Tick Tock Break The Clock: Breaking CAPTCHAs on the darkweb

David Holm Audran, Marcus Braunschweig Andersen, Mark Højer Hansen, Mikkel Møller Andersen, Thomas B. Thomas, Kasper H. Hansen, Dimitrios Georgoulias and Emmanouil Vasilomanolakis
Aalborg University, Copenhagen, Denmark
{*daudra21, mban21, mhha21, mman21, tfrede21, khha18*}*@student.aau.dk*, {*dge, emv*}*@es.aau.dk*

Keywords: darkweb, CAPTCHA, web crawler, machine learning, darkweb marketplace, darkweb forum

Abstract: Nowadays, almost all major websites employ CAPTCHAs. This prevents website scraping, fake account creation as well as DDoS or bruteforce attacks. For anonymity reasons, mainstream CAPTCHAs such as Google's reCAPTCHA cannot be used on the darkweb. Due to the evolution of machine learning and computer vision, the CAPTCHA challenges used there, such as the clock CAPTCHA, are usually more arduous than those found on the clearweb. This paper presents an automated system that uses machine learning to break clock CAPTCHA challenges with a high success rate. We evaluate our system in a real world setting against 725 clock challenges from live darkweb marketplaces. Our results show an accuracy of 96.83% while maintaining low time requirements while analyzing, predicting and submitting the CAPTCHA solution.

1 INTRODUCTION

CAPTCHAs are widely used around the web to prevent an assortment of attacks such as DDoS, web crawling or creation of fake accounts. Since their inception in 1996 (Guerar et al., 2021) a long range of technologies have been developed. To attempt breaking these, a great deal of computer vision technologies have been utilized as the majority of CAPTCHAs are image-based. Accompanying this, machine learning can now be used as it greatly increases the computers' chance to successfully solve such a puzzle.

The advancements of the aforementioned technologies, and especially of machine learning, have deprecated many CAPTCHA models. Despite the issues this may present, it also creates an invitation for creating new, more arduous models, that can distinguish a human from a machine. There is an ongoing arms race, where one side is trying to create resilient CAPTCHAs, while the other side is trying to break them. This is also the case on the darkweb.

The CAPTCHAs found on the darkweb differ from those on the clearweb since the desire to remain anonymous restricts which models can be used. Newer CAPTCHAs, such as reCAPTCHA, require a connection to Google services to check information about the client, and is therefore not a viable choice for darkweb sites. This has in turn increased the effort put into making different types of CAPTCHAs to be utilized on the darkweb (Georgoulias et al., 2021).

Using machine learning to develop a model that can successfully solve CAPTCHA challenges requires a labelled data set containing examples of the CAPTCHA and corresponding answers. Such supervised machine learning approaches cannot be trained without proper labelled data. To fulfill this requirement the CAPTCHA could be downloaded and the answer manually filled in, however this could take a lot of time depending on the amount of data needed. Therefore having access to the source code of the CAPTCHA can automate its generation. However, in most cases, especially on the darkweb, the CAPTCHA code generation is proprietary and has to be reverse engineered. Furthermore when utilizing machine learning models, the model produced can be limited to solve only one type of CAPTCHA. A small modification to the CAPTCHA algorithm might render the machine learning model highly ineffective.

A predominant darkweb CAPTCHA scheme is the so-called clock CAPTCHA (Georgoulias et al., 2021) (see Figure 1a). The challenge is to correctly submit the time of an analogue clock that contains several misleading geometric shapes in under 60 seconds. The clock comes in different variations, however the general idea is the same. To this day, no method of breaking the clock scheme has been developed and disclosed. The main goal of this work is to break the basic version of the clock CAPTCHA

scheme along with one variation, utilizing machine learning. To achieve this goal we use the deep learning architecture ResNet50 to create a model which is able to predict the time of the clock CAPTCHA given an image of it. In addition, a web scraper is built providing the ability to automatically solve a challenge using the trained model in real time on a Tor hidden service. We limit ourselves and do not further expand our work on additional clock variations, since it would prove to be a never ending task, due to the highly dynamic nature of the darkweb.

One major challenge when attempting to automate data collection from platforms on the darkweb, is bypassing the Distributing Denial of Service (DDoS) protection mechanisms. Darkweb marketplaces and vendor shops utilize CAPTCHAs with the goal of taking away the automation capabilities of web crawlers (Soska and Christin, 2015). In essence, these mechanisms are put in place to force individuals into manually providing responses to challenges. This can be a time-consuming task, especially when attempting to deploy crawlers in multiple platforms simultaneously, making the entire process of data collection challenging. Successfully bypassing these mechanisms in an automated manner, provides ease and speed to the process, making research efforts more effective. Hence, we note that our work is only intended for assisting researchers and the developed system is available to researchers upon request, due to ethical considerations (see Section 2).

