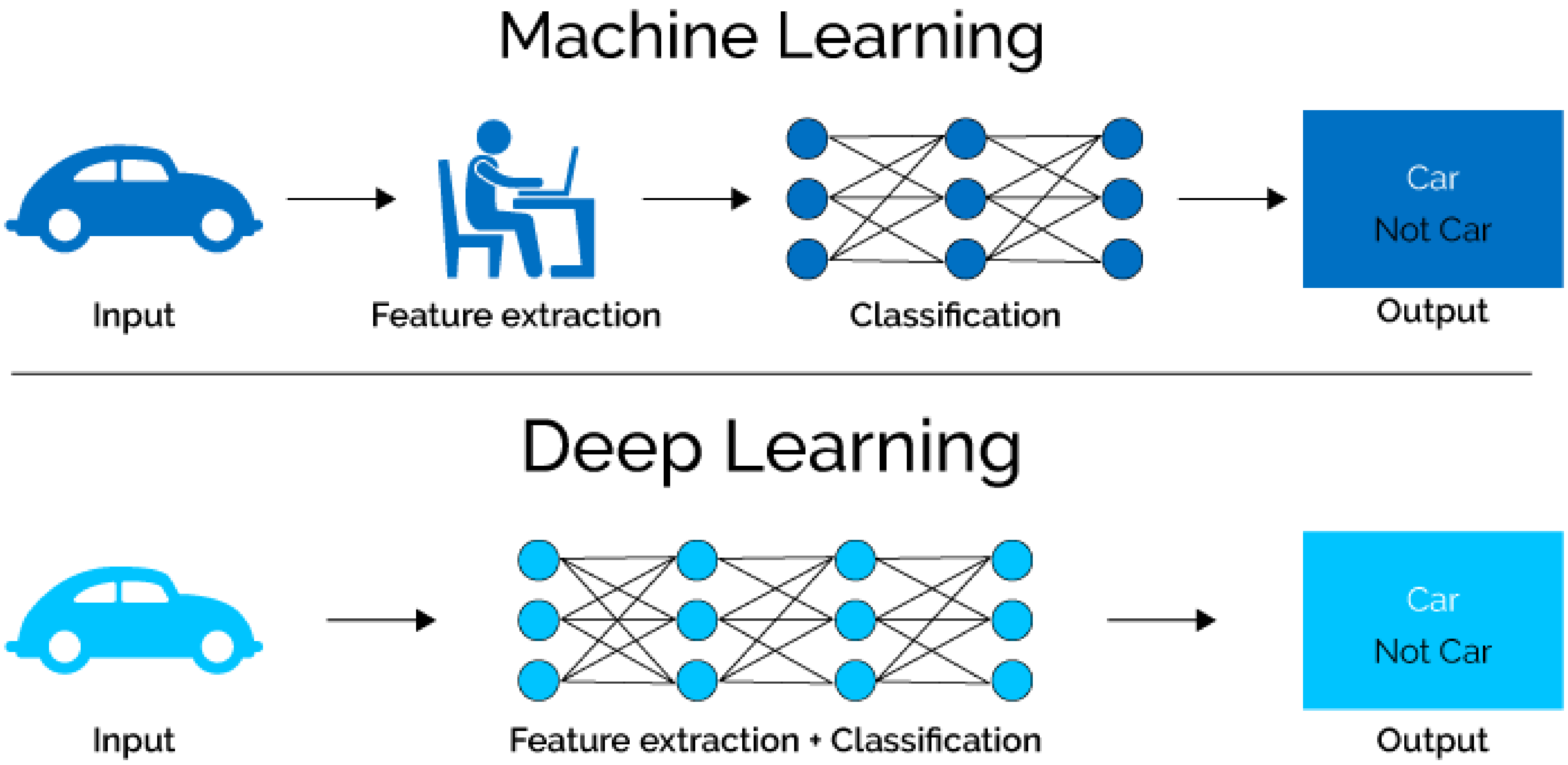
SEJONG RCV Winter School

- Object Detection(SSD) -

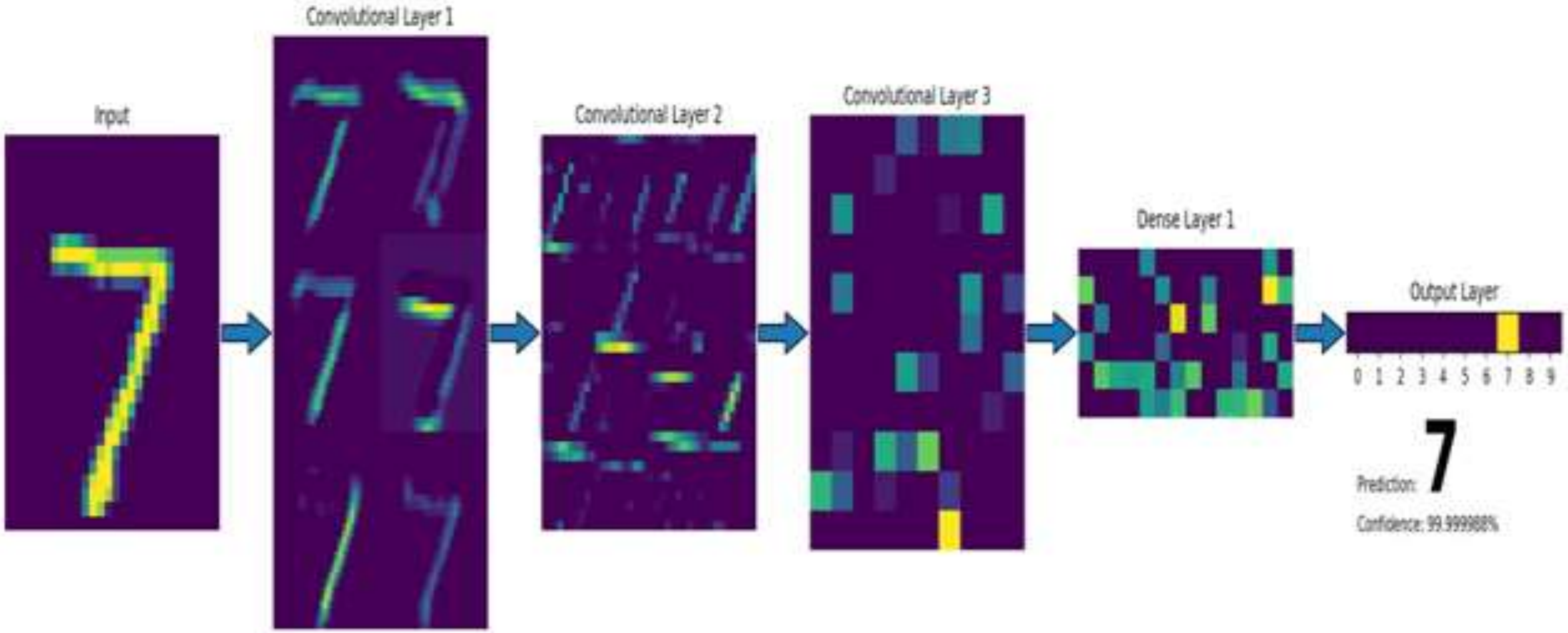
Object Detection

Classification



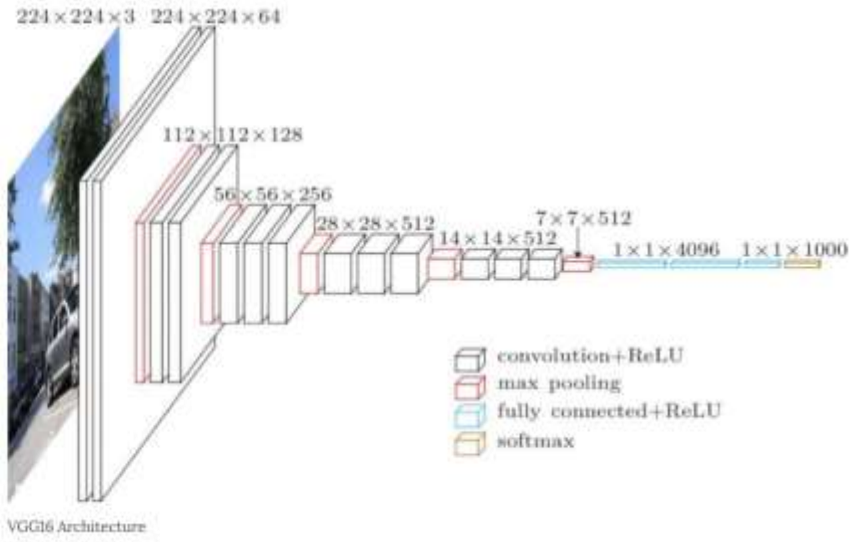
https://mc.ai/deep-learning-image-classification-example-in-matlab/

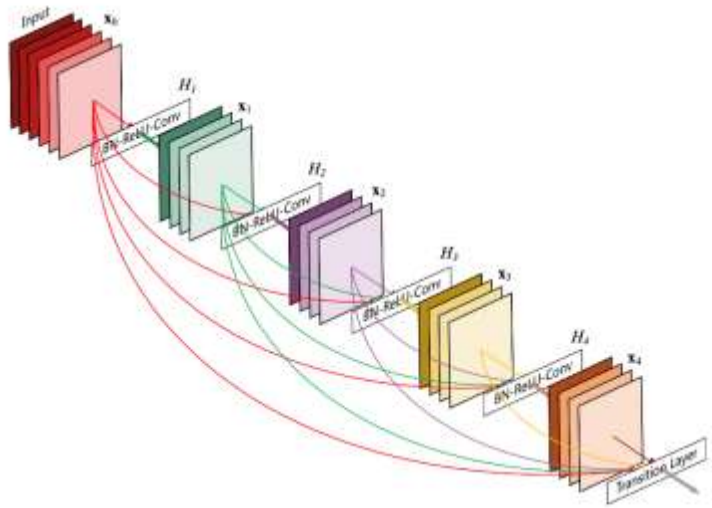
Classification

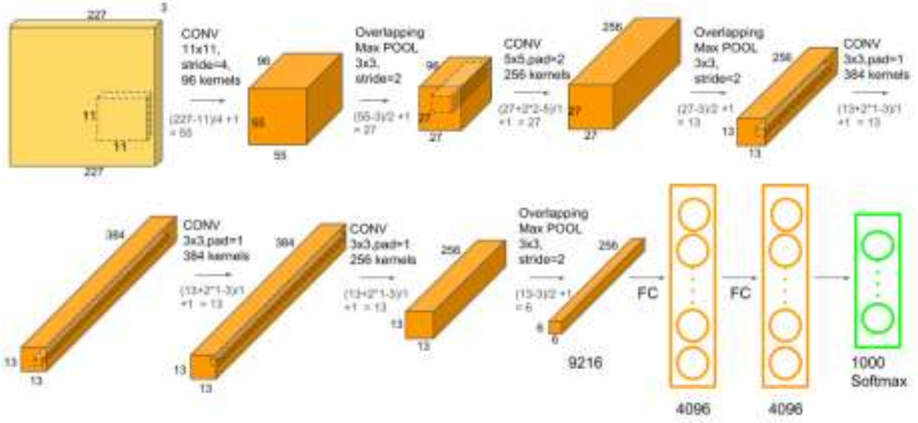


https://gfycat.com/ko/helpfulscratchyarcticseal-machine-learning-neural-networks-data-science

