PhotoOCR: Reading Text in Uncontrolled Conditions

Alessandro Bissacco[*], Mark Cummins[*], Yuval Netzer[*], Hartmut Neven
Google Inc.

Abstract

We describe PhotoOCR, a system for text extraction from images. Our particular focus is reliable text extraction from smartphone imagery, with the goal of text recognition as a user input modality similar to speech recognition. Commercially available OCR performs poorly on this task. Recent progress in machine learning has substantially improved isolated character classification; we build on this progress by demonstrating a complete OCR system using these techniques. We also incorporate modern datacenter-scale distributed language modelling. Our approach is capable of recognizing text in a variety of challenging imaging conditions where traditional OCR systems fail, notably in the presence of substantial blur, low resolution, low contrast, high image noise and other distortions. It also operates with low latency; mean processing time is 600 ms per image. We evaluate our system on public benchmark datasets for text extraction and outperform all previously reported results, more than halving the error rate on multiple benchmarks. The system is currently in use in many applications at Google, and is available as a user input modality in Google Translate for Android.

1. Introduction

Extraction of text from uncontrolled images is a challenging problem with many practical applications. Reliable text recognition would provide a useful input modality for smartphones, particularly in applications such as translation where the text may be difficult for a user to input by other means. Text extraction is also useful in robotics, as a search signal in large image collections, in wearable devices and numerous other areas.

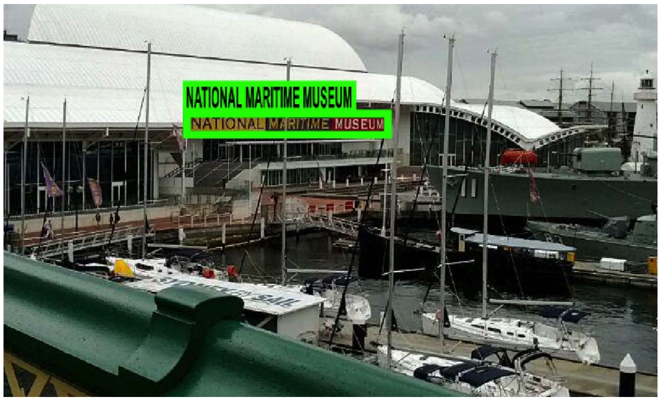


Figure 1: An example of scene text detected and recognized by our system.

Commercially available OCR systems are designed primarily for document images such as those from a flatbed scanner, and perform poorly on general imagery. They typically rely on brittle techniques such as binarization, where the first stage of processing is a simple thresholding operation used to divide an image into text and non-text pixels [19]. Challenging input for existing commercial systems include both scene text (such as Figure 1) and also more document-like text that suffers from blur, low resolution or other degradations that are common in smartphone imagery (see Figure 2).

To address these problems, this paper describes the design of a complete OCR system built using modern computer vision techniques. In particular we take advantage of substantial recent progress in machine learning [6, 10] and large scale language modelling [1]. Our system outperforms previous approaches by a wide margin, more than halving the error rate on the main public benchmarks. We scale the individual components of our system to a regime orders of magnitude larger than explored in prior work. In particular, our deep neural network character classifier is trained on up to 2 million manually labelled examples, and our language model is learned on a corpus of more than a trillion tokens. We maintain sub-second recognition latency primarily though careful engineering. We have trained versions of our system for 29 languages with Latin script, plus Greek, Hebrew and four Cyrillic languages.

2. Related Work

Design of a complete OCR system for natural images is a substantial task, and as such there are relatively few examples in the literature. Many publications address sub-tasks

[*] These authors contributed equally.

such as text detection and isolated character classification. One of the best performing text detection methods is the stroke width transform of [8]. Isolated character classification is widely used as a machine learning benchmark [17]. However the mid-level problem of fusing character classifier and language model signals for complete text extraction is less commonly addressed. Application papers often perform text detection and preprocessing before applying a commercial OCR system designed for printed documents, as for example in [4].

