

Traffic Density Estimator

By

Abdullah AlOwais

Reshoof Alzweaid

Rahaf Hasan

Afrah alanzi

Table of Contents

3.....	Introduction
3.....	Project Overview
4.....	Methodology
5.....	Code
5.....	EDA
6.....	YOLO Code
9.....	Results
9.....	EDA
9.....	Results after using YOLO
11.....	Task Schedule
11.....	Conclusion

4.....	Figure 1: YOLO Methodology
5.....	Figure 2: EDA1
6.....	Figure 3: EDA2
6.....	Figure 4: Code Fine- Tuning Yolo
7.....	Figure 5: Code
7.....	Figure 6: Code
8.....	Figure 7: Accuracy Curve
8.....	Figure 8: Classification Loss Learning Curve
9.....	Figure 9 : EDA Result
9.....	Figure 10: Result 1
10.....	Figure 11: Result 2
10.....	Figure 12: Result 3
11.....	Figure 13: Result 4

Introduction

Riyadh, the capital of Saudi Arabia, faces significant traffic challenges due to rapid urbanization and high vehicle density, leading to frequent congestion, especially during peak hours. To address these issues, a Traffic Density Estimator was developed using the YOLOv8 model, a state-of-the-art deep learning algorithm designed for processing and analyzing visual data. YOLOv8 automatically detects features in images, making it ideal for tasks like counting vehicles on roads. By analyzing traffic images, the Traffic Density Estimator accurately calculates the number of cars, providing valuable insights that can help reduce congestion. This system allows passengers to make informed decisions about alternative routes, ultimately improving traffic flow and safety in the city.

Project Overview

The Traffic Density Estimator project aims to tackle the significant traffic challenges in Riyadh, the capital of Saudi Arabia, by leveraging advanced deep learning techniques. Due to rapid urbanization and a high volume of vehicles, the city's roads are frequently congested, especially during peak hours. This congestion not only causes delays but also raises safety concerns and impacts the overall quality of life.

To address these issues, the project employs the YOLOv8 model, a state-of-the-art deep learning algorithm that excels in processing and analyzing visual data, such as images and videos. The YOLOv8 -based Traffic Density Estimator is designed to automatically detect and count the number of vehicles on the road by analyzing traffic images. This involves several key steps: feature extraction through convolutional layers, dimensionality reduction via pooling layers, and vehicle count prediction using fully connected layers.

The insights provided by the Traffic Density Estimator enable real-time monitoring of traffic conditions, helping to identify congested areas quickly. With this information, passengers can be informed of alternative routes, reducing overall traffic congestion and improving the efficiency of road use.

Methodology

YOLOv8 is an advanced version of the YOLO (You Only Look Once) object detection model, known for its efficiency and precision. The backbone of YOLOv8 consists of 53 convolutional layers, which are crucial for extracting detailed features from images. In total, the model includes 225 layers, designed to handle complex visual tasks with high accuracy. YOLOv8's architecture enables real-time object detection, making it highly effective for applications such as traffic density estimation, where speed and accuracy are essential.

