Scene Text Recognition

Vincent BODIN & Thomas MOREAU

Abstract

Scene text recognition refers to finding automatic ways of extracting text in pictures of everyday life. As computer vision is more and more successful, scene text recognition has naturally become a significant issue nowadays. Unlike OCR which is more or less well understood and implemented, scene text recognition still needs many improvements to be considered as a powerful tool. We implemented a paper [1] that explains how to extract words efficiently. A graphical model is introduced inside, with the creation of graph representing possible detected words. The algorithm TRW-S [2], which is a slightly different version of Belief Propagation (BP) is used to extract the optimal words.

1 Introduction

Understanding scenes semantically has gained considerable consideration for those last few years with the successes of computer vision. In particular, a certain attention has been granted to street scenes, whether it takes the form of object detection, object recognition or segmentation. We decided to consider a slightly different problem, though the methods may be quite similar. We decided to focus on text recognition in streets. It is indeed a project that has potentially considerable repercussions in our everyday life.

For this, we decided to implement a paper [1] that details a method to retrieve in natural pictures some characters and interpret it as words. Their results seem to be the highest reached till now in this field of computer vision and it includes some graphical models and algorithm for minimizing a discrete energy on a graph (TRW-S, [2]). It mainly requires two steps:

Step 1: Detect all the possible characters in the picture. We should be very exhaustive in here, we do not want to miss any character because that would basically hamper us from retrieving any word. This character detection is possible through machine learning methods. We learn K SVMs - one-versus-all method and we learn one SVM for each character - and then we scan the image and test for each patch whether there can potentially be a character or not.

Step 2: Among all those possible characters, we try to prune and reconstruct words. From all the sliding windows remaining after this process, we build a graph that takes its value in $\mathcal{K}_\epsilon$ the set of all characters plus a void label to discard the false positives. We then assign an energy on this graph depending on unary energies of each nodes and pairwise energies. Then we will run an algorithm to extract a lower bound of the energy minimization problem - which is NP-hard.

In this project we will essentially be interested in the second step of the project, the first one being implemented in *Object Recognition and Computer Vision* course. We present in Fig.(1) the task we are concretely given: extract texts. This image is from SVT (Street View Text) but is not performed automatically, that is what we aim.

Figure 1: Task

2 The graphical model construction

2.1 Learning characters and words

Learning characters. We need to build K classifiers ($K = 62$ in English) for each character to recognize them in natural pictures. For this purpose, we use the following methods:

- Use a database to identify characters: ICDAR 2003 [3], Chars74K [4];
- extract features: Histogram Of Gradient (HOG) [5]. They extract the information of a patch by computing the gradient on bins of sizes 4×4, saved in an histogram of 12 orientations;
- build K SVMs (K is the number of classes, 62 in English) with RBF Kernel, Fig.(2):

$$\exp(-\gamma|x - x'|^2), \gamma > 0 \tag{1}$$

We use the one-versus-all method, we used a Python library **scikit-learn** [6]. There are two parameters to optimize:

1. γ: parameter that contains the information of how local we want to consider the SVM, the biggest γ, the smallest the radius of the RBF Kernel.
2. Tikhonov regularization term C: penalizes big norm vectors.

We ran a 14-hour cross validation on an exponential search-grid: test error fell from 25% without any optimization of parameter by 16% after a 6-fold-cross-validation. By test error we mean the error while running our classifier on the training set. **The error is still very high and needs a lot of improvement.**

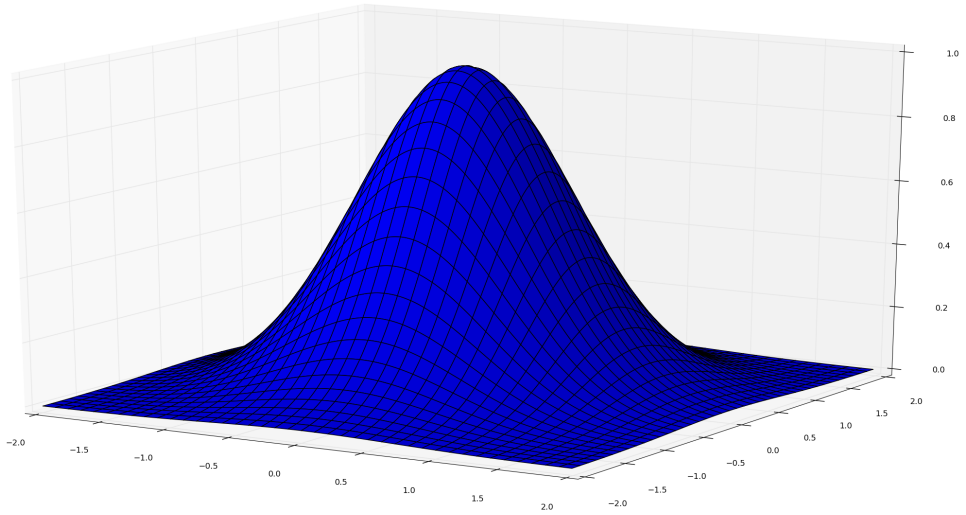

Figure 2: RBF Kernel

Learning words. We have to build a prior lexicon that contains how characters interact to create words, for this:

