CS771 Introduction to Machine Learning Assignment 3

Gunj Mehul Hundiwala
22111024
gunjmehul22@iitk.ac.in

Kartick Verma
22111029
kartickv22@iitk.ac.in

Kush Shah
22111033
kushshah22@iitk.ac.in

Raj Kumar
22111050
rajkumar22@iitk.ac.in

Saqeeb
22111053
saqeeb22@iitk.ac.in

1 Algorithm description

A straightforward approach to find the characters from CAPTCHA in the provided dataset is by feeding them straight into a model like logistic regression, CNN, SVM, but it would make the procedure excessively expensive (model size would be large as well) and the accuracy will be poor. So, we opted to pre-process the CAPTCHA image by isolating the background and stray lines from the foreground characters. Then, after segmenting the provided picture into individual characters, we fed the resultant characters into a logistic regression for classification.

A strong linear model was obviously our preferred option because of the numerous character orientations in the image and the variety of backdrops, and it performed well on the provided dataset as well. The complete procedure is described in depth in the parts that follow:

1.1 Pre-Processing Phase

The images given (Fig1) were in RGB format having obfuscation lines which makes them hard to segment individual characters.

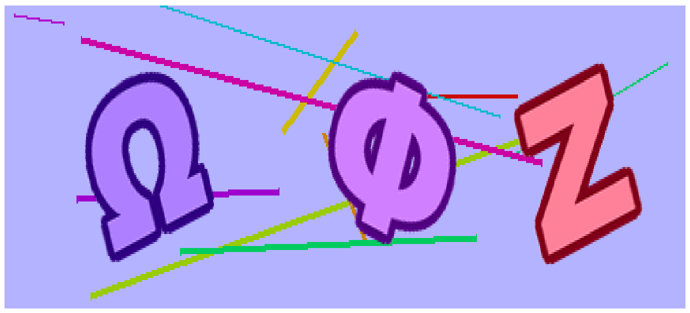

Figure 1: Sample image

The subsequent steps were adopted to render a picture appropriate for segmentation (removal of background and obfuscation lines):

- In order to learn the colour of the background, we first found out the corner pixels and assumed it to be our background. To avoid the condition where the obfuscation line lies on the corner, we find the pixel of all four corners and find the pixel which is appearing most frequently in corners. And assumes it to be our background.

To nullify the background pixel we subtracted the colour from the image by simply replacing it with white colour i.e [255,255,255] (Fig2)

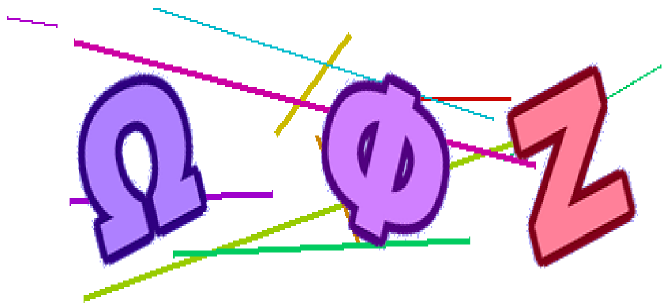

Figure 2: Background Extracted

Here we also neglect the case in which an obfuscation line crosses all four corners as it may appear very rarely (as given in the problem also).

We also try to find the pixel which is having max frequency throughout the image as it will be our background. But finding background by this approach will be very slow thus affecting our performance.

- Dilate (1) the image using cv2.dilate() (3) function using a 5x5 kernel for 1 iterations, here we tend use a standard 5x5 kernel and single iterations to be dilate was found by random observation of 10 images for kernel size 3x3, 4x4, 5x5. From the dry runs we found 5x5 (size of kernel) to be appropriate for removal of obfuscation lines while at the same time preserving the characteristics of the individual letters. (Fig3)

