What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis

Jeonghun Baek[1] Geewook Kim[2*] Junyeop Lee[1] Sungrae Park[1]
Dongyoon Han[1] Sangdoo Yun[1] Seong Joon Oh[1] Hwalsuk Lee[1†]
[1]Clova AI Research, NAVER/LINE Corp. [2]Kyoto University
{jh.baek,junyeop.lee,sungrae.park,dongyoon.han,sangdoo.yun,hwalsuk.lee}@navercorp.com
geewook@sys.i.kyoto-u.ac.jp coallaoh@linecorp.com

A. Contents

B. Dataset Matters in STR - examples

IC03 - 7 missing words boxes in 860 evaluation dataset. The original IC03 evaluation dataset has 1,110 images, but prior works have conducted additional filtering, as described in §2. All papers have ignored all words that are either too short (less than 3 characters) or ones that contain non-alphanumeric characters. Although all papers have supposedly applied the same data filtering method and should have reduced the evaluation set from 1,110 images to 867 images, the reported example numbers are different: either the expected 867 images or a further reduced 860 images. We identified the missing examples as shown in Figure A.

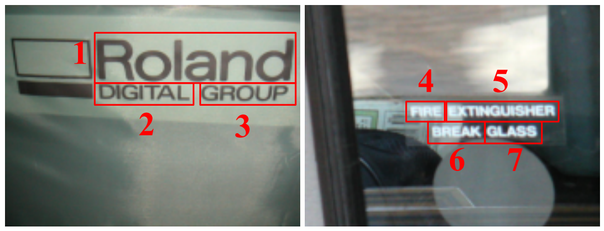

Figure A: 7 missing examples of IC03-860 evaluation dataset. The examples represented by the red-colored word boxes of the two real scene images are included in IC03-867, but not in IC03-860.

IC15 - Filtered examples in evaluation dataset. The IC15 dataset originally contains 2,077 examples for its evaluation set, however prior works [3, 2] have filtered it down to 1,811 examples and have not given unambiguous specifications between them for deciding on which example to discard. To resolve this ambiguity, we have contacted one of the authors, who shared the specific dataset used for the evaluation. This information is made available with the source code on Github. A few sample images that have been filtered out of the IC15 evaluation dataset is shown in Figure B.

IC03 and IC13 - Duplicated images between IC03 training dataset and IC13 evaluation dataset. Figure C shows two images from the subset given by the intersection between the IC03 training dataset and the IC13 evaluation dataset. In our investigation, a total of 34 duplicated scene images have been found, amounting to 215 duplicate word boxes, in total. Therefore, when one assesses the performance of a model on the IC13 evaluation data, he/she should be mindful of these overlapping data.

*Work performed as an intern in Clova AI Research.
†Corresponding author.

Model		Train Data	IIIT	SVT	IC03		IC13		IC15		SP	CT
			3000	647	860	867	857	1015	1811	2077	645	288
CRNN [9]	reported in paper	MJ	78.2	80.8	89.4	–	–	86.7	–	–	–	–
CRNN	our implementation	MJ	81.2	81.8	91.1	–	–	87.6	–	–	–	–
RARE [10]	reported in paper	MJ	81.9	81.9	90.1	–	88.6	–	–	–	71.8	59.2
RARE	our implementation	MJ	83.1	84.0	92.2	–	91.3	–	–	–	74.3	64.2
GRCNN [13]	reported in paper	MJ	80.8	81.5	91.2	–	–	–	–	–	–	–
GRCNN	our implementation	MJ	82.0	81.1	90.3	–	–	–	–	–	–	–
FAN (w/o Focus Net)[3]	reported in paper	MJ+ST	83.7	82.2	–	91.5	–	89.4	63.3	–	–	–
FAN (w/o Focus Net)	our implementation	MJ+ST	86.4	86.2	–	94.3	–	90.6	73.3	–	–	–

Table A: Sanity checking our experimental platform by reproducing the existing four STR models: CRNN [9], RARE [10], GRCNN [13] and FAN (w/o Focus Net [3]).

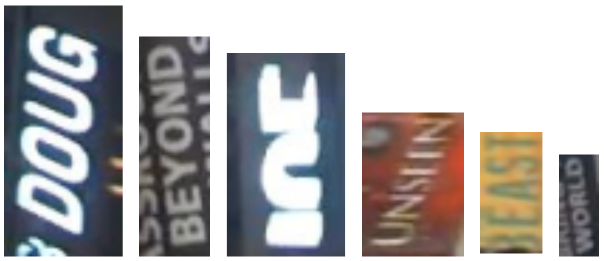

Figure B: Images that were filtered out of the IC15 evaluation dataset.

Figure C: Duplicated scene images. These are example images that have been found in both the IC03 training dataset and the IC13 evaluation dataset.

C. STR Framework - verification

To show the correctness of our implemented module for our framework, we reproduce the performances of existing models that can be re-built by our framework. Specifically, we compare the results of our implementation of CRNN, RARE, GRCNN, and FAN (w/o Focus Net) [9, 10, 13, 3] from those of publicly reported by the authors. We implemented each module as described in their original papers, and also we followed the training and evaluation pipelines of their original papers to train the individual models. Table A shows the results. Our implementation has overall similar performance with reported result in their paper, which verify the sanity of our implementations and experiments.

D. STR Framework - architectural details

In this appendix, we describe each module of our framework in terms of its concept and architectural specifications. We first introduce common notations used in this appendix and then explain the modules of each stage; Trans., Feat., Seq., and Pred.

Notations For a simple expression for a neural network architecture, we denote 'c', 'k', 's' and 'p' for the number of the output channel, the size of kernel, the stride, and the padding size respectively. BN, Pool, and FC denote the batch normalization layer, the max pooling layer, and the fully connected layer, respectively. In the case of convolution operation with the stride of 1 and the padding size of 1, 's' and 'p' are omitted for convenience.

