Using Generative Adversarial Networks to Break and Protect Text Captchas

GUIXIN YE, ZHANYONG TANG, and DINGYI FANG, Northwest University, China
ZHANXING ZHU and YANSONG FENG, Peking University, China
PENGFEI XU and XIAOJIANG CHEN, Northwest University, China
JUNGONG HAN, University of Warwick, United Kingdom
ZHENG WANG, University of Leeds, United Kingdom

Text-based CAPTCHAs remains a popular scheme for distinguishing between a legitimate human user and an automated program. This article presents a novel genetic text captcha solver based on the generative adversarial network. As a departure from prior text captcha solvers that require a labor-intensive and time-consuming process to construct, our scheme needs significantly fewer real captchas but yields better performance in solving captchas. Our approach works by first learning a synthesizer to automatically generate synthetic captchas to construct a base solver. It then improves and fine-tunes the base solver using a small number of labeled real captchas. As a result, our attack requires only a small set of manually labeled captchas, which reduces the cost of launching an attack on a captcha scheme. We evaluate our scheme by applying it to 33 captcha schemes, of which 11 are currently used by 32 of the top-50 popular websites. Experimental results demonstrate that our scheme significantly outperforms four prior captcha solvers and can solve captcha schemes where others fail. As a countermeasure, we propose to add imperceptible perturbations onto a captcha image. We demonstrate that our countermeasure can greatly reduce the success rate of the attack.

CCS Concepts: • **Security and privacy** → *Graphical/visual passwords*; **Authentication**;

Additional Key Words and Phrases: Text captchas, generative adversarial networks, transfer learning, security, authentication

ACM Reference format:
Guixin Ye, Zhanyong Tang, Dingyi Fang, Zhanxing Zhu, Yansong Feng, Pengfei Xu, Xiaojiang Chen, Jungong Han, and Zheng Wang. 2020. Using Generative Adversarial Networks to Break and Protect Text Captchas. *ACM Trans. Priv. Secur.* 23, 2, Article 7 (April 2020), 29 pages.
https://doi.org/10.1145/3378446

Extension of Conference Paper: a preliminary version of this article entitled "Yet Another Text Captcha Solver: A Generative Adversarial Network Based Approach" by G. Ye et al. appeared in ACM Conference on Computer and Communications Security, 2018 [74]. The work was partly supported by the National Natural Science Foundation of China (NSFC) through Grant Agreements No. 61972314, No. 61672427, and No. 61872294; in part by the International Cooperation Project of Shaanxi Province (2019KW-009) and the Ant Financial through the Ant Financial Science Funds for Security Research.

Authors' addresses: G. Ye, Z. Tang (corresponding author), D. Fang, P. Xu, and X. Chen, Northwest University, China; emails: gxye@stumail.nwu.edu.cn, {zytang, dyf, pfxu, xjchen}@nwu.edu.cn; Z. Zhu and Y. Feng, Peking University, China; emails: {zhanxing.zhu, fengyansong}@pku.edu.cn; J. Han, University of Warwick, United Kingdom; email: jungong.han@warwick.ac.uk; Z. Wang (corresponding author), University of Leeds, United Kingdom; email: z.wang5@leeds.ac.uk.

2471-2566/2020/04-ART7 $15.00
https://doi.org/10.1145/3378446

1 INTRODUCTION

The completely automated public Turing test, or CAPTCHA[1] for short, is often used to distinguish legitimate users from malicious bots [67]. Captchas exist in various forms, including texts [66, 67, 68], images [13], audio [55], video [56], and games [19]. Among these, text captcha is a popular scheme and remains being used by the majority of the top-50 popular websites ranked by alexa.com, including Microsoft, Google, eBay, and many others.

Breaking a particular captcha scheme[2] is rarely news today. This is a heavily studied area, and many scheme-specific captcha solvers have been proposed in the past. The seminal work presented by Greg and Malik dated back to 2003 was among the first attempts to automatically solve text captchas [28]. However, most of the prior attacks are specifically tuned for a few specific captcha schemes and adapting them for a new scheme would require significant human intervention for tuning the model and collecting training data—manually labeled captcha images. Just like cryptography, text captchas are evolving and becoming more robust where many of the advanced features make prior attacks no longer applicable [18].

By employing analytical models and algorithms, some of the more recent works have improved the generalization ability of a captcha solver [8, 10, 18]. The idea behind such schemes is that a captcha solver can be tuned to target a new scheme by changing and adapting the model parameters and algorithm thresholds. These schemes, however, are only effective in solving text captchas with simple security features. Their success often relies on a good character segmentation method [12], but the recent development of text captchas has made character segmentation more challenging by introducing advanced security features like more complex backgrounds as well as distorted and overlapping characters.

In this article, we present a new approach for building text captcha solvers. Compared to prior attacks, our approach requires significantly fewer numbers of manually labeled captchas but delivers better performance for solving a wider range of schemes. Our work is inspired and enabled by the recently proposed generative adversarial network (GAN) [24] and its breakthrough effectiveness in image translation tasks [35]. To construct a solver for a given captcha scheme, we first automatically learn a GAN-based captcha synthesizer using a small set of labeled real captcha images. Next, we use the learned synthesizer to automatically generate a large number of training samples without human involvement, from which we learn a base solver. We then apply transfer learning [52] to fine-tune and improve the base solver. As a significant departure from prior attacks, our approach greatly reduces the cost and human efforts in creating and tuning a captcha solver as well as the underpinning analytical models and algorithms. Our approach is generally applicable, because *the process* for building a solver is mostly automatic and is not coupled to a specific scheme. We show that our approach can result in a highly effective solver for a large set of currently used text captcha schemes, making our attack a severe threat to text captchas.

We evaluate the proposed scheme through extensive experiments. We apply our approach to 33 text captcha schemes, 11 of which were being used by 32 of the top-50 popular websites ranked by alexa.com as of April 2019. We compare our approach to four prior captcha solvers [8, 10, 18, 20]. Experimental results show that our approach needs as few as 500 as opposed to millions [23] labeled captcha images to learn a successful solver. Despite our approach uses a significantly fewer number of real captchas, it gives a higher success rate. Experimental results show that our approach can successfully crack all tested schemes, judged by the commonly used standard [10], and it can solve a captcha in less than 50 milliseconds using a modest desktop GPU.

[1] To aid readability, we will use the acronym in lowercase thereafter.

[2] In this article, the term *breaking captchas* refers to automatically solve the captcha challenge using a computer program, i.e., recognizing the characters of a text captcha image.

As a countermeasure, we turn again to the GAN framework. We show that by inserting some imperceptible perturbations or noise to the captcha images, one can significantly decrease the effectiveness of our attack. This provides a new way to protect the popular text captcha schemes against machine-learning-based attacks before a better alternative is adopted.

To sum up, this article makes the following technical contributions. It is the first to:

- employ the generative adversarial paradigm to build a successful solver for text captchas based on a small number of real captcha images;
- apply transfer learning to fine-tune a captcha solver that learned from synthetic data;
- show how a generic learning-based approach can be applied to target a rich set of captcha schemes, which not only requires less human efforts to construct but also leads to better performance over prior attacks;
- propose a new countermeasure for text captchas based on the generative adversarial paradigm.

2 BACKGROUND

In this section, we present the threat model and introduce the preliminaries of text captchas and the GAN architecture.

2.1 Threat Model

Like many prior works, our attack employs supervised learning techniques to build a captcha solver. The quality of a machine-learned model depends on the volume and quality of the training data. In this work, we assume the adversary has access to a small number of manually labeled captcha images for the target scheme. We refer these captchas as *real captchas*, because they are generated by the target scheme. The real captchas can be labeled either by the attacker or paid crowdsourcing workers. Specifically, our attack needs as few as 500 real captchas to build a successful captcha solver. Prior attacks based on machine learning models often require thousands or sometimes millions of examples to learn a good captcha solver [1, 22, 23]. For example, the work presented in [23] requires millions of captcha images to learn an effective CNN model to solve reCAPTCHA. Compared to these prior attacks, our approach incurs significantly less overhead and cost for collecting and labeling the data.

We also assume the adversary has sufficient computing power to run machine learning algorithms. In this article, we show that the learning can be performed on a typical GPU cloud server, and the learned solver can run efficiently on a modest desktop GPU.

2.2 Security Features of Text Captchas

A text captcha image often consists of distorted characters, a noisy background or occluding lines, which are coined as security features. Without loss of generality, to make our experiments manageable, we restrict our scope to six widely used security features employed by the current text captcha schemes. They are used by the top-50 popular websites ranked by alexa.com at the time this work was conducted.

Figure 1 illustrates some of the security features targeted in this work. These include anti-segmentation and anti-recognition features. The *anti-segmentation* feature, labeled as 1, 2, and 3 in Figure 1, aims to increase the difficulty of character segmentation. A *anti-recognition* feature, however, makes it difficult for a computer program to recognize the characters. This is achieved by using a variety of font styles and distorted characters, as depicted in Figure 1 with labels 4, 5, and 6. Later, in Table 1, we summarize how these security features are used in different captcha schemes.

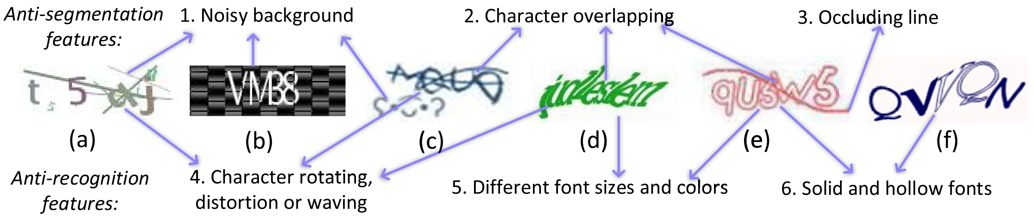

Anti-segmentation features:
1. Noisy background
2. Character overlapping
3. Occluding line
(a)
(b)
(c)
(d)
(e)
(f)
Anti-recognition features:
4. Character rotating, distortion or waving
5. Different font sizes and colors
6. Solid and hollow fonts

Fig. 1. Security features of current text-based captchas used in this work. Labels 1, 2, 3 show the *anti-segmentation* features and labels 4, 5, 6 present the *anti-recognition* features.

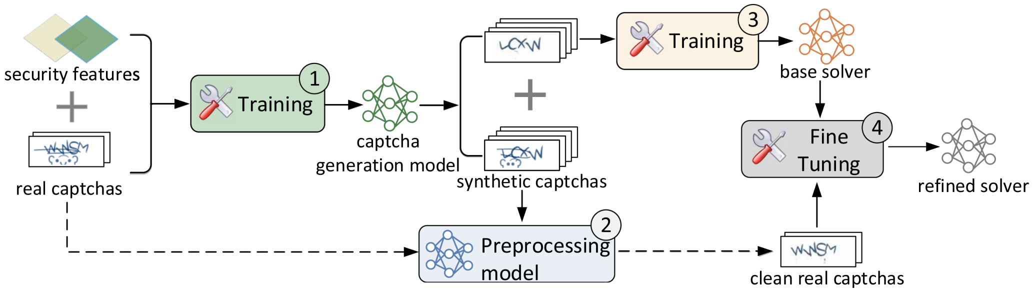


Fig. 2. Overview of our approach. ① We first use a small set of real captchas and the security features of the target scheme to learn a captcha generation model. ② The captcha generation model is then applied to automatically generate synthetic captchas (with and without confusing background patterns) to learn a preprocessing model to remove security features, e.g., noisy backgrounds and occluding lines, from the input captcha image. ③ At the same time, the synthetic captchas (without security features) are used to train a base solver. ④ Finally, we use a few clean real captchas (that have been processed by the preprocessing model) to fine-tune the base solver to build the final solver.

2.3 Generative Adversarial Networks

Our work is the first to apply the recently proposed GAN architecture [24] to learn a captcha solver. A classical GAN consists of two modules. The first is a *generative* network for generating synthetic data, and the second is a *discriminator* network to filter out the synthetic examples from the real ones. To train the generative and discriminator networks, we use *backpropagation* [31], a well-established training method for neural networks. During each training iteration, the generator aims to produce better synthetic samples while the discriminator would become more skilled at flagging synthetic samples. GANs have demonstrated promising results in natural language processing [42, 76] and image generation [35, 77] tasks.

