

**CRACKING CAPTCHA**  
**COMPLETELY AUTOMATED PUBLIC TURING TEST TO**  
**TELL COMPUTERS AND HUMANS APART**

**BY**

**KANIKA SINGH-101443**



**JULY ( 2013 ) - MAY ( 2014 )**

**PROJECT SUPERVISOR- MR. SUMAN SAHA**

**Submitted in partial fulfillment of Degree of Bachelor of  
Technology**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING  
AND INFORMATION TECHNOLOGY**

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY**

## **CERTIFICATE**

This is to certify that the work entitled-“**BREAKING CAPTCHAS-Completely automated public turing test to tell Computers and Humans Apart**” submitted by **Kanika Singh-101443**, in fulfillment for award for degree of Bachelor of Technology in Information Technology of JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY has been carried out under my supervision. This work has not been submitted partially or wholly to any other University for any award of this or any other degree.

Mr. Suman Saha

Senior Lecturer

Department of Computer Science Engineering and Information Technology

Jaypee University of Information Technology

Waknaghat

## Acknowledgement

*“It is not possible to prepare a project without the assistance & encouragement of other people. This one is certainly no exception.”*

On the very outset of this report, we would like to extend our sincere & heartfelt obligation towards all the people who have helped us in this endeavor. Without their active guidance, help, cooperation & encouragement, we would not have made headway in the project.

We would like to show our greatest appreciation to Mr. Suman Saha. We feel motivated everytime we get his encouragement. For his coherent guidance throughout the tenure of the project, we feel fortunate to be taught by Mr. Suman Saha, who gave us his unwavering support. Besides being our mentor, he taught us that there is no substitute for hardwork.

We owe our heartiest thanks to Brig. (Retd.) S.P. Ghrrera (HOD, CSE/IT Department) and Dr. Hemraj Saini (BTech Project Coordinator) who has always inspired us to take initiatives.

In the light of new developments and recent findings, we devote the task that was asked from us at Jaypee University of Information Technology to **BREAKING CAPTCHA- Completely automated public turing test to tell Computers and Humans Apart”**.

Kanika Singh-101443

## Table of Contents

TOPIC	PAGE NO.
CERTIFICATE	I
ACKNOWLEDGEMENT	II
TABLE OF CONTENTS	III
LIST OF FIGURES	V
LIST OF ABBREVIATIONS	VI
ABSTRACT	VII
CHAPTER I	
INTRODUCTION	8-11
What are captchas ?	8
1. History	8
2. Need for CAPTCHA	9
3. ReverseTuringTest	11
CHAPTER II	
PROBLEM STATEMENT AND MOTIVATION	12-13
CHAPTER III	
LITERATURE SURVEY	14-26

1. A Low Segmentation attack	14-22
• Binarization	16
• Fixingbroken characters	17
• VerticalSegmentation	18
• ColorFilling Segmentation	18
• Thickarcremoval	20
• LocatingConnectedCharacters	21
• SegmenConnected Characters	22
2. CHELLAPILLA ALGORITHM	23-27
• Projection	23
• Proposedalgorithm	25
3. Machine learning in NN and SVM	26

## CHAPTER IV

PROPOSED ALGORITHM AND IMPLEMENTATION	28-61
1. Proposed Algorithm	28
2. Code	34
3. Output	49
4. Results after Testing	50
5. Analysis	51

References	53
------------	----

## **List of Figures**

Figure 1: CAPTCHA Image

Figure 2: Recognition rate for individual characters using different distortions

Figure 3: Samples in MSN scheme

Figure 4: Binarization

Figure 5: Connection one pixel gap

Figure 6: Vertical segmentation

Figure 7: 8-connectivity v/s 4-connectivity

Figure 8: CFS segmentation

Figure 9: Circle detection

Figure 10: Discriminative feature checking

Figure 11: Relative feature checking

Figure 12: “Approximation” for locating connected characters

Figure 13: Problems in Chellapilla’s algorithm

Figure 14: Examples of clutter and projection

Figure 15: Algorithm proposed

Figure 16: Types of CAPTCHAs

Figure 17: System Architecture

Figure 18: Starting the application

Figure 19: Loading CAPTCHA and saving output

Figure 20: Text file saved

Figure 21: Some of the sample CAPTCHAs

Figure 22: SVM mode

## List of abbreviations

**BMP:** Bit map

**CAPTCHA:** Completely Automated Public Turing Tests to Tell Computers and Humans Apart

**E-Commerce:** Electronic Commerce

**E-mail:** Electronic Mail

**GIF:** Graphical Interchange File format

**IEEE:** Institute of Electrical and Electronics Engineers

**JPEG:** Joint photographic expert group

**NN:** Neural Networks

**OCR:** Optical Character Recognition

**PNG:** Portable network graphics

**SVM:** Support vector machine

**TIFF:** Tag image file format

**Txt:** Text file format

## **Abstract**

Completely Automated Public Turing Tests to Tell Computers and Humans Apart (CAPTCHAs) are the automatic filters that are widely used these days to disallow any automated script that can perform the work of a human. CAPTCHAs are built in such a way that it is very difficult for any automated script to break them. The state of the art of CAPTCHA design suggests that such text-based schemes should rely on segmentation resistance to provide security guarantee, as individual character recognition after segmentation can be solved with a high success rate by standard methods such as neural networks. We analyse the security of a text-based CAPTCHAs and the loopholes in designing of these captchas. Defeating a CAPTCHA test requires two procedures: segmentation and recognition. In this project, an approach to break text based CAPTCHAs has been proposed that first preprocesses the given CAPTCHA, segments its characters, and then recognizes the characters depending on it's features. The breaking of CAPTCHAs give strength to CAPTCHAs which in turn help to develop more robust CAPTCHAs.



# Chapter-I

## Introduction

### What are CAPTCHAs?

CAPTCHAs are a computer program or system intended to distinguish human from machine input, typically as a way of thwarting spam and automated extraction of data from websites. It stands for **C**ompletely **A**utomated **P**ublic **T**uring **T**est to **T**ell **C**omputers and **H**uman **A**part, and Public means that the code and the data used should be publicly available. A more technical definition of CAPTCHA is : “CAPTCHA is a cryptographic protocol whose underlying hardness assumption is based on an AI problem”.

### History

Since the early days of the Internet, users have wanted to make text illegible to computers. The first such people were hackers, posting about sensitive topics to online forums they thought were being automatically monitored for keywords. To circumvent such filters, they would replace a word with look-alike characters. *HELLO* could become |3|\_|\_() or )-(3££0, as well as numerous other variants, such that a filter could not possibly detect *all* of them. This later became known as **leetspeak**.

The first discussion of automated tests which distinguish humans from computers for the purpose of controlling access to web services appears in a 1996 manuscript of **Moni Naor** from the Weizmann Institute of Science, entitled "Verification of a human in the loop, or Identification via the Turing Test".

