# 주간 보고서-14주차 (2024.06.03~ 2024.06.09)

### 팀명 : 무계획

### 회의 참석자 : 서태원, 신재환, 최승렬, 류은환

### 회의 일자 및 회의 장소 :

2024.06.03(월)

- 서태원
  - 회의 시간 : 11:00 ~ 15:00, 20:00 ~ 23:00
  - 회의 장소 : 7호관 302호
- 신재환
  - 회의 시간 : 20:00 ~ 23:40
  - 회의 장소 : 7호관 302호
- 최승렬
  - 회의 시간 : 14:40 ~ 23:40
  - 회의 장소 : 7호관 302호
- 류은환
  - 회의 시간 : 20:00 ~ 23:30
  - 회의 장소 : 7호관 302호

2024.06.04(화)

- 서태원
  - 회의 시간 : 22:00 ~ 24:00
  - 회의 장소 : 7호관 302호
- 신재환
  - 회의 시간 : 11:30 ~ 12:30, 17:10 ~ 23:40
  - 회의  장소 : 7호관 302호
- 최승렬
  - 회의 시간 : 16:45 ~ 00:15
  - 회의 장소 : 7호관 302호
- 류은환
  - 회의 시간 : 17:00 ~ 23:30
  - 회의 장소 : 7호관 302호

2024.06.05(수)

- 서태원
  - 회의 시간 : 16:00 ~ 22:30
  - 회의 장소 : 7호관 302호
- 신재환
  - 회의 시간 : 17:10 ~ 23:30
  - 회의 장소 : 7호관 302호

2024.06.06(목)

- 신재환
  - 회의 시간 : 13:25 ~ 23:40
  - 회의 장소 : 7호관 302호
- 최승렬
  - 회의 시간 : 14:45 ~ 23:35
  - 회의 장소 : 7호관 302호

- 최승렬
  - 회의 시간 : 14:10 ~ 23:30
  - 회의 장소 : 7호관 302호
- 류은환
  - 회의 시간 : 14:30 ~ 17:30, 20:00 ~ 23:30
  - 회의 장소 : 7호관 302호

2024.06.07(금)

- 서태원
  - 회의 시간 : 22:10 ~ 23:40
  - 회의 장소 : 디스코드 음성채널
- 신재환
  - 회의 시간 : 15:00 ~ 23:40
  - 회의 장소 : 7호관 302호, 디스코드 음성채널
- 최승렬
  - 회의 시간 : 15:00 ~ 23:35
  - 회의 장소 : 7호관 302호, 디스코드 음성채널
- 류은환
  - 회의 시간 : 15:00 ~ 23:30
  - 회의 장소 : 7호관 302호, 디스코드 음성채널

2024.06.09(일)

- 최승렬
  - 회의 시간 : 14:10 ~ 21:10
  - 회의 장소 : 7호관 302호
- 류은환
  - 회의 시간 : 15:30 ~ 23:30
  - 회의 장소 : 7호관 302호

2024.06.08(토)

- 신재환
  - 회의 시간 : 13:20 ~ 23:20
  - 회의 장소 : 7호관 302호
- 최승렬
  - 회의 시간 : 12:45 ~ 15:45, 19:25 ~ 23:20
  - 회의 장소 : 7호관 302호

총 활동 시간 :

- 서태원 : 19시간 30분
- 신재환 : 46시간 25분
- 최승렬 : 57시간 10분
- 류은환 : 33시간

## 진행 사항 :

### 현재 역할

서태원

**[ 업무 ]**

모듈 대체 관련 자료 및 근거 조사

RepNCSPELAN4, Bottleneck 학습

신재환

**[ 업무 ]**

PTQ, PTQ 모델 체크포인트 맞추기

QAT

류은환

**[ 업무 ]**

모듈 대체 관련 자료 및 근거 조사

수정 모듈 준비 및 코드 수정

함수 파라미터 분석

최승렬

**[ 업무 ]**

RepNCSPELAN4 내부 conv 중요도 측정

conv내의 필터 중요도 측정

conv 레이어 필터 pruning

---

# Test11

- RepNCSPELAN4의 cv2.Sequential, cv3.Sequential을 Conv로 수정하였음
- 보조분기 제거 (Converted)
- LeakyReLU 적용(yolov9-c.yaml 파일에 parameters 부분에 activation: nn.LeakyReLU(0.1)를 추가)
- 비율에 맞추어 레이어 크기 감소

yolov9-c.yaml

```
# YOLOv9

# parameters
nc: 80  # number of classes
depth_multiple: 1.0  # model depth multiple
width_multiple: 1.0  # layer channel multiple
activation: nn.LeakyReLU(0.1)
#activation: nn.ReLU()

# anchors
anchors: 3

# YOLOv9 backbone
backbone:
  [
   [-1, 1, Silence, []],

   # conv down
   [-1, 1, Conv, [32, 3, 2]],  # 1-P1/2

   # conv down
   [-1, 1, Conv, [64, 3, 2]],  # 2-P2/4

   # elan-1 block
   [-1, 1, RepNCSPELAN4, [128, 64, 32, 1]],  # 3

   # avg-conv down
   [-1, 1, ADown, [128]],  # 4-P3/8

   # elan-2 block
   [-1, 1, RepNCSPELAN4, [256, 128, 64, 1]],  # 5

   # avg-conv down
   [-1, 1, ADown, [256]],  # 6-P4/16

   # elan-2 block
   [-1, 1, RepNCSPELAN4, [256, 256, 128, 1]],  # 7

   # avg-conv down
   [-1, 1, ADown, [256]],  # 8-P5/32

   # elan-2 block
```

```
    [-1, 1, RepNCSPELAN4, [256, 256, 128, 1]],  # 9
  ]

# YOLOv9 head
head:
  [
   # elan-spp block
   [-1, 1, SPPELAN, [256, 128]],  # 10

   # up-concat merge
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 7], 1, Concat, [1]],  # cat backbone P4

   # elan-2 block
   [-1, 1, RepNCSPELAN4, [256, 256, 128, 1]],  # 13

   # up-concat merge
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 5], 1, Concat, [1]],  # cat backbone P3

   # elan-2 block
   [-1, 1, RepNCSPELAN4, [128, 128, 64, 1]],  # 16 (P3/8-small)

   # avg-conv-down merge
   [-1, 1, ADown, [128]],
   [[-1, 13], 1, Concat, [1]],  # cat head P4

   # elan-2 block
   [-1, 1, RepNCSPELAN4, [256, 256, 128, 1]],  # 19 (P4/16-medium)

   # avg-conv-down merge
   [-1, 1, ADown, [256]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5

   # elan-2 block
   [-1, 1, RepNCSPELAN4, [256, 256, 128, 1]],  # 22 (P5/32-large)


   # multi-level reversible auxiliary branch

   # routing
   [5, 1, CBLinear, [[128]]], # 23
   [7, 1, CBLinear, [[128, 256]]], # 24
```

```
    [9, 1, CBLinear, [[128, 256, 256]]], # 25

    # conv down
    [0, 1, Conv, [32, 3, 2]],  # 26-P1/2

    # conv down
    [-1, 1, Conv, [64, 3, 2]],  # 27-P2/4

    # elan-1 block
    [-1, 1, RepNCSPELAN4, [128, 64, 32, 1]],  # 28

    # avg-conv down fuse
    [-1, 1, ADown, [128]],  # 29-P3/8
    [[23, 24, 25, -1], 1, CBFuse, [[0, 0, 0]]], # 30

    # elan-2 block
    [-1, 1, RepNCSPELAN4, [256, 128, 64, 1]],  # 31

    # avg-conv down fuse
    [-1, 1, ADown, [256]],  # 32-P4/16
    [[24, 25, -1], 1, CBFuse, [[1, 1]]], # 33

    # elan-2 block
    [-1, 1, RepNCSPELAN4, [256, 256, 128, 1]],  # 34

    # avg-conv down fuse
    [-1, 1, ADown, [256]],  # 35-P5/32
    [[25, -1], 1, CBFuse, [[2]]], # 36

    # elan-2 block
    [-1, 1, RepNCSPELAN4, [256, 256, 128, 1]],  # 37



    # detection head

    # detect
    [[31, 34, 37, 16, 19, 22], 1, DualDDetect, [nc]],  # DualDDetect(A
   ]
```

