Captcha Recognition using CNN

Yichen Wang
Stanford University
yicwang@stanford.edu

Abstract

An automated captcha image recognition application trained with deep learning convolution neural network.

Introduction

Captcha is invented for ages, mainly for improving the security of the applications. It is delivered in the format of image (most common) or audio, which only humans are supposed to understand and provide the correct response. For example, distortion and rotation of characters, adding background noises, etc. are giving almost zero challenge to humans, but will confuse traditional OCR or computer vision algorithm to recognize them correctly. With the power of deep learnings, above technics can be easily cracked. The model will give very high accuracy when predicting the captchas, and the application implemented this model can be used as a demo for demonstrating the weakness of traditional captcha technologies.

Dataset and Features

There are tons of formats and styles of captcha images, but at the end of the day, they are all generated from "programmable" ways. i.e. even they look very differently from ones to ones, the differences for generating those images are maybe just couple of parameters. From a result of background researches, there are different libraries are used by different applications in Python, Java, Javascript etc. For demonstration purposes, lepture/captcha (https://github.com/lepture/captcha) will be the library used in this project.

Below are a few captcha examples that used in some companies:

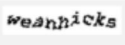
Scheme	Website(s)	Example	Security Features		Excluded Characters
			Anti-segmentation	Anti-recognition	
Wikipedia	wikipedia.org		Overlapping characters, Enligh letters	Rotation, distortion, waving	–
Microsoft	{live, bing, miscosoft}.com {office, linkedin}.com		Overlapping characters, solid background	Different font styles, varied font sizes, rotation, waving	0, 1, 5, D, G, I, Q, U
eBay	ebay.com		Overlapping characters, Only arabic numerals	Character rotating, distortion and waving	–
Baidu	{baidu, qq}.com		Occluding lines, overlapping, only Enligh letters	Varied font size, color, rotation, disortion and waving	Z
Google	google.{com,co.in,co.jp, co.uk,ru,com.br,fr com.hk,it,ca,es,com.mx} youtube.com		Overlapping characters, Enligh letters	Varied font sizes & color, rotation, disortion, waving	–
Alipay	{alipay, tmall}.com {taobao, login.tmall}.com alipayexpress.com		English letters and arabic numerals, overlapping characters	Rotation and distortion	0, 1, I, L, O
JD	jd.com		English letters and arabic numerals, overlapping characters	Rotation and distortion	0, 1, 2, 7, 9, D, G, I, J, L, O, P, Q, Z
Qihu360	360.cn		English letters and arabic numerals, overlapping characters	Varied font sizes, rotation and distortion	0, I, L, O, T, i, l, o, t, q
Sina	sina.cn		English letters and arabic numerals, overlapping characters	Rotation, distortion, waving	1, 9, 0, D, I, J, L, O, T, i, j, l, o, t, g, r
Weibo	weibo.cn		English letters and arabic numerals, overlapping characters, occluding lines	Rotation and distortion	0, 1, 5, D, G, I, Q, U
Sohu	sohu.com		Complex background, occluding lines, and overlapping	Varied font size, color and rotation	0, 1, i, l, o, z

Table 1: Text-based captcha schemes tested in our experiments.

As we are using the same library for both developing and testing, so the distribution among trainset/devset/testset are all same. Moreover, given they are all from the same source, only two datasets division is needed: trainset/testset.

The character sets allowed is small alphabets, big alphabets, and digits, counts in total of 62. The output will be total character sets allowed times the length if represented in one-hot format. The representation of the training data X are [?, 60, 160, 3], which representing [samples, height, width, channels]. The representation of training data Y is using a one-hot representation, [?, 62 * max_length]. The most common length 4 will be used in this project.

Methods & Discussions

A typical convolutional neural network architecture is used for this model.

There are a lot of hyperparameters can be tuned, and different values are explored:

1. Number of Convolution Layers
   Tried with 2 and 3 convolution layers, they are giving about the same performance while 3 convolution layers approach just take more time to train. Hence, it turns out for this type of task, 2 convolution layers are sufficient;
2. Number of Fully Connected Layers

Tried with 1 and 2 FC layers, 2 FC layers converged faster, but with enough epochs, the performance also turns out to be the same;

3. Kernel sizes used in Conv2D
   When increasing kernel sizes from 3*3 to 5*5, the training time for every epoch almost tripled, while gives about the same performance.
4. Learning Rate
   Tried with 0.009 and 0.001, seems 0.001 will be a more reasonable approach.

There are other parameters that are also being played, but those are not really affecting the final results like:

1. Minibatch size
2. Epochs (when greater than 100)
3. Training set sizes (when greater than 4096)

Besides above parameters, below are also being noticed as important things in the application:

1. RGB Image VS Grey Image
   The application is developed from dealing with RGB images, hence all datasets and convolutional weights/kernels are prepared to be compatible with 3 channels. However, inspired by ref[2], it is not necessary to deal with 3 channels most of the time. When image is being greyed, the key information will not be lost, but the whole training is expected to consume much less resources and converge much faster.
2. File IO
   The draft version of the application was written to generate entire trainset/devset up in front, and we read every image from filesystems and digitized it to numpy arrays during training when needed. From that revision of the code, the training speed was really slow. Spent quite some time debugging on it, it turns out the file I/O were wasting a lot of time.

   The fix is to trade memory to time. Since there is a programmable API for generating the data, instead of writing to and reading from file, all images in the format of numpy array will be stored in memory up in front. This will bypass the heavy file IO operation, and speeds up the training significantly.

Results

Train Accuracy: 100%
Test Accuracy: 99%

Conclusion/Future Work

So, with the power of deep learning, the accuracy of cracking captcha generated by lepture/captcha is pretty high. It is also strongly suggesting that the same performance will be seen with captchas generated from other libraries as well.

During the implementation of this application, a lot of API changes are being seen in the TensorFlow community. Quite some APIs mentioned in the deep learning courses and assignments are outdated and marked for deprecation. Some are easy to figure out, while some

are going to be non back-compatible changes. Given that the newer version of TensorFlow is integrating Keras, defining a model using Keras style will be needed to be adapted to when moving forward.

More steps ahead, with a known model working, it is nicer to have a Rest API server running, so application will have better interface to be utilized.

Code

https://github.com/yicwang/captcha_recog

References

[1] https://github.com/lepture/captcha
[2] https://www.zdnet.com/article/new-machine-learning-algorithm-breaks-text-captchas-easier-than-ever/