International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Optimizing Real-Time Object Detection on Edge Devices: A Transfer Learning Approach

Harshad Lokhande[1*], Sanjay Ganorkar[2*]

Submitted: 03/02/2024 **Revised**: 11/03/2024 **Accepted**: 17/03/2024

Abstract. The issue of object detection in remote surveillance using edge devices presents a complex scenario, largely as a result of the limitations inherent in edge computing settings and the requirements for instantaneous data processing. Current video surveillance systems exhibit proficient video capture functionalities; however, data analysis at the server level is impeded by constraints in transmission power and the availability of cloud computing resources. Consequently, Internet of Things (IoT) devices are primarily relegated to the role of data acquisition. Our study proposes a novel fusion of ResNet18, K-Means clustering, and int8 quantization over tinyML, compressing the model to <100KB and enabling sub-1mW consumption on edge devices. This methodology extends the viability of deploying sophisticated machine-learning models on microcontrollers powered by coin cells, broadening their applicability in various settings for object detection. Employing int8 quantization, our model attains a notable improvement in latency by 45%, coupled with a 70% reduction in RAM consumption and a 65% decrease in flash storage. This research delves into the significance of optimization in the process of choosing latency-efficient deep neural network (DNN) models for various edge computing configurations, emphasizing the delicate balance between hardware capabilities and optimization approaches. Subsequent research efforts will concentrate on refining quantization algorithms to further mitigate the precision differential in computational models.

Keywords: K-Means, DNN , IoT, Video Surveillance, edge computing, tinyML

1. Introduction

The problem of object detection in remote surveillance on edge devices is multifaceted, primarily due to the constraints of edge computing environments and the demands of real-time processing. Edge devices often suffer from limited computational resources and power constraints, which pose significant challenges for deploying complex object detection models like Fast R-CNN and Single Shot Detector, as well as more advanced models such as YOLOv4 (Choi Keong-Hun and Ha Jong-Eun., 2022; Li Yong chang *et* al., 2023). These limitations necessitate the exploration of techniques like model compression, quantization, and the use of hardware accelerators to enhance performance and efficiency. In the context of smart cities, where real-time video surveillance is crucial, the reliance on centralized cloud frameworks for processing large volumes of media data introduces latency and dependency on stable internet connectivity, which is not always available (A Krishna *et* al., 2021; Ng. C.H., *et* al. , 2022).

A distributed real-time object detection framework that leverages edge-cloud collaboration has been proposed to address these issues, enabling faster processing and responsiveness by bringing computation closer to the data source (Che Rong *et* al., 2019; B. R. Solunke and S. R. Gengaje, 2023). Moreover, the deployment of object detection models on edge devices for applications like autonomous driving and video surveillance requires not only computational efficiency but also privacy preservation. Distributed cloud-edge models, such as a modified version of YOLOv3 based on split learning, have been developed to meet these computational and privacy requirements by sharing only part of the model while keeping data local (Ayoub Benali Amjoud., 2023). Performance degradation due to the inability of edge devices to handle the computational load of well-trained DNN models is another challenge. This has led to the exploration of multi-model multi-device detection parallelism and model lightweight techniques like PG-YOLO and Edge-YOLO, which aim to improve detection speed and accuracy on edge devices with limited computing power (S Shivappriya *et* al., 2021; Karim Shahid *et* al., 2020). Additionally, offloading computation-intensive workloads to remote edge servers and employing device-aware DNN offloading decision algorithms can further optimize resource utilization and enhance object detection performance on mobile edge devices (Delia *et* al., 2018). Traditional methods of object detection, while foundational in the development of computer vision, often required manual feature extraction and were limited by their inability to adapt and learn from

[1*,2]*Department of E&TC Engineering, Sinhgad College of Engineering, SPPU, Pune*
***Corresponding Author:** Harshad Lokhande*
**Department of E&TC Engineering, Sinhgad College of Engineering, SPPU, Pune*

new data autonomously (H N Lokhande and S R Ganorkar, 2020).

These methods faced significant challenges, particularly in complex and dynamic environments such as remote surveillance on edge devices, where the variability of

detection systems (B Solunke *et* al., 2023). Deep learning-based object detection systems have shown superior performance in various applications, including remote surveillance, by effectively detecting and classifying objects in real-time, even in challenging conditions (M Gayathri *et* al., 2023). These systems benefit from the self-learning capabilities of deep learning, which simplifies operations and enhances efficiency by automatically extracting complex features (Khedkar Mrunal, 2021).

