|  |
| --- |
| BABY DULLI TEAM(TEAM7) |
| AI OBJECT DETECTION |
| BASED ON  SYNTHETIC DATA |
|  |

|  |
| --- |
| 김상욱, 김신웅, 좌대현, Stefan |

**1. Introduction**

**1.1 Motivate**  
 Recently, chat gpt, a conversational artificial intelligence chatbot, has appeared and many people are using it. Chat GPT is also one of the deep learning programs. In addition to this, there are many areas where deep learning models are applied in various fields. However, in some cases, real-world data is not enough to train machine learning models. Synthetic data appeared to solve this problem. Synthetic data, which is data generated in a digital environment rather than directly collected in a real environment, is widely used. I wondered if these synthetic data could predict real data well. Among various types of synthesized data, we decided to use image data that is visually the easiest to understand. We want to predict real data based on image synthesis data and evaluate its performance.  
  
**1.2 Theoretical background**

**1.2.1 Object detection**

Object detection is a computer vision task that involves identifying and localizing objects within an image or video. Deep learning models for object detection are typically based on convolutional neural networks . These models consist of multiple layers of interconnected neurons designed to automatically learn and extract meaningful features from input data, such as images. The key idea behind object detection is to combine the power of CNNs for feature extraction with additional components that enable accurate object localization and classification.  
  
**1.2.2 Synthetic data**  
 Synthetic data refers to artificially generated data that is created using algorithms or models rather than being collected from real-world sources. It is designed to mimic the statistical properties and characteristics of real data. Synthetic data can be used as a substitute or complement to real data in various applications.  
  
**1.2.3 Hyperparameter**  
 Hyperparameters are configuration settings or variables that are set before the learning process begins in a machine learning or deep learning algorithm. Unlike model parameters, which are learned during training, hyperparameters are predefined by the user or researcher and affect the behavior and performance of the learning algorithm. They govern various aspects of the learning process and model architecture, such as the learning rate, regularization strength, network size, and number of iterations.

**1.3 research purpose**

We use train data, which consists of synthetic data, and test data, which is real photo data. Our experiment has two main purposes. The first is to find the location of the car. When there is a car in a random picture, locate the car and draw a square box. Inside the train data, there is a txt file with the coordinates of the car in each picture along with the synthetic data pictures. After learning this, see if you can draw a box by finding the location of the car well in the actual photo data. The second is to classify the type of car that has been located. There are 34 types of cars in the txt file containing the coordinates of traindata. By learning this, the car model in the photo is classified. The purpose of our study is to find out how well we can locate the object in the picture and classify the type of car after finding the object.

**2. Method**

**2.1 object detection model**

object detection model에는 크게 두 가지의 모델이 있다. one-stage detection과 two-stage detection이다.

