Traffic Signal Improvements Using YOLOv8

How can traffic at Center Drive and Archer Road in Gainesville, Florida be improved with computer vision?

Harrison Stark

Contents

1 Introduction

Problems that arise within transportation almost invariably require an iterative solution. Whether it be the introduction of new technologies surrounding magnetic levitation (Maglev) trains or even the implementation of sensors to detect cars at an intersection, the solutions must be tweaked and edited before a final solution, which may not work as expected, can be implemented ("Maglev train", n.d.). With these changes to the transportation infrastructure comes significant cost. From a few hundred dollars for a sensor at a traffic light to tens of thousands to millions of dollars per iteration for a Maglev train, change does not come cheap ("Traffic Signal Components", n.d.).

The intersection of Center Drive and Archer Road in Gainesville, Florida, is a well-known intersection to many. Pedestrians, drivers of cars, trucks, motorcycles, bicycles, buses, scooters, skateboards, and essentially anyone in a small radius of this intersection knows the struggles associated with this intersection. The lights are not long enough at any of the four corners for the pedestrians or for motor vehicles, and all around is quite dangerous. Timing of this light is seemingly impossible as traffic is coming from all directions at nearly all times of day. The thought of working on this intersection, much less improving it, is seemingly impossible. Sure, one may implement massive changes to the infrastructure, add traffic guards, or even alter the timing of the lights to be biased towards either the vehicles or the pedestrians, but all of these changes come at a large cost.

With new state-of-the-art (SOTA) developments in computer vision, a sect of artificial intelligence dealing with processing images and videos, a low-cost, efficient, and effective implementation of cameras and algorithms to fix the issues that this intersection faces is right around the corner. Figures 1 and 2 shows the intersection of Center Drive and Archer Road in Gainesville, Florida. The goal of this paper will be to explore the implementation of computer vision technology into this intersection and offer a potential low-cost solution that could potentially be expanded to more intersections.

Figure 1: This is an adapted image from Google Maps that shows the intersection of Center Drive and Archer Road in Gainesville, Florida from the view of one on Center Drive facing south (“Google Maps”, n.d.).

Figure 2: This is an adapted image from Google Maps that shows the intersection of Center Drive and Archer Road in Gainesville, Florida from the view of one on Archer Road facing east (“Google Maps”, n.d.).

2 Background Information

Computer vision ”is a field of artificial intelligence that enables computers and systems to derive meaningful information from digital images” (“What is Computer Vision?”, n.d.).

2.1 Classification Models

A classification model for computer vision will take an image and determine which class it belongs to based on training data. For example, an accurate computer vision algorithm for classification of a fruit between oranges and apples will take in an image of, for example, an orange, and return ”orange.” This of course is the human-readable version of it. In the computer, the image will become a tensor, a multidimensional vector that represents each pixel value in the image with dimensions for color, and it will output a probability vector showing how likely it is that based on the training set, the input is in fact an orange. So given an image that the model is quite sure is an orange, it will output something along the lines of .001, .999, where the .001 is the probability that the image is an apple, and the .999 represents the probability that the image is an orange. This example can be seen in Figure 3.

In the computer vision models that this paper will be exploring, specifically those that produce a bounding box around objects, there is a slightly different method implemented that use similar classification.