Classification





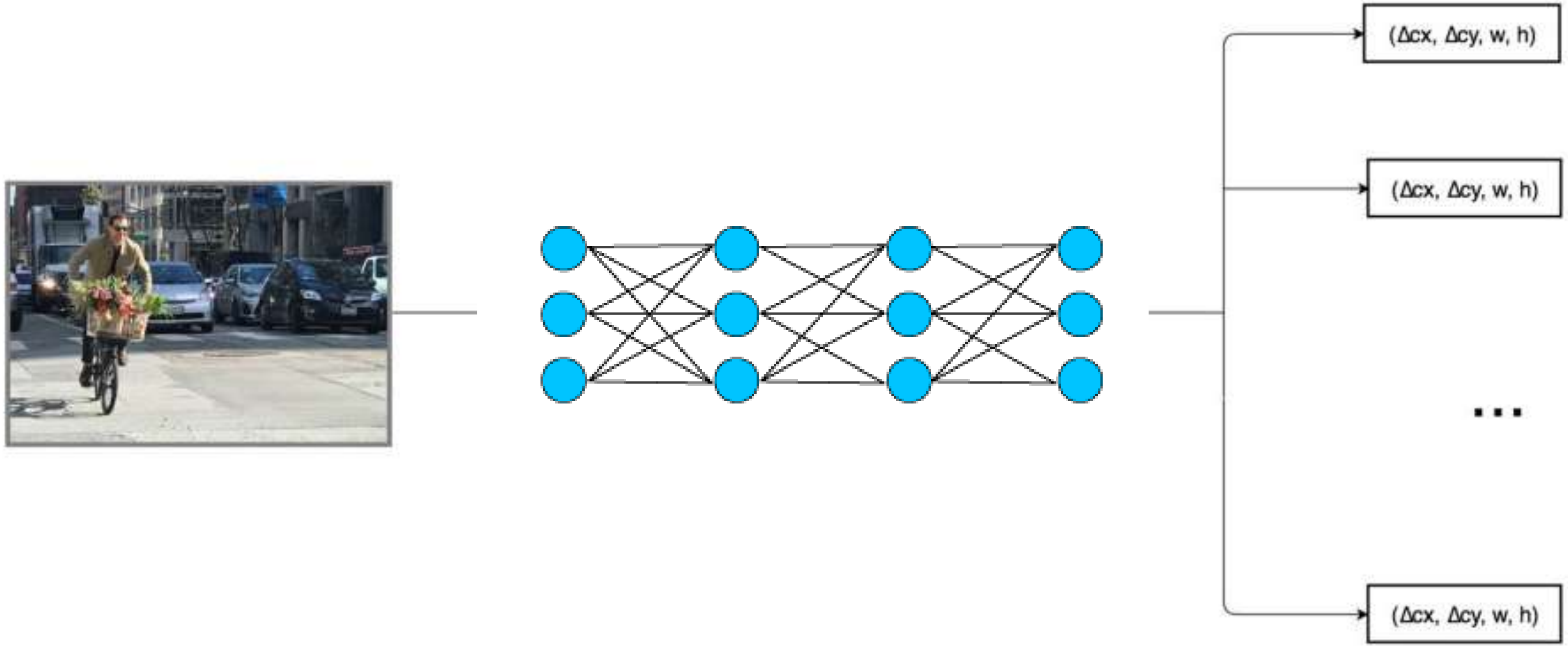


VGG16Net

ResNet

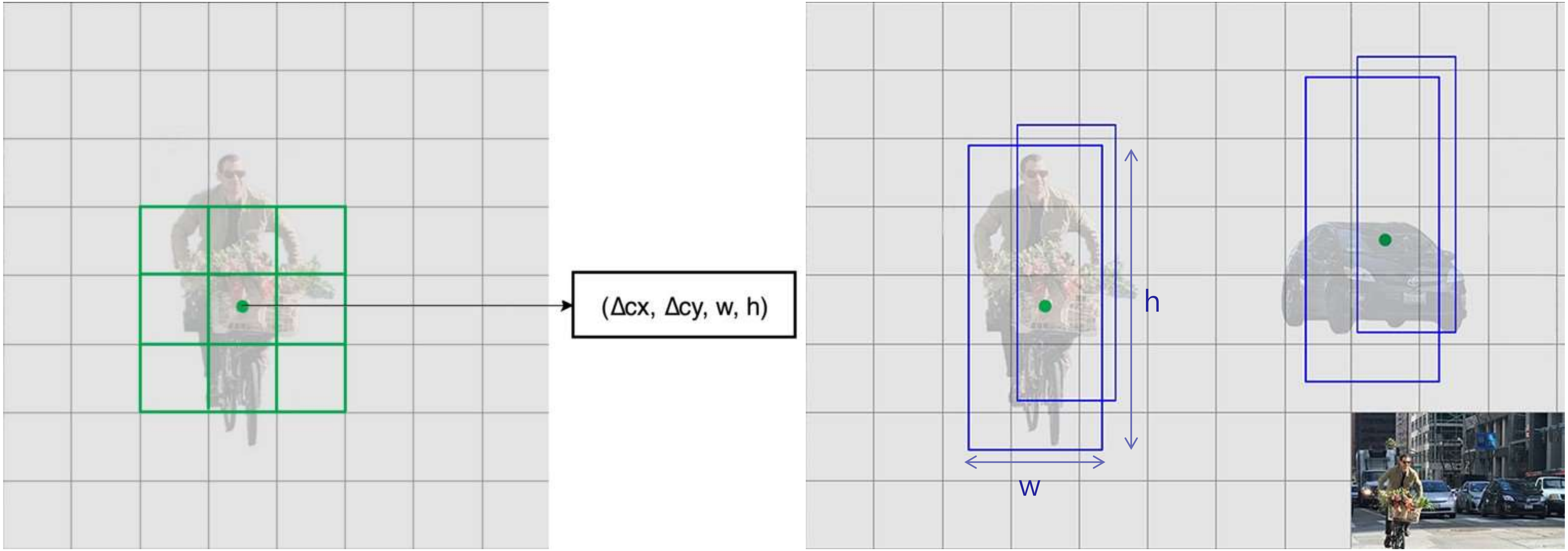
AlexNet

Localization



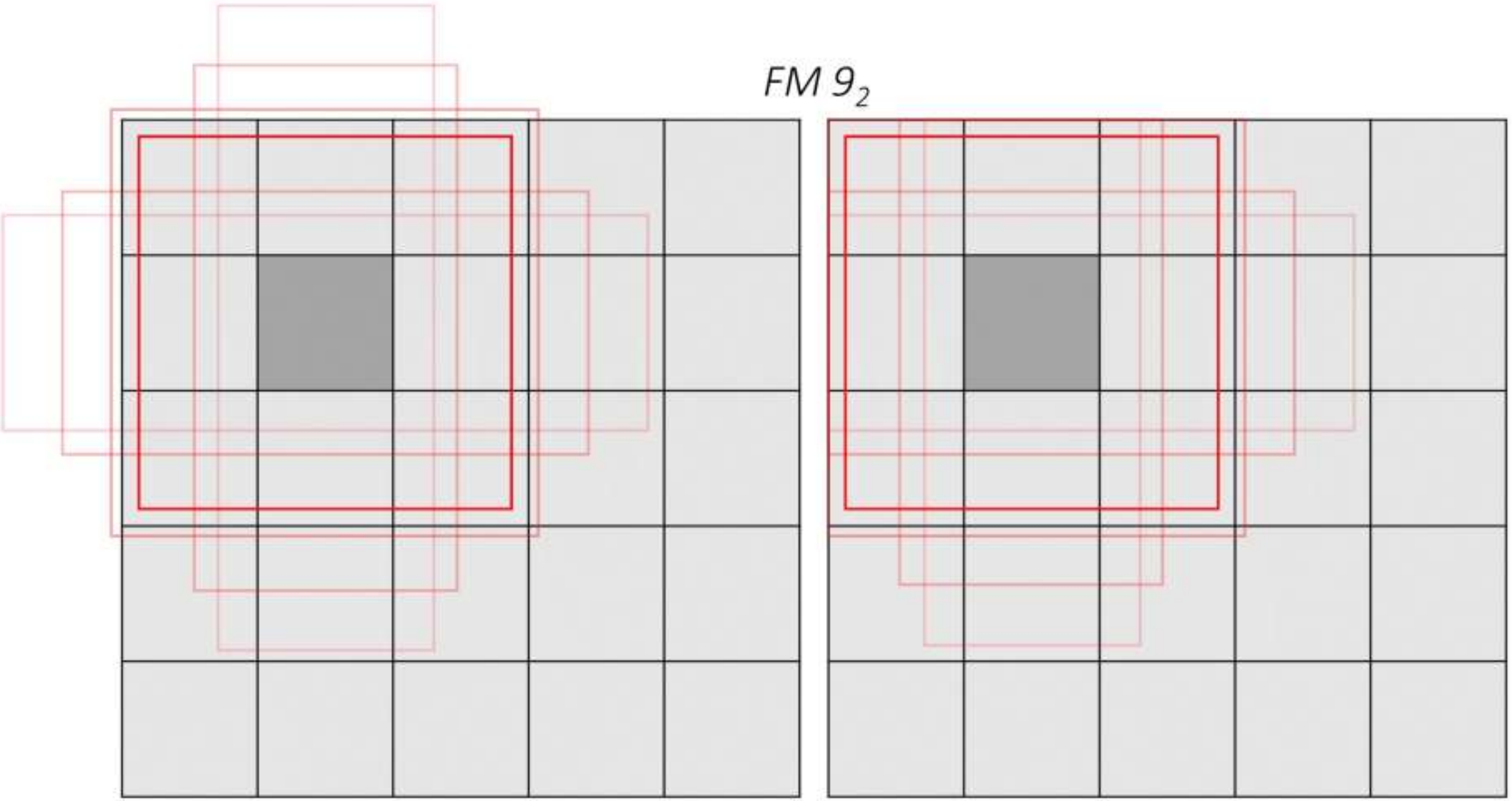
https://mc.ai/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing/

Localization



https://mc.ai/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing/

Localization



When priors at a location overshoot the edges of the feature map, they are clipped

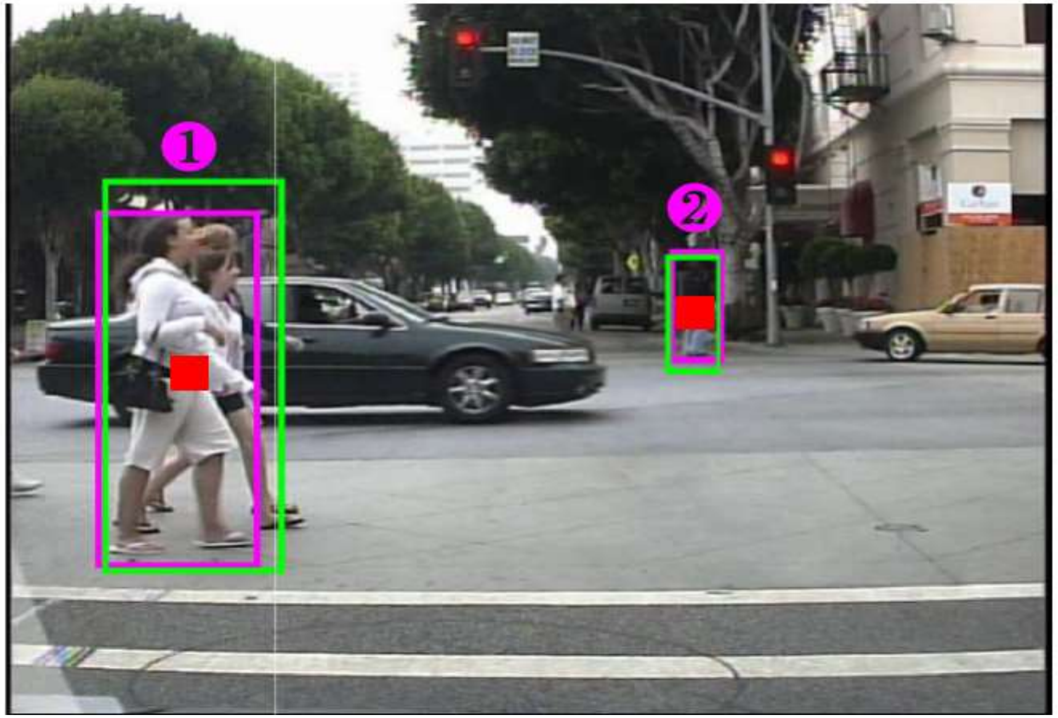
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

