Enhancing Real-Time Object Detection on Low-Powered GPU Surveillance Systems Using YOLOv8 Models with Frame Skipping Approach

Shivanshu Ade[1], Mrs. Rohini Chavan[2]
[1]Vishwakarma Institute of Information Technology, Pune, 411048, India, shivanshu.22011180@viit.ac.in
[2]Vishwakarma Institute of Information Technology, 411048, Pune, India, rohini.chavan@viit.ac.in

ABSTRACT

Object detection has been a pivotal area of exploration within computer vision. While various methodologies were developed, many needed help, to attain real-time processing or were on the brink of achieving it. The advent of high-powered GPUs marked a significant advancement, enabling some systems to achieve real-time object detection. However, these systems were cost-intensive and consumed substantial power. Our study addresses these challenges by focusing on object detection systems equipped with low-powered GPUs, such as the Nvidia MX450 used in our research. Central to our approach is frame skipping, where a predetermined number of frames are bypassed during object detection processing. This strategy reduces the computational burden by excluding frames that contribute minimally to object detection results, thereby optimizing resource utilization. We applied this frame-skipping approach to trained YOLOv8 models, specifically YOLOv8n and YOLOv8m, for conducting object detection on a 1-minute and 56-second video. The implementation of our approach yielded real-time object detection performance, with an execution time of 116.01 seconds for the YOLOv8m model and 116.73 seconds for the YOLOv8n model.

Keywords: Frame-Skipping Approach, Real-Time Object Detection, YOLOv8.

1. INTRODUCTION

Within the domain of intelligent object-detection systems, the challenge of achieving real-time object detection on low-powered GPUs has long been a hurdle. The constraints imposed by cost considerations often lead to the deployment of systems with limited computational capabilities, making it impractical to leverage high-powered GPUs for real-time analysis. This limitation is particularly pronounced in scenarios where the need for continuous monitoring and analysis demands efficient and timely detection of objects.

To address this challenge, our research focuses on the development of a novel approach utilizing the YOLOv8 model, augmented with frame-skipping techniques, to enable real-time object detection on low-powered GPU surveillance systems. The essence of our methodology lies in the strategic analysis of video frames, wherein a specific number of pre-determined frames are skipped during object detection processes. This deliberate frame skipping not only optimizes computational resources, by excluding video frames that contribute minimally to object detection results but also ensures that real-time detection tasks can be performed even on systems with limited GPU capabilities.

By adopting this approach, we aim to bridge the gap between cost-effective surveillance solutions and the demand for real-time object detection in diverse operational environments. Our work contributes to the advancement of surveillance technology by offering a practical and efficient solution that leverages existing hardware infrastructures without compromising detection speed or accuracy.

In this paper, we present a comprehensive analysis of our YOLOv8 model, pre-trained and trained on the KITTI dataset, with frame skipping showcasing its efficacy in real-world scenarios.

We evaluate its performance metrics including its execution speed, classification accuracy, mean average precision (mAP) of detection, and resource utilization, to demonstrate its suitability for low-powered GPU surveillance systems. Additionally, we compare our approach with existing systems/methods to highlight its advantages and contribution to real-time object detection in surveillance applications. In section 2, we conduct a literature survey of the previous works. In section 3, we explain the flow of our approach and implement it. In Section 4, we show and analyze the results of our approach. In Section 5, we conclude our research and present the scope for future research.

2. LITERATURE SURVEY

Some of the applications where CNN has significantly advanced are all related to computer vision like object detection [14], segmentation [15], face detection [12], image classification, and object tracking [16][7]. For 3D image registration, conventional techniques like Scale-Invariant Feature Transform (SIFT) have been utilized, such as Fast Point Feature Histograms (FPFH) [8] and Normal Aligned Radial Features (NARF) [9]. Moreover, object detection has made use of classification techniques like support vector machines and nearest-neighbor methods [10][11]. In contrast to more conventional computer vision techniques like SIFT-based methods, recent advancements in CNNs, as demonstrated by the YOLO model series, have demonstrated notable improvements [6][4][5]. With YOLO reaching up to 44-46 fps on an Nvidia Titan-X GPU, these CNN-based techniques offer faster processing speeds [13]. Although CNNs need to be trained, they are flexible and can be used to find solutions for a wide range of problems, frequently requiring less hardware. Due to high power consumption, hardware implementations lag behind

software despite their speed advantages [17]. Efficient methods are required to fully utilize CNNs for low-power applications.

