International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-6, September 2015
ISSN: 2395-3470
www.ijseas.com

Breaking a Visual CAPTCHA

Dr. Atul Khurana
Computer Science & Engineering Department
Aryabhatta Group of Institutes
dr.atulkhurana@gmail.com

Abstract

In this paper we explore A CAPTCHA is a type of challenge-response test used in computing to determine whether the user is human. "CAPTCHA" is a contrived acronym for "Completely Automated Public Turing test to tell Computers and Humans Apart", trademarked by Carnegie Mellon University. A CAPTCHA involves one computer (a server) which asks a user to complete a test. While the computer is able to generate and grade the test, it is not able to solve the test on its own. Because computers are unable to solve the CAPTCHA, any user entering a correct solution is presumed to be human. The term CAPTCHA was coined in 2000 by Luis von Ahn, Manuel Blum, Nicholas J. Hopper (all of Carnegie Mellon University), and John Langford (then of IBM). A common type of CAPTCHA requires that the user type the letters of a distorted image, sometimes with the addition of an obscured sequence of letters or digits that appears on the screen.

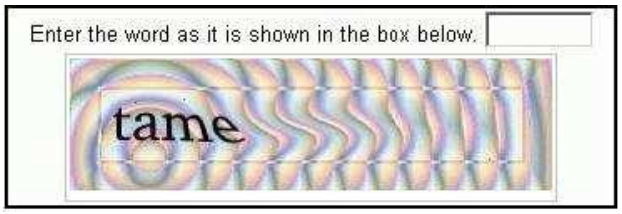


Figure 1.1: A CAPTCHA in use at Flipcart

1. Introduction

A CAPTCHA is a program [19] that can generate and grade tests that:

1. Most humans can pass, BUT

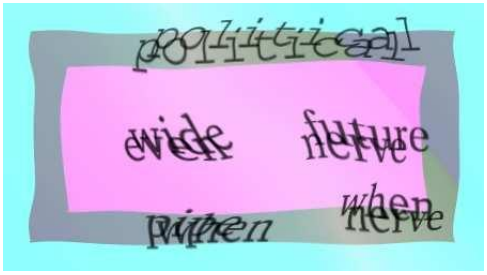


Figure 2.1: A CAPTCHA in use at Amazon

2. Current computer programs can't pass

CAPTCHA stands for “Completely Automated Public Tur-ing test to Tell Computers and Humans Apart”.

The concept behind such a program arose from real world problems faced by internet companies such as Yahoo and AltaVista. Yahoo offers free email accounts. The in-tended users are humans, but Yahoo discovered that various web pornography companies and others were using “bots” to sign up for thousands of email accounts every minute from which they could send out junk mail. The solution was to require that a user solve a CAPTCHA test before they re-ceive an account. The program picks a word from a dictio-nary, and produces a distorted and noisy image of the word (see Fig. 1). The user is presented the image and is asked to type in the word that appears in the image. Given the type of deformations used, most humans succeed at this test, while current programs (including OCR programs) fail the test. The goal of screening out the “bots” has been achieved.

Manuel Blum's group at CMU have designed a number of different CAPTCHAs. We strongly recommend visit-ing *http://www.captcha.net* to view some examples of these. Gimpy is based on word recognition in the presence of clut-ter. The task is to identify 3 of the approximately 7 words in a cluttered image. The Yahoo test is an easier version

of this, the so-called EZ-Gimpy, which involves only a sin-gle word. Pix is based on image recognition of typically more abstract categories. Chew and Baird [5] also devel-oped a test involving heavily degraded fonts. CAPTCHAs have also been designed based on auditory recognition [4]. For visually impaired humans, these may be the most ap-propriate tests.

The underlying principle behind the design of CAPTCHAs is a reduction to a hard AI problem. In the words of Ahn, Blum and Langford [19]: *Any program that passes the tests generated by a CAPTCHA can be used to solve a hard unsolved AI problem.* Their hope is thus to provide challenge problems for AI researchers. If the problem cannot be solved by computer programs, it can be used as a CAPTCHA and provide practical help to companies like Yahoo, AltaVista etc. If it can be solved, that marks scientific progress on a hard AI problem.