**2.1.1 One-Stage Detection**

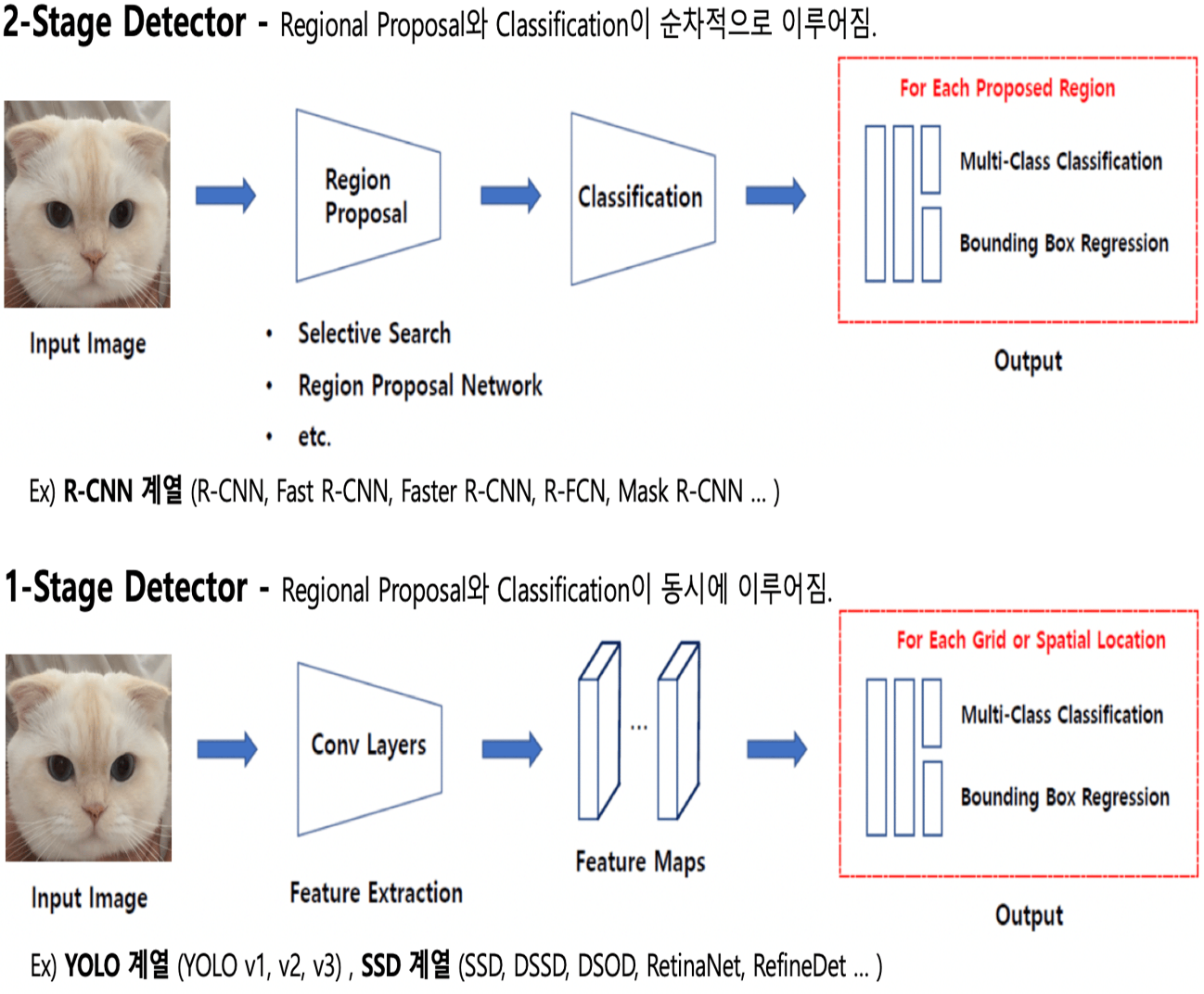
In one-stage detection methods, also known as single-shot detectors, the object detection process is directly performed in a single pass through the network. These methods aim to simultaneously predict the object classes and bounding box coordinates in a single step. One-stage detectors typically use a series of convolutional and pooling layers to extract features from the input image. These features are then passed through several additional convolutional layers responsible for predicting the class probabilities and bounding box coordinates for potential objects at different spatial locations and scales. One-stage detectors often use anchor boxes or default boxes that act as references for detecting objects. The predicted bounding boxes are then refined using post-processing techniques such as non-maximum suppression (NMS) to eliminate redundant detections.The advantages of one-stage detectors include their simplicity, real-time processing capability, and the ability to detect objects at multiple scales. However, they may struggle with accurately localizing small objects and handling object instances with significant scale variations.