However, deploying these advanced models on edge devices for remote surveillance poses challenges due to the limited computing resources and power budgets of such devices (Zhang Lei., Wang Yuehuan and Huo Yang, 2021; Gautam Aayushi, 2021). To address these challenges, recent research has focused on optimizing deep learning models for edge devices without significantly compromising accuracy (Chugh Ujjawal, 2021). Techniques such as domain-based transfer learning have been proposed to enable real-time computation on resource-constrained devices by reducing model complexity and focusing on domain-specific objects (M Sonali, Diware, A Shweta and Iskande, 2012). Additionally, optimization techniques like pruning, quantization, and the use of lightweight neural networks have been explored to achieve real-time inference capabilities, demonstrating promising results in maintaining high detection performance with minimal precision degradation.

The subsequent section presents a synopsis of compression methodologies for Neural Networks deployed on restricted devices (S Amira, Mahmoud, et al., 2020 ; Aftab, F., Bazai *et* al., 2023). Consequently, when setting weights to zero in a non-systematic manner, it becomes imperative to monitor all such sequences to ensure the subsequent round of pruning. Unsystematic pruning has the potential to achieve a proportion exceeding 95% of zeros, thereby reaping the benefits of quantization. On the other hand, structured pruning, entailing the designation of fixed rows and columns as "zeros", can lead to the removal of up to 90% of the weights within the Alexnet architecture, with a marginal impact on accuracy (Man Yanmao, Li Ming, M Ryan and Gerdes, 2023).

An alternative approach for reducing the dimensions of a model involves the utilization of compression methodologies. The strategy known as Learned objects and conditions could severely impact performance (Avellaneda Diego *et* al., 2023). Deep learning methods, leveraging convolutional neural networks (CNNs), have significantly advanced the field by automating feature extraction and improving the adaptability and accuracy of object

Intermediate Representations (LIT) typically encompasses the acquisition of representations (or features) at intermediary levels within a complex neural network. These acquired features play a pivotal role in the overall performance of the model. Another effective technique for model reduction is the application of Shift attention layers (SHIFT) which involves the transformation of convolutions through pruning (Hadidi Ramyad, et al.,2019; Chen Xin, 2022).

Optimal results are achieved through the combination of ResNet18 and the Distillation process. Conversely, the impact of Pruning with ResNet18 is not particularly significant. Note-worthy is that K-means clustering primarily concentrates on memory, therefore adjusting only the chip voltage can lead to a respectable test accuracy in ResNet18+K-means (EL-Hadad , 2022; Shirpour Mohsen et al.,2023).

The techniques like Haar Cascade in OpenCV are not enough suitable to deploy on Microcontrollers. The training on the Edge device is a difficult task. Hence, Transfer Learning emphasizes the importance of utilizing pruning and incorporating MobileNet SSD v2 alongside transfer learning to improve the model's performance in environments with limited resources and compare with different combinations to identify optimum hyperparameters to get minimum latency and greater accuracy on RaspberryPi. This strategy effectively strikes a balance between computational efficiency and accuracy.

Our study provides a real-time solution to read the video in runtime by analysing the object for object detection on low-power computing devices. The contents covered in this section provide a concise review of the existing systems utilized for object detection on edge devices, showcasing the various optimization techniques employed to discern the disparities between 32-bit and 8-bit quantization modelling (Kantham Laxmi *et* al., 2023). Subsequently, the forthcoming chapter delves into the methodology elucidating the architectural framework and the mathematical analysis pertaining to the delicate balance between model size and accuracy. Moreover, the subsequent findings are carefully compared with the previously mentioned optimization methods to conduct a thorough assessment.

2. Methodology

An edge impulse platform is utilized for the training of the Raspberry Pi model with object detection. The flowchart illustrates a methodical approach for developing and optimizing a deep learning model, utilizing the MobileNetV2 architecture within the TensorFlow framework. As shown in Figure 1, the process commences with the Data Preparation phase, where the dataset is curated and prepared for processing, ensuring its compatibility for model ingestion.

