Abstract

Segmentation-based methods are widely used for scene text detection due to their superiority in describing arbitrary-shaped text instances. However, two major problems still exist: 1) current label generation techniques are mostly empirical and lack theoretical support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inference support, discouraging elaborate label design; 2) as a result, most methods rely heavily on text kernel segmentation which is unstable and requires deliberate tuning. To address these challenges, we propose a human cognition-inspired framework, termed, Conceptual Text Region Network (CTRNet). The framework utilizes Conceptual Text Regions (CTRs), which is a class of cognition-based tools inheriting good mathematical properties, allowing for sophisticated label design. Another component of CTRNet is an inferen...
In summary, our contributions are threefold: 1) We propose CTRs as a class of well-designed label generation tools which inhered good properties such as bijectivity; 2) We propose a segmentation network and a post-processing algorithm that are capable of learning and processing geometric features, omitting the need for text kernel segmentation while establishing an accurate and interpretable inference pipeline. 3) Our CTRNet achieves state-of-the-art performances on a comprehensive set of benchmarks, showing great accuracy and consistency.

2. Related Work

Deep learning-based text detection methods have achieved outstanding results over the past few years. The majority of these methods utilize CNNs and can be roughly categorized into anchor-based methods and segmentation-based methods. In addition, it is of interest that text detection methods in general adopt specific and tailored label generation and post-processing methodologies to utilize the geometric nature of text instances. Thus, we also introduce the label generation and post-processing techniques adopted by text detection methods.

2.1. Anchor-Based Methods

Anchor-based methods are often based on popular object detection frameworks such as Faster R-CNN [1] and SSD [2].

TextBoxes++ [22] modifies anchor boxes and convolutional kernels of SSD to handle the unique aspect ratios of text instances. TextBoxes++ further introduces quadrangle regression to allow for multi-oriented text detection. RRD [24] extracts rotation-invariant and rotation-sensitive features for text classification and regression respectively, eliminating the incompatibility between these two tasks when detecting multi-oriented text. RRPN [24] based on Faster R-CNN, developed Rotation Region Proposal Networks to detect multi-oriented text instances. SPCNet [26] and Mask Text Spotter [27] view text detection as an instance segmentation problem and utilize Mask R-CNN [28] for arbitrary-shaped text detection. More recent anchor-based methods, such as Boundary [29], use Region Proposal Networks [11] and regression methods to obtain the boundary points of text instances and later rectify the detection results with differentiable operations such as Thin-Plate Splines (TPS) [30]. These structures include text recognition and enable the model to train in an end-to-end manner.

Anchor-based methods normally require few post-processing steps and perform reasonably well when handling multi-oriented text. However, most of them rely on complex multiple stages and hand-crafted anchor settings, which makes these approaches overcomplicated and less effective when handling long text instances.

2.2. Segmentation-Based Methods

Segmentation-based methods mainly view text detection as a segmentation problem and utilize Fully Convolutional Networks (FCNs) [3] or their variants. Zhang et al. [31] first employ FCN to extract text blocks and apply MSER to detect character candidates from the text blocks. Yao et al. [32] adopt FCN to predict multiple properties of text instances, such as text regions and orientations, then implement a clustering algorithm to obtain the detection result. PixelLink [33] performs link prediction to separate adjacent text regions. EAST [9] and DeepReg [34] detect the bounding boxes of words in a per-pixel fashion without using anchors or proposal networks. Newer segmentation-based approaches, such as TextSnake [10], PSENet [11], and PAN [13], detect arbitrary-shaped text instances through similar pipelines that involve text kernel segmentation and text reconstruction. Similar to anchor-based methods, there are also attempts to train segmentation-based models in an end-to-end manner. Text Perceptron [35], for example, adopts an FPN to predict fiducial points for TPS transformations.

Segmentation-based models are generally suitable for arbitrary-shaped text detection thanks to their flexibility. However, current approaches tend to endure less ideal label generation techniques and rely heavily on text kernel segmentation, which is unstable and requires deliberate tuning. In contrast,
our method includes CTRs, a class of well-designed tools for label generation, and omits the need for text kernel segmentation through elaborate label design.

2.3. Label Generation and Post-Processing Techniques

Text detection methods generally adopt tailored label generation and post-processing algorithms to make full use of the geometric nature of text instances. In specific, segmentation-based models tend to develop more sophisticated label generation techniques than anchor-based models and classical text detection methods.

Yin et al. [36] adopt multiple stages of post-processing steps including automatic threshold learning and machine learning-based text candidates filtering to refine the results generated by MSER. As a typical anchor-based method, RRPN [25] adopts a post-processing algorithm to compensate for the wrong detection results when handling long text instances. As for segmentation-based models, TextSnake [10] uses sophisticated label generation methods including a rule-based algorithm to extract text kernels. It later adopts a series of post-processing algorithms to reconstruct text instances and filter out false positives. PSENNet uses polygon clipping for label generation. For post-processing, it utilizes an algorithm which expands the regions of the segmented text kernels to reconstruct the text instances. Moreover, according to its open-sourced official implementation, PSENNet adopts a threshold-based filtering technique to filter out false positive predictions.

Current label generation and post-processing techniques generally take advantage of the geometric nature of text instances. However, the label generation methods are generally rule-based, and lacks theoretical support. Also, most post-processing techniques do not utilize much of the geometric features of text instances. In contrast, the label generation process of CTRNet is based on better mathematical principles, and its post-processing pipeline fully utilizes the geometric features of text instances.

3. Methodology

In this section, we first introduce the inference pipeline of CTRNet. Next, we present the components of CTRNet, including the definition of a CTR, the network and label design, and the post-processing algorithm. Finally, the label generation process and the loss design are presented.

