Cloud based Single Shot Multi Box Detection (SSD) Architecture Models for Jakarta Traffic Jam Monitoring

Alfan Presekal, Muflih Fathan, Misbahul Fajri, I Gde Dharma Nugraha, Kalamullah Ramli

***Abstract:** Traffic jam is still one of the main problems in Jakarta Indonesia. To encounter this problem, we proposed system to monitor the road and perform calculation on vehicles speed and road density. To achieve this, we used SSD to detect vehicle traffic on Jakarta's road which obtained from public IP Camera. The main contribution of this work are utilization of public IP Camera and remote traffic analytics via cloud based artificial intelligence system. As a result, the program can perform monitoring on roads condition in real time. The accuracy of the system in average is 80% with highest accuracy achieved 92% to detect vehicle speed and road density.*

***Keywords :** Object Detection, Vehicle Detection, Deep Learning, Single Shot Multibox Detection, Cloud Computing.*

I. INTRODUCTION

Traffic jam is one of the bigest problems in Jakarta Indonesia. Because of the traffic jam, people spend more time on the road. According to the research from Boston Consulting Group's every people in Jakarta spend at least 22 days or 400 hours in a year on the road [1]. With lack of commuting time efficiency, it will cause stress to workers and a financial loss. Its estimated that for every month per person lost equal to 800 US dollar [2].

The government already tries several policies to reduce traffic jam [3], especially during rush hour. But the proposed solution is not fully effective since traffic jam still occur. With artificial intelligence's development, we could try to use artificial intelligence to help reduce traffic jam. The main function of artificial intelligence is to predict data. With artificial intelligence, we could estimate vehicle speed and road density at a time. With the real-time data government could create policy based on artificial intelligence prediction.

Before artificial intelligence could predict the data, they need data to be learned.

Revised Manuscript Received on December 30, 2019.
* Correspondence Author

Alfan Presekal*, Lecture on Computer Engineering, Department of Electrical Engineering, Universitas Indonesia.
Muflih Fathan, Student of Computer Engineering, Universitas Indonesia.
I Gde Dharma Nugraha, Lecture in Department of Electrical Engineering, Universitas Indonesia.
Kalamullah Ramli, Professor on Computer Engineering in Computer Engineering, Universitas Indonesia.

For that, we could use object detection first to detect vehicle and calculate vehicles speed and roads density, we collect the data and will be further processed by the artificial intelligence.

There are several researches related to this work. Chen Kuang-Hsuan et al. use pre-trained SSD model and fine-tuning it with Taiwan's car dataset to made more robust SSD model [4]. Gu Xiaong Feng et al. proposed a modified method from convolutional neural network that add inception modules and Spatial Pyramid Pooling (SPP) layer between convolutional layers and fully connected layers. The results are claimed more accurate than several region-based methods (R-CNN, Faster R-CNN, etc.) but its still has a problem to detect small objects [5].

In this paper, we try to collect data, vehicles speed, and road density. We will try to test how accurate our system to calculate vehicles speed and road density. We use 5 Public IP Cameras to capture roads video and SSD model to detect objects (vehicles).

II. PROPOSED METHOD

The System will be running in google cloud, with details specification at Table-I.

Table-I: Specification System

No	Hardware / Software	System (Google Cloud)
1.	CPU	16 vCPU Intel(R) Xeon(R) CPU @ 2.20GHz
2.	GPU	NVIDIA® Tesla® P100 16GB GDDR5 VRAM
3.	Memory	60 GB
4.	CUDA Version	10.0
5.	CUDA Compute Capability	6.0

We use 5 different videos for system input. We took videos from Publics IP Camera which could be accessed at Jakarta smart city website (smartcity.jakarta.go.id). Each video's duration will be between 15 to 20 minutes.

The Details about videos could be seen at Table-II.

