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

Chien-Yao Wang1, Alexey Bochkovskiy, and Hong-Yuan Mark Liao1 1Institute of Information Science, Academia Sinica, Taiwan

kinyiu@iis.sinica.edu.tw, alexeyab84@gmail.com, and liao@iis.sinica.edu.tw

**Abstract**

*YOLOv7 surpasses all known object detectors in both speed and accuracy in the range from 5 FPS to 160 FPS and has the highest accuracy 56.8% AP among all known real-time object detectors with 30 FPS or higher on GPU V100. YOLOv7-E6 object detector (56 FPS V100, 55.9% AP) outperforms both transformer-based detector SWIN-L Cascade-Mask R-CNN (9.2 FPS A100, 53.9% AP) by 509% in speed and 2% in accuracy, and convolutional-based detector ConvNeXt-XL Cascade-Mask R-CNN (8.6 FPS A100, 55.2% AP) by 551% in speed and 0.7% AP in accuracy, as well as YOLOv7 outperforms: YOLOR, YOLOX, Scaled-YOLOv4, YOLOv5, DETR, Deformable DETR, DINO-5scale-R50, ViT-Adapter-B and many other object detectors in speed and accuracy. Moreover, we train YOLOv7 only on MS COCO dataset from scratch without using any other datasets or pre-trained weights. Source code is released in <https://github.com/WongKinYiu/yolov7>.*

**1. Introduction**

Real-time object detection is a very important topic in computer vision, as it is often a necessary component in computer vision systems. For example, multi-object track-ing [94, 93], autonomous driving [40, 18], robotics [35, 58], medical image analysis [34, 46], etc. The computing de-vices that execute real-time object detection is usually some mobile CPU or GPU, as well as various neural processing units (NPU) developed by major manufacturers. For exam-ple, the Apple neural engine (Apple), the neural compute stick (Intel), Jetson AI edge devices (Nvidia), the edge TPU (Google), the neural processing engine (Qualcomm), the AI processing unit (MediaTek), and the AI SoCs (Kneron), are all NPUs. Some of the above mentioned edge devices focus on speeding up different operations such as vanilla convolu-tion, depth-wise convolution, or MLP operations. In this pa-per, the real-time object detector we proposed mainly hopes that it can support both mobile GPU and GPU devices from the edge to the cloud.

In recent years, the real-time object detector is still de-veloped for different edge device. For example, the devel-

1

Recently, model re-parameterization [13, 12, 29] and dy-namic label assignment [20, 17, 42] have become important topics in network training and object detection. Mainly af-ter the above new concepts are proposed, the training of object detector evolves many new issues. In this paper, we will present some of the new issues we have discovered and devise effective methods to address them. For model re-parameterization, we analyze the model re-parameterization strategies applicable to layers in different networks with the concept of gradient propagation path, and propose planned re-parameterized model. In addition, when we discover that with dynamic label assignment technology, the training of model with multiple output layers will generate new issues. That is: “How to assign dynamic targets for the outputs of different branches?” For this problem, we propose a new label assignment method called coarse-to-fine lead guided label assignment.

The contributions of this paper are summarized as fol-lows: (1) we design several trainable bag-of-freebies meth-ods, so that real-time object detection can greatly improve the detection accuracy without increasing the inference cost; (2) for the evolution of object detection methods, we found two new issues, namely how re-parameterized mod-ule replaces original module, and how dynamic label as-signment strategy deals with assignment to different output layers. In addition, we also propose methods to address the difficulties arising from these issues; (3) we propose “ex-tend” and “compound scaling” methods for the real-time object detector that can effectively utilize parameters and computation; and (4) the method we proposed can effec-tively reduce about 40% parameters and 50% computation of state-of-the-art real-time object detector, and has faster inference speed and higher detection accuracy.

**2. Related work**

**2.1. Real-time object detectors**

Currently state-of-the-art real-time object detectors are mainly based on YOLO [61, 62, 63] and FCOS [76, 77], which are [3, 79, 81, 21, 54, 85, 23]. Being able to become a state-of-the-art real-time object detector usually requires the following characteristics: (1) a faster and stronger net-work architecture; (2) a more effective feature integration method [22, 97, 37, 74, 59, 30, 9, 45]; (3) a more accurate detection method [76, 77, 69]; (4) a more robust loss func-tion [96, 64, 6, 56, 95, 57]; (5) a more efficient label assign-ment method [99, 20, 17, 82, 42]; and (6) a more efficient training method. In this paper, we do not intend to explore self-supervised learning or knowledge distillation methods that require additional data or large model. Instead, we will design new trainable bag-of-freebies method for the issues derived from the state-of-the-art methods associated with (4), (5), and (6) mentioned above.