Subsequent to that work, two teams of people have claimed to be the first to invent the CAPTCHAs used throughout the Web today. The first team consists of **Mark D. Lillibridge, Martin Abadi, Krishna Bharat, and Andrei Z. Broder**, who used CAPTCHAs in 1997 at AltaVista to prevent bots from adding URLs to their search engine. Looking for a way to make their images resistant to OCR attack, the team looked at the manual of their Brother scanner, which had recommendations for improving OCR's results (similar typefaces, plain backgrounds, etc.). The team created puzzles by attempting to simulate what the manual claimed would cause bad OCR.

The second team to claim inventorship of CAPTCHAs consists of **Luis von Ahn and Manuel Blum**, who described CAPTCHAs in a 2003 publication and subsequently received much coverage in the popular press. Their notion of CAPTCHA covers any program that can distinguish humans from computers, including many different examples of CAPTCHAs.

The controversy of inventorship has been settled by the existence of a 1998 patent by **Lillibridge, Abadi, Bharat, and Broder**, which predates other publications by several years. Though the patent does not use the term CAPTCHA, it describes the ideas in detail and precisely depicts the graphical CAPTCHAs used in the Web today.

### **Need for CAPTCHAs**

CAPTCHAs are used to protect many types of websites including free-email providers, ticket sellers, social networks, wikis and blogs. Free services on the internet may be abused by automated computer programs (often referred to as scripts or bots – here, we use bot). Such bots may be intended to broadcast junk emails, post advertisements, or ask the server to respond at a very high frequency. All of these forms of misuse will decrease the usefulness of internet services. To prevent such abuse, CAPTCHAs have been designed.

**(A) Online Polls.** In November 1999, slashdot.com released an online poll asking which was the best graduate school in computer science (a dangerous question to ask over the web!). As is the case with most online polls, IP addresses of voters were recorded in order to prevent single users from voting more than once. However, students at Carnegie Mellon found a way to stuff the ballots by using programs that voted for CMU thousands of times. CMU's score started growing rapidly. The next day, students at MIT wrote their own voting program and the poll became a contest between voting bots". MIT finished with 21,156 votes, Carnegie Mellon with 21,032 and every other school with less than 1,000. Can the result of any online poll be trusted? Not unless the poll requires that only humans can vote.

**(B) Free Email Services.** Several companies (Yahoo!, Microsoft, etc.) free email services, most of which suffer from a specific type of attack: bots that sign up for thousands of email accounts every minute. This situation can be improved by requiring users to prove they are human before they can get a free email account. Yahoo!, for instance, uses a CAPTCHA of our design to prevent bots from registering for accounts. Their CAPTCHA asks user to read a distorted\_ word ( current computer programs are not as good as humans at reading distorted text).

**(C) Search Engine Bots.** Some web sites don't want to be indexed by search engines. There is an html tag to prevent search engine bots from reading web pages, but the tag doesn't guarantee that bots would not read the pages; it only serves to say \no bots, please. However, in order to truly guarantee that bots won't enter a web site, CAPTCHAs are needed.

**(D) Worms and Spam:** CAPTCHAs also offer a plausible solution against email. CAPTCHA guarantees an email acceptance only if you know there is a human behind the other computer. A few companies, such as [www.spamarrest.com](http://www.spamarrest.com) are already marketing this idea.

**(E) Preventing Dictionary Attacks.** Pinkas and Sander have suggested using CAPTCHAs to prevent dictionary attacks in password systems. The idea is simple: prevent a computer from being able to iterate through the entire space of passwords by requiring a human to type the passwords.

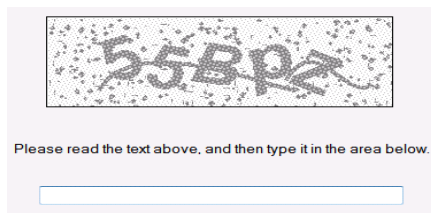


Figure 1 : CAPTCHA Image

### **Reverse Turing Test**

- The process usually involves one computer ( a server ) asking a user to complete a simple test which the computer is able to generate and grade.
- Thus, it is described as a **Reverse Turing Test** because it is administered by a machine and targeted to a human.
- In contrast to standard Turing test that is typically administered by human and targeted to a machine.

## Chapter-II

### Problem Statement and Motivation

Considering the recent news that the hotmail and yahoo image CAPTCHAs would be gamed, it is important to identify the weaknesses in the present day CAPTCHAs and build more robust CAPTCHAs. A good CAPTCHA must be not only human friendly, but also robust enough to resist to computer programs that attackers write to automatically pass CAPTCHA tests (or challenges).

Early research suggested that computers are very good at recognising single characters, even if these characters are highly distorted. Figure 2 shows characters under typical distortions, along with success rates that a neural network can achieve to recognise them. It is established that if the positions of characters are known in challenge images generated by a CAPTCHA, then breaking this scheme is just a pure recognition problem, which is a trivial task with standard machine learning techniques such as neural networks and SVM.







Characters under typical distortions	Recognition rate
	~100%
	96+%
	100%
	98%
	~100%
	95+%

Figure 2. Recognition rate for individual characters under different distortions

However, when the location of characters in a CAPTCHA challenge is not known a-priori, state of the art (including machine learning) methods do not work well in locating the characters, let alone recognising them.

The state of the art of CAPTCHA design suggests that the robustness of text-based schemes

should rely on the difficulty of finding where the character is (segmentation), rather than which character it is (recognition) . That is, such CAPTCHAs should be *segmentation-resistant*. In other words, *if breaking a (text-based) CAPTCHA can be successfully reduced to a problem of individual character recognition, then this scheme is effectively broken*.

So, we have divided our project into two parts:

- I. **Segmentation**- In this, we use different techniques to find the location of the characters.
- II. **Recognition** - In this section, we train our machine to recognize various characters even with distortion using machine learning like NN, SVM.

Hence, by recognizing the loopholes in the present day CAPTCHAs, we can suggest methods to develop more robust CAPTCHAs and hence, improve security and prevent bots from various malicious activities.

## **Chapter-III**

### **Literature Survey**

#### **A low cost segmentation attack**

A simple, low-cost segmentation attack that has achieved a success rate of higher than 90% on the latest version of this Microsoft CAPTCHA. With the aid of this segmentation attack, we estimate that the MSN scheme can be broken with an overall (segmentation and then recognition) success rate of about 60%. In contrast, its design goal was that “automatic scripts should not be more successful than 1 in 10,000” attempts (i.e., a success rate of 0.01%) [4]. In fact, although the MSN scheme was believed to be “extremely difficult and expensive for computers to solve” because of the difficulty of segmentation that its designers introduced, it takes only slightly more than 80ms in average for our attack to completely segment a challenge on a desktop computer. It shows that a CAPTCHA that was carefully designed by serious professionals to be segmentation-resistant is nevertheless vulnerable to novel but simple attacks.

The attack achieves the following:

- Identify and remove random arcs
- Identify all character locations in the right order. Specifically, divide each challenge into 8 ordered segments, each containing a single character.

