Deep Convolutional Neural Networks for Text Spotting in Natural Images

Ankush Gupta∗ Andrew Zisserman∗ Andrea Vedaldi∗

Department of Engineering
University of Oxford

Figure 1: Example output of the text spotting algorithm explored in this study, over a range of text styles, sizes and types (natural / digital). Bounding boxes show the detected text regions. Text recognition (in order of confidence) for left: {oxford, street}, centre: {university, radcliffe, museum, boolean, camera, pitt, library, theatre, liver, ionian, tourist}, right: {willard, feynman, usa, richard, gibbs, barbara, josiah, yon, usa, ebullition, physicist, john, yunnan, mathematician, thermodynamics, usa, cd, susan, box, geneticist, 31} (faulty or incomplete detections are underlined).

Abstract

In this work we investigate and extend the current state-of-the-art system for text spotting in natural images [Jaderberg et al. 2014a]. First, we extend text recognition to be case-sensitive and include special characters and punctuation marks. Next, we improve text recognition at various word-length scales using separate deep convolutional neural networks for different length intervals. Finally, we introduce the improvements made to the text-spotting algorithm for our entry in the ICDAR 2015 Robust Reading Competition, which was placed at the top.

1 Introduction

Communicating and representing information requires change — a flat or a constant signal contains no information, but just a single level of difference or just two states suffice to communicate and encode every bit (!) of information. Text characters utilise our visual bandwidth — character would not exist if our eyes only had one pixel — to represent information more concisely. In this work we explore and extend the current methods to enable computer recognition of text in natural images. We use convolutional neural networks extensively for recognising word images.

Outline

In Section 2 we briefly review some related work. In Section 3, we summarise the text-spotting pipeline of [Jaderberg et al. 2014a] which we extend and improve in this study. An important text recognition system for unrestricted text strings is reviewed in Section 4. In the following Section 5, we review the synthetic text generation procedure — the powerhouse for our deep and hungry models. In Section 6 we describe our first
contribution of extending text recognition to include symbols and punctuations and case-sensitivity. In the next Section 7 we improve text recognition for different word lengths and in the following Section 8, we describe the changes made to the pipeline for our ICDAR 2015 Robust Reading Competition entry. We finally conclude in Section 9.

2 Related Work

In this study we closely follow and extend the work presented in [Jaderberg et al. 2014b; Jaderberg et al. 2014a]. However, we briefly review below some of the previous work on two traditional stages in an end-to-end text-spotting system — text detection or localisation and text recognition. The decomposition of the end-to-end task into these two stages was first proposed by [Chen and Yuille 2004].

Text Detection / Localisation. Two broad lines of work can be identified for text detection — (1) connected components: in which pixels are segmented into characters and then words are formed by grouping these characters and (2) sliding window: in which the text detection problem is seen as the traditional object detection problem and solved for windows at multiple scales, positions and aspect ratios. Extremal regions [Matas et al. 2004] is a popular technique first used for text detection in [Neumann and Matas 2011] and also successfully used in the ICDAR 2015 Robust Reading Competition’s localisation challenge by the StradVision team.1 Stroke width transform of [Epshtein et al. 2010] finds regions of constant distance between two parallel lines (or a stroke) and groups them into characters. [Jaderberg et al. 2014c] take the sliding window approach and use a text / no-text convolutional neural network (CNN) classifier to generate a saliency map; [Wang et al. 2012] also use a similar approach.

