Scene text detection

Nguyen Duy Khanh, Nguyen Phuc Thanh, Nguyen Duc Quyet, Nguyen Huu Tuan Duy, Nguyen Phuong Uyen

Hanoi University of Science and Technology
Symposium On Information and Communication Technology

1. Introduction

Scene text detection simply means finding the regions in the image where the text can be present (often in bounding boxes).

While performing text detection, we may encounter two types of cases:

- **Images with Structured text**: This refers to the images that have a clean/uniform background with regular font. Text is mostly dense with proper row structure and uniform text color.
- **Images with Unstructured text**: This refers to the images with sparse text in a complex background. The text can have different colors, size, fonts, and orientations and can be present anywhere in the image. Performing text detection on these images is known as /textbfscene text detection.

From the descriptions above, it is presumed that the Scene text detection is more challenging, due to various aforementioned difficulties.

While performing text detection, we have 3 options:

- Character level detection
- Word level detection
- Line level detection In this project, we deal with word level in Scene text detection case.

2. Related work

The text detection methods can be broadly classified into 2 categories:

- Convectional methods

  Conventional methods rely on manually designed features. For instance, Stroke width Transform (SWT) and Maximally Stable Extremal Regions (MSER) based meth-ods generally extracts the character candidates via edge detection or extremal region extraction. While in the deep learning based methods, features are learned from the training data. These are generally better than the conventional ones, in terms of both accuracy and adaptability in challenging scenarios.

- Deep learning based method There are 2 main approachs
  - **Regression based method**: Regression-based methods are a series of models which directly regress the bounding boxes of the text instances.Regression-based methods usually enjoy simple post-processing algorithms (e.g. nonmaximum suppression). However, most of them are limited to represent accurate bounding boxes for irregular shapes, such as curved shapes.
  - **Segmentation based method**: usually combine pixel-level prediction and post-processing algorithms to get the bounding boxes.

3. Dataset

The dataset used is provided by VinAI and BKAI. The VinAI data has 2000 images for training, and BKAI has 500 images for validating and testing.

The ground truth is given as separate text files (one per image) where each line specifies the coordinates of one word's bounding box and its transcription in a comma-separated format.

"Do Not Care" regions are indicated in the ground truth with a transcription of "###"

4. Data augmentation

4.1. Basic augmentation methods

In order to increase the variety of our training data, we apply the following functions to the image:

- Flip
- Add noise(we use Gaussian noise)
- Rotate and re-scale
- Random crop

4.2. Advanced augmentation

We will try to imitate natural characteristics

- Contrast: Camera sensors have many imperfections and tunable settings
- Grid: More suitable for STD than some object detection augmentation like CutOut, CutMix and MixUp
- Color:
- Shadow, Rain, Snow: Scene text may be captured under different weather conditions

5. Metric

evaluating performance of an object detector boils down to determining if a detection is correct or not.

Definition of terms:

- True Positive (TP) — Correct detection made by the model.
- False Positive (FP) — Incorrect detection made by the detector.
- False Negative (FN) — A Ground-truth missed (not detected) by the object detector.

- True Negative (TN) —This is backgound region correctly not detected by the model. This metric is not used in object detection because such regions are not explicitly annotated when preparing the annotations. To define those terms, we need another helper metric called Intersection over Union (IoU).

To define those terms, we need another helper metric called **Intersection over Union (IoU)**.

Intersection over Union (IoU): IoU metic in object detection evaluates the degree of overlap between the ground(gt) truth and prediction(pd). The ground-truth and the prediction can be of any shape-rectangular box, circle, or even irregular shape). It is calculated as follows:

$$IoU = \frac{gt \cap pd}{gt \cup pd}$$

IoU ranges between 0 and 1 where 0 shows no overlap and 1 means perfect overlap between gt and pd. IoU is useful through thresholding, that is, we need a threshold (α, say) and using this threshold we can decide if a detection is correct or not.

For IoU threshold at α, True Positive(TP) is a detection for which IoU(gt,pd) α and False Positive is a detection for which IoU(gt,pd)< α. False Negative is a ground-truth missed together with gt for which IoU(gt,pd)< α.

Precision is the degree of exactness of the model in identifying only relevant objects. It is the ration of TPs over all detections made by the model.

$$P = \frac{TP}{TP + FP} = \frac{TP}{alldetections}$$

Recall measures the ability of the model to detect all ground truths— proposition of TPs among all ground truths .

$$R = \frac{TP}{TP + FN} = \frac{TP}{allgroundtruths}$$

F1 score is harmonic mean of precision and recall (supposed that these two value are not zero)

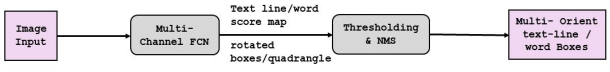
$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

6. Method

6.1. EAST

EAST is a deep learning-based algorithm that detects text with a single neural network with the elimination of multi-stage approaches. The key component of this algorithm is a neural network model, which is trained to directly predict the existence of text instances and their geometries from full images. The model is a fully-convolutional neural network adapted for text detection that outputs dense per-pixel predictions of words or text lines. This eliminates intermediate steps such as candidate proposal, text region formation and word partition. The post-processing steps only include thresholding and NMS on predicted geometric shapes.

