Deep Neural Networks for Text Detection and Recognition in Historical Maps

Jerod Weinman*, Ziwen Chen, Ben Gafford, Nathan Gifford, Abyaya Lamsal, and Liam Niehus-Staab
Grinnell College
Grinnell, Iowa, USA
*jerod@acm.org

***Abstract*—We introduce deep convolutional and recurrent neural networks for end-to-end, open-vocabulary text reading on historical maps. A text detection network predicts word bounding boxes at arbitrary orientations and scales. The detected word images are then normalized for a robust recognition network. Because accurate recognition requires large volumes of training data but manually labeled data is relatively scarce, we introduce a dynamic map text synthesizer providing a practically infinite stream of training data. Results are evaluated on a labeled data set of 30 maps featuring over 30,000 text labels.**

***Index Terms*—Geographic information systems (GIS), text detection, character recognition, map processing, neural networks**

I. Introduction

Maps intertwine text and graphics to represent information across multiple geographic or political features and spatial scales. These complex historical documents are accompanied by several unique challenges and opportunities. Text can appear in nearly any orientation, many sizes, and widely spaced with graphical elements or even other text within the local field of view. Many previous works have hand-crafted algorithms for dealing with such complexities.

This work brings map document processing into the deep learning era, eliminating the need for complicated algorithm design. We tailor deep-learning models and methods for finding and reading the textual component of maps. First, because text can appear in nearly any orientation in a map, we alter scene text detection methods so that the semantic baseline of the text must be learned and predicted in addition to its geometric orientation (cf. Figure 2). Second, we adapt a convolutional and recurrent neural network framework so that text recognition is robust to text-like graphical distractors. Finally, we describe a synthetic data generation process that dynamically provides the many training examples needed for accurate recognition without overfitting.

We test our methods on a new manually annotated data set of over 30 maps and 30,000 text box labels, detecting text with roughly 88% recall and precision and reading text with just 9% character error (22% word error). End-to-end performance stands at 62% recall and precision on this challenging problem.

II. Background

In this section we briefly review the current state of automated map processing, robust reading for scene text recognition, and the use of synthetic data for training reading systems. We provide many additional connections to the literature as they relate to the specifics of our text detection and recognition models in Sections III and IV, respectively.

A. Map Processing and Understanding

Deep learning approaches are now successfully reading text in scene images, but these approaches have not been widely applied to historical maps; see Chiang et al. [1] for a comprehensive review of automated map processing. Traditional map understanding systems have typically processed features bottom-up, using binarized connected component analysis [2], [3] or other engineered features [4]. After locating and identifying initial characters, Tarafdar et al. [5] add a secondary search for missing characters if the first stage output represents a partial lexicon word; exemplars of detected characters are used as models in the search. Yu et al. [6] merge OCR output from several aligned maps to reduce errors.

Several works utilize geographical dictionaries (gazetteers) to aid in recognition by automatically georeferencing maps to produce pixel-specific lexicons [7], [8]. Weinman [9] also associates cartographic character styles with geographic categories to further bias and improve text recognition.

In this work we show that deep neural networks originally designed for object recognition can be successfully adapted for automatically detecting and recognizing a wide variety of text on historical maps.

B. Robust Reading

Robust reading performance has improved significantly in the last decade. While most scene text work has addressed either detection or recognition, newer approaches are focusing on end-to-end systems and evaluations.

He et al. [10] and Liu et al. [11] simultaneously learn shared features for both text detection and text recognition by completely coupling the loss functions for both stages. This approach reduces computation, and they argue it tends to improve recall because the detection features must also perform recognition. However, the approach is limited to scenarios where training data is plentiful; typically far more labeled training examples are needed for learning to recognize text than to detect it accurately.

A lexicon assists recognition when there is strong contextual information. Rather than use a lexicon as an error-correcting post-processor, recognition processes can efficiently integrate

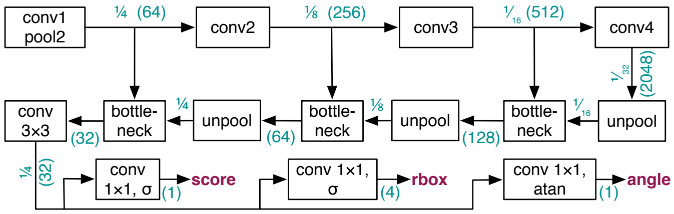

Figure 1. Map Text Detection (MapTD) network, based on EAST [24]. ResNet-50 performs feature extraction (top row) for an input image. Feature merging (middle row) upsamples and concatenates outputs from neighboring layers before applying 1×1 and then 3×3 bottleneck convolutions (output dimensions in parentheses). ReLU activations follow all convolutions *except* the output layers (bottom row), which use sigmoid (σ) or inverse tangent.

a lexicon stored in a trie, as has been done for handwriting [12], noisy documents [13], [14], and scene text [15]. Scheidl et al. [16] recently adapted the technique for the CTC model decoding task [17], which we employ in this work.

C. Synthetic Training Data

Complex models with many learned parameters require significant amounts of training data. Because labeling training data is often expensive and time-consuming, document analysis systems are often trained with artificial or augmented data streams. Displacements of existing samples can be used to create new training examples for handwriting recognition tasks [18], [19]. Individual characters [20], [21] and entire words [22] have been synthesized for scene text recognition. Gupta et al. [23] showed the importance of providing realistic contexts in augmented training data by plausibly layering synthesized words onto real images for scene text detection.

This work connects the data synthesizer to the model learner, providing a virtually infinite data stream with no disk usage and almost no overfitting.

