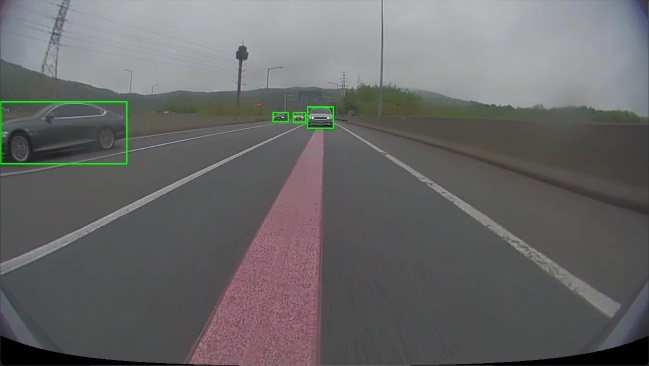
**데이터셋과 모델 성능 상관관계 분석 보고서**

1. **서론**
   1. **연구 배경**  
      : 인공지능 모델의 성능에서 주요한 변수 두가지는 모델의 구조와 학습에 사용되는 데이터셋이다. 본 프로젝트의 주요한 목표인 성능 향상을 위해, 제공받은 train set과 test set 각각의 특성을 분석하고 yolov5n, v8n 모델의 성능 테스트 결과를 기반으로 성능 향상 방법을 모색한다.
   2. **주요 아이디어**
      1. **클래스별 비율과 예측 정확도**  
         : 제공받은 데이터셋에 대해 총 6개의 클래스가 있기 때문에, 각 클래스별로 이미지 수와 포함하고 있는 객체의 수를 조사하여 클래스별 예측 정확도와의 상관 관계 파악이 필요하다.
      2. **각 클래스에 대해 BBox size별 예측 정확도 조사**  
         : 모델이 탑재될 주행 차량 후방카메라가 가진 시점의 특성상(사진1), 거리에 의한 객체의 BBox size의 차이가 다른 환경(ex. cctv, 사진2)보다 크게 벌어진다.   
         [사진 1] [사진 2]  
         모델이 실사용 될 환경 중 하나인 도로에서는 그 특성상 고속으로 주행중인 객체들도 있기 때문에, 작은 사이즈의 객체들에 대해 검출 정확도를 따로 조사할 필요가 있다. 또한 큰 사이즈의 객체일수록 가까이 있다는 경향성이 있기 때문에 중요도가 높아 마찬가지로 검출 정확도에 대한 조사가 필요하다.
      3. **필요한 추가 데이터셋 검토**  
         : 항목 i과 ii에서 BBox size별, 클래스별로 예측 정확도 조사를 할 필요가 있다고 판단했다. 따라서 각 요소가 모델의 성능에 영향을 미치는지 분석하여 필요한 추가 데이터셋의 특징들을 정해두고 그에 맞게 수집한다.
      4. **모델의 구조 변경 방향 모색**  
         : 만약 BBox의 size별 검출 정확도가 유의미하게 차이가 나는 경우 그 결과에 기반하여 향후 개발할 모델의 방향성을 설정한다. 예를 들어 모델이 주로 작은 객체에서의 정확도가 떨어진다면, 모델의 작은 객체 검출 능력이 강화되도록 연구 방향을 정한다.
2. **본론**
   1. **분석 방법**
      1. **Method\_dataframe.py**
         1. def get\_info\_from\_txt(file\_path)

return line\_list

: txt파일로부터 label 정보를 모두 읽어옴

* + - 1. def coordinate\_conveter(cx, cy, w, h, size):

return [x1, y1, x2, y2]

: txt파일에 있는 label은 BBox의 중심 좌표를 기준으로 서술되어 있기 때문에, 이후 시각화의 편의를 위해 좌상단, 우하단 꼭지점 좌표로 변환

* + - 1. def get\_size(center\_x, center\_y, w, h, img\_size):

return size

: BBox의 중심 좌표와 w, h, 이미지 사이즈를 통해 박스의 넓이를 계산

* + - 1. def get\_IoU(true\_box\_center, pred\_box\_center, size= (1280, 720)):

return iou

: true값의 label 하나와 pred값의 label 하나를 비교해 IoU를 계산하여 반환

* + - 1. def compare\_line\_to\_line(line\_true, line\_pred, iou\_th, size= (1280, 720)):

return [int(true\_bbox\_size), iou\_tf, class\_tf, cls\_dict[line\_pred[0]], iou, conf]

