Compare Different Model's Performance on Breaking CAPTCHA

Haotian Sun, Jiaying Li, Yutong Zhang
Stanford University
{htsun, jiayingl,yutong16}@stanford.edu

Abstract

CAPTCHAs are widely used to secure the web system in current days. We conducted several tests on the most general type of CAPTCHAs that contains numbers and letters to study if the the deep learning model can simply break them. We used the image segmentation approach as the first step in our training, but we realized that the 4-character image with overlapping and rotations can be hardly detected correctly. Later, we used end-to-end learning and conducted several Convolutional Neural Network models in our study, including Simple CNN, VGG-16 and ResNet, and we reached an accuracy for more that 80%. The results show that a large training dataset is one of the major feature that will determine the performance of the training model. The complexity of dataset will burden the classification process as well.

1 Introduction

Nowadays, many people use computers to make bulk fake ID registration in system. To prevent this attack, CAPTCHA, a figure, including multiple letters and numbers that occluded with noises and curves, is eveolved. It is interesting to know how well the deep learning will perform on breaking these CAPTHCAS. Our goal is to identify the characters in generated CAPTCHA figures by training computers with neural network models, which is a process of translating figure in png format to a string with the predicted characters. The input to our algorithm is RGB matrix for the figure, with shape 140x80 (width x height), and the label for a figure is a 62 x n multi-hot vector (n=#characters). We used the following model to conduct the different training processes: Fully Connected Neural Network(FC), Simple CNN, LeNet with Segmentation of Image and Revised VGG-16. In order to further study the training performance under different difficulty of CAPTCHA, we prepared several different datasets and made multiple tests.

2 Related work

Currently the common method to break the CAPTCHA is to use CNN with active learning, which is an end-to-end approach. Fabian Stark et al. [1] discussed how to choose the new samples to re-train the network and present results on an auto-generated CAPTCHA dataset. Dongliang Xu et al.[2] also propose to use active learning to address the problem of the lack of initial data of CAPTCHA. Ye Wang et al. [3] have provided another solution to solve the problem. They present a self-adaptive algorithm to segment different kinds of characters optimally, and then utilize both the existing methods and their own constructed convolutional neural network as an extra classifier. Moreover, Iam J. Goodfellow et al. [4] discussed their success in aiming high prediction accuracy by using 5 convolutional layers in their model, and they suggested that 8 layers might have a better performance

(a) No Rotation and No Overlap (b) With Rotation and Overlap

Figure 1: Examples of Data with Different Difficulties

for the CAPTCHA detection for their dataset. Also, another way to break the CAPTCHA is using some data preprocessing methods to clean the data, as mentioned in Jeff Yan et al.[5].Another relavant work that used the segmentation method is done by Adam Geitgey [6]. However, the implementation that he made will be hard to get high accuracy on our dataset, since the noises and curves will impact the detection of contour as described in [6].

3 Dataset and Features

Datasets are generated from a python package called captcha[7]. This package can automatically generate png format image in RGB (3 channels).We changed several parameters that controls the rotation, warping, and overlapping of characters in the source code. The ratio for training, validation and testing sets is 10:1:1, which is 10240, 1024 and 1024 data in each set. The characters that we used are numbers from 0 to 9 and letters from a-z and A-Z in font DroidSansMono. Dots and curves are added to the figure with the internally implemented PIL package in captcha[7]. The labels are in the size of 1xnx62 as multi-hot vectors.

For the test of Fully Connected Neural Network, a data preprocessing phase is used to transfer the RGB matrix into gray scale. For the test under LeNet model, we segmented the single png figure into seperate images. Moreover, to see if the model can have a better performance when images are in grayscale, we prepared a comparison of output between grayscale dataset (1 channel) and RGB dataset (3 channel) for the VGG-16 Revised Model.

