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

Abstract

Many new proposals for scene text recognition (STR) models have been introduced in recent years. While each claim to have pushed the boundary of the technology, a holistic and fair comparison has been largely missing in the field due to the inconsistent choices of training and evaluation datasets. This paper addresses this difficulty with three major contributions. First, we examine the inconsistencies of training and evaluation datasets, and the performance gap results from inconsistencies. Second, we introduce a unified four-stage STR framework that most existing STR models fit into. Using this framework allows for the extensive evaluation of previously proposed STR modules and the discovery of previously unexplored module combinations. Third, we analyze the module-wise contributions to performance in terms of accuracy, speed, and memory demand, under one consistent set of training and evaluation datasets. Such analyses clean up the hindrance on the current comparisons to understand the performance gain of the existing modules. Our code is publicly available[1].

1. Introduction

Reading text in natural scenes, referred to as scene text recognition (STR), has been an important task in a wide range of industrial applications. The maturity of Optical Character Recognition (OCR) systems has led to its successful application on cleaned documents, but most traditional OCR methods have failed to be as effective on STR tasks due to the diverse text appearances that occur in the real world and the imperfect conditions in which these scenes are captured.

To address these challenges, prior works [23, 24, 15, 17, 26, 28, 4, 16, 5, 2, 3, 18] have proposed multi-stage pipelines, where each stage is a deep neural network addressing a specific challenge. For example, Shi *et al*. [23] have suggested using a recurrent neural network to address the varying number of characters in a given input, and a connectionist temporal classification loss [6] to identify the number of characters. Shi *et al*. [24] have proposed a transformation module that normalizes the input into a straight text image to reduce the representational burden for downstream modules to handle curved texts.

However, it is hard to assess whether and how a newly proposed module improves upon the current art, as some papers have come up with different evaluation and testing environments, making it difficult to compare reported numbers at face value (Table 1). We observed that 1) the training datasets and 2) the evaluation datasets deviate amongst various methods, as well. For example, different works use a different subset of the IC13 dataset as part of their evaluation set, which may cause a performance disparity of more than 15%. This kind of discrepancy hinders the fair comparison of performance between different models.

Our paper addresses these types of issues with the following main contributions. First, we analyze all training and evaluation datasets commonly used in STR papers. Our analysis reveals the inconsistency of using the STR datasets and its causes. For instance, we found 7 missing examples in IC03 dataset and 158 missing examples in IC13 dataset as well. We investigate several previous works on the STR datasets and show that the inconsistency causes incomparable results as shown in Table 1. Second, we introduce a unifying framework for STR that provides a common perspective for existing methods. Specifically, we divide the STR model into four different consecutive stages of operations: transformation (Trans.), feature extraction (Feat.), sequence modeling (Seq.), and prediction (Pred.). The framework provides not only existing methods but their possible variants toward an extensive analysis of module-wise con-

*Work performed as an intern in Clova AI Research.

†Corresponding author.

[1] https://github.com/clovaai/deep-text-recognition-benchmark

	Model	Year	Train data	IIIT	SVT	IC03		IC13		IC15		SP	CT	Time	params
				3000	647	860	867	857	1015	1811	2077	645	288	ms/image	$\times 10^6$
Reported results	CRNN [23]	2015	MJ	78.2	80.8	89.4	–	–	86.7	–	–	–	–	160	8.3
	RARE [24]	2016	MJ	81.9	81.9	90.1	–	88.6	–	–	–	71.8	59.2	<2	–
	R2AM [15]	2016	MJ	78.4	80.7	88.7	–	–	90.0	–	–	–	–	2.2	–
	STAR-Net [17]	2016	MJ+PRI	83.3	83.6	89.9	–	–	89.1	–	–	73.5	–	–	–
	GRCNN [26]	2017	MJ	80.8	81.5	91.2	–	–	–	–	–	–	–	–	–
	ATR [28]	2017	PRI+C	–	–	–	–	–	–	–	–	**75.8**	69.3	–	–
	FAN [4]	2017	MJ+ST+C	87.4	85.9	–	94.2	–	93.3	70.6	–	–	–	–	–
	Char-Net [16]	2018	MJ	83.6	84.4	**91.5**	–	90.8	–	–	60.0	73.5	–	–	–
	AON [5]	2018	MJ+ST	87.0	82.8	–	91.5	–	–	–	**68.2**	73.0	**76.8**	–	–
	EP [2]	2018	MJ+ST	88.3	**87.5**	–	94.6	–	**94.4**	**73.9**	–	–	–	–	–
	Rosetta [3]	2018	PRI	–	–	–	–	–	–	–	–	–	–	–	–
	SSFL [18]	2018	MJ	**89.4**	87.1	–	**94.7**	**94.0**	–	–	–	73.9	62.5	–	–
Our experiment	CRNN [23]	2015	MJ+ST	82.9	81.6	93.1	92.6	91.1	89.2	69.4	64.2	70.0	65.5	4.4	8.3
	RARE [24]	2016	MJ+ST	86.2	85.8	93.9	93.7	92.6	91.1	74.5	68.9	76.2	70.4	23.6	10.8
	R2AM [15]	2016	MJ+ST	83.4	82.4	92.2	92.0	90.2	88.1	68.9	63.6	72.1	64.9	24.1	2.9
	STAR-Net [17]	2016	MJ+ST	87.0	86.9	94.4	94.0	92.8	91.5	76.1	70.3	77.5	71.7	10.9	48.7
	GRCNN [26]	2017	MJ+ST	84.2	83.7	93.5	93.0	90.9	88.8	71.4	65.8	73.6	68.1	10.7	4.6
	Rosetta [3]	2018	MJ+ST	84.3	84.7	93.4	92.9	90.9	89.0	71.2	66.0	73.8	69.2	4.7	44.3
	Our best model		MJ+ST	**87.9**	**87.5**	**94.9**	**94.4**	**93.6**	**92.3**	**77.6**	**71.8**	**79.2**	**74.0**	27.6	49.6

