

PRDTinyML: deep learning-based TinyML-based pedestrian detection model in autonomous vehicles for smart cities

Norah N. Alajlan¹, Abeer I. Alhujaylan¹, Dina M. Ibrahim^{1,2}

¹Department of Information Technology, College of Computer, Qassim University, Buraydah, Saudi Arabia

²Department of Computers and Control Engineering, Faculty of Engineering, Tanta University, Tanta, Egypt

Article Info

Article history:

Received Mar 23, 2024

Revised Feb 19, 2025

Accepted Mar 1, 2025

Keywords:

Deep learning

IoT

Pedestrian detection

Smart cities

TinyML

ABSTRACT

Detecting pedestrians and cars in smart cities is a major task for autonomous vehicles (AV) to prevent accidents. Occlusion, distortion, and multi-instance pictures make pedestrian and rider detection difficult. Recently, deep learning (DL) systems have shown promise for AV pedestrian identification. The restricted resources of internet of things (IoT) devices have made it difficult to integrate DL with pedestrian detection. Tiny machine learning (TinyML) was used to recognize pedestrians and cyclists in the EuroCity persons (ECP) dataset. After preliminary testing, we propose five microcontroller-deployable lightweight DL models in this study. We applied SqueezeNet, AlexNet, and convolution neural network (CNN) DL models. We also use two pre-trained models, MobileNet-V2 and MobileNet-V3, to determine the optimal size and accuracy model. Quantization aware training (QAT), full integer quantization (FIQ), and dynamic range quantization (DRQ) were used. The CNN model had the shortest size with 0.07 MB using the DRQ approach, followed by SqueezeNet, AlexNet, MobileNet-V2, and MobileNet-V2 with 0.161 MB, 0.69 MB, 1.824 MB, and 1.95 MB, respectively. The MobileNet-V3 model's DRQ accuracy after optimization was 99.60% for day photos and 98.86% for night images, outperforming other models. The MobileNet-V2 model followed with DRQ accuracy of 99.27% and 98.24% for day and night images.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Dina M. Ibrahim

Department of Information Technology, College of Computer, Qassim University

Buraydah 51452, Saudi Arabia

Email: d.hussein@qu.edu.sa

1. INTRODUCTION

Intelligent transportation systems have been developed in recent years, especially in smart cities, to reduce the amount of traffic in metropolitan areas. Furthermore, it can decrease the number of accidents, injuries, and deaths that are caused by them, lowering the amount of fuel that is consumed, lowering the amount of pollution that is introduced into the environment, and so on. Different technologies, such as the internet of things (IoT), machine learning (ML), deep learning (DL), and image processing, are utilized by these systems to provide a wide range of applications [1].

One of the most important functions for autonomous vehicles (AV) is the ability to detect human motion in their path. Pedestrian detection is a computer vision (CV) technique that is used to detect human motion. This is helpful for ensuring the safety of people, identifying and pursuing a perpetrator in a crowd, preventing accidents, and avoiding moving vehicles and obstacles. A sophisticated array of sensors, including

radar, cameras, and light detection and ranging (LiDAR), can carry out such detection duties. A new system that helps prevent unexpected accidents called advanced driving assistance system (ADAS) was introduced recently [2]. The evolution of ADAS from assisted to fully autonomous driving is already the norm. Currently, the majority of sensors used in AVs rely on radars, LiDARs, and RGB images. In comparison to self-driving providers, image-based sensors have the advantage of being inexpensive, which allows for easy control over production costs. In addition to improving autonomous driving, using image sensors can lower accident rates. Above everything else, pedestrian safety is essential [3].

DL is an advanced branch of ML that finds complex representations of simpler ones. DL approaches often use artificial neural networks with multiple hidden layers and nonlinear processing units. Deep means numerous hidden layers utilized to adjust data display. Each hidden layer of neural networks depicts its input data using feature learning. The layer absorbs more abstraction than the previous layer. DL architectures map the hierarchy of features learnt at numerous levels to ML output in a single framework. Similar to ML methods, DL architecture has two main categories: unsupervised and supervised learning, including deep neural networks [1].

A smart transportation system that uses DL to identify pedestrians and vehicle riders is just one of several applications that combine IoT sensors with DL. This application will encounter new obstacles when it incorporates DL with IoT devices. There are a lot of obstacles to integrating DL with the IoT in image-based pedestrian and vehicle-rider detection scenarios. Several studies and polls on pedestrian detection systems have found that training DL models are the biggest obstacle [1], [2]. In contrast, DL models require a substantial amount of training time because of their complexity, poor training speed, and high computation cost. The study in [1] surveyed the state of the art in AV pedestrian detecting systems in great detail. Although they claimed that DL learning-based models have solved many problems with pedestrian detection, these models are painfully slow, have poor interoperability, and require an excessive amount of time to train. There was also a need for a substantial amount of data for accurate forecasting. To that end, accuracy and speed pose the greatest challenges to pedestrian detecting systems. Nevertheless, to keep a high level of accuracy, researchers were advised by [4] to employ lightweight sensors or to analyze biological signals using video data.

New technology known as Tiny machine learning (TinyML) has emerged to address these issues and overcome the obstacles of DL integration with IoT devices. One definition of TinyML is the combination of ML/DL with IoT devices. With TinyML technology, DL models may be deployed on microcontroller-powered devices with limited resources. The accuracy of the classifier is not sacrificed by the low-cost boards fitted with microcontrollers; they have restricted processing, extremely low power (mW range and less), and small memory sizes [5]-[7].

Therefore, this study set out to improve smart city transportation by using TinyML to pedestrian and vehicle-rider objects. We used three optimization strategies to implement five lightweight DL models in the present work. The three DLs that have been considered are convolution neural network (CNN) deep models, AlexNet, and SqueezeNet. In addition, we used the MobileNet-V2 and MobileNet-V3 deep models that had already been trained. A pair of quantization techniques, post-training quantization (PTQ) and quantization aware-training (QAT), are employed to carry out the optimization procedures. Following this, we will use the Interpreter to assess the DL models after they have been converted to the TensorFlow Lite (TFLite) format.

Here is the outline for the remainder of the paper's content. In section 2, we take a high-level look at TinyML, discussing its definition and the benefits it offers. The relevant research on pedestrian detection is reviewed in section 3. We show our methodology approach to the research in section 4. The application and evaluation of our methods are detailed in section 5. The results are discussed and compared in section 6. Finally, section 7 illustrates the final conclusions.

2. OVERVIEW OF TINYML

TinyML is an emerging field that accelerates IoT inventions in smart cities, such as smart transportation, autonomous driving, smart agriculture, and smart environment. TinyML implements DL tasks locally on machines under a milliwatt [8], [9]. Reduces computational power and data to improve DL algorithms [9]. TinyML is a tiny machine learning -aware framework, architectures, tools, techniques, and approaches that can perform on-device statistical analysis for a wide range of sensing modalities (vision, audio, speech, motion, chemical, physical, textual, and cognitive) at MilliWatt or below with targeting battery-operated embedded edge devices [10]. Hardware, software, and algorithms make up TinyML. Hardware can include IoT devices with or without accelerators. Microcontroller units (MCUs) are appropriate TinyML hardware platforms due to their specs [11]. A CPU, flash memory, random-access memory (RAM), and input/output peripherals make up the microcontroller chip. Microcontrollers are tiny ($\sim 1 \text{ cm}^3$), low-cost, and low-power [9]. Their clock speed is 8 MHz to 500 MHz, power is 15 mA, RAM

storage is 8 KB to 320 KB, and flash memory is 32 KB to 2 MB. Recently, software titans have shown interest in TinyML. Microsoft offers EdgeML [12], Google has TFLite, which lets IoT devices run neural network (NN) models [13]. However, TinyML as DL algorithms should be tiny (a few KB). Quantization, pruning, and knowledge distillation may be used to condense DL models for IoT devices with limited resources [11].

Optimization and conversion of DL models: to enable IoT device DL inference, DL model optimization methods build lightweight models. The weight of DL models makes integrating them with IoT devices difficult. Devices that consume resources make DL inference difficult. DL model optimization compresses and accelerates superfluous parameters without affecting accuracy. The common DL model optimization strategies are quantization, network pruning, and weight factorization [14]. Quantization is a standard TinyML compression approach, thus this study uses it. Quantization methods use less flash memory and RAM while preserving model accuracy [15].

Quantization approaches use optimization principles to meet device restrictions with limited resources [15]. Quantization involves substituting high-precision DL parameters with low-precision ones without affecting model architecture. By 32-bit floating-point values [14], [16]. The quantization (1) uniformly maps real values to integer values.

$$\text{Quant}(R) = \text{Int}(R / S) + Z \quad (1)$$

Where *Int* rounds to the nearest integer to set the output value. *R* are real numbers, *S* is a scaling factor, and *Z* is an integer zero point. Scaling is a positive integer that determines quantization step size. Real number values depend on scaling factor. *R* has several real number ranges, each mapped to a numerical value.

Skirmantas Kligys of Google Mobile Vision pioneered QAT. Quantization methods like this innovation try to correct for quantization error and reduce accuracy loss. Train quantized deep neural networks during the forward pass. The training of DL models still uses floating-point or non-quantized values. The quantized model's average outcomes improve and the learning phase stabilizes. Pre-training DL models using appropriate configuration parameters using floating point values [17].

After DL model training, post-training quantization (PTQ) quantifies DL parameters. Once DL model training is complete, freeze and quantize parameters. Unlike QAT, PTQ quantified parameter weight without training parameter modifications [17]. The quantization error of each parameter in the PTQ causes activation error. Additionally, decreasing weight values increase quantization error. Thus, accuracy decreases. PTQ's downside is quantization errors at the output, which might cause the classifier model to misclassify the input data, lowering accuracy compared to the training model. PTQ's three methods-Float16 quantization, dynamic range quantization (DRQ), and full integer quantization (FIQ)-reduced the size of a training DL model with convolutional layers by four times and sped up execution by 10-50% [18]. DRQ and FIQ were chosen for current study because these methods scaled the models four times smaller and the 16-bit floating point twice as small: (i) FIQ: it converts all model weight and activation parameters from 32-bit floating-point to 8-bit integer integers. Although FIQ quantization has benefits, it reduces accuracy. It quantizes the model, reducing accuracy but keeping it within acceptable limits. In FIQ, weights and activations are quantized by scaling over 8-bit integers. (ii) DRQ: the default method for PTQ reduces model size and optimizes inference delay [19]. After training DL models from floating-point numbers to fixed-point integer numbers, PTQ quantizes and activates their weights. Operators dynamically quantize activations based on their range to 8-bit integers and conduct computations with 8-bit integer weights and activations to reduce inference time in "dynamic range". After multiplication and accumulation, activation values are dequantized to 32-bit floating-point numbers. In contrast to FIQ, DRQ converts activation values to integers "on-the-fly" during inference time. Since DRQ quantization does not require a representative dataset, it has advantages over the whole integer. As DRQ stores activations values as floating-point numbers during the "stand-by" phase, Edge TPU cannot be used as quantized model custom hardware because it only supports integer arithmetic [20].

Most smart city apps function well in smart environments, transportation, agriculture, and homes using DL and IoT devices. This complicates survey and review research. New tech TinyML leverages DL in many IoT apps. It optimizes DL models for IoT devices' limited resources to improve accuracy and performance. DL meets smart city IoT devices with TinyML. Next subsections explain how lightweight DL model training allows IoT devices with limited resources and power to infer.

Train lightweight DL models: complex deep-learning models are needed for smart city apps. Training large DL models requires GPUs and hours. Large storage memory is needed for electric IoT devices. For accuracy, IoT devices must be powerful and resource-intensive. TinyML reduces DL model training for microcontroller deployment, making low-resource IoT devices smart. Pruning and quantization compress models. This lightweight training paradigm for IoT devices improves battery life and saves operational costs for smart city IoT users [21], [22].

TinyML data is processed locally on computing devices to infer lightweight DL models in IoT devices. Real-time local data processing lets IoT devices assess and respond fast, especially in crises. Also decreases cloud burden [23], [24]. Gateways, cloud services, and DL models help IoT devices input and handle data. Large model sizes slow and delay device DL inference. For TinyML, inferring the DL model and processing data on the device solves response times, network bandwidth, and storage issues. Processing raw data on the device reduces latency and bandwidth [25], [26].

TinyML is good for IoT devices with MCUs because it uses small batteries and less electricity. Due to their strong processors and GPUs, IoT devices demand lots of power. TinyML, requiring 150 μ W to 23.5 mW, connects IoT devices, such as scooters and segways, from anywhere [27]. Itinerant cognitive apps can help smart cities.

TinyML uses little resources to infer DL models to smarten low-resource IoT devices. TinyML fixes smart city IoT difficulties. Smart city apps collect, process, and store data using sensors and microcontrollers and cloud DL. Cloud computing uses TFLOPs, GPUs, and TPUs. to run 16-32 GB DL models quickly, delaying data and leveraging network capacity. Second, GPUs and microprocessors enable real-time DL model processing in smart cities without cloud computing. Laptops, tablets, and smartphones with NPU technology with 8 RAM can handle huge storage and computing capacity. Mobile smart city apps need lots of computer power, storage, and battery life [28], [29]. TinyML uses DL on a 2MB RAM microcontroller to save money and resources. TinyML advances smart city innovation across sectors.

3. RELATED WORKS

In order to identify pedestrians from images and videos collected by IoT devices, a number of studies have utilized a wide range of DL models, including CNN, region-convolutional neural networks (R-CNN), visual geometry group-16 (VGG-16), you only live once (YOLO) variants, AlexNet, and much more. For instance, Zhu *et al.* [30], used background removal and DL to study long-distance pedestrian recognition challenges. This process comprises two steps. First, this model presents ML detachment framework facts. The attention module improves RefineDet's recognition of small pedestrians in the second step. This approach requires additional benchmarks from various cartographic regions with high foot activity to assure validity. Kim *et al.* [31] investigated the detection of pedestrians in smart buildings. owing to the fact that the identification of pedestrians is difficult owing to noise in images as well as certain environmental conditions and parameters. The deep CNN was utilized by the researchers in order to develop a vision-based model. Additionally, the optimized version of the VGG-16, which was referred to as the OVGG-16, served as the architectural core that was utilized in order to differentiate pedestrians from the numerous available images. For the purpose of evaluating the suggested method, the researchers utilized the INRIA dataset, which comprised 6,817 photos, including 3,239 photographs of pedestrians. The image quality in this dataset was 227 \times 227 pixels. In comparison to previous approaches to ML, the proposed method has a high level of accuracy (about 98.8%) when it comes to the accurate identification of pedestrians. This was demonstrated by the findings of the researchers' investigations.