6.1.1. *Pipeline*



The general pipeline of this approach is as shown in the figure below. At first, an image is sent to a **Fully Convolutional Network(FCN)** in which pixel-level text score maps and geometry maps are generated. This way a general Dense box is generated. two geometric shapes for text regions are available. One is a rotated box(RBOX) and another is a quad box(QUAD). The loss function for both the maps ie score map and geometry map is generated then Thresholding is applied on that and if the score is greater than the predefined prediction that region is passed further to the **non-max suppression (NMS)**. And the output after NMS is the final output.

6.1.2. *Neural Network Design*

There are many aspects that should be taken care of while designing a neural network. There is a lot of variety in the size of images and text in the natural scene. The geometry of the text cannot be generalized easily. For this Hypernet is used in the feature maps. By the use of Hypernet and UShape a network that can utilize different levels of features and with minimum computation cost. The schematic view of this approach is shown below.

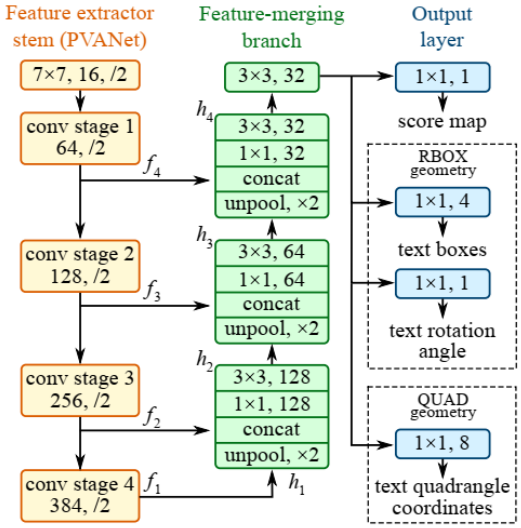


Figure 1: Structure of our text detection FCN

This model can be categorized into three branches:

- Feature Extracting Stem
- Feature Merging Branch
- Output layer

6.1.3. *Feature Extracting Stem*

The stem can be a convolutional network pre-trained on ImageNet [4] dataset, with interleaving convolution and pooling layers. Four levels of feature maps, denoted as f_i, are extracted from the stem, whose sizes are $\frac{1}{32}$, $\frac{1}{16}$, $\frac{1}{8}$ and $\frac{1}{4}$ of the input image, respectively. In Fig. 1, PVANet [17] is depicted. In our experiments, we also adopted the well-known VGG16 [32] model, where feature maps after pooling-2 to pooling-5 are extracted.

6.1.4. *Feature Extractor*

This part can be any convolutional neural network with convolutional layer and pooled layer interleaving pre-trained on Imagenet data for examples PVANet, VGG16 and RESNET50. From this network, four levels of feature maps f1, f2, f3 and f4 can be obtained. Because we are extracting features it is called a Feature Extractor.

6.1.5. *Feature-merging Branch*

In the feature-merging branch, we gradually merge them:

$$g_i = \begin{cases} \text{unpool}(h_i) & \text{if } i \leq 3 \\ \text{conv}_{3\times3}(h_i) & \text{if } i = 4 \end{cases} \tag{1}$$

$$h_i = \begin{cases} f_i & \text{if } i = 1 \\ \text{conv}_{3\times3}(\text{conv}_{1\times1}([g_{i-1}; f_i])) & \text{if } i = 4 \end{cases} \tag{2}$$

where g_i is the merge base, and hi is the merged feature map, and the operator $[;]$ represents concatenation along the channel axis. In each merging stage, the feature map from the last stage is first fed to an unpooling layer to double its size, and then concatenated with the current feature map. Next, a $conv_{1\times1}$ bottleneck [8] cuts down the number of channels and reduces computation, followed by a $conv_{3\times3}$ that fuses the information to finally produce the output of this merging stage. Following the last merging stage, a $conv_{3\times3}$ layer produces the final feature map of the merging branch and feed it to the output layer.

The number of output channels for each convolution is shown in Fig. 1. We keep the number of channels for convolutions in branch small, which adds only a fraction of computation overhead over the stem, making the network computation-efficient. The final output layer contains several $conv_{1\times1}$ operations to project 32 channels of feature maps into 1 channel of score map F_s and a multi-channel geometry map F_g.

For RBOX, the geometry is represented by 4 channels of axis-aligned bounding box AABB R and 1 channel rotation angle . The formulation of R is the same as that in [9], where the 4 channels represents 4 distances from the pixel location to the top, right, bottom, left boundaries of the rectangle respectively.

For QUAD Q, we use 8 numbers to denote the coordinate shift from four corner vertices $\{p_i|i \in \{1, 2, 3, 4\}\}$ of the quadrangle to the pixel location. As each distance offset contains two numbers $(\Delta x_i, \Delta y_i)$, the geometry output contains 8 channels.

6.1.6. *Label Generation*

Score Map Generation for Quadrangle

Without loss of generality, we only consider the case where the geometry is a quadrangle. The positive area of the quadrangle on the score map is designed to be roughly a shrunk version of the original one, illustrated in Fig. 4 (a).