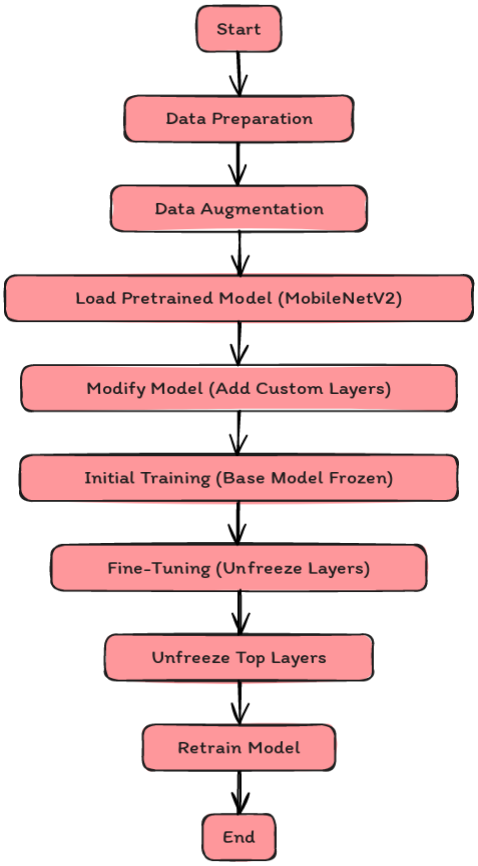

Figure 1. Implementation workflow with Transfer Learning

Following this, the Data Augmentation phase is implemented to improve the model's capacity to generalize from the training data by introducing variations in the dataset. This is accomplished through methods like random flipping, cropping, and brightness adjustments, thereby replicating a wider array of data variances. The workflow advances to the Load Pretrained Model stage, where the MobileNetV2 architecture is instantiated with preloaded weights. This method leverages transfer learning, enabling the model to utilize knowledge obtained from a previously trained context, thereby improving its learning efficiency and efficacy. During the Modify Model stage, the architecture is adjusted by adding custom layers on the MobileNetV2 base, tailoring the model to the specific task at hand.

This customization is crucial in refining the model's output to meet the desired objectives. The Initial Training phase encompasses training the customized model while maintaining the base MobileNetV2 layers frozen, directing the learning process towards the newly added layers. This step is essential for the initial adaptation to the task-specific features without disrupting the pre-learned representations in the base model. The workflow then moves to the Fine-Tuning phase, where specific layers from the base model are unfrozen in the Unfreeze Top Layers step, enabling the fine-tuning of deeper representations within the model. This phase plays a vital role in enhancing the model's performance, aligning it more closely with the task-specific intricacies. In the Retrain Model stage, the model undergoes additional training in this new configuration, allowing the unfrozen layers to adjust their weights in synchronization with the custom layers added earlier. The progression of data from the input (specifically the first Conv2D layer) throughout the network until it reaches the output (which is the Dense layer with softmax activation). The utilization of the Adam optimizer facilitates the adjustment of the network weights by the computed gradients. Conv2D layer is represented with ReLU (Rectified Linear Unit) as the activation function (see equation 1).

$$Y = ReLU\ (W * X + b)\ \mathrm{Y} \quad (1)$$

$$\theta = \theta - \eta \cdot \nabla\theta J\ (\theta;\ \mathrm{X},\ \mathrm{Y}\ ) \quad (2)$$

(see equation 2), the Adam optimizer updates the weights θ based on the gradient ∇θJ of the loss function J concerning the weights considering ηas the learning rate.

The selection of the number of convolution filters as 8 and 16 in Conv2D layers is based on the consideration that early layers are responsible for capturing basic, low-level features such as edges and textures, whereas deeper layers capture more complex, high-level features (H N, Lokhande, S R Ganorkar, 2021).

The utilization of a 3x3 kernel size enables the capturing of local features with a reduced number of parameters, thereby decreasing the computational load. This decision has been experimentally validated in various successful architectures, such as VGG and ResNet (Mellit Adel, 2023 ; Ngoc Son and S. N. Truong, 2020) . The application of max pooling with a 2x2 size and a stride of 2 reduces the spatial dimensions of the feature maps by a factor of 2. This downsampling process aids in making the representation smaller and more manageable, while also introducing a level of translation invariance to the network. During training, 25% of the nodes are randomly dropped out, denoting a dropout rate of 0.25. It is noteworthy that this rate is somewhat arbitrary and is typically determined through experimentation and validation performance (Kumar Ravi and Jatoth, 2023; Walid Ali *et* al., 2021)