2

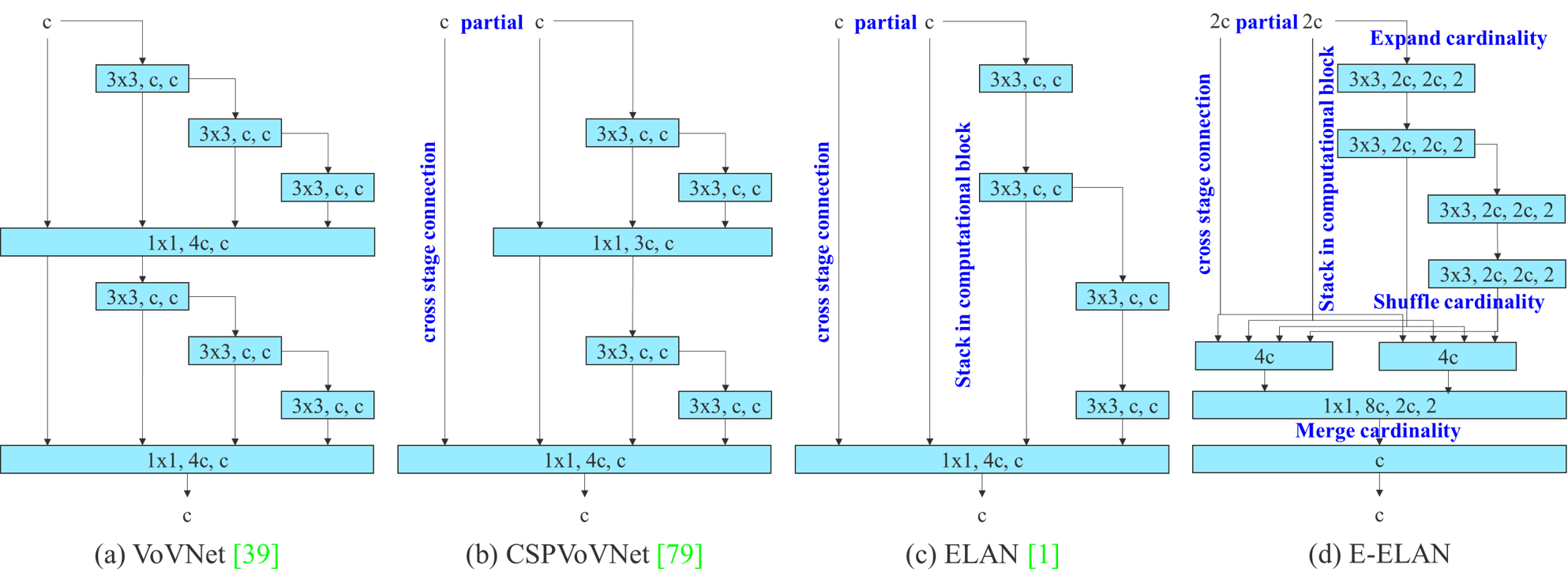


Figure 2: Extended efficient layer aggregation networks. The proposed extended ELAN (E-ELAN) does not change the gradient transmis-sion path of the original architecture at all, but use group convolution to increase the cardinality of the added features, and combine the features of different groups in 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.

**3. Architecture**

**3.1. Extended efficient layer aggregation networks**

In most of the literature on designing the efficient ar-chitectures, the main considerations are no more than the number of parameters, the amount of computation, and the computational density. Starting from the characteristics of memory access cost, Ma *et al*. [55] also analyzed the in-fluence of the input/output channel ratio, the number of branches of the architecture, and the element-wise opera-tion on the network inference speed. Doll´ar *et al*. [15] addi-tionally considered activation when performing model scal-ing, that is, to put more consideration on the number of el-ements in the output tensors of convolutional layers. The design of CSPVoVNet [79] in Figure 2 (b) is a variation of VoVNet [39]. In addition to considering the aforementioned basic designing concerns, the architecture of CSPVoVNet [79] also analyzes the gradient path, in order to enable the weights of different layers to learn more diverse features. The gradient analysis approach described above makes in-ferences faster and more accurate. ELAN [1] in Figure 2 (c) considers the following design strategy – “How to design an efficient network?.” They came out with a conclusion: By controlling the longest shortest gradient path, a deeper net-work can learn and converge effectively. In this paper, we propose Extended-ELAN (E-ELAN) based on ELAN and its main architecture is shown in Figure 2 (d).

Regardless of the gradient path length and the stacking number of computational blocks in large-scale ELAN, it has reached a stable state. If more computational blocks are stacked unlimitedly, this stable state may be destroyed, and the parameter utilization rate will decrease. The proposed

