2020.Sejong.RCVWS

-2.5D Pedestrian Detection-

bigchan@rcv.sejong.ac.kr

jiwon@rcv.sejong.ac.kr

3D Detection

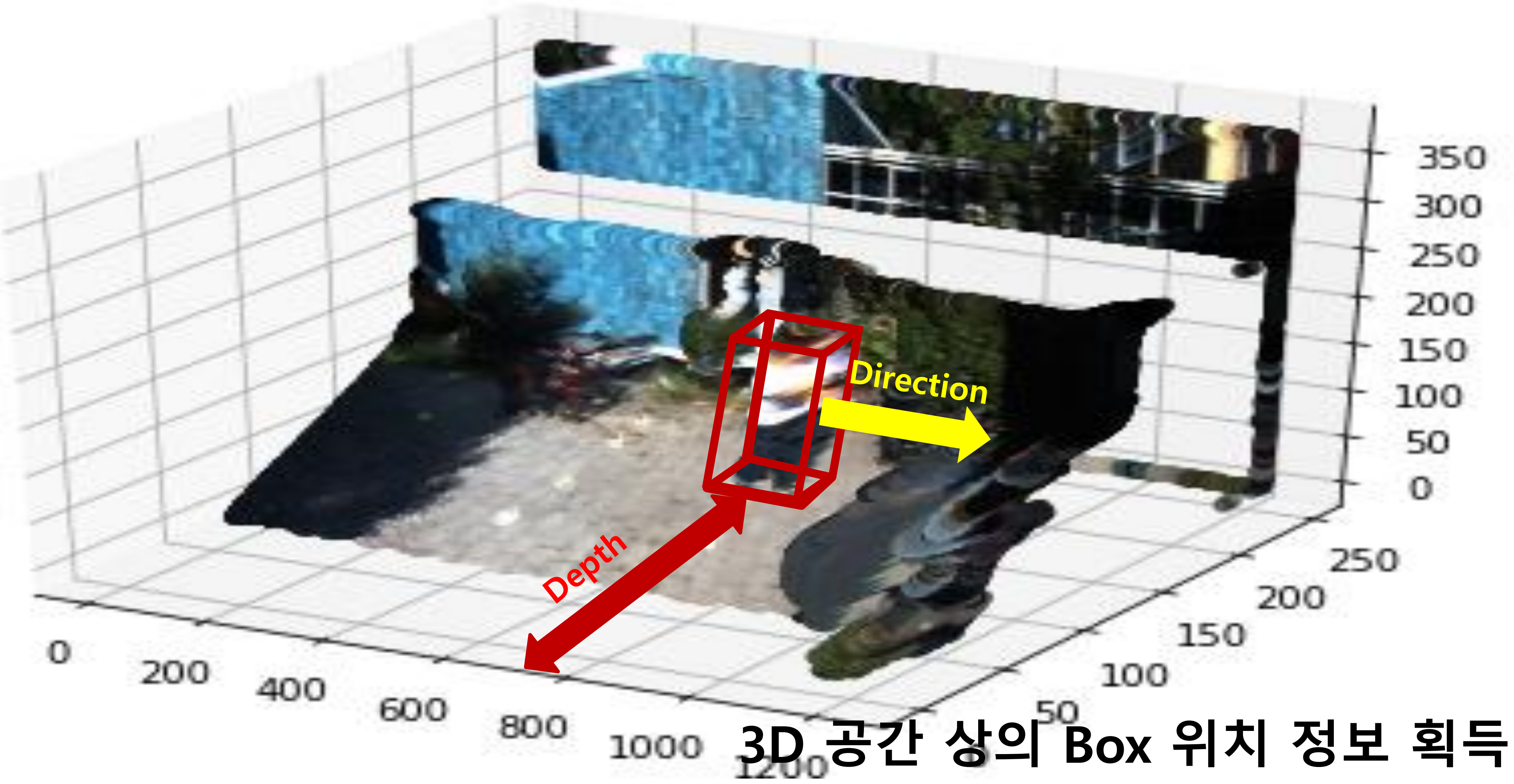
3D vs 2D Detection

-2D

2D Detection= object의 [x1,y1,x2,y2] 구함

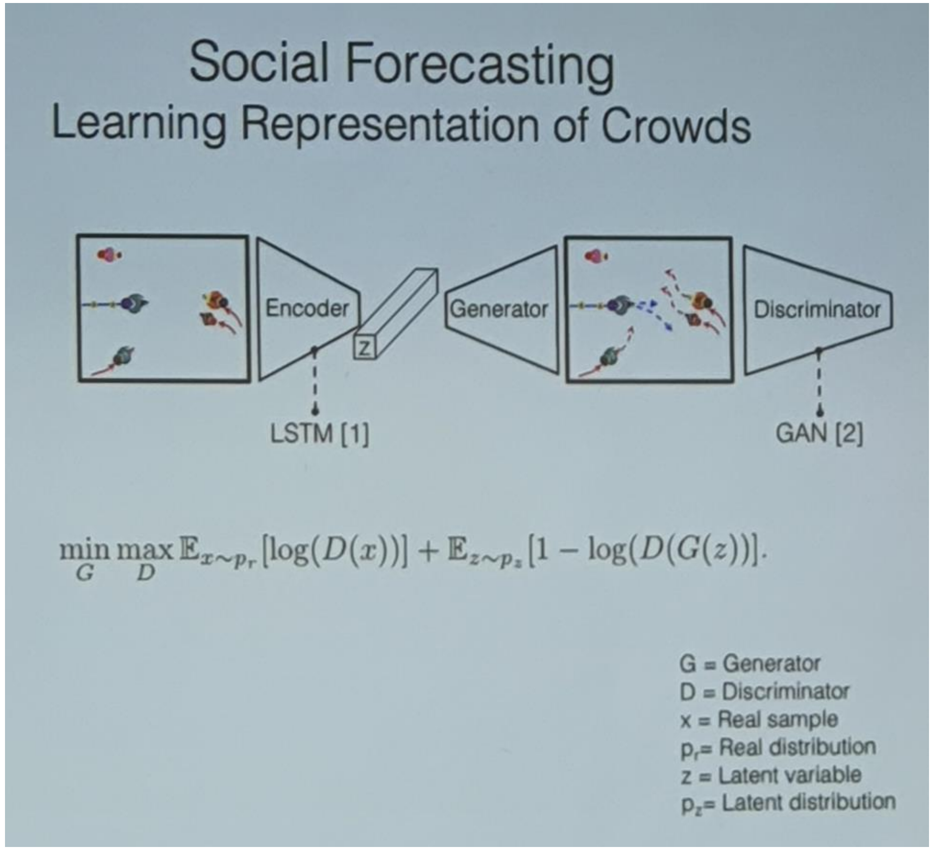
3D vs 2D Detection

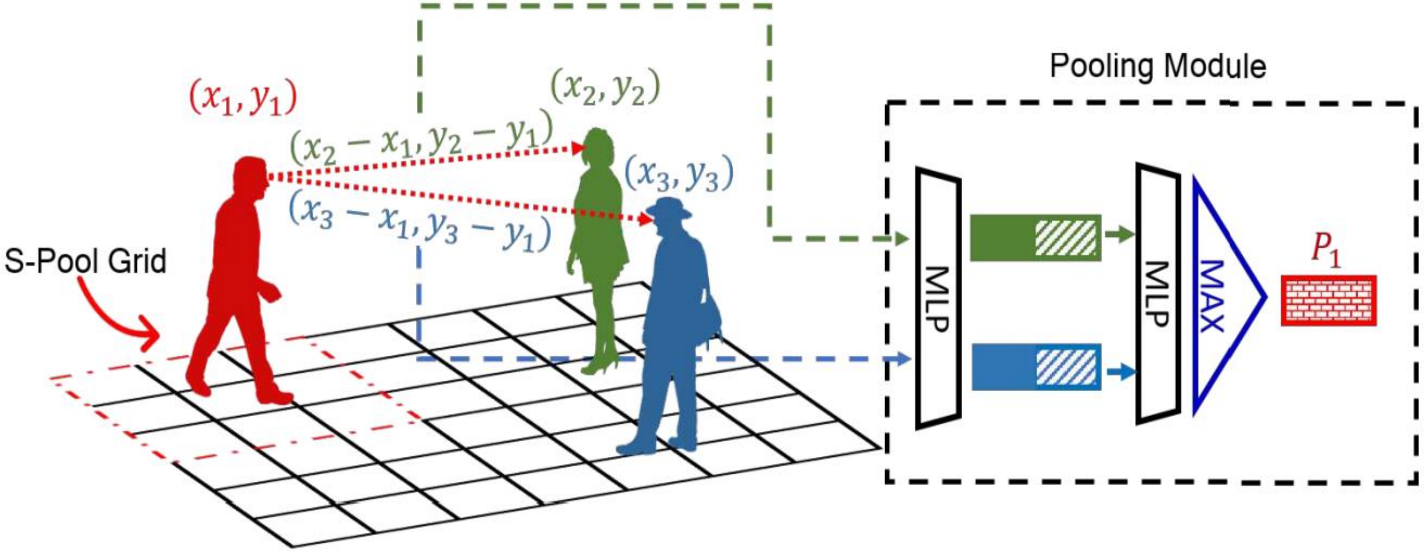
-3D



3D 공간 상의 Box 위치 정보 획득

Direction





- 보행자의 예상 경로 예측

Gupta, Agrim, et al. "Social gan: Socially acceptable trajectories with generative adversarial networks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

