ASSIGNMENT 3

Group name: Naive Vectors

1

1.1 Dataset

The given dataset consists of 2,000 CAPTCHA images, each containing 3 Greek characters from the given 24 reference Greek characters. The font and font size of all these characters is the same. However, each character might be rotated by either 0°, ±10°, ±20°, ±30°.

We found the frequency of each individual character from the given 'labels.txt' file. The frequency ranged from 281 (highest) to 225 (lowest). Thus, we didn't find any label imbalance in the dataset. The character versus frequency plot can be seen in the plot below:

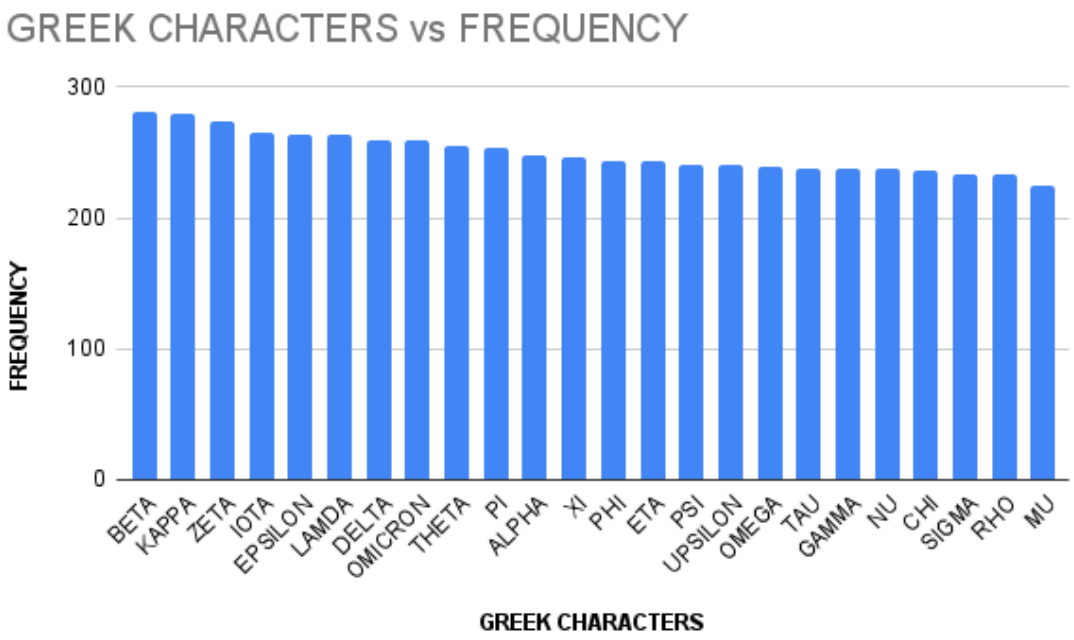


Figure 1: Greek Characters vs Frequency

We divided the dataset into 2 sets, training and test. Training set contains 80% of the total data i.e. 1600 images. Test set contains the remaining 20% of the total data i.e. 400 images.

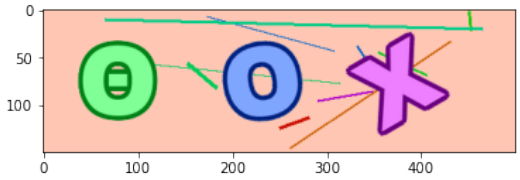
1.2 Image Processing

Before training and prediction of characters in images, the image is processed to remove the obfuscating lines and colors. The main goal behind this was to convert the images into simple forms. Then,

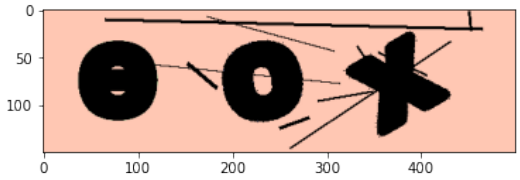
it will be possible to use a simple (linear) model that could learn from the given data and give low model size and fast prediction times.

Steps:

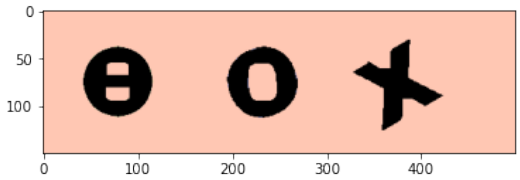
1. *Convert the image to HSV format.* Since the borders of characters are thicker than the background and the background seems to always have a lighter 'hue', this seemed like the best image format to work with. Also tried converting to grayscale and working with RGB itself but those formats were tricky to remove noise from.



