CAPTCHA Breaking with Deep Learning

CS 229 Final Project, Autumn 2017

Nathan Zhao **Yi Liu** **Yijun Jiang**

1 Introduction

CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) are distortions of texts or images so that they are still recognizable to most humans but can become difficult for a computer to recognize. Additionally, many CAPTCHAs also inject noise or additional structures such as blobs or lines into the image to further make it difficult to recognize. They are primarily employed as security measures on websites to prevent bots from accessing or performing transactions on the site.

On the other hand, machine learning (ML) algorithms, in particular deep learning (DL) neural networks have been trained with significant success on similar problems such as handwritten digit recognition. This motivates us to build an ML-based CAPTCHA breaker that maps CAPTCHAs to their solutions.

2 Related work

As CAPTCHAs are actively used by many websites to protect traffic, major corporations have already invested significant resources in breaking CAPTCHAs to assess the strengths of shortcomings of these data techniques. Google's StreetView team, for example, have used their algorithms for recognizing signs in images on the CAPTCHA problem, achieving 99.8% [5] success on particular types of difficult-to-read CAPTCHAs.

3 Dataset and features

We have used *PyCaptcha*, a python package for CAPTCHA generation, to make custom CAPTCHA image dataset. This package offers several degrees of freedom such as font style, distortion and noise, which we can exploit to increase the diversity of our data and the difficulty of the recognition task.

For the first step, we have created single-letter CAPTCHA images (40-by-60 pixels) by feeding *PyCaptcha* uppercase letters ranging from A to Z from a restricted set of fonts. The resulting images were labelled by the corresponding letters. This thus gave us a supervised classification problem with 26 classes.

For an actual CAPTCHA breaker, we need to map an image into a string of letters. Therefore, we also generated a four-letter CAPTCHA image (160-by-60 pixels) dataset. However, we cannot take each distinct four-letter string as a label, as that will give too many classes. We stored the full four-letter string, but as is discussed in a later section, we only used 26 single-letter labels in our multi-letter CAPTCHA algorithms.

A typical four-letter CAPTCHA image that we have generated is shown in Fig.1.

Finally, we generated a set of 1580 single-letter and four-letter CAPTCHAs that came from a different library of fonts which further measures how well our algorithms have truly learned the fundamental archetype of each letter based on a restricted set of training data. This is referred to as the "prediction" dataset.

Fig.1 (Left) a typical CAPTCHA, which is an image distortion of the string "ADMD". (Right) a sample of two letters from the prediction dataset.

4 Methods

4.1 Single-letter CAPTCHA recognition

4.1.1 t-Distributed stochastic neighbor embedding (t-SNE)

As our training images represent a 2400-dimensional (40-by-60 pixels, pre-processed into grayscale) feature space, we used a nonlinear dimensionality reduction technique. t-SNE embeds high-dimensional data into a 2D or 3D space by converting Euclidean distances between points into conditional probabilities. Here it was used to visualize the data as well as to show how well data can be clustered. We applied Alexis Cook's *tsne* Python package.

4.1.2 k-means clustering

We implemented the k-means clustering algorithm using the *kmeans* classifier in the *scikit-learn* library. The input data were grouped into 26 clusters. To determine the prediction for each cluster, we picked the most common label in each group and used it as the prediction for the whole cluster. The k-means result gives a rough indication of how inherently 'learn-able' the dataset is.

4.1.3 Support vector machine (SVM)

We built a support vector machine with the *LinearSVC* classifier in the *scikit-learn* library. Multi-class classification was done by "one-vs-the rest" strategy. L2 regularization and hinge loss function were used.

4.1.4 Convolutional neural network (CNN)

We have used *TensorFlow* back-end and *Keras* as a front-end to train our CNN, as these packages are well-designed and easy to get started for a deep learning project. We designed our own network from scratch using these packages, trying out several but ultimately settling on the architecture as shown in Fig.2. In our design, we considered factors such as the number of parameters, layers, activations, filter sizes, etc.