We, as vision researchers, decided to take up the challenge of defeating EZ-Gimpy and Gimpy. This paper describes an algorithm which achieves this goal. On EZ-Gimpy our program correctly identifies the word 92% of the time, which implies that Yahoo can no longer use it to screen out "bots". Our success rate of 33% on Gimpy also renders it ineffective as such a screening tool.

These two datasets provide more than just a colourful toy problem to work on. In this paper we explore two major aspects of object recognition: difficult clutter, and the trade-offs between recognition based on parts versus the whole object. The CAPTCHAs we use present challenging clut-ter since they are designed to be difficult for computer pro-grams to handle. Recognition of words lends itself easily to being approached either as recognition by parts (individual letters or bigrams) or whole objects (entire words).

More importantly, these datasets are large. There are about 600 words that need to be recognized. Also, since the source code for generating these CAPTCHAs is available ("P" for *public*), we have access to a practically infinite set of test images with which to work. This is in stark contrast to many object recognition datasets in which the number of objects is limited, and it is difficult to generate many rea-sonable test images.

There are definitely limitations to this dataset in terms of studying general object recognition. Most notably, these are 2D objects and there is no variation due to 3D pose. In ad-dition, there are no shading and lighting effects in synthetic images of words.

We believe that the ability to do quantitative experiments using a large set of test images with difficult clut-ter outweighs these drawbacks. We hope that other researchers will attempt their techniques on these and other CAPTCHAs.

Many computer vision researchers have worked on the problem of object recognition. Of these approaches, the most plausible candidates for success in this domain are Le-Cun et al. [11] and Amit et al. [1]. LeCun et al. use convo-lutional neural networks to perform handwritten character recognition. Amit et al. use point features combined with graphs to match various deformable objects. These tech-niques are somewhat robust to clutter, but it is not obvious that they could deal the kind of "adversarial" clutter found in Gimpy images.

The structure of the paper is as follows. In section 2 we describe our general purpose matching framework. In section 3 we explore the tradeoffs in parts-based and holistic matching. We describe two algorithms based on this frame-work in sections 4 and 5, and apply them to EZ-Gimpy and Gimpy. We conclude in section 6.

2. Matching using Shape Contexts

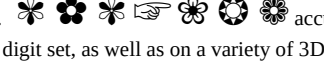
We are given a database of images of known objects. Our task is to find instances of these objects in a cluttered environment. In the case of Gimpy images, these objects are words or letters, possibly in a variety of fonts.

In our work, we will compare objects as shapes – each represented by a discrete set of points  sampled from the internal and external contours of the shape. Belongie et al. [2, 3] introduced the *shape context* descriptor based on such a representation and used it in a deformable template approach to match handwritten digits and 3D objects. Consider the set of vectors originating from a point to all other sample points on a shape. These vectors express the configuration of the entire shape relative to the reference point. Obviously, this set of  vectors is a rich description, since as gets large, the representation of the shape becomes exact.

The full set of vectors as a shape descriptor is much too detailed since shapes and their sampled representation may vary from one instance to another in a category. The *distri-bution* over relative positions is a more robust and compact, yet highly discriminative descriptor. For a point  on the shape, compute a coarse histogram  of the relative coor-dinates of the remaining  points,

This histogram is defined to be the *shape context* of  . The descriptor should be more sensitive to differences in nearby pixels, which suggests the use of a log-polar coordinate sys-tem. An example is shown in Fig. 3(c). A related approach, developed for 3D data, is the *spin images* technique of John-son and Hebert [9].

2.1. Matching Framework

The work by Belongie et al. [3] resulted in extremely good performance, e.g.  accuracy on the MNIST handwritten digit set, as well as on a variety of 3D object recogni-