3 OVERVIEW OF OUR APPROACH

Figure 2 depicts the four steps of building a captcha solver using our approach. Each of the steps is described as follows.

① ***Training data synthesis.*** To reduce the efforts for collecting and labeling real captchas and at the same time provide sufficient training data to build an effective captcha solver, we seek ways to generate synthetic training data. We do so by learning a captcha synthesizer for a target captcha scheme (Figure 2 ①). Our captcha synthesizer is a neural network trained under the generative adversarial paradigm. Our GAN consists of two components. The first is a captcha generation model that tries to produce captchas that are as similar as possible to the target captchas. The second is a discriminator that tries to identify the synthetic captchas from the real ones. This

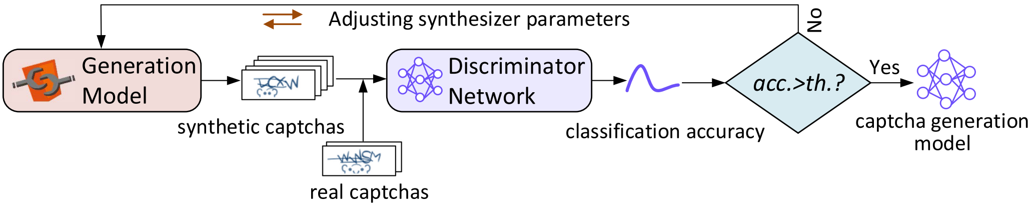


Fig. 3. The training process of our GAN-based text captcha synthesizer.

generation-discrimination process terminates when the discriminator fails to identify a large portion of the synthetic captchas. After training, we then use the learned captcha generation model to automatically produce a large number of captchas together with their characters. This is described in Section 4.1.

② ***Preprocessing.*** To assist the captcha solver, we build a preprocessing model (Figure 2 ②) to remove as much captcha security features as possible. The preprocessing model also tries to standardize the font style by e.g., filling hollow characters and standardizing spaces or gaps between characters. We leverage a specific GAN called Pix2Pix [34] to build the pre-processor model. The pre-processor model is trained by using *solely* synthetic captcha samples. Each training sample contains two captcha images: one has security features, and the other does not. We learn a preprocessing model for each captcha scheme and the training process is fully automatic. We describe this process in more details at Section 4.2.

③ ***Training the base solver.*** In this step, we use the *preprocessed synthetic* captcha images together with their corresponding character labels to learn a base solver (see Figure 2 ③). Our base solver is a standard Convolutional Neural Network (CNN). The trained solver takes in a preprocessed captcha image and outputs the corresponding label. This is detailed in Section 4.3.

④ ***Fine-tuning the base solver.*** In this final step, we apply transfer learning to further improve the base solver (see Figure 2 ④). Specifically, we use the set of manually labeled real captchas that we used to train the captcha synthesizer to update the weights at some network layers of the base solver. This is described with more details in Section 4.3.

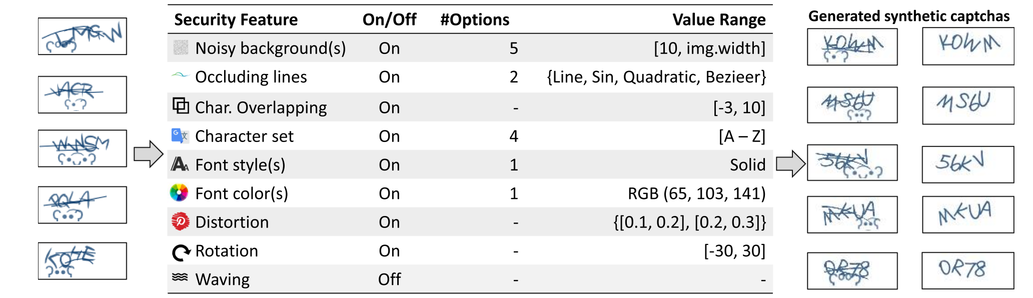
4 IMPLEMENTATION DETAILS

This section provides details on how to build a captcha synthesizer to generate synthetic training data (Section 4.1), and how to learn a preprocessing model (Section 4.2) and a captcha solver (Section 4.3) using synthetic captcha images.

4.1 Training Data Synthesis

Prior work shows that to learn an effective CNN-based solver for text captchas would require as many as 2.3 million of labeled training samples [20]. Collecting and labeling such a large volume of captchas would require intensive human efforts and incur significant cost. Our approach overcomes this issue by using synthetic training data. To this end, we first learn a captcha synthesizer and use the synthesizer to populate the training data with a large number of synthetic captchas that are similar to the target captchas. This allows the training dataset to cover the problem space far more finely than what could be achieved by exclusively using manually labeled real captchas.

As we have briefly described in Section 3, our GAN-based captcha synthesizer consists of a captcha generation model and a discriminator. Figure 3 illustrates the process of training a captcha synthesizer. The training process is largely automatic except that it needs 500 manually labeled



Security Feature	On/Off	#Options	Value Range
Noisy background(s)	On	5	[10, img.width]
Occluding lines	On	2	{Line, Sin, Quadratic, Bezieer}
Char. Overlapping	On	-	[-3, 10]
Character set	On	4	[A – Z]
Font style(s)	On	1	Solid
Font color(s)	On	1	RGB (65, 103, 141)
Distortion	On	-	{[0.1, 0.2], [0.2, 0.3]}
Rotation	On	-	[-30, 30]
Waving	Off	-	-

(a) Real Baidu captchas of different security features

(b) Synthetic parameters

(c) w/ security features

(d) w/o security features

Fig. 4. Example synthetic captchas for Baidu scheme. Our captcha synthesizer is trained using a set of real captchas (a). The parameter setting (b) defines the security feature space. The trained captcha synthesizer is used to produce synthetic captchas with (c) and without (d) the security features (i.e., noisy backgrounds and occluding lines in this example) included.

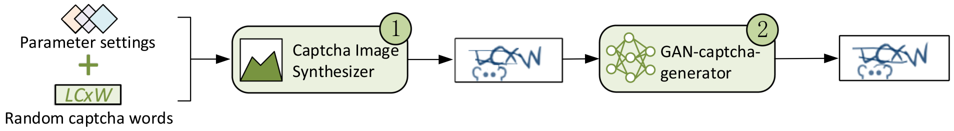


Fig. 5. Overview of our captcha generation model. Our generator model includes an image synthesizer (①) and a GAN-captcha-generator (②). The image synthesizer takes in a word of characters and the security feature setting to produce an initial captcha image. The GAN-captcha-generator then modifies the initial captcha image at the pixel level, aiming to make the resultant captchas are similar to the ones of the target scheme. Once the training process is completed, the captcha generator model can be used to automatically generate the captcha images based on any given word of characters.

real captcha images of the target scheme and a set of user-defined security features. The security feature definition is given by setting a set of pre-defined parameters. As an example, Figure 4 lists all pre-defined parameters of the Baidu captcha scheme. For this example, the waving feature is turned off as it is not used by the Baidu scheme. It is to note that these parameters can be easily extended and adjusted to target other captcha schemes.

Captcha generation model. Figure 5 shows that our captcha generation model is comprised of a captcha image synthesizer and a GAN captcha generator. The image synthesizer automatically generates captcha images for a given parameter setting and a sequence of characters (i.e., a word), while the GAN captcha generator modifies the synthetic captcha at the pixel level. The image synthesizer takes in a security feature configuration and tries to find a set of parameter values so that the synthetic captchas are as similar as possible to the ones from the target captcha scheme. We use the grid search method presented in Reference [4] to find the optimal parameters for a given captcha scheme. Like the image generator, the GAN captcha genertor learns how to modify the generated images at the pixel level so that the resulting captcha contains security features that are similar to the real ones of the target scheme. The similarity is measured by the ratio of synthetic captchas that cannot be distinguished from the real ones by the discriminator. In other words, the more synthetic captchas that can "fool" the discriminator, the higher quality the generated synthetic captchas will be. We also use the similarity score to update the parameter values of the captcha image synthesizer during the grid search process. Specifically, if the similarity score is above 0.65, the parameter values will be reduced according to a given attenuation coefficient, or vice verse. It is to note that once the captcha generation model is learned, it can automatically generate a synthetic captcha image based on any given characters.

Captcha discriminator. Our discriminator model is also a CNN defined in Reference [59]. The last layer of the CNN gives the probability of an input captcha being a synthetic one. We use batches of captcha images to train the discriminator, where each mini-batch consists of randomly sampled synthetic captchas, x, and real captchas, y, and the target labels are 1 for every x_i and 0 for every y_j. The discriminator network updates its parameters by minimizing the following loss function:

$$\mathcal{L}_D = -\sum_i \log D(x_i) - \sum_j \log(1 - D(y_j)), \tag{1}$$

where $D(\cdot)$ is the probability of the input being a synthetic captcha, and $1 - D(\cdot)$ is that of a real one. In this work, we use the Jensen-Shannon divergence [15] to evaluate the difference of the distribution between the synthetic and real captcha images when training the discriminator. We have also considered the Wasserstein distance [2] during our initial experiment but found that the Jensen-Shannon divergence works better in our problem setting. Specifically, we found that the Jensen-Shannon divergence metric can be used to better distinguish between two real and synthetic captures that are visually similar. This capability helps us to better optimize the generation parameters to improve the performance of the captcha generation model.

Training. We use the minibatch stochastic gradient descent (SGD) and the Adam solver [38] with a learning rate of 0.0002 to train our captcha synthesizer. The objective of our captcha synthesizer can be expressed as

$$\mathcal{L}_{cGAN} = E_{x,y \sim p_{data}(x,y)}[logD(x,y)] + E_{x \sim p_{data}(x), z \sim p_z(z)}[log(1 - D(x, G(x,y)))], \tag{2}$$

where x and y are a synthetic and a real captcha, respectively, and z is the noise.

Our overall training objective follows the general GAN approach [59], using the $L1$ norm with the regularization term λ set to 0.0001. The training objective is defined as

$$G^* = \arg\min_G, \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G), \tag{3}$$

where the generator, G, tries to minimize the difference between the generated captchas and the real ones, while the discriminator, D, seeks to maximize it.

Here, the $L1$ loss function is defined as

$$\mathcal{L}_{L1}(G) = E_{x,y \sim p_{data}(x,y), z \sim p_z(z)}[||y - G(x,z)||_1]. \tag{4}$$

During training, when updating the parameters of the synthesizer, we fix the parameters of the discriminator; and when updating the discriminator, we fix the parameters of the synthesizer. Training terminates when the discriminator fails to identify more than 5% of the synthetic captchas. Once the synthesizer is trained, it can be used to quickly generate synthetic captchas. In our case, it takes less than one hour to generate a million captcha images.

