Security Checks X-Ray Image Prohibited Object Detection

Yuxiang Chai
New York University
yc3743@nyu.edu

Jierui Peng
New York University
jp4906@nyu.edu

Chutong Qiu
New York University
cq2039@nyu.edu

Abstract

Security inspection is very important in maintaining social security. Thus, in order to improve its performance, we design this research based on object detection. With the PIDray dataset that contains images of both secure and prohibited items, we use the principles of Faster RCNN, Sparse RCNN, and YoloV5 to build the models. Notably, we separately provide both pre-trained and non pretrained models for each principle. Finally, we evaluate and compare these models, finding that without considering the inference time, two-stage object detection models are a better choice. Code repository is here: https://github.com/YuxiangChai/CV_final_project

1. Introduction

Security inspection, as an important process to ensure the safety of public life and property, is widely used in public, especially in public transport stations and large event sites where massive numbers of people gather. But because current machine detection is not perfect, many occasions still arrange many staff to assist the machine or even completely replace the machine for manual inspection.

Manual inspection, however, is subject, and therefore very susceptible to external factors. Thus, the staff are likely to give unfair or inaccurate assessments. Therefore, it's still necessary and important to improve the performance of machine detection.

As a result, we propose this study about X-Ray detection of prohibited items. In this study, we utilize three object detection models: Faster RCNN, Sparse R-CNN, and YoloV5. With a security dataset called PIDray, we build, train, evaluate, and compare these models, expecting to contribute to improving the performance of security detection.

2. Related Works

2.1. Two-Stage Object Detector

Two-stage object detectors usually leverage a module that provides object proposals on a given image and use another module to classify and fintune on those proposals. Therefore the first stage is to find proposals for the image and the representing model is Region Proposal Network [22]. The second stage is to finetune the proposals. The representing model is Faster RCNN [22]. Also, there are some former researches such as R-CNN [6], Fast RCNN [5] and researches based on Faster RCNN such as Mask RCNN [7].

2.2. Single-Stage Object Detector

Single-stage object detectors are models that are developed end-to-end, without the RPN module, meaning that given the input image, the model predicts the bounding box and the class at the same time. Usually, those models use pre-defined anchors as proposals and make predictions on the anchors. Some representing models are SSD [16], Yolo serie [1, 10, 11, 19–21, 27], RetinaNet [13]. Single-stage detectors are usually faster and have a smaller size than two-stage detectors, but two-stage detectors usually have a better performance than single-stage detectors.

2.3. Other Object Detectors

In recent years, many new architectures have emerged. Anchor-free detectors, *e.g.* CenterNet [4] and Sparse RCNN [23], are trying to get rid of the constraints caused by predefined anchors or proposals found by RPN. And as Transformers [24] are used in the computer vision field, DETR [2], Swin Transformer [17] also achieved state-of-the-art performance.

2.4. Transfer Learning

Transfer learning aims at enhancing the model performance on the target domain by transferring the knowledge learned in a different and usually larger domain. It is widely applied because many tasks only provide a small domain and with the knowledge learned from the larger domain, the performance is better. [30] provides a comprehensive survey on transfer learning.

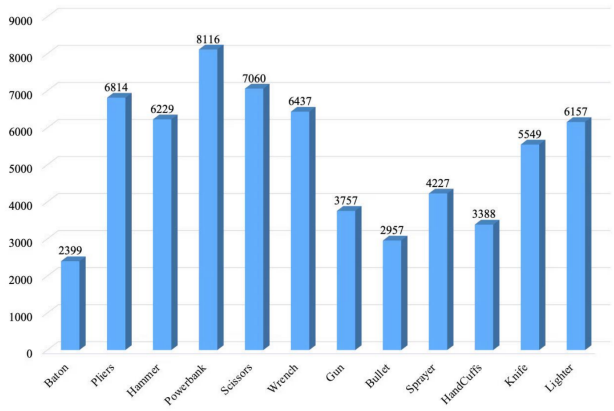


Figure 1. Distribution of objects. Image from [25]

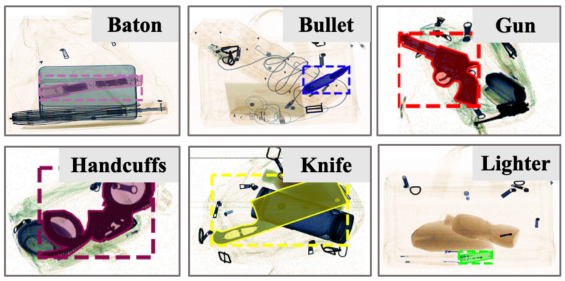


Figure 2. Dataset object examples. Image from [25]

