Replication of "Real-time Scene Text Localization and Recognition" and "Text Localization in Real-world Images using Efficiently Pruned Exhaustive Search"

Daniel DeTone
EECS 592, W14
Department of Computer Science and Electrical Engineering
University of Michigan at Ann Arbor, MI
ddetone@umich.edu

Abstract

I present a replication of two papers: "Real-time Scene Text Localization and Recognition" [10] and "Text Localization in Real-world Images using Efficiently Pruned Exhaustive Search" [9]. The combination of these two papers by Neumann et. al. present a text localization system for scene images. [10] presents a method for generating potential character regions in the image, and [9] presents a method for combining these regions into words. Overall, even with the combination of two papers, this was a difficult project to replicate. This was to be expected to a certain extent as they are both papers from conference proceedings. In short, I was able to implement the majority of both papers, with the exception of a few portions of their system. I did however, gain great insights into the underlying algorithms while studying the system's vague details. In this paper I present potentially novel pseudocode for a key algorithm in [9], tips for which data structures to use for implementation of this system, and the identity of the key components of the papers which are required for full replication. Lastly, I present qualitative experimental results which I obtained without the aforementioned key details.

1. Introduction

Scene text localization and recognition is a challenging, open problem with many practical applications. Examples of practical applications of such systems include helping visually impared people, automated translation of text written in a foreign language, and automated indexing of images based on the textual content (e.g. Google Street View, Flickr, etc) [11].

Performing text recognition in scene text images is a more difficult problem than that typically seen in the Optical Character Recognition (OCR) setting, which is largely considered a mature field. This is because OCR systems are designed primarily for document images such as those from a flatbed scanner, where the text has a high contrast with the background and is horizontally algined. These systems tend to perform poorly on scene text images (uncontrolled images), where the input can contain blur, low resolution, and character deformations. OCR systems typcially rely on brittle techniques such as binarization, where the first stage of processing is a simple thresholding operation used to divide text and non-text pixels. It is for these reasons that text must first be localized in the image as a preprocessing step for OCR.

In this paper, I attempted replication of the text localization portion of [10]. In order to do so, it was also necessary for me to implement [9]. As these are both short conference proceedings, replication was difficult. This is understandable as the authors only have a short space to describe their work. There were a few critical details left out of the original papers, thus I was unable to fully replicate the results presented in [10]. Instead of full replication, this paper presents a compliation of my findings which should serve as supplementary material to another who wishes to replicate [10] and/or [9]. The primary contributions of this replication study are:

1. Psuedocode for potentially novel implementation of efficiently pruned exhaustive search, the critical algorithmic component in [9]
2. Detailed explaination of the concepts in [9] and [10], with illustrations to help simplify the reader's understanding of combining these works
3. The identification of details which are required from

the author for complete replication of [10] and [9]

2. Text Localization and Recognition State of the Art

State of the art scene text localization systems are generally split into two groups: the sliding window approach and the connected components approach.

Sliding window approaches [7], [6], and [13] limit the search to a subset of image rectangles. The primary strength of these methods are their robustness to noise and blur, because because they exploit features created throughout the entire region of interest. The primary drawback of these methods are their efficiency: the number of rectangles which need to be evaluated grows very rapidly when text with different scale, aspect ratio and rotations are considered.

The connected components approach is recently more popular [4], [5] and [12]. These methods search for individual characters by grouping pixels which possess similar characteristics into the same group. The primary drawback of these methods are their sensitivity to clutter and occlusions which change the connected component structre. The advantage to these methods are that their complexity typically does not depend on properties of the text such as a variety of scales, orientations and fonts. They also provide a character segmentation which is useful in the OCR stage.

The best performing end-to-end scene text detection and recognition system is [1]. In this method, text line region hypotheses are generated using three efficient localization methods tuned for high recall. Then, a cascade of classifiers are used make character proposals from the text line region hypotheses. Next, leveraging recent deep learning developments in machine learning, [1] trains a deep neural network for character classification. Finally, datacenter-scale distributed language modelling is used to synthesize the most probable letter combination. This system outperforms all previously reported results, more than halving the error rate on multiple benchmarks.