III. MapTD: Map Text Detection Network

To detect oriented bounding boxes holding text, we train a network inspired by EAST [24] and sharing several similarities with the FOTS [11] text detection branch. First we describe the structure of the extracted features, then we detail the output channels of our model, and finally we provide the loss functions used to train the network, shown in Figure 1.[1]

A. Feature Extraction

Whereas the EAST model uses PVANet [25] as its convolutional backbone, both FOTS and our model use ResNet-50 [26]. We employ the same feature-merging branch as EAST (ultimately inspired by U-Net [27]) to process the extracted feature maps, which are $1/32$, $1/16$, $1/8$, and $1/4$ the size of the input image. The model repeatedly uses a bilinear upsampling to double a layer's feature map size, then applies a 1×1 bottleneck convolution to the concatenated feature maps followed by a 3×3 feature-fusing convolution. One last 3×3 convolution is applied to the merged features to produce the final feature map used for the output layers.

[1] github.com/weinman/MapTD; DOI: 11084/23321

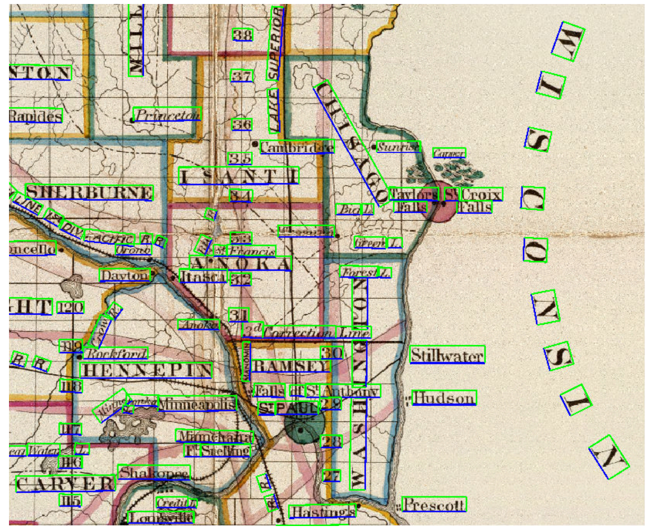

Figure 2. Ground truth rectangles with semantic orientation (text baseline in blue). Interrupted or highly spaced words are annotated in multiple rectangles.

B. Output Layers for Detection

The output convolutional feature map (lower-left of Figure 1) feeds into three dense, fully-convolutional outputs: a score map for predicting the presence of text, a box geometry map for specifying distances to a rotated rectangle's edges, and the angle of rotation. Each output map is $1/4$ the input image size.

The score map is much like that of both EAST and FOTS, but the box distance map differs in two key ways. First, rather than directly regress the distances using the raw convolutional outputs (as in EAST and its inspiration DenseBox [28]), we use a sigmoid nonlinearity to compress extremes to the range $(0, 1)$ for stability. For reconstruction, the compressed result is then multiplied by the larger dimension of the input image.

The second substantial difference is in the meaning of the output channels. In both EAST and FOTS, the channels of the box geometry map represent the distances to the top, bottom, left, and right sides of the rotated bounding box centered at the feature map's location. Importantly, these sides are assigned *geometrically*. The "top" side of an axis-aligned box is always the uppermost, regardless of whether it is a wide box with text reading horizontally or a tall narrow box with text reading vertically. In our approach, we define these edges *semantically*, so that the "top" side of a box is always the side running along the top edge of the text, the "bottom" side is the one most closely aligned with the baseline curve, and the "left" side is where one would start reading (in English). See Figure 2.

This semantic geometry has two primary benefits. First, training data labeled to include semantic orientation provides an additional helpful signal to the network. For example, rather than having to learn the arbitrary appearance of the character "R" at any orientation, it can learn to factor appearance and orientation. The secondary benefit is that the boxes produced by the detector can be easily normalized and presented to the recognition network without need for further layout analysis.

Using semantic information this way is similar to the approach of Xue et al. [29], who train a detection network to produce geometric box distances like EAST and FOTS. However, their model also learns to predict semantic boundary regions, i.e., small blocks covering the left-most character and a portion of the image beyond. Their motivation is that such regions can be distant from the central reference point used for predicting rectangle boundary distances, limiting accuracy for distant boundaries. Although the theoretical receptive field size of the Resnet-50 network is nearly 500 pixels, the effective receptive field size is likely much smaller [30].

C. Loss Functions

Each output layer in Figure 1 requires a suitable loss function. For EAST, Zhou et al. [24] shrink the regions labeled as text in the ground truth by a factor of 0.3 along the bounding box edges. As in Xue et al. [29], we treat the regions outside the smaller shrunk rectangle but within the ground truth text rectangle "don't care" regions excluded from all loss function calculations. In addition, we also exclude locations where two ground truth rectangles overlap, avoiding the need for any prioritization scheme in predicting edge distances.

1) Score: We use a version of the Dice loss [31],

$$L_{\text{score}}\left(\mathbf{p},\mathbf{t}\right) = 1 - 2\frac{\sum_i p_i t_i}{\sum_i p_i^2 + \sum_i t_i^2 + \epsilon} \tag{1}$$

where $p_i \in [0,1]$ are the values of the score map predictions and $t_i \in \{0,1\}$ are the values in the corresponding ground truth at each location i. We add $\epsilon = 10^{-5}$ for numerical stability (i.e., when the ground truth contains no text).

2) Rbox: For the rotated rectangle predictions, we use the IoU loss (as in EAST [24]). Let d_i^e represent the predicted distance to the specified edge e (top, bottom, left, or right) from location i, and g_i^e the distance as marked in the ground truth. Note these values are only defined within the shrunk rectangles where $t_i = 1$; all other locations are excluded from the loss function. The area of the intersection and union of the rectangles defined by $\mathbf{d}_i$ and $\mathbf{g}_i$ are given by the following:

$$\begin{aligned} \left|\mathbf{d}_i \cap \mathbf{g}_i\right| &= \left(\min\left(d_i^{\text{t}}, g_i^{\text{t}}\right) + \min\left(d_i^{\text{b}}, g_i^{\text{b}}\right)\right) \cdot \\ &\quad \left(\min\left(d_i^{\text{l}}, g_i^{\text{l}}\right) + \min\left(d_i^{\text{r}}, g_i^{\text{r}}\right)\right) \qquad (2) \\ \left|\mathbf{d}_i \cup \mathbf{g}_i\right| &= \left|\mathbf{d}_i\right| + \left|\mathbf{g}_i\right| - \left|\mathbf{d}_i \cap \mathbf{g}_i\right|, \qquad (3) \end{aligned}$$

