Automatic Container Code Recognition via Faster-RCNN

Wang Zhiming, Wang Wuxi
School of Computer and Communication Engineering
University of Science and Technology Beijing, China
e-mail: wangzhiming@ustb.edu.cn,
wuxiwang25@163.com

Xing Yuxiang
Department of Engineering Physics
Tsinghua University, Beijing, China
e-mail: xingyx@mail.tsinghua.edu.cn

***Abstract*—Automatic container code recognition (ACCR) plays an important role in customs logistics and transport management. Due to complicated lighting conditions and background pollution, automatic detection and recognition of container codes remains a difficult task. In this work, we exploit Faster-RCNN, a robust object detection algorithm based on deep learning algorithm, to detect and recognize container codes. First, container code characters are detected as 36 classes of small objects, consisting of 26 capitals and 10 digits. Next, a novel post processing algorithm based on binary search tree is adopted to find container code from detected characters. Experimental results validate the proposed approach, and the overall accuracy on a dataset with 831 container codes achieves 97.71%.**

Keywords-container code recognition; deep learning; object detection; faster-RCNN

I. INTRODUCTION

Ship containers are widely used in modern transportation. Each container holds a unique identification serial code. Automatic recognition of these codes plays a key role in making the transportation efficient and economical. Since 1990's [1], various algorithms have been proposed for automatic reading of container codes. However, there are still many difficulties to deal with. The main difficulties come from: (1) corrugated container surface, variations in color, font type and font size, blur, noise and optical distortions. (2) External lighting factors such as illumination variation, rain and fog.

In recent years, deep learning has achieved great success in many areas such as image classification and object detection. For object detection, some algorithms provide excellent results, such as RCNN [2], Fast-RCNN [3], Faster-RCNN [4], YOLO [5], SSD [6] and RetinaNet [7]. Though container code can hardly be considered as one object due to its complex arrangement, it can be regarded as composed of several small objects (characters). From this perspective, we can adopt one of objection detection algorithms aforementioned to detect and recognize container codes.

In this paper, we propose a container code recognition method briefly comprised of two phases, one is characters detection and recognition, and the other is post processing for container code search. In the first phase, we adopt Faster-RCNN to detect and recognize candidate characters, which gives robust detection result and guarantees the performance of our automatic container codes recognition system. In the second phase, a novel post processing is used to combine valid characters into a container code. The elaborate designed searching algorithm is quite suitable for finding most likely container code from multiple detected characters each with a recognition probability.

II. RELATED WORKS

A. Container Code Recognition

From 1990, there have many studies on container code extraction and recognition. Goccia's [8] method is robust to large variation of brightness yet sensitive to damaged characters, segmentation uncertainties and distortion due to its edge detection process. Igual adopted top-hot transform to obtain characteristics of objects after segmentation[9]. Kwang-Baek et al. [10] proposed an adaptive resonance theory based self-organizing supervised learning algorithm to recognize characters extracted by a fuzzy method. But it but fails when the character is damaged or the information is lost during the binarization process. Tai [11] extracts interesting regions by edge detection and grouping, then detects and segments container codes by horizontal and vertical projection. However, some characters are arranged close to each other, which make the projection operation fail to separate them. Tseng [12] proposed an algorithm that filtered noise by morphological operation and extracted characters by 8-adjacent connected component labeling. Finally, a multi-template was constructed to solve the multi-font problem. Chen [13] segmented the characters by the locally thresholding and geometry features. Then, these characters were used to construct the eigen-feature model based on principal component analysis, which was used to identify the character appearance.