3.1. Inference Pipeline

The inference pipeline of the CTRNet framework is divided into two parts: a segmentation network and a post-processing algorithm (see Fig. 2).

We first employ an FPN-based segmentation network to predict 6 geometric features for each pixel, as shown in Fig. 2 (a). The post-processing algorithm later utilizes the geometric features and converts them into pairs of colinear offsets from text kernels and text edges, as illustrated in Fig. 2 (b). Using the offsets, the algorithm can easily locate the text kernels and reconstruct the text instances, obtaining the detection result.

For the details of the inference pipeline, Sec. 3.3.1 includes the specifics of the network structure, Sec. 3.3.2 explains the selection of the geometric features, and Sec. 3.4 introduces the process of the post-processing algorithm.

3.2. Conceptual Text Regions

In this section, we introduce Conceptual Text Regions, a class of human cognition-inspired tools aiming to describe the impression inside our mind when we conceive an arbitrary-shaped text instance. We argue that such impression ought to be a text line of rectangular shape since it is the natural form of text and corresponds to the way we read. In addition, such
impression should be smooth and invertible, since we are able
to conceive every pixel within the text instance.

To achieve that, we develop a harmonic mapping-based rectify-
ting method. The reason behind this choice is that harmonic
mappings enable our proposed rectifying method to be smooth
and bijective \([37]\), while other rectifying methods, such as TPS,
do not ensure bijectivity \([38]\). To prove the necessity of using
harmonic mappings instead of TPS, a comprehensive ablation
study is provided in Sec. 4.3.1.

We start with the construction of a harmonic mapping that
maps an arbitrary text instance into an arbitrary rectangle. We
first denote a rectangle of width \(w\) and height \(h\) by \(R_{w,h}\).
Afterwards, a harmonic mapping, denoted by \(H_{w,h}\), which maps
an arbitrary text instance \(S\) to \(R_{w,h}\) is constructed as follows as:

1. We first define a bijective boundary mapping
   \[
   b : \partial S \rightarrow \partial R_{w,h} ,
   \]
   using the second boundary parameterization method de-
scribed in Sec. 1.2.5 of \([39]\). The rationale behind this selec-
tion is that such method has good physical implications and
ensures bijectivity \([39]\).

2. \(H_{w,h}\) is then obtained by solving the Laplace’s equation
   \[
   \Delta H_{w,h} \equiv 0 \quad \text{(2)}
   \]
   subject to the continuous Dirichlet boundary condition
   \[
   H_{w,h}|_{\partial S} = b . \quad \text{(3)}
   \]

To implement the above discussed procedure, we first obtain a
refined triangulation \(T\) of an arbitrary-shaped text instance
\(S\) using the constrained Delaunay triangulation method \([40]\)
with the maximum triangle area set to \(10^{-3}\) (see Fig. 3 (b)).
Second, we identify the four prominent corners of the text in-
stance according to its reading sequence, which is illustrated
as the green dots in Fig. 3 (b). Please note that this piece of
information is provided in most text detection datasets, includ-
ing the datasets we work with. Third, we obtain the boundary
vertices, \(\{v_1, \ldots v_{n+1}\}\), of triangulation \(T\). For convenience: 1)
\(v_1 = v_{n+1}\); 2) \(v_1\) is the top left corner; 3) the vertices follow
clockwise order. Fourth, we obtain the indices of the corner
vertices, \(\{c_1, c_2, c_3, c_4\}\), with \(c_1 < c_2 < c_3 < c_4\). After these
preparations, Algorithm 1 generates \(H_{w,h}\) and \(H^{-1}_{w,h}\). The Finite
Element Method (FEM) \([41]\) is used in this algorithm, and the
implementation details of the FEM is provided in Sec. 4.2.

Following the Radó-Kneser-Choquet theorem \([42\,43\,44]\), \(H_{w,h}\)
is bijective due to the convexity of \(R_{w,h}\) \([37]\). With the help
of \(H_{w,h}\) and \(H^{-1}_{w,h}\), we define the horizontal and vertical lines
within \(S\) by constructing \(H^{-1}_{w,h} : R_{1,1} \rightarrow S\) and afterwards map-
ning the horizontal and vertical lines within \(R_{1,1}\) back into \(S\).
Fig. 4 illustrates a visual explanation of this definition. Please
note that the width and height of the rectangle \(R_{w,h}\) do not affect

\begin{algorithm}
\caption{The Calculation of \(H_{w,h}\) and \(H^{-1}_{w,h}\)}
\textbf{Input:} \\
\text{Triangulation: } T \\
\text{Boundary vertices: } \{v_1, \ldots, v_{n+1}\} \\
\text{Corner indices: } \{c_1, c_2, c_3, c_4\} \\
\text{Target width and height: } \{w, h\} \\
\textbf{Output: } H_{w,h}, H^{-1}_{w,h} \\

\begin{algorithmic}[1]
\STATE // Obtain the bijective boundary mapping \(b\).
\STATE 1: Define \(b(v_{c_1}), b(v_{c_2}), b(v_{c_3}), \) and \(b(v_{c_4})\) as \((0, h), (w, h),\)
\STATE \((w, 0), \) and \((0, 0)\), respectively.
\STATE 2: \(c_5 := n + 1\) \ // It is for convenience.
\STATE 3: \textbf{for } i \in \{1, 2, 3, 4\} \textbf{ do}
\STATE 4: \textbf{ for } j \in \{c_i, c_i + 1, \ldots, c_i + 1\} \textbf{ do}
\STATE 5: \(b(x_j) := \frac{\sum_{k \in c_i} ||v_{k+1} - v_k||}{\sum_{k \in c_i} ||v_{k+1} - v_k||} (b(v_{c_i}) - b(v_{c_j})) + b(v_{c_i})\)
\STATE 6: \textbf{end for}
\STATE 7: \textbf{ end for}
\STATE // Solve the Laplace’s equation on \(T\) using the FEM.
\STATE 8: \(H_{w,h}(T) := \text{FEM}(T, \Delta H_{w,h} \equiv 0, H_{w,h}|_{\partial T} = b)\)
\STATE // Obtain \(H_{w,h}\) and \(H^{-1}_{w,h}\) through interpolation.
\STATE 9: \(H_{w,h} = \text{LinearInterpolation}(T, H_{w,h}(T))\)
\STATE 10: \(H^{-1}_{w,h} = \text{LinearInterpolation}(H_{w,h}(T), T)\)
\STATE 11: \textbf{return } H_{w,h}, H^{-1}_{w,h}
\end{algorithmic}
\end{algorithm}
this definition, thus, we select $R_{w,h}$ for convenience.

