Traffic Signs Detection and Classification

Computer Vision, YOLO

Introduction

Motivation: Addressing the critical need for improved road safety and adherence to traffic regulations through advanced technology.

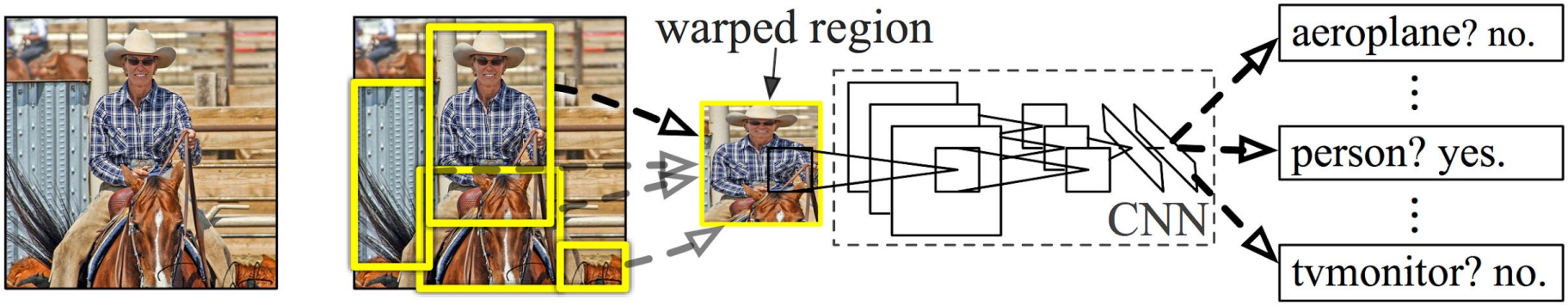
Objective: Introducing YOLO (You Only Look Once) as a powerful tool for automated traffic sign detection, not only to prevent accidents but also to ensure compliance with speed limits and other traffic regulations.

Significance: Highlighting the potential impact of YOLO-based systems in reducing accidents, and preventing traffic violations by detecting traffic and road signs.

Content

1. Review: R-CNN
2. YOLO: -- Detection Procedure
   -- Network Design
3. Code Review
4. Model Metrics
5. Flask App
6. Challenges and Future
7. Learnings Applied

R-CNN: *Regions with CNN features*



1. Input image

2. Extract region proposals (~2k)

3. Compute CNN features

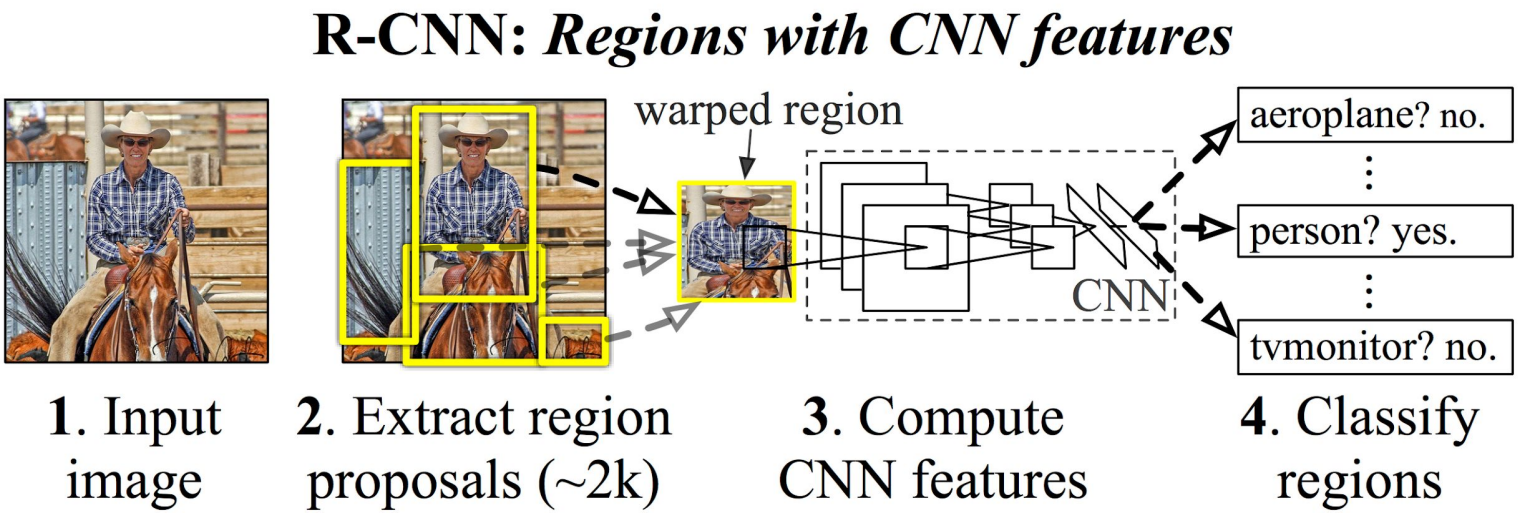
4. Classify regions

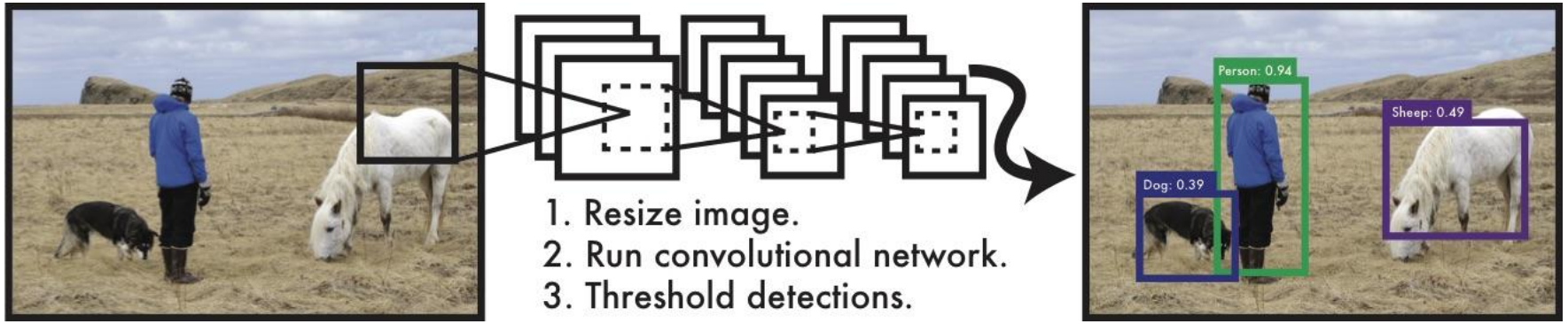
Proposal + Classification

Shortcomings

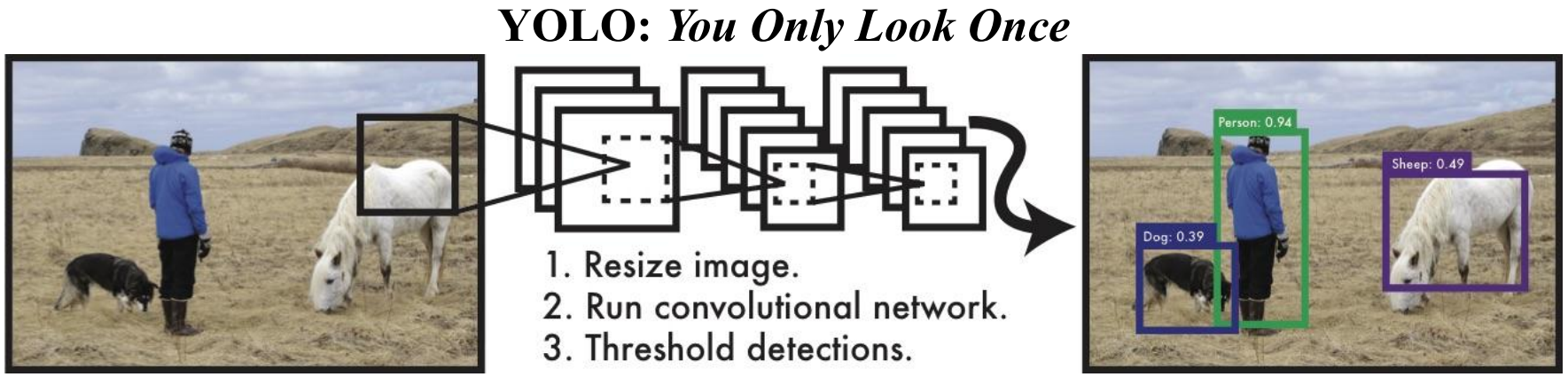
1. Slow, impossible for real-time detection

2. Hard to optimize




1. Resize image.
2. Run convolutional network.
3. Threshold detections.
Person: 0.94
Dog: 0.39
Sheep: 0.49

With YOLO, you only look once at an image to perform detection



Object detection using YOLO

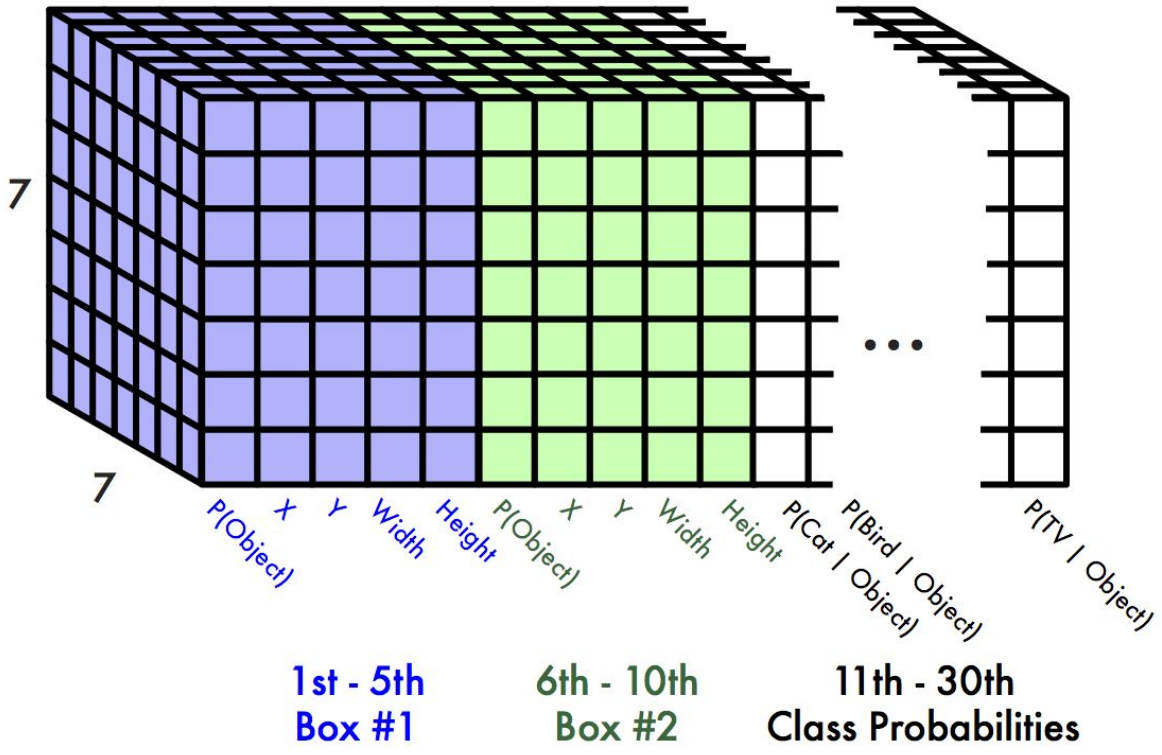
Computer vision is a field of artificial intelligence that focuses on teaching computers to interpret and understand visual information. One popular and powerful technique used in computer vision for object detection is called YOLO, which stands for "You Only Look Once".

YOLO aims to identify and locate objects in an image or video stream in real-time. Unlike traditional methods that rely on complex pipelines and multiple passes, YOLO takes a different approach by treating object detection as a single **regression** problem.

This parameterization fixes the output size

Each cell predicts:

- For each bounding box:
  - 4 coordinates (x, y, w, h)
  - 1 confidence value
- Some number of class probabilities

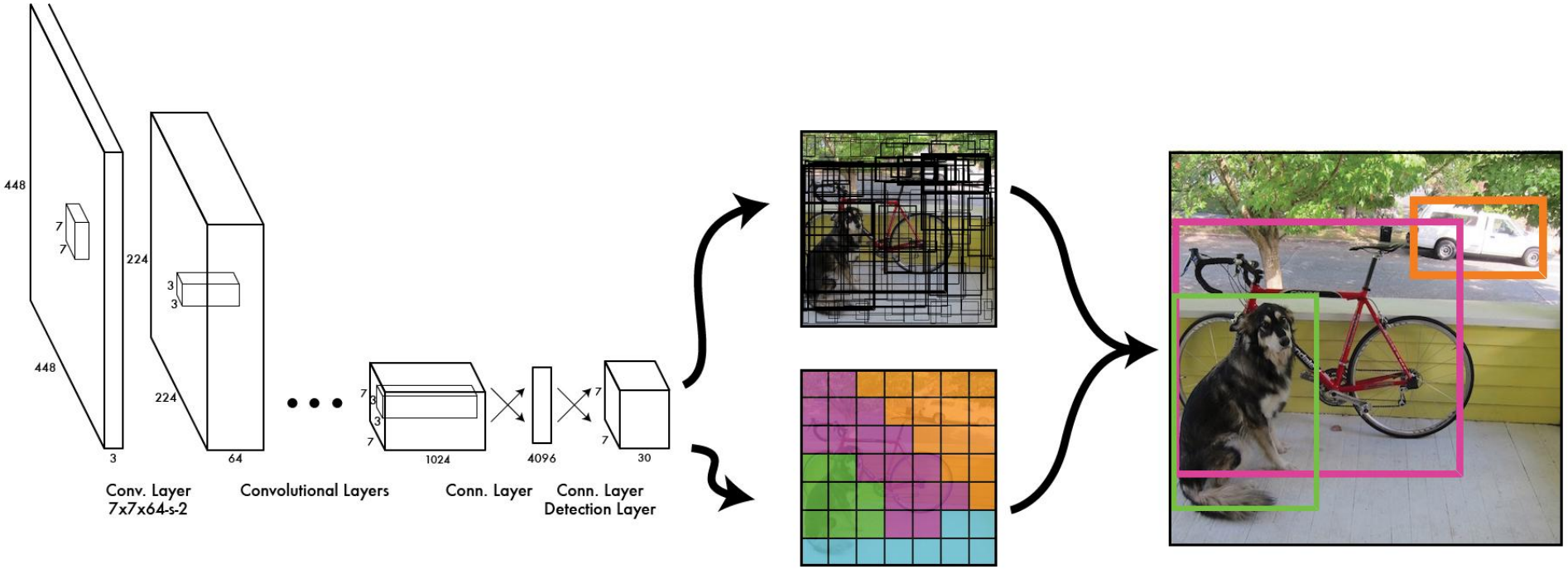


For Pascal VOC:

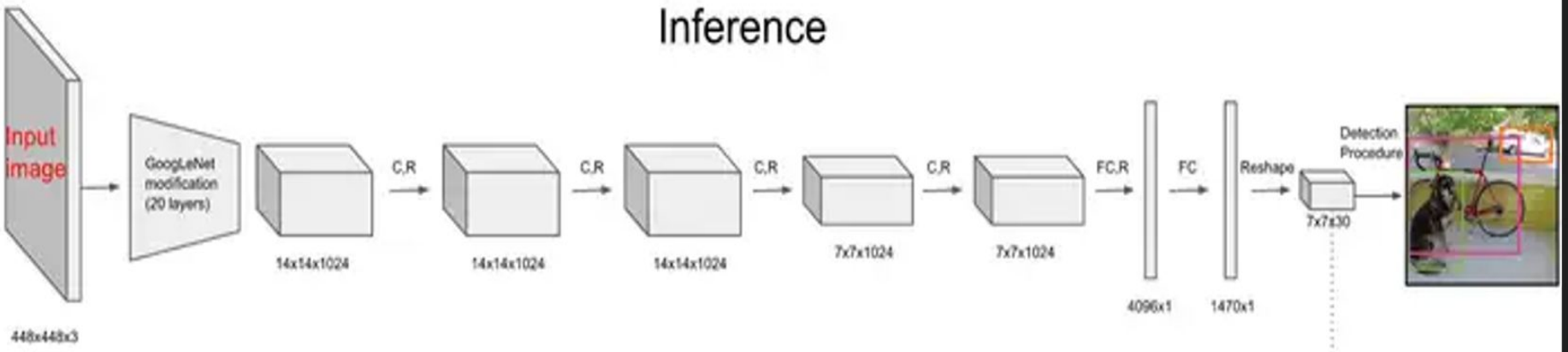
- 7x7 grid
- 2 bounding boxes / cell
- 20 classes

7 x 7 x (2 x 5 + 20) = 7 x 7 x 30 tensor = **1470 outputs**

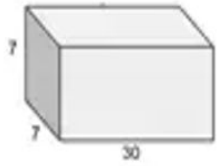
Thus we can train one neural network to be a whole detection pipeline