- Load a database of words [7];
- count for each pair of characters (c_i, c_j) their frequency of occurrence $p(c_i, c_j)$ in the database (bi-gram model);

This frequency of occurrences will be used to penalize through an energy term the pair of words that are rare in the dictionary (*e.g.* 'xz').

2.2 Character detection: sliding windows

The first task is to detect all possible characters in the image, this is done through the sliding windows algorithm:

Algorithm 1 Sliding Window scanning

Input: Image
Output: l list of characters
  l = [ ];
  for all windows in Image **do**
    $p \leftarrow$ svm.predict_proba
    GS $\leftarrow \max_c p(c|\phi_i) \exp\left(-\frac{(a_i-\mu_{a_j})^2}{2\sigma_{a_j}^2}\right)$
    if GS > 0.1 **then**
      l.append(window)
    end if
  end for

We did not specify in here the grid we took for the width and the height of the windows: for the moment, to avoid too many detection, we rather tune it by ourselves, giving a range close to the actual size of the letters in the image.

Having all those windows, many of them may represent the same character and we need to merge them. This is done by a specific Non-Maximum Suppression (NMS) for each character.

Algorithm 2 Non-Maximum Suppresion (NMS)

Input: l list of windows, c character with maximum probability for each window, threshold
Output: l_p pruned list

```
while l is not empty do
    w_1, c_1 pair of characters in l with highest probability
    m = [w]
    for w_2, c_2 in l, c do
        if criterion > threshold and c_1 == c_2 then m.append(w_2)
        end if
    end for
    l_p.append(mean(m),p)
end while
```

2.3 Graph construction

Given the n remaining windows we build a graphical model that will contain the underlying information of how related are the letters in our image. The key idea is that two letters apart should not be close in the reconstruction of a word and might not even have any influence one on another. First we should ensure that we reorganize our data in a lexicographic way so that we only put links between a node and the one that are very close to it as we will see above. This is the main preprocessing needed.

Step 1: For each sliding window kept, assign a random variable X_i that takes labels $x_i \in \mathcal{K}_\epsilon$ (add ϵ a void label).

Step 2: When two sliding windows are 'close enough' build an edge between the nodes.

Step 3: Assign CRF energy.

The energy is specified as a unary energy that penalizes a single character and a pairwise energy that takes into account how the letters interact to ceate words. The unary energy is:

$$\begin{aligned} E_i(x_i = c_j) &= 1 - p(c_j|x_i) \\ E_i(x_i = \epsilon) &= \max_j p(c_j|x_i) \exp\left(-\frac{(\mu_{a_j} - a_i)^2}{2\sigma_{a_j}^2}\right) \end{aligned} \tag{2}$$

As for the pairwise energy, they should penalize pairs of letters that do not occur often in the lexicon, the form is the following:

$$\begin{aligned} E_{i,j}(x_i = c_i, x_j = c_j) &= E^l(x_i, x_j) - \lambda_0 \exp\left(-\psi(x_i, x_j)\right) \quad (\forall c_i \neq \epsilon, c_j \neq \epsilon) \\ E_{i,j}(x_i = c_i, x_j = \epsilon) &= \lambda_0 \exp\left(-\psi(x_i, x_j)\right) \\ E_{i,j}(\epsilon, \epsilon) &= 0 \\ \psi_(x_i, x_j) &= (100 - \text{overlap}(x_i, x_j))^2 \\ E^l(x_i = c_i, x_j = c_j) &= \lambda_l(1 - p(c_i, c_j)) \qquad \text{(lexicon prior)} \end{aligned} \tag{3}$$

So that we eventually obtain an energy on the graph $E : \mathcal{K}_\epsilon^n \to \mathbb{R}$ of the form:

$$E(\mathbf{x}) = \sum_{i=1}^{n} E_i(x_i) + \sum_{\mathcal{E} \text{ edges}} E_{i,j}(x_i, x_j) \tag{4}$$

Hence we are led to:

We have to minimize the energy over the graph, which is a NP-hard problem, we will relax it!