3

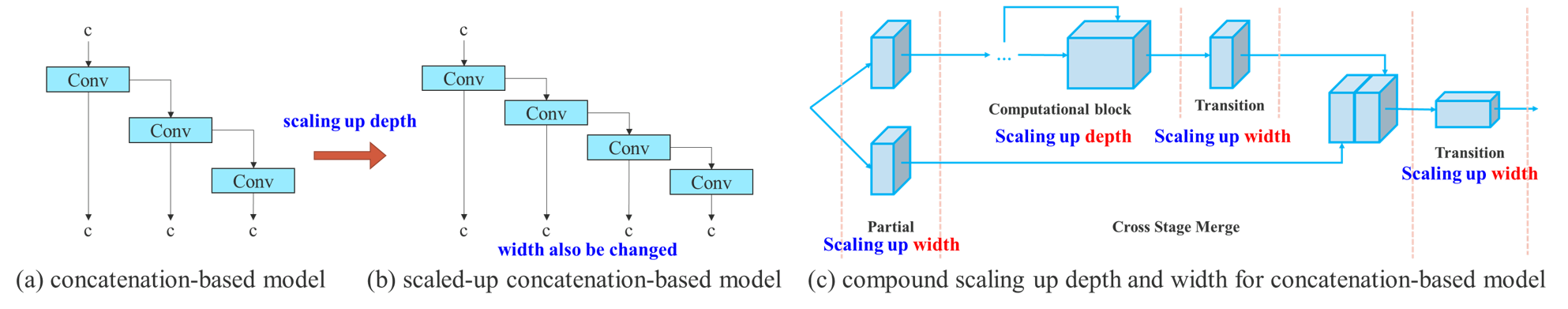


Figure 3: Model scaling for concatenation-based models. From (a) to (b), we observe that when depth scaling is performed on concatenation-based models, the output width of a computational block also increases. This phenomenon will cause the input width of the subsequent transmission layer to increase. 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.

The above methods are mainly used in architectures such as PlainNet or ResNet. When these architectures are in execut-ing scaling up or scaling down, the in-degree and out-degree of each layer will not change, so we can independently an-alyze the impact of each scaling factor on the amount of parameters and computation. However, if these methods are applied to the concatenation-based architecture, we will find that when scaling up or scaling down is performed on depth, the in-degree of a translation layer which is immedi-ately after a concatenation-based computational block will decrease or increase, as shown in Figure 3 (a) and (b).

It can be inferred from the above phenomenon that we cannot analyze different scaling factors separately for a concatenation-based model but must be considered to-gether. Take scaling-up depth as an example, such an ac-tion will cause a ratio change between the input channel and output channel of a transition layer, which may lead to a de-crease in the hardware usage of the model. Therefore, we must propose the corresponding compound model scaling method for a concatenation-based model. When we scale the depth factor of a computational block, we must also cal-culate the change of the output channel of that block. Then, we will perform width factor scaling with the same amount of change on the transition layers, and the result is shown in Figure 3 (c). Our proposed compound scaling method can maintain the properties that the model had at the initial design and maintains the optimal structure.

**4. Trainable bag-of-freebies**

**4.1. Planned re-parameterized convolution**

Although RepConv [13] has achieved excellent perfor-mance on the VGG [68], when we directly apply it to ResNet [26] and DenseNet [32] and other architectures, its accuracy will be significantly reduced. We use gradi-ent flow propagation paths to analyze how re-parameterized convolution should be combined with different network. We also designed planned re-parameterized convolution ac-cordingly.

4

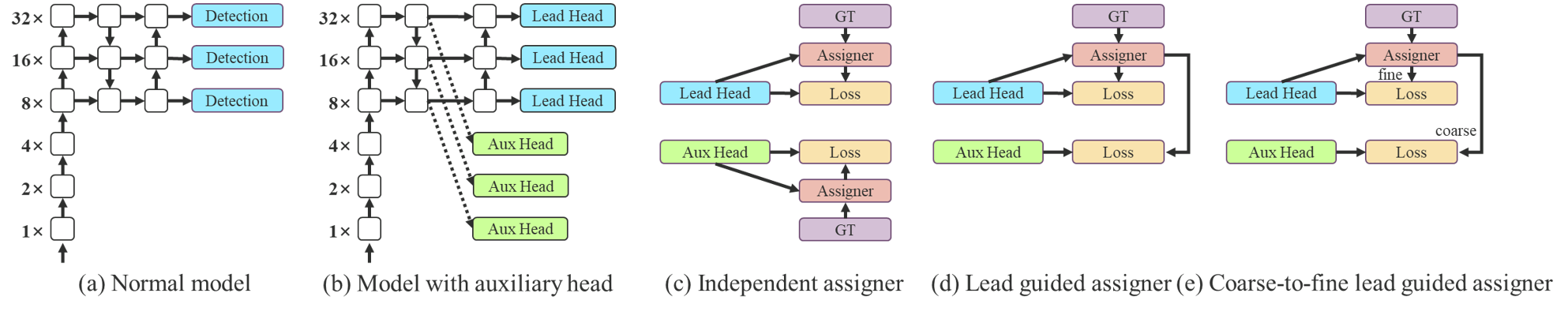


Figure 5: Coarse for auxiliary and fine for lead head label assigner. Compare with normal model (a), the schema in (b) has auxiliary head. Different from the usual independent label assigner (c), we propose (d) lead head guided label assigner and (e) coarse-to-fine lead head guided label assigner. The proposed label assigner is optimized by lead head prediction and the ground truth to get the labels of training lead head and auxiliary head at the same time. The detailed coarse-to-fine implementation method and constraint design details will be elaborated in Apendix.

