Machine Vision Homework

Detect cars in a video using YOLOv8 in Python

Pucsok Sándor

PP22N9

2023

# Table of content

[Table of content 1](#_Toc151756639)

[1. Introduction 2](#_Toc151756640)

[2. Preprocess data 3](#_Toc151756641)

[2.1 Obtaining dataset 3](#_Toc151756642)

[3. Train network with own dataset 4](#_Toc151756643)

[4. Results 8](#_Toc151756644)

[5. Conclusion 12](#_Toc151756645)

[6. References 13](#_Toc151756646)

# Introduction

In this assignment, I will present the implementation of training the XOLOv8 model using a preprocessed dataset that I downloaded. The dataset underwent thorough preparation to ensure its suitability for training, and the XOLOv8 model was chosen for its efficiency in handling computer vision tasks. Following the training phase, I conducted testing on the model's performance by evaluating it on a set of video data.

After the training phase, I assessed the model's effectiveness through a testing process conducted on video data. This step is crucial to validate the model's generalization capabilities and its ability to accurately detect and classify objects in real-world scenarios. The following sections will delve into the details of the dataset preparation, training methodology, and the results obtained from the video testing. Through this assignment, I aim to demonstrate the practical application of the XOLOv8 model in computer vision tasks and highlight its performance on a real-world dataset.

1. **Pre-process data**

This chapter describes how to get and prepare a dataset for teaching.

* 1. **Obtaining dataset**

I downloaded the images and label txt-s for teaching from page The KITTI Vision Benchmark Suite [1]. There were 7481 pictures and labels in this package, but not in the format I will need for teaching YOLOv8. I only needed pictures with cars on them, but there were many spleens with only pedestrians, cyclists or other objects on them. As shown in Figure 2.1, the txt is structured as follows. For me, the coordinates of the bounding box are the important ones, i.e. the 5th,6th,7th,8th elements.

A screenshot of a computer screen

Description automatically generated

2.1. Figure Label format

As shown in the initial state of txt in Figure 2.2.

A car on a street

Description automatically generated

A white background with black numbers

Description automatically generated

2.1.1. Figure Example picture and original format of label

So, I made a code to filter out the txt's that have the word "Car" in them, and put those txt's and the images with the same name in a separate folder. of the original 7481 images and titles, 6679 images remain after filtering.

Then the coordinates were recalculated in the required format for YOLO. In addition, I deleted the lines that did not start with "Car" to avoid confusion when teaching. In the next step, I added a "0" character in front of each line, since this first character will be the "car" class when teaching. Then normalized the next 4 coordinates and finally created the final label for teaching, which is shown in Figure 2.3

A car on a street

Description automatically generated

A number of a number

Description automatically generated with medium confidence

2.1.2. Figure Final label format

where the data are: object class, x center, y center, width, height of bounding box.

1. **Train network with own dataset**

In this chapter I will show you how I have trained the net. in the first step I organized the already prepared data, creating two folders, one for images and one for labels. In both folders I created the following folders: train, val, test, which is shown in Figure 3.1.

3.1 Figure Train fields

|  |  |
| --- | --- |
|  |  |

The partitioning of the folders was as follows, the training set consisted of 3000 images, the validation and the test set of 400-400. After creating the teaching folders I created the corresponding *.yaml* file.

1. Code pathsForTraining.yaml

# Train/val/test sets

path: C:\Users\pucso\Desktop\Vehicle\_detecting\dataForYOLO # dataset root dir

train: images\train # train images

val: images\val # val images

test: images\test #test images

# Classes

names:

  0: car

Once everything was ready to be taught, I wrote the code to teach it, which was as follows, shown in code snippet two. The provided code conducts training with a YOLOv8 (You Only Look Once) object detection model using the ultralytics library. The ultralytics is a framework that facilitates easy and efficient training and evaluation of YOLO models.

2. Code trainOwnYoloV8.py

# Import the YOLO class from the ultralytics library

from ultralytics import YOLO

# Initialize the YOLO object and provide the path to the trained model weights

model = YOLO("yolov8s.pt")

# Perform training of the model with the specified data and for one epoch

# The data is stored in a YAML file containing paths to training images and labels

results = model.train(data="pathsForTraining.yaml", epochs=1)

Details of the code:

* *from ultralytics import YOLO*: Import the YOLO class from the ultralytics library.
* *model = YOLO("yolov8s.pt")*: Initialize the YOLO object with the path to the weights of the YOLOv8 model. At this point, the model is ready for training and evaluation.
* *results = model.train(data="pathsForTraining.yaml", epochs=1)*: Conduct training with the specified data. The data parameter refers to a YAML file containing paths to images and labels used for training. The epochs parameter determines the number of training epochs. The results of the training are stored in the results variable.
* After training, the *results* variable contains the obtained results and statistics, which can be utilized for evaluating and fine-tuning the model.

