MERGERITY

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

AI 팀 Navy

xxx[1] :1저자를 의미

YOLO(v1~v7) 타임라인

YOLO (2015) → YOLO-v2 (2016) → YOLO-v3 (2018) → YOLO-v4 (2019) → YOLO-v5 (2020)

made by Joseph Redmon[1]

maed by Alexey Bochkovskiy[1] Chien-Yao Wang

Based

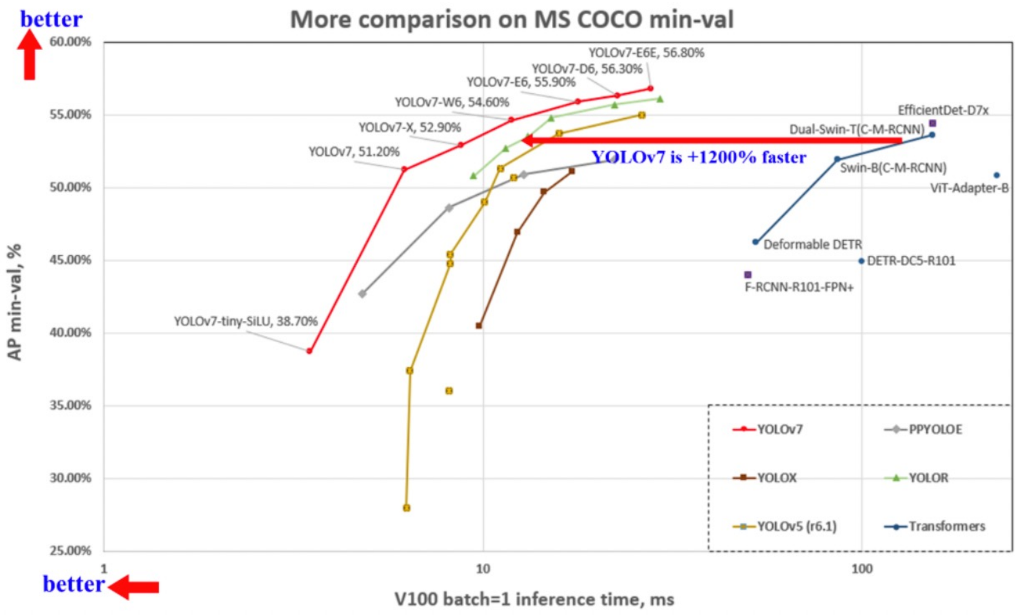
→ YOLOX, YOLO-v6, YOLOR, PPYOLO (2021) → YOLO-v7(Detectron based, No paper) (2022) YOLO-v7(Official)

maed by Chien-Yao Wang[1] Alexey Bochkovskiy

목차

- Abstract
- Related work
- Architecture
- Trainable bag-of-freebies
- Experiments
- 참고문헌

Abstract



- **5fps~160fps** 사이에서 가장 뛰어난 성능
- v100 GPU 30fps 이상의 Detector 중 가장 높은 **AP 56.8** 달성
- **Inference cost를 증가시키지 않는 최적화 기법** 제안
  - trainable bag-of-freebies
    - **Re-parameterization, dynamic label assignment**
- Extend&Compound scaling method 제안
  - 효율적인 parameter사용과 계산을 가능하게 하는 **detector architecture 제안**
- **파라미터 수 40%, 계산 량 50% 감소**
- **Proposed**
  - **Architecture**
    - **E-ELAN**
    - **Compound Scaling Method**
  - **Trainable bag-of-freebies**
    - **Planned re-parameterized convolution**
    - **Coarse for auxiliary and fine for lead loss**

Related work

- Model Re-parameterization
  - inference stage에서 여러 computational modules을 하나로 합치는 것
    - 일종의 ensemble 기법
  - 2가지 ensemble 기법으로 나뉨
    - Model-level Re-Parameterization
      - 동일한 모델을 다른 데이터로 여러 번 학습 시킨 후 weight 평균내기
      - 서로 다른 iteration에서 weight 평균 내기
    - Module-level Re-Parameterization
      - 학습동안 모듈을 여러 개의 module branch로 나누고 inference에서 하나로 통합
  - 본 논문에서는 여러 architecture에서 적용가능한 Module-level Re-parameterization 기법 제안

Related work

- Model scaling
  - 이미 설계된 모델을 scale up or down 하는 방법
    - 다른 device에 fit 시키기는 방법 중 하나
  - Scaling factor : 네트워크의 parameter수, inference speed, accuracy의 밸런스를 맞추기 위함
    - Resolution(input image size)
    - depth(number of layer)
    - width(number of channel)
    - stage(number of feature pyramid)
  - Network architecture search(NAS)
    - scaling method, 자동으로 scaling factor 찾음
    - 모든 scaling factor를 독립적으로 생각하고 찾으며 compound scaling factor도 각 요소를 독립적으로 생각함 -> **새로운 compound scaling method 제안**

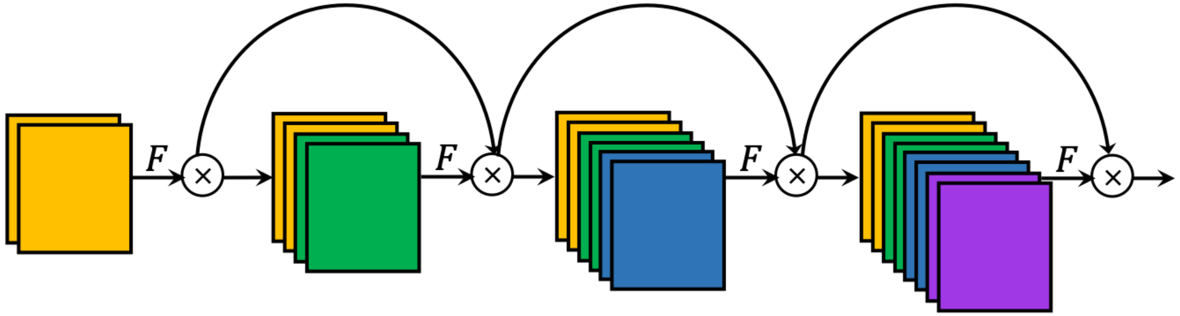
Architecture – Extended efficient layer aggregation networks

- Backbone 네트워크 제안
- 효율적인 architecture 고려사항
  - parameter 수, 계산 량, 계산 밀도(density)
  - 메모리 측면 : input/output 채널 비율, architecture branch 수, element-wise operation 등
  - convolution layer의 출력 Tensor(activation function)

