Article:
Diao, Y, Tang, X, Wang, H et al. (2022) A large-scale container dataset and a baseline method for container hole localization. Journal of Real-Time Image Processing, 19 (3). pp. 577-589. ISSN: 1861-8200

https://doi.org/10.1007/s11554-022-01199-y

A large-scale container dataset and a baseline method for container hole localization

Yunfeng Diao[1,2], Xin Tang[1,2], He Wang[3], Emma Christophine Florence Taylor[3] Shirui Xiao[1,2]
Mengtian Xie[1,2] Wenming Cheng[1,2]
[1]Southwest Jiaotong University, China
[2] Sichuan Key Laboratory of Rail Transit Operation and Maintenance Technology and Equipment, China
[3]University of Leeds, UK

Abstract

Automatic container handling plays an important role in improving the efficiency of the container terminal, promoting the globalization of container trade, and ensuring worker safety. Utilizing vision-based methods to assist container handling has recently drawn attention. However, most existing keyhole detection/localization methods still suffer from coarse keyhole boundaries. To solve this problem, we propose a real-time container hole localization algorithm based on a modified salient object segmentation network. Note that there exists no public container dataset for researchers to fairly compare their approaches, which has hindered the advances of related algorithms in this domain. Therefore, we propose the first large-scale container dataset in this work, containing 1700 container images and 4810 container hole images, for benchmarking container hole location and detection. Through extensive quantitative evaluation and computational complexity analysis, we show our method can simultaneously achieve superior results on precision and real-time performance. Especially, the detection and location precision is 100% and 99.3%, surpassing the state-of-the-art-work by 2% and 62% respectively. Further, our proposed method only consumes 70 ms (on GPU) or 1.27s (on CPU) per image. We hope the baseline approach, the first released dataset will help benchmark future work and follow-up research on automatic container handling. The dataset is available at https://github.com/qkicen/A-large-scale-container-dataset-and-a-baseline-method-for-container-hole-localization.

1. Introduction

In an era featured by the globalization of production and consumption patterns, the demand for container trade volumes has been increased [1]. In the last decade, the growth of container trade volumes has far exceeded that of global GDP (Gross Domestic Product), distinctly manifesting an explosive trend [2]. Container terminals stand at a compulsory step of container trade, its efficiency of container handling has a great impact on container trade volume. In container terminals, since crane cabins are elevated at a great height, it is difficult for the crane operator to alone complete container handling on the ground [3]. The ground workers, therefore, need to assist the operator with frequent observation and communication, which immensely decreases the efficiency of container handling as well as raises the labor costs. As is shown in a port performance report [4], the global average productivity of one crane is 26.7 containers per hour, which represents the utilization of just 50%. Therefore, designing a real-time automatic container localization method becomes an urgent need since it can greatly reduce the handling time and thus increase the crane utilization rate. More importantly, long-time manual work may cause work accidents. The investigation in [5] shows from 2015 to 2020, accidents caused by human factors account for more than half of all accidents at container terminals. Based on these reasons, there is a vital need for an efficient and safe automation method to replace manual operation.

Convolutional Neural Networks (CNN), as one of the most popular deep learning architectures, have been proven effective in many computer vision applications [6]. Automatic container hole detection using computer vision and CNNs has also recently drawn attention [7–10]. Benefiting from advances in object detection [11], detection-based methods [9, 10] can detect the bounding box of the corner casting or keyholes with high precision, but it is actually hard to apply such methods in the container terminal. This is because the container can only be grabbed when the lock on the spreader falls accurately into the lock hole of the container. In this process, the position of keyhole center needs to be obtained. Accordingly, a pixel-level keyhole localization method is necessary for automatic

* Yunfeng Diao and Xin Tang contributed equally to this paper.
† Corresponding author: Wenming Cheng (wmcheng@home.swjtu.edu.cn) and Yunfeng Diao (dyf@my.swjtu.edu.cn)

(a) (b)

Figure 1. (a) is a real container keyhole and (b) is a miniature container keyhole.

container handling. Existing hole localization methods [7, 8] try to obtain the edge of the holes through traditional edge detection algorithms such as Canny Detection and Hough Transformation [12]. However, real containers are often corroded under harsh environments, making the edge detection algorithm easily interfered by the rust around the keyhole(Fig. 1).

Another limiting factor is that there is no public container dataset for container detection/localization. Research-ers have to waste a lot of time labeling their private datasets. In addition, the absence of a public dataset has also made it difficult for researchers to fairly compare their approaches in the same benchmark. Hence, we in this work hope to propose the first public large-scale container dataset to facilitate related studies on container hole detection/localization.

Formally, we propose a pixel-level keyhole segmentation approach with real-time performance in the container terminals and hope it will serve as a benchmark for future methods in the community. The method contains two steps. In the first step, object detection model is designed to detect the rough keyhole region from the container image. In the second step, we propose a new Light-EGNet model to segment the exact keyhole region, and then the keyhole center can be accurately positioned by calculating the centroid of its area.

To evaluate our proposed method and advance related research, we release the first public large-scale container dataset. In addition, we propose a new evaluation metric which can convert the pixel deviation in the digital image into the physical deviation from the hole center. Experimental results show that our proposed method can simultaneously achieve superior results on precision and real-time performance. Through comprehensive comparisons with state-of-the-art works, our proposed method shows the superiority by big margins and thus can be used as a baseline for benchmarking future methods. We hope the baseline approach, new released dataset and evaluation metric will be helpful for benchmarking future work and furthering related research on automatic container handling. The contributions of this paper are summarized as follows:

1. Release the first large-scale publicly available container dataset, containing 1700 real container images and 4810 corner hole images.

2. Propose a new container hole segmentation method based on a new Light-EGNet model, and verify its precision and real-time performance through exhaustive quality and computational complexity evaluation.

3. Propose a strict practical metric for evaluating the practical hole center deviation.

2. Related work

2.1. Object detection based on deep learning

Deep learning-based object detection not only focuses on the position where the object exists, but also recognizes the object class [11] . Its methods can be divided into two classes, depending on whether to exist region proposal. Two-stage detection methods, e.g. [13–15], first use region proposal to generate potential bounding boxes in an image, and then run a classifier on the region proposal to find a probable object. Finally, post-processing is used to filter the non-objected bounding boxes and duplicated objects from all probable objects. One-stage detection predicts the position and class of detected objects by the end-to-end way. Typical one-stage methods include YOLO-based algorithm [16, 17], FCOS algorithm [18], SSD algorithm [19], etc. Compared with two-stage object detection, one-stage methods have higher real-time performance while compromising the detection precision slightly. However, regardless of the one-stage or two-stage object detection model, the model's output is a bounding box containing the object but cannot represent the object contour. Accordingly, simply adaption of detection techniques to container detection will suffer from coarse container keyhole localization [10].

