YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors

***Abstract*—YOLOv7, standing out as a top-tier object detector, excels in both speed and accuracy across a broad spectrum, ranging from 5 FPS to 160 FPS. With a remarkable 56.8% Average Precision (AP), it claims the highest accuracy among real-time object detectors clocking 30 FPS or higher on the GPU V100. Notably, the YOLOv7-E6 variant, boasting 56 FPS on V100 and a 55.9% AP, outperforms leading competitors such as SWINL Cascade-Mask R-CNN and ConvNeXt-XL Cascade-Mask R-CNN by substantial margins—509% in speed and 2% in accuracy, and 551% in speed and 0.7% in AP, respectively. YOLOv7 outshines other prominent detectors, including YOLOR, YOLOX, Scaled-YOLOv4, YOLOv5, DETR, Deformable DETR, DINO-5scale-R50, ViT-Adapter-B, demonstrating superior performance in both speed and accuracy. Notably, YOLOv7 achieves these results through training exclusively on the MS COCO dataset from scratch, without relying on additional datasets or pre-trained weights.**

I.INTRODUCTION

Real-time object detection holds immense significance in the field of computer vision, serving as a crucial element in various applications like multi-object tracking, autonomous driving, robotics, and medical image analysis. The hardware responsible for executing real-time object detection typically includes mobile CPUs or GPUs, alongside specialized neural processing units (NPUs) developed by major manufacturers. Notable examples include the Apple neural engine, Intel's neural compute stick, Nvidia's Jetson AI edge devices, Google's edge TPU, Qualcomm's neural processing engine, MediaTek's AI processing unit, and Kneron's AI SoCs. These NPUs are designed to accelerate different operations, such as vanilla convolution, depth-wise convolution, or MLP operations. In our research, the primary aim of the proposed real-time object detector is to offer support for a wide range of devices, spanning from mobile GPUs to GPUs in edge and cloud computing environments.

Lately, there has been a continued effort to develop real-time object detectors tailored for diverse edge devices. For instance, initiatives like MCUNet and NanoDet have concentrated on creating low-power single-chip solutions, specifically aiming to enhance inference speed on edge CPUs. On the other hand, methods like YOLOX and YOLOR have prioritized improving the inference speed on various GPUs. Recent advancements in real-time object detection have prominently centered around crafting efficient architectures. Those intended for CPU usage typically draw inspiration from MobileNet, ShuffleNet, or GhostNet. Meanwhile, GPU-focused detectors often leverage ResNet, DarkNet, or DLA, optimizing their architectures using the CSPNet strategy. The proposed methods in this paper diverge from the current mainstream approaches to real-time object detection. Beyond architectural optimization, our emphasis lies in refining the training process. We will concentrate on optimized modules and methods that can enhance the training cost to improve object detection accuracy, all while keeping the inference cost in check. These specialized modules and optimization methods are termed "trainable bag-of-freebies."

This paper contributes in several key ways: (1) Introducing a set of trainable bag-of-freebies methods, enabling significant enhancement in real-time object detection accuracy without introducing additional inference costs. (2) Uncovering two novel challenges in the evolution of object detection methods: the replacement of original modules by re-parameterized ones and the dynamic label assignment strategy for varying output layers. Proposed solutions are presented to address these challenges. (3) Presenting "extend" and "compound scaling" methods specifically designed for the real-time object detector, effectively leveraging parameters and computation resources. (4) Demonstrating the effectiveness of our proposed method by achieving a notable reduction of approximately 40% in parameters and 50% in computation compared to the state-of-the-art real-time object detector. The outcome includes faster inference speed and heightened detection accuracy.

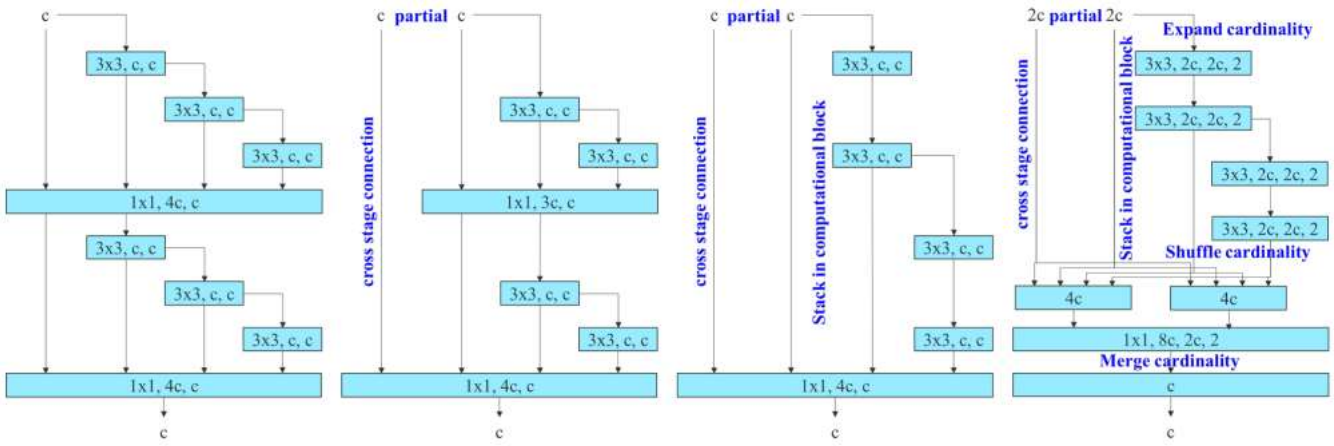


Figure 2: Extended efficient layer aggregation networks. The proposed extended ELAN (E-ELAN) does not change the gradient transmission path of the original architecture at all, but use group convolution to increase the cardinality of the added features, and combine the features of different groups in a shuffle and merge cardinality manner. This way of operation can enhance the features learned by different feature maps and improve the use of parameters and calculations.

