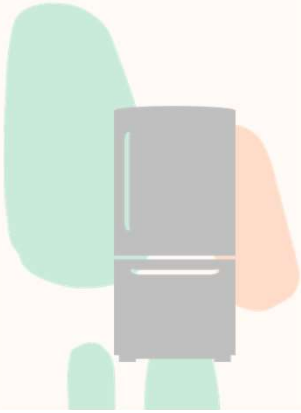
[SNU Bigdata Fintech] 기계학습과 딥러닝

Deep Learning Project

4조

| 박석훈 | 박태준 | 윤성규 | 임동건 |



Contents

- **Introduction**
- **Dataset**
- **Model :** YOLO(v5 - v8)
- **Results**
- **Application**

Introduction

YOLO 하다 GOAL로 간다

주제 : 요리 도우미 (부족한 식재료 확인)

- 모두에게 친근한 소재 - 요리
- 이미지 처리 모델 - 객체 탐지에 대해 알아보는 기회
  - 객체 탐지에 많이 사용한다는 **YOLO**를 이용해 식재료들을 파악한다면
  - 나아가 파악한 식재료로 할 수 있는 요리 또는 내가 하고 싶은 요리에 필요한 재료 들을 바로 확인 할 수 있음

요리 도우미가 필요한 이유

1. 사회 현상의 변화

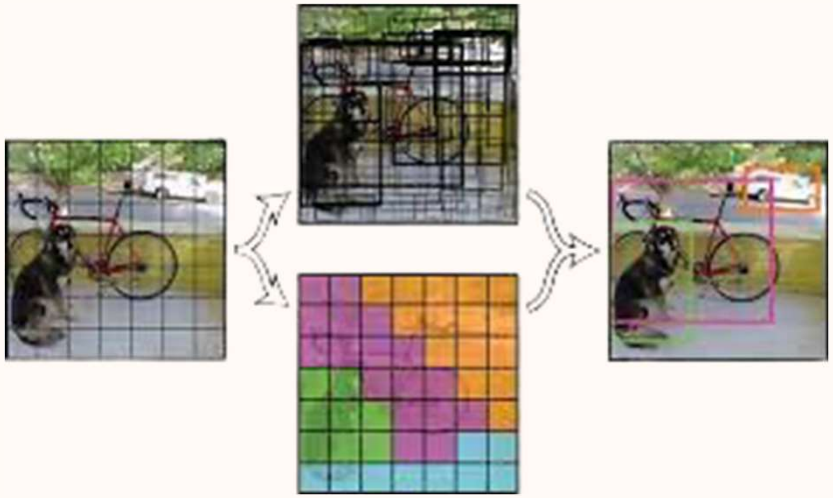
- 1인 가구가 증가하면서 혼자 요리 해먹는 경우 증가
- 요리 유튜버들의 영향력 증가하면서 그들의 레시피를 따라하는 등 사람들의 요리에 대한 관심도 증가

2. 요리 초보자의 어려움

- 가지고 있는 재료를 통해 만들 수 있는 요리 파악이 어려움
- 요리하기 위해 내가 필요한 재료파악이 어려움

요리 도우미를 통해 위의 **2**가지 어려움을 해결 가능

YOLO에 대한 간단한 소개



- **YOLO (You Only Look Once)**
  - 최첨단 실시간 Object Detection 시스템
  - 기존의 모델보다 빠르고 정확한 데이터 처리

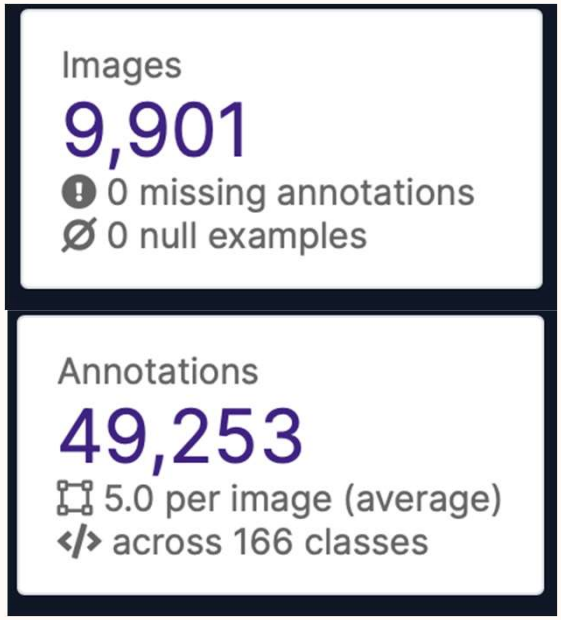
- YOLO는 입력 이미지에 객체의 위치와 객체를 감지해 출력하는 **Model**
- YOLO는 2015년 처음 등장한 후 몇 차례 업데이트 되었고 현재 **v8**까지 출시
- 특히 실시간 의사 결정을 필요로 하는 분야에서 두각을 드러내고 있음

Dataset

Dataset

- **Roboflow Dataset** (9901 images, 49,253 annotations)
  - 대략 150여 종의 많은 label을 가지고 있어
    우리 식탁에 올라오는 대부분의 식재료를 파악 가능

- 추가적으로 많은 사진들을 통해 모델을 잘 학습 시킬 수 있다고 생각
- **Noise** 및 **Rotation** 이미지 데이터 존재
- 단순 식재료 뿐만 아니라 식재료를 보관하고 있는 냉장고 안을 찍은 데이터들도 존재



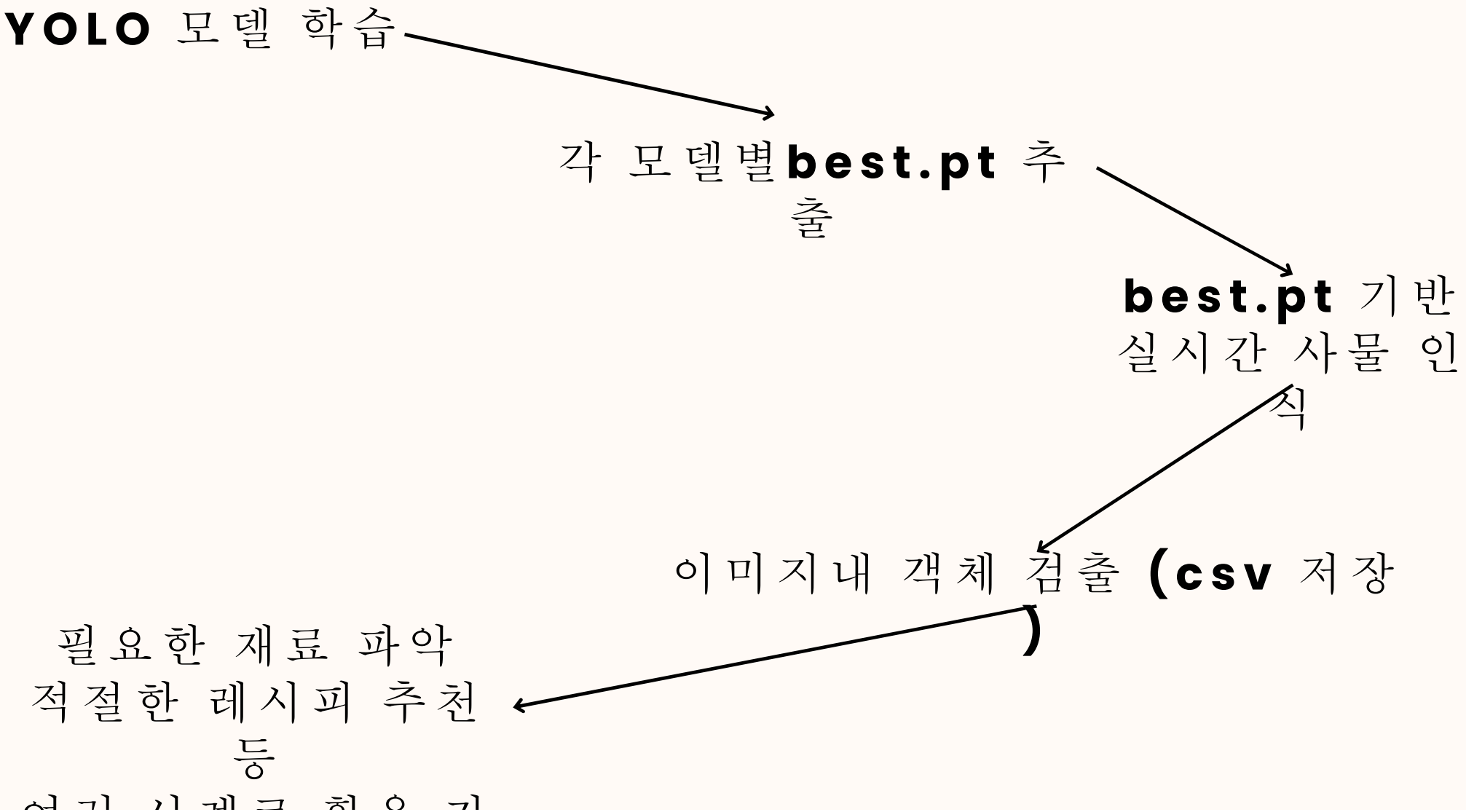
출처 : https://universe.roboflow.com/dsstudy-h0rzy/food-ingredients-image-detection_team4/dataset/1

Dataset

- **150** 여종의 라벨들
  - 고기류 chicken_breast, beef, turkey...
  - 해산물류 shrimp, salmon, crab...
  - 채소류 pepper, Tomato, red_onion...
  - 과일류 Apple, Banana, lemon...
  - 가공식품류 ham, hash_brown, fish_cake...



