Text detection and recognition in natural images

Steven Bell
Stanford University
sebell@stanford.edu

Abstract

Natural scenes pose a major challenge to traditional optical character recognition methods because they often contain noise, occlusions, distortions, or relatively small amounts of highly styled text. In this work, we build a probabilistic system which unifies the tasks of text detection and recognition with a language model. We use an efficient multi-scale character detector to locate characters within an image without performing segmentation. This is followed by a graph-based search which groups the detections into words and evaluates their relative probabilities, avoiding binary decisions except where computationally necessary.

1. Introduction

The modern world is filled with text, which humans use effortlessly to identify objects, navigate, and make decisions. Although the problem of reading text with a computer has been largely solved within limited domains – such as identifying license plate numbers, digitizing books and documents, or reading handwritten mailing addresses – the more general question of how to detect and read text in natural, unstructured images remains an open challenge.

The applications for such a system are numerous. For example, buildings photographed in Google's Street View could be automatically tagged with their address if a computer could detect and read the building numbers. Likewise, robots could be made to more intelligently act in a world annotated for humans, and translation systems for blind persons or foreign travelers would be improved.

Several factors make reading text in natural scenes particularly difficult. In unstructured scenes, such as a photograph of a storefront, clutter dominates the image. Text regions are of varying size and orientation, scattered, and possibly occluded. Letters may appear in multiple fonts and some letters may be ambiguous based on their shape alone.

Many approaches to this problem treat the recognition problem as two separate tasks, first locating text and drawing a bounding box, and then binarizing and reading it with traditional OCR methods. By forcing binary decisions at several points, this approach forfeits information which could be used to produce a more accurate result.

This project uses a probabilistic framework where image regions are assigned probabilities of containing different characters, and the result is read with the help of language characteristics, specifically the co-appearance of letters.

To limit the scope of the project, we considered only horizontal strings of upright lowercase letters. We worked primarily on synthetic images with automatically generated ground truth, which reduced the time spent creating training data.

2. Prior work

Optical character recognition for printed documents and handwritten digits has a long history. Many methods have been proposed and used successfully, but most of these assume that the input image has been cleanly binarized and that characters can be segmented easily, which is rarely the case in unstructured images. Because of the complexity of the input images, text recognition in natural scenes is more closely related to object detection than it is to traditional OCR.

A large body of past work has focused purely on the challenge of locating text within scenes, spurred primarily by the ICDAR text detection challenges of 2003 and 2011. These works can be roughly categorized into connected-component based methods, which segment the image and attempt to pick out which blobs are characters, and patch-based methods, which use convolutional filters, image patch correlations, wavelets, or other features to label the probability that an image patch contains text [1, 2].

Another domain has focused on the task of reading scene text given a bounding box, centered around the ICDAR robust reading challenge. Weinman et al. use a Bayesian model to unify text segmentation and reading, and show that reading accuracy can be improved by incorporating additional context such as language characteristics and the visual similarity of letters [3].

More recently, several groups have created end-to-end

detection and recognition systems. These use a variety of features for detecting and classifying characters, including Gabor filters, and connected-component dimensions and densities [4]. A support vector machine or linear discriminant analysis is used to perform classification based on the features. This work is an extension of these, where higher-level information is used to aid text detection, not just recognition.

3. Method

3.1. Character detection

The core of our algorithm is a multi-scale Histogram of Oriented Gradients (HOG) detector. The HOG descriptor is a patch-based algorithm which provides a dense estimate of shape over an entire image, using histograms of quantized gradient orientations for small image patches. It was originally introduced for pedestrian detection [5], but has since been applied to a wide variety of recognition tasks, including character classification [1].

Because it works on a images patches rather than on pre-segmented components, the descriptor is robust against characters which are accidentally split apart or joined together, which are difficult for a connected-component recognizer to handle. By using gradients rather than raw pixel values and by normalizing the resulting histogram, HOG is invariant to illumination and brightness changes. HOG inherently encodes position information, but also allows a degree of variance by virtue of its coarse spatial binning.

However, for HOG to work correctly, the sizes of the letters must roughly match the dimensions of the training examples, so the descriptor must be run at multiple scales. Additionally, characters have an extreme range of aspect ratios, which means that the detector must also run across a range of widths.

To detect characters, we use logistic regression on blocks of the HOG descriptor. Logistic regression was chosen since it is efficient to evaluate and provides a direct estimate of probability. We train and save a unique detector for each character, then the HOG descriptor is computed on the input image and each detector is run in a sliding window fashion across it. This produces a 2-D matrix of detection probabilities for each character. Points with probabilities meeting a threshold - generally far more than the number of characters actually present in the image - are carried forward to the next stage.

