Breaking the PayPal HIP: A Comparison of Classifiers

Kurt Alfred Kluever
Document and Pattern Recognition Lab
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623 USA
kurt@kloover.com

May 20, 2008

Abstract

Human Interactive Proofs (HIPs) are a method used to differentiate between humans and machines on the internet. Providers of online services such as PayPal.com use HIPs to prevent automated signups and abuse of their services. In this experiment, a three step algorithm has been developed to break the PayPal.com HIP. The image is *preprocessed* to remove noise using thresholding and a simple cleaning technique, and then *segmented* using vertical projections and candidate split positions. Four *classification* methods have been implemented: pixel counting, vertical projections, horizontal projections and template correlations. The system was trained on a sample of twenty PayPal.com HIPs to create thirty-six training templates (one for each character: 0-9 and A-Z). A sample of 100 PayPal.com HIPs were used for testing. The following HIP success rates have been achieved using the different classifiers: 8% pixel counting, vertical projections 97%, horizontal projections 100%, template correlations 100%. Three of the classifiers out perform the 88% HIP success rate of [6].

1 Introduction

Human Interactive Proofs (HIPs) are a method used to automatically differentiate between humans and machines on the internet. HIPs should be easy for a machine to automatically generate, easy for a human to solve, and difficult, or impossible, for a machine to solve. They are typically implemented as an image of distorted text which the user must correctly transcribe. However, nearly all of the text recognition based HIPs are insecure against attacks based on neural networks [3], shape matching [11], and simple pattern recognition [17]. In this experiment, the PayPal.com HIP has been successfully broken using the conventional OCR process: *pre-processing, segment, classify.* Four classifiers have been implemented and the results are compared using the following metrics: *HIP accuracy*, *character accuracy*, *confidence*, and *running time*. The classifiers are based on the following methods: pixel counting, vertical projections, horizontal projections, and template correlations. Section 2 provides a background on OCR and HIPs. The experiment's methodology is explained in Section 3. Section 4 details the training stage. Section 5 explains and compares the results of the system. Section 6 summarizes the approaches and the results of the work presented.

2 Background [1]

2.1 OCR

Optical Character Recognition (OCR) is the process of translating images of handwritten, typewritten, or printed text into a format understood by machines for the purpose of editing, indexing, searching, or compression [14]. The OCR process can be broken down into three tasks: pre-processing, segmentation, and classification.

2.1.1 Pre-processing

Pre-processing is necessary if the input image is noisy due to old paper, poor printers, bad scanners, etc. Generally, the pre-processing task consists of noise removal, skew correction, and thresholding. Noise removal can be achieved through filtering the image to remove extraneous or stray marks. Skew correction can be performed by estimating the angle of the text. Thresholding is the process of setting all intensity values greater than some threshold value to "on" and is often used as a method of binarizing an image. Thresholding is often used to remove noise when the salient information has either very low or very high intensity values. These techniques attempt to provide clean (or as clean as possible) input to the segmenter.

2.1.2 Segmentation

Segmentation is the process of breaking the input image into segments which contain a single entity. Character-based segmentation is the decomposition of an image into subimages which only contain a single character. Segmentation is dependent on local decisions with regards to shape similarity, and sometimes global decisions with regards to surrounding context. It is a critical step in most OCR systems, and typically the cause of a high proportion of OCR errors. In 1996, Richard Casey and Eric Lecolinet surveyed the available methods and defined three categories for offline character segmentation methods based on how segmentation and classification interact in the OCR process [2]:

- Dissection Approach: a single partitioning of the image into subimages based on "character-like" properties, followed by classification of the subimages
- Classification-based Approach: segmentation where the image is iteratively searched for components that most closely match the classes in the alphabet
- Holistic Approach: recognize words as single units (no character segmentation)

The task of segmenting characters varies in difficulty based on the input type. Fixed pitch machine printed text can typically be

[1] Some of the contents of the background section is adapted from the author's previous research and writings performed during thesis prep.

segmented fairly easily using simple projection analysis (we have done so in this paper). In order to achieve high accuracy on complex problem domains, segmentation and recognition cannot be treated independently. However, a simple problem domain can use more basic techniques and still achieve high accuracy.

Dissection is the decomposition of the input image into a sequence of subimages. The criterion for good segmentation using the dissection approach is the agreement of character properties in the segmented subimage and the expected symbol. The character properties include height, width, separation from neighboring components, disposition along the baseline, etc. Interaction with the classifier is limited to reprocessing of ambiguous recognition results. For example, if the classifier can't make any decision at all, the segment may need to be re-split into two new segments.

The earliest and simplest form of dissection relies on vertical whitespace between successive characters. To make segmentation even easier, a fixed pitch if often used (*pitch* is the number of characters per unit of horizontal distance). In applications that use machines to print the text, text is often printed with a fixed pitch using limited font sets. Hoffman and Mccullough [7] designed a system that could aid in segmentation when a fixed pitch could not be enforced. The system consisted of three steps: 1) detection of the start of a character based on an a priori pitch measurement, 2) a decision to begin testing for the end of the character, and 3) detection of the end of a character. The authors reported a 97% accuracy, but the results were heavily dependent on the quality of the input image.

Projection analysis is another simple one-dimensional segmentation method that uses the vertical projection (or vertical histograms) of "on" pixels to determine dissection candidates. A vertical projection is simply a column-wise count of "on" pixels. If the count falls below a predefined threshold, the column is a candidate for splitting the image. The segmentation boundaries can be further emphasized by observing the derivative of the vertical projection data (well-defined peaks will occur at the character boundaries). Projection analysis works well on high quality machine printed documents. However, vertical projection performs poorly on text which is italicized machine printed text, handwritten text (naturally slanted), or has not been properly skew-corrected.

In graph theory, a connected component is a maximal connected subgraph where two vertices belong to the same connected component if and only if there is a path between them. Connected component analysis can be applied to character segmentation by viewing the character's pixels as block or line adjacency graphs. Intuitively, a character is a single connected component because all pixels "touch" each other (with the exceptions of i's and j's). Connected component analysis is a two-dimensional analysis that works well on proportional fonts and handwritten characters. Connected component analysis works by labeling connected areas of black pixels as components. The components are further processed by drawing bounding boxes around them or based on a detailed analysis of the image. Predefined rules are specified to determine the maximum or minimum size of the bounding boxes. Unfortunately, if characters are broken into multiple pieces (due to pre-processing artifacts, noise, etc.), connected component analysis yields poor segmentation results.

Classification-based segmentation bypasses the requirement to discretely segment the word. No complex dissection algorithm is required. Instead, the segmenter interacts directly with the classifier. Without regard to content, a mobile variable-width window blindly divides the image into many overlapping pieces and chooses the correct segmentation based on the classifier's confidence of the sampled window. Therefore, the criterion for good segmentation is the classification confidence given by the classifier of the subimage.

If the words to be recognized are dictionary words, N-gram statistics can be introduced to classification-based approaches to prune the search space [8]. For example, assuming we have already recognized the letters 't' and 'h', there is a higher probability that the next letter is an 'e' instead of a 'c'. After narrowing the possible guesses, a dictionary can be used to eliminate incorrectly spelled words in favor of correctly spelled words.

Holistic approaches attempt to recognize entire words as single units. The criterion for good segmentation are same as the criterion for good dissection, but using words as the alphabet instead of individual symbols or characters. Unlike the previously mentioned methods, a holistic approach requires a predefined lexicon. For many applications, such as check recognition or postal code reading, this constraint is satisfiable.

2.1.3 Character classification

Character classification is strongly dependent on feature vectors which are extracted from the characters. Feature extraction is the process of transforming the input data into a reduced representation. It is commonly employed when their is too much input data to efficiently process or if the input data is redundant (lots of *data* but not much *information*). This simplification of the input image provides an accurate description of a larger set of data. A common feature vector is image projections which represent the character as a vector of projection counts (discussed above). Naive methods feed entire image matrices to the classifier, while others require experts to develop visual cues to distinguish characters from one another [13]. However, if no such experts exist, other dimensionality reductions, such as *Principle Component Analysis* (PCA), can still be performed. PCA is used to reduce the dimensionality of multidimensional data sets by removing characteristics about the data which have low impact on the variance of the overall dataset. Similarly, it attempts to retain characteristics which contribute most to its variance.

Once the feature vectors are computed, classification can be performed. Classification can be done by finding the nearest neighbor, neural networks [10, 1, 13], or other techniques. However, the classification method is largely non-important in the recognition process; by far the most important decision is the selection of features.

2.2 HIPs

Human Interactive Proofs (HIPs) are a class of automated challenges used to differentiate between legitimate human users and automated, malicious robots on the internet. HIPs have many practical security applications, including preventing the abuse of online services such as free email providers. The term HIP is preferred over the more common (and unfortunately trademarked) term, *Completely Automated Public Turing tests to tell Computers and Humans Apart* (CAPTCHAs). HIP challenges should be easy for a machine to automatically generate, easy for a human to solve, and difficult, or impossible, for another machine to solve. The key to developing a successful HIP challenge is to choose a difficult artificial intelligence problem where a gap exists between human and machine capabilities.

