지능화 캡스톤 프로젝트

프로젝트 #2 결과 발표

YOLOv8을 이용한 해상 객체 검출

2024. 6. 10.

충북대학교 산업인공지능학과

[7조] 김봉균, 김혜영, 이준혁

INDUSTRIAL AI RESEARCH CENTER

프로젝트 수행체계

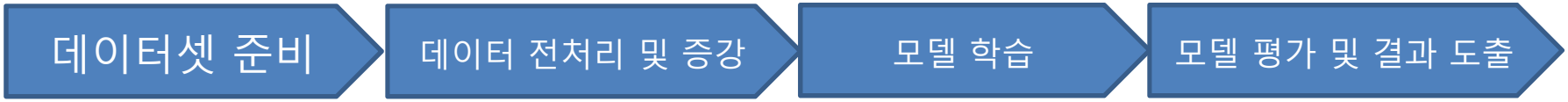
수행방법

-7조는 3인으로 구성
-자료조사/학습/발표/증량 등으로 업무를 분할하여 수행함
-개개인의 지역이 멀어 현재 줌 회의 또는 1주일 2회씩 카카오톡 등으로 수행
-대면 회의 수행
-학과노트북을 대여하여 파이참, 주피터 노트북 사용

업무분장

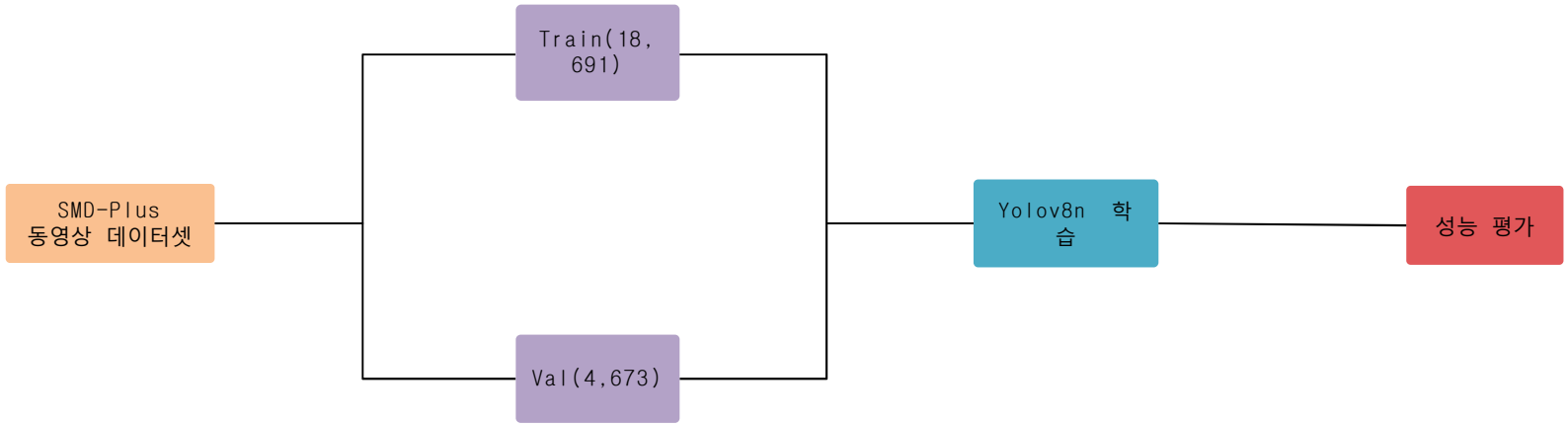
이름	수행내용	비고
김봉균	• 자료조사 • 비교모델 학습 • 모델 학습(N)	
김혜영	• 데이터 추출 • 모델 학습(L) • 프로젝트 총괄	
이준혁	• 모델 평가 및 분석 • 발표자료 준비 등 • 모델 학습(S)	

모델 개발 프로세스



1 Step	2 Step	3 Step	4 Step
데이터셋 준비	데이터 처리	모델 학습	모델 평가

- 데이터셋 다운로드
- 데이터(이미지) 추출
- .mat 주석 생성
- 하이퍼파라미터 설정 및 학습
- 모델 평가 및 시각화

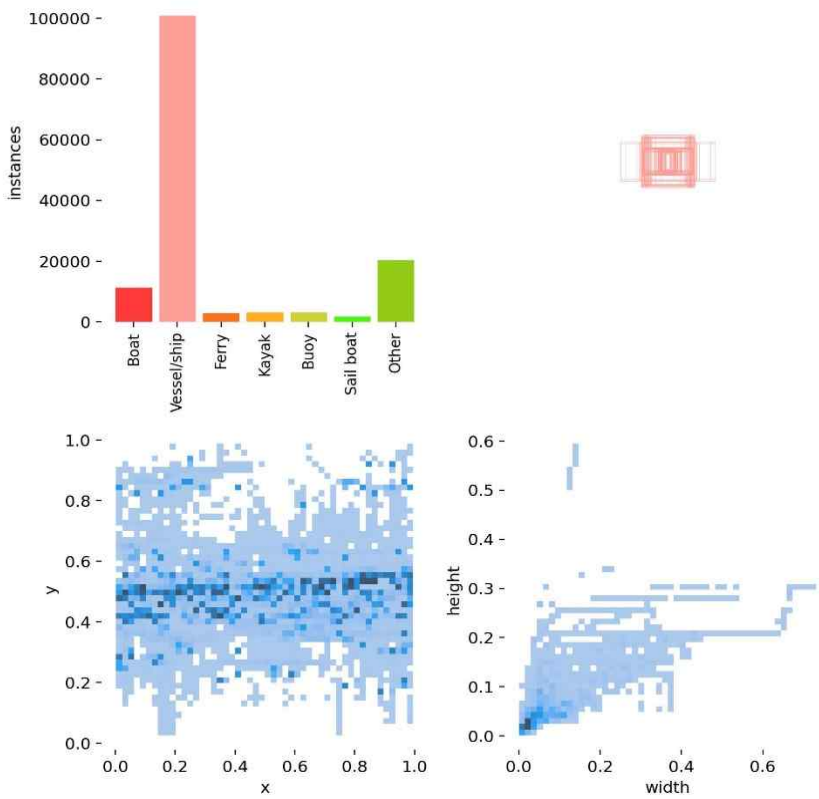


데이터셋

데이터셋의 구성

- ***SMD-Plus 공시 데이터셋*** *다운로드 : 총 51개의 동영상 (Onshore 40, Onboard 11)*
- 동영상 데이터셋에서 이미지 & 레이블 추출
- Boat, Vessel/ship, Ferry, Kayak, Buoy, Sail boat, Other 총 7개의 클래스
- **Vessel/ship** 클래스의 인스턴스 수가 가장 많고, 가장 빈번

