Object Detection

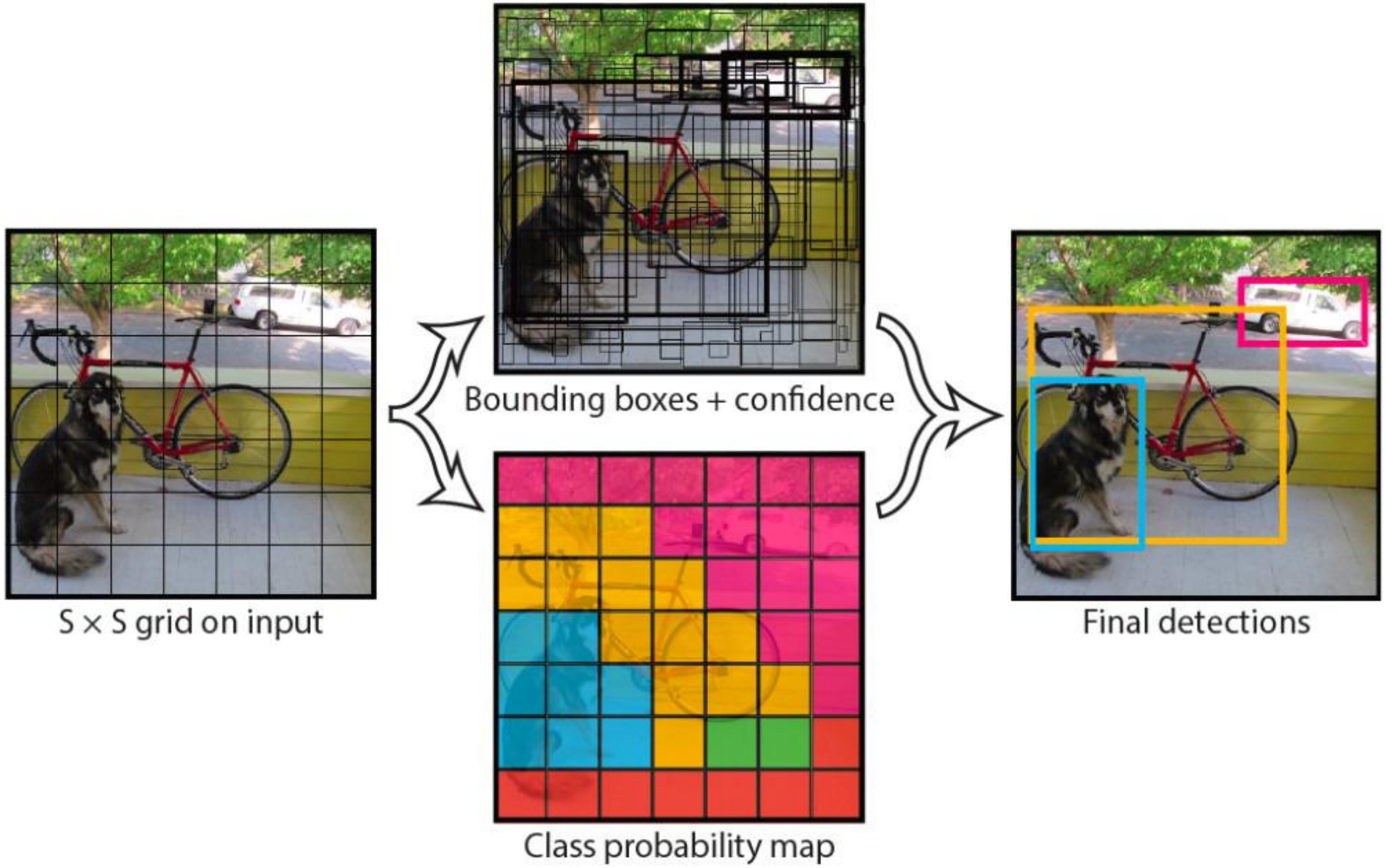
강규태

Contents

1. Introduction
2. Preprocess
3. Model Architecture
4. Loss & Training
5. Test, NMS
6. Result Analysis
7. Future Work

1. Introduction

1 YOLO



이미지를 S x S grid cell**로 나누고**

S x S x B개의 bounding box **만드는**

Object detection

최종 output**은**

S x S x (5 x B + C)

(5: x, y, w, h, c)

x : grid cell 내의 x 위치 (0~1)

y: grid cell 내의 y 위치 (0~1)

w: 전체 이미지 대비 width (0~1)

h: 전체 이미지 대비 height (0~1)

C: 이미지 내 object가 있을 거라고 확신하는 정도

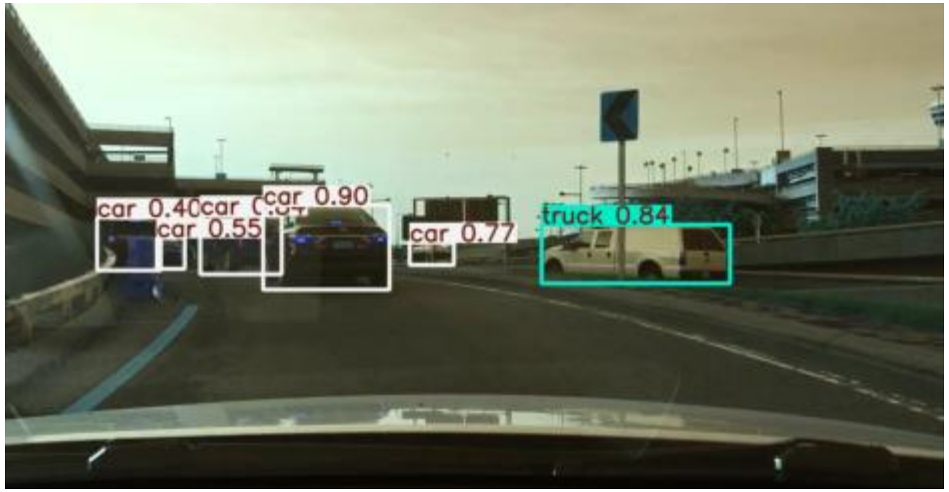
2. Preprocess

2 Preprocessing

[raw data]

BDD100K Only

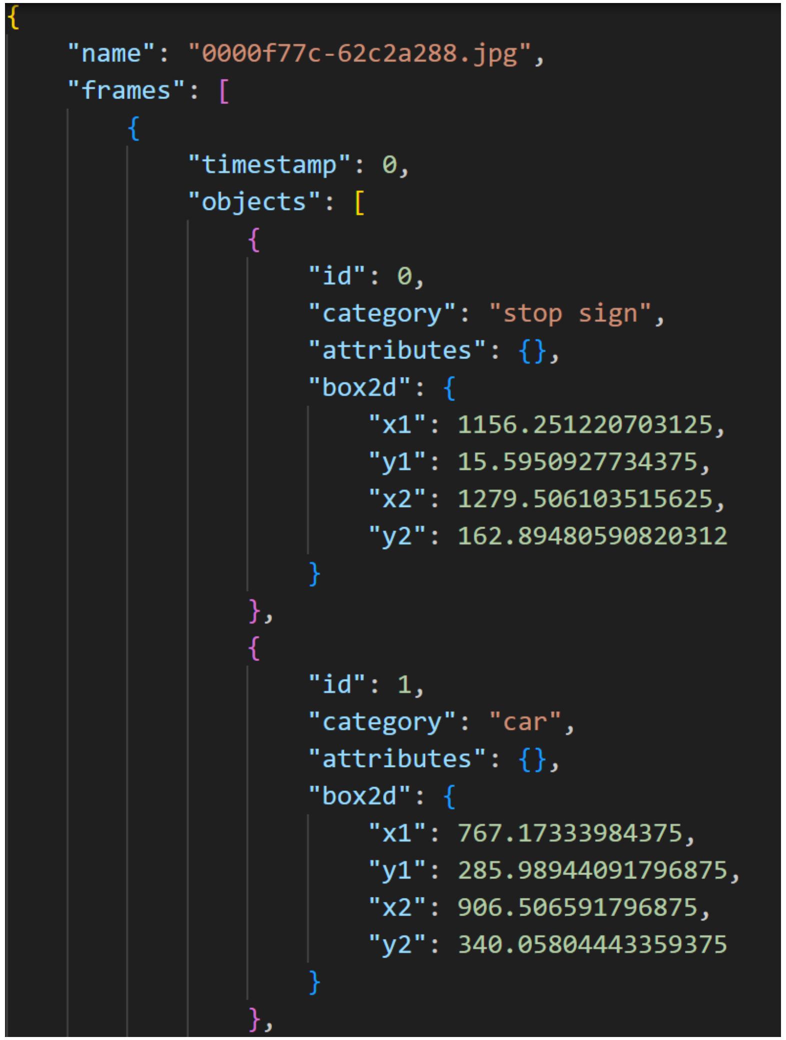
[custom data]



BDD100K + Yolo_v8

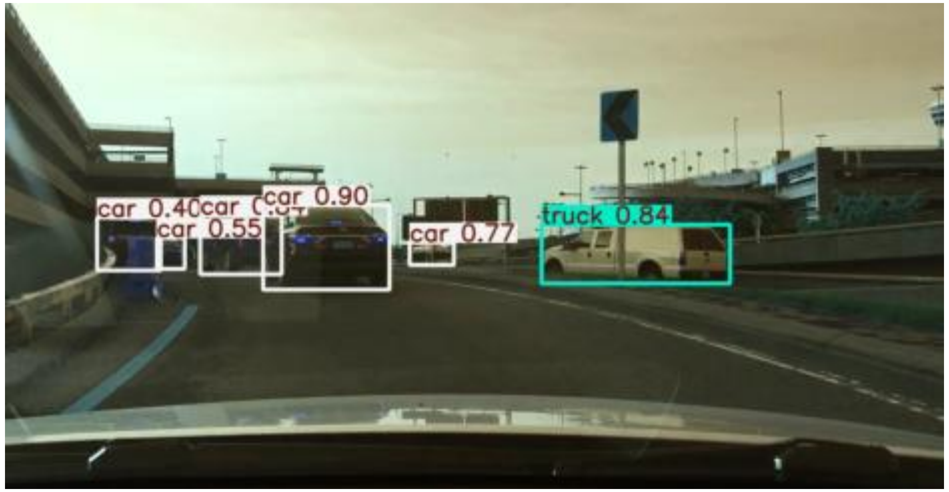
Berkeley DeepDrive

2 Preprocessing



```
{
    "name": "0000f77c-62c2a288.jpg",
    "frames": [
        {
            "timestamp": 0,
            "objects": [
                {
                    "id": 0,
                    "category": "stop sign",
                    "attributes": {},
                    "box2d": {
                        "x1": 1156.251220703125,
                        "y1": 15.5950927734375,
                        "x2": 1279.506103515625,
                        "y2": 162.89480590820312
                    }
                },
                {
                    "id": 1,
                    "category": "car",
                    "attributes": {},
                    "box2d": {
                        "x1": 767.17333984375,
                        "y1": 285.98944091796875,
                        "x2": 906.506591796875,
                        "y2": 340.05804443359375
                    }
                },
```