For a quadrangle $\mathbf{Q} = \{p_i|i \in \{1, 2, 3, 4\}\}$, where $p_i = \{x_i, y_i\}$ are vertices on the quadrangle in clockwise order. To shrink $\mathbf{Q}$, we first compute a reference length r_i for each vertex p_i as

$$\begin{aligned} r_i = min(&D(p_i, p_{(i \mod 4)+1}), \\ &D(p_i, p_{((i+2) \mod 4)+1}) \end{aligned} \tag{3}$$

Geometry Map Generation

The geometry map is either one of RBOX or QUAD. The generation process for RBOX is illustrated in Fig. 4 (c-e).

For those datasets whose text regions are annotated in QUAD style (e.g., ICDAR 2015), we first generate a rotated rectangle that covers the region with minimal area. Then for each pixel which has positive score, we calculate its distances to the 4 boundaries of the text box, and put them to the 4 channels of RBOX ground truth. For the QUAD ground truth, the value of each pixel with positive score in the 8-channel geometry map is its coordinate shift from the 4 vertices of the quadrangle.

6.1.7. *Loss Functions*

The loss can be formulated as

$$L = L_{\text{s}} + \lambda_{\text{g}} L_{\text{g}}$$

where L_{s} and L_{g} represents the losses for the score map and the geometry, respectively, and λ_{g} weighs the importance between two losses. In our experiment, we set λ_{g} to 1 .

Loss for Score Map

In most state-of-the-art detection pipelines, training images are carefully processed by balanced sampling and hard negative mining to tackle with the imbalanced distribution of target objects [9, 28]. Doing so would potentially improve the network performance. However, using such techniques inevitably introduces a non-differentiable stage and more parameters to tune and a more complicated pipeline, which contradicts our design principle.

To facilitate a simpler training procedure, we use classbalanced cross-entropy introduced in [38], given by

$$\begin{aligned} L_{\text{s}} = &\text{ balanced-xent } \left(\hat{\mathbf{Y}}, \mathbf{Y}^*\right) \\ &= -\beta \mathbf{Y}^* \log \hat{\mathbf{Y}} - (1-\beta)\left(1-\mathbf{Y}^*\right) \log(1-\hat{\mathbf{Y}}) \end{aligned}$$

where $\hat{\mathbf{Y}} = F_{\text{s}}$ is the prediction of the score map, and $\mathbf{Y}^*$ is the ground truth. The parameter β is the balancing factor between positive and negative samples, given by

$$\beta = 1 - \frac{\sum_{y^* \in \mathbf{Y}^*} y^*}{|\mathbf{Y}^*|}.$$

This balanced cross-entropy loss is first adopted in text detection by Yao et al. [41] as the objective function for score map prediction. We find it works well in practice.

Loss for Geometries

One challenge for text detection is that the sizes of text in natural scene images vary tremendously. Directly using L1 or L2 loss for regression would guide the loss bias towards larger and longer text regions. As we need to generate accurate text geometry prediction for both large and small text regions, the regression loss should be scale-invariant. Therefore, we adopt the IoU loss in the AABB part of RBOX regression, and a scale-normalized smoothed-L1 loss for QUAD regression.

RBOX For the AABB part, we adopt IoU loss in [46], since it is invariant against objects of different scales.

$$L_{\text{AABB}} = -\log \text{IoU}\left(\hat{\mathbf{R}}, \mathbf{R}^*\right) = -\log \frac{\left|\hat{\mathbf{R}} \cap \mathbf{R}^*\right|}{\left|\hat{\mathbf{R}} \cup \mathbf{R}^*\right|}$$

where $\hat{\mathbf{R}}$ represents the predicted AABB geometry and $\mathbf{R}^*$ is its corresponding ground truth. It is easy to see that the width and height of the intersected rectangle $\left|\hat{\mathbf{R}} \cap \mathbf{R}^*\right|$ are

$$w_{\mathbf{i}} = \min\left(\hat{d}_2, d_2^*\right) + \min\left(\hat{d}_4, d_4^*\right)$$
$$h_{\mathbf{i}} = \min\left(\hat{d}_1, d_1^*\right) + \min\left(\hat{d}_3, d_3^*\right)$$

where d_1, d_2, d_3 and d_4 represents the distance from a pixel to the top, right, bottom and left boundary of its corresponding rectangle, respectively. The union area is given by

$$\left|\hat{\mathbf{R}} \cup \mathbf{R}^*\right| = |\hat{\mathbf{R}}| + |\mathbf{R}^*| - \left|\hat{\mathbf{R}} \cap \mathbf{R}^*\right|.$$

Therefore, both the intersection/union area can be computed easily. Next, the loss of rotation angle is computed as

$$L_\theta\left(\hat{\theta}, \theta^*\right) = 1 - \cos\left(\hat{\theta} - \theta^*\right).$$

where $\hat{\theta}$ is the prediction to the rotation angle and θ^* represents the ground truth. Finally, the overall geometry loss is the weighted sum of AABB loss and angle loss, given by

$$L_{\mathrm{g}} = L_{\mathrm{AABB}} + \lambda_\theta L_\theta.$$

Where λ_θ is set to 10 in our experiments.