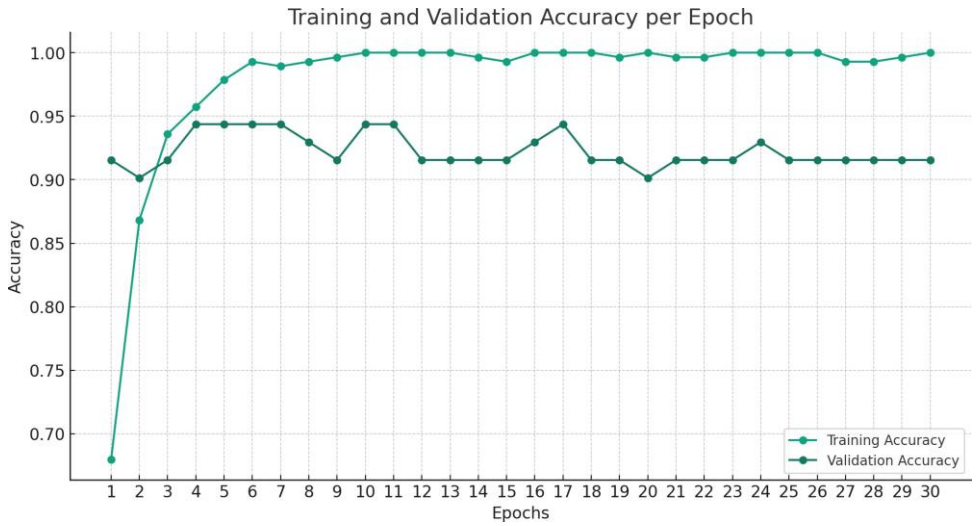

Figure 2. Training and Validation Accuracy

Figure 2 presents the progression of accuracy in both training and validation throughout 30 epochs, partitioned into two distinct phases: the initial training phase, which encompasses epochs 1 to 20, and the subsequent fine-tuning phase, which comprises epochs 21 to 30. Throughout the initial training phase, the accuracy of the training process exhibits a consistent upward trend, culminating in near-perfect accuracy by the 10th epoch. Concurrently, the accuracy of the validation process also experiences improvement, albeit with some intermittent fluctuations.

Table 1 Hyperparameter Configurations and Performance Metrics

Trial ID	Input Dimensions	DSP Blocks	Dense Neurons	Dropout	Est. Latency in ms
1	160 x 160	RGB	64	0.1	20245.3
2	96 x 96	Grayscale	64	0.1	34.52
3	96 x 96	RGB	64	0.5	1207.44
4	96 x 96	RGB	64	0.25	95.88
5	160 x 160	Grayscale	64	0.1	90.01
6	96 x 96	RGB	16	0.1	12.68
7	96 x 96	RGB	64	0.005	15.44

The table 1 provides a detailed overview of the hyperparameter configurations tested during the training of the model, along with the estimated latency, which serves as metrics for evaluating the performance and

complexity of each configuration. Each row corresponds to a specific trial with its respective hyperparameter configuration and performance metrics. The trial ID 6, gave optimized results with 12.68 ms on-device inferencing latency.

3. Results and Discussion

To train and test the transfer learning model, the Face Mask Dataset is used with variations in data by applying Data Augmentation. The comparison shown in Tables 2, demonstrates quantized as well as unoptimized models to exhibit identical latency for image processing (1 ms). To compare these models based on latency, RAM usage, flash storage, and accuracy. Let's define the following variables:

For the quantized (int8) model:

- Let L_q be the latency, which is 13 ms for the total process.
- Let $R_q$be the RAM usage, which is 721.4K.
- Let F_q be the flash storage, which is 626.5K.
- Let $A_q$be the accuracy, which is 83.87%.

For the unoptimized (float32) model:

- Let L_u be the latency, which is 23 ms for the total process.
- Let R_u be the RAM usage, which is 2.4M.
- Let F_u be the flash storage, which is 1.8M.
- Let $A_u$be the accuracy, which is 88.17%.