Table 1: Performance of existing STR models with their **inconsistent** training and evaluation settings. This inconsistency hinders the fair comparison among those methods. We present the results reported by the original papers and also show our re-implemented results under unified and consistent setting. At the last row, we also show the best model we have found, which shows competitive performance to state-of-the-art methods. MJ, ST, C, and, PRI denote MJSynth [10], SynthText [7], Character-labeled [4, 28], and private data [3], respectively. Top accuracy for each benchmark is shown in **bold**.

tribution. Finally, we study the module-wise contributions in terms of accuracy, speed, and memory demand, under a unified experimental setting. With this study, we assess the contribution of individual modules more rigorously and propose previously overlooked module combinations that improves over the state of the art. Furthermore, we analyzed failure cases on the benchmark dataset to identify remaining challenges in STR.

2. Dataset Matters in STR

In this section, we examine the different training and evaluation datasets used by prior works, and then their discrepancies are addressed. Through this analysis, we highlight how each of the works differs in constructing and using their datasets, and investigate the bias caused by the inconsistency when comparing performance between different works (Table 1). The performance gaps due to dataset inconsistencies are measured through experiments and discussed in §4.

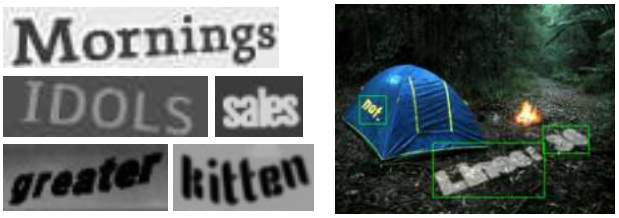
2.1. Synthetic datasets for training

When training a STR model, labeling scene text images is costly, and thus it is difficult to obtain enough labeled data for. Alternatively using real data, most STR models have used synthetic datasets for training. We first introduce two most popular synthetic datasets used in recent STR papers:

- **MJSynth (MJ)** [10] is a synthetic dataset designed for STR, containing 8.9 M word box images. The word box generation process is as follows: 1) font rendering, 2) border and shadow rendering, 3) background coloring, 4) composition of font, border, and background, 5) applying projective distortions, 6) blending with real-world images, and 7) adding noise. Figure 1a shows some examples of MJSynth,

- **SynthText (ST)** [7] is another synthetically generated dataset and was originally designed for scene text detection. An example of how the words are rendered onto scene images is shown in Figure 1b. Even though SynthText was designed for scene text detection task, it has been also used for STR by cropping word boxes. SynthText has 5.5 M training data once the word boxes are cropped and filtered for non-alphanumeric characters.

Note that prior works have used diverse combinations of MJ, ST, and or other sources (Table 1). These **inconsistencies** call into question whether the improvements are due to the contribution of the proposed module or to that of a better or larger training data. Our experiment in §4.2 describes the influence of the training datasets to the final performance on the benchmarks. We further suggest that future STR researches clearly indicate the training datasets used and com-



(a) MJSynth word boxes (b) SynthText scene image

Figure 1: Samples of MJSynth and SynthText used as training data.

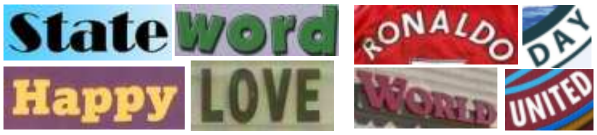
pare models using the same training set.

2.2. Real-world datasets for evaluation

Seven real-world STR datasets have been widely used for evaluating a trained STR model. For some benchmark dataset, **different subsets** of the dataset may have been used in each prior work for evaluation (Table 1). These difference in subsets result in **inconsistent** comparison.