2.2. Salient object segmentation

Salient object segmentation can capture the distinctive region that attracts human attention from the complex scene. In the perspective presented in [20], most salient object segmentation methods can directly discover the most salient object, while a few salient object detection methods include two stages: detecting the salient object region and segmenting the most salient object from that region. It is worth noting that the salient object region can include one or more objects. By means of pixel-wise segmentation, the accurate contour of the most salient object can be obtained. In particular, relying on deep network structures and deep supervised learning at different convolutional hierarchies,

salient object detection methods [21–24] perform remarkably on the evaluation benchmark of precision, such as Precision-recall and F-measure [20]. In this paper, to satisfy the real-time performance in the specific container handling domain, we propose a modified edge guidance network [24] named Light-EGNet and apply it to container keyhole localization.

2.3. Vision-based Container Detection

Early research has well explored a variety of handcrafted features [25, 26] to detect the container. Both Wei et al. [25] and Yoon et al. [26] use a stereo camera to measure the depth information of the container and then detect the container edges through Hough line transform [27]. With the in-depth study of container detection, later researchers have tended to investigate how to detect tiny but crucial features, e.g., container keyholes and corner casting. [7] first utilizes color segmentation to extract the container region from the background. Further, the coarse corner casting boundaries are defined via calculation based on the international standard of the shipping container size. Lastly, the keyhole can be located by using the General Hough Transform fitting algorithm onto ellipses [27]. However, in the real terminals, such a method is sensitive to lighting, view angles, etc. To enhance robustness, Mi et al. [28] propose a sliding window algorithm for the right corner casting detection, by first extracting the features of casting in the right corner through HOG [29] and then using SVM [30] to recognize the right corner casting. Finally, the position of left corner casting can be obtained by mirror algorithm. Although the mirroring algorithm can improve the real-time performance, the process of sliding window needs to traverse the whole image window by window, still consuming a lot of time. In addition, the size of the sliding window is a fixed default, lacking adaptations to changes in view angles. To tackle this issue, Diao et al. [8] design a new adaptive local sliding window algorithm to detect the corner casting.

Benefiting from the ability of deep neural networks, Lee et al. [9] apply the LSTM model [31] to improve the detection efficiency. [10] also gets a similar real-time performance through using YOLO algorithm [16]. All existing deep learning-based methods focus on detecting corner casting. However, only outputting the coarse corner boundaries is not far enough to guide the crane to automatically move the container. To this end, a pixel-wise keyhole localization is essentially needed. The existing keyhole localization methods [7, 8] rely on Canny edge detection [32] and ellipses fitting method [33]. However, such approaches can not guarantee location precision as handcrafted features are sensitive to interference in actual working conditions.

Container dataset. In [8], Diao et al. collect the images of the container miniatures as the dataset. In [9, 10, 28], corner casting datasets are captured on a practical container terminal. To our best knowledge, all previous corner castings datasets do not be released. In addition, there is currently no dataset labeling the keyhole contours. Thus no comparisons can be conducted. More importantly, the accuracy, generalization and robustness of a deep neural network cannot be boosted due to a lack of sufficient large-scale datasets. We, therefore, release the first public large-scale container dataset.

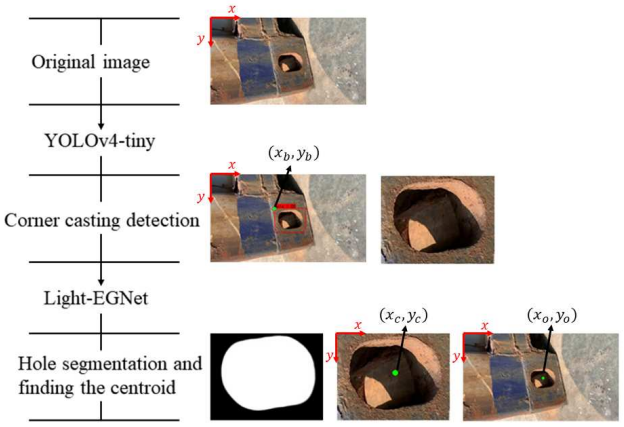

Figure 2. Overview of container hole localization.

3. Methodology

Inspired by [24], we propose a light edge guidance network (Light-EGNet) for pixel-wise keyhole localization. Given that segmenting the hole from the original inputs will cause a huge computation consumption, we thus first extract the bounding box of a keyhole using YOLOv4-tiny [17], and then utilize edge guidance network to segment the hole edge from the extracted coarse hole region. In the second step, we note that directly adapting the original edge guidance network (EGNet) [24] causes parameter redundancy and heavy memory footprint, while directly pruning the redundant layer compromises the precision (see Sec.5.2). To eliminate the trade-off between precision and speed, we propose a new light edge guidance network, named Light-EGNet. Light-EGNet reduces the original model parameters while still retaining the original keyhole semantic features. The overview of the algorithm is shown in Fig.2, in which the global coordinate of the keyhole center (x_o, y_o) is calculated by Eq.1.

$$(x_o, y_o) = (x_b, y_b) + (x_c, y_c) \tag{1}$$

3.1. Container keyhole detection

Container hole detection aims to extract the bounding box of the container hole from the original inputs. Given

that the YOLOV4-tiny object detection model [17], which has achieved state-of-the-art performance in many real-time applications [34, 35], is applicable to small networks while maintaining optimal speed and accuracy, we adopt it for keyhole detection. In order to improve the detection speed of YOLOV4-tiny model, we resize the original image resolution 960×540 to 416×416 as the model input. We follow [17] to employ CSPDarkNet53-tiny [17] as the backbone to extract the latent feature of container hole since CSPDarkNet53-tiny matches almost all optimal architecture features obtained by network architecture search technique [36]. To have a good adaption of YOLOV4-tiny to container hole detection, the semantic information at different depths of the convolutional layer of the backbone is used to predict container hole with different sizes. Then, YOLO heads output the confidence score and position of container hole. Finally, unreliable and duplicated container hole can be eliminated by the threshold value of confidence score and non-maximum suppression.

3.2. Container keyhole segmentation

In the second step, we aim to locate the hole center though segmenting the hole region. EGNet [24] integrates salient object features and edge features, thus can make the predicted edge of the keyhole more accurate. However, the original EGNet model is time-consuming for the specific domain due to lacking the adaption to hole segmentation. We, therefore, propose a modified EGNet model to solve this problem.

