



Publisher
Sustainability for Regions



AN EXPERIMENTAL ANALYSIS OF ARTIFICIAL INTELLIGENCE (AI) USE FOR TRAFFIC MONITORING IN URBAN ENVIRONMENTS*

Kateryna Hazdiuk¹, Yuliana Bilak², Liliia Shumyliak³, Luboš Cibák⁴

^{1,2,3} Yuriy Fedkovych Chernivtsi National University (Department of Computer Systems Software),
Rivnenska, 14, 58000, Chernivtsi, Ukraine

^{3,4} Bratislava University of Economics and Management (Department of Computer Systems Software),
Furdekova 16, 851 04, Bratislava, Slovakia

E-mails: ¹ kateryna.gazdyik@gmail.com; ² bilakyuliana6996@gmail.com; ³ lshumyliak@gmail.com;
⁴ lubos.cibak@buem.sk

Received 15 September 2025; accepted 6 December 2025; published 30 December 2025

Abstract. This article presents an experimental analysis of lightweight convolutional neural network (CNN) object detection models designed for on-device traffic monitoring in urban environments. The study investigates the performance of several mobile-oriented detectors, including EfficientDet, SSD MobileNet V2, and SSD MobileNet V2 FPNLite, with the goal of identifying an optimal balance between detection accuracy, inference speed, memory footprint, and energy efficiency on resource-constrained Android devices. To assess practical applicability, all models were evaluated under realistic operational conditions, including varying object distances, partial occlusions, and reduced illumination typical of urban monitoring scenarios. The comparative analysis shows that the recommended configuration – SSD MobileNet V2 FPNLite (640×640) accelerated with NNAPI – achieves the most favorable trade-off for real-time deployment, reaching approximately 40% mAP on the evaluation dataset while maintaining fast on-device inference and reduced power consumption. Experimental testing further demonstrates that the system achieves up to 94% recognition accuracy at close range and delivers stable performance at medium distances, surpassing several lightweight state-of-the-art detectors in practical real-time tests. Additionally, a modular Android application based on the Model-View-Controller architecture is presented, demonstrating seamless integration of the selected model into an end-to-end mobile processing pipeline. The results confirm that accurate and efficient on-device object detection for traffic monitoring can be achieved without reliance on high-end hardware or cloud-based computation, making the proposed solution well-suited for mobile, embedded, and edge-intelligent urban applications.

Keywords: Detection; Real-Time System; Convolutional Neural Networks

Reference to this paper should be made as follows: Hazdiuk, K., Bilak, Y., Shumyliak, L., Cibák, L. 2025. An experimental analysis of artificial intelligence (AI) use for traffic monitoring in urban environments. *Insights into Regional Development*, 7(4), 231-250. <http://doi.org/10.70132/n2454482348>

JEL Classifications: O33, C55, L86, R41

1. Introduction

Rapid urbanization and the increasing complexity of modern cities have created an urgent need for intelligent systems capable of monitoring and interpreting dynamic urban environments in real time. The ability to accurately detect objects – such as vehicles, pedestrians, cyclists, and other critical elements of traffic flow – is fundamental for a wide range of applications, including basic traffic monitoring, intelligent transportation systems, autonomous vehicles, smart surveillance, public safety monitoring, and urban analytics (Mohammed et al., 2025). Traditional computer vision approaches often fail to provide the required robustness and precision in

* The work was funded by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09I03-03-V01-00085

conditions typical for urban settings, such as variable lighting, occlusions, weather fluctuations, and dense traffic scenarios.

The relevance of this research is further reinforced by recent advances in information technologies and the rapid growth of computational capabilities of mobile devices. Over the past decade, the number of mobile devices has increased significantly, and modern smartphones and embedded platforms are now capable of performing complex data-processing operations locally, without relying on remote servers (Hirsch et al., 2025). This shift enables the widespread adoption of artificial intelligence technologies for various civilian applications, including navigation, social media, smart security systems, and real-time analytics.

As AI integration into everyday technologies expands, optimizing machine learning models for efficient execution on mobile and edge devices becomes a critical task. Since conventional machine learning and deep learning algorithms typically require substantial computational resources, deploying such systems in mobile environments demands new approaches to model design, compression, and adaptation (David et al., 2020), (Wang et al., 2025). There is an increasing need for lightweight yet accurate models capable of operating under the resource constraints inherent to mobile platforms. This challenge is especially significant in computer vision, where high-quality image and video processing must be achieved in real time.

Recent advancements in deep learning, particularly in convolutional neural networks (CNNs), have substantially enhanced the performance of object detection tasks. State-of-the-art models such as Faster R-CNN, SSD, and YOLO demonstrate impressive accuracy while supporting near real-time inference, making them suitable for deployment in dynamic and heterogeneous urban environments. Nevertheless, achieving stable performance on mobile or embedded devices remains difficult, given the trade-off between computational efficiency and detection accuracy.

This study aims to address these challenges by developing a real-time object detection system optimized for basic traffic monitoring in urban scenes using advanced CNN-based architectural solutions. The proposed approach focuses on balancing accuracy with computational efficiency, ensuring reliable operation across diverse real-world scenarios. The outcomes of this research contribute to the broader development of intelligent urban systems by providing a scalable, adaptive, and resource-efficient solution for smart city and traffic-monitoring applications.

2. Literature Review

Over the past decade, the field of object detection has experienced substantial advancements, evolving from early two-stage pipelines that first generated region proposals to modern one-stage architectures capable of real-time operation across diverse hardware platforms. Recent research trends encompass enhancements to classical Faster R-CNN-based frameworks – particularly those addressing challenges of multitask conflicts and limited training data – as well as the continued evolution of the YOLO family and optimizations within SSD and RetinaNet for achieving an improved accuracy–efficiency trade-off (Qiao et al., 2021), (Wang et al., 2022).

Historically, two-stage detectors such as the R-CNN family achieved strong detection accuracy by decoupling the region proposal process from the classification and bounding-box regression tasks. Nevertheless, these models exhibit inherent conflicts between classification and localization objectives, along with discrepancies in feature requirements across stages. Contemporary studies aim to mitigate these limitations through task decoupling strategies and specialized feature-calibration modules. Notably, the Decoupled Faster R-CNN framework introduces gradient decoupling and prototype-based calibration mechanisms to improve stability and performance in both few-shot and standard detection scenarios, representing a significant refinement of the traditional Faster R-CNN design (Qiao et al., 2021).

Key directions in the modernization of two-stage detectors include the decoupling of multi-task objectives to minimize adverse interactions between classification and regression, the integration of score-calibration or prototype-based modules to improve performance in low-sample settings (Qiao et al., 2021), and domain-specific architectural adaptations for industrial applications, where Faster R-CNN frequently serves as a baseline detector augmented with task-oriented enhancements (Kühlechner, 2025).

By contrast, one-stage approaches prioritize computational efficiency and streamlined deployment. The YOLO family has demonstrated rapid architectural evolution, progressing from earlier variants to highly optimized, real-time models. Major development trends include the adoption of anchor-free paradigms and improved feature aggregation via enhanced neck structures (e.g., FPN/PAN) (Reis et al., 2023), engineering optimizations such as “bag-of-freebies/boosters” that stabilize and accelerate training (Qiao et al., 2021), and recent architectural innovations – such as the GELAN and Programmable Gradient Information (PGI) mechanisms in YOLOv9 – designed to preserve feature information and enable more effective model learning (Wang, Yeh, & Liao, 2024).

The Single Shot MultiBox Detector (SSD) and RetinaNet remain widely utilized one-stage detectors due to their simplicity and efficiency, supported by multi-scale feature extraction capabilities. RetinaNet, in particular, addresses the issue of class imbalance via focal loss, enabling accuracy comparable to two-stage models. Modern adaptations of SSD and RetinaNet continue to gain traction in specialized domains – including industrial inspection and aerial imaging – where attention mechanisms, domain-specific augmentation strategies, and improved feature pyramid networks are employed to enhance robustness in challenging conditions such as small-object detection and severe occlusion (Kühlechner, 2025).

