Automatic Waste Detection on ZeroWaste

Zhengqi Dong, Zeyu Gu,Tao Zhang, Yufan Lin
{dong760, zgu, mtao, eric1025}@bu.edu

1. Task

Currently the issue of waste detection has drawn the public's attention. Due to a recent report by The World Bank, the waste production is predicted to be 3.4 billion tons by 2050 which would become a disaster [1]. How to deal with waste will be an inevitable problem. So, effectively and accurately classifying waste has become a solution. In this project, our group mainly focuses on the detection of waste. We will use the Zero Waste dataset, which is the largest public waste detection dataset, as a foundation to start our implementation. We aim to use computer vision and deep learning techniques to build a more efficient and accurate model for detection.

2. Related Work

The task of identifying different types of waste in images boils down to the task of object detection. This section presents some state-of-the-art architecture used in object detection[2, 3] as well as some publicly available waste datasets.

2.1 SOTA architectures

There have been several successful architectures proposed during the last ten years that have achieved state-of-the-art results in object detection. Here, we discuss two of the state-of-the-art architectures for object detection.

YOLOv4 YOLOv4[13] is a one-stage object detector that builds on previous YOLO models. It uses YOLOv3 as its head, SPP[19] and PANet[20] as the neck, and CSPDarknet53 as the backbone. The main contribution of YOLOv4 is that the authors introduced vision techniques called Bag of Freebies(BoF) and Bag of Specials(BoS) and how the combination of these techniques added to the network can create accurate and efficient object detectors

Scaled YOLOv4 Scaled YOLOv4 [3] is a redesign of the YOLOv4 network based on the CSP [5] approach and is able to be scaled both up and down to accommodate both small and large networks while maintaining optimal speed and accuracy. The CSP is a new way to architect the CNN that can save computations for various CNN networks. The authors of Scaled YOLOv4 propose a network scaling approach that can modify not only the depth, width, resolution, but also the structure of the network [3].

YOLOR YOLOR[21] stands for "You Only Learn One Representation". It is an object detection model, but it is not a continuation of the YOLOv1-v4 series. The design of YoloR was inspired by the way that knowledge was being processed in the human brain, where we not only can understand the physical world based on explicit knowledge (aka normal or conscious learning, e.g., data we perceived via vision, hearing, tactile) but also on implicit knowledge (aka subconsciousness learning, e.g., past experiences). Based on this idea, the YoloR paper described a new approach to combine these two pieces of knowledge to form an effective unified CNN network that can perform various tasks simultaneously, such as kernel space alignment, prediction refinement, and multi-task learning. Specifically, in the YoloR model, explicit knowledge refers to direct knowledge based on observation, such as input data, annotated labels, or rough features being extracted in the shallow neural network, and implicit knowledge refers to knowledge that does not directly correspond to observation, such as those abstract high–level features that were extracted from deep layers.[27,28]

Dynamic R-CNN Dynamic R-CNN[17] focuses on adjusting the second stage classifier and regressor to fit the distribution change of proposals. It contains two main parts, the dynamic label assignment and dynamic smooth L1 loss. Compared to Faster R-CNN, these two methods will improve the overall performance of Dynamic R-CNN.[17]
In [7], Bashkirova et al. showed that object detection methods especially struggled with labeling small objects correctly, so this is one aspect we will pay attention to in our project.

2.2 Waste Detection Datasets

As automatic waste detection attracted more attention, several waste datasets [15, 16] have been created for

researchers to develop a better methodology to tackle the problem.

TACO TACO[15] is an open image dataset of waste in the wild. The TACO dataset currently contains 1500 annotated images with 60 classes[15]. The annotations are provided in the well-known COCO format. TACO is often used to evaluate the performance of object detection models. One downside of the TACO dataset is that it may contain some user-induced errors and bias due to the crowd-sourcing nature of the dataset.

ReSort-IT ReSort-IT[16] is one of the more recent datasets created for the purpose of developing better object detection models based on deep learning. It contains 16000 synthetic images. The dataset is publicly available on Github. One downside of this dataset is its synthetic nature, which may differ a lot from real-world waste site scenarios.
We discuss our choice of the dataset in section 4.

3. Approach

The goal of object detection is to detect the classes of objects and their location information in the given image. In the ZeroWaste[7] paper, the authors have applied three detection models, RetinaNet, MaskRCNN, and TridentNet, to show the feasibility of this approach. The best result was shown in the RetinaNet model with AP=24.2, AP50=36.3, AP75=26.6, AP_s=4.8, AP_m=10.7, and AP_l=26.1, which will be used as the baseline model for performance comparison in our project.
Our work will be extended based on the ZeroWaste project and push the limit to the next level. Our approaches will first mainly focus on Object Detection. After achieving a desirable accuracy and detecting speed on `ZeroWaste-f`, the fully-annotated dataset, we will then consider other techniques or approaches to further improve the performance, which potentially can make the training more efficient and scalable to the real-world image dataset, such as exploring an advanced model in image segmentation, solving the long-tailed problem, improving accuracy on small object detection, and laborious process of data collection and annotation.
In order to achieve this goal, there were several tasks that we planned to implement:

1. Test and determine which is the best one-stage object detection model for our project, by leveraging the trade-off between accuracy and speed amount three models: YOLOv4 [12], scaled YOLOv4 [13], and YoloR [14].
2. Implement and test the two-stage object detection model Dynamic R-CNN based on the dataset [17].
3. Data collection: What is the accuracy we can get from those models with different throughputs 15 fps, 30 fps, 45 fps, and 60 fps? Report the test/val data size, AP, AP50, AP75, AP_S AP_M, and AP_L on both validation and testing datasets (Read here to learn more about the evaluation metrics, COCO Evaluation Metrics, A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit, Object Detection Metrics With Worked Example)
4. Compare the result we collected from a one-stage object detection model with a two-stage model Dynamic R-CNN, and determine which one is more suitable for our project?

In addition to the basic tasks described above, we also explored different techniques that could potentially improve the performance of the base models. We separate the discussion on optimization techniques used on YOLO models and Dynamic R-CNN.

3.1 Implementation of YOLO models

We trained our selected YOLO models (YOLOv4 and YOLOR) based on WongKinYiu's PyTorch implementation. We provided the link to these implementations in our repo readme as well as in the reference section of this report. These PyTorch implementations are close mimics of the original Darknet framework, which are highly customizable, allowing us to change the network structure and training options. Since the implementation was originally designed to work on the COCO dataset, we have made several modifications for it to work on our customized dataset. We have incorporated our forked version of the original implementation in our repo. After configuring the model, we put our focus on exploring some of the optimization techniques first proposed in the YOLOv4 paper.

3.1.1 Optimization Techniques on YOLO models

The YOLOv4 paper proposed two sets of techniques that can be used to improve the model performance. The first set of techniques is called the "bag of freebies". These are techniques that increase the accuracy of object detection but do not increase the inference cost (e.g. latency in the inference step). Most

of these methods make improvements in the data augmentation or data management phase of the training pipeline. The other set of techniques is called the “bag of specials”. These are often plugin modules or post-processing methods that only increase the inference cost by a small amount but can significantly improve the accuracy of object detection. They often enhance certain attributes in a model, such as receptive field, or strengthening feature integration capability. We mainly put our focus on the “set of freebies” techniques as they are more manageable to implement. We list the techniques we explored below:

Photometric Distortion: Changing the brightness, contrast, saturation and noise in an image to generate more varieties of the same image. We only augmented the hsv colorspace of the images.

