

```
Total BFLOPS 6.944
avg_outputs = 452201
Allocate additional workspace_size = 104.87 MB
Loading weights from backup/custom_yolov3_last.weights...
seen 64, trained: 6 K-images (0 Kilo-batches_64)
Done! Loaded 16 layers from weights-file
Detection layer: 15 - type = 27
/content/Test/Origin/Origin1.png: Predicted in 60.641000 milli-seconds.
OpenCV exception: show_image_cv
```

(위 사진은 외부 이미지를 테스트했을 시 출력되는 아웃풋)

다만 라벨링이 제대로 되었는지에 대한 평가를 위해 이미지를 출력하려했으나 Colab 상에서 강제된 드라이버 버전과 Cuda 버전과의 차이로 인해 OpenCV 단에서 문제가 일어나 확인이 어려웠습니다.

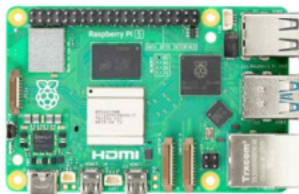


해당 검증 부분은 빠른 시일 내 Pi cam 선정 후 실제 화면을 통해 테스트해보겠습니다.

4. 2 차 검증

4.1 테스트 환경 구축

- 3.2 모델 학습 과정에서 YoloV3 을 통해 예측해보았으나 **호환성 문제**로 결과 이미지가 생성되지 않고, GPU 가 있음에도 정적인 이미지에서 **객체 하나 인식하는데 60ms 가 소모**되는 것으로 보아 실시간 처리 용도로는 어려움이 있을 것으로 판단되었습니다.

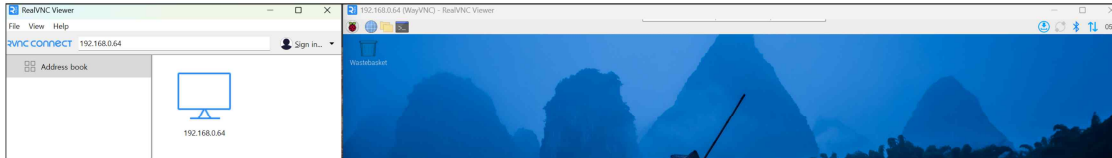
따라서 기존 라즈베리파이 모델 단일로 하지 않고, 신규 라즈베리파이 모델 중에서 개발보드+가속기+카메라 등이 하나로 판매될 정도로 호환성이 높은 제품을 탐색하였고 아래 제품으로 선정하였습니다.

개발보드	라즈베리파이5 4GB (쿼드코어 ARM Cortex-A76)	
카메라	라즈베리파이 카메라모듈 3 (자동 초점기능 탑재 화각 : 수평 66도, 수직41)	
가속기	AI HAT+ Hailo-8L (PCIe Gen 3로 통신)	

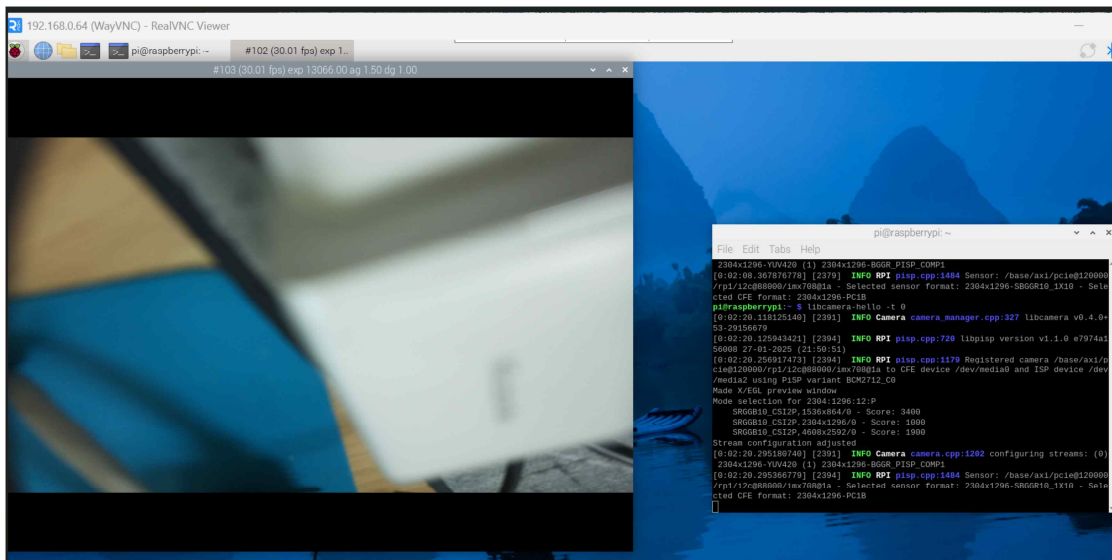
해당 제품으로 테스트하기 위해 OS 설치, SSH 환경 구축, 모듈 검증 과정을 진행하였습니다.

OS 설치 - Raspberry Pi Imager 사용. 저의 경우에 Mini hdmi cable 이 없어서 OS 설치와 SSH, VNC, 언어 세팅 등을 동시에 진행하였습니다.

SSH 환경 구축 - RealVNCViewer 사용 (아래 이미지는 직접 동작시킨 모습)



Camera 모듈 검증 - 관련 라이브러리 설치 후 동작 확인해보았습니다.



