Object Detection

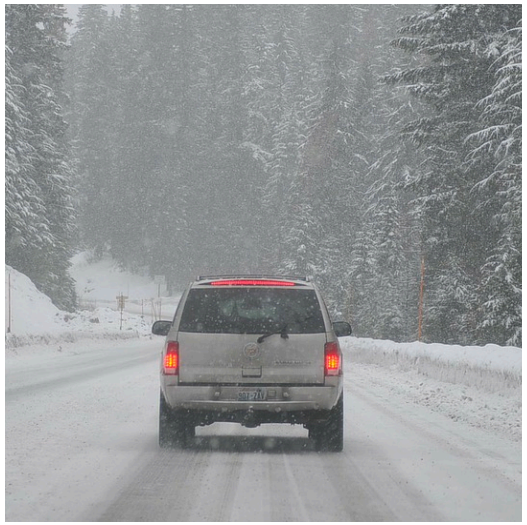
Suleyman Demirel University

CSS634: Deep Learning

PhD Abay Nussipbekov

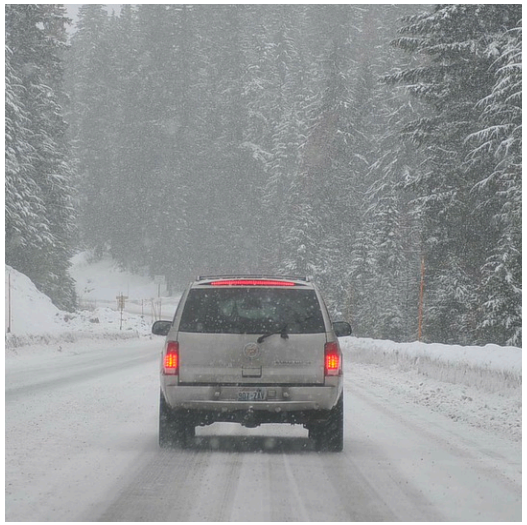
What are localization and detection?

Image classification

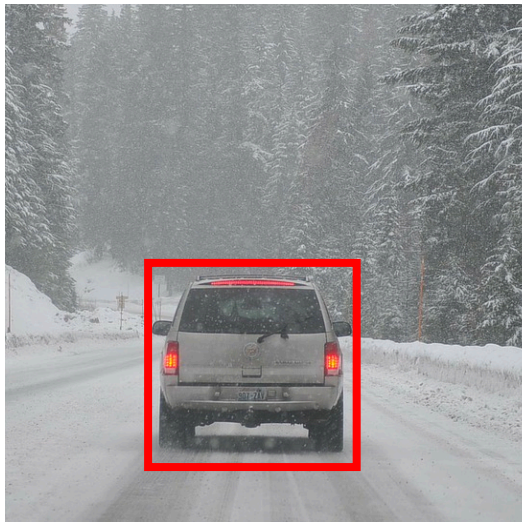


What are localization and detection?

Image classification



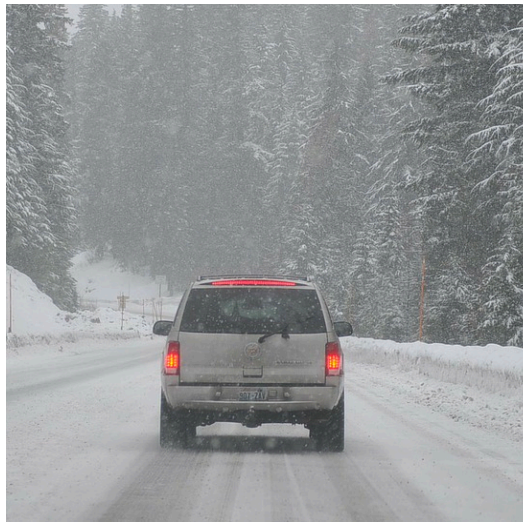
Classification with localization



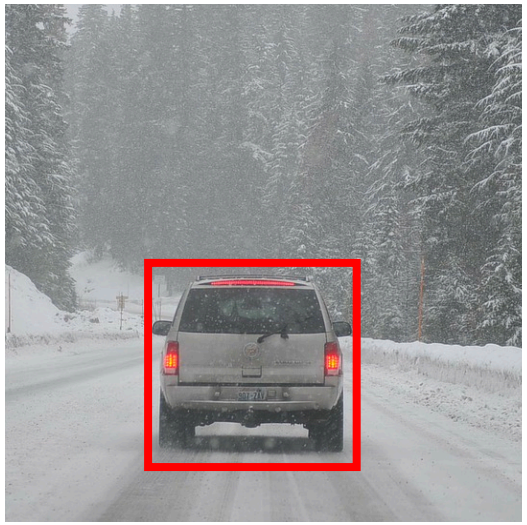
Detection

What are localization and detection?

Image classification



Classification with localization

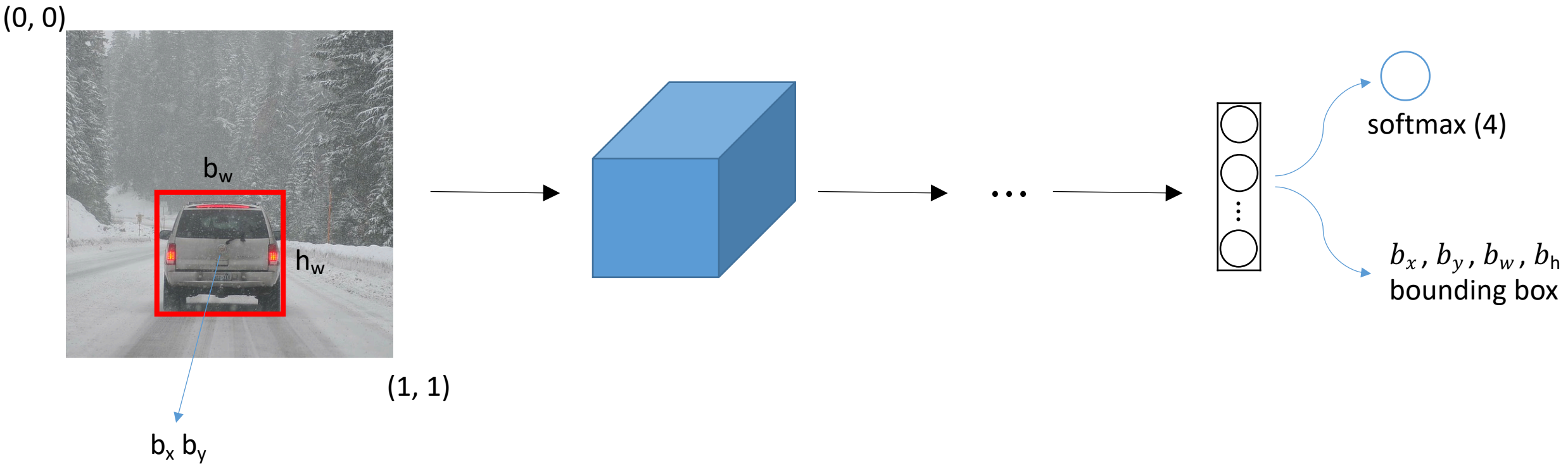


One object

Detection

Multiple objects

Classification with localization



1 - pedestrian

2 - car

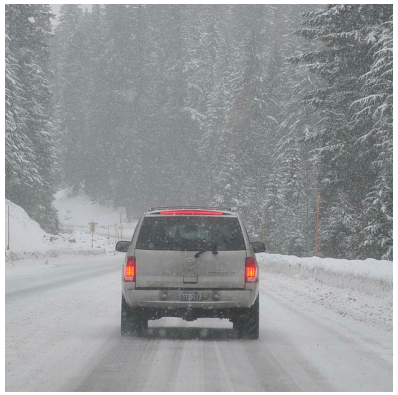
3 - motorcycle

4 - background

Defining the target label y

1 - pedestrian
2 - car
3 - motorcycle
4 - background

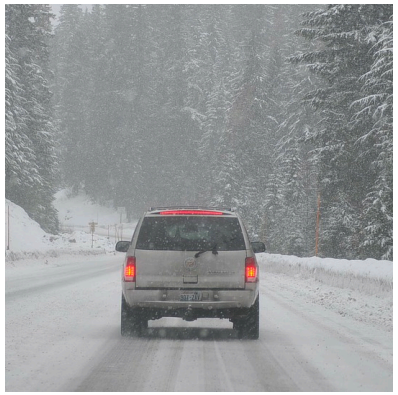
Need to output b_x, b_y, b_h, b_w, class label (1-4)



Defining the target label y

1 - pedestrian
2 - car
3 - motorcycle
4 - background

Need to output b_x, b_y, b_h, b_w, class label (1-4)

X =  

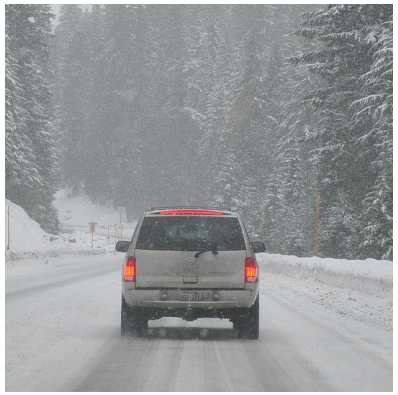
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Defining the target label y

1 - pedestrian
2 - car
3 - motorcycle
4 - background

Need to output b_x, b_y, b_h, b_w, class label (1-4)

X =



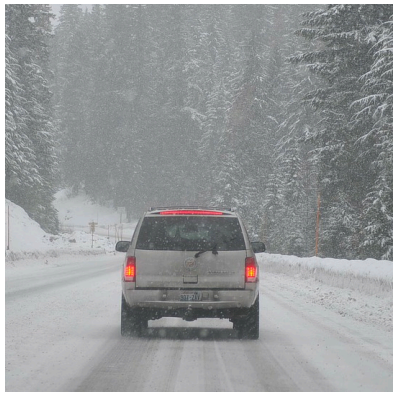
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \qquad y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Defining the target label y

1 - pedestrian
2 - car
3 - motorcycle
4 - background

Need to output b_x, b_y, b_h, b_w, class label (1-4)

X =



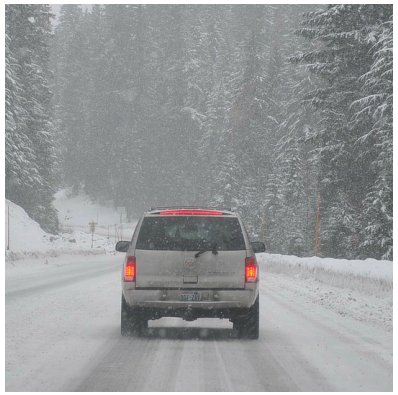
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \qquad y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

Defining the target label y

1 - pedestrian
2 - car
3 - motorcycle
4 - background

Need to output b_x, b_y, b_h, b_w, class label (1-4)

