Vehicle Count Model using OpenCV and Python

Arya Dwivedi
Jawaharlal Nehru University, Delhi

Table of Contents

1.) INTRODUCTION:

Traffic analysis is important to accurately reflect the real-world traffic situation to not only avoid accidents and inconvenience but also improve the present condition of roads and traffic systems.

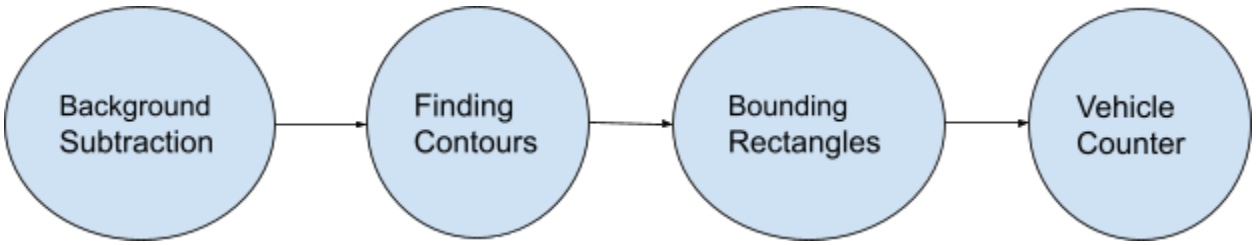
This analysis is done in order to eliminate bottlenecks in the current road transport system and provide an efficient and effective road transport system.

Vehicle counter system can be used for a variety of applications such as optimizing the traffic, suggesting the better route, knowing how safe the road is, etc. One common approach to build these systems is by temporarily deploying a video camera that acquires footage over a period of study. The video produced by the camera is then analyzed to produce data on the routes like counting the number of vehicles, categorizing them, etc.

What I am seeking to do in this project is to build a Vehicle counting model of our own that would reduce the cost of acquiring the data, and making it possible to acquire more information about traffic flow in a timely manner. Using a small inexpensive embedded camera and Deep learning algorithms, we can build a vehicle counter that provides real time information about hundreds of roads and traffic as well as give traffic planners and city operators an unprecedented level of information about traffic conditions.

2.) APPROACH:

Video analysis is a challenging problem due to a number of factors. Shadows, distracting features, Objects other than vehicles, etc, are some examples that make this model more prone to errors and a major source of inaccuracy. However, to build a model to contend with these challenges, the approach I followed was:



2.1) Background Subtraction (BS) is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene) by using static cameras.

In OpenCV we have a few algorithms to do this operation –

- BackgroundSubtractorMOG – It is a Gaussian Mixture-based Background/Foreground Segmentation Algorithm.
- BackgroundSubtractorMOG2 – It is also a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It provides better adaptability to varying scenes due illumination changes etc.
- BackgroundSubtractorKNN – This algorithm implements the K-nearest neighbours background subtraction . Very efficient if the number of foreground pixels is low.

I used BackgroundSubtractorKNN for the project.

2.2) Finding Contours:

Contours are defined as the line joining all the points along the boundary of an image that are having the same intensity. There are multiple contours of different areas in an image . In our case, the contour having the maximum area is the desired region. Hence, it is better to have as few contours as possible.

2.3) Bounding Rectangles:

It is a feature of contours. We used straight bounding rectangles to detect objects. Bounding rectangles are important to estimate the vehicle position more accurately than traditional object detecting methods.

2.4) Vehicle counter:

Finally, to count the number of vehicles passing through a line whose coordinates were set by us, we introduced a few variables like offset (the lines between which a contour detected will be added to the vehicle counter), A valid contour list was created and all the mid points of the contours detected that fell into the valid contour category, were appended to the valid contour list. Valid contour category was introduced to avoid the small area contours that would have increased the errors in our project. That way, objects other than vehicles were detected but didn’t increase the count of vehicles.

A valid contour within the offset liness implies that a vehicle passed through the main line, thus incrementing the vehicle counter.

3.) WHY KNN?