Tomè *et al.* [32] proposed a system for the identification of pedestrians that makes use of DL principles. In addition to this, the researchers suggested a new paradigm for recognizing pedestrians. DL models called AlexNet and GoogLeNet were utilized by the researchers in order to facilitate the implementation of their proposed algorithm on contemporary hardware. Additionally, the researchers proposed new methods for various stages of pedestrian identification. They utilized the NVIDIA Jetson TK1, a computing platform that is based on graphics processing units (GPUs), as well as the Caltech Pedestrian dataset in order to implement and assess the offered methodologies and solutions. This particular dataset includes approximately ten hours of video content that pertains to autos and was gathered in a variety of weather conditions. This dataset had 137 minutes of video content with a total of 2,300 different pedestrians and 250 thousand frames each minute. Half of the frames contained no pedestrians, whereas thirty percent of the frames contained two or three pedestrians. As a result of the execution of their proposed technology, the accuracy of the AlexNet model is 80.1% and for GoogLeNet model is 80.3%. these researchers were able to demonstrate that it is highly efficient and accurate in spotting pedestrians in real time. Further researchers proposed multimodal imaging and ML methods to design, implement, and evaluate pedestrian detection methods [33]. Two cameras were calibrated using a specially designed reference checkerboard. They proposed method for superimposing multimodal images using RGB data and thermal to produce the final images. They created new dataset with 8,000 multimodal images in good and limited visibility. For detect pedestrian in a new representation of multimodal data, they proposed adapted YOLO model which contain edge device variant for TFLite. The best result achieved by adapted YOLOv5s model for subset images for good visibility with 93.4% for edge devices. Sequent, YOLOv5s model achieved 87.8% and 86.8% for limited visibility.

Within the context of the automated driving approach, Chen *et al.* [34] investigated the various architectures that are currently in use for pedestrian detection. Following an explanation of the necessity of employing ways to identify a pedestrian and ascertain his or her itinerary, these researchers went on to explore the procedure of detecting a person while driving a vehicle. They then described how to use DL techniques (such as R-CNN and support vector machine (SVM)) to discover two-step and one-step patterns, and then they tested the usefulness of the patterns that were identified to detect pedestrians from the patterns that were discovered. Last but not least, the researchers investigated and contrasted the approaches that were suggested by other researchers in order to identify pedestrians. The KTH dataset, the UCF series dataset, the Hollywood2 dataset, and the Google AVA dataset are some of the datasets that they introduced. These datasets are utilized to investigate the proposed approaches for identifying pedestrian activity. The accuracy rate obtained by the R-CNN model is 85.5%. Moreover, according to the requirements of automated driving for real-time pedestrian detection, researchers in [35] proposed DL model called LeNet-5 convolutional neural network (LteNet-5CNN) to detect the pedestrians accurately. They train and test a model using Caltech dataset. The accuracy results were above 95%, which better than the results from SA-Fast R-CNN and classical LeNet-5 CNN models. After model training, they applied mode real-time on smart car using Intel Core i7 processor with 2.5 GHz, and a memory size of 4G. They experiment showed improved in detection time with 0.394 s compared with the mainstream LeNet CNN detection time and 0.155 s, in compared with SA-FastR-CNN.

For the purpose of providing real-time reactions in driver assistance systems, Said and Barr [36] proposed a novel software that makes use of a DL algorithm for the purpose of detecting pedestrians in a quick and accurate manner. The utilization of item classification, pedestrian identification, and position tracking are all included in this program. Within the context of the implementation, learning, and testing of the proposed method, the TensorFlow DL framework, the NVIDIA, cuDNN, and OpenCV acceleration libraries, as well as the Caltech dataset, were utilized. For the purpose of developing driver assistance systems, this program is installed for deployment in mobile phones or embedded systems that are connected to self-driving cars.

Ahmed *et al.* [37] contrasted the approaches taken to diagnose bicycles and pedestrians. A self-driving vehicle's detection stage is crucial for building smart applications, according to their statement, since it's there that objects can be located and detected using deep learning techniques like fast R-CNN, faster R-CNN, and single shot detector (SSD) in video and image frames. Lastly, it is possible to monitor and identify bikes or pedestrians using tracking findings. The primary objective of this research was to examine the current techniques used to detect bicycles. According to their research, one way to keep roads safe is to use methods that can accurately identify people on foot and bikes, such as sensor fusion and intent estimate and the accuracy of the R-CNN is 91%. On another hand, researcher's impalement fine tuning on Faster R-CNN model. Wherein, fine-tuning has great interest due to its retraining convolutional networks, and shown state-of-the-art performance for detection, and segmentation [38]. Thus, they evaluate and comparing of Faster R-CNN Inception v2, single shot multibox detector (SSD) Inception v2, and SSD Mobilenet v2 models. They trained the models on MS COCO dataset and tested on their own dataset and public domain dataset introduced by the Cityscapes project. The best accuracy results of DL models were 81.0%, 78.1% and 64.3% for Faster R-CNN Inception v2, SSD Inception v2 and SSD Mobilenet v2 models respectively.

A system for tracking and detecting pedestrians using deep neural networks was proposed by Yu *et al.* [39]. This system utilized a unmanned aerial vehicle (UAV) and Kalman Filter forecasting approach to monitor objects and pedestrians. Additionally, a dataset (YOLOv3) was utilized in order to put the proposed method into action. Examining the accuracy and execution time of the suggested approach for monitoring and recognizing pedestrians, as well as watching and identifying items, was done in order to determine how effective the method is. When it came to identifying pedestrians, the proposed method was shown to have less errors, as demonstrated by the results of the testing studies.

Recent research, Liu *et al.* [40], address the challenges faced dense pedestrian detection such as complex models which are difficult to deploy on IoT devices, high computational weight, low detection accuracy for occluded pedestrians and small targets. They proposed lightweight cascade fusion network (CFNet) and a CBAM attention module. The WiderPerson public dataset was chosen that contains real street intersection images, they created a pedestrian detection dataset suitable for dense scenes, accounting for varied pedestrian occlusion scenarios. The experiment results show the accuracy is improved approximately 2.4% with 88% for pedestrian, 43.3% for rider and 32.9% for partially visible person. Finally, the number of parameters is reduced by 0.5 MB with 2.7 M. The experiment verified that the proposed method is valuable for device with limited computational resources. In addition, for low-cost DL model [41] Researchers proposed low-cost DL model on intelligent vehicles for road-segmentation and pedestrian detection. Firstly, they used DL-based consecutive triple filter size (CTFS) approach to segment the road. Secondly, detect the pedestrian using YOLOv7 model. Two datasets were used which are camVid dataset for roed segmentation

and PascalVOC dataset for pedestrian detection. The results were 95.84% for road segmentation for using Jaccard index value, along with 65.50% for the pedestrian detection using average precision value.

For optimize the AVs system for pedestrian's detection and effective collision risk reduction on complex scenarios. Researchers on [42] strive to optimize the reliability and the efficacy of pedestrian detection in complex scenarios. They introduced a new pedestrian tracking models that leverages both the YOLOv8 and the StrongSORT model, that is an advanced DL multi-object tracking method. The experiment applied on the MOT16 and MOT17 datasets. Three evaluation metrics were used namely IDentity F1 (IDF1), higher order tracking accuracy (HOTA), and multiple objects tracking accuracy (MOTA). The best result of YOLOv8 model was 0.92 using IDF1 metrics. For StrongSORT model the best result was 55.338 using IDF1 on MOT17 dataset. In another case, Lee *et al.* [43] introduced novel system called NAVIBox that designed to detect the vehicle-pedestrian by using various of vision sensors that deployed on IoT devices in the field. They utilized YOLOv8 for object detection real-time. The results of their experiment were 0.71 0.67 of precision and recall for pedestrian and 0.81, 0.85 precision and recall for vehicles.

Generative adversarial networks (GANs) were proposed by Dinakaran *et al.* [44] in order to develop a novel cascaded SSD architecture for the purpose of remote pedestrian detection. Additionally, DCGAN is utilized in this architecture in order to enhance the image quality for the purpose of distant pedestrian detection. For the purpose of identifying the objects depicted in the image, this proposed method makes use of a number of factors. In order to put the strategy that has been suggested into action, the dataset that is from the Canadian Institute for advanced research (CIFAR) is utilized. It has been demonstrated through the results of trials that the proposed method possesses a high degree of accuracy equal 80.7% when it comes to distinguishing pedestrians and cars from a considerable distance.

4. METHOD

This section describes the proposed methodology of our experiment for the pedestrian and vehicle-rider detection case using TinyML, which we called PRD-TinyML model. Which aims to detect pedestrian and vehicle-rider using small DL models. The workflow of methodology is divided into many phases each phase is linked to the next phase. Figure 1, illustrates the phases of implementing the proposed method, whilst the details of the implementation are described as follows:

4.1. Phase 1: data collection and pre-processing

In this phase, EuroCity persons (ECP) dataset is used to detect the pedestrians in the road which indicates the it is pedestrian or rider [45]. In case of rider, there are seven objects or types of drivers: bicycle, motorbike, scooter, tricycle, wheelchair, buggy, and co-rider. In addition, for each category from the pedestrian or the riders there are images during day and also during night. Subsequently, implement pre-processing on the dataset images by detecting the objects using three methods namely single shot multibox detection (SSD), Dlip library (Dlip), and Haar-based cascading classifier (Haar) in order to use as input for DL models to detect of pedestrians.

4.2. Phase 2: model training and evaluation

In this phase, several types of supervised DL models are developed to detect the pedestrian objects, aim to examine their performance, and obtained the best of them in size and accuracy. Three DL models we developed namely SqueezeNet, AlexNet, and CNN, further adapt two pre-trained models which are MobileNet-V2 and MobileNet-V3 models. These models were pre-trained on ImageNet dataset. Subsequently, save models for fed to the optimization model phase.

4.3. Phase 3: model optimization and conversion

This phase aims to optimize the saved DL models through reducing the size of models after that converted to Tensor Flow Lite format. First, optimized the saved models by using various of quantization methods namely DRQ, FIQ, and QAT as described in the next section. Quantization aims to convert the weights of models or activation or both from float32 to Int8 format which provides a significant reduce in model size. Thus, it led to a decrease the memory footprint in devices. Second, convert DL models to TFLite format (.tflite) for enable inference of DL models.

4.4. Phase 4: running interpreter

Running the TFLite interpreter to evaluate the TFLite DL models on the host computer. By loading the TFLite model to Interpreter and used the testing dataset for evaluation.

4.5. Phase 5: models' conversion to C array

Convert TFLite models using TFLite micro tools that converts DL models to C byte array. Using XXD that generate C source file for the TFLite models as char array. After that, deploy the model into an independent platform that using C++ language as Arduino software and compile with IoT devices as microcontroller.

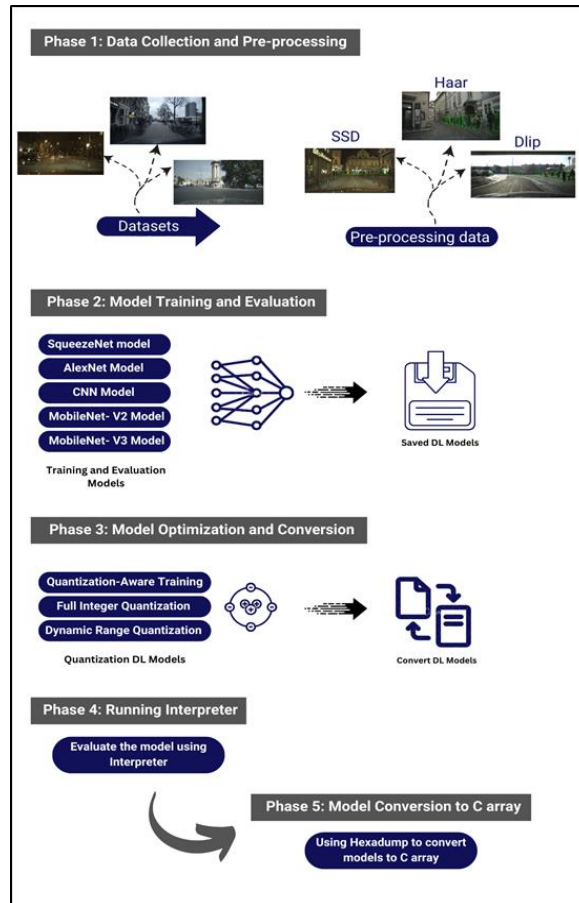


Figure 1. Proposed PRD-TinyML methodology

5. IMPLEMENTATION AND TESTING

5.1. Implementation environment

The environment used to perform the experiment is Google Colaboratory [46], [47] or “Colab” for short, is a product from Google Research. We implement the experiment using Colab pro version [27] that was chosen to execute the experiment in this paper due to the features of the environment provided. Wherein, Colab pro version came with 2 terabytes of storage, 25 gigabytes of RAM, and a GPU processor P100. The experiment was performed using open-source libraries and artificial intelligence (AI) frameworks. The experiment was written in Python3 programming language and used NumPy [48], Matplotlib [49], Panada [50], and Scikit-learn [51] libraries. The frameworks used are TensorFlow [52], TFLite [53], and TFLite micro [54] frameworks that describe in next section.

5.2. Experiment and implementation

The following subsections, explain in detail the implementation of each phase for the proposed methodology. Phase 1 (data collection and pre-processing), phase 2 (model training and evaluation). Phase 3 (optimization and conversion model), phase 4 (running the interpreter). Finally, phase 5 (convert model to C array).

5.2.1. Phase 1: data collection and pre-processing

Collection and pre-processing dataset phase, aims to collect a good training dataset of pedestrian and rider detection case. In addition, we perform pre-processing on the pedestrian and rider, day and night,

images of the EPC dataset. In order to optimize achievement of higher performance in train and evaluation deep learning models. This section, introduce the detail of the collected datasets. Thereafter, it presents the steps to datasets Pre-processing.

A. The collected dataset

EuroCity [45], TUD known as Brussels Pedestrian dataset [55], Caltech [56], CityPersons [57], INRIA [58], KITTI [59], and ETH datasets are examples of well-known pedestrian detection datasets that are currently available. The following are some specialized datasets for pedestrian identification, which are frequently utilized in research. The current dataset for pedestrian detection is called ECP [60], and it performs better than both the Caltech dataset and the CityPersons dataset in terms of the difficulty and heterogeneity of the data. Note that it is based on information collected from thirty-one cities spread out across twelve states in Europe. Day and night photographs are taken in Europe, which performs the role of an umpire due to the fact that ECP day is referred to as daytime and ECP night is referred to as midnight. The limit box that has been defined is greater than 200 K. All of the investigations and comparisons that were carried out in ECP [60] were carried out during the daylight in conjunction with other methods. In addition, a diagnostic server is accessible. On the other hand, there are restrictions on test sets and frequency submissions.

