I AM A ROBOT: SOLVING CAPTCHA WITH DEEP LEARNING

A PREPRINT

Jonathon Howland
Tippie College of Business Analytics
University of Iowa
Iowa City, IA
`jonathon-howland@uiowa.edu`

May 18, 2021

ABSTRACT

CAPTCHA puzzles are designed to be difficult for computers to solve. However, using a relatively simple (15 layers) convolutional neural network, it is possible to solve CAPTCHA's that require the identification of a five-character string with an accuracy of 0.987. These CAPTCHA's were all generated in the same manner and have similar characteristics, but this accuracy is still likely above a human's.

1 Introduction

We've all seen them: the little boxes you had to check on websites stating that "I'm not a robot". Then, you had to look at some jumbled, scribbled-on, multi-colored characters and type what they said. And, honestly, I failed sometimes. Fortunately, we don't see those very often anymore, and there's a reason: the robots are smart enough to solve them. Which is impressive, considering the fact that I'm a human and I have trouble with them sometimes.

Those sets of characters are known as CAPTCHAs. CAPTCHA stands for "Completely Automated Public Turing test to tell Computers and Humans Apart". It's a bit contrived, but it gets the point across. For those who are unfamiliar with it, a Turing Test is a general term for a test that shows a computer has intelligence comparable to a human's in a given task. These were designed to be difficult for a machine learning model to solve. In fact, that is the entire point of their existence. [1]

Over time, however, we've begun seeing fewer CAPTCHAs. Instead, they are being replaced with image recognition tests. The reason is that with the advent of deep learning, it is possible to train a model that can decipher these CAPTCHAs. In general, it is still a very difficult problem. Characters can be smashed together to appear as one instead of two or more. Lines and patterns can look like extra characters. Some even have multiple colors within each character.

For this particular project, we've made the problem a bit easier. We are only looking at CAPTCHAs with exactly five characters. The images are also all from a similar type of CAPTCHA. They all have five distorted, colored characters on a distorted background. All of the images have markings in across them, similar to if somebody took a pen and drew some scribbles on them (a couple examples can be seen in the "Data" section).

The fact that we know each image has exactly five characters is probably the most helpful thing for us. Our goal is to go through the image and identify each character sequentially. Knowing that there are always five helps in that if there are, for example, two "u"s squished together, we know that they must be not be a "w" (though we may not know if they are two "u"s, a "u" and a "v", etc.).

In order to solve this problem, we will utilize a convolutional neural network to identify the individual characters in the label. Once the characters have been identified, they can be compared to the ground truth. Each CAPTCHA is one image, and the labels are five character strings that we can compare to our results from the neural network.

2 Problem Definition

To be precise, the goal is to identify the five characters contained in a CAPTCHA image, which will then be compared to the five characters contained in the label to determine accuracy. The CAPTCHA images and labels are the only inputs for this project, and the predicted labels are the outputs.

Each CAPTCHA image is a 150x40 color image containing five characters, as well as various other markings. However, to make it easier, we will normalize all of them to 64x64 color images. In order for the machine learning process to run more smoothly, we will also normalize all the RGB values for the pixels to the range [-1, 1], as opposed to their original ranges of [0, 255].

The true labels can be obtained from the file names, which are all in the format of <label>.jpg, where <label> is the five-character string containing the true values of the CAPTCHA. Due to the massive size of the dataset, I will not be manually confirming the accuracy of the labels. They were pre-defined by the original creator of this data, Parsa Sam [3].

These labels needed to be properly encoded, which is a process that will be discussed in detail later. Once the encoding was done, the resized and normalized images were passed into the neural network with the encoded labels.

3 Data

I say that the dataset is massive, although not compared to the likes of ImageNet. The data I am using is comprised of 113,062 CAPTCHA images and labels that were generated by the Captcha Library for PHP, which was created by Grégoire Passault [2]. While this isn't a huge number as far as deep learning goes, it is still far more than I am able to go through by hand.

As I have stated previously, each image contains 5 characters and some markings in the form of irregular curves across the image. From a cursory glance through the files, most, if not all, of the images have a similar format. So, I have selected two examples to show the basic idea of this particular CAPTCHA (Figures 1 and 2).

The images and file names can be fed into my code to extract the corresponding label for each image. Aside from that, most of the cleaning and parsing has already been done for me, which makes this project a lot easier.

I got this data from Kaggle.com, and Parsa Sam, the user who has it posted, has done a very nice job of getting the data into an easy format to feed into TensorFlow [3]. To summarize, the data is in the form of approximately 113,000 JPG image files, with each file's name containing the label in the form <label>.jpg.

However, I was not able to get all 113,000 files into a format that could be easily used by those with access to only the Google Colaboratory notebook. As such, the model was built and trained on only 27,364 files. These files were semi-randomly selected from the original 113,000, and should give a good representation of the total dataset.

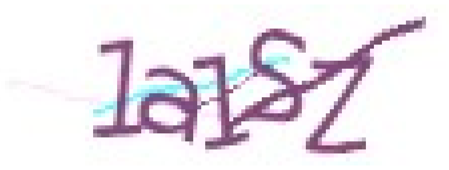

Figure 1: The CAPTCHA image with file name "1a1SZ.jpg". The corresponding label is therefore "1a1SZ", which seems to be the characters this CAPTCHA contains.

