Indian Institute of Technology Kanpur
CS771 Introduction to Machine Learning, 2017-18-a

Assignment Number: 3
Student Name: Ankur Kumar, Sahil Bansal, Abhishek Pardhi, Suket Raj, Parth Maniar
Roll Number: 200140, 200836, 200026, 201013, 200671
Date: November 27, 2022

QUESTION 1

Algorithm: Pre Processing

We pre-processed the images using broadly 3 functionalities, removing the extra lines and segment it to divide it into 3 parts which contain just the character, and finally extracting required features from these segmented images.

Line removal

We use the following procedure for a single given image, given its background color and the neighbourhood parameter(defined below).

Approach

Our approach here is to generate a new image pixel-wise while removing pixels which do not match with their neighbours , for this we check for every pixel on the image we check neighbours of the pixel and see if they are uniform:

1. if they are uniform this must mean that the pixel either belongs to the background or it must belong to the character, assuming that we check neighbourhood big enough.
2. if the set of pixels examined are not uniform then the pixel must belong to a line or the border of the character and we can change it to a pixel with same color as the background. So after the procedure we would be left with no or much thinner lines .

Implementation

We use the convulve2d function from scipy.signal to convulve the image layerwise (3 different layers). We have a parameter for size of neighbourhood to examine. We use a kernel of square dimensions $2n + 1$ with each entry as 1, to cover n neighbouring pixels and check for uniformity, for the second parameter in convulve2d function.
After convulving the image we get another image say image2 we shall now update each value of image matrix of the origional image such that each entry is $(2n + 1)^2$ times the original value, now using this we generate a new image pixelwise such that a pixel:

1. is same as pixel from origional image if the convulved image pixel is same as updated original image as equality most likely implies that each pixel examined in the neighbourhood was indeed the same and hence must belong to either character or background. Thus retaining the charcters in the final image.

2. is same as background if the convulved image pixel is different from updated original image, this was done because this cannot belong to the character and hence must belong to either the line or the border of the character which we can afford to dissolve. Thus getting rid of extra lines from our image.

Result

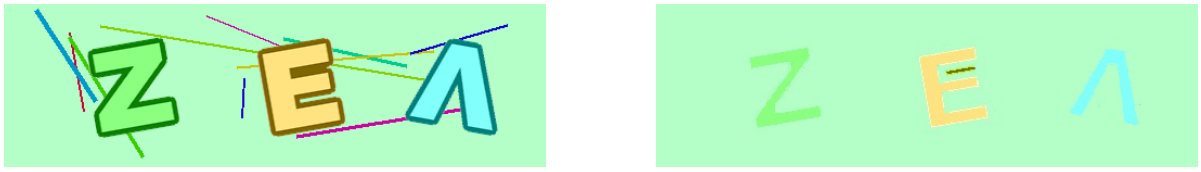

Figure 1: Result of removing lines from an image

Segmentation

We work on a given line removed image, along with 2 threshold values described later. We call the segmented images as raw images.

Approach

Our approach here is to identify the fill color of the characters and isolate 3 smallest slices of the image containing the characters.

Implementation

We compute the frequency of colors in the pixels and keep a track of the same. The most frequent color must be the background (this was how background colour was identified for lines removal). Colors with frequency below a threshold (parameter default 400, from processing multiple images) are ignored and other colors are observed.
If only 2 colors remain then all the 3 characters must be of the same color, else if only 3 colors remain then there are only 2 unique character colors, else all characters have different colors.

We now initialise a histogram where we count pixels belonging to any character(either of the 3 colors found above) in a single column of the matrix of image. Now we run traverse through this histogram and find a contigous segment of columns which have a value more than a threshold (parameter, default 3 as residual lines cannot exceed this thickness).
We ignore columns with value less than a threshold. If during iteration we encounter columns crossing the threshold we mark that as the start of a segment and continue till we find a column which drops below the threshold, after which we create a segment from these columns. We continue till we find 3 segments for each character. We return these image slices.

