Solving Captchas using Optical Character Recognition

Anupam Kumar*, Vivek Modi*, Arun Shakya*, Preet Patel*
{anupam.kumar,vivek.modi,arun.shakya,preet.patel}@iitgn.ac.in
Undergraduate, Indian Institute of Technology, Gandhinagar, Gujarat, India

Abstract

Complete Automated Public Turing Test to Tell Computers and Humans Apart(CAPTCHA) is a critical human-machine distinction technology for websites to prevent the automatic malicious program attack. Although CAPTCHA recognition techniques have proven very effective against safety breaches, it also promotes some other technologies like digit and handwriting recognition. This project presents a comparative study between classical machine learning, and Neural Network approaches when deployed to predict CAPTCHAs. It also shows how good data plays a vital role in the efficacy of any machine learning model.

1 Introduction

CAPTCHA is a mechanism implemented to differentiate real users from automated ones(such as web crawlers and internet bots). Because of the rapid development of the internet industry, more and more network security issues happen; CAPTCHA has a wide range of applications in network protection and information security. Furthermore, as more technologies have evolved in machine learning, it has become easier to detect and recognize text from images. Therefore, conducting and promoting research on recognizing CAPTCHAs has become extremely important because it helps find loopholes in generated CAPTCHAs and consequently leads to safety and security.

Text-based CAPTCHAs exist in many languages, but the majority of text-based CAPTCHAs consist of English uppercase(A to Z) or lowercase letters(a to z) and numerals(0 to 9). To make the captchas more robust and secure, different techniques such as adding noise, rotating warping, text distortion, and merging of characters are used.

Machine learning algorithms did a decent job in recognizing these text-based CAPTCHAs. There are two types of algorithms that are generally used to recognize CAPTCHAs segmentation-based and segmentation-free. Segmentation-based algorithms first retrieve individual characters, and then they recognize the character. This paper has used segmentation-based algorithms to do a comparative study between classical machine learning and neural networks-based approaches to solve CAPTCHAs.

The remainder of this paper is organized as follows: Section 2 talks about the related works.Section 3 shows dataset used. Section 4 introduces our approach towards solving the CAPTCHAs. Section 5 experimental settings. Section 6 expalins our prediction and evaluation pipeline. Section 7 shows our results for different models. Section 8 talks about the limitation of our study. Finally section 9 shows our future prospects.

2 Related Work

As mentioned earlier, we have considered two main approaches to recognize captchas naming the basic Machine learning algorithms and deep learning methods. Most of the papers performed Neural network approaches, while on the other end, some used classical image processing methods to solve CAPTCHAs. For example, a sliding window approach to segment the characters and recognize them one by one. We have implemented the Support Vector Machine approach, and Convolution Neural Network approaches. The base idea for implementing the SVM approach is taken from Ondrej Bostik and Jan Klecka [ref]. The research paper has used different supervised learning approaches such as basic Neural network, k-Nearest Neighbours, Support vector machines, and Decision trees. They have used a Bubble Captcha concept to generate every character for testing OCR algorithms. It takes a two-dimensional array representing the binary grid for every character. Then they used feature extraction and various preprocessing methods to ensure proper input to the algorithms. The SVM method creates a classifier that operates based on Error-Correcting Output code(ECOC). Furthermore, the SVM binary classifier can be enhanced for multi-class classification. In the Deep-Captcha research papers by Zahra Noury and Mahdi Rezaei [ref], they have used the approach similar to the traditional CNN approach for recognition anti-recognition Captchas. They have used a convolutional layer with 32 input neurons,512 units for dense layers with the ReLU activation function, and a 5*5 kernel with 2*2 Max-pool. They also used a sigmoid function to recognize the alphanumerical-based Captchas.

In another work done by Sivakorn et al., they have created a web-browser-based system to solve image CAPTCHAs. Their system uses the Google Reverse Image Search (GRIS) and other open-source tools to annotate the images. The authors suggest different types of noise, including crossing line noise or point-based scattered noise, to improve the complexity and security of the CAPTCHAs patterns.

3 Dataset

Generally, to solve CAPTCHAs, we need two types of datasets 1. Dataset of individual characters (A to z and 0 to 9) to train our model, 2. Dataset of CAPTCHAs. We have categorized our dataset into two parts, generated and outsourced. In addition, there are two datasets that we have developed on our own :

- 1) Salt and pepper CAPTCHAs dataset: A dataset of 10k CAPTCHA images with salt and pepper noise. We have built these images by writing a text on image canvas and then introducing salt and pepper noise by changing 2-5% of image pixels.
- 2) Color noise trans dataset is given below

Apart from this dataset, we have also used the char74 dataset, which contains colored fonts of different sizes and styles. We have analyzed the performance of three different models: SVM, fully connected Neural Network, and CNN for comparative study. The main contribution of this paper include:

We propose a comparative study between different segmentation based CAPTCHAs models.

