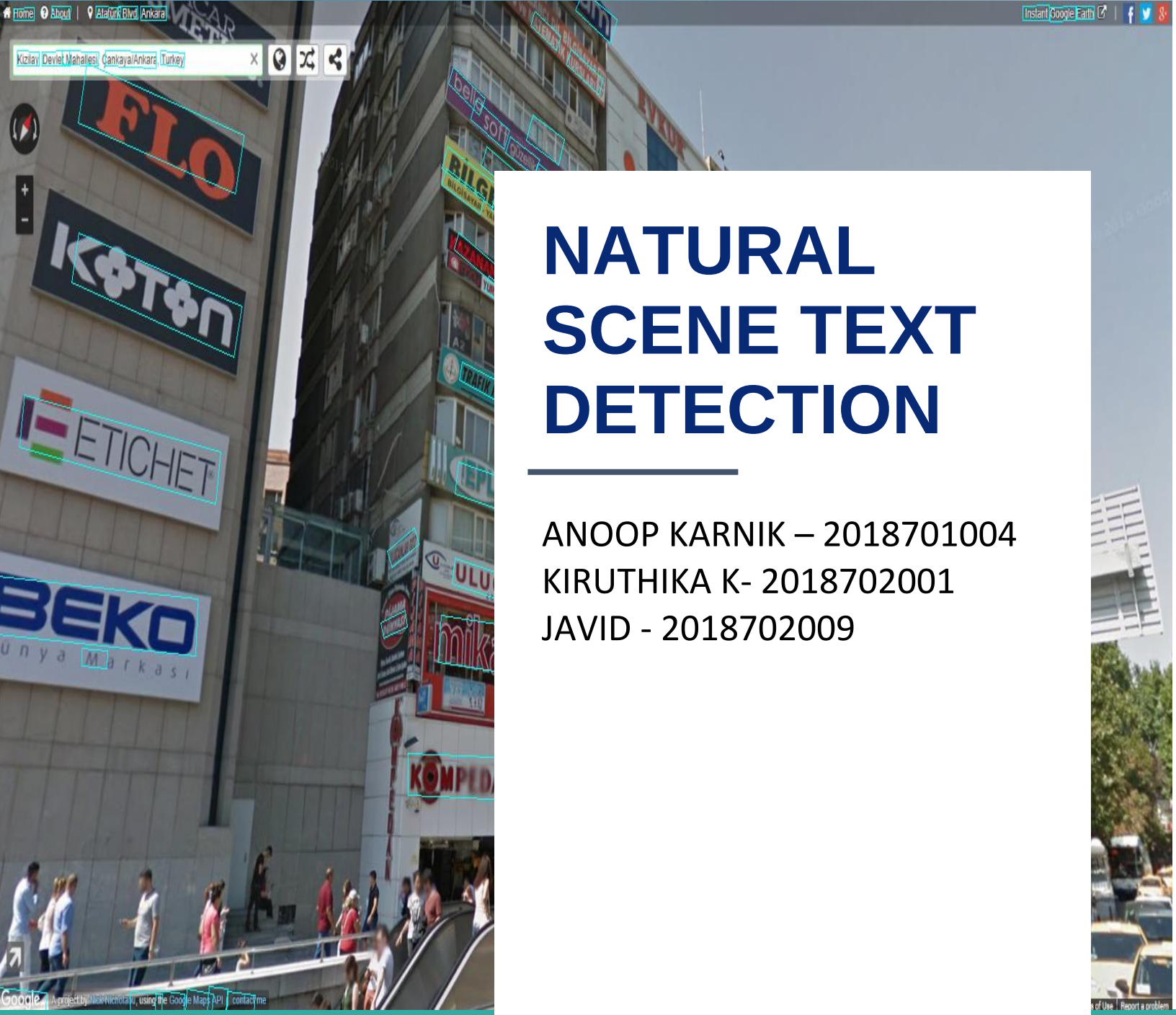


NATURAL SCENE TEXT DETECTION

ANOOP KARNIK – 2018701004
KIRUTHIKA K- 2018702001
JAVID - 2018702009

CV PROJECT

Abstract

Full end-to-end system for text detection in natural images is a challenging problem with many practical application. Models using carefully hand engineered features or large amounts of prior knowledge have been shown to give good results in the literature. Our objective in this project is to build a complete yet simple end to end system to detect the text present in natural scene images. The ICDAR 2003 dataset of train and test has been used to test our results.