D.1. Transformation stage

The module of this stage transforms the input image X into the normalized image $\tilde{X}$. We explained the concept of TPS [10, 8] in §3.1, but here we deliver its mathematical background and the implementation details.

TPS transformation: TPS generates a normalized image that shows a focused region of an input image. To build this pipeline, TPS consists of a sequence of processes; finding a text boundary, linking the location of the pixels in the boundary to those of the normalized image, and generating a normalized image by using the values of pixels and the linking information. Such processes are called as localization network, grid generator, and image sampler, respectively. Conceptually, TPS employs a smooth spline interpolation between a set of F fiducial points that represented a focused boundary of text in an image. Here, F indicates the constant number of fiducial points.

The localization network explicitly calculates x, y-coordinates of F fiducial points on an input image, X. The coordinates are denoted by $\mathbf{C} = [\mathbf{c}_1, \ldots, \mathbf{c}_F] \in \mathbb{R}^{2\times F}$, whose f-th column $\mathbf{c}_f = [x_f, y_f]^\intercal$ contains the coordinates of the f-th fiducial point. $\tilde{\mathbf{C}}$ represents pre-defined top and bottom locations on the normalized image, $\tilde{X}$.

The grid generator provides a mapping function from the identified regions by the localization network to the normalized image. The mapping function can be parameterized by a matrix $\mathbf{T} \in \mathbb{R}^{2\times(F+3)}$, which is computed by

$$\mathbf{T} = \left(\mathbf{\Delta}_{\tilde{\mathbf{C}}}^{-1} \begin{bmatrix} \mathbf{C}^\intercal \\ \mathbf{0}^{3\times2} \end{bmatrix} \right)^\intercal \tag{1}$$

where $\mathbf{\Delta}_{\mathbf{C}'} \in \mathbb{R}^{(F+3)\times(F+3)}$ is a matrix determined only by $\tilde{\mathbf{C}}$, thus also a constant:

$$\mathbf{\Delta}_{\tilde{\mathbf{C}}} = \begin{bmatrix} \mathbf{1}^{F\times1} & \tilde{\mathbf{C}}^\intercal & \mathbf{R} \\ \mathbf{0} & \mathbf{0} & \mathbf{1}^{1\times F} \\ \mathbf{0} & \mathbf{0} & \tilde{\mathbf{C}} \end{bmatrix} \tag{2}$$

where the element of i-th row and j-th column of $\mathbf{R}$ is $d_{ij} \ln d_{ij}^2$, d_{ij} is the euclidean distance between $\tilde{c}_i$ and $\tilde{c}_j$. The pixels of grid on the normalized image $\tilde{X}$ is denoted by $\tilde{P} = \{\tilde{p}_i\}_{i=1,\ldots,N}$, where $\tilde{p}_i = [\tilde{x}_i, \tilde{y}_i]^\intercal$ is the x,y-coordinates of the i-th pixel, N is the number of pixels. For every pixel $\tilde{p}_i$ on $\tilde{X}$, we find the corresponding point $p_i = [x_i, y_i]^\intercal$ on X, by applying the transformation:

$$p_i = \mathbf{T}\,[1, \tilde{x}_i, \tilde{y}_i, r_{i1}, \ldots, r_{iF}]^\intercal \tag{3}$$

$$r_{if} = d_{if}^2 \ln d_{if} \tag{4}$$

where d_{if} is the euclidean distance between pixel $\tilde{p}_i$ and the f-th base fiducial point $\tilde{c}_f$. By iterating Eq. 3 over all points in $\tilde{\mathcal{P}}$, we generate a grid $\mathcal{P} = \{p_i\}_{i=1,\ldots,N}$ on the input image X.

Finally, the image sampler produces the normalized image by interpolating the pixels in the input images which are determined by the grid generator.

TPS-Implementation: TPS requires the localization network calculating fiducial points of an input image. We designed the localization network by following most of the components of prior work [10], and added batch normalization layers and adaptive average pooling to stabilize the training of the network. Table B shows the details of our architecture. In our implementation, the localization network has 4 convolution layers, each followed by a batch normalization layer and 2 x 2 max-pooling layer. The filter size, padding size, and stride are 3, 1, 1 respectively, for all convolutional layers. Following the last convolutional layer is an adaptive average pooling layer (APool in Table B). After that, two fully connected layers are following: 512 to 256 and 256 to 2F. Final output is 2F dimensional vector which corresponds to the value of x, y-coordinates of F fiducial points on input image. Activation functions for all layers are the ReLU.

Layers	**Configurations**		**Output**
Input	grayscale image		100×32
Conv1	c: 64	k: 3×3	100×32
BN1	-		100×32
Pool1	k: 2×2	s: 2×2	50×16
Conv2	c: 128	k: 3×3	50×16
BN2	-		50×16
Pool2	k: 2×2	s: 2×2	25×8
Conv3	c: 256	k: 3×3	25×8
BN3	-		25×8
Pool3	k: 2×2	s: 2×2	12×4
Conv4	c: 512	k: 3×3	12×4
BN4	-		12×4
APool	$512 \times 12 \times 4 \rightarrow 512 \times 1$		512
FC1	$512 \rightarrow 256$		256
FC2	$256 \rightarrow 2F$		$2F$

Table B: Architecture of the localization network in TPS. The localization network extracts the location of the text line, that is, the x- and y-coordinates of the fiducial points F within the input image.

