Detecting and Recognizing Text in Natural Images using Convolutional Networks

Aditya Srinivas Timmaraju, Vikesh Khanna
Stanford University
Stanford, CA - 94305
adityast@stanford.edu, vikesh@stanford.edu

Abstract

Detecting and recognizing text in natural images is a challenging problem that has recently received attention. Most methods attempting this have relied on elaborate models incorporating carefully hand crafted features or large amounts of prior knowledge. All methods assume the knowledge of a lexicon of 20-30 words that form the super set of all possible words that can occur in a test image. This makes the problem a lot easier than it would otherwise be. In this report, we present a method of detecting as well as recognizing text contained in natural images using Convolutional Neural Networks (CNNs). We use two CNNs, one trained for the character detection task and another for the character recognition task, as building blocks for the algorithm. Significantly, we do not assume any prior knowledge of a lexicon, unlike all previous works. We demonstrate state-of-the-art experimental results for three different tasks: character detection, character recognition and word recognition, while also showing qualitative results for text recognition. We use the standard Chars74k and ICDAR 2003 datasets for the same.

1. Introduction

Being able to locate, detect and identify the text present in any natural image presents us with many opportunities. It finds application in many fields, ranging from brand detection/recognition in images and videos to systems that aid blind people navigate. A system that is capable of automatically sifting through a set of images (from sources such as Instagram/Pinterest) and identifying presence of a certain brand is of immense monetary value to that brand. It allows for targeted marketing/advertising to a highy selective audience.

However, the task of text detection, localization and recognition is a difficult problem, owing to a lot of variability present in natural images, in terms of illumination, viewpoint, orientation and to a certain extent, even the intra-class variability that is inherent in characters of different fonts/styles and those written by different people. For this reason, a lot of text detection and recognition systems rely on cleverly hand-crafted features [2] [3] to represent images. Some state-of-the-art methods also rely on sophisticated models like conditional random fields, on top of the raw detection/recognition outputs in the end-to-end systems. In our work, given an image, we label the region of interest containing text using a bounding box and find the word predictions. A sample output of our full end-to-end text recognition system is illustrated in Fig 1. All characters occuring in the image are assumed to be in one of 62 possible classes (a-z, A-Z, 0-9). We make the following contributions: 1) We show competitive results without assuming the knowledge of a lexicon, which is too strong an assumption for certain applications 2) We introduce the novel concept of fuzzy character search (FCS), that circumvents errors due to bad character detections and also facilitates doing away with the lexicon. In a clever way, FCS uses a dictionary but does not restrict the choice of word predictions to words from the dictionary, since proper nouns could also occur in images. We demonstrate with examples, how the current state-of-the-art methods fail in the above cases and how FCS helps in those cases.

2. Related Work

Until recently, the end-to-end text recognition problem had only received a modest amount of interest from the vision community. In [6], the authors propose a recognition pipeline comprising multiple feedback loops for hypothesis validation. Briefly, they find Maximally Stable Extremal Regions (MSERs) and then classify these regions as either character or non-character regions. They form possible lines containing text and use character recognition, followed by typographic and language modeling to refine the recognition output. In [7], the authors use Random Ferns for character detection and the HOG descriptor [1] to describe image patches. They evaluate the word detection and recognition performance of a two-step approach comprising a text detector and an OCR engine. They show that their HOG-based system outperforms that using a conventional

Figure 1. Sample output of our end-to-end system. The red bounding box localizes the text containing part of the image. The predicted word is labeled on the box

OCR engine.

In [8], the authors use the output score of a Convolutional Neural Network (CNN) for character detection and another CNN for recognizing characters that have been detected. They use a multi-scale sliding window approach and consider windows in different rows independently. They make the assumption that text in the image occurs horizontally. For each row, the detection scores of different sliding windows are computed and NMS (non-max suppression) is used to suppress spurious peaks. This way, bounding boxes are constructed for rows with non-zero number of peaks. Another NMS is performed to prune bounding boxes with 50% overlap or more. The character recognition CNN is then used on each of these and beam-search is performed to output words. They assume the knowledge of a lexicon of about 20-30 words (which are different for different images, and potentially procured using Geo-location and other meta data), and they only output words from this lexicon. The method used in [7] also assumes the knowledge of such a lexicon.