In this paper we show that: i) It is possible to develop a machine learning model to correctly solve the clock CAPTCHA with 96.83% accuracy; ii) the developed model can be utilized by a web scraper to access services using the clock CAPTCHA, in a timely manner; iii) the developed model can easily be adapted to incorporate modified versions of the clock while maintaining high accuracy.

2 ETHICAL ISSUES

In this section we want to address the ethical issues associated with this paper. Our work is not intended to be used in takedown attempts against the platforms implementing the showcased CAPTCHAs, since literature characterizes them as a non-violent alternative to street drug trafficking (Martin and Christin, 2016). Instead, our goal is to illustrate that solving the clock CAPTCHA can be automated, and then utilized by a web crawler to further automate data harvesting for research purposes.

With regard to the site access, the marketplaces that were part of our study are both publicly available and free to access. Furthermore, we want to point out that we did not in any way hinder the operation of any of these platforms, or the experience of their users. In order to complete our experiments, we only used site mechanisms that are available to all users, and in a manner that did not consume any additional resources from the marketplaces' side. Lastly, our research did not involve any kind of user private data, hence there is no risk of exposing the identity or private information of any individuals.

3 BACKGROUND AND RELATED WORK

Even though CAPTCHAs on the clearweb and the darkweb have inherent differences, the methods for breaking CAPTCHAs are the same in both domains. Such methods, can be categorized in the following three categories: machine learning methods, non machine learning methods and hybrid methods.

3.1 Machine Learning Methods

The development of deep learning has resulted in great advances in CAPTCHA breaking. Challenges previously deemed impossible for computers to solve (e.g. advanced image recognition) are now solvable.

(Noury and Rezaei, 2020) showed a method for breaking text-based CAPTCHAs of a fixed length using convolutional neural networks, achieving up to 98.94% accuracy. Similarly, a deep learning method for breaking text-based CAPTCHAs are described by (Tang et al., 2018), where convolutional layers are combined with max-pooling layers.

When it comes to breaking image-based CAPTCHAs, sophisticated deep learning methods are necessary. Mittal et al. (Mittal et al., 2018) describe a method for breaking a CAPTCHA using the Inception V3 image recognition model, achieving a mean accuracy of 91% in real time.

Another example of using deep learning to break an image based CAPTCHA, is by (Hossen and Hei, 2021) using the neural-network ResNet18 architecture. The authors utilize a pre-trained instance of the model, that is trained using the ImageNet[1] data set. This minimizes the required training needed for the specific challenge. The focus of Hossen and Hei was to provide a low-cost method for breaking the CAPTCHA system, and they only required 143 minutes of training. They achieved an accuracy of 88% on the test set. Additionally, they achieved an accuracy

[1] https://www.image-net.org/

of 95.93%, when providing the model with real-world examples of challenges.

3.2 Non Machine Learning Methods

While it is not widespread to apply non machine learning methods for breaking CAPTCHAs, there have been attempts using optical character recognition (OCR). (Csuka and Gaastra, 2018) propose a method using the OCR engine Tesseract (Smith, 2007) for breaking text-based CAPTCHAs on the darkweb. They describe the performance of this method as inferior to the applied machine learning method in terms of success rate, but it does operate faster and provides immediacy compared to the machine learning method.

According to (Weng et al., 2019) another non machine learning method for malicious activity, is using underground CAPTCHA solving services. These services consists of large amount of human labor solving CAPTCHAs in exchange for money.

3.3 Hybrid Methods

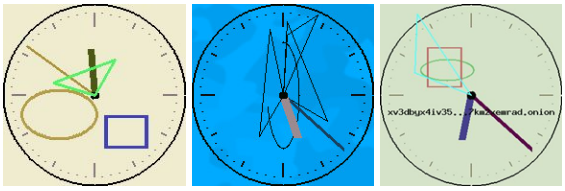
Any information the computer is able to retrieve or extract on the CAPTCHA challenge at hand improves the accuracy of the computers decision. A method used by (Sivakorn et al., 2016) aimed at breaking Google's widely used reCAPTCHA (Shet, 2014) utilised both deep learning methods and Google's own reverse image search engine. The challenge presented by reCAPTCHA is an image-based CAPTCHA, as described in Section 3.4 of this paper. The solution to breaking this, presented by (Sivakorn et al., 2016), consists of modules, that each assigns labels to an image. Most of the modules are deep learning based, but one of the modules is the Google reverse image search engine. The tags and labels provided by each module are then compared, and a decision for each image in the challenge is made.