Layers	**Configurations**		**Output**
Input	grayscale image		100×32
Conv1	c: 64	k: 3×3	100×32
Pool1	k: 2×2	s: 2×2	50×16
Conv2	c: 128	k: 3×3	50×16
Pool2	k: 2×2	s: 2×2	25×8
Conv3	c: 256	k: 3×3	25×8
Conv4	c: 256	k: 3×3	25×8
Pool3	k: 1×2	s: 1×2	25×4
Conv5	c: 512	k: 3×3	25×4
BN1	-		25×4
Conv6	c: 512	k: 3×3	25×4
BN2	-		25×4
Pool4	k: 1×2	s: 1×2	25×2
Conv7	c: 512 s: 1×1	k: 2×2 p: 0×0	24×1

Table C: Architecture of VGG.

D.2. Feature extraction stage

In this stage, a CNN abstract an input image (i.e., X or $\tilde{X}$) and outputs a feature map $V = \{v_i\}, i = 1, \ldots, I$ (I is the number of columns in the feature map).

VGG: we implemented VGG [11] which is used in CRNN [9] and RARE [10]. We summarized the architecture in Table C. The output of VGG is 512 channels $\times$ 24 columns.

Recurrently applied CNN (RCNN): As a RCNN module, we implemented a Gated RCNN (GRCNN) [13] which is a variant of RCNN that can be applied recursively with a gating mechanism. The architectural details of the module are shown in Table D. The output of RCNN is 512 channels $\times$ 26 columns.

Layers	Configurations	Output
Input	grayscale image	100×32
Conv1	c: 64 k: 3×3	100×32
Pool1	k: 2×2 s: 2×2	50×16
GRCL1	$\begin{bmatrix} \text{c}:64, \text{k}:3 \times 3 \end{bmatrix} \times 5$	50×16
Pool2	k: 2×2 s: 2×2	25×8
GRCL2	$\begin{bmatrix} \text{c}:128, \text{k}:3 \times 3 \end{bmatrix} \times 5$	25×8
Pool3	k: 2×2 s: 1×2 p: 1×0	26×4
GRCL3	$\begin{bmatrix} \text{c}:256, \text{k}:3 \times 3 \end{bmatrix} \times 5$	26×4
Pool4	k: 2×2 s: 1×2 p: 1×0	27×2
Conv2	c: 512 k: 3×3 s: 1×1 p: 0×0	26×1

Table D: Architecture of RCNN.

Layers	Configurations	Output
Input	grayscale image	100×32
Conv1	c: 32 k: 3×3	100×32
Conv2	c: 64 k: 3×3	100×32
Pool1	k: 2×2 s: 2×2	50×16
Block1	$\begin{bmatrix} \text{c}:128, \text{k}:3 \times 3 \\ \text{c}:128, \text{k}:3 \times 3 \end{bmatrix} \times 2$	50×16
Conv3	c: 128 k: 3×3	50×16
Pool2	k: 2×2 s: 2×2	25×8
Block2	$\begin{bmatrix} \text{c}:256, \text{k}:3 \times 3 \\ \text{c}:256, \text{k}:3 \times 3 \end{bmatrix} \times 2$	25×8
Conv4	c: 256 k: 3×3	25×8
Pool3	k: 2×2 s: 1×2 p: 1×0	26×4
Block3	$\begin{bmatrix} \text{c}:512, \text{k}:3 \times 3 \\ \text{c}:256, \text{k}:3 \times 3 \end{bmatrix} \times 5$	26×4
Conv5	c: 512 k: 3×3	26×4
Block4	$\begin{bmatrix} \text{c}:512, \text{k}:3 \times 3 \\ \text{c}:512, \text{k}:3 \times 3 \end{bmatrix} \times 3$	26×4
Conv6	c: 512 k: 2×2 s: 1×2 p: 1×0	27×2
Conv7	c: 512 k: 2×2 s: 1×1 p: 0×0	26×1

Table E: Architecture of ResNet.

Residual Network (ResNet): As a ResNet [7] module, we implemented the same network which is used in FAN [3]. It has 29 trainable layers in total. The details of the network is shown in Table E. The output of ResNet is 512 channels $\times$ 26 columns.

D.3. Sequence modeling stage

Some previous works used Bidirectional LSTM **(BiLSTM)** to make a contextual sequence $H = \text{Seq.}(\text{V})$ after the Feat. stage [9].
BiLSTM: We implemented 2-layers BiLSTM [6] which is used in CRNN [9]. In the followings, we explain a BiLSTM layer used in our framework: A l^{th} BiLSTM layer identifies two hidden states, $h_i^{(t),\text{f}}$ and $h_i^{(t),\text{b}}$ $\forall t$, calculated through time sequence and its reverse. Following [9], we additionally applied a FC layer between BiLSTM layers to determine one hidden state, $\hat{h}_t^{(l)}$, by using the two identified hidden states, $h_t^{(l),\text{f}}$ and $h_t^{(l),\text{b}}$. The dimensions of all hidden states including the FC layer was set as 256.
None indicates not to use any Seq. modules upon the output of the Feat. modules, that is, $H = V$.

D.4. Prediction stage

A prediction module produces the final prediction output from the input H, (i.e., $Y = y_1, y_2, \dots$), which is a sequence of characters. We implemented two modules: Connectionist Temporal Classification (CTC) [5] based and Attention mechanism (Attn) based Pred. module. In our experiments, we make the character label set C which include 36 alphanumeric characters. For the CTC, additional *blank* token is added to the label set due to the characteristics of the CTC. For the Attn, additional end of sentence (EOS) token is added to the label set due to the characteristics of the Attn. That is, the number of character set C is 37.
Connectionist Temporal Classification (CTC): CTC takes a sequence $\mathbf{H} = h_1, \dots, h_T$, where T is the sequence length, and outputs the probability of π, which is defined as

$$p(\pi|H) = \prod_{t=1}^{T} y_{\pi_t}^t \tag{5}$$

where $y_{\pi_t}^t$ is the probability of generating character π_t at each time step t. After that, the mapping function M which maps π to Y by removing repeated characters and blanks. For instance, M maps "`aaa--b-b-c-ccc-c--`" onto "`abbccc`", where '-' is *blank* token. The conditional probability is defined as the sum of probabilities of all π that are mapped by M onto Y, which is

