Solving Text-Based CAPTCHA

Navdeep Sood
nssood@bu.edu
Department of Computer Science
Boston University
Boston, MA 02215

Ronald Vargas
rvar@bu.edu
Department of Computer Science
Boston University
Boston, MA 02215

Abstract

Several websites require image-based authentication to add a layer of security to websites meant to be difficult enough so that simple computerized algorithms cannot identify what the image is actaully displaying. In this paper, we try to improve upon existing forms of algorithms that seek to recognize and bypass skewed text-based Completely Automated Turing Test to tell Computers and Humans Apart (CAPTCHA) images. Based on an convolutional neural network implemented by Geitgey [1], we implement a *k*-NN classification that creates one classification for each possible alphanumeric character that may be displayed in the image. We evaluate our model by taking a small portion of our training set and calculating accuracy on the remainder of the data after training has completed. We demonstrate that we are able construct an algorithms that is able to detect altered characters in forms of CAPTCHA.

1 Introduction

Text recognition is a fundamental aspect of machine learning and artificial intelligence. As we strive to make processing paperwork processing faster and more reliable, we turn to technology to create efficient code that will do the job that would otherwise take many man-hours to complete. Many text recognition algorithms have risen as a result of this quest for efficiency though most have done so by implementing algorithms that use neural networks or support vector machines. At the same time, privacy concerns arose as automated computers were sent to the internet to try and capture data while mimicking the strokes of average individuals. This lead to the creation of CAPTCHA. This comprised of altered phrases or skewed characters that were not easily recognized even by humans. The solution to this was creating a convolutional neural network comprised of two convolutional layers and one hidden layer to extract edges around the characters that would then reveal an pattern for every type of alphanumeric characters, skewed or unchanged. In this work we tackle the problem of trying to implement a similar algorithm that groups and efficiently recognizes deformed and off-axis characters that would not be classified as alphanumeric characters otherwise. Our work follows a close structure to Scanned Digits Recognition using k-Nearest Neighbor [2] and aims to achieve greater accuracy through larger training data and image sampling. We formalize this task by extracting consistent sample characters from a set of sample CAPTCHA images. We then create histograms of gradients for each digit that will ultimately train our *k*-NN classifiers. It can be trained in an unsupervised manner to perform perceptual grouping in order to learn an efficient representation for each classification.

2 Data Processing

We begin by extracting the individual character from our CAPTCHA samples that contain a string of 4 skewed characters. This is done by thresholding the images. This will make it easier for the

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

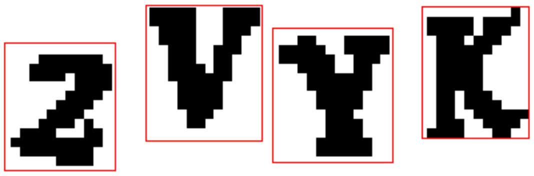

Figure 1: Character Extraction

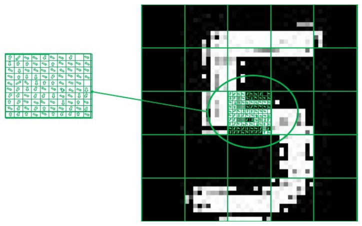

Figure 2: Gradient Visualization

algorithm to find the contiguous regions that it will then split by using OpenCV's *findContours()* that will provide us with coordinates for the boundaries of the characters. See Figure 1. For consistency, images are set to the same *16x16* resolution to more easily train our model. We then upscale our extracted characters to *50x50*.

2.1 Error Handling

CAPTCHA images are meant to be very off-centered and abnormally italicized characters meant only to be recognizable by actual humans. Because this is also present in the training set, the *findContours()* function detects up to two characters, but classifies them as one. In this case, the larger coordinates are divided along the center of the length and standardize the two characters individually. This decreases the likelihood of creating a sizable percentage in error.

2.2 Feature Extraction

Upon completion of character separation, scikit-image's Histogram of Gradients (HOG) function

```
hog_image = hog(image, orientations=8,
pixels_per_cell=(x, y), cells_per_block=(z, a));
```

where x and y are the sizes of subsections of the image. z and a represent the number of gradients that will appear in each subsection per row and per column respectively. HOG was chosen because the program counts occurrences of gradient orientation in the specified portions of the image selected. This happens by distributing the gradients or edge directions and is then divided into smaller regions called cells, and for the pixels within each cell, a histogram of gradient directions is then generated. This is illustrated in Figure 2.