361 layers, 11315520 parameters, 11315488 gradients, 54.5 GFLOPs

**test10보다 inference time이 감소된 것을 확인할 수 있음.**

| MODEL NAME | test11-converted_COCO_500 |
|---|---|
| mAP50 score: | 0.58 |
| mAP50-95 score: | 0.418 |
| RPI avg inference time: | 816.07ms |
| RPI max inference time: | 867.64ms |
| RPI min inference time: | 805.6ms |
| RPI avg CPU temp: | 67.16C |
| PRAMS: | 5362432 |
| GFLOPS: | 22.2 |
| LAYERS: | 162 |

| MODEL NAME | test11-converted_COCO_500_onnx |
|---|---|
| RPI avg inference time: | 383.42ms |
| RPI max inference time: | 547.74ms |
| RPI min inference time: | 373.56ms |
| RPI avg CPU temp: | 70.83C |

## auxiliary branch

보조분기는 수정하지 않고 main만 수정

yolov9-branch

coco128 500에폭으로 테스트

| c_128_500 | c_128_500_converted |
|---|---|
| mAP50: 0.804 | mAP50: 0.803 |
| mAP50-95: 0.615 | mAP50-95: 0.628 |

보조분기에서 main으로 넘겨주는 파라미터 크기가 맞지 않음으로 학습을 할 수 없음

RuntimeError: Given groups=1, weight of size [64, 256, 1, 1], expected input[1, 128, 64, 64] to have 256 channels, but got 128 channels instead

## 여러 보조분기 사용

train_triple을 돌리는 과정에서 val_triple에서 문제가 생겨 train_tripe에 validate 부분을 모두 삭제 후 학습

| YOLOv9-c-triple-converted |
| --- |
| mAP50: 0.696 |
| mAP50-95: 0.505 |

모델이 더 큰 모델이고 train시 validate를 사용할 수 없기 때문에 오히려 mAP가 낮아진 것으로 보임

YOLOv9-c-triple

```
# YOLOv9

# parameters
nc: 80  # number of classes
depth_multiple: 1.0  # model depth multiple
width_multiple: 1.0  # layer channel multiple
#activation: nn.LeakyReLU(0.1)
#activation: nn.ReLU()

# anchors
anchors: 3

# YOLOv9 backbone
backbone:
  [
   [-1, 1, Silence, []],

   # conv down
   [-1, 1, Conv, [64, 3, 2]],  # 1-P1/2

   # conv down
   [-1, 1, Conv, [128, 3, 2]],  # 2-P2/4

   # elan-1 block
   [-1, 1, RepNCSPELAN4, [256, 128, 64, 1]],  # 3

   # avg-conv down
   [-1, 1, ADown, [256]],  # 4-P3/8

   # elan-2 block
```

```
    [-1, 1, RepNCSPELAN4, [512, 256, 128, 1]],  # 5

    # avg-conv down
    [-1, 1, ADown, [512]],  # 6-P4/16

    # elan-2 block
    [-1, 1, RepNCSPELAN4, [512, 512, 256, 1]],  # 7

    # avg-conv down
    [-1, 1, ADown, [512]],  # 8-P5/32

    # elan-2 block
    [-1, 1, RepNCSPELAN4, [512, 512, 256, 1]],  # 9
   ]

# YOLOv9 head
head:
  [
   # elan-spp block
   [-1, 1, SPPELAN, [512, 256]],  # 10

   # up-concat merge
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 7], 1, Concat, [1]],  # cat backbone P4

   # elan-2 block
   [-1, 1, RepNCSPELAN4, [512, 512, 256, 1]],  # 13

   # up-concat merge
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 5], 1, Concat, [1]],  # cat backbone P3

   # elan-2 block
   [-1, 1, RepNCSPELAN4, [256, 256, 128, 1]],  # 16 (P3/8-small)

   # avg-conv-down merge
   [-1, 1, ADown, [256]],
   [[-1, 13], 1, Concat, [1]],  # cat head P4

   # elan-2 block
   [-1, 1, RepNCSPELAN4, [512, 512, 256, 1]],  # 19 (P4/16-medium)

   # avg-conv-down merge
```

```
    [-1, 1, ADown, [512]],
    [[-1, 10], 1, Concat, [1]],  # cat head P5

    # elan-2 block
    [-1, 1, RepNCSPELAN4, [512, 512, 256, 1]],  # 22 (P5/32-large)


    # multi-level reversible auxiliary branch

    # routing
    [5, 1, CBLinear, [[256]]], # 23
    [7, 1, CBLinear, [[256, 512]]], # 24
    [9, 1, CBLinear, [[256, 512, 512]]], # 25

    # conv down
    [0, 1, Conv, [64, 3, 2]],  # 26-P1/2

    # conv down
    [-1, 1, Conv, [128, 3, 2]],  # 27-P2/4

    # elan-1 block
    [-1, 1, RepNCSPELAN4, [256, 128, 64, 1]],  # 28

    # avg-conv down fuse
    [-1, 1, ADown, [256]],  # 29-P3/8
    [[23, 24, 25, -1], 1, CBFuse, [[0, 0, 0]]], # 30

    # elan-2 block
    [-1, 1, RepNCSPELAN4, [512, 256, 128, 1]],  # 31

    # avg-conv down fuse
    [-1, 1, ADown, [512]],  # 32-P4/16
    [[24, 25, -1], 1, CBFuse, [[1, 1]]], # 33

    # elan-2 block
    [-1, 1, RepNCSPELAN4, [512, 512, 256, 1]],  # 34

    # avg-conv down fuse
    [-1, 1, ADown, [512]],  # 35-P5/32
    [[25, -1], 1, CBFuse, [[2]]], # 36

    # elan-2 block
    [-1, 1, RepNCSPELAN4, [512, 512, 256, 1]],  # 37
```

```
# multi-level reversible auxiliary branch

 # routing
 [5, 1, CBLinear, [[256]]], # 38
 [7, 1, CBLinear, [[256, 512]]], # 39
 [9, 1, CBLinear, [[256, 512, 512]]], # 40

 # conv down
 [0, 1, Conv, [64, 3, 2]],  # 41-P1/2

 # conv down
 [-1, 1, Conv, [128, 3, 2]],  # 42-P2/4

 # elan-1 block
 [-1, 1, RepNCSPELAN4, [256, 128, 64, 1]],  # 43

 # avg-conv down fuse
 [-1, 1, ADown, [256]],  # 44-P3/8
 [[38, 39, 40, -1], 1, CBFuse, [[0, 0, 0]]], # 45

 # elan-2 block
 [-1, 1, RepNCSPELAN4, [512, 256, 128, 1]],  # 46

 # avg-conv down fuse
 [-1, 1, ADown, [512]],  # 47-P4/16
 [[39, 40, -1], 1, CBFuse, [[1, 1]]], # 48

 # elan-2 block
 [-1, 1, RepNCSPELAN4, [512, 512, 256, 1]],  # 49

 # avg-conv down fuse
 [-1, 1, ADown, [512]],  # 50-P5/32
 [[40, -1], 1, CBFuse, [[2]]], # 51

 # elan-2 block
 [-1, 1, RepNCSPELAN4, [512, 512, 256, 1]],  # 52

 # detection head

 # detect
```