To close this gap, research is being done in areas like real-time processing with GPUs and multi-core architectures [18]. Using a transfer learning backbone [1], highlighted a new GPU-efficient model for multi-scale feature interaction that enhances the accuracy and speed of object detection models on edge GPU devices. The model is optimized in a general way rather than being customized for each device, even though the results are satisfactory.

A method for optimizing YOLO using OpenCV for CPU-based computer object detection in real-time [2] was presented. On a range of non-GPU systems, the method demonstrates object detection with confidence levels between 80% and 99%, achieving a frame rate of 10-16 fps. 31.05% is the mean average precision (mAP) that was attained. However, on high-resolution videos with frame rates usually at 24, 30, or 60 frames per second, this method might not produce real-time results. It also emphasizes a lighter version of the YOLO model, like the nano or small weighted models, which may lose accuracy in comparison to heavier-weighted YOLO models, like the medium, large, or extra-large weighted models.

Samira, Rahem, and Anil presented a model called MvcYOLO [3] which aimed to optimize the object detection procedure by utilizing YOLO within the Darknet framework. Handling object coordinates differently, resolves problems like needless computations and conversions. By generating new coordinates directly from YOLO's supplied coordinates, MvcYOLO reduces the need for extra conversions and boosts the productivity of drawing object regions on RGB images. By using this method, YOLOv3-based object detection performs better overall and with less computational overhead. Nevertheless, the method does not work on systems with a GPU with less power because it requires a powerful GPU, the Nvidia Quadro p5000.

There hasn't been any work that has tried to solve this issue of achieving real-time object detection on low-powered GPU systems by taking into consideration the negligible difference between two consecutive video frames. In this paper, we try to optimize object detection using the frame-skipping approach. We have performed this research to evaluate our proposed approach.

3. OBJECT DETECTION USING FRAME SKIPPING APPROACH

We have proposed a unique object detection approach in this research that uses the frame-skipping technique., which skips a specific pre-determined number of frames during the detection process being performed on a video. First, we collect the dataset (KITTI dataset) and train the pre-trained YOLOv8 models (YOLOv8n and YOLOv8m). Then we import these trained YOLOv8 models, to perform the frame-skipping methods using these models. The kind of YOLOv8 model we are utilizing for the detection procedure dictates how many frames are skipped in a single jump. To ascertain how near our approach is to real-time, we finally test it on a video and quantify the execution time.

3.1. KITTI Dataset

Research in computer vision and autonomous driving makes extensive use of the 2D KITTI dataset. Real-world views of urban surroundings are presented using images taken from a moving vehicle that is fitted with cameras and sensors. With its annotated item bounding boxes for vehicles, pedestrians, cyclists, etc, the dataset is useful for applications like object detection, tracking, and scene comprehension. Though companion KITTI datasets also contain such information, the "2D" part of the dataset alludes to its primary focus on 2D images and annotations rather than full 3D scene reconstruction or depth information.

3.2. YOLOv8 Model

YOLOv8, the most recent model in the YOLO model lineage of ultralytics, gets its name from the phrase "You Only Look Once." Its name accurately conveys its exceptional ability to accurately predict every object in an image with just one forward pass. Large-scale datasets like COCO and ImageNet are used for pre-training these YOLO models. With this dual training approach, they can quickly adapt and pick up new classes with relative ease in addition to excelling at making accurate predictions for the classes they are initially trained on.

