DATA MINING ASSIGNMENT 3

ASSIGNMENT DEEP LEARNING: BREAKING CAPTCHA

June 8, 2018

Student ID: 20174797
Student Name: Baturay Ofluoglu
University of Antwerp
Computer Science: Data Science

Contents

INTRODUCTION

The purpose of this project is to break a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart). As it is understood from the abbreviation, CAPTCHA is a type of Turing Test developed by Alan Turing who is known as the father of theoretical computer science and artificial intelligence. Turing Test reveals the question whether a computer can think like a human. Based on the definition, an ideal CAPTCHA should be recognized by a human but not by a computer. Generally, it is an image or a sound.

Thanks to developing algorithms and increasing processing power breaking CAPTCHA is getting easier. Within the scope of this project, I will try to break an image CAPTCHA by using the convolutional neural network.

THE GENERAL OVERVIEW OF THE PROJECT

1. Understanding the Data
2. Building Convolutional Neural Network
3. Predicting CAPTCHA without Filtering
4. Predicting CAPTCHA without Filtering
5. Evaluating The Performance of the Model

1 UNDERSTANDING THE DATA

The training data includes 34,275 CAPTCHA images, and the verification data includes 8568 CAPTCHA images. Note that training images does not contain verification images. Each image has 280x280 dimensions. For the simplicity, each image contains only one letter with a specific font. The letters consist of 26 lowercase Latin characters. I create the images in grayscale because I want to keep the problem simple. However, the variety of the training set can be increased such as more fonts, colorful images, etc. to find out more patterns.

The label is hidden under the image name of the CAPTCHA. An example file name is $'[0]\_1.jpg'$. In this case, *0* is the label where *0* symbolize the letter *a*.

Figure 1: Example of a CAPTCHA Image

2 BUILDING CONVOLUTIONAL NEURAL NETWORKS

In this project, I prefer to use convolutional neural networks (CNN) to break CAPTCHA by using TENSORFLOW library. It is a state of the art algorithm for image classification, and it needs less preprocessing compared to other classification algorithms. It is also much faster than dense neural networks because it requires fewer weights and the number of necessary hidden fully connected layers are less.

The main advantage of this algorithm is that it can find high-level features automatically. This is a good aspect of our project because the CAPTCHA images are very noisy.

The algorithm has five main layers which are convolutional, pooling, fully connected and prediction layers. Each layer also may have multiple sub-layers.

2.1 Convolutional Layer

The primary duty of convolutional layer is to find out the pixel patterns in image classification. If one applies more filters than the probability of finding different patterns increase but the execution time of the algorithm decreases. Convolution layer has kernel parameters, and each kernel helps algorithm to see the patterns.

In this project, I use three convolutional layers and 5x5 kernel matrices at all layers.

1st Convolutional Layer

Before applying the convolutional layer, I resize the train and validation set images from 280x280x3 shape to 50x50x3. These resized train images constitute the input of the 1st convolutional layer. The smaller images are better concerning execution time because more dimensions mean more convolution operation for the fixed kernel size. Note that, you may also lose some information if you make image dimensions very small. So, there is a trade-off between execution time and missing data.

For the number of filters, I plan to use 32 filters to find out the patterns of the image.

2nd and 3rd Convolutional Layer

Differently, from the 1st convolutional layer, I applied 64 filters instead of 32. The reason is that the input image dimension of the 2nd convolutional layer is 25x25x32 while the input image shape of the 1st convolutional layer is 50x50x3 (skipping pooling effect for now). Because the shape is bigger, I increase the number of filters to find more patterns.

2.2 Pooling Layer

In convolutional neural networks, the pooling layer comes after each convolutional layer to reduce the dimension of the convolutional layer output. I use two by two filter matrix and set the stride parameter to two. It means that this layer divides the output of the convolutional layer into 2x2 pixel matrices and these matrices should not intersect because the stride is two. After division, from each split matrix, only maximum pixel value can pass, and all the others are filtered out. So, from 4 value only one value can , and it reduces the dimension four times. For example, after applying pooling layer on top the 1st convolutional layer, the matrix shape will drop from 50x50x32 to 25x25x32.

2.3 Flatten Layer

After the 3rd pooling layer, the output matrix has 6x6x64 dimensions, and we need to give this matrix to a dense neural network as an input to train our algorithm. The dense neural network only accepts vectors as input. So, at this step, we convert the three-dimensional matrix to a vector.

