Empirical Evaluation of Vehicle Detection, Tracking And Recognition Algorithms Operating On Real Time Video Feeds

Yunik Tamrakar

Empirical Evaluation of Vehicle Detection, Tracking And Recognition Algorithms Operating On Real Time Video Feeds

by
Yunik Tamrakar

A thesis submitted to the faculty of The University of Mississippi in partial fulfillment of the requirements of the Sally McDonnell Barksdale Honors College.

Oxford
May 2020

Approved by

Advisor: Dr. Yixin Chen

Reader: Dr. Byunghyun Jang

Reader: Dr. Charles Fleming

ABSTRACT

A traffic surveillance camera system is an important part of an intelligent transportation system.(Zhang et al., 2013) This system is capable of performing useful object detections on the incoming feed. These detected objects can then be used for tracking purposes which forms the basis for monitoring important traffic data such as collisions, vehicle count, pedestrian count and so on. Furthermore, other additional information such as the weather conditions, time of day as well as date can also be extracted from a live feed. (Sun et al., 2004) Different algorithms can yield different results for any given video input. Not only that, various parameters such as the resolution, frames per second(fps) count, lighting conditions in the video input also affect performance. Therefore, it is imperative to compare between the various image processing and deep learning algorithms and evaluate their performance before deploying them in real time.

ACKNOWLEDGEMENTS

First of all, I'd like to thank my thesis advisor Dr. Yixin Chen for giving an opportunity to participate in this project. His guidance, feedback and encouragement was pivotal. Also, I am extremely grateful to the entire team of UrbanLogiq, particularly to Ms. Duong Vu for providing the necessary datasets and research material for the project. I would also like to thank Dr. Byunghyun Jang, Dr. Charles Fleming, the Computer Science department and the Sally McDonnell Barksdale Honors College for their valuable assistance in this endeavour.

Last but not the least, I'm grateful to all my family and friends for giving me constant motivation and inspiration in every turn of life.

PREFACE

The video feeds have been taken from the real time camera feed from the streets of a real city (labelled X here for confidentiality). The video data set includes clips from a specific traffic intersection and has a regular flow of cars, trucks and pedestrians. The videos are available in both the 10 fps and 30 fps speed.

The algorithms discussed in this paper were run on the videos mentioned above. The videos contain recordings from both day and night time settings. For the purpose of testing the algorithms, other traffic feeds from YouTube have also been utilized.

TABLE OF CONTENTS

LIST OF FIGURES

SECTION 1

INTRODUCTION

The field of Computer vision has made a lot of strides over the past decade. Where once traditional image processing techniques were dominant, with the advancement in GPUs, cloud computing and hardware, deep learning has established itself as another key component in building intelligent systems (Hegde et al., 2019).

There are numerous object detection as well as classification algorithms that can be used for performing the detection and classification task. Some of these algorithms are based on deep learning whereas some rely on more fundamental image processing techniques. Depending on the context of where the intelligent detection/classification system will be deployed, some algorithms might prove to more efficient or more accurate as compared to the others.

One important consideration we need to make before deploying our traffic monitoring software in the real field is to find a balance between speed and accuracy. The model should be able to perform fairly accurate detections and at the same time, be fast enough to run real time such that it does not miss any frames. Therefore, this project tests some popular algorithms on the X video data set (The city's name is not disclosed for confidentiality) and evaluates their performance for various vision based tasks namely: Optical Character Recognition, Object Detection, Object Tracing, Object Counter and Object Classification. The project uses 26 frames per second videos and runs the algorithms on each frame. However, the results that appear in this paper are screenshots of select frames from the video.

SECTION 2

OPTICAL CHARACTER RECOGNITION

Optical Character Recognition (OCR) has become a very important and useful tool in various fields such as school, office, industry and so on. It has significantly reduced the need for human intervention in tasks such as converting written texts to digital format, grading choice based papers as well as extracting useful characters and sentences from images and videos (Mori et al., 1999). For our purposes, we only consider the OCR from videos and images. The OCR model that was used for this study is Pytesseract.

2.1 **Pytesseract**

Pytesseract is an optical character recognition tool for python. It is a wrapper for Google's Tesseract which is an open source OCR engine. (Zelic, 2020). It is based on the Long Short Term Memory (LSTM) architecture, which is one form of a Recurrent Neural Network (RNN).

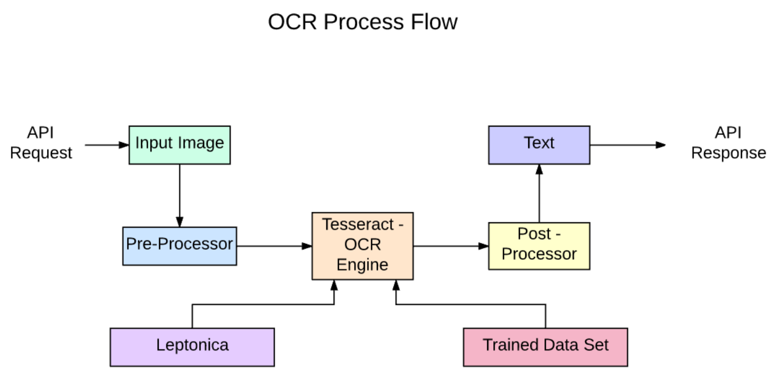


Figure 2.1: OCR Process Flow (Parthasarathy, 2018)

Despite the engine's own pre-processing pass, the user will be required to perform his own pre-processing for better results. In this study, several pre-processing techniques were applied for better results.

2.1.1 **Pre-Processing**

The Pre-processing pass consisted of the following phases, namely: Region of Interest (ROI) extraction, Scaling, Contrast Enhancement and Thresholding.

2.1.1.1 **ROI Extraction**

Each individual frame in the video consisted of the traffic feed and a small section where the date and time was displayed. This caption section was treated as the ROI and since, the ROI was fixed for the subsequent frames, cropping of the ROI did not introduce any problems.