SMD-Plus	
Class	Objects(#)
Boat	14,021
Vessel/Shi	125,872
Ferry	3431
Kayak	3798
Buoy	3657
Sail Boat	1926
Others	24,993
Removed	-
Removed	-



데이터셋

data augmentation (hyp.yaml) 및 train.py 파일 작성

```yaml
# Hyperparameters
lr0: 0.001
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 0.05
cls: 0.5
hsv_h: 0.015 # 색상 변화
hsv_s: 0.7 # 색상 변화
hsv_v: 0.4 # 색상 변화
degrees: 10.0 # 회전 (추가)
translate: 0.1 # 평행 이동
scale: 0.5 # 스케일링
shear: 0.0 # 전단 변형
perspective: 0.0 # 원근법 변환
flipud: 0.0 # 수직 반전
fliplr: 0.5 # 수평 반전
mosaic: 1.0 # 모자이크 데이터 증강
mixup: 0.1  # 낮은 확률로 mixup (추가)
copy_paste: 0.5  # Copy-paste 증강 (추가)
```

```python
# train.py
import yaml
from ultralytics import YOLO

def main():
    # Load the hyperparameters from the YAML file
    with open("hyp.yaml", "r") as file:
        hyp_params = yaml.safe_load(file)  # YAML 파일을 읽어와 딕셔너리로 변환

    # Load the model
    model = YOLO("yolov8s.pt")  # 사전 훈련된 모델 불러오기

    # Train the model with hyperparameters
    results = model.train(
        data="data.yaml",  # 데이터 설정 파일
        epochs=15,         # 훈련 에폭 수
        imgsz=640,         # 이미지 크기
        **hyp_params,      # 딕셔너리의 하이퍼파라미터를 풀어서 전달
        device=0           # 사용할 장치 지정 (예: 첫 번째 GPU)
    )

if __name__ == '__main__':
    main()
```

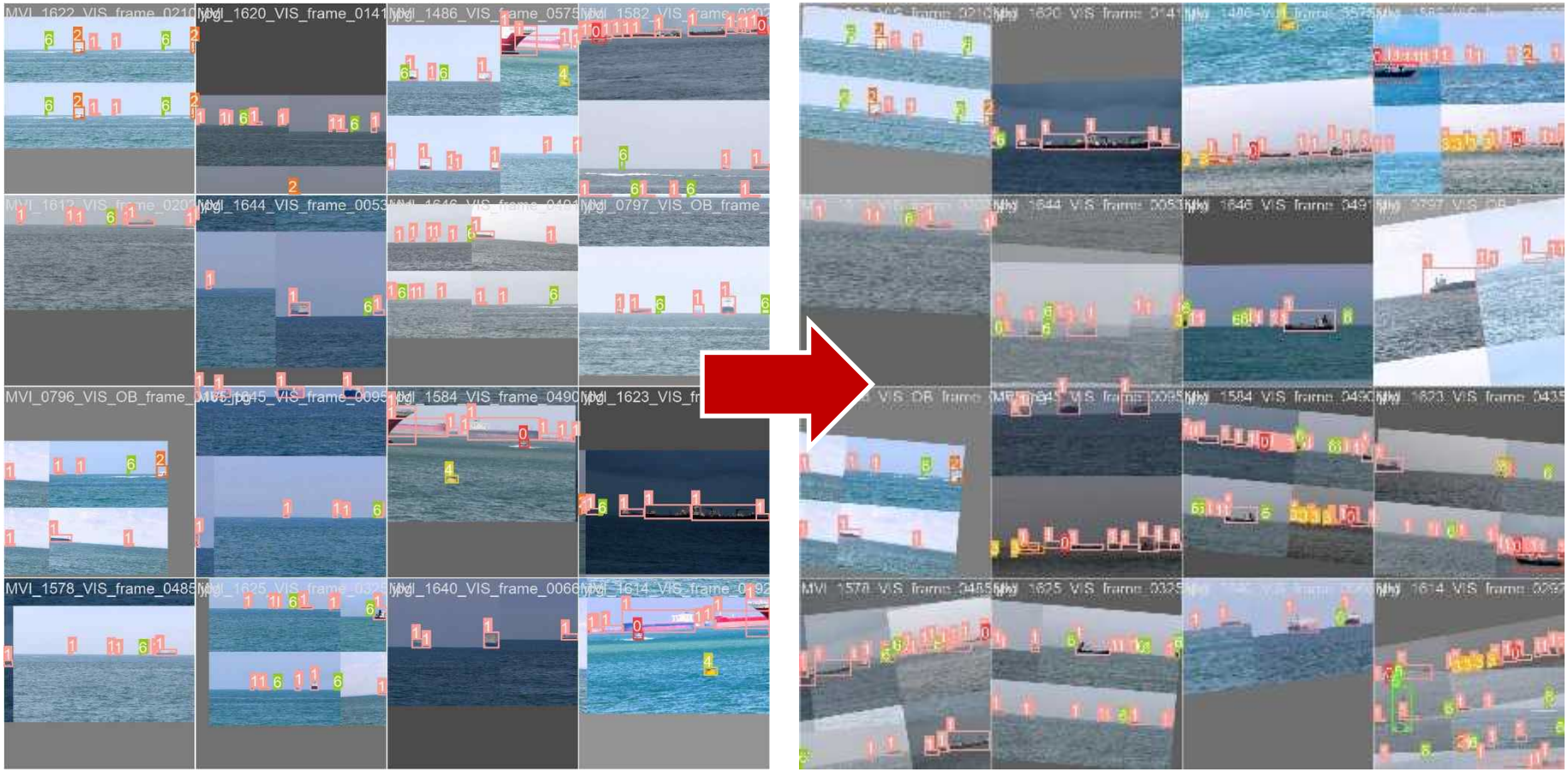
데이터셋

data augmentation 하이퍼파라미터 비교

	DEFAULT	CUSTOM
모델	YOLOv8n	YOLOv8n
에폭	**30**	**15**
배치	16	
이미지사이즈	640	
장치	0 (cuda)	
옵티마이저	auto → **AdamW**	
학습률	0.000909	
가중치 감소 (decay=0.0)	57	
가중치 감소 (decay=0.0005)	64	
편향 (decay=0.0)	63	
회전	**0**	**±10도 무작위 회전**
mosaic	**1**	**1**
mixup	**0**	**0.1 (10% 비율)**
copy_paste	**0**	**0.5 (50% 비율)**