3.4 Darkweb CAPTCHAs

CAPTCHA schemes like Google's reCAPTCHA cannot be used on the darkweb, due to anonymity issues. For this reason, more traditional CAPTCHA scheme types are used. The most prominent ones being image-based CAPTCHAs and text-based CAPTCHAs (Georgoulias et al., 2021).

Text-based CAPTCHAs present an image of a string of random letters, with the goal being for the user to identify each letter. The images are often obscured, colored or blurred, to make it harder for machines to recognize the letters, but still remaining fairly easy for a human.

Image-based CAPTCHAs work by presenting a question and a set of images to the user, and then requiring the user to pick an image/images thereof, that correspond to the question asked (Alqahtani and Alsulaiman, 2020). Alternatively, the user might be presented with a single image and are then required to answer the question by describing the image or its contents usually by picking from a set of options. The images shown are usually in poor quality and/or have shapes, lines or gibberish text that are easy for a person to distinguish from but hard for a machine. Image-based CAPTCHAs are considered to be the most advanced and secure type of CAPTCHA, as it is based on image details, which makes it hard for a machine to solve (Brodić et al., 2016).



(a) *Dread* forum clock (b) *Cartel* marketplace clock (c) *Cocorico* marketplace clock

Figure 1: Three variations of the clock CAPTCHA in the darkweb

The CAPTCHAs found on the darkweb, although using these traditional ideas, have been improved upon, making them more innovative than the ones found on the clearweb. One of the most predominant schemes can be found on Figure 1, which illustrates different adaptations of the clock CAPTCHA. To solve the scheme, the correct time must be selected within a certain time frame. The challenge for machines being to distinguish the patterns and shapes from the clock's pointers. This type of image-based CAPTCHA is widely adopted on darkweb marketplaces, with some variations depending on the hidden service. Two well-known hidden services the *Dread forum* and the *White House Market* have provided a public GitHub repository with code for what they call "EndGame V2 - Onion Service DDOS Prevention Front System" [2]. This system provides several services with one of them being the clock CAPTCHA scheme as seen on Figure 1a. Due to this variation being publicly available, it is one of the more commonly used. Other variations can be seen on Figures 1b and 1c. The clock found on the *Cartel* marketplace differentiates by having a different background with patterns as well as placing shapes around the center.

[2]https://github.com/onionltd/EndGame

The *Cocorico* marketplace, places the url of the site horizontally across the middle of the clock, which is a technique often used on darkweb CAPTCHAs.

4 THREAT MODEL

In this section we discuss the threat model followed in this paper in terms of the capabilities of the attacker and the required accuracy of an attack to be considered practically successful.

4.1 Attacker capabilities

We assume that the attacker is able to produce labelled data for the supervised machine learning algorithm. This assumption comes with the limitation that without labelled data the whole process would require significant manual work.

Moreover, we assume that the attacker is able to both contact the (Tor) hidden service of the target website and is also able to download an image of the CAPTCHA clock. Furthermore, on the one hand the attacker requires high computational power (especially RAM) to be able to train the residual neural network that we will be utilizing in this paper. On the other hand, upon training the model, the attacker is able to run the prediction on a typical computer with no significant computational capabilities. In that sense, and excluding the training phase, our threat model follows the work of (Bock et al., 2017).

4.2 Attack accuracy

The definition of when a CAPTCHA scheme is considered broken is not simple. The debate for this is very opinionated and no definitive accuracy threshold has been agreed upon (Bursztein and Bethard, 2009).

When designing a CAPTCHA scheme the original design goal states that "*automatic scripts should not be more successful than 1 in 10000 attempts*", which equates to an accuracy of 0.01% (Chellapilla et al., 2005). This is widely regarded as too ambitious, as random guesses would be able to reach an accuracy higher than this. Instead, many regard 1% accuracy to be the threshold, as the accuracy of random guesses would be within the acceptable margin, and therefore not able to deem the scheme broken (Bursztein et al., 2011). Others argue that 5%, or even higher percentages, are more reasonable (Baecher et al., 2010).

For attackers aiming at breaking CAPTCHAs, the accuracy goal is usually a lot higher. (Hossen and Hei, 2021) present an accuracy goal of above 50%, aiming at developing a low-cost attack against the hCaptcha system. (Aboufadel et al., 2005) state that a CAPTCHA is considered broken if a computer algorithm can solve the scheme 4 out of 5 times on average, implying an accuracy goal of above 80%.