Result

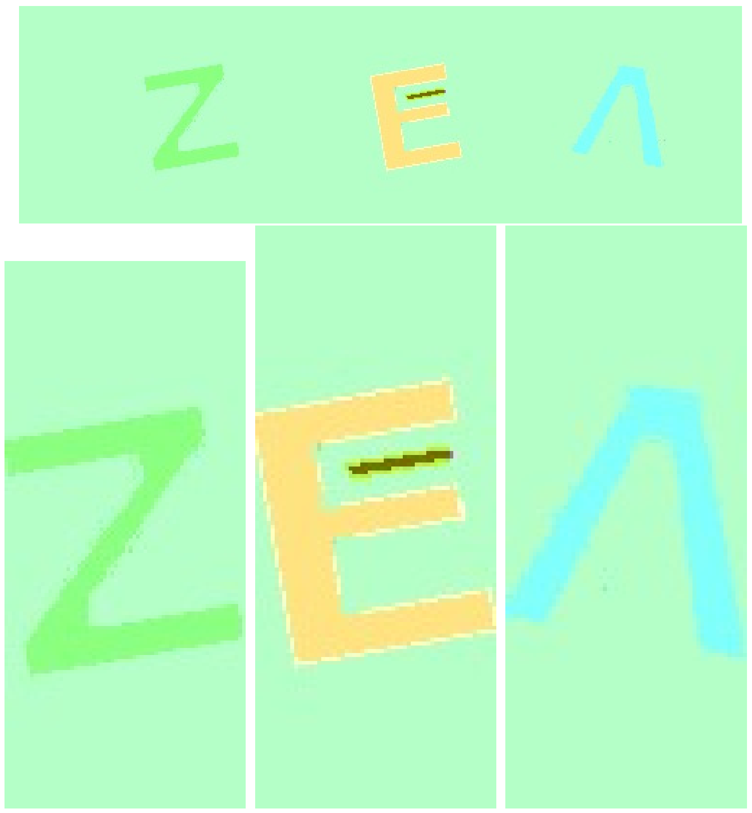
Figure 2: Result of segmentation of a line removed image

Feature Extraction

Given the segmented image containing a single character we extract features from it to train the ML model. All binary images were resized to 64×64 image. Note that this method also removes any residual lines as shown for 'E' below

Approach

We used the idea described in suggested methods of removing all the background and converting the image into binary image where the pixel belonging to the alphabet is given value 255(white) and all other pixels are given value 0 (black).

Implementation

The automated cropping was done same as segmentation by building histogram of non background pixels and removing starting and ending rows containing less than a threshold number(default value 3) of pixels. pixels belonging to character were identified using the colour information. The flattened image is used as input to ML model.

Result

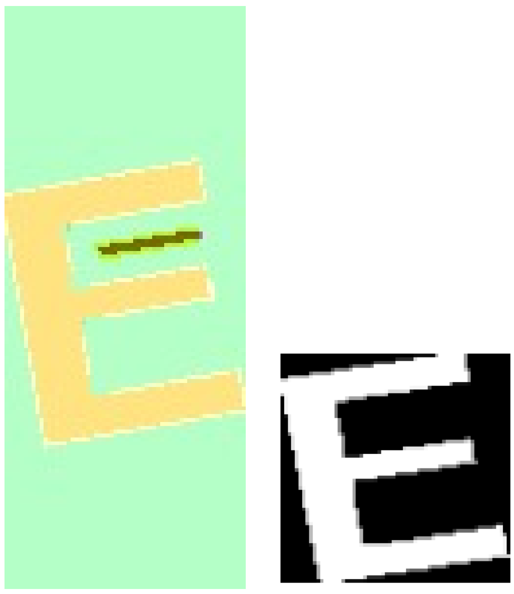
Figure 3: Result of feature extraction

ML model

We experimented with multiple models including **feed-forward neural networks, convolutional neural network** (trained on raw images as they can extract the features themselves efficiently) but the speed-accuracy trade offs were not good, so we tried **SVM** on raw images the results were satisfactory but not that good (about 90% validation accuracy) so our final model was trained on extracted features and results are described below.

Hyperparameter Tuning

- For SVM we tuned first the regularization parameter C small values of C (eg. 1) were giving low training accuracies (only 80%) so we increased it and settled at C=5.
- Then we experimented with multiple kernel types including linear, polynomial kernels, but since we could not find any benefits of using other kernels we stick to linear kernel.

Results

- Training Time: $<$ 2 mins
- Prediction Time: around 1 min for 4K images.
- Training Accuracy: 100%
- Validation accuracy: 100% !!

Dataset

The complete dataset of 2000 images were divided into three sets: a test set containing 100 images for complete testing, the remaining images were processed, segmented and features were extracted and then divided into training and validation set in the ratio 85:15

Results

Here are the results of final testing conducted using eval.py file provide on the 100 test images-

- Time taken: 35 s (0.35 per image)
- Total score: 0.9800

Libraries used:

- **Sklearn**: for convolution operation and model training.
- **Opencv and numpy**: for image preprocessing
- **pickle, json**: for loading/storing model and label maping.