3. Text Localization Overview

3.1. Extremal Regions (ERs)

An extremal region is a region of pixels r in a scalar channel C for which every boundary pixel surrounding the region r have strictly higher values than the region r itself. We denote the threshold at which the outer boundary pixels have a strictly higher value than as θ. This is show in Figure 1.

The primary benefit of segmenting the text characters as extremal regions is their efficiency. Because of the inherent tree struture as shown in Figure 1, one can incrementally computer features of the region as the tree is traversed. This allows for calculation of region features in $O(N)$, where N is the number of pixels in the image. This is a unique feature of extremal regions which makes them especially appealing for real-time applications.

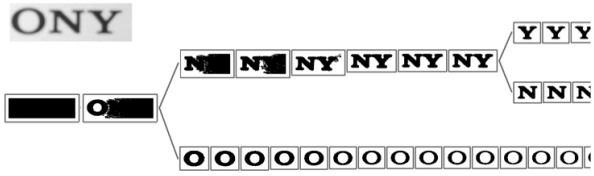


Figure 1. *Inclusion-exlusion of Extremal Regions*. This figure shows a subset of the extremal region tree generated by [10]. As one traverses down the tree, the threshold θ that defines the extremal regions decreases, creating smaller and smaller extremal regions. The input is the image in the top-left portion of the figure containing "ONY".

3.2. Exhaustive Search

In this section, an overview of the exhuastive search introduced in [9] is presented. Then, psuedocode for the algorithm along with an illustrated example are shown to help guide the reader to towards an efficient implementation of this concept is described.

Let E denote the set of ERs (extremal regions) found in image **I**. Performing extremal region filtering reduces the cardinality of the set S of all sequences made up of E from 2^{2^n} to 2^n. While this is a great reduction in complexity, it is still exponential, thus it is unnacceptable for a real-time text localization system.

We can thus define "upper-bound" verification functions $\hat{v}_1, \hat{v}_2, ..., \hat{v}_n$ which determine whether s^L is either a sequence or subsequence of size L letters:

$$\hat{v}^L(s^L) = 1 \iff \exists s' : s^L \subseteq s', v(s') = 1 \tag{1}$$

Where v is a verification function which is true if its input sequence is an unextendable word. It then follows that the enumeration of $\varepsilon = \{\omega \in S : v(\omega) = 1\}$ can be equivalently defined as finding the set of unextendable sequences:

$$\varepsilon^1 = \{r \in E | \hat{v}_1(r) = 1\} \tag{2}$$

$$\varepsilon^2 = \{(r_1, r_2) | r_1, r_2 \in \varepsilon^1, r_1 \neq r_2, \hat{v}_2(r_1, r_2) = 1\} \tag{3}$$

$$\varepsilon^3 = \{(r_1, r_2, r_3) | (r_1, r_2), (r_2, r_3) \in \varepsilon^2, r_i \neq r_j, \forall i, j, \hat{v}_3(r_1, r_2, r_3) = 1\} \tag{4}$$

$$\varepsilon^n = \{(r_1, r_2, ..., r_n) | (r_1, r_2 ..., r_{n-1}), (r_2, r\, 3 ..., r_n) \in \varepsilon^{n-1}, r_i \neq r_j, \forall i, j, \hat{v}_n(r_1, r_2, ..., r_n) = 1\} \tag{5}$$

By formulating the exhaustive search in this way, we can prevent the combinatorial explosion of enumerating the S^L sets of all sequences of length L. This is possible because non-text subsquences are excluded without the need to build a complete sequence.

3.3. Verification Functions

The verification functions $(\hat{v}_1, \hat{v}_2, ..., \hat{v}_n)$ are first introduced in [9]. When posing the text localization problem in the setting where extremal regions are the character hypotheses, as the authors do in [10], the $\hat{v}_1$ function is the cascade of sequential classifiers which classify the extremal regions. These verification function provide a critical function in that they must be very properly tuned to work properly. If they are too harsh (e.g. they prune too many functions), then often the text in the scene will not be detected. If they are not strong enough, then the computational complexity grows greatly and often makes text localization intractable.