Figure : One stage detection

**2.1.2 Two-Stage Detection**

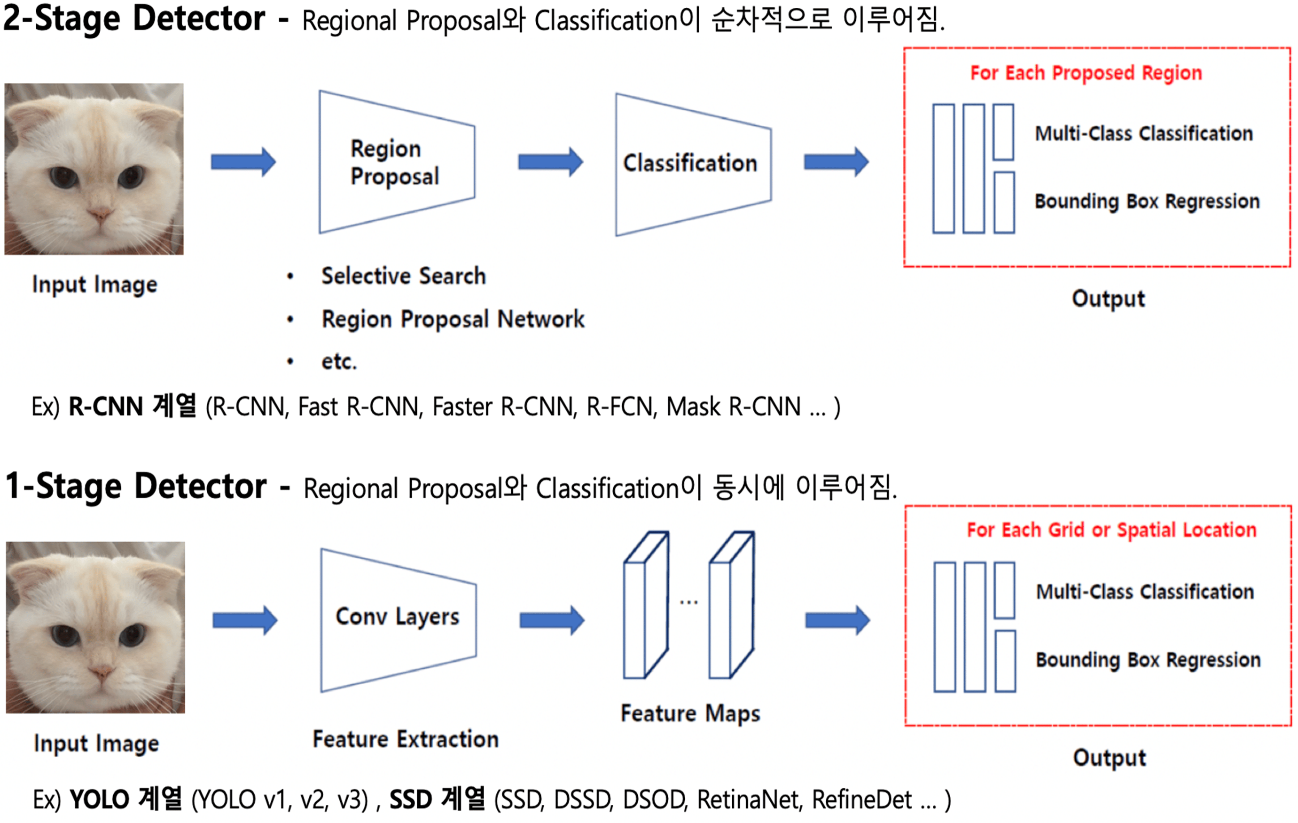
Two-stage detection methods, also known as region-based detectors, split the object detection process into two stages: region proposal and object classification/refinement. These methods first generate a set of potential object regions or proposals, and then these regions are further analyzed to classify and refine the bounding box predictions. In the first stage, a region proposal network is used to generate potential object regions by proposing a set of candidate bounding boxes. These candidate boxes, often referred to as region proposals, are determined based on their likelihood of containing objects. The RPN typically uses a set of anchor boxes and learns to predict the offsets and objectness scores for these anchors. In the second stage, the proposed regions are fed into a separate network to extract features and perform object classification and bounding box regression. This stage involves fine-tuning the region proposals, refining the bounding box coordinates, and predicting the class labels of the objects.

Figure : Two stage detection

**2.2 Compare Yolo&Fast-RCNN**

To decide which model we should choose for our project, we can compare the performance of two state-of-the-art models: faster-RCNN and YOLOv8.

**2.2.1 YOLO**

YOLO is a popular real-time object detection algorithm known for its speed and accuracy. It approaches object detection as a regression problem and performs detection directly on the full image in a single pass, rather than using a two-stage process. The YOLO algorithm divides the input image into a grid and predicts bounding boxes and class probabilities for objects within each grid cell.

**2.2.2 Fast R-CNN**

The faster-RCNN model is a two stage model, which consists of a Region Proposal Network (RPN) and a Fast R-CNN network. The RPN generates region proposals, which are then fed into the Fast R-CNN network for object classification and bounding box regression. The downside of using a two stage model is that passing through the two layers is slower than just passing through one. However the model does yield accuracy predictions, even when the images are warped. This is because the images have to be warped to a fixed size anyway when passed to the second stage of the model