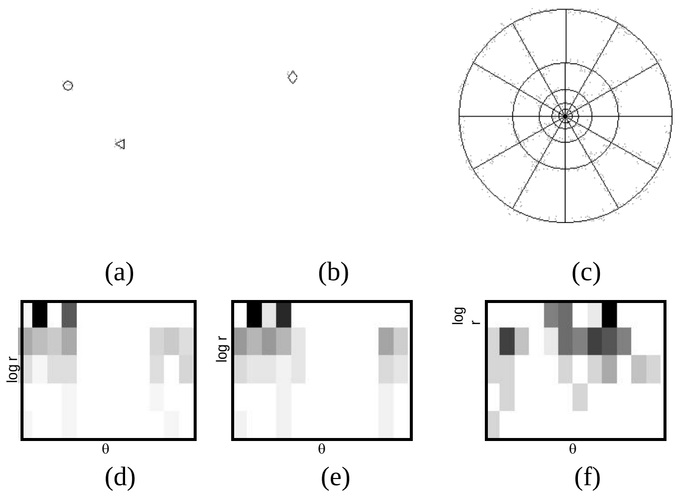


Figure 3: Shape contexts. (a,b) Sampled edge points of two shapes. (c) Diagram of log-polar histogram bins used in computing the shape contexts. (d-f) Example shape con-texts for reference samples marked by ✡ ✂ ✃ ✑ ✡ in ☎ ✄ (a,b). Each shape context is a log-polar histogram of the coordinates of the rest of the point set measured using the reference point as the origin. (Dark=large value.) Note the visual similarity of the shape contexts for and ✃, which were computed for relatively similar points on the two shapes. By contrast, the shape context for ✄ is quite different.

tion problems. However, applying this deformable matching algorithm to a large database of models would be computationally prohibitive. To deal with this problem, Mori et al. [14] described a two stage approach to object recogni-tion, namely:

1. *Fast pruning:* Given a query image, we should be able to quickly retrieve a small set of likely candidate shape and location pairs from a potentially very large collec-tion of stored shapes.

2. *Detailed matching:* Once we have a small set of candidate shapes, we can perform a more expensive and more accurate matching procedure to find the best matching shapes to the query image.

In this paper we use a matching framework of this style. In the following sections we first describe a new descriptor that is an extension of shape contexts. We then provide the details for the two stages in our recognition framework, fast pruning and detailed matching.

2.2. Generalized Shape Contexts

We have extended the shape context descriptor by encoding more descriptive information than point counts in the histogram bins. In this work, to each edge point ✻ ✕ we attach a unit length tangent vector ✆ ✕ that is the direction of the edge at ✻ ✕. In each bin we sum the tangent vectors for all points falling in the bin.

✦ † ✕ ✩ ✫ ✪ ✯ ✂ ✝ ✞ ✆ ✕ ✡ where ✏ ✂ ☎ ✲ ✳ ✂ † ✔ ✕ ✞ ✡ ✍ ✩ ✸ ✲ ✤ † ➾ ✕ ✯ ✺ ✖ bin ✩ ✫ ✪ ✯ ✲ ✓

✠ ☛ ✡ ✌ ☞ ✎ ✍

Each bin now holds a single vector in the direction of the dominant orientation of edges in the bin. We compare them using a distance similar to the ✑ ✚ distance in [3].

➾ ✩ † ✕ ✡ † ✯ ✂ ✞ $\frac{\text{† ✕ ✩ ✫ ✪ ✯ † ✩ ✫ ✪ ✦}}{\text{✖ ✦ ✜ ✦ ✖}}$ ✦ ✦ ✔ ✓ bins

We call these extended descriptors ✕ *generalized shape contexts*.

2.3. Fast Pruning

The goal of the fast pruning stage is to construct a small set of hypotheses of objects at various locations in the query image. We use tests based on *representative shape contexts* to accomplish this goal.

The matching process proceeds in the following manner. For each of the known shapes ✕, we precompute a large number (about 100) of shape contexts ☎ ✕ ✓ ✶ ✔ ✤ ✂ ✣ ✡ ✦ ✤ ✡ ✌ ☞ ✍ ☞ ✌ ☞ ✓ .But for the query image ✢, we only ✢ ✣ ✢ compute a small number ✤ of representative shape contexts. To compute these ✤ shape contexts we randomly select ✏ sample points from the image. We then do comparisons with each ✧ of the known shapes using only these shape contexts in a voting scheme.

