CSAD: Unsupervised Component Segmentation for Logical Anomaly Detection (Supplementary Material)

Yu-Hsuan Hsieh
ss111062646@gapp.nthu.edu.tw
Shang-Hong Lai
lai@cs.nthu.edu.tw

National Tsing Hua University
Hsinchu, Taiwan

1 Semantic Pseudo-label Generation

1.1 Image Tag Generation

Table 3 shows the image tags of all categories generated by RAM++ [3] and the tags after manual filtering. We use the first image in the training set to generate the tags; for categories containing multiple types of products, we generate tags for each type and merge them together.

1.2 Component Mask Generation

In the component mask generation, we use the pretrained weight *sam_hq_vit_h.pth* from SAM-HQ [4], a variation of SAM trained on high-quality datasets, as our weight in SAM. We use the pretrained weight *grounding-dino-swinT-OGC.pth* and the same *sam_hq_vit_h.pth* as our weights in Grounded-SAM.

1.3 Mask Refinement

In the mask refinement process, we use two algorithms to obtain the refined mask from X^{sem} and X^{seg}, the filter-by-grounding algorithm and the filter-by-combine algorithm. The filter-by-combine algorithm is used to filter out overlapped masks, while filter-by-grounding algorithm is used to filter out noise masks. For the "screw bag" category, we apply both the filter-by-grounding algorithm and the filter-by-combine algorithm. As for the "juice bottle," we only apply the filter-by-combine algorithm. Detailed description of these two algorithms is given below.

Filter-by-grounding The filter-by-grounding algorithm filters out noise masks by removing masks in X^{seg} that are not highly overlapped with X^{sem}. The Python code using Numpy is shown in Figure 7

Filter-by-combine In the masks SAM generates, some overlapped masks need to be processed. For example, in the "screw bag" category, the SAM generates not only the masks of the head and body of the screw but also the mask of the whole screw. In this case, we aim to filter out the mask of the whole screw since we want precise component segmentation at this stage. The filter-by-combine algorithm checks if one mask can be the combination of some small masks. If so, this mask is dropped. After the process, we can remove the masks that are highly overlapped with other masks and maintain a fine-grained segmentation map. The Python code using Numpy is shown in Figure 8.

1.4 Component Feature Extraction

We use ImageNet pretrained WideResNet-50 as our feature extractor. The feature maps are obtained from the fourth layer of the feature extractor, and the rotation augmentation of each component is set to 60.

1.5 Resulting Semantic Pseudo-label Maps

Figure 6 shows the example images of X^{seg}, X^{sem} and the pseudo-label maps, as described in Sec.6.2. There are components missing in the pseudo-label maps in the category "breakfast box" and "juice bottle", and we fill up the holes for each component to mitigate this issue.

2 Component Segmentation

2.1 Training Process of Segmentation Network

Figure 1 shows the training process of the segmentation network. All images with semantic pseudo-label maps are augmented through LSA and the predictions of augmented images and augmented pseudo-label maps are used to calculate Cross Entropy loss, Dice loss, and Focal loss. Predictions of unlabeled images and randomly picked pseudo-label maps are used to calculate Histogram Matching loss, another Entropy loss that calculates the average entropy of each pixel's prediction is applied to reduce the uncertainty of the prediction.

2.2 LSA Augmentation

Figure 2 shows the proposed LSA augmentation procedure. To simulate logical anomalies, the position of the additional component is set at least $0.1 \times \textit{image\_size}$ far from the original position. Figure 3 shows the original images, the pseudo-label maps, the images augmented by LSA, and the augmented pseudo-label maps.

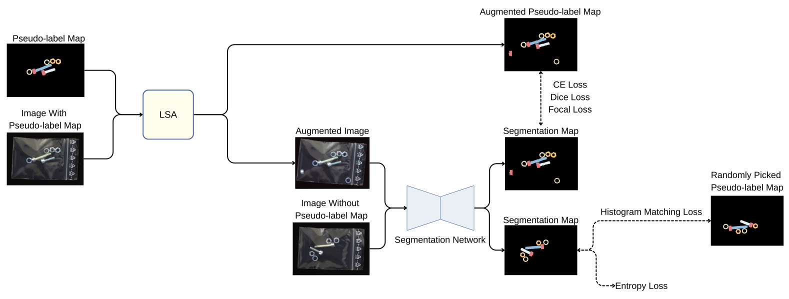

Figure 1: Diagram of the training process of the segmentation network.

