SCENE TEXT UNDERSTANDING IN NATURAL IMAGES WITH CONVOLUTIONAL NEURAL NETWORKS

A Dissertation in
Information Sciences and Technology
by
Dafang He

© 2019 Dafang He

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2019
The dissertation of Dafang He was reviewed and approved* by the following:

C. Lee Giles  
David Reese Professor of Information Sciences and Technology  
Dissertation Advisor, Chair of Committee

Daniel Kifer  
Associate Professor of Computer Sciences and Engineering

Zihan Zhou  
Assistant Professor of Information Sciences and Technology

James Wang  
Professor of Information Sciences and Technology

Mary Beth Rosson  
Director of Graduate Programs, Information Sciences and Technology

*Signatures are on file in the Graduate School.
Abstract

Text in images contains rich semantic information. The ability to read text has many different applications, including in autonomous driving, image or video indexing, and the creation of assistive technology for visually impaired people. This task is typically called scene text understanding. In order to understand text in natural images, there are usually several sub-fields related to it: (1) scene text detection, (2) scene text recognition, and (3) scene text verification or retrieval. In this dissertation, I investigate scene text understanding with a focus on text detection and text verification.

Scene text detection aims at finding the location of each text instance. A bounding box needs to be predicted for each text instance. Scene text detection shares several common difficulties with regular object detection, including noise and variance of scales. However, one of the major differences between regular object detection and scene text detection is that the former is usually needed to predict an oriented or even curved bounding box for each text instance. Scene text recognition usually follows scene text detection in an end-to-end text reading system. The recognition model needs to transcribe each single text instance. Scene text verification verifies the existence of text in natural images. It is the most critical part of building a scene text retrieval system.

In this dissertation, I explore various methods for scene text detection and verification with a convolutional neural network (CNN). Specifically, for scene text detection, I propose three algorithms and one training framework. The first algorithm adopts a traditional region proposal method with a novel CNN classifier that aggregates local context into classification. The second detection algorithm uses a fully convolutional neural network for semantic text segmentation. A novel instance-aware segmentation is proposed to further split the extracted text block into text instances. The third work focuses on arbitrary-oriented scene text detection. It proposes a general and novel framework called Detect-Associate-Segment.
(DAS) for detecting arbitrary-oriented text. A model based on keypoint detection is designed under the framework which achieves state-of-the-art performance in various benchmark datasets. In addition to its investigation of detection algorithms, this dissertation explores a new training framework for scene text detection. A novel contour segmentation task is introduced to assist with scene text detection and is found to improve the final performance. This dissertation considers a new end-to-end model design for scene text verification that outperforms traditional algorithms by a large margin. It is demonstrated on a large scale in a scene text dataset with millions of street view images.
Table of Contents

List of Figures \hspace{1cm} x

List of Tables \hspace{1cm} xviii

Acknowledgments \hspace{1cm} xx

Chapter 1

Introduction 1

1.1 Scene Text Detection 2

1.1.1 Traditional Methods 2

1.1.2 Contemporary Methods 3

1.1.3 Unique Challenges of Scene Text Detection 5

1.2 Scene Text Recognition 6

1.3 End-to-End Scene Text Reader 7

1.4 Scene Text Verification/Retrieval 7

1.5 Datasets 8

1.5.1 Real Labeled Datasets 8

1.5.1.1 ICDAR Datasets 8

1.5.1.2 Street View Text (SVT) 8

1.5.1.3 UberText 9

1.5.1.4 MSRA-TD500 9

1.5.1.5 ctw1500 9

1.5.1.6 TotalText 9

1.5.2 Synthetic Data 9

1.6 Summary of Contributions 10

1.7 Dissertation Structure 10
Chapter 2

**Aggregating Local Context for Accurate Scene Text Detection**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Introduction</td>
<td>12</td>
</tr>
<tr>
<td>2.2 Methodology</td>
<td>13</td>
</tr>
<tr>
<td>2.2.1 Cascaded Classification and Local Context Aggregation</td>
<td>14</td>
</tr>
<tr>
<td>2.2.2 Hard Example Mining</td>
<td>20</td>
</tr>
<tr>
<td>2.2.3 Character Component Graph</td>
<td>20</td>
</tr>
<tr>
<td>2.2.4 Line Extension and Refinement</td>
<td>22</td>
</tr>
<tr>
<td>2.3 Experiments and Evaluation</td>
<td>26</td>
</tr>
<tr>
<td>2.4 Conclusions</td>
<td>29</td>
</tr>
</tbody>
</table>

Chapter 3

**Multi-scale FCN with Cascaded Instance Aware Segmentation for Arbitrary Oriented Word Spotting In The Wild**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Introduction</td>
<td>30</td>
</tr>
<tr>
<td>3.2 Multi-Scale, Shared-Net FCN</td>
<td>33</td>
</tr>
<tr>
<td>3.2.1 Design Rationale</td>
<td>33</td>
</tr>
<tr>
<td>3.2.2 Architecture</td>
<td>34</td>
</tr>
<tr>
<td>3.2.3 Training with Per-scale Loss</td>
<td>34</td>
</tr>
<tr>
<td>3.3 Cascaded Text Instance Segmentation</td>
<td>36</td>
</tr>
<tr>
<td>3.3.1 Architecture</td>
<td>36</td>
</tr>
<tr>
<td>3.3.2 Pipeline</td>
<td>37</td>
</tr>
<tr>
<td>3.3.3 Optimization</td>
<td>40</td>
</tr>
<tr>
<td>3.3.4 Error Handling in Instance-Aware Segmentation</td>
<td>41</td>
</tr>
<tr>
<td>3.4 Experiments</td>
<td>42</td>
</tr>
<tr>
<td>3.4.1 Curved Text</td>
<td>42</td>
</tr>
<tr>
<td>3.4.2 Evaluation on Instance Segmentation</td>
<td>45</td>
</tr>
<tr>
<td>3.4.3 ICDAR 2013</td>
<td>47</td>
</tr>
<tr>
<td>3.4.4 ICDAR 2015</td>
<td>48</td>
</tr>
<tr>
<td>3.4.5 Street View Text and CUTE80</td>
<td>48</td>
</tr>
<tr>
<td>3.4.6 Limitation</td>
<td>48</td>
</tr>
<tr>
<td>3.5 Conclusion</td>
<td>49</td>
</tr>
</tbody>
</table>

Chapter 4

**Detect-Associate-Segment (DAS): A framework for arbitrary oriented scene text detection**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Introduction</td>
<td>50</td>
</tr>
<tr>
<td>4.2 Uniquely Identifying Text Instance</td>
<td>51</td>
</tr>
<tr>
<td>4.3 DAS framework</td>
<td>52</td>
</tr>
<tr>
<td>4.4 Key Point Based Text Detection Model</td>
<td>54</td>
</tr>
</tbody>
</table>
Chapter 6
Guided Attention for Large Scale Scene Text Verification

6.1 Introduction ............................................. 85
6.2 Method .................................................. 88
   6.2.1 Model Architecture ................................. 88
      6.2.1.1 CNN Encoder with Coordinate Map .......... 88
      6.2.1.2 Guided Attention Decoder .................... 89
   6.2.2 Model Training and Sub-batch Architecture .... 91
   6.2.3 Hard Negative Mining(HNM) ...................... 91

6.3 Experiments ............................................. 93
   6.3.1 Dataset Description and Experiment Setting .......... 93
      6.3.1.1 SVBM ........................................ 93
      6.3.1.2 UberText and FSNS ............................ 94
   6.3.2 Qualitative Results ................................ 94
      6.3.2.1 Subset of Text ................................ 95
      6.3.2.2 Random Shuffled Text ......................... 95
      6.3.2.3 Non-Business Text .............................. 96
      6.3.2.4 Hard Negative Mining ......................... 96
   6.3.3 SVBM Quantitative Evaluation ...................... 97
      6.3.3.1 Baseline Models ............................... 97
      6.3.3.2 Comparison w.r.t baselines .................... 97
      6.3.3.3 Evaluation w.r.t Maximum Length ............. 98
      6.3.3.4 Evaluation w.r.t Hard Negative Mining ........ 98
   6.3.4 UberText and FSNS Quantitative Evaluation ....... 99

6.4 Conclusion .............................................. 99

Chapter 7
Conclusion and Future Work .............................. 102

7.1 Conclusion ............................................. 102
7.2 Future Work ........................................... 104

Appendix A
Supplementary Material for Chapter 3 .................. 105

Appendix B
Supplementary Material for Chapter 4 .................. 108

B.1 Corner Association Results ........................... 108
B.2 Annotation for TotalText Dataset .................... 108
B.3 Text Center Line Model Prototype .................... 109
List of Figures

1.1 (a) Illustration of relative regression based on default anchor box. 
(b) Direct regression [1] centered at point P. It direct regress the 
distance to the four border of the bonding box. .......................... 4
1.2 Illustration of multi-oriented or arbitrary oriented text groundtruth 
in standard datasets. ............................................................... 5
1.3 A typical end-to-end scene text reader built with CTC text recognizer. 7

2.1 Scene text that have been successfully detected by the proposed 
algorithm. Images are from ICDAR 2013 and SVT dataset. ......... 14
2.2 Images from the scene text detection pipeline. From left to right: 
(1) Region proposals; (2) Coarse predictions; (3) Predictions after 
aggregating local context; (4) Final detected text lines. ............... 15
2.3 The proposed TLFAN architecture for scene text detection. Left: 
CNN structure of the proposed network. The bottom 3 layers of 
convolution and pooling are shared, and for context branch, another 
CNN layer and pooling layer is added to produce deeper represen-
tation. (b) The whole architecture of the network. One column is 
for the given patch that the model tries to classify. The other is 
for extracting context information for this patch, and the generated 
feature vector will be further used to give an accurate prediction of 
the region. ................................................................. 16
2.4 Cropped image patches which demonstrate that local context helps 
in distinguishing between background noise and text. (a) The orig-
inal image patches cropped tightly from the generated regions. (b) 
The horizontal context images corresponding to the regions on the 
left. All the examples here are background noises which easily cause 
false positive if only a tight bounding box is considered for classifi-
cation. Instead, the proposed model can effectively aggregates local 
context and give accurate prediction. ......................... 17
2.5 Several training samples. Left: positive training samples and their context input patches. Right: negative training samples and their context input patches.

2.6 The comparison of performance between a text/non-text classifier proposed in [2], and the proposed method in two challenging image in ICDAR 2013 test set. From left to right: (1) original image, (2) the saliency image generated by [2], (3) the saliency image generated by the proposed method and (4) final detected text lines by the proposed pipeline.

2.7 The proposed algorithm which could effectively resolve conflicted candidate lines and find best line assignment of text regions. The final detected lines could be in any orientation. From left to right: (a) The constructed character component graph. (b) A set of generated stable pairs which will be used to create candidate lines. (c) Candidate lines represented as different colored bounding boxes. (d) Detected text lines.

2.8 The proposed line extension pipeline: (a) the original image with the detected lines, (b) cropped line image patch the model used for estimation of GMM, (c) the skeleton of all the region components, (d) cropped image patch whose color distribution needs to be estimated, (e) predicted color probability image, (f) MSER result on the estimation patch, (Because of the large contrast in the predicted image patch, there are only few bounding box that I need to verify) (g) regions after running the standard CNN branch and non-maximum suppression and (h) The final detected line.

2.9 The proposed line refinement illustration: (a) The proposed architecture which used the CNN from detection part. Training is only on the fully connected layers for classification and regression. (b) Several examples of training images. The red lines are drawn from minimal and maximal vertical coordinates. The green dots are the vertical coordinates for oriented bounding box. (c) Several testing result images. The oriented bounding box is drawn with green lines instead of dots for better visualization of the orientation.

2.10 The proposed line refinement pipeline. (a) For each cropped lines, it does sliding window prediction and merge the results. (b) Several line regression results based on the proposed framework. The red lines correspond to standard regression, and the green lines represent oriented regression.
2.11 For each pair of images, left: the original incomplete annotation, right: detection result of the proposed model as well as the fully annotated ground truth. The fully annotated dataset provides oriented bounding box annotation. Green boxes represent the detected result which matches the ground truth. Yellow boxes represent the

ground truth.

2.12 Example images where the proposed model failed to detect all the lines or detected the wrong lines. The green boxes are the text lines that are correctly detected. The blue boxes are text lines that the proposed model fail to detect, and the red boxes are false positives, or incomplete detection.

3.1 Scene texts that have been successfully detected by proposed systems.

3.2 The pipeline of the proposed algorithms with demonstration of the two level cascades. Multi-scale FCN is used to extracts text blocks. Then for each extracted text block, it predicts the text center line by the proposed TL-CNN. After extracting all the text lines, IA-CNN is used to extract each instance word. The pipeline could extract text line in arbitrary orientations.

3.3 The architecture of the multi-scale FCN. The input is downsampled to 0.5 and 0.25 of the scale of the input image. Prediction is done by jointly considering features from three scales. The parameters of the convolutional parts are shared among all three branches.

3.4 Comparison of the multi-scale, shared-net FCN with a single branched FCN. Left: input image. Middle: single branched FCN. right: shared-net, multi-scale FCN.

3.5 Results of the instance segmentation model. It can capture word or text line instances in a wide variety of circumstances with arbitrary orientations. The segmented results could be directly thresholded to get the final bounding boxes.

3.6 The architecture visualization (left) and details (right) of the TL-CNN and IA-CNN. The TL-CNN (left branch) is for producing the instance level words center lines. The IA-CNN (right branch) is for producing each word instance segmentation once. Their inputs have different number of channels since the IA-CNN needs a 4-channel tensor. ReLU layer and batch normalization [3] are ignored for simplicity.
3.7 The method of instance segmentation. It tries to decompose the hard task of instance segmentation into easier sub-tasks. First TL-CNN is used to generate text center line labeling. Then I simply decompose from the probability map a set of text lines. Each of these text lines will generate an input to the IA-CNN together with the original input image. Then IA-CNN can produce an accurate instance segmentation of each text lines with arbitrary orientation.

3.8 Examples of images that text block FCN might fail to remove. Instead, TL-CNN can remove them. I only show the output map from TL-CNN of the first image to save space. The output of others are similar (completely black). Text lines can not be extracted from them and thus I can remove them safely.

3.9 Illustration of creating noisy training set for IA-CNN network. The bounding box of the word is for illustration purposes. $Length_1$ and $Length_2$ are the length of longer side and shorter side or the word patch, respectively. $p_1, p_2$ and the corresponding green line represent the ground truth center line. $q_1, q_2$ and the corresponding red line represent the shifted noisy line. $L_1$ and $L_2$ are the offsets from $p_1$ to $q_1$ along the longer side and shorter side, respectively.

3.10 Some training samples of the instance segmentation model.

3.11 Some training negative samples of the instance segmentation model.

3.12 Some training negative samples of the instance segmentation model.

3.13 Some examples showing the error handling of IA-CNN step. From left to right: input text block image, extracted text line instance, raw instance segmentation result, final extracted instance box. Note that, the extracted text instances are not correct, but the model can finally recover the results. In the raw instance segmentation results, each pixel might belong to two instance masks, since the operation of IA-CNN is done separately for each extracted text line. The instance mask drawn later will cover the previous one for the raw instance results shown here.

3.14 On the top two rows, I show some examples of the noisy training data for instance-aware segmentation. The input line image is shown in the image with a black line crossing the word. Here I only visualize one instance line per image for illustration purposes. On the last two rows, I show examples demonstrating the effectiveness of augmented the noisy training data. From left to right: Input text block image, instance results with model trained on good quality data, instance results with model trained on noisy data. More results about such error handling are in supplementary material.
3.15 Results of the instance segmentation model on several curved text blocks. It can accurately capture curved text from a lot different scenes. From left to right: (1) The input image. (2) Text line captured by text line model (3) Instance segmentation results on these curved data.

3.16 The precision (red) and recall (green) curve with respect to the choice of $T$ on 1300 randomly cropped text block image from IC13, IC15 training set.

3.17 Example images that the proposed algorithm fail to detect correctly. Blue rectangles mean that the model fails to localize the texts.

4.1 Scene texts that have been successfully detected by the proposed systems. Images are from ICDAR2015 and TotalText dataset. For TotalText dataset, I visualize the text region with polygons that encapsulate the text region.

4.2 Regular horizontal bounding box can not perfectly distinguish one text instance from another.

4.3 Examples of instance clues that can be used to uniquely identify one text instance. From left to right: original image, text center line and text corners. Clues corresponding to different text instance are drawn with different colors.

4.4 The pipeline of the proposed key point based text detection method. It follows the D(etect)-A(ssociate)-S(egment) framework. **Detect:** The detected corner points are visualized with different colors. **Associate:** Regression is used to link the corner points with the assist of embeddings. **Segment:** For each possible hexagonal, I pair it with the instance input to the network for segmentation. The hexagon is drawn only for visualization purpose.

4.5 The annotation for the 6 corner points with different colors. I ignore text instance id $k$ here. The links are visualized for pairs of neighbor corner points in $M$.

4.6 Visualized Ground Truth. From left to right: the original image with the connected corner points visualized, visualized corner points, regression targets for $l_1$ corner points.

4.7 Left: Visualized process of using embedding to remove incorrect regression link (from $r_2$, $c_2$). Red cross visualizes the removed links. Right: Invalid polygon will be removed in geometry constraints step.

4.8 Left: Visualized fails of embedding links. Embeddings are more useful for nearby text.
4.9 Visualized results of corner points association. The predicted points from heat map is visualized with solid dot and regressed points are visualized with circles. Better when zoomed in. 

4.10 Visualized text segmentation results. From left to right (each three images): input image, visualized instance input, visualized segmentation results.

4.11 Visualized end-to-end example with the input hexagon (left) and the instance mask (right). In these extremely curved cases, segmentation can clearly improve the performance.

4.12 The network generates 3 kinds of different feature maps: key point heat maps, key point regression maps, key point embedding maps.

4.13 Some failure examples from the corner points association step. Red box: false positive. Green Box: true positive. Yellow Box: missed detection.

5.1 Scene texts that have been successfully detected with the model trained with proposed framework. Images are from ICDAR2013, ICDAR2015, UberText dataset. Better when zoom in.

5.2 Contour examples which are generated by polygon annotations. Most datasets provide quadrilateral annotations (e.g. ICDAR, UberText). The blue color is only for visualization.

5.3 The generated ground truth for training the contour network. Better when zoomed in.

5.4 Contour detection is easy to learn and can help scene text detection.

5.5 Auxiliary TextContourNet: A general framework of adding the contour task as an auxiliary loss.

5.6 Two frameworks for using in a cascade manner the contour information for scene text detection. **up**: Join the contour in the early stage. In this case, there is no shared convolutional parameters. **down**: Join the contour in the later stage and share the features and parameters in the encoder.

5.7 F-1 score w.r.t the IOU threshold in scene text detection evaluation for three different models. The $\text{CascadeContourNet}_2$ and $\text{AuxiliaryContourNet}_1$ consistently outperform baseline model.

5.8 Some detection results with their contour prediction. Better when zoom in.

5.9 Comparison of the detection results from the proposed cascade contour model with baseline. For each column: (1) detection results from proposed model, (2) predicted contour and (3) detection results from baseline model.
5.10 Some imperfect or failure examples. ........................................... 84

6.1 An example application of scene text verification task. The proposed model takes a business storefront image and a potential business name, and then directly outputs how likely the storefront image matches the business name. The red text means it is the ground truth for this image and the proposed model gives it a high score. . 85

6.2 The architecture of the proposed scene text verification model. The model takes in a sequence of characters and an image, and predict the probability of the input text sequence being present in the input image. .......................................................... 89

6.3 Illustration of encoding the coordinates as a one-hot feature vector \( f_{xy} \) and concatenate it with the original visual features \( f_v \). A feature map with \( H \times W \times C \) will have corresponding coordinate feature map with size \( H \times W \times (H+W) \) ........................................... 90

6.4 Example showing how to determine the hardness of a negative example based the corresponding positive candidates. The different color represents the different hardness levels. ................................. 92

6.5 A qualitative evaluation on several specific behaviors of the proposed model when it is given the same input image with different input text. On the left is the input image. The first section in the table on the right shows results when input is a subset of ground truth text. The 2nd section in the table demonstrates when randomly shuffled groundtruth text is used as input. The last section represents results when the input text are not the corresponding business name, but are also in the image. ................................. 94

6.6 Two visual examples illustrating the performance gain after mining hard negative candidates for training. I observe that after HNM, the gap between the best prediction and the second best prediction has usually been increased. ................................. 96

6.7 (a) The Precision-Recall curve of the proposed model compared with other baselines. (b) The Precision-Recall curve of the proposed model w.r.t. different maximum length of text. (c) The Precision-Recall curve comparison before and after HNM. ................. 97

6.8 (a) The recall w.r.t threshold for different models on UberText. I only evaluate positive text here. (b) P-R curve for OCR [4] and the ensembled model on FSNS. ................................. 99
6.9 Example images from the test set. I only visualize the top predicted candidates. The first three rows show the examples that we give good predictions (high score for positive candidate, low score for negative ones). The last row shows several failure examples.

A.1 More results of the word or text line instance segmentation

A.2 More results of the word or text line instance segmentation

B.1 More results of the corner association step. Focused on curved text.
   Better when zoomed in.


B.3 Visualized corner points annotation.

B.4 The pipeline of the prototype model using Text Centerline Segmentation.