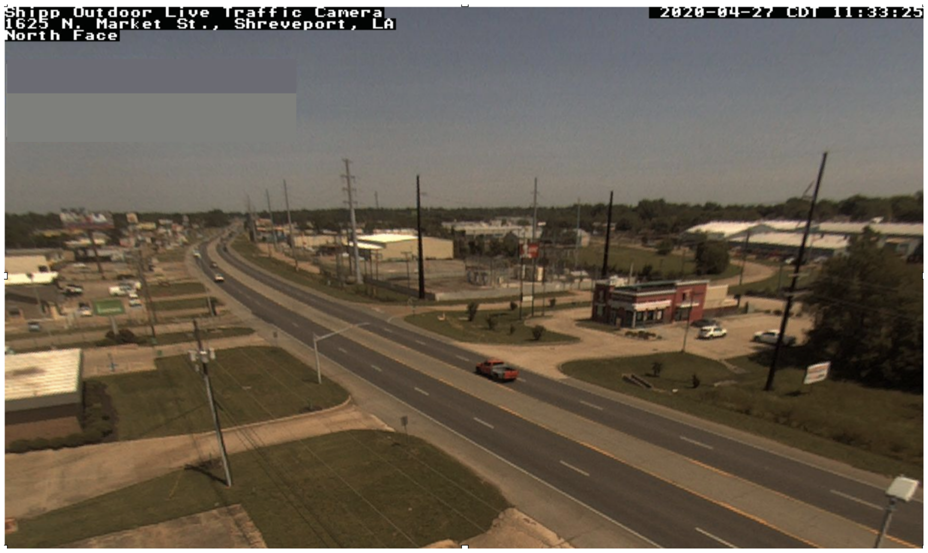

Shipp Outdoor Live Traffic Camera
1625 N. Market St., Shreveport, LA
North Face
2020-04-27 CDT 11:33:25

Figure 2.2: Sample frame from the video feed.

The individual frames were first converted to grayscale, followed by cropping in OpenCV and saved as an image file locally.

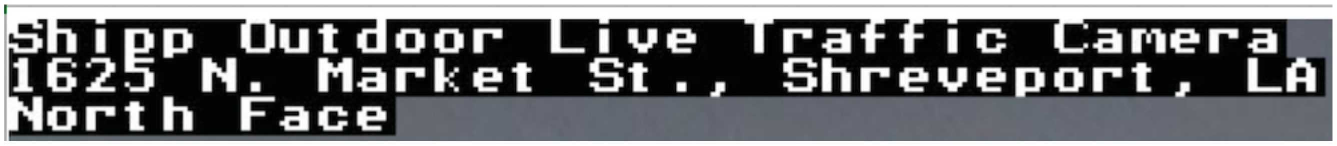

Shipp Outdoor Live Traffic Camera
1625 N. Market St., Shreveport, LA
North Face

Figure 2.3: Cropped Region of Interest Containing Characters

2.1.1.2 Contrast Enhancement, Filtering and Thresholding

The cropped grayscale ROI was then subjected to contrast enhancement using Histogram equalization. The resulting image was then treated using a Gaussian filter to remove the additional noise from the image. This was followed by thresholding, which is a basic yet very important image segmentation technique. Using this method, the characters in the foreground were separated from the background. Various thresholding techniques were explored with the aid of OpenCV and Python and the results were saved locally.

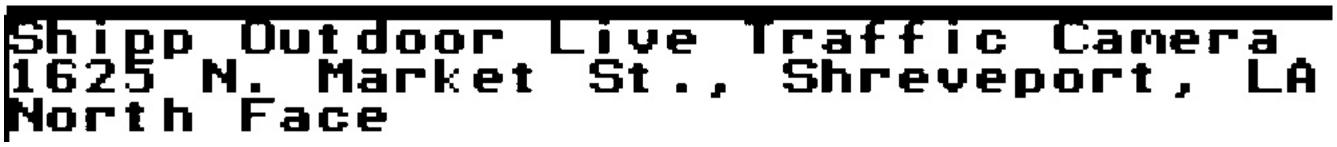

Shipp Outdoor Live Traffic Camera
1625 N. Market St., Shreveport, LA
North Face

Figure 2.4: Simple Binary Inverse Thresholding

2.1.2 Feeding the input into the Pytesseract Engine

The Threshold image was then passed onto the Pytesseract Engine and the results were observed using the default segmentation mode.

2.2 Observations

The Pytesseract engine did not produce robust results when colored images were fed into the system. In fact, for the given video frames, no output was observed when the RGB versions were provided as input. Even the grayscale versions did not produce any output on their own and thus, image processing techniques such as contrast enhancement, smoothing and thresholding had to be applied to yield some results.

Another key observation was that the output of the OCR engine was affected by the type of the type of thresholding that was being used. The best results were produced when the simple binary threshold technique was applied. While this output was not totally accurate, it did yield some meaningful information, which on further fine tuning can be improved.

The results below were obtained while applying the thresholding on the sample frame.

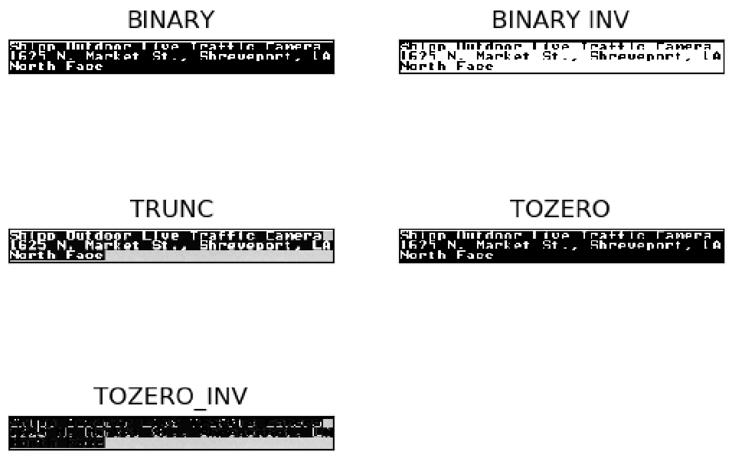
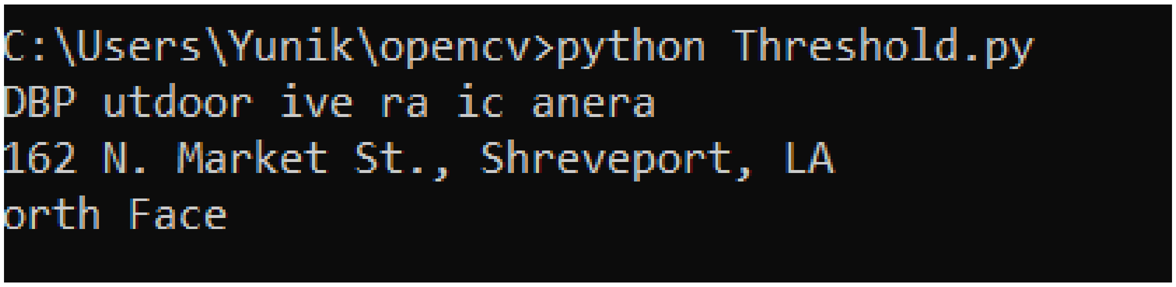


Figure 2.5: Different methods of thresholding

The black text on the white foreground produced by the Binary Inverse thresholding produced the best results. The final console output after performing all the pre-processing is shown in Figure. 2.6:



```
C:\Users\Yunik\opencv>python Threshold.py
DBP utdoor ive ra ic anera
162 N. Market St., Shreveport, LA
orth Face
```

Figure 2.6: Console output from the Pytesseract OCR engine

While the output is not totally accurate and there is still room for improvement, it can definitely be considered useful. However, it is to be noted that this output was observed for a single static frame of the video; applying the same technique for each and every frame of the video did reduce the accuracy of the output.