3.2. Line detection

Given a set of detections, the next step is to find the most likely lines of characters. We do this by taking weighted votes across all character detections. The vote V can be written

$$V(v) = \sum_{i=1}^{n} p(d_i) w(v - y_i)$$

where v is the pixel height, x_i and y_i are the detection position, $p(d)$ is the probability of a detection d (i.e, the result of logistic regression), and w is a window function weighted toward the center of the detection. We currently use a Hamming window, but a triangular or Gaussian window would also be appropriate.

An example of a detection and the corresponding line is shown in Figure 1.

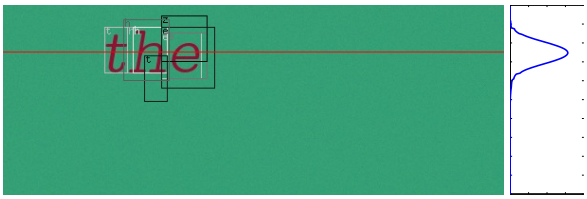

Figure 1. Detection of lines in an image. The left image shows the detections and the resulting line. The right shows the voting result.

3.3. Recognition and reading

Given a set of lines, we can determine the probability that each detected character belongs to that line based on its vertical position. Using a straightforward application of Bayes' rule we can write

$$p(l_{ij}|y_i) = \frac{p(y_i|l_j)p(l_j)}{p(y_i)}$$

where l_{ij} represents the probability that detection i belongs to line j, and $p(y_i)$ is the probability of detection i appearing at vertical position y_i.

Assuming that $p(y_i)$, and $p(l_j)$ are constants (i.e, that the positions of the characters and lines are uniformly distributed in the test images) the equation simplifies to

$$p(l_{ij}|y_i) \propto p(y_i|l_j)$$

The distribution on the right can easily be calculated using detected lines and ground truth bounding boxes for each character.

To read words, we scan horizontally along each line and build a directed graph of word possibilities. An empty root node begins the graph, and letters are successively added as the algorithm scans left to right.

Overlapping detections of the same letter are merged together, while overlapping detections of differing letters create multiple paths in the graph. Each node is connected to all of the possible letters immediately preceding it, that is, the nodes in the graph closest to the leaves which do not overlap. An example is shown in Figure 2.

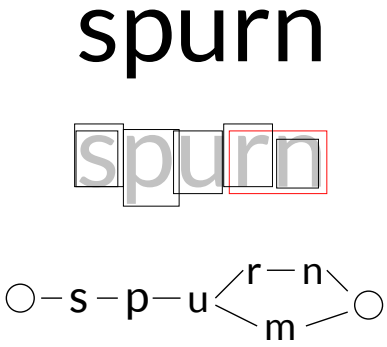

Figure 2. Completed word graph for a simulated example of the word 'spurn'. A false detection of the letter 'm' causes the graph to have multiple paths.

With the graph completed, we can can assign edge weights based on the probability of the letter combination appearing in English text, and node weights using the probability of detection and the probability that the character belongs to the line. By taking negative log-likelihoods of the probabilities, finding the most likely word becomes a minimum-cost graph traversal, which can be solved with Djikstra's algorithm or A*.

Using a complete dictionary of English words would allow more aggressive and accurate guessing of words. However, the corresponding disadvantage is that a large proportion of words in natural images are non-dictionary words, such as proper names.

4. Implementation

We implemented our own HOG descriptor in MATLAB, which efficiently calculates the descriptor at multiple scales by computing the gradients once, building a multidimensional equivalent of an integral image, and then computing the descriptor at each scale. While our code is theoretically more efficient than recomputing the HOG descriptor from scratch at each scale, we found that in practice our MATLAB implementation runs significantly slower than the C implementation of HOG included in the open-source VLFeat library.

4.1. Detectors

The detectors are trained by computing the HOG descriptor for all input images and using logistic regression to obtain a linear classifier. Input images are scaled so that the resulting descriptor is $8 \times n \times 31$, where n varies with the aspect ratio between 3 and 12. This gives feature vectors with between one and three thousand elements.

We initially normalized all of the characters to a constant width and height based on their character bounding boxes. However, this vastly increases the range of scales which the detectors must run at, since 'm' and 'w' are several times wider than 't' or 'f' in many fonts. Additionally, the characters 'i' and 'l', are extremely distorted (stretched to fill the whole width) and thus the training images bear little resemblance to the actual characters which must be detected. Instead, we computed the median width for each character using the training set bounding boxes, and then normalized each character to a single height and the median aspect ratio. This preserves the relative shape, but means that a separate training set must be generated for each character.