We introduce the datasets by categorizing them into regular and irreguglar datsets. The benchmark datasets are given the distinction of being "regular" or "irregular" datasets [24, 28, 5], according to the difficulty and geometric layout of the texts. First, **regular datasets** contain text images with horizontally laid out characters that have even spacings between them. These represent relatively easy cases for STR:

- **IIIT5K-Words (IIIT)** [20] is the dataset crawled from Google image searches, with query words that are likely to return text images, such as "billboards", "signboard", "house numbers", "house name plates", and "movie posters". IIIT consists of 2,000 images for training and 3,000 images for evaluation,
- **Street View Text (SVT)** [27] contains outdoor street images collected from Google Street View. Some of these images are noisy, blurry, or of low-resolution. SVT consists of 257 images for training and 647 images for evaluation,
- **ICDAR2003 (IC03)** [19] was created for the ICDAR 2003 Robust Reading competition for reading camera-captured scene texts. It contains 1,156 images for training and 1,110 images for evaluation. Ignoring all words that are either too short (less than 3 characters) or ones that contain non-alphanumeric characters reduces 1,110 images to 867. However, researchers have used two different versions of the dataset for evaluation: versions with 860 and 867 images. The 860-image dataset is missing 7 word boxes compared to the 867 dataset. The omitted word boxes can be found in the supplementary materials,



(a) Regular (b) Irregular

Figure 2: Examples of regular (IIIT5k, SVT, IC03, IC13) and irregular (IC15, SVTP, CUTE) real-world datasets.

- **ICDAR2013 (IC13)** [13] inherits most of IC03's images and was also created for the ICDAR 2013 Robust Reading competition. It contains 848 images for training and 1,095 images for evaluation, where pruning words with non-alphanumeric characters results in 1,015 images. Again, researchers have used two different versions for evaluation: 857 and 1,015 images. The 857-image set is a subset of the 1,015 set where words shorter than 3 characters are pruned.

Second, **irregular datasets** typically contain harder corner cases for STR, such as curved and arbitrarily rotated or distorted texts [24, 28, 5]:

- **ICDAR2015 (IC15)** [12] was created for the ICDAR 2015 Robust Reading competitions and contains 4,468 images for training and 2,077 images for evaluation. The images are captured by Google Glasses while under the natural movements of the wearer. Thus, many are noisy, blurry, and rotated, and some are also of low resolution. Again, researchers have used two different versions for evaluation: 1,811 and 2,077 images. Previous papers [4, 2] have only used 1,811 images, discarding non-alphanumeric character images and some extremely rotated, perspective-shifted, and curved images for evaluation. Some of the discarded word boxes can be found in the supplementary materials,
- **SVT Perspective (SP)** [21] is collected from Google Street View and contains 645 images for evaluation. Many of the images contain perspective projections due to the prevalence of non-frontal viewpoints,
- **CUTE80 (CT)** [22] is collected from natural scenes and contains 288 cropped images for evaluation. Many of these are curved text images.

Notice that, Table 1 provides us a critical issue that prior works evaluated their models on **different benchmark datasets**. Specifically, the evaluation has been conducted on different versions of benchmarks in IC03, IC13 and IC15. In IC03, 7 examples can cause a performance gap by 0.8% that is a huge gap when comparing those of prior performances. In the case of IC13 and IC15, the gap of the example numbers is even bigger than those of IC03.

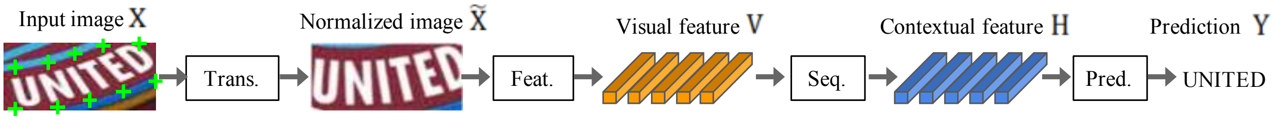


Figure 3: Visualization of an example flow of scene text recognition. We decompose a model into four stages.

3. STR Framework Analysis

The goal of the section is introducing the scene text recognition (STR) framework consisting of four stages, derived from commonalities among independently proposed STR models. After that, we describe the module options in each stage.

Due to the resemblence of STR to computer vision tasks (*e.g*. object detection) and sequence prediction tasks, STR has benefited from high-performance convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The first combined application of CNN and RNN for STR, Convolutional-Recurrent Neural Network (CRNN) [23], extracts CNN features from the input text image, and reconfigures them with an RNN for robust sequence prediction. After CRNN, multiple variants [24, 15, 17, 16, 26, 4, 3] have been proposed to improve performance. For rectifying arbitrary text geometries, as an example, transformation modules have been proposed to normalize text images [24, 17, 16]. For treating complex text images with high intrinsic dimensionality and latent factors (*e.g*. font style and cluttered background), improved CNN feature extractors have been incorporated [15, 26, 4]. Also, as people have become more concerned with inference time, some methods have even omitted the RNN stage [3]. For improving character sequence prediction, attention based decoders have been proposed [15, 24].

The four stages derived from existing STR models are as follows:

1. **Transformation (Trans.)** normalizes the input text image using the Spatial Transformer Network (STN [11]) to ease downstream stages.
2. **Feature extraction (Feat.)** maps the input image to a representation that focuses on the attributes relevant for character recognition, while suppressing irrelevant features such as font, color, size, and background.
3. **Sequence modeling (Seq.)** captures the contextual information within a sequence of characters for the next stage to predict each character more robustly, rather than doing it independently.
4. **Prediction (Pred.)** estimates the output character sequence from the identified features of an image.