B.5 Results produced by the prototype model based on text centerline. Each row from left to right: the input image, the text-nontext segmentation results, the text centerline segmentation results and the final instance segmentation results.
List of Tables

2.1 The table of several merged classes. The characters’ shapes are almost the same to each other when considered in small image patch in each cell of the table. I merge them into one class. 18

2.2 Localization performances on: top: IC13 (%), bottom: IC03 (%) data sets. Bold number outperforms previous methods. ‘Proposed no post’ represents final results without line extension and refinement steps. 27

2.3 Text detection performance on SVT. The bold results outperforms the previous state-of-the-art results. (1) partial annotated: detection recall measured with the partial annotation. The accuracy here does not makes sense, so I only tested its recall. (2) fully annotated: detection precision, recall, F-measure with full annotation. 27

3.1 Localization performances (%) on ICDAR 2013 data sets. Bold number outperforms other methods. 47

3.2 Localization performances (%) on ICDAR 2015 data sets. Bold number outperforms other methods. Some methods do not have references. 48

3.3 Text Localization evaluation (%) on SVT (left) and CUTE80 (right) dataset. I only evaluate recall on these datasets because they are only partially annotated. 48

4.1 Localization Performance (%) on ICDAR2013 dataset. R: Recall. P: Precision. 64

4.2 Localization Performance (%) on ICDAR2015 dataset. R: Recall. P: Precision. 64

4.3 Localization Performance (%) on MSRA-TD500 dataset. R: Recall. P: Precision. 65

4.4 Localization Performance (%) on TotalText dataset. 65
5.1 Capacity study (Ability in fitting training data): Localization performance (%) on ICDAR2015 dataset. I use this study to show how well the model can fit the training data. 79

5.2 Localization performance (%) on ICDAR2013 dataset. R: Recall. P: Precision. MS represents multi-scale testing. I compare the proposed model with the baseline as well as other methods. 79

5.3 Localization performance (%) on ICDAR2015 dataset. R: Recall. P: Precision. 80

5.4 Localization performance (%) on UberText dataset. TextFasterRCNN is a proposed baseline method based on tensorflow object detection API. 81
Acknowledgments

First, I would like to thank my advisor, Professor C. Lee Giles, for his guidance during my years at Penn State. I came to Penn State with little experience in conducting research, and it wouldn’t have been possible for me to accomplish this dissertation without his guidance. In addition to his expertise in research, Dr. Giles shared his life experiences with me and motivated me to overcome various difficulties I had during the years. It was a great experience to be part of his lab, and I would like to extend my deepest gratitude to him.

I also would like to give special thanks to Professor Daniel Kifer, who offered invaluable guidance during my graduate school years. His insightful ideas and support helped me solve many difficult research problems.

I want to thank Professor Zihan Zhou and Professor James Wang for their helpful advice as my dissertation committee members.

I have been extremely fortunate to work with many excellent researchers during these years. I’d like to thank each one of them: Xiao Yang, Dr. Wenyi Huang, Dr. Chen Liang, Dr. Scott Cohen, Dr. Brian Price, Dr. Alexander G. Ororbia, Dr. Yeqing Li, Dr. Alexander Gorban, Dr. Qian Yu, Dr. Derrall Heath, Dr. Julian Ibarz, Ke Yuan, and Ankur Mali.

I will always treasure my years of doctoral study at Penn State, and I would like to thank all my friends with whom I have shared these years. They have made the experience at Penn State enjoyable.

Finally, I would like to thank my parents, Bing Liu and Zhi He. They have offered me their encouragement and support all these years. Their love is what makes me strong and persistent.

The research described in this dissertation was supported by an NSF grant (CCF 1317560) and a hardware grant from NVIDIA.
The task of scene text understanding in natural images or videos contains several sub-tasks: scene text detection, scene text recognition, and scene text retrieval.

The algorithm for scene text detection aims at localizing text in an image. The expected output of the algorithm is a set of bounding boxes that represent the locations of different text instances in an image. In contrast, scene text recognition takes an image patch that is assumed to contain only a single text instance and predicts the text contained in it. An end-to-end scene text reading system typically involves the sequential application of these two modules. The locations of text instances in an image are detected during the first phase, and the image patch of each text instance is cropped and sent to text recognition module to read. Generally speaking, the bottleneck in an end-to-end scene text reading system is the text detection part. The challenges of localizing arbitrary-oriented text in a scene image remain unresolved.

Scene text retrieval involves retrieving images with certain text in them. A typical solution is to build an end-to-end scene text reader with a detector and a recognizer so that all the text in the images is read and images are retrieved based on text matching. Scene text verification verifies the existence of a text in an image and can be seen as a sub-task for scene text retrieval.

In the remainder of this chapter, I discuss the three fields with a focus on scene text detection and scene text verification. A brief introduction to state-of-the-art scene text recognition is also included. After discussing these three problems, I introduce related public datasets and competitions.
1.1 Scene Text Detection

Scene text detection has received increasing attention from computer vision researchers in the past decade. The focus of the problem has shifted from the detection of horizontal text [5, 6, 7, 8, 9, 2, 10, 11, 12] to the detection of multi-oriented [13, 14, 15, 1] or even arbitrary-oriented [16, 17] text. However, detecting multi-oriented or arbitrary-oriented text lines in scene images remains difficult and only partially resolved. I categorize text detection algorithms into two major genres: (1) traditional methods, which do not use CNN or use CNN only as a classifier; and (2) contemporary methods, which adopt CNN to a greater degree and can be further categorized based on the specific designs they use.

1.1.1 Traditional Methods

Early work on scene text detection focused on sliding window-based approaches. Chen et al. [5] proposed an end-to-end scene text reading system using the AdaBoost algorithm for aggregating weak classifiers trained on several carefully designed, handcrafted features into a strong classifier. They used a sliding window to find candidate text regions. However, the scales of text lines in scene images vary significantly so sliding-window-based methods are typically very inefficient.

Region-based methods subsequently began to receive more attention. Most traditional region-based approaches can be classified as either: (1) Stroke width transform (SWT) [6] and its variants [18, 19]; or (2) Extreme region (ER) detector [7, 10] and maximally extreme region (MSER) detector [20].

SWT explicitly explores the assumption that text consists of strokes that have nearly constant width. It begins by detecting edges in an image and attempts to group pixels into regions based on the orientation of the edges. However, its performance suffers significantly when the text is set against a low-contrast background or is in an unusual font style.

ER-based methods [7, 10, 21, 20] are more computationally efficient than SWT and achieve high recall. Neumann et al. [7] proposed an ER-based text detection algorithm utilizing several carefully designed region features such as the hole area ratio and Euler number. However, heuristically designed region features are not sufficiently representative to eliminate background noise that is similar to the text.
Several other works [21, 22] have followed similar classification patterns. These region features are fast to compute, but they typically lack the ability of robust classification.

CNN has also been introduced in scene text detection. For traditional methods, they majorly use CNN as a powerful classifier to classify each region of interest [10].

1.1.2 Contemporary Methods

As mentioned in [14], reading multi-oriented text lines is a much harder problem than reading only horizontal text lines. Yet the ability to read multi-oriented or even curved text [23] is important in many scenarios.

Traditional methods usually have difficulty handling multi-oriented text for the following reasons: (1) multi-oriented text reading assumes that individual characters can be identified, which is not the case in blurred images or when the characters are connected; and (2) when detecting multi-oriented text lines, traditional grouping algorithms easily find incorrect lines because of false-positive character predictions and confusion in grouping.

Contemporary scene text detectors represent a breakthrough in detecting multi-oriented or even curved text. Generally speaking, we can classify the methods into three categories: [24]: (1) segmentation-based methods, (2) proposal-based methods, and (3) regression-based methods.

Segmentation-based methods [14] use FCN to classify each pixel as text or non-text. Text blocks are extracted from the segmentation map, and complex post-processing is applied to extract text instances. For example, Zhang [14] used low-level region proposal and region grouping after text block extraction to obtain each individual word. Such methods are usually slow and less accurate in separating text instances. Proposal-based methods [25, 26, 27, 15] adapted a state-of-the-art object detector [28, 29] to detect scene text. Proposal-based methods first use region proposal networks to generate text proposals and regress to the correct text bounding boxes with regard to default anchor locations. Special attention usually has to be paid in designing the anchors and scales so as to achieve good performance in text detection. This is because text usually has large aspect ratios and many images contain extremely small text. In addition to the design of anchors and
Figure 1.1: (a) Illustration of relative regression based on default anchor box. (b) Direct regression [1] centered at point P. It directly regress the distance to the four border of the bonding box.

scales, the models use different regression schemes to detect oriented text. Jiang et al. [15] assumed a text bounding box is an inclined rectangle and proposed to regress to the top-left and top-right corners of it as well as to the height of the inclined rectangle.

Regression-based methods [30, 1, 31] adopt a different approach. Instead of regressing from default anchors, they directly regress from each pixel location to the oriented bounding box. They are thus usually more efficient as only one stage is evolved. For example, Zhou et al. [1] proposed modeling the bounding box as an oriented rectangle and predicting the distance from each output pixel location to each of the four sides of the corresponding text bounding box.

Fig. 1.1 is representative of relative regression (proposal-based) schemes and direct regression [1].

In addition to the three major categories of methods, key-point-based method was recently introduced to scene text detection [32]. However, this method is limited to the detection of non-curved text. This category of methods has been less explored than the others, and in this dissertation, one novel design found in this category is discussed.

In Fig. 1.2, we show several images with multi-oriented text annotations. For multi-oriented text, quadrilaterals rather than horizontal rectangles are typically used to represent the locations of the text instances. For curved text, a polygon that captures the location of the text instance is extracted. This is also a difference between generic object detection and scene text detection.
1.1.3 Unique Challenges of Scene Text Detection

Scene text detection has a similar goal as generic object detection as they are both trying to find the locations of specific object(s). However, scene text detection differs from generic object detection in several key aspects:

- The orientation of text plays an important role in describing a text instance. In object detection, we typically use a horizontal rectangular bounding box to describe the location of an object. Each bounding box can be encoded as two corners or four values. However, the location of a text instance is better described as a polygon in order to capture its accurate region. Such a polygon annotation helps in the later recognition stage, as one single horizontal bounding box might contain several text instances that can cause ambiguity in text recognition.

- The features of one character are typically not enough to distinguish the character from a noisy background. For example, characters such as ”l” or ”T” are similar to various background noises, such as bricks or corners. Background context is needed to accurately localize text and remove false positives.
• In scene text detection, there are usually two annotation settings: (1) line-based annotation and (2) word-based annotation. Each has particular difficulties. In line-based annotation, the model is expected to detect each single text line which could have much higher aspect ratios than regular object detection. In word-based annotation, it is usually difficult to split two close words within the same text line.

1.2 Scene Text Recognition

Scene text recognition, which has made great progress over the years, takes the input of a cropped word image and reads the text on it. This task can be naturally tackled as a sequence learning problem and several studies [33, 34, 35, 23, 36] have proposed combining recurrent neural network (RNN) and CNN into a unified framework to read the text on it. The CNN is used to extract representative visual features from the word patch, and the RNN is used to predict the word in a sequence manner.

Several key designs in text recognition include but are not limited to: (1) Ctc loss, which was originally proposed in [37] and which enables the size of the output of RNN to be different from the number of labels; (2) the attention mechanism in RNN [38]; and (3) the spatial transform network (STN) [39] for reading oriented text by first learning to transform the input word patch.

In general, state-of-the-art text recognition methods have already achieved high performance, especially for horizontal text. The difficulties of text recognition are primarily related to two aspects: (1) some words are oriented or even curved, and (2) many languages (e.g., Chinese) have large vocabulary sets. Researchers have designed different models to solve these problems. Baoguang et al. [23] proposed using an STN to transform the input. STN was optimized with the recognition network and could correct the input image patch by transforming the distorted word into horizontally placed word. Xiao et al. [36] and Zbigniew et al. [4] proposed using spatial attention to read distorted text. At each time step, attention is focused on one specific region of the word image and the corresponding sequence of characters is predicted.
1.3 End-to-End Scene Text Reader

Several works [40, 41, 42] have proposed designing an end-to-end scene text reader. The methods they use combine a text detector and recognizer in an end-to-end fashion. The studies use either SSD-like [41] or Faster RCNN-like [40] proposal-based methods as the detection part. Li [40] adopted a sequential attention mechanism for recognition that can only read horizontal text. Convolutional features are shared between the detector and recognizer.

A typical end-to-end scene text reading pipeline is shown in Fig 1.3.

![Figure 1.3: A typical end-to-end scene text reader built with CTC text recognizer.](image)

1.4 Scene Text Verification/Retrieval

Scene text retrieval [43, 44, 45] is a highly application-oriented research area. Mishra et al. [43] first proposed studying the problem of retrieving images with contained text information. Scene text verification is the most critical part of scene text retrieval. The goal is to design a model that indicates whether a text string exists in an image.

The traditional solution to such a problem involves detecting and recognizing each word in the image and simply using text-based matching in the query stage. Such a method is usually less desirable than others for several reasons: (1) End-to-end text reading is still a challenging and unresolved research area. The models are trained with different metrics and might not be well-suited to retrieval scenarios. For example, in image retrieval, it is preferable to return more samples, while in detection and recognition, accuracy is usually more important; (2) The detector
might be trained on different domains of the images, leading to the poor performance of the scene text reader; and (3) Training a scene text detector requires a large volume of well-labeled training data with bounding boxes and transcriptions. It is useful to investigate methods which do not need such well labeled data.

In [43], the researchers proposed an approach without explicit text detection and text recognition. The method first detects characters and then scores the image based on the set of the dictionary and the detected characters. Order and location constrain is added in the scoring part. Recent work [45] has led to the design of a model that uses SSD [29] to generate a text bounding box and at the same time generates a compact representation of the text for searching.

1.5 Datasets

1.5.1 Real Labeled Datasets

There are competitions and real datasets available for scene text understanding. We give a brief introduction to the datasets here.

1.5.1.1 ICDAR Datasets

ICDAR 2013 Focused Scene Text Detection Dataset. It contains 229 and 233 training and testing images, respectively. All the text is horizontal or close to horizontal, and horizontal rectangles are annotated for each word in a given image.

ICDAR 2015 Incidental Scene Text Detection Dataset. This dataset contains 1,000 training images and 500 testing images. Images are taken using portable devices with motion blur. The text is multi-oriented with quadrilateral annotation.

1.5.1.2 Street View Text (SVT)

The SVT dataset contains 249 testing images used for evaluation. The images are from street views. The dataset was first introduced by Wang et al. [46].
1.5.1.3 UberText

The UberText [47] dataset is a newly released dataset with images taken from street views. It contains both 1K and 4K resolution images with training, testing, and validation splits. The training set contains 16,927 images, and the testing set contains 10,157 images.

1.5.1.4 MSRA-TD500

MSRA-TD500 [18] is an early presented dataset with multi-oriented text. The images include both English and Chinese scene text with a split of 300. It focuses on text line-based detection so there are many long text. The annotation contains a horizontal rectangular bounding box as well as the orientation angle of each text instance.

1.5.1.5 ctw1500

ctw1500 dataset [48] is also a newly released dataset focusing on irregular text detection. It contains 1,000 images for training and 500 for testing. Text bounding boxes are annotated as polygons with 14 points.

1.5.1.6 TotalText

TotalText [49] is a newly released dataset containing polygon annotations with 1,255 training images and 300 testing images. It focuses on the evaluation of text detection for irregularly shaped text.

1.5.2 Synthetic Data

Synthetic data [50, 51] has become one of the major sources of training data for scene text understanding. For the scene text detector, synthetic data are usually used in pretraining [52]. For the scene text recognizer, synthetic data provide far more varieties of text in terms of the dictionary of words it can provide and models trained with it could achieve state-of-the-art performance [53] using real testing data.
1.6 Summary of Contributions

This dissertation summarizes scene text understanding studies, with a focus on two particular topics: scene text detection and scene text verification. It investigates models that use CNN in different ways. The research contributions are the following:

- A scene detection approach using heuristic region proposal methods with a specifically designed CNN classifier. [11]. The classifier aims at classifying each possible region by its local background context for better accuracy.

- A scene text detector that uses a fully convolutional neural network (FCN) to extract text blocks and further uses instance-aware segmentation for text instance detection. The proposed approach can tackle arbitrary-oriented text.

- A scene text detector algorithm framework: detect, associate, and segment (DAS). A model based on such a design is proposed and is found to achieve state-of-the-art performance for both multi-oriented text and arbitrary-oriented text datasets.

- A novel training framework for a scene text detector. The framework improves on a state-of-the-art scene text detector by simply adding an auxiliary task in a cascaded fashion.

- A text verification approach that successfully tackles millions of image data and outperforms other designs, especially the traditional text detector and recognizer framework, by a large margin.

1.7 Dissertation Structure

This dissertation summarizes my research on scene text detection and verification. Chapter 2 introduces a scene text detector that aggregates local context information for better text-nontext region classification. Afterwards, region grouping is used to form text lines. This could be categorized as a traditional region-based method.
Chapter 3 introduces a text detector that uses FCN for text block extraction. A novel text center line network and an instance-aware segmentation network is presented to further segment text instances. This is a modern segmentation-based method that can handle arbitrary-oriented text.

Chapter 4 introduces a detection framework (DAS) representing detect, associate, and segment. The framework is general and novel. It aims at detecting text with arbitrary orientations (e.g., curved text). A key-point-based model under such a framework is designed and achieves state-of-the-art performance with various datasets.

Chapter 5 introduces a training framework for scene text detection. It introduces a novel concept - contour segmentation in the scene text area - and demonstrates that by adding this contour segmentation in a cascaded manner in scene text detection, the trained detector performs better.

Chapter 6 introduces a novel end-to-end model for scene text verification. The model takes the input of a text string and an image pair and directly predicts the probability of that text string existing in the image. It successfully improves upon existing solutions by a large margin with a large-scale street view dataset.

Chapter 7 concludes the dissertation and discusses possible future directions for research on scene text understanding.
Aggregating Local Context for Accurate Scene Text Detection

2.1 Introduction

The strong power of CNN makes scene text detection more robust to outlier noises. The typical pipeline for traditional methods is to first generate a set of region proposals and then apply a binary classifier trained on text-nontext data. Although these methods are efficient, a CNN classifier trained on millions of samples is not stable for the robust scene text classification of complex scenes. This is because context is often necessary for disambiguation. For example, in order to distinguish between the character “I” and the character “1” with background noise like the space between bricks or at the edge of a window, context is crucial. Fig. 2.4 (a) shows some examples of text-like background noise that is confusing when appearing without context.

Zhu et al. [54] proposed using highly semantic context for text detection. The researchers explored the assumption that text is typically on a specific background, such as a sign board, but seldom on others, such as the sky in a natural image. However, modeling text that potentially exists in a wide variety of places is impossible to do in an exhaustive manner. It is also unknown whether a new class of semantic objects that does not appear in the training set harms the results. In addition, in a lot of images that are not purely natural, text may be placed in
unusual places (e.g., sky). In those cases, the model might hurt the performance.

In this chapter, a text localization algorithm is proposed. This algorithm effectively aggregates local context information in detecting candidate text regions. A region’s local context is defined as its horizontal surroundings.

The basic idea is that the surrounding information of a candidate region usually contains strong cues about whether the region is text or is background noise similar to text. The surrounding context thus should help the model localize text more accurately. Some examples of these context images are shown in Fig. 2.4 (b). I also propose a grouping algorithm to form lines from verified regions, as well as a line refinement step to extend text lines by searching for missing character components and to regress text lines to obtain accurate bounding boxes.

To be more specific, the contributions of this chapter are the following:

1. A method that effectively aggregates local information for cascaded and accurate classification of proposed regions. This step could be part of any other traditional region-based framework.

2. An effective grouping algorithm as well as a novel text line refinement step. Text line refinement includes a Gaussian Mixture Model (GMM)-based text line extension module to find new character components, and a sliding window-based oriented bounding box regression and filtering module. They are efficient and robust for post-processing and give an accurate oriented bounding box instead of a mere horizontal bounding box.

3. A cascaded end-to-end traditional detection pipeline for accurate scene text detection in unconstrained images and experiments on several benchmark datasets.

The proposed method is described in detail in Sec. 2.2, and the experimental results are shown in Sec. 4.5. Several text detection results from images using the proposed algorithm are shown in Fig. 2.1.

2.2 Methodology

The proposed detection pipeline is as follows. First, an ER detector is conducted on 5 channels of an input image. For each detected region, the model first classify it
to get a coarse prediction and filter out most non-text regions. Then local context is aggregated to classify the remaining regions in order to obtain a final prediction. Text lines will be formed on top of a character component graph by grouping the verified regions with similar properties together. A text line refinement step is also designed to further filter out false positives and obtain accurate oriented bounding boxes. Several successive image examples of the proposed pipeline are in Fig. 2.2.

### 2.2.1 Cascaded Classification and Local Context Aggregation

Context information is critical for object detection and recognition [55, 56]. Previous traditional region based methods often focus on classifying each region independently and the image patch is cropped tightly from a generated region [10]. Here, the local context of a given region is defined as its horizontal local surroundings and I argue that surrounding information should be incorporated in determining whether a given region should be classified as text or not.
It is easy observe that characters, which are often represented as simple shapes such as “I” or “l”, cannot be well distinguished from background noises that are similar to them. However, text in an image is often represented as lines of characters, and for a given region, its local surroundings give rich information about whether it is text or not. There are works that explored relation between text regions such as [21, 22]. They proposed to use graphcut on top of MSER region graph to refine the results. Instead, I try to aggregate more higher level information in classification step of each region by the proposed network. Some background regions, which are difficult to be distinguished from text when cropped from a tight bounding box, can be accurately predicted by the proposed model after aggregating this context information.