3.2.1 Edge guidance network (EGNet)

The pipeline of EGNet is shown in Fig.3. VGG16 is used as the backbone via truncating the last fully connected layers and adding another side path to the last pooling layer. Thus the six side features from the backbone network are represented as Block1-2, Block2-2, Block3-3, Block4-3, Block5-3 and Block6-3. Block N-M represents the N^{th} convolutional block in the backbone network containing M convolutional layers. The side path $S^{(1)}$ is thrown away and the other five side paths $S^{(2)}, S^{(3)}, S^{(4)}, S^{(5)}, S^{(6)}$ remained as Block1-2 is too close to the input. For simplicity, these five features could be denoted by a backbone features set C:

$$C = \left\{C^{(2)}, C^{(3)}, C^{(4)}, C^{(5)}, C^{(6)}\right\} \tag{2}$$

where $C^{(2)}$ denotes the Block2-2 features and so on.

Progressive salient object features extraction module (PSFEM). To obtain more robust features, three convolutional layers and followed ReLu layer after each convolutional layers are added on each side path (Conv in Fig.3). Besides, deep supervision is used on each side path. A convolutional layer is adapted to convert the feature maps to the single-channel prediction mask and is denoted as D in Tab. 1. The details of the convolutional layers are shown in Tab. 1.

Table 1. Details of each side output in EGNet. T denotes the 'Conv' block. Each T contains three convolutional layers: T_1, T_2, T_3 and three followed ReLu layers. We show the kernel size, padding and channel number of each convolutional layer. D denotes the transition layer which converts the multi-channel feature map to one-channel activation map. S denotes the side path.

S	T1			T2			T3			D		
2	3	1	128	3	1	128	3	1	128	3	1	1
3	3	1	256	3	1	256	3	1	256	3	1	1
4	5	2	512	5	2	512	5	2	512	3	1	1
5	5	2	512	5	2	512	5	2	512	3	1	1
6	7	3	512	7	3	512	7	3	512	3	1	1

Non-local salient edge features extraction module(NLSEM). Given that Block2-2 can preserve better edge information [37], the local edge information is directly extracted from Block2-2. To restrain the non-salient edge, the top-level location information is propagated to the side path $S^{(2)}$ via a top-down location propagation. The fused features $\overline{C}^{(2)}$ can be obtained via:

$$\overline{C}^{(2)} = C^{(2)} + Up(\phi(Trans(\hat{F}^{(6)}; \theta)); C^{(2)}) \tag{3}$$

where $Trans(*; \theta)$ is a convolutional layer with parameter θ to change channel numbers and $\phi()$ is a ReLU activation function. $Up(*; C^{(2)})$ is bilinear interpolation operation to up-sample * to the same size as $C^{(2)}$. $\hat{F}^{(6)} = f(C^{(6)}; W_T^{(6)})$ means the enhanced features in side path $S^{(6)}$, where $W_T^{(i)}$ denotes the parameters in $T^{(i)}$. For simplicity, we denote $UpT(\hat{F}^{(i+1)}; \theta, C^{(i)}) = Up(\phi(Trans(\hat{F}^{(i+1)}; \theta)); C^{(i)})$. The enhanced features in $S^{(3)}, S^{(4)}, S^{(5)}$ can be computed via:

$$\hat{F}^{(i)} = f(C^{(i)} + UpT(\hat{F}^{(i+1)}; \theta, C^{(i)}); W_T^{(i)}) \tag{4}$$

where $f(*; W_T^{(i)})$ denotes a series of convolutional and non-linear operations with parameters $W_T^{(i)}$. Similarly, the final salient edge features F_E in $S^{(2)}$ can be computed as $f(\overline{C}^{(2)}; W_T^{(2)})$. The loss for modeling the salient edge feature can be defined as:

$$\begin{aligned} L^{(2)}(F_E; W_D^{(2)}) = &- \sum_{j \in Z_+} \log Pr(y_j = 1 | F_E; W_D^{(2)}) \\ &- \sum_{j \in Z_-} \log Pr(y_j = 0 | F_E; W_D^{(2)}) \end{aligned} \tag{5}$$

where Z_+ and Z_- are the salient edge pixels set and the non-salient edge pixels set, respectively. W_D denotes

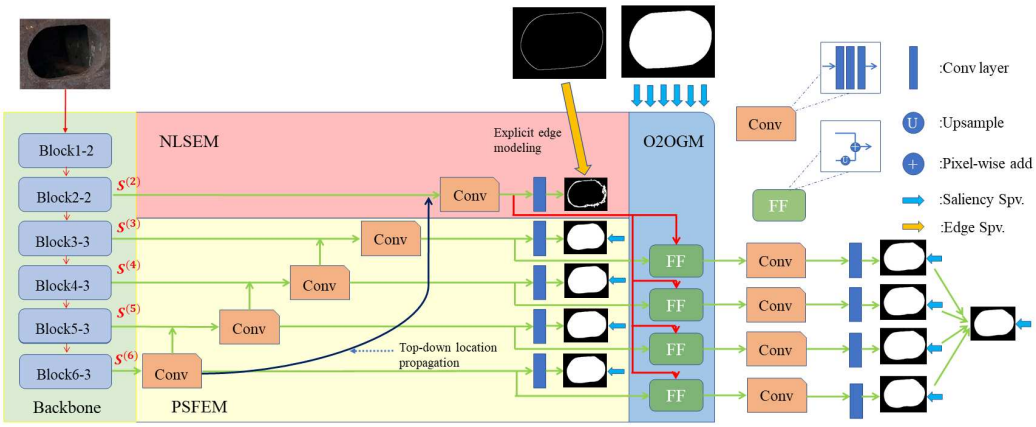

Figure 3. The pipeline of EGNet [24]. The thick red lines is represented information flows between the scales. PSFEM: progressive salient object features extraction module. NLSEM: non-local salient edge features extraction module. O2OGM: one-to-one guidance module. FF: feature fusion; Spv: supervision. $S^{(i)}$ represents the Nth side path and Block N-M represents the N^{th} convolutional block in the backbone network containing M convolutional layers.

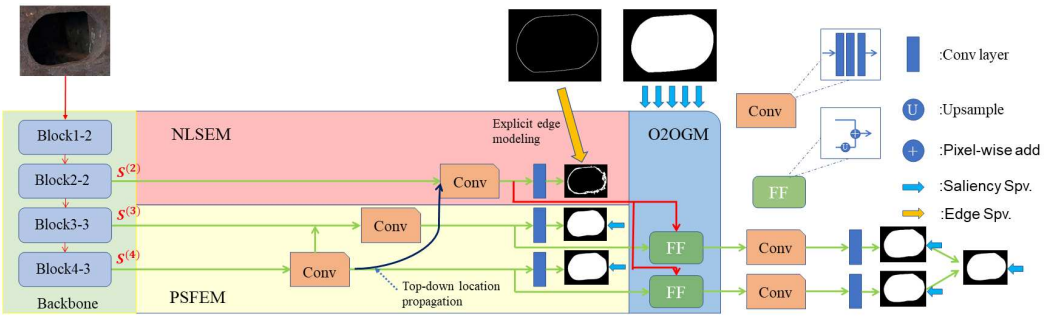

Figure 4. The pipeline of Silly-EGNet model.