Some of the sample challenges in this scheme were:

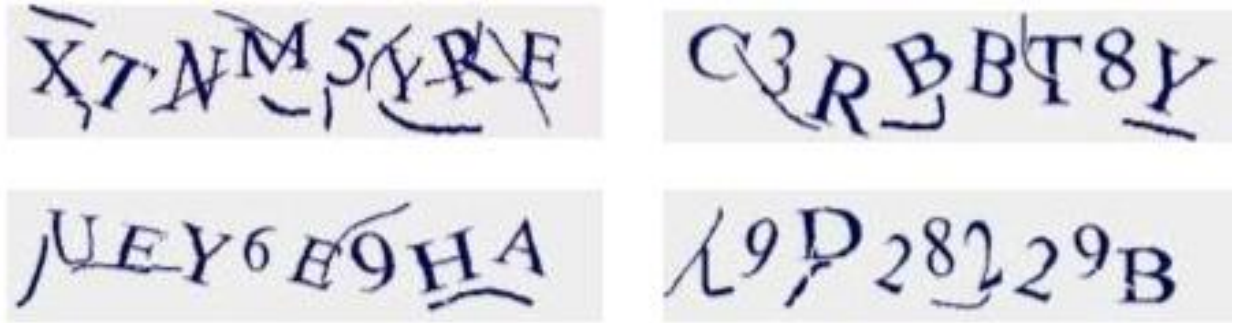


Figure 3. Samples used in MSN scheme

Eight characters are used in each challenge;

- Only upper case letters and digits are used.
- Foreground (i.e. challenge text) is dark blue and background light gray.
- Warping (both local and global) is used for character distortion.

Local warp produces “small ripples, waves and elastic deformations along the pixels of the character”, and it foils “feature-based algorithms which may use character thickness or serif features to detect and recognise characters” .

Global warp generates character-level, elastic deformations to foil template matching algorithms for character detection and recognition.

The following random arcs of different thicknesses are used as the main anti-segmentation measure.

- ❖ **Thick foreground arcs:** These arcs are of foreground color. Their thickness can be the same as the thick portions of characters. They do not directly intersect with any characters, so they are also called “non-intersecting arcs”.
- ❖ **Thin foreground arcs:** These arcs are of foreground color. Although they are typically not as thick as the above type of arcs, their thickness can be the same as the thin portions of characters. They intersect with thick arcs, characters or both, and thus also called “intersecting thin arcs”.



- ❖ **Thin background arcs:** These arcs are thin and of background color. They cut through characters and remove some character content (pixels).

Both local and global warping is commonly used for distortion in text-based CAPTCHAs.

What is special in the design of the MSN scheme is the following: in contrast to many other schemes that use background textures and meshes in foreground and background colors as clutter to increase robustness, random arcs of different thicknesses are used as clutter in this scheme. The idea is that these arcs are themselves good candidates for false characters, and therefore such a design was expected to provide strong segmentation resistance. That is, current computer programs would fail to segment the distorted text into individual characters due to the introduction of random arcs.

### Step1. Binarization

We first convert a color challenge to a two-color image using a threshold method: pixels with intensity higher than a certain threshold value are converted to white, and those with a lower intensity to black.

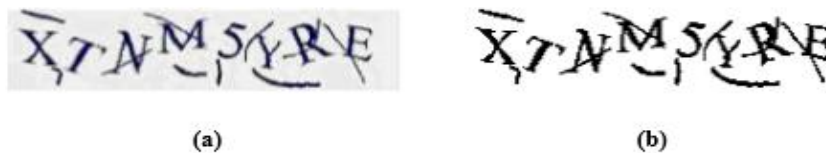


Figure 4. a) Original Image b) Binarized Image

## Step 2. Fixing broken characters

Thin background arcs remove some character content, and sometimes they also create a cracking characters. For example, the second character ('T') in Fig 5 is broken due to this reason.

The current step attempts to fix broken characters for two purposes: i) to keep a character as a single entity and consequently enhance our follow-up segmentation methods, and ii) to prevent small portions of characters from being removed as an arc later on.

We observed that thin background arcs are typically 1-2 pixels wide after binarization, and the following simple method works well to identify and fix broken characters caused by such arcs.

- ✓ Find pixels that are of background color and have left and right neighbours with foreground color .
- ✓ Find pixels that are of background color and have top and bottom neighbours with foreground color .
- ✓ Convert pixels identified above to foreground color.

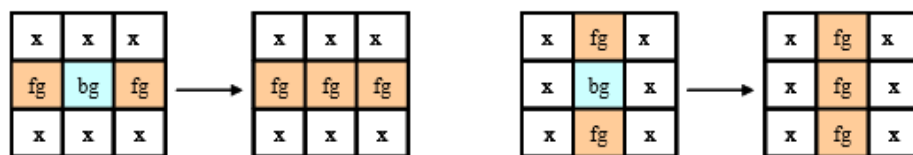


Figure 5. Connecting one pixel gap

### Step 3. Vertical segmentation

A vertical segmentation method is applied to segment a challenge vertically into several no. of *chunks*, each of which might contain one or more characters. The process of vertical segmentation starts by mapping the image to a histogram that represents the number of foreground pixels per column in the image. Then, vertical segmentation lines separate the

image into chunks by cutting through columns that have no foreground pixels at all. Fig

6

shows that such vertical histogram segmentation cuts challenge (a) into two chunks, and the

other (b) into five.

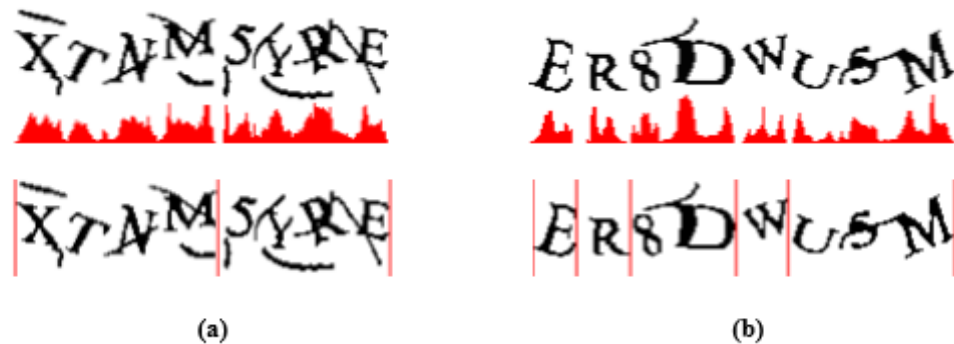


Figure 6. Vertical Segmentation

### Step4. Color filling segmentation

In this step, a “color filling segmentation (CFS)” algorithm is applied to *each* chunk segmented in the previous step. The basic idea of this algorithm is to detect every connected

component, which we call an *object*, in a chunk. An object can be an arc, character, connected arcs, or connected characters. The algorithm works as follows. First, detect a foreground pixel, and then trace all its foreground neighbours until all pixels in this connected component are traversed – that is, an object is detected. Next, the algorithm locates a foreground pixel outside of the area of the detected object(s), and starts another traversal process to identify a next object. This process continues until all objects in the chunk are located. This method is effectively like using a distinct color to flood each connected component, so we call it the “color filling” segmentation. In the end, the number of colours used to fill a chunk is the number of objects in the chunk.