We provide Figure 3 for an overview and all the architectures we used in this paper are found in the supplementary materials.

3.1. Transformation stage

The module of this stage transforms the input image X into the normalized image $\tilde{X}$. Text images in natural scenes come in diverse shapes, as shown by curved and tilted texts. If such input images are fed unaltered, the subsequent feature extraction stage needs to learn an invariant representation with respect to such geometry. To reduce this burden, thin-plate spline **(TPS)** transformation, a variant of the spatial transformation network (STN) [11], has been applied with its flexibility to diverse aspect ratios of text lines [24, 17]. TPS employs a smooth spline interpolation between a set of *fiducial points*. More precisely, TPS finds multiple fiducial points (green '+' marks in Figure 3) at the upper and bottom enveloping points, and normalizes the character region to a predefined rectangle. Our framework allows for the selection or de-selection of TPS.

3.2. Feature extraction stage

In this stage, a CNN abstract an input image (i.e., X or $\tilde{X}$) and outputs a visual feature map $V = \{v_i\}, i = 1, \ldots, I$ (I is the number of columns in the feature map). Each column in the resulting feature map by a feature extractor has a corresponding distinguishable receptive field along the horizontal line of the input image. These features are used to estimate the character on each receptive field.

We study three architectures of **VGG** [25], **RCNN** [15], and **ResNet** [9], previously used as feature extractors for STR. VGG in its original form consists of multiple convolutional layers followed by a few fully connected layers [25]. RCNN is a variant of CNN that can be applied recursively to adjust its receptive fields depending on the character shapes [15, 26]. ResNet is a CNN with *residual connections* that eases the training of relatively deeper CNNs.

3.3. Sequence modeling stage

The extracted features from Feat. stage are reshaped to be a sequence of features V. That is, each column in a feature map $v_i \in V$ is used as a frame of the sequence. However, this sequence may suffer the lack of contextual information. Therefore, some previous works use Bidirectional LSTM **(BiLSTM)** to make a better sequence $H = \text{Seq.}(\text{V})$ after the feature extraction stage [23, 24, 4]. On the other hand, Rosetta [3] removed the BiLSTM to reduce computational complexity and memory consumption. Our framework allows for the selection or de-selection of BiLSTM.

3.4. Prediction stage

In this stage, from the input H, a module predict a sequence of characters, (i.e., $Y = y_1, y_2, \dots$). By summing up previous works, we have two options for prediction: (1) Connectionist temporal classification **(CTC)** [6] and (2) attention-based sequence prediction **(Attn)** [24, 4]. CTC allows for the prediction of a non-fixed number of a sequence even though a fixed number of the features are given. The key methods for CTC are to predict a character at each column ($h_i \in H$) and to modify the full character sequence into a non-fixed stream of characters by deleting repeated characters and *blanks* [6, 23]. On the other hand, Attn automatically captures the information flow within the input sequence to predict the output sequence [1]. It enables an STR model to learn a character-level language model representing output class dependencies.

4. Experiment and Analysis

This section contains the evaluation and analysis of all possible STR module combinations (2×3×2×2= 24 in total) from the four-stage framework in §3, all evaluated under the common training and evaluation dataset constructed from the datasets listed in §2.

4.1. Implementation detail

As we described in §2, training and evaluation datasets influences the measured performances of STR models significantly. To conduct a fair comparison, we have fixed the choice of training, validation, and evaluation datasets.

STR training and model selection We use an union of MJSynth 8.9 M and SynthText 5.5 M (14.4 M in total) as our training data. We adopt the AdaDelta [29] optimizer, whose decay rate is set to $\rho = 0.95$. The training batch size is 192, and the number of iterations is 300 K. Gradient clipping is used at magnitude 5. All parameters are initialized with He's method [8]. We use the union of the training sets IC13, IC15, IIIT, and SVT as the validation data, and validated the model after every 2000 training stpdf to select the model with the highest accuracy on this set. Notice that, the validation set does not contain the IC03 train data because some of them were duplicated in the evaluation dataset of IC13. The total number of duplicated scene images is 34, and they contain 215 word boxes. Duplicated examples can be found in the supplementary materials.

Evaluation metrics In this paper, we provide a thorough analysis on STR combinations in terms of accuracy, time, and memory aspects altogether. For accuracy, we measure the success rate of word predictions per image on the 9 real-world evaluation datasets involving all subsets of the benchmarks, as well as a unified evaluation dataset (8,539 images in total); 3,000 from IIIT, 647 from SVT, 867 from IC03, 1015 from IC13, 2,077 from IC15, 645 from SP, and 288 from CT. We only evaluate on alphabets and digits. For each STR combination, we have run five trials with different initialization random seeds and have averaged their accuracies. For speed assessment, we measure the per-image average clock time (in millisecond) for recognizing the given texts under the same compute environment, detailed below. For memory assessment, we count the number of trainable floating point parameters in the entire STR pipeline.

Environment: For a fair speed comparison, all of our evaluations are performed on the same environment: an Intel Xeon(R) E5-2630 v4 2.20GHz CPU, an NVIDIA TESLA P40 GPU, and 252GB of RAM. All experiments are performed with NAVER Smart Machinie Learning (NSML) platform [14].