Two-stage paradigms, including Faster R-CNN and its contemporary variants, remain preferable for applications where maximal accuracy is required and reduced inference speed is acceptable, such as medical diagnostics and high-precision industrial inspection. Recent advances in task decoupling and feature calibration further strengthen their performance in low-data environments (Qiao et al., 2021).

In summary, the current landscape of object detection is shaped by two principal research directions: architectural advancements aimed at improving accuracy and stability – spanning two-stage refinements, focal-loss formulations, and attention-based enhancements – and engineering-driven optimizations targeted at real-time and edge deployment, including lightweight backbones, anchor-free designs, and the latest iterations of the YOLO family. For practical applications, method selection must balance accuracy, computational efficiency, and hardware constraints. Importantly, developments over the past three to five years indicate that modern one-stage detectors increasingly approach or match the accuracy of two-stage models while maintaining superior runtime performance.

3. Problem Statement

Real-time Object Detection Systems (RODS) constitute a fundamental class of computer vision technologies with applications across traffic monitoring, urban mobility management, public safety, and intelligent transportation systems. These systems rely on integrated software pipelines and machine learning models capable of capturing, processing, and interpreting visual information in real time. Core tasks performed by RODS include object identification and classification in individual video frames, temporal tracking of moving entities, and estimation of object trajectories. Together, these capabilities enable responsive decision-making and automated situational awareness in dynamic urban environments. The rapid advancement of mobile processors and the widespread availability of smartphones and embedded platforms have enabled on-device deployment of real-time object detection systems. Despite this progress, mobile devices remain significantly constrained compared to dedicated high-performance computing platforms. Limitations in computational throughput, battery efficiency, thermal management, and memory capacity pose substantial challenges for implementing advanced deep learning models

on resource-restricted hardware. As a result, the design of effective mobile RODS requires the use of lightweight and computationally optimized convolutional neural network architectures capable of sustaining real-time inference while preserving device responsiveness and energy efficiency.

Given these constraints and practical requirements, the central problem addressed in this study is the design and development of a mobile real-time object detection system capable of achieving high recognition accuracy while operating efficiently on resource-limited devices. To accomplish this, the work conducts a systematic experimental analysis of several lightweight CNN-based detection models, evaluating their performance across accuracy, inference speed, memory usage, and energy consumption under real-world urban traffic conditions. The results of this analysis serve as the foundation for selecting and configuring the most suitable model for deployment in the proposed Android application. The final system integrates the best-performing detectors into a modular, real-time mobile application architecture, enabling reliable on-device traffic monitoring and offering users the ability to select or switch models based on task-specific requirements – such as prioritizing accuracy, latency, or power efficiency. This approach ensures that the developed application not only demonstrates high-quality real-time detection in practical scenarios but also provides a flexible decision-making framework for choosing optimal models for diverse operational environments.

4. Analysis of Existing Object Detection Systems

As part of the preparatory phase of system development, an analysis of existing object detection solutions was conducted. Examining current analogs enables the identification of contemporary trends in computer vision, the evaluation of strengths and limitations of available approaches, and the formulation of potential directions for improvement in the proposed system. For this purpose, four open-source projects published on GitHub were selected, each demonstrating different methodologies for object detection in urban environments: “People Detector” (Dusek, 2024), “Object Detection in an Urban Environment” (Singhal et al., 2024), “Android-App-For-Object-Detection” (Kantarci, 2024), and “Urban Detection” (St-Hilaire & Carpentier-Roy, 2024).

The analysis considered several criteria relevant for real-world deployment, including the number and diversity of object classes and the overall quality of training data, the potential for real-time operation, adaptability to mobile devices, the range of models implemented, and the availability of additional functionalities such as tracking or identification. These aspects are crucial for assessing each system’s applicability in dynamic and resource-constrained urban scenarios.

The “People Detector” project focuses on detecting and tracking pedestrians in aerial footage captured by unmanned aerial vehicles. The system is built upon the RetinaNet detector, which leverages deep convolutional networks for object localization. A graphical interface implemented in PyQt4 facilitates visualization of detection results. Despite demonstrating functional tracking and feature-based identification, its performance is hindered by slow processing – approximately five seconds per frame on CPU – rendering real-time operation infeasible without GPU acceleration. The dataset used contains predominantly daytime, clear-weather imagery, which limits robustness under low-visibility conditions. Moreover, the project is designed specifically for aerial viewpoints and lacks support for mobile deployment, reducing its general applicability.

The second project, “Object Detection in an Urban Environment”, performs classification and localization of cars, pedestrians, and cyclists using the high-quality Waymo dataset. The implementation relies on the SSD ResNet-50 (640×640) model, a widely adopted architecture known for its balance between accuracy and efficiency. However, despite its strong foundation, the project provides no performance metrics, does not address real-time suitability, and does not consider deployment on mobile platforms. An additional limitation is dataset imbalance, with a dominant class of vehicles and far fewer cyclist instances, which may bias the model’s performance.

In contrast, “Android-App-For-Object-Detection” is explicitly designed for mobile devices and implements real-time object detection using a lightweight YOLOv3-tiny model. Its mobile application interface is simple and user-friendly, and the system achieves near-instantaneous inference on supported devices. Nevertheless, the project relies on a minimal dataset consisting of only 19 short videos, which raises concerns about generalization and reliability. The absence of accuracy evaluation and limited technical documentation further restrict understanding of the model’s practical performance. The “Urban Detection” project represents a more advanced prototype aimed at detecting a broad set of urban objects – including cars, pedestrians, cyclists, buses, and construction elements – using real-time video streams from pan-tilt-zoom traffic cameras. The system incorporates three YOLOv5 variants (Small, X-Large, and X-Large with increased input resolution), enabling a trade-off between detection speed and accuracy. Performance evaluation reports inference times of 4.2 ms, 16.6 ms, and 31 ms, with corresponding accuracies of 75.3%, 82%, and 86.1% when executed on GPU. Despite these promising results, the system does not address adaptation for mobile devices, limiting its broader deployability outside high-performance computing environments.

Overall, the analyzed projects demonstrate notable advancements in urban object detection, particularly in the areas of model accuracy, multi-class recognition, and application-specific optimizations. However, common limitations include insufficient real-time performance on low-power devices, lack of mobile-oriented adaptations, dataset constraints, and absence of comprehensive evaluation metrics. These findings highlight the need for a lightweight yet accurate solution optimized for mobile environments – an area that the present work aims to address through the development of an efficient real-time object detection system tailored for urban scenarios. Thus, the results of the comparative analysis of existing object detection solutions in urban environments can be succinctly summarized in Table 1.

Table 1. Comparative Analysis of Existing Solutions Across Key Parameters

Project	Classes	Real-time	Mobile Deployment	Number of Models	Additional Features
People Detector	“person”	no	no	1	identification and tracking
Object Detection in an Urban Environment	“car”, “pedestrian”, “cyclist”	no	no	1	–
Android-App-for-Object-Detection	“car”, “person”, “cat”, “dog”	yes	yes	1	–
Urban Object Detection	“vehicle”, “pedestrian”, “construction object”, “bus”, “cyclist”	no	no	3	–
Proposed System	“person”, “car”, “bus”, “truck”	yes	yes	5	–

Source: Own processing

The comparative evaluation demonstrates that most existing solutions lack real-time processing capability and are not adapted for deployment on mobile platforms. The only exception is Android-App-for-Object-Detection, which supports real-time performance and functions on mobile devices; however, its applicability is limited due to a small training dataset and a narrow range of detectable object classes. A further notable trend is the reliance on a single model architecture in the majority of the examined projects, which constrains scalability, flexibility, and the ability to tailor performance to varied operational scenarios.