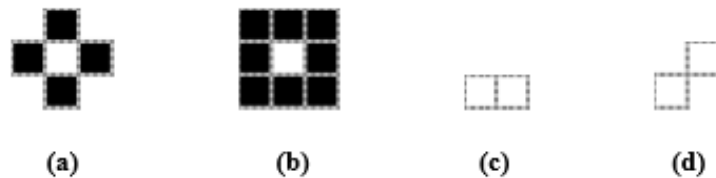


Figure 7. 8-connectivity v/s 4-connectivity

- a) 4-connected b) 8-connected c) 8-connected but not 4-connected d) both 8-connected and 4-connected

With our CFS method, as shown in Fig 8, we determine that there are six objects in the first chunk and five in the second.



Figure 8. CFS segmentation

## Step 5. Arc Removal

- 1) **Circle detection**, which detects if an object contains a circle. If an object contains a circle, we know it is definitely not an arc, and all other arc removal methods can be skipped.

The circle detection method works as follows.

- Draw a bounding box around an object, so that this bounding box does not touch any part of the object.
- Apply the color filling algorithm to the top-left pixel, i.e., flood all background pixels that are connected to the top-left pixel, with a color that is different from foreground and background .
- Scan the bounding box for pixels of the background color. If such a pixel is found, then a circle is detected. Otherwise, no circle is detected.

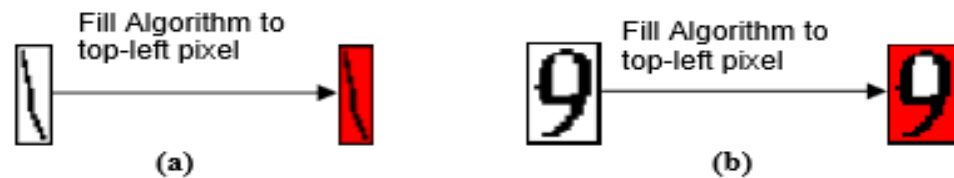


Figure 9. a) No circle detected b) Circle detected

- 2) Scan all objects that contain **no circles for discriminative features** (other objects are ignored).

Such discrimination is largely about pixel count checking.

If an object has a pixel count smaller than or equal to 50, it is removed as an arc.



Fig 10. Arc removal - discriminative feature checking. (a) output from color filling; (b) an arc in the second chunk is removed.

3) **Relative position checking.** This step examines the relative position of objects in a chunk, and is applied to all chunks that contain more than one object.

The basic idea behind this step is that the relative positions of objects can tell arcs and real characters apart.



Fig 11. Arc removal - relative position checking. As shown in (b), further arcs in (a) are removed and histogram is updated.

### Step 6. Locating Connected Components

After removing arcs, an immediate step is to locate, if any, connected characters, which either vertical or color filling segmentation has failed to segment. Among  $n$  objects output by the previous step, if  $n < 8$ , then at least one of the objects contains two or more characters and these characters are connected (typically by thin intersecting arcs). This step estimates how many characters are connected and locates them.



Figure 12. "Approximation" for locating connected characters

### Step 7. Segment Connected Components

The previous step has identified any object(s) containing connected characters, as well as the

number of these characters, denoted by  $c$ , contained in each object. We observed that often, a

simple method works to segment the connected characters in an object as follows.

- 1) Work out the width of the object by identifying its left-most and right-most pixels;
- 2) Vertically divide the object into  $c$  parts of the same width, each part being a proper segment.

### Analysis

Usability of this MSN scheme is reasonably good. For example, characters are not so distorted as to damage their recognisability by most human users. One particular good usability feature is that this scheme does not significantly disadvantage people whose mother

tongue does not use the Latin alphabet. In some schemes, characters are distorted to be similar to handwriting - native speakers might find it easy to recognise them, but just imagine how difficult it would be for a user to recognise handwriting in a language that she knows nothing about.

## CHELLAPILLA’S ALGORITHM

The algorithm design includes preprocessing, image opening and labeling (three phases) to defeat Yahoo’s CAPTCHA system. The preprocessing phase includes thresholding and up-sampling – first, converting the original image into a two colored image, and then enlarging it. Image opening is the key step allowing segmentation of the characters from CAPTCHA images. In this phase, the preprocessed image will go through an erosion process several times and will then be dilated several times. Erosion will erase the character borders one pixel per time, whereas dilation will mend the borders one pixel per time. Once the thin clutter items have been deleted by the erosion process, they no longer appear following the dilation process, resulting in some items of clutter being deleted. The labeling phase then finds all of the connected components in the image, and considers the larger ones as characters. Since this phase only outputs the larger items as its result, small connected components will be considered to be clutter and will be eliminated in this phase. This algorithm is useful when the clutter is of thinner width than the characters. But, this algorithm produces errors because it can not recognize the difference between characters and clutters of similar width. The algorithm may categorize the clutter as character data, so the clutters will not be deleted. An example is shown in Fig. 13(a) and Fig. 13(b), where “S” and “8” are still connected and “G” and “H” have the same problem.

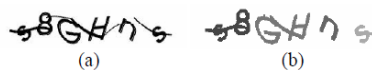


Figure 13. Some problems in Chellapilla’s algorithm

### **III. PROPOSED SEGMENTATION ALGORITHM**

Chellapilla et al. gave the research community an effective way to address the recognition problem, but their segmentation algorithm does not represent a complete solution. This paper will therefore now propose a novel techniques - *projection* - that is intended to improve the success rate of segmentation, and which yield a more effective segmentation algorithm.

#### **A. Projection**



The projection technique in this paper is based upon the idea of projecting the image data onto the X-axis. In practice, this is implemented by summing the number of non-white pixels in each column of the image parallel to the Y-axis. Fig. 14 shows some example of clutter and their corresponding projections. It's easily seen that the projections of these clutter items onto the X-axis appear smaller and flatter than a normal character's projection onto the X-axis. The projection in the X-axis will tend to appear large and unstable, when a component represents a character rather than an item of clutter.

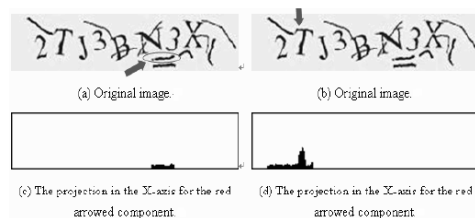


Figure 14. Examples of Clutter and its projection Image

Therefore, by computing a component's projection value and its variance value it is possible to differentiate between components that are clutter and components that are characters. Fig. 15(a) gives another type of clutter which is intersected by other characters and forms a part of a larger component. This type of clutter also has a smooth and small appearance when projected into the X-axis, as shown in Fig. 15(b). Therefore, it is possible to use the projection technique to find out the position of the clutter within a component, making it more straightforward to clean up the component.