Deep learning has become popular since Hiton's eminent work in 2006 [14]. So far, many advances based on convolutional neural network (CNN) have already been achieved in ACCR field. Wu [15] firstly located code-character regions by a horizontal high-pass filter and scan line analysis. Next, they extracted candidate regions and classified them into single-character blocks and multi-character blocks. Finally, they recognized characters by a segmentation-based approach for the single-character block and a hidden Markov model (HMM) based method for the multi-character one. Lang et al. [16] used the projection method and connected component analysis (CCA) to isolate character region and then the interesting characters were

978-1-7281-3326-3/19/$31.00 ©2019 IEEE

recognized by CNN algorithm and template matching (TM) algorithm. Finally, a combinatorial strategy was proposed to obtain the container code. Yoon et al. [17] adopted a multiple sequence alignment (MSA) to select the optimal alignment code from multiple views, which improved container code recognition accuracy. Ankit et al. [18] proposed an end-to-end pipeline using Region Proposals generated by Connected Components for text detection, then they used two Spatial Transformer Networks for text recognition, one for alphabets and the other for digits. Lin et al. [19] adopted maximally stable extremal region (MSER) to extract connected component for geometrical clustering and spatial structure template matching for location and various CNN-classifiers for identification.

B. *Object Detection*

Generic object detection based on deep learning has been extensively studied, and the performance has been greatly improved in recent years. There are mainly two categories of algorithms, one-stage detectors and two-stage detectors.

One-stage detectors: The three mainly approaches in one-stage detection category are YOLO [5], SSD [6] and RetinaNet [7]. YOLO is the first algorithm where object detection is reframed as a regression problem. It directly predicts bounding boxes and class probabilities in a single feed forward pass for raw images thus speeding up the detector. Since there are only two boxes predicted in each grid, YOLO struggles with small objects. SSD, which used a dense set of boxes and combined predictions from multiple feature maps with different resolutions to handle objects of various sizes, improved the performance compared with YOLO. Because of sampling from these dense set of detections at test time, it gives lower performance on MS COCO [20] dataset whose objects tend to be smaller as compared to two-stage detectors. RetinaNet claims the best accuracy even compared with two-stage approaches due to its novel focal loss function. However, experimental results in our container recognition show its performance is worse than the two stage Faster RCNN [4].

Two-stage detectors: The most popular two-stage detectors are R-CNN [2], Fast-RCNN [3] and Faster-RCNN [4]. R-CNN pioneered the development of modern detectors by combining region proposals with convolutional neural networks (CNNs). However, it has large time and space complexity due to its complex multistep training pipeline. Fast-RCNN was proposed soon after RCNN. It mainly reduced the time consuming by introducing a single step training pipeline and a multitask loss, which enabled the classification and regression heads to be trained simultaneously. However, the region proposal step is still a bottleneck due to the use of selective search algorithm. To address this problem, Faster-RCNN introduced the region proposal network (RPN) to replace selective search, which makes Faster-RCNN faster and gives higher accuracy.

Pros and Cons: One-stage frameworks, which directly map from image pixels to bounding box coordinates and class probabilities, can reduce computation complexity compared to two-stage frameworks. However, two-stage frameworks, using sliding windows, usually provide higher accuracy especial for small objects, which is more appropriate for our container code recognition task.

III. METHOD

Our main objective is to detect and recognize container codes by treating every character as a small object. A valid container code consists of 4 capitals and 7 digits. There are 26 capital candidates and 10 digital candidates, that is, A-Z and 0-9 respectively. Based on this strategy, we designed a system for container code recognition mainly involving two parts: character recognition based on Faster-RCNN and container code search based on a binary search tree. The proposed pipeline is illustrated in Fig. 1.

