주간 보고서-6주차 (2024.04.08 ~ 2024.04.14)

팀명 : 무계획

회의 참석자 : 서태원, 신재환, 최승렬, 류은환

회의 일자 및 회의 장소 :

2024.04.08(월)

- 15:00 ~ 17:00, 회의 장소 : 7호관 302호
- 18:00 ~ 22:30, 회의 장소 : 7호관 302호

2024.04.09(화)

- 18:30 ~ 22:00, 회의 장소 : 7호관 302호

2024.04.10(수)

- 14:00 ~ 17:30, 회의 장소 : 7호관 302호
- 18:30 ~ 22:30, 회의 장소 : 7호관 302호

2024.04.11(목)

- 14:00 ~ 17:30, 회의 장소 : 7호관 302호
- 18:30 ~ 22:00, 회의 장소 : 7호관 302호

총 활동 시간 : 22시간 30분

진행 사항 :

팀원 간 업무사항 할당

👨‍💼서태원
현재 일:

[업무]

논문에서 사용한 parameter와 현재 실험에 사용한 parameter 비교 (가능하면)

일부 아키텍처에 대해 수정 방향 탐색 후 실험
train 코드 분석

👨‍💻류은환
현재 일:

[업무]
COCO 데이터셋을 YOLO 학습에 사용 하기 위한 전처리 과정 진행 중 (JSON → TXT)
YOLOv9 논문과 현재 학습한 모델과 비교 학습 모델 PC 성능 확인

파라미터 yaml 파일 분석 및 정리 (학습 파라미터)

train 코드 분석

👨‍💻신재환
현재 일:

[업무]
라즈베리 파이에 모델 올리기
라즈베리파이에서
성능 지표 리포팅 코드 제작 (평가 Tool 제작)
val, detect 코드 분석

👨‍💻최승렬
현재 일:

[업무]

라즈베리 파이에 모델 올리기
라즈베리파이에서
성능 지표 리포팅 코드 제작 (평가 Tool 제작)
val, detect 코드 분석

COCO 데이터셋을 YOLO 학습에 사용 하기 위하여 Roboflow를 사용하여 json파일을 txt파일로 전환함.

yolov9 코드 이해

AttributeError: 'list' object has no attribute 'view’ 에러가 발생

→ 학습을 시키기 위해선 train.py 대신 train_dual.py을 사용해야 함. yolov9을 제대로 활용하기 위해서 코드를 보다 깊게 이해할 필요성을 느낌