Working example. We use the Baidu captcha scheme as a working example to illustrate the process for training a captcha sythesizer. The training process consists of multiple steps. In the initial step, we provide some (i.e., 500) real captchas for the GAN learning engine. We also give the initial parameter values for the captcha image synthesizer. Similarly, the GAN captcha generator is initialized with random weights. During each iteration of the GAN training process, the captcha generation model (that consists of the captcha image synthesizer and the GAN captcha generator) a batch of synthetic captchas that are examined by the captcha discriminator. If the discriminator can successfully distinguish a large number of synthetic captchas from the real ones, then the

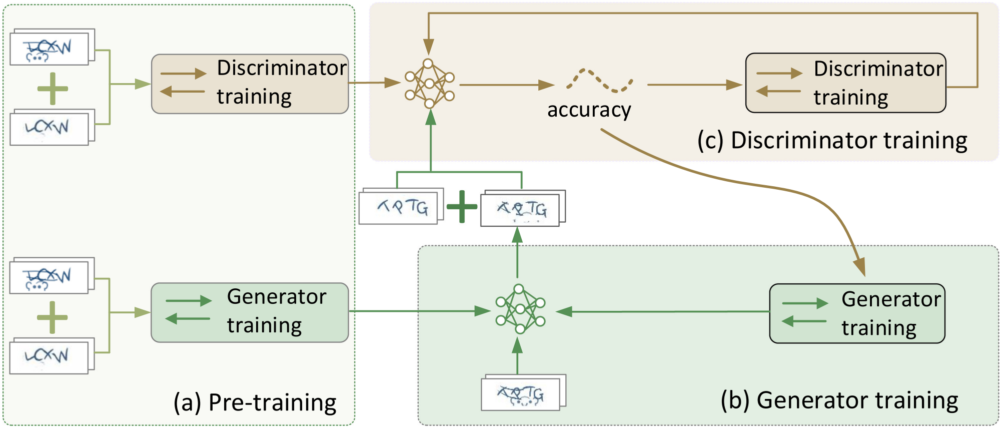


Fig. 6. The training process of our GAN-based pre-processing model. The generator tries to remove as much noisy backgrounds and occluding lines from the input captchas, while the discriminator tries to identify which of the input clean captchas are produced by the generator. All the captchas used in the training are generated by our captcha generation model.

grid search method is employed to adjust the parameter values for synthesizing another batch of captchas. This iteratively training process continues until the discriminator can distinguish less than 5% of the synthetic captchas from the real ones (see Section 6.5). When the process is terminated, the learning engine will output the optimal parameter values to be used by the captcha image synthesizer and the GAN captcha generator for synthesizing captcha images with security features. To generate captchas without security features, we simply turn off the feature option of the captcha image synthesizer. For examples, Figure 4(a) shows real Baidu captchas and Figures 4(c) and 4(d) are the synthetic captchas with and without background security features produced by our captcha generation model. As can be seen from the figure, the security features of the synthetic captchas are visually similar to the real captchas.

4.2 Captcha Preprocessing

Modern captcha schemes often integrate advanced security features like a noisy background (Figures 1(a), 1(b), and 1(c)) and distorted hollow fonts (Figures 1(d), 1(e), and 1(f)). These features make prior pre-processing methods like [17, 73] invalid (see Section 6.4). In our work, we build a GAN-based preprocessing model to remove these security features. Like the synthesizer, we train a preprocessing model for each captcha scheme. In our initial experiment, we also tried to build a general pre-processing model across different captcha schemes. However, we found that a scheme-specific model performs better. Note that we use only synthetic captchas to train the preprocessing model. Specifically, we adopt the *Pix2Pix* image-to-image translation framework [34], which was originally developed to transform an image from one style to another. In our case, the images to be translated are captcha images with background noise such as the Baidu captcha shown in Figure 1(b) or different font styles such as the Microsoft captcha shown in Figure 1(d). Note that our model removes multiple security features (e.g., Figure 4(b)) at once.

Our GAN-based preprocessing model also consists of a generator and a discriminator. Figure 6 depicts the training process. The generator works at the pixel level, which tries to amend some pixels of the input captcha images (e.g., removing noise from the background shown as Figure 6(b)). By contrast, the discriminator tries to distinguish the preprocessed captchas from the *clean captchas* that are produced by the captcha generation model described in Section 4.1.

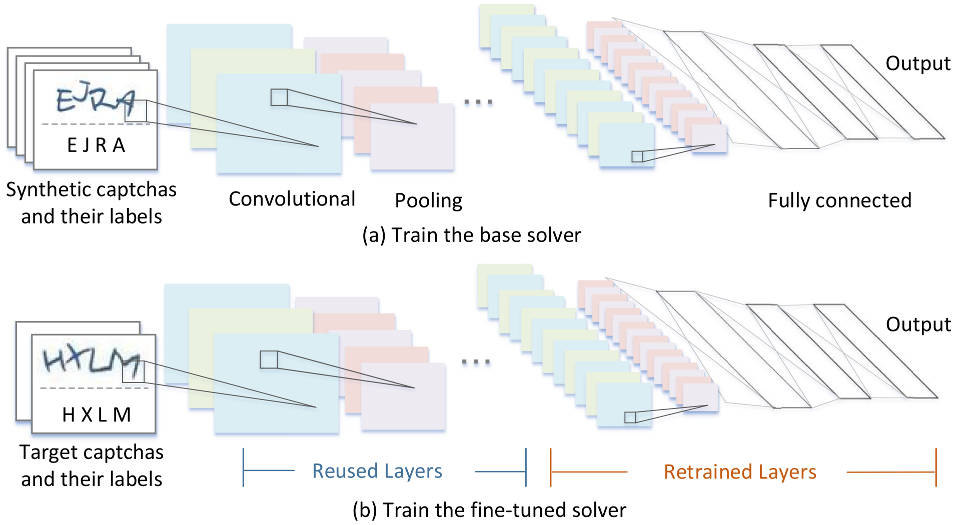

EJRA
E J R A
Synthetic captchas and their labels
Convolutional
Pooling
...
Output
Fully connected
(a) Train the base solver
HXLM
H X L M
Target captchas and their labels
...
Output
Reused Layers
Retrained Layers
(b) Train the fine-tuned solver

Fig. 7. Our CNN-based captcha solver. We first use synthetic captchas to train the based solver (a) which is then refined using a small number (500 in this work) of real captchas (b).

Training. Before training, we first pre-train an initial generator and discriminator using some synthetic captchas (Figure 6(a)). The captchas used in the pre-training process are organized as pairs where each pair contains (1) a synthetic captcha image with the target security features and (2) a corresponding image without these security features. Once the pre-training process is finished, we continue to train them under the generative adversarial framework. The training process is similar to how we train our captcha synthesizer (Section 4.1). Over time, the generator would become better in removing security features, and the discriminator would become better in recognizing security features (even the changes are small). Training terminates when the discriminator fails to identify more than 5% of the preprocessed images from the clean counterparts (Figure 6(c)). After that, we use the trained generator to preprocess *unseen* captcha images of the target scheme.

4.3 Build and Fine-tune the Solver

To build a captcha solver, we follow a two-step approach. We first train a base solver from *synthetic* captchas. We then fine-tune the base solver using the same set of *real* captchas used to build the captcha synthesizer.

Network structure of our solver. Our captcha solver is built upon a classical CNN called *LeNet-5* [41], and it tries to identify the characters of the preprocessed captchas. Unlike *LeNet-5*, which was initially designed to recognize single characters, we introduce some additional layers (2× convolutional and 3× pooling layers) to extend its capability to recognize multiple characters. Figure 7(a) shows the structure of our solver, which has five convolutional layers, five polling layers followed by two fully connected layers. Each of the convolutional layers is followed by a pooling layer. We use a 3×3 filter for the convolutional layer and a max-pooling filter for the pooling layer. We use the default parameters of *LeNet-5* for the rest of the network structures.

It is to note that we have also considered other influential CNN structures including *ResNet* [30], *Inception* [64] and *VGG* [60]. We found that there is little difference in solving text captchas among these models. We choose *LeNet-5* due to its simplicity, which gives the quickest inference (i.e., prediction) time and requires the least training samples for fine-tuning the base solver.

Training the base solver. We train a base solver for a target captcha scheme. In case that the number of characters of a captcha image is not fixed for a scheme, we also train a base solver for each possible number of characters. We use 200,000 synthetic captchas generated by our captcha generation model to train the base solver. Each training sample consists of a *clean* captcha (produced by the preprocessing model) and an integer vector that stores the character IDs of the captcha. Note that we assign a unique ID to each candidate character of the target captcha scheme. We use a Bayesian-based parameter tuner [21] to automatically choose the hyperparameters for training the base solver. Training a base solver takes around five hours using four NVIDIA P40 GPUs on a cloud server (see Section 5.3). The trained base solver can then be applied to any *unseen* captcha image of the target scheme.

Refining the base solver. To fine-tune the base solver, we apply *transfer learning* [75] to update later layers (i.e., those that are closer to the output layer) of the base solver, by using the 500 labeled real captchas that were previously used to train the synthesizer. The idea of transfer learning, in a nutshell, is that in neural network classification, information learned at the early layers of neural networks (i.e., closer to the input layer) will be useful for multiple classification tasks. The later the network layers are, the more specialized the layers become [52]. We exploit this property to calibrate the base solver to minimize any bias and over-fitting that may arise from the synthetic training data.

Figure 7(b) illustrates the process of applying transfer learning to refine the base solver. Transfer learning in our context is as simple as keeping the weights of the early layers and then update the parameters of the later layers by applying the standard training process using the real captchas. This process takes less than 5 minutes on our training platform.

5 EXPERIMENTAL SETUP

5.1 Captcha Schemes

Our evaluation targets 11 current text captcha schemes used by 32 of the top-50 popular websites ranked by alexa.com.[3] We note that some of the websites use the same captcha scheme, e.g., Youtube uses the Google scheme, and Live, Office and Bing use the Microsoft scheme. The websites we examined cover a wide range of domains including e-commerce, social networks, search, and information portals. Table 1 gives some examples of the captcha schemes tested in this work. We note that many captcha schemes exclude some specific characters that are likely to cause confusion after performing the character distortion, for improving the usability of the captchas. Examples of such characters include "o" and "0," "1," and "l," and so on (see Table 1).

In addition to the 11 current schemes, we also extend our evaluation to 22 other captcha schemes (See Table 5) used in prior studies to provide a fair comparison with previous attacks. It is worth mentioning that while we collected the captchas from the official websites, many of the captcha schemes we tested are also used by third-party websites and applications as a security mechanism.

5.2 Collecting and Synthesizing Captchas

We use two sets of captchas in evaluation: one for training and the other for testing. Most of training data are synthetic captchas generated by our captcha generation model. The testing data are collected from the target website for training and testing our GAN-based synthesizer and the fine-tuned solver.

Synthesizing training captchas. We first initialize the security feature parameters as described in Section 4.1 and then use the initial parameters to generate the first batch of synthetic

[3] We have refreshed the captcha dataset used in our previous work [74] when conducting this evaluation.

Table 1. Text-based Captcha Schemes Tested in Our Experiments

Scheme	Website(s)	Example	Security Features		Excluded Characters
			Anti-segmentation	Anti-recognition	
Google	google.{com,co.in,co.jp, co.uk,ru,com.br,fr com.hk,it,ca,es,com.mx} youtube.com		Overlapping characters, Enligh letters	Varied font sizes & color, rotation, disortion and waving	–
Microsoft	{live, bing, miscosoft, office, linkedin}.com		Overlapping characters, solid background	Different font styles, varied font sizes, rotation, waving	0, 1, 5 D, G, I, O, U
Alipay	{alipay, tmall, taobao, login.tmall, alipayexpress}.com		English letters and arabic numerals, overlapping characters	Rotation and distortion	0, 1 I, L, O
eBay	ebay.com		Overlapping characters, Only arabic numerals	Rotating, distortion and waving	–
Wikipedia	wikipedia.org		Overlapping characters, Enligh letters	Rotation, distortion and waving	–
Baidu	{baidu, qq}.com		Occluding lines, character overlapping, only Enligh letters	Varied font size, color, rotation, disortion and waving	Z
Sina	sina.cn		English letters and arabic numerals, overlapping characters	Rotation, distortion and waving	1, 9, 0 D, I, J, L, O, T i, j, l, o, t, g, r
Weibo	weibo.cn		English letters and arabic numerals, overlapping characters, occluding lines	Rotation and distortion	0, 1, 5 D, G, I, Q, U
Sohu	sohu.com		Complex background, occluding lines, and overlapping	Varied font size, color and rotation	0, 1 i, l, o, z
Qihu360	360.cn		English letters and arabic numerals, overlapping characters	Varied font sizes, rotation and distortion	0 I, L, O, T i, l, o, t, q
JD	jd.com		English letters and arabic numerals, overlapping characters	Rotation and distortion	0, 1, 2, 7, 9 D, G, I, J, L, O, P, Q, Z

captchas—which are then used together with 500 real captchas to train our synthesizer. After we have trained the synthesizer, we then use it to generate synthetic samples to learn the preprocessing model and the base solver. Specifically, we use 20,000 and 200,000 synthetic captchas to train the preprocessing model and the base solver, respectively.

