deCAPTCHA
CS771A Project Report

MLG 35
Anil Kumar Gupta (*ganil@iitk.ac.in*)
Ayush Tulysan (*ayusht@iitk.ac.in*)
Debabrata Ghosh (*drtghosh@iitk.ac.in*)
Peshal Agarwal (*peshal@iitk.ac.in*)
Angad Chandhok (*angad@iitk.ac.in*)
Mentor: Dr. Purushottam Kar

1 Problem Statement

To show that the CAPTCHAs used by our institute's webmail doesn't comply with the Turing test requirements. The goal here is to try and break a single type of CAPTCHA among the aforementioned and identify challenges and problems in doing so. Hence the idea is to learn a mathematical model for solving CC/CSE captcha. Whichever model we choose, it should be able to recognize the characters in a new captcha.

Once we have one such model, our next target is breaking more complex CAPTCHAs. This process will alternate between generating more complicated CAPTCHAs and strengthening our model to translate them well.

2 Problem Motivation

Completely Automated Public Turing test to tell Computers and Humans Apart or CAPTCHA is a type of challenge-response test used by many web applications to ensure that the response is not generated by a computer (source: Wikipedia).

Figure 1: The current CC captcha.

Without a strong captcha, the purpose of the above test fails. Even if the adversarial algorithm can break the test with a 50% chance, it becomes pretty simple for a machine to fool a web server.

Once we encounter a problem with the CAPTCHA, our next step is to take measures to solve it which motivates us to pick up this project and play around with the ease with which webmail CAPTCHAs can be broken in order to find ways of preventing that from happening and attempt to further improve the CAPTCHA strength against automated attacks.

3 Existing Work/Literature Survey

In our project, we solely focus on IIT Kanpur's webmail (CC and CSE) CAPTCHAs which were previously not undertaken but there were a few works on other modules of squirrelmail CAPTCHAs as:

- A project done by a Ankush Sachdeva, that works for IITK webmail, considers single color background and allows character rotation/size-variance (webmail1.iitk.ac.in). His work was based on TkInterface(Tkinter) package and edge detection using distance from center of mass and pixel to pixel matching.

Figure 2: IITK webmail1 CAPTCHA.

- Kavish Dahekar, IIT Guwahati worked on a Feed Forward NN with no hidden layers and backpropagation with learning rate 0.5 using the overall forward step $y = Wx + b$ with the aid of OpenCV 3 and Tensorflow packages.

Figure 3: Captcha used by IIT Guwahati webmail: webmail.iitg.ernet.in

Other than that there have been a plethora of works on breaking or generating other different kinds of CAPTCHAs. This can be broadly partitioned into two types:

- Papers and Research work
  1. Techniques for segmenting uniformed or proportional fonts, broken and touching characters; techniques based on text image features and techniques based on recognition results were covered.
  2. Efficient methods based on shape context matching that can identify the word in an EZ-Gimpy image with a success rate of 92%, and the requisite 3 words in a Gimpy image 33% of the time were developed.
  3. Divided into two sequential tasks: detecting words regions in the image, and recognizing the words within these regions using a CNN classifier that enables efficient feature sharing.
  4. Human Interaction Proofs (HIPs) generated automatically are too easy to be broken by computers and difficult to be recognised by humans, while there are techniques to develop segmentation based HIPs which are easy for humans but tough for computers.
  5. An Active Deep Learning strategy was proposed that makes use of the ability to gain new training data for free without any human intervention which is possible in the special case of CAPTCHAs and was discussed how to choose the new samples to retrain the network and present results on an auto-generated CAPTCHA dataset which dramatically improved the performance of the network than initially having only few labeled training data.
- Code base available online
  1. CAPTCHA-breaking by lllcho on Github lllcho uses a Deep learning approach to decode the captchas. These models break the captchas in *pinyin* script. The claimed accuracy of about 0.99 doesn't agree with the sample run. lllcho uses Keras which is a Theano based deep learning library for implementing the deep learning model.
  2. Deep Learning blog about breaking captchas using Neural nets.
  3. Decoding Captchas by Ben E. Boyter ad-hoc method by another user to break simple captchas using clustering and noise removal to extract individual characters. A vector space implementation has been suggested to predict classes for characters.

4 Novel Contribution

Although a lot of significant and high end work has been done in breaking different kind of CAPTCHAs involving character recognition, segmentation to state-of-the-art deep learning models, there was no previous work done on the specific modules of webmail captchas. So implicitly the work we have done is totally fresh to the field. Also we approached to solve the problem in a very simple but efficient ad hoc method which is unique.