데이터셋

Data augmentation 실행 결과



모델 학습

딥러닝 학습 환경

- *(HW) PC 사양, 학습시간*

  CPU : Intel(R) Core(TM) i7-1165G7 @ 2.80GHz

  RAM : 16GB

  GPU : NVIDIA GeForce GTX 1650 Ti

- *(SW)* ***Pytorch, CUDA***
- *하이퍼파라미터 :* ***640 imgsz, 15~30epoch, 0.0000909lr, batch size 16,***

  AdamW Optimizer

- *모델 :* ***YOLOv8n***

```
train: Scanning C:\Users\User\Desktop\capstone\finalProject\dataset\labels\train.cache... 18691 images, 568 backgrounds, 0 corrupt: 100%|██████████| 18691/18691 [00:00<?, ?it/s]
val: Scanning C:\Users\User\Desktop\capstone\finalProject\dataset\labels\val.cache... 4673 images, 133 backgrounds, 0 corrupt: 100%|██████████| 4673/4673 [00:00<?, ?it/s]
Plotting labels to runs\detect\train\labels.jpg...
  0%|          | 0/1169 [00:00<?, ?it/s]optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.000909, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)
Image sizes 640 train, 640 val
Using 8 dataloader workers
Logging results to runs\detect\train
Starting training for 15 epochs...
```

제안하는 모델

학습 출력 결과 (default)

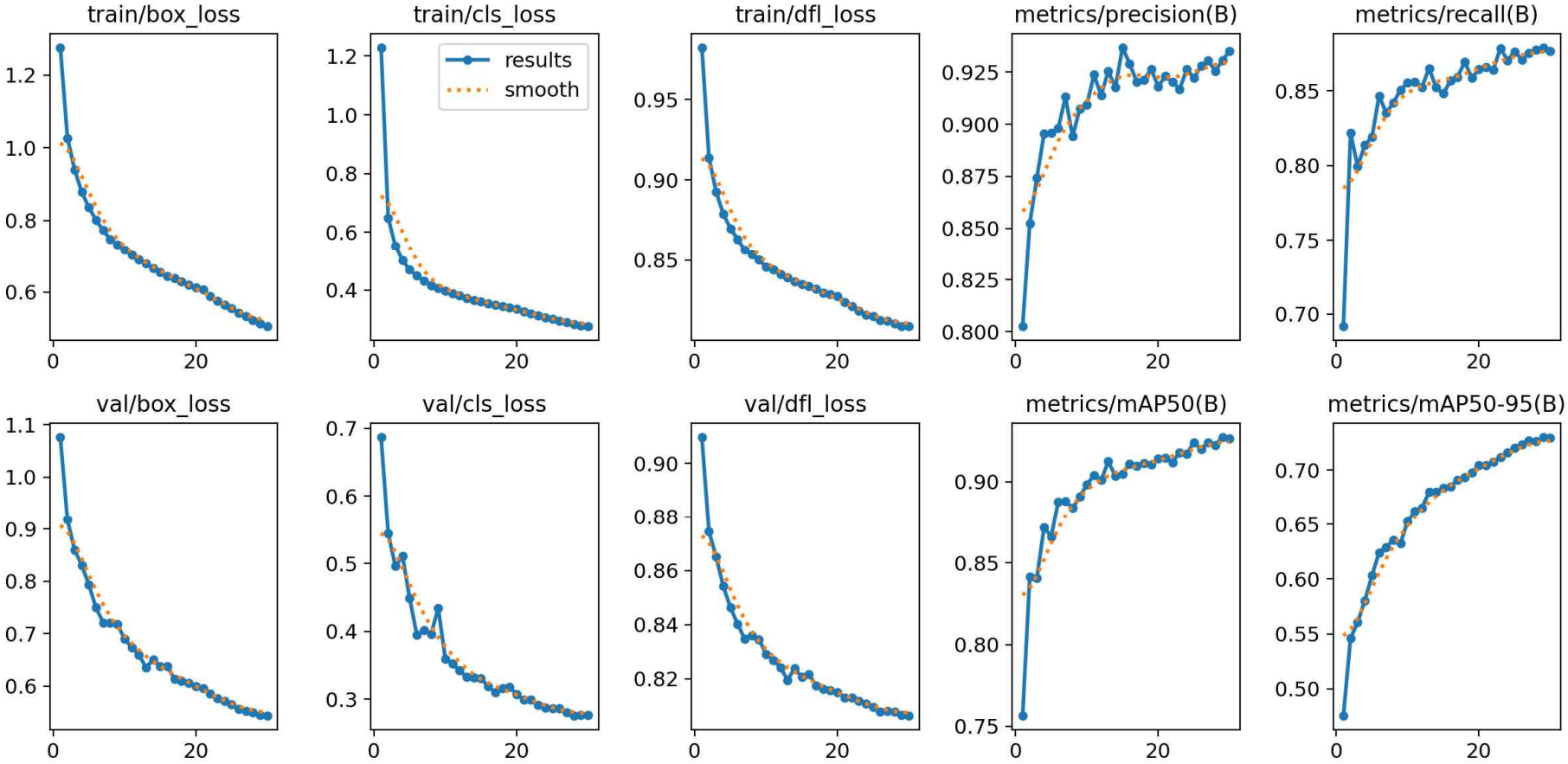
- 재현율, 정밀도, mAP
- 학습 시간 : 1epoch 당 약 20분