Collecting testing captchas. The real captchas are automatically collected using a web crawler written in Python. Each collected captcha is manually labeled by three paid participants (nine participants in total) recruited from our institution. We use only captchas where a consensus has been reached by all the three annotators. In total, we have used 1,500 real captchas for each target scheme. We randomly divided the collected captchas into two sets, one set of 500 captchas for training our synthesizer and the final solver, and the other set of 1,000 captchas for testing our solver. It takes up to 30 minutes (less than 10 minutes for most schemes) to collect 500 captchas and less than 2 hours to label them by one user. This suggests that the effort and cost for launching our attack on a particular captcha scheme is low.

5.3 Implementation and Hardware Platforms

Our prototype system[4] is implemented using Python. The preprocessing model is built upon the *Pix2Pix* framework [34], implemented using Tensorflow v.1.12, and the captcha solver is coded

[4] Code and data are available at: https://goo.gl/92VxXC.

Table 2. The Overall Success Rate and Solver Running Time

Scheme	Success rate		Running Time per Captcha (*ms*)
	Base Solver	Fine-tuned Solver	
Sohu	83%	**92%**	43.78
eBay	52%	**86.6%**	4.22
JD	60%	**86%**	43.18
Wikipedia	7%	**78%**	4.71
Microsoft	36.6%	**69.6%**	46.06
Alipay	23%	**61%**	3.75
Qihu 360	48.6%	**56%**	41.03
Sina	40.6%	**52.6%**	42.81
Weibo	4.7%	**44%**	3.41
Baidu	6%	**34%**	41.57
Google	0%	**3%**	4.02

using Keras v.2.1. We use two different hardware platforms. For training, we use a cloud server with a 2.4 GHz Intel Xeon CPU, four NVIDIA Tesla P40 GPUs and 256 GB of RAM, running the Centos 7 operating system with Linux kernel 3.10. The trained models are then run and tested on a desktop PC with a 3.2 GHz Intel Xeon CPU, a NVIDIA Titan GPU and 64 GB of RAM, running the Ubuntu 16.04 operating system with Linux kernel 4.10. All trained models run on the Titan GPU for inference.

6 EXPERIMENTAL RESULTS

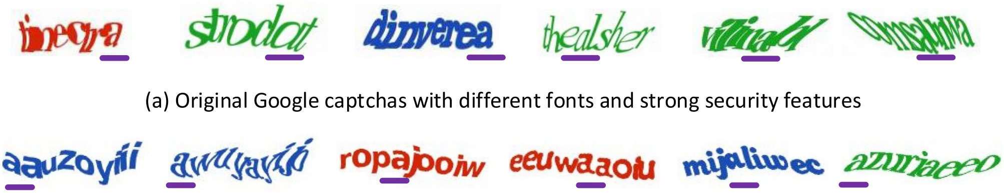
In this section, we first present the overall success rate of our approach for solving 11 current captcha schemes. We then compare our approach against prior attacks on another 22 schemes. Next, we analyze the working mechanism of our approach before discussing the impact of security features on user experience and the generalization ability of our approach.

6.1 Evaluation on Current Captcha Schemes

Table 2 presents the success rate and the average running time in solving a captcha image for 11 current schemes. There is no difference in solving time between the base and the fine-tuned solvers, because they use the same network structure. For each captcha scheme, we report the average running time across 1,000 captchas. We observe little variation in the running time, less than 0.5% across test runs. Note that in this evaluation, all captcha images of a scheme contain the same number of characters. In Section 6.3, we show how our approach can be extended to target a variable number of characters.

6.1.1 Overall Success Rate. Our base solver, built from synthetic data, is able to solve most of the captcha schemes with a success rate of over 20%. This demonstrates the capability of CNN models in performing image recognition. However, it gives a low success rate for some of the schemes such as Weibo (4.7%) and Google (0%). The fine-tuned solver, refined using transfer learning, significantly boosts the performance of the base solver. In particular, it improves the success rate for Wikipedia from 7% to 78%, Weibo from 4.7% to 44%, Alipay from 23% to 61%, and Microsoft from 36.6% to 69.6%. This result shows that transfer learning in combination with captcha synthesis can reduce the data collection efforts for building an effective text captcha solver.

The refined solver also improves the success rate for Google captcha from 0% to 3%. This relatively low success rate is because of the strong security features like distorted, overlapping, waving

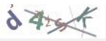
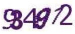
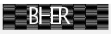
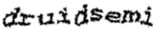
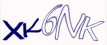
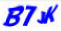
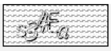
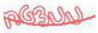
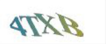
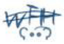
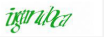


(a) Original Google captchas with different fonts and strong security features

(b) Synthetic Google captchas using our captcha generator

Fig. 8. Examples of real Google captchas (a) and the synthetic versions (b).

Table 3. Example Text-based Captchas that are Incorrectly Labeled by Our Fine-tuned Solver

Scheme	Captcha Image	Ground Truth	Solver Output	Human Attempts
Sohu		d4sk	d4sh	1.6
eBay		934912	994912	1.8
JD		BHER	BFER	1.5
Wikipedia		druidsemi	druidseml	1.5
Microsoft		XK6NK	XK6VK	1.2
Alipay		B7JK	B7YK	1.6
Qihu 360		s34Ea	s3VFa	1.8
Sina		nG3uu	nG3uv	1.4
Weibo		4TXB	4TX8	1.4
Baidu		WFIH	WFEH	1.8
Google		irgandoca	igiruloca	>10

characters and dynamic font styles employed by the scheme. These features make it difficult for our captcha generation model to generate high-quality synthetic data. Figure 8 shows that our synthetic captchas are not sufficiently similar to the real captchas (especially for the font styles). We also observe that some security features like overlapping, rotated, distorted characters and dynamic font styles can provide stronger protection under our attack over features like noisy background and occluding lines. Nevertheless, 3% is still above the 1% threshold for which a captcha is considered to be ineffective [10]. We stress that no prior attack before ours can successfully crack the current Google captcha scheme under this criterion.

6.1.2 Incorrectly Labeled Captchas. Table 3 gives some example captchas that are incorrectly labeled by our fine-tuned solver. For most of these captchas, our solver only incorrectly recognize one character and the mis-identified character is similar to the ground truth. For example, for the eBay captcha shown in Table 3, our solver incorrectly label character “3” to “9” due to character overlapping. For the Google scheme, our solver often fails to label several characters in the middle due to excessive character distoration and overlapping. However, our annotators were also struggling to recognize the characters for those captchas. To quantify the difficulty, we asked ten annotators to label those captchas and count the number of attempts required to succeed. The

Table 4. How Often a Common English Prefix and Suffix Appears at the 5,000 Captcha Images from Google and Wikipedia

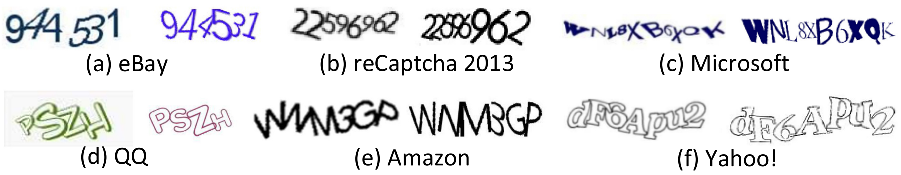
Prefixes	**Number**		**Suffixes**	**Number**	
	Google	**Wikipedia**		**Google**	**Wikipedia**
dis-	76	21	-ing	337	95
pre-	49	10	-est	166	105
mis-	44	9	-ion	129	26
anti-	15	3	-ness	77	6
semi-	7	2	-tion	63	12
fore-	3	2	-less	28	5
inter-	3	1	-ation	21	4
under-	1	0	-ative	8	2
trans-	0	1	-itive	3	0

last column of Table 3 gives the averaged number of attempts required by our annotators to successfully recognize images of a captcha scheme. The results suggest that our annotators found it difficult to recognize most of the captcha schemes in the first attempt. In particular, due to the strong distorted and occulting lines of the Google captcha scheme, more than half of our annotators failed to recognize a Google captcha image within ten attempts.

6.1.3 Exploiting Captcha Patterns to Improve the Success Rate. Some captcha schemes like Google and Wikipedia have more than eight characters in a single captcha image. We call these long-character captcha schemes. We notice that the characters of a long-character captcha image tend to follow some patterns, where some English word prefixes or suffixes appear frequently. We think this might be a feature for helping a human user to better recognize the characters. To verify our hypothesis, we collected and manually labeled 5,000 captcha images in addition to the 1,000 testing captchas used for the Google and the Wikipedia schemes. We then count how often a commonly used English word prefix and suffix appears in the 5,000 captchas for each of the two schemes, by using the list of prefixes and suffixes suggested in [45].

Table 4 lists some of the frequently appeared prefixes and suffixes, containing at least three characters. We see that a three-character prefix or suffix appears at least 9 times (up to 76) in the 5,000 captcha images of a scheme. This is greater than the averaged frequency of 1.99 if those characters are evenly and randomly distributed across the 26 English alphabet letters over the 5,000 captchas of a scheme. We also observe a similar pattern for prefixes or suffixes with four or more characters, although they have less frequency of appearance over the three-character counterparts.

Heuristics. We wondered if one can exploit this observation to improve the success rate of a captcha solver. In other words, can we build a context-sensitive captcha solver to correct some of the characters after performing image recognition? To this end, we develop a heuristic to post-process the characters given by the fine-tuned solver to target the English word prefixes and suffixes listed in Table 4. Specifically, for a solved captcha word, we first identify whether the word contains a candidate pattern. A candidate pattern is a sequence of characters that are similar to a word prefix or suffix, but only with a few characters that are different from a standard word prefix or suffix. For example, "trani" is a candidate pattern for word prefix "trans" as both words are only different in the last character, "i." A solved captcha word can also contain multiple. In this case, we will use the prefixes and suffixes listed in Table 4 to search for the possible candidate patterns. Using this strategy, our heuristic would correct the candidate pattern "seml" to "semi." Doing so gives a correct prediction for the Wikipedia captcha shown in Table 3. Applying this strategy to



(a) eBay (b) reCaptcha 2013 (c) Microsoft

(d) QQ (e) Amazon (f) Yahoo!

Fig. 9. Examples of the captcha schemes (left) tested in prior work, and the synthetic versions (right) generated by our captcha generation model. Our generation model is highly effectively in synthesizing captcha images.

the 1,000 test captchas images for `Google` and `Wikipedia`, we improve the success rate for the `Google` scheme from 3% to 5.1% and the `Wikipedia` scheme from 78% to 79.8%.

6.1.4 Training and Deployment Overhead. It took us around 2 days to train a captcha synthesizer and the preprocessing model together on our training platform, and less than 50 milliseconds to solve a captcha on our evaluation platform using a desktop GPU. For captcha schemes with a confusing background or occluding lines (e.g., Baidu and Sina captchas in Table 2), our solver can take 10× longer than others to solve process a captcha image. This overhead comes from the preprocessing model. As we train a scheme-specific preprocessing model with different network structures, the stronger the security features are, the more complex the preprocessing will be (and hence longer running times). Nonetheless, our approach can solve all the testing schemes under the commonly used criterion [10] with a quick running time.

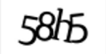
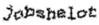
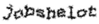
6.2 Comparison to Prior Attacks

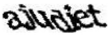
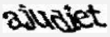
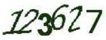
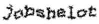
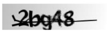
We now compare our approach with four state-of-the-art methods [8, 10, 18, 20] on 24 distinct captcha schemes, including the `eBay` and `Wikipedia` schemes from Table 1 and other 22 schemes. To provide a fair comparison, we try to use captchas that prior methods were tested on. When possible, we use the same dataset or captchas from the original scheme on which the prior work was evaluated. For those obsolete captcha schemes (21 out of 24 schemes), we collected the test data from public datasets, or using captcha generation tools developed by independent researchers. Specifically, we use (1) public datasets of previous captcha schemes, (2) online captcha generators, such as `captchas.net`, which was used by some of the previous captcha schemes, and (3) open source captcha generators used by prior work.