2.3 Conclusions

Therefore, it can be concluded that Tesseract is a viable option for performing OCR on the given video feed but only after performing several pre-processing passes locally. Tesseract makes use of deep learning and is thus computationally expensive. So, a suggested course of action would be to call the Tesseract API only after a reasonable time interval, say a time interval of one minute, so as to maintain a healthy working speed and also to minimize the inaccuracies caused by using Tesseract on each and every video frame.

SECTION 3

VEHICLE DETECTION

Vehicle detection is an integral component of traffic surveillance (Hu et al., 2018). Over the years, numerous approaches have been devised to tackle this problem. These detection methods can be broadly classified into three categories: motion-based approaches, feature-based approaches, Deep learning based approaches (including CNNs, SSDs, YOLO) (Chen et al., 2018). For the purpose of this research, one approach was tested for each of the first two categories. The deep-learning based approach utilized two different alogorithms.

3.1 **Motion-Based Approach**

3.1.1 **Frame Subtraction**

In this relatively simple method, the difference between two consecutive frames was calculated to detect the moving object using OpenCV and Python. First the video was converted to grayscale, followed by smoothing, thresholding and contour extraction. The bounding boxes were drawn around the moving contours determined by the algorithm.

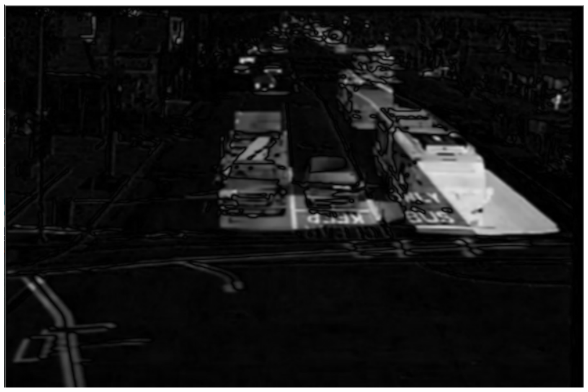


Figure 3.1: Frame Difference

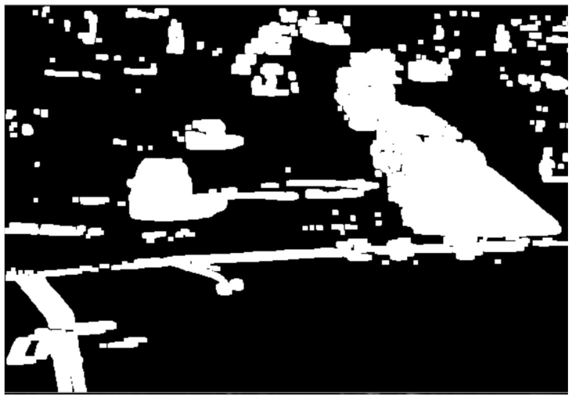


Figure 3.2: Frame Difference

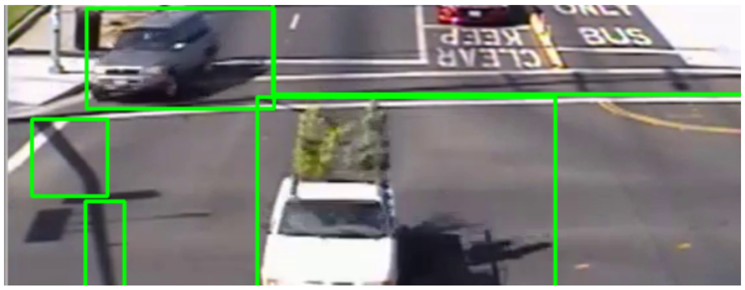


Figure 3.3: Object Detection based on Consecutive Frame Difference

As we can observe Figure 3.3, the bounding boxes around the moving boxes are not very accurate. In fact, this approach does not account for still vehicles and this algorithm is affected by video quality, shifting pixels in between the frames, illumination conditions as well as the presence of shadows.

3.2 Feature-Based Approach

3.2.1 Haar Cascades

Haar Cascades are machine-learning object detection algorithm which is based on the concept of feature detection proposed by Viola and Jones, also known as the Viola-Jones Algorithm (Kasinski and Schmidt, 2007). The Viola-Jones algorithm uses a sliding window to look for useful features which is then used to train a model and then use it as a classifier.

The Haar Cascades utilize the Viola-Jones algorithm to search for a specific type of feature, known as haar like features. The Haar-like features can be classified into three major types: edge features, line features and four-rectangle features. The Algorithm looks for the

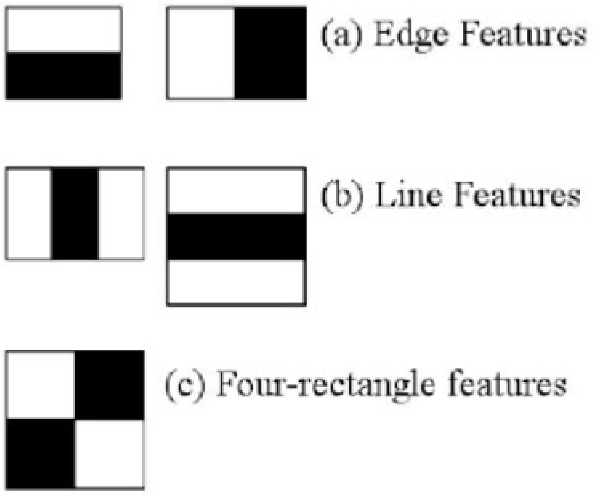


Figure 3.4: Haar Like Features (Zelic, 2020)

haar-like features in an image as shown in Figure.3.4 and this process is optimized by using techniques such as intergral image computation, scaling, adaptive boosting and cascading. Adaptive boosting or adaboost is an important Machine learning technique where weaker, complimentary features are ensembled with the stronger features to improve accuracy of the model.

A pre-trained haar-cascade model for vehicles was used for the research (Andrewssobral, 2020). The performance of the model was analyzed for different samples and the results were noted.