Text Recognition. Text recognition methods assume that a perfect text detector has produced cropped images of words. We closely follow the work in [Jaderberg et al. 2014b] which proposes three CNN models for text recognition — (1) using a fixed lexicon and solving a large classification problem, (2) reading the word character-by-character from the whole image at once and (3) reading the word as a collection of n-grams, again solving a large (but smaller than (1)) classification problem. In [Jaderberg et al. 2015] they attempt to combine the (2) and (3) methods using structured output learning. This approach of using a full word image for text recognition has developed recently due to availability of large datasets and computation resources. Earlier work focused on using character level recognition as a building block for word-level recognition: [Alsharif and Pineau 2013] use a combination of segmentation-correction and character recognition CNNs together with a Hidden Markov Model (HMM) with a fixed lexicon to generate the final recognition result; [Jaderberg et al. 2014c] compute a score for words from a fixed lexicon using dynamic programming based on a binary text / no-text classifier, a character classifier, and a bigram classifier; the PhotoOCR system [Bissacco et al. 2013] uses beam search on scores output by a neural network acting on HOG features of word segments.

3 Overview of the Pipeline

In this section, we briefly review the end-to-end text-spotting pipeline proposed by [Jaderberg et al. 2014a], which we extend and improve upon in this study. A detailed description can be found in the original paper. The task of text-spotting can be broken (roughly) into two sequential stages — (1) text localisation: detecting word regions in the image, and (2) text recognition: recognising the words within these regions. Note that these two stages are not completely sequential as recognition can drive localisation as well (see Section 3.2).

3.1 Text Localisation

Text detection or localisation aims to find the bounding boxes of text regions in the input image; specifically we aim to find bounding boxes at the word level.

Region Proposals

The text region proposals are generated from two complementary detection mechanisms chosen for their speed and high recall rates — the Edge Boxes region proposal algorithm [Zitnick and Dollár 2014] and an Aggregate Channel Feature detector [Dollár et al. 2014]. The Edge Boxes algorithm relies on the intuition that objects can be seen as collection of edge contours, while a Aggregate Channel Features detector is a AdaBoosted random forest trained on pyramids of

1StradVision: http://www.stradvision.com/
Figure 2: A schematic of the CNN architecture used for text-recognition. The filter dimensions are shown at the top. The boxes visually represent the feature-maps. The feature-map dimensions are shown below them — \((n \times m \times f)\) means that there are \(f\) features of size \(n \times m\). The dotted box shows the base-architecture common to both the 90k-lexicon model and the character sequence model.

gradient histograms. The region proposals found by the two methods do not have particularly high recall rates (92% and 72% respectively), but the union of the two sets of bounding boxes found has a recall rate of 98%, indicating that these two methods are complementary [Jaderberg et al. 2014a].

Filtering & Refinement

The region proposal mechanism produces a couple of thousand bounding boxes to achieve high recall. Most of these boxes are false positives (low precision) and have poor overlap with the actual text regions. Hence, to reduce the number of the false positives, so that the expensive text recognition stage is made computationally feasible, a word/ no-word random forest classifier is run on the histogram-of-oriented-gradients (HOG) features of each of these bounding boxes [Breiman 2001; Dalal and Triggs 2005]. All the bounding boxes are re-scaled to a standard size (32 \(\times\) 100 pixels) before extracting the HOG features. To improve the overlap of the proposals with text regions, the coordinates of the each of the remaining bounding boxes are regressed using a deep convolutional neural network (CNN). The CNN takes a whole word image as input and outputs a probability distribution \(P_W\) over words in the lexicon used for training the CNN. Each proposal is assigned a score equal to \(\max_{w \in W} P_W\) and a word label corresponding to \(\arg\max_{w \in W} P_W\). However, this set of proposals still contains duplicates and false positives. To improve the precision and overlap of the bounding boxes with text regions, multiple rounds of greedy non maximum suppression (NMS) and bounding-box regression are performed on the detections with the same word label and detections with different word labels separately. NMS on same word label detections can be seen as positional voting for a word. Finally, the proposals are filtered based on thresholding their scores.

90k-Lexicon Recognition CNN Model

In this section, we describe the CNN model used for text recognition. In this model, the text recognition problem is formulated as a multi-class classification problem on a fixed lexicon of words. Words from the Hunspell spell checking system [Németh 2010] enhanced with different word endings constitute the lexicon. The resulting lexicon is big enough (~90k words) to capture most of the commonly occurring English words. The CNN architecture has five convolutional layers and three fully connected layers. Each layer, except for the last, uses rectified linear unit (ReLU) non-linearity. Softmax over these 90k classes and log-loss are used on the output of the final fully connected layer. Figure 2 presents the architecture in more detail. This
model is trained using stochastic gradient descent on mini-batches with dropout regularisation; gradients are computed using back-propagation.