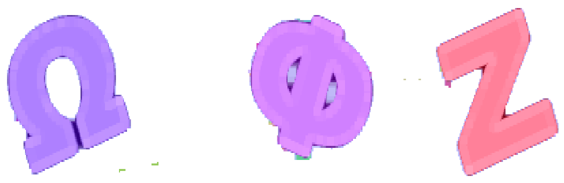

Figure 3: Obsfuscation lines removed

- Convert the image to gray-scale (5) to convert each pixel in the range 0 to 255, where 0 denotes black colour and 255 denotes white. (Fig4)

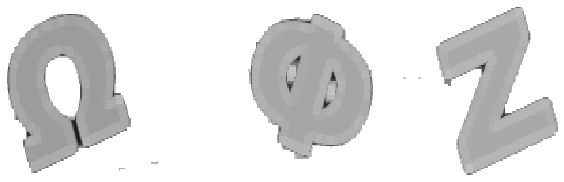

Figure 4: Gray-Scaled

1.2 Segmentation Phase

The process we used to segment the image into individual characters is described below (4). (Fig5) show the individual character segments identified by our algorithm after segmentation.

- Since the boundaries of the characters don't intersect each other (i.e characters are not overlapping), we can separate the characters in the image by scanning over the columns i.e from left to right, which would help in separating adjacent characters.

- For this we iterate over the columns of the image and check the frequency of the number of non white pixels in each column. Whenever we get non white pixels in a column it will indicate the start of a character (here we have used a window size of 30 pixels to detect characters and ignore any remaining noise after dilation) and continue checking the frequency of non white pixels in each column until we get a column with white pixels. Non white pixels followed by white pixels indicate the end of a character. We get the bounding box of the three characters by traversing along columns till the end of the image.

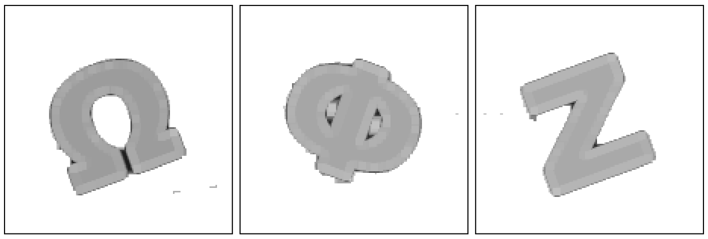
Figure 5: Segmented Character 150x150

- Out of 2000 images, we found 37 images where our method was not able to segment the given image in three characters. So we divide the image equally in three segments of size 150x150 by leaving a margin of 15 pixels in the beginning and 10 pixels margin in the following two. Extracting each character from the image using the bounding box we get from the above approach.

- Resize the image into 30x30 pixels and flatten the image to convert it into a 1D numpy array. Which forms the feature vector of one character. (Fig6)

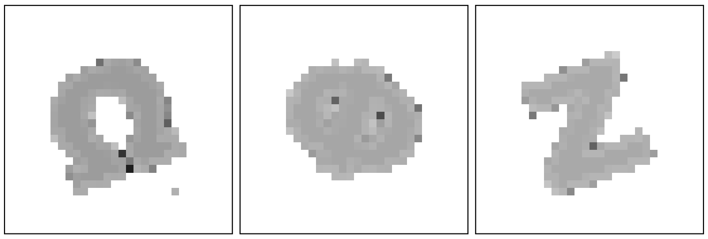
Figure 6: Segmented Character 30x30

1.3 Labels Encoding

We encoded character labels to numeric labels for the ease of training. We used the following dictionary to encode our character labels.

’ALPHA’ : 0,
’BETA’ : 1,
’CHI’ : 2,
’DELTA’ : 3,
’EPSILON’: 4,
’ETA’ : 5,
’GAMMA’ : 6,
’IOTA’ : 7,
’KAPPA’ : 8,
’LAMDA’: 9,
’MU’ :10,
’NU’ : 11,
’OMEGA’ : 12,
’OMICRON’:13,
’PHI’ : 14,
’PI’ : 15,
’PSI’ : 16,
’RHO’ : 17,
’SIGMA’ : 18,
’TAU’ : 19,
’THETA’ : 20,
’UPSILON’ : 21,
’XI’ : 22,
’ZETA’: 23

1.4 Model Selection, Training and Testing Phase

Once we have the feature vector (from pre processing and segmentation phase) and its corresponding labels (between 0-23), this converts our problem into a classification problem, we can apply various models such as SVM classifier, Logistic Regression, CNN (Convolutional Neural Network), etc. to solve this problem. We applied all models and found out logistic regression (2) to be best performing.

