Assessing Threat Posed to Video CAPTCHA by OCR-Based Attacks

by

Alex Canter

A Project Report Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in
Computer Science

Supervised by

Dr. Richard Zanibbi

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

June 2013

The project “Assessing Threat Posed to Video CAPTCHA by OCR-Based Attacks” by Alex Canter has been examined and approved by the following Examination Committee:

Dr. Richard Zanibbi
Associate Professor
Project Committee Chair

Dr. Zack Butler
Associate Professor

Dr. Stephanie Ludi
Associate Professor

Acknowledgments

I would like to thank and acknowledge the members of the DPRL at R.I.T. for their continued support and friendship throughout my work. In particular, I am grateful to Francisco Alvaro, Lei Hu, David Stalnaker, and Siyu Zhu for taking time out of their own work to aid my own.

I would like to thank my committee members, Dr. Richard Zanibbi, Dr. Zack Butler and Dr. Stephanie Ludi, for their time and patience with regards to my work.

I extend my sincere gratitude and appreciation to my project advisor, Dr. Richard Zanibbi, for his invaluable guidance, support, time, and understanding (and tolerance) this past year.

Lastly, I thank my parents for their perseverance in dealing with me for the last 24 years. It really is quite a feat.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part (*CAPTCHA*) is a common and critical component in web security. Motivation for CAPTCHAs is to stop the automated abuse of web services [9]. A CAPTCHA uses a challenge task to differentiate humans from computers. A usability vs. security trade-off is present in current CAPTCHA implementations. Video CAPTCHA is a more recent implementation, aiming to reduce this trade-off by providing increased usability. Compared to traditional text and audio CAPTCHAs, little research in video CAPTCHAs exists.

A video CAPTCHA [8] proposed by *Kluever & Zanibbi* has shown promise in usability studies [9]. This CAPTCHA requires a user to naturally describe a presented video, using three *tags*. Further research is needed in the CAPTCHA's security against OCR-based attacks on text contained within the videos it uses. If such text is extracted and entered as tags, the CAPTCHA challenge may be passed. We describe an evaluation of the CAPTCHA's security against this approach. Attacks are made against the CAPTCHA, aiming to extract and OCR text within video key-frames.

A natural text segmenter is developed, using a bagged classifier of C4.5 decision trees over text-location masks[1]. We find that natural text can be segmented with an average f-measure of 76%, through the use of only geometric features. Tesseract-OCR is used to classify text within segments, but is shown to be unreliable when classifying natural text. Video CAPTCHA is found to be vulnerable to OCR-based attacks on its key frames. Text

[1] Text masks are defined manually, as text localization is outside the scope of this work.

in challenge videos is found to be highly indicative of text that will pass a challenge. In a series of 13 manual attacks on a randomly selected video sample, 9 (69.2%) succeed in passing the challenge. Automated attacks are made against the same 13 videos, resulting in 2 (15.4%) of 13 attacks succeeding.

Despite the low break rate, we find that automated attacks are able to extract groundtruth tags in 9 (69.2%) of the 13 challenge videos. It is concluded that a more accurate way of choosing a subset of extracted text to use would boost the automated break rate significantly. We conclude that video CAPTCHA is vulnerable to OCR-based attacks that employ a more sophisticated tag selection method.

This report details both the target video CAPTCHA system and Tesseract. Methodology for attacks and natural-text segmentation are then described. An evaluation of the experiments conducted is then presented, detailing the performance of both the segmenter and attacks. Evaluation results are discussed, followed by suggestions for future work.

Chapter 2

Background

An introduction to the key components and concepts that make up our main problem is presented here. Topics covered are: *(a)* CAPTCHA and its common implementations, including video CAPTCHA; *(b)* prior work in developing and breaking video CAPTCHA; *(c)* the video CAPTCHA system that this project aims to break; *(d)* optical character recognition (OCR) and the Tesseract OCR engine.

2.1 CAPTCHA

The **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part (*CAPTCHA*, as coined by M. Blum et al. in [1]) is a critical component in how we interact with the web. The ultimate goal of CAPTCHAs is to automatically differentiate a human from a computer. This differentiation is typically accomplished through a simple *challenge* task; passing a challenge should ideally be simple for humans and difficult for computers. The motivation for CAPTCHAs is to stop the automated abuse of online services [9]. Example use cases include adding a layer of security to account registration and email forms. An example of a text-based CAPTCHA used in account registration is shown in Figure 2.1. The motivation for this project is to evaluate the security of a *video understanding* based CAPTCHA system [8], which is a highly usable implementation of a *Video CAPTCHA* (discussed later).

CAPTCHAs have been in use since 2000, with researchers at Carnegie Mellon University spearheading development. AltaVista introduced the first public use of a CAPTCHA,

and patented its own version in 2001 [17]. Since then, there has been a multitude of different approaches for distinguishing humans from computers.

Research in CAPTCHAs involves the fields of computer vision, machine learning, pattern recognition, and human-computer interaction. The main problem that improving CAPTCHAs present to researchers and developers is finding a way to best minimize the usability vs. security trade-off. An ideal CAPTCHA would always pass a human (perfect usability), while always failing a computer (perfect security). The trend however, has been a somewhat inverse correlation between usability and security. Researchers continue to aim for improvements in the usability, accessibility, and security of CAPTCHA systems.

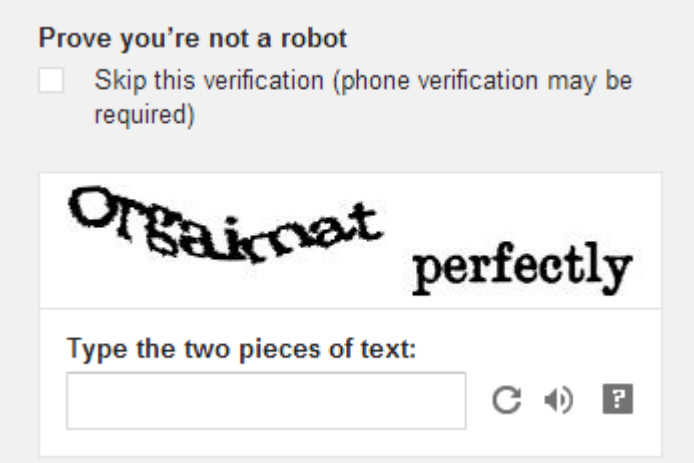


Figure 2.1: A text-based CAPTCHA presented by Google during their account creation process. A user must enter each displayed character correctly in-order to proceed. The volume icon indicates that a user may elect for an audio-based CAPTCHA test instead.

Many different types of CAPTCHA systems have been developed with the goal of minimizing the usability vs. security trade-off. CAPTCHAs generally involve presenting a user with some form of information, and requiring the user to act based on their understanding or observation of that information. CAPTCHA implementations differ in the type of information they present, as well as in how that information is presented. Common types of information presented to users are: [22]:

Text The 'classic' CAPTCHA that we are accustomed to seeing. This approach involves the server-side rendering of distorted text characters, and their presentation to the user.

Images Example: selecting all similar images in a set.

Audio Example: Entering alphanumeric characters spoken by a distorted voice, in the presence of background noise.

2.2 Video CAPTCHA

Video CAPTCHA is a newer and less commonly seen CAPTCHA system. The term "video CAPTCHA" is used here to refer to any CAPTCHA that uses a video as its means to present information to a user. Although prior work in video CAPTCHA is limited, both commercial and academic applications do exist.

One commercial system is *NuCaptcha* [6], which is displayed in Figure 2.2. NuCaptcha encapsulates a text-based CAPTCHA inside of a video; the video adds animations to the text, ideally making it more difficult to crack than a text-based CAPTCHA alone. NuCaptcha was cracked by Stanford researchers working alongside the NuCaptcha team in 2012 [2]. By performing standard OCR on snapshots of the video's moving text, the researchers obtained a crack rate of more than 90% across NuCaptcha's challenges.

Figure 2.2: An online demonstration of the *NuCaptcha* video CAPTCHA service. The text shown in red is animated, and must be entered correctly for the test to be passed.

Researchers at Wuban University describe a CAPTCHA based on moving, 3D object identification [3]. It is described by the authors as resistant to common, static OCR attacks.

2.3 Video Understanding Approach

The motivation for this project is the evaluation of a *video understanding* based CAPTCHA system's vulnerability to in-scene, text based attacks. This CAPTCHA, seen in Figure 2.3, is proposed in [8, 9] by *Kluever & Zanibbi*. To pass a challenge, a user is asked to naturally describe a short video by entering three descriptive tags for it. The challenge is passed if one or more of the user's answer tags matches any of the video's automatically generated, groundtruth tags. No similar video CAPTCHA currently (at least publicly) exists, although similar methods have been tried on still images. An in-scene text attack on this system aims to find answer tags by performing Optical Character Recognition (OCR) on words visible within the challenge video's key frames.

Usability and security for the video CAPTCHA are evaluated in [9]. A promising 60% of 184 usability study participants found video CAPTCHAs to be "more enjoyable" than traditional text CAPTCHAs. Human pass rates in the studies are referred to as comparable to then-existing CAPTCHA systems. However, a usability vs. security trade-off is visible in the results. Security is measured by the pass rate of tag-frequency based attacks; these attacks involve submitting the three tags that describe the largest portion of videos in the dataset. Usability is measured by the human pass rate of the user study participants. Results in [9] suggest that different configurations of the system can result in significant security vs. usability changes. A configuration for the system $\tau = (n, t, s, l)$ defines four different parameters: n = # of related tags in the groundtruth; t = threshold for frequency pruning; s = whether or not word stemming is used; l = whether or not inexact matching is used. Example configurations and their resulting pass rates for both humans and attacks are shown in Table 2.1

Table 2.1: Different parameters in video-understanding based CAPTCHA [9] affecting the pass rates of humans and the break rates frequency-based attacks. Configurations define the parameters: n = # of related tags in the groundtruth; t = threshold for frequency pruning; s = whether or not word stemming is used; l = whether or not inexact matching is used.

n	*t*	*s*	*l*	*Human Pass Rate (%)*	*Attack Break Rate (%)*
0	1.0	FALSE	FALSE	69.73	12.86
25	0.006	TRUE	TRUE	86.49	5.26
90	0.006	TRUE	TRUE	90.19	12.63

Figure 2.3: The video-understanding based CAPTCHA system used for this project. [8]

Challenge generation and grading are both automated, and make use of videos and tags submitted to the video sharing website YouTube.com. A *tag* is a descriptive word that a video's uploader assigns to that video. Details of these procedures can be found in [8] and [9]; what follows is a high-level summary of the information present in these sources.

To select a challenge video, a random walk over the dataset containing all YouTube videos is performed. The dataset is treated as an undirected, bipartite graph representing the many-to-many relationship between videos and tags. The algorithm to randomly walk this graph is shown in Algorithm 2.1, as it is described in [9]. Given a starting tag t, a current video v is selected such that v is tagged with t. A new video v_{next} is chosen from the set of videos related to v such that some tag w is associated with both v and v_{next}. w now becomes t, and the process repeats. This process is repeated for a random number of iterations between 1 and a specified max-depth.

Generation of groundtruth tags for a challenge video adds both the set of tags directly assigned to that video, and a fixed number of tags pulled from videos related to it. Related videos are obtained by querying the YouTube API, and then sorted by cosine similarity with the challenge video. Cosine similarity is often used in information retrieval to evaluate the relevance of retrieved text documents. Cosine similarity of two vectors u and v is defned as:

$$CosSim(u, v) = \frac{u \cdot v}{\|u\|\|v\|} \tag{2.1}$$

Videos are represented by occurrence bit-vectors of their tags. A bit representing a tag t within a video's occurrence vector is 1 if t is present in that video's author tags, otherwise it is 0. Occurrence vectors of related videos are sorted in decreasing order of cosine similarity to the challenge video's occurrence vector. A fixed, maximum number of related tags are then added to the challenge video's groundtruth.

Any predefined *stop words* are then removed from the groundtruth. Stop words are frequently occurring tags that provide little descriptive value, such as "if", "then", and "he". Different inflections and derivations of each groundtruth tag are then added through *word stemming* [18]. Word stemming reduces tags to their root forms, lowering the chance for false negatives caused by non-descriptive differences from the groundtruth. For example: if the groundtruth contains *dogs*, then the root *dog* is also added. Frequency-based pruning of the groundtruth is performed as a final step. A tag-frequency distribution is generated by recording the number of times different tags are seen during a series of random walks. If any tag has an estimated frequency $\geq$ a pruning threshold t, that tag is pruned from the groundtruth.

Grading involves the *inexact* matching of each answer tag to any groundtruth tag. If at least one match exists for any answer tag, then the challenge is passed. Inexact matching looks at the normalized Levenshtein distance (also known as string edit distance) between two tags being compared. The Levenshtein distance between two strings s and t is the minimum number of deletions, insertions, and/or substitutions needed to transform s to t or t to s; the distance is then normalized by string length. If the normalized distance between

two tags is ≥ 0.8, then they are considered to be a match (a distance of 1.0 signifies a perfect match). This makes grading more forgiving of spelling errors and typos, and was found to have little impact on security against frequency-based attacks.