3. Dataset

In this project, we are utilizing the dataset called **PIDray** [25]. PIDray contains 47,677 slices of X-Ray images collected in different scenarios (such as airport, subway stations, and railway stations). It covers total 12 categories of prohibited items, namely guns, knife, wrench, pliers, scissors, hammer, handcuffs, baton, sprayer, power-bank, lighter and bullet. The distribution of object is shown in Fig. 1 and examples are shown in Fig. 2. Each image is provided with image-level and instance-level annotation. The annotation contains labeled bounding box and segmentation mask. Images are split into 29,457 (around 60%) and 18,220 (around 40%) images as train and test sets. Meanwhile, based on the difficulty degree of prohibited item detection, the test images are grouped into three subsets: *easy*, *hard* and *hidden*. Specifically, images in the subset marked as *easy* only contain one prohibited item. Images in the subset named *hard* contain more than one prohibited items. The subsets marked as *hidden* contains images that have deliberately hidden prohibited items.

4. Methods

We mainly use three models to detect prohibited objects.

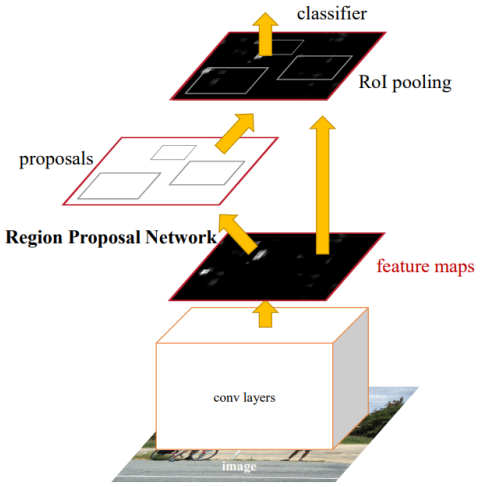


Figure 3. Illustration of RPN structure. Image from [22]

4.1. Faster R-CNN

Faster R-CNN [22], illustrated in Fig. 3 is a two-stage object detection architecture presented in 2015. It is composed from four parts. The first part is the backbone which is used to extract image feature maps. The second component is the Region Proposal Network placed upon the feature map and the third part is the RoIPolling layer which is used to extract accurate feature maps for proposals. The last part is the detection head that classifies the proposal as well as regresses the bounding box.

RPN is a small neural network sliding on the last feature map of the convolution layers. The main purpose of this network is to output proposals that might cover an object. It uses N anchor boxes at each location. Anchors are translation invariant, which means the same ones is used for every location. Regression, which is one of convolutional layer in RPN, can give offsets from anchor boxes, and classification, which is another convolutional layer, gives probability of the anchors that show objects.

After output all predicted boxes, Non-Maximum Suppression (NMS) [9] is applied over the predicted bounding boxes using their predicted scores as criteria for filtration. Before processed by NMS, the Regions of Interest (ROI) are preprocessed by clipping and removing those with height, width beyond a threshold value. The top ROIs alone are taken and sorted by their confidence scores. After this process, NMS enumerates and compares all ROIs. If the IOU of this comparison results in a value greater than a predefined threshold, then that latter ROI is popped from the list.

4.2. YoloV5

Among single-stage detectors described in Sec. 2.2, we select YoloV5 [10] as our experiment model. The reasons

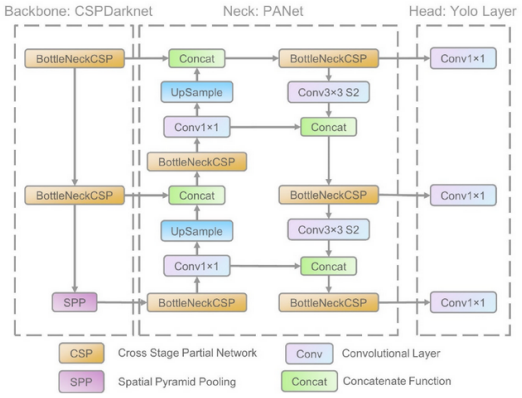


Figure 4. Illustration of YoloV5 model structure.