In this study, we made use of the ECP dataset, which offers a significant number of annotations that are extremely diverse, accurate, and detailed [61]. These annotations include walkers, cyclists, and other riders who are seen in urban traffic scenes. The photographs that make up this dataset were taken while the vehicle was in motion in a total of thirty-one cities across twelve European nations. The ECP dataset is roughly one order of magnitude larger than the datasets that were previously utilized for the purpose of person detection in traffic scenes. It contains over 238,200 person instances that have been carefully tagged in over 47,300 photos total. Additionally, the dataset includes a substantial quantity of person orientation annotations, accounting for more than 211,200 in total. Table 1 shows the number of images and instances in day and night for the ECP dataset categories. It consists of two main categories: pedestrian and rider, about 40,217 images during day 183,004 instances for pedestrians and 18,216 instances for riders. While during night, the total number of images is 7,118 have about 35,309 instances for pedestrians and 1,564 rider instances. For the rider category, there are 7 different objects: bicycle, motorbike, scooter, tricycle, wheelchair, buggy, and co-rider. Table 2 shows the number of images in each of these objects during day and night.

Table 1. Distributed images and instances in day and night for the ECP dataset categories

Categories of ECP	Day		Night	
	Images	Instances	Images	Instances
Pedestrian	40,217	183,004	7,118	35,309
Rider		18,216		1,564

Table 2. Distributed rider objects in day and night for the ECP dataset

Objects of rider	Number of objects		
	Day	Night	Sum
Bicycle	9,666	614	10,280
Motorbike	2,196	229	2,425
Scooter	5,748	683	6,431
Tricycle	94	5	99
Wheelchair	125	4	129
Buggy	322	26	348
Co-Rider	671	137	808

B. The pre-processing of the dataset

To ensure that deep learning models are trained and evaluated to their full potential, pre-processing is commonly employed to get datasets ready for use as training sets. We used pre-processing on two types of pedestrian detection images: those with people walking and those with riders. Bicycles, motorcycles, scooters, tricycles, wheelchairs, buggies, and co-riders are the seven items that fall within the rider group. There are multiple steps in the preprocessing. The first step is to use OpenCV's object detection algorithms, such as the SSD, the Dlib package, and the Haar-based cascade classifiers detection as shown in Figure 2. After reading and resizing photos from the ECP dataset, we proceed to resize all of the images using detection face algorithms. Afterwards, we merged the two sets of data. Image transformation into a form suitable for use as inputs to deep models is the responsibility of the second stage. Step three involves

redefining the categories label in datasets to binary by converting multi-class labels to binary labels using the LabelBinarizer method. As a fourth step in ensuring image diversity, we randomly divided our dataset into 70% training and 30% testing. For reproducible results, it's also recommended to set the random generator's seed to 40. Next, we'll shuffle the data by giving the actual value for shuffling. There are 32 examples used for training in each iteration, which is known as the batch size. The testing dataset, in contrast, scaled images using the MinMaxScalar class.



Figure 2. Sample for the images in the pre-processing methods

5.2.2. Phase 2: model training and evaluation

In this research, we developed five lightweight DL models detect the pedestrian in AV, aiming to obtain the best performance. These models have the advantage of small size due to the limitations of IoT devices. After we conducted a lot of experiment on many of DL models architectures and comparing their size with performance. We developed three deep models which are SqueezeNet, AlexNet, and CNN models, and adaptive two pre-trained deep models namely MobileNet-v2, and MobileNet-v3 that were pre-trained on ImageNet dataset. The following subsections describe the architecture of deep models used in the experiment.

A. The SqueezeNet DL model

In this research, we modified on the original architecture of SqueezeNet deep model. Aiming to produce a lightweight deep learning model with maintaining competitive accuracy. Hence the SqueezeNet model after modify architecture is lighter than the original architecture [26], [62]. We developed the model with few of parameters and high performance, able to detect the pedestrians. The main idea of SqueezeNet model architecture is the fire module which comprises a squeeze convolution layer with only a (1×1) filter, feeding to an expanded layer that has a mix of (1×1) and (3×3) convolution filters as shown in Figure 3. The SqueezeNet model architecture first comprises a stand-alone convolution layer with input shape $(192 \times 192 \times 3)$, followed by a BatchNormalization layer and the rectified linear unit (ReLU) activation function. The ReLU activation function is involved in each layer. MaxPooling layers with strides equal to two values come after each of the five fire modules. At the end of the model, there is a GlobalAveragePooling2D layer and a dense layer with a SoftMax activation function to detect the pedestrian. as illustrated in Figure 3. Table 3 shows the SqueezeNet model architecture and the number of parameters. The total of parameters is 121,500, that divided into 120,930 trainable parameters and 570 non-trainable parameters. Trainable parameters mean the parameters which are updated during the training process to obtain optimal values. However, non-trainable parameters are not updated and optimized during the training process: as a result, they do not contribute to the classification process.

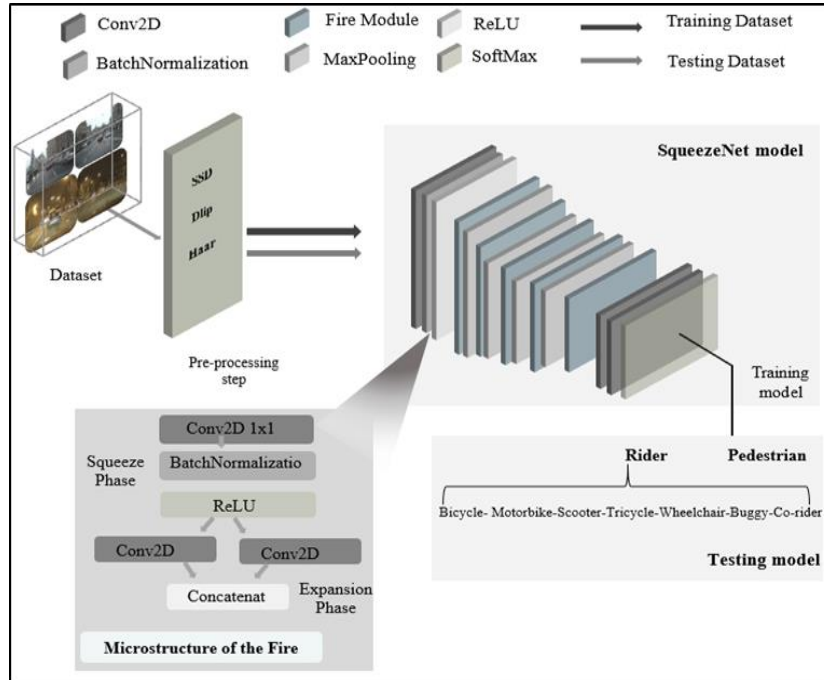


Figure 3. The architecture of the SqueezeNet DL model

Table 3. The architecture summary of the SqueezeNet DL model

Layer name	Image size	Filters	Layer name	Image size	Filters
InputLayer	192×192	-	Activation	48×48	64
Conv2D	192×192	32	2Conv2D	48×48	64
BatchNormalization	192×192	32	Concatenate	48×48	128
Activation	192×192	32	MaxPooling2D	24×24	128
Conv2D	192×192	24	Conv2D	24×24	48
BatchNormalization	192×192	24	BatchNormalization	24×24	48
Activation	192×192	24	Activation	24×24	48
2Conv2D	192×192	24	2Conv2D	24×24	48
Concatenate	192×192	48	Concatenate	24×24	96
MaxPooling2D	96×96	48	MaxPooling2D	12×12	96
Conv2D	96×96	48	Conv2D	12×12	24
BatchNormalization	96×96	48	BatchNormalization	12×12	24
Activation	96×96	48	Activation	12×12	24
2Conv2D	96×96	48	2Conv2D	12×12	24
Concatenate	96×96	96	Concatenate	12×12	48
MaxPooling2D	48×48	96	GlobalAveragePooling2D	-	48
Conv2D	48×48	64	Dense	-	2
BatchNormalization	48×48	64	-	-	-
Total parameters:			121,500		
Trainable parameters:			120,930		
Non-trainable parameters:			570		

B. The AlexNet DL model

The AlexNet deep model is selected in this research as it is a lightweight efficient model with few parameters. Beside that it achieved high accuracy even with its small size, as in previous TinyML studies [6]. In this research we modify the original AlexNet model architecture in [5] to be smaller in size with few parameters. Thus, the architecture of AlexNet after modified consists of six layers with an input shape of 192×192×3. Firstly, the first three layers are convolution layers, each layer followed by a BatchNormalization layer, a ReLU activation layer, and MaxPooling layers. The MaxPooling layers has padding that has a valid value, strides equal to two values, and pool size equal to two. Secondly, there are flatten layer and a dense layer, with dropout having a 0.4 value. Finally, the output layer called Dense layer including a SoftMax activation function to enable detect pedestrian. Figure 4 and Table 4 present the architecture of the AlexNet deep model after modified. The parameters total 486,960 which are divided into 486,132 trainable parameters and 828 non- trainable parameters.

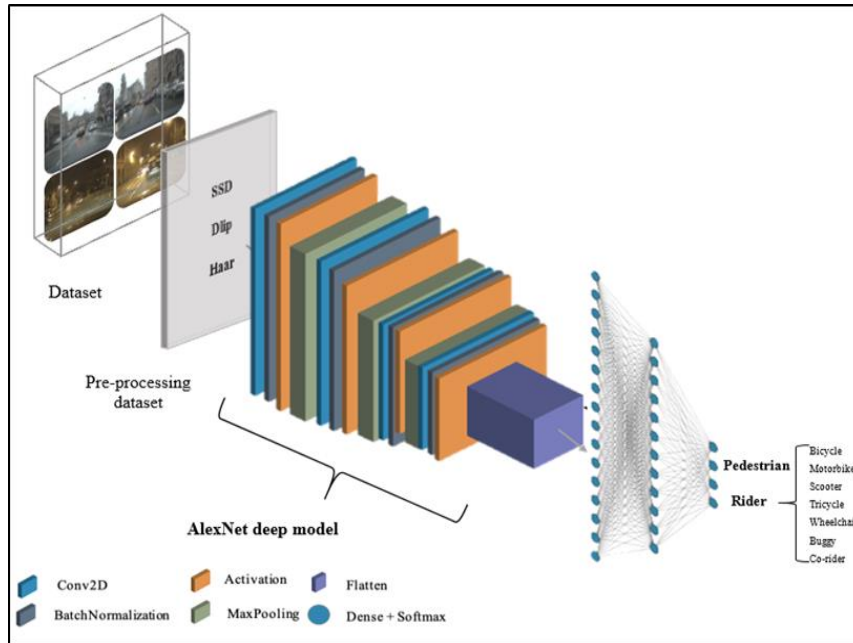


Figure 4. The architecture of the AlexNet DL model

Table 4. The architecture summary of the AlexNet DL model

Layer name	Image size	No. of filters
InputLayer	46×46	-
Conv2D	46×46	32
BatchNormalization	46×46	32
Activation	46×46	32
MaxPooling2D	23×23	32
Conv2D	13×13	64
BatchNormalization	13×13	64
Activation	13×13	64
MaxPooling2D	6×6	64
Conv2D	4×4	128
BatchNormalization	4×4	128
Activation	4×4	128
MaxPooling2D	2×2	128
Flatten	-	512
Dropout	-	512
Dense	-	2
Total parameters:	486,960	
Trainable parameters:	486,132	
Non-trainable parameters:	828	

C. The CNN deep learning model

The CNN deep model is selected in this research, due to it designed to visual data. In addition, it shows effective results in pattern identification and the problem of feature extractions [57]. In the current research, the CNN deep model is proposed for detecting pedestrian, while maintaining performance of accuracy with small size of model. The architecture of CNN model consists of the input layer that receives input images with (145×145×3). The three layers are 2DConvolutional layers that extracted features which differentiate different images from one another, along with ReLU, the activation function. Between the 2D convolutional layers there are two MaxPooling layers which added between them, with the pool size set to 2, and padding having a valid value. Followed by flatten layer which transforming the shape of data to a one-dimensional (1D) data vector. The final layer is an output layer, namely, a dense layer which contains the SoftMax activation function for detecting Pedestrians. The proposed of CNN model architecture can be seen in Figure 5. The parameters in the CNN model total 39,554, with all parameters being trainable parameters, with zero non- trainable parameters. Table 5 presents the CNN model architecture and the number of parameters.

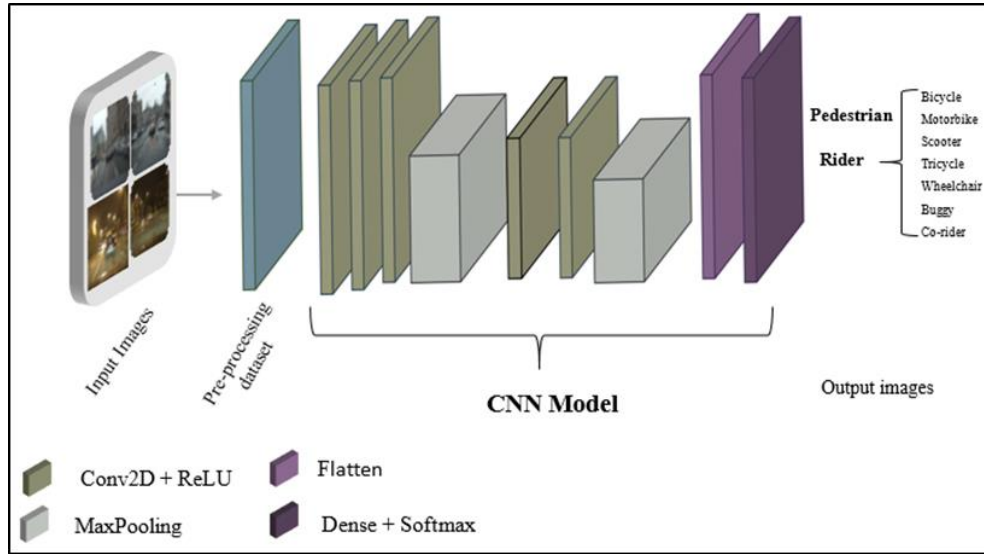


Figure 5. The architecture of the CNN DL model

Table 5. The architecture summary of the CNN DL model

Layer name	Image size	No. of filters
InputLayer	143×143	-
Conv2D	143×143	10
Conv2D	141×141	10
MaxPooling2D	70×70	10
Conv2D	68×68	10
Conv2D	66×66	10
MaxPooling2D	32×32	32
Flatten	-	4608
Dense	-	2
Total parameters:		39,554
Trainable parameters:		39,554
Non-trainable parameters:		0

D. The MobileNet-V2 DL model

The MobileNet-V2 deep model [63] is selected in this research due to its advantages. It is fast, lightweight, and achieved high accuracy. Besides that, it is appropriate for training with limited datasets, wherein the MobileNet-V2 deep model is pre-trained on datasets, such as ImageNet. The current research used the MobileNet-V2 model that pre-trained on ImageNet dataset, and re-trains it on the training dataset to fine-tune its parameters. Thus, it leads to faster training, and meets the requirements of a large dataset. The architecture of model is comprised of a pre-trained MobileNet-V2 model with input shape (224×224×3) and alpha value of 0.75. Followed by reshaping layers which are the MaxPooling 2D, flatten, and Dense layers with SoftMax activation function to detect the Pedestrians status. Figure 6 shows the architecture of MobileNet model. Table 6 contains the contain information regarding the architecture of the model and number of parameters. The number parameters are 1,256,178 which are divided into 1,250,398 for trainable parameters and 5,780 for non-trainable parameters.

