Problem 1

Homework required one to design an algorithm to recognize the provided grayscale CAPTCHA based on provided training set. Three separate stages can be deduced

1. Training set feature extraction.
2. CAPTCHA character isolation.
3. Character recognition.

In the following sections, I will walk through the trial-and-error process I had gone through.

Trial 1

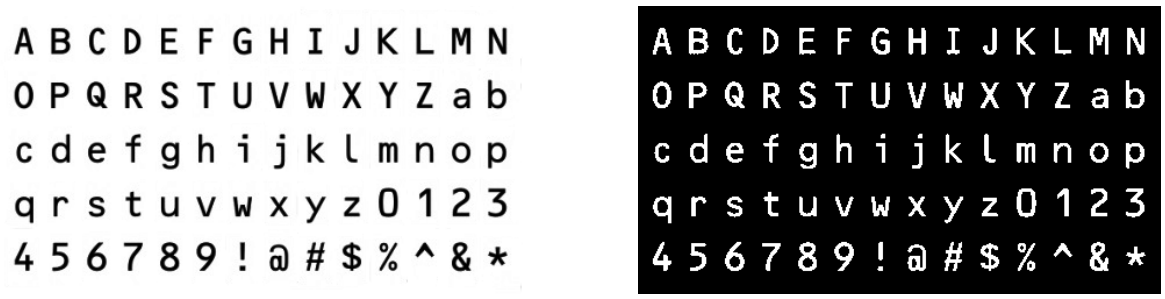
Training set feature extraction

We are interested in the characters themselves, a mask with ones on the characters is closer to expectation. Trivial observation indicates no significant noise exists, and gray level is marginal to non-existence along character edges, we can try to threshold it directly by averaging global minimum and maximum.

$$T = \sum_i \sum_j \frac{max(I(i,j)) - min(I(i,j))}{2} \tag{1}$$

Due to the fact that `TrainingSet.raw` is provided as black-on-white style, thresholding criteria is reversed.

$$I(i,j) = \begin{cases} 1, I(i,j) < T \\ 0, I(i,j) \geq T \end{cases} \tag{2}$$


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 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 0 1 2 3
4 5 6 7 8 9 ! @ # $ % ^ & *

(a) Grayscale (b) Binary

Figure 1: Pre-processed training set

The training set characters are displayed in montage, we need to isolate them to individual characters before proceeding to next step. The montage contains 14-by-5 characters, with consistence spacing between horizontal and vertical neighbors. We can easily calculate the appropriate spacing to separate them. Margins of the image can causes misalignment after cropping, margin size of 5 and 7 pixels are empirically chose for horizontal and vertical edges respectively.

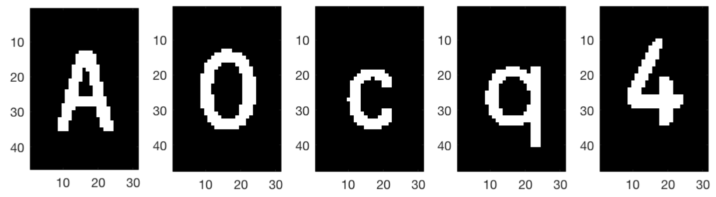


Figure 2: Cropped characters, first column is shown.

To perform further processing, I choose to resize them to image of size 32-by-32 pixels to ease the process.

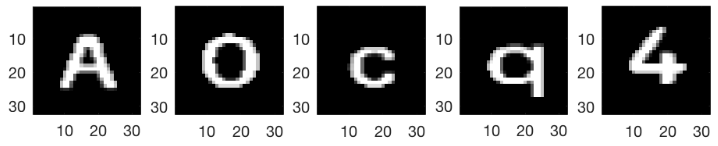
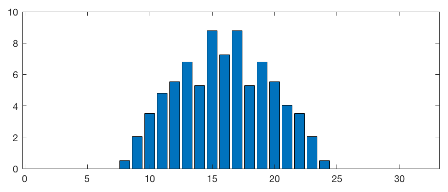


Figure 3: Resized characters to square, first column is shown.