Inference



Tensor values interpretation



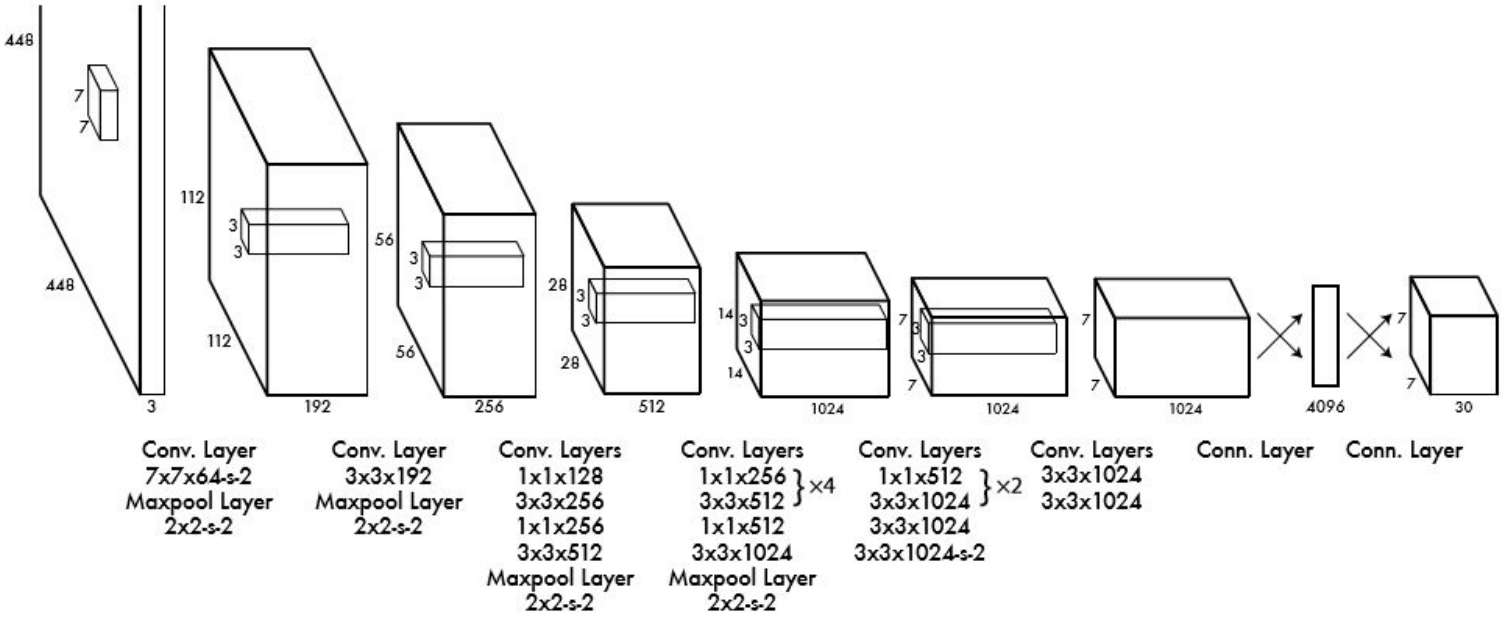
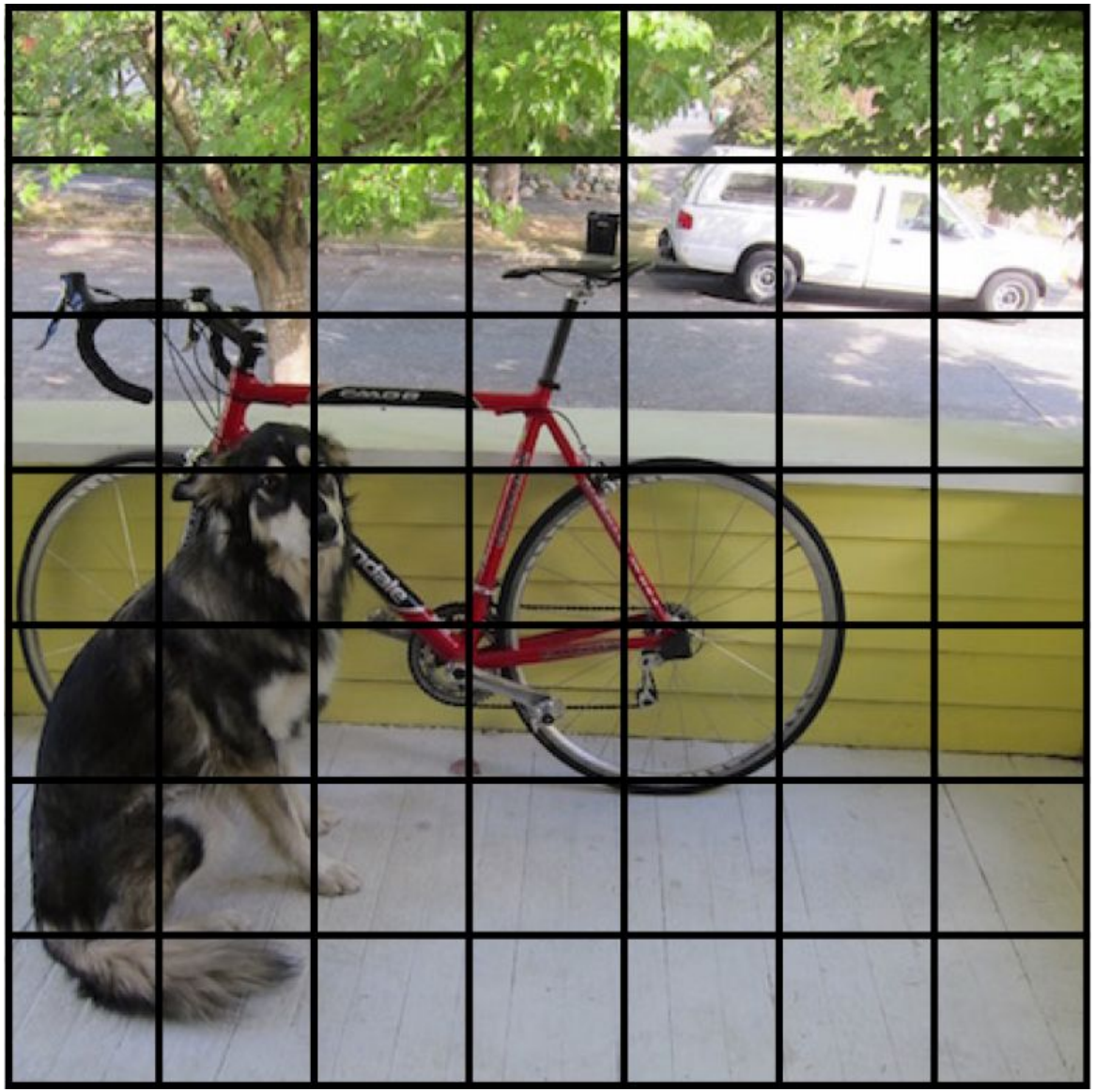
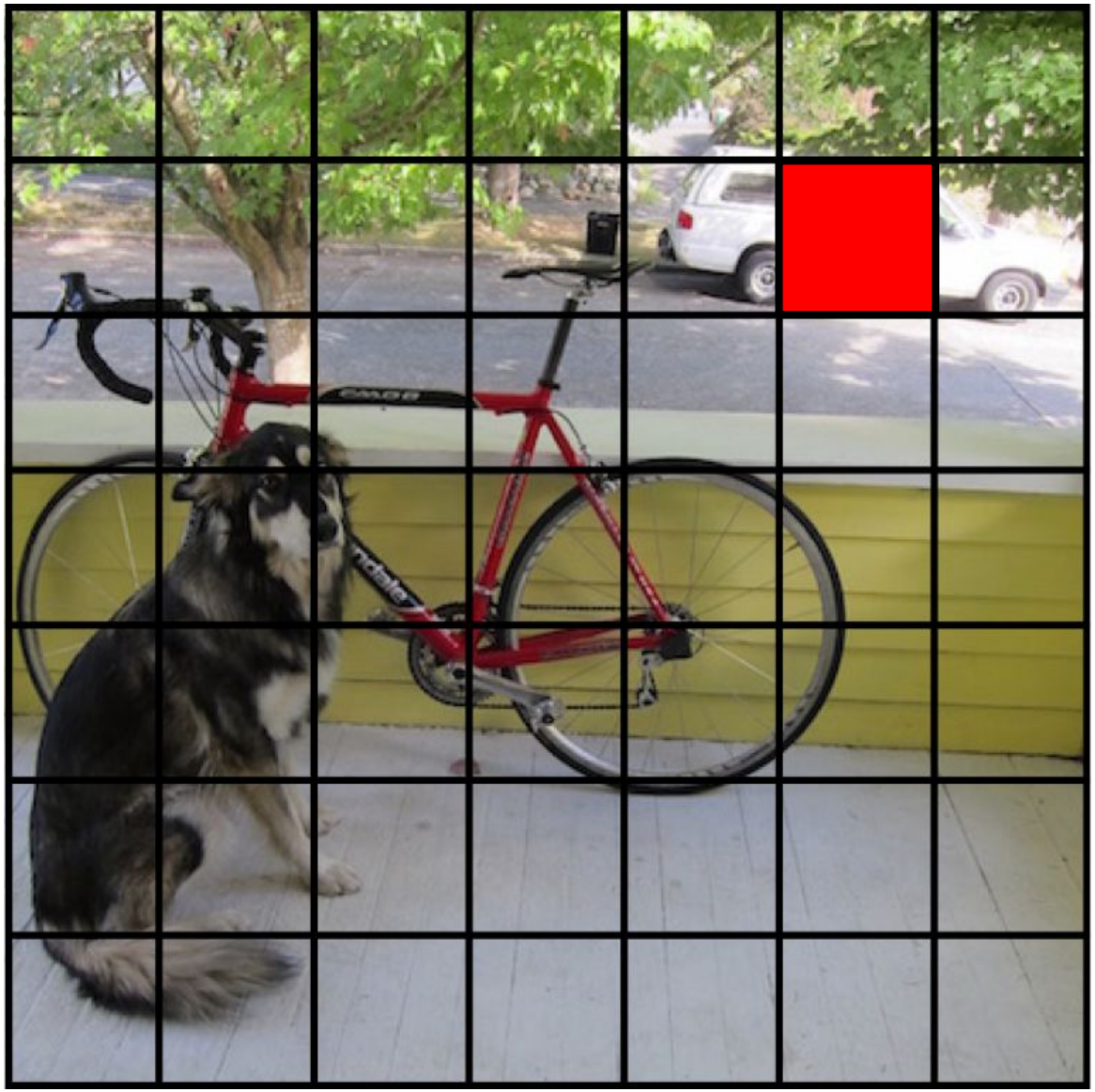


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224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

We split the image into a grid

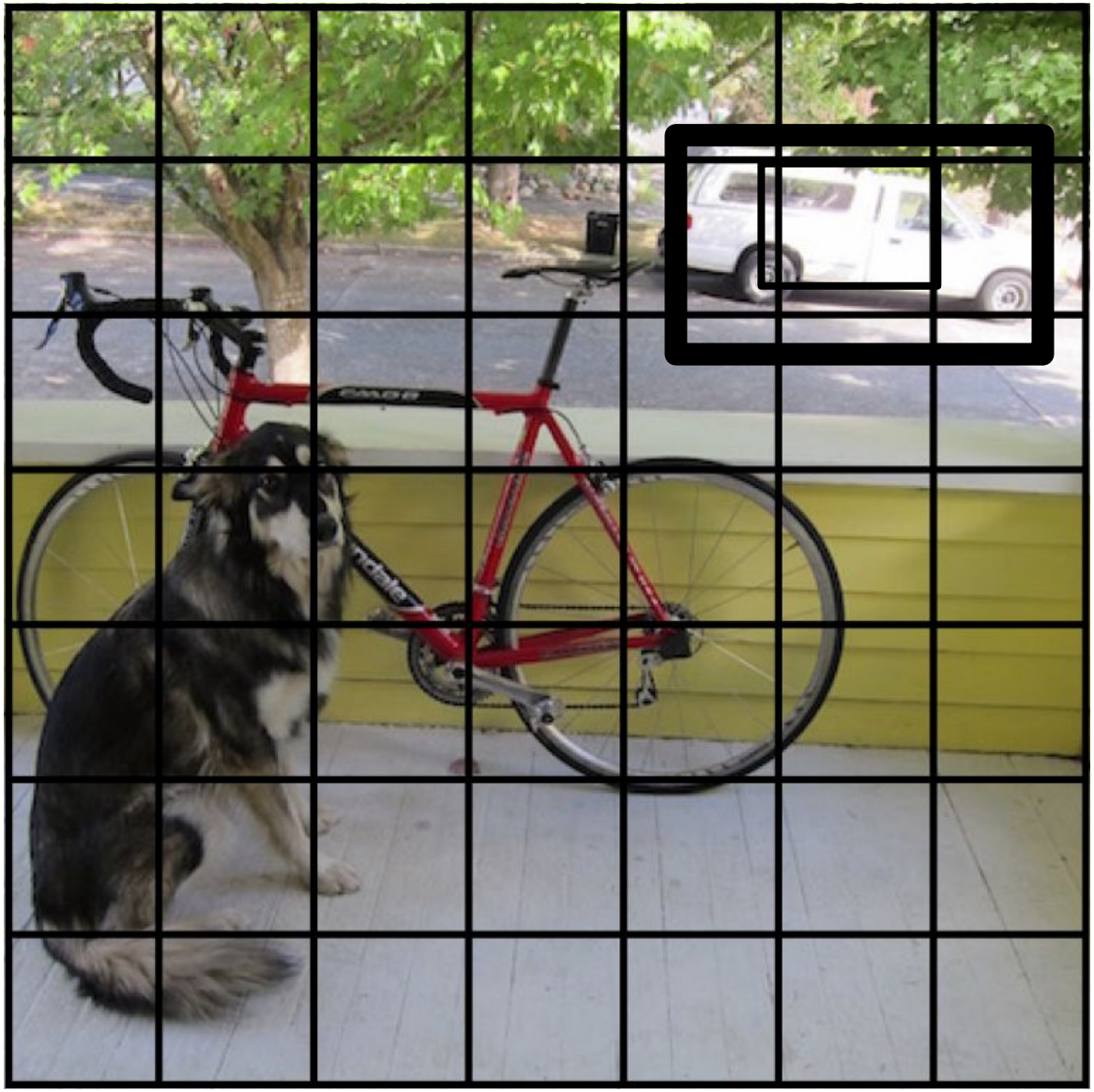


Each cell predicts boxes and confidences: P(Object)

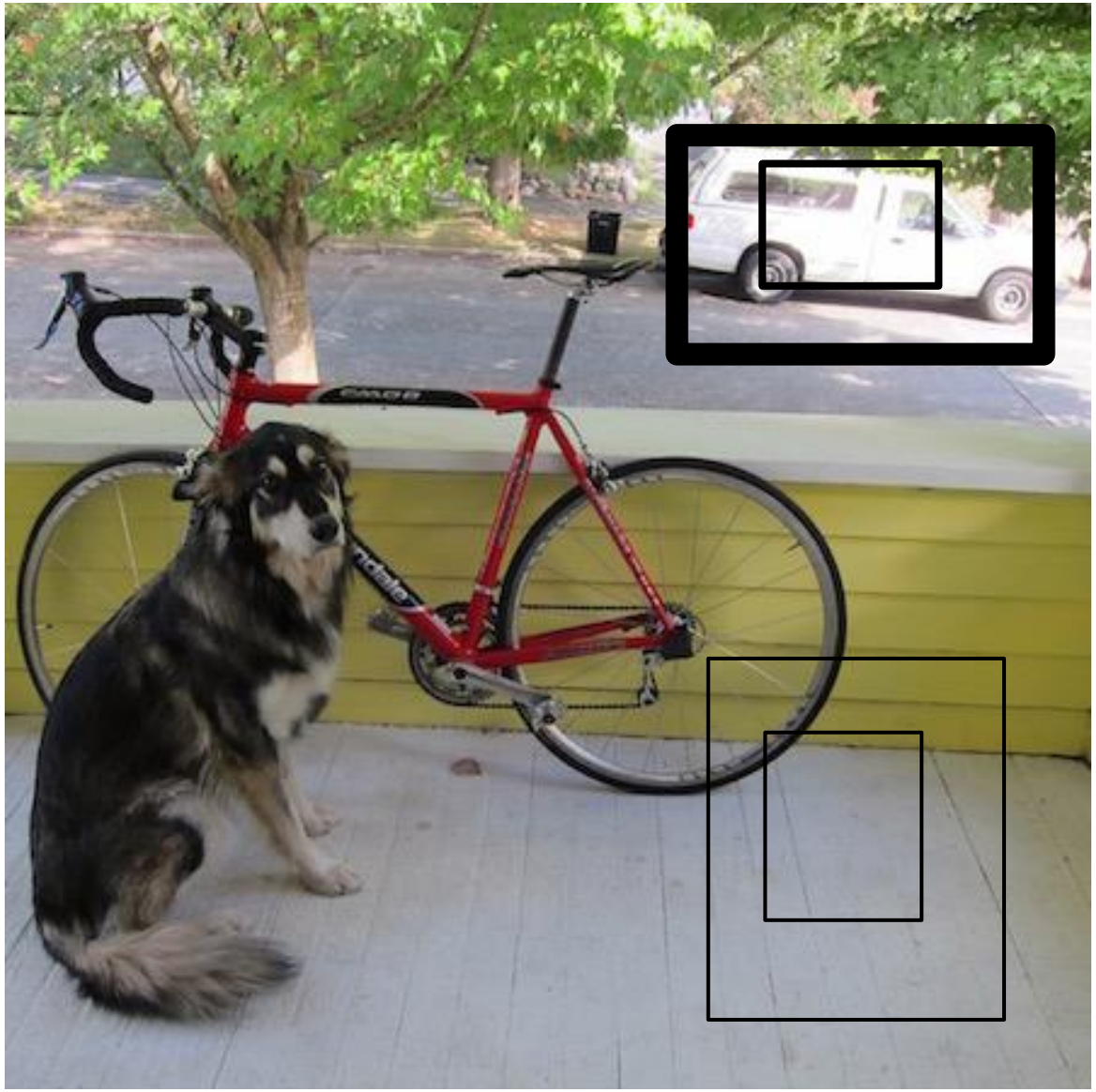


Each cell predicts boxes and confidences: P(Object)

Each cell predicts boxes and confidences: P(Object)

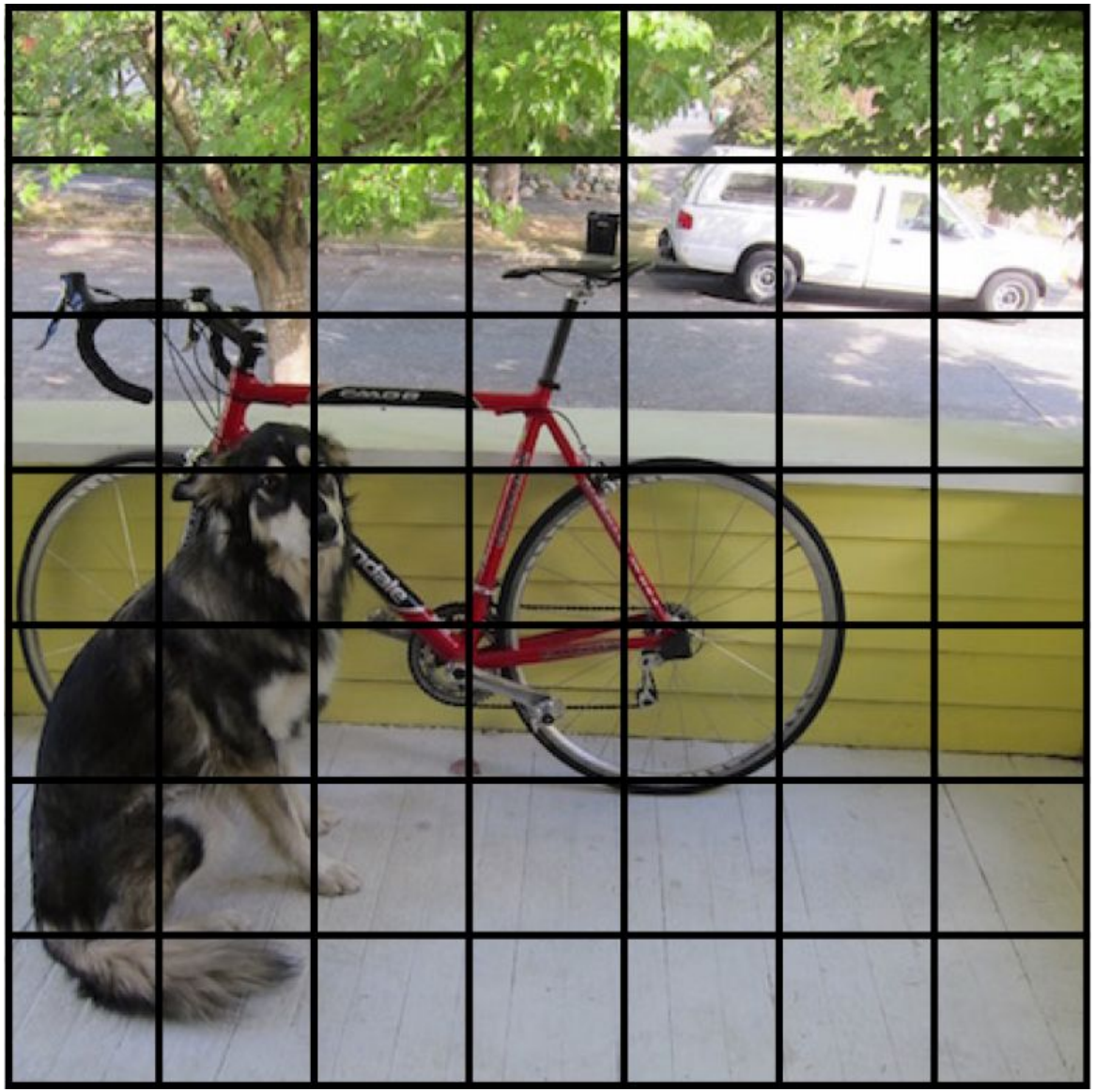


Each cell predicts boxes and confidences: P(Object)

