CS771 Assignment-3

Group-41

Abhishek Soni (170034) **Aniket Gupta (170108)** **Bholeshwar Khurana (170214)**

Saurav Prakash (170646) **Sayak Chakrabarti (170648)**

Abstract

CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) are the most popular way to prevent a bot to log in to unauthorized systems. Usually CAPTCHAs contain some characters with some obfuscations in them. The aim of this project is to identify those characters correctly and recognize them in proper order.

1 Introduction

CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) are popular ways of preventing bots from attempting to log on to systems by extensively searching the password space. In its traditional form, an image is given which contains a few characters (sometimes with some obfuscation thrown in). The challenge is to identify what those characters are and in what order. In this assignment, we wish to crack these challenges.

For this assignment we have been given 2000 training images with correct code for each captcha being present as its filename. Each captcha image is of 600×150 pixels in size and the image is composed of 3 or 4 uppercase alphabet characters.

2 First Step: Observations

Solving the problem of recognizing CAPTCHAs required several acute observations for training a simpler model. The observations made were:

- The border colour is darker than the fill colour of the character.
- Although the characters might be rotated, they are not rotated much. Usually rotations are multiples of a smaller angle. In the given dataset the rotations were multiples of 10°, being $10^\circ, 20^\circ, 30^\circ$.
- The obfuscation lines in the backgrounds are relatively less thicker than the characters in the CAPTCHA
- Since the images were generated using Python Imaging Library (PIL), the number of obfuscation transformations were quite limited and it could be inverted to reveal the true encoding in the CAPTCHA.

Preprint. Under review.

3 Methods Applied:

Several methods known till date are available to solve the problem of CAPTCHAs. We used powerful and non-linear techniques like Deep learning (CNN specifically) to solve the problem. The following methods were applied for CAPTCHA character extraction:

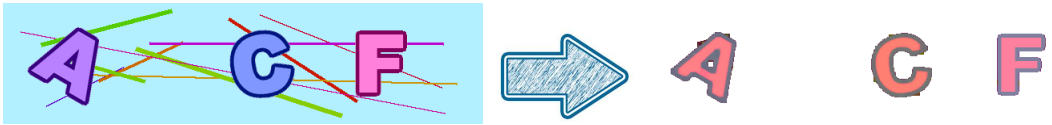
3.1 Background Extraction and Line Removal:

The CAPTCHA was first converted to its HSV (Hue-Saturation-Value) format. In order to find the color of the background we used the corner pixels and since there might be some obfuscation lines in the corners as well, so we took the majority of the 4 corners to choose the background color. We, then simply made the background black. This does not create any hindrance as the background is always lighter than the characters. We, then used **erosion** and **dilation** to remove the obfuscation lines and restore the thickness of the original letters.

Erosion erodes away the boundaries of foreground objects which are in white while the background is black. We used a linear kernel (6×6 matrix of ones) as a convolution during erosion. A pixel in the original image will be considered 1 only if all the pixels under the kernel are 1, otherwise it is eroded. Hence all the pixels near boundary will be discarded depending upon the size of kernel. The thickness or size of the foreground, i.e. white region, decreases in the image. It is useful for removing small white noises , detach two connected objects etc.

Next we used dilation. It is quite like the opposite of erosion where the pixel element is 1 if at least one pixel under the kernel is 1. We used the same kernel as in erosion. This increases the white region in the image, providing easier contours (in next subsection). Usually erosion followed by dilation removes noises, in our cases obfuscation lines. This process of erosion followed by dilation is called **opening** which is used to remove the noise present in the background as well as inside the characters. Then we changed the background black to white, to help with segmentation.

This is how a CAPTCHA "ACF" turned out after this process.



3.2 Extraction of Individual Letters:

Our goal is to separate out each letter present in the CAPTCHA and write them as separate images. In order to separate letters we find **contours** in the CAPTCHA. Contours are curves joining all the continuous points along the boundary, having same color or intensity. These contours pass through each character. In captchas, two letters can also be close or joined to each other. To counter this problem we check if any contour is wide enough, if so we split it into half. We store the coordinates in a list.

