CS771 Assignment 2

Team PAC
Group No.23

Chetanya Bhan	**Nikhil Meena**	**Paturi Bhavya**	**Somya Gupta**
210283	210667	210712	211049

1 Approach

We utilized Logistic Regression, a linear model available in the SciKit Learn Python library, to solve this problem. Each image is processed and simplified before feeding to the machine learning model. We realised that determining the parity of the last letter in each image, which represents a 4-letter hexadecimal code, is sufficient to determine whether the number is odd or even.

This is because in base 10, a hexadecimal number like ABCD is expressed as:
$\mathrm{A}\times 16^3 + B \times 16^2 + C \times 16 + D$

The first three terms, being multiples of 16, are always even. Therefore, the parity of the sum is solely determined by the parity of the last term.

To isolate the last letter of the code, each training image [100x500] is vertically cropped, considering only the portion from pixel 358 to pixel 458. We identify and store the background color, as well as the pixels that correspond to the background. Then, we convert the image to grayscale using an OpenCV function. Afterward, we set the stored background pixels to white. We apply Otsu thresholding to convert the image into black and white, which further enhances the important details while losing some obfuscating lines. The processed image information is stored in an array and flattened into a 1D array, which is then used to train a Logistic Regression model from SciKit Learn. Finally, the trained model is saved as a .joblib file for future use. We had also used dilate and erode methods, as suggested in the pdf, after carrying out Otsu thresholding. Dilation (kernel size 4) followed by Erosion (kernel size 3) worked satisfactorily well in removing obfuscating lines. But as the model developed further, it could produce 100% accuracy without the Dilation and Erosion processes and so, we did not include them in the final solution.

1.1 Extracting Background Colour

As mentioned in the assignment pdf, the position of the pixels of background colour have been extracted by comparing the corner colours by additionally comparing the values if the obfuscating lines are the the corners as well,

And, if all of the corner checks fail due to obfuscating lines being present at all the corners an extra compare function has been added which will find the background colour using the maximum frequency of the colour that is present in the image.

1.2 Otsu Thresholding

Otsu's thresholding method arrives at a best threshold value to convert grayscale images into monochrome. For each threshold value, a measure of spread of pixels falling on either sides of the

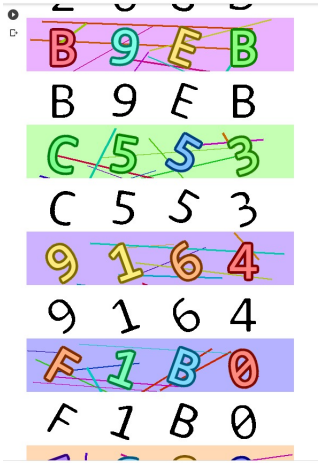
Preprint. Under review.

threshold, the foreground and background. The sum of the foreground and background pixels is minimised to arrive at the best threshold and accordingly the image is converted into monochrome.

The thresholding of image has been done to remove the unwanted colour in the image and to make the training testing data more similar (i.e. to reduce the inconsistencies in the brightness of the images).

The best thresholding value that worked for all the training images is 120 to convert to 255 and rest to 0.

The image processing steps produced results as shown. The processed images were much simpler to learn by linear ML models like Logistic Regression.



2 Logistic Regression- The Model

Logistic regression is a supervised machine learning algorithm used mostly for classification problems. The algorithm first carries out a linear regression. The output, which is continuous in nature, is then processed by a logistic function (sigmoid) function and converted into a predicted probability (value lying in between 0 and 1). The model is described by the following equation-

$$p(X; b, w) = \frac{1}{1 + e^{-w \cdot X + b}} \tag{1}$$

3 Training Algorithm

We us the scikit learn method .fit() to train the algorithm. Training algorithm used for Logistic Regression model is the negative log likelihood function, or the log loss function. p_k== p(x_k) where p_k are probabilities that $y_k$will be 1. Thus log loss of kth point is -ln p_k if y_k =1 and it is -ln(1-p_k) if y_k =0. The sum, the total loss, is the overall negative log-likelihood -l. Best fit is obtained at w and b, where -l is minimized, or equivalently maximize the log-likelihood, which means maximizing the likelihood function itself. This method is known as maximum likelihood estimation.

4 Hyperparameters

Used default Hyperparameters of sklearn library and gave 100% accuracy. So, we kept them as default. The default hyperparameters used in .fit() method are batch_size=32, nb_epoch=10 and verbose=1. batch_size is the number of samples per gradient update. nb_epoch is the number of times to iterate over the training data. verbose stands for verbosity and verbose=1 stands for verbose. verbose=0 is silent. verbose=3 stands for one log line per epoch. We also used a value of max$_iter = 100 while using the Logistic Regression function$.

5 Validation

Accuracy obtained on a 90:10 test-train dataset split was 100%. Because of the high accuracy, the model to be submitted was trained over all 2000 images. On a testing size of 2000 images time taken per image is 0.004 seconds with Parity match score 1.0.

6 Other Approaches

Another approach for solving the problem can be to not use any ML model and rather use image processing to make the predictions since the CAPTCHA images that have been given to us are of fixed pattern and do not change their relative position.

Initially store the reference images their rotated versions sequentially(i.e. 0-56 for even the rest for odd, just for faster computation) in a numpy array so that we can compare them later.

Remove the background of the image using similar step as done in previous method.

Using hit and trial doing bitwise AND operation between training and testing image we found out that the image is centered around between 358-458 pixels don't need to be scaled to fit the reference image.

Also, the training image is shifted a bit vertically by 5 pixels. After fixing its position it can be directly compared with the reference images.

We initially know that with which image we're comparing our image with so we can keep track of the image with which our image matches the most.

To keep the operation faster more meaningful instead of subtracting the image we used more efficient bitwise XOR operation between images. As it clears out the possibility of counting the obfuscating lines while also performing the comparision between the images.

Since, we've kept record of the image which matches the most we can add an additional check function in the end to append the results in the array. i.e. if(ans[1]>55): l.append("ODD") else: l.append("EVEN")

We got 100% accuracy using this method & time taken per image was 0.0034 seconds. Still, we used the Logistic regression model because the size increased due to additional reference images and the time taken was little inconsistent.

We had also trained a model using K Nearest Neighbours Classifier following the same image processing methods. But the time taken was 0.006 per image and so we chose the Logistic Regression model over it.

7 Conclusion

We have explored solutions utilizing simple machine learning models and another solution which does not utilize a machine learning model. After improving the code to make room for faster predictions, the solutions seemed to take almost the same prediction time. So we are submitting the solution involving a linear model, logistic regression, which produced results of 0.004 second time taken per image and a parity match score of 1.0. The trained model is submitted as a .joblib file of size 81 kB, much smaller than the file sizes of the other solutions. Moreover the time taken by the non-ML solution was inconsistent. So we opted for the Logistic Regression model over the non-ML model and other ML models.

Table 1: Results

Model	Train Time per Image
Non-ML model	0.0034 sec (inconsistent)
Logistic Regression	0.0042 sec
kNN Classifier	0.0061 sec

8 References

[1]http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html#:~:text=Otsu%27s%20thresholding%20method%20involves%20iterating%20through%20all%20the,foreground%20and%20background%20spreads%20is%20at%20its%20minimum.
[2] https://www.geeksforgeeks.org/understanding-logistic-regression/
[3] https://www.geeksforgeeks.org/erosion-dilation-images-using-opencv-python/
[4]] https://en.wikipedia.org/wiki/Logistic_regression/
[5]] https://stackoverflow.com/questions/37973005/what-do-model-predict-and-model-fit-do