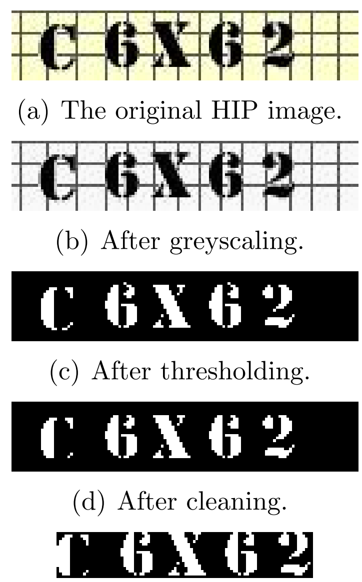
Researchers have suggested HIPs based on hard artificial intelligence problems such as natural language processing [16], character recognition [4], image understanding [5], and speech recognition [9]. Most commercial implementations require the user to transcribe a string of distorted characters with background noise. This type of HIP can be considered broken through techniques such as shape matching [11], distortion estimation [12], and even simple pattern recognition [17]. Unfortunately, most commercial implementations are even easier to break than the research/academic implementations (as this experiment clearly demonstrates).

3 Method

To break the PayPal.com HIP, the problem can be reduced to an OCR task. As mentioned before, the OCR task can be broken down into three steps: *pre-processing*, *segmentation*, and *classification*. For clarity sake, the code is also separated into these three distinct steps.

3.1 Pre-processing

The pre-processing step is arguably the most important step when the image contains adversarial noise (such as HIPs). The noise placed on top of the HIP challenge images is designed to confuse off-the-shelf OCR systems. Before successful segmentation or classification can occur, the noise must be removed. The first step in our process is to convert the image to greyscale. The image is then thresholded to remove the noise (horizontal and vertical lines). The background thresholding technique is incredibly simple but removes nearly all of the noise in the image. Occasionally, additional noise still remains after this step. Therefore, additional cleaning is performed to remove pixels where the entire row has very few "on" pixels. A bounding box is then placed around the string of characters and cropped out.



(a) The original HIP image.

(b) After greyscaling.

(c) After thresholding.

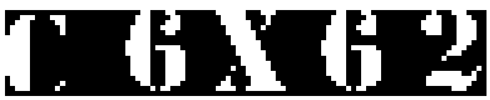
(d) After cleaning.

(e) After bounding.

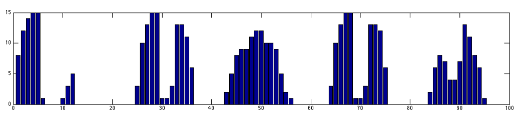
Figure 1: The pre-processing step.

3.2 Segmentation

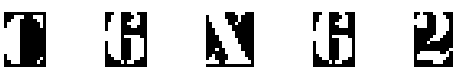
Next, the pre-processed image is fed into the segmenter. A connected components approach was first attempt, but unfortunately the pre-processing step occasionally breaks characters into multiple segments (see the first character in Figure 2a). However, the segmentation process is simplified because the PayPal HIP is always rendered with exactly five characters. Vertical projections [7] and candidate split positions are used to determine segmentation boundaries. Splitting on every projection with zero "on" pixels occasionally causes characters to be split into multiple segments (as was the problem with the CC-based approach). However, empirical exploration shows that every character is at least ten pixels wide. A Hoffman and Mccullough style approach [7] is used and a columnwise scan is performed from the left side of the image to the right side. When the start of a character is detected, ten pixels are skipped and the scan is continued. When the end of a character is detected, the segment is cropped out of the image. The segment is padded out using 0's to a fixed size (20×20) as a requirement of the classification process (correlation requires that the dimensions of the two matrices must agree). This process is repeated until the end of the image has been reached.



(a) Input to segmenter.



(b) Vertical projection analysis.



(c) After splitting the image.

(d) After padding.

Figure 2: The segmentation step.

3.3 Classification

The segmenter feeds the five individual characters to the classifier. Note that the input images are binary images, consisting of 1's for the foreground (the character) and 0's for the background. The classification procedures (defined below) are invoked with the unknown sample image I and the set of template images T. Several of the classifiers computed correlation coefficients. The correlation coefficient (CORR2) between two input vectors or matrices i and j, can be computed as follows:

$$\frac{\sum_m \sum_n (i_{mn} - \bar{i})(j_{mn} - \bar{j})}{\sqrt{\left(\sum_m \sum_n (i_{mn} - \bar{i})^2\right)\left(\sum_m \sum_n (j_{mn} - \bar{j})^2\right)}}$$

where $\bar{i}$ is the mean of the input matrix i and $\bar{j}$ is the mean of the input matrix j.

3.3.1 Pixel Counting

The pixel counting classifier compares the Euclidean distance between pixel counts. The pixels for a binary image I can be counted using the following algorithm:

PIXELCOUNT(I)

$k \leftarrow 0$
for $r \leftarrow 1$ **to** $I_{numRows}$
    do for $c \leftarrow 1$ **to** $I_{numCols}$
        do $k \leftarrow k + I[r][c]$
return k

The pixel count of the input image is then compared against the pixel counts of each of the template images. The index of the template that has the least difference in pixel count is returned as the match:

CLASSIFYPC(I, T)

$D \leftarrow \emptyset$
for each Template $t_i \in T$
    do $d_i \leftarrow abs$(PIXELCOUNT(t_i) -
        PIXELCOUNT(I))
return k such that $d_k = min(D)$

Note that this method does not take any spatial layout into account. Therefore, an image with a pixel in each of it's four corners will match perfectly with a template image with a block of four pixels in the center of the image, even though they are visually dissimilar.

3.3.2 Vertical Projections

The vertical projection classifier compares correlation coefficients of vertical projections. The vertical projection of an image I can be calculated using the following algorithm:

VERTICALPROJECTION(I)

$V \leftarrow \emptyset$
for $r \leftarrow 1$ **to** $I_{numRows}$
  do for $c \leftarrow 1$ **to** $I_{numCols}$
    do $v_c \leftarrow v_c + I[r][c]$
return V

The vertical projection of the input image is then compared against the vertical projections of each of the template images. The index of the template whose vertical projection has the the highest correlation coefficient with the input image's vertical projection is returned as the match:

CLASSIFYVP(I, T)

$R \leftarrow \emptyset$
for each Template $t_i \in T$
  do $r_i \leftarrow$ CORR2(
    VERTICALPROJECTION(t_i),
    VERTICALPROJECTION(I))
return k such that $r_k = max(R)$

3.3.3 Horizontal Projections

The horizontal projection classifier compares correlation coefficients of horizontal projections. The horizontal projection of an image I can be calculated using the following algorithm:

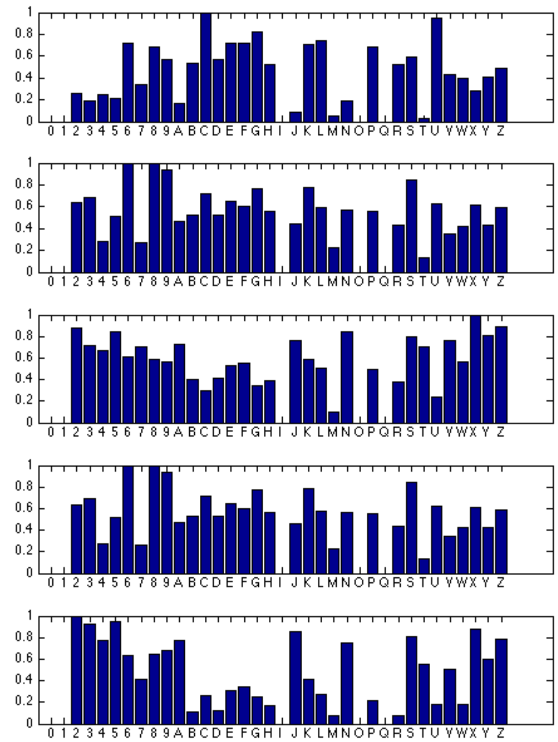


Figure 3: VP confidences for "C6X62".

HORIZONTALPROJECTION(I)

$V \leftarrow \emptyset$
for $r \leftarrow 1$ **to** $I_{numRows}$
  do for $c \leftarrow 1$ **to** $I_{numCols}$
    do $v_r \leftarrow v_r + I[r][c]$
return V

The horizontal projection of the input image is then compared against the horizontal projections of each of the template images. The index of the template whose horizontal projection has the the highest correlation coefficient with the input image's horizontal projection is returned as the match:

ClassifyHP(I, T)

$R \leftarrow \emptyset$
for each Template $t_i \in T$
  do $r_i \leftarrow$ Corr2(
    HorizontalProjection(t_i),
    HorizontalProjection(I))
return k such that $r_k = max(R)$

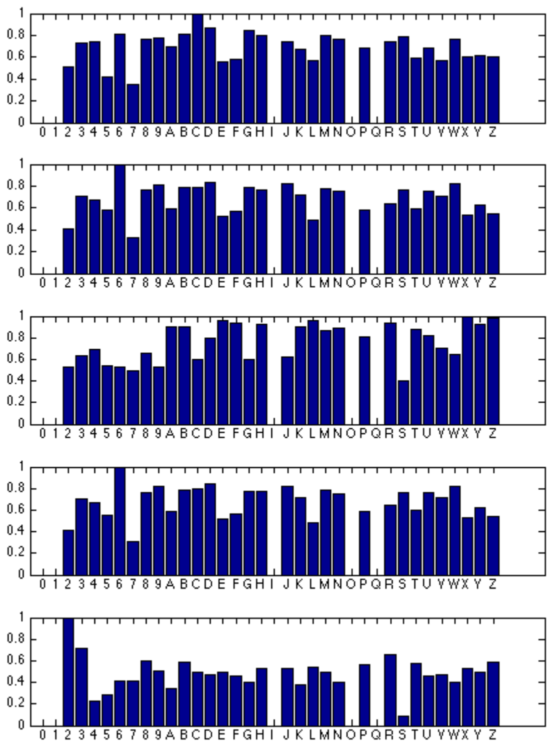


Figure 4: HP confidences for "C6X62".