**4.2. Coarse for auxiliary and fine for lead loss**

Deep supervision [38] is a technique that is often used in training deep networks. Its main concept is to add extra auxiliary head in the middle layers of the network, and the shallow network weights with assistant loss as the guide. Even for architectures such as ResNet [26] and DenseNet [32] which usually converge well, deep supervi-sion [70, 98, 67, 47, 82, 65, 86, 50] can still significantly improve the performance of the model on many tasks. Fig-ure 5 (a) and (b) show, respectively, the object detector ar-chitecture “without” and “with” deep supervision. In this paper, we call the head responsible for the final output as the lead head, and the head used to assist training is called auxiliary head.

Next we want to discuss the issue of label assignment. In the past, in the training of deep network, label assignment usually refers directly to the ground truth and generate hard label according to the given rules. However, in recent years, if we take object detection as an example, researchers often use the quality and distribution of prediction output by the network, and then consider together with the ground truth to use some calculation and optimization methods to generate a reliable soft label [61, 8, 36, 99, 91, 44, 43, 90, 20, 17, 42]. For example, YOLO [61] use IoU of prediction of bounding box regression and ground truth as the soft label of object-ness. In this paper, we call the mechanism that considers the network prediction results together with the ground truth and then assigns soft labels as “label assigner.”

Deep supervision needs to be trained on the target ob-jectives regardless of the circumstances of auxiliary head or lead head. During the development of soft label assigner re-lated techniques, we accidentally discovered a new deriva-tive issue, i.e., “How to assign soft label to auxiliary head and lead head ?” To the best of our knowledge, the relevant literature has not explored this issue so far. The results of the most popular method at present is as shown in Figure 5 (c), which is to separate auxiliary head and lead head, and then use their own prediction results and the ground truth

5

Table 1: Comparison of baseline object detectors.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **#Param.** | **FLOPs** | **Size** | **AP***val* | **AP***val* 50 | **AP***val* 75 | **AP***val S* | **AP***val M* | **AP***val L* |
| **YOLOv4 [3]** | 64.4M | 142.8G | 640 | 49.7% | 68.2% | 54.3% | 32.9% | 54.8% | 63.7% |
| **YOLOR-u5 (r6.1) [81]** | 46.5M | 109.1G | 640 | 50.2% | 68.7% | 54.6% | 33.2% | 55.5% | 63.7% |
| **YOLOv4-CSP [79]** | 52.9M | 120.4G | 640 | 50.3% | 68.6% | 54.9% | 34.2% | 55.6% | 65.1% |
| **YOLOR-CSP [81]** | 52.9M | 120.4G | 640 | 50.8% | 69.5% | 55.3% | 33.7% | 56.0% | 65.4% |
| **YOLOv7** | 36.9M | 104.7G | 640 | **51.2%** | **69.7%** | **55.5%** | **35.2%** | **56.0%** | **66.7%** |
| improvement | -43% | -15% | - | +0.4 | +0.2 | +0.2 | +1.5 | = | +1.3 |
| **YOLOR-CSP-X [81]** | 96.9M | 226.8G | 640 | 52.7% | **71.3%** | 57.4% | 36.3% | 57.5% | 68.3% |
| **YOLOv7-X** | 71.3M | 189.9G | 640 | **52.9%** | 71.1% | **57.5%** | **36.9%** | **57.7%** | **68.6%** |
| improvement | -36% | -19% | - | +0.2 | -0.2 | +0.1 | +0.6 | +0.2 | +0.3 |
| **YOLOv4-tiny [79]** | 6.1 | 6.9 | 416 | 24.9% | 42.1% | 25.7% | 8.7% | 28.4% | 39.2% |
| **YOLOv7-tiny** | 6.2 | 5.8 | 416 | **35.2%** | **52.8%** | **37.3%** | **15.7%** | **38.0%** | **53.4%** |
| improvement | +2% | -19% | - | +10.3 | +10.7 | +11.6 | +7.0 | +9.6 | +14.2 |
| **YOLOv4-tiny-3l [79]** | 8.7 | 5.2 | 320 | 30.8% | 47.3% | 32.2% | **10.9%** | 31.9% | 51.5% |
| **YOLOv7-tiny** | 6.2 | 3.5 | 320 | **30.8%** | **47.3%** | **32.2%** | 10.0% | **31.9%** | **52.2%** |
| improvement | -39% | -49% | - | = | = | = | -0.9 | = | +0.7 |
| **YOLOR-E6 [81]** | 115.8M | 683.2G | 1280 | 55.7% | 73.2% | 60.7% | 40.1% | **60.4%** | 69.2% |
| **YOLOv7-E6** | 97.2M | 515.2G | 1280 | **55.9%** | **73.5%** | **61.1%** | **40.6%** | 60.3% | **70.0%** |
| improvement | -19% | -33% | - | +0.2 | +0.3 | +0.4 | +0.5 | -0.1 | +0.8 |
| **YOLOR-D6 [81]** | 151.7M | 935.6G | 1280 | 56.1% | 73.9% | 61.2% | **42.4%** | 60.5% | 69.9% |
| **YOLOv7-D6** | 154.7M | 806.8G | 1280 | 56.3% | 73.8% | 61.4% | 41.3% | 60.6% | 70.1% |
| **YOLOv7-E6E** | 151.7M | 843.2G | 1280 | **56.8%** | **74.4%** | **62.1%** | 40.8% | **62.1%** | **70.6%** |
| improvement | = | -11% | - | +0.7 | +0.5 | +0.9 | -1.6 | +1.6 | +0.7 |