[custom data]



BDD100K + Yolo_v8

2 Preprocessing - Augmentation

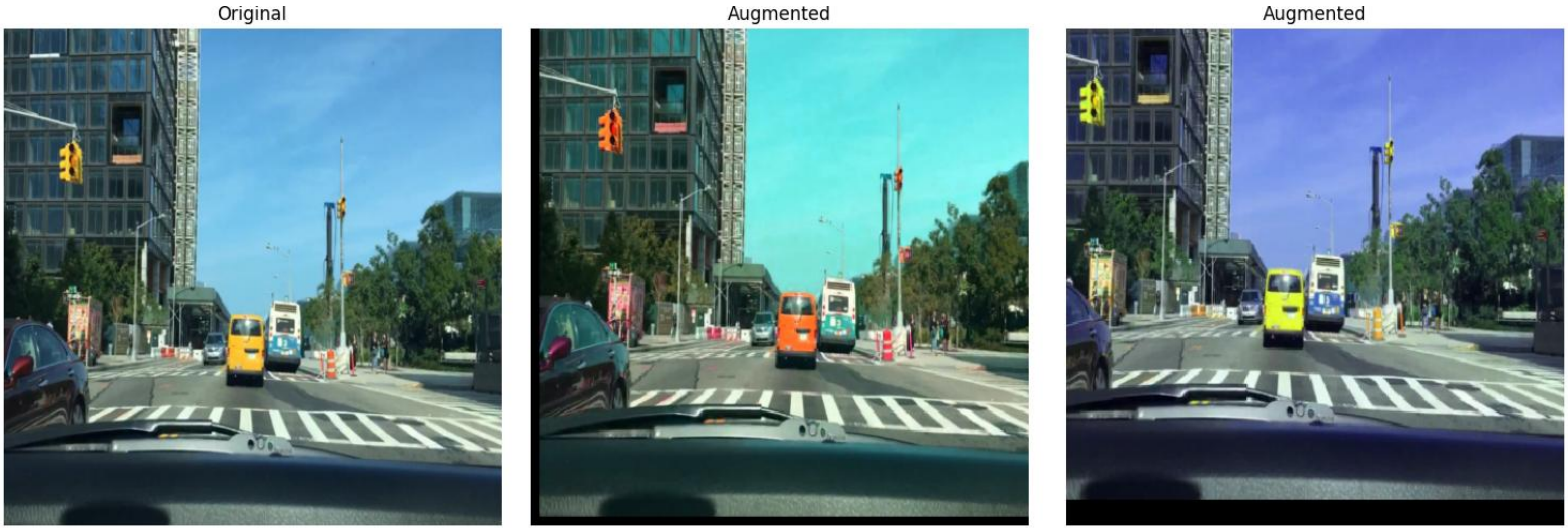
1. Random Translation (평행이동)

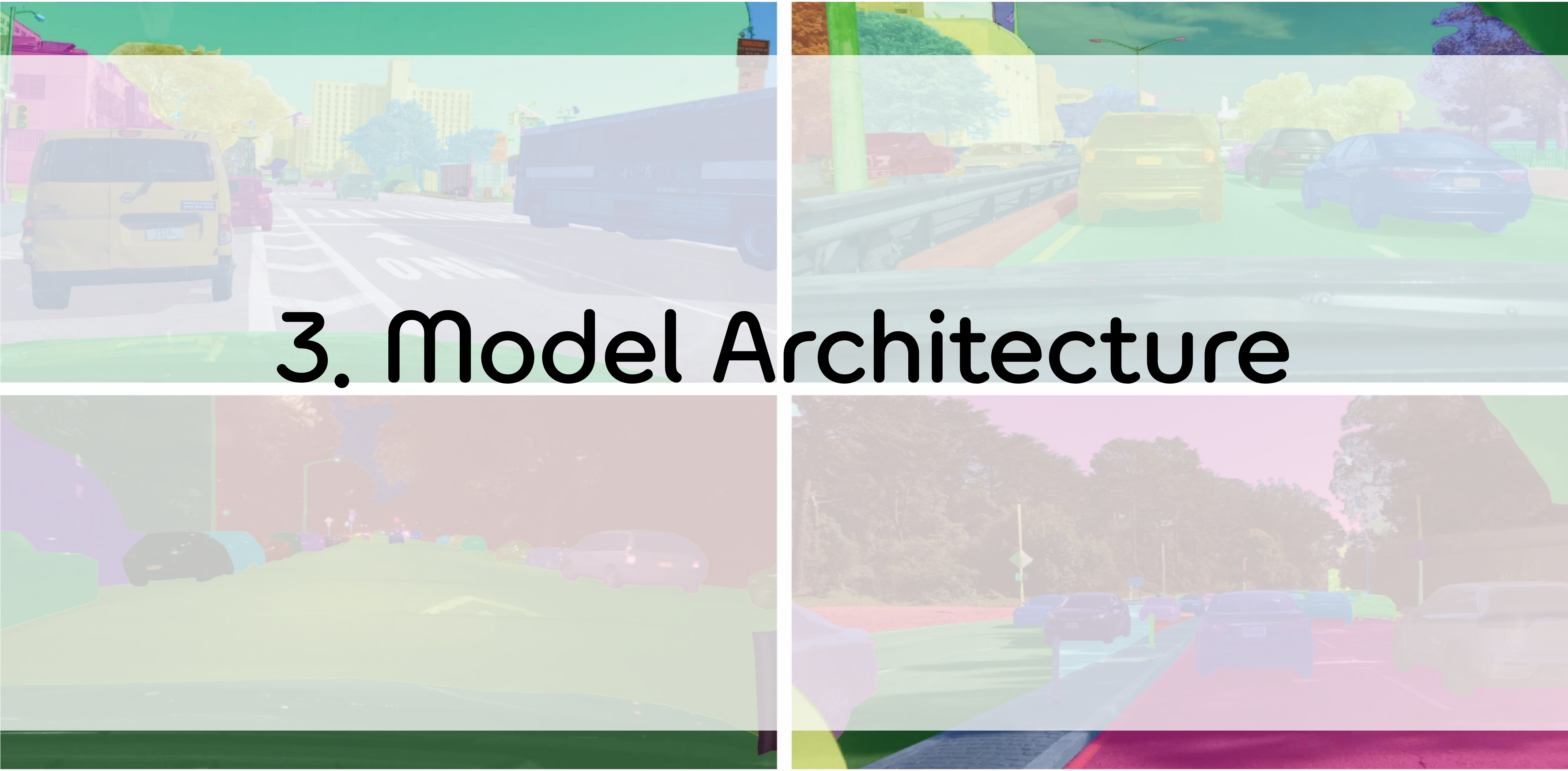
2. Random Scaling (확대 / 축소)

3. Random Hue Shift (색조 변화)

4. Random Saturation (채도 변화)

2 Preprocessing – Augmentation





3. Model Architecture

3 Model Architecture

- config.py
- data2.py
- eval2.py
- loss2.py
- model2.py
- train2.py
- utils2.py

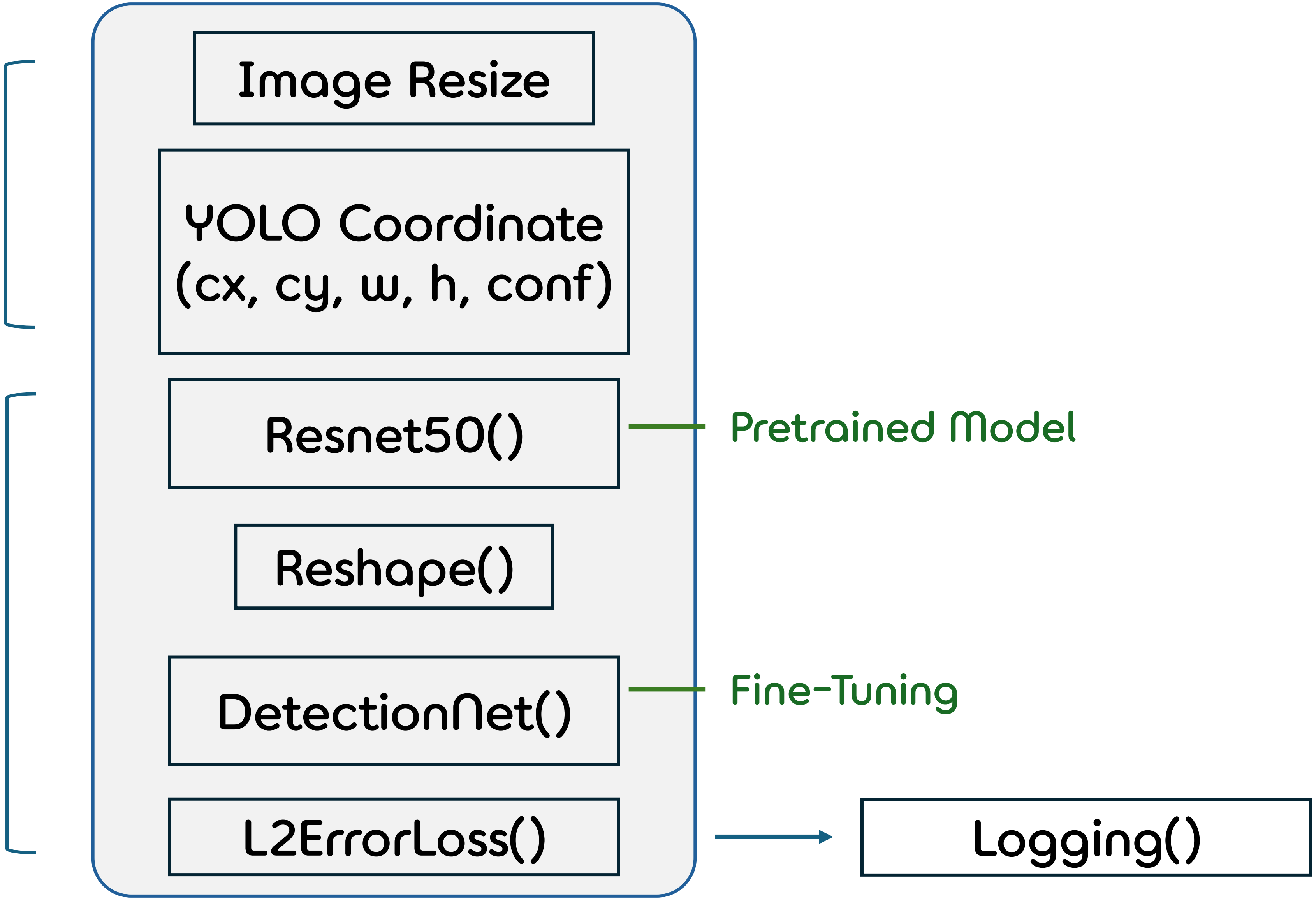


Image Resize

YOLO Coordinate (cx, cy, w, h, conf)