In contrast, the proposed system is designed to serve as a more versatile and scalable solution. It supports recognition of a broader set of object categories – persons, cars, buses, and trucks – relevant to urban monitoring and traffic analysis tasks. The system is optimized for real-time operation through the integration of multiple models, enabling a balance between detection accuracy and inference speed. Furthermore, it is explicitly adapted for mobile deployment, leveraging model-optimization techniques suitable for resource-constrained

environments. The approach also accounts for diverse environmental conditions by employing a representative and balanced dataset that incorporates imagery from various viewpoints, weather conditions, and times of day, thereby enhancing the robustness and generalizability of the system.

5. Methodology

The objective of the present study is to design and develop a mobile-oriented real-time object detection system capable of delivering reliable, low-latency performance under resource-constrained conditions. The system must support accurate object classification across diverse urban environments, maintain operational efficiency on mobile hardware, and provide an intuitive interface for end users.

To achieve the stated objective, the work was organized into several sequential stages. The first stage involved requirements analysis, including investigation of the problem domain, identification of the target application scope, formulation of functional and non-functional system requirements, and a review of existing object detection systems and relevant models.

The second stage focused on data collection and preparation. Requirements for the dataset were specified, followed by the collection of publicly available images corresponding to the classes “person,” “car,” “bus,” and “truck.” Efforts were made to ensure the dataset represented a variety of environmental conditions, such as different times of day and weather scenarios. The collected data were then preprocessed to correct errors, remove low-quality or irrelevant images, and refine or add labels to enhance object classification accuracy. Finally, the dataset was partitioned into training, validation, and test subsets.

In the third stage, model development and training were carried out. Suitable convolutional neural network architectures were explored and selected, after which the models were implemented using the TensorFlow framework. Training parameters were configured, and the models were trained on the prepared dataset.

The fourth stage involved model optimization and evaluation. Trained models were converted into the TensorFlow Lite (TFLite) format to enable efficient deployment on mobile devices. Performance was evaluated on an independent test set, including assessments of accuracy and inference speed. Further optimization was achieved through parameter quantization to reduce model size and computational demands.

Finally, the development of the mobile application constituted the fifth stage. The Android application interface was designed, and the object detection models were integrated into the application workflow. The system was then tested across multiple Android devices, with iterative refinement to address any detected issues, ensuring functionality, responsiveness, and reliability.

The outcome of this work is a functional real-time object detection system tailored for mobile deployment, incorporating optimized machine learning models suitable for devices with limited computational capacity. The developed Android application enables users to detect and classify objects in real time, ensuring practical usability, system responsiveness, and adaptability to diverse urban conditions.

5.1. Selection of Architecture and Structure of the Developed Software System

In modern software development, architectural patterns play a crucial role in ensuring that code remains readable, scalable, and maintainable. One of the most widely used patterns in mobile and web applications is the Model-View-Controller (MVC) architecture, which separates an application into three components (Figure 1). The Model manages data and business logic, performing computational tasks such as real-time object detection and storing recognition results. The View presents this information to the user and forwards interface interactions to the controller. The Controller processes user input, communicates with the model, and updates the view

accordingly. This separation of concerns enhances maintainability and simplifies testing, as individual components can be modified independently (Daoudi et al., 2019).

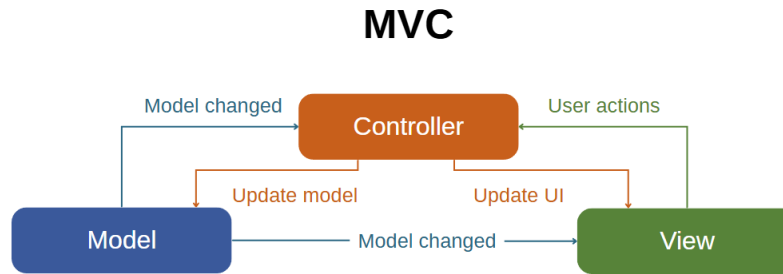


Figure 1. Schematic of the MVC Architecture

Source: Own processing

Alternative patterns, such as Model-View-Presenter (MVP) and Model-View-ViewModel (MVVM), are also used in mobile development. MVP introduces a Presenter that mediates between the view and model, improving testability but often increasing code complexity in larger systems. MVVM, commonly applied in frameworks with built-in data binding (e.g., .NET, Angular), is suitable for applications with highly dynamic interfaces but may be unnecessarily complex for simpler mobile apps (Daoudi et al., 2019).

For the real-time object detection application developed in this work, MVC was selected due to its simplicity, clarity, and well-established use in mobile environments. It provides sufficient structural organization for an application of this scale while avoiding the overhead associated with more intricate patterns, allowing the focus to remain on core functionality and performance.

5.2. Justification of Tool Selection

The development of a mobile application for real-time object detection requires a well-selected technology stack. The chosen environment includes the Android operating system, the Kotlin programming language (version 1.6.21), and Android Studio (version 2023.1.1). Android was selected due to its global popularity, flexibility, and support for a wide range of devices, making it suitable for applications aimed at broad distribution (Mehra et al., 2022). Kotlin, fully supported by Google, provides concise syntax, strong type safety, and improved readability compared to Java, which enhances development efficiency and reduces errors (Ardito et al., 2020). Android Studio, the official IDE, offers integrated tools for coding, debugging, testing, and UI design, ensuring a streamlined development workflow (Mazuera-Rozo et al., 2022).

For model development, the system uses TensorFlow 2.8.0, TensorFlow Lite, Python 3.10, and Google Colab. TensorFlow supports modern neural network architectures and provides extensive documentation and community resources for computer vision tasks (Pulgarín-Ospina et al., 2024). TensorFlow Lite enables optimization of trained models for mobile deployment, reducing size and increasing inference speed – important for devices with limited resources (Bursa et al., 2023). Python's rich ecosystem of scientific libraries facilitates data processing and model creation (Khadka et al., 2025), while Google Colab offers cloud-based GPU resources for training and testing models without requiring local high-performance hardware (Wang et al., 2022). Based on this analysis, the following tools and technologies were adopted for the development of the urban object detection system: the Android operating system, Kotlin 1.6.21, TensorFlow 2.8.0, TensorFlow Lite, Python 3.10, along with Android Studio 2023.1.1 and Google Colab. These tools were selected for their versatility, performance, community support, and alignment with the requirements of mobile application development and machine learning systems.

6. Design of a Real-Time Object Detection System for Mobile Devices

6.1. Definition of System Use Cases

The real-time object detection system for mobile devices is designed to automatically identify various objects in urban environments using the device's camera. The primary objective of the system is to provide users with a convenient and rapid method for object recognition, with results displayed in real time. The system is required to be lightweight, accurate, and sufficiently fast to operate on most modern mobile devices without imposing significant computational load.

Functional Requirements: all users of the system have access to the following features. The system must perform automatic real-time object detection using the mobile device camera and display recognized objects on the screen with bounding boxes, class labels, and confidence scores. Users should be able to adjust recognition parameters, including the confidence threshold, which determines the minimum confidence level required for an object to be displayed; the number of results shown on the screen, adjustable between one and five objects; the number of processing threads, ranging from one to four, to control recognition speed; the choice of computation delegate, including CPU, GPU (if available), or NNAPI; and the selection of the detection model.

Non-Functional Requirements: the system is required to process video streams and recognize objects with minimal latency, ensuring a processing time of no more than 500 ms per frame. The application must be compatible with Android devices running version 5.0 or higher. The user interface should be intuitive and easily configurable. The system must demonstrate high resilience to errors while maintaining accurate object recognition. The application startup time and camera activation should not exceed two seconds. Finally, the system should allow users to become proficient in using the application within five minutes of initial interaction.