- **Hyper-parameter tuning**: There is a regularizer parameter C in Logistic Regression which we have to tune in order to find out best results. To tune the parameter we use grid search in order to find out the best values of C. We took different values of C and plotted a graph of the accuracy with various combinations of other parameters such as penalty and max_iter as well. The best result came out when the parameter penalty of ’l2’ is used and max_iter is equal to 5000 and the value of regularizer parameter C=1. (Fig7)

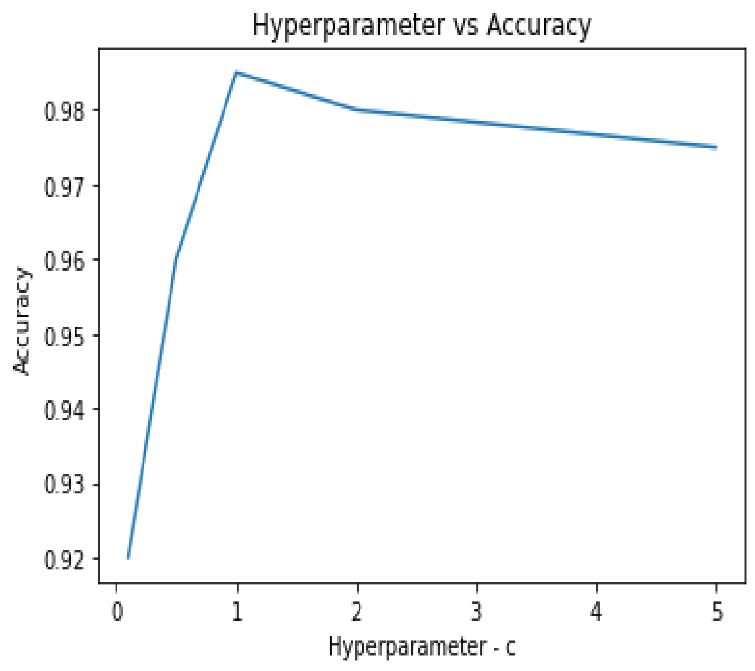

Figure 7: Hyperparameter C

- **Testing & prediction**: For prediction we mapped numeric labels to character labels using the following dictionary

0 : 'ALPHA',	12: 'OMEGA',
1: 'BETA',	13: 'OMICRON',
2: 'CHI',	14: 'PHI',
3: 'DELTA',	15: 'PI',
4: 'EPSILON',	16: 'PSI',
5: 'ETA',	17: 'RHO',
6: 'GAMMA',	18: 'SIGMA',
7: 'IOTA',	19: 'TAU',
8: 'KAPPA',	20: 'THETA',
9: 'LAMDA',	21: 'UPSILON',
10: 'MU',	22: 'XI',
11: 'NU',	23: 'ZETA'

1.5 Observation

To find out the accuracy of our model we use a 70-30 train test split. And the following table will tell about the performance of different models we tried on given dataset. We found out that logistic regression, predict the result with very high accuracy of 98.5% and takes very less time in doing so.

Model	**Size**	**Train Time**	**Accuracy**	**Test Time**
CNN	5MB	540 sec	95.75%	60 sec
SVM Linear	4MB	4 sec	96.15%	11 sec
SVM Gaussian	8MB	8 sec	97.85%	106 sec
Logistic Regression	174KB	264 sec	98.5%	8 sec

Comparsion table

References

[1] https://www.geeksforgeeks.org/erosion-dilation-images-using-opencv-python/

[2] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[3] https://docs.opencv.org/2.4/modules/imgproc/doc/imgproc.html

[4] https://www.geeksforgeeks.org/image-segmentation-using-pythons-scikit-image-module/

[5] https://www.geeksforgeeks.org/python-grayscaling-of-images-using-opencv/amp/