: true.txt의 line 하나와 pred.txtd의 line 하나를 비교하여 ground\_true의 bbox\_size, 예측 결과의 IoU가 thresh hold를 넘었는지, 클래스 일치 여부, 예측한 클래스값, IoU값, confidence값을 반환

* + - 1. def ftf\_by\_true(true\_file\_path, pred\_file\_path, iou\_th= 0.5, conf\_th= 0.3):

return detect\_condition\_list

: true.txt 파일 하나와 pred.txt 파일 하나를 비교해 true.txt의 각 line들에 대한 검출 상태들을 한꺼번에 반환, 방식은 true.txt의 line 하나를 pred.txt의 모든 line과 대조하여 그 결과들 중 IoU가 0 이상인 경우들만을 모아 IoU값, 클래스의 예측 여부, confidence 수치를 기반으로 최상의 line 하나를 선택해 detect\_condition을 도출함  
  
detect\_condition 결과에 대한 분류는 아래와 같음

Detect  
- IoU\_th < IoU, 클래스 일치, conf\_th < confidence  
- 검출이 완전히 됨  
conf\_lack  
- IoU\_th < IoU, 클래스 일치, conf\_th > confidence  
- 검출이 됐지만 confidence가 부족   
Detect\_clsF  
- IoU\_th < IoU, 클래스 불일치, conf\_th < confidence  
- 객체의 위치는 제대로 찾았지만 클래스 오분류  
pos\_clsT  
- IoU\_th > IoU > 0 , 클래스 일치, conf\_th < confidence  
- 객체의 위치를 근접하게 찾았으며 클래스가 일치하지만 confidence부족  
pos\_clsF  
- IoU\_th > IoU > 0, 클래스 불일치, conf\_th < confidence  
- 객체의 위치를 근접하게 찾았지만 클래스 오분류, confidence 부족  
False  
- IoU = 0  
- 검출 실패

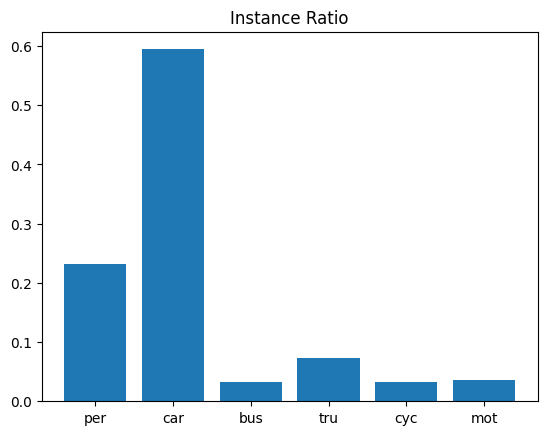
* + - 1. def get\_meta\_df(true\_label\_path, pred\_label\_path, iou\_th= 0.5, conf\_th= 0.1):

return result\_df

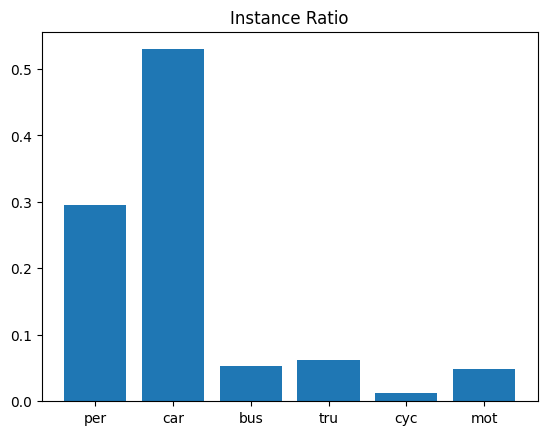
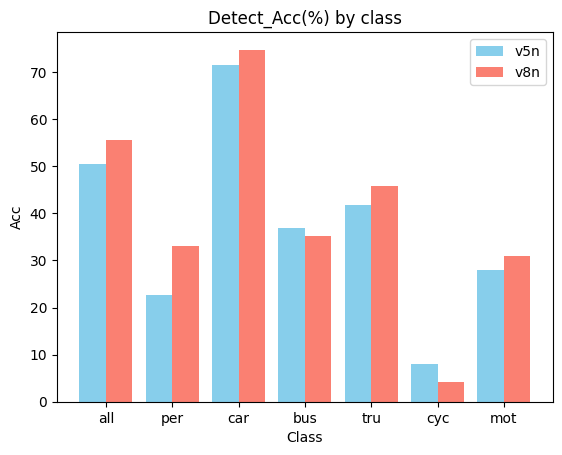
: ground ture인 true.txt 파일들 전부와 NPU에서 모델을 실행한 결과인 pred.txt 파일 전부를 비교하여 객체의 검출 결과를 데이터 프레임 형태로 반환