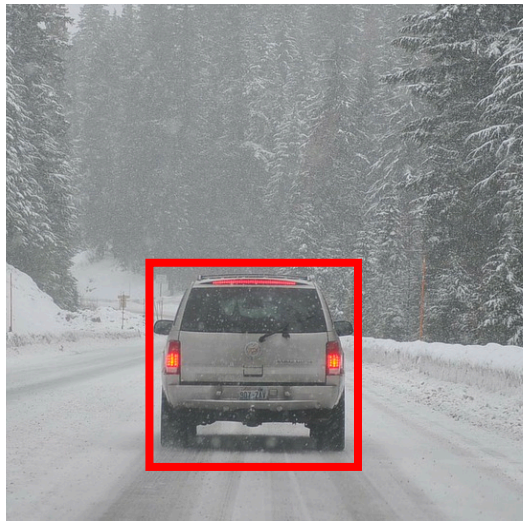
X =



$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + \cdots + (\hat{y}_8 - y_8)^2 & if\ y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & if\ y_1 = 0 \end{cases}$$

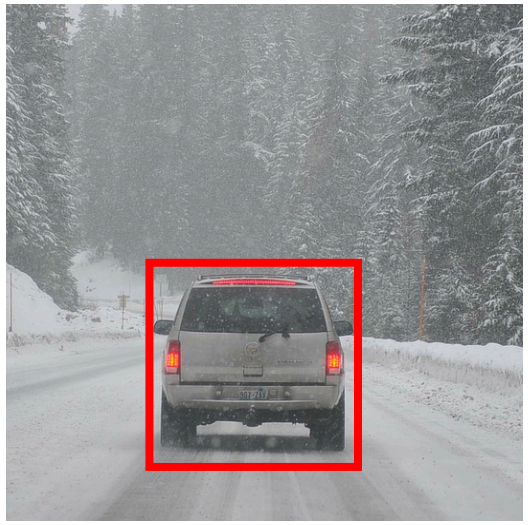
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

Landmark detection



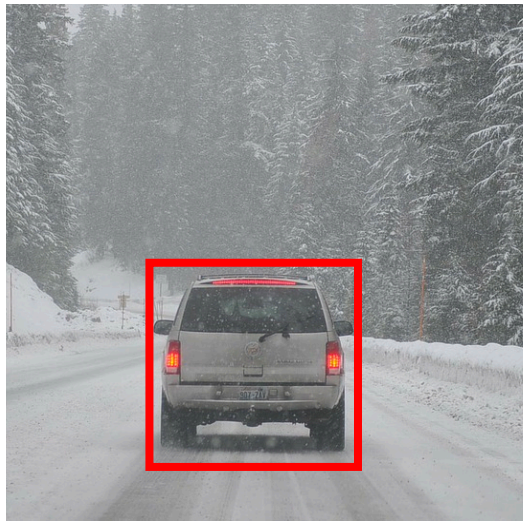
b_x, b_y, b_h, b_w

Landmark detection

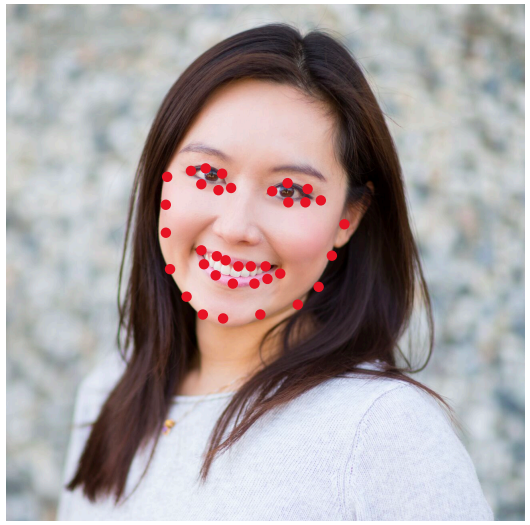


b_x, b_y, b_h, b_w

Landmark detection



b_x, b_y, b_h, b_w

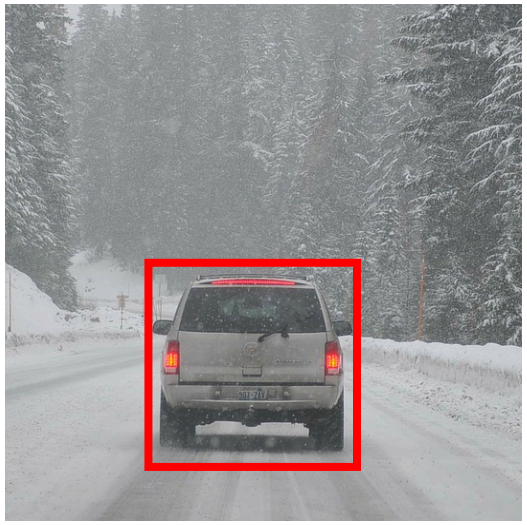


l_{1x}, l_{1y}
l_{2x}, l_{2y}

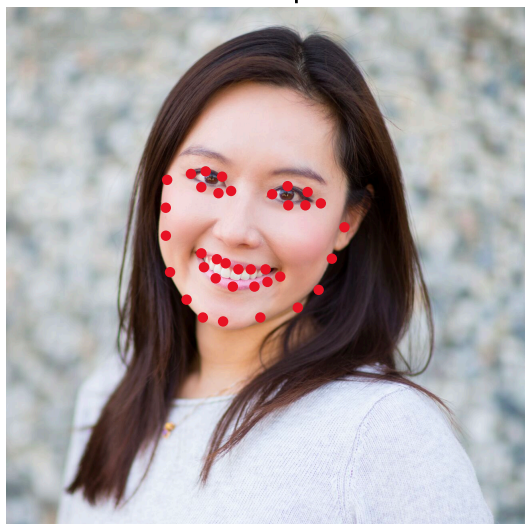
x, y

l_{64x}, l_{64y}

Landmark detection



b_x, b_y, b_h, b_w

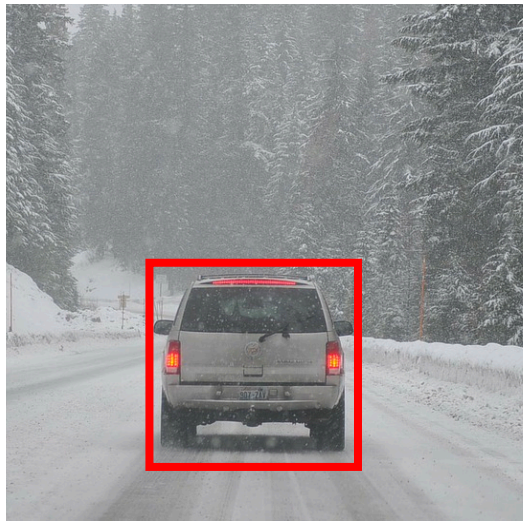


l_{1x}, l_{1y}
l_{2x}, l_{2y}

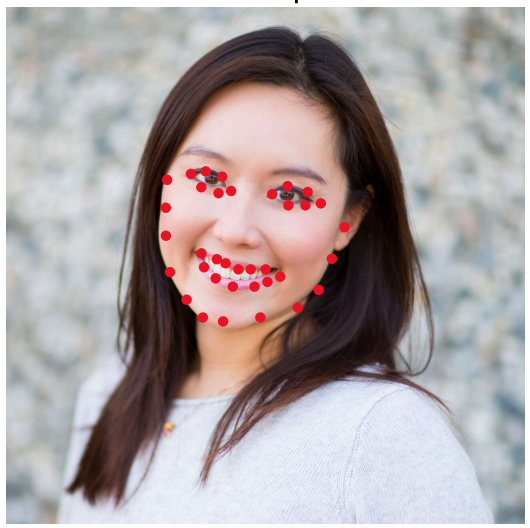
l_{64x}, l_{64y}

x, y

Landmark detection



b_x, b_y, b_h, b_w



l_{1x}, l_{1y}
l_{2x}, l_{2y}

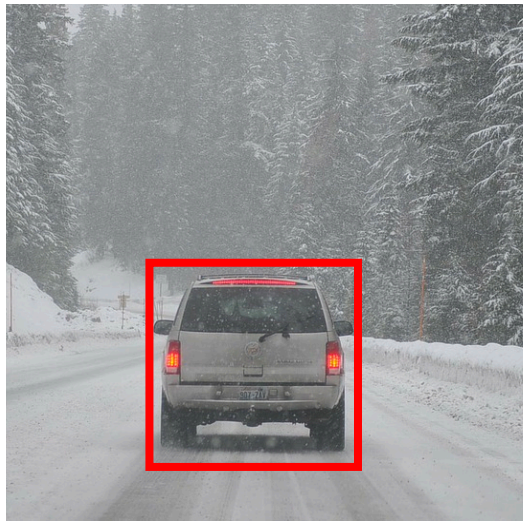
x, y

l_{64x}, l_{64y}

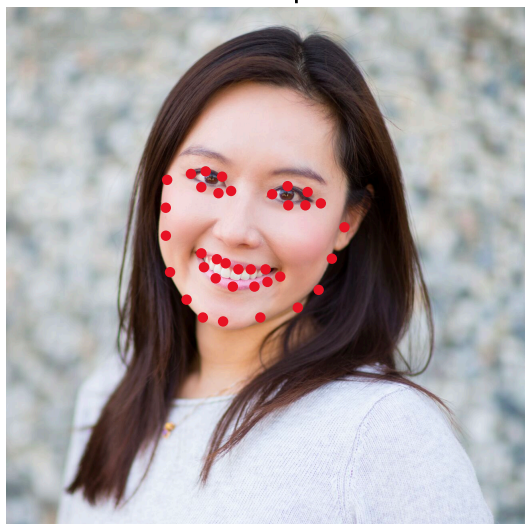
l_{1x}, l_{1y}
l_{2x}, l_{2y}

l_{32x}, l_{32y}

Landmark detection



b_x, b_y, b_h, b_w



l_{1x}, l_{1y}
l_{2x}, l_{2y}

x, y

l_{64x}, l_{64y}

l_{1x}, l_{1y}
l_{2x}, l_{2y}

l_{32x}, l_{32y}

Car detection example

Car detection example

Training set:

x	y
	1

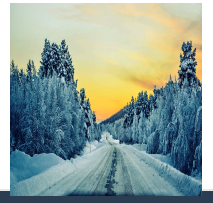
Car detection example

Training set:

x	y
	1
	1

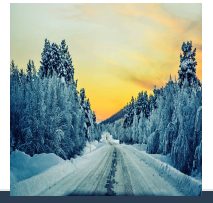
Car detection example

Training set:

x	y
	1
	1
	1
	0
	0

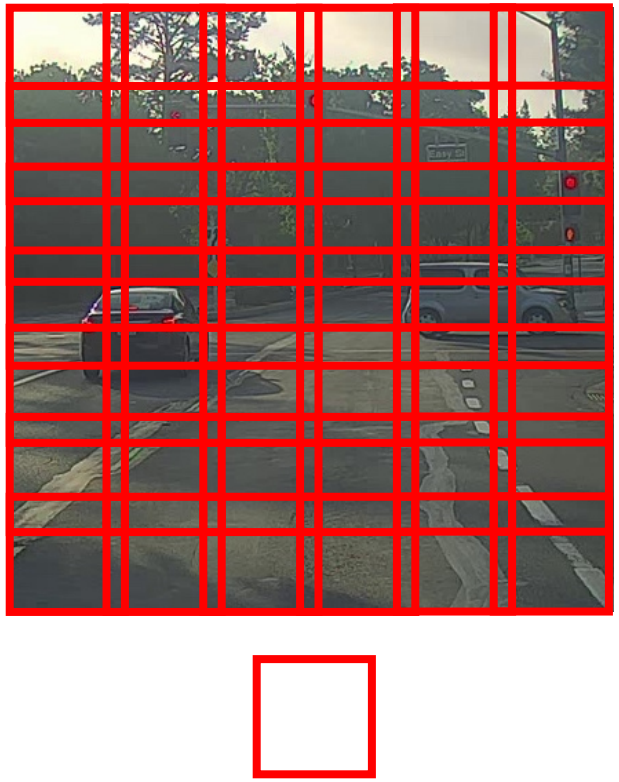
Car detection example

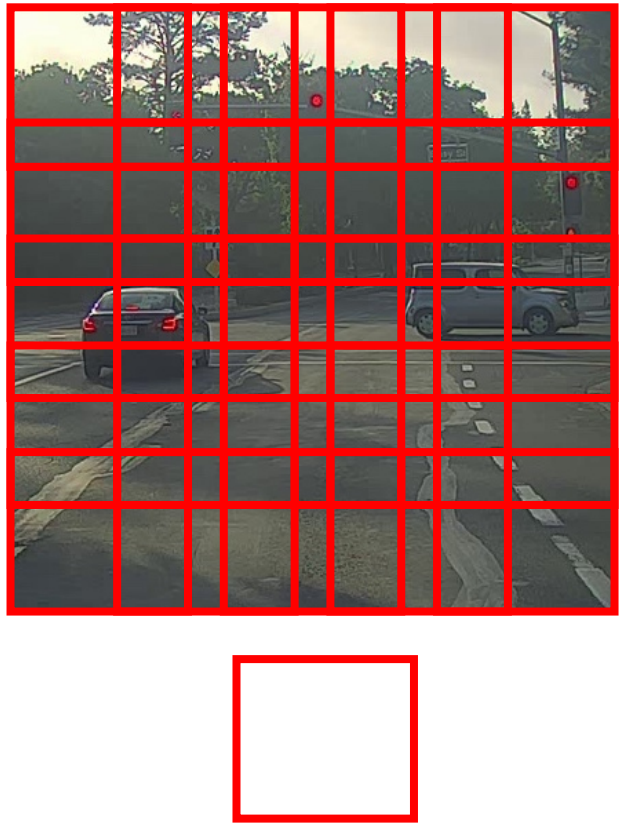
Training set:

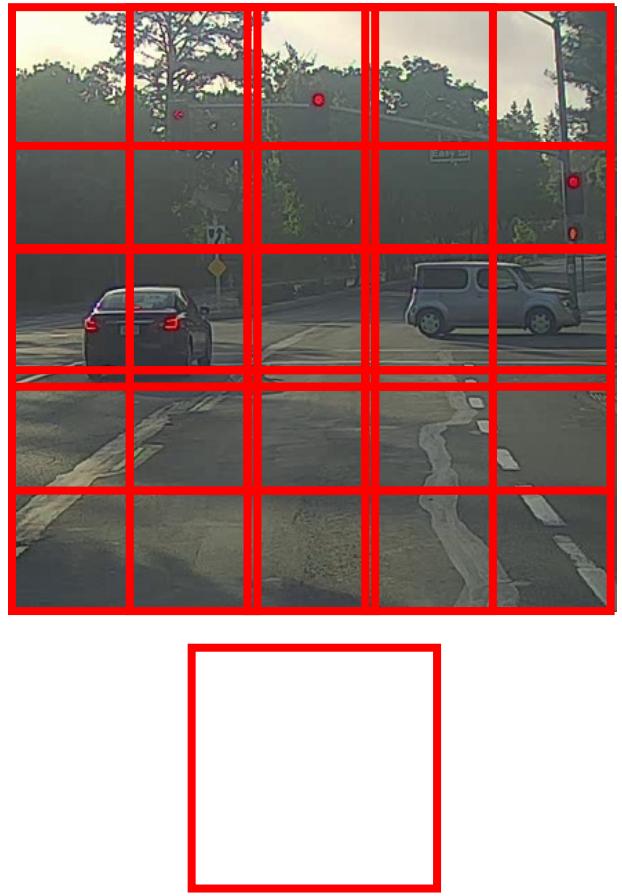
x	y
	1
	1
	1
	0
	0

→ ConvNet → y

Sliding windows detection





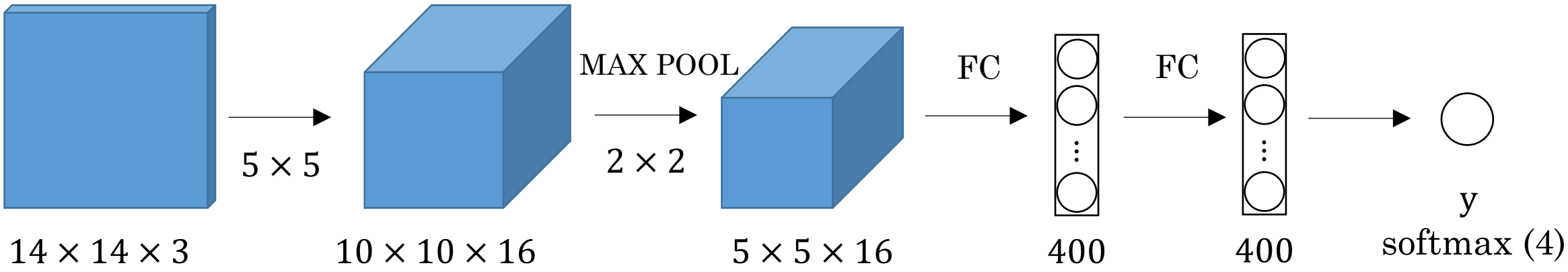


Sliding windows detection

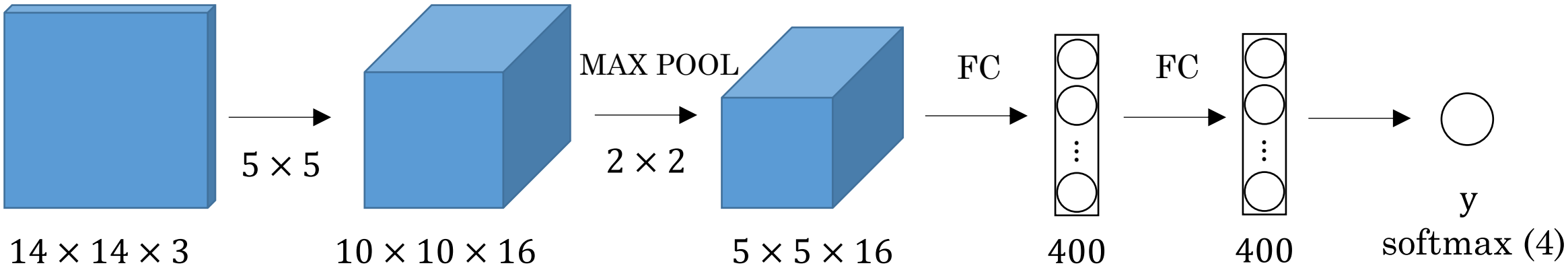
Traditional CV: classification algorithms simple -> fast
Deep learning: ConvNet are computationally heavy

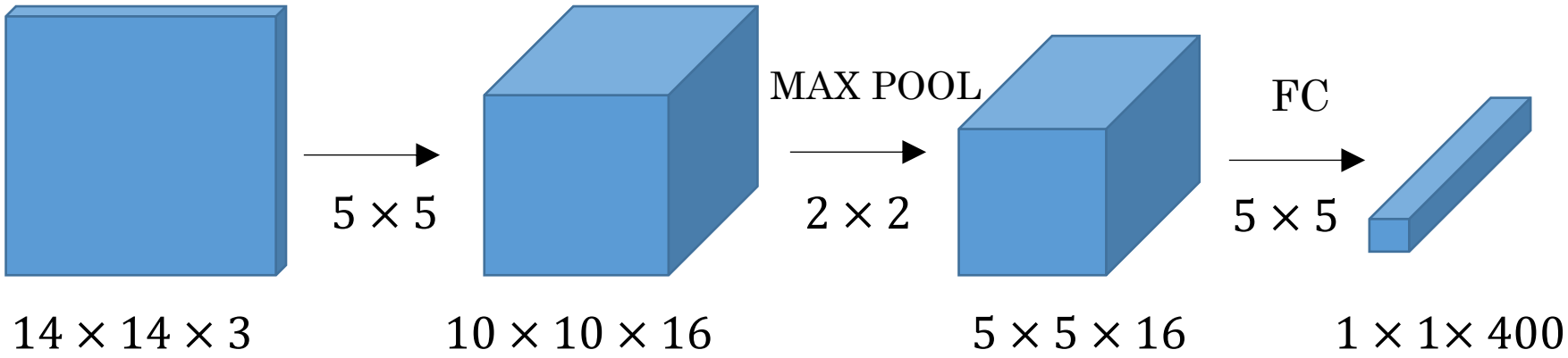
Need some solution

Turning FC layer into convolutional layers

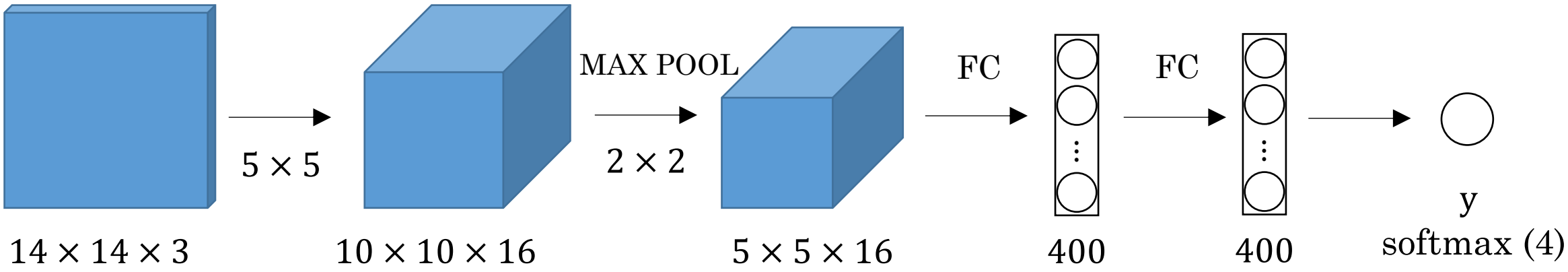


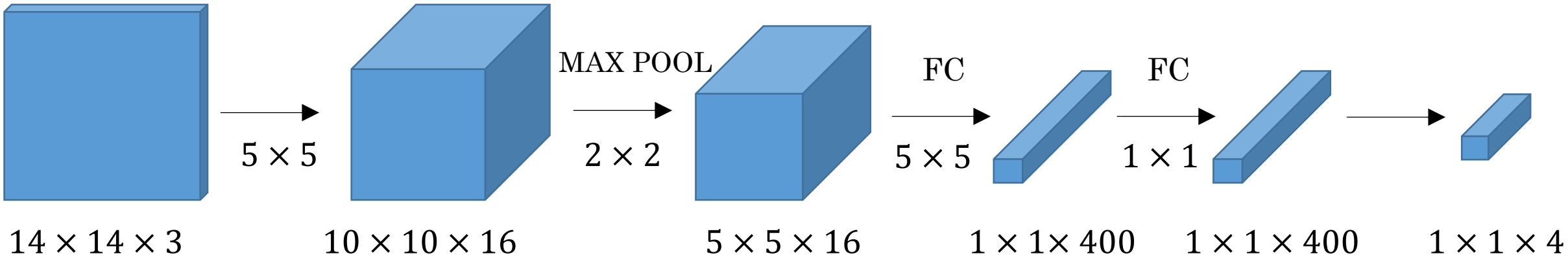
Turning FC Layer Into Convolutional Layers





