Searching a High Performance Feature Extractor for Text Recognition Network

Hui Zhang, Quanming Yao, *Member, IEEE* James T. Kwok, *Fellow, IEEE*
Xiang Bai, *Senior Member, IEEE*

Abstract—Feature extractor plays a critical role in text recognition (TR), but customizing its architecture is relatively less explored due to expensive manual tweaking. In this work, inspired by the success of neural architecture search (NAS), we propose to search for suitable feature extractors. We design a domain-specific search space by exploring principles for having good feature extractors. The space includes a 3D-structured space for the spatial model and a transformed-based space for the sequential model. As the space is huge and complexly structured, no existing NAS algorithms can be applied. We propose a two-stage algorithm to effectively search in the space. In the first stage, we cut the space into several blocks and progressively train each block with the help of an auxiliary head. We introduce the latency constrain into the second stage and search sub-network from the trained supernet via natural gradient descent. In experiments, a series of ablation studies are performed to better understand the designed space, search algorithm, and searched architectures. We also compare the proposed method with various state-of-the-art ones on both hand-written and scene TR tasks. Extensive results show that our approach can achieve better recognition performance with less latency. Code is avaliable at https://github.com/AutoML-Research/TREFE

Index Terms—Neural architecture search (NAS), Convolutional neural networks (CNN), Text recognition (TR), Transformer

1 Introduction

Text recognition (TR) [1], [2], which targets at extracting text from document or natural images, has attracted great interest from both the industry and academia. TR is a challenging problem [3] as the text can have diverse appearances and large variations in size, fonts, background, writing style, and layout.

Figure 1 shows the typical TR pipeline. It can be divided into three modules: (i) An optional pre-processing module which transforms the input image to a more recognizable form. Representative methods include rectification [3], super-resolution [4] and denoising [5]. (ii) A feature extractor, which extracts features from the text images. Most of them [3], [6], [7] use a combination of convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The CNN extracts spatial features from the image, which are then enhanced by the RNN for the generation of robust sequence features [8], [9]. (iii) A recognition head, which outputs the character sequence. Popular choices are based on connectionist temporal classification [8], segmentation [10], sequence-to-sequence attention [3], and parallel attention [6].

The feature extractor plays an important role in TR. For example, in [3], [9], significant performance gains are observed by simply replacing the feature extractor from VGG [11] to ResNet [12]. Furthermore, the feature extractor often requires a lot of compute and storage [14], [17]. However, while a lot of improvements have been proposed for the pre-processing module and TR head, design of the feature extractor is less explored. Existing methods [3], [6], [7] often directly use CNNs and RNNs that are originally designed for other tasks (Table 1), without specially tuning them for TR. Examples include the direct use of ResNet that is originally used for image classification, and BiLSTM [18] from language translation. Moreover, TR systems often have inference latency constraints on deployment to real-world devices [19], [20]. However, existing designs do not explicitly take this into account. Manually tweaking the TR system to satisfy the latency constraint while maintaining a high recognition accuracy can be hard [20].

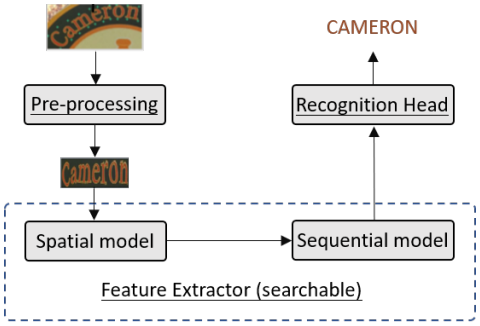


Fig. 1. The typical text recognition (TR) pipeline. In this paper, we focus on the search of a good feature extractor.

Recently, it has been shown that neural architecture search (NAS) [21] can produce good network architectures in tasks such as computer vision (e.g., image classification [22], [23], semantic segmentation [24] and object detection [25]). Inspired by this, rather than relying on experts

- *H. Zhang is with 4Paradigm Inc; Q. Yao is with Department of Electronic Engineering, Tsinghua University; J. Kwok is with Department of Computer Science and Engineering, Hong Kong University of Science and Technology; X. Bai is with Department of Electronics and Information Engineering, Huazhong University of Science and Technology.*
- *Corespondance is to Q.Yao at qyaoaa@tsinghua.edu.cn*

Manuscript received xxx; revised xxx.

TABLE 1
Some well-known hand-designed text recognition (TR) algorithms with NAS-based methods.

	method	spatial model		sequential model	search algorithm	deployment-aware
		downsampling path	conv layer			
hand-designed	CRNN [8]	fixed	vgg [11]	BiLSTM	—	×
	ASTER [3]	fixed	residual [12]	BiLSTM	—	×
	GFCN [13]	fixed	gated-block [13]	-	—	×
	SCRN [14]	fixed	residual [12]	BiLSTM	—	×
NAS	STR-NAS [15]	fixed	searched	BiLSTM	grad.	×
	AutoSTR [16]	two-dim	searched	BiLSTM	grid+grad.	×
	TREFE (proposed)	two-dim	searched	searched	NG	√

to design architectures, we propose the use of one-shot NAS [22], [23], [26] to search for a high-performance TR feature extractor. Specifically, we first design TR-specific search spaces for the spatial and sequential feature extractors. For the spatial component, the proposed search space allows selection of both the convolution type and feature downsampling path. For the sequential component, we propose to use transformer instead, which has better parallelism than the BiLSTM commonly used in TR. However, we find the vanilla transform is hard to beat BiLSTM. Thus, we further explore the recent advances of the Transformer, and search for variants of the transformer [27].

As the resultant supernet is huge, we propose to use the two-stage one-shot NAS approach [20], [28]. In the first stage, inspired by the success of progressive layer-wise training of deep networks [29], [30], we train the supernet in a greedy block-wise manner. In the second stage, instead of using evolutionary algorithms or random search as in [20], [28], [31], we use natural gradient descent [32] to more efficiently search for a compact architecture from the trained supernet. Resource constraints on the deployment environment can also be easily added in this stage, leading to models that are deployment-aware. Extensive experiments on a number of standard benchmark datasets demonstrate that the resultant TR model outperforms the state-of-the-arts in terms of both accuracy and inference speed.

Concurrently, Hong *et al.* [15] also considered the use of NAS in scene text recognition. However, they only search for the convolution operator, while we search for both the spatial feature and sequential feature extractors with deployment constraints (see Table 1). As a result, our search space is much larger and more complex, and a straightforward application of existing NAS algorithms is not efficient.

This paper is based on an earlier conference version [16], with the following major extensions:

- The search space is expanded by including search on the sequential model (Section 3.1.2) and allow more possibility of downsampling path for the spatial model (Section 3.1.1). Experiments in Section 4.5 demonstrates that both parts contribute the performance improvement over AutoSTR.
- The search algorithm is redesigned (Section 3.2). In [16], the search is performed in a step-wise search space that cannot explore all candidate architectures and assess the real deployment performance. In this paper, we first construct a supernet that contains all candidates in the search space (Section 3.2.1), which also allows direct evaluation of the deployment performance of each compact architecture (Section 3.2.3). In order to train the supernet easily, we propose a progressive training strategy (Section 3.2.2). In the search stage for candidate structures, we propose to use natural gradient descent [32] and introduce latency constraint for deployment awareness (Section 3.2.3).
- Experiments are much more extensive. While the conference version [16] only evaluates on scene text datasets, in Section 4.2 we also evaluate on handwritten text datasets. Besides, more recent state-of-the-art baselines are included. We also provide a detailed examination of the searched architectures (Section 4.3) and search algorithm (Section 4.4).

2 Related Works

2.1 Text Recognition (TR)

In the last decade, deep learning has been highly successful and achieves remarkable performance on the recognition of handwritten text [13], [33] and scene text [3], [34]. However, the large variations in size, fonts, background, writing style, and layout still make TR from images a challenging problem [35]. Existing TR methods usually has three modules: (i) pre-processing module, (ii) feature extractor, and (iii) TR head (Figure 1).

2.1.1 Pre-Processing Module

The pre-processing module makes the input text image more easily recognizable. Shi *et al.* [3], [36] uses a learnable Spatial Transformer Network (STN) to rectify the irregular text to a canonical form before recognition. Subsequent methods [14], [37] further improve the transformation to achieve more accurate rectifications. Wang *et al.* [4] introduces a super-resolution transformation to make blurry and low-resolution images clearer. Luo *et al.* [5] first separates text from the complex background to make TR easier.

2.1.2 Spatial and Sequential Feature Extractors

Given an $H \times W$ input image, the feature extractor [3], [6], [8] first uses a spatial model (which is a deep convolutional network) to extract a $H' \times W' \times D$ feature map, where H', W' are the downsampled height and width, and D is the number of channels. As an example, consider the widely-used ASTER [3]. Its pre-processing module arranges the characters horizontally to a 32×100 text image. Spatial features are extracted by the ResNet [12] (blocks 0-5 in Figure 2). Specifically, 2 sets of convolution filters (with stride $(2, 2)$) first downsample the image to 8×25, and then 3 sets of convolution filters (with stride $(2, 1)$) further downsample it to a 25-dimensional feature vector. It also follows

the common practice of doubling the number of convolution filters when the feature map resolution is changed [8], [9], [12], [14], [38] (see also Figure 3).

The spatial model output is enhanced by extracting contextual information using a sequential model. Specifically, its $H' \times W' \times D$ feature map is first reshaped to a $D \times T$ matrix, where $T = H'W'$, and then processed as a sequence of T feature vectors. In ASTER, BiLSTM [18] layers are built on top of the convolutional layers (BiLSTM 1-2 in Figure 2).

	Layers	Out Size	Configurations
Feature extractor	Block 0	32×100	3×3 conv, s 1×1
	Block 1	16×50	$\begin{bmatrix} 1 \times 1 \text{ conv, } 32 \\ 3 \times 3 \text{ conv, } 32 \end{bmatrix} \times 3$, s 2×2
	Block 2	8×25	$\begin{bmatrix} 1 \times 1 \text{ conv, } 64 \\ 3 \times 3 \text{ conv, } 64 \end{bmatrix} \times 4$, s 2×2
	Block 3	4×25	$\begin{bmatrix} 1 \times 1 \text{ conv, } 128 \\ 3 \times 3 \text{ conv, } 128 \end{bmatrix} \times 6$, s 2×1
	Block 4	2×25	$\begin{bmatrix} 1 \times 1 \text{ conv, } 256 \\ 3 \times 3 \text{ conv, } 256 \end{bmatrix} \times 6$, s 2×1
	Block 5	1×25	$\begin{bmatrix} 1 \times 1 \text{ conv, } 512 \\ 3 \times 3 \text{ conv, } 512 \end{bmatrix} \times 3$, s 2×1
	BiLSTM 1	25	256 hidden units
	BiLSTM 2	25	256 hidden units

Fig. 2. Feature extractor in ASTER [3]. For a convolutional layer, "Out Size" is the feature map size (height $\times$ width). For a sequential layer, "Out Size" is the sequence length. The symbol "s" is the stride of the first convolutional layer in a block.

Design of the feature extractor in TR is relatively less explored. Often, they simply adopt existing architectures [3], [6], [8], [38]. Manual adjustments can be very time-consuming and expensive. Moreover, they do not consider the latency constraints when the model is deployed on real devices (see Table 1).