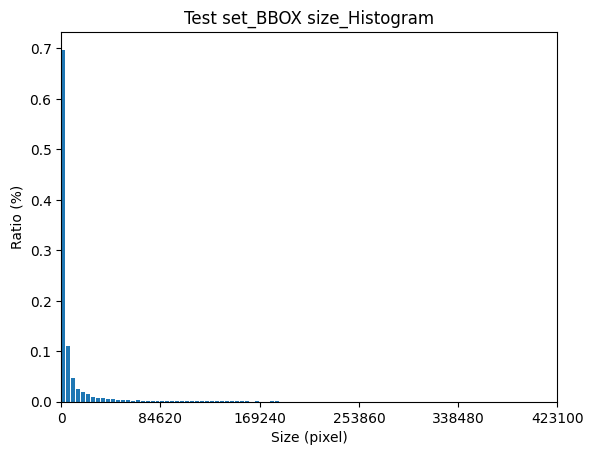
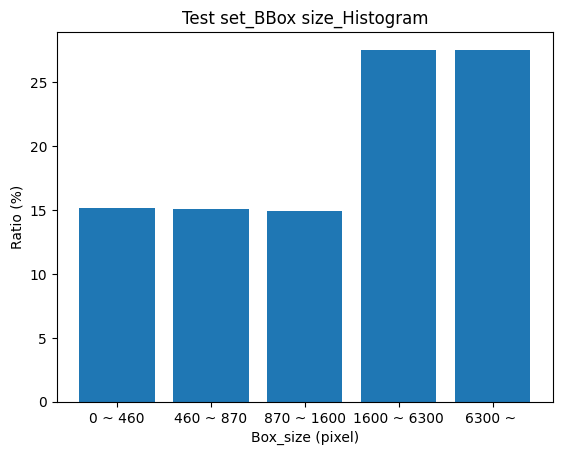
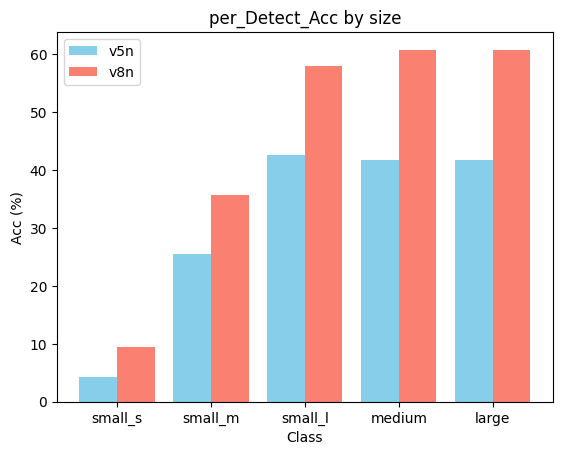
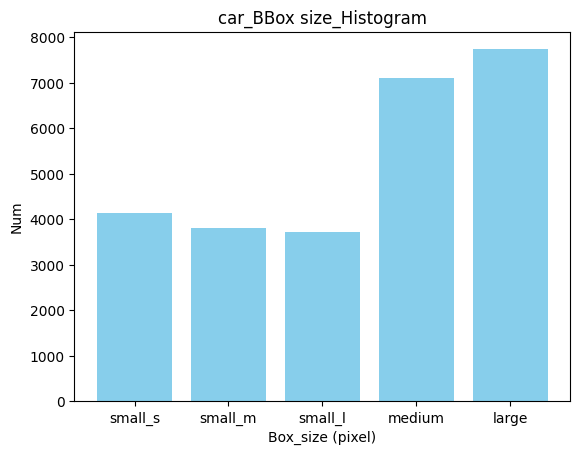
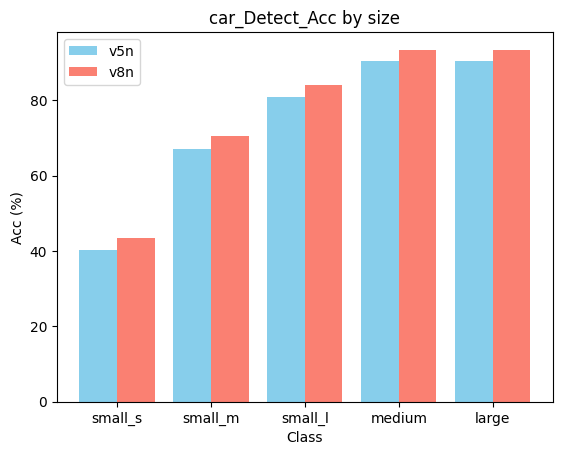
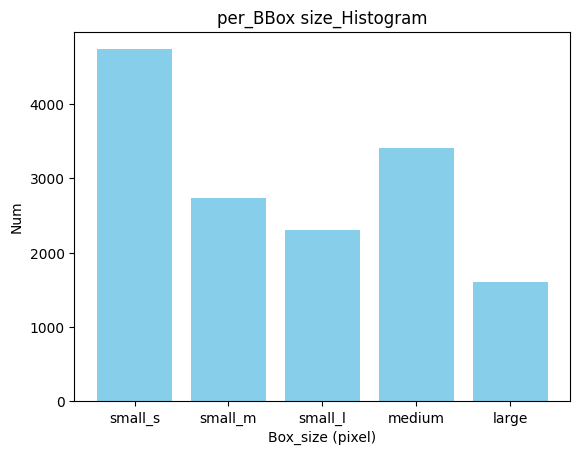
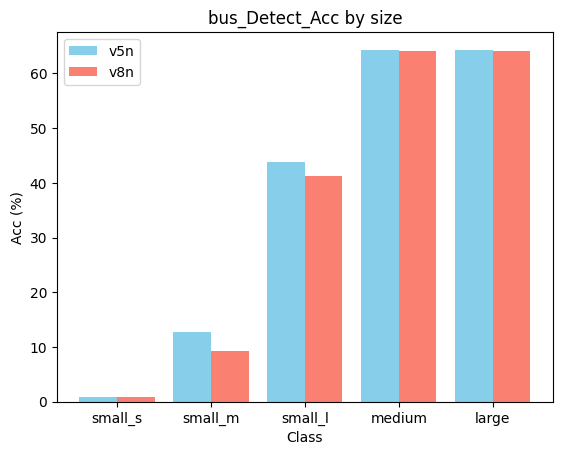
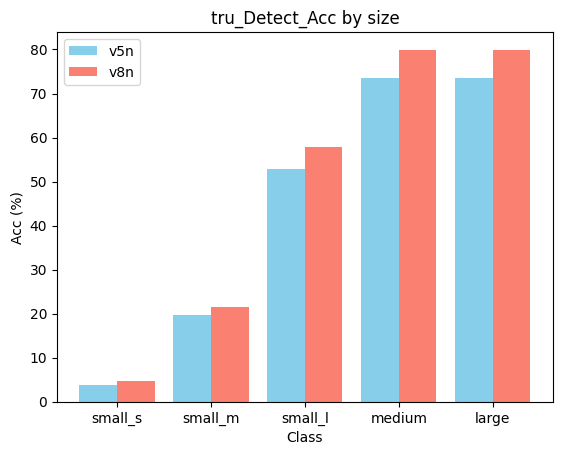
