Traffic jams could pose a significant threat to other users of the roads, especially in the case of highway, where vehicle speed tend to be higher. Has it ever occurred to choose the most congested road, while other roads were more viable? What if you're a firefighter, an officer or a paramedic that is required to rush to a specific district? Ever thought that traffic police, having the knowledge of traffic-congested roads, could designate new paths as the most viable ones, in order to ease traffic on the streets and reduce the potential danger for accidents? Part from that, the traffic control center operators could highly benefit from such a possibility, since they will be able to track possible accidents in time and respond to them accordingly. Another use of such knowledge could be in self driving cars or navigation systems to avoid such roads and use alternatives.

Everyone, more or less, would like to know beforehand about the traffic congestion on the streets they're about to drive through, whether that's via a smartphone notification, a web app, or a traffic lightboard. The idea behind the project is to spot vehicles that either stop moving, or move especially slow via usage of CCTV’s that are positioned in most modern highways. This idea occurred to us thanks to Lesson 4's exercise.

The model to track the cars, is the “vehicle-detection-adas-0002”, the same one that was used in the lesson. The input for the application is video from a CCTV, we used one that was available on youtube (link: <https://www.youtube.com/watch?v=PNCJQkvALVc>). We cut a small part of it, using the following app: <https://github.com/mifi/lossless-cut>. The part of the video that was used for the analysis is contained in the project repo (<https://github.com/aaleksopoulos/openvino_project>)

In order to run the model, we run the “app.py” file, with some parameters, following the example of the course exercises. The parameters are

-m: required, the path to the model to be used, as in "models\ssd\_mobilenet.xml" (or ‘/’ in case of Linux/MacOS)

-i: required, the input file, as "input.mp4"

-d: optional, the device name used, default "CPU"

-t: optional, the accuracy threshold of the algorithm, default 0.5

-c: color of the bounding boxes, could either be "YELLOW", "GREEN" AND "BLUE", default "YELLOW"

In the git repo of the project, the “run.txt” file contains a sample command to be used.

The output of the procedure is a video file, that in yellow boxes are presented the vehicles that are tracked to either stop or move slowly. Of course, there are some False Positives (vehicles that are moving just fine but they are tracked from the algorithm), but not as many in number. It seems that most of such cases are large vehicles (trucks or buses), and they could be explained from the way the algorithm tracks the stopped vehicles.

In order to track the stopped vehicles, two different ways were implemented. One that used centroids and one that used the Intersection over Union (IoU, <https://en.wikipedia.org/wiki/Jaccard_index>) idea. In the first, each car, which is defined as an instance of the Tracked\_Cars class (as implemented in the tracked\_cars.py file) is defined by the center coordinates of its bounding box, while the algorithm tries to detect other vehicles that are presented nearby. It turned out that this approach was too fuzzy and did not work well, so we also tried a modification of the IoU idea. In order to track the stopped cars, we wanted the IoU ratio between two consequent frames to be high (above 0.92 in our case). Each tracked vehicle that fulfilled that criteria is surrounded by a bounding box.

Of course, the idea presented in the current project could be further expanded. It could be used in a real “async” way, in which input of more than one CCTV could be used. Also, in similar way, there could be used more models, like identifying humans or animals that may be present in highways that also could pose a threat to the safety of both them and other users of the road. As an expansion of the current project it could also be used to detect congested traffic.

Finally, some screenshots of the output of the algorithm are presented, in which the traffic jam around 1:35 was detected

![Εικόνα που περιέχει τρόπος, σκηνή, δρόμος, ουρανός

Περιγραφή που δημιουργήθηκε αυτόματα]()

![Εικόνα που περιέχει τρόπος, σκηνή, δρόμος, ουρανός

Περιγραφή που δημιουργήθηκε αυτόματα]()![Εικόνα που περιέχει τρόπος, σκηνή, δρόμος, ουρανός

Περιγραφή που δημιουργήθηκε αυτόματα]()![Εικόνα που περιέχει τρόπος, σκηνή, δρόμος, ουρανός

Περιγραφή που δημιουργήθηκε αυτόματα]()![Εικόνα που περιέχει τρόπος, σκηνή, δρόμος, ουρανός

Περιγραφή που δημιουργήθηκε αυτόματα]()![Εικόνα που περιέχει τρόπος, σκηνή, δρόμος, ουρανός

Περιγραφή που δημιουργήθηκε αυτόματα]()![Εικόνα που περιέχει τρόπος, σκηνή, δρόμος, ουρανός

Περιγραφή που δημιουργήθηκε αυτόματα]()![Εικόνα που περιέχει τρόπος, σκηνή, δρόμος, ουρανός

Περιγραφή που δημιουργήθηκε αυτόματα]()![Εικόνα που περιέχει τρόπος, σκηνή, δρόμος, ουρανός

Περιγραφή που δημιουργήθηκε αυτόματα]()![Εικόνα που περιέχει τρόπος, σκηνή, δρόμος, ουρανός

Περιγραφή που δημιουργήθηκε αυτόματα]()