

GROUP 17

# LANE DETECTION FOR AUTONOMOUS VEHICLES

***Presented by***

D PRANEETH - CB.SC.U4AIE24109

MEERA S - CB.SC.U4AIE24133

MITHUL AKSHAY - CB.SC.U4AIE24135

RAAMAN NAMPUTHIRI - CB.SC.U4AIE24148



# INTRODUCTION

With the rapid growth of smart transportation technologies, improving road safety and making driving more efficient have become top priorities. **Artificial intelligence and computer vision** are playing a major role in transforming the way we drive by reducing accidents, increasing driver awareness, and laying the foundation for self-driving cars.

Our project, the **Lane and Object Detection System with Real-Time Feedback**, aims to bridge the gap between regular driving and autonomous vehicles. It uses advanced tools like **YOLOv3** for detecting objects in real-time and **OpenCV** for identifying lane boundaries, providing a reliable and helpful system to assist drivers on the road.



# OBJECTIVES

## **Lane Detection:**

- Accurately identify and track lane boundaries in real-time using image processing techniques like Gaussian Blur, HSV masking, and Hough Line Transformation

## **Object Recognition:**

- Implement YOLOv3 to detect and classify objects such as cars within the video feed with high confidence and precision.

## **Dynamic Driving Instructions:**

- Provide real-time feedback to drivers based on lane and object detection outcomes, such as:
  - “Stay on the Lane” to maintain safe lane discipline.
  - Go Slow, Traffic Ahead” when vehicles or obstacles are detected.



# METHODOLOGY

## 1. Load Required Files:

- Ensure the essential resources for the program are available. This includes the YOLOv3 weights file, configuration file, and the COCO names file, which contains the predefined object classes.

## 2. Load Object Class Names:

- Extract the list of object class names (e.g., "person", "car", etc.) from the COCO names file. These classes are used to identify and label objects detected by the YOLOv3 model.



### 3. Loading the Video

- **Input:**

- The system begins by loading a video file using `cv2.VideoCapture()`.
- This video simulates the driving environment by capturing road lanes, vehicles, and other objects.

- **Error Handling:**

- The program checks if the video file is successfully opened. If not, it exits with an error message to avoid further processing.

### 4. Load and Verify the Input Video

- Open the input video file to process it frame by frame.
- For each frame, read and process it sequentially.
  - If there is an issue reading the video (such as reaching the end of the file or encountering an error), the loop will terminate gracefully.



## 5. Noise Reduction

Since edge detection is highly susceptible to noise in the image, it is crucial to preprocess the image with noise reduction using a Gaussian filter.

**Process:**

### 1. Application of Gaussian Blur

- A Gaussian filter is applied to the image, using a kernel of weights to smooth the image.
- The kernel is a matrix of values generated using gaussian function. Pixels closer to the centre of the kernel have higher weights, while those further away have lower weights.
- $G(x,y) = (1/2\pi\sigma^2) * e^{-(x^2+y^2)/2*\sigma^2}$
- The kernel values follow a Gaussian distribution, giving more weight to the central pixel and decreasing weights for distant neighbours.





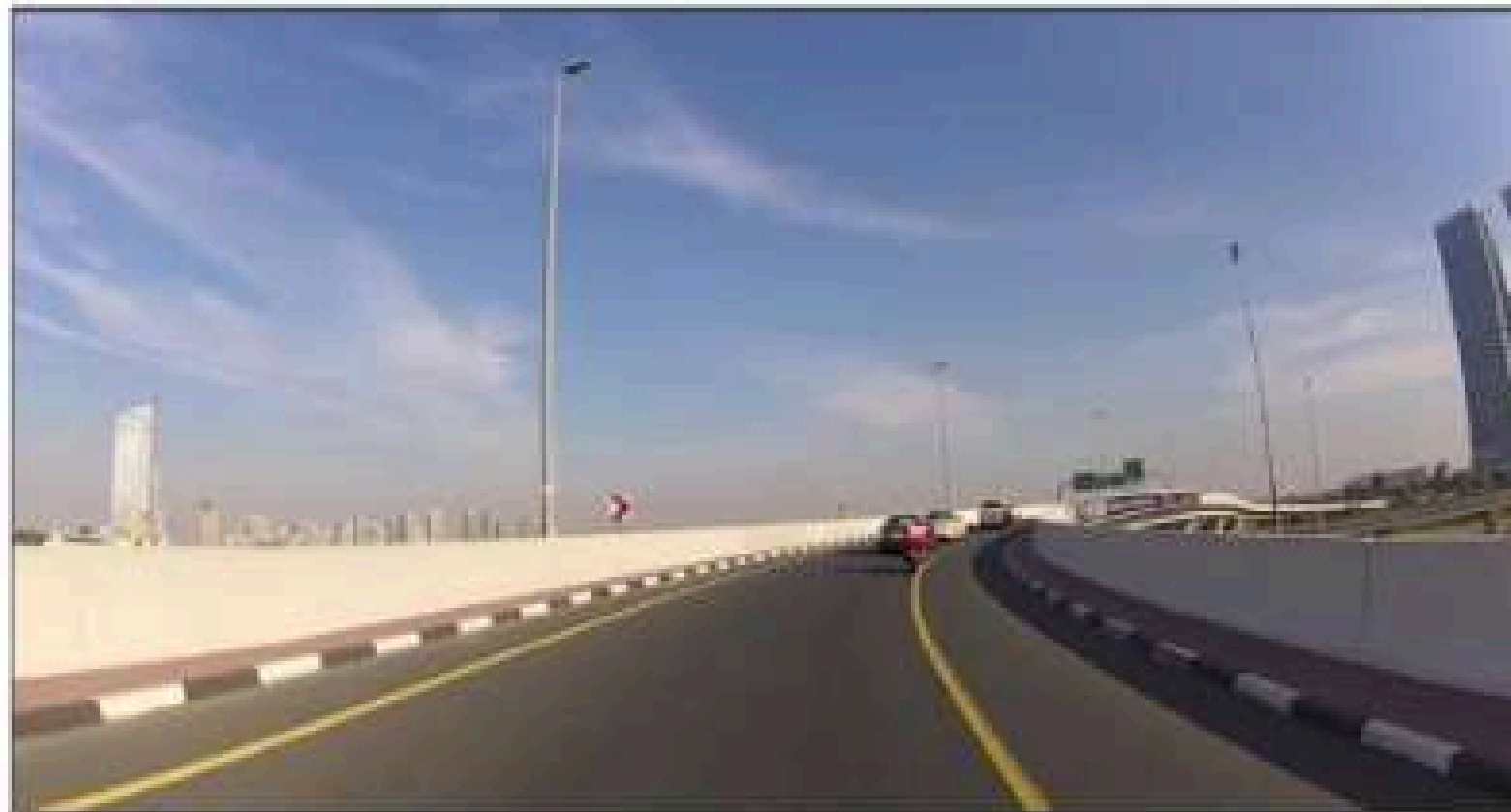
## 2. Pixel Value Adjustment:

- Each pixel's value is replaced by the weighted average of its neighbouring pixels.
- Pixels closer to the central pixel contribute more to the average, reducing the effect of random variations.

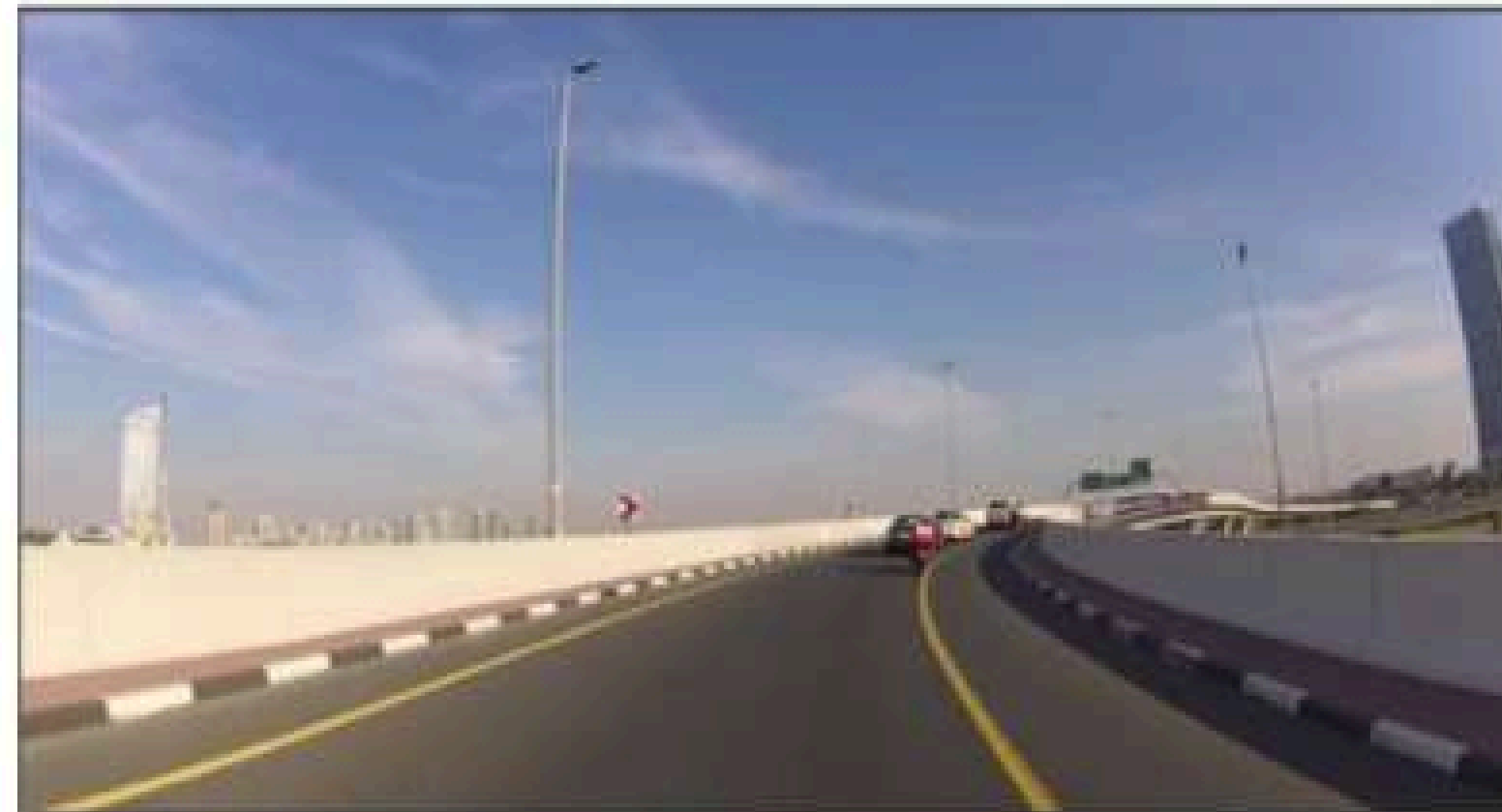
## 3. Noise Removal:

- This operation effectively removes random noise, which can interfere with edge detection and lane detection accuracy.
- By smoothing the image, it ensures only significant edges are highlighted, improving subsequent processing steps like edge detection and line extraction.