Among fully complete systems for the scene text extraction problem, language modelling is often less developed than the image processing components. For example the method of [18] uses a bigram language model together with a set of hand-designed image features and a support vector machine classifier for text detection and recognition. In the method of [27], text detection is assumed and recognition is performed by fusing appearance, self-similarity, lexicon and bigram language signals in a sparse belief propagation framework. The work of [23] focuses on the use of appearance similarity constraints to improve performance. Several other systems simplify the problem by assuming that the words to be recognized come from a fixed lexicon. The system of [20] describes a large-lexicon design, using weighted finite state transducers to perform joint inference over appearance, self-consistency and language signals. Other systems assume a small lexicon (transforming the problem from OCR to word spotting). One such method is [25], where detection and character classification are performed in a single step using randomized ferns. The method of [15] also relies on small lexicons to allow for a strong bigram constraint in a conditional random field (CRF) model.

Finally, we note that the classic work on handwriting recognition systems [14] tackles essentially the same problem as discussed in this paper, and our system bears many similarities to that design. Work on low-resolution document OCR [11] is also closely related.

3. System Design

In general outline our system takes a conventional multi-stage approach to text extraction, similar to designs such as [2]. We begin by performing text detection on the input image. The detector returns candidate regions containing individual lines of text. Detection is tuned for high recall, relying on later stages of processing to reject false positives. Candidate text lines from the detection stage are processed for text recognition. Recognition begins with a 1D over-segmentation of the text line to identify candidate character regions. We then search through the space of segmentations to maximize a score which combines the character classifier and language model likelihoods. The top hypotheses produced by this search are then re-scored using additional signals which are too expensive to apply during initial inference. The reason for this staged approach is computational: it would be prohibitively expensive to apply full inference at all locations and scales in the input image, or to apply our character classifier at all locations in each candidate text region. Our primary intended application is text extraction as an input modality for smartphone users, which limits total acceptable processing time to at most one or two seconds per image. This constraint informs several of our design decision. The following sections describe each stage of the process in detail.

3.1. Text Detection

A detailed description of the text detection portion of our system is outside the scope of this paper. Briefly, we combine the output of three different text detection approaches. The first approach is a Viola-Jones style boosted cascade of Haar wavelets [24]. The second approach extracts connected components from a binarized image and uses graph cuts to find a text/non-text labelling by minimizing an MRF with learned energy terms, broadly similar to [21]. The final detector is a novel approach based on anisotropic Gaussian filters. This portion of the system also deals with splitting text regions into individual lines and correcting orientation to horizontal, both of which are relatively trivial. For the remainder of the paper we will focus on extracting text from the horizontal line region candidates.

3.2. Over-Segmentation

The over-segmentation step divides the text line into segments which should contain no more than one character (but characters may be split into multiple segments). This is a 1D segmentation task. We combine the output of two different segmentation methods to improve recall.

The first segmentation method used is typical of document OCR systems. The input image is binarized using Niblack binarization [19], a morphological opening operation is applied, and connected components are extracted from the resulting binary image. This simple approach is very effective on easier images, but can fail in the presence of blur, low resolution, complex backgrounds, etc.

The second segmentation approach is intended to handle these more complex cases. It consists of a binary sliding window classifier, trained to detect segmentation points between characters. We use a combination of HOG features [5] and Weighted Direction Code Histogram features (WDCH) [13]. WDCH features are similar to HOG computed on a binarized version of the image, and are typical in OCR systems. A binary logistic classifier is trained on the combined feature vector. The classifier is evaluated in a window of width equal to the line height with a stride of 0.1 times line height, and all responses above a threshold are accepted as segmentation points. We evaluated multiple classifier and feature options for this segmenter and chose this configuration as balancing speed and recall.

The segmentation stage outputs a vector $\boldsymbol{B}$ containing the positions of the detected segmentation points, including the start and end points of the text detection box.

3.3. Beam Search

We now search the space of segmentations to find one which maximizes our score function, given by:

$$S(\boldsymbol{b}, \boldsymbol{c}) = \frac{1}{N} \sum_{i=1:N} \log \Psi\left(c_i, b_i, b_{i+1}\right) + \alpha \log \Phi\left(c_i, c_{i-1}, \ldots, c_1\right) \tag{1}$$

Here $\boldsymbol{b} \subset \boldsymbol{B}$ is a vector of $N+1$ segmentation points b_i which define a segmentation of the line into N segments. $\boldsymbol{c}$ is vector of class assignments, the i^{th} segment being assigned label c_i. $\Psi\left(c_i, b_i, b_{i+1}\right)$ is the classifier probability for class assignment c_i to the pixels between b_i and b_{i+1}. $\Phi\left(c_i, c_{i-1}, \ldots, c_1\right)$ is the language model probability for the i^{th} class assignment given the previous class assignments in the line. α controls the relative strength of the character classifier and language model. The total score is thus the average per-character log-likelihood of the text line under the classifier and language model. We choose the per-character average as it gives a score which is comparable across lines with different numbers of recognized characters. We now wish to find $\boldsymbol{b}, \boldsymbol{c}$ that maximize S.