After finishing the train, I wanted to test the model on a video, so I wrote a separate code for that. This Python script utilizes the YOLOv8 model to perform real-time object detection on a video file. The YOLOv8 model is loaded with pre-trained weights, and each frame of the video is processed to detect and annotate cars. The results are displayed on the video frames in real-time, and the annotated video is saved as 'output.mp4' after processing. The script uses the OpenCV library for video handling and visualization. You can see the code in action in the following image, which is extracted from the video. I shot the video from the quay at Szent Gellért Square, where I filmed passing cars.

Cars on the road with a boat in the water

Description automatically generated

3.2. Figure Video evaluation during the run

3.1 Code evaluation.py

import cv2

from pathlib import Path

from ultralytics import YOLO

# Path to YOLOv8 model weights

yolo\_weights\_path = r'C:\Users\pucso\Desktop\Vehicle\_detecting\runs\detect\train2\weights\best.pt'

# Path to the video

video\_path = r'C:\Users\pucso\Desktop\Vehicle\_detecting\P1010559.MOV'

# Import YOLOv8 library

from ultralytics import YOLO

# Load YOLOv8 model

model = YOLO(yolo\_weights\_path)

# Open video file

cap = cv2.VideoCapture(video\_path)

while True:

    # Read video frame

    ret, frame = cap.read()

    if not ret:

        break

    # Use YOLOv8 model on the frame

    results = model(frame)

    # Display results on the frame

    for result in results:

        boxes = result.boxes

        for box in boxes:

            if len(box.xyxy[0]) != 0:

                x1, y1, x2, y2 = tuple(map(int, box.xyxy[0]))

                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)

                labels = "Car"

                conf = "{:.2f}".format(box.conf[0])

                cv2.putText(frame, f'{labels} {conf}', [x1, y1], cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)

    # Display

    cv2.imshow('YOLOv8 Detection', frame)

    # Exit if the user presses the 'esc' key

    if cv2.waitKey(1) == 27:

        break

# Save the output video

fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')

out = cv2.VideoWriter('output.mp4', fourcc, 30.0, (frame.shape[1], frame.shape[0]))

out.write(frame)

out.release()

# Release resources

cap.release()

cv2.destroyAllWindows()

3.2 Code evaluation.py

1. **Results**

After teaching, the validation process is coming:

* The validation is made on 400 images and two epochs.

# Import the YOLO class from the ultralytics library

from ultralytics import YOLO

# Initialize the YOLO object and provide the path to the trained model weights

model = YOLO(r"C:\Users\pucso\Desktop\Vehicle\_detecting\runs\detect\train2\weights\best.pt")

# Perform training of the model with the specified data and for one epoch

# The data is stored in a YAML file containing paths to training images and labels

results = model.val(data="pathsForTraining.yaml", epochs=2)

4. Code Validate network

A computer code with numbers and letters

Description automatically generated

* Precision = 0,817
* Recall = 0,732

In the following pictures you can see the performing of the network:

4.1. Figure Validation progress

A blue squares with white text

Description automatically generated

4.2. Figure Normalized confison matrix

|  |  |
| --- | --- |
|  | |
|  |  |

4.2. Figure F1 score

|  |  |
| --- | --- |
|  |  |
|  | |

4.3. Figure Results

In the following pictures you can see the original and predicted labels:

A collage of images of a street

Description automatically generated

A collage of a road

Description automatically generated

4.4. Figure Original and Predicted labels

# 5. Conclusion

In conclusion, the endeavor to train a YOLOv8 network for car detection has provided valuable insights into the complexities of the task. Despite the initial progress made, it is evident that achieving a higher level of accuracy requires further iterations of training. The iterative nature of deep learning models, especially in complex domains like object detection, emphasizes the importance of continuous refinement. Additional training sessions with carefully curated datasets, parameter tuning, and perhaps exploring advanced architectures could contribute to the enhancement of the model's precision. Acknowledging the need for ongoing improvement, this project lays the foundation for future developments in the pursuit of an optimized and robust car detection system using YOLOv8.

1. **References**

[1] Dataset website: <https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d>