R-CNN: Pipeline

$$t_x = (G_x - P_x)/P_w$$
$$t_y = (G_y - P_y)/P_h$$
$$t_w = \log(G_w/P_w)$$
$$t_h = \log(G_h/P_h)$$

R-CNN in detail

- Bounding box regression*
  - Proposal $P = (P_x, P_y, P_w, P_h)$ / Ground-truth $G = (G_x, G_y, G_w, G_h)$
  - **For box-scale invariance**, regression target t for the training pair (P, G)



① $G = (57.4, 141.0, 97.2, 237.0)$
$P = (61.6, 120.0, 107.3, 261.7)$

$$t_x = (57.4 - 61.6)/107.3 = -0.039$$
$$t_y = (141.0 - 120.0)/261.7 = 0.080$$
$$t_w = \log(97.2/107.3) = -0.043$$
$$t_h = \log(237.0/261.7) = -0.043$$

② $G = (408.0, 167.0, 29.9, 73.0)$
$P = (405.8, 170.8, 31.5, 76.9)$

$$t_x = (408.0 - 405.8)/31.5 = 0.070$$
$$t_y = (167.0 - 170.8)/76.9 = -0.049$$
$$t_w = \log(29.9/31.5) = -0.023$$
$$t_h = \log(73.0/76.9) = -0.023$$

SSD(Single Shot MultiBox Detector)

https://mc.ai/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing/

SSD: Single Shot MultiBox Detector

Wei Liu[1], Dragomir Anguelov[2], Dumitru Erhan[3], Christian Szegedy[3],
Scott Reed[4], Cheng-Yang Fu[1], Alexander C. Berg[1]

[1]UNC Chapel Hill [2]Zoox Inc. [3]Google Inc. [4]University of Michigan, Ann-Arbor
[1]wliu@cs.unc.edu, [2]drago@zoox.com, [3]{dumitru,szegedy}@google.com,
[4]reedscot@umich.edu, [1]{cyfu,aberg}@cs.unc.edu

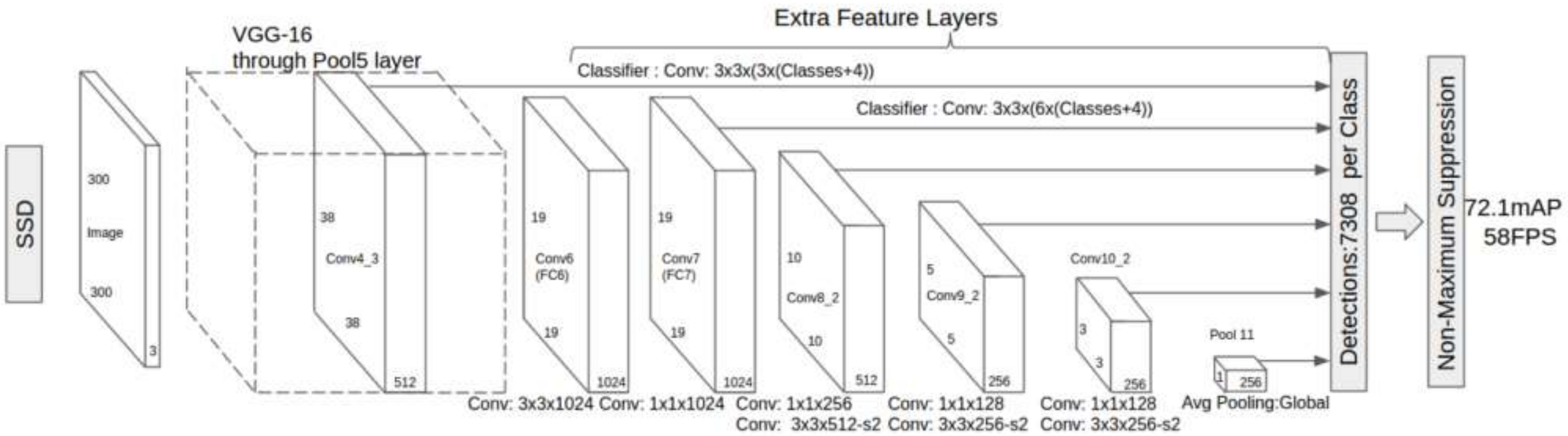
Abstract. We present a method for detecting objects in images using a single deep neural network. Our approach, named SSD, discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. SSD is simple relative to methods that require object proposals because it completely eliminates proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network. This makes SSD easy to train and straightforward to integrate into systems that require a detection component. Experimental results on the PASCAL VOC, COCO, and ILSVRC datasets confirm that SSD has competitive accuracy to methods that utilize an additional object proposal step and is much faster, while providing a unified framework for both training and inference. For 300×300 input, SSD achieves 74.3% mAP[1] on VOC2007 `test` at 59 FPS on a Nvidia Titan X and for 512×512 input, SSD achieves 76.9% mAP, outperforming a comparable state-of-the-art Faster R-CNN model. Compared to other single stage methods, SSD has much better accuracy even with a smaller input image size. Code is available at: `https://github.com/weiliu89/caffe/tree/ssd`.

Keywords: Real-time Object Detection; Convolutional Neural Network

Classification + Localization

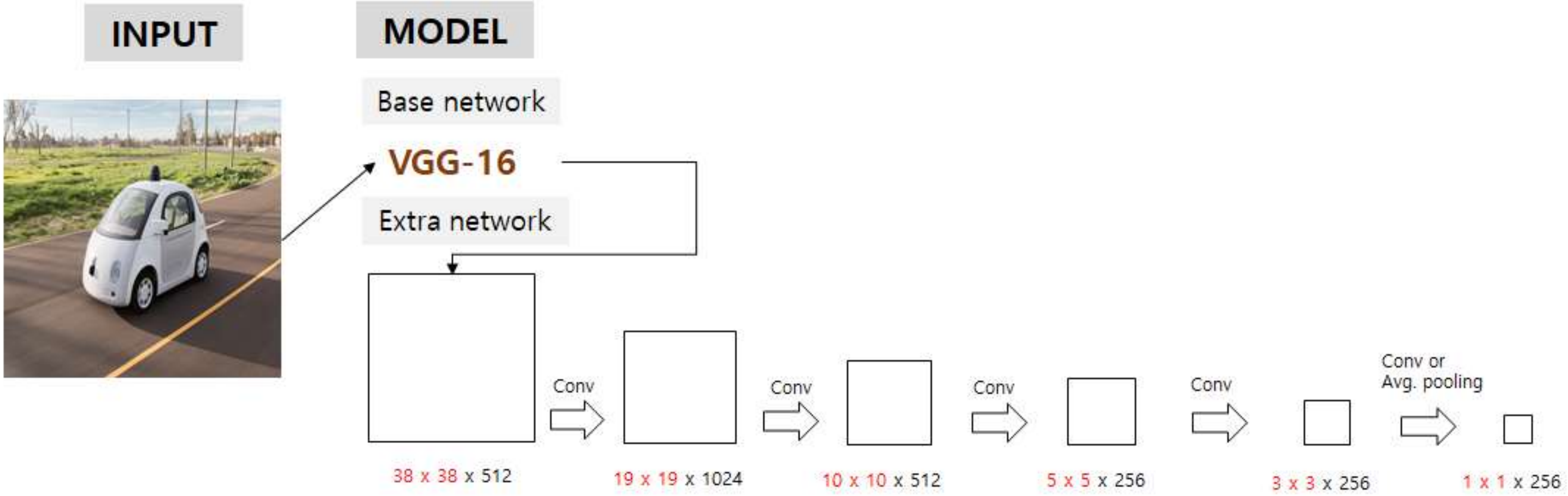
https://arxiv.org/pdf/1512.02325.pdf

Classification + Localization



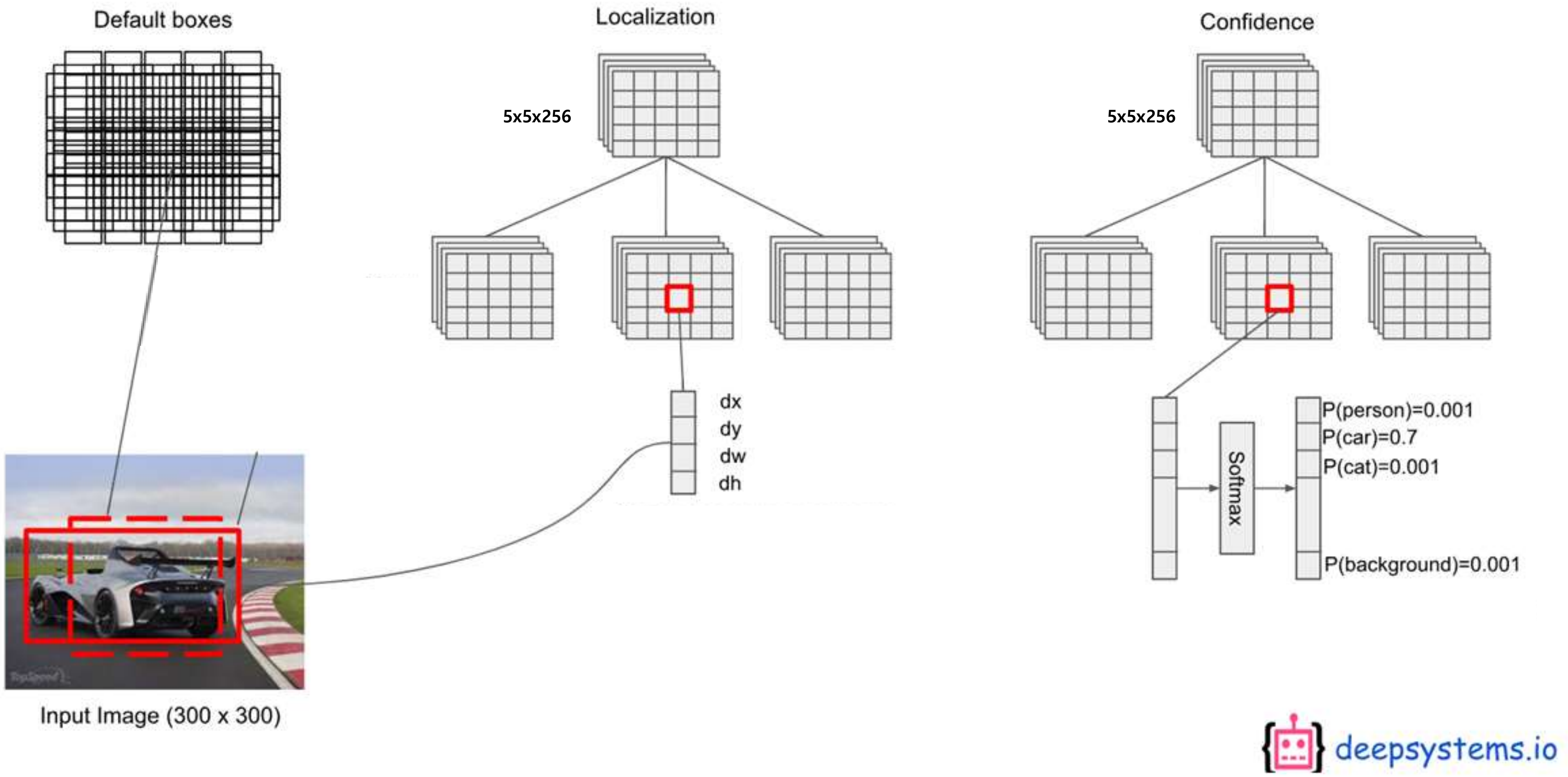
SSD(Single Shot multi-box Detector)