Geometric Distortion: This augmentation technique includes random scaling, cropping, flipping, and rotating.

The two techniques described above were mainly used for generating more training data as we only have ~3000 images in the training dataset. Besides increasing the number of training images, these techniques can also have some effects on improving model performance. Image scaling is helpful in increasing the accuracy of small object detection. Image rotation and augmenting colorspace also prevent the model from depending too much on the orientation or the pixel pattern of the image during the “learning” process.

Mosaic: Mosaic[13] combines 4 training images into one in certain ratios. This is a new data augmentation technique proposed in the YOLOv4 paper. By combining 4 images together, we allow the model to detect objects out of their normal contexts. It also allows the model to learn how to identify objects at a smaller scale. Also, since batch normalization calculates activation statistics from 4 different images on each layer, this significantly reduces the need for a large mini-batch size.

Mixup: Mixup[22] simply averages out two images and their corresponding labels according to a specific ratio and forms new data. This provides continuous samples of data in between the different classes, which intuitively expands the distribution of the training set and thus makes the network more robust in the testing phase.

Dropblock regularization: DropBlock[23] can be thought of as a form of structured dropout. In Dropblock, a block section of the pixels in the image is dropped out, whereas in dropout, individual pixels are dropped out randomly. The problem with dropout is that it may not work well on convolutional layers. In convolutional layers, neighboring pixels are highly correlated, so even if some of the pixels are dropped, the spatial information may still remain detectable. Instead of dropping individual pixels, we drop a section of block_size x block_size pixels from the image in dropblock. This technique forces the network to learn features that it may not otherwise rely upon. Figure 1 shows the difference between dropout and Dropblock.

Spatial Dropout regularization: Spatial dropout[24] is another dropout technique that is designed for convolutional layers. Spatial dropout extends the dropout value across the entire feature map. The adjacent pixels in a dropped out feature map are either all 0 or all being active. Figure 2 shows the difference between dropout and spatial dropout.

In our experiments, we injected the dropblock and spatial dropout module in the FPN block, following the approach mentioned in the PP-YOLO[25] paper.

The techniques described in this section did improve the model performance, but only to a limited extent. Due to time constraints, we weren’t able to test the boost in performance resulting from each of these techniques. Instead, we tested the data augmentation techniques as a whole and the regularization techniques individually. The result is shown in table 1.

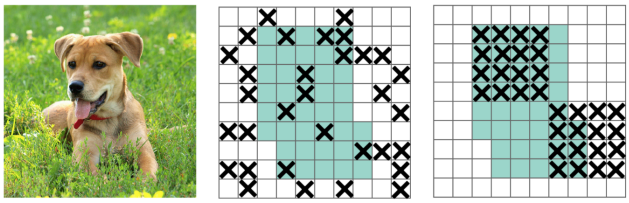


Figure 1. image in the middle is dropout, the image on the right is dropblock

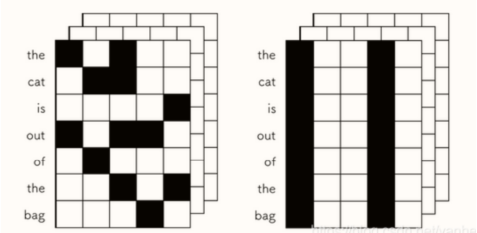


Figure 2. The image on the left is dropout, the image on the right is spatial dropout

Methods	mAP@.5	mAP@.5:.95
YOLOv4 with no augmentation	0.27	0.164
YOLOv4 with augmentations	0.406	0.26

YOLOv4 with augmentation and dropblock	0.454	0.302
YOLOv4 with augmentation and spatial dropout	0.44	0.292

Table 1. Comparisons between techniques

3.2 Implementation of Dynamic R-CNN

We trained and tested the two-stage object detection model Dynamic R-CNN on SCC with one GPU node. The model is integrated into MMDetection, which is an object detection toolbox that contains a rich set of object detection and instance segmentation methods as well as related components and modules [18]. The full details of training Dynamic R-CNN is shown in our repo. To implement this two-stage detection model, we basically set up the environment on the SCC to download the toolbox and tested the model within MMdetection.

3.2.1 Optimization Techniques on Dynamic R-CNN models

In order to enhance the performance of Dynamic R-CNN, we used simpler techniques including utilizing a pre-trained model and adjusting the regularization amount through the parameter called the weight decay for the training.

Pre-trained model: we used a pre-trained Dynamic R-CNN model based upon the COCO dataset instead of the one from scratch. The pre-trained model is included in the MMDetection toolbox package [18]. As our dataset is in COCO dataset format, this optimization technique has been proved to improve the performance in detection.

Weight Decay: Having a model with complex weights will benefit the training process. But the degree of complexity needs to be controlled in order to achieve optimal accuracy. Weight decay, a coefficient in front of the L2 regularization, is hence utilized to penalize the complexity.

4. Datasets

Our project, following Professor Kate's group's work on automated waste recycling, will be based on the industrial-grade waste detection and segmentation dataset named ZeroWaste, specifically the fully-labeled one, `ZeroWaste-f`.

The original `ZeroWaste-f` dataset was split into training, validation, and test sets and stored in the widely used MS COCO format for object detection and segmentation using the open-source Voxel51 toolkit [7]. There are 4503 labeled images in total, and four major classes, including cardboard, soft plastic, rigid plastic, and metal, more specifically, 3002 for training, 572 for validation, and 929 for testing.

The Dynamic R-CNN model is trained on the original dataset as it is. In the YOLO models, however, due to different format requirements, all data used for training YOLO models have been reorganized and processed. Specifically, we split them into training, validation, and testing in a 70/20/10 ratio respectively, so 3148 in training, 900 in validation, and 448 in testing images.

The original images have sizes of 1920x1080. With a small data pre-processing, the whole dataset has been resized to 640x640 to speed up the training process and reduce the memory cost.

5. Evaluation Metrics

At the end of the output layer, we have a confidence score ranging between 0 and 1 for each object, but it does not tell us which class it belongs to. To convert those scores into a class label, a threshold is used, e.g., if threshold=0.5, then only scores equal to or above the threshold will be classified as a corresponding class. Two evaluation criteria that are often used to evaluate the performance: 1) precision tells us how accurate or reliable our prediction is, and 2) recall tells us how good our model is in finding the positives among all predictions (The mathematical formula is shown in Figure 3 below).

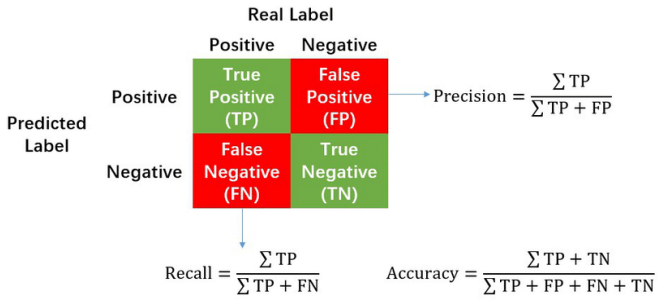


Figure 3: Confusion matrix [33]

According to the Neyman-Pearson lemma [34], two types of errors (false positive and false negative) that exist in these two criteria (precision and recall) are not always equal. For balancing the trade-off between these two criteria, a mean average precision is used to summarize the precision-recall curve across all K

classes into a single value, and the specific formula of average precision is shown below:

$$AP = \sum_{i=1}^{n-1} (r_{i+1} + r_i)\, p_{interp}(r_{i+1})$$

Where, p_{interp} is the interpolated precision at recall level r_i, and r_i is the recall level at which the precision was interpolated, and the formula of mean average precision over K classes is defined below:

$$\text{m}AP = \frac{\sum_{i=1}^{K} AP_i}{K}$$

In Pascal VOC2008 [31], an 11-point interpolated average precision was used. However, in 2015, a more precise 101-point interpolated average precision was used in the COCO dataset evaluation metrics [32].