Resnet50()

Pretrained Model

Reshape()

DetectionNet()

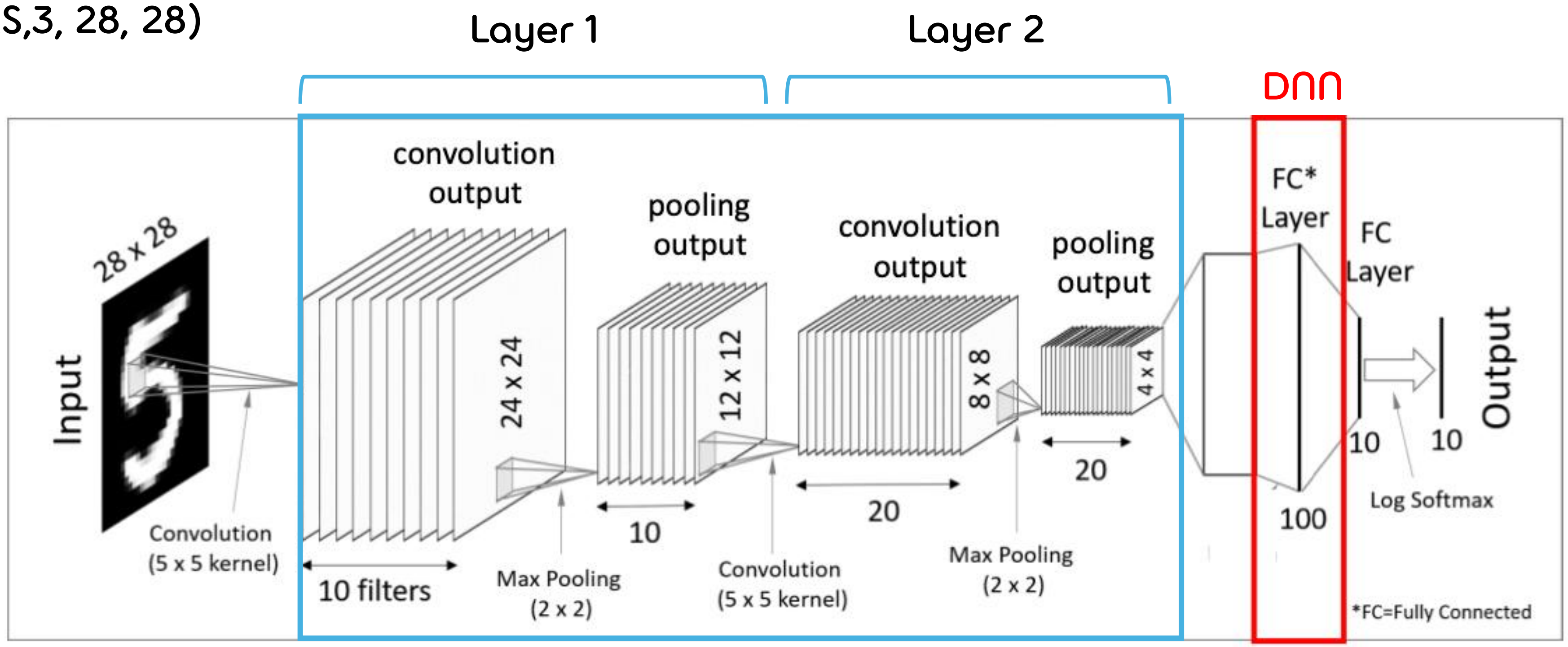
Fine-Tuning

L2ErrorLoss()

Logging()

Remark : CNN

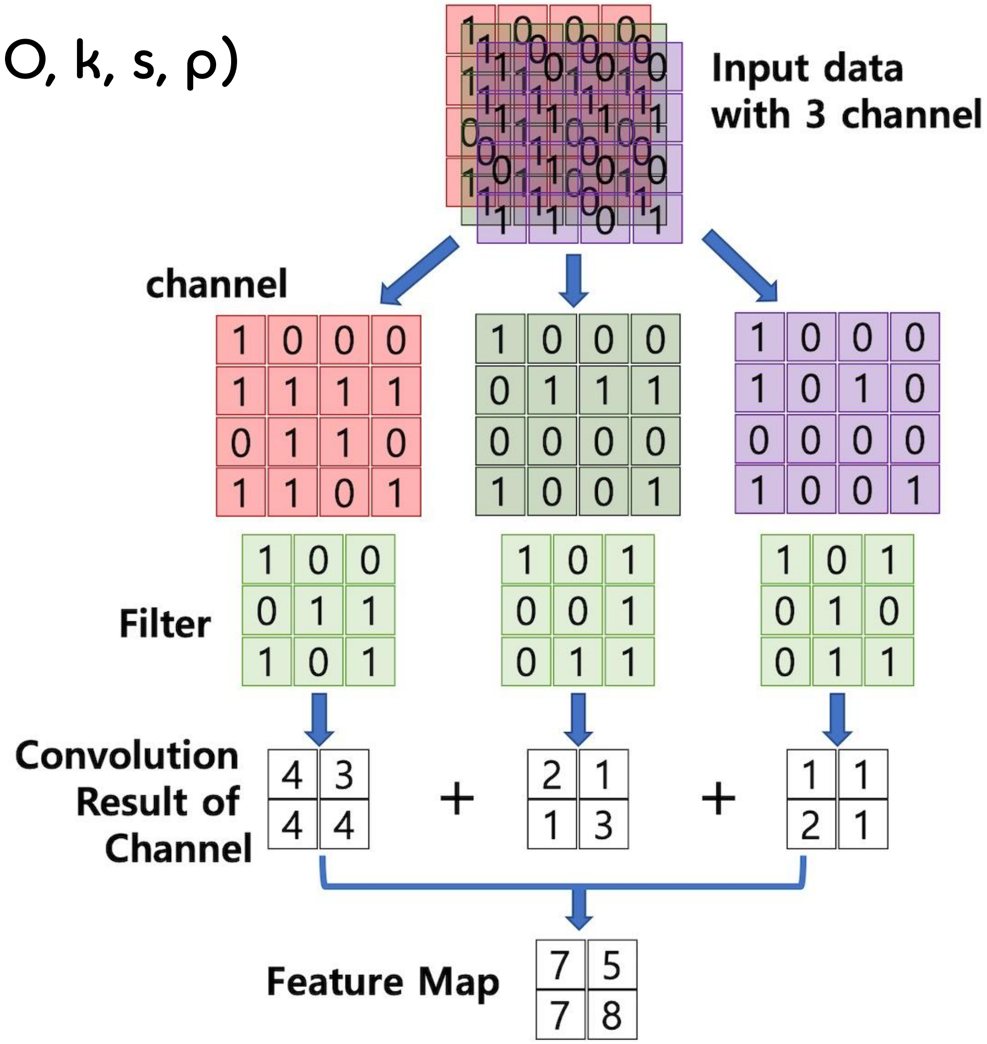
Image Matrix
X = (Batch, S,S,3, 28, 28)



Layer 1 & 2 : Convolution - Activate - Pooling

Remark : Convolution Layer

conv(I, O, k, s, p)