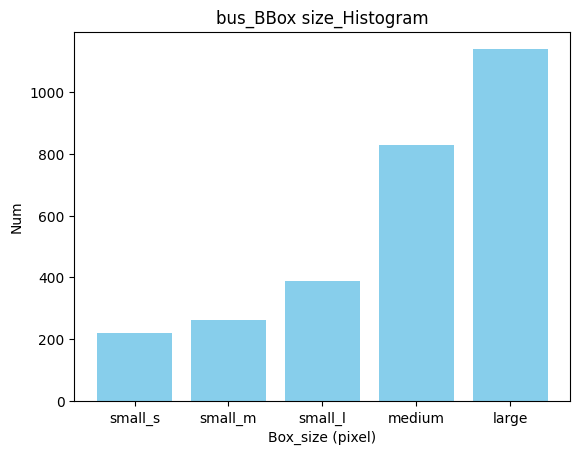
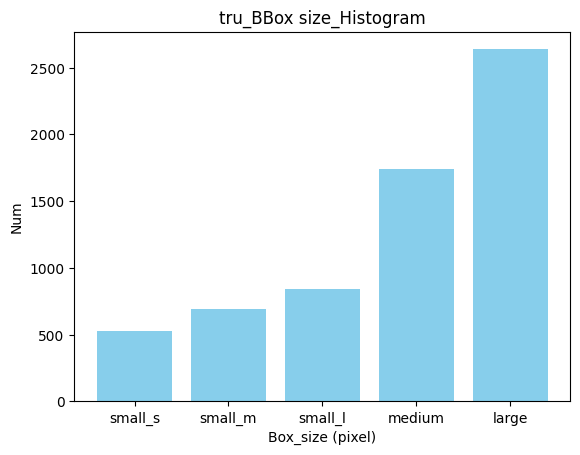
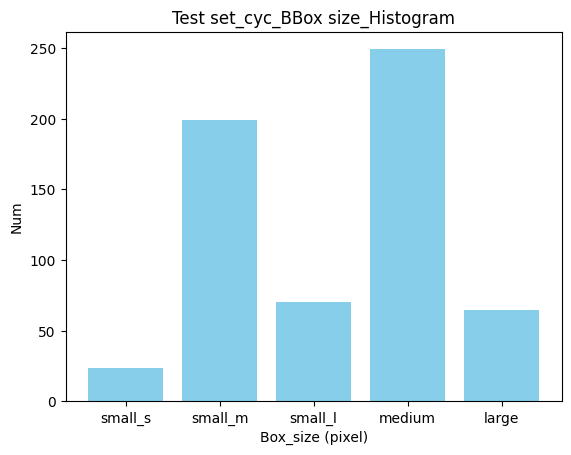
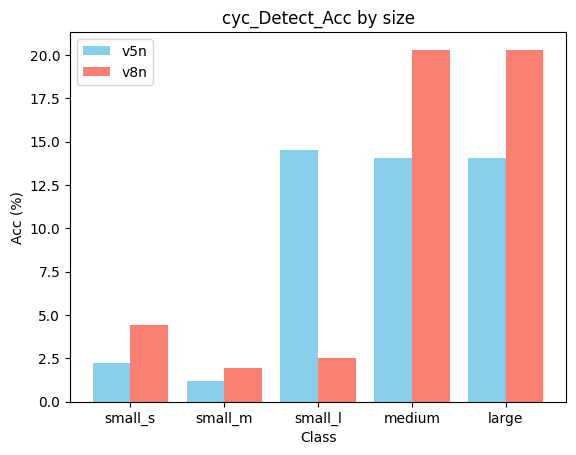
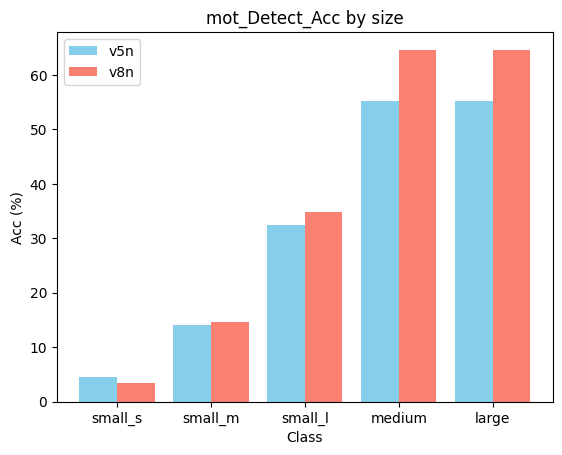
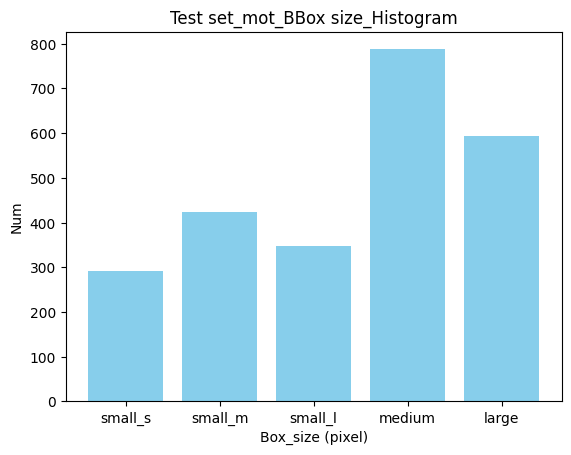
* + 1. **분석 과정**
       1. 모델을 NPU에서 실행한 결과를 산출한다.
       2. ground true와 비교하여 meta data를 데이터 프레임의 형태로 산출한다.
       3. 객체의 클래스와 BBox size를 기준으로 detect\_condition을 구분한다.
       4. detect\_condition이 Detect인 객체들은 Recall(재현율)과 같음, 실작동에서는 pred.txt에서 confidencd, IoU를 기준으로 최종 판단을 실행할거라고 가정했기 때문에 Recall을 기준으로 성능을 평가했다.
  1. **분석 결과**
     1. **클래스별 비율과 예측 정확도**

