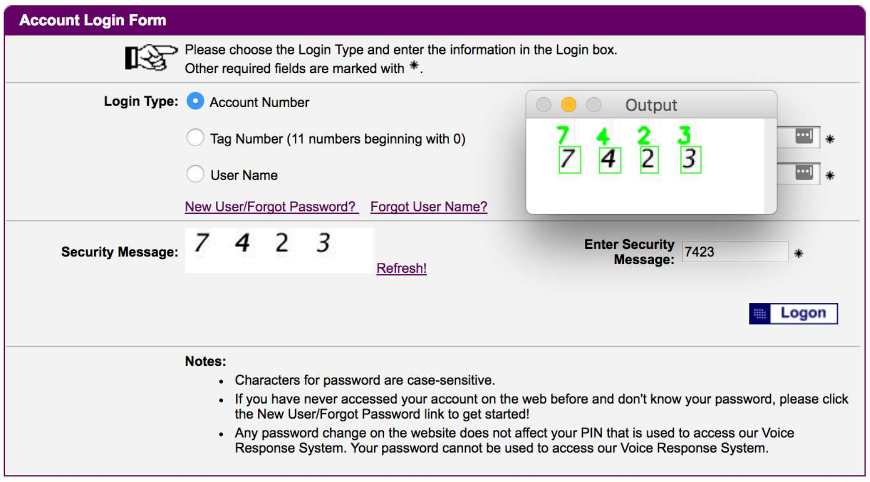
How to break a CAPTCHA system in 15 minutes with Machine Learning

Let's hack the world's most popular Wordpress CAPTCHA Plug-in

Adam Geitgey Follow
Dec 13, 2017 · 8 min read

Everyone hates CAPTCHAs—those annoying images that contain text you have to type in before you can access a website. CAPTCHAs were designed to prevent computers from automatically filling out forms by verifying that you are a real person. But with the rise of deep learning and computer vision, they can now often be defeated easily.

I've been reading the excellent book ***Deep Learning for Computer Vision with Python*** by Adrian Rosebrock. In the book, Adrian walks through how he bypassed the CAPTCHA on the E-ZPass New York website using machine learning:



Account Login Form

Please choose the Login Type and enter the information in the Login box.
Other required fields are marked with *.

Login Type: [✓ Account Number] Tag Number (11 numbers beginning with 0) User Name

New User/Forgot Password? Forgot User Name?

Security Message: 7 4 2 3 Refresh!

Enter Security Message: 7423 *

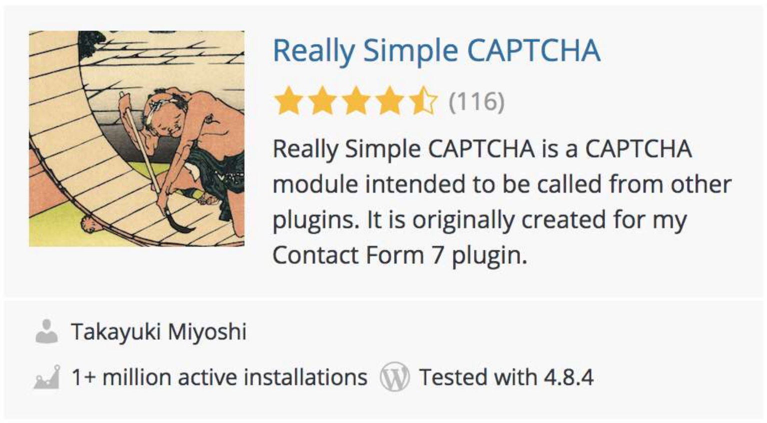
Logon

Notes:
- Characters for password are case-sensitive.
- If you have never accessed your account on the web before and don't know your password, please click the New User/Forgot Password link to get started!
- Any password change on the website does not affect your PIN that is used to access our Voice Response System. Your password cannot be used to access our Voice Response System.