Introduction

Multimedia content generally consists of images and videos taken in wild with no prior restriction on their content or language. We generally refer to them as natural scene images signifying the content possesses various kinds of scenic information in them. The text present in them is eventually referred to as scene text. Detecting text in a natural scene is an important part of many Computer Vision tasks. Unlike character detection for scanned documents, detecting text in unconstrained images is complicated by a wide range of variations in backgrounds, textures, fonts, and lighting conditions.

Overview

The text detection part scans the image for possible character like pattern and provides bounding box for the detected text. Further we explore detection of lines and words from the bounding boxes provided by the text detection system.

GITHUB LINK: https://github.com/anoopkarnik/SceneTextDetection

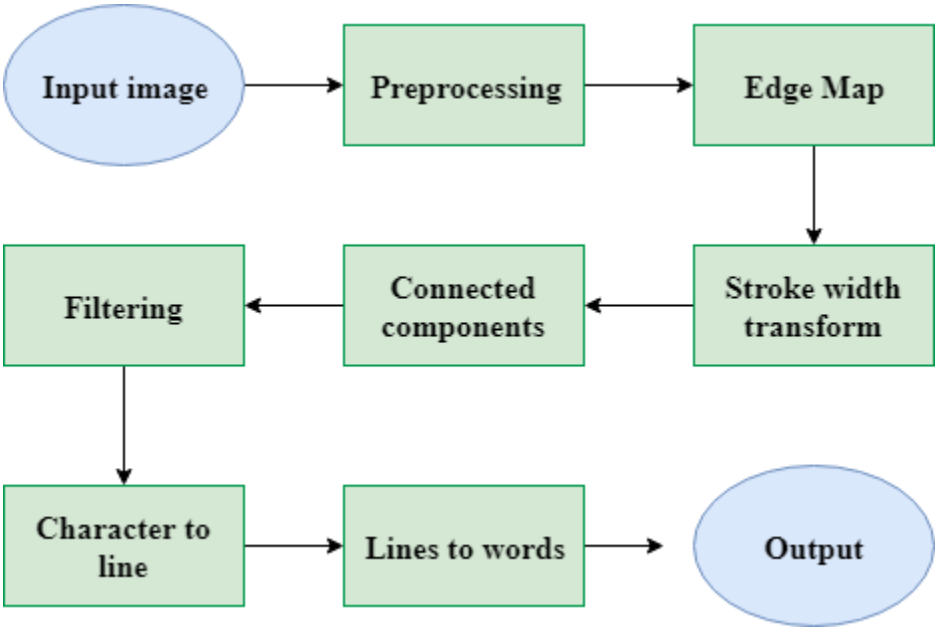


Fig.1. System overview

Preprocessing

The image is gray-scaled and resized to 640x480 dimension. A gamma corrected grayscale image is used to obtain edges using Canny edge detector. The gradients of the edge image to be used in the stroke width transform is obtained by using a Scharr filter and the gradient direction in each pixel is calculated.