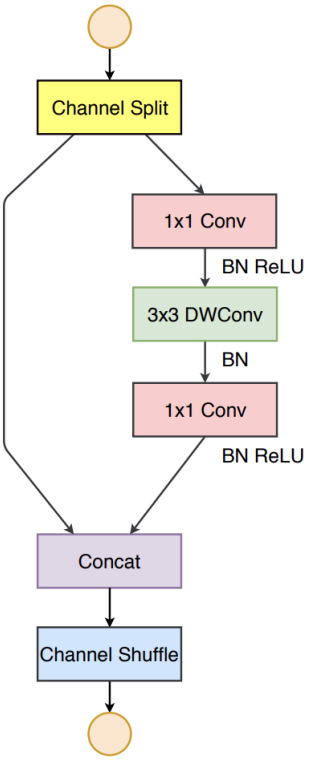
```
    [[ 31, 34, 37, 46, 49, 52, 16, 19, 22], 1, TripleDDetect, [nc]],
  ]
```

reparameterization

```
import torch
from models.yolo import Model

device = torch.device("cuda")
cfg = "./models/detect/gelan-c.yaml"
model = Model(cfg, ch=3, nc=71, anchors=3)
#model = model.half()
model = model.to(device)
_ = model.eval()
ckpt = torch.load('/home/mgh/MGH/yolov9-branch/best.pt', map_locatio
model.names = ckpt['model'].names
model.nc = ckpt['model'].nc

idx = 0
for k, v in model.state_dict().items():
    if "model.{}.".format(idx) in k:
        if idx < 22:
            kr = k.replace("model.{}.".format(idx), "model.{}.form
            model.state_dict()[k] -= model.state_dict()[k]
            model.state_dict()[k] += ckpt['model'].state_dict()[kr]
        elif "model.{}.cv2.".format(idx) in k:
            kr = k.replace("model.{}.cv2.".format(idx), "model.{}.cv
            model.state_dict()[k] -= model.state_dict()[k]
            model.state_dict()[k] += ckpt['model'].state_dict()[kr]
        elif "model.{}.cv3.".format(idx) in k:
            kr = k.replace("model.{}.cv3.".format(idx), "model.{}.cv
            model.state_dict()[k] -= model.state_dict()[k]
            model.state_dict()[k] += ckpt['model'].state_dict()[kr]
        elif "model.{}.dfl.".format(idx) in k:
            kr = k.replace("model.{}.dfl.".format(idx), "model.{}.df
            model.state_dict()[k] -= model.state_dict()[k]
            model.state_dict()[k] += ckpt['model'].state_dict()[kr]
    else:
        while True:
            idx += 1
            if "model.{}.".format(idx) in k:
```

```
                break
        if idx < 22:
            kr = k.replace("model.{}.".format(idx), "model.{}.form
            model.state_dict()[k] -= model.state_dict()[k]
            model.state_dict()[k] += ckpt['model'].state_dict()[kr]
        elif "model.{}.cv2.".format(idx) in k:
            kr = k.replace("model.{}.cv2.".format(idx), "model.{}.cv
            model.state_dict()[k] -= model.state_dict()[k]
            model.state_dict()[k] += ckpt['model'].state_dict()[kr]
        elif "model.{}.cv3.".format(idx) in k:
            kr = k.replace("model.{}.cv3.".format(idx), "model.{}.cv
            model.state_dict()[k] -= model.state_dict()[k]
            model.state_dict()[k] += ckpt['model'].state_dict()[kr]
        elif "model.{}.dfl.".format(idx) in k:
            kr = k.replace("model.{}.dfl.".format(idx), "model.{}.df
            model.state_dict()[k] -= model.state_dict()[k]
            model.state_dict()[k] += ckpt['model'].state_dict()[kr]
_ = model.eval()

m_ckpt = {'model': model.half(),
          'optimizer': None,
          'best_fitness': None,
          'ema': None,
          'updates': None,
          'opt': None,
          'git': None,
          'date': None,
          'epoch': -1}
torch.save(m_ckpt, "./yolov9-converted.pt")
```

**ShuffleNetV2 블록에 대한 조사를 시행함.**




(c)

## Performance

Top-1 Accuracy: 모델이 예측한 가장 가능성이 높은(가장 높은 확률을 가진) 클래스가 실제 정답 클래스와 일치하는 경우의 비율

Top-5 Accuracy: 모델이 예측한 상위 5개의 클래스 중 하나가 실제 정답 클래스와 일치하는 경우의 비율

Top-1 err: 모델이 예측한 가장 가능성이 높은 클래스가 실제 정답 클래스와 일치하지 않는 경우의 비율

ImageNet 2012 classification dataset

| Model | Complexity (MFLOPs) | Top-1 err. (%) | GPU Speed (Batches/sec.) | ARM Speed (Images/sec.) |
|---|---|---|---|---|
| ShuffleNet v2 0.5× (ours) | 41 | **39.7** | 417 | **57.0** |
| 0.25 MobileNet v1 [13] | 41 | 49.4 | **502** | 36.4 |
| 0.4 MobileNet v2 [14] (our impl.)* | 43 | 43.4 | 333 | 33.2 |
| 0.15 MobileNet v2 [14] (our impl.) | 39 | 55.1 | 351 | 33.6 |
| ShuffleNet v1 0.5× (g=3) [15] | 38 | 43.2 | 347 | 56.8 |
| DenseNet 0.5× [6] (our impl.) | 42 | 58.6 | 366 | 39.7 |
| Xception 0.5× [12] (our impl.) | 40 | 44.9 | 384 | 52.9 |
| IGCV2-0.25 [27] | 46 | 45.1 | 183 | 31.5 |
| ShuffleNet v2 1× (ours) | 146 | **30.6** | 341 | **24.4** |
| 0.5 MobileNet v1 [13] | 149 | 36.3 | **382** | 16.5 |
| 0.75 MobileNet v2 [14] (our impl.)** | 145 | 32.1 | 235 | 15.9 |
| 0.6 MobileNet v2 [14] (our impl.) | 141 | 33.3 | 249 | 14.9 |
| ShuffleNet v1 1× (g=3) [15] | 140 | 32.6 | 213 | 21.8 |
| DenseNet 1× [6] (our impl.) | 142 | 45.2 | 279 | 15.8 |
| Xception 1× [12] (our impl.) | 145 | 34.1 | 278 | 19.5 |
| IGCV2-0.5 [27] | 156 | 34.5 | 132 | 15.5 |
| IGCV3-D (0.7) [28] | 210 | 31.5 | 143 | 11.7 |

yolo는 상대적으로 크기가 작은 모델임 (파라미터 수가 적음)