In our research, we have imported two different pre-trained YOLOv8 models from the ultralytics library, namely 'YOLOv8n' and 'YOLOv8m'. The 'n' in 'YOLOv8n' stands for 'nano'. This model is the fastest variant of the YOLOv8 models, however with the least accuracy when it comes to object detection and classification. On the other hand, the 'm' in 'YOLOv8m' stands for 'medium'. This model is slower as compared to the 'nano' variant but more accurate when it comes to detection and classification. These models are trained using the KITTI dataset, each for 200 epochs with a batch size of 16. Both of these models can perform real-time under powerful GPUs like the Nvidia's RTX 30 and 40 series, but when it comes to low-powered GPUs like the Nvidia's MX450, which we have used for this research purpose, only the 'nano' variant of the YOLOv8 model can perform object detection close to real-time whereas the 'medium' variant of the YOLOv8 model is nowhere near real-time. To solve this issue for low-powered GPUs and obtain real-time object detection using our frame-skipping method, we have used these two variants of the YOLOv8 model. Training these two models on the KITTI dataset we get these results as shown in Figure 1. a and Figure 1. b.

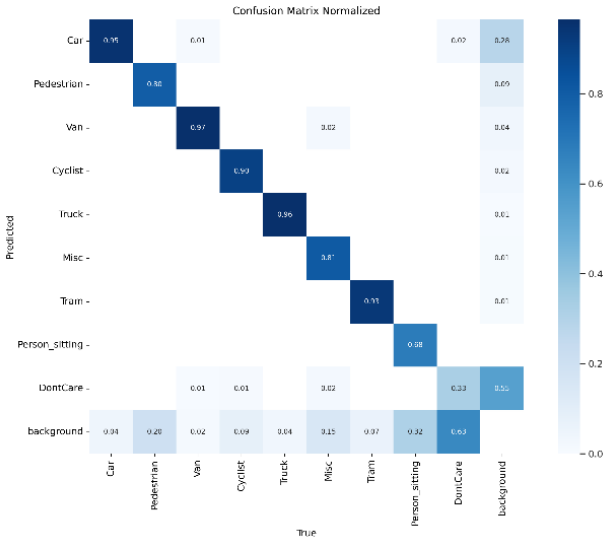

Figure 1. a. Confusion matrix (Normalized) of YOLOv8n model

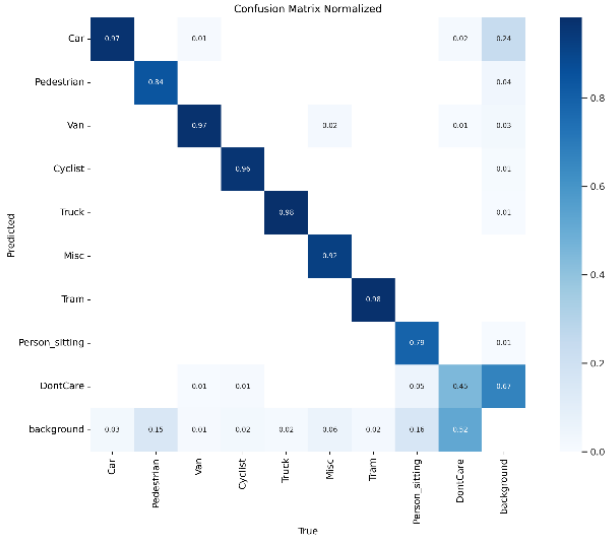

Figure 1. b. Confusion matrix (normalized) of YOLOv8m model

From the confusion matrix in Figure 1.a. and 1.b., we were able to get a classification accuracy of 73.4% for the YOLOv8n (nano variant) model and 78.69% for the YOLOv8m (medium variant) model. The respective Accuracy and the Mean Average Precision (mAP50 and mAP50-95) of both models are shown in Table 1 below.

Table 1. Accuracy and mAP values table of trained YOLOv8 models

Sr. No	Model (Variant)	Accuracy	mAP50	mAP50-95
1.	YOLOv8n	73.4%	0.835	0.62
2.	YOLOv8m	78.69	0.879	0.678

3.3. Frame Skipping Approach

Frames in a video refer to the individual images that, when played in a sequence at a rapid rate, create the illusion of motion. The frame rate of a video is measured in frames per second (FPS), indicating how many frames are displayed in one second. Standard video frame rates include 24 FPS (common in movies), 30 FPS (common in TV broadcasts), 60 FPS (common in gaming and high-definition videos), and higher frame rates used in specialized applications. The difference between two consecutive frames is known as frame difference and for a video with a frame rate ranging from 24 to 60 FPS the difference between two consecutive frames is almost negligible as shown below in Figure 2.