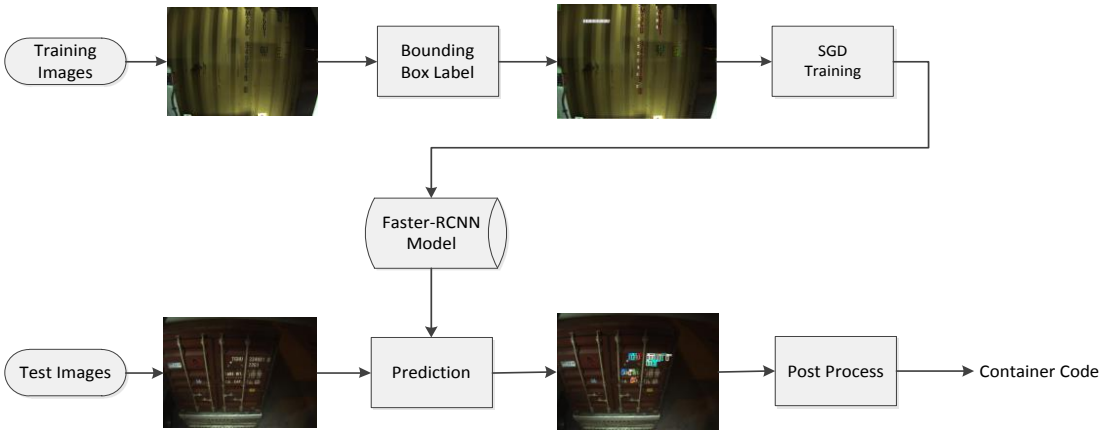

Figure 1. Container code recognition flow.

A. *Character Detection and Recognition*

Faster-RCNN is a powerful algorithm for generic object detection. Though it's a two-stage, time-consuming framework contrast to some one-stage frameworks, it meets the high accuracy demand of our task. It introduced an approach called region proposal network (RPN) to learn the "objectness" of all instances and accumulated the proposals

for the detector part of the backbone. The detector further classified and refined bounding boxes around those proposals. RPN is an efficient fully convolutional data-driven method which helps Faster-RCNN improve state-of-the-art accuracy by improving the region proposal quality.

Faster-RCNN uses a novel concept of anchor to predict regions of multiple scales and aspect ratios across the entire image. An anchor is simply a rectangular crop of an image. A set of anchors with diverse set of dimensions are used to model objects of all shapes and sizes. They are cropped out by sliding windows. These anchors are then trained end-to-end to regress to the ground truth and an "objectness" score is calculated per anchor. The anchor is translation invariant and consequently enhances the robustness of Faster-RCNN.

There are two outputs for each anchor, the target class probability and the box coordinate offset. The intersection-over-union (IoU), which is calculated with the ground truth and the anchor, is used to determine whether a class assigned to an anchor is an instance of a ground truth object or the background class. We just follow the author's settings in our experiment. Only if a region is allotted to a foreground class, its offsets are calculated as follows:

$$t_x = (x_g - x_a)/w_a \quad (1)$$

$$t_y = (y_g - y_a)/h_a \quad (2)$$

$$t_w = \log(w_g/w_a) \quad (3)$$

$$t_h = \log(h_g/h_a) \quad (4)$$

where x_g, y_g, w_g, h_g are the x, y coordinates, width and height of the ground truth box and x_a, y_a, w_a, h_a denotes the x, y coordinates, width and height of the region. Variables t_x, t_y, t_w, t_h are the target offsets.

Finally, we use Faster-RCNN to train our character detection model end to end with the Inception-Resnet [21] network as the backbone. The training algorithm enables fine-tuning all network layers, which is vital for maximum classification accuracy with very deep networks.

Figure 2. Examples of character detection result.

Fig. 2 shows two character detection results from our trained model. We can see that the Faster-RCNN robustly provides quite good detection results with high accuracy.

B. Finding the Container Code by a Binary Search Tree

The main idea of our method is to cluster container codes together by the orientation of characters. Many scene texts hold the attribute to align in the same orientation, so do container codes. Though there are many types of alignments for container codes, they are usually aligned horizontally or vertically. Therefore, we can cluster characters in the same orientation together first. These clustered characters usually constitute the entire container code or a part of it. However, sometimes characters do not align exactly in the same orientation due to image distortion. To address this problem, we cluster characters horizontally or vertically together only if the line connecting their centroids deviates from X-axis or Y-axis within a small angular range, which we called *tolerable error angle*. Suppose the range is $[a, b]$. These boundaries of angles deviation as shown in Fig. 3 are determined according to the practical experience.

