CS771A - Assignment 3

Aaditya Singh
160002

Aakarsh Gupta
160003

Method Description

Captcha Recognition

We attempt to solve the task of captcha recognition in this assignment, where we are given a colored image which consists of some letters (possibly rotated) and obfuscations, and we wish to identify those characters and the order in which they appear. An example is shown in Figure 1. The steps we followed are given below to perform this task.

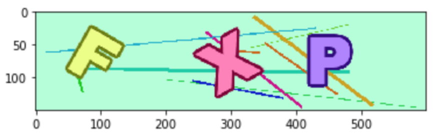

Figure 1: A random image present in the training dataset. Note the presence of obfuscating lines and rotated alphabets.

Identifying Background Pixels

We found out the background color by taking the mode color in RGB space of the four background corner color and removed that color from the entire image, an example of which is shown in 2. We also took the mean of the four colors instead but that approach has an inherent flaw that even one obfuscating line reaching up to a corner disrupts the process. We also tried to adopt HSV color space and remove the color with the maximum hue value but that wasn't a defining characteristic of the background colors as we found out by experimenting with the dataset.

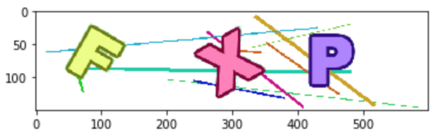

Figure 2: The image (above) after the removal of its background color.

Removing Obfuscating Lines

We read about Erosion and Dilation from OpenCV documentation [1] and found that Dilation is suitable for our purpose because the background after color removal is white (brightest), and we wish for this white region to grow into its surrounding obfuscating lines. Dilation scans a kernel over the image, finds the maximal pixel value overlapped by it and replaces the image pixel in the anchor position with this value. After experimenting with the kernel size and number of iterations, we found a 4*4 kernel of ones and 4 iterations to be the most suitable for removing the obfuscating lines without a major loss in the alphabet legibility as can be seen in Figure 3.

Preprint. Under review.

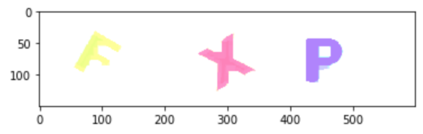

Figure 3: The image (above) after the removal of obfuscating lines.

Character Segmentation

We read about Contours from OpenCV documentation [2], but found it to be a bit complicated to understand and adopt for our much simpler task where there are only a few alphabets ensured with spacing in between them. Therefore, we came upon the idea of using **Breadth First Search** (BFS) for finding the connected components in the image after converting it into grayscale as shown in the Figure 4. Once the connected components are found, we just find the maximum and minimum row and column of each pixel belonging to one component to find out the bounding box dimensions of that component as shown in the Figure 5.

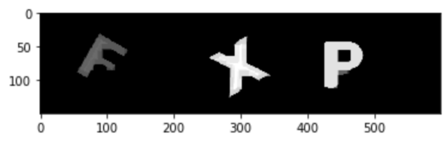

Figure 4: The image (above) after converting it into grayscale.

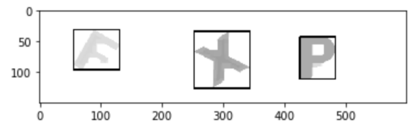

Figure 5: The image (above) after applying BFS for segmentation.

Character Recognition

We are using Linear SVM to classify the segmented images of characters into 26 classes, one for each alphabet. After obtaining the bounding boxes we cropped them out, padded them with zeros to make it of the shape (110*110), flattened them into a vector and fed the collection of such vectors as the input to train our one-vs-rest linear SVM models. To test the performance of our model, we split the images into training and validation at 75:25 ratio and used the longest common subsequence metric for evaluation. The hyperparameters C and max iterations were experimented with and found to be optimum at 1.0 and 10000.

Hyper-parametric Tuning and other Meta learning

Removing Obfuscating Lines

As discussed before, we used Dilation from OpenCV [1] to remove the background lines, which has two hyperparameters the kernel size used for dilation and the number of iterations of dilation. We started with the kernel size of 3x3 and 1 iteration. Keeping the size of the kernel same, we kept on increasing the number of iterations and then increased the value of the kernel size and did the same. After experimenting with the kernel size and number of iterations, we found a 4*4 kernel of ones and 4 iterations to be the most suitable for removing the obfuscating lines without a major loss in the alphabet legibility.

Character Recognition

We split the dataset randomly into a train:validation ratio of 75:25 and used the validation set to tune the hyperparameters, which were C and maximum number of iterations. Note that we kept the size of the cropped out bounding boxes of letters slightly greater than the size of the maximum bounding box (110*110) in order to keep the model size as small as possible. We varied C in the range 1 to 20 and observed that there wasn't a significant change in the validation score. On the other hand there were convergence issues for 1000 iterations but the performance was same for 10000 or more iterations. Thus, we chose C equal to 1.0 for simplicity and max iterations to be 10000.

Other Methods

We tried out multiple methods other than the Linear SVM method to perform character recognition after obtaining the bounding boxes for each character from the aforementioned technique

1. We tried the template matching technique using the correlation between the bounding boxes obtained after segmentation and the templates. The given reference images were hollow alphabet images and the alphabets in the captcha were solids images. So we replaced the given reference images with the with solid alphabet templates available at [4]. Then we found the closest template (with respect to the cross-correlation score) to each of the letters present in a bounding box and predicted that alphabet as the output for the letter in that bounding box. But this approach yielded poor results (0.32 on validation set) so we did not use it as our final approach.

2. We also built a Convolutional Neural Network (CNN) to accomplish the character recognition task. We trained the CNN to perform 26-class classification, each class representing an alphabet. The bounding box obtained was padded with zeros to the shape (110x110) (to maintain uniformity) and then they were reshaped to (64*64). These (64x64) images were the input to the CNN. The network consisted of a 2d convolutional layer with 32 3*3 kernels, 2d max pooling layer with 2*2 pooling, one dense layer of output length 48 and another dense layer of output length 26. All the hidden layers have ReLU activation and the output layer has Softmax activation. We have used categorical cross-entropy loss function and Adam optimizer with a learning rate of 1e-3 (default) for 10 epochs, for which we obtained a validation accuracy of 99% and a validation score of 0.98. The average prediction time and the size of this model per image was found to be much larger than that for the Linear SVM model. So we chose the Linear SVM method over CNN.

References

[1] Eroding and Dilating: OpenCV

`https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html`

[2] Contours: OpenCV

`https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_table_of_contents_contours/py_table_of_contents_contours.html`

[3] SVM: scikit-learn

`https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC`

[4] Alphabets: Printable Alphabets

`https://printable-alphabets.com/printable-letters-a-z-templates`

[5] Sequential Model: Keras

`https://keras.io/getting-started/sequential-model-guide/`