Breaking Captcha Images using Machine Learning

Garvit Harisinghani
gharisin@andrew.cmu.edu

Jaagrit Arora
jarora@andrew.cmu.edu

Sankalp Devasthali
sdevasth@andrew.cmu.edu

ABSTRACT

The rise in automated traffic since the late 1990s raised a challenge for web administrators - to restrict automated traffic on their sites. The growth of computing power, internet access and improved programming paradigms meant that malicious users were able to craft techniques to create massive amounts of automated traffic. At best, this increased server loads and reduced response times. At worst, such traffic peaks could potentially crash a site. Thus, there was a need to differentiate between human users and Bots.

CAPTCHA was born in 2003 at Carnegie Mellon University by a team led by Luis von Ahn [1]. CAPTCHA is an acronym that stands for: Completely Automated Public Turing Test to tell Computers and Humans Apart. It is a challenge-response test that determines if a user is human or not. The complexity of the CAPTCHA test has grown in conjunction with the growth in sophistication of machine learning algorithms.

This project is an attempt to utilize the power of Machine Learning to "break" CAPTCHA images and validate their obsolescence today. The project's objectives have evolved with time to create a more robust system capable of recognizing characters in real world images. The outcome of the project was achieved with the process created to break CAPTCHA images achieving a test accuracy of 93.27% on randomly generated CAPTCHA images, against a baseline of 72.84 %. The convolutional neural network classifiers using the same training and test achieved a test accuracy of 97.48%. On the more complex and noisy Street View House Numbers[10] (SVHN) dataset, the neural network achieved a test accuracy of 70.08%.

1. Introduction

CAPTCHA has been used since the early 2000s as a defense mechanism against automated traffic from bots and/or malicious users. It has become a de facto standard for differentiating human users from bot traffic [2]. However, CAPTCHA has had some criticism as it is difficult for disabled persons to be able to either read or solve CAPTCHA images.

Modern text-based CAPTCHAs require the use of three techniques in tandem to solve a captcha challenge correctly:

- Invariant recognition: The ability to recognize a character with large and differing amounts of distortion and variation.
- Segmentation: The ability to successfully separate one letter from another, even if they seem joined together
- Context: Differentiation between an 'O' letter and 0 digit can be done only when context is considered [3]

1.1 Evolution of CAPTCHA:

Early creations of Captcha were relatively simple – images of characters and digits in different fonts arranged at angles, all at random. As is the case with every technology, hackers and security researchers alike devised methods to beat these early systems. The CAPTCHA ecosystem has evolved to keep pace with these developments. reCAPTCHA was conceived again at Carnegie Mellon University by Luis von Ahn, Colin McMillen, David Abraham and Manuel Blum.

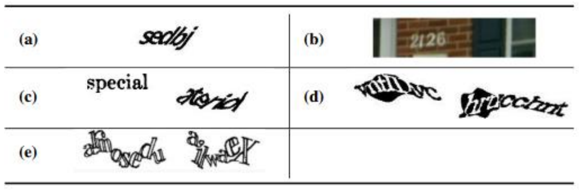


Fig 1: Variety of CAPTCHA

Since then, the types of CAPTCHA can be classified as follows (Fig. 1):

a. Distorted one word
b. Street numbers image challenge
c. Scanned word challenge
d. Distorted two words
e. Fallback [3]

Lastly, there are also image-based CAPTCHAs: Match images from a set of 9 images that are similar to a given image

However, the most advanced system in common use today is reCAPTCHA and Google's noCAPTCHA system. The latter is used more than 100 million times daily. In most cases, users simply click on a challenge and the system determines if a user is human or not. If it determines that a user is not human, it poses challenges where a user must connect images based on a given keyword.

With this project, we attempt to demonstrate that CAPTCHA is obsolete, and that character recognition can no longer serve as an effective check against automated traffic. The project also serves as this team's introductory/intermediate experience with the process of using machine learning for problem solving, where we go through the entire process of data analytics and character recognition.