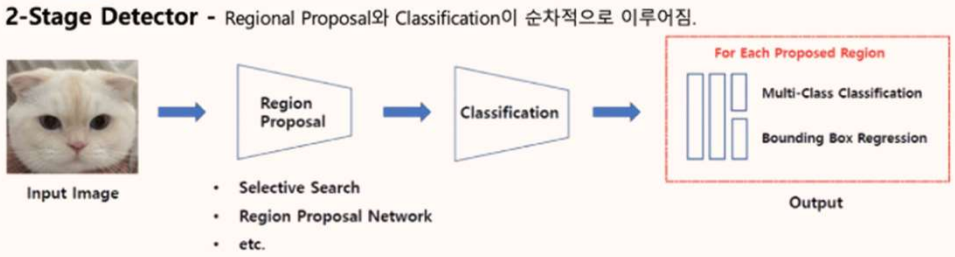



YOLO 모델 학습
각 모델별best.pt 추출
best.pt 기반 실시간 사물 인식
이미지내 객체 검출 (csv 저장)
필요한 재료 파악
적절한 레시피 추천
등

model

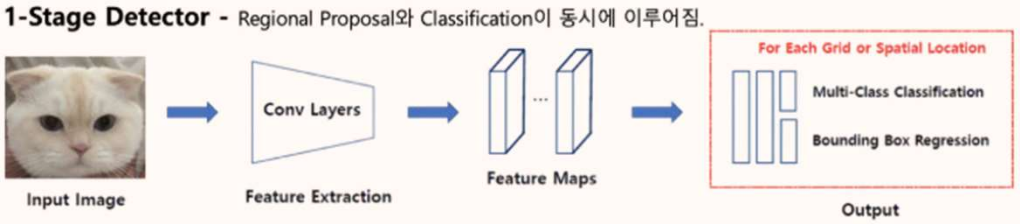
YOLO, One-stage-detector

Two-stage-detector

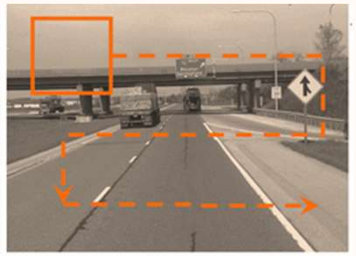


- Regional Propsal + Classification
  순차적으로 진행
- 비교적 느리지만 높은 정확도

One -stage-detector



- Regional Propsal + Classification
  동시에 진행
- 비교적 빠르지만 낮은 정확도



Regional Proposal : 물체가 있을만한 영역을 빠르게 찾아내는 알고리즘

YOLO 모델특징

1. You Only Look Once : '이미지 전체를 단 한번만본다'

- 딥러닝 객체 탐지 모델 중 R-CNN은 일저안 이미지를 여러장 쪼개서 CNN 모델을 통과시킴.
- 이로인해 한장의 이미지에서 객체 탐지를 수행해도 수천장의 이미지를 모델에 통과시키는 형태
- 반면, YOLO는 이미지 전체를 말 그대로 한번만 보는 구조라 볼 수 있음

2. Unified : 통합된 모델 사용

- 다른 객체 탐지 모델은 다양한 전처리 모델과 인공 신경망을 겹합하지만, YOLO는 단 하나의 인공 신경망에서

  전부 처리하여, 다른 모델에 비해 간단

3. Real -time Object Detection : 실시간 객체 탐지

- YOLO가 유명해진 주 이유로써, 실시간으로 여러장의 이미지를 탐지할 수 있어 영상을 스트리밍 하면서 동시에 화면상의 물체를 부드럽게 실시간 구분이 가능

YOLO 모델 이해하기

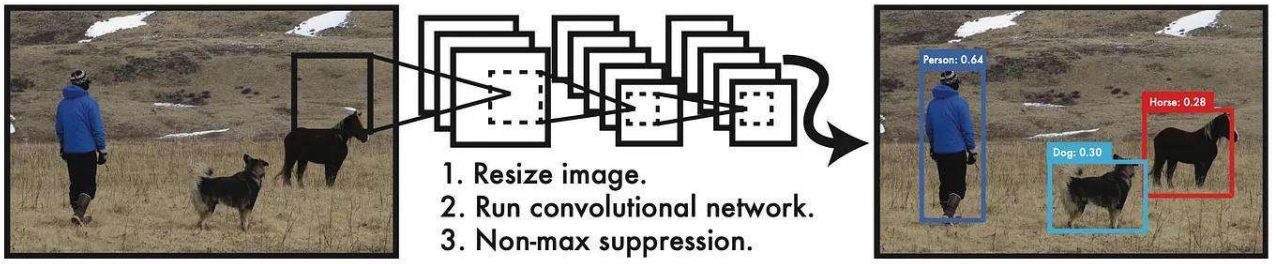


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

- R-CNN과 달리 합성곱 신경망을 단 한번 통과시키며, 신경망의 결과로 각 객체의 바운딩 박스와 해당 객체가 무엇인지 분류확률을 출력함
- 최종적으로 확률값을 Non -max suppression을 통해 리전(위치)를 결정

<Non - max suppression(비-최대 억제) : object detector가 예측한 bounding box 중에서 정확한 bounding box를 선택하도록 하는 기법>

YOLO 모델 이해하기

이미지를 7*7(S*S) 그리드 셀로 나우며, 나눈 셀 중 물체의 중앙과 가까운 셀이 객체를 탐지하는 역할

각 셀은 바운딩 박스와 B와 분류한 클래스 확률인 C를 예측

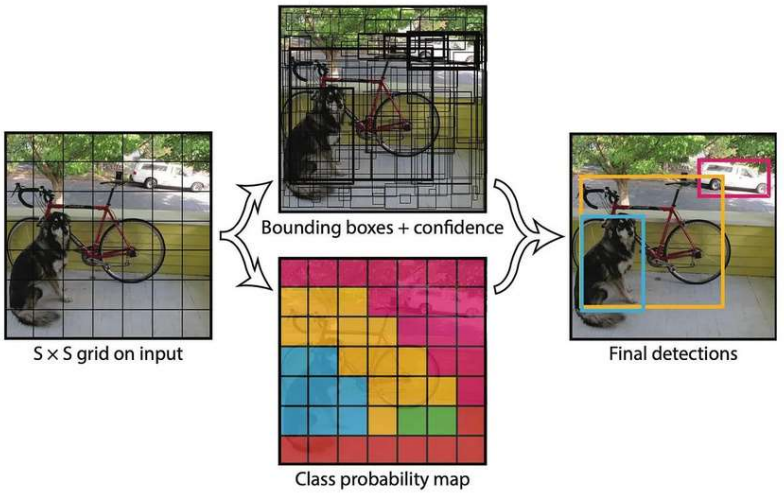


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

- 바운딩 박스(B) : 바운딩 박스 B는 X,Y좌표, 가로, 세로크기 및 Confidence Score 수치정보
- Score : B가 물체를 영역으로 잡고있는지와 클래스를 잘 예측하였는지를 의미
- (Score = Pr(object)*IOU, Pr(object)는 바운딩 박스안에 물체가 존재할 확률이며, 만약 바운딩 박스가 배경만을 영역으로 잡고 있다면 Pr(Object)값이 0이므로 score는 0이 됩니다)
- 3. IOU :Intersection over union의 약자, 학습 데이터의 바운딩 박스와 예측한 바운딩 박스가 일치하는 정도를 나타냄
- 4.클래스 확률 C : 그리드 셀 안에 있는 그림의 분류 확률을 의미함 Pr(Class_i | Object)