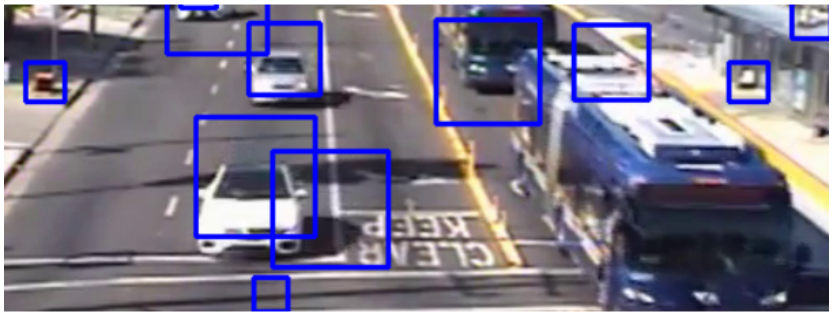


Figure 3.5: Haar Cascade Vehicle Detection for X feed

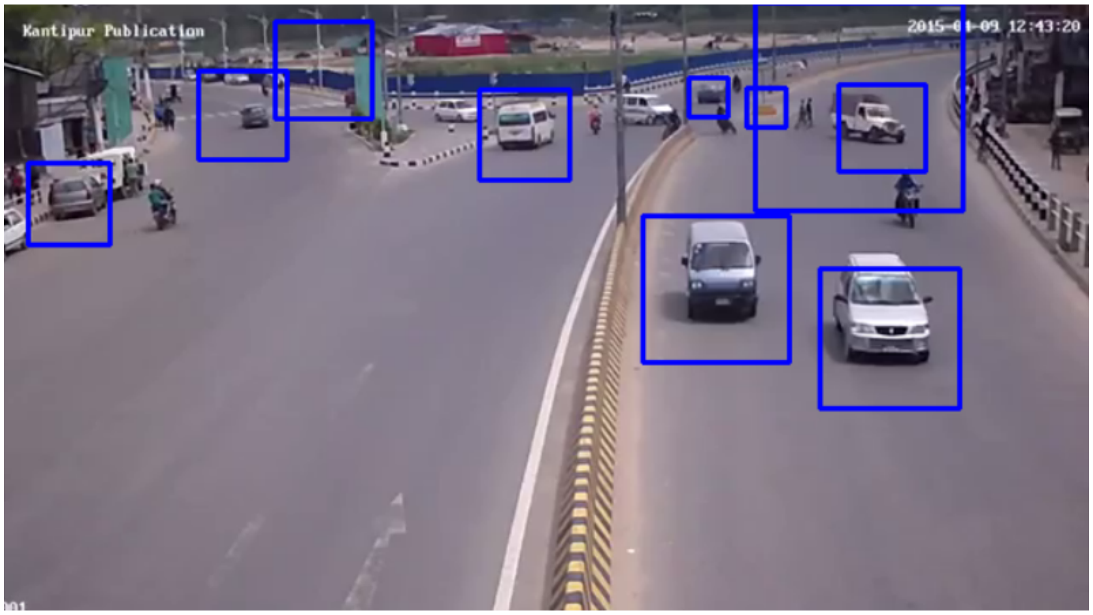


Figure 3.6: Haar Cascade Vehicle Detection for sample video

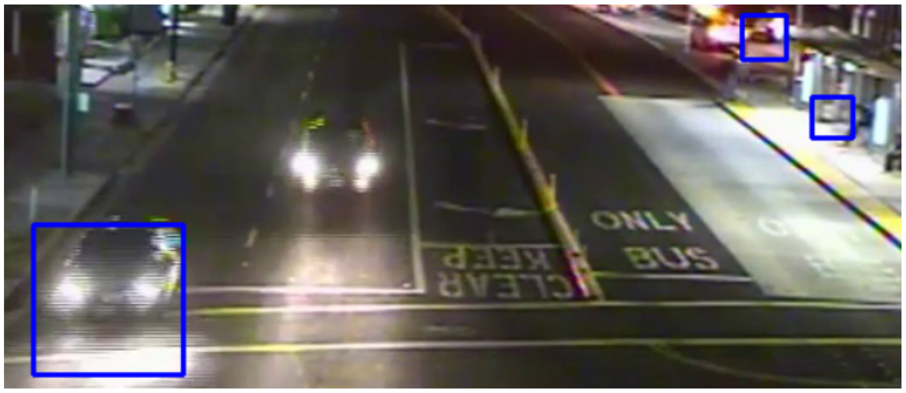


Figure 3.7: Haar Cascade Vehicle Detection for X at night

From the figures provided above, it was observed that the haar cascade is not consistent in its detections. In the X feed (Figure 3.5), the algorithm ends up detecting several false positives and even runs into the same shadow problem that the frame difference algorithm ran into, as discussed in Section 3.1.1. However, the results were much better for the second sample video as the feed was cleaner and it did not have the artifacts that altered the performance of the haar cascade as in the first feed. The night feed did not have the shadow issue but still was not as accurate.

Therefore, the generic use of haar-cascades was found to be inconsistent among other issues. For better results, the haar-cascades should be fine-tuned and possibly trained to match a given dataset.

3.3 Deep Learning Based Approach

3.3.1 Single-Shot Multibox Detectors (SSDs)

SSDs is one of the faster and more accurate object detection algorithms in the world of computer vision today. While detecting objects, or in this case vehicles, the algorithm will divide the image into numerous boxes and all the boxes are passed through a single CNN at the same time.

In other words, it will perform its computations in one shot. The other approach would be making object proposals or using a sliding window in a frame and running a