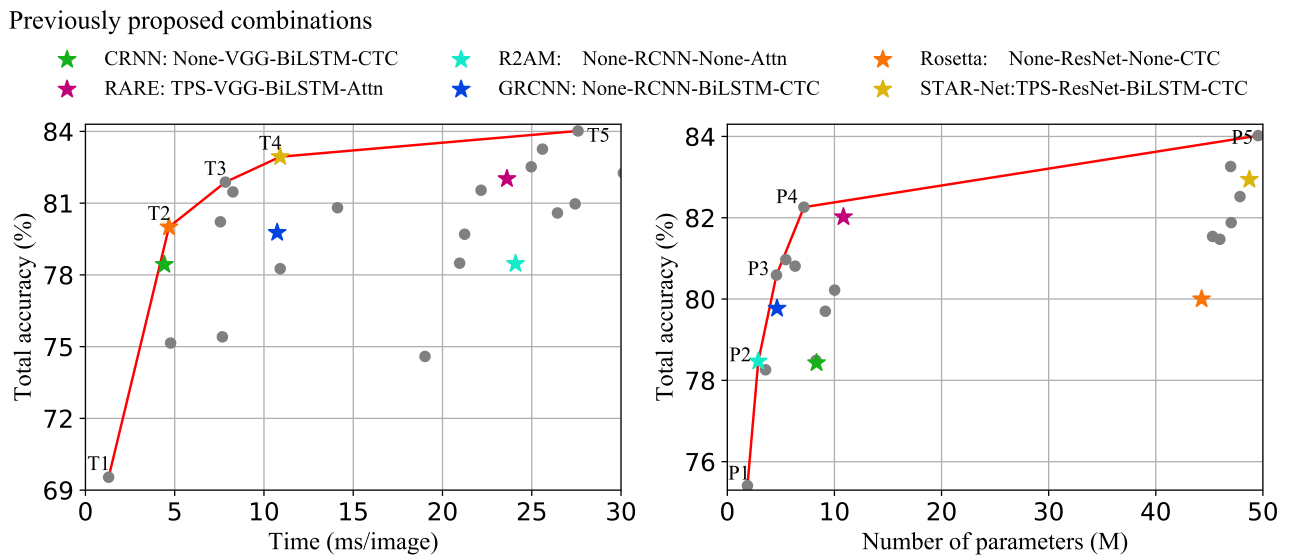
4.2. Analysis on training datasets

We investigate the influence of using different groups of the training datasets to the performance on the benchmarks. As we mentioned in §2.1, prior works used different sets of the training datasets and left uncertainties as to the contributions of their models to improvements. To unpack this issue, we examined the accuracy of our best model from §4.3 with different settings of the training dataset. We obtained 80.0% total accuracy by using only MJSynth, 75.6% by using only SynthText, and 84.1% by using both. The combination of MJSynth and SynthText improved accuracy by more than 4.1%, over the individual usages of MJSynth and SynthText. A lesson from this study is that the performance results using different training datasets are incomparable, and such comparisons fail to prove the contribution of the model, which is why we trained all models with the same training dataset, unless mentioned otherwise.

Interestingly, training on 20% of MJSynth (1.8M) and 20% of SynthText (1.1M) together (total 2.9M $\approx$ the half of SynthText) provides 81.3% accuracy – better performance than the individual usages of MJSynth or SynthText. MJSynth and SynthText have different properties because they were generated with different options, such as distortion and blur. This result showed that the diversity of training data can be more important than the number of training examples, and that the effects of using different training datasets is more complex than simply concluding more is better.

4.3. Analysis of trade-offs for module combinations

Here, we focus on the accuracy-speed and accuracy-memory trade-offs shown in different combinations of modules. We provide the full table of results in the supplementary materials. See Figure 4 for the trade-off plots of all 24 combinations, including the six previously proposed STR models (Stars in Figure 4). In terms of the accuracy-time trade-off, Rosetta and STAR-net are on the frontier and the other four prior models are inside of the frontier. In terms



#	Trans.	Feat.	Seq.	Pred.	Acc.	Time	params
					%	ms	$\times 10^6$
T1	None	VGG	None	CTC	69.5	1.3	5.6
T2	None	**ResNet**	None	CTC	80.0	4.7	46.0
T3	None	ResNet	**BiLSTM**	CTC	81.9	7.8	48.7
T4	**TPS**	ResNet	BiLSTM	CTC	82.9	10.9	49.6
T5	TPS	ResNet	BiLSTM	**Attn**	84.0	27.6	49.6

(a) Accuracy versus time trade-off curve and its frontier combinations

#	Trans.	Feat.	Seq.	Pred.	Acc.	Time	params
					%	ms	$\times 10^6$
P1	None	RCNN	None	CTC	75.4	7.7	1.9
P2	None	RCNN	None	**Attn**	78.5	24.1	2.9
P3	**TPS**	RCNN	None	Attn	80.6	26.4	4.6
P4	TPS	RCNN	**BiLSTM**	Attn	82.3	30.1	7.2
P5	TPS	**ResNet**	BiLSTM	Attn	84.0	27.6	49.6

(b) Accuracy versus memory trade-off curve and its frontier combinations

Figure 4: Two types of trade-offs exhibited by STR module combinations. Stars indicate previously proposed models and circular dots represent new module combinations evaluated by our framework. Red solid curves indicate the trade-off frontiers found among the combinations. Tables under each plot describe module combinations and their performance on the trade-off frontiers. Modules in bold denote those that have been changed from the combination directly before it; those modules improve performance over the previous combination while minimizing the added time or memory cost.