2.1.3 Text Recognition (TR) Head

The TR head is used to recognize the text sequence. In recent years, many recognition heads have been proposed. Connectionist temporal classification (CTC) [39] trains a classifier to match the prediction with the target text sequence without need for prior alignment. The segmentation-based TR head [7], [10] attempts to locate each character and then applies a character classifier. Using the sequence-to-sequence model [40], the TR head in [3], [6], [38] uses the attention mechanism [41] to learn an implicit language model, which can be parallelized by take the reading order as query [7], [42]. CTC and parallelized attention are more latency-friendly than the sequence-to-sequence model, especially when the output sequence is long.

2.2 One-Shot Neural Architecture Search (NAS)

Traditionally, neural network architectures are taken as hyper-parameters, and optimized by algorithms such as reinforcement learning [43] and evolutionary algorithms [44]. This is expensive as each candidate architecture needs to be fully trained separately. One-shot NAS [22], [23], [26] significantly reduces the search time by sharing all network weights during training. Specifically, a supernet [22], [23], [45] subsumes all candidate architectures in the search space, and is trained only once. Each candidate architecture is a sub-network in the supernet, and its weights are simply inherited from the trained supernet without training.

There are two approaches in one-shot NAS. The first one combines supernet training and search in a *single* stage. Representative methods include DARTS [22], SNAS [46], ENAS [23], ProxylessNAS [26], and PNAS [47]. A weight is used to reflect the importance of each candidate operator. These weights are then learned together with the parameters of all candidate operators. Finally, operators with higher weights are selected from the trained supernet to construct the searched structure. However, operators that do not perform well at the beginning of the training process may not be fairly updated, leading to the selection of inferior architectures [48], [49]. The second approach follows a *two-stage* strategy, which decouples supernet training and search. In the pioneering work [45], a large supernet is trained and sub-networks are obtained by zeroing out some operators. The best architecture is selected by measuring the performance of each sub-network. Single-path one-shot (SPOS) [20] uniformly samples and updates a sub-network from the supernet in each iteration. After the supernet has been sufficiently trained, the best sub-network is selected by an evolutionary algorithm. Once-for-all (OFA) [28] is similar to SPOS, but proposes a *progressive shrinking* scheme, which trains sub-networks in the supernet from large to small. More recently, DNA [49] adopts knowledge distillation to improve fairness in supernet training, and BossNAS further improve DNA by leveraging self-supervised learning [50].

On deploying deep networks to a specific device, issues such as model size and inference time can become important. Above one-shot NAS methods have also been recently introduced to solve this problem. For example, MNASNet [51] and MobileNetV3 [52] measure the actual execution latency of a sampled architecture, and use it as a reward to train a recurrent neural network controller. ProxylessNAS [26] and FBNet [53] introduce a regularizer to the search objective which measures the expected network latency (or number of parameters), and stochastic gradient descent is used to search the architectures. In SPOS [20], an evolutionary algorithm is used to select architectures meeting the deployment requirements.

2.3 Transformer

For sequence modeling in natural language processing (NLP), the LSTM has gradually been replaced by the Transformer [27], which is more advantageous in allowing parallelism and extraction of long-range context features. The transformer takes a length-T sequence of D-dimensional features $\mathbf{Z} \in \mathbb{R}^{D \times T}$ as input. Using three multilayer perceptrons (MLPs), $\mathbf{Z}$ is transformed to the query $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$, respectively (all with size $D \times T$). Self-attention (SA) generates attention scores $\mathbf{A}$ from $\mathbf{Q}$ and $\mathbf{K}$:

$$\mathbf{A} = \mathbf{Q}^\top \mathbf{K} / \sqrt{D}, \tag{1}$$

where $\sqrt{D}$ is a scaling factor, and then use $\mathbf{A}$ to form weighted sums of columns in $\mathbf{V}$:

$$\mathrm{SA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{V} \cdot \mathrm{softmax}(\mathbf{A}),$$

where $[\mathrm{softmax}(\mathbf{A})]_{i,j} = e^{\mathbf{A}_{i,j}} / \sum_{t=1}^{T} e^{\mathbf{A}_{i,t}}$. Multiple attention heads $\{(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i)\}$ can also be used, leading to multi-head self-attention (*MHSA*) [27]:

$$\mathit{MHSA}\,(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{Concat}\,(\mathrm{head}_1, \ldots, \mathrm{head}_I)\,\mathbf{W}^O,$$

where $\text{head}_i = \text{SA}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i)$, $\mathbf{W}^O$ is a learnable parameter, and Concat$(\dots)$ concatenates multiple column vectors to a single vector. Finally, the *MHSA* output is followed by a two-layer feedforward network (*FFN*):

$$\text{MLP}(\mathbf{x}) = \text{ReLU}(\mathbf{x}\mathbf{W}_1)\mathbf{W}_2, \tag{2}$$

where $\mathbf{x} = \textit{MHSA}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, $\mathbf{W}_1, \mathbf{W}_2$ are learnable parameters, and ReLU is the rectified linear activation function.

With the initial success of the transformer, a lot of efforts have been made to improve its two core components: *MHSA* and *FFN*. The RealFormer [54] adds a skip-connection in *MHSA* to help propagation of the raw attention scores and stabilize training. The addition of relative distance awareness information can also improve self-attention [55]. Besides, there are efforts to improve the computational efficiency of *MHSA* (as in Reformer [56] and Performer [57]). As for *FFN* component, the Evolved Transformer [58] uses NAS to find a better *FFN* structure.

3 Proposed Methodology

As discussed in Section 2.1.2, the feature extractor has two components: (i) a spatial model for visual feature extraction, and (ii) a sequential model for sequence feature extraction. In TR, the feature extractor design is not well studied. Existing TR algorithms often simply reuse spatial and sequential model architectures that are originally designed for other tasks [3], [6], [8], [38]. However, it has been recently observed that different tasks (such as semantic segmentation [24] and object detection [25]) may require different feature extraction architectures. Thus, existing spatial and sequential feature extractors may not be suitable for TR. On the other hand, manual adjustments can be very time-consuming and expensive. Moreover, when the model is deployed on real devices, latency constraints cannot be easily considered. Inspired by NAS, we propose the TREFE algorithm, which automatically searches for a high-performance Text REcognition Feature Extractor.

3.1 Formulating TR as a NAS Problem

In this section, we first formulate TR as a NAS problem. The search space is inspired by existing manually-designed TR models and recent advances in the transformer architecture.

3.1.1 Search Space for the Spatial Model

Recall that the spatial model is a CNN (Section 2.1.2). Each convolutional layer $\mathcal{C}$ can be represented as $\mathcal{C}(\mathsf{X}; ct, s^h, s^w)$, where X is the input image tensor, ct is the type of convolution (e.g., a 3×3 convolution or 5×5 depth-wise separable convolution), and (s^h, s^w) are the strides in the height and width dimensions. The downsampling path, which downsamples the image to the feature map along with the convolution operations, can significantly affect the CNN's performance [8], [24], [25]. Instead of using manual designs, we explore the space of spatial model architectures and automatically search for the (ct, s^h, s^w) values in each layer.

The whole spatial model structure can be identified by $C \equiv \{ct_i\}_{i=1}^M$ and $S \equiv \{(s_i^h, s_i^w)\}_{i=1}^M$, where M is the number of convolution layers, $ct_i \in O_c$, the set of candidate convolution operations, and $(s_i^h, s_i^w) \in O_s$, the set of candidate stride values. Following [3], [38], [59], we assume that the input to the spatial model is of size $32 \times W$. The detailed choices of O_s and O_c are as follows.

- Following [3], [8], [14], [38], [60], we set $O_s = \{(2,2), (2,1), (1,1)\}$. We do not include $(1,2)$, as the horizontal direction should not be downsampled more than the vertical direction [8], [9], [14], [37], [60], as this can make neighboring characters more difficult to separate. Moreover, we double the number of filters when the resolution in that layer is reduced (i.e., when $(s_i^h, s_i^w) = (2,1)$ or $(2,2)$).
- As in NAS algorithms [15], [26], [52], [53], O_c contains inverted bottleneck convolution (MBConv) layers [61] with kernel size $k \in \{3, 5\}$ and expansion factor $e \in \{1, 6\}$.
- As in [3], [8], [16], we use an output feature map of size $1 \times W/4$. Since the input size of the spatial model is $32 \times W$, we have for each downsampling path,

$$S^h \equiv 32 = \prod_{i=1}^M s_i^h, \quad \text{and} \quad S^w \equiv 4 = \prod_{i=1}^M s_i^w. \tag{3}$$

Figure 3(a) shows the search space of the spatial model structure with M layers. Each blue node corresponds to a $h \times w \times c$ feature map $\boldsymbol{\beta}_{(h,w,c)}^l$ at layer l.[1] Each green edge corresponds to a candidate convolution layer $\mathcal{C}$ transforming $\boldsymbol{\beta}_{(h,w,c)}^l$, while each gray edge corresponds to a candidate stride in O_s. A connected path of blue nodes from the initial size $([32, W])$ to the size of the last feature map $([1, W/4])$ represents a candidate spatial model.

3.1.2 Search Space for the Sequential Model

Recall that in TR systems, the sequential model component is usually a recurrent network (Section 2.1.2), such as the BiLSTM [18]. Here, we instead use the transformer [27] which has higher parallelism. However, a straightforward application of the vanilla Transformer may not be desirable, as it can have performance inferior to the BiLSTM on tasks such as named entity recognition [62] and natural language inference [63]. In the following, we describe the proposed chanages to the Transformer structure.

Let each transform layer be $\mathcal{R}(\mathsf{V}, rt)$, where V is the input tensor and rt is the type of transform layer (e.g., a transformer layer without attention scaling). The structure of the sequential model is defined as $R \equiv \{rt_i\}_{i=1}^N$ where $rt_i \in O_r$, the set of candidate layers, and N is the number of transformer layers. Consider the ℓth transform layer. Inspired by recent advances on the transformer (Section 2.3), one can vary its design in the following four aspects. The first three are related to the *MHSA*, while the last one is on the *FFN* (Figure 4).

(i) As in the RealFormer [54], one can add a residual path from the previous layer to the current layer to facilitate propagation of attention scores. In other words, instead of using $\mathbf{A}^\ell = \mathbf{Q}^\top \mathbf{K}$, we can have

$$\mathbf{A}^\ell = \mathbf{Q}^\top \mathbf{K} + \mathbf{A}^{\ell-1},$$

for transform layer $\ell > 1$ (excluding the input layer).

