21ECE372T-DEEP LEARNING FOR DATA ANALYTICS

Report

Submitted by

RA2211053040023 – SHARVESH S

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

FACULTY OF ENGINEERING AND

TECHNOLOGY SRM INSTITUTE OF SCIENCE

AND TECHNOLOGY

Vadapalani Campus, Chennai –600026

APRIL-2025

Real-Time Object Detection Using YOLOv5 and OpenCV

Abstract: This project presents the implementation of a real-time object detection system using YOLOv5 (You Only Look Once version 5) and OpenCV. Instead of using a live webcam feed, we use a prerecorded video to simulate real-time detection. YOLOv5 is a state-of-the-art deep learning algorithm optimized for speed and accuracy in object detection tasks. This project demonstrates how pretrained YOLOv5 models can be integrated with OpenCV to detect and label objects in each frame of a video, making it suitable for surveillance, traffic monitoring, and intelligent automation.

Introduction

Motion detection is a fundamental task in the field of computer vision and surveillance systems. It involves identifying movement within a scene captured by a video camera. This project utilizes Python and OpenCV to detect motion in real-time from a webcam. The output is visualized using bounding boxes around moving objects and textual alerts indicating when motion is detected.

Objectives

- To implement a deep learning-based object detection system using YOLOv5.
- To process video input frame-by-frame and detect multiple objects in real-time.
- To visualize detections using bounding boxes and labels.
- To understand the integration of pretrained YOLO models with OpenCV.

Tools and Technologies Used

- **Programming Language:** Python
- **Libraries**: OpenCV, Torch, PyTorch Hub, Matplotlib
- **Deep Learning Model:** YOLOv5s (pretrained)
- **Platform:** Google Colab / Jupyter Notebook
- **Hardware:** GPU (Colab Runtime)

Working Principle

The YOLOv5-based object detection system operates using the following steps:

- Load the YOLOv5 pretrained model via PyTorch Hub.
- Load a video file as input using OpenCV.
- For each frame: a. Resize and preprocess the frame. b. Perform inference using YOLOv5. c. Parse the detections and draw bounding boxes with class labels.
- Display the annotated frame or save the output video.

Code Overview

1. Installing Dependencies

Before you can use YOLOv5, you need to install the required libraries:

```
!pip install torch torchvision

!pip install opencv-python-headless

!pip install yolov5
```

What this does:

- torch and torchvision: Required for running PyTorch-based models.
- opencv-python-headless: OpenCV for reading and processing image/video without GUI support (needed in Colab).
- yolov5: Installs the Ultralytics YOLOv5 package with model definitions and utility functions.

2. Importing Libraries

```
import torch

import cv2

import numpy as np

from matplotlib import pyplot as plt

from google.colab import files
```

Explanation:

- torch: For running the YOLOv5 model.
- cv2: OpenCV for handling images and videos.
- numpy: Image/frame conversion.
- matplotlib.pyplot: Displaying images inside Colab (since cv2.imshow() doesn't work).
- google.colab.files: Upload image or video files from your computer to Colab.

3. Loading the YOLOv5 Model

```
model = torch.hub.load('ultralytics/yolov5:v7.0', 'yolov5s')
```

Details:

- Loads the **YOLOv5 small (s)** model from Ultralytics GitHub repository.
- You can change 'yolov5s' to:
  - 'yolov5m' → medium
  - 'yolov5l' → large
  - 'yolov5x' → extra-large

This model comes pretrained on **COCO dataset** (80 classes like person, car, dog, etc.).

4. Uploading a File (Image or Video)

```
uploaded = files.upload()

video_path = list(uploaded.keys())[0]
```

What's happening:

- Prompts user to upload a file.
- uploaded is a dictionary of {filename: filecontent}
- The uploaded file path is extracted and stored in video_path.