Table-II: Video Input

No.	Name	Cars	Bus/Truck	Brief Condition
1.	Video A	799	49	Bright, Crowded
2.	Video B	305	63	Bright, Deserted
3.	Video C	237	87	Bright, Deserted
4.	Video D	399	89	Bright, Crowded
5.	Video E	329	39	Bright, Crowded

Beside different location, each video also could be distinguished by their condition. Some videos, we took at night and day and we took some video at rush hour, thus the road was quite crowded, and the vehicles were close to each other.

We choose SSD [6] (Single Shot Technique) over Region-based technique, because our system need process in real time. Thus, we choose SSD even though it has worse accuracy than Region-based technique [7]. We use SSD model with VGG19 [6] as feature extractor. We use several variances of SSD (SSD7, SSD300, and SSD512) to get different result.

We use pretrained model which trained with dataset COCO+VOC PASCAL 7+12 as used in previous research. In the previous research is concluded that the best dataset is to use COCO + VOC PASCAL 7 + 12 [9]. In that research, also concluded that the best threshold value is at 0.7 or 0.8 [9], but in this system, we will use 0.55 as threshold value.

The system will start when videos is a given as input to the system. Before the video processed by SSD. We took one frame from the video and we draw two Region of Interest (ROIs) and two lines. We used green's and red's ROI for a green line and a red line. These two lines and ROIs will be used to counting vehicle, vehicle's speed, and road's density. After that, we save the image (the one with lines and ROIs).

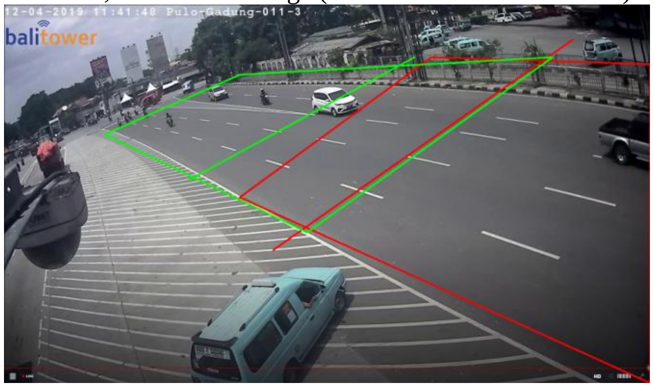

Fig. 1. ***Two ROIs (Red and Green)***

After we have an image with ROI, the system will read every frame and each frame will be processed in the SSD model to detect objects, but before SSD processed, the frame needs to be resized to 300x300 corresponding SSD model that we use (SSD300). SSD will detect objects (vehicle) and draw a boundary box for every object. From that, we calculate the object's centroid position and give every object a unique ID.

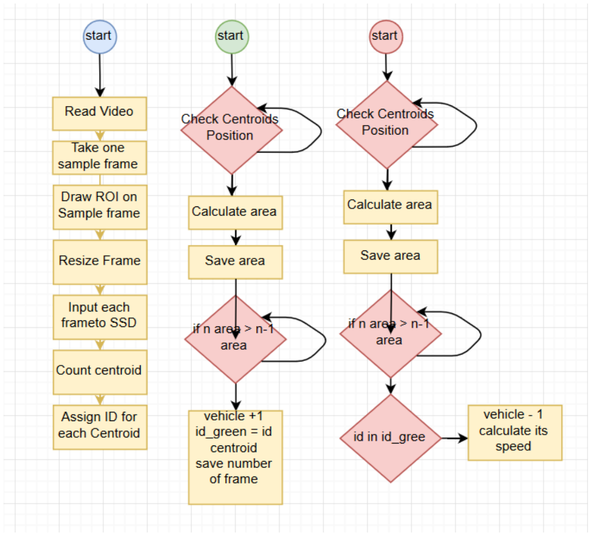

Fig. 2: ***System Flowchart, Colour of circles representations algorithm in line (red and green)***

To calculate vehicles speed and road's density we use two lines (green and red). Where the green line indicates those objects (vehicle) enter the ROI and the red lines indicate objects left the ROI. To know when objects pass the line, we calculate the triangle area (object's centroid and two points end of the line), and we compare area in frame *n* with frame *n* - 1 for every object. If the area at frame *n* bigger than the area at frame *n* - 1, we considered those objects have passed the lines. After we know objects have passed the line, we store its ID and number of frames.