In these cluttered images, many of the shape contexts contain noisy data, or are not located on the shape ✕. Hence, for each of the known shapes ✕ we find the best ✢ ✪ representative shape contexts, the ones with the smallest distances. Call this set of indices X_1 ✢. The distance between ✢ and ✕ is then: ★

✏ ✚

✩ ✡ ✕ ✯ ✯ ✂ ✒ ✢ ✣ ✢ ✍ ✢ ✣ ✢ ✣ ✯ ✯ ✯ ➾ ✏ ✢ ✫ ✞ where ✩ ✯ ✯ ✯ ✯ ✂ ✩ ✫ ☞ ✎ ✯ ✯ ✯ ✡ ➾ ✩ ✻ ✩ ✡ ✕ ✓ ✯ ✍

✳ ✻ is a normalizing factor that measures how discriminative the representative shape context ✩ is:

✩

$$\frac{\text{✂ ✢}}{\text{✱ ✩ ✞ ✱ ✯ ✿ ✮}} \quad ✩ ✩ ✡ \quad ✢ ✣ ✢ ✍ ✢ ✣ ✢ ✯ ✯ ✯$$

❀ ✡ ✌ ☞ ✫ ❁

[1] The same shape can appear many times in one query image. Therefore, there can be many sets per model. We group nearby representative shape contexts to obtain these sets.

✫ ❄ ❅ ❆ ✫ ❇

3

where is the set of all shapes.

We also obtain an estimate of the position of the shape . If representative shape context at point in the image best matches point on shape , it "votes" for shape at location , with weight is:

The end result of our fastpruning procedure is a set of tuples. We can threshold this set based on distance to obtain a small set of candidates.

Voting ideas have been used before in object recognition (e.g. Lamdan et al. [10]). Our method is different from ones such as geometric hashing in that it uses soft weighting of votes and locations instead of making hard decisions.

2.4. Detailed Matching

The detailed matching stage uses a deformable template ap-proach. In such an approach, a query shape is compared to a candidate model shape by attempting to deform the model into alignment with the query shape. We implement this using iterations of correspondence through shape con-text matching and deformation using thin plate splines. A cost for performing this deformation is computed and used along with the shape context distance as the matching cost between the model and the query. More details can be found in [3].

3. Approaches to Recognition

There are two important types of data available to us in solving word recognition tasks – lexical information and visual cues. The lexical information could be the set of words, the set of bigrams, or the set of 26 letters, depending upon the scope of one's view. The visual cues are the grayscale patches, edges and other features in the image. In order to design a recognition algorithm, one must exploit the information present in these two cues. In this work we study the role of each, and the tradeoffs that exist in using them.

In the field of handwriting recognition, the different levels of lexical information are well studied. Some approaches, such as [12], take a holistic approach, recognizing entire words, while others such as [15] focus more on rec-ognizing individual characters. Plamondon and Srihari [16] provide an extensive survey of this work.

Holistic approaches incur more computational cost since there are more models, but have more expressive and discriminative power since the visual cues are gathered over large areas. The more efficient, essentially parts-based algorithms are faster, but can have difficulty in clutter when the visual cues of the parts are corrupted.

In the following sections we will develop two algorithms. The first takes a bottom-up, parts-based approach, finding characters first, and using the lexical information to decide which characters can be formed into words at a later stage. The second algorithm uses a holistic approach and tries to find words immediately. We devise pruning meth-ods to deal with the associated efficiency problems.

In our experiments we will use algorithm A to break EZ-Gimpy, and algorithm B for EZ-Gimpy and Gimpy. We will show how the clutter in Gimpy corrupts the visual cues to the degree that individual letters are no longer discrim-inable. In such scenarios a holistic approach is desirable. However, with good visual cues as in EZ-Gimpy, an effi-cient parts-based approach has its merits.

4. Algorithm A

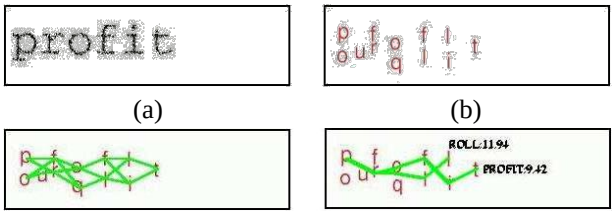
Our first algorithm for finding words in images works from the bottom-up, starting with visual cues and incorporates lexical information later on. It consists of three steps:

1. Perform a series of quick tests to hypothesize locations of letters in the image
2. Extract strings of these hypothesized letters that form candidate words
3. Choose the most likely word(s) by evaluating a match-ing score for each of these words

An example of processing an image using the three stages of our method is shown in Fig. 4.