Captchas Datasets are given below

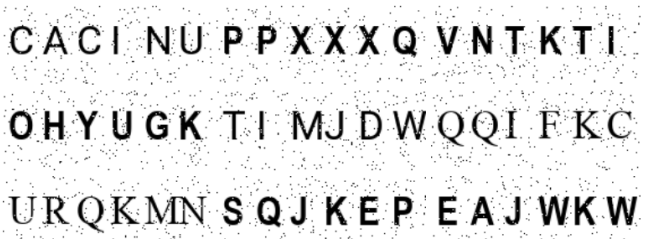

Figure 1. Salt and pepper noise

Figure 2. Captcha without noise

4 Methodology

4.1 Preprocessing

The first step in recognizing captchas is to preprocess the given data to get valuable information. In this part, we will take the individual captchas and use various approaches to modify their quality and fetch important information. Here we have used methods like sampling, dimensionality reduction, etc. We have converted each image to gray. We also applied threshold and contours to the input image. Along with this, we also used various methods such as resize and inversion to finally to width 185 and height 70 pixels. Then we will perform segmentation.

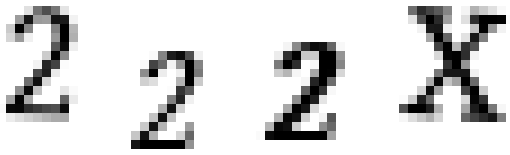

Figure 3. Rdmpage Captcha

Figure 4. Char74 for single character dataset

Figure 5. Single character without noise

Figure 6. Coloured and noisy for single character

4.2 SVM Implementation

After Character segmentation to produce separate characters, We have used Scikit-learn's Support vector Machine model to classify the preprocessed characters. We have used the SVM binary classifier that uses one to one coding design for every combination of two classes where one classifier is positive, one is negative while the rest are neglected. Therefore, the number of binary learners for K classes is K(K-1)/2. The above two classes will represent two hyperplanes for SVM. The Hinge Loss between the planes are - We will then compute

$$\text{Hingeloss} = \max(0, (1 - y_i * (w^T * x_i + b))/2$$

Figure 7. SVM Hingeloss

the gradient of cost function by following -

$$\nabla_w J(\mathbf{w}) = \frac{1}{N}\sum_i^n \begin{cases} \mathbf{w} & \text{if } \max(0,\ 1 - y_i * (\mathbf{w} \cdot x_i)) = 0 \\ \mathbf{w} - C y_i\, x_i & \text{otherwise} \end{cases}$$

Figure 8. SVM gradient of cost function

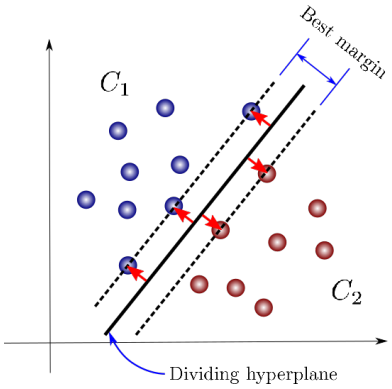

Figure 9. SVM

4.2.1 Fully connected Neural Network.

The input image is directly fed to the fully connected neural network. All three hidden layers have activation ReLU, while the last layer has softmax activation.

4.2.2 CNN Implementation.

In a fully connected neural network, the network learns to extract features from the input independently. This task is computationally heavy and requires more memory. To solve this problem, we use CNN. The convolutional layers and max pool layers allow us to extract the features with more efficiency, and less memory is required for this task. Furthermore, after these features are extracted, applying fully connected layers gives us better performance even with a few layers.

The input image is passed through a convolutional layer with filter size = 1, kernel size = 5, and activation ReLU,