3.4. Difficulties with Implementation

It is in describing these functions $\hat{v}_2, ..., \hat{v}_n)$ that the authors Neumann et. al. present the greatest difficulty to a replication of the their work. This is the major difficulty which I encountered. For example, in [10], they describe the rules for the $\hat{v}_2$ with a single sentence: *The rules require that height ratio, centroid angle and region distance normalized by region width fall within a given interval obtained in a training stage*. While these three rules that they present seem reasonable, a full replication requires, at the minimum, a short description of what the features were, how they were determined and labeled, and which learning algorithm was used. Additionally, a short sentence in the experiments section of the paper would be very helpful. When implementing this portion of the paper, I chose thresholds manually which seemed to give decent results.

Additionally, in description of their $\hat{v}_3$ verification function, they provide only a short phrase in parenthesis *mutual vertical distance of the text lines is constrained based on thresholds in training*. I did not understand this phrase, and was unable to implement the $\hat{v}_3$ verification function. I attempted contact with authors, but have gotten no response.

4. Implementation Tips

4.1. Data structure for Extremal Regionss

First, one must consider memory efficient methods for storing all the regions $r \in E$. Namely, we aim to answer: what is the minimum amount of data that needs to be stored to reconstruct after finding $\varepsilon^1, \varepsilon^2, ..., \varepsilon^n$ to form word lines? The naive approach would be to store the source pixels for which r contains. While this would still keep the memory complexity $O(cN)$, where c is the number of channels used to generate E and N is the number of pixels in the source image, we can reconstruct r with a smaller footprint. The three pieces of information we need to encode are:

1. Root pixel location
2. Threshold at which the ER was found
3. The channel used to find the ER

Using this information to store the extremal regions results in a memory footprint of $O(R)$, where R is the number of regions which in practice is much smaller than the naive approach as rarely are extremal regions' sizes close to one pixel. To find the connected component from these three pieces of information, one can use an efficient flood-fill algorithm as implemented in [3] which has linear run-time efficiency.

This storage also becomes critical when doing comparisons of extremal regions, as these are the three characteristics which must be unique for an extremal region to be unique. This is discussed further in both section 4.2 and section 4.3.

4.2. Comparison of Sequences of ERs

One critical component of the exhaustive search is subsequence comparison, which become apparent in the iterative definition 2, especially in 5. In 5 it is neccesary to find the sequence:

$$S^{mid} = \{(r_1, r_2, ..., r_{n-1}) \cap (r_2, r_3, ..., r_n)\} \quad (6)$$

Where $r_1, r_2, ..., r_n$ are defined in 5. Since the intersection has $n!$ combinations, where n is the length of each sequence of ERs, a naive approach making all these comparisons can be intractable. Instead, if one implements the sequences of ERs in an ordered set, one can implement linear comparison assuming that one has stored each ER according to 4.1. Binary sort is fast, and can be implemented in $O(nlog(n))$ time.

4.3. Algorithm for Subsequence Comparison

The naive approach to comparing each sequence to other in order to generate 5 is to iterate through the elements in ε_n with a nested loop, resulting in $O(NM^2)$ runtime complexity, where N is the number of sequences of sequence length M. One can instead efficiently enumerate the possible sequence combinations in 5, by storing element (r_{i1}) with the sequence $(r_{i2}, ..., r_{in})$ as its key in a multiple element hash map for each element in ε_n. This results in an efficency of $O(NM)$. As the authors of [9] did not provide implementation details for their algorithm, this may be a novel algorithmic approach to solving the sequence comparison problem. See Algorithm 1 for more details.

5. Experimental Results

The work of [10] presents and end-to-end scene text recognition system which both localizes text and performs recognition of the found text. This replication study focuses on the localization portion of the system in [10].