In object detection, except for the evaluation of prediction, we also need to evaluate the performance of the bounding box for each object and understand how tightly they are fitted to the ground truth annotation. Therefore, a quantitative measurement, IoU(Intersection of Union), is used to evaluate the percentage of the intersection region of predicted and ground-truth bounding boxes over their union area.Throughout many years of object detection competition, different variations of mAP were created based on the strictness of their evaluation. Based on the definition of the COCO dataset [32], the three variations of mAP that were calculated based on different IoU(Intersection over Union) thresholds and were used in evaluating the performance of our project shown in Table 5 are:

1) $AP_{.5..0.95}$: It's the mAP averaged over 10 IoU thresholds (i.e., .5, .55, .6, …, .95), aka $mAP^{IoU@[.5:\ .05:\ .95]}$, and is used as the primary COCO challenge metric.
2) AP_{50}: mAP at IoU=.50 and is identical to the PASCAL VOC metric.
3) AP_{75}: mAP at IoU=.75, another strict metric.

In addition to different IoU thresholds, three variants were used based on the size of detected objects:

1) AP_S: mAP for small objects that covers area less than 32x32
2) AP_M: mAP for medium objects that covers area greater than 32x32 but less than 96x96
3) AP_L: mAP for large objects that covers area greater than 96x96

Note: COCO evaluation metrics make no distinction between AP and mAP[32], and this also applies to our project (alike AR and mAR).

6. Result

In this section, we show our experimental results after training each model on the ZeroWaste dataset. In our experiments, we trained each model for 300 epochs to produce comparable results. We weren't able to test out Scaled-YOLOv4 as planned due to time constraints and some technical difficulties, so we excluded Scaled-YOLOv4 in our discussion and result below.

6.1 Dynamic R-CNN Testing Result

In our experiment, we used the Dynamic R-CNN network which is based on Faster R-CNN architecture, utilizing dynamic ROI head as ROI head and SmoothL1 Loss as loss function. Also, we used ResNet-50-FPN as the backbone.

We trained the model using different numbers of weight decay over 300 epochs to see what the optimal quantity is. Table 2 shows the validation results under different regularization coefficients. The optimal weight decay shown is hence 0.0001.

Weight Decays	Maximum mAP during validation
0.00005	0.214
0.0001	0.241
0.0002	0.212
0.0006	0.202
0.001	0.198

Table 2: Validation result of the model over different weight decays

In Table 3, it shows the average precision of the test result before and after using the two optimization techniques including utilizing a pre-trained model and adjusting the weight decay parameter.

Dynamic R-CNN	From Scratch	Pretrained with optimal weight decay of 0.0001
AP@[0.5:0.95]	27.1	30.8
AP50	40.3	42.6
AP75	30.3	33.5

Table 3: Result of two models before and after the optimization for 300 epochs

We can see that the pretrained model and optimized optimal weight decay in Table 3 show the average precision has been increased, which illustrates the effectiveness of these techniques.

6.2 One-Stage Detector Testing Result

One-stage object detection models skip the process of region proposal and run the detection model directly over a dense sampling of location, and NMS (non-max suppression) is used to produce the bounding box with the highest IOU for each object. In our project, we applied two different one-stage object detection models, including YOLOv4 and YOLOR. For YOLOv4, we used the configuration based on the CSPDarknet53 backbone with leaky relu activation function in convolutional layers. For the network neck, the architecture used CSPSPP mentioned in [12], FPN[26], and PAN[20]. The detection head is composed of YOLO layers. Our best performing YOLOv4 model was the one trained on the augmented data and with a Spatial dropout module injected in the FPN blocks. The configuration file for this model is the "yolov4-custom-2.cfg", which is modified based on "yolov4-csp-x-leaky.cfg"(config files are in the repo). Figure 4 shows the graph for some of the training metrics(box loss, classification loss, etc.) for our best performing YOLOv4 model. For YOLOR, we used the YOLOR p6 configuration, which uses silu activation function in convolutional layers. All models were trained with 300 epochs with the exception of the latest obtained YOLOv4 result, which was trained for 100 epochs. The YOLOv4 result is still comparable with other results since there is a slight overfitting problem when training the model for more epochs. Both YOLOv4 and YOLOR were initialized using weights pretrained on MS_COCO dataset, and the test results are included in Table 5.

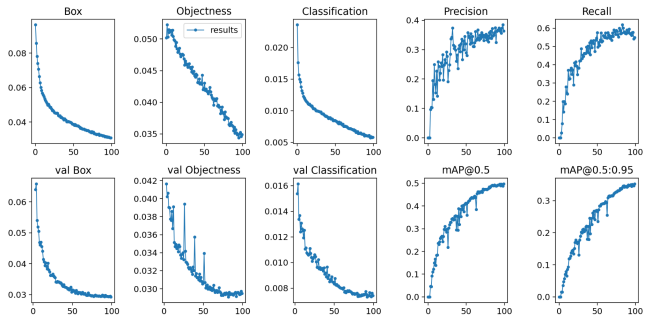


Figure 4. YOLOv4 training performance trained with 100 epochs on images with size 640x640

The impact of Image size on performance:

In this section and the section that follows, we use YOLOR as an example to illustrate some of the problems we noticed during training. In the original implementation of YOLOR, all the input images are required to be in square shape and have a size in multiples of 64, so we conducted three experiments on sizes 448x448, 614x614, and 1088x1088.

The result of mAP@.5 and inference speed are shown in Table 4 and plotted in Figure 5 respectively. We can see that the image size does have an impact on the overall performance, where the performance of larger images can produce higher precision but can be slower in inference speed. However, unfortunately, because training a model on images with higher resolution can require more GPU memory than small size images, the training process on 1088x1088 images size was stopped due to the limited amount of memory available in GPU (614x614 requires to allocate ~10.8GB GPU memory, and the maximum size GPU memory available in SCC Tesla V100 is 16GB).

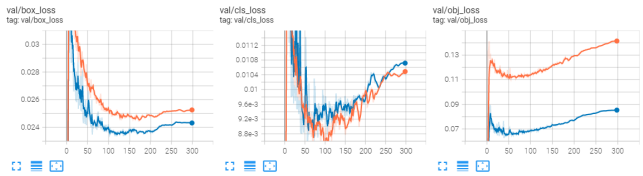


Figure 5: Training performance plots of two trainable image size 448x448 vs 614x614

image_size	mAP_0.5	Inference FPS(ms)
448x448	65.9	164.4
614x614	71.27	99.9
1088x1088	CUDA out of memory	CUDA out of memory

Table 4: Training result of all three groups of image size 448x448, 614x614, 1088x1088

Training Epochs:

As we can see in Figure 6, those training and validation loss being plotted in TensorBoard show a clear "sweet spot" where the model starts to overfit the training data. As we can see after about 150 epochs of training, the validation loss across three criteria starts to increase while the training loss is still decreasing (obj_loss use BCELoss – PyTorch BCEWithLogitsLoss to compute the objectness of bounding boxes, and cls_loss use BCELoss to compute the classification loss for each object). Therefore, it might be a good

strategy to just train the model with 150 epochs to reduce the computational cost. The original implementation of the models had checkpoints implemented after each epoch, and we only used the weights associated with the epoch that produced the best mAP result on the validation set to do test inference.