Figure 2. Frame Difference between 6 consecutive frames of a 60 FPS video

In our research, we have taken into consideration this negligible difference between multiple consecutive frames to implement our Frame Skipping Approach. The frame-skipping approach is the process of skipping a specific predetermined number of frames during object detection, excluding frames that contribute minimally to our object detection results, thereby optimizing resource utilization. This helps reduce the number of frames of a video while performing object detection on it, so our detection models can provide much faster results without the need for much higher-powered GPUs, thus reducing the cost and power consumption in real-world applications.

3.4. Flow of Implementation

The flow of an implementation of our object detection using the frame-skipping method can be shown in Figure 3.

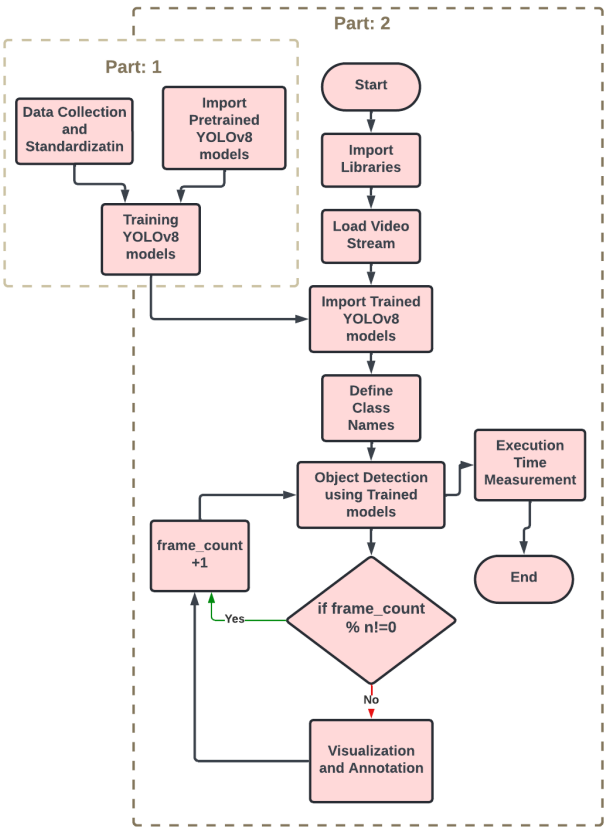

Figure 3. Flow chart of Object Detection Using Frame Skipping Approach

Our Approach is divided into two parts-

3.4.1. Part 1:

a. Data Collection:

For our research study, we have employed the KITTI dataset, comprising 7480 training images paired with their respective label files. Additionally, an equivalent number of testing images are available, albeit without accompanying label files. We have partitioned the training dataset into two subsets: 7000 images and label files designated for training purposes and stored in the training folder, and the remaining 480 images and label files allocated for validation purposes and stored in the validation folder. To manage these datasets effectively, we are creating a YAML file to store the paths leading to our training and validation dataset folders.

b. Importing pre-trained YOLOv8 models:

We have incorporated two pretrained YOLOv8 models, specifically YOLOv8n (nano variant) and YOLOv8m (medium variant), from the Ultralytics package into our analysis. Our selection criteria were based on their suitability for real-time object detection on low-powered GPU systems like the Nvidia MX450. The YOLOv8n model, being optimized for efficiency, aligns well with our system's hardware constraints, making it a suitable candidate for real-time processing. Conversely, the YOLOv8m model offers enhanced accuracy in object detection and classification but falls short in terms of real-time detection speed.

Our primary objective is the performance enhancement of the YOLOv8m (medium variant) model to surpass the YOLOv8n (nano variant) model in terms of execution time. This optimization entails enhancing the medium variant's efficiency without compromising its classification accuracy, thereby achieving a balance between speed and precision in object detection tasks.

c. Training YOLOv8 models on Collected Data:

After importing these models, we pass a YAML file containing the paths to the training and validation folders of the KITTI dataset to both models for training. The training parameters are set at 200 epochs with a batch size of 16. Our primary objective in training both models is to reduce the number of classes for classification, focusing on our specific purpose to achieve higher accuracy, precision, and speed. By limiting the classes, the model can concentrate more on learning the unique features and characteristics of each class, leading to better discrimination. Furthermore, a reduced number of classes is likely to result in faster inference times due to fewer computations, leading to quicker processing per image or video frame. After training these models we performed object detection on a video, without using a frame-skipping approach to check whether reducing the number of classes helps the model run faster. The results can be summarized in Table 2.