As stated on the primary author's webpage [8], a portion

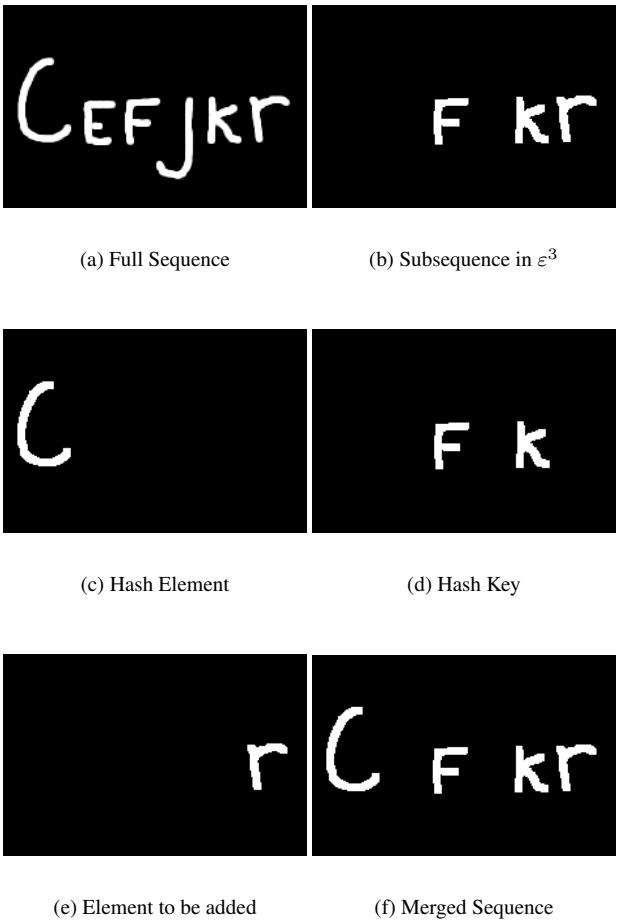
Algorithm 1: Efficient Subsequences Comparison using Multi Map Hashing

Data: ε^2 Set of all pairwise ERs for which $\hat{v}_2(r_1, r_2) = 1$
Result: $\hat{\varepsilon}$ Set of all enumerated sequences of ERs

initialize: $m \leftarrow 2$;
while ε^{m-1} *is not empty* **do**
  intitialize: MultiHashMap;
  foreach $s_i = (r_{i1}, ..., r_{in}) \in \varepsilon^m$ **do**
    add key $(r_{i2}, ..., r_{in})$ with elt r_{i1};
  end
  foreach $s_i = (r_{i1}, ..., r_{in}) \in \varepsilon^m$ **do**
    if *key* $(r_{i1}, ..., r_{in-1})$ found *in MultiHashMap* **then**
      $new\_elt \leftarrow merge(r_{found}, (r_1, ..., r_{in}))$;
      if $\hat{v}_{n+1}(new\_elt)$ **then**
        add to ε^{m+1};
      end
    end
  end
  $m \leftarrow m + 1$;
end

of the work in [10] was implemented by Lluis Gomez as part of the developmental OpenCV 3.0 [2]. This code implements the extremal region filtering and sequential classifier described in Section 3 of [10], and provided the basis for this replication study. Using this code is straightforward for those familiar with OpenCV.

The above mentioned code generates a large set of connected components from 9 different scalar channels. To generate words from these potential characters, on must form sequences of extremal regions as described in [10]. There was no open source implementation of this work, thus I implemented it all from scratch using C++. The majority of the time I spent in replicating this paper was spent implementing 1 in an efficient manner. The two works are replicated, with the exception of the $\hat{v}_3$ function. This function must be critical for achieving real-time performance, as this was the biggest difficulty for me. Due to this high computational time, it was impossible to evalute the algorithm on the ICDAR 2011 dataset, as the algorithm took too long to complete on very large images. Figure 5 shows some qualitative results of my replication on smaller images. The entirety of the C++ code which I used was submitted to the Ctools Drop Box.



(a) Full Sequence (b) Subsequence in ε^3

(c) Hash Element (d) Hash Key

(e) Element to be added (f) Merged Sequence

Figure 2. *Addition of sequence to* ε^4. Figure (a) shows the fully enumerated word ω. Figure (b) shows the subsequence in ε^3 which may be potentially merged with another element in ε^3, made up of the union of sequences in Figure (c) and (d). Figure (c) shows the sequence used as the hash key. Figure (d) shows the element stored at the hash key in figure (c) (not unique). Figure (e) is the element to be added to the union of Figure (c) and (d). Figure (f) is the merged sequence as the key in Figure (c) was found in the map.