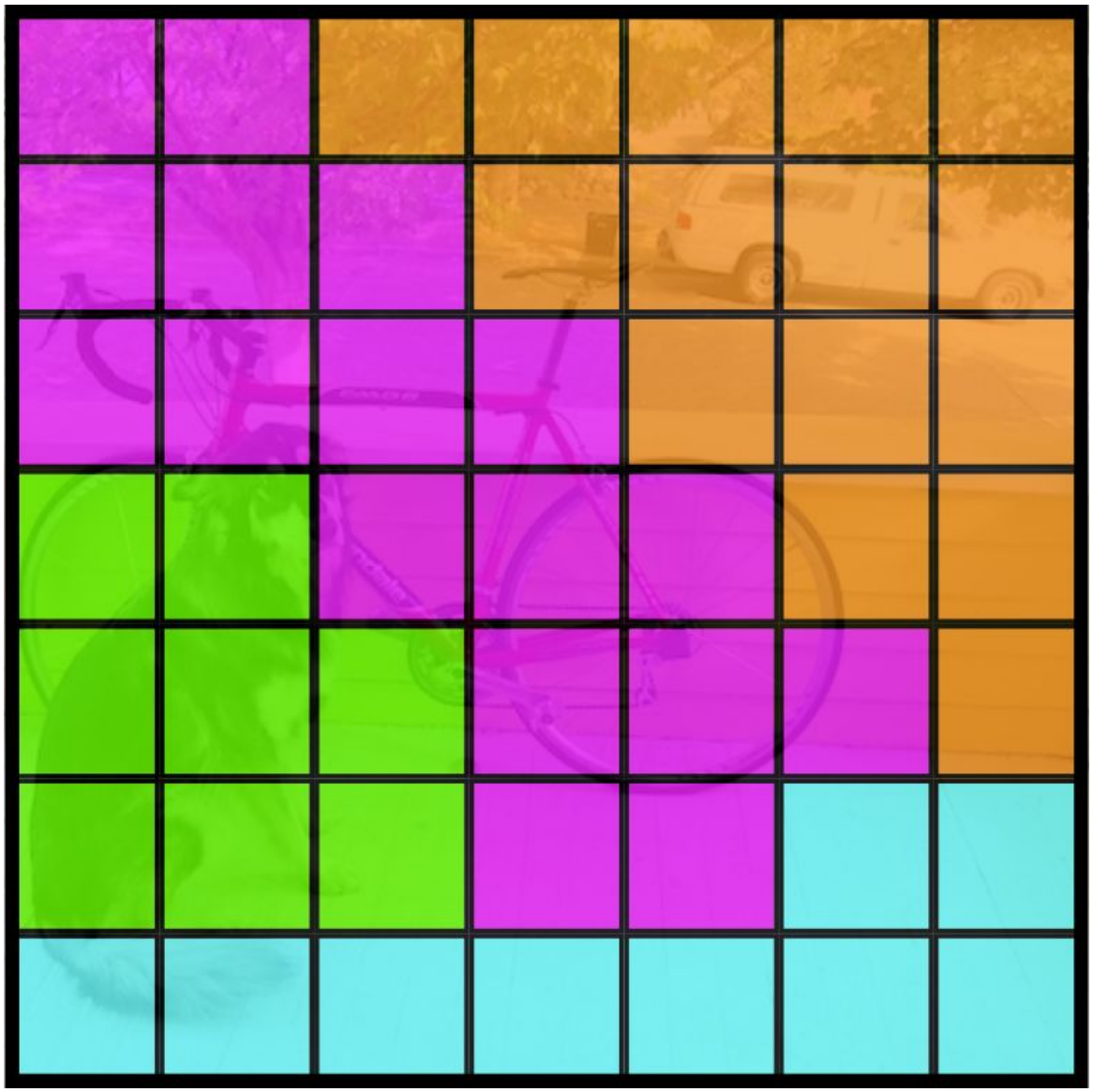


Each cell predicts boxes and confidences: P(Object)

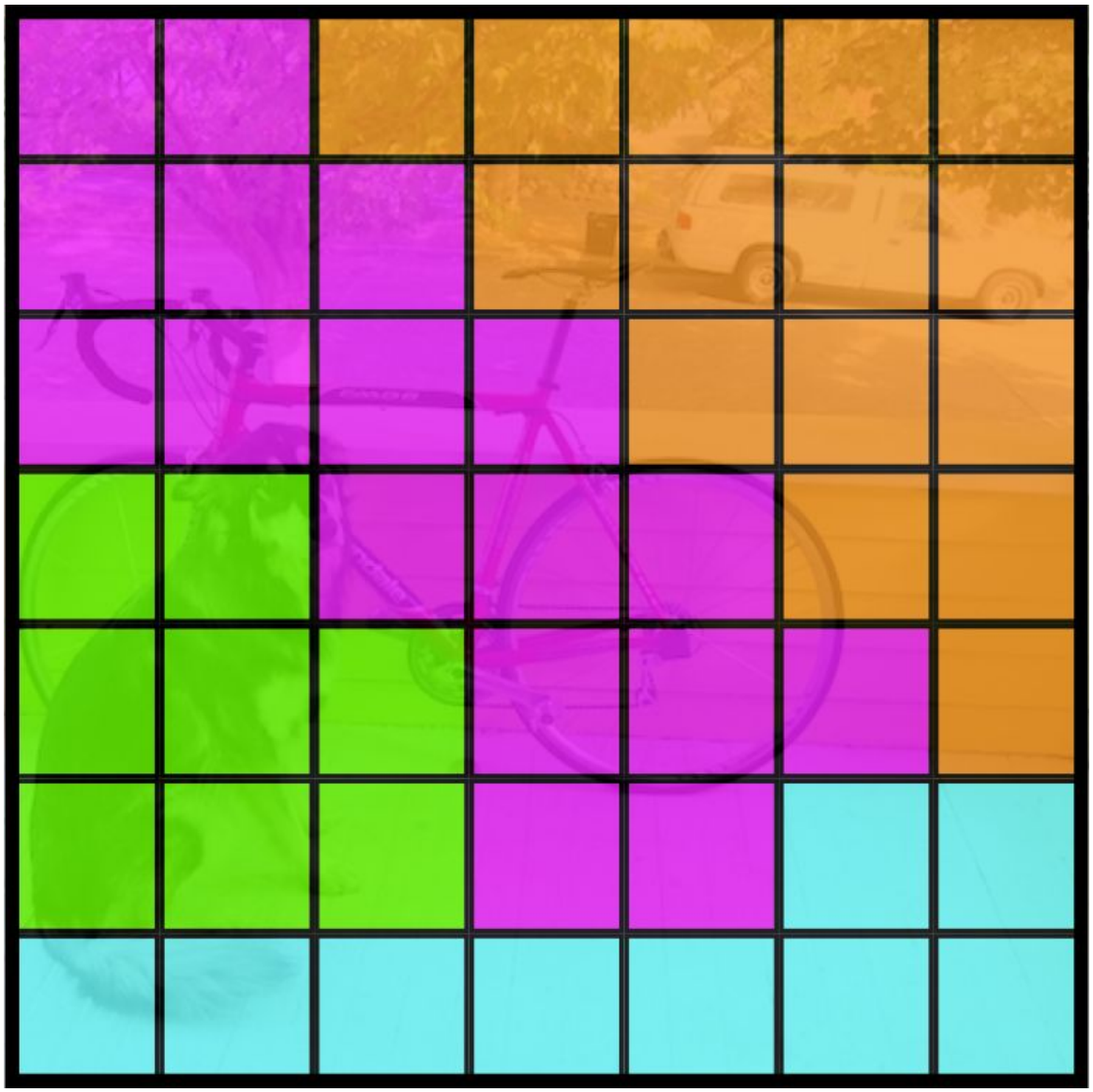
Each cell also predicts a class probability.



Each cell also predicts a class probability.



Conditioned on object: P(Car | Object)



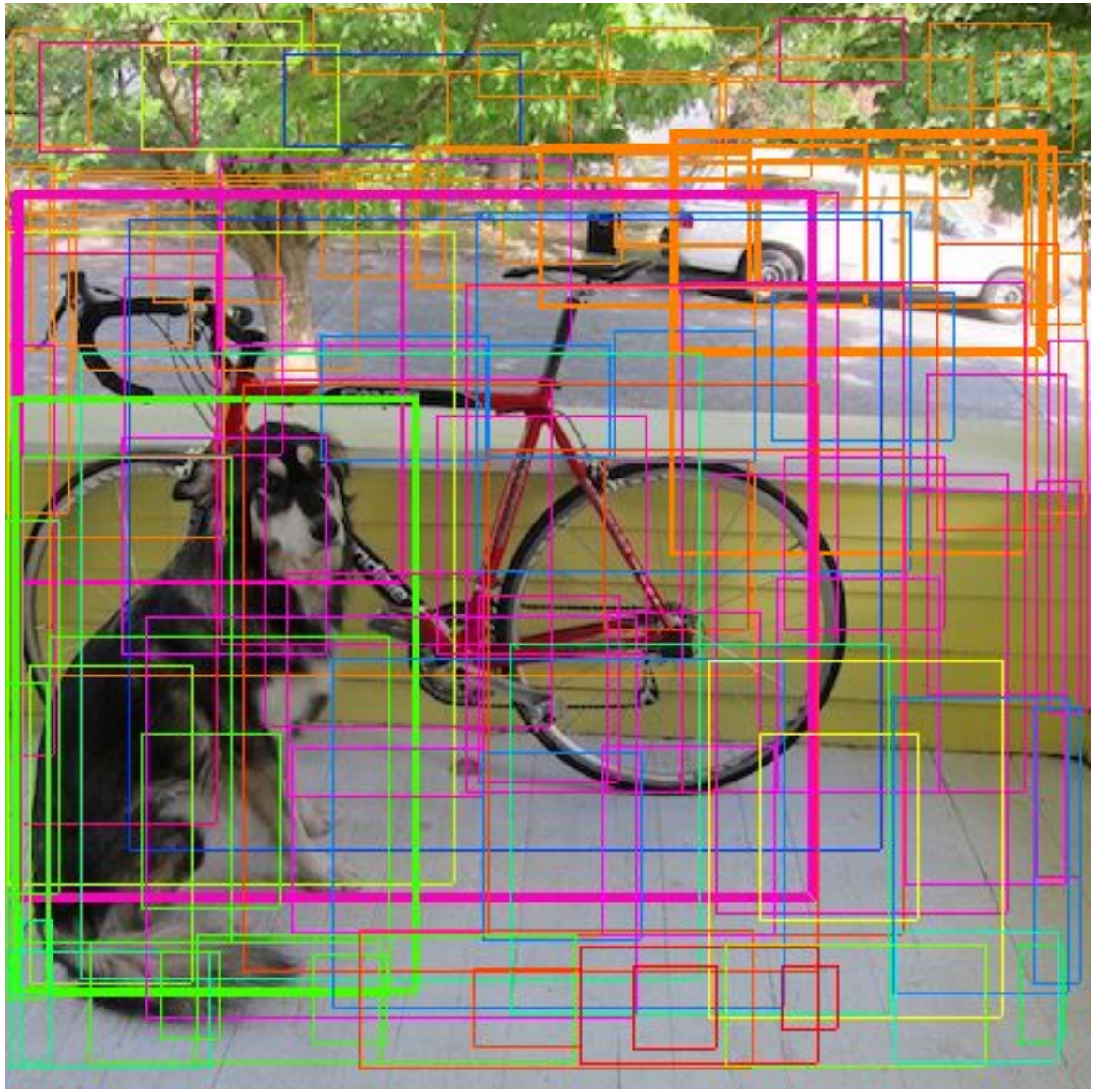
Bicycle

Car

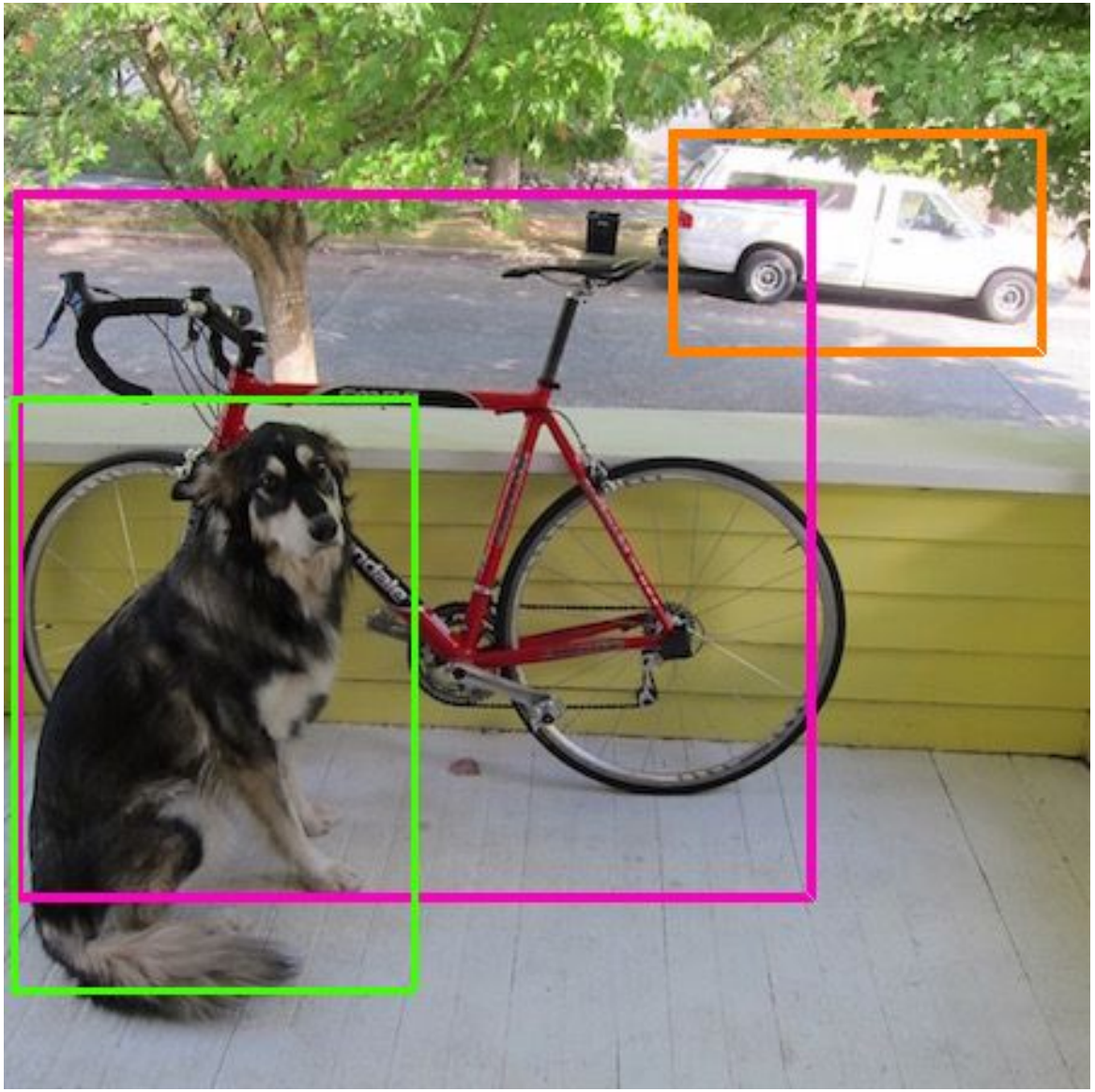
Dog

Dining Table

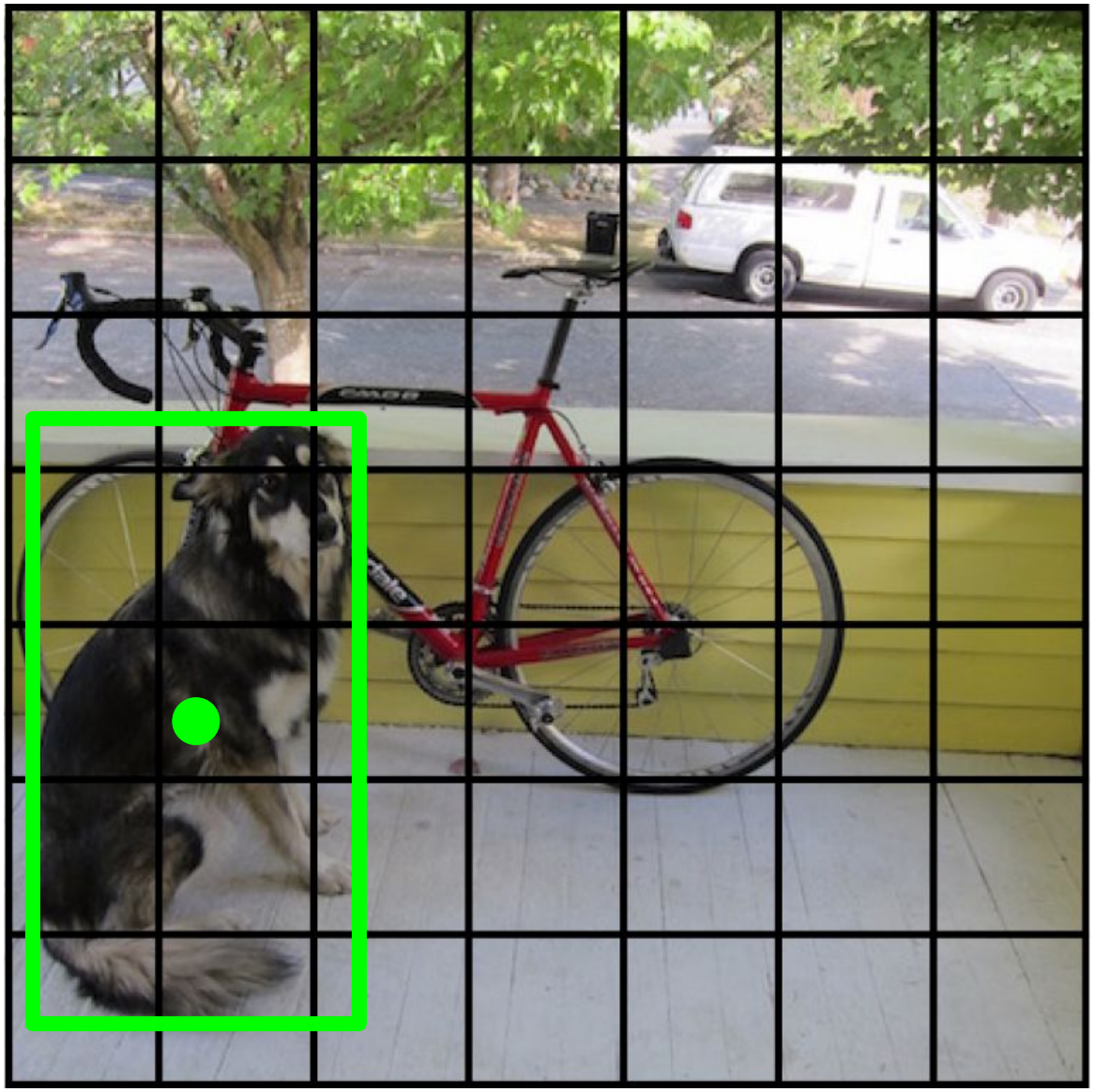
Then we combine the box and class predictions.