크기가 작은 모델일수록, 정보 손실이 치명적으로 작용함

이 문제를 해결하기 위해, 가역함수를 도입함

YOLOv7의 핵심

→ trainable bag-of-freebies**,** 훈련 프로세스를 최적화

yoloV7의 문제점

→ 정보 병목 현상 (infomation bottleneck)

→ feed-foward 프로세스에서 정보 손실 문제가 발생

이 문제는 네트워크의 downscaling operations에 의해 입력 데이터의 중요 정보가 손실되는 것을 말함

yolov9은 이걸 해결함.

기존의 infomation bottleneck 해결 방법

→ **reversible architectures (가역적 아키텍쳐)**

→ **masked modeling (마스크 모델링)**

→ **deep supervision (심층 감독)**

기존 방법의 한계

→ 실시간 객체 감지, yolo 시리즈

→ 소규모 모델 아키텍처에선 덜 효과적

Infomation bottleneck 문제를 직접적으로 해결하고, 객체 감지의 정확성과 효율성을 향상하기 위해 두가지 기술 도입

→ PGI **(Programmable Gradient Information)**

→ GELAN **(Generalized Efficient Layer Aggregation Network)**

## YOLOv9의 핵심 구성

- **The Information Bottleneck Principle**
- **Reversible Functions**
- **Programmable Gradient Information (PGI)**
- **Generalized Efficient Layer Aggregate Network (GELAN)**

첫번째로 생각해야 할 문제

- 정보 병목 현상
  - 신경망의 구성에 있어서, 데이터가 한 계층을 통과하면, 정보가 줄어들 수 밖에 없음
  - 이 줄어드는 것을 정보 병목이라고 함

두번째로 생각해야 할 문제

- 정보 병목을 해결하는 이론적 방법은 가역 함수임. 이를 어떻게 활용해서 해결할 수 있는지

일단 가역함수부터 이해해보면

[ 입력과 출력 간에 양방향 매핑이 가능함 ] → 역함수 존재

즉, 출력된 결과를 보고, 다시 입력 데이터를 만들어 낼 수 있음

핵심은, 입력 데이터가 뭔지 완벽하게 다시 알아낼 수 있다는 것.

가역 함수를 이용하면, 네트워크가 모든 레이어에 대해 입력 정보를 유지할 수 있음

이걸 기울기 계산에 사용 → 기울기 계산할 때 입력 데이터랑 비교하면서 계산함

그래서 모델 업데이트 할 때 좋다는 의미(역전파)

역전파는 **출력층에서 발생한 오차를 입력층까지 전파하여 가중치를 업데이트** 하는 것

**역전파 (Backward Propagation)**

- 출력층에서 시작하여 입력층까지 오차를 역방향으로 전파
- 각 가중치의 변화량(그라디언트)을 계산하여 가중치를 업데이트
- 이 과정에서 체인 룰(chain rule)을 사용하여 각 층의 그라디언트를 계산

역전파 시, 입력 데이터의 정확한 정보를 알고 있으면 오차 계산과 가중치 업데이트의 정확성을 높일 수 있음.

정리하자면

입력1 → 레이어2 → 출력1, 이 출력1이 다시 입력2 → 레이어2 → 출력2 인데

출력 2에서 입력 2 계산 가능, 입력 2는 출력 1, 출력 1에서 입력 1 계산 가능함.

따라서, 모든 레이어에서 원래 입력 값이 뭔지 알 수 있음.

입력 값이 뭔지 알 수 있다는 것은

입력 데이터 보존 → **정보 손실이 없다는 의미**

정보 손실이 없다 → **그라디언트 계산 시, 정보 손실이 없음**

하지만 그냥 가역함수로 구성해버리면, 경량 모델에서 저장 공간의 한계로 인해 원시 데이터를 보존하기 어려워짐

따라서 PGI를 도입함 → 얕은 경량 신경망에 적합한 심층 신경망 훈련 방법

## PGI

- a **main branch for inference**, (추론을 위한 기본 분기)
- an **auxiliary reversible branch for reliable gradient calculation**, (기울기 계산을 위한 보조 가역 분기)
- multi-level auxiliary information to tackle deep supervision issues effectively without adding extra inference costs. (다단계 보조 정보)

→ 이건 모델 구조 전체에 대한 내용

→ 보조 분기를 이용해서 완전한 데이터 보존, 정보 손실 극복

## GELAN (일반화된 효율적인 계층 집합 네트워크)

GELAN은 PGI 프레임워크에 맞는 고유한 디자인으로 설계됨

- CSPNet’s gradient path planning
- ELAN’s inference speed optimizations
- 두가지 구조를 결합함

```
# GELAN

class SPPELAN(nn.Module):
    # spp-elan
    def __init__(self, c1, c2, c3):  # ch_in, ch_out, number, shortc
        super().__init__()
        self.c = c3
        self.cv1 = Conv(c1, c3, 1, 1)
        self.cv2 = SP(5)
        self.cv3 = SP(5)
        self.cv4 = SP(5)
        self.cv5 = Conv(4*c3, c2, 1, 1)

    def forward(self, x):
        y = [self.cv1(x)]
        y.extend(m(y[-1]) for m in [self.cv2, self.cv3, self.cv4])
        return self.cv5(torch.cat(y, 1))

class SP(nn.Module):
    def __init__(self, k=3, s=1):
        super(SP, self).__init__()
        self.m = nn.MaxPool2d(kernel_size=k, stride=s, padding=k //

    def forward(self, x):
        return self.m(x)
```

- SPPELAN 모듈
  - **Spatial Pyramid Pooling** (SPP)
  - ELAN 구조 내에 SPP를 통합함
  - It starts with a convolutional layer that adjusts the channel dimensions, followed by a series of spatial pooling operations to capture **multi-scale contextual**