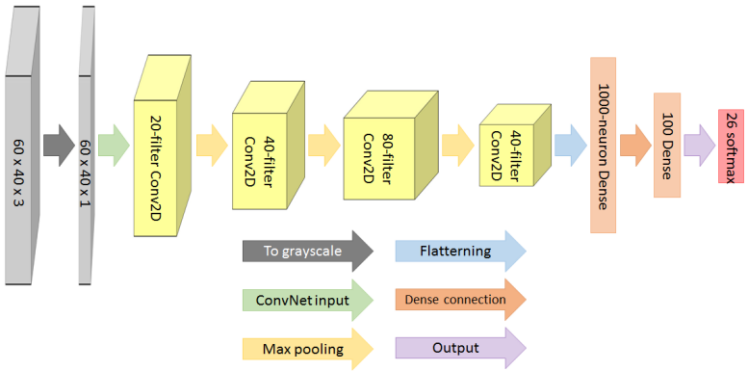

Fig.2 Structure of our convolutional neural network

In addition to designing our own neural net architecture, we also considered transfer learning with a pre-trained neural network on ImageNet. The specific net we used is called VGG-19, which contains over 20 million free parameters (shown in Fig.3). We intentionally froze many of the last convolutional layers while keeping the first few, as they are the ones which are able to characterize high-level features important for pattern recognition.

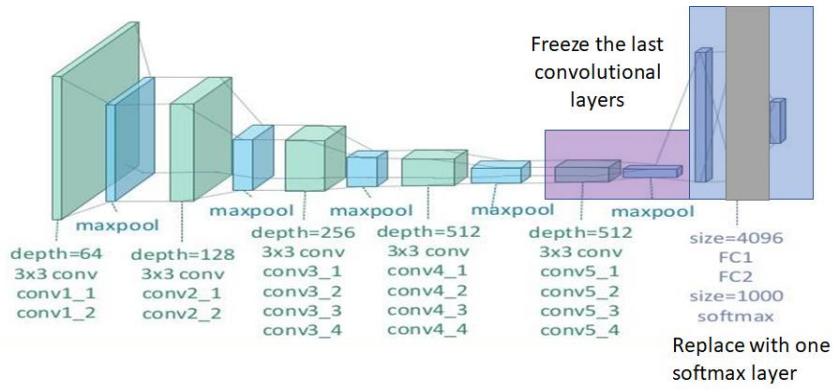

Fig.3 Structure of VGG-19 and our freezing of many last convolutional layers

While we attempted to minimize as many training parameters we have, we found that in freezing out more layers, the performance of the network on our dataset got worse. Therefore, in the end, we have left 19 million training parameters in our optimal network (which we train on using an Nvidia GTX 1080 GPU).

4.2 Multi-CAPTCHA recognition algorithms

4.2.1 Moving-window algorithm

We have developed two algorithms to recognize a four-letter CAPTCHA image of 160-by-60 pixels. The first one, referred to as the "moving-window algorithm", directly uses the well-trained single-letter CNN reported above. A subset of the full image defined by a 40-by-60 "window" was revealed to the CNN. This window was translated across the full image, with a prediction made at each position. It is expected that when the window moves across a specific letter, the softmax probability of that letter will first increase, then plateau at a high confidence when the window is well-centered, and finally decrease. Thus, from the dependence of prediction probability on window position, with some threshold parameters being manually set, we were able to extract the full four-letter string. Fig.5 shows an example of a correct prediction on the CAPTCHA string "UXZE", where four distinct plateaus correspond to the four digits in the string.

4.2.2 Multi-CNN algorithm

The second algorithm, referred to as the "multi-CNN algorithm" uses four fixed and overlapping windows to crop out four 70-by-60 sub-images from the full image of 160-by-60. Using our four-letter CAPTCHA dataset, for each of the four sub-image crops, we trained one CNN with the labels being one of the four digits in the strings. To be specific, the first crop was from pixel 0 to 69 in length, and labelled by the first digit; the second crop was from pixel 30 to 99, labelled by the second digit; the third from 60 to 129, labelled by the third digit; and the fourth from 90 to 159, labelled by the last digit. All four crops used the full width. Each of the four trained CNNs was used to predict one digit in the test images.

5 Experimental results

5.1 Single-letter CAPTCHA recognition

First, we characterized the separability or "learnability" of the dataset using t-SNE. As is shown in Fig.4, the minority of data can be well clustered but the majority of data are mixed together, therefore the data is generally not well-separated and non-trivial to "learn". Additionally, our result in using k-means shows a base-line accuracy of 30%, which confirms this intuition.