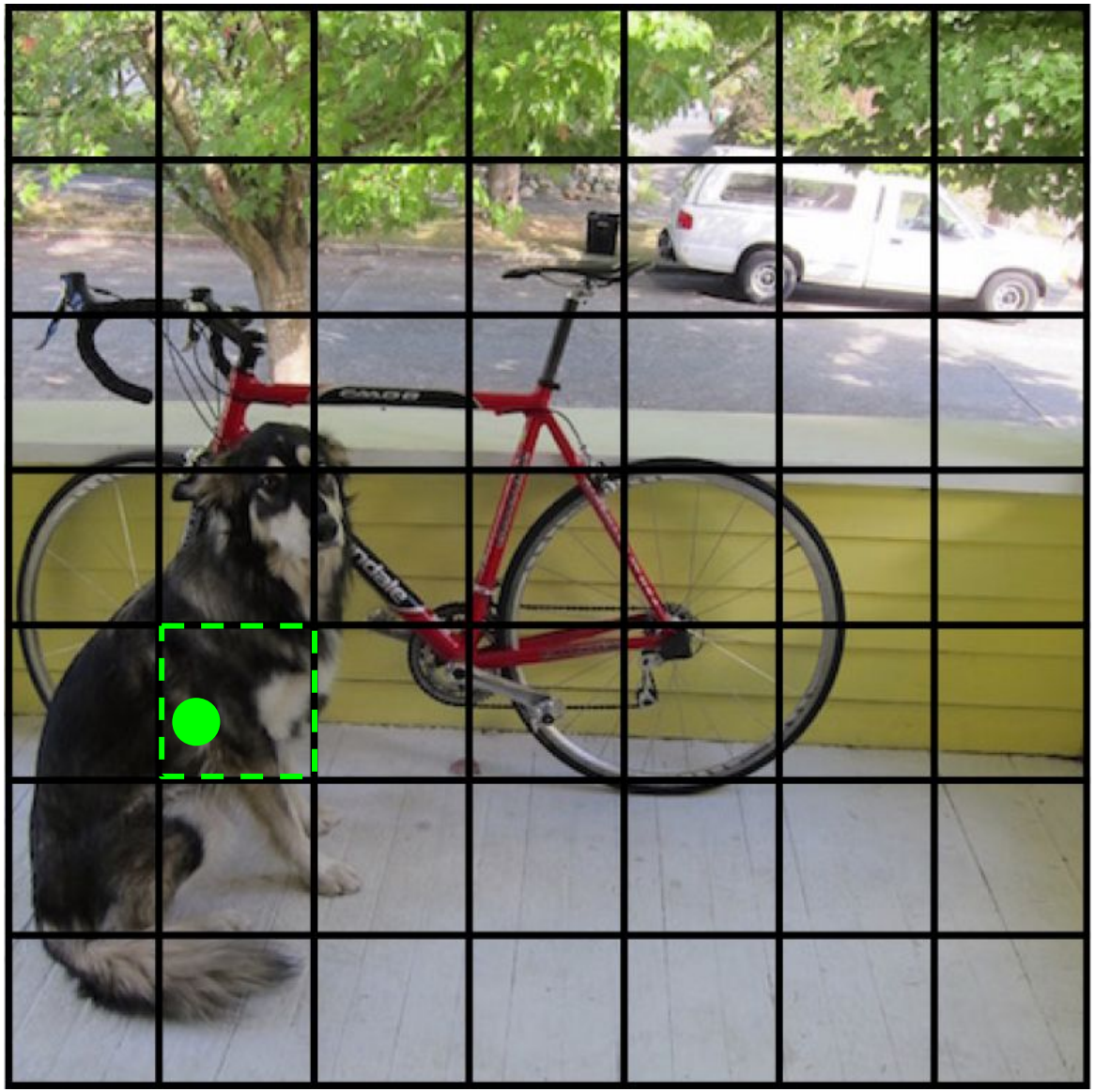
Finally we do NMS and threshold detections



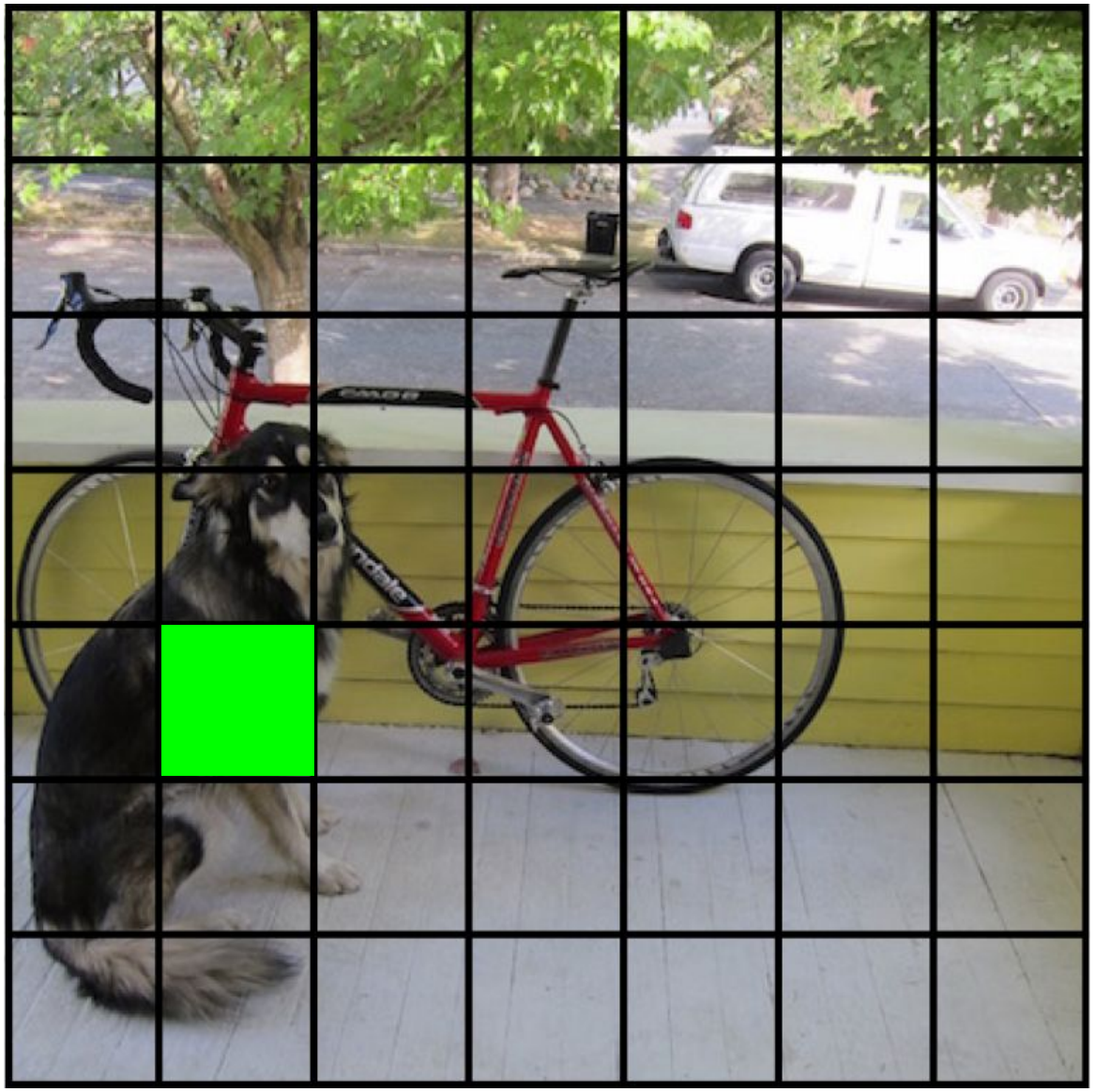
During training, match example to the right cell