Input data with 3 channel

channel

1	0	0	0
1	1	1	1
0	1	1	0
1	1	0	1

1	0	0	0
0	1	1	1
0	0	0	0
1	0	0	1

1	0	0	0
1	0	1	0
0	0	0	0
1	0	0	1

Filter

1	0	0
0	1	1
1	0	1

1	0	1
0	0	1
0	1	1

1	0	1
0	1	0
0	1	1

Convolution Result of Channel

4	3
4	4

+

2	1
1	3

+

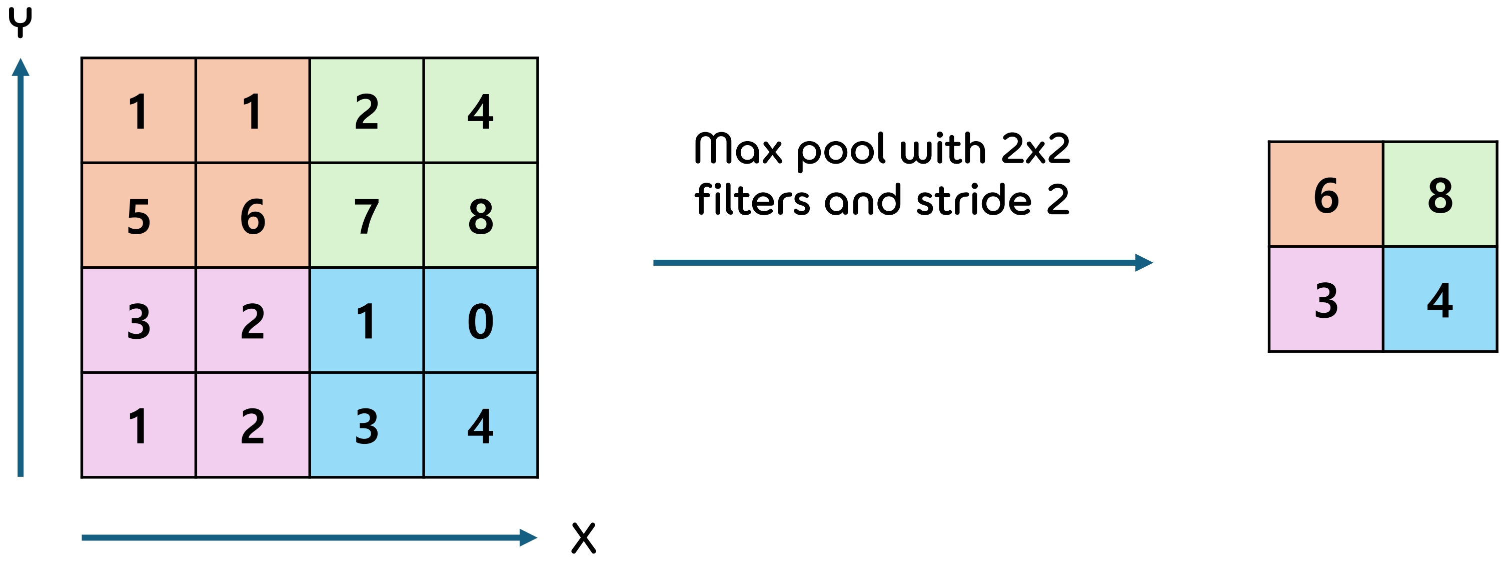
1	1
2	1

Feature Map

7	5
7	8

- I = 3) R,G,B 3차원의 행렬 데이터
- O = 1) Feature Map의 개수
- k = 3) Filter의 Size
- s = 1) Convolution의 보폭
- p = 0) 가장 자리에 값을 채워 줌 (보통 0)

Remark : Pooling



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

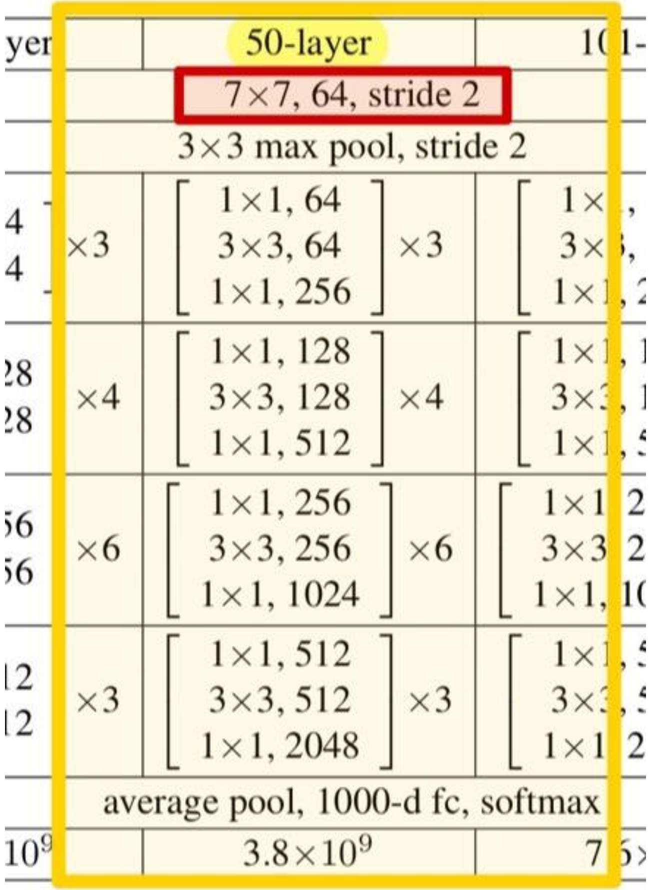
Max pool with 2x2 filters and stride 2

6	8
3	4

1. 정보 손실 위험
2. 학습 대상 x
3. 위치 정보 보존 취약

Stride Convolution 으로 대체
Pooling 지양하는 추세

Remark : Resnet50



yer		50-layer	10 1-
	7×7, 64, stride 2		
	3×3 max pool, stride 2		
4 4	×3	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	1× , 3× , 1×1, 2
28 28	×4	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	1×1, 1 3×, 1 1×1, 5
56 56	×6	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	1×1 2 3×3 2 1×1, 10
12 12	×3	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	1×1, 5 3×, 5 1×1 2
	average pool, 1000-d fc, softmax		
10^9		3.8×10^9	7.6×

X = (n,3,448,448)

CONV1 = (n,64,224,224)

POOL1 = (n,64,112,112)

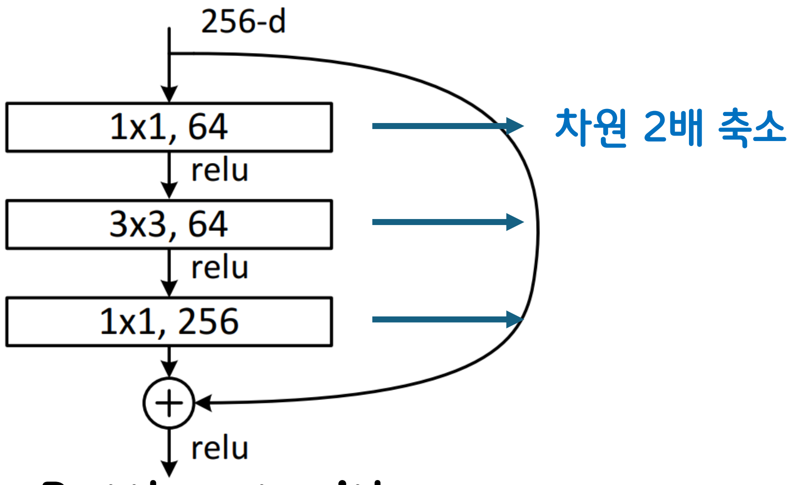
CONV2 = (n,256,112,112) x3

CONV3 = (n,512,56,56) x4

CONV4 = (n,1024,28,28) x6

CONV5 = (n,2048,14,14) x3

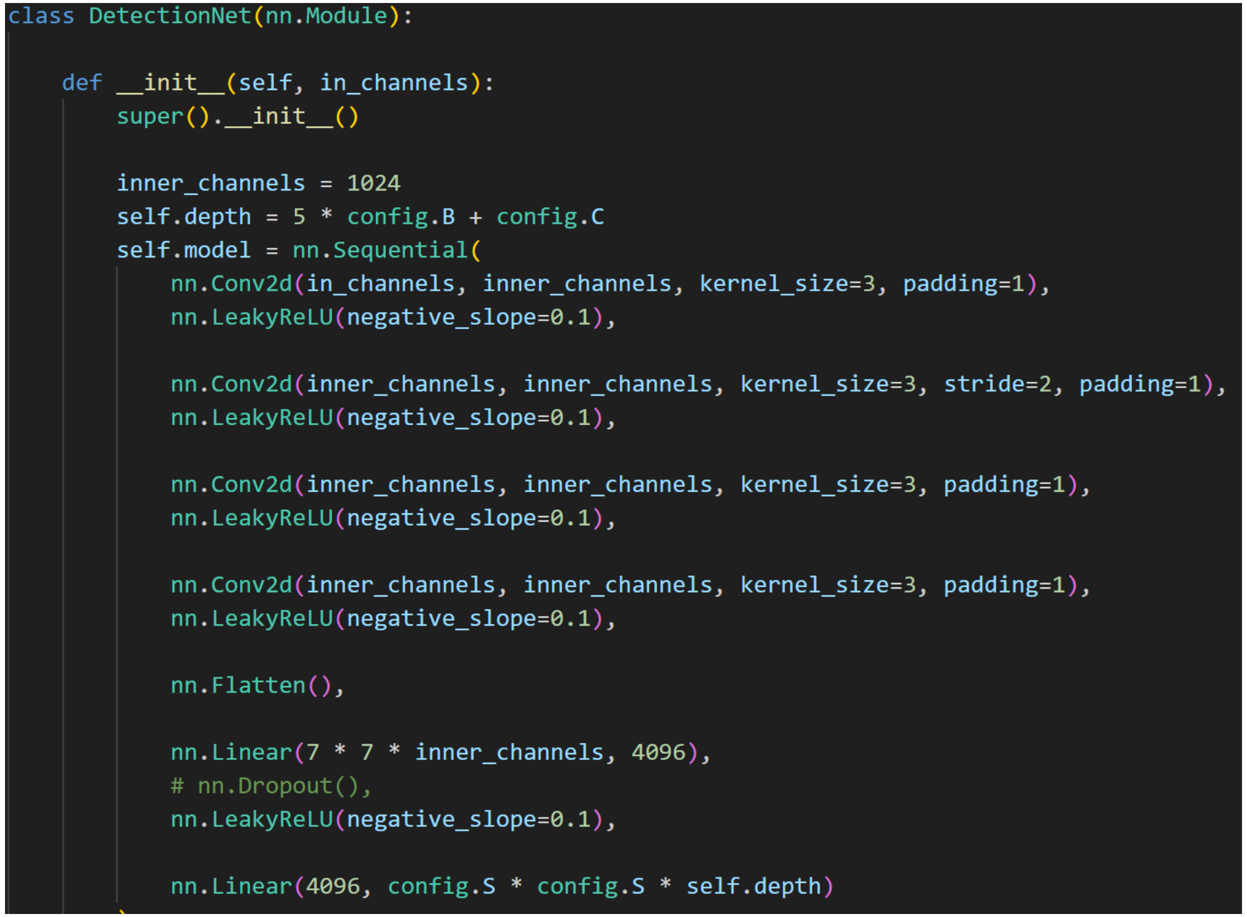
Del AVG-Pool / Full-Connected



Bottlenet with residual function

Deep Residual Learning for Image Recognition

Remark : DetectionNet()



```
class DetectionNet(nn.Module):

    def __init__(self, in_channels):
        super().__init__()

        inner_channels = 1024
        self.depth = 5 * config.B + config.C
        self.model = nn.Sequential(
            nn.Conv2d(in_channels, inner_channels, kernel_size=3, padding=1),
            nn.LeakyReLU(negative_slope=0.1),

            nn.Conv2d(inner_channels, inner_channels, kernel_size=3, stride=2, padding=1),
            nn.LeakyReLU(negative_slope=0.1),

            nn.Conv2d(inner_channels, inner_channels, kernel_size=3, padding=1),
            nn.LeakyReLU(negative_slope=0.1),

            nn.Conv2d(inner_channels, inner_channels, kernel_size=3, padding=1),
            nn.LeakyReLU(negative_slope=0.1),

            nn.Flatten(),

            nn.Linear(7 * 7 * inner_channels, 4096),
            # nn.Dropout(),
            nn.LeakyReLU(negative_slope=0.1),

            nn.Linear(4096, config.S * config.S * self.depth)
```