**Design Rationale:** The architecture of the proposed framework is shown in Fig. 2.3. This network is called *text local feature aggregation network* (TLFAN). This is a two-column CNN with joint feature prediction on the top. It is designed for cascaded classification that will be explained in the next part. One CNN branch with fully connected layers is for coarse prediction, and I refer to this branch as standard CNN. The other branch takes an input with aggregated local information to generate a context vector, and I refer it as context CNN. The first column of
the architecture is for learning features from the tight image patch the model is focusing at, and the other is for learning features from its surroundings.

This CNN structure is specifically designed for scene text reading and several design rationales are here: (1) local context typically provides rich information about whether a specific region is text or not. Some background noise can not be well distinguished with text from a mere tight bounding box. In Fig. 2.4, there are some example image patches cropped from IC13 where traditional text/non-text binary classification will easily fail. The proposed model, by aggregating local context, can robustly distinguish it from text. (2) Since the input to the proposed TLFAN and its context have the same scale, both CNN can share parameters and thus reduce the number of parameters that need to be learned. I also tried to use central surround network [57] which will consider a larger surrounding region. However, by doing so, more parameters either in CNN part or in fully connected part needs to be learned. This will not improve the performance as much as the proposed manner, and it is likely to cause overfitting, since it considers information that is mostly unrelated.

Figure 2.3: The proposed TLFAN architecture for scene text detection. Left: CNN structure of the proposed network. The bottom 3 layers of convolution and pooling are shared, and for context branch, another CNN layer and pooling layer is added to produce deeper representation. (b) The whole architecture of the network. One column is for the given patch that the model tries to classify. The other is for extracting context information for this patch, and the generated feature vector will be further used to give an accurate prediction of the region.

**Region Proposal and Cascaded Classification:** I use an ER detector as region proposal because of its efficiency and robustness. Extreme regions are extracted from RGB, Gray scale and gradient of intensity channels. In order to achieve high recall, approximately thousands of regions will be generated from the
Figure 2.4: Cropped image patches which demonstrate that local context helps in distinguishing between background noise and text. (a) The original image patches cropped tightly from the generated regions. (b) The horizontal context images corresponding to the regions on the left. All the examples here are background noises which easily cause false positive if only a tight bounding box is considered for classification. Instead, the proposed model can effectively aggregates local context and give accurate prediction.

5 channels for each image. I preprocess each region as described in [10] and resize them into $32 \times 32$. The standard CNN branch is then used to remove false positives in a similar manner as [10]. For regions with aspect ratios larger than 3.0, sliding window will be conducted on top of it since each region might contain blurred text with several characters connected. In experiment, **91.5%** of the regions will be removed, and it achieves **92%** recall on the IC13 testing set.

After this step, the retained regions will be passed into the context branch to generate context vectors. To be more specific, I calculate the width and height of the region, and extend the patch in the horizontal direction so that the resulting context input patch is with 3 times the width of the original image patch. Mean value is padded when there is not enough space in the image for the context input. Because of the strong ability of CNNs in extracting high level information, this context vector provides rich cues in the classification of the given region. This step can further remove some false positives that are similar to text and only regions with high confidence of text will be retained. In IC13 testing set, **94.5%** regions will be removed and it achieves **91%** recall.

In the final prediction, the two column structure is still used instead of only the context branch for tow reasons: (1) The generated feature vector of standard CNN already contains rich information and it is the feature of the major region for classification. (2) It will be much easier to train since the standard CNN already
produce a really meaningful result. For the context column, they only need to 'figure out' that in some certain cases, the input is not a text even though the feature produced by standard CNN “says” that it is close to text. Such cases include, but not limited to, repetitive patterns, corner of objects and so on.

**Training:** The proposed model is more difficult to train than a simple text/non-text classifier. I follow the same manner as described in [50, 58] by synthesizing image patches for training which provides unlimited number of training data. Since the proposed architecture needs context information, the synthetic positive images need to cover different situations that will happen in real natural images. These include characters with one or two near neighboring characters. Randomly cropped images with their context from several image sources will be considered as negative samples. Several example images for training are shown in Fig. 2.5.

In order to train a better classifier, a two-step training scheme is introduced. First I train a character recognizer with negative samples. This is a 46 classes classification problem, and the positive 45 classes contain all 10 digits and letters with both capitalized or lower cases. Here I merge several similar shaped characters into one class. For example, 'O' and 'o', '0' will be merged into one class.

The model is trained with negative log likelihood as the loss function:

\[
NLL(\theta, D) = - \sum_i \log(Y = y^i|x^i, \theta)
\] (2.1)

and the 46 classes training makes the learned filters better than binary text/non-text training.


Table 2.1: The table of several merged classes. The characters’ shapes are almost the same to each other when considered in small image patch in each cell of the table. I merge them into one class.

The parameters of the trained convolutional layers are then used to initialize the proposed TLFAN architecture and only the fully connected layers will be tuned. After the loss become stable, I train the two parts jointly with smaller learning
rate for finetuning. In addition to this, it is necessary to collect harder examples, and I will explain in Sec. 2.2.2.

Fig. 2.6 shows several images demonstrating the effectiveness of the proposed network. It can effectively use local context to determine whether a given region is text or not, and thus make the prediction more robust. The generated saliency image is the raw output by sliding the classifier on the whole image. Note that even a well-trained text/non-text classifier [2] has problem when background is noisy. However, by aggregating local context, the proposed model gives much more robust performance.

Figure 2.6: The comparison of performance between a text/non-text classifier proposed in [2], and the proposed method in two challenging image in ICDAR 2013 test set. From left to right: (1) original image, (2) the saliency image generated by [2], (3) the saliency image generated by the proposed method and (4) final detected text lines by the proposed pipeline.
2.2.2 Hard Example Mining

In this section, I describe the process of collecting hard training samples. This could also be seen as a way of bootstrapping.

Mining hard examples is a critical step for training accurate object detector, and here I focus on hard negative examples. One of the reasons is that most training examples cropped from negative images have limited geometric patterns. Training on these negative examples will make the model less robust to noisy backgrounds. So here I collect more hard negative training data from two sources: (1) ImageNet: I specifically collect images from several challenging topics, such as buildings, fences, and trees, and windows. These are objects that typically cause troubles in text detection, since their shapes are close to text. (2) Synthetic hard negative samples: I also synthesized a large bunch of negative samples. These samples are not texts but with similar structures as texts, such as stripes and cluttered rectangles. I follow a typical, iterative manner by training until the loss becomes stable and testing on these data to collect hard examples.

I found that this step improved the robustness of the text detector, especially on hard testing images. These hard examples are also used in the later part for line refinement training.

2.2.3 Character Component Graph

Grouping text lines from regions is conducted after classification. This step is critical for traditional region based method since each region is supposed to be one single character. Previous methods [6] typically used relatively heuristic rules to connect text components. This will cause troubles when there are false positives regions. Here I treat it as an assignment problem on top of a character component graph, and the best assignment without conflicts will be chosen based on scores calculated by several text line features as well as several empirical and useful selection standards.

In this section, I first describe the ways the graph is built. Then I discuss how to optimize on the assignment of each character component. The proposed algorithm is illustrated in Fig. 2.7.

Character Graph I first build a connected graph on top of the extracted re-
regions. Each node represents a verified region, and each edge represents the neighborhood relation of the text components. A function $Sim$ is then defined which calculates the similarity between two regions with several low level properties:

1. Perceptual divergence $p$, which is calculated as the KL divergence between the color histogram of two regions: $\sum_i x_p(i) \ast \log \frac{x_p(i)}{y_p(i)}$ where $x$ and $y$ represent two regions and $x_p(i)$ represent the $i$th entry in its color histogram.
2. Relative Aspect ratio: $a = \frac{x_{\text{AspectRatio}}}{y_{\text{AspectRatio}}}$.
3. Height ratio $h = \frac{x_{\text{Height}}}{y_{\text{Height}}}$.
4. Stroke width ratio $s = \frac{x_{\text{StrokeWidth}}}{y_{\text{StrokeWidth}}}$, which is calculated by distance transform on the original region.

Logistic regression is trained on these four region features extracted from IC13 training set to determine whether two given regions are similar. For each region, I further define its neighbor as: $y \in N(x)$ if $\text{Dis}(x, y) < \theta_1$ and $Sim(x, y) > \theta_2, \forall x, y \in R$, where $R$ represents all the regions that have been verified by previous process, and $N(x)$ represents neighbor of region $x$. $\text{Dis}$ means the distance between the center of the two regions. In my experiment, I set threshold $\theta_1$ as $3 \ast \text{Max}(x_{\text{Height}}, x_{\text{Width}})$ where $x_{\text{Height}}, x_{\text{Width}}$ means the height and width of the region. $\theta_2$ is set to 0.5 to filter out regions with less probability as being its neighbor.

**Stable Pair** A stable pair is defined as a pair of regions $x$ and $y$ where they belong to neighbor of each other, and they have a similarity score $Sim(x, y) > 0.8$. In addition, their distance should be no more than twice the shortest distance from all other neighbors to the region. This definition aims at obtaining more “probable pairs”, since only pairs which “prefer” each other as their neighbors will be considered as “stable”. After going through all the regions in $O(n^2)$ time complexity, a set of stable pairs are obtained. Outliers are typically not able to form a stable pair with real characters, since real character will not prefer an outlier as its neighbor.

In the first row of image in Fig. 2.7, the defined stable pair criterion successfully prevent generating vertical lines as possible candidate lines.

**Optimization** In order to optimize the assignment of each region to one of the lines, for each stable pair, I estimate its orientation based on their center points, and then conduct a greedy search algorithm to find components that align with
the current line.

Note that it is possible to find conflicting lines because two lines might share the same region components. In order to resolve the conflict, a score is calculated for each line based on the following properties: the average, standard deviation, max value, min value of the pairwise angle, perceptual divergence, size ratio and distance of neighbor components along the line. These 16 features in total will be calculated and a random forest is trained to give each line an alignment score.

For conflict alignment, I will choose the the best assignment. Here, several empirical but useful standards are also applied. For example, assignment which creates more long lines will be preferred than assignment which creates more short lines as shown in Fig. 2.7. This step aims at resolving the different possible alignments and find true text lines.

![Figure 2.7](image.png)

Figure 2.7: The proposed algorithm which could effectively resolve conflicted candidate lines and find best line assignment of text regions. The final detected lines could be in any orientation. From left to right: (a) The constructed character component graph. (b) A set of generated stable pairs which will be used to create candidate lines. (c) Candidate lines represented as different colored bounding boxes. (d) Detected text lines.

### 2.2.4 Line Extension and Refinement

After generating lines of regions, a line refinement step is taken to finalize the results. This step aims at two targets: (1) extending lines so as to find missing components. (2) filtering out false positive and predict a tight oriented bounding box. They aim at finding a better bounding box that cover the whole word. This is important for an end-to-end system which incorporates text recognition since the performance of recognition highly relies on accurate bounding boxes.
Gaussian Mixture Model Based Line Extension  Even with carefully designed classifier and grouping algorithm, there still might be some letters in a line that are not found in the previous steps. Here I propose a simple model to recover the proposed lines.

The model is based on the assumption that lines of characters typically have different color distribution with its direct background, and characters in one line typically follow the same color distribution. The algorithm pipeline is shown in Fig. 2.8.

For each line that has been found by previous approaches with more than 2 regions, I crop the patch from the lines and estimate a GMM on the color of the patch. The Gaussian component associated with foreground (text region) is estimated by a voting mechanism: (1) I calculate the skeleton from the region patch, (2) for all pixels in the skeleton patch, the Gaussian component it belongs to is calculated by a simple voting mechanism conducted among these pixels. The reason for only using skeleton pixels lies on the fact that pixels that are close to the boundary are not accurate enough for color distribution estimation.

For each line, I consider an square image patch whose side length is twice the height of the line, with its location on the two end of the lines. I estimate the color distribution in the region and a MSER detector is conducted on top of predicted color probability image. MSER regions whose size is too large or too small when compared to the height of the line are filtered out. Then the retained region is classified as being a text or not using the standard CNN branch in TFLAN. If its probability is larger than 0.4, then the model will group it into the lines. If one additional character is found, the GMM is updated and the algorithm will tries to find more characters until no more character could be found.

Previous methods [21, 22] used a graphcut algorithm on top of the extracted regions which serves as similar purpose. However, if the graphcut is directly conducted on all the verified regions, it is still in doubt that whether it will also create more false positives. Here I use a more conservative method and only consider regions which could be easily attached to the current line that has already been found.

Line filtering and Sliding Window Regression In this section, I propose a joint line filtering and regression model. The model is based on making prediction
Figure 2.8: The proposed line extension pipeline: (a) the original image with the detected lines, (b) cropped line image patch the model used for estimation of GMM, (c) the skeleton of all the region components, (d) cropped image patch whose color distribution needs to be estimated, (e) predicted color probability image, (f) MSER result on the estimation patch, (Because of the large contrast in the predicted image patch, there are only few bounding box that I need to verify) (g) regions after running the standard CNN branch and non-maximum suppression and (h) The final detected line.

in a sliding window manner on all the text lines that have been verified in the previous steps. Existing methods [58] typically have two steps: (1) word-level verification. (2) bounding box regression. A CNN model is used for filtering and regression. Since fully connected layer only takes fixed sized input, the image patch needs to be resized in order to fit into the network. However, the length of text lines could vary according to the font, and number of characters. It is not desirable to resize an image patch with a text line of 2 to 3 characters to the same size as a text line with 10 characters. Instead, I consider joint regression and filtering in a sliding window manner.

The proposed architecture is shown in Fig. 2.9(a). The CNN is taken directly from the previous detection architecture. The proposed CNN model takes an input of 48×64 color image patch, and gives 7 prediction. One prediction is a simple part-of-word/non-word prediction which predicts whether the input patch containing part of word, or several characters. It is trained with negative log likelihood loss. The rest of 6 values all represent vertical coordinates because the model is doing inference in a sliding window manner along the text line. Two of them are the minimal and maximal vertical values of the text in the current patch, and they are the same as the vertical coordinates that are predicted in traditional bounding box regression. The other four values represents the minimal and maximal vertical values in left and right side of the patch which are used to predict an oriented
bounding box. Some training examples are shown in Fig. 2.9(b). The regression model is trained with standard mean square error loss: \( \text{MSE}(x, y) = -\sum_i (x_i - y_i)^2 \). where \( x \), \( y \) represent the predicted coordinates, and the ground truth coordinates, respectively. By predicting these four values, an oriented bounding box could be estimated. Since there are only a few lines in an image, even sliding window will not need much computation.

Several predicted results have been shown in figure 2.9(c) on images cropped from IC13 and SVT data sets.

![Figure 2.9: The proposed line refinement illustration](image)

- (a) The proposed architecture which used the CNN from detection part. Training is only on the fully connected layers for classification and regression.
- (b) Several examples of training images. The red lines are drawn from minimal and maximal vertical coordinates. The green dots are the vertical coordinates for oriented bounding box.
- (c) Several testing result images. The oriented bounding box is drawn with green lines instead of dots for better visualization of the orientation.

In order to refine the text lines based on the proposed architecture, I first crop the text line patch from the image, and resize the height of the text line to 48. A sliding window of height 48 and width 64 on the cropped patch. Background noise lines will be filtered out by part-of-word/non-word classification. If the patch is predicted as part-of-word by the classifier, the oriented bounding box regression would be conducted. A step by step example is shown in Fig. 2.10. Here I only show lines with text on them.
Figure 2.10: The proposed line refinement pipeline. (a) For each cropped lines, it does sliding window prediction and merge the results. (b) Several line regression results based on the proposed framework. The red lines correspond to standard regression, and the green lines represent oriented regression.

2.3 Experiments and Evaluation

In this section, the evaluation of the proposed method on several benchmark datasets is conducted. Precision, recall and F-measure scores are reported on the detection results.

Implementation Details: I implemented the algorithms in python and torch 7 [59] on a workstation with 64G RAM and Nvidia GPU tesla k40 and 16 processors (3000MHz). All the generation of region proposals and post processing with different channels are parallelized.

ICDAR Robust Reading: For IC13, it provides an online evaluation system where I evaluate the proposed method. For IC03, I evaluate result according to the metric in [60]. The results are shown in Table. 2.2. Evaluation shows that the proposed algorithm gives good performance in both data sets.

Street View Text: One of the problems of SVT dataset is that it is not fully annotated: some text in the image are not included in the annotation. This problem has been mentioned in [58], and I call this annotation as partial annotated. The proposed algorithm could efficiently detect most of the text in images and thus the unlabeled text will decrease the precision of detection result and makes it hard to compare with other methods. So I manually labeled all the text in the images following simple rules: (1) text is not too blurry to read by human, (2) it contains more than 2 characters. I call this version fully annotated dataset and I
<table>
<thead>
<tr>
<th>method</th>
<th>precision</th>
<th>recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neumann [7]</td>
<td>73</td>
<td>65</td>
<td>69</td>
</tr>
<tr>
<td>Shi [22]</td>
<td>83</td>
<td>63</td>
<td>72</td>
</tr>
<tr>
<td>Bai [61]</td>
<td>79</td>
<td>68</td>
<td>73</td>
</tr>
<tr>
<td>Zamberletti [20]</td>
<td>86</td>
<td>70</td>
<td>77</td>
</tr>
<tr>
<td>Tian [9]</td>
<td>85</td>
<td><strong>76</strong></td>
<td>80</td>
</tr>
<tr>
<td>Zhang [62]</td>
<td>88</td>
<td>74</td>
<td>80</td>
</tr>
<tr>
<td>Proposed no post</td>
<td>89</td>
<td>73</td>
<td>80</td>
</tr>
<tr>
<td><strong>Proposed</strong></td>
<td><strong>90</strong></td>
<td><strong>75</strong></td>
<td><strong>81</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>method</th>
<th>precision</th>
<th>recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Li [21]</td>
<td>79</td>
<td>64</td>
<td>71</td>
</tr>
<tr>
<td>Yao [18]</td>
<td>69</td>
<td>66</td>
<td>67</td>
</tr>
<tr>
<td>Kim [63]</td>
<td>78</td>
<td>65</td>
<td>71</td>
</tr>
<tr>
<td>Yi [64]</td>
<td>73</td>
<td>67</td>
<td>66</td>
</tr>
<tr>
<td>Epshtein [6]</td>
<td>73</td>
<td>60</td>
<td>66</td>
</tr>
<tr>
<td>Zamberletti [20]</td>
<td>71</td>
<td><strong>74</strong></td>
<td>70</td>
</tr>
<tr>
<td><strong>Proposed</strong></td>
<td><strong>84</strong></td>
<td>70</td>
<td><strong>76</strong></td>
</tr>
</tbody>
</table>

Table 2.2: Localization performances on: top: IC13 (%), bottom: IC03 (%) data sets. Bold number outperforms previous methods. 'Proposed no post' represents final results without line extension and refinement steps.

tested the proposed algorithm on both versions of the dataset for evaluation. The performance is shown in Table. 3.3. Fig. 2.11 illustrates several examples of partial annotated dataset, fully annotated dataset as well as detection results. Experiments on the fully annotated dataset shows that the proposed detection algorithm have good performance in SVT dataset as well.

|                   | partial annotated | fully annotated |
|---                | recall | precision | recall | F-measure |
| jaderberg et al. [58] | 0.71   | -        | -      | -        |
| Proposed          | **0.75** | **0.87** | **0.73** | **0.79** |

Table 2.3: Text detection performance on SVT. The bold results outperforms the previous state-of-the-art results. (1) partial annotated: detection recall measured with the partial annotation. The accuracy here does not makes sense, so I only tested its recall. (2) fully annotated: detection precision, recall, F-measure with full annotation.

**Limitation** The proposed method achieved fairly good results in terms of precision, recall, and F-measure on standard datasets. However, it can still fail on
several extremely challenging cases: (1) Text lines that are too blurry will cause problem in accurate region proposal generation as well as classification. (2) Strong reflection, too low contrast will still cause troubles in the detection. (3) Curved text lines might cause incomplete detection. Fig. 2.12 shows several failure cases that the proposed algorithm cannot get good results.

Figure 2.11: For each pair of images, left: the original incomplete annotation, right: detection result of the proposed model as well as the fully annotated ground truth. The fully annotated dataset provides oriented bounding box annotation. Green boxes represent the detected result which matches the ground truth. Yellow boxes represent the ground truth.

Figure 2.12: Example images where the proposed model failed to detect all the lines or detected the wrong lines. The green boxes are the text lines that are correctly detected. The blue boxes are text lines that the proposed model fail to detect, and the red boxes are false positives, or incomplete detection.
2.4 Conclusions

In this chapter, I proposed a novel scene text detection algorithm. The algorithm is based on traditional region-based method design with a module to efficiently aggregate local context information into detection as well as a text line refinement. The method performs well on a standard horizontal text benchmark dataset.
Chapter 3

Multi-scale FCN with Cascaded Instance Aware Segmentation for Arbitrary Oriented Word Spotting In The Wild

3.1 Introduction

Figure 3.1: Scene texts that have been successfully detected by proposed systems.

In this chapter, I introduce an algorithm that uses FCN in scene text detection. The algorithm runs in a cascaded fashion, meaning that it can handle true arbitrary-oriented text. It has two cascaded levels. At the whole image level, it uses a multi-scale FCN to extract representative features and remove most nega-
Figure 3.2: The pipeline of the proposed algorithms with demonstration of the two level cascades. Multi-scale FCN is used to extracts text blocks. Then for each extracted text block, it predicts the text center line by the proposed TL-CNN. After extracting all the text lines, IA-CNN is used to extract each instance word. The pipeline could extract text line in arbitrary orientations.