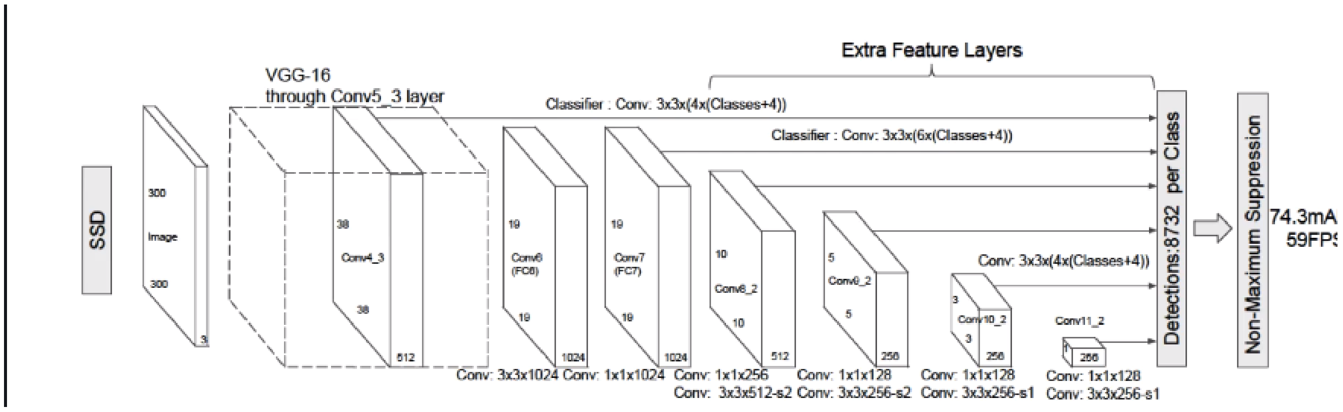


Figure 3.8: SSD architecture

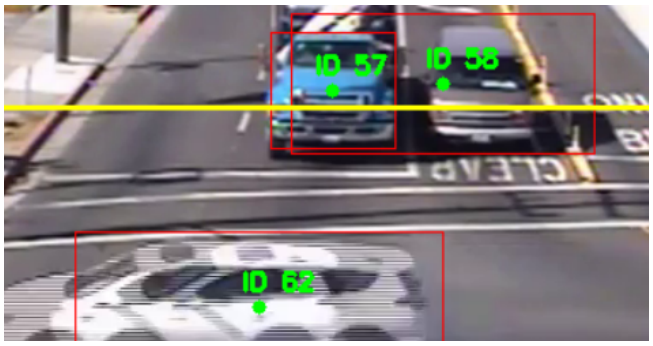


Figure 3.9: Detections performed by the SSD algorithm on the X Daytime feed

CNN for each sliding window. Running multiple CNNs would be an extremely expensive computation process. Thus, the single-shot approach presented by SSDs have been found to be faster as well as accurate, making it a suitable choice for real time computer vision tasks (Liu et al., 2016).

In this project, the pre-trained SSD model that was used was based on the MobileNet architecture. The SSD model is a Caffe version of the original version that was written in Tensorflow by Howard et al. (Howard et al., 2017) and was trained by chuanqi305 (chuanqi305, 2020). While the MobileNet architecture is not as fast as the ResNet or VGG architecture, it tends to run a lot faster, which is important for real time deployment.

The model was run using the Deep Neural Network (dnn) model in OpenCV and the observations were recorded.

The performance of the SSD Algorithm for the daytime feed was better than that of the methods evaluated previously. This can be observed in Figure 3.9. We can note that the shadows are no longer detected as a moving vehicle and the shifting pixels in between the frames too did not cause any problems. However, one major issue that was noted while implementing this particular model was that it had major problems detecting the vehicles in the lane heading away from the camera.

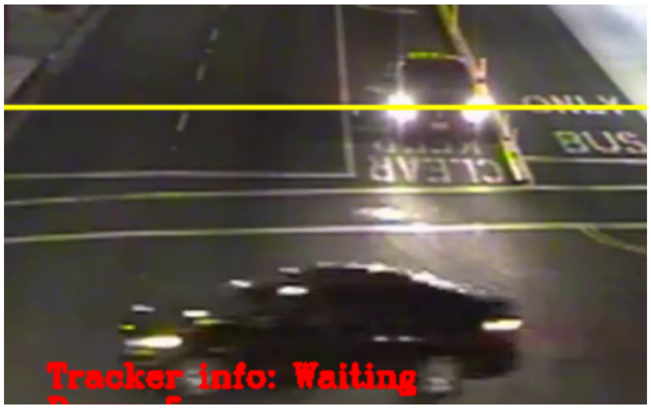


Figure 3.10: SSD algorithm on the X Night feed

For the nightime feed, the SSD algorithm did not perform as well as there was a significant number of missed detections. Even the regions of the frame where notable accuracy was observed during the day time feed, demonstrated signifcant number of misses at night.

3.3.2 **You Only Look Once (YOLO)**

YOLO is another popular and fast algorithm that is widely used for vehicle detection as well as classification. Similar to the SSD, YOLO is also based on the single shot principle, which uses the underlying CNN only once to reduce computation time. YOLO has been proven to work extremely fast, with frame rates of up to 45 frames per second. Furthermore, a lighter version of YOLO called tinyYOLO has been able to attain fps counts of up to 155 fps (Redmon et al., 2016). Also, a more recent study on YOLO-lite has shown an even improved performance that reaches speeds of upto 244fps (Huang et al., 2018)

A YOLOv3 model was used for this project. YOLOv3 is substantially larger than the basic YOLO model and more accurate as well (Redmon and Farhadi, 2018). This model was pre-trained on the COCO dataset by the DarkNet team (Tensorflow). The model was used with the DNN module in OpenCV and tested on both the day and night time feed from the X dataset.

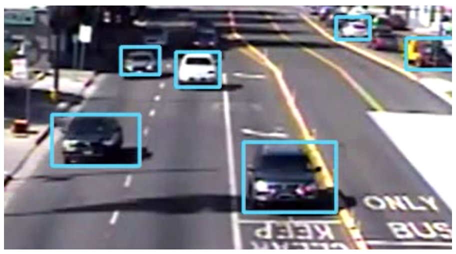


Figure 3.11: YOLO detctions on the X day time feed

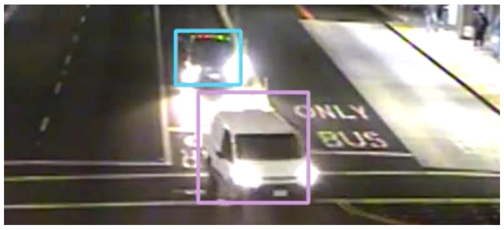


Figure 3.12: YOLO detctions on the X night time feed

As can be observed, YOLO provided the best detection results among all the algorithms during both the day and night time. It was not affected by the shadows or noise during the day and also did not have any problems detecting the vehicles moving away from the camera.

3.3.3 Other Deep Learning Approaches

Other than SSD, the only other Deep Learning algorithm used was the YOLO (You Only Learn Once) algorithm, which will be elucidated further in the Classification section of the paper. More popular deep learning methods such as a Convolutional Neural Networks were not explored as they are significantly slower as compared to the Single Pass Detection algorithms (SSD and YOLO) and as a result, are not very convenient for real time detections.

SECTION 4

VEHICLE CLASSIFICATION

Vehicle classification has emerged as another important application of computer vision and AI techniques. It has found widespread application in fields like surveillance, security system, traffic congestion, avoidance, accident prevention etc (Kamavisdar et al., 2013). While numerous classification algorithms based on digital image processing, deep learning and data mining have been devised, this paper uses only deep learning approaches for the empirical study. The two algorithms that have been explored for the classification problem are: Single-Shot Multibox Detectors (SSD) and You Only Learn Once (YOLO), both of which are single shot algorithms that run the underlying Convolutional Neural Network only once.

It is important to note that accurate detection is crucial for performing the classification task properly. The performance of both the SSD and YOLO algorithm for detection has been explained previously, so this section will only touch briefly on the classification aspect.