II.RELATEDWORK

2.1Real-timeobjectdetectors

Currently, leading real-time object detectors rely on YOLO and FCOS, known for their speed and effectiveness [61, 62, 63, 76, 77]. Achieving state-of-the-art status involves key traits: (1) a faster and stronger network; (2) improved feature integration [22, 97, 37, 74, 59, 30, 9, 45]; (3) precise detection methods [76, 77, 69]; (4) robust loss functions [96, 64, 6, 56, 95, 57]; (5) efficient label assignment [99, 20, 17, 82, 42]; and (6) streamlined training. This paper focuses on addressing issues related to (4), (5), and (6) without exploring additional data or large models, proposing a new trainable bag-of-freebies method.

2.2Modelre-parameterization

Model re-parametrization techniques [71, 31, 75, 19, 33, 11, 4, 24, 13, 12, 10, 29, 14, 78] consolidate computational modules during inference, akin to ensemble methods. They fall into two categories: module-level ensemble and model-level ensemble. Model-level re-parameterization typically involves training identical models with different data or averaging weights across models. Module-level re-parameterization, a more recent focus, divides a module into branches during training, later merging them for inference. Our approach introduces a new re-parameterization module with application strategies tailored to diverse architectures.

2.3. Model scaling

During grouped convolutions, homogenous operations are carried out individually by each branch, and the outputs are Model scaling [72, 60, 74, 73, 15, 16, 2, 51] adjusts the size of a pre-designed model to fit various computing devices. This involves modifying scaling factors like resolution, depth, width, and stage to balance parameters, computation, inference speed, and accuracy. Network architecture search (NAS) is a common method, automatically finding suitable scaling factors. However, NAS can be computationally expensive. In [15], the researcher explores the relationship between scaling factors, parameters, and operations, aiming to estimate rules directly. Most model scaling methods analyze individual scaling factors independently, even in compound scaling methods. For our concatenation-based architecture, akin to DenseNet or VoVNet, a new compound scaling approach is necessary, considering the interdependence of scaling factors.

III.ARCHITECTURE

The YOLOv7 architecture is based on previous YOLO model architectures, namely YOLOv4, Scaled YOLOv4, and YOLO-R.

1.Efficient Layer Aggregation Networks (E-ELAN)

Efficient architecture design typically focuses on parameters, computation, and computational density considered memory access cost, analyzing the impact of input/output channel ratio, branches, and element-wise operations on inference speedincorporated activation in model scaling, emphasizing the output tensor elements. CSPVoVNet [79] addressed gradient paths for diverse feature learning. ELAN [1] controlled gradient paths for effective learning in deep networks. We propose Extended-ELAN (E-ELAN), shown in Figure 2 (d), expanding learning ability without disrupting the gradient path. E-ELAN utilizes group convolution to modify computational blocks, maintaining the original ELAN architecture. It employs expand, shuffle, merge cardinality to enhance learning across computational block groups. This strategy, applied within transition layers, ensures continual improvement in network learning ability without compromising stability.

3.2. Model scaling for concatenation-based models

Model scaling aims to adjust model attributes for different inference speeds. EfficientNet [72] scales width, depth, and resolution, while scaled-YOLOv4 [79] adjusts the number of stages. analyzed vanilla and group convolution impact on parameters and computation during width and depth scaling, applying it to architectures like PlainNet or ResNet. However, concatenation-based models, like the one shown in Figure 3 (a)

and (b), exhibit changes in in-degree and out-degree during depth scaling, complicating independent analysis of scaling factors.

For concatenation-based models, a compound scaling method is essential. Scaling depth affects the ratio of input to output channels in a transition layer, impacting hardware usage. Our proposed method synchronizes depth scaling with width scaling on transition layers (Figure 3c), maintaining the model's original design properties and optimal structure. This compound approach ensures a holistic consideration of scaling factors in concatenation-based architectures, preserving model integrity during upscaling or downscaling.

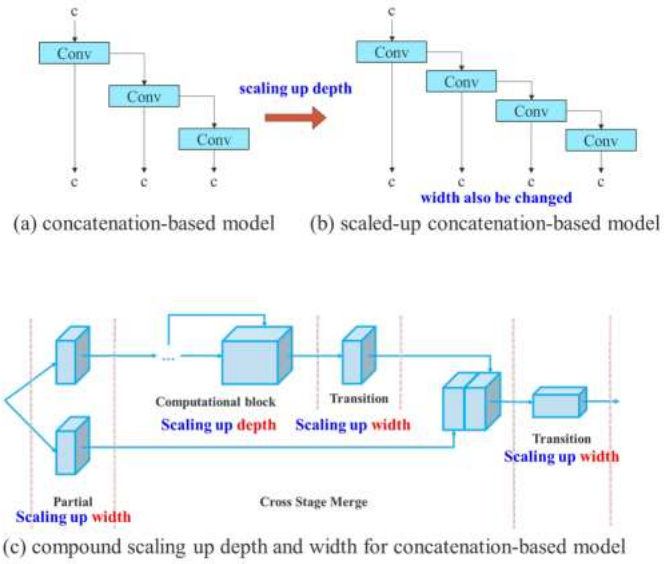
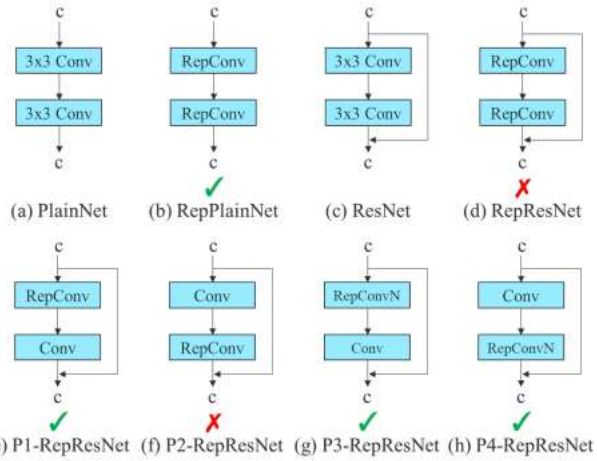


