DeCAPTCHA

Mriganka Shekher Chakravarty (19111053)
mriganka@iitk.ac.in

Sayed Abbas Haider Abidi (19111081)
sahabidi@iitk.ac.in

Tushar Shandhilya (19111418)
stushar@iitk.ac.in

Jaydeep Meda (19111039)
jaydeepm@iitk.ac.in

Sharath HP (19111082)
sharhp@iitk.ac.in

Abhishek Jaiswal (19111262)
abhijais@iitk.ac.in

1 Algorithm Description

A simple method to solve the problem of finding the characters in the given supplied images by feeding them directly in to a powerful model such as neural networks would render the process too expensive (model size would be large as well). We instead chose to first pre-process the image by separating the background from the foreground characters and after segmenting the given image into individual characters, fed the resulting characters into a custom convolutional neural network (CNN) for classification. Owing to the different orientations of the various characters in the image and different backgrounds, a powerful non-linear model was conspicuously our first choice and it proved to do well on the given dataset as well. The following sections give a detailed description of the entire workflow.

1.1 Pre-Processing Phase

The images given were in RGB format having obfuscation lines which makes them hard to segment individual characters. The following steps were adopted to render an image suitable for segmentation (removal of obfuscation lines):

- Converted the RGB image to HSV format .
- Learned the color of the background and subtracted the color from the image to effectively nullify the background pixels.
- Eroded the image using cv2.erode() (1) function using a 3x3 kernel for 8 iterations, Here we used a standard 3x3 kernel and the number of iterations to be eroded was found by randomly looking at 20 images for iterations 4,5,6,7,8,9. From the dry runs we found 8 (number of iterations) to be suitable for removal of obfuscation lines while at the same time preserving the characteristics of the individual letters. A sample image can be seen in Figure 1 and Figure 2
- Converted the image to gray-scale and thresholded to convert it into a binary image with only black and white pixels.

Preprint. Under review.

Figure 1: Pre-Processed image before thresholding

Figure 2: Pre-Processed images before thresholding

1.2 Segmentation Phase

The heuristic we used to segment the image into individual characters is described below. Figure 3 and Figure 4 show a visualization of the individual character segments identified by our algorithm after segmentation.

- After closely analysing the given training samples of the CAPTCHAs we noticed that, even though the images were tilted around a pivot, the boundary of the letters do not intersect. Meaning we can unambiguously separate the characters in the image by scanning along vertical direction, which would help separate adjacent characters.
- To segment out the characters from the given CAPTCHAs, by sweeping the entire image with a vertical line from left end to the right end i.e, a column vector.
- We assumed the start of a contour, whenever we detected a lit pixel in the sweeping column vector. This could only happen as the characters were separable by a vertical line, as mentioned above. We further assumed that we finished sweeping over a meaningful character once all the pixel in the sweeping vector died. For the last character in the CAPTCHA, we assumed that we successfully scanned a meaningful character either if all pixels on the sweeping vector blackened out, or we reached the end of the image.
- With the above method, we get the number of characters present in the image as well as the individual segments containing a character each.

As an alternative, we also tried segmenting the images using the cv2.canny() (1)function to find the edges and then subsequently finding the contours using cv2.findContours() (1) function after preprocessing. We were unable to get a perfect enough segmentation using this method hence we stuck to the approach described above. Using the segmentation we are able to find number of characters present in the image.

Figure 3: Segmented characters of image in Figure 1

Figure 4: Segmented characters of image in Figure 2

1.3 Classification Phase

After getting the separated characters we feed them to a CNN for classification to find which characters are present in the segmented image. Firstly all of the images were resized to 20x20 pixels for input to CNN and the output labels were one-hot encoded using LabelBinarizer(scikit-learn). These encodings were dumped to a file to be used during prediction. Our simple model architecture is described below:

- First Convolutional Layer (8, 5x5 Filters with padding('same') and 'relu' activation
- MaxPool with (2x2) stride and (2x2) pool size
- Dropout with 20% input set to zero
- Second Convolutional Layer (8, 5x5 Filters with padding('same') and 'relu' activation
- MaxPool with (2x2) stride and (2x2) pool size
- Dropout with 20% input set to zero
- Then the output was flattened and connected to a fully connected layer with 128 neurons and relu activation
- Final output layer with 26 neurons(output size) and softmax activation.