We perform this maximization using beam search [22], which is the typical approach for similar problems in speech recognition. The result space naturally forms a graph defined by the segmentation points. Beam search is a best-first search through this graph which relies on the fact that each node in the graph is a partial result (corresponding to the recognition of part of the text line) which can be scored by our scoring function. At each step of the beam search, all successors of the current search nodes are scored, but only a fixed number of top scoring nodes (the *beam width*) are retained for the next search step. We initialize the search simply at the left edge of the text detection box. We prune the search slightly by dropping segmentation candidates whose aspect ratio is too large.

Clearly there is no guarantee of locating an optimal solution using beam search. The reason for using this approach in preference to a framework such as a CRF is that it permits greater flexibility in the design of the score function. In particular, the language model imposes high-order constraints (up to order eight in our case) which make exact inference in a CRF-type framework effectively intractable. In our experience, it is better to use a good score function with approximate inference than a weaker score function with perfect inference. Results are presented in Section 5 which suggest that in any case our approximate inference often locates the optimal solution, despite the combinatorial search space. The practical bottleneck on recognition performance appears to come from classifier and language model quality, rather than failure to find the solution which maximizes the score function.

3.4. Character Classifier

We use a deep neural network for character classification. We have explored networks trained on both raw pixels and HOG features. The networks trained on raw pixels achieve similar or slightly better performance than those trained on HOGs. However, we find that the raw pixel networks are deeper and wider than HOG-input networks at comparable performance. Even accounting for HOG computation time, the HOG-input networks have lower computational cost. Since we are designing for speed as well as high accuracy, we find that overall the HOG-input networks are preferable. This finding may be specific to text, since the strong gradients present in characters are a good match for HOG features.

Our best performing configuration is a network with five hidden layers in configuration 422-960-480-480-480-480-100. The layers are fully connected and use rectified linear units [16]. The 422-parameter input layer consists of HOG coefficients and three geometry features. The output layer is a softmax over 99 character classes plus a noise class. We quantize weights and activations to 8 bits, which improves speed with negligible accuracy penalty. Details of training are discussed in Section 4.

We run this classifier on all combinations of segments chosen by the beam search. Since the segmentation provided is only 1D, for each segment we refine the top and bottom boundaries by a simple heuristic which snaps to the first strong edge. This provides a more tightly cropped character to the classifier. We compute two HOG features on this character patch. The first uses a 5x5 grid with 5 bins per histogram, computed directly on the (non-square) patch. The second uses unsigned gradient histograms in a 7x7 grid with 6 bins per histogram. The character patch is normalized to 65x65 pixels for computing this second feature. The three geometry features used in addition to HOG encode the original aspect ratio and the position of the top and bottom edge of the pixels relative to the height of the overall text detection.

It is worth mentioning that we also explored convolutional neural networks, but they are not computationally competitive with non-convolutional networks in our architecture. Our segmentation based approach offers fewer opportunities for the convolutional network to reuse computations compared to a sliding window design.

3.5. Language Model

In structured classification tasks such as OCR and speech recognition, a strong language prior makes a major contribution to final performance. Some classes are almost indistinguishable as isolated examples, such as the number "1" and the letter "l"; these cases must be disambiguated by context.

We use a standard ngram approach for language modelling. Because our system is designed for use in datacenter environments, we adopt a two-level language model design. A compact character-level ngram model is held in RAM by each OCR worker. This model provides the beam search language score, $\Phi(c_i, c_{i-1}, \ldots, c_1)$. This score forms part of the inner loop of our system, so rapid evaluation is essential. Our second level language model is a much larger distributed word-ngram model using the design of [1]. This model is shared by all OCR workers in a data center and is accessed over the network. Due to the latency overhead this imposes, we evaluate this model only during reranking (Section 3.6).