(ii) As in [55], [62], one can add a relative distance embedding Rel to improve the distance and direction awareness of the attention score $\mathbf{A}$ in (1) as:

$$\mathbf{A} = \mathbf{A} + \mathsf{Rel},$$

1. In general, the (h, w, c) values can vary with l.

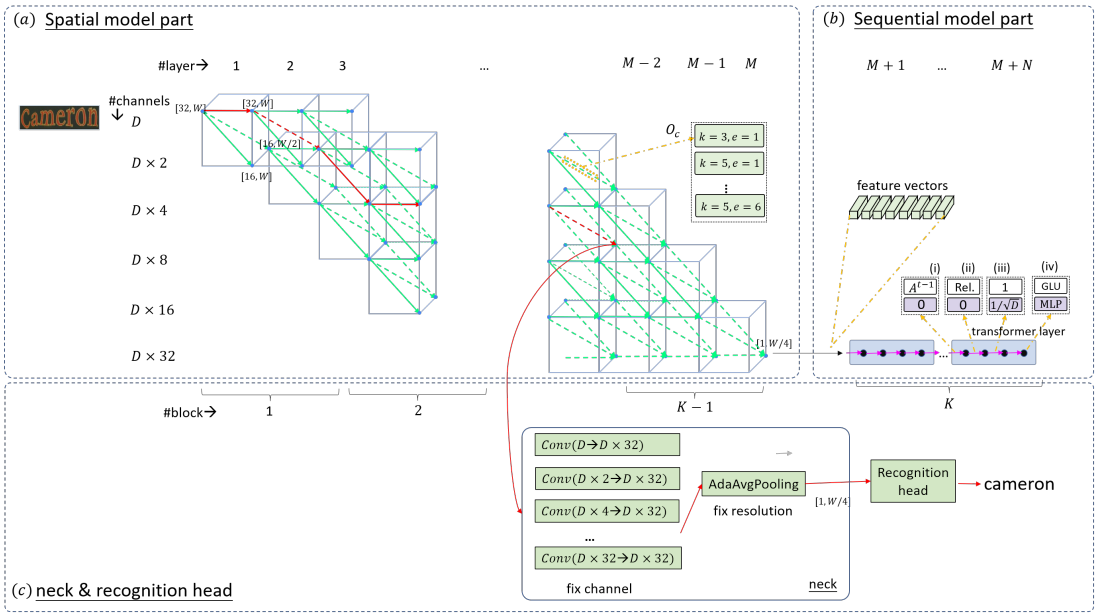


Fig. 3. Graph illustration of the proposed method TREFE. The search space of TREFE contains both spatial model (see Section 3.1.1) and sequential model (see Section 3.1.2) part, and the neck is used for supernet training (see Section 3.2.2).

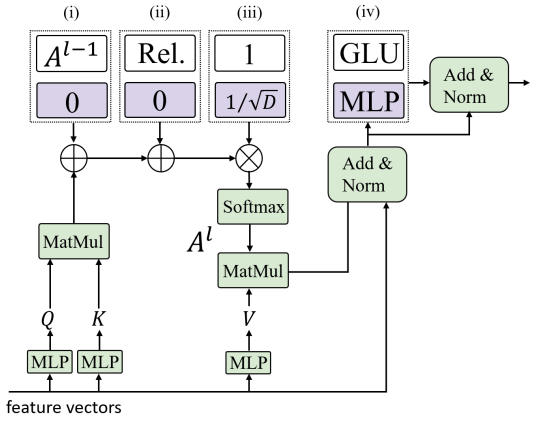


Fig. 4. Search space of a transformer layer. "Rel." is short for relative distance embedding. The purple boxes represent the original structures, and boxes with a white background indicate the alternative choices.

where $\text{Rel}_{tj} = \mathbf{q}_t \mathbf{r}^\top + \mathbf{u}\mathbf{k}_j^\top + \mathbf{v}\mathbf{r}^\top$, $\mathbf{u}, \mathbf{v}$ are learnable parameters, $\mathbf{q}_t$ is the tth query (tth column in $\mathbf{Q}$), $\mathbf{k}_j$ is the jth key (jth column in $\mathbf{K}$), $\mathbf{r} = [\mathbf{r}_i]$ is the relative position between $\mathbf{q}_t$ and $\mathbf{k}_j$, and is defined as

$$\mathbf{r}_i = \begin{cases} \sin\left((t-j)/(10000^{\frac{i}{D}})\right) & i \text{ is even} \\ \cos\left((t-j)/(10000^{\frac{i-1}{D}})\right) & i \text{ is odd} \end{cases}.$$

(iii) In computing the attention in (1), instead of including the scaling factor $1/\sqrt{D}$, one can drop this as in [62], leading to simply $\mathbf{A} = \mathbf{Q}^\top \mathbf{K}$.

(iv) Inspired by [34], [64], one can replace the *FFN*'s MLP in (2) with the gated linear unit (GLU) [64]:

$$\text{GLU}(\mathbf{x}) = (\mathbf{x}\mathbf{W}_1) \otimes \sigma(\mathbf{x}\mathbf{W}_2), \tag{4}$$

where $\mathbf{x}$ is the *MHSA* output, $\otimes$ is the element-wise product, and σ is the sigmoid function. This allows the *FFN* to select relevant features for prediction.

For a sequential model with N layers, we attach N copies of Figure 4 to the sequential model output in Figure 3(a). In Figure 3(b), each blue rectangle denotes a transformer layer, and each black node denotes the design choices in the transformer layer ((i), (ii), (iii), (iv) in Figure 4). The (magenta) path (with color magenta) in Figure 3(b) along the black nodes constructs a candidate sequential model.

3.1.3 Resource Constraints

As in Section 2.2, there can be resource constraints on deployment. We introduce resource constraints of the form:

$$\text{latency}\big(\mathcal{N}(\mathbf{w}, S, C, R); \mathcal{E}\big) \le r_{\max}, \tag{5}$$

where $\mathcal{N}(\mathbf{w}, S, C, R)$ is the TR model with network weights $\mathbf{w}$ and feature extractor architecture determined by (S, C, R), $\mathcal{E}$ is the environment, and $r_{\max}$ is the budget on the resource. For simplicity, only one resource constraint is considered. Extension to multiple resource constraints is straightforward.

3.1.4 Search Problem

Consider a feature extractor with M layers in the spatial model and N in the sequential model. Let $\mathcal{L}_{\text{tra}}$ be the training loss of the network, and $\mathcal{A}_{\text{val}}$ be the quality of the network on the validation set. The following formulates the search for an appropriate architecture $\mathcal{N}$.

Definition 1. *The search problem can be formulated as:*

$$\begin{aligned} &\arg\max_{S,C,R} \mathcal{A}_{val}(\mathcal{N}(\mathbf{w}^*, S, C, R)) \\ &s.t. \begin{cases} \mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{L}_{tra}(\mathcal{N}(\mathbf{w}, S, C, R)), \\ latency\big(\mathcal{N}(\mathbf{w}, S, C, R); \mathcal{E}\big) \le r_{\max}, \\ C \in \mathbb{C}, \quad R \in \mathbb{R}, \quad S \in \mathbb{P}, \end{cases} \end{aligned} \tag{6}$$

where

$$\mathbb{C} \equiv \underbrace{O_c \times \cdots \times O_c}_{M}, \ \mathbb{R} \equiv \underbrace{O_r \times \cdots \times O_r}_{N},$$

and $\mathbb{P} \equiv \{\{(s_i^h, s_i^w) \in O_s\}_{i=1}^M \mid \prod_{i=1}^M s_i^h = S^h, \prod_{i=1}^M s_i^w = S^w\}$ *encodes the constraints in (3).*

The combination of (S, C, R) generates a large number of candidate architectures. For example, in the experiments (Section 4), we use a spatial model with 20 MBConv layers. Every blue node in Figure 3 has 3 stride (s_h, s_w) candidates and 4 operator (O_c) candidates. There are a total of 155,040 candidate downsampling paths,[2] $155,040 \times 4^{20} \approx 1.7 \times 10^{17}$ candidate spatial model structures; whereas the sequential model has 4 transformer layers, and $(2^4)^4 = 65,536$ candidate structures. The whole search space thus has $1.7 \times 10^{17} \times 65,536 \approx 1.1 \times 10^{22}$ candidates, which is prohibitively large. Besides, note that problem (6) is a bi-level optimization problem. As in most NAS problems [21], it is typically expensive to solve. In particular, training each candidate architecture to get $\mathbf{w}^*$ is expensive. Hence, directly optimizing problem (6) is not practical.

3.2 Search Algorithm

Inspired by recent advances in NAS [21], we propose to solve (6) using one-shot NAS [20], [22], [26], which greatly simplifies the search by training only one supernet. However, the one-stage approach requires training the whole supernet, which demands tremendous GPU memory as the proposed search space is huge [26]. Hence, we will use the two-stage approach. However, the two-stage methods cannot be directly used. As the search space is huge, only a small fraction of the candidate architectures can be trained, and the untrained architectures will perform badly [49].

In Section 3.2.1, we first discuss design of the supernet. Inspired by layer-wise pre-training [29], [30], we propose in Section 3.2.2 a progressive strategy that trains the supernet in a block-wise manner. In Section 3.2.3, we propose to use natural gradient descent [32] to better search for a sub-architecture during the second stage.

3.2.1 Designing the Supernet in One-Shot NAS

There are two basic requirements in the supernet design [22], [23], [45]: (i) all candidates in the search space should be included; and (ii) each candidate can be expressed as a path in the supernet.

The proposed supernet has two parts: the spatial model and sequential model. It closely matches the search space in Figure 3. The spatial component (Figure 3(a)) is a 3D-mesh, in which each edge determines an operation that transforms the feature representation. A connected path from $[32, W]$ to $[1, W/4]$ represents a downsampling path. The choice of operations and downsampling path together determine the CNN. Figure 3(b) shows the sequential model component of the supernet.

3.2.2 Training the Supernet

The main challenge is how to fully and fairly train all candidate architectures in the supernet. A typical solution is to sample architectures uniformly and then train [20], [31]. However, uniform sampling in a huge search space is not effective. To alleviate this problem, we propose to divide the supernet (denoted Φ) into K smaller blocks $(\Phi_1, \Phi_2, \ldots, \Phi_K)$ and optimize them one by one. Since the spatial model is much larger than the sequential model, we take the whole sequential model as one block (Φ_K), and divide the spatial model into $K-1$ equal-sized blocks $(\Phi_1, \Phi_2, \ldots, \Phi_{K-1})$ (Figure 3(a)).

Algorithm 1 shows the training process. The weights for blocks $\Phi_1, \ldots, \Phi_K$ are progressively stacked, and updated together with the weights of the neck and recognition head by SGD. When training block Φ_k, we fix the trained weights for blocks $\Phi_1, \ldots, \Phi_{k-1}$, and skip the remaining blocks $\Phi_{k+1}, \ldots, \Phi_K$ (step 4). In each iteration, a path $\boldsymbol{\alpha}_k$ is uniformly sampled from Φ_k (step 6). Let $\mathbb{S}_k$ be the set of paths in $\Phi_1 \cup \cdots \cup \Phi_{k-1}$ whose output feature maps match in size with the input feature map of $\boldsymbol{\alpha}_k$. A new path in $\Phi_1 \cup \cdots \cup \Phi_k$ is formed by uniformly selecting a path $\boldsymbol{\alpha}_k^{\leftarrow} \in \mathbb{S}_k$ and connecting it with $\boldsymbol{\alpha}_k$ (step 7). As blocks $\Phi_{k+1}, \ldots, \Phi_K$ are skipped, the output of $\boldsymbol{\alpha}_k$ is connected to the recognition head via an extra auxiliary neck[3] (step 9) so that $\boldsymbol{\alpha}_k$'s output channel number and resolution match with those of the head's input.

Algorithm 1 Training the supernet.

1: Split the supernet into K blocks;
2: Insert an auxiliary neck between feature extractor and recognition head;
3: **for** block $k = 1, \ldots, K$ **do**
4: fix supernet weights for blocks $\Phi_1, \ldots, \Phi_{k-1}$;
5: **for** iteration $t = 1, \ldots, T$ **do**
6: sample a path $\boldsymbol{\alpha}_k$ from Φ_k;
7: sample a path $\boldsymbol{\alpha}_k^{\leftarrow}$ in $\mathbb{S}_k$ and connect it with $\boldsymbol{\alpha}_k$;
8: sample a mini-batch B_t from training data;
9: update weights of $\boldsymbol{\alpha_k}$, the neck and the recognition head by SGD on B_t;
10: **end for**
11: **end for**
12: **return** trained weights $\mathbf{W}^*$ of Φ.