Figure 3: Model scaling for concatenation-based models. From (a) to (b), we observe that when depth scaling is performed on concatenation-based models, the output width of a computational block also increases. This phenomenon will cause the input width of the subsequent transmission layer to increase. Therefore, we propose (c), that is, when performing model scaling on concatenation-based models, only the depth in a computational block needs to be scaled, and the remaining of transmission layer is performed with corresponding width scaling.

4. Trainable bag-of-freebies

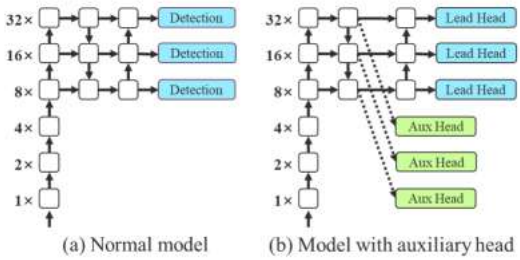
4.1. Planned re-parameterized convolution

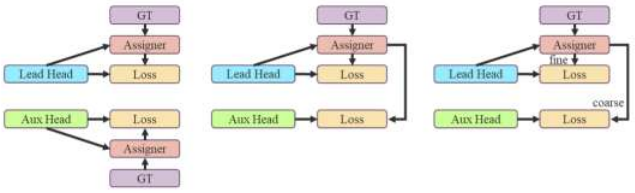
While RepConv [13] excels with VGG [68], its performance drops significantly when applied directly to ResNet [26] and DenseNet [32]. We leverage gradient flow propagation paths to analyze optimal integration of re-parameterized convolution with diverse networks. After scrutinizing RepConv's combination with different architectures, we identify that RepConv's identity connection disrupts ResNet's residual and DenseNet's concatenation, diminishing gradient diversity. To address this, we utilize RepConv without the identity connection (RepConvN) for our planned re-parameterized convolution architecture. When replacing a convolutional layer with residual or concatenation, we advocate omitting the identity connection. Figure 4 illustrates our designed "planned re-parameterized convolution" in PlainNet and ResNet. A comprehensive experiment on planned re-parameterized convolution in residual-based and concatenation-based models will be presented in the ablation study session.



4.2. Coarse for auxiliary and fine for lead loss

Deep supervision involves adding auxiliary heads to guide training in deep networks, notably enhancing performance even for well-converging architectures like ResNet and DenseNet. Label assignment, traditionally based on ground truth, has evolved in recent years, with methods like YOLO using network predictions to generate soft labels. In the context of deep supervision, a novel issue arises: "How to assign soft labels to both auxiliary and lead heads?" Current methods separate them for individual label assignment, but our proposed approach uses the lead head prediction to guide both heads, creating a coarse-to-fine hierarchical label assignment for improved learning. This strategy, depicted in Figure 5 (d) and (e), is a new and unexplored aspect in the literature, offering a promising direction for deep supervision techniques.





(c) Independent assigner (d) Lead guided assigner (e) Coarse-to-fine lead guided assigner

The lead head guided label assigner utilizes the prediction result of the lead head and the ground truth to generate soft labels through an optimization process, serving as the target training model for both auxiliary and lead heads. This approach leverages the strong learning capability of the lead head to create more representative soft labels, considering the distribution and correlation between source and target data. The learning process resembles a form of generalized residual learning, where the auxiliary head directly learns information from the lead head, enabling the latter to focus on residual information. The coarse-to-fine lead head guided label assigner generates two sets of soft labels—coarse and fine labels—by relaxing constraints for positive sample assignment. This addresses the weaker learning ability of the auxiliary head, optimizing its recall in object detection tasks. The final output filters high precision results from high recall results, ensuring the hierarchical labels dynamically adjust during the learning process, with the optimizable upper bound of fine labels consistently higher than coarse labels.

IV.EXPERIMENTS

5.1. Experimental setup

We conducted experiments and validated our object detection method using the Microsoft COCO dataset. All models were trained from scratch without pre-trained models. The train 2017 set was used for training, and the val 2017 set for verification and hyperparameter tuning. The performance was evaluated on the test 2017 set, comparing it with state-of-the-art object detection algorithms. Basic models were designed for edge GPU (YOLOv7-tiny), normal GPU (YOLOv7), and cloud GPU (YOLOv7-W6). Model scaling was applied for different service requirements, resulting in YOLOv7-X with stack scaling on the neck and compound scaling for depth and width. For YOLOv7-W6, the proposed compound scaling method yielded YOLOv7-E6 and YOLOv7-D6. EELAN was used for YOLOv7-E6, completing YOLOv7-E6E. YOLOv7-tiny, designed for edge GPUs, used leaky ReLU as the activation function, while SiLU was employed for other models. Scaling factors for each model are detailed in the Appendix. All things considered, the protocols for the experiments were carefully thought out and carried out to ensure impartiality, consistency, and an even playing field for all the models that were evaluated for the study. The presented comparative metrics and accuracy rates offer a solid foundation for assessing and comprehending the performance differences amongst the various architectures investigated in the ImageNet-1K classification framework.