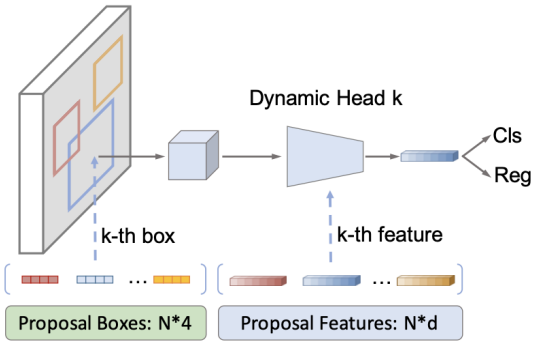


Figure 5. Illustration of Sparse R-CNN pipeline. Image from [23]

are that YoloV6 [11] and YoloV7 [27] are developed on the basis of YoloV5 code base and YoloV5 trains faster than both YoloV6 and YoloV7. Although the performance of YoloV5 on COCO dataset [14] doesn't achieve those of YoloV6 and YoloV7, we think that it is the best choice for this project due to limited training time and computation resources.

YoloV5 contains mainly three parts, backbone, neck and head, as illustrated in Fig. 4. The backbone of YoloV5 is combining Cross Stage Partial [26] block modules and residual modules [8]. The neck of YoloV5 is a combination of Feature Pyramid Network [12] and Path Aggregation Network [15]. The head of YoloV5 is a common Yolo detection head. They provide five versions of the model with different depths and widths.

YoloV5 utilizes Mosaic Augmentation as well as fast scaling augmentation on its input images. The loss calculation of YoloV5 is a combination of CIOU Loss [29] for bounding box loss calculation and Cross Entropy Loss for classification loss calculation. And the output of YoloV5 utilizes DIOU-NMS [28] to get a better Non Maximum Suppression output performance.

4.3. Sparse R-CNN

Sparse R-CNN [23] is a combination of Faster RCNN and DETR [2], which abandons anchor, RPN and NMS, but uses proposal bounding boxes and proposal features that are similar to DETR. Sparse RCNN mainly consists four parts: a backbone network, a dynamic instance interactive head and two task-specific prediction layers. The backbone network produces multi-scale feature maps from the input image. Proposal boxes and proposal features are two learnable parameters. They use RoIAlign to extract feature map for each box and then feed the feature to the head, where each box has its own head. Then Hungarian algorithm is applied to find the match of predictions and ground truth boxes.

The loss function of Sparse RCNN consists of three parts, focal loss of classification, L1 loss of center of the box and GIOU loss of width and height of the box. It also applies set prediction loss on a fixed-size set of proposal boxes.

5. Experiments

5.1. Faster RCNN

For this experiment, we use MMDetection [3] to implement the model training and testing. MMDetection is developed by OpenMMLab, and is one of the popular packages in object detection field. We implement Faster R-CNN based on this package, and utilize the training and testing tools to perform experiment.

Faster R-CNN [22] has 42 millions parameters. We are interested in seeing the different performances between pretrained Faster R-CNN and non-pretrained Faster R-CNN. Therefore, we conduct two experiments based on this model architecture. The first experiment is to train the model from scratch, which means the weights are randomly initialized. The second experiment focuses on the behavior of the pretrained model. We load the model with weights pretrained on the COCO dataset [14]. In addition, to decrease the computing time and improve the training process, we resize the image to 320×320, and then use random flip augmentation.

For both experiments, we utilize step learning rate scheduler. In this scheduler, initial learning rate is set to 0.02 and after 8 epochs the learning rate decays to 0.0002. We set 50 epochs training and run the process on NYU HPC Greene Cluster, with one RTX8000 GPU, setting the batch size at 16. After all training epoch, both models are evaluated based on the test tools from mmdetection [3] and testing set. We mainly focus on the evaluations on bounding boxes and classification.

5.2. YoloV5

YoloV5 [10] by Ultralytics provides five different sizes of the model with different depths and widths. We run two experiments using the medium-size model `yolov5m`, which has 20.9 million parameters. The first experiment is to train the model from scratch, i.e. weights are randomly initialized. And the second experiment is to train the model which is loaded with weights pretrained on the COCO dataset. Since the numbers of classes are different, the YoloV5 head is randomly initialized and the backbone with the neck is initialized with the pretrained weights.

