Building a Neural Network for General Captcha Recognition

Liam Lowndes

101041818

COMP 4107: Matthew Holden

April 15, 2023

Introduction

The purpose of this report is to explore the image processing techniques that neural networks utilize to break original text captchas. I hope to show through the demonstration of my code that that the current power of neural networks is significant enough for us to reflect on our current security standards and question whether they are still secure. Captcha stands for "Completely Automated Public Turing test to tell Computers and Humans Apart" and it is evident when looking at modern day security norms that the text captcha is no longer in use. It has instead been replaced with the far more popular image captcha.

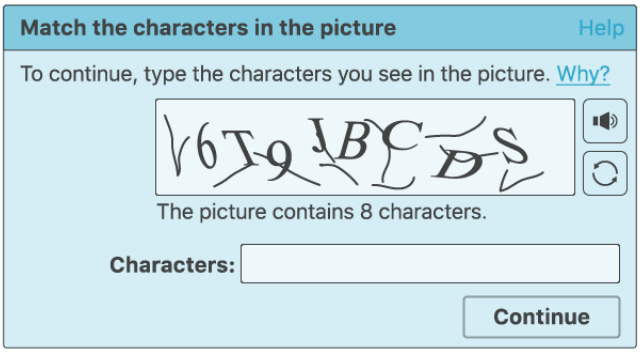


Figure 1.1: (CloudFlare 2022)

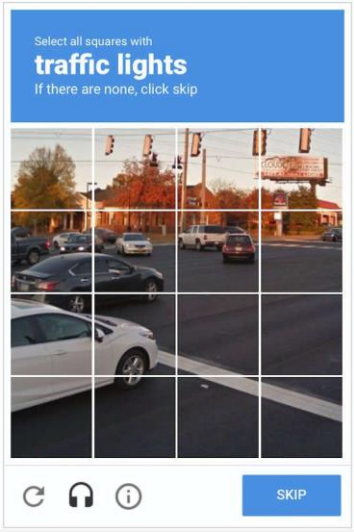


Figure 1.2: (Thompson 2021)

Throughout the course of this report, I will express the relative ease that a neural network can have when trying to learn how to identify the letters in a captcha such as the one in figure 1.1. A popular technique used in tackling this problem is creating a convolutional Neural Network (CNN), as stated in the report *Workshop New Challenges in Neural Computation 2015*

"Conventional methods aim at detecting the text within natural images in two disjoint steps localizing the regions of words or single characters within the image, segmenting and then recognizing them" (Hammer, Martinetz and Villmann 2015)

These CNN's are then trained on datasets that contain many variations on the same type of captcha. This is where an attempt was made to take the technique one step further. While training

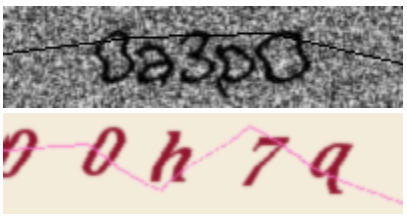


Figure 2

my model, instead of training separate models for each type of captcha I tried training a singular model on two different types of captchas. The two captchas shown in figure 2 are taken from the training dataset. The top captcha is created in greyscale, with one type of font, has wave added to the characters, background noise, Gaussian blur, and a curved line drawn through the center of the image. The bottom captcha is similar except the line, characters, and background all have color. The font is also randomized for each colored image and the background contains less noise.

The type of neural network I used to accomplish my goal is known as a Convolutional Neural Network (CNN) and it is a specifically designed deep learning model that excels at image recognition. Th reason that CNNs are particularly effective is that they are good at breaking an image down into its core patterns such as edges, textures, and complex shapes. The captcha used to be seen to protect website owners from attacks but with this paper I aim to show the relative weakness of this old practice. The specific objective of this report is to demonstrate how the CNN I created can take in and read these two types of captchas with a high degree of success.

Methods

The code that I have implemented can be broken down into four general chunks, which are: the dataset, helper functions, neural network architecture, and the evaluation model. I will walk through each chunk of my code and describe how it achieved my objective of recognizing captchas. First there is the data itself, before the training began, I created 22,000 captchas. The first 20,000 are in the training and validation set and are half colour captchas and half greyscale captchas. The final 2000 captchas are for testing and are also split half colour and half greyscale.