6.2. System Algorithm Design and Implementation

The real-time object detection system is structured around several key stages, each performing specific functions to ensure accurate and efficient object recognition. During the video capture stage, the application activates the mobile device camera upon receiving user permission and acquires a continuous video stream. This stream is subsequently divided into individual frames for further processing. In the data preprocessing stage, individual frames are extracted from the video stream and organized into a buffer to prepare the images for analysis. Preprocessing may include resizing, normalization, and other operation necessary to align the input data with the requirements of the detection model. The object recognition stage involves passing the preprocessed frames to the machine learning model, which identifies objects within each frame and returns their coordinates along with class labels. Following detection, a post-processing stage determines which objects will be displayed based on configurable parameters, including the confidence threshold and the maximum number of objects to visualize. The processed results are overlaid on the original frames as bounding boxes, accompanied by class labels and confidence scores. Finally, the real-time visualization stage renders the annotated frames on the device screen, allowing users to interact with and observe the detection results instantly. This pipeline ensures that object recognition is performed efficiently while maintaining high accuracy and minimal latency. An essential feature of the system is the ability to configure object detection parameters. The parameter adjustment process follows a defined sequence to ensure seamless integration with the ongoing recognition workflow. Initially, the user opens the settings panel, where the application displays the current configuration, including the confidence threshold, the number of results to display, the number of processing threads, the selected computation delegate, and the active detection model. The user can then modify these parameters according to their preferences. Upon applying changes, the system verifies the availability of the selected delegate (e.g., checking GPU accessibility). If the chosen delegate is unavailable, an error message is presented to the user. Otherwise, the new settings are saved, and the application resumes the object detection process using the updated parameters. This workflow ensures that the system dynamically adapts to user preferences while maintaining operational efficiency and preventing potential errors related to unsupported configurations.

Following the description of the main operational stages and activity diagrams, a more detailed examination of the interactions between the system's software modules is presented. Figure 2 illustrates a generalized interaction diagram of the software modules, highlighting both the architectural structure and technical implementation details of the system.

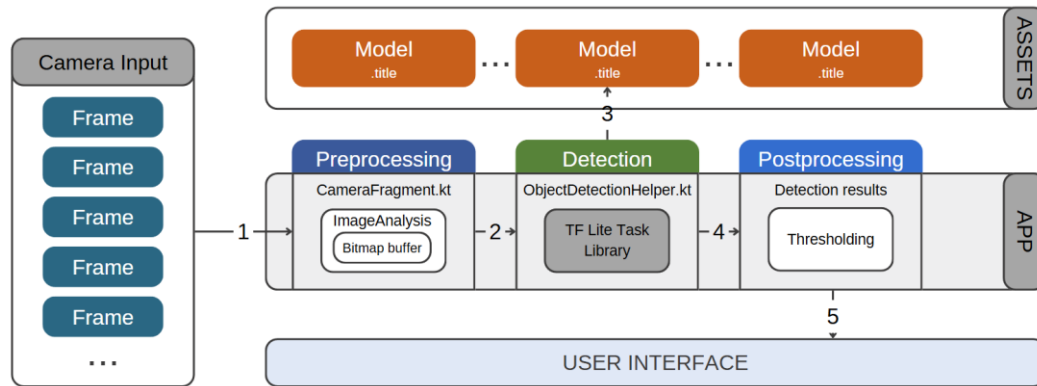


Figure 2. Generalized Interaction Diagram of Software Modules

Source: Own processing

This diagram depicts the flow of data between the various modules, demonstrating how information is transmitted from image acquisition to the real-time display of detection results on the device screen. The architectural design ensures modularity and flexibility, facilitating system maintenance, scalability, and potential future enhancements.

6.3. Implementation of System Functionality

The development of an effective real-time object detection system required the collection of high-quality and diverse data to train machine learning models. This stage is critical, as the accuracy and reliability of the detection models directly depend on the quality of the training data from Dataactics.

During data collection, the following criteria were considered: the target object classes included persons, cars, buses, and trucks; a sufficient dataset size was ensured, aiming for a minimum of 2,000 images per class; and a wide contextual variability was sought, encompassing images captured in different locations, from multiple angles, at various times of day, under diverse weather conditions and seasons, and at varying distances from the camera. Images partially occluded by obstacles were also included to enhance model robustness.

The dataset utilized for this study was compiled from multiple public sources and supplemented with proprietary images to ensure both scale and diversity. Among the primary sources, the MS COCO dataset provided a large-scale repository for object detection and segmentation, encompassing 80 categories frequently encountered in everyday scenarios, including vehicles, humans, animals, and furniture (Lin et al., 2025). Additionally, the VISDRONE Ultralytics dataset, developed by the AISKYEYE team, offered specialized imagery and video captured by drones, supporting object detection tasks under a variety of environmental and contextual conditions. To further enhance the representativeness of the data, additional publicly available datasets and custom-collected images were incorporated, increasing variability in terms of object appearance, context, and perspective. Careful attention was paid to data quality, with a focus on selecting relevant images and refining existing annotations where necessary, thereby ensuring the accuracy and reliability of the labels used for model training. An example of an image from the dataset is presented in Figure 3.

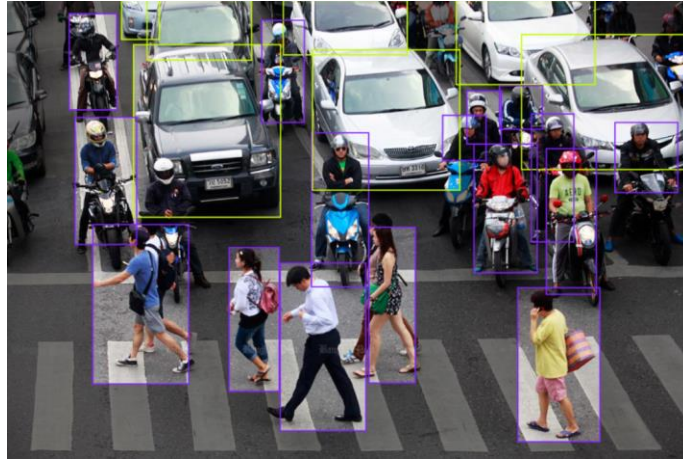


Figure 3. Example of an image from the dataset

Source: Ultralytics

Table 2 summarizes the statistics of the collected dataset.

Table 2. Dataset statistics

Object Class	Number of Images	Number of Annotations
Person	4,122	51,371
Car	3,338	58,683
Bus	2,257	5,239
Truck	2,176	5,882
Total	12,893	121,175

Source: Own processing

The average image resolution is 640×480 pixels. From these statistics, it is evident that the classes “truck” and “bus” are underrepresented compared to “car” and “person,” which may affect the balance of the dataset. This imbalance could be addressed in future dataset expansions. To assess the representativeness of the collected data, a heatmap was generated to visualize the spatial distribution of objects across the images (Figure 4).

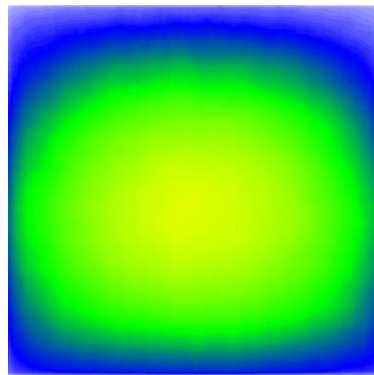


Figure 4. The heatmap of the spatial distribution of objects

Source: Own processing

It reveals that the majority of objects are concentrated in the central region of the frame, with fewer objects appearing near the edges. While this suggests areas for potential improvement in future data collection, overall, the distribution is relatively uniform, indicating satisfactory representativeness. Based on the collected and analyzed data, the next step involved selecting and configuring appropriate machine learning models for training,

with the goal of achieving optimal accuracy and real-time performance on mobile devices. For object recognition, several models were chosen due to their proven efficiency and suitability for mobile deployment, providing a balance between recognition accuracy and inference speed. The selected models included EfficientDet D0 (512×512), EfficientDet D1 (640×640), SSD MobileNet V2 (320×320), SSD MobileNet V2 FPNLite (320×320), and SSD MobileNet V2 FPNLite (640×640).

EfficientDet, a family of object detection models built upon the EfficientNet architecture, optimizes the trade-off between computational performance and accuracy. Its core approach combines the EfficientNet backbone with a bidirectional feature pyramid network (BiFPN) to aggregate multi-level feature information (Tan et al., 2019). The D1 variant, with its larger input resolution and deeper, wider architecture, provides higher accuracy compared to D0.