Algorithm 2.1: Procedure to randomly walk the graph of all YouTube videos, as defined in [9].

input : $MaxDepth$: Maximum depth of the random walk

1 Randomly select a depth d for the walk, such that $1 \leq d \leq MaxDepth$.
2 Randomly select a word t from a predefined dictionary.
3 Query the YouTube API to locate the tag vertex u corresponding to tag t.
4 **while** $i < d$ **do**
5 Select a random edge (u, v) where v is a video vertex.
6 Select a random edge (v, w) where w is a tag associated with video v.
7 Assign $u \longleftarrow w$.
8 Increment i.

2.4 Tesseract OCR Engine

Tesseract is an open-source OCR engine, that is widely regarded as being among the most accurate of such (non commercial) tools. It began as a project at HP Labs in 1985, where it was actively developed for ten years. Development ceased after this time period, until Google Inc. picked up and open-sourced the project in 2006. It is now freely available on Google Code[1], under the Apache 2.0 license. It is available for Windows, Mac OSX, and Linux.

Tesseract can currently detect and output text in over 60 languages. It can be used either via the command line, or accessed through its C++ API. Third party add-ons exist that offer a GUI. Among other key features, Tesseract supports: *(a)* confidence outputs for individual words, *(b)* automatic adjustment for crooked text-lines, and *(c)* performing OCR on image

[1] https://code.google.com/p/tesseract-ocr

sub-regions. Recommended features of images on which Tesseract is to perform OCR are:

Text Uniformity Text is in a uniform, non-handwritten font. Spacing between lines and letters is consistent.

Proper Text Size All text has a minimal font-size of 10pt, and a minimal X-height of 20 pixels. X-height refers to the height in pixels of a lowercase 'x' is a given font.

High Resolution The image has a resolution of between 300dpi (dots per inch) and 500dpi.

Little Noise Text has been de-noised as much as possible.

Binary Colors The image has only the colors black and white.

A detailed overview of Tesseract's components can be found in [20]. Tesseract uses both an *adaptive classier* and a *static character classifier*. The classifiers differ in their *(a)* given training data, and *(b)* method of character normalization. The static character classifier is given raw input, while the adaptive classifier is given the output from the static character classifier. The adaptive classifier provides better discrimination in documents with multiple fonts, by being more sensitive to font features and less sensitive to noise.

Tesseract's features, which are a key strength of Tesseract [20], are different for the training and recognition phases. Different features are needed to recognize damaged text, which may only be present during testing. Training features are *prototypes* (similar to a character's contours), which are composed of the segments of a polygonal approximation of the character's shape. Segment's contain the features: *x position*, *y position*, *angle*, and *length*. Recognition features are small line segments extracted from the character's outline; segments are of a fixed, normalized length. Segments are matched many-to-one to potential prototypes during classification, and the prototype most closely matching the segment features is chosen as the final character. Features contained within segments are the same as a prototype's segment, minus the *angle*.

Although Tesseract is widely considered to be the most accurate non-commercial OCR system, it has limitations that impact this project. The first issue is that Tesseract strongly prefers its input to be binary images [20], and performs with less accuracy and predictability on images with color noise [16]. Given that the vast majority of videos used in video-understanding systems (YouTube videos, in the case of [9]) are color, a good thresholding procedure is needed. It is found in tests described in [16] that properly grayscaling color images has a significantly positive impact on Tesseract's OCR results. Text resolution and size must be considered as well. Tesseract's documentation states that the accuracy of results drop drastically when the source image has either a resolution less than 300 dots-per-inch (*DPI*), or text smaller than 10 points. Many videos on YouTube are not of good quality, even when the DPI is specified as high. The text size in the videos is also unpredictable.

Although not officially stated, it can be inferred that Tesseract expects clean, book-page like text. Tools for training Tesseract all generate images containing such text, and generally involve choosing a font style and size. Documentation also recommends a minimum *X-height*, which is the height of a lower case 'x' in a specific font. This information points towards optimal inputs being uniform text with little noise, such as scanned documents.

Tesseract has been successful as a viable tool for natural-text OCR. A group of graduate students[2] successfully used Tesseract as the OCR component of a system that extracts and classifies text within natural scenes. This system focuses on heavy pre-processing, using the *Stroke-Width Transform* (*SWT)* described in [4]. SWT produces an array containing the estimated stroke width (*SW*) of the strokes pertaining to each pixel of an image. The SW of Text in natural scenes tends to be nearly constant across all of its connected components, particularly when compared to other elements of the scene [4]. Leveraging this, text is extracted by performing a SWT on a Canny edge-detection representation of the image. Connected components (CCs) of text can then be found by comparing the SWs of

[2]This is a student project for a robotics course offered at the University of Pennsylvania. https://alliance.seas.upenn.edu/ meam620/wiki/index.php?n=MenglongZhu2011.Final

each component, and taking only those that do not have highly varying SWs. Connected components are then grouped into words by comparing various, basic geometric properties relative to the other components. These groups are used to generate bounding box coordinates, which are fed to Tesseract for OCR. Much of the work in this system is in *text localization*, the process of finding image sub-regions containing text. After localization and pre-processing, the segmentation step seems comparatively trivial due to the cleanliness of the post-processed image. This system demonstrates that if the location of text is known in a natural scene, then Tesseract may be used to successfully classify that text in many instances.

Section 3.3 proposes a similar but different strategy. Text is first located and labeled manually, rather than the use of text localization techniques. Pre-processing of text regions focuses more on removing noise caused by the manual text labeling, but uses similar principles of smoothing and removing small components described above. CCs belonging text-line are grouped via separate, minimum spanning trees. Features used to classify CCs are similar, but are plugged into a bagged decision tree classifier instead of used as-is. The principle that basic, geometric properties can accurately group CCs into words is still present.

2.5 Problem

The goal of this project is to measure the magnitude of the problem that OCR attacks pose to video-understanding based CAPTCHA. The data flow for an attack algorithm is shown in Figure 2.4. We want to know whether video CAPTCHAs can be passed by extracting any text-words present within the video's *key frames*, and entering a subset of those words into the CAPTCHS challenge as answer tags. A key-frame of a video is any frame at which the video presents a significant, visual change (e.g. cutting to a new scene). The following, two hypotheses will be tested:

1. Tesseract can accurately classify words within the key-frames of videos used by Video CAPTCHA, when given the predicted bounding-boxes of the words.

2. Words present in videos are indicative of tags that Video CAPTCHA will accept.

As a concrete example of the problem statement, consider a video being presented by a Video CAPTCHA to a human. Say this video has a key frame containing a sign that reads, *"Danger: Construction Site. Mandated by New York law"*. In this case, a user should pass this test if they enter a set of tags from this sign, e.g. {*'Construction', 'New', 'York'*}. Now consider the same scenario, but with the video being presented to a computer. We wish to know *(a)* if and how well the computer can extract these words, and *(b)* whether the extracted text could then be used to pass the CAPTCHA test.

Key-frame sub regions containing words are automatically extracted using a word segmenter (section 3.3). Tesseract-OCR (section 2.4) is used to classify the extracted words into actual text. Attacks are then made against video CAPTCHA (section 3.4) using the classified words.

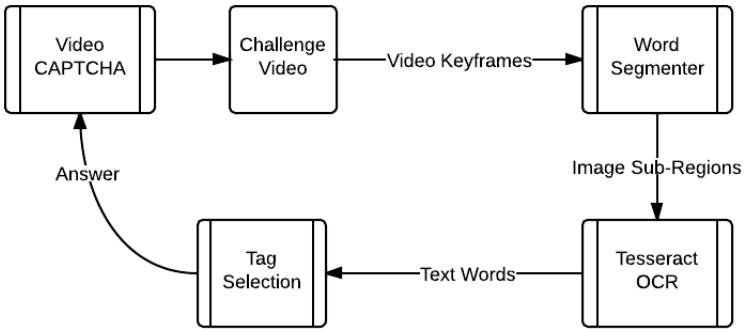


Figure 2.4: A flowchart displaying the inputs and outputs for each step in solving breaking Video CAPTCHA.

2.5.1 Defining an Attack

To test the hypotheses, *attacks* will be made against the Video CAPTCHA system. The goal of an attack is to pass the challenge being presented by a Video CAPTCHA; challenge grading is described in section 2.3.

An attack on the system is formally defined as the following process:

1. Consider as targets the set V of YouTube videos that may be obtained using the random-walk and tag-expansion algorithms described in [9].

2. Let $V_i \in V$ denote the video currently being presented.

3. Let F_i denote the key-frames of V_i. Manually create a set F_m^i of binary masks corresponding to each frame $\in F_i$.

4. Extract word bounding-boxes from a each mask $\in F_m^i$; send the mask and extracted regions to Tesseract for OCR.

5. Save Tesseract's output to an extracted-tag set T_i.

6. Define an *attack* as an attempt at extracting T_i given V_i, and computing an answer set $A_i \subset T_i$ that hopes to pass as input to the CAPTCHA. The process of selecting tags from the set T_i is described in section 3.4.

Chapter 3

OCR-Based Video CAPTCHA Attacks

The process of attacking a video CAPTCHA (defined in section 2.5.1) consists of five stages.

1. Key frame extraction and character labelling.
2. Pre-processing of the labelled masks.
3. Word segmentation on the pre-processed masks.
4. OCR on the original mask, using the word sub-regions obtained from a segmentation over the pre-processed mask.
5. Answer tag generation.

3.1 Key-Frame Extraction and Masking

Key-frames (F_i) for a video are identified using basic *image differencing*. An image is generated that represents the pixel-level differences between two other images. If two frames in the same video differ beyond a certain threshold (defined by the *FFmpeg*[1] tool), then the location of the latter frame is considered to be a key-frame. This leverages the tendency for non key-frames in videos to differ by very little. Using only key frames of a video, as opposed to each frame, is beneficial for two main reasons: *(a)* processing time is

[1] http://www.ffmpeg.org/

drastically reduced; *(b)* less garbage data is present in the OCR results, causing the process of filtering T_i to produce more indicative tags.

Extracted frames are converted to *character masks*. Masks represent the location of text characters within the original image. We manually[2] groundtruth these masks, using color quantization and filtering methods (further details are described in [21]). Quantized masks may contain noise from the process, but should not contain any non-text objects from the original image. An image before and after masking can be seen in Figure 3.1.

3.2 Image Pre-Processing

Before being segmented, character masks are fully binarized and denoised. As shown by the multitude of speckle-like CCs in Figure 3.1b, the manual masking process is very prone to leaving noise behind. Pre-processing drastically reduces the number of connected components (CCs) that must be considered by the segmenter. The goal of pre-processing is to denoise the original mask while fully preserving its structure. OCR may then be formed on the original mask, using word sub-regions defined over the pre-processed mask. The benefits of denoising are seen during word segmentation; noise confuses and slows down the segmenter significantly.

A median filter is first used on the grayscale mask, with parameters $radius = 2px$ and $\% = 75$. This removes a substantial amount of the noise caused by the character-masking process. Noise around the edges of the mask, as well as general small speckles, are greatly reduced.

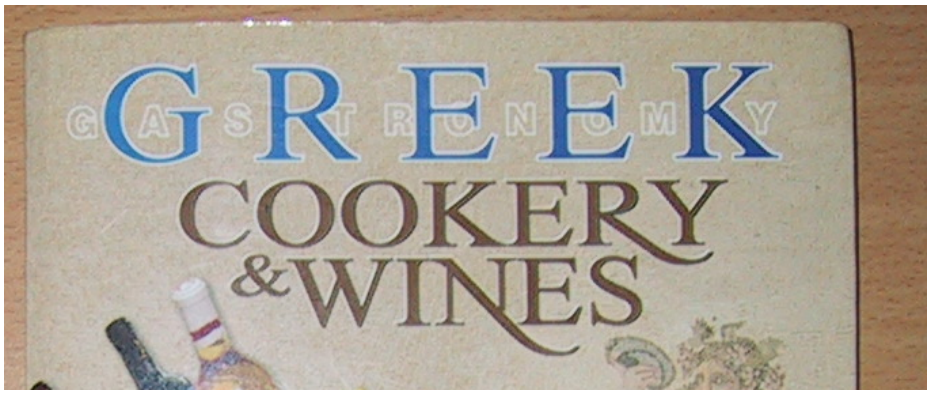
Binarization is then performed on the mask. This process converts the character mask from grayscale to binary (containing only the colors black and white). This step removes a great deal of color noise around the edges of the mask, and lowers the amount of image information that the segmenter must consider. A simple thresholding procedure is used to classify each pixel in the mask as black (background) or white (foreground). For each pixel

[2] A manual approach is taken as text localization is outside the scope of this work.