X = (N,2048,14,14) / 14x14 filter (number 2048)

CONV1 = (N,1024,14,14)

CONV2 = (N,1024,7,7)

CONV3 = (N,1024,7,7)

CONV4 = (N,1024,7,7)

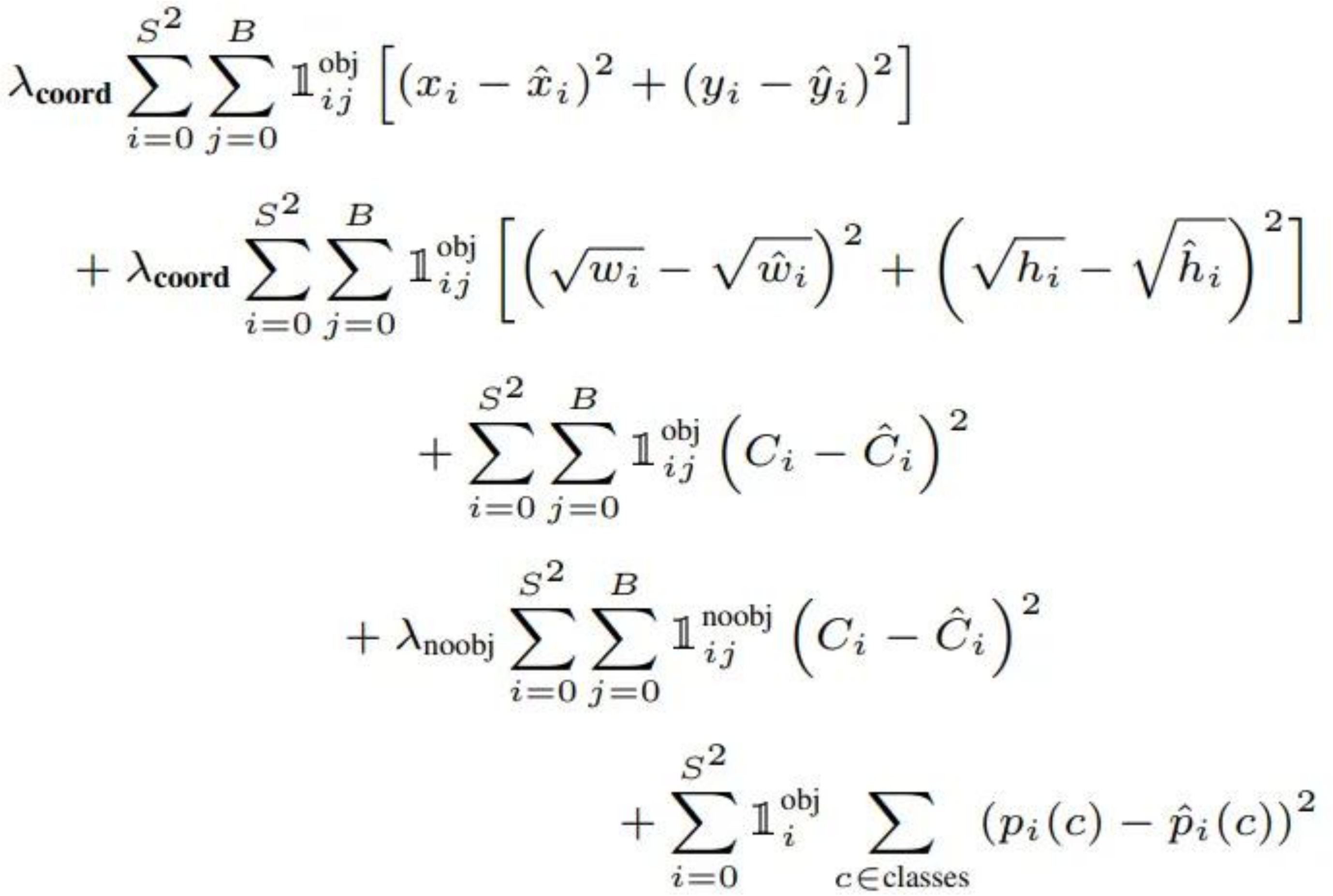
Flatten = (N,50176)

FC1 = (N,4096)

FC2 = (N, T) * T = (7, 7, 5B + C)

4. Loss & Training

4 L2ErrorLoss()



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

Remark : IOU

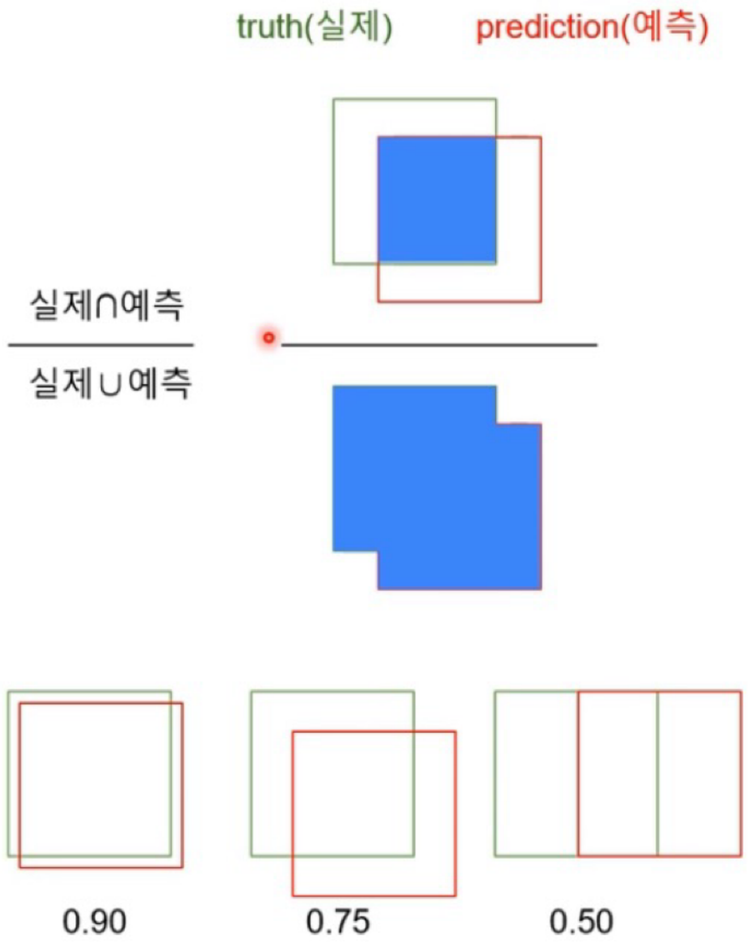
confidence score

$$\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

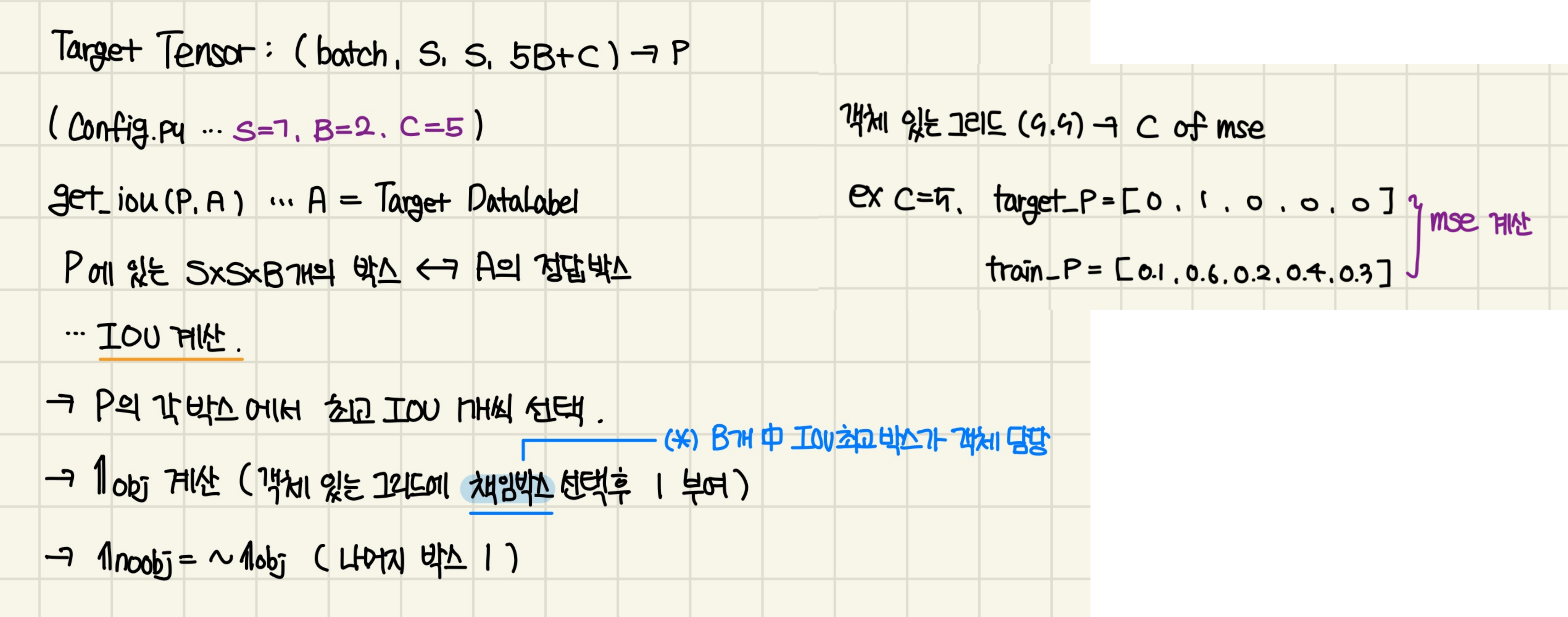
(객체가 확실히 없으면 0) * IOU = 0

(객체가 확실히 있으면 1) * IOU = IOU

IOU



4 L2ErrorLoss()



Target Tensor : (batch, S, S, 5B+C) → P

(Config.py … S=7, B=2, C=5)

get_iou(P, A) … A = Target DataLabel

P에 있는 SxSxB개의 박스 ↔ A의 정답박스

… IOU 계산.

