Reimagining CAPTCHA Security: A Case for Retiring Alphanumeric Text-Based CAPTCHAs

1 Naman Vashishta
namanv.inbox@gmail.com

2 Anshita Bhardwaj
bhardwaj.anshi@gmail.com

3 Pratheek Tirunagari
pratheektirunagari@gmail.com

***Abstract*—We align ourselves to the arguments of [1], [2], [3], [4], highlighting the ease of breaking alphanumeric CAPTCHAs today. CAPTCHAs are meant to be reverse Turing tests, implemented in order to differentiate humans from bots. Today, a number of sites still naively use alphanumeric CAPTCHAs as a first line of defence against automated attacks, flagging themselves as vulnerable targets to potential attackers. Recent advances in convolutional neural networks led way for novel cutting edge object detection [5], [6]. Combined with long-known image segmentation and processing algorithms, old-school alphanumeric CAPTCHAs are solvable to a extremely high degree. In this paper, we aim to show that it is time to phase out alphanumeric text-based captchas once and for all.**

I. Introduction

The problem of CAPTCHA breaking is important because it is essentially a contradiction to the web-based Turing test (and by extension, the reverse Turing test). In the last decade, the use of text-based CAPTCHAs to block automated requests have largely declined, due to improvements in the ability of paid automatic solvers, and the increasing accuracy of machine vision algorithms via convolutional neural networks, which makes it easy for any programmer to prepare and deploy a decently accurate model. However, many production-level systems have previously adopted text-based CAPTCHAs as security strategies and have not updated their systems with a more secure model. Some common types of text-based CAPTCHAs are displayed in Fig. 13.

A. Problem Framing

We frame the CAPTCHA-breaking task as a character-level object detection problem. A correct solution involves producing a subset of characters, chosen from a set of 62 alphanumeric (A-Z, a-z, 0-9) characters that exactly matches the characters that were used to generate the CAPTCHA. We constrain the solution further as follows: the predicted string must match the original string's length and sequence; and all characters are mutually exclusive, meaning that the prediction must be case-sensitive, so the model must be able to handle hard-to-distinguish characters like 'o', 'O' and '0'.

B. Methodology

Our training stage will involve a training set, which we describe in the next section. Input images pass through an Otsu preprocessor, which thresholds the images to remove noise, then are fed into the neural network for training. The test stage is similar, where the test set will also be fed through the Otsu preprocessor, then into the neural network for inference. We use different metrics for character-level prediction (IOUs, class-level mAP) and for accurate CAPTCHA string sequencing ($mAP^{len(captcha)}$). We also introduce a novel step, which we term unseen testing. In this step, our pipeline is fed CAPTCHA images that are totally unseen, independent of the training and test sets. The same metrics apply to evaluate our model.

II. Dataset and Visualization

A. Terminology

All text-based CAPTCHAs are technically 'synthetic', having been rendered at some point using different libraries and languages. However, in principle, we have no knowledge of the exact methods particular sites use to generate their CAPTCHAs. We thus consider scraped or downloaded CAPTCHAs as in-the-wild, or organic data, having been generated and deployed for actual use on production-level systems. We contrast this organic data to images that we are able to generate ourselves, which we will term as synthetic data.

B. Gathering Organic Data

The organic dataset is made up of roughly 15k text-based CAPTCHA images retrieved from the web. The organic portion of our dataset is merged from two online sources. We collected 15k real-world labelled CAPTCHAs of 2 different dimensions from these sources (see Fig. 1):

- 10k black and white, WordPress PHP plugin with over 1+ million downloads, of dimensions (72×24). These are scaled up to (200×75) for training.
- 5k noiseless CAPTCHAs, scraped from online sources, of dimensions (200×75)

(a) Black and white (b) Noiseless

Fig. 1: Organic CAPTCHA classes

(a) Organic (b) Synthetic

Fig. 2: Comparison between organic and synthetic

C. Learning labels from Synthetic Data

While our organic data is immensely valuable as real-world ground truths, we are unable to localize characters within the images in the current dataset. We hypothesize that synthetic data will be able to help us to solve this problem, inspired by synthetic data learning [7].