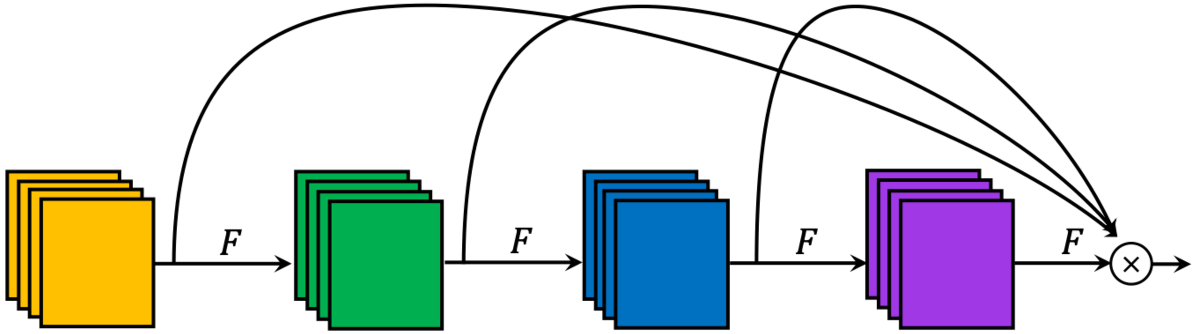
Architecture – Extended efficient layer aggregation networks(base 아키텍처 별 특징)

- DenseNet
  - input dimension 의 크기를 줄이기 위해 1x1 conv가 필수적
  - Vanishing Gradient 개선, Feature Propagation 강화, Feature Reuse, Parameter 수 절약
  - depthr가 깊어질수록 channel 수 증가
- VoVNet
  - input channel 수가 일정 -> 1x1 conv가 필요없음
- CSPVoVNet(Scaled-YOLOv4)
  - Cross Stage Partial(CSP)를 추가 -> input channel의 절반은 trainsition layer로 전달, 나머지 절반은 VoVNet 구조로 전달-> feature를 추출 후 transition layer에서 결합
  - 나눠진 channel 때문에 기존 gradient flow가 줄어들어 되어 과도한 양의 gradient information을 방지
  - 하지만 channel을 나누고 병합하는 과정을 통해 gradient path는 2배 증가 -> 다양한 feature 학습 가능

Architecture – Base architecture 참고



(a) Dense Aggregation (DenseNet)



(b) One-Shot Aggregation (VoVNet)

Architecture – Extended efficient layer aggregation networks(base 아키텍처 별 특징)

- ELAN
  - CSPVoVNet 장점 계승
  - 가장 짧고, 가장 긴 Gradient path의 차이를 극대화 -> 모듈 간소화
    - deep 한 네트워크 학습 가능
    - 모델 수렴 효과적
  - Computation block 을 깊게 쌓아도 학습 잘 됨
  - But 무한대 가까이 쌓을 경우 stable state 붕괴, parameter utilization 하락
- **E-ELAN(Proposed)**
  - ELAN의 문제점 해결
  - Expand, shuffle, merge cardinality를 추가 -> computational block을 많이 쌓아도 학습 능력 뛰어남
  - (Scaling에 따라) computational block 만 바뀌고 transition layer는 바뀌지 않음
  - feature를 다양하게 하고 parameter 사용과 계산을 향상시킴

Architecture – Extended efficient layer aggregation networks

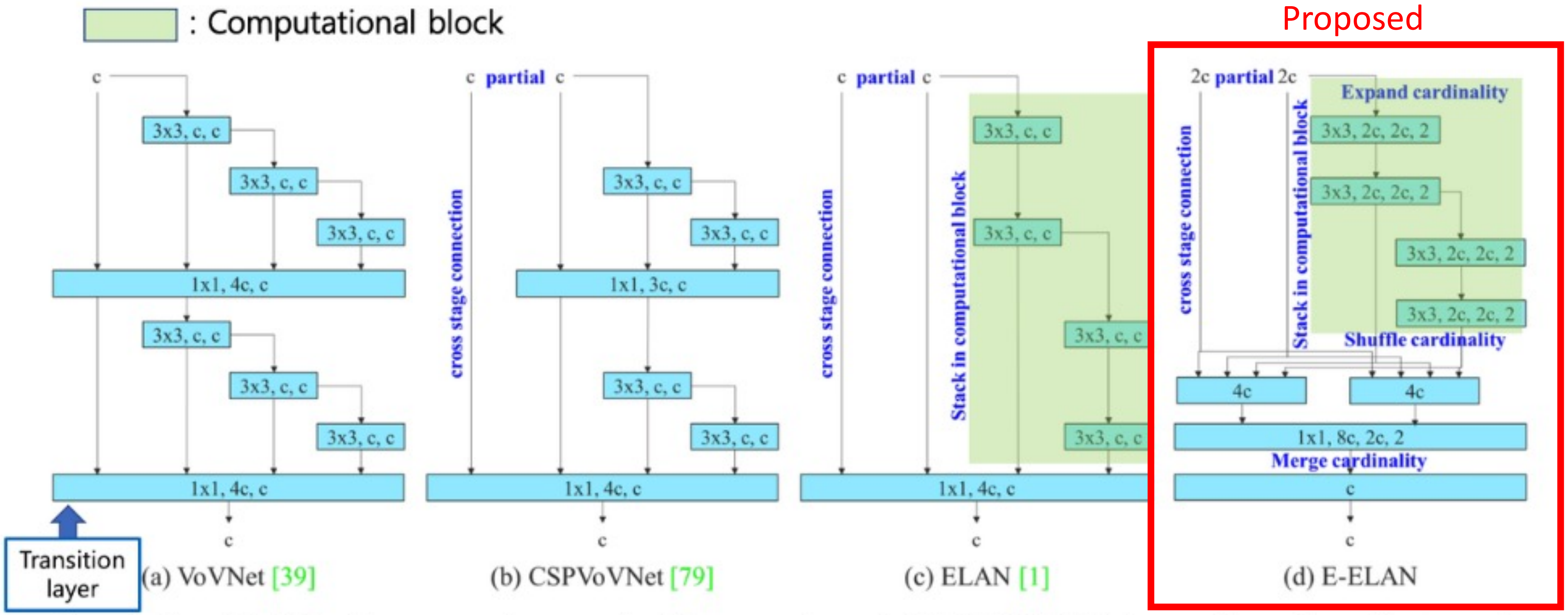


Figure 2: Extended efficient layer aggregation networks. The proposed extended ELAN (E-ELAN) does not change the gradient transmission path of the original architecture at all, but use group convolution to increase the cardinality of the added features, and combine the features of different groups in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