Original Image



Gaussian Blur





## 5. Image Preprocessing

To detect lanes effectively, the BGR image is converted to the HSV (Hue, Saturation, Value) colour space. The HSV model is advantageous as it separates image intensity (Value) from colour information (Hue and Saturation). This separation makes it highly effective in identifying specific colours, such as yellow lanes, regardless of varying lighting conditions.

### Process:

1. **Conversion:** The image is transformed from BGR (Blue, Green, Red) format to HSV.

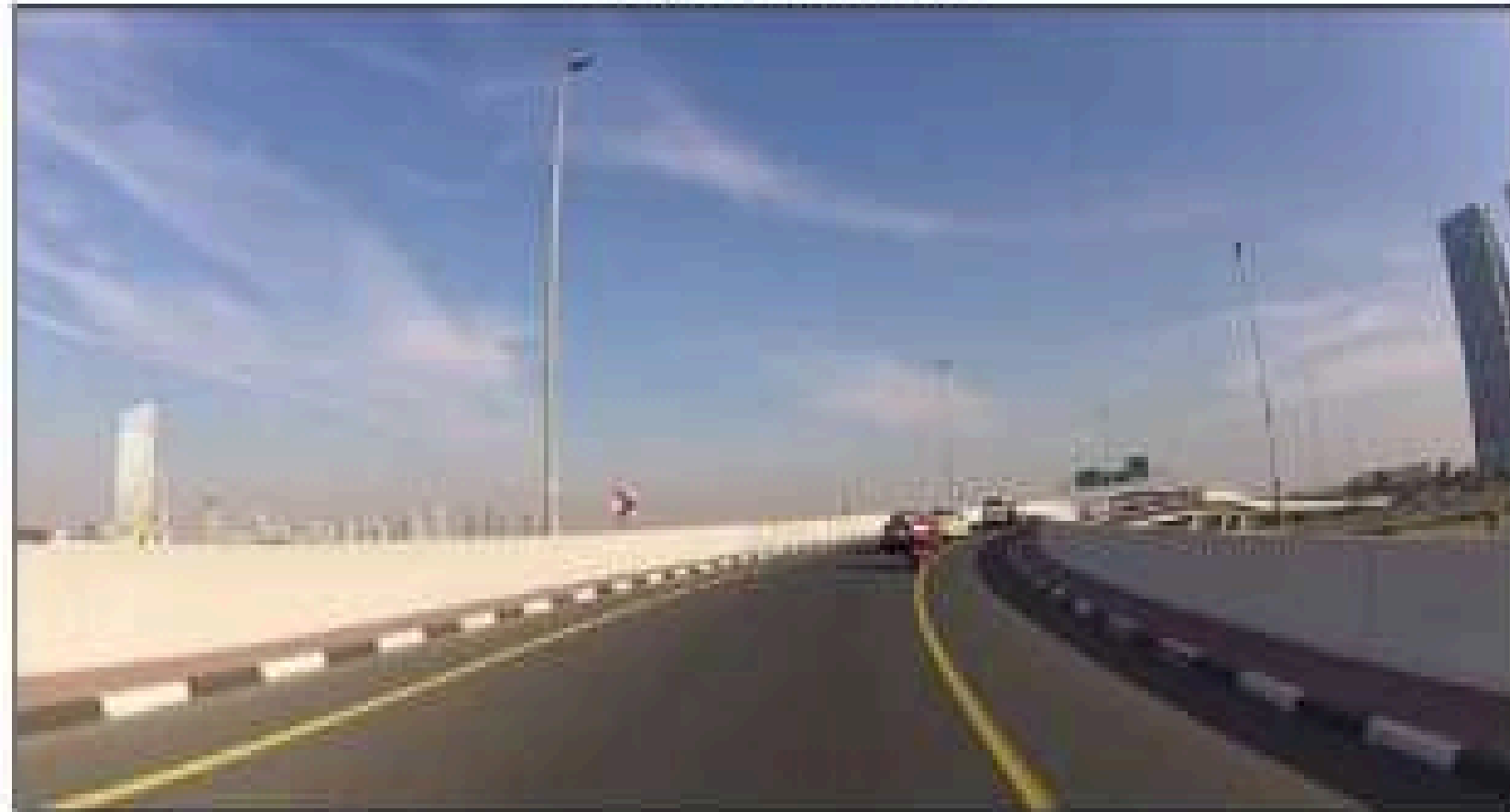
- **Hue:** Represents the type of colour (e.g., yellow, blue).
- **Saturation:** Indicates the intensity or purity of the colour.
- **Value:** Reflects the brightness or luminance of the colour.