For this task, we have written a custom Python CAPTCHA generation library that is able to generate character-based CAPTCHAs with variations in font, color, and spacing; carry out transformations like warping, vertical offset, rotation; and create irregularities in the form of background variation, point and line noise, and line and arc occlusion. Most importantly, we are able to create pixel-perfect localization labels for our synthetic data.

D. Deep learning for annotation

In order to validate our hypothesis that we should be able to learn labels for our organic dataset, we analyzed the PHP portion of our dataset. The dataset draws characters exclusively from a set of variants of Gentium (regular, italic, bold and bold italic) with replacement, including random rotations and spacing within certain thresholds. We then generate 50k synthetic samples, which we argue are highly similar to the 10k organic samples, and learn localization labels on the organic set from the ones in the synthetic set, using the same architecture as our final model, RetinaNet [6]. We then manually fixed any labels that were wrong, and fixed bounding boxes that were off. This task was made largely trivial because our proxy annotation model did an extremely good job, and enabled us a semi-automated method of gaining bounding box information for our entire organic dataset.

E. Synthetic Data

Early on, we noted that in this problem, in-the-wild, real-world CAPTCHAs have extremely small dissimilarity to synthetic data: the procedure to generate synthetic data is almost identical to the procedures used to generate 'production-grade' alphanumeric CAPTCHAs. Typically, a 4-6 character alphanumeric string is generated at pseudo-random, then rendered together with variable font families, font weights and sizes, positional offsets, typographic artifacts such as negative kerning, affine transformations, and filtering. There have been many instances of successful learning with synthetic data [2], [7].

To create diversity sufficient to replace the lack of organic data available, we modified the image parameters as follows:

1) Uniform random character sampling: We randomly sample alphanumeric characters, using the full range of 62 upper and lower case ASCII and single digit numerals.

2) Noise generation: We design **curves**, arc and line occlusions that occur randomly, creating $0-3$ horizontal cuts that pass through at least 50% of the letters. We also introduce **dots**, creating $20-35$ dots scattered throughout the image. By creating these artifacts, we hope to increase the invariance to noise in our classification step.

3) Color randomization: Colors are randomly allocated within the RGB spectrum. For visibility, we constrain background colors to values above $(238, 238, 238)$ and foreground colors below $(200, 200, 200)$.

4) Font selection: We render the most common types of fonts in this experiment. In general, there are a few broad classes of fonts, including sans-serif, serif, slab, and monospace. We choose the most common sans-serif and serif fonts from the Google Fonts open-source directory. We picked Droid Sans Mono and Roboto for sans-serifs; and Gentium and Merriweather for serifs. We also introduce different font weights within the font families.

5) Alignment and size: The alignment of the strings are varied. Each of the characters may or may not be in the same line. We define curves with randomized parameters that are used to offset character heights.

F. Visualizations

Our final combined dataset consists of $50,000$ CAPTCHA images, with three main divisions: color, noiseless, and black and white. The table below describes each class of CAPTCHA and its characteristics, and the distribution is shown in Fig. 3.

Class	Character count	Character types
Color	$4-6$	Full alphanumeric
Noiseless	6	Lowercase and digits
BW	4	Uppercase and digits

1) Relationship analysis using PCA & t-SNE: We use Principal Component Analysis and t-distributed Stochastic Neighbour Embedding to grasp the correlation among the characteristics in our dataset, which aids in reducing the