In both experiments, we scale the image size to 320×320 and use one cycle learning rate scheduler, set with three warmup epochs, the initial learning rate at 0.0005, the final learning rate at 0.00001 and 500 epochs. We run the training process on NYU HPC Greene Cluster, with one RTX8000 GPU, setting the batch size at 128. After each training epoch, we evaluate on test set and record the log information.

5.3. Sparse RCNN

Sparse RCNN has 105.96 million parameters with backbone using ResNet-50 with FPN. We also conduct two experiments: non-pretrained Sparse RCNN with randomly initialized weights, and pretrained Sparse RCNN with weights pretrained on the COCO dataset.

Both experiments are launched with MMDetection [3]. The number of stages of the backbone is set to 6. We set 50 epochs for both pretrained and non-pretrained Sparse RCNN, with 12 number of classes for classification, 16 batch size of a single RTX8000 GPU on NYU HPC Greene Cluster. We use AdamW [18] optimizer with initial learning rate at 0.000025. We use step learning rate scheduler and after epoch 8, the learning rate reduces to 0.0000025. Besides, in order to shorten the training timeand improve the experiment performance, we scale the images to 320 and apply random flip augmentation.

6. Results

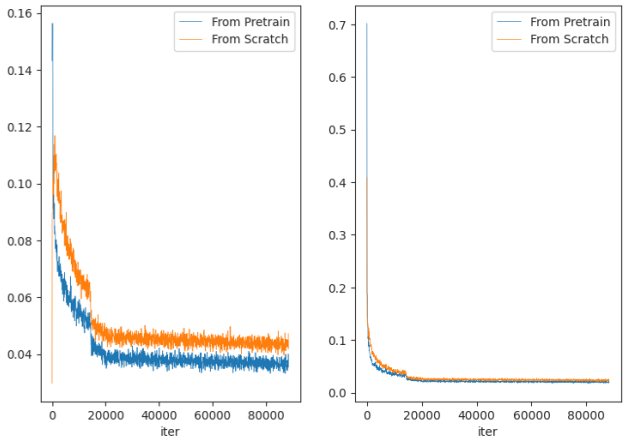
We use standard COCO mAP as our performance evaluation metric. COCO mAP is computed in the following steps. **1)** Select 10 IoU thresholds between 0.50 and 0.95 and compute the Average Precision for each class. **2)** Compute the mean of the Average Precision among 12 classes. We decide to use *easy* test set to evaluate our model performance.

6.1. Pretrain VS Scratch

Faster R-CNN Based on the training process, the losses of regression and classification have converged. Fig. 6 illustrates the loss curves of regression and classification from

Method	mAP
Faster-RCNN / Scratch	0.656
Faster-RCNN / Pretrain	0.697
Sparse-RCNN / Scratch	0.639
Sparse-RCNN / Pretrain	0.689
YoloV5 / Scratch	0.643
YoloV5 / Pretrain	0.646

Table 1. Final test results on *easy* test set.



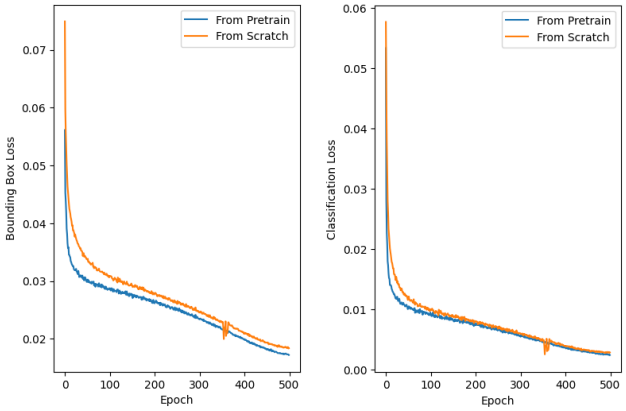
(a) Regression loss VS Iteration (b) Classification loss VS Iteration

Figure 6. Faster R-CNN Training Loss

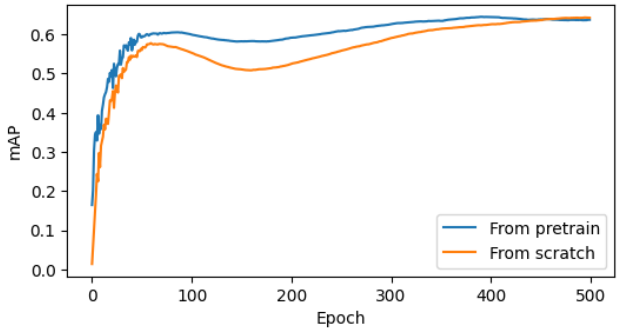
Faster R-CNN training process. From Fig. 6a, the pretrained model shows better performance, while according to Fig. 6b, the loss curves are similar. This might be caused by randomly initializing both model heads.