2. **Thresholding:**

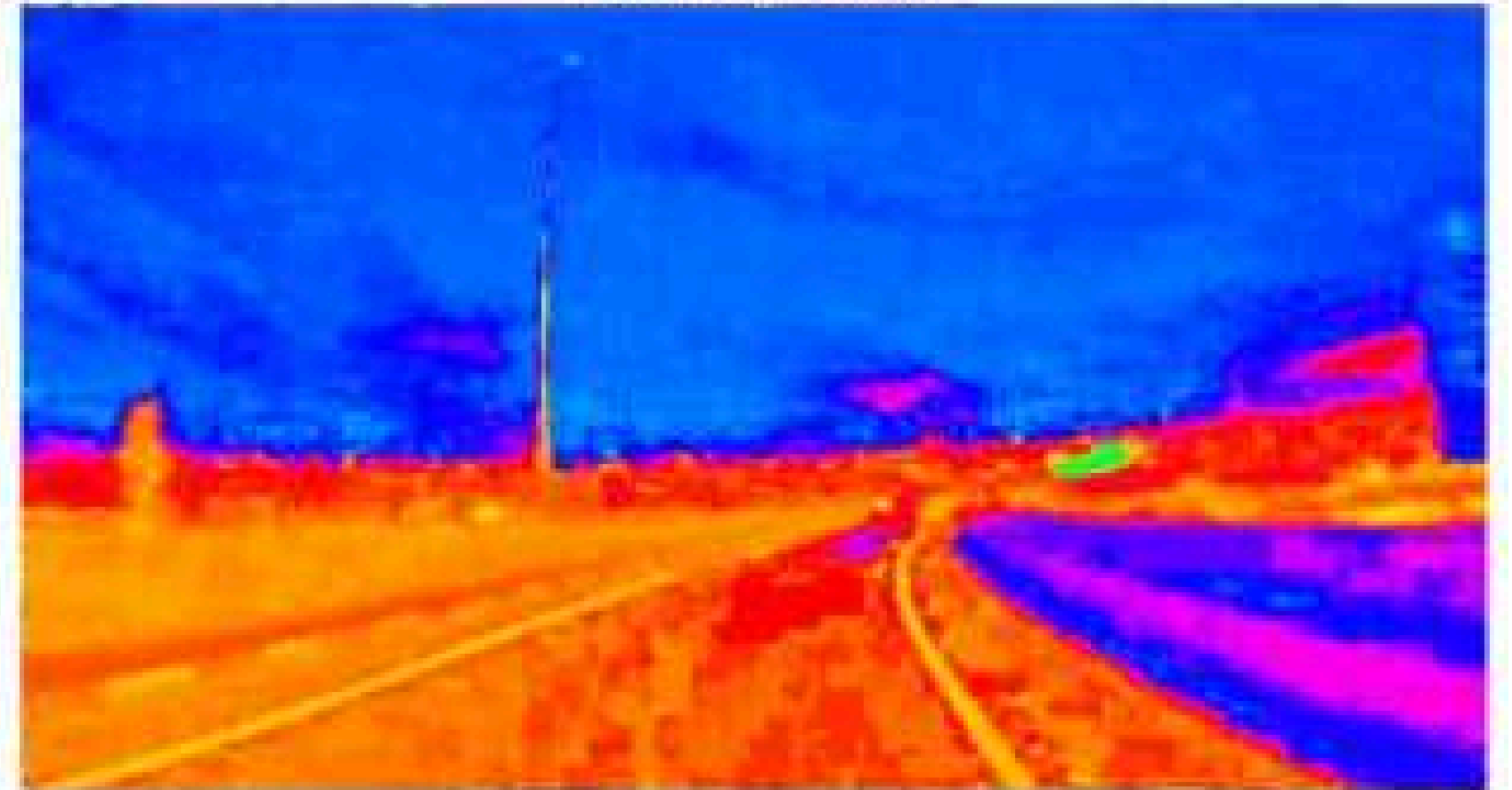
- A specific range of HSV values (defined by lower and upper bounds) is selected to isolate the desired colour (e.g., yellow lanes).
- Pixels that fall within this range are set to white in a binary mask, effectively highlighting the lanes of interest.
- Pixels outside this range are set to black, eliminating unwanted background information.



Original Image



Hue Channel





Saturation Channel



Value Channel





## 6. Canny edge detection

Canny Edge Detection is used to detect edges (sudden changes in intensity) in an image. This step is crucial for identifying lane boundaries after colour filtering.

**Process:**

### 1. Gradient calculation:

The first step in the Canny Edge Detection algorithm involves identifying the regions in an image where the pixel intensity changes significantly. This is achieved by calculating the gradient of the image.

- Here we find gradients in Gx and Gy directions.
- By finding gradients, we can determine the major changes in intensities.
- The edge direction is found using the formula:

$$\theta = \tan^{-1}(G_y/G_x)$$

- Gradient magnitude:

$$G = \sqrt{G_x^2 + G_y^2}$$



Gradient X



Gradient Y





## 2. Non-Maximum Suppression

After gradient calculation, edges may appear thick and blurry. Non-maximum suppression is applied to thin out these edges and retain only the most significant pixels along the direction of the edge.

The process involves:

- For each pixel, the gradient direction ( $\theta$ ) is used to approximate the neighbouring pixels along the edge direction.

