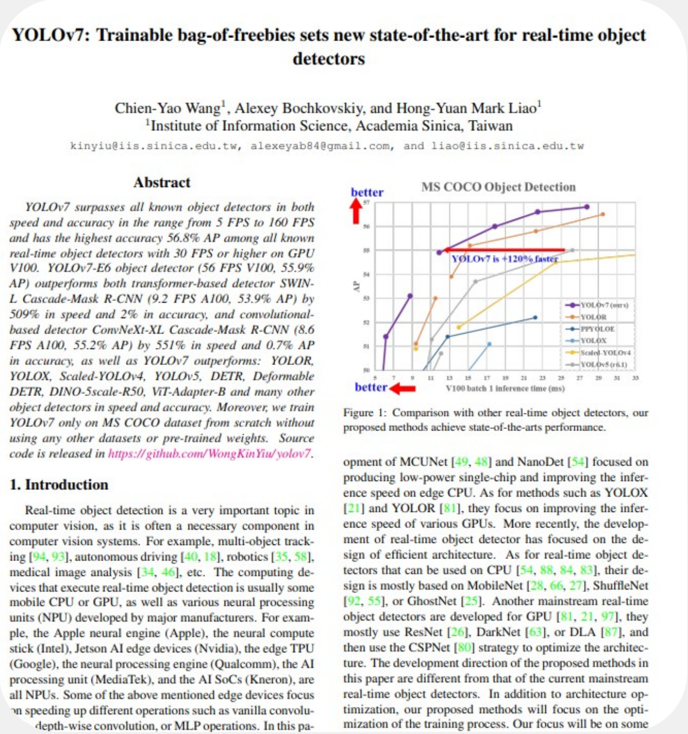
YOLOv7

Paper Review

Introduction



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

Chien-Yao Wang[1], Alexey Bochkovskiy, and Hong-Yuan Mark Liao[1]
[1]Institute 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.

MS COCO Object Detection

better

YOLOv7 is +120% faster

AP

V100 batch 1 inference time (ms)

YOLOv7 (ours)
YOLOR
PPYOLOE
YOLOX
Scaled-YOLOv4
YOLOv5 (r6.1)

Figure 1: Comparison with other real-time object detectors, our proposed methods achieve state-of-the-arts performance.

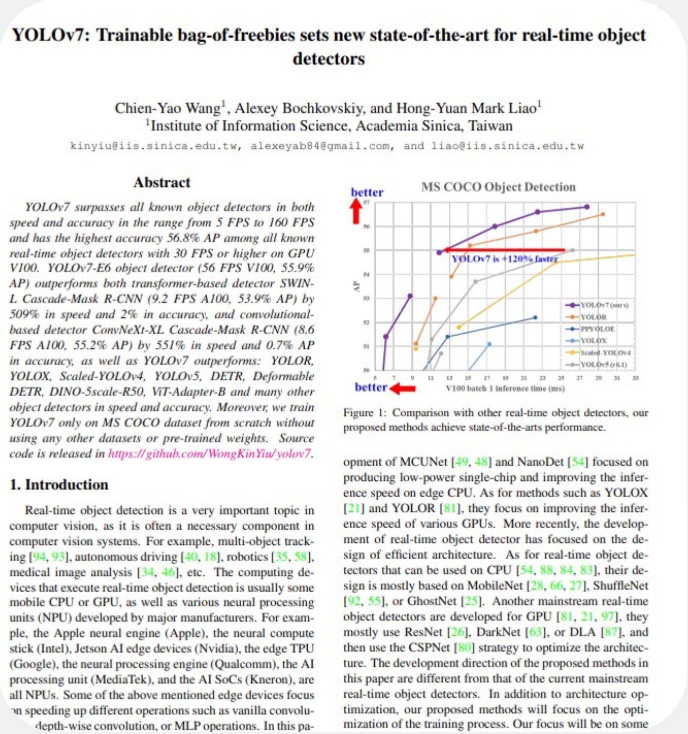
1. Introduction

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 tracking [94, 93], autonomous driving [40, 18], robotics [35, 58], medical image analysis [34, 46], etc. The computing devices 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 example, 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 convolution, depth-wise convolution, or MLP operations. In this pa-

opment of MCUNet [49, 48] and NanoDet [54] focused on producing low-power single-chip and improving the inference speed on edge CPU. As for methods such as YOLOX [21] and YOLOR [81], they focus on improving the inference speed of various GPUs. More recently, the development of real-time object detector has focused on the design of efficient architecture. As for real-time object detectors that can be used on CPU [54, 88, 84, 83], their design is mostly based on MobileNet [28, 66, 27], ShuffleNet [92, 55], or GhostNet [25]. Another mainstream real-time object detectors are developed for GPU [81, 21, 97], they mostly use ResNet [26], DarkNet [63], or DLA [87], and then use the CSPNet [80] strategy to optimize the architecture. The development direction of the proposed methods in this paper are different from that of the current mainstream real-time object detectors. In addition to architecture optimization, our proposed methods will focus on the optimization of the training process. Our focus will be on some

- **Chien-Yao Wang**
- **Alexey Bochkovskiy**
- **Hong-Yuan Mark Liao**

Introduction



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

Chien-Yao Wang[1], Alexey Bochkovskiy, and Hong-Yuan Mark Liao[1]
[1]Institute 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.

Figure 1: Comparison with other real-time object detectors, our proposed methods achieve state-of-the-arts performance.