5. Reading and Processing Video Frame-by-Frame

```
cap = cv2.VideoCapture(video_path)

while cap.isOpened():

    ret, frame = cap.read()

    if not ret:

        break
```

Explanation:

- cv2.VideoCapture() opens the uploaded video.
- Inside the loop:
  - cap.read() reads each frame from the video.
  - ret is a boolean that indicates if the frame was read successfully.
  - frame is the image data.

6. Preprocessing the Frame and Running Detection

```
frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

results = model(frame_rgb)
```

🔧 **Details:**

- YOLOv5 expects **RGB** format, but OpenCV reads in **BGR** format.
- cv2.cvtColor(frame, cv2.COLOR_BGR2RGB) converts the color space.
- results = model(frame_rgb) passes the frame into the YOLOv5 model and returns the detection results.

7. Rendering and Visualizing the Results

```
results.render()

frame_bgr = cv2.cvtColor(frame_rgb, cv2.COLOR_RGB2BGR)

plt.imshow(cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB))

plt.axis('off')

plt.show()
```

What's happening here:

- results.render() overlays the bounding boxes and labels **in-place** on frame_rgb.
- Converts back to BGR → needed for accurate visualization with cv2 functions.
- matplotlib is used to show the annotated frame:
  - imshow() displays the frame.
  - axis('off') removes ticks and labels for a clean view.

8. Release the Video Resource

```
cap.release()
```

Why:

- Frees up the video file resource after reading is done.
- Good practice to avoid memory leaks.

9. (Optional) Suppressing Warnings

You encountered this warning:

FutureWarning: torch.cuda.amp.autocast(...) is deprecated...

To suppress it:

import warnings

warnings.filterwarnings("ignore", category=UserWarning, module="torch")

Or just ignore it — it's from **internal YOLOv5 code**, not your implementation.

Summary of Workflow

1. Install all required packages.
2. Load YOLOv5 model from Ultralytics using torch.hub.
3. Upload a video file using files.upload().
4. Read and process the video frame-by-frame using OpenCV.
5. Detect objects using YOLOv5.
6. Visualize detections on each frame using matplotlib.
7. Optionally suppress warnings and clean up resources.

Output (Description)

- The system reads a video file frame-by-frame.
- On each frame, the model detects objects like: person, car, dog, bicycle, etc.
- It draws bounding boxes with confidence scores and labels.
- Detected frames are displayed sequentially using matplotlib.

Output and Results: The system processes each frame of the input video and overlays bounding boxes around detected objects. Each box includes a class label (e.g., person, car, dog) and a confidence score. The processed video is saved as output_yolo.avi and can be visualized in any media player.

Applications:

- Traffic surveillance and vehicle counting
- Pedestrian detection in smart cities
- Industrial automation and robotics
- Wildlife observation and conservation

Limitations:

- Requires sufficient GPU resources for smooth performance
- Accuracy depends on lighting and video quality
- May struggle with small or occluded objects

Future Enhancements:

- Implement real-time webcam support
- Add multi-object tracking (Deep SORT integration)
- Train YOLOv5 on custom datasets for specific applications