4 Additional Recognition Models

In addition to the 90k-lexicon model used in the text-spotting pipeline, [Jaderberg et al. 2014b] also explore a character-sequence CNN text recognition model. This model is less precise than the lexicon model, but is useful to capture numeric text, out-of-lexicon words and text with symbols and punctuations (explored in Section 6). This model is detailed below.

Character Sequence Model

Rather than keying into a fixed lexicon of words for recognition, this model performs unrestricted text recognition by using multiple classifiers for each character position. Each character position is classified into one of 37 categories: 26 alphabetical characters (a-z), 10 numerical characters (0-9) and 1 null character. This model has 23 character positions. Hence, output of the final layer is softmax-ed independently over the 37 possible characters for each of the 23 classifiers. Since words have variable lengths (say \( n \)), which are unknown at the test time, the model is trained to predict the characters of the word in the first \( n \) positions; the remaining \((23 - n)\) character positions are set to be the null character. This means that this model has to learn the concept of word-length, and character localisation within a word in addition to which character is there at each location.

5 Synthetic Text Generation

The CNN models described in Section 3.2 have approximately 100 million parameters. Supervised training of such large models requires a lot of labelled training data. Collecting such a large dataset manually would require mining natural images which contain text and then human-labelling and localising the text regions within those images. This can be extremely expensive. It also limits what the model can be trained to learn. To alleviate these problems [Jaderberg et al. 2014b] use a synthetic text image generation engine, which samples from over 1400 fonts, adds text, background and border colour, distorts the text with projective transformations, blends the text image with patches from natural images and finally adds Gaussian noise, blur and introduces JPEG compression artefacts. Their synthetic word image dataset (referred to as MJSynth henceforth), containing 9 million image samples and is freely available for download. Figure 3 shows the stages of this synthetic text generation engine.

We used this synthetic text engine extensively in this study. All 1400 fonts have been chosen from Google Fonts. For rendering and manipulating the text, we use PyGame; the engine itself is written in Python. On the Visual Geometry Group’s ti-tan cluster, image samples are generated at the rate of one sample per second; at this rate, it would take 10 hours to generate a 10 million image dataset using 300 compute nodes. We added an additional stage to the procedure of [Jaderberg et al. 2014b] in which we randomly padded the text image on top and bottom by other word examples before applying projective transformations. This was done to emulate the distribution of natural text images more closely. Please see Figure 4 for examples of this procedure.

6 Extending Text Recognition

In [Jaderberg et al. 2014a] recognition is case-insensitive and is restricted to 36 alpha-numeric characters — 26 a-z characters and 10 numeric 0-9 characters. We extend the character-set of the recognition output to be case-sensitive, i.e. recognise lower and upper case text differently, and include special symbols (e.g. @ # $ % & etc.) and punctuation marks (e.g. ! ; : , . ? / etc.) (henceforth, collectively referred to as ’symbols’). This increases the size of the character-set to 94 — of which 52 are the a-zA-Z characters, 10 are the digits 0-9 and 32 are symbols and punctuation characters found commonly on a standard keyboard. Since symbols and numeric character can be arranged in arbitrary configurations, recognition methods based on classification of words in large lexicons are not well suited.

\(^2\)MJSynth dataset is available at: [http://www.robots.ox.ac.uk/~vgg/data/text](http://www.robots.ox.ac.uk/~vgg/data/text)

\(^3\)PyGame: [http://www.pygame.org](http://www.pygame.org)
Hence, we extend the character-sequence model (Section 4) to have additional case-sensitive alphabets and symbols per character position.