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 tracking [94, 93], autonomous driving [40, 18], robotics [35, 58], medical image analysis [34, 46], etc. The computing devices 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 example, 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 convolution, depth-wise convolution, or MLP operations. In this pa-

opment of MCUNet [49, 48] and NanoDet [54] focused on producing low-power single-chip and improving the inference speed on edge CPU. As for methods such as YOLOX [21] and YOLOR [81], they focus on improving the inference speed of various GPUs. More recently, the development of real-time object detector has focused on the design of efficient architecture. As for real-time object detectors that can be used on CPU [54, 88, 84, 83], their design is mostly based on MobileNet [28, 66, 27], ShuffleNet [92, 55], or GhostNet [25]. Another mainstream real-time object detectors are developed for GPU [81, 21, 97], they mostly use ResNet [26], DarkNet [63], or DLA [87], and then use the CSPNet [80] strategy to optimize the architecture. The development direction of the proposed methods in this paper are different from that of the current mainstream real-time object detectors. In addition to architecture optimization, our proposed methods will focus on the optimization of the training process. Our focus will be on some

- Abstract
- How does it work?
- What approached they used?
- Why did they use the particular methods?
- Model Comparisons
- Why it's so awesome?

3

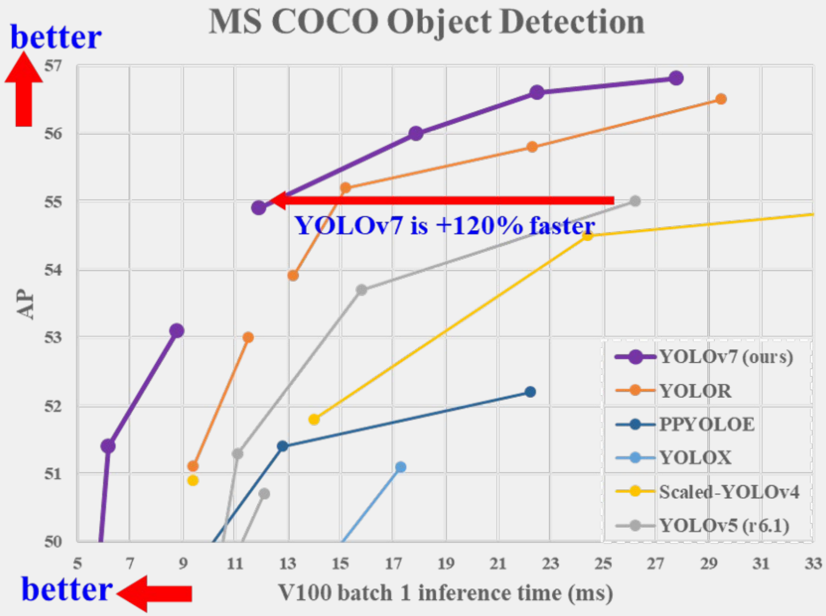
Abstract

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

- **5FPS to 160FPS**
- **56.8% AP**
- **50% reduction in cost**

YOLOv7 Comparison

- YOLOR
- YOLOX
- Scaled-YOLOv4
- YOLOv5
- DETR
- Deformable DETR
- DINO-5scale-R50
- ViT-Adapter-B



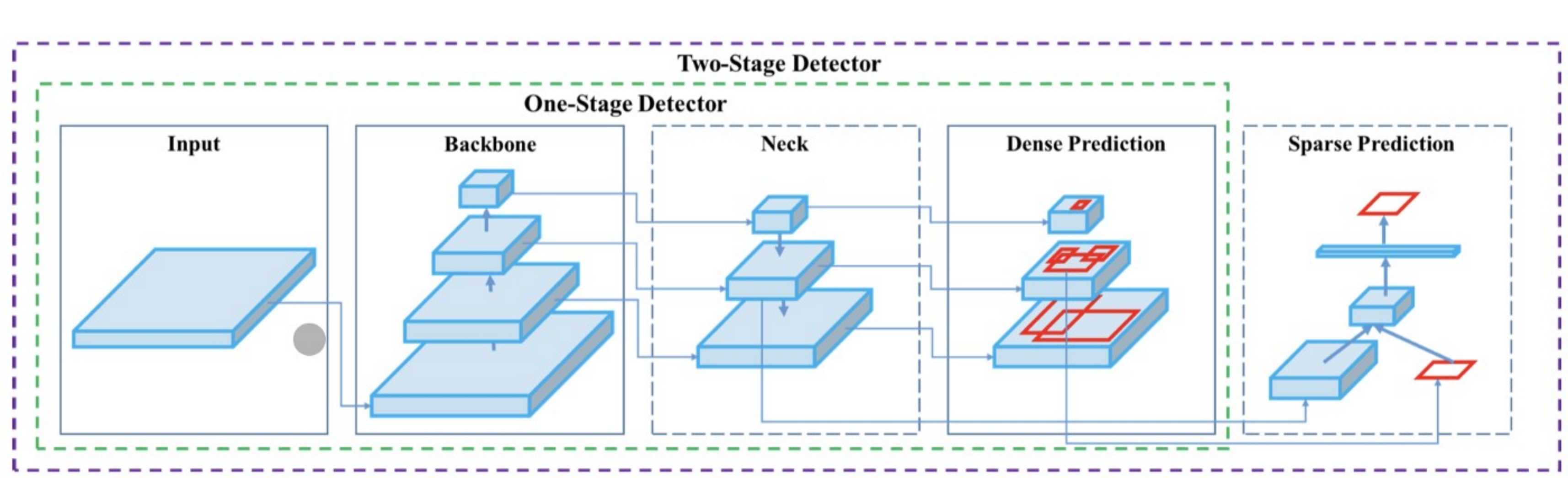
Comparison with YOLOv4

- Both are bag of freebies model
- 75% fewer parameters than YOLOv4.
- 36% lesser computational time.
- 1.5x times higher AP than YOLOv4