For our character-level ngram model, we are quite limited in RAM budget. A copy of this model is held by each OCR worker. A single worker is designed to recognize multiple languages simultaneously; at present we support 29 languages with Latin script. Consequently we allocate only 60 MB of RAM per language for the character ngram model. In this regime, in common with [3], we have found little benefit in going beyond 8-gram models. We train each of these model on 10^8 characters of training data, retaining all ngrams which occur 40 times or more. For a fixed size model, we find negligible benefit in increasing the training data beyond 10^8 characters. We have also compared smoothing methods and we find that the very simple Stupid Backoff method [1] performs as well as more complex methods in terms of final recognition performance. Since it permits an optimized implementation, we choose this approach.

In addition to the character ngram model, we also maintain a simple dictionary of the top 100k words per language. We use this as an additional signal in our language score; it provides a small performance increase over the character ngram model alone. The dictionary provides a soft scoring signal only; recognition is not limited to dictionary words.

Our second-level distributed language model uses word 4-grams. The English model is trained on a 1.3×10^{12} token training set. The final model contains 2×10^{10} ngrams over a 1M word vocabulary. Our models of non-English languages are approximately 3x smaller in ngram count and 10x smaller in training set size. At serving time the combined multi-language model is distributed over 80 machines and occupies approximately 400 GB of RAM. In the word-level language model we also incorporate a small number of parsers for common patterns such as emails and URLs, which are not well captured by ngrams.

3.6. Reranking

The beam search terminates with ranked list of recognition hypotheses, of *beam width* size. We perform several post-processing and reranking operations to this list:

Punctuation Search

The first operation we perform is a punctuation search. Punctuation is recognized in the same way as any other character class in the initial beam search, but recall is comparatively low since it can be difficult to distinguish small punctuation characters from background clutter. We thus perform a second pass search for punctuation taking advantage of the stronger scale and location constrains available after initial recognition.

Secondary Language Scoring

As discussed in Section 3.5, we use a distributed word-level language model which cannot be accessed during the beam search for latency reasons. We instead use this model to re-score the final recognition hypotheses. We re-score in a straightforward way by adding log-likelihoods:

$$S^*(\boldsymbol{b}, \boldsymbol{c}) = S(\boldsymbol{b}, \boldsymbol{c}) + \beta L(\boldsymbol{c}) \qquad (2)$$

where $L(\boldsymbol{c})$ is the score from the word-ngram model, and β is a weighting parameter. As in Equation 1, we use mean per-character log-likelihood for the language model score, in order to make it comparable between lines of different lengths. Out-of-vocabulary words are assigned a fixed per-character log-likelihood.

Shape Model

We also re-score the hypotheses using shape information. The shape model computes the expected relative size and position of character bounding boxes for the recognized text in 20 common fonts, and scores the line based on the deviation from the best matching font. The shape model gives a small but consistent improvement to overall performance. In principle this could be incorporated into the beam search score, but for computational reasons we use it as a reranker.

Junk Filter

Finally we apply a junk filter with a few simple heuristics to discard common OCR errors such as recognizing repeating vertical structures as "iiiii". While these are naturally penalized by the language model, a simple filter provides some additional precision improvement.

4. Training

We train our neural network character classifier using stochastic gradient descent with Adagrad [7] and dropout [10], using the distributed training design described in [6]. We minimize log-loss. We achieve best performance with dropout of 5% in the input layer and 12.5% in the hidden layers. This dropout setting is lower than has been reported elsewhere, but seems consistent with the relatively small size of our network and the large quantity of training data used. We train the network on 200 cores for two days. We trained several dozen networks with varying depth and width parameters and selected the candidate which offered the best compromise between accuracy and computational cost. We have tried training the best network for longer, but it does not appear to improve further.

Our training set consists of 2.2 million manually labelled characters. The training set is extracted from a random sample of the past OCR queries of users who have agreed for their data to be used to improve Google services. Text lines in the imagery are manually annotated with the class of each character and the segmentation points between characters. We do not use synthetic data or synthetic distortions. The data is used to train both our segmenter and character classifier. We first train the segmenter on the labelled segmentation points. To train the character classifier, we do not use the manually segmented characters directly since they may be dissimilar to what our automatic segmentation produces. Instead we run our segmenter on the annotated lines and choose segments visited by the beam search which have high overlap with a manually labelled character as positive training examples. We also select training examples for a "noise class" consisting of segments visited by the beam search which overlap partial/multiple ground truth characters or background clutter. The final training set consists of 45% noise examples and 55% character class examples, for 3.9 million total training examples.