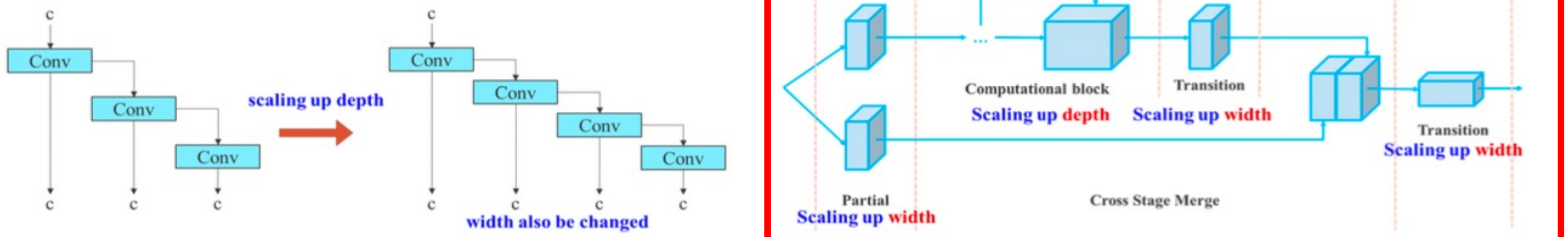
Architecture – Model scaling for concatenation-based models

- Model scaling : inference speed를 충족시키기 위해 다른 모델을 만들고 attribute를 맞추는 것
  - scaling 종류
    - width : filter(=channel) scaling
    - depth : layer 수 scaling
    - resolution : input image 해상도
    - compound : width + depth + resolution scaling을 조절

Architecture – Model scaling for concatenation-based models

- 실험을 통해 concatenation-based 모델의 경우 depth가 깊어질 때 width도 같이 증가하는 것을 확인
- depth만 증가시키는 scaling 방법 제안
  - layer의 in-degree, out-degree가 바뀌지 않아 파라미터와 계산 량의 scaling factor의 각 영향을 독립적으로 분석가능

Proposed



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

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.

Architecture – Model scaling for concatenation-based models

Ablation study

Table 3: Ablation study on proposed model scaling.

Model	#Param.	FLOPs	Size	$\mathbf{AP}^{val}$	$\mathbf{AP}^{val}_{50}$	$\mathbf{AP}^{val}_{75}$
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

Trainable bag-of-freebies
- Planned re-parameterized convolution

- RepConv 기반
  - 3x3 Conv, 1x1 Conv, Identity connection을 하나의 convolution layer로 합치는 방식
    - Identity connection(or mapping, shortcut) : 입력 값을 그대로 전달하는 방법, ResNet과 같은 Concatenation based 모델에서 사용됨

- RepConv안의 **Identity connection**(Concetenation)이 Resnet의 **residual, concatenation**을 파괴하는 것을 발견
  - **RepConv뒤에 Concatenation을 할 경우 성능 저하 발생**
- **identity connection없는 RepConv(RepConvN)인 planned re-parameterized convolution 설계**

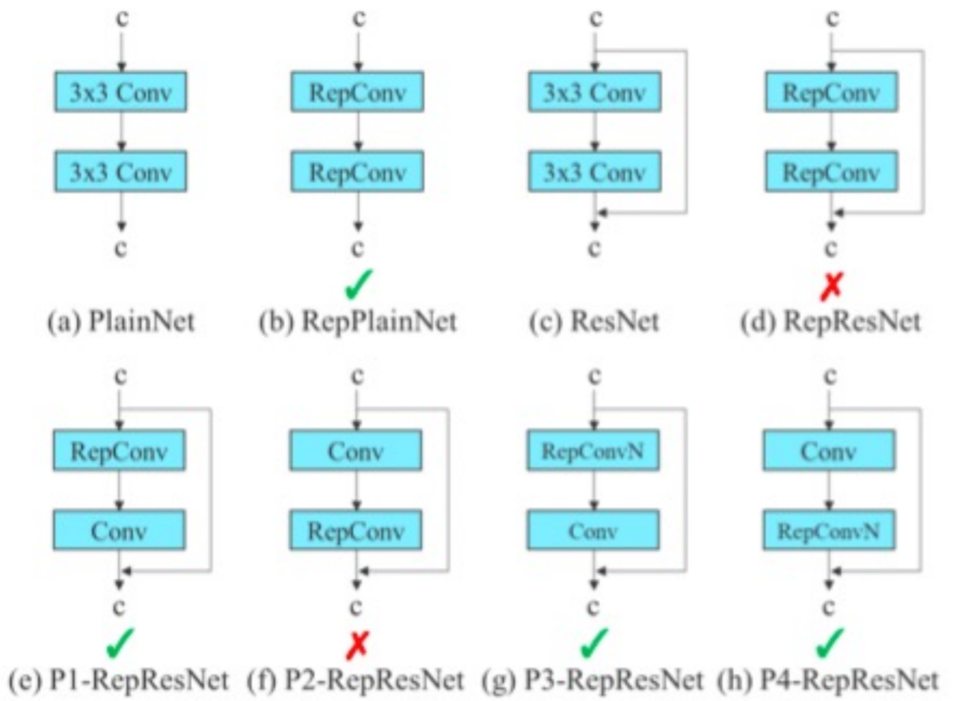
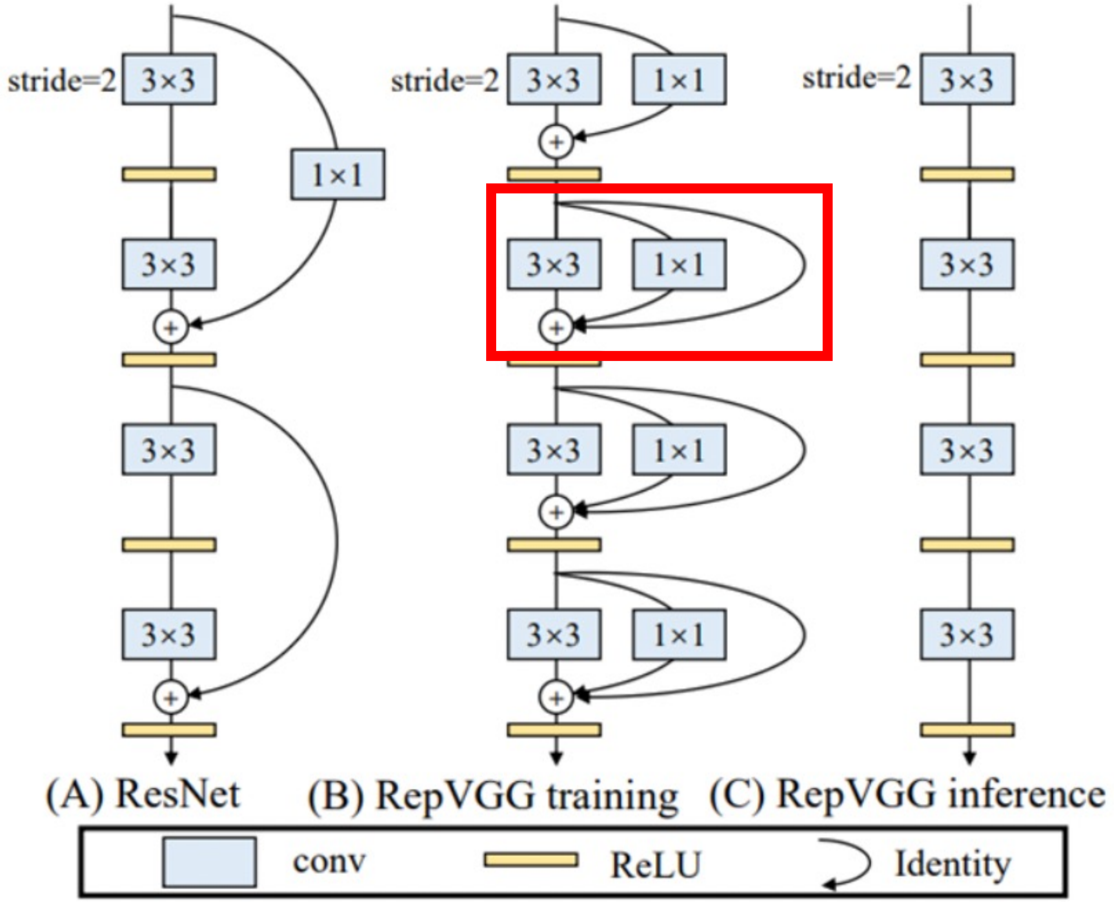