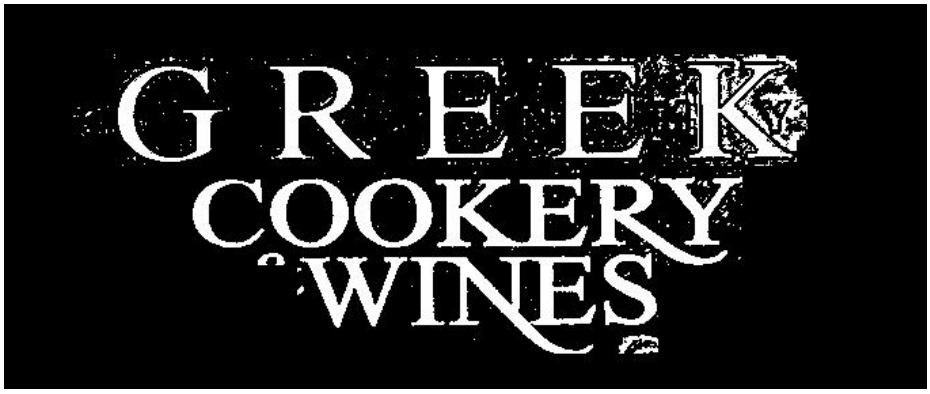
p in a mask M with threshold M_t, $p = white$ if $M[p] > M_t$. Any pixels not labeled as white in this way are labeled as black. Each threshold M_t is chosen using Otsu's method on M, a widely used thresholding technique described in [15].

Lastly, small connected components are filtered out. This step removes any remaining speckle noise. First, any CCs with an area $\leq 5px$ are removed. The mean size μ of the reaming connected components is then computed. Any CC smaller than $1/5$ of μ is removed from the mask.

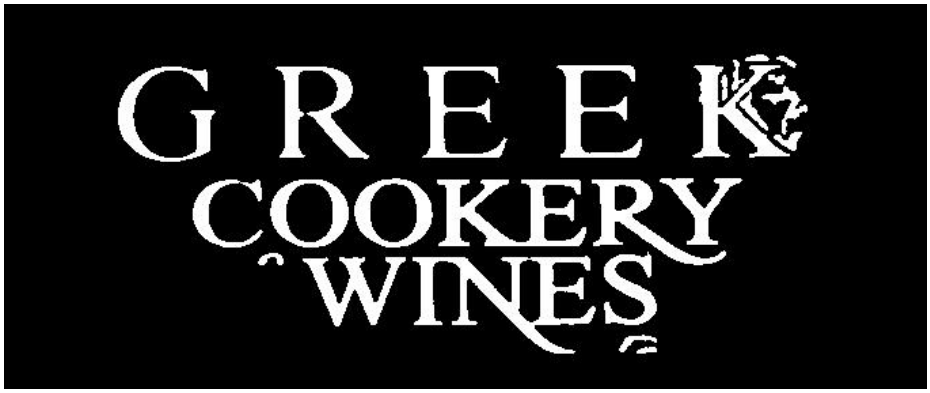
An example of the final product is shown in Figure 3.1.



(a) Initial image, before masking.



(b) Input after masking.



(c) Binarized and denoised image mask.

Figure 3.1: Displaying steps in the pre-processing chain. *(a)* The original image in which words are to be segmented, before masking. *(b)* Locations of characters are repsetned by a grayscale masking of the original image, using color quantization and filtering. *(c)* The mask is binarized using Otsu's thresholding method, and denoised via a median filter and removal of small CCs.

3.3 Segmenter Design

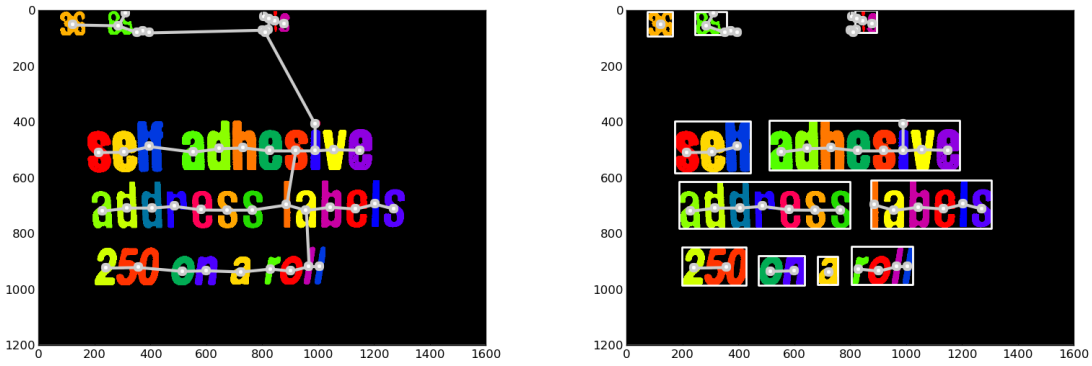
This section describes two algorithms that are developed to segment text-characters present in natural scenes into words. The segmenters accept text pixels as input, in the form of a binary character-mask (as described above). Segmentation output is a set of bounding boxes, where each bounding box should fully enclose exactly one word on the input mask. Providing Tesseract with these sub regions prior to OCR is expected to improve OCR results, by focusing Tesseract on only the relevant portions of an image. This is particularly important for text in natural scenes, as there is a large amount of noise present that is likely to confuse Tesseract. If the noise is severe enough, characters will be unrecognizable by both humans and OCR techniques.

First, a baseline segmenter is developed. The baseline is a semi-naive approach. A more advanced segmentation method is then developed to improve on the baseline, using bagged C4.5 decision trees. The goal of this second approach is to maximize the reliability and accuracy of results. The information gained through the development of the baseline approach is leveraged in-order to accomplish this.

3.3.1 Baseline Approach

The baseline segmenter is described by Algorithm 3.1 below, and visualized in Figure 3.2.

Connected components (CCs) of the input image (img) are first identified, and placed into a labeling array *l*. The value of *l[i,j]* is the unique, numerical identifier of the CC that occurs at *img[i,j]*. If no CC exists (i.e. a background location) at *img[i,j]*, then *l[i,j]* = 0. The CCs of *img* are now described by *l*.



(a) Constructed, Euclidean Minimum Spanning Tree.

(b) Segmentation result after cutting.

Figure 3.2: *(a)* Connected components of the pre-processed image are labelled (a color change signifies a new CC), and a Euclidean Minimum Spanning Tree is constructed from the CCs centers of mass (displayed as nodes); *(b)* The EMST is cut, using the function *slice(u,v,α=0.3)* described in equation 3.1. Boxes around the resulting word segments are shown.

CCs are then merged such that they represent whole words in the image. To accomplish this, all CCs are connected and represented by a Euclidean Minimum Spanning Tree *(EMST)*[3]. Given a connected graph $G = (V, E)$, a *spanning tree* of G is a subgraph that both connects all V and is a tree. Given a set of weights belonging to E, a Minimum Spanning Tree (MST) of G is a spanning tree of G with the lowest possible sum of edge weights. A Euclidean MST is an MST over a graph defined in Euclidean space, with edge weights representing the Euclidean distances between two vertices. Kruskal's algorithm [10] is used to construct an EMST.

A vertex in the EMST represents the center of mass of a CC in Euclidean space. An edge e=$\{u,v\}$ in the tree represents an undirected, "connected" relationship between CCs u and v. An edge's value is the Euclidean distance between the two centers connected by that edge. An example of such an EMST is shown in Figure 3.2a. Any two vertices in the EMST that can reach each other are considered to be part of the same word. Since this

[3]EMST is generated using the SciPy library's implementation: http://docs.scipy.org

initial EMST connects all CCs, the entire mask is initially set as being one word. The edges of the EMST must now be cut to form separate words.

Each edge is considered for a cut one at a time. The decision to cut an edge *e*={*u,v*} is a function of the: *(a)* relative differences in the average size of the CCs u and v[4] *sizeAverage(u,v)*; *(b)* relative differences in positions of the bounding boxes of *u* and *v*, *dx(u,v)* and *dy(u,v)*; *(c)* a static, thresholding parameter α. The edge is cut if the inequality described by equation 3.1 holds true for that edge. The optimal value for α is obtained via a grid search on α over the range $0.0 \leq \alpha \leq 2.0$ (described in section 4).

$$split(e = \{u, v\}, \alpha) = \begin{cases} dist(u, v) > sizeAverage(u, v)\alpha & = \text{true} \\ \text{otherwise} & = \text{false} \end{cases} \tag{3.1}$$

The relative size difference between *u* and *v* (described by equation 3.2) is defined as the average length of the diagonals of their bounding boxes.

$$sizeAverage(u, v) = \frac{\sqrt{u.w^2 + u.h^2} + \sqrt{v.w^2 + v.h^2}}{2} \tag{3.2}$$

The relative difference in position between *u* and *v*, described in equation 3.5, is defined as the maximum of *dx* and *dy* (equation 3.3). The *dx* and *dy* values measure the maximum distances between the vertical and horizontal edges respectively of each bounding box.

$$dx(u, v) = max(u.x - (v.x + v.w), v.x - (u.x + u.w)) \tag{3.3}$$

$$dy(u, v) = max(u.y - (v.y + v.h), v.y - (u.y + u.h)) \tag{3.4}$$

$$dist(u, v) = max(dx(u, v), dy(u, v)) \tag{3.5}$$

[4]For convenience, the minimal bounding-box of a connected component *n* is denoted by: *n.x, n.y, n.w, n.h*, where *x* and *y* represent the coordinate of the top-left corner of the bounding box.

After cutting the EMST in this way, the result is a minimum spanning forest where each tree in the forest represents a word. Word level bounding-boxes are then easily obtained by merging the bounding boxes of the the CCs within each forest. These merged bounding boxes are returned as word segments. An example of a resulting segmentation can be seen in Figure 3.2b.

Algorithm 3.1: Baseline Word Segmentation

input : a binary image mask $mask$, representing the locations of text within an image

a thresholding value α

output: word level bounding-boxes of $mask$

$l \longleftarrow$ the connected-component labelling array of $mask$

$centers \longleftarrow$ centers of mass for each $label \in l$

$emst \longleftarrow$ MinimumSpanningTree($V = centers, E = dists$)

for *edge* $e = \{u, v\} \in emst.E$ **do**

    if $split(u, v, \alpha) = true$ **then**

        remove the edge e from $emst$

$words \longleftarrow$ bounding boxes of each connected component of $emst$

return $[words]$

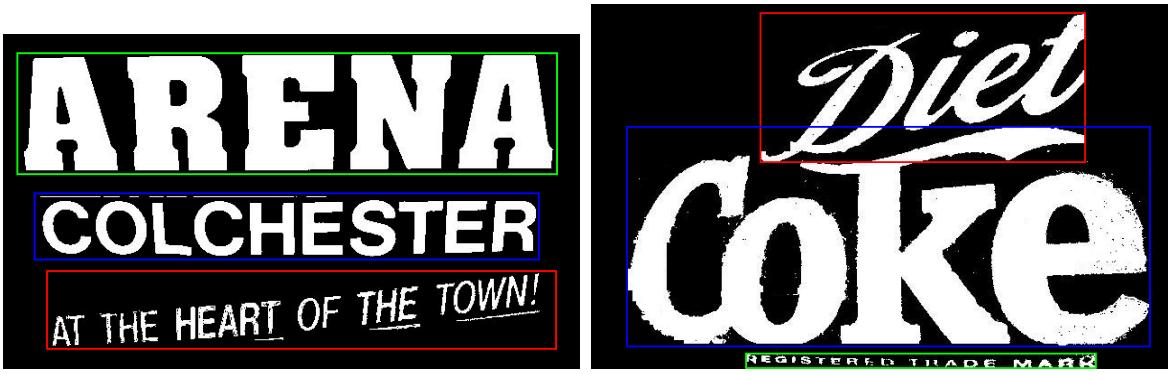
3.3.2 Bagged Decision-Tree Approach

An improved segmentation method takes a more adaptive, intelligent approach to the problem. Beginning steps are connected component (CC) analysis and labeling, as performed in the baseline. Three additional steps make up the segmenter: 1. text line segmentation, 2. feature extraction; 3. word segmentation (classification).

A bagged decision tree classifier is pre-trained to segment *text lines* into words. Examples are constructed from CCs lying along the same text line, using the text line as context. A text line can be defined [13] as a group of text characters that are visually aligned (i.e.

characters that share a writing line). Alignment is determined in-terms of some primitive image unit, such as CCs or pixels. A text line segmentation is the merging of primitives into their corresponding text lines. An example of a text line segmentation on two natural text images is shown in Figure 3.3.

Segmenting CCs into text-lines takes advantage of two observed properties of natural text. First, groups of CCs within the same text line are likely to have little feature variance; an example is shown in Figure 3.3a. Considering only text on the same line means treating each group with similar variance as its own, separate image. Second, two CCs on different text lines are extremely unlikely to be part of the same word. This makes an initial text line segmentation very unlikely to cause errors by itself.



(a) Simple text line segmentation.

(b) A more complex text line segmentation problem.

Figure 3.3: Examples of two text line segmentations of different natural text masks. Changes in bounding box color signify a different text line. *(a)* Displaying how even when the features of all CCs vary substantially, features within the same text lines vary very little. *(b)* A segmentation made harder by the fancy ”hat” on the ”k” character; displaying how the character variance between words in natural text can make the process more ambiguous.