→ P의 각 박스에서 최고 IOU 1개씩 선택.

(*) B개 中 IOU최고박스가 객체 담당

→ 1_{obj} 계산 (객체 있는 그리드에 책임박스 선택후 1 부여)

→ $1_{noobj} = \sim 1_{obj}$ (나머지 박스 1)

객체 있는 그리드 (9.9) → C of mse

ex C=5, target_P = [0, 1, 0, 0, 0]

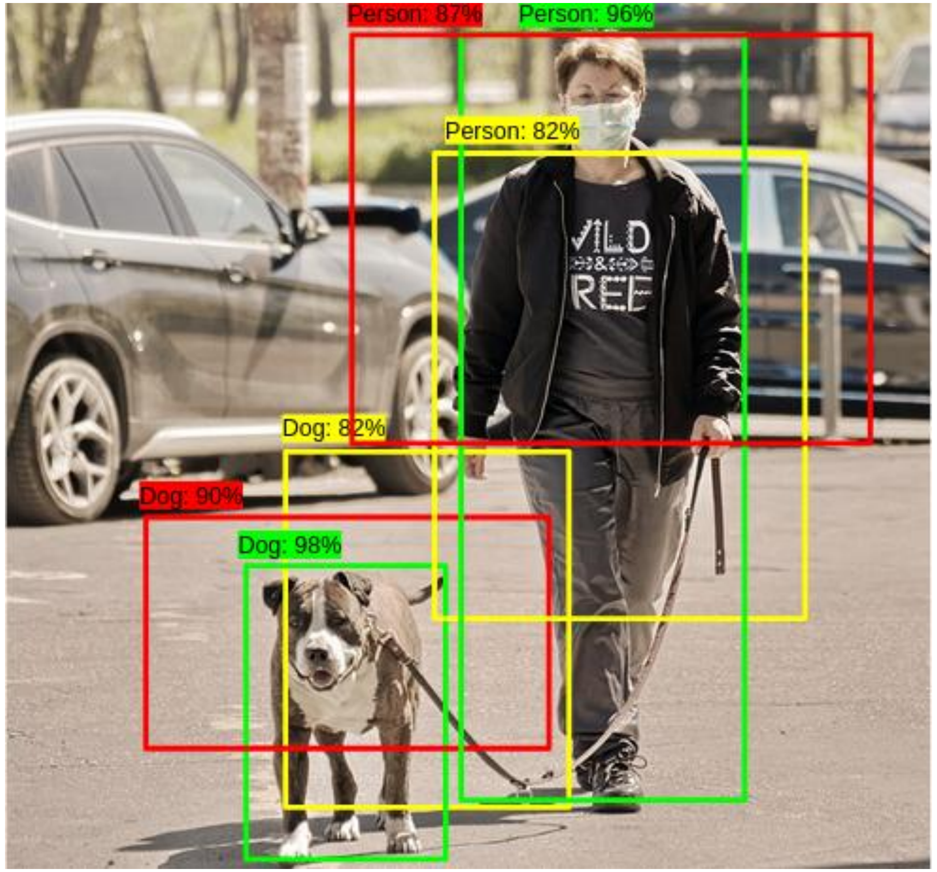
train_P = [0.1, 0.6, 0.2, 0.4, 0.3]