Depth

Ground Truth

Too Near 0~2m

Near 2~4m

Moderate 4~6m

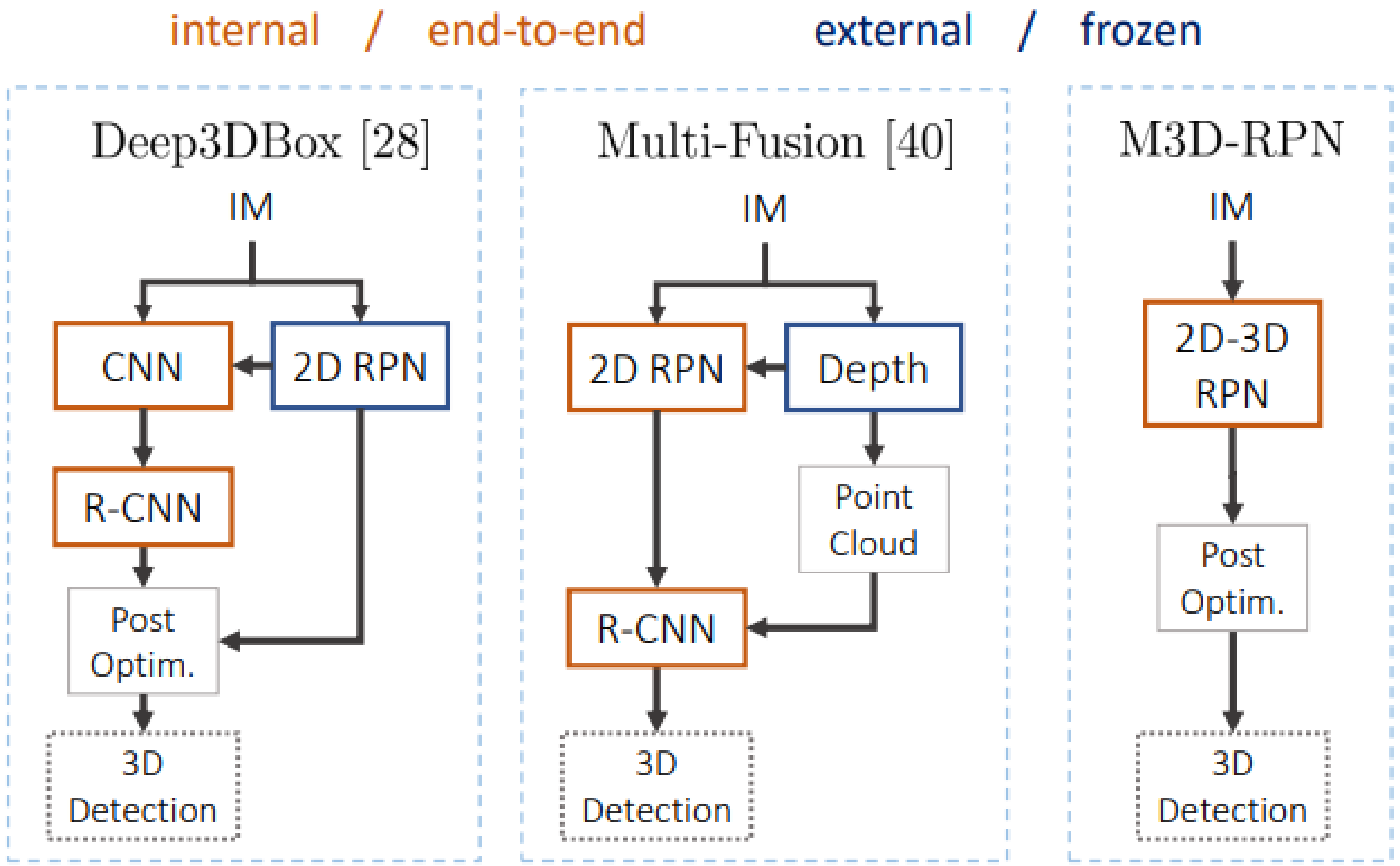
Far 6~ m

M3D-RPN: Monocular 3D Region Proposal Network for Object Detection

Garrick Brazil, Xiaoming Liu Michigan State University,
East Lansing MI

ICCV2019

Single-Shot Network



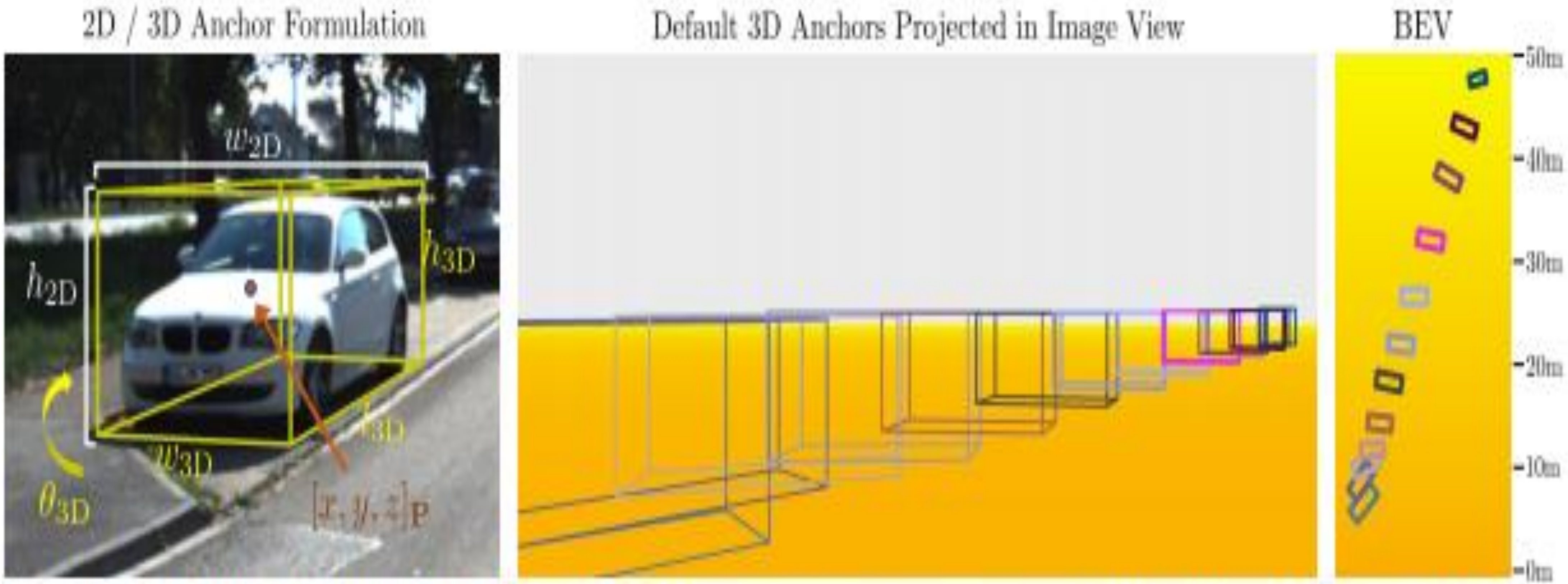
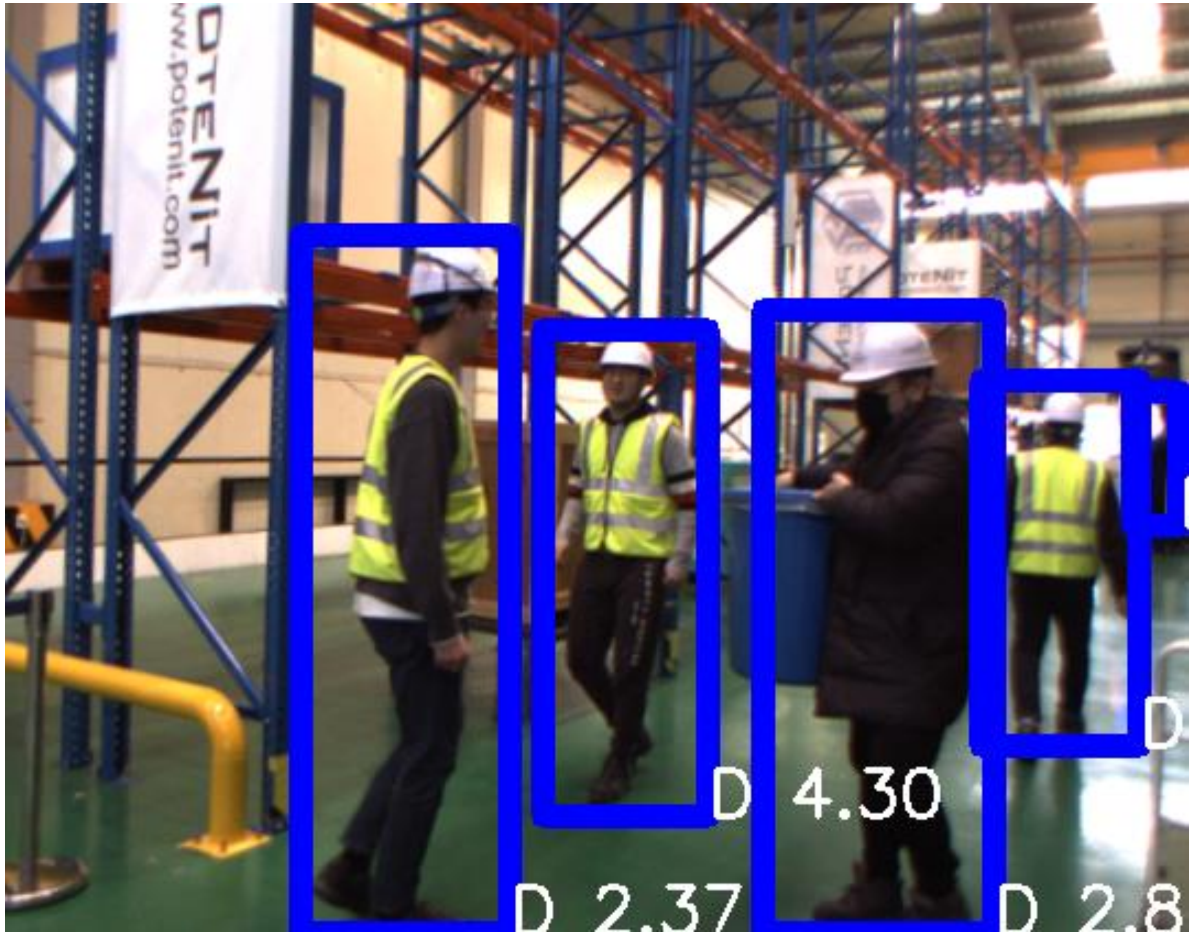


Figure 4. **Anchor Formulation** and **Visualized 3D Anchors**. We depict each parameter of within the 2D / 3D anchor formulation (left). We visualize the precomputed 3D priors when 12 anchors are used after projection in the image view (middle) and Bird's Eye View (right). For visualization purposes only, we span anchors in specific x_{3D} locations which best minimize overlap when viewed.

3D vs 2.5D

-2.5D



보행자의 Direction 불필요

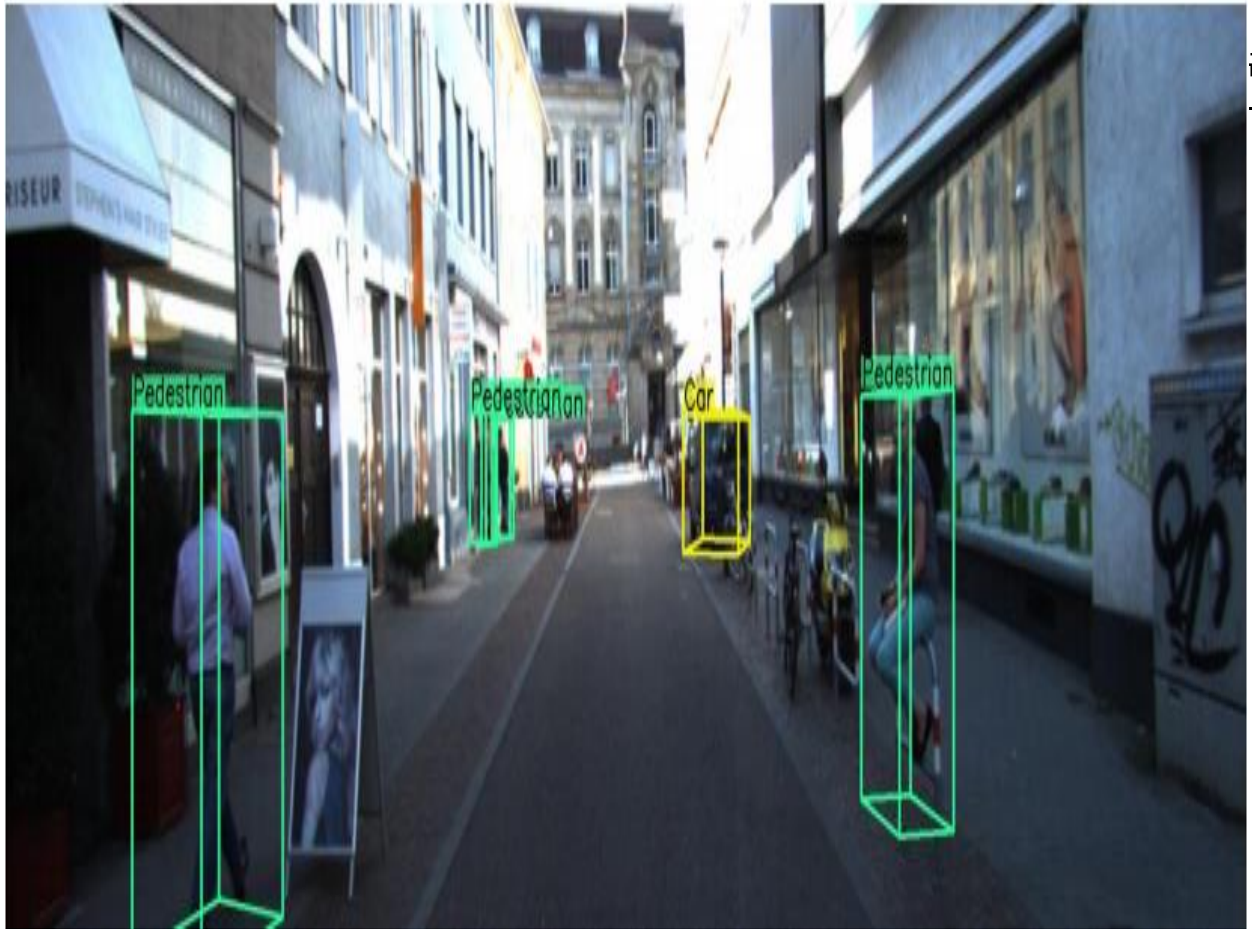
보행자의 Depth 값만 필요

박스에 대표되는 Depth 값 하나만 있으면 됨

GT 제작에 용이

3D vs 2.5D

-3D



#Values	Name	Description
1	type	Describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'DontCare'
1	truncated	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries
1	occluded	Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of object, ranging [-pi..pi]
4	bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x,y,z in camera coordinates (in meters)
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi]
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.

Pedestrian 0.00 0 -0.20 712.40 143.00 810.73 307.92 1.89 0.48 1.20 1.84 1.47 8.41 0.01
직육면체의 정보가 다 필요.