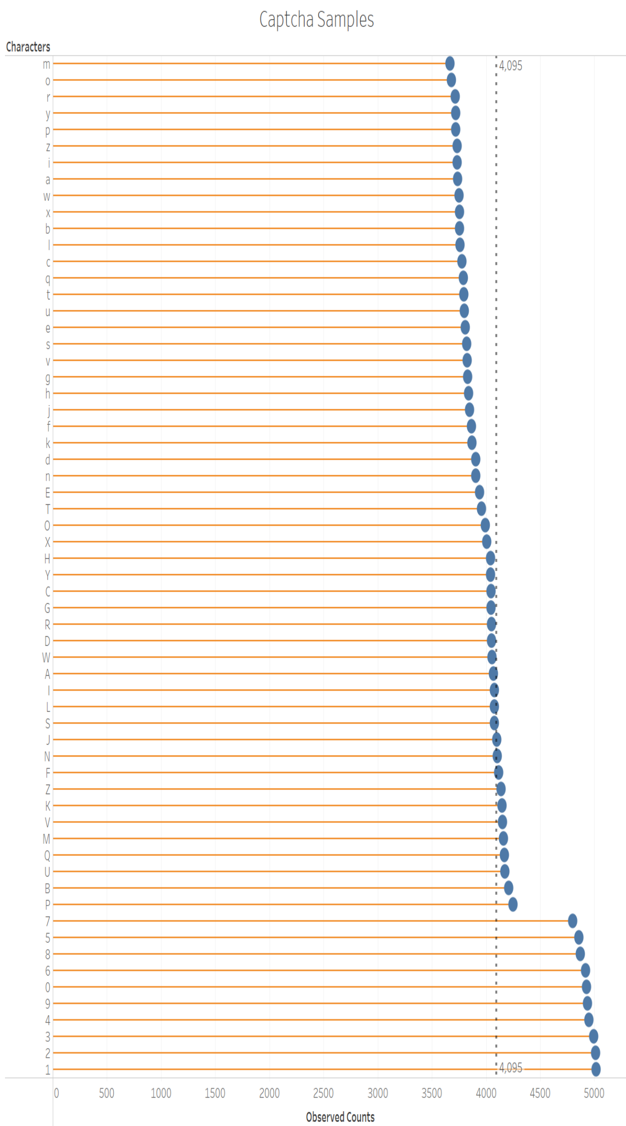


Fig. 3: Distribution of characters in dataset. Notice that the overall dataset is not uniform distributed. The observed counts for letters are significantly less than digits. This is because the lowercase letters are not rendered for the black and white class, uppercase letters are not rendered for the noiseless class, while digits are rendered for all classes.

images' dimensions. Each character in the CAPTCHAs is segmented, extracted and resized to fit a (60×60) pixels box (Fig. 4). Secondly, a threshold transformation is applied to convert the characters into black and white images with a singular RGB value, ranging from 0 to 255. This results in a 3600-dimensional vector of pixel values. Both PCA and t-SNE are employed for dimensionality reduction, seeking the most informative directions or principal components containing significant variance relative to other data points. While PCA focuses on linear reduction, t-SNE adopts a non-linear, probabilistic approach. It converts Euclidean distances into conditional probabilities and applies the Student's-t distribution to calculate similarity metrics between data points. Our results can be seen in Fig. 5 and Fig. 6.

Notably, the PCA scatter plots reveal distinct groupings discernible from the clustered and colorful patterns. However, there is no clear distinction between the clusters. The t-SNE plots are more useful in distinguishing the various characters. For example, the left section of the plot represents characters with mostly straight line segments (l, T, i) while the top section represents characters with looops (o, O, 8, b). Due to the close clustering and similarity between many of the characters, we conclude that the CAPTCHA images we're dealing with are challenging and hence, we expect reduced accuracy scores for certain characters that are identical to others, such as o & O, l & I.

III. Data Processing

A. *Early attempts at algorithmic character segmentation*

We consider several preprocessing techniques, primarily to reduce noise. The studyv aims to simplify input into a form where the letters can be extracted th4rough segmentation algorithms. Some preprocessing algorithms we attempted are:

- **Background/colour removal**. Against CAPTCHAs employing color as a defensive measure, this approach proves highly effective. By converting the image to grayscale, foreground pixels retain their original intensity while background pixels become white. The resulting grayscale image effectively neutralizes the color-based defense mechanism utilized by certain CAPTCHAs. However, there are CAPTCHA variants with gradient and pattern backgrounds, making an algorithmic background classification difficult.
- **Upsampling**. By subdividing the pixels of the image, the segmentation algorithm gains precise manipulation over specific areas. This division enables finer control and enhances the algorithm's ability to affect targeted regions within the image.