Turning FC Layer Into Convolutional Layers





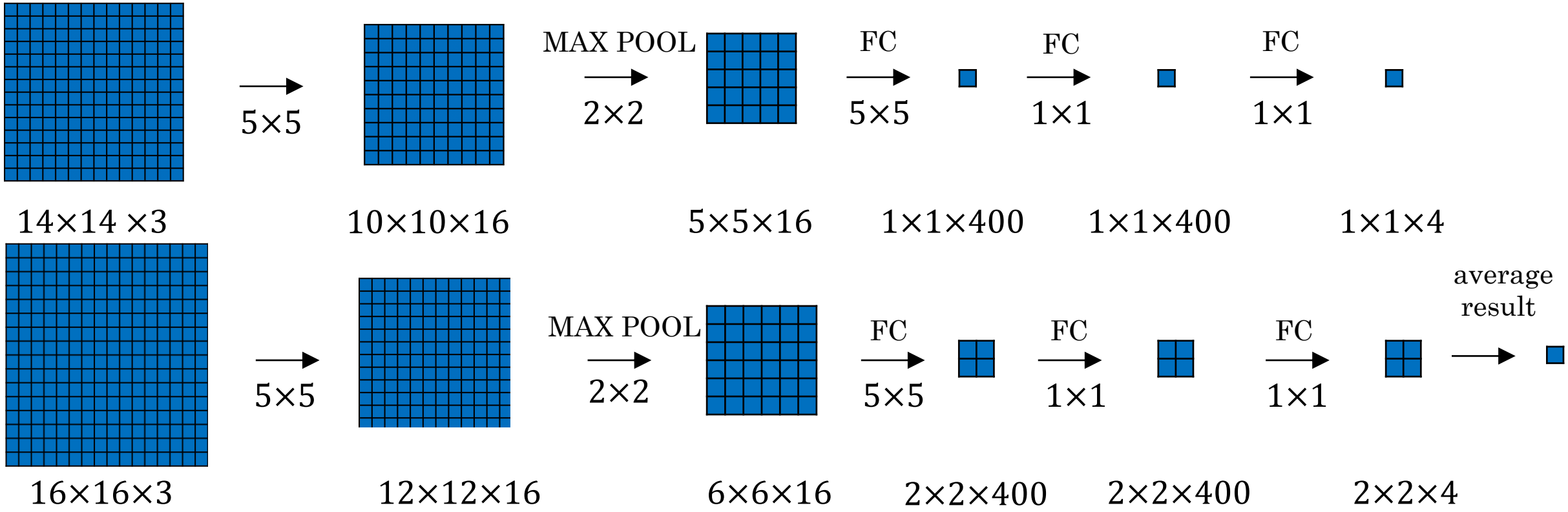
Turning FC Layer Into Convolutional Layers

Why would we need to replace FC layers with Conv Layers?

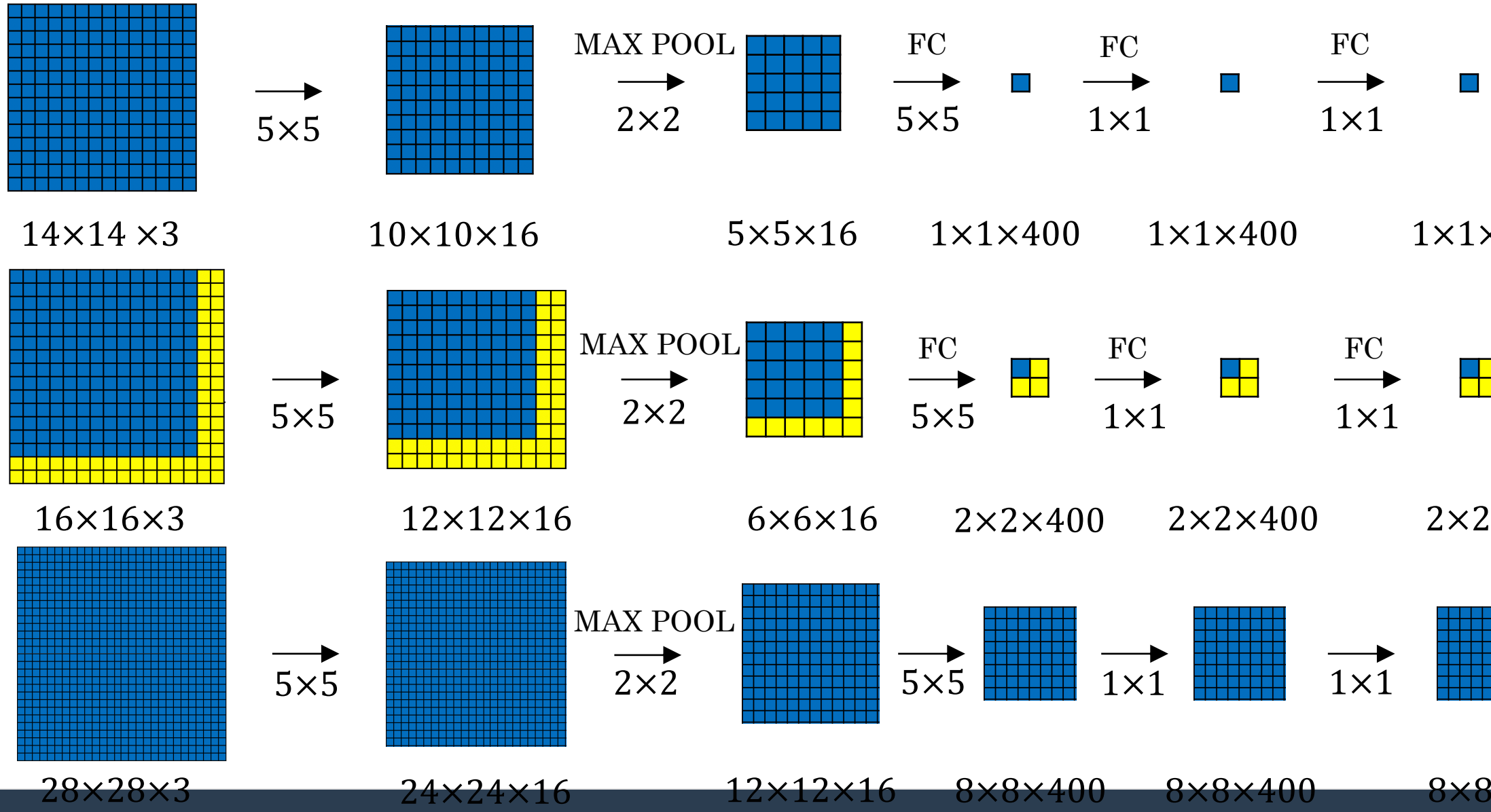
Turning FC Layer Into Convolutional Layers

Why would we need to replace FC layers with Conv Layers?

This allows you to treat the convolutional neural network as one end to end filter. You can then spatially apply the neural net as a convolution to images larger than the original training image size, getting a spatially dense output. Then you average these outputs.

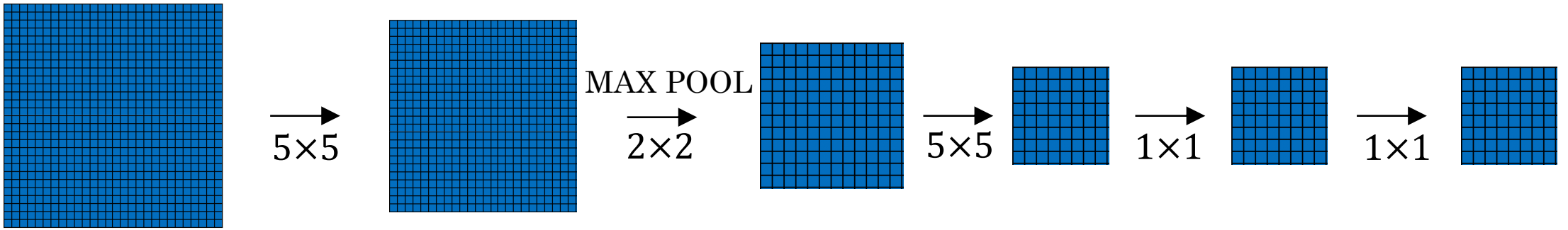


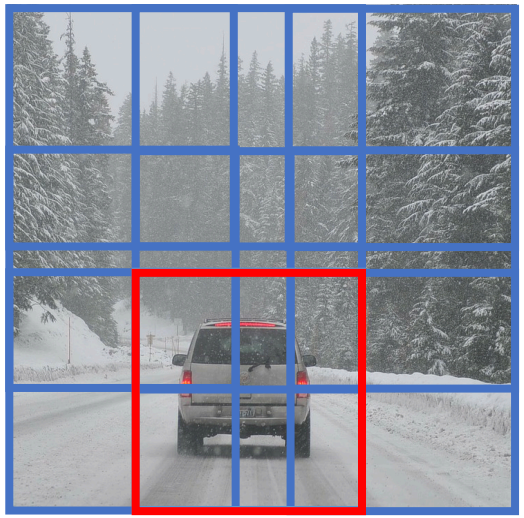
Convolution implementation of sliding windows

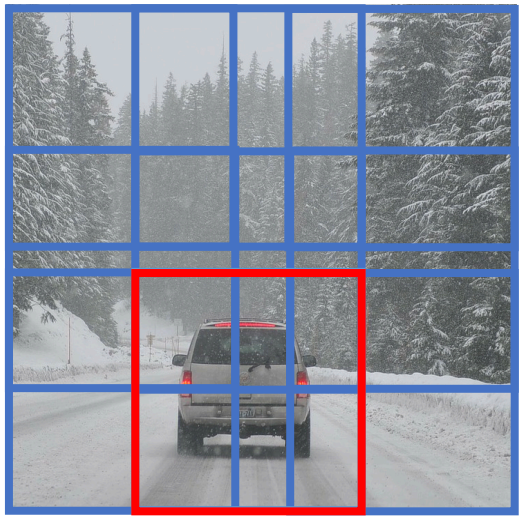


[Sermanet et al., 2014, OverFeat: Integrated recognition, localization and detection using convolutional networks]