### 6.1 Single Character Recognition

Clearly, including these additional character classes increases the variability in the scale and shape of the characters to be recognised. To investigate the feasibility and difficulty of recognising these characters, we first trained a deep network to classify an image of a single character into one of 94 character categories.

![Image of training images](image)

**Figure 4:** Samples of training images used for single character case-sensitive and symbol recognition.

#### Network Architecture & Dataset

**CNN Architecture.** The same base architecture as in section 4 was used, i.e. the same number of convolutional and fully-connected layers; the last layer was replaced to output score for 94 categories, which was then passed through softmax. The network was trained through MatConvNet [Vedaldi and Lenc 2014] using stochastic gradient descent on mini-batches and dropout regularisation.

**Dataset.** Approximately 10 million 32×32 pixel sized images of single characters were generated using the synthetic text generation engine. These images were generated by padding the character of interest on the left and right by two other random characters, which could also be blanks (or whitespace), and then cropping out the central character with some context on the sides. Figure 4 shows a few examples of these training images.

#### Experiments & Discussion

Table 1 summarises the performance of the network in recognising single character images taken from a 10% test-set split of the synthetic data. The case-insensitive error was obtained by comparing the ground-truth label with the prediction without considering the case (e.g. for an image of ‘w’, both ‘w’ and ‘W’ were accepted). We note that case-insensitive accuracy is significantly higher than case-sensitive accuracy (a gap of 3%). This is mainly because characters like ‘v’, ‘u’, ‘w’, ‘s’, etc. have similar looking upper and lower case forms. Further, confining the output categories for alpha-numeric categories seems to matter less (an improvement of ~0.5%) than it does for symbols (an improvement of ~1%), meaning that symbols are more likely to be mis-classified as alphanumeric characters than the other way round.

<table>
<thead>
<tr>
<th>Test Set</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All Categories</td>
</tr>
<tr>
<td>all</td>
<td>92.9</td>
</tr>
<tr>
<td>alpha-num case-insensitive</td>
<td>91.8</td>
</tr>
<tr>
<td>alpha-num case-sensitive</td>
<td>94.9</td>
</tr>
<tr>
<td>symbols only</td>
<td>95.2</td>
</tr>
</tbody>
</table>

**Table 1:** Recognition accuracy for single character images. For the column titled ‘All Categories’, the recognition was made by taking the maximum scoring character from all the 94 character categories, whereas for the column titled ‘Confined’, the maximum score was taken over either alpha-numeric characters only (26×2+10 = 62 categories; for rows 2 and 3) or over the symbols only (32 categories; for row 4).

### 6.2 Full Word Case-Sensitive and Symbol Recognition

The next step is to make full word predictions. Character level recognition has been used as a building block for word-level recognition using viterbi / maximum likelihood computations for character sequences from a fixed lexicon. However, as demonstrated in [Jaderberg et al. 2014b] a simpler CNN model trained to output word annotations, given as input complete word images, simplifies the procedure while still achieving state-of-the-art results; we follow this approach.

#### Network Architecture, Dataset & Training

**CNN Architecture.** We train a character-sequence model similar to the one in Section 4 but with additional outputs for each character position in the final layer. The network has 7 character positions with 94
possible characters (+1 null or blank character) at each location.\footnote{We train for 7 character positions instead of the original 23 positions to keep the scale of the problem small. We can easily generate more training data and train for longer words.}

Figure 5: Samples of training images used for case-sensitive and symbol recognition of whole word images.

Dataset. Again, to train this network we generate synthetic data. However, unlike [Jaderberg et al. 2014a], we do not restrict the word samples to the Hunspell lexicon because it does not contain any case-sensitive information or symbolic characters. To model the natural distribution of symbols, punctuations and case-sensitivity, we draw text strings from NewsGroup20 data-set [Lang and Mitchell 1999] and combine it with samples from Hunspell and also introduce a small fraction of the word strings chosen completely at random to model any unseen text and numerical text strings. We generate 10 million word images. Figure 5 shows some samples from this synthetic dataset.