References

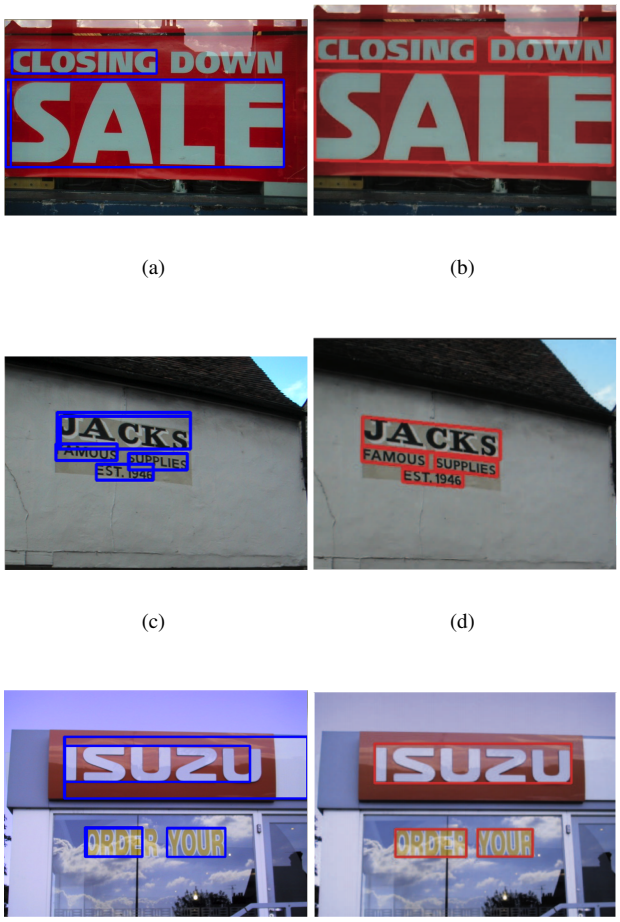
[1] A. Bissacco, M. Cummins, Y. Netzer, and H. Neven. Photoocr: Reading text in uncontrolled conditions. In *ICCV*, pages 785–792, 2013. 2

[2] G. Bradski. *Dr. Dobb's Journal of Software Tools*, 2000. 4

[3] S. V. Burtsev and Y. P. Kuzmin. An efficient flood-filling algorithm. *Computers and Graphics*, 17(5):549–561, 1993. 3

[4] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In *CVPR*, pages 2963–2970. IEEE, 2010. 2

[5] Y. feng Pan, X. Hou, and C. lin Liu. Text localization in natural scene images based on conditional random field, 2009. 2



(a) (b)

(c) (d)

(e) (f)

Figure 3. *Text Localization Results*. The images on the left hand side (blue bounding boxes) of this figure represent the output from my implementation, and the images on the right hand side (red bounding boxes) represent the output of the original implementation of [10] and [9].

[6] J. jin Lee, P. hean Lee, C. Koch, and A. Yuille. Adaboost for text detection. In *in Natural Scene, International Conference on Document Analysis and Recognition*, pages 429–434, 2011. 2

[7] R. Lienhart and A. Wernicke. Localizing and segmenting text in images and videos. *IEEE Trans. Circuits Syst. Video Techn.*, 12(4):256–268, 2002. 2

[8] L. Neumann. Center for machine perception - lukas neumann @ONLINE, June 2013. 3

[9] L. Neumann and J. Matas. Text localization in real-world images using efficiently pruned exhaustive search. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 687–691, IEEE Computer Society Offices, 2001 L Street N.W., Suite 700 Washington, DC 20036-4928, United States, sept. 2011. IEEE Computer Society Conference Publishing Services. 1, 2, 3, 5

[10] L. Neumann and J. Matas. Real-time scene text localization and recognition. In *CVPR*, pages 3538–3545. IEEE, 2012. 1, 2, 3, 4, 5

[11] L. Neumann and J. Matas. Scene text localization and recognition with oriented stroke detection. In *2013 IEEE International Conference on Computer Vision (ICCV 2013)*, pages 97–104, California, US, December 2013. IEEE. 1

[12] C. Shi, C. Wang, B. Xiao, Y. Zhang, and S. Gao. Scene text detection using graph model built upon maximally stable extremal regions. *Pattern Recogn. Lett.*, 34(2):107–116, Jan. 2013. 2

[13] K. Wang, B. Babenko, and S. Belongie. End-to-end scene text recognition. *Computer Vision, IEEE International Conference on*, 0:1457–1464, 2011. 2