Using Machine Learning To Solve Text-based CAPTCHAs

Turhan Kimbrough
Department of Computer Science
Towson University
Towson, Maryland
tkimbr1@students.towson.edu

***Abstract*—CAPTCHA is an acronym for the "Completely Automated Public Turing test to tell Computers and Humans Apart". It is a mechanism which is used to distinguish real human users from bots. CAPTCHAs come in a variety of forms, including the deciphering of obfuscated text, transcribing of audio messages, tracking mouse movement, and more. This research will focus on automating the process of deciphering text-based CAPTCHAs using machine learning techniques. Specifically, supervised learning and convolutional neural networks are used to develop a model which is capable of over 99% accuracy for certain datasets. The goal of this research is to demonstrate the weaknesses associated with text-based CAPTCHA mechanisms, especially with the prevalence of machine learning tools.**

***Keywords*—machine learning, neural networks, supervised training, CAPTCHA**

I. Introduction

CAPTCHA is an acronym for the "**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part", which is a challenge-response test used in computing services to verify that the user is a human. The premise of a CAPTCHA is to provide a test which is relatively easy for a human to solve, but difficult for bots. This is one of many mechanisms used to combat against the growing usage of malicious software automation. Due to the ubiquity of automation software, cybercriminals have been able to easily create bots to perform malicious acts. These acts include denial-of-service attacks, autonomous social media communication agents, scalping scarce merchandise, and more.

Due to the wide availability of CAPTCHA-generating software, they have become a popular mechanism to integrate into websites. In particular, text-based CAPTCHAs are often available as a low-cost and simple solution. Text-based CAPTCHAs typically consist of alphanumeric characters in an image, which has been manipulated to prevent it from being easily parsed by a machine. Users are then challenged to decipher the text in the CAPTCHA, and if the answer is correct, they can continue using the service. CAPTCHAs are typically available from content management systems (such as WordPress) or can be integrated into a website via API.

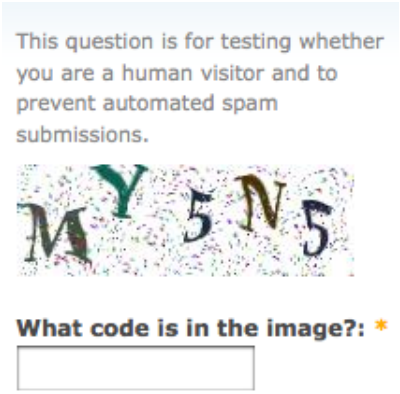

Figure 1. Example of a text-based CAPTCHA.

While this mechanism can mitigate the majority of software bots, it is not effective against bots utilizing machine learning technology. This research paper demonstrates that a machine learning agent is capable of solving CAPTCHAs with the use of open-source tools, supervised training, and convolutional neural networks. The goal of this research is show how adversaries can use readily available technologies to exploit text-based CAPTCHA mechanisms. This paper will cover background/related work, the methodology used for solving CAPTCHAs, challenges which were present, key contributions, results, and a section covering future work.

II. Background/Related Work

In this section, there will be a brief review of similar work which has been done on using machine learning to solve CAPTCHA tests.

A. Solving reCAPTCHAs With Reinforcement Learning

Researchers at [] have demonstrated the ability to solve mouse-based reCAPTCHAs using reinforcement learning. Google's reCAPTCHA mechanism is more difficult to solve compared to traditional CAPTCHAs due to its usage of mouse-tracking to determine if the user is a human. While the exact algorithm is unknown due to the closed-source nature of the reCAPTCHA technology, the researchers uses a black-box approach to solve reCAPTCHAs.

The approach models mouse movements as transitions on a 2-dimensional grid of pixels. The *Markov Decision Process* is used to generate a series of movements (up, down, left, right), which is combination of random and controlled outcomes. The mouse-control agent is then trained through reinforcement learning to generate a series of movements which mimic the behavior of humans. This methodology was able to achieve a success rate of 97.4% on a 100x100 grid and 96.7% on a 1000x1000 display.