2.3 Training Objective

k-NN is a non-parametric method for classification whose outcome relies on the k nearest points of data to the point being selected. We are interested in the particular grouping of data points that correspond to each letter of the English alphabet as well as numbers. Our decision to use the same training set as the one generated by Geitgey [1] meant we did not include classifications for lower case characters.

Table 1: Training Method Accuracy

Model	Method	Accuracy (%)
CNN	ReLU	99
CNN	Sigmoid	92.4
k-NN	32-Classifiers	99.56

2.4 Training

The training primarily uses HOG to find trends and patterns in the training data. These histograms are used to create 32 character classifiers (digits 2-9, alphabet letters excluding "I" and "O"). Histograms are created by dividing the extracted character into a 5x5 grid. Each square in the grid is a grid itself of 5x5 pixels. A HOG function is used to obtain a histogram of the entire extracted character, and the data is stored into an array that contains its character and label to be used later for pattern analysis. The model is set to classify using $k = 5$, meaning that a data point will be categorized according to the histograms of its five nearest neighbors. The training time for the *k*-NN model was 42.5 seconds on average. The value $k = 5$ was chosen because it provided a sufficiently high accuracy of 99.56% without compromising training time. This accuracy value is displayed in context with the accuracy values of the other utilized training methods in Table 1. There were 38,700 single character data points in total, of which 10% were removed (not used in training) for use in testing. Note that *k*-NN stores all training data and classifications leading to an overall larger model in size compared to convolutional neural networks which only store weight attributes.

3 Testing

Testing was conducted using a random sample of 10% of the data points. The model was designed, given a testing data point input, to output a predicted letter along with a set of probabilities corresponding to the likelihood of the data point being each character. A separate algorithm was devised that extracts individual characters from full CAPTCHA samples, similar to those explained in Section 2, and passed into the prediction algorithm which checked to see whether the machine's prediction was in fact the actual character present in the image. The testing accuracy achieved was 99.5% on the testing data set, which consisted of 1024 extracted characters. That is, the model correctly identified 99.5% of the testing data inputs. A table comparing the baseline and training accuracy based on our model is shown on Table 1.

In early experiments, we modified the source's algorithm and set the activation functions to Sigmoid: a change from the baseline Rectified Linear Units (ReLU) activation method. ReLU's advantages for this approach became evident. The ReLU activation function has an advantage over sparsity as well as a reduced likelihood of a vanishing gradient. We can prove this by fist stating the definition of a ReLU is:

$$ReLU(a) = max(0, a) \text{ where } a = Wx + b$$

for some weight W and bias b. The reduced likelihood of a vanishing gradient arises when $a > 0$. For this case, the gradient has a constant value. In contrast, the gradient of sigmoids becomes increasingly small as the absolute value of x increases [3]. This ultimately signals that the constant gradient of ReLUs results in faster learning. This is where we decided on an alternate approach to trying to improve upon the creation and success of the *k*-NN approach.

4 Conclusion

The K-NN model trains remarkably faster than the convolutional neural network while achieving similar accuracy using the same training and testing data. The K-NN approach posed a great challenge for us in terms of implementation. Even though the baseline approach implemented by Geitgey [1] proved remarkably well achieving nearly 100% accuracy, we created a successful competitor. With a new training time of approximately 1 minute, we achieved a 75% quicker algorithm compared to a conventional convolutional neural network. Pre-processing the data set to our model proved to be a challenge as the images were at first not suited for our model. With careful processing and training using histogram of oriented gradients we were able to successfully train a model that achieved training and testing accuracy that matched those of the convolutional neural network, which is considered to be a state of the art image processing algorithm.

References

[1] Geitgey, A. (2017, Dec 13) How to Break CAPTCHA in 15 Minutes. *Medium*. Retrieved from *https://medium.com/@ageitgey/how-to-break-a-captcha-system-in-15-minutes-with-machine-learning-dbebb035a 710*

[2] Monghnieh, H. (2018, Feb 4) Scanned Digits Recognition using k-Nearest Neighbor (k-NN). *Towards Data Science*. Retrieved from *https://towardsdatascience.com/scanned-digits-recognition-using-k-nearest-neighbor-k-nn-d1a1528f0dea*

[3] Guo, B. (2016, Nov 26) ReLu compared against Sigmoid, Softmax, Tanh. *Quora*. Retrieved from *https://algorithmsdatascience.quora.com/ReLu-compared-against-Sigmoid-Softmax-Tanh*