GAPTCHA

Logan Lieou

November 17, 2021

Abstract

Over the years computer vision has become more and more complex over time, and the ability for companies and websites to prevent botting online has become a more difficult issue. The Completely Automated Public Turing Test to Tell Computers and Humans Apart (CAPTCHA) test was originally invented to overcome this issue but as time goes on it becomes inc. more difficult to prevent bots from bypassing these tests, in this project we attempt to show how one may go about bypassing the CAPTCHA test using AI.

1 Data

For this project we used data from a publicly avalible set of CAPTCHA images on kaggle. Each image is 200×50 with 4 channels, I assume RGB + some sort of mask for CAPTCHA, this is easily fixed with some preprocessing though, mapping $200 \times 50 \times 4$ into an image that is $200 \times 50 \times 1$. A big issue we face with the overall program is the preprocessing and pipelining of data from raw images to information that we can handle with our model and process in order to get adaquate predicitions. Mapping information from string to a tensor is difficult and there are various approches one such approach is from here [1]

```
images = sorted(list(map(str, list(data_dir.glob("*.png")))))
labels = [img.split(os.path.sep)[-1].split(".png")[0] for img in images]
characters = set(char for label in labels for char in label)
```

first the author creates a list of images and labels, labels are extracted from image titles.

```
char_to_num = layers.StringLookup(
 vocabulary=list(characters), mask_token=None
)
num_to_char = layers.StringLookup(
 vocabulary=char_to_num.get_vocabulary(),
 mask_token=None, invert=True
)
```

then they write a method to map from characters to tensors and tensors to characters, this this is very abstracted and very declarative but the solution is overall better than the the hard code one hot tensor that we ended up implementing this, would hypothetically scale across different types of image information and data and allow for perhaps more flexible models.

1.1 One Hot Encoding

One hot encoding is done by creating a matrix comparing across all classes giving 1 if true, 0 if false, in this case it's 1 if a letter is valued, and 0 if it's not. On the model output you would threshold these values in order to get 1 or 0 predictions.

```
ims = [imread("data/train/" + x) for x in os.listdir("data/train/")]
lbs = [x[:-4] for x in os.listdir("data/train/")]
```

I start off the same way as in [1] with a slightly different aproach and I think my approach is technically slower because they are using collections. I get all labels, and image paths. Next I write the forward mapping:

```
def forward_map(word):
 one_hot_word = np.zeros((len(word), len(symbols)))
 for n, letter in enumerate(word):
     one_hot_word[n][symbols.index(letter)] = 1
 return one_hot_word
```

this maps a word to a one hot tensor of size $(5, 100)$ both aproaches are hard code restricted to a set of characters, and word length, this is really unfortunate because this has the tendency to hyper fit our model, there's ways around this by writing a more general data pipeline or using bounding boxes to individually classify characters, which may have been a better aproach overall. The reverse mapping:

```
def inverse_map(one_hot_word):
 word = ""
 for i in range(len(one_hot_word)):
     for j in range(len(one_hot_word[0])):
         if one_hot_word[i][j] >= 1.:
             word += symbols[j]
 return word
```

Takes in a a one hot tensor and outputs a word.

2 Models

2.1 Intutive Explaination

In this paper I cover my particular model the CRNN architecutre whatever you want to call it as that's what the architecutre most resembles. The model consists of 3 parts the convolutional, recurrent and transcription layer. When you pass an image to the model the convolutional neural network extracts and learns feature mappings, then you pass these the the RNN, in this case and in the case of my model you use a bidirectional LSTM in order to predict on the variable length label sequence LSTMs are very good at retaining and disregarding previous information due to it's residual nature. [2]

2.2 Mathematical Explaination

2.3 Code Explaination

3 Loss

We used categorical cross entropy as well as CTC loss.

3.1 Categorical Cross Entropy

Categorical cross entropy is a common loss function when dealing with many catgegories of data to classify we get the loss

$$CE = -\Sigma_{x \in X} f(x) log(g(x))$$

to minimize. Cross entropy measures the the average number of bits from distrobution g reletive to a given distrobution f for the same underlying value x [3]

3.2 Connectionist Temporal Classification

CTC was first proposed in [4]

4 Training

A simple training function may look something like this:

```
for epoch in range(10):
   for X, y in train_dataset:
      optim.zero_grad()
      preds = model(X)
      loss = loss_fn(preds, y)
      optim.step()
```

where you take feature inputs and validate these against labels the adjust the model accordingly using your optimizer. The interesting part here is of course the forward and inverse mappings of the data, y must be encoded with the forward mapping then y and model(X) can be compared.

5 Problems

Our models had issues with overfitting, this is due to how preprocessing works as covered before, in addition this also somewhat has to do with the way that tensorflow works.

5.1 Preprocessing

Preprocssing hard fit the model to only predict a certian length of characters, as well as in some cases only a certian subset of characters, as well although this is very good for our results it's not very good for generalizing the model, you could potentially improve this by having a dataset that contains all characters.

5.2 Tensorflow

The problem with tensorflow in this case is the fact that tensorflow expects you pass an explicitly sized tensor into the first layer of the model, this forces the programmer to code around the input tensor in a particular way that you wouldn't otherwise have to for example in the case of torch nn.Conv2d this could be seen as a good thing as the program is declarative but it makes is exceptionally difficult to work with if you're trying to minipulate tensors between layers as I was in the case of my model.

6 Conclusion

References

[1] AKNain, "Ocr model for reading captchas." https://keras.io/examples/vision/captcha_ocr/. Accessed: 2021-11-16.

[2] X. Feng, H. Yao, and S. Zhang, "Focal ctc loss for chinese optical character recognition on unbalanced datasets," *Complexity*, vol. 2019, 2019.

[3] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[4] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, pp. 369–376, 2006.