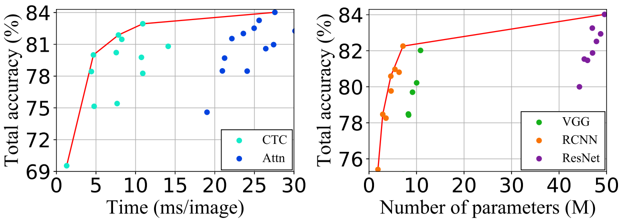


Figure 5: Color-coded version of Figure 4, according to the prediction (left) and feature extraction (right) modules. They are identified as the most significant factors for speed and memory, respectively.

of the accuracy-memory trade-off, R2AM is on the frontier and the other five of previously proposed models are inside of the frontier. Module combinations along the trade-off frontiers are labeled in ascending ascending order of accuracy (T1 to T5 for accuracy-time and P1 to P5 for accuracy-memory).

Analysis of combinations along the trade-off frontiers. As shown in Table 4a, T1 takes the minimum time by not including any transformation or sequential module. Moving from T1 to T5, the following modules are introduced in order (indicated as bold): ResNet, BiLSTM, TPS, and Attn. Note that from T1 to T5, a single module changes at a time. Our framework provides a smooth shift of methods that gives the least performance trade-off depending on the application scenario. They sequentially increase the complexity of the overall STR model, resulting in increased performance at the cost of computational efficiency. ResNet, BiLSTM, and TPS introduce relatively moderate overall slow down (1.3ms→10.9ms), while greatly boosting accuracy (69.5%→82.9%). The final change, Attn, on the other hand, only improves the accuracy by 1.1% at a huge cost in efficiency (27.6 ms).

As for the accuracy-memory trade-offs shown in Table 4b, P1 is the model with the least amount of memory consumption, and from P1 to P5 the trade-off between memory and accuracy takes place. As in the accuracy-speed trade-off, we observe a single module shift at each step up to P5, where the changed modules are: Attn, TPS, BiLSTM, and ResNet. They sequentially increase the accuracy at the cost of memory. Compared to VGG used in T1, we observe that RCNN in P1-P4 is lighter and gives a good accuracy-memory trade-off. RCNN requires a small number of unique CNN layers that are repeatedly applied. We observe that transformation, sequential, and prediction modules are not significantly contributing to the memory consumption (1.9M→7.2M parameters). While being lightweight overall, these modules provide accuracy improvements (75.4%→82.3%). The final change, ResNet, on the other hand, increases the accuracy by 1.7% at the cost of increased memory consumption from 7.2M to 49.6M floating point parameters. Thus, a practitioner concerned about memory consumption can be assured to choose specialized transformation, sequential, and prediction modules relatively freely, but should refrain from the use of heavy feature extractors like ResNets.

The most important modules for speed and memory. We have identified the module-wise impact on speed and memory by color-coding the scatter plots in Figure 4 according to module choices. The full set of color-coded plots is in the supplementary materials. Here, we show the scatter plots with the most speed- and memory-critical modules, namely the prediction and feature extraction modules, respectively, in Figure 5.

There are clear clusters of combinations according to the prediction and feature modules. In the accuracy-speed trade-off, we identify CTC and Attn clusters (the addition of Attn significantly slows the overall STR model). On the other hand, for accuracy-memory trade-off, we observe that the feature extractor contributes towards memory most significantly. It is important to recognize that the most significant modules for each criteria differ, therefore, practitioners under different applications scenarios and constraints should look into different module combinations for the best trade-offs depending on their needs.