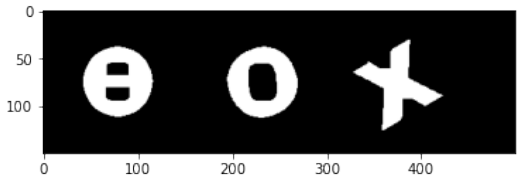
2. *Separate the background.* The first pixel of the image (top left pixel) is always the background HSV value. A mask is created with pixels in range with HSV of (0, 0, 0) to HSV of the background. After performing bitwise AND of the HSV image with the mask, only the background pixels remain. The lines and characters are removed, resulting in a black region where they were present:



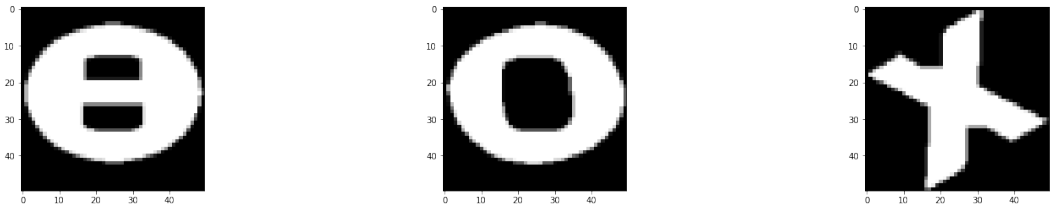
3. *Remove the obfuscating lines.* After masking, dilating the resulting image with a 3 x 3 kernel for 5 iterations removes the lines. The number of iterations is 5 based on trial runs with values from 1 to 10. 5 iterations seemed to give the best results so it was kept.



4. *Removing all color.* Extract the V channel of HSV image and perform inverse binary thresholding. This will turn the characters into white pixels and rest everything will turn black. Now we can perform segmentation on this image.



5. *Segmenting characters.* After observing the image obtained from the previous step, it can be noticed that the characters can be segmented if the image's columns are scanned from left to right to check if there are any white pixels present. Similarly, the rows are also scanned from top to bottom to crop out unnecessary portions of the image.



6. *Standardising image size for model training.* The character images extracted till the above step are in varying sizes. This step converts all of them to 20 x 20 pixels. This was done because the model will expect consistent data. After trying with 10 x 10, 20 x 20, 50 x 50 and 100 x 100 pixel sizes, 20 x 20 seemed the best in terms of maintaining prediction accuracy and keeping model size low.
7. *Preprocessing Training and Test Set Images.* The preprocessing procedure described is applied on all the images in training and test sets. Thus, for each image, the 3 characters within the image are extracted by preprocessing and output saved in a numpy array. Their corresponding character labels are also saved in another numpy array. This way for a total of 1600 images in training set, each containing 3 characters, we get 4,800 inputs for training set. Similarly, for a total of 400 images in test set, we get 1,200 inputs for test set.

1.3 Model Selection

We used a cross-validation based strategy for selecting the based model. Using sklearn's 'cross_val_score' function, we split the training set into 8 splits and ran different models on these splits. This function works such that a given model will be trained on 7 splits and tested on the remaining split. This process is repeated 8 times so that model is tested on every split once. This way we tested different classification models and made a note of their accuracies as well as model sizes. These are listed below:

Model	Average Cross-Val Accuracy	Model Size
Logistic Regression	1.0	77 KB
Random Forests	1.0	6,705 KB
K Nearest Neighbors	0.99	1,895 KB
SVC	1.0	3,832 KB

Table 2 : Model Performances

From the above table, it is difficult to differentiate between models based on accuracy alone as all have almost the same accuracy. But the main difference between these models is the model size. While logistic regression model size is just 77 KB, all the other models have a models size of more than 1 MB. Thus, for our task we decided to go ahead with logistic regression.

We retrained the logistic regression based model on the complete training set. Then we tested this model on the test set. It gave an accuracy of 1.0. Thus, we will be using this logistic regression based model considering its high accuracy with low model size.