A text line segmentation approach is adopted from *OTCYMIST* [11], a participant in the ICDAR 2011 Robust Reading Competition [7]. The procedure used here is defined in Algorithm 3.2. First, all CCs in the mask are sorted in decreasing order of the top-most edge of their bounding boxes. An agglomerative clustering procedure then clusters CCs into line groups. Each CC initially belongs to its own group. During each pass of clustering, two

groups are greedily merged if the centroid of one group is within the vertical range of the other group's bounding box. Clustering is stopped when the process converges.

Algorithm 3.2: Text line segmentation via agglomerative clustering

input : CCs - Connected components of a binary text-mask.

output: A text line grouping of CCs.

```
1  Initialize a set of groups G
2  Assign to G each cc ∈ CCs
3  Sort each G in decreasing order by the top-most bounding box edge of each cc ∈ G.
4  while G has not converged do
5      Set G as converged
6      for group g ∈ G do
7          Assign to g_y the y-coordinate of the centroid of g
8          for group q ∈ {G − g} do
9              Assign to bb the bounding box of group q
10             if g_y ≥ bb.y and g_y ≤ bb.y + bb.h then
11                 Merge q into g
12                 Remove q from G
13                 Set G as non converged
14                 Break and begin a new pass
15 return G
```

Feature Extraction

A Euclidean Minimum Spanning Tree (EMST) is created over each text line, as opposed to the entire image; EMST creation is described in the baseline approach. This allows each text line to be segmented in its own context. Features compare each pair of CCs connected by an EMST edge.

Features are geometric properties of the two CCs being compared. In the list below, A and B represent the two CCs being compared, and Ln represents the text line containing

both A and B.

Center Distance: Euclidean distance between the centroids of A and B. Normalized by the average size of their bounding boxes as defined in equation 3.2.

Bounding Box Distance: Minimum bounding box distance $dist(A, B)$, as described in equation 3.5. Normalized by average bounding box size, as described in equation 3.2. This feature is similar to the single feature used in the baseline. It is defined as:

$$BBDist(A, B) = \frac{dist(A, B)}{sizeAverage(A, B)} \tag{3.6}$$

Convex Hull Distance: Minimum distance between the convex hulls of A and B. This is more precise than the bounding box distance, as the convex hull of a shape is essentially a fitted, bounding polygon of that shape. A *convex set* in Euclidean space is a set of points P such that for each pair of points $(P_i, P_j) \in P$ and the straight line segment s connecting them, each point along s is also $\in P$. A *convex hull* of a shape X is the intersection of all convex sets that contain the points $\in X$ (i.e. the smallest such convex set).

The minimum distance between two convex hulls X and Y is defined [5] as the minimum distance between the convex hull of their *Minkowski difference* and the origin. The Minkowski difference of X and Y is defined as $X \ominus Y = \{x - y : x \in X, y \in Y\}$, where x and y are position vectors representing points in the hulls. A visual example of this property is shown in Figure 3.4. The distance is normalized by the area of $A \ominus B$.

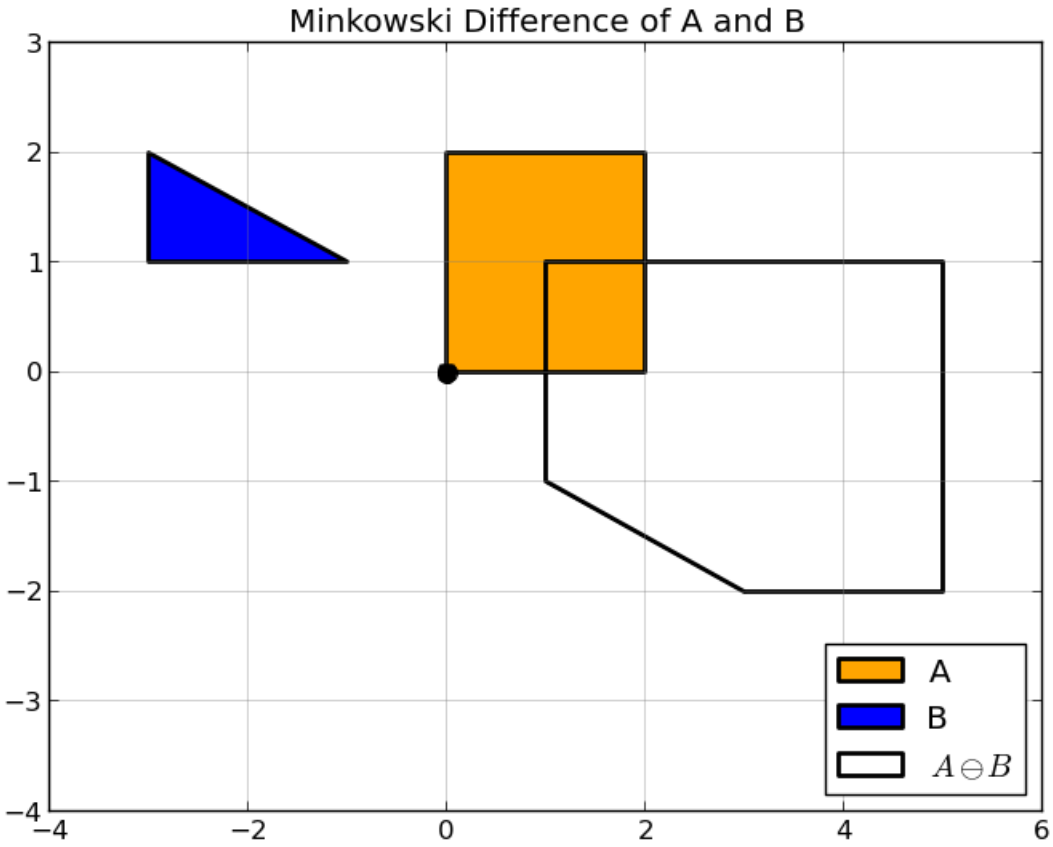


Figure 3.4: The Minkowski difference $A \ominus B$ of two convex sets A and B. The minimum distance between A and B is found by taking the minimum distance of $A \ominus B$ to the origin $(0, 0)$.

Nearest Neighbour Distance: A nearest neighbour distance between A and B is described by a *k-nearest neighbour graph* over the text line containing A and B. The feature is normalized by the total number of CCs within Ln. This distance is not commutative, i.e. if A is the kth nearest neighbour of B, B is not necessarily the kth nearest neighbour of A. To account for this, the minimum of the two nearest neighbour distances is used.

1-NN: Whether A is the nearest neighbour of B or B is the nearest neighbour of A. This is a discrete feature, with possible values of $\{True, False\}$.

X-Gap: Compares the horizontal gap dx between A and B with the horizontal gap

between CCs immediately neighbouring A and B.

Let A_n and B_n denote the CCs on Ln requiring the fewest edge traversals to reach when starting from A and B respectively (i.e. the immediate neighbours of A and B). For each $a \in A_n$ and each $b \in B_n$, record to a set D the values $dx(A, a)$ and $dx(B, b)$. Select $d = max(D)$, and compute the percentage d is of $dx(A, B)$. This percentage is the feature value.

Y-Gap: Equivalent to the x-gap, but measures vertical distance using $dy(A, B)$ instead of horizontal distance.

Between Angle: The angle θ of the slope between the centers of A and B. This feature purposefully does not take into account the quadrant in which θ lies. Let (X_{cx}, X_{cy}) denote the (x,y) coordinate of the center of a CC X. The feature $BetweenAngle(A, B)$ is then defined as:

$$BetweenAngle(A, B) = arctan(|\frac{B_{cx} - A_{cx}}{B_{cy} - A_{cy}}|) \tag{3.7}$$

Aspect Ratio Difference: Ratio of the area of the convex hull of A to the area of the convex hull of B. The ratio is taken of the larger CC to the smaller CC.

Area Difference: Difference in the $width/height$ ratios of the bounding boxes containing A and B. It is defined as:

$$AreaDiff(A, B) = |(A_w/A_h) - (B_w/B_h)| \tag{3.8}$$

X/Y Overlap: Percentage of A that overlaps B. X-overlap and Y-overlap are the percentages of horizontal and vertical overlap respectively, and are used as two separate features.

Classification

A bagged decision-tree classifier segments text lines into text words. Figure 3.5 demonstrates a text word segmentation. The classifier looks at an edge of an EMST, and predicts whether or not that edge should be cut. A *bagged classifier* forms an *ensemble* of base learners to equally vote on each classification outcome. For each ensemble learner i, a

training set T_i is created for it by randomly sampling the initial training set T uniformly and with replacement. This method of sampling is known as *bootstrapping* [12] (explaining the naming origins of ***B**ootstrap **AGG**regat**ING***). In the case of this work, 10 such classifiers are used to form the ensemble. The size of each sample is the size of the initial training set.

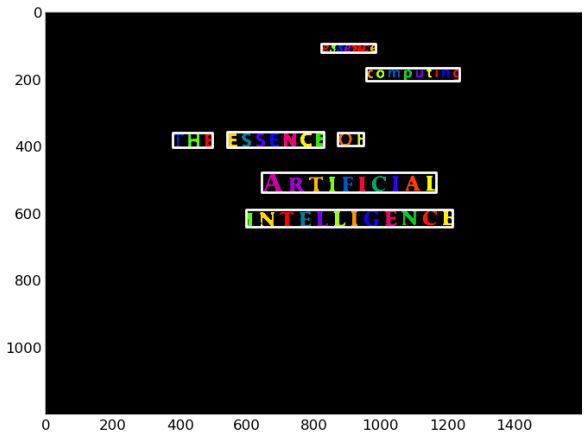


Figure 3.5: Text lines are segmented into text words, via a bagging of C4.5 decision tree classifiers.

A C4.5 [19] decision tree is used as the base learner. For a set X of feature vectors and a set Y of class labels, a *decision tree* forms a predictive, tree structure over a set of training examples $S = \{(x \rightarrow y)|x \in X, y \in Y\}$. The tree classifies an example by answering a series of questions based on pre-defined decision tests, eventually arriving at the decision to classify the example. Tests are defined by decision nodes, and classifications by leaf nodes. The tree constructs decision nodes by partitioning S on a feature f, selected based on some criteria. A decision node is created containing some test on f, with a branch representing each possible outcome of the test.

C4.5 is a type of decision tree that uses the *gain ratio* of a feature as the selection criterion. Gain ratio is a normalized version of *information gain*; information gain seeks the partition feature that will result in the least confusion (*entropy*) within the data. Information gain has an inherent bias towards splits containing a greater number of possible outcomes, as these are often excellent choices as far as entropy is concerned. However, favoring splits

with many outcomes often results in a tree that overfits the data. The gain ratio criterion reduces this bias, by emphasizing the information gained from a split that is actually useful for classification [19].

The feature vector X is created by extracting the features of each text line in each training image (features are described in section 3.3.2). An EMST $M_L = (V, E)$ is created over each text line L. Training examples are then generated by extracting features from each pair of CCs (A, B) directly connected by an edge $e = (A, B) \in M_l.E$. If e should be cut by the segmenter, the example is classified as $split$; otherwise the example is classified as $merge$.

3.4 Attacking Video CAPTCHA

This section presents the methodology used to perform both manual and automated attacks on video CAPTCHA. Manual attacks are performed in-order to remove the variables introduced by both Tesseract and the word-segmentation algorithms.

A single sample set $S \subset V$ of size 13 is used throughout all attacks. S_v denotes the video currently being processed. S is generated by performing a random walk over YouTube videos, as described by Algorithm 2.1. S is then manually filtered such that:

- Each video contains human readable text in at least one of its key frames.
- Each video has at least three groundtruth tags associated with it.
- No video runs over five minutes in length, or features inappropriate content.

The following is a template for attacking a challenge video S_v. First, the tag set T_v for the attack is generated. For a manual attack, each identified key-frame in S_v is observed, and any text present is manually inserted into T_v. Only words that are considered to be human readable are manually recorded. In the case of an automated attack, each key frame is fed to the word segmenter as described in section 3.3. Tesseract then classifies the text present within each resulting, image sub region. All text returned by Tesseract is added to

T_v. The answer set $A_v \subset T_v$ is then created and sent to the CAPTCHA as answer tags. The attack is successful if the CAPTCHA is passed using A_v.

The answer set A_v is created by filtering the extracted-tag set T_i to include three answer tags. A challenge allows for only one attempt at, so tags from T_v must be carefully chosen for inclusion in A_v. This filtering process leverages publicly available information on the CAPTCHA system [8, 9].

The first step is to pre-process the tags in T_v. In this step, tags that would clearly be rejected by the CAPTCHA system are excluded from further consideration. Pre-processing consists of:

1. String sanitization
2. Removal of tags with < 3 characters
3. Removal of stop words
4. Frequency-based pruning

String sanitization intends to reduce noise and ambiguity in the extracted tags. Each tag in T_v is stripped of excess white-space (which may be present due to errors in OCR), and then has any ending punctuation removed. Punctuation at the start or in the middle of a tag is not removed. This is to avoid invalidating tags that have punctuation in their semantic meanings; semantically meaningful punctuation is less likely to occur at the end of a tag. This is particularly relevant due to YouTube tags often referencing pop-culture, or otherwise not being ”proper” words. For example: a tag that references an online handle, ”R.I.T.”, should have all punctuation left intact.