	FC NN	Seg+LeNet	Simple CNN	VGG-16 Revised	ResNet
Image Size	64x64	128x64	140x80	140x80	140x80
Input Size	64x64x1	128x64x3	140x80x3	140x80x3	140x80x3
Data Format	RGB	RGB	RGB	RGB	RGB
# of Classes	62	62	62	62	62
# of Characters (n)	4	4	4	4	4
Rotation Degree	0, 0	-30, 30	-60, 60	-60, 60	-60, 60
Overlap	0	0	0.2	0.2	0.2
Warp	0	0	0.3	0.3	0.3
Level of Difficulty	Easy	Medium	Hard	Hard	Hard

Table 1: Data Used in Each Model

	RGB Dataset with VGG-16	grayscale Dataset with VGG-16 Revised
Image Size	140x80x3	140x80x3
Input Size	140x80x1	140x80x1
Data Format	RGB	grayscale
# of Classes	62	62
# of Characters (n)	4	4
Rotation	-60, 60,	-60, 60
Overlap	0.2	0.2
Warp	0.3	0.3
Level of Difficulty	Hard	Hard

Table 2: Same Model with Different Dataset Difficulties

4 Methods

4.1 Loss Function

The function that we used to penalize our predictions from model is cross entropy (CE) loss:

$$CrossEntropyLoss = \sum(-y * log(p) - (1 - y) * log(1 - p)) \tag{1}$$

Since we have 62 classes of outputs, we will use the softmax activation functions in our model for classification.

4.2 Fully Connection Neural Network

We designed our baseline model using a fully connected neural network with 5 hidden layers. The input is a 1024x1 vector and the output is a 62x1 vector. We added sigmoid and softmax activation functions to improve the non-linearity of our model. Original input values are divided by 255 to be limited between 0 and 1. This model is only used to test the performance of FC on single character detection for CAPTCHA.

4.3 LeNet Plus Segmentation

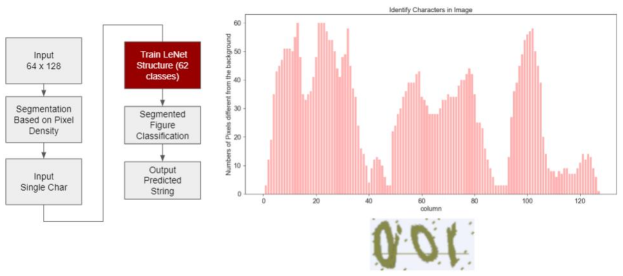

Input
64 x 128
Segmentation Based on Pixel Density
Input
Single Char
Train LeNet Structure (62 classes)
Segmented Figure Classification
Output Predicted String
Identify Characters in Image
Numbers of Pixels different from the background
column
0
10
20
30
40
50
60
0
20
40
60
80
100
120

Figure 2: Segmentation Model (Left: Logic of the Model; Right: Sample with its Pixel Density)

We used LeNet-5[10] to train our generated single letter with the only change in the number of output from the original model, 62 classes. Our training size is 62 classes with 1000 per class, with image size 64*64. We resize the image to 28*28 and convert the image from GDB to grayscale. We formed an algorithm for our segmentation process. Our input image contains four characters, and the size of image is 128*64.

We convert the image from GDB to grayscale, and then we count the total number of pixels that are different from the background pixels in each column and sum up every 16 columns' count into a trunk. We set a threshold value from experience. For each trunk which is less than the threshold(900), we choose the min_column (column in that trunk with lowest counts for not-background pixels) as segmentation border for one piece. We processed the segmented pieces with the same method again. If the pieces are larger than 4, we combine two consecutive pieces with minimum sum of trunk values until we get four pieces. If the pieces are less than 4, we segment the piece with largest trunk value evenly along the horizontal axis. Finally, we get four pieces of single character for each image.

4.4 VGG-16 Revised (RGB and Gray Scale) & Simple Convolutional Neural Network

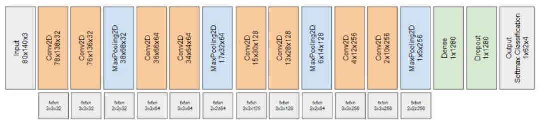


Figure 3: Architecture of VGG-16 Revised Network

We used a revised VGG-16[9] network for our test under different datasets (image in RGB and grayscale) in separate training processes. This CNN concatenates four stacks of Conv2D(ReLu) with max pooling layers and 2 fully connected layer(ReLu) by the end to generate a 1 x 248 dimension output as shown in Figure 4. Batch Normalization and dropout are applied as regularization during the processes, and the model is rely on Adadelta Optimizer from Keras[14]. Also, we tested a performance by deducting two stacks of Conv2D+MaxPooling, and called it Simple CNN.