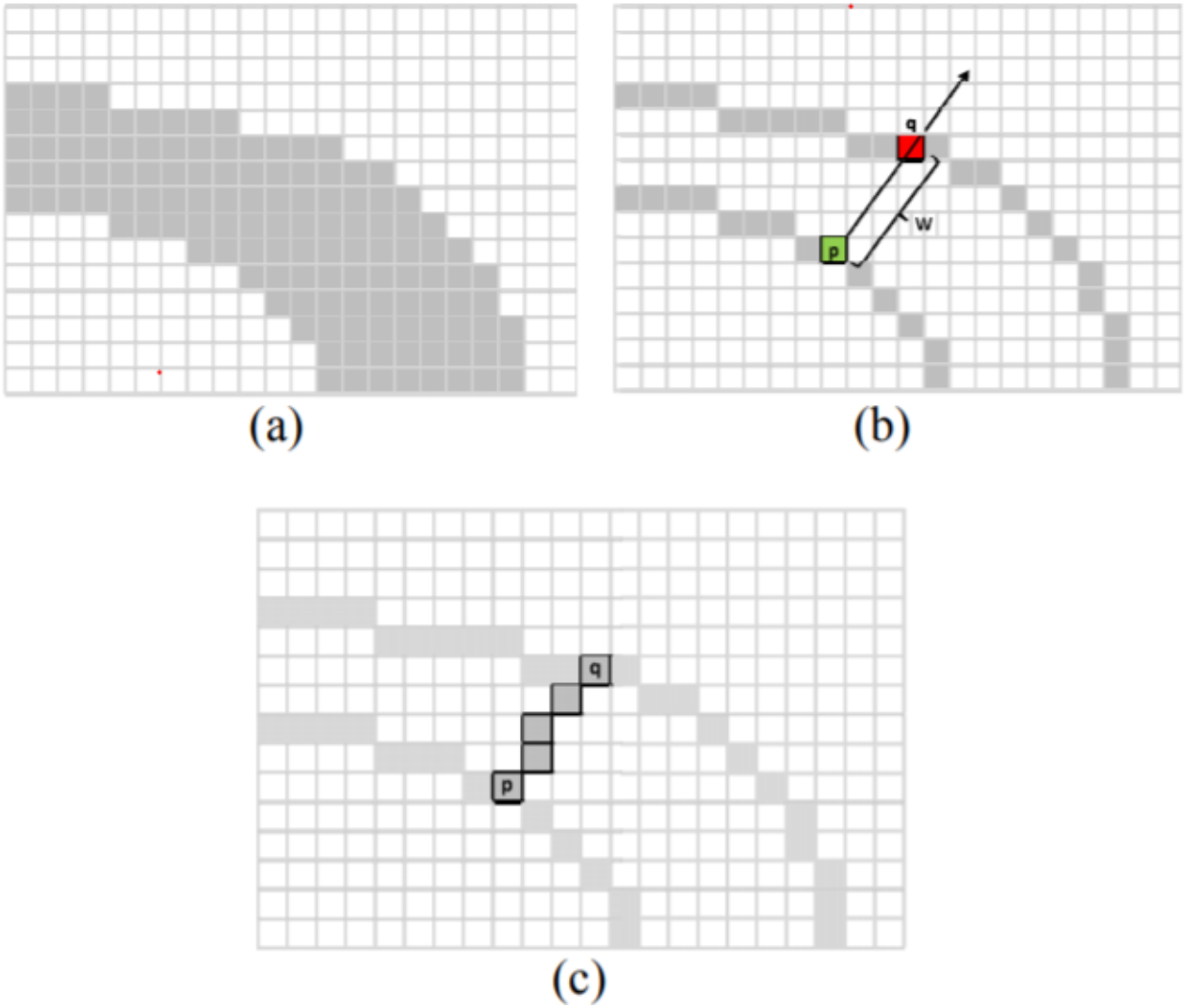
Stroke width transformation

Text tends to have higher uniformity in its stroke when compared to any other non text regions. The Stroke Width is calculated in the entire image and regions with uniform Stroke Width are grouped together. Then these regions are filtered based on variance. A stroke in the image is a continuous band of a nearly constant width. An example of a stroke. The Stroke Width Transform (SWT) is a local operator which calculates for each pixel the width of the most likely stroke containing the pixel. First, all pixels are initialized with ∞ as their stroke width. Then, we calculate the edge map of the image by using the Canny edge detector. We consider the edges as possible stroke boundaries, and we wish to find the width of such stroke. If p is an edge pixel, the direction of the gradient is roughly perpendicular to the orientation of the stroke boundary. Therefore, the next step