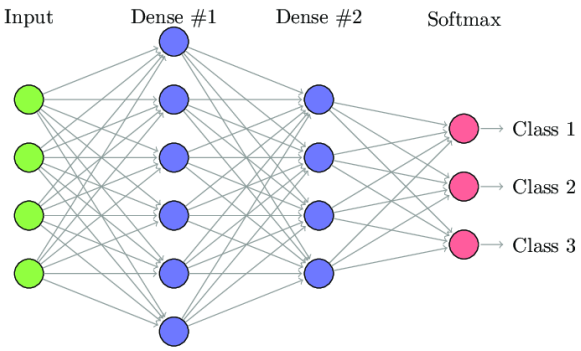

Figure 10. Fully Connected Neural Network

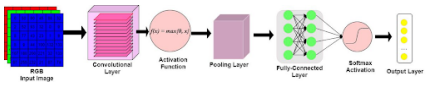

Figure 11. cnn Implementation

Layer (type)	Output Shape	Param #
conv2d_54 (Conv2D)	(None, 32, 32, 1)	26
max_pooling2d_18 (MaxPooling	(None, 16, 16, 1)	0
conv2d_55 (Conv2D)	(None, 16, 16, 1)	2
up_sampling2d_18 (UpSampling	(None, 32, 32, 1)	0
conv2d_56 (Conv2D)	(None, 32, 32, 1)	2
flatten_18 (Flatten)	(None, 1024)	0
dense_108 (Dense)	(None, 1024)	1049600
dropout_90 (Dropout)	(None, 1024)	0
dense_109 (Dense)	(None, 512)	524800
dropout_91 (Dropout)	(None, 512)	0
dense_110 (Dense)	(None, 256)	131328
dropout_92 (Dropout)	(None, 256)	0
dense_111 (Dense)	(None, 128)	32896
dropout_93 (Dropout)	(None, 128)	0
dense_112 (Dense)	(None, 64)	8256
dropout_94 (Dropout)	(None, 64)	0
dense_113 (Dense)	(None, 26)	1690

Figure 12. Our CNN model

followed by max pool of size (2,2). To further extract the features, a second convolutional layer is used with kernel size = 1 (to keep the image shape intact), stride = 1, and activation ReLU. Furthermore, upsampling is done to regain the original dimensions. Now, these extracted features are passed through the fully connected layers with activation ReLU (except for the last layer). This allows the model to learn and classify the characters among 26 letters. The last layer has an activation softmax to obtain the output in the range [0,1]; it is commonly used for multi-classification. When all the features are connected to the fully connected layer, it

can cause overfitting in the training dataset. To overcome this problem, a dropout layer is added where a few neurons are dropped from the neural network during the training process resulting in a reduced size of the model. After adding a dropout of 0.1, 10% of the nodes are dropped out randomly from the neural network.

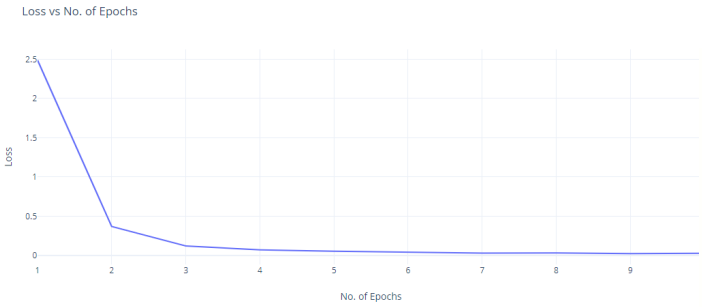

Figure 13. Loss vs Epochs

5 Segmentation

After inverting the image, we use cv2.findContours() to find the distinct contours in the image. Due to noise or distortion in the image, it might be possible that some small contours are also detected. Therefore, we specify the following parameters for selecting the most probable characters: min-height, max-height, min-width, max-width.

6 Prediction and Evaluation Pipeline

We first convert the image into a grayscale image. Then, we apply simple thresholding using cv2.THRESH_BINARY to remove any noise from the image, which will help us in segmenting the images. After thresholding, we get a binary image. After this, we use cv2.findContours() to find the distinct contours in the image. Due to noise or distortion in the image, it might be possible that some small contours are also detected. We specify the following parameters for selecting the most probable characters: min_height, max_height, min_width, max_width.

We will add some arbitrary border into this extracted single character image. Then, we will reshape this image and convert it into an array of numbers. And we are ready to dump this into model.predict().

model.predict() gives us an array of probabilities for each of the possible targets(these are 26 in total, corresponding to each alphabet). First, we extract the argument with the highest probability and use this as the predicted character. Next, we sort these predicted characters using their position on the x-axis. Finally, we combine the individual characters into a string, and our predicted captcha is ready.

We are computing two types of accuracies that are interrelated. The first accuracy is Character-accuracy which represents the percentage of characters correctly predicted by the model. The second one is the Captcha-accuracy which represents the percentage of correctly predicted captchas predicted by our model. These two are related, but we consider Captcha-accuracy as the final result.