At the text block level, I design a text instance segmentation network to obtain each text instance from the text block. The challenging text instance segmentation task is divided into two easier sub-tasks in a cascaded fashion, as inspired by [65]: (1) I first find the text center lines by training an FCN that predicts the center line of each instance word. (2) Then each single text center line is extracted from the previous output. (3) Finally, the extracted text line information is appended to the text block image as an instance clue to further extract each text line segmentation. Only simple low-level processing is needed in order to achieve instance-level (word or text line) segmentation.

Fig. 5.1 shows several examples of end-to-end results. The proposed pipeline,
which contains both the text block FCN and the word instance segmentation, is shown in Fig. 3.2. Additional results can be found in Appendix A.

In summary, the contributions of this chapter are the following:

1. A unique instance segmentation-based model for obtaining word instance. This model breaks the task of instance segmentation into easier tasks, achieving a good performance in separating word instances in a cascaded fashion. The text instance segmentation model has several advantages over traditional methods: (1) it is invariant to orientation, (2) it is able to find text instances even when characters are connected, and (3) it is able to separate text lines that are close to or even touching each other. The step of obtaining text instances is crucial for end-to-end text reading since current scene text reading methods can only read a single word or line. The proposed algorithm, as the first attempt at designing an instance segmentation model for scene text detection, should be of great value in future research.

2. A multi-scale FCN model for scene text block detection. The model can help identify text block regions with large-scale variances and combine more context information for each prediction. The text block FCN aims at removing most false-positive regions by extracting multi-scale feature representations. It serves as the first step of the cascaded framework.

3. Thorough evaluations have been conducted on several benchmark datasets, including the challenging IC15 and the curve text-based CUTE80.

The remainder of the chapters are organized as follows: In Sec. 3.2, the proposed model of a multi-scale, shared-net FCN is presented in detail. In Sec. 3.3, the model of word instance segmentation is described. Experiment information and the conclusion are located in Sec. 3.4 and Sec. 3.5, respectively.

This work has similar features to that of Zhang et al. [14], in that both use an FCN and treat the detection problem as a segmentation problem. However, this work differs from the other researchers’ work in several key aspects that ultimately make my proposed algorithm more robust and general: (1) Instead of using proposal and low-level line orientation estimation, I design a novel instance segmentation scheme for separating text lines. I do not make any assumptions about
the text orientation within each text block, and only a few low-level processing steps are needed. The proposed model can handle arbitrary-oriented text lines, including curved text lines which cannot be handled by Zhang’s method. (2) I use a multi-scale, shared-net FCN to capture more context information and text with larger variances of scales, which leads to better text block detection results.

The work falls into the category of cascaded methods for text detection [66, 67].

### 3.2 Multi-Scale, Shared-Net FCN

#### 3.2.1 Design Rationale

Fully convolutional neural network (FCN), which was originally presented in [68] for scene labeling, has been adopted by [14] in scene text detection. One problem that needs to be carefully considered is scale of the object, and several variants have been proposed to solve the problem [69, 70].

In [14], Zhang et al. tried to capture multi-scale information in an image using a single branch FCN with a skip-net model. However, it is still in doubt that text, which potentially varies a lot in terms of its relative size to the image, could be well captured without larger context information. In IC15 training sets, I calculate the
relative scale of the text line, which is defined as the shorter length of its oriented bounding box divided by the height of the image. The relative scale could vary from 0.005 to 0.78. This means that a robust algorithm should be able to capture text in a wide variety of scales. Let’s assume that the input image in testing has a height of 500 pixels. Then the height of a text line could vary from 5 ($0.01 \times 500$) to 160 ($0.32 \times 500$). I argue that a single FCN is not enough to accurately capture such large variances of text.

In addition to the scale problem, background contexts could effectively improve the scene text localization performance. Similar ideas were proposed in [11, 54], which incorporated contexts in a region proposal framework. Here I have the same hypothesis and claim that it is also helpful for a FCN based scene text detection method and will improve the performance of the pixel labeling problem.

Based on the above two claims, I specifically design a multi-scale, shared-net FCN which has a larger receptive field and can capture more useful context information. It improves the performance of text block detection.

### 3.2.2 Architecture

The architecture of the proposed model is shown in Fig 3.3. There are three branches with shared convolutional parameters. The encoded information, after two unpooling layers [71], is merged to produce the final results.

For each prediction in the final output, it is a joint prediction from all three branches. By joint prediction, it can capture larger context information and give more accurate prediction.

In Fig 3.4, several examples of the performance of the FCN model with comparison with single branched FCN is visualized. It could be seen that a larger context is helpful for both removing false positives and obtaining better responses around text.

### 3.2.3 Training with Per-scale Loss

During training, IC13, IC15 training images and the synthetic images from [51] are augmented by random scaling and rotation. All these data are used for training the FCN model. Note that the training data contain texts in a large variance of
scales.

Training a multi-scale, shared-net FCN is relatively harder than training a single FCN. I follow several works [72, 73, 74] that use a per-scale loss which could make the learned features from multiple scales more discriminative, and thus accelerate training and improve performance. The loss function is described in Equation 3.1.

$$\text{P-NLL}(\theta, D) = - \sum_k \log P(Y = y^k | x^k, \theta) + \sum_{i=1}^M -\alpha_i \times \left( \sum_j \log P(Y = y^j_i | x^j_i, \theta) \right)$$  \hspace{1cm} (3.1)

$M$ represents the number of scales being used. $\alpha_i$ represents the weight for $i$th scale. $x^j$, $y^j$ represent the output and groundtruth, respectively. $\alpha_i$ is initialized to be larger in order to learn discriminative features for each scales. I then gradually decrease their values and focus on the training of the joint prediction.
3.3  Cascaded Text Instance Segmentation

Given a text block, which might contain several nearby word instances, I specifically designed an instance segmentation model to segment each word instance. Instance segmentation has attracted an increasing attention in computer vision community [75, 76, 77, 65]. It is a much harder task than semantic segmentation because it has to separate out different instances of the same class.

In scene text detection, I define an instance as a word or a text line which is not separable purely visually. The input in this step is the cropped text block image obtained from text block FCN, which might contain several lines or words. I propose two networks in a cascaded fashion to solve the problem: Text Line CNN (TL-CNN) and Instance-Aware CNN (IA-CNN). The TL-CNN produces a segmentation that corresponds to the center of each text line. The IA-CNN, by taking the input as one of the text center lines, generates a segmentation mask over that text line instance. This step is crucial, not only for the evaluation of detection performance, but also for combining text recognition into an end-to-end system, since the input to recognition is typically a single word or text line.

In addition to the ability of decomposing each text block, this component also serves as a further step of removing false positives. Both sub-nets are able to further remove certain negative detection. Some examples of the extracted instances from text block images can be seen in Fig. 3.5.

3.3.1  Architecture

The architecture of the instance segmentation network is shown in Fig. 3.6. There are two branches with shared CNN parameters except for the first convolutional layer and all the fully convolutional layers.

The left branch is the TL-CNN. The network has been trained to embed the “instance” information since it has to figure out where the center of each word is. Here I only consider word instance with more than 2 characters, since they are more distinguishable. However, instead of giving a hard threshold of removing detected text line with less than 3 components [14], the network needs to learn the features that correspond to a word instance. I hypothesize that such instance-aware features are complementary to features extracted by previous text block FCN.
Figure 3.5: Results of the instance segmentation model. It can capture word or text line instances in a wide variety of circumstances with arbitrary orientations. The segmented results could be directly thresholded to get the final bounding boxes.

The features extracted from text block FCN are more like traditional “textness” features. These features only capture whether a given region looks like text, but no instance information is embedded, and thus might be misled by some background noise. In Fig. 3.8, I show several examples that the text block FCN easily predicts as positive but are rejected by the TL-CNN.

The right branch is the instance-aware segmentation (IA-CNN) branch, whose input is a 4 channel tensor with size $4 \times h \times w$. $h$ and $w$ correspond to the height and width of the input text block image, respectively. For the input tensor, the first 3 channels are R,G,B channels of the text block image. The 4th channel is the text center line channel corresponding to the instance to be segmented. Jointly, the model can predict instance level segmentation with the two network in a cascaded fashion.

### 3.3.2 Pipeline

The pipeline of the proposed instance segmentation is shown in Fig. 3.7. It decomposes the hard task of instance segmentation into cascaded tasks.
Figure 3.6: The architecture visualization (left) and details (right) of the TL-CNN and IA-CNN. The TL-CNN (left branch) is for producing the instance level words center lines. The IA-CNN (right branch) is for producing each word instance segmentation once. Their inputs have different number of channels since the IA-CNN needs a 4-channel tensor. ReLU layer and batch normalization [3] are ignored for simplicity.

<table>
<thead>
<tr>
<th></th>
<th>VGG-16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv</td>
<td>2,2</td>
</tr>
<tr>
<td>MaxUnpool</td>
<td>512→256,3,3</td>
</tr>
<tr>
<td>Full-Conv</td>
<td>256→256,3,3</td>
</tr>
<tr>
<td>MaxUnpool</td>
<td>2,2</td>
</tr>
<tr>
<td>Full-Conv</td>
<td>256→128,3,3</td>
</tr>
<tr>
<td>Full-Conv</td>
<td>128→128,3,3</td>
</tr>
<tr>
<td>MaxUnpool</td>
<td>2,2</td>
</tr>
<tr>
<td>Full-Conv</td>
<td>128→64,3,3</td>
</tr>
<tr>
<td>Full-Conv</td>
<td>64→16,3,3</td>
</tr>
<tr>
<td>Full-Conv</td>
<td>16→2,1,1</td>
</tr>
</tbody>
</table>

Figure 3.7: The method of instance segmentation. It tries to decompose the hard task of instance segmentation into easier sub-tasks. First TL-CNN is used to generate text center line labeling. Then I simply decompose from the probability map a set of text lines. Each of these text lines will generate an input to the IA-CNN together with the original input image. Then IA-CNN can produce an accurate instance segmentation of each text lines with arbitrary orientation.
Figure 3.8: Examples of images that text block FCN might fail to remove. Instead, TL-CNN can remove them. I only show the output map from TL-CNN of the first image to save space. The output of others are similar (completely black). Text lines can not be extracted from them and thus I can remove them safely.

Given a cropped text block $B$ from the original image generated by thresholding the text block FCN output. I generate a probability map $B_L$ by TL-CNN. Note that an expansion of the text block rectangle of 0.1 of its height and width is used in order to crop some background information, and better for further feature extraction. Each pixel $p_i$ in $B_L$ represents probability of whether it belongs to the center line of one text instance. Once obtaining the output, I simply threshold the output with $T$, which is chosen based on the evaluation that will be discussed later, and do a morphological closing operation followed by a connected component analysis on it. I extract each component $C$ and do the following operations: (1) discard those components whose height and width are less than 0.1 of the height and width of the text block, respectively, (2) discard the text block from which the overall coverage of the extracted text lines to the oriented text block bounding box is less than 0.6.

The obtained text line connected components $C_s$ are separated, and several text line images $I_s$ will be generated from it. Each image $I_i$ has -128 value background and 128 value on the corresponding text line. This channel together with the original text block is padded into a 4-channel tensor. Then this tensor will be used as an input to the IA-CNN branch. The output is the single text line instance
segmentation $I_i$ which corresponds to the input text center line image. Note that, when two extracted instances have high overlap, it is needed to keep the union of them as a post processing step. Even not common, this might happen when the TL-CNN outputs two disconnected lines for one instance. In such case, the pipeline will extract two text center lines and predict two instances. However, the proposed IA-CNN segmentation will know that they are actually pointing to the same instance and the outputs will have high overlap. This can be seen as a way of error handling in the IA-CNN part, and it will be discussed later. After finishing merging such instances, I simply threshold each instance segmentation probability map $I_i$ with 0.5 to obtain the text region and corresponding bounding box.

Fig. 3.5 shows more results. In testing, for each cropped image patch, I resize its larger dimension to be in range of 100 to 150 pixel while keeping the aspect ratio unchanged. The range is chosen for two reasons: (1) Too small size of a image will cause text line instances to be unclear and hard to separate. (2) The FCN, which is initialized with VGG-16, has a receptive field larger than 200. So for each pixel in the output, it has all the context needed to decide “where one instance is”. This is crucial for the TL-CNN to find each text line and give a good prediction.

### 3.3.3 Optimization

In training, I use a simple iterative scheme that iterate between training a TL-CNN and training a IA-CNN. For TL-CNN, all the convolutional layers are initialized with VGG-16 model [71]. For IA-CNN, the shared CNN parts are also initialized with VGG-16 model. All other layers are initialized with zero mean and standard deviation 0.1, Gaussian distribution. Inputs to both branches are normalized to have zero mean. Values of input range from -128 to 127. In optimization, the model iteratively optimize the loss function 3.2.

$$NLL(\theta_1, \theta_2, D) = -\alpha \times \sum_j \log P(Y = y^j|x^j, \theta_1) - (1 - \alpha) \times \sum_i \log P(Y = y^i|x^i, \theta_2)$$ (3.2)
In the equation, $\alpha$ controls which branch of the network is in training. It can only equal to 0 or 1. $x^i, y^i$ represent the prediction and groundtruth, respectively. Note that part of $\theta_1$ and $\theta_2$ are shared. When the loss of two branches become stable, I start to finetune the shared convolutional parts with smaller learning rate in the same iterative manner.

### 3.3.4 Error Handling in Instance-Aware Segmentation

Error handling is an important part of gaining robust performance for the instance-aware segmentation network. This is because in prediction the extracted text line could not be perfect. There might be variances of the line width, and the predicted line might not be centered well. The two end points of the line might have small offset from the center of the two sides. In order to make the model robust, I randomly add noise to the training data. Specifically, line width and the location of the two end points of each line are randomly changed. Under certain constraints, noisy training samples could be obtained, and make the model more robust in testing. Fig. 3.9 illustrates how I create noisy training samples.

In the illustration image, $L_2$ represents the offset from $p_1$ to $q_1$ along the shorter side of the oriented word box, $L_1$ represents the offset from $p_1$ to $q_1$ along the longer side. The length of $L_1$ and $L_2$ and the width of text center line $W$ are defined in equations below it.

The offset from $p_2$ to $q_2$ is processed in the same way as $p_1$ to $q_1$. By doing such randomization on the training set, more robust model can be obtained. Several training examples are shown in Fig 3.14. Note that I also randomly sample negative text center lines that are on background region of a text block. For these text center lines, the corresponding ground truth are masks of all background label. The training set is from synthetic text block dataset and also from [51]. The line information could also be seen as a hint which tells the network where to find the instance, and thus it doesn’t need to be perfect. Some training samples could be seen in Fig. 3.10, 3.11 and negative training samples 3.12.

Fig. 3.13 shows several examples demonstrating the effectiveness of the Error handling module.
Figure 3.9: Illustration of creating noisy training set for IA-CNN network. The bounding box of the word is for illustration purposes. \( \text{Length}_1 \) and \( \text{Length}_2 \) are the length of longer side and shorter side or the word patch, respectively. \( p_1, p_2 \) and the corresponding green line represent the ground truth center line. \( q_1, q_2 \) and the corresponding red line represent the shifted noisy line. \( L_1 \) and \( L_2 \) are the offsets from \( p_1 \) to \( q_1 \) along the longer side and shorter side, respectively.

\[
L_1 = \alpha \times \text{Length}_1 \\
L_2 = \beta \times \text{Length}_2 \\
W = \text{floor}(\gamma) \\
\alpha \sim \text{uniform}(-0.5, 0.5) \\
\beta \sim \text{uniform}(0.0, 0.25) \\
\gamma \sim \text{uniform}(1, 4)
\]

3.4 Experiments

3.4.1 Curved Text

Curved text typically can not be handled in scene text detector [14], and many works do not consider curved text since they have the assumption that text lines are straight. However, many texts in signs or logos are curved and the ability to read curved text is important and will help many applications.

The proposed model can effectively capture the curved text by the joint power
Figure 3.10: Some training samples of the instance segmentation model.

Figure 3.11: Some training samples of the instance segmentation model.

Figure 3.12: Some training negative samples of the instance segmentation model.
Figure 3.13: Some examples showing the error handling of IA-CNN step. From left to right: input text block image, extracted text line instance, raw instance segmentation result, final extracted instance box. Note that, the extracted text instances are not correct, but the model can finally recover the results. In the raw instance segmentation results, each pixel might belong to two instance masks, since the operation of IA-CNN is done separately for each extracted text line. The instance mask drawn later will cover the previous one for the raw instance results shown here.
Figure 3.14: On the top two rows, I show some examples of the noisy training data for instance-aware segmentation. The input line image is shown in the image with a black line crossing the word. Here I only visualize one instance line per image for illustration purposes. On the last two rows, I show examples demonstrating the effectiveness of augmented the noisy training data. From left to right: Input text block image, instance results with model trained on good quality data, instance results with model trained on noisy data. More results about such error handling are in supplementary material.

of TL-CNN and IA-CNN. In Fig 3.15, I show some curved text testing results on CUTE80 dataset [78]. It could be observed that even with extreme curvature, the proposed model can successfully estimate the text center line and further infer the instance mask for each text line.

Another surprising fact is that, no curved training data is needed in the training set for both TL-CNN and IA-CNN. I hypothesize that this is because the model learned the intrinsic representation of an instance line which does not rely on whether it is straight or not.

3.4.2 Evaluation on Instance Segmentation

In order to evaluate the performance of the TL-CNN and the following IA-CNN module, I collect 1500 cropped images from IC13, and IC15 training set. They contain text blocks with 1-5 lines in each image.

Note that this evaluation is meaningful since the training data for the two networks are synthetic images, so these public training sets are used as validation
Figure 3.15: Results of the instance segmentation model on several curved text blocks. It can accurately capture curved text from a lot different scenes. From left to right: (1) The input image. (2) Text line captured by text line model (3) Instance segmentation results on these curved data.

purposes. Fig. 3.16 shows the precision and recall curve.

It can be seen that the choice of $T$ only has little effect on the performance. The evaluation is based on the metric in [60]. Note that this evaluation framework artificially lowers recall. I have found that the relatively lower recall is generally caused by the fact the proposed model usually predicts a line as one instance when there is less visual cue to separate out each word. This has little effect on end-to-end performance because current state-of-the-art recognition model [53] can directly read a line and thus will not hurt end-to-end scene text reading. I use 0.85 as the final choice of $T$ in further evaluations.

I thoroughly evaluate the proposed algorithm on four widely used benchmark
Figure 3.16: The precision (red) and recall (green) curve with respect to the choice of $T$ on 1300 randomly cropped text block image from IC13, IC15 training set.

<table>
<thead>
<tr>
<th>method</th>
<th>precision</th>
<th>recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neumann [7]</td>
<td>73</td>
<td>65</td>
<td>69</td>
</tr>
<tr>
<td>Shi [22]</td>
<td>83</td>
<td>63</td>
<td>72</td>
</tr>
<tr>
<td>Bai [61]</td>
<td>79</td>
<td>68</td>
<td>73</td>
</tr>
<tr>
<td>Zamberletti [20]</td>
<td>86</td>
<td>70</td>
<td>77</td>
</tr>
<tr>
<td>Tian [9]</td>
<td>85</td>
<td>76</td>
<td>80</td>
</tr>
<tr>
<td>Zhang [62]</td>
<td>88</td>
<td>74</td>
<td>80</td>
</tr>
<tr>
<td>Zhang [14]</td>
<td>89</td>
<td>78</td>
<td>83</td>
</tr>
<tr>
<td>tian [13]</td>
<td><strong>93</strong></td>
<td><strong>83</strong></td>
<td><strong>88</strong></td>
</tr>
<tr>
<td>Proposed</td>
<td><strong>93</strong></td>
<td>79</td>
<td>85</td>
</tr>
</tbody>
</table>

Table 3.1: Localization performances (%) on ICDAR 2013 data sets. Bold number outperforms other methods.

datasets: IC13-focused text detection, IC15-scene text detection, CUTE80 and SVT. I choose them based on two criteria: they are widely used for evaluation and comparison or interesting for practical applications. I give a brief description and performance comparison of each dataset separately.

### 3.4.3 ICDAR 2013

ICDAR 2013 dataset is probably the most widely used dataset. I evaluate them by submitting results into the ICDAR system. The evaluation protocol is based on [60]. Results are in Table 3.1.
Table 3.2: Localization performances (%) on ICDAR 2015 data sets. Bold number outperforms other methods. Some methods do not have references.

<table>
<thead>
<tr>
<th>method</th>
<th>precision</th>
<th>recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>HUST</td>
<td>44</td>
<td>38</td>
<td>41</td>
</tr>
<tr>
<td>StradVision1</td>
<td>53</td>
<td>46</td>
<td>50</td>
</tr>
<tr>
<td>StradVision2</td>
<td>77</td>
<td>37</td>
<td>50</td>
</tr>
<tr>
<td>Zhang [14]</td>
<td>71</td>
<td>43</td>
<td>54</td>
</tr>
<tr>
<td>tian [13]</td>
<td>74</td>
<td>52</td>
<td>61</td>
</tr>
<tr>
<td>Proposed</td>
<td><strong>76</strong></td>
<td><strong>54</strong></td>
<td><strong>63</strong></td>
</tr>
</tbody>
</table>

3.4.4 ICDAR 2015

The results on ICDAR 2015 are in Table 3.2. Note that some results are from ICDAR website, so there is no reference for them yet.

3.4.5 Street View Text and CUTE80

SVT dataset [46] contains images taken from street view, and CUTE80 dataset [78] contains texts that are in curved shape. They represent interesting aspects of scene text detection, and are also highly application oriented. Both datasets have the problem that they are not fully-annotated. So here I only evaluate the recall of proposed method in the two datasets. The results are shown in Table 3.3.

