rely on flood-fill algorithms to identify disconnected regions in the image, wherein each disconnected region represents a single character. If a line of comparable thickness to the average width of the characters in the image and identical color is drawn such that it passes through every character, flood-fill algorithms will fail to correctly identify regions. This results in a single image containing multiple connected characters being passed to the recognition algorithm, which will in turn fail to produce a correct output.

III. Methodology

CAPTCHAs are recognized by first segmenting the original image to extract all character candidates. Each character candidate is a single image that is normalized and provided to the classifier, which returns a prediction label. Prediction labels are interpreted and concatenated to form the response to the CAPTCHA.

A. Character Feature Description

A naive approach to describing a character would be to perform a one-to-one comparison of pixel intensities for images of identical dimensions. This approach fails to generalize well when comparing objects with known features, such as characters, because small offsets or differences in scale cause a large difference in measurement.

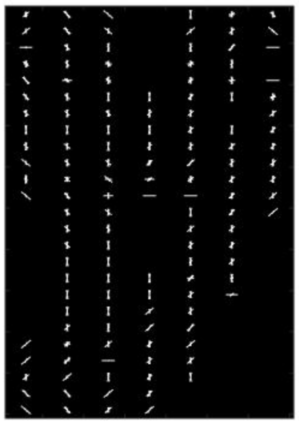
For each image provided to the classifier, we assume that:

1) The image has dimensions identical to that of every other image that the classifier has seen;
2) The image contains only one character. That is to say, all pixels in the image contribute towards the representation of a character; and
3) The image is a binary image. That is to say, every pixel in the image has an intensity of either 1 or 0.

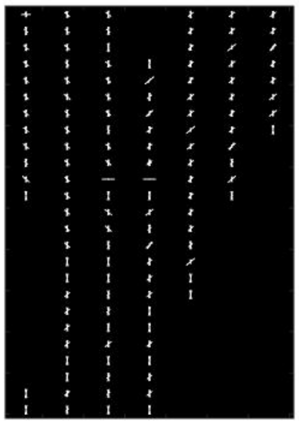
Next, we calculate the Histogram of Oriented Gradients (HOG) for the image using a cell size HOG_c which is 5% of the overall size of the image, and the L2-norm for block normalization [16]. HOG descriptors count the occurrences of gradient orientation in localized portions of an image, and are invariant to scale, rotation, and lighting. Since most characters have distinctive shapes, but are likely to be distorted even after extraction from the CAPTCHA image, the use of HOG descriptors is a good metric for comparing similar characters.

Specifically, the HOG descriptor provided as input (on behalf of the image) is a vector of length n_{HOG}, where:

$$n_{\text{HOG}} = (n_{\text{blocks}})(\text{HOG}_b)(n_{\text{bins}}) \quad (1)$$



(a) HOG descriptors for the character candidate 'Y' extracted from a CAPTCHA image.



(b) HOG descriptors for a training character for 'Y' from the Chars74 data set.

Fig. 1: Comparison of HOG descriptors.

where n_{bins} is the number of bins used for grouping gradient orientations, HOG_b is the size of each block, and n_{blocks} is given by:

$$n_{\text{blocks}} = \left\lfloor \frac{\dfrac{I_s}{\text{HOG}_c - \text{HOG}_b}}{\text{HOG}_b - \left\lceil \dfrac{\text{HOG}_b}{2} \right\rceil} + 1 \right\rfloor \tag{2}$$

where I_s is the dimensions of the image. For all subsequent work, unless otherwise mentioned, we have arbitrarily specified I_s to be $[50, 35]$, i.e. a height of 50 pixels and width of 35 pixels, $\text{HOG}_b = [2, 2]$, and $n_{\text{bins}} = 9$.

As an illustration, Figure 1a and Figure 1b display the HOG descriptors for an extracted character and reference character respectively. Note how both images have similar gradient representations.

B. *Training Data*

Microsoft Research's Chars74 data set [17] was used to train the classifiers described in subsequent sections. The data set provides two categories of images:

1) Photographs of segmented characters in various natural scenes; and
2) Characters from computer fonts in normal, bold, and italic weights.

The latter category is of particular interest because CAPTCHA images are commonly generated using computer fonts. Each entry in the data set was converted to a binary representation and cropped to an image size of I_s as described above.

C. *Labeling Nomenclature*

The set of characters C which the classifier must be able to identify is:

- The numbers 0 to 9;
- The lowercase characters 'a' to 'z'; and
- the uppercase characters 'A' to 'Z'.

