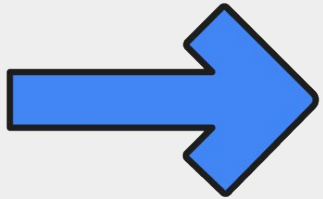
Google Developer Group
Incheon National University

YOLO_v10: Real-Time End-to-End Object Detection

GDG INU AI Part Paper Seminar

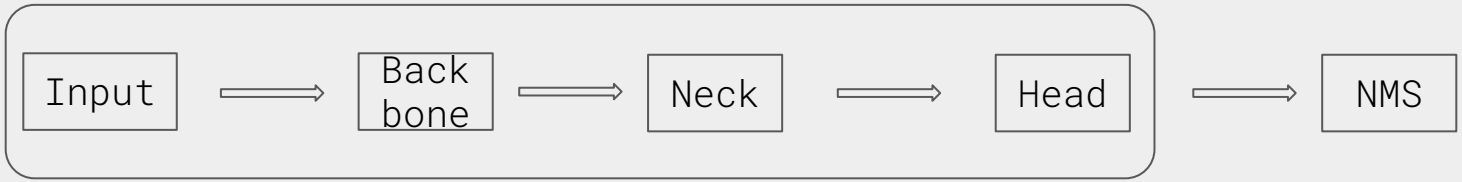
AI Member 옥정빈



Index

Abstract #0

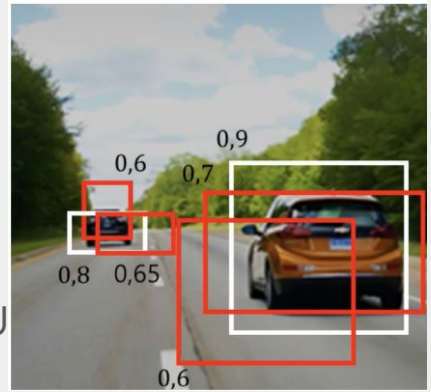
- YOLO, Forward Processing + Post-processing(NMS)
- v10에서는 Model Architecture와 Post-processing을 개선하여 이러한 문제를 해결함
- Post-processing(NMS)을 없애기 위해 consistent dual assignments를 제시
- Holistic efficiency-accuracy driven model design을 통해 새로운 Architecture를 제시



Abstract #1

Non-Maximum Suppression

1. Confidence score의 threshold를 지정, threshold 이상의 BBox만 남게 됨.
2. 남은 BBox들을 C.S. 기준으로 내림차순 정렬
3. 맨 앞에 있는 (Highest C.S,) BBox 하나를 기준으로 잡고, 다른 BBox와 IoU 값을 구함.
4. 이때 IoU가 IoU_threshold 이상인 BBox들은 제거.



1. C.S의 threshold = 0.4
2. [0.9, 0.8, 0.7, 0.65, 0.6, 0.6]
3. 0.9인 BBox와 나머지를 비교. 이때 0.8은 겹치지 않으므로 남겨둠.

   0.7 박스와는 IoU가 threshold 이상이므로 이 박스는 0.9와 같은 것을 가리킨다고 간주하고 제거.
4. 그림에서 결과적으로 하얀 박스가 NMS 후의 박스.

Introduction #0

- Forward + NMS 는 정확도와 지연시간 측면에서 최적의 결과를 내지 못함.
- YOLO는 일반적으로 One-to-Many(o2m) label assignment 방법을 사용함.
- 이러한 방법은 NMS를 필요로 하여 추론 속도를 저하시키며 End-to-End가 불가하고 NMS의 H.P. 설정에 따라 모델이 민감하게 변함.

One-to-One(o2o) vs. One-to-Many(o2m)

o2o: 하나의 G.T.에 하나의 sample을 할당. 고양이1->(3, 4), 고양이2->(6, 3)
o2m: 하나의 G.T.에 여러 개의 samples를 할당. 고양이1->(3, 4), (4, 6), (3, 5), 고양이2->(6,3), (5, 3), (4, 5) => o2m 경우 정확성이 높고 Augmentation에 유리