Adrian didn't have access to the source code of the application generating the CAPTCHA image. To break the system, he had to download hundreds of example images and manually solve them to train his system.

But what if we want to break an open source CAPTCHA system where we do have access to the source code?

I went to the WordPress.org Plugin Registry and searched for "captcha". The top result is called "Really Simple CAPTCHA" and has over 1 million active installations:


Really Simple CAPTCHA

(116)

Really Simple CAPTCHA is a CAPTCHA module intended to be called from other plugins. It is originally created for my Contact Form 7 plugin.

Takayuki Miyoshi

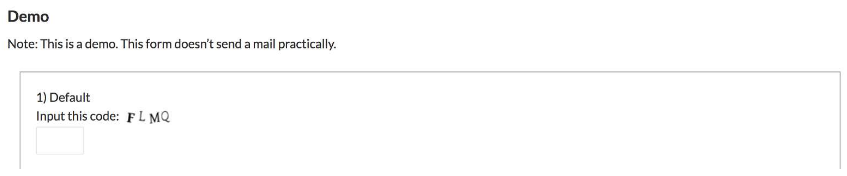
1+ million active installations Tested with 4.8.4

And best of all, it comes with source! Since we'll have the source code that generates the CAPTCHAs, this should be pretty easy to break. To make things a little more challenging, let's give ourself a time limit. Can we fully break this CAPTCHA system in less than 15 minutes? Let's try it!

Important note: *This is in no way a criticism of the 'Really Simple CAPTCHA' plugin or its author. The plugin author himself says that it's not secure anymore and recommends that you use something else. This is just meant as a fun and quick technical challenge. But if you are one of the remaining 1+ million users, maybe you should switch to something else :)*

The Challenge

To form a plan of attack, let's see what kinds of images Really Simple CAPTCHA generates. On the demo site, we see this:


Demo

Note: This is a demo. This form doesn't send a mail practically.

1) Default

Input this code: F L M Q

A demo CAPTCHA image

Ok, so the CAPTCHA images seem to be four letters. Let's verify that in the PHP source code:

```
        public function __construct() {
                /* Characters available in images */
                $this->chars = 'ABCDEFGHJKLMNPQRSTUVWXYZ234

                /* Length of a word in an image */
                $this->char_length = 4;

                /* Array of fonts. Randomly picked up per c
                $this->fonts = array(
                        dirname(   FILE   ) . '/gentium/Gen
```

Yep, it generates 4-letter CAPTCHAs using a random mix of four different fonts. And we can see that it never uses "O" or "I" in the codes to avoid user confusion. That leaves us with a total of 32 possible letters and numbers that we need to recognize. No problem!

Time elapsed so far: 2 minutes

Our Toolset

Before we go any further, let's mention the tools that we'll use to solve this problem:

Python 3

Python is a fun programming language with great libraries for machine learning and computer vision.

OpenCV

OpenCV is a popular framework for computer vision and image processing. We'll use OpenCV to process the CAPTCHA images. It has a Python API so we can use it directly from Python.

Keras

Keras is a deep learning framework written in Python. It makes it easy to define, train and use deep neural networks with minimal coding.

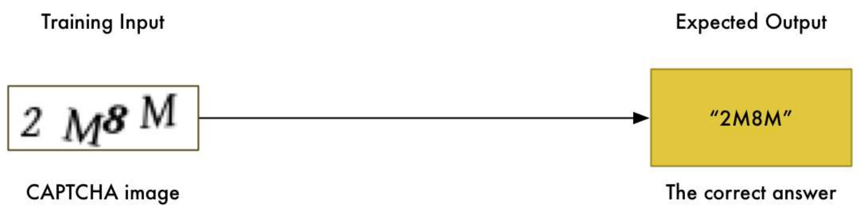
TensorFlow

TensorFlow is Google's library for machine learning. We'll be coding in Keras, but Keras doesn't actually implement the neural network logic itself. Instead, it uses Google's TensorFlow library behind the scenes to do the heavy lifting.