**2.3 YOLOv8 architecture**

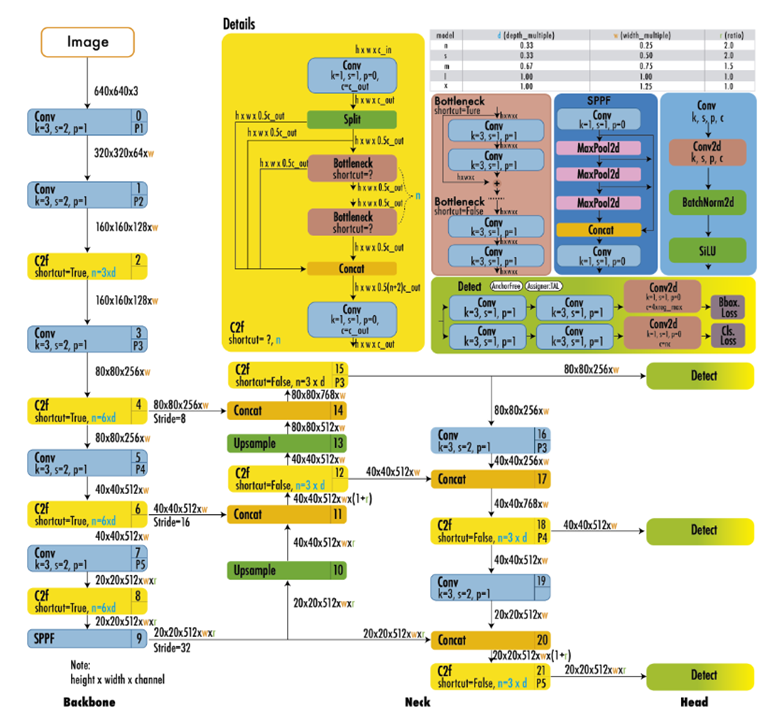
If we choose Fast-RCNN, do not benefit from this advantage that the model does yield accuracy predictions, even when the images are warped. since the images are all taken from the same perspective. As our dataset is large, and the YOLOv8 model is far faster to train than the faster-RCNN model whilst still yielding similar accuracy. So we choose YOLOv8 model. The YOLOv8 model can be divided into three parts; the backbone, neck and head.

Backbone: A series for Convolutional Layers. It is a modified version of CSPDarknet53 architecture, which extracts features from the input image.

Neck: A series of fully connected (dense) layers. It connects the backbone and head, responsible for feature fusion based on the PANet architecture.

Head: Predicts bounding boxes, objectness scores, and class probabilities, using the non

maximum suppression (NMS) algorithm to filter overlapping boxes and keep the most confident

**3. Experiment**

**3.1 train data set**

We have an image dataset consisting of 6,480 train data and 3,400 test data. The image shown in Figure [3-1] is a synthetic data generated in a digital environment, not an actual image. The train data contains text files for each class, which include the 34type class information of car and box coordinates. We will train the model based on this information.

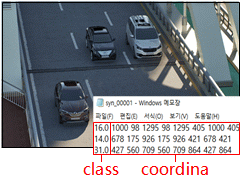
We allocated 10% of the data as the validation set, which is relatively small compared to the usual allocation of 20-30% of data for the validation set. The reason behind this decision is that we observed underfitting, where the model's size is quite large and optimization takes longer. In order to obtain a sufficient amount of data, we allocated only 10% for the validation set.  
**3.2 data process**

Figure -1 : Train data

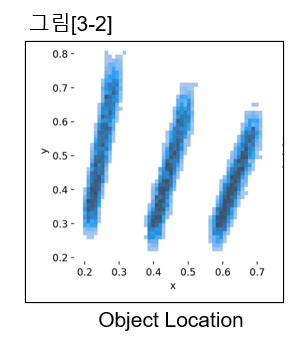
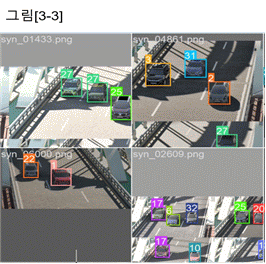
As evident in Figure [3-1], the road direction starts from the top left and continues towards the bottom right, suggesting that objects are expected to be detected within that coordinate range. However, when visualizing the actual coordinates where objects are detected, as shown in Figure [3-2], they are relatively evenly distributed, contrary to the direction of the road.   
 The reason for this is the utilization of augmentation techniques during model training, as depicted in Figure[3-3]. Various augmentations were applied, including horizontal flipping, horizontal shifting, and saturation adjustment, among others, to ensure effective classification of challenging test data and improve overall performance.

Figure -2

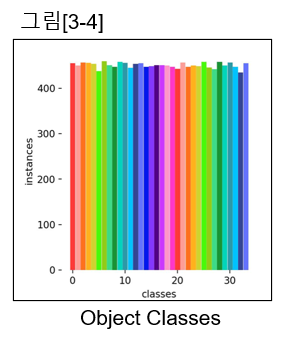


Figure -3 : Training image

After completing the training, we visualized the distribution, as shown in Figure [3-4], to assess whether the model learned uniformly across all classes. The visualization demonstrates a balanced training outcome, indicating that the model learned evenly across different classes.