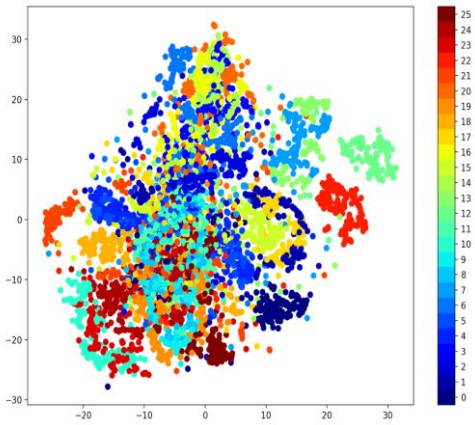

Fig.4 Clustering result of single-letter CAPTCHAs by t-SNE

Algorithm	Test Score	Prediction
Linear-SVM (single letter)	69%	0.1%
k-means (single letter)	30%	--NA--
CNN (single letter)	99%	60%
CNN (VGG-19 transfer, single letter)	95%	70%
CNN moving window (4 letters)	38%	0.50%
Multi-CNN (4 letters)	76%	0.75%

Table.1 Performances of different algorithms on font-restricted test dataset and prediction dataset

Unsupervised learning and linear classifiers did not perform well in recognizing single-letter CAPTCHAs. The test accuracy of SVM is only 69%, which is reasonable because SVM is susceptible to geometrical noises and not good at generalizing the abstract features of letters. It is also not surprising that the prediction accuracy of SVM is only 0.1% since the letter fonts in the prediction dataset are quite different from the font in the training dataset. For k-means, the performance is even worse, because, as we can see from t-SNE results in Fig. 2, most of the data cannot be well clustered. The unsatisfying performance of these algorithms shows the drawbacks of non-DL based learning in recognizing CAPTCHAs.

Meanwhile, a convolutional neural network trained on a single letter achieved significantly superior accuracies. The in-house architecture easily achieved test scores of 99% while the transferred VGG-19 achieved test accuracies of 95%. The disparity in the two can be attributed to the fact that we trained a lot of parameters in VGG-19, which probably led to some overfitting. However, we observe that when we try predicting on single-letter CAPTCHAs generated from new fonts, the accuracy of the transfer learned CNN wins out, achieving 70% accuracy vs 60% accuracy for our in-house architecture.

5.2 Multi-letter CAPTCHA recognition

For multi-letter CAPTCHA recognition, the moving-window algorithm achieved only half the accuracy of the multi-CNN algorithm. This is mainly because the moving-window algorithm involves explicit character segmentation, which is nontrivial given the fact that the letters have different widths and gaps. As a result, the moving-window algorithm may mis-recognize two letters as one letter, or part of a letter as another letter. In the multi-CNN algorithm, however, each sub-image crop is wide enough to fully contain the target letter. Therefore, it is the neural network's job to identify the single letter from a background of other digits and noise. The downside of this is that it inevitably makes the CNNs harder to train. In fact, in principle we could use the full image to train the four single-letter CNNs, but then the background is even noisier so that the CNNs did not learn well.

The prediction accuracies for both multi-letter CAPTCHA algorithms are below 1%. This is because both algorithms are based on single-letter CNNs. The accuracy of single-letter CNN must be raised to the power of 4 to give an upper bound for the multi-letter algorithms. We are also aware that there is significant error when the algorithms try to

identify where a letter is. Assuming 60% accuracy for single-letter CNN and 50% accuracy for position identification, we expect an overall upper bound of $(60\% \times 50\%)^4 = 0.8\%$ accuracy, which agrees with our experimental results.

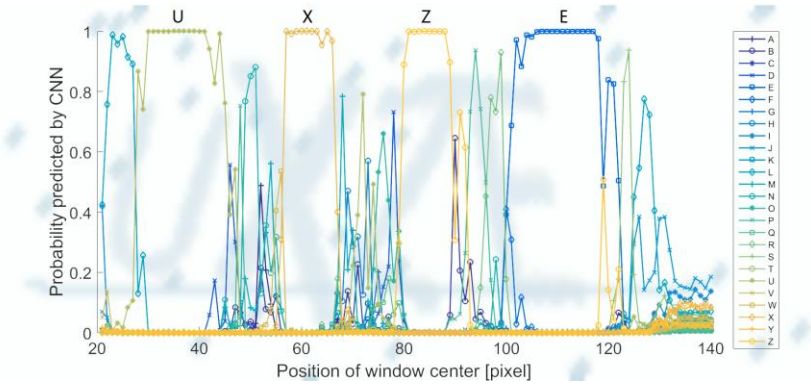

Fig.5 A correct prediction of the CAPTCHA "UXZE", where four distinct plateaus correspond to the four digits in the string. The original CAPTCHA in the background is aligned with the horizontal axis for comparison.