3.3.4 Template Correlations

The template correlation classifier calculates the 2D correlation coefficients for the input image I and the templates T. The index of the template with the highest 2D correlation coefficient with the input image is returned as the match:

ClassifyTC(I, T)

$R \leftarrow \emptyset$
for each Template $t_i \in T$
  do $r_i \leftarrow$ Corr2(t_i, I)
return k such that $r_k = max(R)$

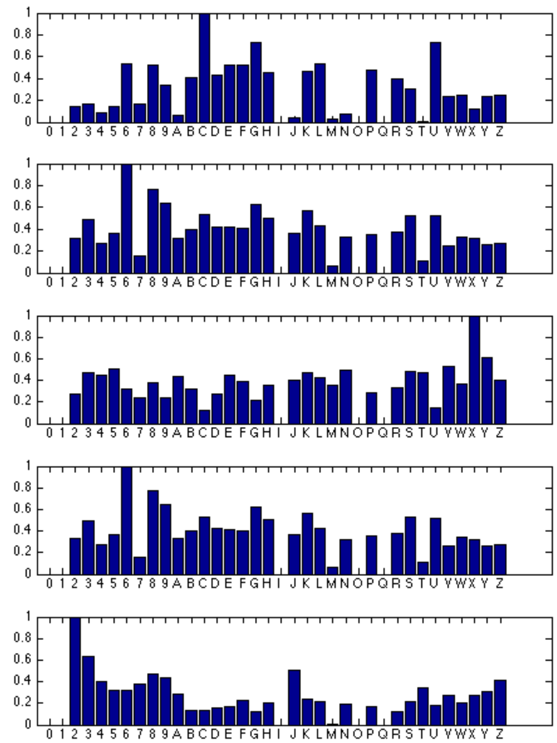


Figure 5: TC confidences for "C6X62".

4 Training

The templates were created from a set of twenty training PayPal HIPs. The set of images were randomly chosen and contain all characters in the character set (note that PayPal does not use `I`, `O`, `Q`, `0`, or `1` in the character set to increase usability for humans). The images were processed using the same pre-processing and segmentation algorithm as specified above. In many cases, the training data has multiple samples for a given character c: $s_1^c, s_2^c, \cdots, s_n^c$. If multiple samples for a single character c exist, the final template t^c is computed by averaging all samples for a given character:

$$t^c = \Big(\sum_{i=1}^{n} s_i^c\Big)/n$$

Informally, this creates more robust templates, as we are using multiple training samples to generate the templates. It can be thought of as many training samples voting on whether or not the ground truth should contain a given pixel.

5 Results

5.1 Testing Results

A sample of 100 random PayPal HIPs were used for testing. The samples were manually downloaded and labeled by visual inspection. The same pre-processing and segmentation algorithms were used for all classifiers. The classifiers are evaluated with several metrics: *HIP accuracy* refers to the percentage of the 100 testing samples which were correctly recognized. *Character accuracy* refers to the percentage of the 500 characters of the 100 testing samples which were correctly recognized. The HIP accuracy should be roughly equal to character accuracy raised to the fifth power (serial repetition). The classifiers which utilize correlation can also return a *confidence value*. The character confidence can be represented by the correlation coefficient (1.0 means a perfect match). A overall string confidence C can be calculated by multiplying each of the character correlation coefficients r_i together:

$$C = \prod_{i=1}^{5} r_i$$

Note that this confidence metric cannot be used with the pixel counting classifier because the that classifier does not utilize correlation during classification. *Running time*, measured in seconds, was clocked on a 2 GHz Intel Core 2 Duo with 2 GB of memory, running Mac OS X 10.4.11 and MATLAB R2007a. Full outputs from all four classifiers are located in Appendix B.

The following is a comparison of the four classifiers: pixel counting (PC), vertical projections (VP), horizontal projections (HP), and template correlations (TC).

	PC	VP	HP	TC
A_{Char}	63.2%	99.4%	100%	100%
A_{HIP}	8%	97%	100%	100%
C_{avg}	n/a	98.9%	98.8%	95.9%
C_{min}	n/a	89.1%	93.9%	67.9%
T_{avg}	0.02s	0.06s	0.06s	0.06s

where A_{Char} is the character accuracy, A_{HIP} is the HIP accuracy, C_{avg} is the average overall string confidence, C_{min} is the lowest overall string confidence observed during testing, and T_{avg} is the average recognition running time in seconds.

The only benefit that the pixel counting method has over the others is run time. The vertical projection classifier made three misclassifications: '27LP5' recognized as '27LP2', '5RESL' recognized as '2RESL', and '5SMWJ' recognized as '2SMWJ'. We can see that all three mistakes were made due to a misclassification of a '5' as a '2'. This shows that vertical projections are not a sufficient method for differentiating between 5's and 2's.

Figure 6 plots the confidence of the three classifiers (vertical projections shown in blue, horizontal projections shown in green, and template correlations shown in red) for all 100 testing samples. We can see that even though they use different classification techniques, the confidence values seem to be consistent with one another. That is to say, that all three classifiers achieve low confidences on the same images. For example, sample #72 achieved very low confidence for both the horizontal projection and template correlation classifiers. Similarly, all three classifiers performed very well on sample #28. This indicates that samples with low confidence values may exhibit some pre-processing artifacts that make classification a difficult task, no matter what the technique.

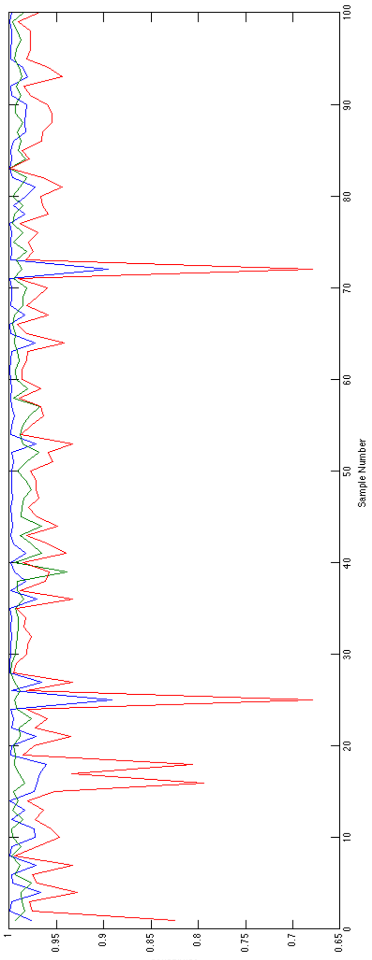


Figure 6: A comparison of confidences for the 100 test samples. VP=blue, HP=green, TC=red.

5.2 Example

This section visually demonstrates the entire recognition process using an example PayPal HIP image. Figure 1 illustrates the preprocessing step. Figure 2 displays the the segmentation process. Figure 3 shows the confidence values for the vertical projection classifier. The overall confidence for the example HIP is 0.992 $(0.995 * 0.999 * 1.0 * 0.998 * 1.0)$. Notice that there are several other characters with very high confidences. Figure 4 shows the confidence values for the horizontal projection classifier. The overall confidence for the example HIP is 0.994 $(0.995 * 1.0 * 1.0 * 0.999 * 1.0)$. Notice that the range of the confidence values is fairly small and several peaks exist. Figure 5 shows the confidence values for the template correlation classifier. The overall confidence for the example HIP is 0.992 $(1.0 * 0.999 * 0.997 * 0.996 * 1.0)$. Notice that there is only a single peak in the confidence values for every character. This indicates that the classifier is very good at discriminating between character classes.

6 Conclusion

We have presented a robust way to automatically recognizing the character strings inside of a PayPal.com HIP using a three step *pre-process, segment, classify* algorithm. Four classifiers (pixel counting, vertical projections, horizontal projections, and template correlations) were implemented, evaluated, and compared. Two of the classifiers have achieved perfect HIP accuracy on the test set of 100 images. Upon visual inspection of the correlation coefficients for several test images, we see that the template correlation classifier discriminates better than the other classifiers and is strongly recommended.

References

[1] Hadar I. Avi-Itzhak, Thanh A. Diep, and Harry Garland. High accuracy optical character recognition using neural networks with centroid dithering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):218–224, February 1995.

[2] Richard G. Casey and Eric Lecolinet. A survey of methods and strategies in character segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):690–706, July 1996.

[3] Kumar Chellapilla and Patrice Y. Simard. Using machine learning to break visual human interaction proofs (HIPs). In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 265–272, Cambridge, MA, December 2004. MIT Press.

[4] Monica Chew and Henry S. Baird. Baffletext: A human interactive proof. In *IST/SPIE Document Recognition and Retrieval X Conference*, pages 305–316, January 2003.

[5] Monica Chew and J. Doug Tygar. Image recognition captchas. In Kan Zhang and Yuliang Zheng, editors, *In Proceedings of the 7th International Information Security Conference (ISC 2004)*, volume 3225 of *Lecture Notes in Computer Science*, pages 268–279, Palo Alto, CA, September 2004. Springer Berlin / Heidelberg.

[6] Sam Hocevar. Pwntcha. Online http://libcaca.zoy.org/wiki/PWNtcha, January 2005.

[7] Richard L. Hoffman and J. Warren McCullough. Segmentation Methods for Recognition of Machine-Printed Characters. *IBM Journal of Research and Development*, 15(2):153–165, March 1971.

[8] Jr. Kenneth Crawford Hayes. Reading handwritten words using hierarchical relaxation. In *Computer Graphics and Image Processing*, volume 14, pages 344–364, December 1980.