Introduction #1

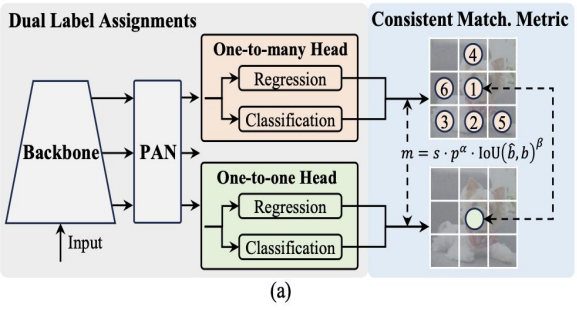
1. NMS-Free YOLO Training
   - Consistent Dual Assignments
   - Consistent Matching Metric

2. Efficiency-Accuracy Driven Model Design
   - Lightweight Classification Head
   - Spatial-Channel Decoupled Downsampling
   - Rank-Guided Block Design

3. Accuracy Enhancement
   - Large-Kernel Conv (to expand Receptive Field)
   - Partial Self-Attention, PSA

Methodology #0

1. Consistent Dual Assignments for NMS-Free Training



Optimization Objectives(B.B_Loss + Cla_Loss)

o2o 매칭에서는 Top-1 선택 방식을 사용하여, Hungarian Matching과 유사한 성능을 유지하면서도 추가적인 학습 시간을 단축하는 데 성공함.

Hungarian Matching & Top one selection

방식	특징	최적화 방식	장점	단점
Hungarian Matching	1:1 매칭	비용 행렬을 최적화 (IoU, L1, ClsLoss 등)	중복 탐지를 방지하고 GT에 정확한 예측값을 매칭	계산량이 많음
Top-One Selection	가장 높은 IoU의 예측값 선택	단순 IoU 기반 선택	연산량이 적고 빠름	Hungarian Matching처럼 최적화되지 않아 매칭 품질이 낮을 수 있음

Methodology #1

1-1. Consistent matching metric

o2o 및 o2m 매칭에서는 객체와 예측 간의 일치도를 평가하는 매칭 기준이 필요함.

이를 위해, 본 연구에서는 다음과 같은 매칭 기법을 도입하여 두 탐지 헤드가 동일한 매칭 기준을 공유하도록 설계함.

$$\hat{m}(\alpha, \beta) = s \cdot p^{\alpha} \cdot IoU(\hat{b}, b)^{\beta}$$

$$m_{o2m} = m(\alpha_{o2m}, \beta_{o2m})$$

$$m_{o2o} = m(\alpha_{o2o}, \beta_{o2o})$$

- P : 분류 점수 (classification score)
- b_hat, b : 각각 예측된 경계상자와 실제 경계상자
- s : 공간적 사전정보로, 예측된 앵커 포인트가 객체 내부에 있는지 확인
- alpha, beta : 각각 분류(classification)과 위치 회귀(Location Regression) 간의 영향을 조절하는 하이퍼파라미터

Methodology #2

1-1. Consistent matching metric

1. 훈련 초기에는 두 헤드가 동일한 초기값을 갖고 동일한 예측을 수행한다고 가정할 수 있음. 즉, o2o과 o2m은 동일한 분류 점수 p와 IoU 점수를 생성
2. o2m에서 가장 높은 IoU값을 u*라고 하자.
3. 매칭 지표에서 가장 높은 값을 m*_o2m, m*_o2o라고 하자.
4. 이때. Classification target(분류 목표)는 다음과 같이 유도 된다.

$$t_{o2m,j} = u^* \cdot \frac{m_{o2m,j}}{m^*_{o2m}}, \quad \text{for } j \in \Omega$$

$$t_{o2o,i} = u^* \cdot \frac{m_{o2o,i}}{m^*_{o2o}} = u^*$$

$$m_{o2m} = m(\alpha_{o2m}, \beta_{o2m})$$
$$m_{o2o} = m(\alpha_{o2o}, \beta_{o2o})$$

L_cls(o2m) = Sum{L_task(p_j, t_o2m,j)}
L_cls(o2o) = L_task(p_i, t_o2o,i)

*L_task : task aligned loss

Methodology #3

1-1. Consistent matching metric

두 방식 간의 Supervision Gap(o2o와 o2m의 목표값 차이) 는 다음과 같이 1- Wasserstein dist. 로 정의할 수 있다.