3. Our Approach

Deviating from some of the above approaches that rely on hand-crafted features, we use CNNs for this problem. We now briefly give an overview of our system. We tackle the problems of detection and recognition in two cascaded stages. Given a novel image instance, we consider a sliding window over the whole span of the image at multiple scales (Figure 10). For each of these windows, we consider the image patch within that window and pass it to the character detection engine. We train the detection CNN in such a way that this character detection engine outputs a high score if there is a centered character in the input image patch. For these firings from the detection engine, we pass the corresponding patches to the recognition engine to determine which character is present in them. We use two Convolutional Neural Networks, one for the detection part and another for the recognition part. Also, we do not make the assumption that a lexicon which contains all words that could occur in a given test image exists.

Figure 2. The first four images are augmented negative examples and the fifth image is the original positive example, with a centered B.

3.1. Detection CNN

The detection CNN takes as input a 32x32 image patch and outputs the probabilities associated with two classes: character present and character absent. We call the probability associated with the character present class as the 'detection score' of that input patch. We use character containing patches from Chars74k dataset [2] as positive examples and images from CIFAR-10 dataset [4] as negative examples, in addition to synthetically generated images as described in the following section on data augmentation.

3.1.1 Data Augmentation

Through our initial experiments, we observed that even patches that contain non-centered characters were giving reasonably high detection scores. However, this would be a problem since it would lead to spurious detection peaks when using a sliding window approach such as ours and the one in [8]. So, we augmented our training data with negative examples that contained characters pushed to the left/right/top/bottom of the input patches, as illustrated in Fig 2. We also augmented our training set with negative examples that contain parts of two characters in them (see Fig 3). The motivation is to prevent the detection CNN from firing in windows that contain the transitions between two adjacent characters in a word.

We now verify that our data augmentation trick indeed works in the way it should. We input to the detection CNN, a set of patches that contain characters on their periphery, and also the same characters aligned in the center. For each patch, we also enumerate their respective detection CNN scores. These results have been illustrated in Fig 4 and Fig 5. In Fig 6, an input word-containing image and its detection CNN scores are depicted. Notice how the character containing regions in the image look red in the heat map (i.e., higher detection CNN scores). Also, notice how the score drops in the transitions between characters in the word, as is desired.

Figure 3. Patches containing transitions between characters in the same word. These are used as additional negative examples for the character detection task.

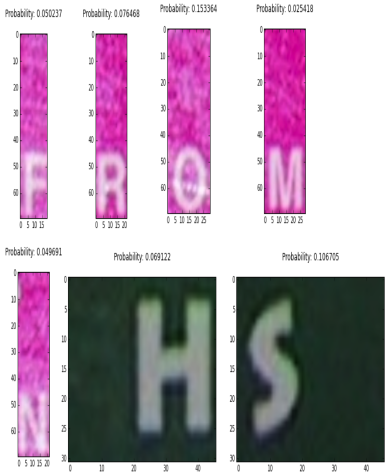

Figure 4. Patches with non-centered characters and their corresponding scores. As is expected, we see that they have very low detection scores, validating our data augmentation trick.

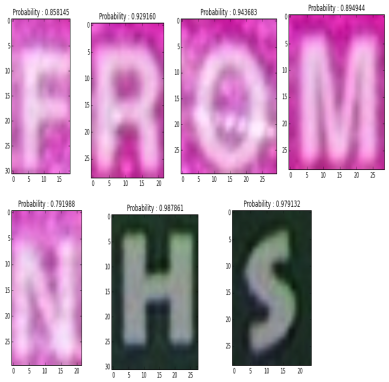

Figure 5. Patches with centered characters and their corresponding scores. In contrast to non-centered patches, we see that these patches have very high detection scores

3.2. Text Bounding Box Detection

We now describe how exactly we propose to use the detection CNN to locate the text bounding boxes in a given image. In the detection CNN, we use two neurons in the output layer, and their outputs correspond to the classes "character present" and "character absent". Given the current sliding window portion of the image, we compute the softmax probability for the "character present" class, as the detection score for this portion of the image. In each row of

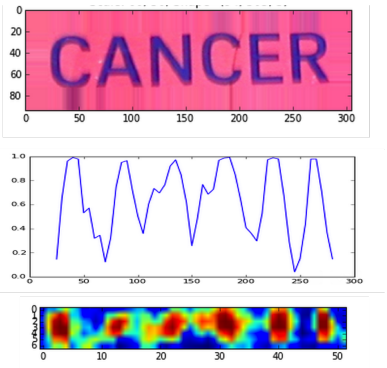

Figure 6. The first image contains the input word. The second image depicts a plot of the detection CNN scores along one row of the sliding window. The third image is a heat map of the detection CNN scores for different positions of the sliding window in the image (red indicates higher values).