the parameters of the transition layer as shown in Tab. 1. $Pr(y_j = 1|F_E; W_D^{(2)})$ is the prediction map in which each value denotes the salient edge confidence for the pixel. Similarly, the supervision of salient object detection can be defined as:

$$L^{(i)}(\hat{F}^{(i)}; W_D^{(i)}) = -\sum_{j \in Y_+} \log Pr(y_j = 1|\hat{F}^{(i)}; W_D^{(i)}) - \sum_{j \in Y_-} \log Pr(y_j = 0|\hat{F}^{(i)}; W_D^{(i)}), i \in [3, 6] \tag{6}$$

where Y_+ and Y_- denote the salient and non-salient region pixels set. Finally, the total loss of features extraction can be written as:

$$\mathbb{L} = L^{(2)}(F_E; W_D^{(2)}) + \sum_{i=3}^{6} L^{(i)}(\hat{F}^{(i)}; W_D^{(i)}) \tag{7}$$

One-to-one guidance module (O2OGM). O2OGM aims to utilize the salient edge features to guide the salient object features to have superiority on both segmentation and localization. To this end, the salient edge features are fused into enhanced salient object features in each sub-side path to get the salient edge guidance features(s-features):

$$G^{(i)} = UpT(\hat{F}^{(i)}; \theta; F_E) + F_E, i \in [3, 6] \tag{8}$$

Then the enhanced s-features $\hat{G}^{(i)}$ can be calculated by Eq.4. The loss of each sub-side output prediction map can be calculated as:

$$L^{(i)}(\hat{G}^{(i)}; W_{D'}^{(i)}) = -\sum_{j \in Y_+} \log Pr(y_j = 1|\hat{G}^{(i)}; W_{D'}^{(i)}) - \sum_{j \in Y_-} \log Pr(y_j = 0|\hat{G}^{(i)}; W_{D'}^{(i)}), i \in [3, 6] \tag{9}$$

where $W_{D'}$ denotes the model parameters in O2OGM. Then, the multi-scale prediction graph is fused to obtain a fusion graph, and the loss of this step is:

$$L_f^{(i)}(\hat{G}; W_{D'}) = \sigma(Y, \sum_{i=3}^{6} \beta_i f(\hat{G}^{(i)}; W_{D'}^{(i)})) \tag{10}$$

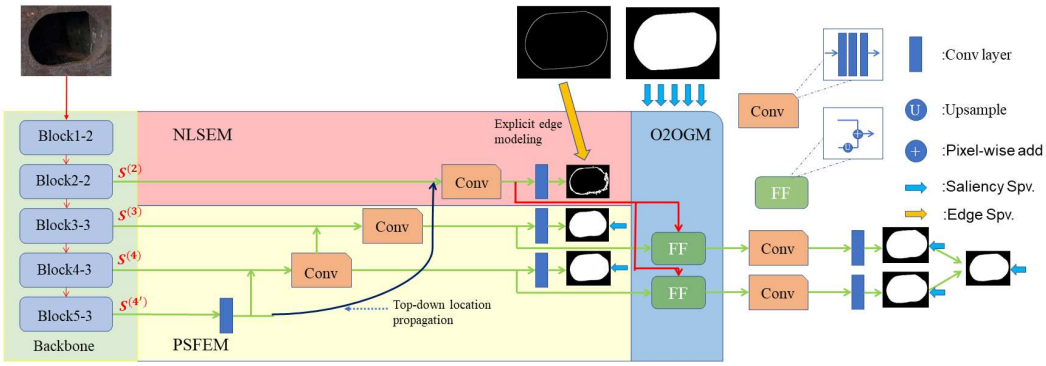

Figure 5. The pipeline of Light-EGNet.

where the $\sigma(*,*)$ denotes the cross-entropy loss between prediction map and saliency ground-truth. Thus the loss for O2OGM ($\mathbb{L}'$) and the total loss for EGNet (($\mathbb{L}_t$)) can be written as:

$$\mathbb{L}' = L_f^{(i)}(\hat{G}; W_{D'}) + \sum_{i=3}^{6} L^{(i)}(\hat{G}^{(i)}; W_{D'}^{(i)}) \tag{11}$$

$$\mathbb{L}_t = \mathbb{L} + \mathbb{L}' \tag{12}$$

3.2.2 The modifications of EGNet

Silly-EGNet. In the practical application of container handling, keyhole localization has high real-time constraints. However, it is hard for original EGNet to conduct real-time hole segmentation, because EGNet has a deep backbone network and five side paths, whose learned redundant features cause huge computations consumption. Especially, as the last two of the five side paths account for 60% of the total parameters of EGNet, it is not essential to learn such deep features for keyhole segmentation. Given that the hole features are not intricate and container hole images are small as input data, one natural choice for modifying EGNet is directly pruning the backbone block and side paths with redundant features. Specially, we prune the last two side paths($S^{(5)}$ and $S^{(6)}$) and the corresponding convolutional blocks (Blocks5-3 and Block6-3) in the backbone network, and thus replace $S^{(6)}$ with $S^{(4)}$ for top-down location propagation in NLSEM. The pruned architecture is shown in Fig.4, named Silly-EGNet. Correspondingly, the backbone features set C (Eq. 2) becomes:

$$C = \left\{C^{(2)}, C^{(3)}, C^{(4)}\right\} \tag{13}$$

The fused features $\overline{C}^{(2)}$ (Eq. 3) become:

$$\overline{C}^{(2)} = C^{(2)} + Up(\phi(Trans(\hat{F}^{(4)}; \theta)); C^{(2)}) \tag{14}$$

Light-EGNet. Although Silly-EGNet can avoid heavy memory footprint, it severely sacrifices location precision. We suspect this is because Block4-3 is relatively near to Block2-2, and thus the side path $S^{(4)}$ cannot propagate the top-level location information to the side path $S^{(2)}$ in NLSEM. To tackle this problem, we use a subpath $S^{(4')}$ instead of $S^{(4)}$ for the operation of top-down location propagation. The new modified EGNet is called Light-EGNet and its overview architecture is shown in Fig. 5. The Block5-3 is added to provide to provide high-level hole semantic information for subpath $S^{(4')}$. Different from EGNet and Silly-EGNet adding three convolutional layers and three followed ReLu layers(Conv in Fig. 3, 4) on each side path, only one convolutional layer and one followed ReLu layer is added on subpath $S^{(4')}$. Considering that keyhole edge feature information is simple and not rich, choosing one convolutional layer is sufficient to fuse its deep semantic features in comparing with 'Conv' containing three convolution layers. Also, less convolution layer can reduce the floating-point operations consuming to reduce its processing time. Similar to Tab. 1, the details of the convolutional layers in Light-EGNet can be found in Tab. 2. Thus its fused feature $\overline{C}^{(2)}$ is denoted as:

$$\overline{C}^{(2)} = C^{(2)} + Up(\phi(Trans(\hat{F}^{(4')}; \theta)); C^{(2)}) \tag{15}$$

where the enhanced features in subpath $S^{(4')}$ can be represented as $\hat{F}^{(4')} = f(C^{(5)}; W_T^{(4')})$. Note that the subpath $S^{(4')}$ is only activated when obtaining $\overline{C}^{(2)}$ via the operation of top-down location propagation in NLSEM. Thus the total loss can be denoted as:

$$\begin{aligned}
\mathbb{L} &= L^{(2)}(F_E; W_D^{(2)}) + \sum_{i=3}^{4} L^{(i)}(\hat{F}_i; W_D^{(i)}) \\
\mathbb{L}' &= \sigma(Y, \sum_{i=3}^{4} \beta_i f(\hat{G}^{(i)}; W_{D'}^{(i)}) + \sum_{i=3}^{4} L^{(i)}(\hat{G}^{(i)}; W_{D'}^{(i)}) \\
\mathbb{L}_t &= \mathbb{L} + \mathbb{L}'
\end{aligned} \tag{16}$$

Table 2. Details of each side output in Light-EGNet. T denotes the 'Conv' block. Each T contains three convolutional layers: T_1, T_2, T_3 and three followed ReLu layers. We show the kernel size, padding and channel number of each convolutional layer. D denotes the transition layer which converts the multi-channel feature map to one-channel activation map. S denotes the side path. Note that Silly-EGNet only have side $S^{(2)} - S^{(4)}$, not including $S^{(4')}$

S	T1			T2			T3			D		
2	3	1	128	3	1	128	3	1	128	3	1	1
3	3	1	256	3	1	256	3	1	256	3	1	1
4	5	2	512	5	2	512	5	2	512	3	1	1
$4'$	5	2	512	n/a	n/a	n/a	n/a	n/a	n/a	3	1	1

4. Container dataset and benchmark metrics

4.1. Container dataset

We release two datasets: *Container* dataset and *Container Miniature* dataset. The former is captured from the real container yard, for holes detection/location. While the latter is captured from the simulation laboratory, for container/container holes detection. For more details about Container Miniature dataset please refer to our preliminary work [8]. The *Container* dataset consists of 1700 container images marked with container holes and 4810 container hole images marked with holes region, which both are captured from the container station. Some examples of the *Container* dataset can be found in Fig.6. The container images are randomly selected from the video sequences. To increase the diversity of the container dataset, we change the view angels from time to time, and smoothly move the camera along in either a horizontal or vertical direction while keeping the container casting within the perception field. Note that we add some slightly blurred images into the dataset, which aims to simulate the real conditions, meanwhile, improve the generalization and robustness of the models. The collected images are diverse in view angles, lighting conditions, background and container texture. We also release 144 video streams to encourage future studies to draw attention to time-series data.

Keyhole detection. The collected containers have different colors, appearance, and views. All container images are annotated carefully. Specially, we use bounding boxes to mark container hole on the container images, and then generate the class label correspondingly, as shown in the first two column of the Fig.6. Note that the size of the bounding box should be slightly greater than that of container hole so that the entire container hole can be detected during the test.

Keyhole edge detection and hole center location. A lot of rust around the container holes(the third column in Fig.6 is not only a challenge to the robustness of the models, but also makes it difficult to mark the hole center directly. In addition, the location precision may have a big deviation from the ground truth if determining the position of the hole center only relies on eye observation [8]. To tackle this problem, third-order Bezier curves [38] defined in Eq.17 are utilized to match the hole edge, and the hole region(the last column in Fig.6) enclosed by curves can be filled. The process of hole edge matching is presented in Fig.7.

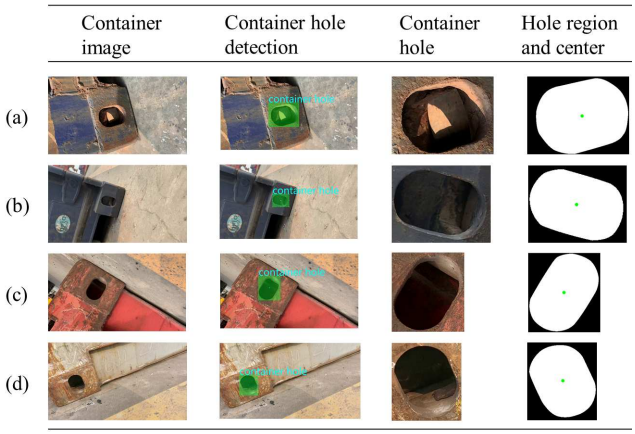

Figure 6. Examples(a-d) of Container dataset. The first column is the raw container images; the second column is the images marked the keyhole's bounding box; the third column is the raw keyhole images; the last column is the hole images marked the hole edge region.

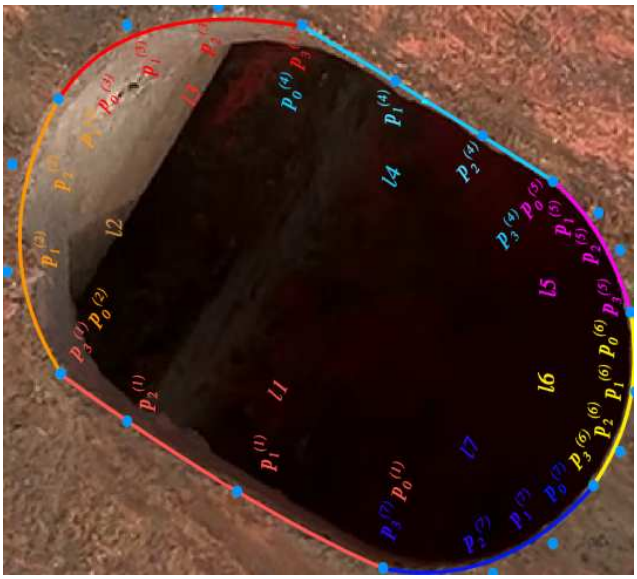

Figure 7. The process of labeling the hole edge utilizing Third-order Bezier curves.

$$B^{(i)}(t) = (1-t)^3 P_0^{(i)} + 3t(1-t)^2 P_1^{(i)} + 3t^2(1-t)P_2^{(i)} + t^3 P_3^{(i)}, t \in [0,1] \quad (17)$$

where $P_0^{(i)}$ and $P_1^{(i)}$ represent the start point and the endpoint of the curve i, respectively. $P_2^{(i)}$ and $P_3^{(i)}$ are the control point for the curvature of the curve i. $B^{(i)}(t)$ is the coordinate of the intermediate point of the curve. In Fig.6, the matched hole is described by a mass of white area (the fourth column), hole center (green point) can be calculated by hole region centroid.