We selected the previous version of YOLO [3, 79] and the state-of-the-art YOLOR [81] as baselines for comparison with our proposed YOLOv7 models trained under the same settings, as shown in Table 1. YOLOv7 demonstrates significant improvements over YOLOv4, with 75% fewer parameters, 36% less computation, and a 1.5% higher AP. Compared to YOLOR-CSP, YOLOv7 boasts a 43% reduction in parameters, 15% less computation, and a 0.4% higher AP. For the tiny model, YOLOv7-tiny achieves a 39% reduction in parameters and 49% less computation compared to YOLOv4-tiny-31, while maintaining the same AP. On the cloud GPU model, our approach achieves a higher AP, reducing parameters by 19% and computation by 33%.

Table 2: Comparison of state-of-the-art real-time object detectors.

Model	#Param.	FLOPs	Size	FPS	AP^{test} / AP^{val}	AP^{test}_{50}	AP^{test}_{75}	AP^{test}_{S}	AP^{test}_{M}	AP^{test}_{L}
YOLOX-S [21]	9.0M	26.8G	640	102	40.5% / 40.5%	-	-	-	-	-
YOLOX-M [21]	25.3M	73.8G	640	81	47.2% / 46.9%	-	-	-	-	-
YOLOX-L [21]	54.2M	155.6G	640	69	50.1% / 49.7%	-	-	-	-	-
YOLOX-X [21]	99.1M	281.9G	640	58	51.5% / 51.1%	-	-	-	-	-
PPYOLOE-S [85]	7.9M	17.4G	640	208	43.1% / 42.7%	60.5%	46.6%	23.2%	46.4%	56.9%
PPYOLOE-M [85]	23.4M	49.9G	640	123	48.9% / 48.6%	66.5%	53.0%	28.6%	52.9%	63.8%
PPYOLOE-L [85]	52.2M	110.1G	640	78	51.4% / 50.9%	68.9%	55.6%	31.4%	55.3%	66.1%
PPYOLOE-X [85]	98.4M	206.6G	640	45	52.2% / 51.9%	69.9%	56.5%	33.3%	56.3%	66.4%
YOLOv5-N (r6.1) [23]	1.9M	4.5G	640	159	- / 28.0%	-	-	-	-	-
YOLOv5-S (r6.1) [23]	7.2M	16.5G	640	156	- / 37.4%	-	-	-	-	-
YOLOv5-M (r6.1) [23]	21.2M	49.0G	640	122	- / 45.4%	-	-	-	-	-
YOLOv5-L (r6.1) [23]	46.5M	109.1G	640	99	- / 49.0%	-	-	-	-	-
YOLOv5-X (r6.1) [23]	86.7M	205.7G	640	83	- / 50.7%	-	-	-	-	-
YOLOR-CSP [81]	52.9M	120.4G	640	106	51.1% / 50.8%	69.6%	55.7%	31.7%	55.3%	64.7%
YOLOR-CSP-X [81]	96.9M	226.8G	640	87	53.0% / 52.7%	71.4%	57.9%	33.7%	57.1%	66.8%
YOLOv7-tiny-SiLU	6.2M	13.8G	640	286	38.7% / 38.7%	56.7%	41.7%	18.8%	42.4%	51.9%
YOLOv7	36.9M	104.7G	640	161	51.4% / 51.2%	69.7%	55.9%	31.8%	55.5%	65.0%
YOLOv7-X	71.3M	189.9G	640	114	53.1% / 52.9%	71.2%	57.8%	33.8%	57.1%	67.4%
YOLOv5-N6 (r6.1) [23]	3.2M	18.4G	1280	123	- / 36.0%	-	-	-	-	-
YOLOv5-S6 (r6.1) [23]	12.6M	67.2G	1280	122	- / 44.8%	-	-	-	-	-
YOLOv5-M6 (r6.1) [23]	35.7M	200.0G	1280	90	- / 51.3%	-	-	-	-	-
YOLOv5-L6 (r6.1) [23]	76.8M	445.6G	1280	63	- / 53.7%	-	-	-	-	-
YOLOv5-X6 (r6.1) [23]	140.7M	839.2G	1280	38	- / 55.0%	-	-	-	-	-
YOLOR-P6 [81]	37.2M	325.6G	1280	76	53.9% / 53.5%	71.4%	58.9%	36.1%	57.7%	65.6%
YOLOR-W6 [81]	79.8G	453.2G	1280	66	55.2% / 54.8%	72.7%	60.5%	37.7%	59.1%	67.1%
YOLOR-E6 [81]	115.8M	683.2G	1280	45	55.8% / 55.7%	73.4%	61.1%	38.4%	59.7%	67.7%
YOLOR-D6 [81]	151.7M	935.6G	1280	34	56.5% / 56.1%	74.1%	61.9%	38.9%	60.4%	68.7%
YOLOv7-W6	70.4M	360.0G	1280	84	54.9% / 54.6%	72.6%	60.1%	37.3%	58.7%	67.1%
YOLOv7-E6	97.2M	515.2G	1280	56	56.0% / 55.9%	73.5%	61.2%	38.0%	59.9%	68.4%
YOLOv7-D6	154.7M	806.8G	1280	44	56.6% / 56.3%	74.0%	61.8%	38.8%	60.1%	69.5%
YOLOv7-E6E	151.7M	843.2G	1280	36	56.8% / 56.8%	74.4%	62.1%	39.3%	60.5%	69.0%

[1] Our FLOPs is calaculated by rectangle input resolution like 640 × 640 or 1280 × 1280.
[2] Our inference time is estimated by using letterbox resize input image to make its long side equals to 640 or 1280.