We used scikit learn (2) train-test split of 75-25 and scikit-learn GridSearch function to tune the neural network architecture(results in Appendix) and finally choose the parameters with best accuracy and then did a final training on full dataset. The model was trained using 'adam' for 40 epochs with a batch size of 26 and categorical crossentropy loss.
Keras library (3) is used to make a convolutional neural network to make the classifier model. Following the above procedure on segmented characters in the same order in which they were generated gave us the final list of characters in the image in the required order.

This part of code was written using reference from: (4)

2 Code

Code Link :- cse.iitk.ac.in/users/stushar/assn3/submit.zip

References

[1] https://docs.opencv.org/2.4/modules/imgproc/doc/imgproc.html

[2] https://scikit-learn.org/stable/documentation.html

[3] https://keras.io/

[4] https://medium.com/@ageitgey/how-to-break-a-captcha-system-in-15-minutes-with-machine-learning-dbebb035a710

[5] https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras

3 Appendix

Some HyperParameter Tuning Results

Best: 0.999610 using {'n2Neurons': 8, 'N3Neurons': 128, 'NNeurons': 8}
Mean, StdDev, Parameters
0.997661 (0.001654) with: 'n2Neurons': 6, 'N3Neurons': 128, 'NNeurons': 6
0.998441 (0.001459) with: 'n2Neurons': 6, 'N3Neurons': 128, 'NNeurons': 8
0.998830 (0.000955) with: 'n2Neurons': 6, 'N3Neurons': 128, 'NNeurons': 10
0.997271 (0.000551) with: 'n2Neurons': 6, 'N3Neurons': 64, 'NNeurons': 6

0.998051 (0.001103) with: 'n2Neurons': 6, 'N3Neurons': 64, 'NNeurons': 8
0.999220 (0.000551) with: 'n2Neurons': 6, 'N3Neurons': 64, 'NNeurons': 10
0.998441 (0.000551) with: 'n2Neurons': 6, 'N3Neurons': 200, 'NNeurons': 6
0.998441 (0.000551) with: 'n2Neurons': 6, 'N3Neurons': 64, 'NNeurons': 4
0.998441 (0.000551) with: 'n2Neurons': 6, 'N3Neurons': 64, 'NNeurons': 6
0.999220 (0.000551) with: 'n2Neurons': 6, 'N3Neurons': 64, 'NNeurons': 8
0.996101 (0.003070) with: 'n2Neurons': 6, 'N3Neurons': 75, 'NNeurons': 4
0.998830 (0.000000) with: 'n2Neurons': 6, 'N3Neurons': 75, 'NNeurons': 6
0.998830 (0.000000) with: 'n2Neurons': 6, 'N3Neurons': 75, 'NNeurons': 8
0.998051 (0.000551) with: 'n2Neurons': 8, 'N3Neurons': 32, 'NNeurons': 4
0.998441 (0.000551) with: 'n2Neurons': 8, 'N3Neurons': 32, 'NNeurons': 6
0.996881 (0.002205) with: 'n2Neurons': 8, 'N3Neurons': 32, 'NNeurons': 8
0.998830 (0.000955) with: 'n2Neurons': 8, 'N3Neurons': 64, 'NNeurons': 4

size = 665 kb
neurons = 8 64
Total time taken is 29.107095 seconds
Fraction of code lengths that match is 1.000000
RXD true= RAD
KMS true= EMS
Code match score is 0.999147

size = 405 kb
neurons = 8 8 128
Total time taken is 28.908505 seconds
Fraction of code lengths that match is 1.000000
LNSX true= LHSX
JZN true= JZH
RXD true= RAD

size = 10mb
neurons = 32 64 512
Total time taken is 32.807008 seconds
Fraction of code lengths that match is 1.000000
AMS true= EMS
Code match score is 0.999574