Note that we compute L_{AABB} regardless of rotation angle. This can be seen as an approximation of quadrangle IoU when the angle is perfectly predicted. Although it is not the case during training, it could still impose the correct gradient for the network to learn to predict $\hat{\mathbf{R}}$.

QUAD We extend the smoothed-L1 loss proposed in [6] by adding an extra normalization term designed for word quadrangles, which is typically longer in one direction. Let all coordinate values of Q be an ordered set

$$C_{\mathbf{Q}} = \{x_1, y_1, x_2, y_2, ..., x_4, y_4\}$$

then the loss can be written as

$$\mathrm{L_g} = L_{\mathbf{QUAD}}\left(\hat{\mathbf{Q}}, \mathbf{Q}^*\right) = \min_{\tilde{\mathbf{Q}} \in P_{\mathbf{Q}^*}} \sum_{c_i \in C_{\mathbf{Q}}, \tilde{c}_i \in C_{\tilde{\mathbf{Q}}}} \frac{\text{smoothed}_{L1}(c_i - \tilde{c}_i)}{8 \times N_{\mathbf{Q}^*}}$$

where the normalization term NQ is the shorted edge length of the quadrangle, given by

6.2. PSE

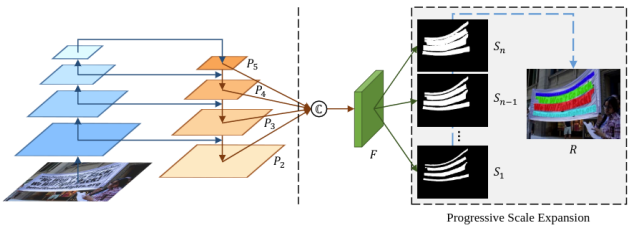


Figure 2: Illustration of PSENet's overall pipeline. The left part is implemented from FPN. The right part denotes the feature fusion and the progressive scale expansion algorithm.

With the development of Convolutional Neural Network, a lot of methodologies based of bounding box regression has been proposed to locate the rectangle or quadrangle bounding box with certain orientations. The backward of these frameworks is that natural scenes can be text with arbitrary shapes.

Many semantic segmentation-based methods can be applied to handle the curve text detection problems. Although pixel-wise segmentation can extract the regions of arbitrary-shaped text instances, it may still fail to separate two text instances when they are relatively close, because their shared adjacent boundaries will probably merge them together as one single text instance. To address this problem, Progressive Scale Expansion Network (PSENet) is proposed with two advantages: able to locate texts with arbitrary shapes, and able to identify closely adjacent text instances.

6.2.1. *Pipeline*

The overall pipeline of the proposed PSENet is illustrated in Fig. 2 Inspired by FPN, low-level feature maps are concatenated with high-level feature maps and there are four concatenated feature maps. These maps are further fused in F to encode informations with various receptive views. Intuitively, such fusion is very likely to facilitate the generations of the kernels with various scales. Then the feature map F is projected into n branches to produce multiple segmentation results S_1, S_2, ..., S_n. Each S_i would be one segmentation mask for all the text instances at a certain scale. The scales of different segmentation mask are decided by the hyper-parameters. Among these masks, S_1 gives the segmentation result for the text instances with smallest scales (i.e., the minimal kernels) and S_n denotes for the original segmentation mask (i.e., the maximal kernels). After obtaining these segmentation masks, progressive scale expansion algorithm is used to gradually expand all the instances' kernels in S_1, to their complete shapes in S_n, and obtain the final detection results as R.

6.2.2. *Progressive Scale Expansion Algorithm*

=

Algorithm 1 Scale Expansion Algorithm

Require: Kernels: C, Segmentation Result: S_i
Ensure: Scale Expanded Kernels: E

function EXPANSION(C, S_i)
  $T \leftarrow \emptyset$; $P \leftarrow \emptyset$; $Q \leftarrow \emptyset$
  for each $c_i \in C$ **do**
    $T \leftarrow T \cup \{(p, label) \mid (p, label) \in c_i\}$
    $P \leftarrow P \cup \{p \mid (p, label) \in c_i\}$
    Enqueue(Q, c_i) // push all the elements in c_i into Q
  end for
  while $Q \neq \emptyset$ **do**
    $(p, label) \leftarrow$ **Dequeue**(Q) // pop the first element of Q
    if $\exists q \in$ **Neighbor**(p) and $q \notin P$ and $S_i[q] =$ True **then**
      $T \leftarrow T \cup \{(q, label)\}$; $P \leftarrow P \cup \{q\}$
      Enqueue($Q, (q, label)$) // push the element $(q, label)$ into Q
    end if
  end while
  $E =$ **GroupByLabel**(T)
  return E
end function

Figure 3: The pseudocode of the progressive scale expansion algorithm.