the sliding windows, we concatenate this set of scores into a vector. We apply non-max suppression (NMS) on this vector to locate peaks in it. We then ignore all peaks that are less than 50% of the maximum peak. The peaks thus obtained are expected to correspond to those sliding windows that contain centered characters in them. We look for contiguous peaks (that are within a threshold of each other) and that defines our word-containing bounding boxes in the image. Once the set of bounding boxes are computed for the given image, another NMS is performed to prune bounding boxes with 50% overlap or more. If two bounding boxes exceed that level of overlap, the one with the lower average detection score is eliminated and the higher one is retained.

We define the confidence margin of a given image patch (that is known to contain a character) to be the difference between the output class probabilities of the recognition CNN of the top two classes, sorted in the decreasing order of probabilities.

3.3. Fuzzy Character Search (FCS)

Once we arrive at the set of possible word-containing bounding boxes using the method described above, we now set out to recognize the words contained in them. For each text-containing bounding box, we consider the peaks from the detection CNN. After that, around each peak, we consider shifted patches and compute the confidence margins of each of those patches. For each peak, we consider the top 3 character proposals, which correspond to the three patches with the highest confidence margins. If the detection score corresponding to a peak is lower than a threshold, we also add a blank character proposal in addition to the top 3. We label these peaks as *uncertain* peaks. Once we have such proposals for each detection peak, we consider all possible words than can be formed. Among these words, if there is a word that is in the dictionary, we pick it. If none of the

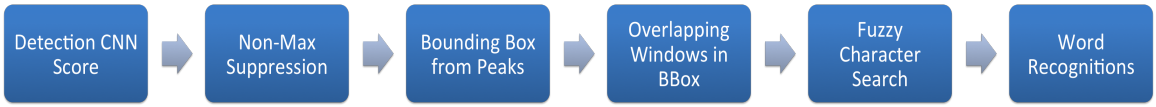

Figure 7. Pipeline of the proposed end-to-end system

Figure 8. Input Image

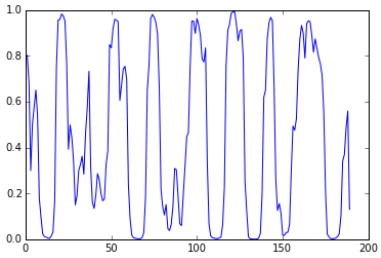

Figure 9. Plot depicting detection CNN scores for a sliding window along the word

words belong to the dictionary, we pick the word with the highest average confidence margin score. This leaves room for words that cannot be found in the dictionary (such as proper nouns) to also be predicted.

Here, we underscore that unlike the dynamic programming based approach that had to be applied to each word in the lexicon in previous methods [8] [7] (that would be computationally intractable at the scale of a full dictionary), we only perform a dictionary look-up, as to whether or not a given word exists. Also, this dictionary does not restrict the choice of words, since FCS can also output words that are not present in the dictionary.

We now show an example word recognition result, with and without using FCS. Fig 8 contains the input image. The plot of detection CNN scores for sliding windows across this image is depicted in Fig 9. Without FCS, the spurious detection peak would have given rise to the prediction "ENTERGENGY". However, since the detection peak corresponding to the letter 'T' is *uncertain*, one of its character proposals is a blank character. This allows the algorithm to neglect it when constructing word proposals. Also, one of the character proposals of the incorrect recognition 'G' is found to be 'C'. This lets FCS accurately predict the word as "EMERGENCY".

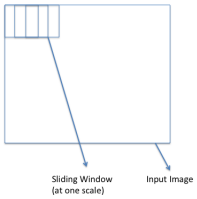

Figure 10. Sliding windows over the input image. We consider these windows at multiple scales.

Layer	Architecture
CONV1	filter:5; stride:1; pad: 2; depth:32
POOL1 (MAX)	filter:3; stride:2
ReLU1	-
CONV2	filter:5; stride:1; pad: 2; depth:32
ReLU2	-
POOL2 (AVE)	filter:3; stride:2
CONV3	filter:5; stride:1; pad: 2; depth:64
ReLU3	-
POOL3 (AVE)	filter:3; stride:2
Affine	output: 64
Affine	loss: softmax

Table 1. CNN 1 Architecture

3.4. Network Architecture

As explained before, we use two CNNs, one for the character detection task and another for the character recognition task. We have used 2 CNN architectures in total, and we label them as CNN-1 and CNN-2. The architectures are enumerated in Tables 1 and 2. We use rectified linear units (ReLU) as our non-linearities and use dropout as regularization. An illustration of the proposed end-to-end system is presented in Fig 7.