4.2. Benchmark metrics

Metrics for keyhole detection. We follow [9, 10] to evaluate the container hole detection performance using precision and recall rate, which are described in Eq.18-Eq.19.

$$precision = \frac{N_{ms}}{N_s} \times 100\% \quad (18)$$

where N_{ms} and N_s represent the number of correctly detected container holes and the total number of the detected container holes respectively.

$$recall = \frac{N_{ms}}{N_m} \times 100\% \quad (19)$$

where N_m represents the total number of container holes in the dataset.

Metrics for keyhole localization. The existing keyhole localization metric calculate the pixel deviation on digital images [8]. But it is not applicable in the physical world and thus can not be the localization criteria in actual container terminals. To solve this problem, we release a new adaptive metric which can convert the pixel deviation into the physical deviation of the hole center:

$$l = \sqrt{(\hat{x_o} - x_o)^2 + (\hat{y_o} - y_o)^2} \times \sqrt{\frac{S^{True}}{S^{pixel}}} \quad (20)$$

$$p^i = \begin{cases} 1, & l < \delta \; mm \\ 0, & else \end{cases} \quad (21)$$

$$location\ precision = \frac{\sum_{i=1}^{N_s} p^i}{N_s} \quad (22)$$

where S^{pixel} is the pixel region within the keyhole in the image, S^{True} is the practical region within the keyhole. Referring to [39], the practical area S^{True} is 3496.32mm. l is the Euclidean distance deviation, δ is a fixed threshold. If $l < \delta$, we define the keyhole center localization is corrected. According to the empirical results in container yards, the deviations within 1.5 mm will not affect the container handling, so we set $\delta = 1.5$. The averaged location precision can be calculated by Eq.22.

Table 3. Training settings for YOLOv4-tiny

Parameter	Value
Number of epochs	100
Optimizer	Adam(weight decay = 5e-4)
Learning rate adjustment strategy	StepLR(stepsize=1,gamma=0.95)
Batch size	16
Learning rate	e^{-3}, e^{-4}

Table 4. Training settings for EGNet-based networks

Parameter	Value
Number of epochs	50
Optimizer	Adam(weight decay = $5e^{-4}$)
Batch size	1
Learning rate	$5e^{-5}$

5. Experiments

5.1. Setup

All experiments are conducted on the PyTorch platform with one NVIDIA RTX 3090 GPU and Intel I7-10th CPU, and evaluated on the new released *Container* dataset. We follow a ratio of 80% and 20% to randomly split the dataset into a train and test set. For training YOLOv4-tiny, we follow the setting in [17]. We chose Adam optimizer and StepLR learning rate adjustment strategy. The backbone network of the model is loaded with pre-training weights. The number of epochs was set to 100. In the first 50 epoch, the parameters update of the backbone are frozen, the initial learning rate is e^{-3}. In the last 50 epochs, all the parameters were updated, the initial learning rate is e^{-4}. More details of YOLOv4-tiny training are shown in Tab.3. For training EGNet-based networks, we follow the setting in [24]. The gradients of the model parameters are updated every 10 epochs. Further information about models training is described in Tab.4.

5.2. Ablation study

The major variants of our model are EGNet-based networks with different components. Therefore, we first show an ablation study to evaluate the performance of EGNet, Silly-EGNet and Light-EGNet. The original EGNet [24] is used as a baseline method. Both Silly-EGNet and Light-EGNet are our proposed modifications for EGNet, whose architecture details can be found in Sec. 3.2.2. Results are shown in Tab. 5. The Silly-EGNet consumes the least time but the location precision is 3.7% less than EGNet, while as we will show in Sec.5.4, EGNet cannot directly be applied to the practical terminals due to its terrible real-time performance. In comparison with EGNet and Silly-EGNet, Light-EGNet eliminates the trade-off between precision and efficiency. Its average time is halved compared to

Table 5. Effectiveness of the different components on EGNet-based models. Precision means location precision. Times means the averaged testing time of one input for segmentation.

Location methods	Precision(%)	Times(s)
EGNet	99.5	0.038
Silly-EGNet	95.8	0.013
Light-EGNet	99.3	0.018

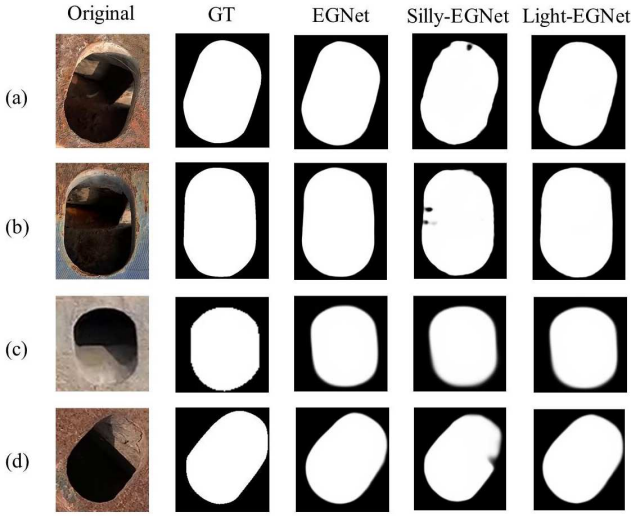

Figure 8. The results of EGNet-based salient object segmentation. GT presents the ground truth of hole region.

EGNet, while the precision still reaches 99.3%, which is only a 0.2% difference with EGNet. We suspect the reason for superior results is that although we prune the model, we still retain most of the high-level semantic feature information, which can store more valid features utilizing fewer network layers. The visual results in Fig.8 further prove our assumption. The segmentation results from Silly-EGNet often cannot retain complete keyhole edge, while the Light-EGNet retains the complete hole segmentation information like EGNet. We leave the computational analysis to Sec.5.4. Unless specified otherwise, we use Light-EGNet in our proposed method by default.

5.3. Evaluation results and comparisons

Since the container keyhole localization algorithm we proposed is a two-stage method containing hole detection and hole segmentation, we report and analysis its detection and segmentation performance respectively. For evaluating keyhole detection performance, we choose 4 recent the state-of-the-art methods [7, 9, 10, 28] for comparisons. [8] is excluded as their results on container miniature datasets are difficult to fairly compare with other results on actual container datasets. [9, 10] are deep learning-based methods and [7, 28] are methods based on hand-crafted features. The results are shown in Tab. 6. We do not list the processing time because the used devices are different and their codes are also not public. We observe that our method simultaneously has 100% detection precision and recall, which outperforms all the other methods on detection performance.