Classification + Localization



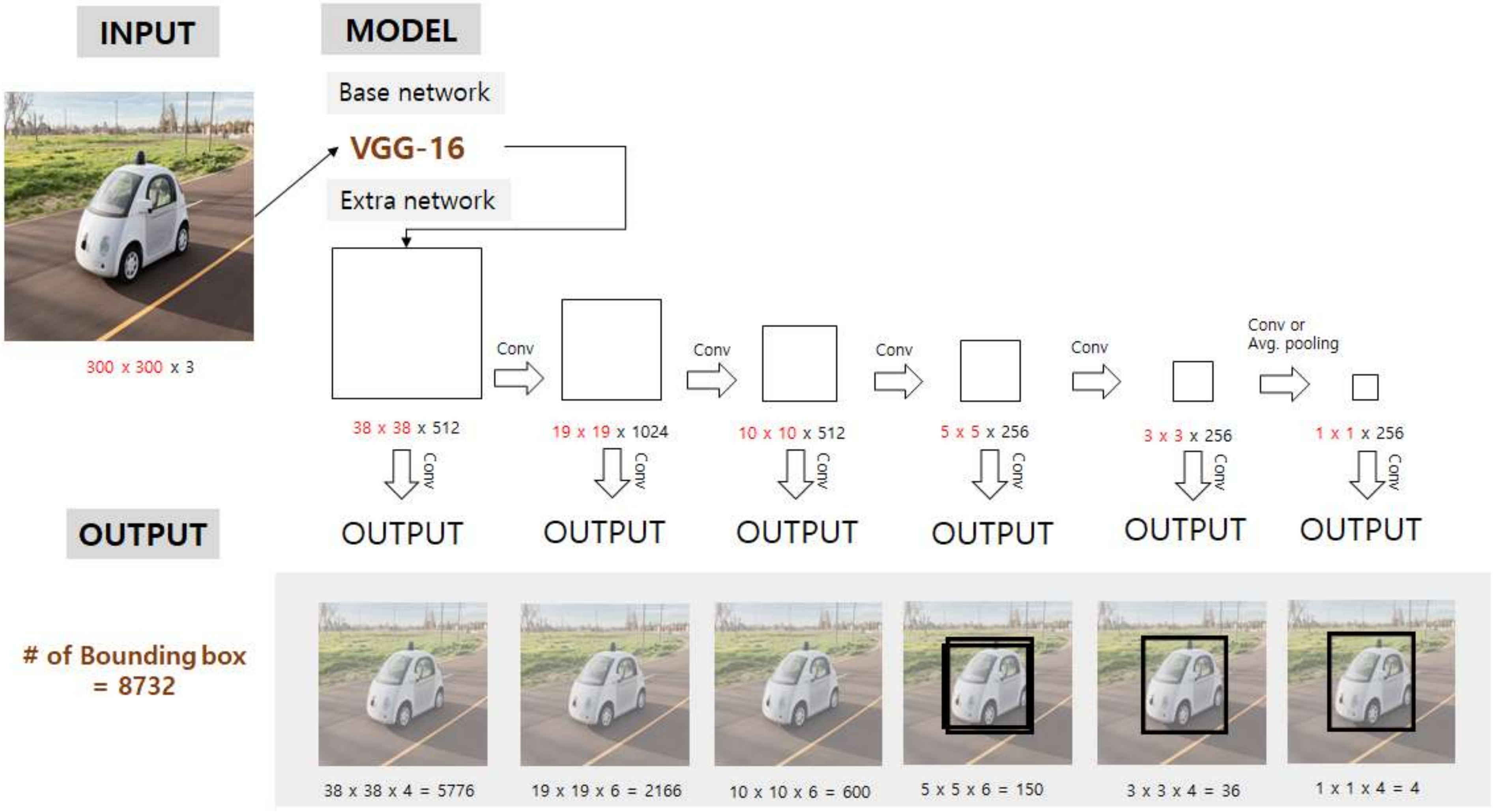
https://taeu.github.io/paper/deeplearning-paper-ssd/

Classification + Localization



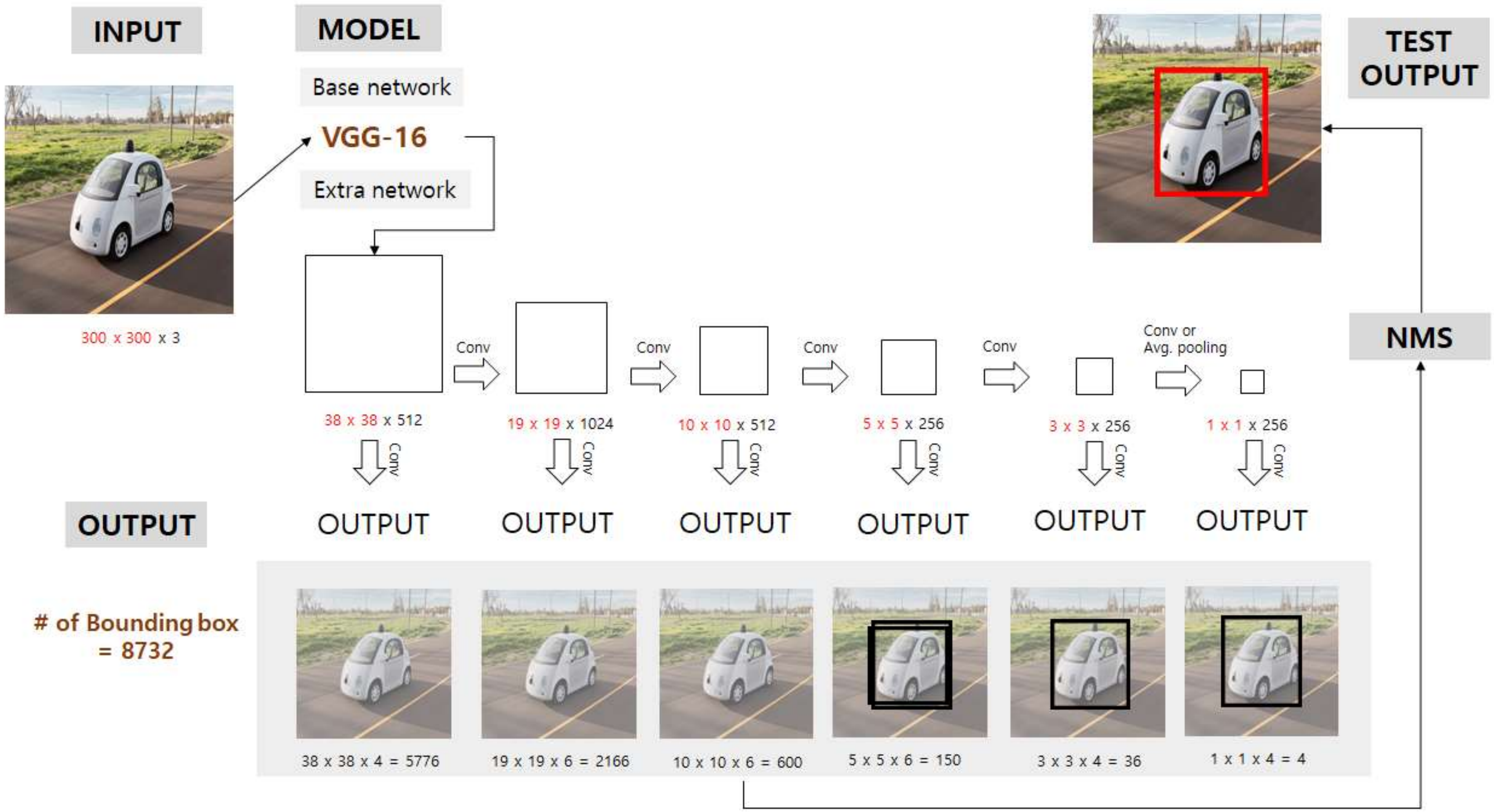
https://taeu.github.io/paper/deeplearning-paper-ssd/

Classification + Localization



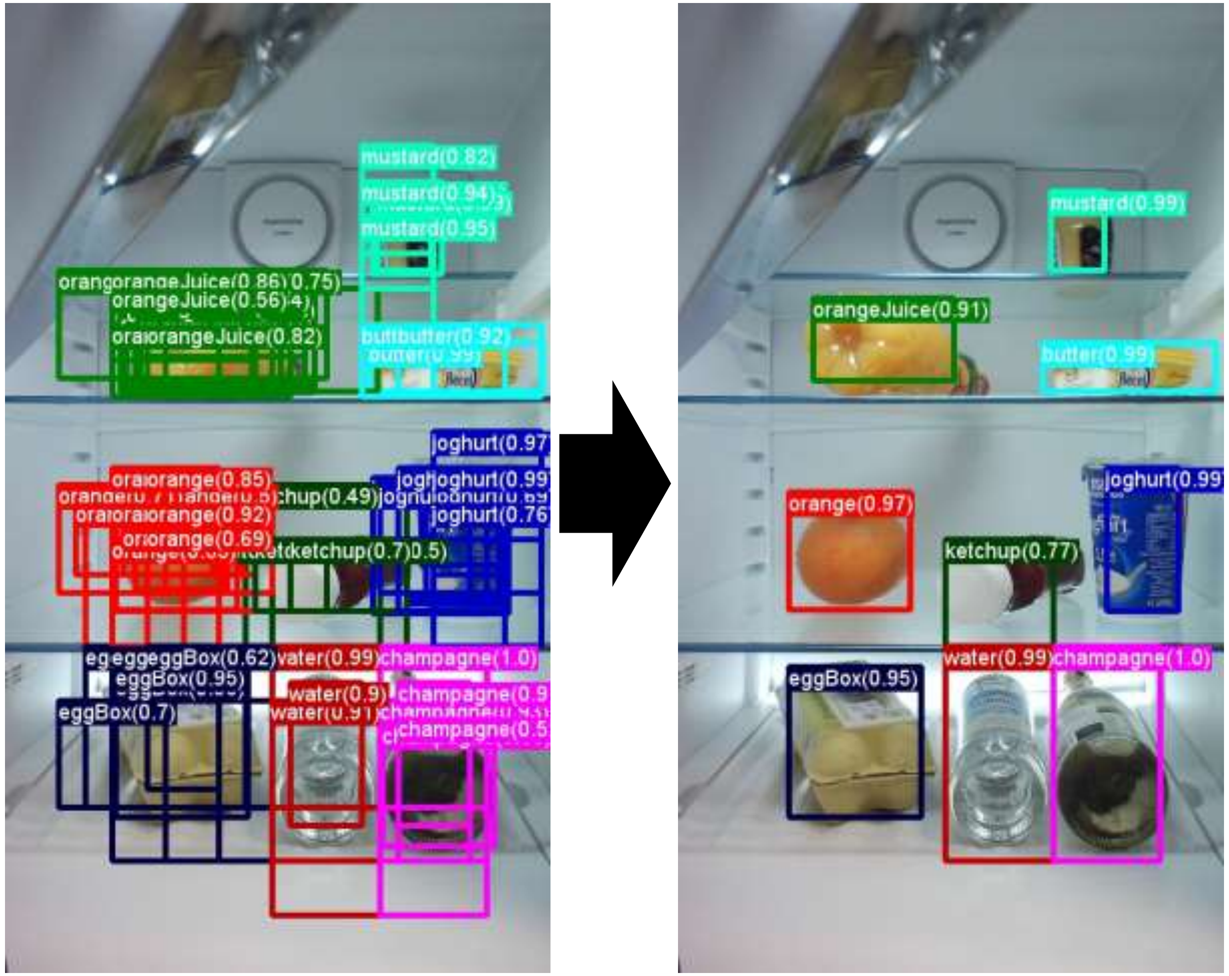
https://taeu.github.io/paper/deeplearning-paper-ssd/