1. **Hyper Parameter와 train_dual.py, val_dual.py, detect_dual.py 코드를 분석 및 이해함**

```
lr0: 0.01  # 초기 학습률 (SGD=1E-2, Adam=1E-3)
lrf: 0.01  # 사이클이 한번 돌때마다 학습률에 곱해주는 값 (lr0 * lrf)
# 초기에 큰 학습률로 빠르게 학습하고, 점점 작은 학습률로 최적값에 가까이 수

momentum: 0.937  # SGD momentum/Adam beta1
# SGD(Stochastic Gradient Descent) 확률적 경사 하강법
# 경사 하강법과 다르게 한번 학습할 때 모든 데이터에 대해 가중치를 조절하는
# 랜덤하게 추출한 일부 데이터에 대해 가중치를 조절
#
# Momentum 관성
# 경사 하강법과 마찬가지로 매번 기울기를 구하지만, 가중치를 수정하기전 이전
# 같은 방향으로 일정한 비율만 수정되게 하는 방법
#
# Adam
# RMSprop와 Momentum을 동시에 적용한 것과 같은 알고리즘
#
# beta1
# Adam 알고리즘에서 사용되는 모멘텀 계수 (beta1, beta2) 기본값은 (0.9,

weight_decay: 0.0005  # optimizer weight decay 5e-4
# Overfitting을 방지하기 위해 모델의 weight의 제곱합을 패널티 텀으로 주

warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr (warmup 초기 학습
# 학습 초반에 모든 가중치들이 무작위로 초기화 되어 있는 상태이고 최종 결과값
# 따라서 처음부터 높은 learning rate 를 gradient에 곱해서 weight를 늘
# 불안정한 상태에 빠질 수 있다.
# 이때 warmup을 하면 weight를 조금씩 키워서 안정화된 상태로 만들 수 있다
#
# 근데 위에서는 초기 학습률보다 높은 값을 사용하는지 추가 조사 필요

box: 7.5  # box loss gain
# 손실 함수에서 박스 손실 성분의 가중치로,
# 바운딩 박스 좌표를 정확하게 예측하는 데 얼마나 중점을 두는지에 영향을 줌
```

```
# 이 값이 높을 수록 모델은 객체의 위치와 크기를 정확하게 예측하는 것에 중점
#
# GT의 중앙점이 위치하는 좌표를 표시하여 object loss를 구하고,
# GT의 너비와 높이를 grid에 맞게 정규화한 것으로 box loss를 구함
# GT: Ground Truth 라벨링 값

cls: 0.5  # cls loss gain
# 총 손실 함수에서 분류 손실의 가중치로,
# 다른 구성 요소에 비해 정확한 클래스 예측의 중요도에 영향
# 이 값이 높을 수록 객체의 클래스를 정확하게 분류하는 것에 중점을 둠
cls_pw: 1.0  # cls BCELoss positive_weight
# 양성 샘플에 더 큰 가중치를 둔 클래스 손실함수

obj: 0.7  # obj loss gain (scale with pixels)
# 실제 객체와의 손실함수
obj_pw: 1.0  # obj BCELoss positive_weight
# 양성 샘플에 더 큰 가중치를 둔 객체 손실함수

dfl: 1.5  # dfl loss gain
# Distribution Focal Loss
# 세분화된 분류를 위해 특정 YOLO 버전에서 사용되는 분포 초점 솔실의 가중치
#
# Focal Loss와 Cross Entropy Loss를 결합한 형태로,
# 클래스 간 불균형 문제를 해결하고 분류 성능을 향상시키는 데 도움을 준다
#
# Focal Loss와 Cross Entropy Loss의 가중치를 조정하는 방식으로 동작
# Focal Loss는 높은 확률을 가진 예측값에 대해 더 큰 가중치를 부여
# Cross Entropy Loss는 모든 클래스에 대해 동일한 가중치를 부여
# DFL은 이 두 가지 손실 함수를 결합하여 높은 확률을 가진 클래스에 대해서만
# Focal Loss의 가중치를 적용하는 방식으로 동작

iou_t: 0.20  # IoU training threshold
# IoU 임계값
# 값이 높을 수록 높은 IoU값을 가진 바운딩 박스만 Positive로 인식

anchor_t: 5.0  # anchor-multiple threshold
# anchor박스의 multiple 임계값
# 값이 높을 수록 크기가 더 큰 앵커와도 매칭이 됨
```

```
# anchors: 3  # anchors per output layer (0 to ignore)

# Augmentation
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=
# 클래스 불균형 문제를 해결하기 위해 어렵거나 쉽게 오분류되는 케이스에 더 큰
# 쉬운 케이스에는 낮은 가중치를 부여

hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
# 색조, 채도, 명도

degrees: 0.0  # image rotation (+/- deg)
# 이미지를 회전
translate: 0.1  # image translation (+/- fraction)
# 이미지를 이동하여 학습
scale: 0.9  # image scale (+/- gain)
# 이미지의 크기를 변환
shear: 0.0  # image shear (+/- deg)
# 전단 변환 / 이미지를 밀어서 한쪽을 찌그러트림
perspective: 0.0  # image perspective (+/- fraction), range 0
# 원근 변환

flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
# 상하, 좌우 변환

mosaic: 1.0  # image mosaic (probability)
# 4장의 이미지를 한 장으로 만들어줌
mixup: 0.15  # image mixup (probability)
# 두 개의 이미지를 사용하여 서로 다른 계수 비율로 곱하고 중첩한 다음 이러힌
copy_paste: 0.3  # segment copy-paste (probability)
# 이미지 segment를 다른 사진에 붙여서 학습
```