Table 6. The architecture summary of the MobileNet-V2 DL model

Layer name	Image size	No. of filters
MobileNet-V2	7×7	2,480
GlobalAveragePooling2D	-	2,480
Dense	-	2
Total parameters:		1,256,178
Trainable parameters:		1,250,398
Non-trainable parameters:		5,780

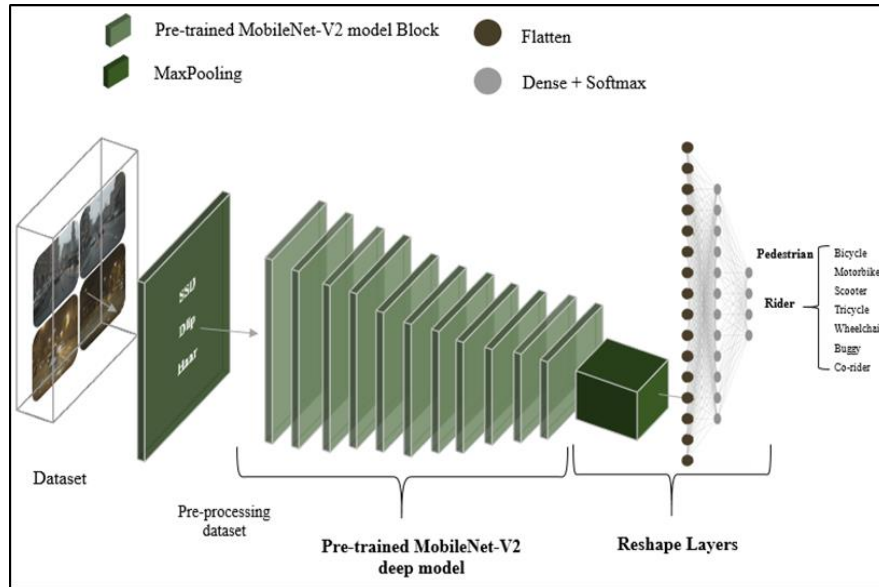


Figure 6. The architecture of the MobileNet-V2 DL model

E. The MobileNet-V3 DL model

The MobileNet-V3 deep model [64], updated version of MobileNet-V2 deep model. Google produced the MobileNet-V3 model, which is part of the neural network family architecture, for efficient image classification on devices and related tasks. MobileNet-V3 is selected in the current research aims is to develop mobile CV architecture to improve latency and accuracy on mobile devices. Google produced two versions of MobileNet-V3, namely, MobileNet-V3 large and MobileNet-V3 small. The current research selects the pre-trained MobileNet-V3 small model from the Keras library. The adjusted model comprised of pre-trained weights on the ImageNet dataset, top layer set to false and alpha value equal to 1.0. The input shape of the model is $224 \times 224 \times 3$, while the trainable parameters of the model are set to false value. Followed by the reshape layer which consists of the flatten layer and dense layer with the SoftMax activation function to detect the pedestrians. The total of parameters is 1,039,180 which is divided into 999,855 and 39,325, for trainable parameters and non-trainable parameters respectively. Both of Figure 7 and Table 7 presents the more information regarding the architecture of the de-tails of MobileNet-V3 model.

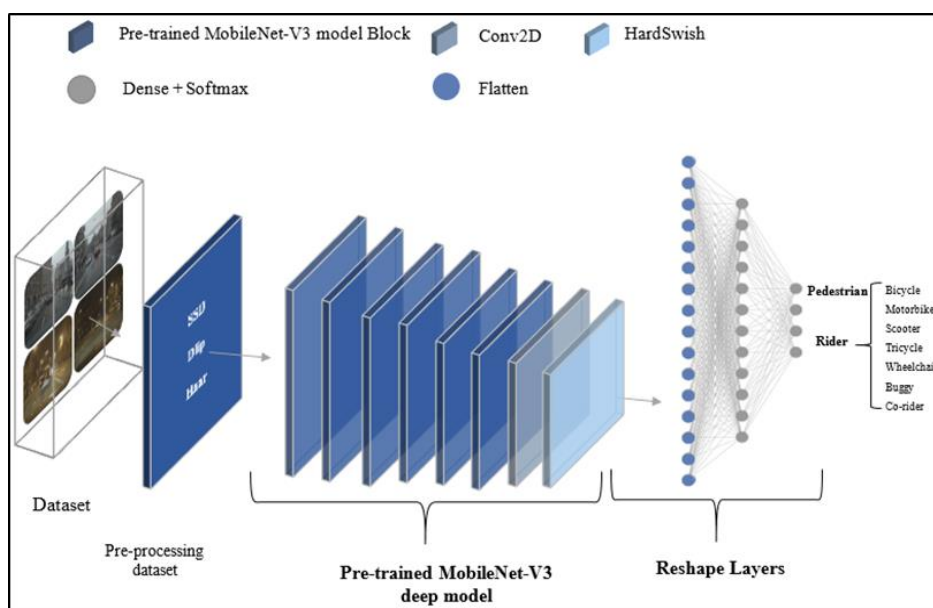


Figure 7. The architecture of the MobileNet-V3 DL model

Table 7. The architecture summary of the MobileNet-V3 DL model

Layer name	Image size	No. of filters
MobileNet-V3	7×7	462
Flatten	-	14,112
Dense	-	2
Total parameters:		1,039,180
Trainable parameters:		999,855
Non-trainable parameters:		39,325

5.2.3. Phase 3: optimization and conversion models

Using quantization techniques, we initially reduce the model size throughout the optimization and conversion model's phase. Two quantization approaches, post-training quantization and QAT, are used in our experiment, as described in section 2. To begin improving ML and DL models for deployment and execution, we install the TensorFlow model optimization toolkit in QAT [65]. After that, we apply `quantize_model` to SqueezeNet, AlexNet, CNN, and MobileNet-V2 models so that they are all cognizant of quantization. However, we performed custom quantization on the MobileNet-V3 model by quantifying just the dense layer. As the Hypermeter configuration setting, this is the setting that the compiled model uses. We used a training dataset to train the models, and a testing dataset to measure their performance. All of the models have 30 iterations, and we are setting the value of shuffling data to true.

We use DRQ and FIQ as part of our post-training quantization process. After training is complete, set the optimization in DRQ to quantize the model weights using 8 bits. Afterwards, you may convert the models to TFLite format (a serial format based on FlatBuffers library) using the TFLite converter tools. Take note: change the dataset type to float32 before changing the models. On the other side, the FIQ turned all of the DL parameters, such as activation and weights, into int8-bit integers using the representative dataset.

5.2.4. Phase 4: running interpreter

In order to complete the quantized model evaluation, TFLite interpreter was utilized. Following the loading of the quantized model, the testing dataset was utilized for the purpose of evaluating the models based on their accuracy. Initially, the quantized model should be assigned to the interpreter. After that, the shape and kind of interpreter should be adjusted. In our experiment, we will resize the shape of the input and output dependent on the shape of the test dataset for both the input and the output. "numpy.float32" is the class of the interpreter that is used for DRQ, and "numpy.int8" is the class that is used for FIQ. The interpreter type is determined by the quantized model type that is input. The next step is to make the testing dataset the input to the interpreter, and then to send it on to the invoke interpreter so that processing may be carried out.

Particularly noteworthy is the fact that in QAT, prior to setting the tensor, apply the quantization value of the input interpreter to the scale and zero-point variables. Following this, assign the test images by dividing them by the sum of their respective values. The results of the interpreter are sent to the Argmax function, which then evaluates the function based on metrics such as accuracy.

5.2.5. Model conversion to C array

TFLite for microcontroller generates C byte arrays for TFLite models using standard tools and stores them in read-only program memory [66]. Models will be 'model.cc'. Installing the XXD package creates a hex dump of the file and converts it to binary [67]. The unix command XXD generates a C source file with TFLite models as char arrays. TFLite models are turned into hex dumps with all parameters and architectures specified as function calls that pass activations between layers. The output is "unsigned char g_model" in a big file with few entry points. You can incorporate the created file in your program. Can then directly include and compile the file in IDE or toolchain [10]. Remember to modify array declarations to const for optimal memory efficiency on embedded devices [66].

6. RESULTS AND DISCUSSION

This section illustrates the results of our experiment in detail using a variety of metrics for each of DL model in training and evaluation phase. Followed by the results of the size of each DL model after compression in optimizations phase using post training quantization (DRQ, FIQ) and quantization aware-training, as in subsection 6.1. In addition to the evaluation of the DL models use a variety of metrics. This section is structured as follows: subsection 6.2 training and evaluation models results, subsection 6.3 optimization and conversion model. Finally, subsection 6.4 is the interpreter phase result.

6.1. Results of DL model’s training and evaluation

6.1.1. SqueezeNet model’s result

Table 8 shows the performance metrics used to evaluate and train the SqueezeNet model, which are based on loss and accuracy values. All three pre-processing methods-SSD, Dlip, and Haar-have their results described, the SSD approach outperformed the others in the SqueezeNet model in terms of performance. In both the training and evaluation phases, the SSD technique achieved an accuracy of 0.9825. In contrast, both the training and evaluation sets have a loss value of 0.0175. Accuracy and loss values were calculated between the training and assessment phases with respect to the number of epochs, as shown in Figure 8.

Table 8. Results of evaluating the five DL models on the ECP dataset for the main two categories, pedestrian and rider in terms of: precision, recall, F1-score, accuracy, and loss

Model	Category	Precision	Recall	F1-score	Accuracy	Loss
SqueezeNet	Pedestrian	99%	100%	100%	0.9825	0.0175
	Rider	100%	99%	99%		
AlexNet	Pedestrian	98%	98%	100%	0.9880	0.0120
	Rider	98%	100%	98%		
CNN	Pedestrian	96%	97%	96%	0.9750	0.0250
	Rider	96%	97%	97%		
MobileNet-V2	Pedestrian	98%	1.00	99%	0.9930	0.0070
MobileNet-V3	Pedestrian	100%	99%	100%	0.9985	0.0015
	Rider	100%	100%	100%		

On the basis of this outcome, we provide a detailed description of the performance results using precision, recall, and F1-score. Moreover, in regard to the confusion matrix. The outcomes of the SqueezeNet model evaluation, including precision, recall, and F1-score, are displayed in Table 8. Both the pedestrian and the rider have confusion matrices with values of 0.985 and 0.98, respectively.

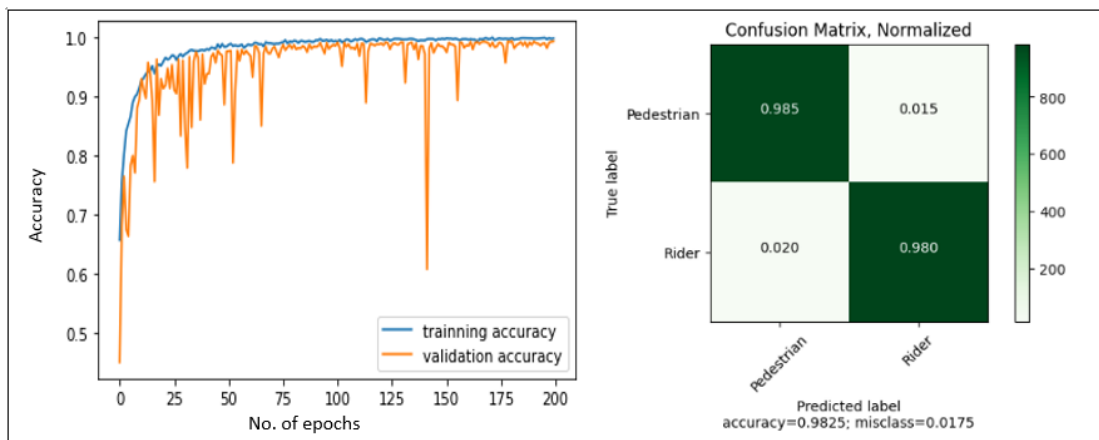


Figure 8. The accuracy and confusion matrix on the ECP dataset for the evaluation of the SqueezeNet DL model

6.1.2. AlexNet model’s results

Both the pedestrian and rider statuses were successfully addressed by the findings of the improved AlexNet deep model in the pedestrian instance. All of the pre-processing methods’ training and evaluation results for the AlexNet model were displayed in Table 8. In the evaluation phase, the AlexNet model that used the SSD approach performed better than the others. In terms of accuracy, it reached 0.988 during the evaluation phase. Nevertheless, 0.0120 is the loss value throughout training. The accuracy and loss of the AlexNet deep model throughout evaluation and training were displayed in Figure 9. Accordingly, we included additional outcome metrics including recall, accuracy, F1-score, and confusion matrix for it in Table 8.