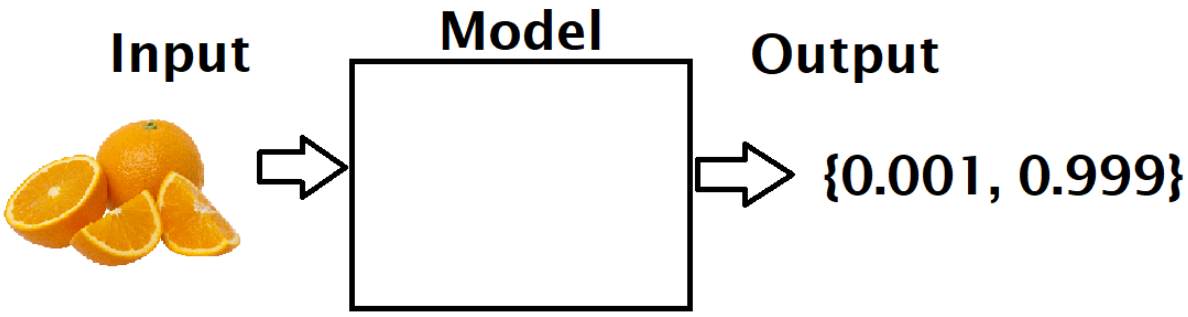

Figure 3: This is an image that shows an example input of an image of an orange and an example output of a traditional classifier model with 0.001 as the probability of the image being an apple and 0.999 as the probability of the image being an orange ("File:Orange-Fruit-Pieces.jpg", n.d.).

2.2 Bounding Box Models

This paper will focus specifically on a newer type of computer vision model that has been in development over the past few years, and that is a model that implements bounding boxes. An impressive SOTA model for this is called the YOLO (You Only Look Once) algorithm ("YOLO: A Brief History", n.d.). This type of algorithm implements many complicated techniques, but in essence they split up an image into smaller sub-images, and the model is trained to detect objects in those sub-images and generate a bounding box around that image with the predicted image type ("YOLO: Algorithm for Object Detection Explained [+Examples]", n.d.). For example, if an image was input into a well-developed model that detected apples and oranges that has an orange in the upper left corner and an apple in the bottom right corner, it would draw a bounding box around each fruit and give the probability of each sub-image being of each type. This example can be seen in Figure 4.

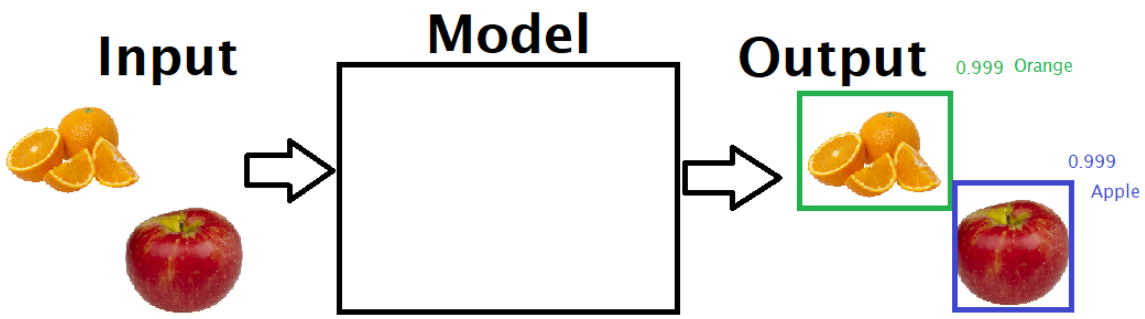

Figure 4: This is an image that shows an example input of an image of an orange in the upper left and an apple in the bottom right and an example output of a machine learning model that outputs bounding boxes with expected probabilities, showing that there is a 0.999 probability of an orange being in the upper left and a 0.999 probability of an apple being in the bottom right ("File:Honeycrisp-Apple.jpg", n.d.; "File:Orange-Fruit-Pieces.jpg", n.d.).

2.3 Model Choice

Initially, upon researching models to be used, I came across NVIDIA's TAO toolkit, an all in one artificial intelligence toolkit to streamline a workflow ("NVIDIA TAO Toolkit", 2023). The NVIDIA model I originally decided upon was the DetectNet_v2 ("DetectNet_v2", 2021). The implementation of this specific model was to be done following the tutorial posted on the NVIDIA YouTube channel, but that proved much, much harder than expected ("Get Started With NVIDIA TAO Toolkit", 2022).