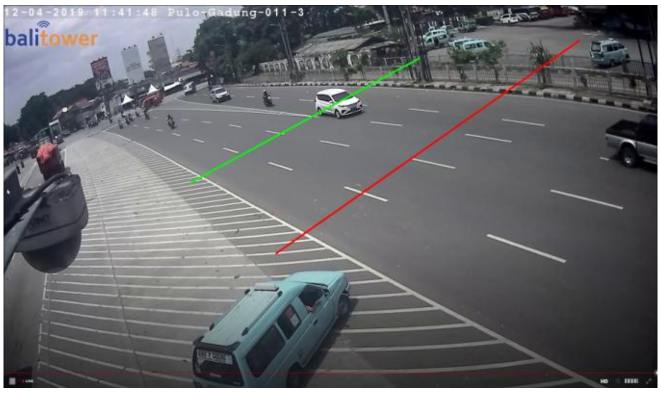

Fig. 3. Red and Green Line for counting vehicle

The two lines (green and red) use the same algorithm to detect if objects have passed the lines. The difference between these two lines is what they do after objects have passed them. In green lines, if objects passed the line, it will increase counter "vehicle in ROI" and store its ID and number of frames.

In red lines, if there are objects that passed line, it will decrease counter "Vehicle in ROI", but before that, we will check objects ID if they are in array ID green lines, if objects ID is not in array ID green lines, we will assume those objects were not passed the green line. There are several reasons why that case could occur, one of them is that at frame = 0 objects are in ROI, so they did not pass the green lines and it could be because model SSD could not detect objects before.

After making sure those objects have passed the green line, we decrease counter "Vehicle in ROI" and calculate its speed with the number of frames when objects in ROI. For every object that did not pass the green line, we do not decrease counter and not calculate its speed.

III. COMPUTATION ALGORITHM

The output from SSD is a frame with a boundary box in each object. We need the object's centroid to find objects position and to calculate vehicle speed and road density. We find centroid with the formula (1) below:

$$X_3 = X_1 + \left(\frac{X_2 - X_1}{2}\right) \qquad Y_3 = Y_1 + \left(\frac{Y_2 - Y_1}{2}\right) \quad (1)$$

Where X and Y are the points of the boundary box's object. We will give an ID to each centroid to differ them.

A. Centroid Tracker

Now we have centroids with ID for each. We need ID to make sure those objects have passed lines or not. But the main benefit to using ID for centroid is to track centroid. Objects in this system are vehicles in a road, thus these objects will always move and not to stay in one place and in other cases it's very likely that SSD sometimes couldn't detect objects in one frame after they could in a frame before. To know if the objects at frame n and frame n - 1 (or even n - 10) is the same object we use centroid tracker.

Centroid tracker is an algorithm that uses Euclidian distance to calculate centroid between frames [10]. Each centroid in frame n - 1 and frame n will calculate its Euclidian distance. Algorithm will choose a shortest centroid in frame n to centroid in frame n-1. The closest one or shortest is considered same object.

B. Centroid Position Against ROI

The next step is to determine the centroid position, is centroid outside or inside ROI. If centroid position is outside ROI, we could ignore it because that object probably is not a vehicle (wrong detection at SSD) or a vehicle outside the road (park, etc.). Every centroid which outside ROI will be ignored and system will not calculate its area to reduce system workload and increase system accuracy.

In this system, there are two ROIs resepectively (*ROI_Red* and *ROI_Green*) as we mention above. ROI_Green is the first half of ROI (Road) where there is a green line and ROI_Red is the last half of ROI where there is a red line. We determine the centroid position with FillPoly Algorithm. Fill Poly is algorithm where we compare color (value) of centroid with a unique color, and the second algorithm is Intersection in which we will count how many lines intersect with ROI.