3.2.3 Search for a Sub-Network

Recall that a path $\boldsymbol{\alpha}$ in the supernet corresponds to an architecture $\{S, C, R\}$ of the feature extractor. Let the trained weight of the supernet returned from Algorithm 1 be $\mathbf{W}^*$. Since the constraints $C \in \mathbb{C}$, $R \in \mathbb{R}$, and $S \in \mathbb{P}$ have been implicitly encoded by the supernet structure, and the supernet weights are already trained, problem (6) then simplifies to

$$\begin{aligned} \boldsymbol{\alpha}^* = \arg\max\nolimits_{\boldsymbol{\alpha}} \mathcal{A}_{\text{val}}\big(\mathcal{N}(\mathbf{W}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha})\big) \\ \text{s.t.} \quad \text{latency}\big(\mathcal{N}(\mathbf{W}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}); \mathcal{E}\big) \le r_{\max}, \end{aligned} \tag{7}$$

where $\mathbf{W}^*(\boldsymbol{\alpha})$ is the weight for path $\boldsymbol{\alpha}$, which can be easily extracted from $\mathbf{W}^*$ without re-training. SPOS [20] uses an evolutionary algorithm (EA) to solve (7). However, EA can suffer from the pre-maturity problem [65], in that the population is dominated by good architectures generated in the early stages. Diversity is rapidly reduced, and the EA converges to a locally optimal solution.

To avoid the above problem, we consider using stochastic relaxation on $\boldsymbol{\alpha}$ as in [46], and transform problem (7) to:

$$\begin{aligned} \max\nolimits_{\boldsymbol{\theta}} \ \mathbb{E}_{\boldsymbol{\alpha} \sim P_{\boldsymbol{\theta}}(\Phi)} \left[\mathcal{A}_{\text{val}}\big(\mathcal{N}(\mathbf{W}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha})\big)\right] \\ \text{s.t.} \quad \text{latency}\big(\mathcal{N}(\mathbf{W}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}); \mathcal{E}\big) \le r_{\max}, \end{aligned} \tag{8}$$

2. A backtracking algorithm computes the number of candidate downsampling paths. Please refer to Appendix A for details.

3. In the experiments, the neck is a small network with six parallel convolution layers and a adaptive pooling layer [12].

TABLE 2
Comparison between AutoSTR [16] and TREFE. Here, "SeqAtt" denotes sequential attention, and "ParAtt" denotes parallel attention.

	spatial model		sequential	recognition head		search	deployment
	downsampling path	operators	model	scene text	handwritten text	algorithm	awareness
AutoSTR	limited choices	same	fixed BiLSTM	SeqAtt	—	decoupled search	not support
TREFE	all possible paths		searched transformer	ParAtt	CTC	joint search	support

where $\mathbb{E}$ denotes the expectation, $P_{\boldsymbol{\theta}}$ is an exponential distribution (with parameter θ) on the search space Φ (details are in Appendix C). Sampling from $P_{\boldsymbol{\theta}}$ helps to explore more diverse architectures.

Algorithm 2 shows the search procedure. To optimize $\boldsymbol{\theta}$, we first sample a mini-batch B of architectures using the exponential distribution $P_{\boldsymbol{\theta}}$ (step 5). For each sampled architecture, its latency and validation set performance are measured (steps 7-8). Note that this takes negligible time compared to supernet training. Architectures that do not meet the latency requirements are dropped. The sampled architectures and corresponding performance scores are used to update $P_{\boldsymbol{\theta}}$ by natural gradient descent [32] (steps 18). Specifically, at the tth iteration, $\boldsymbol{\theta}$ is updated as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \rho \mathbf{F}^{-1}(\boldsymbol{\theta}_t)\mathbf{g}, \tag{9}$$

where ρ is the step-size,

$$\mathbf{F}(\boldsymbol{\theta}) = \mathbb{E}_{P_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\theta}} \ln P_{\boldsymbol{\theta}}(\boldsymbol{\alpha}) \left[\nabla_{\boldsymbol{\theta}} \ln P_{\boldsymbol{\theta}}(\boldsymbol{\alpha})\right]^{\top}\right], \tag{10}$$

is the Fisher information matrix [32], and $\mathbf{g}$ is the gradient

$$\mathbf{g} = \mathbb{E}_{P_{\boldsymbol{\theta}}}\Big[\mathcal{A}_{\text{val}}(\mathcal{N}(\mathbf{W}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}))\nabla_{\boldsymbol{\theta}} \ln\left(P_{\boldsymbol{\theta}}\left(\boldsymbol{\alpha}\right)\right)\Big]. \tag{11}$$

Note that $\mathbf{F}(\boldsymbol{\theta})$ and $\mathbf{g}$ in (10), (11) cannot be exactly evaluated as they require expensive integrations over the whole distribution. Thus, they are approximated by averages over the sampled architectures (steps 14-15). Finally, step 20 returns the architecture with the best validation performance (that also satisfies the latency requirement). Finally, Algorithm 3 shows the whole training procedure for TREFE.

Algorithm 2 Search for a sub-network.

1: $\boldsymbol{\alpha}^* \leftarrow \emptyset, \text{perf}^* \leftarrow -\inf$;
2: **for** iteration $t = 1$ **to** T **do**
3: $\quad j = 0, \mathbf{g} = 0, \mathbf{F}(\boldsymbol{\theta}) \leftarrow \mathbf{0}$;
4: $\quad$ **while** $j < B$ **do**
5: $\quad\quad$ sample $\boldsymbol{\alpha} \sim P_{\boldsymbol{\theta}}\left(\Phi\right)$;
6: $\quad\quad$ obtain network weight $\mathbf{w}^* \leftarrow \mathbf{W}^*(\boldsymbol{\alpha})$;
7: $\quad\quad$ $r \leftarrow \text{latency}\big(\mathcal{N}(\mathbf{w}^*, S, C, R); \mathcal{E}\big)$;
8: $\quad\quad$ $\text{perf} \leftarrow \mathcal{A}_{\text{val}}(\mathcal{N}(\mathbf{w}^*, S, C, R))$;
9: $\quad\quad$ **if** $r \leq r_{\max}$ **then**
10: $\quad\quad\quad$ **if** $\text{perf} > \text{perf}^*$ **then**
11: $\quad\quad\quad\quad$ $(\boldsymbol{\alpha}^*, \text{perf}^*) \leftarrow (\boldsymbol{\alpha}, \text{perf})$;
12: $\quad\quad\quad$ **end if**
13: $\quad\quad$ **end if**
14: $\quad\quad$ $\mathbf{F}(\boldsymbol{\theta}) \leftarrow \big(j\mathbf{F}(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \ln P_{\boldsymbol{\theta}}(\boldsymbol{\alpha}) \left[\nabla_{\boldsymbol{\theta}} \ln P_{\boldsymbol{\theta}}(\boldsymbol{\alpha})\right]^{\top}\big)/(j+1)$;
15: $\quad\quad$ $\mathbf{g} \leftarrow \big(j\mathbf{g} + \mathcal{A}_{\text{val}}(\mathcal{N}(\mathbf{W}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}))\nabla_{\boldsymbol{\theta}} \ln\left(P_{\boldsymbol{\theta}}\left(\boldsymbol{\alpha}\right)\right)\big)/(j+1)$;
16: $\quad\quad$ $j \leftarrow j + 1$;
17: $\quad$ **end while**
18: $\quad$ update $\boldsymbol{\theta}$ via (9) using (10) and (11);
19: **end for**
20: **return** searched architecture $\boldsymbol{\alpha}^*$.

Algorithm 3 Text REcognition Feature Extractor (TREFE).

1: Build a supernet Φ (see Section 3.2.1);
2: Train Φ progressively on training data via Algorithm 1;
3: Search $\boldsymbol{\alpha}^*$ from Φ on validation data via Algorithm 2;
4: Re-train the $\boldsymbol{\alpha}^*$ from scratch;

3.3 Comparison with AutoSTR

There are several differences between AutoSTR [16] and the proposed TREFE (Table 2):

1) Search algorithm: AutoSTR only searches the spatial model, and the downsampling paths and operators are searched separately (using grid search and ProxylessNAS, respectively). On the other hand, TREFE jointly searches both the spatial and sequential models (including the downsampling paths, operators and transformer architecture).
2) To make AutoSTR efficient, only 10 types of downsampling paths are allowed during the search (step 1 in Section 3 of [16]). On the other hand, by using the progressive training strategy, TREFE can efficiently explore all possible downsampling paths by sharing weights in a supernet. Thus, combined with the joint search process, TREFE can find better spatial and sequential models than AutoSTR.
3) As in ASTER [3], AutoSTR uses a sequential attention-based sequence-to-sequence decoder [66] as the recognition head. As characters are output one-by-one, this is not latency-friendly, especially when the output text sequence is long (as in handwritten text). In contrast, TREFE outputs the text sequence in parallel by using parallel attention [6], [42] (resp. CTC head [8]) as the recognition head for scene (resp. handwritten) text.
4) In TREFE, architectures not meeting the resource constraints are dropped during search (Algorithm 2). Thus, TREFE is deployment-aware.

As will be shown empirically in Sections 4.3.2 and 4.5, the above contribute to performance improvements over AutoSTR.

4 Experiments

In this section, we demonstrate the effectiveness of TREFE on long text (i.e., line level) [38] and short text (i.e., word level) [38] recognition by performing extensive experiments on handwritten and scene text TR datasets.

4.1 Setup

Handwritten TR Datasets. The following datasets are used:

1) **IAM** [67]: This dataset contains English handwritten text passages. The training set contains 6482 lines from 747 documents. The validation set contains 976 lines from

116 documents, and the test set contains 2915 lines from 336 documents. The number of characters is 80.
2) **RIMES** [68]: This contains French handwritten mail text from 1300 different writers. The training set contains 10532 lines from 1400 pages. The validation set contains 801 lines from 100 pages, and the test set contains 778 lines from 100 pages. The number of characters is 100.

The input image is of size 64×1200. Moreover, image augmentation is used as in [38].

Scene TR Datasets. Following [3], [6], the task is to recognize all 36 case-insensitive alphanumeric characters from the scene text. Both synthetic datasets and real scene image datasets are used. The input image is resized to 64×256. The two synthetic datasets are:

1) SynthText (**ST**) [69]: This contains 9 million text images generated from a lexicon of 90k common English words. Words are rendered to images using various background and image transformations.
2) MJSynth (**MJ**) [70]: This contains 6 million text images cropped from 800,000 synthetic natural images with ground-truth word bounding boxes.

The real scene image datasets include:

1) IIIT 5K-Words (**IIIT5K**) [71]: This contains 5,000 cropped word images collected from the web. 2,000 images are used for training and the remaining 3,000 for testing.
2) Street View Text (**SVT**) [72]: This is harvested from Google Street View. The training set contains 257 word images, and the test set contains 647 word images. It exhibits high variability and the images often have low resolution.
3) ICDAR 2003 (**IC03**) [73]: This contains 251 full-scene text images. It contains 1,156 training images. Following [72], we discard test images with non-alphanumeric characters or have fewer than three characters. As in [9], two versions are used for testing: one with 867 images, and the other has 860.
4) ICDAR 2013 (**IC13**) [74]: This contains 848 training images. Two versions are used for testing: one has 857 images, and the other has 1,015.
5) ICDAR 2015 (**IC15**): This is from the 4th Challenge in the ICDAR 2015 Robust Reading Competition [75]. The data are collected via Google glasses without careful positioning and focusing. As a result, there are a lot of blurred images in multiple orientations. The training set has 4468 images. Two versions of testing datasets are used: one has 1,811 images and the other has 2,077.
6) SVT-Perspective (**SVTP**): This is used in [76] for the evaluation of perspective text recognition performance. Samples are selected from side-view images in Google Street View, and so many of them are heavily deformed by perspective distortion. It contains 645 test images.