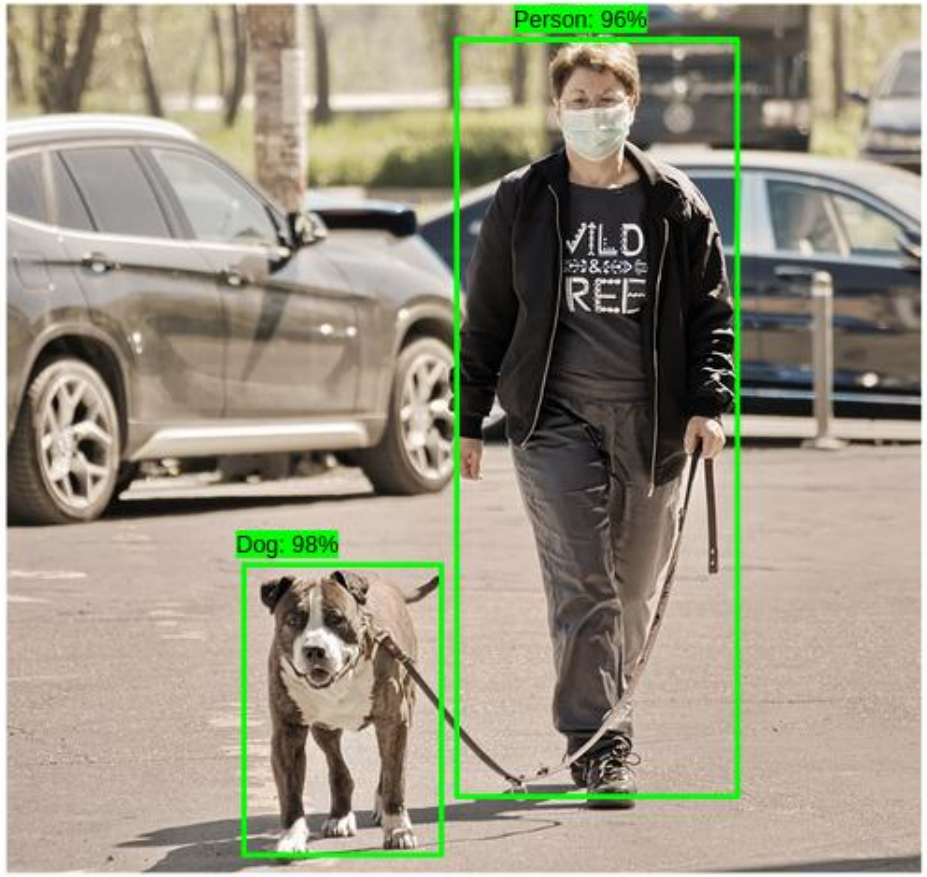
mse 계산

5. Test, NMS

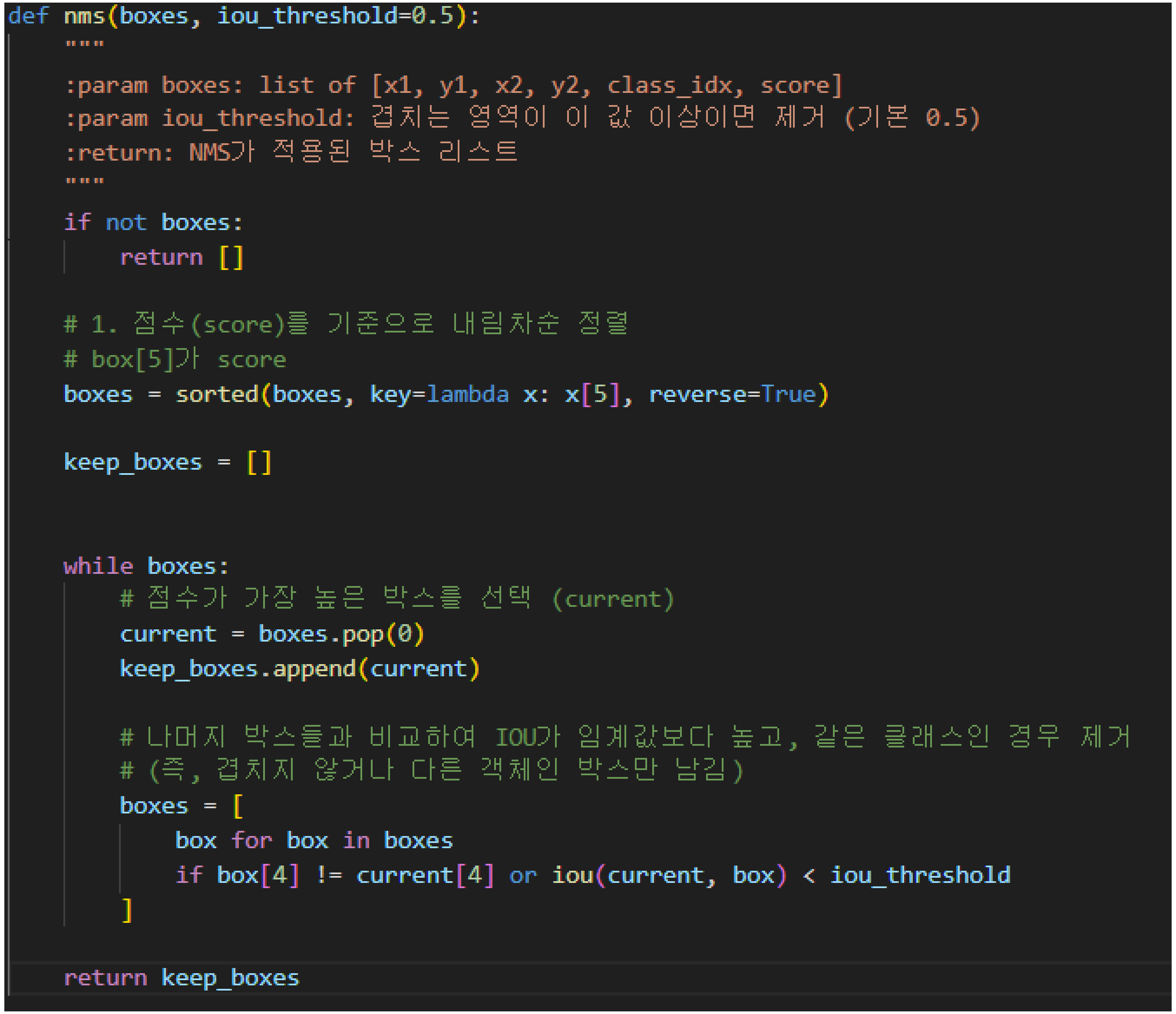
5 nms

1. Class 별로 class- specific confidence score 가 임계값 이하인 box 제거
2. 남은 bbox간 IOU 계산해 IOU가 임계값 이상이면 중복탐지 간주하고 최고값 제외 나머지 제거





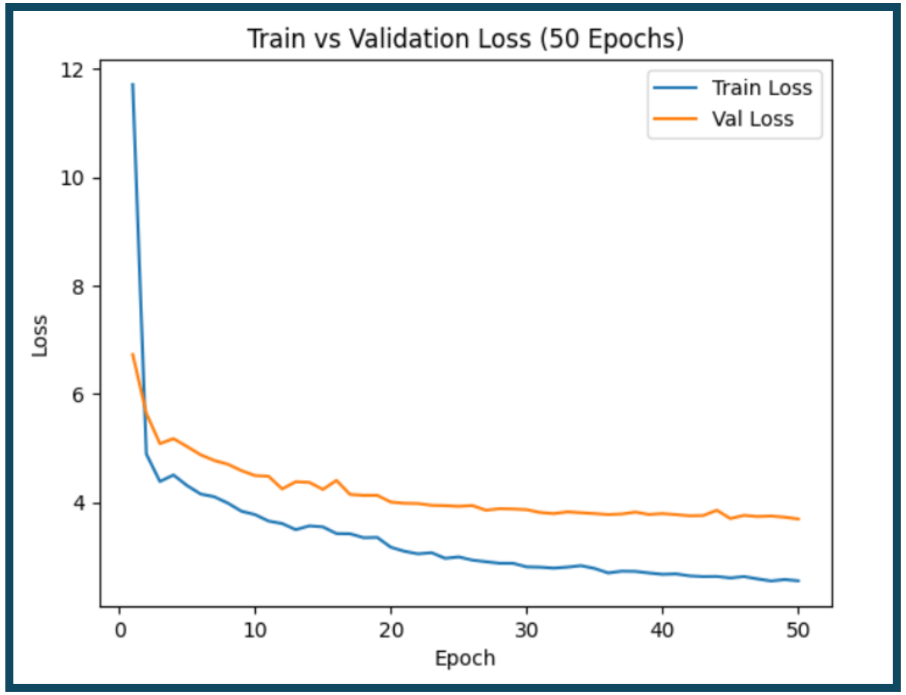
5 nms

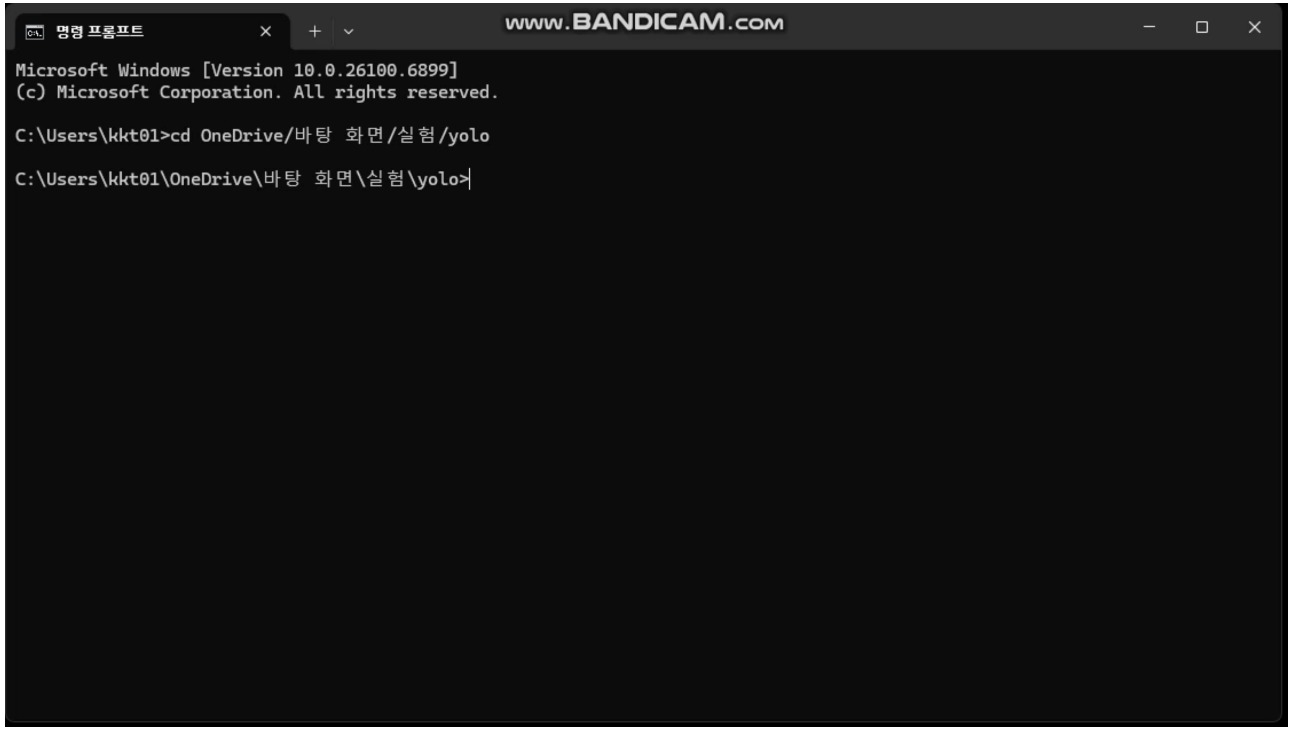


```python
def nms(boxes, iou_threshold=0.5):
    """
    :param boxes: list of [x1, y1, x2, y2, class_idx, score]
    :param iou_threshold: 겹치는 영역이 이 값 이상이면 제거 (기본 0.5)
    :return: NMS가 적용된 박스 리스트
    """
    if not boxes:
        return []

    # 1. 점수(score)를 기준으로 내림차순 정렬
    # box[5]가 score
    boxes = sorted(boxes, key=lambda x: x[5], reverse=True)

    keep_boxes = []


    while boxes:
        # 점수가 가장 높은 박스를 선택 (current)
        current = boxes.pop(0)
        keep_boxes.append(current)

        # 나머지 박스들과 비교하여 IOU가 임계값보다 높고, 같은 클래스인 경우 제거
        # (즉, 겹치지 않거나 다른 객체인 박스만 남김)
        boxes = [
            box for box in boxes
            if box[4] != current[4] or iou(current, box) < iou_threshold
        ]

    return keep_boxes
```

6. Result Analysis

6 Result Analysis : Train

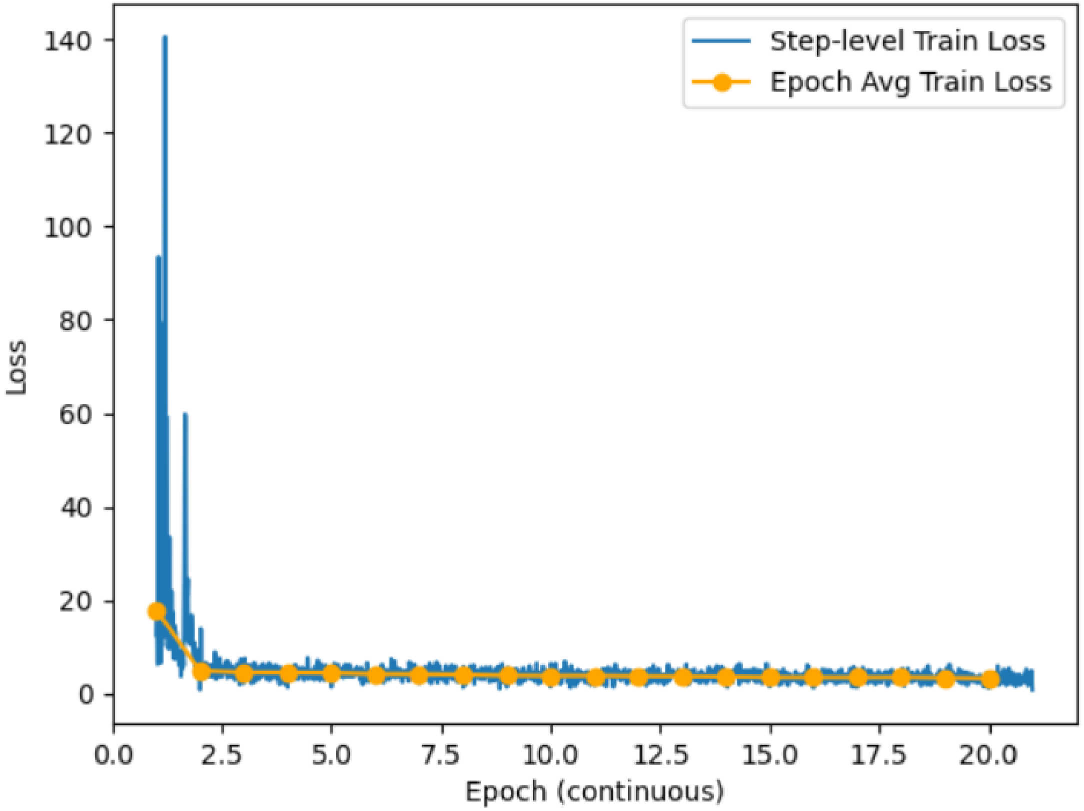




```
Microsoft Windows [Version 10.0.26100.6899]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kkt01>cd OneDrive/바탕 화면/실험/yolo

C:\Users\kkt01\OneDrive\바탕 화면\실험\yolo>
```

Cpu : i7-13600KF
GPU : RTX 3070 (8GB)
RAM : DDR5 32GB

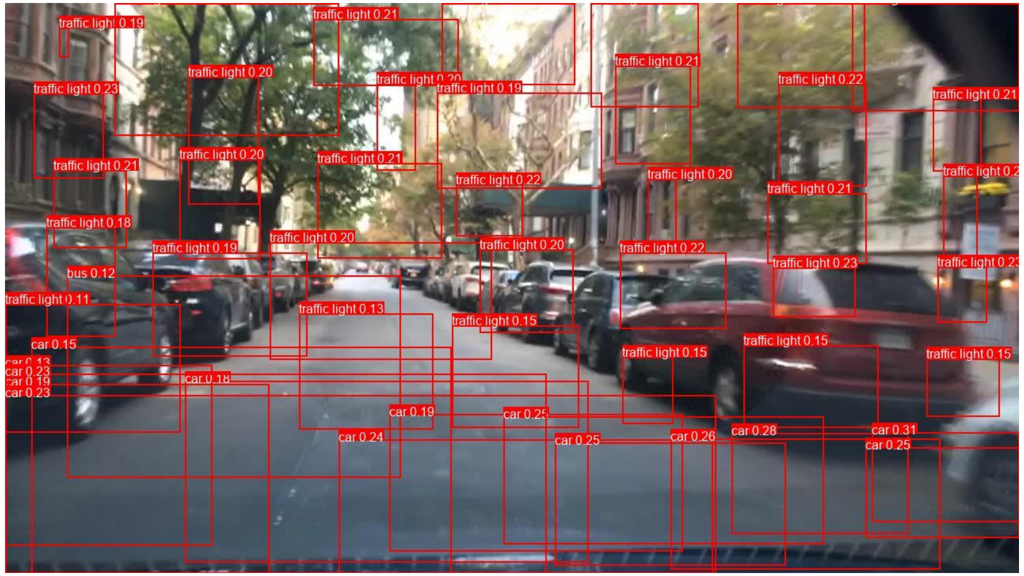
6 Result Analysis



It’s Ok !