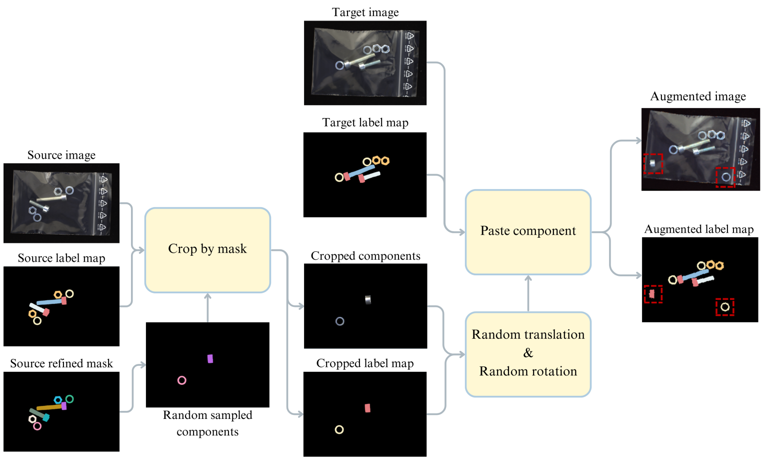

Figure 2: Diagram of the LSA augmentation process. The source image is randomly sampled from the images used in supervised training.

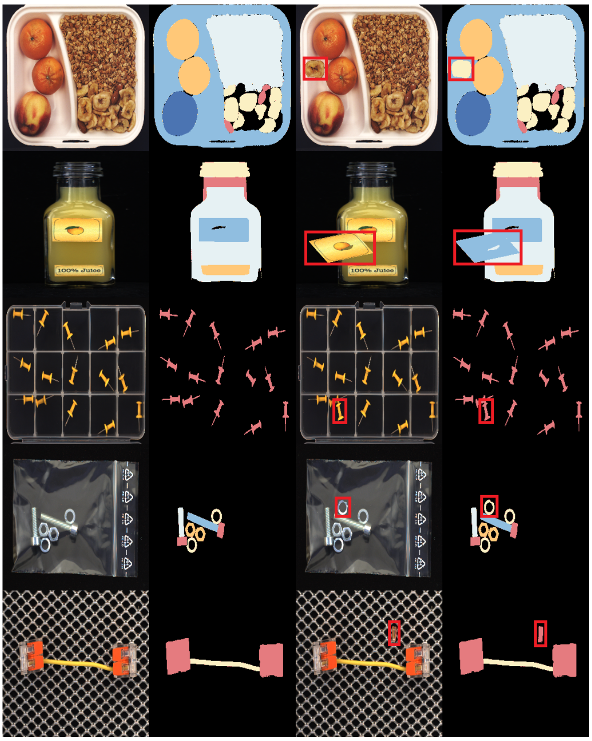

Figure 3: Example images of the LSA. From left to right presents the original image, pseudo-label map, LSA augmented image, and LSA augmented pseudo-label map. The red bounding box indicates the additional component added to the image.

3 Implementation Detail

The performance scores of the compared methods in all our experiments are retrieved from the original papers, except that the score of PatchCore [7] is from the paper of PSAD [5] and the score of SimpleNet [6] is from the paper of EfficientAD [2]. In the speed comparison section, we use the official implementation except for the EfficientAD, which does not have an official implementation. Instead, we use the implementation of Anomalib[1].

4 Anomaly Localization

To obtain anomaly localization results, we merge the anomaly maps from the LGST and Patch Histogram branches. The LGST anomaly maps are generated by averaging the difference maps of local and global student networks. For the Patch Histogram branch, we analyze the difference between the current histogram and the mean histogram of the training set. This analysis identifies missing or additional component classes. For additional components, we assign the class region the anomaly score based on the difference of the histogram. For missing components, we search for a normal image in the training set with the closest class histogram and assign the anomaly score based on the difference to that class region of the normal image. The final patch histogram anomaly map is the sum of all anomaly maps of different patch sizes. The example images and the corresponding anomaly maps are shown in Figure 4.

Note that the latency of anomaly map generation is not included in the experiments since it does not affect the image level detection result.

4.1 Anomaly Localization Performance

The anomaly localization performance is shown in Table 1. While our method does not outperform EfficientAD due to the limitations outlined in Sec. 6.2 , our Patch Histogram branch improves anomaly localization performance in both logical and structural anomalies, with an average sPRO improvement of 1.0%.