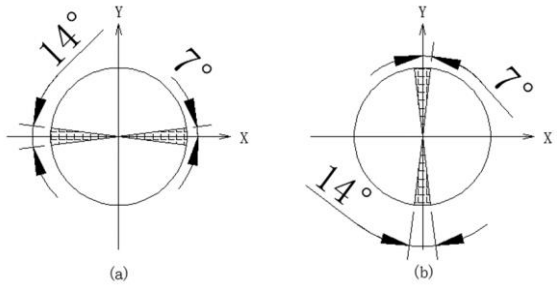

Figure 3. Tolerable error angle ranges for horizontal (a) and vertical (b).

Now we need an efficient algorithm to find which characters constitute a container code based on prior knowledge of the regulation on container code. As defined in the international standard of ISO 6346, each container code is a unique sequence consists of four capitals, six numbers, and a check digit for verification. Considering efficiency, we proposed a search algorithm based on binary search tree for container code extraction. To ensure that all candidates can be equally clustered, it is necessary to make each detected instance rather than each class to build such a tree. It can be implemented using the dictionary data structure, whose keys are indexes of detected instances. To probe the dictionary, we use the technique of lazy deletion when a character is inserted into a binary search tree. Fig.4. illustrates such a type of tree constructed starting from the candidate character '*S*'. Suppose the number of instances is *N + 1* and their indexes would start from 0 and end with *N*. We iterate each candidate instance in decreasing order according to their probabilities. In other words, instances with high probabilities have the higher priority to be visited. In practice, we need not to sort the predicted results as Faster-RCNN has already accomplished this task. '*S*' is the first one to be visited as its highest probability, and it would be inserted as the root of the tree because it is empty in the beginning. In the process of processing, those candidates clustered together with '*S*' would be inserted into proper positions of the tree as children. We use the *x*-coordinate to help us determine where to insert the node for horizontal arrangement and the *y*-coordinate for vertical case. In the example in Fig. 4, the character '*G*' lies on the left of '*S*', which means that the *x*-coordinate of its centroid is less than that of '*S*'. As a result, '*G*' was inserted as the left child of '*S*'. Similarly, the

x-coordinate of character 'U''s centroid is larger than that of 'S', and therefore it was inserted as the right child of 'S'. So on for the rest. If one character's x-coordinate is equal to that of 'S', we just give it up as 'S' is the most appropriate candidate for its position due to its highest probability. That's why we iterate the detected instance in order according to their probabilities.

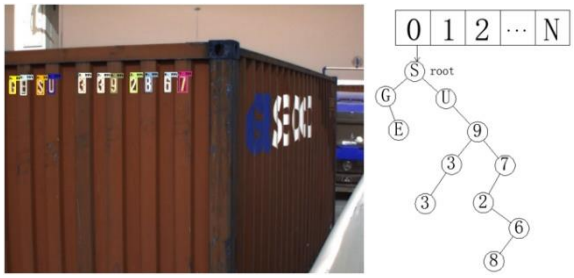

Figure 4. An example of a binary search tree held by a candidate S with the index 0.

After we completed the construction of the binary search tree for per instance, we need to collect container codes. For horizontal alignment an in-order traversal of the tree would obtain the entire container code or its components, while for vertical model an extra reverse operation is needed. We will explain it in detail in the following experiment section. As the code characters may be multi-line, by row and may also by column, we have to find all the possible components that follow the alignment rule. For instance, we need to collect components with 4 capitals, 7 digits, etc. We then combine these components together to generate the overall container code. Of course, for the best case we can directly obtain the container code whose length is 11.

The main function of post process is our cluster algorithm described in Algorithm 1. To be brief, we omit some details here (I_i denotes an instance with index i).