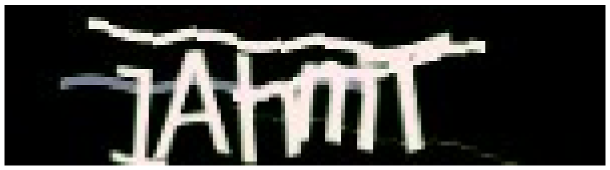

Figure 2: The CAPTCHA image with the label "1AhmT". Again, the label seems to match the characters shown in the image. Though, honestly, I have trouble with this one even as a human.

4 Method

Before any learning could be done with a neural network, the data first had to be in a format that a neural network could handle. The images were relatively simple, and I've already discussed the details of their preprocessing in the Problem Description section. The labels, one the other hand, were a little more complicated. I was also not able to get any of TensorFlow's built-in encoding functions (such as one_hot) to work, so I had to create my own encoding function. As such, I would like to take some time to go through it here, as it is fundamental for understanding the design, training, and testing of the neural network.

The labels were initially in the form of five character strings, as was mentioned earlier. However, the actual labels that the neural network needed to train with are the individual characters themselves, not the full strings. The order is also important ("1Ast2" is not the same as "A1t2s" for this problem). Therefore, one-hot encoding was not going to work.

The solution was an extension of one-hot encoding with multiple values. The logic is as follows: Each character has one of 62 values (26 lowercase English letters, 26 uppercase English letters, and 10 numeric digits). Thus, for each character, we can use one-hot encoding to mark which character it is. This was done using a static list of all possible characters to determine the index of the "one" in the one-hot encoding. After doing this for each character in the string, we concatenate all of the resulting vectors. This results in a 310-dimensional "five-hot" encoded vector as our label.

Now that our labels and images are ready, it's time to discuss the neural network architecture itself. It's fairly simple, as far as deep neural networks go. There are convolution layers (Conv2D), each with 64 filters, a kernel of size (5,5), and padding to keep the outputs the same size as the input. There are also pooling layers (MaxPool2D) using the default settings in TensorFlow. All of the convolution and pooling layers were the same in that they had these structures.

Training and testing sets were both put into batches of size 64. These batches were then fed into a convolution layer, followed by a ReLU activation. This was followed by a second convolution layer, a pooling layer, and another ReLu activation. Then, a dropout layer with a dropout rate of 0.5 was used to prevent overfitting. After that, the process of convolution, ReLU, convolution, pool, and ReLU was repeated.

Next, a flattening layer was used, then a fully connected layer with 512 nodes (I originally tried 1024, but encountered issues with GPU usage). Finally, a fully connected layer with 310 nodes and the sigmoid activation was used for outputs. The sigmoid activation function was chosen to keep the values from exploding and make the cross entropy optimization easier. A diagram of this structure can be seen in Figure 3.

I chose to use the sigmoid activation function over the softmax activation function because of the fact that softmax results in a vector of probabilities (see Equation 1). Since there are five different values in the final label, it didn't make sense to sum them up to a single probability. So, I instead elected to use the sigmoid function (Equation 2), which still normalizes the values between 0 and 1, but without the added restriction that they must sum to 1.

Once the neural network's architecture was built, it was time to train it. I used 80% of the data (22,611 images and labels) to train the model.For the loss function, I used binary cross entropy. I also used ADAM as an optimizer, with a learning rate of $\alpha = 0.001$. With these settings, it took 20 epochs for the training process, after which the loss function began to plateau.

I chose to use binary cross entropy over categorical cross entropy due to have multiple class labels for each image. With a categorical cross entropy, the implementation in TensorFlow is that each individual value is its own class, which is not what I wanted. Instead, the binary cross entropy has only two classes: 1 or 0. This worked better for my application, as my only goal was getting the values at the indices to match (if the true value at an index is 1, I want my predicted value to also be 1, and the same logic applies to 0 as well).

$$softmax(\boldsymbol{x_i}) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{1}$$