that of fine label, it may produce bad prior at final predic-tion. Therefore, in order to make those extra coarse positive grids have less impact, we put restrictions in the decoder, so that the extra coarse positive grids cannot produce soft label perfectly. The mechanism mentioned above allows the importance of fine label and coarse label to be dynam-ically adjusted during the learning process, and makes the optimizable upper bound of fine label always higher than coarse label.

**4.3. Other trainable bag-of-freebies**

In this section we will list some trainable bag-of-freebies. These freebies are some of the tricks we used in training, but the original concepts were not proposed by us. The training details of these freebies will be elab-orated in the Appendix, including (1) Batch normalization in conv-bn-activation topology: This part mainly connects batch normalization layer directly to convolutional layer. The purpose of this is to integrate the mean and variance of batch normalization into the bias and weight of convolu-tional layer at the inference stage. (2) Implicit knowledge in YOLOR [81] combined with convolution feature map in addition and multiplication manner: Implicit knowledge in YOLOR can be simplified to a vector by pre-computing at the inference stage. This vector can be combined with the bias and weight of the previous or subsequent convolutional layer. (3) EMA model: EMA is a technique used in mean teacher [75], and in our system we use EMA model purely as the final inference model.

6

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% 63.8% 66.1% 66.4% |
| **PPYOLOE-M [85]** | 23.4M | 49.9G | 640 | 123 | 48.9% / 48.6% | 66.5% | 53.0% | | 28.6% | | 52.9% |
| **PPYOLOE-L [85]** | 52.2M | 110.1G | 640 | 78 | 51.4% / 50.9% | 68.9% | 55.6% | | 31.4% | | 55.3% |
| **PPYOLOE-X [85]** | 98.4M | 206.6G | 640 | 45 | 52.2% / 51.9% | 69.9% | 56.5% | | 33.3% | | 56.3% |
| **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% 66.8% |
| **YOLOR-CSP-X [81]** | 96.9M | 226.8G | 640 | 87 | 53.0% / 52.7% | 71.4% | 57.9% | | 33.7% | | 57.1% |
| **YOLOv7-tiny-SiLU** | 6.2M | 13.8G | 640 | 286 | 38.7% / 38.7% | 56.7% | 41.7% | | 18.8% | | 42.4% | 51.9% 65.0% 67.4% |
| **YOLOv7** | 36.9M | 104.7G | 640 | 161 | 51.4% / 51.2% | 69.7% | 55.9% | | 31.8% | | 55.5% |
| **YOLOv7-X** | 71.3M | 189.9G | 640 | 114 | 53.1% / 52.9% | 71.2% | 57.8% | | 33.8% | | 57.1% |
| **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% 67.1% 67.7% 68.7% |
| **YOLOR-W6 [81]** | 79.8G | 453.2G | 1280 | 66 | 55.2% / 54.8% | 72.7% | 60.5% | | 37.7% | | 59.1% |
| **YOLOR-E6 [81]** | 115.8M | 683.2G | 1280 | 45 | 55.8% / 55.7% | 73.4% | 61.1% | | 38.4% | | 59.7% |
| **YOLOR-D6 [81]** | 151.7M | 935.6G | 1280 | 34 | 56.5% / 56.1% | 74.1% | 61.9% | | 38.9% | | 60.4% |
| **YOLOv7-W6** | 70.4M | 360.0G | 1280 | 84 | 54.9% / 54.6% | 72.6% | 60.1% | | 37.3% | | 58.7% | 67.1% 68.4% 69.5% 69.0% |
| **YOLOv7-E6** | 97.2M | 515.2G | 1280 | 56 | 56.0% / 55.9% | 73.5% | 61.2% | | 38.0% | | 59.9% |
| **YOLOv7-D6** | 154.7M | 806.8G | 1280 | 44 | 56.6% / 56.3% | 74.0% | 61.8% | | 38.8% | | 60.1% |
| **YOLOv7-E6E** | 151.7M | 843.2G | 1280 | 36 | 56.8% / 56.8% | 74.4% | 62.1% | | 39.3% | | 60.5% |
| 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. | | | | | | | | | | | |  |
| **5.2. Baselines** | **5.3. Comparison with state-of-the-arts** | | | | | | | | | | |

We choose previous version of YOLO [3, 79] and state-of-the-art object detector YOLOR [81] as our baselines. Ta-ble 1 shows the comparison of our proposed YOLOv7 mod-els and those baseline that are trained with the same settings.