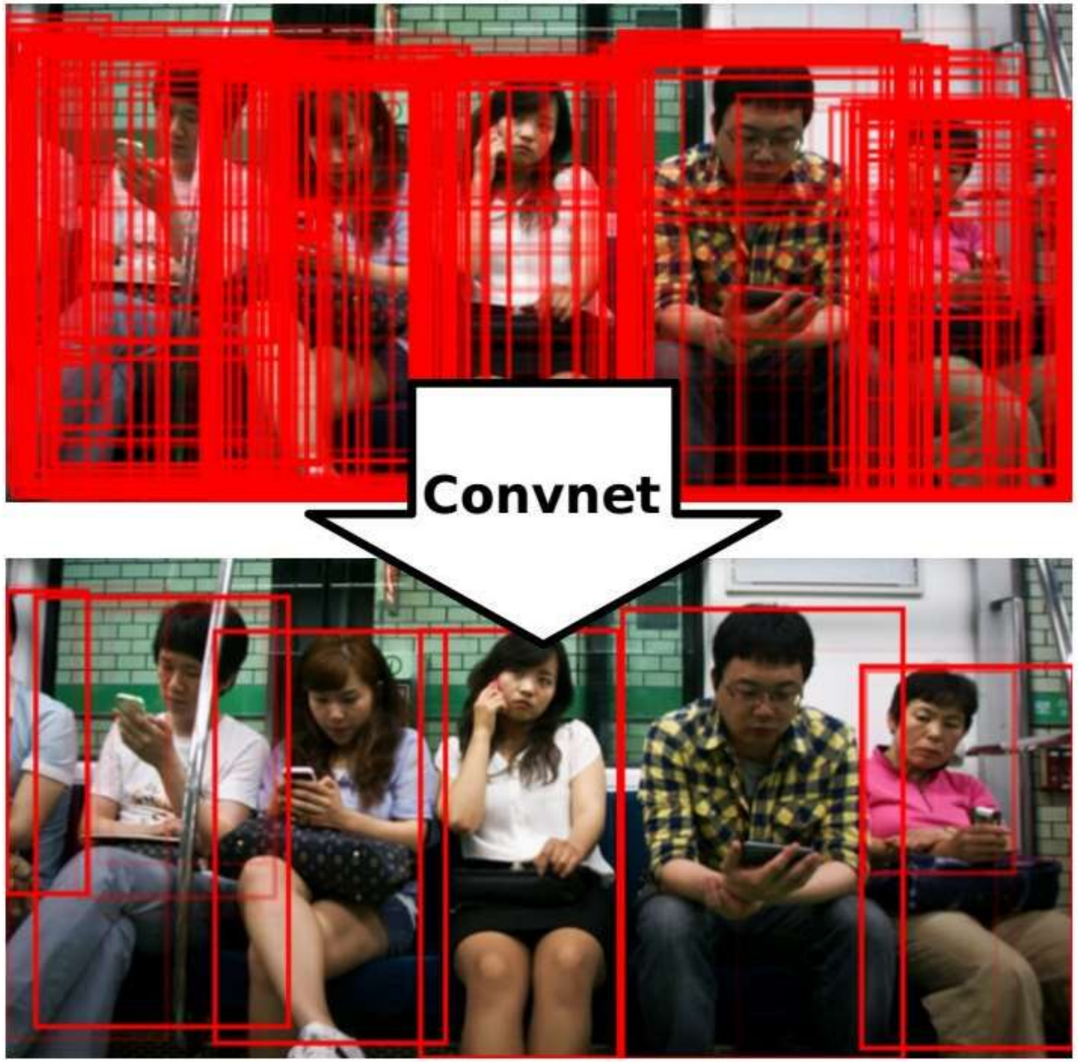
Classification + Localization



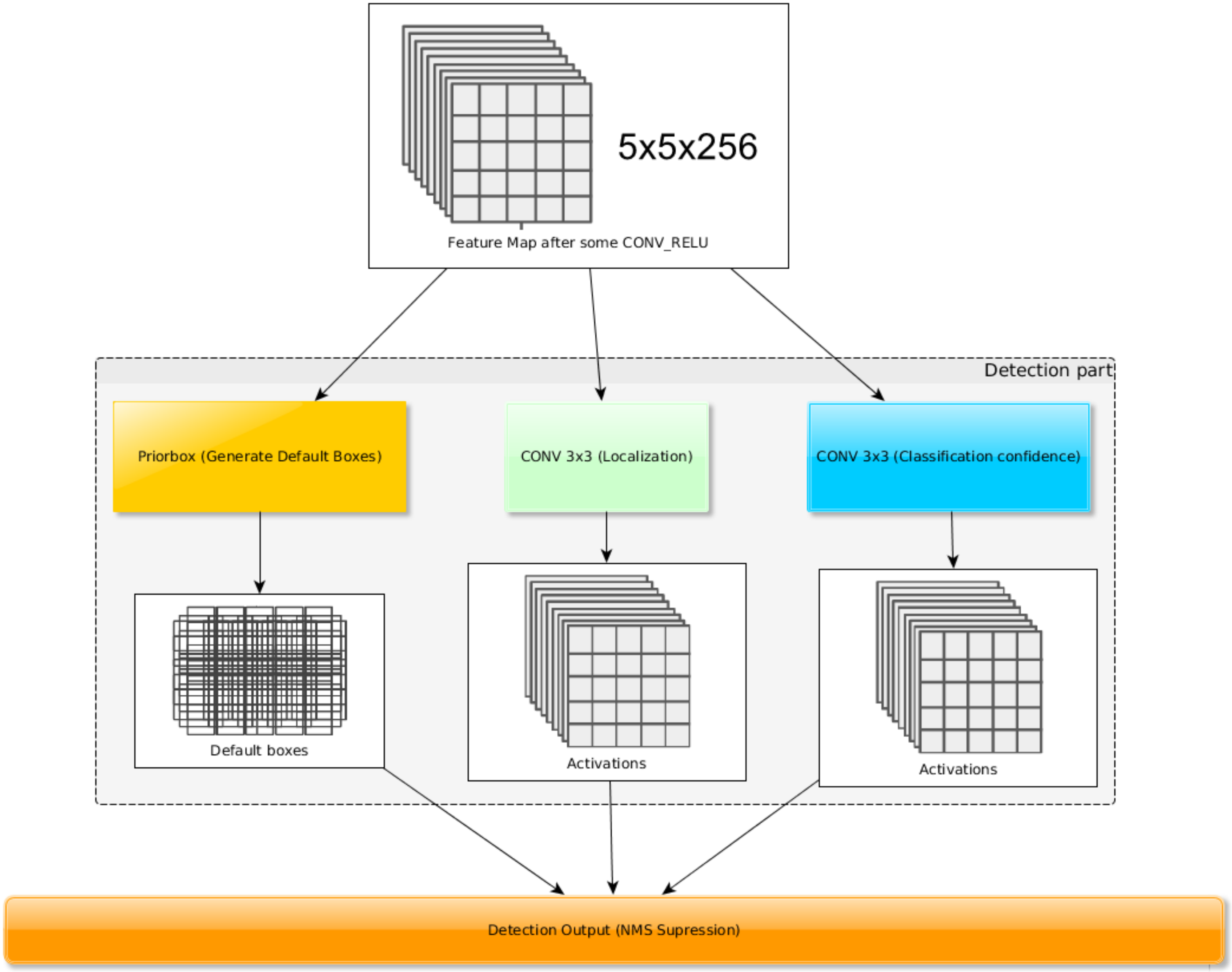
https://taeu.github.io/paper/deeplearning-paper-ssd/

NMS


mustard(0.82)
mustard(0.94)
mustard(0.95)
orangeJuice(0.82)
orange(0.85)
orange(0.92)
orange(0.69)
joghurt(0.97
joghurt(0.76
eggBox(0.62)
eggBox(0.95)
eggBox(0.7)
water(0.99)
champagne(1.0)
mustard(0.99)
orangeJuice(0.91)
butter(0.99)
orange(0.97)
ketchup(0.77)
joghurt(0.99
eggBox(0.95)
water(0.99)
champagne(1.0)