5 Methodology and Results

Both CC and CSE use squirrelmail's application for providing web based mail clients. This application provides choice among 16 different modules for generating CAPTCHAs. Thankfully, this application was open source. We downloaded the repository, added the CAPTCHA plugin and thus, we had a webmail server running at our local machines. This provided us with liberty to generate as many training examples as we needed.

5.1 CSE Webmail Captcha

5.1.1 Drawbacks

We first look into the features of the CSE Captcha. These features are prone to exploitation using a simple ad-hoc method:

- The font size of all characters and numbers is set to be uniform.
- Orientation of all characters and numbers is set to be uniform (all of them are upright) with no tilt or angle.
- All the characters have a uniform texture with singly filled color.
- Each CAPTCHA has exactly a combination of five characters or numbers and the positioning of each of these five is almost same across all CAPTCHAs.

Figure 4: CAPTCHAs at CSE webmail.

5.1.2 Ad-hoc method to break CSE captcha

This method makes use of the observation that the characters never change their shape or size. And characters always occupy a very precise location. At first we generate the CSE CAPTCHAs using the server setup. The setup was altered so that labelled images could be obtained by directly from backends modules rather than through HTTP requests to the web server.

- **Training**: For training purpose we need enough data so that we can capture all possible characters and numbers appearing in the CAPTCHA while being generated from the configuration file as well as the positioning of each of them as first to five. After collecting the positions of the 1st, 2nd, 3rd, 4th and 5th characters or numbers we assign them according to the label e.g. from the CAPTCHA reading '27bjr', we get the positions of 2,7,b,j,r while sitting on the 1st, 2nd, 3rd, 4th and 5th positions respectively as shown in the figure. From the training data it is vividly visible that for each of five positions we can choose a fix bounding box which fully contains the certain character or digit placed there applicable for all the CSE Webmail CAPTCHAs.

Figure 5: CSE CAPTCHA character positions.

Hence, we get the following bounding boxes of the five positions defined using distance from left, right and bottom boundaries:

par[0] = [16, 17, 87]
par[1] = [33, 17, 70]
par[2] = [51, 17, 52]

par[3] = [66, 17, 37]
par[4] = [85, 17, 18]

and using RGB color space or color model we can see there are only three kinds of color used in the CAPTCHA with RGB values [20,40,100] used in the main CAPTCHA, [100,120,180] used in the background noise [255,255,255] i.e. white as the background color. We use masking in each of three RGB layers on top of each positioning to get characters or numbers for them as follows:

Figure 6: 2 and b in different postions obtained from training.

- **Testing**: After collecting all the templates for relevant(generated in training data) numbers or characters in every position of the five defined, we simply do a template matching in each of the predefined bounding boxes of the new CAPTCHA generated (in test dataset) against the collection we acquired via training after using same kind of masking on test data. Using this simple ad-hoc method we got an accuracy as high as **96.24**% in labelling our test data of 2631 CAPTCHAs.

5.2 CC Webmail Captcha

5.2.1 Exploitable Features

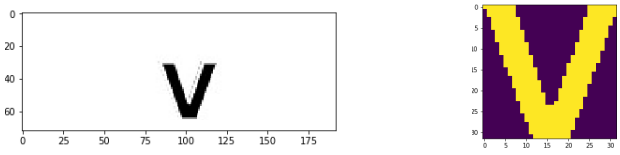
- The font size of all characters and numbers is set to be uniform.
- Orientation of all characters and numbers is set to be uniform (all of them are upright) with no tilt or angle.
- No stray lines cross over any character.
- Characters have a solid border and fill. The borders are contrasting to the fill

Despite these flaws, this test has some features which makes it more difficult than the previous

- Stray lines, ellipses and arcs in the image make it difficult to segment the image
- Although the fill of the characters appears solid, a minute gradient exists across the width.
- The character count varies between 3 and 4

5.2.2 Training

1. Generate single characters using the script that produces CC captcha



2. Extract the closest bounding box of the character and convert it into a binary(0-1) image for creating the template.

3. Generate labelled dataset for three character CAPTCHAs. Each image's label consist of the character sequence it contains and the RGB values of fill color in each character
4. Crop the boundary by 2 pixels from each side to remove the border
5. Reduce each of 256x256x256 color into 4x4x4 colors by dividing by 64.