Table 2. Time of Execution and Accuracy of YOLOv8 trained and pre-trained models

Sr. No.	Model	Accuracy	Time of Execution (Avg.)
1.	YOLOv8n (pre-trained)	—	124.56 sec
2.	YOLOv8n (trained)	73.4%	80.65 sec
3.	YOLOv8m (pre-trained)	—	185.96 sec
4.	YOLOv8m (trained)	78.69%	144.15 sec

As observed from Table 2, it can be inferred that a model trained to detect and classify a relatively lower number of classes operates much faster than the default models. The comparison does not include the accuracy values for the pre-trained default models, as these models were trained on more accurate data and for a significantly larger number of epochs. Therefore, direct comparison with our trained models, which were trained for a relatively lower number of epochs, is not feasible. However, this limitation can be addressed by training the pre-trained models for a larger number of epochs and with more accurate data. Despite this option, our primary focus remains on achieving real-time detection, and we will adhere to this main objective.

3.4.2. Part 2:

a. Importing Libraries:

Our algorithm begins by importing all the essential Python packages, including OpenCV, math, torch, Ultralytics, cvzone, and time. These packages provide the necessary functionalities for image processing, mathematical operations, deep learning, object detection, visualization, and timing measurements, facilitating the implementation of our algorithm for real-time object detection and analysis.

b. Load Video Stream:

Then, we proceed to load a video using the OpenCV library's functionality. This video is used as the input for the analysis of our proposed technique, leveraging the capabilities of our trained YOLOv8 models. The video that we have loaded is a 1:56 minute (116 seconds) long video of a corner camera view of street intersections as shown in Figure 4.

Figure 4. Images of two different frames of the loaded video

c. Import Trained YOLOv8 models:

The process involves importing the 'best.pt' weights of the trained YOLOv8n and YOLOv8m models for our research. These weights are the most accurate versions of these trained models and contribute significantly to improved classification during the object detection process.

d. Define Class Names:

Class names for objects of interest are defined to label the detected objects correctly. In our dataset, we are primarily concerned with fewer objects, so we have categorized them into 9 classes: 'Car', 'Pedestrian', 'Van', 'Cyclist', 'Truck', 'Misc', 'Tram', 'Person_sitting', and 'DontCare'. Our YOLOv8 models are trained to classify objects into these 9 categories, ensuring accurate and meaningful labeling during the object detection process.

e. Object Detection Using Trained YOLOv8 models:

This phase represents the real-time object detection process using the frame-skipping approach. A while loop is used to continuously read frames from the loaded video in the video capture object. Each frame is then processed for object detection using the imported trained YOLOv8m (medium variant) model. This processing skips a specific number of predefined frames to optimize computational resources, focusing on the other non-skipped frames. For our research purpose, we will be considering two different cases of skipping every two frames and performing object detection on every 3rd frame and the other case of skipping three frames and performing object detection on every 4th frame which comes after that. A note about the skipping procedure needs to be made that there are two types of skipping methods. Firstly we check each frame count and skip it only if it satisfies the frame skipping procedure, secondly, rather than checking each frame, the algorithm straight forward jumps to the frame which satisfies the frame count condition. The problem with the latter approach is that it creates discrete motion in object movements inside the video rather than continuous and smoother movement, which may result in a video with a lagging effect. So that is why our method involves processing each frame and passing only those frames for object detection which satisfies the frame-skipping condition. As stated earlier we will be experimenting with two types of test cases on the YOLOv8m model.

For case 1: The process checks each frame count and if it is not divisible by 3 it skips that frame and iterates to the next frame count and if it is divisible by 3, that frame is passed on to the YOLOv8m model for object detection, where visualized and annotated results are provided.