Thus, it is possible to define an indexing function I for some training label l_t such that:

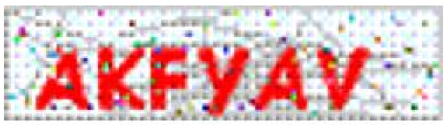
$$I(l_t) = \begin{cases} l_t + 1, & 0 \le l_t \le 9 \\ \text{UENC}(l_t) - 54, & \text{A} \le l_t \le \text{Z} \\ \text{UENC}(l_t) - 35, & \text{a} \le l_t \le \text{z} \end{cases} \tag{3}$$

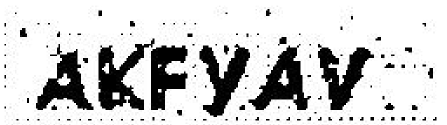
where $\text{UENC}(x)$ is the Unicode number for the character x. In effect, the foregoing function is a standardization step that maps the range of inputs supported by the classifier such that $I(l_t) \in [1, 62]$ for any $l_t \in C$. For example, if $l_t = \text{b}$, $I(l_t) = 12$, and if $l_t = \text{B}$, $I(l_t) = 38$.

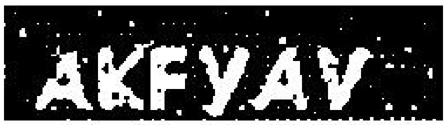
D. *Input Data*

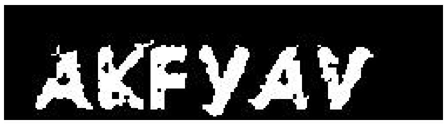
An image (Figure 2a) containing a character candidate is pre-processed in the following manner:

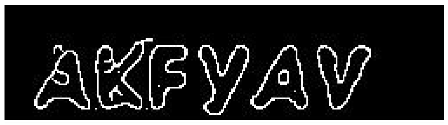
1) The image is read and thresholded to a binary representation (Figure 2b). To counteract intentional variations in illumination, the thresholding is performed using a sliding window that is 10% of the overall size of the image in both dimensions. Mean values above a threshold parameter (calculated using Otsu's method [18]) are set as 1, and 0 otherwise.
2) Spurs and artifacts are removed from the binary image by applying a convolution filter over a 3×3

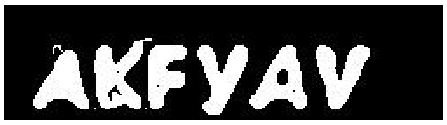

(a) Original Image

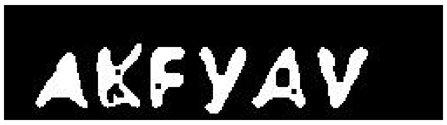

(b) Thresholded Image


(c) Spur Removal


(d) Noise Removal


(e) Edge Detection


(f) Flood-fill Bounded Regions


(g) Morphological Thinning

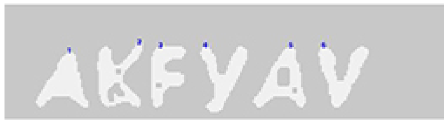

(h) Connected Components

Fig. 2: Segmentation Pipeline.

neighborhood to remove pixels which do not have 8-connectivity with their neighbors (Figure 2c).

3) Noise is removed from the image by performing a morphological closing operation (Figure 2d).
4) Edges are extracted from the image using a Canny edge-detection algorithm, smoothed, and joined to create a closed contour (Figure 2e).
5) Regions bounded by two close contours found in the previous step are flood-filled. This produces character candidates which have better shape definition and are less noisy compared to a straightforward morphological dilation operation (Figure 2f).
6) Apply a morphological thinning operation to enhance the shape of each character (Figure 2g).

At the end of the following steps, we expect to obtain a single image containing multiple connected components, wherein each connected component represents a character candidate to be identified (Figure 2h). Each connected component is extracted by tracing boundaries, and cropped to an image size of I_s as described above.

E. *k*-Nearest Neighbor Classifier

Characters can be classified using their nearest neighbors by measuring the Euclidean distance between a pair of HOG descriptors. The approach itself is straightforward, but tends to yield the poorest results because the classification mechanism is sensitive to outliers and dependent on the accuracy of the distance measurement (which is already an additional layer of abstraction over the HOG descriptors).

To increase the effectiveness of the classifier, we perform outlier reduction using Hart's condensed nearest neighbor algorithm [19]. In addition, k-fold cross-validation is performed to determine the optimal value of k to be used. Lastly, we apply the RELIEFF algorithm described by Kononenko et. al to assign variable weights to more important features [20]. RELIEFF is expected to significantly improve the performance of the classifier because certain combinations of pixel neighborhoods in an image cropped tightly around a character can be expected to shaded depending on the character contained within. For example, if a region in the center of the image is filled, it is more probable that the character is an ‘X’ or ‘Y’ rather than an ‘O’ or ‘L’.