As in [9], we train the model using the two synthetic datasets. The validation data is formed by combining the training sets of IC13, IC15, IIIT5K, and SVT. The model is then evaluated on the test sets of the real scene image datasets without fine-tuning.

Performance Evaluation. As in [33], [38], the following measures are used for performance evaluation on the handwritten text datasets:

1) Character Error Rate (CER) $= \sum_{i=1}^{G} edit(y_i, \hat{y}_i) \ / \ \sum_{j=1}^{G} length(y_i)$, where G is the dataset size, y_i is the ground-truth text, $\hat{y}_i$ is the predicted text, and *edit* is the Levenshtein distance [77];
2) Word Error Rate (WER): defined in the same manner as CER, but at word level instead of character level.

For the scene text datasets, following [3], [14], [37], [38], we use word accuracy for performance evaluation. Moreover, we also report the speed by measuring network latency on a NVIDIA 3090 GPU device with the TensorRT library[4]. Specifically, following [19], we use artificial images as input and perform inference on the TR network 1000 times. The average time is recorded as network latency.

Implementation Details. For scene TR, a Spatial Transformer Network for rectification [3] is used for preprocessing. No extra pre-processing network is used for handwritten TR. The TR head is based on CTC [8] for handwritten TR, and parallel attention [6] for scene TR. The proposed method is developed under the PyTorch framework. We deploy models via TensorRT for high-performance inference and measure the network latency with the FP32 (32-bit floating point computation) mode as in [19].

We use 20 MBConv layers for the spatial model and 4 transformer layers for the sequential model. To reduce computational complexity, the supernet starts with a fixed "stem" layer that reduces the spatial resolution with stride 2×2. In Algorithm 1, we set $K = 5$ and train each block for 300 epochs on the IAM training set. For the larger scene text datasets, we train each block for 1 epoch. Following [3], [42], we use ADADELTA [79] with cosine learning rate as optimizer. The initial learning rate is 0.8, and a weight decay of 10^{-5}. The batch size is 64 for handwritten TR, and 256 for scene TR. The entire architecture search optimization takes about 3 days for handwritten TR, and 5 days for scene TR.

Before evaluating the obtained architecture on a target dataset, we first retrain the whole TR system from scratch. Using the feature extractor architecture obtained from the search procedure, the TR system is optimized by the ADADELTA optimizer with weight decay of 10^{-5}. A cosine schedule is used to anneal the learning rate from 0.8 to 0. We train the network for 1000 (resp. 6) epochs with a batch size of 64 (resp. 560) for handwritten (resp. scene) TR.

4.2 Comparison with the State-of-the-Arts

In this section, we compare TREFE with the state-of-the-art methods on handwritten text and scene TR. For simplicity, we do not use any lexicon or language model.

4.2.1 Scene Text Recognition

In this experiment, we compare with the following state-of-the-arts: (i) AON [78], which extracts directional features to boost recognition; (ii) EP [60], which uses edit-distance-based sequence modeling; (iii) SAR [17], which introduces 2D attention; (iv) ASTER [3], which uses a rectification network for irregular-sized images; (v) SCRN [14], which improves rectification with text shape description and explicit symmetric constraints; (vi) ESIR [37], which iterates image rectification; (vii) AutoSTR [16], which is the method

4. https://developer.nvidia.com/tensorrt

TABLE 3
Comparison with the state-of-the-arts on the scene text datasets. The number under the dataset name is the corresponding number of test samples. Word accuracies for the baselines are copied from the respective papers ('-' means that the corresponding result is not unavailable). The best result is in bold and the second-best is underlined.

	word accuracy									latency (ms)
	IIIT5K	SVT	IC03		IC13		IC15		SVTP	
	3000	647	860	867	857	1015	1811	2077	645	
AON [78]	87.0	82.8	-	91.5	-	-	-	68.2	73.0	-
EP [60]	88.3	87.5	-	**94.6**	-	94.4	-	73.9	-	-
SAR [17]	91.5	84.5	-	-	-	91.0	69.2	-	76.4	4.58
ESIR [37]	93.3	90.2	-	-	91.3	-	-	76.9	79.6	-
ASTER [3]	93.4	89.5	94.5	-	-	91.8	-	76.1	78.5	3.18
SCRN [14]	94.4	88.9	95.0		-	93.9	-	**80.8**	78.7	-
DAN [38]	93.3	88.4	**95.2**	-	94.2	-	-	71.8	76.8	2.92
SRN [42]	94.8	91.5	-	-	**95.5**	-	82.7	-	**85.1**	3.11
TextScanner [7]	93.9	90.1	-	-	-	92.9	-	79.4	83.7	-
RobustScanner [6]	**95.3**	88.1	-	-	-	**94.8**	-	77.1	79.5	4.17
PREN [34]	92.1	**92.0**	94.9	-	94.7	-	-	79.2	83.9	3.75
AutoSTR [16]	94.7	90.9	93.3	-	94.2	-	81.7	-	81.8	3.86
TREFE	94.8	91.3	93.7	93.4	95.4	93.0	**84.0**	80.2	84.5	**2.62**

proposed in the conference version of this paper; (viii) DAN [38], which decouples alignment attention with historical decoding; (ix) SRN [42], which uses a faster parallel decoding and semantic reasoning block; (x) TextScanner [7], which is based on segmentation and uses a mutual-supervision branch to more accurately locate the characters; (xi) RobustScanner [6], which dynamically fuses a hybrid branch and a position enhancement branch; (xii) PREN [34], which learns a primitive representation using pooling and weighted aggregator. Both SCRN and TextScanner also use character box annotations.

Table 3 shows the results. As can be seen, TREFE has comparable recognition performance and the lowest latency. This demonstrates the effectiveness and efficiency of TREFE.

4.2.2 Handwritten Text Recognition

In this experiment, we compare TREFE with the following state-of-the-arts: (i) Bluche et al. [80], which uses a deep architecture with multidimensional LSTM to extract features for text recognition; (ii) Sueiras et al. [81], which extracts image patches and then decodes characters via a sequence-to-sequence architecture with the addition of a convolutional network; (iii) Chowdhury et al. [82], which proposes an attention-based sequence-to-sequence network; (iv) Bhunia et al. [83], which uses an adversarial feature deformation module that learns to elastically warp the extracted features; (v) Zhang et al. [84], which uses a sequence-to-sequence domain adaptation network to handle various handwriting styles; (vi) Fogel et al. [85], which generates handwritten text images using a generative adversarial network (GAN); (vii) Wang et al. [38], which alleviates the alignment problem in the attention mechanism of sequence-to-sequence text recognition models; (viii) Coquenet et al. [13], which replaces the sequential model with lightweight, parallel convolutional networks; and (ix) Yousef et al. [33], which does not use a sequential model but instead applies convolutional blocks with a gating mechanism; (x) Shi et al. [8], which uses VGG as the spatial model and BiLSTM as the sequential model, and (xi) AutoSTR [37]. We do not compare with STR-NAS [15] (which is concurrent with an earlier conference version [16] of TREFE) as its reported performance is significantly worse.

TABLE 4
Comparison with the state-of-the-arts on the IAM dataset. The best result is in bold and the second-best is underlined.

	WER (%)	CER (%)	latency(ms)
Bluche et al. [80]	24.60	7.90	-
Sueiras et al. [81]	23.80	8.80	-
Chowdhury et al. [82]	16.70	8.10	8.71
Bhunia et al. [83]	17.19	8.41	-
Zhang et al. [84]	22.20	8.50	10.52
Fogel et al. [85]	23.61	-	12.16
Wang et al. [38]	20.60	7.00	7.64
Coquenet et al. [13]	28.61	7.99	**2.08**
Yousef et al. [33]	-	4.76	21.48
Shi et al. [8]	21.67	6.28	4.71
AutoSTR [16]	45.23	26.24	11.42
TREFE	**16.41**	**4.45**	2.85

Tables 4 and 5 show results on the IAM and RIMES datasets, respectively. For the baselines, their WER's and CER's are copied from the respective papers[5], while their latencies are obtained by measurements on our reimplementations.[6] Note that these two datasets have different number of characters, [7] thus the latency for IAM and RIMES are different. As can be seen, AutoSTR cannot obtain good architecture and its latency is large. The architecture obtained by TREFE has the best WER and CER performance. While the method in Coquenet et al. [13] has the lowest latency, the TREFE model has much lower error rates.

4.3 Understanding Architectures Obtained by TREFE

In this section, we provide a closer look at the architectures obtained by the proposed TREFE.

4.3.1 Feature Extractors

Table 6 (left) shows the spatial and sequential model architectures ($\mathcal{N}^*_{\text{spa}}$ and $\mathcal{N}^*_{\text{seq}}$, respectively) obtained by TREFE on

5. Note that Yousef et al. [33] does not report the WER.

6. We do not report latency results for Bluche et al. [80], Sueiras et al. [81] and Bhunia et al. [83], as some implementation details are missing.

7. Methods in Zhang et al. [84] and Yousef et al. [33] do not have results on RIMES, while Wang et al. [38] only report results on an ensemble.

TABLE 5
Comparison with the state-of-the-arts on the RIMES dataset. The best result is in bold and the second-best is underlined.

	WER(%)	CER(%)	latency(ms)
Bluche et al. [80]	12.60	<u>2.90</u>	-
Sueiras et al. [81]	15.90	4.80	-
Chowdhury et al. [82]	<u>9.60</u>	3.50	9.11
Bhunia et al. [83]	10.47	6.44	-
Fogel et al. [85]	11.32	-	12.38
Coquenet et al. [13]	18.01	4.35	**2.09**
Shi et al. [8]	11.15	3.40	4.73
AutoSTR [16]	20.40	11.31	12.31
TREFE	**9.16**	**2.75**	<u>2.86</u>

the IAM dataset. For the first half of $\mathcal{N}^*_{\text{spa}}$ (layers 1 to 10), the resolution is relatively high and is only slowly reduced by a total factor of 4. For the second half of $\mathcal{N}^*_{\text{spa}}$ (layers 10 to 20), the resolution is reduced more rapidly by a total factor of 32. A similar observation can also be made on the architecture obtained on the scene text dataset (Table 6 (right)). We speculate that a larger resolution can help the network to preserve more spatial information. This is also consistent with some manually-designed network architectures. For example, in SCRN [14], a ResNet50 with FPN [86] is first used to extract features from text images. This is followed by a few convolutional layers which downsample the feature map resolution rapidly by a factor of 32. This observation may inspire better designs of the text image feature extractor in the future.

4.3.2 Varying the Resource Constraints

In this section, we demonstrate the deployment-aware ability of TREFE by performing experiments with different latency constraints on the IAM dataset. These include: (i) no latency constraint, (ii) reduce the runtime to $5/6$ of that of the architecture obtained under no latency constraint, and, (iii) reduce the runtime to $2/3$ of that of the architecture obtained under no latency constraint. We compare TREFE with random search, which is often a strong baseline in NAS [31]. Specifically, we randomly pick 6 architectures from the search space that satisfy the required latency constraint, and then train them from scratch using the same setting as in Section 4.1.