We experimented with a variety of training set sizes and relative ratios of positive and negative examples. Further details of the training data are described in Section 5.1.

We used the liblinear package to perform logistic regression in MATLAB. Training on 15,000 images takes approximately an hour on a 2 GHz Intel Core 2 Duo laptop with 3.2 GB of RAM. [1]

To run the detector, we take the dot product between the detector and a flattened block of the image HOG features in a sliding window across the image. By keeping the detector and the image HOG features as 3-dimensional matrices, this becomes a series of cross-correlations between the corresponding planes of the descriptors, which can be computed efficiently as a 3-dimensional convolution in MATLAB.

4.2. Reading

Character position statistics were calculated by taking the mean and variance of ground truth character centers. The vertical position standard deviation is on the order of 2-4 pixels for a text height of 72 pixels, somewhat smaller than we expected. Character coappearance statistics were calculated using 18 popular documents from Project Gutenberg, comprising approximately 12 million characters.

5. Testing

5.1. Synthetic dataset generation

Because obtaining a large dataset with ground truth is difficult and time consuming, we relied on a synthetically generated dataset, which labels ground truth for several stages of the pipeline. This approach was used successfully by Neumann et al. in [6] to train character classifiers for natural images using only computer-generated data. Despite using synthetic data, building a good training set turned out to be the most difficult and time-consuming challenge of the project.

The input to our data generation program is a dictionary of words and a list of fonts installed on the computer. For each requested training sample, the program selects a word and font at random, and generates a black-and-white image containing the word. This image can easily be segmented to

[1] I attempted to run my code on the corn cluster, but the network/disk latency from AFS made it far slower than running locally, since the training includes reading tens of thousands of tiny images.

provide character bounding boxes and other ground truth. A second color image is generated by selecting random colors for the background and text. Noise is added, and the final result is saved. Compression artifacts add a small amount of additional noise.

For character detector training, it is important to include a large number of examples with many small variations. Since the HOG cells form a coarse grid and are run at a discrete set of scales, it is important for the detector to find matches which may have small scale and position differences.

To achieve this, we took each of 1000 input examples and created 15 new images for training. Approximately half of the images contain small position offsets and scaling, while the other half contain large offsets or scales. The former are kept as positive examples, while the latter are marked as negative examples, along with images of all the other letters. This helps prevent the detector from misfiring on parts of letters which might otherwise be considered matches.

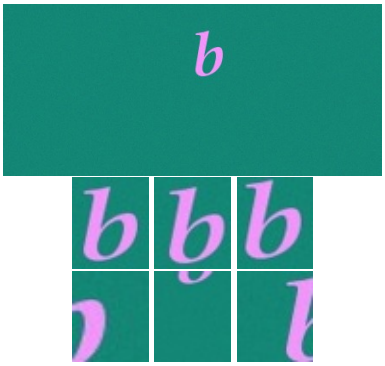

Figure 3. Top: Sample input image. Middle: Positive training examples which exhibit small variations. Bottom: Negative training examples which exhibit large variations.

We found that it was important to provide a small padding space around each training example. If the outside edge of a letter is cut off, the gradient along that edge no longer contributes to the descriptor and the result is much weaker. This is similar to the finding in [5] that space around the pedestrian is important, and that increasing the size of the pedestrian but removing the border space causes a lower accuracy.

After training a set of detectors with this dataset, we created an augmented dataset to reduce false positives. The detectors were run on the original 1000 input images, and training examples were pulled from all false positives.

5.2. Evaluation

To provide a quantitative metric on the accuracy of our detector, we ran it on a separate test set of 100 images. The precision-recall curves are shown in Figure 4, and confusion matrices for the test set are shown in Figure 5.

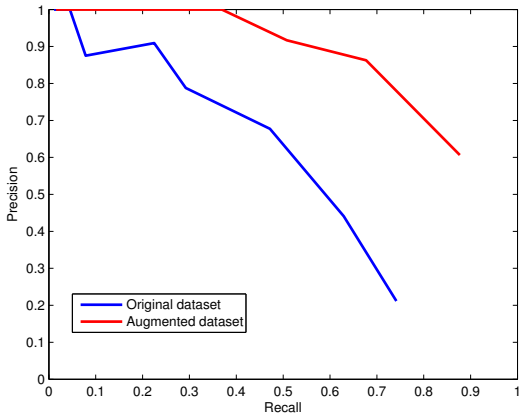

Figure 4. Precision-recall curves for the original and augmented datasets. Most of the images contained multiple detections of at least one character; these were combined into one for calculating precision and recall.