Nearest neighbor classifiers suffer at runtime because any classification attempt involves a comparison with almost all, if not all data points in the training set. To speed up the process, the classifier is indexed using a kd-tree, which allows for lookups to skip over large, irrelevant sections of the data.

F. Multi-class Support Vector Machine

A traditional support vector machine (SVM) performs binary classification of any given input. However, it is possible to extend the functionality of SVMs to handle multiple classes using a "one-versus-all" approach.

Given a set of unique training labels l_t as in subsection III-C, it is possible to decompose the problem into a set of $|l_t|$ binary classifiers, each classifying some label $x \in l_t$ such that x yields a positive response, and not x yields a negative response. Each SVM is trained by using the Iterative Single Data Algorithm (ISDA) described by Kecman et. al to minimize soft margins [21], and the final prediction is taken from the classifier that yields the best confidence when classifying x.

G. Feed-Forward Neural Network

We utilize a cascading feed-forward neural network that accepts as input a vector corresponding to the HOG descriptor features for the character being analyzed. Thus the network accepts input as a vector of length n_{HOG} and has one hidden layer with x nodes, where x is defined as follows:

$$x = \arg\max_{l_t} I(l_t) \tag{4}$$

where I is the indexing function and l_t is a training label, both described in subsection III-C.

The formulation of a neural network requires slight modification to the training labels provided to the function—it is not effective to provide l_t directly, because doing so would require the network to perform categorical classification. Instead, l_t should be represented as an indexed vector, i.e. a sparse vector of length x where $x_{I(l_t)} = 1$. In this manner, the output will also be an indexed vector of length x, which can be interpreted to yield a representation in terms of I.

The network is trained by back-propagation using a scaled conjugate gradient.

IV. Results and Discussion

The effectiveness of each classifier is tested using a real-world CAPTCHA data set comprising of images drawn from two different CAPTCHA generating libraries. We use the following function to compare some classification output x with the correct response y for the CAPTCHA:

$$\epsilon(x, y) = 1 - \frac{\text{lev}(x, y)}{\max\{|x|, |y|\}} \tag{5}$$

where $\text{lev}(x, y)$ is the Levenshtein distance between the two strings. In effect, the function evaluates the percentage of the string that is identified correctly. While a caveat exists that an actual CAPTCHA test will likely reject any answer which is not entirely correct, utilizing such a metric would only prevent any fruitful analysis of our results.

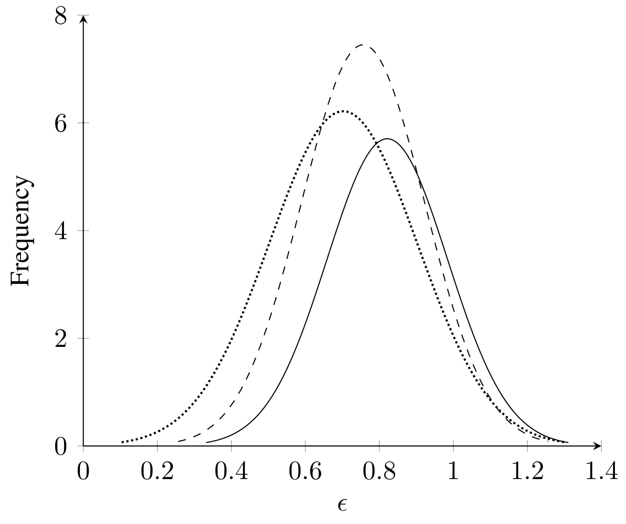


Fig. 3: Distribution of recognition results for the k-nearest neighbor classifier (solid line, $\mu = 0.821$, $\sigma = 0.163$), support vector machine (dashed line, $\mu = 0.756$, $\sigma = 0.166$), and neural network (dotted line, $\mu = 0.7024$, $\sigma = 0.200$).

Figure 3 displays the distribution of recognition rates for the three methods we have considered. It is interesting to note that the k-nearest neighbor approach yields the best results compared to the other two methods. In particular, both the SVM and neural network were more prone to misclassifying characters with similar shapes, such as 'O' and 'Q', and 'B' and '8'. Since a nearest neighbor algorithm performs a feature-by-feature comparison, whereas the SVM and neural network abstract the features into a higher representation, it is expected that the nearest neighbor classifier is more sensitive to such details.