The algorithm compares the gradient magnitude of the current pixel with those of its neighbours along the edge direction. Only the pixel with the maximum magnitude is retained, while others are suppressed (set to zero).



Magnitude



Non-Maximum Suppression





### 3. Double thresholding:

Double thresholding involves using two threshold values: minimum and maximum.

- **High Threshold:** Edges with intensity gradients greater than the maximum threshold are considered "strong edges" and are classified as definite edges.
- **Low Threshold:** Edges with intensity gradients below the minimum threshold are discarded as non-edges.
- Gradients between the two thresholds are marked as "weak edges" and are further processed to determine their significance.
- Threshold values are chosen experimentally by testing the algorithm on sample images and adjusting the values until the desired edge detection quality is achieved



Double Thresholding



Hysteresis





#### 4.Hysteresis Thresholding:

Hysteresis thresholding is used to finalize the edges by classifying weak edges.

- Weak edges are retained as part of the edge only if they are connected to a strong edge. This ensures continuity in edge structures and removes isolated weak edges.
- If a weak edge is not connected to any strong edge, it is discarded as noise.

This two-step thresholding process ensures that edges are accurately detected while reducing false positives caused by noise.



## 7. Hough Line Transform

The Hough Transform is used to detect the lane markings in the video.

Before applying the Hough Transform, the code first detects edges in the image using the Canny edge detector.

By detecting the lines that correspond to the lane markings, the code can determine if the vehicle is staying within the lanes

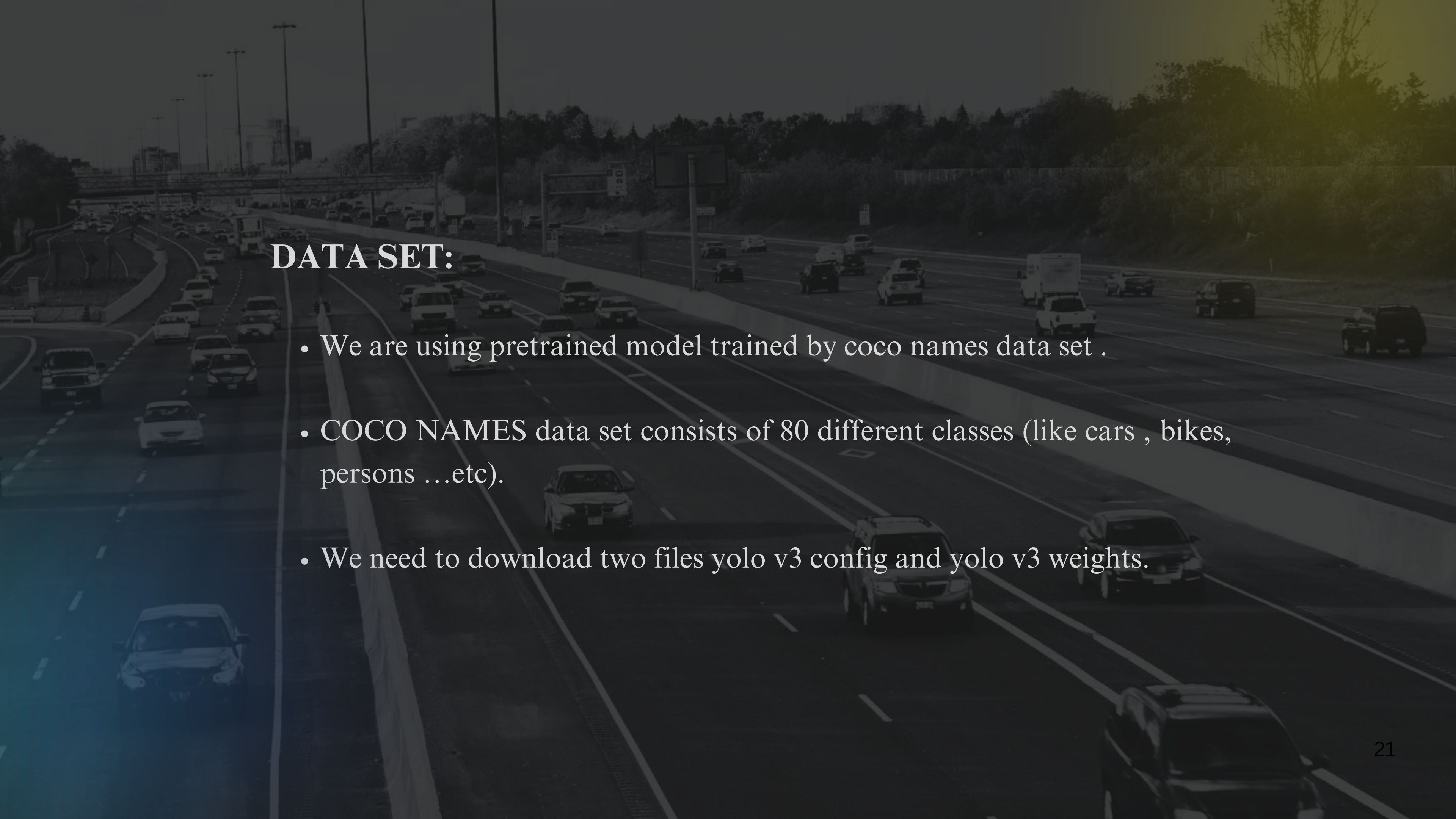
It can detect lines even if they are partially visible, noisy, or have gaps.