Table 2 Comparison between latency between int8 and unoptimized int32 Model

	Quantized int8 Model			Unoptimized int32 Model		
	Image	Transfer Learning	Total Time	Image	Transfer Learning	Total Time
Latency	1 ms	12 ms	13 ms	1 ms	22 ms	23 ms
RAM	4.0 K	721.4 K	721.4 K	4.0 K	2.4 M	2.4 M
Flash Accuracy	-	626.5 K	83.87 %	-	1.8 M	88.17 %

The relative difference in latency between the two models can be expressed as ΔL*(see equation 3)*,

$$\Delta L = \frac{L_u - L_q}{L_u} \times 100\ \% \qquad (3)$$

equation 4 is the relative difference in accuracy is ΔA:

$$\Delta A = \frac{A_u - A_q}{A_u} \times 100 \qquad (4)$$

To analyze the trade-off between model size and accuracy, we can formulate *equation* 5 as a utility function U that takes into account both the efficiency (in terms of size and speed) and the accuracy.

$$U = \alpha \times \frac{1}{L_q + R_q + F_q} + (1 - \alpha) \times A_q \qquad (5)$$

where α is a weighting factor that determines the importance of efficiency versus accuracy.

- When α is close to 1, the utility function places more emphasis on efficiency, prioritizing lower latency, RAM usage, and flash storage requirements.
- When α is close to 0, the utility function places more emphasis on accuracy, prioritizing the model's performance in terms of its prediction accuracy.

However, in the case of transfer learning, the quantized model showcases a significantly lower latency (12 ms compared to 22 ms). This indicates that the quantized model is indeed faster, a crucial factor in real-time applications. In addition, the quantized model utilizes significantly less RAM (721.4K) in comparison to the unoptimized model (2.4M). This reduced memory footprint makes the quantized model more suitable for deployment on devices with limited RAM, such as mobile phones or embedded systems.

Table 3 Comparison between Inferencing on Edge Computing Devices

Edge Computing Device	Image	Transfer Learning	Total
Arduino Nano 33 BLE Sense	11 ms	1781 ms	1792 ms
Raspberry Pi 4	1 ms	6 ms	7 ms
Nvidia Jetson Nano	1 ms	22 ms	23 ms
Espressif ESP-EYE (ESP32 240MHz)	15 ms	2334 ms	2349 ms
Arduino Portenta H7	1 ms	139 ms	140 ms

Table 3 presents a comparative analysis between Edge Computing Devices with latency by Quantized (int8) optimization with Tensorflow lite in DNN trained on its own custom dataset. It gives a comparison for selecting the optimized choice of hardware by a specific optimization method (e.g., Quantized (int8)), and the corresponding latencies for image processing, transfer learning, and the overall combined latency. The Arduino Nano 33 BLE Sense (Cortex-M4F 64MHz) displays a total latency of 1,792 ms, primarily stemming from transfer learning activities. Conversely, more robust hardware such as the Raspberry Pi 4 and Nvidia Jetson Nano showcase considerably lower total latencies of 7 ms and 23 ms respectively, when employing quantized (int8) optimizations.

Although there is a drop in accuracy when quantizing the model (from 88.17% to 83.87%), it is often considered acceptable due to the advantages in latency, RAM, and flash storage usage. The decision to embrace this trade-off depends on the specific requirements of the application.

Table 4 Confusion matrix (on validation set)

	Masked	Unmasked
Masked	86.7 %	13.3 %
Unmasked	3.8 %	96.2 %
F1 Score	0.92	0.88

In Table 4, the Confusion matrix shows that the model attained a notable level of accuracy, specifically 90.1% when evaluated on the validation set. This observation serves as an indicator of the model's overall commendable performance. Furthermore, the model exhibits a low error rate in its predictions, as evidenced by a validation loss of 0.23. The model's proficiency in predictive capabilities is substantiated by a thorough examination of the confusion matrix. The F1 scores also reflect a noteworthy performance for both classes, with a score of 0.92 for 'Masked' and 0.88 for 'Unmasked'.

Conclusion

The int8 quantized model exhibits superiority in real-time applications, edge computing, and mobile deployments, catering to scenarios with limited memory, storage, and processing capacities. These improvements are accompanied by a marginal decline in accuracy of 4.3 percentage points, a compromise considered justifiable given the considerable efficiency gains and the potential for utilization in settings with strict limitations on resources. The int8 quantized model demonstrates superiority in real-time scenarios, edge computing, and mobile implementations, where constraints on memory, storage, and processing capabilities play a pivotal role. Despite the minor decrease in model accuracy, the commendable F1 scores of 0.92 for the 'Masked' category and 0.88 for the 'Un- masked' category indicate a well-maintained equilibrium between precision and recall, highlighting the model's pragmatic feasibility.