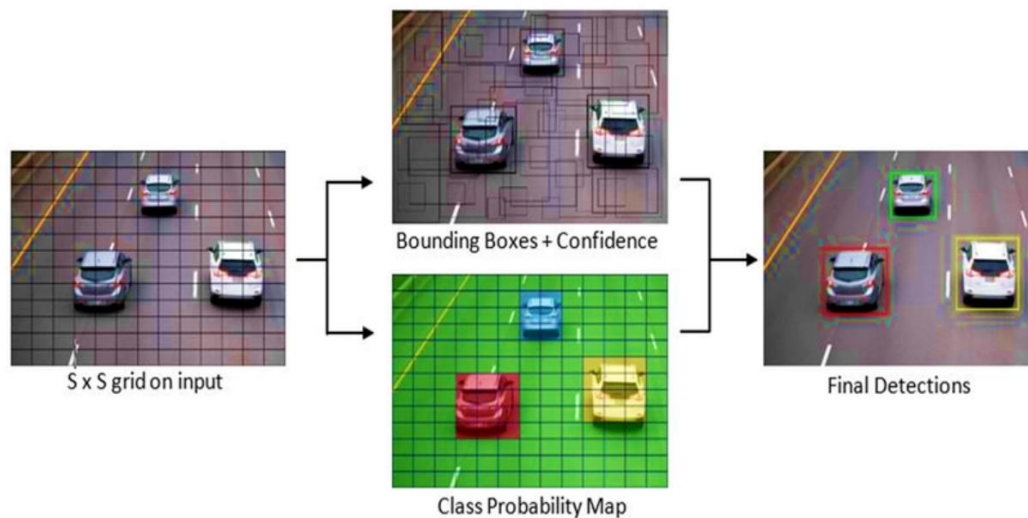


Figure 1: YOLO Methodology

Code

We developed a Traffic Density Estimator using YOLOv8, which is designed to calculate and measure traffic congestion by accurately counting the number of vehicles on the road. This tool provides real-time data on traffic density, allowing drivers to identify congested areas and choose alternative routes, ultimately improving traffic flow and reducing delays.

EDA

```

Data Exploration

train_images_path = os.path.join('/content/Vehicle_Detection_YOLOv8-3', 'train', 'images')
valid_images_path = os.path.join('/content/Vehicle_Detection_YOLOv8-3', 'valid', 'images')

num_train_images = 0
num_valid_images = 0

train_image_sizes = set()
valid_image_sizes = set()

for filename in os.listdir(train_images_path):
    if filename.endswith('.jpg'):
        num_train_images += 1
        image_path = os.path.join(train_images_path, filename)
        with Image.open(image_path) as img:
            train_image_sizes.add(img.size)

for filename in os.listdir(valid_images_path):
    if filename.endswith('.jpg'):
        num_valid_images += 1
        image_path = os.path.join(valid_images_path, filename)
        with Image.open(image_path) as img:
            valid_image_sizes.add(img.size)

print(f"Number of training images: {num_train_images}")
print(f"Number of validation images: {num_valid_images}")

if len(train_image_sizes) == 1:
    print(f"All training images have the same size: {train_image_sizes.pop()}")
else:
    print("Training images have varying sizes.")

if len(valid_image_sizes) == 1:
    print(f"All validation images have the same size: {valid_image_sizes.pop()}")
else:
    print("Validation images have varying sizes.")

Number of training images: 536
Number of validation images: 90
All training images have the same size: (640, 640)
All validation images have the same size: (640, 640)

```

Figure 2: EDA1

```
image_files = [file for file in os.listdir(train_images_path) if file.endswith('.jpg')]

num_images = len(image_files)
selected_images = [image_files[i] for i in range(0, num_images, num_images // 8)]

fig, axes = plt.subplots(2, 4, figsize=(20, 11))

for ax, img_file in zip(axes.ravel(), selected_images):
    img_path = os.path.join(train_images_path, img_file)
    image = Image.open(img_path)
    ax.imshow(image)
    ax.axis('off')

plt.suptitle('Sample Images from Training Dataset', fontsize=20)

Text(0.5, 0.98, 'Sample Images from Training Dataset')
```

Figure 3: EDA2

YOLO Code

```
▼ Fine-Tuning Yolo

• and Start training it

results = model.train(
    data='/content/Vehicle_Detection_YOLOv8-3/data.yaml',
    epochs=50,
    patience=15,
    imgsz=640,
    batch=16,
    optimizer='Adam',
    lr0=0.001,
    dropout=0.1,
    seed=0,
)
```

Figure 4: Code Fine- Tuning Yolo

```
[40] best_model = YOLO('/content/runs/detect/train/weights/best.pt')
      metrics = best_model.val(split='val')
      metrics_df = pd.DataFrame.from_dict(metrics.results_dict, orient='index', columns=['Metric Value'])
      metrics_df.round(3)
```

Ultralytics YOLOv8.2.77 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3,005,843 parameters, 0 gradients, 8.1 GFLOPs
val: Scanning /content/Vehicle_Detection_YOLOv8-3/valid/labels.cache... 90 images, 0 backgrounds, 0 corrupt
os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX is multithreaded, so this
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 90 937 0.932 0.924 0.973 0.736
Speed: 1.5ms preprocess, 27.7ms inference, 0.0ms loss, 6.6ms postprocess per image
Results saved to runs/detect/val

	Metric Value
metrics/precision(B)	0.932
metrics/recall(B)	0.924
metrics/mAP50(B)	0.973
metrics/mAP50-95(B)	0.736
fitness	0.760

Figure 5: Code

▼ converting from ".avi" to ".mp4"

```
[48] !ffmpeg -y -loglevel panic -i /content/runs/detect/predict/sample_video_after.avi sample_video_after.mp4
```

Video("sample_video_after.mp4", embed=True, width=960)