Then there were the helper functions getNames (GN) and oneHotEncode (OHE). The GN helper function is a simple function that reads in the answers to the captchas and returns them as a list. Before each training session there was a quick script ran that retrieved all of the file names of the captchas, which were their respective answers, and output them to a text file. This is the file that GN process and creates the answer list “captchaAnswers” from. Next the OHE function converts

the labels into a format that can be easily processed by the neural network. The format represents each character, which in this case is all lower-case and upper-case letters as well as all ten digits, in a binary vector. The length of this vector in this case is thus 26 + 26 + 10 which is 62. The arbitrary order I chose was lower case letters, then uppercase letters, then digits. So 1 with 61 0's after it would represent the letter "a" and a 1 at the 28th position and a 1 everywhere else would represent "B" and so on. This is what creates a machine-readable interpretation of our labels that we can feed into the neural network.

Figure 3: Architecture of the Captcha Recognition Neural Network Model

#	Layer Type	Output Size	Details
1	Input	50x200x1	
2	Convolutional	50x200x32	3x3 conv, stride=1, padding='same'
3	Batch Norm	50x200x32	
4	Max Pooling	25x100x32	2x2 pool, stride=2, padding='same'
5	Convolutional	25x100x64	3x3 conv, stride=1, padding='same'
6	Batch Norm	25x100x64	
7	Max Pooling	13x50x64	2x2 pool, stride=2, padding='same'
8	Convolutional	13x50x128	3x3 conv, stride=1, padding='same'
9	Batch Norm	13x50x128	
10	Max Pooling	7x25x128	2x2 pool, stride=2, padding='same'
11	Convolutional	7x25x256	3x3 conv, stride=1, padding='same'
12	Batch Norm	7x25x256	
13	Max Pooling	4x13x256	2x2 pool, stride=2, padding='same'
14	Convolutional	4x13x512	3x3 conv, stride=1, padding='same'
15	Batch Norm	4x13x512	
16	Max Pooling	2x7x512	2x2 pool, stride=2, padding='same'
17	Flatten	7168	
18	Dense	1024	L2 regularization=0.001
19	Batch Norm	1024	
20	Dropout	1024	Dropout rate=0.5
21	Dense	310	Soft max activation L2 reg=0.001
22	Reshape	5x62	

Moving onto the model of the neural network that I used to be trained on the dataset. I will break each line from the model down to create some insight on its inner workings. Please refer to the following table in Figure 3 to help with understanding the neural network's general structure. First, we can look at the input layer which has a shape of 50x200x1 which represents a singular captcha that is 50 pixels high, 200 pixels wide, and a singular channel, grayscale. The way in which these images are read is from a singular csv file named captcha.csv. Each row in the file represents one image where each input in the row represents a single pixel. All the color images are converted to a greyscale image before they are put into the csv file. Then we move to the first 2D convolutional layer. It has 32 filters of size 3x3, a Rectified Linear Unit (ReLu) activation function and 'same' padding. The ReLu function is defined as f(x) = max (0, x) and is helpful when you are trying to learn complex patterns and representation from the input data. As pointed out in class this semester by Professor Mathew Holden, the ReLu function is a good choice due to is computational efficiency and avoids the vanishing gradient problem. Finally, this layer (and many layers in this neural network) have 'same' padding which ensures that the output have the same dimensions as the input of the following layer.

Then the batch normalization function is called. This function is used in between layers to stabilize the input values normalizing them. In short this is a technique that helps speed up the training process. Next there is a max pooling layer which uses a 2x2 operation which reduces the number of inputs each layer is working with and allows the model to learn different features of the dataset at different scales. After this the pattern of a convolutional layer, batch normalization, and max pooling happen four more times. The only thing that changes is the convolutional layer uses filter sizes: 64, 128, 256, and 512. These layers continue to help the model learn new features.