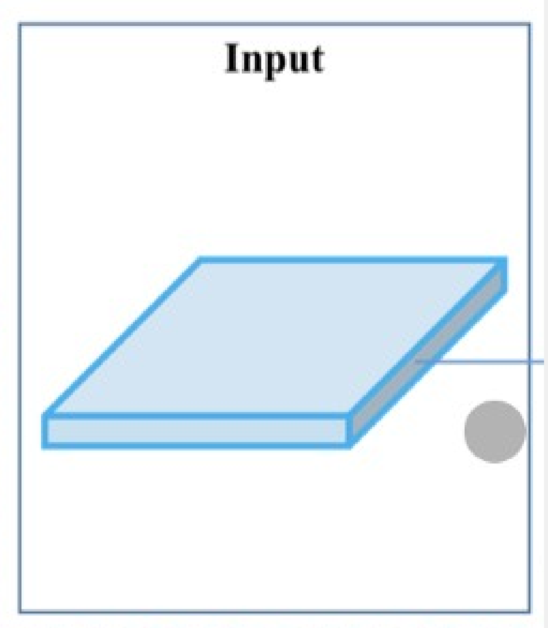
AUGMENTED STARTUPS
Computer Vision | AI | Robotics

How does it work

YOLO Architecture

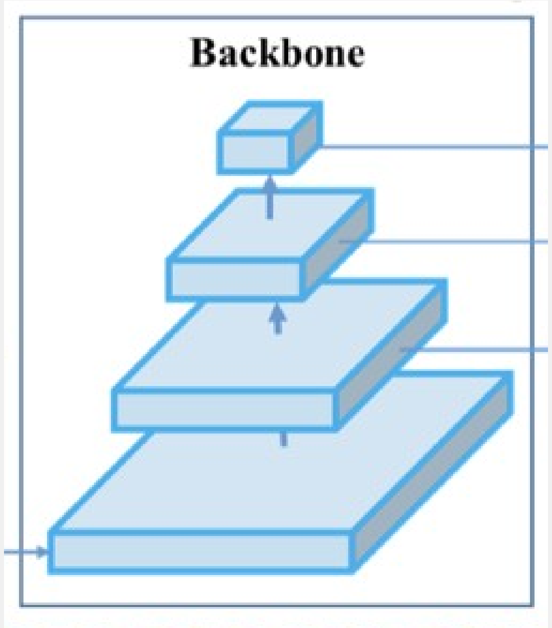


Architecture Selection



Input Image

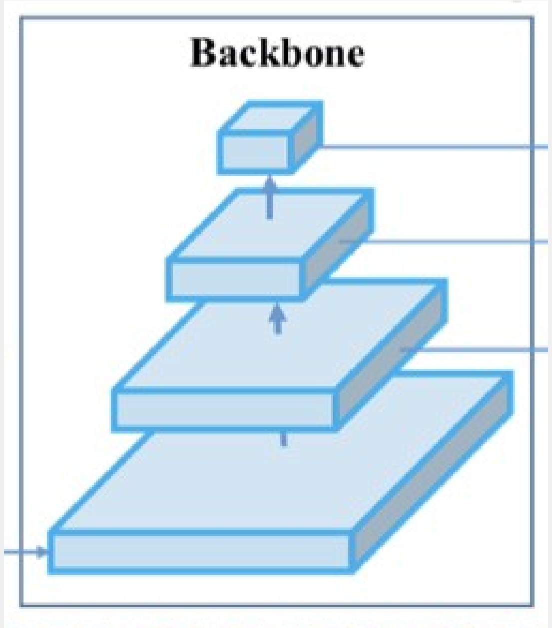
Architecture Selection



Pretrained Weights

1. VGG16
2. ImageNet
3. RetineNet
4. Resnet50

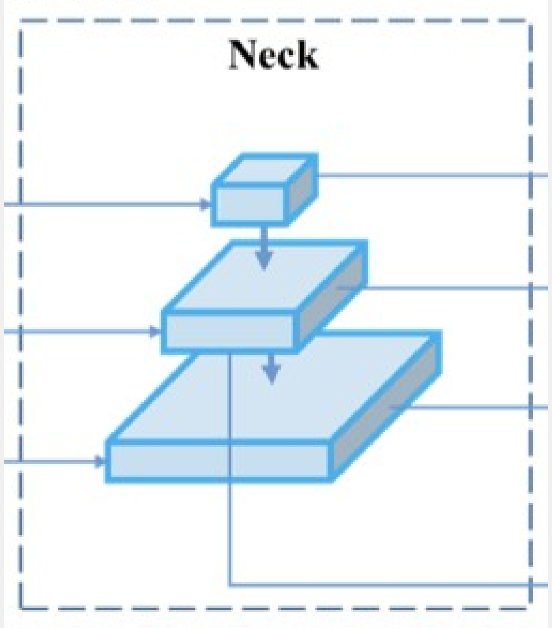
Architecture Selection



For YOLOv7

1. VoVNET
2. CSPVoVNET
3. ELAN
4. E-LAN

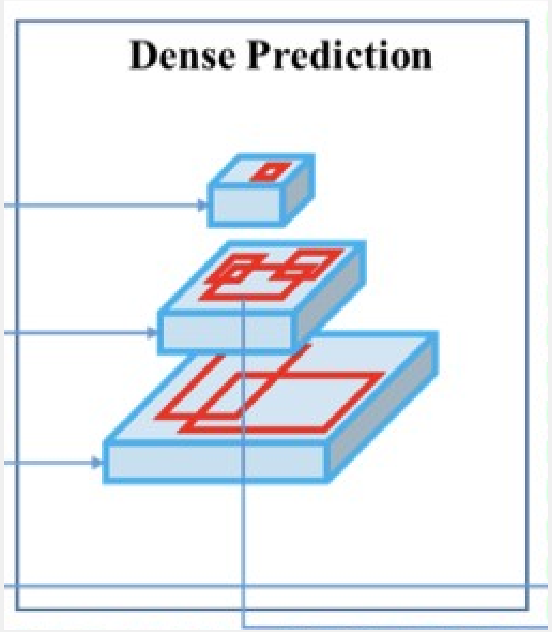
Architecture Selection



For Enhancement

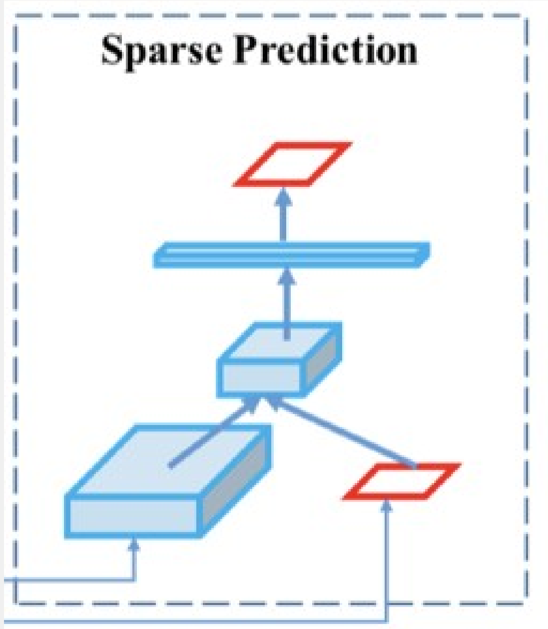
1. FPN
2. RFB
3. PAN

Architecture Selection