**information**.

- 이는 채널 크기를 조정하는 컨볼루셔널 레이어로 시작하여 **다중 규모 상황 정보를** 캡처하기 위한 일련의 공간 풀링 작업이 이어짐.
- 출력은 연결되어 다른 컨볼루셔널 레이어를 통과하여 특징을 통합함으로써 다양한 공간 계층에서 상세한 특징을 추출할 수 있도록 네트워크의 용량을 최적화함.

```
class RepNCSPELAN4(nn.Module):
    # csp-elan
    def __init__(self, c1, c2, c3, c4, c5=1):  # ch_in, ch_out, numb
        super().__init__()
        self.c = c3//2
        self.cv1 = Conv(c1, c3, 1, 1)
        self.cv2 = nn.Sequential(RepNCSP(c3//2, c4, c5), Conv(c4, c4
        self.cv3 = nn.Sequential(RepNCSP(c4, c4, c5), Conv(c4, c4, 3
        self.cv4 = Conv(c3+(2*c4), c2, 1, 1)

    def forward(self, x):
        y = list(self.cv1(x).chunk(2, 1))
        y.extend((m(y[-1])) for m in [self.cv2, self.cv3])
        return self.cv4(torch.cat(y, 1))

    def forward_split(self, x):
        y = list(self.cv1(x).split((self.c, self.c), 1))
        y.extend(m(y[-1]) for m in [self.cv2, self.cv3])
        return self.cv4(torch.cat(y, 1))

class RepNCSP(nn.Module):
    # CSP Bottleneck with 3 convolutions
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # c
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c1, c_, 1, 1)
        self.cv3 = Conv(2 * c_, c2, 1)  # optional act=FReLU(c2)
        self.m = nn.Sequential(*(RepNBottleneck(c_, c_, shortcut, g
        , e=1.0) for _ in range(n)))

    def forward(self, x):
        return self.cv3(torch.cat((self.m(self.cv1(x)), self.cv2(x))
```

- RepNCSPELAN4
  - 특징 추출 프로세스를 더욱 간소화하는 것을 목표로 하는 CSP-ELAN의 고급 버전을 나타냄.

## Pruning

L2 norm을 활용하여 RepNCSPELAN4안의 conv 레이어의 중요도와 이 conv레이어 안의 필터의 중요도를 계산하는 함수를 추가.

```
class RepNCSPELAN4(nn.Module):
    # csp-elan
    def __init__(self, c1, c2, c3, c4, c5=1):  # ch_in, ch_out, numb
        super().__init__()
        self.c = c3//2
        self.cv1 = Conv(c1, c3, 1, 1)
        self.cv2 = nn.Sequential(RepNCSP(c3//2, c4, c5), Conv(c4, c4
        self.cv3 = nn.Sequential(RepNCSP(c4, c4, c5), Conv(c4, c4, 3
        self.cv4 = Conv(c3+(2*c4), c2, 1, 1)

    def forward(self, x):
        y = list(self.cv1(x).chunk(2, 1))
        y.extend((m(y[-1])) for m in [self.cv2, self.cv3])
        return self.cv4(torch.cat(y, 1))

    def forward_split(self, x):
        y = list(self.cv1(x).split((self.c, self.c), 1))
        y.extend(m(y[-1]) for m in [self.cv2, self.cv3])
        return self.cv4(torch.cat(y, 1))

    def get_filter_importance_l2(self):
        importances = []
        for module in [self.cv1, self.cv2, self.cv3, self.cv4]:
            if isinstance(module, nn.Sequential):
                for submodule in module:
                    if isinstance(submodule, Conv):
                        importance = torch.norm(submodule.conv.weigh
                        importances.append(importance)
            elif isinstance(module, Conv):
                importance = torch.norm(module.conv.weight, p=2, dim
                importances.append(importance)
        return torch.cat(importances)
```

```
    def compare_layer_importance(self):
        layer_importances = []
        for module_name, module in self.named_modules():
            if isinstance(module, Conv):
                importance = torch.norm(module.conv.weight, p=2).ite
                layer_importances.append((module_name, importance))

        layer_importances.sort(key=lambda x: x[1])

        return layer_importances
```

이를 통해 RepNCSPELAN4안의 conv 레이어의 중요도가 제일 낮은 순으로 top10개를 출력하여 어느 conv 레이어의 중요도가 낮은지 확인함.

```
import torch
from models.common import RepNCSPELAN4, Conv
from models.yolo import Model

def get_layer_importance_l2(module):
    if isinstance(module, Conv):
        return torch.norm(module.conv.weight, p=2).item()
    return None

def compare_layer_importance(model):
    layer_importances = []
    for module_name, module in model.named_modules():
        if isinstance(module, Conv):
            importance = get_layer_importance_l2(module)
            layer_importances.append((module_name, importance))

    layer_importances.sort(key=lambda x: x[1])

    return layer_importances

if __name__ == "__main__":
    weights = "/home/mgh/MGH/log/yolov9-c-converted.pt"
    device = torch.device("cuda" if torch.cuda.is_available() else "

    ckpt = torch.load(weights, map_location=device)
    model = Model(ckpt['model'].yaml).to(device)
```

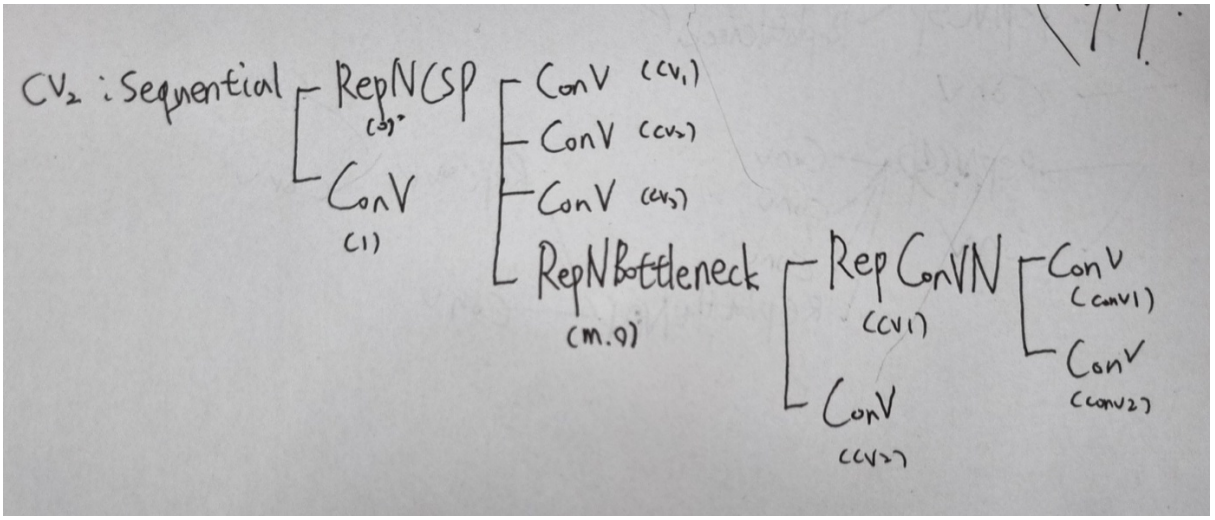
```
    model.load_state_dict(ckpt['model'].float().state_dict())
    model.eval()

    all_layer_importances = []
    for name, module in model.named_modules():
        if isinstance(module, RepNCSPELAN4):
            print(f"Comparing Conv layers in {name}:")
            layer_importances = compare_layer_importance(module)
            all_layer_importances.extend([(f"{name}.{lname}", import


    all_layer_importances.sort(key=lambda x: x[1])
    print(f"Top 10 Conv layers with lowest importance in the entire
    for i, (name, importance) in enumerate(all_layer_importances[:10
        print(f"{i + 1}. Layer: {name}, Importance: {importance}")
```