2.5D Detection

2.5D Depth Regression

기존 2D Detection

Detection model

loc

cls

X	Y	X	Y

2.5D Depth Regression

2D Detection+ Depth Regression

Detection model

loc

cls

X Y X Y + Depth

단순히 Depth 만 Regression 하면 성능이 나오지 않음

Multi classification

Ground Truth

Too Near 0~2m
Near 2~4m
Moderate 4~6m
Far 6~ m

2.5D Depth classification

Detection model

loc

X	Y	X	Y

cls

Backgroud	Pedestrian

기존 2D Detection

2.5D Depth classification

Detection model

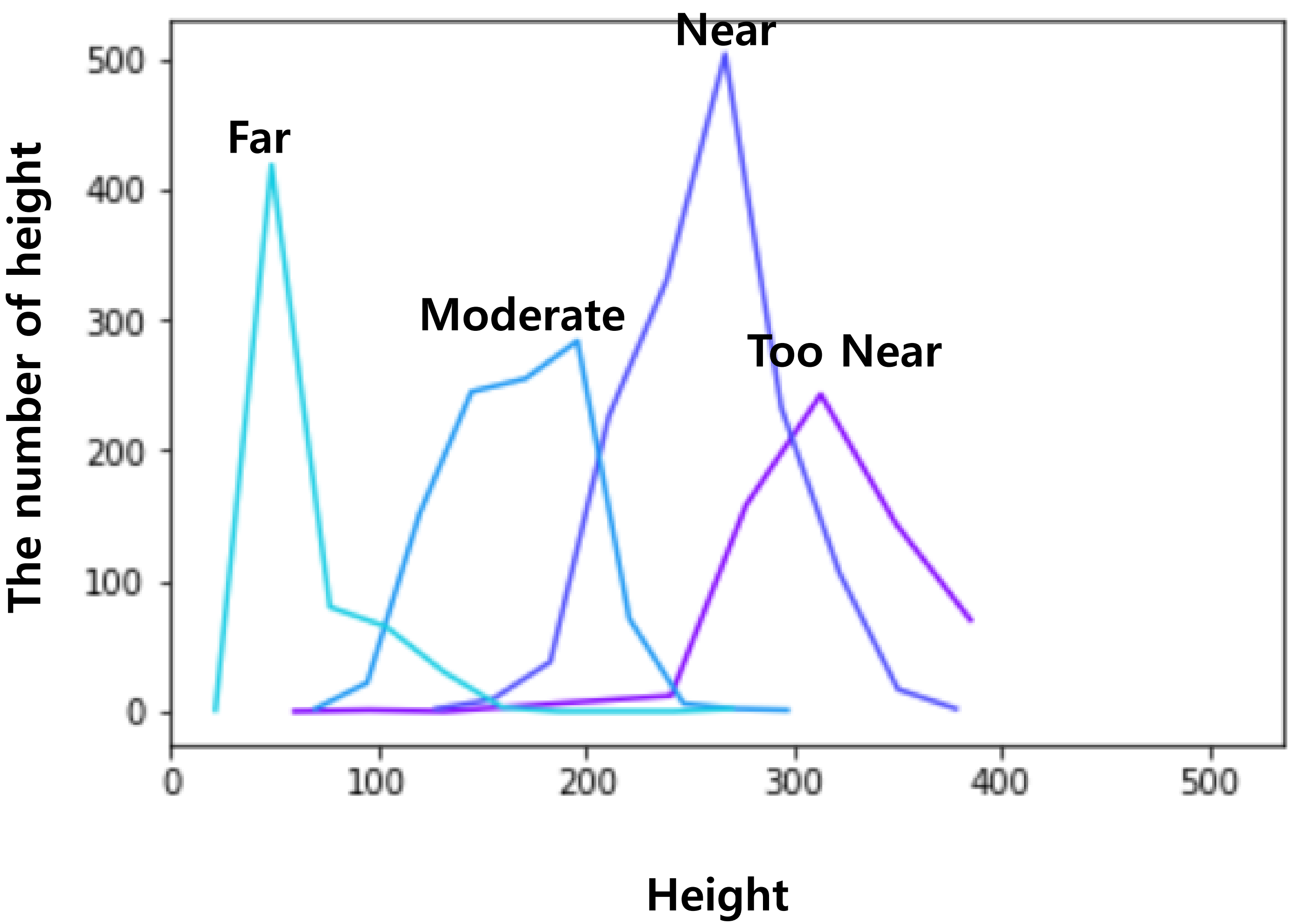
loc

X	Y	X	Y

cls

	Backgroud	Pedestrian2M	Pedestrian4M	... etc
Multi class	0	0	1	... etc
Multi label	0	1	1	... etc

Depth Label Processing



Depth Label Processing

https://github.com/sejong-rcv/2020.RCVWS/blob/master/6%EC%9D%BC%EC%B0%A8_2.5D_Detection/Make_DepthLabel.ipynb

Label를 나누는 코드는 위에 올라 와 있습니다.

Depth Label Processing

기존 json(Label) 을 통해 Depth 값 받기

↓

정해둔 Depth 범위에 따라 라벨링 새로 해서 Json을 저장

↓

새로 저장한 Json에서 Depth 와 Height를 불러와 라벨에 따라 저장

↓

각 라벨을 10개의 구간으로 나눠서 그 구간에 빈도수를 저장

↓

저장된 빈도수와 그 빈도수의 heigh를 시각화

Depth Label Processing

Test

(Easy) Label을 변화 시켜 보면서 Height가 안 겹치도록 경향성을 실험해보기

(Hard) MultiLabel classification 을 위한 json 제작

System Architecture

R2T2 시스템을 통한 데이터 수집

모델 학습

Calibration Parameter 이용해 영상 변환 및 뎁스 추출

Annotation 통해 Ground Truth (Bounding Box) 추출

2D Detect model 학습

2.5D Detect model 학습

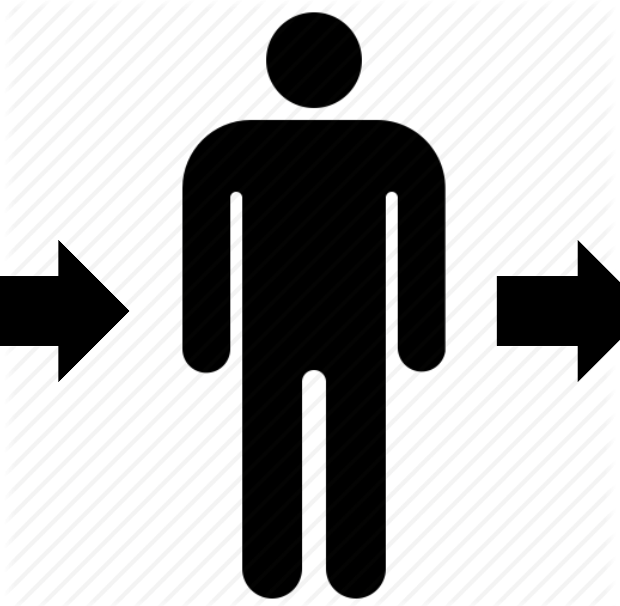
모델 실시간 Inference

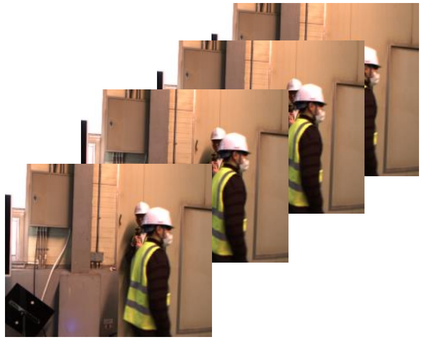
실시간으로 받은 영상 Rectify 와 스티칭 진행

Python -> C++로 변환된 모델에 forward

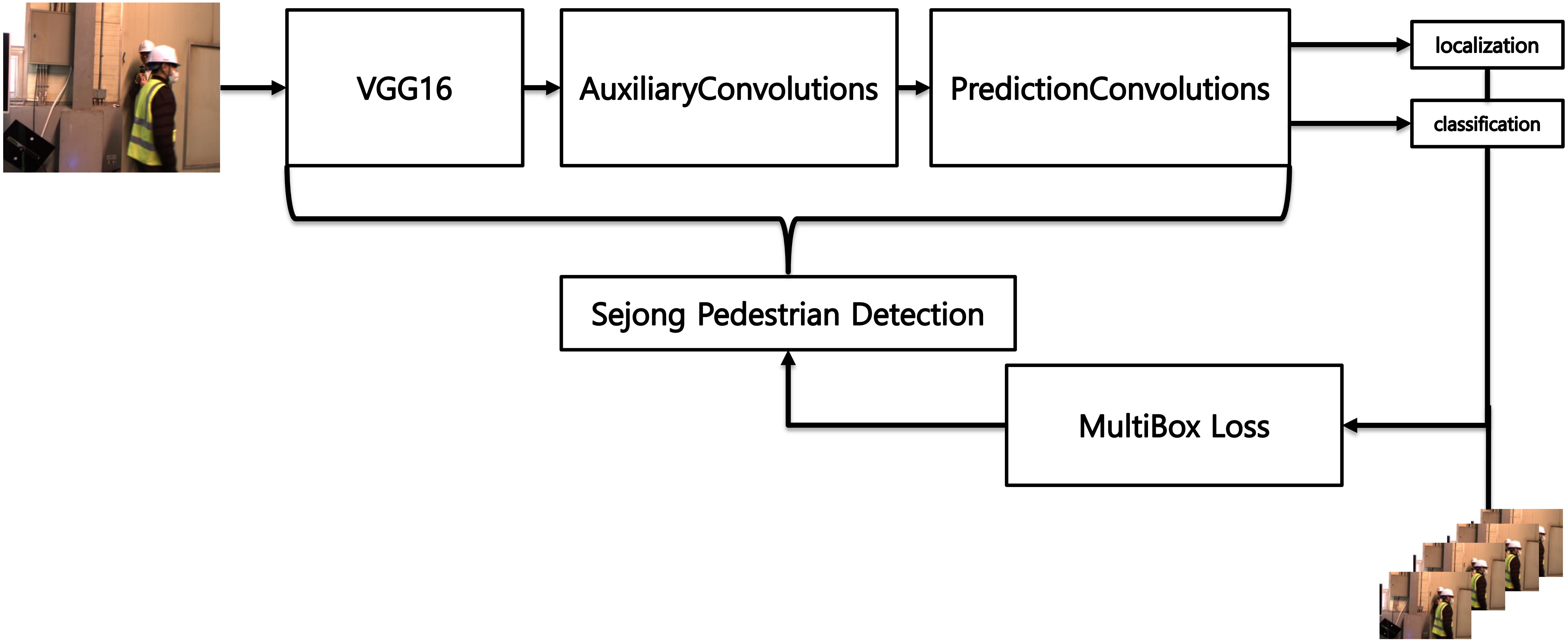
Lunch time

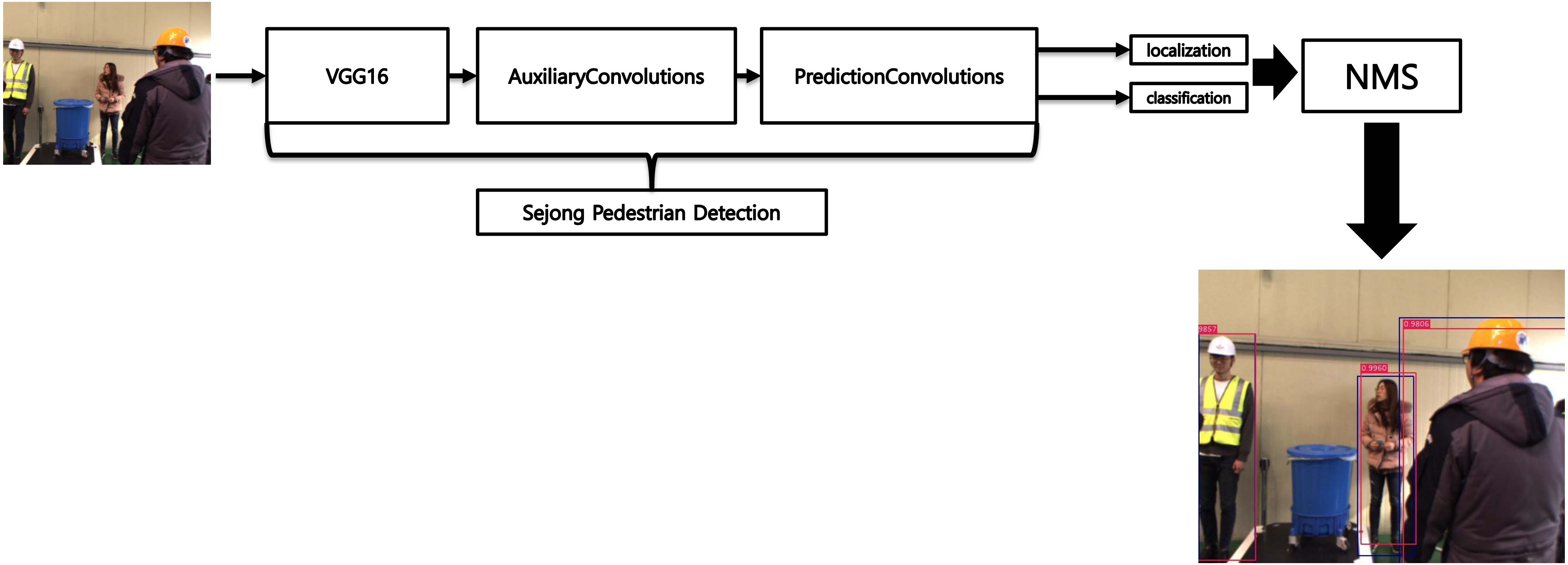
bbox 220 400 280 600
Label person
Occlusion 2
…