4. Datasets and Evaluation Metrics

We evaluate the two CNNs separately on the Chars74K dataset [2]. The English dataset contains 74,000 images of

Layer	Architecture
CONV1	filter:5; stride:1; depth:20
POOL1 (MAX)	filter:2; stride:2
CONV2	filter:5; stride:1; depth:50
POOL2 (MAX)	filter:2; stride:2
Affine	output:500
ReLU	-
Affine	loss: softmax

Table 2. CNN 2 Architecture

Figure 11. Example images from the 74k dataset

characters from natural images, hand-drawn characters, and characters of different computer fonts. The dataset comprises annotated characters from 62 classes (A-Z, a-z and 0-9). An illustration of images sampled from this dataset is presented in Fig 11.

For the character detection task, we trained our CNN using the 74,000 images from the Chars74k dataset as positive examples, and we used 64,000 images from the CIFAR-10 [4] dataset, along with augmented patches containing characters on the periphery, as negative examples. The idea is to get the CNN to learn whether or not there exists a centered character in the current input image it is fed. We used 128,000 images for training and 10,000 images for testing.

For the recognition task, we trained our CNN using the 74,000 images from the Chars74k dataset, and classified each input image into one of the 62 possible classes. We used 70,000 images for training and 4,000 for testing. We also separately evaluated our character recognition CNN on the ICDAR 2003 dataset [5].

For the word recognition task, we used a subset of the ICDAR 2003 word dataset [5], which was released as part of the "robust reading and text locating" competition. We evaluate word recognition performance using two different metrics. The first metric is the accuracy of the predictions, and we consider a prediction to be correct only if it is identical to the word (including the letter-case). We then report accuracy to be the ratio of correct predictions to the total number of predictions. The second metric we use (which we call LCS) is the ratio of the length of the longest common subsequence to the length of the longer word (between the predicted and the actual words). We report the average LCS over all word predictions.

Architecture	Task	Training Acc.	Testing Acc.
CNN-1	Detection	98	92.72
CNN-2	Detection	98.4375	98.53
CNN-2	Recognition	100	86.52

Table 3. Summary of results

5. Experimental Results

We have used both CNN 1 and 2 architectures for the character detection task and CNN 2 for the character recognition task. The batch sizes during Stochastic (mini-batch) Gradient Descent (SGD) were 64 images. The results have been summarized in Table 3. We see that with architecture 2, our CNNs achieve higher accuracies than the state-of-the-art method in [8], for both the character detection and character recognition tasks. For the word recognition task, we have achieved an accuracy of 69.697%. The average LCS has been found to be 0.89.