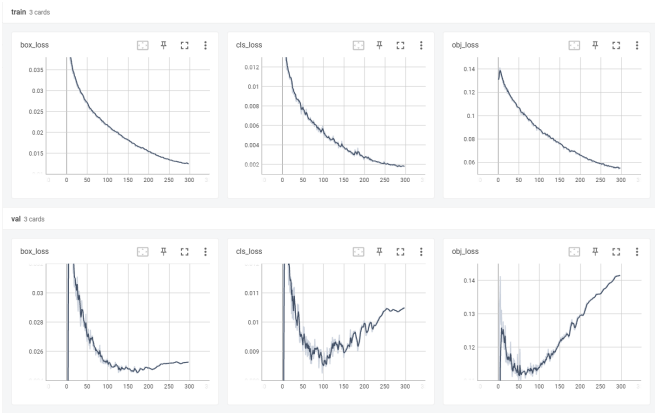


Figure 6: Training Performance of YoloR trained with 300 epochs on image size 448x448.

6.3 Overall Comparison

The experimental results of TridentNet [7], Dynamic R-CNN[17], YOLOv4[13], and YOLOR[21] shows that YOLOR has outperformed all other methods by a large amount. The result of YOLOR has doubled the performance of TridentNet (best result in [7]) (e.g., AP@[0.5:0.95] from 24.2 to 58.7, and AP50 from 36.3 to 72.2). Therefore, we put more focus on optimizing its performance, including different approaches mentioned in the extended works in Section 3 and some fine-tuning techniques in Section 3.3.

Model	$AP_{.5..0.95}$	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
TridentNet (best result shown in [7])	24.2	36.3	26.6	4.8	10.7	26.1
Dynamic R-CNN	30.8	42.6	33.5	4.9	14.6	33.7
YoloV4	39.1	52.9	43.2	12.9	25.5	46.7
Scaled YoloV4	N/A	N/A	N/A	N/A	N/A	N/A
YoloR	62.1	74.2	67.7	28.4	48.0	69.9

Table 5: Mean average precision on the test set of `ZeroWaste-f` of MS-COCO-pretrained TridentNet, Dynamic R-CNN, YoloV4, Scaled YoloV4, YoloR finetuned on `ZeroWaste-f`. Please refer to Appendix E for detailed training, validation, and testing result, and Appendix C&D for predicted image samples.

7. Conclusion

Among all models we have tested so far, YOLOR has achieved the best performance over all evaluation criteria and has outperformed TridentNet greatly, 2.57x higher on AP and 5.92x higher on AP_S. The detailed benchmark comparison between YOLOR and TridentNet is shown in Table 5. Another single-stage detector that we have tested, YOLOv4, also achieved a fairly good result and surpassed the performance of TridentNet, owing to the “Bag of Freebies” [13] technique that was introduced in the YOLOv4 paper. Besides, we noticed that Dynamic R-CNN, as a two-stage detector, did not achieve better performance than a single-stage detector in terms of the mean average precision, even with optimization techniques being added. This indicates that the performance of two-stage detectors might not always perform better than the single-stage detector, and some models can be more suitable for specific datasets than others.

8. Future Works

Memory Optimization: In our experiments, we have encountered CUDA out of memory issues. The main reason for CUDA running out of memory when training with a high-resolution image or a larger batch size was the expensive memory consumption of the Adam optimizer. We believe that one approach to solve the issue is to use some model parallelism strategy to partition the optimizer state among multiple GPU nodes. As we can see in Figure 7, only one of the GPU devices was used when training the YoloR model and left the other three idled. There are some off-the-shelf solutions available to use, and we list two here as examples: 1) use Zero Redundancy Optimizer provided by deepspeed to offload GPU memory to both CPU or/and NVMe memory (require some modification to the code); 2) use DistributedDataParallel provided by PyTorch to perform distributed training.

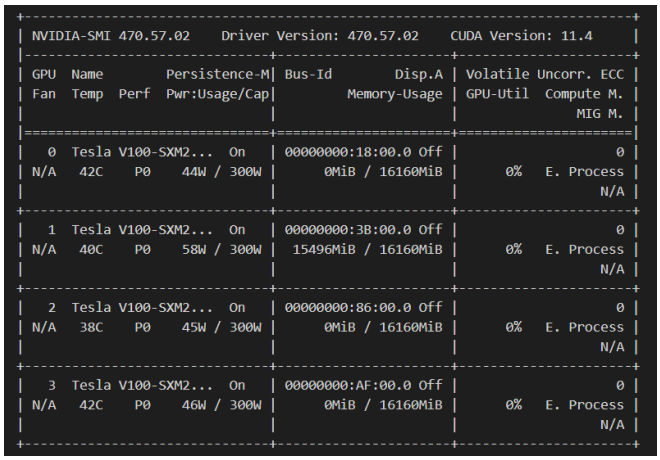


Figure 7: Tesla V100-SXM2 GPU architecture specification.

Small Objects Detection: As we can see in Table 5, the small object detection is still pretty hard across all the models. The test accuracy of YoloR in small object detection is about 40.6% of the accuracy in larger objects and 59.2% in medium-size objects, which shows a spatial room for improvement. There are two approaches that we can try to improve the performance of small object detection: 1) apply tiling on a higher resolution image before training, e.g., perform 2x2 tiling on 1088x1088 images size; [30] 2) apply SAHI(sliding Aided Hyper Inference) that described in this repo. [29]

9. Code Repository

Please check out our repository following this link: https://github.com/Boston-University-Projects/EC523_DL_CV_Project

References

[1] World Bank Group. “Global Waste to Grow by 70 Percent by 2050 Unless Urgent Action Is Taken: World Bank Report.” World Bank, 24 Sept. 2018, www.worldbank.org/en/news/press-release/2018/09/20/global-waste-to-grow-by-70-percent-by-2050-unless-urgent-action-is-taken-world-bank-report.

[2] Y. Li, Y. Chen, N. Wang, and Z. Zhang, Scale-Aware Trident Networks for Object Detection. 2019.

[3] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, Scaled-YOLOv4: Scaling Cross Stage Partial Network. 2021.

[4] K. He, G. Gkioxari, P. Dollár, and R. Girshick, Mask R-CNN. 2018.

[5] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, CSPNet: A New Backbone that can Enhance Learning Capability of CNN. 2019.

[6] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. 2018.

[7] Dina Bashkirova and K. Saenko, “ZeroWaste: Towards Deformable Object Segmentation in Extreme Clutter,” 2021.

[8] Koech, Kiprono Elijah. “On Object Detection Metrics with Worked Example.” Medium, Towards Data Science, 18 Dec. 2021, https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e.

[9] Papers With Code, Semantic Segmentation Benchmarks, https://paperswithcode.com/task/semantic-segmentation

[10] Papers With Code, Object Detection Benchmarks, https://paperswithcode.com/task/object-detection

[11] L. H. Li *et al.*, “Grounded Language-Image Pre-training,” *Arxiv*, 2021.