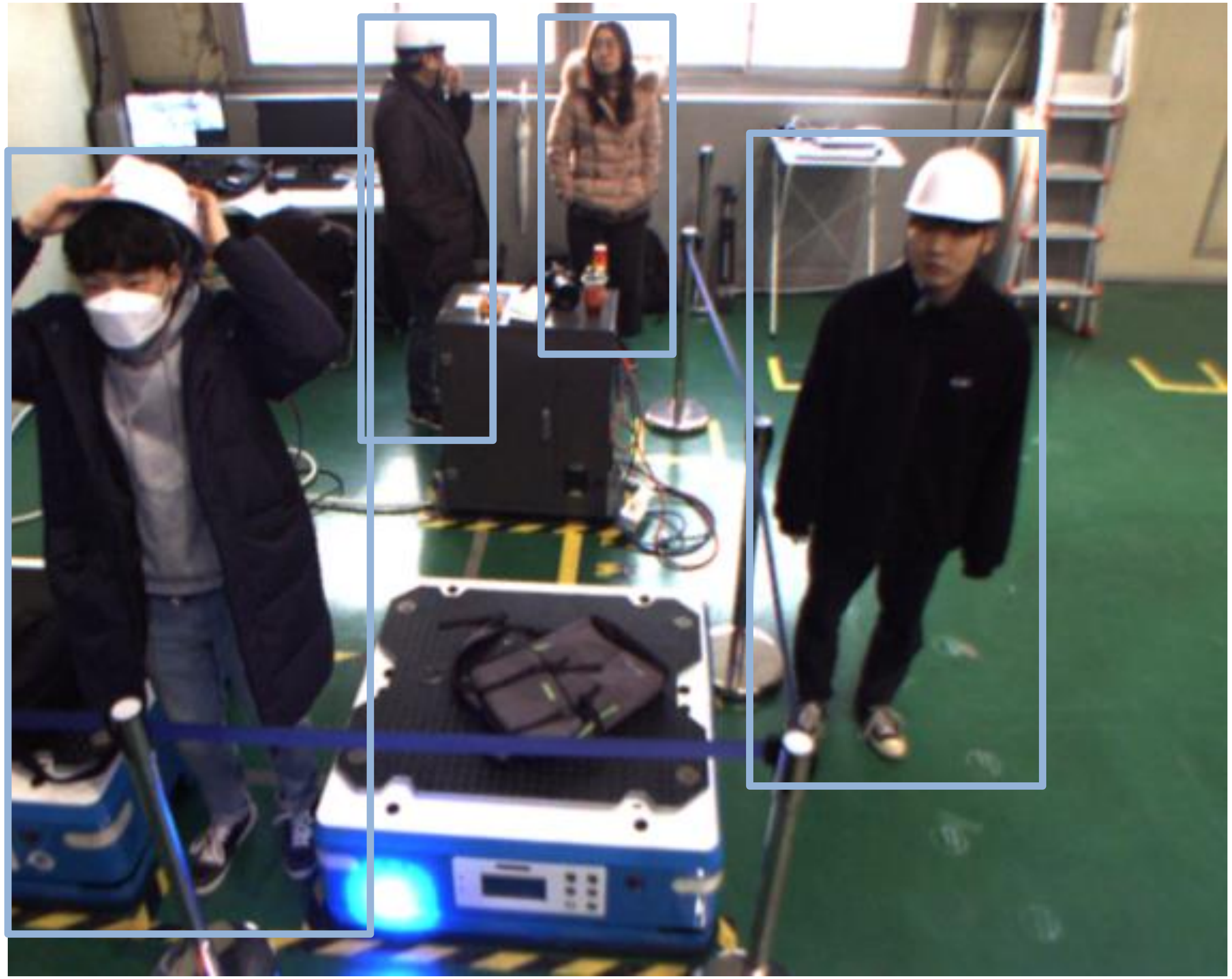


Json, xml, yaml ···

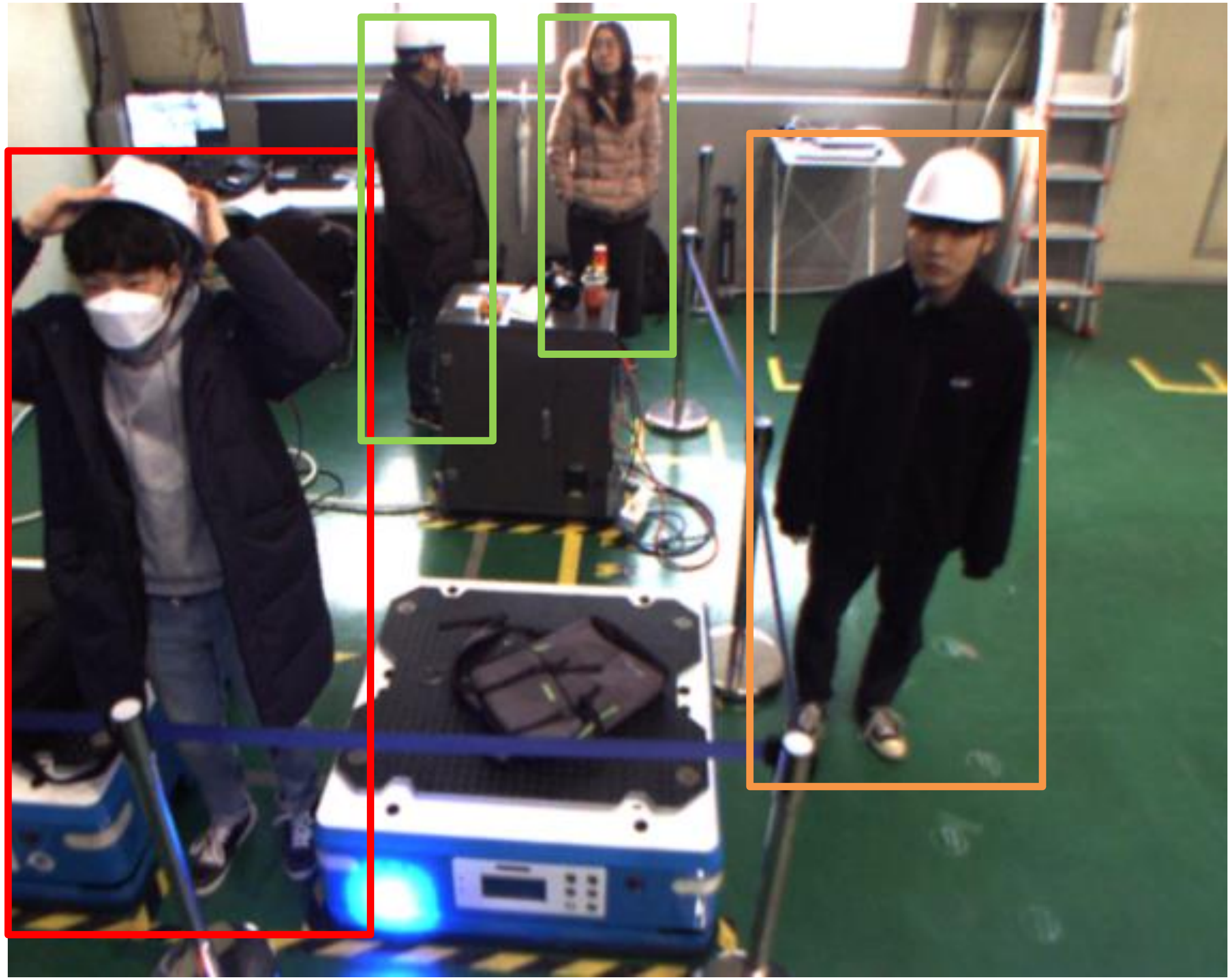

VGG16
AuxiliaryConvolutions
PredictionConvolutions
localization
classification
Sejong Pedestrian Detection
MultiBox Loss


VGG16
AuxiliaryConvolutions
PredictionConvolutions
localization
classification
NMS
Sejong Pedestrian Detection
9857
0.9806
0.9960

2.5D Object Detection (Multi Class)



1~2m	2~3m	3~4m	4~5m	5m~
0	0	0	0	0

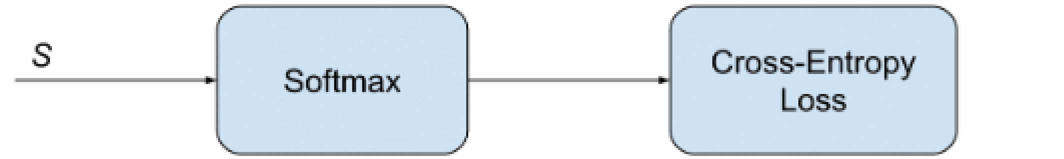


	1~2m	2~3m	3~4m	4~5m	5m~
(green)	0	0	0	0	1
(red)	1	0	0	0	0
(orange)	0	0	1	0	0

3.2) Categorical Cross-Entropy Loss

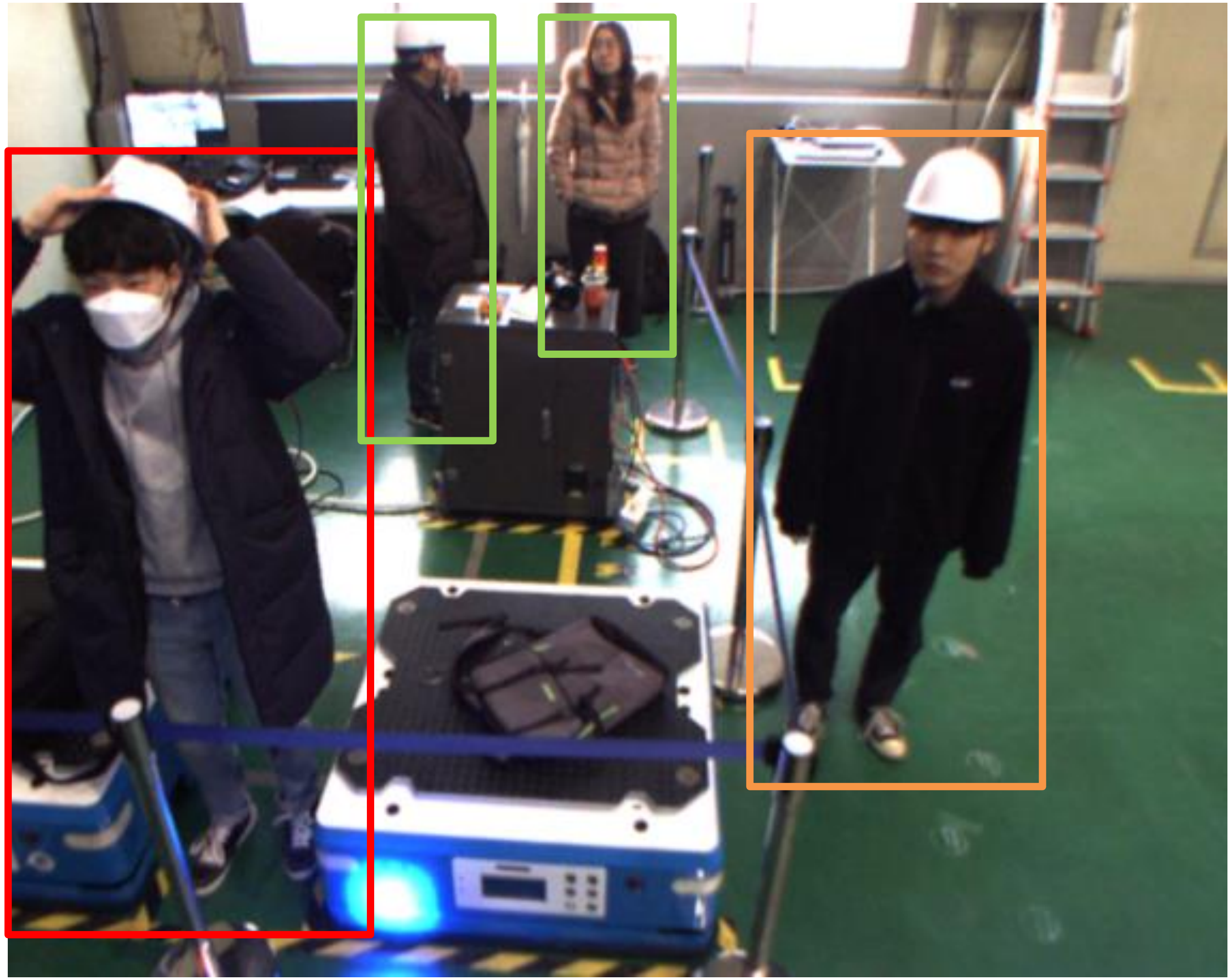
Softmax activation 뒤에 Cross-Entropy loss를 붙인 형태로 주로 사용하기 때문에 Softmax loss 라고도 불립니다.
→ **Multi-class classification에 사용됩니다.**

우리가 분류문제에서 주로 사용하는 활성화함수와 로스입니다. 분류 문제에서는 MSE(mean square error) loss 보다 CE loss가 더 빨리 수렴한 다는 사실이 알려져있습니다. 따라서 multi class에서 하나의 클래스를 구분할 때 softmax와 CE loss의 조합을 많이 사용합니다.



