CS771

ML Class of '14

Problem 3.1

(DeCAPTCHA)

Your task is to design an algorithm that can take a set of such 500×150 images and return the code present in that image.
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. You have to give a detailed explanation even if you used an algorithm/implementation from the internet make sure to give proper credit to the person/source from where you took code/algorithm. There is no penalty for using someone else's code/algorithm but there would be heavy penalty for doing so without giving proper credit to that person/source.

Answer: The model works in two stages, firstly by separating the component characters using methods of computer vision and then using a convolutional neural network based deep learning model to predict the individual characters. Each of these stages have been elaborated below.

During the first phase, there are multiple morphological processes done on the image. The first one of them is to remove the "stray lines" which is done by first dilating the image and then eroding them. The letters are restored to roughly the same state after the removal of stray lines. An image after this operation is shown below:

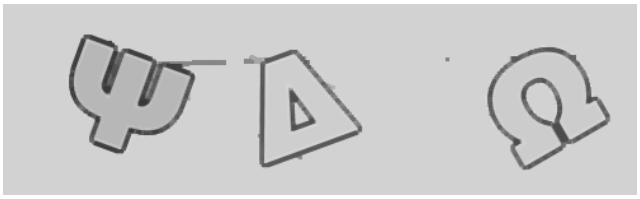


Figure 1: Stray lines removed

The next morphological transformation essentially works to separate the margins and make them bold using a morphological kernel. The image after this operation is shown below:

Figure 2: Bolder margins

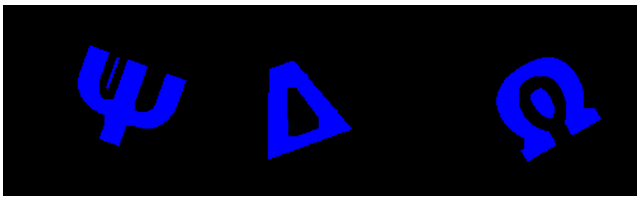
In the above image, the white parts of the image can be viewed as a set of connected components. We can again see that turning the outermost connected component black will separate out the letters.

Preprint. Under review.

Owing to the small size of the letters in comparison to the overall image this component will also be the largest one. So now our aim is to find the largest connected component in the image.
This is simply performed by using an `opencv` function that computes the connected components of the image along with some metadata. We use 8 connectivity to define the connectivity of two pixels. This is essentially a strong notion of connectivity that does not allow two sections to be connected if they are only linked at a corner. The image now looks like this.

Figure 3: Blacking the background

Observe that we have mostly segmented out the letters. Now we just want to remove the random white splotches. All these splotches are essentially contours. So we get the contours and their hierarchy. Contours can be organized into a tree. The contours enclosed by a contour are its children.
First, we include only the highest level contours (only the roots or children of the roots). This is to remove the problem of letters like O or Θ which have multiple levels of elliptical contours and we only want the first two levels.
We can also see that the useful contours tend to be close to polygonal in shape. So we first approximate the contour with a suitable polygon which is defined at runtime using the contour characteristics, and compare the areas of the approximation and the original contours. If the areas are very close then it means that the contour is useful (since useful contours tend to be very well approximated by polygons in our use case). However, we do not reject the contour if it is not the case.
Now we will finally check and remove the tiny splotches. We sort the remaining contours by area. We then proceed in descending order. If the ratio of areas of two successive contours is too high, then we stop the traversal and reject the rest of the contours. The traversed contours are accepted.
All of the accepted contours till now are drawn on an image. An example is here:



Splitting an image into three parts is simple. We randomly sample N points and store the white pixels. Then we sort them on their x coordinates. We will see two large jumps in the x-coordinates. We split the pixels at those two points. Thus, we have three groups of pixels. We compute the average for all three groups. Then we take a square centered at the average for all three groups.
Finally, the segments from the processed image is sent to a neural network built using TensorFlow.

Model is described as follows :

- `Conv2D(32, (3,3), activation='relu')`
- `MaxPooling2D(2, 2)`
- `Flatten()`
- `Dense(32, activation='relu')`
- `Dense(24, activation='softmax')`

We chose this model because of its simplicity in order to keep the size small and its ability to learn and recognise characters from the segmented images.

We use `Adam(learning_rate=0.01)` optimiser with `loss=`

'sparse_categorical_crossentropy' as it is more effective in comparison to GD. Hyperparameters and optimiser were tuned using results obtained on the validation data set.

Statistics on a spilt of (1500 Train , 500 Test):

- Size of the Model: 11.9 MB
- Testing time per image: 0.127 secs
- Code Match Score: 0.996