is to calculate the gradient direction gp of the edge pixels, and follow the ray r=p+n*gp (n>0) until we find another edge pixel q. If the gradient direction gq at q is roughly opposite to gp, then each pixel in the ray is assigned the distance between p and q as their stroke width, unless it already

has a lower value. If, however, an edge pixel q is not found, or gq is not opposite to gp, the ray is discarded. In order to accommodate both bright text on a dark background and dark text on a bright background, we need to apply the algorithm twice: once with the ray direction gp and once with –gp. After the first pass described above, pixels in complex locations might not hold the true stroke width value. For that reason, we will pass along each non-discarded ray, where each pixel in the ray will receive the minimal value between its current

value, and the median value along that ray.



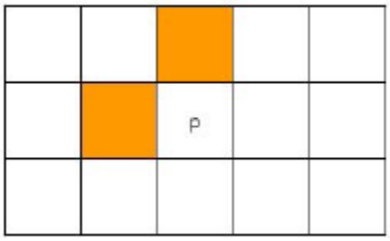
(a) (b) (c)

Fig.2. SWT

Connected component algorithm

The algorithm consists of two passes. In the first pass, the algorithm goes through each pixel. It checks the pixel above and to the left. And using these pixel's labels (which have already been assigned), it assigns a label to the current pixel. And in the second pass, it cleans up any mess it might have created, like multiple labels for connected regions.

The First Pass: In the first pass, every pixel is checked. One by one, starting at the top left corner, and moving linearly to the bottom right corner.



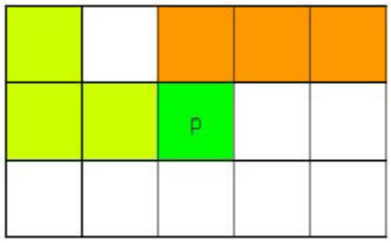
If you're considering the pixel 'p', you'll only check the orange pixels. Thus, at any given time, you only need to have two rows of the image in memory. We'll go through each step of the first pass one by one. Here we check if we're interested in a pixel or not. If the pixel is a background pixel (its value is zero, or whatever other criteria you want), we simply ignore it and move on to the next pixel. If not, you go to the next step. Here, you're fetching the label of the pixels just above and to the left of 'p'. And you store them (into A and B here). Now, there are a few possible cases here:

The pixel above or/and the the left aren't background pixels: In this case, things proceed as usual. You just go to the next step. A or/and B will have actual values (the labels).

Both pixels are background pixels: In this case, you cannot get labels. So, you create a new label, and store it into A and B. You figure out which one is smaller: A or B and then you set that label to pixel 'p'.

Suppose you have a situation where pixel above has a label A and the pixel to the left has a label B. But you know that these two labels are connected (because the current pixel "connects them").

Thus, you need to store the information that the labels 'A' and 'B' are actually the same. And you do that using the union-find data structure. You set the label 'A' as the child of 'B'. Using this information, the algorithm will clean up the mess in the second pass.



The only thing you need to remember is that the smaller label get assigned to 'p', and the larger number becomes a child of the smaller number. Go to the next pixel.

The second pass: Again, the algorithm goes through each pixel, one by one. It checks the label of the current pixel. If the label is a 'root' in the union-find structure, it goes to the next pixel. Otherwise, it follows the links to the parent until it reaches the root. Once it reaches the root, it assigns that label to the current pixels.