Figure 6: Filters learnt in the first convolutional layer by character sequence model of Section 3.2 trained using dropout (left) and the CharSymb model of Section 6.2 trained using batch-normalisation and no dropout (right). Both the filter-banks learn Gabor filters and localised peak detectors (characterised by a bright circular spot).

Training. The network is trained using stochastic gradient descent on mini-batches. The training set was balanced across word-lengths such that every mini-batch had the same number of training images for each word-length. Unlike Section 3.2 and 6.1, we do not use dropout regularisation. Instead we use the recently introduced batch normalisation technique, normalising the output of every convolutional and fully-connected layer (except for the last) to have zero mean and unit variance [Ioffe and Szegedy 2015]. \footnote{The variance is not exactly one, because of the \( \epsilon \) used for numerical stability.} Training using batch-normalisation was found to be an order of magnitude faster. There is hardly any visual difference in the filters (in at least the first convolutional layer) of the model trained using dropout and the model trained using batch-normalisation (and no dropout); these filters are visualised in Figure 6.

Experiments & Discussion

We evaluate this whole word image case-sensitive and symbol recognition network (henceforth called CharSymb model), on the training-set of the ICDAR 2015 Robust Reading Competition’s word recognition sub-task [Karatzas et al. 2015]. This competition has two data-sets — (1) Born-Digital: text images from electronically generated images and (2) Focused Scene Text: images containing text found in the natural environment. Figure 7 shows sample images from these two datasets. The Born Digital images are typically of lower resolution and have more symbols/numerical characters.

Table 2 summarises the accuracy of the CharSymb model and compares it with the character-sequence model of [Jaderberg et al. 2014a] (henceforth called MJCharnet model). As in the single character image experiments, there is a huge performance gap in case-sensitive and case-insensitive recognition (expe-
<table>
<thead>
<tr>
<th>Dataset</th>
<th>CharSymb Model</th>
<th>MJCharnet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
<td>Symbol</td>
</tr>
<tr>
<td>Born Digital</td>
<td>56.4</td>
<td>70.6</td>
</tr>
<tr>
<td>Focused Scene</td>
<td>55.2</td>
<td>78.9</td>
</tr>
</tbody>
</table>

Table 2: Accuracy of the CharSymb model (Section 6.2) and the MJCharnet [Jaderberg et al. 2014a] (and Section 4) model on the ICDAR 2015 Robust Reading Competition’s training images. ‘All Categories’ and ‘Confined’ are as in Table 1. The alternately dark and light coloured columns correspond to case-sensitive and case-insensitive recognition experiments respectively. In ‘All’ experiments, all the images were tested upon; in ‘Symbol’, only the images of text string with at least one symbol were used; in ‘Al-Num’ only the images of text-string of alpha-numeric characters were used (i.e. images with at least one symbol were excluded).

![Figure 8](image-url)

Figure 8: Performance of the CharSymb and MJCharnet model across word lengths. The black curve is the performance of the CharSymb model on the 10% split of the synthetic data on which it was trained. All other curves are for images from the ICDAR datasets of only alpha-numeric characters (no symbols) because the MJCharnet model is not trained for symbols. Data-points at lengths 1 and 2 for the Born Digital have been omitted because there were only very small number of images at these lengths (0 and 6 in number respectively).

Figure 9 shows the performance of [Jaderberg et al. 2014b] character-sequence model (called MJCharnet model henceforth) in recognising images of different word-lengths. We note that the model performs poorly for short (≤ 4) and long words (≥ 12). This lent us an opportunity to improve. We investigated the effect of (1) input image aspect ratio, (2) model capacity (3)
and distribution of word-lengths in the training set on the recognition accuracy of a CNN model across word-lengths.

**Input Image Size.** We trained three different models one each for input images of size \{32\times 64, 32\times 96, 32\times 128\}. However, not much improvement was found in the accuracy of the trained models. Note, this could have been a potential problem with how the MJCharnet model was trained because the filter sizes are fixed at either 5\times 5 or 3\times 3; the recognition could have suffered if the input image was too big or too small for the filters. However, no such effect was found.