<table>
<thead>
<tr>
<th>method</th>
<th>recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaderberg [58]</td>
<td>71</td>
</tr>
<tr>
<td>He [11]</td>
<td>75</td>
</tr>
<tr>
<td>Proposed model</td>
<td><strong>78</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>method</th>
<th>recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tian [13]</td>
<td>60</td>
</tr>
<tr>
<td>He [11]</td>
<td>56</td>
</tr>
<tr>
<td>Risnumawan [78]</td>
<td>68</td>
</tr>
<tr>
<td>Proposed</td>
<td><strong>73</strong></td>
</tr>
</tbody>
</table>

Table 3.3: Text Localization evaluation (%) on SVT (left) and CUTE80 (right) dataset. I only evaluate recall on these datasets because they are only partially annotated.

3.4.6 Limitation

The proposed algorithm can handle text detection in a lot of different and challenging scenes. However, for some certain cases the proposed model will fail. Fig. 3.17
shows some failing results. Extremely low contrast, too blurry text or text lines with scattered characters will cause problem.

Figure 3.17: Example images that the proposed algorithm fail to detect correctly. Blue rectangles mean that the model fails to localize the texts.

3.5 Conclusion

In this chapter, I presented a novel algorithm for arbitrary-oriented scene text detection. I combined a multi-scale FCN with a novel, cascade-style instance segmentation method for end-to-end scene text detection. I demonstrated that instance segmentation, which is becoming increasingly popular among computer vision researchers, is also helpful for end-to-end text reading systems, especially in the segmentation of arbitrary-oriented text.
Chapter 4

Detect-Associate-Segment (DAS): A framework for arbitrary oriented scene text detection

4.1 Introduction

In this chapter, I propose a new framework for arbitrary-oriented scene text detection and describe a keypoint model that I designed based on it. The contributions of this chapter are threefold:

- The proposed framework is novel and general. It is novel in the sense that it differs from previous proposal or regression-based methods. It is also general, and many different pipelines could be built based on it.

- A discussion about the information that could be trained and used to uniquely identify one text instance is presented. Based on this discussion, I propose two model prototypes for detecting arbitrary-oriented text, one of which is text center line-based and the other of which is keypoint-based.

- Designs of a keypoint model based on the proposed framework for arbitrary-oriented scene text detection are presented. Extensive experiments have been conducted on various text detection datasets, and state-of-the-art performance has been achieved.
I discuss the information that could be used to uniquely identify a text instance in Sec. 4.2. Sec. 4.3 illustrates the DAS framework. A detailed description of the proposed keypoint-based model for arbitrary-oriented text detection under the framework is in Sec. 4.4. Sec. 4.5 contains the experimental results for the proposed model using several text detection datasets. The conclusion is provided in Sec. 4.6.

Figure 4.1: Scene texts that have been successfully detected by the proposed systems. Images are from ICDAR2015 and TotalText dataset. For TotalText dataset, I visualize the text region with polygons that encapsulate the text region.

4.2 Uniquely Identifying Text Instance

It is widely known that a regular (horizontal) bounding box cannot uniquely identify a text instance, as shown in Fig. 4.2. Such uncertainty happens more often when text is long, oriented, and cluttered. Thus a traditional Mask-RCNN [79] approach usually has trouble segmenting individual text instances.

Many approaches have been designed for predicting quadrilaterals rather than regular horizontal rectangles [1, 30]. These approaches are suitable for multi-oriented text. However, quadrilaterals are also limited in their ability to identify text instances. For example: rotated version of Fig. 4.2.

Recognizing the aforementioned disadvantages of rectangles and quadrilaterals in uniquely identifying arbitrary-oriented text, I introduce several alternatives for
this purpose. Note that I aim not to find the final region of text, but to propose several ideas to uniquely identify one text instance.

**Text Center Line** In Chapter 3, I discussed the concept of a text center line. This concept has been studied in relation to arbitrary-oriented text detection [16]. Unlike a semantic text segmentation mask, which would have trouble separating nearby text instances, a text center line could be trained to uniquely identify a single text instance. Fig. 4.3 gives an example of a visualized text center line.

**Corner Points** Corners of text could also be used to uniquely identify a text instance. Fig. 4.3 shows that four corners of a text could be used to identify the text instance on which the model is focused. Liu et al. [32] proposed locating such corners for multi-oriented text detection.

### 4.3 DAS framework

I briefly introduce the proposed Detect-Associate-Segment (DAS) framework for arbitrary oriented scene text detection in this section.

**Detect** The first step of the framework is to detect instance clues. For example, text center line could be detected with a general encoder-decoder CNN design as
Figure 4.4: The pipeline of the proposed key point based text detection method. It follows the D(etect)-A(ssociate)-S(egment) framework. **Detect:** The detected corner points are visualized with different colors. **Associate:** Regression is used to link the corner points with the assist of embeddings. **Segment:** For each possible hexagonal, I pair it with the instance input to the network for segmentation. The hexagon is drawn only for visualization purpose.

demonstrated in chapter 3 and [16]. Corner points could also be localized with similar fashion. These instance clues are trainable and will not be shared between different text instances. In another word, there is a one-to-one or many-to-one relationship between the instance clues to each text instance. For example, each text center line uniquely matches to one text instance.

**Associate** The second step tries to associate detected instance clues. It aims at grouping the detected instance clues corresponding to the same text instance. For text center line extraction, it needs to group all the pixels belonging to one text center line together. For corner based points detection, it needs to group the corners which belong to the same text instance.

**Segment** After obtaining such grouped instance clues, further instance-aware segmentation from the given region is conducted. The instance clues serve as an extra information for the network to figure out which text instance the model is focusing on and thus it can give an unambiguous instance mask prediction.

From next section, I will discuss one model design based on such framework in details. The model is based on key points detection, association as well as optional instance-aware segmentation. Note that, text center line could also be used as the basis for arbitrary oriented text detection. Some discussion about it is
Figure 4.5: The annotation for the 6 corner points with different colors. I ignore text instance id $k$ here. The links are visualized for pairs of neighbor corner points in $M$.

Figure 4.6: Visualized Ground Truth. From left to right: the original image with the connected corner points visualized, visualized corner points, regression targets for $l_1$ corner points.

in Appendix supplementary material. I will focus on key point based model design in this chapter.

4.4 Key Point Based Text Detection Model

Key points based approaches have gained attention in object detection [80, 81]. They could be classified as bottom-up, single stage methods and are thus efficient. The proposed model is also inspired from the two methods. Fig. 4.4 visualizes the pipeline of the proposed approach.

4.4.1 Key Points

I first introduce which key points to be detected. In corner net [80], the top left and bottom right points are used. However, they are less semantically meaningful as they could lie far from the object region. In extreme net [81], the authors propose
to use 4 extreme points to relieve such problem.

In scene text, however, directly using the corner points would be a more semantically meaningful way. In order to more accurately capture the shape of a text region, I propose to use 6 key points for each text instance. The 6 key points include 4 regular corners (denoted as \( l_1, l_2, r_1, r_2 \)) which are at the beginning and end part of a text as well as 2 center corners (denoted as \( c_1, c_2 \)) which lie at the center of each long side of the text region. Fig 4.5 demonstrates the annotation for the 6 corner points used in this paper. Fig 4.6 visualizes the 6 corner points. By using 6 corner points instead of 4 regular corner points, the model is able to capture a more accurate shape of a text region. It is specifically designed for curved text as it can be used to determine the rough area (polygon) of a text instance and thus the model can do instance-aware segmentation from the cropped region of interest or simply use it as final result.

4.4.2 Key Points Detection

Unlike [32] which uses a proposal based approach to localize corners, I follow [80] to train it as a segmentation problem. Specifically, the network will generates one heat map for each corner point. Sigmoid is used as the last layer to generate the probability for each location. For each corner point location, I follow [80] to generate a positive region with radius determined by the short side (height) of the text bounding box (polygon or quadrilateral). Within the positive region, an unnormalized 2D Gaussian which centered at the ground truth point location is used as the expected score for the corresponding location.

Let \( y^c_p \) denotes the ground truth score at position \( p \) for key point \( c \), \( H^c_p \) represents the predicted score for key point \( c \) at \( p \). The loss for the key points detection task is as follow:

\[
L_{det} = -\frac{1}{N} \sum_{c=1}^{C} \sum_{p=1}^{H \times W} \begin{cases} 
(1 - H^c_p)^\alpha \log(H^c_p), & \text{if } y^c_p = 1 \\
(1 - y^c_p)^\beta (H^c_p)^\alpha \log(1 - H^c_p), & \text{others wise}
\end{cases}
\]

\( H, W \) is the height and width of the output feature map. \( N \) is the number of text instances, and \( C \) is number of key points (equals to 6 in current setting). \( \alpha \) and \( \beta \) are hyper parameters and I follow [80] to set \( \alpha \) to 2 and \( \beta \) to 4.
4.4.3 Key Points Association

Associating key points aims at finding possible text regions. It is the vital component of the whole pipeline. I propose three steps to find and verify groups of text corners: (1) I train regression tasks for each corner points. The regression task tries to link the corner points. For example, the regression targets at \( l_1 \) is the location difference between \( c_1 \) and \( l_1 \), and the regression targets at \( c_1 \) is the difference between \( r_1 \) and \( c_1 \). Details will discussed. (2) Inspired by [80], I train embeddings to “pull” corner points belonging to the same text and “push” corner points belonging to different text instance. The embeddings are used to remove some incorrect regression result. (3) Some heuristic but effective geometry constraints to remove incorrectly associated corner points. Detailed description of the proposed three steps are following.

**Corner Points Regression** Unlike [80, 81] which train the regression task to regress to a more precise location of a corner itself, I take a step further to train to regress to the next corner in a predefined order. Here I define the order to be \( l_1 \rightarrow c_1 \rightarrow r_1 \rightarrow r_2 \rightarrow c_2 \rightarrow l_2 \rightarrow l_1 \), which links the corners clock-wisely.

During inference, let us denote the detected set of corner points from predicted heat map for each \( l_1 \), \( l_2 \) ... \( r_2 \) as \( S_{l_1}, S_{l_2} .. S_{r_2} \). The algorithm starts from each detected corner point \( l^k_1 \) in \( S_{l_1} \) and find the regressed next corner point \( \tilde{c}^k_1 \). It then tries to find the nearest \( c^k_1 \) in \( S_{c_1} \) and use it as next starting point. The algorithm iteratively searches for the corner point until all the 6 corner points are found and thus obtain a text region candidate.

In training, let us denote all the text instances as \( T \) which is annotated as \( T^k = \{l_1^k, l_2^k, c_1^k, c_2^k, r_1^k, r_2^k\} \). Let us define a function \( next \) which gives the next corner point in clockwise order. For example \( next(l_1^k) = c_1^k \). Then the regression target at each corner point \( p \) is then

\[
\alpha_p = [next(p)_x - p_x, next(p)_y - p_y]
\]

Regression loss to be optimized is as:

\[
L_{reg} = \frac{1}{N} \sum_{k=1}^{N} \sum_{p_k \in T^k} SmoothL1(\alpha_p, \delta_p)
\]
Corner Points Embeddings

Associative embedding was originally proposed by Newell et al [82]. Here it is needed to determine whether corner points are belonging to the same text instance or not. Even though all the embeddings for the 6 corner points of one text instance should be close, I train the network to pull embeddings only for the neighbor corner points. Fig. 4.5 gives a demonstration of the corner points and the neighbor connections between them. To be more specific, the neighbor set is defined as: \( M = \{ (l_1, l_2), (l_1, c_1), (l_2, c_2), (c_1, r_1), (c_2, r_2), (r_1, r_2) \} \).

The “pull” and “push” losses are only computed for corner point pairs within the neighbor set \( M \) as following.

\[
L_{\text{pull}} = \frac{1}{N} \sum_{k=1}^{N} \sum_{(p_1, p_2) \in M} \left[ (e_{p_1}^k - e_{p_1, p_2}^k)^2 + (e_{p_2}^k - e_{p_1, p_2}^k)^2 \right]
\]

\[
L_{\text{push}} = \frac{1}{N(N-1)} \sum_{k=1}^{N} \sum_{j=1}^{N} \sum_{(p_1, p_2) \in M} \left[ \max(0, \delta - |e_{p_1, p_2}^k - e_{p_1, p_2}^j|) \right]
\]

\( e_{p_1}^k, e_{p_2}^k \) represents the embeddings for corner point \( p_1, p_2 \) for text instance \( k \), respectively. Pairs \( (p_1, p_2) \) is selected from neighbor set \( M \). \( e_{p_1, p_2}^k \) denotes the average of \( e_{p_1}^k \) and \( e_{p_2}^k \). By training “push” and “pull” loss with the neighbor corner points, the embedding of all the corner points with the same text instance are implicitly forced to be close. I follow [80] and use 1-dimension embeddings.

During inference, the embeddings are used to assist the regression process. In each regression step, after the algorithm finds the next corner point to be linked, a link verification step is conducted based on the embeddings between the two predicted corner points. A threshold \( \mu \) is used to “cut” the unexpected links as [80] and \( \mu \) is set to 1 in experiments. Fig. 4.7 visualizes the use of embeddings to remove incorrect links.

Note that the regression step is already pretty accurate, and using embedding alone will lead to poor performance. The embedding is mostly useful when it tries to regress from one point, for example: \( l_1^k \), to the next point \( c_1 \), but the corresponding \( c_1^k \) is not detected. In this case, if there is no embedding used, the nearest neighbor search might link incorrect corners (probably corners from other text instance).
Figure 4.7: Left: Visualized process of using embedding to remove incorrect regression link (from $r_2$, $c_2$). Red cross visualizes the removed links. Right: Invalid polygon will be removed in geometry constraints step.

Figure 4.8: Left: Visualized fails of embedding links. Embeddings are more useful for nearby text.

Fig. 4.8 visualizes the embedding links predicted. It is generally good when text are close but fails more often when text are far way as there are many corner points belonging to different text instances far away being connected. I believe that this is because of the limited effective receptive field and the fact that a heuristic threshold is needed to cut the link. At the same time, regression fails, although seldom, when nearest neighbor search find nearby wrong corners. Embedding is perfectly suitable in relieving this problem.

**Geometry Constraints** After obtaining groups of corner points (each group consists of 6 corner points), I do further verification based on heuristic geometry property. (1) All groups of corner points which forms invalid polygon defined by connecting corner points clock-wisely will be removed. This happens when these links cross. An example of this invalid polygon is visualized in 4.7. (2) Simple coordinate constraints. For example, x coordinates of $l_1$ should be less than x coordinates $c_1$, etc. (3) Text follows stroke-like shapes, so vector $l_1 \rightarrow l_2$ and $l_2 \rightarrow c_2$ should not be collinear. Same applies to other regular corner points.

The overview of the algorithm used to extract groups of associated corner points
Figure 4.9: Visualized results of corner points association. The predicted points from heat map is visualized with solid dot and regressed points are visualized with circles. Better when zoomed in.

is as Alg. 1.

Fig. 4.9 visualizes some results of corner point association. More results are in Appendix.

### 4.4.4 Unambiguous Instance-Aware Segmentation

Each group of corner points obtained from the previous step indicates one possible text instance. The corner points used here can uniquely identify one text instance, however, they are not the perfect way to approximate text region when text are curved. Fig. 4.11 gives several examples of this. Here I introduce the instance-aware text segmentation module which gives unambiguous text segmentation.

Traditional proposal based approaches such as MaskRCNN [79] crop a feature map using ROI align based on the predicted bounding box and then do segmentation on it. This is not optimal as I described earlier that regular rectangular bounding box can not uniquely identify one text instance. However, since the cor-
Algorithm 1: Corner Association

Input: Corner Heatmap, Regression Map, Embedding Map \( P_C, P_R, P_E \) for \( c \in \{l_1, l_2...r_2\} \). All inputs are with shape \( H \times W \).

Output: Groups of Associated Corner Points \( G_k = C_{k1}^l, C_{k2}^l, C_{k6}^r \)

// Extract corner points in heatmap \( P_C \).
// \( C_c \) are sets of points
\( C_c \leftarrow \) ExtractPeak\( (P_C) \) \( c \in \{l_1, l_2...r_2\} \)

// Embedding relation for each pair of corner points in \( M \)
\( L(c1,c2) \leftarrow \) ExtractLink\( (C_{c1}, C_{c2}, P_{c1}^E, P_{c2}^E) \) \( (c1,c2) \in M \)

// Iteratively search corners, first initialize
// next function gives next set of points to check \( l_1 \rightarrow c_1 \)
\( preIdx = l_1 \)
\( G = [] \)
for \( p^{preIdx} \in C^{preIdx} \) do
    \( nextIdx = next(preIdx) \)
    // find estimated point based on regression
    for \( nextIdx \neq l_1 \) do
        \( p^{nextRegressed} \leftarrow \) Regression\( (p^{preIdx}, P_R) \)
        // find nearest corner points with regressed point
        \( q \leftarrow \) findNearest\( (p^{nextRegressed}, C_{nextIdx}) \)
        // verify based on Embedding Link
        if IsCorrect\( (L^{preIdx, nextIdx}, p^{preIdx}, q) \) then
            \( preIdx \leftarrow nextIdx \)
        else
            break
    if \( nextIdx == l_1 \) then
        Update\( (G) \)
    \( G = GeometryFilter(G) \)

Corner points are already detected from the previous steps, they could be perfectly served as additional features and indicate which text instance the model is focusing at. This idea is the same from previous chapter which used text center line as additional feature for text segmentation. I use the 6 corners to indicate which text instance to segment. Specifically, for each detected group \( k \) of corner points \( l_1^k, c_1^k, c_2^k, l_2^k, r_1^k, r_2^k \), the model estimate a rectangle bounding box based on the corner location, and crop \( I_v^k \) from the original image. Another instance clue map \( I_c^k \) is then created with the same height and width with \( I_v^k \). In the instance clue map, 6 circles in the corner locations are drawn with a fixed radius equal to 2. The final input for segmentation is then the concatenation of \( I_v^k \) and \( I_c^k \) on the depth dimension and thus it will be an tensor with shape \([H, W, 4]\). It is unambiguous as the extra instance clue map indicate which text to segment even when there are
Fig. 4.10 shows some instance-aware segmentation examples. Even when there are multiple texts in the image, given the instance clue map, the network still able to figure out which instance to segment.

### 4.4.5 Overview of CNN architecture

I follow [80, 81] and use hourglass network [83] as the basis CNN architecture for generating the result feature maps for key point detection, regression and embeddings. Fig. 4.12 visualizes the CNN design. The generated results maps are as follows.

**Key Point Response Heat Map**: 6 heat maps for the corner points.

**Key Point Regression Map**: 6 regression maps for the 6 corner points. It aims at correcting location mistakes produced by the CNN encoder decoder pipeline [80].

**Key Point Embedding Map**: 6 key points embedding maps. The embed-
Figure 4.12: The network generates 3 kinds of different feature maps: key point heat maps, key point regression maps, key point embedding maps.

...dings are used in the association stage.

Note that I am not using the corner pooling approach proposed by [80] because in my setting, all the key points are semantically meaningful and I simply need to localize them in the given input image. The loss is then \( L = L_{\text{det}} + \gamma \times L_{\text{ref}} + \rho \times (L_{\text{push}} + L_{\text{pull}}) \). \( \gamma \) and \( \rho \) are simply set to 1 in experiment.

For segmentation branch, I use a VGG16 based encoder decoder network. It is trained separately with the key point detection branch with pixel wise cross entropy loss.

4.5 Experiments

4.5.1 Implementation

The pipeline is implemented in the Pytorch framework. It is based on open sourced version\(^1\) of [81]. Data augmentation is used during training with random cropping, scaling. All the input during training is resized and padded to have the same size 768 \( \times \) 768 The CNN part of the model is initialized from pretrained CornerNet model [80].

In order to demonstrate the effectiveness of the proposed framework, I did experiments on various datasets including multi-oriented text and arbitrary oriented text. For datasets with multi-oriented text (ICDAR2013, ICDAR2015, MSRA), I will not run the segmentation branch but simply run the corner points association.

\(^1\)https://github.com/xingyizhou/ExtremeNet
Corner points already give an accurate prediction for multi-oriented text. In addition to this, it is easy to adapt the proposed model to 4 corner points based model if quadrilateral prediction is needed. However, I keep the same configuration in this experiment for consistency.

4.5.2 Quantitative Results

**ICDAR2015 and ICDAR2013** It contains 233 testing images denoted as \(IC_{13\text{test}}\) and 229 training images denoted as \(IC_{13\text{train}}\). The bounding boxes are annotated with top left corner location and bottom right corner location as regular bounding boxes. Let’s denote the annotation as \(tl_x, tl_y, br_x, br_y\), I calculate the center corner points \(c_1\) and \(c_2\) as: \(c_1 = (tl_x/2 + br_x/2, tl_y)\) and \(c_2 = (tl_x/2 + br_x/2, t2_y)\) and then transform the annotation into 6 corner points annotation.

ICDAR 2015 Incidental Scene Text Detection Dataset. This contains 1000 training images and 500 testing images denoted as \(IC_{15\text{train}}\) and \(IC_{15\text{test}}\). Images are taken with portable devices with motion blur. The text are multi-oriented with quadrilateral annotation. Let’s denote each quadrilateral bonding box as \(p_1, p_2, p_3, p_4\), they are annotated in an clock-wise manner. I can directly use them as the 4 corners \(l_1, r_1, r_2, l_2\) in corner annotation. \(c_1\) is calculated as the middle point of \(l_1, r_1\) and \(c_2\) as the middle point of \(l_2\) and \(r_2\).