Ok, back to the challenge!

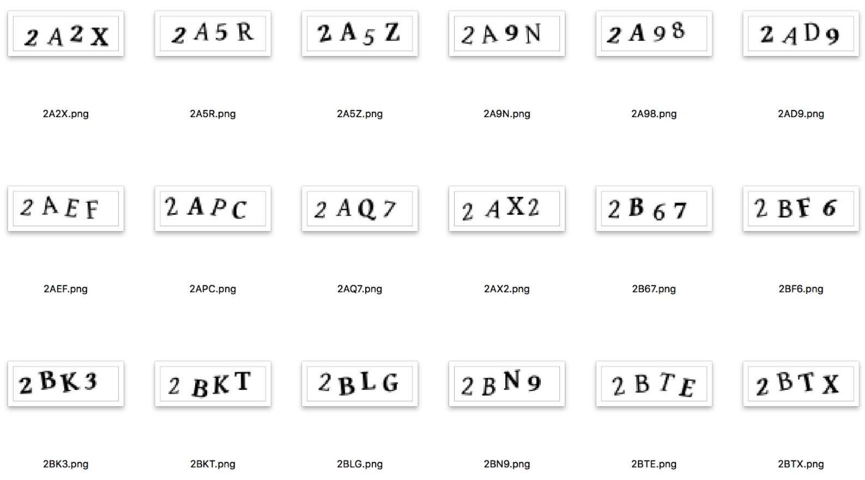
Creating our Dataset

To train any machine learning system, we need training data. To break a CAPTCHA system, we want training data that looks like this:



Since we have the source code to the WordPress plug-in, we can modify it to save out 10,000 CAPTCHA images along with the expected answer for each image.

After a couple of minutes of hacking on the code and adding a simple *'for'* loop, I had a folder with training data—10,000 PNG files with the correct answer for each as the filename:

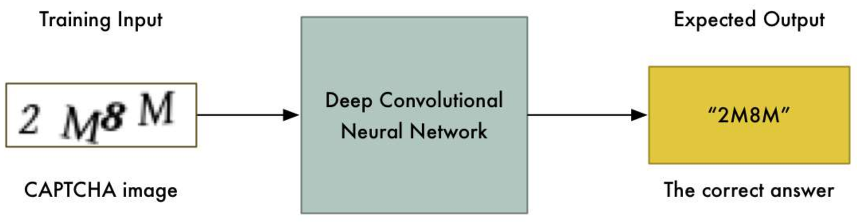


This is the only part where I won't give you working example code. We're doing this for education and I don't want you to actually go out and spam real WordPress sites. However I will give you the 10,000 images I generated at the end so that you can replicate my results.

Time elapsed so far: 5 minutes

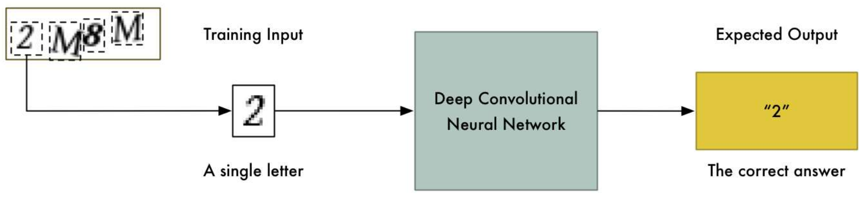
Simplifying the Problem

Now that we have our training data, we *could* use it directly to train a neural network:

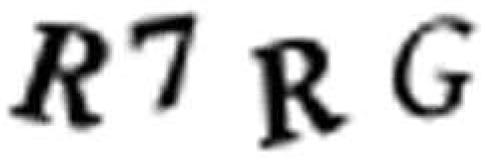


With enough training data, this approach might even work—but we can make the problem a lot simpler to solve. The simpler the problem, the less training data and the less computational power we'll need to solve it. We've only got 15 minutes after all!

