Project Title	Vehicle stop detector
Team Name	Team1
Team members	정찬희 오우진

Motivation and Objectives

1. **Motivation**

Recently, there has been a surge in unfortunate traffic accidents in South Korea, prompting consecutive revisions to road traffic laws to ensure pedestrian safety. Among numerous changes, the most notable one requires drivers to momentarily stop under various road conditions, such as at intersections and school zones. However, public infrastructure is not yet adequately prepared to enforce these new regulations.

One limitation lies in the currently installed speed cameras' inability to detect whether individual vehicles come to a stop. These cameras measure average speed by tracking intervals between two separate sensor points, making it challenging to distinguish cars from pedestrians or other objects. Moreover, it is impractical to have police officers stationed at every relevant location. To address these concerns, we propose developing a CCTV program capable of automatically determining whether vehicles stop as required and collecting information on law violators. This solution would encourage drivers to comply with traffic laws even in the absence of police officers and enable governing agencies to manage the legal system effectively with minimal manpower.

2. **Objectives**

- Design a model capable of fast inference on live video feeds
- Finetune the model using diverse datasets to enhance accuracy
- Detect and track individual vehicles, determining stationary vehicle
- Extract license plate number from captured image
- Employ threading or frame skipping techniques to optimize processing speed

Technical contributions (Skill, Knowledge, Novelty)

1. **Baseline**

We initiated our project by utilizing an existing platform that already included some features needed for vehicle stop detection. This project, named "Track and Count People with Jetson Nano[1]," is devised to track individuals and determine whether they have crossed a designated centerline. The project uses 'SSD_Mobilenet_v1 TRT(FP16)' model for person detection, with an input video resolution of 300x300. They applied threading to accelerate overall process by designating separate threads, one for inference and one for people counting and image display. It simply compares whether the center of bbox passes the centerline so the algorithm is relatively simple.

2. **Skill/Knowledge**

2.1 Dataset

We searched on the internet for datasets that contain vehicles because the model we want to build only needs to detect cars. In order to make the model more robust, we tried to combine various datasets with different angles and different types of cars included[4]. The problem is that it was not easy to find a dataset that exactly fits our purpose. Therefore, we downloaded datasets created to detect not only cars but also other objects together, or datasets that detect by classifying cars, and then, we modified the annotation file to suit our purpose (deleting labeling for objects other than cars, and unified cars into one label if they are labeled differently by type). In this way, it was possible to build a decent sized dataset.

2.2 Model

To enable real-time inference on the Jetson Nano, we require a lightweight model that still maintains sufficient accuracy. Our initial attempt employed 'Yolov4-tiny-288(FP16)'[2] converted into TensorRT, which demonstrated the highest FPS. However, we need more accurate model, prompting our design of the YoloV4-tiny-320(FP16), increasing the input size of the first layer to 320. Consequently, this amplified the mAP in comparison to the 288 model with only a slight reduction in FPS. Although the Yolov8 model was considered, its significantly lower FPS rendered it unsuitable for Jetson Nano implementation.

We fine-tuned our model, which was pre-trained with the MS COCO dataset, by undergoing 30,000 iterations to increase vehicle detection accuracy. Setting a singular class, 'vehicle,' allowed us to prioritize detecting vehicle stoppage rather than identifying specific types of vehicles.

3. **Novelty**

Our overall process flow consists of the following steps: First, the system acquires input video, detects vehicles utilizing Yolov4-tiny, and obtains bboxes. Next, it tracks bboxes of identical vehicles, implements an algorithm for stationary detection, and finally visualizes the information using OpenCV.

3.1 Kalman Filter Tracking

We employed the Kalman filter for object tracking. Utilizing prior distributions, it predicts the subsequent location of the bounding box. The Kalman filter enables the prediction of the next bbox location, which is then compared with the actual bbox. If the Intersection over Union(IoU) value surpasses a predetermined threshold, the algorithm infers that it is same object.

3.2 Stop detecting algorithm

To determine if a vehicle has stopped, we compare the center of the bounding box in the previous and current frames and calculate the distance. If the distance is smaller than the threshold value, we classify the vehicle as not moving. If this occurs for over 15 frames, we consider the vehicle as having stopped, and the system outlines the stopped car with a green bounding box. If the car moves before reaching 15 frames, we reset the count to zero. Since we can track vehicles independently, each vehicle's frame count is handled separately.

We wanted our algorithm to detect only for vehicles exclusively in the right turn lane. Initially, we attempted to determine the vehicle's intention to turn based on the bbox location. However, the bbox trajectory for right turns varies depending on the video's angle and position. Another method entails training the model to detect lanes and discern whether the identified lane is for right turns. However, this approach might be inconsistent, depending on the vehicle's movement direction in the video. For instance, if the car moves upward, the right turn lane is the rightmost lane, whereas if the car moves downward, the lane is on the leftmost side. As a result, we made the assumption that our CCTV focuses solely on the right turn lane.

3.3 Capturing vehicle & License plate number recognition

We initially captured images of all vehicles and store them. To obtain better quality and larger image sizes, we incorporated code to save the largest image for the same vehicle. Once a vehicle is confirmed to have stopped, we delete its corresponding image. Consequently, we can only obtain images of vehicles that did not stop.