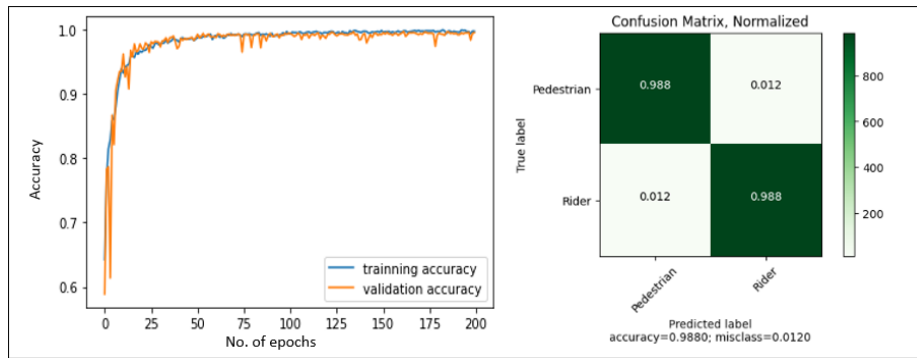


Figure 9. The accuracy and confusion matrix on the ECP dataset for the evaluation of the AlexNet DL model

6.1.3. CNN model's results

With minimal architectural complexity, the lightweight CNN model was able to detect pedestrian status with great performance. The outcomes of the CNN model with each pre-processing approach are displayed in Table 8. In the evaluation phase, the CNN model that used the SSD approach had the best accuracy performance with 0.975 and a loss of 0.0250. Figure 10 shows the loss value, training accuracy, and evaluation accuracy of the CNN deep model. Extend the greatest performance outcome from the CNN model by utilizing the SSD approach. Table 8 shows the outcomes of other measures that were added to it, including recall, precision, and F1-score. Figure 10 also served as the introduction of the confusion matrix. Both the pedestrian and rider of status confusion matrices for the ECP dataset demonstrated excellent performance, with scores of 0.97 and 0.98, respectively.

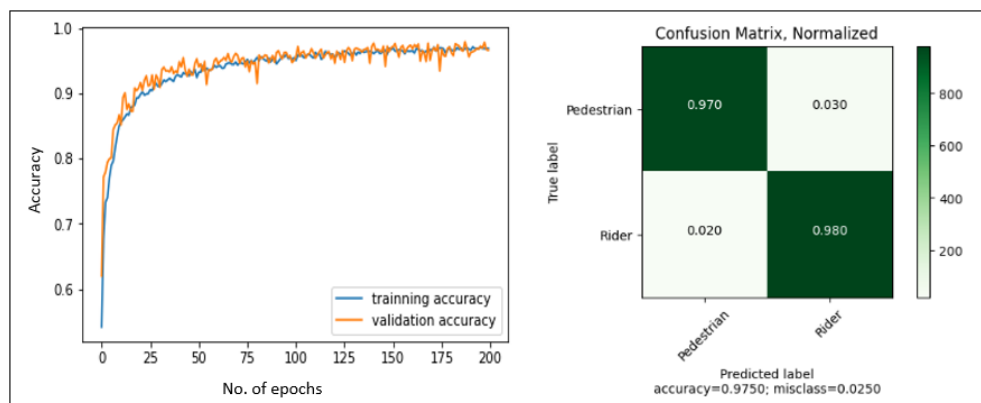


Figure 10. The accuracy and confusion matrix on the ECP dataset for the evaluation of the CNN DL model

6.1.4. MobileNet-V2 model's results

The pre-trained MobileNet-V2 deep model performed admirably in the task of pedestrian and rider detection. Training and evaluation results for the MobileNet-V2 model using SSD, Dlip, and Haar pre-processing methods are displayed in Table 8. In terms of evaluation phase performance, the MobileNet-V2 model employing the SSD approach surpasses the prior three methods. In terms of accuracy, it reached 0.993 throughout the evaluation phase. The evaluation, however, yields a loss value of 0.0070. As the number of epochs increased, Figure 11 showed the loss value, training accuracy, and assessment accuracy of the MobileNet-V2 deep model. We displayed the additional metrics for the SSD approach as F1-score, recall, and precision based on these results in Table 8. Nevertheless, as demonstrated in Figure 11, the confusion matrices of the MobilNet-V2 model based on the pedestrian category are 0.998, while those with the rider category are 0.988.

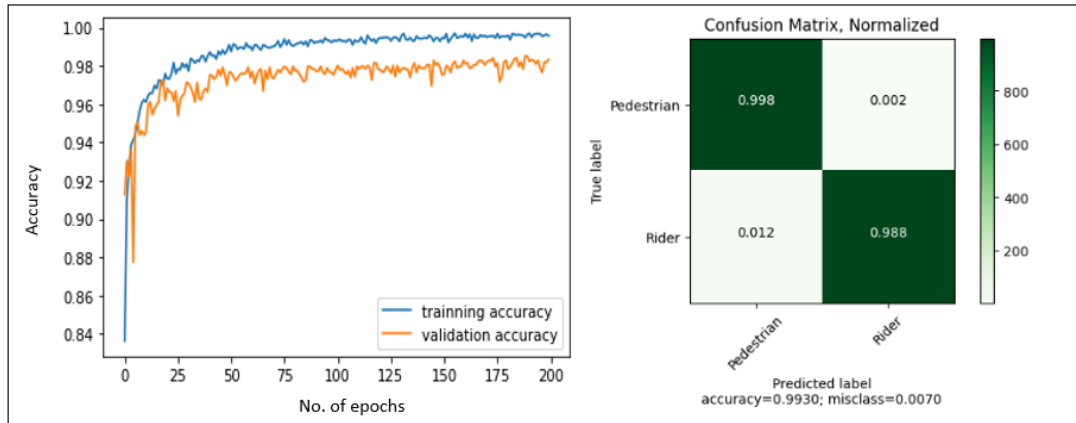


Figure 11. The accuracy and confusion matrix on the ECP dataset for the evaluation of the MobileNet-V2 DL model

6.1.5. MobileNet-V3 model’s results

Training and evaluation results for all pedestrian detection methods are very good with the pre-trained tiny MobileNet-V3 deep model. While being evaluated, the SSD approach achieved a high level of accuracy. The Dlip approach and the SSD method came next. Figure 12 shows that it achieved a training accuracy of 0.9956 and an evaluation accuracy of 0.9885. Table 8 shows that the loss values for the training phase are 0.0015 and for the evaluation phase they are 0.0015. Use the outcome from the prior table as a foundation. We also detailed other outcomes for the SSD approach, including recall, precision, F1-score, and confusion matrix. Table 8 displayed the outcomes for F1-score, precision, and recall. Figure 12 also displayed the confusion matrix result.

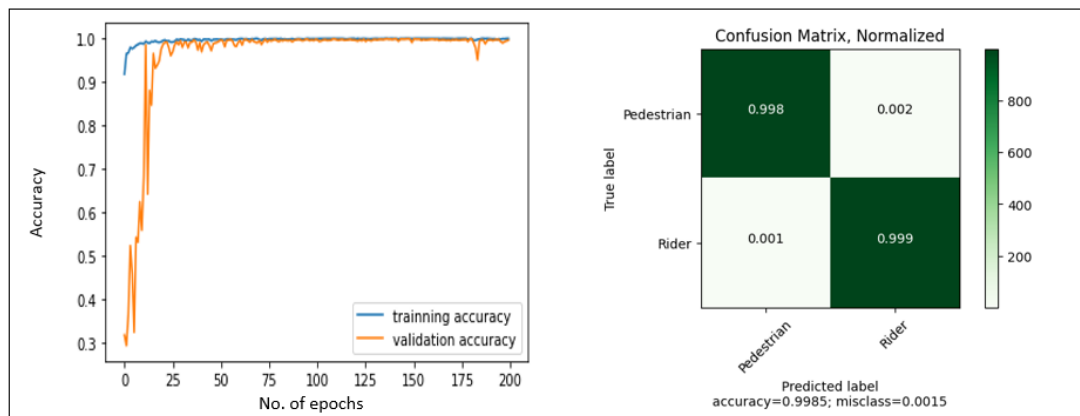


Figure 12. The accuracy and confusion matrix on the ECP dataset for the evaluation of the MobileNet-V3 DL model

The results of evaluating the five DL models on the ECP dataset for all the objects’ instances in the rider category in terms of accuracy and loss is illustrated on Table 9. As shown the MobileNet-V3 outperforms the other four models for classifying the rider instances in the seven objects: bicycle, motorbike, scooter, tricycle, wheelchair, buggy, and co-rider. As concluded from the table, MobileNet-V3 DL model outperform the other model with accuracy 0.9829 and loss equal to 0.017 while the CNN DL model gives the lowest results with accuracy 0.9154 and loss 0.084. For this reason, we present the confusion matrix for evaluation of the lowest and highest models. Figure 13 shows the confusion matrix for evaluation of the CNN DL model on the ECP dataset for all the seven objects’ instances in the rider category, while Figure 14 shows the confusion matrix for evaluation of the MobileNet-V3 DL model.

Table 9. Results of evaluating the five DL models on the ECP dataset for all the objects' instances in the rider category in terms of accuracy and loss

Rider objects	SqueezeNet		AlexNet		CNN		MobileNet-V2		MobileNet-V3	
	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
Bicycle	0.8964	0.084	0.9308	0.081	0.8611	0.088	0.9854	0.023	0.9861	0.014
Motorbike	0.9334	0.051	0.9452	0.051	0.9024	0.069	0.9469	0.013	0.9469	0.016
Scooter	0.9815	0.064	0.9775	0.064	0.9915	0.084	0.9965	0.017	0.9965	0.012
Tricycle	0.9632	0.054	0.9632	0.054	0.8934	0.085	0.9632	0.051	0.9632	0.025
Wheelchair	0.9337	0.021	0.9397	0.077	0.9045	0.097	0.9955	0.019	0.9960	0.018
Buggy	0.9469	0.081	0.8631	0.084	0.9308	0.089	0.9469	0.017	0.9960	0.017
Co-Rider	0.9492	0.012	0.9492	0.032	0.9334	0.078	0.9962	0.022	0.9970	0.018
Average	0.9434	0.052	0.9383	0.063	0.9154	0.084	0.9758	0.023	0.9829	0.017

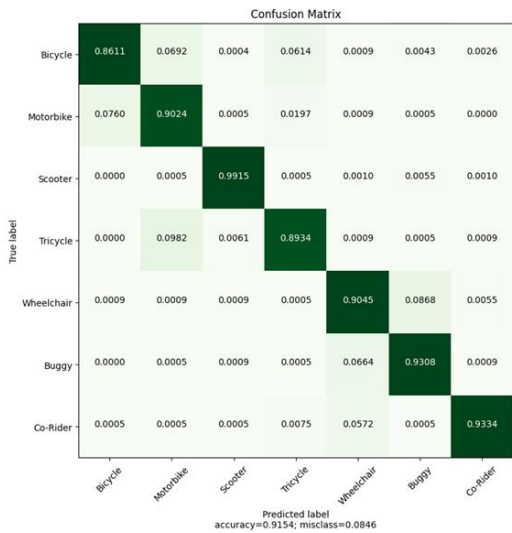


Figure 13. The confusion matrix for evaluation of the CNN DL model on the ECP dataset for all the objects' instances in the rider category

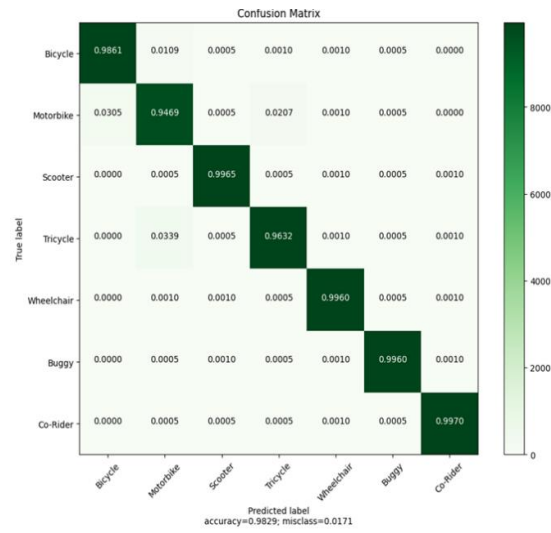


Figure 14. The confusion matrix for evaluation of the MobileNet-V3 DL model on the ECP dataset for all the objects' instances in the rider category

6.2. Optimization and conversion models result

The optimizing and conversion models' phase aims to compress and converting of 32 floating-point DL models size that is already trained to be Int8 integers using quantization methods. After that, convert the models to TFLite format. The subsections present the of size the original models which means the trained model. As well as the size of models after performing quantization and conversion to the TFLite model.

6.2.1. SqueezeNet model's size results

SqueezeNet deep model, it shows superior size results where the measured size of the SqueezeNet model is 3.092 MB. In the optimization and conversion model phase, the DRQ obtained best measure size of SqueezeNet with 0.161 MB. Means the DRQ has decreased in size by approximately 94.7% compared to the original model. Followed by FIQ with 0.163MB and the QAT with 0.161 MB of size as shown in Table 9.

6.2.2. AlexNet model's size results

Conversely, when we made some structural changes to the AlexNet deep model, it demonstrated a significant decrease in size during the optimization and conversion phases when compared to the original model. All quantization methods converge on the same reduction outcomes. Table 10 shows that the original deep model's size was decreased by 92.5% (from 9.272 MB to 0.69 MB) using the DRQ approach. Then came QAT, which had a size reduction of 0.694 MB and FIQ, which had a size reduction of 92.3% of the original model. The result is a 92.5% reduction in memory footprint achieved using the AlexNet deep model.

Table 10. The size of the five DL models after optimization and conversion

DL models	Original model (MB)	QAT (MB)	FIQ (MB)	DRQ (MB)	Decreased (%) from the original
SqueezeNet model	3.092	0.165	0.163	0.161	94.7%
AlexNet model	9.272	0.694	0.693	0.69	92.5%
CNN model	1.664	0.072	0.079	0.07	95.7%
MobileNet-V2	7.493	2.635	1.958	1.95	73.9%
MobileNet-V3	6.995	4.074	1.824	1.813	74.1%

6.2.3. CNN model’s size results

During the optimization phase, the CNN deep model demonstrated good results in reducing the size of DL models. The size of the CNN model produced better results than other DL models, and the CNN model ended up with the least size. Table 9 shows that the DRQ approach obtained 0.07 MB, while the CNN model had few megabytes. The original model’s size was cut in half, from 1.664 MB to just 0.07 MB, a reduction of 95.7%. Similarly, for DRQ results with 0.072 MB and 0.079 MB, respectively, QAT and FIQ converge. The size of the TFLite CNN deep model was just a few kilobytes, and it had a very high accuracy rate. This means it could work well even on low-powered IoT devices.

6.2.4. MobileNet-V2 model’s size results

The outcomes of the optimization and conversion phase for the pre-trained MobileNet-V2 deep model were displayed in Table 9. The DRQ approach resulted in a 73.9% reduction in the size of the MobileNet-V2 model, going from 7.493 MB to 1.95 MB. In the end, the FIQ approach reached a size of 1.958 MB, whereas the QAT method reached 2.635 MB.

