Video Processing and Annotation Report

B21CS009,B21CS083

May 4, 2024

1 Introduction

This report outlines the process of video processing, object annotation using a subset of the **BDD100K** dataset and **open source videos** . The primary goal is to demonstrate a pipeline for analyzing and annotating video content, followed by stabilizing the annotated video for improved viewing and analysis. We used **OpenCV** (Open Source Computer Vision Library): OpenCV is a popular library for computer vision tasks. In this code, OpenCV is used for tasks such as video capture, frame extraction, object annotation (drawing bounding boxes), and video stabilization (using optical flow) and another method we utilization of the **YOLO (You Only Look Once)** model for object detection within video content. The YOLO model is renowned for its real-time object detection capabilities, making it a popular choice for various computer vision tasks.

2 *Using OpenCV*

2.1 Dataset Information

The **BDD100K** dataset is a diverse driving video dataset(has >1000 video samples) that includes various weather conditions, times of day, and driving scenarios through *survilleance cams.* For this project, a subset of the BDD100K dataset was utilized, focusing on a specific video segment for analysis and annotation.

2.2 Video Conversion

The initial step involved converting a selected video segment from MOV format to MP4 format using *FFmpeg* as the project required. This conversion process ensured compatibility and ease of processing for subsequent steps.

Figure 1: Frame 1035 extracted from 0225f53-67614580.mp4

2.3 Video Metadata Extraction

Upon conversion, metadata such as the *number of frames*, *frame dimensions* (height and width), and *frames per second (FPS)* were extracted from the video. This metadata provided essential information for frame extraction and video stabilization.

2.4 Frame Extraction and Display

Frames were extracted from the video using OpenCV, and a subset of these frames was displayed in a grid format using Matplotlib. This step allowed for visual inspection and analysis of different moments within the video. See above, to see the is a extracted frame.

2.5 Object Annotation

Object annotations were added to specific frames within the video using information sourced from a CSV file (`mot_labels.csv`). The CSV file contained details about objects[Car,Truck, Pedestrain, Other Vechile, Rider, Bicycle, Other person, Trailer, Motorcycle, Bus] detected in the video frames, including bounding box coordinates and object categories. Each ***object*** is given a **different colour box**. Like *red* for *cars*, *green* for *Bicycle*, *dark blue* for *Rider*,etc. Also boxes coordinates can be saved in an array with labelled boxes. One **box coordinates** ,**height** and **width** are showed in the code.

2.6 Annotated Video Creation

Annotated frames were compiled to create a new video(**final.mp4)** with object annotations. The annotations included colored bounding boxes around objects based on their categories, enhancing the visual representation of detected objects within the video. **Link to final video** :

Figure 2: Multiple Frames from the video 0225f53-67614580.mp4

Figure 3: Objects detected within Boxes

https://drive.google.com/file/d/1B4wr8SyUvbwu-8OkQeawgEEDnT7U7Zwc/view?usp=sharingFinal Video

2.7 Video Stabilization

The final step involved stabilizing the annotated video using the *Lucas-Kanade optical flow method* implemented with OpenCV. This process calculated optical flow between consecutive frames, estimated affine transformations, and applied these transformations to stabilize the video.

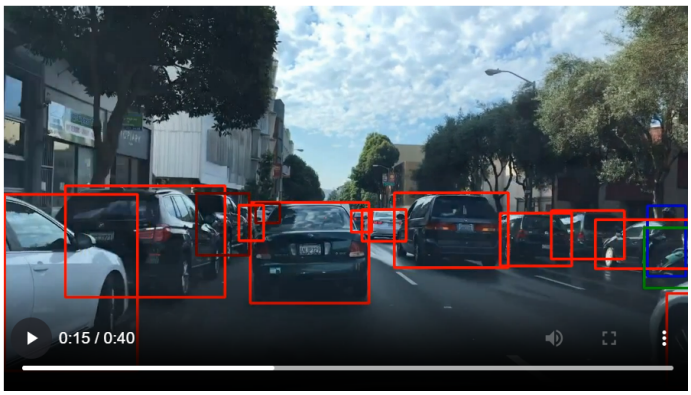

Figure 4: Different Objects detected within the final video(Only a frame is shown for now)

2.8 Output and Conclusion

The output of the process was a video with annotations, suitable for further analysis, presentations, or reports. This pipeline was an effective approach to **video processing** and **object annotation**.