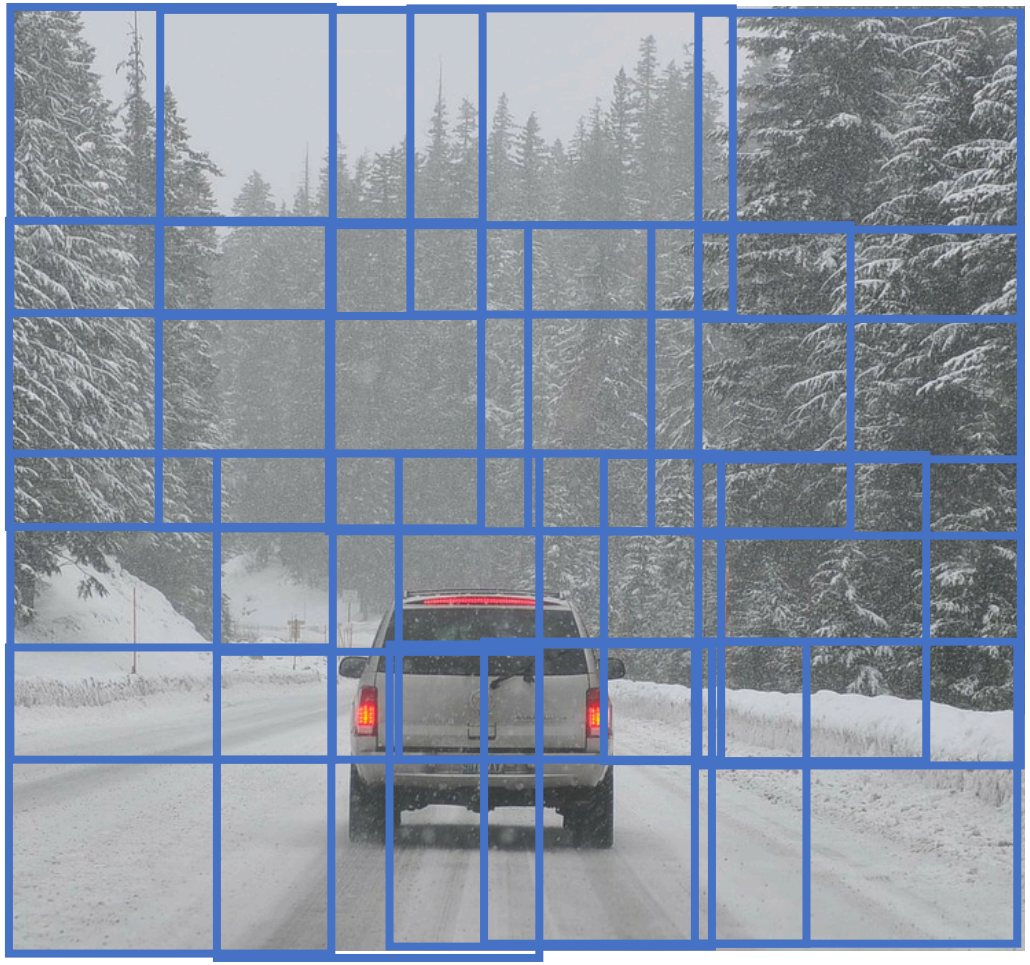
Convolution implementation of sliding windows



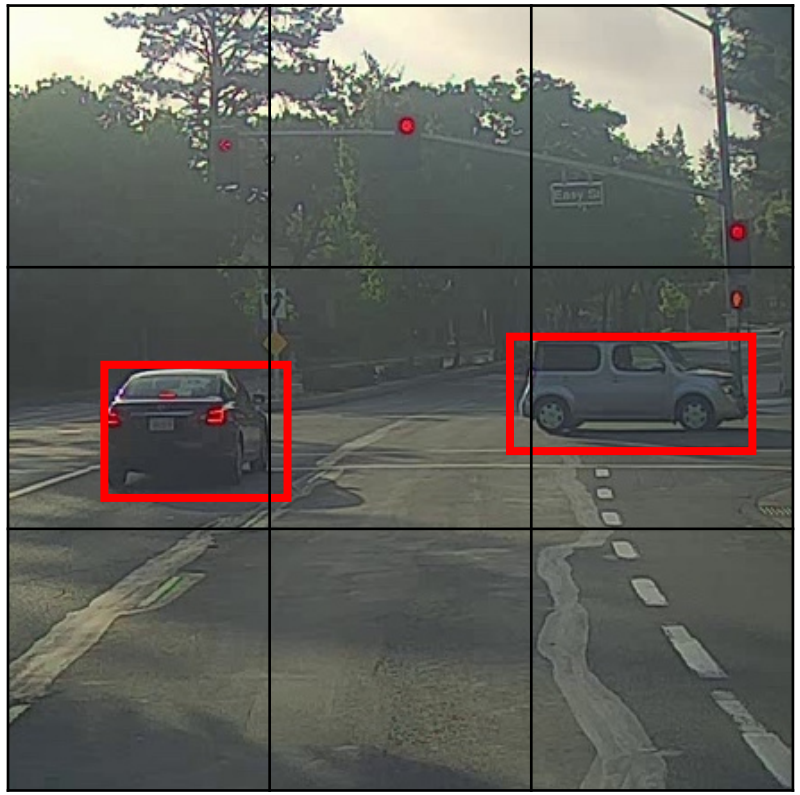




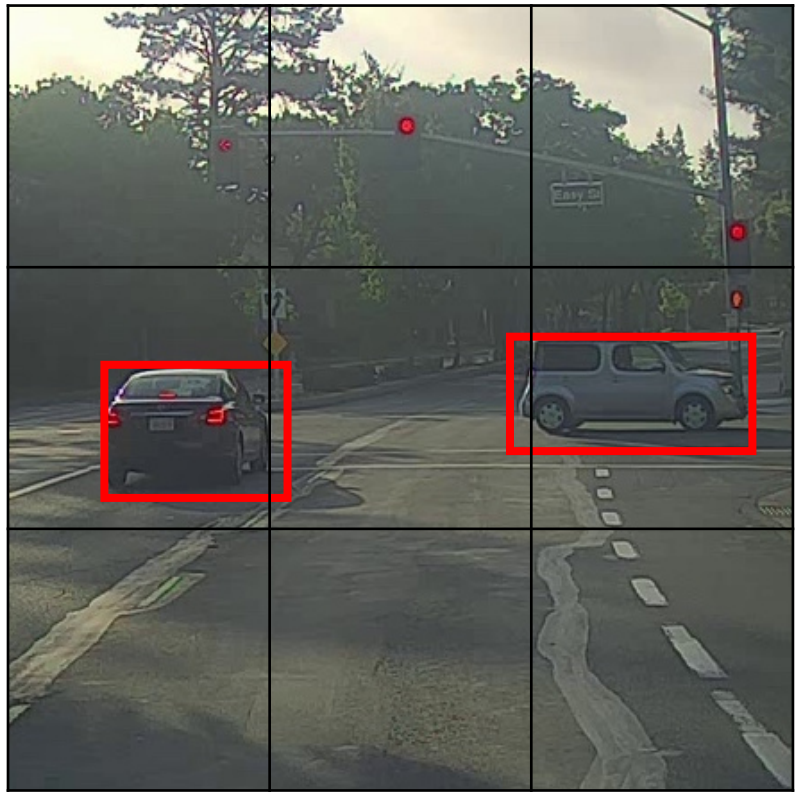
Output accurate bounding boxes



YOLO algorithm



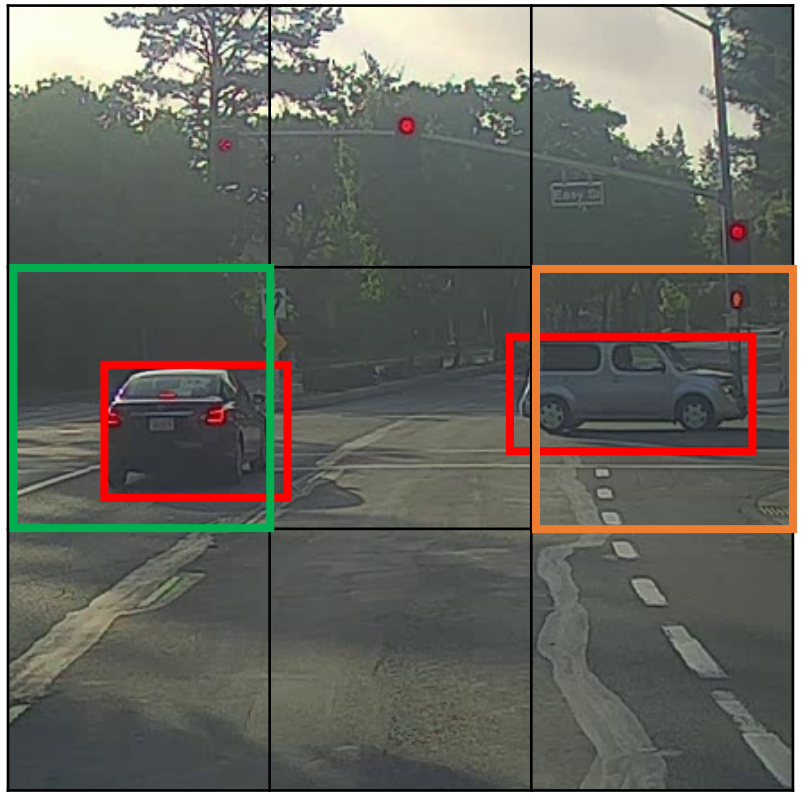
YOLO algorithm



Labels for training
for each grid cell:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

YOLO algorithm

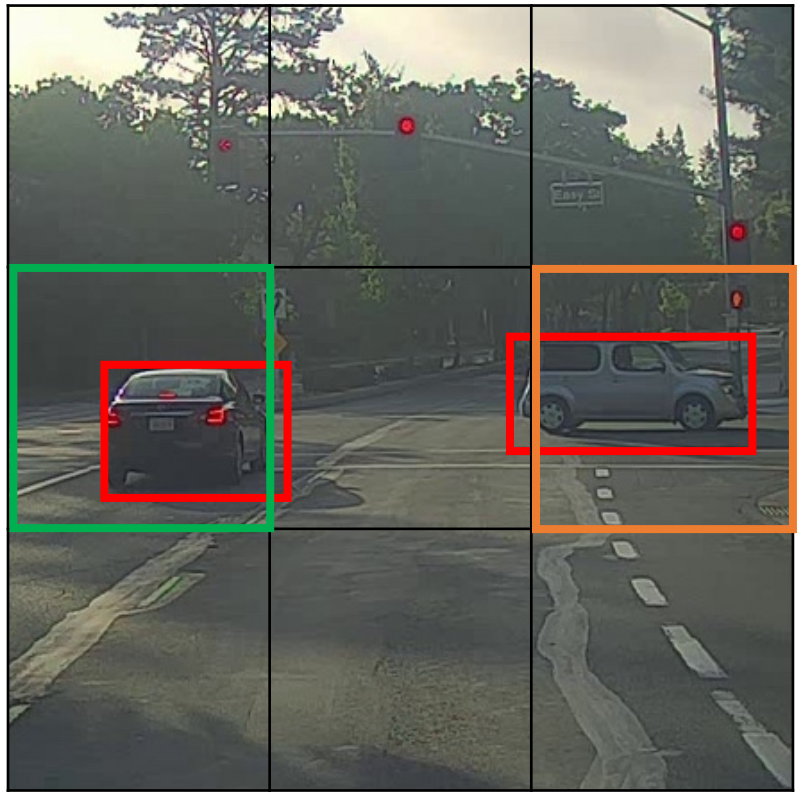


Labels for training
for each grid cell:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