Model	Breakfast Box	Juice Bottle	Pushpins	Screw Bag	Splicing Connectors	Average
GCAD	-	-	-	-	-	89.1
EfficientAD	-	-	-	-	-	92.5
LGST	83.2	95.2	87.8	81.1	94.4	88.3
CSAD	84.0	95.3	88.1	84.6	94.5	89.3
LGST	76.7 / 89.6	95.2 / 95.3	98.5 / 77.1	73.9 / 88.3	94.9 / 93.9	87.8 / 88.8
CSAD	78.1 / 89.9	91.7 / 98.8	99.0 / 77.2	82.1 / 87.1	94.8 / 94.1	89.1 / 89.4

Table 1: Anomaly localization performance of the MVTec LOCO AD dataset. The result is reported in sPRO. For the last two rows, the score is given by (logical sPRO / structural sPRO).

5 Supplementary Experiments

5.1 Performance and Speed of Different Branches in CSAD

Table 2 shows the anomaly detection performance and the speed of different branches in our method. The results indicate that while the LGST and Patch Histogram branches individually do not achieve optimal performance, their combination yields the highest scores for both logical and structural anomalies. Remarkably, the Patch Histogram branch enhances structural anomaly detection despite not being specifically designed for it. Furthermore, LGST and Patch Histogram exhibit latencies of 5.7 milliseconds and 5.6 milliseconds, respectively. The overall latency of CSAD is reduced to 8.9 milliseconds, with a 2.4-millisecond reduction attributed to the reuse of features extracted by the teacher.

Branch		**LA**	**SA**	**Mean**	**Latency(ms)**	**Throughput(fps)**
PatchHist	LGST					
✓		<u>91.4</u>	75.4	83.4	**5.6**	<u>603.5</u>
	✓	87.7	<u>93.1</u>	<u>90.4</u>	<u>5.7</u>	**1402.9**
✓	✓	**96.7**	**94.0**	**95.3**	8.9	321.8

Table 2: Anomaly detection performance and speed comparison of different branches in our method. LA and SA denote logical anomalies and structural anomalies, respectively. Anomaly detection performance is measured in AUROC.

5.2 Hyperparameter of Component Clustering

Figure 5 illustrates the result of component clustering under different hyperparameters. Three bandwidths were tested in MeanShift clustering. A larger bandwidth results in less similar components being clustered together, potentially causing coarse semantic segmentation. For instance, a bandwidth of 4.0 clusters both long and short screw bodies together, which should be separated. In our experiments, We select bandwidth=3 for "screw bag" and 3.5 for all other categories.

6 Limitation

6.1 Limitation of Patch Histogram

The limitation of our patch histogram lies in the selection of patch sizes. Categories with strict component position constraints, such as "splicing connectors," benefit from smaller patch sizes, while those without such constraints may be adversely affected like "screw bag." Therefore, selecting the appropriate patch size requires prior knowledge of the product and its specific application scenarios.

6.2 Limitation of Semantic Pseudo-label Generation

Our pseudo-label generation process generally produces precise label maps. However, it is limited by the performance of the foundational model used in our implementation. This limitation was evident when the model failed to accurately segment obscured almonds in the "breakfast box," causing these regions to be incorrectly classified as background. Another example is the fruit icon missing in the "juice bottle", which is filtered out in the component clustering stage since the number of each unique fruit icon did not exceed half of the training samples, as described in Section 3.1 of the paper. Example images are shown in Figure 6. Despite this limitation, our Patch Histogram module is still capable of identifying logical anomalies that arise from such segmentation errors. Future efforts could focus on improving segmentation in such challenging scenarios.