After string sanitization, *stop words* $\in T_v$ can be more accurately removed. Stop words add no descriptive value, and are automatically rejected by the system [9]. Examples of stop words are, ”if”, ”then”, ”you”, etc. The listing of stop words used to filter T_v is the same list used by the CAPTCHA system to filter inputs. This ensured that none of the three guesses are wasted due to the inclusion of such tags, and that further steps are not

bogged down by having to consider them. Any reaming, pre-processed tags composed of two or fewer characters are then excluded. These are likely non-descriptive tags that escaped removal thus far; very short are unlikely to be descriptive.

Frequency-based pruning eliminates tags with an estimated *global frequency* that would likely cause them to be rejected by the CAPTCHA system. The global frequency of a tag is defined as the probability of randomly selecting that tag from a sample of the population, consisting of all tags in all YouTube videos. This step mirrors the frequency-based pruning method employed by the CAPTCHA system. The CAPTCHA system uses a pruning threshold ($t = 0.005$) [9] to purge tags that occur with an estimated frequency $\geq t$ (see section 2.3). To prevent selecting tags that are likely to be pruned in this way, all tags $\in T_v$ that have a global frequency $\geq t$ are removed.

In order to perform the frequency-based pruning step, our own tag-frequency data must be obtained. Even if it was assumed that we had access to the same random-walk data used by the CAPTCHA system, that data is currently about five years old. Given that cultural trends change frequently, five year old tags for a social site (such as YouTube) are likely to be less descriptive of the overall data than they once were. A new frequency table is generated in the same way that the CAPTCHA system generated its own table. The process consists of updating the system's code [8] and re-running the tag-scraping routine over 30,522 videos.

The final step is to select the three remaining tags with the highest local frequencies. A tag's *local frequency* is defined as the probability of selecting it at random from T_v. In other words, this step examines the number of times a word appears in S_v. Tags present in multiple frames do not necessarily have higher local frequencies, as the same tag appearing many times in a single frame is counted equally. The three tags with the highest local frequency are selected. Use of this simple heuristic is based on the assumption that the more often a descriptive word appears in a video, the more descriptive of the video that word is. Clearly this is not always the case, but it is a reasonable assumption to make without going into the realm natural language processing.

In the event of a tie (i.e. two tags have the same local frequency), the tag with the greater *global* frequency is chosen. Since all tags that would be pruned due to a high global frequency have been removed, there is far less danger in attacking based purely on tag frequency. Tags with a higher, global frequency are statistically more likely to describe to any given video. If the two tags also have the same global frequencies, one is selected purely at random.

Chapter 4

Experiments

The experiments used to test the hypotheses are presented here. Through these experiments, three key questions are evaluated:

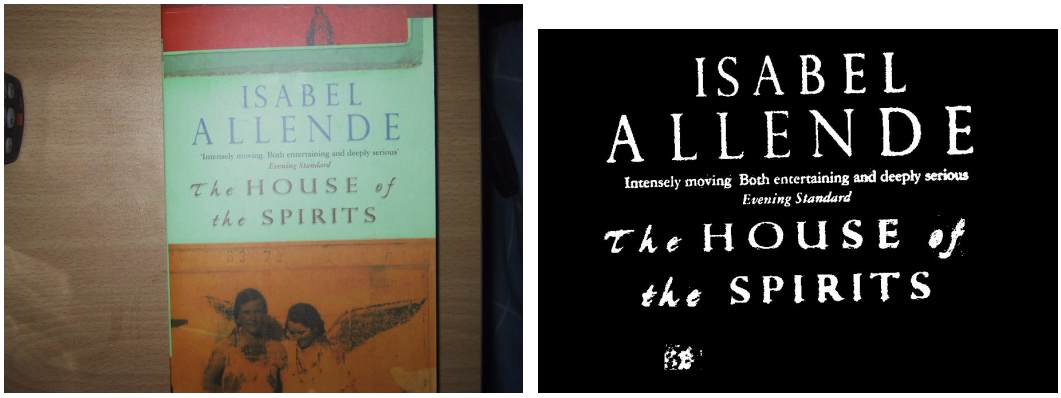
1. Are the words present within the CAPTCHA's challenge videos useful in passing the challenges?

2. Can Tesseract's OCR classify those words accurately and reliably?

3. Can we automatically obtain those words reliably and accurately?

A measure of the ability to obtain image sub-regions of natural text words is first evaluated. Evaluating the performance of the natural text segmenters uses a dataset of 221 manually generated, binary character-masks (as described in section 3.2). The original images used to generate the masks are all from the *ICDAR 2003 Robust Text Detection Competition* [14]. An example of a competition image and its corresponding mask is shown in Figure 4.1. Groundtruth image sub-regions are provided for the words in each image.

Types of imagery present in the ICDAR dataset include both photographs and scans of book covers, signs, logos, and others. The image masks have varying levels of noise, from minor to severe. Noise is present in the masks due to *(a)* the noise present in the original ICDAR images, and *(b)* the color quantization and filtering methods used to generated the masks. Noise helps the results better reflect Tesseract's performance in natural scenes, in which noise levels are often unpredictable.



(a) Original, scaled ICDAR 2003 image. (b) The manually generated, binary mask.

Figure 4.1: *(a)* An image from the ICDAR 2003 Robust Text Detection Competition; *(b)* the manually generated, binary mask of that image.

Data is randomly split into a training set containing 151 images, and a testing set containing 70 images. Parameters for both the baseline and bagged segmenter are learned over the training set. For the baseline segmenter, only the thresholding parameter α must be learned. A grid search over the training set is performed to find the optimal α, using the range $0.000 \leq \alpha \leq 2.000$. The α with the best resulting evaluation metric is chosen for use on the testing set. Different values are compared and evaluated based on their resulting recall, precision, and f-measures over the training set.

For the bagged segmenter, the C4.5 decision-trees are trained over all features extracted from each image of the training set (as described in section 3.3). As there are more *merge* examples than there are *split* examples, the dataset is proportioned to include 1 split example for every 2.5 merge examples. This avoids the issue of C4.5 over-fitting one dimension of the problem, while under performing on the other. More merge examples than splits are used for two reasons: 1. over segmentation is preferred to under segmentation, as Tesseract can still classify two words in the same sub-region; 2. C4.5 is likely to have insufficient examples to generalize from if a 1:1 class ratio is used in this instance. A ratio of 1:2.5 helps alleviate over fitting the merge class, while still supplying sufficient information.

Tesseract is then run in *PSM_WORD* mode on each original image, using the sub-regions predicted by the segmenter. The term ”original image” refers to the character

mask before pre-processing. The pre-processed image is created to aid the segmenter, and may cause distortions in the characters that would make them poor candidates for OCR. PSM_WORD instructs Tesseract to consider each sub-region to represent exactly one word. OCR results are compared against the groundtruth text for evaluation.

A series of attacks is then made video CAPTCHA, using a sample set containing 13 videos. A manual and an automated are both conducted for each sample video, using the method described in section 3.4.

4.1 Metrics

This section details the evaluation metrics used in the experiments.

4.1.1 Segmenter Evaluation

The performance of both the baseline and bagged segmenters are evaluated at the *object* level, as opposed to the *primitive level*. Primitives are individual connected components of an image; objects are then sets of symmetrical relationships between connected components (primitives). In the most basic case where each connected component in the image fully and only represents one letter, primitives are letters and objects are words.

The goal of a segmentation is to recognize the set of relationships between primitives that most correctly represents all objects (i.e. to get the words). The result of a segmentation is evaluated based on how many objects are fully recognized and represented. An object is considered recognized if and only if the exact set of relationships that make up that object is detected. There is no in-between score for finding objects; an object is either found exactly, or it is fully over/under-segmented. This method of evaluation is performed by the *LgEval* tool [23], which treats the image as a graph of its primitives.

The final output of the segmenter is a predicted primitive-graph. Output is compared to a corresponding ground-truth graph for evaluation. For a predicted graph/segmentation G: the set of ground-truth objects associated with G is denoted G_t, and the set of all objects found/predicted by G is denoted G_p. Metrics used to evaluate the predicted graphs are then:

Recall The fraction of ground-truth objects that are found.

$$recall = \frac{|G_t \bigcap G_p|}{|G_t|}$$

Precision The fraction of found objects that are in the ground truth.

$$precision = \frac{|G_t \bigcap G_p|}{|G_p|}$$

F-Measure Overall effectiveness, defined as the harmonic mean of recall and precision.

$$F = 2\frac{(\text{precision}) \times (\text{recall})}{(\text{precision}) + (\text{recall})}$$

4.1.2 Attack Evaluation

The concept of *relevant tags* in a set is used in evaluating attack performance. A relevant tag in a CAPTCHA challenge is any tag that is in the CAPTCHA system's groundtruth set for that challenge. The set of relevant tags pertaining to V_i is denoted V_i^r. It should be noted that for any V_i, the set V_i^r contains a fixed number of tags belonging to videos related to V_i Related tags are extracted and stored before each experiment, to allow for reproducible results.

The following details how the results of the manual and automated attacks are evaluated. Two key processes are measured in-terms of recall, precision, and f-measure: *(a)* text extraction from V_i's key frames into T_i, and *(b)* the selection of three guess-tags from T_i into A_i.

Text extraction metrics evaluate T_i in-terms of its correlation with the groundtruth tags associated with V_i. The interpretation of text-extraction metrics varies in accordance with

the type of attack they are associated with. In a manual attack, the only possible word-recognition error associated with text extraction is due to human error; it is assumed that this error is very low. In this case then, text extraction measures how indicative words found in V_i's key frames are of tags that will be accepted by the video CAPTCHA. In automated attacks, extraction errors result from errors in segmentation and/or errors in OCR/Tesseract. In this case, text extraction measures the ability to obtain and recognize the words in a video.

The metrics used to evaluate the tag extraction process are as follows:

Extraction Recall The fraction of relevant tags that are extracted.

$$TE_{recall} = \frac{\left|T_i^r\right|}{\left|V_i^r\right|}$$

Extraction Precision The fraction of extracted tags that are relevant.

$$TE_{precision} = \frac{|T_i|}{\left|T_i^r\right|}$$

F-Measure The overall effectiveness of tag extraction.

$$TE_{FM} = 2\frac{TE_{precision} \times TE_{recall}}{TE_{precision} + TE_{recall}}$$

Tag selection metrics evaluate the process of selecting extracted tags to use as answer tags. Errors in this process may result only from the pre-processing, filtering, and selection steps. Only precision is used to evaluate the tag selection process, as recall is not applicable.

Attack Set Precision How accurate the tag selection process is.

$$TS_{precision} = \frac{\left|T_i^r \bigcap A_i^r\right|}{\left|A_i^r\right|}$$

A global *Attack Success* measure simply records whether the CAPTCHA challenge was passed. Attack success is summed for each attack in an experiment set to obtain a measure for general effectiveness.

Chapter 5

Results and Discussion

This section provides the results for both types of segmentations and attacks.

5.1 Evaluating Segmenter Designs

Results presented here represent segmentation performance over the ICDAR dataset.

The baseline segmenter is evaluated using the tuning parameter $\alpha = 0.316$, which was found to be optimal in a grid search over 20 even increments in the range $0.0 \leq \alpha \leq 2.0$. Performance over each of the candidate α values is shown in Figure 5.1, in-terms of recall, precision, and F-measure. A different view of the grid-search results is shown in Figure 5.2, which displays the frequency at which each α was optimal for an image.