For the Latin alphabet version of our system, we learn 99 character classes plus the noise class. The character classes cover upper and lowercase letters, punctuation and some additional common characters such as currency symbols. Many characters exist in multiple diacritic variants, such as "aäáâ". For training we consider these variants together as a single class, and rely on the language model to select the correct variant at recognition time. Our final output is thus from a larger space of several hundred possible character classes.

Finally we set free parameters of the system, such as the language model weights α and β, by optimizing end-to-end system performance on a validation set using Powell's method.

Self-supervised training data

We built an additional larger training set using a self-supervision mechanism. A full description of the system is beyond the scope of the paper, however we present a brief summary here. We began by running our OCR system on 5 million images from the logs of Google Goggles and Google Translate for Android. The key idea is that much of the text in these real-world images also exists verbatim somewhere on the web. If we achieve partially correct text extraction from an image, we can often locate the source text by issuing multiple web queries. We can verify the match by performing edit distance based alignment. This produces a partial ground truth for the image. An example result is shown in Figure 2. The extracted character bounding boxes come from our OCR system, but any errors in the character labels are corrected by alignment against the source text. The accuracy of labels produced by this approach is at least as good as our manually labelled data. The majority of matches come from images of dense text such as newspaper articles. However, we find matches for a wide variety of other objects such as food containers, posters, wine bottles, packaged goods, etc.

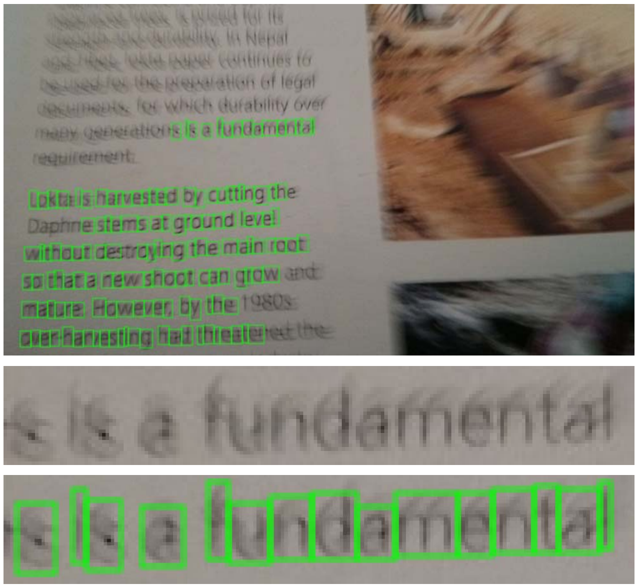

documents, for which durability over
many generations is a fundamental
requirement.
Lokta is harvested by cutting the
Daphne stems at ground level
without destroying the main root
so that a new shoot can grow and
mature. However, by the 1980s
over-harvesting had threatened the
is a fundamental

Figure 2: An example of training data automatically generated by our self-supervision approach. The green boxes show characters which have been extracted by our system and verified by alignment against an online text. The lower part of the figure shows an enlarged view of one of the automatically ground truthed regions. To preserve privacy the image shown here was taken by the authors, but it is typical of the user imagery in our logs.

From 5 million source images, we find acceptable matches for 200k, yielding 40 million automatically labelled characters. These characters are obviously biased towards easier examples, so to augment our training set we use 4 million characters with the lowest confidence scores.

5. Results

For comparison to other published work, we first present results on public OCR benchmarks. The most suitable pub-

