Assignment - 2

Alok Kumar **210101**

Paridhi Vaish **210700**

Abstract

To design an algorithm that can take a set of images and return the parity of the hexadecimal number in that image.

1 Model Description

Given:

- Size of training images : 500×100 pixels
- Size of reference images : 100×100 pixels
- Each image contains a code that is a 4 digit hexadecimal number. The font of all these characters are the same, as is the font size. The Latin characters A-F is always in upper case.
- However, each character may be rotated (degree of rotation is always either $0, \pm 10, \pm 20, \pm 30$ degrees and each character may be rendered with a different color.
- The background color of each image may vary. However, all background colors are light in shade.
- Each image also has some stray lines in the background which are of varying thickness, varying color and of a shade darker than that of the background.

This problem can be broadly classified as a multi-class classification problem where we aim to recognize the character by training our model on a set of reference images, and hence find its parity.

To prove that the parity of a hexadecimal number depends only on the parity of its last character, we can use modular arithmetic.

Let's consider a hexadecimal number, represented as $x = x_n x_{n-1} \ldots x_1 x_0$, where x_i represents the ith digit of the number and n is the total number of digits. Rewriting it in decimal representation :

$$x = (16^n x_n + 16^{n-1} x_{n-1} + \ldots + 16 x_1) \times 16 + (x_0) = y \times 16 + (x_0),$$

where $y = 16^n x_n + 16^{n-1} x_{n-1} + \ldots + 16 x_1$ is an integer.

This can be rewritten as :

$$x = 2 \times (y \times 8 + (x_0/2))$$

x is even only if $x_0/2$ is an integer. Which means x_0 should be an even number.

Therefore, we have shown that if the last digit of a hexadecimal number is even, then the entire number is even, and if the last digit is odd, then the entire number is odd. Thus, the parity of a hexadecimal number depends only on the parity of its last digit.

Preprint. Under review.

We have implemented the Subtraction Algorithm for this case due to the existence of reference characters and being relatively easy than other CAPTCHAs out there.

Subtraction: Two images with same dimensionality are superimposed and corresponding pixels are subtracted in the vector matrix. In the resulting matrix, lesser the sum of elements in matrix, lesser the difference and hence, more the resemblance.

The algorithm we are using is Subtraction whose Mean Absolute Error(MAE) is defined as

$$\frac{1}{No.\ of\ Pixels} \sum_{D} Sum\ of\ pixels(D)$$

where D is the Subtraction Matrix. Lesser the MAE, more accurately the model predicts the parity of the hexadecimal number.

2 Algorithm Description

Step 1: Pre-processing of train and reference images

- **Monochrome** : All images are converted to gray-scale and a further conversion to monochrome by a threshold offered by *Otsu's Method*.
- **Inversion** : Images are inverted so that the background black matches with extra black pixels introduced around the corners due to rotation.
- **Erosion** : This is used to reduce the thickness of lines present in the image. As a result, the obstructing lines are almost removed. The thickness of letters also got reduced but they are still good enough to be further used.
- **Rotation** : 7 copies of each reference character are made with rotation angles of $-30, -20, -10, 0, 10, 20, 30$ degrees and stored as new reference images in an array. These are used for comparison on the training/testing images.

Step 2: Cropping and Comparison with Reference Set

- **Cropping** : Obtained test image is cropped using a manually tuned parameter in order to extract last character.
- Dimensionality of images are made consistent through appropriate cropping of reference images.
- Each image is then compared with each of 16×7 reference images through the process of **'Subtraction'**.

Step 3: Correct classification based on Subtraction matrix

- Each test image is compared with 16×7 reference images and sum of elements of difference matrix(D) is stored in a new matrix.
- Least element of the new matrix gives us the most resembled character and the algorithm outputs the same.

Step 4: Parity detection

After successful identification of last character, parity of the last character is found by using the modulo operator which gives us the parity of the hexadecimal number itself.

3 Hyper-Parameter Tuning

"Selecting an optimal value for a hyper-parameter is often considered an art rather than a science."

Hence, there exists no fixed algorithm to find out the best hyper-parameters. Instead, it depends on its utility in the Machine Learning model .

The hyper-parameters used in our attempt are as follows:

1. Monochrome Threshold
2. Eroding Thickness
3. Cropping Last Character
4. Horizontal Cropping of Reference Images
5. Vertical Cropping of Reference Images
6. Resizing Parameter

3.1 MonochromeThreshold

Firstly, the images are converted to gray-scale to remove the unnecessary information of colors. Then, the images are converted to monochrome using a thresholding parameter, which is determined using the *Otsu's Method.*

3.2 Eroding Thickness

The erosion parameter is the kernel size. Kernel is the structuring element that determines the precise effect of the erosion on the input image. A too low size would not affect at all while a too large kernel size erodes the boundaries of the characters too. A kernel size of 2 gave optimum results as to eroding obstructing lines as well as maintaining legible thickness of characters in image.