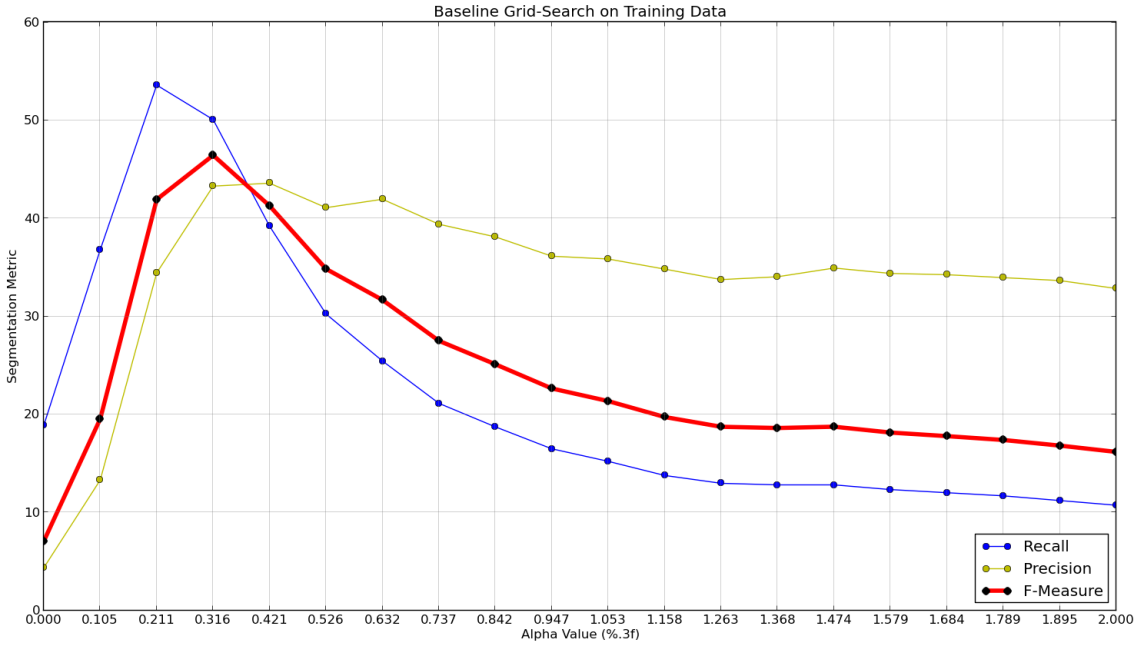


Figure 5.1: Results of the grid search performed to an optimal value for α. For each candidate α value, the resulting recall, precision, and F-measure resulting from a baseline segmentation over the training set are shown. $\alpha = 0.316$ is shown to be optimal, with $F = 46.44\%$.

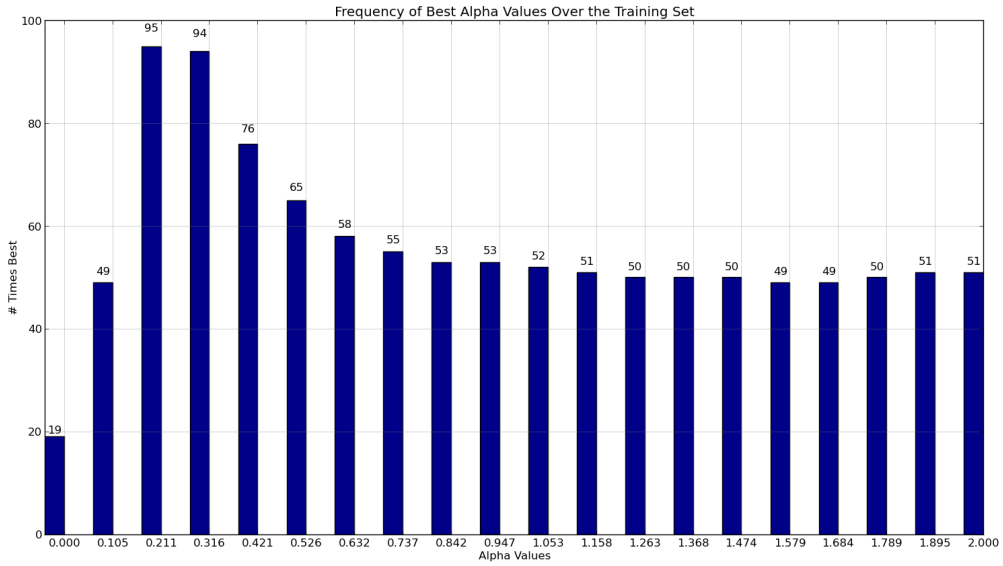


Figure 5.2: Displaying the results of the grid search to find α, in-terms of the number of times each α value was best at segmenting an image. Values are compared by their resulting F-measures. If two or more values yield the same F-measure, they are each recorded as optimal.

Evaluation metrics for the both the baseline and bagged-classifier segmenters are shown in Figure 5.3. Separate evaluations are given over both the training and testing datasets. Baseline measures are obtained using $\alpha = 0.316$. Frequency distributions of F-meausres obtained by each segmenter over the testing set are shown in Figure 5.4 and Figure 5.5.

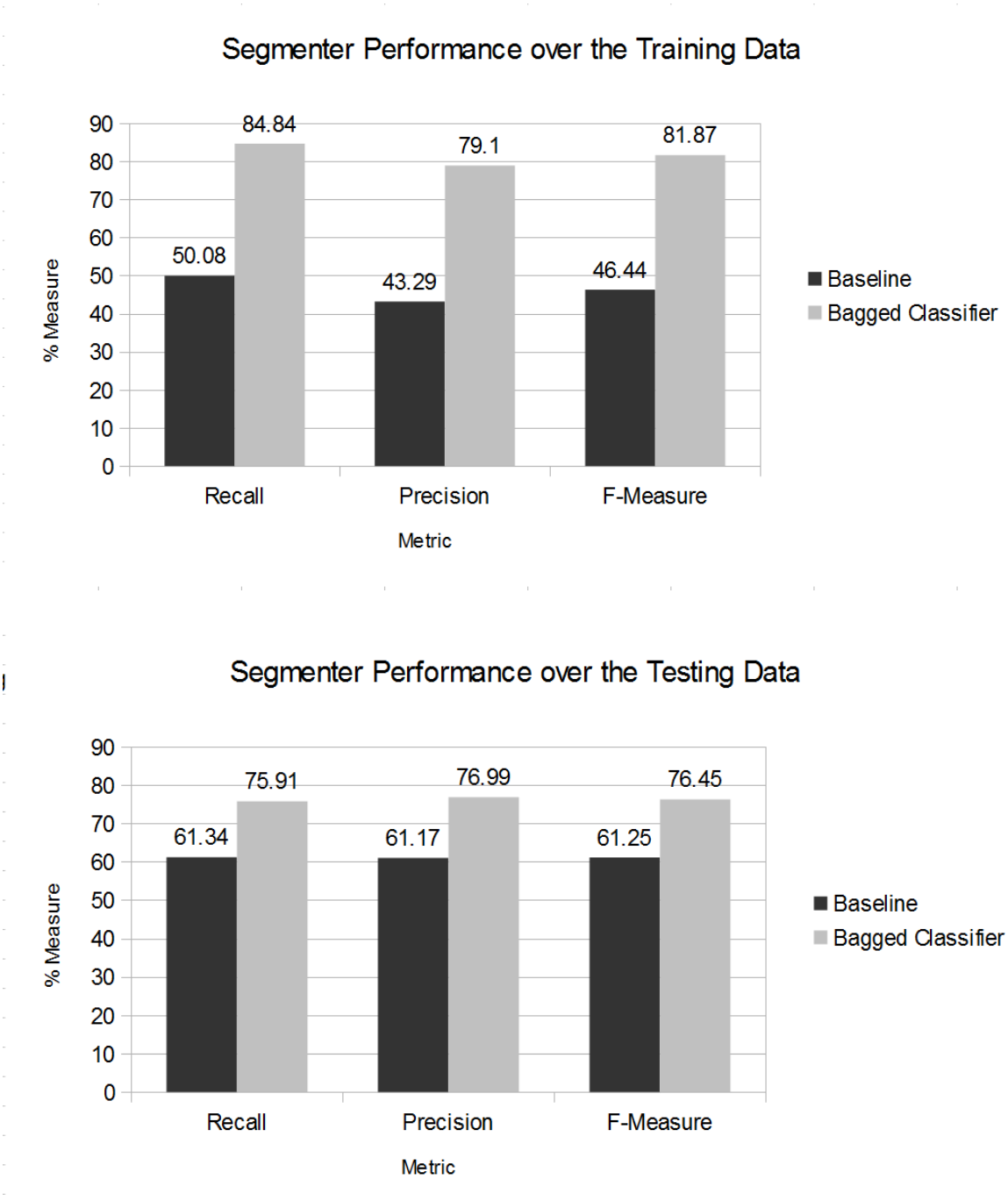


Figure 5.3: Comparing recall, precision and f-measure achieved by the baseline and bagged-classifier segmenters over the *(a)* training data and *(b)* the testing data.

Given the unpredictable nature of natural text, the low numbers for the baseline are

not surprising. However, the causes of both errors and successes later prove critical in improving the approach. One cause of errors is the Euclidean Minimum Spanning Tree construction. Natural text CCs are sometimes arranged such that an EMST over them will never connect all CCs belonging to one or more, particular words. This means that the usage of a raw EMST inherently causes errors in the segmentation, regardless of the edges that are cut. EMST errors were most frequently observed when two diagonal text lines were close to each other, causing the EMST to connect upwards/downwards instead of left/right. A vertical connection generally means connecting different words.

The importance of pre-processing the masks is made clear during preliminary experiments with the baseline. An initial pre-processing step consisted of only a binary closing on the mask, using a fixed size structuring element. Much noise was leftover, which had a very negative impact on segmentation results on the training set. This initial pre-processing yielded an F-measure of 21%, less than half of the 46% shown in Figure 5.3. In addition to greatly improving accuracy, further pre-processing also reduced run time significantly.

The baseline achieved poor performance over the training set when compared with the testing set. Training set F-measure is 46%, while testing set F-measure is 61%. This 15% difference suggests two things: 1. there is a great deal more variance in the training set when compared with the testing set; 2. only comparing bounding box distance is not enough to describe this variance. There are also $1/3$ more images in the training set, making the fitting of a single, static parameter over the data more difficult. Baseline predictions on the testing set are still 10% greater than chance. This implies the baseline feature is quite descriptive, and will be useful when combined with additional information. The feature is carried over to the C4.5 learners in the bagged classifier.

Looking at Figure 5.4, it is observed that baseline segmentation is generally all or nothing. Over the training data, 46% (70) of mask segmentations result in an f-measure in the range [90,100]; 20% of such resulting f-measures fall in the range [0,10]. A likely explanation for this is the static α parameter not providing enough information. In evaluation, a segment is either completely found or completely missed, i.e. there is no middle ground.

If a word has one character with different properties than the rest, then it may be missed due to the inability of a single feature to describe enough variance. The unpredictability of natural text makes this case likely to appear.

A segmentation approach using bagged C4.5 trees over segmented lines is shown to work well. When compared to the baseline, Figure 5.3 shows F-measure increases of 35% over the training set and 15% over the testing set. The large increase in training F-measure points to a model being created that better represents the training data. Pruning of the C4.5 trees accounts for less than perfect accuracy, and is needed to avoid overfitting (which is surely not an issue for the baseline).

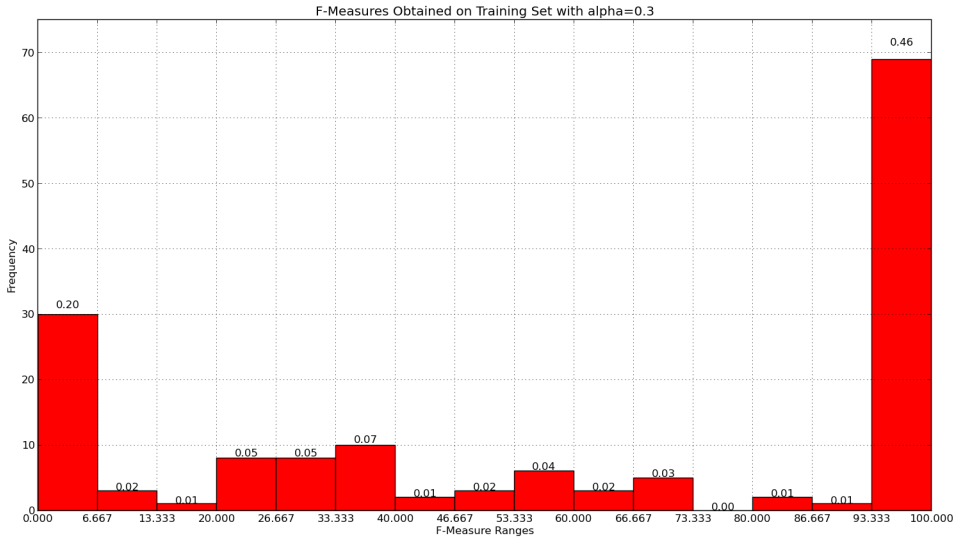
Many reasons exist for these improved results. An immediate cause for improvement is the changing in the way EMSTs are used in segmentation, mitigating the inherent errors caused by EMSTs during baseline experiments. Creating a separate EMST over each text line addresses the observed cause of the error. Text line segmentation also greatly helps in eliminating erroneous choices. The use of a majority-vote classifier is also helpful; bagging of C4.5 trees showed a 4% improvement over the use of C4.5 alone.