2.4 Fully Connected Layers

There are two fully connected layers in this network, and the first layer has 1024 neurons. In this layer, there is a 40% drop out rate to overcome the over-fitting problem. The activation function of this layer is ReLU to handle non-linear relationships.

The second fully connected layer has 26 neurons because we have 26 classes to predict. Each neuron is dedicated to one class.

2.5 Predicted Class

This layer calculates the probability of each class based on the weights coming from the 2nd fully connected layer. After it calculates all the probabilities, it returns the class that has the highest probability.

2.6 Cost Function

The cost function of this network is cross entropy. It is a widely used error function in CNN for classification problems. It has some advantages over mean squared error or classification accuracy because it considers the probability distributions of the classes.

Let me give an example to make it more clear. For this example, please refer to Table1. At this table, there are two classes which are A and B. Also, we have two different CNN models as Model1 and Model2. At this case, if we consider accuracy as a performance metric, then both models accuracy are 50% because they both predict first row correct and the second row incorrect. The difference is Model 2 gives more probability to the correct classes both at the first and second rows. So, even if the accuracy is same, Model 2 is a better model regarding probabilities. Because cross entropy evaluates the performance model by using probability distributions, I prefer to use cross-entropy as a cost function.

Table 1: Example For Cross Entropy

Model 1		Model 2		Target	Prediction
A	B	A	B		
0.55	0.45	0.9	0.1	A	A
0.8	0.2	0.6	0.3	B	A

2.7 Cost Function Optimization

The main purpose of the model is to minimize the cost function. In this optimization problem, our objective is cross entropy function. We also know that weights of the hidden layers have a direct effect on objective function. Thus, weights are our decision variables and they need to be adjusted to optimize objective. I use Adam Optimizer as an optimization method. This optimizer is an extended version of the classical stochastic gradient descent algorithm. Adam algorithm maintains the learning rate for each network weight. So, it converges faster than gradient descent algorithm because it uses fixed learning rate.

3 PREDICTING CAPTCHA WITHOUT FILTERING

After building the convolutional neural network, I start training the model. As I mentioned before, I use 80% of this data as train set and the remaining as validation set.

At this section, I keep the noises at the training set. It means that I did not apply any preprocessing or filtering to the training images.

I set the learning rate as $10e-4$ and make 1000 iterations to optimize cross entropy function. Based on these parameters, I obtained 98.7% training accuracy, 98.3% validation accuracy and 0.050% validation loss. The model trained in 2 hours and 40 minutes with dual 2 GHz Intel Core i5. Note that because I do not have a dedicated GPU, I used CPU.

At the end of 1000 iterations, the model still did not converge because the objective function value keeps decreasing. The last 10 epochs results are shared below.

4 PREDICTING CAPTCHA WITH FILTERING

In this section, I train the CNN model with the filtered images. As I explained before CAPTCHA is a Turing Test and during the creation of the CAPTCHA images a lot of noises are added to CAPTCHA images to make recognition harder for the computers but not for the humans. So, there are a lot of dots, meaningless shapes and lines apart from the letters. My purpose is to detect all the meaningless shapes and filter them out.

Figure 2: CNN Model Performance Measures without Filtering

```
Epoch 20 --- Training Accuracy:  98.3%, Validation Accuracy:  96.6%, Validation Loss: 0.116
Epoch 21 --- Training Accuracy:  98.7%, Validation Accuracy:  96.6%, Validation Loss: 0.106
Epoch 22 --- Training Accuracy:  98.6%, Validation Accuracy:  98.0%, Validation Loss: 0.066
Epoch 23 --- Training Accuracy:  98.6%, Validation Accuracy:  97.4%, Validation Loss: 0.087
Epoch 24 --- Training Accuracy:  98.6%, Validation Accuracy:  97.0%, Validation Loss: 0.094
Epoch 25 --- Training Accuracy:  98.6%, Validation Accuracy:  97.3%, Validation Loss: 0.086
Epoch 26 --- Training Accuracy:  98.6%, Validation Accuracy:  98.1%, Validation Loss: 0.057
Epoch 27 --- Training Accuracy:  98.7%, Validation Accuracy:  97.5%, Validation Loss: 0.073
Epoch 28 --- Training Accuracy:  98.7%, Validation Accuracy:  97.2%, Validation Loss: 0.078
Epoch 29 --- Training Accuracy:  98.8%, Validation Accuracy:  97.8%, Validation Loss: 0.072
Epoch 30 --- Training Accuracy:  98.7%, Validation Accuracy:  98.3%, Validation Loss: 0.050
```

Even if training data seems in grayscale, it has three channels which are red, green and blue. I first convert three channel to one channel in grayscale to save memory. After, I set a binary threshold to the images. It means that pixels can be either black or white. There cannot be any color in-between. I then find all the contours at a particular image. It sees all the dots, lines, letters and meaningless other shapes. The primary purpose is to keep letters and remove all the other contours. At this point, I calculate the area of each contour and eliminate the contours that have less than a particular area. Thanks to this method, I remove most of the noises from the training images.

