Shubham Gupta

Report – Final Project

Abstract

This project features a real-time object detection and tracking system for a single class (Xbox controller) based on a proprietary CNN (convolutional neural network). A dataset of 500+ images was collected in YOLO format, wherein the convolutional neural network (CNN) preprocesses and normalizes the inputs. Furthermore, we trained a lightweight CNN with four convolutional blocks to predict class confidence along with bounding box coordinates. Our model achieved an average intersection-over-union (IoU) of 0.75 on validation data while using a standard laptop CPU, processing 8-12 frames per second. CSRT tracker from OpenCV was then integrated to enhance temporal consistency by preserving object identity across detection frames. Our demo successfully tracked controllers in complex lighting conditions and simple backgrounds, proving that minimal architecture yielding real-time performance on basic hardware is possible.

1. Introduction

Robotics, augmented reality, and human-computer interaction features depend on detecting and tracking objects in real time. Demand is high for state-of-the-art detectors (e.g., YOLOv5, SSD), but most are too cumbersome to be run on a CPU in restricted environments. The goal of this project is to build an entire pipeline for detecting a single object class (Xbox controller) with minimal resource reliance, running off webcam footage without a GPU. Further, we want to achieve this at interactive frame rates.

To solve challenges like developing a dataset of a niche class as well as crafting a lightweight model architecture, we design anew with accuracy and speed in mind. Restricting the class to a single one helps attention data collection and model training efforts. Normalized bounding box annotation eases mapping and is simple enough to directly adopt in Equal OPENCV. BOXCAPTURE is a part of openCV that captures live feed from a webcam, sports bounding box drawing capabilities, and track integration that makes it practical for this use case.

This document contains our implementation steps, training outcomes, a limit discussion, and forthcoming actions.

2. Methods

2.1 Dataset Preparation

We obtained and organized a collection of 550 images depicting Xbox controllers in various states from Roboflow Universe. For training purposes, we extracted 400 images, while keeping 150 images for validation. All images were resized to a uniform resolution of 416×416 pixels, and annotations transformed into the normalized YOLO format (class_id x_center y_center width height). The only identified class, xbox_controller, was substituted with label 0.

2.2 Data Loader

Our load_data generator (in src/utils.py) iterates over sorted image paths, reads each with OpenCV, converts BGR→RGB, resizes to 416×416, and normalizes to [0,1]. We parse the corresponding .txt file if present, convert each normalized center-width-height to absolute corner coordinates ([x1,y1,x2,y2]), and yield (image, boxes, class_ids) tuples.

2.3 Model Architecture & Training

A CNN architecture was defined in TensorFlow/Keras (in src/train.py) with 4 conv blocks consisting of a triplet of (Conv2D→ReLU→MaxPool2D) with 16 and 64 filters, followed by a Conv2D(128) and GlobalAveragePooling. The two heads in the network are:

- bbox: Dense(4, sigmoid) outputs normalized bounding-box coordinates in [0,1]sr
- cls: Dense(1, sigmoid) outputs class confidence.

For box regression, we used Mean Squared Error (MSE) and for classification, we used Binary Cross-Entropy (BCE), applying uniform loss weights for both. We trained for 10 epochs on a CPU laptop with batch size 8. Iteration indefinite handler in our data_gen ensured that the generator would restart when the data ran out.

2.4 Real-Time Inference

In src/detect.py, we implemented webcam capture using cv2.VideoCapture(0), loaded trained weights (models/ckpt_last.weights.h5), and performed inference after resizing the inputs to 416×416. Subsequently, we remapped the predictions to the initial frame dimensions. An avalanche of predictions with a confidence score lower than 0.5 was filtered out to minimize erroneous positives. The bounding boxes alongside the confidence indications were superimposed in green on the live video feed.

2.5 Tracking Integration

For sustained object identity across detections that are not frequent, we used a basic frame-skipping method. Redetection happened every tenth frame. Existing bounding boxes received updates from the OpenCV CSRT tracker during frames between the redetections. Upon each detected box, trackers underwent initialization. Per frame trackers underwent updating. This action produced trajectories with greater smoothness as well as it reduced detection pauses.

3. Results

3.1 Detection Accuracy

With a set of 150 images for validation, the model showed

- The mean IoU was 0.75.

- Measured at 0.5 confidence, reached 0.82.
- Recall was 0.78.

Visual study of the results showed consistent location identification when objects were partly hidden or under changing light. Scenarios where the model did not work included very sharp angles and blur caused by movement.

3.2 Inference Speed

I measured end-to-end performance on an Intel i5-8265U CPU:

- **Detection-only:** 12.4 FPS
- **Detection + Tracking:** 8.6 FPS

This simple architecture and small input size enabled interactive performance, suitable for non-GPU edge devices.

4. Discussion and Conclusion

This project demonstrates that a small bespoke CNN can detect and track a single object class in real time on CPU-only hardware. By addressing a single class, we reduced annotation overhead and model complexity. The incorporation of a light tracker improved temporal consistency without sacrificing throughput.

Limitations: Generality of the model is limited to Xbox controllers; new classes would entail retraining or architectural changes. Performance is hindered by severe occlusion, rapid movement, or cluttered backgrounds. Additionally, the static input size (416×416) introduces scaling artifacts when resizing images of different aspect ratios.

Future Work: Extension to multiple classes using a multitask head, experimentation with more efficient backbones (e.g., MobileNetV3), and deployment on embedded devices (e.g., Raspberry Pi) using hardware accelerators. Incorporating online learning to tune the model to new environments could enhance robustness.

References

[1] Tzutalin, "LabelImg: Image Annotation Tool," GitHub, 2016.

[2] Redmon, J., Farhadi, A., "YOLOv3: An Incremental Improvement," arXiv:1804.02767, 2018.

[3] Bradski, G., "OpenCV: Open Source Computer Vision Library," Dr. Dobb’s Journal, 2000–2008.