Characters seem to have enough differences from scanline perspective, therefore, I project sum each character along horizontal and vertical direction to derive their 2-by-32 feature vector in Figure 4.


(a) X direction

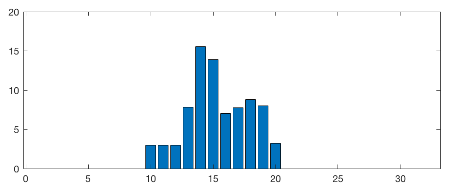
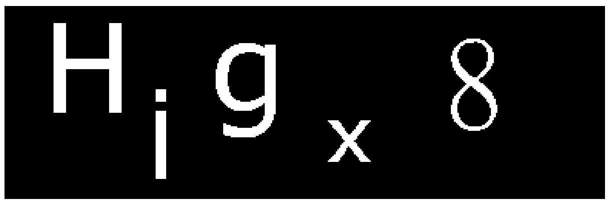

(b) Y direction

Figure 4: Cumulative sum projection of A

CAPTCHA Character isolation

Both sample images are easy to process as the training set.


(a) Sample 1

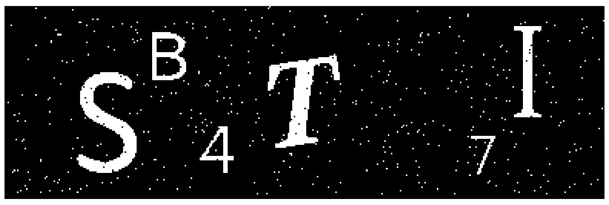
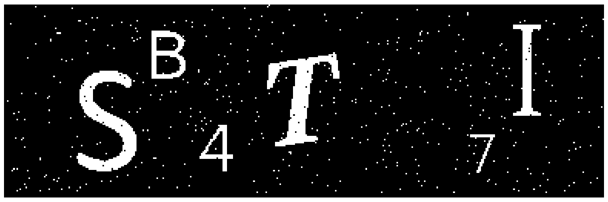

(b) Sample 2

Figure 5: Binary sample images

Sadly, `sample2.raw` contains unwanted speckle noises, visualized in Figure 5b. Therefore, median filter is applied specially for it. Kernel size of 3 is arbitrarily chosen by trials.


(a) Before

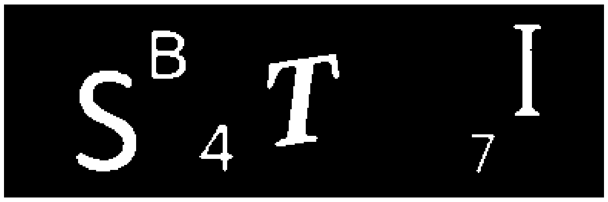
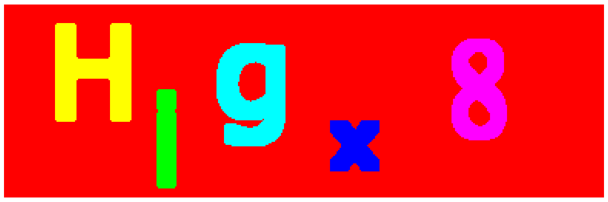

(b) After

Figure 6: Median filter for `sample2.raw`

For efficient labeling of connected components, disjoint-set algorithm based on handouts of Duke University *CS100e: Program Design and Analysis II* (`https://www2.cs.duke.edu/courses/cps100e/fall09/notes/UnionFind.pdf`) is used. However, characters such as `i` will be labeled as separate components, hence, dilation is first performed prior to the labeling, and masked out dilated regions later on. Dilation kernel size is empirically chosen to be 7. 8-connected component is used.


(a) Regions dilated

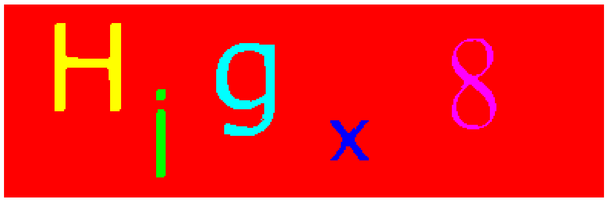

(b) Masked result

Figure 7: Character labeling (`sample1.raw`)

Using labels, we can crop out each characters and sort them according to their horizontal position, implicit requirement for English reading convention.