Filtering non-text detection

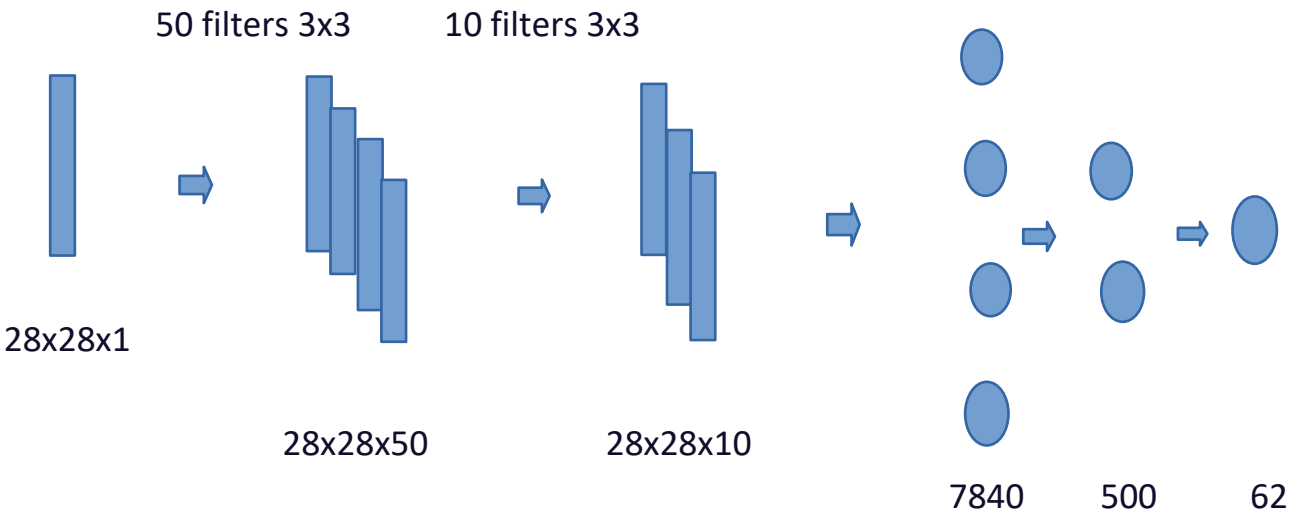
The text detected by the above said method was found to produce a lot of false positives. So, we use certain inherent properties of text in natural scenes to eliminate some of these false positives.

We compute the variances of the obtained stroke width image and compare the variance of the stroke width of each component with the mean stroke width of the same. Rejecting large value of this ratio removes foliage like textures that is common in natural scene images.

We also remove detected text which are too big and too small and with unnatural aspect ratios.

Finally a neural network is trained to reject anymore false positives left. The neural network trains on ICDAR 2003 character recognition trial train dataset of 6185 images with labels for each image which can be 0-9,A-Z and a-z. We use two convolutions layers with 50 and 10 filters respectively, kernel size =3 , stride =1 and padding = 1. Then we use two fully connected layers with 500 and 62 outputs and use relu activation on all layers except the end one where we use sigmoid activation. With a batch size of 50 and 100 epochs with 0.01 learning rate using cross entropy loss and Adam optimization we train our model. We save the state dict of model as model.pt file. We use this state dict to run

on a image window we determined and if the max probability of any of 62 outputs we get from the model is less than 0.1 we term that window as having non text.



Finding lines and words

Once we detect individual characters we need to find lines and words.

We find individual lines by comparing the row location of the bounding box for each pixel. If the row locations of one character lies within the other character they belong to a line else a separate line is created. The min and max row values are found using a median value to get the bounding box of the line.