Luckily the CAPTCHA images are always made up of only four letters. If we can somehow split the image apart so that that each letter is a separate image, then we only have to train the neural network to recognize a single letter at a time:



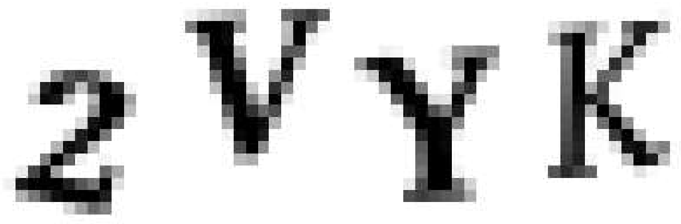
I don't have time to go through 10,000 training images and manually split them up into separate images in Photoshop. That would take days and I've only got 10 minutes left. And we can't just split the images into four equal-size chunks because the CAPTCHA randomly places the letters in different horizontal locations to prevent that:



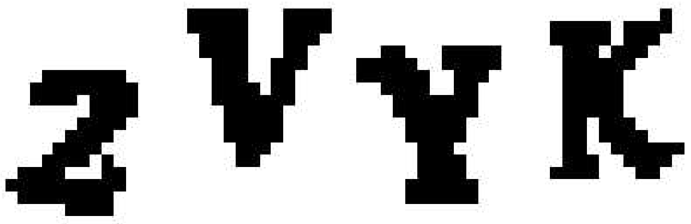
The letters in each image are randomly placed to make it a little more difficult to split apart the image.

Luckily, we can still automate this. In image processing, we often need to detect "blobs" of pixels that have the same color. The boundaries around those continuous pixels blobs are called *contours*. OpenCV has a built-in *findContours()* function that we can use to detect these continuous regions.

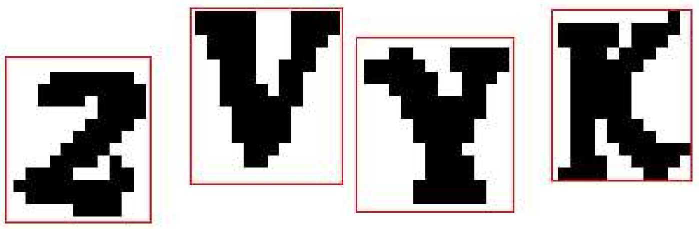
So we'll start with a raw CAPTCHA image:



And then we'll convert the image into pure black and white (this is called t*hresholding*) so that it will be easy to find the continuous regions:

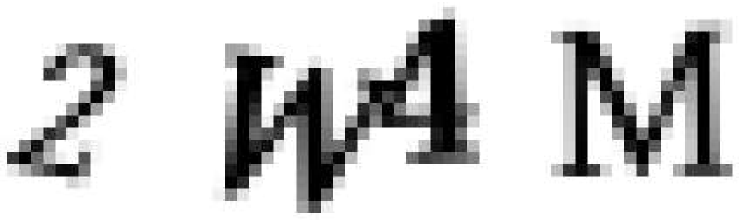


Next, we'll use OpenCV's *findContours()* function to detect the separate parts of the image that contain continuous blobs of pixels of the same color:

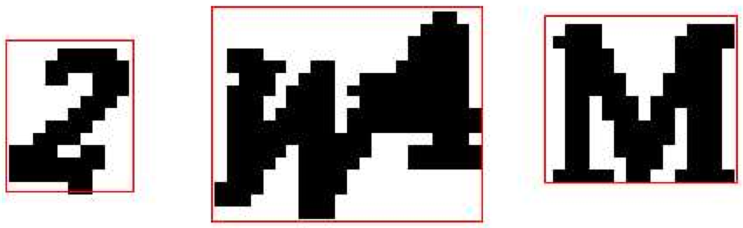


Then it's just a simple matter of saving each region out as a separate image file. And since we know each image should contain four letters from left-to-right, we can use that knowledge to label the letters as we save them. As long as we save them out in that order, we should be saving each image letter with the proper letter name.