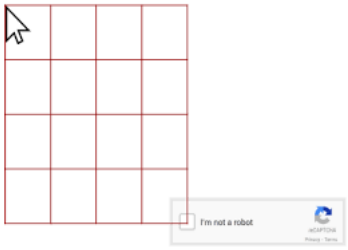

Figure 2. Grid world model.

B. Generic Solving of Text-based CAPTCHAs

Researchers at [] have provided a basic framework on solving CAPTCHAs using segmentation and character recognition techniques. More specifically, their technique is able to detect individual characters in CAPTCHAs with occluding lines. Traditional approaches to CAPTCHA-solving typically use two separate algorithms for segmentation and character recognition. The researchers have instead, developed an algorithm which combines the steps of segmentation and character recognition.

The researchers approached this solution by studying previous schemes which were used to analyze characters in CAPTCHAs. Many of them were unable to segment CAPTCHAs which used *negative kerning*, a technique where negative space is used between characters to ensure occlusion by their neighbors. The algorithm developed by the researchers use machine learning to perform 3 steps: *cut-point detection*, *slicing*, and *scoring*. Cut-point detection analyzes all combinations of character partitioning. The slicer will then segment each character using every partitioning combination, placing them on a graph afterwards. Finally, the scorer assigns a weight for each partitioning combination and determines which characters are most likely present in the CAPTCHA.

Figure 3. Example of negative kerning.

C. Immutable Adversarial Examples for CAPTCHA Generation

While the two works above demonstrate the exploitation of CAPTCHAs, the work presented here will demonstrate an approach for securing CAPTCHAs. Researchers at [] have demonstrated the ability to produce CAPTCHAs which are resistant to noise-removal attempts. Often, CAPTCHA solving techniques using neural networks apply filters to a CAPTCHA image to reduce noise and exaggerate feature sets. Generating CAPTCHAs with immutable noise would pose a significant challenge to many of the machine learning assisted CAPTCHA solvers.

The researchers were able to create immutable adversarial CAPTCHAs by using a modification of the *fast gradient sign* (FGS) method. The FGS method works by slightly altering the gradient values of certain image pixels to maximize *loss* during the training process of a neural network. The modified FGS method takes target label and confidence level as inputs in addition to the original image. This allows for noise generation to work towards a specific goal, distributing the noise in such a way that is difficult to filter.

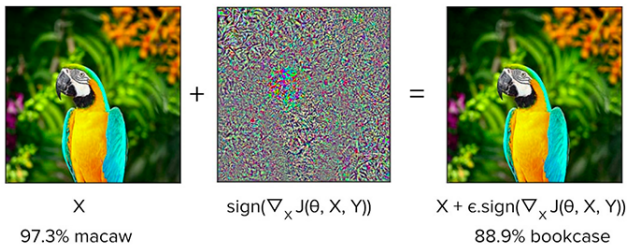

Figure 4. FGS method fooling a machine learning model.

III. Methodology

In this section, a proof-of-concept CAPTCHA-solving model is constructed using open-source tools and machine learning principles. The first two subsections will give background information on the tools/principles being applied. The rest of the subsections will walk through the procedure for creating the machine learning model.

A. Open-source Tool Selection

Python 3 will be the programming language of choice, due to its easy-to-use syntax, portability, and wide range of modules. To complement Python 3, the *Python Image Library (PIL)* will be used to generate CAPTCHA images, and *TensorFlow* will be the core library for building the machine learning model. Lastly, the code will be written for the *Jupyter Notebook* environment, a popular open-source web application for creating/sharing documents in the scientific community.

B. Applied Machine Learning Principles

Two core machine learning principles will be applied when creating the CAPTCHA-solving model, *convolutional neural networks (CNN's)* and *supervised learning*. These two principles are commonly used for image-processing, a perfect use-case for CAPTCHA-solving.