val_dual.py

1. `save_one_txt()` 함수:
    - 탐지 결과를 텍스트 파일로 저장하는 함수
    - 탐지된 객체의 좌표, 신뢰도, 클래스 정보를 텍스트 형식으로 파일에 저장
2. `save_one_json()` 함수:
    - 탐지 결과를 COCO 데이터셋 형식의 JSON 파일로 저장하는 함수
    - 탐지된 객체의 이미지 ID, 카테고리 ID, 바운딩 박스 좌표, 신뢰도 정보를 JSON 형식으로 저장
3. `process_batch()` 함수:
    - 한 배치의 탐지 결과와 실제 레이블을 비교하여 올바른 탐지 여부를 판단하는 함수
    - 탐지된 객체와 실제 레이블 간의 IoU (Intersection over Union)를 계산하여 일치 여부를 판단
4. `run()` 함수:
    - 검증 프로세스의 메인 함수
    - 데이터 로더 설정, 모델 로딩, 검증 수행, 평가 지표 계산 등을 수행
    - 배치 단위로 이미지를 읽어와 모델에 입력하고, 탐지 결과를 후처리하여 평가 지표를 계산
5. `parse_opt()` 함수:
    - 커맨드라인 인자를 파싱하여 검증에 필요한 옵션을 설정하는 함수
    - 데이터셋 경로, 모델 경로, 배치 크기, 이미지 크기, 신뢰도 임계값 등을 설정할 수 있음
6. `main()` 함수:
    - 스크립트의 진입점 역할을 하는 함수
    - `parse_opt()` 함수를 호출하여 커맨드라인 인자를 파싱하고, 검증 작업을 수행
    - 검증 작업에는 일반적인 검증, 속도 측정, 속도와 mAP 간의 관계 분석 등이 포함

검증 과정에서는 데이터 로더를 통해 이미지를 배치 단위로 읽어오고, 모델에 입력하여 탐지 결과를 얻음. 탐지 결과는 후처리 과정을 거쳐 최종 탐지 결과로 변환되며, 실제 레이블과 비교하여 평가 지표를 계산

또한, 이 스크립트는 탐지 결과를 텍스트 파일이나 JSON 파일로 저장할 수 있으며, COCO 데이터셋 형식으로 저장된 JSON 파일을 사용하여 COCO API를 통해 평가 지표를 계산할 수도 있음

detect_dual.y

- speed 평균 구하는 코드

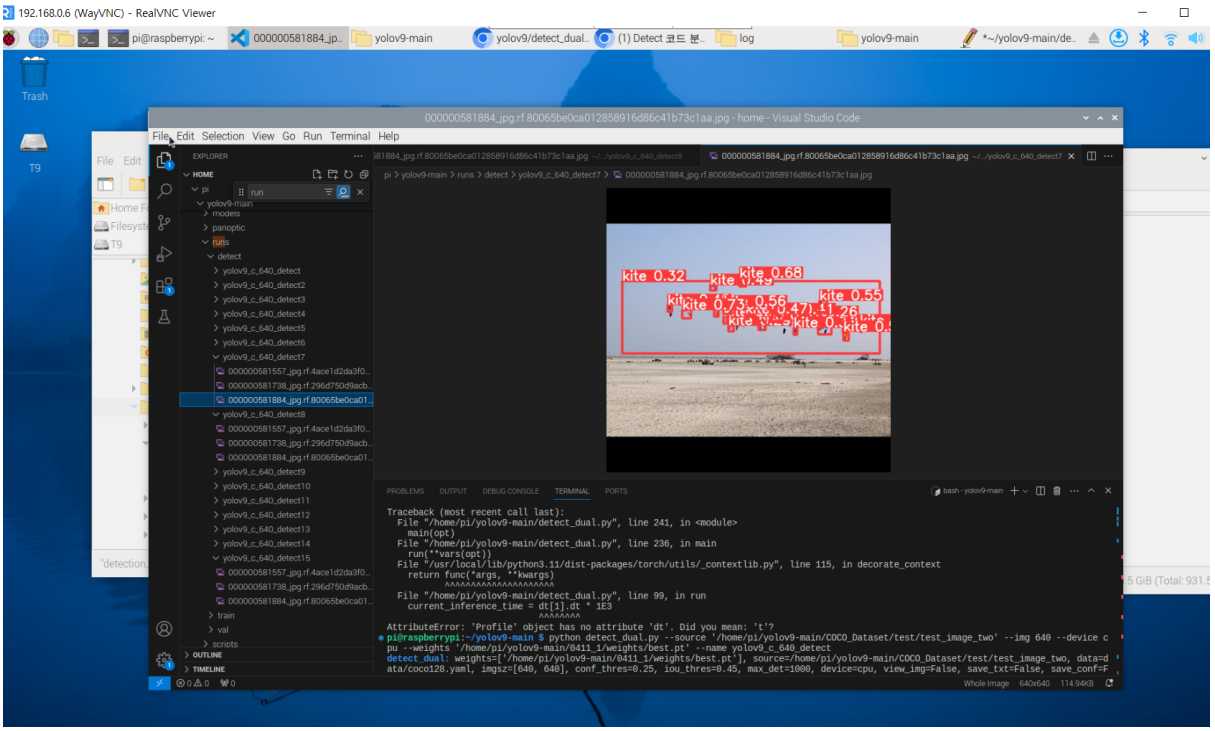
```
#기존 코드
# Print results
t = tuple(x.t / seen * 1E3 for x in dt)  # speeds per image
LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.
if save_txt or save_img:
    s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels
    LOGGER.info(f"Results saved to {colorstr('bold', save_dir
if update:
    strip_optimizer(weights[0])  # update model (to fix Sourc
```