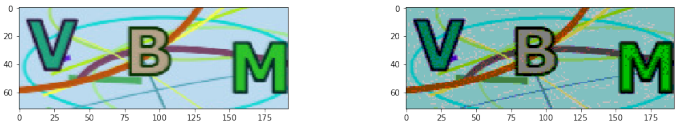


Figure 7: Before binning (left) and after binning colors (right)

6. Reduce the color RGB values obtained from the label. Now, both image and labels are similar dimensional. These labels can now be used to extract characters from the image. Following is a sample output of this extraction on above image

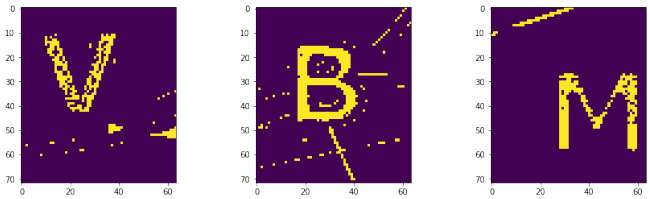


Figure 8: Binary images of 'V', 'B' and 'M' (from left to right) with scores V: 0.549, B: 0.634 and M:0.526

7. For each of these 3 images calculate the intersection over union score with the all 35 templates. Keep the maximum score for each character to form a 35 dimensional vector

```
[ 0.30795848  0.24942792  0.19733925  0.23024831  0.2557652   0.29052632
  0.31967213  0.23569024  0.3097166   0.25641026  0.25331126  0.25
  0.23486239  0.2516269   0.26256983  0.21272727  0.20950704  0.31464174
  0.30392157  0.30382294  0.28832117  0.22395023  0.26916525  0.20748299
  0.28994083  0.20454545  0.2755102   0.23473282  0.2408377   0.31531532
  0.54939759  0.28532609  0.32511211  0.40909091  0.27207637]
```

Figure 9: 35 dimensional vector corresponding to binary image of 'V' (shown above)

The above vector corresponds to one character. One can argue that the character corresponding to maximum score can be used as prediction. But our experimental results do not align with this. Even after binning the colors into 64 bins, a lot of characters' fill gets broken into two bins, which results into score vectors which do not follow the maximum rule.

8. Train a Multiclass linear SVM(OvA) which treats this vector as a feature vector and outputs a label.

This SVM classifies input into 35 possible classes by an accuracy of 98.3 %. Although, it can identify characters very well, it fails in differentiating between characters and background class

5.2.3 Testing

1. Crop the boundary by 2 pixels from each side and reduce each of 256x256x256 color into 4x4x4 colors by dividing by 64.
2. Find top 8 color frequencies in reduced image, ignore the top most (representing the background color)
3. Trisect each image(according to width) and create binary image for each of the top seven colors

4. Calculate the Intersection over union score for each of these 7 images with the template of each character.
5. Keep the maximum score for each character to form a 35 dimensional vector.
6. Pass this vector to the SVM to obtain a 36 dimensional vector representing the probability of being each character(or background) and pick the character with maximum value
7. From the seven images the one with the maximum value of the probability is the output character

5.2.4 Limitations

- Not flexible with number of characters in CAPTCHA
- Requires modification if characters are tilted or vary in size
- Wrongly predicts similar looking characters such as 'O' and 'Q'.

6 Experimental Details

The initial idea was originated from the observation that each character has solid fill (with a small gradient) and the characters are confined to a region in the image. So, it should be easy to segment the image between characters, background and stray lines. We wanted to create clusters for image pixels and locations in order to crop the individual characters, so we used the very popular **SLIC**: Simple Linear Iterative Clustering with 40 clusters.

- Forms any number of clusters for each image based on a 5 dimensional vector for each pixel (location (x, y) and color (L,a,b))

- This extracts some characters *"nicely"*

Figure 10: Slicing of character 'F'

- But for some, it expands the characters (leaks) or breaks them.

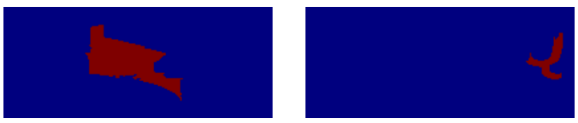


Figure 11: Slicing of characters 'L'(left) and 'Q'(right)

Since, we get both smugged and broken character images, increasing or decreasing the number of clusters would not help. For segmenting the image, the popular and widely used method SLIC fails. We then go on to create an adHoc method specific to our problem.