From the results we see that if compared with YOLOv4, YOLOv7 has 75% less parameters, 36% less computation, and brings 1.5% higher AP. If compared with state-of-the-art YOLOR-CSP, YOLOv7 has 43% fewer parameters, 15% less computation, and 0.4% higher AP. In the performance of tiny model, compared with YOLOv4-tiny-31, YOLOv7-tiny reduces the number of parameters by 39% and the amount of computation by 49%, but maintains the same AP. On the cloud GPU model, our model can still have a higher AP while reducing the number of parameters by 19% and the amount of computation by 33%.

7

If we compare YOLOv7 with YOLOR using the input resolution 1280, the inference speed of YOLOv7-W6 is 8 fps faster than that of YOLOR-P6, and the detection rate is also increased by 1% AP. As for the comparison between YOLOv7-E6 and YOLOv5-X6 (r6.1), the former has 0.9% AP gain than the latter, 45% less parameters and 63% less computation, and the inference speed is increased by 47%. YOLOv7-D6 has close inference speed to YOLOR-E6, but improves AP by 0.8%. YOLOv7-E6E has close inference speed to YOLOR-D6, but improves AP by 0.3%.

**5.4. Ablation study**

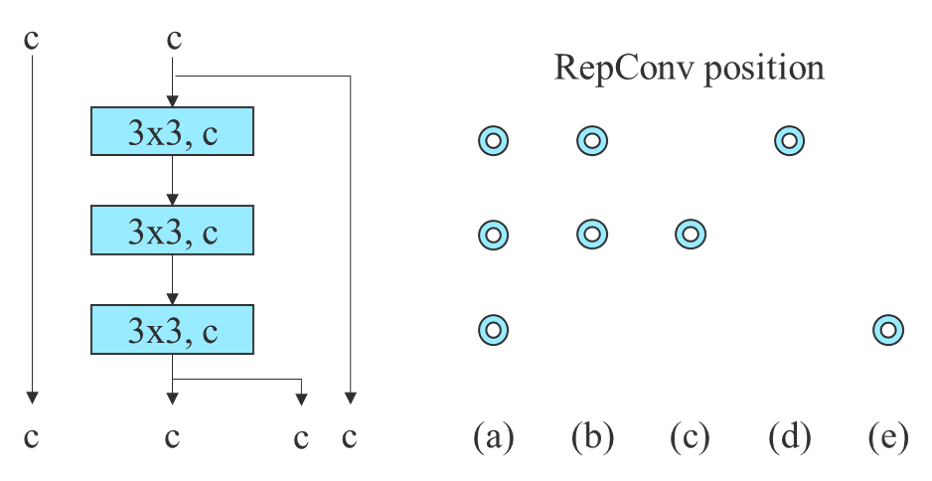


Figure 6: Planned RepConv 3-stacked ELAN. Blue circles are the position we replace Conv by RepConv.

|  |  |  |
| --- | --- | --- |
| **5.4.1** | **Proposed compound scaling method** | Table 4: Ablation study on planned RepConcatenation model. |

Table 3 shows the results obtained when using different model scaling strategies for scaling up. Among them, our proposed compound scaling method is to scale up the depth of computational block by 1.5 times and the width of tran-sition block by 1.25 times. If our method is compared with the method that only scaled up the width, our method can improve the AP by 0.5% with less parameters and amount of computation. If our method is compared with the method that only scales up the depth, our method only needs to in-crease the number of parameters by 2.9% and the amount of computation by 1.2%, which can improve the AP by 0.2%. It can be seen from the results of Table 3 that our proposed compound scaling strategy can utilize parameters and com-putation more efficiently.

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** | | | | | | |
| In order to verify the generality of our proposed planed re-parameterized model, we use it on concatenation-based model and residual-based model respectively for verifica- | | | | | | | |
| tion. | The concatenation-based model and residual-based | | | | | | |
| model we chose for verification are 3-stacked ELAN and CSPDarknet, respectively.  In the experiment of concatenation-based model, we re-place the 3 *×* 3 convolutional layers in different positions in 3-stacked ELAN with RepConv, and the detailed configura-tion is shown in Figure 6. From the results shown in Table 4 we see that all higher AP values are present on our proposed planned re-parameterized model. | | | | | | | |

In the experiment dealing with residual-based model, since the original dark block does not have a 3 *×* 3 con-