Category	RAM++ Output Tags	Filtered Tags
Breakfast Box	almond, apple, banana, cereal, container, fill, food, fruit, grain, granola, mixture, nut, oatmeal, oats, orange, plastic, raisin, seed, topping, tray	almond, apple, container, oatmeal, orange, banana
Juice Bottle	alcohol, apple juice, beverage, bottle, liquor, glass bottle, juice, lemonade, liquid, olive oil, orange juice, yellow, alcohol, banana, bottle, glass bottle, glass jar, jug, juice, liquid, milk, alcohol, beverage, bottle, cherry, condiment, liquor, glass bottle, honey, juice, liquid, maple syrup, sauce, syrup, tomato sauce	alcohol, apple juice, beverage, bottle, liquor, glass bottle, juice, lemonade, liquid, olive oil, orange juice, yellow banana, bottle, glass bottle, glass jar, jug, juice, liquid, milk beverage, bottle, cherry, condiment, liquor, glass bottle, honey, juice, liquid, maple syrup, sauce, syrup, tomato sauce.
Pushpins	box, case, container, fill, needle, pin, plastic, screw, tool, tray, yellow	pin, pushpin, drawing pin
Screw Bag	bag, bolt, container, nut, package, plastic, screw, tool, metal bolts, metal hex nuts, metal washers, metal screws, zip-lock bag, screw, ring	metal bolts, metal hex nuts, metal washers, metal screws, zip-lock bag, screw, ring, bag, bolt, container, nut, tool
Splicing Connectors	attach, cable, connect, connector, hook, electric outlet, plug, pole, socket, wire	cable, connector, hook, electric outlet, plug, pole, socket, wire

Table 3: Image tags of all categories.

References

[1] Samet Akcay, Dick Ameln, Ashwin Vaidya, Barath Lakshmanan, Nilesh Ahuja, and Utku Genc. Anomalib: A deep learning library for anomaly detection. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 1706–1710. IEEE, 2022.

[2] Kilian Batzner, Lars Heckler, and Rebecca König. Efficientad: Accurate visual anomaly detection at millisecond-level latencies. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 128–138, 2024.

[3] Xinyu Huang, Yi-Jie Huang, Youcai Zhang, Weiwei Tian, Rui Feng, Yuejie Zhang, Yanchun Xie, Yaqian Li, and Lei Zhang. Open-set image tagging with multi-grained text supervision. *arXiv e-prints*, pages arXiv–2310, 2023.

[4] Lei Ke, Mingqiao Ye, Martin Danelljan, Yifan Liu, Yu-Wing Tai, Chi-Keung Tang, and Fisher Yu. Segment anything in high quality. In *NeurIPS*, 2023.

[5] Soopil Kim, Sion An, Philip Chikontwe, Myeongkyun Kang, Ehsan Adeli, Kilian M Pohl, and Sang Hyun Park. Few shot part segmentation reveals compositional logic for industrial anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 8591–8599, 2024.

[6] Zhikang Liu, Yiming Zhou, Yuansheng Xu, and Zilei Wang. Simplenet: A simple network for image anomaly detection and localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20402–20411, 2023.

[7] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. Towards total recall in industrial anomaly detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14318–14328, 2022.