Originally, the model was going to be implemented entirely using the resources provided in the University of Florida's supercomputer HiPerGator, "a cluster that includes the latest generation of processors and offers nodes for memory-intensive computation" ("HiPerGator", n.d.). The problems started to arise since I had no experience whatsoever using HiPerGator and the documentation was not very user-friendly for first-time users. I then attempted to follow the tutorial on my local machine, but had even more problems since I was not using Linux as

they were in the video and really struggled with interpreting and understanding the commands they were inputting into the command-line interface.

Then, after struggling with DetectNet_v2 for a long time, it was recommended that I look into the YOLO algorithm. Upon researching, I found it quite attractive as a potential solution for the intersection. Not only is it very simple to implement in code but it is lightweight, fast, and very accurate, even at a distance. According to their documentation, the newest version, YOLOv8, places a focus on ”speed, size, and accuracy” (“YOLO: A Brief History”, n.d.). Additionally, all of their models are pre-trained and they have a number of different models with different sizes for different applications.

3 Methodology

To test the YOLOv8 model, I was given a long video of a much more polluted intersection in Gainesville, Florida, specifically West University Avenue and 13th Street. I used the most lightweight model and compared it to the ”heaviest” model, the ”n” model and the ”x” model, respectively. To compare them, I analyzed the average processing time per frame per model and visually inspected how much of the traffic was actually being detected.

3.1 Variables

The independent variables are the model type. The dependent variables are time and accuracy. The control is the computer used.

3.2 Dataset Used

The detect model for YOLOv8 is pre-trained on the COCO dataset which has 80 different classes which include traffic lights, person, bicycle, car, and motorcycle, which are extremely pertinent (Ultralytics, n.d.).

3.3 Process

1. To process the videos to a manageable size, I cut the video to be 10 seconds long.

2. I first navigated to the directory that I installed the YOLOv8 model to on my desktop.

3. I made a Conda environment using the command ”conda create -n yolo” and activated it using ”conda activate yolo.”

4. I then ran the command ”pip install ultralytics” to download all the necessary requirements for YOLOv8.

5. Finally, I ran the command ”yolo detect predict model=yolov8n.pt source=video_name.mp4” to process the video, with ”yolov8n.pt” being replaced with ”yolov8x.pt for the ”heavier” model.

4 Results

When using YOLOv8n, the model was able to process 12 frames, images, per second. However, when using YOLOv8x, the model was able to process only 1 frame, image, per second.

The resulting videos can be found at this link.

The first frame of the YOLOv8n processed video can be seen in Figure 5, and the same frame processed by YOLOv8x can be seen in Figure 6.

Figure 5: This image shows the first frame of the output video from the YOLOv8n model.