[12] C.-Y. Wang, A. Bochkovskiy, en H.-Y. M. Liao, “Scaled-YOLOv4: Scaling Cross Stage Partial Network”, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, bll 13029–13038.

[13] A. Bochkovskiy, C.-Y. Wang, en H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection”, CoRR, vol abs/2004.10934, 2020.

[14] C.-Y. Wang, I.-H. Yeh, en H.-Y. M. Liao, “You Only Learn One Representation: Unified Network for Multiple Tasks”, arXiv preprint arXiv:2105. 04206, 2021.

[15] Proença, P. F., & Simões, P. (2020). TACO: Trash Annotations in Context for Litter Detection.

[16] Koskinopoulou, M., Raptopoulos, F., Papadopoulos, G., Mavrakis, N., & Maniadakis, M. (2021). Robotic Waste Sorting Technology: Toward a Vision-Based Categorization System for the Industrial Robotic Separation of Recyclable Waste. IEEE Robotics Automation Magazine, 28(2), 50–60. https://doi.org/10.1109/MRA.2021.3066040

[17] Zhang, H., Chang, H., Ma, B., Wang, N., & Chen, X. (2020, August). Dynamic R-CNN: Towards high quality object detection via dynamic training. In European conference on computer vision (pp. 260-275). Springer, Cham.

[18] Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., ... & Lin, D. (2019). MMDetection: Open mmlab detection toolbox and benchmark. arXiv preprint arXiv:1906.07155.

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” in Computer Vision – ECCV 2014,

Springer International Publishing, 2014, pp. 346–361. doi: 10.1007/978-3-319-10578-9_23.
[20] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, Path Aggregation Network for Instance Segmentation. arXiv, 2018. doi: 10.48550/ARXIV.1803.01534.
[21] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, You Only Learn One Representation: Unified Network for Multiple Tasks. arXiv, 2021. doi: 10.48550/ARXIV.2105.04206.
[22] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, mixup: Beyond Empirical Risk Minimization. arXiv, 2017. doi: 10.48550/ARXIV.1710.09412.
[23] G. Ghiasi, T.-Y. Lin, and Q. V. Le, DropBlock: A regularization method for convolutional networks. arXiv, 2018. doi: 10.48550/ARXIV.1810.12890.
[24] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, Efficient Object Localization Using Convolutional Networks. arXiv, 2014. doi: 10.48550/ARXIV.1411.4280.
[25] X. Long et al., PP-YOLO: An Effective and Efficient Implementation of Object Detector. arXiv, 2020. doi: 10.48550/ARXIV.2007.12099.
[26] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, Feature Pyramid Networks for Object Detection. arXiv, 2016. doi: 10.48550/ARXIV.1612.03144.
[27] V. Meel, "Yolor - you only learn one representation (what's new, 2022)," *viso.ai*, 17-Jan-2022. [Online]. Available: https://viso.ai/deep-learning/yolor/. [Accessed: 05-May-2022].
[28] A. Tewari, "Yolor model architecture," *OpenGenus IQ: Computing Expertise & Legacy*, 09-Mar-2022. [Online]. Available: https://iq.opengenus.org/yolor/. [Accessed: 05-May-2022].
[29] Akyon, F.C., Altinuc, S.O. and Temizel, A., 2022. Slicing Aided Hyper Inference and Fine-tuning for Small Object Detection. *arXiv preprint arXiv:2202.06934*.
[30] J. Solawetz, "Small object detection guide," *Roboflow Blog*, 28-Feb-2022. [Online]. Available: https://blog.roboflow.com/detect-small-objects/. [Accessed: 05-May-2022].
[31] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2009.
[32] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," arXiv [cs.CV], 2014.
[33] "Confusion Matrix in Python," Infinite Solutions Blog, [Online]. Available: https://www.ris-ai.com/confusion-matrix#What-is-a-Confusion-Matrix?. [Accessed: 05-May-2022].
[34] I. Miller, M. Miller, και J. E. Freund, *John E. Freund's Mathematical Statistics with Applications*. Pearson, 2014.

Method	pretrained_model	Img Size(px)	FPS(ms)	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
TridentNet (best result shown in [7])	N/A	N/A	N/A	24.2	36.3	26.6	4.8	10.7	26.1
YoloR	yolor_p6.pt	640x640	99.9	62.1	74.2	67.7	28.4	48.0	69.9
Improvement on TridentNet	N/A	N/A	N/A	2.57 x	2.04 x	2.55 x	5.92 x	4.49 x	2.68 x

Table 6. Detailed test result for YoloR

Name	**Task**	**File names**	**No. Lines of Code**
Zhengqi Dong	• Initialized and set up Github repo, group communication channel, and shared Google Drive • Led weekly meeting and provided help and suggestion on other group member's work.	many batch scripts under ./roboflow-ai/, and many place under ./roboflow-ai/yolor folder,	Too many to count…

	• Reformatted image dataset, and performed some preprocessing, such as reorganized dataset with new split ratio, image resizing, tiling, etc. • Fixed bugs that were broken when performing training and testing on a customized yolo formatted dataset. • Modified cocoAPI in test.py to perform evaluation based on our project's need (e.g., AP_{small}, AP_{medium}, and AP_{large}) • Wrote batch script to train yoloR and use it as a template for other group members. • Ran training and testing code for many versions of dataset that has been augmented or preprocessed with different strategies, with download code written in ./rboflow-ai/download_data.py • Wrote a clear and thorough walkthrough guidance in jupyter notebook to reproduce the result, which was stored in ./rboflow-ai/yolor/waste_detection_yoloR.ipynb • Wrote section 'Setting up Environment' and 'Appendix A' of README.md and Resources.md • Wrote section 2.1(YOLOR), 3, 4, 5, 6.2, 6.3, 7, and 8 of report	mainly in: - train.py - test.py - waste_detection_yoloR.ipynb	
Zeyu Gu	• modified YOLOv4 code for it to work on our customized dataset • Modified test evaluation section of YOLOv4 to make cocoAPI work on our customized dataset • Implemented dropblock and spatial dropout module in YOLOv4 • Wrote code to enable user to control augmentation when training YOLOv4 • Created custom YOLOv4 config file • Wrote training batch scripts following Zhengqi's template • Wrote Jupyter notebook on how to train YOLOv4 and dynamic R-CNN • Trained and tested YOLOv4 with techniques mentioned in this report • Wrote most part of the repo readme • Wrote section 2, 3.1, 3.1.1, and parts of section 7 of the report	Under YOLOv4/PyTorch_YOLOv4 folder: - train.py - test.py - models/models.py - utils/layers.py - utils/parse_config.py - Some config files Under notebooks directory: - waste_detection_yolov4.ipynb - waste_detection_D-RCNN.ipynb	~400 excluding the config files because a lot of config files have repetitive parts
Tao Zhang	• Discovered and utilized the MMdetection toolbox for Dynamic R-CNN model • Modified the dataset setting (including the number of classes and the name for each class in the toolbox) to train on Zerowaste Dataset • Modified the training parameters settings including the learning rate and number of training epochs (300) • Utilized the pre-trained model from the toolbox to enhance the performance • Adjusting the weight decay to improve the performance • Wrote batch scripts so that it can be trained automatically on SCC • Trained and tested the Dynamic R-CNN model • Wrote section 3.2, 3.2.1, 4 of the report • Revised the section 6.1, 6.3 of the report	Dynamic R-CNN: Faster_rcnn_r50_fpn.py Coco_detection.py Default_runtime.py Some Config files Batch script files	Revised and added hundreds lines of code to execute the Dynamic R-CNN model to be trained and tested on the dataset
Yufan Lin	• Discovered and utilized analysis tool for Dynamic R-CNN model • Modified the optimizer setting • Utilized the evaluation metrics from analysis tool to evaluate the performance of Dynamic R-CNN model • Utilized the bbox map evaluation from analysis tool to evaluate the performance of Dynamic R-CNN model • Made all of the front end , back end and web page template part from scratch of the UI to demo the result of our models. • Wrote section 1 of the report • Revised the section 3.2,6.3 of the report	Under Dynamic R-CNN: Optimizer file analysis_tools(folder) • eval_metric.py • analyze_results.py • Analyze_logs.py Under Web: • templates(folder) • 523project_database.sql • project_web.py	Revised and added several hundreds of lines for both Dynamic R-CNN and web UI

Table 7. Team member contributions

Appendix A. Detailed Roles

See the table above (Table 7) for each teammate. Include the file names and number of lines of code written.