**Training Set Distribution & Model Capacity.** The training set used for MJCharnet model was uneven across word-lengths, with most of the examples concentrated between lengths 7 and 12. We note that the MJCharnet model indeed performs better in this word-length range as compared to other lengths. We equalised the training set and re-trained a model. The ‘Single 1-15’ data-series in Figure 9 shows the accuracy across word-lengths for this model. We note that this model indeed learnt to perform better for short and long words as compared to MJCharnet model but at the cost of doing worse for mid-length words.

Clearly, this is a case of model under-capacity. Hence, we trained three different models, specialising in words length intervals; we trained a model each for the intervals \{1-to-7, 6-to-10, 9-to-15\}. Performance for these models is also shown in Figure 9. We note that the models for length intervals 1-to-7 and 6-to-10 clearly outperform (within an acceptable margin of error) the MJCharnet model. For the interval 9-to-15, we do worse for lengths 9 and 10, (and come very close for 11), and outperform again for lengths 12 to 15. Note, that lengths 9 and 10 are covered by 6-to-10 model also, so this is not a concern. One explanation for why the new models perform slightly worse than MJCharnet model for some lengths is because these models were trained only on images in their respective word-length intervals, whereas the MJCharnet model was trained on images of words of length 1 to 23.\(^6\)

\(^6\)We only looked at a maximum word-length of 15 because there are very few words of length more than 15 in the Hunspell lexicon; all these experiments can be easily extended for longer word-lengths, for example by training a fourth model for the 14-to-23 length interval.

![Figure 9: Performance distribution across word lengths.](image)

**Length Classification**

To deploy these three length-specialised models in a text recognition system, we need to identify which length interval a given word image lies in. For this, we trained a 15-way word-length classifier, which had the same base-architecture as the MJCharnet model (and as described in Section 3.2). The last layer had 15 outputs, one for each length. The accuracy of this classifier is also shown in ‘length’ curve in Figure 9. We note that the accuracy of this classifier is very good for short words but gets worse for longer words. However, for recognition we only need length-interval level classification. To get length-interval level classification accuracy, a word image of length \(n\) was considered correctly classified if the predicted-length \(m\) lied in the correct length-interval corresponding to \(n\). If \(n\) was a length in the overlap regions, i.e. \(n\) was either 6, 7, 9 or 10, the classification was considered successful if \(m\) lied in at least one of the two correct length-intervals. At the length-interval level, all word-lengths have near perfect classification; this is shown in the ‘interval’ curve. The ‘classify+recog’ curve shows the accuracy of the combined word-length classification and recognition system.
Datasets for Testing & Training

All the models were evaluated on the test set of the MJSynth dataset (Section 5). The MJ model was made available by the original authors and is publicly available; this model was trained using the training set of MJSynth dataset. The new models were trained on word-length-equalised mini-batches. For this the training set of the MJSynth dataset was augmented with new synthetically generated training images for short and long words as the MJSynth training data was concentrated in the mid-length range. Then at the training time, each mini-batch was chosen to have the same number of samples for each word-length. The words for these new training images were sampled form the Hunspell lexicon as was done for generating the MJSynth dataset.

The models were trained through SGD on mini-batches using batch-normalisation on the greyostrich GPU server hosted by the Statistics and AIMS CDTs.

8 ICDAR Challenge

We entered our text-spotting and localisation pipeline in the ICDAR 2015 Robust Reading Competition’s Focused Scene Text Challenge [Karatzas et al. 2015]. Our method was the top performing entry for the end-to-end text spotting challenge, and was the second best entry for the localisation challenge.