epoch	train/box_	train/cls_lc	train/dfl_lc	metrics/pr	metrics/re	metrics/m	metrics/m	val/box_lo	val/cls_los	val/dfl_los	lr/pg0	lr/pg1	lr/pg2
1	1.2756	1.2295	0.98217	0.80253	0.69178	0.75666	0.47561	1.0763	0.68685	0.90963	0.000303	0.000303	0.000303
2	1.0267	0.64812	0.91379	0.8525	0.82175	0.84157	0.5461	0.91784	0.54384	0.87444	0.000586	0.000586	0.000586
3	0.93931	0.5527	0.8928	0.87439	0.79946	0.84089	0.56044	0.85963	0.49569	0.86512	0.000849	0.000849	0.000849
4	0.87811	0.50197	0.87906	0.8954	0.81381	0.87166	0.58035	0.83116	0.51063	0.8542	0.000819	0.000819	0.000819
5	0.83553	0.46923	0.87002	0.89581	0.81901	0.86614	0.60301	0.79322	0.4487	0.8464	0.000789	0.000789	0.000789
6	0.80082	0.44899	0.86339	0.89822	0.84663	0.88786	0.62421	0.74993	0.39511	0.84031	0.000759	0.000759	0.000759
7	0.77244	0.43095	0.85656	0.91321	0.83523	0.88816	0.62913	0.72145	0.40123	0.8347	0.000729	0.000729	0.000729
8	0.74818	0.41643	0.8536	0.89411	0.84199	0.88404	0.63552	0.72114	0.39554	0.8361	0.000699	0.000699	0.000699
9	0.7333	0.40536	0.85021	0.90743	0.85066	0.89085	0.63256	0.71834	0.43432	0.83454	0.000669	0.000669	0.000669
10	0.71744	0.39738	0.84573	0.90942	0.85541	0.89842	0.65264	0.68983	0.35958	0.82917	0.000639	0.000639	0.000639
11	0.7038	0.38866	0.84416	0.92376	0.85611	0.90411	0.66214	0.67426	0.35211	0.82692	0.000609	0.000609	0.000609
12	0.68957	0.38084	0.84121	0.91385	0.85244	0.90109	0.66543	0.66047	0.3424	0.82426	0.000579	0.000579	0.000579
13	0.67926	0.37331	0.83931	0.92548	0.86506	0.91248	0.67997	0.63596	0.33244	0.81955	0.000549	0.000549	0.000549
14	0.66586	0.36488	0.83647	0.91782	0.85218	0.90375	0.68022	0.65061	0.33181	0.82386	0.000519	0.000519	0.000519
15	0.6553	0.35975	0.83507	0.93688	0.84819	0.90505	0.68363	0.63829	0.33044	0.82067	0.000489	0.000489	0.000489
16	0.64498	0.35417	0.83357	0.92901	0.8567	0.91121	0.68463	0.63872	0.31937	0.82162	0.000459	0.000459	0.000459
17	0.63894	0.34933	0.83214	0.92031	0.85929	0.90992	0.6912	0.61355	0.30995	0.81757	0.000429	0.000429	0.000429
18	0.62922	0.34497	0.82965	0.9213	0.86934	0.91125	0.69338	0.61081	0.31588	0.8163	0.000399	0.000399	0.000399
19	0.62102	0.34	0.82865	0.92641	0.85852	0.91048	0.69739	0.60606	0.31848	0.81574	0.000369	0.000369	0.000369
20	0.6137	0.33626	0.8277	0.91821	0.86459	0.91419	0.70451	0.60027	0.30751	0.81505	0.000339	0.000339	0.000339
21	0.60678	0.32593	0.82383	0.92307	0.8657	0.91491	0.70449	0.59605	0.2992	0.81314	0.000309	0.000309	0.000309
22	0.58976	0.31855	0.82143	0.92025	0.86409	0.912	0.7077	0.58586	0.29953	0.81298	0.000279	0.000279	0.000279
23	0.57663	0.3121	0.81836	0.91684	0.87848	0.9181	0.7114	0.57662	0.29155	0.81168	0.000249	0.000249	0.000249
24	0.565	0.30583	0.81607	0.92641	0.87044	0.91726	0.71586	0.57161	0.28771	0.8109	0.000219	0.000219	0.000219
25	0.55487	0.30091	0.81492	0.9223	0.87634	0.92429	0.71992	0.56568	0.28681	0.8096	0.000189	0.000189	0.000189
26	0.54219	0.29462	0.81238	0.92799	0.87083	0.92016	0.72305	0.5567	0.28643	0.80777	0.000159	0.000159	0.000159
27	0.53293	0.28962	0.81215	0.93051	0.87515	0.92414	0.72702	0.55326	0.2806	0.80815	0.000129	0.000129	0.000129
28	0.52226	0.28413	0.81041	0.92553	0.87743	0.92265	0.72625	0.55055	0.27606	0.80784	9.91E-05	9.91E-05	9.91E-05
29	0.51321	0.27964	0.80897	0.93069	0.87901	0.92736	0.7299	0.54522	0.2766	0.80652	6.91E-05	6.91E-05	6.91E-05
30	0.50563	0.27588	0.80875	0.93521	0.87646	0.92654	0.72926	0.54437	0.27682	0.80644	3.91E-05	3.91E-05	3.91E-05

제안하는 모델

학습 출력 결과 (default)

- (Box, Class, DFL) Loss : 꾸준히 감소하여 약 0.4 ~ 0.8
- precision = 0.92, recall = 0.85, mAP(@0.5) = 0.92, mAP(@0.5:0.95) = 0.7



제안하는 모델

학습 출력 결과 (custom)