The outcomes emphasize the practicality of integrating int8 quantization in situations that call for an optimal combination of performance and resource effectiveness on Edge Devices like Raspberry Pi. Nevertheless, the noted accuracy reduction and increased latency underscore a crucial area for future investigation if the size of the image is more than 160 x 160 with RGB. Going forward, research efforts should focus on enhancing quantization methodologies to further alleviate accuracy compromises while continually advancing computational efficiency.

Acknowledgement

We would like to convey our profound appreciation to the Research Center at Sinhgad College of Engineering for their exceptionally generous provision of hardware and software support, bestowed under the esteemed auspices of Savitribai Phule Pune University. It is paramount to acknowledge that this invaluable support has played an indispensable role in facilitating the successful execution of our research endeavours. Moreover, we are obliged to express our deepest appreciation to MIT Art Design and Technology University for their invaluable assistance in the comprehensive collection of data. The contributions rendered by their esteemed personnel and abundant resources have proven to be pivotal in augmenting the depth and breadth of our study. It is without a doubt that our research has greatly thrived due to the remarkable collaboration and abundant resources provided by both institutions and for this, we are profoundly grateful.

References

[1] Choi, K.-H. and Ha, J.-E., 2022. Results Comparison of Applying Object Detection and Object Segmentation Methods on Visual Surveillance Dataset.

[2] Li, Y., Jia, J., Zuo, Y. and Zhu, W., 2023. Effective out-of-distribution Detection for TinyML. Available at: https://doi.org/10.1109/icassp49357.2023.10094746.

[3] Krishna, A., Pendkar, N., Kasar, S., Mahind, U. and Desai, S., 2021. Advanced Video Surveillance System. In: 2021 3rd International Conference on Signal Processing and Communication (ICPSC), IEEE.

[4] Che, R., Wang, L., Wang, Y., Lin, Q. et al., 2019. Research on Intelligent Video Surveillance System in Remote Area Based on NB-IoT. In: Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence. ACM.

[5] Solunke, B.R. and Gengaje, S.R., 2023. Traditional and Deep Learning based Object Detection Methods. In: International Conference on Emerging Smart Computing and Informatics (ESCI). Available at: https://doi.org/10.1109/esci56872.2023.10099639.

[6] Benali, A. and Amrouch, M., 2023. Object Detection using Deep Learning, CNNs and Vision Transformers: A Review. IEEE Access, 11, pp.35479-35516.

[7] Shivappriya, S., Jasmine, M., Priyadarsini, P. et al., 2021. Cascade Object Detection and Remote Sensing Object Detection Method Based on Trainable Activation Function. Remote Sensing.

[8] Shahid, K., Zhang, Y., Yin, S. et al., 2020. A brief review and challenges of object detection in optical remote sensing imagery. Multiagent and Grid Systems.

[9] Rusci, M., Fariselli, M., Capotondi, A. et al., 2020. Leveraging Automated Mixed-Low-Precision Quantization for Tiny Edge Microcontrollers. Communications in Computer and Information Science, 1325, pp.296-308.

[10] Dokic, K., Martinovic, M., Mandusic, D. et al., 2020. Inference speed and quantisation of neural networks with TensorFlow Lite for Microcontrollers framework. In: 2020 5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), IEEE, pp.1-6.

[11] Velasco-Montero, D., Jorge, et al., 2018. Performance analysis of real-time DNN inference on Raspberry Pi. Real-Time Image and Video Processing, 2018.

[12] Lokhande, H.N. and Ganorkar, S.R., 2020. Challenges in Scene Interpretation for Video Surveillance. Test-Engineering and Management, 83(2).

[13] Avellaneda, D., Mendez, D. and Fortino, G., 2023. A TinyML Deep Learning Approach for Indoor Tracking of Assets. Sensors, 23(3), 1542. Available at: https://doi.org/10.3390/s23031542.

[14] Solunke, B., Sachin, R. and Gengaje, 2023. A Review on Traditional and Deep Learning based Object Detection Methods. International Conference on Emerging Smart Computing and Informatics (ESCI).

[15] Gayathri, M.D., Lakshmanan, V. and Krishna, 2023. Object Surveillance Detection. International Journal For Science Technology And Engineering. Available at: https://doi.org/10.22214/ijraset.2023.49595.