4.5 ResNet

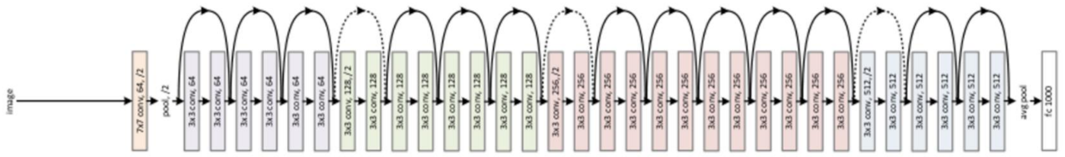


Figure 4: Architecture of ResNet50 [15]

We tried to use a ResNet-50[8] to compare with VGG-16. One of the biggest advantages of the ResNet is while increasing network depth, we can also get a fast training and high accuracy. Since ResNet requires a squre input, we change our input size into 140x140 instead of 140x80. The results of comparison amoung different networks are shown in Figure 5.

5 Experiments/Results/Discussion

	FC NN	LeNet + Segmentation	Simple CNN	VGG-16 Revised	ResNet50
Mini Batch Size	32	128	32	32	32
Dropout	0.7	1.0	0.25	0.25	0.7
Learning Rate	0.001	0.001	0.001	0.001	0.001
Optimizer	Adam	SGD	Adadelta	Adadelta	Adadelta
Epoch	60	50	60	60	60
Loss Function	CE	CE	CE	CE	CE
Test Accuracy	2.92%	< 1.00%	46.88%	85.16%	81.29%

Table 3: Hyperparameters and Results for Each Model

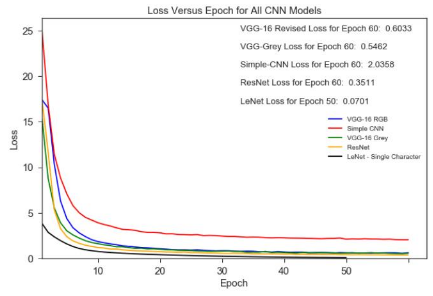
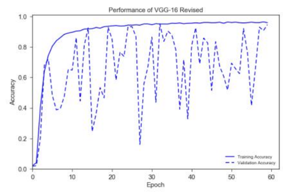
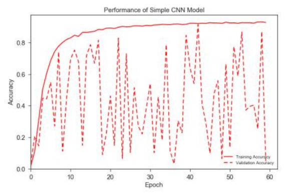


Figure 5: Loss for All CNN Models

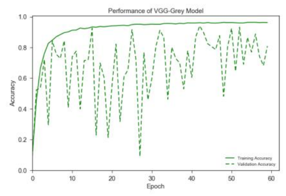
In our experiment, we first used our own segmentation algorithm to process the images. Since the segmentation result can be only checked manually, we test 10 images(40 characters). The segmentation accuracy for these 10 images is 86%(single character). Then we use our generated single letter data(64*64) to train model and get 99.31% accuracy and 0.0701 loss after 50 epochs. However, when we use our single character data extracted from the our formed segmentation, the accuracy is lower than 20%. The test accuracy is lower than 1%. So we decide to jump to next approach. The reason for the poor performance can be attributed to the resize process and segmentation algorithm. After resizing them to same size as input, the deformation for each single



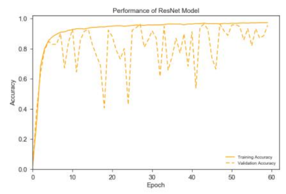
(a) Accuracy of Revised VGG-16



(b) Accuracy of Simple CNN Model



(c) Accuracy of Revised VGG-16 with grayscale Input



(d) Accuracy of Simple CNN Model

Figure 6: Examples of Data with Different Difficulties

character is different. Then it is hard to get high accuracy from trained model.Also, the formed algorithm is not perfect for largely overlapped and not perfect for images with many arcs and noises.