3 TRW-S algorithm

3.1 Preliminaries

Introduced by V. Kolmogorov [2], it minimizes a discrete energy on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$:

$$E(\mathbf{x}|\theta) = \theta_c + \sum_{s \in \mathcal{V}} \theta_s(x_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(x_s, x_t) \tag{5}$$

For a general graph, it is NP-hard. In [8], *belief propagation* was described - which gives the exact optimum for trees, but might get stuck into a loops otherwise. In [9], *max-product tree-reweighed message passing* (TRW) is developed. *Sequential tree-reweighted message passing* (TRW-S), [2] is better because the bound is tighter (and never decreases).

3.2 Belief Propagation, [8]

It is dynamic programming in linear time that is exact for trees. Denote by $\mathcal{X}_s$ the labels that can take node s, and $x_s \in \mathcal{X}_s$.

Step 1: Messages are going from leaves to root:

$$M^{s \to t}(x_t) = \min_{x_s} \left(\theta_s(x_s) + \theta_{s,t}(x_s, x_t) \right), \quad x_t \in \mathcal{X}_t \tag{6}$$

Step 2: Wait for the node t to receive all messages from its children $\mathcal{T}$ and send messages from intern nodes to other intern nodes:

$$M^{t \to u}(x_u) = \min_{x_t} \left((\theta_t(x_t) + M^{s \to t}(x_t)) + \theta_{t,u}(x_t, x_u) \right), \quad x_u \in \mathcal{X}_u \tag{7}$$

Step 3: Wait for the node u to receive all messages from its children $\mathcal{U}$, and then send a message to its parent. Iterate till we reach the root node.

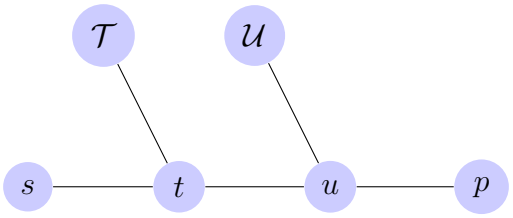

Figure 3: Message passing in BP: first from s to t and from π_t to t

Step 4: Outward pass from the root to all the leaves.

This algorithm allows us to compute min-marginals on a tree for, say node u and label j:

$$\min_{\mathbf{x}} \{E(\mathbf{x}) | x_u = j\} = \theta_u(j) + M^{p \to q}(j) + \sum_{q \in \mathcal{U} \cup \{t\}} M^{q \to u}(j) \tag{8}$$

In a general graph, we use the same rules but it might not converge.

3.3 TRW-S algorithm

The energy minimization is NP-hard, so we relax the constraints:

$$x_p \in \{0, 1\} \Rightarrow x_p \in [0, 1] \tag{9}$$

and we try to solve the dual problem of this new linear program problem. For this we formulate lower bounds on the function and maximize the bound. Improvement of TRW-S is that **the bound never decreases**. Recall that our goal is to compute:

$$\Phi(\mathbf{x}) = \min_{\mathbf{x}} E(\mathbf{x}|\theta) \tag{10}$$

The key idea is to:

1. Split θ in $\theta_1 + \theta_2 + \cdots$;
2. Compute minimum for each component:

$$\Phi_i(\mathbf{x}) = \min_{\mathbf{x}} E(\mathbf{x}|\theta_i) \tag{11}$$

3. Combine the θ_i to get an actual bound on Φ.

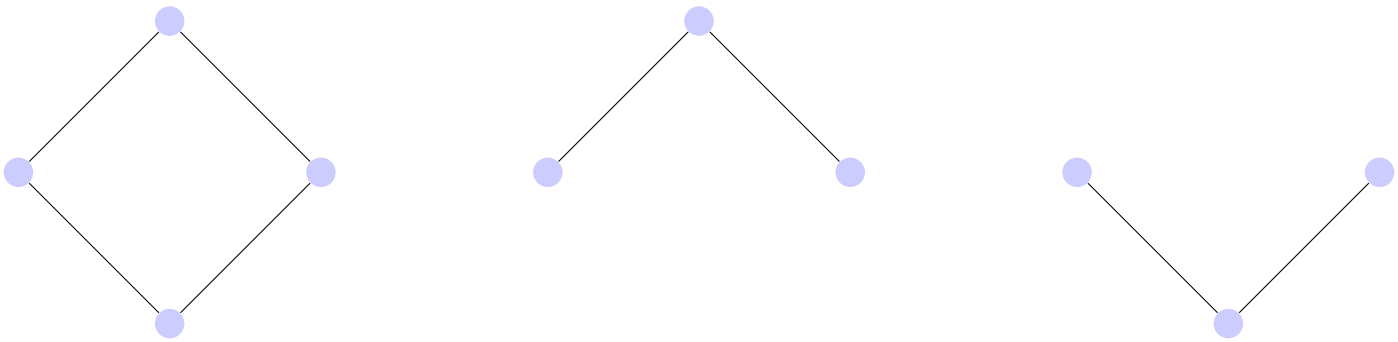

Figure 4: Decomposition of a graph $\mathcal{G}$ into trees T and T'