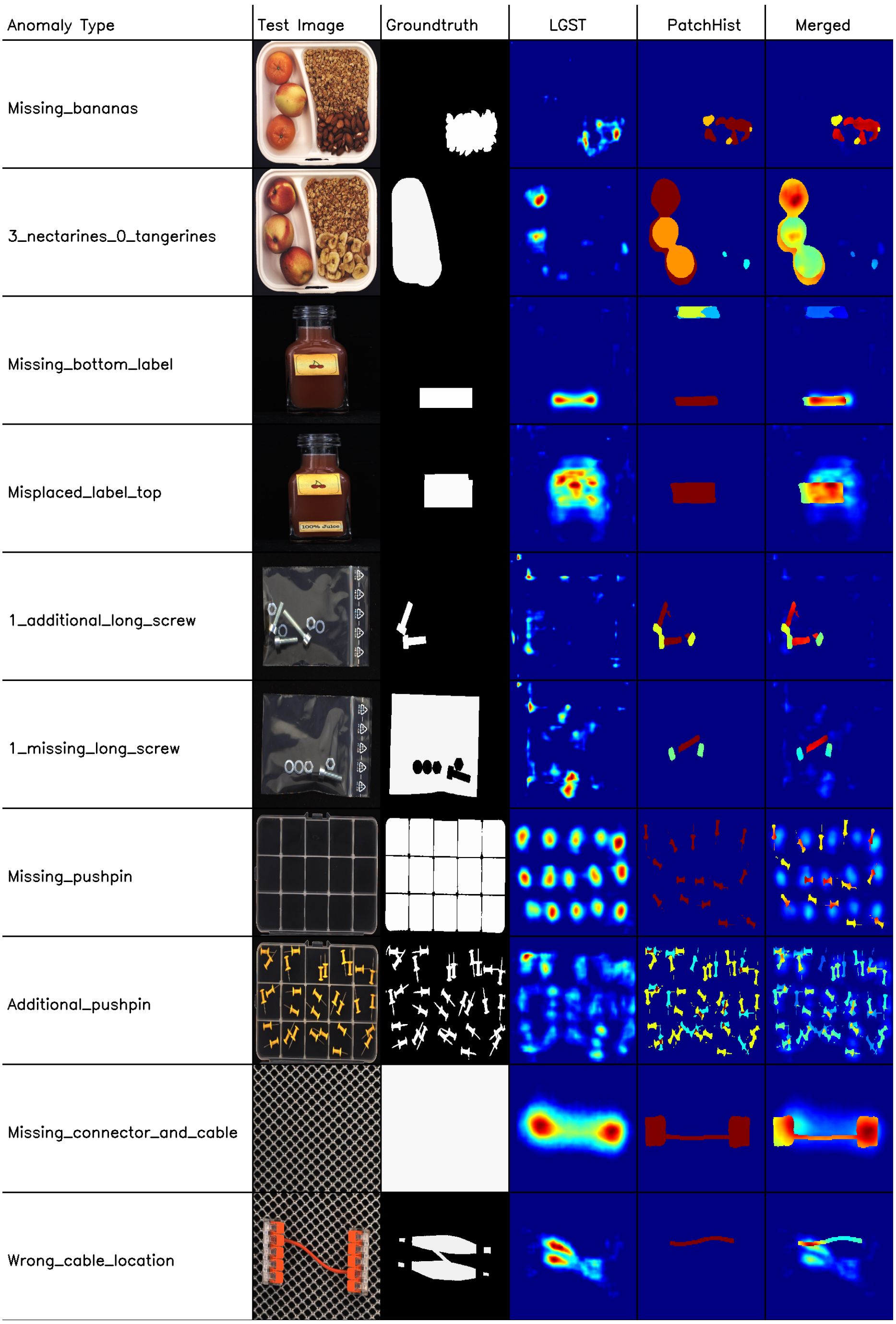

Figure 4: Example images and anomaly maps of anomaly localization result. For each category, we show two types of different logical anomalies.

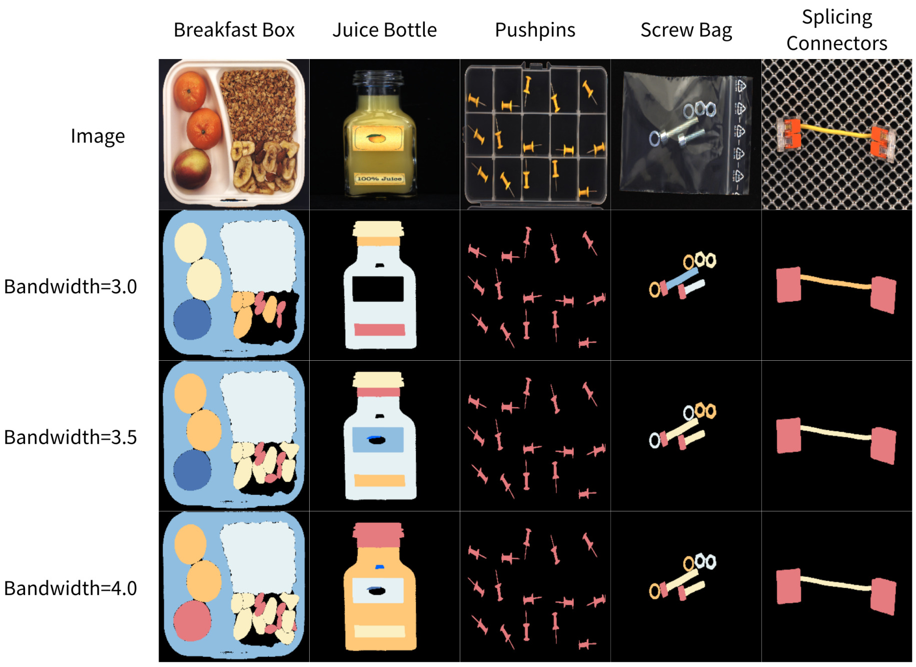

Figure 5: Result of component clustering under different hyperparameters. For each row, the same color indicates the same cluster of components.