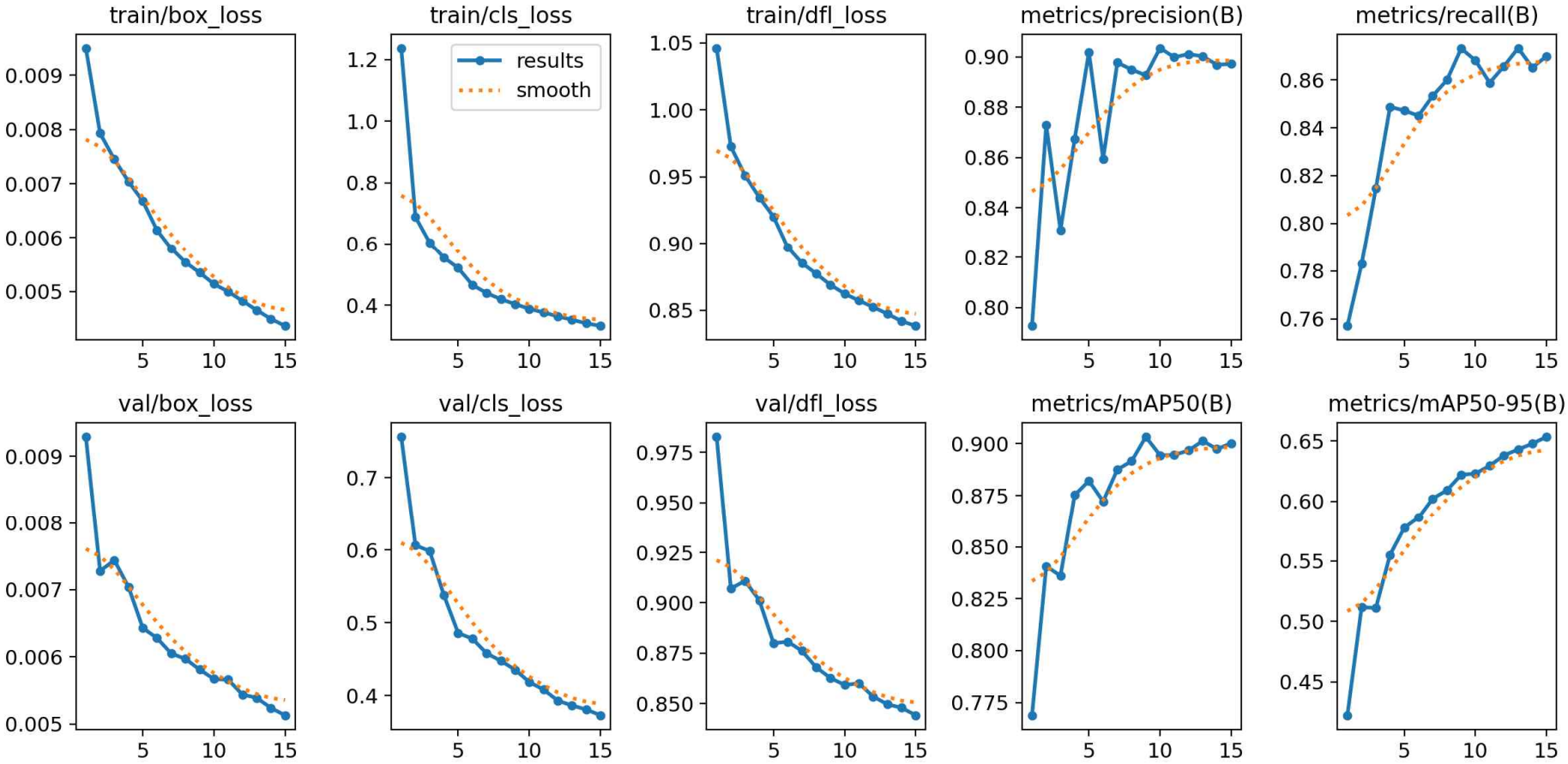
- 재현율, 정밀도, mAP
- 학습 시간 : 1epoch 당 약 20분

epoch	train/box_	train/cls_lc	train/dfl_lc	metrics/pr	metrics/re	metrics/m	metrics/m	val/box_lo	val/cls_los	val/dfl_los	lr/pg0	lr/pg1	lr/pg2
1	0.0095	1.2378	1.046	0.79278	0.75707	0.76886	0.42229	0.00929	0.75618	0.98272	0.000303	0.000303	0.000303
2	0.00793	0.68819	0.97251	0.87291	0.78286	0.84066	0.51178	0.00728	0.60628	0.90715	0.000566	0.000566	0.000566
3	0.00745	0.60436	0.95094	0.83104	0.81456	0.83603	0.51106	0.00744	0.59837	0.91086	0.000789	0.000789	0.000789
4	0.00704	0.55462	0.93459	0.86732	0.84871	0.87521	0.55518	0.00704	0.53762	0.90126	0.000729	0.000729	0.000729
5	0.00668	0.52105	0.9201	0.90181	0.84716	0.88211	0.57765	0.00643	0.48591	0.87996	0.000669	0.000669	0.000669
6	0.00614	0.46561	0.897	0.85942	0.845	0.87221	0.58625	0.00628	0.478	0.88071	0.000609	0.000609	0.000609
7	0.0058	0.43946	0.88514	0.89768	0.85333	0.88755	0.60219	0.00605	0.45804	0.87614	0.000549	0.000549	0.000549
8	0.00554	0.41933	0.87716	0.8949	0.8599	0.89162	0.60922	0.00597	0.44765	0.86799	0.000489	0.000489	0.000489
9	0.00535	0.40347	0.86883	0.89263	0.87307	0.90334	0.62201	0.00581	0.43556	0.86271	0.000429	0.000429	0.000429
10	0.00514	0.38805	0.86227	0.90346	0.86806	0.89446	0.62301	0.00567	0.41859	0.85925	0.000369	0.000369	0.000369
11	0.005	0.37664	0.85709	0.8998	0.85876	0.89461	0.6295	0.00566	0.40823	0.86004	0.000309	0.000309	0.000309
12	0.00483	0.36365	0.85235	0.90108	0.86563	0.89676	0.63809	0.00544	0.39317	0.85354	0.000249	0.000249	0.000249
13	0.00466	0.35287	0.84727	0.90017	0.87323	0.90137	0.643	0.00539	0.38641	0.84967	0.000189	0.000189	0.000189
14	0.0045	0.34109	0.84181	0.89682	0.86511	0.89744	0.64781	0.00524	0.38096	0.84793	0.000129	0.000129	0.000129
15	0.00437	0.33285	0.83831	0.89717	0.86987	0.90024	0.65346	0.00513	0.37302	0.84422	6.91E-05	6.91E-05	6.91E-05

제안하는 모델

학습 출력 결과 (custom)