$$p(Y|H) = \sum_{\pi : M(\pi) = Y} p(\pi|H) \tag{6}$$

At testing phase, the predicted label sequence is calculated by taking the highest probability character π_t at each time step t, and map the π onto Y:

$$Y^* \approx M(\arg\max_{\pi} p(\pi|H)) \tag{7}$$

Attention mechanism (Attn): We implemented one layer LSTM attention decoder [1] which is used in FAN, AON, and EP [3, 4, 2]. The details are as follows: at t-step, the decoder predicts an output y_t as

$$y_t = \mathrm{softmax}(\mathrm{W_o s_t + b_o}) \tag{8}$$

where W_0 and b_0 are trainable parameters. s_t is the decoder LSTM hidden state at time t as

$$s_t = \mathrm{LSTM}(\mathrm{y_{t-1}, c_t, s_{t-1}}) \tag{9}$$

and c_t is a context vector, which is computed as the weighted sum of $H = h_1, ...h_I$ from the former stage as

$$c_t = \sum_{i=1}^{I} \alpha_{ti} h_i \tag{10}$$

where α_{ti} is called attention weight and computed by

$$\alpha_{ti} = \frac{exp(e_{ti})}{\sum_{k=1}^{I} exp(e_{tk})} \tag{11}$$

where

$$e_{ti} = v^{\intercal} \tanh(W s_{t-1} + V h_i + b) \tag{12}$$

and v, W, V and b are trainable parameters. The dimension of LSTM hidden state was set as 256.

D.5. Objective function

Denote the training dataset by $TD = \{X_i, Y_i\}$, where X_i is the training image and Y_i is the word label. The training conducted by minimizing the objective function that negative log-likelihood of the conditional probability of word label.

$$O = - \sum_{X_i, Y_i \in TD} \log p(Y_i | X_i) \tag{13}$$

This function calculates a cost from an image and its word label, and the modules in the framework are trained end-to-end manner.

E. STR Framework - full experimental results

We report the full results of our experiments in Table F. In addition, Figure D–G show two types of trade-off plots of 24 combinations in respect of accuracy versus time and accuracy versus memory. In Figure D–G, all the combination are color-coded in terms of each module, which helps to grasp the effectiveness of each module.

#	Trans.	Feat.	Seq.	Pred.	IIIT	SVT	IC03		IC13		IC15		SP	CT	Acc.	Time	params
					3000	647	860	867	857	1015	1811	2077	645	288	Total	ms	$\times 10^6$
1	None	VGG	None	CTC	76.2	73.8	86.7	86.0	84.8	81.9	56.6	52.4	56.6	49.9	69.5	1.3	5.6
2				Attn	80.1	78.4	91.0	90.5	88.5	86.3	63.0	58.3	66.0	56.1	74.6	19.0	6.6
3[1]			BiLSTM	CTC	82.9	81.6	93.1	92.6	91.1	89.2	69.4	64.2	70.0	65.5	78.4	4.4	8.3
4				Attn	84.3	83.8	93.7	93.1	91.9	90.0	70.8	65.4	71.9	66.8	79.7	21.2	9.2
5		RCNN	None	CTC	80.9	78.5	90.5	89.8	88.4	85.9	65.1	60.5	65.8	60.3	75.4	7.7	1.9
6[2]				Attn	83.4	82.4	92.2	92.0	90.2	88.1	68.9	63.6	72.1	64.9	78.5	24.1	2.9
7[3]			BiLSTM	CTC	84.2	83.7	93.5	93.0	90.9	88.8	71.4	65.8	73.6	68.1	79.8	10.7	4.6
8				Attn	85.7	84.8	93.9	93.4	91.6	89.6	72.7	67.1	75.0	69.2	81.0	27.4	5.5
9[4]		ResNet	None	CTC	84.3	84.7	93.4	92.9	90.9	89.0	71.2	66.0	73.8	69.2	80.0	4.7	44.3
10				Attn	86.1	85.7	94.0	93.6	91.9	90.1	73.5	68.0	74.5	72.2	81.5	22.2	45.3
11			BiLSTM	CTC	86.2	86.0	94.4	94.1	92.6	90.8	73.6	68.0	76.0	72.2	81.9	7.8	47.0
12				Attn	86.6	86.2	94.1	93.7	92.8	91.0	75.6	69.9	76.4	72.6	82.5	25.0	47.9
13	TPS	VGG	None	CTC	80.0	78.0	90.1	89.7	88.7	87.5	65.1	60.6	65.5	57.0	75.1	4.8	7.3
14				Attn	82.9	82.3	92.0	91.7	90.5	89.2	69.4	64.2	73.0	62.2	78.5	21.0	8.3
15			BiLSTM	CTC	84.6	83.8	93.3	92.9	91.2	89.4	72.4	66.8	74.0	66.8	80.2	7.6	10.0
16[5]				Attn	86.2	85.8	93.9	93.7	92.6	91.1	74.5	68.9	76.2	70.4	82.0	23.6	10.8
17		RCNN	None	CTC	82.8	81.7	92.0	91.6	89.5	88.4	69.8	64.6	71.3	61.2	78.3	10.9	3.6
18				Attn	85.1	84.0	93.1	93.1	91.5	90.2	72.4	66.8	75.6	64.9	80.6	26.4	4.6
19			BiLSTM	CTC	85.1	84.3	93.5	93.1	91.4	89.6	73.4	67.7	74.4	69.1	80.8	14.1	6.3
20				Attn	86.3	85.7	94.0	94.0	92.8	91.1	75.0	69.2	77.7	70.1	82.3	30.1	7.2
21		ResNet	None	CTC	85.0	85.7	94.0	93.6	92.5	90.8	74.6	68.8	75.2	71.0	81.5	8.3	46.0
22				Attn	87.1	87.1	94.3	93.9	93.2	91.8	76.5	70.6	78.9	73.2	83.3	25.6	47.0
23[6]			BiLSTM	CTC	87.0	86.9	94.4	94.0	92.8	91.5	76.1	70.3	77.5	71.7	82.9	10.9	48.7
24[7]				Attn	**87.9**	**87.5**	**94.9**	**94.4**	**93.6**	**92.3**	**77.6**	**71.8**	**79.2**	**74.0**	**84.0**	27.6	49.6

[1] CRNN. [2] R2AM. [3] GRCNN. [4] Rosetta. [5] RARE. [6] STAR-Net. [7] our best model.