For case 2: The process checks each frame count and if it is not divisible by 4 it skips that frame and iterates to the next frame count and if it is divisible by 4, that frame is passed on to the YOLOv8m model for object detection, where visualized and annotated results are provided.

During the detection process, the objects being detected are visualized using bounding boxes along with labeling them with the defined class names and confidence scores. The ‘cvzone’ library is utilized for drawing bounding boxes and text on the image. The process continues till the end of the video or until the user presses the ‘q’ key to exit at which point the video capture object is released, windows are closed, and the execution time is printed. Later we also perform object detection using the YOLOv8n (nano variant) model, but with skipping every alternate frame to bring the detection speed of this model close to real-time. This will help us in comparing the performance of our YOLOv8m model, using the frame skipping approach, with the YOLOv8n model.

f. Execution Time Measurement:

The Execution Time measurement will form the most important parameter for testing our frame-skipping approach since our main purpose is to achieve real-time object detection. This parameter is essential because it directly reflects the computational efficiency and speed of our algorithm. By measuring the time taken for each frame to undergo object detection processing, we can evaluate how well our model performs in real-time scenarios. A lower execution time indicates faster object detection, which is crucial for applications requiring timely and accurate results, such as surveillance systems, autonomous vehicles, or video analysis for live events. Therefore, monitoring and optimizing the execution time helps ensure the effectiveness and practicality of our real-time object detection with a frame-skipping approach.

For our research study, we plan to collect 20 individual execution time measurements for performing object detection on the video, which is 1:56 minutes (116 seconds) long, which we have taken into consideration for testing our approach on each frame skipping case as described earlier. Additionally, we will gather execution time data for the YOLOv8n model with and without frame skipping using the same video. These measurements will be averaged to obtain representative values for comparison and analysis. This approach allows us to quantitatively assess the performance differences between different frame-skipping strategies and model configurations.

4. OBSERVATIONS AND RESULTS

Our research encompassed multiple test cases as described previously. The quantitative and visual representations of these test cases are encapsulated in Figure 5 and Table 3, providing a comprehensive overview of our experimentation and results.

Figure 5. Object Detection is performed by our Trained Models.

Table 3. Evaluation Results of YOLOv8 Models using our approach.

Sr. No.	Model used	No. of Frames skipped	Execution Time (Avg.)
1. **Case 1**	YOLOv8m	2	139.21 sec
2. **Case 2**	YOLOv8m	3	116.01 sec
3.	YOLOv8n	1	116.73 sec
4.	YOLOv8n	0	137.35 sec

From the data presented in Table 3, we observed significant improvements in real-time object detection performance using our approach with the YOLOv8m (medium variant) and YOLOv8n models.

- In Case 1, utilizing our frame-skipping approach with the YOLOv8m model, skipping every 2 frames and processing every 3rd frame, resulted in an execution time of 139.21 seconds (about 2 minutes and 19 seconds), indicating that this approach did not achieve real-time object detection.
- In Case 2, where we skipped 3 frames and processed every 4th frame using the YOLOv8m model, real-time object detection was achieved with an execution time of 116.01 seconds (approximately 1 minute and 56 seconds).
- Similarly, with the YOLOv8n model and our frame-skipping approach of skipping every alternate frame, real-time object detection was achieved with an execution time of approximately 116.73 seconds (about 1 minute and 17 seconds).
- In contrast, when performing object detection without our frame-skipping approach, the YOLOv8n model required significantly more time, resulting in an execution time of about 137.35 seconds (approximately 2 minutes and 17 seconds), which is not within the real-time performance range.

These results highlight how well our frame-skipping strategy works to improve the YOLOv8 models' real-time object identification abilities, especially when used in certain frame-skipping settings.

5. CONCLUSION AND FUTURE SCOPE

Based on our experimental findings, we ascertain that our approach facilitates real-time object detection, albeit the optimal frame skipping rate depends on several factors such as the specific YOLO model, GPU specifications, and dataset characteristics. Determining the ideal frame-skipping rate typically involves iterative testing and optimization based on these factors.

Our findings suggest that real-time object detection can be accomplished by surveillance systems with less powerful GPUs without requiring high-end GPUs, leading to reduced power consumption and cost-effective solutions. Notably, the frame-skipping technique does not compromise model accuracy since each considered frame undergoes individual object detection processing.