Table 6. Comparison with state-of-the-art methods. DP means detection Precision, DR means detection recall and LP means location precision.

Method	DP	DR	LP
HOG+SVM [28]	97.63%	n/a	n/a
Binarization,morphology [7]	96.4%	n/a	0.3%
RNN+LSTM [9]	98%	84%	n/a
Improved YOLO [10]	96%	83%	37.3%
Ours(YOLOv4-tiny+Light-EGNet)	**100%**	**100%**	**99.3%**

We next show the location performance. Unlike hole detection algorithm only used for assisting manual operation, any localization method that cannot survive under the strict practical location metric will not be applied to automatic container handling. The existing keyhole localization method [7] utilizes Canny edge and ellipse fitting to segment the keyhole edge. We use [7] as a baseline for comparison. Considering that bounding box-based detection methods is a common strategy in the container hole detection domain, we thus choose [10] as another baseline. Because the code in [7, 10] are not public, we reproduce their method and conduct on our newly released Container dataset for a fair comparison. In practice, we use YOLOv4-tiny to replace YOLO in [10] as they have similar structure and YOLOv4-tiny has better detection performance.

As is shown in Tab.6. Light-EGNet can achieve 99.3% location precision, while [7] and [10] only have 0.3% and 37.3% location precision respectively. We further conduct qualitative evaluation as shown in Fig. 9. The center predicted by [10] obviously deviates from the true hole center, and some predicted centers by [7] is even totally far away from the true center(row (d), third column in Fig.9). The huge performance difference between ours and [7] is easily understandable as such hand-craft feature detection is not robust to the rust around keyholes and is sensitive to lighting, view angles, etc, while bounding box-based detection methods [10] suffer from coarse obejct boundaries. Light-EGNet focuses on the complementarity between keyhole edge information and keyhole object information and thus naturally has better location precision.

5.4. Complexity and real-time evaluation

In order to make the model work on low computational devices, it is important to reduce model complexity while

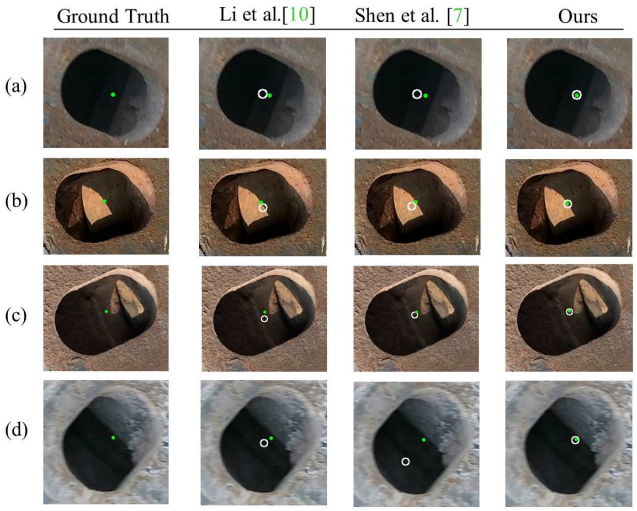

Figure 9. The visual comparisons with [7, 10]. The green point represents the true hole center and the white circle represents the predicted hole center.

Table 7. Complexity of EGNet-based models.

Method	Parameters(M)	GFLOPs	Model Size(Mb)
EGNet	108	96.7	278.5
Silly-EGNet	37.3	42.3	182.0
Light-EGNet	45.5	55.7	217.0

Table 8. Real-time performance of the EGNet-based models on various devices. Pre means location precision and Times means the averaged testing time of one input. xxx/xxx is the processing time pre/post AMP accelerating. Note that CPU I7-10th and GTX 1660Ti cannot use AMP due to lacking Tensor Core.

Reference	Unit type	Pre(%)	Times(s)	△T(%)
EGNet	RTX3090	99.5	0.038/0.021	N/A
	GTX1660Ti		0.12	
	CPU I7-10th		2.24	
Silly-EGNet	RTX3090	95.8	0.013/0.0075	65.7
	GTX1660Ti		0.05	58.3
	CPU I7-10th		0.45	79.9
Light-EGNet	RTX3090	99.3	0.018/0.0093	52.6
	GTX1660Ti		0.06	50.0
	CPU I7-10th		1.20	46.4

still retain a high precision. Therefore, we count the number of model parameters, floating-point operations (FLOPs) and model size to evaluate the model complexity. FLOPS represents the number of multiplication and addition operations in a forward propagation process, reflecting the model complexity. For convolutional kernels we compute FLOPs as:

$$FLOPs = 2HW(C_{in}K^2 + 1)C_{out} \tag{23}$$

where H,W and C_{in} are height, width, and number of channels of the input feature map, K is the kernel width (assumed to be symmetric), and C_{out} is the number of output channels. For fully connected layers we compute FLOPs as:

$$FLOPs = (2I - 1)O \tag{24}$$

where I is the input dimensionally and O is the output dimensionality. The number of model parameters and saved model size reflect the complexity of model space. The complexity evaluation are shown in Tab.7, and model details can be found in Tab. 1, 2. Silly-EGNet and Light-EGNet reduce the model complexity by about 50%. What's more, Light-EGNet utilizeS subpath $S^{(4')}$ instead of $S^{(6)}$ in top-down location propagation in NLSEM, which can simultaneously retain high-level location information(99.3% location precision) and reduce redundant features.

Table 9. Total time of the proposed two-step methods

Detection+Location	Unit type	Total times(s)
YOLOv4-tiny+EGNet	RTX3090	0.09
	GTX1660Ti	0.19
	CPU i7-10th	2.31
YOLOv4-tiny+Silly-EGNet	RTX3090	0.06
	GTX1660Ti	0.12
	CPU i7-10th	0.52
YOLOv4-tiny+Light-EGNet	RTX3090	0.07
	GTX1660Ti	0.13
	CPU i7-10th	1.27

Light-EGNet is expected to have a good real-time performance on RTX3090 GPU with no surprise. We further compare the processing time on GTX1660Ti GPU and Intel I7-10th CPU, two commonly used devices in practical applications. The experiment platform is Pytorch. In order to compare the processing time, time-saving($\triangle T$) is calculated by Eq.25:

$$\triangle T = \frac{T_{EG} - T_{com}}{T_{EG}} \tag{25}$$

where T_{EG} represents the location time of EGNet model and T_{com} is the location time of the specific method for comparison. We also adopt the automatic mixed precision (AMP) technique [40] to further speed up the inference process on GPU. Tab.8 shows the averaged testing time for keyhole location and time-saving on devices with RTX3090, GTX1660Ti GPU and i7-10th CPU. Fist, Silly-EGNet has the best real-time performance, but heavily compromise location precision. Unlike EGNet and Silly-EGNet, Light-EGNet eliminates the precision-speed trade-off. In addition, utilizing software optimization such as AMP can further reduce the processing time of the EGNet-based methods on GPU devices. In Tab.9, we show the total time including detection and segmentation. We find our proposed method can still easily meet the real-time requirement on any device.