We may now define a CTR as $R_{w,h}$, where $w_c$ and $h_c$ are the average length of all horizontal and vertical lines within $S$, respectively. They can be calculated through the formulae:

$$w_c = \int \int_{R_{1,1}} \| \partial_x H_1^{-1}(x,y) \| \, dx \, dy,$$  

(4)

$$h_c = \int \int_{R_{1,1}} \| \partial_y H_1^{-1}(x,y) \| \, dx \, dy.$$  

(5)

In our implementation, we calculate $w_c$ and $h_c$ through numerical integration.

We now conclude that, for any text instance $S$, there exists a unique CTR, that is $R_{w_c,h_c}$. It inheres the bijection $H_{w_c,h_c} : S \rightarrow R_{w_c,h_c}$. This means that for any given $(x,y) \in S$, there is a one-to-one corresponding $(x',y') \in R_{w_c,h_c}$, and vice versa. This property will be sufficiently utilized in Sec. 3.5 for label generation.

### 3.3. Network and Label Design

In this section, we explain in detail the architecture of our segmentation network and our label design.

#### 3.3.1. Network Design

The purpose of the segmentation network is to predict the geometric features which can be later used. To achieve that, we adopt the well-tested FPN-based segmentation neural network. We now introduce the details of its architecture.

In a general sense, the segmentation network takes in an image of arbitrary shape and produces a feature map of the same width and height. The feature map has 6 channels, which corresponds to 3 classification results and 3 regression results.

In specific, the detailed construction is demonstrated in Fig. 5. To ensure a fair comparison, we adopt the ResNet50 [6] as our backbone network, consistent to multiple recent works achieving state-of-the-art performances [11, 12, 13]. The ResNet50 structure is responsible for downsampling and is divided into 4 stages. Feature maps generated during downsampling are recorded and aggregated with their peer upsampling stages. To generate the prediction result, each one of the feature maps from the upsampling stages, illustrated in orange in Fig. 5, are upsampled and concatenated together. Finally, after a convolutional layer, a pixelwise prediction layer, and an upsampling layer, the prediction result is obtained.

To train the segmentation network, label design, label generation, and a loss function are needed. They are discussed in Sec. 3.3.2, Sec. 3.5, and Sec. 3.6, respectively. Besides, hyperparameters used to train this network in our experiments are unreservedly provided in Sec. 4.2.

#### 3.3.2. Label Design

In this section, we design the geometric features that are learnable by the segmentation network and can be later converted into the offsets required by the post-processing algorithm.

For each point within a text instance, the post-processing algorithm takes in two colinear offsets and locates the corresponding text kernel and text edge. We represent each pair of those offsets with two colinear vectors $v_k = (r_k, \theta)$ and $v_e = (-r_e, \theta)$ (represented using the polar coordinate system), where $v_k$ points to the text kernel, and $v_e$ points to the text edge. Thus, for each pixel within a text instance, there is a corresponding pair of $v_k$ and $v_e$.

For our segmentation network to predict a pair of $v_k$ and $v_e$, in a pixelwise manner, we design the labels with caution to ensure they are learnable by the network. Specifically, we first use one classification head to predict the text regions. Second, we adopt two regression heads to predict radiuses $r_k$ and $r_e$. As for the angle $\theta$, instead of predicting it directly like the text regions and the radiuses, it is conventional [10] to decompose it into

$$\theta = \begin{cases} \frac{2 \arctan \frac{\sin \theta}{1 + \cos \theta}}{\pi}, & \text{if } \cos \theta \neq -1 \\ \pi, & \text{otherwise} \end{cases}$$  

(6)

and predict $\sin \theta$ and $\cos \theta$ instead. However, in our case, the feature maps of $\sin \theta$ and $\cos \theta$ exhibit drastic jump discontinuity (see Fig. 6 (c-d)). The discontinuity and rapid change make the regression heads hard to train and yield inferior results [45, 46]. Numerical experiments proving such deficiency
are provided in Sec. 4.3.2. Inspired by [46], we propose the reference angle-based encoding, which instead deconstructs $\theta$ into
\[
\theta = (-1)^n \alpha + (q_1 + q_2)\pi ,
\]
where $\alpha$ is the reference angle of $\theta$, and the combination of $q_1$ and $q_2$ depicts the quadrant that $\theta$ is located in. Specifically, $q_1$ is 0 if $\theta \in [0, \pi)$ and is otherwise 1. Likewise, $q_2$ is 0 if $\theta \in \left[\frac{\pi}{2}, \frac{3\pi}{2}\right)$ and is otherwise 1. The motivation behind this transformation is to convert the regression target from the discontinuous $\theta$ into $\alpha$, which is generally smooth and continuous within text instances (see Fig. 6 (b)), and handle the discontinuity through classification. We subsequently predict $\alpha$, $q_1$, and $q_2$ with one regression head and two classification heads, respectively. To summarize, we adopt three classification heads to predict $r_k$, $r_{e}$, and $\alpha$.