We can re-write the parameter θ as a combination of θ^T defined on the trees T and T':

$$\begin{aligned} \theta &\equiv \frac{1}{2}\theta^T + \frac{1}{2}\theta^{T'} \\ \Phi(\theta) &\geq \frac{1}{2}\Phi(\theta^T) + \frac{1}{2}\Phi(\theta^{T'}) \end{aligned} \tag{12}$$

If we maximize the term on the right, we get a so-called 'lower bound' on the energy. The first idea is now to run a BP algorithm on each tree, but that does not impose that the energy never decreases. Hence the TRW-S makes up for this issue by **node averaging**.

Say for a node $s \in \mathcal{V}$ we have $s \in T \cap T'$ and say the value in s is in T (4,1) and in T' (0,2), then when we recombine the initial graph, we associate to s (2,1.5) (if s is not in any other $\tilde{T}$). Here is the idea of TRW-S:

Algorithm 3 TRW-S algorithm

Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with energy $E(\mathbf{x}|\theta)$
Output: E^* lower bound on the energy
  decompose $\mathcal{G}$ into trees $T \in \mathcal{T}$ entirely covering $\mathcal{G}$.
  for $p \in \mathcal{V}$ **do**
    Run BP on all trees T such that $p \in T$
    Average node p
  end for

It is proved that the lower bound never decreases this way.

As for implementation, running BP on all trees is inefficient because we can reuse on the order of nodes in BP to get a faster algorithm. It suffices to consider a smart order on the tree T so that p can be computed with just the inward pass, take the root. Then the algorithm is in $O(n)$ for each BP run.

4 Implementation and Results

The implementation has been done with python2.7. The svm part use the implementation of the library **scikit-learn**. We have used the implementation of TRW-S of the libraries**Opengm** and **TRW-S.1-3** you can find on the web.

To tune the SVM parameters, we have used a per-character grid-search over the parameters of regularization C and the localization of our RBF kernel γ. This allows us to gain a character detection of 10% while testing on our training set.

We use a simple process to test our implementation. We started with the simplest case of OCR, with a printed word on a white background, already localized. We then used harder images as our results got better.

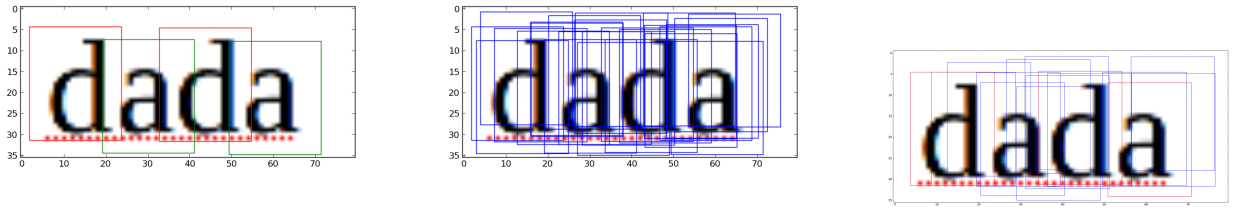

Figure 5: Test on an artificial word 'dada'. Word retrieved: 'dda' or 'dd depending on the lexicon prior used

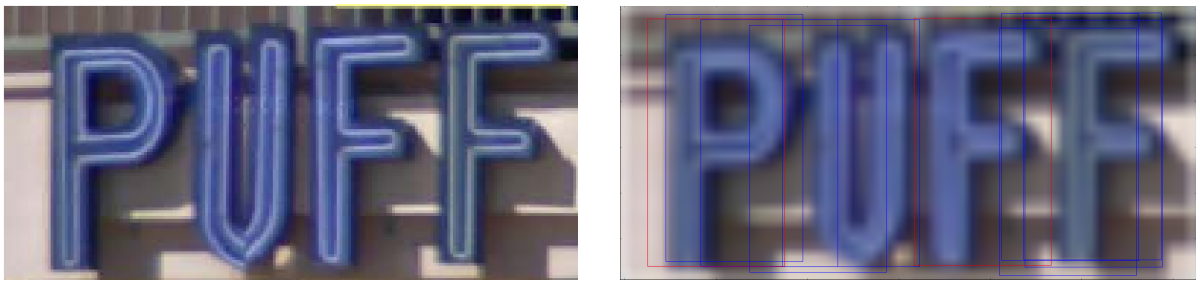

Figure 6: Natural testing word. Word retrieved: 'PE'