For evaluation on ICDAR 2013, I train on \(IC_{13\text{train}} + IC_{15\text{train}}\) and test on \(IC_{13\text{test}}\). The result is in Table 5.2.

For evaluation on ICDAR 2015, the trained model is the same one as tested for ICDAR2013 with results in Table 5.3. I also did ablation study here to remove the embeddings in the association step. It can be seen that by adding the embedding removal step, accuracy and F-1 score has been improved while recall has dropped slightly. This is because of some incorrect embedding predictions which unexpectedly “cut” good connections.

**MSRA-TD500** I follow [32] and use HUST-TR400 which contains 400 images as additional training data. The dataset contains text line based annotation and are thus focusing on model’s ability in detecting long text. I follow a similar preprocessing manner as for ICDAR2015 dataset to create corner annotations. In test stage, the inputs are resized and padded to \(768 \times 768\). The result is in 4.3.
## ICDAR2013

<table>
<thead>
<tr>
<th>Method</th>
<th>R</th>
<th>P</th>
<th>F-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhang et al. [14]</td>
<td>89</td>
<td>78</td>
<td>83</td>
</tr>
<tr>
<td>StradVision [84]</td>
<td>79</td>
<td>90</td>
<td>84</td>
</tr>
<tr>
<td>Multi-scale [85]</td>
<td>79</td>
<td>83</td>
<td>85</td>
</tr>
<tr>
<td>Yao et al. [86]</td>
<td>80</td>
<td>89</td>
<td>84</td>
</tr>
<tr>
<td>DirectReg [30]</td>
<td>81</td>
<td>92</td>
<td>86</td>
</tr>
<tr>
<td>Lyu et al. [32]</td>
<td>79.4</td>
<td><strong>93.3</strong></td>
<td>85.8</td>
</tr>
<tr>
<td>proposed</td>
<td><strong>82.4</strong></td>
<td><strong>91.2</strong></td>
<td><strong>86.6</strong></td>
</tr>
</tbody>
</table>

Table 4.1: Localization Performance (%) on ICDAR2013 dataset. R: Recall. P: Precision.

## ICDAR2015

<table>
<thead>
<tr>
<th>Method</th>
<th>R</th>
<th>P</th>
<th>F-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhang et al. [14]</td>
<td>43</td>
<td>71</td>
<td>54</td>
</tr>
<tr>
<td>StradVision [84]</td>
<td>37</td>
<td>77</td>
<td>50</td>
</tr>
<tr>
<td>Multi-scale [85]</td>
<td>54</td>
<td>76</td>
<td>63</td>
</tr>
<tr>
<td>Yao et al. [86]</td>
<td>59</td>
<td>72</td>
<td>65</td>
</tr>
<tr>
<td>DirectReg [30]</td>
<td>82</td>
<td>80</td>
<td>81</td>
</tr>
<tr>
<td>SegLink [52]</td>
<td>77</td>
<td>73</td>
<td>75</td>
</tr>
<tr>
<td>EAST [1]</td>
<td>77.90</td>
<td>82.38</td>
<td>80.07</td>
</tr>
<tr>
<td>TextSnake [16]</td>
<td><strong>84.9</strong></td>
<td>80.4</td>
<td>82.6</td>
</tr>
<tr>
<td>Lyu et al. [32]</td>
<td>70.7</td>
<td><strong>94.1</strong></td>
<td>80.7</td>
</tr>
<tr>
<td>Proposed (no embeddings)</td>
<td>76.8</td>
<td>90.2</td>
<td>82.96</td>
</tr>
<tr>
<td>Proposed</td>
<td>76.2</td>
<td>92.5</td>
<td><strong>83.56</strong></td>
</tr>
</tbody>
</table>

Table 4.2: Localization Performance (%) on ICDAR2015 dataset. R: Recall. P: Precision.

It achieves comparable performance with state-of-the-art method proposed in [32]. However, their method can only deal with multi-oriented text while the proposed model can tackle arbitrary oriented text.

**TotalText** TotalText [49] is a newly released dataset containing polygon annotations with 1255 training images and 300 testing images. It focuses on arbitrary oriented text detection and contains many curved text instances. For pre-processing, because the dataset contains a small number of text polygon annotations with odd number of points, I remove them by masking out the text area of those polygons. For other polygons with even number of points, I first find the four regular corners and then find the center corners in order to generate the 6 points
Table 4.3: Localization Performance (%) on MSRA-TD500 dataset. R: Recall. P: Precision.

<table>
<thead>
<tr>
<th>Method</th>
<th>R</th>
<th>P</th>
<th>F-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD-ICDAR [18]</td>
<td>52</td>
<td>53</td>
<td>50</td>
</tr>
<tr>
<td>Zhang et al [14]</td>
<td>67</td>
<td>83</td>
<td>74</td>
</tr>
<tr>
<td>Yao et al [86]</td>
<td>75.3</td>
<td>76.5</td>
<td>75.9</td>
</tr>
<tr>
<td>EAST [1]</td>
<td>67.4</td>
<td>87.3</td>
<td>76.1</td>
</tr>
<tr>
<td>Lyu et al [32]</td>
<td>76.2</td>
<td>87.6</td>
<td>81.5</td>
</tr>
<tr>
<td>Proposed</td>
<td>74.3</td>
<td><strong>89.5</strong></td>
<td>81.2</td>
</tr>
</tbody>
</table>

Table 4.4: Localization Performance(%) on TotalText dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>R</th>
<th>P</th>
<th>F-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SegLink [52]</td>
<td>30.3</td>
<td>23.8</td>
<td>26.7</td>
</tr>
<tr>
<td>EAST [1]</td>
<td>50.0</td>
<td>36.2</td>
<td>42.0</td>
</tr>
<tr>
<td>TextSnake [16]</td>
<td><strong>74.5</strong></td>
<td>82.7</td>
<td>78.4</td>
</tr>
<tr>
<td>Proposed (corner)</td>
<td>72.1</td>
<td>87.4</td>
<td>79.01</td>
</tr>
<tr>
<td>Proposed</td>
<td>73.3</td>
<td><strong>88.2</strong></td>
<td><strong>80.1</strong></td>
</tr>
</tbody>
</table>

Table 4.4 shows the results of the proposed model on TotalText Dataset with comparison with other methods. I compare two versions of the proposed methods: (1) with only corner point predictions, (2) with segmentation. For most text, hexagon can already accurately capture the text region and thus with only corner point prediction, state-of-the-art performance is already achieved. For the second model with segmentation branch added, I can tackle more curved text. Note that, I do not run segmentation on all the predicted text instance. Only predicted hexagon with clear curve shape (angle between $l_1 > c_1$ and $c_1 > r_1$ is larger than 50 degree) will need to run the segmentation branch. This also saves the inference time and is one of the advantage of predicting 6 corner instead of 4.

4.5.3 Limitation

The proposed model achieves state-of-the-art performance on different types of text. But it still fails on certain difficult cases. Fig. 4.13 shows several failure
Figure 4.13: Some failure examples from the corner points association step. Red box: false positive. Green Box: true positive. Yellow Box: missed detection.

examples. Note that I simply visualize the failed corner points association result.

One limitation of the proposed model is that, its recall is not high compare with other state-of-the-art methods. Another limitation is that, if it is needed to tackle extremely curved text, both key point detection and segmentation needs to be run separately and are thus not so efficient. However, if only multi-oriented text is considered or hexagon is good enough to represent bounding box for text, then the model is both accurate and efficient. In fact, in the experiment hexagon already gives pretty good result when I am not dealing with extremely curved text. In addition to this, it is also easy to adapt the proposed model to predict different kinds of polygons with fixed number of corners (e.g., octagon).

4.6 Conclusion

In this chapter, I proposed a novel and general framework (DAS) for detecting arbitrary-oriented text. This framework can predict the text instance-level mask for each text instance. The framework contains three steps: detect, associate, and segment. I designed a novel keypoint-based model under the proposed framework, which achieves state-of-the-art performance using various datasets. This demonstrates its efficiency and robustness in tackling both multi-oriented and curved text.
TextContourNet: a Flexible and Effective Framework for Improving Scene Text Detection Architecture with a Multi-task Cascade

5.1 Introduction

This chapter mainly investigates designing training frameworks for scene text detection by adding auxiliary tasks.

As previously mentioned, contemporary scene text detectors usually fall into one of three categories: (1) segmentation-based methods, (2) regression-based methods, and (3) proposal-based methods. Regression and proposal methods usually perform better in multi-oriented text detection. However, both of these types of methods have their drawbacks. Proposal-based methods are usually less accurate in terms of recovering multi-oriented bounding boxes [30]. Regression-based methods, though able to predict accurate oriented bounding boxes, exhibit difficulties when text has large variances of scales since each output bounding box is generated from a single output pixel. Usually, then, a multi-scale testing is needed. He [25] proposed jointly learning a text attention map that suppresses background interference for a better word-level text proposal. The text attention
Figure 5.1: Scene texts that have been successfully detected with the model trained with proposed framework. Images are from ICDAR2013, ICDAR2015, UberText dataset. Better when zoom in.

map is essentially a text-nontext segmentation and could be used as a mask to remove background noise. This design could be seen as using a semantic segmentation to assist with text proposal generation. Such a combination also achieved better performance.

Learned contour detection [87, 88, 89] has become popular in other research fields. Unlike regular edge detection, learned contour detection provides more instance-level semantic information.

Inspired by these two research fields, in this chapter, I introduce a training framework that similarly uses an easier task to assist with the scene text detection task. However, instead of using a semantic text/nontext segmentation mask to suppress background interference, I propose learning a contour map that directly encodes the text instance information to better facilitate the text detection task.

In summary, the contributions of this chapter are:

(1) Instance-level contour segmentation for scene text is introduced. In general contour detection, a contour can be seen as a subset of edges that preserves instance-level semantic meanings. In scene text, I propose that, instead of accurate edges for each letter, the contour of a word is the polygon that best encapsulates
that word.

(2) I experimentally demonstrate that such an instance-level contour can be easily learned with a traditional encoder-decoder CNN design. Unlike contours defined in natural images that require substantially more labor for annotation, a scene text contour needs only the original bounding box annotation.

(3) I propose using the learned contour segmentation to assist with scene text detection. The contour map provides extra instance-level semantic information that better assists in the detection task than the instance-agnostic semantic information provided by the text/nontext attention map [25]. It is also easier to learn than the regular text detection task, which means that it follows the general design principle of using an easier task to assist with a harder task.

(4) I propose two general designs for incorporating the contour task: contour as auxiliary loss and contour as a multi-task cascade. Extensive experiments on public datasets with different model configurations under the two frameworks have been conducted and show that both framework designs improve the performance of a state-of-the-art scene text detector. Such improvements cannot be achieved by simply choosing a deeper network backbone.

In the following sections, I begin by discussing related works in contour detection. In Sec. Text Contour, I discuss contour definition as well as the basic CNN encoder-decoder network for contour detection. In Sec. Contour For Text Detection, I introduce the framework for incorporating learned contours in scene text detection. I show the effectiveness of the proposed framework in Sec. Experiments. Fig. 5.1 shows the detection results from a model trained using the proposed framework.

## 5.2 Contour Detection

Contour detection was originally designed for extracting edges in natural images which preserves semantic meanings. Usually additional effort is needed in labeling [88, 89] the images in order to train a learning-based model. For example, Yang [88] proposed to use a dense CRF to refine the annotation so as to obtain trainable contour annotations. They also used the learned contour for proposal generation. In this chapter, I take a step further and use the learned contour di-
rectly as a feature to facilitate scene text detection. It follows general multi-task learning design.

The text contour method defined in this work is unique in that: (1) the contour is not a subset of low-level image edges but the border of each text instance and (2) no extra annotation is needed which makes the current framework much easier to implement.

5.3 Text Contour

5.3.1 Definition

As opposed to the definition of contour in most of the previous literature, He [90] proposed to learn the contour of text, tables, and figures in PDF documents. The contour does not align with any low-level image edges. Instead, the network has to learn more semantic information in order to decide where is the contour. This makes it an unique learning task.

Here I adopt a similar idea and define the contour for scene text to be the polygon that approximates the border of each instance. The polygon could be a quadrilateral but also an octagon depending on the annotation of the dataset. Fig. 5.2 provides several visualized examples of word contours. Depending on the dataset annotations, different contour definitions could be used.

Figure 5.2: Contour examples which are generated by polygon annotations. Most datasets provide quadrilateral annotations (e.g: ICDAR, UberText). The blue color is only for visualization.

It is easy to notice that if quadrilateral is chosen, the contour could be rendered easily with the annotation provided by most datasets (e.g: ICDAR2015). In this work I use quadrilateral setting and demonstrate that the trained model could be helpful for text detection task.
5.3.2 Learning to Extract Contour

In order to learn to segment the contour of text, I design an encoder-decoder CNN with a skip link for multi-scale feature learning. This will be the basic architecture for contour network. Sigmoid is applied to the last convolutional layer and I use a mean squared error (MSE) for training. The loss value for training is denoted as $L_{\text{contour}}$.

**Ground Truth Generation** In order to train the network to detect scene text contour, ground truths need to be provided. Quadrilateral bounding box is used to generate contour ground truths. Even though this might not capture the most precise boundary of scene text, a model trained with it can still give good prediction results.

In consideration of the imperfectness of the ground truth quadrilateral bounding boxes (as they are annotated for detection task), I follow the work of He [90] for creating a smoothed border for training. Formally, I follow the equation 5.1 for generating the contour. $S_{\text{contour}}$ represents the pixels in the contour generated based by the ground truth annotation. Values of the regression targets are empirically selected. This way a contour prediction that is a few pixels away from the ground truth will not be penalized that much.

$$x_i = \begin{cases} 
1 & \text{if } i \in S_{\text{contour}} \\
0.9 & \text{if dist}(i, j) == 1 \text{ and } \exists j \in S_{\text{contour}} \\
0.6 & \text{if dist}(i, j) <= 3 \text{ and } \exists j \in S_{\text{contour}} 
\end{cases}$$

Fig 5.3 shows an example input image and the corresponding contour ground truth.

**Learnability** Even though the contour segmentation task needs to learn instance-level information, I claim that the task is easier to learn when compared with learning a regular detection task. The major reason is that the network only needs to learn which pixels separate text from non-text background or separate two text instances (if they are close). It implicitly learns instance information without the need for global context. This is because even when looking at a relatively local region, it is still possible to identify whether it is the boundary or not. This is opposed to the regression task in scene text detection, for which one needs to iden-
Figure 5.3: The generated ground truth for training the contour network. Better when zoomed in.

Figure 5.4 illustrates this idea. Suppose that the blue point and the red point represent the output pixels for a text detection network and a contour detection network, respectively. The blue and red circle represents their receptive fields. \( L \) represents one of the regression targets (The distance to a border of text). The text detector has to identity where is the border of the text instance and how far the current pixel is from the boundary. This is much harder when centered at the boundary and needs to predict the distance to the other side boundary of the text bounding box. However, predicting whether the pixel belongs to a border (contour) is much easier. On the other hand, if rough contour information is provided to the network, the regression task can be much easier to learn. This is the main idea behind using contour to assist scene text detection.

## 5.4 Contour For Text Detection

Since learning the text contour from input image is capable, I propose to use it as an additional task for scene text detection. I describe two proposed frameworks for using text contours for scene text detection as well as the scene text detector that I adopt in this work. Basically, the two general frameworks are: (1) Using the contour segmentation as a sub-task and jointly train the network. This is called Auxiliary TextContourNet. (2) Using the learned contour as features for scene text detection in a cascade fashion. This is called Cascade TextContourNet.
5.4.1 Auxiliary TextContourNet

The framework for Auxiliary TextContourNet is shown in Fig. 5.5. It follows basic encoder-decoder design with several layers of shared convolutions. Note that in the illustration of the framework, only the encoder convolution layers are shared due to space limitations. In experiments, I show results of two designs with different convolutional layers shared.

5.4.2 Cascade TextContourNet

The contour could also be used in a cascade fashion. As opposed to previous work [25] which applies the semantic text-nontext segmentation map as a mask, this framework treats the explicitly learned contour as another feature layer and
lets the network jointly learn from both the visual features extracted from original image as well as the contour prediction map.

I believe this framework has several advantages: (1) Applying the semantic text-nontext map as mask makes it hard for the network to recover errors from the segmentation task. Instead, here the model jointly learns the task from contours as well as the extracted visual features. Errors made from contour network doesn’t necessarily lead to false detection results. (2) The text-nontext prediction is a regular task for many scene text detector [1, 30, 24]. Adopting it as a subtask will have no benefit for these detector. (3) The text-nontext segmentation only provides semantic information while contour segmentation provides instance-level semantic information. Being able to provide instance-level information in the early stage of the network allows the network to learn to propose an instance bounding box more easily and accurately.

This also follows the general design intuition of using an easier task to assist a harder task. Previous works [91, 92] also use such ideas for training deep models. Such a cascade fashion has also been adopted by other works [93, 94].

In detection task, both settings could be adopted and I experimentally show that both of them improve the performance of the original scene text detector. By explicitly learning to segment the contour and using it as features for detection (Cascade TextContourNet), the detector could perform better than using the contour detection as an auxiliary loss (Auxiliary TextContourNet).

In order to use the contour segmentation output as features, I propose two schemes: (1) Early Merge: merge the extracted contour information in the early encoder stage. (2) Late Merge: Merge the extracted contour information in later decoder stage. Fig. 5.6 shows the visualization of the proposed frameworks. Note that, for early merge, since the encoder features needs to be recomputed and the input to the detection network is a 4-channel tensor, the convolutional features are not shared between the detection task and the contour segmentation task.

5.4.3 Scene Text Detector

I adopt the work [1] as the base detector design. It is one of the representative, state-of-the regression-based scene text detector models. Here I give a brief de-
Figure 5.6: Two frameworks for using in a cascade manner the contour information for scene text detection. **up**: Join the contour in the early stage. In this case, there is no shared convolutional parameters. **down**: Join the contour in the later stage and share the features and parameters in the encoder.

scription of the method.

Given an image $I$ with height $I_h$ and width $I_w$, an encoder-decoder network is used to predict $M$ channels of output with height and width $S_h$, $S_w$, respectively. The value of $M$ depends on the different geometric shapes the model is trained to predict. Two geometries were proposed in the original paper [1] and here I use $RBOX$ setting.

In $RBOX$, a rotated bounding box is predicted for each output pixel location. There are 6 output maps in total and the first channel is the score map with each pixel valued from $[0,1]$. It corresponds to the confidence for each location to be text or not. Note that this is also a major difference between a regression-based method and a proposal-based method. For a proposal-based method, the class of each output location depends on the intersection over the union (IOU) score of the default anchor and the ground truth bounding boxes. Here the class simply depends on whether the pixel belongs to the region of text or not. It should be noted that the ground truth bounding boxes are shrunk before creating the score map targets to better separate out nearby words. Dice loss is used for training
the score map, which directly optimizes the IOU of the segmentation results. This loss is denoted as $L_{\text{score}}$ in 5.2. $y_s^*$ denotes the ground truth score map and $\hat{y}_s$ is the predicted score map. $\beta$ is the weight for positive class and negative class.

$$L_{\text{score}} = 1 - \frac{2 \times \hat{y}_s \ast y^*}{(\sum \hat{y}_s + \sum y^*)} \quad (5.2)$$

The other output channels correspond to the geometric information of the predicted bounding boxes. For the RBOX scheme for each positive class location, its distances to the 4 boundaries of the rotated bounding box are used as ground truth, and IOU loss [95] is used for loss calculation because it’s invariant to object scale such that different scales of text will have the same contribution. The loss is denoted as $L_{\text{IOU}}$. The orientation angle of the word $\hat{\theta}$ is also used as another target and the loss is denoted as $L_\theta(\hat{\theta}, \theta^*)$, where $\theta^*$ is the ground truth angle.

As such, 5 channels for geometry will be predicted with the total geometry loss denoted as $L_{\text{geo}}$. The total training loss $L_{\text{det}}$ is the weighted sum of $L_{\text{score}}$ and $L_{\text{geo}}$ in Eq. 5.4. More details can be found in the original paper [1].

$$\begin{array}{l}
L_\theta(\hat{\theta}, \theta^*) = 1 - \cos(\hat{\theta} - \theta^*) \\
L_{\text{geo}} = \lambda_{\text{IOU}} L_{\text{IOU}} + L_\theta(\hat{\theta}, \theta^*)
\end{array} \quad (5.3)$$

$$L_{\text{det}} = \lambda_{\text{geo}} + \lambda_{\text{cls}} L_{\text{score}} \quad (5.4)$$

### 5.4.4 Joint Training

By incorporating the contour detection with scene text detection, the loss for both frameworks is defined in 5.5. $\beta$ is set to 0.1 and $\lambda_{\text{cls}}$ is set to 0.01 for experiments.

$$L = L_{\text{geo}} + \lambda_{\text{cls}} L_{\text{score}} + \beta L_{\text{contour}} \quad (5.5)$$
5.5 Experiments

5.5.1 Implementation

The pipeline is implemented in the Tensorflow [96] framework. The baseline method is based on the East implementation \(^1\) which is a modified version of the original model [1] with slightly better performance. Data augmentation during training was similar to SSD [29] with random cropping and scaling. The training process contains two steps: (1) train all the augmented images with fixed input size $512 \times 512$. (2) fine tune the trained model with fixed input size $768 \times 768$. I use Adam optimizer for training.