7 Experimental Results

7.1 SVM

Captchas: Salt-n-pepper noise dataset (Self-generated)
Single Characters: Without noise dataset (Self-generated)

```
Total images processed: 10000
Accuracy:  60.08
WCITEB  :  WCTEB  :  False
```

Captchas: Without noise dataset (Self-generated)
Single Characters: Without noise dataset (Self-generated)

```
Total images processed: 10000
Accuracy:  73.99
PRQVTU  :  PRQVTU  :  True
```

Figure 14. SVM results

SVM Results
Insights: There is a 14% drop in accuracy when minor noise is introduced. This shows that the model is highly sensitive to variation/noise in captchas. Hence, SVM implementation (classical machine learning) is less preferred for character classification. Need: A better learn-able model is required. Solution: Neural Network based models

7.2 Fully-Connected Neural Network

Captchas: Without noise dataset (Self-generated)
Single Characters: Chars74k dataset (Outsourced)

	Pos	Neg	Acc
Char	43548	13407	76.460363
Captcha	3152	6848	31.520000

```
Current file:  10000
FXYQIR : PXYQR
```

Captchas: Rdmpage dataset (Outsourced)
Single Characters: Chars74k dataset (Outsourced)

	Pos	Neg	Acc
Char	24625	14763	62.519041
Captcha	1789	8166	17.970869

```
Current file:  9955
7M5T : 7M5E
```

Figure 15. Fully Connected Neural Network Result 1

Fully Connected Neural Network Result 1
Insights: As we have seen earlier, the chars74k dataset, rdmpage dataset, and without noise dataset for captchas have

completely different character font styles and sizes. This results in a lot of variance in the dataset, hence decreasing the accuracy. Need: To balance the bias/variance trade off in the datasets. Solution: To add similar font styles for both datasets (captchas and single characters). After considering the above results, we re-generated the datasets and new results are as given below.

Fully Connected Neural Network Result 2

Captchas: Without noise dataset (Self-generated)
Single Characters: Without noise dataset (Self-generated)

	Pos	Neg	Acc
Char	49044	7860	86.187263
Captcha	6554	3446	65.540000

```
Current file:  10000
FXYQIR : FXYQR
```

Captchas: Salt-n-pepper dataset (Self-generated)
Single Characters: Without noise dataset (Self-generated)

	Pos	Neg	Acc
Char	4606	1142	80.13222
Captcha	449	551	44.90000

```
Current file:  1000
JFVCTQ : JFVCTQ
```

Figure 16. Fully Connected Neural Network Result 2

Insights: The accuracy was significantly improved. The captcha accuracy is still low for the noisy dataset. Need: Better pre-processing of datasets. Solution: Keep pre-processing same for both the captchas and single character datasets

Fully Connected Neural Network Result 3

Captchas: Salt-n-pepper dataset (Self-generated)
Single Characters: Coloured & Noisy dataset (Self-generated)

	Pos	Neg	Acc
Char	48972	8368	85.406348
Captcha	6383	3617	63.830000

```
Current file:  10000
ZJPVCH : ZJPVCH
```

Figure 17. Fully Connected Neural Network Result 3

Insights: The accuracy has considerably improved, but the approach is computationally heavy and requires a lot of memory. Need: Reduce the computational complexity and make the approach efficient. We need better feature extraction. Solution: Use CNN based approach for feature extraction followed by fully-connected nn.

Captchas: Salt-n-pepper dataset (Self-generated)
Single Characters: Coloured & Noisy dataset (Self-generated)

	Pos	Neg	Acc
Char	48459	8881	84.511685
Captcha	5849	4151	58.490000

```
Current file:  10000
REWBDL : REWBDL
```

Figure 18. CNN Results

7.3 Convolution Neural Network

CNN Results

Insights: The computational power required was less for the accuracy obtained. Only 3 convolutional layers were used. Need: An end to end architecture to solve captchas

8 Limitations

Our implementation cannot process anti-segmentation CAPTCHAs due to the box-based segmentation algorithm.

9 Future Work

Using our boxed segmentation method we could not recognize the anti-segmented captchas (captcha with connected characters) with good accuracy. An End-to-end CNN based model could be developed to give better results, that will be computationally heavy.

References

[1] https://github.com/rdmpage/solving-captchas-code-examples/tree/master/g

[2] http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/

[3] Sivakorn, Suphannee, Iasonas Polakis, and Angelos D. Keromytis. "I am robot:(deep) learning to break semantic image captchas." In 2016 IEEE European Symposium on Security and Privacy (Euro SP), pp. 388-403. IEEE, 2016

[4] Ye, Guixin, Zhanyong Tang, Dingyi Fang, Zhanxing Zhu, Yansong Feng, Pengfei Xu, Xiaojiang Chen, and Zheng Wang. "Yet another text captcha solver: A generative adversarial network based approach." In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 332-348. 2018

[5] Osadchy, Margarita, Julio Hernandez-Castro, Stuart Gibson, Orr Dunkelman, and Daniel Prez-Cabo. "No bot expects the DeepCAPTCHA! Introducing immutable adversarial examples, with applications to CAPTCHA generation." IEEE Transactions on Information Forensics and Security 12, no. 11 (2017): 2640-2653

[6] Kwon, Hyun, Yongchul Kim, Hyunsoo Yoon, and Daeseon Choi. "Captcha image generation systems using generative adversarial networks." IEICE TRANSACTIONS on Information and Systems 101, no. 2 (2018): 543–546