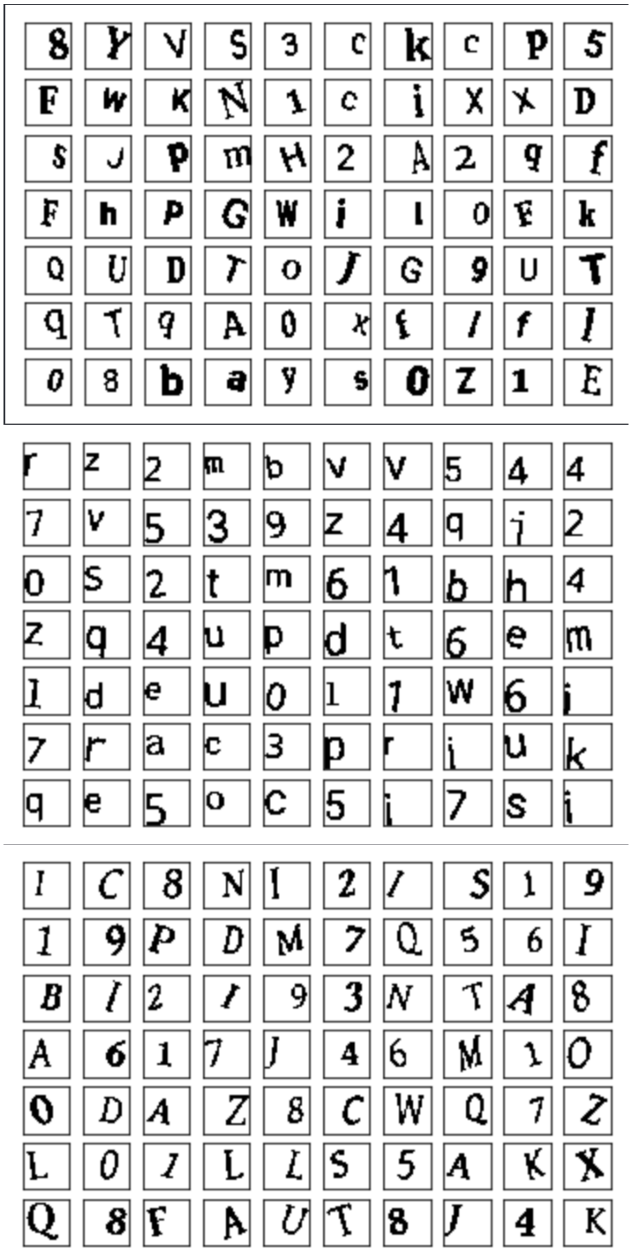


Fig. 4: Color, noiseless and BW class characters for PCA and t-SNE

While upsampling might be useful for complex images like detailed photographs, less complex images like ours do not require it.

- **Thresholding**. By discarding pixels with low intensity, which are considered as noise, a binary image is generated. This transformation into binary form is essential, as numerous segmentation algorithms rely on such input to effectively carry out their operations. We eventually use an augmented form of thresholding in the final

Fig. 5: Principal components of the noiseless and BW sets

preprocessing algorithm.

- **Line removal**.Start by removing the linear segments which are not in the character's group. This is particularly hard as our dataset has arc and line occlusion.

The main focus of our study is to achieve effective text segmentation, specifically targeting the extraction of individual character blocks from the background. Our aim is to precisely locate and isolate the characters within the image. Traditional attacks on CAPTCHA systems rely on scheme-specific techniques, tailored to exploit their vulnerabilities. In our research, we explore the approach of flood-filling segmentation, a method that treats connected components of black pixels as separate entities. By applying this technique to a binarized image, we can accurately identify and extract the collection of objects representing the characters. This segmentation process enables us to overcome the challenges posed by CAPTCHA schemes and achieve successful text extraction and analysis. However, this nearest-neighbour method doesn't account for noise and compression artifacts. We found that some in-the-wild CAPTCHA sets contain JPEG artifacts, making flood-fill a poor choice as a boundary detector.

Fig. 6: t-SNE plots of the color and BW sets

B. Otsu Thresholding

We choose Otsu thresholding [8] as it provides a robust binarization result, only occasionally letting noise through the filter. When analyzing a bimodal image, which refers to an image with a histogram exhibiting two distinct peaks, certain characteristics come into play. keeping the specific image at focus, Otsu thresholding computes a threshold value in the between the value of those peaks. In its quest to minimize the weighted within-class variance, Otsu's algorithm aims to identify an optimal threshold value, denoted as t, based on a specific relationship:

$$\sigma_w^2 = q_1(t) \cdot \sigma_1^2(t) + q_2(t) \cdot \sigma_2^2(t) \tag{1}$$

here, $q_i(t)$ refers to cumulative sum of probabilities, and σ_i^2 is the variance. We first apply a 3x3 Gaussian blur to denoise the input, subsequently running Otsu thresholding on the filtered image. This results in a slightly better segmentation, as seen in Fig. 7.