But wait—I see a problem! Sometimes the CAPTCHAs have overlapping letters like this:

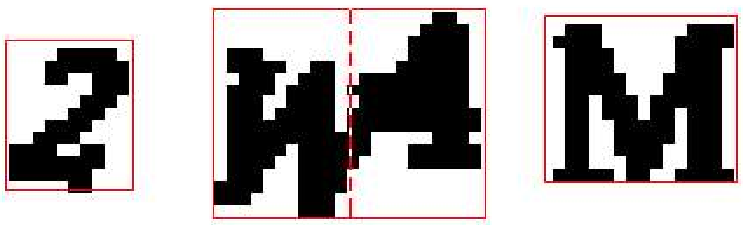


That means that we'll end up extracting regions that mash together two letters as one region:



If we don't handle this problem, we'll end up creating bad training data. We need to fix this so that we don't accidentally teach the machine to recognize those two squashed-together letters as one letter.

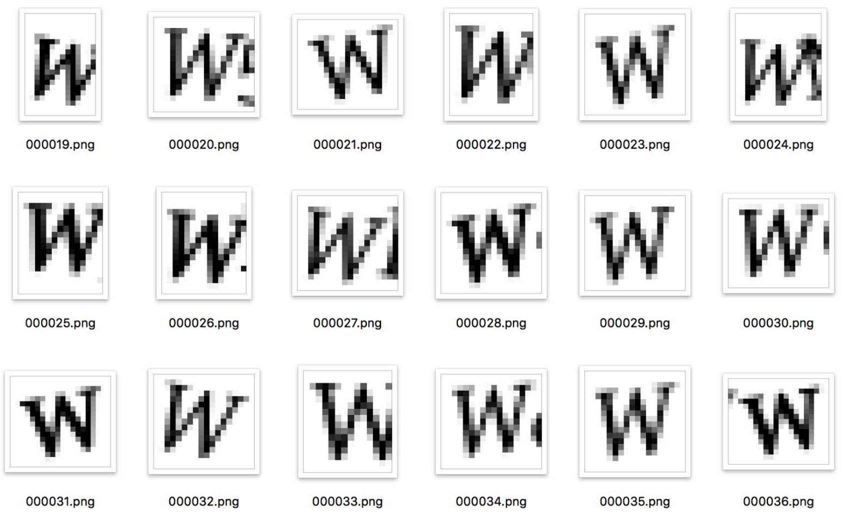
A simple hack here is to say that if a single contour area is a lot wider than it is tall, that means we probably have two letters squished together. In that case, we can just split the conjoined letter in half down the middle and treat it as two separate letters:



We'll split any regions that are much wider than they are tall in half and treat it as two letters. It's hacky, but it works fine for these CAPTCHAs.

Now that we have a way to extract individual letters, let's run it across all the CAPTCHA images we have. The goal is to collect different variations of each letter. We can save each letter in it's own folder to keep things organized.

Here's a picture of what my "W" folder looked like after I extracted all the letters:



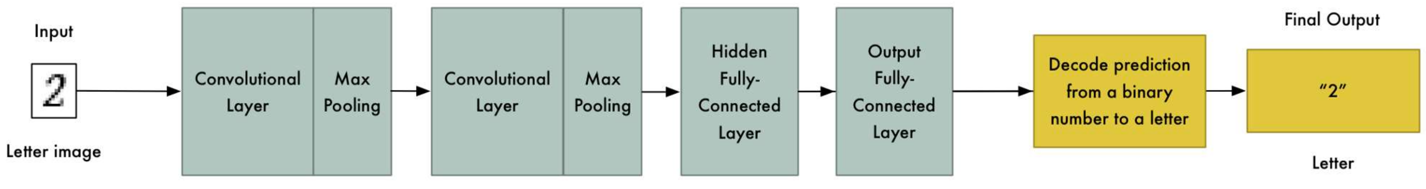
Some of the "W" letters extracted from our 10,000 CAPTCHA images. I ended up with 1,147 different "W" images.