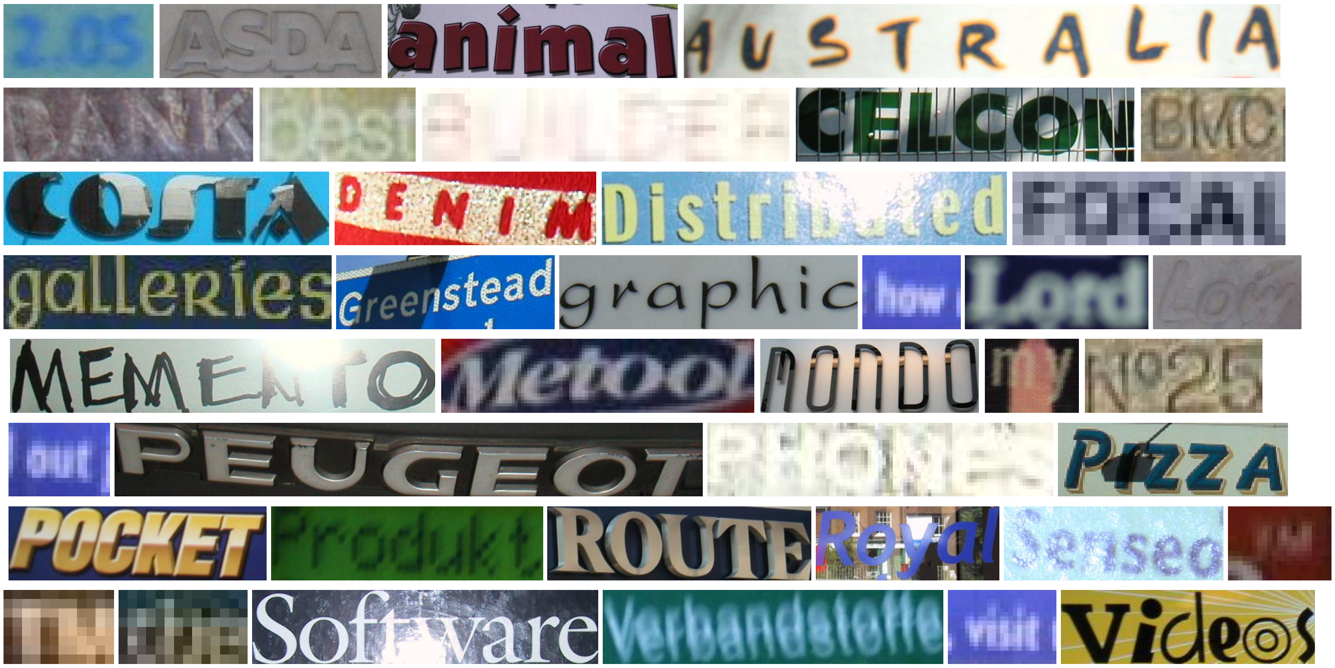


Figure 3: Some examples of text correctly read by our system. Images are from the ICDAR 2013 test set.

Algorithm	Word Recognition Rate (%)	Norm. Edit Distance
PhotoOCR	**82.83**	**122.7**
NESP	64.20	360.1
PicRead	57.99	332.4
Baseline (ABBYY)	45.30	539.0

Table 1: Results on the ICDAR 2013 Robust Reading Competition scene text test set [12], showing closest competitors to PhotoOCR on recognition rate and edit distance metrics. The baseline result is from a commercially available OCR system.

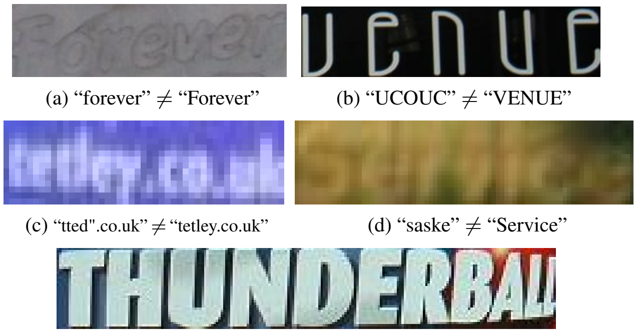
Algorithm	Word Recognition Rate (%)
PhotoOCR	**90.39**
Goel et al. [9]	77.28
Mishra et al. [15]	73.26
Novikova et al. [20]	72.9
Wang et al. [26]	70.0
Baseline (ABBYY) [9]	35.0

Table 2: Cropped word recognition accuracy on the Street View Text dataset (with lexicon) [25].

lic benchmark for unconstrained OCR is the ICDAR 2013 Robust Reading Competition scene text test set [12]. This is a cropped word recognition task with an unlimited vocabulary. Our results are shown in Table 1. Our system sets new records on both competition metrics. Our word recognition rate is 82.83%. If we ignore capitalization and punctuation differences, accuracy rises to 87.6%. Examples of true and false positives are shown in Figures 3 and 4. The complete set of classifications is available on the competition website.

Another popular benchmark is the Street View Text (SVT) dataset [25]. This benchmark measures word-spotting, a simplified OCR problem in which each image is annotated with a lexicon of about 50 words, of which one is the ground truth and the others are distractors. We have not designed our system for this task, but we can make simple use of the lexicon by performing unaided OCR and then selecting the lexicon word with smallest edit distance as the final result. Results are shown in Table 2. Our system has less than half the error rate of the nearest competitor, despite not being designed to fully exploit the task constraints. In fact, if we ignore the lexicon entirely, we still achieve 77.98% accuracy (ignoring case and punctuation differences), which would still be a new record for this dataset.