$$A = t_{o2o,i} - I(i \in \Omega) t_{o2m,i} + \sum_{k \in \Omega \setminus \{i\}} t_{o2m,k} = t_{o2o,i} + \sum_{k \in \Omega} t_{o2m,k} - 2I(i \in \Omega) t_{o2m,i}$$

$$t_{o2m,j} = u^* \cdot \frac{m_{o2m,j}}{m^*_{o2m}}, \quad \text{for } j \in \Omega$$
$$t_{o2o,i} = u^* \cdot \frac{m_{o2o,i}}{m^*_{o2o}} = u^*$$

$$m_{o2m} = m(\alpha_{o2m}, \beta_{o2m})$$
$$m_{o2o} = m(\alpha_{o2o}, \beta_{o2o})$$

$$\alpha_{o2o} = r \cdot \alpha_{o2m}, \quad \beta_{o2o} = r \cdot \beta_{o2m} \text{ 즉, } m_{o2o} = m^r_{o2m}$$

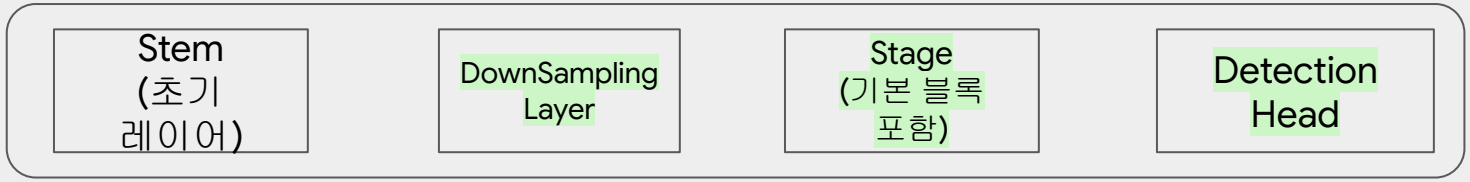
단순화를 위해 r=1로 설정한다. 그리고 o2m에서 알파=0.5, 베타=6.0 을 사용할 때 가장 좋은 성능을 보였다.

Methodology #4

2. Holistic Efficiency-Accuracy Driven Model Design

YOLO는 Post-Processing뿐만 아니라, 모델 아키텍쳐 자체에서도 효율성과 정확성 간의 균형을 맞추는 데 어려움이 있다.

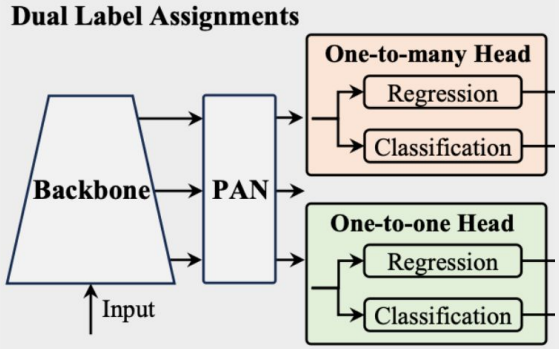
여기에는 상당한 연산적 중복(Computational Redundancy)와 제한된 기능성(Constrained Capability)이 존재하여, 고효율 및 고성능을 동시에 달성하는 것이 어렵다.



Methodology #5

2-1. Efficiency Driven Model Design

(1) Lightweight classification head



실험결과, YOLO의 성능은 분류보다 회귀에 더 큰 영향을 받음을 확인하였다. 따라서 경량화된 분류헤드 (Depthwise Separable Conv를 사용한 3x3커널 2개 + 1x1커널)를 적용하여 성능 저하 없이 연산 비용을 줄였다.

Depthwise Separable Conv vs. Std Conv

E.g.) C_in = 32, C_out = 64, H, W, k_size = 3 인 경우
Depthwise Separable은 두 번(Depthwise + Pointwise)에 걸쳐 계산한다. 이때 계산량을 비교해보자면
첫 번째 Depthwise_O = H x W x C_in x 3 x 3 = H x W x 288 -> 32개 채널마다 서로다른 커널을 사용
두 번째 Pointwise_O = H x W x C_in x C_out = H x W x 2048 -> 위의 결과로 생성된 32개의 채널을 64개로 늘리기 위해 1x1 커널을 사용

Std Conv_O = H x W x C_in x C_out x 3x3 = H x W x 18432,

연산량이 훨씬 줄어들지만 성능을 비슷한 결과가 나옴.