However, one might postulate that the reason the SVM and neural network have failed to perform as well is because more calibration must be done to adjust the parameters for these classifiers such that the best output can be obtained. The convergence parameters for the SVM and neural network are non-trivial to determine, but have a large impact on the performance of the classifier. In comparison, the nearest neighbor classifier is a lot more straightforward because the only parameters that need be considered are the distance metric and feature weights. It is also possible that neural networks and SVMs perform better when isolated to a specific category of problem, i.e. a specific type of CAPTCHA rather than a mixed set. Attempting to generalize either of them might result in over-fitting.

V. Conclusion and Further Work

We have shown that k-nearest neighbor classification, support vector machines, and neural networks provide good performance when identifying character candidates derived from text-based CAPTCHA images. In particular, for the problem formulation we have created, a k-nearest neighbors classifier produces the best results, with an approximate success rate of 82%.

Since the focus of this paper was not on computer vision techniques (or the outright creation of a system designed to circumvent CAPTCHAs in an end-to-end manner), it is immediately clear that better performance on all methods discussed can be achieved by making improvements to the character extraction process [7]. However, the results we have achieved precisely demonstrate that with the appropriate diligence, these methods are robust to a large extent even when the quality of the input is poor. More often than not, this reflects the varying conditions encountered in the real world, as compared to the sterile environment created for purposes of academic discourse.

Recent advances in the fields of deep learning and convolutional neural networks have produced results which far exceed that of the methods we have discussed, both in terms of accuracy and problem complexity. Following in the footsteps of Google, as exemplified by their achievements at circumventing reCaptcha (see subsection I-A), it is reasonable to assume that applying such an implementation to our benchmarking data set would yield results which are indistinguishable from that which a human could provide.

References

[1] K. A. Kluever and R. Zanibbi, “Balancing usability and security in a video captcha,” in *Proceedings of the 5th Symposium on Usable Privacy and Security*. ACM, 2009, p. 14.

[2] Y. Soupionis and D. Gritzalis, “Audio captcha: Existing solutions assessment and a new implementation for voip telephony,” *Computers & Security*, vol. 29, no. 5, pp. 603–618, 2010.

[3] J. Yan and A. S. El Ahmad, “Captcha security: A case study,” *IEEE Security and Privacy*, vol. 7, no. 4, pp. 22–28, 2009.

[4] 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.

[5] A. Gupta, A. Jain, A. Raj, and A. Jain, “Sequenced tagged captcha: Generation and its analysis,” in *Advance Computing Conference, 2009. IACC 2009. IEEE International*. IEEE, 2009, pp. 1286–1291.

[6] 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.

[7] 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*, ser. CCS ’08. New York, NY, USA: ACM, 2008, pp. 543–554. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455839

[8] P. Golle, “Machine learning attacks against the asirra captcha,” in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS ’08. New York, NY, USA: ACM, 2008, pp. 535–542. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455838

[9] E. Bursztein, M. Martin, and J. Mitchell, “Text-based captcha strengths and weaknesses,” in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 125–138.

[10] 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.

[11] A. Hindle, M. W. Godfrey, and R. C. Holt, “Reverse engineering captchas,” in *Reverse Engineering, 2008. WCRE’08. 15th Working Conference on*. IEEE, 2008, pp. 59–68.

[12] K. Chellapilla, K. Larson, P. Y. Simard, and M. Czerwinski, “Computers beat humans at single character recognition in reading based human interaction proofs (hips).” in *CEAS*, 2005.

[13] ——, “Building segmentation based human-friendly human interaction proofs (hips),” in *Human Interactive Proofs*. Springer, 2005, pp. 1–26.

[14] 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.

[15] L. Von Ahn, M. Blum, N. Hopper, and J. Langford, “Captcha: Telling humans and computers apart automatically,” in *Proceedings of Eurocrypt*, 2003.

[16] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[17] T. E. de Campos, B. R. Babu, and M. Varma, “Character recognition in natural images,” in *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, February 2009.

[18] N. Otsu, “A threshold selection method from gray-level histograms,” *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[19] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *Information Theory, IEEE Transactions on*, vol. 13, no. 1, pp. 21–27, 1967.

[20] I. Kononenko, E. Šimec, and M. Robnik-Šikonja, “Overcoming the myopia of inductive learning algorithms with relieff,” *Applied Intelligence*, vol. 7, no. 1, pp. 39–55, 1997.

[21] V. Kecman, T.-M. Huang, and M. Vogt, “Iterative single data algorithm for training kernel machines from huge data sets: Theory and performance,” in *Support vector machines: theory and applications*. Springer, 2005, pp. 255–274.