Hough Line Transform

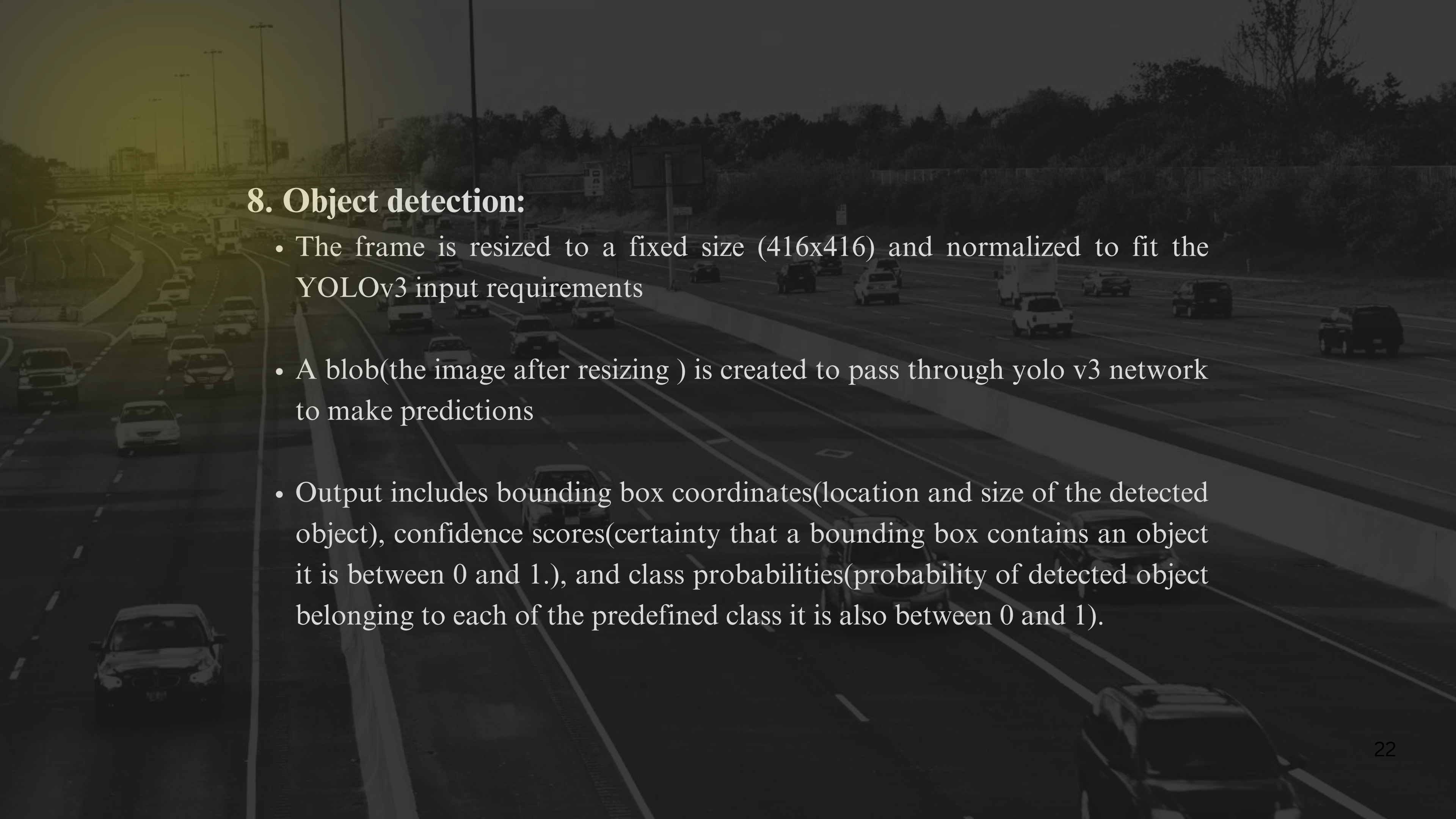




## DATA SET:

- We are using pretrained model trained by coco names data set .
- COCO NAMES data set consists of 80 different classes (like cars , bikes, persons ...etc).
- We need to download two files yolo v3 config and yolo v3 weights.