|  |  |  |
| --- | --- | --- |
| Class | Ratio(%) | Instance Num |
| Person | 23.19 | 20605 |
| Car | 59.46 | 52833 |
| Bus | 3.19 | 2836 |
| Truck | 7.24 | 6438 |
| Cycle | 3.34 | 2968 |
| Motorcycle | 3.58 | 3182 |

* + - 1. Train set의 클래스별 비율  
         : 학습에 사용된 데이터에서는 Bus와 Cycle, Motorcycle의 비율이 적으며 특히 객체수가 부족한 것으로 보인다. Yolo 공식 문서에서는 클래스별로 10000개 이상의 인스턴스 수를 권장하기 때문에, 만약 학습후에 해당 클래스들의 정확도가 떨어진다면 추가 데이터 수집을 고려해야 한다.  
         <https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/>

|  |  |  |
| --- | --- | --- |
| Class | Ratio(%) | Instance Num |
| Person | 29.52 | 14780 |
| Car | 52.95 | 26508 |
| Bus | 5.28 | 2644 |
| Truck | 6.15 | 3080 |
| Cycle | 1.21 | 606 |
| Motorcycle | 4.88 | 2441 |

* + - 1. Test set의 클래스별 비율  
         : test set에서도 마찬가지의 경향성을 보인다. 따라서 모델의 성능을 평가함에 있어 비중이 적은 클래스에서의 예측 정확도가 떨어지는 경우, 전체 성능 지표에 대한 영향력이 작기 때문에 단순히 전체 mAP만 비교하면 모델이 가진 취약점을 놓칠 수 있다. 따라서, 모델의 구조를 변경하고 성능을 평가할 때 클래스별 예측 정확도에 대한 관찰도 필요하다.
      2. 기본 Train set으로 학습한 v5n, v8n 모델의 클래스별 예측 정확도  
         : 전체 클래스에 대한 성능은 v8n 모델이 약 5% 정도 높게 나왔다.  
          두 모델의 경향성을 비교했을 때, 먼저 공통점은 Car에서의 정확성이 큰 차이로 높고, 그 다음으로 Truck과 Bus에 대한 정확성이 높고, Cycle에서 특히 낮은 성능을 보인다는 것을 알 수 있다. 이는 Train set에서 Car에 대한 데이터가 충분하기 때문이라고 해석할 수 있다. 하지만 Truck, Bus, Motorcycle에 대해서는 단순히 데이터의 부족이라고 해석하기 보다는 BBox size별로 정확도를 비교하여 추가적으로 검토할 필요가 있다.  
          또한 Person의 학습 데이터가 가장 많은데도 Bus와 Truck에 비해 예측 정확도가 떨어지는 문제점을 발견했는데 마찬가지로 BBox size별로 정확도를 비교하여 분석할 필요가 있다.  
          차이점은 Person에서 드러나는데 v8n의 경우가 약 10%로 더 높아 유의미한 차이를 보이고 있다. 이 결과는 모델의 구조적인 차이로 인한 결과일 가능성이 크다고 판단해, 해당 클래스에 대해 마찬가지로 BBox size별 정확도를 이용해 추가적인 검토가 필요하다.

* + 1. **각 클래스에 대해 BBox size별 예측 정확도 조사**
       1. BBox size 구분 방법  
          : 먼저 Test set의 모든 객체들의 BBox size를 조사하여 히스토그램을 만들어 분석했다. 해당 데이터 셋에서는 423100이 가장 큰 BBox\_size이며, 그래프에서 알 수 있듯이 작은 사이즈에 압도적으로 몰려있다.  
            