$$IoU = \frac{area(B_{gt} \cap B_p)}{area(B_{gt} \cup B_p)} =$$

네트워크 디자인

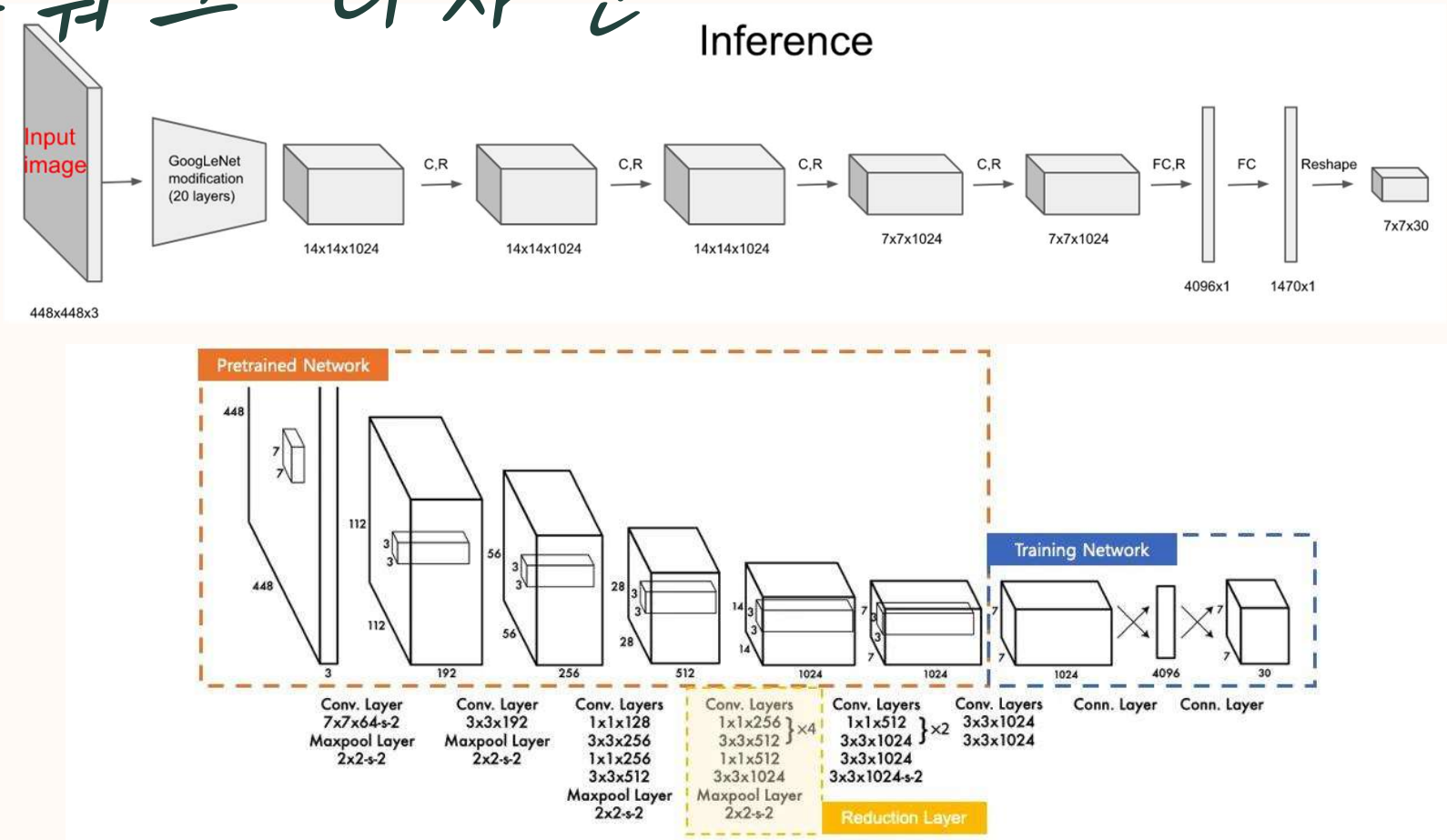


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

네트워크 디자인

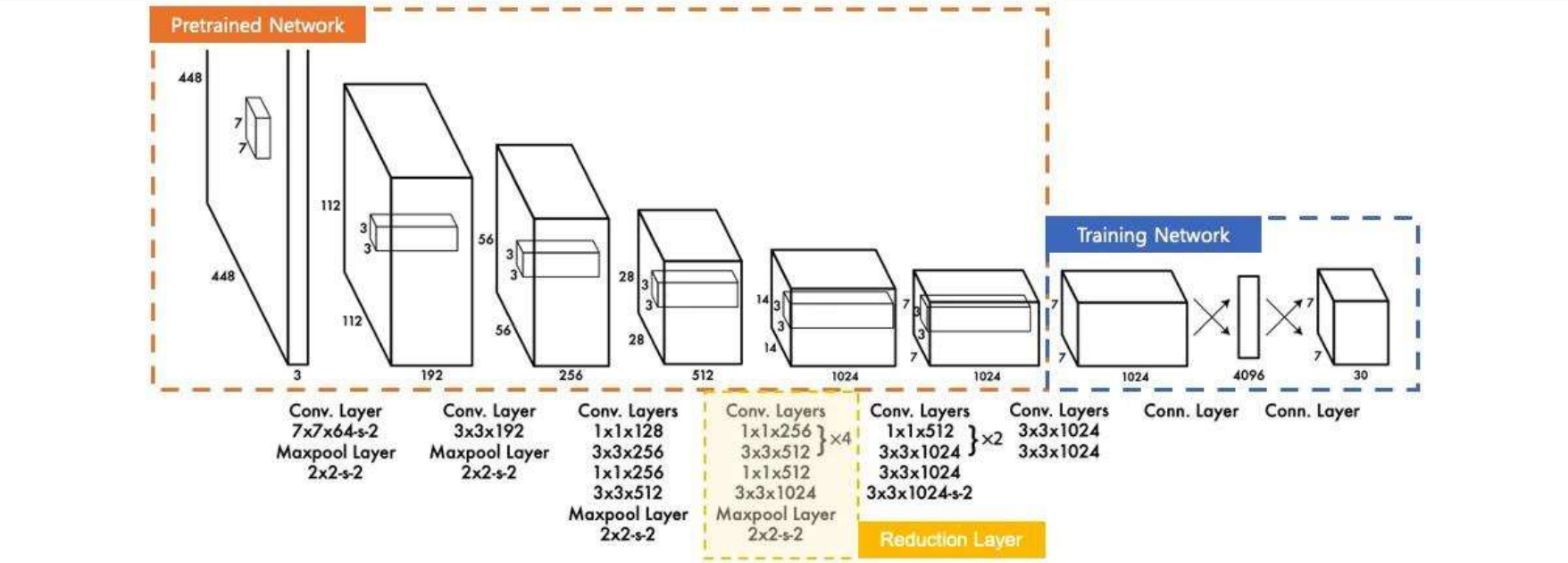
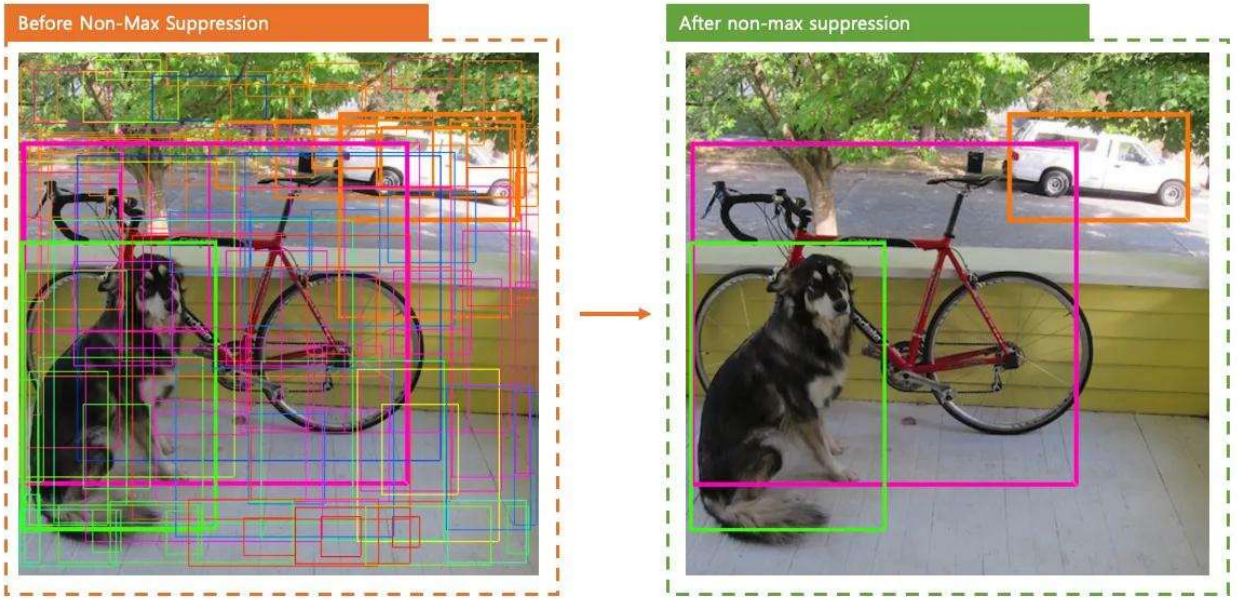