In reality, the viable accuracy is dependent on the amount of resources the attacker possesses and the cost of the attack (Bock et al., 2017). An attacker with many resources, that would be able to attack the CAPTCHA scheme tens or hundreds of thousands times, might only need an accuracy of 1% for the attack to be worthwhile. Similarly, an attacker with limited resources might need an accuracy of above 50% to even consider the attack. Furthermore, many darkweb sites implement a lockout function, that blacklists the user if the CAPTCHA scheme has been failed three times. This obstacle demands a certain level of accuracy of the attack, for it to be viable to use in automation, i.e., with a web scraper.

Based on the aforementioned previous work and the fact that in most darkweb marketplaces and forums one can try to solve a CAPTCHA at least twice before being blacklisted we expect an accuracy higher than 80% to be more than satisfactory. For a 80% model success rate and the user having 2 attempts to successfully solve the CAPTCHA, the probability P of a crawler providing the correct solution at least once is calculated at 96% :

$$
\begin{aligned}
&P(SucceedAtLeastOnce) = 1 - P(FailBothAttempts) \\
&= 1 - 1^{\text{st}}Fail * 2^{\text{nd}}Fail \\
&= 1 - 20\% * 20\% \\
&= 96\%
\end{aligned}
$$

5 SYSTEM OVERVIEW

Our automated CAPTCHA breaking system solves the darkweb clock scheme using machine learning. The system can be reduced into three main steps: i) model setup, ii) model training and iii) model usage. 2. The first two steps take place on an AI cloud, where

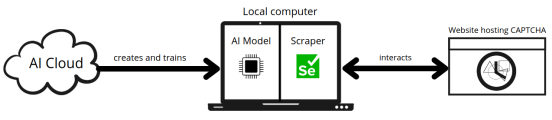


Figure 2: Architectural overview of the entire CAPTCHA breaking system

the model is set up and trained with the generated pictures. The AI Cloud is a separate system, which is utilized due to the computational resources it provides. The model is then downloaded and saved to a local computer. On the local computer the model is used by the scraper, which connects to a Tor site, that uses

the clock CAPTCHA. The scraper downloads the image from the CAPTCHA and predicts the answer to it, using the model from the AI Cloud. The architectural overview of the system is visualized in Figure The system depends on two different programming environments. The image generation is C++ code, based on the Lua code found in the EndGame repository, and the machine learning training and web scraper is created in Python. The aforementioned individual steps are elaborated in the following sections.

6 RESNET50 VS THE DARKWEB CLOCK CAPTCHA

In this section we go over the details that surround setting up, training, and using the model.

6.1 Setting up the model

We divide the model assembly into 3 distinct procedures: the data generation, preprocessing, and the use of ResNet50.

6.1.1 Data generation

In order to obtain labeled data, the code for the clock CAPTCHA was extracted from the EndGame code base mentioned in Section 3.4 and rewritten into a C++ program with additional functionality. The program is able to generate PNG clock images along with a text file containing the time shown on each of the generated clocks. The program has the ability to generate both the clock from the code base, and the clock found on the Cartel marketplace, as seen on Figure 1b. The program takes two arguments, where the first argument is the number of iterations. One iteration will generate 720 images for each of the two clock types. Hereof the number 720 stems from the fact that an analogue clock is divided into 12 hours and 60 minutes, amounting to $12 * 60 = 720$ different clock hand positions. The program will generate an image for each of these possibilities. The second argument should be either 0, 1 or 2, and will determine if the program should generate the Dread forum clock (0), the Cartel marketplace clock (1), or both (2).

6.1.2 Preprocessing

In order for the model to be able to train on the generated data, it is first necessary to preprocess the images. First, the images are loaded along with the corresponding labels. We decided to retain as much information as possible from the images, and therefore kept all 3 RGB channels in the picture instead of converting them to e.g., greyscale. The size of the image is then re-scaled to ensure that it is 190x190 pixels, as that is the size used in most cases in the darkweb. The images are then normalized, changing the pixel intensity range values from $0-255$ to $0-1$. This makes it easier for the model to converge, and limits the amount of zero-gradients during training.

6.1.3 Using ResNet50

To solve the clock CAPTCHA we use a residual neural network. The challenge lies withing accurately determining the time on the clock, which has 720 possible solutions. Therefore, the type of deep learning problem is a multi-class classification problem with 720 classes, with one class per possible time on the clock. Provided a CAPTCHA challenge, the classifier generates 720 probabilities, each giving how likely the corresponding class is for the provided challenge. With this list of probabilities, it is then possible to extract the highest one to find the classifiers best guess for a solution to the challenge.