          따라서 1280 \* 720 크기의 이미지를 default로 하여, BBox의 w픽셀수와 h픽셀수를 곱해 내부 픽셀의 개수를 박스의 크기로 상정했으며  
            
          구간은 [0 ~ 460], [460 ~ 870], [870 ~ 1600], [1600 ~ 6300], [6300 ~ 921600]으로 나누었다. (921600 = 1280 \* 8=720) 0 ~ 1600 크기의 객체가 전체의 약 45%를 차지하며 이하 small size(s, m, l) 객체라고 분류한다. 1600 ~ 6300 크기의 객체는 medium size, 6300을 초과하는 크기의 객체는 large size로 분류한다. 해당 구간대로 분류하면 small, medium, large 객체의 비율은 45% : 27.5% : 27.5%로 구성된다.
       2. Car  
          : 해당 클래스에서는 객체의 크기가 작아질수록 예측 정확도가 하락하는 현상을 확인할 수 있다. 학습에 사용된 데이터를 살펴봤을 때, small\_s 크기의 객체가 약 7천장으로 충분하므로 데이터 셋을 추가하기 보다는 모델의 구조 변경을 통한 성능 향상이 먼저라는 결론을 얻을 수 있다.
       3. Person  
          : 해당 클래스에서는 오른쪽 하단의 그래프를 보았을때 small\_s 사이즈의 객체가 가장 많으며, small\_m, small\_l 사이즈의 객체도 큰 비중을 차지한다. 결국 small(s, m, l)사이즈의 객체가 약 66%의 비율을 차지하는데, 왼쪽 하단의 그래프에서 알 수 있듯이 small\_s 객체에서 특히 성능이 떨어지며 small\_m 객체에서도 낮은 성능을 보인다.  
           위의 데이터 분포 분석 결과는 **[i. 클래스별 비율과 예측 정확도 - 3]** 에서 확인한 person의 학습 데이터가 충분한데도 예측 정확도가 떨어지는 문제점에 대한 설명이 될 수 있다.  
           또한 Car 클래스와 비교했을 때, small\_s 객체에 대한 정확도는 약 4배의 차이가 나는 문제점이 있다. 학습에 사용된 small\_s 객체의 숫자를 확인한 결과 Person의 경우 약 5천장, Car의 경우 약 7천장으로 두 경우 모두 데이터의 개수는 충분하므로 해당 문제점은 모델의 작은 객체에 대한 감지 성능을 향상하여 해결해야 한다는 결론을 얻었다.
       4. Bus, Truck  
          : 먼저 두 클래스의 BBox\_size별 예측 정확도를 비교했을 때, 객체 크기의 감소에 따른 성능 하락은 유사하다는 것을 확인할 수 있다. 이에 대한 분석을 위해 두 클래스의 객체 크기별 Test set의 분포도를 확인했을 때, 분포의 형태가 비슷하기 때문이라는 것을 알 수 있다.  
           하지만 Truck의 경우가 평균적으로 정확도가 높은데 이에 대한 분석을 위해 학습에 사용된 데이터 수를 확인해보면 Truck의 경우 총 객체 수가 25000개, Bus의 경우 5000개이다. 따라서 해당 문제점은 Bus 데이터의 보강으로 해결할 수 있다는 결론을 얻을 수 있다.
       5. Cycle  
          : 해당 클래스는 **[i. 클래스별 비율과 예측 정확도 - 3]**에서 가장 낮은 성능을 보였는데, 그에 대한 이유는 두가지의 관점에서 분석할 수 있다.  
           첫번째로, 학습에 사용된 객체의 수를 확인하면 약 3000개로 성능을 보장하기에 부족한 숫자이다.  
           두번째로는 Test set의 분포를 확인했을 때 small\_m 크기의 객체가 두번째로 비중이 높았는데, 다른 클래스들과 마찬가지로 해당 경우에 대한 예측 정확도가 떨어지기 때문이다. 게다가 Test set에 포함된 cycle의 객체 수는 총 600개로 성능을 평가하기에 충분한 숫자가 아니다.  
           따라서 앞서 말한 이유들이 합산되어 Cycle에 대한 예측 성능이 가장 낮게 나타난 것으로 분석할 수 있다. 이 문제점을 해결하기 위해서는 모델의 작은 객체 성능 향상이 필요할 뿐만 아니라 적절한 데이터셋의 추가 역시도 필요하다는 결론을 얻을 수 있었다.
       6. MotorCycle  
          : MotorCycle 클래스의 경우에는 학습에 사용된 객체의 수가 3200개로 Cycle과 비슷하지만, Test set의 분포가 주로 medium, large 크기의 객체들의 비중이 높기 때문에 성능이 Cycle에 비해 높게 나온 것으로 판단된다.  
           하지만 small\_s, small\_m 객체들에 대한 정확도는 많이 떨어지기 때문에 Cycle에서처럼 작은 객체의 비중이 큰 경우에 대해서 성능을 평가할 경우 현재와는 반대의 지표가 나올 것이라고 생각된다. 따라서 Cycle에서와 같은 방법으로 문제점을 해결해야 한다는 결론을 얻었다.
    2. **필요한 추가 데이터셋 검토**
       1. 데이터셋 분포 관점  
          : 향후 모델의 small 객체들에 대한 성능을 향상시키더라도 학습에 사용되는 데이터셋에 해당 객체들이 충분한 수만큼 포함되지 않는다면 성능 분석이 제대로 이루어지지 않을 수 있다. 예를 들어 모델은 충분한 능력이 생겼는데, 학습에 사용된 데이터가 부족하여 성능은 여전히 낮게 나오는 경우 올바른 분석이 어려울 것이다.  
           따라서 앞서 분석한 문제점이 발견된 클래스들에 대해서 객체의 사이즈별로 적절히 분배된 데이터셋을 추가한 후, 추가하지 않은 데이터셋으로 학습을 병행하며 결과를 분석할 필요가 있다.
       2. 클래스별 학습 데이터 양 관점  
          : 클래스별로, 객체의 크기별로 적절한 분포를 가진 학습데이터를 구성했더라도, 절대적인 수가 부족하다면 모델의 성능을 향상하기에는 충분하지 않을 수 있다.  
           또한 다른 특성을 가진 데이터를 추가하여 오히려 모델의 성능을 떨어뜨릴 수도 있기 때문에 점진적으로 데이터를 추가하며 모델의 성능 변화를 관찰해야 한다.