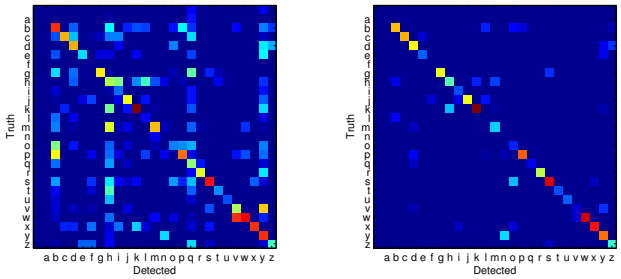

Figure 5. Confusion matrices without (left) and with (right) augmented training set, tested on a separate testing set of 100 images.

While this classifier performs fairly well on the test set which contains single characters, it fares poorly on words and sentences, often misfiring on combinations of letters and spaces between letters. Additionally, it fails badly on the letters ‘i’ and ‘l’, and has some difficulty with ‘f’ and ‘t’, because the training set for these letters contains many negative examples which appear identical to positive examples. Figure 6 shows several negative examples for the letter ‘i’, which could easily be considered positive examples.

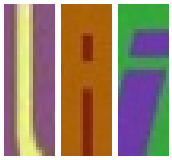

Figure 6. Negative training examples for the letter ‘i’ which are very similar to positive examples. Because of these, the ‘i’ detector does a very poor job.

In order to mitigate these problems, we created a new

training set based on images with complete words. As before, positive examples are taken from the character bounding box with small perturbations. We used a larger border space to allow better differentiation between characters such as 'a', 'd', and 'q', which can look identical if the ascender or descender is ignored. Negative examples are drawn from random patches of the image. We used an input set of 2000 images with an average of 20 letters each, which produced approximately 46,000 training images per letter, split 40% positive/60% negative.

Subjectively, the resulting detector performs better on the problematic letters above. However, it performed poorly on the single-character test set, with precision less than 10% across all recall values. Most of the false positives were on flat patches of noise. Possible solutions for this might be to include a higher percentage of flat examples in the training set, or to replace HOG's local normalization with a wide or global normalization.

The detection algorithm is not accurate enough to provide reliable input to the graph construction and evaluation algorithm, so we did not get to the point of testing it. Running the graph code on ground truth bounding boxes is not particularly enlightening, since it consists of a single true path.

Example results for some images in the ICDAR dataset are shown in Figure 7. On real photographs, our detector frequently reports hundreds of false positives, particularly textured regions of the image such as brick. Using textured backgrounds or image patches as training examples would probably produce better results here than flat colors.

Figure 7. Sample detection results from the ICDAR dataset. The detector finds most of the characters, but includes many false positives. In the bottom image, only the strongest results are shown.

6. Future work

Some letters, particularly the lowercase letter 'a', can be written in multiple ways. It would therefore be more appropriate to split the character into two classes with separate detectors but the same label.

Due to the computational load, we did not have time to experiment with as many parameter permutations as we originally hoped. Further work should test larger descriptors (10 or 12 blocks high) and a range of padding widths.

References

[1] K. Wang and S. Belongie, "Word spotting in the wild," in *Computer Vision – ECCV 2010*, ser. Lecture Notes in Computer Science, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Springer Berlin / Heidelberg, 2010, vol. 6311, pp. 591–604.

[2] A. Coates, B. Carpenter, C. Case, S. Satheesh, B. Suresh, T. Wang, D. Wu, and A. Ng, "Text detection and character recognition in scene images with unsupervised feature learning," in *International Conference on Document Analysis and Recognition (ICDAR)*, Sep. 2011, pp. 440–445.

[3] J. Weinman and E. Learned-Miller, "Improving recognition of novel input with similarity," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, Jun. 2006, pp. 308–315.

[4] L. Neumann and J. Matas, "Real-time scene text localization and recognition," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, june 2012, pp. 3538–3545.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005.*, vol. 1, Jun. 2005, pp. 886 –893 vol. 1.

[6] L. Neumann and J. Matas, "A method for text localization and recognition in real-world images," in *Computer Vision ACCV 2010*, ser. Lecture Notes in Computer Science, R. Kimmel, R. Klette, and A. Sugimoto, Eds. Springer Berlin / Heidelberg, 2011, vol. 6494, pp. 770–783.

A. Project sharing

This project was done in combination with CS 231A (Computer Vision). I was the only student working on the project in either class.