The architecture we have chosen to use is the ResNet50 architecture. This architecture consists of an initial convolutional layer followed by a max pooling layer, 48 convolutional layers divided into residual building blocks with 3 layers in each, aw well as an average pooling layer. Moreover, we added a dropout layer with a rate of 0.7, that randomly sets inputs to 0, with a frequency of the rate at each step[3]. This aids in avoiding over-fitting the model, essentially dropping some information randomly during training. In addition, a final dense layer is added, mapping the output of the final layer to the number of possible classes, in our case 720, using softmax activation.

The entire implementation of our deep learning model is implemented in Python using the Keras API [4]. Keras provides both an untrained and trained version of the ResNet50 architecture. We utilized the untrained architecture and performed the necessary alterations of it to suit our challenge.

6.2 Training the model

Training of a deep residual network is a complex and resource heavy task, which can take a long time. The model used in this paper builds upon the ResNet architecture, and is trained on an AI Cloud with optimized hardware for this specific task. The node which

[3] https://keras.io/api/layers/regularization_layers/dropout/

[4] https://keras.io/

training has been performed on contains 96 'Intel(R) Xeon(R) Platinum 8168 CPU @2.70GHZ' CPUs, has 128GB RAM allocated and utilizes two 'Tesla V100-SXM3-32GB' GPUs to parallelize the training. Being able to utilize hardware this powerful cut the otherwise long training phase very short.

While training the deep learning model, we found that a batch size of 64 gave the best results. In order to optimally utilize the GPUs available to us for training, the batch size is scaled up according to the number of GPUs, giving each GPU the intended batch size to work with. The same is done for the learning rate.

The dataset used consists of $72,000$ samples, and is split into 80% for training and 20% for testing. The training set is split up further, using 20% as the validation set. The reasoning behind this, is with access to the publicly available source code used for generating the CAPTCHA challenge we are attacking, we solved the challenge of data collection by being able to generate our own labelled data automatically. This allowed for the generation of perfectly balanced datasets of any size, that are identical to the data the model would be faced with in the evaluation and testing phase. Furthermore the whole dataset is shuffled before being split up into training and test sets, to ensure that every class is represented in each of the sets.

The metrics the model is judged by is the *loss* and the *accuracy*. The *loss* is sparse categorical cross entropy loss, a function used to calculate the loss of the predictions made by the model in its current state, compared to the true labels. The loss is used to indicate to the model, how well it predicted in the current iteration of training. The categorical cross entropy for n number of predictions, is defined as:

$$Loss = -\sum_{i=1}^{n} y_i \cdot \log \hat{y}_i \tag{1}$$

where y_i is the actual label, and $\hat{y}_i$ is the prediction made by the model. The *accuracy* is standard classification accuracy, i.e., the number of accurate predictions divided by the total number of predictions. The reasoning behind using standard classification accuracy is that the dataset which the model builds upon is equally distributed among all possible classifications.

Three different models have been trained on the AI Cloud. Initially our focus was the model with the most generic type of the clock, as seen on Figure 1a. The first edition of the model was only trained on the Endgame variation of the clock. The results from the Dread forum clock showed a high accuracy and low loss (see Table 1). However, when tested against the clock variations illustrated on Figures 1b and 1c, the model was unsuccessful. At that point it was obvious that the slightest changes in the clock resulted in the performance of the model deteriorating drastically. To combat this, but to also test the adaptability of the model training approach to modified versions of the clock, clocks like the one found on the Cartel marketplace had to be generated. After analyzing the properties of the specific clock variation, and modifying the clock generation code, we were able to successfully generate these as well. Hence, we decided to build a combined model for two clock types which is trained on a equally distributed amount of both clocks at a 1:1 ratio, $36,000$ samples of each. The training was set to 200 epochs, with early stopping if the validation loss reaches a lower value than 0.05, using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.0001. Training on the generic clock alone lasted for 708 seconds, reaching 11 epochs. Training on the Cartel marketplace clock lasted for 301 seconds, reaching 4 epochs. Training on the combined model lasted for 759 seconds, reaching 12 epochs. The results of the training can be seen in Table 1.

Model	Accuracy	Loss
Dread Forum Clock	0.992	0.029
Cartel Marketplace Clock	0.996	0.025
Combined	0.988	0.048

Table 1: Overview of the performance of the different models on the test set.