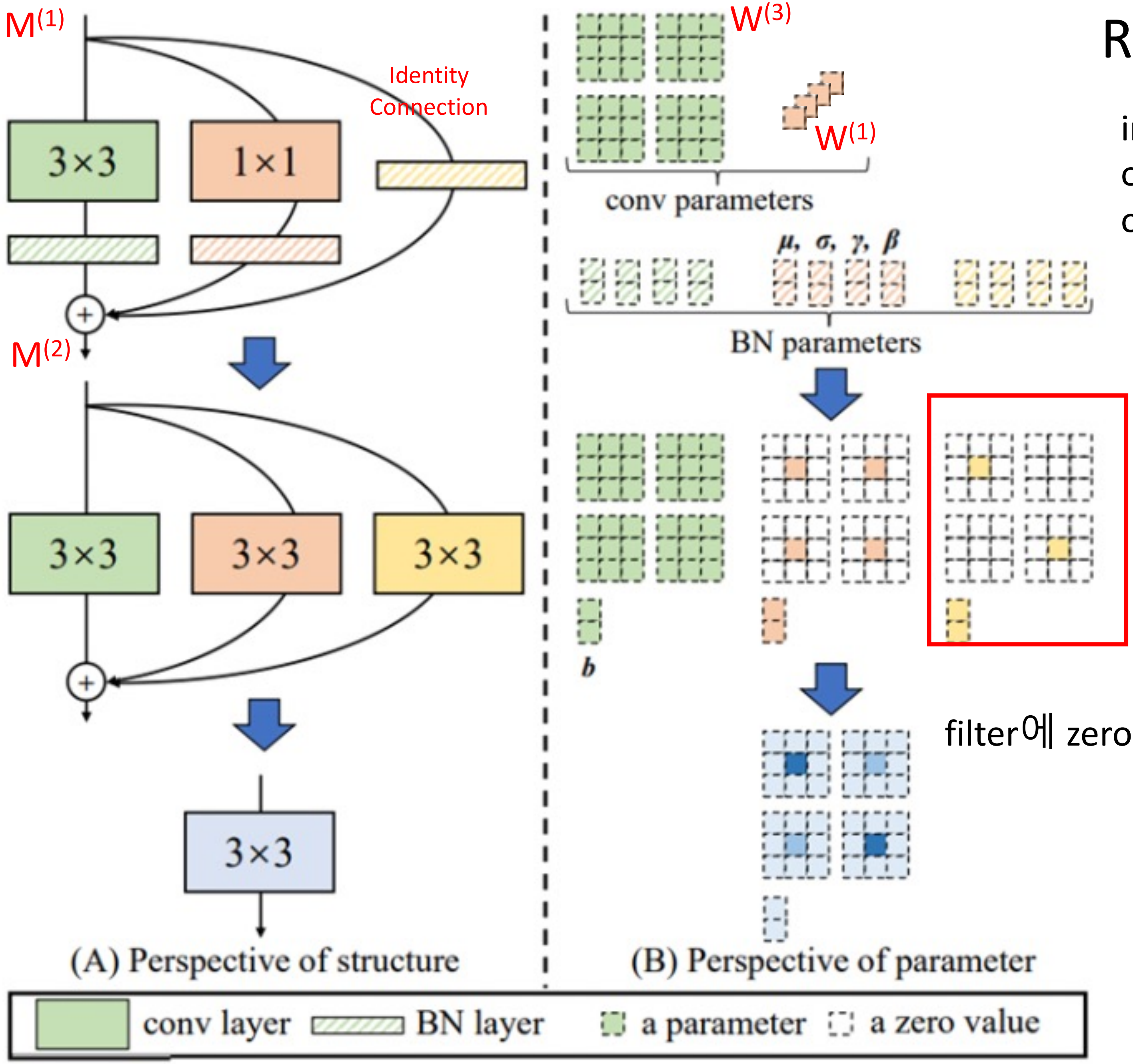
Figure 4: Planned re-parameterized model. In the proposed planned re-parameterized model, we found that a layer with residual or concatenation connections, its RepConv should not have identity connection. Under these circumstances, it can be replaced by RepConvN that contains no identity connections.

RepVGG

- ResNet의 형태에서 영감을 받은 Re-parameterized 모델
    - ResNet과 같은 concatenation 기반 모델의 가장 큰 문제점은 Inference 속도가 느리다는 것
    - VGG와 같은 모델은 Inference 속도는 빠르지만 낮은 정확도
    - RepVGG는 ResNet과 VGG의 장점을 결합한 모델
- Training시에는 ResNet과 같이 여러 Identity connection을 사용하고 Inference시에는 Convolution Filter(3x3,1x1)과 Identity connection을 하나의 3x3 Convolution Filter로 통합
    - 3x3 Filter이 대부분의 GPU에서 가장 빠른 연산속도를 보여주기 때문에 모두 3x3 Filter 로 통합