For each captcha scheme, we collected 1,500 samples—from which we use 500 for training and 1,000 for testing. Figure 9 gives some examples of the real captchas and the one produced by our generation model. The figure suggests that our generation model can produce captchas that are visually similar to real examples from the target scheme.

Table 5 compares our fine-tuned solver to previous attacks. Our approach outperforms all comparative schemes by delivering a significantly higher success rate. For many of the testing schemes, our approach boosts the success rate by 40%. It can successfully solve all the captchas of Blizzard, Megaupload and Authorize used in Reference [10]. Our approach achieves a success rate of 87.4% and 90% for `reCAPTCHA` 2011 and 2013, respectively. This scheme was previously deemed to be strong where the human accuracy is 87.4% [20]. That is to say, our solver matches the capability of humans in solving `reCAPTCHA`. To achieve a comparable accuracy for `reCAPTCHA`, a `CNN`-based captcha solver [23] would require 2.3 million unique real captcha images [20], but our approach needs only 500. We note that unlike all the competitive approaches that require manually tuning a character segmentation method, we forgo this process. Thus, our approach requires less expert involvement but gives better performance.

Table 5. Comparing Our Approach Against Four Prior Attacks [8, 10, 18, 20] on 24 Captcha Schemes Where Prior Methods Were Tested

Captcha Scheme	**Captcha Example**	**Success rate**		**Captcha Scheme**	**Captcha Example**	**Success rate**	
		B-11 [10]	*Ours*			*Gao's*	*Ours*
Megaupload		93%	100%	Baidu (2016)		46.6%	97.5%
Blizzard		70%	100%	QQ		56%	94%
Authorize		66%	100%	Taobao		23.4%	90.7%
Captcha.net		73%	99.6%	Sina		9.4%	90%
NIH		72%	99%	reCAPTCHA (2011)		77.2%	87.4%
Reddit		42%	98%	eBay		58.8%	86.6%
Digg		20%	95%	Amazon		25.8%	79%
eBay		43%	86.6%	Wikipedia		23.8%	78%
Slashdot		35%	86.4%	Microsoft		16.2%	72.1%
Wikipedia		25%	78%	Yahoo! (2016)		5.2%	63%

Captcha Scheme	**Captcha Example**	**Success rate**		**Captcha Scheme**	**Captcha Example**	**Success rate**	
		B-14 [11]	*Ours*			*George's*	*Ours*
reCAPTCHA (2013)		22.3%	90%	PayPal		57.1%	92.4%
Baidu (2013)		55.2%	89%	reCAPTCHA (2011)		66.6%	87.4%
reCAPTCHA (2011)		22.7%	87.4%	Yahoo! (2016)		57.4%	63%
eBay		51.4%	86.6%				
Baidu (2011)		38.7%	83.1%				
Wikipedia		28.3%	78%				
Yahoo! (2014)		5.3%	75.1%				
CNN		51.1%	51.6%				

Here *B-11* and *B-14* represent the method of References [10] and [11], respectively.

6.3 Targeting Schemes with a Variable Number of Characters

One potential criticism of our approach described so far is that it only targets captcha schemes with a fixed number of characters. However, our approach can be extended to target schemes with a variable number of characters. One way for doing that is to have a model to predict how many characters a preprocessed image may contain, and then use a captcha solver that is specifically built for that number of characters.

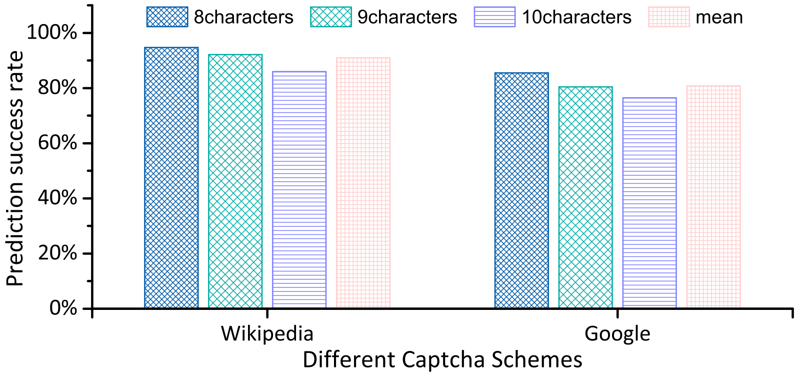


Fig. 10. The success rate of our prediction model when targeting captchas with variable number of characters.

To test this strategy, we use a CNN to build a character number predictor. Our model consists of four convolutional layers, four pooling layers and a fully connected layer, and a max-pooling layer follows each of the convolutional layers. The filter size in each convolutional layer is 5 × 5, and other parameters are the same as our base captcha solver.

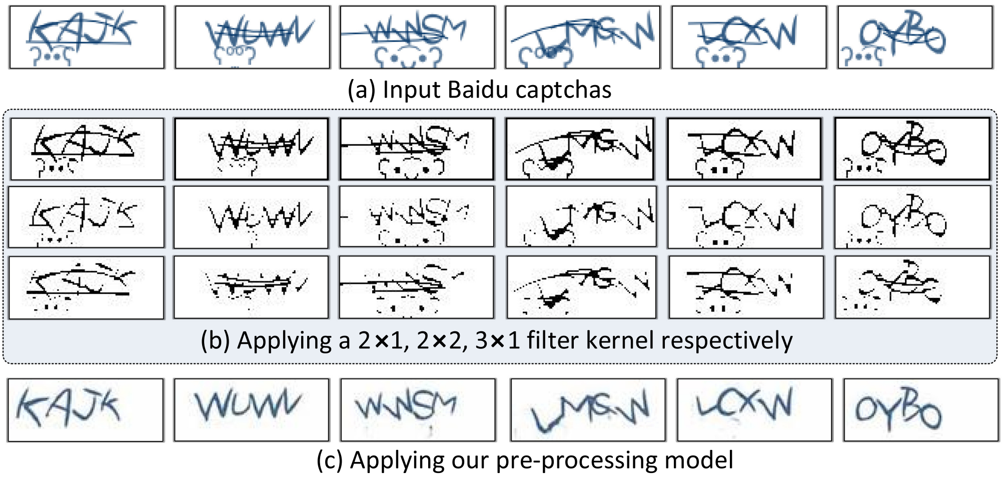
We evaluated our predictor using Google and Wikipedia captchas, both use a variable number (8, 9, or 10) of characters. For each scheme, we use 100,000 synthetic captchas (around 33,333 captchas per character length) for training the predictor and 3,000 (1,000 per character length) real captchas for testing. Figure 10 shows the accuracy for predicting the number of characters in a captcha image. Our predictor gives an accuracy of 90.9% and 80.8% for Wikipedia and Google schemes, respectively.

When combining the predictor with our fine-tuned solver (but not using the context-aware heuristic described in Section 6.1.3), we see a slight drop in the accuracy. This is expected as our character-number predictor is not perfect. The combination gives a success rate of 70.9% and 2% for Wikipedia and Google schemes, respectively. The resulting success rates are still higher than the 1% threshold for which a captcha scheme is seen to be ineffective [10].

6.4 Preprocessing Security Features

Recall that the second step of our attacking pipeline is to remove the security features and standardize the font style of an input captcha. In this experiment, we compare our preprocessing model against prior preprocessing methods on removing noisy backgrounds [8, 10, 36], and standardizing font styles [12, 17] and character gaps [18].

Removing security features. The classical methods used in prior attacks for preprocessing captchas is filtering [8, 10, 36]. The idea is to apply a fix-sized window, or filter kernel, throughout the image to remove the occluding lines and noise while keeping edges of the characters. As can be seen from Figure 11, finding the right filter kernel size is difficult. This is because the filter either fails to eliminate the background and occluding lines or it overdoes it by eroding edges of the characters (Figure 11(b)). While filtering was effective for prior text-based captchas, the latest captcha schemes have introduced more sophisticated security features, which make it no longer feasible. In contrast to filtering, our preprocessing model can successfully eliminate nearly all the background noise and occluding lines from the input image, leading to a much cleaner captcha image while keeping the character edges, as depicted in Figure 11(c).

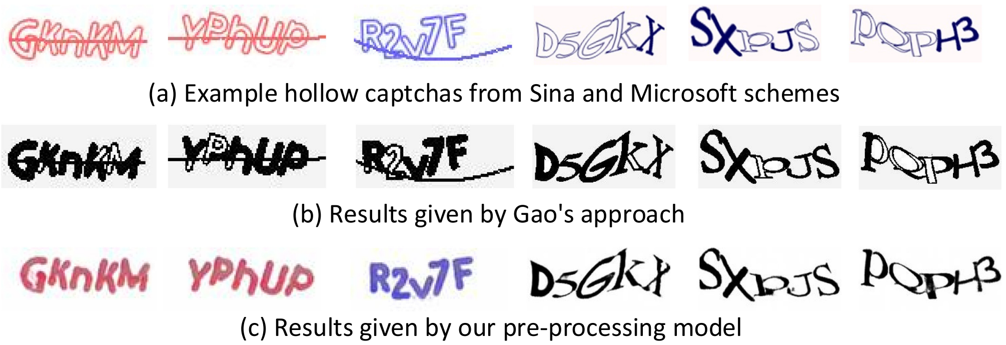


(a) Input Baidu captchas

(b) Applying a 2×1, 2×2, 3×1 filter kernel respectively

(c) Applying our pre-processing model

Fig. 11. For the input images (a), a filter-based method fails to remove security features (b) while our approach can (c).



(a) Example hollow captchas from Sina and Microsoft schemes

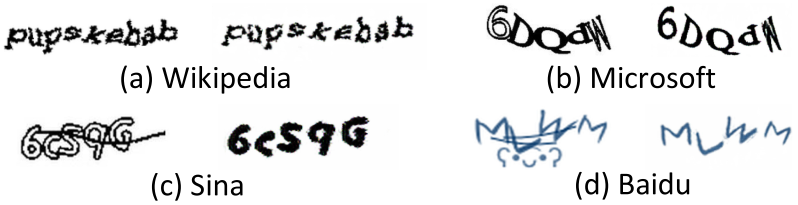
(b) Results given by Gao's approach

(c) Results given by our pre-processing model

Fig. 12. Comparing font style standardization between a state-of-the-art hollow captcha solver [17] and our preprocessing model. Our preprocessing model is able to fill the hollow parts more effectively.

Filling hollow characters. Figure 12 compares our preprocessing model against a state-of-the-art hollow captcha solver [17]. The task in this experiment is to fill the hollow parts of the characters. Here, we apply both schemes to the testing hollow captchas from `Sina` and `Microsoft` schemes. Figure 12(a) gives some of the examples from these two schemes, while Figures 12(b) and 12(c) present the corresponding results given by the hollow filling method in Reference [17] and our approach, respectively. As can be seen from the diagrams, our preprocessing model is able to fill most of the hollow strokes, but the state-of-the-art method leaves some hollow strokes unfilled. Therefore, our approach is more effective in standardizing the font style. We also note that unlike prior attacks, which require manually designing and tuning an individual method to process each security feature, our approach automatically learns how to process all features at one go. Therefore, our approach requires less effort for implementing a holistic preprocessing model.

Standardizing character gaps. Prior work has reported that the robustness of a text captcha scheme largely dependents on the difficulty of character segmentation rather than character recognition [12]. Many modern text captchas are designed to make it harder for a computer program to segment the characters. The examples given in Figure 13 show that our preprocessing model


(a) Wikipedia

(b) Microsoft

(c) Sina

(d) Baidu

Fig. 13. Character segmentation produced by our preprocessing model. For each scheme, the left image is the input captcha, and the right image is the output of our preprocessing model.