6 Result Analysis

Before NMS

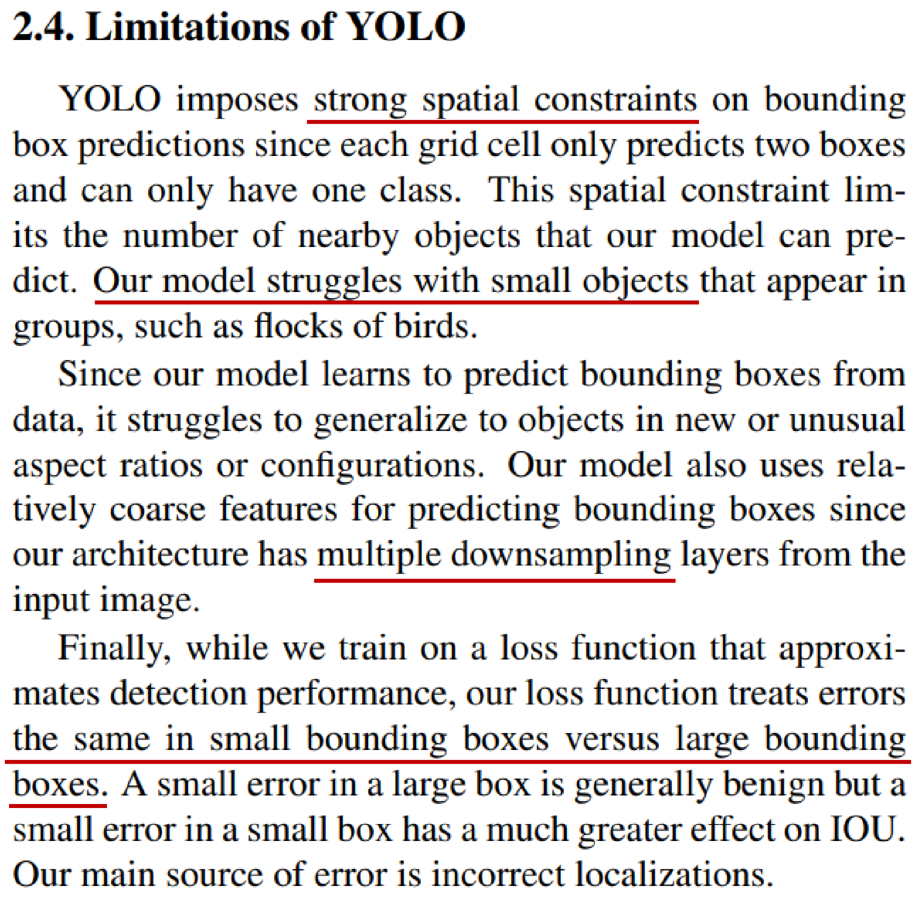


Test Result

1. Optimizer (SGD, Adamw)
2. Activation (Sigmoid, ReLU)
3. Loss function
4. 100K Dataset

No change

6 Result Analysis



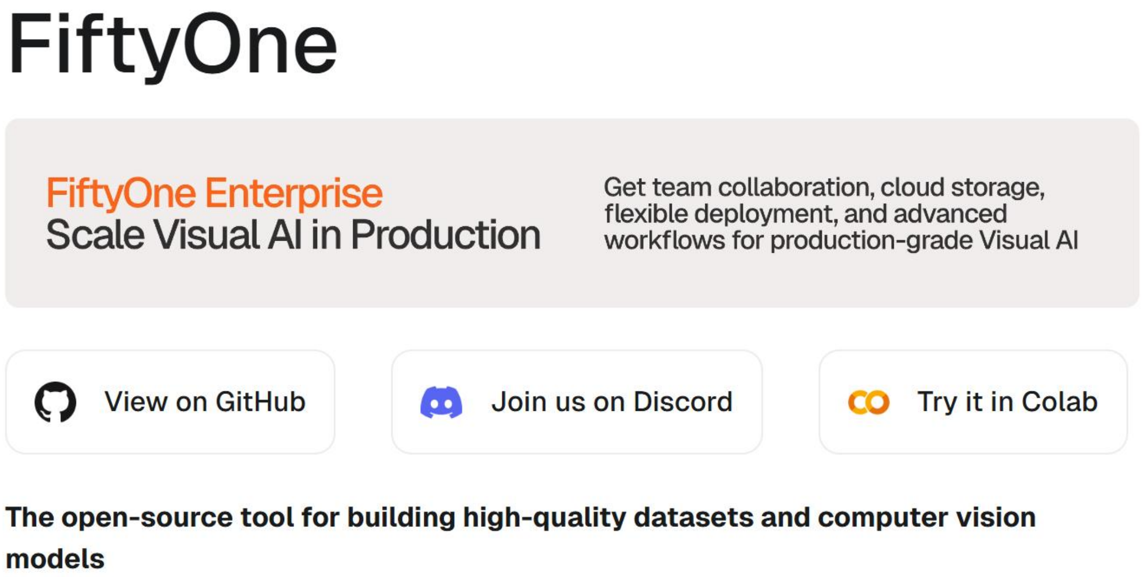
2.4. Limitations of YOLO

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. Our model struggles with small objects that appear in groups, such as flocks of birds.

Since our model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. Our model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image.

Finally, while we train on a loss function that approximates detection performance, our loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU. Our main source of error is incorrect localizations.

YOLO paper



FiftyOne

FiftyOne Enterprise
Scale Visual AI in Production

Get team collaboration, cloud storage, flexible deployment, and advanced workflows for production-grade Visual AI

The open-source tool for building high-quality datasets and computer vision models

Dataset Change !

6 Result Analysis

BDD100K

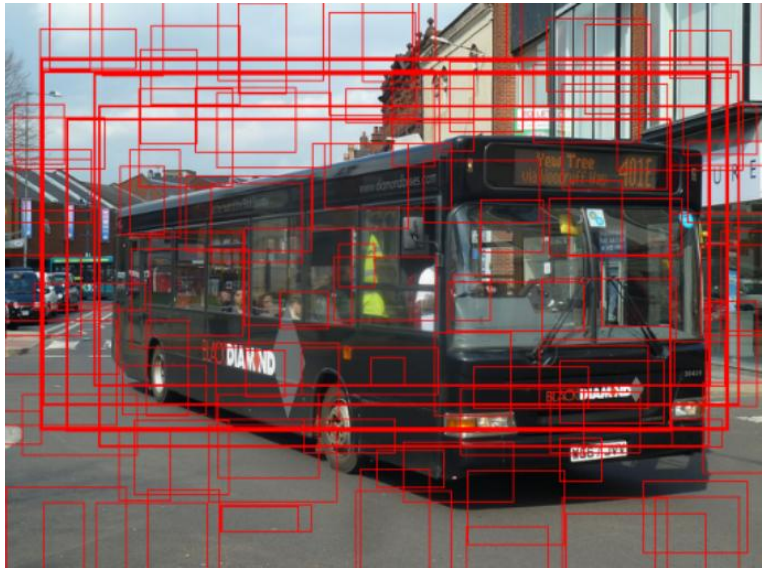
Evalution Data

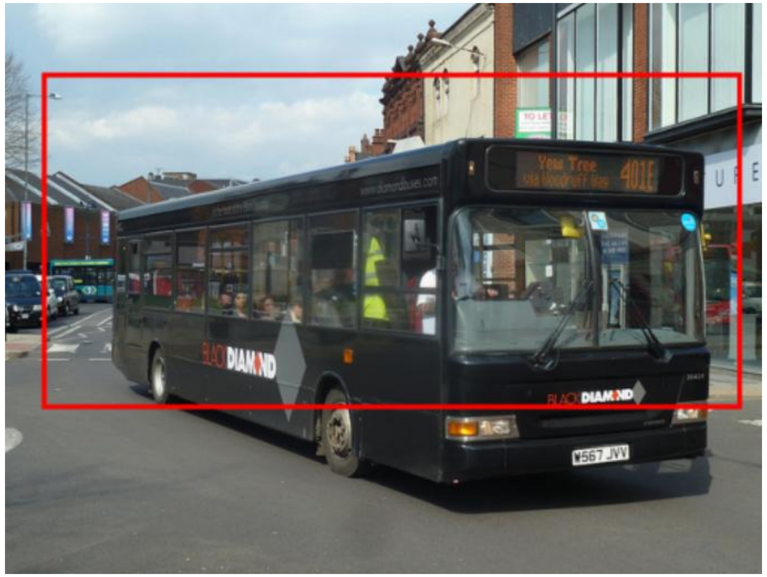
COCO

Train Data

6 Result Analysis

JSONDetection(Dataset) → COCODetection(Dataset)





6 Result Analysis

1. ‘YOLO’라는 CNN 기반의 객체 탐지 모델의 원리

2. Grid 기반 Box 알고리즘이 밀집 장면에서 병목이 된다

작은 객체 / 밀집 객체

3. YOLOv1의 성능은 데이터 분포에 강하게 의존한다

7. Future Work

7 Future Work

1. **자율주행 및 운전자 보조시스템**(ADAS) **핵심 모듈**

   신호등 인식 및 속도제어, 보행자 충돌방지

   Detection + Prediction

2. **지능형 교통 시스템**(smart city)

   스마트 신호 제어, 차종별 통행량 분석

3. **악천후 및 야간 환경 모니터링**

Thank You