8

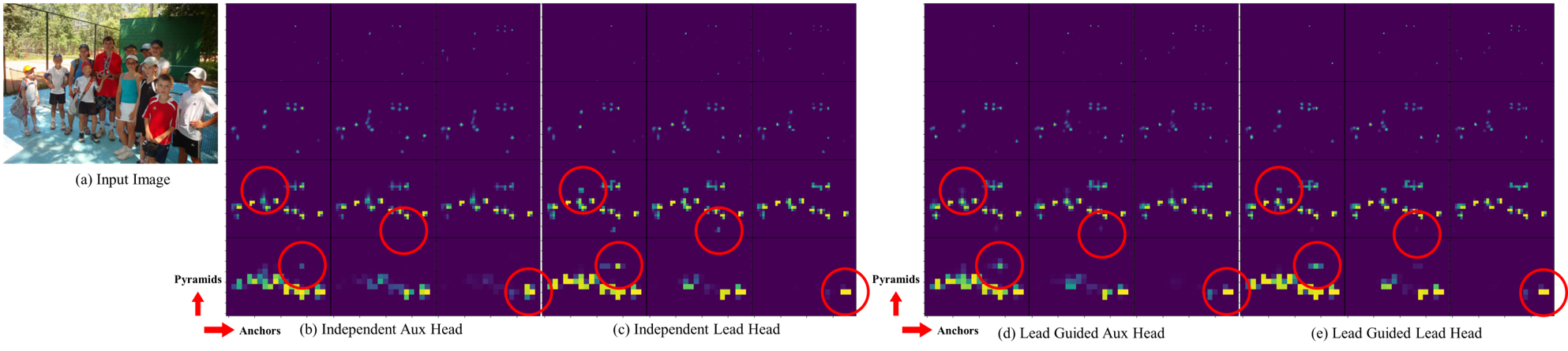


Figure 8: Objectness map predicted by different methods at auxiliary head and lead head.

**5.4.3**  **Proposed assistant loss for auxiliary head**

In the assistant loss for auxiliary head experiments, we com-pare the general independent label assignment for lead head and auxiliary head methods, and we also compare the two proposed lead guided label assignment methods. We show all comparison results in Table 6. From the results listed in Table 6, it is clear that any model that increases assistant loss can significantly improve the overall performance. In addition, our proposed lead guided label assignment strat-egy receives better performance than the general indepen-dent label assignment strategy in AP, AP50, and AP75. As for our proposed coarse for assistant and fine for lead label assignment strategy, it results in best results in all cases. In Figure 8 we show the objectness map predicted by different methods at auxiliary head and lead head. From Figure 8 we find that if auxiliary head learns lead guided soft label, it will indeed help lead head to extract the residual informa-tion from the consistant 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 |

In Table 7 we further analyze the effect of the proposed coarse-to-fine lead guided label assignment method on the decoder of auxiliary head. That is, we compared the results of with/without the introduction of upper bound constraint. Judging from the numbers in the Table, the method of con-straining the upper bound of objectness by the distance from the center of the object can achieve better 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 |