Appendix B. Code repository

Please check out our repository at this link: https://github.com/Boston-University-Projects/EC523_DL_CV_Project

Appendix C: YoloR prediction and labeling

samples on the test set of `ZeroWaste-f` of with the best_overall model trained on `ZeroWaste-f` over 300 epochs

Figure 8: test_batch2_labels.jpg

Figure 9: test_batch2_pred.jpg

Appendix D: YoloV4 prediction and labeling

Figure 10: test_batch2_labels.jpg

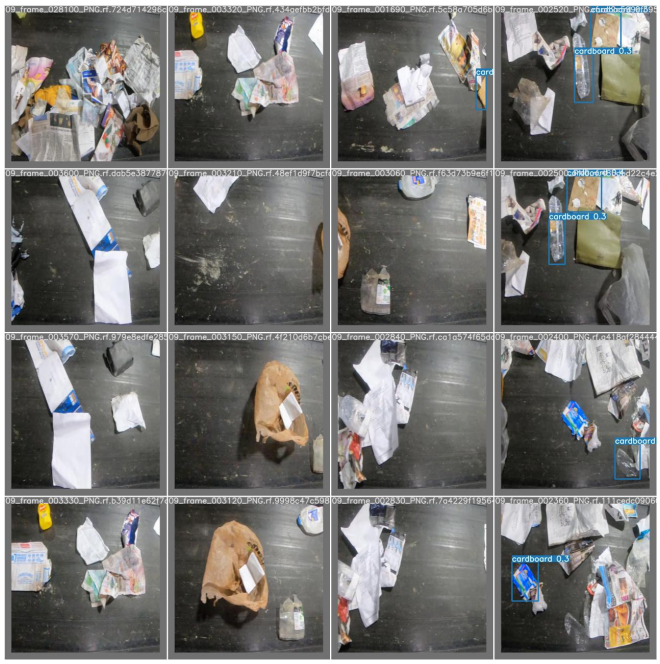


Figure 11: test_batch2_pred.jpg

Appendix E: YoloR Precision-Recall Curve

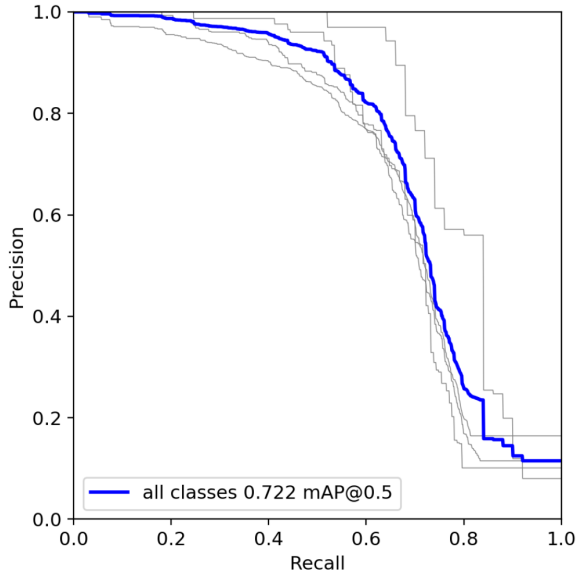
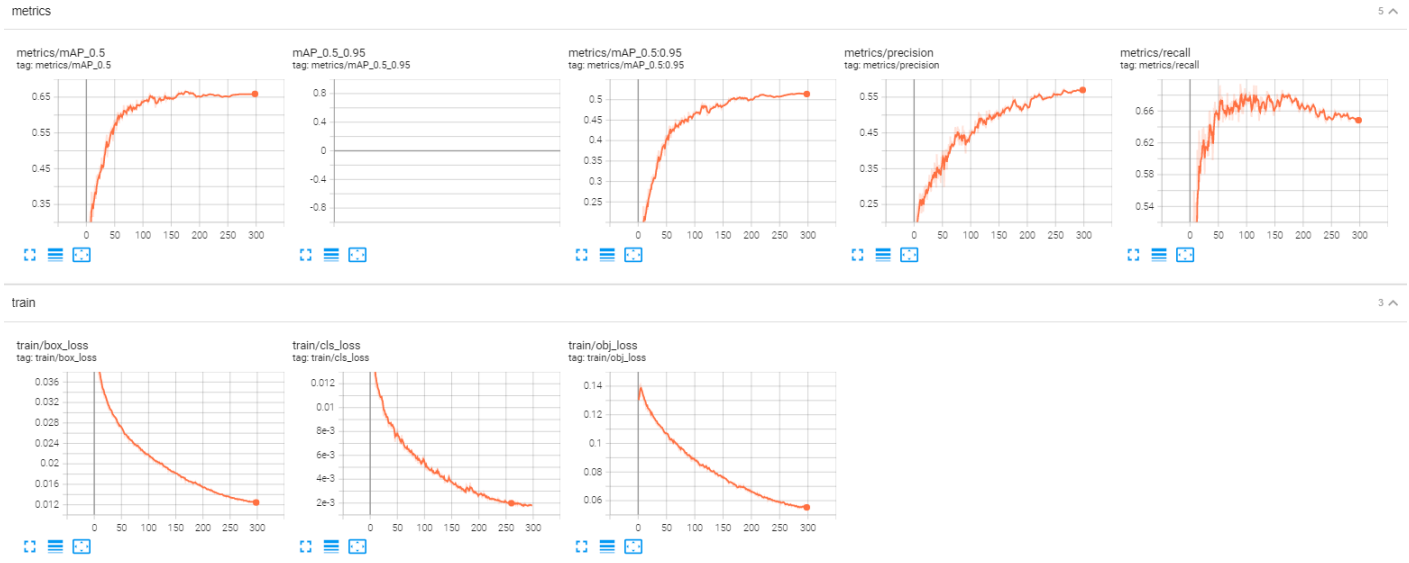


Figure 12: YoloR Precision-Recall Curve evaluated on test dataset on best_overall.pt model that was trained with 300 epochs

Appendix F: YoloR Detailed Training, Validation, and Testing Result

Hyperparameters {'lr0': 0.01, 'lrf': 0.2, 'momentum': 0.937, 'weight_decay': 0.0005, 'warmup_epochs': 3.0, 'warmup_momentum': 0.8, 'warmup_bias_lr': 0.1, 'box': 0.05, 'cls': 0.5, 'cls_pw': 1.0, 'obj': 1.0, 'obj_pw': 1.0, 'iou_t': 0.2, 'anchor_t': 4.0, 'fl_gamma': 0.0, 'hsv_h': 0.015, 'hsv_s': 0.7, 'hsv_v': 0.4, 'degrees': 0.0, 'translate': 0.5, 'scale': 0.5, 'shear': 0.0, 'perspective': 0.0, 'flipud': 0.0, 'fliplr': 0.5, 'mosaic': 1.0, 'mixup': 0.0}