Table 7 shows the performance of models obtained by TREFE and random search. As can be seen, TREFE obtains architectures with better WER and CER. Though the TREFE models have higher latencies, the gap with random search closes rapidly as the target latency is reduced.

4.3.3 Varying the Spatial Model Design

Here, we compare the spatial model $\mathcal{N}^*_{\text{spa}}$ obtained by TREFE with the popular hand-designed architectures of VGG [11] (used in CRNN [8]) and ResNet [12] (used in ASTER [3]). We also compare with the spatial model obtained by DARTS [22], a representative NAS method. To ensure that the constraint in (6) is satisfied by the DARTS model, we replace the basic block in ASTER with a CNN cell from DARTS, which is searched on the image classification task using the ImageNet dataset. All architectures are trained and evaluated under the same settings as in Section 4.1. The experiment is performed on the IAM, RIMES and IIIT5K datasets. Note that random search is not compared, since it has been shown to be less effective than TREFE in Section 4.3.2.

Table 8 shows the results. As can be seen, TREFE (i.e., $\mathcal{N}^*_{\text{spa}} + \mathcal{N}^*_{\text{seq}}$) outperforms the other baselines on all datasets. The performance of DARTS is even worse than the hand-designed architectures. This demonstrates that ignoring the domain knowledge and directly reusing structures obtained by a state-of-the-art NAS method may not be good.

4.3.4 Varying the Sequential Model Design

In this experiment, we compare the sequential model $\mathcal{N}^*_{\text{seq}}$ obtained by TREFE with the (i) hand-designed BiLSTM in [18], which is a strong baseline for sequential context modeling in current text recognition systems [3], [7], [38], and (ii) vanilla Transformer [27]. We also compare with the sequential models obtained by two NAS methods: (i) the recurrent cell obtained by DARTS [22] on the Penn Treebank; and (ii) the evolved Transformer [58], which is obtrained on the WMT'14 En-De translation task [87]. In these baselines, we keep the spatial model $\mathcal{N}^*_{\text{spa}}$ in TREFE, but replace its sequential model $\mathcal{N}^*_{\text{seq}}$ with the models in these baselines. The resultant architectures are trained (from scratch) and evaluated under the same settings as in Section 4.1. The experiment is again performed on the IAM, RIMES and IIIT5K datasets.

Table 9 shows the results. As can be seen, TREFE (i.e., $\mathcal{N}^*_{\text{spa}} + \mathcal{N}^*_{\text{seq}}$) consistently outperforms all the baselines, including the transformer (VT). Thus, architecture search of the sequential model is necessary.

4.4 Understanding the Search Process

4.4.1 Supernet Training

In this section, we compare the proposed supernet training strategy (in Section 3.2.2) with the following strategies:

1) SPOS [20]: a widely-used one-shot NAS method which uniformly samples and updates $\boldsymbol{\alpha}$ from the supernet;
2) Random path: The proposed progressive supernet training pipeline (Algorithm 1), which randomly selects $\boldsymbol{\alpha}^{\leftarrow}_k$ and $\boldsymbol{\alpha}_k$;
3) Best path: This is based on the proposed procedure, but instead of random sampling a path $\boldsymbol{\alpha}^{\leftarrow}_k$ from the trained $\Phi_1, \ldots, \Phi_{k-1}$, it picks the $\boldsymbol{\alpha}^{\leftarrow}_k$ with the best validation performance (details are in Appendix B);
4) Co-update: This is also based on the proposed procedure, but instead of fixing the weights of $\boldsymbol{\alpha}^{\leftarrow}_k$, it updates them together with $\mathbf{W}_k(\boldsymbol{\alpha}_k)$.

For SPOS, we train the whole supernet for 1500 epochs. To have a fair comparison, for TREFE and its variations ((2), (3) and (4)), we first divide the supernet into 5 blocks and then train each block for 300 epochs. The total number of training epochs for all methods are thus the same.

Training. Figure 5 shows the training losses for the various supernet training strategies on the IAM dataset. As can be seen, SPOS is difficult to converge, while TREFE and its variants show good training convergence.

Table 11 shows the training cost of the supernet. As can be seen, "random path" and "best path" have lower training time than "SPOS" and "co-update", as only parts of the selected path $\boldsymbol{\alpha}_k$ need to be updated.

TABLE 6
Architectures obtained on the IAM dataset (left) and scene text dataset (right). ✓ (*resp.* ×) means to follow alternative (*resp.* original) choices in Section 3.1.2.

	layer	operator	resolution	#channels
$\mathcal{N}^*_{\text{spa}}$	stem	Conv(k:3)-BN-ReLU	[32, 600]	16
	1	MBConv(k:5,e:6)	[32, 600]	16
	2	MBConv(k:5,e:6)	[32, 600]	16
	3	MBConv(k:5,e:6)	[16, 600]	16
	4	MBConv(k:5,e:6)	[16, 600]	16
	5	MBConv(k:5,e:6)	[16, 600]	16
	6	MBConv(k:5,e:6)	[16, 600]	16
	7	MBConv(k:5,e:6)	[16, 600]	16
	8	MBConv(k:3,e:1)	[16, 600]	16
	9	MBConv(k:5,e:6)	[8, 600]	32
	10	MBConv(k:5,e:6)	[8, 600]	32
	11	MBConv(k:5,e:6)	[8, 600]	32
	12	MBConv(k:5,e:6)	[8, 600]	32
	13	MBConv(k:5,e:1)	[4, 600]	64
	14	MBConv(k:5,e:1)	[2, 300]	128
	15	MBConv(k:5,e:1)	[2, 300]	128
	16	MBConv(k:5,e:6)	[2, 300]	128
	17	MBConv(k:5,e:1)	[2, 300]	128
	18	MBConv(k:5,e:1)	[2, 300]	128
	19	MBConv(k:5,e:6)	[1, 150]	256
	20	MBConv(k:5,e:6)	[1, 150]	256

	layer	(i):residual	(ii):Rel	(iii):scaling	(iv):FFN	#hidden
$\mathcal{N}^*_{\text{seq}}$	21	×	×	✓	GLU	256
	22	✓	×	✓	MLP	256
	23	×	✓	×	GLU	256
	24	✓	✓	✓	GLU	256

	layer	operator	resolution	#channels
$\mathcal{N}^*_{\text{spa}}$	stem	Conv(k:3)-BN-ReLU	[32, 128]	32
	1	MBConv(k:5,e:6)	[32, 128]	32
	2	MBConv(k:3,e:6)	[16, 128]	32
	3	MBConv(k:5,e:6)	[16, 128]	32
	4	MBConv(k:5,e:6)	[16, 128]	32
	5	MBConv(k:3,e:6)	[16, 128]	32
	6	MBConv(k:3,e:1)	[16, 128]	32
	7	MBConv(k:5,e:6)	[16, 128]	32
	8	MBConv(k:5,e:1)	[16, 128]	32
	9	MBConv(k:5,e:1)	[16, 128]	32
	10	MBConv(k:5,e:6)	[16, 128]	32
	11	MBConv(k:3,e:6)	[8, 64]	64
	12	MBConv(k:3,e:6)	[8, 64]	64
	13	MBConv(k:3,e:6)	[4, 64]	128
	14	MBConv(k:5,e:6)	[4, 64]	128
	15	MBConv(k:5,e:6)	[4, 64]	128
	16	MBConv(k:3,e:6)	[4, 64]	128
	17	MBConv(k:3,e:6)	[4, 64]	128
	18	MBConv(k:5,e:6)	[2, 64]	256
	19	MBConv(k:5,e:6)	[1, 32]	512
	20	MBConv(k:5,e:6)	[1, 32]	512

	layer	(i):residual	(ii):Rel	(iii):scaling	(iv):FFN	#hidden
$\mathcal{N}^*_{\text{seq}}$	21	×	✓	✓	MLP	512
	22	×	✓	✓	MLP	512
	23	✓	×	✓	MLP	512
	24	✓	✓	✓	GLU	512

TABLE 7
Comparison of TREFE and random search under different latency constraints on the IAM dataset.

Latency constraint	TREFE			random search		
	WER (%)	CER (%)	latency (ms)	WER (%)	CER (%)	latency (ms)
(i) no constraint	**16.41**	**4.45**	2.85	17.04	4.68	**2.20**
(ii) reduce latency to 5/6 of unconstrained model	**16.67**	**4.56**	2.37	17.04	4.68	**2.20**
(iii) reduce latency to 2/3 of unconstrained model	**19.18**	**5.19**	1.86	19.39	5.40	**1.84**

TABLE 8
Performance comparison of different spatial model architectures. "VT" is short for Vanilla Transformer, "DARTS" means reusing CNN architecture searched by DARTS [22].

	architecture		IAM			RIMES			IIIT5K	
	spatial	sequential	WER (%)	CER(%)	latency(ms)	WER(%)	CER(%)	latency(ms)	Acc(%)	latency(ms)
hand-designed	ResNet	VT	22.10	6.19	2.00	12.90	3.78	2.01	93.8	2.01
	VGG	$\mathcal{N}^*_{\text{seq}}$	20.80	5.97	**1.21**	11.47	3.38	**1.21**	92.8	1.27
	ResNet	$\mathcal{N}^*_{\text{seq}}$	18.67	5.24	2.19	10.24	3.10	2.21	93.4	2.16
NAS	DARTS	$\mathcal{N}^*_{\text{seq}}$	23.24	7.02	3.81	12.03	3.81	3.92	91.9	3.19
	$\mathcal{N}^*_{\text{spa}}$	$\mathcal{N}^*_{\text{seq}}$	**16.41**	**4.45**	2.85	**9.16**	**2.75**	2.86	**94.8**	**2.62**

TABLE 11
Training costs (in GPU days) of the supernet.

SPOS	Random path	Best path	Co-update
4.0	1.4	1.2	3.0

Ranking Correlation. As in [20], [49], we examine the ranking correlation between the performance of a model trained from scratch (denoted "stand-alone model") and that with weights inherited from the supernet ("one-shot model"). Specifically, for the stand-alone models, we random sample 70 architectures from the proposed search space and train them from scratch for 250 epochs. The weights of the corresponding one-shot models are obtained from the trained supernet and then finetuned on the training set for 5 epochs (with learning rate 0.01). As in [88], the ranking correlation between the stand-alone model's validation set CER and that of the one-shot model is measured by the following three commonly used metrics: (i) Kendall's, (ii) Spearman's ρ, and (iii) Pearson's r. They all have with values in $[-1, 1]$.

Figure 6 shows the correlation plots for the four strategies, and Table 12 shows the ranking correlations. As can be seen, the correlations are the smallest for SPOS, and highest for "random path", demonstrating the advantage of the proposed strategy for sampling and updating $\boldsymbol{\alpha}_k^{\leftarrow}$.