[9] Greg Kochanski, Daniel P. Lopresti, and Chilin Shih. A reverse turing test using speech. In *Proceedings of the 7th International Conference on Spoken Language Processing*, pages 1357–1360, Denver, Colorado, September 2002.

[10] Nallasamy Mani and Bala Srinivasan. Application of artificial neural network model for optical character recognition. In *International Conference on Systems, Man and Cybernetics*, volume 3, pages 2517–2520. IEEE Computer Society, October 1997.

[11] Greg Mori and Jitendra Malik. Recognizing objects in adversarial clutter: breaking a visual captcha. In *Conference on Computer Vision and Pattern Recognition*, volume 1, pages 134–141, Madison, WI, USA, June 2003. IEEE Computer Society.

[12] Gabriel Moy, Nathan Jones, Curt Harkless, and Randall Potter. Distortion estimation techniques in solving visual captchas. In *Conference on Computer Vision and Pattern Recognition*, volume 02, pages 23–28, Los Alamitos, CA, USA, June 2004. IEEE Computer Society.

[13] E.M. de A. Neves, A. Gonzaga, and A.F.F. Slaets. A multi-font character recognition based on its fundamental features by artificial neural networks. In *Proceedings of the 2nd Workshop on Cybernetic Vision*, pages 196–201, December 1996.

[14] Wikipedia. Optical character recognition — wikipedia, the free encyclopedia, 2008. [Online; accessed 8-May-2008].

[15] R. Allen Wilkinson and Michael D. Garris. Comparison of massively parallel hand-print segmenters. Technical report, National Institute of Standards and Technology (CSL). Advanced Systems Division, Gaithersburg, MD, September 1992.

[16] Pablo Ximenes, Andre dos Santos, Marcial Fernandez, and Joaquim Celesti. A captcha in the text domain. In R. Meersman, Z. Tari, and P. Herrero, editors, *On the Move to Meaningful Internet Systems Workshop*, volume 4277/2006 of *Lecture Notes in Computer Science*, pages 605–615. Springer Berlin / Heidelberg, November 2006.

[17] Jeff Yan and Ahmad Salah El Ahmad. Breaking visual captchas with naive pattern recognition algorithms. In *Proceedings of the 23rd Annual Computer Security Applications Conference*, pages 279–291, December 2007.

Appendices

A MATLAB Code

This section contains the MATLAB code used to break the PayPal HIP. The code is thoroughly commented and should be self explanatory.

A.1 recognizeAll.m

```
function recognizeAll()
% Performs recognition of the entire testing set of PayPal CAPTCHA images
% by preprocessing, segmentation, and classification.
%
% Created by Kurt Alfred Kluever (kurt@kloover.com)

    testingDir = 'testing/';

    testingSamples = dir(strcat(testingDir, '*.jpg'));

    numTestingSamples = size(testingSamples, 1);

    charCorrect = 0;
    charWrong = 0;
    hipCorrect = 0;
    confidences = zeros(1, numTestingSamples);
    tic

    % For each of the testing images...
    for i=1:numTestingSamples
       fn = strcat(testingDir, testingSamples(i).name);

       % Perform recognition and record the result and confidence
       [chars c] = recognize(fn);

       confidences(i) = c;

       fn = strrep(fn, testingDir, '');
       fn = strrep(fn, '.jpg', '');

       % Print out the results
       fprintf('Actual: %s Decoded: %s Confidence: %f', fn, chars, c);
       if (strcmp(fn, chars) == 0)
           fprintf(' Incorrect\n');
       else
           fprintf(' Correct\n');
           hipCorrect = hipCorrect + 1;
       end

       for j=1:5
          if (strcmp(fn(j), chars(j)) == 0)
              charWrong = charWrong + 1;
          else
              charCorrect = charCorrect + 1;
          end
       end
    end
    toc

    charAcc = charCorrect / (charCorrect + charWrong);
    hipAcc = hipCorrect / numTestingSamples;
    avgConfidence = sum(confidences) / numTestingSamples;
    minConfidence = min(confidences);

    fprintf('Character Accuracy: %f\n', charAcc);
    fprintf('HIP Accuracy: %f\n', hipAcc);
    fprintf('Average confidence: %f\n', avgConfidence);
    fprintf('Minimum confidence: %f\n', minConfidence);
end
```

A.2 recognize.m

```
function [decoded confidence] = recognize(imageFileName)
% Performs recognition of PayPal CAPTCHA images by preprocessing,
% segmentation, and classification. To switch which classifier is being
% used, simply uncomment out the one you wish to use!
%
% Created by Kurt Alfred Kluever (kurt@kloover.com)

    % If the templates haven't been created yet, make them now!
    if (exist('templates.mat', 'file') == 0)
        fprintf('Training templates do not exist. Creating them now...');
        makeTemplates();
        fprintf('DONE\n');
    end
    % 1. Load and preprocess the image
    preprocessed = preprocess(imread(imageFileName));

    % 2. Segmentation the image into characters
    segmented = segment(preprocessed);

    % 3. Classifiy the characters
    [decoded confidence] = classify(segmented, 'PixelCounts');
%    [decoded confidence] = classify(segmented, 'VerticalProjections');
%    [decoded confidence] = classify(segmented, 'HorizontalProjections');
%    [decoded confidence] = classify(segmented, 'TemplateMatching');
end
```

A.3 preprocess.m

```
function bounded = preprocess(i)
% Performs preprocessing on the input image. The image is first converted
% to greyscale and then thresholded. Random noise is removed via
% thresholding. A bounding box is then placed around the entire image.
% Returns the preprocessed image.
%
% Created by Kurt Alfred Kluever (kurt@kloover.com)

    % Convert the color image to grey scale
    greyScale = rgb2gray(i);

    % Threshold out the background noise
    thresholded = greyScale < 30;

    % Remove any random noise that will hurt the char offset
    rows = size(thresholded, 1);
    row = 1;
    while (row < rows)
        rowSum = sum(thresholded(row,:));
        if (rowSum < 5 && rowSum > 0)
           thresholded(row,:) = 0;
        end
        row = row + 1;
    end

    % Place a bounding box around the image
    bb = regionprops(double(thresholded), 'BoundingBox');

    % Crop out the contents of the bounding box
    bounded = imcrop(thresholded, bb.BoundingBox);
end
```

A.4 segment.m

```
function [retVal] = segment(bounded)
% Performs character segmentation of the preprocessed input image.
% Returns the segmented set of characters.
%
% Created by Kurt Alfred Kluever (kurt@kloover.com)

    % Create the return value (5 images, 20x20 in size)
    retVal = zeros(20, 20, 5);

    % Get the size of the input image
    [rows cols] = size(bounded);

    col=3;
    startCol = 1;
    charIndex = 1;

    % While we aren't at the end of the image...
    while(col < cols)
        col = col + 10;
        % Scan forward while columns contain data
        while (col+2 <= cols) && (sum(sum(bounded(:,col:col+2))) > 0)
            col = col + 1;
        end

        % Crop the character out of the image
        a = imcrop(bounded, [startCol 1 (col - startCol) rows]);

        % Pad out to 20 rows and 20 cols with 0's
        [charRows charCols] = size(a);
        a = padarray(a, [(20 - charRows) (20 - charCols)], 'post');

        % Set the character into the return value
        retVal(:,:,charIndex) = a;

        % Increment the characters index
        charIndex = charIndex + 1;

        % Increment the column counter
        col = col + 1;
        startCol = col;

        % Advance the column counter while there is white space
        while (col <= cols) && (sum(sum(bounded(:,col))) == 0)
            col = col + 1;
            startCol = startCol + 1;
        end
    end
end
```

A.5 classify.m

```
function [decoded confidence] = classify(chars, method)
% Performs character classification of the segmented input image using
% pixel counts, vertical projections, horizontal projections, or template
% matching.
% Returns the classified string of characters and the confidence.
%
% Created by Kurt Alfred Kluever (kurt@kloover.com)

    % Load the templates
    load templates;

    % Turn off the warnings about dividing by zero
    warning off MATLAB:divideByZero

    % Setup the decoded result
    decoded = char(zeros(1,5));

    % Confidence starts at 1.0 (perfect)
    confidence = 1.0;

    % For each of the 5 characters in the image...
    for i=1:5

        % Correlation results for all the template images
        allCorrs = zeros(1, 36);

        % For each of the templates...
        for j=1:36
            if (strcmp(method, 'PixelCounts') == 1)
                % Note that we subtract it from 50 so that we can still
                % use max() to find the correct index below!
                tempSum = sum(sum(templates(:,:,j)));
                inSum = sum(sum(chars(:,:,i)));
                allCorrs(j) = 50 - abs(tempSum - inSum);
            elseif (strcmp(method, 'VerticalProjections') == 1)
                tempVP = sum(templates(:,:,j));
                inVP = sum(chars(:,:,i));
                allCorrs(j) = corr2(tempVP, inVP);
            elseif (strcmp(method, 'HorizontalProjections') == 1)
                tempHP = sum(templates(:,:,j)');
                inHP = sum(chars(:,:,i)');
                allCorrs(j) = corr2(tempHP, inHP);
            else % Do template matching by default
                temp = templates(:,:,j);
                in = chars(:,:,i);
                allCorrs(j) = corr2(temp, in);
            end
        end

        % Used for plotting data
%        subplot(5,1,i); bar(abs(allCorrs));
%        set(gca,'XTick',1:36)
```

```
%        set(gca,'YTick',0:0.2:1)
        %set(gca,'XTickLabel',{'0','1','2','3','4','5','6','7','8','9',
        %    'A','B','C','D','E','F','G','H','I','J','K','L','M',
        %    'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'})

        % Find the first template with the best correlation
        index = find(allCorrs == max(allCorrs), 1);

        % Update the classification confidence
        confidence = max(allCorrs) * confidence;

        % Convert it from an index back into ASCII
        if (index <= 10) % number
            index = index + 47;
        elseif (index >= 11 && index <= 36)% upper case character
            index = index + 54;
        else
            % We should never get here
        end

        % Store the decoded character
        decoded(i) = char(index);
    end
end
```