9

Table 9: More comparison (batch=1, no-TRT, without extra object detection training data)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **#Param.** | **FLOPs** | **Size** | **FPS***V* 100 | **AP***test*/ **AP***val* | **AP***test* 50 | **AP***test* 75 |
| **YOLOv7-tiny-SiLU** | 6.2M | 13.8G | 640 | 286 | **38.7%** / **38.7%** | **56.7%** | **41.7%** |
| **PPYOLOE-S [85]** | 7.9M | 17.4G | 640 | 208 | **43.1%** / **42.7%** | **60.5%** | **46.6%** |
| **YOLOv7** | 36.9M | 104.7G | 640 | 161 | **51.4%** / **51.2%** | **69.7%** | **55.9%** |
| **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% | - | - |
| **PPYOLOE-M [85]** | 23.4M | 49.9G | 640 | 123 | 48.9% / 48.6% | 66.5% | 53.0% |
| **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-M (r6.1) [23]** | 21.2M | 49.0G | 640 | 122 | - / 45.4% | - | - |
| **YOLOv7-X** | 71.3M | 189.9G | 640 | 114 | **53.1%** / **52.9%** | **71.2%** | **57.8%** |
| **YOLOR-CSP [81]** | 52.9M | 120.4G | 640 | 106 | 51.1% / 50.8% | 69.6% | 55.7% |
| **YOLOX-S [21]** | 9.0M | 26.8G | 640 | 102 | 40.5% / 40.5% | - | - |
| **YOLOv5-L (r6.1) [23]** | 46.5M | 109.1G | 640 | 99 | - / 49.0% | - | - |
| **YOLOv5-M6 (r6.1) [23]** | 35.7M | 200.0G | 1280 | 90 | - / 51.3% | - | - |
| **YOLOR-CSP-X [81]** | 96.9M | 226.8G | 640 | 87 | 53.0% / 52.7% | **71.4%** | **57.9%** |
| **YOLOv7-W6** | 70.4M | 360.0G | 1280 | 84 | **54.9%** / **54.6%** | **72.6%** | **60.1%** |
| **YOLOv5-X (r6.1) [23]** | 86.7M | 205.7G | 640 | 83 | - / 50.7% | - | - |
| **YOLOX-M [21]** | 25.3M | 73.8G | 640 | 81 | 47.2% / 46.9% | - | - |
| **PPYOLOE-L [85]** | 52.2M | 110.1G | 640 | 78 | 51.4% / 50.9% | 68.9% | 55.6% |
| **YOLOR-P6 [81]** | 37.2M | 325.6G | 1280 | 76 | 53.9% / 53.5% | 71.4% | 58.9% |
| **YOLOX-L [21]** | 54.2M | 155.6G | 640 | 69 | 50.1% / 49.7% | - | - |
| **YOLOR-W6 [81]** | 79.8G | 453.2G | 1280 | 66 | **55.2%** / **54.8%** | **72.7%** | **60.5%** |
| **YOLOv5-L6 (r6.1) [23]** | 76.8M | 445.6G | 1280 | 63 | - / 53.7% | - | - |
| **YOLOX-X [21]** | 99.1M | 281.9G | 640 | 58 | 51.5% / 51.1% | - | - |
| **YOLOv7-E6** | 97.2M | 515.2G | 1280 | 56 | **56.0%** / **55.9%** | **73.5%** | **61.2%** |
| **YOLOR-E6 [81]** | 115.8M | 683.2G | 1280 | 45 | 55.8% / 55.7% | 73.4% | 61.1% |
| **PPYOLOE-X [85]** | 98.4M | 206.6G | 640 | 45 | 52.2% / 51.9% | 69.9% | 56.5% |
| **YOLOv7-D6** | 154.7M | 806.8G | 1280 | 44 | **56.6%** / **56.3%** | **74.0%** | **61.8%** |
| **YOLOv5-X6 (r6.1) [23]** | 140.7M | 839.2G | 1280 | 38 | - / 55.0% | - | - |
| **YOLOv7-E6E** | 151.7M | 843.2G | 1280 | 36 | **56.8%** / **56.8%** | **74.4%** | **62.1%** |
| **YOLOR-D6 [81]** | 151.7M | 935.6G | 1280 | 34 | 56.5% / 56.1% | **74.1%** | **61.9%** |
| **F-RCNN-R101-FPN+ [5]** | 60.0M | 246.0G | 1333 | 20 | - / 44.0% | - | - |
| **Deformable DETR [100]** | 40.0M | 173.0G | - | 19 | - / 46.2% | - | - |
| **Swin-B (C-M-RCNN) [52]** | 145.0M | 982.0G | 1333 | 11.6 | - / 51.9% | - | - |
| **DETR DC5-R101 [5]** | 60.0M | 253.0G | 1333 | 10 | - / 44.9% | - | - |
| **EfficientDet-D7x [74]** | 77.0M | 410.0G | 1536 | 6.5 | 55.1% / 54.4% | 72.4% | 58.4% |
| **Dual-Swin-T (C-M-RCNN) [47]** | 113.8M | 836.0G | 1333 | 6.5 | - / 53.6% | - | - |
| **ViT-Adapter-B [7]** | 122.0M | 997.0G | - | 4.4 | - / 50.8% | - | - |
| **Dual-Swin-B (HTC) [47]** | 235.0M | - | 1600 | 2.5 | **58.7%** / **58.4%** | - | - |
| **Dual-Swin-L (HTC) [47]** | 453.0M | - | 1600 | 1.5 | **59.4%** / **59.1%** | - | - |
| **Model** | **#Param.** | **FLOPs** | **Size** | **FPS***A*100 | **AP***test*/ **AP***val* | **AP***test* 50 | **AP***test* 75 |
| **DN-Deformable-DETR [41]** | 48.0M | 265.0G | 1333 | 23.0 | - / 48.6% | - | - |
| **ConvNeXt-B (C-M-RCNN) [53]** | - | 964.0G | 1280 | 11.5 | - / 54.0% | 73.1% | 58.8% |
| **Swin-B (C-M-RCNN) [52]** | - | 982.0G | 1280 | 10.7 | - / 53.0% | 71.8% | 57.5% |
| **DINO-5scale (R50) [89]** | 47.0M | 860.0G | 1333 | 10.0 | - / 51.0% | - | - |
| **ConvNeXt-L (C-M-RCNN) [53]** | - | 1354.0G | 1280 | 10.0 | - / 54.8% | 73.8% | 59.8% |
| **Swin-L (C-M-RCNN) [52]** | - | 1382.0G | 1280 | 9.2 | - / 53.9% | 72.4% | 58.8% |
| **ConvNeXt-XL (C-M-RCNN) [53]** | - | 1898.0G | 1280 | 8.6 | - / 55.2% | 74.2% | 59.9% |

**8. More comparison**   
 YOLOv7 surpasses all known object detectors in both speed and accuracy in the range from 5 FPS to 160 FPS and has the highest accuracy 56.8% AP test-dev / 56.8% AP min-val among all known real-time object detectors with 30 FPS or higher on GPU V100. YOLOv7-E6 object detector (56 FPS V100, 55.9% AP) outperforms both transformer-based detector SWIN-L Cascade-Mask R-CNN (9.2 FPS A100, 53.9% AP) by 509% in speed and 2% in accuracy,

10