Methodology #6

2-1. Efficiency Driven Model Design

(2) Spatial-channel decoupled downsampling

기존 YOLO에서는 일반적인 3x3(stride=2) 커널을 사용하여 다운샘플링을 수행한다. 하지만 이는 공간 정보에 대한 손실이 크며, 채널 크기 조절을 비효율적으로 수행하는 문제가 있다.

이를 해결하기 위해, 채널 크기를 먼저 조절한 후 (1x1 Pointwise Conv), 공간을 축소하는 방식(3x3 Depthwise Conv)으로 다운샘플링 순서를 변경하였다.
그 결과 성능이 0.7% AP 개선되었다.

Methodology #7

2-1. Efficiency Driven Model Design

(3) Rank-Guided Block Design(순위 기반 블록 설계)

YOLO 모델의 각 Stage(블록)은 내재적인 중요도(Intrinsic Rank)가 다르다.
즉, 어떤 블록은 중요한 정보를 담고 있지만, 어떤 블록은 중복된 연산을 포함할 수 있다.

이를 해결하기 위해 블록의 순위를 정렬하고, 가장 덜 중요한 블록을 경량화된 구조로 대체하는 방식을 적용했다.
예를 들어, Stage 8과 4에서 기존 병목 블록을 CIB(Compact Inverted Block)으로 교체하면 성능 저하 없이 연산량을 줄일 수 있다.

Methodology #8

2-2. Accuracy Driven Model Design

(1) Large-Kernel Conv (7x7)

일반적인 YOLO에서는 3x3을 주로 사용하지만 Receptive Field(수용영역)가 제한적이다.
따라서 실험 결과 7x7 커널을 사용할 경우 수용영역이 증가하면서 성능이 0.4% AP 개선되었다.
(다만, 대형 모델 (YOLOv10-M이상)에서는 이미 넓은 수용 영역을 가지므로 추가적인 성능 향상이 없음을 확인했다.)

(2) Partial Self-Attention, PSA

기존의 Transformer 기반 객체 탐지 모델들은 Global Attention을 사용했지만, 이는 연산량이 매우 크다.
따라서 PSA를 적용하여 일부 중요한 정보에만 Attention을 적용하는 방식을 채택하였다.
실험결과, PSA는 기존 Transformer 블록 대비 0.3% AP 성능 향상을 제공하면서, 지연 시간은 0.05ms 감소하였다.
(PSA 블록의 개수를 증가시키면 추가적인 성능향상이 가능하지만, 연산 비용 증가를 고려하여 1개만 적용하였다.)