Table F: The full experimental results for all 24 STR combinations. Top accuracy for each benchmark is shown in **bold**. For each STR combination, we have run five trials with different initialization random seeds and have averaged their accuracies.

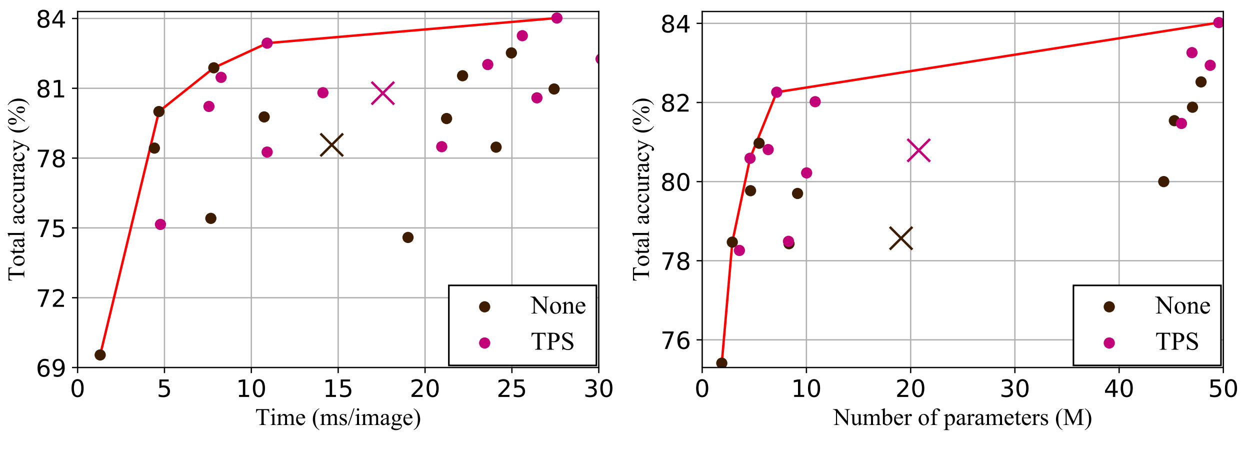

Figure D: Color-coded version of Figure 4 in §4.3, according to the transformation stage. Each circle represents the performance for each different combination of STR modules, while the each cross represents the average performance among STR combinations without TPS (**black**) or with TPS (**magenta**). Choosing to add TPS or not does not seem to give a noticeable advantage in performance when looking at the performances of each STR combination. However, the average accuracy does increase when using TPS compared to when it is unused, at the cost of longer inference times and a slight increase in the number of parameters.

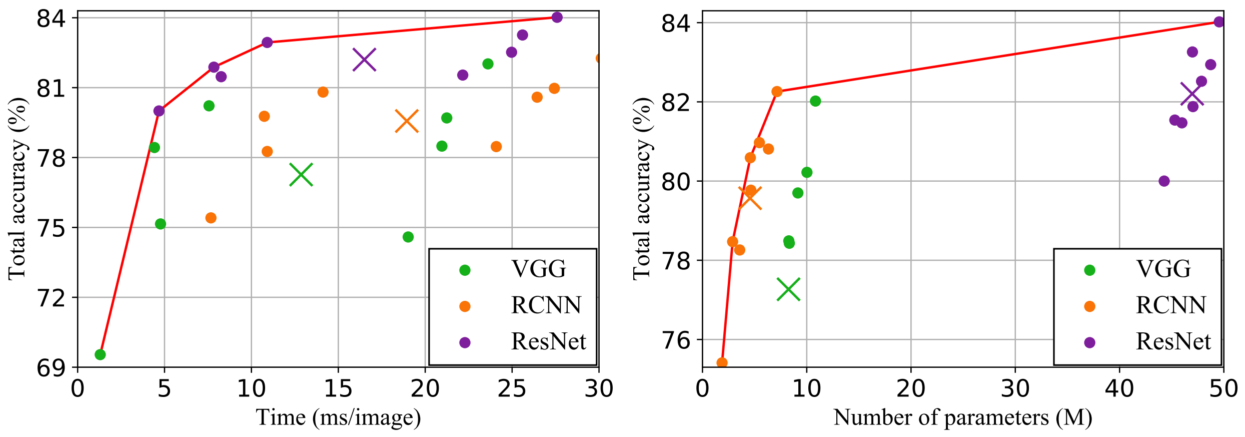

Figure E: Color-coded version of Figure 4 in §4.3, according to the feature extraction stage. Each circle represents the performance for each different combination of STR modules, while the each cross represents the average performance among STR combinations using VGG (**green**), RCNN (**orange**), or ResNet (**violet**). VGG gives the lowest accuracy on average for the lowest amount of inference time required, while RCNN achieves higher accuracy over VGG by taking the longest time for inferencing and the lowest memory usage out of the three. ResNet, exhibits the highest accuracy at the cost of using significantly more memory than the other modules. In summary, if the system to be implemented is constrained by memory, RCNN offers the best trade-off, and if the system requires high accuracy, ResNet should be used. The time difference between the three modules are so small in practice that it should be considered only in the most extreme of circumstances.