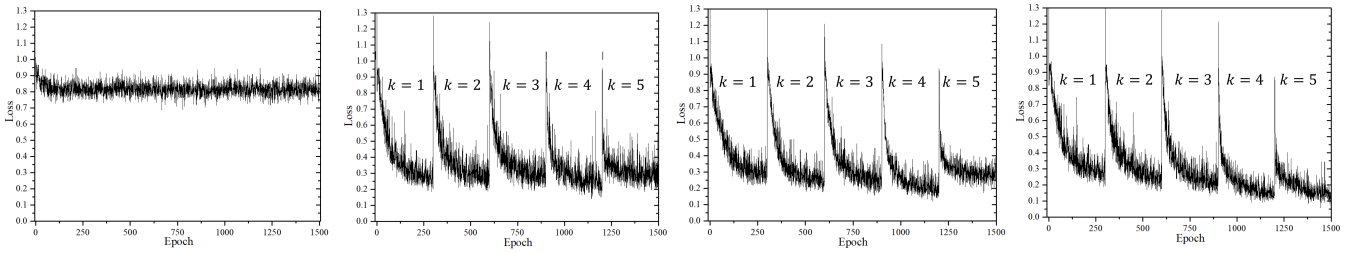
Effect of the Number of Blocks K. In this experiment, we perform ablation study on K. We train the supernet using Algorithm 1 with $K = 3, 5, 7$. The ranking correlation between the validation set CER of the stand-alone and one-

TABLE 9
Performance comparison of different sequential model architectures. "VT" is short for Vanilla Transformer, "Lat" denotes latency. "DARTS" means reusing RNN architecture searched by DARTS [22], and "ET" is the Evolving transformer [58].

	architecture		IAM			RIMES			IIIT5K	
	spatial	sequential	WER(%)	CER(%)	latency(ms)	WER(%)	CER(%)	latency(ms)	Acc(%)	latency(ms)
hand-designed	ResNet	VT	22.10	6.19	2.00	12.90	3.78	2.01	93.8	2.01
	$\mathcal{N}^*_{\text{spa}}$	BiLSTM	18.65	5.01	4.03	10.47	3.20	4.04	94.4	3.26
		VT	19.84	5.29	**2.64**	11.79	3.41	**2.65**	94.3	2.45
NAS	$\mathcal{N}^*_{\text{spa}}$	DARTS	19.42	5.28	28.35	10.97	3.23	28.86	92.8	8.57
		ET	20.27	5.56	3.27	11.77	3.33	3.27	93.1	3.18
		$\mathcal{N}^*_{\text{seq}}$	**16.41**	**4.45**	2.85	**9.16**	**2.75**	2.86	**94.8**	2.62

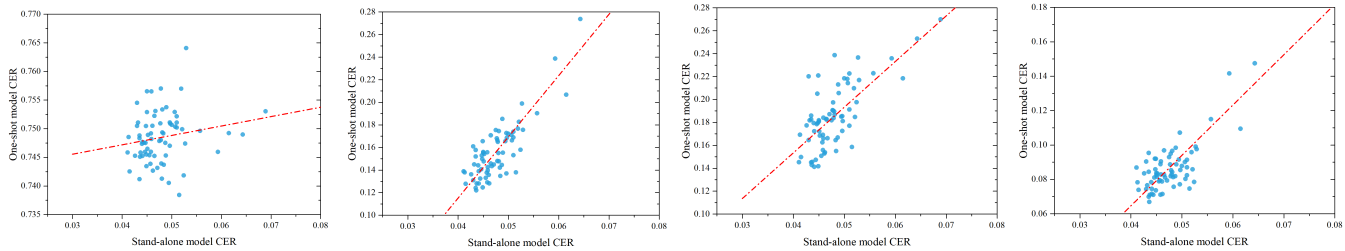
TABLE 10
Comparison of TREFE, AutoSTR and different variants on IAM and RIMES dataset.

	spatial model	sequential model	recogition head	IAM			RIMES		
				WER (%)	CER (%)	latency (ms)	WER (%)	CER (%)	latency (ms)
TREFE	$\mathcal{N}^*_{\text{spa}}$	$\mathcal{N}^*_{\text{seq}}$	CTC	16.41	4.45	2.85	9.16	2.75	2.86
AutoSTR	CNN	BiLSTM	SeqAtt	45.23	26.24	11.42	20.40	11.31	12.31
variant-1	CNN	BiLSTM	CTC	24.09	6.84	3.18	13.54	3.98	3.15
variant-2	CNN	$\mathcal{N}^*_{\text{seq}}$	CTC	19.68	5.49	1.88	10.45	3.18	1.88
variant-3	$\mathcal{N}^*_{\text{spa}}$	BiLSTM	CTC	18.65	5.01	4.03	10.47	3.20	4.04



(a) SPOS. (b) Random path. (c) Best path. (d) Co-update.

Fig. 5. Comparison of the supernet training loss curves for SPOS [20], TREFE (with random path) and its variations (best path and co-update).



(a) SPOS. (b) Random path. (c) Best path. (d) Co-update.

Fig. 6. Correlation plots between the validation CERs of the stand-alone model and one-shot model for SPOS [20], TREFE ("random path") and its variants ("best path" and "co-update"). The red lines is the linear regression fit.

TABLE 12
Ranking correlations for different strategies.

	Kendall's τ	Spearman's ρ	Pearson's r
SPOS	0.143	0.231	0.185
Random path	0.501	0.686	0.871
Best path	0.447	0.598	0.708
Co-update	0.371	0.515	0.806

shot models is shown in Table 14. As can be seen, TREFE is robust to the value of K. Besides, the supernet training cost decreases with K (they are 2.0, 1.4, and 1.3 GPU days, for $K = 3, 5, 7$, respectively). When K increases, each block becomes smaller, and the training time reduction in training smaller blocks is more significant than the larger number of blocks that have to be trained.

4.4.2 Search on Supernet

TREFE uses natural gradient descent (denoted NGD) to optimize the parameters $\boldsymbol{\theta}$ in (7). In this experiment, we compare the efficiency of NGD with random search and evolutionary architecture search in [20] on the IAM dataset. In each search iteration, 16 new architectures are sampled. Figure 7 shows the number of search iterations versus the validation CERs of the best 16 models obtained up to that iteration. As can be seen, NGD and evolutionary search

TABLE 13
Mean and standard deviation of the performance of TREFE with five repetitions.

IAM			RIMES			IIIT5K	
WER (%)	CER (%)	latency (ms)	WER (%)	CER (%)	latency (ms)	Acc (%)	latency (ms)
16.41±0.21	4.45±0.34	2.85±0.12	9.16±0.12	2.75±0.19	2.86±0.27	94.8±0.1	2.62±0.23

TABLE 14
Effect of K on the ranking correlation.

K	Kendall's τ	Spearman's ρ	Pearson's r
3	0.471	0.640	0.850
5	0.501	0.686	0.871
7	0.503	0.683	0.870

clearly outperform random search, and NGD performs the best. In general, evolutionary algorithm can be easily trapped in local minima due to inbreeding, and so the performance of evolutionary search cannot improve after 10 search iterations.

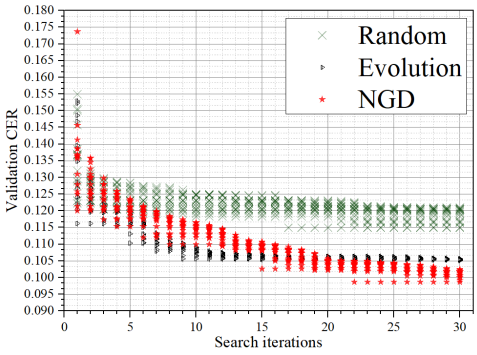


Fig. 7. Validation CER versus the number of search iterations for natural gradient descent (NGD), evolutionary algorithm and random sampling on the IAM dataset.

As sampling is involved in the search, i.e., (10) and (11), we study the variance of the proposed TREFE by running it five times with different random seeds. Table 13 shows the mean and standard deviation of the performance on the IAM, RIMES and IIIT5K datasets. As can be seen, the variance is small in all cases.

4.5 Comparison with AutoSTR

Following the comparison between AutoSTR and TREFE in Section 3.3, in this section we perform an ablation study on AutoSTR, TREFE and different variants. Table 10 shows the comparison on handwritten text recognition using the IAM and RIMES datasets, and Table 15 shows the performance comparison on scene text recognition using the IIIT5K dataset.

- On comparing AutoSTR with variant-1 in Table 10 (resp. Table 15), using sequential attention (SeqAtt) as the recognition head leads to much higher latency and also higher error than the use of CTC (resp. parallel attention) in handwritten (resp. scene) text recognition.
- The improvement of variant-2 over variant-1 shows effectiveness of the searched transformer.
- The improvement of variant-3 over variant-1 shows the effectiveness of the proposed search algorithm for the spatial model.
- Finally, the improvement of TREFE over variant-1 shows the improvement with both the learned spatial and sequential models.

TABLE 15
Comparison of TREFE, AutoSTR and different variants on IIIT5K dataset.

	spatial model	sequential model	recogition head	Acc (%)	latency (ms)
TREFE	$\mathcal{N}^*_{\text{spa}}$	$\mathcal{N}^*_{\text{seq}}$	ParAtt	94.8	2.62
AutoSTR	CNN	BiLSTM	SeqAtt	94.7	3.86
variant-1	CNN	BiLSTM	ParAtt	93.3	2.01
variant-2	CNN	$\mathcal{N}^*_{\text{seq}}$	ParAtt	93.6	1.53
variant-3	$\mathcal{N}^*_{\text{spa}}$	BiLSTM	ParAtt	94.4	3.26

5 Conclusion

In this paper, we propose to find suitable feature extraction for the text recognition (TR) by neural architecture search. We first design a novel search space for the problem, which fully explore the prior from such a domain. Then, we propose a new two-stages-based search algorithm, which can efficiently search feature downsampling paths and convolution types for the spatial model, and transformer layers for the sequential model. Besides, our algorithm can be deployment aware, and finding good architecture within specific latency constraints. Experiments demonstrate that our searched models can greatly improve the capability of the TR pipeline and achieve state-of-the-art results on both handwritten and scene TR benchmarks.

Acknowledgments

This work was supported by the National Natural Science Foundation of China 62225603.

References

[1] S. Long, X. He, and C. Yao, "Scene text detection and recognition: The deep learning era," arXiv preprint arXiv:1811.04256, Tech. Rep., 2018.
[2] Y. Zhu, C. Yao, and X. Bai, "Scene text detection and recognition: Recent advances and future trends," *Frontiers of Computer Science*, 2016.
[3] B. Shi, M. Yang, X. Wang, P. Lyu, C. Yao, and X. Bai, "ASTER: An attentional scene text recognizer with flexible rectification," *TPAMI*, 2018.
[4] W. Wang, E. Xie, X. Liu, W. Wang, D. Liang, C. Shen, and X. Bai, "Scene text image super-resolution in the wild," in *ECCV*. Springer, 2020.
[5] C. Luo, Q. Lin, Y. Liu, L. Jin, and C. Shen, "Separating content from style using adversarial learning for recognizing text in the wild," *IJCV*, 2021.

[6] X. Yue, Z. Kuang, C. Lin, H. Sun, and W. Zhang, "Robustscanner: Dynamically enhancing positional clues for robust text recognition," in *ECCV*, 2020.

[7] Z. Wan, M. He, H. Chen, X. Bai, and C. Yao, "Textscanner: Reading characters in order for robust scene text recognition," in *AAAI*, 2020.

[8] 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," *TPAMI*, 2016.

[9] J. Baek, G. Kim, J. Lee, S. Park, D. Han, S. Yun, S. J. Oh, and H. Lee, "What is wrong with scene text recognition model comparisons? dataset and model analysis," in *ICCV*, 2019.