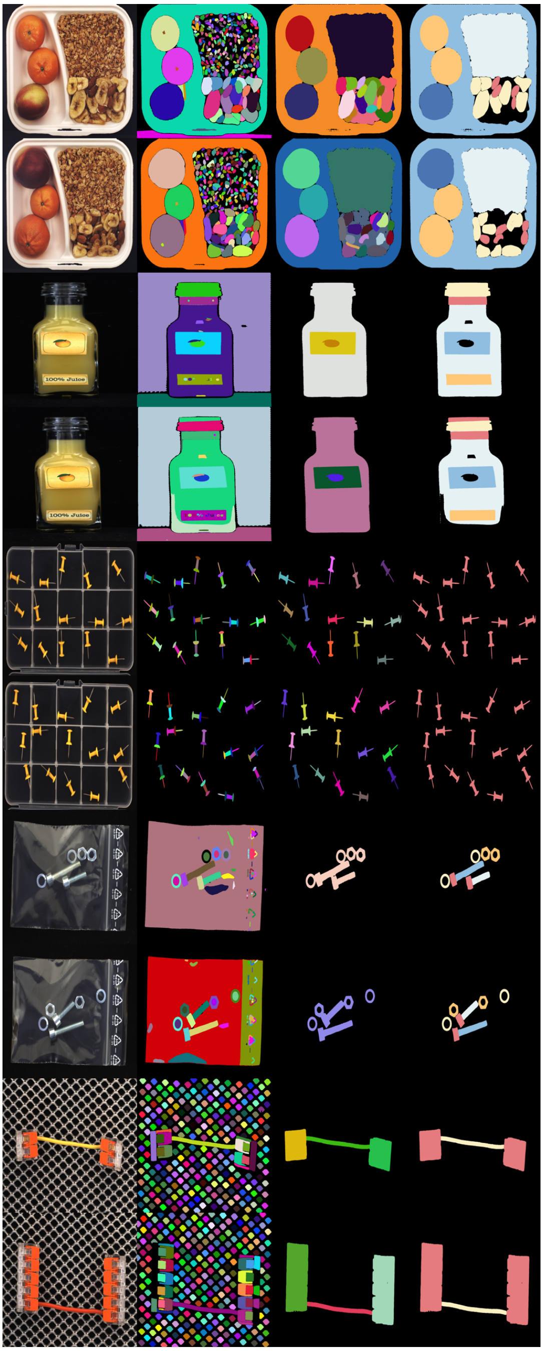

Figure 6: Example images of pseudo label generation of the five categories of the MVTecc LOCO, from left to right, represent normal image, X^{seg}, X^{sem}, and pseudo label image. In each category, random colors are assigned to different masks in X^{seg} and X^{sem} while the same color represents the same class in the pseudo-label image.

Python code of filter-by-grounding algorithm.

```python
import numpy as np
def filter_masks_by_grounding(grounding_mask,masks):
    """
        grounding_mask: binary mask
        masks: list of N binary masks
    """
    new_mask = list()
    for mask in masks:
        if np.sum(np.logical_and(grounding_mask,mask))/np.sum(mask!=0) > 0.9:
            new_mask.append(mask)
    return new_mask
```

Figure 7: Python code of the algorithm filter-by-grounding.

Python code of filter-by-combine algorithm.

```python
import numpy as np
def intersect_ratio(mask1,mask2):
    intersection = np.logical_and(mask1,mask2)
    if intersection.sum() == 0:
        return 0
    ratio = np.sum(intersection)/min([np.sum(mask1!=0),np.sum(mask2!=0)])
    ratio = 0 if np.isnan(ratio) else ratio
    return ratio

def filter_by_combine(masks):
    """
        masks: list of N binary masks
    """

    masks = sorted(masks,key=lambda x:np.sum(x)) # small to large
    combine_masks = np.zeros_like(masks[0])
    result_masks = list()
    wait_masks = list()
    for i,mask in enumerate(masks):
        if intersect_ratio(combine_masks,mask) < 0.9 or i == 0:
            combine_masks = np.logical_or(combine_masks,mask)
            result_masks.append(mask)
        else:
            wait_masks.append(mask)

    # second chance
    if len(wait_masks) != 0:
        for mask in wait_masks:
            ratio = np.sum(np.logical_and(combine_masks,mask))/np.sum(mask!=0)
            if ratio < 0.9:
                combine_masks = np.logical_or(combine_masks,mask)
                result_masks.append(mask)
    return result_masks
```

Figure 8: Python code of the algorithm filter-by-combine.