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

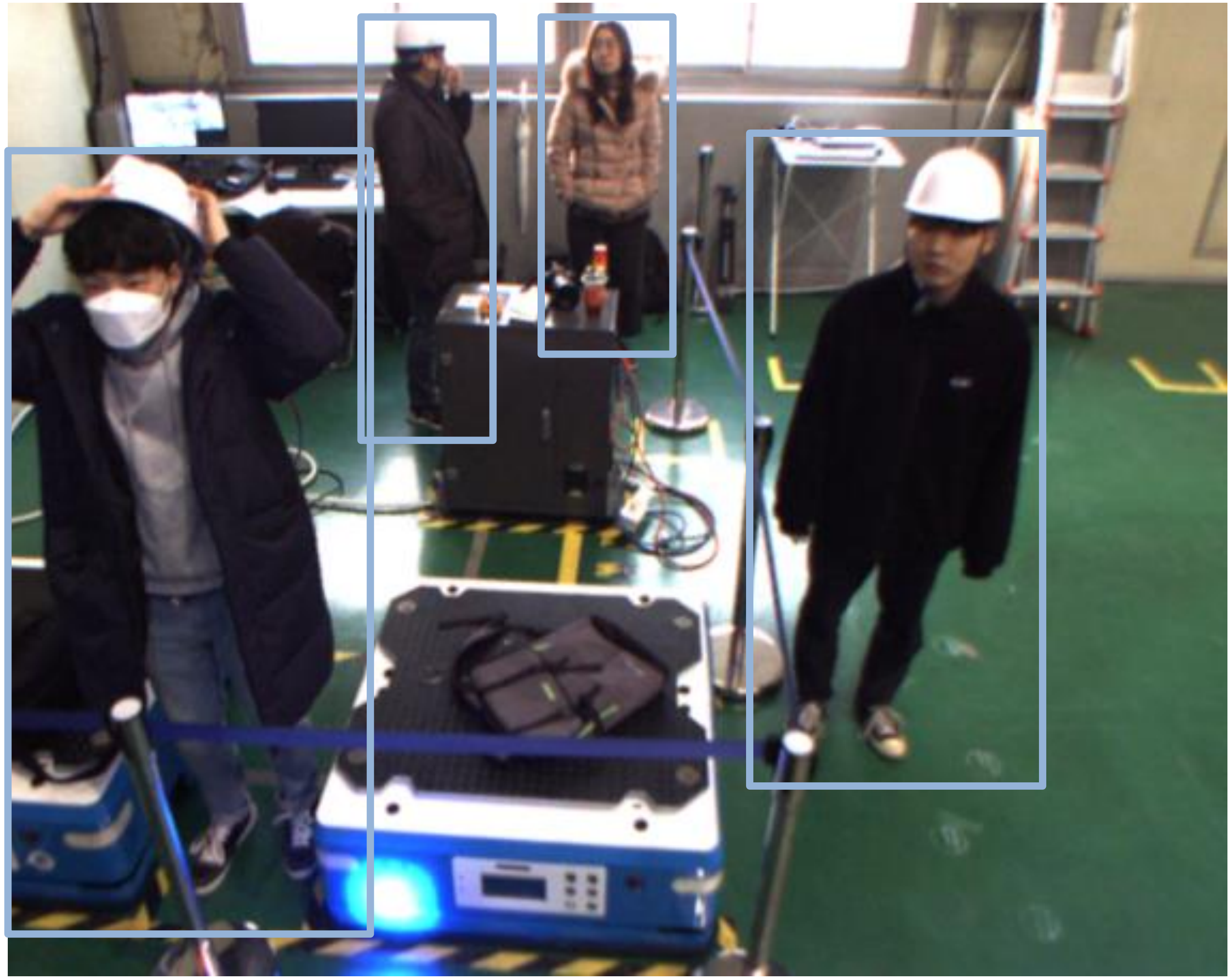
https://gombru.github.io/2018/05/23/cross_entropy_loss/
https://ratsgo.github.io/deep%20learning/2017/09/24/loss/



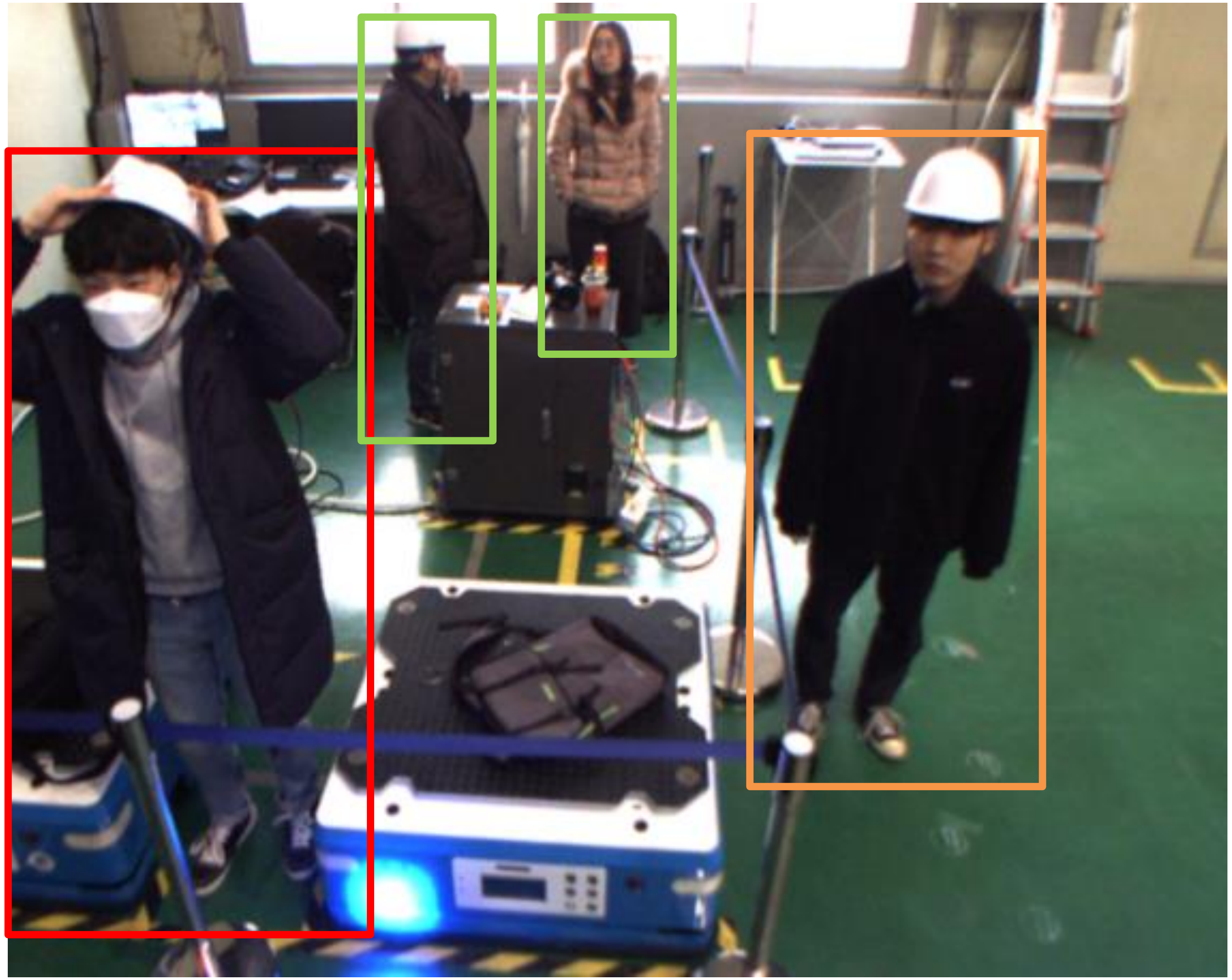
	1~2m	2~3m	3~4m	4~5m	5m~
	0	0	0	0.2	0.8
	0.7	0.3	0	0	0
	0	0.4	0.5	0.1	0

2.5D Object Detection (Multi Label)

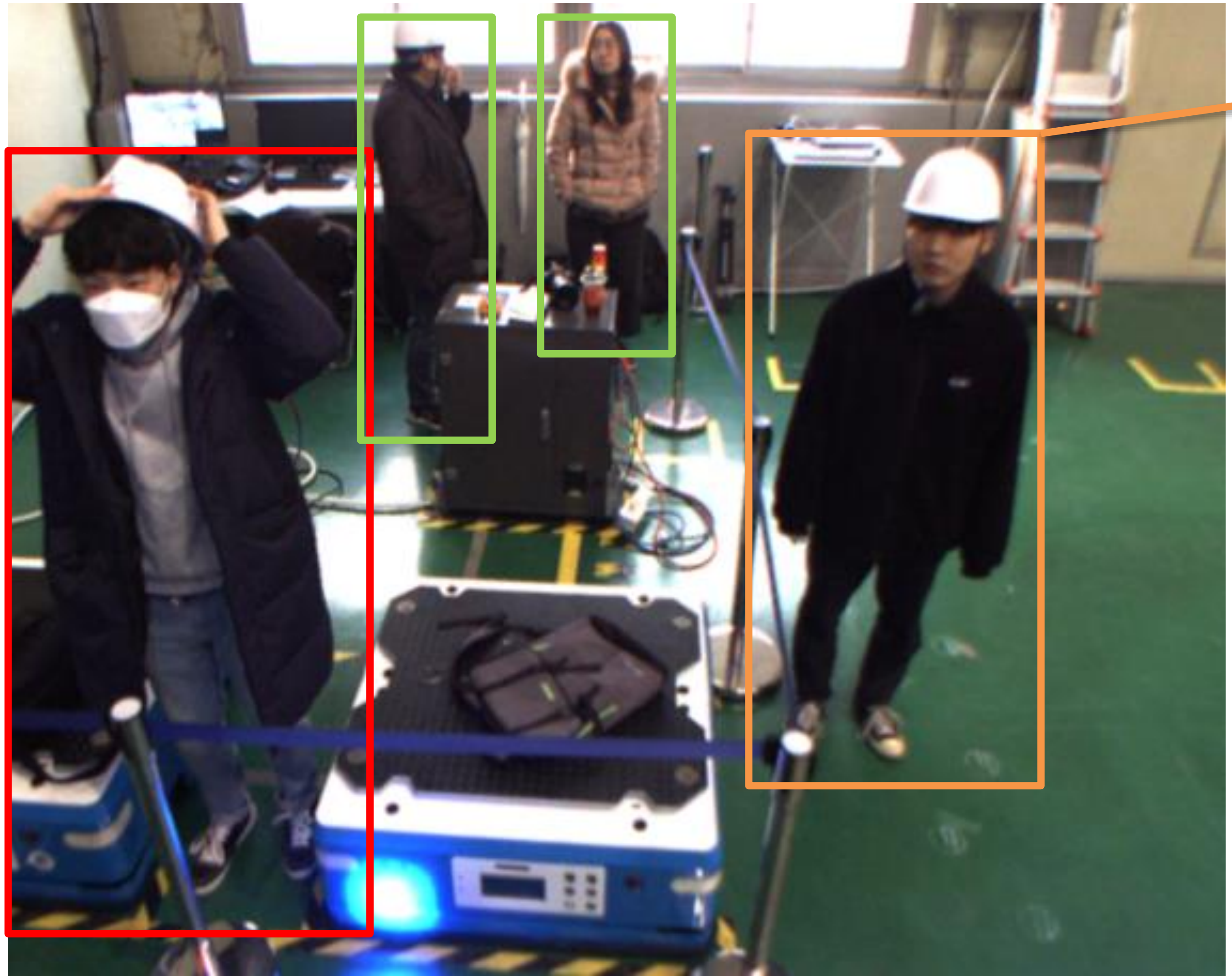
https://wordbe.tistory.com/entry/ML-Cross-entropyCategorical-Binary%EC%9D%98-%EC%9D%B4%ED%95%B4



1~2m	2~3m	3~4m	4~5m	5m~
0	0	0	0	0

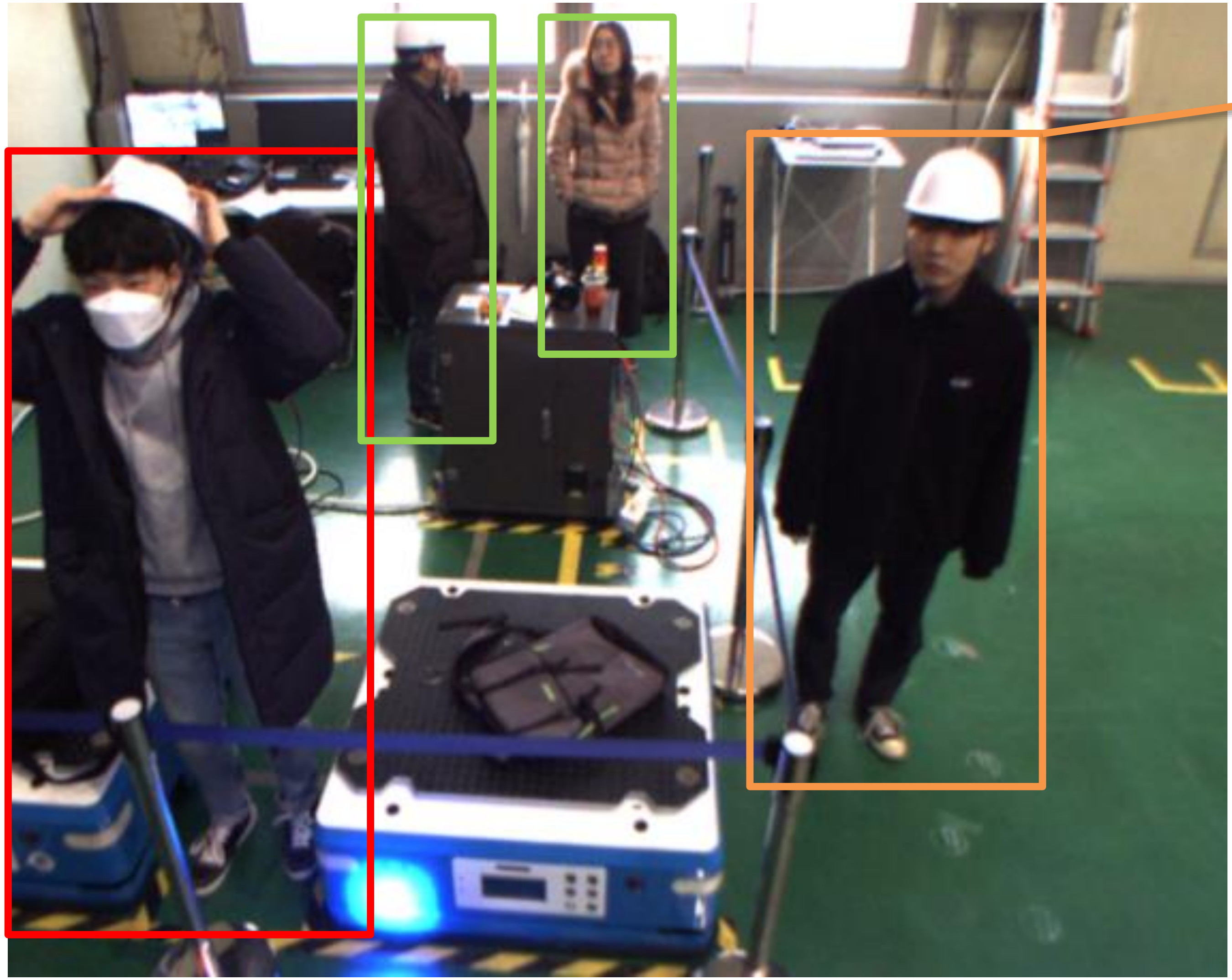


	1~2m	2~3m	3~4m	4~5m	5m~
—	0	0	0	0	1
—	1	0	0	0	0
—	0	0	1	0	0



3.01 ?