[10] M. Liao, J. Zhang, Z. Wan, F. Xie, J. Liang, P. Lyu, C. Yao, and X. Bai, "Scene text recognition from two-dimensional perspective," in *AAAI*, 2019.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, Tech. Rep., 2014.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[13] D. Coquenet, C. Chatelain, and T. Paquet, "Recurrence-free unconstrained handwritten text recognition using gated fully convolutional network," in *ICFHR*, 2020.

[14] M. Yang, Y. Guan, M. Liao, X. He, K. Bian, S. Bai, C. Yao, and X. Bai, "Symmetry-constrained rectification network for scene text recognition," in *ICCV*, 2019.

[15] S. Hong, D. Kim, and M.-K. Choi, "Memory-efficient models for scene text recognition via neural architecture search," in *WACV Workshops*, 2020, pp. 183–191.

[16] H. Zhang, Q. Yao, M. Yang, Y. Xu, and X. Bai, "AutoSTR: Efficient backbone search for scene text recognition," in *ECCV*, 2020.

[17] H. Li, P. Wang, C. Shen, and G. Zhang, "Show, attend and read: A simple and strong baseline for irregular text recognition," in *AAAI*, 2019.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *NeComp*, 1997.

[19] W. Chen, X. Gong, X. Liu, Q. Zhang, Y. Li, and Z. Wang, "Fasterseg: Searching for faster real-time semantic segmentation," in *ICLR*, 2020.

[20] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *ECCV*, 2020.

[21] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *JMLR*, 2019.

[22] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *ICLR*, 2019.

[23] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *ICML*, 2018.

[24] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, "Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation," in *CVPR*, 2019.

[25] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, and J. Sun, "DetNAS: Backbone search for object detection," in *NeurIPS*, 2019.

[26] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *ICLR*, 2019.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017.

[28] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *ICLR*, 2020.

[29] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *NIPS*, 2007.

[30] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *NeComp*, 2006.

[31] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *UAI*, 2019.

[32] S.-I. Amari, "Natural gradient works efficiently in learning," *NeComp*, 1998.

[33] M. Yousef and T. E. Bishop, "Origaminet: Weakly-supervised, segmentation-free, one-step, full page textrecognition by learning to unfold," in *CVPR*, 2020.

[34] R. Yan, L. Peng, S. Xiao, and G. Yao, "Primitive representation learning for scene text recognition," in *CVPR*, 2021.

[35] Q. Ye and D. Doermann, "Text detection and recognition in imagery: A survey," *TPAMI*, 2014.

[36] B. Shi, X. Wang, P. Lyu, C. Yao, and X. Bai, "Robust scene text recognition with automatic rectification," in *CVPR*, 2016.

[37] F. Zhan and S. Lu, "ESIR: End-to-end scene text recognition via iterative image rectification," in *CVPR*, 2019.

[38] T. Wang, Y. Zhu, L. Jin, C. Luo, X. Chen, Y. Wu, Q. Wang, and M. Cai, "Decoupled attention network for text recognition." in *AAAI*, 2020.

[39] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *ICML*, 2006.

[40] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014.

[41] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *EMNLP*. The Association for Computational Linguistics, 2015.

[42] D. Yu, X. Li, C. Zhang, T. Liu, J. Han, J. Liu, and E. Ding, "Towards accurate scene text recognition with semantic reasoning networks," in *CVPR*, 2020.

[43] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.

[44] L. Xie and A. Yuille, "Genetic CNN," in *ICCV*, 2017.

[45] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *ICML*. PMLR, 2018, pp. 550–559.

[46] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *ICLR*, 2018.

[47] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *ECCV*, 2018.

[48] X. Chu, B. Zhang, R. Xu, and J. Li, "Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search," in *ICCV*, 2021.

[49] C. Li, T. Tang, G. Wang, J. Peng, B. Wang, X. Liang, and X. Chang, "Blockwisely supervised neural architecture search with knowledge distillation," in *CVPR*, 2020.

[50] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar *et al.*, "Bootstrap your own latent: A new approach to self-supervised learning," in *NeurIPS*, 2020.

[51] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MNasNet: Platform-aware neural architecture search for mobile," in *CVPR*, 2019, pp. 2820–2828.

[52] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *ICCV*, 2019, pp. 1314–1324.

[53] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *CVPR*, 2019.

[54] R. He, A. Ravula, B. Kanagal, and J. Ainslie, "Realformer: Transformer likes residual attention," in *ACL Findings*, 2021.

[55] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," in *NAACL*, 2019.

[56] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *ICLR*, 2020.

[57] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser *et al.*, "Rethinking attention with performers," in *ICLR*, 2021.

[58] D. So, C. Liang, and Q. V. Le, "The evolved transformer," in *ICML*, 2019.

[59] Z. Cheng, F. Bai, Y. Xu, G. Zheng, S. Pu, and S. Zhou, "Focusing attention: Towards accurate text recognition in natural images," in *CVPR*, 2017.

[60] F. Bai, Z. Cheng, Y. Niu, S. Pu, and S. Zhou, "Edit probability for scene text recognition," in *CVPR*, 2018.

[61] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018.

[62] H. Yan, B. Deng, X. Li, and X. Qiu, "Tener: Adapting transformer encoder for named entity recognition," *arXiv preprint arXiv:1911.04474*, 2019.

[63] Q. Guo, X. Qiu, P. Liu, Y. Shao, X. Xue, and Z. Zhang, "Star-transformer," in *NAACL*, 2019.

[64] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *ICML*, 2017.

[65] Y. Leung, Y. Gao, and Z. Xu, "Degree of population diversity - a perspective on premature convergence in genetic algorithms and its markov chain analysis," *TNNLS*, 1997.

[66] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *International Conference on Learning Representations*, 2015.

[67] U. Marti and H. Bunke, "The IAM-database: an english sentence database for offline handwriting recognition," *IJDAR*, 2002.

[68] E. Grosicki and H. El-Abed, "Icdar 2011-french handwriting recognition competition," in *ICDAR*, 2011.

[69] Y. Liu, Z. Wang, H. Jin, and I. Wassell, "Synthetically supervised feature learning for scene text recognition," in *ECCV*, 2018.

[70] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," arXiv preprint arXiv:1406.2227, Tech. Rep., 2014.

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

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

[73] S. M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, R. Young, K. Ashida, H. Nagai, M. Okamoto, and H. Yamamoto, "Icdar 2003 robust reading competitions: entries, results, and future directions," *IJDAR*, 2005.

[74] D. Karatzas, F. Shafait, S. Uchida, M. Iwamura, L. G. i Bigorda, S. R. Mestre, J. Mas, D. F. Mota, J. A. Almazan, and L. P. De Las Heras, "Icdar 2013 robust reading competition," in *ICDAR*, 2013.

[75] D. Karatzas, L. Gomez-Bigorda, A. Nicolaou, S. Ghosh, A. Bagdanov, M. Iwamura, J. Matas, L. Neumann, V. R. Chandrasekhar, S. Lu *et al.*, "Icdar 2015 competition on robust reading," in *ICDAR*, 2015.

[76] T. Quy Phan, P. Shivakumara, S. Tian, and C. Lim Tan, "Recognizing text with perspective distortion in natural scenes," in *CVPR*, 2013.

[77] V. I. Levenshtein *et al.*, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, 1966.

[78] Z. Cheng, Y. Xu, F. Bai, Y. Niu, S. Pu, and S. Zhou, "Aon: Towards arbitrarily-oriented text recognition," in *CVPR*, 2018.

[79] M. Zeiler, "Adadelta: an adaptive learning rate method," arXiv preprint arXiv:1212.5701, Tech. Rep., 2012.

[80] T. Bluche, "Joint line segmentation and transcription for end-to-end handwritten paragraph recognition," in *NIPS*, 2016.

[81] J. Sueiras, V. Ruiz, A. Sanchez, and J. F. Velez, "Offline continuous handwriting recognition using sequence to sequence neural networks," *NeComp*, 2018.

[82] A. Chowdhury and L. Vig, "An efficient end-to-end neural model for handwritten text recognition," in *BMVC*, 2018.

[83] A. K. Bhunia, A. Das, A. K. Bhunia, P. S. R. Kishore, and P. P. Roy, "Handwriting recognition in low-resource scripts using adversarial learning," in *CVPR*, 2019.

[84] Y. Zhang, S. Nie, W. Liu, X. Xu, D. Zhang, and H. T. Shen, "Sequence-to-sequence domain adaptation network for robust text image recognition," in *CVPR*, 2019.

[85] S. Fogel, H. Averbuch-Elor, S. Cohen, S. Mazor, and R. Litman, "Scrabblegan: semi-supervised varying length handwritten text generation," in *CVPR*, 2020, pp. 4324–4333.

[86] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *CVPR*, 2017.

[87] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand *et al.*, "Findings of the 2014 workshop on statistical machine translation," in *Workshop on Statistical Machine Translation*, 2014, pp. 12–58.

[88] C. Li, T. Tang, G. Wang, J. Peng, B. Wang, X. Liang, and X. Chang, "BossNAS: Exploring hybrid CNN-transformers with block-wisely self-supervised neural architecture search," in *ICCV*, 2021.

Hui Zhang is currently a research engineer in 4Paradigm Inc., Beijing, China. He received a B.S. degree in Computer Science and Technology department from the HuaZhong University of Science and Technology (HUST), Wuhan, China, in 2018, and the M.S. degree in Electronics and Information Engineering also from HUST in 2020. His research interest is computer vision algorithms and systems.

Quanming Yao (member, IEEE) is a tenure-track assistant professor in the Department of Electronic Engineering, Tsinghua University. He was a senior scientist in 4Paradigm, who is also the founding leader of the company's machine learning research team. He obtained his Ph.D. degree at the Department of Computer Science and Engineering of Hong Kong University of Science and Technology (HKUST). His research interests are in machine learning, graph neural networks, and automated machine learning. He is a receipt of Forbes 30 Under 30 (China), Young Scientist Awards (issued by Hong Kong Institution of Science), Wuwen Jun Prize for Excellence Youth of Artificial Intelligence (issued by CAAI), and a winner of Google Fellowship (in machine learning).

James T. Kwok (Fellow, IEEE) received the Ph.D. degree in computer science from The Hong Kong University of Science and Technology in 1996. He is a Professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include machine learning, deep learning, and artificial intelligence. He received the IEEE Outstanding 2004 Paper Award and the Second Class Award in Natural Sciences by the Ministry of Education, China, in 2008. He is serving as an Associate Editor for the IEEE Transactions on Neural Networks and Learning Systems, Neural Networks, Neurocomputing, Artificial Intelligence Journal, International Journal of Data Science and Analytics, Editorial Board Member of Machine Learning, Board Member, and Vice President for Publications of the Asia Pacific Neural Network Society. He also served/is serving as Senior Area Chairs / Area Chairs of major machine learning / AI conferences including NIPS, ICML, ICLR, IJCAI, AAAI and ECML.

Xiang Bai (Senior Member, IEEE) received his B.S., M.S., and Ph.D. degrees from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2003, 2005, and 2009, respectively, all in electronics and information engineering. He is currently a Professor with the School of Artificial Intelligence and Automation, HUST. He is also the Vice-director of the National Center of AntiCounterfeiting Technology, HUST. His research interests include object recognition, shape analysis, scene text recognition and intelligent systems. He serves as an associate editor for Pattern Recognition , Pattern Recognition Letters, Neurocomputing and Frontiers of Computer Science.