RepVGG re-parameterized 과정



input channel : 2
output channel : 2
output 채널 개수(c2=2)만큼 BN parameter 계산됨

indentity Connection :
feature map을 그대로 유지해주는 연산

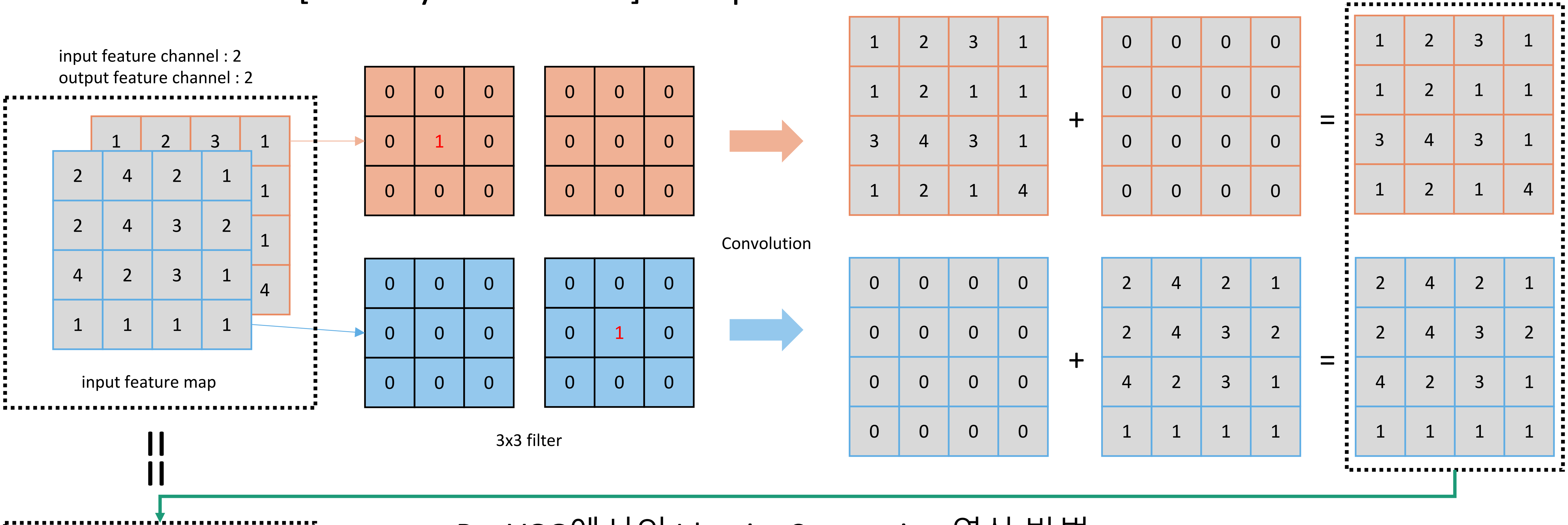
filter에 zero-padding을 가하여 3x3 conv로 재구성

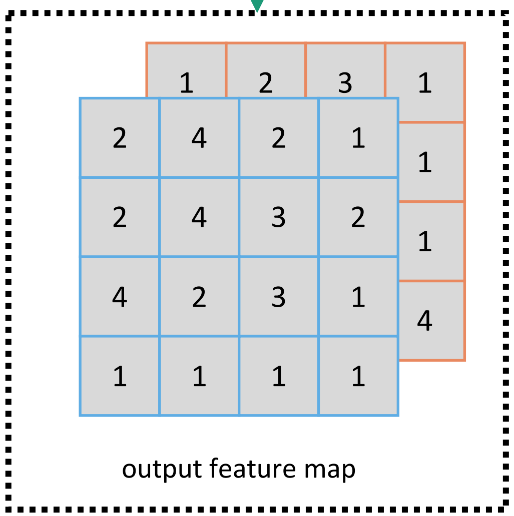
$$
\begin{aligned}
\mathrm{M}^{(2)} &= \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(3)}, \boldsymbol{\mu}^{(3)}, \boldsymbol{\sigma}^{(3)}, \boldsymbol{\gamma}^{(3)}, \boldsymbol{\beta}^{(3)}) \\
&+ \mathrm{bn}(\mathrm{M}^{(1)} * \mathrm{W}^{(1)}, \boldsymbol{\mu}^{(1)}, \boldsymbol{\sigma}^{(1)}, \boldsymbol{\gamma}^{(1)}, \boldsymbol{\beta}^{(1)}) \\
&+ \mathrm{bn}(\mathrm{M}^{(1)}, \boldsymbol{\mu}^{(0)}, \boldsymbol{\sigma}^{(0)}, \boldsymbol{\gamma}^{(0)}, \boldsymbol{\beta}^{(0)}) \,.
\end{aligned}
$$

$$
\mathrm{bn}(\mathrm{M}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta})_{:,i,:,:} = (\mathrm{M}_{:,i,:,:} - \boldsymbol{\mu}_i)\frac{\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i} + \boldsymbol{\beta}_i \,.
$$

$$
\mathrm{W}'_{i,:,:,:} = \frac{\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i}\mathrm{W}_{i,:,:,:} \,, \quad \mathbf{b}'_i = -\frac{\boldsymbol{\mu}_i \boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i} + \boldsymbol{\beta}_i \,.
$$

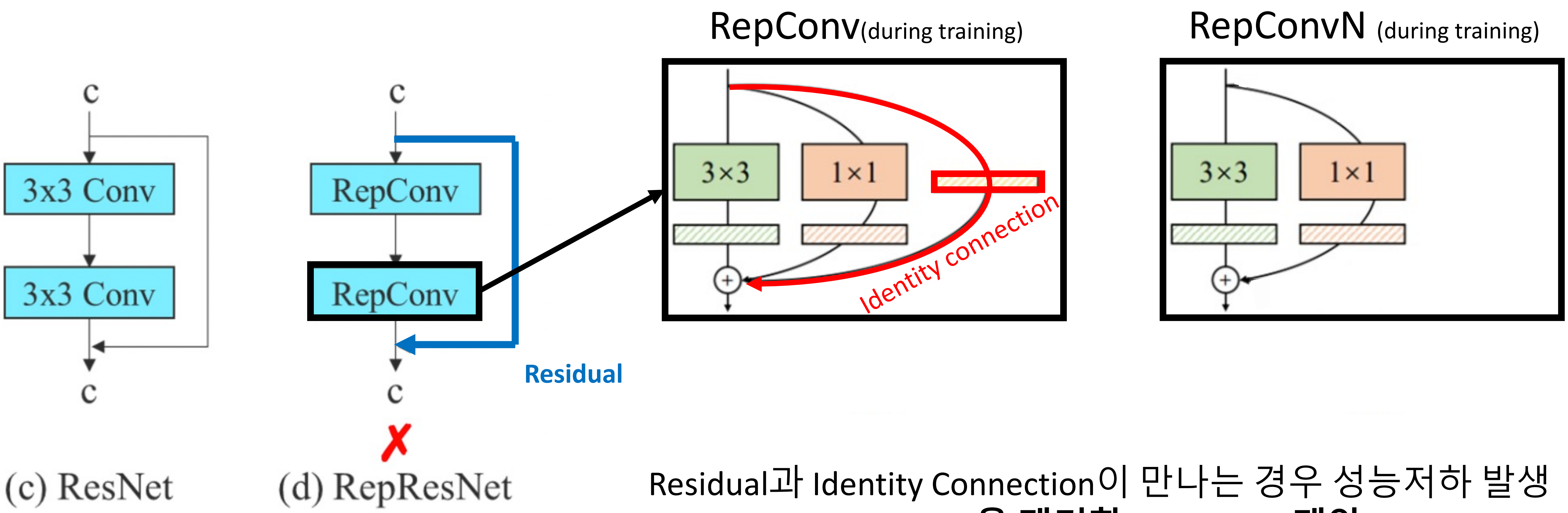
How to calculate [Identity Connection] in RepVGG