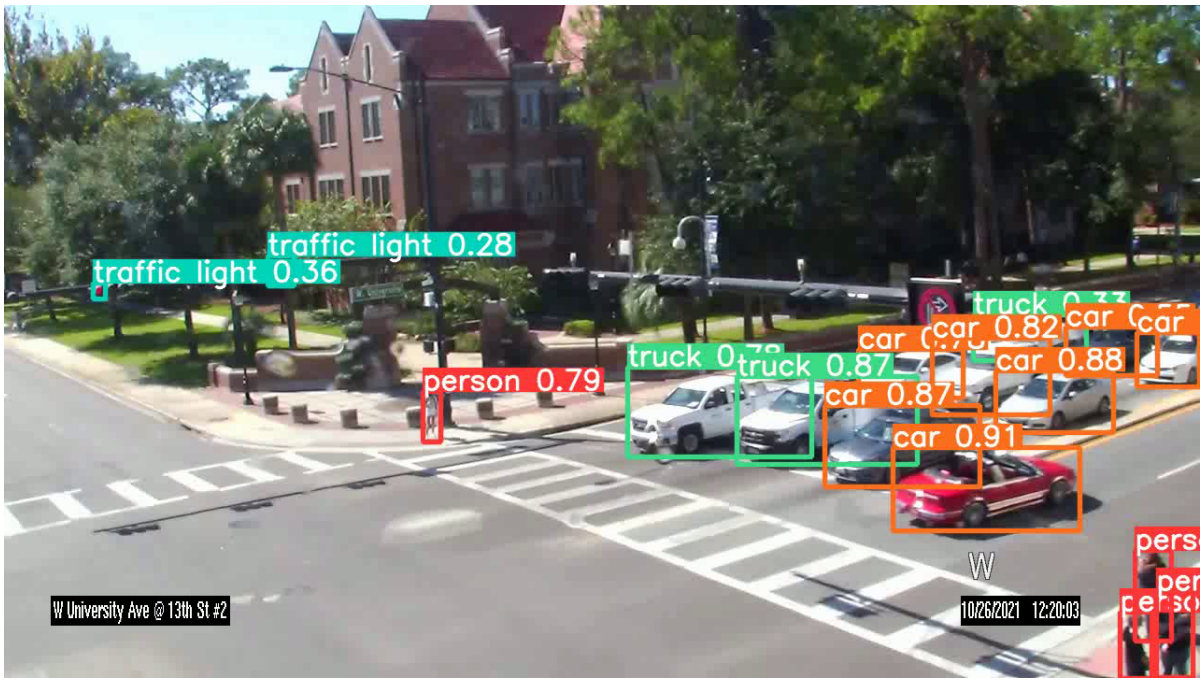

Figure 6: This image shows the first frame of the output video from the YOLOv8x model.

4.1 Discussion of Results

YOLOv8n is a very good option for an application at which speed is necessary in processing an image. In the first frame of the video, it certainly finds all of the cars, and the people that are closer to the camera. However, it does not see any of the traffic lights or the one person very far away. Additionally, it is not distinguishing between the types of vehicles. Although slower, YOLOv8x sees the traffic lights, sees the single person on the opposite end of the intersection, distinguishes between trucks and cars with high precision, and has a much higher accuracy overall.

When thinking about the implications of the speed of processing images for computer vision, the first thing that comes to mind is how quick, precise, and accurate a self-driving car must be. The processing speed for this situation however, is not nearly as important as its accuracy. When comparing YOLOv8n and YOLOv8x, I was almost certain that YOLOv8x should be used for its increased accuracy. If this model is to be used to assist in improving the traffic at Center Drive and Archer Road, a location with a large number of pedestrians, speed is not a matter for concern as per second, not much movement happens at the corners of the crosswalks or in the traffic slowing down to a red light.

5 Conclusion and Proposal

It seems that YOLOv8 is almost too good to be true in terms of solving the problem at this intersection; YOLOv8 processes images at a very high rate, identifies vehicles, pedestrians, and classifies between vehicle types while locating the objects in the images very accurately. I propose that cameras be install at the end of each traffic signal pole and be connected to either a nearby computer or HiPerGator to be processed in real time. This would allow for the camera to be angled downwards towards both the pedestrians and the traffic at each corner and each side, respectively. An image showing the proposed camera positions for a solution can be found in Figure 7. Supposing HiPerGator takes even one second to process each image using YOLOv8x, a Python script could be running that looks for pedestrians, and returns a request for a signal change if there are any. Additionally, the signals could be re-timed to more efficiently allow

for pedestrians to cross while not holding up traffic. Additionally, if there is a lot of traffic (the number of vehicles that defines this must be tested and adjusted), the light may be extended until fewer cars are detected, with the preference of the light toward the side that has more traffic. Although YOLOv8 is an amazing algorithm, tests must be run and traffic flow must be analyzed to ensure that it is improving the flow of traffic and the safety of pedestrians crossing the intersection. Additionally, tests must be run across all conditions to ensure that the camera signal is not obstructed by changes in weather or light. YOLOv8 is a fantastic algorithm that has the potential to be implemented effectively not only at this specific intersection, but at all intersections across the world.