1. YOLO
2. SSD
3. RPN

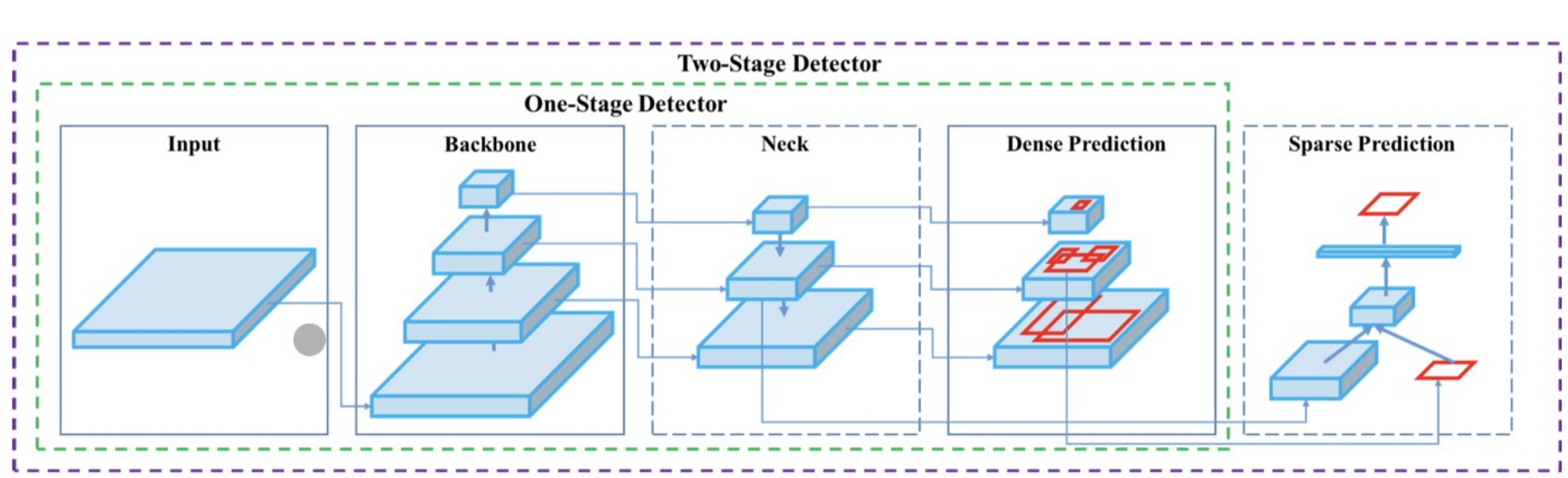
Architecture Selection



For Two Stage Detectors

1. FRCNN
2. RFCN

YOLO Architecture



Training Optimization

Bag of Freebies

1. Batch Normalization
2. Implicit Knowledge
3. EMA Model

Training Optimization

The author uses

- Gradient prediction to generate **coarse-to-fine hierarchical labels**.
- Extended efficient **layer aggregation networks.**
- Model scaling for concatenation-based models
- **Identity connection** in one convolutional layer.
- Train YOLOv7 only on MS COCO dataset from scratch **without** using any **other datasets** or **pre-trained weights**