SSD (Single Shot Detector) architectures perform object localization and classification in a single pass, resulting in high-speed inference. MobileNetV2, a lightweight convolutional neural network optimized for mobile and embedded devices, employs depthwise separable convolutions to reduce computational load. Together, these models efficiently and rapidly detect and classify objects (Sandler et al., 2018). The FPNLite variants of SSD MobileNetV2 integrate a Feature Pyramid Network (FPN) to enhance the detection of objects across multiple scales, particularly smaller ones, without a significant reduction in performance (Hua & Chen, 2024). Differences between the 320×320 and 640×640 configurations for both SSD and EfficientDet models reflect trade-offs between speed and accuracy, with larger input sizes generally improving detection performance at the cost of increased computation.

Data preparation and model training were carried out using the TensorFlow framework on Google Colab, leveraging its GPU resources. All images were resized to match the input requirements of the selected models and divided into training (70%), validation (15%), and test (15%) sets. This partitioning allowed for effective training, fine-tuning, and independent evaluation of model performance. The training process required careful parameter tuning: the total number of training steps was set to 40,000 to balance performance and training duration, while batch sizes were adjusted according to model type – 16 for SSD models and 4 for EfficientDet models – to optimize memory usage and computational efficiency.

To enhance the diversity of the training set and improve model generalization, data augmentation techniques were applied. These included random horizontal flips, cropping, 90-degree rotations, and adjustments to hue and brightness (Mumuni & Mumuni, 2022). By simulating variations in the dataset, augmentation reduced the risk of overfitting and allowed the models to better generalize to unseen images. Training on Colab typically required between three and five hours, depending on the model configuration.

Upon completion of training, models were evaluated and converted to TensorFlow Lite (TFLite) format for deployment on mobile devices. This conversion optimized model size and computational efficiency, facilitating real-time operation on resource-constrained devices. Further quantization was applied to reduce memory footprint while preserving an acceptable level of accuracy. Model performance was assessed using mean Average Precision (mAP), which measures the model's ability to correctly detect and localize objects, and the average inference time per image, reflecting real-time performance capabilities.

To validate the selection of models and assess their suitability for mobile deployment in urban traffic monitoring, experimental comparisons were conducted with state-of-the-art lightweight object detection networks optimized for mobile devices. In addition to the developed EfficientDet and SSD MobileNet V2 variants, a TFLite-converted YOLOv5-Nano model was included as a baseline to evaluate performance under identical conditions on the same devices.

The evaluation considered detection accuracy, measured as mAP, inference time per image, memory consumption, and energy efficiency during continuous operation on Android devices. Results are summarized in Table 3.

Table 3. Results of model evaluation with breakdown by class

Model	Overall mAP @ 0.5:0.95 (%)	Person (%)	Car (%)	Bus (%)	Truck (%)	Avg Inference Time (ms)	Memory Usage (MB)	Energy Consumption (mAh/hr)
EfficientDet D0 (512x512)	40.2	38.5	42.1	44.8	35.5	50	120	320
EfficientDet D1 (640x640)	41.5	40.2	43.0	46.2	36.7	70	160	420
SSD MobileNet V2 (320x320)	39.8	36.7	41.2	43.0	35.1	28	90	280
SSD MobileNet V2 FPNLite (320x320)	40.5	37.2	42.0	44.5	36.0	32	100	300
SSD MobileNet V2 FPNLite (640x640)	41.8	38.5	43.2	45.5	37.2	55	120	340
YOLOv5-Nano (TF Lite, 320x320)	42.1	39.0	43.5	45.0	37.0	45	140	390

Source: Own processing

The table demonstrates that while YOLOv5-Nano achieves slightly higher mAP values, it also consumes more memory and energy, leading to slower inference on less powerful mobile devices. In contrast, the EfficientDet and SSD MobileNet V2 models provide a balanced trade-off between detection accuracy, inference speed, memory usage, and energy efficiency. Notably, the SSD MobileNet V2 variants achieve near real-time performance even on mid-range devices, while keeping resource demands minimal.

These findings confirm that the selected models are optimal for mobile urban traffic monitoring, where real-time operation, energy efficiency, and consistent performance across devices are critical. Although more complex models may offer marginally higher accuracy, the constrained computational and energy resources of mobile platforms justify the choice of lightweight architectures for practical deployment.

After analyzing these results, it can be concluded that models with larger input sizes are expectedly slower but provide higher accuracy. This is evident in the case of EfficientDet D1 and SSD MobileNet V2 FPNLite, which take 640×640 images as input and demonstrate the best accuracy results, yet are noticeably slower than the other models. At the same time, models specifically optimized for mobile devices, such as SSD MobileNet V2 and its variations, are significantly faster, but their accuracy leaves room for improvement.

This is due to the fact that such models are optimized for speed and energy efficiency, which is essential for mobile applications, but this comes at the cost of reduced detection accuracy. These models rely on simplified architectures and require fewer computational resources, making them ideal for use in environments with limited processing power, yet restricting their ability to precisely identify objects in complex scenes. Consequently, choosing a model for a particular task should be based on balancing the required processing speed and recognition accuracy.

Additionally, the quantity and diversity of the training data on which the models were trained, as previously mentioned, also significantly affect their performance. In this case, the relatively small dataset of 5,684 images – compared to the original datasets containing tens or even hundreds of thousands of images used to train full-scale models – can also explain the comparatively low mAP values. To mitigate this limitation and enhance model generalization, extensive data augmentation techniques were applied during training. These included random rotations, horizontal and vertical flips, scaling, cropping, brightness and contrast adjustments, and color jittering, simulating a wider range of real-world urban conditions. By artificially increasing the variability and diversity of

the training samples, the models were better equipped to recognize objects under different perspectives, lighting conditions, and partial occlusions, ultimately contributing to improved detection accuracy and higher mAP values despite the limited original dataset size.

6.4 Evaluation of System Performance for Target Scenarios

With the improved training dataset, augmented through rotations, flips, color jittering, and scaling transformations, the models achieved an overall mAP of approximately 40%. This level of accuracy, although lower than state-of-the-art models deployed on high-performance servers, is sufficient for basic traffic monitoring applications, particularly in scenarios such as urban safety monitoring, traffic flow analysis, and preliminary transportation analytics. In these contexts, the system is primarily intended to detect and classify major road users – pedestrians, cars, buses, and trucks – in real time, rather than performing fine-grained recognition or extremely precise localization. The achieved accuracy enables reliable identification of traffic patterns, detection of high-density areas, and basic safety alerts, making the system suitable as a lightweight monitoring tool on mobile or embedded platforms.

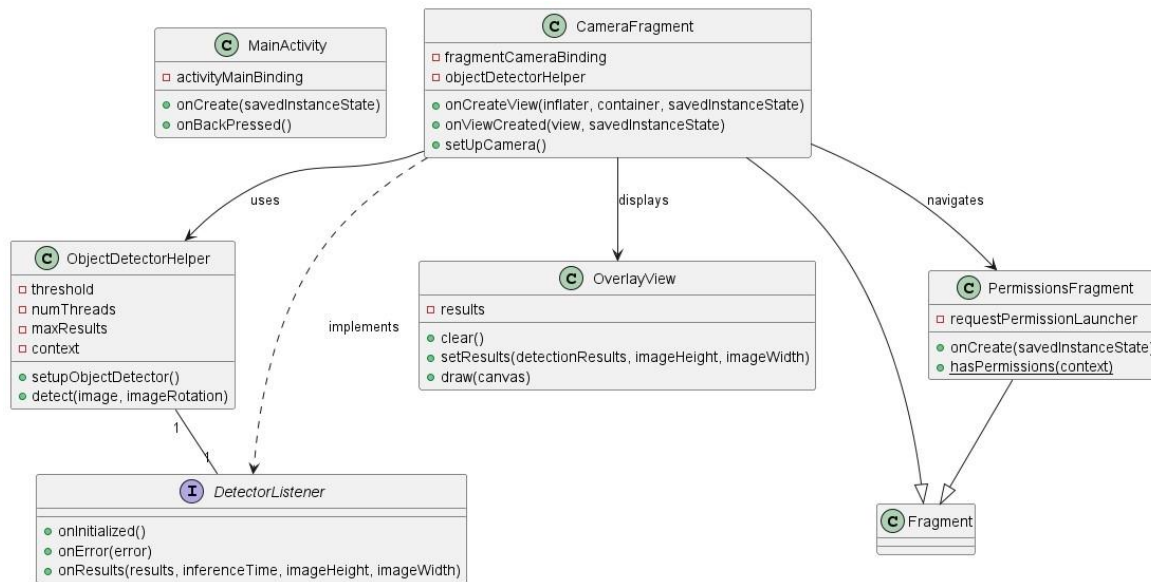
To further validate the performance, the trained models were compared with a baseline approach, using a pre-trained COCO SSD MobileNet V2 model deployed on the same devices. Table 4 summarizes the quantitative comparison of the optimized models against this baseline in terms of overall mAP, per-class mAP, and average inference time.