Character recognition in complex images has long been a staple of the machine learning problem space. There are numerous complexities in extracting characters, removing backgrounds and textures and recognizing blurry images. There is no universal system that can classify images like these with absolute certainty. However, the growth in computing power and the increasing sophistication of machine learning algorithms mean that systems and processes can be devised that are capable of matching human accuracies in character recognition.

The SVHN dataset was created in 2011 to serve as a standard in measuring the accuracy of digit recognition algorithms on real world images. This dataset was used in the project to measure the model accuracy on real world images.

1.2 Disadvantages of CAPTCHA:

1 Difficulty of use for some users – disabled and the visually impaired
2 Actively dissuades users from using a site [4]

2. Data Understanding and Preparation

For this project our approach involves using multiple machine learning techniques to solve the problem, and present evidence for all the models we test. We identified two datasets - a dataset comprising of a collection of 10,000 CAPTCHA images generated from a WordPress plugin and the publicly available Google Street View House Number (SVHN) images.

2.1 WordPress Dataset

The CAPTCHA dataset has been split into training (80%) and test (20%). The size of the dataset is 40MB, as an image is approximately 1KB in size. Each image is a PNG file with a resolution of 78x24. Each image is a random combination of 4 letters and/or digits in random fonts and at random angles. The distortion in the shapes of the letters and their orientation serves as the 'noise' in the dataset, and the machine learning model must be robust enough so that it can have a decent accuracy. The CAPTCHA generator used in this case specifically does not generate 'O' and 'I' in CAPTCHA images to avoid user confusion with '0' and 'l'. Thus, the number of possible 'character' labels is 32 (24 alphabets + 8 digits). This implies a total number of possible combinations at 32^4. Due to the large number of samples and the randomness built into the generator, the distribution of digits across the images is almost close to equal. The data is loaded as images, and then converted from grayscale to a binary color code of 0 for black and 1 for white colored pixel. The image is then broken into its constituent characters.

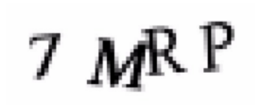


Fig 2: Sample CAPTCHA image

2.2 Street View House Numbers dataset

The Street View House Numbers (SVHN) dataset is a collection of images that contain house numbers from actual streets, pulled from Google Street View. The dataset has been created for the express purpose of improving the capability of machine learning algorithms. It can be thought of as an evolved version of the MNIST dataset. However, due to the real-world nature of the images and the inherent noise, classification of images is significantly harder.

The training dataset contains 73257 images (~670MB). The test dataset consists of 26032 images(~167MB). The number of digits in each image is variable. There are an additional 531131 additional images available. The number of class labels is 10 for each of the 10 individual digits.

These images are available in two formats:

Format 1:

Each of the three datasets contain color images of varying sizes, with a variety of backgrounds and textures. The bounding boxes for each of the images have been predefined in MATLAB data format (.mat). The bounding boxes define the area to be considered for processing of each digit. Digits need to be extracted and processed manually.

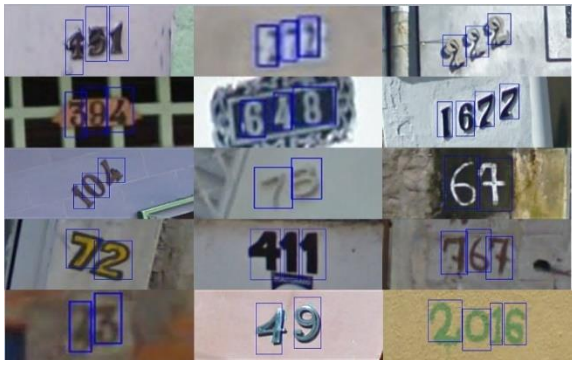


Fig 3: SVHN dataset with bounding boxes

Format 2:

Individual images in this data are in an MNIST like format, with the digits from images extracted into 32x32 individual images. However, some of these images have 'distractor digits' i.e. image contains multiple digits, only one of which is the digit of interest. For e.g.