All models have a CNN backbone of Resnet50 [97] with a feature pyramid design [98] unless specified otherwise. The CNN is initialized with pretrained image classification model. The output resolution is $1/4$ of the input image resolution. The CNN backbone part is initialized with a pretrained model. For the Auxiliary TextContourNet framework, I implement two variants of it: (1) Only the CNN encoder (Resnet50) parts are shared ($AuxiliaryContourNet_1$). (2) The CNN encoder and decoder parts are all shared ($AuxiliaryContourNet_2$). The only difference between $AuxiliaryContourNet_2$ with the baseline model is that an additional output channel is produced and trained as the contour segmentation. For Cascade TextContourNet, I designed two models corresponding to the two schemes proposed: (1) Early Merge: I resize and concatenate the output contour to feed as one input channel ($CascadeContourNet_1$). The input to the detection network is thus a 4-channel tensor. (2) Late Merge: I concatenate the output contour with the last layer in the detector branch in depth dimension. Then three convolutional layers are added with depth 32, kernel size $3 \times 3$ before producing the final detection output ($CascadeContourNet_2$).

5.5.2 Quantitative Experiments

Let’s denotes the training sets for ICDAR 2013, ICDAR 2015 and Uber Text as $IC13_{train}$, $IC15_{train}$ and $Uber_{train}$. The corresponding test sets are $IC13_{test}$, $IC15_{test}$ and $Uber_{test}$.

\(^1\)https://github.com/argman/EAST
5.5.2.1 Capacity Study

In this study, I demonstrate the capacity of the proposed framework by training on $IC_{13train} + IC_{13test} + IC_{15train} + IC_{15test}$ and test on $IC_{15test}$. This experiment aims at showing that with quadrilateral annotation, the contour task can substantially improve the quality of the feature representation learned and the model’s ability in fitting the training data. With potentially much more data that are commonly used in industry, such study shows that by adopting the proposed framework, the model can learn a much better feature representation from the training data and to achieve a much better performance. The results are in Table 5.1.

It can be seen that adding contour as a auxiliary task can improve the f-measure by approximately 1 percent while adding it as a cascade task can improve it by 3 percent with the late merge mechanism ($CascadeContourNet_2$). I also observed that for early merge cascaded model ($CascadeContourNet_1$) the performance dropped instead. I believe that this is because the learned contour map is highly semantic and the network could not learn good features when I concatenate it with the raw input images. In later experiments, I only use late merge for comparison.

Baseline 101 represents the model trained with Resnet101 backbone. It is also initialized from a pretrained model. It could be observed that even with a much deeper network, the capacity of the network almost stayed the same.

All the following studies correspond to regular experimental settings. They could be seen as evaluating the generalizability of the proposed framework.

5.5.2.2 ICDAR2013

I train the model with different proposed framework configurations on $IC_{13train} + IC_{15train}$ and test on $IC_{13test}$ with results in Table 5.2.

5.5.2.3 ICDAR2015

I train the model with different proposed framework configurations on $IC_{13train} + IC_{15train}$ and test on $IC_{15test}$. The trained model is the same one as tested for ICDAR2013 with results in Table 5.3. By adopting the proposed framework, the
<table>
<thead>
<tr>
<th>Method</th>
<th>R</th>
<th>P</th>
<th>F-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>86.2</td>
<td>90.40</td>
<td>88.25</td>
</tr>
<tr>
<td>Baseline 101</td>
<td>86.72</td>
<td>90.95</td>
<td>88.78</td>
</tr>
<tr>
<td>AuxiliaryContourNet_1</td>
<td>86.42</td>
<td>92.43</td>
<td>89.32</td>
</tr>
<tr>
<td>AuxiliaryContourNet_2</td>
<td>86.61</td>
<td>92.88</td>
<td>89.635</td>
</tr>
<tr>
<td>CascadeContourNet_1</td>
<td>85.22</td>
<td>84.33</td>
<td>84.77</td>
</tr>
<tr>
<td>CascadeContourNet_2</td>
<td>91.42</td>
<td>93.44</td>
<td>92.58</td>
</tr>
</tbody>
</table>

Table 5.1: Capacity study (Ability in fitting training data): Localization performance (%) on ICDAR2015 dataset. I use this study to show how well the model can fit the training data.

<table>
<thead>
<tr>
<th>Method</th>
<th>R</th>
<th>P</th>
<th>F-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhang et al. [14]</td>
<td>89</td>
<td>78</td>
<td>83</td>
</tr>
<tr>
<td>StradVision [84]</td>
<td>79</td>
<td>90</td>
<td>84</td>
</tr>
<tr>
<td>Multi-scale [85]</td>
<td>79</td>
<td>83</td>
<td>85</td>
</tr>
<tr>
<td>Yao et al. [86]</td>
<td>80</td>
<td>89</td>
<td>84</td>
</tr>
<tr>
<td>DirectReg [30]</td>
<td>92</td>
<td>81</td>
<td>86</td>
</tr>
<tr>
<td>Baseline MS</td>
<td>83.61</td>
<td>91.21</td>
<td>87.24</td>
</tr>
<tr>
<td>AuxiliaryContourNet_1 MS</td>
<td>84.20</td>
<td>92.48</td>
<td>88.15</td>
</tr>
<tr>
<td>AuxiliaryContourNet_2 MS</td>
<td>84.04</td>
<td>93.40</td>
<td>88.47</td>
</tr>
<tr>
<td>CascadeContourNet_2 MS</td>
<td>85.22</td>
<td>93.02</td>
<td>88.95</td>
</tr>
</tbody>
</table>

Table 5.2: Localization performance (%) on ICDAR2013 dataset. R: Recall. P: Precision. MS represents multi-scale testing. I compare the proposed model with the baseline as well as other methods.

The model’s performance improves slightly and the cascade setting achieves the best performance.

### 5.5.2.4 UberText

I train on Uber_{train} and test on Uber_{test}. Uber text also contains quadrilateral annotation. Here I simply fit a minimum oriented rectangle for training and testing. For other baseline approaches, I implemented the faster RCNN [28] based on Google Object Detection api\(^2\) and modified the regression scheme as in Equation 5.6.

---

\(^2\)github.com/tensorflow/models/tree/master/research/object_detection
<table>
<thead>
<tr>
<th>Method</th>
<th>R</th>
<th>P</th>
<th>F-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhang et al. [14]</td>
<td>43</td>
<td>71</td>
<td>54</td>
</tr>
<tr>
<td>StradVision [84]</td>
<td>37</td>
<td>77</td>
<td>50</td>
</tr>
<tr>
<td>Multi-scale [85]</td>
<td>54</td>
<td>76</td>
<td>63</td>
</tr>
<tr>
<td>Yao et al. [86]</td>
<td>59</td>
<td>72</td>
<td>65</td>
</tr>
<tr>
<td>DirectReg [30]</td>
<td>82</td>
<td>80</td>
<td>81</td>
</tr>
<tr>
<td>SegLink [52]</td>
<td>77</td>
<td>73</td>
<td>75</td>
</tr>
<tr>
<td>Baseline</td>
<td>77.90</td>
<td>82.38</td>
<td>80.07</td>
</tr>
<tr>
<td>AuxiliaryContourNet₁</td>
<td>77.35</td>
<td>83.88</td>
<td>80.48</td>
</tr>
<tr>
<td>AuxiliaryContourNet₂</td>
<td>78.58</td>
<td>85.02</td>
<td>81.64</td>
</tr>
<tr>
<td>CascadeContourNet₂</td>
<td>79.91</td>
<td>86.12</td>
<td>82.90</td>
</tr>
</tbody>
</table>

Table 5.3: Localization performance (%) on ICDAR2015 dataset. R: Recall. P: Precision.

\[ t^*_x = \frac{(g_{xj} - x_a)}{w_a} \]
\[ t^*_y = \frac{(g_{yj} - y_a)}{h_a} \]
\[ j = 1, 2, 3, 4 \] (5.6)

\( t^*_x \) and \( t^*_y \) is the encoded \( x, y \) coordinates, respectively. They follow the same design pattern as in the original faster rcnn for object detection when I encode the center. Here for each oriented bounding box, 4 coordinates needs to be encoded. Note that here I ignore the proposal index \( i \) that is used in the following equation.

By using such regression scheme, the overall loss function for training this baseline faster rcnn scene text detector is as (5.7)

\[ L_{\text{baseline}} = \frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(p_i, p^*_i) + \lambda \ast \frac{1}{N_{\text{reg}}} \sum_i p_i \sum_{j=1}^{4} L_{\text{reg}}(t_{ij}, t^*_{ij}) \] (5.7)
Table 5.4: Localization performance (%) on UberText dataset. TextFasterRCNN is a proposed baseline method based on tensorflow object detection API.

<table>
<thead>
<tr>
<th>Method</th>
<th>Recall</th>
<th>Precision</th>
<th>F-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextFasterRCNN resnet101</td>
<td>55.80</td>
<td>53.04</td>
<td>54.38</td>
</tr>
<tr>
<td>TextFasterRCNN resnet50</td>
<td>49.74</td>
<td>47.29</td>
<td>48.48</td>
</tr>
<tr>
<td>Baseline</td>
<td>70.33</td>
<td>83.82</td>
<td>76.48</td>
</tr>
<tr>
<td>AuxiliaryContourNet$_1$</td>
<td>73.57</td>
<td>82.92</td>
<td>77.96</td>
</tr>
<tr>
<td>AuxiliaryContourNet$_2$</td>
<td>73.33</td>
<td>82.53</td>
<td>77.64</td>
</tr>
<tr>
<td>CascadeContourNet$_2$</td>
<td><strong>74.02</strong></td>
<td><strong>84.26</strong></td>
<td><strong>78.81</strong></td>
</tr>
</tbody>
</table>

I trained two faster-rcnn model with the backbones resnet50 and resnet101 with results in 5.4.

5.5.3 Consistency

Scene text detection is usually evaluated and compared based on a quadrilateral IOU threshold 0.5. However, such an IOU threshold may not reflect the model’s real performance when combining it with a scene text reader. This is because usually a much higher IOU is needed in order to read the text correctly. This idea is also mentioned in [24]. Here I show that the model trained with the proposed framework could consistently outperform a baseline method with a different IOU threshold. The results are in Fig. 5.7. I only show the results for AuxiliaryContourNet$_1$ and CascadeContourNet$_2$ for visualization purpose.

5.5.4 Qualitative Analysis

5.5.4.1 Detection with Contour

Fig. 5.8 gives some example results with their contour predictions. It can be observed that the proposed model can effectively extract text instance for these cases. Two things to be noted here: (1) For uber text, sometimes the text instance is annotated as a line of text. The learned contour net can be extracted effectively. (2) Even when the text contour detection is not perfect (see the third image), by jointly learning the contour as well as the detection visual features, the model can still give an accurate prediction. This joint learning scheme differs from the
Figure 5.7: F-1 score w.r.t. the IOU threshold in scene text detection evaluation for three different models. The \textit{CascadeContourNet}_2 and \textit{AuxiliaryContourNet}_1 consistently outperform baseline model.

idea [25] as even though contour is not perfect, the detection results could still be good. Note that all these visualized results are generated from regular training setting, not from the model in capacity study.

\subsection{5.5.4.2 Comparison with Baseline}

Fig. 5.9 shows the examples of detection results from contour cascade model compared with the baseline method. The cascade model can gives better instance predictions.

\subsection{5.5.4.3 Failure Examples}

Several imperfect or failure examples are in Fig. 5.10. I also show the contour prediction. Some symbols are easily detected as text and will lead to imperfect predictions. This is a common mistakes made by scene text detector. When building an end-to-end pipeline with scene text reading, that could be removed partially by checking the transcribed text.
5.6 Conclusion and Future Work

In this chapter, I proposed a novel and effective training framework for improving scene text detection. The framework incorporates text instance contour segmentation to help improve scene text detection. Unlike contour segmentation in regular images, this framework is extended specifically to the scene text area. It is a more semantic contour since it is not a subset of image edges. It also contains instance-level information. Under such a definition, the proposed framework does not need any extra annotation. Traditional quadrilateral annotation is sufficient for training the model. The experimental results show that models trained with the proposed framework have improved capacity and generalizability.
Figure 5.9: Comparison of the detection results from the proposed cascade contour model with baseline. For each column: (1) detection results from proposed model, (2) predicted contour and (3) detection results from baseline model.

Figure 5.10: Some imperfect or failure examples.
6.1 Introduction

In this chapter, I introduce a model that can verify the existence of a text string in an image in an end-to-end manner.

Figure 6.1: An example application of scene text verification task. The proposed model takes a business storefront image and a potential business name, and then directly outputs how likely the storefront image matches the business name. The red text means it is the ground truth for this image and the proposed model gives it a high score.
The traditional solution is to detect and recognize all the text in the image so that text-based matching can be conducted during the querying stage. However, in many situations, an explicit text transcriber is unnecessary. Often a context is available. This can be used to focus the attention (text candidates). For example, if a person is attempting to find reviews of a restaurant, he or she might like to simply take a picture of the restaurant. This model can then provide the reviews for them. In such cases, the algorithm only needs to pay attention to the name of the restaurant in order to identify it. It does not need to read any other text on the street, such as a 20% off advertising promotion for clothes or the opening hours on the door. Furthermore, a list of restaurant candidates could be obtained based on the geo-location. Therefore, the actual problem that needs to be solved is the likelihood that a sequence of words is in the image.

Using an undiscriminating transcriber to answer this question is problematic since it may provide extra transcriptions that are irrelevant to the question (i.e., noise) and/or provide incorrect transcriptions that confuse the later process (text matching).

In order to address these concerns, I propose a new end-to-end model that takes input an image and text string pair and predicts the possibility that the text string is contained in the image, as shown in Fig. 6.1. The model is expected to give a high probability if the input text sequence is in the image, and thus it attempts to verify the existence of the text string. It could also be regarded as attempting to find a match between an image and a text string.

Many applications could be built if the model was able to give such a unified text-image score. For example, in addition to the restaurant case, Google Street View contains countless images taken on the street. Such a model could enable Google Street View to identity the business name from storefront images for backend data processing.

The model is called *Guided Attention*. It does not need to perform any explicit text detection or recognition. Instead it uses an attention mechanism that is guided by the input text string and that decides whether the string is in the image.

This is the first work that has sought to solve this problem in an end-to-end manner, and I study the problem in the context of business matching (i.e., given a storefront image, the model should be able to predict which business it represents).
I use a dataset consisting of millions of business storefront images collected through Google Maps API \(^1\) for this task. Each image is associated with a list of candidate business names, and the algorithm must find the correct candidate among them.

I call this the Street View Business Matching (SVBM) dataset. My experiments show the effectiveness of the proposed model in solving this challenging real-world problem.

The contributions of this chapter are:

1. **New Problem** I study a new problem: scene text verification in an image. The goal of the task is to verify the existence of a certain text string and is closely related to scene text reading. It could be tackled with a traditional method based on scene text detection and recognition. It could also be seen as a sub-problem for scene text retrieval.

2. **New Dataset** A newly collected large-scale dataset (SVBM) that contains 1.5 million images is used for the experiments. The dataset is substantially larger than any existing public scene text datasets (e.g., ICDAR2015 contains only 1,500 images). The dataset contains storefront images from Google Street View, and the algorithm must verify the existence of the business names in the images. The dataset contains various of images with different languages and is much more challenging than existing public scene text datasets.

3. **New Framework** I propose an end-to-end trainable model to address this problem. This model does not perform or require any explicit text detection or recognition. The model is completely different from traditional methods and requires only image-text pairs for training rather than a fully annotated dataset (e.g., bounding box labels), the latter of which is more expensive to create than simple yes/no labels. This greatly reduces the burden of dataset collection. The model achieves better performance in terms of its precision/recall curve than existing text reading-based solutions on SVBM. This new model provides insight into the scene text area and could inspire further research works.

\(^1\)https://developers.google.com/maps/
The proposed model has a loose connection to visual question answering [99] or image text matching [100]. However, the proposed model also has several unique properties: (1) The input text is character sequences instead of words. (2) Simple binary classification is used as the evaluation metric. (3) The order of input words should not affect the results. For example, if the sequence of words “Street View Image” is contained in an image, then “Image Street View” should also be considered as positive.

4. **Extensive Experiments** Extensive experiments have been done to study different properties of the framework. Major experiments have been performed using the SVBM dataset. I have also evaluated the trained model using two public datasets: UberText [47] and FSNS [101]. These experiments confirm that the proposed *Guided Attention* model is better suited for solving this task than a more traditional scene text reading-based solution. By combining the proposed method and the traditional method, an even better performance on the task can be achieved. Ablation studies have also been conducted; these studies show how important some design aspects of the model are for performance.

6.2 **Method**

6.2.1 **Model Architecture**

The architecture is shown in Fig. 6.2. The proposed model consists of two major components: (1) a CNN-based image encoder network with coordinate encoding, (2) a guided attention decoder which selectively pools features from the encoded feature map and generates the final result.

6.2.1.1 **CNN Encoder with Coordinate Map**

I trimme InceptionV3 [102] to construct image encoder, which builds a deep CNN by stacking several carefully designed sub-network structures with different scales.

Let \( I \) be the input image, and CNN encoded visual feature is denoted as \( f_v = \text{CNN}(I) \). In order to capture the spatial information for each location in the
feature map, I follow [4] to create a coordinate map $f_{xy}$ for each image. The equation of such coordinate encoding could be expressed as $f_{xy} = Ecode(i, j)$. $i, j$ denotes the x, and y indices of each spatial location in the feature map. Function $Encode$ computes the one-hot encoding for each positioning $i, j$, and I concatenate the coordinate map $f_{xy}$ with original cnn feature $f_v$ in depth dimension. I use $\tilde{f}$ to denote the features augmented with position information. By combining such position information with each feature vector, the following decoding process could refer to them for better attention calculation and achieves better decoding results. Such scheme has been adopted by [4, 36] and has been proved to be effective in scene text related tasks. Fig 6.3 illustrates the coordinate encoding.

### 6.2.1.2 Guided Attention Decoder

In the next step, the model tries to decode useful information from the cnn features $\tilde{f}$ guided by the input character string. Let $N$ be the number of characters of the input string and $S = S_1, \ldots, S_N$ be the character level one-hot encoding of it. The embeddings of the characters are represented as $S^e = S_1^e, \ldots, S_N^e$, which are learned end-to-end during training. The goal is to compute

$$p_{valid} = \mathbb{P}(y = 1|S_1, \ldots, S_N, I),$$  \hspace{1cm} (6.1)$$

where $y \in \{0, 1\}$ is the indicator of the existence of the text.
Figure 6.3: Illustration of encoding the coordinates as a one-hot feature vector $f_{xy}$ and concatenate it with the original visual features $f_v$. A feature map with $H \times W \times C$ will have corresponding coordinate feature map with size $H \times W \times (H+W)$.

LSTM [103] is used as recurrent function to encode sequential features. Let us denote $h_t$ as the hidden state of the LSTM in time step $t$, then the update function of hidden state could be expressed as Eq. 6.2.

$$h_t = \text{LSTM}(h_{t-1}, S_t^e, CT(x_t)),$$

(6.2)

where $CT(x_t)$ represents the context vector generated in time step $t$. It can be computed based on Eq. 6.3.

$$CT(x_t) = \sum_{i=1}^{W} \sum_{j=1}^{H} \alpha^t_{(i,j)} \ast \tilde{f}_{(i,j)},$$

(6.3)

$\alpha^t_{(i,j)}$ represents the attention map. It is computed based on Eq. 6.4. $e^t_{(i,j)}$ represents how relevant is the feature $\tilde{f}_{(i,j)}$ to the current character embedding $S_t^e$. In this work, I choose to use attention function proposed in [104] as Eq. 6.5.

$$\alpha^t_{(i,j)} = \frac{\exp(e^t_{(i,j)})}{\sum_{u=1}^{W} \sum_{v=1}^{H} \exp(e^t_{(u,v)})},$$
\[ c'_{(i,j)} = f_{\text{attn}}(h_{t-1}, \tilde{f}_{(i,j)}), \quad (6.4) \]
\[ f_{\text{attn}}(h_{t-1}, \tilde{f}_{(i,j)}) = v^T \tanh(W h_{t-1} + U \tilde{f}_{(i,j)}), \quad (6.5) \]

\( W, V \) are weight matrix that could be learned.

Let us denotes \( y = y_1, \ldots, y_N \) as the output sequence probability and \( y' = y'_1, \ldots, y'_N \) as the groundtruth labels. Each \( y_i \) is the prediction based on information till character \( S_i^c \). In training, cross entropy is used as loss function, and only the output of the last time step for each image and candidate pair is calculated as in Eq. 6.6.

\[ \text{loss} = -\frac{1}{M} \sum_{i=1}^{M} (y'_i \log(y'_n_i) + (1 - y'_i) \log((1 - y'_n_i))), \quad (6.6) \]

\( n_i \) is the length of the \( i \)th candidate and \( M \) is the number of training pairs.

### 6.2.2 Model Training and Sub-batch Architecture

In problem setting, each image usually contains several positive (random shuffling of positive words) and a list of negative text.

In order to save the computation, there are \( M \) parallel text input for each image and the CNN tower will only need to be computed once. The number \( M \) is equal to \( N_p + N_n \) where \( N_p, N_n \) represent the number of positive and negative examples sampled for each image, respectively. The total loss thus becomes a weighted cross entropy based on the ratio between positive examples and negative examples. In experiment, \( N_p \) is usually set to be 1, and \( N_n \) to be 4. So there are 5 parallel recurrent sequences for each convolutional tower.