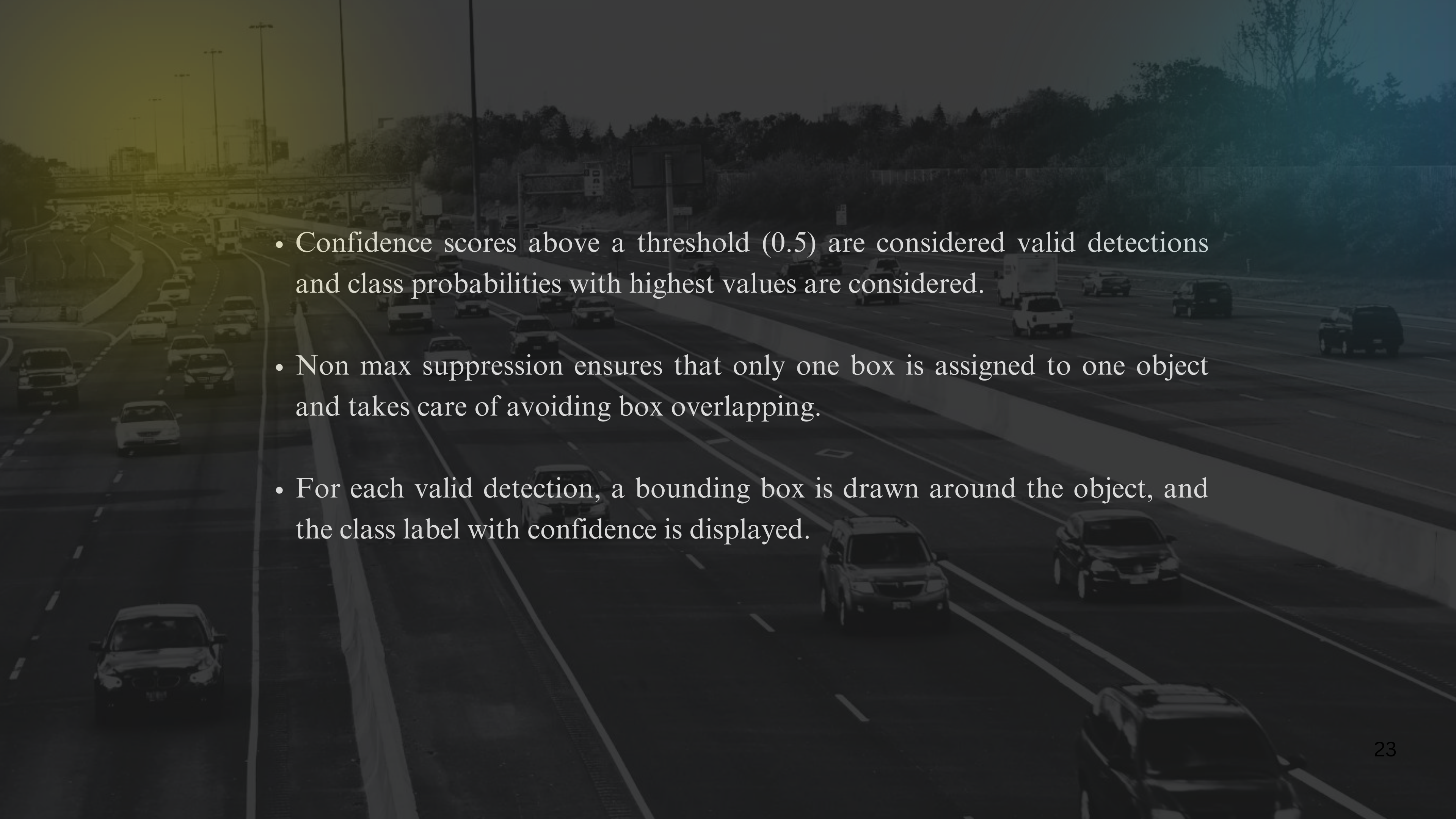




## 8. Object detection:

- The frame is resized to a fixed size (416x416) and normalized to fit the YOLOv3 input requirements
- A blob(the image after resizing ) is created to pass through yolo v3 network to make predictions
- Output includes bounding box coordinates(location and size of the detected object), confidence scores(certainty that a bounding box contains an object it is between 0 and 1.), and class probabilities(probability of detected object belonging to each of the predefined class it is also between 0 and 1).



- 
- Confidence scores above a threshold (0.5) are considered valid detections and class probabilities with highest values are considered.
  - Non max suppression ensures that only one box is assigned to one object and takes care of avoiding box overlapping.
  - For each valid detection, a bounding box is drawn around the object, and the class label with confidence is displayed.