When two or more characters are connected by this form of clutter, they can be effectively split up by deleting these clutter items. It uses a sliding window to this type of clutter, because it has a smaller projection size than a normal character for a small part of the image. In other words, the X-axis projection value of these clutter items will be smaller than some threshold for a part of the image, so it is possible to use the sliding window to check the projection value continually. When the projection values in the sliding window are smaller than the threshold, the algorithm marks the position as containing a clutter

item, so that it may be erased. Fig. 15(b) gives an example of the operation of the sliding window approach with the above clutter. Suppose the width of the sliding window, and the threshold, are both 5. When the sliding window moves to the edge of the “E” and the “5”, the projection values in the sliding window are not all smaller than the threshold, so no action will be taken at this position. When the sliding window moves to the edge of the “5” and the “K”, all of the X-axis projection values are smaller than the threshold, so the algorithm will mark this place as containing a clutter, and clean it from the image. After the cleaning process, the connection between the “5” and the “K” characters is removed, and these characters are split into separate components. Note that the projection values of some characters, such as “0”, “O”, “D”, “8”, or “B”, can also be smaller than the threshold continually over a region of the image, resulting in this character being damaged by the decisions made by the projection method. Fortunately, these characters have closed regions within them, containing the background color. This property can be utilized to avoid damaging the character by mistake.

### B. Proposed Algorithm .

This algorithm is based on Chellapillas et al.’s algorithm and has five phases, which are preprocessing, opening, labeling, projection and character extracting. The behavior is shown in Fig. 16. The first three phases have a similar process for these of Chellapilla’s algorithm except the preprocessing phase.

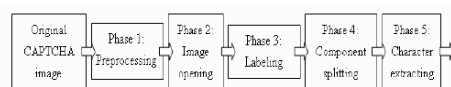


Figure 15

To prevent the mistake generated by the projection technique, the algorithm will detect closed regions in the preprocessing phase. It begins by computing all of the background-colored connected components first. The background-colored connected components which do not belong to a closed region will combine together, and become the largest connected component in the picture. The algorithm marks the connected components that are smaller than the largest component, as closed regions, which will not be considered by the sliding

window technique later. The proposed projection technique is employed in the fourth phase. After operations of this phases, the original image has separated into many different connected components. The final phase, the character extracting phase, deletes the redundant components, and outputs the location of the characters. It is known that the characters have the biggest and most unsmooth projection values in the X-axis, so the algorithm erases the components which have small and smooth projection values. The remaining components are sorted by their size; and the algorithm outputs the largest 8 components (for MSN CAPTCHAs), or the largest 6 components (for Yahoo CAPTCHAs, which typically have 5-6 components).

### **Neural Networks**

**Artificial neural networks** are computational models inspired by animal central nervous systems (in particular the brain) that are capable of machine learning and pattern recognition. They are usually presented as systems of interconnected "neurons" that can compute values from inputs by feeding information through the network.

The neural net approach utilized three separate steps. The first step simply translated the binary character data into a friendlier form. The second step took the output of the first and trained a backpropagation network on it, outputting all the resulting weights and general network information. The third step took the output of the second and created a network. It then ran a full character set through the network and output identification information for all the characters the set contained. The reasons for implementing the neural net OCR as three programs were all practical. By keeping the first step separate, the preprocessing code from the feature extraction OCR program could be used, eliminating this one area of difference between the two algorithms. The second step was separated just because

learning was such a slow process. Several machines could thus be dedicated to nothing but learning while a different machine was used to analyze the results.

## SVM

Support Vector Machines (SVMs) are a set of related supervised learning methods which can be used for both classification and regression. In simple words, given a set of training examples, each marked as belonging to one of two categories, an SVM classification training algorithm tries to build a decision model capable of predicting whether a new example falls into one category or the other. If the examples are represented as points in space, a linear SVM model can be interpreted as a division of this space so that the examples belonging to separate categories are divided by a clear gap that is as wide as possible. New examples are then predicted to belong to a category based on which side of the gap they fall on.

A linear support vector machine is composed of a set of given support vectors  $\mathbf{z}$  and a set of weights  $\mathbf{w}$ . The computation for the output of a given SVM with  $N$  support vectors  $z_1, z_2, \dots, z_N$  and weights  $w_1, w_2, \dots, w_N$  is then given by:

$$F(x) = \sum_{i=1}^N w_i \langle z_i, x \rangle + b$$

A decision function is then applied to transform this output in a binary decision. Usually,  $\text{sign}(\cdot)$  is used, so that outputs greater than zero are taken as a class and outputs lesser than zero are taken as the other.

## Chapter-IV

### Proposed Solution

CAPTCHAs can be divided into four categories namely

1. Simple (No mesh) Background,
2. Black Mesh Background,
3. White Mesh Background
4. Loosely Connected Characters.

In Simple (No Mesh) , the characters are written on a simple background whereas in Black and White Mesh ,the characters are written on black and white mesh respectively. In Loosely Connected Characters type, the pixels of the characters are loosely connected. Algorithm developed uses feature extraction technique to recognize the characters and thus analyses security of all types of CAPTCHAs mentioned here.

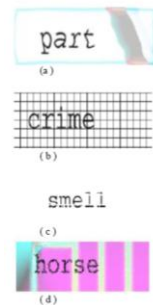


Figure 16. Types of CAPTCHAs

As mentioned earlier an algorithm has different phases such as preprocessing, segmentation, feature extraction and character recognition which are explained in detail as mentioned below:

## SYSTEM ARCHITECTURE

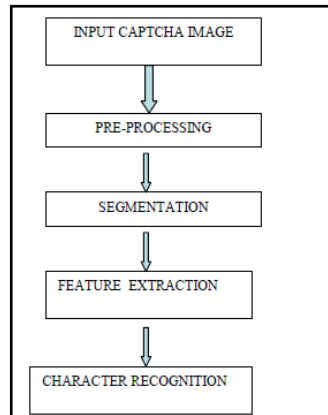


Figure 17. System Architecture

### ***1. Preprocessing***

Preprocessing will convert input CAPTCHA image into cleared image by first converting into gray scale, then carry out binarization , then removes line, & dots (if any present on image ) Since there are four types of CAPTCHA, each one has different preprocessing operations which are as follows:

#### ***1.1. Simple (No Mesh) Background***

- (a) The given CAPTCHA is first converted to gray scale.
- (b) Binarization of an image can lead to an image whose pixels have only two possible intensity values

### ***1.1.1 Color To Gray***

The CAPTCHA image is given as an input to the program. The CAPTCHA image is then converted into gray scale image. It is converted into gray scale because CAPTCHA image contains many colors and to work on each of them is very difficult so converting it into gray scale helps only to work on 256 intensity values.

#### **Algorithm to Convert 24 bit to 8 bit Gray Scale Image**

- Get Image File As Input File
- Extract Header Information
- If input is 24 bit , Create Output Header & Palette.
- Read three pixels from input file & calculate average.
- Write Average in Output File.
- Repeat steps 4 & 5 till end of file.

#### ***1.1.1. Binarization***

Binary images are images whose pixels have only two possible intensity values. They are normally displayed as black and white. Numerically, the two values are often 0 for black, and either 1 or 255 for white. Binary images are often produced by thresholding a grayscale or color image, in order to separate an object in the image from the background. The color of the object (usually white) is referred to as the foreground color. The rest (usually black) is referred to as the background color.