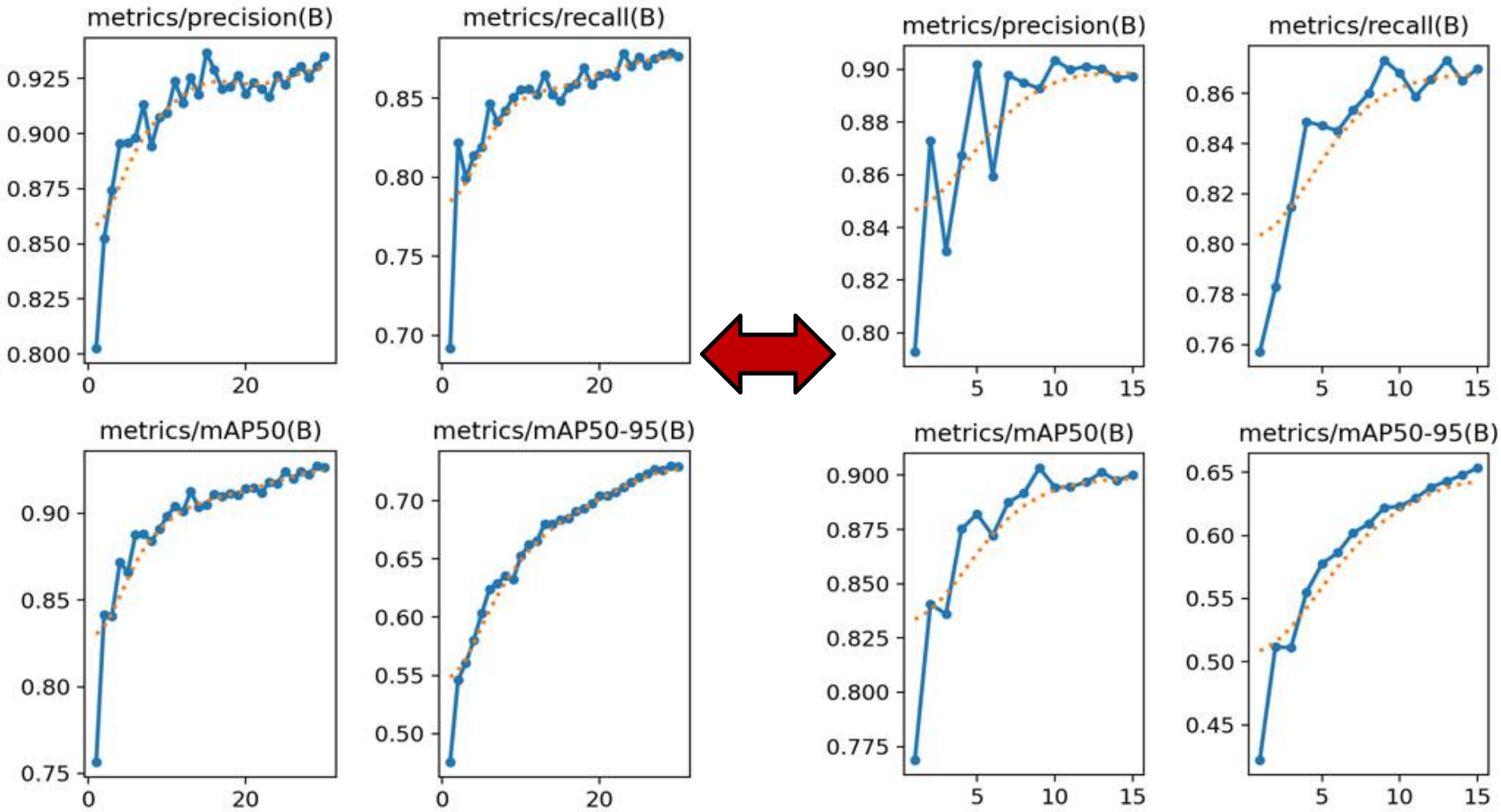
- (Box, Class, DFL) Loss : 꾸준히 감소하여 약 0.005 ~ 0.4
- precision = 0.86, recall = 0.86, mAP(@0.5) = 0.9, mAP(@0.5:0.95) = 0.65



제안하는 모델

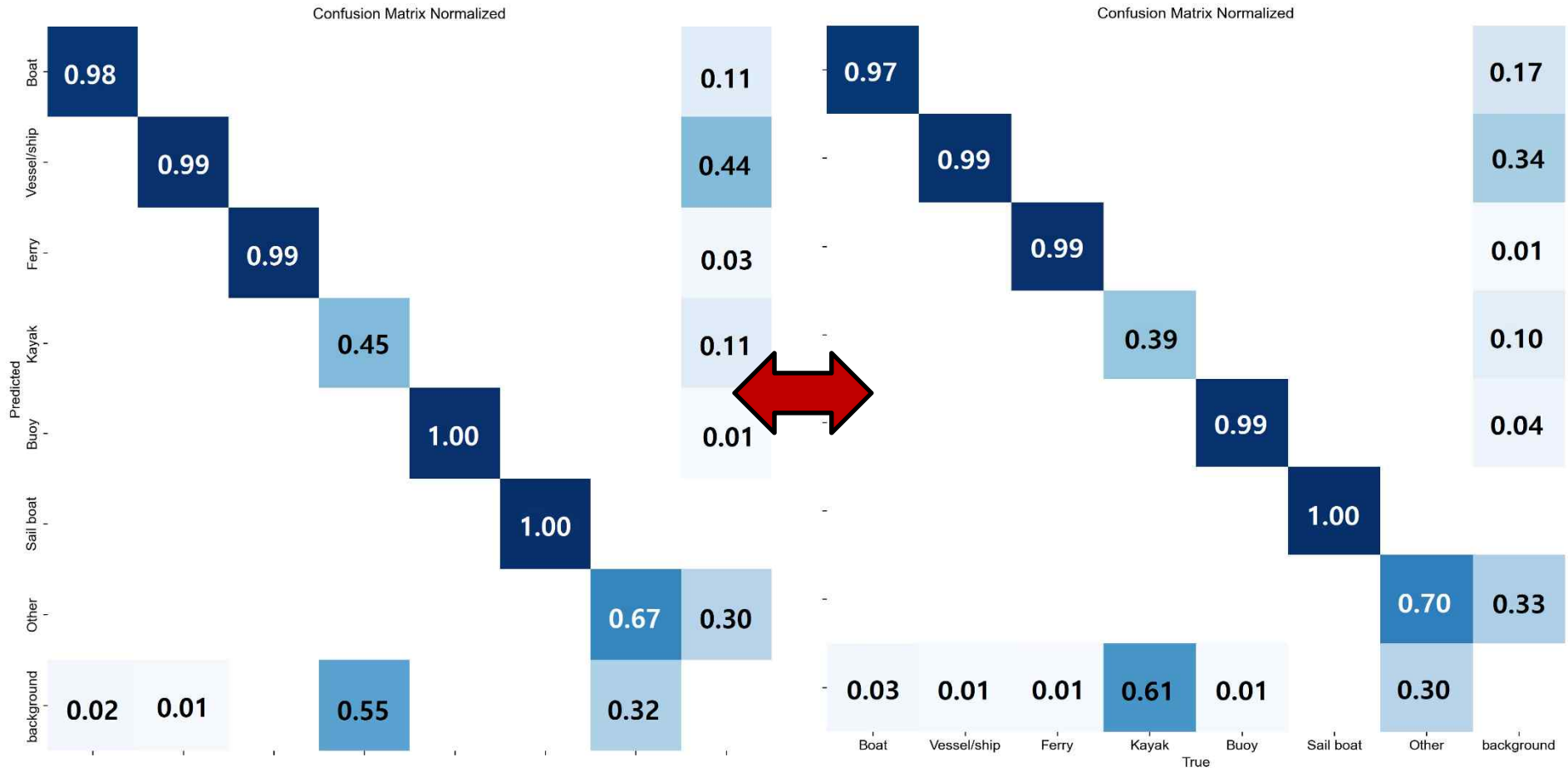
학습 곡선 및 성능 비교 (default VS custom)