1. **결론 및 향후 연구 계획**
   1. **결론 요약**
      1. **클래스별 예측 정확도**  
         : 연구 과정에서 클래스의 학습에 사용된 데이터셋의 구성과 테스트에 사용된 테스트셋의 구성에 따라 예측 정확도가 낮거나 높은 이유들을 분석했다.  
          Car의 경우 데이터셋의 문제가 없으며, 모델의 전체적인 성능이 향상된다면 자연스레 해당 클래스에 대한 예측 정확도가 올라갈 것으로 판단했다. 하지만 Person의 경우 데이터셋에는 우선 문제가 없지만, 모델의 small 객체에 대한 성능이 특히 향상될 필요가 있다는 결론을 얻었다. 나머지 클래스들에 대해서는 공통적으로 크기별 분포와 환경이 비슷한 적절한 데이터셋의 추가와 모델의 small 객체에 대한 성능 향상이 필요하다고 판단했다.
      2. **BBox Size별 예측 정확도**  
         : 모든 클래스에 대해 관찰할 수 있는 경향성은 객체의 BBox Size가 작아질수록 예측 정확도가 감소한다는 것이다. 따라서 향후 모델의 구조를 변경하는 과정에서 small 객체에 대한 성능 향상이 필요하다는 결론을 얻었다.
      3. **데이터셋**  
         : 학습에 사용된 데이터셋과 테스트에 사용된 데이터셋들의 특성들을 분석했을 때, 일부 클래스들에 대해 적절한 데이터셋을 추가할 필요가 있다는 결론을 도출했다. 또한 성능을 테스트할 때에는 전체적인 지표만 볼 것이 아니라 세부적으로 관찰할 필요가 있다는 점과 가능하다면 테스트 셋에 추가적인 변경하는 방안도 고려할 필요가 있다는 것을 알 수 있었다.
   2. **향후 연구 계획**
      1. **모델의 구조 변경 방향**  
         : Yolo 모델의 구조를 분석했을 때, Neck에서 P3, P4, P5가 각각 작은 크기, 중간 크기, 큰 크기의 객체에 대한 정보들을 추출한다는 것을 알 수 있었다. 따라서 우리는 기존보다 더 작은 객체에 대한 정확성을 향상시킬 필요가 있기 때문에 기존 Yolo 모델에서 추가적으로 P2에서의 특징을 추출하면 더 작은 객체에 대한 성능이 증가할 것이라는 가설을 갖고 연구를 진행할 것이다.
      2. **수집할 데이터셋**  
         : 먼저 데이터의 수가 적고, 테스트에서 성능이 부족하게 나온 클래스들을 위주로 적절한 데이터를 수집한다. 이후 점진적으로 추가하여 학습을 진행해 이전과의 성능 비교를 하며 데이터의 추가 여부를 결정한다.