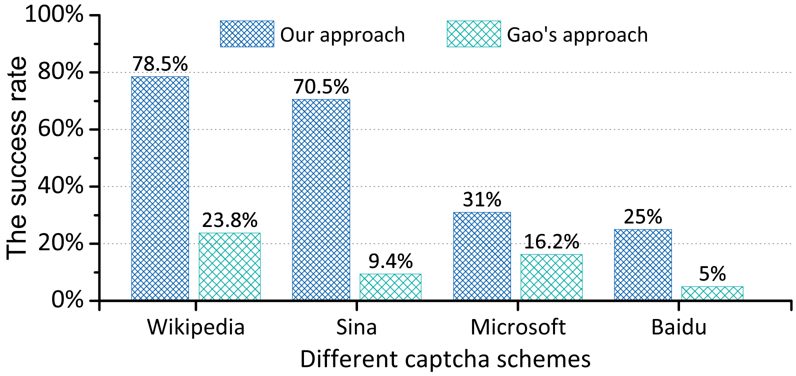


Fig. 14. Using our character segmentation approach can help to improve the success rate of prior work [18].

is effectively in standardizing the gap between characters. To evaluate the effectiveness of our preprocessing model for character segmentation, we use the same network structure to train a model solely for character segmentation. We then use the preprocessing model to replace the native character segmentation model used in Reference [18] but keep the remaining parts unchanged. Figure 14 shows that our preprocess along can help to greatly improve the success rate of a previous solver.

6.5 Synthesizer Training Termination Criteria

Our captcha synthesizer is trained under the GAN framework, and training terminates when the discriminator fails to classify a certain ratio of synthetic captchas (Section 4.2). Figure 15 reports how the termination criterion affects the quality of the synthetic captchas. The x-axis shows the ratio (from 0.8 to 0.97) of synthetic captchas that are misclassified as a real captcha by the discriminator when training terminates. The y-axis shows the success rate achieved by the fine-tuned solver for five current captcha schemes, where the base solver is trained on the resulting synthetic captchas using different termination criteria but the fine-tuned solver is trained on the same set of real captchas.

In general, the more synthetic captchas that the discriminator fails on, the higher the quality the generated synthetic captchas will be, which in turns leads to a more effective captcha solver. However, the increase in the success rate reaches a plateau at 0.95. Further increasing the similarity of the synthetic captchas to real ones does not improve the success rate due to overfitting. Based on this observation, we choose to terminate synthesizer training when the GAN discriminator can successfully distinguish less than 5% (i.e., fail on 95% or more) of the synthetic captchas. We found that this threshold works well for all captcha schemes tested in this work.

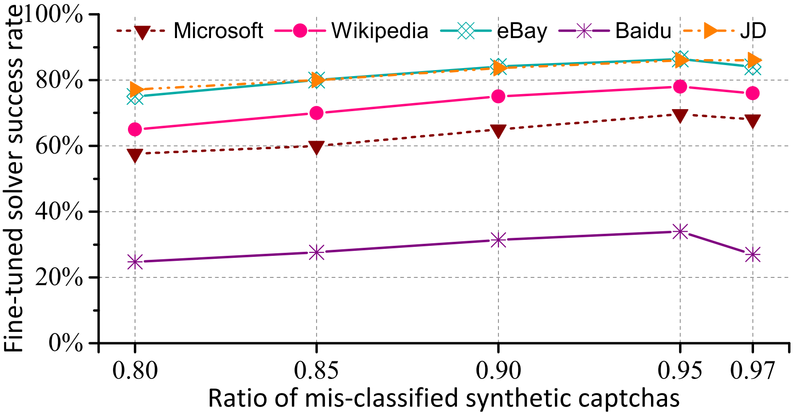


Fig. 15. How the synthesizer training termination criterion affects the solver performance. Training terminates when the discriminator fails to classify a certain ratio of synthetic captchas.

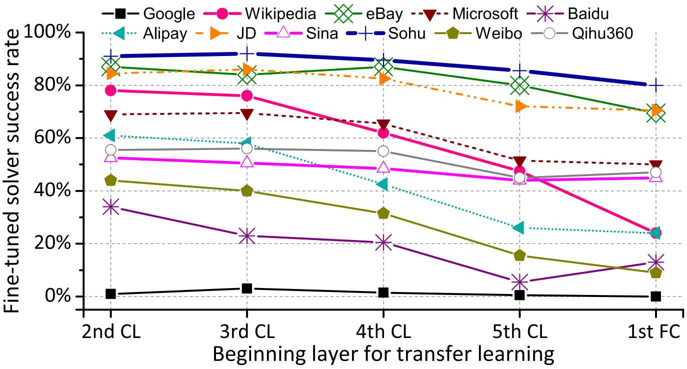


Fig. 16. How the beginning layer for transfer learning affects the resulting performance of the fine-tuned solver.

6.6 Transfer Learning

Recall that we only use 500 real captchas to refine the base solver by employing transfer learning (Section 4.3). Our strategy for transfer learning is to only retrain some of the latter neural network layers of the base solver (see Figure 7). In this experiment, we investigate how the choice of transfer learning layers affects the performance of the fine-tuned solver. To that end, we apply transfer learning to different levels of the base solver, by changing the starting point of transfer learning from the 2nd convolutional layer (CL) all the way down to the first fully connected layer (FC).

6.6.1 Identify the Best Beginning Layers. We apply transfer learning to different levels of the base solver. This is achieved by changing the starting point of transfer learning from the 2nd convolutional layer (CL) all the way down to the first fully connected layer (FC). To determine the best starting layer for transfer learning, we apply cross-validation to the real captcha training dataset. Specially, we divide the 500 real captchas into two parts, the first part of 450 captchas is used to refine the base solver, and the rest 50 captchas are used to validate the refined solver. We vary the beginning layer for transfer learning and then test the refined base solver on the validation set to find out which beginning layer leads to the best performance. Figure 16 reports performance of the resulting fine-tuned solvers trained under different transfer learning configurations for the

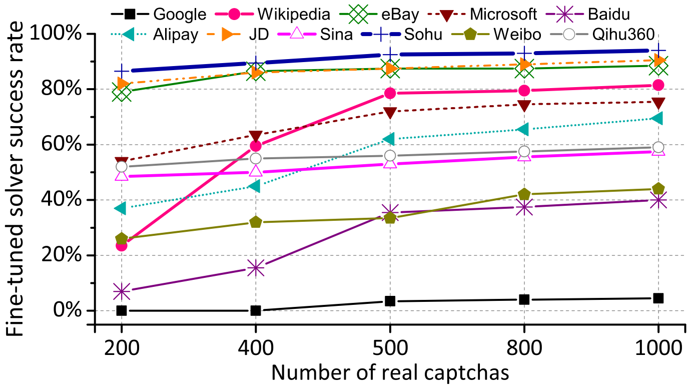


Fig. 17. The achieved success rates when the fine-tuned solver is trained using different number of real captchas.

11 current captcha schemes given in Table 1. Overall, applying transfer learning to the second or third CL onward leads to the best performance. Furthermore, this refining process only takes several minutes as it uses just 500 captchas.

6.6.2 *Finding Suitable Training Data Size.* In this experiment, we evaluate how the number of real captchas used in transfer learning affects the success rate of the fine-tuned solver. Figure 17 shows the success rates of the fine-tuned solver when using different numbers of real captchas in transfer learning. When the number of training examples is 500, our approach reaches a high success rate. For most captcha schemes, the success rate drops significantly when the number of training examples less than 400. Nevertheless, our approach can achieve a high success rate when the number of training examples is 500. Such a number allows an attacker to collect from the target website easily.

6.7 Captcha Usability Study

Our evaluation also includes a user study to quantify the impact of security features on user experience (i.e., captcha usability) and the success rate of our solver. Specifically, we have conducted an online survey by recruiting 20 participants to fill in an anonymous questionnaire. Our participants are at the age group of under 30s and are familiar with text captchas. In the questionnaire, we present 100 synthetic captchas with different security strength. We divide the synthetic captchas into six categories based on the number of characters and the security parameters used for generating the captcha. In the survey, we give each participant one minute to label a captcha and ask each participant to rate the usability of five captchas from each category on a 5-point Likert-scale, where 1 = very poor and 5 = excellent usability.

Table 6 gives the criteria used to determine the captcha difficulties and an example captcha for each category. For each category, we also give the averaged success rates achieved by our participants and our solver, as well as the averaged rating given by the participants.

We see that using more security features increases the difficulty for a computer program to solve a captcha challenge, but it also decreases user experience. This can be illustrated that the averaged human success rate for the captchas in category 6 of Table 6 is below 70%, meaning that nearly one-third of the time a user will enter a wrong answer for captchas in this category. Therefore, captchas in this category were given the lowest usability score of 2.1 is not surprising. We also observe that various security features have a different impact on the effectiveness of our captcha solver. For example, our solver can better handle captchas with noisy backgrounds in categories

Table 6. Example Captchas Used in Our User Study, the Success Rates of Humans and Our Approach, and the Usability Rating

No.	Example	Security Features		Success Rate		Usability
		Anti-segmentation	Anti-recognition	Humans	Ours	
1		English letters, arabic numerals	Rotation, varied font sizes	95.25%	100%	4
2		English letters	Rotation, varied font sizes	90.25%	88%	2.75
3		English letters, complex background	Rotation, distortion	91%	96%	2.8
4		English letters, overlapping characters, complex background	Varied font sizes, rotation, distortion	89.25%	86%	2.7
5		English letters	Varied font sizes, ratation, distortion	79.75%	77%	2.8
6		English letters, overlapping characters	Varied font sizes, rotation, distortion, waving	68.75%	40%	2.1

3 and 4 than that with distorted characters in categories 5 and 6. As a result, although a captcha image with a noisy background may have equally poor usability as another one with distorted or overlapping characters, the two captcha images could have different degrees of robustness under our attack. Moreover, as we expect, the success rate of a computer solver drops as the difficulty of the captcha increases.

We also find that noisy backgrounds have a negative impact on the user experience, because our participants gave an averaged usability score of less than 3 for captchas in categories 3 and 4 of Table 6. However, background confusion has little contribution to the security strength of captchas under our attack. This can be confirmed from the similar, or even better-solving performance given by our solver when compared to human participants for captchas in the two categories. This finding suggests that complex background confusion perhaps should be abandoned in future text captcha schemes. Overall, this user study shows that a GAN-based captcha solver can achieve comparable performance for solving text captchas when compared to humans, but balancing the security and usability of a text captcha scheme is not trivial.

6.8 Generalization Ability

Given the scope of this work, we cannot test our approach on all current captcha schemes. To evaluate our approach's generalization ability for character recognition, we apply it to the MNIST dataset. This dataset contains a large number of handwritten digits of different forms.

We follow the same methodology as we have used throughout the evaluation to build a MNIST solver, i.e., by first building a synthesizer, then a base solver and a fine-tuned solver. We train the synthesizer using up to 500 real MNIST images. Next, we build the base solve using 100,000 synthetic images before fine-tuning the base solver using the same set of real MNIST images. We compare our solver with the recently proposed RCN [20], which was shown to be effective by using a small number of training samples. We test both approaches on images that are not seen in the training phase.

As can be seen from Table 7, our approach gives a marginally lower accuracy when using 20 real MNIST images per digit, but it outperforms the RCN when using 40 or more real images per

Table 7. Success Rate of Comparing to RCN [20] for Classifying the MNIST Dataset

# of per digit	20	40	60	80	100
RCN	96.5%	97.3%	97.6%	97.8%	98%
Our fine-tuned solver	**96.2%**	**98.3%**	**98.9%**	**99.2%**	**99.8%**

digit. In other words, our approach is effective on another image classification dataset, indicating that our approach has a good generalization ability.

Extend to other captcha schemes. We believe our approach is generally applicable and can be naturally extended for video and image captchas by adapting the network architecture to recognize objects from the inputs; and favorably, the process of synthetic data generation, model training and tuning still is unchanged. This flexibility allows one to attack various types of captchas, not just text-based ones. For example, to target NuCAPTCHA [56], a motion-based captcha scheme, we need to replace our CNN solver with a model similar to the Mask R-CNN [29]. The idea is to first segment the video frames into images and then recognize characters from individual images. After replacing the solver structure, we also need to extend our GAN-based captcha synthesizer to generate a sequence of synthetic images (as recognition is performed at the image level). For motion-based captchas, the key is to maintain the temporal relationships among images, for which a temporal CNN can be useful [40].