6.2.5. MobileNet-V3 model’s size results

Table 9 shows that the pre-trained MobileNet-V3 deep model is 6.995 MB less than the original model. Both the DRQ and FIQ approaches allowed the MobileNet-V3 to gain significant size after quantization and conversion; the former achieved 1.824 MB and the latter achieved 1.813 MB. This resulted in a reduction in the original model size of about 74.1 percent. In contrast, the MobileNet-V3 deep model in QAT weighed in at 4.074 MB.

6.3. The results of the interpreter phase

In converting and optimization phase we optimize and convert model to TFLite format. After that, we used the interpreter to evaluate the performance accuracy of TFLite models after using the quantization. Notice, in QAT first training and evaluation the DL models then converted to TFLite format and used the interpreter to evaluate the DL models.

6.3.1. SqueezeNet deep model results

SqueezeNet model performance accuracy after conversion to TFLite format for all quantization methods was shown in Table 11. First, we quantize all model layers in quantization aware-training training and assessment. The daytime Dlip approach had the highest accuracy, 99.56% and 98.27% in training and evaluation. The training phase loss is 0.0245 and the evaluation phase is 0.0935. SSD with day images had higher accuracy in DRQ, FIQ, and quantization aware-training for TFLite model phase evaluation utilizing interpreter (98.27%, 99.29%, and 99.24%). Which reduced accuracy by less than 1% compared to the old model’s 99.56%. Dlip methods with day images follow. Inference time was lowest for Haar technique night.

Table 11. Quantization results of SqueezeNet model

Model	Quantization method	Day			Night			
		SSD	Dlip	Haar	SSD	Dlip	Haar	
SqueezeNet model	Training:	Accuracy:	99.22%	99.56%	98.57%	98.77%	98.88%	98.36%
		Loss:	0.0321	0.0245	0.0648	0.0455	0.0392	0.0612
	Evaluation:	Accuracy:	98.27%	97.78%	94.98%	94.52%	97.52%	92.06%
		Loss:	0.0751	0.0935	0.1814	0.1996	0.1125	0.3431
	Interpreter:	Accuracy:	98.27%	97.77%	94.98%	94.52%	97.52%	92.06%
		Time:	8.9875	8.8828	3.9732	5.4673	6.8101	3.4623
	FIQ Interpreter:	Accuracy:	99.29%	98.65%	95.10%	98.26%	98.14%	94.50%
		Time:	9.2829	10.9602	4.2913	6.6926	8.0926	4.0766
	DRQ Interpreter:	Accuracy:	99.24%	98.86%	95.34%	98.26%	98.07%	94.65%
		Time:	11.4784	9.2534	4.0842	5.4771	6.9075	3.5623

6.3.2. AlexNet deep model results

The accuracy of the AlexNet deep model's performance for all quantization methods after conversion to TFLite format is provided in Table 12. For daytime images, the TFLite AlexNet model's results utilizing the SSD approach are superior to those of conventional quantization methods. The quantization aware-training approach also managed a loss of 0.6256 and an accuracy of 97.30%. With an accuracy of 97.26% and 96.98%, respectively, the QAT and DRQ got the best results in the interpretation phase. In comparison to the old model's accuracy of 97.3% during the evaluation phase, these findings show that there was no degradation in accuracy. In addition, the interpreter's accuracy was 61.70% during the FIQ phase, which was the lowest.

Table 12. Quantization results of AlexNet deep model

Model	Quantization method		Day			Night			
			SSD	Dlip	Haar	SSD	Dlip	Haar	
AlexNet model	QAT	Training:	Accuracy:	98.92%	98.14%	98.77%	99.22%	91.36%	98.67%
			Loss:	0.2139	0.3546	0.1943	0.1017	0.2572	0.2336
		Evaluation:	Accuracy:	97.30%	96.58%	94.15%	96.18%	92.02%	94.02%
			Loss:	0.6256	0.9501	1.986	0.9207	1.8679	2.9832
		Interpreter:	Accuracy:	97.29%	96.57%	94.14%	95.30%	92.02%	94.01%
			Time:	9.8782	5.8422	3.2779	4.9489	4.7229	4.6784
	FIQ	Interpreter:	Accuracy:	61.70%	75.76%	62.24%	69.93%	78.46%	56.50%
			Time:	46.0652	38.1622	24.1734	29.1898	22.5477	24.5777
	DRQ	Interpreter:	Accuracy:	96.98%	93.27%	92.35%	94.09%	92.12%	90.67%
			Time:	25.6976	23.3022	7.1562	8.9807	8.9907	7.8013

6.3.3. CNN deep model results

Following the conversion of the CNN deep model to the TFLite format, Table 13 demonstrated the performance accuracy of the CNN deep model for all quantization methods tested. During the evaluation phase of the quantization aware-training process, the results obtained by utilizing the Dlip technique with day photos achieved the best performance outcomes. During the training phase, it achieved an accuracy of 96.68%, and during the evaluation phase, it achieved an accuracy of 96.13%. When it comes to training and evaluation, the loss values are 0.1622 and 0.2357, respectively. On the other hand, the accuracy results of the CNN model that uses the quantization method are comparable during the interpreter phase. The outcomes of the quantization aware-training sessions were 96.89%, 96.45% for the FIQ, and 96.67% for the DRQ.

Table 13. Quantization results of CNN deep model

Model	Quantization method		Day			Night			
			SSD	Dlip	Haar	SSD	Dlip	Haar	
CNN model	QAT	Training	Accuracy	96.16%	96.68%	95.19%	96.38%	95.42%	95.48%
			Loss	0.1795	0.1622	0.2357	0.2333	0.3456	0.3425
		Evaluation	Accuracy	96.90%	96.13%	94.27%	94.00%	93.42%	91.45%
			Loss	0.1288	0.2313	0.2867	0.5527	0.5316	0.5313
		Interpreter	Accuracy	96.89%	96.18%	94.38%	94.08%	93.41%	91.45%
			Time	14.8474	9.6489	4.3466	4.9809	4.9978	4.0254
	FIQ	Interpreter	Accuracy	96.45%	96.52%	93.30%	94.69%	94.51%	91.29%
			Time	14.4274	10.8634	4.8818	6.1945	5.6767	5.1152
	DRQ	Interpreter	Accuracy	96.67%	96.58%	93.54%	94.86%	94.51%	91.45%
			Time	7.4934	6.4582	2.6295	2.8551	2.7644	3.3932

6.3.4. MobileNet-V2 deep model results

The model optimization and conversion phase was a success for the pre-trained MobileNet-V2 model. Table 14 displays the excellent results obtained by the TFLite MobileNet-V2 deep model. By utilizing the Dlip approach with day photos, the TFLite MobileNet-V2 model achieves superior results compared to other quantization methods. Its accuracy in the training phase was 99.66% and in the evaluation phase it was 98.12% in QAT. On the other hand, during the evaluation phase, the loss value is 0.0732 and during training it is 0.0236. Even in the interpreter phase, DRQ got the best result with an accuracy rate of 99.51%. FIQ also got there, with an accuracy rate of 99.46%. Specifically, 62.0635 and 59.3225 seconds are the inference times. When compared to the original model's accuracy of 99.66%, their results were impressive. In contrast, the QAT approach was able to get 98.84% accuracy in 92.53 seconds of inference time.

Table 14. Quantization results of MobileNet-V2 deep model

Model	Quantization method			Day			Night		
				SSD	Dlip	Haar	SSD	Dlip	Haar
MobileNet-V2	QAT	Training	Accuracy	99.64%	99.66%	99.41%	99.11%	99.56%	98.67%
			Loss	0.0227	0.0236	0.0298	0.0345	0.0224	0.0472
		Evaluation	Accuracy	98.89%	98.12%	98.28%	96.87%	98.13%	96.43%
			Loss	0.0386	0.0732	0.0567	0.1221	0.0815	0.1299
	Interpreter	Accuracy	98.84%	98.17%	98.14%	96.86%	98.12%	96.77%	
		Time	92.5306	62.0635	65.7546	53.1376	51.8511	41.7846	
	FIQ	Interpreter	Accuracy	99.46%	99.03%	98.83%	97.56%	97.65%	97.69%
			Time	96.8919	59.3225	64.5348	54.3569	51.0124	44.1712
	DRQ	Interpreter	Accuracy	99.51%	98.91%	98.90%	97.65%	97.56%	97.69%
			Time	110.5129	75.8372	74.8445	51.2149	49.2187	45.3419

6.3.5. MobileNet-V3 deep model results

The pre-trained MobileNet-V3 deep model obtained well performance in optimization phase using quantization methods and after converted to the TFLite model to detect the pedestrian objects. Due to the quantized whole model does not support in MobileNet-V3 model, wherein the architecture of MobileNet-V3 contains TFOPLambda layer. Thus, we quantized only the last layer which are Dense layer. Since the SSD method with day images obtained best performance result in with (99.94%) and (99.60%) of accuracy in training and evaluation phases respectively. However, the loss values are (0.0032) and (0.0154). In Interpreter phase, the results of performance pre-trained MobileNet-V3 model using SSD method with day images gained best performance results. DRQ achieved the highest accuracy with (99.64%), followed by QAT with (99.60%). On the contrary, the FIQ achieved low accuracy with (96.01%) as shown in Table 15.

Table 15. Quantization results of MobileNet-V3 deep model

Model	Quantization method			Day			Night		
				SSD	Dlip	Haar	SSD	Dlip	Haar
MobileNet-V3	QAT	Training	Accuracy	99.94%	99.68%	99.03%	99.63%	99.40%	99.28%
			Loss	0.0032	0.0061	0.0306	0.0098	0.0161	0.0194
		Evaluation	Accuracy	99.60%	98.80%	91.76%	97.48%	97.51%	97.10%
			Loss	0.0154	0.0842	0.5509	0.1696	0.1362	0.2951
	Interpreter	Accuracy	99.60%	98.85%	91.63%	97.47%	97.40%	97.09%	
		Time	87.7556	66.8714	36.6223	55.4781	41.5858	31.0865	
	FIQ	Interpreter	Accuracy	96.01%	99.20%	97.96%	99.39%	98.00%	97.09%
			Time	89.8324	69.9036	0.00453	55.0924	43.4813	31.8549
	DRQ	Interpreter	Accuracy	99.64%	99.14%	97.96%	99.39%	98.10%	97.86%
			Time	62.5053	50.562	29.0627	40.9614	32.451	24.7911

7. DISCUSSION AND COMPARISON

When it comes to pedestrian detection using IoT devices, the complexity and high processing costs of DL models provide the biggest obstacle. As a result, investing in training takes a lot of time and drains the resources of IoT devices. To improve upon the memory and complexity efficiency of previous DL models for pedestrian and rider detection, we presented five compact DL models in this study. The SqueezeNet, AlexNet, and CNN models are three of our lightweight DL implementations. Also, we're making use of two DL models-MobileNet-V3 and MobileNet-V2-that were previously trained on the ECP dataset. These models can distinguish between pedestrians and riders in a variety of lighting and environmental situations. Riders include seven distinct types of vehicles: bicycles, motorbikes, scooters, tricycles, wheelchairs, buggy, and co-riders. In addition, pre-processing algorithms like SSD, Dlip, and Haar were utilized, which could identify pedestrians in various scenarios. The training and evaluation phases also assessed the proposed models for recall, accuracy, precision, and loss, among other metrics. We evaluated the performance of the converted to TFLite models using accuracy in the interpreter phase. Aside from the models' sizes.

7.1. The training and evaluation phase

We began by testing the DL models' accuracy during the assessment phase of the training and evaluation process. All of the DL models' performance accuracy is displayed in Figure 15. In the MobileNet-V2, SqueezeNet, and MobileNet-V3 models, the best results and highest performance accuracy of deep models were obtained by employing the SSD approach with day images. The corresponding percentages were 99.27%, 98.86%, and 98.69%. Furthermore, the SSD technique demonstrated superior accuracy with 98.67% using nighttime images in the pre-trained MobileNet-V3 model. Based on the results, it appears that

the SSD approach yielded the most accurate results. Whereas, the SSD approach can pinpoint pedestrians and riders in photos with remarkable accuracy. While the SSD technique produced an accuracy of 86.77% with night photos, the Haar method yielded the lowest performance accuracy for the AlexNet model (88.33%) when applied to day images. Figure 16 also shows the average amount of time it takes for the three interpreters to work with each of the five DL models that were applied to the three SSD, Dlip, and Haar approaches for ECP images taken during the day and night, respectively, using the values from Tables 11 to 15.

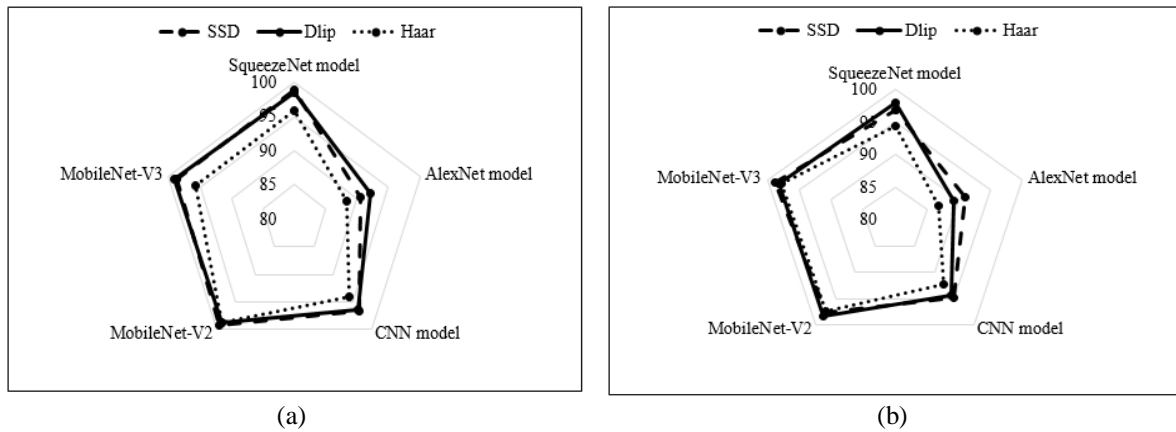


Figure 15. Average accuracy between the three interpreters for the five DL models applied on the three SSD, Dlip, and Haar methods (a) ECP images during day and (b) ECP images during night

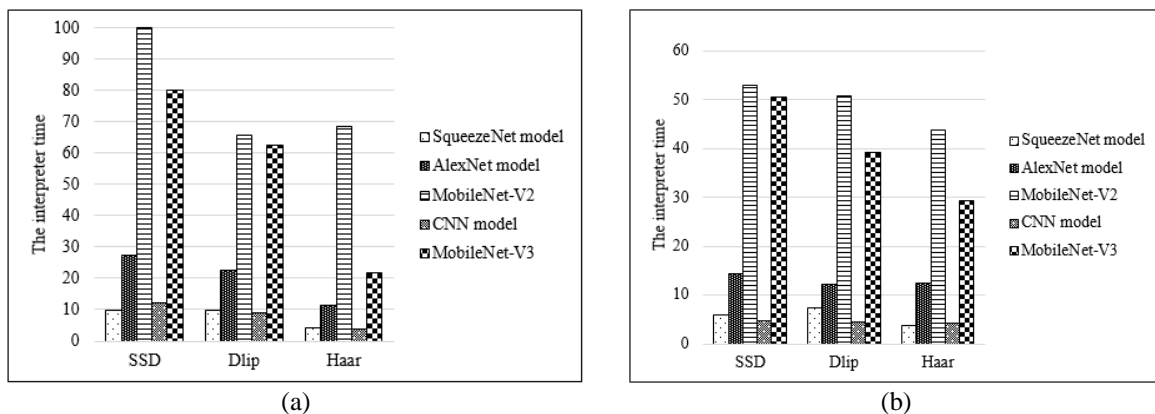


Figure 16. Average time between the three interpreters for the five DL models applied on the three SSD, Dlip, and Haar methods (a) ECP images during day and (b) ECP images during night

7.2. The optimization and conversion model phase

We found that when we compared the accuracy with the model size, the results demonstrated good performance, and we also noticed a low variance. The quantized approaches significantly reduce the model size with a slight loss of accuracy, as demonstrated in Figure 17. Using fixed-point on 8-bit integers during implementation, in particular, reduces inference time and average power usage. Shorter inference times are intriguing because they allow one to either reduce the working frequency of the microcontroller, which is a critical characteristic in IoT devices for power consumption. Furthermore, there is a notable decrease in memory footprint while executing with 8-bit integers.