### 3.4. Post-Processing

To acquire the detection result, a robust algorithm is introduced in this section to convert the geometric features predicted by the segmentation network into text predictions.

After prediction, the network produces 6 geometric features for each pixel, which are text regions, $r_k$, $r_e$, $\alpha$, $q_1$, and $q_2$. To begin with, we define the confidence value of a point as its confidence value for text region segmentation. As a preprocessing procedure, we first discard all points whose confidence value is lower than 0.65, and binarize $q_1$ and $q_2$ with a threshold of 0.5. For each remaining point, we then obtain its corresponding pairs of colinear vectors $v_k = (r_k, \theta)$ and $v_e = (-r_e, \theta)$, where $\theta$ is calculated through Eq. 7. Afterwards, the post-processing algorithm takes in all pairs of $v_k$ and $v_e$. It is divided into three simple operations: clustering, reconstruction, and filtering.

#### 3.4.1. Clustering

As shown in Fig. 7 (a), for the points within text instances, denoted by $p$, we first locate their corresponding kernel points, defined as $p_k = p + v_k$. Next, the connected-components algorithm is applied to all kernel points $p_k$, assigning a class to each kernel point. Finally, we let all $p$ adopt the same class as their corresponding kernel points, forming distinct clusters.

#### 3.4.2. Reconstruction

For each clustered point $p_k$, we first locate its corresponding kernel point, $p_k = p + v_k$, and edge point, $p_e = p + v_e$. Next, we further extend the line segment $p_k p_e$ to restore the full height of the text instance, as shown in Fig. 7 (b). Finally, all points on the line segment are assigned the same class as the clustered point $p$, obtaining the reconstruction result.

#### 3.4.3. Filtering

During clustering and reconstruction, rich information about the text instances are generated, which can be used to filter out false positives.

Specifically, for each text instance, we first define its confidence value as the average of the confidence values within it. Second, we define its distortion as $\sigma_\alpha$, that is the standard deviation of all $\alpha$ values within the text instance. Third, we calculate its aspect ratio through formula $A / 4\mu_{r_k + r_e}$, where $A$ represents the area of the text instance, and $\mu_{r_k + r_e}$ is the average value of
For each point within a text instance, to calculate these geometric information, an ablation study is carried out in Sec. 4.5.2. To better understand this approximation, please recall the definition of $w_c$ and $h_c$ explained in Sec. 3.2. Finally, we train a Support-Vector Machine (SVM) \cite{47} taking the confidence values, distortion, and aspect ratios as features to filter out the false predictions.

The reasons behind the choice of an SVM instead of other techniques, for example a Deep Neural Network (DNN) \cite{48}, are twofold:

1. SVMs yield good performance with less data. In our case, the training samples are mostly identical, since most of the text instances are horizontal or upright. It results in a small amount of effective data. However, DNNs require a lot of data to avoid overfitting, while SVMs do not \cite{49}, thus SVMs are optimal in this case.

2. SVMs are easier to train. Unlike DNNs, SVMs generate training results quickly and deterministically \cite{49}. Allowing us to adopt cross-validation to perform an efficient grid search for the optimal hyperparameters.

Compared with the conventional false positive filtering techniques that rely on hand-crafted thresholds for confidence and area values, as observed in TextSnake \cite{10} and PSENet \cite{11}, our approach not only omits the need for human tuning but also leverages the more interpretable geometric information such as the distortion and aspect ratios. To validate the necessity of such geometric information, an ablation study is carried out in Sec. 4.3.3.

The implementation details of the SVM are provided in Sec. 4.2. Besides, a running time analysis for the SVM is carried out in Sec. 4.5.2.

### 3.5. Label Generation

In this section, we leverage CTRs to obtain the labels designed in Sec. 3.3.2, which are text regions, $r_k$, $r_e$, $r_k - r_e$, $q_1$, and $q_2$. For each point within a text instance, to calculate these geometric features, we need to first obtain its corresponding $v_k$ and $v_e$. These offset vectors can then be converted into the geometric features with Eq. 7.

In specific, for any text instance, it takes four steps: 1) Defining the text kernel and text edge within its CTR, denoted by $C_k$ and $C_e$; 2) Defining the offset vectors within its CTR, denoted by $v_k’$ and $v_e’$; 3) Obtaining $v_k$ and $v_e$ by mapping $v_k’$ and $v_e’$ back into the text instance using $H_{w,b}$; 4) Calculating the geometric features using Eq. 7. Detailed explanations are provided below.

#### 3.5.1. Defining $C_k$ and $C_e$

For any given text instance, we first obtain its CTR, denoted by $C$, and $C = R_{w,b}$, where $w_c$ and $h_c$ are calculated through Eq. 4 and Eq. 5.

It is trivial to define the text edge as $C_e = \partial C$. The text kernel, on the other hand, is defined as:

$$C_k = \begin{cases} \{(x,y)\frac{h_c}{2} \leq x \leq h_c, y = \frac{h_c}{2}\}, & h_c < w_c, \\ \{(w_c, \frac{h_c}{2})\}, & \text{otherwise}. \end{cases}$$

Fig. 8 (a) provides an illustration for the above definitions.

#### 3.5.2. Defining $v_k’$ and $v_e’$

For the CTR, $C$, we use $p’$ to represent a point inside it. For each $p’$, it corresponds to a point in $C_k$ and a point in $C_e$. They are denoted by $p_k’$ and $p_e’$, respectively. For $p_k’$, it is defined as the nearest point to $p’$ in $C_k$. For $p_e’$, on the other hand, it is defined as the intersection between $C_e$ and a ray cast from $p_k’$ through $p’$ (see Fig. 8 (b)). We then define the offset vectors $v_k’$ and $v_e’$ as:

$$v_k’ = p_k’ - p’,$$

$$v_e’ = p_e’ - p’.$$  

An illustration for the above definitions can be found in Fig. 8 (b).