While our study provides valuable insights, future quantitative research can delve deeper into optimizing the frame-skipping approach. Additionally, exploring higher variants of YOLOv8 models, such as YOLOv8l (large variant) and YOLOv8x (extra-large variant), known for superior classification accuracy, presents promising avenues for further investigation and refinement.

6. DECLARATIONS

Competing Interests: We declare no competing interests regarding the frame-skipping technique discussed in this research paper. Our focus is on optimizing computational resources and achieving real-time object detection, without any conflicting interests influencing our approach or findings.

Authors' contribution: Shivanshu Ade- Data collection, investigation, methodology, implementation, writing-original draft; Mrs. Rohini Chavan- Investigation, review and editing.

Funding: not applicable

Availability of data and materials: The KITTI Dataset used for this research can be accessed through the official website- https://www.cvlibs.net/datasets/kitti/.

7. REFERENCES

[1] Prakhar Ganesh, Yao Chen, Yin Yang, Deming Chen, Marianne Winslett, "YOLO-ReT: Towards High Accuracy Real-time Object Detection," Computer Vision Foundation (CVF), 2022.

[2] Md. Bahar Ullah, "CPU Based YOLO: A Real-Time Object Detection Algorithm," IEEE Region 10 Symposium (TENSYMP), 5-7 June 2020.

[3] Samira Karimi Mansoub, Rahem Abri, Anil Hakan Yarici, "Concurrent Real-Time Object Detection on Multiple Live Streams Using Optimization CPU and GPU Resources in YOLOv3," The Fourth International Conference on Advances in Signal, Image and Video Processing, 2019.

[4] J. Redmon, and A. Farhadi, "YOLO9000: better, faster, stronger," Computer Vision and Pattern Recognition (CVPR), 2017.

[5] J. Redmon, and A. Farhadi, "YOLOv3: An Incremental Improvement," Technical report, 2018.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[7] S. Veeraraghavan, and A. Chellappa, "Object detection, tracking and recognition for multiple smart cameras." Proc. IEEE, 2008.

[8] R. Usu, R. B. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'09), 2009.

[9] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, "NARF: 3D range image features for object recognition," IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010.

[10] J. Liebelt, C. Schmid, and K. Schertler, "Viewpoint independent object class detection using 3D feature maps," IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08), 2008.

[11] M. R. Lata, and M. Y. Alvino, "FPGA implementation of support vector machines for 3D object identification," In Proceedings of the 19th International Conference on Artificial Neural Networks: Part I (ICANN'09), 2009.

[12] D. Han, J. Choi, J. Cho, and D. Kwak, "Design and VLSI implementation of a high-performance face-detection engine for mobile applications," IEEE International Conference on Consumer Electronics, 2011.

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," IEEE, 1998.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[15] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017.

[16] E. Gundogdu, and A. A. Alatan, "Good features to correlate for visual tracking," IEEE Transactions on Image Processing, 2018.

[17] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," IEEE Micro, 2011.

[18] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, 2017.

[19] Takuya Fukagai, Kyosuke Maeda, Satoshi Tanabe, Koichi Shirahata, Yasumoto Tomita, Atsushi Ike, Akira Nakagawa, "SPEED-UP OF OBJECT DETECTION NEURAL NETWORK WITH GPU," ICIP, 2018.

[20] Xun Yin, Li Chen, Xiaoyun Zhang, Zhiyong Gao, "Object Detection Implementation and Optimization on Embedded GPU System," IEEE, 2018.

[21] Mohammad Javad Shafiee, Brendan Chywl, Francis Li, Alexander Wong, "Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video," https://arxiv.org/abs/1709.05943v1, 18 Sep 2017.

[22] Dweepna Garg, Parth Goel, Shamil Pandya, Amit Ganatra, Ketan Kotecha, "A Deep Learning Approach for Face Detection using YOLO".

[23] Sakshi Gupta, Dr. T. Uma Devi, "YOLOv2 based Real-Time Object Detection," International Journal of Computer Science Trends and Technology (IJCST), May-June 2020.