8.1 Localisation Challenge

The task here was to localise, i.e. report bounding boxes for text regions in the given images. A localisation was considered successful at 0.5 intersection-over-union (IoU) overlap with the ground-truth bounding box. We describe here few improvements over the standard pipeline of [Jaderberg et al. 2014a] made for this challenge:

- The text-spotting pipeline of [Jaderberg et al. 2014a] only relies on the recognition from the 90k-lexicon model. We combined the outputs from both the 90k-lexicon model and the character-sequence model. Using two recognition models gives two independent scores of confidence for each region proposal. A threshold over the two scores was defined which was found to be more robust than thresholding over just the 90k-lexicon model’s score.
- The edit-distance between the 90k recognition and the character-sequence recognition was also a factor in rejecting/accepting a region proposal.
- In general, the recognitions of the character-sequence model are worse than those of the 90k-lexicon model. However, the character-sequence model gives a better estimate of the word-length. Hence, ratio of the word-length of the character-sequence model prediction to the aspect ratio of the proposed bounding-box was used in filtering the region proposals.

8.2 End-to-End Text Spotting Challenge

In this challenge, the goal was to localise and recognise text regions in the given images. A text-region was successfully ‘spotted’ if the detected bounding box’s overlap with the ground truth had at least a 0.5 IoU overlap and the text recognition was 100% correct (note, no fuzzy/ edit-distance based metric was used). To aid recognition, a lexicon of 50 words per test image con-
sisting all the text instances occurring in the image and some distractors was also made available. Apart from the improvements in localisation mentioned in the last section, we also made some improvements to recognition stage for this challenge. We again used both the 90k-lexicon and the character sequence CNN models for recognition. The final recognition was made through the consensus of the annotations and recognition scores obtained using these two models. The consensus depended on five factors: (1) the two CNN recognition scores, (2) the edit-distance between the two CNN annotations, and (3) the respective minimum edit-distances of the two annotations to the per-image lexicon. Using these two different CNNs for recognition was particularly useful because of their complementary nature – fixed lexicon encoding is more robust than character sequence encoding while the latter can capture out-of-lexicon and numeric text.

Figure 10 shows the precision-recall (PR) curve for the two challenges and compares them with the PR curve of the [Jaderberg et al. 2014a] pipeline. These PR curves were obtained by sweeping the threshold on the recognition scores used for filtering the region proposals in the final step. The highest f-score of our improved pipeline is 9% higher than the f-score of the [Jaderberg et al. 2014a] pipeline.

9 Conclusions & Extensions

In this study we extended the recognition stage to be case-sensitive and include special characters and punctuation marks. Second, we improved text recognition at various word-length scales using separate deep convolutional neural networks for different length intervals. And lastly introduced improvements employed for text recognition in the ICDAR 2015 Robust Reading Competition.

Text spotting in natural images while not fully solved, works well for images in which text is in focussed and centred within the frame. However in images with incidental text, i.e. images not taken explicitly to capture the text, the problem is much more challenging. Further, in such images text size is usually small, of low resolution and appears at steep angles due to projective distortions. These images present a huge challenge. Further, lexicon constrained text recognition performs is found to perform better than unconstrained text recognition. Most text recognition system focus on small windows around the text region and disregard the context in which the text is found. Text distribution can be further constrained by pooling in context from the environment; surely this needs to be explored further.

Lastly, the text spotting pipeline can be sped up by feature sharing — currently, many different stages compute features independently. One possible approach could be to learn a deep network end-to-end, i.e. one which combines localisation with recognition — image-in, annotation out system. It could finally be possible today to build such a system using the synthetic data engine and deeper networks.

10 Acknowledgements

We are thankful to the EPSRC for the AIMS CDT programme. Ankush is supported by the Clarendon Scholarship and the Balliol College Eddie Dinshaw Scholarship. We express gratitude towards Max Jaderberg for getting us started with his code and towards Karel Lenc for helping out with MatConvNet questions and maintaining the shallow GPU cluster and also for introducing us to the mysterious working of the titan cluster with whom thousands of pleasant compute hours were spent. We are also thankful to the quietude and freedom made available at the Balliol College graduate accommodation for enabling us to work on this project.

References