#### 3.5.3. Obtaining $v_k$ and $v_e$

As explained in Sec. 3.2, a CTR naturally inheres a bijective harmonic mapping $H_{w,b}$ that maps any point within its corresponding text instance into itself. With the help of $H_{w,b}$, mapping $v_k’$ and $v_e’$ back into the original text instance is as simple as:

$$v_k = H_{w,b}^{-1}(p_k’) - H_{w,b}^{-1}(p’),$$

$$v_e = H_{w,b}^{-1}(p_e’) - H_{w,b}^{-1}(p’).$$

Please note that after this mapping, $v_k$ and $v_e$ do not theoretically ensure colinearity. However, the variance is generally unnoticeable. Due to this reason, for $v_k = (r_k, \theta_k)$ and $v_e = (r_e, \theta_e)$, we use $v_e = (-r_e, \theta_k)$ to approximate the latter. The approximation is reasonable, since it hardly sacrifices any precision, and is necessary since our segmentation network is designed to handle colinear vectors. After this approximation, we use $\theta$ to represent $\theta_k$ for simplicity.
3.5.4. Obtaining the Labels

For the given text instance, we have obtained all of its \( v_k \)
and \( v_r \). We then utilize Eq. 7 to convert them into the trainable
geometric features required by the segmentation network.

In specific, for each point within the text instance, denoted
by \( p \), its corresponding offsets are \( v_k = (r, \theta) \) and \( v_r = (-r, \theta) \).
We first set the value for text regions as 1, at point \( p \). Second,
the features \( r_k \) and \( r_r \) are naturally obtained from the above
expressions. Third, according to Eq. 7, we can deconstruct \( \theta \) into
\((-1)^{\alpha} (q_1 + q_2) \pi \), thus obtaining the values for \( \alpha, q_1, \) and \( q_2 \),
at point \( p \).

To be complete, for the points that are not included in any
text instance, the values for text regions are set to 0, and the
values for the other geometric features are set to -1, since they
are undetermined.

3.6. Loss Function

In this section, we introduce the loss function adopted to train
the segmentation network.

The loss function can be formulated as:

\[
L = \lambda L_{text} + (1 - \lambda)(L_{r_k} + L_{r_r} + L_{q_1} + L_{q_2} + L_{r_e}),
\]

where \( \lambda \) balances the importance of \( L_{text} \). The binary cross-entropy
loss is adopted for \( L_{text} \), \( L_{q_1} \), and \( L_{q_2} \), and the Smooth-
L1 loss is selected for \( L_{r_k}, L_{r_r}, \) and \( L_{r_e} \), to ensure a more stable
training process \[1\]. To overcome the data imbalance problem
when segmenting text regions, we adopt OHEM 3:1 \[50\] for
training process \[1\]. To overcome the data imbalance problem
\[
\text{during training, the } \lambda \text{ for loss balancing is set to 0.67 and the negative-positive ratio of OHEM is set to 3. Following [11], we}
\text{pre-train our model on ICDAR 2017 MLT for 50K iterations with a learning rate of 1 × 10^{-3}. We fine-tune the pre-trained model on each benchmark dataset for 10K iterations with a}
\text{learning rate of 1 × 10^{-4} to convey a fair comparison against other state-of-the-art methods. For CTW1500 dataset specifically,}
\text{we also train it from scratch for 10K iterations with a learning rate of 1 × 10^{-4} to}
\text{convey a fair comparison against other methods that do not utilize external data.}
\text{During both training and inference phase, all images are proportionally resized to ensure a suitable shorter side length (736}
\text{for ICDAR 2015 and 640 for others).}
\text{For the construction of CTRs, we use scikit-fem [54] to implement the FEM and solve the Laplace’s equation. All}
\text{of the partial differential equations are solved with the results}
\text{recorded before training to ensure a faster training process.}
\text{The SVM adopts the Radial Basis Function (RBF)
[55] kernel, and its hyperparameters C and gamma are}
tuned through grid search within sets \{0.1, 1, 10, 100\} and
\{1, 0.1, 0.001, 0.0001, 0.00001\}, respectively. We use 5-fold
cross-validation to evaluate each pair of the hyperparameters.
The training data for the SVM is generated by running the seg-
mentation model on the training set after each training epoch,
so there is no test data leakage during the process.

4.4. Ablation Study

We conduct a comprehensive ablation study on CTW1500
and ICDAR 2015 datasets to demonstrate the effectiveness
of our framework design.

4.4.1. Harmonic Mappings

We now explain the necessity of using harmonic mappings
instead of TPS to construct CTRs.
Figure 9: Comparison between the TPS method and our proposed harmonic mapping-based method. “HM” indicates harmonic mappings. It can be observed in (a) that the results of harmonic mappings fit the target rectangle perfectly, also they are smoother and more uniform. In contrast, the rectifying results of TPS in (b) are irregular, and they do not fill in the whole rectangle region.

From a theoretical perspective, TPS does not promise bijectivity [38], and there is no simple condition that can ensure a bijective TPS transformation. Thus, it is impossible for TPS to map the geometric features back into the arbitrary-shaped text instances without relying on complicated rules. In comparison, a harmonic mapping-based transformation is bijective as long as the target polygon is convex [37], which is not only elegant but also consistent to our needs. Fig. 9 shows the quality difference between our bijective harmonic mapping-based method and the TPS method.

From a practical perspective, although TPS performs well in many end-to-end text recognition tasks [29, 35], it is not suitable for our CTRNet framework which requires a refined mapping. To show that, we first adopt the nearest-neighbor interpolation method to process the irregular results generated by the TPS transformation. Second, distinct negative effects on the inference pipeline caused by the TPS method can be observed in Fig. 10. Third, the numerical experiments prove our standpoints. As shown in Tab. 1, using TPS introduces a 0.6% performance drop on CTW1500.