- Default 모델이 전체적으로 더 높은 성능을 보임
- Custom 모델도 그래프 우상향 경향 → epoch 높일 시 우수한 성능 보일 것으로 예상



제안하는 모델

학습 곡선 및 성능 비교 confusion matrix (default VS custom)



제안하는 모델

검출 결과


Vessel/ship 0.91
Vessel/ship 0.87
Vessel/ship 0.51
Other 0.81
Other 0.71
Vessel/ship 0.92
Other 0.80

제안하는 모델

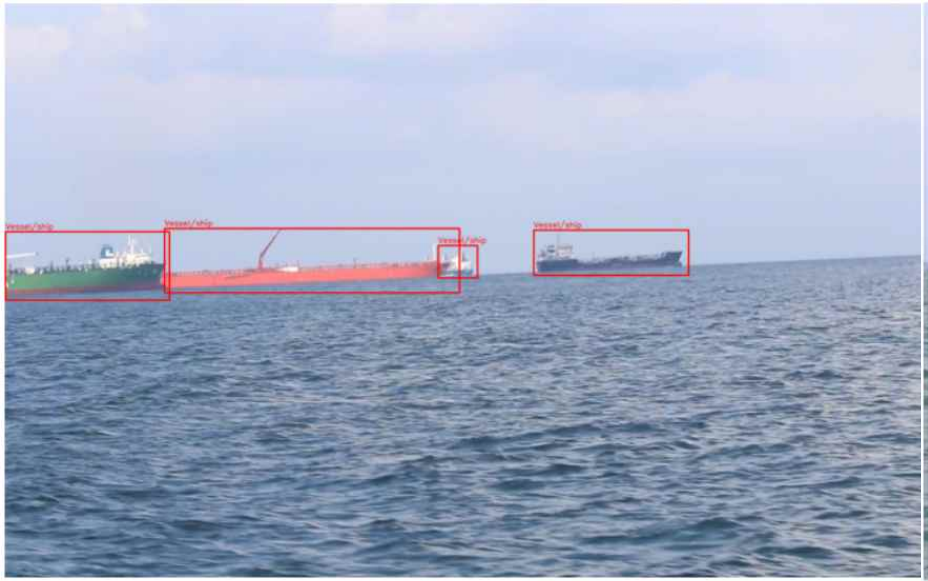
검출 결과


Vessel/ship 0.93
Vessel/ship 0.85
Vessel/ship 0.96
Vessel/ship 0.90

비교 모델

비교를 위해 트레인 조건을 변경하여 단시간 훈련 진행
-> 모든 지표상 성능이 1번 비교모델에 비해 현저히 저하됨을 확인함
epochs=5, batch=10, imgsz=300

클래스	모델 1 Precision	모델 1 Recall	모델 1 mAP50	모델 1 mAP50-95	모델 2 Precision	모델 2 Recall	모델 2 mAP50	모델 2 mAP50-95
Boat	0.957	0.983	0.993	0.787	0.942	0.976	0.968	0.668
Vessel/ship	0.985	0.995	0.995	0.915	0.930	0.979	0.989	0.725
Ferry	0.968	0.996	0.993	0.828	0.931	0.823	0.925	0.444
Kayak	0.718	0.491	0.666	0.295	-	-	-	-
Buoy	0.987	0.996	0.995	0.812	0.000	0.000	0.000	0.000
Sail boat	0.989	1.0	0.995	0.934	-	-	-	-
Other	0.913	0.693	0.854	0.538	0.612	0.411	0.459	0.150



rocessing image: C:₩Users₩User₩Desktop₩boat₩dataset_preparation₩images₩train₩MVI_0788_VIS_OB_fram

rocessing image: C:₩Users₩User₩Desktop₩boat₩dataset_preparation₩images₩train₩MVI_0789_VIS_OB_frame_0261.jpg