4.1 **SSD For Classification**

The MobileNet SSD consisted of 21 object labels including cars, buses and people. As discussed before, the model tended to miss several detections. However for the detected objects, the results were recorded. The missed detections have been circled in red.

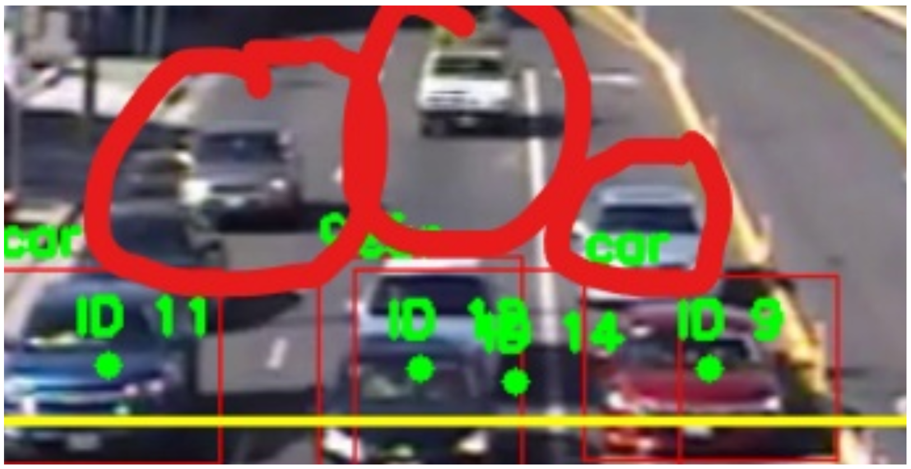


Figure 4.1: Cars Classification

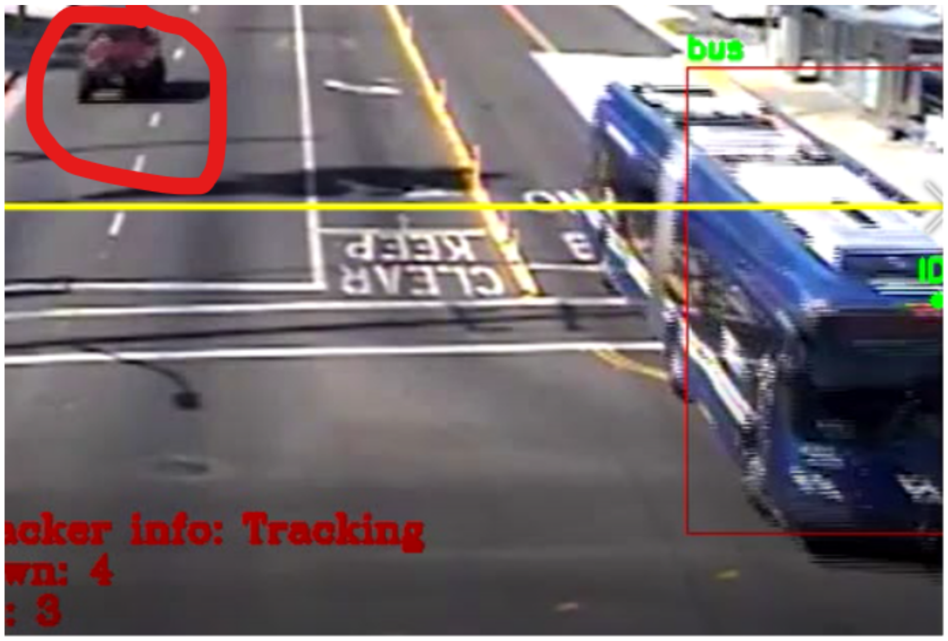


Figure 4.2: Bus Classification

From the results above, it was found that the model is capable of detecting different types of vehicles but only if their scale is large enough. The algorithm does not detect or classify relatively smaller objects such as pedestrians or even vehicles moving away from the camera. Another issue was inconsistencies in the detections. As shown in Figure.4.3, the algorithm labelled the same bus from Figure 4.2 as a car. To deal with this issue, the detections were filtered out on the basis of confidence values.

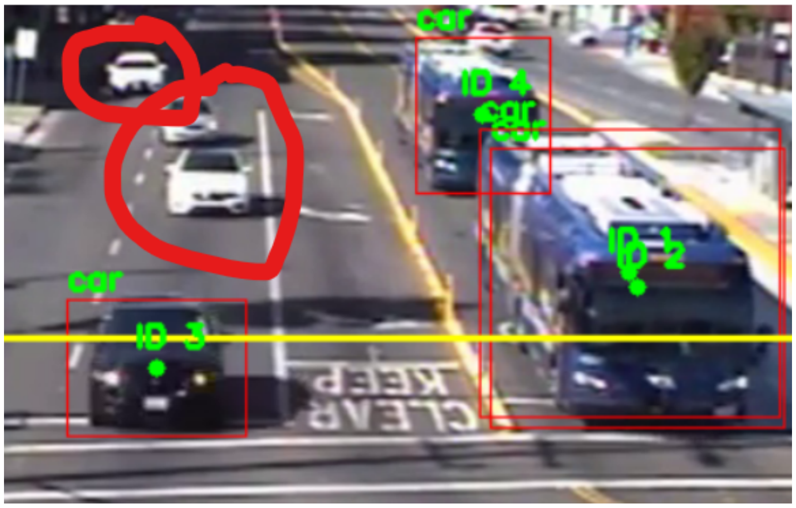


Figure 4.3: Bus Classification

The total number of detections for different confidence thresholds was recorded. The following result was obtained.

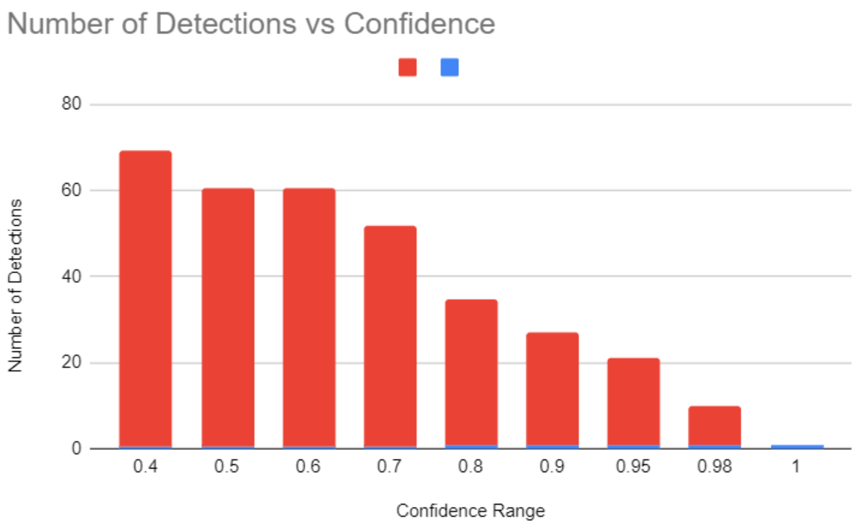


Figure 4.4: Detections vs Confidence

For a 5-minute 30 fps daytime clip from the X dataset, most of the detections were found to be in the 0.4 to 0.7 confidence range. This resulted in mixing up between the labels. To decrease false positives, the confidence threshold was increased to values greater than 0.7. That however resulted in a significant number of missed detections.