7 POTENTIAL COUNTERMEASURES

7.1 Security Enhance Through Adversarial Example Generation

Recent works have shown that adversarial examples generated by inserting some perturbations onto a target image can confuse a machine-learned image classifier [65]. Recent work has exploited this observation to improve the security of captcha images [51, 58]. However, the perturbations or noise generated by prior methods [51, 58] are often noticeable by a human eye and as a result, our preprocessing model is effective in removing these perturbations. Hence, a better approach is to make the perturbations imperceptible, so that it has less impact on the user experience while increases the difficulty for training a successful preprocessing model.

However, one of the challenges for generating imperceptible perturbations is that the generation scheme is tightly coupled to both the captcha image and the captcha solver. This raises a practical issue, because the captcha designer often does not have a copy of the solver implementation. To demonstrate this point, we use synthetic Baidu captchas to train five captcha solvers (in addition to our *LeNet-5*-based model), based on five established CNN models: MaxoutNet [26], NetInNet [44], GoogleNet [63], VGG [60], ResNet18 [30]. We then apply each trained solver to a captcha image with different imperceptible perturbations. Table 8 shows the original captcha image and its adversarial versions, and the prediction given by different solvers. To aid readability, we mark the perturbations, which are imperceptible to the participants in our user study, using a black box. As can be seen from the table, a perturbation scheme tuned for a particular network cannot invalid others.

One approach for improving the generalization ability of the perturbation scheme is to find ways to generate perturbations that can invalidate the commonly used image classification models. To this end, we implement a prototyping adversarial generator to target the CNN-based models listed in Table 8.

7.1.1 Countermeasure Prototype. Our prototype has three components: a feature location module, a perturbation generation model, and an adversarial solver, described as follows.

Table 8. Examples of Original Captcha (*No. 1*) and the Corresponding Adversarial Captchas with Different Perturbations (*Nos. 2–6*)

No.	Captcha	Ours	MaxoutNet	NetInNet	GoogleNet	VGG	ResNet18
1		LMGW(√)	LMGW(√)	LMGW(√)	LMGW(√)	LMGW(√)	LMGW(√)
2		NMGW(×)	VMGW(×)	IMGW(×)	IMGW(×)	LMGW(√)	NMGW(×)
3		LWWW(×)	LMMW(×)	LWNW(×)	LWWN(×)	VWNW(×)	LMGW(√)
4		LMGW(√)	LMSW(×)	LWSW(×)	LWSW(×)	LMGW(√)	LWGW(×)
5		LMGW(√)	LNGW(×)	VWGW(×)	LNNW(×)	LWWW(×)	LWGW(×)
6		LWMW(×)	LWGW(×)	LWWW(×)	LWGW(×)	LWWW(×)	LWMW(×)

The feature location module finds which areas of the captcha image are most important for successful recognition of a given captcha image across network architectures. To do so, we first apply sliding windows to divide a captcha image into a number of areas from the direction of top to bottom, left to right. We then add random noise into each area to observe whether the CNN model can misclassify the captcha image and the areas that can confuse the CNN model will be selected as the critical locations. Once these critical areas are located, the perturbation generation module (built upon Reference [43]) will generate the adversarial captcha image by inserting the perturbations into each of these areas. Note that the perturbation generator may produce different perturbations for different areas. We run the perturbed images through a set of pre-trained captcha solvers (built upon different network architectures) to check if the perturbed images can confuse all the solvers. If not, then we ask the perturbation generator to create a new set of perturbations until this success criterion is met or the generation time has exceeded a threshold (set to three seconds in our case). To enhance the transferability of the synthetic adversarial captchas, we are inspired by Xie et al. [69] and apply random multiscale transformations to each critical areas at each iteration. In the latter case, we choose the image that can confuse the largest number of targeting solvers.

7.1.2 Evaluation of Countermeasure. We evaluate our captcha generator using the 1,000 real Sohu captchas that were used in the evaluation reported in Section 6. We choose this scheme, because our solver is highly effective in solving it by giving the highest success rate. The results for using and without using our perturbation scheme are shown in Figure 18. Our perturbation scheme significantly reduces the success rate for solving the Sohu scheme when using a CNN-based solver.

7.1.3 Limitations of Countermeasure. We acknowledge that our countermeasure does not eliminate the vulnerability of text captchas under deep-learning-based attacks, as an attacker can still use a network that is different from the ones targeting by our perturbation generator. However, we find that changing the number of layers or neurons, or the size of the convolutional layers of a solver has little impact on our perturbation scheme. We also find that using a deeper network does not significantly improve the success rate for solving perturbed captcha images, because most of the captcha images are of small sizes and hence a deeper network does not offer additional benefits. Nonetheless, we want to stress that while our countermeasure can help to improve the security strength of current text captcha schemes, they will become inevitably less

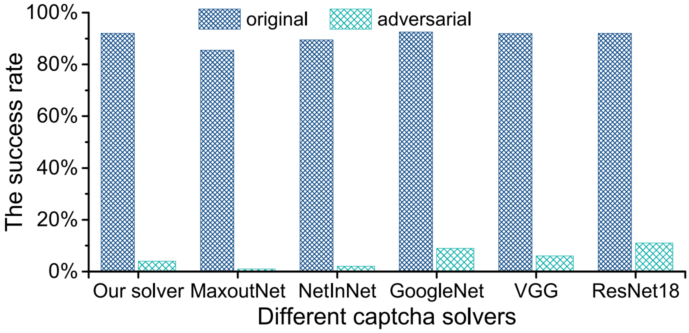


Fig. 18. The success rates when targeting the original Sohu captchas and the adversarial versions generated by our scheme.

secure when more advanced deep neural network architectures are proposed. Therefore, the community should revisit the use of text captchas.

7.2 Other Alternative Countermeasures

Some alternatives have been proposed to replace text captchas. These include video-based captachas like NuCAPTCHA [56] and game-based CAPTCHAs [49]. The former was shown to be vulnerable [7, 71]. The latter seemingly offers some promises, but the recent breakthrough of deep reinforcement learning in game playing may pose a threat to such schemes [48]. To have a robust countermeasure, one probably need to combine multiple mechanisms similar to the multi-factor authentication protocol [37, 57]. Nonetheless, how to balance the security strength and usability of a scheme is still an outstanding problem.

8 RELATED WORK

The work presented by Mori et al. [28] was among the first text captcha solvers. Their approach employs a set of analytical models and heuristics to attack Gimpy and EZ-Gimpy, two early simple text-based captcha schemes. Since then, a large body of work arose for exploring ways to improve the security of text captchas, building upon attacks on existing captcha schemes. Due to these successful attacks, text captchas are going through an iterative development process, which are still preferred by many users, primarily for the familiarity and a sense of security and control [39].

Segmentation-based attacks. This type of attacks first segments characters of a captcha image and then identifies each segmented character using machine-learning algorithms. Yan et al. show a simple character segmentation method [72], which counts the number of pixels of individual characters, can break most of the captchas from Captchaservices.org. Later, they show an improved segmentation method can be used to attack the early captcha schemes used by Yahoo!, Microsoft, and Google [73]. Unlike our approach, all the aforementioned attacks are tightly coupled to the captcha scheme and hard to generalize. This means to target a new scheme, they would require human involvement to revise the existing heuristics and possibly to design new heuristics.

Deep-learning-based attacks. Decaptcha [8] employs machine-learning-based classifiers to develop a generic attack for text-based captchas. It can break 13 captcha schemes but achieves zero success on more difficult schemes including reCAPTCHA and Google scheme. By contrast, our approach not only gives a higher accuracy on the schemes where Decpatcha succeeds but also delivers a success rate of 87.4% on reCAPTCHA for which Decaptcha has a success rate of zero (see Table 5). Recently, George et al. presents Recursive Cortical Network (RCN) for image recognition [20]. The RCN is effective in recognizing individual characters but are less effective for solving text-based captchas when compared to our approach. In particular, on the PayPal dataset, our approach boosts the success rate from 57.1% to 92.4%. Stark et al. [62] show that active learning can be used to reduce the number of captchas required to learn a solver. However, this approach requires having access to a captcha generator of the target scheme, which is often not available to the adversary. However, active learning is complementary to our approach as it allows the learning engine to use a fewer number of training samples to speed up the training process.

Other attacks. The work presented by Gao et al. targets captchas of hollow characters [16]. Their approach first fills the hollow character strokes, and then searches for the possible combinations of adjacent character strokes to recognize individual characters. While effective on hollow characters, this approach is ineffective on captcha images with overlapping and distorted characters. Their more recent work [18] uses the Log-Gabor filter to first extract character components from the captcha image; it then uses the k-Nearest Neighbor algorithm to recognize individual characters using the extracted information. Due to the limitation of the Log-Gabor filter, their method is ineffective for captcha images with noisy backgrounds, e.g. Baidu captcha shown in Figure 1(b).

Alternative captcha schemes. It is worth mentioning that there are also other captcha schemes built around images [3, 14, 27, 50, 54], audio data [6, 55], or recently adversarial captchas [58]. Many of these were proposed to replace text captchas. However, these alternative schemes are less popular than text captchas and were shown to be vulnerable too [9, 19, 46, 49, 61, 66]. In particular, a significant weakness of an image-based scheme is that the number of images used by the scheme is typically limited. As a result, an adversary may exploit side channels to obtain and label a large portion of the images used by a scheme [32].

Adversarial machine learning. As a final remark, we point out that our work builds upon the foundations of adversarial machine learning [24, 33]. This technique is shown to be useful in constructing adversarial applications to bypass malware detection [53, 70], escape from spam mail filtering [5], or confuse machine learning classifiers [25, 47]. However, no work to date has employed the technique to construct a generic solver for text captchas, and our work is the first to do so.

9 CONCLUSION

This article has presented the first GAN-based generic solver for text captcha. Our solver is built by first learning a captcha synthesizer to automatically generate synthetic training examples to build a base solver, and then refining the base solver using transfer learning. This feature allows our approach relies on fewer real captchas to construct the solver, and can target a wide range of schemes. As a result, our approach needs less human involvement compared to prior methods.

Our approach was evaluated on 33 text captcha schemes, including 11 schemes that were being used by 32 of the top 50 popular websites at the time this study was conducted. Experimental results show that our approach outperforms four start-of-the arts by successfully solving more captchas. We show that our approach is robust and generally applicable, which can break many advanced security features used by modern text captchas. Our results suggest that these advanced features only make it difficult for a legitimate user but would fail to stop automated programs. As a countermeasure, we show that by inserting some imperceptible perturbations on a captcha image, one can enhance the security strength of text captchas under deep-learning-based attacks.

REFERENCES

[1] Abdalnaser Algwil, Dan C. Ciresan, Beibei Liu, and Jeff Yan. 2016. A security analysis of automated chinese turing tests. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*. 520–532.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *Proceedings of the International Conference on Machine Learning*. 214–223.

[3] Elias Athanasopoulos and Spiros Antonatos. 2006. Enhanced CAPTCHAs: Using animation to tell humans and computers apart. In *Proceedings of the IFIP International Conference on Communications and Multimedia Security*. 97–108.

[4] Charles Audet and J. E. Dennis Jr. 2006. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. Optimiz.* 17, 1 (2006), 188–217.

[5] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. 2006. Can machine learning be secure? In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*. 16–25.

[6] Jeffrey P. Bigham and Anna C. Cavender. 2009. Evaluating existing audio CAPTCHAs and an interface optimized for non-visual use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1829–1838.

[7] Elie Bursztein. 2012. How we Broke the NuCaptcha Video Scheme and What we Proposed to Fix it. Retrieved from https://elie.net/blog/security/how-we-broke-the-nucaptcha-video-scheme-and-what-we-propose-to-fix-it.

[8] Elie Bursztein, Jonathan Aigrain, Angelika Moscicki, and John C. Mitchell. 2014. The end is nigh: Generic solving of text-based CAPTCHAs. In *Proceedings of the USENIX Workshop on Offensive Technologies (WOOT'14)*.