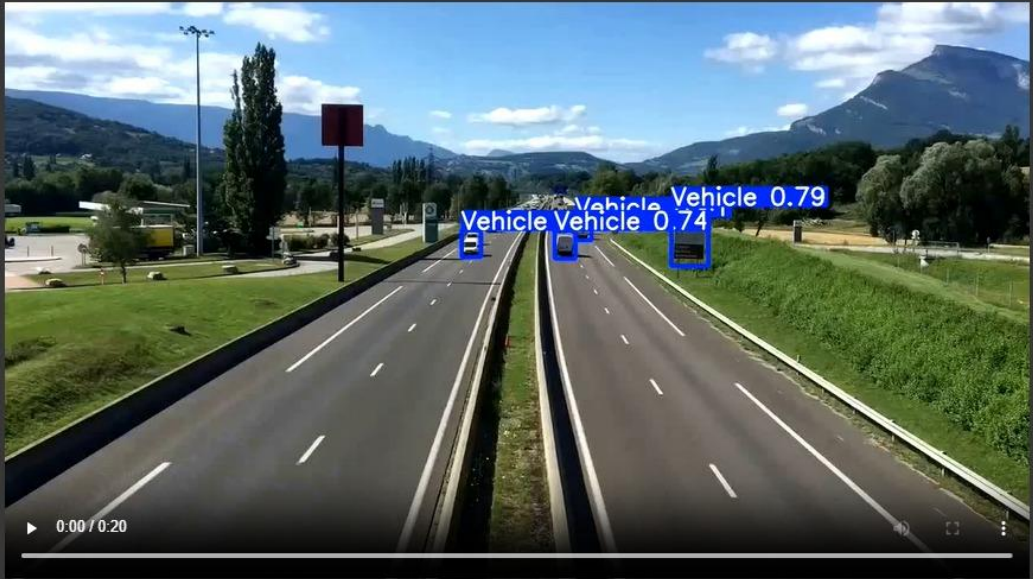


Figure 6: Code

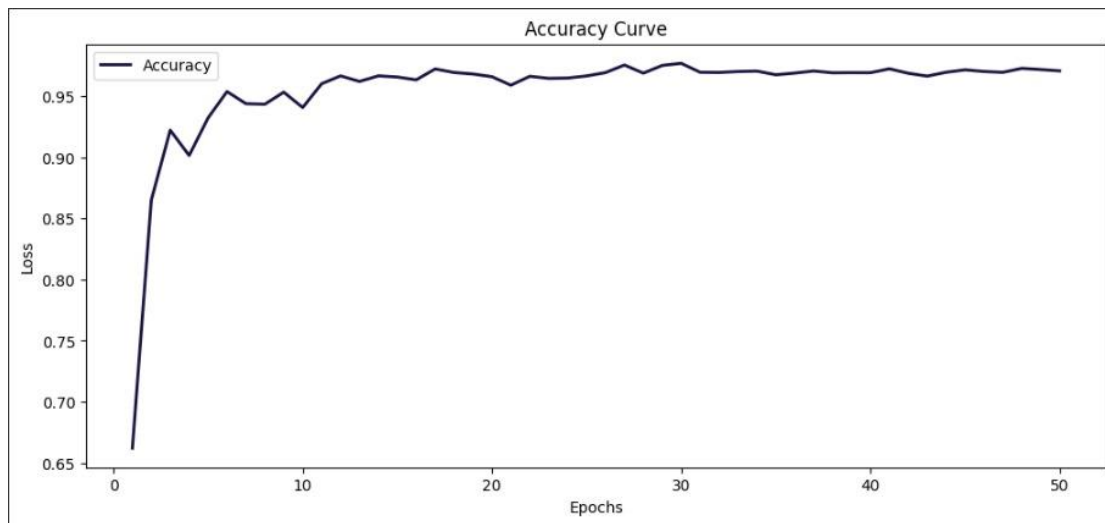


Figure 7: Accuracy Curve

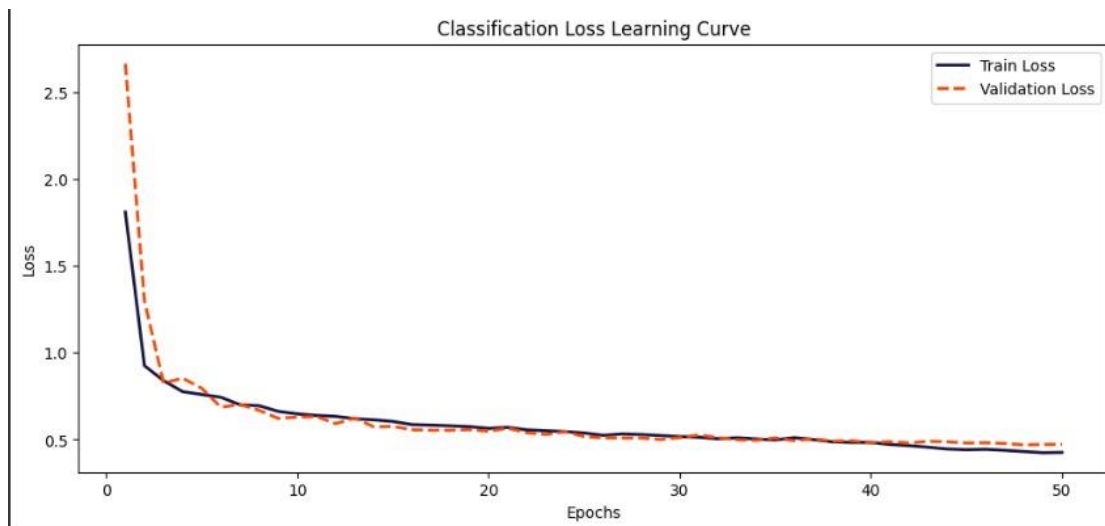


Figure 8: Classification Loss Learning Curve

Results

EDA

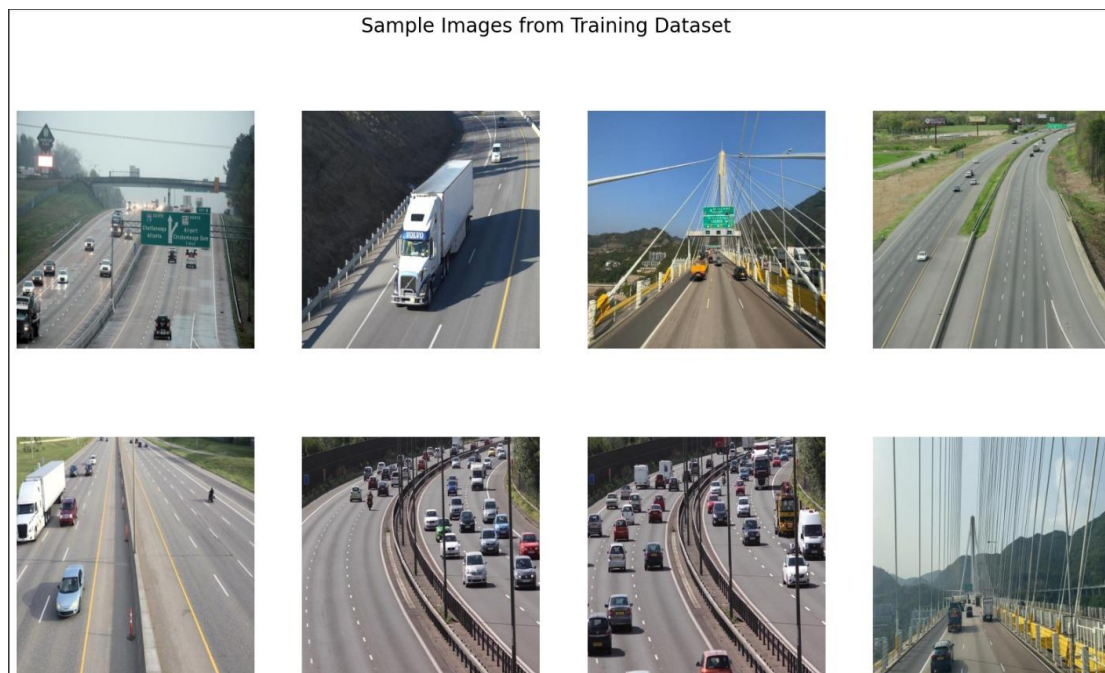