Experiments #0

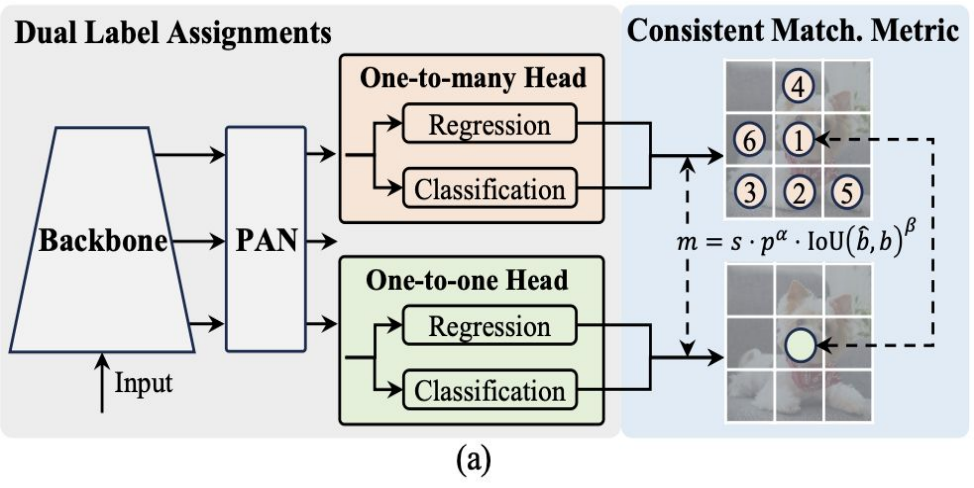
Model	#Param.(M)	FLOPs(G)	AP^{val}(%)	Latency(ms)	$Latency^{f}$(ms)
YOLOv6-3.0-N [29]	4.7	11.4	37.0	2.69	1.76
Gold-YOLO-N [60]	5.6	12.1	39.6	2.92	1.82
YOLOv8-N [21]	3.2	8.7	37.3	6.16	1.77
YOLOv10-N (Ours)	**2.3**	**6.7**	**38.5 / 39.5**†	**1.84**	**1.79**
YOLOv6-3.0-S [29]	18.5	45.3	44.3	3.42	2.35
Gold-YOLO-S [60]	21.5	46.0	45.4	3.82	2.73
YOLO-MS-XS [8]	4.5	17.4	43.4	8.23	2.80
YOLO-MS-S [8]	8.1	31.2	46.2	10.12	4.83
YOLOv8-S [21]	11.2	28.6	44.9	7.07	2.33
YOLOv9-S [65]	7.1	26.4	46.7	-	-
RT-DETR-R18 [78]	20.0	60.0	46.5	4.58	4.49
YOLOv10-S (Ours)	**7.2**	**21.6**	**46.3 / 46.8**†	**2.49**	**2.39**
YOLOv6-3.0-M [29]	34.9	85.8	49.1	5.63	4.56
Gold-YOLO-M [60]	41.3	87.5	49.8	6.38	5.45
YOLO-MS [8]	22.2	80.2	51.0	12.41	7.30
YOLOv8-M [21]	25.9	78.9	50.6	9.50	5.09
YOLOv9-M [65]	20.0	76.3	51.1	-	-
RT-DETR-R34 [78]	31.0	92.0	48.9	6.32	6.21
RT-DETR-R50m [78]	36.0	100.0	51.3	6.90	6.84
YOLOv10-M (Ours)	**15.4**	**59.1**	**51.1 / 51.3**†	**4.74**	**4.63**
YOLOv6-3.0-L [29]	59.6	150.7	51.8	9.02	7.90
Gold-YOLO-L [60]	75.1	151.7	51.8	10.65	9.78
YOLOv9-C [65]	25.3	102.1	52.5	10.57	6.13
YOLOv10-B (Ours)	**19.1**	**92.0**	**52.5 / 52.7**†	**5.74**	**5.67**
YOLOv8-L [21]	43.7	165.2	52.9	12.39	8.06
RT-DETR-R50 [78]	42.0	136.0	53.1	9.20	9.07
YOLOv10-L (Ours)	**24.4**	**120.3**	**53.2 / 53.4**†	**7.28**	**7.21**
YOLOv8-X [21]	68.2	257.8	53.9	16.86	12.83
RT-DETR-R101 [78]	76.0	259.0	54.3	13.71	13.58
YOLOv10-X (Ours)	**29.5**	**160.4**	**54.4 / 54.4**†	**10.70**	**10.60**

Baseline 모델은 YOLOv8로 선택. YOLOv10은 N/S/M/L/X 5가지 변형 모델을 제공한다. 추가적으로 M의 너비 스케일 계수를 증가시킨 B 모델도 도입하였다.

본 연구에서 제안하는 YOLOv10은 COCO 데이터셋에서 학습 초기화 없이 동일한 설정으로 검증되었고, 모든 모델의 Latency는 T4 GPU에서 TensorRT FP16 환경에서 측정되었다.

YOLOv10은 다양한 모델 크기에서 SOTA와 End-to-End 추론 속도를 달성하였다.

Summary



L_cls(o2m) = Sum{L_task(p_j, t_o2m,j)}
L_cls(o2o) = L_task(p_i, t_o2o,i)

*L_task : task aligned loss

L_Reg(o2m) = Sum{ 1-IoU)
L_Reg(o2o) = 1 - IoU

$$t_{o2m,j} = u^* \cdot \frac{m_{o2m,j}}{m^*_{o2m}}, \quad \text{for } j \in \Omega$$
$$t_{o2o,i} = u^* \cdot \frac{m_{o2o,i}}{m^*_{o2o}} = u^*$$

$$m_{o2m} = m(\alpha_{o2m}, \beta_{o2m})$$
$$m_{o2o} = m(\alpha_{o2o}, \beta_{o2o})$$

$$\hat{m}(\alpha, \beta) = s \cdot p^{\alpha} \cdot IoU(\hat{b}, b)^{\beta}$$

Google Developer Group
Incheon National University