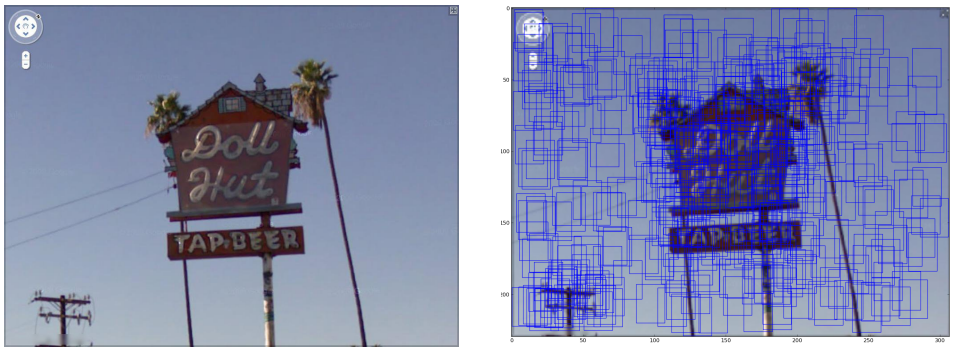

Figure 7: Natural testing image

We can see in the Fig.(5) that most of the time there are way too many characters that are detected but we are 'practically' able to select the correct ones with the graphical model. In the Fig.(6), we can see two problems. The first one is the F is detected as a E by our SVM because of shadows perhaps. Then, the graphical model penalizes the overlap too much and we end up with separated characters instead of a word detection. Indeed the energies introduce a term increasing as the overlapping gets bigger, to indeed avoid finding say a r instead of a n (very close is juste move left the sliding window) as the NMS is character specific. But it seems the tuning of those parameters is tricky as penalizing too much the overlapping leads to single separated characters instead of a whole word. The Fig.(7) shows that with bigger images and multiple scale, we got a graphical model too complicated and lost the intrinsic information.

5 Improvements

- The SVM performance is very low here even after a large cross-validation process. A first step would be to build a more robust character detection. One idea would be to pre-select the sliding windows based on the score of a character detector model. This could still be imprecise as there is a big variability over the characters.
- Another observation we can make is that the model has no term in the energy that encourages words bigger than one letter in the energy. We can often observe that the resulting word are single separated characters whereas we are looking for a whole word.

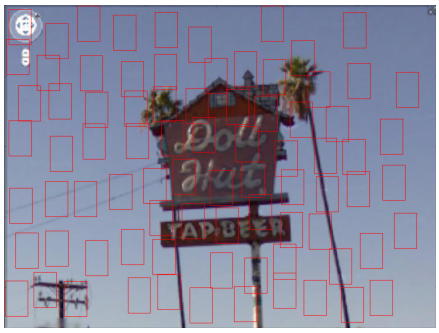

Figure 8: Spreading of the detected characters

- The prior we use here is a bi-gram prior. It only take into account the succession of letter. One information we are not using is the fact that a 'P' is more frequent at the beginning of a word that at the end. We could thus follow the idea given in the paper [1] to build a Node-specific prior. The idea is to compute for each graphical model a prior based on the place of the node in the image and in the graphical model. But this seem to be only possible after a cleaning of the graphical model in the first step. We should reduce the false positive rate and then extract all the connected components of our graph to build our lexical prior in this context.

References

[1] C. V. Jawahar A. Mishra, K. Alahari. Top-down and bottom-up cues for scene text recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[2] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Learning (PAMI)*, 2006.

[3] Robust ocr. `http://algoval.essex.ac.uk/icdar/Datasets.html`. Accessed: 2013-12-16.

[4] The chars74k dataset: Character recognition in natural images. `http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/`. Accessed: 2013-12-16.

[5] B. Triggs N. Dalal. Histogram of oriented gradients for human detectection. *CVPR*, 2005.

[6] Scikit-learn. `http://scikit-learn.org/stable/modules/svm.html#parameters-of-the-rbf-kernel`. Accessed: 2013-12-16.

[7] Robust word recognition. `http://algoval.essex.ac.uk/icdar/Datasets.html`. Accessed: 2013-12-16.

[8] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.

[9] A. S. Willsky M. J. Wainwright, T. S. Jaakkola. Map estimation via agreement on trees: Message-passing and linear-programming approaches. *IEEE Transactions on Information Theory*, 51:3697 – 3717, November 2005.