The CNN model had the smallest model size with 0.07 MB when utilizing DRQ, according to our data. However, when the DRQ approach was used, the accuracy result did not degrade. The SqueezeNet model came next, achieving a 94.7% reduction in model size with an approximate after-quantization size of 0.161 MB. Also, the AlexNet model size dropped significantly from 9.272 MB to about 0.69 MB when using the quantization method. However, with 1.95 MB and 1.831 MB, respectively, the pre-trained models MobileNet-V2 and Mo-bileNet-V3 had the biggest model sizes compared to our development model. With a

performance accuracy of 99.60%, the MobileNet-V2 model outperformed the MobileNet-V3 model, which achieved 98.89%. After converting to the TFLite model, however, the MobileNet-V2 model maintained a high level of accuracy while achieving excellent performance. After being quantized and converted, MobileNet-V3 produced a significant decrease in accuracy and a high variation.

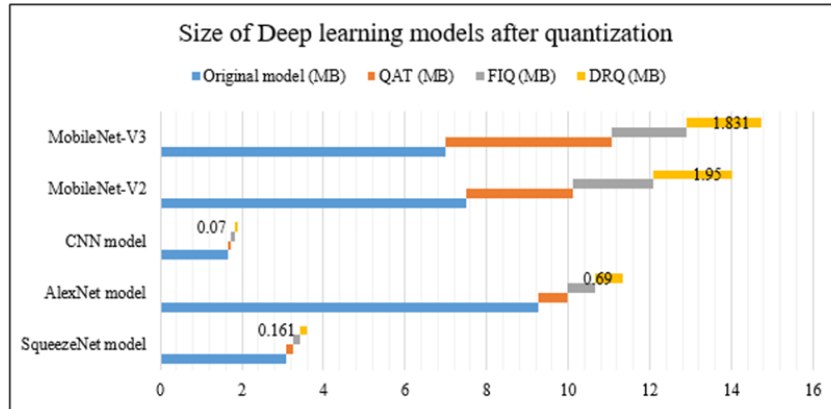


Figure 17. The size of the five DL models for the original models and after optimization and conversion phase

7.3. The interpreter phase

We conducted experiments to determine how effectively the DL models fared after we applied the quantization techniques. Compared to the accuracy of the initial model without optimization, our experiment got high results in the interpreter phase. In quantization aware-training method, we presented the accuracy of evaluation phase for all DL models. SqueezeNet quantifies all model layers in Quantization aware-training and assessment. The daytime Dlip method was most accurate in training and evaluation, 99.56% and 98.27%. Training phase loss is 0.0245, assessment phase 0.0935. SSD-during day, TFLite model phase evaluation using interpreter had higher DRQ, FIQ, and quantization aware-training accuracy (98.27%, 99.29%, and 99.24%). This reduced accuracy by less than 1% from the old model’s 99.56%. Day-image Dlip approaches follow. For Haar technique night, inference time was lowest.

SSD quantization achieves better results than conventional quantization in the TFLite AlexNet model. Quantization-aware training had 0.6256 loss and 97.30% accuracy. QAT and DRQ had the highest interpretation accuracy at 97.26% and 96.98%. These statistics show no decline in accuracy compared to the old model’s 97.3% during evaluation. FIQ was the lowest phase for interpreter accuracy at 61.70%.

The CNN model’s Quantization aware-training assessment phase yielded the greatest results using the Dlip approach with day photographs. During training, it had 96.68% accuracy and during evaluation, 96.13%. Training and evaluation loss values are 0.1622 and 0.2357. However, the CNN model with quantization has comparable Interpreter accuracy. Quantization aware-training sessions yielded 96.89%, 96.45% for the FIQ, and 96.67% for the DRQ.

The pre-trained Mo-bileNet-V2 model optimized and converted successfully. The Dlip approach with day photographs give the TFLite Mo-bileNet-V2 model better quantization results than previous methods. The training phase has 99.66% accuracy and the evaluation phase 98.12% in QAT. However, evaluation loss is 0.0732 and training loss is 0.0236. Even in interpreter, DRQ had the best accuracy at 99.51%. With 99.46% accuracy, FIQ also qualified. Inference times are 62.0635 and 59.3225 seconds. Compared to the original model’s 99.66% accuracy, their results were excellent. In contrast, QAT achieved 98.84% accuracy in 92.53 seconds.

The pre-trained MobileNet-V3 deep model performed well in quantization for optimization and then converted to the TFLite model to recognize pedestrian objects. TFOPLambda layer in MobileNet-V3 architecture prevents quantized entire model support. We quantized only the dense final layer. The SSD approach using day photos performed best in training and evaluation with 99.94% and 99.60% accuracy. However, loss values are 0.0032 and 0.0154. Performance pre-trained MobileNet-V3 model utilizing SSD technique with day images performed best in interpreter phase. DRQ was most accurate (99.64%), followed by QAT (99.60%). FIQ accuracy was low (96.01%).

7.4. Comparison between our models and the previous studies

Using data from section 3's previous studies, Table 16 compares the accuracy of several pedestrian identification methods using deep learning models. We discovered that their experimental accuracy ranged from 80.1% to 98.8% by comparing their performance in the assessment phase with earlier studies on pedestrian identification datasets. Our experimental results in this study surpass those of the most recent, comprehensive research in the field. Using an average of the SSD, Dlip, and Haar approaches, our SqueezeNet model achieved a result of 98.93% for day images and 98.09% for night images. Both the previous research and all the other models used in this study were outperformed by the MobileNet-V3 and MobileNet-V2 models, which achieved the highest results (99.66% for day images and 99.56% for night photos, respectively).

Table 16. Comparing results from previous studies throughout the evaluation process for accuracy

Reference	DL Models	Performance (accuracy)
[31]	Vgg-16	98.8%
[32]	AlexNet	80.1%
	GoogLeNet	80.3%
[34]	R-CNN	85.5%
[44]	DCGAN	80.7%
Our proposed	SqueezeNet-day images	98.93%
	SqueezeNet-night images	98.09%
	AlexNet-day images	93.25%
	AlexNet-night images	92.65%
	CNN-day images	96.73%
	CNN-night images	94.80%
	MobileNet-V2-day images	99.27%
	MobileNet-V2-night images	98.24%
	MobileNet-V3-day images	99.66%
	MobileNet-V3-night images	99.56%

8. CONCLUSION

As we discussed in our recent publication titled "TinyML: enabling of inference deep learning models on ultra-low-power IoT edge device," we use TinyML in this research to solve the problems of integrating DL models on IoT devices in smart cities. In order to facilitate integration on low-power, resource-constrained IoT devices, five lightweight DL models were assessed. In order to identify the rider and pedestrian, three DL models-SqueezeNet, AlexNet, and CNN-were created by adapting two pre-train models, MobileNet-V2 and MobileNet-V3.

After comparing their performance in the assessment phase with past studies on pedestrian identification datasets, we found that their experimental accuracy ranged from 80.1% to 98.8%. This was obtained by comparing their performance with the datasets. The results of our experiments in this study are superior to those of the most recent and thorough research that has been conducted in this area. The SqueezeNet model that we developed attained a result of 98.93% for daytime photographs and 98.09% for nighttime photographs by utilizing an average of the SSD, Dlip, and Haar analysis methods. The results of the MobileNet-V3 and MobileNet-V2 models, which achieved the highest results (99.66% for day images and 99.56% for night shots, respectively), were superior to those of the prior research as well as all of the other models that were utilized in this study.

As far as we know, no previous studies on pedestrian detection have utilized TinyML. Our work demonstrates that the CNN architecture has a smaller model size of 0.07 MB compared to other TinyML-related studies. Additionally, our revised SqueezeNet design achieved a reduced size of 0.161MB compared to the original SqueezeNet model. The adjusted AlexNet model reached a size of 0.69 MB, while the pre-trained MobileNet-V2 and MobileNet-V3 models achieved sizes of 1.95 MB and 1.831 MB respectively. The results show that there is no loss of accuracy after applying an optimization strategy of around less than 1%. The experiment results indicate that it has significant potential to operate efficiently on resource-limited IoT devices such as microcontrollers.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Norah N. Alajlan	✓	✓	✓	✓	✓			✓	✓	✓	✓			
Abeer I. Alhujaylan		✓					✓	✓		✓			✓	✓
Dina M. Ibrahim	✓	✓	✓	✓		✓				✓	✓	✓	✓	

C : Conceptualization
 M : Methodology
 So : Software
 Va : Validation
 Fo : Formal analysis

I : Investigation
 R : Resources
 D : Data Curation
 O : Writing - Original Draft
 E : Writing - Review & Editing

Vi : Visualization
 Su : Supervision
 P : Project administration
 Fu : Funding acquisition

CONFLICTS OF INTEREST

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author, [Dina M. Ibrahim], upon reasonable request.

REFERENCES





- [1] S. Iftikhar, Z. Zhang, M. Asim, A. Muthanna, A. Koucheryavy, and A. A. A. El-Latif, "Deep learning-based pedestrian detection in autonomous vehicles: substantial issues and challenges," *Electronics (Switzerland)*, vol. 11, no. 21, p. 3551, Oct. 2022, doi: 10.3390/electronics11213551.
- [2] F. He, P. K. Olia, R. J. Oskouei, M. Hosseini, Z. Peng, and T. BaniRostam, "Applications of deep learning techniques for pedestrian detection in smart environments: a comprehensive study," *Journal of Advanced Transportation*, vol. 2021, pp. 1–14, Oct. 2021, doi: 10.1155/2021/5549111.
- [3] C.-H. Hsia, H.-C. Peng, and H.-T. Chan, "All-weather pedestrian detection based on double-stream multispectral network," *Electronics*, vol. 12, no. 10, p. 2312, May 2023, doi: 10.3390/electronics12102312.
- [4] T.-Y. Chow, K.-H. Lee, and K.-L. Chan, "Detection of targets in road scene images enhanced using conditional GAN-based dehazing model," *Applied Sciences*, vol. 13, no. 9, p. 5326, Apr. 2023, doi: 10.3390/app13095326.
- [5] N. N. Alajlan and D. M. Ibrahim, "TinyML: enabling of inference deep learning models on ultra-low-power IoT edge devices for AI applications," *Micromachines*, vol. 13, no. 6, p. 851, May 2022, doi: 10.3390/mi13060851.
- [6] N. N. Alajlan and D. M. Ibrahim, "DDD TinyML: a TinyML-based driver drowsiness detection model using deep learning," *Sensors*, vol. 23, no. 12, p. 5696, Jun. 2023, doi: 10.3390/s23125696.
- [7] N. N. Alajlan and D. M. Ibrahim, "TinyML: Adopting tiny machine learning in smart cities," *Journal of Autonomous Intelligence*, vol. 7, no. 4, Jan. 2024, doi: 10.32629/jai.v7i4.1186.
- [8] C. R. Banbury *et al.*, "Benchmarking TinyML systems: challenges and direction," *arXiv preprint arXiv:2003.04821v3*, 2020, [Online]. Available: <http://arxiv.org/abs/2003.04821>.
- [9] J. Lin, W. M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," *Advances in Neural Information Processing Systems*, vol. 2020-December, 2020.
- [10] P. Warden and D. Situnayake, "TinyML: machine learning with TensorFlow Lite on Arduino and ultra-low-power microcontrollers," *Mike Louki; O'Reilly Media: Sebastopol*, pp. 1–504, 2019, [Online]. Available: <https://www.oreilly.com/library/view/tinyml/9781492052036/>.
- [11] P. P. Ray, "A review on TinyML: state-of-the-art and prospects," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, Apr. 2022, doi: 10.1016/j.jksuci.2021.11.019.
- [12] "Machine learning for resource-constrained edge devices." <https://github.com/Microsoft/EdgeML>.
- [13] C. A. Estrebou, M. Fleming, M. D. Saavedra, F. Adra, and A. E. De Giusti, "Lightweight convolutional neural networks frameworks for really small TinyML devices," in *Communications in Computer and Information Science*, vol. 1532 CCIS, 2022, pp. 3–16.
- [14] TensorFlow, "TensorFlow Lite," *TensorFlow*, 2024. <https://www.tensorflow.org/lite/guide?hl=id> (accessed Jan. 10, 2025).
- [15] "Google," *Google Colab*. <https://colab.research.google.com/notebooks/intro.ipynb> (accessed Jan. 10, 2025).
- [16] T. Carneiro, R. V. M. Da Nobrega, T. Nepomuceno, G. Bin Bian, V. H. C. De Albuquerque, and P. P. R. Filho, "Performance analysis of Google colab as a tool for accelerating deep learning applications," *IEEE Access*, vol. 6, pp. 61677–61685, 2018, doi: 10.1109/ACCESS.2018.2874767.
- [17] "Python data analysis library," *Pandas*. <https://pandas.pydata.org> (accessed Jan. 10, 2025).
- [18] Q. Abbas and A. Alsheddy, "Driver fatigue detection systems using multi-sensors, smartphone, and cloud-based computing platforms: a comparative analysis," *Sensors*, vol. 21, no. 1, p. 56, Dec. 2020, doi: 10.3390/s21010056.
- [19] "NumPy," *NumPy*. Jan. 10, 2025.
- [20] Matplotlib, "Visualization with Python," *Matplotlib Documentation*, 2023. <https://matplotlib.org/> (accessed Jan. 10, 2025).
- [21] G. Signoretto, M. Silva, P. Andrade, I. Silva, E. Sisinni, and P. Ferrari, "An evolving TinyML compression algorithm for IoT environments based on data eccentricity," *Sensors*, vol. 21, no. 12, p. 4153, Jun. 2021, doi: 10.3390/s21124153.
- [22] Norah N. Alajlan and Dina M. Ibrahim, "Deep smart cities: a review of smart cities applications-based an inference of deep learning upon IoT devices," *Journal of Advanced Research in Applied Sciences and Engineering Technology*, vol. 47, no. 2, pp. 94–120, Jun. 2024, doi: 10.37934/araset.47.2.94120.