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

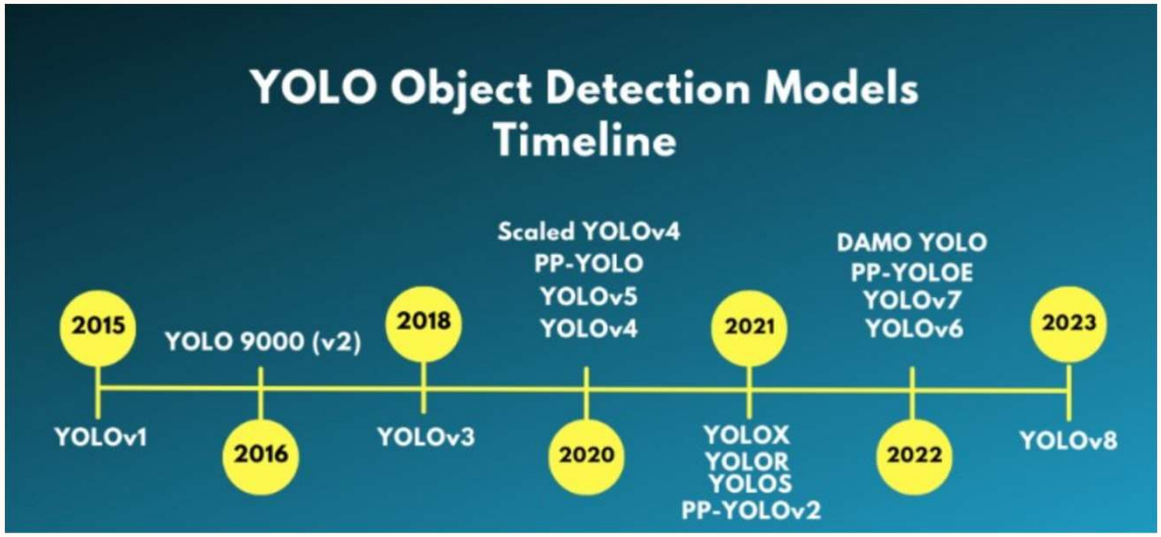
Loss Function

$$
\begin{aligned}
&\textcircled{1}\,\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \textcircled{2}\,\mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&\textcircled{5}\; + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \textcircled{3}\,\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \textcircled{4}\,\mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&\textcircled{6}\; + \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \qquad (3)
\end{aligned}
$$



YOLO의 역사

YOLO v1 : 기존 모델들 보다 성능과 속도 향상
YOLO v2 : v1에서 Bounding Box 도입 성능과 속도 향상
YOLO v3 : v2에서 Skip connection 개념을 도입해 성능 향상
YOLO v4 : v3에서 작은 객체 검출을 위해 CSPNet 도입

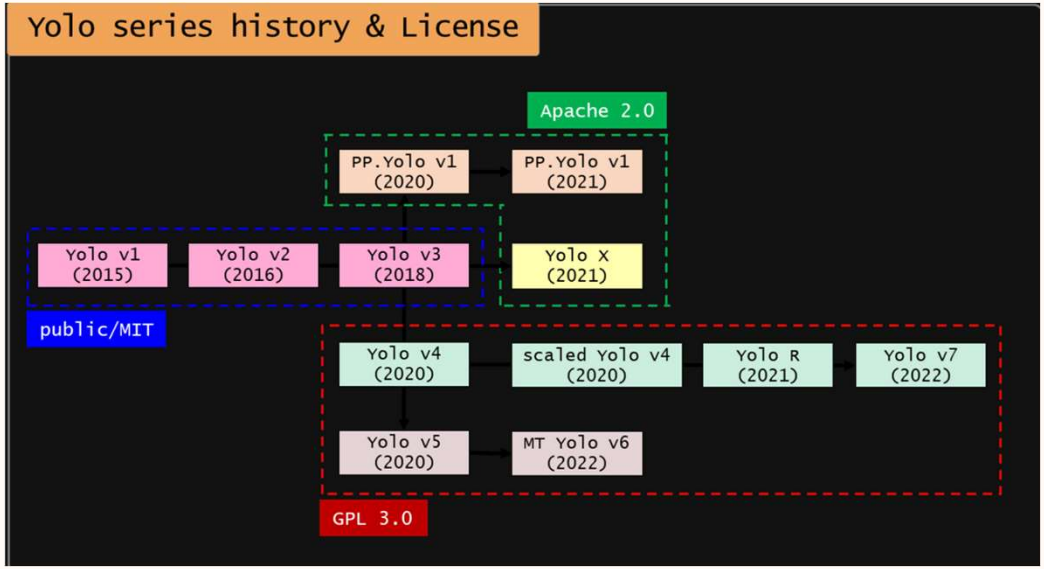


YOLO의 역사

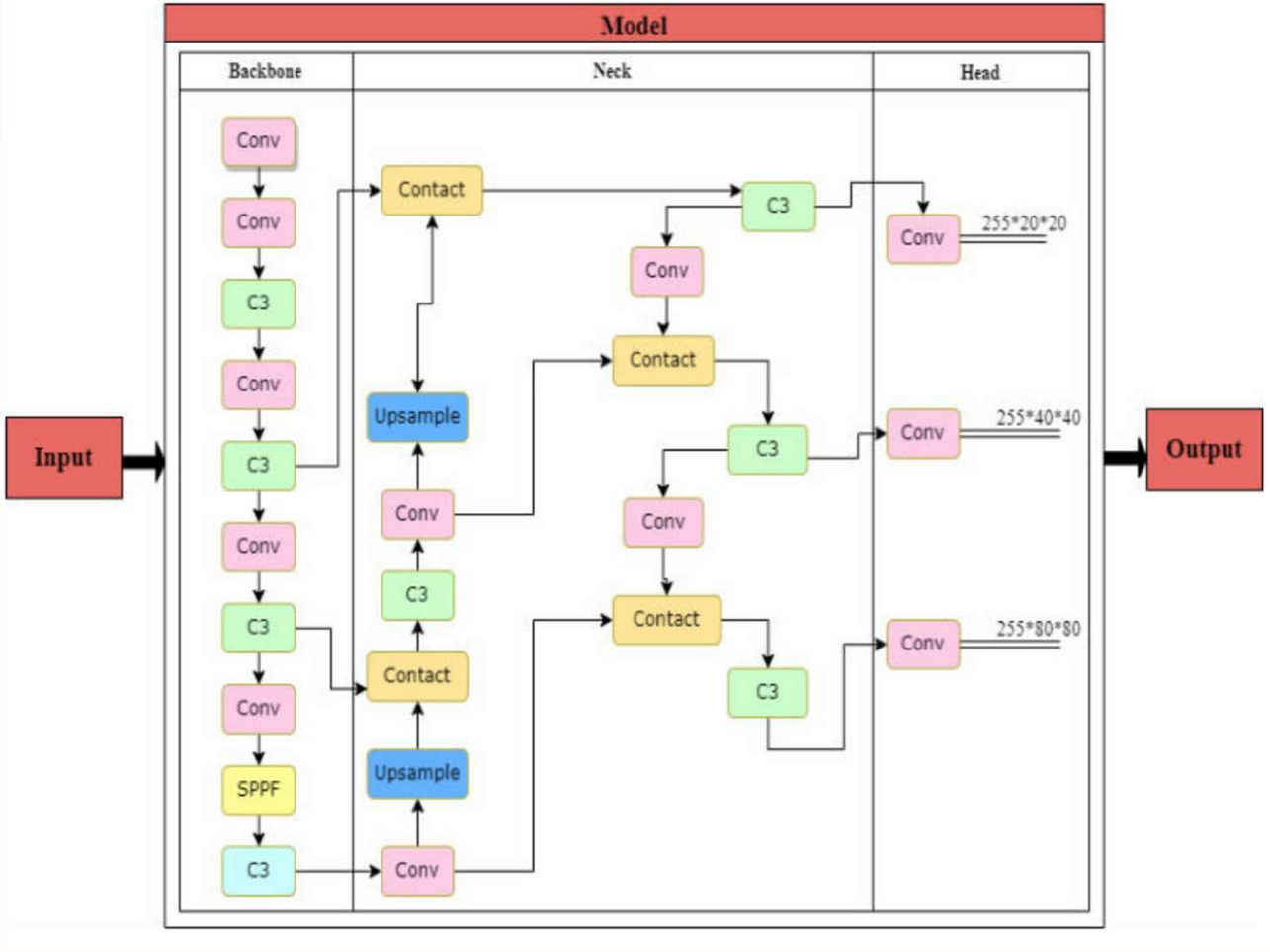
YOLO v5 : v4의 모델보다 SPP 대신 SPPF를 사용하여 속도 향상
YOLO v6 : v5 모델 보다 더 깊게 생성해 정확도 향상
YOLO v7 : Soft Label 도입해 성능 향상
YOLO v8 : 새로운 저장소를 출시하여 객체 감지, 통합 프레임워크 구축