Currently, each pixel has three 8-bit channel RGB representation. Each channel can take values between 0 to 255. We reduce this space to 2-bit channel which can attain values from 0 to 3. Now each pixel only has 64 possible values. (see more details in 5.2.2)

6.1 Deep Learning

6.1.1 Training

We used standard vanilla Resnet18 model for training.

1. Cropped the boundary of the CAPTCHA by two pixels from each side to remove the black border.
2. Trisected the image according to width to obtain each character.
3. Scaled each part in 224X224 and converted it to a tensor.
4. Trained the Resnet on these tensor-label pair.

6.1.2 Testing

We went through the following steps for testing purpose:

1. Likewise for training again cropped the boundary of a new CAPTCHA by two pixels from each side to remove the black border.
2. Trisected the image according to width to obtain each character.
3. Scaled each part in 224X224 and converted it to a tensor.
4. Fed each tensor to the model in order to obtain the sequence of the characters of the CAPTCHA.

We got as high as 99.72% accuracy on individual characters of the similar test images used in ad-hoc method but around 95% accuracy on the whole.

Figure 12: Varying size

Figure 13: Rotating characters

The advantage of using deep learning model is that it can also handle CC CAPTCHAs which allow varying font size (36 to 52) and rotation of the characters by an angle ranging from -20 to 20 degrees with sufficient accuracies but our ad-hoc method can not e.g the deep learning model predicts or labels CAPTCHAs incorporating *rotation of characters* with **81.7**% accuracy (over 2000 images) and CAPTCHAs with *varying font size* of characters with **78.5**% accuracy (over 1000 images) which can be improved upto 95% accuracy.

Hence by adding the features like variant font size or rotation while generating the CC CAPTCHAs, we can substantially undermine our previous ad-hoc method of breaking the freshly generated CAPTCHAs with significant accuracy.

6.2 Data

As we earlier mentioned, both CC and CSE use squirrelmail's application for providing web based mail clients which is open source. We downloaded the repository, added the CAPTCHA plugin and thus, we had a webmail server running at our local machines providing us a way to generate as many training examples as we needed. Now for getting the labelled data; in case of CSE it was a piece of cake aided by the configuration file to generate labelled CAPTCHA images (i.e. with file name reading the CAPTCHA) but for CC we had to do *manual tagging* to fulfill our requirement of data.

6.3 Benchmarks

There was no available data on the work done involving the same module of squirrelmail CAPTCHAs and hence actual bench-marking is not possible in this case. Although we can mention the work on IITG webmail where 3000 characters from the collected data were used for testing the trained model using FFNN. The accuracy was observed to be about 95%. In our case Resnet gives around 99% accuracy on simple upright CAPTCHAs (individual characters).

7 Future Work

In retrospect, we can think of extending our adhoc method to breaking the CC CAPTCHAs which also encounter *varying font size* of the characters, *rotation* by an angle of the characters or numbers or having upside down characters. We can also affix the case of not only 3 but also *4-character CAPTCHAs*. There are also a lot of explorations possible into the deep learning area to improvize and improve their performance in breaking different kind of webmail CAPTCHAs.

References

[1] SLIC based segmentation. http://www.jayrambhia.com/blog/superpixels-slic.

[2] Ankush Sachdeva. https://github.com/ankushsachdeva/CAPTCHA.

[3] Kavish Dahekar. https://github.com/kavishdahekar/IITG-Captcha-Solver-OpenCV-TensorFlow.

[4] Squirrelmail's code used by CC/CSE to provide webmail facility. Using this ensures that the images generated for training phase are exactly the ones used by CC/CSE.

[5] Lu, Y. (1995). Machine printed character segmentation; An overview. Pattern Recognition, 28(1), 67-80. DOI: 10.1016/0031-3203(94)00068-W.

[6] Greg Mori and Jitendra Malik. Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. CVPR'03 Proceedings of the 2003 IEEE computer society conference on Computer vision and pattern recognition.

[7] Jaderberg, A. Vedaldi, A. Zisserman. Deep Features for Text Spotting. European Conference on Computer Vision, 2014.

[8] K. Chellapilla, K. Larson, P. Simard, M. Czerwinski. Designing Human Friendly Human Interaction Proofs (HIPs) ACM's CHI 2005, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Pages 711-720.

[9] Cremers, D., Hazırbaş, C., and Stark, F. (2015). CAPTCHA Recognition with Active Deep Learning.