After this process is complete there is a call to the flatten function which turns the 3D shape of the data into a 1D vector which can then be processed by the Dense layers. The first dense layer has 1024 neurons with its activation function set to ReLu. It also has L2 regularization which is a way in which we can prevent overfitting to the data. This technique forces the model to prefer smaller weights by adding a penalty the loss function which prevents the weights from getting too large which can lead to overfitting. There is then another batch normalization and a dropout

layer set to 0.5. This will randomly set 50% of the inputs to 0 during training which means that the model is forced to generalize a little more rather than learn than overfit to the specific dataset.

Then there is the final dense layer which has 310 (5 characters chosen * 62 possible characters to chose from) and uses the soft max function to produce a probability distribution over the possible candidates of correct combinations of characters. L2 regularization is also applied to this dense layer. Finally, there is a reshape layer that shapes the output layer to (5, 62) and allows for an output that can be human processed.

The optimizer that I used for this model was the Adaptive Moment Estimation (Adam) optimizer which is used by the model to iteratively minimize the loss function. This optimizer function was chosen over Stochastic Gradient Descent (SGD) because it has a couple of advantages. Adam can usually converge quicker because of its adaptive learning rates, and due to these adaptive learning rates Adam is better at handling noise which was important in the understanding of noisy captchas data.

Finally, I implemented an early stopping function to prevent overfitting and chose categorical cross-entropy as my loss function. The early stopping function is a way in which you can monitor the loss of your model and stop the training process when it stops improving. It was set to monitor the loss on the validation set. It also had a patience of five which meant that it waited for five epochs of no improvement before it stopped training. I then chose categorical cross-entropy as the loss function because it is good for multi-class classification. Since my model had many possible outputs that were one-hot encoded, this seemed like the proper fit.

These were the methods I used to achieve the success that is described in the results section. The techniques aren't perfect however and I will now discuss then limitations of the specific implementation that I have used. The first limitation came with the use of convolutional layers and batch normalization which adds a lot of computational complexity. The size of the convolutional filters and number of convolutional and batch normalization layers is partially what lead to why the training of each epoch for this model took around 4 minutes. The model was set to train for 100 epochs, but the early stopping function was called at the 22nd epoch. This still resulted in an hour and a half of training.

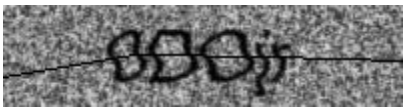
In addition to computational complexity there was also a possibility of overfitting and the need for high memory requirements when it came to fully connected dense layers. The L2 regularization was used to help mitigate some of the overfitting risk but the dense layer having many parameters is one possible cause of some overfitting to the dataset. It is also important to point out that with a fully connected layer and many parameters you are going to have higher memory requirements and a slower runtime. This was also a factor when running this model on limited hardware. Overall, however the model was able to be trained, on a laptop, in a reasonable amount of time and was able to get decent results which will be explained in the next section.

Results

Training the model on my laptop to roughly an hour and a half and resulted in a model that had good performance. The performance metric can be broken down into two categories: number of correct answers, and number of correct characters. A correct answer is when the model can predict the captcha completely correctly. When ran on the test dataset the model was able to achieve 1400 correct guesses out of the 2000 images, meaning that it had a 70% success rate. When diving in deeper to the results however you can see that for the wrong predictions the model is getting 3 to 4 out of the 5 characters correct. I believe that it is also helpful to see the total number of characters to model was able to guess correctly out of the 10,000 (2,000 * 5) characters in the test dataset. The result was much more impressive with a success rate of 9,123/10,000 correct character guesses or a 91.23% success rate.

You can see that sometimes the captchas that need to be decoded are just harder than others. In the case of figure 4, the model output the answer “0DDjr” when the correct answer was “0DOjr”. It is understandable to see that one of the things the model struggles with is telling between a zero, a capital “O”, and a capital “D”. This is something that humans also tend to struggle with and goes to show that this model would need more training to be able to make better predictions on the tougher captchas.

Figure 4:



Discussion

When first creating this model, my dataset was comprised only of the greyscale type captcha's, and I observed its obvious limitation. This limitation being that it could only handle this one type of captcha that I created, which would be of little practical use. There are two routes if you wanted to use this method practically, the first of which would be to find the type of captcha you wanted to crack and train the model on a dataset based off that style of captcha. It would be unlikely that you could find a publicly available dataset that was available to train from. You would have to do your best at recreating the kind of captcha you want to train on. The second route is to train the model off multiple different types of captchas and hope that it picks up general features rather than just specific details about the given dataset. This is the process that I began in my code, but the model is now only able to read from the two types of captchas that I generated. When tested against a third style of captcha, shown in figure 5, the total correct complete guesses were 0% and the total correct characters was ~13.36%. This is superior to randomly guessing which would result in around a ~1.6% correct character rate but is still insufficient for the task at hand. What I believe this shows however is that given enough unique types of captchas, I believe that a sufficient general model could be produced.

Figure 5: (FOURNIERP 2019)

The implication of this work shows that neural networks are a threat to traditional security techniques. This technique of training a neural network to crack a certain type of captcha has already been discussed at great length in academia and it is clear to see why companies like Google have moved on to No CAPTCHA. This is a system that combines a background check with a backup image check. If the automatic background check passes, when you click on the "I'm not a robot" widget it will simply bring you to where you need to go. However, if it fails the back-up image test, as seen in figure 1.2, is displayed on the user must verify which images contain a certain object. (Google 2013) However, this will soon be outdated as well. There are many tools, such as Open AI's DALL-E 2 model which can create images based off extremely specific prompts.

It is not unlikely that one of these models could be used to identify traffic lights in the case of figure 1.2, or even quickly create training data for a neural network like mine, so that the No CAPTCHA can be cracked. It is important to think of these things now so that we can shift to more robust security measures and avoid harmful breaches.

As far as adding to this work in the future I would say that it is worthwhile thinking about a "jack-of-all-trades" model versus "master-of-one" type models that I see are popular. I believe it would be interesting to see someone who has access to greater computing power and bigger, more robust datasets, try training a model based at least some-what off my neural network architecture. It would be interesting to experiment to see if the model could learn enough general features so that it could crack captchas of a type it has never seen before. There is also a paper in the journal Mathematical Biosciences and Engineering named "CAPTCHA recognition based on deep convolutional neural network" which points out that:

"CAPTCHA recognition technology can also be applied in the area of license plate recognition, optical character recognition, handwriting recognition and so on." (Jing, et al. 2019)

In conclusion this report was able to show the efficacy of using a neural network to classify the characters within a 5-character captcha. The methods used to create the model as well as the features of the dataset were broken down and the subsequent results were explained. Given a relatively small amount of computing power and time my model was able to achieve a 70% success rate on complete captchas and a 91.23% success rate on correct character recognition. It is of my opinion that given a larger dataset, more computing power, and some updates to the efficiency and efficacy of the neural networks architecture I believe that the accuracy rate can be vastly increased, and the model may even be able to gain generalized knowledge so that it can crack captcha types it has never seen before.

Works Cited

CloudFlare. 2022. *How CAPTCHAs work | What does CAPTCHA mean?* 03 2. Accessed 04 19, 2023. https://www.cloudflare.com/learning/bots/how-captchas-work/.

FOURNIERP. 2019. *CAPTCHA Images.* Accessed 2023. https://www.kaggle.com/datasets/fournierp/captcha-version-2-images.

Google. 2013. *Google reCAPTCHA.* https://www.google.com/recaptcha/about/.

Hammer, Barbara, Thomas Martinetz, and Thomas Villmann. 2015. *Workshop New Challenges in Neural Computation 2015.* MACHINE LEARNING REPORTS.

Jing, Wang, Qin Jiaohua, Xiang Xuyu, Tan Yun, and Pan Nan. 2019. *CAPTCHA recognition based on deep convolutional neural network* . Mathematical Biosciences and Engineering.

Thompson, Clive. 2021. *Why CAPTCHA Pictures Are So Unbearably Depressing.* 08 5. Accessed 04 19, 2023. https://onezero.medium.com/why-captcha-pictures-are-so-unbearably-depressing-20679b8cf84a.