Another interesting point that we notice from Fig. 6 is the loss curves of the pretrained model and non-pretrained model show a falling off a cliff at around 8 epochs. This is due to the decay of the learning rate.

YoloV5 Due to the training time limit, we trained both experiments with 500 epochs. Although we can't tell if they converged, the mAP doesn't change significantly for both experiments. From Fig. 7a, Fig. 7b and Fig. 7c, we can see that at the beginning, transfer learning can boost the performance of YoloV5 because it reduces the bounding box regression loss significantly. The classification losses of the two experiments do not have a huge difference because the heads are all initialized randomly. As it continues training, the regression losses are get closer and in the end, two experiments perform similarly. We can tell that under the situation of this data domain and YoloV5, transfer learning costs less time to get to the best performance than training from scratch, but training from scratch can also achieve the same performance using a longer time. We can see a per-

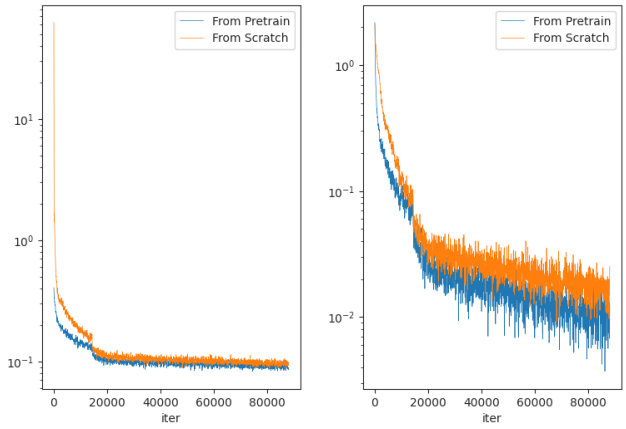


(a) Regression loss VS Epoch (b) Classification loss VS Epoch



(c) mAP on *easy* test set

Figure 7. YoloV5 Training Loss and mAP[1]



(a) Regression loss VS Iteration (b) Classification loss VS Iteration

Figure 8. Sparse RCNN Training Loss (Y axis in log scale).

formance decrease at around epoch 60 for both experiments, given that the training losses are continuously dropping. We don't figure out why but we guess that the model falls into local minima. And after some epochs, their performances both continue to grow.

Sparse RCNN We train models with 50 epochs, which is sufficient. As shown in Fig. 8, both pretrained and non pretrained Sparse RCNN step into the convergent phase successfully. The regression loss curves Fig. 8a show that the pretrained weights lead to a lower loss since the very beginning due to the knowledge learned from other domains. The classification loss curves Fig. 8b are similar but pretrained weights loss still lies lower. Possibly the pretrained model can extract better feature maps compared with the model trained from scratch.

What's more, we can see that the loss curves of both pretrained and non pretrained model drop dramatically at about epoch 8. The decay in learning rate is the reason for this phenomenon. The initial learning rate is set to be 0.000025. But with learning rate scheduler, the learning rate decreases to 0.0000025 at epoch 8.

Other Observations From Fig. 9 we notice an interesting phenomenon that for all three models, gun, sprayer and knife classes have significantly lower average precision compared with other classes under the fact that the numbers of objects in three classes are not particularly low. We don't have an explanation for this phenomenon now and it's worth feature research because guns and knives are extremely dangerous objects to carry on public transportation.

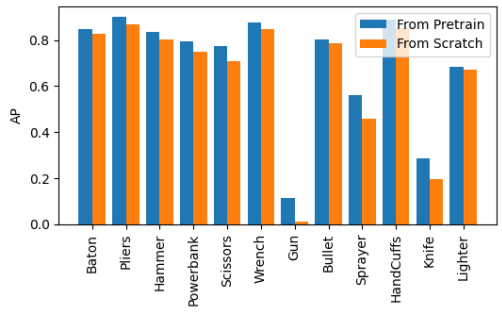
6.2. Model Comparison

In the X-Ray image domain, with the default value of hyperparameters for three models, we can tell that YoloV5 medium-size model performance is slightly worse than the performance of Faster-RCNN, which is the same as our intuition that two-stage models perform better than single-stage models. YoloV5 trains much faster than Faster RCNN in each epoch, but it turns out that Faster RCNN converges earlier than YoloV5 with much fewer epochs. However, Sparse RCNN doesn't perform as well as our expectation. From the original paper, Sparse RCNN should train faster than Faster RCNN and achieve higher mAP. But with the default hyperparameters, Sparse RCNN trains slower than Faster RCNN and performs worse in both experiments.