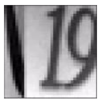
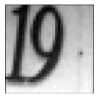
Label 1:  Label 9: 

Fig 4: Extracted SVHN dataset

The images in both datasets need to undergo significant preprocessing before they can be used for training and testing. The following procedures were used on the images to make them more suitable for character recognition:

1. Image thresholding (Otsu's binarization)
2. Denoising

The justification for each of these processes and their place in the process has been discussed in detail in the next section.

3. Methodology

The team followed the standard method of solving a machine learning problem: Dataset selection, feature engineering, model building, and testing. Python was chosen as the language of choice as it has a robust library for image processing, mathematical computation and machine learning. It also supports rapid and agile model building and testing.

There were three challenges that had to be dealt with:

1. Minimizing noise
2. Breaking the image into constituent characters
3. Ensuring correct binary representation of the image

Listed below are the solutions for each of the three problems listed above:

1. Image thresholding was used to remove grays in the image and reduce all pixels to black (0) or white (1). This ensured that the individual characters would be better defined, and that gaps between the characters would also be better seen.
2. The algorithm used to break the images into digits operated on the assumption that there are at least 2 columns of white pixels in between characters. This heuristic is further explained later in this section.
3. The desired binary representation of the image required the image pixels to be black while the background pixels/noise to be white. Since the images were captured in a variety of lighting conditions and camera angles, the background and foreground isolation process was not reliable. Ideally, the house number itself should be identified automatically as the foreground and colored black (value: 1). However, the darker and more blurred images tended to have black backgrounds with the digits themselves in white. This reversal also had to be dealt with by inverting the pixel values where required.

For the WordPress dataset, the approach was to classify individual symbols instead of the complete image for the following reasons:

1. Computational Efficiency: 18x24 features vs 72x24 features for the complete image
2. Practical Considerations: The increase in the number of possible target labels from 32 to 32^4 would mean that the model would have to be trained on vast amounts of data. This is simply not possible. With the limited data available, the resultant model would also not be resistant to noise.

The following terms need to be defined before describing the neural network created for this project:

- Kernel: 'A small matrix (the size of the window) used to apply effects to an image'[11]

- Window: A subset of the image that is currently being used for processing

- Convolution: A convolution in image processing is an operation used for feature extraction from an image. Techniques such as sharpening, blurring and/or edge detection can all be implemented by varying the kernel used in the window. Generally, the output of the convolution is a dot product (scalar) of the window and the kernel.

- Conv2D layer: Defines and learns the kernel to create the output to the next layer. Performs the convolution on the input image (sent as a 2D matrix). The output of this convolutional layer is fed into the fully connected layer.

- Dense/Fully Connected Layer: A layer that performs a linear operation for every input supplied to it. Every output node/tensor gets a linear transformation of fewer inputs dictated by the weights associated with the edges.

- Max Pooling layer: Replace each window with a single value. In this case, that value is the maximum value in that window.

- SoftMax Activation: This is simply a collection of neurons with a sigmoid activation function

The final design of the neural network is as follows:

Input Layer → Convolutional Layer → Dense Layer → SoftMax Activation Layer

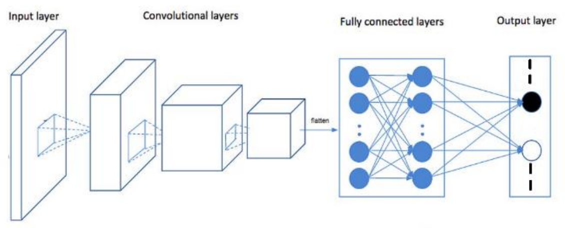


Fig 5: Convolutional Neural Network

3.1 WordPress Dataset

The dataset of 10,000 CAPTCHA images was imported using a combination of OpenCV2 and glob modules. OpenCV2 provides numerous inbuilt functions to process images such as thresholding, resizing, and image reading. The training labels were obtained from the file names of individual images. The test dataset consists of 2,000 (20% of total) randomly sampled images from the dataset. From here on, the remaining 8,000 images were used for training.