- [23] F. Wang, M. Zhang, X. Wang, X. Ma, and J. Liu, "Deep learning for edge computing applications: a state-of-the-art survey," *IEEE Access*, vol. 8, pp. 58322–58336, 2020, doi: 10.1109/ACCESS.2020.2982411.
- [24] M. Garifulla *et al.*, "A case study of quantizing convolutional neural networks for fast disease diagnosis on portable medical devices," *Sensors*, vol. 22, no. 1, p. 219, Dec. 2021, doi: 10.3390/s22010219.
- [25] D. Kwon, R. Malaiya, G. Yoon, J.-T. Ryu, and S.-Y. Pi, "A study on development of the camera-based blind spot detection system using the deep learning methodology," *Applied Sciences*, vol. 9, no. 14, p. 2941, Jul. 2019, doi: 10.3390/app9142941.
- [26] S. Bhattacharya, S. R. K. Somayaji, T. R. Gadekallu, M. Alazab, and P. K. R. Maddikunta, "A review on deep learning for future smart cities," *Internet Technology Letters*, vol. 5, no. 1, Jan. 2022, doi: 10.1002/itl2.187.
- [27] S. S. I, R. Ramli, M. A. Azri, M. Aliff, and Z. Mohammad, "Raspberry Pi based driver drowsiness detection system using convolutional neural network (CNN)," in *2022 IEEE 18th International Colloquium on Signal Processing & Applications (CSPA)*, May 2022, pp. 30–34, doi: 10.1109/CSPA55076.2022.9781879.
- [28] "Google. Model optimization," *TensorFlow Lite*. https://www.tensorflow.org/lite/performance/model_optimization (accessed Jan. 10, 2025).
- [29] "Build and convert models | TensorFlow Lite," *TensorFlow Lite*. https://www.tensorflow.org/lite/microcontrollers/build_convert (accessed Jan. 10, 2025).
- [30] Y. Zhu, J. Yang, X. Xieg, Z. Wang, and X. Deng, "Long-distanceinfrared video pedestrian detection using deep learning and backgroundsubtraction," *Journal of Physics: Conference Series*, vol. 1682, no. 1, p. 012012, Nov. 2020, doi: 10.1088/1742-6596/1682/1/012012.
- [31] B. Kim, N. Yuvaraj, K. R. S. Preethaa, R. Santhosh, and A. Sabari, "Retracted article: enhanced pedestrian detection using optimized deep convolution neural network for smart building surveillance," *Soft Computing*, vol. 24, no. 22, pp. 17081–17092, Nov. 2020, doi: 10.1007/s00500-020-04999-1.
- [32] D. Tomè, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi, and S. Tubaro, "Deep convolutional neural networks for pedestrian detection," *Signal Processing: Image Communication*, vol. 47, pp. 482–489, Sep. 2016, doi: 10.1016/j.image.2016.05.007.
- [33] A. Dradrach, J. Konert, and J. Ruminski, "Multimodal camera for pedestrian detection with deep learning models," in *2023 IEEE International Conference on Industrial Technology (ICIT)*, Apr. 2023, vol. 2023-April, pp. 1–6, doi: 10.1109/ICIT58465.2023.10143046.
- [34] L. Chen *et al.*, "Survey of pedestrian action recognition techniques for autonomous driving," *Tsinghua Science and Technology*, vol. 25, no. 4, pp. 458–470, Aug. 2020, doi: 10.26599/TST.2019.9010018.
- [35] C.-W. Zhang, M.-Y. Yang, H.-J. Zeng, and J.-P. Wen, "Pedestrian detection based on improved LeNet-5 convolutional neural network," *Journal of Algorithms & Computational Technology*, vol. 13, Jan. 2019, doi: 10.1177/1748302619873601.
- [36] Y. F. Said and M. Barr, "Pedestrian detection for advanced driver assistance systems using deep learning algorithms," *IJCSNS International Journal of Computer Science and Network Security*, vol. 19, no. 9, p. 9, 2019, [Online]. Available: <https://www.researchgate.net/publication/337050186>.
- [37] S. Ahmed, M. N. Huda, S. Rajbhandari, C. Saha, M. Elshaw, and S. Kanarachos, "Pedestrian and cyclist detection and intent estimation for autonomous vehicles: a survey," *Applied Sciences*, vol. 9, no. 11, p. 2335, Jun. 2019, doi: 10.3390/app9112335.
- [38] C. Amisse, M. E. Jijón-Palma, and J. A. S. Centeno, "Fine-tuning deep learning models for pedestrian detection," *Boletim de Ciências Geodésicas*, vol. 27, no. 2, 2021, doi: 10.1590/s1982-21702021000200013.
- [39] H. Yu *et al.*, "The unmanned aerial vehicle benchmark: object detection, tracking and baseline," *International Journal of Computer Vision*, vol. 128, no. 5, pp. 1141–1159, May 2020, doi: 10.1007/s11263-019-01266-1.
- [40] Q. Liu, H. Ye, S. Wang, and Z. Xu, "YOLOv8-CB: dense pedestrian detection algorithm based on in-vehicle camera," *Electronics*, vol. 13, no. 1, p. 236, Jan. 2024, doi: 10.3390/electronics13010236.
- [41] G. Y. Öztel and İ. Öztel, "Deep learning-based road segmentation & pedestrian detection system for intelligent vehicles," *Sakarya University Journal of Computer and Information Sciences*, vol. 6, no. 1, pp. 22–31, Apr. 2023, doi: 10.35377/saucis...1170902.
- [42] M. Sukkar, M. Shukla, D. Kumar, V. C. Gerogiannis, A. Kanavos, and B. Acharya, "Enhancing pedestrian tracking in autonomous vehicles by using advanced deep learning techniques," *Information*, vol. 15, no. 2, p. 104, Feb. 2024, doi: 10.3390/info15020104.
- [43] H. Lee, H. Cho, B. Noh, and H. Yeo, "NAVIBox: real-time vehicle-pedestrian risk prediction system in an edge vision environment," *Electronics*, vol. 12, no. 20, p. 4311, Oct. 2023, doi: 10.3390/electronics12204311.
- [44] R. K. Dinakaran *et al.*, "Deep learning based pedestrian detection at distance in smart cities," in *Advances in Intelligent Systems and Computing*, vol. 1038, 2020, pp. 588–593.
- [45] M. Braun, S. Krebs, F. Flohr, and D. M. Gavrilu, "EuroCity persons: a novel benchmark for person detection in traffic scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 8, pp. 1844–1861, 2019, doi: 10.1109/TPAMI.2019.2897684.
- [46] "TensorFlow Lite," *Google*. <https://www.tensorflow.org/lite/models/convert/> (accessed Jan. 10, 2025).
- [47] T. Authors, "TensorFlow Lite for Microcontrollers," *TensorFlow*, 2021. <https://www.tensorflow.org/lite/microcontrollers> (accessed Jan. 10, 2025).
- [48] F. Wang, W. Gong, and J. Liu, "On spatial diversity in WiFi-based human activity recognition: a deep learning-based approach," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2035–2047, Apr. 2019, doi: 10.1109/JIOT.2018.2871445.
- [49] Matplotlib, "Visualization with Phytion", [Online]. Available: <https://matplotlib.org/> (accessed Jan. 10, 2025).
- [50] R. A. Khalil, N. Saeed, M. Masood, Y. M. Fard, M.-S. Alouini, and T. Y. Al-Naffouri, "Deep learning in the industrial internet of things: potentials, challenges, and emerging applications," *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11016–11040, Jul. 2021, doi: 10.1109/JIOT.2021.3051414.
- [51] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge-computing-enabled smart cities: a comprehensive survey," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10200–10232, Oct. 2020, doi: 10.1109/JIOT.2020.2987070.
- [52] T. J. Saleem and M. A. Chishti, "Deep learning for the internet of things: potential benefits and use-cases," *Digital Communications and Networks*, vol. 7, no. 4, pp. 526–542, Nov. 2021, doi: 10.1016/j.dcan.2020.12.002.
- [53] Q. Chen *et al.*, "A survey on an emerging area: deep learning for smart city data," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 3, no. 5, pp. 392–410, Oct. 2019, doi: 10.1109/TETCI.2019.2907718.
- [54] U. Alvi, M. A. K. Khattak, B. Shabir, A. W. Malik, and S. R. Muhammad, "A comprehensive study on IoT based accident detection systems for smart vehicles," *IEEE Access*, vol. 8, pp. 122480–122497, 2020, doi: 10.1109/ACCESS.2020.3006887.
- [55] A. Ess, B. Leibe, and L. Van Gool, "Depth and appearance for mobile scene analysis," in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8, doi: 10.1109/ICCV.2007.4409092.





- [56] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: an evaluation of the state of the art," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 743–761, Apr. 2012, doi: 10.1109/TPAMI.2011.155.
- [57] S. Zhang, R. Benenson, and B. Schiele, "CityPersons: a diverse dataset for pedestrian detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, vol. 2017-Janua, pp. 4457–4465, doi: 10.1109/CVPR.2017.474.
- [58] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005, vol. 1, pp. 886–893, doi: 10.1109/CVPR.2005.177.
- [59] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2012, pp. 3354–3361, doi: 10.1109/CVPR.2012.6248074.
- [60] S. Gilroy, M. Glavin, E. Jones, and D. Mullins, "Pedestrian occlusion level classification using keypoint detection and 2D Body surface area estimation," in *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, Oct. 2021, vol. 2021-October, pp. 3826–3832, doi: 10.1109/ICCVW54120.2021.00427.
- [61] M. Braun, S. Krebs, F. Flohr, and D. M. Gavrila, "EuroCity persons: a novel benchmark for person detection in traffic scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 8, pp. 1844–1861, Aug. 2019, doi: 10.1109/TPAMI.2019.2897684.
- [62] M. Ramzan, H. U. Khan, S. M. Awan, A. Ismail, M. Ilyas, and A. Mahmood, "A survey on state-of-the-art drowsiness detection techniques," *IEEE Access*, vol. 7, pp. 61904–61919, 2019, doi: 10.1109/ACCESS.2019.2914373.
- [63] S. Ding, Z. Yuan, P. An, G. Xue, W. Sun, and J. Zhao, "Cascaded convolutional neural network with attention mechanism for mobile EEG-based driver drowsiness detection system," in *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Nov. 2019, pp. 1457–1464, doi: 10.1109/BIBM47256.2019.8982938.
- [64] M. K. Hussein, T. M. Salman, A. H. Miry, and M. A. Subhi, "Driver drowsiness detection techniques: a survey," in *2021 1st Babylon International Conference on Information Technology and Science (BICITS)*, Apr. 2021, pp. 45–51, doi: 10.1109/BICITS51482.2021.9509912.
- [65] A.-C. Phan, N.-H.-Q. Nguyen, T.-N. Trieu, and T.-C. Phan, "An efficient approach for detecting driver drowsiness based on deep learning," *Applied Sciences*, vol. 11, no. 18, p. 8441, Sep. 2021, doi: 10.3390/app11188441.
- [66] L. Xu, S. Li, K. Bian, T. Zhao, and W. Yan, "Sober-Drive: A smartphone-assisted drowsy driving detection system," in *2014 International Conference on Computing, Networking and Communications (ICNC)*, Feb. 2014, pp. 398–402, doi: 10.1109/ICCNC.2014.6785367.
- [67] S. Park, F. Pan, S. Kang, and C. D. Yoo, "Driver drowsiness detection system based on feature representation learning using various deep networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10118 LNCS, 2017, pp. 154–164.

BIOGRAPHIES OF AUTHORS







Norah N. Alajlan     received the B.S. and M.S. degrees from the Information Technology Department. She has published more than six articles in various refereed international journals. Her research interests include tiny machine learning (TinyML), deep learning, and IoT. She has been serving as a reviewer for Internet of Things Journal, Alexandria Engineering Journal, Computers Electrical Engineering, and IEEE Access Journal such as Wireless Networks (WINE), and the Journal of Mobile. She can be contacted at email: 411200285@qu.edu.sa.



Abeer I. Alhujaylan     has been an assistant professor at the Information Technology Department, College of Computer, Qassim University, Saudi Arabia since 2020. She was born in Saudi Arabia, and her B.Sc. obtained from the Qassim University, College of Computer, Computer Science Department in 2008. Her M.Sc., and Ph.D. degrees were obtained from the King Saud University, Computer Collage, Computer Science Department in 2014 and 2020, respectively. Her research interests include optimization using metaheuristics, deep learning, and machine learning. She can be contacted at email: a.alhujaylan@qu.edu.sa.



Dina M. Ibrahim     has been an associate professor at the Information Technology Department, College of Computer, Qassim University, Saudi Arabia, since 2015. In addition, Dina works as an assistant professor at the Computers and Control Engineering Department, Faculty of Engineering, Tanta University, Egypt. She was born in the United Arab Emirates, and her B.Sc., M.Sc., and Ph.D. degrees were obtained from the Computers and Control Engineering Department, Faculty of Engineering, Tanta University, in 2002, 2008, and 2014, respectively. Dina worked as a consultant engineer, database administrator, and vice manager on the Management Information Systems (MIS) Project, at Tanta University, Egypt, from 2008 until 2014. Her research interests include networking, wireless communications, machine learning, security, and the IoT. She has published more than 70 articles in various refereed international journals and conferences. She has been serving as a reviewer in the Wireless Network (WINE) Journal since 2015. She has also served as a co-chair of the International Technical Committee for the Middle East Region of the ICCMIT conference since 2020. She can be contacted at email: d.hussein@qu.edu.sa or dina.mahmoud@f-eng.tanta.edu.eg.