The general philosophy that we follow is the use of a coarse to fine matching strategy for computational efficiency. We start with quick, approximate tests that prune away much of the search space. These tests may have high false positive rates, but should reject very few correct hypotheses. We work our way down to using more computationally expensive, but more accurate, tests for the final comparison that determines which word is present. However, we need only apply these expensive tests to a small number of remaining hypotheses. Previous work by Geman and Jedynak [6] and Viola and Jones [18] follow a similar vein of thought.

Specifically, the first step in our method prunes a large space of letter locations (26 letters that could occur any-where in the image) down to about 100 hypothesized tuples. The second step then analyzes this set of tuples to produce 1 to 10 actual words that are found in the dictionary. Finally, the expensive task of assigning a score to a word is performed on this small set. In the following sections we will provide the details of each step in our algorithm.



(a) (b) (c) (d)

Figure 4: The 3 steps in algorithm A demonstrated on a simple example: (a) Gimpy image (b) Locations of hypothesized letter locations (c) DAG of possible strings of letters (d) Scores of top matching words (shown on paths through DAG). The word "profit" is found as the best matching word.

4.1. Finding Letter Hypotheses

The first step in this method is to hypothesize a list of ✆✡✠✡☛✆✌✆☛✡✠☞✌✎✍✶✆✌✞ tuples. This stage must be fast – at each location ✧ in the ✻ image any one of the 26 letters could be present.

We start with a training set of 26 images, each an unclut-tered image of a single letter. We run Canny edge detection on these images and construct a generalized shape context for each edge point in each training image. This gives us a pre-computed database of approximately 2600 generalized shape contexts.

We employ the voting scheme, based on the representative shape contexts technique in [14], described in sec-tion 2.3. First, we sample ☜✣ the number of Canny edge points[2] in the image at which we compute representative shape contexts. This number is typically 30 to 50.

The process outlined above gives us a set of ✆✡✠☞✡☛✆✌✆☛✡✠☞✌✎✍✶✆✌✞✡✍✌☛✞ tuples. We threshold based on the scores to obtain a final set of ✆✡✠☞☛✆✌✆☛✡✠✡✌✎✍✶✆✌✞ hypotheses ✧ ✻ ✣ ✧ ✧ ✻ (5 to 30 total).

4.2. Extracting Candidate Words

At this point we have a set of ✆☞✠✡☛✆✌✆☛✡✠☞✌✎✍✶✆✌✞ hypothe-ses strewn about the image. There tend ✧ to ✻ be clusters of hypotheses around real letters in the image, as shown in Fig. 4(b). The next step is to find sequences of these let-ters that form candidate words. To this end we construct a directed acyclic graph (DAG) in which there is a node for each letter hypothesis ✠✕, and an edge ☛✂ ☆✠✕✡✠ ✯ between

[2]Note that in images containing background texture, the ✓ Canny edge detector will fire nearly everywhere. We need a way of choosing representative shape contexts that are likely to be on or near letters, instead of being "wasted" on the background. We sample our representative shape contexts near high values of the *texture gradient* [13] operator. This operator is designed to measure texture differences and therefore in the interior of any homogeneous texture, it will have a small response.

two nodes if letter hypothesis ✠ can be used as the letter succeeding ✠✕ in a word. We compute ✓ this consistency of a pair of letters by considering their bounding boxes (ob-tained from training images of letters). First, letter ✠✕ must be at a location to the left of letter ✠, since all words are read left to right. Second, the bounding ✓ boxes should nei-ther overlap too much nor be too far away from each other. This enforces spatial continuity within words. Each path in this DAG represents one possible word.

In a typical Gimpy image there can easily be ✣ to ✣ paths through this DAG. However, most of these paths ✂✂ will ☎✄ be nonsense, strings of letters such as "ghxr". Again we wish to do some quick pruning, in order to reduce the num-ber of paths through this DAG. We use tri-grams for this pruning. A tri-gram is a sequence of 3 letters, hence there are ✤✝✆✞✟ possible tri-grams. An -letter word has ✠✣ tri-grams. The probability of all ✠✣ trigrams for a word appearing by chance in our DAG is very low. We use this fact to devise our pruning method.

We precompute all tri-grams of all words in the dictionary (EZ-Gimpy uses 561 words). After obtaining the DAG of letter hypotheses for our test image, we can quickly com-pute all the tri-grams (usually around 1000) present in it. Then we prune – a dictionary word cannot be present in the image unless all of its tri-grams are present. Finally, for the small number of remaining words, we check that each of them is actually found as a path through the DAG, since the ✠✣ ✣ tri-grams might not actually occur in sequence. All words that pass this final test are then listed in the final set of candidate words. Note that our technique is only one ex-ample strategy for string matching. This is a well studied subject [7].