	1~2m	2~3m	3~4m	4~5m	5m~
green	0	0	0	0	1
red	1	0	0	0	0
orange	0	0	1	0	0



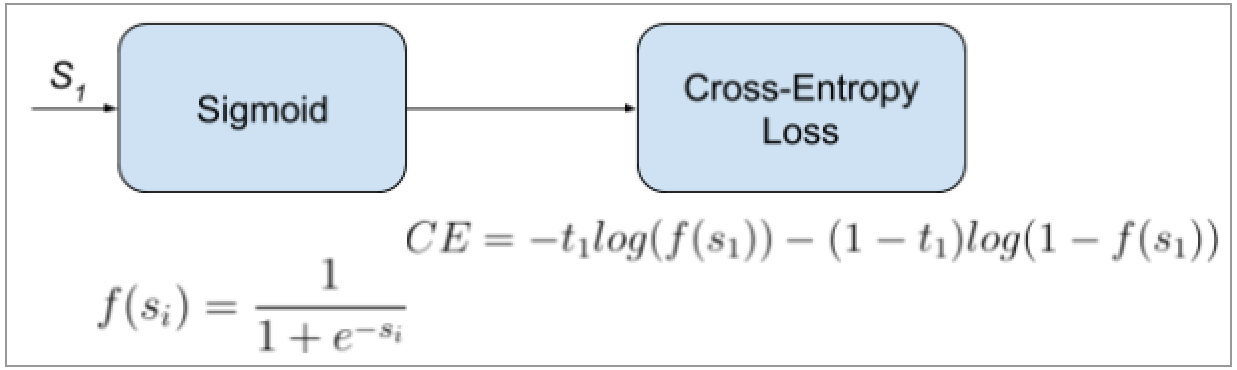
3.01 ?

	1~2m	2~3m	3~4m	4~5m	5m~
	0	0	0	0	1
	1	0	0	0	0
	0	1	1	0	0

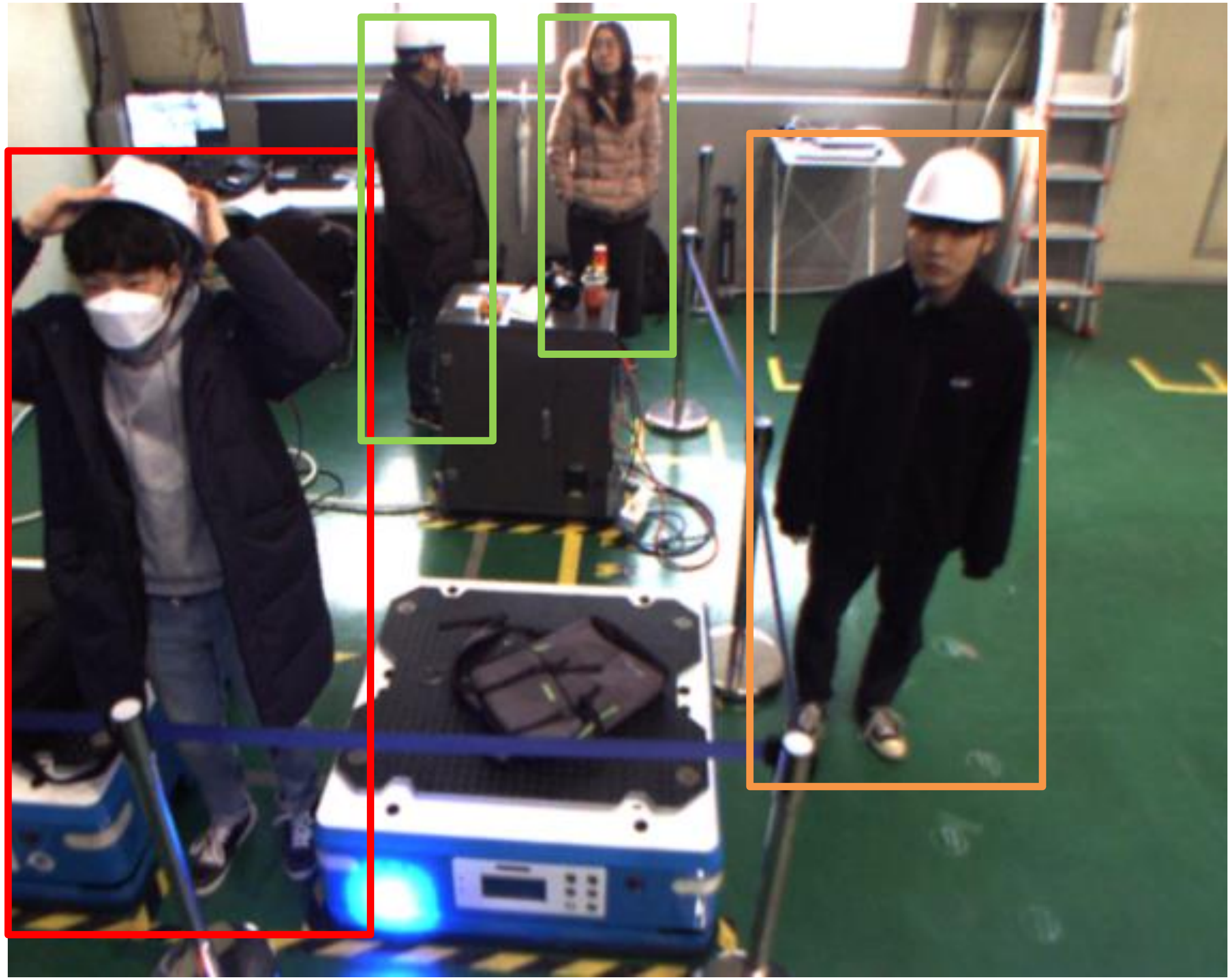
3.3) Binary Cross-Entropy Loss

Sigmoid activation 뒤에 Cross-Entropy loss를 붙인 형태로 주로 사용하기 때문에 Sigmoid CE loss라고도 불립니다.
→ **Multi-label classification에 사용됩니다.**

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$



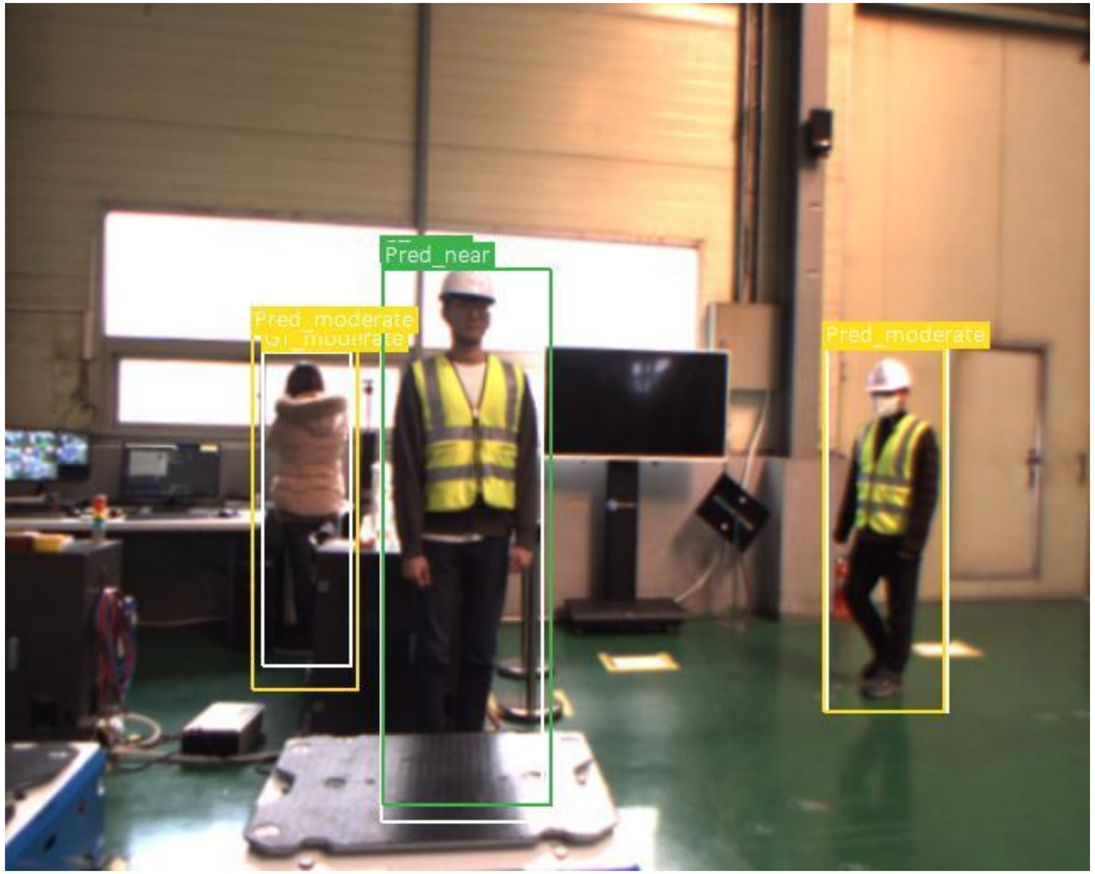
https://gombru.github.io/2018/05/23/cross_entropy_loss/
https://ratsgo.github.io/deep%20learning/2017/09/24/loss/