The procedure of scale expansion is illustrated in Fig. 3. The expansion is based on BreadthFirst-Search algorithm which starts from the pixels of multiple kernels and iteratively merges the adjacent text pixels. Note that there may be conflicted pixels during expansion. The principle to deal with the conflict in our practice is that the confusing pixel can only be merged by one single kernel on a first-come-first-served basis. Thanks to the "progressive" expansion procedure, these boundary conflicts will not affect the final detections and the performances. In the pseudocode, T, P are the intermediate results. Q is a

queue. Neighbor(·) represents the neighbor pixels of p. GroupByLabel(·) is the function of grouping the intermediate result by label. "$S_i[q]$ = True" means that the predicted value of pixel q in S_i belongs to the text part.

6.2.3. *Label Generation*

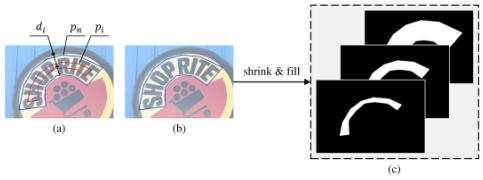


Figure 4: The illustration of PSENet's label generation

As illustrated in Fig. 4 , PSENet produces segmentation results (e.g. S_1, S_2, ..., S_n) with different kernel scales. Therefore, it requires the corresponding ground truths with different kernel scales as well during training. In our practice, these ground truth labels can be conducted simply and effectively by shrinking the original text instance. The polygon with blue border in Fig. 4 (b) denotes the original text instance and it corresponds to the largest segmentation label mask (see the rightmost map in Fig. 4 (c)). To obtain the shrunk masks sequentially in Fig. 4 (c), the Vatti clipping algorithm is utilized to shrink the original polygon p_n by d_i pixels and get shrunk polygon p_i (see Fig. 4 (a)). Subsequently, each shrunk polygon p_i is transferred into a 0/1 binary mask for segmentation label ground truth. We denote these ground truth maps as G_1, G_2, ..., G_n respectively. Mathematically, if we consider the scale ratio as r_i , the margin d_i between p_n and p_i can be calculated as:

$$d_i = \frac{Area(p_n) * (1{r_i}^2)}{Perimeter(p_n)}$$

where Area(·) is the function of computing the polygon area, Perimeter(·) is the function of computing the polygon perimeter. Further, we define the scale ratio r_i for ground truth map G_i as:

$$r_i = \frac{1 - (1 - m) * (n - i)}{n - 1}$$

where m is the minimal scale ratio, which is a value in (0, 1]. The values of scale ratios (i.e., r_1, r_2, ..., r_n) are decided by two hyper-parameters n and m, and they increase linearly from m to 1.

6.2.4. *Loss Function*

For learning PSENet, the loss function can be formulated as:

$$L = \lambda L_c + (1 - \lambda) L_s$$

where L_c and L_s represent the losses for the complete text instances and the shrunk ones respectively, and λ balances the importance between L_c and L_s.

It is common that the text instances usually occupy only an extremely small region in natural images, which makes the predictions of network bias to the non-text region, when binary cross entropy is used. In this project, we adopt dice coefficient in our experiment. The dice coefficient $D(S_i, G_i)$ is formulated as:

$$D(S_i, G_i) = \frac{2 \sum_{x,y} (S_{i,x,y} * G_{i,x,y})}{\sum_{x,y} S_{i,x,y}^2 + \sum_{x,y} G_{i,x,y}^2}$$

where and refer to the value of pixel (x, y) in segmentation results S_i an ground truth G_i, respectively.

Online Hard Example Mining (OHEM) is adopted to L_c during training to better distinguish these patterns. L_c focuses on segmenting the text and non-text region. Let us consider the training mask given by OHEM as M, and thus L_c can be written as:

$$L_c = 1 - D(S_n * M, G_n * M)$$

L_s is the loss for shrunk text instances. Since they are encircled by the original areas of the complete text instances, we ignore the pixels of non-text region in the segmentation result Sn to avoid a certain redundancy. Therefore, L_s can be formulated as follows:

$$L_s = 1 - \frac{\sum_{i=1}^{n-1} D(S_i * W, G_i * W)}{n - 1}$$

$$W_{x,y} = \begin{cases} 1, & if \ \ S_{n,x,y} \geq 0.5; \\ 0, & otherwise \end{cases}$$

Here, W is a mask which ignores the pixels of non-text region in S_n, and $S_{n,x,y}$ refers to the value of pixel (x, y) in S_n.

6.2.5. *Implementation Details*

The backbone of PSENet is implemented from FPN. We firstly get four 256 channels feature maps (i.e. P_2, P_3, P_4, P_5) from the backbone. To further combine the semantic features from low to high levels, we fuse the four feature maps to get feature map F with 1024 channels via the function C(·) as:

$$F = C(P_2, P_3, P_4, P_5) = P_2 || Up_{\times 2}(P_3) || Up_{\times 4}(P_4) || Up_{\times 8}(P_5)$$

where "||" refers to the concatenation and $Up_{\times 2}(.)$, $Up_{\times 4}()$, $Up_{\times 8}()$ refer to 2, 4, 8 times upsampling, respectively. Subsequently, F is fed into Conv(3, 3)-BN-ReLU layers and is reduced to 256 channels.Next, it passes through multiple Conv(1, 1)-Up-Sigmoid layers and produces n segmentation results S_1, S_2, ..., S_n. Here, Conv, BN, ReLU and Up refer to convolution, batch normalization, rectified linear units and upsampling.