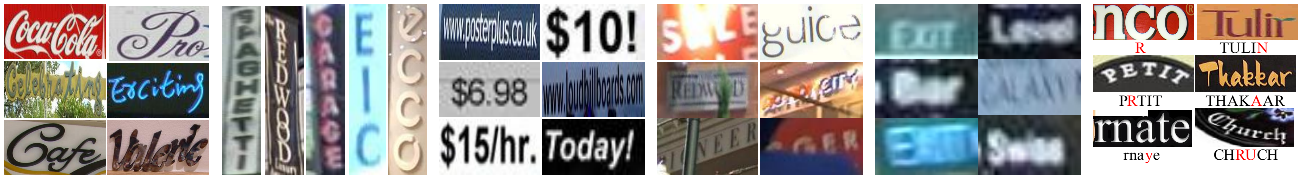
4.4. Module analysis

Here, we investigate the module-wise performances in terms of accuracy, speed, and memory demand. For this analysis, the marginalized accuracy of each module is calculated by averaging out the combination including the module in Table 2. Upgrading a module at each stage requires additional resources, time or memory, but provides performance improvements. The table shows that the performance improvement in irregular datasets is about two times that of regular benchmarks over all stages. when comparing accuracy improvement versus time usage, a sequence of ResNet, BiLSTM, TPS, and Attn is the most efficient upgrade order of the modules from a base combination of None-VGG-None-CTC. This order is the same order of combinations for the accuracy-time frontiers (T1→T5). On the other hand, an accuracy-memory perspective finds RCNN, Attn, TPS, BiLSTM and ResNet as the most efficient upgrading order for the modules, like the order of the accuracy-memory frontiers (P1→P5). Interestingly, the efficient order of modules for time is reverse from those for memory. The different properties of modules provide different choices in practical applications. In addition, the module ranks in the two perspectives are the same as the order of the frontier module changes, and this shows that each module contributes to the performances similarly under all combinations.

Stage	Module	Accuracy		Time	params
		Regular (%)	Irregular (%)	ms/image	$\times 10^6$
Trans.	None	85.6	65.7	N/A	N/A
	TPS	86.7(**+1.1**)	69.1(**+3.4**)	**3.6**	**1.7**
Feat.	VGG	84.5	63.9	1.0	5.6
	RCNN	86.2(**+1.7**)	67.3(**+3.4**)	**6.9**	1.8
	ResNet	88.3(**+3.8**)	71.0(**+7.1**)	4.1	**44.3**
Seq.	None	85.1	65.2	N/A	N/A
	BiLSTM	87.6(**+2.5**)	69.7(**+4.5**)	**3.1**	**2.7**
Pred.	CTC	85.5	66.1	0.1	0.0
	Attn	87.2(**+1.7**)	68.7(**+2.6**)	**17.1**	**0.9**

Table 2: Study of modules at the four stages with respect to total accuracy, inference time, and the number of parameters. The accuracies are acquired by taking the mean of the results of the combinations including that module. The inference time and the number of parameters are measured individually.

Qualitative analysis Each module contributes to identify text by solving targeted difficulties of STR tasks, as described in §3. Figure 7 shows samples that are only correctly recognized when certain modules are upgraded (e.g. from VGG to ResNet backbone). Each row shows a module upgrade at each stage of our framework. Presented samples are failed before the upgrade, but becomes recognizable afterward. TPS transformation normalizes curved and perspective texts into a standardized view. Predicted results show dramatic improvements especially for "POLICE" in a circled brand logo and "AIRWAYS" in a perspective view of a storefront sign. Advanced feature extractor, ResNet, results in better representation power, improving on cases with heavy background clutter "YMCA", "CITYARTS") and unseen fonts ("NEUMOS"). BiLSTM leads to better context modelling by adjusting the receptive field; it can ignore unrelatedly cropped characters ("I" at the end of "EXIT", "C" at the end of "G20"). Attention including implicit character-level language modeling finds missing or occluded character, such as "a" in "Hard", "t" in "to", and "S" in "HOUSE". These examples provide glimpses to the contribution points of the modules in real-world applications.



(a) Difficult fonts. (b) Vertical texts. (c) Special characters. (d) Occlusion. (e) Low resolution. (f) Label noise.

Figure 6: Samples of failure cases on all combinations of our framework.

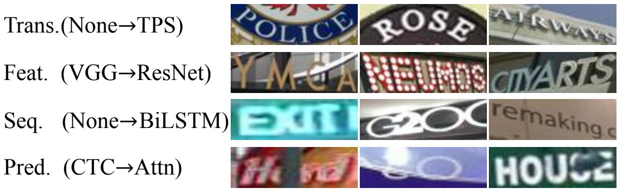


Figure 7: Challenging examples for the STR combinations without a specific module. All STR combinations without the notated modules failed to recognize text in the examples, but upgrading the module solved the problem.

4.5. Failure case analysis

We investigate failure cases of all 24 combinations. As our framework derived from commonalities among proposed STR models, and our best model showed competitive performance with previously proposed STR models, the presented failure cases constitute a common challenge for the field as a whole. We hope our analysis inspires future works in STR to consider addressing those challenges.

Among 8,539 examples in the benchmark datasets (§2), 644 images (7.5%) are not correctly recognized by *any* of the 24 models considered. We found six common failure cases as shown in Figure 6. The followings are discussion about the challenges of the cases and suggestion future research directions.

Calligraphic fonts: font styles for brands, such as “Coca Cola”, or shop names on streets, such as “Cafe”, are still in remaining challenges. Such diverse expression of characters requires a novel feature extractor providing generalized visual features. Another possible approach is regularization because the model might be over-fitting to the font styles in a training dataset.

Vertical texts: most of current STR models assumes horizontal text images, and thus structurally could not deal with vertical texts. Some STR models [28, 5] exploit vertical information also, however, vertical texts are not clearly covered yet. Further research would be needed to cover vertical texts.