where the rectangle area is $|\mathbf{r}| \triangleq \left(r^{\text{t}} + r^{\text{b}}\right)\left(r^{\text{l}} + r^{\text{r}}\right)$. The loss is an average over training locations i,

$$L_{\text{rbox}}\left(\mathbf{d},\mathbf{g}\right) = \frac{1}{N}\sum_{i:t_i=1} -\log\frac{\left|\mathbf{d}_i \cap \mathbf{g}_i\right| + 1}{\left|\mathbf{d}_i \cup \mathbf{g}_i\right| + 1}, \tag{4}$$

where the numerator and denominator are shifted for numerical stability and N is the total number of training locations.

3) Angle: The rotation angle loss is simply the cosine loss

$$L_{\text{angle}}\left(\boldsymbol{\theta},\boldsymbol{\theta}^*\right) = \frac{1}{N}\sum_{i:t_i=1} 1 - \cos\left(\theta_i - \theta_i^*\right), \tag{5}$$

where θ_i and θ_i^* are predicted and actual rectangle angles.

The total loss is finally given by

$$L_{\text{total}} = \varsigma L_{\text{score}}\left(\mathbf{p},\mathbf{t}\right) + \rho L_{\text{rbox}}\left(\mathbf{d},\mathbf{g}\right) + \alpha L_{\text{angle}}\left(\boldsymbol{\theta},\boldsymbol{\theta}^*\right) \tag{6}$$

Our experiments use the weights $\varsigma = 0.01$, $\rho = 1$, and $\alpha = 20$.

Table I
ARCHITECTURE OF THE TEXT RECOGNITION NETWORK.

Layer	Op	Krn. Sz.	Stride (v,h)	Out Dim	H	ΔW	Pad
0	Input			1	32		
1	Conv	3×3	1,1	64	30	−2	valid
	Conv	3×3	1,1	64	30		same
	Max Pool	2×2	2,2	64	15	//2	valid
2	Conv	3×3	1,1	128	15		same
	Conv	3×3	1,1	128	15		same
	Max Pool	2×2	2,1	128	7	−1	valid
3	Conv	3×3	1,1	256	7		same
	Conv	3×3	1,1	256	7		same
	Max Pool	2×2	2,1	256	3	−1	valid
4	Conv	3×3	1,1	512	3		same
	Conv	3×3	1,1	512	3		same
	Max Pool	3×1	3,1	512	1		valid
5	Bi-LSTM			512	1		
	Bi-LSTM			512	1		
6	Output			62	1		

D. Locality-Aware Non-Maximal Suppression

We threshold scores p_i to densely identify candidate rectangle locations i. To reduce the $O\left(M^2\right)$ computational overhead of filtering M score-thresholded predictions where $p_i > \tau$ with standard NMS, we adopt the so-called Locality-Aware NMS method of Zhou et al. [24], which iteratively merges (rather than selects) geometries of thresholded predictions row by row. We modify their routine to produce an average score for the weighted rectangles, rather than a total score. Thus, each final merged prediction is accompanied by a meaningful score that can be used downstream (e.g., for recognition and calculating the average precision metric).

IV. MAP TEXT RECOGNITION NETWORK

To recognize text predicted by the detection branch, we use a stacked, bidirectional LSTM built on CNN features and trained with CTC loss.[2] The model is an adaptation of Shi et al.'s CRNN (convolutional recurrent neural network) architecture [32] (details in Table I). Every convolutional layer uses a ReLU activation. While our model resembles the Shi et al. CRNN—an adaptation of the deep VGG architecture [33] for reading English—we note some important differences.

Their CRNN uses only a single 3×3 convolution in the first two conv/pool stages. In our network, every convolutional layer uses a paired sequence of 3×3 kernels, which increases the theoretical receptive field of each feature. This allows us to omit the computationally expensive $2 \times 2 \times 512$ final convolutional layer in their CRNN. In its place, we max pool over the three rows of features to collapse to a single 512-dimensional feature vector for each time step (i.e., horizontal spatial location) used as input to the first bidirectional LSTM layer. These changes preserve the receptive field while reducing the number of convolution parameters by 15%.

The CRNN's first convolutional layer uses zero-padding; applied to a grayscale image, this can cause spurious filter responses around the border. We instead opt to trim the first

[2] `github.com/weinman/cnn_lstm_ctc_ocr`; DOI:11084/23322

convolution map to valid responses only, using no padding. Because the kernel is only 3×3, this erodes only a single pixel from around the outside edge while retaining the ability to make distinctions about fine-details near the edges with a relatively limited pool of 64 first-layer filters. After this initial stage, the convolutional maps can essentially learn to emit positive or negative correlations (then ReLU trimmed), so zero-padding in the subsequent stages remains sensible.

We use batch normalization after the second convolution of each layer (1–4). Including the first two layers (unlike CRNN) adds computation but should decrease the number of iterations necessary for convergence.

The output feature map must have at least one horizontal "pixel" per character. The two initial stages of the CRNN downsample with a stride of two in both spatial dimensions. We preserve sequence length by downsampling horizontally only after the first max pooling stage. This allows our model to recognize shorter sequences of compact characters. Specifically, a cropped image of single character (or digit) can be as narrow as 8 pixels and two characters can be as narrow as 10 pixels wide. Shi et al. [32] compensate for this limitation by instead scaling inputs to be at least 100 pixels wide, distorting the images. For recognition, we normalize the input image height to 32 pixels, preserving the aspect ratio and horizontally extending a cropped region to a minimum width of 8 pixels.