A.6 makeTemplates.m

```
function makeTemplates()
% Creates the classification templates using a set of training images. The
% training images are labeled as ground truth by their filenames.
%
% Created by Kurt Alfred Kluever (kurt@kloover.com)

    % Set aside room for the templates and the counts
    templates = zeros(20,20,36);
    counts = zeros(1,36);

    % Load the images from the training directory
    trainingDir = 'training/';
    trainingSamples = dir(strcat(trainingDir, '*.jpg'));
    [numTrainingSamples] = size(trainingSamples);

    % For each of the training images...
    for i=1:numTrainingSamples
        filename = strcat(trainingDir, trainingSamples(i).name);

        % Perform preprocessing
        bounded = preprocess(imread(filename));

        % Perform segmentation
        chars = segment(bounded);

        % Remove the directory name from the filename
        filename = strrep(filename, trainingDir, '');

        % For each of the characters...
        for i=1:5
            % Convert from ASCII character to ASCII int (an index)
            asciiz = uint8(filename(i));
            if (asciiz >= 48 && asciiz <= 57) % number
                asciiz = asciiz - 47;
            elseif (asciiz >= 65 && asciiz <=90) % upper case character
                asciiz = asciiz - 54;
            else
                % We should never, ever get here!
            end

            % Add the template to the training library
            templates(:,:,asciiz) = templates(:,:,asciiz) + chars(:,:,i);

            % Update the number of times we have seen this character
            counts(asciiz) = counts(asciiz) + 1;
        end
    end

    % Go through and normalize (average) the templates
    for i=1:36
        templates(:,:,i) = templates(:,:,i) / counts(i);
    end

    % Save them out to the file for later
    save('templates.mat', 'templates', 'counts');
end
```

B MATLAB Output

This section contains the output of the recognizeAll MATLAB script using the different classifiers. The output contains the recognition attempts for each of the 100 testing samples, as well as the metrics on which the classifiers are evaluated.

B.1 Pixel Counting

```
Actual: 27LP5 Decoded: 27V62 Confidence: 302190625.000000 Incorrect
Actual: 2CAD2 Decoded: 24AD2 Confidence: 304166666.666667 Incorrect
Actual: 2GX7T Decoded: 2S87T Confidence: 303125000.000000 Incorrect
Actual: 2MR5A Decoded: 5MB2A Confidence: 300152222.222222 Incorrect
Actual: 2YGS4 Decoded: 2YGS4 Confidence: 295082083.333333 Correct
Actual: 373MH Decoded: 373MH Confidence: 291946875.000000 Correct
Actual: 3B5T3 Decoded: 3B5T3 Confidence: 286080000.000000 Correct
Actual: 3EYHU Decoded: 33YHT Confidence: 309375000.000000 Incorrect
Actual: 3HKSY Decoded: 3HKSJ Confidence: 307312500.000000 Incorrect
Actual: 45V7U Decoded: 45V7T Confidence: 298099816.203704 Incorrect
Actual: 4BK54 Decoded: 4BK54 Confidence: 300165742.592593 Correct
Actual: 4EE3C Decoded: 43334 Confidence: 296096111.111111 Incorrect
Actual: 4R2WH Decoded: 4B2WH Confidence: 305775937.500000 Incorrect
Actual: 4RCX4 Decoded: 4BC84 Confidence: 293114869.444444 Incorrect
Actual: 5LX29 Decoded: 5V829 Confidence: 302190625.000000 Incorrect
Actual: 5RESL Decoded: 2B3SV Confidence: 283318000.000000 Incorrect
Actual: 5SE4L Decoded: CS34L Confidence: 295020000.000000 Incorrect
Actual: 5SMWJ Decoded: 2SMWA Confidence: 300655833.333333 Incorrect
Actual: 6V2EP Decoded: 6V236 Confidence: 301173010.185185 Incorrect
Actual: 7KPZ3 Decoded: 7K6N3 Confidence: 301166250.000000 Incorrect
Actual: 8285R Decoded: 82G5B Confidence: 295082083.333333 Incorrect
Actual: 8HL84 Decoded: GHL84 Confidence: 304239375.000000 Incorrect
Actual: 8K6GS Decoded: GK6G6 Confidence: 290183720.750000 Incorrect
Actual: 9LLZK Decoded: 9VLZK Confidence: 300568125.000000 Incorrect
Actual: A4BL7 Decoded: 94BL7 Confidence: 298154587.500000 Incorrect
Actual: AU4CX Decoded: AT4C8 Confidence: 301104166.666667 Incorrect
Actual: AW94W Decoded: AWA4W Confidence: 301142968.750000 Incorrect
Actual: AZX7V Decoded: AN87V Confidence: 308343750.000000 Incorrect
Actual: B4ABZ Decoded: B4ABN Confidence: 304208333.333333 Incorrect
Actual: BDKJC Decoded: BDKAC Confidence: 303125000.000000 Incorrect
Actual: BE3S7 Decoded: B33S7 Confidence: 297123750.000000 Incorrect
Actual: BM9X8 Decoded: BMA8G Confidence: 299145000.000000 Incorrect
Actual: C3MLK Decoded: C3MLK Confidence: 293101666.666667 Correct
Actual: C3TH2 Decoded: 43TH2 Confidence: 295102500.000000 Incorrect
Actual: C6X62 Decoded: C6882 Confidence: 301104166.666667 Incorrect
Actual: C852Z Decoded: 4G52N Confidence: 287272447.000000 Incorrect
Actual: CAZMA Decoded: CANMA Confidence: 305250000.000000 Incorrect
Actual: CNDW7 Decoded: CNDW7 Confidence: 292621415.625000 Correct
Actual: CSSBK Decoded: LSSBK Confidence: 300125000.000000 Incorrect
Actual: DHUBH Decoded: DHTBH Confidence: 294152512.500000 Incorrect
Actual: DKGWG Decoded: DKGWG Confidence: 286709871.875000 Correct
Actual: E7XGT Decoded: 37889 Confidence: 300155625.000000 Incorrect
Actual: EC6YM Decoded: 3C6YM Confidence: 291147655.555555 Incorrect
Actual: EF3SP Decoded: 3F3S6 Confidence: 300731666.666667 Incorrect
Actual: EL2ZY Decoded: 3L2NY Confidence: 306250000.000000 Incorrect
Actual: EUYPS Decoded: 3TJ6S Confidence: 302180277.777778 Incorrect
Actual: F93UV Decoded: VT3TV Confidence: 294139173.611111 Incorrect
Actual: FUSGP Decoded: VTSG8 Confidence: 302114583.333333 Incorrect
Actual: FUTPC Decoded: FT96C Confidence: 288792619.500000 Incorrect
Actual: FYUYM Decoded: FYTYM Confidence: 292658333.333333 Incorrect
Actual: GBAR3 Decoded: 8BJB3 Confidence: 310416666.666667 Incorrect
Actual: GF3GD Decoded: GV3GD Confidence: 269425785.416667 Incorrect
Actual: GH25P Decoded: SH25G Confidence: 304239375.000000 Incorrect
Actual: GMLVU Decoded: SMLVT Confidence: 307305555.555556 Incorrect
Actual: GR2T4 Decoded: SB294 Confidence: 307312500.000000 Incorrect
Actual: GRY8X Decoded: SBY86 Confidence: 298124166.666667 Incorrect
Actual: GYPGJ Decoded: GY8GA Confidence: 288150625.000000 Incorrect
Actual: HN94A Decoded: HNA4A Confidence: 307312500.000000 Incorrect
Actual: HTJGD Decoded: HTASD Confidence: 297062500.000000 Incorrect
Actual: HX72C Decoded: H872C Confidence: 297092812.500000 Incorrect
Actual: J2LE3 Decoded: A2L33 Confidence: 306250000.000000 Incorrect
Actual: J4KCB Decoded: A4KCB Confidence: 289060000.000000 Incorrect
Actual: JM9SZ Decoded: AMTSN Confidence: 302166666.666667 Incorrect
Actual: JZ53E Decoded: AN533 Confidence: 298124166.666667 Incorrect
Actual: K8LY2 Decoded: KGLY2 Confidence: 297123750.000000 Incorrect
Actual: KBVUK Decoded: KBVTK Confidence: 305229166.666667 Incorrect
```

```
Actual: KWPRN Decoded: KW6BN Confidence: 296603125.000000 Incorrect
Actual: L4HGA Decoded: V4HSA Confidence: 302190625.000000 Incorrect
Actual: LK4YE Decoded: VK4J3 Confidence: 301173148.148148 Incorrect
Actual: LL6FJ Decoded: LLSV9 Confidence: 304218750.000000 Incorrect
Actual: M224L Decoded: M224V Confidence: 303208101.851852 Incorrect
Actual: M6AXZ Decoded: M6TSZ Confidence: 304647055.555556 Incorrect
Actual: M8H4U Decoded: MGH4T Confidence: 296126325.000000 Incorrect
Actual: MG8ES Decoded: MGG3S Confidence: 292131262.500000 Incorrect
Actual: NU8JM Decoded: NTGAM Confidence: 301125000.000000 Incorrect
Actual: NYPU3 Decoded: NY8T3 Confidence: 306250000.000000 Incorrect
Actual: P2N4P Decoded: 62N48 Confidence: 308347222.222222 Incorrect
Actual: R32WR Decoded: B32WB Confidence: 298624375.000000 Incorrect
Actual: RDPL2 Decoded: BD6V2 Confidence: 287172666.666667 Incorrect
Actual: RNWBS Decoded: BNWBS Confidence: 310937500.000000 Incorrect
Actual: RP5GX Decoded: B65S8 Confidence: 308347222.222222 Incorrect
Actual: S9FES Decoded: STF3S Confidence: 308928571.428571 Incorrect
Actual: TBLU2 Decoded: TBLT2 Confidence: 312500000.000000 Incorrect
Actual: TGBSX Decoded: TGBS8 Confidence: 303125000.000000 Incorrect
Actual: U3V4C Decoded: T3V4C Confidence: 294098476.388889 Incorrect
Actual: UE64X Decoded: T3648 Confidence: 300152222.222222 Incorrect
Actual: UWUCH Decoded: TWTCH Confidence: 304749843.750000 Incorrect
Actual: W4GKY Decoded: W4SKY Confidence: 299645208.333333 Incorrect
Actual: W9B2K Decoded: W9B2K Confidence: 292666893.750000 Correct
Actual: XA3FW Decoded: 6A3VW Confidence: 301678334.027778 Incorrect
Actual: XCERZ Decoded: GC3BN Confidence: 300093750.000000 Incorrect
Actual: XKSLN Decoded: 8K6VN Confidence: 305243055.555556 Incorrect
Actual: XWET3 Decoded: 8W3T3 Confidence: 298500000.000000 Incorrect
Actual: YAWG2 Decoded: YAWS2 Confidence: 310937500.000000 Incorrect
Actual: YBAUK Decoded: YWATK Confidence: 295312500.000000 Incorrect
Actual: YRN26 Decoded: YBN26 Confidence: 298124166.666667 Incorrect
Actual: YT4TZ Decoded: Y949N Confidence: 298154587.500000 Incorrect
Actual: Z69AB Decoded: NS9JB Confidence: 301166250.000000 Incorrect
Actual: ZXTRV Decoded: N8TBV Confidence: 311458333.333333 Incorrect
Actual: ZYZBH Decoded: ZYNBH Confidence: 293436000.000000 Incorrect
Elapsed time is 2.787840 seconds.
Character Accuracy: 0.632000
HIP Accuracy: 0.080000
Average confidence: 299477068.519656
Minimum confidence: 269425785.416667
```

