CS771 - Introduction to Machine Learning - Assignment 3

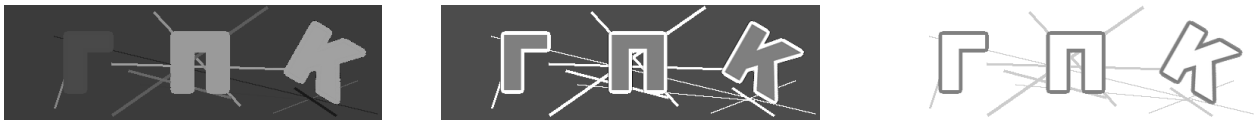
Problem 3.1 - DeCAPTCHA

Describe the method you used to find out what characters are present in the image. Give all details such as algorithm used including hyperparameter search procedures, validation procedures.

For Processing of images:

The processing of each image required separation from the background, removal of noise in the form of obfuscating lines, and segmenting each image to create parts with exactly one character. This was accomplished as described below:

1) We first used Gaussian blur to remove noise from the image. We made use of the cv2 library from *opencv* to extract the characters present in the image. Since the HSV color space is meant for object tracking, we used the *cv.cvtColor()* method to change from RGB space to HSV space. We then split the image in the HSV color space. We found that the the V channel was the clearest with the most removal of background and noise.



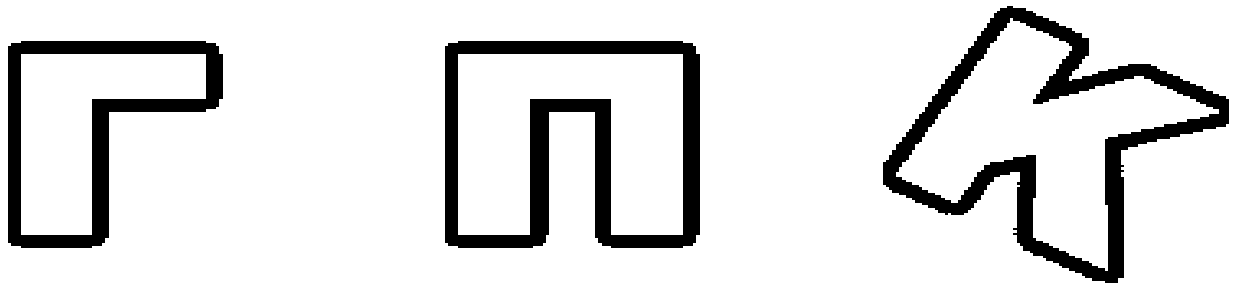
(a) H channel (b) S channel (c) V channel

Figure 1: HSV color space

2) We then employed the cv.threshold() method to get an idea of the letters. The input to this function is a grayscale image where all pixels exceeding the value 200 (lighter than 200) are assigned 255. Basically, we applied binary threshold on the V channel to find the border of the letters in a given image.

Figure 2: Binary Threshold Image

3) We looked at vertical columns of pixels that contain very few non-background pixels to identify regions where a character is present. The final product left at this stage was a segmented image with three segments - each with one character present in it.



(a) Char 1 (b) Char 2 (c) Char 3

Figure 3: Segmented Image

4) We also tried to attempt to rotate back the piece itself so that it matches one of the standard reference images but there was not much change in accuracy, although the image processing time greatly increased. Hence we dropped the idea of rotation.

Deep Learning:

The model loads all these segmented images and labels from train to begin with. We have used CNN for learning the image classification model. We built the model using two 2D Convolution (Conv2D) layers and two 2D Maxpooling layers alternating with each other. They are followed by the final Dense output layer. Some important hyperparameters in the model were the number of filters involved in the Conv2D layers and the kernel size in the MaxPooling2D layers. They were decided as follows:

1) We started with 32 filters in both the Conv2D layers. Though the model was able to achieve nearly 100% accuracy,our model size was huge. Upon changing filters to 16 and 32 in the two layers respectively, we observed that model still achieved same accuracy but in fewer epochs.

   So we kept on reducing filters till the model trained properly. It turned out that the model achieved full 100% accuracy on keeping 2 filters for first layer and 4 for the second layer.

   Reducing the number of filters also reduced model size by a considerable amount. Hence, the first layer of the model is a 2D Convolution layer with ***2 filters*** and a (4, 4) kernel size and the next Conv2D layers also has a (4, 4) kernel size but with ***4 filters***.

2) Though the model didn't require MaxPooling2D to increase its accuracy, adding Maxpooling layers greatly reduced the number of features and in turn, it reduced the model size. Since the processed images have very few necessary features (pixels) with most of them as white background, increasing the kernel size of max pooling doesn't throw away any important feature points. Hence, there are two alternating maxpooling layers with stride size 2 and kernel sizes ***(2, 2)*** and ***(10, 10)***.

3) We also tried to design a model with only one Conv2D layer but it couldn't learn the data. So having more layers was necessary. Only one additional Conv2D layer was sufficient, and our model was learning properly.

4) Finally, we flattened the output from the above four layers and fed into the Dense layer. This layer outputs a 24 dimensional output vector using the ***softmax*** activation function.

Our initial model had a prediction time of **10.264966 seconds** on 2000 data points and **100% accuracy** with a model size of 2.1 MB. After optimization, with the same prediction time and accuracy, the final model size comes out to be **968 KB.**