비교 모델

YOLOv8s,m,l 학습 진행

문제점

- Loss 데이터에 대해 NAN으로 측정

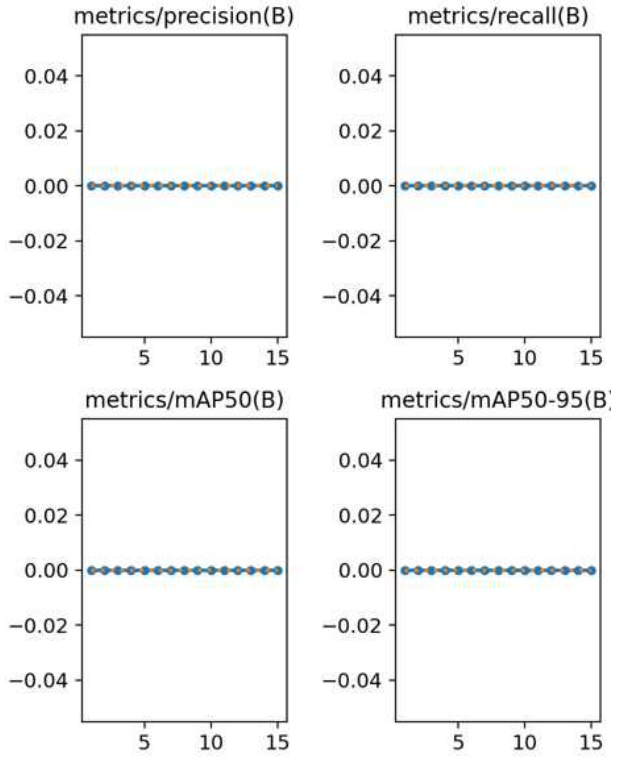
원인

- 특정 GPU(NVIDIA GeForce GTX 1650 Ti)에 대해 발생

해결

- AMP 비활성화 등 설정해봤으나 동일한 오류

epoch	train/box_	train/cls_lc	train/dfl_lc	metrics/pr	metrics/re	metrics/m	metrics/m	val/box_lo	val/cls_los	val/dfl_los	lr/pg0	lr/pg1	lr/pg2
1				0	0	0	0				0.000303	0.000303	0.000303
2				0	0	0	0				0.000566	0.000566	0.000566
3				0	0	0	0				0.000789	0.000789	0.000789
4				0	0	0	0				0.000729	0.000729	0.000729
5				0	0	0	0				0.000669	0.000669	0.000669
6				0	0	0	0				0.000609	0.000609	0.000609
7				0	0	0	0				0.000549	0.000549	0.000549
8				0	0	0	0				0.000489	0.000489	0.000489
9				0	0	0	0				0.000429	0.000429	0.000429
10				0	0	0	0				0.000369	0.000369	0.000369
11				0	0	0	0				0.000309	0.000309	0.000309
12				0	0	0	0				0.000249	0.000249	0.000249
13				0	0	0	0				0.000189	0.000189	0.000189
14				0	0	0	0				0.000129	0.000129	0.000129
15				0	0	0	0				6.91E-05	6.91E-05	6.91E-05



결론

논문(YOLOv5)과의 결과 비교

YOLOv5-L 모델 기준
precision = 0.688, recall = 0.514, mAP = 0.597(0.5) / 0.316(0.5:0.95)

동일한 데이터셋으로 YOLOv8 모델이 전체적으로 더 나은 성능 보임

Dataset	Copy & Paste	Network	Object Class							P	R	mAP	
			c1	c2	c3	c4	c5	c6	c7	0.5	0.5	0.5	0.5:0.95
SMD-Plus	None	YOLO-V4	0.160	0.622	0.868	0.632	0.00995	0.995	0.274	0.476	0.566	0.509	0.258
		YOLO-V5-S	0.372	0.691	0.827	0.569	0.00573	0.995	0.089	0.716	0.517	0.507	0.254
		YOLO-V5-M	0.588	0.882	0.816	0.615	0.00063	0.97	0.111	0.741	0.513	0.569	0.298
		YOLO-V5-L	0.673	0.789	0.846	0.571	0.0123	0.995	0.131	0.803	0.505	0.574	0.286
	Online	YOLO-V4	0.172	0.539	0.868	0.721	0.114	0.995	0.243	0.486	0.621	**0.522**	**0.308**
		YOLO-V5-S	0.471	0.864	0.869	0.549	0.162	0.995	0.123	0.650	0.536	**0.576**	**0.291**
		YOLO-V5-M	0.588	0.706	0.842	0.607	0.259	0.991	0.123	0.709	0.486	**0.588**	**0.338**
		YOLO-V5-L	0.714	0.806	0.828	0.582	0.232	0.995	0.147	0.811	0.534	**0.615**	**0.33**
	Offline	YOLO-V4	0.217	0.445	0.881	0.647	0.108	0.995	0.172	0.481	0.610	0.495	0.284
		YOLO-V5-S	0.475	0.386	0.887	0.603	0.0985	0.994	0.152	0.582	0.482	0.514	**0.291**
		YOLO-V5-M	0.49	0.809	0.852	0.603	0.0592	0.995	0.169	0.724	0.788	0.568	0.309
		YOLO-V5-L	0.618	0.789	0.847	0.667	0.0319	0.995	0.231	0.688	0.541	0.597	0.316

결론

결과 요약

데이터셋 구성 및 증강

- **SMD-Plus 데이터셋 활용:** 총 10,000개의 이미지와 라벨을 포함하는 SMD-Plus 공시 데이터셋을 사용하여 모델을 학습
- **다양한 데이터 증강 기법 적용:** 회전, mixup, copy&paste 등 다양한 증강 기법을 적용하여 모델의 일반화 성능을 향상

모델 학습 및 최적화

- **하이퍼파라미터 튜닝:** 최적의 하이퍼파라미터를 찾기 위해 학습률, 배치 크기, 옵티마이저 등을 조절

성능 평가

- **YOLOv5와 비교 우수한 검출 성능:** YOLOv8모델 학습 결과 mAP(Mean Average Precision, @0.5) 0.92, precision 0.92, 그리고 recall 0.85 을 달성하여 더 우수한 성능을 보인다고 할 수 있음
- **Custom 하이퍼파라미터와 비교:** epoch을 늘려 학습을 시켰을 시 더 나은 성능 보일 것으로 예상되나 30 epoch을 돌린 default 모델이 성능이 약소하지만 더 높게 나옴

결론

- 본 연구를 통해 YOLOv8 기반의 해상 객체 검출 시스템이 높은 성능과 효율성을 가지고 있음을 확인한 바, 해상 안전 및 자율 항해 시스템 분야에서 중요한 역할을 할 수 있음을 시사
- GPU 문제를 해결 해 더 다양한 모델 학습이 필요하며, 하이퍼파라미터 최적화 및 학습시간을 줄일 수 있는 방법을 모색해야 함

기타의견

이번 프로젝트를 진행하며 느낀 고찰(이준혁)

1. **COLAB 환경 진행 시**: COLAB 특성상 구글 드라이브와의 호환/연동은 좋음, 그렇지만 로컬 경로 지정이 들어가게 되면 경로상 이상이 없는데도 읽어올 수 없는 문제점들이 지속적으로 생김

-> 대용량 파일 작업 시에는 런타임 오류와 더불어 구글 드라이브의 용량까지 고려해야 하는 상황이 도래, 파이참 또는 주피터 노트북이 유리함을 체득하였음.

2. 경로찾기의 중요성/어려움: 이번 과제를 진행하며 가장 애로점이 많았던 것은 재대로된 경로를 설정했음에도 불러오지 못하는 경우가 빈번하였음.

-> / // ₩ 등의 경로지정 문제에 관하여 많은 고민을 할 수 있었음.

감사합니다