4.2 YOLO For Classification

As discussed in the previous section, the YOLO algorithm demonstrated a clear advantage when it came to accurate detections. Furthermore, YOLO demonstrated a much better performance for classification.

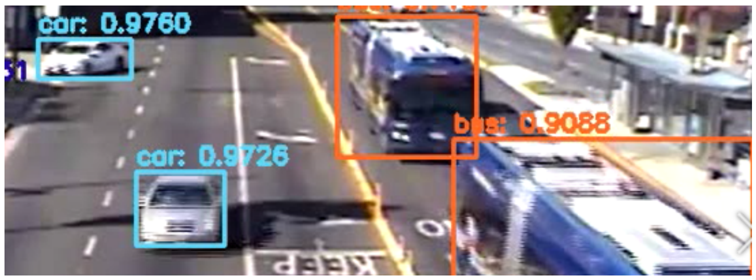


Figure 4.5: YOLO Classification for Cars and Buses

As shown in the figure above, YOLO can distinguish between cars and buses without running into any major issue, as in the case of SSD. We can also note that the confidence of the classified objects are in the high 70s and even the 90s. This is a major improvement over the SSD model.

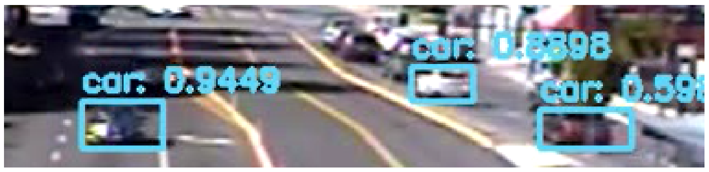


Figure 4.6: YOLO Classification on the Day Feed

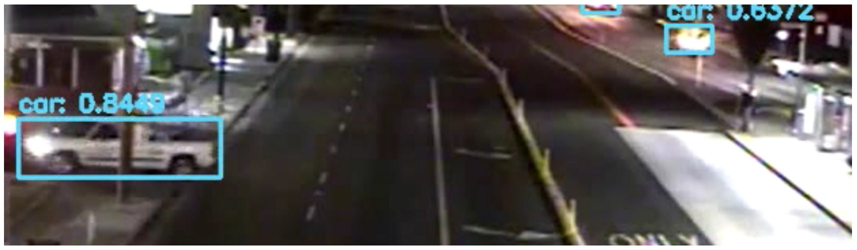


Figure 4.7: YOLO Classification on the Night Feed

The YOLO model runs smoothly on the night time feed as well and does not struggle to detect the vehicles moving away from the camera. Thus, scaling was not an issue. The shadows during the day time caused no problems either.

4.3 **Conclusion**

The YOLOv3 algorithm produced the most accurate results and was not affected by illumination, shadows, noise or scaling. The shadows and noise were not an issue for SSD, but illumination and scaling was a challenge. Haar-cascades and frame-delta approach struggled with false-positives and the other aforesaid issues.

SECTION 5

VEHICLE TRACKING AND OBJECT COUNTING

The vehicle tracking component of the project was built after the detection and classification pass was completed. The detected objects were tracked using the Centroid-Tracking algorithm and the counting was done using OpenCV and Python.

5.1 Centroid-Tracking

The Centroid tracking algorithm assumes we are taking in the set of x,y coordinates for each of the objects that has been detected in each frame. Each of these bounding boxes are assigned unique IDs and a centroid is calculated.

For each subsequent frame, centroids are calculated. However, new IDs are not assigned for each object. The algorithm determines an association between the old object centroids and the new object centroids. The Euclidean distance between each pair of old centroid and the new centroid is calculated (Fu and Han, 2012).

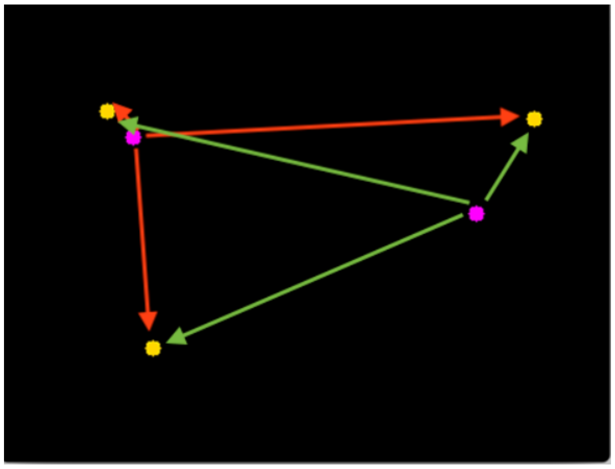


Figure 5.1: Euclidean distance calculation between each pair of new and old object

In the figure 5.1, the purple points represent the old object centroids and the yellow points represent the new object centroids in the subsequent frame. Therefore, there are initially two objects in the frame (purple) and in the next frame, there are three (yellow). The distance between each of the new and old object centroids pair is calculated. The green arrow represents the distance between the first old object centroid and the new centroids, whereas the red represents the distance between the second old object centroid and the new centroids,

The Centroid-tracking assumes that an object will move slightly in between frames, but the distance between the centroids for frames F_t and F_{t+1} will be smaller than all other distances between objects. Therefore, the pairs with the minimum euclidean distance were treated as existing objects. And, the rest were registered as new objects, assigned unique object IDs and the centroid was computed.

Ultimately, when the objects disappeared from the frame for 50 frames, they were deregistered. To make programming easier, each detected object was treated as a *TrackableObject* object in Python, which contained a unique ID and the centroid as its object properties.

5.2 **Object Counting**

For convenience, a horizontal line was drawn across each frame. The object had to pass the line to be counted as an object. Two counters were used in this project: one counted the number of vehicles moving upwards (towards the top of the frame) and the other counted the vehicles in the downward direction (bottom of the frame).