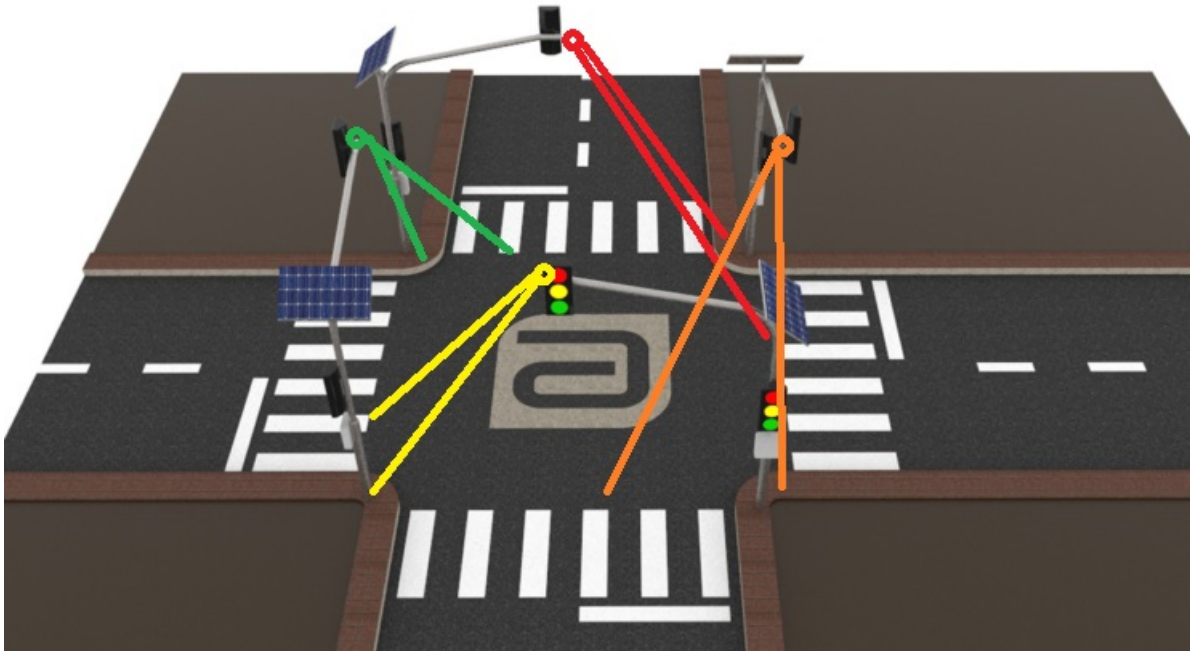

Figure 7: This adapted image shows the location and direction of the proposed cameras for the solution to the problem with each camera as a unique color with two paths showing the pedestrians and the traffic that it will be facing (“Wireless Solar Traffic Light System”, 2021).

5.1 Artificial Intelligence Ethics Discussion

The question of ethics in artificial intelligence is being asked at an increasing rate, especially with models such as OpenAI’s ChatGPT advancing at an astonishing rate (“Introducing Chat-

GPT", n.d.). With YOLOv8 however, the ethical analysis that must be done is external to the model itself; that is the analysis must be done within its applications. Implementing this into traffic signals seems like a fantastic idea, but it may not be ethically sound to be recording live data, even if it's not stored, of vehicle tag information or of pedestrian faces. Additionally, what would happen if a pedestrian was harmed due to the model not correctly identifying them as a person? What if that pedestrian was wearing a very reflective color or was not recognized at all? What if a malicious third party put a cardboard cutout of a pedestrian at the corner of an intersection to fool the model into thinking that a pedestrian was waiting forever? There are so many variables at play, and I believe that testing implementation at a single intersection like Center Drive and Archer Road in Gainesville, Florida, especially with the number of pedestrians coming from both the school and the hospital, would be fantastic. Not only would the testing environment be variable, but it would show real results. Keeping the safety of the pedestrians in mind, I think testing in a virtual environment to get hyperparameters set properly would be in the best interest of all parties. Should the implementation of computer vision into traffic signals put any pedestrians at risk, however, testing should be halted immediately; the safety of life is and should always be at the forefront of testing artificial intelligence in the real world.