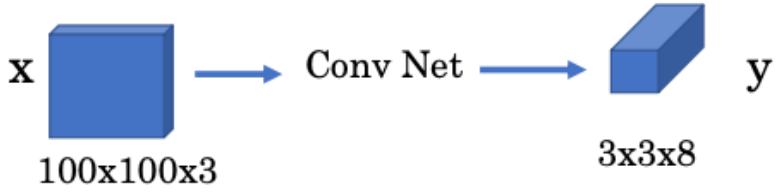
YOLO algorithm



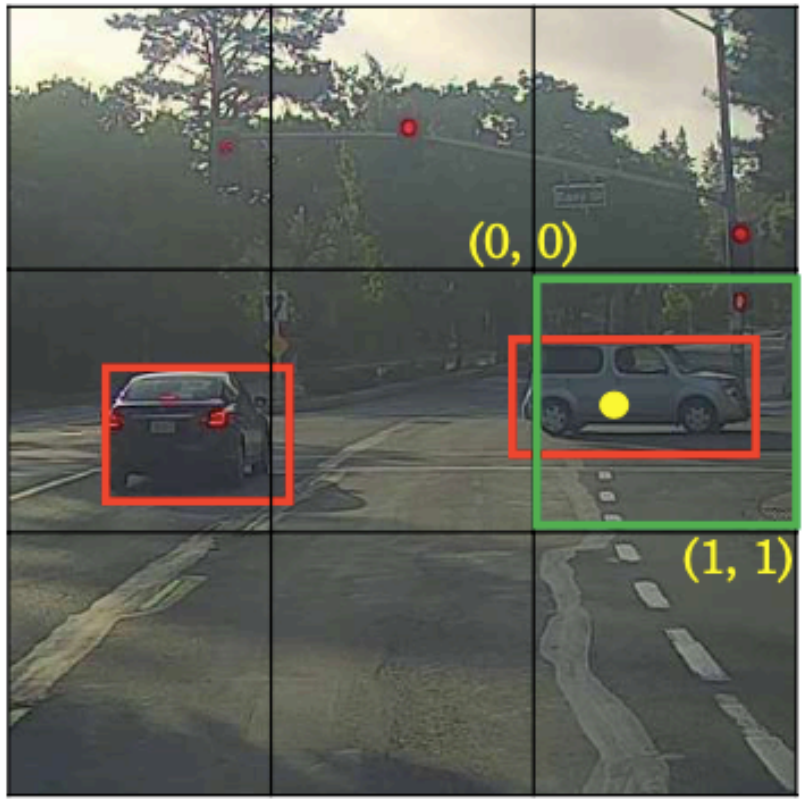
Labels for training
for each grid cell:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

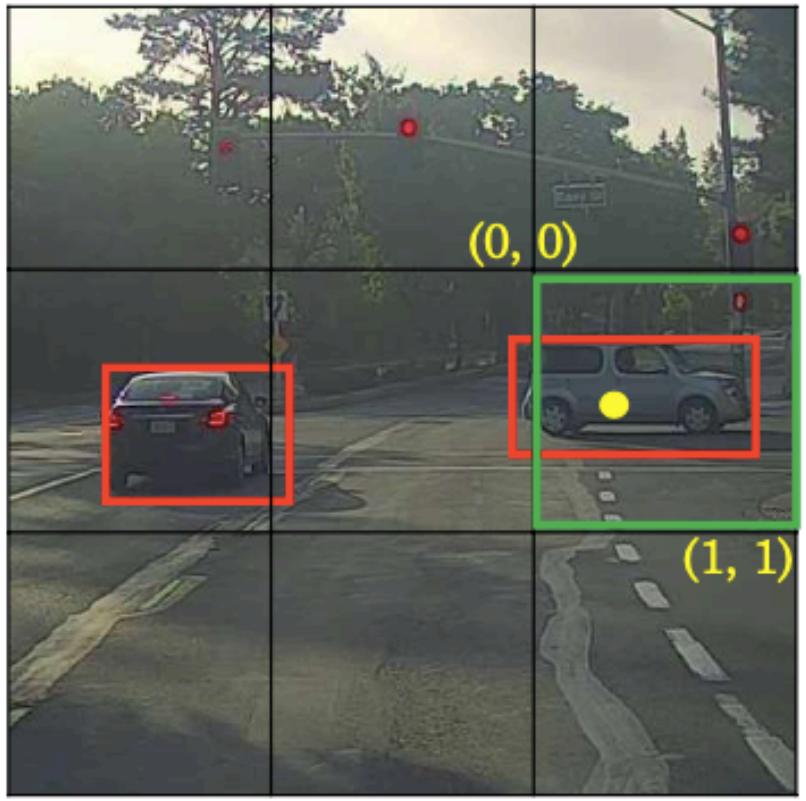


Specifying the Bounding Boxes



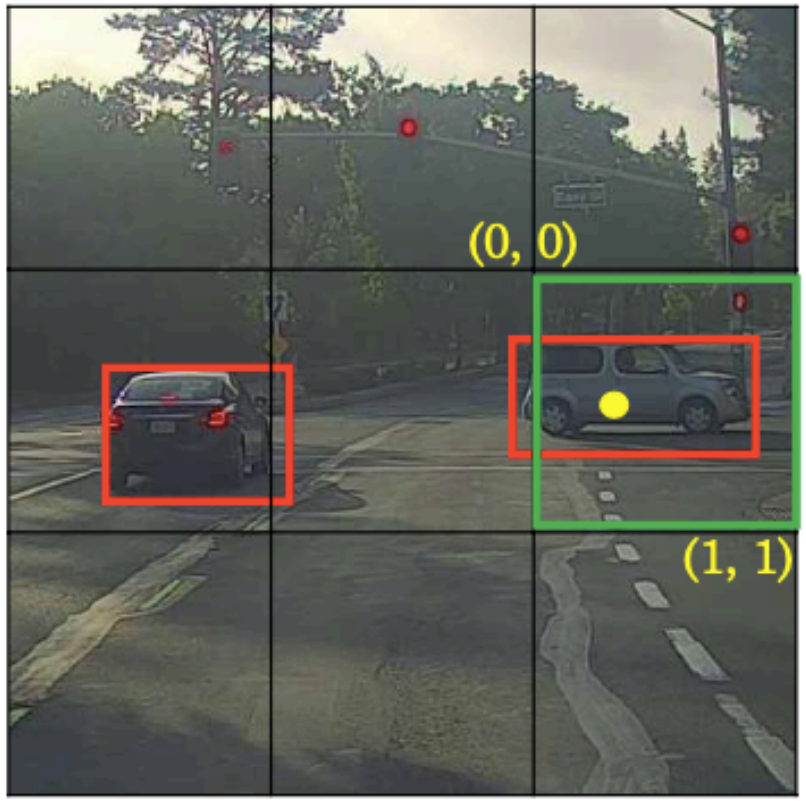
$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_w \\ b_h \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Specifying the Bounding Boxes



$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_w \\ b_h \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{matrix} \\ 0.4 \\ 0.3 \\ 0.9 \\ 0.5 \\ \\ \\ \end{matrix}$$

Specifying the Bounding Boxes

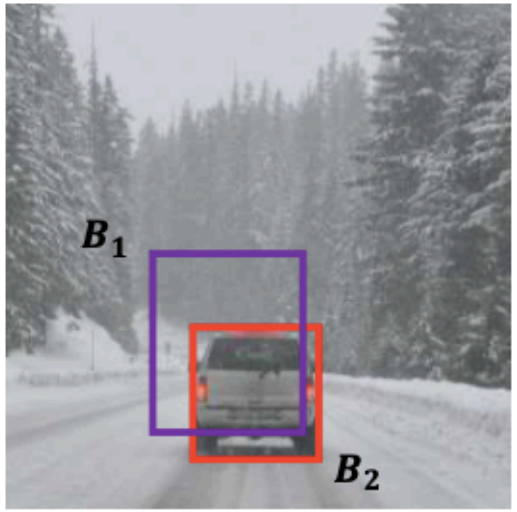


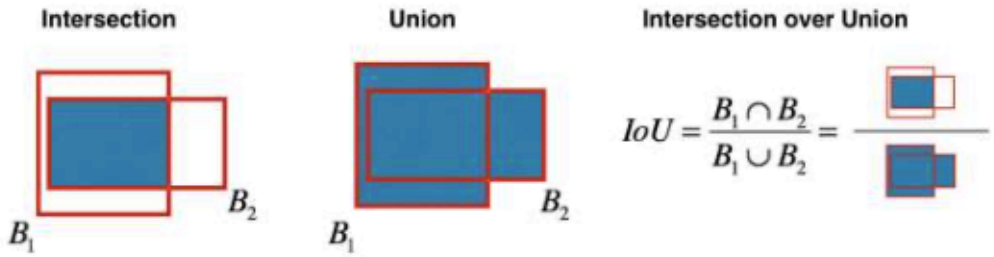
$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_w \\ b_h \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{matrix} 0.4 \\ 0.3 \\ 0.9 \\ 0.5 \end{matrix}$$