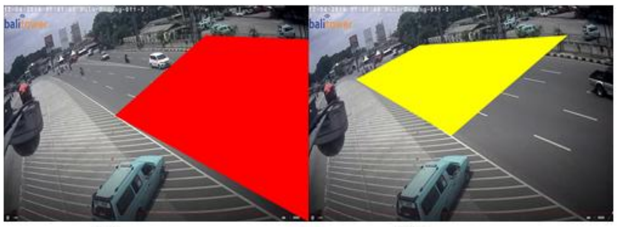
(a) (b)

Fig.4. Red for ROI_Red and Yellow for ROI_Green

In FillPoly Algorithm, we use a function in OpenCV library `fillPoly`, `fillPoly` could give a color to a region. We give green color (0,255,0) in RGB to ROI_Green and red (0,0,255) to ROI_Red. If a centroid has an exact value for three dimensions (RGB) with ROI_Green or ROI_Red, that centroid is in ROI, and vice versa.

Pseudocode FillPoly ROI
```
1. FillPolyROI (centroid, img):
2. count = []
3. for i in range(len(centroid)):
4.  same = 0
5.  for j in range (0,3):
6.   if img [300, 300, j] != img[int(centroid[i][1]),int(centroid[i][0]), j]:
7.   count.append (0)
8.   break
9.  else:
10.   same+= 1
11.  if same == 3:
12.   count.append(1)
13.return count
```

C. Passed The Lines

After we make sure that centroids are inside ROI, we need to find out if objects have passed the lines or not. We could find out with comparing triangle area (centroid and line) at frame n and frame n-1. If area at frame n bigger than when at frame n-1, we could tell that the object has passed the line. Because of objects (vehicles) will always move towards the line, thus at each frame, the triangle area will smaller. The area will become bigger after objects passed the line.

For every object that already passed the line, we will store its ID as a sign that the object has already passed the line. We use ID to find out which objects have already passed the lines, so we could ignore those objects and not calculate their speed to reduce the system's workload. We use the same algorithm (calculate and compare area) for green line and red line.

Pseudocode Areas Calculation
```
1.areaCentroid (centroid, count):
2.global pt1_h, pt2_h, dist
3.areaArray = []
4.for i in range(len(count)):
5. if count[i] == 1:
6.  dist_b1 = math.sqrt((centroid[i][1] –       pt1_h[1])**2 + (centroid[i][0] -pt1_h[0])**2 )
7.  dist_b2 = math.sqrt((pt2_h[1] – centroid[i][1])**2 + (pt2_h[0] – centroid[i][0])**2 )
8.  s = (dist_b1+dist_b2+dist)/2
9.  area = math.sqrt(s*(s-dist)*(s-dist_b1)*(s -dist_b2))
10.   areaArray.append(area)
11. else:
12.   areaArray.append(0)
13.return areaArray
```

D. Calculate Vehicle Speed and Road Density

To calculate vehicle speed and road density we should make sure that the correspondent object has already passed the green line and red line sequentially. If objects passed the red line without passing the green line, we would not calculate its speed nor counting those objects. Because we don't have its ID nor its frame, so we couldn't know which object and for how many frames that these object in ROI.

We calculate road density by subtracting counter vehicle in green line with the counter vehicle in red line.