Figure 3: CAPTCHA without filter

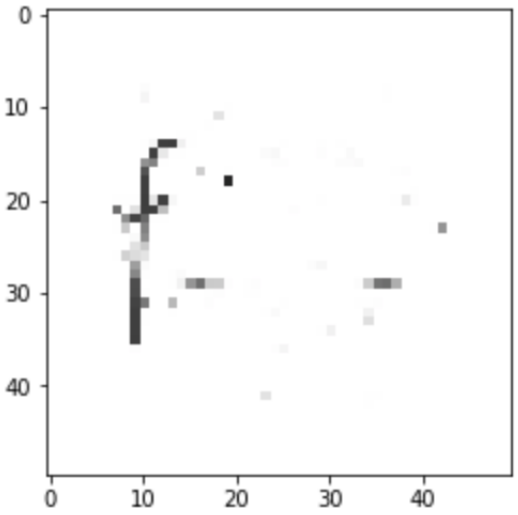
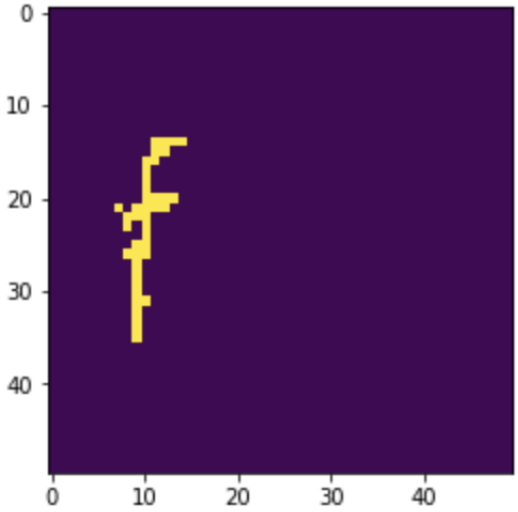


Figure 4: CAPTCHA with filter



After filtering, the model is trained in 800 iterations. At this time, it converges much earlier. The best final performance metrics for training accuracy is 98.7%, 98.6% for the validation accuracy, 0.033 for the validation loss. The last 10 epochs results are shared below. As you can see from the figure 5, the metrics are converged.

Figure 5: CNN Model Performance Measures without Filtering

```
Epoch 15 --- Training Accuracy:  98.7%, Validation Accuracy:  98.4%, Validation Loss: 0.041
Epoch 16 --- Training Accuracy:  98.7%, Validation Accuracy:  97.8%, Validation Loss: 0.056
Epoch 17 --- Training Accuracy:  98.7%, Validation Accuracy:  98.0%, Validation Loss: 0.048
Epoch 18 --- Training Accuracy:  98.7%, Validation Accuracy:  98.4%, Validation Loss: 0.034
Epoch 19 --- Training Accuracy:  98.7%, Validation Accuracy:  98.5%, Validation Loss: 0.036
Epoch 20 --- Training Accuracy:  98.7%, Validation Accuracy:  97.8%, Validation Loss: 0.055
Epoch 21 --- Training Accuracy:  98.7%, Validation Accuracy:  98.2%, Validation Loss: 0.045
Epoch 22 --- Training Accuracy:  98.7%, Validation Accuracy:  98.4%, Validation Loss: 0.033
Epoch 23 --- Training Accuracy:  98.7%, Validation Accuracy:  98.6%, Validation Loss: 0.033
Epoch 24 --- Training Accuracy:  98.7%, Validation Accuracy:  97.8%, Validation Loss: 0.054
```

5 Comparison of Filtered and Not Filtered Models

Regarding final model performances, filtered version is not much better than the model without filtering, but it converges much earlier. It converges after Epoch 13. The training time with filtering takes approximately 1 hours and 20 minutes which is much better than without filtering. For example, at the 4th epoch validation loss is 0.948 for the model without filtering and 0.164 for the model with filtering. Filtered model performs almost eight times better than the unfiltered model at the 4th epoch. It can be said that concerning execution

time and performance metrics, it is reasonable to filter training images.