To choose which method would be best suited to complete the task, I tried to compare background subtractor methods on the dataset provided. The code that I used to do the same is given below:

```
vidCap = cv2.VideoCapture('1615363610851.mp4')

BS_KNN = cv2.createBackgroundSubtractorKNN()

BS_MOG2 = cv2.createBackgroundSubtractorMOG2()

while vidCap.isOpened():

        ret, frame = vidCap.read()

        knn_fgMask = BS_KNN.apply(frame)

        cv2.imshow('KNN-Method: Foreground Mask', knn_fgMask)

        mog2_fgMask = BS_MOG2.apply(frame)

        cv2.imshow('MOG2-Method: Foreground Mask', mog2_fgMask)
```

```
        if (cv2.waitKey(1) & 0xFF == ord('q')):

                break

cv2.destroyAllWindows()

vidCap.release()
```

On the basis of the results, KNN appeared to be a more suitable method since there was much more unnecessary detailing involved in the MOG2 method. We were supposed to have much less contours since we are only supposed to detect vehicles and not the surrounding objects. The distracting features of the objects detected were adjusted by the variable 'cont_valid'.

4.) OTHER METHODS:

We used OpenCV's built-in feature, KNN method, to extract the foreground mask. Other methods for extracting foreground mask are:

1) Calculating absolute differences between frames.
2) Using the MOG/MOG2 method.

One method that we could have used was calculating absolute difference between frames, and then applying image processing techniques like thresholding and blurring. This is the manual method where we use while loop iteration after reading the first frame. We read the video frame by frame in the while loop, and then find the absolute difference between each

and every frame. The iteration continues until all the frames are read. This method however, is not practicable for long duration count and when flow is high.

The other method was to use a built-in feature, MOG2 background subtractor, to extract the foreground mask. Although the results were pretty much similar but we happen to notice that there is little more detailing in the MOG2 method, that is not required in our project.

Our aim is to detect vehicles and not the surrounding objects, so to avoid other objects, the KNN method was preferable over the MOG2 method. Here are some pictures to compare the two methods:

MOG2 applied frame:

KNN applied frame:

Absolute Difference:

I have used both Absolute differencing method and KNN method in my task, but KNN model seems to have produced a better result as compared to any other method of background subtraction.

5.) WHY CURRENT MODEL BETTER THAN OTHERS?

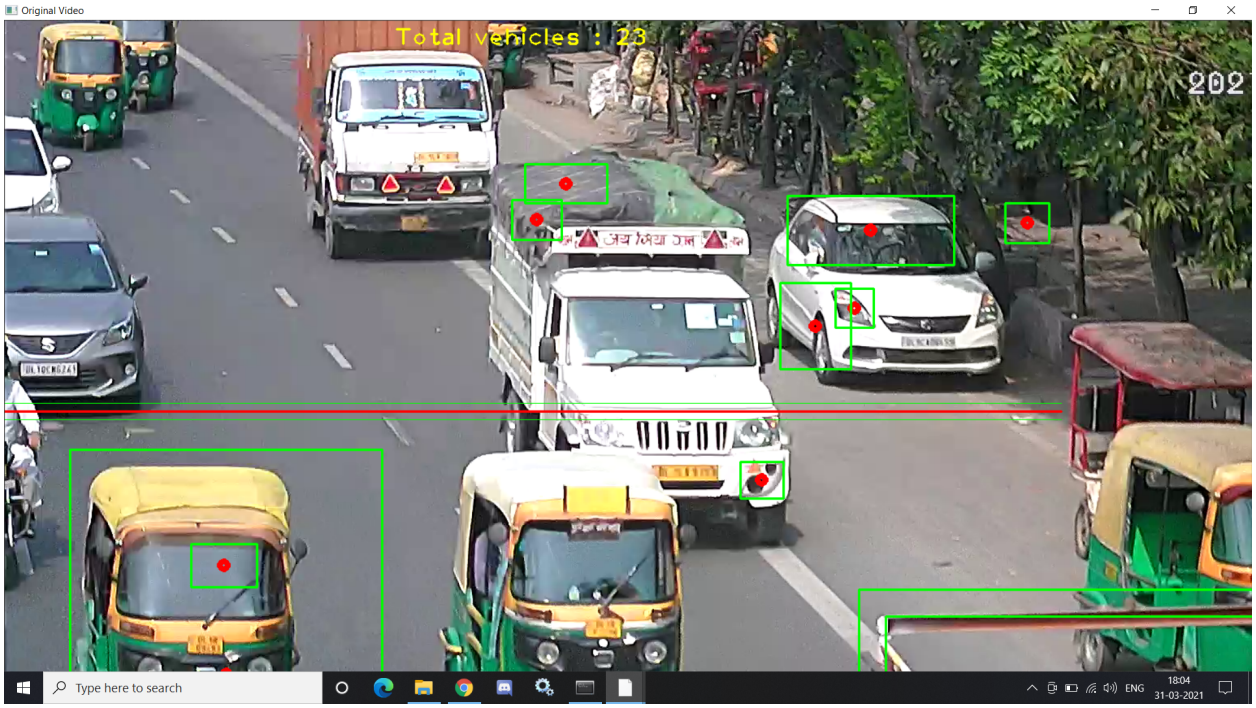
As compared to other models that I tried, my current model gave the most appropriate results.