We considered that the result is how many objects (vehicle) in ROI. After that, we divide the result with the road's maximum capacity [10].
We could calculate the vehicle's speed after we know the object's ID and in which frame objects enter (passed green line) and left (passed red line) the ROI. After that, we find the difference between them. Speed could be calculated with the formula below[11][12][13]:

$$V = \frac{m}{F_{red} - F_{green}} \times \text{FPS} \tag{2}$$

With m is ROI's real distance (meters), Fred is the sequence of frames when objects left ROI, Fgreen is the sequence of frames when objects enter ROI, and FPS is video rate which is how many frames in one second. We could multiply it with 3.6 to get km/s unit.
for road's density, we used traffic flow formula [14][15]:

$$K = \frac{Q}{V} \tag{3}$$

Where K is density (vehicle/hour), Q is the number of vehicles, and V is average speeds. Because we used 15 minutes of video as tests, density (K) will be calculated per minutes (vehicle/minutes). Thus speed that already calculated before will be converted to meter/minutes.

IV. RESULT AND ANALYSIS

A. System Performance and Accuracy

The system runs on several different variables. In addition to video input and the model used, we also use thresh_frame. Thresh_frame is a threshold, up to how many frames since an object disappears its ID is retained. In this trial a limit of 50 and 10 frames was used.

From this trial, the size of FPS and computation time will reflect the performance of the system, as well as the number of vehicles detected on green lines and red lines that describe the accuracy of the system.

Table-III: ***Average Result based on Video***

No.	Video	FPS	RedAcc	GrnAcc
1	Video A	23.925	81.810	76.680
2	Video B	24.65	66.033	44.701
3	Video C	24.43	80.170	68.133
4	Video D	24.31	92.111	85.861
5	Video E	24.6125	78.601	69.429

Table-IV is the result of the average trial based on the input video. With each video input it takes about 15 minutes. The results of the FPS values obtained are around 24 FPS that affect the duration of the computational results, the smaller the FPS the more time is needed. Only Video A (23.925) have FPS below 24. This is due to the large number of vehicles in video where there are 848 vehicles in Video A. This number is larger than other videos 368, 324, 368 for Video B, Video C, and Video E respectively. The greater number of vehicles the more centroid positioning and calculation will be carried out which also affects performance.
Accuracy is obtained with the following formula:

$$100 - \frac{|Number\ of\ Detections - Total\ Vehicles|}{Total\ Vehicles} \tag{4}$$

The worst accuracy is found in the Video B with 66.033% on the red line and 44.701% on the green line. This can be caused by several factors, mainly the road conditions and road structure, and the position of the camera. In table 4.2 we can also see the accuracy on the red line and the green line. All accuracy on the red line (second line) have better values. This is caused by several conditions, namely the object has not been detected by the SSD model when it passes the green line but has been detected on the red line. So that the number of vehicles detected on the red line is larger and more accurate.

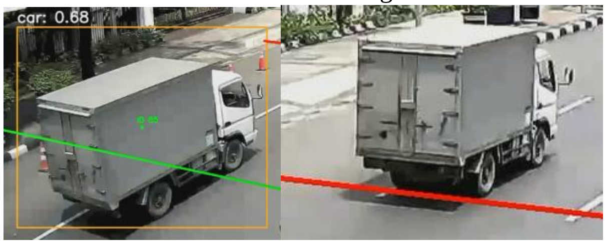

Fig.5. Example how model could detect truck at one time, but couldn't detect after that.

The other reason is, sometimes trucks and buses are detected as car and sometimes is not. The model could not detect them as a bus or truck class.

Table-IV: Average Result based on thresh_frame

Thresh_frame	FPS	RedAcc	GrnAcc
10	24.365	87.296	78.061
50	24.406	72.193	59.861

Accuracy in thresh_frame 10 is much better than thresh_frame 50. It happens because, in thresh_frame 50, ID objects are not deleted even after objects pass the ROI. If there are objects that enter the ROI, the system will assume it's the same objects and assign him its ID, therefore the number of vehicles that detected is much lesser than thresh_frame 10. The other problem that occurs in both is sometimes, the model doesn't assign object an ID, even though the model could detect it and classification it. Because of it, the number of detected vehicles is decreased because the system will ignore objects that don't have ID.