4.3. Choosing the Most Likely Word

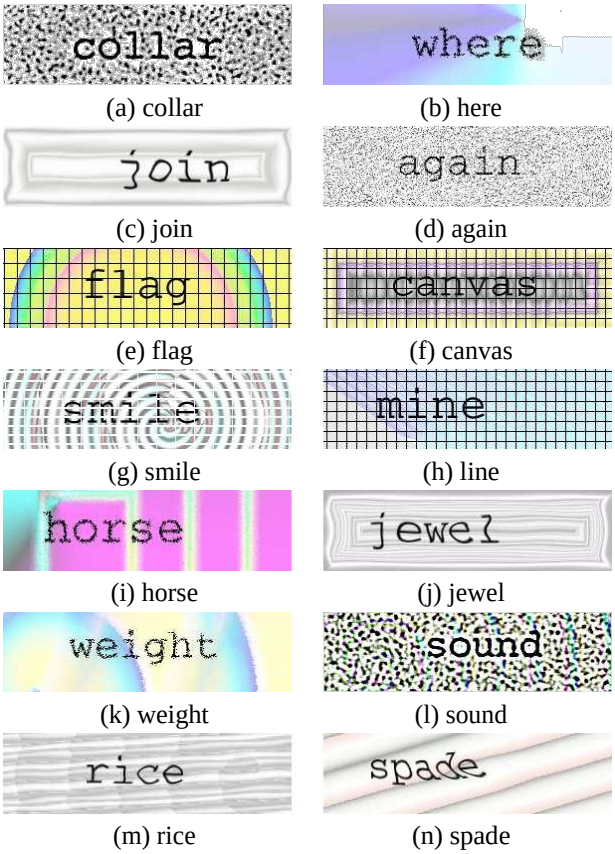
We now have a set of candidate words and wish to choose the most likely one to use as our answer to the Gimpy test. This final step is the most computationally intensive. How-ever, we are left with only a few words which need to be evaluated.

The score we use for ranking the set of candidate words is based on deformable matching cost of individual letters, similar to that used by Belongie et al. [3] used for matching handwritten characters. Scores are obtained through matching costs of generalized shape contexts. The score for a word is the average score for matching each of its letters.

This is the final step. We compute matching scores for each of our candidate words. Our answer to the Gimpy test is the word with the best matching score.

4.4. Results

An EZ-Gimpy CAPTCHA consists of a single deformed word in a cluttered image. See Fig. 5 for examples of typi-



(a) collar (b) here (c) join (d) again (e) flag (f) canvas (g) smile (h) line (i) horse (j) jewel (k) weight (l) sound (m) rice (n) spade

Figure 5: Examples of results on EZ-Gimpy images. The best matching word is shown below each image. Examples (a),(c-g),(i-n) produced correct results, (b) and (h) incorrect results.

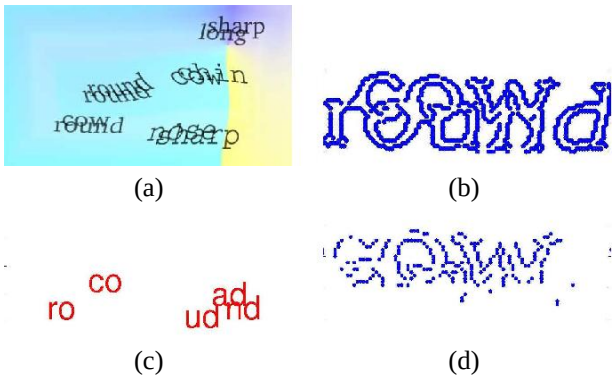
cal EZ-Gimpy images. We used algorithm A to quickly find the word present in these images.

For our experiments, we generated 200 examples of the EZ-Gimpy CAPTCHA using the code available at *http://www.captcha.net*. Of these examples, 9 were used for tuning parameters in the bounding box similarity computation and texture gradient modules. The remaining 191 examples were used as a test set. In 158 of these 191 test cases, the top matching word from our method was the cor-rect one, a success rate of 83%.