[16] Khedkar, M., 2021. Wireless Intruder Detection System for Remote Locations. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 12(12), pp.1390-1401. Available at: https://doi.org/10.17762/turcomat.v12i12.7621.

[17] Zhang, L., Wang, Y. and Huo, Y., 2021. Object Detection in High-Resolution Remote Sensing Images Based on a Hard-Example-Mining Network. IEEE Transactions on Geoscience and Remote Sensing.

[18] Gautam, A., Sukhwinder, and Singh, 2021. Deep Learning Based Object Detection Combined with Internet of Things for Remote Surveillance. Wireless Personal Communications.

[19] Chugh, U., Mitra, A., Ankur, et al., 2021. In: IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, pp.1-5.

[20] Sonali, M., Diware, A., Shweta, and Iskande, 2012. Remote Surveillance System for Mobile Application. Network and Complex Systems, 3, pp.14-19.

[21] Amira, S., Mahmoud, M., Sayed, et al., 2020. Object Detection Using Adaptive Mask RCNN in Optical Remote Sensing Images. International Journal of Intelligent Engineering and Systems.

[22] Yanmao, M., Li, M., Ryan, M. and Gerdes, 2023. Remote Perception Attacks against Camera-based Object Recognition Systems and Countermeasures. ACM Transactions on Cyber-Physical Systems.

[23] EL-Hadad, R., Tan, Y.-F, Tan, W.-N, 2022. Anomaly Prediction in Electricity Consumption Using a Combination of Machine Learning

Techniques. International Journal of Technology. Volume 13(6), pp. 1317-1325

[24] Hadidi, R., Cao, J., Michael, S. et al., 2019. Robustly Executing DNNs in IoT Systems Using Coded Distributed Computing. In: Proceedings of the 56th Annual Design Automation Conference, ACM, pp.1-2. Available at: https://doi.org/10.1145/3316781.3322474.

[25] Rusci, M., Capotondi, A., Conti, F. et al., 2018. Work-in-Progress: Quantized NNs as the Definitive Solution for Inference on Low-Power ARM MCUs?. In: International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp.1-2. Available at: doi:10.1109/CODESISSS.2018.8525915.

[26] Chen, X., Guo, A., Si, X., Yang, J. et al., 2022. A Quantization Model Based on a Floating-point Computing-in- Memory Architecture. In: IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), pp.493-496.

[27] Shirpour, M., Khairdoost, N., Michael, et al., 2023. Traffic Object Detection and Recognition Based on the Attentional Visual Field of Drivers. IEEE Transactions on Intelligent Vehicles.

[28] Kantham, L., Durgam, R., Jatoth, K. et al., 2023. Real-time Classification of Vehicle Logos on Arduino Nano BLE using Edge Impulse. In: 4th International Conference on Signal Processing and Communication (ICSPC), pp.316-320.

[29] Lokhande H.N., Ganorkar S. R., 2021. Masked Face Detection And Voice Based Alert System For Blind People Under Covid-19. International Journal of Engineering Development and Research, 9, pp.104-108.

[30] Mellit, A., Nicola, A., Blasuttigh, P., Massi, P. et al., 2023. TinyML for fault diagnosis of Photovoltaic Modules using Edge Impulse Platform. In: 2023 11th International Conference on Smart Grid (icSmartGrid), pp.1-5.

[31] Son, N. and Truong, S.N., 2020. A Low-cost Artificial Neural Network Model for Raspberry Pi. Engineering Technology & Applied Science Research, 10(2), pp.5466-5469.

[32] Ravi, K. and Jatoth, 2023. Real-time classification of haze and non-haze images on Arduino Nano BLE using Edge Impulse. In: 4th International Conference on Computing and Communication Systems (I3CS), pp.1-7.

[33] Ali, W., Daher, A., Rizik, M. et al., 2021. Porting Rulex Software to the Raspberry Pi for Machine Learning Applications on the Edge. Sensors.

[34] Aftab, F., Bazai, S.U., Marjan, S., Baloch, L., Aslam, S., Amphawan, A., Neo, T.-K., 2023. A Comprehensive Survey on Sentiment Analysis Techniques. International Journal of Technology. Volume 14(6), pp. 1288-1298

[35] Ng.C.H., Connie, T., Choo, K.Y., Goh, M.K.O., 2022. Fusion of Visual and Audio Signals for Wildlife Surveillance. International Journal of Technology. Volume 13(6), pp. 1213-1221