3 *Using YOLO*

The YOLO (You Only Look Once) model revolutionized object detection by offering real-time detection capabilities. Unlike traditional methods that perform multiple passes over an image, YOLO processes the entire image in a *single pass*, providing *fast* and *accurate object detection*. In YOLO, the image is divided into a *grid*, and each grid cell predicts *bounding boxes and class probabilities*. This enables YOLO to detect multiple objects with a single algorithm run. YOLO's raw output contains *multiple bounding boxes* for the same object. These boxes may vary in shape and size.

To select the best bounding box for a given object, YOLO employs a *Non-maximum suppression (NMS) algorithm*. NMS eliminates redundant bounding boxes and retains only those with the highest confidence level.

3.1 Implementation

We implemented the **YOLO** model using the *inference-gpu library*. The following steps were taken:

- The *YOLO* model was initialized using the ***YOLOWorld class*** from the inference.models module.
- The classes for object detection were set using a predefined list, all the necessary objects are stored in the classes.

Figure 5: Multiple frames from the video used for **YOLO**

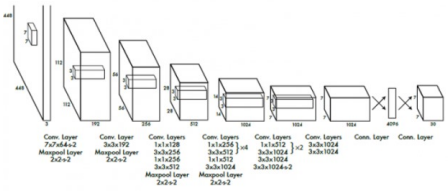

Figure 6: YOLO Object Detection Algorithm

- Process input video frames, perform object detection, annotate detected objects, and save the annotated video.

3.2 Video Metadata Extraction

Upon conversion, metadata such as the *number of frames, frame dimensions* (height and width), and *frames per second* (FPS) were extracted from the video.
Link for **Input Video** :
https://drive.google.com/file/d/1Lvl5d98-JWO3INmRb4NXKx2GUgDEr6fd/view?usp=sharing

3.3 Object Annotation

To enhance the visual representation of detected objects in the video, we implemented object annotation using *bounding boxes* and *labels*. Bounding boxes were drawn around detected objects using the ***BoundingBoxAnnotator*** with a thickness of 2. These bounding boxes help to visually delineate the detected objects within each frame of the video. Object labels were added to the annotated frames using the ***LabelAnnotator*** with a text thickness of 2, a text scale of 1, and black text color. These labels provide textual information about the class of each detected object. By combining bounding boxes and object labels, we effectively highlighted and identified each detected object within the video frames.

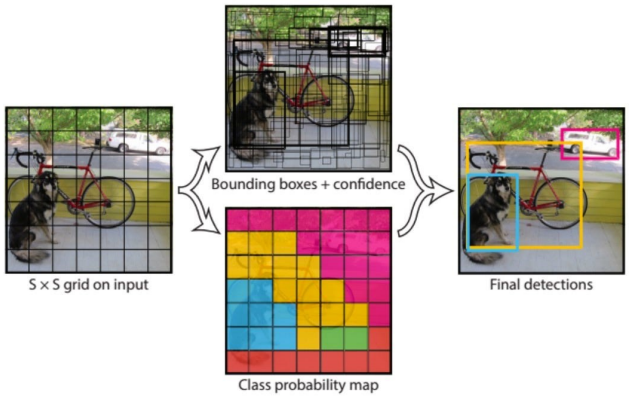

Figure 7: Our system models detection as a regression problem. It divides the image into an S X S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S X S X (B $*$ 5 + C) tensor.

Figure 8: Multiple frames from output video

3.4 Output & Conclusion

The YOLO model effectively detected and annotated objects in the input video. The output video shows bounding boxes around detected objects, with labels indicating the type of object. This implementation demonstrates the effectiveness of the YOLO model for real-time object detection in videos. Link for **Output Video**:

https://drive.google.com/file/d/1etfCMfpOtv0iZw4-RoAdv6fL9EjYXlCJ/view?pli=1Output Video:

4 Useful Links

- https://ffmpeg.org/ffmpeg.htmlhttps://ffmpeg.org/ffmpeg.html
- https://learnopencv.com/deep-learning-with-opencvs-dnn-module-a-definitive-guide/OpenCV DNN Module and Deep Learning (A Definitive guide) (learnopencv.com)
- https://chat.openai.com/
- https://opencv.org/OpenCV - Open Computer Vision Library
- https://yolov8.org/yolov8-label-format/YOLOv8 Label Format: A Step-by-Step Guide - YOLOv8
- https://medium.com/@connect.vin/yat-an-open-source-data-annotation-tool-for-yolo-8bb75bce1767YAT - An open-source data annotation tool for YOLO — by Vinay — Medium