5.2 Reflection and Further Research

Should this experiment be replicated, I think it would be pertinent that testing the model be done on a supercomputer such as HiPerGator to see how much the model speed could be improved. Additionally, it would be interesting to see how quickly it could run the suggested four video signals simultaneously. I think that should another researcher take on this task, they might want to connect with a researcher who is proficient in running computer simulations to see what effect this would have on a traffic simulator. I have learned an immense amount about not only the YOLO algorithm, but also the research process and the difficulty of interpreting documentation for a specific use case. Finally, another important step that could be taken in improving the accuracy of the model, and to speed it up, would be to use the YOLOv8n model in place of the YOLOv8x model and train it using data that is hand-picked for this task. It

has been proven that YOLOv8n can reach amazing accuracy using hand-made data. A great example of this is in an experiment in which damaged car parts were detected with accuracy in the range of 90-100% (Remanan, 2023). I hope that this research is carried on and eventually testing is done at a real world location. Although artificial intelligence has taken massive strides in the past few years, immense testing must be done prior to the implementation of a computer vision algorithm in a traffic intersection.

6 References

References

DetectNet_v2. (2021). https://docs.nvidia.com/metropolis/TLT/tlt-user-guide/text/object_detection/detectnet_v2.html

File:Honeycrisp-Apple.jpg. (n.d.). https://commons.wikimedia.org/wiki/File:Honeycrisp-Apple.jpg

File:Orange-Fruit-Pieces.jpg. (n.d.). https://commons.wikimedia.org/wiki/File:Orange-Fruit-Pieces.jpg

Get Started With NVIDIA TAO Toolkit. (2022). https://www.youtube.com/watch?v=jGYxYInlEAk

Google Maps. (n.d.). https://www.google.com/maps

HiPerGator. (n.d.). https://www.rc.ufl.edu/about/hipergator/

Introducing ChatGPT. (n.d.). https://openai.com/blog/chatgpt

Maglev train. (n.d.). https://www.newworldencyclopedia.org/entry/Maglev_train

NVIDIA TAO Toolkit. (2023). https://developer.nvidia.com/tao-toolkit

Remanan, S. (2023). Damaged Car Parts Detection using YOLOv8n: From Data to Deployment. https://blog.paperspace.com/damaged-car-parts-detection-using-yolov8n-an-end-t/

Traffic Signal Components. (n.d.). http://wbt.dot.state.fl.us/ois/tsmo/TrafficSignalBudgetingCostsAccessibleTa
htm

Ultralytics. (n.d.). Ultralytics/coco.yaml. https://github.com/ultralytics/ultralytics/blob/main/ultralytics/datasets/coco.yaml

What is Computer Vision? (n.d.). https://www.ibm.com/topics/computer-vision

Wireless Solar Traffic Light System. (2021). https://lightingequipmentsales.com/wireless-solar-traffic-light-system.html

YOLO: A Brief History. (n.d.). https://docs.ultralytics.com/#yolo-a-brief-history

YOLO: Algorithm for Object Detection Explained [+Examples]. (n.d.). https://www.v7labs.com/blog/yolo-object-detection

7 Appendix

7.1 Python Code

The Python code on the following page was the code used in Jupyter Notebook through HiPerGator to produce the videos in the Google Drive folder provided. Note that the output is stopped at only 21 frames, but was run for the entire length of the video, and YOLOv8n was changed to YOLOv8x for generating the second video.