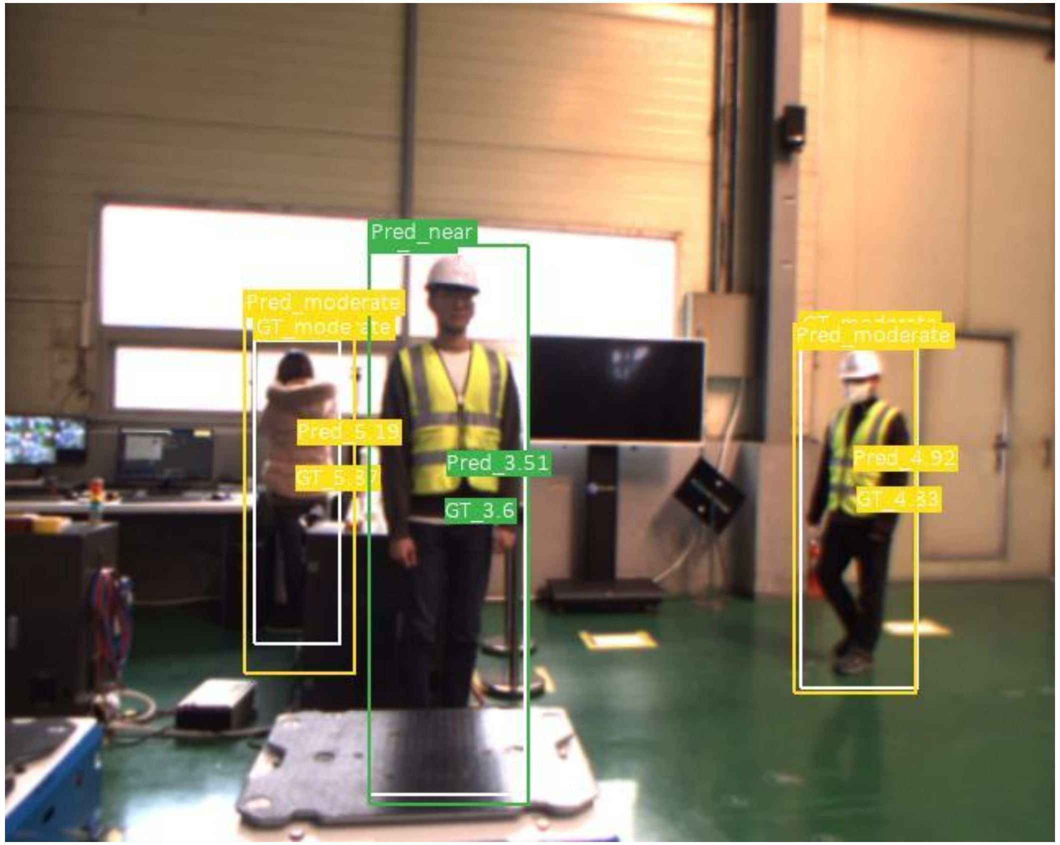


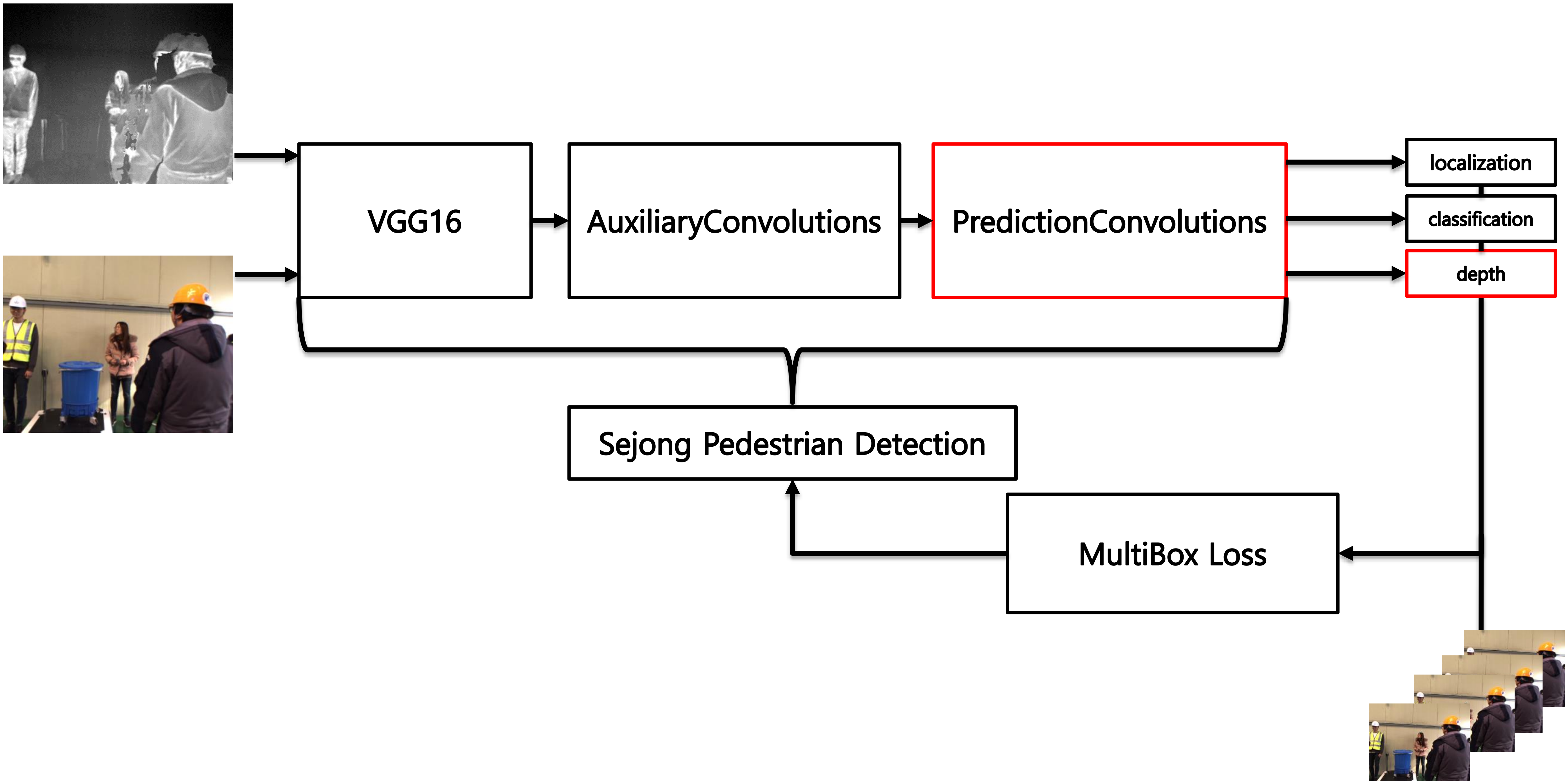
1~2m	2~3m	3~4m	4~5m	5m~
0	0	0	0.2	0.8
0.7	0.3	0	0	0
0	0.4	0.5	0.1	0

1~2m	2~3m	3~4m	4~5m	5m~
-0.5	0.01	0.01	0.01	0.99
0.89	0.01	-0.5	0.01	0.01
0.01	0.89	0.97	0.01	-0.5

2.5D Object Detection (Depth Regression)


Pred_near
Pred_moderate
GT_moderate
Pred_moderate


Pred_near
Pred_moderate
Pred_5.19
Pred_3.51
GT_3.6
Pred_moderate
Pred_4.92

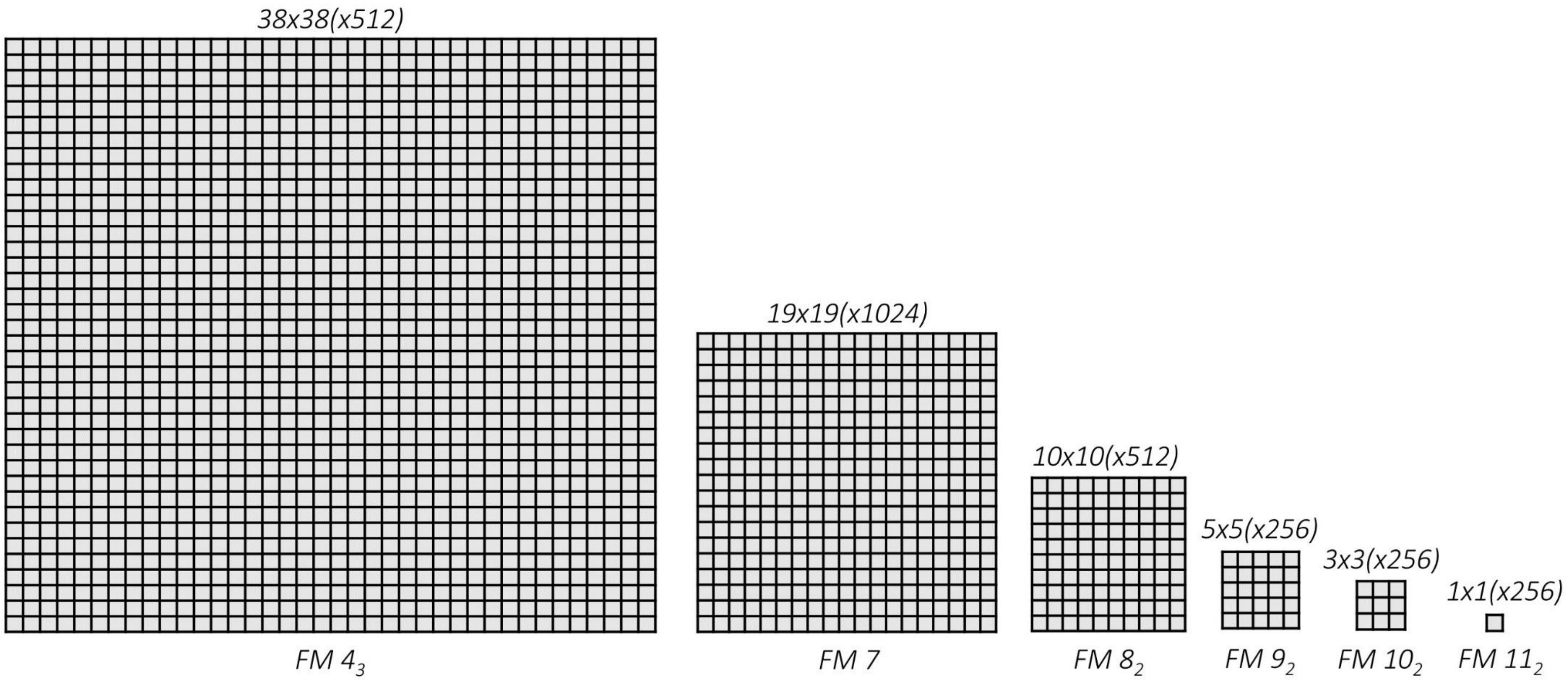

VGG16
AuxiliaryConvolutions
PredictionConvolutions
localization
classification
depth
Sejong Pedestrian Detection
MultiBox Loss

Prediction Convolutions

```python
class PredictionConvolutions(nn.Module):

    def __init__(self, n_classes):

        super(PredictionConvolutions, self).__init__()

        self.n_classes = n_classes

        n_boxes = {'conv4_3': 6,
                   'conv6': 6,
                   'conv7': 6,
                   'conv8': 6,
                   'conv9': 6,
                   'conv10': 6,}

        self.loc_conv4_3 = nn.Conv2d(512, n_boxes['conv4_3'] * 4, kernel_size=3, padding=1)
        self.loc_conv6 = nn.Conv2d(512, n_boxes['conv6'] * 4, kernel_size=3, padding=1)
        self.loc_conv7 = nn.Conv2d(512, n_boxes['conv6'] * 4, kernel_size=3, padding=1)
        self.loc_conv8 = nn.Conv2d(512, n_boxes['conv7'] * 4, kernel_size=3, padding=1)
        self.loc_conv9 = nn.Conv2d(512, n_boxes['conv8'] * 4, kernel_size=3, padding=1)
        self.loc_conv10 = nn.Conv2d(512, n_boxes['conv9'] * 4, kernel_size=3, padding=1)

        self.cl_conv4_3 = nn.Conv2d(512, n_boxes['conv4_3'] * n_classes, kernel_size=3, padding=1)
        self.cl_conv6 = nn.Conv2d(512, n_boxes['conv6'] * n_classes, kernel_size=3, padding=1)
        self.cl_conv7 = nn.Conv2d(512, n_boxes['conv6'] * n_classes, kernel_size=3, padding=1)
        self.cl_conv8 = nn.Conv2d(512, n_boxes['conv7'] * n_classes, kernel_size=3, padding=1)
        self.cl_conv9 = nn.Conv2d(512, n_boxes['conv8'] * n_classes, kernel_size=3, padding=1)
        self.cl_conv10 = nn.Conv2d(512, n_boxes['conv9'] * n_classes, kernel_size=3, padding=1)

        self.init_conv2d()
```

```python
def forward(self, conv4_3_feats, conv6_feats, conv7_feats, conv8_feats, conv9_feats, conv10_feats):

    batch_size = conv4_3_feats.size(0)

    l_conv4_3 = self.loc_conv4_3(conv4_3_feats)
    l_conv4_3 = l_conv4_3.permute(0, 2, 3, 1).contiguous()
    l_conv4_3 = l_conv4_3.view(batch_size, -1, 4)

    l_conv6 = self.loc_conv6(conv6_feats)
    l_conv6 = l_conv6.permute(0, 2, 3, 1).contiguous()
    l_conv6 = l_conv6.view(batch_size, -1, 4)

    l_conv7 = self.loc_conv7(conv7_feats)
    l_conv7 = l_conv7.permute(0, 2, 3, 1).contiguous()
    l_conv7 = l_conv7.view(batch_size, -1, 4)

    l_conv8 = self.loc_conv8(conv8_feats)
    l_conv8 = l_conv8.permute(0, 2, 3, 1).contiguous()
    l_conv8 = l_conv8.view(batch_size, -1, 4)

    l_conv9 = self.loc_conv9(conv9_feats)
    l_conv9 = l_conv9.permute(0, 2, 3, 1).contiguous()
    l_conv9 = l_conv9.view(batch_size, -1, 4)

    l_conv10 = self.loc_conv10(conv10_feats)
    l_conv10 = l_conv10.permute(0, 2, 3, 1).contiguous()
    l_conv10 = l_conv10.view(batch_size, -1, 4)
```

Prediction Convolutions



Prediction Convolutions

KERNEL 3X3 | PAD 1 | CH __ → number of priors per location for this feature map * 4	KERNEL 3X3 | PAD 1 | CH __ → number of priors per location for this feature map * number of object classes

Prediction Convolutions

```
self.loc_conv4_3 = nn.Conv2d(512, n_boxes['conv4_3'] * 4, kernel_size=3, padding=1)
self.cl_conv4_3 = nn.Conv2d(512, n_boxes['conv4_3'] * n_classes, kernel_size=3, padding=1)
```

```
(Pdb) conv4_3_feats.shape
torch.Size([8, 512, 64, 80])
```

```
l_conv4_3 = self.loc_conv4_3(conv4_3_feats)
```

```
(Pdb) l_conv4_3.shape
torch.Size([8, 24, 64, 80])
```

```
l_conv4_3 = l_conv4_3.permute(0, 2, 3, 1).contiguous()
```

```
(Pdb) l_conv4_3.shape
torch.Size([8, 64, 80, 24])
```

```
l_conv4_3 = l_conv4_3.view(batch_size, -1, 4)
```

```
(Pdb) l_conv4_3.shape
torch.Size([8, 30720, 4])
```

```
(Pdb) conv4_3_feats.shape
torch.Size([8, 512, 64, 80])
```

```
c_conv4_3 = self.cl_conv4_3(conv4_3_feats)
```

```
(Pdb) c_conv4_3.shape
torch.Size([8, 12, 64, 80])
```