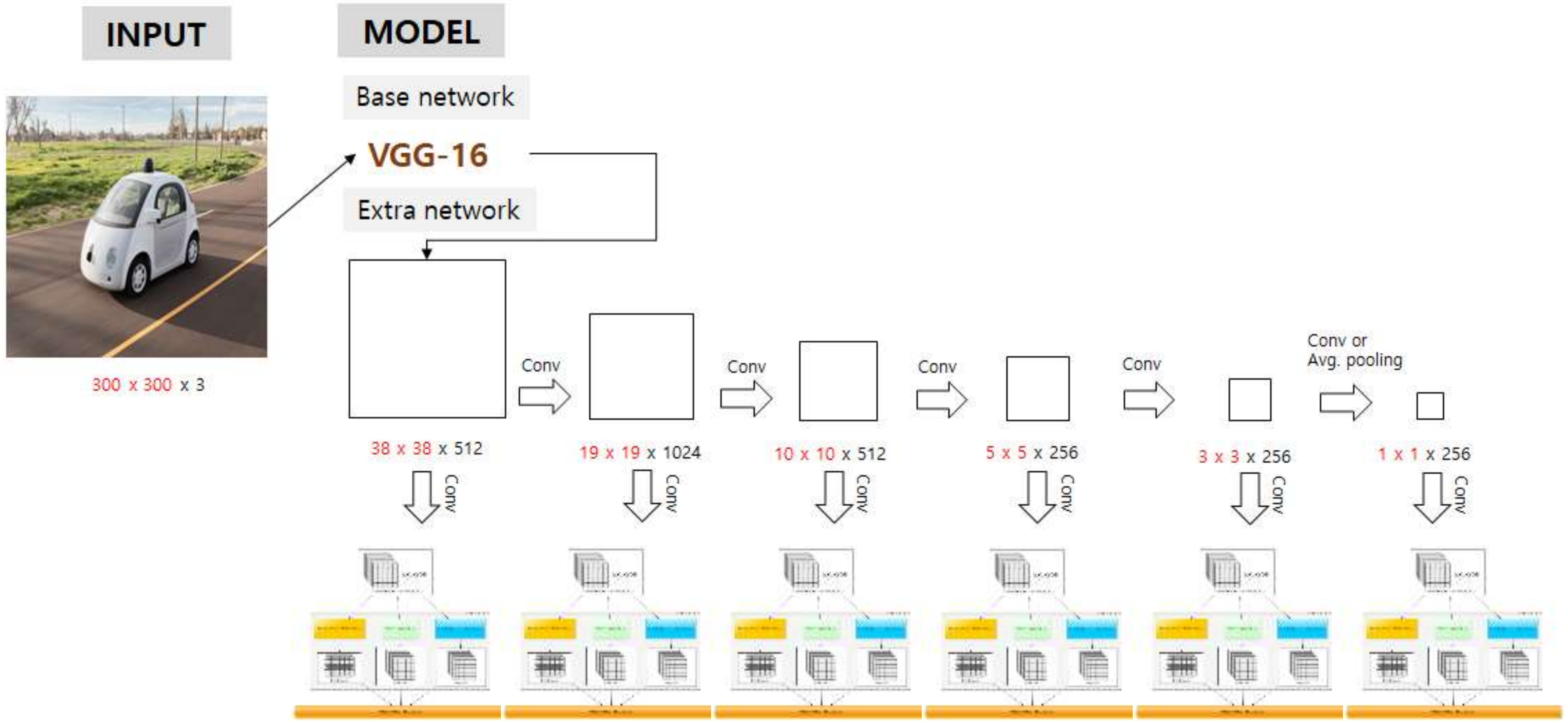

Convnet

Classification + Localization

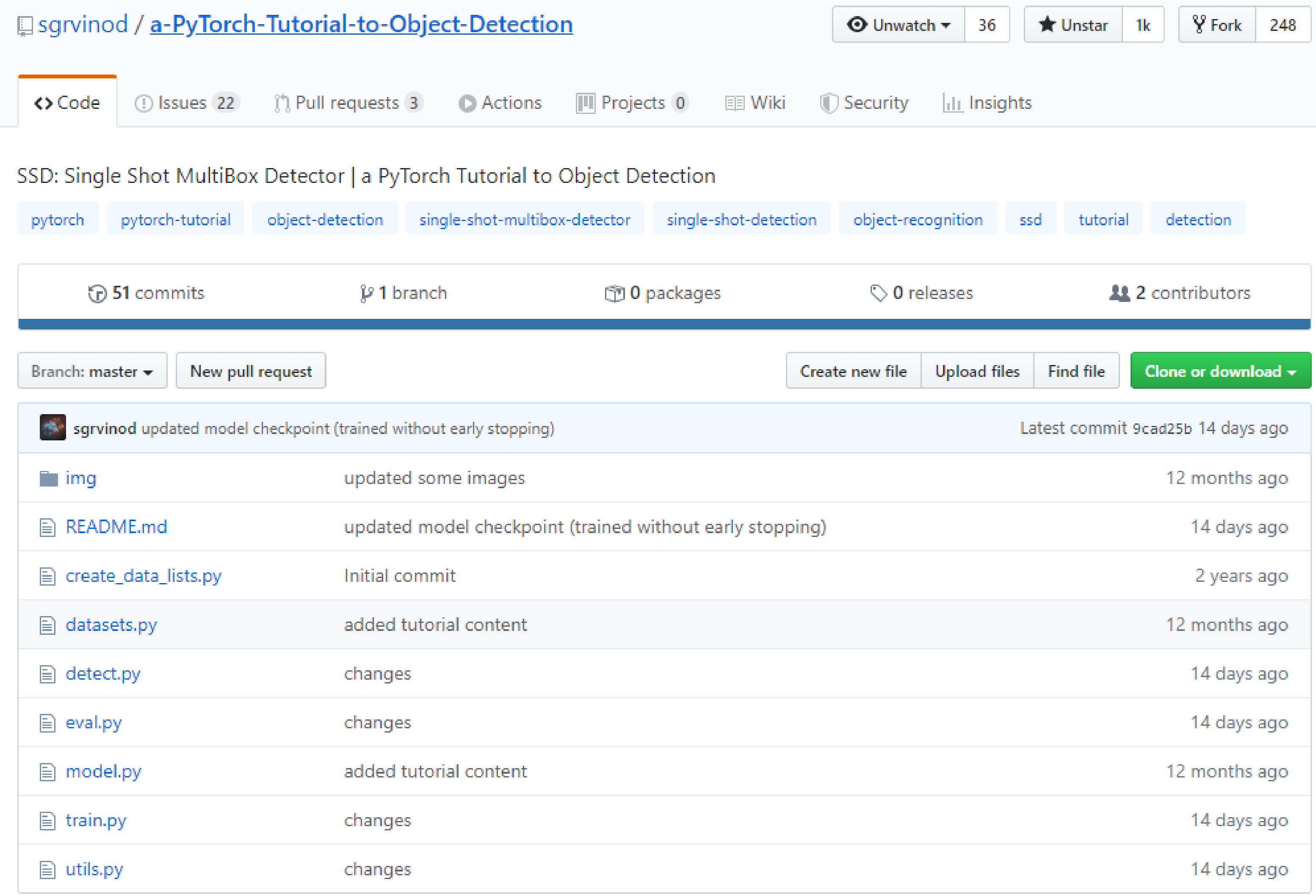


https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/single-shot-detectors/ssd.html

Classification + Localization



https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/single-shot-detectors/ssd.html



https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SEJONG RCV Winter School

- Object Detection(Halfway) -

Multispectral Deep Neural Networks for Pedestrian Detection

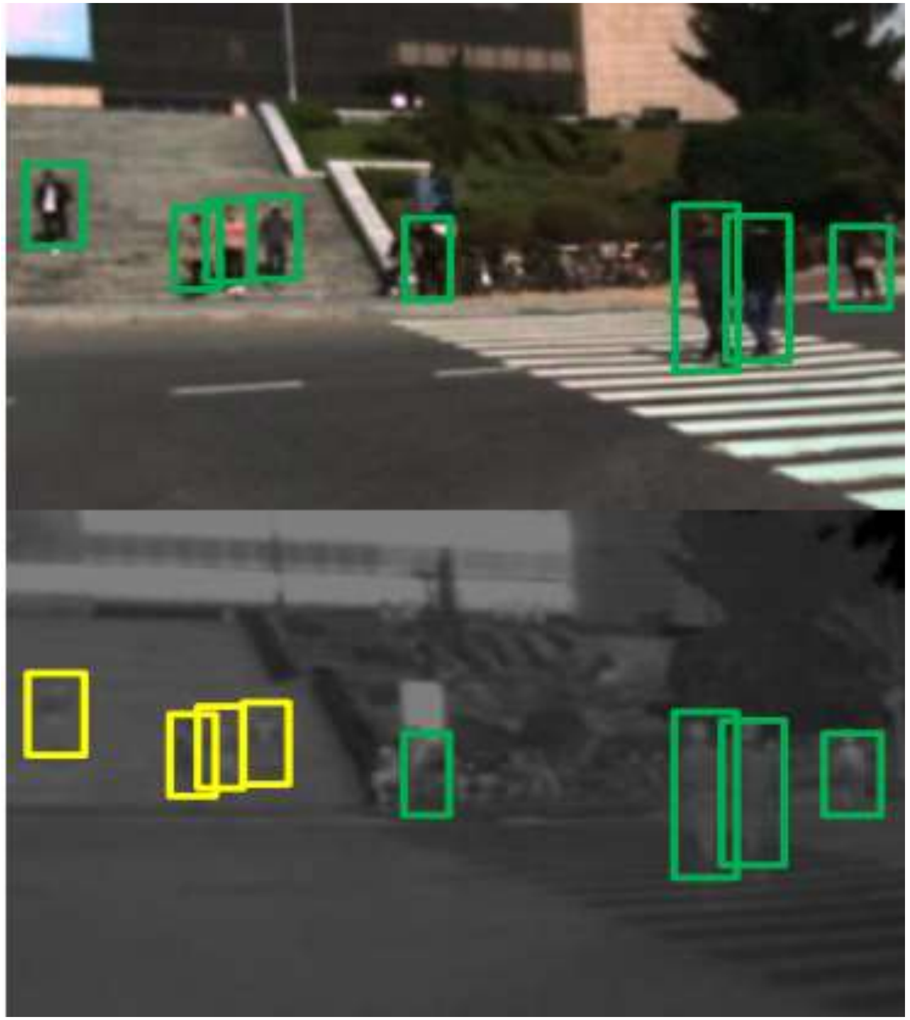
Jingjing Liu[1]
http://paul.rutgers.edu/~jl1322
Shaoting Zhang[2]
szhang16@uncc.edu
Shu Wang[1]
sw498@cs.rutgers.edu
Dimitris N. Metaxas[1]
https://www.cs.rutgers.edu/~dnm

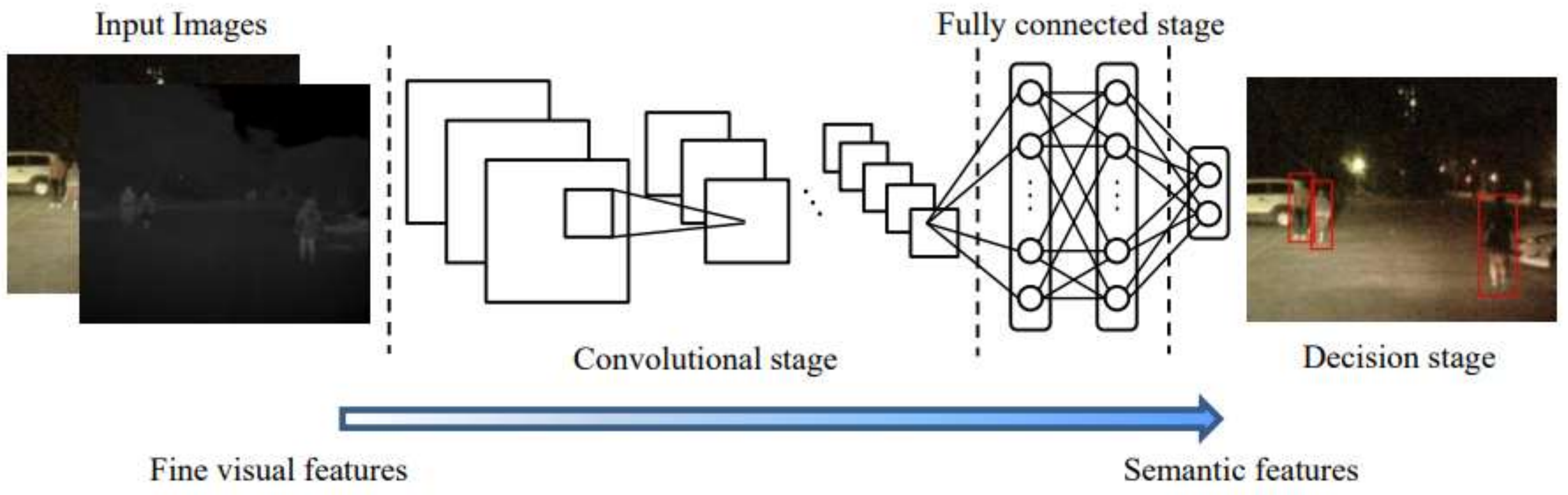
[1] Department of Computer Science
Rutgers University
Piscataway, NJ, USA