Like the CRNN, the output layer feeds into a CTC loss layer [17, pp. 68–70], which connects network outputs to an unaligned label sequence of characters by marginalizing over all alignments via dynamic programming. We restrict our model to alphanumeric outputs and the "empty" output (used for signaling repeated characters) $y \in [\mathtt{A-Za-z0-9}] \cup \{\epsilon\}$, using case-sensitive training and evaluation.

A. Open Vocabulary Recognition

In the CTC model, one label is emitted for each time step t in the sequence $(1 \ldots T)$; because a label can be repeatedly emitted for the same input object, repeated labels in an input sequence $\mathbf{y}$ are collapsed and the model must learn to emit a "blank" label ϵ (the empty string) that also functions as a sentinel for a given labeling. The inference task for recognition is then to find the collapsed label sequence $\mathbf{c}$ with the highest score, defined as the sum of the scores for all uncollapsed sequences $\boldsymbol{\pi}$ projecting to that final sequence:

$$P(\mathbf{c} \mid \mathbf{x}) = \sum_{\boldsymbol{\pi}:\mathcal{F}(\pi)=\mathbf{c}} \prod_{t=1}^{T} y_{\pi_t}^{t} \tag{7}$$

where $\boldsymbol{\pi}$ is the raw sequence of labels, $\mathcal{F}$ is the collapsing operation and y_k^t is the network's (softmax) score for output character k at time step t, given an input image $\mathbf{x}$.

Finding the most probable label sequence $\mathbf{c}^*$ for a given input image is generally intractable. Shi et al. [32] use a greedy labeling, choosing the best label at each time step before collapsing the sequence. We instead use beam search [17] (with a beam width of 128) to find the approximate best label sequence. Though more expensive—taking $8\times$ longer to run—it reduces error by 20% over greedy search in our experiments.

B. Lexicon-based Recognition

Shi et al. [32] use the best unconstrained output sequence to find the lexicon words with lowest edit distance. These candidates can then be scored exactly by dynamic programming. Because maps contain distractor ink that often looks much like text (e.g., city markers, and a variety of other lines), we cannot assume that the unconstrained output will be close to the correct sequence. Instead, we restrict the beam search to a trie-organized lexicon [16] so that only dictionary words can be produced.[3] (See Section VI-A2 for lexicon details.)

Because geography constrains cartography, some map locations have limited lexical possibilities. Other markings such as road numbers or distances are far less restricted. We therefore employ a hybrid vocabulary mode for recognition in which we establish a fixed prior probability λ for a word being from a lexicon. Let $\hat{\mathbf{c}}_U \triangleq \operatorname{argmax}_{\mathbf{c}} P(\mathbf{c} \mid \mathbf{x})$ be the best unconstrained word and $\hat{\mathbf{c}}_L \triangleq \operatorname{argmax}_{\mathbf{c} \in L} P(\mathbf{c} \mid \mathbf{x})$ be the best word drawn from a lexicon L. We then choose the collapsed labeling with the higher posterior probability

$$\hat{\mathbf{c}} = \begin{cases} \hat{\mathbf{c}}_L & \lambda P(\hat{\mathbf{c}}_L \mid \mathbf{x}) > (1-\lambda) P(\hat{\mathbf{c}}_U \mid \mathbf{x}) \\ \hat{\mathbf{c}}_U & \text{otherwise.} \end{cases} \tag{8}$$

This formulation assumes the input is independent of the lexicon: $P(\mathbf{x} \mid L) = P(\mathbf{x})$. As Graves points out [17, p. 75], this assumption is false in general but works well in practice. We use $\lambda = 0.999$ in our experiments, corresponding to a log-probability bias of -3 against non-dictionary words.

V. Dynamic Map Text Synthesis

The original CRNN model [32] was trained on MJSynth [22] (synthetic images for scene text recognition) and subsequently tested on a variety of scene text recognition benchmarks without any fine-tuning, often achieving superior performance. Our experiments show that the MJSynth data is ill-suited for recognizing map text, which exhibits different factors complicating OCR. Because large amounts of training data are needed to fit the highly parameterized recognition model in a way that generalizes well to previously unseen data, we have created our own synthetic map text generator.

Rather than create a fixed synthetic data set for training, we *dynamically* synthesize images on-the-fly as part of the training process. Because the training algorithm never sees the same image twice, it must learn to focus on the general patterns in the data, rather than specific images. Although there are 7.2 million training images in MJSynth, our recognition model (Table I) has 15.2 million parameters. Experiments below show that the model begins to overfit after 8 epochs of MJSynth data (Figure 6). Using the map text synthesizer, we train on over 112 million distinct images (equivalent to more than 15 MJSynth epochs) without overfitting.

Map word image synthesis divides into three primary components (see Figure 3). A text and background layer are first rendered as vector graphics, which are then merged and rasterized for post-processing. The synthesizer has nearly

[3] `github.com/weinman/CTCWordBeamSearch`; DOI:11084/23323

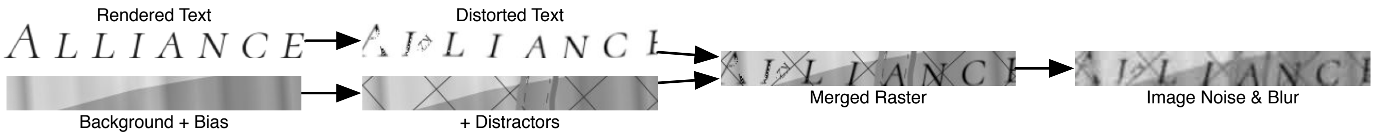

Figure 3. Dynamic map text synthesis pipeline.

Figure 4. Sample training and test images for text recognition. TOP: Training word images from synthesizer. BOTTOM: Test word images cropped from maps.

200 configurable parameters, all of which are detailed (with defaults) in the provided code.[4]

Background Layer: To assist the recognition model in learning what to ignore, the synthesizer mimics a wide variety of cartographic features that complicate text recognition. The background is divided into multiple regions, each with its own brightness. We create a bias field by randomly selecting anchor points for linear gradients blended with the piecewise background brightness. Finally, different kinds of distractor marks are simulated: thick boundaries, independent lines and curves (like rivers, roads, and railroads), grids, parallel lines or curves with varying distances, ink splotches, points (like city markers) and textures (with regular polygons as the textons). In addition, a layer of overlapping distractor text is added with a small probability to random locations and orientations.