Special characters: since current benchmarks do not evaluate special characters, existing works exclude them during training. This results in failure prediction, misleading the model to treat them as alphanumeric characters. We suggest training *with* special characters. This has resulted in a boost from 87.9% to 90.3% accuracy on IIIT.

Heavy occlusions: current methods do not extensively exploit contextual information to overcome occlusion. Future researches may consider superior language models to maximally utilize context.

Low resolution: existing models do not explicitly handle low resolution cases; image pyramids or super-resolution modules may improve performance.

Label noise: We found some noisy (incorrect) labels in the failure examples. We examined all examples in the benchmark to identify the ratio of the noisy labels. All benchmark datasets contain noisy labels and the ratio of mislabeling without considering special character was 1.3%, mislabeling with considering special character was 6.1%, and mislabeling with considering case-sensitivity was 24.1%.

We make all failure cases available in our Github repository, hoping that they will inspire further researches on corner cases of the STR problem.

5. Conclusion

While there has been great advances on novel scene text recognition (STR) models, they have been compared on inconsistent benchmarks, leading to difficulties in determining whether and how a proposed module improves the STR baseline model. This work analyzes the contribution of the existing STR models that was hindered under inconsistent experiment settings before. To achieve this goal, we have introduced a common framework among key STR methods, as well as consistent datasets: seven benchmark evaluation datasets and two training datasets (MJ and ST). We have provided a fair comparison among the key STR methods compared, and have analyzed which modules brings about the greatest accuracy, speed, and size gains. We have also provided extensive analyses on module-wise contributions to typical challenges in STR as well as the remaining failure cases.

Acknowledgment

The authors would like to thank Jaeheung Surh for helpful discussions.

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] Fedor Borisyuk, Albert Gordo, and Viswanath Sivakumar. Rosetta: Large scale system for text detection and recognition in images. In *KDD*, pages 71–79, 2018.

[4] 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.

[5] 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.

[6] 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.

[7] Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. Synthetic data for text localisation in natural images. In *CVPR*, 2016.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

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

[10] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. In *Workshop on Deep Learning, NIPS*, 2014.

[11] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *NIPS*, pages 2017–2025, 2015.

[12] Dimosthenis Karatzas, Lluis Gomez-Bigorda, Anguelos Nicolaou, Suman Ghosh, Andrew Bagdanov, Masakazu Iwamura, Jiri Matas, Lukas Neumann, Vijay Ramaseshan Chandrasekhar, Shijian Lu, et al. Icdar 2015 competition on robust reading. In *ICDAR*, pages 1156–1160, 2015.

[13] Dimosthenis Karatzas, Faisal Shafait, Seiichi Uchida, Masakazu Iwamura, Lluis Gomez i Bigorda, Sergi Robles Mestre, Joan Mas, David Fernandez Mota, Jon Almazan Almazan, and Lluis Pere De Las Heras. Icdar 2013 robust reading competition. In *ICDAR*, pages 1484–1493, 2013.

[14] Hanjoo Kim, Minkyu Kim, Dongjoo Seo, Jinwoong Kim, Heungseok Park, Soeun Park, Hyunwoo Jo, KyungHyun Kim, Youngil Yang, Youngkwan Kim, et al. Nsml: Meet the mlaas platform with a real-world case study. *arXiv:1810.09957*, 2018.

[15] Chen-Yu Lee and Simon Osindero. Recursive recurrent nets with attention modeling for ocr in the wild. In *CVPR*, pages 2231–2239, 2016.

[16] Wei Liu, Chaofeng Chen, and Kwan-Yee K Wong. Char-net: A character-aware neural network for distorted scene text recognition. In *AAAI*, 2018.

[17] 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.

[18] Yang Liu, Zhaowen Wang, Hailin Jin, and Ian Wassell. Synthetically supervised feature learning for scene text recognition. In *ECCV*, 2018.

[19] Simon M Lucas, Alex Panaretos, Luis Sosa, Anthony Tang, Shirley Wong, and Robert Young. Icdar 2003 robust reading competitions. In *ICDAR*, pages 682–687, 2003.

[20] Anand Mishra, Karteek Alahari, and CV Jawahar. Scene text recognition using higher order language priors. In *BMVC*, 2012.

[21] Trung Quy Phan, Palaiahnakote Shivakumara, Shangxuan Tian, and Chew Lim Tan. Recognizing text with perspective distortion in natural scenes. In *ICCV*, pages 569–576, 2013.

[22] Anhar Risnumawan, Palaiahankote Shivakumara, Chee Seng Chan, and Chew Lim Tan. A robust arbitrary text detection system for natural scene images. In *ESWA*, volume 41, pages 8027–8048. Elsevier, 2014.

[23] 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.

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

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

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

[27] Kai Wang, Boris Babenko, and Serge Belongie. End-to-end scene text recognition. In *ICCV*, pages 1457–1464, 2011.

[28] Xiao Yang, Dafang He, Zihan Zhou, Daniel Kifer, and C Lee Giles. Learning to read irregular text with attention mechanisms. In *IJCAI*, 2017.

[29] Matthew D Zeiler. Adadelta: an adaptive learning rate method. In *arXiv:1212.5701*, 2012.