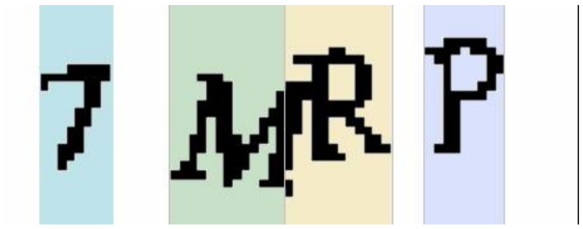


Fig 6: Sample WordPress CAPTCHA Image

For the aforementioned reasons, a method was devised to break the image into its constituent characters. Image thresholding was used to ensure binary pixel levels of black (0) or white (1) for the images. The image matrix was parsed column wise and split to extract each continuous group of black pixels as individual characters of each image (also called contour extraction). A heuristic served as the basis to split the image into its constituent characters. The image generated from this heuristic was again split into two if the width exceeded the height by a factor of 0.75. This was done to account for the cases where two characters were joined together on account of their orientation. The resultant constituent images look like the following:

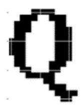


Fig 7: Sample Broken Image from WordPress dataset

Each extracted character image was resized to a final size of 18x24 pixels to standardize the number of features. This resized image was fed into the model as training data. The final training data set consisted of 31428 flattened images having 432 features.

Three techniques were considered for the purposes of this project:

1. Naïve Bayes (as baseline),
2. Multi-class One-vs-Rest (Support Vector Machines) SVM
3. Neural Network

The Naïve Bayes classifier served as a classifier as it provided a probabilistic method[12] to predict the digits by assuming all features are Independent and Identically Distributed. On the other hand, SVM was chosen as the individual digits formed usable clusters in the high dimensional space – which meant that a method that tried to define decision boundaries based on the distances between data points of different class labels would work well [5]. The two-dimensional t-SNE representation showed that the 'clusters' of digits in high dimensional space were relatively well separated. The behavior of t-SNE is consistent with its expected output where the separation is well-defined towards the edges of the graph while the center of the images consists of labels that are not so well-separated [6] [7] [8] [9].

Finally, the Neural Network was used to determine how modern and more complex techniques would be behave when compared to more 'classical' techniques.

The approach with Neural Networks involved creating two models – a single layered network with 32 outputs and SoftMax activation, and a convolutional neural network with four layers – a convolution layer, a max-pooling layer, a flatten layer, and a dense layer of 32 outputs having SoftMax activation. The optimizer chosen in both implementations was ADaptive Moment Estimation (Adam). It is computationally efficient, requires minimal hyperparameter tuning and is suitable for large datasets. Categorical Cross-Entropy was selected as the loss function.

The model was trained using an unregularized and 'l2' regularized Linear SVM as well as using a simple and a Convolutional Neural Network on the training data.

Following is an illustration of t-SNE keeping preserving similarity while reducing dimensionality: In the box defined above, the digits from left to right (yellow onwards) are 6 ⟶ 8 ⟶ 5 ⟶ 3 ⟶ S, all of which look similar and have been placed together.