Utilizing the outcomes of the preceding segmentation phase, all images undergo a binarization process. Following this, a classifier is employed to recognize individual characters. Recent attacks against CAPTCHAs have moved beyond simple criteria like object pixel count, opting instead for machine learning classifiers trained on the segmentation algorithm's output.

(a) Original colored

(b) After preprocessing

Fig. 7: Pre- and post-Otsu thresholding

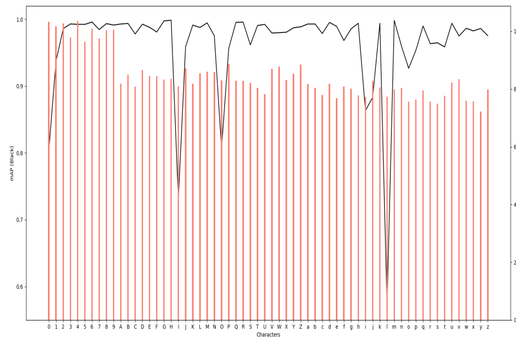


Fig. 8: Results: mAP (black) and number test instances (pink) per character

IV. Model Architecture

We refer cutting-edge object detection-techniques, using the RetinaNet [6] meta-architecture for our classification and localization task. Typically, one-stage detectors like YOLO [9] and SSD [10] demonstrate fast inference with low accuracy. We experiment casually with them, but find that RetinaNet is a faster, one-stage detector with comparable accuracy to slower two-stage detectors, like Feature Pyramid Networks (FPN) [11] and variants of Faster-RCNN [12]. Within this context, we encounter a dual aim: first, to categorize anchor boxes within anchor set A into one of 62 classes, and second, to estimate ground-truth object boxes through regression. To achieve this, focal loss [6] is incorporated to de-emphasize the impact of straightforward classification instances during training, thereby directing the model's focus toward more challenging examples.

$$FL(p_t) = -(1 - p_t)^\gamma \log(p_t) \tag{2}$$

A. Backbone

With a CNN-FPN backbone, the one-stage RetinaNet incorporates two heads for classification and regression tasks. The backbone takes the job of computing the convolutional feature maps over the input image. We theorize that we can build a better model by going deep, since we are able to control the size of the dataset easily. We choose ResNet50 [5] and augment it with a FPN, which creates a multi-scale feature pyramid instead of individual feature maps. This allows each layer to access different feature maps to detect objects at various scales.

B. Classification

For every anchor in the anchor set A and character class, the classification head predicts the likelihood of an alphanumeric character appearing in the image. It is built with four 3x3 convolution layers employing ReLU activations, concluded by a final 3x3 convolution layer containing $(classes \times A)$ filters. A sigmoid activation is used to compute the probability score. The classification task utilizes focal loss for optimization.

C. Regression

Offset is restored from each anchor box by the regression head to an object closet to its reality-state. The convolutional layers in the regression head are identical to the classification head, but the prediction of relativity between the true box and the offset is performed by the sub-network. While the architecture is similar to the classification head, the parameters are kept separate. The regression task uses smooth L1 loss.

D. Training

We train with Keras, using the Adam [13] optimizer for 32k iterations with a batch size of 40, and image augmentations including small amounts of rotation, translation, shear, and scaling. Flipping is not used for obvious reasons.

V. Evaluation Results

We refer to the metrics discussed in Section 1.2. Our results are good on the testing set, reaching a mAP of 0.9624 after training for a total of 32k iterations. Based on Fig. 8, we perform well on the test set on a per-character basis. However, in order to actually break the CAPTCHA, we have to successfully predict all the characters in the image. On average, for a 6-character CAPTCHA, this would mean that we only score $mAP^{len(captcha)} = 0.9624^6 = 0.7945$. Running the tests visually on test samples, as in Figs. 9-11, we find that our predictions are extremely accurate. We then run inference on the unseen set, with an example shown in Fig. 12, to surprising results. Given these results, we are confident that with additional training on even more variants of text-based CAPTCHAs, this network will maintain its high mAP on test sets, while becoming even more robust to unseen data.