We set n to 6 and m to 0.5 for label generation and get the scales 0.5, 0.6, 0.7, 0.8, 0.9, 1.0. During training, we ignore the blurred text regions labeled as DO NOT CARE in all datasets. The λ of loss balance is set to 0.7. The negative-positive ratio of OHEM is set to 3. The data augmentation for training data is listed as follows:

- The images are rescaled with ratio 0.5, 1.0, 2.0, 3.0 randomly;
- Tthe images are horizontally fliped and rotated in range $[-10°, 10°]$ randomly;
- 640 × 640 random samples are cropped from the transformed images;
- The images are normalized using the channel means and standard deviations

For quadrangular text dataset, we calculate the minimal area rectangle to extract the bounding boxes as final predictions. For curve text dataset, the RamerDouglas-Peucker algorithm is applied to generate the bounding boxes with arbitrary shapes.

6.3. DBNet

Most segmentation-based methods require complex post-processing for grouping the pixel-level prediction results into detected text instances, resulting in a considerable time cost in the inference procedure. The aforementioned PSENet [3] proposed the post-processing of progressive scale expansion for improving the detection accuracies

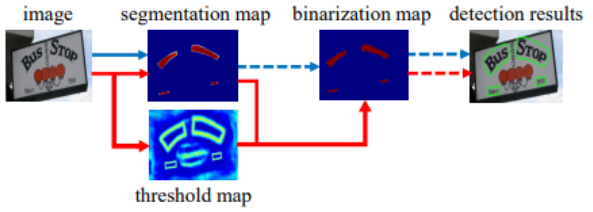


Figure 5: Traditional pipeline (blue flow) and our pipeline (red flow). Dashed arrows are the inference only operators; solid arrows indicate differentiable operators in both training and inference.

Most existing detection methods use the similar postprocessing pipeline as shown in 5(following the blue arrows): Firstly, they set a fixed threshold for converting the probability map produced by a segmentation network into a binary image; Then, some heuristic techniques like pixel clustering are used for grouping pixels into text instances. Alternatively, our pipeline (following the red arrows in Fig. 2) aims to insert the binarization operation into a segmentation network for joint optimization. In this manner, the threshold value at every place of an image can be adaptively predicted, which can fully distinguish the pixels from the foreground and background. However, the standard binarization function is not differentiable, we instead present an approximate function for binarization called Differentiable Binarization (DB) [2], which is fully differentiable when train

6.3.1. *System description*

Firstly, the input image is fed into a feature-pyramid backbone. Secondly, the pyramid features are up-sampled to the same scale and cascaded to produce feature F. Then, feature F is used to predict both the probability map (P) and the threshold map (T). After that, the approximate binary map $\hat{B}$ is calculated by P and F. In the training period, the supervision is applied on the probability map, the threshold map, and the approximate binary map, where the probability map and the approximate binary map share the same supervision. In the inference period, the bounding boxes can be obtained easily from the approximate binary map or the probability map by a box formulation module. (Fig 6)

6.3.2. *Binarization*

Standard binarization:Given a probability map $P \in R^{H*W}$ produced by a segmentation network, where H and W indicate the height and width of the map, it is essential to convert it to a binary map $P \in R^{H*W}$, where pixels with Standard binarization Given a probability map $P \in R^{H*W}$ produced by a segmentation network, where H and W indicate the height and width of the map, it is essential to convert it to a binary map $P \in R^{H*W}$, where pixels with value 1 is considered as valid text areas. Usually, this binarization process can be described as follows:

where t is the predefined threshold and (i, j) indicates the coordinate point in the map.

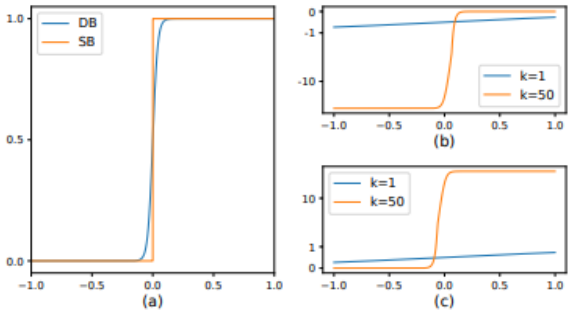


Figure 7: Illustration of differentiable binarization and its derivative. (a) Numerical comparison of standard binarization (SB) and differentiable binarization (DB). (b) Derivative of l+. (c) Derivative of l-

Differentiable binarization: The standard binarization described in Eq. 1 is not differentiable. Thus, it can not be optimized along with the segmentation network in the training period. To solve this problem, we propose to perform binarization with an approximate step function:

$$\hat{B}_{i,j} = \frac{1}{1 + e^{-k(P_{i,j} - T_{i,j})}}$$

where $\hat{B}$ is the approximate binary map; T is the adaptive threshold map learned from the network; k indicates the amplifying factor. k is set to 50 empirically. This approximate binarization function behaves similar to the standard binarization function (see 6) but is differentiable thus can be optimized along with the segmentation network in the training period. The differentiable binarization with adaptive thresholds can not only help differentiate text regions from the background, but also separate text instances which are closely jointed.