Please note the statistical bias that a considerable portion of text instances in CTW1500 are of rectangular shapes. Considering that harmonic mappings and TPS are almost identical when handling rectangular text instances, the negative impact of TPS is a lot more significant than what the number shows. Due to the same reason, we did not perform the same experiment on ICDAR 2015. Since ICDAR 2015 only contains rectangular text instances, there is no significance in such experiment.

We also perform a comparison on the time complexity. To perform the same task of label generation for CTW1500, on average, the harmonic mapping-based method took 1.27 seconds per text instance, and the TPS method took 1.09 seconds per text instance. We conclude that both methods introduce runtime overheads, and TPS is faster. However, it does not affect the training efficiency, since label generation is only performed once before training. The labels are reused efficiently during training, with no run-time overhead.

4.3.2. Reference Angle-Based Encoding

As shown in Tab. 1, the reference angle-based encoding technique explained in Sec. 3.3.2 brings performance gains of 7.5% and 10.8% on CTW1500 and ICDAR 2015 respectively. Thus, we conclude that regression heads do not work well when their target feature maps exhibit abrupt jump discontinuity (see Fig. 6 (c-d)).

4.3.3. Geometric Information

As shown in Tab. 1, utilizing the additional geometric information introduced in Sec. 3.4.3, that is the distortion and aspect ratios, during the filtering process brings a 0.8% performance gain on both CTW1500 and ICDAR 2015, proving the necessity of utilizing such information during the filtering process.
4.3.4. Kernel Scale

It is stated in [10] that text kernels should be granted thickness since a single-point line is prone to noise. And such statement has been practiced in various methods [10] [11] [13]. We are able to increase the kernel thickness by assigning a radius to all kernel points during post-processing. The radius is equal to \( s \cdot h \), where \( s \) is the kernel scale, ranging from 0 to 1, and \( h \) is the height of that text instance, given by \( 2\sigma_l + 2\sigma_r \). As shown in Fig. 11, in our case, the F-measure has a negative correlation with the kernel scale, while the inference speed is observed to have a positive correlation with the kernel scale. However, the gain in inference speed is minor. It is thus optimal to set the kernel scale to 0.

4.4. Comparisons With State-of-the-Art Methods

We compare our proposed method with previous methods on four standard benchmarks, including two benchmarks for arbitrary-shaped text, and two benchmarks for multi-oriented text.

4.4.1. Detecting Arbitrary-Shaped Text

Our CTRNet demonstrates great performance and shape robustness on two arbitrary-shaped text benchmarks: Total-Text and CTW1500.

As shown in Tab. 2 and Tab. 3, our method consistently outperforms previous methods by a large margin, in terms of F-measure. Specifically, CTRNet outperforms the previous state-of-the-art methods by 2.0% and 0.6% on CTW1500 and Total-Text respectively, demonstrating the superiority of CTRNet when detecting arbitrary-shaped text.

Moreover, when training from scratch on CTW1500, compared with other results that do not rely on external data, CTRNet outperforms the most accurate method PAN-640 by 2.5%. Notably, it is only 0.2% behind the previous state-of-the-art method, which is trained with external data, proving the outstanding robustness of our method.

It is worth noting that, compared with PSENet-1s, whose network architecture is almost identical to ours, CTRNet brings huge improvements of 3.5% on CTW1500 and 4.7% on Total-Text. Such improvement is a solid proof for the validity of CTRs and our post-processing algorithm.

4.4.2. Detecting Multi-Oriented Text

Experiments on MSRA-TD500 and ICDAR 2015 show that CTRNet is robust when detecting multi-oriented text. On MSRA-TD500, as shown in Tab. 4, CTRNet outperforms the state-of-the-art methods by 0.5%. Specifically, it surpasses PAN and TextSnake, which relies heavily on text kernel seg-

| Table 2: Experiment results on CTW1500. “Ext.” indicates whether external data is used. “P”, “R”, and “F” represent precision, recall, and F-measure respectively. * indicates the results from [10]. |
| ---- | ---- | ---- | ---- | ---- |
| Method | Ext. | Venue | P | R | F |
| CTPN* [8] | - | ECCV'16 | 60.4* | 53.8* | 56.9* |
| EAST* [9] | - | CVPR’17 | 78.7* | 49.1* | 60.4* |
| CTD+TLOC [18] | - | - | 77.4 | 69.8 | 73.4 |
| TextSnake [10] | ✓ | ECCV’18 | 67.9 | 85.3 | 75.6 |
| PSENet-1s [11] | ✓ | CVPR’19 | 80.6 | 75.6 | 78.0 |
| CSE [56] | ✓ | CVPR’19 | 81.1 | 76.0 | 78.4 |
| PAN-640 [13] | ✓ | ICCV’19 | 84.6 | 77.7 | 81.0 |
| PSENet-1s [11] | ✓ | CVPR’19 | 84.8 | 79.7 | 82.2 |
| Text Perceptron [35] | ✓ | AAAI’20 | 88.7 | 78.2 | 83.1 |
| DB-ResNet50 [12] | ✓ | AAAI’20 | 86.9 | 80.2 | 83.4 |
| CTRNet | - | - | 88.6 | 79.0 | 83.5 |
| PAN-640 [13] | ✓ | ICCV’19 | 86.4 | 81.2 | 83.7 |
| CTRNet | ✓ | - | 88.2 | 83.3 | 85.7 |