Text Layer: The text caption is chosen randomly from a list of possible strings or a random sequence of digits, followed by the font (typeface, size, and weight) and letter spacing for rendering. The horizontal scale is modified before rotating and rendering the text along a curved baseline. To simulate document defects, circular spots are randomly cut from the text and the layer's opacity can be lowered. Image padding/cropping is added to simulate poor localization.

Image Post-processing: After merging the text over the background layer, we add Gaussian noise to the rasterized image, followed by a blur and JPEG compression artifacts.

VI. EXPERIMENTS

This section details the experimental setup and parameters, providing a comprehensive evaluation of our approach.[5]

A. Benchmark and Supporting Data

1) Maps: Our benchmark data [34] is an independent re-annotation of a previous benchmark featuring 31 historical U.S. maps (1866–1927). Every piece of text (place names, highway numbers, coordinates, graticule labels, etc.) is marked with a ground truth Unicode string transcription and bounding polygon. When its typographic kerning or tracking (intra-word, inter-character spacing) is large or other map words cross through, the word annotation is segmented into multiple bounding polygons. For accurate transcription, many words required cross-reference in historical cartographic resources. The annotations total 33,868 segments in 32,659 words.

To increase the amount of training data and statistical reliability of our results, we use a ten-fold cross-validation with 27 maps for training and 3 maps for testing each model. One map is held out for validation across all folds. We also ran separate "leave-one-atlas-out" experiments, with maps divided into eight folds by the atlas from which they originated, so that all the maps in the test set come from the same atlas and no examples from that atlas are in the training set.

2) Gazetteer and Lexicon: While some of the text on a map is numbers, individual characters, or other non-dictionary strings, most of the text is either a toponym (proper noun place-name), or a general lexicon word. We leverage this property to improve recognition.

For general recognition, we create a lexicon from SCOWL,[6] with American English including proper names, abbreviations, and contractions (with all punctuation removed) up to the 50th frequency percentile, adding uppercase and leading caps versions of all words for a total of 237,873 words.

To support the special task of toponym recognition, we retain the identification of each map's region (i.e., a list of states or counties) in the original dataset. With this extra annotation, we generate a gazetteer of words from placenames by harvesting words from the following feature categories of the US Geographic Names Information System (GNIS)[7]: *Bay*, *Cape*, *Civil*, *Island*, *Lake*, *Military*, *Park* (State or National), *Populated Place*, *Post Office* (historical), *Range*, *Reservoir*, *Stream*, *Summit*, and *Swamp*. After adding uppercase variants, this yields a median of 34,400 toponyms per map, and a median combined lexicon size of 261,200 words per map.

[4] `github.com/weinman/MapTextSynthesizer`; DOI:11084/23324

[5] Code and data archived at `https://hdl.handle.net/11084/23320`

[6] `http://wordlist.sourceforge.net`, revision 6

[7] `https://geonames.usgs.gov`

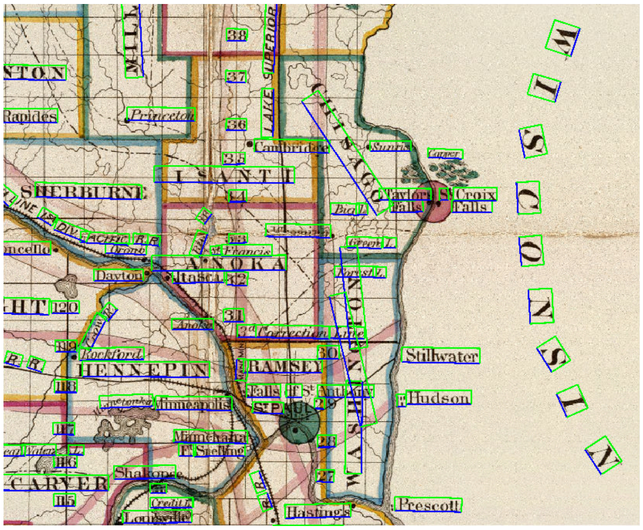

Figure 5. Detected text rectangles (predicted baseline in blue).

B. Word Detection

To evaluate the MapTD network (Section III), we train and test on various splits of the benchmark map data.

1) Training Details: We train the detection network by minibatch stochastic gradient descent using the Adam optimizer with the default values for the momentum hyperparameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ as reported by Kingma and Lei Ba [35]) and a minibatch size of 16 images. We use a learning rate of 10^{-4} for the first 2^{17} steps and then 10^{-5}, training for a total of 2^{20} steps. Throughput is roughly 1.9 batches/sec on a Titan X GPU. Because the full map images are far too large to fit into GPU memory, we randomly sample 512×512 tiles from the training data. Text sizes already vary widely in the data, so we do not rescale training images. For testing, we generate predictions on overlapping 4096×4096 tiles of the image (striding by half the tile size), using non-maximal suppression to filter the redundant predictions.

2) Results: Table III highlights results training with ten-fold cross-validation. As is done by the ICDAR Robust Reading Competition for scene text [36], the "Overall" results are a pooled tally for all the ground truth rectangles in all test map images (30 in total). Ground truth rectangles in Figure 2 can be compared to the predictions in Figure 5.

Leave-one-atlas-out training results in a 9% drop in F-score. Experience with similar fonts and map styles may be important for the trained network performance, but lower performance may also be caused by smaller training sets (i.e., maps from the same atlas will each have fewer training examples). The change in test performance is slightly more correlated with change in overall training set size ($\rho = 0.6777$) than with the number of training maps from the same atlas ($\rho = 0.6425$).

C. Word Recognition

To evaluate the recognition capability of the word model (described in Section IV), we test it on the cropped words from the ground truth map data. We train the model using either the MJSynth data, the MapTextSynth data stream (described in Section V), or the actual map data.