We also present results on our internal test set consisting of images sampled from the logs of Google Translate for Android. This test set reflects the performance of OCR as a smartphone input modality in real world conditions.



(a) “forever” ≠ “Forever”

(b) “UCOUC” ≠ “VENUE”

(c) “tted".co.uk” ≠ “tetley.co.uk”

(d) “saske” ≠ “Service”

(e) “THUNDERBU” ≠ “THUNDERBALL”

Figure 4: Some examples of images where the extracted text is incorrect. The captions show the recognized text versus the ground truth.

Training Set Size	Character Classifier Accuracy (%)	Word Recognition Rate (%)
1.1×10^7 (*)	92.18	70.99
3.9×10^6	91.79	70.47
1.9×10^6	90.98	68.83
9.7×10^5	90.60	69.20
4.9×10^5	89.19	65.77
2.5×10^5	88.38	60.50
1.2×10^5	86.74	53.20
6.3×10^4	85.21	46.74

Table 3: Performance versus training set size on the Google Translate test set. The training set includes noise samples, labelled characters comprised 55% of the total. Recognition rate excludes the noise class. The largest training set uses self-supervised data. See text for details.

Compared to the ICDAR imagery, typical Google Translate imagery has much lower resolution and is more often subject to blur, but is less likely to feature unusual fonts or complex backgrounds. Figure 2 is a typical example. A further salient difference in the tasks is that ICDAR tests isolated word recognition, which limits the performance gains available from language modelling. In contrast, the Google Translate task evaluates the system on lines of text. For comparability to ICDAR results, we measure performance given ground truth text detection. Results and analysis for this test set are given in Table 3 and Figures 5 and 6.

The impact of training set size on final performance is shown in Table 3. Classifier accuracy increases with training set size up to our largest available set (Figure 5). The use of self-supervised data in the largest training set seems to give results in line with the overall trend. We emphasize that our neural network configuration was chosen to balance speed and accuracy. Higher classifier performance can be achieved using a larger network, but at unacceptable computational cost.

The word recognition rate of our system incorporating all signals is shown in Figure 6 and the right column of Table 3. Full system performance also increases up to the largest training set size, although it appears to flatten a little more than classifier accuracy. Word recognition rate is measured as an exact match to the ground truth string, including case and punctuation.

We also analysed the impact of the language models on the word error rate of the overall system. A baseline system without any language model (character classifier alone) recognizes only 39.2% of words correctly. Adding the character-level language model reduces the word error rate by 39.5%. Much of this is attributable to resolving simple ambiguities such as “0” vs “O”, case consistency, etc. Adding the word level model gives a further 4% word error rate reduction. This seems small relative to the impact of the character language model, but represents resolving many harder examples that are difficult to disambiguate in any other way.

Finally, we estimated the performance impact of the beam search by computing results with a beam width 1000x larger than used in our baseline configuration. The larger beam width improves recall by only 0.5%, suggesting that approximate inference is not an important limit on performance.

Our processing time per image is 600 ms average and 1.4 secs at 95% percentile. This includes both text detection and recognition for a full image containing multiple text lines; note however that we parallelize the computation by sending each line of text to a different OCR worker. These latency measurements apply to all the presented results with the exception of the ICDAR and SVT sets, where we used a configuration of the system tuned for higher recall at greater computational cost. In this configuration no text detection is required, and recognition alone took 1.4 secs average on a single machine.

6. Conclusion

We have presented a complete system for text extraction from challenging images. Our system is built on recent machine learning approaches for improved classifier performance, combined with large training sets and distributed language modelling. The system achieves record performance on all major text recognition benchmarks, and high quality text extraction from typical smartphone imagery with sub-second latency. Our analysis demonstrates the importance of training set size, even in designs where the classifier complexity is bounded by latency constraints.

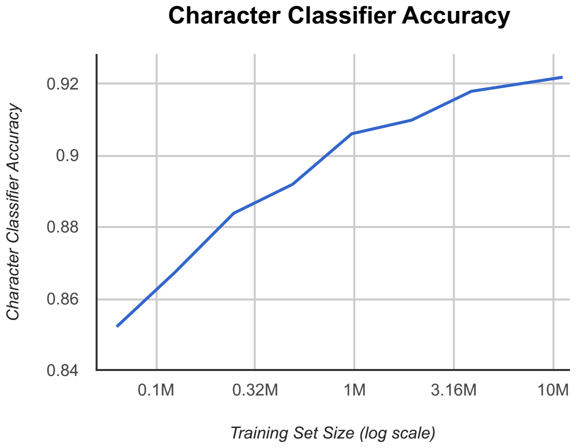


Figure 5: Character classifier accuracy versus training set size. The final point uses self-supervised data. See Table 3.