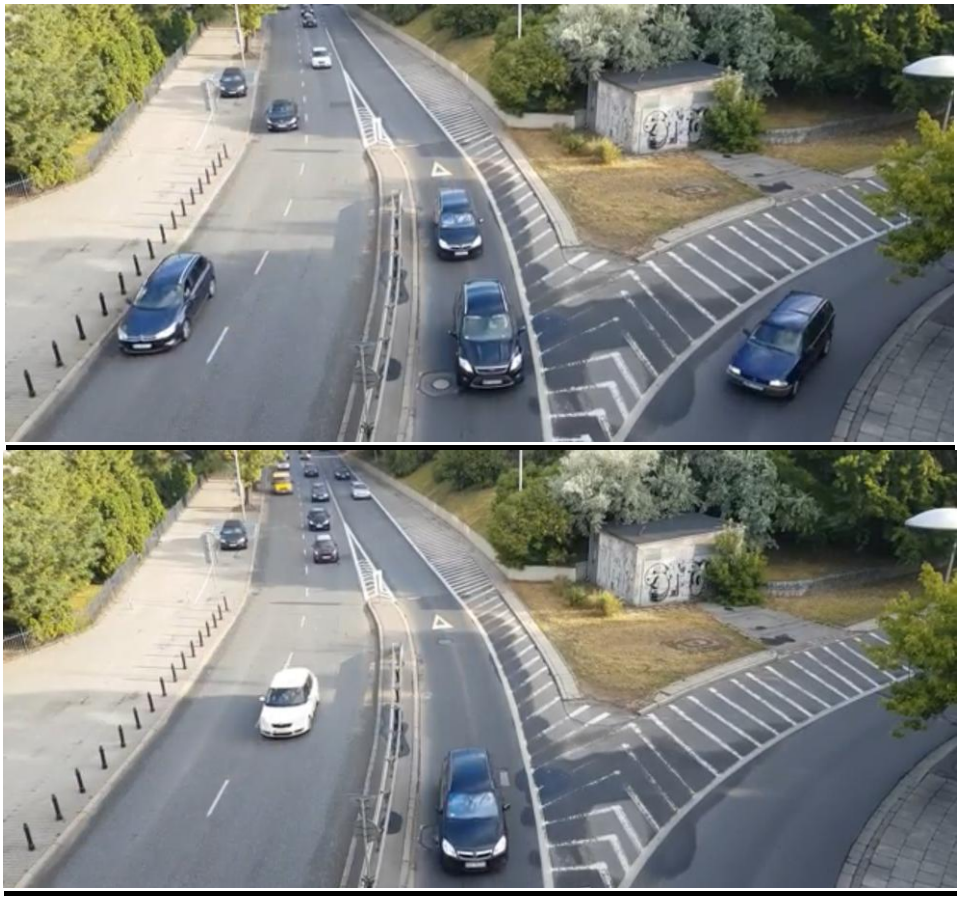
Conclusion: This project showcases the practical application of YOLOv5 for real-time object detection on video streams. By leveraging PyTorch, pretrained models, and OpenCV, we achieved an efficient detection pipeline that can be used for a wide range of AI-driven vision tasks. It builds a foundation for further exploration into intelligent video analytics and smart surveillance systems.

References

- OpenCV Documentation: https://docs.opencv.org
- Python Official Docs: https://docs.python.org
- StackOverflow and GitHub for community support and code examples

INPUT:

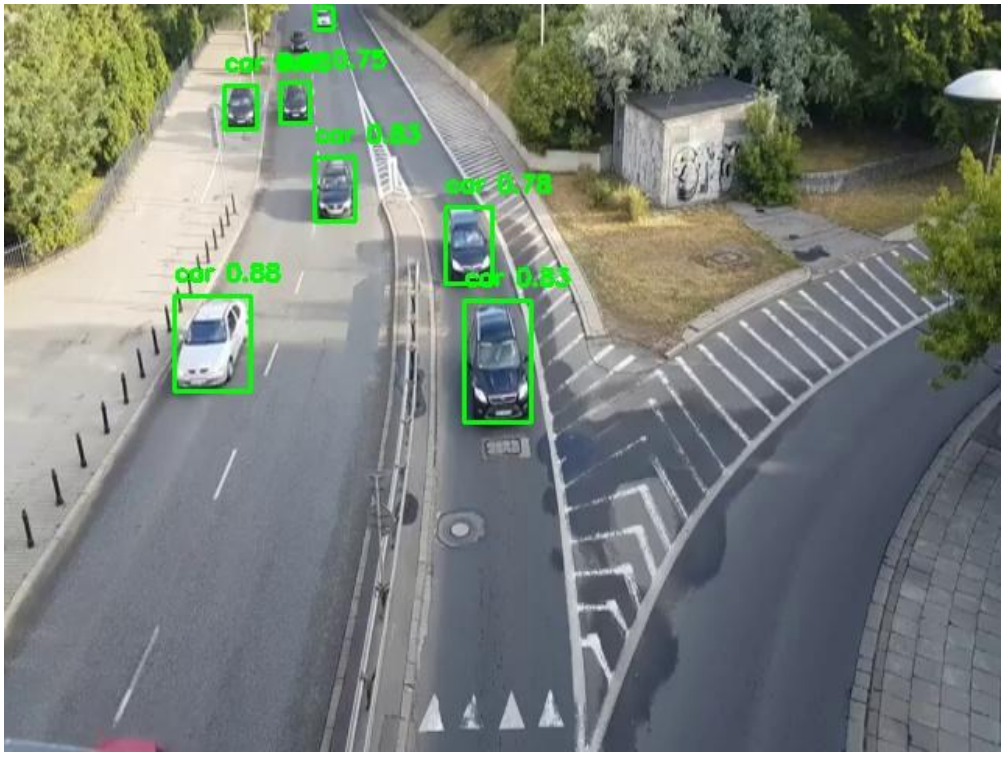
1. TRAFFIC CAM FOOTAGE FROM A NEIGHBOURHOOD