6.3 Using the model

To use the model a web scraper was developed capable of navigating to a given list of URLs either via direct input or via reading the URLs from a text file. The scraper loads the machine learning model and utilizes Selenium [5] to open a web browser and navigate to the site. The browser chosen in this case was Google Chrome which does not support Tor natively however does support utilizing a proxy to connect with. A Tor proxy was therefore set up on the default 9050 port. After connecting the webdriver which selenium operates by, the browser navigates to the site requested. As most of these sites contain a queuing system to avoid DDoS attacks, it waits until the clock CAPTCHA appears on screen before continuing. Once this happens the scraper finds the clock image, downloads it, resizes it to 190x190 pixels, passes it into the model and awaits its response. Finally, the scraper passes the response to the site and clicks the submit button. If the prediction is accurate, the CAPTCHA is bypassed successfully. If the model produces an incorrect result, the site generates a new clock challenge and the procedure is repeated. Should this result also be incorrect, the scraper exits

[5] https://www.selenium.dev/

the site, since after the third incorrect submission the Tor identity of the scraper will be banned from the site. The scraper automatically detects whether or not it was successful by checking if the CAPTCHA exists after clicking the submit button. If not, it assumes it has successfully bypassed the CAPTCHA mechanism and navigated to the home page of the site.

The trained model, along with its weights is loaded by the scraper using the Keras API. A function in the Python script used to make predictions, is then able to make a prediction for a single image, and return the numerical label as a tuple, in an hour/minute format. It then utilizes a dictionary loaded from a JSON file, containing the mapping of numerical labels to actual labels, to convert the values.

To test the model we utilize the scraper on 9 popular darkweb websites, that either contain the Cartel marketplace or the Dread Forum CAPTCHA clock variation. Each website is visited 20 times, however visits that experience connection errors are being excluded from the final data set. The time measurements are taken purely on the runtime of each metric, while ignoring any time used on waiting in the DDoS queue, connecting to the site, etc.

7 RESULTS AND DISCUSSION

To perform the evaluation of our deep learning model we will be using the SKLearn Metrics module [6], which provides a function to write out a classification report. It provides the metrics precision, recall and F1-score for each class in the data set, and an average for each metric across all classes. For each class, a true positive is correctly labelling the image as the given class, and a false positive is labelling the image as the given class even though it is not. A true negative is correctly not labelling the image as the given class, and a false negative is incorrectly not labelling the image as the given class.

	Precision	Recall	F1-score	Support
0:0	1.00	1.00	1.00	20
0:1	1.00	1.00	1.00	20
0:2	1.00	1.00	1.00	20
...				
...				
11:58	1.00	0.90	0.95	20
11:59	1.00	1.00	1.00	20
Accuracy			0.99	14400
Macro avg	0.99	0.99	0.99	14400
Weighted avg	0.99	0.99	0.99	14400

Table 2: ResNet50 model performance test set

The evaluation of our model will be performed on a new labelled test data set consisting of $14,400$ images, with a combination of two different variations of the clock CAPTCHA. This test data set was balanced with 10 instances of each possible class for both variations of the clocks. The evaluation was performed on a standard computer with an Intel i5-4210U (1.70 GHz) CPU and 8GB of RAM, and the Ubuntu 20.04.3 LTS operating system.

The time required for this computer to load the model, load all of the $14,400$ challenges and provide a solution was 41 minutes and 44 seconds. This is an average of 0.17 seconds for each prediction on a standard computer. As shown in Table 2, our model achieves an accuracy of 99% on the test data set, and an average precision, recall and F1-score of 99% across all classes.

7.1 Clocks in the wild

The scraper was programmed with the functionality to run in both a sequential and a parallel mode. The parallel mode utilizes a thread-pool allowing the scraper to run on several sites at once. Nevertheless, due to the fact that threads share resources, the time to solve a CAPTCHA on a site is increased quite drastically.

7.1.1 Sequential Mode

In the sequential mode, the scraper performed very well in both finding the CAPTCHA and inserting the result as indicated in Figure 3. The scraper utilized the *combined* model as described in Section 6.2 as the initial *Dread forum* model was unable to bypass the clock found on the Cartel marketplace (see Figure 1b). As seen in Figure 3, the time spent by the scraper to find and download the image is about 0.05 seconds on average. The prediction itself averages at 0.12 seconds, and inserting the answer that the model predicted takes 0.3 seconds, all in all resulting in a scraper which can solve a clock CAPTCHA scheme on a site in approximately 0.5 seconds on average.