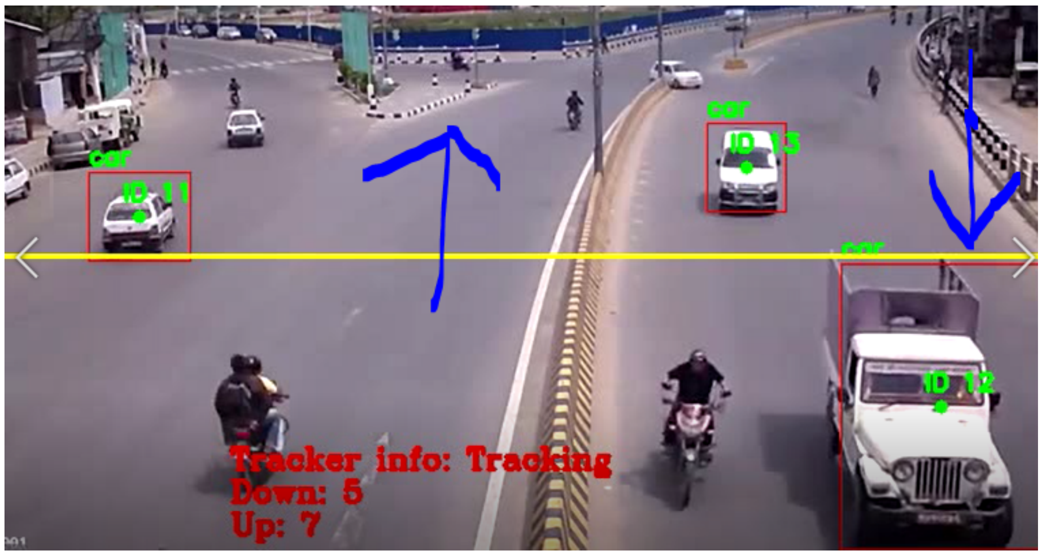


Figure 5.2: Counting algorithm. Yellow represents the reference line (Rosebrock, 2020). SSD Mobilenet v2 was used for detection.

To determine if the vehicle is moving up or down, the difference between the current y-coordinate of the current centroid and the mean of all previous centroid locations is computed. The mean is used so that our tracking is more stable.

5.3 Conclusion

While this method did a fair job of counting and tracking the objects, it has several drawbacks.The problem of occlusion was a major concern. When two objects appeared to overlap with each other, the object IDs tended to get switched. Also ,it requires object detection to be run on every frame, which when used with computationally expensive detectors will slow down the performance.

SECTION 6

CONCLUSION

Most of the evaluated algorithms were affected in some measure by factors such as illumination, noise, shadows, scaling and even the input video. Only the results given by YOLO was steady across the various conditions mentioned above. Using a pre-trained model introduced inconsistencies in the study. This demonstrates the importance of training the dataset that is very similar to the given problem firsthand, so as to improve the accuracy of the detections and customize the model accordingly.

Thus, based on the observations, YOLOv3 was concluded to be the most robust detection/ classification algorithm and centroid-tracking was determined to be a simple yet effective tracker/counter for the X dataset. The Mobilenet SSD was found to have handled noise and shadow well but struggled with consistency. Whereas, the haar-cascades and frame difference methods provided the least accurate results. And last but not the least, Pytesseract provided useable OCR results for the videos, although the demand for pre-processing was very high.

SECTION 7

POSSIBLE EXTENSIONS AND FUTURE WORK

Based on the observations, the YOLOv3 algorithm provided a very sturdy method for both detection and classification. The results from this model can be used to extend the project into performing other important tasks such as collision detection in a robust manner. Also, the YOLO model can further be improved by training the model to match the dataset for the X videos. While the MobileNet SSD that was used here had its drawbacks, researches have suggested it is capable of performing much better(Redmon and Farhadi, 2018). Thus, this model can be improved and trained to match the data so as to improve the results and explore it as a possible alternative to YOLO if need be.

Future work for the project includes detecting the environmental conditions such as weather, detecting the type of collision, that is primary or secondary collision, status of traffic signal, the vehicles involved in the collision, estimated speed, etc.

Bibliography

Andrewssobral (2020), andrewssobral/vehicle$_detection_haarcascades.$

Chen, L., F. Ye, Y. Ruan, H. Fan, and Q. Chen (2018), An algorithm for highway vehicle detection based on convolutional neural network, *EURASIP Journal on Image and Video Processing*, 2018(1), 109.

chuanqi305 (2020), chuanqi305/mobilenet-ssd.

Fu, Z., and Y. Han (2012), Centroid weighted kalman filter for visual object tracking, *Measurement*, 45(4), 650–655.

Hegde, R. B., K. Prasad, H. Hebbar, and B. M. K. Singh (2019), Comparison of traditional image processing and deep learning approaches for classification of white blood cells in peripheral blood smear images, *Biocybernetics and Biomedical Engineering*, 39(2), 382–392.

Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam (2017), Mobilenets: Efficient convolutional neural networks for mobile vision applications, *arXiv preprint arXiv:1704.04861*.

Hu, X., X. Xu, Y. Xiao, H. Chen, S. He, J. Qin, and P.-A. Heng (2018), Sinet: A scale-insensitive convolutional neural network for fast vehicle detection, *IEEE transactions on intelligent transportation systems*, 20(3), 1010–1019.

Huang, R., J. Pedoeem, and C. Chen (2018), Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers, in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 2503–2510, IEEE.

Kamavisdar, P., S. Saluja, and S. Agrawal (2013), A survey on image classification approaches and techniques, *International Journal of Advanced Research in Computer and Communication Engineering*, 2(1), 1005–1009.

Kasinski, A., and A. Schmidt (2007), The architecture of the face and eyes detection system based on cascade classifiers, in *Computer Recognition Systems 2*, pp. 124–131, Springer.

Liu, W., D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg (2016), Ssd: Single shot multibox detector, in *European conference on computer vision*, pp. 21–37, Springer.

Mori, S., H. Nishida, and H. Yamada (1999), *Optical character recognition*, John Wiley & Sons, Inc.

Parthasarathy, B. (2018), Build your own ocr(optical character recognition) for free.

Redmon, J., and A. Farhadi (2018), Yolov3: An incremental improvement, *arXiv preprint arXiv:1804.02767*.

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi (2016), You only look once: Unified, real-time object detection, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Rosebrock, A. (2020), Simple object tracking with opencv.

Sun, Z., G. Bebis, and R. Miller (2004), On-road vehicle detection using optical sensors: A review, in *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No. 04TH8749)*, pp. 585–590, IEEE.

Tensorflow (), tensorflow/models.

Zelic, F. (2020), [tutorial] ocr in python with tesseract, opencv and pytesseract.

Zhang, B., Y. Zhou, and H. Pan (2013), Vehicle classification with confidence by classified vector quantization, *IEEE Intelligent transportation systems magazine*, 5(3), 8–20.