The ultimate goal of the project is to issue fines on vehicles that violate the rules. To achieve this, we need to capture images of non-stopping cars and identify their license plate numbers for issuing fines. Our initial approach involved applying deep learning-based number recognition techniques to extract license plate information from captured images. However, the poor image quality with low resolution made this approach impossible to implement. To improve image quality, we would need to add a super-resolution process to transform images, but such a process would add too much computational load for the Jetson Nano to handle.

Thus, instead of using a deep learning-based plate number recognition model, we opted for a more traditional image processing method[3]. We first find an image's contour, using OpenCV to determine the location of numbers. Next, with an image of the plate number, we use Tesseract to detect numbers and Korean characters in the image, returning the results.

There are limitations to our method, as it can only be applied to videos with a resolution higher than 1080x1080. Higher-quality video would yield more precise numbers, but this would extend the image processing time and subsequently decrease the FPS of whole process.

3.4 Frame skipping

The baseline project uses threading to enhance performance. The child thread begins with an initial image inference and then transfers the image and bounding box information to the main thread while performing inference for the subsequent frame. We attempted to apply the same approach to our model.

However, we faced some issues with threading. The time required for inference and processing the algorithm varied. Due to OpenCV not supporting CUDA, resulting in longer resizing and display times for images. Consequently, some images were skipped during processing, even after their inferences had been completed.

To address this problem, we added a ‘time.sleep’ code that pauses the process for a few milliseconds to set a constant time for the child thread. Despite improving thread synchronization, this slowed down the overall performance of the system.

Ultimately, we decided to skip the inference for some frames. Since our main goal is to detect whether a car has stopped, nearby frames are relatively similar and skipping some frames does not significantly impact overall performance. By reusing information from the bounding box, we were able to speed up the process.

4. Feedback from presentation

There was a concern regarding the loss of accuracy in determining whether a vehicle has stopped when applying frame skipping. However, it turns out that incorporating frame skipping can actually improve both speed and accuracy. Let's discuss how this enhancement in accuracy has been achieved.

A fundamental issue our model faced in determining vehicle stops was moving car could appear to be stationary depending on the angle of the shot. Although adjusting the shooting angle adequately could prevent this issue, we sought a technical solution.

By skipping some frames, our model only performs inference on one frame per 5 frames. This approach helps resolve the problem for the following reason: Our stop determination algorithm is designed to consider a vehicle as stopped if its bbox remains in the same location for a specific duration. Therefore, a vehicle moving at an angle where the bbox's location does not change significantly may be incorrectly identified as stopped.

Implementing frame skipping reduces the likelihood of such misjudgment, as the gap between the current bbox and previous bbox is larger than that in the model without frame skipping.

5. Result & Future work

5.1 Result

In our final implementation, we employed an input video size of 600x600 and a frame rate of 30 FPS, utilizing the Yolov4-tiny-320 model converted to TensorRT. By employing frame skipping (performing inference every 5 frames), we were able to achieve an FPS of 25.7, which provides sufficient speed for practical applications in real-world environments. Without frame skipping, the FPS decreased to 14.9, indicating a significant performance improvement when using frame skipping. Furthermore, we found that an increase in input video resolution led to a reduction in FPS. By maintaining a resolution of 600x600, we successfully utilized this model for real-time detection of stationary vehicles.

5.2 Future work

First, we could enhance the inference speed by applying OpenCV supported with CUDA. By applying this, the time taken for image processing would be reduced. Second, we can give input a video with higher resolution to get more accurate plate number. Also, we can get a step further by adding a feature that decides whether the vehicle can turn right without stopping.

To address the problem of difficulty in determining stop when filming from the front angle of the car, it is possible to consider a change in the size of the bbox instead of simply using a change in the location of the bbox (using the fact that it grows closer and becomes smaller when it moves away). Additionally, one of the improvement directions is to combine a model that detects a vehicle that violates the law and a code that extracts a vehicle's license plate using the image stored there.

Discussion and Conclusion

- We learned how to implement custom object detection model on Jetson Nano
- Optimization technique accelerates model inference speed
- Threading can help accelerate overall process

Citation

[1] https://github.com/JardinRyu/Jetson_Nano_People_Counting

[2] https://github.com/jkjung-avt/tensorrt_demos

[3] https://maxtime1004.tistory.com/38

[4] Datasets used

universe.roboflow.com/juliana-at/labels-abzqw/dataset/4

universe.roboflow.com/roboflow-100/vehicles-q0x2v/dataset/2

kaggle.com/datasets/ryankraus/traffic-camera-object-detection

universe.roboflow.com/yoohyojeong/car_1/dataset/1

universe.roboflow.com/school-rrpnj/-tnvrs/dataset/2

universe.roboflow.com/machin-learning-class-41/car-images-aotbt/dataset/1

universe.roboflow.com/smart-helmet/vehicle-temp/dataset/1

universe.roboflow.com/projects-67ens/carcounting-doet1/dataset/3

universe.roboflow.com/exjobb-dq06p/vehicles-k83q3/dataset/2

universe.roboflow.com/test-nkacj/testv1-mcc5w/dataset/1