Captcha Solving Using Neural Networks

Isaiah Brandt-Sims and Thunder Keck
Stanford University 2018
isaiahbs@stanford.edu and tkeck@stanford.edu

Abstract

The abstract should consist of 1 paragraph describing the motivation for your paper and a high-level explanation of the methodology you used/results obtained.

As beginners in the field of neural networks, we set out to tackle a project that would develop our understanding, challenge us to use the techniques learned in CS230, and teach us the value of neural networks. We wanted to pick a problem that would not be achievable without neural networks. So, we chose a problem that was made intentionally difficult for computers. We attempted to create a neural network that can determine the solution to captcha images based off of an RGB image. We tried many models, architectures, and languages. While we had varying accuracy across all of the models, we were able to produce a few effective networks. This paper is an overview of our work and what we learned.

Related Work

Some related work we found were github repos by other students working on basic to complex models trying to do image classifying. Some examples of these would be Panagiotis Tigas(https://github.com/ptigas/simple-captcha-solver) who wrote a NN that solves only 16 characters all being lowercase. Another was Jackon Yang(https://github.com/JackonYang/captcha-tensorflow) who inspired us to first start our repo in tensorflow instead of keras. His models only went up to 4 character strings also limiting the number of characters to just numbers for most of his trials except using the whole 62 characters like us but for only 2 characters in a given image. Jackon did not list the results of his model so we couldn't directly compare our results to his.

There has been a lot of previous work done in this area but we tried to stand out by pushing the limit of number of characters in a given captcha and by how many unique characters we used. We believe we achieved that goal.

Introduction

The title of captchas is notably “Prove you’re not a robot.” and they are intended to only be completed by humans. Often times, even humans have trouble determining the string represented. Therefore, solving them based only off of an image is a very challenging computer programming strategy. If it weren’t for neural networks and CS230, we would not have the skills necessary to tackle this problem. However, when equipped with deep learning techniques, and access to very powerful computers. We were able to prove that our computers were not robots, thousands of times, in fact. This illustrates the strength of deep learning. Since captchas are images generated by computers using traditional programming methodology, attempting to guess them with deep learning is essential a battle between deep learning and regular programming. We sought to develop models that could achieve enough accuracy to conceivable pass the captcha test.

We created several models in attempting to create a captcha solver. Several smaller networks capable of solving captchas of length 1 - 3. We created models in multiple libraries and using many architextures. We did this to develop familiarity with the problem and models available to us. We then explored which models were providing us with the best results and choose to develop them further.

Our different models work with various string lengths and alphabets (possible characters to use in the string). The inputs to our models are RGB images with values ranging from 0-255 before we preprocess them to be between 0.0 and 1.0. The outputs are “n-hot” arrays which can be thought of as the concatenation of n one hot arrays with each index corresponding to a character in the captcha alphabet (“ “, 0-1, a-z, A-Z). Thus the output, represents a n-character string.

The two models we choose to explore the most were a ResNet50 and a flexible fully connected neural network. The ResNet50 is a 50 Layer Residual Network used for image classification. It is generally better than a traditional Deep CNN because accuracy doesn’t degrade as it goes deeper and learning isn’t as difficult. The ResNet can be looked at as subtraction of features learned from the input of a given layer.

The fully connected network uses is written in tensorflow. I uses relu activation and a sigmoid output layer. It is training using a mean squared difference loss function, we found this to be most effective when working with our output format. For simplicity sake, we typically used the same number of nodes for the all of the layers. We found that scaling up

the number of nodes was most effective for dealing with an increase in alphabet length. And scaling up the layers, worked well for increases in string length. For example, with string length: 1 and an alphabet: (0-10), a network with 1 layer and 10 nodes was able to achieve 0.99 accuracy. With string length 1 and an alphabet: (0-10, a-z) a network with 1 layer and 100 nodes was able to achieve 0.87 accuracy. With an alphabet (0, 10) and a string length of 4. A model with 4 layers and 100 nodes was able to achieve an accuracy of 0.49. This observation helped us to predict the most effective layer sizes for our larger models without having to experiment on them which can be very costly in terms of time and resources.

Dataset and Features

The most interesting part of our project is that we were able to use a real captcha generator from the python library. This allowed us virtually unlimited data during training. Therefore, we were truly able to investigate the benefit of having a large dataset. Interestingly, we notices no notable increase in training ability after around 20,000 images. This could perhaps be due to a lack of variety in captchas. We investigated making the images black and white to decrease our input feature size but we found that color differentiation was important in character recognition.

Methods

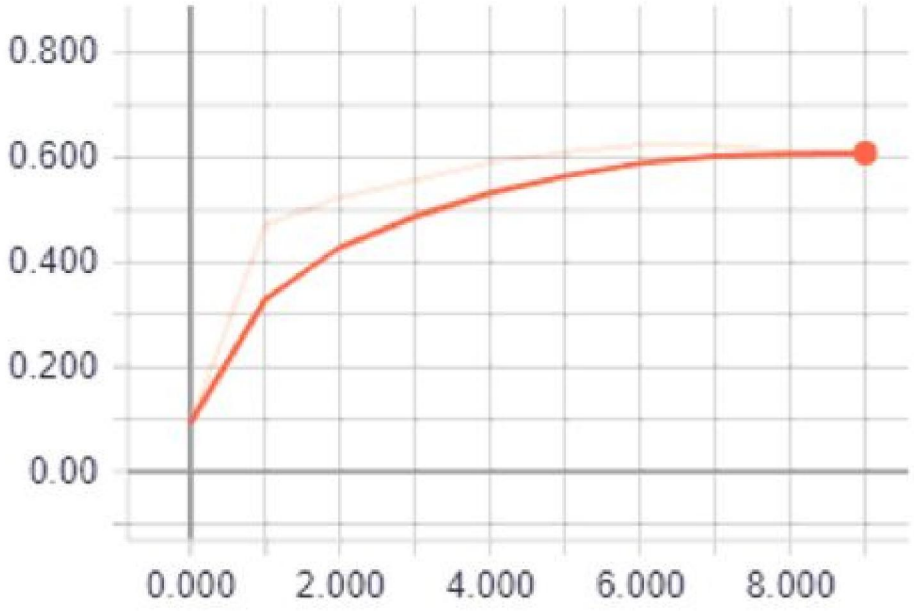
The fully connected neural network uses mean squared difference and xavier initialization. It uses an adam optimizer and a learning rate of 0.001.

Describe your learning algorithms, proposed algorithm(s), or theoretical proof(s). Make sure to include relevant mathematical notation. For example, you can include the loss function you are using. It is okay to use formulas from the lectures (online or in-class). For each algorithm, give a short description of how it works. Again, we are looking for your understanding of how these deep learning algorithms work. Although the teaching staff probably know the algorithms, future readers may not (reports will be posted on the class website). Additionally, if you are using a niche or cutting-edge algorithm (anything else not covered in the class), you may want to explain your algorithm using ½ paragraphs. Note: Theory/algorithms projects may have an appendix showing extended proofs (see Appendix section below).

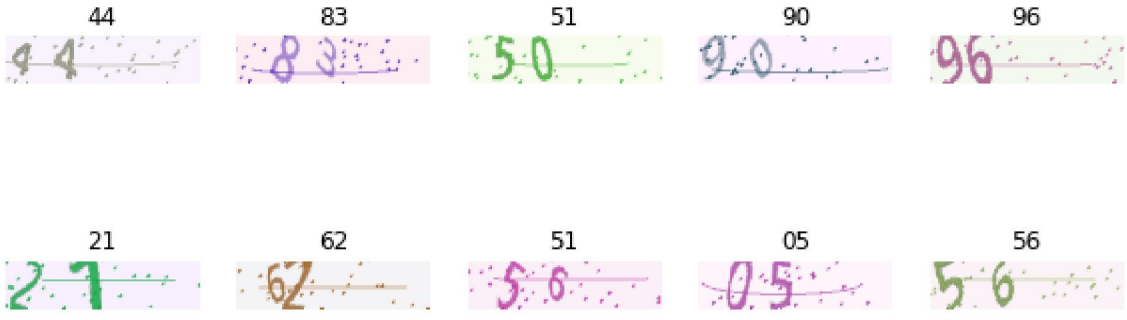
Experiments/Results/Discussion

Because we set out to develop models that could surpass a captcha test, we cared mostly about accuracy. We mostly cared about achieving accuracy that could be used to solve a captcha within a reasonable amount of attempts. Obviously our networks had to achieve

levels far beyond random guessing but we did not care much about surpassing an accuracy of say 25%. For the tensor flow model, we use the number of correct guesses over the total number of guesses. For the Keras models we use the build in categorical accuracy function. When we started we were getting decent accuracies above 50% with just 10 characters so everything seemed like it was going the right way as shown in the image below.



For the full alphabet of captcha characters and testing with variable string lengths of 4, 5, and 6. Our best model achieved an accuracy of 28%. We view this as a success based on our purposes because it could solve a captcha in an average of less than four guesses.
Below, we have pictured the guesses on the test set done by the tensorflow network over an alphabet of (0 - 10) and a string length of 2. In this case it was set up with 5 layers and a layer size of 100 nodes.



We have definitely not overfit our train sets because we made new training sets for every instance of our program. This made training realistics and in reality our test and train sets could have been produced in unison.

Two major bugs that we ran into during our process ate up a good portion of our resources. The first bug was the problem of exploding and vanishing gradients. We spend a long time trying to adjust our, initialization, learning rate, and network structures. However we found it was a result of our input features being passes in in the form 0-255 rather than 0-1. We also had many of of Keras models training with binary cross entropy as the accuracy function and this was producing misleading results. We thought we we achieved 0.98 accuracy but really our models were training to guess all zeros.

One additional note to consider when, dealing with a problem such as captcha solving is that there is intentional difficult present. Captchas are meant to be hard and many times humans cannot solve them on their first attempt we determined human error to be 0.34 on this generator by attempting to manually solve randomly sized and labeled capchas.

The next step would be to use other captcha generators. In order to grow our networks flexibility. Also, we would like to create an end to end program such as a web browser extension that can automatically surpass the captcha test.

Contributions

Isaiah worked on modifying multiple existing networks to fit our project so that we could choose the best one. Thunder worked on creating a flexible model that we could adjust to different alphabets and string lengths we wanted to work with. Isaiah's work is on the Keras Models and Thunder’s work uses tensorflow. Although we included only a few programs for simplicity sake. We both have written and tested many, many networks. The examples we included are the easiest to use or best illustrate our findings. Thunder also wrote the methods to flatten, unflatten, convert to one hot, and convert from one hot for the data manipulation.

Code

https://github.com/Isaiah21/cs230

References

Captcha: https://pypi.org/project/captcha/

Keras: https://keras.io/
MATPLOTLIB: https://matplotlib.org/
OpenCV: https://pypi.org/project/opencv-python/
PIL: http://www.pythonware.com/products/pil/
Python: https://github.com/python
RESNET50: https://keras.io/applications/#resnet16
String: https://docs.python.org/2/library/string.html
System Argparse: https://docs.python.org/2/library/argparse.html
Tensor Board: https://www.tensorflow.org/guide/summaries_and_tensorboard
TensorFlow: https://www.tensorflow.org/
Time: https://docs.python.org/3/library/time.html
VGG16: https://keras.io/applications/#vgg16
VGG19: https://keras.io/applications/#vgg19

Jackon Yang. “Captcha Solving Using TensorFlow.” Internet: https://github.com/JackonYang/captcha-tensorflow, July 25, 2018 [Mon Dec 3].

Panagiotis Tigas.“simple-captcha-solver.” Internet: https://github.com/ptigas/simple-captcha-solver, July 25, 2018 [Mon Dec 3].

CS230 Hands-on session 6: “TensorFlow Blitz with PyTorch Bits” Internet: https://colab.research.google.com/drive/1nilPoE43JgHaBzUHLeUFPx1rtM8uambw#scrollTo=uB6vfKePX53D , July 25, 2018 [Mon Dec 3].