Text Detection and Recognition from Natural Scene using Stroke Width Transform[1] and Deep Feature Classification

Ishtiak Zaman (izaman@indiana.edu)

Image Reprinted From: https://www.pinterest.com/pin/336995984590869672/

Introduction

In a natural scenery, there could be multiple instances of text that an agent may want to read. In this project, we detect and recognize text characters from an image.

The project has two major parts. First part is to detect and localize text characters in an image. This part is able to filter out the characters from the whole image that can be used in the second part. We are using Stroke Width Transform[1] method to detect and localize text.

In the second part, we have a set of localized characters. We extract the deep features of each characters and classify the characters using a trained SVM[5].

Background and Related Work

There has been many works related to text detection and text recognition using various kind of methods. Most commercially available OCR software can successfully read the texts from printed documents, but fails to read texts in natural scenery due to different orientation, color, font, noises etc.

The PhotoOCR[6] uses three different kind of methods to detect texts, a mixture of Viola-Jones style boosted cascade of Haar wavelets, connected components from a binarized image and uses graph cuts to find a text / non-text labelling by minimizing an MRF with learned energy terms, and anisotropic Gaussian filters. Finally a trained network recognizes the text.

There has been excellent work on detecting text regardless of text size, font, color, orientation, language using Stroke Width Transform[1]. Text characters have similar stroke width compared to other elements in a natural scene, this property is used to detect text successfully. I am using similar technology in this project to detect text using Stroke Width Transform.

Text Character Detection and Localization

To detect and localize every characters in the image we followed the following steps:

Edge Map Detection:
The first step is to get a thin noiseless edge map. We've used Canny edge detector[5] to detect the edges with non-maximum suppression. The following figure shows the

original image with edges:

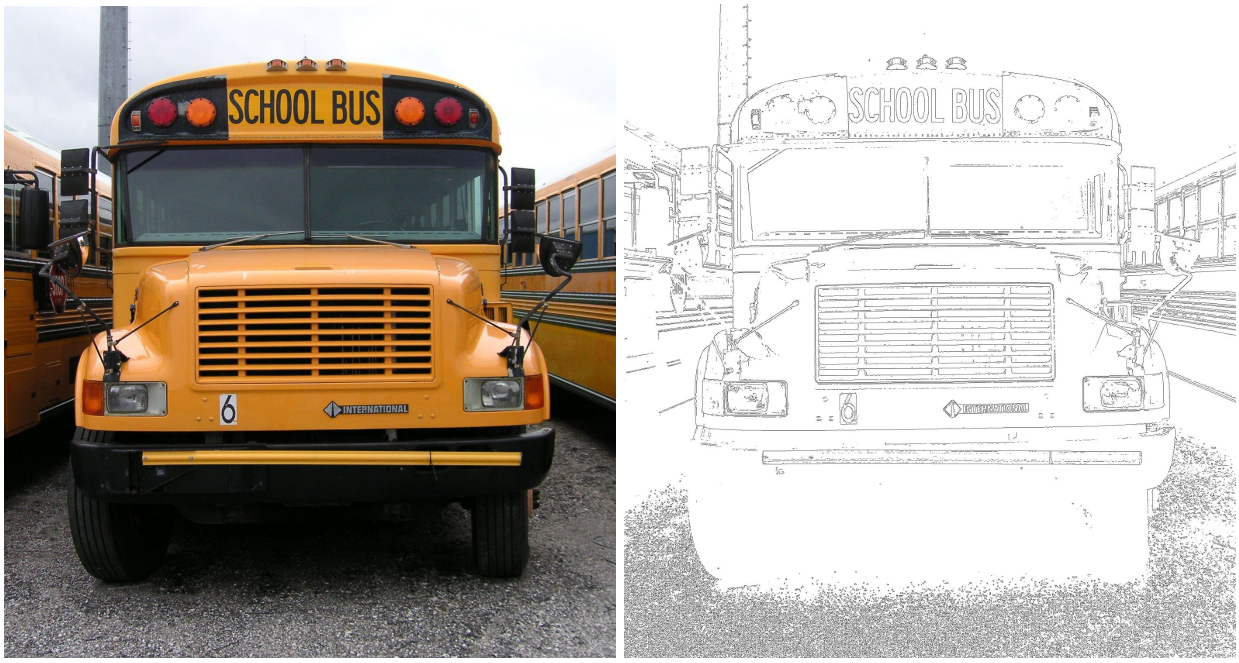

Fig: Original Image (Reprinted from: http://www.fts4buses.com/vehicles/2002-international-bus-8135) and generated Edge Map

Stroke Width Transform[1]:
The next step is getting stroke width of every pixel of the images using Stroke Width Transform[1] method. In this method, when an edge pixel is found, we start searching through a straight line in the direction of the edge gradient direction of that pixel to find another edge pixel with opposite edge gradient direction. If such a pixel is found, all the pixels between the two pixels are assigned a stroke width that is the euclidean distance between the two pixels. This method is applied for all pixels to get a map of stroke width of all pixels.