Figure -4:object classes

**3.3 test image**

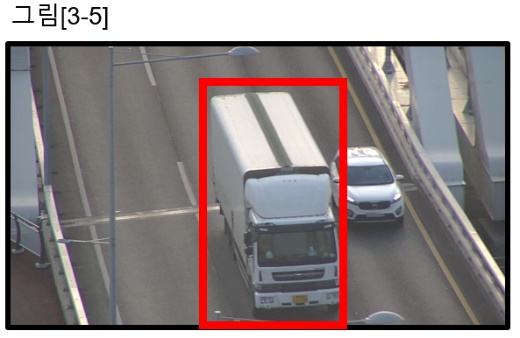
Among the test data images, there were instances of vehicle types, such as the one shown in Figure [3-5], that were not included in the class labels. We also aimed to examine how the model classifies such test images.

Figure -5 : Test image

**4. train process**

**4.1 YOLO모델 선택**

Yolov8모델 hyper parameter인 batch, imagesz, learning rate가 바꿨을 때 학습에 가장 큰 영향을 주는 요인이라 생각했고 주로 이것들을 바꾸면서 성능을 비교했다. Epoch은 높게 잡아두고 학습을 진행하는 과정에서 val\_cls loss와 val\_box loss들을 모니터링하고 줄어들지 않거나 충분히 줄어들었다 하는 시점의 weights 값을 저장해두고 학습이 진행되는 과정을 더 지켜보다 계속 loss가 줄지 않는다면 학습을 마무리하고 예측을 진행해주었다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명그리고 위 사진과 같이 Yolov8 모델은 Model size에 따라 세부모델로 yolov8n, yolov8s, yolov8m, yolov8l, yolov8x 이렇게 나뉜다. Yolov8n부터 yolov8x로 갈수록 층의 개수가 많아지고 그만큼 성능이 올라가는 것이다. 학습 시간이 더 걸리더라도 yolov8x모델을 써서 최대한 정확도를 높이려고 하였다.

**4.2 Hyperparameter 설정 및 성능**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **imgsz** | **batch** | **epoch** | **Lr** | **val/box\_loss** | **val/cls\_loss** | **Score**  **(map0.85)** | **RMK(비고)** |
| 640  \*640 | 7 | 129 | 0.001 | 0.11407 | 0.06415 | 0.6208 |  |
| 640  \*640 | 16 | 350 | 0.0001 | 0.11233 | 0.06538 | 0.58922 | predict imgsz  = (640, 640) |
| 640  \*480 | 10 | 70 | 0.0001 |  |  | 0.3333 |  |
| 1024  \*1024 | 7 | 99 | 0.001 | 0.08612 | 0.05331 | 0.64958 |  |
| 1024  \*1024 | 6 | 98 | 0.005 | 0.0933 | 0.06186 | 0.7348 | predict imgsz = (1024, 1024) |
| 1024  \*1024 | 7 | 130 | 0.001 | 0.08649 | 0.05237 | 0.7613 |  |
| 1024  \*1024 | 7 | 130 | 0.001 | 0.08649 | 0.05237 | 0.89721 | predict imgsz=(1024,1024) |
| 1024  \*1024 | 7 | 150 | 0.001 | 0.08708 | 0.05021 | 0.8841 | 더 학습했지만 점수가 좋아지지 않음. |
| 1024  \*1024 | 7 | 94 | 0.001 | 0.0866 | 0.05299 | 0.882 | pretrained=True(학습된 weight 사용) |
| 1024  \*1024 | 16 | 247 | 0.001 | 0.09415 | 0.0545 | 0.8689 | batch 보다  imagesz가 더 중요 |

위 표와 같이 각 하이퍼 파라미터 조합에서 어떤 성능을 보였는지, 손실이 어떻게 줄어드는지를 확인하고 어떻게 학습하면 성능이 좋아질지 고민하고 발전시켜 나갔다. 가장 큰 성능 변화는 imgsz를 1024로 올렸을 때이다.

먼저 학습을 할 때 제약으로 층이 많고 무거운 모델이라 그런지 시간도 많이 걸렸고 용량으로 인해 배치사이즈나 이미지사이즈를 일정 이상 올리기 힘들었다. 때문에 코랩프로 결제와 컴퓨팅 단위 구입도 여러번 했고 제한된 시간, 용량 내에서 최대한의 결과를 올리려고 노력했다.