Figure 12. Plot of training loss for the Character Detection task

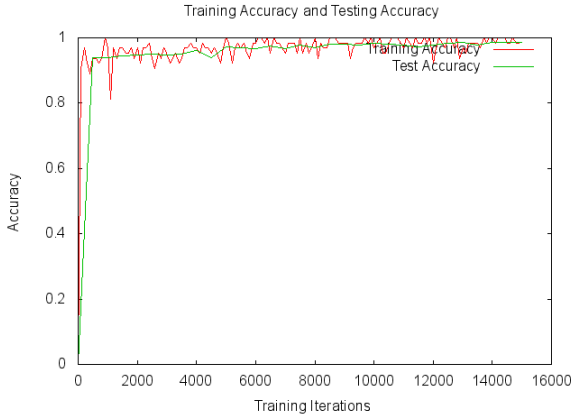

Figure 13. Plot of training & test accuracy for the Character Detection task

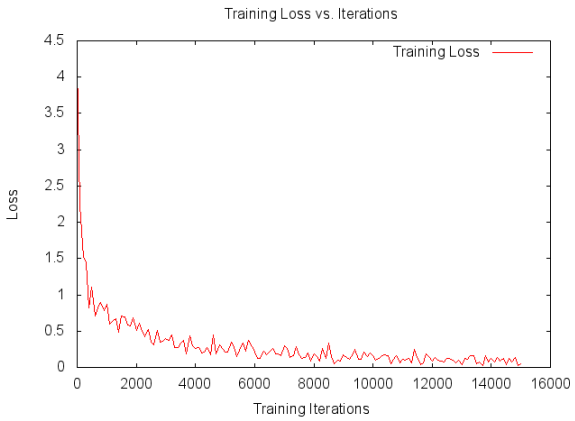

Figure 14. Plot of training loss for the Character Recognition task

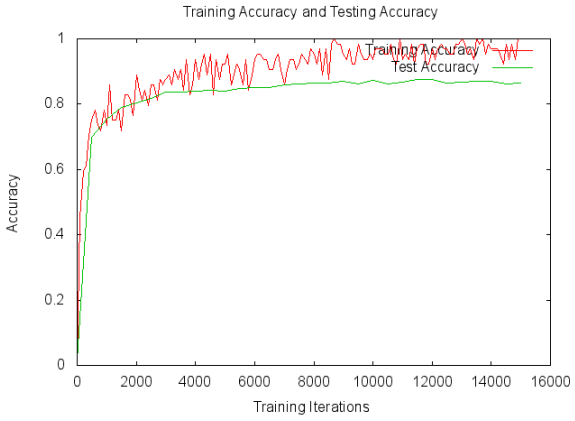

Figure 15. Plot of training & test accuracy for the Character Recognition task

5.1. Analysis

The plots in Fig 12 and 14 depict variation of training loss with the number of iterations, for the detection and recognition task respectively. We can see that the decrement in loss is neither linear nor does it saturate or overshoot, indicating the learning rate is neither too low nor too high during the CNN evaluation. All the plots are for the CNN 2 architecture.

From Table 3, we see that our detection CNN performs remarkably well, and would be thus expected to barely contribute to any impediment to the performance of an end-to-end system. An output of our full-blown end-to-end text recognition system is depicted in Fig 16.

6. Conclusion and Future Work

In this report, we have described a method to perform end-to-end text detection and recognition in natural images. We have demonstrated state-of-the-art results on character detection and recognition tasks. We also introduce Fuzzy

Figure 16. Output of the end-to-end system

Character Search and circumvent the assumption of prior knowledge of a 20-30 word lexicon, which the previous methods make. However, there are more ways in which the performance of text recognition can be improved.

In our experiments, we observed that the system gets confused between 0 and O, 1 and I etc. To help resolve these cases better, it would help to have a CNN dedicated to predicting whether a given patch contains a letter or a digit, and after that predict the character. Secondly, since the letter I occupies lesser width than other letters, the sliding window that contains I will also contain its preceding or succeding letters. So, it would be helpful to train the recognition CNN to recognize bi-grams that contain I, instead of I alone.

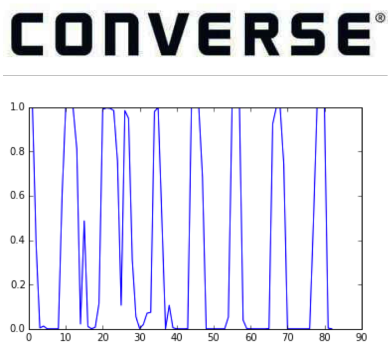

Figure 17. The top image is the input image. The bottom image is the detection CNN scores profile. Notice the spurious repeated peak between positions 20 and 30.

Once we have text-containing bounding boxes, apart from the confidence margins for each window, we also have the detection and recognition scores. We can also compute the variance in relative spacings between consecutive peaks, which will be useful in pruning spurious detection peaks that survive NMS, like the third peak in Fig 17. Each of these metrics are useful features in determining the right word. So, we can train a classifier to learn a complex func-

tion of these features, to predict the final word, given a bounding box. We are confident that a more complex linear function learned through a classifier can improve the fuzzy search algorithm significantly.

References

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In CVPR, 2005.

[2] T. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. International Conference on Computer Vision Theory and Applications, 2009.

[3] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE.

[4] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Computer Science Department, University of Toronto, Technical Report 1.4 (2009): 7.

[5] S. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young. Icdar 2003 robust reading competition. ICDAR, 2003.

[6] L. Neumann and J. Matas. A method for text localization and recognition in real-world images. In ACCV, 2010.

[7] K. Wang, B. Babenko, and S. Belongie. End-to-end scene text recognition. Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011.

[8] T. Wang, D. Wu, A. Coates, and A. Y. Ng. End-to-end text recognition with convolutional neural networks. Pattern Recognition (ICPR), 2012 21st International Conference on. IEEE.