B.2 Vertical Projections

```
Actual: 27LP5 Decoded: 27LP2 Confidence: 0.976889 Incorrect
Actual: 2CAD2 Decoded: 2CAD2 Confidence: 0.999403 Correct
Actual: 2GX7T Decoded: 2GX7T Confidence: 0.997374 Correct
Actual: 2MR5A Decoded: 2MR5A Confidence: 0.966513 Correct
Actual: 2YGS4 Decoded: 2YGS4 Confidence: 0.996232 Correct
Actual: 373MH Decoded: 373MH Confidence: 0.997336 Correct
Actual: 3B5T3 Decoded: 3B5T3 Confidence: 0.971717 Correct
Actual: 3EYHU Decoded: 3EYHU Confidence: 0.999560 Correct
Actual: 3HKSY Decoded: 3HKSY Confidence: 0.997098 Correct
Actual: 45V7U Decoded: 45V7U Confidence: 0.972766 Correct
Actual: 4BK54 Decoded: 4BK54 Confidence: 0.974106 Correct
Actual: 4EE3C Decoded: 4EE3C Confidence: 0.997275 Correct
Actual: 4R2WH Decoded: 4R2WH Confidence: 0.983383 Correct
Actual: 4RCX4 Decoded: 4RCX4 Confidence: 0.999162 Correct
Actual: 5LX29 Decoded: 5LX29 Confidence: 0.973967 Correct
Actual: 5RESL Decoded: 2RESL Confidence: 0.970542 Incorrect
Actual: 5SE4L Decoded: 5SE4L Confidence: 0.967560 Correct
Actual: 5SMWJ Decoded: 2SMWJ Confidence: 0.960504 Incorrect
Actual: 6V2EP Decoded: 6V2EP Confidence: 0.998750 Correct
Actual: 7KPZ3 Decoded: 7KPZ3 Confidence: 0.997631 Correct
Actual: 8285R Decoded: 8285R Confidence: 0.972187 Correct
Actual: 8HL84 Decoded: 8HL84 Confidence: 0.997577 Correct
Actual: 8K6GS Decoded: 8K6GS Confidence: 0.995977 Correct
Actual: 9LLZK Decoded: 9LLZK Confidence: 0.998529 Correct
Actual: A4BL7 Decoded: A4BL7 Confidence: 0.891131 Correct
Actual: AU4CX Decoded: AU4CX Confidence: 0.997247 Correct
Actual: AW94W Decoded: AW94W Confidence: 0.965818 Correct
Actual: AZX7V Decoded: AZX7V Confidence: 0.999481 Correct
Actual: B4ABZ Decoded: B4ABZ Confidence: 0.999381 Correct
Actual: BDKJC Decoded: BDKJC Confidence: 0.998347 Correct
Actual: BE3S7 Decoded: BE3S7 Confidence: 0.998217 Correct
Actual: BM9X8 Decoded: BM9X8 Confidence: 0.997814 Correct
Actual: C3MLK Decoded: C3MLK Confidence: 0.998588 Correct
Actual: C3TH2 Decoded: C3TH2 Confidence: 0.997867 Correct
Actual: C6X62 Decoded: C6X62 Confidence: 0.999225 Correct
Actual: C852Z Decoded: C852Z Confidence: 0.971098 Correct
Actual: CAZMA Decoded: CAZMA Confidence: 0.998813 Correct
Actual: CNDW7 Decoded: CNDW7 Confidence: 0.982417 Correct
Actual: CSSBK Decoded: CSSBK Confidence: 0.994183 Correct
Actual: DHUBH Decoded: DHUBH Confidence: 0.998789 Correct
Actual: DKGWG Decoded: DKGWG Confidence: 0.982206 Correct
Actual: E7XGT Decoded: E7XGT Confidence: 0.995325 Correct
Actual: EC6YM Decoded: EC6YM Confidence: 0.998932 Correct
Actual: EF3SP Decoded: EF3SP Confidence: 0.996486 Correct
Actual: EL2ZY Decoded: EL2ZY Confidence: 0.997153 Correct
Actual: EUYPS Decoded: EUYPS Confidence: 0.998035 Correct
Actual: F93UV Decoded: F93UV Confidence: 0.996609 Correct
Actual: FUSGP Decoded: FUSGP Confidence: 0.997908 Correct
Actual: FUTPC Decoded: FUTPC Confidence: 0.997816 Correct
Actual: FYUYM Decoded: FYUYM Confidence: 0.997840 Correct
Actual: GBAR3 Decoded: GBAR3 Confidence: 0.995165 Correct
Actual: GF3GD Decoded: GF3GD Confidence: 0.997139 Correct
Actual: GH25P Decoded: GH25P Confidence: 0.971557 Correct
Actual: GMLVU Decoded: GMLVU Confidence: 0.998557 Correct
Actual: GR2T4 Decoded: GR2T4 Confidence: 0.997531 Correct
Actual: GRY8X Decoded: GRY8X Confidence: 0.995025 Correct
Actual: GYPGJ Decoded: GYPGJ Confidence: 0.997729 Correct
Actual: HN94A Decoded: HN94A Confidence: 0.998921 Correct
Actual: HTJGD Decoded: HTJGD Confidence: 0.997868 Correct
Actual: HX72C Decoded: HX72C Confidence: 0.998697 Correct
Actual: J2LE3 Decoded: J2LE3 Confidence: 0.999015 Correct
Actual: J4KCB Decoded: J4KCB Confidence: 0.998579 Correct
Actual: JM9SZ Decoded: JM9SZ Confidence: 0.997847 Correct
Actual: JZ53E Decoded: JZ53E Confidence: 0.972809 Correct
Actual: K8LY2 Decoded: K8LY2 Confidence: 0.998339 Correct
Actual: KBVUK Decoded: KBVUK Confidence: 0.999290 Correct
Actual: KWPRN Decoded: KWPRN Confidence: 0.983888 Correct
Actual: L4HGA Decoded: L4HGA Confidence: 0.998319 Correct
Actual: LK4YE Decoded: LK4YE Confidence: 0.997780 Correct
Actual: LL6FJ Decoded: LL6FJ Confidence: 0.997150 Correct
Actual: M224L Decoded: M224L Confidence: 0.999451 Correct
Actual: M6AXZ Decoded: M6AXZ Confidence: 0.895080 Correct
Actual: M8H4U Decoded: M8H4U Confidence: 0.998581 Correct
Actual: MG8ES Decoded: MG8ES Confidence: 0.997565 Correct
Actual: NU8JM Decoded: NU8JM Confidence: 0.998089 Correct
Actual: NYPU3 Decoded: NYPU3 Confidence: 0.997687 Correct
Actual: P2N4P Decoded: P2N4P Confidence: 0.999295 Correct
Actual: R32WR Decoded: R32WR Confidence: 0.983157 Correct
Actual: RDPL2 Decoded: RDPL2 Confidence: 0.995750 Correct
Actual: RNWBS Decoded: RNWBS Confidence: 0.982853 Correct
Actual: RP5GX Decoded: RP5GX Confidence: 0.972836 Correct
Actual: S9FES Decoded: S9FES Confidence: 0.996818 Correct
Actual: TBLU2 Decoded: TBLU2 Confidence: 0.999953 Correct
Actual: TGBSX Decoded: TGBSX Confidence: 0.997203 Correct
Actual: U3V4C Decoded: U3V4C Confidence: 0.998725 Correct
Actual: UE64X Decoded: UE64X Confidence: 0.995831 Correct
Actual: UWUCH Decoded: UWUCH Confidence: 0.982718 Correct
Actual: W4GKY Decoded: W4GKY Confidence: 0.983459 Correct
Actual: W9B2K Decoded: W9B2K Confidence: 0.983053 Correct
Actual: XA3FW Decoded: XA3FW Confidence: 0.981909 Correct
Actual: XCERZ Decoded: XCERZ Confidence: 0.997322 Correct
Actual: XKSLN Decoded: XKSLN Confidence: 0.998212 Correct
Actual: XWET3 Decoded: XWET3 Confidence: 0.980490 Correct
Actual: YAWG2 Decoded: YAWG2 Confidence: 0.985825 Correct
Actual: YBAUK Decoded: YBAUK Confidence: 0.998544 Correct
Actual: YRN26 Decoded: YRN26 Confidence: 0.998172 Correct
Actual: YT4TZ Decoded: YT4TZ Confidence: 0.997950 Correct
Actual: Z69AB Decoded: Z69AB Confidence: 0.997897 Correct
Actual: ZXTRV Decoded: ZXTRV Confidence: 0.999718 Correct
Actual: ZYZBH Decoded: ZYZBH Confidence: 0.997633 Correct
Elapsed time is 6.175383 seconds.
Character Accuracy: 0.994000
HIP Accuracy: 0.970000
Average confidence: 0.989917
Minimum confidence: 0.891131
```