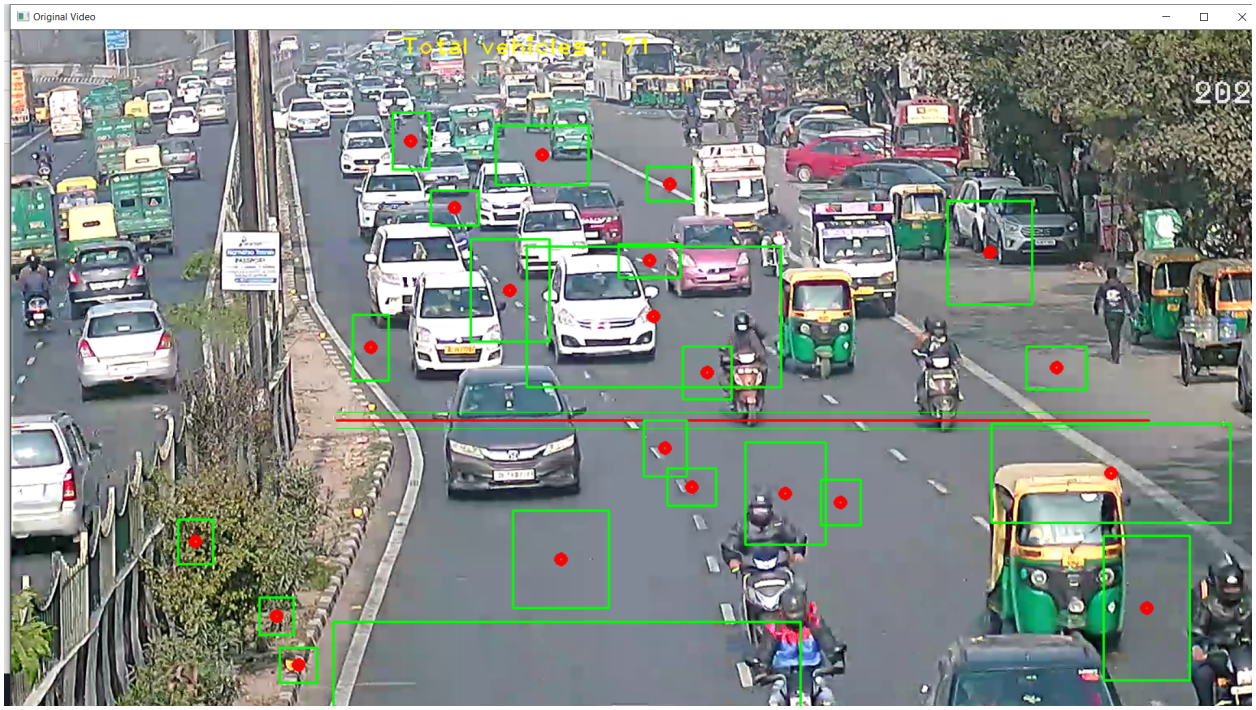
- A minimum contour height and width were set to ignore the small contours, or contours with lesser area to avoid errors.
- A ‘cont_valid’ list was maintained, in which only those contours were added that were appropriate in area and lied within the parameters set by us.
- Out of the valid contours list, only those contours that lied within the offsets were used to increment the number of vehicles.
- Using the KNN background subtraction, those objects with lesser area were avoided, and thus detection of vehicles was easier. This helped in the night mode too.
- By adjusting the parameters such as offset and position of line, we designed an algorithm to count the vehicles crossing the line of given coordinates. (The coordinates were decided by ourselves).