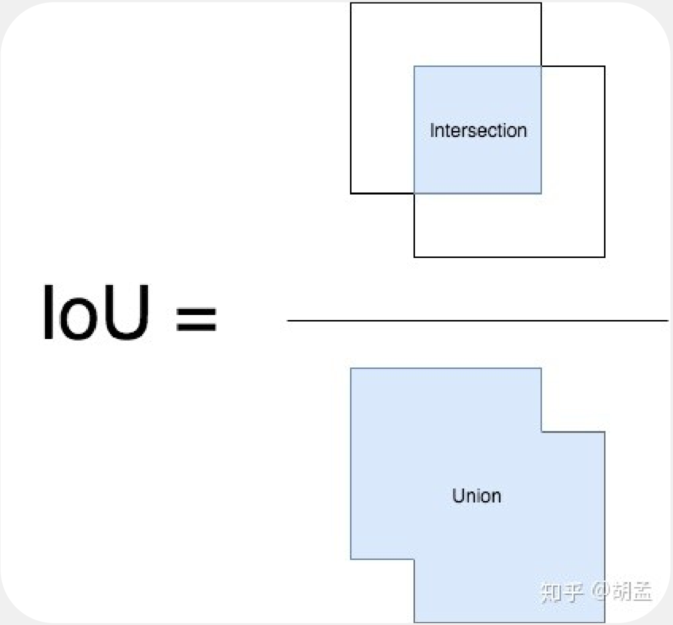
Training Optimization

The model gets higher AP when IoU is increased

Model	Presicion	IoU threshold	AP^{val}
YOLOv7-X	FP16 (default)	0.65 (default)	**52.9%**
YOLOv7-X	FP32	0.65	**53.0%**
YOLOv7-X	FP16	0.70	**53.0%**
YOLOv7-X	FP32	0.70	**53.1%**
improvement	-	-	+0.2%