Time elapsed so far: 10 minutes

Building and Training the Neural Network

Since we only need to recognize images of single letters and numbers, we don't need a very complex neural network architecture. Recognizing letters is a much easier problem than recognizing complex images like pictures like cats and dogs.

We'll use a simple convolutional neural network architecture with two convolutional layers and two fully-connected layers:


Input
2
Letter image
Convolutional Layer
Max Pooling
Convolutional Layer
Max Pooling
Hidden Fully-Connected Layer
Output Fully-Connected Layer
Decode prediction from a binary number to a letter
Final Output
"2"
Letter

If you want to know more about how convolutional neural networks work and why they are ideal for image recognition, check out Adrian's book or my previous article.

Defining this neural network architecture only takes a few lines of code using Keras:

```
# Build the neural network!
model = Sequential()

# First convolutional layer with max pooling
model.add(Conv2D(20, (5, 5), padding="same", input_shape=(
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Second convolutional layer with max pooling
model.add(Conv2D(50, (5, 5), padding="same", activation="r
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Hidden layer with 500 nodes
model.add(Flatten())
```

Now we can train it!

```
# Train the neural network
model.fit(X_train, Y_train, validation_data=(X_test, Y_test
```

After 10 passes over the training data set, we hit nearly 100% accuracy. At this point, we should be able to automatically bypass this CAPTCHA whenever we want! We did it!

Time elapsed: 15 minutes (whew!)

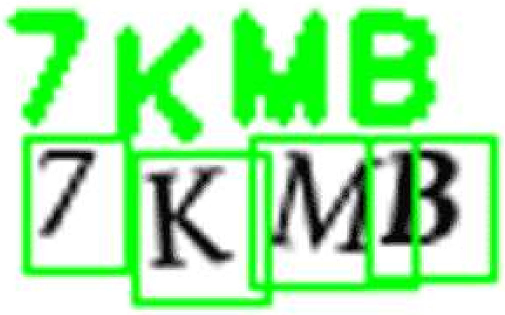
Using the Trained Model to Solve CAPTCHAs

Now that we have a trained neural network, using it to break a real CAPTCHA is pretty simple:

1. Grab a real CAPTCHA image from a website that uses this WordPress plugin.
2. Break up the CAPTCHA image into four separate letter images using the same approach we used to create the training dataset.

3. Ask our neural network to make a separate prediction for each letter image.
4. Use the four predicted letters as the answer to the CAPTCHA.
5. Hijinks ensue!

Here's how our model looks decoding real CAPTCHAs:



Or from the command line:

```
Adams-MacBook-Pro:captcha ageitgey$ python3 solve_captchas_with_model.py
```

Try it out!

If you want to try this yourself, you can grab the code here. It includes the 10,000 sample images and all the code for each step in this article. Check out the included README.md file for instructions on how to run it.

But if you want to learn what every line of the code does, I highly recommend grabbing a copy of ***Deep Learning for Computer Vision with Python***. It goes into a lot more detail and has tons of detailed examples. It's the only book I've seen so far that covers both *how things* *work* and how to *actually use them in the real world to solve difficult problems*. Check it out!

. . .

If you liked this article, consider signing up for my Machine Learning is Fun! newsletter:

Sign up for the Machine Learning is Fun! Newsletter

Email

Sign up

☐ I agree to leave Medium and submit this information, which will be collected and used according to Upscribe's privacy policy.

You can also follow me on Twitter at @ageitgey, email me directly or find me on linkedin. I'd love to hear from you if I can help you or your team with machine learning.