During training, match example to the right cell

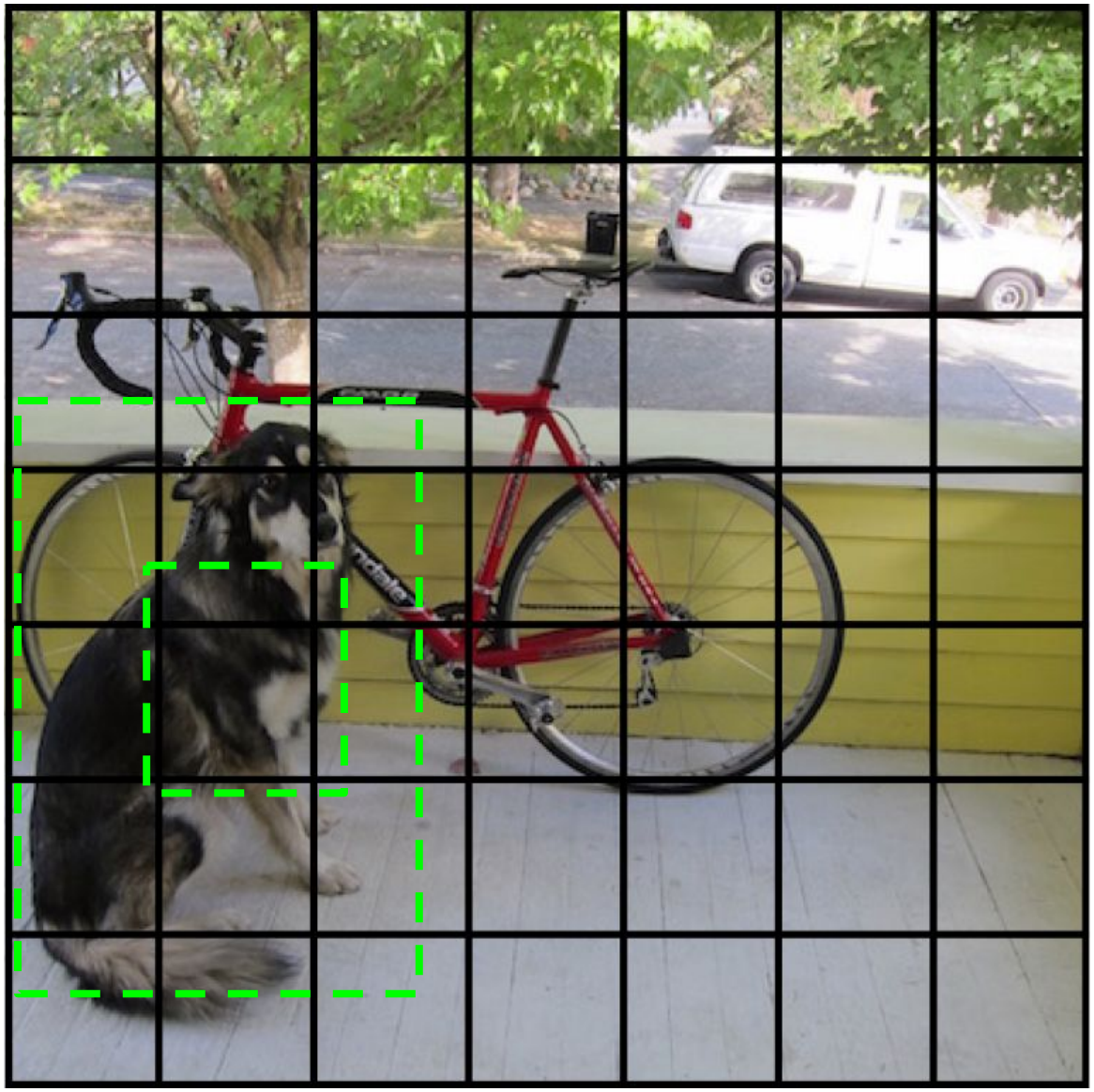


Adjust that cell’s class prediction

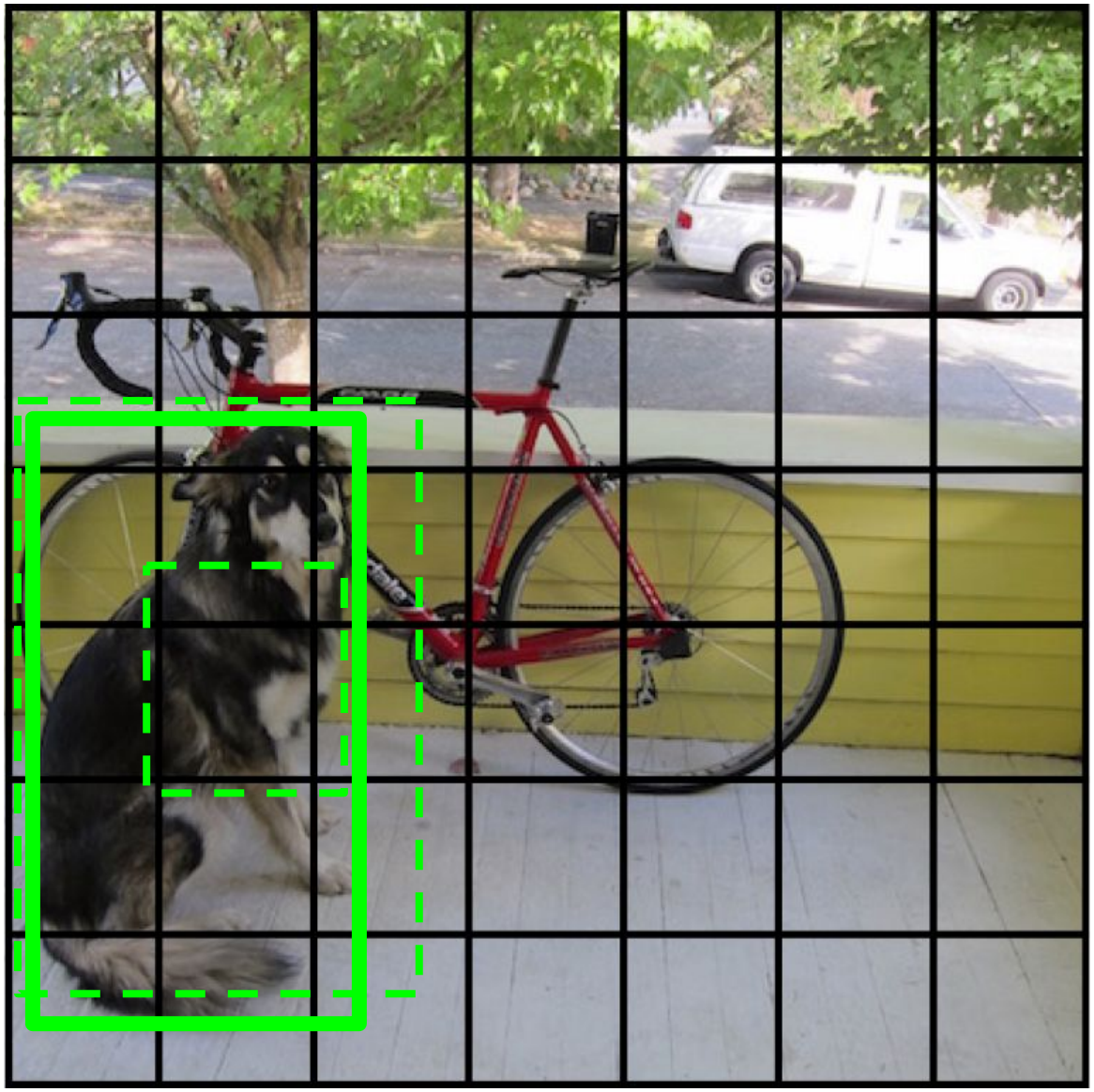


Dog = 1
Cat = 0
Bike = 0
…

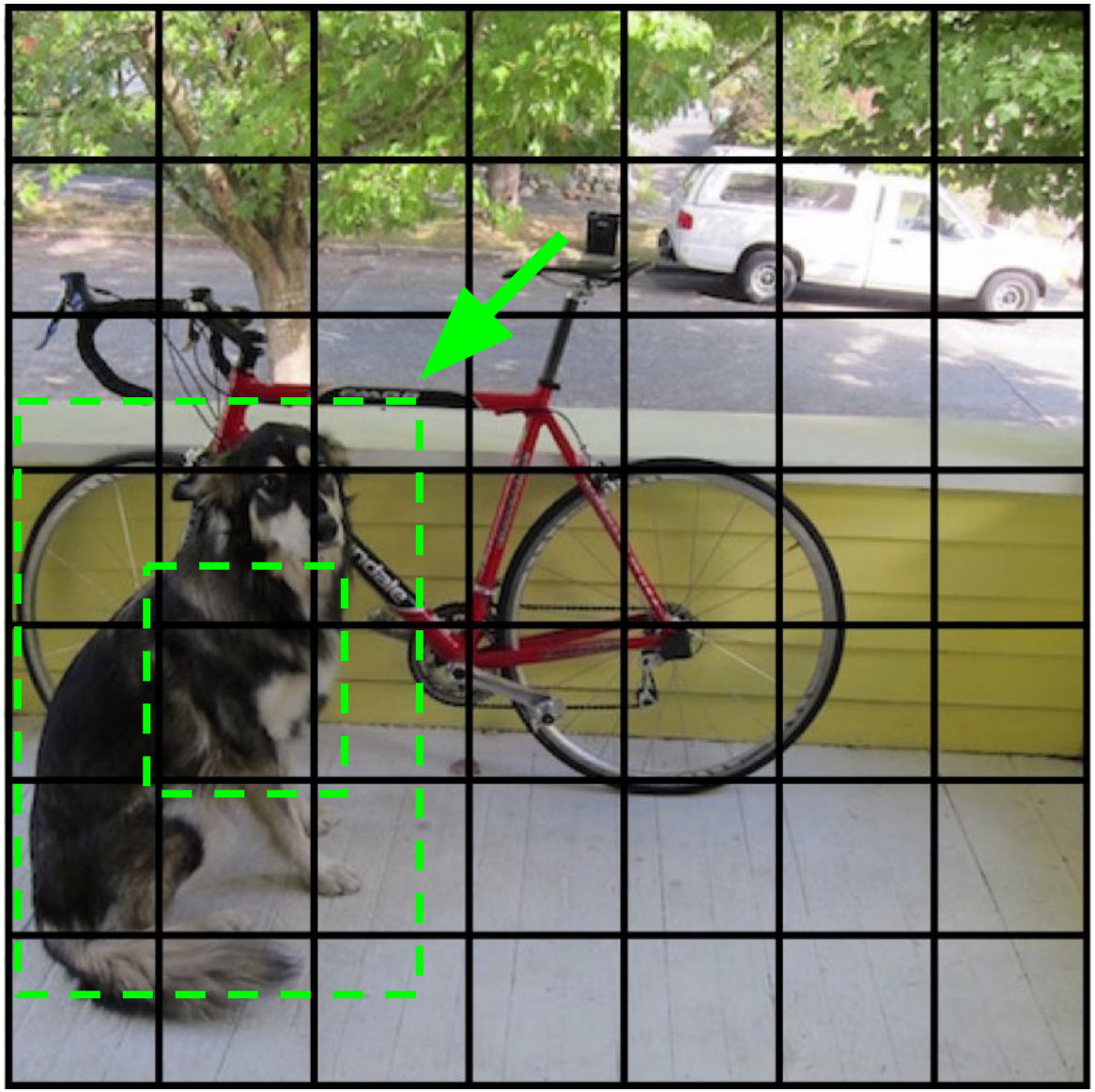
Look at that cell's predicted boxes



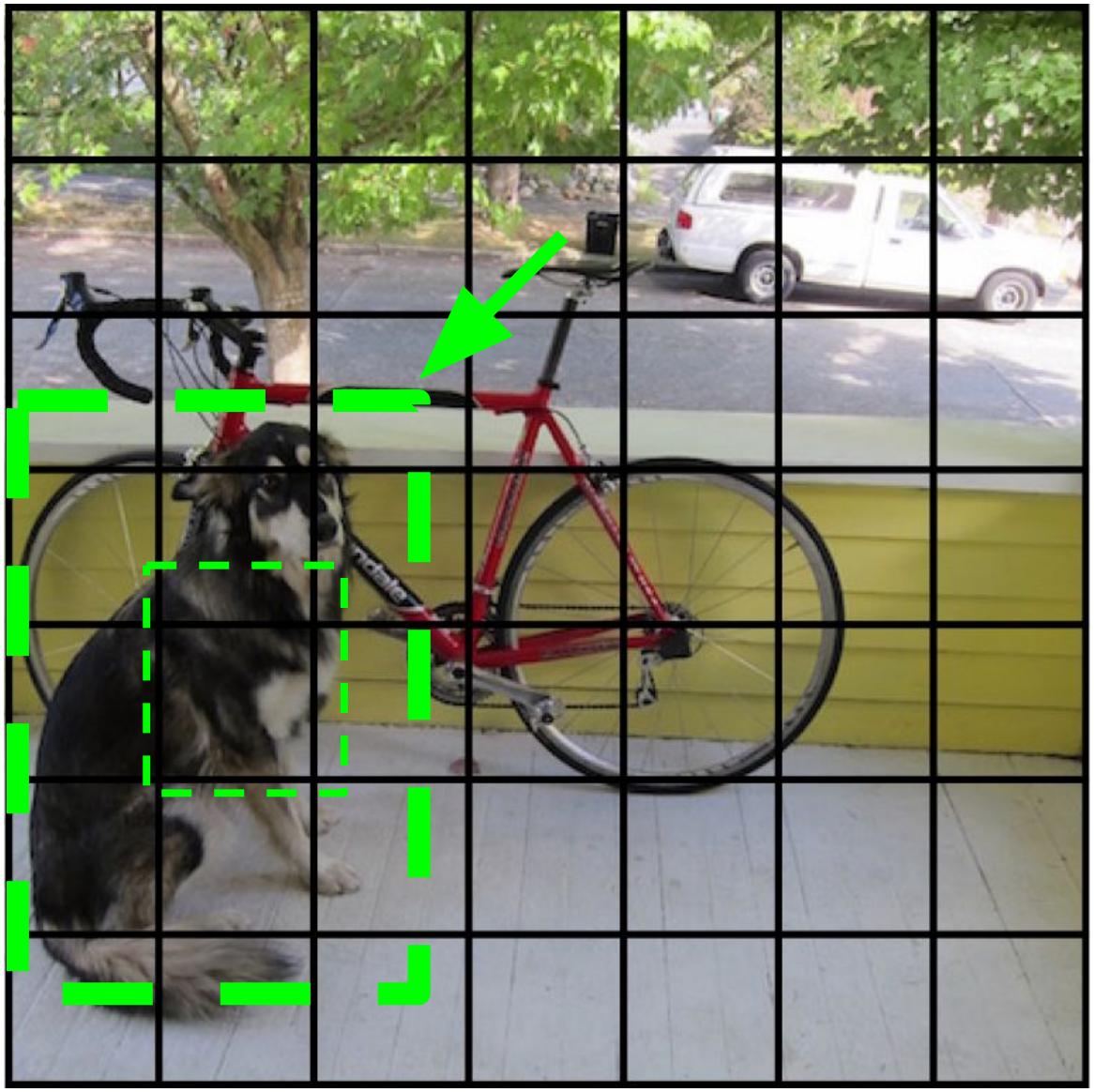
Find the best one, adjust it, increase the confidence



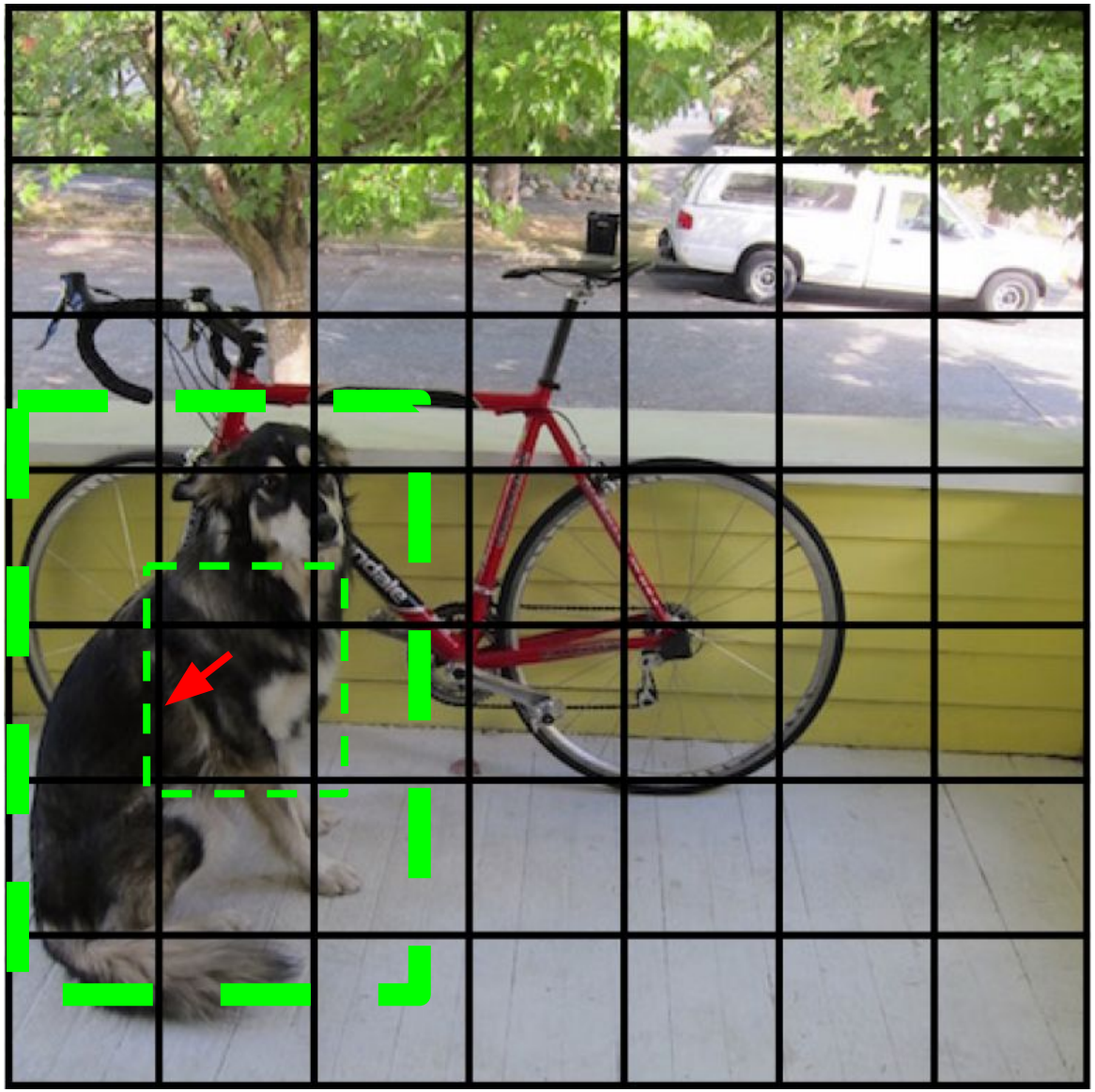
Find the best one, adjust it, increase the confidence



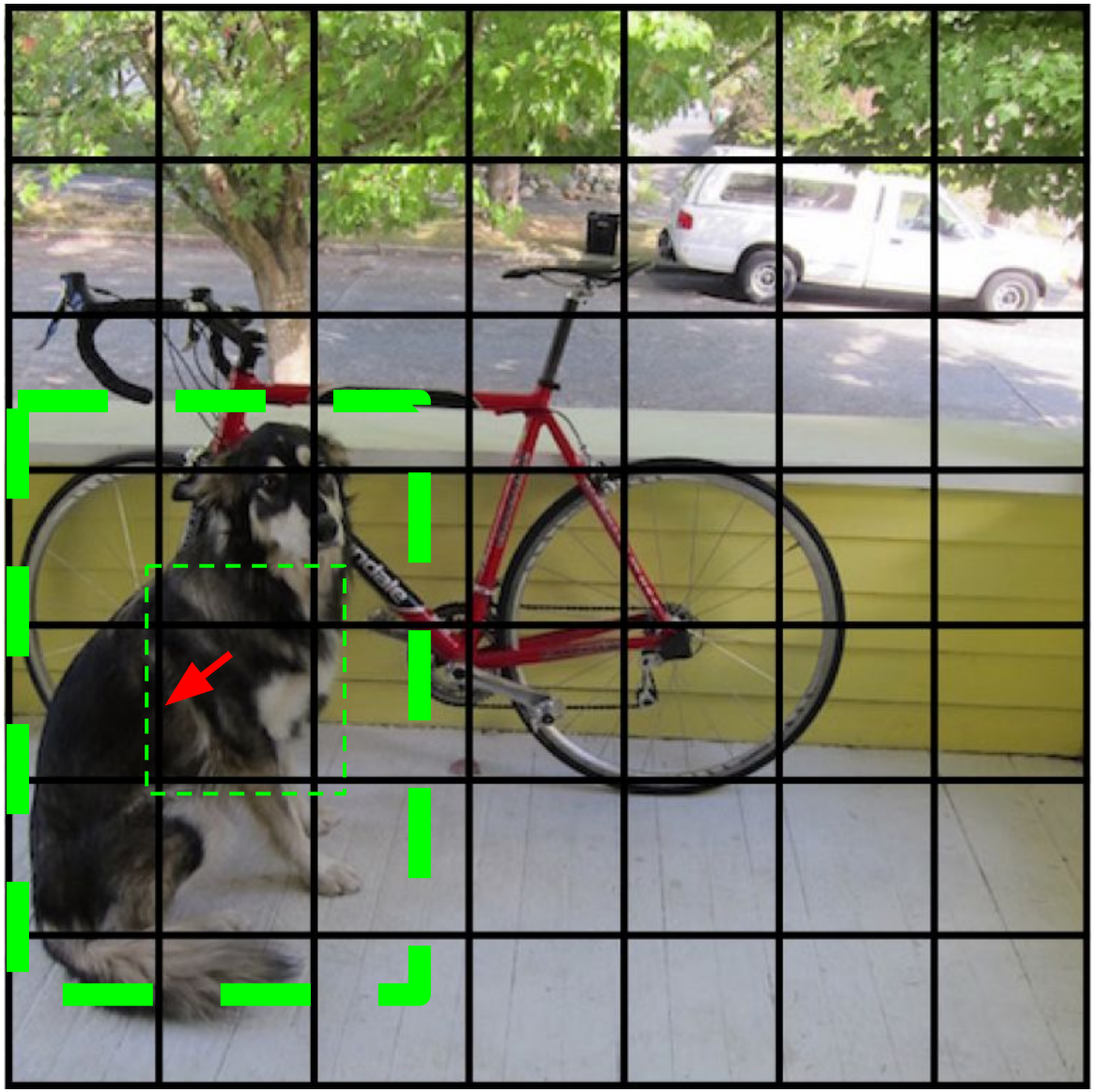
Find the best one, adjust it, increase the confidence