- **LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per image at shape {(1, 3, *imgsz)}' % t)**
- 모든 사진을 추론했으면 아래와 같이 평균 연산속도가 나옴.

```
#test image를 3개만 골라서 평균 inference를 구해봄
image 1/3 /home/pi/yolov9-main/COCO_Dataset/test/test_image_t
image 2/3 /home/pi/yolov9-main/COCO_Dataset/test/test_image_t
image 3/3 /home/pi/yolov9-main/COCO_Dataset/test/test_image_t
Speed: 4.3ms pre-process, 8559.3ms inference, 2.1ms NMS per i
Results saved to runs/detect/yolov9_c_640_detect7
```

YOLOv9 논문과 현재 학습한 모델과 비교하기 위해 직접 돌려보며 확인해 본 결과 mAP50이 0.2정도 낮게 나온것을 확인함. epoch 수에서 차이가 있어서 25였던 epoch을 100으로 늘린 뒤에 다시 실험해보았고, 논문에서 측정된 mAP50와 비슷한 값을 확인할 수 있었음.

라즈베리파이에서의 처리속도 및 성능 비교



COCO Dataset by Microsoft

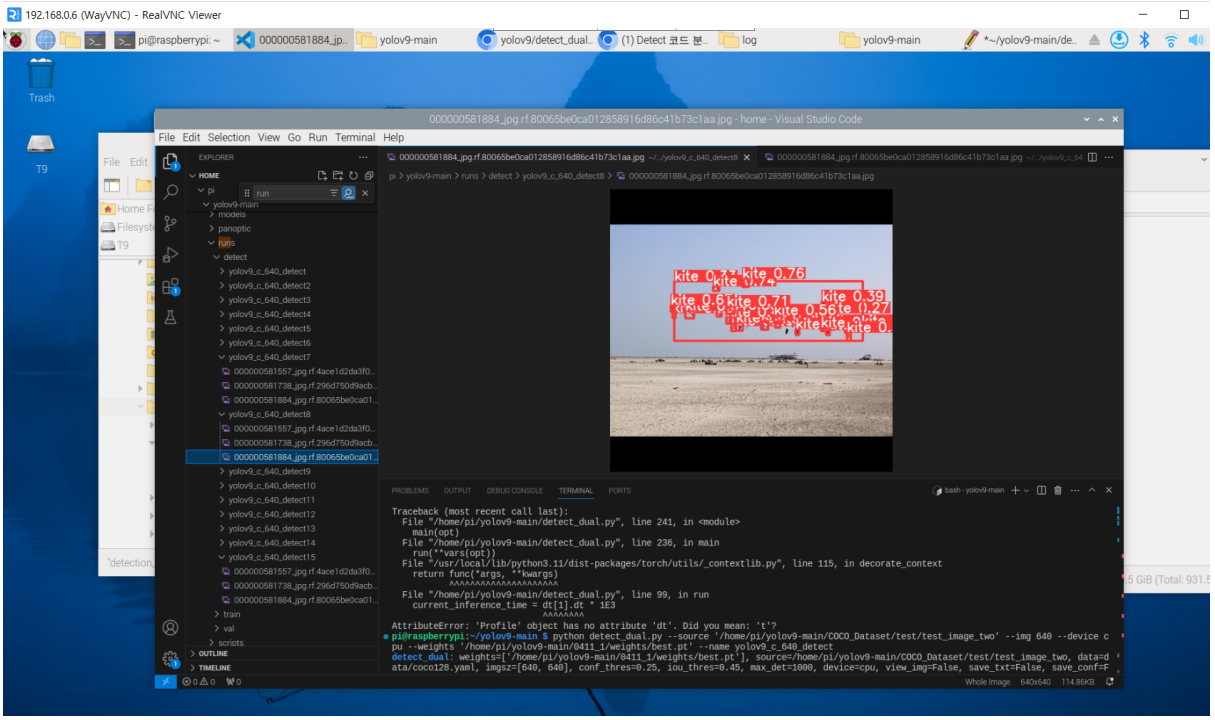
image 개수 : 123272

epoch : 25

mAP50 : 0.521

batch : 16

img : 640



COCO Dataset by Microsoft

image 개수 : 123272

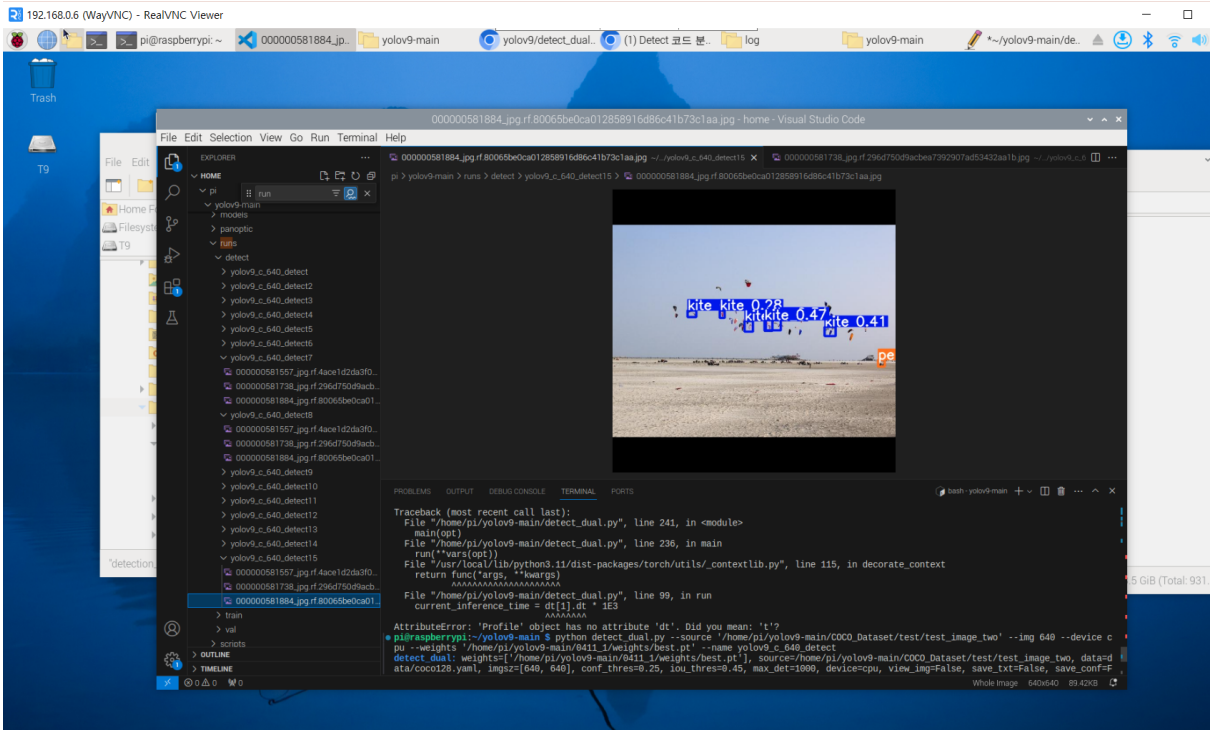
epoch : 100

mAP50 : 0.634

batch : 16

img : 640

또한 Dataset에서 이미지 개수가 적은 COCO 128 dataset으로 변경하여 학습시킨뒤 성능을 확인함



COCO 128 Dataset by team Roboflow

image 개수 : 372

epoch : 1000

mAP50 : 0.924

batch : 16

img : 640

비고

--resume → 학습이 도중에 중단되어도 이전에 중단된 체크포인트에서부터 이어서 학습을 진행하는 것을 확인함

추후 계획

- 라즈베리파이에서 성능 지표 리포팅 코드(평가 Tool) 제작 및 피드백
- backbone layer 수정 방향 탐색, 수정 후 학습 및 결과 확인
- Dataset 경량화 실험 중 class 개수를 줄여 결과 확인