Algorithm 1: Make Tree for Per Instance

Input: classes, boxes, scores of predicted instances and error angles
Output: a dictionary may hold multiple binary search trees
Filter out instances whose scores are less than the score threshold
Initialize a dictionary *Dict*
For i = 0 to *len*(boxes):
  Search instance I_i in *Dict* by i
  If instance I_i has been visited:
    Continue
  Else:
    Create a new binary search tree T
    Insert instance I_i into T as root
    Mark I_i as visited
    Put T into *Dict* with index i
  Endif

  For j = 0 to *len* (boxes):
    Search instance I_j in *Dict* by j
    If instance I_j not visited:
      Compute the angle $\boldsymbol{\theta}$ between the X-axis and the line connecting centroids of I_i and I_j
      If $\boldsymbol{\theta} \in$ **error angles**: // I_j and I_i can cluster together
        Insert instance I_j into T
        Mark I_j as visited
      Endif
    Endif
  End for
End for

Though there is a nested loop in above algorithm 1, there are as many binary search trees as the number of character lines, which is very small empirically. For each tree, the average cost for each traversal is $O(\log S)$, where S represents the size of the tree with the most nodes. Likewise, S is small as many of the detected characters have been filtered out by the threshold and therefore the number of clustered candidates is usually less than or equal to 11, which is the length of a container code. As trees growing, the less candidates we need to visit and the less time we need to iterate. Suppose the number of character lines is R and the average cost of our algorithm would be $O(RS\log S)$, which implies that our cluster algorithm is quite efficient and with low computational complexity.

C. Verification and Multi-Channel Fusion

According to the international container code coding rule, not all of the characters from a container code are independent. The first 10 characters are multiplied by different weights and accumulated. Next, a modular operation on the result with 10 gives the last check digit. We verify the found container code using this coding rule, and only results that pass this verification are accepted for the following process.

Container code is painted on the container's front, rear, left, and right planes. To avoid tracking and monitoring failure, several cameras are installed to capture each plane of a container. We adopt a multi-channel fusion strategy to further increase the accuracy of recognition. That is, we do recognition for a group of images (also known as multiple channels) and each image may give a recognition result. To determine which recognition result is the mostly correct one, we calculate the probability sum of all instances constructing a container code and select the one with maximum probability sum. With this multi-channel fusion strategy, as long as one channel is recognized correctly, we will get the correct results.

IV. Experimental Results and Analyses

Implementation Details: We conducted our experiments on a server equipped with a Tesla K80 GPU with 11.92GB of RAM. The proposed method takes approx. 1.3 seconds per container image for recognition of the complete container code. Our training model is optimized by SGD with momentum = 0.9. Learning rate is set to 0.0003 and the training process is set to 200K steps.

Dataset: We use a dataset consisting of 6,302 container images from a practical project, among which about 90% labeled images are used for training and about 10% for testing. The image resolution is 960×1,280.

Score threshold: We select candidate characters from faster RCNN output for cluster according to the threshold, that is, those detected characters with scores less than the pre-defined threshold would be filtered out. The determination of the score threshold value is important. If it is set too small, too many characters would be selected for cluster. As those extra characters usually clustered together with the true container codes, it would lead to a decrease in accuracy. On the contrary, if it is set too large, there would be too less characters selected to combine into a container code. In our experiments we follow the default value of 0.5.

Comparison with RetinaNet: In our experiments the Faster-RCNN model and the RetinaNet model are based on Inception-ResNet-v2 and ResNet50 respectively. We evaluated the precision, recall and F-score of detected characters for the test set. The results are shown as Table 1 and we can see that RetinaNet is in higher precision yet with lower recall and F-score. Therefore, we can draw a conclusion that Faster-RCNN is more robust than RetinaNet.

TABLE I. CONTAINER CHARACTERS DETECTION RESULT

Detector	P	R	F
Faster-RCNN	95.59%	96.53%	96.06%
RetinaNet	96.20%	82.57%	88.87%

Finally, we list the overall recognition results in Table 2.