Note that it is also possible to use a soft probabilistic strategy involving HMMs choose the most likely word from the DAG, instead of making hard pruning decisions.

Some examples of the 191 EZ-Gimpy CAPTCHA im-ages used, and the top matching words are shown in Fig. 5. The full set of 191 test images and results can be viewed online at *http://www.cs.berkeley.edu/ mori/gimpy/ez/*.

Figure 6: Examples of letter sized patches from Gimpy CAPTCHAs. It is very difficult to read these isolated letters without the long range context in which they appear. We must instead read whole words at once.



(a) (b) (c) (d)

Figure 7: Processing a Gimpy image using algorithm B. (a) Input image. We process each pair of words separately. (b) Edge detection output on a single pair of words. (c) Hypothesized bigrams. (d) Pixels remaining after guess of word “round”. Multiple iterations of step (d) – guess a word, remove pixels, guess remaining word are performed.

5. Algorithm B

The second algorithm is a holistic one that attempts to find entire words at once, instead of looking for letters. In severe clutter, many of the parts can be occluded or highly ambiguous. In the case of Gimpy, letters are often not enough. Figure 6 shows some example letter-sized patches from Gimpy images. It is nearly impossible to determine which character(s) is present in these patches. However, when one sees the entire word, it is clear which word is displayed. This is the motivation for a holistic approach to recognition.

We construct shape contexts that are elliptical in shape, with an outer radius of about 4 characters horizontally, and about ✂ of a character vertically. These are our features for doing the recognition at a word-level. However, a brute-force approach, using representative shape contexts to compare to every word in the dictionary, is infeasible because of computational cost. Instead, we do some pruning to reduce our set of words down to a manageable size, and then use this holistic approach to match complete words. Figure 7 shows the steps in processing an image using this algorithm.

5.1. Pruning with Bigrams

Figure 9 shows some examples of Gimpy CAPTCHAs. Each image contains 10 words (some duplicates) overlaid in pairs, with some background clutter. We examine each pair of words separately. The task is to correctly recognize any 3 of these words. In these images, the most salient parts of the words are often the beginning or ending few letters. Moreover, given the opening or closing bigram, the number of possible candidate words is very small. The pruning we do is based on these observations. In the 411 word gimpy dictionary there are 128 distinct opening and 112 closing bigrams. If we can accurately determine these bigrams we will be left with a small number of candidate words.

This bigram pruning is done using the fast pruning method outlined above. In our experiments on Gimpy, we truncate the list of bigrams (either beginning or ending), or-dered by shape context distance, such that we are left with a shortlist of about 20 candidate words.

Some holistic matching can be done here to reduce the size of the shortlist. In our experiments we sorted these can-didate words using the word-sized shape contexts as fea-tures in the fast pruning process, to be left with about 4 words.

5.2. Layers of Words

In a Gimpy CAPTCHA, the words appear in pairs. Given a guess of one of the words, we can try to remove it, and recognize the second word based on the remaining pixels. The removal of pixels is done with an assignment problem. We match edge pixels on our guess of the first word with the edges in the image. We remove the edges in the image that were used in the matching, and then repeat our bigram pruning to obtain a new shortlist of possible second words. This type of analysis is similar to work on transparency such as [8, 17].

5.3. Final Score

After the layers analysis we are left with 16 pairs of words (4 guesses for the first word, each with its own 4 guesses for the second word). The final step is to assign scores to each of these words.

For each pair, we produce a synthetic image of the two words overlaid at their estimated locations (given by the pruning method). We then compute shape contexts in the synthetic image. The final score for a word is the represen-tative shape contexts pruning cost using a sizable fraction (.3 in experiments) of the edge pixels from the word as the value for ✪, the number of edge points to match. Most of the edges in the center of the word are corrupted beyond recog-nition, hence it isn't useful to use them in the final scoring.

The two words in each pair are scored separately, we select the best ✪✂ edge points from each word. In the Gimpy CAPTCHA, the task is to recognize 3 words. We choose the 3 words with the best score as our answer.

5.4. Results

We tested algorithm B on 24 instances of the Gimpy CAPTCHA. In order to pass, the program must guess 3 words, all of which are in the image. Our results are shown in Fig. 8 and 9. We successfully guess 3 words from the image 33% of the time. With our 33% accuracy, this CAPTCHA would be ineffective in applications such as screening out “bots” since a computer could flood the application with thousands of requests.