### ***3.1.2. Black Mesh Background***

- (a) The given CAPTCHA is first converted to gray scale and binarized as mentioned above.
- (b) Binarization of an image can lead to an image whose pixels have only two possible intensity values.
- (c) CAPTCHA image has noise such as horizontal lines, vertical lines and dots that needs to remove so as to get clear image.

#### ***3.1.2.1 Line Removal***

The CAPTCHA images sometimes contain horizontal lines and vertical lines. To remove lines the number of continuous black pixels in row or columns is counted. If the count is more than 80% of total width or height of the image, then detected as a line and thus removed it by making it white.

#### ***3.1.2.2 Discontinuity Removal***

After the lines are removed, characters become discontinuous. To remove this discontinuity, the original image (image before line removal) is compared with the newer one and fills the gaps in between so that characters remain continuous. This will help in recognizing the characters.

#### ***1.2.3 Dot Removal***

After Binarization sometimes CAPTCHA images may contain unnecessary set of black pixels i.e. dot. To remove them, the image is scanned. Then after getting first black pixel check its neighboring 8 pixels. If all of them are white, then make the black pixel white. If



these dots are bigger (greater than one pixel) then count the pixels in it, if the count is less than 40 then make it white. This is because, 40 pixels can't make a character, so it is an unnecessary dot.

### ***1.3. White Mesh Background***

- (a) The given CAPTCHA is first converted to gray scale and binarized.
- (b) The image is now inverted. Now the same black mesh background as stated above is applied
- (c) The image is now inverted to get back the original image.

### ***1.4. Loosely Connected Characters***

- (a) The given CAPTCHA is first converted to gray scale and binarized.
- (b) The noise is removed and get the clear image as stated above.

## ***2. Segmentation***

After the image passes the preprocessing stage, characters need to be segmented because if characters are joined then it is very difficult to recognize them. In CAPTCHA all the characters are distant so it is not very difficult to segment them. Checking continuous black pixels separates characters. Once the program checks black pixels it is made red so that program can understand that the specific character is already separated.

### ***3. Feature Extraction and Character Recognition***

Each character has some unique set of features. The features, which were used, are as follows:

(i) Number of Holes - Each character is checked whether it has a hole or not. Characters a, b, d, e, g, o, p, q each have 1 hole and rest of the characters do not have a hole.

(ii) Height of Character - Each character is categorized into small or big on the basis of its height. A threshold is taken which categorizes the given character.

(iii) Maximum Number of White-Black Transitions (Vertical intersection) - A line cutting the character is drawn and the maximum numbers of white-black transitions that are possible are noted. Example: - When a line is drawn through character 'a', maximum 3 transitions are possible. Similarly transitions for other characters were also noted.

(iv) Nature of Vertical Stroke: - A vertical stroke (Blue in color) is drawn for character with 'big' height along the vertical stroke of the character. If the character is of 'small' height '\*' is noted. If there is no hole in character '1' is noted. And if there is a hole in character then the position of vertical stroke relative to the hole i.e. either left or right is noted. According to the features of the characters, characters are recognized. This module can also be useful in recognizing handwritten characters.

#### **Work Done**

**Technology used**: ASP.NET and C# as the programming languages.

The Integrated Development Platform used is **Visual Studio 2010**.

We have applied the above algorithm for Image with Simple Background. The algorithm removes the noise from the image. After this , it performs segmentation and does character recognition. It is working for all the standard fonts defined by Microsoft.

**Input:** CAPTCHA in .png,. jpeg, tiff or bmp format.

**Output:** A text file with the characters in the CAPTCHA

### **CODE :-**

#### **Start.cs**

### **EXPLANATION :-**

Start.cs gives us the initial display i.e. the start button which begins the output of the program. In the given code we have used the using directive. Using directives helps us to access namespaces that the application will be using frequently and save the programmer from specifying a fully qualified name every time that a method that is contained within is used .Namespace keyword is used which is used to declare a scope. Here ImageTextReader is a fully qualified namespace. Partial keyword is used to split the class definition over separate files. The start button has been assigned as the name button 1. There is a function named button1\_click which creates an object f1 of class form1 and calls the function show and hide.

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ImageTextReader
{
    public partial class Start : Form
    {
        public Start()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Form1 f1 = new Form1();
            f1.Show();
            this.Hide();
        }
    }
}
```

## Layout.cs

### EXPLANATION :-

Using directive is used to access various namespaces. The System namespace contains fundamental classes and base classes that define commonly used values and reference data types ,events and event handlers, interfaces , attributes and processing exceptions. This code describes the layout of the entire application..There is a class layout which contains all the functions describing the dimensions and colour of the various rectangular boxes ,dialog boxes and drawing objects used in the application.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace ImageTextReader
{
    class Layout
    {
        #region Class Local Variables
```

```
private Bitmap WholeBMP;
private Bitmap SaveBMP;
private Bitmap TLBMP;
private Bitmap TRBMP;
private Bitmap BLBMP;
private Bitmap BRBMP;

private Rectangle DrawRect;
private Point TLpt;
private Point TRpt;
private Point BLpt;
private Point BRpt;
private int Counter = 0;

private ImageAttributes Ia;

#endregion
private System.ComponentModel.IContainer components;
private System.Windows.Forms.Timer T1;

private System.Windows.Forms.Button cmdGo;

protected void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
}
```

```

    }
    if (WholeBMP != null)
        WholeBMP.Dispose();

    if (SaveBMP != null)
        SaveBMP.Dispose();
    if (TLBMP != null)
        TLBMP.Dispose();
    if (TRBMP != null)
        TRBMP.Dispose();
    if (BLBMP != null)
        BLBMP.Dispose();
    if (BRBMP != null)
        BRBMP.Dispose();
    if (Ia != null)
        Ia.Dispose();

    }

    #region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.components = new System.ComponentModel.Container();
        this.cmdGo = new System.Windows.Forms.Button();
        this.T1 = new System.Windows.Forms.Timer(this.components);

        //

```

```

// cmdGo
//
this.cmdGo.Location = new System.Drawing.Point(328, 336);
this.cmdGo.Name = "cmdGo";
this.cmdGo.Size = new System.Drawing.Size(48, 24);
this.cmdGo.TabIndex = 0;
this.cmdGo.Text = "GO";
this.cmdGo.Click += new System.EventHandler(this.Explode);
//
// T1
//
this.T1.Tick += new System.EventHandler(this.T1_Tick);
//
// Form1
//

}
#endregion

private void Form1_Load(object sender, System.EventArgs e)
{
}

protected void OnPaint(PaintEventArgs e)
{
    Graphics G = e.Graphics;

    if ( WholeBMP != null )
    {
        G.DrawImage(WholeBMP, DrawRect);
    }
}