between 0 and 1

can be greater than 1

Evaluating Object Localization

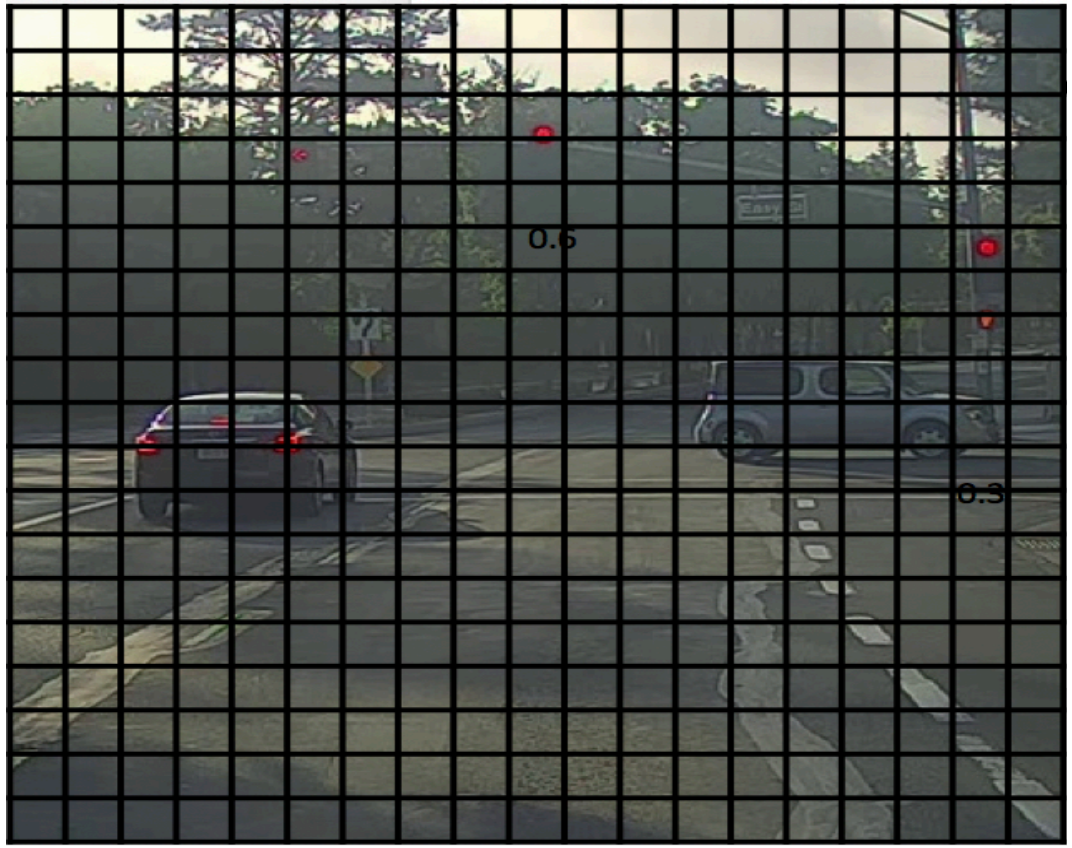




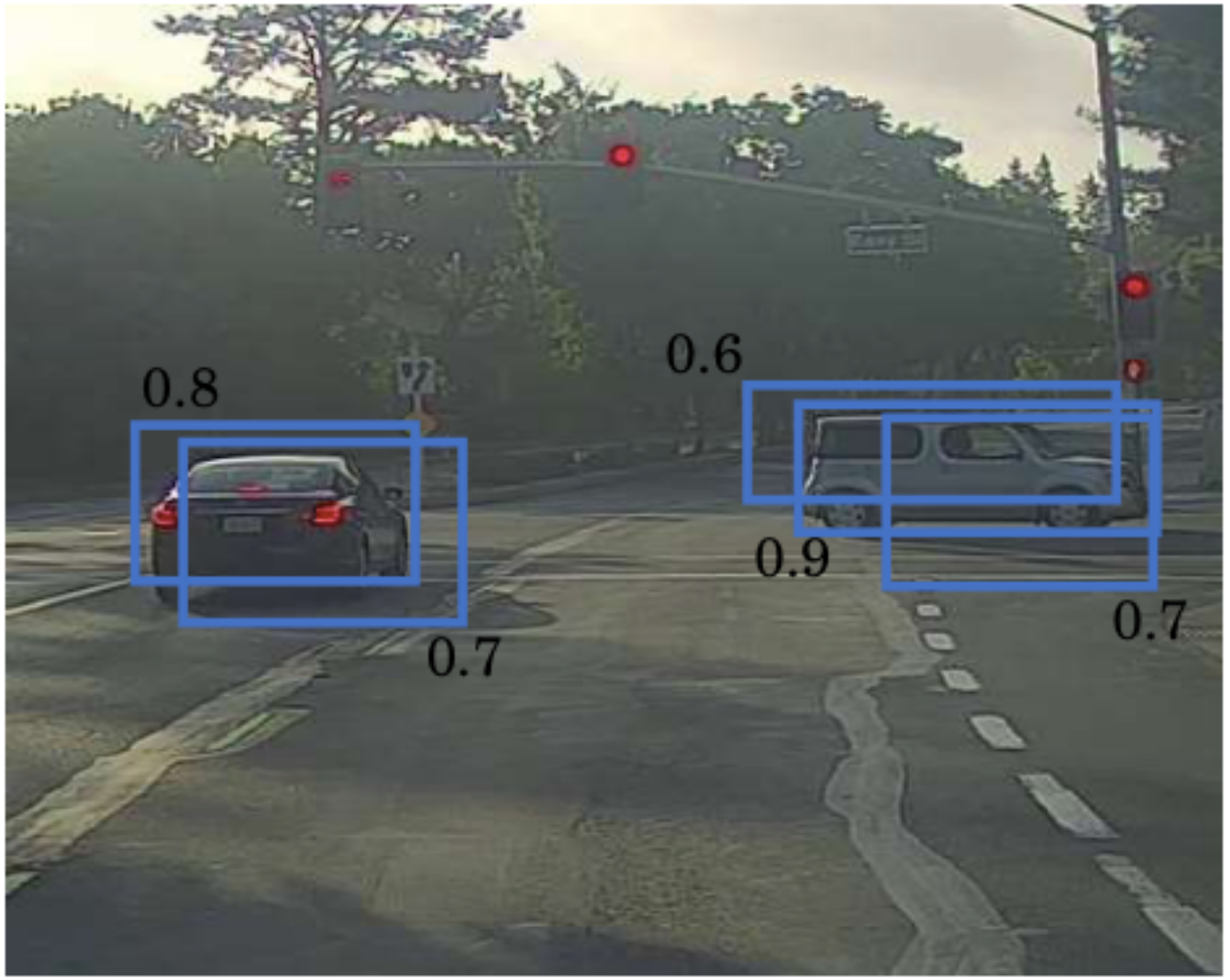
"Correct" if IoU ≥ 0.5

More generally, IoU is a measure of the overlap between two bounding boxes.