# Correct Words	Percentage of Tests
✣	92%
✣	75%
	33%

Figure 8: Results on Gimpy. The task is to guess 3 words in the image. We pass the test 33% of the time. However, one or two words guessed are correct very frequently.

The sharp decrease in accuracy in Fig. 8 is not unex-pected. If we assume that the accuracy of each of the 3 words is independent, and a single word accuracy of 70%, we would only expect to pass the test ☞✆✂✂ ☞✄❀ of the time.

We also applied algorithm B (searching for entire words, without the layers analysis) to the same 191 instances of EZ-Gimpy as algorithm A. We found the correct word in 176 of these, for a success rate of 92%.

6. Conclusion

In this paper we have explored methods for performing ob-ject recognition in clutter. We have explored the tradeoffs between using high level lexical information, in our case the dictionary of words, to guide recognition versus relying on low level cues. We tested our algorithms on two word-based CAPTCHAs that allow us to do experiments with many test images.

Clutter that involves other real objects, like the words in the Gimpy images, makes recognition much more difficult than the heavily textured backgrounds of EZ-Gimpy. Our algorithms were able to effectively deal with both of these types of clutter and break these two CAPTCHAs with a high frequency of success.

The problem of identifying words in such severe clutter provides valuable insight into the more general problem of object recognition in scenes. Parts are often ambiguous in clutter, but at the same time anchoring around obvious parts can improve efficiency. We have presented algorithms that

dry,clear,medical | door,farm,important
public,nose,receipt | with,true,sponge
carriage,potato,clock | church,tongue,bad
narrow,bulb,right | sudden,apple,oven

Figure 9: Results on Gimpy. The 3 words that are guessed using Algorithm B are shown below each image.

are instances of a general framework that can be applied to other recognition problems.

References

[1] Y. Amit and A. Kong. Graphical templates for model registration. *IEEE Trans. PAMI*, 1996.

[2] S. Belongie, J. Malik, and J. Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *NIPS*, November 2000.

[3] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. PAMI*, 24(4):509–522, April 2002.

[4] N. Chan. BYAN: Sound Oriented CAPTCHA, website http://drive.to/research.

[5] M. Chew and H. S. Baird. Baffletext: A human interac-tive proof. In *Proceedings of SPIE-IS&T Electronic Imag-ing, Document Recognition and Retrieval X*, pages 305–316, January 2003.

[6] D. Geman and B. Jedynak. Model-based classification trees. *IEEE Trans. Info. Theory*, 47(3), 2001.

[7] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Press Syndicate of the University of Cambridge, 1997.

[8] A. Jepson, D. Fleet, and M. Black. A layered motion representation with occlusion and compact spatial support. *European Conference on Computer Vision*, 1:692–706, 2002.

[9] A. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Trans. PAMI*, 21(5), May 1999.

[10] Y. Lamdan, J. Schwartz, and H. Wolfson. Affine invariant model-based object recognition. *IEEE Trans. Robotics and Automation*, 6:578–589, 1990.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[12] S. Madhvanath and V. Govindaraju. The role of holistic paradigms in handwritten word recognition. *IEEE Trans. PAMI*, 23(2), February 2001.

[13] D. Martin, C. Fowlkes, and J. Malik. Learning to find brightness and texture boundaries in natural images. *NIPS*, 2002.

[14] G. Mori, S. Belongie, and J. Malik. Shape contexts enable efficient retrieval of similar shapes. In *CVPR*, volume 1, pages 723–730, 2001.

[15] J. Pitrelli, J. Subrahmonia, and M. B. Toward island-of-reliability-driven very-large-vocabulary on-line handwriting recognition using character confidence scoring. *ICASSP*, 2001.

[16] R. Plamondon and S. S. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Recognition*, 22(1):63–84, January 2000.

[17] R. Szeliski, S. Avidan, and P. Anandan. Layer extrac-tion from multiple images containing reflections and transparency. *IEEE Computer Vision and Pattern Recognition*, 1:246–253, 2000.

[18] P. Viola and M. Jones. Robust real-time object detection. *2nd Intl. Workshop on Statistical and Computational Theories of Vision*, 2001.

[19] L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart (automatically). *CMU Tech Report CMU-CS-02-117*, February 2002.