| Table 3: Experiment results on Total-Text. “P”, “R”, and “F” represent precision, recall, and F-measure respectively. * indicates the results from [10]. |
| ---- | ---- | ---- | ---- | ---- |
| Method | Venue | P | R | F |
| EAST* [9] | CVPR’17 | 93.4 | 86.9 | 90.2* |
| TextSnake [10] | ✓ | ECCV’18 | 82.7 | 74.5 | 78.4 |
| CSE [56] | ✓ | CVPR’19 | 81.4 | 79.1 | 80.2 |
| PSENet-1s [11] | ✓ | CVPR’19 | 84.0 | 78.0 | 80.9 |
| Text Perceptron [35] | ✓ | AAAI’20 | 88.1 | 78.9 | 83.3 |
| Boundary [29] | AAAI’20 | 85.2 | 83.5 | 84.3 |
| PAN-640 [13] | ✓ | ICCV’19 | 89.3 | 81.0 | 85.0 |
| CTRNet | - | - | 88.4 | 82.9 | 85.6 |

| Table 4: Experiment results on MSRA-TD500. “P”, “R”, and “F” represent precision, recall, and F-measure respectively. |
| ---- | ---- | ---- | ---- | ---- |
| Method | Venue | P | R | F |
| EAST [9] | CVPR’17 | 87.3 | 76.4 | 81.6 |
| RRPN [25] | TMM’18 | 82 | 68 | 74 |
| PixelLink [33] | AAAI’18 | 83.0 | 73.2 | 77.8 |
| TextSnake [10] | ✓ | ECCV’18 | 83.2 | 73.9 | 78.3 |
| Boundary [29] | AAAI’20 | 85.2 | 83.5 | 84.3 |
| PAN-640 [13] | ✓ | ICCV’19 | 84.4 | 83.8 | 84.1 |
| CTRNet | - | - | 92.7 | 79.1 | 85.4 |

| Table 5: Experiment results on ICDAR 2015. The values within parentheses indicate the height of the input image. “P”, “R”, and “F” represent precision, recall, and F-measure respectively. |
| ---- | ---- | ---- | ---- | ---- |
| Method | Venue | P | R | F |
| CTPN [8] | ECCV’16 | 74.2 | 51.6 | 60.9 |
| EAST [9] | CVPR’17 | 83.6 | 73.5 | 78.2 |
| RRPN [25] | TMM’18 | 82 | 73 | 77 |
| RRD [24] | CVPR’18 | 85.6 | 79.0 | 82.2 |
| PixelLink [33] | AAAI’18 | 82.9 | 81.7 | 82.3 |
| TextSnake [10] | ✓ | ECCV’18 | 84.9 | 80.4 | 82.6 |
| Boundary [29] | AAAI’20 | 88.1 | 82.2 | 85.0 |
| PAN-640 [13] | ✓ | AAAI’20 | 88.2 | 82.7 | 85.4 |
| Boundary [29] | AAAI’20 | 88.1 | 82.2 | 85.0 |
| PAN-640 [13] | ✓ | AAAI’20 | 86.9 | 80.2 | 83.4 |
| CSE [56] | CVPR’19 | 86.9 | 80.2 | 83.4 |
| Text Perceptron [35] | AAAI’20 | 91.6 | 81.8 | 86.4 |
| Boundary [29] | AAAI’20 | 91.6 | 81.8 | 86.4 |
| DB-ResNet50 (736) [12] | ✓ | AAAI’20 | 91.8 | 83.2 | 87.3 |
| CTRNet | - | - | 89.5 | 83.5 | 86.4 |
Table 6: Overall results on all four benchmark datasets. “F” represents F-measure. “SVM”, “Post.”, and “Total” indicate the average time consumption of the SVM module, the post-processing algorithm, and the inference pipeline as a whole.

| Dataset      | F     | SVM Time (s) | Post. Time (s) | Total Time (s) |
|--------------|-------|--------------|----------------|----------------|
| CTW1500      | 85.7  | 4.83 x 10^-4 | 0.154          | 0.191          |
| Total-Text   | 85.6  | 5.26 x 10^-4 | 0.092          | 0.127          |
| MSRA-TD500   | 85.4  | 1.24 x 10^-4 | 0.072          | 0.103          |
| ICDAR 2015   | 86.4  | 1.98 x 10^-4 | 0.060          | 0.108          |

Qualitatively, it can be observed in Fig. 12 that through elaborate label design, our CTRNet generates stable and accurate predictions for text kernels, without the need of directly segmenting them. Meanwhile, CTRNet consistently produces accurate and shape-robust detection results when handling horizontal text, multi-oriented text, and arbitrary-shaped text. Quantitatively, it can be observed in Tab. 6 that CTRNet consistently achieves excellence on all four of the datasets. Notably, to the best of our knowledge, CTRNet is among the first methods to achieve F-measures higher than 85.0% on all these datasets, which proves the solid superiority of CTRNet, in terms of accuracy and consistency.

4.5. Overall Results and Running Time Analysis

4.5.1. Overall Results

Qualitatively, it can be observed in Fig. 12 that through elaborate label design, our CTRNet generates stable and accurate predictions for text kernels, without the need of directly segmenting them. Meanwhile, CTRNet consistently produces accurate and shape-robust detection results when handling horizontal text, multi-oriented text, and arbitrary-shaped text. Quantitatively, it can be observed in Tab. 6 that CTRNet consistently achieves excellence on all four of the datasets. Notably, to the best of our knowledge, CTRNet is among the first methods to achieve F-measures higher than 85.0% on all these datasets, which proves the solid superiority of CTRNet, in terms of accuracy and consistency.