3.3 Cropping Last Character

For obtaining the last character which is crucial to parity generation, we crop the test image with a boundary of $10 : 100 \times 360 : 450$ pixels. This was found manually. We used matplotlib.pyplot to display the image in an XY plot. We found the right X and Y axes limits which bounded a straight image closely, $378 : 432$ and $18 : 92$ respectively(refer Fig.1). Then, cropped it with an offset of 18 horizontally on both sides and an offset of 8 vertically above and below so that the boundary could hold the character even if it was rotated to ± 30 degrees (refer Fig.2). This results in a 90×90 image, with boundaries $10 : 100 \times 360 : 450$ pixels.

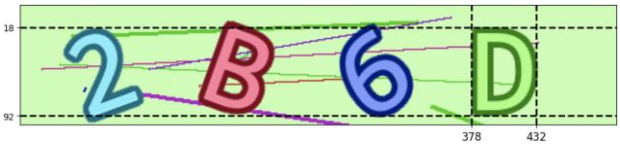


Figure 1: Straight Image Boundary

Figure 2: Boundary chosen to Enclose $\pm 30^{\circ}$ Rotated ones

3.4 Horizontal Cropping of Reference Images

First limitation is that the reference image should also be 90×90 for subtraction. We cropped the image instead of resizing it from 100×100 to 90×90, so that the resolution of image doesn't change. The possible horizontal cropping limits are from $0 : 90$ to $10 : 100$. We have tuned this as a hyper-parameter and found that $5 : 95$ gave 100% accuracy in parity detection.

3.5 Vertical Cropping of Reference Images

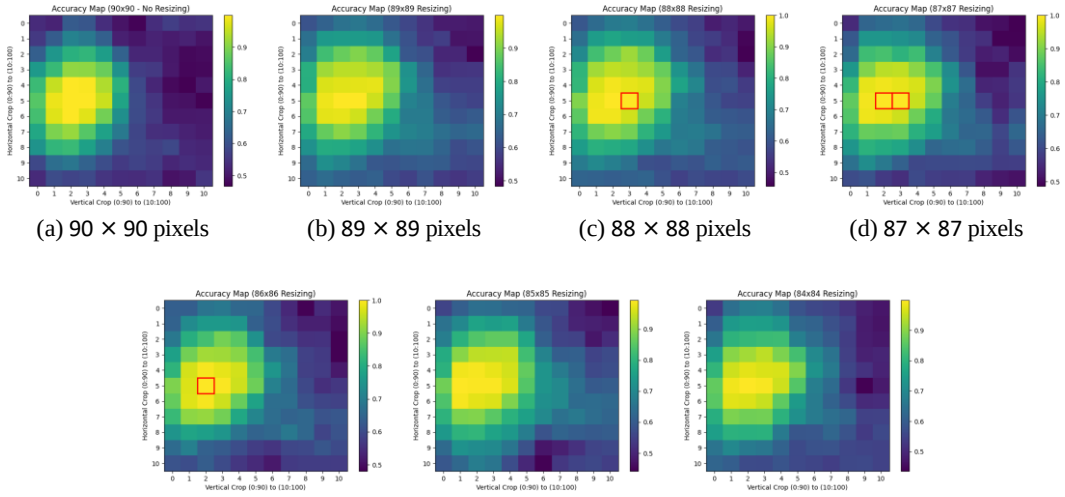
The possible vertical cropping limits are from $0 : 90$ to $10 : 100$. We have tuned this as a hyper-parameter and found that $3 : 93$ gave 100% accuracy in parity detection.

3.6 Resizing Parameter

We resized both test and reference images to the same values after cropping them both to 90×90 matrices. We tried to resize the images initially, to a smaller value say, 50×50 pixels, to reduce computing time and found a significant drop in accuracy. So, we tried tuning it as a hyper-parameter.

The below heat maps show the Parity Accuracy w.r.t to the resizing hyper-parameter and different cropping hyper-parameters of the reference images. So, from Fig.3(c),(d),(e), we chose the best horizontal crop as $5 : 95$. From Fig.3(d), we chose the best vertical crop as $3 : 93$ and the resizing limit as 87×87 pixels.

Hyper-Parameter Tuning



(a) 90×90 pixels (b) 89×89 pixels (c) 88×88 pixels (d) 87×87 pixels

(e) 86×86 pixels (f) 85×85 pixels (g) 84×84 pixels

Figure 3: X axis and Y axis represent the vertical crop offset and the horizontal crop offset of the reference images respectively. The red blocks show 100% accuracy in parity detection.

4 Conclusion

Based on the above images and fine-tuning of various hyper-parameters, we have reached the following conclusions:

- Using pictures in gray-scale and turning them into monochrome resulted in better identification and reduced the execution time around $3 - 4$ milliseconds.
- Optimized sizes of respective images were found through manually looking at the training image and fine-tuning the hyper-parameters for reference images.
- Subtraction method provided an appreciable parity accuracy.

Through the optimization process outlined above, our model achieved a **parity match score of** 1.0 (accuracy of 100%) within an execution time of 7 **milliseconds** per image when 2000 images were used. (subject to change because the first iteration takes around 60 milliseconds due to reference image pre-processing).

References

[1] OpenCV Tutorial for Erosion and Image Morphology

[2] Documentation for *Otsu's method*

[3]Algorithm for Subtraction of Image Matrix