1) Training Details: We also use Adam to train the recognition network. We did not explore alternate values for the momentum hyperparameters, but setting an appropriate schedule for both the learning rate *and* batch size was critical for good performance. Takase et al. [37] show that noisy gradients from small minibatches helps avoid sharp local minima early in training. Increasing minibatch size later in training reduces noise, thereby increasing the stability of the loss function minimization. Their work builds on Smith et al. [38], who argue for increased batch sizes rather than decayed learning rates. However, we found that decreasing the learning rate also became necessary to continue improving the model once the training loss leveled off but remained noisy. We therefore use a schedule that decreases the learning rate in 5 dB increments, doubles the batch size, or both (Table II).

Table II
TRAINING SCHEDULE FOR RECOGNITION NETWORK

Stage	1	2	3	4	5
End Step	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Learning Rate	1e–4	3e–5	3e–5	1e–5	3e–6
Minibatch Size	16	32	64	128	128

We rescale the input image range to $[-0.5, 0.5]$. With enough threads running the map text synthesizer to keep the training queue full, throughput shifts from approximately 2 batches/sec in stage 1 to just under 1 batch/sec in stage 5 on an NVIDIA Tesla K40 GPU. Captions for the dynamic map text are drawn from the GNIS gazetteer (Section VI-A2) for the entire U.S. or random digit strings.

2) Results: Figure 6 shows the relative performance for the various combinations of training and test data. Training on the MJSynth data yields the best performance on the test MJSynth data, but its test performance on real maps is much worse. Moreover, the MJSynth-trained model begins to overfit on both test sets after 2^{19} training steps. Training with data from our map text synthesizer yields far better performance on maps than training with MJsynth and nearly matches the performance of training on the actual map data itself. While the model clearly begins to overfit when trained on the little real map data available, virtually no overfitting occurs when training with synthetic map text. Training and testing on the case insensitive, closed vocabulary MJSynth data, our model's word error rate (1.82%) is about 14% lower than CRNN's.

Table III reports results on each of the maps (grouped by atlas) and the pooled results for each task. Ground truth boxes that have no alphanumeric characters are ignored as "don't cares" for word recognition and end-to-end evaluation.

Table IV compares the results for the cropped word recognition task (i.e., ground truth detections are provided) with open vocabulary (no lexicon), closed vocabulary (lexicon-restricted) and mixed (hybrid) vocabulary modes. Allowing a prior probability (mixed vocabulary) for lexicon words reduces error over a closed-vocabulary approach by 8%. Ignoring the held-out validation data, the lowest test error occurs when training

Table III
INDIVIDUAL MAP RESULTS, GROUPED BY ATLAS. DETECTIONS TRAINED BY TEN-FOLD VALIDATION. RECOGNITION TRAINED WITH MAPTEXTSYNTH.

			Detection					Recognition			End-to-End			
Map	Region	Test Fold	Num. Boxes	Prec.	Recall	F1	AP	Num. Words	Char Err.(%)	Word Err.(%)	Prec.	Recall	F1	AP
1592006	Central Calif.	4	653	77.39	57.12	65.73	53.55	652	13.95	35.58	52.62	35.43	42.35	19.56
5370006	Florida	9	571	82.46	77.41	79.86	71.06	568	15.94	39.96	39.53	35.21	37.24	13.85
5370026	New Mexico	8	400	82.86	79.75	81.27	72.96	397	16.05	32.75	48.64	45.20	46.86	23.56
1070001	Ohio	6	1255	93.83	70.28	80.36	77.32	1242	19.99	24.80	62.82	43.00	51.05	26.59
1070002	Indiana	1	589	87.74	78.95	83.11	76.06	579	10.68	31.43	47.29	40.76	43.78	20.13
1070003	Illinois	1	766	87.06	84.33	85.68	78.70	761	10.44	19.71	58.87	54.53	56.62	33.97
1070004	Michigan	0	908	86.53	82.05	84.23	72.53	901	20.68	40.18	42.17	37.96	39.95	17.60
1070005	Wisconsin	9	586	87.43	79.52	83.29	75.53	579	12.86	32.30	55.69	48.19	51.67	28.09
1070006	Minnesota	7	995	89.35	86.03	87.66	80.87	988	12.38	30.06	48.47	45.09	46.72	22.40
1070007	Iowa	7	641	89.17	88.61	88.89	84.17	639	16.07	37.25	45.42	43.51	44.44	19.35
1070009	Missouri	0	875	83.72	81.71	82.71	69.93	871	12.81	30.65	52.24	48.28	50.18	26.16
1070010	Arkansas	9	505	91.07	90.89	90.98	86.31	504	8.35	20.44	65.22	62.50	63.83	44.77
1070012	Mississippi	8	449	95.68	93.76	94.71	90.67	448	6.46	18.08	64.10	61.38	62.71	41.50
1070013	Alabama	3	474	83.81	87.34	85.54	83.24	469	10.52	29.21	51.06	51.17	51.12	23.89
1070015	Kansas/Nebr.	2	1394	88.21	76.18	81.76	79.84	1385	18.13	39.93	46.36	37.26	41.31	17.73
0019007	U.S. States	2	354	33.02	29.66	31.25	14.71	353	24.54	43.91	14.29	10.48	12.09	1.85
5235001	Can./NW U.S	4	671	71.67	76.90	74.19	69.98	656	10.25	29.27	41.81	40.85	41.33	13.70
5242001	Missouri	6	607	89.01	82.70	85.74	82.81	584	6.13	22.60	63.07	57.02	59.89	36.49
5755018	Illinois	5	2140	98.45	97.94	98.20	98.25	2138	2.92	8.28	85.51	84.75	85.13	71.71
5755024	Iowa	3	2447	97.59	97.47	97.53	98.14	2444	3.41	9.94	86.47	84.70	85.57	71.87
5755025	Nebraska	7	1910	98.52	97.38	97.95	98.26	1908	3.36	9.70	87.49	85.74	86.61	72.24
5755033	Colorado	1	1927	98.08	95.23	96.63	96.33	1925	4.56	12.47	86.03	82.18	84.06	71.35
5755035	Wyoming	3	1379	96.36	94.20	95.27	94.01	1377	5.88	17.21	79.14	75.74	77.40	60.31
5755036	Montana	4	1613	77.71	89.03	82.98	84.69	1605	6.14	15.83	64.15	71.03	67.42	45.33
5028052	N. Carolina	5	1818	92.84	87.68	90.18	88.48	1779	7.53	21.75	67.45	61.38	64.27	39.80
5028054	S. Carolina	8	1407	90.68	88.49	89.57	87.23	1372	10.24	27.99	64.06	60.42	62.19	39.70
5028097	Minneapolis	2	702	92.11	86.47	89.20	86.93	669	6.62	23.32	64.14	58.04	60.94	38.61
5028100	N. Dakota	5	1595	91.34	91.22	91.28	88.85	1569	8.16	24.86	62.24	58.49	60.30	33.85
5028102	S. Dakota	6	1803	93.63	85.64	89.46	86.02	1735	6.75	17.93	73.12	65.07	68.86	45.68
5028149	Oregon	0	1881	91.77	81.18	86.15	80.38	1850	10.26	21.30	71.46	58.12	64.10	42.73
Overall			33315	90.45	86.56	88.46	84.77	32947	9.04	22.13	66.87	61.49	64.07	40.37