Fig. 9: A sample test result on the color set

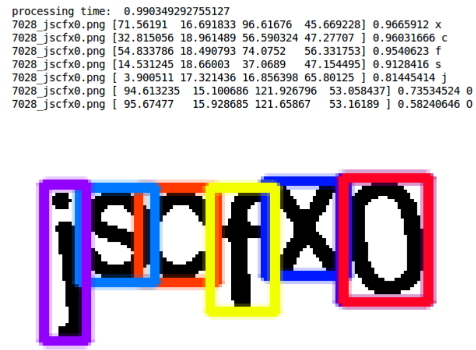


Fig. 10: A sample test result on the noiseless set

VI. Conclusion

Websites must stop using alphanumeric CAPTCHAs as a first line of defence against automated attacks. We show that a simple method to solve alphanumeric CAPTCHAs, even on unseen data, is available to anyone with basic knowledge of image processing and machine learning. It is time to phase out alphanumeric text-based captchas once and for all.

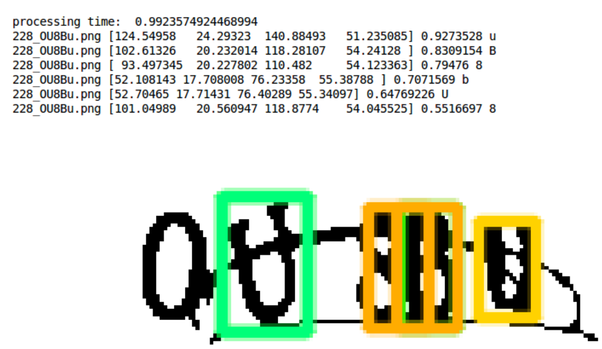


Fig. 11: Failure case on the color set, the only set with noise. We are unable to produce a failure case example for the noiseless and BW sets.

Fig. 12: A test on the unseen set. This CAPTCHA type has never been seen by our model during training. Of the 6 characters in this string, 3q5bed, we are able to correctly predict 5 characters: 3, q, b, e, and d.

REFERENCES

[1] R. Fortune, G. Luu, and P. McMahon, "Cracking captchas."

[2] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Reading text in the wild with convolutional neural networks," *CoRR*, vol. abs/1412.1842, 2014. [Online]. Available: http://arxiv.org/abs/1412.1842

[3] F. Stark, C. Hazırbas, R. Triebel, and D. Cremers, "Captcha recognition with active deep learning," in *GCPR Workshop on New Challenges in Neural Computation*, vol. 10, 2015.

[4] N. Zhao, Y. Liu, and Y. Jiang, "Captcha breaking with deep learning."

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[6] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[7] R. Z. Tuan Anh Le, Atilim Giineş Baydin and F. Wood, "Using synthetic data to train neural networks is model-based reasoning," *Neural Networks (IJCNN) 2017 International Joint Conference*, pp. 3514–3521, 2017.

[8] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[9] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[11] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016. [Online]. Available: http://arxiv.org/abs/1612.03144

[12] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

APPENDIX A - BLOCKCHAIN AND CLOUDSTORE INTERFACES

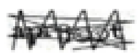
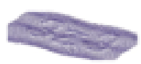
Type	Example	Source	Features
Solid CAPTCHA		Discuz!	Character independent, texture background, some interference
		Slashdot	A large number of interference lines and noise points
		Gimpy	Multiple strings, overlap, distortion
		Google	Unfixed length, distortion, adhesion
		Microsoft	Double-string, unfixed length, uneven thickness, tilting, adhesion
Hollow CAPTCHA		QQ	Hollow, shadows, interference shapes
		Sina	Hollow, adhesion, interference lines
		Yandex	Hollow, virtual contours, distortion, adhesion, interference lines
Three-dimensional CAPTCHA		Scihub	Hollow, shadows, interference lines, noise points
		Teabag	Grids, protrusion, distortion, background and character blending
		Parc	Colorful, shadow, rotation, zoom Special characters
Animation CAPTCHA		Program generating	Multiple characters jumping
		Hcaptcha	Multilayer character images blinking transformation

Fig. 13: Common types of text-based CAPTCHAs used on the web