Training result: (300 epochs, 3092 images, with default augmentation)



```
235882  wandb: Run history:
235883  wandb:       metrics/mAP_0.5
235884  wandb: metrics/mAP_0.5:0.95
235885  wandb:    metrics/precision
235886  wandb:       metrics/recall
235887  wandb:       train/box_loss
235888  wandb:       train/cls_loss
235889  wandb:       train/obj_loss
235890  wandb:         val/box_loss
235891  wandb:         val/cls_loss
235892  wandb:         val/obj_loss
235893  wandb:                x/lr0
235894  wandb:                x/lr1
235895  wandb:                x/lr2
235896  wandb:
235897  wandb: Run summary:
235898  wandb:       metrics/mAP_0.5 0.65907
235899  wandb: metrics/mAP_0.5:0.95 0.51377
235900  wandb:    metrics/precision 0.5706
235901  wandb:       metrics/recall 0.65086
235902  wandb:       train/box_loss 0.01234
235903  wandb:       train/cls_loss 0.00177
235904  wandb:       train/obj_loss 0.05571
235905  wandb:         val/box_loss 0.02527
235906  wandb:         val/cls_loss 0.0105
235907  wandb:         val/obj_loss 0.14147
235908  wandb:                x/lr0 0.002
235909  wandb:                x/lr1 0.002
235910  wandb:                x/lr2 0.002
```

```
1006        # Augment
1007        img4, labels4 = random_perspective(img4, labels4,
1008                                           degrees=self.hyp['degrees'],
1009                                           translate=self.hyp['translate'],
1010                                           scale=self.hyp['scale'],
1011                                           shear=self.hyp['shear'],
1012                                           perspective=self.hyp['perspective'],
1013                                           border=self.mosaic_border)  # border to remove
```

Validation result (876 images):

⇒ As we can see, it's enough to just have model to be trained for 100 epochs.

testing result: 448 images

(dl_env)dong760@scc-x05:/projectnb/dl523/projects/RWD/EC523_DL_CV_Project/roboflow-ai/yolor$ python test.py --conf-thres 0.0 --img 640 --batch 32 --device 0 --data ../zero-waste-10/data.yaml --cfg cfg/yolor_p6.cfg --weights runs/train/yolor_p6_2022_05_01-00_37_11/weights/best_ap.pt --task test --names data/zerowaste.names --verbose --save-json --save-conf --save-txtcheck_file -- > file: ../zero-waste-10/data.yaml

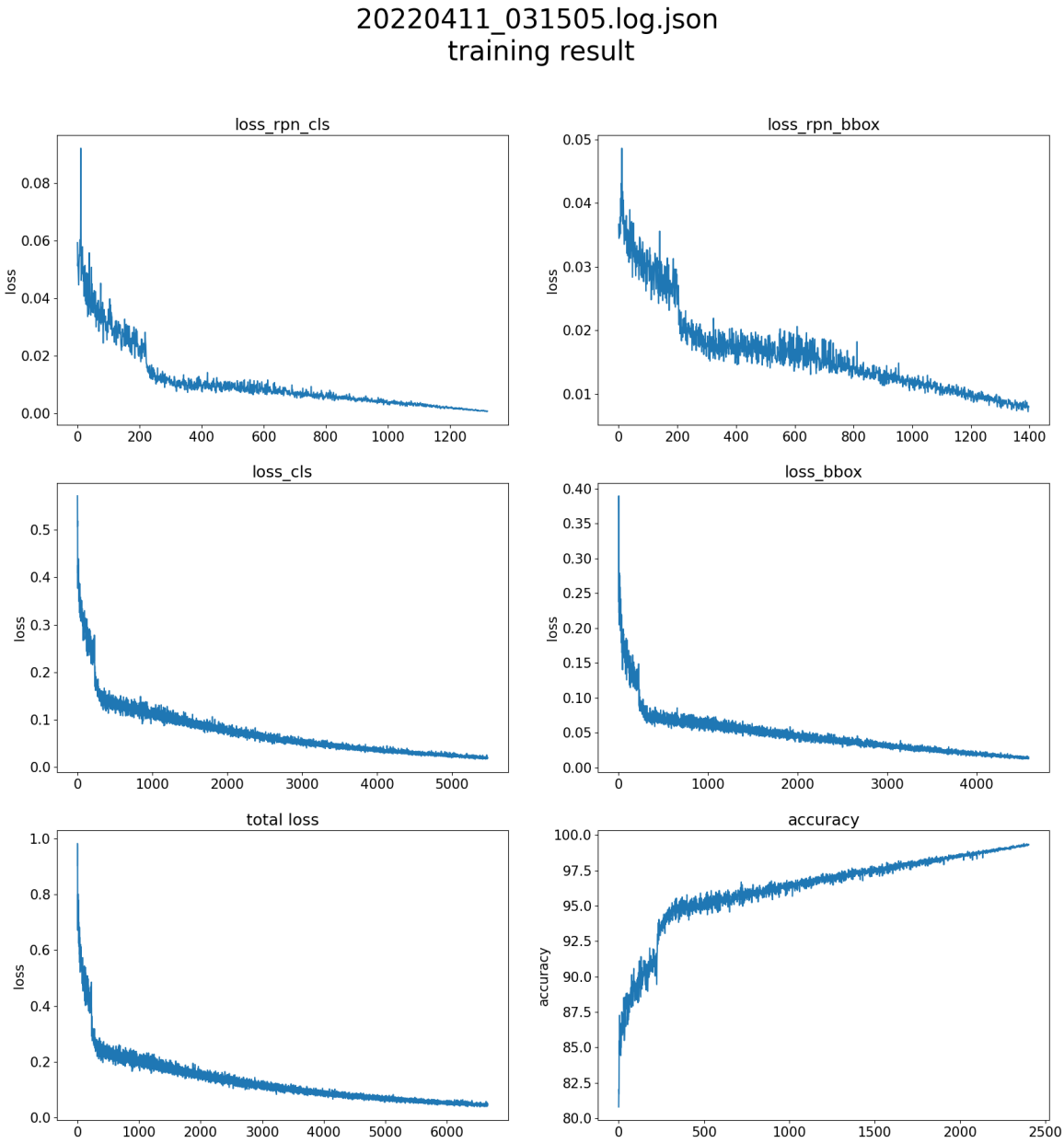
Namespace(augment=False, batch_size=32, cfg='cfg/yolor_p6.cfg', conf_thres=0.0, data='../zero-waste-10/data.yaml', device='0', exist_ok=False, gt_json_dir='../zero-waste-10/test/_annotations.coco.json', img_size=640, iou_thres=0.65,

```
name='exp', names='data/zerowaste.names', project='runs/test', save_conf=True, save_json=True, save_txt=True, single_cls=False, task='test', verbose=True, weights=['runs/train/yolor_p6_2022_05_01-00_37_11/weights/best_ap.pt'])
Using torch 1.11.0 CUDA:0 (Tesla V100-PCIE-16GB, 16160MB)

Model Summary: 665 layers, 36854616 parameters, 36854616 gradients
/usr4/dl523/dong760/.conda/envs/dl_env/lib/python3.8/site-packages/torch/functional.py:568: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered internally at /opt/conda/conda-bld/pytorch_1646755903507/work/aten/src/ATen/native/TensorShape.cpp:2228.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Scanning labels ../zero-waste-10/test/labels.cache3 (442 found, 0 missing, 6 empty, 0 duplicate, for 448 images): 448it [00:00, 17943.38it/s]
                 Class        Images       Targets             P             R       mAP@.5  mAP@.5:.95:     0%|
| 0/14 [00:00<?, ?it/s]=========> type(output): <class 'list'>
                 Class        Images       Targets             P             R       mAP@.5  mAP@.5:.95:     7%|
| 1/14 [00:14<03:11, 14.75s/it]=========> type(output): <class 'list'>
                 Class        Images       Targets             P             R       mAP@.5  mAP@.5:.95:    14%|
| 2/14 [00:26<02:34, 12.88s/it]=========> type(output): <class 'list'>
                 Class        Images       Targets             P             R       mAP@.5  mAP@.5:.95:
100%|                                                                                | 14/14 [03:30<00:00, 15.06s/it]
wandb: Currently logged in as: dragogo (use `wandb login --relogin` to force relogin)
wandb: wandb version 0.12.15 is available!  To upgrade, please run:
wandb:  $ pip install wandb --upgrade
wandb: Tracking run with wandb version 0.12.12
wandb: Run data is saved locally in /projectnb2/dl523/projects/RWD/EC523_DL_CV_Project/roboflow-ai/yolor/wandb/run-20220502_001029-29wlrglc
wandb: Run `wandb offline` to turn off syncing.
wandb: Syncing run sandy-frost-28
wandb: ⭐ View project at https://wandb.ai/dragogo/ec523-zerowaste
wandb: 🚀 View run at https://wandb.ai/dragogo/ec523-zerowaste/runs/29wlrglc
               all        448     2.7e+03       0.691       0.724       0.745       0.618
         cardboard        448    1.75e+03       0.684       0.674       0.698       0.555
             metal        448          50        0.72        0.82       0.825       0.741
       rigid_plastic        448         187       0.712       0.699       0.734       0.621
        soft_plastic        448         713       0.648       0.703       0.725       0.554
Speed: 3.6/58.0/61.7 ms inference/NMS/total per 640x640 image at batch-size 32

Evaluating pycocotools mAP... saving runs/test/exp24/best_ap_predictions.json...
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
Loading and preparing results...
DONE (t=1.04s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=5.41s).
Accumulating evaluation results...
DONE (t=0.82s).
```