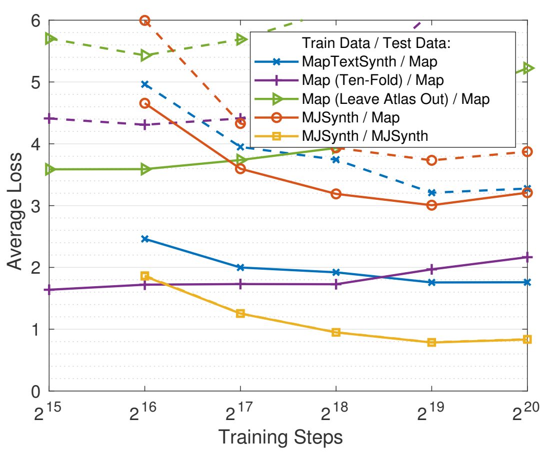

Figure 6. Performance comparison of various train/test data configurations for word recognition tasks using MJSynth, MapTextSynth, and real maps as training data. Dashed lines indicate held-out validation set performance.

Table IV
CASE-SENSITIVE TEXT RECOGNITION RESULTS ON 30 MAPS.

Training	Char. Error (%)			Word Error (%)		
Data	Open	Closed	Mixed	Open	Closed	Mixed
MJSynth [22]	21.22	18.03	17.88	50.77	37.59	37.24
Maps (Atlas)	22.12	17.25	16.92	53.40	39.02	38.60
Maps (10×)	13.03	9.19	8.28	36.61	23.53	21.54
MapTextSynth	13.64	9.88	9.04	37.88	24.05	22.13

to stage 4 on real map data with ten-fold cross-validation, yielding 7.07% character error (18.29% word error).

The larger number of LSTM units (512) seems to be required; using 256 (as in CRNN [32]) increased open-vocabulary character error by 43%. Training a model with 1024 units takes longer (in both time per step and number of steps) and performs worse with the training schedule above.

Our system's cropped word recognition performance (Table IV) is on par with results on the same map images for a smaller set of annotations (e.g., highway numbers and legends were excluded) [9], but which were *manually* normalized (baseline and font size) for testing—an advantage not provided here.

D. End-to-End Performance

The MapTD detections are cropped and normalized to horizontal rotation and image height of 32 pixels, preserving aspect ratio while inflating the size-normalized box width to at least 8 pixels (the minimum required to predict one character). Predictions are determined to be a match if the IoU > 0.5 and the text transcription matches. Inspection of the detection results indicates that poor horizontal word beginning/ending localization increases end-to-end word recognition error rate.

The results in Table III use the semantic orientation described in Section III-B. Training using the geometric rectangle orientation of EAST lowers end-to-end F-score by 3.6%.

VII. Conclusions

Many tools and domain-specific algorithms have been developed for automated map understanding. This work demonstrates that with sufficient training data, deep-learning systems can be easily tailored to extract text from complex historical map images. We expect that pixel-specific lexicons provided by georectification will significantly improve end-to-end results, as they have for cropped word recognition [7], [8], [9].

Scene text extraction has been improved by sharing features with recognition [10], [11], a method that requires training both detection and recognition models on the same data. We show that achieving generalized, usable toponym recognition requires far more training data than available within the labeled training maps. Extending the map text synthesizer to produce plausible small map regions (analogous to synthetic scene text [23]) may also allow for improvement by future integration. The CNN portions of our detection and recognition models remain quite distinct. Future work might include how to train these distinct models jointly or whether a single model with shared features would suffice for this task.

Acknowledgments

The authors thank L. Goldberg for software update assistance and S. Ilic for a critical bug fix. This material is based upon work supported by NVIDIA and the National Science Foundation under Grant No. 1526350. HPC resources were provided by UMass Amherst CICS under a grant from the Collaborative R&D Fund managed by the Mass. Tech. Collaborative.

References

[1] Y.-Y. Chiang, S. Leyk, and C. A. Knoblock, "A survey of digital map processing techniques," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 1:1–1:44, May 2014.

[2] L. A. Fletcher and R. Kasturi, "A robust algorithm for text string separation from mixed text/graphics images," *IEEE Trans. PAMI*, vol. 10, no. 6, pp. 910–918, 1988.

[3] R. Cao and C. L. Tan, "Text/graphics separation in maps," in *Graphics Recognition*, ser. LNCS, D. Blostein and Y.-B. Kwon, Eds., 2002, vol. 2390, pp. 167–177.