```



```
    return;
}

if ( TLBMP != null )
    G.DrawImage( TLBMP, new Rectangle(TLpt, TLBMP.Size),
        0, 0,
        TLBMP.Width, TLBMP.Height,
        GraphicsUnit.Pixel,
        Ia );

if ( TRBMP != null )
    G.DrawImage( TRBMP, new Rectangle(TRpt, TRBMP.Size),
        0, 0,
        TRBMP.Width, TRBMP.Height,
        GraphicsUnit.Pixel,
        Ia );

if ( BLBMP != null )
    G.DrawImage( BLBMP, new Rectangle(BLpt, BLBMP.Size),
        0, 0,
        BLBMP.Width, BLBMP.Height,
        GraphicsUnit.Pixel,
        Ia );

if ( BRBMP != null )
    G.DrawImage( BRBMP, new Rectangle(BRpt, BRBMP.Size),
        0, 0,
        BRBMP.Width, BRBMP.Height,
        GraphicsUnit.Pixel,
        Ia );
}
```

```

private void Explode(object sender, System.EventArgs e)
{
    if ( WholeBMP != null )
    {
        cmdGo.Enabled = false;
        int L = 0;
        int T = 0;
        int Cx = (int)(WholeBMP.Width/2);
        int Cy = (int)(WholeBMP.Height/2);

        Rectangle R1 = new Rectangle( L, T, Cx, Cy );
        Rectangle R2 = new Rectangle( Cx, T, Cx, Cy );
        Rectangle R3 = new Rectangle( L, Cy, Cx, Cy );
        Rectangle R4 = new Rectangle( Cx, Cy, Cx, Cy );

        SaveBMP = WholeBMP;
        TLBMP = WholeBMP.Clone(new Rectangle( L, T, Cx, Cy ),
                               WholeBMP.PixelFormat);
        TRBMP = WholeBMP.Clone(new Rectangle( Cx, T, Cx, Cy ),
                               WholeBMP.PixelFormat);
        BLBMP = WholeBMP.Clone(new Rectangle( L, Cy, Cx, Cy ),
                               WholeBMP.PixelFormat);
        BRBMP = WholeBMP.Clone(new Rectangle( Cx, Cy, Cx, Cy ),
                               WholeBMP.PixelFormat);
        WholeBMP = null;

        int Gap = 10;
        TLpt = new Point( DrawRect.Left-Gap, DrawRect.Top-Gap );
        TRpt = new Point( DrawRect.Left+Cx+Gap, DrawRect.Top-Gap );
        BLpt = new Point( DrawRect.Left-Gap, DrawRect.Top+Cy+Gap );
    }
}

```

```

BRpt = new Point( DrawRect.Left+Cx+Gap, DrawRect.Top+Cy+Gap );

T1.Enabled = true;

}
}

private void T1_Tick(object sender, System.EventArgs e)
{
Counter += 1;
if ( Counter == 62 )
{
Counter = 0;
cmdGo.Enabled = true;
T1.Enabled = false;
WholeBMP = SaveBMP;
}

TLpt.X-=1;
TLpt.Y-=1;

TRpt.X+=1;
TRpt.Y-=1;

BLpt.X-=1;
BLpt.Y+=1;

BRpt.X+=1;
BRpt.Y+=1;

```

```
float[][] m = {new float[] {1, 0, 0, 0, 0},
               new float[] {0, 1, 0, 0, 0},
               new float[] {0, 0, 1, 0, 0},
               new float[] {0, 0, 0, (1-(float)Counter/62), 0},
               new float[] {0, 0, 0, 0, 1}};
```

```
ColorMatrix cm = new ColorMatrix(m);
```

```
Ia = new ImageAttributes();
```

```
Ia.SetColorMatrix( cm, ColorMatrixFlag.Default,
                  ColorAdjustType.Bitmap);
```

```
TLBMP.RotateFlip(RotateFlipType.Rotate90FlipNone);
```

```
TRBMP.RotateFlip(RotateFlipType.Rotate90FlipNone);
```

```
BLBMP.RotateFlip(RotateFlipType.Rotate90FlipNone);
```

```
BRBMP.RotateFlip(RotateFlipType.Rotate90FlipNone);
```

```
}
```

```
}
```

```
}
```

## Form1.cs

### Explanation :-

Using directive is used with various namespaces. Different functions are declared like button1\_click, button2\_click, button3\_click and button4\_click. This code describes the procedure for loading the captcha image, checking for various file formats and performing OCR operations.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Threading;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace ImageTextReader
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();
        }
        string choosefile = "";
        string fileName = "";
```

```

string text = "";
private void button1_Click(object sender, EventArgs e)
{

    if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
    {
        MessageBox.Show("Operation cancelled");
    }
    else
    {
        choosefile = "";
        choosefile = openFileDialog1.FileName;
        pictureBox1.ImageLocation = openFileDialog1.FileName;

    }
}

private void button2_Click(object sender, EventArgs e)
{
    try
    {

        string fileExtension = Path.GetExtension(Convert.ToString(choosefile));

        //get file name without extension
        fileName = Convert.ToString(choosefile).Replace(fileExtension, string.Empty);

        //Check for JPG File Format
        if (fileExtension == ".jpg" || fileExtension == ".JPG" || fileExtension.ToLower()
== ".png" || fileExtension.ToLower() == ".gif" || fileExtension.ToLower() == ".tif") // or //
ImageFormat.Jpeg.ToString()

```

```

    {
        try
        {
            //OCR Operations ...
            MODI.Document md = new MODI.Document();
            md.Create(Convert.ToString(choosefile));
            md.OCR(MODI.MiLANGUAGES.miLANG_ENGLISH, true, true);
            MODI.Image image = (MODI.Image)md.Images[0];
            text = image.Layout.Text;
            textBox2.Text = image.Layout.Text;
            lip();

        }
        catch (Exception)
        {
            MessageBox.Show("This Image hasn't a text or has a problem",
                "OCR Notifications",
                MessageBoxButtons.OK, MessageBoxIcon.Information);

        }
    }
}
catch(Exception ex)
{

    MessageBox.Show("Please Select Image. Error Message : " +
ex.Message.ToString());

}
}

private void button3_Click(object sender, EventArgs e)

```

```

    {
        try
        {
            //create text file with the same Image file name
            FileStream createFile = new FileStream(fileName + "." + comboBox1.Text,
FileMode.OpenOrCreate);

            //save the image text in the text file
            StreamWriter writeFile = new StreamWriter(createFile);
            writeFile.Write(text);
            // Console.WriteLine(image.Layout.Text);

            writeFile.Close();
            MessageBox.Show("Your output is saved in the same folder where the image
exist.");
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error while saving the file. May your system have a
virus.");
        }
    }

private void button4_Click(object sender, EventArgs e)
{
    Application.Exit();
}

private void lip()
{
    try

```



```
{
    FileStream s = new FileStream("t.txt", FileMode.Open);
    //StreamWriter w = new StreamWriter(s);
    //w.Write("Hello World");
    //w.Close();