5.3. Comparison with state-of-the-arts

Comparing our proposed method with state-of-the-art object detectors for general GPUs and Mobile GPUs in Table 2, our approach achieves the best speed-accuracy trade-off. YOLOv7-tiny-SiLU outperforms YOLOv5-N (r6.1) by being 127 fps faster and 10.7% more accurate on AP. YOLOv7 at 161 fps achieves a 51.4% AP, surpassing PPYOLOE-L, which achieves the same AP at only 78 fps. YOLOv7 has a 41% reduction in parameters compared to PPYOLOE-L. YOLOv7-X, with 114 fps inference speed, outperforms YOLOv5-L (r6.1) by improving AP by 3.9%. Compared to YOLOv5-X (r6.1), YOLOv7-X is 31 fps faster while reducing parameters and computation by 22% and 8%, respectively, and improving AP by 2.2%. YOLOv7-W6, at a resolution of 1280, is 8 fps faster and achieves a 1% higher AP than YOLOR-P6. YOLOv7-E6, in comparison with YOLOv5-X6 (r6.1), gains 0.9% AP, reduces parameters by 45%, computation by 63%, and increases inference speed by 47%. YOLOv7-D6 matches the inference speed of YOLOR-E6 but improves AP by

0.8%. YOLOv7-E6E matches the inference speed of YOLOR-D6 but improves AP by 0.3%.

5.4. Ablation study

Table 3 presents results using different model scaling strategies for scaling up. Our proposed compound scaling method involves increasing the depth of computational blocks by 1.5 times and the width of transition blocks by 1.25 times. Compared to methods solely scaling up the width, our approach improves AP by 0.5% with fewer parameters and less computation. In comparison to methods only scaling up the depth, our method requires a mere 2.9% increase in parameters and a 1.2% increase in computation, resulting in a 0.2% improvement in AP. The results demonstrate that our proposed compound scaling strategy efficiently utilizes parameters and computation.

Table 3: Ablation study on proposed model scaling.

Model	#Param.	FLOPs	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
base (v7-X light)	47.0M	125.5G	640	51.7%	70.1%	56.0%
width only (1.25 w)	73.4M	195.5G	640	52.4%	70.9%	57.1%
depth only (2.0 d)	69.3M	187.6G	640	52.7%	70.8%	57.3%
compound (v7-X)	71.3M	189.9G	640	**52.9%**	**71.1%**	**57.5%**
improvement	-	-	-	+1.2	+1.0	+1.5

5.4.2 Proposed planned re-parameterized model

To validate the general applicability of our proposed planned re-parameterized model, we applied it to both concatenation-based and residual-based models, specifically 3-stacked ELAN and CSPDarknet, respectively. In the concatenation-based model experiment, we replaced 3 × 3 convolutional layers at various positions in 3-stacked ELAN with RepConv, as detailed in Figure 6. Table 4 demonstrates consistently higher AP values with our proposed planned re-parameterized model. For the residual-based model experiment, as the original dark block lacked a 3 × 3 convolution block aligning with our design strategy, we introduced a reversed dark block, as depicted in Figure 7. Comparing CSPDarknet with the dark block and reversed dark block, having identical parameters and operations, confirmed the effectiveness of our proposed planned re-parameterized model, as shown in Table 5. Additionally, RepCSPResNet [85] adheres to our design pattern, affirming its compatibility with our approach.

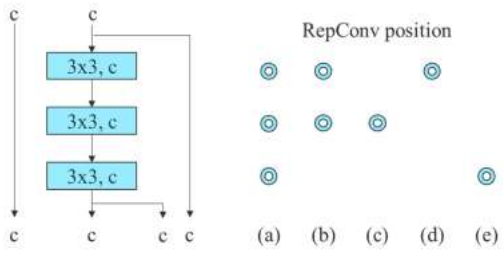


Figure 6: Planned RepConv 3-stacked ELAN. Blue circles are the position we replace Conv by RepConv.

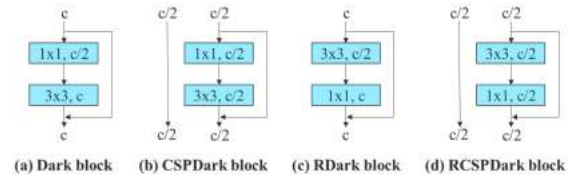


Figure 7: Reversed CSPDarknet. We reverse the position of 1×1 and 3×3 convolutional layer in dark block to fit our planned re-parameterized model design strategy.

Table 5: Ablation study on planned RepResidual model.