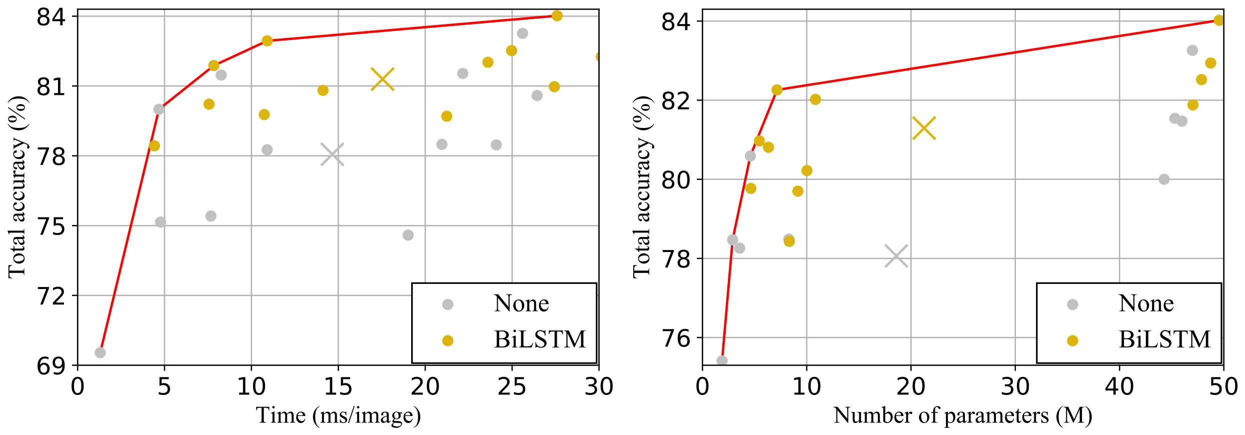

Figure F: Color-coded version of Figure 4 in §4.3, according to the sequence modeling stage. Each circle represents the performance for each different combination of STR modules, while the each cross represents the average performance among STR combinations without BiLSTM (**gray**) or with BiLSTM (**gold**). Using BiLSTM seems to have a similar effect to using TPS, and vice versa, except BiLSTM gives a larger accuracy boost on average with similar inference time or parameter size concessions compared to TPS.

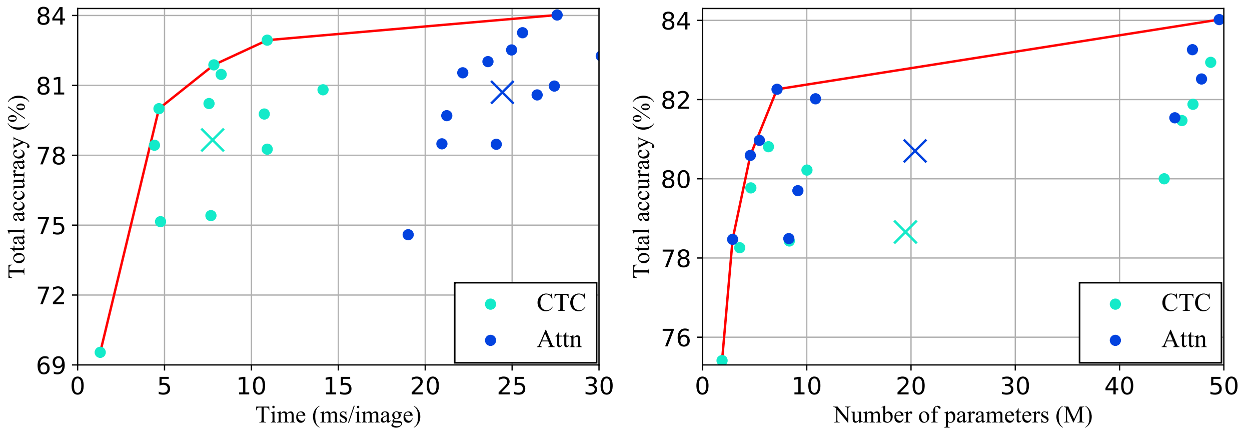

Figure G: Color-coded version of Figure 4 in §4.3, according to prediction stage. Each circle represents the performance for each different combination of STR modules, while the each cross represents the average performance among STR combinations with CTC (**cyan**) or with Attn (**blue**). The choice between CTC and Attn gives the largest and clearest inference time increase for each percentage of accuracy gained. The same cannot be said about the increase in the number of parameters with respect to accuracy increase, as Attn gains about 2 percentage points with minimal memory usage increase.

F. Additional Experiments

F.1. Fine-tuning on real datasets

We have fine-tuned our best model on the union of training sets IIIT, SVT, IC13, and IC15 (in-distribution), the held-out subsets of evaluation datasets of real scene text images. Other evaluation datasets, IC03, SP, and CT (out-distribution), do not have held-out subset for training; SP and CT have not training sets and some training images of IC03 have been found in IC13 evaluation dataset, as mentioned in §4.1, thus it is not appropriate to fine-tuning on IC03 training set.

Our model has been fine-tuned for 10 epochs. The table G shows the results. By fine-tuning on the real data, the accuracy on in-distribution subset (the union of evaluation datasets IIIT, SVT, IC13, and IC15) and on all benchmark data have improved by 2.2 pp and 1.5 pp, respectively. Meanwhile, the fine-tuned performance on the out-distribution subset (the union of evaluation datasets IC03, SP, and CT) has decreased by 1.3 pp. We conclude that fine-tuning over real data is effective when the real-data is close to the test-time distribution. Otherwise, fine-tuning over real data may do more harm than good.

	Accuracy (%)		
	in-distribution	out-distribution	all
Original	83.9	85.1	84.1
Fine-tuned	86.1(**+2.2**)	83.8(*-1.3*)	85.6(**+1.5**)

Table G: Accuracy change with fine-tuning on real datasets.