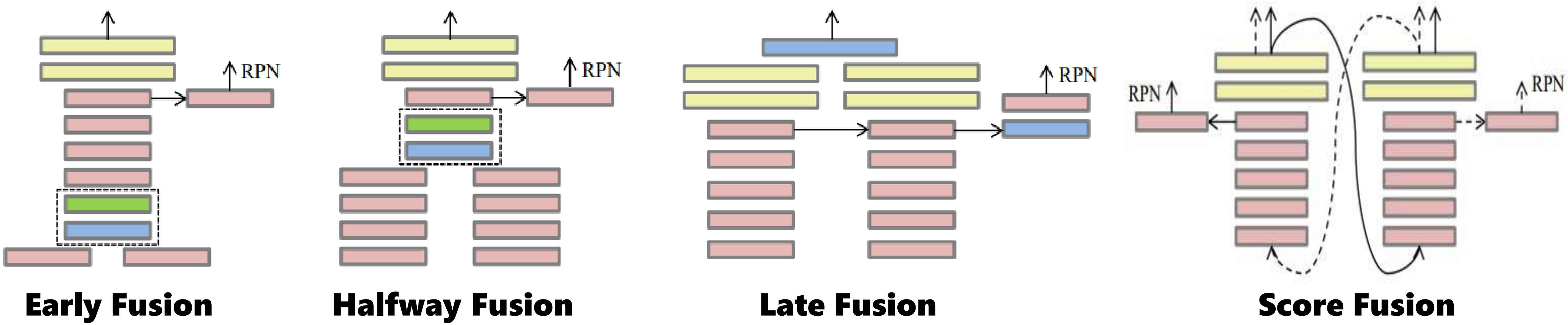
[2] Department of Computer Science
UNC Charlotte
Charlotte, NC, USA

Abstract

Multispectral pedestrian detection is essential for around-the-clock applications, *e.g.*, surveillance and autonomous driving. We deeply analyze Faster R-CNN for multispectral pedestrian detection task and then model it into a convolutional network (ConvNet) fusion problem. Further, we discover that ConvNet-based pedestrian detectors trained by color or thermal images separately provide complementary information in discriminating human instances. Thus there is a large potential to improve pedestrian detection by using color and thermal images in DNNs simultaneously. We carefully design four ConvNet fusion architectures that integrate two-branch ConvNets on different DNNs stages, all of which yield better performance compared with the baseline detector. Our experimental results on KAIST pedestrian benchmark show that the Halfway Fusion model that performs fusion on the middle-level convolutional features outperforms the baseline method by 11% and yields a missing rate 3.5% lower than the other proposed architectures.




Input Images
Fully connected stage
Convolutional stage
Decision stage
Fine visual features
Semantic features


RPN
RPN
RPN
RPN
RPN
Early Fusion
Halfway Fusion
Late Fusion
Score Fusion

SEJONG RCV Winter School

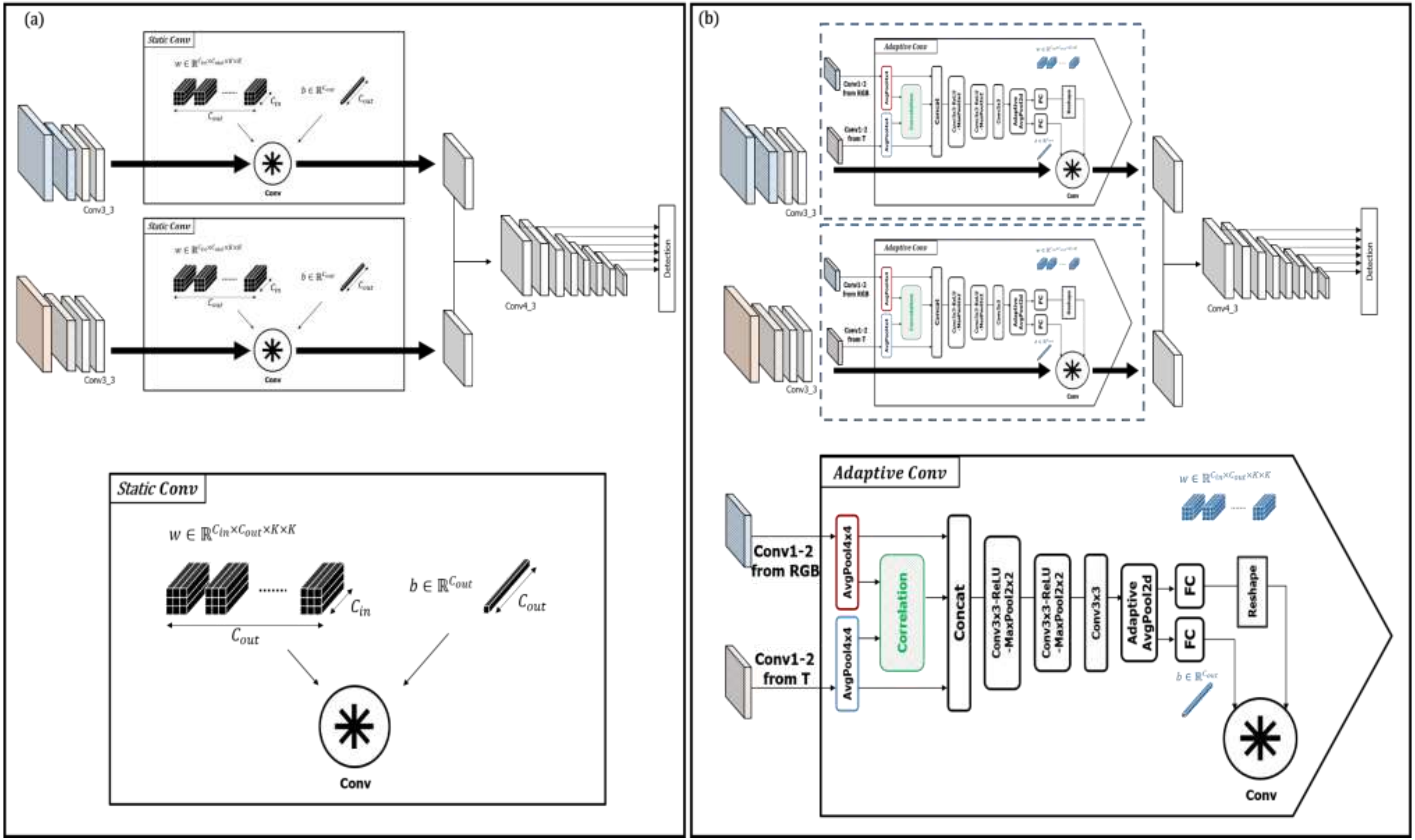
- Object Detection(Static Fusion & Adaptive Fusion) -

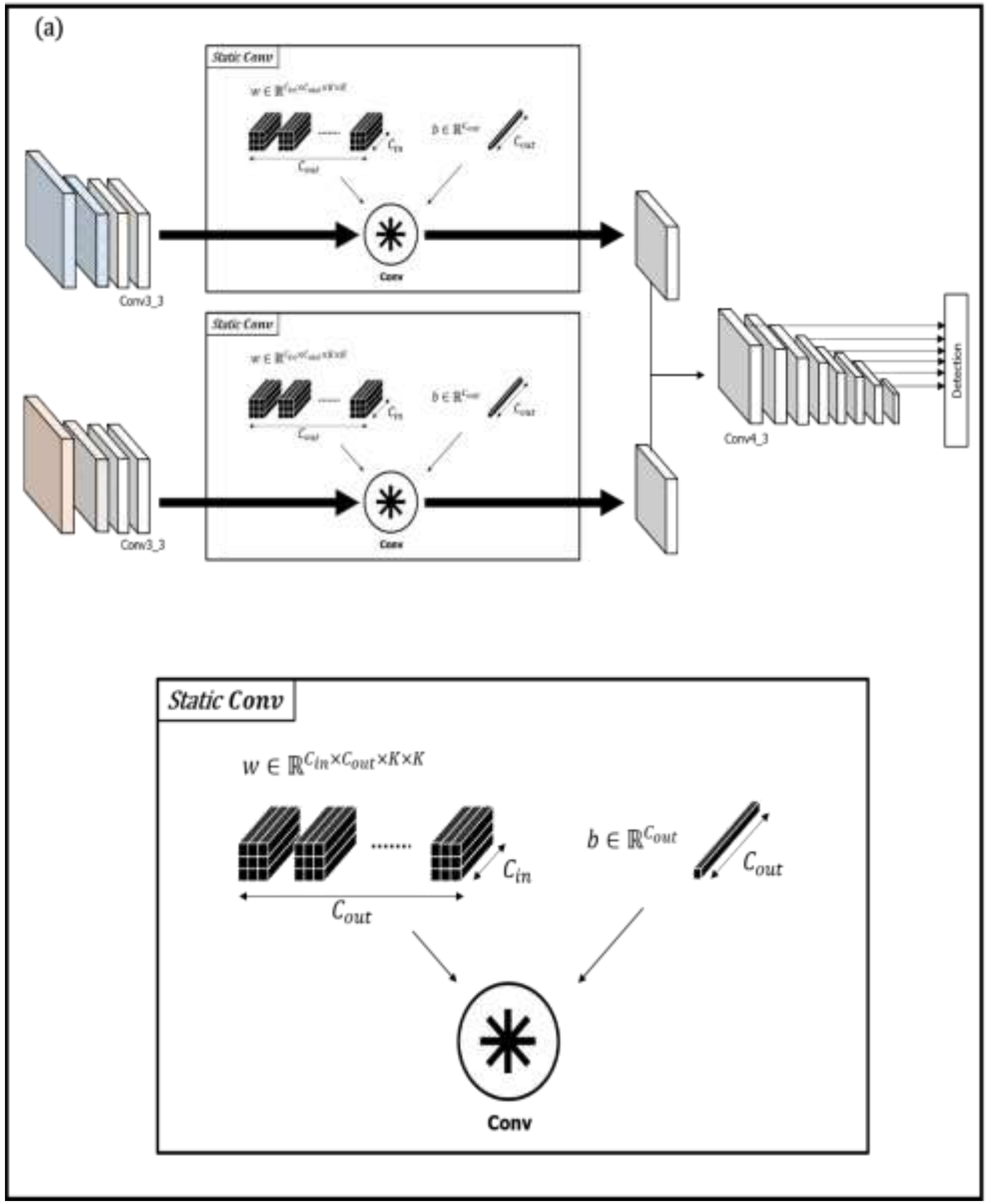
Single-Shot Adaptive Fusion Network for Robust Multispectral Pedestrian Detection

*김지원 [1], *조 원 [1], 남현호 [1], 황순민 [1], 노치원 [2], 김남일 [3], 최유경 [1]
세종대학교 Robotics Computer Vision (RCV) 연구실 [1], POTENIT [2], NAVER LABS[3]
{jwkim, jwon, hhnam, smhwang, ykchoi}@rcv.sejong.ac.kr, cwroh@potenit.com, namil.kim@naverlabs.com