The reasons that DB improves the performance can be explained by the backpropagation of the gradients. Lets take the binary cross-entropy loss as an example. Define $f(x) = 1/(1+e^{-kx})$ as our DB function, where $x = P_{i,j} - T_{i,j}$. Then the losses l_+ for positive labels and l_- for negative labels are:

$$l_+ = -\log \frac{1}{1 + e^{-kx}}$$

$$l_- = -\log (1 - \frac{1}{1 + e^{-kx}})$$

We can easily compute the differential of the losses with the chain rule:

$$\frac{\partial l_+}{\partial x} = -k * f(x) * e^{-kx}$$

$$\frac{\partial l_-}{\partial x} = k * f(x)$$

The derivatives of l_+ and l_- are also shown. We can perceive from the differential that (1) The gradient is augmented by the amplifying factor k; (2) The amplification of gradient is significant for most of the wrongly predicted region (x < 0 for L_+; x > 0 for L_-), thus facilitating the optimization and

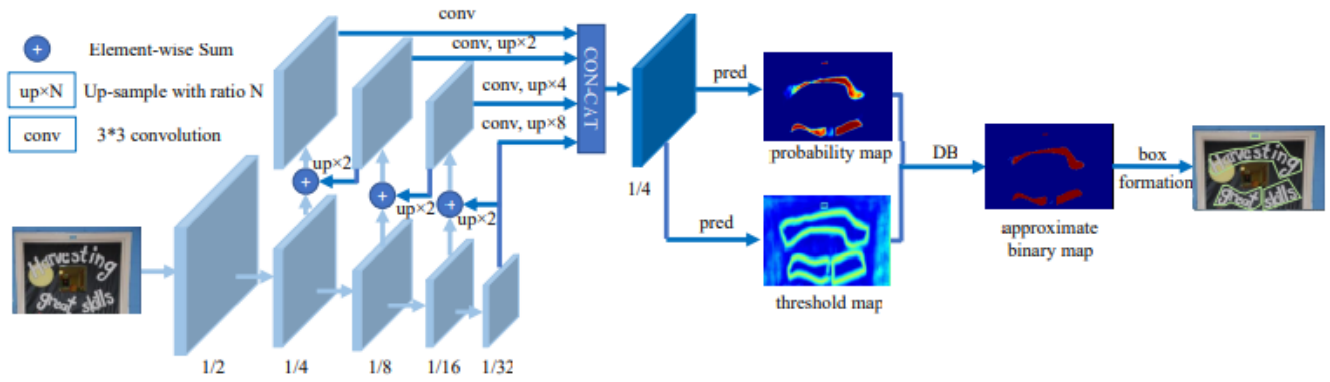


Figure 6: Architecture of our proposed method, where "pred" consists of a 3×3 convolutional operator and two de-convolutional operators with stride 2. The "1/2", "1/4", ... and "1/32" indicate the scale ratio compared to the input image

helping to produce more distinctive predictions. Moreover, as $x = P_{i,j} - T_{i,j}$, the gradient of P is effected and rescaled between the foreground and the background by T.

6.3.3. *Adaptive threshold*

The threshold map with/without supervision is visualized in Fig. 6. The threshold map would highlight the text border region even without supervision for the threshold map. This indicates that the border-like threshold map is beneficial to the final results. Thus, we apply borderlike supervision on the threshold map for better guidance.

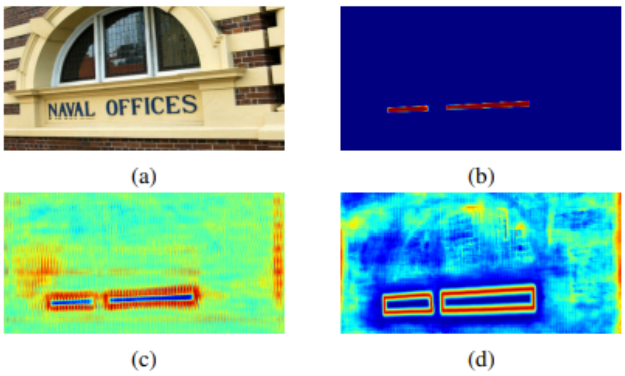


Figure 8: The annotation of text polygon is visualized in red lines. The shrunk and dilated polygon are displayed in blue and green lines, respectively

6.3.4. *Deformable convolution*

Deformable convolution [1] can provide a flexible receptive field for the model, which is especially beneficial to the text instances of extreme aspect ratios. Following [4], modulated deformable convolutions are applied in all the 3 × 3 convolutional layers in stages conv3, conv4, and conv5 in the ResNet-18 backbone.