6. Conclusion

To better further the future studies on automatic container handling, we release the first large-scale publicly available container dataset. Along with the dataset, a new metric for evaluating the practical hole center deviation is also released. Moreover, we propose a new baseline approach, whose detection precision and location precision reach 100% and 99.3% respectively in overall consuming 70 ms per image. Both the detection and location performance surpass the state-of-the-art works. Note that only our proposed method can survive under the strict practical metric, demonstrating the great potential of salient object segmentation-based methods. We expect the first released dataset, new metric and the new idea of locating container holes will shed a light on exploring automatic container handling.

Acknowledgements

We thank Puxin Container Terminal, Sichuan Province, China, for providing practical scenario support. This project has received funding from the Sichuan Science and Technology Program (No.2019YFG0300), and Open Research Project of Technology and Equipment of Rail Transit Operation and Maintenance Key Laboratory of Sichuan Province (No.2019YW001).

References

[1] Kevin PB Cullinane and Teng-Fei Wang. The efficiency of european container ports: a cross-sectional data envelopment analysis. *International Journal of Logistics: Research and Applications*, 9(1):19–31, 2006. 1

[2] Steve Saxon and Matt Stone. Container shipping: The next 50 years. *Travel, Transport & Logistics*, 2017. 1

[3] Tao Cheng and Jochen Teizer. Modeling tower crane operator visibility to minimize the risk of limited situational awareness. *Journal of Computing in Civil Engineering*, 28(3):04014004, 2014. 1

[4] Alex Lennane. Measuring port performance. *The Loadstar*, Nov 2015. 1

[5] Muhammad Arif Budiyanto and Haris Fernanda. Risk assessment of work accident in container terminals using the fault tree analysis method. *Journal of Marine Science and Engineering*, 8(6):466, 2020. 1

[6] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018. 1

[7] Yang Shen, Weijian Mi, and Zhiwei Zhang. A positioning lockholes of container corner castings method based on image recognition. *Polish Maritime Research*, 2017. 1, 2, 3, 9, 10

[8] Yunfeng Diao, Wenming Cheng, Run Du, Yaqing Wang, and Jun Zhang. Vision-based detection of container lock holes using a modified local sliding window method. *EURASIP Journal on Image and Video Processing*, 2019(1):1–8, 2019. 1, 2, 3, 7, 8, 9

[9] Jaecheul Lee. Deep learning–assisted real-time container corner casting recognition. *International Journal of Distributed Sensor Networks*, 15(1):1550147718824462, 2019. 1, 3, 8, 9

[10] Yan Li, Juanyan Fang, and Liandi Fang. Container keyhole positioning based on deep neural network. *International Journal of Wireless and Mobile Computing*, 18(1):40–50, 2020. 1, 2, 3, 8, 9, 10

[11] C Bhagya and A Shyna. An overview of deep learning based object detection techniques. In *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, pages 1–6. IEEE, 2019. 1, 2

[12] Diplaxmi R Waghule and Rohini S Ochawar. Overview on edge detection methods. In *2014 International conference on electronic systems, signal processing and computing technologies*, pages 151–155. IEEE, 2014. 2

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015. 2

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015. 2

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 2

[16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 2, 3

[17] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13029–13038, 2021. 2, 3, 4, 8

[18] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9627–9636, 2019. 2

[19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 2

[20] Ali Borji, Ming-Ming Cheng, Qibin Hou, Huaizu Jiang, and Jia Li. Salient object detection: A survey. *Computational visual media*, 5(2):117–150, 2019. 2, 3

[21] Wenlong Guan, Tiantian Wang, Jinqing Qi, Lihe Zhang, and Huchuan Lu. Edge-aware convolution neural network based salient object detection. *IEEE Signal Processing Letters*, 26(1):114–118, 2018. 3

[22] Jiangjiang Liu, Qibin Hou, Ming-Ming Cheng, Jiashi Feng, and Jianmin Jiang. A simple pooling-based design for real-time salient object detection. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 3917–3926. Computer Vision Foundation / IEEE, 2019. 3

[23] Xuebin Qin, Zichen Vincent Zhang, Chenyang Huang, Chao Gao, Masood Dehghan, and Martin Jägersand. Basnet: Boundary-aware salient object detection. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 7479–7489. Computer Vision Foundation / IEEE, 2019. 3

[24] Jiaxing Zhao, Jiangjiang Liu, Deng-Ping Fan, Yang Cao, Jufeng Yang, and Ming-Ming Cheng. Egnet: Edge guidance network for salient object detection. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 8778–8787. IEEE, 2019. 3, 4, 5, 8

[25] Li Wei and Eung-Joo Lee. Real-time container shape and range recognition for implementation of container auto-landing system. *Journal of Korea multimedia Society*, 12(6):794–803, 2009. 3

[26] Hee-Joo Yoon, Young-Chul Hwang, and Eui-Young Cha. Real-time container position estimation method using stereo vision for container auto-landing system. In *ICCAS 2010*, pages 872–876. IEEE, 2010. 3

[27] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972. 3

[28] Chao Mi, Zhi-Wei Zhang, You-Fang Huang, and Yang Shen. A fast automated vision system for container corner casting recognition. *Journal of Marine Science and Technology*, 24(1):54–60, 2016. 3, 9

[29] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005. 3

[30] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425, 2002. 3

[31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 3

[32] John F. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986. 3

[33] Walter Gander, Gene H Golub, and Rolf Strebel. Least-squares fitting of circles and ellipses. *BIT Numerical Mathematics*, 34(4):558–578, 1994. 3

[34] Tian Hui, YueLei Xu, and Rasol Jarhinbek. Detail texture detection based on yolov4-tiny combined with attention mechanism and bicubic interpolation. *IET Image Processing*, 2021. 4

[35] Huipeng Li, Changyong Li, Guibin Li, and Lixin Chen. A real-time table grape detection method based on improved yolov4-tiny network in complex background. *Biosystems Engineering*, 2021. 4

[36] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020. 4

[37] Pingping Zhang, Dong Wang, Huchuan Lu, Hongyu Wang, and Xiang Ruan. Amulet: Aggregating multi-level convolutional features for salient object detection. In *Proceedings of the IEEE International*

Conference on Computer Vision, pages 202–211, 2017. 4

[38] Gerald Farin. Algorithms for rational bézier curves. *Computer-aided design*, 15(2):73–77, 1983. 7

[39] S. Australia. Freight containers: part 3: corner fittings. *Standards*, 1993. 8

[40] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. In *International Conference on Learning Representations*, 2018. 10