- **YOLO v1 (2016)**: 실시간 객체 검출을 위한 딥러닝 기반의 네트워크
- **YOLO v2 (2017)**: v1에서 성능 개선 및 속도 향상
- **YOLO v3 (2018)**: 네트워크 구조와 학습 방법을 개선하여 Object Detection 정확도와 속도 개선
- **YOLO v4 (2020. 04)**: SPP와 AN 기술을 적용하여 Object Detection 정확도와 속도 개선
- **YOLO v5 (2020. 06)**: 전작보다 정확도 10% 이상 향상, 모델 크기 축소
- **YOLO v6 (2022. 07)**: 훈련 과정의 최적화, Trainable bag-of-freebies 제안
- **YOLO v7 (2022. 09)** 알고리즘의 효율성 향상, 시스템 탑재를 위한 Quantization과 Distillation 방식 도입
- **YOLO v8 (2023. 01)**: 새로운 저장소를 출시하여 객체 감지, 인스턴스 세분화 및 이미지 분류 모델 Train을 위한 통합 프레임 워크로 구축

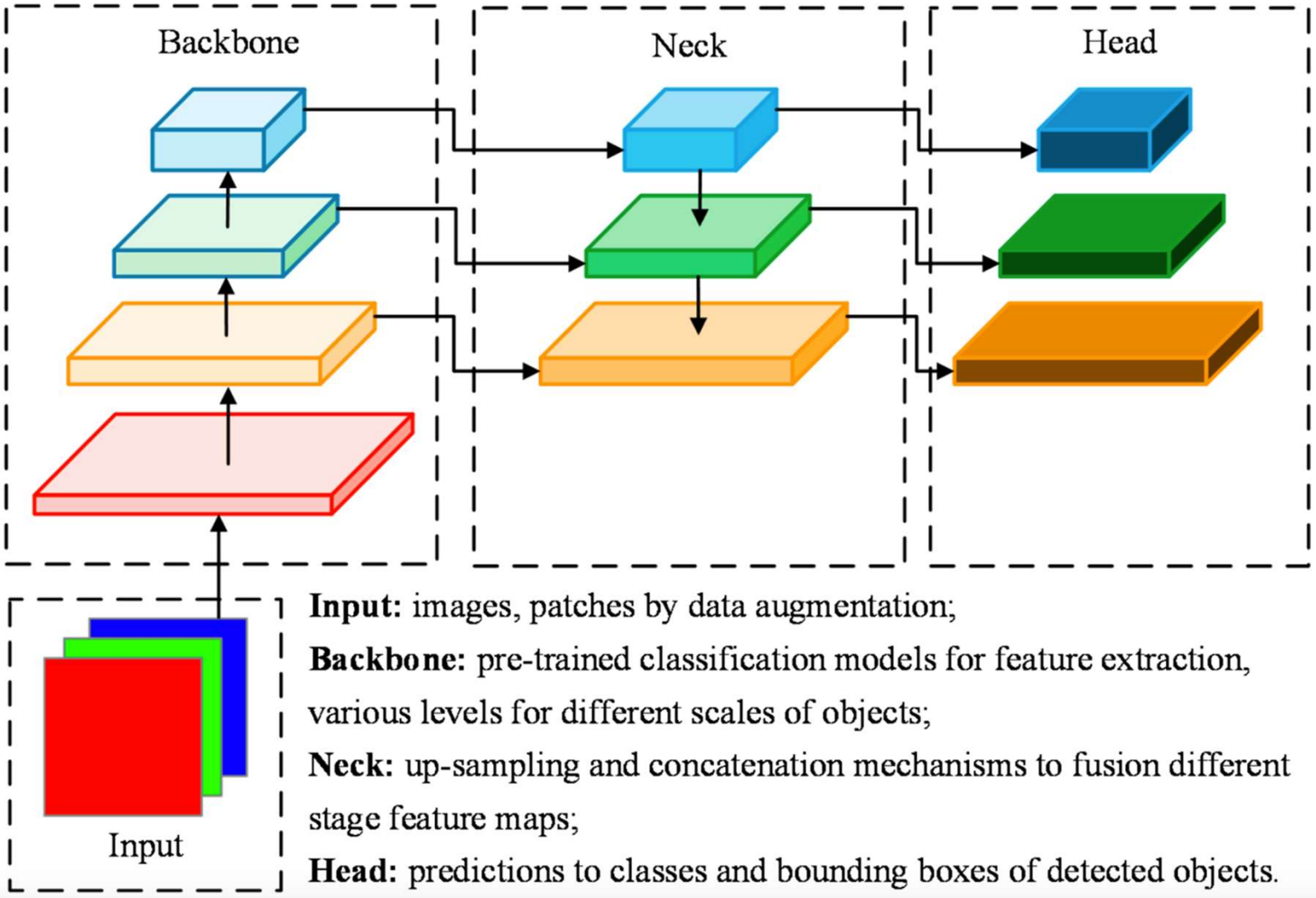
흔히 YOLO 버전을 비교할 때는 v5와 최신 버전인 v8 중에서 선택합니다. YOLOv5는 사용이 편리하며, YOLOv8은 더 빠르고 정확하다는 장점이 있습니다. 궁극적으로 사용할 모델을 결정하는 것은 애플리케이션의 요구 사항에 따라 달라지지만, 대체적으로 실시간 Object Detection 작업이 필요할 경우 YOLOv8을 선택하는 경향이 있습니다.



YOLO V5s 아키텍쳐




```
%%writetemplate
/content/yolov5/models/custom_yolov5s.yaml


# Parameters
nc: {num_classes} # number of classes
depth_multiple: 0.33 # model depth multiple
width_multiple: 0.50 # layer channel multiple
anchors:
 - [10,13, 16,30, 33,23] # P3/8
 - [30,61, 62,45, 59,119] # P4/16
 - [116,90, 156,198, 373,326] # P5/32

# YOLOv5 v6.0 backbone
backbone:
 # [from, number, module, args]
 [[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
 [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
 [-1, 3, C3, [128]],
 [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
 [-1, 6, C3, [256]],
 [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
 [-1, 9, C3, [512]],
 [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
 [-1, 3, C3, [1024]],
 [-1, 1, SPPF, [1024, 5]], # 9
 ]

# YOLOv5 v6.0 head
head:
 [[-1, 1, Conv, [512, 1, 1]],
 [-1, 1, nn.Upsample, [None, 2, 'nearest']],
 [[-1, 6], 1, Concat, [1]], # cat backbone P4
 [-1, 3, C3, [512, False]], # 13

 [-1, 1, Conv, [256, 1, 1]],
 [-1, 1, nn.Upsample, [None, 2, 'nearest']],
 [[-1, 4], 1, Concat, [1]], # cat backbone P3
 [-1, 3, C3, [256, False]], # 17 (P3/8-small)

 [-1, 1, Conv, [256, 3, 2]],
 [[-1, 14], 1, Concat, [1]], # cat head P4
 [-1, 3, C3, [512, False]], # 20 (P4/16-medium)

 [-1, 1, Conv, [512, 3, 2]],
 [[-1, 10], 1, Concat, [1]], # cat head P5
 [-1, 3, C3, [1024, False]], # 23 (P5/32-large)

 [[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
 ]
```


Backbone
Neck
Head
Input
Input: images, patches by data augmentation;
Backbone: pre-trained classification models for feature extraction, various levels for different scales of objects;
Neck: up-sampling and concatenation mechanisms to fusion different stage feature maps;
Head: predictions to classes and bounding boxes of detected objects.