7.1.2 Parallel Mode

In parallel mode, discovering the CAPTCHA image and acquiring it took an average of 0.6 seconds, while the prediction from the model averaged at 2.6 seconds. Lastly, the result submission was executed in approximately 3.6 seconds, contributing towards an average total of 6.9 seconds, for the entire process (see Figure 4). The parallel mode does also require more computational resources, since Selenium opens a new browser instance for each URL to scrape, while

[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

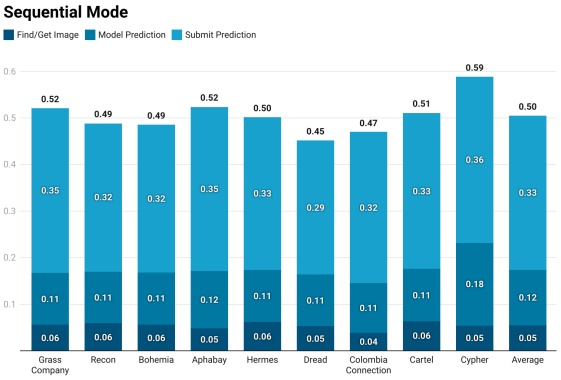


Figure 3: Average runtime of the scraper in seconds, per site and in sequential mode. The presented data were calculated only from successful connections.

the sequential mode opens a new tab in the same browser instance. Hence, choosing the optimal mode depends on the use case (e.g. commencing crawling on one platform at a time, or several at the same time).

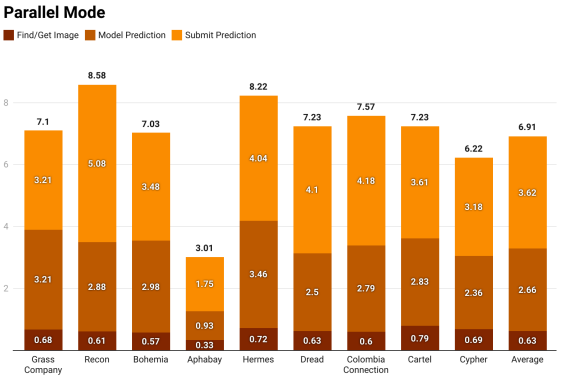


Figure 4: Average runtime of the scraper in seconds, per site and in parallel mode. The presented data were calculated only from successful connections.

7.1.3 Mode Comparison

Running the scraper in sequential mode on all 9 marketplaces is calculated at a total average of 4.5 seconds, for all of the CAPTCHA challenges to be successfully solved. In parallel mode, this number goes up to 6.9 seconds and is equal to the average presented on Figure 4, since in the specific mode this number is already calculated with all of the CAPTCHAs being solved simultaneously. We consider these two averages to be important since they provide an estimation of the time a user would need to run the scraper one platform at a time or on several platforms concurrently, without the identity of the platforms being a factor. Furthermore, the disparity between the two results, is attributed to the aforementioned need for additional computational resources that the scraper requires when operating in parallel mode.

7.1.4 Scraper Evaluation

We tested our system by performing a total of 702 marketplace visits, both in parallel and sequential mode. As shown on Table 3, the scraper had to perform an extra attempt to solve the challenge in 23 occasions. This translates into the scraper being able to solve the CAPTCHAs from the 679 remaining marketplace visits, on the first try. The resulting number of challenges solved amounts to a total of 725, with an overall accuracy of 96.83% (702 out of 725). Lastly, in all of the 702 visits, regardless of whether it took one or two attempts, the scraper managed to provide automated access to the platform via bypassing the CAPTCHA mechanism in 100% of the cases.

The results described above were achieved on a computer with an Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz and 8GB of RAM running the Parrot Linux operating system version 5.0.

CAPTCHAs	Retries	Accuracy	Site Visits	Overall success
725	23	96.83%	702	702 / 702 (100%)

Table 3: Performance of the scraper

8 CONCLUSION

In this work, we present a high performance attack on the clock CAPTCHA found on multiple darkweb marketplaces and forums, utilizing a deep residual machine learning model, trained with a self generated dataset, on a high performance AI Cloud. The result is a model achieving an F1-score of 0.99 on 14,400 separately generated clock instances. Combining this model with a web scraper, we successfully tested our system against 725 CAPTCHA challenges, which belong to two different variations of the darkweb clock CAPTCHA, with a 96.83% accuracy.

One limitation of this paper, is that our model is over-fitted, fitting too closely to the training set and thus does not perform well on unseen data. The cause of this issue stems from the fact that the dataset is uniform in terms of the clock image itself. The model places great importance on the features of these specific clocks, hence modifying the target CAPTCHAs results in a weaker performance of the model. However, we do illustrate that adapting the training data to different variations of the clock, which we can easily generate by modifying our data generation program,

is an effective solution to the over-fitting problem. This gives our automated CAPTCHA solving system great adaptability for future changes.