F.2. Accuracy with varying training dataset size

We have evaluated the accuracy of all 24 STR models against varying training dataset size. Training dataset consists of MJSynth 8.9 M and SynthText 5.5 M (14.4 M in total), same setting as in §4.1. We report the full results of varying training dataset size in Table H. In addition, Figure H–K show averaged accuracy plots. Each plot is color-coded in terms of each module, which helps to grasp the tendency of each module.

From the Table H and the plot of average of all models in Figure H–K, we observe that the average of all 24 models tends to have higher accuracy with more data.

In Figure H, we observe that the curves of without TPS do not get saturated at 100% training data size; more training data are certainly likely to improve them. The curves of TPS show saturated performances at 80% training data. We conjecture this is because TPS usually normalizes the input images and the last 20% of training dataset would be normalized by TPS, rather than improve accuracy. Thus other kinds of datasets, which will not simply be normalized by TPS trained with 80% training dataset, would be needed to better accuracy.

#	Trans.	Feat.	Seq.	Pred.	Training dataset size (%)				
					20	40	60	80	100
1	None	VGG	None	CTC	66.1	68.1	69.2	68.8	68.8
2				Attn	71.5	73.0	74.2	74.7	74.6
3[1]			BiLSTM	CTC	75.8	77.6	77.7	77.9	78.6
4				Attn	75.6	77.9	79.3	79.3	79.7
5		RCNN	None	CTC	69.7	71.2	72.0	75.5	74.7
6[2]				Attn	76.2	77.5	77.3	78.1	78.2
7[3]			BiLSTM	CTC	77.1	78.8	79.6	80.0	79.7
8				Attn	77.9	79.6	80.3	80.5	81.3
9[4]		ResNet	None	CTC	75.9	77.8	78.8	78.9	80.7
10				Attn	78.0	80.3	80.5	81.6	81.5
11			BiLSTM	CTC	78.9	80.7	80.8	81.3	81.7
12				Attn	79.2	81.0	81.9	82.3	82.6
13	TPS	VGG	None	CTC	73.8	74.9	75.4	75.5	75.2
14				Attn	75.9	78.3	78.8	78.5	78.7
15			BiLSTM	CTC	77.9	79.2	79.9	79.5	80.1
16[5]				Attn	79.6	81.1	81.7	82.0	81.9
17		RCNN	None	CTC	77.8	78.5	76.8	78.6	78.1
18				Attn	79.2	79.8	80.5	80.4	80.7
19			BiLSTM	CTC	78.7	80.7	81.2	80.8	80.9
20				Attn	80.4	81.8	82.2	82.5	83.1
21		ResNet	None	CTC	80.7	80.7	80.8	81.7	81.9
22				Attn	80.7	81.6	82.6	83.0	83.3
23[6]			BiLSTM	CTC	80.7	81.8	82.6	83.0	83.2
24[7]				Attn	81.3	82.7	83.2	84.0	84.1

[1] CRNN. [2] R2AM. [3] GRCNN. [4] Rosetta. [5] RARE. [6] STAR-Net. [7] our best model.

Table H: The full experimental results of varying training dataset size for all 24 STR combinations. Each value represent Total accuracy (%), as mentioned in §4.1. Note that, for each STR combination, we have run only one trial and thus the result could be slightly different from the Table F.

In Figure I, we observe that the curves of ResNet do not get saturated at 100% training data size. The averages of VGG and RCNN, on the other hand, show saturated performances at 60% and 80% training data, respectively. We conjecture this is because VGG and RCNN have lower capacity than ResNet and they have already reached their performance limits at the current amount of training data.

In Figure J, we observe that the curves of BiLSTM do not get saturated at 100% training data size. The curves of without BiLSTM show saturated performances at 80% training data. We conjecture this is because using BiLSTM has higher capacity than without BiLSTM and thus using BiLSTM still has room for improving accuracy with more training data.

In Figure K, we observe that the curves of Attn do not get saturated at 100% training data size. The curves of CTC show saturated performances at 80% training data. Again, we conjecture this is because using Attn has higher capacity than CTC and thus using Attn still has room for improving accuracy with more training data.

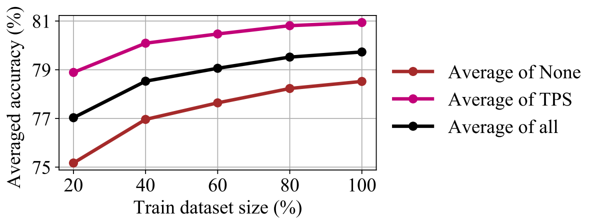

Figure H: Averaged accuracy with varying training dataset size. Each plot represents the average performance among STR combinations without TPS (**brown**), with TPS (**magenta**), or all models (**black**)

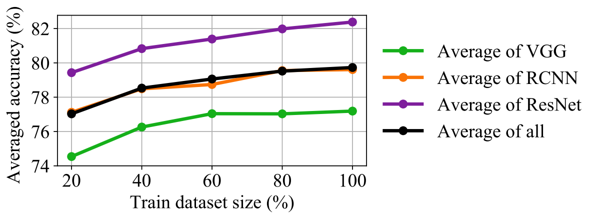

Figure I: Averaged accuracy with varying training dataset size. Each plot represents the average performance among STR combinations using VGG (**green**), RCNN (**orange**), ResNet (**violet**), or all models (**black**)

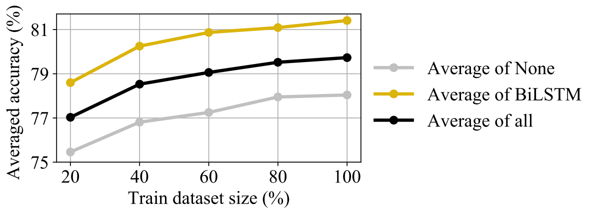

Figure J: Averaged accuracy with varying training dataset size. Each plot represents the average performance among STR combinations without BiLSTM (**gray**), with BiLSTM (**gold**), or all models (**black**)