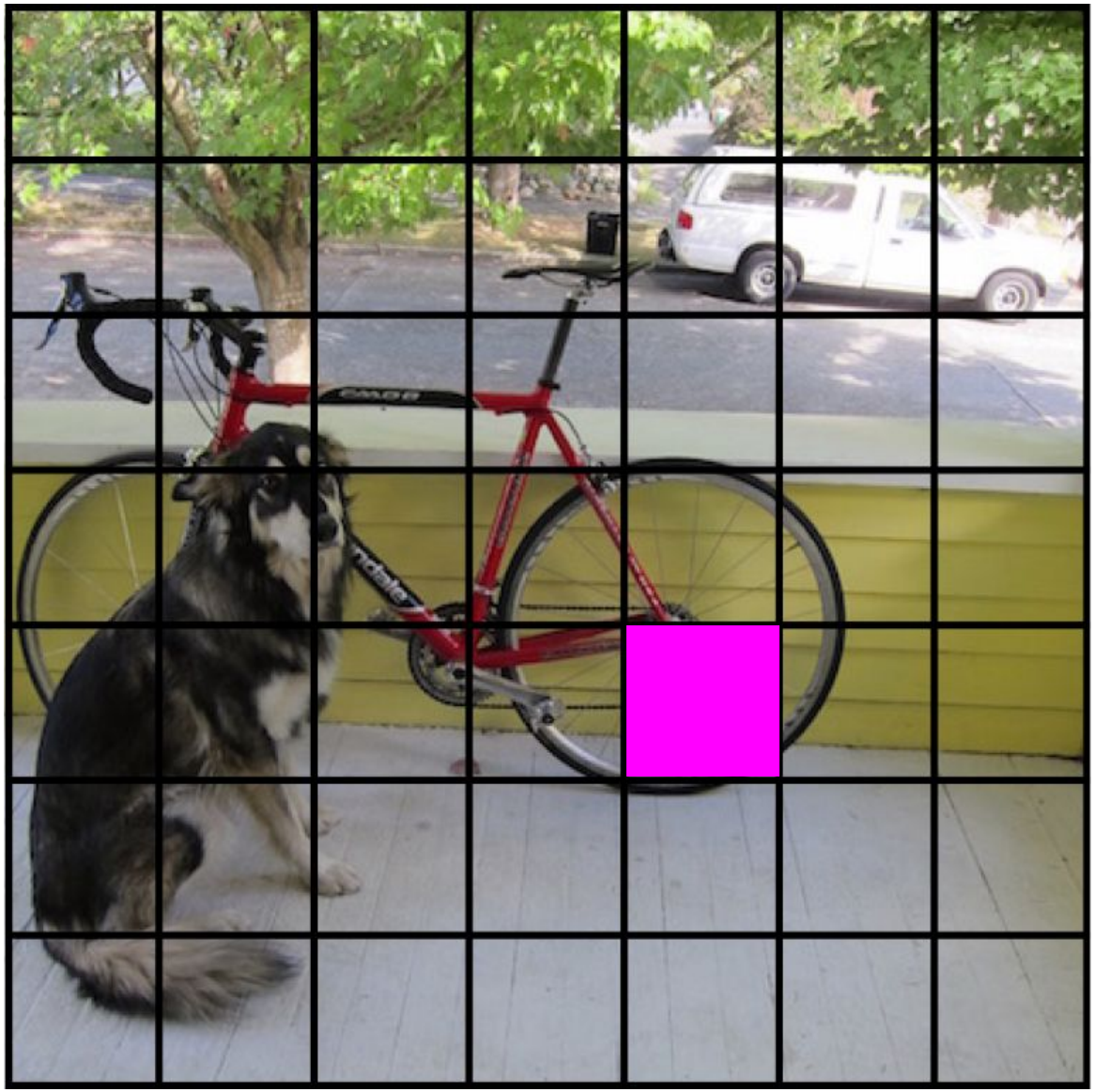
Decrease the confidence of other boxes



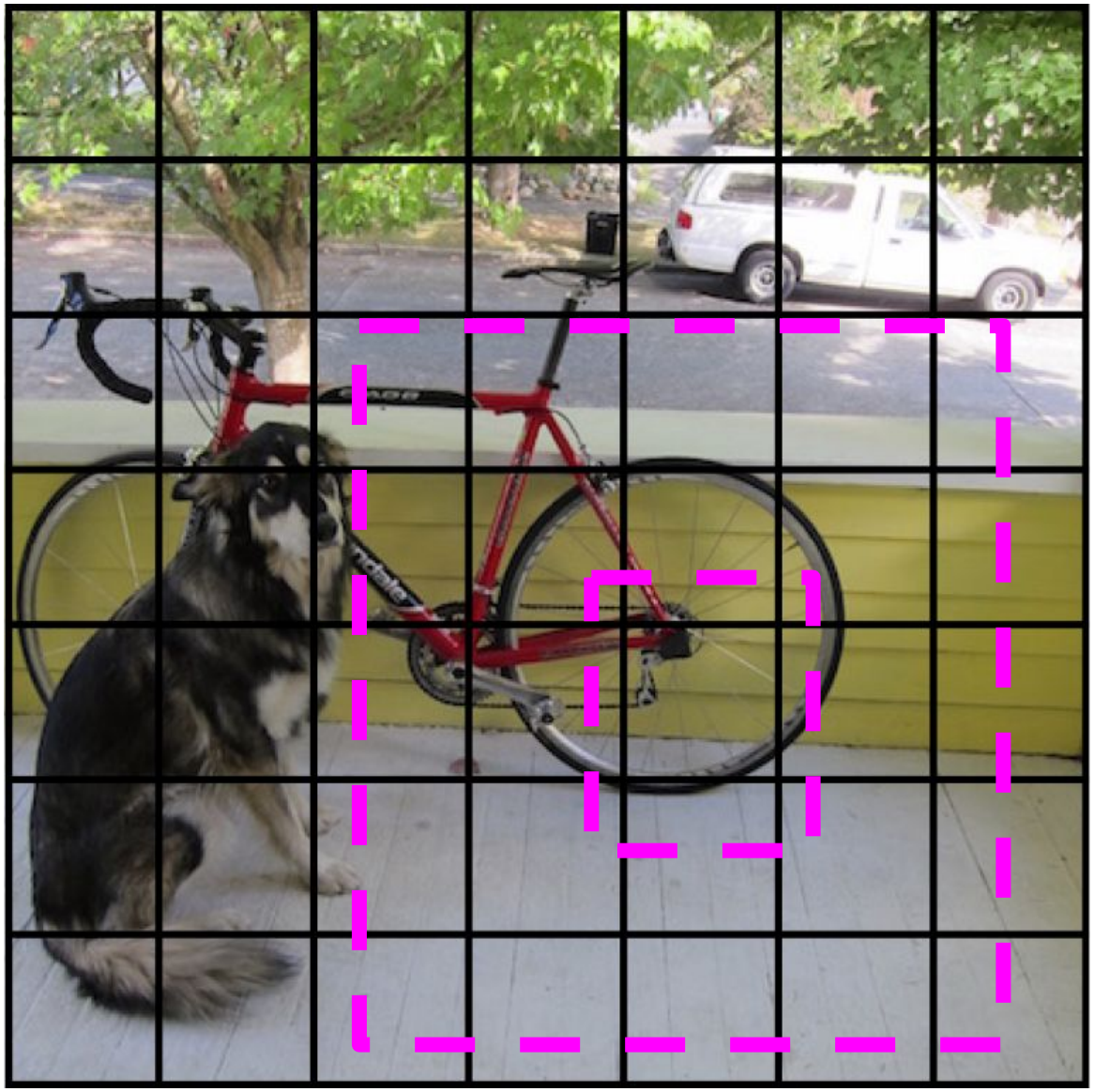
Decrease the confidence of other boxes



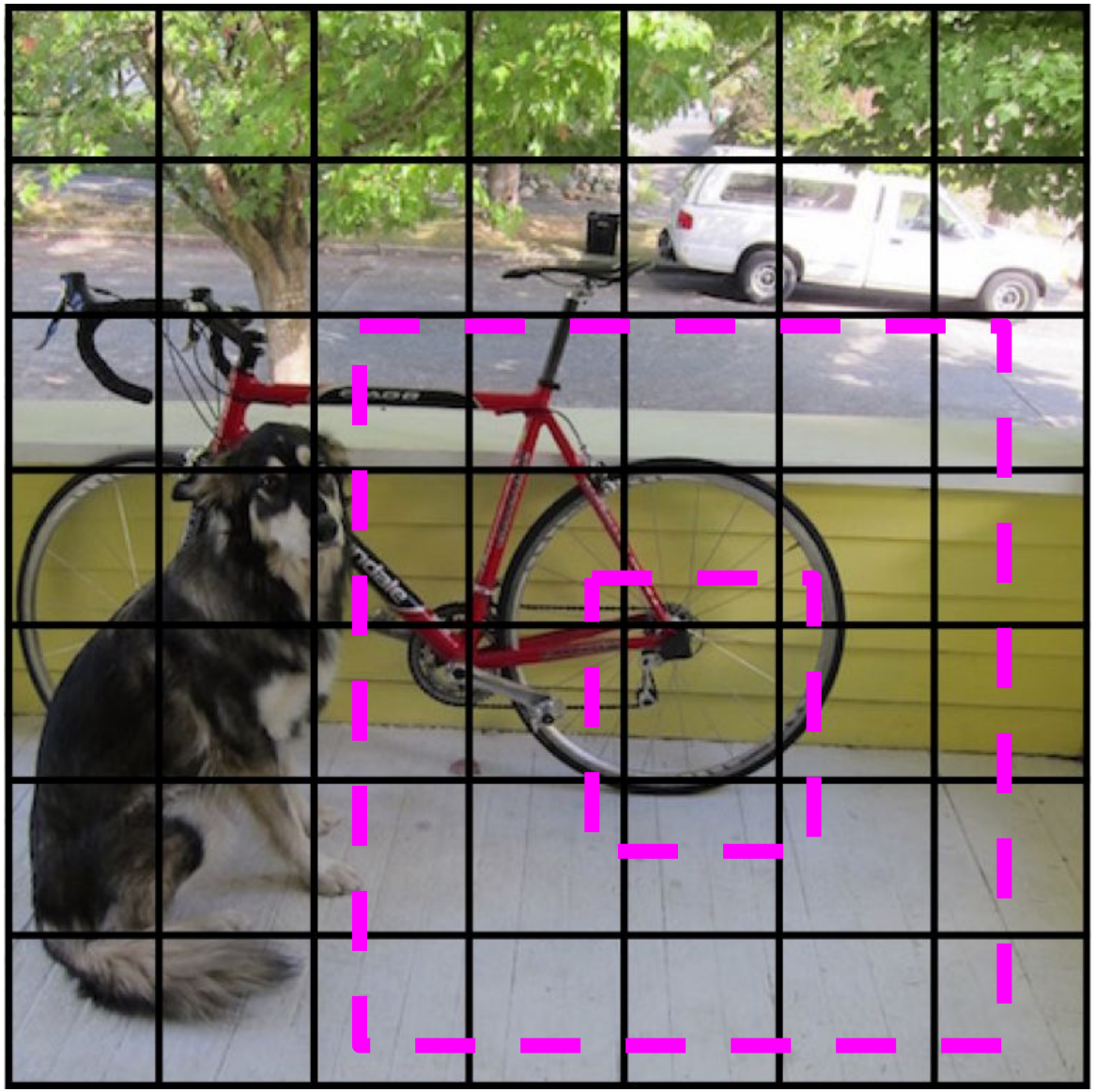
Some cells don’t have any ground truth detections!



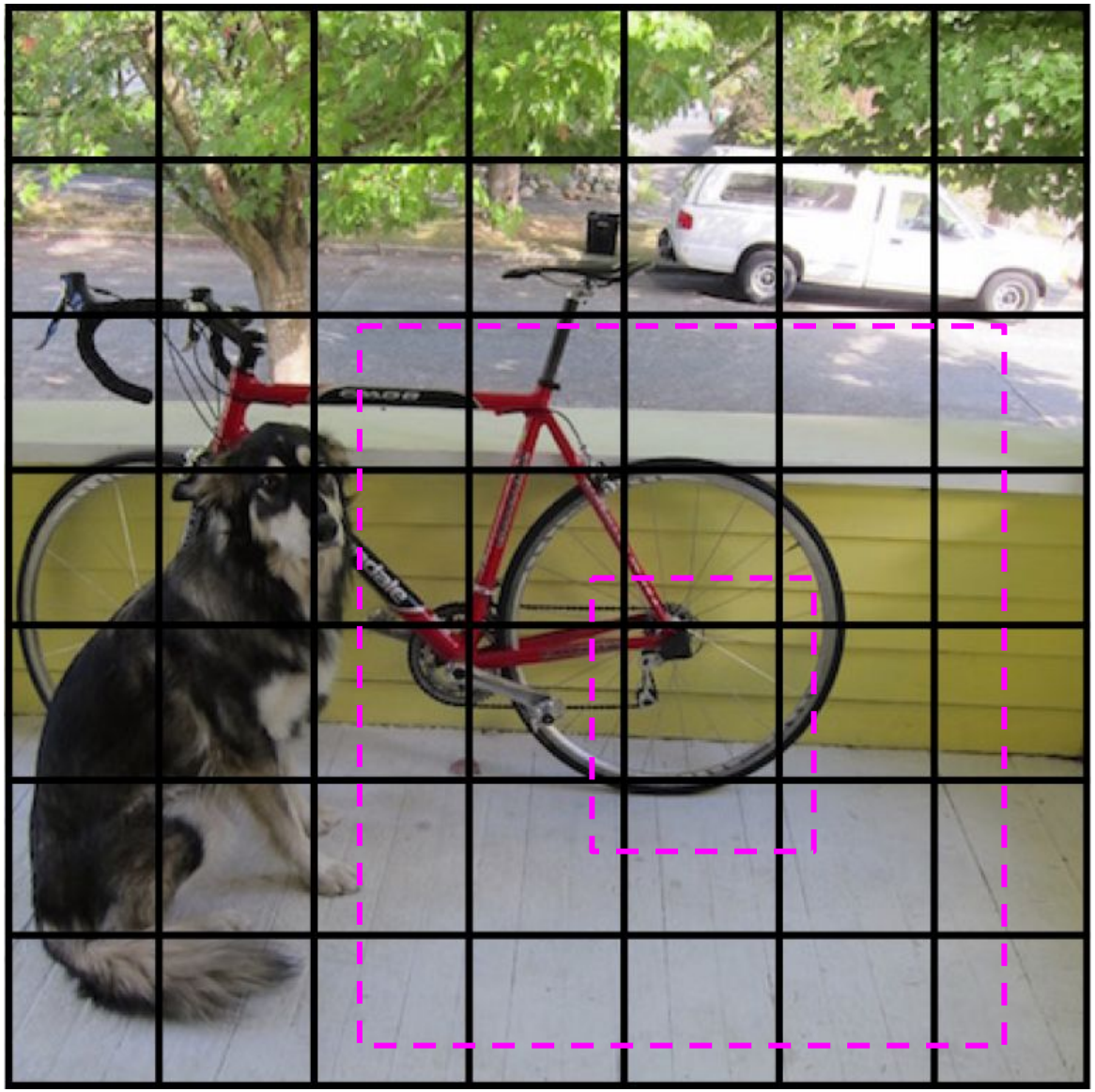
Some cells don’t have any ground truth detections!



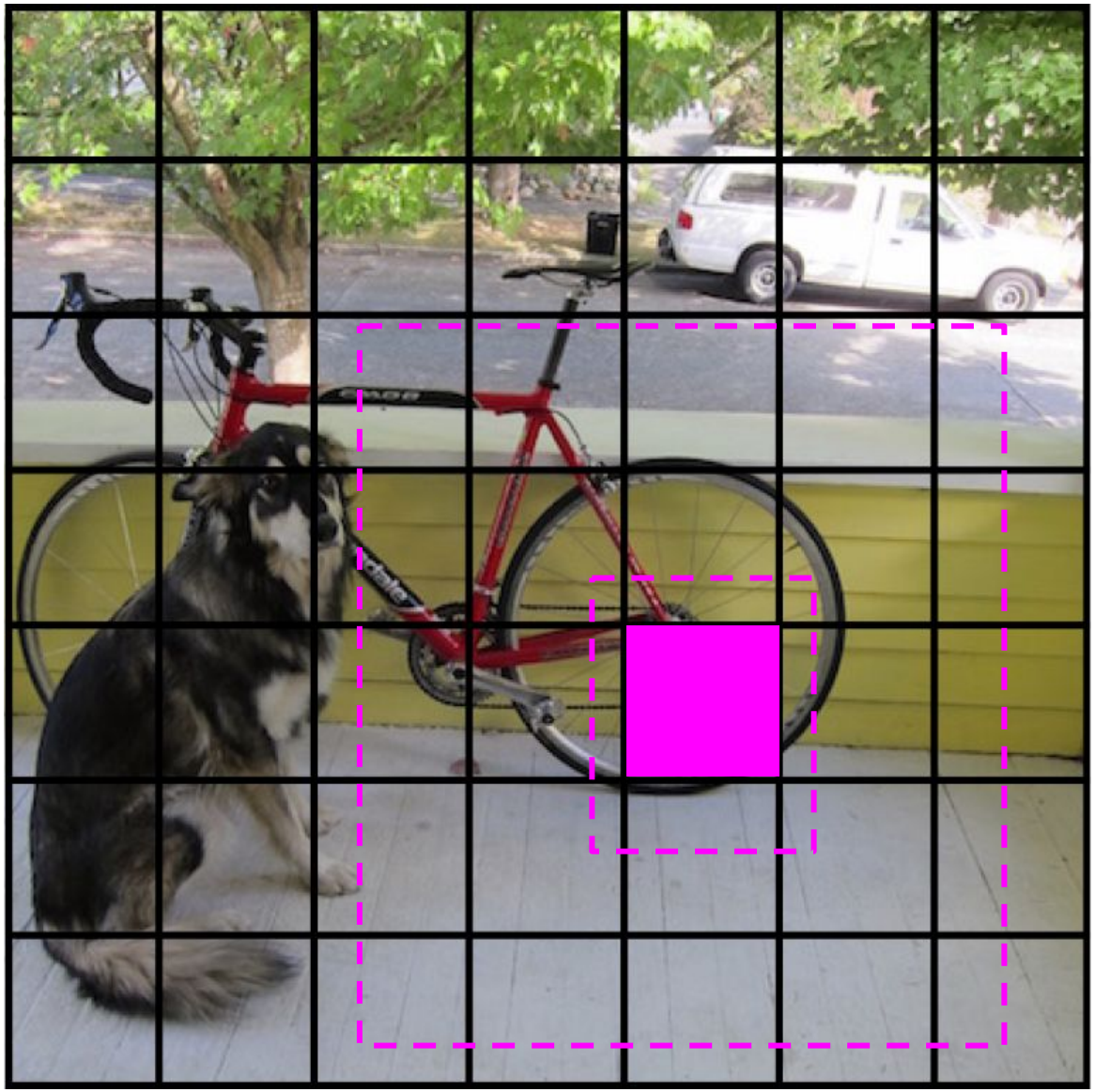
Decrease the confidence of these boxes



Decrease the confidence of these boxes



Don’t adjust the class probabilities or coordinates



Loss Function (sum-squared error)

loss function:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \quad (3)$$

model. We use sum-squared error because it is easy to optimize, however it does not perfectly align with our goal of maximizing average precision. It weights localization error equally with classification error which may not be ideal. Also, in every image many grid cells do not contain any object. This pushes the "confidence" scores of those cells towards zero, often overpowering the gradient from cells that do contain objects. This can lead to model instability, causing training to diverge early on.

To remedy this, we increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that don't contain objects. We use two parameters, λ_{coord} and λ_{noobj} to accomplish this. We set $\lambda_{\text{coord}} = 5$ and $\lambda_{\text{noobj}} = .5$.

$$\lambda_{\text{coord}}=5, \quad \lambda_{\text{noobj}}=0.5$$

Loss Function (sum-squared error)

loss function:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

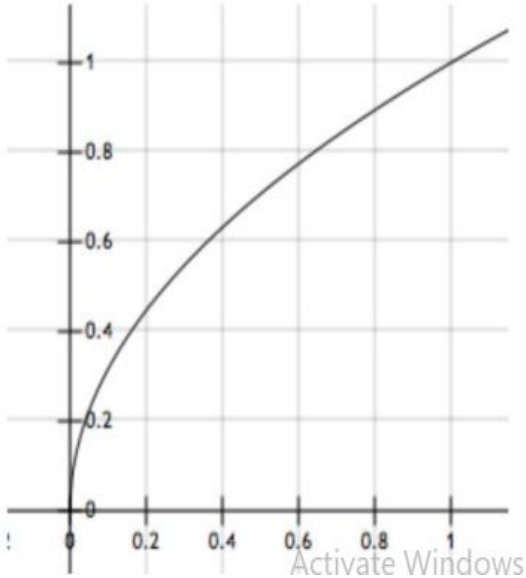
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

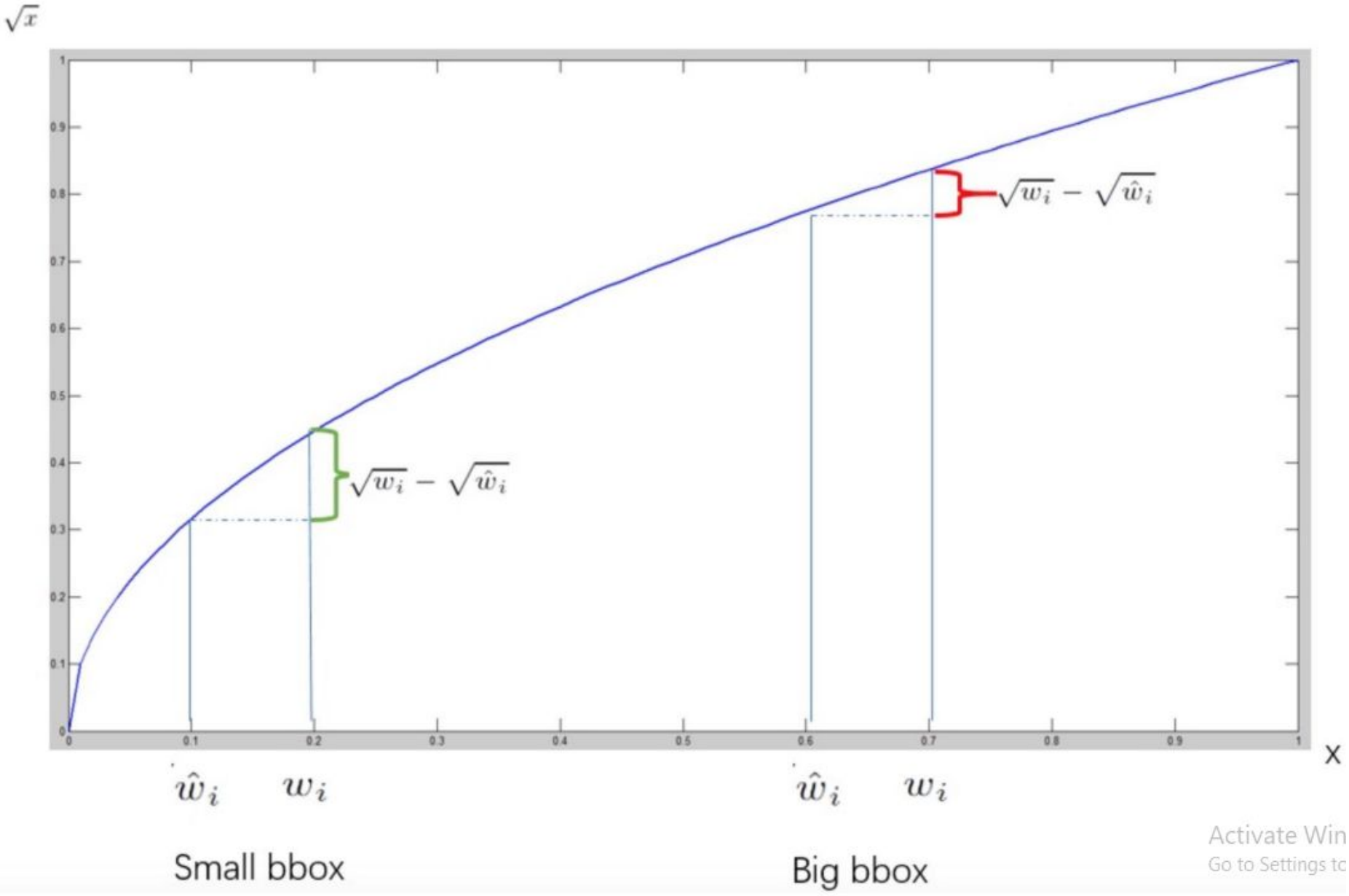
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \quad (3)$$

Sum-squared error also equally weights errors in large boxes and small boxes. Our error metric should reflect that small deviations in large boxes matter less than in small boxes. To partially address this we predict the square root of the bounding box width and height instead of the width and height directly.