CNN's are a subset of *artificial neural networks (ANN's)*, a family of algorithms which mimic the structure of biological neural networks found in animal brains. ANN's consist of an input layer for data, one or more hidden layers for data-processing, and an output layer for decision-making. CNN's use a combination of *convolutional layers* and *pooling layers* to represent the hidden layers in its structure. Convolutional layers are used to create feature maps, a technique used to extract characteristics from image data. A pooling layer is placed after each convolutional layer to reduce the size of each feature map, lowering the computation power required for further processing.

Supervised learning is a technique to train a machine learning model with the use of labelled data. The labels in the dataset define a category or feature for each data instance.

C. Creating the Training Data

Creating the training dataset consists of two parts; generating a large series of CAPTCHA images and creating labels for each CAPTCHA image. In order to satisfy these two requirements, PIL will be used to generate CAPTCHA images with their labels.

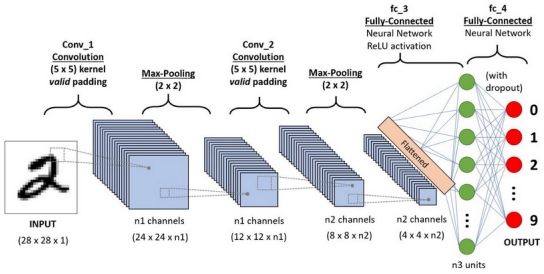

Figure 5. FGS Example structure of convolutional neural network.

In a script using PIL, a total of 10,000 images will be generated to cover all possible variations of 4-digit numbers. Each image will use a dimension of 100x100 pixels and use a consistent font *(DejaVu Sans)*. To obfuscate the text, each image will contain random colored dots and lines. A different color will also be used to print the text in each image. Afterwards, the image will be saved as a PNG image file, with the 4-digit string included in the file name. All CAPTCHA images will be saved in the same directory for processing later on.

Algorithm 1 Generating labelled CAPTCHAs

$count \leftarrow 0$
while $count < 10,000$ **do**
  $number \leftarrow$ GetFourDigitString($count$)
  $font \leftarrow$ GetSystemFont()
  $captcha \leftarrow$ CreateImageWithText($font, number$)
  Resize($captcha, 100, 100$)
  ColorText($captcha$)
  DrawColoredLines($captcha$)
  DrawColoredDots($captcha$)
  $captcha\_name \leftarrow number +$ "$\_image.png$"
  Save($captcha, captcha\_name$)
  $count \leftarrow count + 1$
end

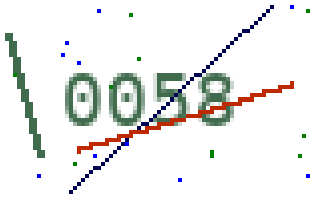

Figure 6. 0058_image.png

The rest of the model-building procedure will take

place in the *Jupyter Notebook* file. *TensorFlow* will need to be imported along with a helper library, *pandas*. The pandas library contains many data structures which are commonly used with TensorFlow. Since the CAPTCHA images generated in the PIL script will need to organized for TensorFlow to work with, the pandas *DataFrame* structure will be used. The DataFrame is a tabular data structure with rows denoting iterations and columns representing data attributes.

All CAPTCHA images will be imported into the Jupyter Notebook file by obtaining the file path to the directory that they were stored. A function will then be called repeatedly to parse each file path and obtain the 4-digit CAPTCHA string associated with each CAPTCHA image. The pandas DataFrame will then be used to store a pair of values for each row, the CAPTCHA image path and the 4-digit CAPTCHA string. This associates the label with its respective data.

Algorithm 2 Get label for CAPTCHA image

Input : String $file\_path$
Output: String $label$
try:
  $file\_name \leftarrow$ SPLITFILENAME($file\_path$)
  SPLITEXTENSION($file\_name$,"$.png$")
  $label \leftarrow$ SPLIT($file\_name$,"_")
  return $label$
catch *InvalidPathException*:
  return $UNSUCCESSFUL$
end

	label	file
0	5458	/home/t-visor/repositories/captcha-tensorflow-...
1	8946	/home/t-visor/repositories/captcha-tensorflow-...
2	4378	/home/t-visor/repositories/captcha-tensorflow-...
3	8860	/home/t-visor/repositories/captcha-tensorflow-...
4	8852	/home/t-visor/repositories/captcha-tensorflow-...