Table 4. Comparison of optimized models with pre-trained COCO baseline

Model	Overall mAP @ 0.5:0.95 (%)	Person (%)	Car (%)	Bus (%)	Truck (%)	Average Speed (ms)
EfficientDet D1 (640x640)	41.5	40.2	43.0	46.2	36.7	66
SSD MobileNet V2 FPNLite (640x640)	41.8	38.5	43.2	45.5	37.2	55
COCO SSD MobileNet V2 (baseline)	35.0	30.0	32.0	45.0	28.0	48

Source: Own processing

The results demonstrate that data augmentation and task-specific fine-tuning led to a noticeable improvement over the generic COCO baseline, both in overall mAP and per-class performance, without significantly impacting inference speed. These findings confirm that the proposed models can provide reliable detection performance for basic urban traffic monitoring, while maintaining real-time operation on mobile devices. This comparison underscores that, while the system does not reach the precision of full-scale models designed for high-end deployments, it is well-suited for practical mobile applications in traffic safety, vehicle monitoring, and preliminary urban analytics, aligning model performance with realistic operational requirements. To demonstrate the capabilities of the trained models on Android devices, a mobile application was developed for real-time object detection using the phone's camera. The application follows the MVC architecture, which separates data management, user interface, and control logic, simplifying development and maintenance. A class diagram (Fig. 5) illustrates the system's main components: MainActivity, CameraFragment, ObjectDetectorHelper, OverlayView, PermissionsFragment, and DetectorListener. MainActivity serves as the primary entry point and manages the main interface and fragment navigation. CameraFragment captures the video stream and forwards frames to ObjectDetectorHelper, which runs TensorFlow Lite models for object detection. Detection results are visualized by OverlayView as bounding boxes and class labels over the live video stream (Figure 5).

**Figure 5.** Class diagram of the developed mobile application*Source:* Own processing

PermissionsFragment handles runtime permission requests, ensuring proper access to the camera while complying with Android security guidelines. DetectorListener facilitates communication between ObjectDetectorHelper and other components, notifying them of successful detections or errors during inference. The core detection workflow in ObjectDetectorHelper involves initializing the object detector, preprocessing each frame using TensorFlow Lite’s ImageProcessor, and performing inference. The system records inference time, enabling quantitative performance evaluation. This approach allows efficient on-device object recognition and real-time visualization. Overall, the application integrates all stages of the development pipeline – from dataset construction and model training to on-device deployment – demonstrating a modular, reliable, and efficient solution for mobile real-time object detection in urban environments.

7. Implementation of the Real-Time System

7.1 Installation and Configuration Procedure

The real-time object recognition system is implemented as an Android mobile application distributed to users in the form of an APK file. Since Android, by default, blocks the installation of applications from unknown sources, users must first adjust their system settings before installation. Specifically, within the “Security” section, the option allowing the installation of applications from unknown sources must be enabled, thereby permitting the installation of local APK packages not originating from the Google Play Store. The application is compatible with devices running Android SDK 21 (Android 5.0) or higher. This requirement is technically grounded, as modern video-processing mechanisms and TensorFlow Lite components rely on API features unavailable in earlier Android versions. Once the APK file is downloaded, the installation is initiated from the file manager, after which the operating system prompts the user to confirm the installation. If additional permission requests appear, the user is redirected to the appropriate settings panel. Upon successful installation, the application icon appears on the device’s home screen. During the first launch, the system requests access to the camera – an essential requirement for real-time recognition. If access is denied, the application cannot function, as the camera serves as the primary data source for detection and classification. Unlike server-based computer vision systems, the presented application does not require an Internet connection; all computations, including object detection and classification, are performed locally on the device through the integrated TensorFlow Lite model. Once the necessary permissions are granted, the application transitions to the live camera view. The system processes each

frame in real time, highlighting detected objects with an orange bounding box supplemented by a class label and confidence score. This enables users to visually observe and interpret detection outcomes directly on the screen in an intuitive manner.

7.2 Interface Structure

The interface of the application is designed to be minimalistic and user-centered, ensuring uninterrupted real-time visualization of detection results. At the first launch, the system displays a camera access prompt (Fig. 6), which is triggered automatically. After permission is granted, the application opens the main working window, where the live camera feed is displayed along with detection overlays. These visual elements – including bounding boxes and textual labels – are updated dynamically for each processed frame, providing users with accurate, immediate feedback on all recognized objects.

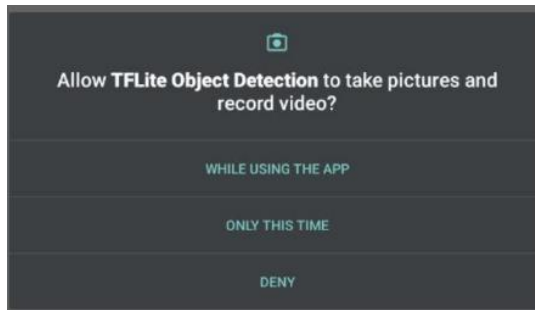


Figure 6. Request to use the camera
Source: Own processing

The main screen of the application presents a window displaying the active video stream from the device camera (Fig. 7). In real time, the application analyzes incoming frames and visualizes the results of object recognition. Detected objects are highlighted with an orange bounding box, accompanied by a label indicating the predicted class and the corresponding confidence score. By swiping upward, the user can access the control panel (Fig. 8), which provides various configuration options. This panel displays the current inference time and allows the adjustment of several parameters, including the confidence threshold, the maximum number of results, and the number of processing threads. Additionally, the user may select the computational delegate (CPU, GPU, or NNAPI) and choose the machine learning model to be used for inference.



Figure 7. Main Screen
Source: Own processing

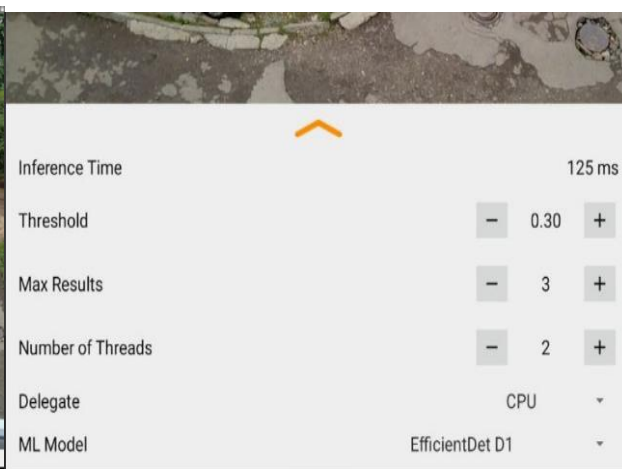


Figure 8. Control Panel
Source: Own processing

7.3. Recommended Mobile Configuration and Real-World Evaluation

The mobile application for real-time object recognition was rigorously evaluated under diverse real-world conditions to assess its performance and reliability. Particular attention was given to object detection accuracy at varying distances, as well as in the presence of occlusions and low-light environments. Testing was conducted on two devices: a smartphone and a tablet. While both devices were equipped with 13 MP cameras, the smartphone featured lower processing capabilities compared to the tablet. The technical specifications of the devices are summarized in Table 5.