Model	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
base (YOLOR-W6)	54.82%	72.39%	59.95%	39.68%	59.38%	**68.30%**
RepCSP	54.67%	72.50%	59.58%	40.22%	**59.61%**	67.87%
RCSP	54.36%	71.95%	59.54%	40.15%	59.02%	67.44%
RepRCSP	**54.85%**	**72.51%**	**60.08%**	**40.53%**	59.52%	68.06%
base (YOLOR-CSP)	50.81%	69.47%	55.28%	33.74%	**56.01%**	65.38%
RepRCSP	**50.91%**	**69.54%**	**55.55%**	**34.44%**	55.74%	**65.46%**

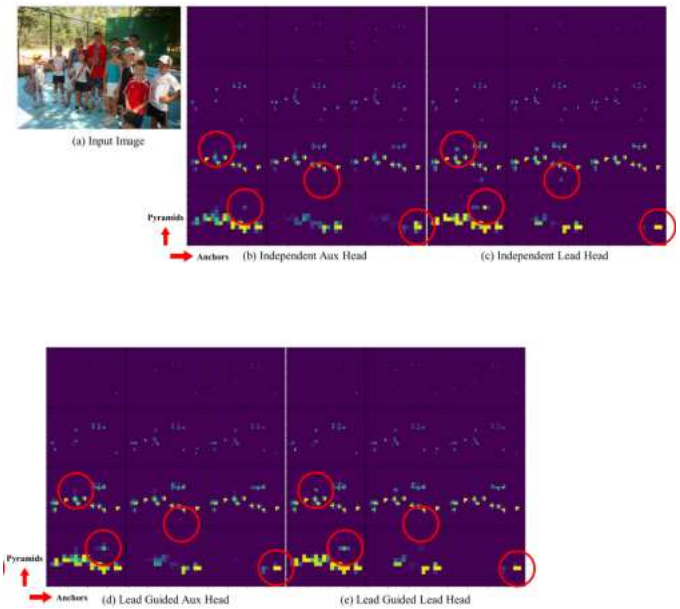


Figure 8: Objectness map predicted by different methods at auxiliary head and lead head.

5.4.3 Proposed assistant loss for auxiliary head

In the experiments involving assistant loss for the auxiliary head, we conducted comparisons between the general independent label assignment methods for the lead head and auxiliary head, along with a comparison of the two proposed lead-guided label assignment methods. The results, presented in Table 6, highlight the significant performance improvement achieved by any model incorporating assistant loss. Notably, our proposed lead-guided label assignment strategy outperforms the general independent label assignment strategy across AP, AP50, and AP75 metrics. Specifically, the proposed coarse label for the assistant and fine label for the lead assignment strategy yields the best overall results. Figure 8 visually depicts the objectness map predicted by different methods for the auxiliary head and lead head,

showcasing that when the auxiliary head learns from the lead-guided soft label, it effectively aids the lead head in extracting residual information from consistent targets.

Table 6: Ablation study on proposed auxiliary head.

Model	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
base (v7-E6)	1280	55.6%	73.2%	60.7%
independent	1280	55.8%	73.4%	60.9%
lead guided	1280	55.9%	73.5%	61.0%
coarse-to-fine lead guided	1280	**55.9%**	**73.5%**	**61.1%**
improvement	-	+0.3	+0.3	+0.4

Table 7 provides a detailed analysis of the impact of the proposed coarse-to-fine lead-guided label assignment method on the decoder of the auxiliary head. Specifically, we compare the results with and without the introduction of the upper bound constraint. The numbers in the table clearly indicate that constraining the upper bound of objectness based on the distance from the center of the object leads to improved performance.

Table 7: Ablation study on constrained auxiliary head.

Model	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
base (v7-E6)	1280	55.6%	73.2%	60.7%
aux without constraint	1280	55.9%	73.5%	61.0%
aux with constraint	1280	**55.9%**	**73.5%**	**61.1%**
improvement	-	+0.3	+0.3	+0.4

Given that YOLOv7 utilizes multiple pyramids for joint object detection predictions, we establish a direct connection between the auxiliary head and the pyramid in the middle layer during training. This training approach compensates for potential information loss in subsequent level pyramid predictions. To address this, our proposed E-ELAN architecture incorporates a partial auxiliary head. We strategically connect the auxiliary head after one set of feature maps before merging cardinality, ensuring the newly generated set of feature maps' weights are not directly updated by the assistant loss. This design allows each pyramid of the lead head to retain information from objects of varying sizes. Table 8 compares the results of two methods: coarse-to-fine lead-guided and partial coarse-to-fine lead-guided. Clearly, the partial coarse-to-fine lead-guided method exhibits superior auxiliary effects.

Table 8: Ablation study on partial auxiliary head.

Model	Size	AP^{val}	AP^{val}_{50}	AP^{val}_{75}
base (v7-E6E)	1280	56.3%	74.0%	61.5%
aux	1280	56.5%	74.0%	61.6%
partial aux	1280	**56.8%**	**74.4%**	**62.1%**
improvement	-	+0.5	+0.4	+0.6

V.OUR CONTRIBUTIONS

1. Trash detector

A system designed to automatically identify trash in photographs taken in public spaces. The objective of the project is to develop an effective solution for trash detection, employing the YOLOV7 technique.

Methodology:

We used the renowned and highly efficient YOLOv7 model for object detection in our project, specifically focusing on identifying trash in public spaces. The extensive training of the YOLOv7 model was conducted using a comprehensive TRASH dataset, comprising approximately 6,000 images depicting various types of garbage, all standardized to a size of 640x640 pixels. The primary objective was to utilize the trained YOLOv7 model to detect whether an object in an image is classified as trash or not. YOLOv7 stands out due to its impressive performance, leveraging features like E-ELAN and a sophisticated architecture.

We train the model for 45 epochs. Which gave us satisfactory results with the mAP of 0.583, where as Higher the value of mean Average Precision better the model.