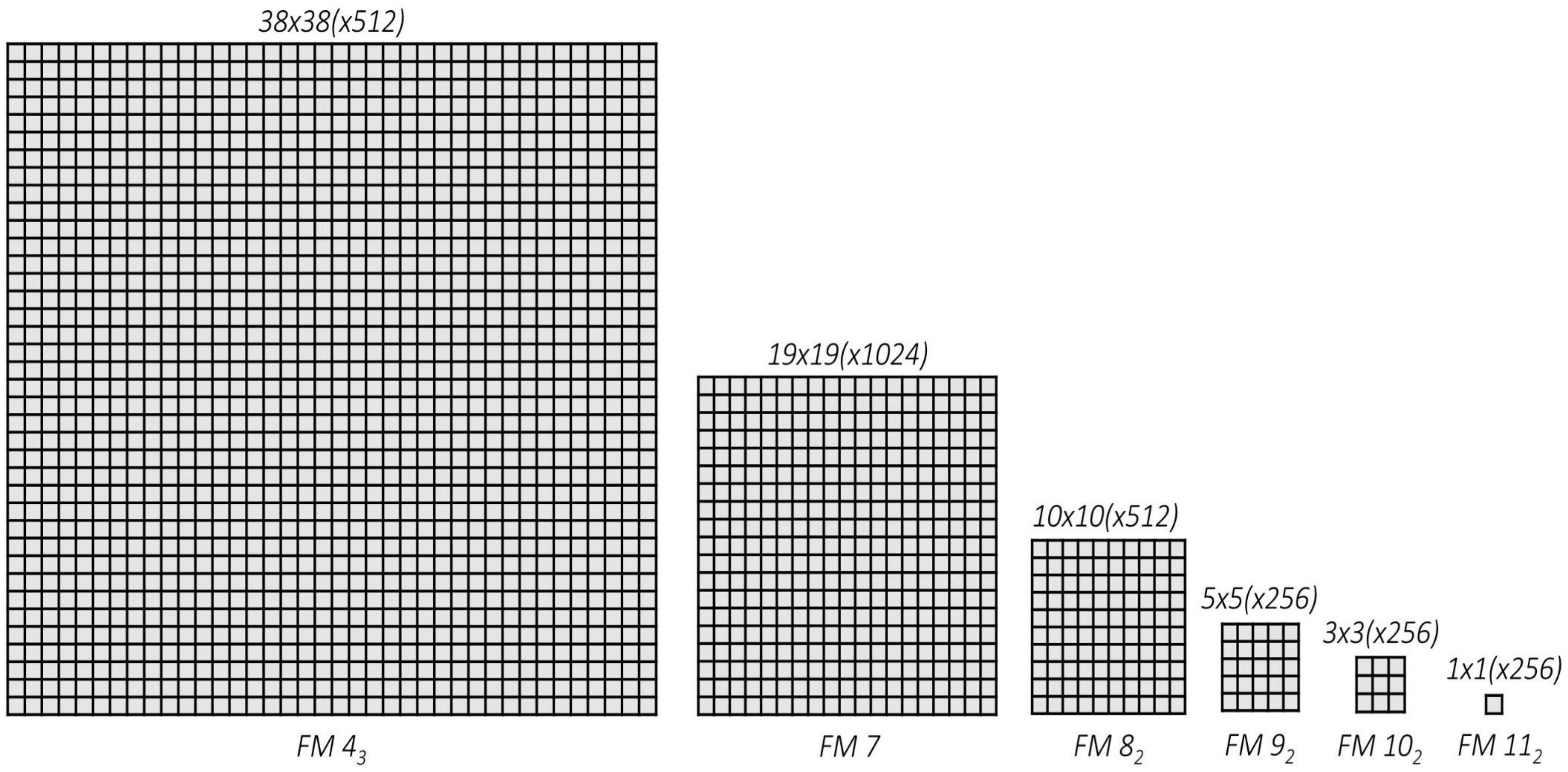
```
c_conv4_3 = c_conv4_3.permute(0, 2, 3, 1).contiguous()
```

```
(Pdb) c_conv4_3.shape
torch.Size([8, 64, 80, 12])
```

```
c_conv4_3 = c_conv4_3.view(batch_size, -1, self.n_classes)
```

```
(Pdb) c_conv4_3.shape
torch.Size([8, 30720, 2])
```

Prediction Convolutions



Prediction Convolutions

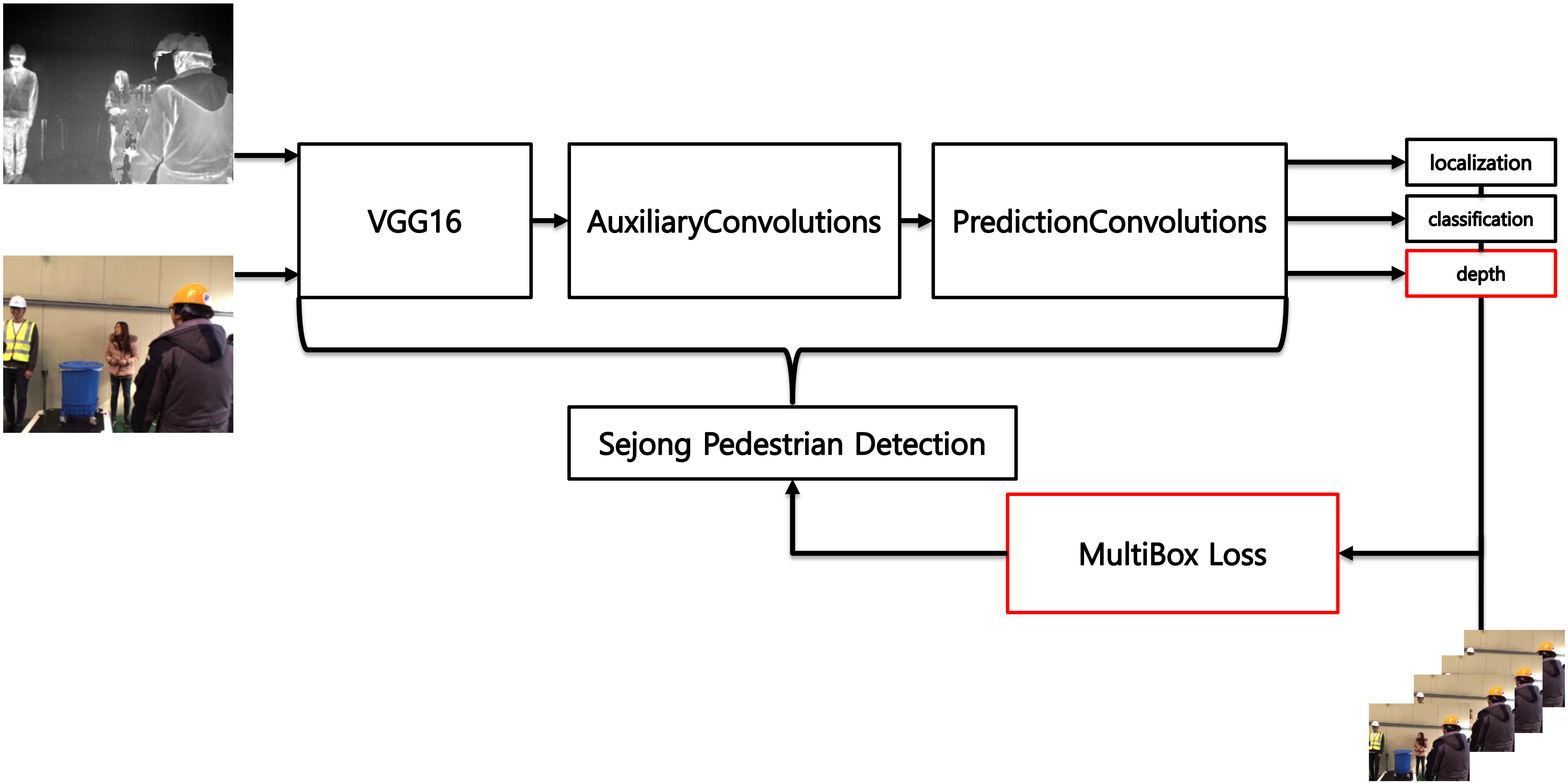
KERNEL 3X3 | PAD 1 |
CH __ → number of priors per location for this feature map * 4

KERNEL 3X3 | PAD 1 |
CH __ → number of priors per location for this feature map * number of object classes

KERNEL 3X3 | PAD 1 |
CH __ → Number of priors per location for this feature map * 1

Prediction Convolutions

```
# Predict depths
d_conv4_3 = self.dp_conv4_3(conv4_3_feats)
d_conv4_3 = d_conv4_3.permute(0, 2, 3, 1).contiguous()
d_conv4_3 = d_conv4_3.view(batch_size, -1)

d_conv6 = self.dp_conv6(conv6_feats)
d_conv6 = d_conv6.permute(0, 2, 3, 1).contiguous()
d_conv6 = d_conv6.view(batch_size, -1)

d_conv7 = self.dp_conv7(conv7_feats)
d_conv7 = d_conv7.permute(0, 2, 3, 1).contiguous()
d_conv7 = d_conv7.view(batch_size, -1)

d_conv8 = self.dp_conv8(conv8_feats)
d_conv8 = d_conv8.permute(0, 2, 3, 1).contiguous()
d_conv8 = d_conv8.view(batch_size, -1)

d_conv9 = self.dp_conv9(conv9_feats)
d_conv9 = d_conv9.permute(0, 2, 3, 1).contiguous()
d_conv9 = d_conv9.view(batch_size, -1)

d_conv10 = self.dp_conv10(conv10_feats)
d_conv10 = d_conv10.permute(0, 2, 3, 1).contiguous()
d_conv10 = d_conv10.view(batch_size, -1)
```


VGG16
AuxiliaryConvolutions
PredictionConvolutions
localization
classification
depth
Sejong Pedestrian Detection
MultiBox Loss

MultiBox Loss

```python
class MultiBoxLoss(nn.Module):

    def __init__(self, priors_cxcy, threshold=0.5, neg_pos_ratio=4, alpha=1., beta=1.):
        super(MultiBoxLoss, self).__init__()
        self.priors_cxcy = priors_cxcy
        self.priors_xy = cxcy_to_xy(priors_cxcy)
        self.threshold = threshold
        self.neg_pos_ratio = neg_pos_ratio
        self.alpha = alpha
        self.beta = beta

        self.smooth_l1 = nn.L1Loss()
        self.MSELoss = nn.MSELoss()
        self.cross_entropy = nn.CrossEntropyLoss(reduce=False, ignore_index=-1)
```

MultiBox Loss

```
def forward(self, predicted_locs, predicted_scores, predeicted_depths, boxes, labels, depths):

    batch_size = predicted_locs.size(0)
    n_priors = self.priors_cxcy.size(0)
    n_classes = predicted_scores.size(2)

    assert n_priors == predicted_locs.size(1) == predicted_scores.size(1) == predeicted_depths.size(1)

    true_locs = torch.zeros((batch_size, n_priors, 4), dtype=torch.float).to(device)
    true_classes = torch.zeros((batch_size, n_priors), dtype=torch.long).to(device)
    true_depths = torch.zeros((batch_size, n_priors), dtype=torch.float).to(device)
```

MultiBox Loss


```
for i in range(batch_size):

    n_objects = boxes[i].size(0)

    overlap = find_jaccard_overlap(boxes[i],self.priors_xy)

    overlap_for_each_prior, object_for_each_prior = overlap.max(dim=0)

    _, prior_for_each_object = overlap.max(dim=1)

    object_for_each_prior[prior_for_each_object] = torch.LongTensor(range(n_objects)).to(device)

    overlap_for_each_prior[prior_for_each_object] = 1.

    # Labels for each prior
    label_for_each_prior = labels[i][object_for_each_prior]
    dpeth_for_each_prior = depths[i][object_for_each_prior]

    label_for_each_prior[overlap_for_each_prior < self.threshold] = 0
    dpeth_for_each_prior[overlap_for_each_prior < self.threshold] = 0

    true_classes[i] = label_for_each_prior
    true_depths[i] = dpeth_for_each_prior

    true_locs[i] = cxcy_to_gcxgcy(xy_to_cxcy(boxes[i][object_for_each_prior]), self.priors_cxcy)
```

MultiBox Loss

```python
positive_priors = true_classes > 0  # (N, 8732)

# LOCALIZATION LOSS

if true_locs[positive_priors].shape[0] == 0:
    loc_loss = 0.
else:
    loc_loss = self.smooth_l1(predicted_locs[positive_priors], true_locs[positive_priors])

# DEPTHS LOSS
if true_depths[positive_priors].shape[0] == 0:
    depth_loss = 0.
else:
    depth_loss = self.smooth_l1(predeicted_depths[positive_priors], true_depths[positive_priors])
```

MultiBox Loss

```
# CONFIDENCE LOSS
n_positives = positive_priors.sum(dim=1)

n_hard_negatives = self.neg_pos_ratio * n_positives  # (N)

conf_loss_all = self.cross_entropy(predicted_scores.view(-1, n_classes), true_classes.view(-1))
conf_loss_all = conf_loss_all.view(batch_size, n_priors)

conf_loss_pos = conf_loss_all[positive_priors]

conf_loss_neg = conf_loss_all.clone()
conf_loss_neg[positive_priors] = 0.
conf_loss_neg, _ = conf_loss_neg.sort(dim=1, descending=True)
hardness_ranks = torch.LongTensor(range(n_priors)).unsqueeze(0).expand_as(conf_loss_neg).to(device)

hard_negatives = hardness_ranks < n_hard_negatives.unsqueeze(1)
conf_loss_hard_neg = conf_loss_neg[hard_negatives]

conf_loss = (conf_loss_hard_neg.sum() + conf_loss_pos.sum()) / ( 1e-10 + n_positives.sum().float() )

# TOTAL LOSS
return conf_loss + self.alpha * loc_loss + self.beta * depth_loss, conf_loss , loc_loss, depth_loss, n_positives
```