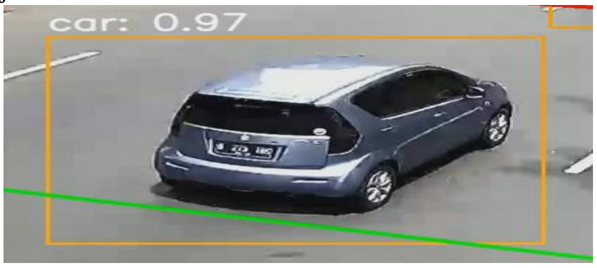

Fig.6. Car which detected by model but doesn't assign ID

As we mentioned before, we will also test our system at night. For that, we use the same location that we used to test at noon. Each video is 10 minutes long.

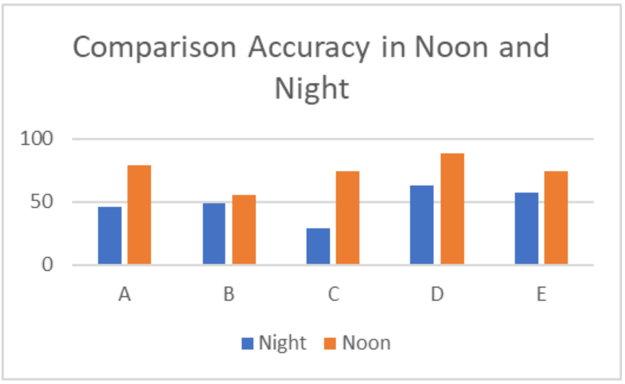

Fig.7. Comparison accuracy's result between test noon and night.

The graphic above is a comparison between accuracy's result at noon and night. As we can see that, for each video, night's accuracy is worse than noon. It even has a big difference in several videos. The biggest difference occurs in Video C with the difference around 45%.

The worse accuracy at night could happen because of several reasons. First, the road is darker than noon, therefore objects (vehicles) became vague as they blend with the background, especially vehicle whose has a dark color.

Second, the lamp of vehicles made the lighting in the video is worse. Because of the position of the camera in Videos is in front of vehicles, thus the camera could only see the circle light of the car's lamp, therefore objects became vaguer. In Figure 8 we could see that the least difference in accuracy occurs in Video B, it happens because only in Video B camera is behind the vehicle, thus it is not too affected by car's lamp.

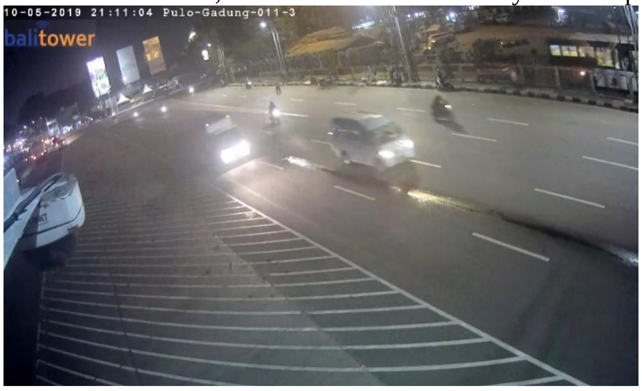

Fig.8. Car's lamp that makes vehicle vaguer.

And the third reason is, all of the videos at night have lower sharpness than video at noon, even though they have the same resolution (1920 x 1080).

B. Speed and Road Density Detection

To calculate speed, we use the best combination in previous trial (SSD300 + thresh_frame = 10). Before we could calculate vehicle speed, we should make sure that vehicle already passed green line and red line. If any vehicle didn't pass one of them, we would ignore the vehicle.