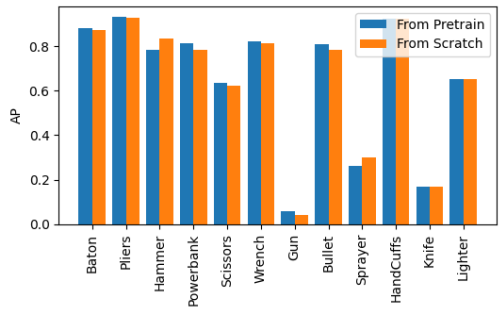
7. Conclusion

When training a model on a custom new dataset, transfer learning can definitely boost the training process of object detection. For most models, such as Faster RCNN and Sparse RCNN, using pretrained model can lead to much better performance (4% higher mAP) after the model converges, though it won't shorten the time to converge. However, some models may not require pretrained weights such as YoloV5, since the pretrain weights don't lift the performance, but using pretrained weights can shorten the training time to converge. Also under the situation that security

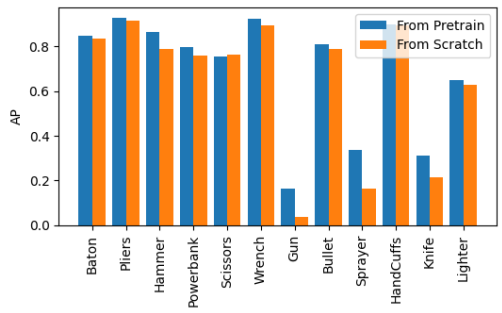
[1] the kink on the loss scratch line is due to the unexpected HPC crush and resume training



(a) Faster RCNN



(b) YoloV5



(c) Sparse RCNN

Figure 9. Class-wise AP on *easy* test set

check doesn't require a fast inference time, using two-stage object detection models is a better choice due to their better performance compared with single-stage models.

Acknowledgement

Thanks to New York University IT HPC team that offered computing resources to support our project.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. 1

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *CoRR*, abs/2005.12872, 2020. 1, 3

[3] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 3, 4

[4] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. *CoRR*, abs/1904.08189, 2019. 1

[5] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. 1

[6] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. 1

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. 1

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 3

[9] Bernt Schiele Jan Hosang, Rodrigo Benenson. Learning non-maximum suppression. *Computer Vision and Pattern Recognition*, abs/1911.08287, 2017. 2

[10] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, (Zeng Yifu), Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, Nov. 2022. 1, 2, 4

[11] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei. Yolov6: A single-stage object detection framework for industrial applications, 2022. 1, 3

[12] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016. 3

[13] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017. 1

[14] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. 3

[15] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. *CoRR*, abs/1803.01534, 2018. 3

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. 1

[17] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *CoRR*, abs/2103.14030, 2021. 1

[18] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. 4

[19] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. 1

[20] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016. 1

[21] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. 1

[22] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. 1, 2, 3

[23] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, and Ping Luo. Sparse r-cnn: End-to-end object detection with learnable proposals. *Computer Vision and Pattern Recognition*, 2020. 1, 3

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. 1

[25] Boying Wang, Libo Zhang, Longyin Wen, Xianglong Liu, and Yanjun Wu. Towards real-world prohibited item detection: A large-scale x-ray benchmark. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5412–5421, 2021. 2

[26] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of CNN. *CoRR*, abs/1911.11929, 2019. 3

[27] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022. 1, 3

[28] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. *CoRR*, abs/1911.08287, 2019. 3

[29] Zhaohui Zheng, Ping Wang, Dongwei Ren, Wei Liu, Rongguang Ye, Qinghua Hu, and Wangmeng Zuo. Enhancing geometric factors in model learning and inference for object detection and instance segmentation. *CoRR*, abs/2005.03572, 2020. 3

[30] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *CoRR*, abs/1911.02685, 2019. 1