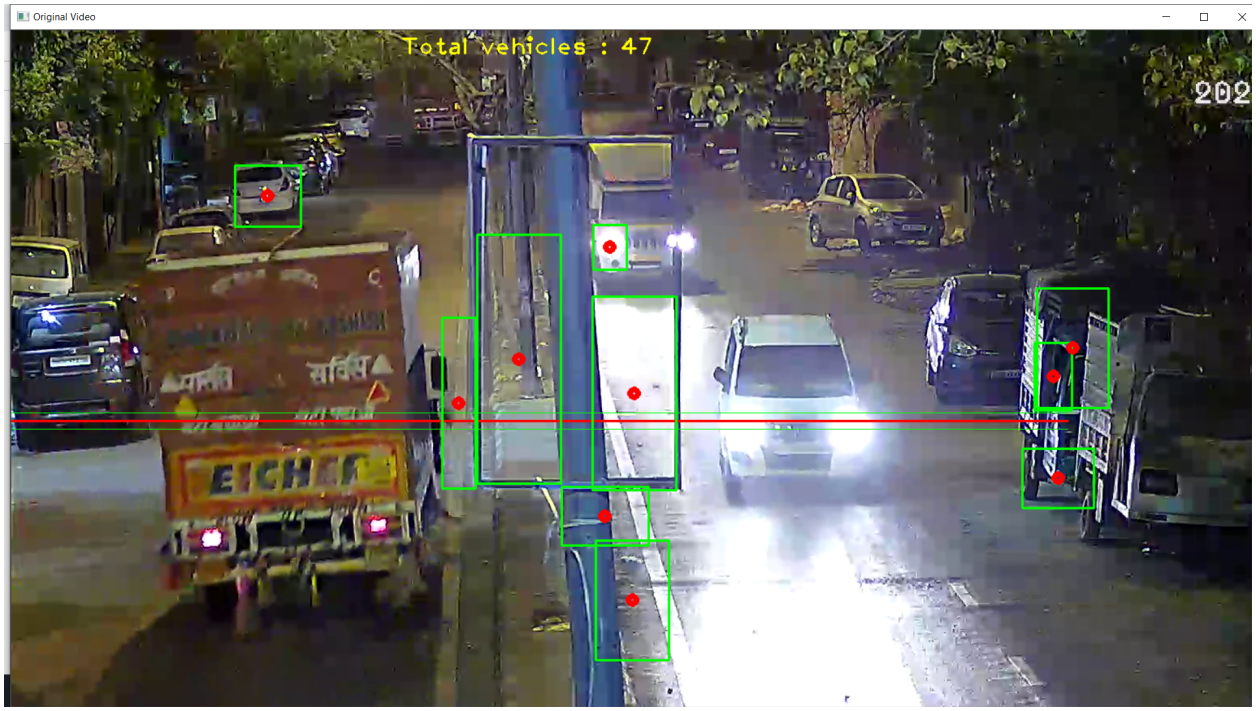
If given more time, improvements could be made to the model such as segmentation of vehicles so that contours do not intersect, or there are not more than one contours for the same object. Also, the following model is limited to a stable camera system, improvements in the model could be

made so that it could be useful in case of dynamic cameras (where the position of cameras is not fixed).

6.) RESULTS:




Original Video
Total vehicles : 71
202


Original Video
Total vehicles : 47
202

Link to the videos of result:

https://drive.google.com/drive/folders/1QS9V1d3nkJeuj6RIbPnga6KWdnH3oo16?usp=sharing

https://drive.google.com/drive/folders/1M16W9RXBWugz4qLYTaNPyNlasibpBX85?usp=sharing