4.5.2. Running Time Analysis

We use a single RTX2080Ti GPU and the Dual Intel Xeon E5-2650 v4 @ 2.20GHz processor for running time analysis. As shown in Tab. 6, it takes an average of 0.132 seconds for CTRNet to process an image. While the SVM module poses unnoticeable time cost, the post-processing as a whole generally takes up more than 70% of the inference time. However, it is worth noting that unlike other segmentation-based methods that implement the post-processing algorithms using the more efficient C++, such as PSENet-1s, our post-processing algorithm, for the purpose of this study, is implemented purely in Python. Nevertheless, the inference speed of CTRNet is still significantly higher than a large set of recent methods [10][11][56], including PSENet-1s.

To carry out a fair comparison, we implement PSENet-1s using its open-sourced official implementation, under our hardware environment. According to our tests on CTW1500, its average running time for each picture is 0.334 seconds. It is slower than CTRNet, which takes 0.191 seconds under the same experimental configuration. This shows that CTRNet has a competitive running time performance.

5. Conclusion

In this paper, we propose an effective framework to detect arbitrary-shaped text with outstanding accuracy and stability. We first introduce Conceptual Text Regions, a class of
cognition-inspired tools for elaborate label generation. Further, we propose a well-designed inference pipeline to predict text instances using the geometric features generated by CTRs. The fact that CTRs allow for sophisticated label design and that the inference pipeline omits the need for text kernel segmentation establishes CTRNet as a robust and accurate arbitrary-shaped text detector. Achieving F-measures greater than 85.0% on all four of CTW1500, Total-Text, MSRA-TD500, and ICDAR 2015, CTRNet demonstrates superior advantages in terms of accuracy and consistency when compared with previous state-of-the-art text detectors.

For future study, we consider it meaningful to investigate the possibility of establishing the CTR method as a universal text label generation technique. It is also of interest to utilize deep reinforcement learning and multi-task learning algorithms \[57, 58, 59, 60\] to optimize the generalization capability of such text recognition framework.

References

[1] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, IEEE Trans. Pattern Anal. Mach. Intell. 39 (2017) 1137-1149. https://doi.org/10.1109/TPAMI.2016.2577031
[2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu, A.C. Berg, Ssd: Single shot multibox detector, in: Proc. Eur. Conf. Comput. Vis., 2016: pp. 21-37. https://doi.org/10.1007/ 978-3-319-46448-0_2
[3] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015: pp. 3431-3440. https://doi.org/10.1109/CVPR.2015. 7298865
[4] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: Proc. Int. Conf. Med. Image Comput. Comput. Interv., 2015: pp. 234-241. https://doi.org/10.1007/ 978-3-319-24574-4_28
[5] Q. Ning, J. Zhu, C. Chen, Very fast semantic image segmentation using hierarchical dilation and feature refining, Cognit. Comput. 10 (2018) 62-72. https://doi.org/10.1007/ s12559-017-9530-0
[6] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016: pp. 770-778. https://doi.org/10.1109/CVPR.2016.90
[7] G. Huang, Z. Liu, L.V.D. Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017. pp. 2261-2269. https://doi.org/10.1109/CVPR. 2017.245
[8] Z. Tian, W. Huang, T. He, P. He, Y. Qiao, Detecting text in natural image with connectionist text proposal network, in: Proc. Eur. Conf. Comput. Vis., 2016: pp. 56-72. https://doi.org/10.1007/ 978-3-319-46884-8_4
[9] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, J. Liang, East: An efficient and accurate scene text detector, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017: pp. 2642-2651. https://doi.org/10. 1109/CVPR.2017.283
[10] S. Long, J. Ruan, W. Zhang, X. He, W. Wu, C. Yao, Textsnake: A flexible representation for detecting text of arbitrary shapes, in: Proc. Eur. Conf. Comput. Vis., 2018: pp. 19-35. https://doi.org/10.1007/ 978-3-030-01216-8_2
[11] W. Wang, E. Xie, X. Li, W. Hou, T. Lu, G. Yu, S. Shao, Shape robust text detection with progressive scale expansion network, in: IEEE Conf. Comput. Vis. Pattern Recognit., 2019: pp. 9328-9337. https:// doi.org/10.1109/CVPR.2019.00956
[12] M. Liao, Z. Cui, C. Yan, K. Chen, X. Bai, Real-time scene text detection with differentiable binarization, in: Proc. AAAI Conf. Artif. Intell., 2020: pp. 11474-11481 https://doi.org/10.1609/aaai.v34i07.6812
[13] W. Wang, E. Xie, X. Song, Y. Zang, W. Wang, T. Lu, G. Yu, C. Shen, Efficient and accurate arbitrary-shaped text detection with pixel aggregation network, in: Proc. IEEE Int. Conf. Comput. Vis., 2019: pp. 8440-8449. https://arxiv.org/abs/1908.05900
[14] T.Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, S. Belongie, Feature pyramid networks for object detection, in: Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017: pp. 936-944. https://doi.org/10. 1109/CVPR.2017.106
[15] F. Wang, X. Wang, J. Tang, B. Luo, C. Li, Vtann: Visual tracking with attentive adversarial network, Cognit. Comput. (2020). https://doi. org/10.1007/s12559-020-09727-3
[16] B. Farhadinia, A cognitively inspired knowledge-based decision-making methodology employing intuitionistic fuzzy sets, Cognit. Comput. 12 (2020) 667-678. https://doi.org/10.1007/ s12559-019-09702-7
[17] X. Bi, X. Zhao, H. Huang, D. Chen, Y. Ma, Functional brain network classification for alzheimer’s disease detection with deep features and extreme learning machine, Cognit. Comput. 12 (2020) 513-527. https://doi.org/10.1007/s12559-019-09688-2
[18] Y. Liu, L. Jin, S. Zhang, S. Zhang, Detecting curve text in the wild: New dataset and new solution, 2017. https://arxiv.org/abs/1712. 02170
[19] C.K. Ch'ng, C.S. Chan, Total-text: A comprehensive dataset for scene text detection and recognition, in: Proc. IAPR Int. Conf. Doc. Anal.