Output:

The model achieved an initial training mAP of very low but after the successful iterations of 45 epochs we got mAP 0.58 and the Recall decreased with the epochs.

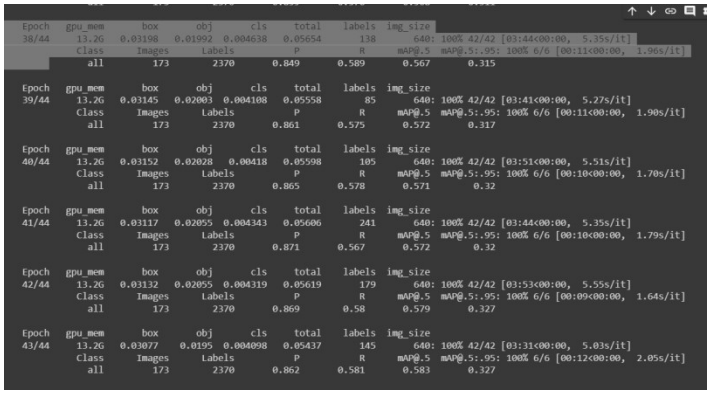


Figure 9: Training and Validation of mAP

The location for the stored weights of the trained model is provided as follows:
"/content/gdrive/MyDrive/yolov7/runs/train/exp/weights"
After each successful epoch, the weights will be saved with the filename "epoch040.pt," where 40 represents the epoch number.

2.Calculation of Average Precision by Testing Model:

We also computed the Mean Average Precision (mAP) during the testing phase of the model, using the PASCAL VOC dataset. It's worth noting that the YOLOV7 author employed the COCO dataset for training the model and determining the weights. The figure below illustrates the mAP calculation during testing on the VOC Pascal dataset.

```
    box       obj       cls     total    labels
0.03265   0.02121  0.004832    0.0587       241
   Images      Labels           P           R
      173        2370       0.848       0.583

    box       obj       cls     total    labels
0.03186   0.01948  0.004805   0.05615       142
   Images      Labels           P           R
      173        2370       0.844       0.582

    box       obj       cls     total    labels
0.03213   0.01995  0.004525    0.0566        91
   Images      Labels           P           R
      173        2370       0.859       0.576

    box       obj       cls     total    labels
```

Figure 10: Testing on VOC PASCAL dataset and Validation of mAP

$$P = \frac{T_p}{T_p + F_p}$$

$$P = \frac{T_p}{T_p + F_n}$$

$$F1 = 2\frac{PxR}{P+R}$$

Testing with a VOC Pascal dataset is giving mean Average Precision of 0.589.

2.F1 and Confidence Curve:

Below is the plotted graph of Confidence V/s F1 Metrics: Where blue, orange and green curve represents all classes, not trash and trash respectively.

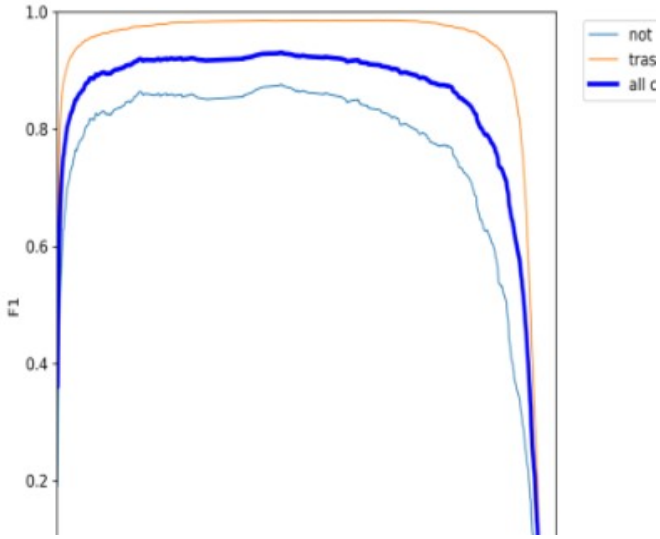


Figure 11 F1 v/s Confidence graph

The resulting graph can help you choose an appropriate confidence threshold that aligns with your application requirements. For example, if precision and recall are equally important, you may look for a point on the graph that maximizes the F1 score. Adjusting the threshold allows you to fine-tune the model's behavior based on the desired trade-off between precision and recall.

3.Precision and Recall Curve:

Figure no. 12 shows how precision and Recall behaving With each other.

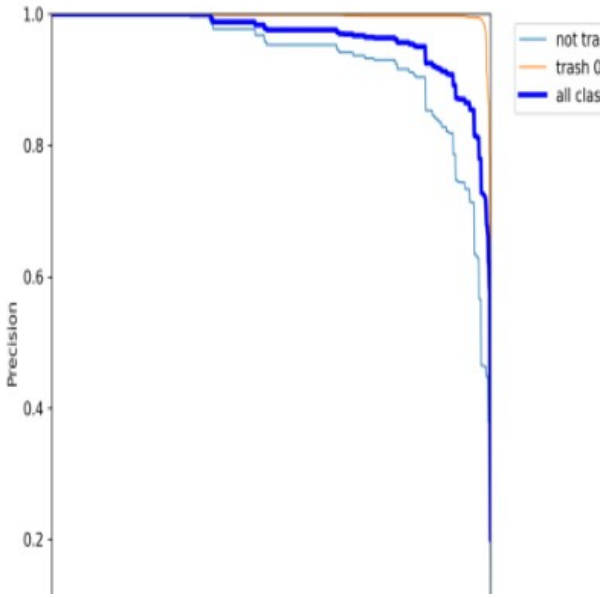


Figure 12 Precision v/s Recall graph

The Precision-Recall (PR) curve is a graphical representation of the trade-off between precision and recall for different thresholds in binary or multilabel classification problems. It provides insights into the model's performance across different levels of confidence or probability thresholds.

