Top-Down and Bottom-up Cues for Scene Text Recognition, A. Mishra, K. Alahari, C. V. Jawahar

Vincent BODIN & Thomas MOREAU

Abstract

Scene text recognition refers to finding automatic ways of extracting text in pictures of everyday life. 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 [?] 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 [?], which is a slightly different version of Belief Propagation (BP) is used to extract the optimal words.

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 as it can be useful in everyday life. 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 [?] 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.

1 Learning preliminaries

1.1 Learning characters

The first task we have to deal with is extracting characters from a natural image. This is done through a specific classifier learned from the databases ICDAR 2003 [?] and Chars74K [?]. From each patch of character, we resize it to a 22×20 window from which we extract a Histogram Of Gradient (HOG) [?] with 12 bins and 4 windows. It empirically seemed to give the best results. We also record the aspect ratio of each patch.

We then need to train our classifier, this is done through a one-versus-all procedure. It yields to $K = |\mathcal{K}|$ classifiers where $\mathcal{K}$ is the set of all possible characters. As we deal with the Western language, the cardinality of $\mathcal{K}$ is equal to 62. The SVMs are trained with a Kernel, the so-called RBF kernel $\exp(-\gamma|x - x'|^2), \gamma > 0$. It allows to distort the euclidean metric to catch local changes. We used a Python library scikit-learn [?]. The issue with this method is the number of parameters we have to deal with: K coefficients γ and regularization C - one for each classifier. We performed an exponential grid-search cross-validation to find the 'almost' optimal values of those coefficients. Yet, the test error only fell from a 25% to 16% with tuned parameters. We will talk about this later on.

1.2 Learning words

The second task for the learning process is to build a lexicon prior, *i.e.* learning how letters interact to create words. The whole idea of this step is to be able to determine among a large number of word possibilities which one is the most likely to be written. The database used was ICDAR 2003 word [?]. We used the bi-gram method which consists in learning how pairs of letters interact together, its empirical frequency ratio $p(c_i, c_j)$. This probability will be re-used afterwards in an energy term penalizing the pairs that might not occur frequently.

2 Character detection

Character detection is done via a sliding window scanning, see Alg.(1). The idea is to scan the image with different aspect ratio patches, denoted by a_i for the scanning window l_i. Then we use our K classifiers to determine for each character if this patch might contain this character or not. A goodness score is attributed according to the following formula:

$$GS(l_i) = \max_c p(c|\phi_i) \exp\left(-\frac{(a_i - \mu_{a_j})^2}{2\sigma_{a_j}^2}\right) \tag{1}$$

where μ_{a_j} is the mean and $\sigma_{a_j}^2$ the variance of the aspect ratio (learned from the database). This GS is made of two parts: the first one is meant to determine if there is a probability that a character is in this patch, this is why we take the maximum of the probability of characters. This needs nonetheless to be penalized by the aspect ratio learned from the database: if a l is found in a window with a large width an small height, we might not want to keep it since it might not be a real l.

Algorithm 1 Sliding Window scanning

Input: Image
Output: l list of characters

```
l = [ ];
for all windows in Image do
    p ← svm.predict_proba
    GS ← max_c p(c|φ_i) exp(−(a_i−μ_{a_j})^2 / 2σ_{a_j}^2)
    if GS > 0.1 then
        l.append(window)
    end if
end for
```

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, given in 2.

3 Recognizing words

Given n remaining windows we build a graphical model that will contain the underlying information of how related the letters are 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 on one 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 pre-processing needed.

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

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 create 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:

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

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 minimizing the energy over the graph. This is a NP-hard problem and we will use an algorithm that provides an approximation: TRWS-S algorithm. We do not detail it here since it is the subject we dealt with in PGM course.

4 Implementation and Results

The implementation has been done with python2.7. The SVM part uses 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 scale 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.