$$S(x) = \frac{1}{1+e^{-x}} \tag{2}$$

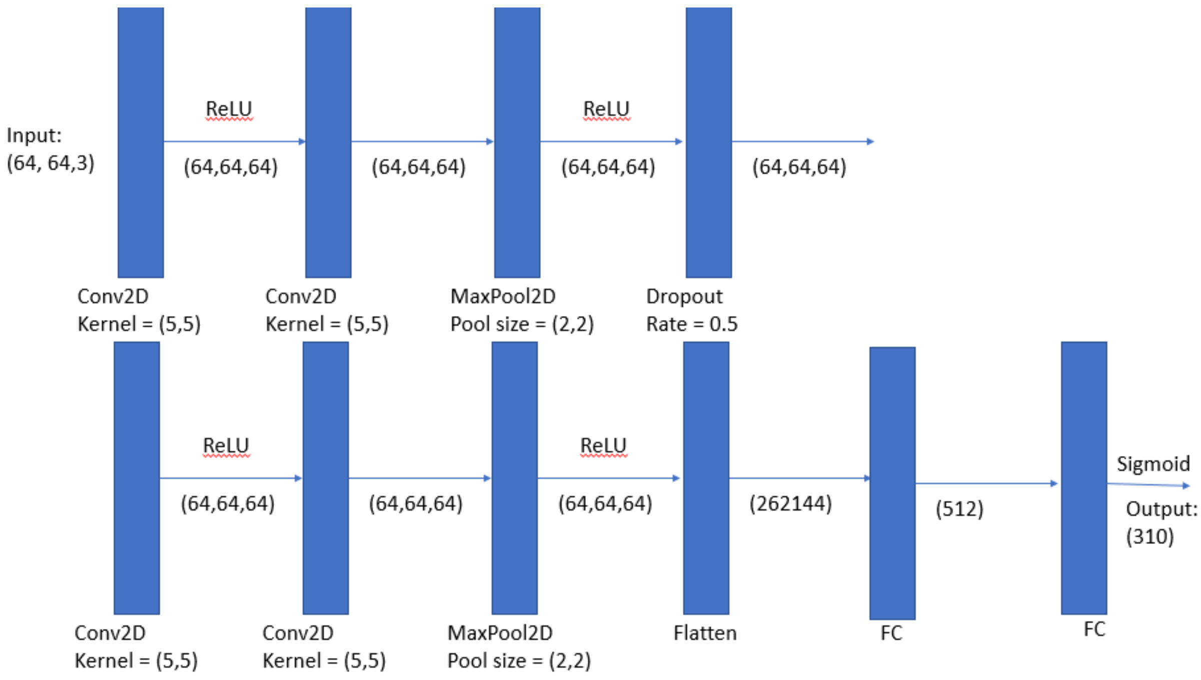

Figure 3: The diagram of the convolutional neural network used in this project. Each box represents a layer, with the label stating the type of layer, and the values representing the dimensions of the inputs and outputs. In the implementation, the ReLU activations are included as separate layers, but here they have been shown above the arrows to save space.

5 Results and Discussion

With all of these settings, I was able to achieve an impressive accuracy. First, however, it is important to discuss how I measured accuracy. I defined accuracy as a correct CAPTCHA prediction. That is to say, all five predicted characters had to match the five true characters, in the correct order. This is different from the accuracy computed by TensorFlow during the training process, so I had to compute it manually.

This model had a validation accuracy of 0.987, which is very impressive considering the strict definition of accuracy I am using. This means that this neural network is able to successfully solve a CAPTCHA puzzle over 98% of the time. While I was unable to find an accurate estimate for human accuracy on this particular type of CAPTCHA, I find it difficult to believe it is substantially above 98%.

There are, however, several facts that make this accuracy less impressive. First, this model was trained on a very specific dataset: colored CAPTCHA images that all contained five characters. Furthermore, all of these CAPTCHA images were generated from the same source code. This means they had very similar patterns and distortions that were designed to confuse humans. With so many examples of the same general type, it's not surprising the model was able to accurately solve the puzzles.

It's also worth noting that the entirety of this architecture is built around the fact that each CAPTCHA contains exactly 5 characters. With new data, this model could likely deal with different types of distortions and markings. However, it can, by design, only predict a five-character string as an output. This means that this model is incapable of accurately solving general CAPTCHA puzzles.

While this sounds crippling, it's neither surprising nor unusual. Neural networks are only capable of predicting data that is similar to what they are built on. It's no different than a neural network designed for image classification being unable to identify an image whose class was not in its training set.

6 Conclusion

It was once thought that CAPTCHA's were unable to be solved by machines. In fact, part of the acronym stands for "to tell Computers and Humans Apart". As has been shown here, that is no longer the case. A relatively simple convolutional neural network was able to solve these puzzles with an accuracy over 98%. This shows that CAPTCHA's are no longer as effective as they used to be. Of course, this is only one type of CAPTCHA. People are constantly creating CAPTCHA's that are more difficult to solve.

While it's great to be excited about the advances in machine learning throughout the past decade, it is also important to remember that no neural network (or any other machine learning algorithm) is perfect or all-knowing. While this model did manage an impressive accuracy on this type of CAPTCHA, this is still only one of the many types of CAPTCHA's that exist. Furthermore, the problem was simplified by having every image contain five characters and similar distortions and markings.

The final takeaway from this project is that while computers are constantly getting smarter, they still have their limitations. If they are trained on a specific type of data, then they will perform very well. However, if they are exposed to new data unlike what they've been trained on, they will likely perform very poorly. It is important to keep these limitations in mind as we continue celebrating the achievements of machine learning.

References

[1] *CAPTCHA - Wikipedia*. 2021. URL: `https://en.wikipedia.org/wiki/CAPTCHA`.

[2] Grégoire Passault. *Captcha Library for PHP*. 2020. URL: `https://github.com/Gregwar/Captcha`.

[3] Parsa Sam. *CAPTCHA Dataset*. 2021. URL: `https://www.kaggle.com/parsasam/captcha-dataset`.