1. Layer: model.21.cv2.0.m.0.cv1.conv2, Importance: 0.3630586862564087
2. Layer: model.15.cv3.0.m.0.cv1.conv2, Importance: 0.3843393921852112
3. Layer: model.21.cv3.0.m.0.cv1.conv2, Importance: 0.40796664357185364
4. Layer: model.21.cv3.0.cv2, Importance: 0.4458220303058624
5. Layer: model.18.cv3.0.m.0.cv1.conv2, Importance: 0.4571390151977539
6. Layer: model.4.cv3.0.m.0.cv1.conv2, Importance: 0.4911297857761383
7. Layer: model.18.cv2.0.m.0.cv1.conv2, Importance: 0.5546313524246216
8. Layer: model.8.cv3.0.m.0.cv1.conv2, Importance: 0.557483971118927
9. Layer: model.2.cv3.0.m.0.cv1.conv2, Importance: 0.5630103349685669
10. Layer: model.21.cv2.0.cv2, Importance: 0.5832948088645935

실제로 각 RepNCSPELAN4의 conv 레이어 16개를 제대로 출력하는지도 확인함.

```
import torch
from models.common import RepNCSPELAN4, Conv
from models.yolo import Model

def get_layer_importance_l2(module):
    if isinstance(module, Conv):
        return torch.norm(module.conv.weight, p=2).item()
    return None
```

```
def compare_layer_importance(model):
    layer_importances = []
    for module_name, module in model.named_modules():
        if isinstance(module, Conv):
            importance = get_layer_importance_l2(module)
            layer_importances.append((module_name, importance))

    layer_importances.sort(key=lambda x: x[1])

    # 중요도가 더 낮은 Conv 레이어 찾기
    if len(layer_importances) >= 2:
        print(f"Conv layers comparison based on L2 norm importance:"
        for i, (name, importance) in enumerate(layer_importances):
            print(f"{i + 1}. Layer: RepNCSPELAN4의 {name}, Importance
    else:
        print("Not enough Conv layers found to compare.")

if __name__ == "__main__":
    weights = "/home/mgh/MGH/log/yolov9-c-converted.pt"  # 모델 가중치
    device = torch.device("cuda" if torch.cuda.is_available() else "

    ckpt = torch.load(weights, map_location=device)  # 체크포인트 로드
    model = Model(ckpt['model'].yaml).to(device)  # 모델 생성
    model.load_state_dict(ckpt['model'].float().state_dict())  # 모델
    model.eval()

    # RepNCSPELAN4 레이어의 Conv 레이어 중요도 비교 및 출력
    repncsplelan4_layers = []
    for name, module in model.named_modules():
        if isinstance(module, RepNCSPELAN4):
            repncsplelan4_layers.append((name, module))

    if repncsplelan4_layers:
        for repncsplelan4_name, repncsplelan4_layer in repncsplelan4_
            print(f"Comparing Conv layers in {repncsplelan4_name}:")
            compare_layer_importance(repncsplelan4_layer)
            print("\n")
    else:
        print("RepNCSPELAN4 layer not found in the model.")
```




또한 그 conv 레이어 안 속 필터 단위로 중요도를 계산한 뒤에 이에 대한 필터 중요도 역시 낮은 순위로 top10을 출력함

```
import torch
from models.common import RepNCSPELAN4, Conv
from models.yolo import Model

def get_filter_importance_l2(model):
    importances = []
    for module_name, module in model.named_modules():
        if isinstance(module, RepNCSPELAN4):
            for sub_module_name, sub_module in module.named_modules(
                if isinstance(sub_module, Conv):
                    importance = torch.norm(sub_module.conv.weight,
                    importances.extend([(f"{module_name}.{sub_module
    return importances

def print_filter_importance_rankings(importance_list, top_k=10):
    sorted_importance = sorted(importance_list, key=lambda x: x[2])
    print(f"Top {top_k} filters with lowest importance (L2 norm):")
    for rank, (module_name, filter_index, importance) in enumerate(s
        print(f"{rank + 1}. Layer: {module_name}, Filter: {filter_in

if __name__ == "__main__":
    weights = "/home/mgh/MGH/log/yolov9-c-converted.pt"
    device = torch.device("cuda" if torch.cuda.is_available() else "
```

```
    ckpt = torch.load(weights, map_location=device)
    model = Model(ckpt['model'].yaml).to(device)
    model.load_state_dict(ckpt['model'].float().state_dict())
    model.eval()


    filter_importance_list = get_filter_importance_l2(model)
    print_filter_importance_rankings(filter_importance_list, top_k=1
```

1. Layer: model.4.cv2.0.m.0.cv1.conv2, Filter: 38, Importance: 0.0000156829
2. Layer: model.4.cv2.0.m.0.cv1.conv2, Filter: 46, Importance: 0.0000641889
3. Layer: model.4.cv2.0.m.0.cv1.conv2, Filter: 18, Importance: 0.0001313469
4. Layer: model.4.cv2.0.m.0.cv1.conv2, Filter: 50, Importance: 0.0005582641
5. Layer: model.4.cv2.0.m.0.cv1.conv2, Filter: 44, Importance: 0.0023514898
6. Layer: model.15.cv2.0.m.0.cv1.conv2, Filter: 13, Importance: 0.0086911321
7. Layer: model.4.cv3.0.m.0.cv1.conv2, Filter: 54, Importance: 0.0087010860
8. Layer: model.15.cv3.0.m.0.cv1.conv2, Filter: 8, Importance: 0.0088394564
9. Layer: model.4.cv3.0.m.0.cv1.conv2, Filter: 57, Importance: 0.0092789866
10. Layer: model.15.cv3.0.m.0.cv1.conv2, Filter: 30, Importance: 0.0117334453

처음에 필터의 중요도를 계산하고, 필터 중요도를 정렬한 뒤에 pruning 비율에 따라 가지치기할 필터의 개수를 결정함.

필터를 제거한 개수의 output 채널 개수를 설정하고 이에 따른 새로운 conv레이어와 bn레이어를 설정.