Loss Function (sum-squared error)

loss function:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \quad (3)$$

$\mathbb{1}_{ij}^{obj}$ — **The *j*th bbox predictor** in ***cell i*** is "responsible" for that prediction

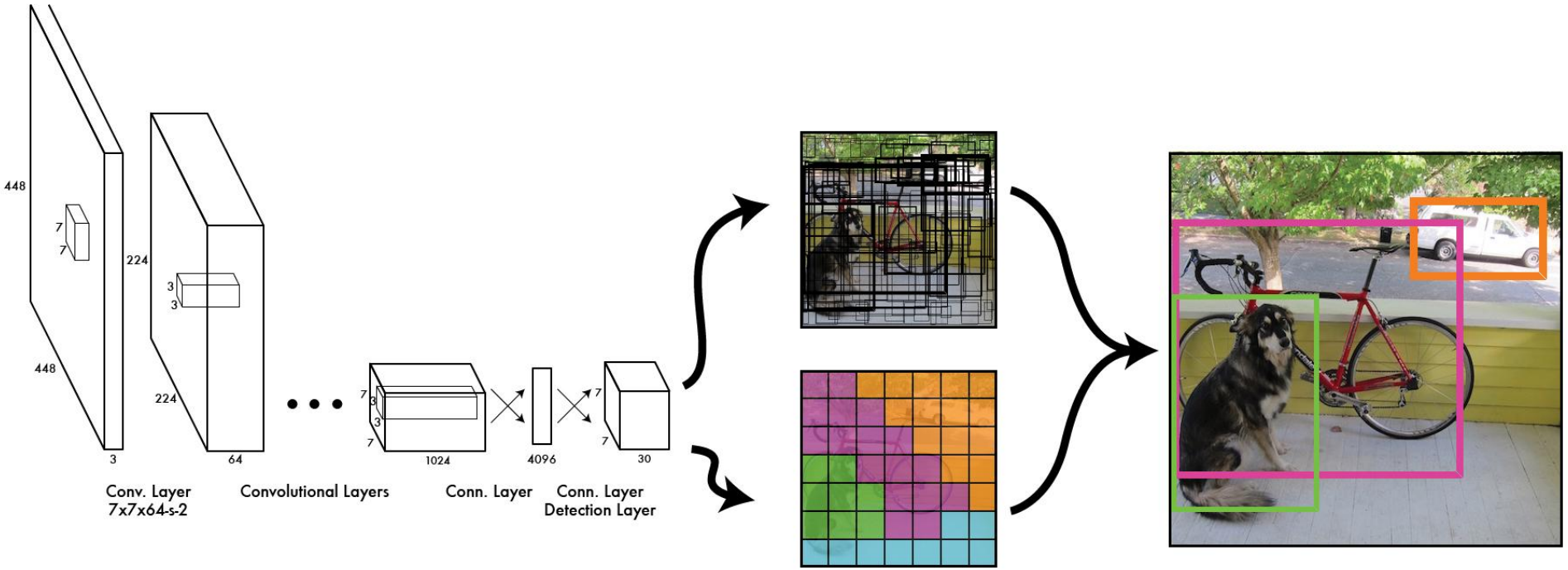
$\mathbb{1}_{ij}^{noobj}$

$\mathbb{1}_{i}^{obj}$ — If object appears in ***cell i***

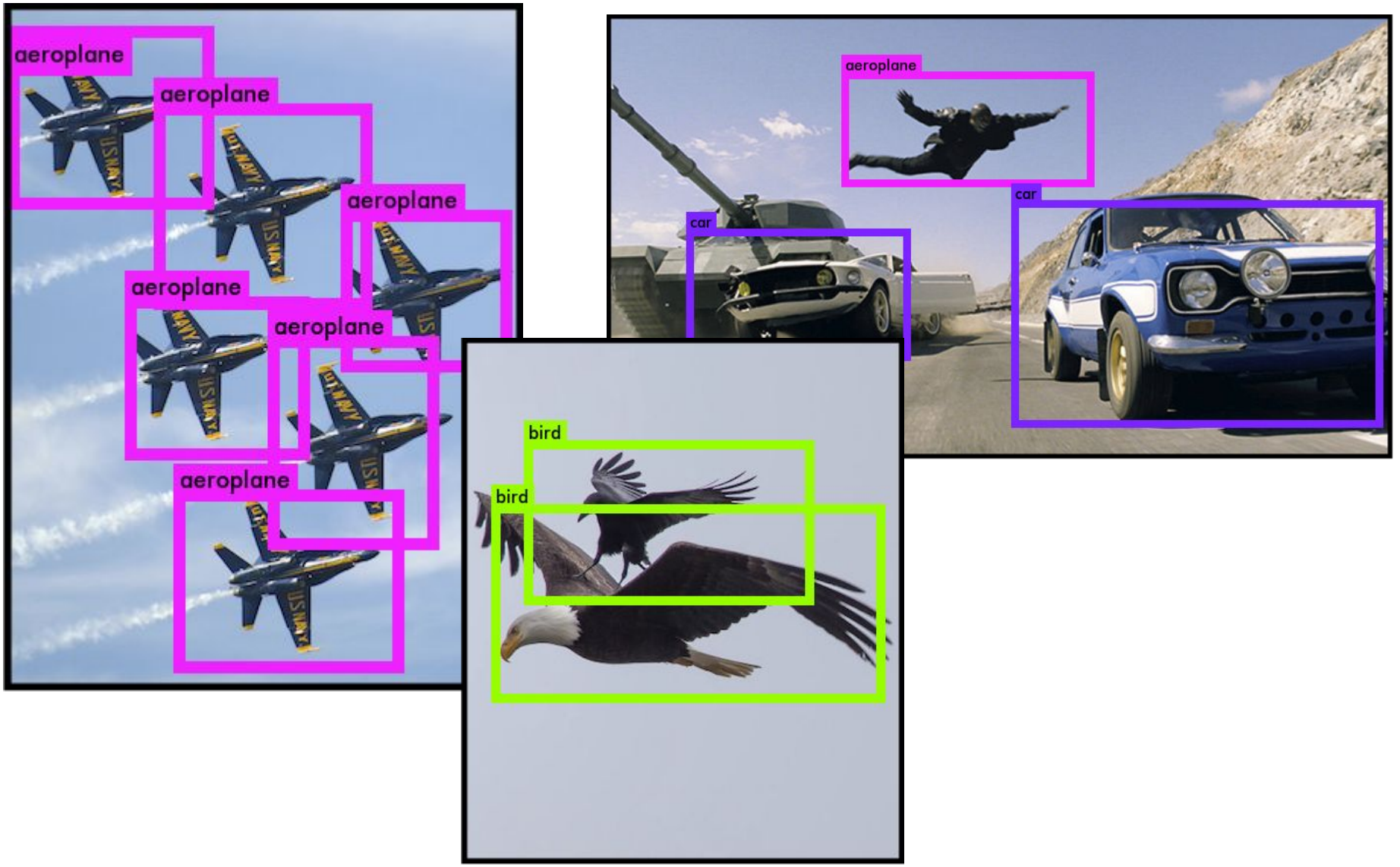
Note that the loss function only penalizes classification error if an object is present in that grid cell (hence the conditional class probability discussed earlier). It also only penalizes bounding box coordinate error if that predictor is "responsible" for the ground truth box (i.e. has the highest IOU of any predictor in that grid cell).

We train with standard tricks:

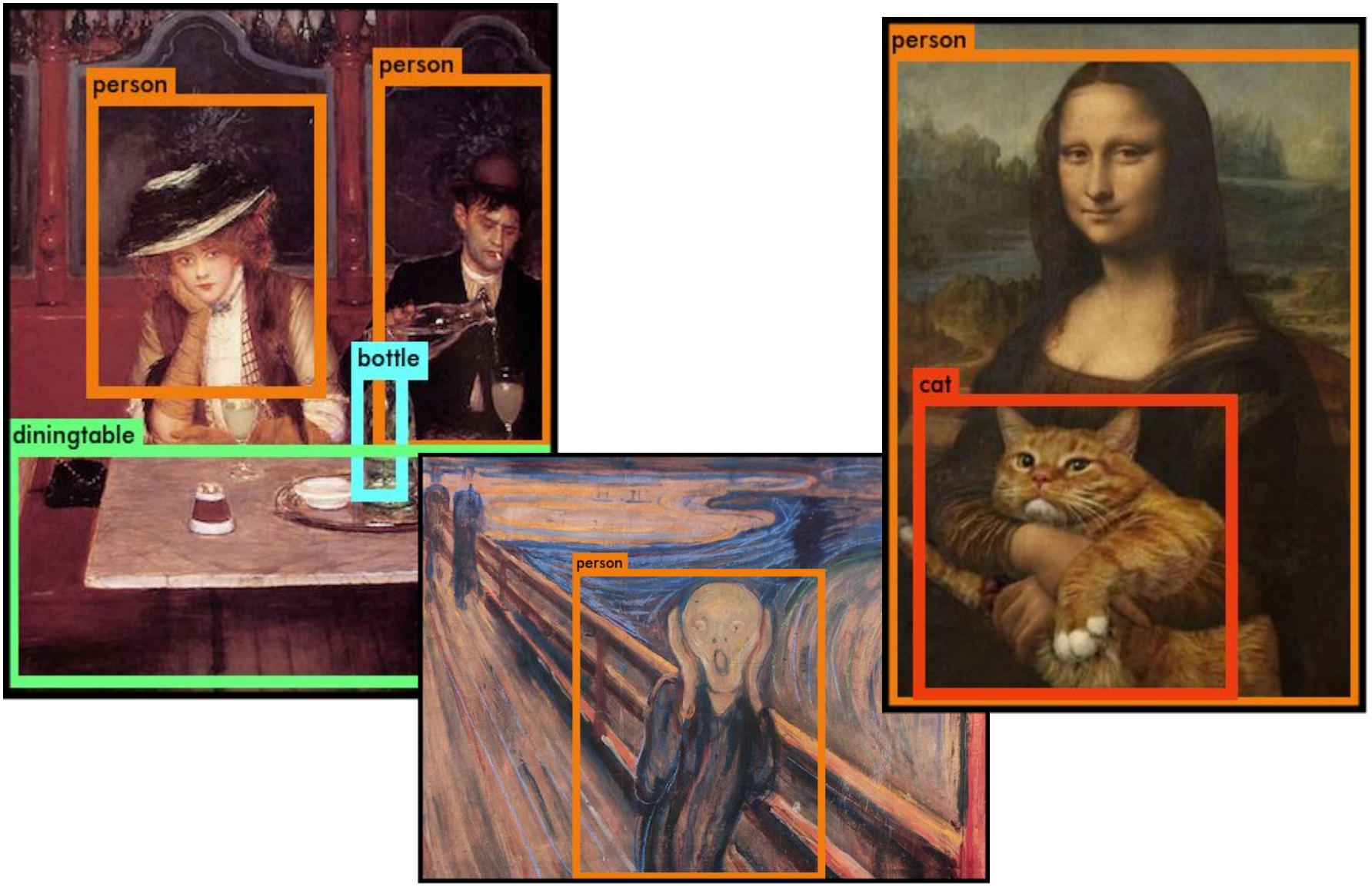
- Pretraining on Imagenet
- SGD with decreasing learning rate
- Extensive data augmentation
- For details, see the paper



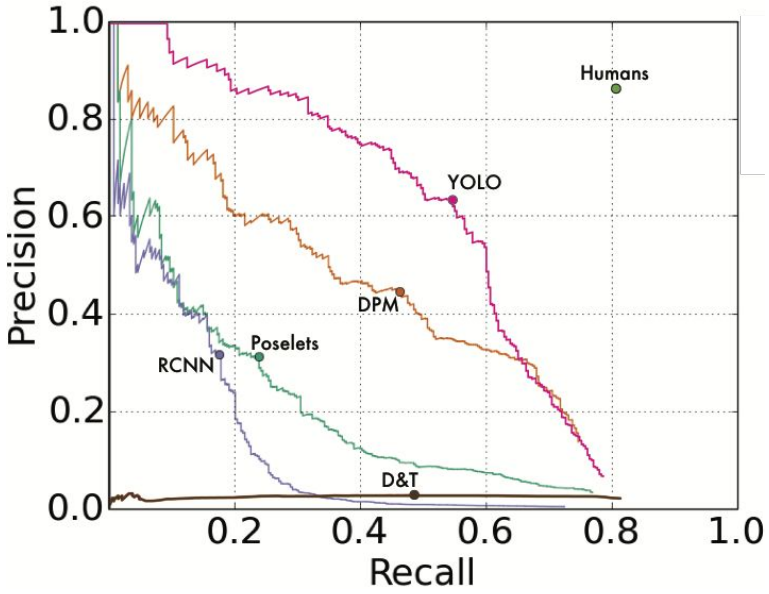
YOLO works across a variety of natural images



It also generalizes well to new domains (like art)



YOLO outperforms methods like DPM and R-CNN when generalizing to person detection in artwork



	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32

S. Ginosar, D. Haas, T. Brown, and J. Malik. Detecting people in cubist art. In Computer Vision-ECCV 2014 Workshops, pages 101–116. Springer, 2014.

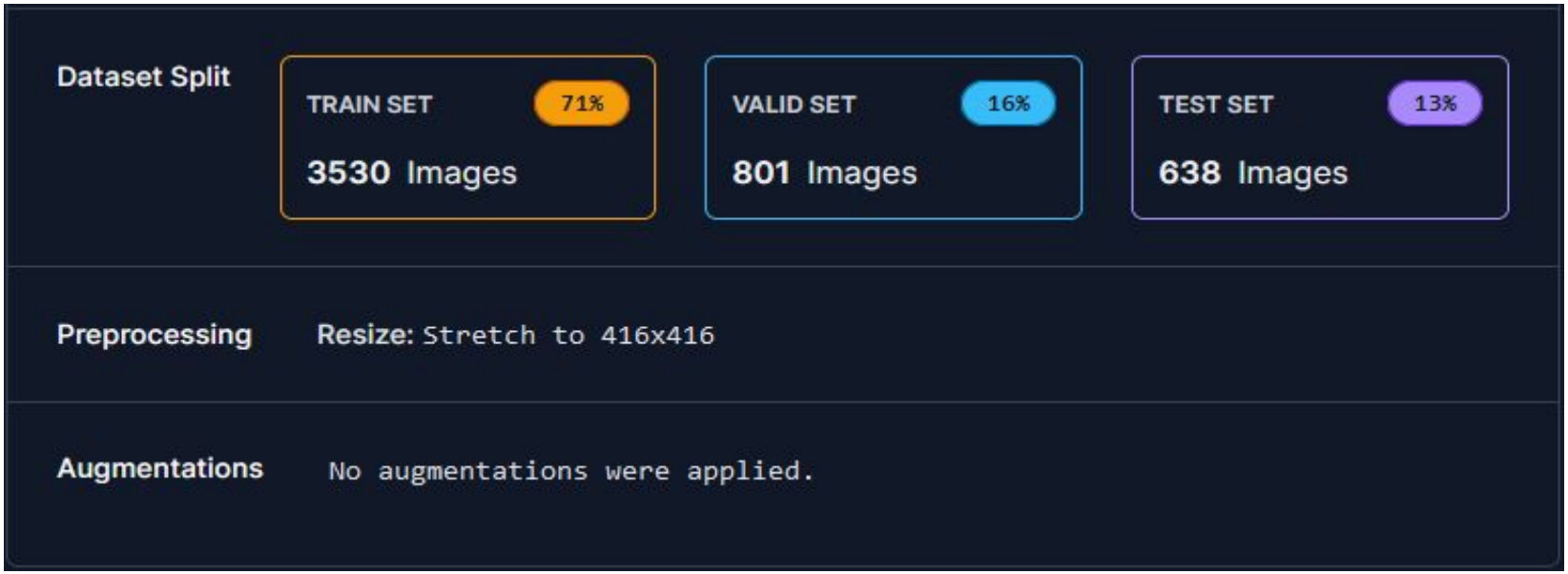
H. Cai, Q. Wu, T. Corradi, and P. Hall. The cross-depiction problem: Computer vision algorithms for recognising objects in artwork and in photographs.

Why YOLO in This Case

1. Extremely Fast, Real Time
2. Global Reasoning
3. Generalizable representation
4. Pre-Trained Model available

About Dataset

The dataset is a traffic sign image dataset containing 4969 samples, which dataset (as you can see in the image) is correctly divided into three parts:
Train,Valid,Test!!



Name of Classes: Green Light, Red Light, Speed Limit 10, Speed Limit 100, Speed Limit 110, Speed Limit 120, Speed Limit 20, Speed Limit 30, Speed Limit 40, Speed Limit 50, Speed Limit 60, Speed Limit 70, Speed Limit 80, Speed Limit 90, Stop

Preprocessing

Key	Value	Description
hsv_h	0.015	image HSV-Hue augmentation (fraction)
hsv_s	0.7	image HSV-Saturation augmentation (fraction)
hsv_v	0.4	image HSV-Value augmentation (fraction)
degrees	0.0	image rotation (+/- deg)
translate	0.1	image translation (+/- fraction)
scale	0.5	image scale (+/- gain)
shear	0.0	image shear (+/- deg)
perspective	0.0	image perspective (+/- fraction), range 0-0.001
flipud	0.0	image flip up-down (probability)
fliplr	0.5	image flip left-right (probability)
mosaic	1.0	image mosaic (probability)
mixup	0.0	image mixup (probability)
copy_paste	0.0	segment copy-paste (probability)