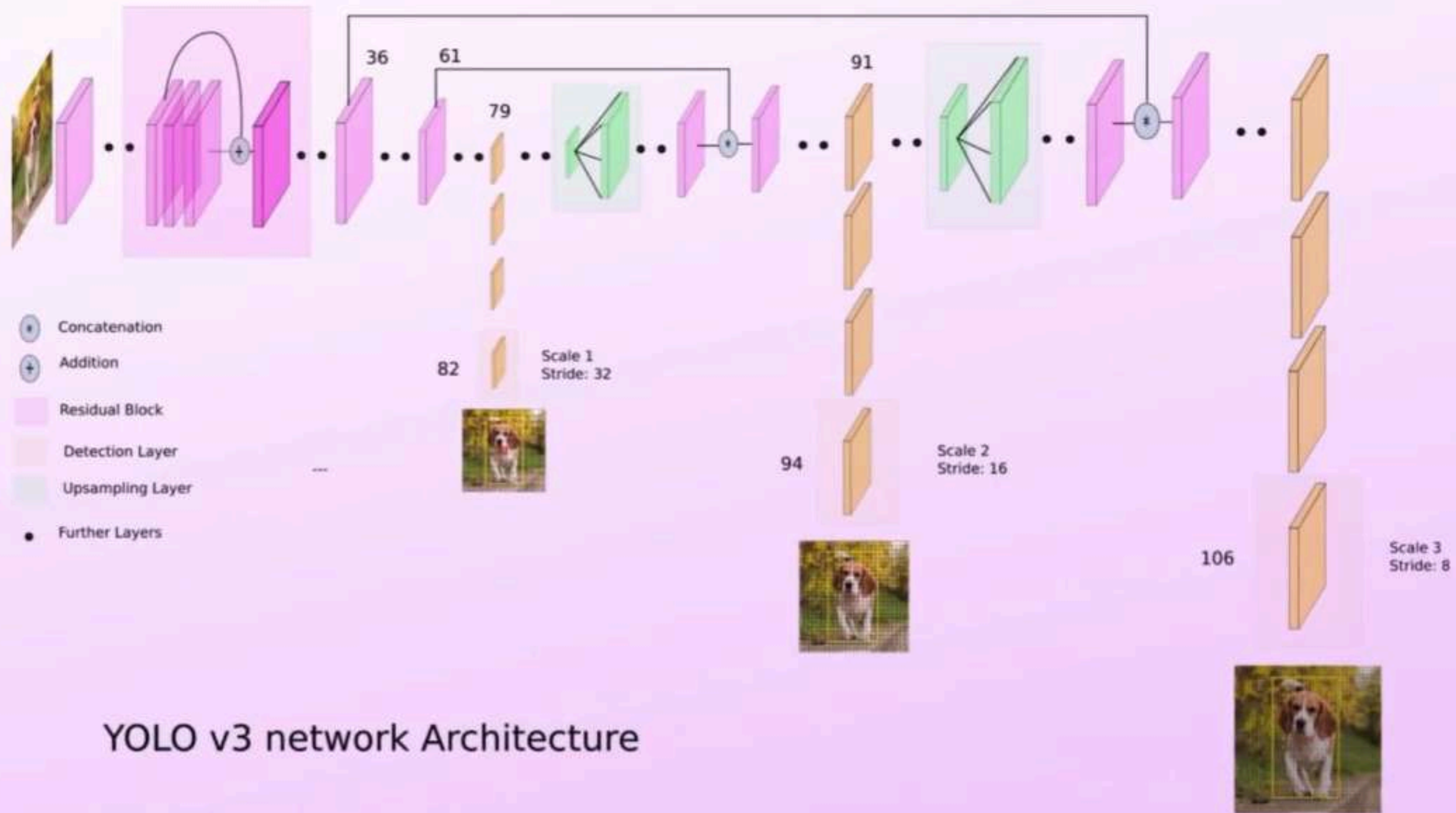


## YOLO V3 ARCHITECTURE:

1. YOLO (You Only Look Once): A real-time object detection algorithm.
2. YOLOv3: The third version of YOLO, known for its speed and accuracy.
3. Input Constraints:
  - YOLOv3 requires the input image to be resized to 416x416 pixels.
  - The image is normalized to values between 0 and 1 by dividing with 255 (scale factor).
  - Ensure image is not cropped while resizing.
4. Output:
  - Bounding box coordinates
  - Confidence score
  - Class probabilities
5. Key Features:
  - Real-Time Detection: Processes videos in real-time (30-45 FPS).
  - Multi-Scale Predictions: Detects objects of all sizes (small, medium, large).
  - Efficient: Lightweight and fast, suitable for mid-range hardware.



# Predictions across Scales







### **Advantage over other models:**

- YoloV1 predicts 98 boxes (7x7 grid cells, 2 boxes per cell @448x448)
  - YoloV2 predicts 845 boxes (13x13 grid cells, 5 anchor boxes per grid cell @416x416)
  - YoloV3 predicts 10647 boxes (13x13, 26x26, 52x52 grid cells, 3 anchor boxes per grid cell @416x416)
- YoloV3 predicts more than 10x the number of boxes predicted by YoloV2

### **Total Predictions:**

- Grids: 13x13, 26x26, 52x52
- Total Grids 3549
- Each grid has 3 boxes
- Total boxes =  $3549 * 3 = 10647$  boxes



# OVERVIEW OF METHODOLOGY

## 1. INPUT VIDEO PROCESSING

Load the video using OpenCV and verify successful loading. Sequentially process each frame.

## 2. PREPROCESSING WITH GAUSSIAN BLUR

Apply gaussian Blur to reduce noise and smooth the image.

## 3. COLOR FILTERING USING HSV

Convert the frame into HSV color space, isolate the lanes using defined HSV ranges.



#### **4. EDGE DETECTION WITH CANNY**

Highlight significant edges using canny Edge Setection with appropriate thresholds.

#### **5. LINE DETECTION USING HOUGH TRANSFORM**

Detect straight lines in the edge-detected image and overlay them on the original frame.

#### **6. VISUALIZATION**

Display the original frame with detcetd lines and the edge-detected frame.



# RESULT

## 1. Lane Detection:

- Green lines highlight the detected lanes.
- Instructions like "Stay on the Lane" or "Go Slow, Traffic Ahead" are displayed.

## 2. Object Detection:

- Bounding boxes with labels (e.g., "car", "person") are drawn around detected objects.







# METRICS

Method	Accuracy (%)
Canny Edge Detection [1]	85.0
Hough Line Transform [5]	80.5
YOLOv3 [4]	85.0



# **LIMITATIONS AND FUTURE SCOPE**

## **Nighttime Performance Issues –**

- The model struggles with lane detection in low-light conditions. This could be due to insufficient contrast between the lane lines and the road.

## **Glare and Shadows –**

- Headlights from other vehicles or strong shadows might cause false detections or missing lane lines.

## **FUTURE SCOPE**

- Train YOLOv3 on a Nighttime Dataset: Collect and annotate nighttime images for better generalization.





**THANK YOU  
SO MUCH!**

*Presented by* Group 17