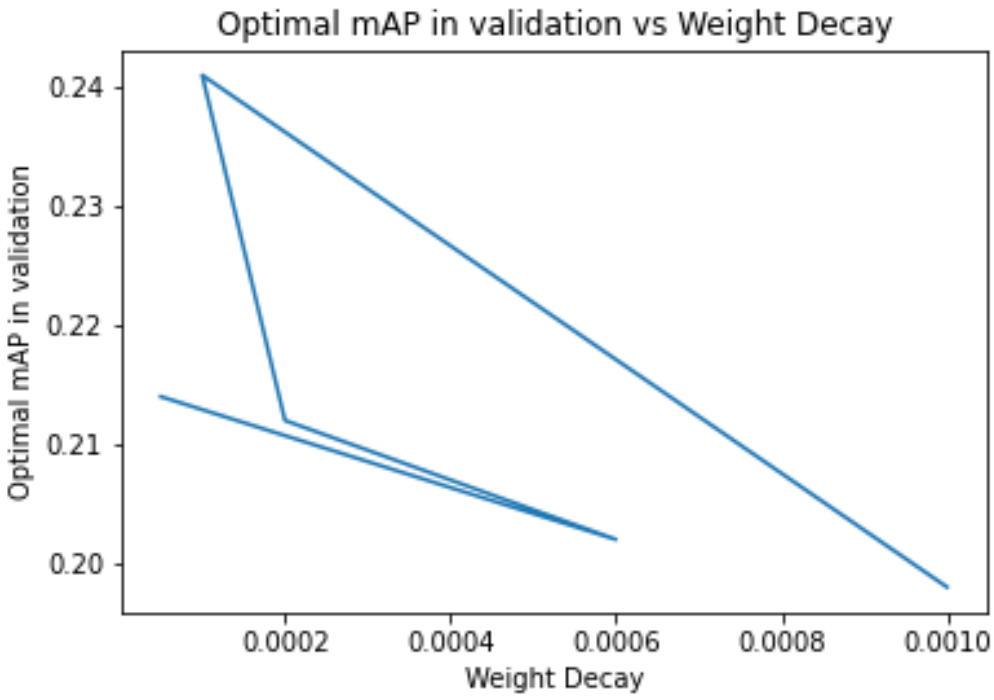
```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.620
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.743
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.677
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.284
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.478
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.699
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.463
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.722
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.767
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.554
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.693
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.816
================> Start Printing evaluated result:
[    0.62046     0.74275     0.67696     0.28352     0.47832     0.69902     0.46331     0.72187     0.76741     0.55393
0.69333     0.81559]
================> End of Printing!
Results saved to runs/test/exp24

wandb: Waiting for W&B process to finish... (success).
wandb:
wandb: Synced sandy-frost-28: https://wandb.ai/dragogo/ec523-zerowaste/runs/29wlrglc
wandb: Synced 6 W&B file(s), 6 media file(s), 0 artifact file(s) and 0 other file(s)
wandb: Find logs at: ./wandb/run-20220502_001029-29wlrglc/logs
```

Note: You can run the following command under roboflow-ai/yolor/ folder to see the plots from TensorBoard, $tensorboard --logdir . --host "0.0.0.0" --port 7777, and all those output log files and various models are saved under ./'runs/train' directory.

Appendix G: Dynamic R-CNN Detailed Training, Validation, and Testing Result
Training Process of the pretrained model (300 epochs)



Validation Results for different weight decays



Testing Process of the model from scratch

```
writing results to ./result/result_10.pkl

Evaluating bbox...
Loading and preparing results...
DONE (t=0.09s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=2.43s).
Accumulating evaluation results...
DONE (t=0.62s).

 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.271
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 0.403
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.303
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.017
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.153
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.298
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.534
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.534
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.534
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.090
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.291
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.576
```

Testing Process of the pretrained model with optimized weight decay

tao — mtao@scc-201:/projectnb/dl523/students/mtao/mmdetection — ssh mtao@scc1.bu.edu — 119×29

```
writing results to ./result/result_11.pkl

Evaluating bbox...
Loading and preparing results...
DONE (t=0.11s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=2.20s).
Accumulating evaluation results...
DONE (t=0.44s).

 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.308
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=1000 ] = 0.426
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=1000 ] = 0.335
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.049
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.146
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.337
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.557
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=300 ] = 0.557
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=1000 ] = 0.557
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=1000 ] = 0.085
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=1000 ] = 0.311
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=1000 ] = 0.598

OrderedDict([('bbox_mAP', 0.308), ('bbox_mAP_50', 0.426), ('bbox_mAP_75', 0.335), ('bbox_mAP_s', 0.049), ('bbox_mAP_m',
 0.146), ('bbox_mAP_l', 0.337), ('bbox_mAP_copypaste', '0.308 0.426 0.335 0.049 0.146 0.337')])
[mtao@scc-201 mmdetection]$
```