When selecting features for the C4.5 trees, the goal is to capture properties which are both simple and highly descriptive of the data. This principle leads to the various distance and gap measures being chosen. X-gap is an example of a feature that is very simple to understand and implement, while also providing great descriptive power (the gap features added nearly 10% to the ensemble's classification rate). The different distance measures (bounding box, convex hull, centroid, and k-nearest neighbor) are similar to each other and easy to implement, while describing different properties of related CCs. Each distance measure may be more or less descriptive than another, often depending on the context of a specific example. For instance: the k-nearest neighbor distance may offer a different look at CCs than the bounding box distance, particularly in the various contexts presented by natural text images. Since the k-nearest neighbor distance is highly sensitive to noise in the image, the other distance measures are needed to cover more of the data. Features concerning overlap and aspect ratio are meant to capture special cases that are encountered in

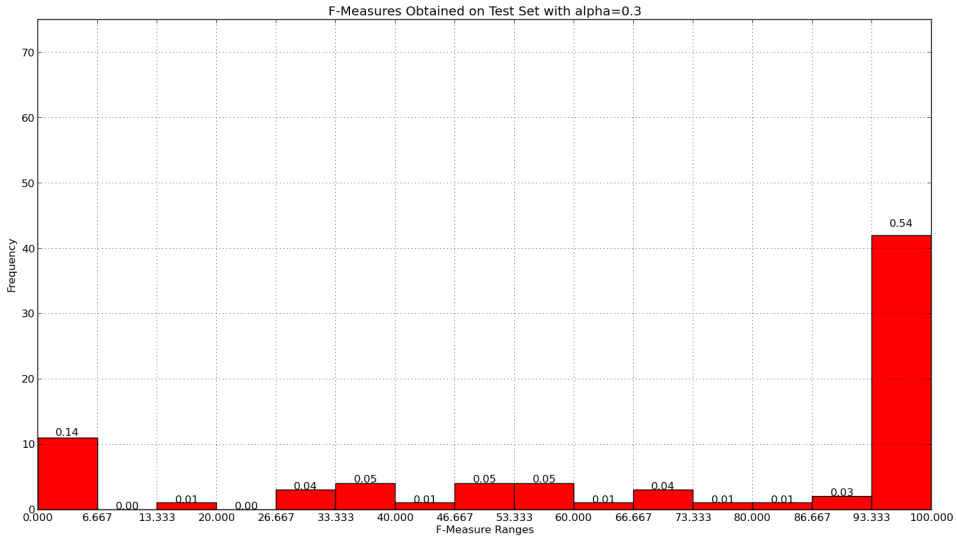
text. Aspect ratio captures characters such as the dash symbol. Overlap captures instances of broken up characters, as well as instances of small text acting as subtext to a larger CC (e.g. a slogan contained below a logo).

Testing set accuracy did not improve quite as much as expected, but improvement is quite noticeable. An inclusion of more visual (non-geometric) features is likely to cause significant improvement. An example of such a feature is the Stroke-Width Transform, discussed in section 2.4. There is only so much information that can be described by geometric measures alone. A 15% improvement may also mean that one or more features are confusing C4.5, causing ambiguity in the data. Removing features one-by-one did not yield improved results however. Additionally, the set of training examples was an issue. Due to the nature of how features are generated, far more *merge* than *split* examples are present in the training set. To help mitigate this, 1 split is taken for every 2.5 merges; experiments using different ratios produced comparably worse results. Availability of more training data in-general would clearly help as well; additional training data is abundant, and just needs to be pre-processed to be used by a classifier.

Overall, the goal of noticeably improving on the baseline is accomplished.

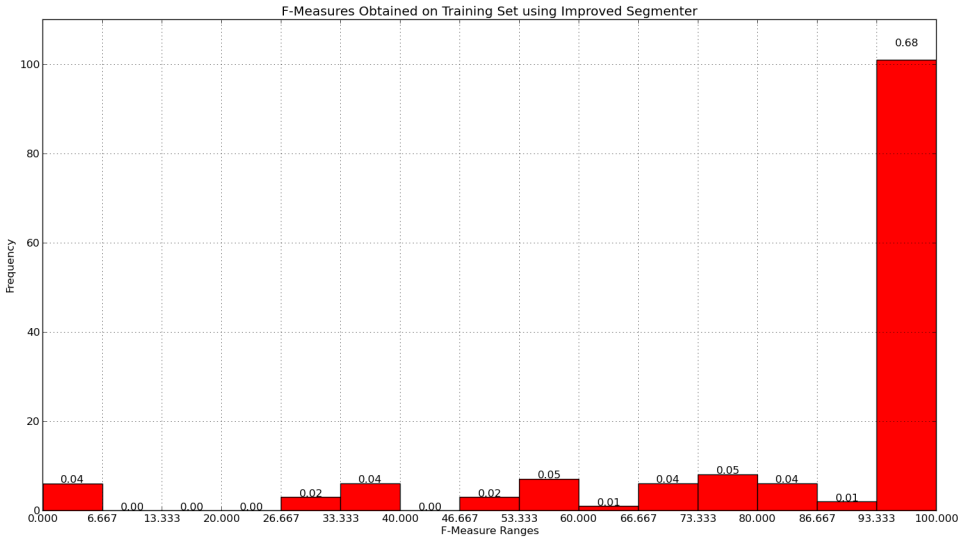


(a) F-measure values achieved by the baseline segmenter over the training set, using $\alpha = 0.316$.

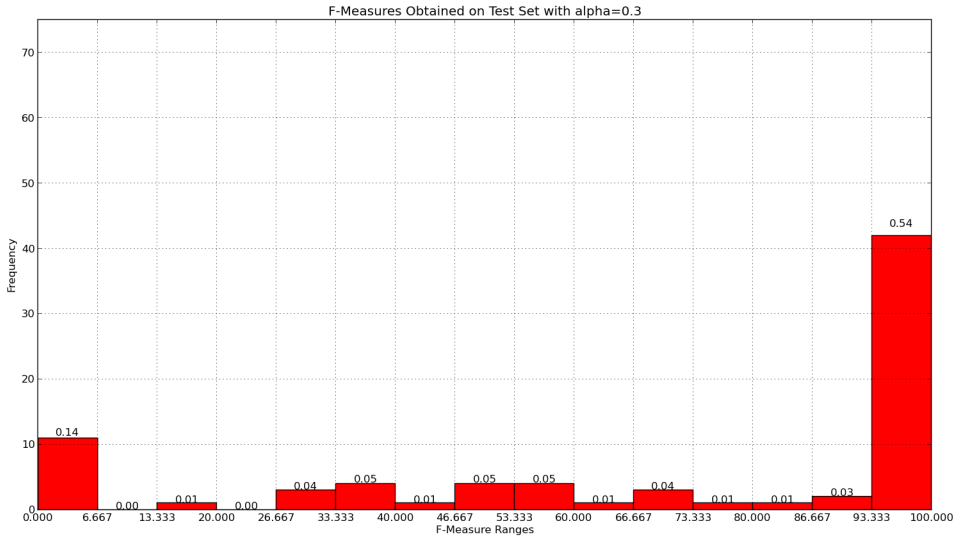


(b) F-measure values achieved by the baseline segmenter over the test set, using $\alpha = 0.316$.

Figure 5.4: Histograms displaying the f-measures resulting from a baseline segmentation on *(a)* the training set, and *(b)* the test set. Relative frequencies are shown on the y-axis; normalized frequencies are labelled above each bar.



(a) F-measure values achieved by the bagged segmenter over the training set.



(b) F-measure values achieved by the bagged segmenter over the test set.

Figure 5.5: Histograms displaying the f-measures resulting from running the bagged segmenter on *(a)* the training set, and *(b)* the test set. Relative frequencies are shown on the y-axis; normalized frequencies are labelled above each bar.

5.2 Results of Attacking Video CAPTCHA

Results for *manually* attacking the videos in the sample set are displayed in Table 5.1. Overall, 9 of the 13 challenge videos are broken. Of the 4 videos not broken, two had only one word in their key frames. This suggests a significant level of correlation between text present in video key frames and the groundtruth tags that will pass video CAPTCHA.

TE_{recall} defines the percentage of the CAPTCHA's groundtruth tags that can be found in the video's key frames. It is expected to be a low value, as the groundtruth for a video also contains tags of related videos that are unlikely to be present within the key frames of the original. This is shown to be true by TE_{recall} having a mean of 17%. Attacks with IDs 1 and 3 did however retrieve the majority of the CAPTCHA's groundtruth tags from the key frames. $TE_{precision}$ indicates how indicative words present in the key frames are of tags that will pass the CAPTCHA. A mean value of 26% indicates that over 1/4 of such text is indeed indicative. Any means of extracting that text would therefore pose a threat to video CAPTCHA.

In theory, a challenge can be passed if even one groundtruth tag is extracted (i.e. $TE_{precision} > 0$). One or more groundtruth tags are shown as extracted in 11 of the 13 attacks. In attacks $\{5, 13\}$, values of $TS_{precision}$ and $TE_{precision}$ show attack failures caused by the tag selection process. Groundtruth tags exist in the extracted tag set, but fail to be extracted during tag selection.

Table 5.1: Results of performing manual attacks on video CAPTCHA, using the 13 videos in the sample set. An ID of μ signifies the mean; an ID of σ indicates the standard deviation.

ID	TE_{recall}	$TE_{precision}$	$TE_{fmeasure}$	$TS_{precision}$	Success
1	56.831	22.27	32.0	33.333	yes
2	2.963	100.0	5.755	100.0	yes
3	65.891	27.157	38.462	66.667	yes
4	25.641	22.727	24.096	100.0	yes
5	2.899	18.182	5.001	0.0	
6	42.945	26.923	33.097	66.667	yes
7	2.797	19.048	4.878	33.333	yes
8	6.944	27.027	11.049	33.333	yes
9	11.486	11.888	11.684	33.333	yes
10	1.55	50.0	3.007	66.667	yes
11	0.0	0.0	0.0	0.0	
12	0.0	0.0	0.0	0.0	
13	2.367	8.333	3.687	0.0	
μ	17.101	25.658	13.286	41.026	
σ	22.334	24.886	13.150	34.970	

Table 5.2: Results of performing automated attacks on video CAPTCHA, using the videos in the sample set. An ID of μ signifies the mean; an ID of σ indicates the standard deviation.

ID	TE_{recall}	$TE_{precision}$	$TE_{fmeasure}$	$TS_{precision}$	Success
1	16.94	3.144	5.304	0.0	
2	2.963	100.0	5.755	100.0	yes
3	0.0	0.0	0.0	0.0	
4	0.641	0.719	0.678	0.0	
5	0.0	0.0	0.0	0.0	
6	3.681	6.897	4.8	33.333	yes
7	1.399	18.182	2.598	0.0	
8	0.0	0.0	0.0	0.0	
9	1.351	3.39	1.932	0.0	
10	0.0	0.0	0.0	0.0	
11	0.0	0.0	0.0	0.0	
12	0.0	0.0	0.0	0.0	
13	0.0	0.0	0.0	0.0	
μ	2.075	10.179	1.621	10.256	
σ	4.451	26.389	2.168	27.377	

Results for *automated* attacks on videos in the sample set are shown in Table 5.2. Although only 2 of the 13 videos are passed (compared to 9 of 13 for the manual attacks), the results do indicate threat posed to video CAPTCHA. In 6 of the automated attacks, at

least one groundtruth tag is extracted from a video's key-frames and correctly classified by Tesseract. Of these 6 attacks however, only 2 (attacks $\{2, 6\}$) succeed in selecting the extracted tags that are relevant. Tag selection often selects tags that are severely mis-classified by Tesseract. These tags resemble random letters and symbols, and consequently escape filtering during answer set pre-processing steps (sanitation, stop word removal, frequency pruning, and length checking). If at least one extracted groundtruth tag is selected as an answer tag whenever possible in an attack, the success rate becomes 6/13. This suggests a bottleneck caused by the tag selection process.

TE_{recall} and $TE_{precision}$ for automated attacks show that problems exist in other areas of the attack process as well. Only 2% of groundtruth tags are extracted on average, compared to 17% for manual attacks. Of the extracted tags, an average of only 10% are groundtruth tags (compared to 25% in manual attacks). Main causes of this are the masking, segmentation, and OCR processes.

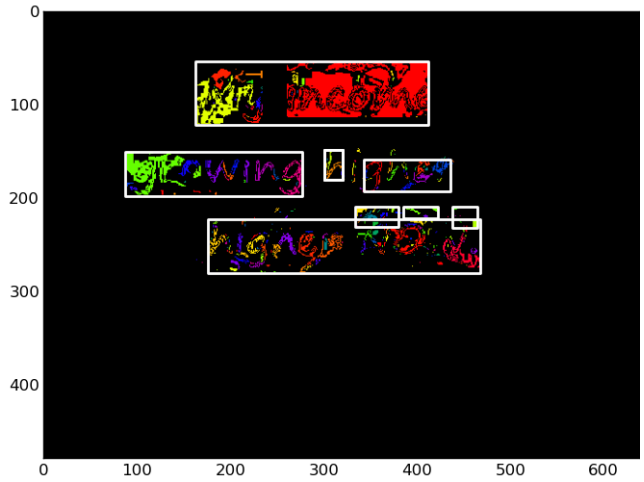


Figure 5.6: An example of the character masking process causing a failure in an automated attack. A noisy character-mask of video key frame is shown. The degree of noise in the mask causes errors in segmentation and OCR.