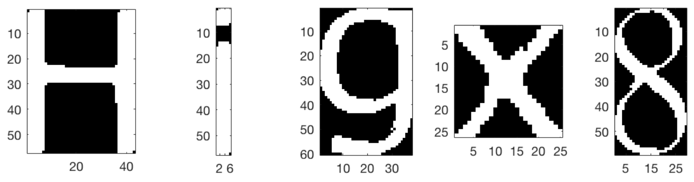
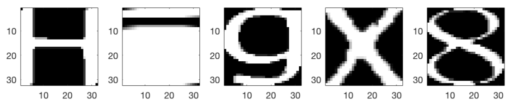


Figure 8: Cropped characters

At last, characters are resized to 32-by-32 squares as well, same goes for `sample2.raw`.


(a) `sample1.raw`

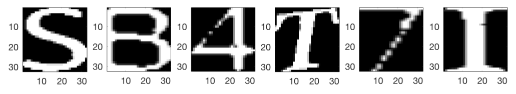

(b) `sample2.raw`

Figure 9: Resized characters

Character recognition

Directly subtract the differences between projection sum, I can retrieve the difference between curves, and use it as an index to find out which character is the most similar.

$$\begin{cases} d_x = \sum |\vec{c_x} - \vec{t_x}|^2 \\ d_y = \sum |\vec{c_y} - \vec{t_y}|^2 \end{cases} \tag{3}$$

where $\vec{c}$ is the feature character of CAPTCHA characters, while $\vec{t}$ is the feature character of training template. d_x and d_y are scalars. All the vectors are normalized prior to operations. A unified distance metric is used

$$d = \sqrt{d_x^2 + d_y^2} \tag{4}$$

Evaluation

100% error on both image.

Trial 2

Training set feature extraction

After reviewing the source code, I realize training set did not stretch the characters. Therefore

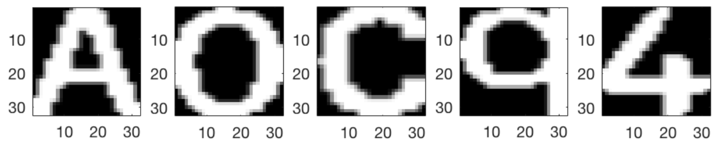


Figure 10: Stretched

Evaluation

90% average error rate. Only `H` is correctly recognized in `sample1.raw`. Apparently, direct curve comparison is not feasible.

Trial 3

Character recognition

If we accumulate the projection result again, then the normalization to acquire the distance metric is essentially calculating the cumulative distribution function for pixel intensity along different direction (horizontal and vertical). By treating the problem as a binary classifier system, we can utilize the receiver operating characteristic (ROC) curve, but instead of favoring true positive direction, which requires maximizing the ROC curve area over $y = x$ line, we try to *minimize* that area, since same character should not deviate its distribution along any axis.

Feature vectors along both axis are now treated as sample points along X and Y direction, which are linearly interpolated to uniform X grid of size 32. The rationale of 32 is due to the resampled image size, applied earlier, is 32-by-32.

We can formulate the calculation of ROC area as

$$A_{ROC} = \sum |\vec{c} - \vec{t}| \tag{5}$$

Evaluation

`sample1.raw` now has 60% error rate (`H^gSD`). `sample2.raw` now has 83% error rate (`209tYI`). Average error rate 72%. If we can loosen the restriction and treat upper and lower cases as the same, then average error rate drop to 64%.

Trial 4

Character recognition

After reading Wikipedia about template matching, I decided to drop the sum projection shit and use sum of absolute differences (SAD).

$$SAD(i,j) = \sum_i \sum_j |I(i,j) - T(i,j)| \tag{6}$$

where I is the sampled CAPTCHA character, while T is the training set template. No more fancy processing, not even correlation

Evaluation

`sample1.raw` now has 20% error rate (`H&gXS`). `sample2.raw` now has 0% error rate (`SB4T7I`). Average error rate 10%.