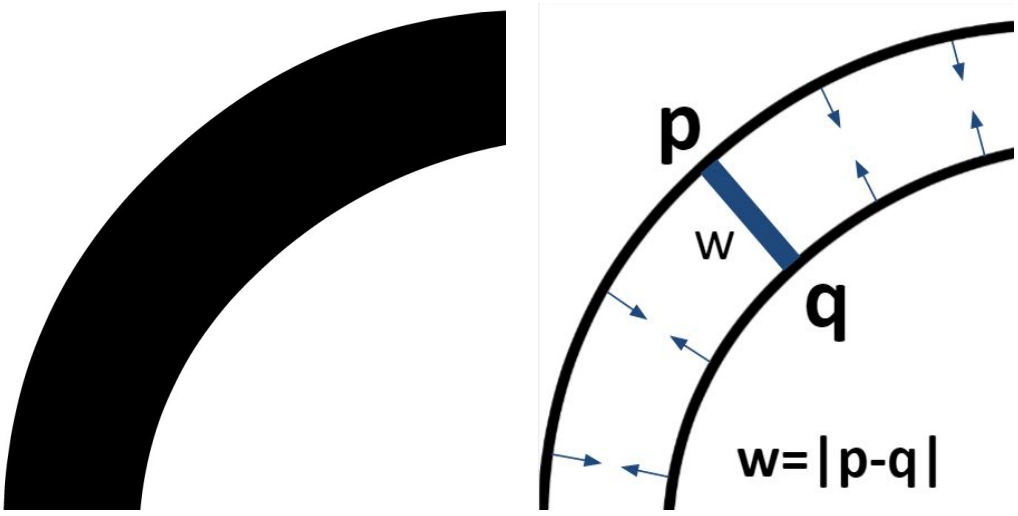

Fig: A Typical Text Stroke and Edge Map with Stroke Width

In the image, for each edge pixel p, we search in the gradient direction of p for another edge pixel q. If gradient direction of q is opposite of p, all the pixels within the search direction has width of |p-q|.

The following figure shows the resulting image after Stroke Width Transform.

Fig: After Stroke Width Transform[1], text characters have the most consistent width

Eight Layer of Filtering:
In the Stroke Width Transform[1] image, the text characters have the most consistent width compared to other items. We first group neighbor pixels together with the condition that the neighbor pixels must have the value ratio less than 5.0 to be in the same group. Then we run the following filters to all the groups to filter out inconsistent items.

- Filter out groups with high width variance.
- Filter out groups with too many pixels compared to the original image.
- Filter out groups with too small mean width .
- Filter out groups with height / width ratio that does not go with text characters.
- Filter out groups with too small too large diameter.
- Filter out groups with unnatural area size compared to the original image.
- Filter out groups with unnatural ratio between diameter and mean width.
- Filter out groups that are too near to the margin.

After the filtering we are able to filter out the non-text groups. The following figure shows the image after filtering:

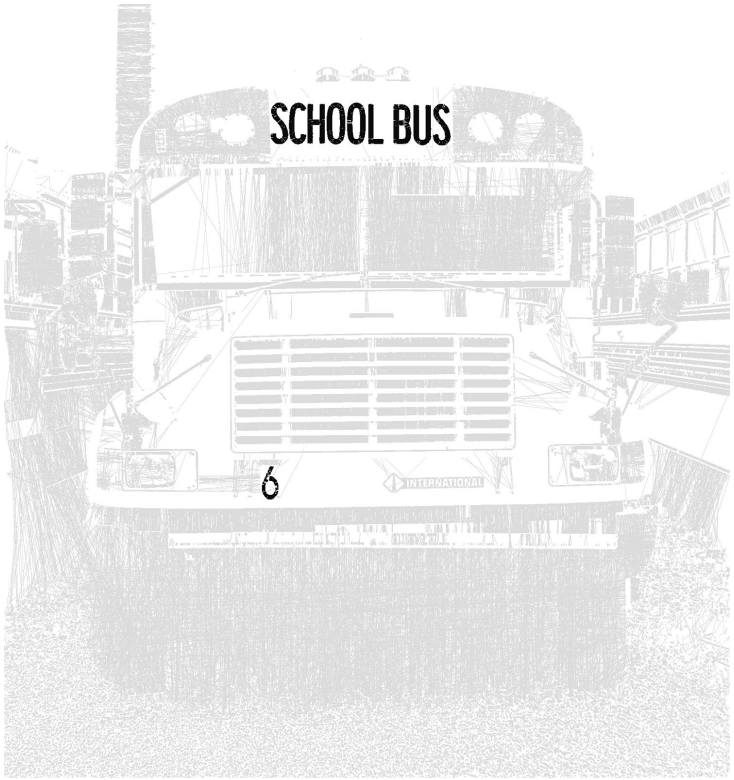

Fig: Non-text groups are filtered out

Localize Characters:
After filtering we have the groups of characters. For each character, we find the minimum and maximum of x and y coordinate to generate a bounding box around them. The following figure shows the bounding boxes.