    //s = new FileStream("Bar.txt", FileMode.Open);
    StreamReader r = new StreamReader(s);
    string t;
    while ((t = r.ReadLine()) != null)
    {
        // Console.WriteLine(t);
        textBox3.Text = textBox3.Text + t+"\n";
    }
    // w.Close();
}
catch
{
}
}
}
```

## OUTPUT

### **Snapshot 1. Starting application**



Figure 18. Starting the application

### **Snapshot 2. Loading Image**

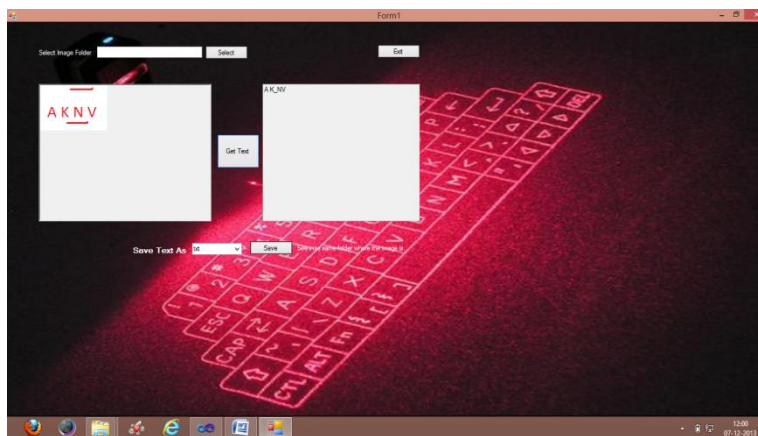


Figure 19. Loading and saving the output

### Snapshot 3. Saved in a text file

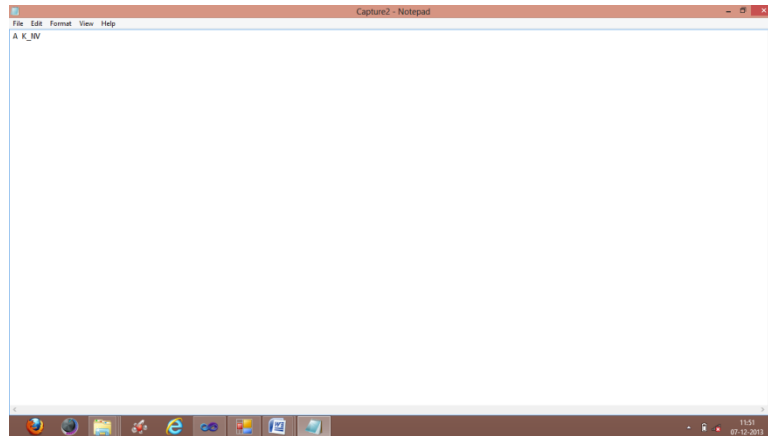


Figure 20. Text

file saved

### Results after testing

We applied our algorithm to about 60 CAPTCHAs and the program passed 39 of them.

Some of the sample captchas used are below:

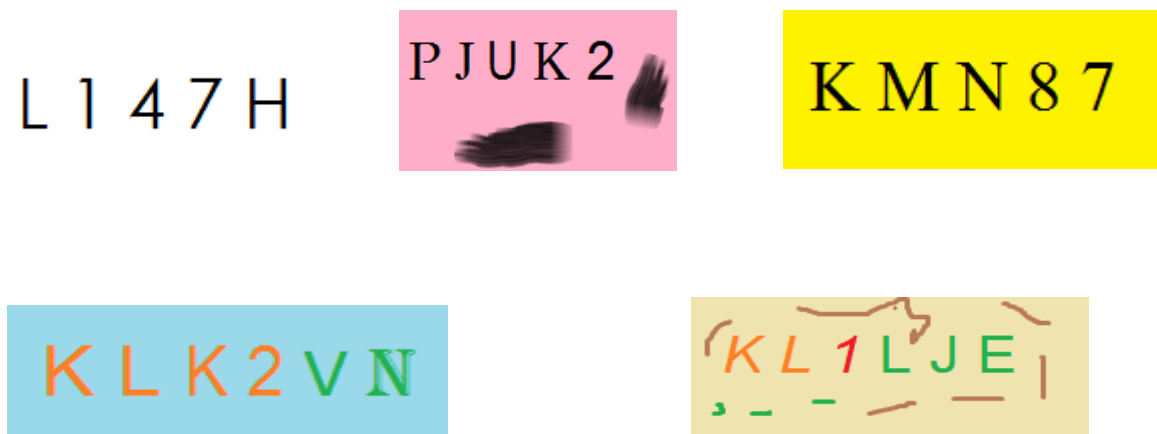


Figure 21. Some of the sample CAPTCHAs

$$\text{Accuracy rate} = \frac{\text{No. of captchas cracked}}{\text{Total captchas}} \times 100 = (39/60) \times 100 = 65\%$$

$$\text{Error rate} = 100 - \text{accuracy rate} = 35\%$$

### **Analysis of the Result**

- It supported almost all the fonts like Algerian, Arial, Verdana etc.
- The noise in the background was removed efficiently 80% of the times.
- It worked for any length of CAPTCHAs.
- If the arcs in the background intersect the character, then it fails sometimes.
- Most of the CAPTCHAs failed when there is overlapping between characters.

Hence, when the CAPTCHAs are designed following points should be taken in consideration:

- Letting characters touch or overlap with each other can provide extra segmentation resistance.

- Making it harder to tell characters and arcs apart (e.g. by juxtaposing characters in any direction).
- By removing some of the pixels in the characters may result in failure of attacks.

## References

- S. Huang,, Y. Lee, G. Bell, Z. Ou. A projection-based Segmentation Algorithm for Breaking MSN and Yahoo captchas.
- J. Yan, A.S. El Ahmad. A low cost attack on a Microsoft Catpcha.
- Mike O'Neill. Neural network for Recognition of Handwritten Digits.
- D. You, G. Kim. An approach for locating segmentation points of handwritten digit strings using a neural network.
- K Chellapilla, K Larson, P Simard and M Czerwinski, "Designing human friendly human interaction proofs", ACM CHI'05, 2005.
- Microsoft Corporation. "Human Interaction Proof (HIP) -- Technical and Market view" 2006.
- G Mori and J Malik. "Recognising objects in adversarial clutter: breaking a visual "captcha" IEEE Conference on Computer Vision & Pattern Recognition (CVPR), 2003.
- G Moy, N Jones, C Harkless and R Potter. "Distortion estimation techniques in solving CAPTCHAS" 2004.
- P Simard, R Szeliski, J Benaloh, J Couvreur and I Calinov, "Using character recognition segmentation to tell computers from humans", International Conference on Document Analysis and Recognition (ICDAR), 2003.

- P Simard, D Steinkraus, J Platt. “Best Practice for Convolutional Neural Networks Visual Document Analysis”, International Conference on the Document Analysis and Recognition (ICDAR), IEEE Computer Society, Los Alamitos, 2007.
- C Pope and K Kaur. “Is It Human or Computer? Defending E-Commerce with CAPTCHA” IT Professional, March 2005.
- J Yan and A S El Ahmad. “Breaking Visual CAPTCHAs with Naïve Pattern Recognition Algorithms”, in *Proc. of the 23rd Annual Computer Security Applications Conference* FL, USA, Dec 2007. IEEE computer society.