Small

YOLOv5s

14 MB_{FP16}
2.2 ms_{V100}
36.8 mAP_{COCO}

Medium

YOLOv5m

41 MB_{FP16}
2.9 ms_{V100}
44.5 mAP_{COCO}

Large

YOLOv5l

90 MB_{FP16}
3.8 ms_{V100}
48.1 mAP_{COCO}

XLarge

YOLOv5x

168 MB_{FP16}
6.0 ms_{V100}
50.1 mAP_{COCO}

모델 비교

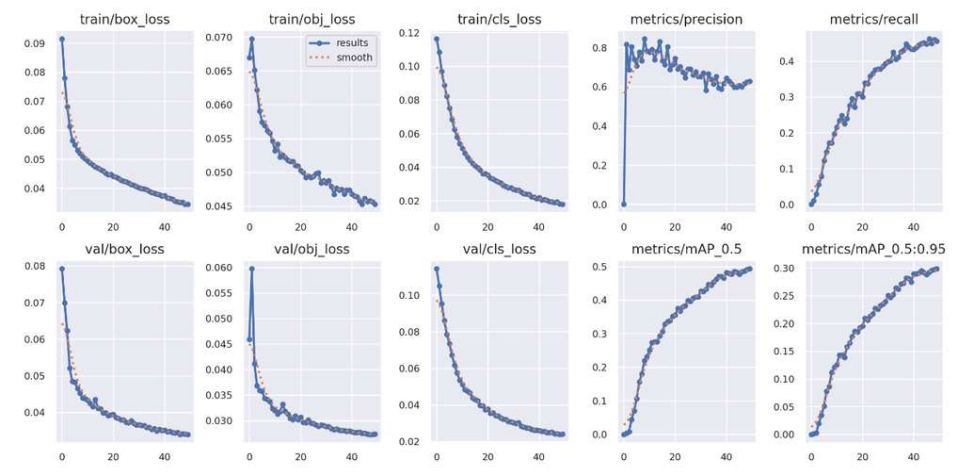
우리가 사용한 세부 모델 :

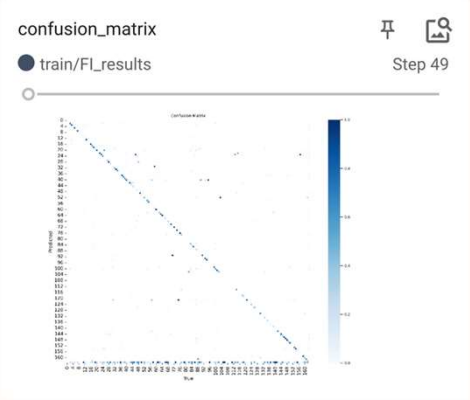
v5 : custom_yolov5s
v6 : yolov6n
v7 : YOLOv7
v8 : 'yolov8n

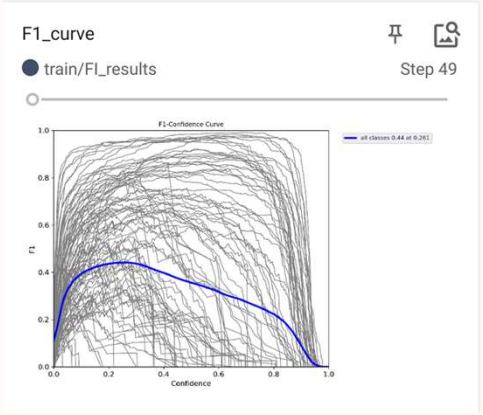
epochs=50으로
train

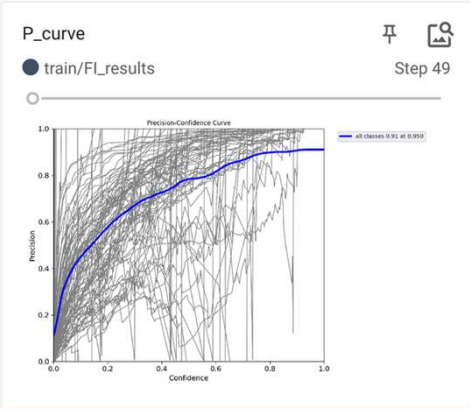
Results

Results : yolov5



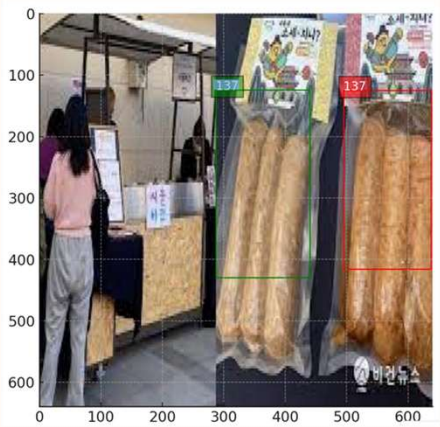






Results : yolovb

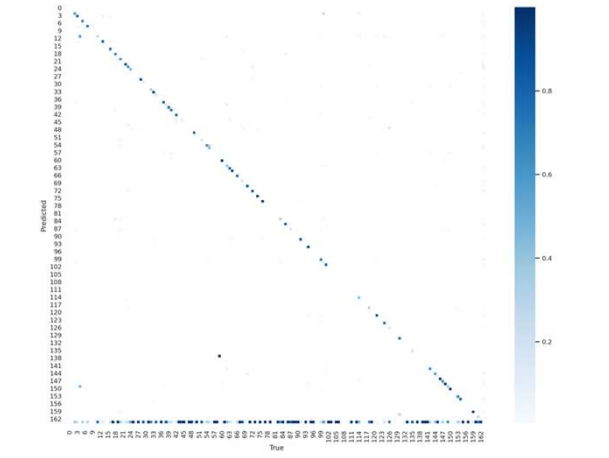
```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.261
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.428
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.282
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.041
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.273
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.242
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.298
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.469
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.504
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.050
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.451
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.521
Results saved to runs/train/exp
Epoch: 38 | mAP@0.5: 0.42836867089299857 | mAP@0.50:0.95: 0.2609907069455362

     Epoch        lr  iou_loss  dfl_loss  cls_loss
     39/49   0.002883    0.4816         0      1.03: 100% 200/200 [00:43<00:00,  4.57it/s]

     Epoch        lr  iou_loss  dfl_loss  cls_loss
     40/49   0.002472    0.4781         0     1.017: 100% 200/200 [00:43<00:00,  4.60it/s]

     Epoch        lr  iou_loss  dfl_loss  cls_loss
     41/49   0.002091    0.4758         0     1.018: 100% 200/200 [00:43<00:00,  4.58it/s]
```

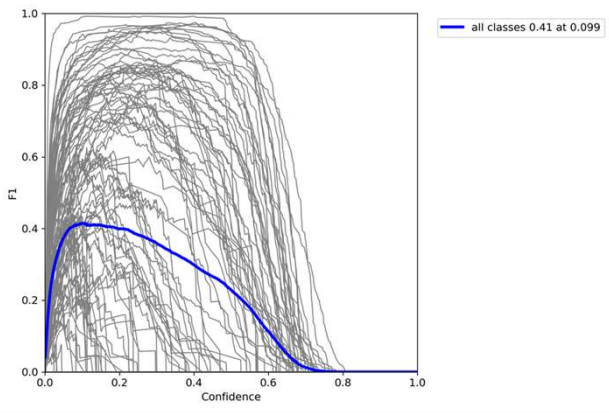


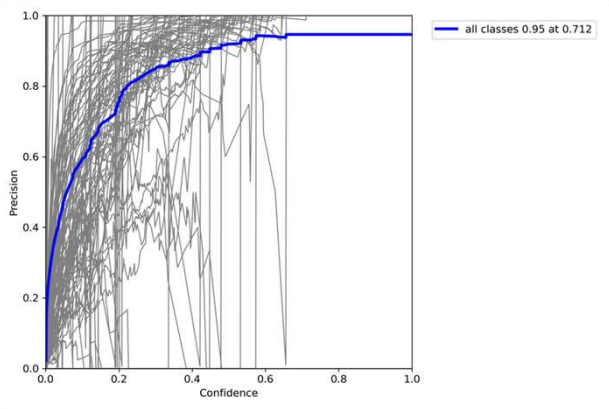
137:

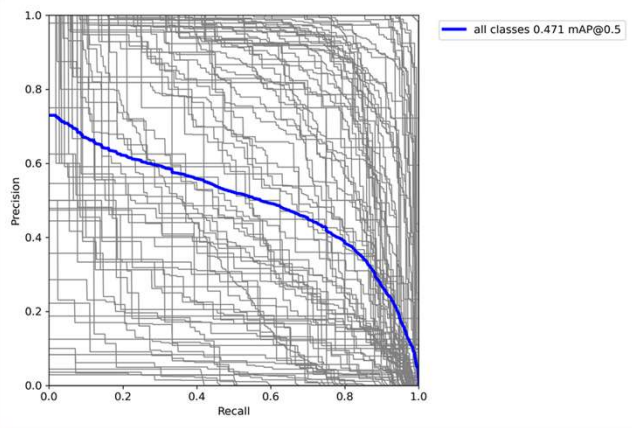
sausage

Results : yolov7

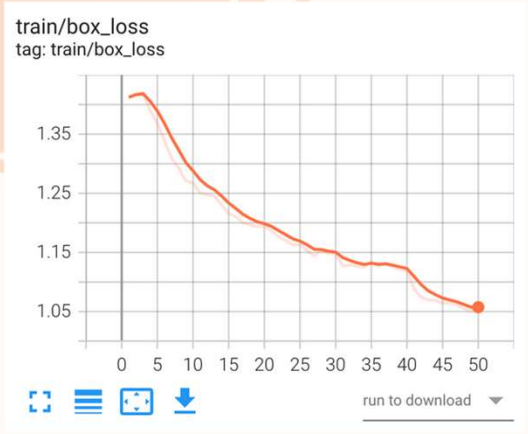


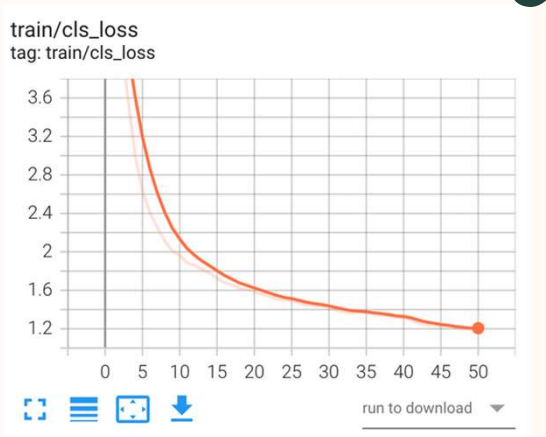


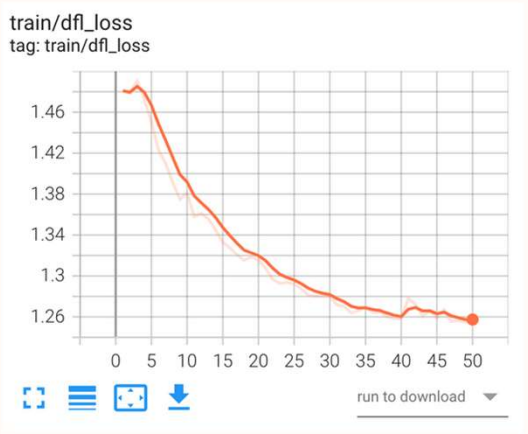


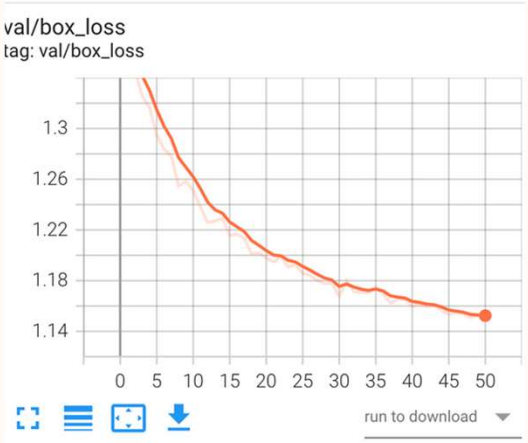


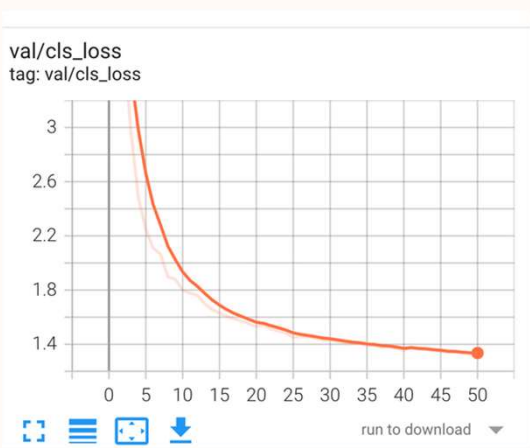
Results : yolov8



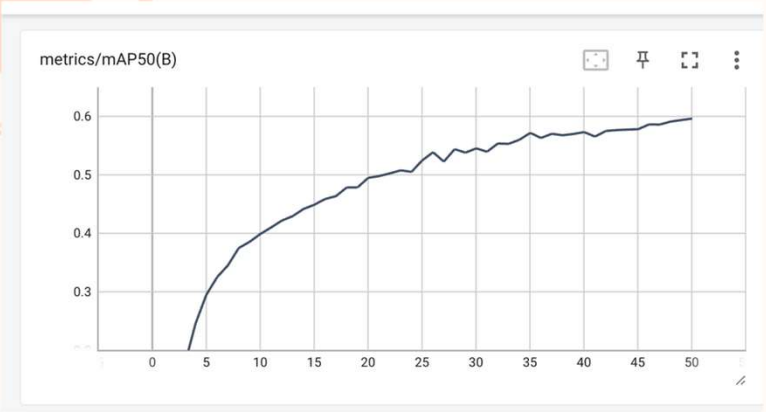


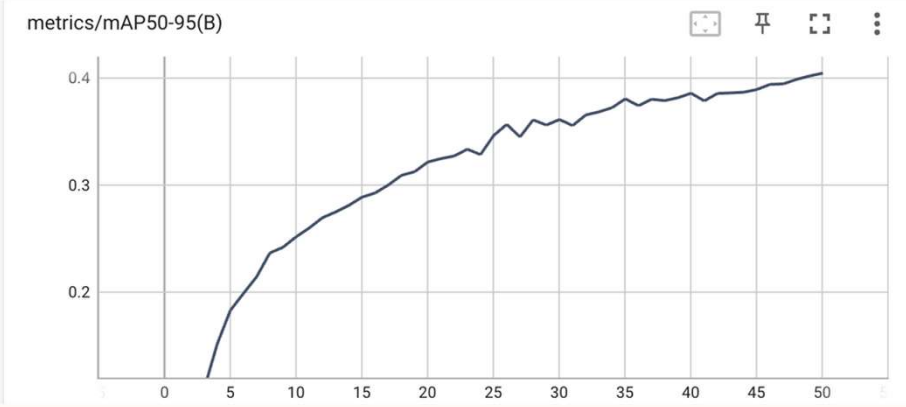


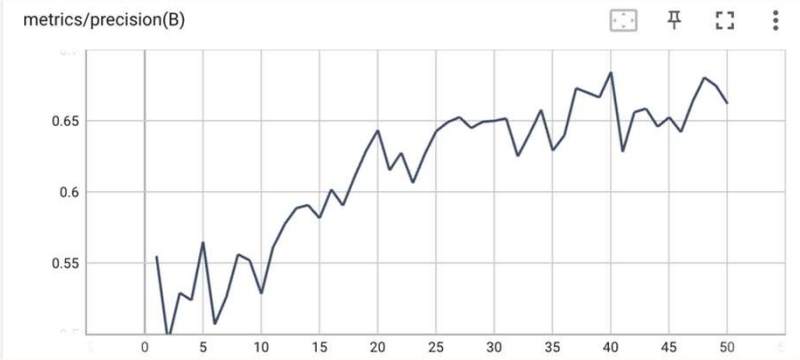


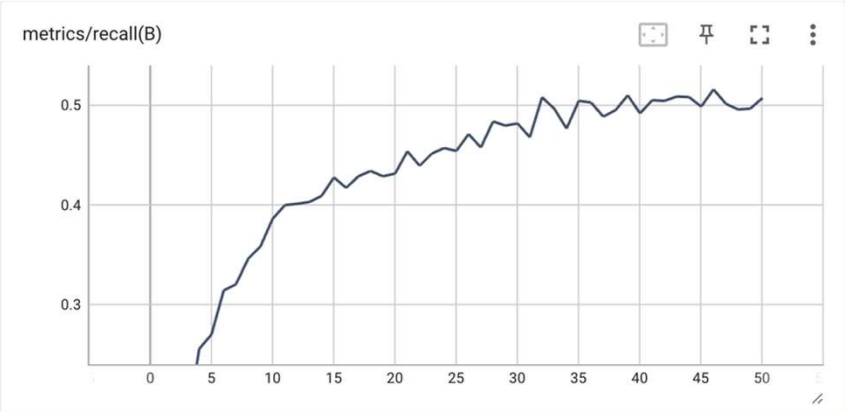


Results : yolov8







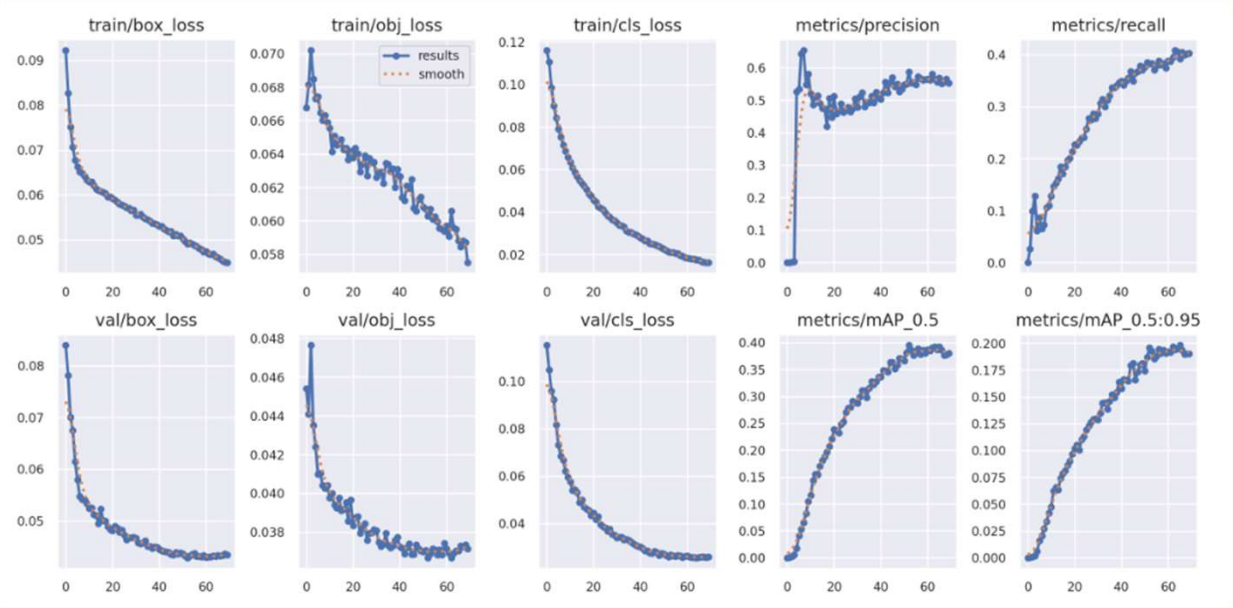


Results : 전체 비교

yolov5 | yolov6 | yolov7 | yolov8

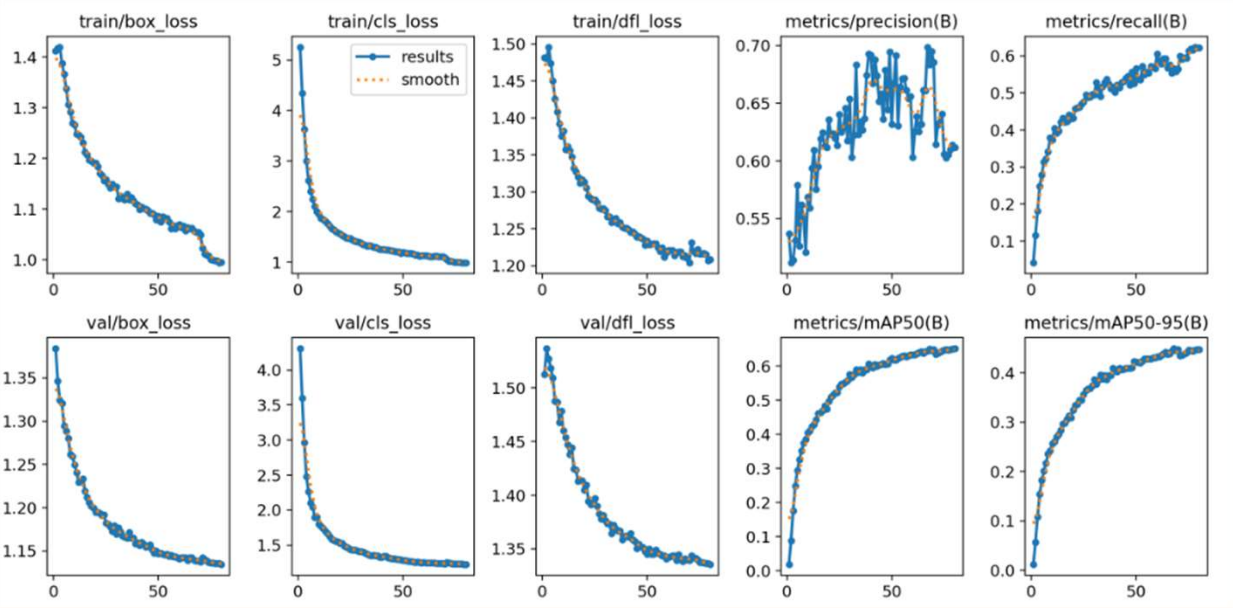
Results(추가)

- yolov5s -> yolov5m