TABLE II. OVERALL CONTAINER CODE RECOGNITION RESULT

Dataset	# of Container Code	# of Images	Correct	Error	Reject
Test	831	2,657	97.71%	0.24%	2.05%

V. CONCLUSION

We propose a simple yet robust approach for container code recognition. In the proposed method, Faster-RCNN is adopted to obtain candidate characters which are fed to post processing based on a binary search tree to finds the container code effectively and efficiently. The proposed method gives an overall accuracy of 97.71% for a test set containing 831 container codes.

Our binary search tree based post processing can be easily extended to other similar character recognition applications, such as license plate recognition, house number recognition and wagon number recognition.

For our future work, we will focus on reducing the complexity of container codes detection and recognition model so that it can be real-time.

REFERENCES

[1] H. C. Lui, C. M. Lee, and F. Gao, "Neural networks application to container number recognition," in *IEEE 14th Annual International Computer Software and Applications Conference*, 1990.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Computer Vision and Pattern Recognition (CVPR)*, 2014.

[3] R. Girshick, "Fast r-cnn," in International Conference on Computer Vision (ICCV), 2015.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Neural Information Processing Systems (NIPS)*, 2015.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision (ECCV)*, 2016.

[7] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2018.

[8] M. Goccia, M. Bruzzo, C. Scagliola, S. Dellepiane, "Recognition of container code characters through gray-level feature extraction and gradient-based classifier optimization," in *IEEE*, 2003.

[9] I. S. Igual, G. A. García and A. P. Jiménez, "Preprocessing and recognition of characters in container codes," in *16th International Conference on Pattern Recognition*, 2002.

[10] K. B. Kim, Y. W. Woo and H. K. Yang, "An Intelligent System for Container Image Recognition Using ART2-Based Self-organizing Supervised Learning Algorithm," in *Springer*, 2006.

[11] S. C. Tai, T. Y. Chang and S. Y. Wang, "A Method of Container Codes Detection and Segmentation by Using Improved Projection Algorithm," in *World Congress on Intelligent Control and Automation (WCICA)*, 2011.

[12] C. C. Tseng, S. L. Lee, "Automatic Container Code Recognition Using Compressed Sensing Method," in *Springer*, 2011.

[13] M. Chen, W. Wu, X. Yang, X. He., "Hidden-Markov-model-based segmentation confidence applied to container code character extraction," in *IEEE Transactions on Intelligent Transportation Systems*, 2011.

[14] E. Hiton, S. Osindero, and Y. Whye, "A fast learning algorithm for deep belief nets," in *IEEE*, 2006.

[15] W. Wu, Z. Liu, M. Chen, Z. Liu., "A New Framework for Container Code Recognition by Using Segmentation-Based and HMM-Based Approaches," in *Elsevier*, 2015.

[16] L. Q. Mei, J. M. Guo, Q. Liu, P. Liu, "A Novel Framework for Container Code-Character Recognition Based on Deep Learning and Template Matching," in *IEEE*, 2016.

[17] Y. W. Yoon, K. D. Ban, H. Yoon, J. Kim, "Automatic Container Code Recognition from Multiple Views," in *ETRI Journal*, 2016.

[18] Verma, M. Sharma, R. Hebbalaguppe, E. Hassan, L. Vig, "Automatic Container Code Recognition via Spatial Transformer Networks and Connected Component Region Proposals," in *IEEE*, 2016.

[19] L. Cao, Z. G Gai, E. X. Liu, et al, "Automatic Container Code Recognition System Based on Geometrical Clustering and Spatial Structure Template Matching," in *CSPS*, 2019.

[20] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Doll´ar, and C. Lawrence Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland*, 2014.

[21] C. Szegedy, S. Ioffe, V. Vanhoucke, and Alexander A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI*, 2017.