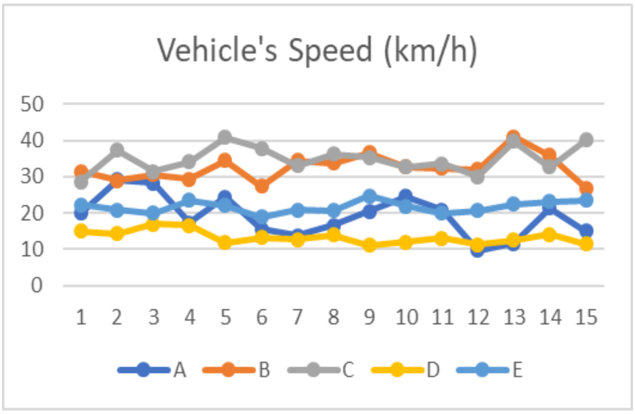

Fig.9. Average speed in Video for each minute.

Figure 9 is displaying average speed in videos for each minute. The average speed in D and B is quite stable around 14 km/h in Video D and 22 km/h in Video B. The average speed in rest of videos is unstable with much changing in every minute. After we calculate average speed, we could calculate road density using traffic flow formulas.

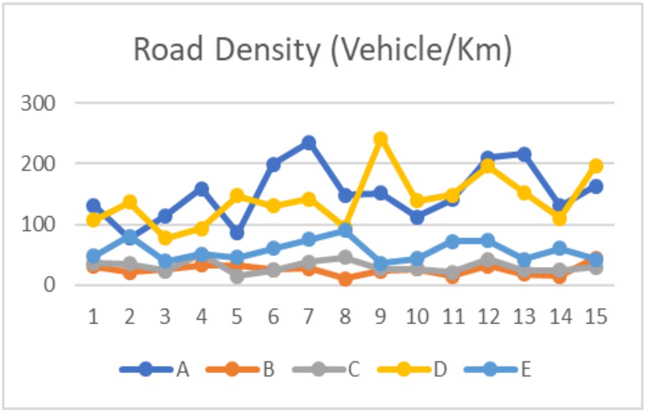

Fig.10. Road density in videos for every minute.

In figure 10 we could see road density in videos for every minute. Video B and C have quite a stable road density for 15 minutes while the other's not. Not only have stable road density, Video B and C also have low road density around 25 vehicles/km for video B and 13 vehicles/km for video C.

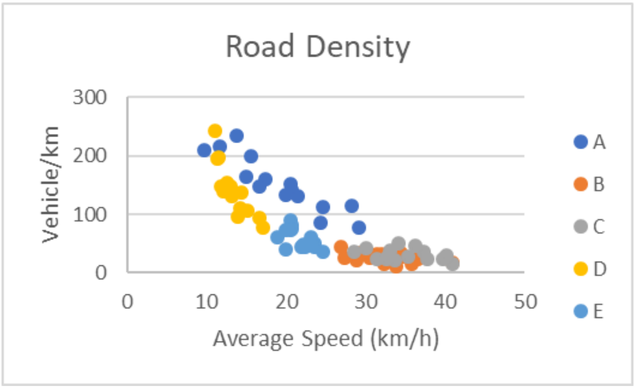

Fig.11. is two-dimensional graphics that consist of average speed and road density in every video.

From Figure 11 we could see the relation between speed and road density if the average speed is high then road density probably is low, vice versa.

thus, the relation between average speed and road density is inversely proportional. But to know road density we couldn't just see the average speed; we should also see the number of vehicles on the road. We can't tell if the road is crowded just based on average speed. When the average speed is low it maybe not crowded, and also when the average speed is high is not guarantee that the road is not crowded. Because with the same average speed, the road density could different depends on how many vehicles on the road.

V. CONCLUSIONS

Based on the experiments, we could conclude that the proposed method could be implemented to monitoring road by calculated vehicle's speed and road density. Although, in our experiment we only calculate in every minute and the obtained data is processed as offline video, but this system could be implement in real time and calculate the traffic density in every hour. Overall the accuracy of the system is about 80% with highest accuracy achieved 92%. The system is still need improvement to increase the accuracy. Especially when there are many vehicles appear.

ACKNOWLEDGMENT

This research is supported by funding from Ministry of Research and Higher Education, Republic of Indonesia 2019.