B.3 Horizontal Projections

```
Actual: 27LP5 Decoded: 27LP5 Confidence: 0.994007 Correct
Actual: 2CAD2 Decoded: 2CAD2 Confidence: 0.984106 Correct
Actual: 2GX7T Decoded: 2GX7T Confidence: 0.986540 Correct
Actual: 2MR5A Decoded: 2MR5A Confidence: 0.987428 Correct
Actual: 2YGS4 Decoded: 2YGS4 Confidence: 0.976827 Correct
Actual: 373MH Decoded: 373MH Confidence: 0.993554 Correct
Actual: 3B5T3 Decoded: 3B5T3 Confidence: 0.988514 Correct
Actual: 3EYHU Decoded: 3EYHU Confidence: 0.997517 Correct
Actual: 3HKSY Decoded: 3HKSY Confidence: 0.987876 Correct
Actual: 45V7U Decoded: 45V7U Confidence: 0.996639 Correct
Actual: 4BK54 Decoded: 4BK54 Confidence: 0.998016 Correct
Actual: 4EE3C Decoded: 4EE3C Confidence: 0.985467 Correct
Actual: 4R2WH Decoded: 4R2WH Confidence: 0.996247 Correct
Actual: 4RCX4 Decoded: 4RCX4 Confidence: 0.990553 Correct
Actual: 5LX29 Decoded: 5LX29 Confidence: 0.995281 Correct
Actual: 5RESL Decoded: 5RESL Confidence: 0.983701 Correct
Actual: 5SE4L Decoded: 5SE4L Confidence: 0.989864 Correct
Actual: 5SMWJ Decoded: 5SMWJ Confidence: 0.994678 Correct
Actual: 6V2EP Decoded: 6V2EP Confidence: 0.993554 Correct
Actual: 7KPZ3 Decoded: 7KPZ3 Confidence: 0.995650 Correct
Actual: 8285R Decoded: 8285R Confidence: 0.989046 Correct
```

```
Actual: 8HL84 Decoded: 8HL84 Confidence: 0.989704 Correct
Actual: 8K6GS Decoded: 8K6GS Confidence: 0.976224 Correct
Actual: 9LLZK Decoded: 9LLZK Confidence: 0.991375 Correct
Actual: A4BL7 Decoded: A4BL7 Confidence: 0.994456 Correct
Actual: AU4CX Decoded: AU4CX Confidence: 0.989085 Correct
Actual: AW94W Decoded: AW94W Confidence: 0.994157 Correct
Actual: AZX7V Decoded: AZX7V Confidence: 0.998466 Correct
Actual: B4ABZ Decoded: B4ABZ Confidence: 0.997231 Correct
Actual: BDKJC Decoded: BDKJC Confidence: 0.993727 Correct
Actual: BE3S7 Decoded: BE3S7 Confidence: 0.992351 Correct
Actual: BM9X8 Decoded: BM9X8 Confidence: 0.991259 Correct
Actual: C3MLK Decoded: C3MLK Confidence: 0.990415 Correct
Actual: C3TH2 Decoded: C3TH2 Confidence: 0.990501 Correct
Actual: C6X62 Decoded: C6X62 Confidence: 0.993211 Correct
Actual: C852Z Decoded: C852Z Confidence: 0.985060 Correct
Actual: CAZMA Decoded: CAZMA Confidence: 0.991499 Correct
Actual: CNDW7 Decoded: CNDW7 Confidence: 0.991150 Correct
Actual: CSSBK Decoded: CSSBK Confidence: 0.939221 Correct
Actual: DHUBH Decoded: DHUBH Confidence: 0.994851 Correct
Actual: DKGWG Decoded: DKGWG Confidence: 0.965679 Correct
Actual: E7XGT Decoded: E7XGT Confidence: 0.976799 Correct
Actual: EC6YM Decoded: EC6YM Confidence: 0.988651 Correct
Actual: EF3SP Decoded: EF3SP Confidence: 0.965904 Correct
Actual: EL2ZY Decoded: EL2ZY Confidence: 0.987320 Correct
Actual: EUYPS Decoded: EUYPS Confidence: 0.986720 Correct
Actual: F93UV Decoded: F93UV Confidence: 0.984464 Correct
Actual: FUSGP Decoded: FUSGP Confidence: 0.976917 Correct
Actual: FUTPC Decoded: FUTPC Confidence: 0.982576 Correct
Actual: FYUYM Decoded: FYUYM Confidence: 0.991917 Correct
Actual: GBAR3 Decoded: GBAR3 Confidence: 0.982097 Correct
Actual: GF3GD Decoded: GF3GD Confidence: 0.968476 Correct
Actual: GH25P Decoded: GH25P Confidence: 0.985166 Correct
Actual: GMLVU Decoded: GMLVU Confidence: 0.988636 Correct
Actual: GR2T4 Decoded: GR2T4 Confidence: 0.985169 Correct
Actual: GRY8X Decoded: GRY8X Confidence: 0.978917 Correct
Actual: GYPGJ Decoded: GYPGJ Confidence: 0.968171 Correct
Actual: HN94A Decoded: HN94A Confidence: 0.995723 Correct
Actual: HTJGD Decoded: HTJGD Confidence: 0.980425 Correct
Actual: HX72C Decoded: HX72C Confidence: 0.992150 Correct
Actual: J2LE3 Decoded: J2LE3 Confidence: 0.993463 Correct
Actual: J4KCB Decoded: J4KCB Confidence: 0.989296 Correct
Actual: JM9SZ Decoded: JM9SZ Confidence: 0.992039 Correct
Actual: JZ53E Decoded: JZ53E Confidence: 0.994268 Correct
Actual: K8LY2 Decoded: K8LY2 Confidence: 0.993956 Correct
Actual: KBVUK Decoded: KBVUK Confidence: 0.996933 Correct
Actual: KWPRN Decoded: KWPRN Confidence: 0.994383 Correct
Actual: L4HGA Decoded: L4HGA Confidence: 0.985420 Correct
Actual: LK4YE Decoded: LK4YE Confidence: 0.985264 Correct
Actual: LL6FJ Decoded: LL6FJ Confidence: 0.981353 Correct
Actual: M224L Decoded: M224L Confidence: 0.995365 Correct
Actual: M6AXZ Decoded: M6AXZ Confidence: 0.986886 Correct
Actual: M8H4U Decoded: M8H4U Confidence: 0.992628 Correct
Actual: MG8ES Decoded: MG8ES Confidence: 0.981723 Correct
Actual: NU8JM Decoded: NU8JM Confidence: 0.995276 Correct
Actual: NYPU3 Decoded: NYPU3 Confidence: 0.986068 Correct
Actual: P2N4P Decoded: P2N4P Confidence: 0.996516 Correct
Actual: R32WR Decoded: R32WR Confidence: 0.994478 Correct
Actual: RDPL2 Decoded: RDPL2 Confidence: 0.985456 Correct
Actual: RNWBS Decoded: RNWBS Confidence: 0.997015 Correct
Actual: RP5GX Decoded: RP5GX Confidence: 0.988254 Correct
Actual: S9FES Decoded: S9FES Confidence: 0.982137 Correct
Actual: TBLU2 Decoded: TBLU2 Confidence: 0.999699 Correct
Actual: TGBSX Decoded: TGBSX Confidence: 0.982786 Correct
Actual: U3V4C Decoded: U3V4C Confidence: 0.990577 Correct
Actual: UE64X Decoded: UE64X Confidence: 0.987239 Correct
Actual: UWUCH Decoded: UWUCH Confidence: 0.991729 Correct
Actual: W4GKY Decoded: W4GKY Confidence: 0.985852 Correct
Actual: W9B2K Decoded: W9B2K Confidence: 0.993492 Correct
Actual: XA3FW Decoded: XA3FW Confidence: 0.993659 Correct
Actual: XCERZ Decoded: XCERZ Confidence: 0.987457 Correct
Actual: XKSLN Decoded: XKSLN Confidence: 0.993031 Correct
Actual: XWET3 Decoded: XWET3 Confidence: 0.986135 Correct
Actual: YAWG2 Decoded: YAWG2 Confidence: 0.989535 Correct
Actual: YBAUK Decoded: YBAUK Confidence: 0.994175 Correct
Actual: YRN26 Decoded: YRN26 Confidence: 0.992761 Correct
Actual: YT4TZ Decoded: YT4TZ Confidence: 0.987157 Correct
Actual: Z69AB Decoded: Z69AB Confidence: 0.991632 Correct
Actual: ZXTRV Decoded: ZXTRV Confidence: 0.996448 Correct
Actual: ZYZBH Decoded: ZYZBH Confidence: 0.984823 Correct
Elapsed time is 6.790168 seconds.
Character Accuracy: 1.000000
HIP Accuracy: 1.000000
Average confidence: 0.988372
Minimum confidence: 0.939221
```