[4] Y.-Y. Chiang and C. A. Knoblock, "Classification of line and character pixels on raster maps using discrete cosine transformation coefficients and support vector machine," in *Proc. ICPR*, vol. 2, 2006, pp. 1034–1037.

[5] A. Tarafdar, U. Pal, P. P. Roy, N. Ragot, and J.-Y. Ramel, "A two-stage approach for word spotting in graphical documents," in *Proc. ICDAR*, 2013, pp. 319–323.

[6] R. Yu, Z. Luo, and Y.-Y. Chiang, "Recognizing text on historical maps using maps from multiple time periods," in *Proc. ICPR*, 2016, pp. 3993–3998.

[7] R. Pawlikowski, K. Ociepa, U. Markowska-Kaczmar, and P. B. Myszkowski, "Information extraction from geographical overview maps," in *Proc. ICCCI*, 2012, pp. 94–103.

[8] J. Weinman, "Toponym recognition in historical maps by gazetteer alignment," in *Proc. ICDAR*, 2013, pp. 1044–1048.

[9] ——, "Geographic and style models for historical map alignment and toponym recognition," in *Proc. ICDAR*, 2017.

[10] T. He, Z. Tian, W. Huang, C. Shen, Y. Qiao, and C. Sun, "An end-to-end textspotter with explicit alignment and attention," in *Proc. CVPR*, 2018, pp. 5020–5029.

[11] X. Liu, D. Liang, S. Yan, D. Chen, Y. Qiao, and J. Yan, "FOTS: Fast oriented text spotting with a unified network," in *Proc. CVPR*, 2018, pp. 5676–5685.

[12] C.-L. Liu, M. Koga, and H. Fujisawa, "Lexicon-driven segmentation and recognition of handwritten character strings for Japanese address reading," *IEEE Trans. PAMI*, vol. 24, no. 11, pp. 1425–1437, November 2002.

[13] S. M. Lucas, G. Patoulas, and A. C. Downton, "Fast lexicon-based word recognition in noisy index card images," in *Proc. ICDAR*, vol. 1, 2003, pp. 462–466.

[14] C. Jacobs, P. Y. Simard, P. Viola, and J. Rinker, "Text recognition of low-resolution document images," in *Proc. ICDAR*, 2005, pp. 695–699.

[15] J. Weinman, Z. Butler, D. Knoll, and J. Feild, "Toward integrated scene text reading," *IEEE Trans. PAMI*, vol. 36, no. 2, pp. 375–387, Feb. 2014.

[16] H. Scheidl, S. Fiel, and R. Sablatnig, "Word beam search: A connectionist temporal classification decoding algorithm," in *Proc. Intl. Conf. on Frontiers in Handwriting Recognition*, 2018, pp. 253–258.

[17] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, ser. SCI. Springer, 2012, vol. 385.

[18] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Proc. ICDAR*, vol. 2, 2003, pp. 958–963.

[19] T. Varga and H. Bunke, "Generation of synthetic training data for an HMM-based handwriting recognition system," in *Proc. ICDAR*, vol. 1, 2003, pp. 618–622.

[20] J. J. Weinman and E. Learned-Miller, "Improving recognition of novel input with similarity," in *Proc. CVPR*, 2006, pp. 308–315.

[21] T. E. de Campos, B. R. Babu, and M. Varma, "Character recognition in natural images," in *Proc. VISAPP*, 2009.

[22] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Reading text in the wild with convolutional neural networks," *Intl. J. Comp. Vis.*, vol. 116, no. 1, pp. 1–20, 2016.

[23] A. Gupta, A. Vedaldi, and A. Zisserman, "Synthetic data for text localisation in natural images," in *Proc. CVPR*, 2016, pp. 2315–2324.

[24] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "EAST: An efficient and accurate scene text detector," in *Proc. CVPR*, 2017, pp. 2642–2651.

[25] S. Hong, B. Roh, K.-H. Kim, Y. Cheon, and M. Park, "PVANet: lightweight deep neural networks for real-time object detection," *arXiv:1611.08588*, 2016.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016, pp. 770–778.

[27] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. MICCAI*, 2015, pp. 234–241.

[28] L. Huang, Y. Yang, Y. Deng, and Y. Yu, "Densebox: Unifying landmark localization with end to end object detection," *arXiv:1509.04874*, 2015.

[29] C. Xue, S. Lu, and F. Zhan, "Accurate scene text detection through border semantics awareness and bootstrapping," in *Proc. ECCV*, ser. LNCS, vol. 11220, 2018, pp. 370–387.

[30] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," in *Proc. NIPS*, 2016, pp. 4898–4906.

[31] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-Net: Fully convolutional neural networks for volumetric medical image segmentation," in *Proc. 3DV*, 2016, pp. 565–571.

[32] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE Trans. PAMI*, vol. 39, no. 11, pp. 2298–2304, 2017.

[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. ICLR*, 2016.

[34] A. Ray, Z. Chen, B. Gafford, N. Gifford, J. Jai Kumar, A. Lamsal, L. Niehus-Staab, J. Weinman, and E. Learned-Miller, "Historical map annotations for text detection and recognition," Grinnell College, Grinnell, Iowa, Tech. Rep., August 2018, http://hdl.handle.net/11084/23294.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2015.

[36] R. Gomez, B. Shi, L. Gomez, L. Numann, A. Veit, J. Matas, S. Belongie, and D. Karatzas, "ICDAR2017 robust reading challenge on COCO-Text," in *Proc. ICDAR*, 2017, pp. 1435–1443.

[37] T. Takase, S. Oyama, and M. Kurihara, "Why does large batch training result in poor generalization? A comprehensive explanation and a better strategy from the viewpoint of stochastic optimization," *Neural Computation*, vol. 30, no. 7, pp. 2005–2023, 2018.

[38] S. L. Smith, P.-J. Kindermans, and Q. V. Le, "Don't decay the learning rate, increase the batch size," in *Proc. ICLR*, 2018.