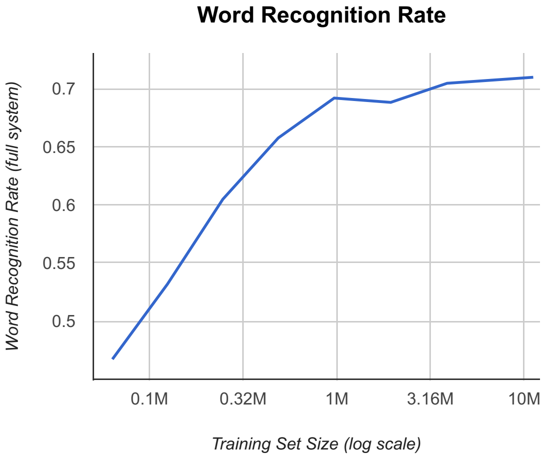


Figure 6: Word recognition rate of the complete system versus training set size. The final point uses self-supervised data. See Table 3.

References

[1] T. Brants, A. Popat, P. Xu, F. Och, and J. Dean. Large language models in machine translation. In *EMNLP*, 2007.

[2] T. Breuel. The OCRopus open source OCR system. In *IS&T/SPIE 20th Annual Symposium*, 2008.

[3] B. Carpenter. Scaling high-order character language models to gigabytes. In *ACL Workshop on Software*, 2005.

[4] X. Chen and A. Yuille. Detecting and reading text in natural scenes. In *CVPR*, 2004.

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

[6] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng. Large scale distributed deep networks. In *NIPS*, 2012.

[7] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[8] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In *CVPR*, 2010.

[9] V. Goel, A. Mishra, K. Alahari, and C. Jawahar. Whole is greater than sum of parts: Recognizing scene text words. In *ICDAR*, 2013.

[10] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[11] C. Jacobs, P. Simard, P. Viola, and J. Rinker. Text recognition of low-resolution document images. In *ICDAR*, 2005.

[12] D. Karatzas, F. Shafait, S. Uchida, M. Iwamura, L. i Bigorda, S. Mestre, J. Mas, D. Mota, J. Almazan, and L. de las Heras. ICDAR 2013 Robust Reading Competition. In *ICDAR*, 2013.

[13] F. Kimura, T. Wakabayashi, S. Tsuruoka, and Y. Miyake. Improvement of handwritten Japanese character recognition using weighted direction code histogram. *Pattern recognition*, 30(8):1329–1337, 1997.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[15] A. Mishra, K. Alahari, and C. Jawahar. Top-down and bottom-up cues for scene text recognition. In *CVPR*, 2012.

[16] V. Nair and G. Hinton. Rectified linear units improve restricted Boltzmann machines. In *ICML*, 2010.

[17] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

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

[19] W. Niblack. *An Introduction to Digital Image Processing*. Prentice Hall, 1986.

[20] T. Novikova, O. Barinova, P. Kohli, and V. Lempitsky. Large-lexicon attribute-consistent text recognition in natural images. In *ECCV*, 2012.

[21] Y. Pan, X. Hou, and C. Liu. Text localization in natural scene images based on conditional random field. In *ICDAR*, 2009.

[22] S. Russell and P. Norvig. *Artificial intelligence: A Modern Approach*. Prentice Hall, 1995.

[23] D. Smith, J. Field, and E. Learned-Miller. Enforcing similarity constraints with integer programming for better scene text recognition. In *CVPR*, 2011.

[24] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[25] K. Wang, B. Babenko, and S. Belongie. End-to-end scene text recognition. In *ICCV*, 2011.

[26] T. Wang, D. Wu, A. Coates, and A. Ng. End-to-end text recognition with convolutional neural networks. In *ICPR*, 2012.

[27] J. Weinman, E. Learned-Miller, and A. Hanson. Scene text recognition using similarity and a lexicon with sparse belief propagation. *Pattern Analysis and Machine Intelligence*, 31(10):1733–1746, 2009.