Figure 7. Portion of resulting DataFrame.

D. Defining the Neural Network Structure

In addition to TensorFlow, the library *Keras* will be used to define the layers of the neural network. The purpose of using this neural network is to extract the features (the sequence of digits) which are present in each CAPTCHA image.

The first layer of the neural network is the *input layer*, which will be the entry point for image data. Here, the input layer will specify each CAPTCHA image to have a height and width of 100 pixels, and detect 3 color channels of (red, green, and blue). This will reject anomalies presented to the machine learning model and enforce uniformity.

The *hidden layers* are responsible for extracting and interpreting the features in the image data. Through experimentation, 3 convolutional layers accompanied by 3 pooling layers have been sufficient for feature extraction. This set produces the optimal number of parameters (or filters) which are required to extract the digits from each CAPTCHA image. Afterwards, a flattening layer will transform the multi-dimensional array representing each image into a single-dimension array for easier processing. 2 dense layers are used afterwards for learning the features which have been extracted from prior layers.

The final layer of the neural network is the *output layer*, which is responsible for predicting which numbers appeared in a CAPTCHA image. In particular, a reshaping layer was used to format the prediction as a probability of numbers appearing for each digit position in the CAPTCHA.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100, 100, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 98, 98, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 21, 21, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              6554624
_________________________________________________________________
dense_1 (Dense)              (None, 40)                41000
_________________________________________________________________
reshape (Reshape)            (None, 4, 10)             0
=================================================================
Total params: 6,651,944
Trainable params: 6,651,944
Non-trainable params: 0
```

Figure 8. Neural network layers.

E. Training The Model

Now that both the model and training data are ready for use, the training process may begin. There are two extra parameters which need to be defined when the model undergoes the learning process, *epochs* and *batch size*. Epochs refer to the number of training iterations while batch size refers to the number of training samples per iteration. Additionally, a batch size can also be specified for *validation*, when the model is evaluated

against a subset of the training data. 10 epochs were used with a batch size/validation batch size of 64. These numbers were chosen as a balancing act between training time and completeness while also avoiding *overfitting*. Overfitting is when a model memorizes the properties of training data to an extent that it performs negatively on new data.

IV. Results

In this section, the metrics of the model will be analyzed along with its performance on new data.

A. Metrics