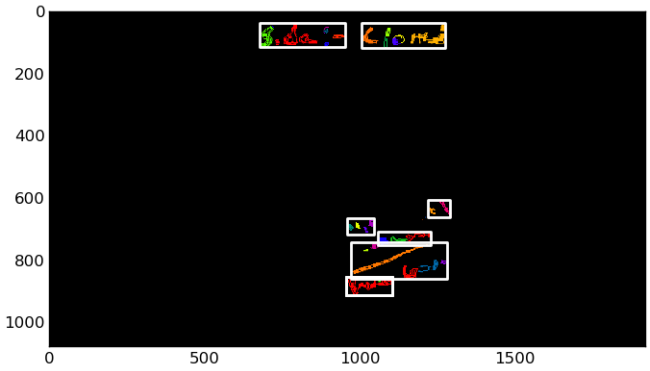


Figure 5.7: A key-frame character mask that includes non-character objects. These objects are OCR'd as garbage data.

Many of the masks were of extremely poor quality, as shown in Figure 5.6. Masks of key frames also frequently contain objects that do not represent text, as shown in Figure 5.7. Such masks confuse the segmenter and cause Tesseract to output garbage text; this accounts for a great deal of irrelevant, extracted tags.

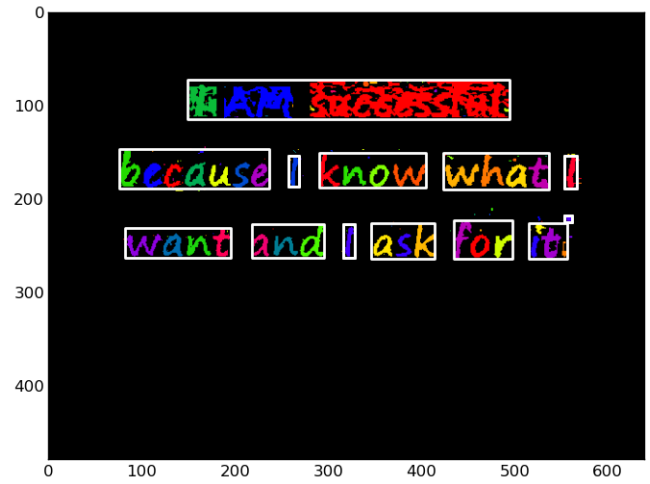


Figure 5.8: The bagged segmenter accurately segmenting characters of a video key-frame into words. Tesseract fails to classify these words, and outputs garbage data.

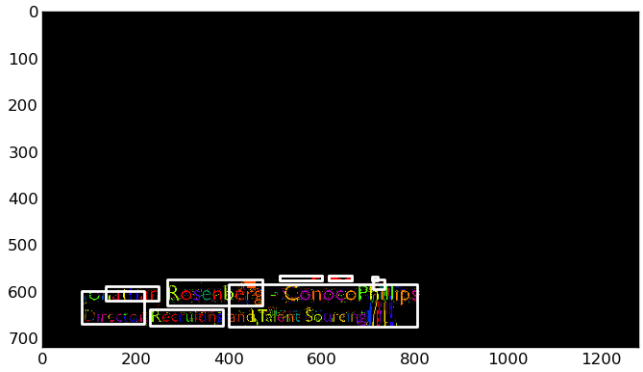


Figure 5.9: The bagged segmenter fails to segment a character mask of a video key-frame. Errors from text-line segmentation can be seen.

Although the segmenter performed well on many of the key-frame masks (as shown in Figure 5.8), it did not perform well on several masks. Both noise in masks and the often low quality of imagery in videos contribute to the errors. The segmenter is seen failing in Figure 5.9, beginning at the text-line level.

Tesseract is a significant source of errors. Garbage text is often given for good segmentations (such as the segmentation shown in Figure 5.8). With better image pre-processing and mask generation techniques, both Tesseract and the segmentation process are expected to perform significantly better.

Chapter 6

Conclusions

Words present in videos are indicative of tags that Video CAPTCHA will accept. A series of thirteen manual attacks against video CAPTCHA demonstrate that text present within challenge video key-frames are highly indicative of tags that will pass a video CAPTCHA. In 10 of the 13 attacks, at least one word in the video could be used to pass the challenge.

Tesseract is not able to accurately classify words within the key-frames of videos used by Video CAPTCHA, when given the predicted bounding-boxes of the words. Tesseract is proven highly unreliable at classifying noisy and/or natural text in video key-frames. However, at least one extracted groundtruth-tag is correctly OCR'd by Tesseract in 9 of the 13 challenge videos. While Tesseract should not be used to reliably classify natural text, it does provide the ability to launch successful OCR-based attacks on video CAPTCHA.

Development and evaluation of a natural text segmenter demonstrates that natural text can be extracted from a character-mask with about 76% reliability. A series of automated attacks show that text within key-frame masks can be similarly segmented, but with less reliability. Even though the automated attacks provide a low break rate, they demonstrate that tags that will pass video CAPTCHA can be extracted and OCR'd the vast majority of the time. The bottleneck in break rate is currently the tag selection process, which has trouble handling garbage and/or noisy text-data from OCR.

We conclude that OCR-based attacks do pose a moderate level of threat to Video CAPTCHA, and that such attacks should be considered during future development of Video CAPTCHAs. Future work should focus on improvements to character-mask generation and OCR. Improving mask generation will lead to less noise, and consequently improved

word segmentation and OCR. Text localization and applicable pre-processing techniques can likely create more effective masks than the manual approach taken in this work. Using text localization and canny edge detections, *OTCYMIST* [11] obtains an f-measure of 70% on the ICDAR 2011 dataset with an MST approach similar to the baseline. An improved tag selection approach is also needed to obtain better break rates. A method that ranks tags belonging to a dictionary as higher priority is recommended. Spellchecking tags should also be considered.

Bibliography

[1] M Blum, LA Von Ahn, J Langford, and N Hopper. The captcha project,completely automatic public turing test to tell computers and humans apart,. *School of Computer Science, Carnegie-Mellon University, http://www. captcha. net*, 2000.

[2] Elie Burszteom. How we broke the nucaptcha video scheme and what we propose to fix it. Feb 2012.

[3] JingSong Cui, LiJing Wang, JingTing Mei, Da Zhang, Xia Wang, Yang Peng, and WuZhou Zhang. Captcha design based on moving object recognition problem. In *Information Sciences and Interaction Sciences (ICIS), 2010 3rd International Conference on*, pages 158–162, 2010.

[4] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2963–2970. IEEE, 2010.

[5] Christer Ericson. *Real-time collision detection*. Morgan Kaufmann, 2004.

[6] NuCaptcha Inc. Nucaptcha ii most secure and usable captcha, 2011.

[7] D Karatzas, S Robles Mestre, J Mas, F Nourbakhsh, and P Pratim Roy. Icdar 2011 robust reading competition-challenge 1: reading text in born-digital images (web and email). In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1485–1490. IEEE, 2011.

[8] K.A. Kluever. Evaluating the usability and security of a video captcha. Master's thesis, Rochester Institute of Technology, 2008.

[9] Kurt Alfred Kluever and Richard Zanibbi. Balancing usability and security in a video captcha. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS '09, pages 14:1–14:11. ACM, 2009.

[10] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[11] Deepak Kumar and AG Ramakrishnan. Otcymist: Otsu-canny minimal spanning tree for born-digital images. In *Document Analysis Systems (DAS), 2012 10th IAPR International Workshop on*, pages 389–393. IEEE, 2012.

[12] I. Ludmila Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*, chapter 7.1, pages 203–207. Wiley-Interscience, 2004.

[13] Laurence Likforman-Sulem, Abderrazak Zahour, and Bruno Taconet. Text line segmentation of historical documents: a survey. *International Journal of Document Analysis and Recognition (IJDAR)*, 9(2-4):123–138, 2007.

[14] S.M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, R. Young, K. Ashida, H. Nagai, M. Okamoto, H. Yamamoto, et al. Icdar 2003 robust reading competitions: entries, results, and future directions. *International journal on document analysis and recognition*, 7(2):105–122, 2005.

[15] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[16] C. Patel, A. Patel, and D. Patel. Optical character recognition by open source OCR tool tesseract: A case study. *International Journal of Computer Applications*, 55(10), 2012.

[17] C. Pope and K. Kaur. Is it human or computer? Defending e-commerce with captchas. *IT professional*, 7(2):43–49, 2005.

[18] Martin F Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

[19] John Ross Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

[20] R. Smith. An overview of the tesseract OCR engine. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 629–633. IEEE, 2007.

[21] David K. Snyder. Text detection in natural scenes through weighted majority voting of DCT high pass filters, line removal, and color consistency filtering. M.s. thesis, Rochester Institute of Technology, College of Science, Center for Imaging Science, Rochester, New York, United States, May 2011.

[22] R. ur Rizwan et al. Survey on captcha systems. *Journal of Global Research in Computer Science*, 3(6):54–58, 2012.

[23] Richard Zanibbi, Harold Mouchere, and Christian Viard-Gaudin. Evaluating structural pattern recognition for handwritten math via primitive label graphs. In *IS&T/SPIE Electronic Imaging*, pages 865817–865817. International Society for Optics and Photonics, 2013.

Appendix A

Sample Set of CAPTCHA Challenge Videos

The table below describes the 13 videos used in the manual and automated attack experiments. Attack ID (*AID*) corresponds to the ID of the attack in Table 5.1 and Table 5.2 that is performed on the video. A video with a YouTube ID (*YID*) of $ytid$ can be directly accessed via the following web URL: http://www.youtube.com/watch?v=$ytid$

YID	AID	Video Title
5tntAwMpV8Q	4	WCMH NBC 4 News Highlights Federal Funding Cuts to Health Centers
csfa34CbBLg	1	Why are you excited about dot jobs?
gGzpGDbrfvk	3	Owl City ft. Carly Rae Jepsen - Good Time Lyrics
he5mZX1sRXk	11	Interactive game book
lG9PkUtj4SI	5	Coffe Table w/ sword maintenance compartments
obR9U4yDaAk	10	DONESBLOG
owxzRRhnGaU	8	Team WWE vs The Nexus Summerslam 2010 highlights - HD
Q7R6mq6TKig	6	Attract money with positive affirmations.. (law of attraction)
Tuwzq4EpQ44	12	How to draw a haunted house, step by step
vez2-ax6IRc	9	Near Charleston, SC Dealership - Finance 2013 Nissan Rogue
wmakOt65At4	2	Country - Empire Of The Sun
YtTOQSRMRcM	7	Truck Driving challenge part 1: Rig Stig & the power slide - Top Gear - BBC
yX1ltt6GAD8	13	Cut the Rope - Spooky Box - Level 12-14 - 3 stars walkthrough

Appendix B

Example Tag Extraction Results

The list below shows 5 challenge videos in the sample set, along with their corresponding manual and automated tag extraction results. Automatically extracted tags indicate segmentation and OCR performance. Various degrees of OCR performance are shown. Some videos result in clean OCR'd text, while others result in "garbage data" composed of seemingly random characters.

YouTube ID: wmakOt65At4

Attack ID: 2

Manually Extracted Tags: Empire of the Sun

Automatically Extracted Tags: EMPIRE OF THE SUN

YouTube ID: yX1ltt6GAD8

Attack ID: 13

Manually Extracted Tags: menu level 12 - 14 menu menu menu menu menu menu menu menu menu menu menu menu menu menu menu excellent! replay next menu excellent! replay next menu excellent! replay next menu excellent! replay next menu star bonus 2616 383 time 0:16 3264 your final score 3720 improved result

Automatically Extracted Tags: 2/] Excel:-{I

YouTube ID: he5mZX1sRXk

Attack ID: 11

Manually Extracted Tags: powered by total immersion

Automatically Extracted Tags: .353

YouTube ID: lG9PkUtj4SI

Attack ID: 5

Manually Extracted Tags: top: open side ** 3 1/2 side:- closed h20 collection m:3 1/2 w:2 1/2 T-T 2 table top design open

Automatically Extracted Tags: rt; -3% . ’ X); 3-ez Tog. Oh” -3% . ’ X); -3% . ’ X); -3% . ’ X); -3% . // 3: ’ X); -3% // $ d_:- rJ Cfo /’ X); -3% . // ’ X); -3% . // /’ X); .,.,-y 3-mf-Lia!”-A: nal A nal nal In-”3’ nal nal nal LI LI H2] .1-is LI 09 .1-is LI {0 5; .1-is VI_’}’-. . :,_ VI_’}’-. . :,_ ..n”””’_.v-n –=rr-’ –=rr-’ of-/’ ”...II,1$

YouTube ID: owxzRRhnGaU

Attack ID: 8

Manually Extracted Tags: N N N N N N N W summer slam the nexus u me never give up r-truth live chris jericho john morrison bret ”the hitman” hart enation lover WWEhighlightss hd ww slim jim slam.c wwe slam.com

Automatically Extracted Tags: WWEhIgh—Ightss g 5.WlMMFD_=kr* (94 $? ”1’EK5) :L** Eljfgii W W WWEhighIightss . - . . . ,f //: . ..-””79 Eln): Vi//’z:,’/Y2: ; I g. ; -. I g. I g. uS1,1V1N.tl*, WWEhughIughlss C C &lMMFD-= J55; (4: If