**4.3 고찰**

학습을 하면서 알게 된 점은 첫번째로 배치사이즈를 내리면 그만큼 가중치 업데이트를 많이 하기에 learning rate와 배치사이즈를 동시에 높이게 되면 학습이 빨라지게 되고 과적합의 위험도 커지게 된다는 것이다. 논문과 각종 딥러닝 사이트를 찾아본 결과 보통 배치사이즈를 가능한 높게 잡는 것이 성과에 좋다는 것을 봤었다. 하지만 또 너무 높게 잡으면 그만큼 학습이 제대로 되지 않기에 적당한 배치사이즈를 찾아나가야 한다. 최적의 조합에서의 배치사이즈는 7이었으나 이는 배치사이즈를 8로 했을 때 메모리 부족으로 인해 7로 낮춰 학습하였던 것이다.

두번째로 이미지사이즈가 생각보다 성능에 훨씬 중요하다는 것이다. 이미지사이즈를 올리면 성능이 좋아질 것이라 생각은 했지만 그것 보다는 학습을 시킬 때 학습률, 배치사이즈를 통해 최적의 지점까지 학습을 시키는 것이 더 중요하다고 생각했었다. 하지만 이미지사이즈가 작으면 모델이 차의 세세한 모양, 부분까지 파악할 수 없기에 최적의 지점까지 학습을 시켜도 한계가 너무 명확했다. GPU 메모리가 충분해 이미지사이즈를 더 올려서 학습했으면 어땠을까 하는 아쉬움도 있다.

세번째로 시간의 제약 역시 큰 영향을 미치기에 파라미터들을 최대한 효율적으로 제어해줘야 한다는 것이다. 딥러닝 강의를 듣고 실습할 때는 별다른 시간 소요 없이 금방금방 학습이 진행돼서 결과를 알 수 있었다. 이와 같은 기억에 여러 파라미터를 다 적용해보고 최적의 결과를 얻으면 되겠다 생각했다. 하지만 직접 모델을 학습해보니 파라미터 조합 하나에 대해 학습하는 것만 해도 4일 이상씩 걸렸고, 시간의 제약 때문에 최대한 효율적으로 학습을 진행해야됐다.

딥러닝 관련 서적을 읽다가 그런 글귀를 봤던 것 같다. ‘딥러닝에는 완전히 구조화되고 맞는 이론은 없고, 데이터마다 너무나 달라지기 때문에 경험적인 부분, 통찰에 의해 발전시켜 나가는 것이라고’ 팀원 모두가 딥러닝에 대해 처음이었기에 어떻게 어디부터 시작해야 될지, 이후에는 어떻게 모델을 발전시켜야 할지 전혀 감이 잡히지 않았다. 일단 뭐라도 해보고 부딪쳐보면서 나름의 성과물을 얻게 됐다. 프로젝트가 원하는 방향으로 순조롭게 이루어지지는 않았지만 직접 프로젝트에 참여해보고 공부하면서 조금씩 알게 되고 순위가 올라가는 것에 기쁨을 느꼈고 좋은 경험이었던 것 같다.

**5. conclusion**

결론적으로 우리는 yolov8x 모델에서의 최적 hyper parameter 조합으로 batch=7, Lr=0.001, imgsz=1024\*1024, epoch=130으로 정하였고 최종 제출 점수(map0.85)는 0.897이 나왔다. 거의 웬만한 차에 대해서는 정확하게 위치를 탐지했고 분류도 준수하게 했다고 볼 수 있다. 학습데이터는 합성데이터, 테스트데이터는 실제 사진인 것을 생각하면 좋은 결과 같다.

모델의 장점은 실제 승용차들에 대한 위치탐지는 매우 잘한다는 것이다. 또한 라벨 분류도 꽤나 훌륭하게 수행하였다. 단점으로는 학습에 너무 많은 시간이 걸리고 메모리를 많이 차지한다는 것이다.

테스트 이미지를 보면 부여된 라벨이 아닌 다른 차종들이 너무나도 많았다. 학습데이터에 있던 차들이 모두 소형, 중형차들이었기에 학습 라벨에 없어도 승용차들에 대해서는 대부분 탐지가 되었다. 이 모델이 실제에 적용되려면 현실에 있는 더 많은 차종들에 대해 학습이 진행될 필요가 있어 보였다. 분류해야 할 라벨의 개수가 많아지면 그만큼 세밀한 디테일을 보고 판단해야 하기에 imagesz도 더 올려서 학습해야 할 것이다.