- train data : data augmentation 적용
- epoch 50 -> 70



Results(추가)

- yolov8n model
- train data : data augmentation 적용
- epoch 50 -> 80



Application

Application

요리에 부족한 식재료 확인
원하는 요리 입력 시 가지고 있는 식재료 기반하여 부족한 식재료 확인

→ 마트 장바구니 추가

냉장고 내부 식재료 확인
냉장고를 열어보지 않고도 내부 식재료 확인

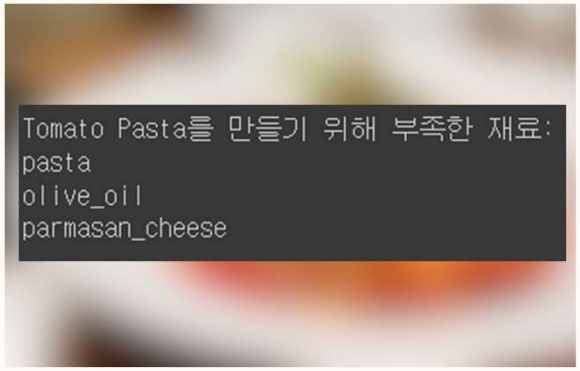
→ 현재 보관 위치 및 상함 여부 확인

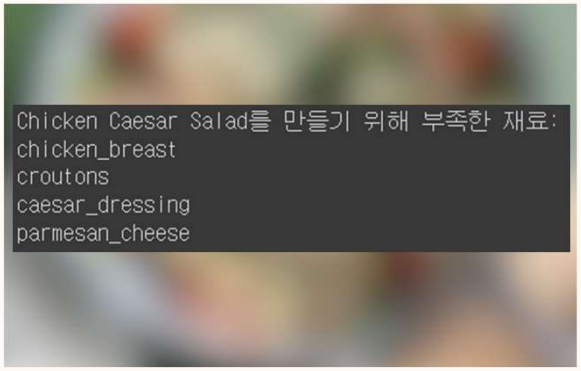
식재료 기반 레시피 추천
가지고 있는 식재료 기반 조리가능한 음식 추천

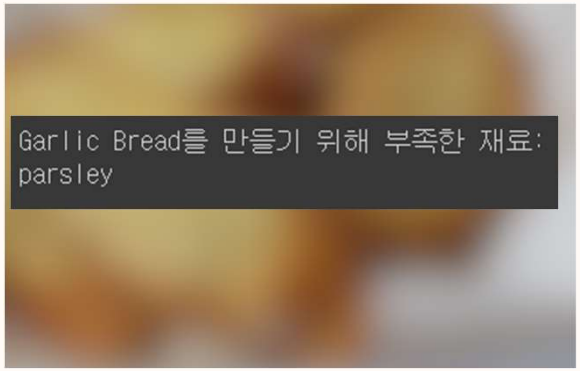
→ 레시피 및 칼로리 확인

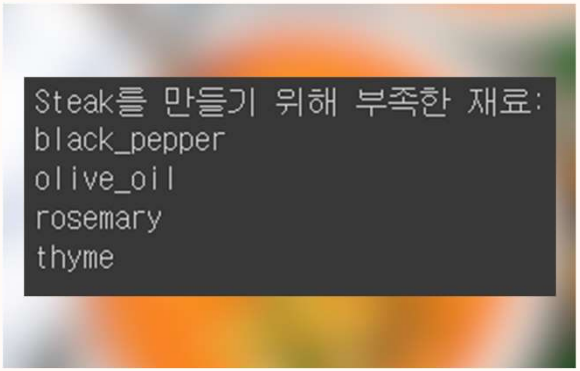
Application

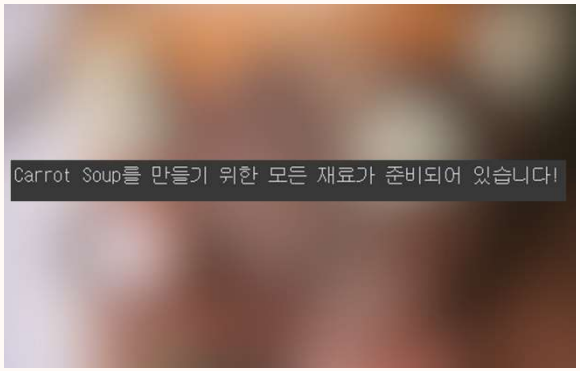
Application











Application

Application

Thank you

Special Thanks to Hyun-Ji Choi