- RepVGG에서의 Identity Connection 연산 방법
- 1x1 convolution filter에 zero-padding을 통해 3x3 filter로 만듦
- output channel 만큼의 filter의 한 channel을 제외하고 나머지 channel은 0값을 가지도록 설정
    - 입력 feature map의 값을 그대로 살리기 위함

Trainable bag-of-freebies
- Planned re-parameterized convolution



Residual과 Identity Connection이 만나는 경우 성능저하 발생
-> **Identity Connection을 제거한 RepConvN 제안**

Trainable bag-of-freebies
- Planned re-parameterized convolution

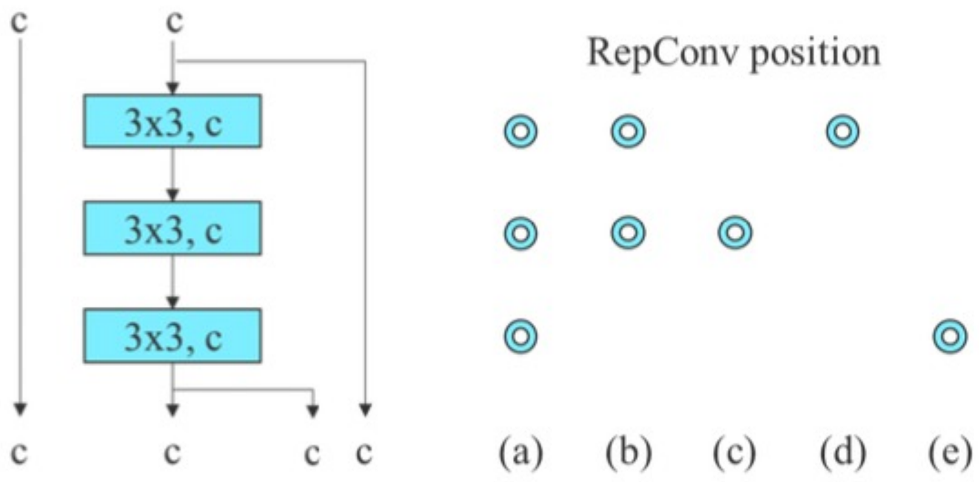


Figure 6: Planned RepConv 3-stacked ELAN. Blue circles are the position we replace Conv by RepConv.

Ablation study 1

- ELAN의 Computation block에서 RepConv를 다양하게 위치시키면서 성능 평가
- (c) 케이스가 전반적으로 가장 좋은 성능 도출

Table 4: Ablation study on planned RepConcatenation model.

Model	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
base (3-S ELAN)	52.26%	70.41%	56.77%	35.81%	57.00%	67.59%
Figure 6 (a)	52.18%	70.34%	56.90%	35.71%	56.83%	67.51%
Figure 6 (b)	52.30%	70.30%	**56.92%**	35.76%	56.95%	67.74%
Figure 6 (c)	**52.33%**	**70.56%**	56.91%	**35.90%**	**57.06%**	67.50%
Figure 6 (d)	52.17%	70.32%	56.82%	35.33%	**57.06%**	**68.09%**
Figure 6 (e)	52.23%	70.20%	56.81%	35.34%	56.97%	66.88%

Trainable bag-of-freebies
- Planned re-parameterized convolution

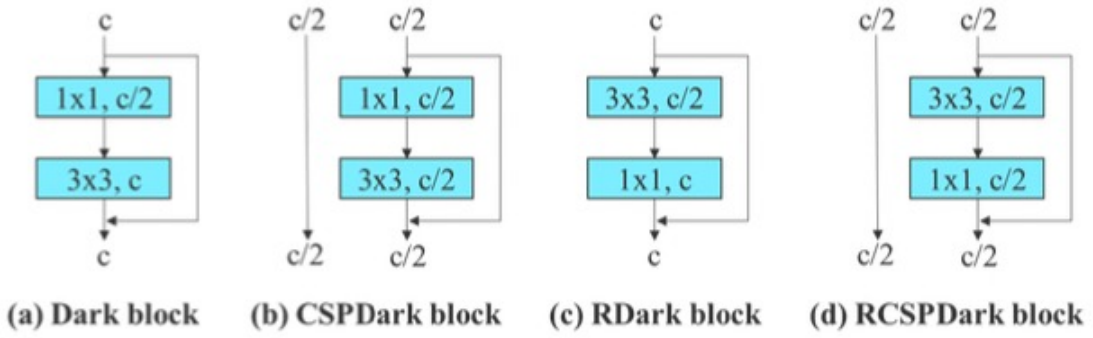


Figure 7: Reversed CSPDarknet. We reverse the position of 1×1 and 3×3 convolutional layer in dark block to fit our planned re-parameterized model design strategy.

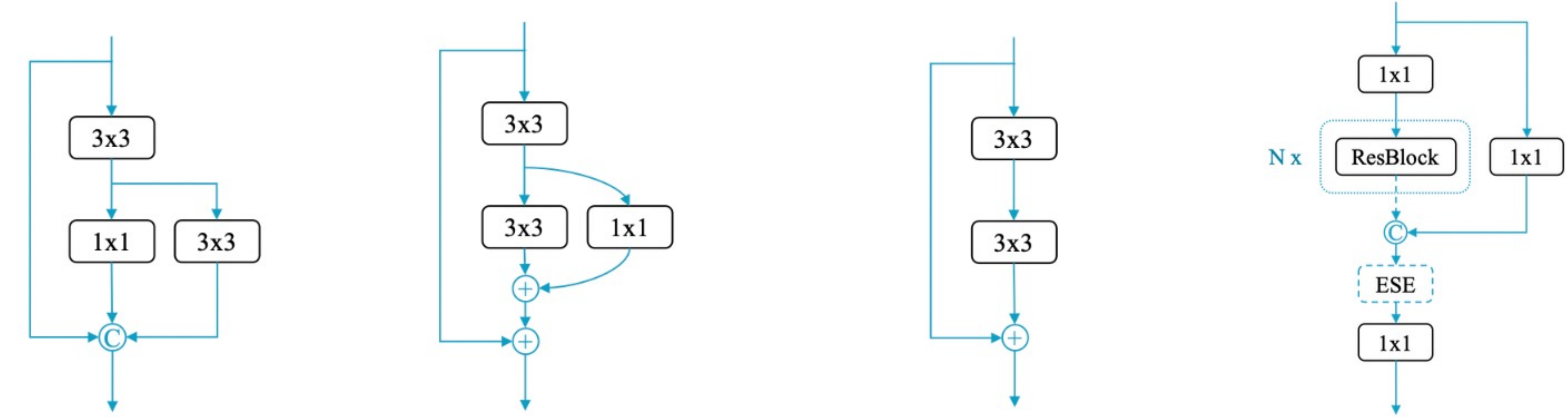
Ablation study 2

- original DarkBlock의 경우 Planned RepConv 전략에 맞지 않기 때문에 Reverse Dark block 설계
    - 1x1 conv와 3x3 conv의 위치를 바꾼 형태
- PPYOLO 와 동일한 컨셉

Table 5: Ablation study on planned RepResidual model.

Model	$\mathbf{AP}^{val}$	$\mathbf{AP}_{50}^{val}$	$\mathbf{AP}_{75}^{val}$	$\mathbf{AP}_{S}^{val}$	$\mathbf{AP}_{M}^{val}$	$\mathbf{AP}_{L}^{val}$
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%**

PPYOLO



(a) Simplified TreeBlock (b) Our RepResBlock during training (c) Our RepResBlock during inference (d) Our CSPRepResStage

hard label : 0 1 0 0
soft label : 0.1 0.7 0.1 0.1

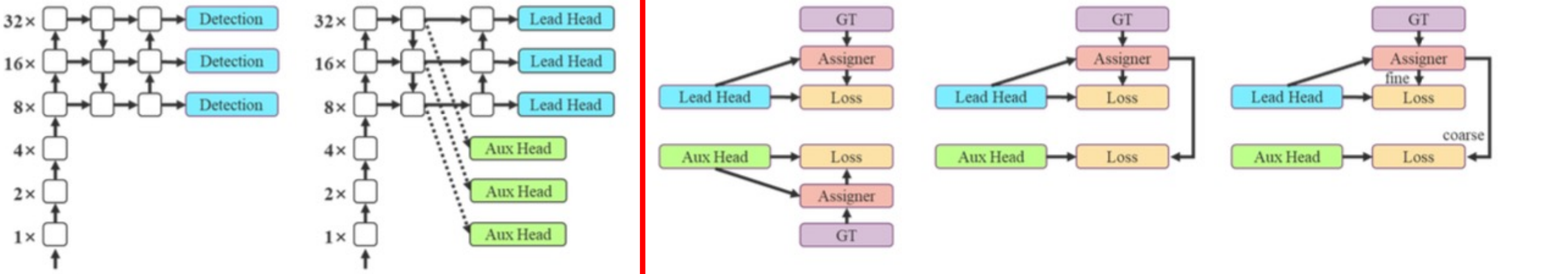
Trainable bag-of-freebies
- Proposed assistant loss for auxiliary head

- lead head : prediction을 뽑는 메인 layer
- auxiliary(aux) head : layer 중간에 위치한 보조 head
- aux head를 사용하는 방법 중 Independent assigner가 가장 보편적인 방식
  - aux, lead head 모두 GT를 통해 hard label을 생성(predict 후 학습하는 과정)
- 최근 연구에서는 네트워크의 예측 결과의 distribution(or probability)를 통해 loss를 계산하는 soft label을 사용
- 최신연구에서는 soft label을 aux head와 lead head에 어떻게 할당할 지에 대한 연구가 이뤄지지 않음 -> aux head가 GT를 참고하지 않는 2가지 label assigner 제안
  - Lead head guided label assigner
  - Coarse-to-fine lead head guided label assigner

Trainable bag-of-freebies
- Proposed assistant loss for auxiliary head

- Lead head guided label assigner : ground-truth 기반으로 계산되어 soft label 생성
    - lead head가 학습한 정보를 aux head가 직접 학습 -> lead head가 아직 학습되지 않음 residual information 학습에 집중 할 수 있음
- Coarse-to-fine lead head guided label assigner -> 의도적으로 aux head에 제약을 걸어 lead head의 성능을 넘지 못하게 함
    - Corse label(Aux head) : fine label 처럼 섬세한 label이 만들어지지 않게 함
    - Fine label(Lead head) : soft label
    - Lead head는 high precision, high recall이 가능, aux head는 recall 에 집중
    - aux head의 parameter가 lead head의 parameter와 유사해지면 성능이 하락

Proposed



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

Trainable bag-of-freebies
- Proposed assistant loss for auxiliary head

Ablation study

- aux head를 merging cardinality 전의 feature map 세트들 중 하나 다음에 붙히는 방식으로 접근
    - 이를 통해 assistant loss(by aux head)를 통해서 새롭게 만들어진 feature map들이 직접적으로 update 되는 것을 막아줌

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: 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

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

Trainable bag-of-freebies
- Proposed assistant loss for auxiliary head

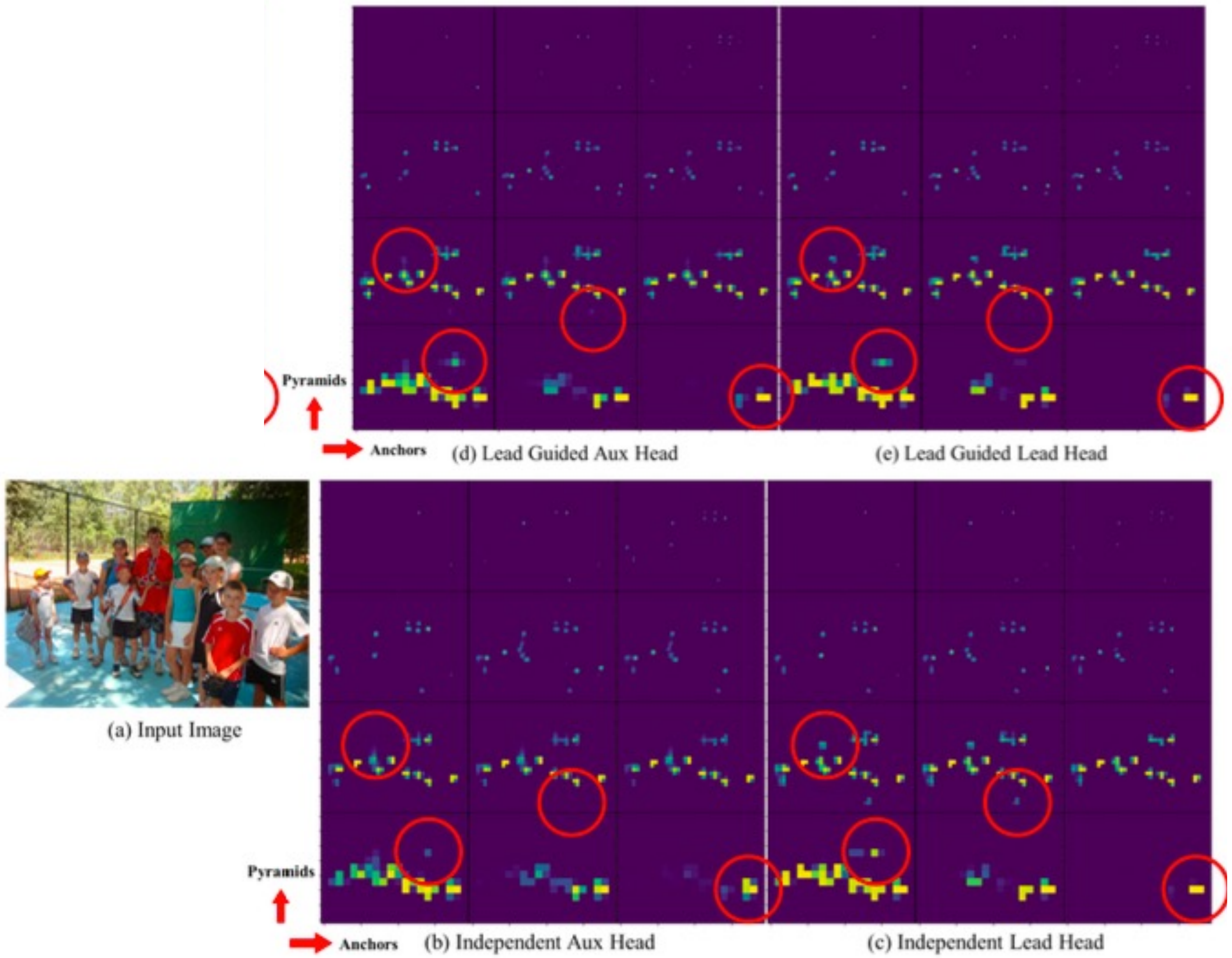
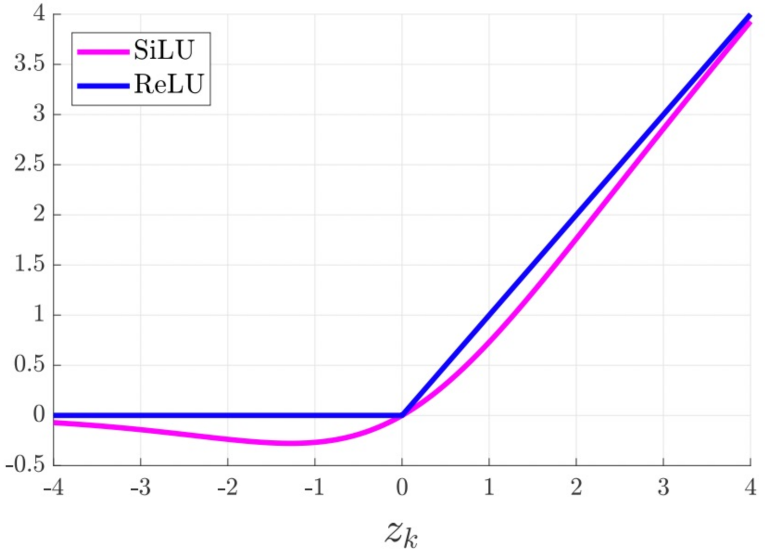


Figure 8: Objectness map predicted by different methods at auxiliary head and lead head.

Experiments



- Dataset : MS-COCO
- Not using Pre-trained model
- 다양한 GPU 환경에 사용할 수 있는 모델 설계
  - Edge GPU : YOLOv7-tiny
    - leaky-ReLU activation 사용(나머지 모델은 SiLU 사용)
  - Normal GPU : YOLOv7
    - compound scaling method를 사용하여 모델 전체 depth와 width에 대해서 scale up
  - cloud GPU : YOLOv7-w6
    - compound scaling method 사용
    - YOLOv7-E6, YOLOv7-D6
    - YOLOv7-E6E : YOLOv7-E6에서 E-ELAN을 사용
- YOLOv4, scaled-YOLOv4, YOLOR을 baseline으로 사용

Experiments

- YOLOv7-tiny-SiLU vs YOLOv5-N : 127 fps 더 빠름, 10.7% 더 정확함
  - YOLOv7의 경우 161fps에서 51.4%AP를 가짐
- YOLOv7 vs PPYOLOE-L : 41% 적은 파라미터 사용량
- YOLOv7-X vs YOLOv5-L
  - YOLOv7-X 's inference speed : 114fps
  - YOLOv5-L's inference speed : 99fps
  - YOLOv7의 AP가 3.9% 향상
- YOLOv7-X vs YOLOv5-X
  - 비슷한 스케일에 YOLOv5-X보다 31fps 더 빠름
  - 22% 적은 파라미터, 8%적은 계산량, 2.2% 높은 AP
- YOLOv7 vs YOLOR (1280 resolution)
  - YOLOv7-W6가 8fps 더 빠름
  - AP 1% 증가
- YOLOv7-E6 vs YOLOv5-X6
  - 0.9% AP 향상, 47% inference 속도 향상
- YOLOv7-D6 vs YOLOR-E6
  - 비슷한 inference 속도
  - AP 0.8% 향상
- YOLOv7-E6E vs YOLOR-D6
  - 비슷한 inference 속도
  - AP 0.3%향상

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

Experiments

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.

참고문헌

- 블로그
  - YOLOv7
    - https://velog.io/@parksh089g/연구실논문리뷰YOLOv7-Trainable-bag-of-freebies-sets-new-state-of-the-art-for-real-time-objectdetectors
    - https://da2so.tistory.com/59#---%25--Extended%25--efficient%25--layer%25--aggregation%25--networks
    - https://eehoeskrap.tistory.com/651
    - https://learnopencv.com/yolov7-object-detection-paper-explanation-and-inference/?utm_source=rss&utm_medium=rss&utm_campaign=yolov7-object-detection-paper-explanation-and-inference
  - RepVGG
    - https://mole-starseeker.tistory.com/87
    - https://velog.io/@wilko97/논문리뷰-RepVGG-Making-VGG-style-ConvNets-Great-Again-2017-CVPR
    - https://lv99.tistory.com/25

참고문헌

- 논문
  - YOLOv7
    - https://arxiv.org/abs/2207.02696
  - YOLOv4
    - https://arxiv.org/abs/2004.10934
  - scaled YOLOv4
    - https://arxiv.org/abs/2011.08036
  - RepVGG
    - https://arxiv.org/abs/2101.03697
  - PPYOLO
    - https://arxiv.org/abs/2104.10419

감사합니다