B.4 Template Correlations

```
Actual: 27LP5 Decoded: 27LP5 Confidence: 0.825181 Correct
Actual: 2CAD2 Decoded: 2CAD2 Confidence: 0.975870 Correct
Actual: 2GX7T Decoded: 2GX7T Confidence: 0.979121 Correct
Actual: 2MR5A Decoded: 2MR5A Confidence: 0.928278 Correct
Actual: 2YGS4 Decoded: 2YGS4 Confidence: 0.970611 Correct
Actual: 373MH Decoded: 373MH Confidence: 0.975391 Correct
Actual: 3B5T3 Decoded: 3B5T3 Confidence: 0.933198 Correct
Actual: 3EYHU Decoded: 3EYHU Confidence: 0.994478 Correct
Actual: 3HKSY Decoded: 3HKSY Confidence: 0.970833 Correct
Actual: 45V7U Decoded: 45V7U Confidence: 0.947059 Correct
Actual: 4BK54 Decoded: 4BK54 Confidence: 0.956189 Correct
Actual: 4EE3C Decoded: 4EE3C Confidence: 0.972240 Correct
Actual: 4R2WH Decoded: 4R2WH Confidence: 0.964177 Correct
Actual: 4RCX4 Decoded: 4RCX4 Confidence: 0.980775 Correct
Actual: 5LX29 Decoded: 5LX29 Confidence: 0.951461 Correct
Actual: 5RESL Decoded: 5RESL Confidence: 0.794538 Correct
Actual: 5SE4L Decoded: 5SE4L Confidence: 0.934453 Correct
Actual: 5SMWJ Decoded: 5SMWJ Confidence: 0.805682 Correct
Actual: 6V2EP Decoded: 6V2EP Confidence: 0.985146 Correct
Actual: 7KPZ3 Decoded: 7KPZ3 Confidence: 0.973909 Correct
Actual: 8285R Decoded: 8285R Confidence: 0.934977 Correct
Actual: 8HL84 Decoded: 8HL84 Confidence: 0.972920 Correct
Actual: 8K6GS Decoded: 8K6GS Confidence: 0.959971 Correct
Actual: 9LLZK Decoded: 9LLZK Confidence: 0.981355 Correct
Actual: A4BL7 Decoded: A4BL7 Confidence: 0.679270 Correct
Actual: AU4CX Decoded: AU4CX Confidence: 0.981999 Correct
Actual: AW94W Decoded: AW94W Confidence: 0.933406 Correct
Actual: AZX7V Decoded: AZX7V Confidence: 0.995734 Correct
Actual: B4ABZ Decoded: B4ABZ Confidence: 0.992869 Correct
Actual: BDKJC Decoded: BDKJC Confidence: 0.981906 Correct
Actual: BE3S7 Decoded: BE3S7 Confidence: 0.980886 Correct
Actual: BM9X8 Decoded: BM9X8 Confidence: 0.976996 Correct
Actual: C3MLK Decoded: C3MLK Confidence: 0.984983 Correct
Actual: C3TH2 Decoded: C3TH2 Confidence: 0.982591 Correct
Actual: C6X62 Decoded: C6X62 Confidence: 0.992224 Correct
Actual: C852Z Decoded: C852Z Confidence: 0.932864 Correct
Actual: CAZMA Decoded: CAZMA Confidence: 0.988927 Correct
Actual: CNDW7 Decoded: CNDW7 Confidence: 0.961476 Correct
Actual: CSSBK Decoded: CSSBK Confidence: 0.957649 Correct
Actual: DHUBH Decoded: DHUBH Confidence: 0.985930 Correct
Actual: DKGWG Decoded: DKGWG Confidence: 0.939548 Correct
Actual: E7XGT Decoded: E7XGT Confidence: 0.958905 Correct
Actual: EC6YM Decoded: EC6YM Confidence: 0.981291 Correct
Actual: EF3SP Decoded: EF3SP Confidence: 0.949183 Correct
Actual: EL2ZY Decoded: EL2ZY Confidence: 0.971211 Correct
Actual: EUYPS Decoded: EUYPS Confidence: 0.979362 Correct
Actual: F93UV Decoded: F93UV Confidence: 0.968414 Correct
Actual: FUSGP Decoded: FUSGP Confidence: 0.971808 Correct
Actual: FUTPC Decoded: FUTPC Confidence: 0.971874 Correct
Actual: FYUYM Decoded: FYUYM Confidence: 0.978115 Correct
Actual: GBAR3 Decoded: GBAR3 Confidence: 0.953717 Correct
Actual: GF3GD Decoded: GF3GD Confidence: 0.959121 Correct
Actual: GH25P Decoded: GH25P Confidence: 0.933124 Correct
Actual: GMLVU Decoded: GMLVU Confidence: 0.987956 Correct
Actual: GR2T4 Decoded: GR2T4 Confidence: 0.976689 Correct
Actual: GRY8X Decoded: GRY8X Confidence: 0.963954 Correct
Actual: GYPGJ Decoded: GYPGJ Confidence: 0.966630 Correct
Actual: HN94A Decoded: HN94A Confidence: 0.989126 Correct
Actual: HTJGD Decoded: HTJGD Confidence: 0.966547 Correct
Actual: HX72C Decoded: HX72C Confidence: 0.986825 Correct
Actual: J2LE3 Decoded: J2LE3 Confidence: 0.986353 Correct
Actual: J4KCB Decoded: J4KCB Confidence: 0.981501 Correct
Actual: JM9SZ Decoded: JM9SZ Confidence: 0.980201 Correct
Actual: JZ53E Decoded: JZ53E Confidence: 0.941834 Correct
Actual: K8LY2 Decoded: K8LY2 Confidence: 0.981948 Correct
Actual: KBVUK Decoded: KBVUK Confidence: 0.991951 Correct
Actual: KWPRN Decoded: KWPRN Confidence: 0.958539 Correct
Actual: L4HGA Decoded: L4HGA Confidence: 0.981440 Correct
Actual: LK4YE Decoded: LK4YE Confidence: 0.970474 Correct
Actual: LL6FJ Decoded: LL6FJ Confidence: 0.959527 Correct
Actual: M224L Decoded: M224L Confidence: 0.990120 Correct
Actual: M6AXZ Decoded: M6AXZ Confidence: 0.679171 Correct
Actual: M8H4U Decoded: M8H4U Confidence: 0.982491 Correct
Actual: MG8ES Decoded: MG8ES Confidence: 0.975059 Correct
Actual: NU8JM Decoded: NU8JM Confidence: 0.979278 Correct
Actual: NYPU3 Decoded: NYPU3 Confidence: 0.969942 Correct
Actual: P2N4P Decoded: P2N4P Confidence: 0.988291 Correct
Actual: R32WR Decoded: R32WR Confidence: 0.958942 Correct
Actual: RDPL2 Decoded: RDPL2 Confidence: 0.964383 Correct
Actual: RNWBS Decoded: RNWBS Confidence: 0.966319 Correct
Actual: RP5GX Decoded: RP5GX Confidence: 0.944119 Correct
Actual: S9FES Decoded: S9FES Confidence: 0.964909 Correct
Actual: TBLU2 Decoded: TBLU2 Confidence: 0.999377 Correct
Actual: TGBSX Decoded: TGBSX Confidence: 0.978971 Correct
Actual: U3V4C Decoded: U3V4C Confidence: 0.986671 Correct
Actual: UE64X Decoded: UE64X Confidence: 0.966077 Correct
```

```
Actual: UWUCH Decoded: UWUCH Confidence: 0.964454 Correct
Actual: W4GKY Decoded: W4GKY Confidence: 0.955301 Correct
Actual: W9B2K Decoded: W9B2K Confidence: 0.955179 Correct
Actual: XA3FW Decoded: XA3FW Confidence: 0.959456 Correct
Actual: XCERZ Decoded: XCERZ Confidence: 0.977761 Correct
Actual: XKSLN Decoded: XKSLN Confidence: 0.984414 Correct
Actual: XWET3 Decoded: XWET3 Confidence: 0.943594 Correct
Actual: YAWG2 Decoded: YAWG2 Confidence: 0.959316 Correct
Actual: YBAUK Decoded: YBAUK Confidence: 0.981986 Correct
Actual: YRN26 Decoded: YRN26 Confidence: 0.977572 Correct
Actual: YT4TZ Decoded: YT4TZ Confidence: 0.977199 Correct
Actual: Z69AB Decoded: Z69AB Confidence: 0.978042 Correct
Actual: ZXTRV Decoded: ZXTRV Confidence: 0.990738 Correct
Actual: ZYZBH Decoded: ZYZBH Confidence: 0.967960 Correct
Elapsed time is 7.277254 seconds.
Character Accuracy: 1.000000
HIP Accuracy: 1.000000
Average confidence: 0.959318
Minimum confidence: 0.679171
```