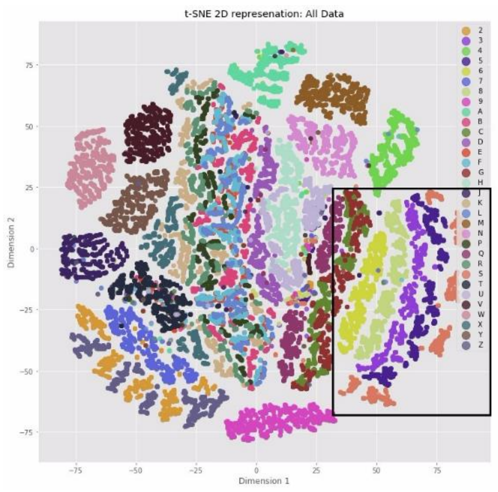


Fig 8: A t-SNE visualization of different clusters of the data. There are 32 clusters in all.

A similar procedure was used while testing where the model predicted individual characters in the test images and the accuracy was calculated for the correct prediction of the entire CAPTCHA image.

3.2 SVHN Dataset

The dataset of 73,257 house number images was imported using SciPy's MATLAB interface and processed using OpenCV2. The training dataset comprised of digit images extracted into 32x32 individual images. The test dataset consists of 26,032 images from the dataset.

The images were denoised to improve edge detection and increase contrast between background and foreground and minimize blur. This was followed by converting images to their binary equivalent. Some images had undiscernible background and foreground representations, and these were dealt with by inverting the pixel values wherever appropriate

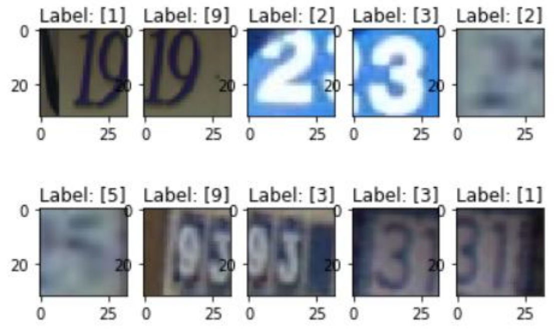


Fig 9: Sample 32x32 SVHN Training Images

The approach with Neural Networks involved creating a convolutional neural network with four layers – a convolution layer, a max-pooling layer, a flatten layer, and a dense layer of 10 outputs having SoftMax activation. The optimizer chosen was ADaptive Moment Estimation (Adam) and Categorical Cross-Entropy was selected as the loss function.

4. Result and Evaluation

The Unregularized SVM model shows an accuracy of 97.9% on the character level and 93.27% on the image level when provided with 2000 CAPTCHA images for the WordPress CAPTCHA dataset. A higher accuracy is obtained by using a CNN where 97.48% of the test data was predicted correctly. For the noisier dataset (SVHN), the accuracy with a CNN model was 70.08% at the digit level. The overall accuracies are listed below:

Model	Validation Accuracy	Test Accuracy
Unregularized SVM	96.60%	93.27%
Regularized SVM	69.36%	69.63%
Simple Neural Network	88.79%	88.28%
Convolutional Neural Network	99.02%	97.48%

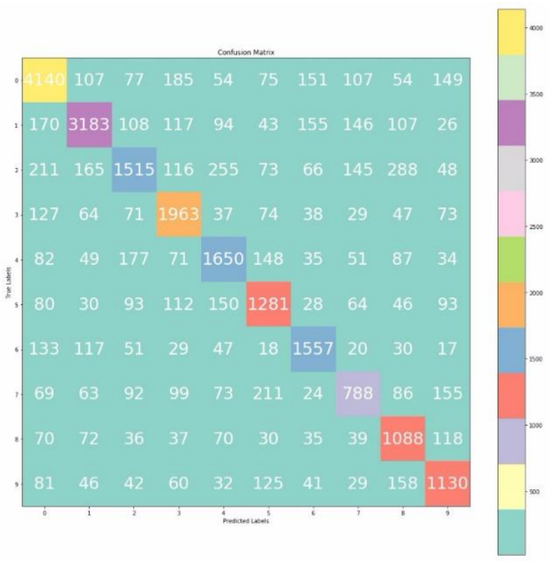
Table1: Accuracies on WordPress CAPTCHA dataset

Model	Validation Accuracy	Test Accuracy
Convolutional Neural Network	72.56%	70.08%

Table2: Accuracies on SVHN dataset

This model uses relatively simple classifiers: linear decision boundaries in high dimensional data. However, since the training data consists of images, image structure can be used to achieve higher accuracy and used against more complex CAPTCHA challenges. To that effect, multiple CNN models were implemented on both the datasets. This method took advantage of the structure of the image and yielded greater accuracies than the simpler, decision boundary-based classifiers.