### 6.2.3 Hard Negative Mining(HNM)

During model training, the sampling scheme of the \( N_n \) number of negative training examples plays an important role for better performance. In text verification, the hardness of a negative candidate could be empirically determined by the edit distance of it with the corresponding positive candidates. Fig. 6.4 illustrates this idea.
Figure 6.4: Example showing how to determine the hardness of a negative example based on the corresponding positive candidates. The different color represents the different hardness levels.

It is also observed that, in the SVBM dataset, most negative candidates are easy cases as they differ from the positive text by quite a lot. However, the negative samples that really confuse the network are from those hard cases. So I incorporate hard negative example mining as a fine tuning process for the trained model. The hardness of an example is defined based on Eq. 6.7.

$$\text{Hardness}(\hat{\text{nég}}) = 1 - \min_{\forall \hat{\text{pos}} \in \hat{P}} \frac{\text{edit_dis}(\hat{\text{nég}}, \hat{\text{pos}})}{\max(\text{len}(\hat{\text{pos}}), \text{len}(\hat{\text{nég}}))}$$ (6.7)

$\hat{P}$ represents the positive example set for a specific image. $\hat{\text{pos}}$ is one positive sample and $\hat{\text{nég}}$ represents the negative sample that needs to compute. The function edit_dis calculates the edit distance between the positive sample $\hat{\text{pos}}$ and the negative sample $\hat{\text{nég}}$. The hardness of a negative sample is determined by the minimal edit distance between the negative candidate and all the positive samples. The higher the score, the harder the negative sample.

The training process including HNM is thus as follows: (1) Train the network from scratch with evenly sampled positive and negative text for each image. (2) Finetune the network with evenly sampled positive text and evenly sampled negative text whose hardness score is larger than $T$. In experiments, I set $T = 0.3$ to keep relatively harder text without removing too much of them. Note that I did this for both training set and testing set, so in the second phase, the testing examples are harder. During the first phase, the classification accuracy reached 95%, but in the beginning of the second phase, the classification accuracy dropped
to 88%. This means that the harder examples, based on the proposed definition, are actually more difficult for the model.

6.3 Experiments

The major experiments are on the SVBM dataset which contains millions of images, and I use it to study different properties of the model as well as the problem of scene text verification. I have also done two experiments on UberText and FSNS datasets with the model trained on SVBM dataset and baseline methods.

6.3.1 Dataset Description and Experiment Setting

6.3.1.1 SVBM

The SVBM dataset is based on Street View images, and each image represents one business storefront. The storefront images could be obtained by the method proposed in [105], and all the images have associated geo-locations (e.g., lat/lon) so that I can retrieve a list of candidate business names by filtering out far-away business. The number of candidates depends on the business density and the search radius. The goal is to find the correct business name among the list of candidates. No text bounding boxes are provided.

The dataset contains 1.39M training images and around 147K testing images. They are collected from various countries such as US, Brazil, England and etc. with various different languages. Each image has a number of candidates ranging from 10 to over 500. One of them is the correct business name, and all others are incorrect. As a preprocessing step, I convert all business names into lower case. So the character set contains a total of 37 alphanumeric characters (including space) plus characters in other languages which have high frequency in training set. I evaluate the dataset based on precision recall curve on a per-candidate basis.

In training, rmsprop i used as optimization method, and the learning rate is set to 0.001. The batch size is set to 32, and the input image is resized and padded to 400 * 400 with random noise for simplicity. Each image is associated with 5 candidate text during training, and thus the batch size for attention decoder is 160(32 * 5). I use 70 cpu machines for training and it takes over 20 days to
converge to an initial checkpoint.

6.3.1.2 UberText and FSNS

There is currently no public dataset that is perfectly suitable for the proposed problem. I choose two public datasets that are relatively larger with a little bit different evaluation schemes. This study aims at demonstrating that by combining the proposed model with other traditional approaches, a better performance can be achieved for the text verification problem.

UberText [47] is a newly released dataset for scene text related problems. It is currently the largest fully annotated public dataset and contains 10K testing images. I evaluate it using the model trained on SVBM to show that the trained model can generalize well to other datasets. I choose the words that are of type: “business names” in the dataset and only evaluate the recall of these positive text in the verification problem.

FSNS [101] contains french street signs. The images are much easier than that in SVBM because the text are focused and clear. I randomly sample 49 text as negative text for each image for evaluation purpose.

6.3.2 Qualitative Results

In this section, I give several visual results illustrating the different properties of the trained guided attention model. More visual examples are in Fig. 6.9.

<table>
<thead>
<tr>
<th>The Little Barbers</th>
<th>0.9625</th>
<th>The Little Barbers</th>
<th>0.9625</th>
<th>NHS</th>
<th>0.8947</th>
</tr>
</thead>
<tbody>
<tr>
<td>Little Barbers</td>
<td>0.9524</td>
<td>Little Barber The</td>
<td>0.8749</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The</td>
<td>0.4268</td>
<td>Barber Little The</td>
<td>0.9249</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Little</td>
<td>0.4317</td>
<td>Barber The Little</td>
<td>0.9517</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Barber</td>
<td>0.9624</td>
<td>Little The Barber</td>
<td>0.9614</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Barber shop</td>
<td>0.1306</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.5: A qualitative evaluation on several specific behaviors of the proposed model when it is given the same input image with different input text. On the left is the input image. The first section in the table on the right shows results when input is a subset of ground truth text. The 2nd section in the table demonstrates when randomly shuffled groundtruth text is used as input. The last section represents results when the input text are not the corresponding business name, but are also in the image.
6.3.2.1 Subset of Text

During training and subsequent evaluations, I use the full business name. However, it is interesting and important to study the property of the model when I use subsets of the business name, or a slightly different business name to see how it performs. This is because that during annotation, it is possible to obtain not exactly the same text as the actual business name. For example, “The Little Barbers” might be annotated as “Little Barbers”, and the proposed model is expected to still give a high score for that. The first section in the table of Fig. 6.5 shows an example image and several such input text strings with their predicted probabilities.

It can be seen that, when I use informative subset text (e.g., “Little Barbers”) from the ground truth as input text to the model, the model still gives a pretty high score. This meets expectation as I care about the existence of the string (words) in the image. However, there are also several interesting findings: (1) If non-informative words are used as input (e.g: “The”), the model gives a relatively low score. (2) If text string that contains other words which are not in the image (e.g., “Barber Shop”) is used, the model gives a low score. These findings are interesting, and I believe that the model is looking for more informative words and makes decisions based on that. “The” is common in business names, so the model gives a lower score if I only use that as input text. “Barber shop” contains other words, which could possibly indicate that it’s some other business’ name. So the proposed model gives a low score to it even though it contains the word “Barber” that is in the image.

6.3.2.2 Random Shuffled Text

As discussed before, the proposed model should be able to ignore the shuffling of input words. The 2nd section of the table in Fig. 6.5 shows the results when shuffled text is used as input with the same image into the model. This property is also important, as the order of words in annotation has no relationship with the spacial ordering of those words in image. Thus the trained model should ignore the word order, and only focus on the existence of the collection of words.

The model is also somewhat invariant to the random shuffling of words as all
the of them received high scores. This is the property that I expect, and it is important to text verification.

### 6.3.2.3 Non-Business Text

The last section of the table in Fig. 6.5 shows an example where I have text inputs that are not the specific business name in the ground truth. The model still gives it a high score since it is in the image, and it also meets expectation. There is also a failure example when “PRIVATE DENTIST” is used as input. This might happen when text are too small w.r.t the image size, so the attention could not capture it well. In addition to that, the fact that the model is trained on business related text, might also caused the failure of this non-business text.

<table>
<thead>
<tr>
<th>Before Hard Negative Mining</th>
<th>After Hard Negative Mining</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peluqueria estetica</td>
<td>0.9324</td>
</tr>
<tr>
<td>stela</td>
<td></td>
</tr>
<tr>
<td>peluqueria toni</td>
<td>0.7613</td>
</tr>
<tr>
<td>francesco hidalgo tello</td>
<td>0.2643</td>
</tr>
<tr>
<td>ferreteria mheva sl</td>
<td>0.2232</td>
</tr>
<tr>
<td>puertodental sl</td>
<td>0.1989</td>
</tr>
<tr>
<td>Peluqueria estetica</td>
<td>0.8924</td>
</tr>
<tr>
<td>stela</td>
<td></td>
</tr>
<tr>
<td>peluqueria toni</td>
<td>0.3293</td>
</tr>
<tr>
<td>francesco hidalgo tello</td>
<td>0.1375</td>
</tr>
<tr>
<td>ferreteria mheva sl</td>
<td>0.1328</td>
</tr>
<tr>
<td>puertodental sl</td>
<td>0.1276</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Before Hard Negative Mining</th>
<th>After Hard Negative Mining</th>
</tr>
</thead>
<tbody>
<tr>
<td>ceip santa catalina</td>
<td>0.9324</td>
</tr>
<tr>
<td>aulario santa catalina</td>
<td>0.9057</td>
</tr>
<tr>
<td>trastero 16</td>
<td>0.0781</td>
</tr>
<tr>
<td>ropa africana</td>
<td>0.0618</td>
</tr>
<tr>
<td>verde limon</td>
<td>0.0606</td>
</tr>
<tr>
<td>peluqueria vanitas</td>
<td>0.0605</td>
</tr>
<tr>
<td>ceip santa catalina</td>
<td>0.8604</td>
</tr>
<tr>
<td>aulario santa catalina</td>
<td>0.1294</td>
</tr>
<tr>
<td>verde limo</td>
<td>0.1186</td>
</tr>
<tr>
<td>la pizarra</td>
<td>0.0970</td>
</tr>
<tr>
<td>trastero 16</td>
<td>0.0868</td>
</tr>
<tr>
<td>apartamento primas</td>
<td>0.0819</td>
</tr>
</tbody>
</table>

Figure 6.6: Two visual examples illustrating the performance gain after mining hard negative candidates for training. I observe that after HNM, the gap between the best prediction and the second best prediction has usually been increased.

### 6.3.2.4 Hard Negative Mining

Fig. 6.6 shows two examples illustrating the performance of the model trained before and after hard negative mining.

After hard negative mining, the probability margins between the positive and best negative sample for the two images have increased by 40%. This makes the model trained with HNM much more robust and the quantitative results of such comparison are in the followings.
Figure 6.7: (a) The Precision-Recall curve of the proposed model compared with other baselines. (b) The Precision-Recall curve of the proposed model w.r.t. different maximum length of text. (c) The Precision-Recall curve comparison before and after HNM.

6.3.3 SVBM Quantitative Evaluation

6.3.3.1 Baseline Models

I compare several baseline models with the proposed approach: (1) Google Cloud OCR (GCOCR) \(^1\). (2) Attention OCR (OCR) [4]. (3) Show and Tell Model [106] with binary classification output.

Model (1) tries to detect and read all the text indiscriminately. Then text based matching with the candidates is conducted to find the best candidate. This is one typical paradigm for this problem, but the model is trained on other fully annotated dataset. Model (2) is trained on the SVBM dataset directly. I simply force it to transcribe the whole image into the positive business name it represents. Text based matching is performed afterwards. Model (3) is a modification based on [104] (by changing the output as a binary classification). It could also be regarded as removing the attention mechanism in the proposed model. So it is denoted as “no attention” in experiment.

6.3.3.2 Comparison w.r.t baselines

I first show the comparison of the final model against other baseline methods. Fig. 6.7a shows the Precision/Recall curve. The proposed end-to-end model (after HNM) outperforms all other baselines by a large margin. This address two points: (1) text detector and recognizer trained on other fully annotated data couldn’t

\(^1\)https://cloud.google.com/vision/docs/ocr
achieve good results in SVBM dataset because they couldn’t accurately find and transcribe the business name. (2) Text verification could be learned in an end-to-end way, and it outperforms transcription based method (OCR). In the following two evaluations, I show different settings that can improve the performance.

6.3.3.3 Evaluation w.r.t Maximum Length

The maximum length of the character sequence could be tuned as an hyperparameter. This is an interesting aspect of the proposed scene text verification problem. This is because when deciding which candidate business name the storefront represents, sometimes, for example, only the first 20 characters are needed. In the example Fig 6.4, only the first 15 characters are needed to determine which candidate is the positive one.

Note that this maximum length $N$ only affects those candidates with length longer than it. I simply cut the longer text to keep the first $N$ characters. This is also a difference between text verification problem w.r.t traditional VQA problem, since usually question is not cut in VQA task.

The Precision/Recall curve w.r.t the maximum length is in Fig. 6.7b. The peak is when the maximum length set to 40. However, other lengths also achieve reasonable performance. Besides, the value is dataset dependant. This experiment aims at illustrating an unique property of scene text verification and is important in deployment since the shorter the sequence, the faster the model could run. So there is a trade-off between the performance and efficiency. 40 is used as final model choice.

6.3.3.4 Evaluation w.r.t Hard Negative Mining

Whether to use hard negative mining makes a big difference in terms of the performance of the trained model. In Fig. 6.7c, I show the results of the models trained before and after hard negative mining. Together I also show the results produced by a model with “no attention”. It can be seen that the attention mechanism is important to get good performance for scene text verification tasks. By adding hard negative mining process, the performance has been further improved, especially in the high accuracy region of the curve.
6.3.4 UberText and FSNS Quantitative Evaluation

In this quantitative study, I study the problem that whether the proposed model can be combined with traditional approaches to better solve text verification problem. The compared ensembled approaches are following: (1) OCR+: Ensemble of Attention OCR [4] and the proposed approach by taking the maximum of the scores output by the two models as the final score. This way of ensemble is used in all of the following compared approaches. (2) GCOCR+: Simple ensemble of GCOCR\(^1\) and the proposed approach. (3) OCR*2+: Simple ensemble of GCOCR\(^1\), OCR [4] and the proposed approach.

Fig. 6.8a, 6.8b show the results for UberText and FSNS, respectively. The results show that when combining the proposed model with other traditional approaches, a better results can be achieved compared with the original model itself. It also demonstrates that the “knowledge” learned from different models are not the same, and the proposed model could also serve as complementary resources of information for traditional approaches.

6.4 Conclusion

In this work, I proposed a new problem: verifying the existence of a text string in an image. I collected a large-scale dataset (SVBM) for for evaluating the performance of my model in addressing this problem. Instead of relying on a traditional
approach based on scene text reading, I propose an end-to-end model that takes both the image and the text as inputs and gives a unified result. I experimentally proved that model designed based on this framework achieved an improved performance using SVBM for matching businesses with their storefront images. This framework can be combined with traditional methods to create even more powerful models, as demonstrated in the experiments using two public datasets. This work does not aim at establishing the most sophisticated architecture, but at proving that an end-to-end solution for such a task can be developed to achieve improved performance without the need to fully annotate images at the text level.
Figure 6.9: Example images from the test set. I only visualize the top predicted candidates. The first three rows show the examples that we give good predictions (high score for positive candidate, low score for negative ones). The last row shows several failure examples.
Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this dissertation, I investigated various machine learning models for scene text understanding in natural images. I focused on the scene text detection and verification tasks and proposed four models/frameworks for detection and one end-to-end model for verification. For the detection task, the proposed models/frameworks range from a region proposal method with a CNN classifier for horizontal text detection to a keypoint model that can detect arbitrary-oriented text. For the verification task, an end-to-end model is proposed and demonstrated with experiments on large-scale, real-world image datasets.

Chapter 2 introduced a CNN model that aggregates local context for character-level classification. I designed a pipeline that incorporates a heuristic region proposal method with the designed CNN architecture in a cascaded fashion. By aggregating local context, the classifier can give more accurate predictions. After classifying each character proposal, several post-processing steps are used to group the characters into text lines. The method follows the traditional text detection pipeline. It is limited in that it can only detect horizontal text and many post-processing steps are needed.

Chapter 3 introduced a multi-scale FCN text block detector followed by instance-aware segmentation for text detection. The multi-scale FCN aggregates features from multiple scales and is more robust in segmenting text pixels. The segmented results could be used to extract text blocks that might contain multiple text in-
stances. An instance-aware segmentation step for each text block was proposed. This involves first extracting the text center line and then using it as an instance clue to further segment the text instance. The method can detect arbitrary-oriented text. The concept of text center line extraction has been used in various works [16]. Its advantage is its ability to tackle arbitrary-oriented text. However, many steps with CNN processing are involved. The method is thus not efficient compared with other contemporary regression-based or proposal-based text detectors.

In Chapter 4, I introduced a design framework (DAS) for arbitrary-oriented scene text detection. The framework contains three steps: detect, associate, and segment. The first step is to detect instance clues, and the second step is to associate the detected instance clues so as to uniquely identify each text instance. The third step is to segment the text instances unambiguously with the associated instance clues. A keypoint model was designed under this framework and achieves state-of-the-art performance using various benchmark datasets with both multi-oriented and arbitrary-oriented text.

In Chapter 5, I introduced a novel training framework to improve scene text detection. An auxiliary task, contour segmentation, was introduced and I discussed how to incorporate it into an existing scene text detector. Two designs were discussed and compared. The first design simply uses this task as an auxiliary loss during training. The second design uses it in a cascaded fashion and feeds the output segmentation results as feature maps for detection. Experiments on benchmark datasets demonstrated that such a task can improve performance. The cascaded design achieves the best performance gain.

Chapter 6 introduced a machine learning model for scene text verification. Unlike most traditional approaches that detect and read text in an image and do text-based matching, the proposed method verifies the existence of a text string in an image in an end-to-end manner and gives a unified score for each text string and image pair. It uses an attention mechanism guided by the input character string to implicitly localize the character sequence and verify the existence of the input text. Experiments on real large-scale dataset show the superiority of the proposed model.
7.2 Future Work

This dissertation investigated various designs for scene text detection and verification.

In terms of the detection task, it would be interesting to further investigate models under the proposed DAS framework. For example, designing models that could automatically determine how many key points to be used to better capture the shape of text instances would be interesting. It would also be helpful if a standardized annotation setting was proposed that would facilitate methods based on keypoint detection. In terms of the proposed keypoint model, designing architecture that seamlessly incorporates the segmentation branch into the model would make it much more efficient.

For the verification task, different attention mechanisms could be investigated with their performance comparison in this task. It would also be interesting to see how to leverage existing fully labeled datasets to help weakly labeled datasets in the verification task.
Appendix A

Supplementary Material for Chapter 3
Figure A.1: More results of the word or text line instance segmentation
Figure A.2: More results of the word or text line instance segmentation
Supplementary Material for Chapter 4

In this supplementary material, I will give more visualized results of the proposed model and a prototype model based on the DAS framework. Specifically, Sec B.1 gives more visualized results from corner association. Sec B.2 describes annotation process and visualizes several annotated training images. Sec B.3 briefly describe a prototype model design based on detecting text center line [16] as instance clue.

B.1 Corner Association Results

Fig. B.1 and Fig. B.2 contains more visualized results of corner association.

B.2 Annotation for TotalText Dataset

TotalText dataset contains text bounding boxes with polygon annotations. Each polygon contains different number of points. In order to change it into 6 corner points annotation, the following process is conducted.

Let’s denote $N_k$ as the number of points for text polygon $polygon_k$. If $N_k$ is odd, I will mask out the text region in the original image. Thus it will not affect training. If $N_k$ is even, then I will further check if it can be divide by 4. If $mod(N_k, 4) = 2$, then it means each long side has odd number of points. I calculate the index for $c_1$ and $c_2$ as $\lfloor \frac{N_k}{4} \rfloor$ and $\lfloor \frac{N_k}{4} \rfloor + \lfloor \frac{N_k}{2} \rfloor$, respectively. It basically
the median point on each side. If $\text{mod}(N_k, 4) = 0$, then it means each long side has even number of points. Then $c_1, c_2$ are the median points of the center two points on each side. Take $c_1$ as an example, it is the median of two points with index $\lfloor \frac{N_k}{4} - 1 \rfloor$ and $\lfloor \frac{N_k}{4} \rfloor$.

Fig. B.3 visualizes several training examples with the 6 corner points annotation and contains masked out text regions.

\section*{B.3 Text Center Line Model Prototype}

In this section, I briefly introduce another model design based on text center line for arbitrary oriented text detection under the proposed DAS framework. Note that this model is not optimized so I only gives a brief description of the pipeline and some visualized qualitative result. It aims at proving the fact that the DAS framework is general and many model pipeline could be designed based on it for arbitrary oriented scene text detection.

The pipeline of the prototype is in Fig. B.4. Given an image, I first use an CNN encoder decoder network to predict a text-nontext segmentation map as well as a
Figure B.2: More results of the corner association step. Focused on multi-oriented text. Better when zoomed in.

text center line map. This is the same as in [16] and it belongs to the “Detect” step. The second step is association. I simply group the pixels which belongs to the text center line together using connected component (CC) analysis. Then each CC represents one text instance. The last step segment takes one of the text center line CC and the corresponding cropped image as input and generate the output instance segmentation map. Fig. B.5 visualizes several end-to-end segmentation results based on the model.
Figure B.3: Visualized corner points annotation.

Figure B.4: The pipeline of the prototype model using Text Centerline Segmentation.
Figure B.5: Results produced by the prototype model based on text centerline. Each row from left to right: the input image, the text-nontext segmentation results, the text centerline segmentation results and the final instance segmentation results.
Bibliography


URL https://doi.org/10.24963/ijcai.2017/458


Vita
Dafang He

Education

The Pennsylvania State University, University Park, PA, USA
Ph.D., Information Sciences and Technology, 2019.

Tsinghua University, Beijing, China
B.S., Automation, 2014.

Experience

Facebook, PhD Software Engineer Intern, Summer 2018

Google, PhD Software Engineer Intern, Summer 2017

Adobe Research, Computer Vision Research Intern, Summer 2016

Selected Publications