5. 학습 모델 변경 및 구조 개선

5.1 YoloV8n

- 이전 테스트 버전에서는 데이터 셋과의 호환성을 위해 원본 저작자가 시행한 YoloV3 모델 또는 V2 tiny 를 그대로 기용하려 했으나, 기존 모델 성능이 낮았던 점과 개발보드를 활용한 레퍼런스들이 모두 YoloV8 을 사용하여 YoloV8 모델 중에서 경량화된 n 모델을 사용하기로 하였습니다.

그에 따라, 기존 Colab 에서 테스트하던 코드 구조와 소스들을 변경해야 했습니다.

먼저 코드 구조를 이해하기 위해 레퍼런스들을 찾아보던 중, 제 상황과 거의 동일한 레퍼런스를 찾았습니다.

<https://colab.research.google.com/drive/1qrTS6c5M8TgtbVh7pWITo-2LnrQuFXAh?usp=sharing>

해당 코드 구조는 복잡한 것 없이

사용할 그래픽카드 정보를 보고 필요 라이브러리 설치 후 데이터 셋 구조만 맞춰서 바로 훈련시킨 뒤 ONNX 파일로 변환하는 것으로 보였습니다.

따라서 저 또한 기존 데이터셋을 YoloV8n 환경에 맞추기 위해 폴더 구조를 변경하였으며
훈련용 데이터와 평가용 데이터를 구분하여 코드 작성하였습니다.

정리하자면 기존 YoloV3 를 학습시켜봤던 테스트코드에서 YoloV8n 모델을 학습시키기 위한 코드로 변경하기 위해

- 데이터 셋 파일 구조 재생성
- 이미지, 라벨 파일 구분 작업
- 이미지, 라벨 1 대 1 매칭 여부에 대한 전수 검사
- .cfg 파일이 아닌, .yaml 파일 생성
- 이에 따라 현재 사용되지 않는 .ipynb , .cfg , Makefile 파일들 제거

등을 진행하였습니다.

5.2 학습 시작

학습 시작 시 많은 어려움이 있었는데 그 중 대표적인 게 아래 두 가지였습니다.

- Colab 에서 사용자 인터페이스 단의 상호작용이 일정 시간 없을 경우 자원을 회수해버리는 점
- epochs 를 50 으로 설정했더니 학습 시간이 너무 오래 걸리는 점

다행히 실행중인 코드에 영향을 주지 않으면서 자원 회수를 막을 방법으로
console 에서 리소스를 일정 주기로 확인하는 방법이 있었고

<https://sjkoding.tistory.com/79>

epochs 는 모델 성능을 재차 확인하면서 추가 학습을 늘려나가도 되지 않을까 싶어 10 으로 조정하였습니다.

또한 학습 중간중간 산출되는 데이터를 기반으로 시각화 작업 또한 진행했습니다.

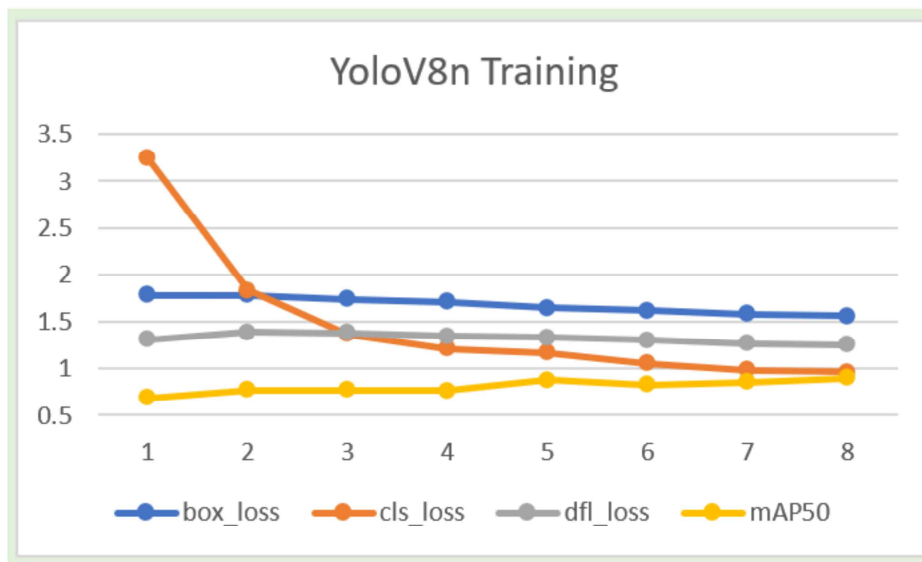
[1 차]

- 이미지 : 4010 장

- 훈련 모델 파라미터

```
results = model.train(  
    data=yaml_path,  
    epochs=10,  
    batch=16,  
    imgsz=640,  
    name='drone_detection'  
)
```

- 중간 결과 (8 번 학습 시.)



확실히 학습량이 누적될 수록 인식률이 좋아지는 것을 확인할 수 있었습니다.

다만 극적인 그래프 변화율이 없어서 모델이 완전히 학습 되었을 시 검증을 해보아야 합니다.

6. □

7.

[참고 자료]

Capstone_Design\1.PrecisionLandingModule\2.Version_1.0\3.LandingStation\1.Src\1. YOLOv8_detect_drones_model 파일의 “Detect_drones_Capstone.ipynb”, 이 외 산출물 참조