The confusion matrix for the SVHN dataset is given below



	0	1	2	3	4	5	6	7	8	9
0	4140	107	77	185	54	75	151	107	54	149
1	170	3183	108	117	94	43	155	146	107	26
2	211	165	1515	116	255	73	66	145	288	48
3	127	64	71	1963	37	74	38	29	47	73
4	82	49	177	71	1650	148	35	51	87	34
5	80	30	93	112	150	1281	28	64	46	93
6	133	117	51	29	47	18	1557	20	30	17
7	69	63	92	99	73	211	24	788	86	155
8	70	72	36	37	70	30	35	39	1088	118
9	81	46	42	60	32	125	41	29	158	1130

Fig 10: Confusion Matrix for CNN for SVHN dataset

Currently, the model is trained using relatively simple machine learning models. However, more complex CAPTCHA challenges would have to be dealt with by better feature engineering and through improvements in the modeling process. Improving image processing, handling post thresholding foreground/background inversion, and building a process to detect and list all digits in an image would be the next phase of the project.

REFERENCES

[1] von Ahn, Luis; Blum, Manuel; Hopper, Nicholas J.; Langford, John (May 2003). *CAPTCHA: Using Hard AI Problems for Security. EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques.*

[2] Chellapilla, Kumar; Larson, Kevin; Simard, Patrice; Czerwinski, Mary. *"Designing Human Friendly Human Interaction Proofs (HIPs)"* (PDF). Microsoft Research. Archived from *the original* (PDF) on 10 April 2015.

[3] Sivakorn, S., Polakis, I., & Keromytis, A. D. (2016, March). I am robot:(deep) learning to break semantic image captchas. In Security and Privacy (EuroS&P), 2016 IEEE European Symposium on (pp. 388-403). IEEE.

[4] https://resources.distilnetworks.com/all-blog-posts/to-captcha-or-not-to-captcha

[5] Arora, S., Bhattacharjee, D., Nasipuri, M., Malik, L., Kundu, M., & Basu, D. K. (2010). Performance comparison of SVM and ANN for handwritten Devanagari character recognition. arXiv preprint arXiv:1006.5902.

[6] Thome, A. C. G. (2012). SVM classifiers–concepts and applications to character recognition. In Advances in Character Recognition. InTech.

[7] Sadri, J., Suen, C. Y., & Bui, T. D. (2003, February). Application of support vector machines for recognition of handwritten Arabic/Persian digits. In Proceedings of Second Iranian Conference on Machine Vision and Image Processing (Vol. 1, pp. 300-307). https://pdfs.semanticscholar.org/d370/8948a7efb2ef007c4a6e80948bc7a0d164b7.pdf

[8] Bursztein, E., Aigrain, J., Moscicki, A., & Mitchell, J. C. (2014, August). The End is Nigh: Generic Solving of Text-based CAPTCHAs. In WOOT. https://www.usenix.org/system/files/conference/woot14/woot14-bursztein.pdf

[9] Bursztein, E., Martin, M., & Mitchell, J. (2011, October). Text-based CAPTCHA strengths and weaknesses. In Proceedings of the 18th ACM conference on Computer and communications security (pp. 125-138). ACM.

[10] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011.

[11] Image kernels explained visually, Victor Powell http://setosa.io/ev/image-kernels/

[12] Chris Piech, CS109, Stanford University, "Naïve Bayes" https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/pdfs/39%20NaiveBayes.pdf