Table 5. Technical specifications of devices used for testing

Device	Camera	OS	Processor
Smartphone	13 MP	Android 5.1	Octa-core (4x1.5 GHz Cortex-A53 & 4x1.0 GHz Cortex-A53)
Tablet	13 MP	Android 11	Qualcomm Snapdragon 662 (4x2 GHz Cortex-A73 & 4x1.8 GHz Cortex-A53)

Source: Own processing

The evaluation focused on measuring the inference time for different models on each device, as summarized in Table 6. As expected, object recognition was consistently faster on the more powerful tablet, while overall performance trends remained comparable across devices, demonstrating model consistency under varying hardware configurations.

Table 6. Object recognition time for different models on smartphone and tablet

Model	Smartphone (ms)	Tablet (ms)
EfficientDet D0 (512x512)	300–400	80–90
EfficientDet D1 (640x640)	400	130
SSD MobileNet V2 (320x320)	100	60
SSD MobileNet V2 FPNLite (320x320)	400–600	150
SSD MobileNet V2 FPNLite (640x640)	600	280

Source: Own processing

At close distances (up to 5 meters), the system demonstrated consistently high recognition accuracy, ranging from 82% to 94%, confirming reliable performance under typical monitoring conditions. At medium distances (5–15 meters), accuracy remained stable with only a moderate decline, averaging 72–88%, which indicates that the model maintains robust detection capabilities even when objects occupy a reduced portion of the frame. At long distances (beyond 15 meters), accuracy decreased but remained operationally acceptable for mobile-based surveillance, ranging between 55% and 70%, with the highest values reaching 74%.

Occlusions caused by environmental elements such as vegetation, poles, or passing vehicles produced a measurable but manageable impact. At close range, partially occluded objects were still recognized with 70–88% accuracy. At medium distances, occlusions reduced accuracy to 55–70%, while at long distances accuracy dropped more substantially to 35–50%, which remains within a realistic range for lightweight real-time detection on mobile devices.

Under low-light conditions, including evening and nighttime illumination, the system maintained recognition accuracy in the range of 58–75%, depending on object type and ambient lighting. These results indicate that the selected model configuration is capable of delivering reliable performance across diverse real-world scenarios, even under reduced visibility and partial occlusions.

Considering these results, SSD MobileNet V2 FPNLite (320×320) was selected as the default model for the application. The choice was based on formal criteria: a balanced trade-off between inference speed, accuracy, energy efficiency, and memory footprint, making it well-suited for deployment on a wide range of mobile devices. The recommended mobile configuration includes this model with a CPU delegate and a confidence threshold of 0.5, providing reliable detection at close and medium ranges while preserving real-time performance and energy efficiency. The corresponding inference time on the smartphone is approximately 400–600 ms, with

reduced times on higher-performance devices. This configuration ensures practical usability without compromising detection capabilities for typical urban traffic monitoring scenarios.

Overall, the combined evaluation and configuration process demonstrates that the developed system is capable of performing effective real-time object detection on mobile devices, achieving a practical balance between accuracy, speed, and resource utilization. These findings confirm that the recommended configuration is appropriate for basic traffic monitoring in urban environments, providing both reliability and operational efficiency under real-world conditions.

Conclusion

This study has presented the development, implementation, and evaluation of a CNN-based real-time object detection system for basic traffic monitoring in urban scenes, integrating deep learning methodologies, robust data collection, and practical mobile deployment. The work demonstrates a complete pipeline, from dataset acquisition and preprocessing to model training, optimization, and real-world evaluation, highlighting the challenges and solutions inherent to deploying high-performance machine learning models on resource-constrained devices for traffic monitoring purposes. A critical first step in system development involved assembling a diverse and representative dataset. Data were collected from publicly available sources such as MS COCO and VISDRONE, supplemented with proprietary images to ensure comprehensive coverage of the target traffic-related object classes – person, car, bus, and truck. Attention was given to the diversity of image conditions, including variations in lighting, angles, distance, seasonal and weather conditions, and partial occlusions. Statistical analysis and heatmap visualization of object distributions confirmed that the dataset adequately represented real-world urban traffic scenarios, although certain classes, such as trucks and buses, were underrepresented, suggesting areas for future dataset expansion.

The selection of machine learning models was guided by the dual goals of achieving reliable traffic object recognition and maintaining real-time performance on mobile devices. EfficientDet and SSD MobileNet V2 variants were chosen for their demonstrated ability to balance speed and precision. Model training leveraged TensorFlow and Google Colab computational resources, employing careful data partitioning into training, validation, and test subsets, as well as extensive data augmentation to enhance generalization. Iterative tuning of hyperparameters, including batch size and training steps and the use of mean Average Precision allowed for performance evaluation across the traffic-relevant object classes.

Post-training optimization was a crucial component, involving conversion to TensorFlow Lite format and quantization, which significantly reduced model size and computational demands while maintaining acceptable accuracy. The optimized models were successfully integrated into an Android application designed using a Model-View-Controller architecture. Key components – including CameraFragment, ObjectDetectorHelper, and OverlayView – enabled seamless real-time traffic object detection and visualization, while the user interface offered adjustable parameters, such as confidence thresholds, inference threads, and model selection, allowing adaptation to different traffic monitoring scenarios.

Comprehensive real-world testing demonstrated that the system performs robustly under a variety of urban traffic conditions. Close-range and medium-range object detection achieved high accuracy (up to 91% at close distances), while long-range detection was more challenging due to smaller object sizes in the frame. Occlusions and low-light conditions were identified as additional factors reducing performance, indicating the need for continued refinement in preprocessing and model design. Performance evaluations across devices with differing computational capabilities confirmed that the system maintains consistent behavior, achieving faster inference on more powerful hardware while preserving accuracy trends across models.

In conclusion, this research successfully demonstrates the feasibility of deploying CNN-based object recognition systems for basic traffic monitoring in urban environments on mobile platforms, achieving a practical balance between accuracy, speed, and usability. The study underscores the importance of diverse, high-quality datasets, careful model selection and tuning, and targeted optimization for mobile execution. The resulting application provides an intuitive and functional interface for real-time traffic object detection, and empirical testing confirms its reliability across varying environmental and hardware conditions.