6.3.5. *Label generation*

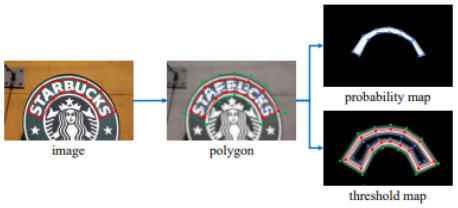


Figure 9: The annotation of text polygon is visualized in red lines. The shrunk and dilated polygon are displayed in blue and green lines, respectively

The label generation for the probability map is inspired by PSENet. Given a text image, each polygon of its text regions is described by a set of segments:

$$G = \{S_k\}_{k=1}^{n}$$

n is the number of vertexes, which may be different in different datasets. For the dataset that we use, n equals to 4. The positive area is generated by shrinking the polygon G to Gs using the Vatti clipping algorithm (Vati 1992). The offset D of shrinking is computed from the perimeter L and area A of the original polygon:

$$D = \frac{A(1 - r^2)}{L}$$

where r is the shrink ratio, set to 0.4 empirically.

With a similar procedure, we can generate labels for the threshold map. Firstly the text polygon G is dilated with the same offset D to Gd. We consider the gap between Gs and Gd as the border of the text regions, where the label of the threshold map can be generated by computing the distance to the closest segment in G.

6.3.6. *Optimization*

The loss function L can be expressed as a weighted sum of the loss for the probability map L_s, the loss for the binary map L_b, and the loss for the threshold map L_t

$$L = L_s + \alpha * L_b + \beta * L_t$$

where L_s is the loss for the probability map and L_b is the loss for the binary map. According to the numeric values of the losses, α and β are set to 1.0 and 10 respectively.

We apply a binary cross-entropy (BCE) loss for both Ls and Lb. To overcome the unbalance of the number of positives and negatives, hard negative mining is used in the BCE loss by sampling the hard negatives.

$$L_s = L_b = \sum_{i \in S_l} y_i \log x_i + (1 - y_i) \log(1 - x_i)$$

S_l is the sampled set where the ratio of positives and negatives is 1 : 3

Lt is computed as the sum of L1 distances between the prediction and label inside the dilated text polygon Gd:

$$L_t = \sum_{i \in R_d} |Y_i^* - x_i^*|$$

where Rd is a set of indexes of the pixels inside the dilated polygon G_d; y^* is the label for the threshold map. In the inference period, we can either use the probability map or the approximate binary map to generate text bounding boxes, which produces almost the same results. For better efficiency, we use the probability map so that the threshold branch can be removed. The box formation process consists of three steps: (1) the probability map/the approximate binary map is firstly binarized with a constant threshold (0.2), to get the binary map; (2) the connected regions (shrunk text regions) are obtained from the binary map; (3) the shrunk regions are dilated with an offset D' the Vatti clipping algorithm(Vati 1992). D' is calculated as:

$$D' = \frac{A' * r'}{L'}$$

here A' is the area of the shrunk polygon; L' is the perimeter of the shrunk polygon; r' is set to 1.5 empirically.

6.3.7. *Experiment*

The model is trained with gpu 1080, with batch size equal to 16 and learning rate is set to 0.003. Optimizer Adam is the one that I used since it claimed to be good in general task

I try different set up like using augmentation and deformable convolution or not to see if they actually benefit the model. All the set ups use the same backbone, which is resnet-18

Here's the result:

basic augment	advanced augment	deformable	F1-score
			0.898
v			0.928
v	v	v	0.931

We can see that the combination of advanced augmentation and deformable convolution does not increase the performance of the model much. I think the reason is that the data set we used is quite easy, which no arbitrary shape like curve, ...Since deformable convolution is more beneficial to that situation, apply it in a 4-point polygon bounding boxes dataset will not enhance the robustness much

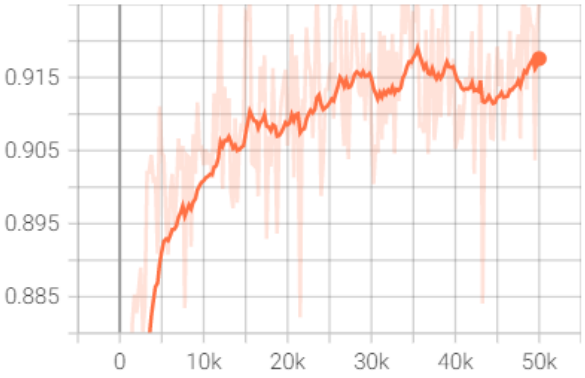


Figure 10: validation F1-score after each epoch

7. Overall results

Model	F1	precision	recall
DBnet	0.931	0.922	0.926
EAST	0.84	0.814	0.868
PSE	0.847	0.92	0.786

8. Conclusion

In this project, we have read and understand different approach for scene text detection task. We also try to implement and train the model in order to reproduce the results of mentioned papers

9. References

[1] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017.

[2] Minghui Liao, Zhaoyi Wan, Cong Yao, Kai Chen, and Xiang Bai. Real-time scene text detection with differentiable binarization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 11474–11481, 2020.

[3] Wenhai Wang, Enze Xie, Xiang Li, Wenbo Hou, Tong Lu, Gang Yu, and Shuai Shao. Shape robust text detection with progressive scale expansion network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9336–9345, 2019.

[4] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9308–9316, 2019.