For the end-to-end approach, we built a basic CNN first and modify the VGG-16 and ResNet-50[16]. In Figure 7b, the training accuracy reaches 90% after 60 epochs of training. But the validation accuracy fluctuates much during the whole training process. In Figure 7a, the training accuracy increases to 90% in 10 epochs and finally reaches 95% after 60 epochs. The fluctuation of validation accuracy relatively decreases and becomes smaller than that of a simple CNN, as VGG structure can extract more feature from the input image. Because of the overlapping and rotation of characters, it's consequent to deepen the network to explore more hidden and undeteced features. We get a highly improved performance on our revised VGG-16 with grey sacle image as inputs. The training accuracy approaches to nearly 98% and the validation accuracy tends to become steady after 40 epochs. We use a grey scale input because of the low importance of color feature in a CAPTCHA. Then we use ResNet-50 to train on the RGB dataset. The loss decreases rapidly in the first 10 epochs. And the validation accuracy becomes steady after 30 epochs and reaches 90% in the end. For the fluctuation of accuracy during validation, we could shuffle the data after each epoch to reduce the fluctuation. And because of the characteristic of Adadelta, the validation accuracy tends to fluctuate when it is converging. We could try to use Adam to reduce the fluctuation. The gap between the testing accuracy and training accuracy may be caused by the limited scale of our dataset which could be improved in the future.

6 Conclusion

In this project, we used different models of neural network to break the CAPTCHA and compare the performance of those models. We firsted did the recognition of single character and try to use segmentation to process the multi-character CAPTCHA. Then we used end-to-end training based on VGG-16 and ResNet-50 to get a better performance with the best test accuracy of 86.28%. Considering the room for improvement of our result, We thorougly analyzed the error and would improve our work in the future.

7 Future Work

We will first increase the size of our dataset to tackle the overfitting problem in our task. In addition, We will conduct transfer learning for other type of CAPTCHAs, and see how the transfer learning can support in character detection. We also plan to study the value of training models for nature character detection prepared with artificial occluded data. Furthermore, we will try to implement some data preprocessing to remove the noise dots and curves from the image.

8 Contributions

Yutong Zhang Developing segmentation algorithm and doing single character classification

Jiaying Li: Building datasets, preprocessing data and data analysis of the experiment.

Haotian Sun: Building basic CNN, modifying VGG-16 & ResNet, models training and testing.

References

[1] Stark, F., Hazırbas, C., Triebel, R., Cremers, D. (2015). Captcha recognition with active deep learning. In GCPR Workshop on New Challenges in Neural Computation (Vol. 10).

[2] Xu, D., Wang, B., Du, X., Zhu, X., Yu, X., Liu, J. (2019). Verification Code Recognition Based on Active and Deep Learning. arXiv preprint arXiv:1902.04401.

[3] Wang, Y., Lu, M. (2018). An optimized system to solve text-based CAPTCHA. arXiv preprint arXiv:1806.07202.

[4] Goodfellow, I., Bulatov, Y., Ibarz, I., Arnoud, S., Shet, V.(2013). Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. arXiv: 1312.6082.

[5]Jeff Yan, Ahmad Salah El Ahmad. 2008. "A Low-cost Attack on a Microsoft CAPTCHA", the 15th ACM conference on Computer and communications security, 543-554

[6] Geitgey, A., (2017). How to break a CAPTCHA system in 15 minutes with Machine Learning. Url: https://medium.com/@ageitgey/how-to-break-a-captcha-system-in-15-minutes-with-machine-learning-dbebb035a710

[7]captcha. Python Package. https://pypi.org/project/captcha/.

[8]He, K., Zhang, X., Ren, S., Sun, J., (2015). Deep Residual Learning for Image Recognition. arXiv:1512.03385.

[9]Simonyan, K., Zisserman, A., (2015).Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556

[10]Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-based learning applied to document recognition", Proc. IEEE, vol. 86, no. 11, pp. 2278-2324, 1998.

[11]Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[12]Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

[13]Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science Engineering, 13, 22-30 (2011)

[14]Chollet, Francois, Keras, GitHub, GitHub Repository: https://github.com/fchollet/keras

[15]https://www.kaggle.com/keras/resnet50

[16]https://github.com/ypwhs/captcha_break

GitHub Code: https://github.com/haotiansun/keras_captcha

https://github.com/zhangyt1234/breaking_captcha