References

- Ardito, L., Coppola, R., Malnati, G., & Torchiano, M. (2020). Effectiveness of Kotlin vs. Java in Android app development tasks. *Information and Software Technology*, 127, 106374. <https://doi.org/10.1016/j.infsof.2020.106374>
- Bursa, S. Ö., Durmaz İncel, Ö., & Işıklar Alptekin, G. (2023). Building lightweight deep-learning models with TensorFlow Lite for human activity recognition on mobile devices. *Annales des Télécommunications*, 78, 687-702. <https://doi.org/10.1007/s12243-023-00962-x>
- Daoudi, W., Shang, W., & Zou, Y. (2019). An exploratory study of MVC-based architectural patterns in Android apps. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19)* (pp. 1711-1720). <https://doi.org/10.1145/3297280.3297447>
- Datactics. (2024). *The importance of data quality in machine learning*. [online]. [cited 19.10.2025]. Available online at: <https://www.datactics.com/blog/the-importance-of-data-quality-in-machine-learning/>
- David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Regev, S., Rhodes, R., Wang, T., & Warden, P. (2020). *TensorFlow Lite Micro: Embedded machine learning on TinyML systems*. <https://www.scribd.com/document/773516821/TensorFlow-Lite-Micro-Embedded-Machine-L>
- Dusek, V. (2024). *People-detector: Python app for people monitoring from UAV* [GitHub repository]. GitHub. [online]. [cited 17.10.2025]. Available online at: <https://github.com/vdusek/people-detector>
- Hirsch, M., Mateos, C., & Majchrzak, T. A. (2025). Exploring smartphone-based edge AI inferences using real testbeds. *Sensors*, 25(9), 2875. <https://doi.org/10.3390/s25092875>
- Hua, H., & Chen, J. (2024). Attention pyramid networks for object detection with semantic information fusion. *International Journal on Semantic Web and Information Systems*, 20(1). <https://doi.org/10.4018/IJSWIS.359769>
- Kantarıcı, S. B. (2024). *Android-App-for-Object-Detection* [GitHub repository]. GitHub. [online]. [cited 17.10.2025]. Available online at: <https://github.com/kantarcise/Android-App-for-Object-Detection>
- Khadka, R., Hassanzade, A., & Heilman, M. (2025). DREAMS: A Python-based framework for deep learning model card generation. *SoftwareX*, 22. <https://doi.org/10.48550/arXiv.2409.17815>
- Kühlechner, R., Zschech, P., Funk, A., Linden, M., & Krcmar, H. (2025). Object Detection Survey for Industrial Applications with Focus on Quality Control. *TechRxiv*. August 26, 2025. <https://doi.org/10.36227/techrxiv.175616901.10303345/v1>
- Lin, T.-Y., et al. (2025). *MS COCO dataset. Papers With Code*. [online]. [cited 12.10.2025]. Available online at: <https://paperswithcode.com/dataset/coco>
- Mazuera-Rozo, C., Escobar-Velásquez, C., Espitia-Acero, J., Vega-Guzmán, D., Trubiani, C., Linares-Vásquez, M., & Bavota, G. (2022). Taxonomy of security weaknesses in Java and Kotlin Android apps. *Journal of Systems and Software*, 187, 111233. <https://doi.org/10.1016/j.jss.2022.111233>
- Mehra, S., Rajput, S., & Paul, J. (2022). Determinants of adoption of latest version smartphones: Theory and evidence. *Technological Forecasting and Social Change*, 175, 121410. <https://doi.org/10.1016/j.techfore.2021.121410>
- Mohammed, A., Goitia, D., Ahmad, S. A., & Ramlal, C. (2025). Revealing resilience: AI anomaly detection driven design considerations for Cyber Physical Systems supporting critical infrastructures in Small Island Developing States. *Insights into Regional Development*, 7(3), 148-169. <https://doi.org/10.70132/p6489444663>

Mumuni, A., & Mumuni, F. (2022). Data augmentation: A comprehensive survey of modern approaches. *Array*, 16, 100258. <https://doi.org/10.1016/j.array.2022.100258>

Pulgarín-Ospina, M., Pineda-Rodríguez, P., & Castaño-Guerrero, J. (2024). Optimizing deep learning models for edge computing in mobile environments. *Procedia Computer Science*, 246, 2549–2557. <https://doi.org/10.1016/j.procs.2024.09.443>

Qiao, L., Li, Y., Song, Y., Zhang, C., Dong, W., & Xu, C. (2021). DeFRCN: Decoupled Faster R-CNN for few-shot object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. https://openaccess.thecvf.com/content/ICCV2021/papers/Qiao_DeFRCN_Decoupled_Faster_R-CNN_for_Few-Shot_Object_Detection_ICCV_2021_paper.pdf

Reis, D., Neto, J. C., Haddad, D. B., & de Medeiros, M. F. (2023). Real-time flying object detection with YOLOv8. <https://doi.org/10.48550/arXiv.2305.09972>

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 4510-4520). <https://doi.org/10.48550/arXiv.1801.04381>

Singhal, K., et al. (2024). *Object-Detection-In-Urban-Environment* [GitHub repository]. GitHub. . [online]. [cited 12.10.2025]. Available online at: <https://github.com/kksinghal/Object-Detection-In-Urban-Environment>

St-Hilaire, M.-O., & Carpentier-Roy, J. (2024). *Urban-detection* [GitHub repository]. GitHub. [online]. [cited 12.10.2025]. Available online at: <https://github.com/VilledeMontreal/urban-detection>

Tan, M., Pang, R., & Le, Q. (2019). *EfficientDet: Scalable and efficient object detection*. <https://doi.org/10.48550/arXiv.1911.09070>
TensorFlow. (2024). *Post-training quantization*. [online]. [cited 11.10.2025]. Available online at: https://www.tensorflow.org/lite/performance/post_training_quantization

Ultralytics. *VisDrone. Ultralytics YOLO Documentation*. [online]. [cited 11.10.2025]. Available online at: <https://docs.ultralytics.com/datasets/detect/visdrone/>

V7. (2024). *Mean average precision (mAP) explained: Everything you need to know* [online]. [cited 11.10.2025]. Available online at: <https://www.v7labs.com/blog/mean-average-precision>

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. <https://doi.org/10.48550/arXiv.2207.02696>

Wang, C.-Y., Yeh, I.-H., & Liao, H.-Y. M. (2024). YOLOv9: *Learning what you want to learn using programmable gradient information*. <https://doi.org/10.48550/arXiv.2402.13616>

Wang, X., Tang, Z., Guo, J., Meng, T., Wang, C., Wang, T., & Jia, W. (2025). Empowering edge intelligence: A comprehensive survey on on-device AI models. *ACM Computing Surveys*, 1(1). <https://doi.org/10.48550/arXiv.2503.06027>

Wang, Y., Wang, J., Zhang, W., Zhan, Y., Guo, S., Zheng, Q., & Wang, X. (2022). A survey on deploying mobile deep learning applications: A systemic and technical perspective. *Digital Communications and Networks*, 8(1), 1-17. <https://doi.org/10.1016/j.dcan.2021.06.001>

Funding: The work was funded by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09I03-03-V01-00085.

Data Availability Statement: More information and data can be obtained from the authors on a reasonable request

Author Contributions: Author Contributions: Authors contributed equally. All authors have read and agreed to the published version of the manuscript.

Kateryna HAZDIUK, PhD., Associate Professor, Head of the Department of Computer Systems Software, Yuriy Fedkovych Chernivtsi National University, Ukraine. Research interests: mathematical modeling and computer simulation of

biosimilar processes and systems; artificial intelligence, deep learning, and neural networks for image classification; modeling the spread of infectious diseases using GeoSEIR(D) and cellular automata models.

ORCID ID: <https://orcid.org/0000-0002-7568-4422>

Yuliana BILAK, Mgr., at the Department of Computer Systems Software, Yuriy Fedkovych Chernivtsi National University, Ukraine. Her current research focuses on adaptive visual-inertial odometry, specifically developing methods for dynamic adjustment of measurement noise covariance based on real-time image quality assessment. Her research interests include computer vision, image processing, deep learning, and visual-inertial navigation systems.

ORCID ID: <https://orcid.org/0009-0000-8393-2863>

Liliia SHUMYLIAK, PhD., Associate Professor at the Department of Computer Systems Software, Yuriy Fedkovych Chernivtsi National University, Ukraine and researcher at the Department of E-Government and Digital Technologies, Bratislava University of Economics and Management, Slovakia. Research interests: cybersecurity in e-government, artificial intelligence, and ICT-driven innovation in public management.

ORCID ID: <https://orcid.org/0000-0002-6593-7334>

Luboš CIBÁK, PhD. MBA., Dr.h.c. doc. Ing., Associate Professor, Head the Department of E-Government and Digital Technologies, Bratislava University of Economics and Management, Slovakia. Research interests: digital transformation of public administration, information systems in the public sector.

ORCID ID: <https://orcid.org/0000-0003-3881-7924>

This is peer-reviewed scientific journal <https://jssidoi.org/ird/page/peer-review-policy>

Copyright © 2025 by author(s). Publishing rights by [UAB Sustainability for Regions](#)

This work is licensed under the Creative Commons Attribution International License (CC BY).

<http://creativecommons.org/licenses/by/4.0/>



Open Access