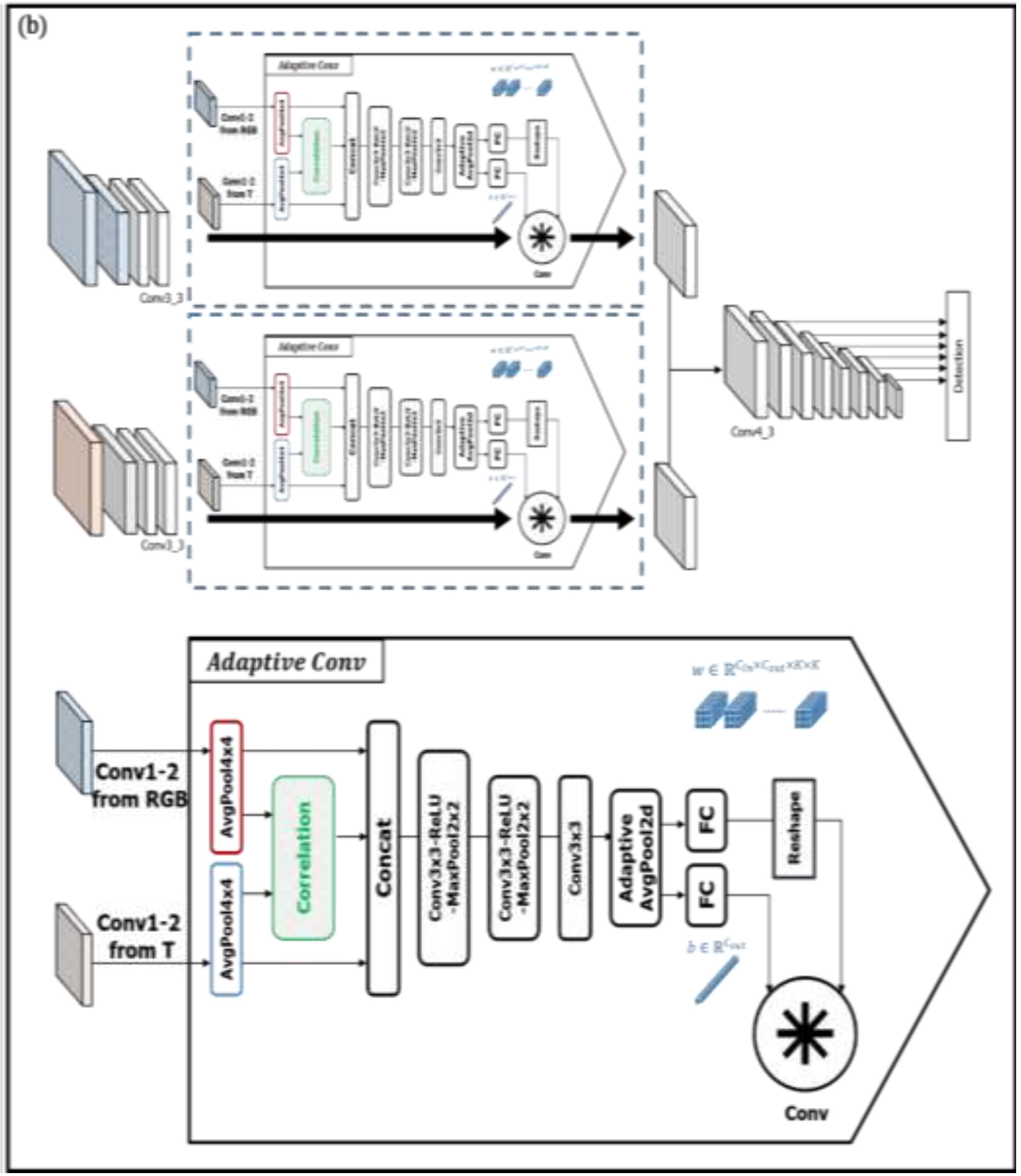
요 약

컴퓨터 비전 분야에서 밤과 낮, 날씨의 변화 등 다양한 자연환경에 대한 환경변화 강인성은 극복 해야할 중요 주제로, 오래전부터 다양한 분야에서 많은 연구들이 진행되었다. 하지만 일정 수준 예측이 가능한 환경변화 강인성 이외에도, 실제 시스템 운영 시에 센서의 파손 및 고장, 먼지/곤충에 의한 센서의 가려짐 등 예상치 못한 상황들이 일어날 수 있고, 알고리즘 학습 시 고려하지 못했던 이러한 변수들은 실제 테스트 중 큰 성능저하를 야기한다. 본 논문에서는 학습과정에서 예상할 수 없었던 변화에 대해서도 강인성을 확보할 수 있는 새로운 학습방법과 입력 영상에 따라 유동적으로 학습 파라미터를 생성하는 모델을 제안한다. 제안하는 방법론을 KAIST multispectral benchmark 데이터 셋 기반으로 다양한 예측 불가한 상황들을 모사하여 검증하였으며, 예측 불가한 상황에 강인하게 동작하는 것뿐 아니라, 정상 조건에서도 SOTA 의 성능을 보이는 것을 입증하였다.

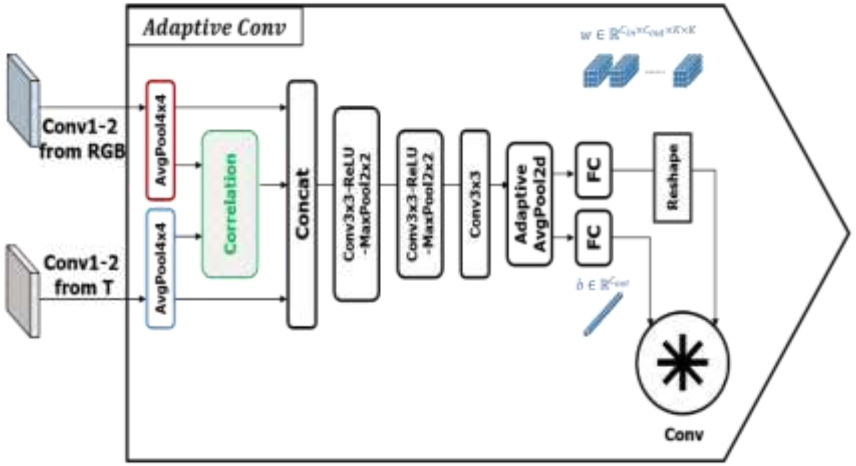

(a)
Static Conv
Conv3_3
Conv4_3
Detection
Conv
(b)
Adaptive Conv
Conv1-2 from RGB
Conv1-2 from T
AvgPool4x4
Correlation
Concat
Conv3x3-ReLU -MaxPool2x2
Conv3x3
Adaptive AvgPool2d
FC
Reshape
Conv




```
############################ RGB #####################################

out_vis = F.relu(self.conv2_1_bn_vis(self.conv2_1_vis(out_vis)))
out_vis = F.relu(self.conv2_2_bn_vis(self.conv2_2_vis(out_vis)))
out_vis = self.pool2_vis(out_vis)

out_vis = F.relu(self.conv3_1_bn_vis(self.conv3_1_vis(out_vis)))
out_vis = F.relu(self.conv3_2_bn_vis(self.conv3_2_vis(out_vis)))
out_vis = F.relu(self.conv3_3_bn_vis(self.conv3_3_vis(out_vis)))

out_vis = F.relu(self.conv1x1_sf_vis(out_vis))
#out_vis = self.fusion_R(conv1_2_feats_vis, conv1_2_feats_lwir, out_vis)

##########################################################################

############################ Thermal #####################################

out_lwir = F.relu(self.conv2_1_bn_lwir(self.conv2_1_lwir(out_lwir)))
out_lwir = F.relu(self.conv2_2_bn_lwir(self.conv2_2_lwir(out_lwir)))
out_lwir = self.pool2_lwir(out_lwir)

out_lwir = F.relu(self.conv3_1_bn_lwir(self.conv3_1_lwir(out_lwir)))
out_lwir = F.relu(self.conv3_2_bn_lwir(self.conv3_2_lwir(out_lwir)))
out_lwir = F.relu(self.conv3_3_bn_lwir(self.conv3_3_lwir(out_lwir)))

out_lwir = F.relu(self.conv1x1_sf_lwir(out_lwir))
#out_lwir = self.fusion_T(conv1_2_feats_vis, conv1_2_feats_lwir, out_lwir)

#########################################################################

out = out_vis*0.5 + out_lwir*0.5

#########################################################################
```




```
########################### RGB ####################################

out_vis = F.relu(self.conv2_1_bn_vis(self.conv2_1_vis(out_vis)))
out_vis = F.relu(self.conv2_2_bn_vis(self.conv2_2_vis(out_vis)))
out_vis = self.pool2_vis(out_vis)

out_vis = F.relu(self.conv3_1_bn_vis(self.conv3_1_vis(out_vis)))
out_vis = F.relu(self.conv3_2_bn_vis(self.conv3_2_vis(out_vis)))
out_vis = F.relu(self.conv3_3_bn_vis(self.conv3_3_vis(out_vis)))

# out_vis = F.relu(self.conv1x1_sf_vis(out_vis))
out_vis = self.fusion_R(conv1_2_feats_vis, conv1_2_feats_lwir, out_vis)

##########################################################################

########################### Thermal ####################################

out_lwir = F.relu(self.conv2_1_bn_lwir(self.conv2_1_lwir(out_lwir)))
out_lwir = F.relu(self.conv2_2_bn_lwir(self.conv2_2_lwir(out_lwir)))
out_lwir = self.pool2_lwir(out_lwir)

out_lwir = F.relu(self.conv3_1_bn_lwir(self.conv3_1_lwir(out_lwir)))
out_lwir = F.relu(self.conv3_2_bn_lwir(self.conv3_2_lwir(out_lwir)))
out_lwir = F.relu(self.conv3_3_bn_lwir(self.conv3_3_lwir(out_lwir)))

# out_lwir = F.relu(self.conv1x1_sf_lwir(out_lwir))
out_lwir = self.fusion_T(conv1_2_feats_vis, conv1_2_feats_lwir, out_lwir)

#########################################################################

out = out_vis*0.5 + out_lwir*0.5

#########################################################################
```




```python
def forward(self, x, y, hx):

    bs = x.size(0)

    _x = F.avg_pool2d(x.detach(), kernel_size=4, stride=4)
    _y = F.avg_pool2d(y.detach(), kernel_size=4, stride=4)

    c = self.corr( _x, _y )
    c = c.view(c.size(0),c.size(1)*c.size(2),c.size(3),c.size(4))

    feat = t.cat( [_x, _y, c], 1 )

    feat = F.adaptive_avg_pool2d( self.conv(feat), (1, 1) )
    feat = self.conv_post( feat )

    feat = feat.view( bs, -1 )

    xx = list()

    wx = self.wx( feat ).view( bs, self.hidden_ch, self.out_ch, self.kernel_size, self.kernel_size)
    bx = self.bx( feat ).view( bs, self.out_ch)


    for ii in range(bs):
        xx.append( F.conv2d(hx[ii:ii+1,...], wx[ii,...], bx[ii,...], padding=self.padding)  )

    x = t.cat( xx, 0 )

    return x
```

```python
class AdaptiveFusion(nn.Module):
    def __init__(self, in_ch, hidden_ch, out_ch, kernel_size,patch_size):

        super(AdaptiveFusion, self).__init__()

        self.patch_size = patch_size

        self.in_ch = in_ch
        self.hidden_ch = hidden_ch
        self.out_ch = out_ch

        self.kernel_size = kernel_size
        self.padding = math.floor(kernel_size/2)

        self.corr = SpatialCorrelationSampler(kernel_size=1,patch_size=self.patch_size,stride=1,padding=0,dilation_patch=2)

        self.conv = nn.Sequential(
                nn.Conv2d( in_ch*2 + (self.patch_size)*(self.patch_size) , hidden_ch, kernel_size=3, padding=1, stride=1, bias=False),
                nn.ReLU(inplace=True),
                nn.MaxPool2d(kernel_size=2, stride=2),

                nn.Conv2d( hidden_ch, hidden_ch, kernel_size=3, padding=1, stride=1, bias=False),
                nn.ReLU(inplace=True),
                nn.MaxPool2d(kernel_size=2, stride=2),

                nn.Conv2d( hidden_ch, hidden_ch, kernel_size=3, padding=1, stride=1, bias=False),
            )
        self.conv_post = nn.Conv2d( hidden_ch, hidden_ch, kernel_size=1, padding=0, stride=1)

        self.wx = nn.Linear( hidden_ch, hidden_ch*out_ch*kernel_size*kernel_size )
        self.bx = nn.Linear( hidden_ch, out_ch )

        self.reset_parameters()
```