Another limitation is that the training of a model requires a lot of memory. Training on $72,000$ images requires somewhere between $64 - 128$GB of RAM, which is not available on a standard computer. This requirement of RAM stems from the individual image file size including all RGB channels and the sheer amount of images we used to train the model.

Lastly, with the goal of further improving our current system, we also experimented with the Resnet18 architecture. We trained a new model using the exact same parameters as we did with the Resnet50 architecture and evaluated it with the SKLearn Metrics module. Our preliminary results suggest that the model is able to achieve an accuracy of 100%, with an average precision, recall and F1-score of 100% across all classes, showing a lot of promise for future implementations. Since Resnet18 is a significantly lighter architecture than Resnet50, we will focus on this model in our future work.

REFERENCES

Aboufadel, E., Olsen, J., and Windle, J. (2005). Breaking the holiday inn priority club captcha. *The College Mathematics Journal*, 36(2):101–108.

Alqahtani, F. H. and Alsulaiman, F. A. (2020). Is image-based captcha secure against attacks based on machine learning? an experimental study. *Computers & Security*, 88:101635.

Baecher, P., Fischlin, M. G. L., Langenberg, R., Lützow, M., and Schröder, D. (2010). Captchas: The good, the bad, and the ugly. *Sicherheit 2010. Sicherheit, Schutz und Zuverlässigkeit*.

Bock, K., Patel, D., Hughey, G., and Levin, D. (2017). uncaptcha: a low-resource defeat of recaptcha's audio challenge. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*.

Brodić, D., Petrovska, S., Jevtić, M., and Milivojević, Z. N. (2016). The influence of the captcha types to its solving times. In *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1274–1277.

Bursztein, E. and Bethard, S. (2009). Decaptcha: breaking 75% of ebay audio captchas. In *Proceedings of the 3rd USENIX conference on Offensive technologies*, volume 1, page 8. USENIX Association.

Bursztein, E., Martin, M., and Mitchell, J. (2011). Text-based captcha strengths and weaknesses. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 125–138.

Chellapilla, K., Larson, K., Simard, P. Y., and Czerwinski, M. (2005). Building segmentation based human-friendly human interaction proofs (hips). In *International Workshop on Human Interactive Proofs*, pages 1–26. Springer.

Csuka, K. and Gaastra, D. (2018). Breaking captchas on the dark web.

Georgoulias, D., Pedersen, J. M., Falch, M., and Vasilomanolakis, E. (2021). A qualitative mapping of dark-web marketplaces. In *Symposium on Electronic Crime Research (eCrime)*. IEEE.

Guerar, M., Verderame, L., Migliardi, M., Palmieri, F., and Merlo, A. (2021). Gotta captcha 'em all: A survey of twenty years of the human-or-computer dilemma. 2021-10-06.

Hossen, M. I. and Hei, X. (2021). A low-cost attack against the hcaptcha system.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Martin, J. and Christin, N. (2016). Ethics in cryptomarket research. *International Journal of Drug Policy*, 35:84–91.

Mittal, S., Kaushik, P., Hashmi, S., and Kumar, K. (2018). Robust real time breaking of image captchas using inception v3 model. In *2018 Eleventh International Conference on Contemporary Computing (IC3)*, pages 1–5.

Noury, Z. and Rezaei, M. (2020). Deep-captcha: a deep learning based CAPTCHA solver for vulnerability assessment. *CoRR*, abs/2006.08296.

Shet, V. (2014). Are you a robot? introducing "no captcha recaptcha". https://security.googleblog.com/2014/12/are-you-robot-introducing-no-captcha.html.

Sivakorn, S., Polakis, J., and Keromytis, A. D. (2016). I'm not a human : Breaking the google recaptcha.

Smith, R. (2007). An overview of the tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 629–633.

Soska, K. and Christin, N. (2015). Measuring the longitudinal evolution of the online anonymous marketplace ecosystem. In *24th USENIX security symposium (USENIX security 15)*, pages 33–48.

Tang, M., Gao, H., Zhang, Y., Liu, Y., Zhang, P., and Wang, P. (2018). Research on deep learning techniques in breaking text-based captchas and designing image-based captcha. *IEEE Transactions on Information Forensics and Security*, 13(10):2522–2537.

Weng, H., Zhao, B., Ji, S., Chen, J., Wang, T., He, Q., and Beyah, R. (2019). Towards understanding the security of modern image captchas and underground captcha-solving services. *Big Data Mining and Analytics*, 2(2):118–144.