Figure 9 : EDA Result

Results after using YOLO

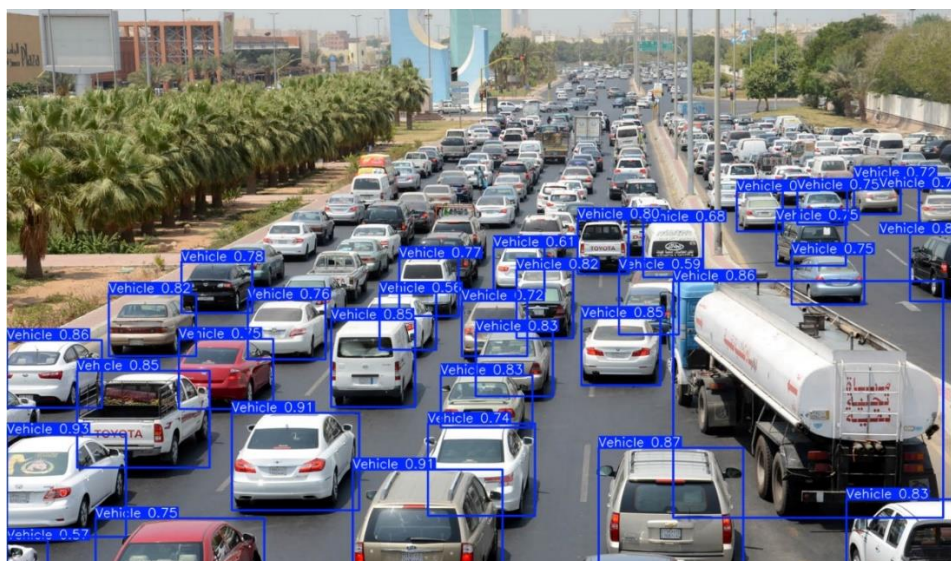


Figure 10: Result 1

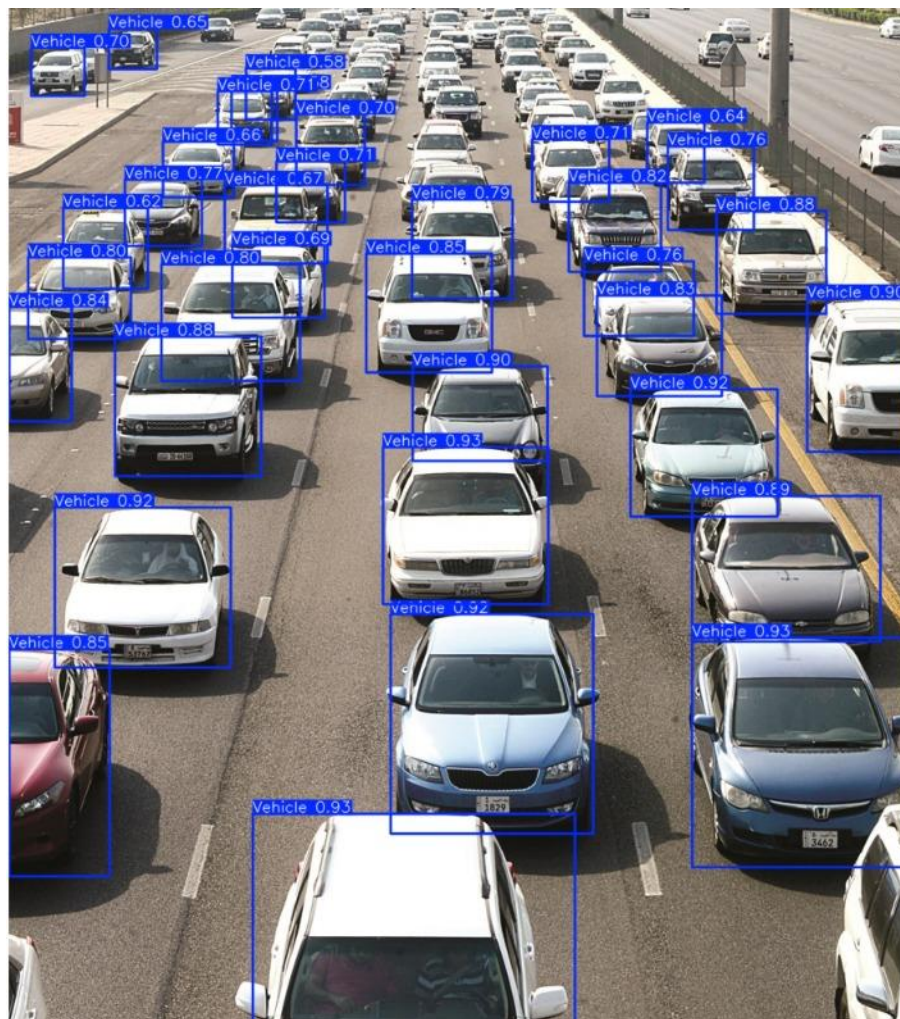


Figure 11: Result 2

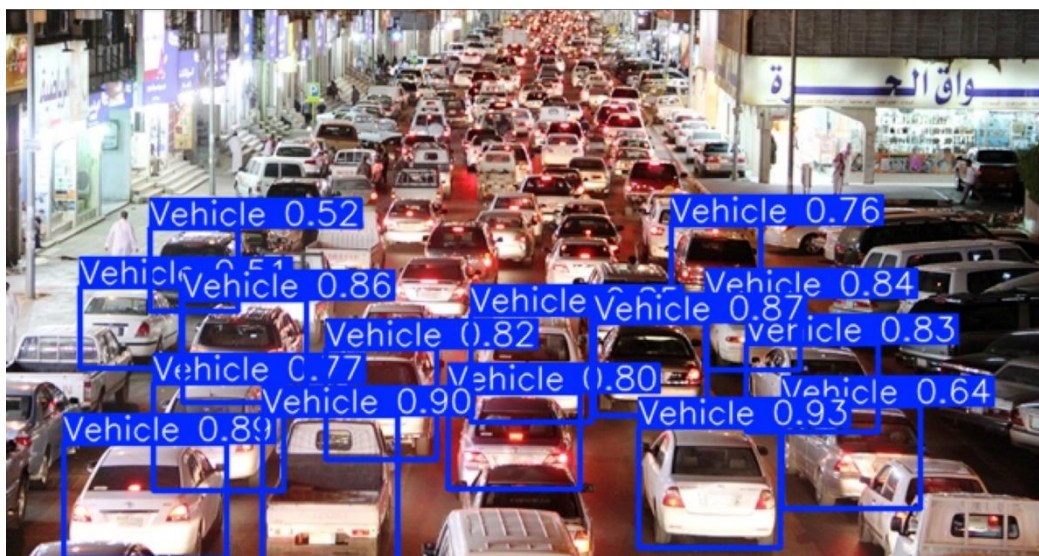


Figure 12: Result 3

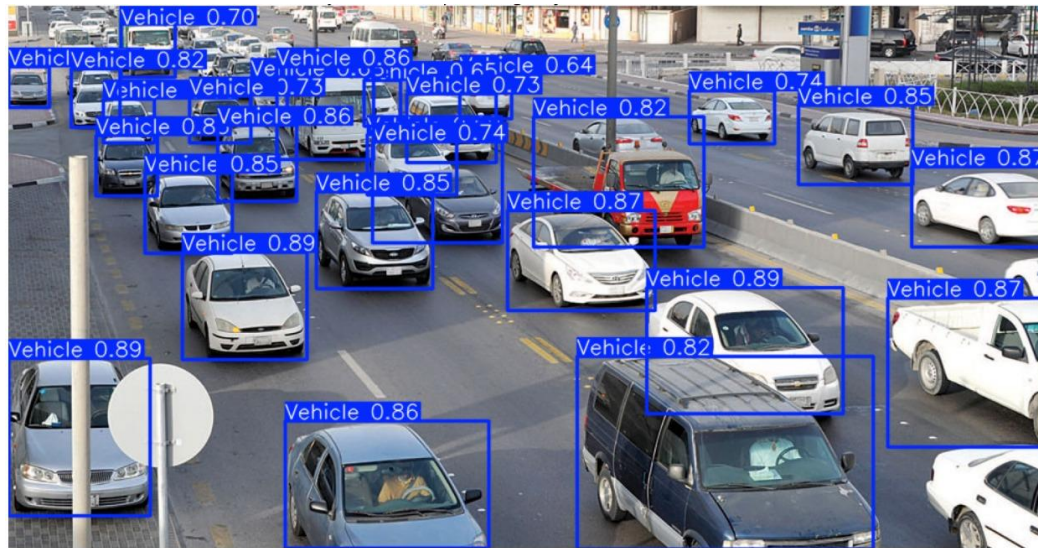


Figure 13: Result 4

Task Schedule

Name	Tasks
Abdullah AlOwais	Model trining
Reshoof Alzweaid	Data collection and Preprocessing
Rahaf Hasan	Report
Afrah alanzi	Presentation

Conclusion

We successfully developed an efficient traffic congestion detection system using the YOLOv8 model, a cutting-edge computer vision technology. The system analyzes images and videos captured from surveillance cameras to accurately assess the level of congestion on the roads, enabling real-time monitoring and more informed decision-making to improve traffic management.