3.Output:

The results from the experiments demonstrate that the system successfully identifies trash within images with high accuracy. The model has shown strong performance on the validation set. Now, let's explore its functionality with some image examples.

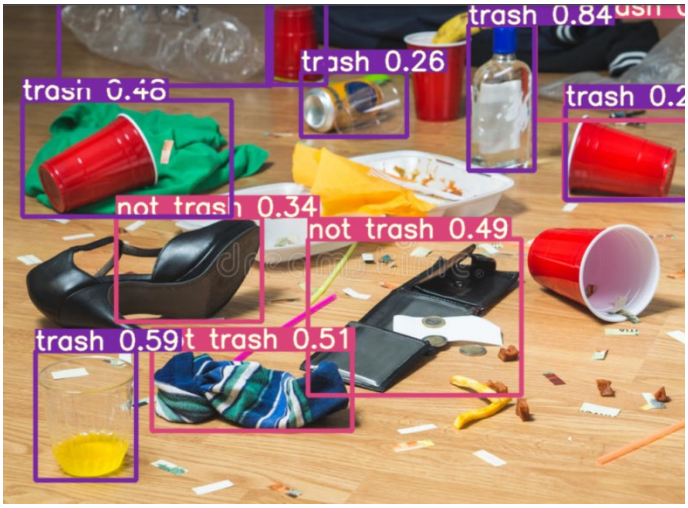
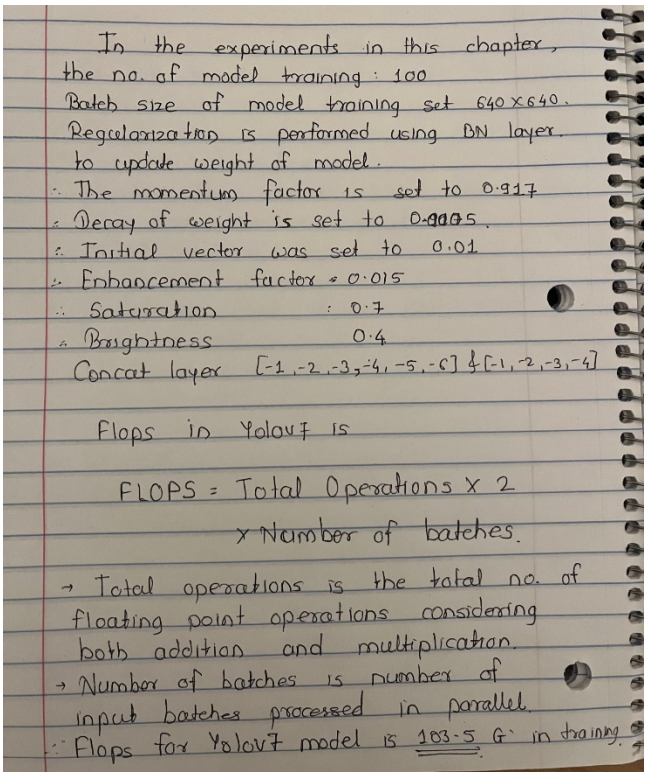


Figure 13 Trash detected

3.Calculation of flops:

FLOPS = Floating point operations per second.We use the term FLOPS to measure the number of operations of a frozen deep learning network.We calculated Flops for the dataset we tested .

The Params and FLOPs of YOLOv7 and RepVGG-YOLOv7 are calculated and the results are shown in Table 3. Based on the data shown in Table 3, it can be seen that the Params and FLOPs of RepVGG-YOLOv7 are smaller than those of YOLOv 7. On the one hand, the reason why the Params and FLOPs of RepVGG-YOLOv7 are slightly lower is that the convolutional and BN layers of RepVGG-YOLOv7 are fused during the inference process. On the other hand, the reason why the Params and FLOPs of RepVGG-YOLOv7 are reduced is that the reparameterization of the RepVGGBlock and the convolutional branches are fused. .


In the experiments in this chapter, the no. of model training : 100
Batch size of model training set 640 x 640.
Regularization is performed using BN layer.
to update weight of model.
∴ The momentum factor is set to 0.937
∴ Decay of weight is set to 0.0005.
∴ Initial vector was set to 0.01
∴ Enhancement factor = 0.015
∴ Saturation : 0.7
∴ Brightness 0.4
Concat layer [-1,-2,-3,-4,-5,-6] & [-1,-2,-3,-4]

Flops in Yolov7 is

FLOPS = Total Operations X 2
x Number of batches.

→ Total operations is the total no. of floating point operations considering both addition and multiplication.
→ Number of batches is number of input batches processed in parallel.
∴ Flops for Yolov7 model is 103.5 G in training.

Figure 14 Calculation of Flops

VI.EXPERIMENTALSETUP

Tools:

Anaconda
(Jupyter Notebook)
Google Colab

Libraries:

- Numpy
- Tensorflow
- PyTorch
- Keras
- Sklearn
- Tkinter
- Matplotlib
- Seaborn

VII.REFERENCEs

[1] https://arxiv.org/pdf/2207.02696.pdf

[2] https://github.com/WongKinYiu/yolov7

[3] https://viso.ai/deep-learning/yolov7-guide/