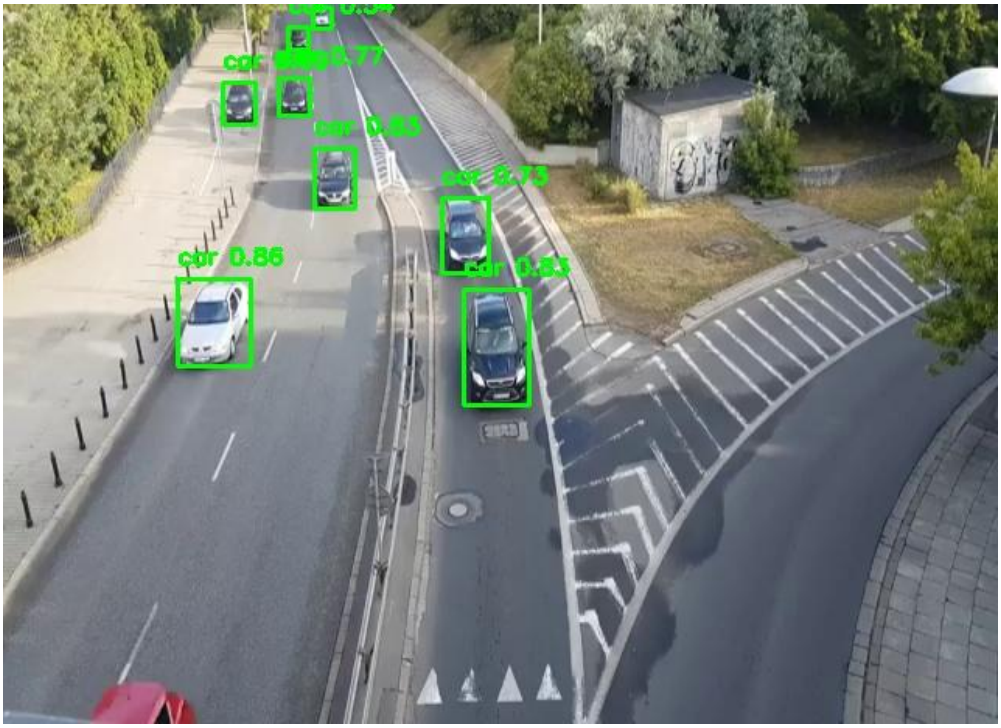


2. CAMERA FOOTAGE FROM AN AIRPORT

OUTPUT:

1. TRAFFIC CAM FOOTAGE FROM A NEIGHBOURHOOD





2. CAMERA FOOTAGE FROM AN AIRPORT