[9] Elie Bursztein and Steven Bethard. 2009. Decaptcha: Breaking 75% of eBay audio CAPTCHAs. In *Proceedings of the Usenix Conference on Offensive Technologies*. 8–8.

[10] Elie Bursztein, Matthieu Martin, and John Mitchell. 2011. Text-based CAPTCHA strengths and weaknesses. In *Proceedings of the Conference on Computer and Communications Security (CCS'11)*. 125–138.

[11] Elie Bursztein, Angelique Moscicki, Celine Fabry, Steven Bethard, John C. Mitchell, and Jurafsky Dan. 2014. Easy does it: More usable CAPTCHAs. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 2637–2646.

[12] Kumar Chellapilla, Kevin Larson, Patrice Y. Simard, and Mary Czerwinski. 2005. Computers beat humans at single character recognition in reading based human interaction proofs (HIPs). In *Proceedings of the Conference on Email & Anti-Spam*.

[13] Monica Chew and J. Doug Tygar. 2004. Image recognition captchas. In *Proceedings of the International Conference on Information Security*. Springer, 268–279.

[14] Jeremy Elson, John R. Douceur, Jon Howell, and Jared Saul. 2007. Asirra: A CAPTCHA that exploits interest-aligned manual image categorization. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'07)*. 366–374.

[15] Bent Fuglede and Flemming Topsoe. 2004. Jensen-Shannon divergence and Hilbert space embedding. In *Proceedings of the International Symposium on Information Theory (ISIT'04)*. IEEE, 31.

[16] Haichang Gao, Mengyun Tang, Yi Liu, Ping Zhang, and Xiyang Liu. 2017. Research on the security of Microsoft's two-layer captcha. *IEEE Trans. Info. Forensics Secur.* 12, 7 (2017), 1671–1685.

[17] Haichang Gao, Wang Wei, Xuqin Wang, Xiyang Liu, and Jeff Yan. 2013. The robustness of hollow CAPTCHAs. In *Proceedings of the ACM Sigsac Conference on Computer & Communications Security*. 1075–1086.

[18] Haichang Gao, Jeff Yan, Fang Cao, Zhengya Zhang, Lei Lei, Mengyun Tang, Ping Zhang, Xin Zhou, Xuqin Wang, and Jiawei Li. 2016. A simple generic attack on text captchas. In *Proceedings of the Network and Distributed Systems Symposium (NDSS'16)*.

[19] Song Gao. 2014. *An Evolutionary Study of Dynamic Cognitive Game CAPTCHAs: Automated Attacks and Defenses*. Dissertations Theses Gradworks. University of Alabama, Birminghan.

[20] Dileep George, Wolfgang Lehrach, Ken Kansky, Miguel Lázaro-Gredilla, Christopher Laan, Bhaskara Marthi, Xinghua Lou, Zhaoshi Meng, Yi Liu, Huayan Wang, et al. 2017. A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs. *Science* 358, 6368 (2017), eaag2612.

[21] C. Gold, A. Holub, and P. Sollich. 2005. Bayesian approach to feature selection and parameter tuning for support vector machine classifiers. *Neural Netw.* 18, 5 (2005), 693–701.

[22] Philippe Golle. 2008. Machine learning attacks against the Asirra CAPTCHA. *Comput. Commun. Secur.* 2008 (2008), 535–542.

[23] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. 2014. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR'14)*.

[24] Ian J. Goodfellow, Jean Pougetabadie, Mehdi Mirza, Bing Xu, David Wardefarley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks. *Adv. Neural Info. Process. Syst.* 3 (2014), 2672–2680.

[25] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Machine Learning (ICML'15)*. 1–10.

[26] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Maxout networks. *arXiv preprint arXiv:1302.4389.*

[27] Rich Gossweiler, Maryam Kamvar, and Shumeet Baluja. 2009. What's up CAPTCHA?: A CAPTCHA based on image orientation. In *Proceedings of the International Conference on World Wide Web (WWW'09)*. 841–850.

[28] Mori Greg and Jitendra Malik. 2003. Recognizing objects in adversarial cultter: Breaking a visual CAPTCHA. In *Proceedings of the IEEE Computer Society Conferene on Computer Vision and Pattern Recognition.*

[29] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'17)*. 2980–2988.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 770–778.

[31] Robert Hecht-Nielsen. 1989. *Theory of the Backpropagation Neural Network.* Harcourt Brace & Co., 593–605.

[32] Carlos Javier Hernandezcastro, Arturo Ribagorda, and Yago Saez. 2009. Side-channel attack on labeling CAPTCHAs. *Comput. Sci. ArXiv Preprint ArXiv:0908.1185.*

[33] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I. P. Rubinstein, and J. D. Tygar. 2011. Adversarial machine learning. *IEEE Internet Comput.* 15, 5 (2011), 4–6.

[34] Phillip Isola. 2017. Pix2Pix: Image-to-Image Translation with COnditional Adversarial Networks. Retrieved from https://github.com/phillipi/pix2pix.

[35] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. 2017. Image-to-Image translation with conditional adversarial networks [C]. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1125–1134.

[36] J. Wilkins. 2010. Strong captcha guidelines v1. 2 [J]. 10 (2010), 8.

[37] Zhiping Jiang, Jizhong Zhao, Xiang-Yang Li, Jinsong Han, and Wei Xi. 2013. Rejecting the attack: Source authentication for wi-fi management frames using csi information. In *Proceedings of the IEEE INFOCOM.* 2544–2552.

[38] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization [C]. In *Proceedings of the International Conference on Learning Representations.*

[39] Kat Krol, Simon Parkin, and M. Angela Sasse. 2016. Better the devil you know: A user study of two CAPTCHAs and a possible replacement technology. In *Proceedings of the NDSS Workshop on Usable Security.*

[40] Colin Lea, Rene Vidal, Austin Reiter, and Gregory D. Hager. 2016. Temporal convolutional networks: A unified approach to action segmentation. In *Proceedings of the European Conference on Computer Vision.* 47–54.

[41] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[42] Jiwei Li, Will Monroe, Tianlin Shi, Sebastien Jean, Alan Ritter, and Dan Jurafsky. 2017. Adversarial learning for neural dialogue generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. 2157–2169.

[43] Bin Liang, Hongcheng Li, Miaoqiang Su, Xirong Li, Wenchang Shi, and XiaoFeng Wang. 2019. Detecting adversarial image examples in deep neural networks with adaptive noise reduction. *IEEE Trans. Depend. Secure Comput.* (2019), 1–1.

[44] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network in network. *arXiv preprint arXiv:1312.4400.*

[45] Elaine K. McEwan. 2008. Root Words, Roots and Affixes. Retrieved from http://www.readingrockets.org/article/root-words-roots-and-affixes.

[46] Hendrik Meutzner and Dorothea Kolossa. 2014. Reducing the cost of breaking audio CAPTCHAs by active and semi-supervised learning. In *Proceedings of the International Conference on Machine Learning and Applications.* 67–73.

[47] Takeru Miyato, Shinichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. 2015. Distributional smoothing by virtual adversarial examples. In *Proceedings of the International Conference on Learning Representations (Poster).*

[48] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *ArXiv Preprint ArXiv:1312.5602.*

[49] Manar Mohamed, Niharika Sachdeva, Michael Georgescu, Song Gao, Nitesh Saxena, Chengcui Zhang, Ponnurangam Kumaraguru, Paul C. Van Oorschot, and Wei Bang Chen. 2014. A three-way investigation of a game-CAPTCHA:automated attacks, relay attacks and usability. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security.* 195–206.

[50] Manar Mohameda, Song Gaob, Niharika Sachdevac, Nitesh Saxena, Chengcui Zhangd, Ponnurangam Kumaraguruc, and Paul C. Van Oorschote. 2017. On the security and usability of dynamic cognitive game CAPTCHAs. *J. Comput. Secur.* 25, 3 (2017), 205–230.

[51] Margarita Osadchy, Julio Hernandez-Castro, Stuart Gibson, Orr Dunkelman, and Daniel Pérez-Cabo. 2017. No bot expects the DeepCAPTCHA! introducing immutable adversarial examples, with applications to CAPTCHA generation. *IEEE Transactions on Information Forensics and Security* 12, 11 (2017), 2640–2653.

[52] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* 22, 10 (2010), 1345–1359.

[53] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. 2017. Generic black-box end-to-end attack against RNNs and other API calls based malware classifiers. *arXiv preprint arXiv:1707.05970.*

[54] Neil J. Rubenking. 2013. Are You a Human. Retrieved from https://www.areyouahuman.com.

[55] Andy Schlaikjer. 2007. A dual-use speech CAPTCHA: Aiding visually impaired web users while providing transcriptions of audio streams. LTI-CMU Technical Report, 07-014.

[56] NuData Security. 2010. NuCaptcha. Retrieved from www.nucaptcha.com.

[57] Muhammad Shahzad, Alex X. Liu, and Arjmand Samuel. 2017. Behavior based human authentication on touch screen devices using gestures and signatures. *IEEE Trans. Mobile Comput.* 16, 10 (2017), 2726–2741.

[58] Chenghui Shi, Xiaogang Xu, Shouling Ji, Kai Bu, Jianhai Chen, Raheem A. Beyah, and Ting Wang. 2019. Adversarial CAPTCHAs. Retrieved from http://arxiv.org/abs/1901.01107

[59] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. 2017. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*.

[60] Karen Simonyan and Andrew Zisserman. 2014. In *Proceedings of the International Conference on Learning Representations*.

[61] Suphannee Sivakorn, Iasonas Polakis, and Angelos D. Keromytis. 2016. I am robot: (Deep) learning to break semantic image CAPTCHAs. In *Proceedings of the IEEE European Symposium on Security and Privacy*. 388–403.

[62] Fabian Stark, Caner Hazirbas, Rudoplh Triebel, and Daniel Cremers. 2015. CAPTCHA recognition with active deep learning. In *Proceedings of the German Conference on Pattern Recognition Workshop*.

[63] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[64] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 2818–2826.

[65] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[66] Jennifer Tam, Jiri Simsa, Sean Hyde, and Luis Von Ahn. 2008. Breaking audio CAPTCHAs. In *Proceedings of the Conference on Neural Information Processing Systems*. 1625–1632.

[67] Luis Von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. 2003. *CAPTCHA: Using Hard AI Problems for Security*. Springer, Berlin, 294–311.

[68] Luis Von Ahn, Manuel Blum, and John Langford. 2004. Telling humans and computers apart automatically. *Commun. ACM* 47, 2 (2004), 56–60.

[69] Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, and Alan L. Yuille. 2019. Improving transferability of adversarial examples with input diversity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'19)*.

[70] Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically evading classifiers: A case study on PDF malware classifiers. In *Proceedings of the Network and Distributed System Security Symposium*.

[71] Yi Xu, Gerardo Reynaga, Sonia Chiasson, Jan-Michael Frahm, Fabian Monrose, and Paul C. Van Oorschot. 2014. Security analysis and related usability of motion-based captchas: Decoding codewords in motion. *IEEE Trans. Depend. Secure Comput.* 11, 5 (2014), 480–493.

[72] Jeff Yan and Ahmad Salah El Ahmad. 2007. Breaking visual CAPTCHAs with Naive pattern recognition algorithms. In *Proceedings of the Computer Security Applications Conference (ACSAC'07). Twenty-Third Annual*. 279–291.

[73] Jeff Yan and Ahmad Salah El Ahmad. 2008. A low-cost attack on a Microsoft captcha. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'08), Alexandria, Virginia, Usa, October*. 543–554.

[74] Guixin Ye, Zhanyong Tang, Dingyi Fang, Zhanxing Zhu, Yansong Feng, Pengfei Xu, Xiaojiang Chen, and Zheng Wang. 2018. Yet another text captcha solver: A generative adversarial network based approach. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 332–348.

[75] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*. MIT Press, 3320–3328.

[76] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2016. SeqGAN: Sequence generative adversarial nets with policy gradient. In *Proceedings of the Thirty-First Association for the Advancement of Artificial Intelligence*. 2852–2858.

[77] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*.

Received June 2019; revised October 2019; accepted January 2020