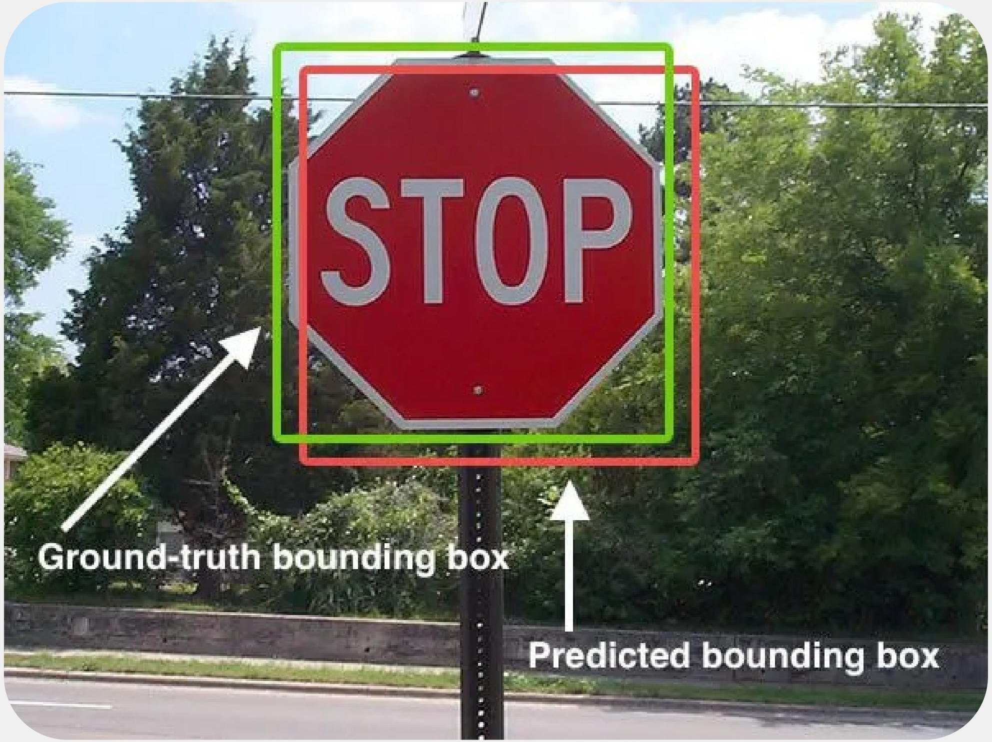
Training Optimization

IoU: Intersection over Union

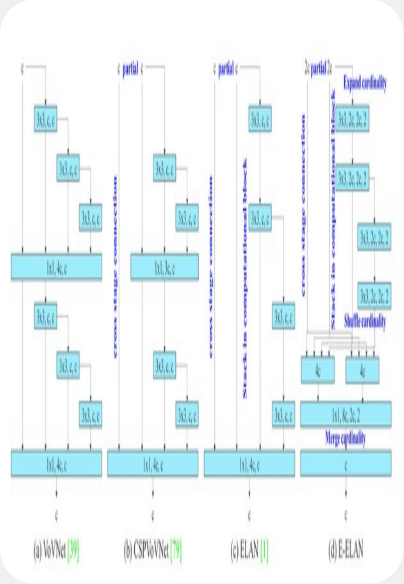


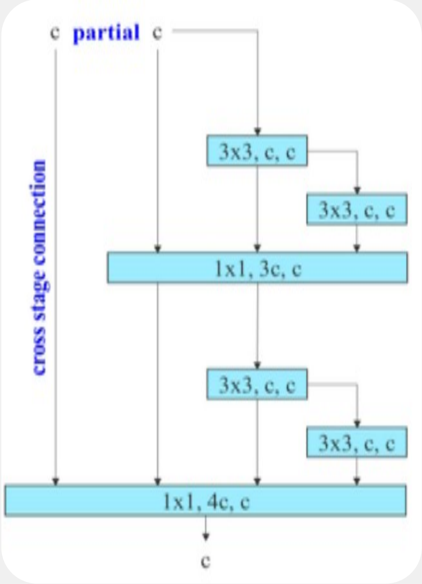
Training Optimization

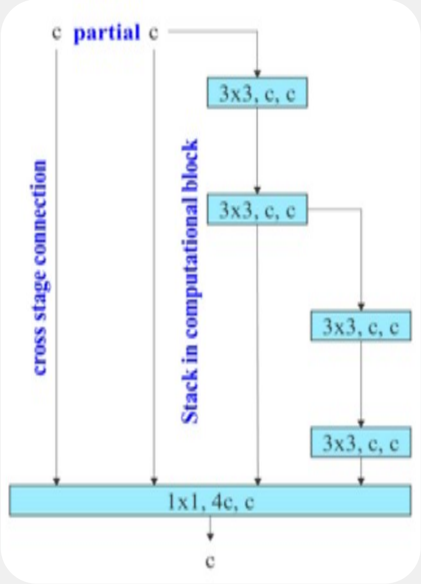
Intersection over Union

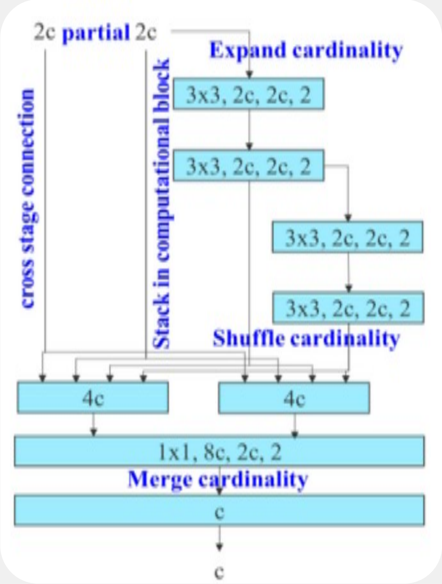


Layer Aggregation Networks









1. VoVNET

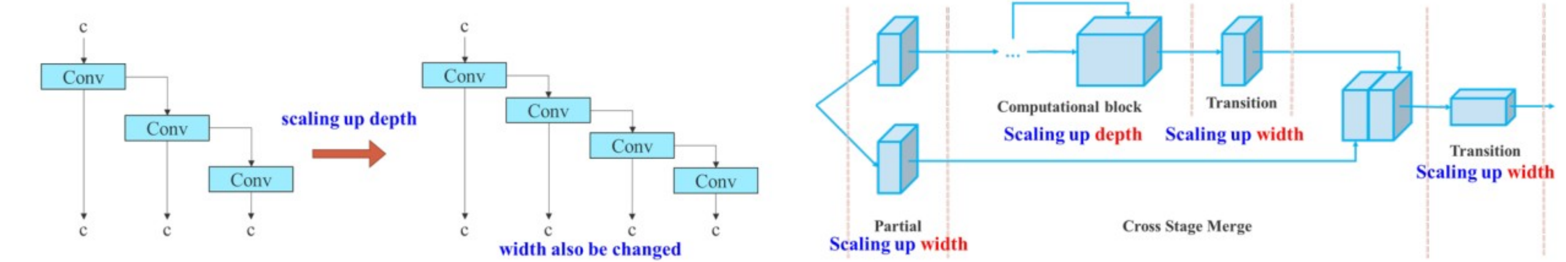
2. CSPVoVNET

3. ELAN

4. E-LAN

Model Scaling

Concatenation based Models

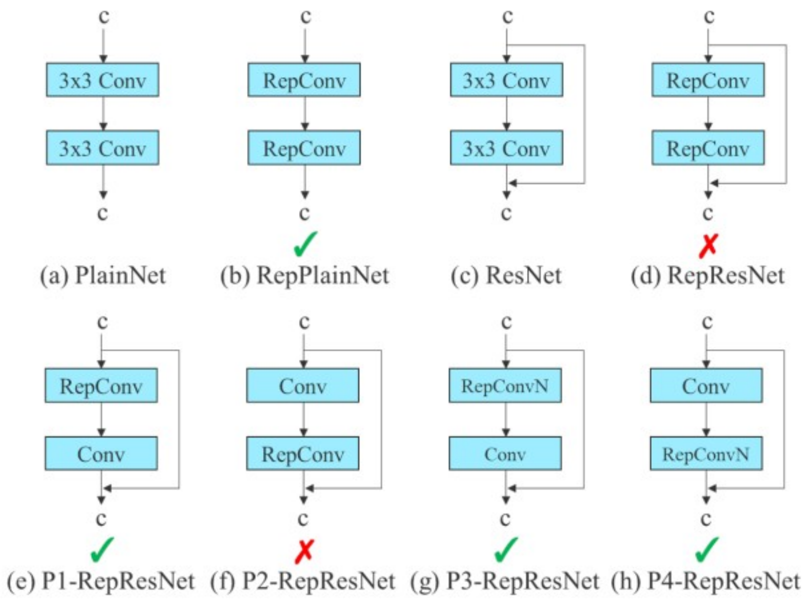


(a) concatenation-based model (b) scaled-up concatenation-based model (c) compound scaling up depth and width for concatenation-based model