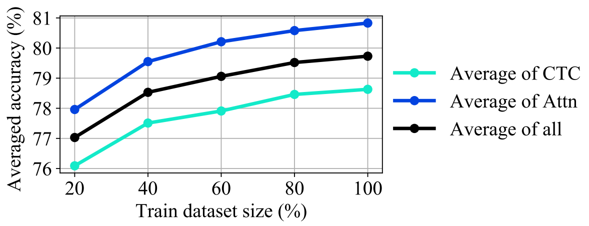

Figure K: Averaged accuracy with varying training dataset size. Each plot represents the average performance among STR combinations with CTC (**cyan**), with Attn (**blue**), or all models (**black**)

F.3. Evaluation on COCO-Text dataset

We have evaluated the models on COCO-Text dataset [12], another good benchmark derived from MS COCO containing complex and low-resolution scene images. COCO-Text contains many special characters, heavy noises, and occlusions; it is generally considered more challenging than the seven benchmarks considered so far. Figure L shows the accuracy-time and accuracy-space trade-off plots for 24 STR methods on COCO-Text. Except that the overall accuracy is lower, the relative orders amongst methods are largely preserved compared to Figure 4. Fine-tuning models with COCO-Text training set has improved the averaged accuracy (24 models) from 42.4% to 58.2%, a relatively big jump that is attributable to the unusual data distribution for COCO-Text. Evaluation and analysis over COCO-Text are beneficial, especially to address remaining corner cases for STR.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[2] Fan Bai, Zhanzhan Cheng, Yi Niu, Shiliang Pu, and Shuigeng Zhou. Edit probability for scene text recognition. In *CVPR*, 2018.

[3] Zhanzhan Cheng, Fan Bai, Yunlu Xu, Gang Zheng, Shiliang Pu, and Shuigeng Zhou. Focusing attention: Towards accurate text recognition in natural images. In *ICCV*, pages 5086–5094, 2017.

[4] Zhanzhan Cheng, Yangliu Xu, Fan Bai, Yi Niu, Shiliang Pu, and Shuigeng Zhou. Aon: Towards arbitrarily-oriented text recognition. In *CVPR*, pages 5571–5579, 2018.

[5] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *ICML*, pages 369–376, 2006.

[6] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. In *TPAMI*, volume 31, pages 855–868. IEEE Computer Society, 2009.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[8] Wei Liu, Chaofeng Chen, Kwan-Yee K Wong, Zhizhong Su, and Junyu Han. Star-net: A spatial attention residue network for scene text recognition. In *BMVC*, volume 2, 2016.

[9] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. In *TPAMI*, volume 39, pages 2298–2304. IEEE, 2017.

[10] Baoguang Shi, Xinggang Wang, Pengyuan Lyu, Cong Yao, and Xiang Bai. Robust scene text recognition with automatic rectification. In *CVPR*, pages 4168–4176, 2016.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[12] Andreas Veit, Tomas Matera, Lukas Neumann, Jiri Matas, and Serge Belongie. Coco-text: Dataset and benchmark

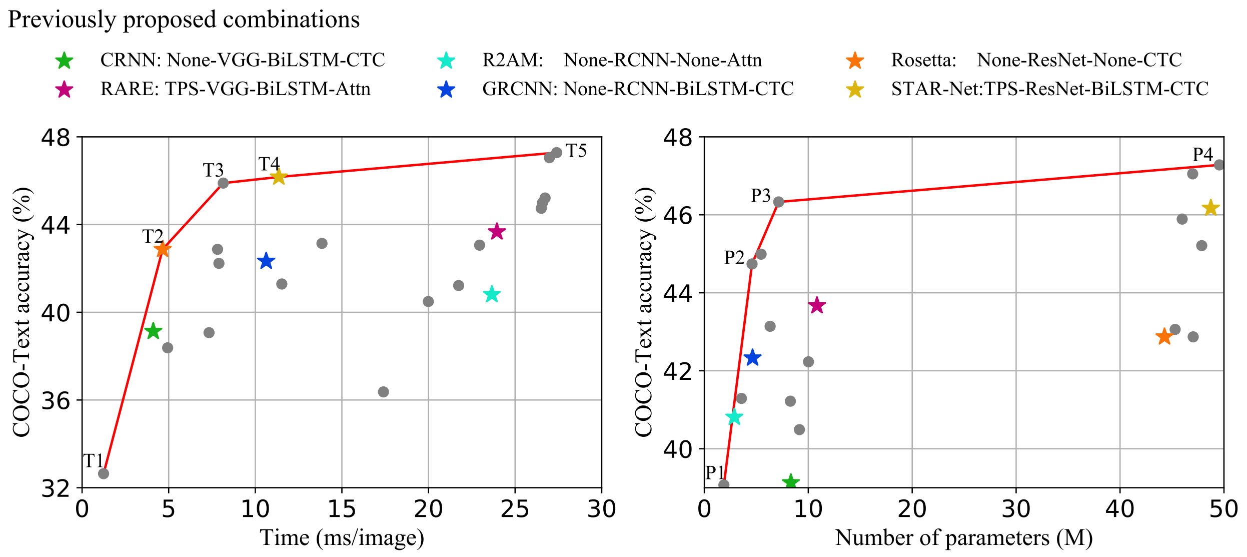

Figure L: COCO-Text accuracy version of Figure 4 in §4.3.

for text detection and recognition in natural images. In *arXiv:1601.07140*, 2016.

[13] Jianfeng Wang and Xiaolin Hu. Gated recurrent convolution neural network for ocr. In *NIPS*, pages 334–343, 2017.