Code

Data.yaml

train: ../train/images

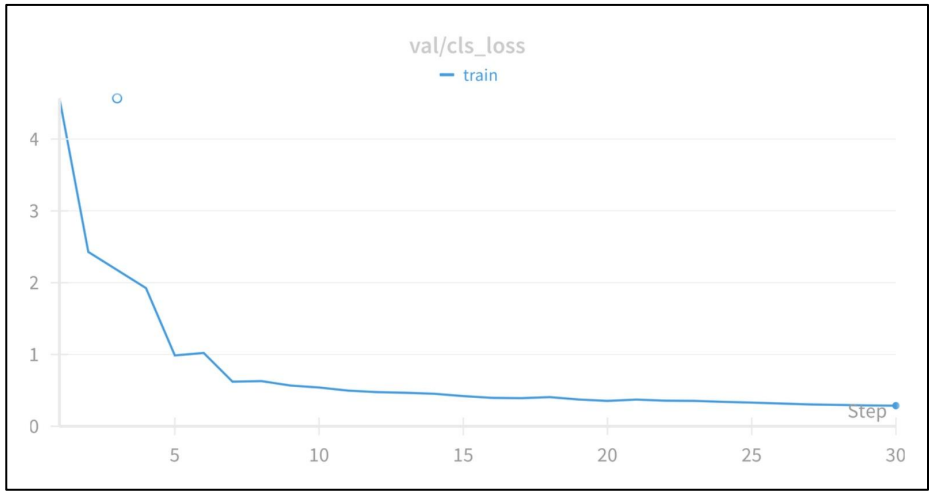
val: ../valid/images

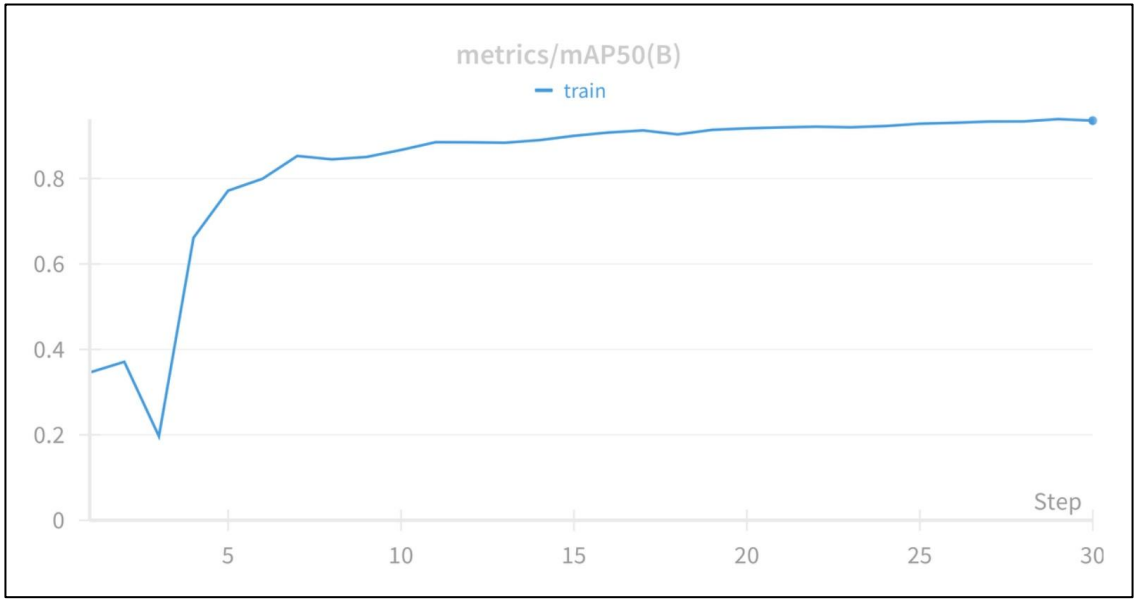
test: ../test/imagesnc: 15

names: ['Green Light', 'Red Light', 'Speed Limit 10', 'Speed Limit 100', 'Speed Limit 110', 'Speed Limit 120', 'Speed Limit 20', 'Speed Limit 30', 'Speed Limit 40', 'Speed Limit 50', 'Speed Limit 60', 'Speed Limit 70', 'Speed Limit 80', 'Speed Limit 90', 'Stop']

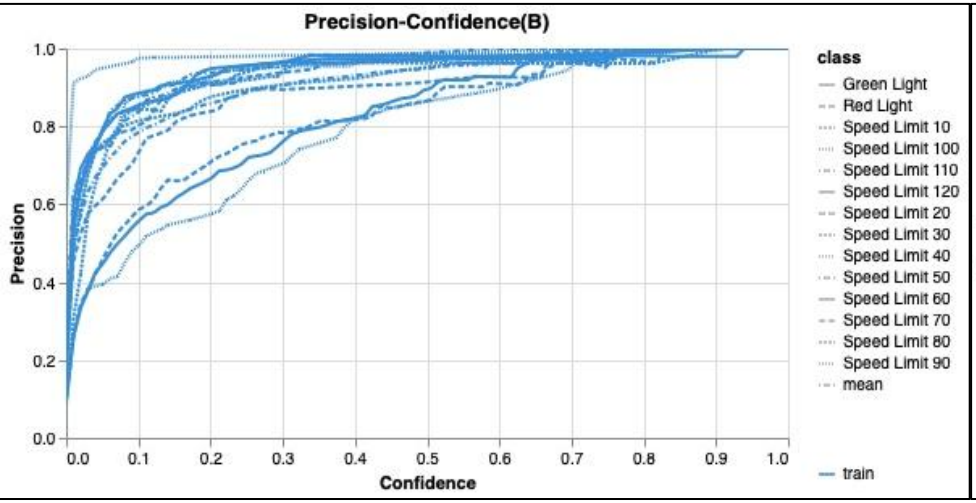
- YOLOv8m is loaded
- Images are resized to 320x320
- All the model metrics are logged using WandB
- The model is fine tuned for 30 epochs and good results are achieved

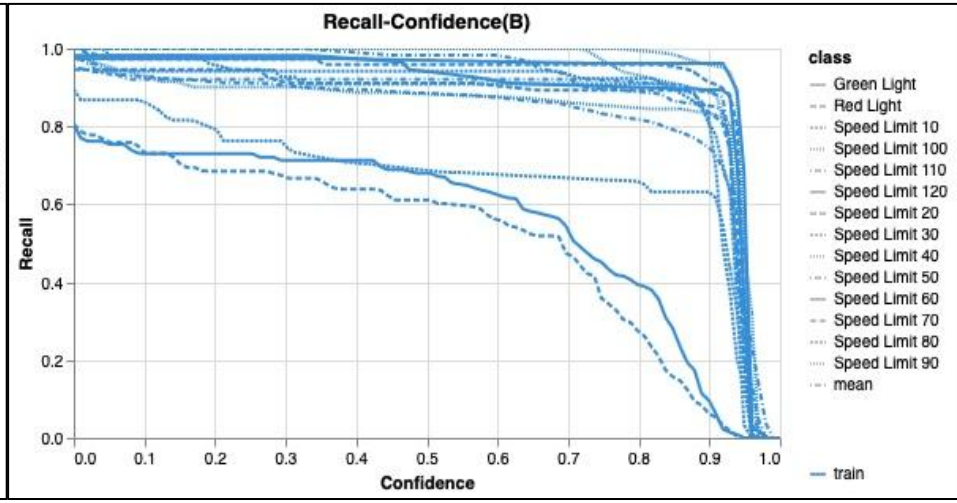
Training Metrics

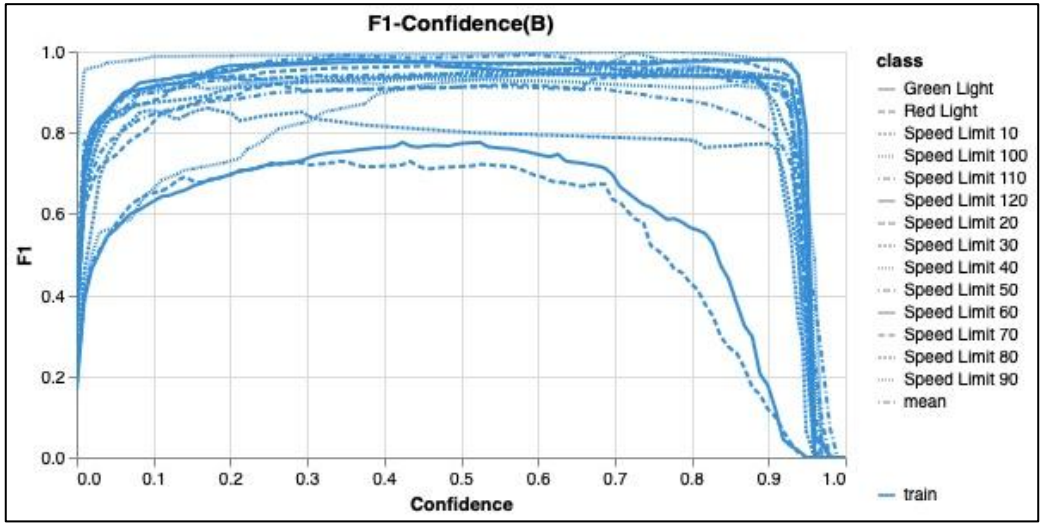




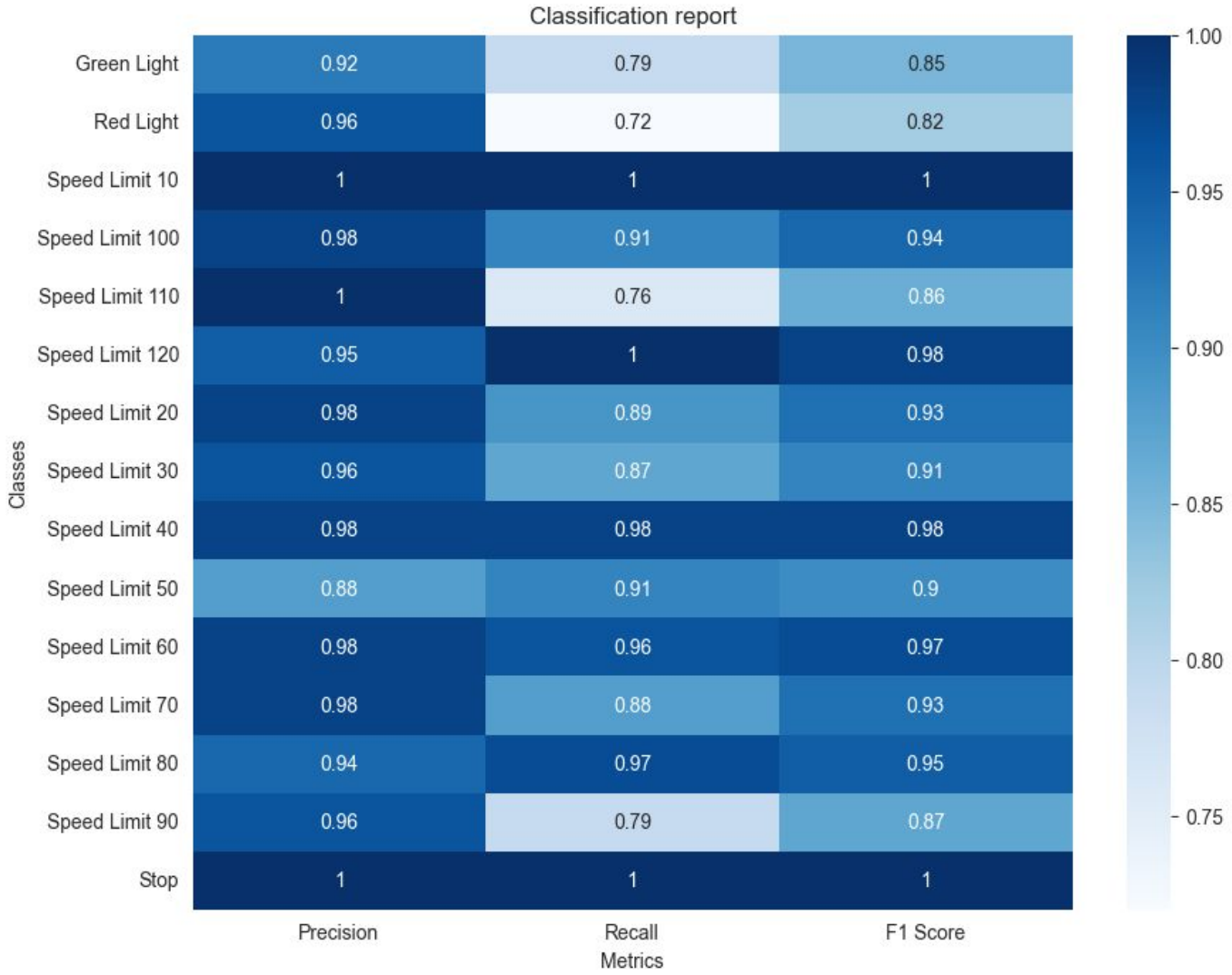
Metrics-Confidence Curve

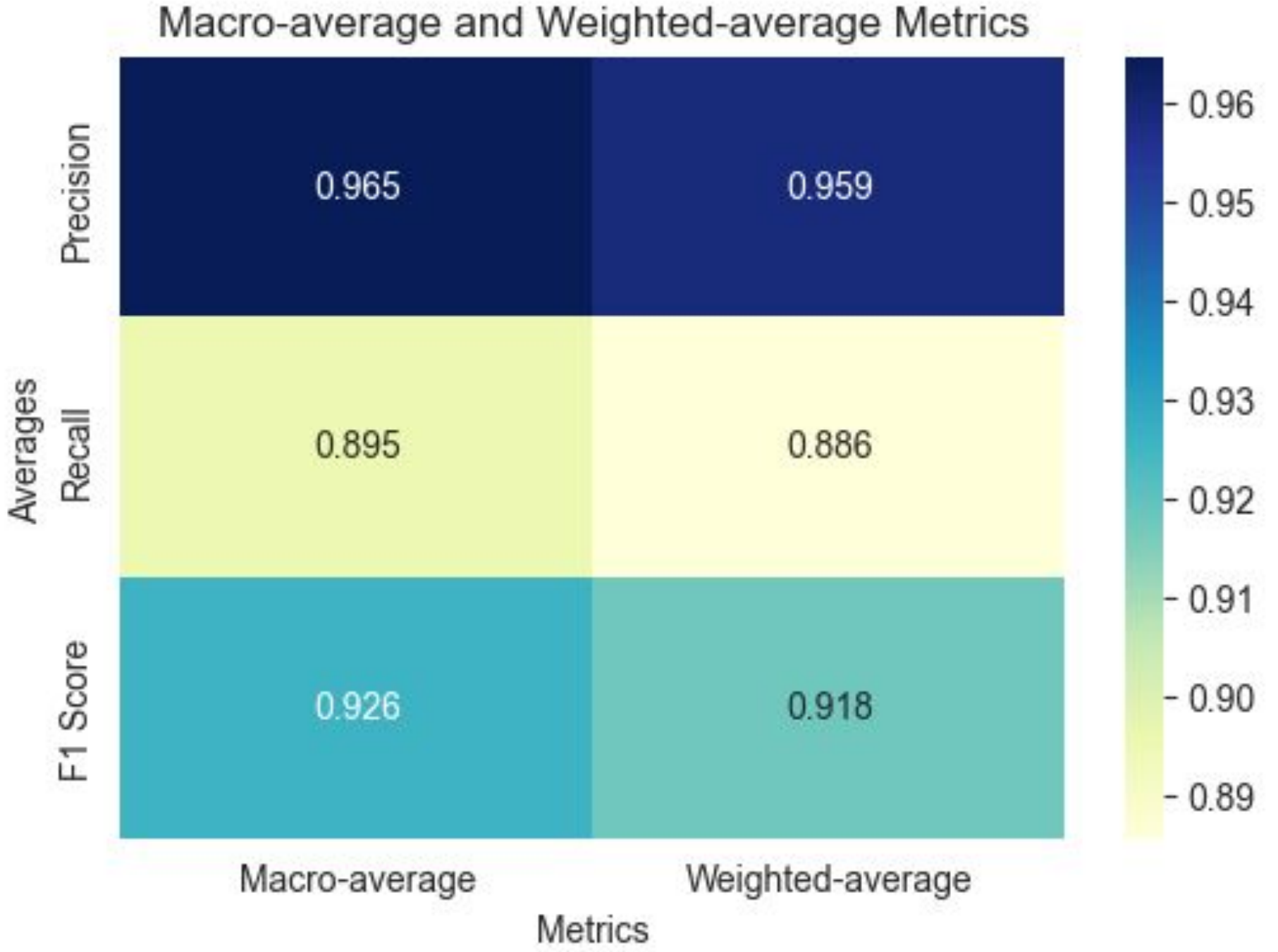






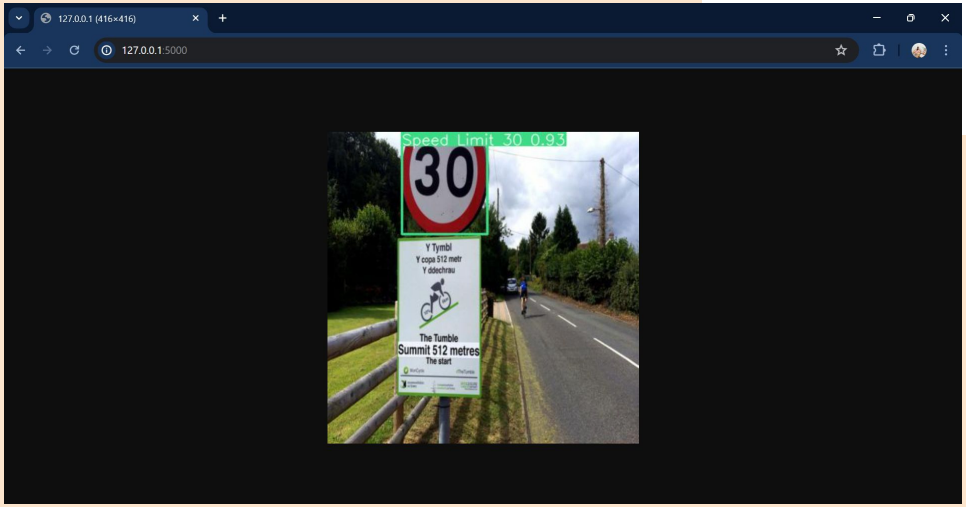
Evaluation on Test Data




Macro-average and Weighted-average Metrics
Precision
Recall
F1 Score
Averages
0.965
0.959
0.895
0.886
0.926
0.918
Macro-average
Weighted-average
Metrics
0.96
0.95
0.94
0.93
0.92
0.91
0.90
0.89

Demo

Flask App



3. Detected Road Sign on the Image.

Detection on Real Images

Challenges

1. **Dataset Acquisition:** Obtaining a diverse dataset with various traffic sign classes and robust data augmentation is crucial for effective model training.

2. **Data Availability:** Limited availability of annotated data, especially for less common sign classes, can hinder the development of accurate detection models.

3. **Real-Time Detection:** Balancing real-time processing with high accuracy is challenging, particularly for swift detection of traffic signs in video frames.

4. **Live Detection:** Integrating live detection into video streams requires optimization for speed and efficiency without compromising accuracy, adding complexity to implementation.

Future

1. **Expanded Classes:** Future efforts target a broader range of traffic sign classes, including regional and lesser-known signs, for enhanced adaptability across diverse environments.

2. **Precision Improvement:** Ongoing research emphasizes enhancing precision and accuracy through advanced techniques like fine-tuning architectures and utilizing multi-scale features.

3. **Real-Time Optimization:** Advancements in hardware and algorithmic optimization facilitate improved real-time detection, employing techniques such as model compression and acceleration for efficient video stream processing.

4. **Data Augmentation:** Advanced methods like generative adversarial networks (GANs) and synthetic data generation address data scarcity, enhancing the diversity and realism of training datasets.

Conclusion

Implementing Sign Detection using YOLOv8 holds tremendous potential across a wide range of practical applications. For example, it can greatly enhance traffic management systems, allowing for the efficient detection and recognition of various traffic signs. This technology can play a vital role in improving road safety by enabling vehicles to accurately interpret and respond to the information conveyed by road signs. Moreover, sign detection using YOLOv8 can assist in urban planning and infrastructure development by analyzing the presence and condition of signs in different areas

The model can be used in self-driving car systems to recognize traffic signs accurately. This would enable autonomous vehicles to follow traffic rules and regulations, analyzing every sign whether it’s about speed limit or stop and go indications to safely navigate the roads.

Learnings Applied

01 **DATA AUGMENTATION**

02 **YOLO**

03 **CLASSIFICATION**

04 **METRICS EVALUATION**

05 **FLASK APP**

Member Contributions

Deepmalya Dutta	MDS202218	Data Augmentation and Preprocessing
D Siva Manoj	MDS202216	YOLO and Other Model training
Vaishali Agarwal	MDS202250	Flask App
Varun Agrawal	MDS202251	Model Evaluation

Thank You