Scaling Factors

- Resolution
- Depth
- Width
- Stage

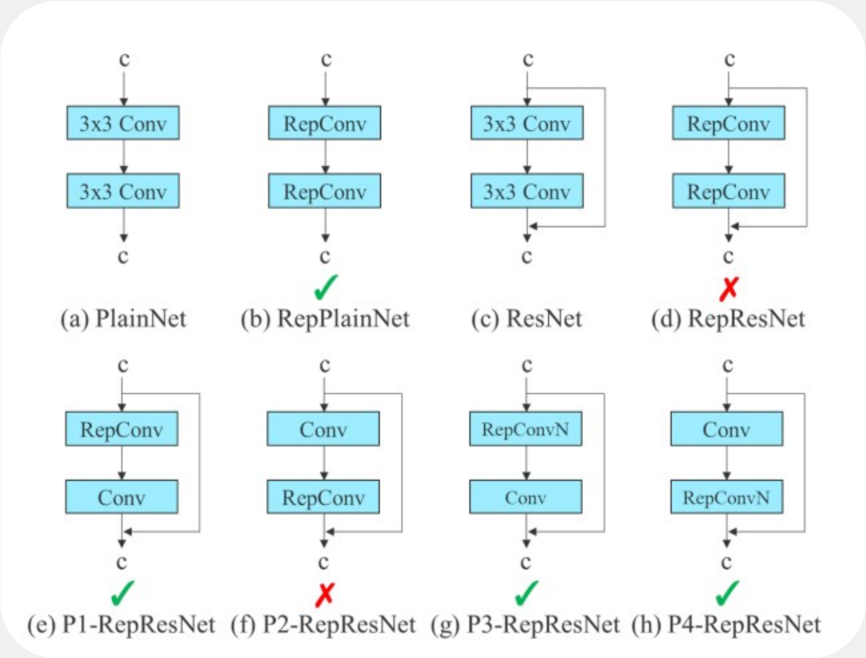
Model Re-parameterizing



Model Level Ensemble

Weighted Average of the weights of model at different iteration

Model Re-parameterizing

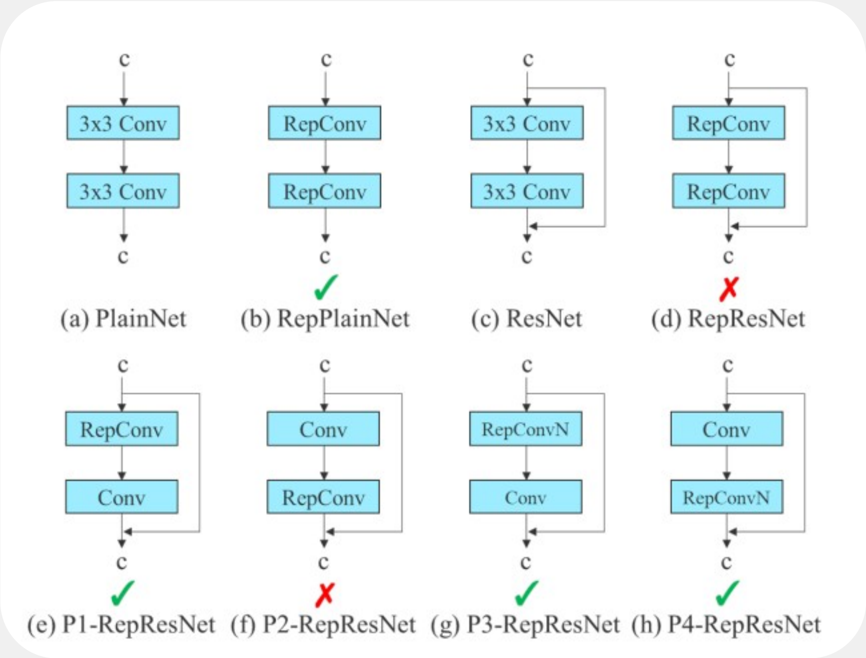


Module Level Ensemble

Train multiple identical models with different training data

Average of the weights of multiple trained models

Model Re-parameterizing

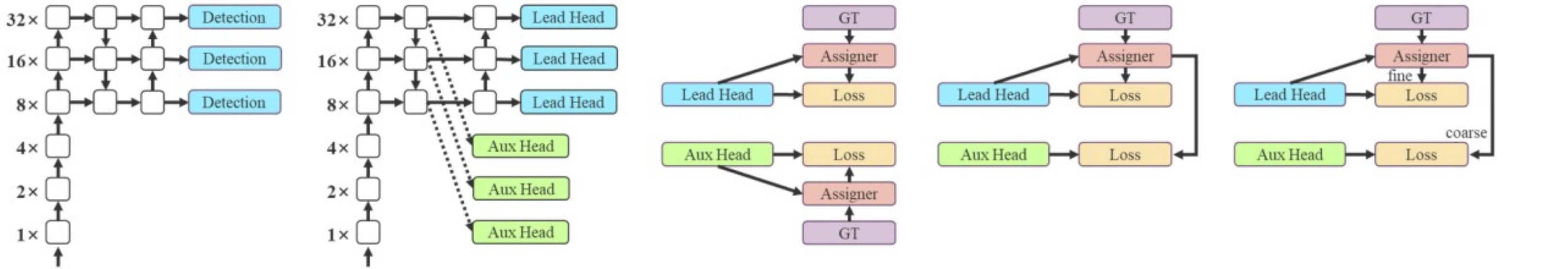


Module Level Ensemble

Split and integrate the branched modules into completely equivalent module

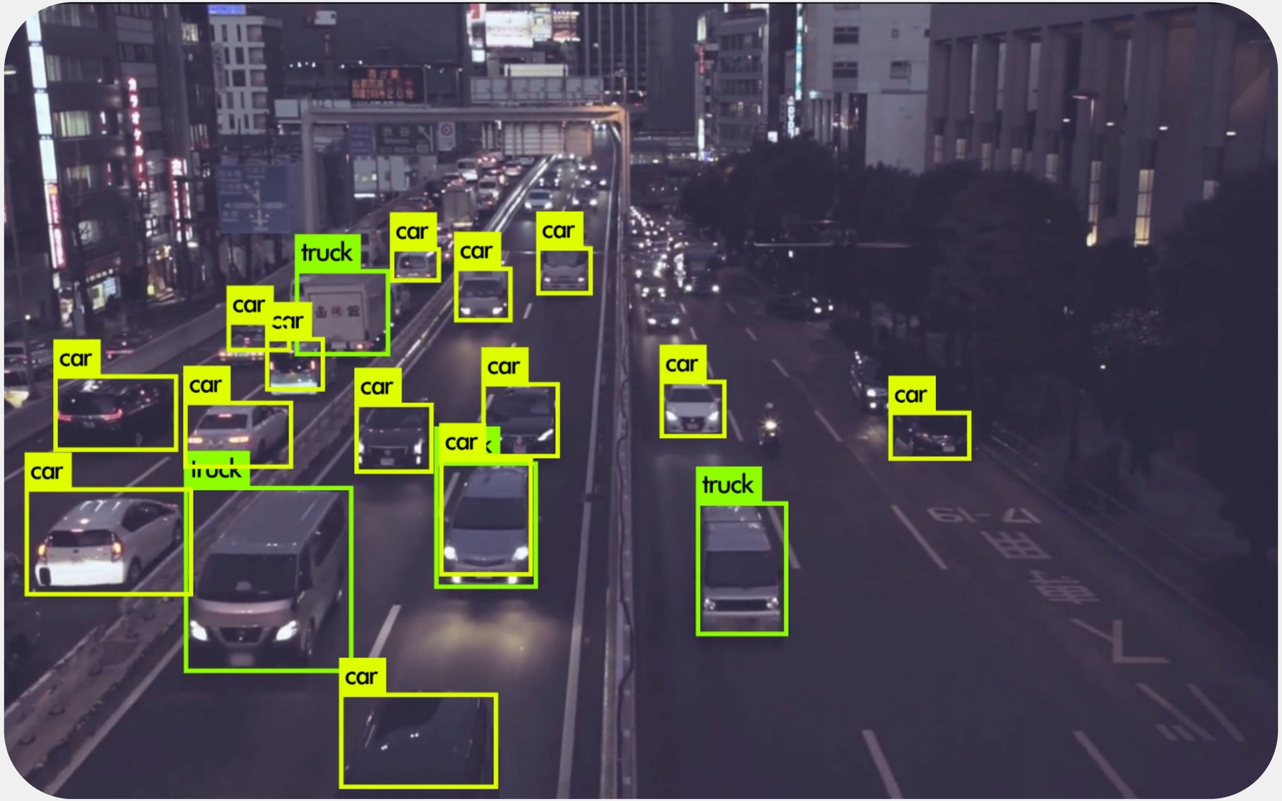
Auxillary Head Coarse-to-Fine

- Detection
- Depth
- Lead Head
- GT
- Assigner
- Loss



(a) Normal model (b) Model with auxiliary head (c) Independent assigner (d) Lead guided assigner (e) Coarse-to-fine lead guided assigner

Results


truck
car
car
car
car
car
car
car
car
car
car
car
car
car
truck
car
truck
car

Model Comparison

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

Model Comparison

AUGMENTED STARTUPS
Computer Vision | AI | Robotics

Model	#Param.	FLOPs	Size	AP^{val}	AP^{val}_{50}
YOLOv4 [3]	64.4M	142.8G	640	49.7%	68.2%
YOLOR-u5 (r6.1) [81]	46.5M	109.1G	640	50.2%	68.7%
YOLOv4-CSP [79]	52.9M	120.4G	640	50.3%	68.6%
YOLOR-CSP [81]	52.9M	120.4G	640	50.8%	69.5%
YOLOv7	36.9M	104.7G	640	**51.2%**	**69.7%**
improvement	-43%	-15%	-	+0.4	+0.2

Summary

- Replacement problem of **re-parameterized module** has been overcome by using **gradient flow propagation** paths to analyse how re-parameterized convolution should be combined with different network.
- Combines 3x3, **1x1 convolution** in one convolutional layer (RepConvN).

Summary

- Overcomes the problem of **dynamic label assignment** by using **coarse-to-fine lead head** guided label assigner.
- Introduces **Extended efficient layer aggregation** networks and Compound scaling for model scaling.