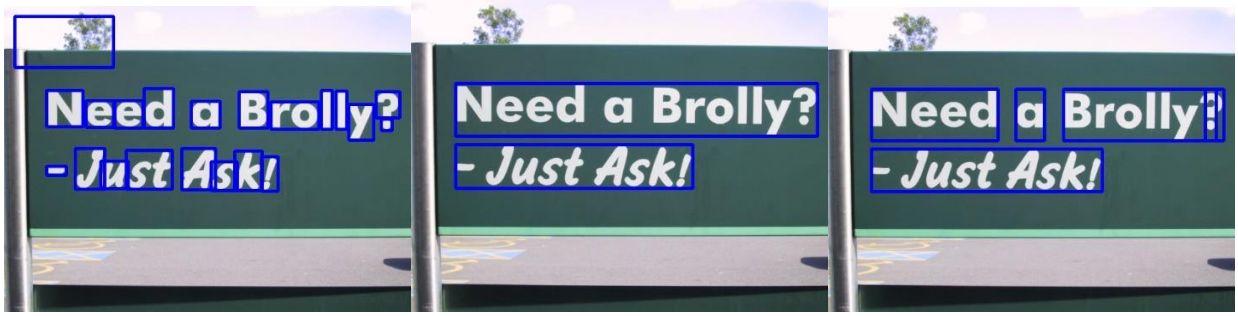
To split the lines into words, for each line, the median of the distance between the characters is found and a line if split into two words between two characters if their distance is greater than thrice the median value.

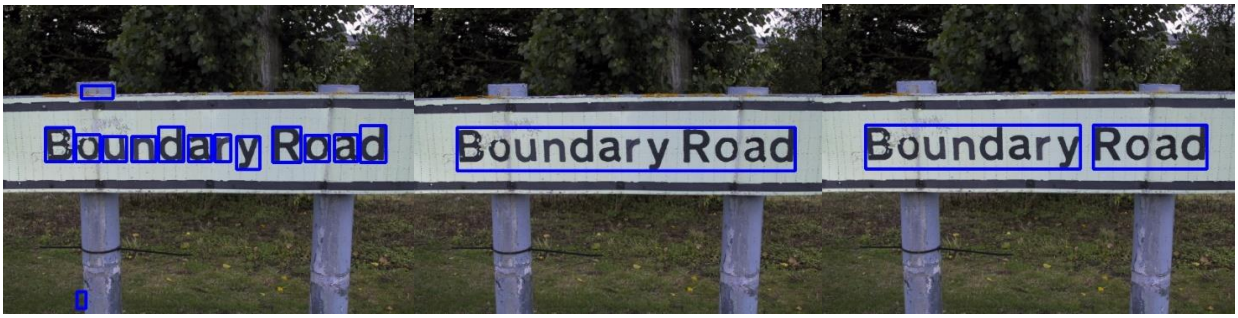
Here too some more boxes are removes as in a single line if only a single text bounding box is present it is removed if a bounding box height is 3 times more than median bounding box present in the line it is removed.

Experiments

We experimented on the ICDAR combination of train and test dataset containing 196 images and obtained the following results:

Success Cases -





We observe that if the immediate background of text has a good difference in color contrast respect to text our algo works perfectly detecting all the letters then lines and words.

Failure Cases:

In the above image the letters are very close together so in connected components they were taken as one single component so 4 letters termed as a single one even the symbol has similar stroke width thought so it too is considered a letter. Similarly as gap between symbol and word is very less it is considered a single word.

In the above image the contrast difference between text and its background is pretty less so its not able to detect a single character in the above image.

In the above image it is able to detect most of the letters but some letters have like e has lots of variance in its stroke width so its is not able to detect it.

The accuracy tested using combination of images from ICDAR 2003 training and test sets around 196 images are:

Detection Accuracy	Positive (IOU>0.5)	Negative (IOU<0.5)	False Detection (IOU=0)
Dark text Bright Background	0.572	0.311	0.117
Bright Text Dark Background	0.591	0.326	0.102

Conclusion

We use the concept of stroke width transform to detect text in natural scene images. We increased the robustness of the system by applying additional filtering techniques. We also detect lines and words in the scene text. We have built a system that provides an end-to-end module to detect straight English text in images posing various challenges. Complete datasets, outputs, instructions to run and code are available in github link : https://github.com/anoopkarnik/SceneTextDetection