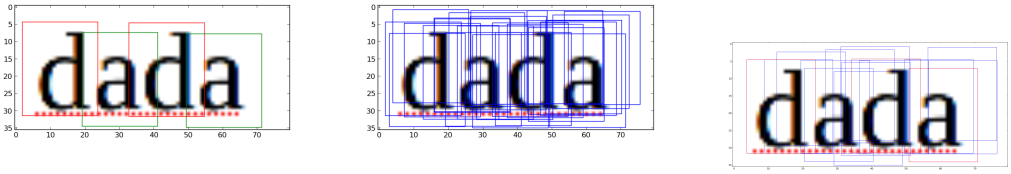

Figure 1: Test on an artificial word 'dada'. Word retrieved: 'dda' or 'dd depending on the lexicon prior used

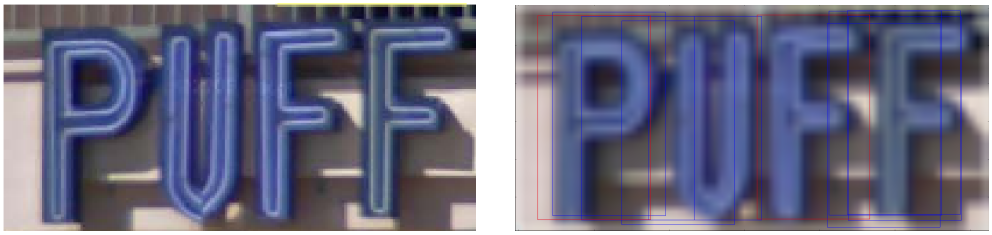

Figure 2: Natural testing word. Word retrieved: 'PE'

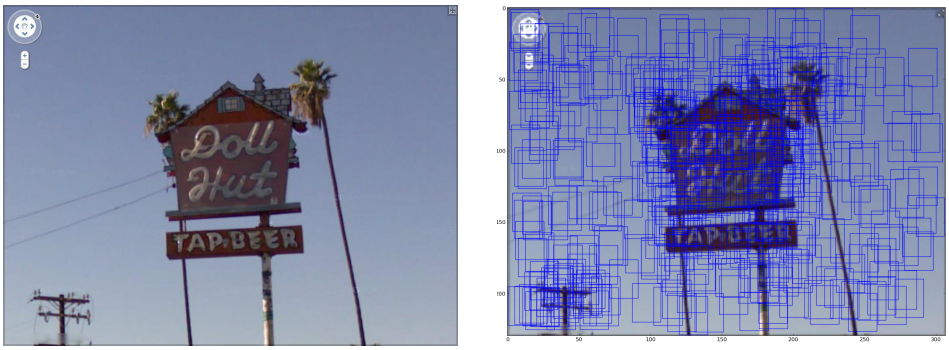

Figure 3: Natural testing image

We can see in the Fig.(1) 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.(2), we can see two problems. The first one is the F is detected as a E by our SVM because of shadow 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 (just 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.(3) shows that

with bigger images and multiple scale, we got a graphical model too complicated and lost the intrinsic information.

5 Improvements

We detail in this last section what could be done to improve still a little bit our detection:

- 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 thing we tried was to create by ourselves negative samples from the same images as those used to perform the detection. We thought that having other negative samples that all the other characters could be a good idea, but it did not change a single thing except the time of computation that increased dramatically.

- Another observation we can make is that the model has no term in the energy that encourages words bigger than one letter. We can often observe that the resulting word are single separated characters whereas we are looking for a whole word.

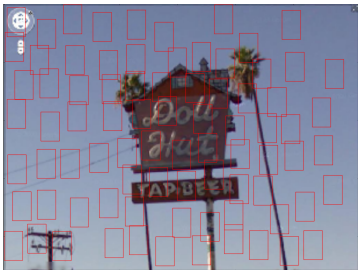

Figure 4: Spreading of the detected characters

- 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 [**?**] 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 seems 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.