We encountered another problem regarding the letter "Q", which initially created wrong contours owing to the straight line present at the bottom of the alphabet. This happened due to numerous obfuscation lines. To prevent this we constrained the height and width of each contour.

Using these methods the alphabets have been successfully separated and stored in corresponding files which represent the label name. We then provide the output images of characters to our NN (Neural Network) model for building and training.

The individual letters after further extraction turned out like this:

3.3 Building and Training Neural Network Model:

We used Convolutional Neural Network (CNN) model for identification of alphabets. In CNN model, each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and finally Softmax function to classify an object with a certain probability. After the extraction of images using contours, it was stored into corresponding files and fed into the CNN model. First, we will prepare the training data for our CNN Model. Then we read the extracted images, convert it to gray scale and resize to a fixed size. The image is converted to a matrix so as to make it compatible for our CNN model. All the image arrays and image labels are stored in lists. For better training we normalize the pixel value between 0 and 1. For training our model using heldout validation we split the data list into training set and test set, represented in Table 1.

Now that the training data is ready we create our model using keras sequential model.

The model used by us takes as input a 15×15 matrix, which is fed into three hidden layers. We tried different models until we arrived at the optimum, i.e. this one. The first hidden layer is a Convolution layer where the kernel is of size 5×5 using 20 filters having unit stride length, after which we do max pooling of dimension 2×2 with stride length 2 in both dimensions. The next hidder layer uses a convolution kernel of size 5×5 and 50 filters, again with same stride length. The activation function used in both these hidden layers were "ReLu". We again do max pooling of dimension 2×2 with stride being 2 in both directions, after which we flatten the matrix and use a dense network of size 55, having "ReLu" activation function and a dropout of 30%. The output size is 26 and softmax is used to provide the probability of each alphabet for prediction.

The loss function is categorical cross entropy and the optimizer used is adaptive momentum estimation, adam estimation.

We tuned the hyperparameters manually. There was a tradeoff between the size of the input and the training accuracy increasing input size also increased the accuracy of the model(which is expected as the letters would be easy to recognize when the they are large).But decreasing input size also decreased the model size, so We had increased the number of nodes in the Fully Convoluted layer to increase back the accuracy.

The model accuracy for training set and validation set for different ratios is:

Ratio (Training:Validation)	Training Set	Validation Set
90:10	100%	100%
80:20	100%	100%
75:25	100%	100%
50:50	100%	100%

Table 1: Accuracy obtained

3.4 Prediction using Deep Learning:

For prediction we use the above procedure for background extraction and line removal (same as section 3.1) on the captcha image. Then we extract individual letters from the captcha image into separate images for each letter (same as section 3.2). The number of individual letters obtained gives us the number of characters present in the captcha. For each image of the letter extracted we modify image letter to make a 15×15 matrix for input to our model. The model will predict the output label for image of each letter and then we convert the prediction value back to a normal letter for that captcha image.

3.5 Testing:

During testing we first preprocessed the image as described in subsections 3.1 and 3.2. The number of letters in which the image was divided gives us the number of letters in the alphabet then we predict each letter using our model and return the final predicted string and length.

4 Code:

Code is uploaded on "cse.iitk.ac.in/users/aniketg/submit.zip".

References

[1] Shubham Chauhan, "Extract Captcha Text using CNN in Python(Captcha solver)", Medium Blog, https://medium.com/analytics-vidhya/cnncaptchasolver-5625b189a14f

[2] Github link to code for "Extract Captcha Text using CNN in Python(Captcha solver)": https://github.com/chauhan01/Captcha-text-extraction

[3] OpenCV: Morphological Transformation, https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html

[4] Geetika Garg, "Neural Network CAPTCHA Cracker", San Jose University, 2015