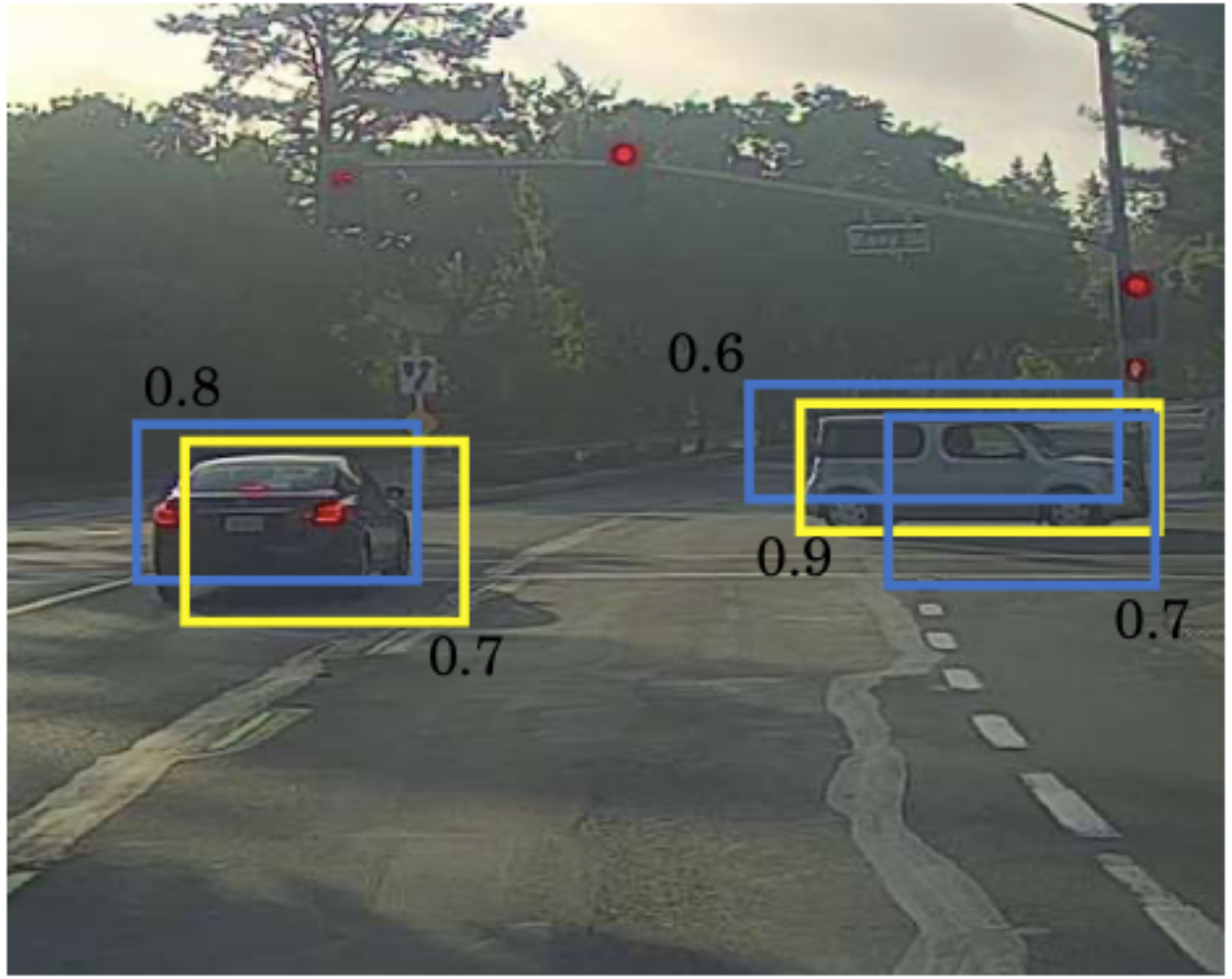
Non-max Suppression Example



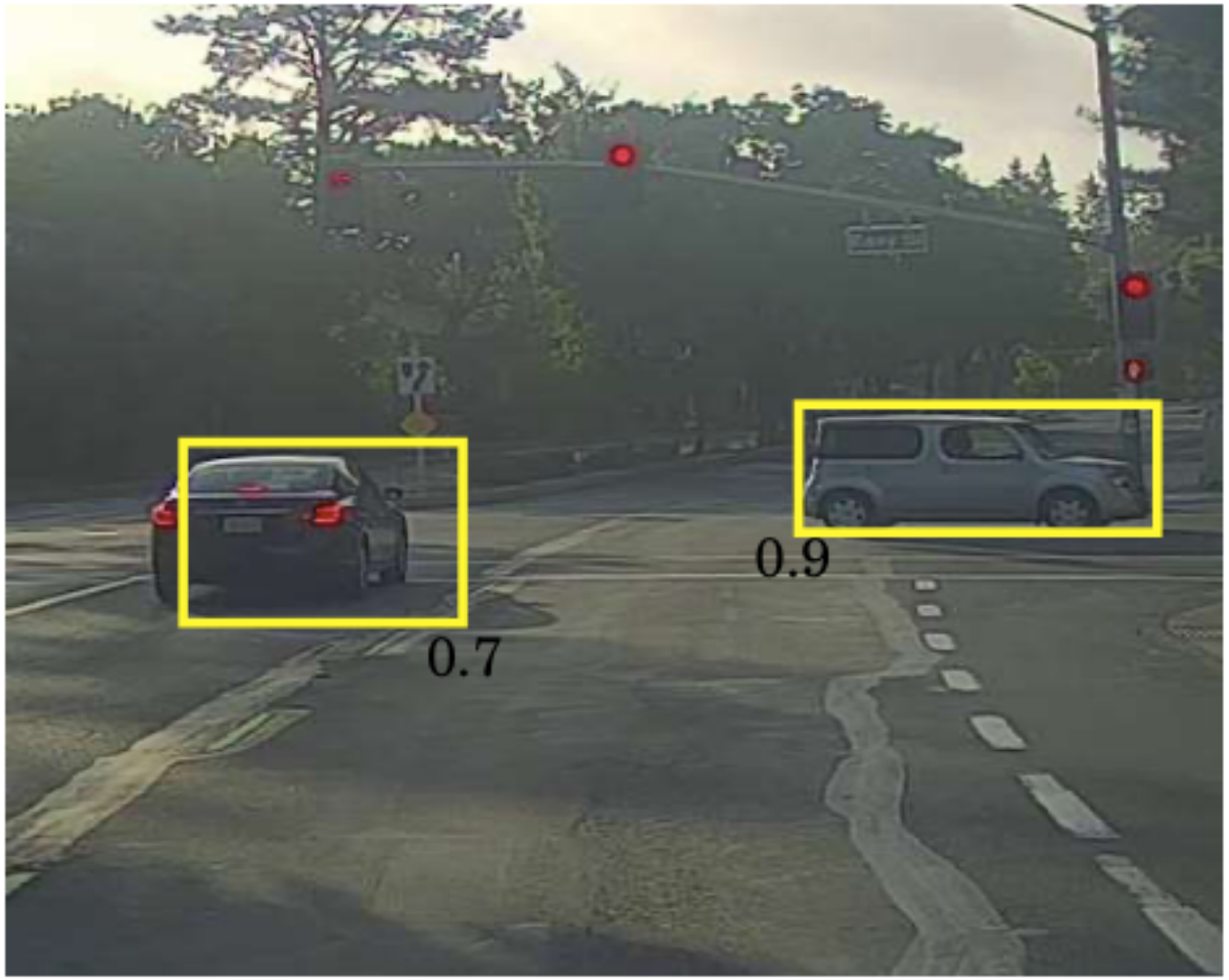
Non-max Suppression Example



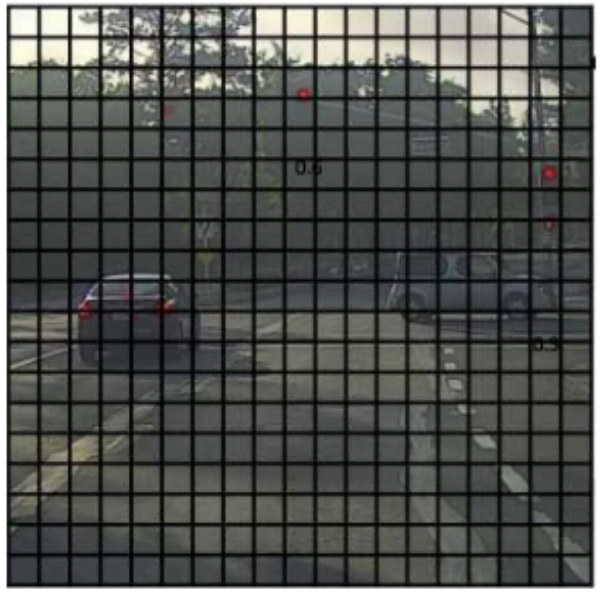
Non-max Suppression Example



Non-max Suppression Example



Non-max Suppression Algorithm



19× 19

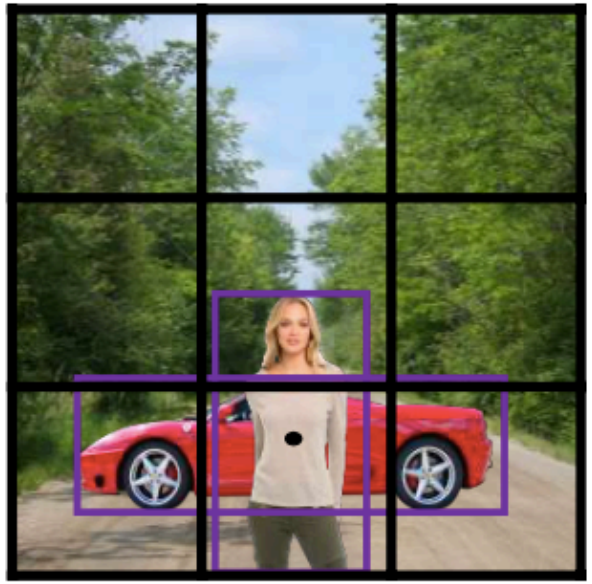
Each output prediction is: $\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

Discard all boxes with $p_c \leq 0.6$

While there are any remaining boxes:

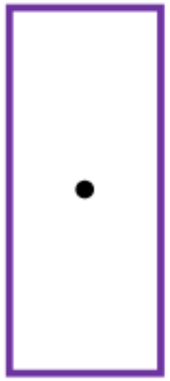
- Pick the box with the largest p_c
  Output that as a prediction.
- Discard any remaining box with
  IoU ≥ 0.5 with the box output
  in the previous step

Overlapping Objects



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Anchor box 1:



Anchor box 2:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

anchor box 1

anchor box 2

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Anchor Box Algorithm

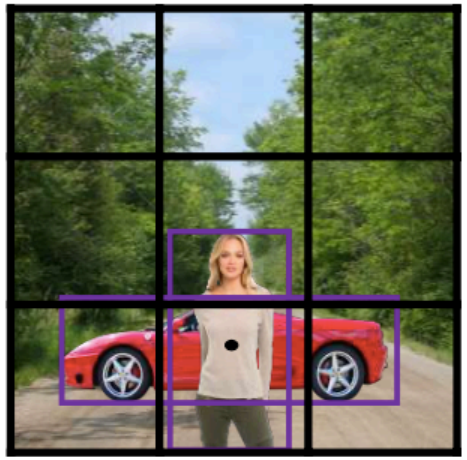
Previously:

Each object in training image is assigned to grid cell that contains that object's midpoint.

With two anchor boxes:

Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

Anchor Box Example

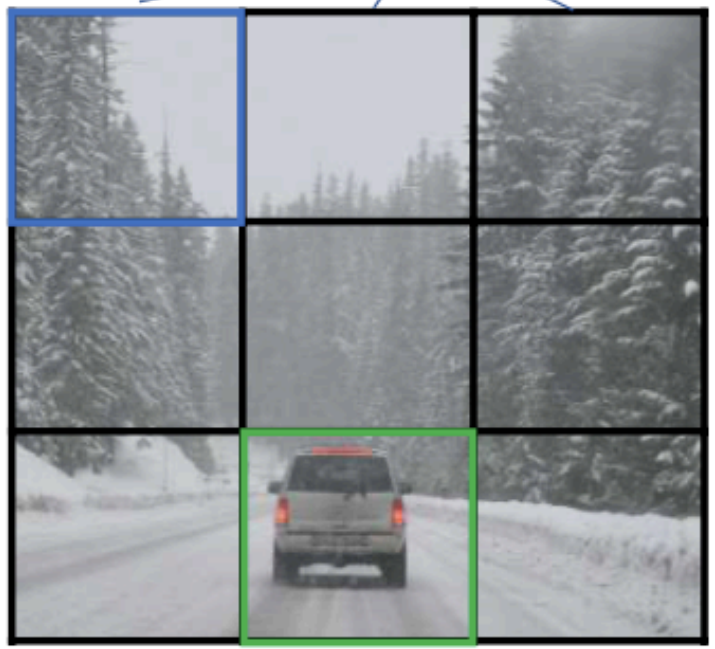


Anchor box 1:

Anchor box 2:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \text{Car, human} \quad y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{Car only} \quad y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Training



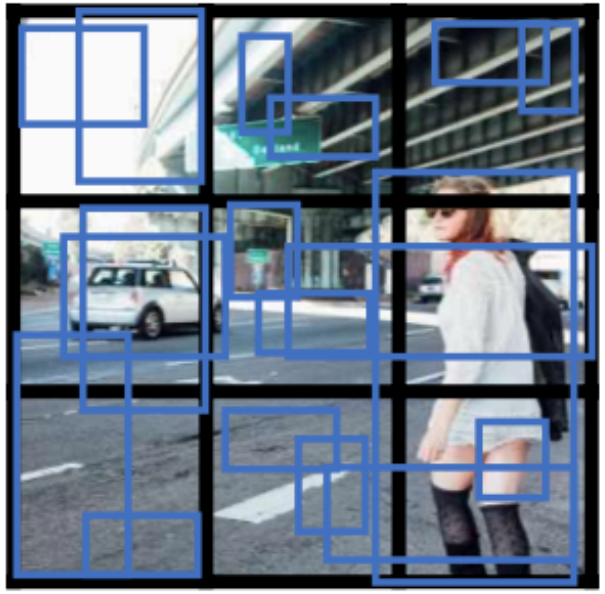
1 - pedestrian
2 - car
3 - motorcycle

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \begin{bmatrix} ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

y is $3 \times 3 \times 2 \times 8$

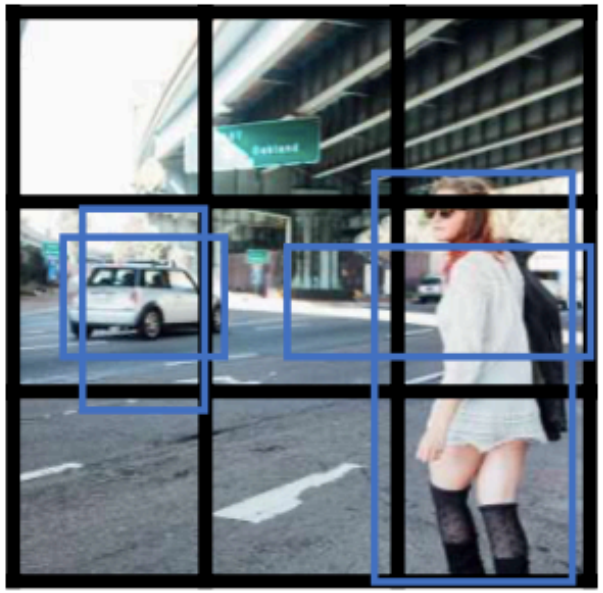
[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Inference



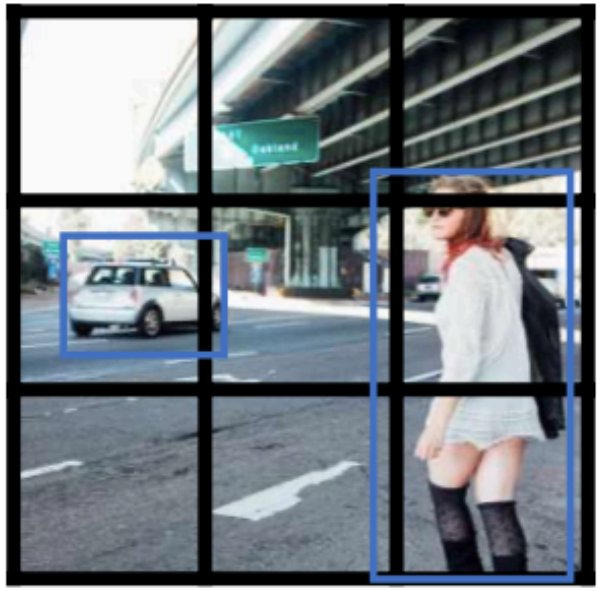
- For each grid call, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

Inference



- For each grid call, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

Inference



- For each grid call, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

Resources Used

- Deeplearning.ai by Andrew Ng