There are two main properties to evaluate in a machine learning model, its *accuracy* and *loss* over the duration of training. These two properties share an inverse relationship; accuracy determining the model's correctness and loss representing errors in prediction. Using an additional library, *Matplotlib*, a graph representation of accuracy/loss can be produced for the model.

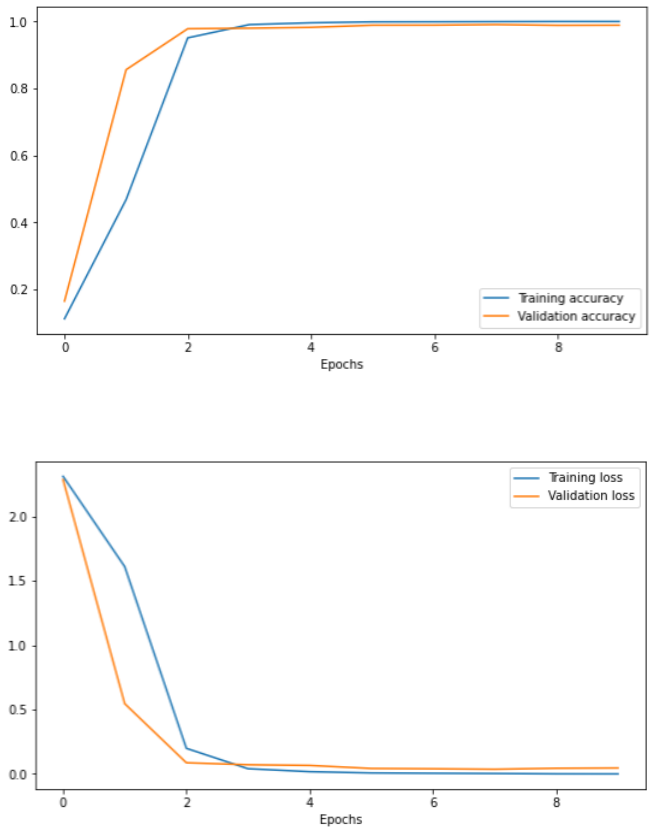

Figure 9. Accuracy and loss metrics of model during training.

Accuracy is represented as a percentage, with higher percentages being favored. On the other hand, loss is represented as a decimal value with lower values being favored. It is important to note that a model will practically never have an accuracy of 100% or a loss value of 0.

The model was able to reach over 90% accuracy and minimize loss to less than .5 after just 2 epochs. After 10 epochs, the model's accuracy reached 99% with a loss of only .03. These metrics indicate that the model has good performance.

B. Performance On New Data

To test the performance of the model, new labelled data will need to be generated. The same methodology for creating the training data will be used for the *test dataset*. The test dataset consists of new CAPTCHA variations which the model has not been exposed to. The trained model will then take each CAPTCHA image from the test dataset as input, and return a 4-digit prediction value. The *Matplotlib* library will be used again to visually represent the model's interpretation of CAPTCHA images. Each graph instance will contain the model's prediction value, CAPTCHA image, and true value (label for CAPTCHA image). An example of the model's performance is demonstrated on 9 new CAPTCHA images, where it was able to accurately guess the contents of 8 images.

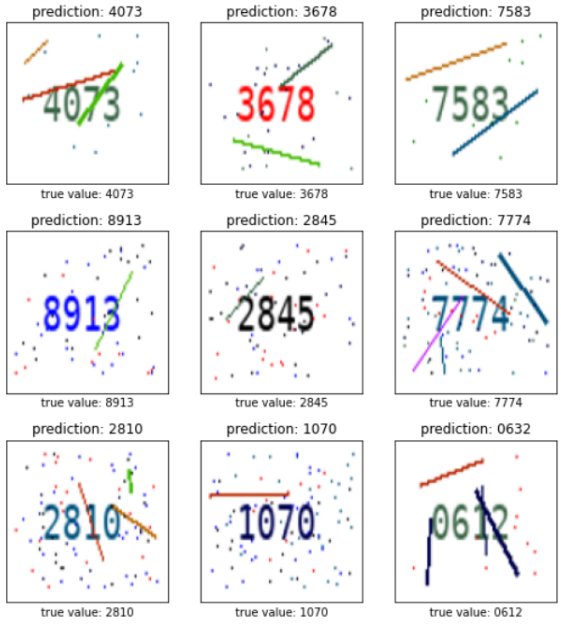

Figure 10. Model interpreting CAPTCHA images.

V. Challenges

In this section, the shortcomings of this research will be discussed. Specifically, an adversary who uses this

technique would be limited by several factors including the variability of CAPTCHAs and integrating the model into automation software.

A. Variability Of Real-world CAPTCHAs

While the model produced in this research is able to accurately solve CAPTCHAs created from the PIL library, it is unable to solve CAPTCHA images used on live websites. This is due to the inherent variability of CAPTCHA-generating algorithms used on different platforms. Real-world CAPTCHAs will use a variety of fonts, image sizes, character-lengths, and distortion techniques to produce their images. The PIL library alone would be infeasible for creating the variation of CAPTCHA images required for use on live websites.

B. Integration Into Automation Software

The model produced in this research is currently confined to the Jupyter Notebook environment. For the model to be of any use to an adversary, it would need to be exported and integrated into another software system. Currently, TensorFlow supports model deployment to a Python application, browser environment, web application, mobile devices, or server environment. However, there is no general procedure for integrating the model into browser automation software (such as Selenium). Since an adversary would want to leverage machine learning for CAPTCHA-solving automation, a real-world bot with such capabilities is currently unfeasible.

VI. Key Contributions

VII. Future Work

References

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.