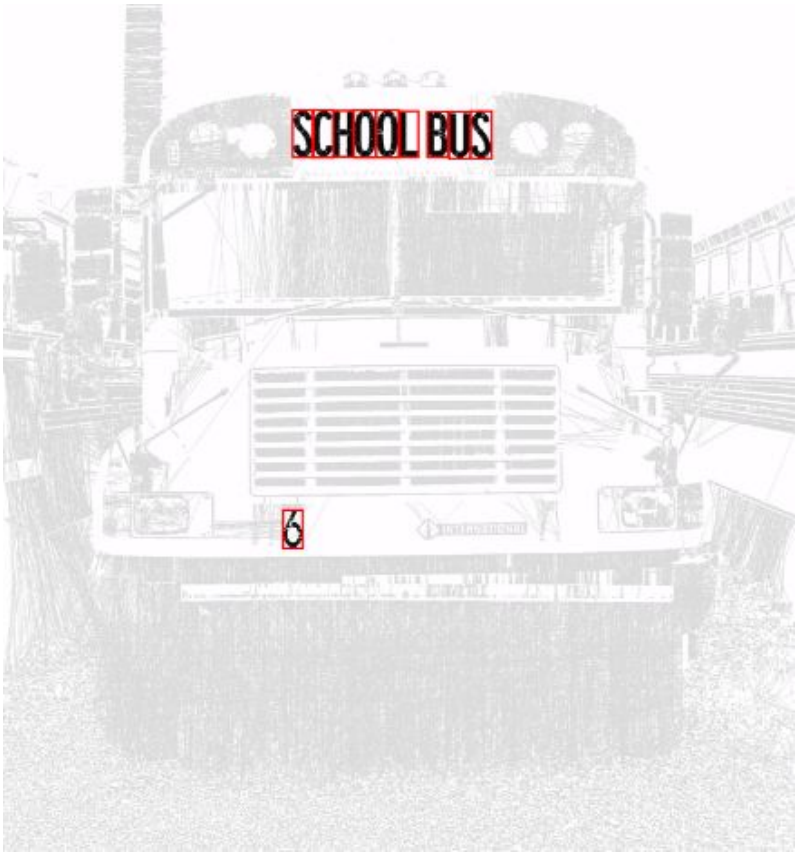

Fig: Bounding boxes for detected characters

Deep Text Recognition

We extracted the character region from the image and used deep feature to index the region and trained/classified using a multiclass SVM[5].

Dataset: We used char74k datasets[2] with 7705 natural images with 62 classes (0-9, A-Z, a-z).

Deep Feature Extraction: We used Overfeat library[3] to extract the deep features from the level 21 output of the trained CNN of Overfeat library[3].

Training: We trained a multiclass SVM[5] with the extracted deep features from the char74k datasets[2].

Classification: For each character region Overfeat library[3] extracts the features and trained SVM[5] model classify the character region into one of the 62 classes.

How to Run

The pre-trained SVM model are already present in the repository. It is possible to run the application directly without training.

Compile*: make final

Run*: ./final <input-image-name>

***Overfeat library[3] required:** To run the application, we need overfeat library[3] with downloaded weights in the project path. The overfeat library[3] location can be changed from the source code editing line 22 at main.cpp:

#define OVERFEAT_PATH "/../overfeat/bin/linux_64/overfeat"

The weights of the overfeat library[3] can be downloaded by running the script *download_weights.py* inside the *overfeat* directory.

Result:

Following figures show sample output of the project:

Fig: Detected and Recognized Text Characters

Fig: Detected and Recognized Text Characters

We can see that, the application sometime detects false positive. Also sometime wrong character is recognized due to similarity between them.

Future Work and Conclusion

To recognize the characters using deep learning instead of SVM which would eliminate the false positives, as deep learning would give a feedback how close the character is to our recognized character. We could eliminate false detection by ignoring low matches.

Research on cursive and light text on dark background. Currently the application cannot detect cursive text and light text on dark background.

References

1. B. Epshtein, E. Ofek, and Y.Wexler. Detecting text in natural scenes with stroke width transform. In CVPR, 2010.
2. http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/
3. http://cilvr.nyu.edu/doku.php?id=software:overfeat:start
4. J. Canny, “A Computational Approach To Edge Detection”, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.
5. https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html
6. PhotoOCR: Reading Text in Uncontrolled Conditions. Alessandro Bissacco, Mark Cummins, Yuval Netzer, Hartmut Neven. In ICCV 2013.