5.3 Error analysis: what is tripping up the net?

We see that our network generally has a larger problem with false-positives vs false-negatives. It seems to mistake a lot of letters as the letter T, Q, G, F, and U (false-positives). Additionally, letters like I, L, and F are the letters that are most likely to be false-negatives, but is less of a problem.

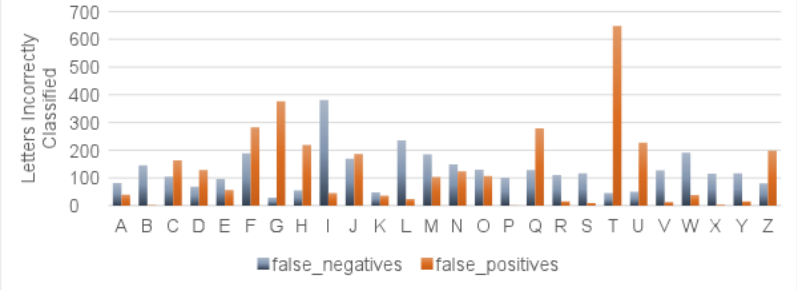

Fig. 6 Distribution of misclassifications in generalizing

6 Conclusions and future work

In this paper, we demonstrated the superiority of convolutional neural networks in learning and breaking letter-based CAPTCHAs. While we show single-letter classification results comparable to what is currently state-of-the-art, our attempts to generalize single letter learning to multi-letter CAPTCHA recognition were less successful, which now motivates several potential future paths.

In terms of future work, many interesting avenues present themselves. The simplest to consider would be massive data augmentation of the original training set. This is to assess whether we can effectively "approximate" all other font styles that might be used to generate a CAPTCHA without explicitly including them during training. Additionally, one could also attempt to optimize the specific architecture using hyperparameter optimization (which also requires significantly more computing power). However, as the parameter space for CNNs is quite large, alternative ideas such as metamodeling may be more feasible as in Ref. [1].

More technical advances would include more advanced neural network architectures such as capsule networks by Geoffrey Hinton [4]. These types of neural network architectures have been shown to generalize very well on image recognition tasks where the perspectives of one object can vary widely, which is basically akin to distorting the image in a highly non-trivial manner. Some recent work also shows recurrent neural networks can also be successful in decoding CAPTCHAs [3].

7 Contributions

Nathan Zhao generated the prediction datasets. He also constructed, trained and optimized the single-letter CNN, and performed transfer learning using the VGG-19 pre-trained network.

Yi Liu used t-SNE to visualize clustering of the single-letter datasets. She then applied SVM and k-means to classify single-letter CAPTCHAs. She also helped training the CNNs for multi-letter CAPTCHA algorithms.

Yijun Jiang generated the font-restricted single-letter and four-letter datasets. He also applied the single-letter CNN to the moving-window algorithm and developed the multi-CNN algorithm for multi-letter CAPTCHAs.

References

[1] Baker, B., Gupta, O. Naik, N. Raskar, R. (2017). Designing Neural Net Architectures Using Reinforcement Learning. International Conference on Learning Representations.

[2] Banday, M.T., Shah, N.A. (2009). A Study of CAPTCHAs for Securing Web Services, in *IJSDIA International Journal of Secure* Digital *Information Age*, 1**(2)**, pp. 66-74.

[3] Garg, Geetika & Pollett, C. (2016). Neural network CAPTCHA crackers. *Future Technologies Conference (FTC)*, San Francisco, CA, 2016, pp. 853-861.

[4] Sabour, S., Frosst, N., Hinton, G.E. (2017), Dynamic Routing Between Capsules. arxiv preprint.

[5] Goodfellow, I.J., Bulatov, Y., Ibarz, J. Arnoud, S., Shet, V. (2013). Multi-digit Number Recognition from Street View: Imagery using Deep Convolutional Neural Networks. arxiv preprint.

[6] *t-SNE* documentation: https://lvdmaaten.github.io/tsne/

[7] *sklearn* documentation: http://scikit-learn.org/stable/

[8] *Keras* documentation: https://keras.io