REFERENCES

1. The Boston Consulting Group. Unlocking Cities, The impact of ridesharing in Southeast Asia and beyond. November 2017
2. Kadarisman, Muh. Gunawan, Aang. Ismiyati. Kebijakan Manajemen Transportasi Darat dan Dampaknya Terhadap Perekonomian Masyarakat di Kota Depok. Jurnal Manajemen Transportasi & Logistik (JMTranslog) – Vol. 03 No. 1. March 2016
3. Chen Kuang-Hsuan et al.Vehicles Detection on Expressway Via Deep Learning: Single Shot Multibox Object Detector. Proceedings of the 2018 International Conference on Machine Learning and Cybernetics.2018
4. Gu Xiao Feng, et al.Real-Time Vehicle Detection and Tracking Using Deep Neural Networks. IEEE. 2016
5. Liu, Wei. Fu, Cheng Yang, et al. SSD : Single Shot MultiBox Detector. UNC Chapel Hill. 2016
6. Hui. Jonathan, 'Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)'.2018.[Online].Available: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359 .[Accessed: 10 – Feb – 2019]
7. Ferrari. Pierluigi, A Keras port of Single Shot MultiBox Detector. GitHub repository: https://github.com/pierluigiferrari/ssd_keras
8. Izzan Dienurrahman, Rancang Bangun dan Analisis Sistem Penghitung Kendaraan Berbasis Deep Learning dengan Arsitektur Single Shot MultiBox Detector (SSD). Universitas Indonesia. 2018
9. Rosebrock. Adrian, 'Simple object tracking with OpenCV'. 2018. [Online].Available: https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/ [Accessed: 22 – Mar – 2019]
10. Helmiriawan, Rancang Bangun dan Analisis Sistem Pemantau Lalu Lintas Menggunakan OpenCV dengan Algoritma Canny dan Blob Detection. Universitas Indonesia. 2012
11. Joshi, Shweta. Vehicle Speed Determination Using Image Processing. Rajarambapu Institute of Technology. 2014
12. Mangala A.G, Dr. Balasubramani. A Review on Vehicle Speed Detection Using Image Processing. International Journal of Current Engineering and Scientific Research (IJCESR) Vol 4. Issue-11. 2017
13. Osman Ibrahim, Ahmed M. Elshafee. Speed Detection Camera System using Image Processing Techniques on Video Streams. IEEE. 2011
14. Ahmed Al-Sobky, Al-Sayed. M.Mousa, Ragab. Traffic Density Determination and Its Applications using Smartphone. Alexandria Engineering Journal (2016) 55, 513 – 523
15. A.D. May, Traffic Flow Fundamentals, Prentice Hall, Englewood Cliffs, 1990.

AUTHORS PROFILE

Alfan Presekal is fulltime lecture on Computer Engineering, Department of Electrical Engineering, Universitas Indonesia. He got Bachelor Degree of Computer Engineering from Universitas Indonesia. He participated in young researcher at Tokyo Institute of Technology from 2012 to 2013. He finished his Master Degree in Secure Software System from Imperial College London, UK.

Muflih Fathan is former student of Computer Engineering, Universitas Indonesia. He got Bachelor Degree of Computer Engineering from Universitas Indonesia in 2018.

Misbahul Fajri is Doctoral student at Department of Electrical Engineering, Universitas Indonesia. He is lecture of Information System in Mercubuana University.

I Gde Dharma Nugraha is fulltime lecture in Department of Electrical Engineering, Universitas Indonesia. He got Bachelor and Master Degree from Electrical Engineering Universitas Indonesia.

Kalamullah Ramli is a Professor on Computer Engineering in Computer Engineering, Universitas Indonesia. He finished his Masters in Telecommunication Engineering at University of Wollongong, NSW, Australia, in 1997. He then continued his Doktorarbeit on Computer Networks in year 2000 at Universitaet Duisburg-Essen, NRW, Germany, and obtained his Dr.-Ing. in year 2003.