그 후 필터를 제거한 후에 새로운 레이어에 가중치를 설정하고 레이어를 교체함.

```
import torch

# 가지치기된 모델 체크포인트 로드
pruned_weights = "/home/mgh/MGH/log/yolov9-c-pruning/yolov9-c-pruned_
ckpt_pruned = torch.load(pruned_weights, map_location='cpu')

# 원본 모델 체크포인트 로드
original_weights = "/home/mgh/MGH/log/yolov9-c-converted.pt"
ckpt_original = torch.load(original_weights, map_location='cpu')
```

```
# 가중치 비교
def compare_weights(ckpt_original, ckpt_pruned):
    original_state_dict = ckpt_original['model'].state_dict()
    pruned_state_dict = ckpt_pruned['model'].state_dict()

    for key in original_state_dict.keys():
        if key in pruned_state_dict:
            original_weight = original_state_dict[key]
            pruned_weight = pruned_state_dict[key]
            if not torch.equal(original_weight, pruned_weight):
                print(f"Layer {key} is different.")
                print(f"Original weight: {original_weight}")
                print(f"Pruned weight: {pruned_weight}")
                print("-" * 50)

compare_weights(ckpt_original, ckpt_pruned)
```

yolov9-c-converted.pt파일과 비교 확인했을 때 가중치가 변화된 부분이 있는 것을 확인하였고, 개수 차이도 있는 것을 확인.

```
import torch

# 가지치기된 모델 체크포인트 로드
pruned_weights = "/home/mgh/MGH/log/yolov9-c-pruning/yolov9-c-pruned
ckpt_pruned = torch.load(pruned_weights, map_location='cpu')

# 원본 모델 체크포인트 로드
original_weights = "/home/mgh/MGH/log/yolov9-c-converted.pt"
ckpt_original = torch.load(original_weights, map_location='cpu')

# Conv 레이어 필터 개수 비교
def compare_conv_filters(ckpt_original, ckpt_pruned):
    original_state_dict = ckpt_original['model'].state_dict()
    pruned_state_dict = ckpt_pruned['model'].state_dict()

    for key in original_state_dict.keys():
        if 'conv.weight' in key:
            original_weight = original_state_dict[key]
            pruned_weight = pruned_state_dict[key]
            print(f"Layer: {key}")
```

```
            print(f"Original filter count: {original_weight.shape[0]
            print(f"Pruned filter count: {pruned_weight.shape[0]}")
            print("-" * 50)

compare_conv_filters(ckpt_original, ckpt_pruned)
```

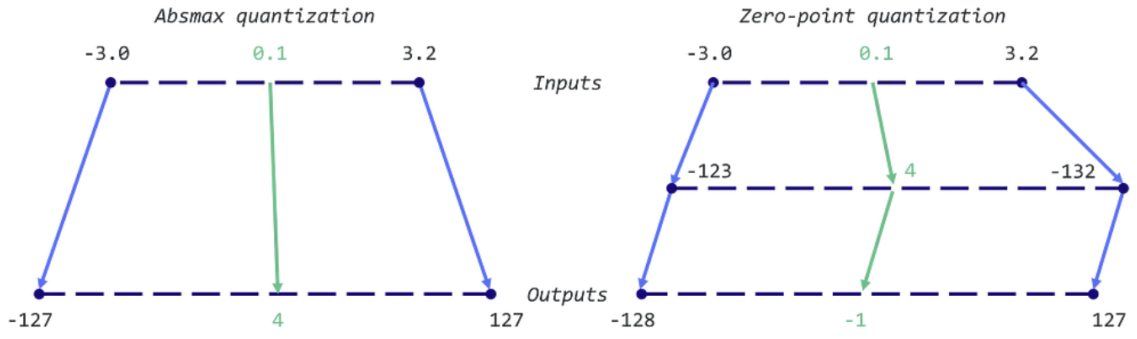
또한 필터의 개수도 원본과 비교해보며 줄어든 것을 확인

```
--------------------------------------------------
Layer: model.21.cv2.0.m.0.cv1.conv1.conv.weight
Original filter count: 128
Pruned filter count: 50
--------------------------------------------------
Layer: model.21.cv2.0.m.0.cv1.conv2.conv.weight
Original filter count: 128
Pruned filter count: 51
--------------------------------------------------
Layer: model.21.cv2.0.m.0.cv2.conv.weight
Original filter count: 128
Pruned filter count: 46
--------------------------------------------------
```

원본 모델의 가중치를 가지치기된 모델의 가중치로 업데이트하고, 이를 새로운 모델로 저장→ 원본 모델의 구조를 유지하면서 가지치기된 모델의 가중치를 적용하여 가지치기가 된 모델과 원본 모델의 가중치 크기 불일치 문제 및 레이어 구조 차이 문제를 방지하고자 함.

## static quantization

- static quantization은 weight와 activation을 모두 양자화하는 방법
- static quantization은 Dynamic quantization과 다르게 activation의 scale factor와 zero point를 계산할 필요가 없어 연산이 빠른 장점이 있음.
- 또한 Convolution, Activation, Batch Normalization모두 같은 비트 수를 사용하기 때문에 각 layer를 융합할 수 있음. 융합한 layer는 병렬 연산이 용이해지기 때문에 연산 효율이 증가
- Static quantization에서 고정된 scale factor와 zero point는 inference하는 데이터에 잘 맞지 않을 수 있음. 따라서 정확도 손실을 최소화하기 위한 calibration(보정)작업을 수행함.
- Calibration을 위해서는 Unlabeled Data가 필요함. 만약 Unlabeled Data를 사용하지 않는다면 scale factor와 zero point가 부정확해줄 것이며 (inference할 데이터에 맞게 보정되지 않으며) inference했을 때 feature값이 실제 차이가 발생하여 정확도 손실이 발생




이미지: Maxime Labonne

- yolov9-c-converted Static 양자화 코드(양 끝단은 양자화 하지 않음)

```
import torch
import torch.quantization
from models.experimental import attempt_load
from utils.dataloaders import create_dataloader
from utils.torch_utils import select_device
import os

# 모델 로드 (FP16 형식을 가정)
model_fp = attempt_load('/home/mgh/MGH/log/yolov9-c-converted.pt')
device = select_device('cpu')  # 양자화는 CPU에서 진행
model_fp = model_fp.float()  # 모델을 float32로 변환

# 양자화 설정 (특정 레이어를 제외한 나머지에 적용)
backend = "qnnpack"
torch.backends.quantized.engine = backend

# 양자화 설정을 제외할 레이어 이름 리스트 (첫단과 끝단)
exclude_layers = ['0', '22']

# 모든 레이어에 기본 qconfig 설정, 특정 레이어는 제외
for name, module in model_fp.named_children():
    if name in exclude_layers:  # 제외할 레이어 이름이 포함된 경우
        module.qconfig = None  # 해당 레이어의 양자화 설정 제거
    else:
        module.qconfig = torch.quantization.get_default_qconfig(back

# 양자화 준비
model_static_quantized = torch.quantization.prepare(model_fp, inplac

# 캘리브레이션을 위한 데이터 로드
```

```
img_size = 640
data = '/home/mgh/MGH/COCO_PAPER/coco/coco.yaml'
batch_size = 16
dataloader = create_dataloader('/home/mgh/MGH/COCO_PAPER/coco/images

with torch.no_grad():
    for i, (img, targets, paths, shapes) in enumerate(dataloader):
        img = img.to(device, non_blocking=True).float() / 255.0
        model_static_quantized(img)
        if i >= 1000:  # 제한된 배치 수로 캘리브레이션
            break

# 양자화 완료
model_static_quantized = torch.quantization.convert(model_static_qua

# 모델 저장
torch.save(model_static_quantized.state_dict(), "./convert_c_quantiz
```

- Qint8로 양자화 된 것을 확인할 수 있음. 추론 시간도 30% 증가하였음.
- 하지만 아래와 같이 raw Predictions를 할 때 같은 값이 반복 되는 것을 볼 수 있음. 즉 Calibration이 잘 되지 않았고, activation에서 scale이랑 zeropoint를 정할 때 문제가 발생한 것으로 보임.

raw predictions: [tensor([[[4.00000e+00, 1.20000e+01, 2.00000e+01,  ..., 5.60000e+02, 5.92000e+02]
      [4.00000e+00, 4.00000e+00, 4.00000e+00,  ..., 4.64000e+02, 4.64000e+02, 4.64000e+02],
      [1.20000e+02, 1.20000e+02, 1.20000e+02,  ..., 4.80000e+02, 4.80000e+02, 4.80000e+02],
      ...,
      [9.76553e-06, 9.76553e-06, 9.76553e-06,  ..., 1.56226e-04, 1.56226e-04, 1.56226e-04],
      [9.76553e-06, 9.76553e-06, 9.76553e-06,  ..., 1.56226e-04, 1.56226e-04, 1.56226e-04],
      [9.76553e-06, 9.76553e-06, 9.76553e-06,  ..., 1.56226e-04, 1.56226e-04, 1.56226e-04]]])

- NVM을 통과했을 때  tesnor에 아무 값도 안 들어 있음. 즉 detection이 되지 않음.

**detection이 안 됐을 때** : Predictions after NMS: [tensor([], size=(0, 6))]

**detection이 잘 됐을 때** : Predictions after NMS: [tensor([[2.23530e+02, 6.30082e+01, 3.90867e+02, 5.52261e+02, 9.09943e-01, 0.00000e+00],

[4.67566e+01, 8.98763e+01, 1.78675e+02, 5.66706e+02, 8.98236e-01, 0.00000e+00],
[1.27727e+02, 1.68548e+02, 1.55837e+02, 2.36333e+02, 8.57770e-01, 2.70000e+01],
[2.86390e+02, 1.45235e+02, 3.07472e+02, 2.26679e+02, 7.74843e-01, 2.70000e+01],
[3.68365e+02, 2.61463e+02, 4.08159e+02, 3.42694e+02, 3.47795e-01, 5.60000e+01]])]

## Quantization Aware Training

- 앞에서 정리된 방법은 모두 Post Training Quantization 즉 학습이 완료된 모델 양자화 정리
- Post-Training Quantization은 32bit를 사용하여 저장한 floating point형 숫자를 더 낮은 비트를 사용하여 표현하는 방법.
- 숫자를 저장할때 정확도 손실이 발생하여 이 모델로부터 다시 원래 숫자를 복원할 때 양자화 하기 전의 값과 차이가 발생, 이 차이로부터 모델 전체의 성능 저하가 발생
- Quantization Aware Training은 학습 할 때 inference시 양자화로 인한 영향을 미리 모델링 하는 방법
- Post Training Quantization은 큰 모델을 양자화 할 때 비해 양자화 모델의 성능 하락을 최소화 할 수 있는 장점이 있음.

## QAT 수행

**디코딩 및 연산 오버헤드 발생**

- 양자화는 모델의 가중치와 활성화를 더 적은 비트로 표현하여 메모리 사용량을 줄이고 연산을 가속화함.  8비트 양자화의 경우, 16비트 또는 32비트의 가중치와 활성화를 8비트 정수로 변환함.
- 8비트 양자화된 값을 다시 16비트 또는 32비트로 디코딩하여 연산을 수행하는 경우, 디코딩 과정에서 오버헤드가 발생함. 이 오버헤드는 특히 빈번한 연산이 필요한 딥러닝 모델에서 성능 저하 발생. 따라서 Fake int8개념 도입됨
- 가중치와 활성화를 메모리에 저장할 때만 8비트로 저장하고, 실제 연산을 수행할 때는 16비트 또는 32비트로 변환함. 이렇게 하면 메모리 사용량을 줄일 수 있지만, 연산 시에는 여전히 높은 정밀도를 유지할 수 있음.
- fake int8"은 8비트 양자화의 디코딩 오버헤드와 정밀도 손실 문제를 완화함. 이를 통해 양자화된 모델이 메모리 효율성을 유지하면서도 실제 연산에서는 높은 성능을 유지. 하지만 추론 속도 증가한 문제가 발생. 디코딩 연산을 추가로 하기 때문에 이런 문제가 발생했다고 생각됨. onnx변환할 때는 fake을 풀고 수행할 예정

## 비고

- RepNCSPELAN4 중 중요도가 낮다고 판단되는 블록을 shufflenet_unit으로 교체(우선 전체 교체)한 test12의 학습을 진행 중에 있음.

**ShuffleNetV2**

```
class ShuffleNetV2(nn.Module):
    def __init__(self, inp, oup, stride):
        super(ShuffleNetV2, self).__init__()

        if not (1 <= stride <= 3):
            raise ValueError('illegal stride value')
        self.stride = stride

        inp_c = inp // 2
        oup_c = oup // 2

        if self.stride > 1:
            self.branch1 = nn.Sequential(
                self.depthwise_conv(inp, inp, kernel_size=3, stride=
                nn.BatchNorm2d(inp_c),
                nn.Conv2d(inp, oup_c, kernel_size=1, stride=1, paddi
                nn.BatchNorm2d(oup_c),
                nn.ReLU(inplace=True),
            )
        else:
            self.branch1 = nn.Sequential(
                nn.Conv2d(inp_c, oup_c, kernel_size=1, stride=1, bia
                nn.BatchNorm2d(oup_c)
            )

        self.branch2 = nn.Sequential(
            nn.Conv2d(inp_c, oup_c, kernel_size=1, stride=1, padding
            nn.BatchNorm2d(oup_c),
            nn.ReLU(inplace=True),
            self.depthwise_conv(oup_c, oup_c, kernel_size=3, stride=
            nn.BatchNorm2d(oup_c),
            nn.Conv2d(oup_c, oup_c, kernel_size=1, stride=1, padding
            nn.BatchNorm2d(oup_c),
            nn.ReLU(inplace=True),
        )
```

```
    @staticmethod
    def depthwise_conv(i, o, kernel_size, stride=1, padding=0, bias=
        return nn.Conv2d(i, o, kernel_size, stride, padding, bias=bi

    def channel_shuffle(self, x, groups):
        # type: (torch.Tensor, int) -> torch.Tensor
        batchsize, num_channels, height, width = x.data.size()
        channels_per_group = num_channels // groups

        # reshape
        x = x.view(batchsize, groups,
                   channels_per_group, height, width)

        x = torch.transpose(x, 1, 2).contiguous()

        # flatten
        x = x.view(batchsize, -1, height, width)

        return x

    def forward(self, x):
        if self.stride == 1:
            x1, x2 = x.chunk(2, dim=1)
            out = torch.cat((self.branch1(x1), self.branch2(x2)), di
        else:
            out = torch.cat((self.branch1(x), self.branch2(x)), dim=

        out = self.channel_shuffle(out, 2)

        return out
```

## 추후 계획

- test12의 성능을 확인, 비교 및 결과 분석.
- 최종 모델 및 시연 준비