YOLO Implementation

April 22, 2023

[ ]:
```
from ultralytics import YOLO
model_n = YOLO('yolov8n.pt')
results_n = model_n.predict(source='13wu.mp4', save=True, save_txt=True)
```

```
Ultralytics YOLOv8.0.25 Python-3.8.13 torch-1.13.1 CPU
YOLOv8n summary (fused): 168 layers, 3151904 parameters, 0 gradients, 8.7 GFLOPs

[h264 @ 0x55894513f0c0] Frame num change from 53 to 54
[h264 @ 0x55894513f0c0] decode_slice_header error
[h264 @ 0x55894513f0c0] Frame num change from 53 to 55
[h264 @ 0x55894513f0c0] decode_slice_header error
[h264 @ 0x55894513f0c0] Frame num change from 53 to 56
[h264 @ 0x55894513f0c0] decode_slice_header error
[h264 @ 0x55894513f0c0] Frame num change from 53 to 57
[h264 @ 0x55894513f0c0] decode_slice_header error
[h264 @ 0x55894513f0c0] Frame num change from 53 to 58
[h264 @ 0x55894513f0c0] decode_slice_header error
[h264 @ 0x55894513f0c0] Frame num change from 53 to 59
[h264 @ 0x55894513f0c0] decode_slice_header error
[h264 @ 0x55894513f0c0] Frame num change from 53 to 0
[h264 @ 0x55894513f0c0] decode_slice_header error
[h264 @ 0x5589462eb700] Frame num change from 53 to 54
[h264 @ 0x5589462eb700] decode_slice_header error
[h264 @ 0x5589462eb700] Frame num change from 53 to 55
[h264 @ 0x5589462eb700] decode_slice_header error
[h264 @ 0x5589462eb700] Frame num change from 53 to 56
[h264 @ 0x5589462eb700] decode_slice_header error
[h264 @ 0x5589462eb700] Frame num change from 53 to 57
[h264 @ 0x5589462eb700] decode_slice_header error
[h264 @ 0x5589462eb700] Frame num change from 53 to 58
[h264 @ 0x5589462eb700] decode_slice_header error
[h264 @ 0x5589462eb700] Frame num change from 53 to 59
[h264 @ 0x5589462eb700] decode_slice_header error
[h264 @ 0x5589462eb700] Frame num change from 53 to 0
[h264 @ 0x5589462eb700] decode_slice_header error
video 1/1 (1/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 10 cars, 715.9ms
video 1/1 (2/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 10 cars, 469.6ms
video 1/1 (3/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 9 cars, 486.7ms
```

```
video 1/1 (4/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 9 cars, 1770.3ms
video 1/1 (5/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 9 cars, 471.8ms
video 1/1 (6/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 10 cars, 1 truck,
1003.1ms
video 1/1 (7/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 9 cars, 1 truck,
627.0ms
video 1/1 (8/6134) /home/hstark/13wu.mp4: 384x640 2 persons, 9 cars, 602.1ms
video 1/1 (9/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 9 cars, 1328.3ms
video 1/1 (10/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 8 cars, 1 truck,
2288.3ms
video 1/1 (11/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 8 cars, 1 truck,
1315.1ms
video 1/1 (12/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 9 cars, 1 truck, 1
stop sign, 2879.5ms
video 1/1 (13/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 10 cars, 1 truck,
1 stop sign, 361.6ms
video 1/1 (14/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 10 cars, 478.0ms
video 1/1 (15/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 9 cars, 1 truck,
3055.7ms
video 1/1 (16/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 10 cars, 1 stop
sign, 595.2ms
video 1/1 (17/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 10 cars, 1394.2ms
video 1/1 (18/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 10 cars, 2426.2ms
video 1/1 (19/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 10 cars, 471.7ms
video 1/1 (20/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 10 cars, 955.3ms
video 1/1 (21/6134) /home/hstark/13wu.mp4: 384x640 3 persons, 10 cars, 1 stop
sign, 422.1ms
```

[ ]: