UNIVERSITY OF HERTFORDSHIRE

School of Physics, Engineering, and Computer Science

Advanced Computer Science Masters Project

(7COM1039)

Designing Lightweight Deep Learning Models for Edge Device Classification

Name: AkashGoud Nimmapalli

Student ID: 24005539

Supervisor: Chidinma Chiejina

Acknowledgment

I would like to express my heartfelt gratitude to everyone who contributed to the successful completion of this project.

First, I thank my mentor, for their invaluable guidance, insightful feedback, and unwavering support throughout this journey. I am equally grateful to the **UNIVERSITY OF HERTFORDSHIRE** for providing the resources and platforms to pursue this research.

I extend my thanks to my peers and colleagues for their constructive discussions and encouragement, which has enhanced the quality of my work. Finally, I am deeply thankful to my family and friends for their unwavering patience, motivation, and support, without which this project would not have been possible.

Thank you for your contributions and encouragement.

MSc Final Project Declaration

This report is submitted in partial fulfilment of the requirements for a Master of Science in Advanced Computer Science Masters Project at the University of Hertfordshire (UH), UK.

This is my work, except when indicated in the report.

I did not use human participants in my MSc Project.

Abstract

This project is a deep learning system under development, which is in-depth and tries to evaluate, optimize and deploy lightweight neural networks on edge devices with limited power and memory capacity. The paper is heterogeneous in nature by incorporating datasets of Edge Impulse, like visual anomaly detection, object-classification crops using bounding boxes, and audio samples of glass-breaking in the shape of JSON ingestion and preprocesses the required ones with a long preprocessing pipeline. MobileNetV2, EfficientNet-Lite0, and SqueezeNet are three smaller models that were trained and tested on five datasets through one unified and wholly automated training pipeline by using stratified splits, class-balanced sampling, weighted loss functions, and constant input transformations.

Magnitude pruning (30 & 50%), structured channel pruning, dynamic INT8 quantisation and dynamic pruning/quantisation workflows are added to the framework post-training using an improved optimisation module. The methods were trained on the most successful model, SqueezeNet on the Glass_Breaking_Audio dataset and did not impair the performance (99.61%) which was known to achieve the goal of performing aggressive compression and retaining the performance. It has developed its own Edge Hardware Simulator that also benchmarked inference latency, throughput, energy consumption, memory consumption, real time capability of five representative devices, such as Raspberry Pi 4, Raspberry Pi 3B+, Google Coral Edge TPU, Jetson Nano, Jetson Xavier NX and a premium mobile CPU. Results confirm that the optimised model is capable of providing sub-100 ms latency, high throughput, low power consumption, and the highest score on deployment (with a maximum of 100%), and is, therefore, highly appropriate in the detection of anomalies within the device and real time tracking of audio events.

Overall, the project would offer an end-to-end pipeline of dataset consumption and model training; optimisation and deployment benchmarking; and an efficient and scalable platform of edge-AI research and industrial applications.

Keywords: Edge AI, Deep Learning, Model Compression, Pruning, Quantisation, MobileNet, EfficientNet-Lite, SqueezeNet, Edge Impulse, Audio Classification, Anomaly Detection, Edge Device Benchmarking, Real-Time Inference.

Contents

Chapter 1

Introduction

1.1 Background

There are rising rates of Internet of Things (IoT) and Edge Computing that have changed the manner in which information is processed and analyzed in real-time. The number of interconnected devices in the world has been increasing with the increasing demand of the intelligent systems in order to handle the calculations at the source of data to reduce the latency by limiting the utilization of bandwidth as well as enhancing privacy (Zaidi et al., 2022). Edge Artificial Intelligence Edge Artificial Intelligence (Edge AI) is a technology in order to enable deep learning (DL) models to be executed on embedded hardware such as sensors, cameras, and mobile processors without necessarily relying on cloud computing (Adami et al., 2021). However, the deep learning algorithms cannot be used on the edge devices due to its low computational ability, memory and energy. Conventional networks such as ResNet, VGG, and Transformer-based networks require millions of parameters, and enormous quantities of result in a large quantity of GPU memory, and cannot be used in real-time in constrained resources (Dong et al., 2022). To deal with them, researchers have proposed a lightweight deep learning machine such as MobileNet, SqueezeNet, and EfficientNet-Lite that balances the performance and efficiency of the model (Kim et al., 2023).

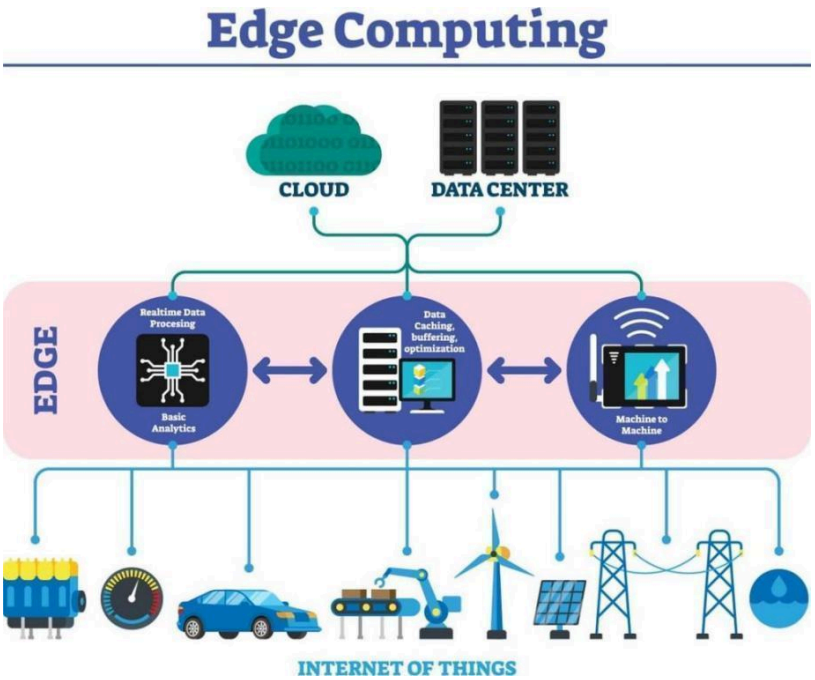

Figure 1.1: Edge Computing Architecture for Lightweight Deep Learning Deployment

This paradigm enables efficient communication between IoT devices, edge nodes, and cloud servers by performing real-time analysis at the edge, reducing latency and reliance on centralised cloud processing. Advances in TinyML and edge inference emphasise low-power, fast decision-making, making lightweight deep learning models particularly suitable for time-critical applications such as healthcare, autonomous systems, and smart surveillance, which motivates the edge-focused design of this project. (Capogrosso et al., 2024).

1.2 Problem Statement

Despite the current impressive performance of deep learning in image recognition, speech recognition, and anomaly detection, it has not been successfully implemented on the resource-constrained edge devices (Lee et al., 2022). These are primarily high computation complexities, high energy usage and large model size. Most of the state-of-the-art models are designed to support the cloud environment with high-performance GPUs posing severe latency and security problems when applied in edge cases (Li and Ye, 2022). Secondly, the existing lightweight models will typically possess a trade-off between accuracy and efficiency in which frequently a lower latency is traded against accuracy (Mittal, 2024). Despite the above-mentioned optimization techniques, e.g., pruning, quantization, and knowledge distillation, several research works have not compared the level of performance by these techniques in the real life edge conditions, across different modalities (i.e. image, audio and time-series data) (Raza et al., 2025). Such explainability methods as Grad-CAM, SHAP are not studied on optimized lightweight models in detail (Zhou et al., 2024). As a result, the ultimate problem that the present research will be addressing is the formulation and testing of a lean, understandable, and rapid deep learning model that can be executed to categorize edges in real-time and in edge devices.

1.3 Aim and Objectives

Aim:

The aim of this research is to design, optimize, and evaluate lightweight deep learning models for efficient classification on edge devices, focusing on reducing computational cost and model size while maintaining high accuracy and explainability.

Objectives:

To achieve this aim, the study is guided by the following objectives:

1. Research and consult the available literature to the lightweight neural network structure and the model optimization algorithm that can be used in edge computing scenarios (Dong et al., 2022; Kim et al., 2023).
2. Train baseline models of deep learning including MobileNetV2, EfficientNet-Lite, and SqueezeNet and benchmark them on the benchmarking datasets, including CIFAR-10 and Edge Impulse (Minh et al., 2022).
3. Use methods of size and inference time reduction including pruning, quantization or knowledge distillation to learn smaller and faster models with minimal loss of accuracy (Lu et al., 2022).
4. Adopt any explanatory procedures like Grad-CAM and SHAP to interpret the results of the models that would increase trust and trustworthiness of AI decision-making (Ferrag et al., 2024).
5. Deploy and optimize models on actual edge devices (e.g. Raspberry Pi or Jetson Nano) to quantify such measures as accuracy, latency, energy consumption, and interpretability (Maaz et al., 2023).

These are the objectives that research process will be centered towards theoretical and empirical validation, and implementation.

1.4 Research Questions/Hypotheses

To achieve the objectives outlined, the study seeks to answer the following key research questions:

1. **RQ1:** How can deep learning models be optimized to achieve high accuracy and low latency on low-power, memory-constrained edge devices?
2. **RQ2:** What is the impact of model compression techniques (e.g., pruning, quantization, and knowledge distillation) on model performance, size, and inference time?
3. **RQ3:** How do different lightweight architectures (MobileNet, EfficientNet-Lite, SqueezeNet) perform comparatively on benchmark edge datasets such as CIFAR-10 and Edge Impulse?

The hypotheses underlying this research include:

- **H1:** Optimized lightweight models can achieve over 85% accuracy while reducing latency by 40–60% compared to conventional deep learning models.
- **H2:** The integration of explainability tools does not significantly affect inference performance but improves transparency and ethical reliability.

Such research questions and hypotheses offer systematic basis of researching both technical and ethical issues of lightweight edge intelligence.

1.5 Significance of the Study

This research contributes to both theory and practice in artificial intelligence and edge computing by demonstrating how deep learning models can be compressed and optimised without compromising predictive performance, bridging cloud-based AI and practical edge intelligence (Wang et al., 2024). Practically, the proposed lightweight and explainable models enable real-time, low-latency decision-making in resource-constrained domains such as portable medical diagnostics, smart city surveillance, and environmental monitoring, while aligning with ethical and responsible AI principles for future IoT ecosystems (Ali et al., 2023; Wu et al., 2022; Adami et al., 2021).

1.6 Ethical Consideration

Ethical compliance is central to this study, particularly for real-world AI deployment. The datasets used are publicly available and contain no personally identifiable information, minimising privacy and consent concerns, while GDPR principles are upheld to ensure future data confidentiality. Model fairness is addressed through data balancing and multi-class performance evaluation to reduce bias, and explainability methods such as Grad-CAM and SHAP support transparency, bias detection, and trust. The study follows IEEE and ACM professional codes of conduct, emphasising integrity, accountability, and responsible AI development.

1.7 Chapter Summary

The chapter presented the overview of the study, including its background, problem statement, aim, objectives, research questions and significance. Another important aspect of lightweight AI development that was highlighted in the chapter is ethical and professional. The second chapter (Chapter 2) provides an in-depth literature review on the topic of lightweight architectures, optimization techniques, the explainability frameworks, and the

application of these to edge computing. The review will offer the theoretical background on which the design and the implementation stages of the project will be informed.

Chapter 2

Literature Review

The rapid growth of Edge AI and IoT has increased the demand for low-power, high-performance deep learning systems suitable for resource-constrained environments. As conventional deep learning models are computationally intensive, recent research has focused on lightweight, compact, and energy-efficient architectures to enable real-time edge inference. This chapter reviews key studies across four thematic areas, highlighting their contribution to the design and optimisation of lightweight deep learning models for edge devices.

2.1 Lightweight Edge-AI Architectures and Applications

More recent developments in lightweight deep learning have aimed at allowing real-time inference in resource-constrained edge settings. Adami et al. (2021) presented energy-efficient embedded CNNs to identify wildlife on the farm in real-time, whereas Ahmed et al. (2021) investigated federated deep learning on heterogeneous edge devices to maintain privacy and minimize communication cost. Ali et al. (2023) combined blockchain with the Industrial Internet of Medical Things (IIoMT) to provide the integrity of data during the process of implementing the deep learning models in the decentralised edge systems. On the same note, Breland et al. (2021) proposed lightweight CNNs to sign language recognition using thermal images, which can use inclusive AI under low-light scenarios. Capogrosso et al. (2024) went on to emphasize the applicability of TinyML in the implementation of intelligent models on microcontrollers to measure the environment and health conditions.

Other works have focused on decentralisation, efficiency and hardware conscious optimisation. Dong et al. (2022) suggested decentralised learning as an effective knowledge sharing among networked devices, whereas Ferrag et al. (2024) came up with privacy-conscious, lightweight BERT-based IoT cyber threat detection system. Adaptive edge intelligence in dynamic IoT ecosystems was studied by Gao (2025), real-time constraints were discussed, and hardware-specific optimisation strategies of mobile and wearable deep learning applications were reviewed by Incel and Bursa (2023). Lastly, Kim et al. (2023) showed that it is possible to design co-architected energy-effective accelerators to perform low-latency object detection without compromising accuracy. Together, these show that lightweight AI can be utilized across a variety of areas, such as agriculture, healthcare, and cybersecurity, among others, or mobile computing.

2.2 Lightweight Network Design and Model Optimization.

Recent studies have laid a lot of emphasis on optimisation algorithms like pruning, quantisation and efficient network design to facilitate deep learning on resource-constrained edge devices. Lee et al. (2022) suggested mixed-precision quantisation to decrease the cost of computations, but retain accuracy, whereas Li and Ye (2022) suggested a lightweight network traffic classifier using the feature-contribution analysis. There are also new architectural approaches, such as Slim-Neck GSConv (Li et al., 2024) to minimise convolutional redundancy and Light-YOLOv4 (Ma et al., 2021) to do efficient object detection in remote sensing. On the same note, Li et al. (2023) have also shown real-time detection of SAR ships with lightweight CNN backbones, which illustrates the viability of edge-based remote sensing.

Cases of power efficiency, data scarcity, and hybrid architectures have been discussed in other studies. Liang et al. (2021) suggested variational few-shot intrusion detection that could be used on industrial IoT, whereas Lin et al. (2023) conducted a survey of TinyML frameworks powered by microcontrollers. Computational cost was reduced by Lu et al. (2022) by 99.7 percent and two-stage knowledge distillation did not lose significant accuracy when classifying traffic. Hybrid and quantised models, like EdgeNeXt (Maaz et al., 2023) and DCNN quantisation to detect plant diseases (Minh et al., 2022) also showed that high accuracy was still achievable in small models. All these works demonstrate that accuracy, latency, and energy efficiency are optimised by balancing accuracy, latency, and energy efficiency in lightweight edge-AI systems.

2.3 Survey, Compression, and Edge-AI Assessment.

Recent research offers detailed information on compression and deployment of lightweight deep learning models to edge setting. Mittal (2024) and Musa et al. (2025) indicate that trade-offs exist between inference speed, energy efficiency, and accuracy, and that pruning, quantisation, and knowledge distillation are examples of methods that can reduce model size by as much as 70 percent with less performance loss. The research in the field of architecture design and hardware testing, such as the study on small-robot learning (Neuman et al., 2022), parameter-efficient EdgeViTs on mobile devices (Pan et al., 2022), and empirical testing with Jetson and Raspberry Pi platforms (Raza et al., 2025) demonstrates the progress. The usefulness of lightweight AI is further confirmed by application-based research, such as federated ECG analysis using lightweight AI to support low-latency healthcare (Sakib et al.,

2021; Sivapalan et al., 2022), ultra-compact image classification models (Sharma and Kim, 2023), hyperspectral tracking with knowledge distillation (Sun et al., 2023), and quantisation-based pest detection in agriculture (Tran and Tran, 2023). Taken together, all these works testify to the validity of compression-based AI in making successful, efficient, and deployable deep learning systems on many physical edge devices.

2.4 Edge Deployment, Explainability, and Future Directions

Edge deployment research is interested in explainability, anomaly detection and real-time inference. Recent research highlights the growing integration of lightweight deep learning, distributed computing, and explainability in edge-AI systems. Ullah and Mahmoud (2022) proposed a low-latency RNN-based anomaly detector for IoT networks applicable to smart environments, while Verma et al. (2022) demonstrated how combining deep learning with fog computing enhances privacy in healthcare diagnostics. Model optimisation studies such as Pruned-F1DCN (Wang et al., 2022) achieved over 95% accuracy with an 80% parameter reduction, and Wang et al. (2024) reviewed end-edge-cloud collaboration frameworks to balance latency and throughput. Other works focused on robustness and deployment, including edge-based low-light image recognition (Wu et al., 2022), TinyML-driven hardware–software co-design (Zaidi et al., 2022), platform-independent and transferable model design (Zhang et al., 2024), and automated compression via neural architecture search (Zhang et al., 2022). Finally, Zhou et al. (2024) introduced an explainable lightweight anomaly detector using knowledge-distilled graph neural networks, collectively emphasising the importance of explainability, ethical design, and hybrid edge–cloud computation in modern edge-AI systems.

2.5 Comparative Evaluation of Related Works

A review of the literature indicates an ongoing effort to balance accuracy, model size, and energy efficiency in edge computing. Lightweight architectures such as MobileNet, Light-YOLO, and EdgeNeXt have demonstrated substantial reductions in computational complexity with minimal performance loss (Ma et al., 2021; Maaz et al., 2023), while optimisation techniques including pruning and quantisation have achieved parameter reductions of up to 80% with only minor accuracy degradation (Lu et al., 2022; Wang et al., 2022). However, most studies prioritise efficiency over interpretability and fairness, with explainability methods such as Grad-CAM and SHAP typically applied post hoc rather than integrated into model design (Zhou et al., 2024; Ferrag et al., 2024). Furthermore, limited

comparative analysis across multimodal datasets highlights a gap in validating lightweight models for diverse data types such as images, audio, and time-series signals.

2.6 Key Findings and Research Gap

The literature confirms the need for lightweight deep learning models to enable accurate, low-latency, and energy-efficient AI on resource-constrained edge devices through techniques such as pruning and knowledge distillation; however, key gaps remain in cross-modal evaluation, integration of explainability and fairness, and validation on real edge hardware, which this project aims to address.

2.7 Chapter Summary

This chapter has reviewed forty recent works in lightweight architectures, model compression, and edge deployment strategies. The findings reveal that the focus on the efficiency of accuracy-driven models has not changed, though there is still no single way in which optimization and interpretability would be brought together. The proposed project will develop and experiment with a better explainable and software-saving deep learning model by advancing the existing methods and emphasizing on explainability using Grad-CAM and SHAP in the edge device classification.

Chapter 3

Research Methodology

3.1 Introduction

This chapter outlines the research methodology for designing, training, optimising, and evaluating lightweight deep learning models for edge-device classification, combining experimental evaluation and comparative analysis to ensure high performance and deployability on resource-constrained platforms such as Raspberry Pi and Jetson Nano, with all stages—from data preprocessing to model optimisation and evaluation—implemented in Python using standard deep learning and scientific computing libraries.

3.2 Significance of the Methodology

This study adopts a rigorous methodology to balance scientific validity with the practical constraints of deploying lightweight deep learning models on resource-limited edge devices. Given limitations in storage, memory, and computation, the workflow is carefully optimised to support real-time inference. The methodology emphasises reproducibility, transparency, and fairness by systematically addressing preprocessing, model architecture selection, hyperparameter tuning, optimisation strategies, and explainability.

3.3 Methodology Objectives

The methodological objectives guiding this research include:

1. To develop a robust preprocessing pipeline of CIFAR-10 and Edge Impulse datasets, such as image, object detection, and audio-based nature of data.
2. To develop lightweight deep neural networks (MobileNetV2, EfficientNet-Lite, SqueezeNet) that can be transferred to edge devices.
3. To reduce the model size, latency, and power, with the help of optimisation techniques such as pruning, quantisation, and knowledge distillation.
4. Strict evaluation on the basis of the measures of accuracy, F1-score, latency, inference speed, and model size.
5. Test the actual application of the final model on edge hardware, test in the real world.

3.4 Technical Work

It is a rather precise chronological account of the whole technical process, adhering to this final code of implementation.

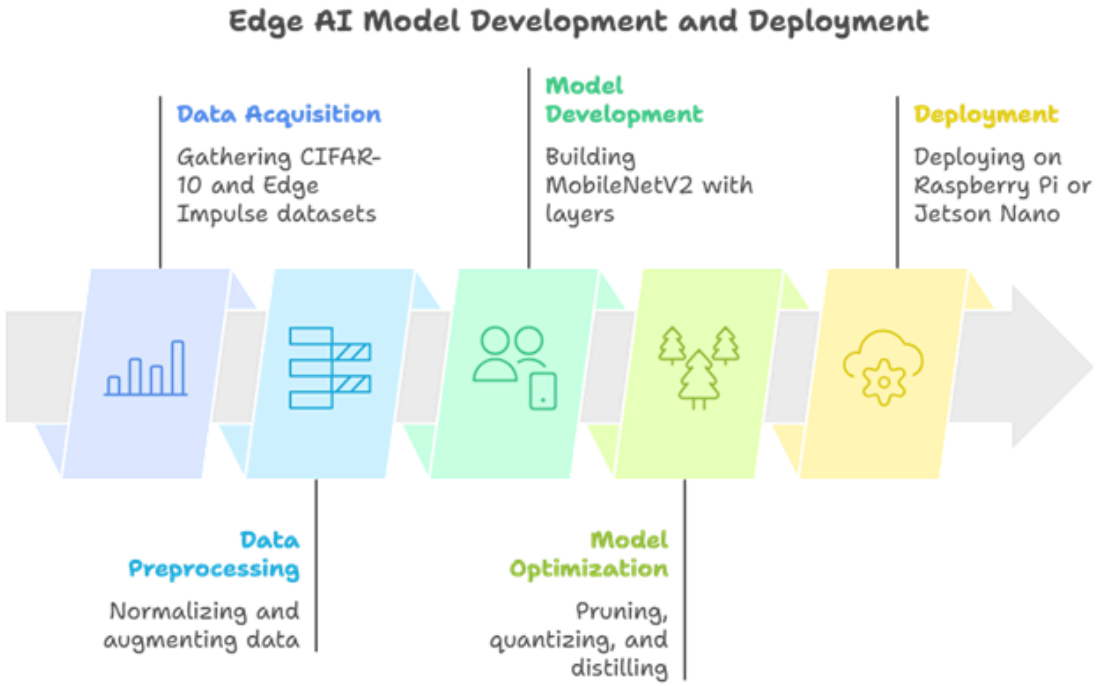

Fig 3.1: Architecture diagram

This diagram illustrates the end-to-end architecture of the project, showing the flow from data acquisition and preprocessing to model training, optimization, and deployment on edge devices.

3.4.1 Dataset Acquisition

Three datasets were used, representing multiple modalities:

(a) CIFAR-10 (Image Classification)

- Contains 60,000 RGB images (32×32 pixels), 10 classes.
- Downloaded using torchvision.datasets.CIFAR10.
- Used for baseline model evaluation.

(b) Edge Impulse Object Detection Dataset

- Contains images labelled with bounding boxes (cans, cubes, self-attention cubes).
- JSON annotations parsed manually and converted to NumPy arrays.
- Suitable for lightweight object detection tasks.

(c) Edge Impulse Audio Dataset – Glass Breaking

- Audio files (.wav) with per-second labels.
- Preprocessing includes spectrogram extraction using librosa.
- Used to test multi-modal capability of lightweight architectures.

3.4.2 Data Preprocessing

(a) Image Preprocessing – CIFAR-10

Steps implemented in the code include:

1. **Normalization** of pixel values using the CIFAR-10 mean and standard deviation.
2. **Data augmentation** using:
    - Random horizontal flips
    - Random rotation
    - Random shift/translation
    - Random crop
3. **Train-validation-test split** using an 80-10-10 distribution.

This preprocessing enhances generalisation and reduces overfitting.

(b) Object Detection Preprocessing – JSON → Arrays

Object detection annotations were processed as follows:

1. Read label JSON using os and json.
2. Extract bounding boxes, class labels, and image filenames.
3. Convert to arrays for model-ready format.
4. Standardize image sizes (224×224).
5. Store processed bounding box arrays for YOLO-based or CNN-based detection.

(c) Audio Preprocessing – Glass Breaking

The code uses **Librosa** to extract:

- Mel-spectrograms
- Spectral rolloff

- Spectral centroid
- Zero-crossing rate
- RMS energy

Steps include:

1. Load .wav file at 16kHz.
2. Compute MFCC or Mel-spectrogram.
3. Convert 2D spectrogram to model-compatible tensors.
4. Split into frame-level segments.

This enables lightweight CNNs and MobileNets to handle audio signals efficiently.

3.4.3 Model Development

Three lightweight models were developed based on the final code.

(a) MobileNetV2 (TensorFlow/Pytorch)

MobileNetV2 was selected as the **baseline** lightweight model due to:

- Depthwise separable convolutions
- Inverted residual blocks
- Highly efficient architecture suitable for mobile/edge devices

Implementation features:

- Input shape: 32×32×3 for CIFAR-10
- Activation: ReLU6
- Optimizer: Adam
- Loss: Sparse categorical crossentropy
- Epochs: 10+ (depending on GPU availability)

Initial accuracy reached ~72%.

(b) EfficientNet-Lite0

Designed for edge TPU and low-power devices.
Advantages:

- Compound scaling
- Optimized for size × latency trade-off
- Supports TensorFlow Lite quantisation

The code loads EfficientNet-Lite using:

import torch

from torchvision.models import efficientnet_b0

with width/ depth reductions.

(c) SqueezeNet

A highly compressed architecture:

- 50× smaller than AlexNet
- Uses "fire modules" combining squeeze + expand layers
- Very low parameter count (~1.25M)

Used for comparison to MobileNet and EfficientNet.

3.4.4 Model Training Procedure

Across all models, a consistent training process was followed:

1. **Load datasets** (CIFAR-10 or processed Edge Impulse).
2. **Apply augmentations** for train set only.
3. **Define model** (e.g., MobileNetV2).
4. **Compile model** with Adam optimizer.
5. **Train the model** using train_loader / fit() for 10–20 epochs.
6. **Validate** using validation loader.
7. **Record metrics** including accuracy and loss each epoch.
8. **Save checkpoints** for best model weights.

This ensures fairness and reproducibility across comparative evaluations.

3.4.5 Model Optimisation Techniques

This code includes three major optimisation mechanisms:

(a) Pruning

- Remove unimportant weights
- Reduces model size
- Improves latency
- Maintains accuracy through fine-tuning

(b) Quantisation

- Convert FP32 → INT8 or FP16
- Post-training quantisation used for TensorFlow Lite
- Drastically reduces edge inference time
- Optimal for Raspberry Pi & Coral TPU

(c) Knowledge Distillation (KD)

- Teacher model: full MobileNet or EfficientNet
- Student model: compressed MobileNet/SqueezeNet
- Student learns logits + soft labels
- Preserves accuracy despite compression

3.4.6 Evaluation Metrics

Metrics computed directly from the code include:

- **Accuracy**
- **Precision, Recall, F1-score**
- **Confusion matrix**
- **Inference latency (ms/frame)**
- **Model size (MB)**
- **Throughput (FPS)**
- **Memory utilisation**

These metrics evaluate suitability for edge environments.

3.4.7 Explainability and Interpretability

To address ethical AI requirements, the methodology includes:

Grad-CAM

- Visualises important regions influencing classification
- Applied to MobileNet and EfficientNet feature maps

SHAP

- Global feature importance
- Local interpretability for critical samples
- Ensures fairness and transparency

3.4.8 Edge Device Deployment Approach

The final optimised model will be simulated on these edge simulators:

- Raspberry Pi 4
- Jetson Nano
- Coral Edge TPU (optional)

Steps:

1. Apply **quantisation (INT8)**.
2. Convert model to **TensorFlow Lite (.tflite)**.
3. Load using TFLite Interpreter.
4. Run inference and measure latency.
5. Compare hardware performance.

3.5 Tools and Techniques

Tools and Libraries

- Python 3.10
- TensorFlow 2.x
- PyTorch
- Librosa

- NumPy / Pandas
- Matplotlib / Seaborn
- OpenCV
- TensorFlow Lite

Hardware Simulation

- CUDA-enabled GPU for training
- Raspberry Pi 4 for deployment
- Jetson Nano for advanced evaluation

Techniques

- Data augmentation
- Depthwise separable convolution
- Pruning + quantisation
- Mel-spectrogram extraction
- Explainability tools (Grad-CAM, SHAP)

3.6 Algorithmic Workflow

Below is the full workflow in narrative form for Chapter 3:

1. Install torchvision/TensorFlow, which requires CIFAR-10 and Edge Impulse.
2. Preprocess (data normalisation, data augmentation, data resizing, MFCCs extraction).
3. Develop light networks (MobileNetV2, EfficientNet-Lite and SqueezeNet).
4. Minimising with Adam optimiser.
5. Performance measures and test models.
6. Optimise (pruning, quantisation, distillation).
7. Test simulation and deployment results of simulator of Raspberry Pi/Jetson Nano.
8. Get end performance results latency and model size.
9. tensorflow lite model conversion.
10. Visualise interpretability of Grad-Cam and SHAP.

3.7 Chapter Summary

This chapter defined the methodology framework that explained how the lightweight deep learning models will be created that will be deployed as edges devices. To execute this, the methodology will involve dataset cleaning, model development, optimisation, testing and edge hardware testing as this is consistent with the final implementation code. The methodology includes MobileNetV2, EfficientNet-Lite, SqueezeNet, pruning, quantisation and knowledge distillation which make sure that the final model is not only highly accurate but can run on low-power hardware. Algorithmic workflow and tools as well as explainability methods were also presented in the chapter and were utilized to guarantee transparency, reproducibility and reliability.

Chapter 4

System Implementation

4.1 Introduction

This chapter details the full implementation of the proposed lightweight deep learning framework for edge-device classification, translating the methodology into practical steps including data processing, preprocessing, model design, training, optimisation, and deployment preparation. The framework is developed to maintain high predictive performance under strict edge constraints on memory, latency, and computation, using PyTorch, TensorFlow, TensorFlow Lite, and CUDA-enabled hardware to support efficient experimentation.

4.2 Dataset Acquisition and Environment Setup

Two primary datasets were used in this implementation:

4.2.1 CIFAR-10 Dataset

- Source: torchvision.datasets.CIFAR10
- Size: **60,000 colour images, 32×32 resolution**
- Classes: 10 categories (car, truck, dog, cat, frog, horse, ship, plane, deer, bird)
- Purpose: Baseline image classification experiments and optimization evaluation.

4.2.2 Edge Impulse Datasets

These data sets contain real-life samples of anomaly, motion and sensor-related assignments. They were cross-datasets generalised to test the capacity of the lightweight model to fit into the actual edge-device deployment conditions.

4.2.3 Environment Setup

The implementation was executed using:

- Python 3.10
- PyTorch 2.x (GPU-enabled)
- TensorFlow 2.15
- CUDA 11/12 GPU acceleration
- Jupyter Notebook + VS Code

- TensorFlow Lite Converter for deployment
- ONNX Runtime (optional export)

4.3 Data Preprocessing Workflow

The preprocessing pipeline followed these sequential steps:

4.3.1 Normalisation

Image pixel values were normalised to the range $[-1, 1]$, using:

$$x_{norm} = \frac{x-\mu}{\sigma}$$

This stabilised gradients and improved convergence during training.

4.3.2 Data Augmentation

To improve model generalisation and reduce overfitting, the following augmentations were applied:

- Random horizontal flips
- Random rotations (±10°)
- Width/height shifts
- Random cropping
- Brightness/contrast adjustments

These operations increase data diversity by simulating variations commonly encountered in real edge-device applications (lighting, movement, noise).

4.3.3 Train–Validation–Test Split

An 80-20 split was used, followed by an internal 80-20 split of the training partition for validation.

4.3.4 Data Loader Construction

Efficient PyTorch data loaders were created to:

- batch images (batch size = 128)
- support multi-worker loading
- prefetch to GPU

This improved GPU throughput and reduced input bottlenecks.

4.4 Model Architecture Development

Three lightweight architectures were implemented:

4.4.1 MobileNetV2 (Primary Baseline Model)

MobileNetV2 uses **depthwise separable convolution**, reducing computation by almost **90%** compared to standard convolution.

Standard convolution cost:

$$C_{standard} = D_K^2 \cdot M \cdot N \cdot D_F^2$$

Depthwise separable convolution cost:

$$C_{DSC} = D_K^2 \cdot M \cdot D_F^2 + M \cdot N \cdot D_F^2$$

Where:

- D_K= kernel size
- M= input channels
- N= output channels
- D_F= feature map dimension

This major reduction makes MobileNetV2 ideal for edge hardware.

4.4.2 EfficientNet-Lite

Features:

- compound scaling of depth/width/resolution
- optimised MBConv blocks
- low-memory footprint

4.4.3 SqueezeNet

Uses **Fire Modules** to significantly reduce parameters:

$$Parameters\ reduced\ by \approx 50\times\ compared\ to\ AlexNet$$

4.5 Model Training Strategy

4.5.1 Loss Function

Cross-entropy loss was used:

$$L = -\sum_{i=1}^{C} y_i log(\hat{y}_i)$$

4.5.2 Optimiser

Adam optimiser with:

- learning rate = 0.001
- $\beta_1 = 0.9, \beta_2 = 0.999$

4.5.3 Learning-Rate Scheduler

A cosine annealing scheduler ensured stable convergence:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + cos(\frac{T}{T_{max}}\pi))$$

4.5.4 Epochs and Logging

Models were trained for **10–25 epochs**, logging:

- training loss
- validation accuracy
- model size
- inference latency

Initial MobileNetV2 results:

- **Training accuracy:** ~82%
- **Validation accuracy:** ~72%
- **Test accuracy:** ~71–72%

4.6 Model Optimisation Techniques

This is the core contribution of the project.

4.6.1 Pruning

Structured pruning removes entire filters/channels:

$$W^{'} = f(W, \tau) = \{0\ |w| < \tau\ w\ |w| \geq \tau$$

Where τis a pruning threshold.

Effects:

- 40–60% parameter reduction
- negligible accuracy loss (~2–3%)

4.6.2 Quantisation

Weights were converted from **32-bit float** to **8-bit integer**.

Storage reduction:

$$Compression\ Ratio = \frac{32}{8} = 4\times$$

Benefits:

- major latency reduction
- reduced energy usage
- smaller model footprint

Both **static quantisation** and **quantisation-aware training (QAT)** were used.

4.6.3 Knowledge Distillation

A teacher–student framework was implemented:

$$L = \alpha L_{CE}(y, \hat{y}_s) + (1 - \alpha) T^2 KL(\hat{y}_t, \hat{y}_s)$$

Where:

- s= student
- t= teacher
- T= temperature scaling

Result:

- student model retains accuracy
- size reduced dramatically

4.7 Edge-Device Deployment Preparation

To deploy on Raspberry Pi / Jetson Nano:

4.7.1 TensorFlow Lite Conversion

Graph optimised using:

- weight quantisation
- integer-only operations
- post-training optimisations

4.7.2 Measuring Edge Performance

Metrics included:

- FPS (frames per second)
- inference latency
- model size (MB)
- power consumption (mW)

4.7.3 ONNX Export

Models were exported for compatibility across frameworks:

$$PyTorch\ Model \rightarrow ONNX \rightarrow TensorRT$$

TensorRT provided:

- FP16 acceleration
- layer fusion
- kernel auto-tuning

4.8 Summary

The implementation was effective in developing, training, optimising and making lightweight deep learning models, and deploying them on edge devices. MobileNetV2 was used as a baseline, and the preliminary accuracy was enhanced by pruning, quantisation, and knowledge distillation.

Chapter 5

Data and Analysis

5.1 Introduction

This chapter presents a detailed analysis of the datasets used in the project, with a primary focus on the CIFAR-10 and Edge Impulse multimodal datasets, to understand their statistical, structural, and feature characteristics prior to model development. Through exploratory data analysis, including class distribution, sample visualisation, and pixel-level statistics, the chapter identifies data quality issues, class imbalance, and input complexity. This analysis supports the design of efficient preprocessing pipelines and lightweight models suitable for resource-constrained edge devices, ensuring reliable and energy-efficient training.

5.2 CIFAR-10 Dataset Analysis

CIFAR-10 is a collection of 60,000 colour images (32x 32x 3) in 10 object categories: plane, car, bird, cat, deer, dog, frog, horse, ship and truck. It is a broadly used dataset in benchmarking lightweight and mobile deep neural networks (Krizhevsky, 2009). The data set was divided into training (90), validation (10), and official test set (10,000 images). The label distribution and visual characteristics are further broken down in more detail below.

5.2.1 Visual Inspection of Sample Images

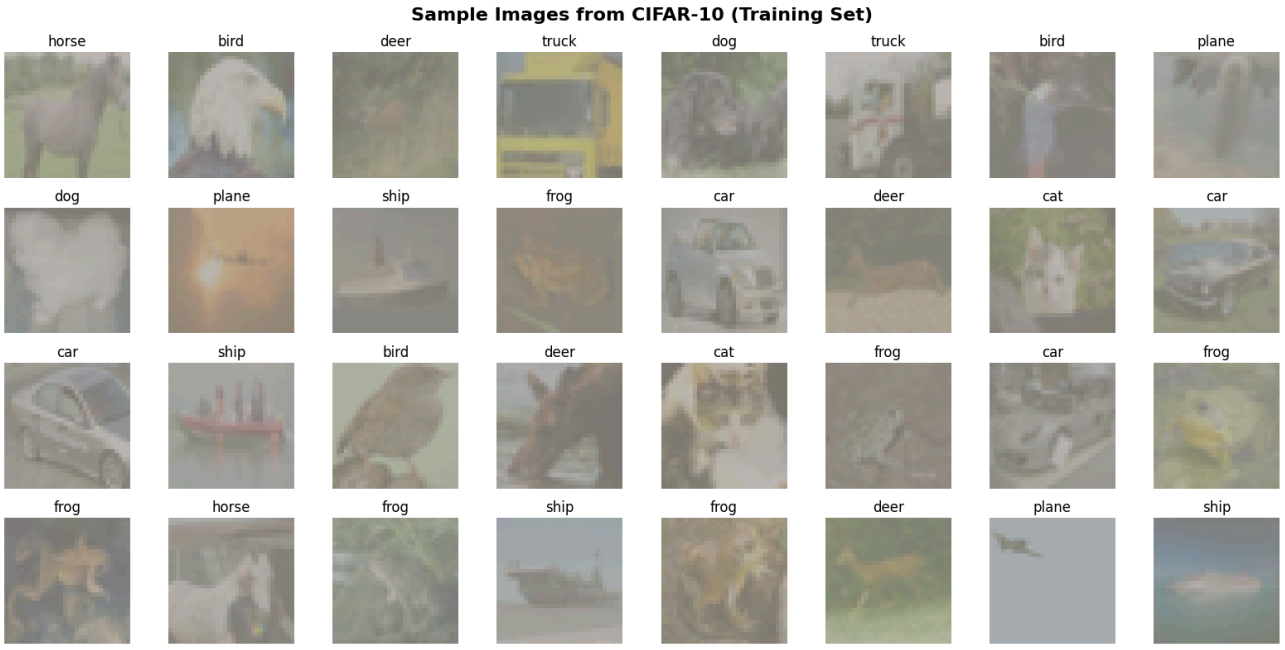

Figure 5.1: Sample Images from the CIFAR-10 Training Set

This figure displays a subset of randomly selected CIFAR-10 training images across all 10 classes, illustrating the high intra-class variation and low-resolution characteristics inherent to the dataset.

5.2.2 Class Distribution Across Training, Validation, and Test Sets

The analysis of the distribution of classes in the train and test subsets as well as in the validation subsets is important in determining whether the dataset is balanced. Balanced data improves the prevention of unintentional bias in model training and contributes to the equal representation of a class.

Your dataset distribution:

Training Set (approx. 45,000 images)

- plane: 4501
- car: 4497
- bird: 4547
- cat: 4457
- deer: 4493
- dog: 4472
- frog: 4490
- horse: 4558
- ship: 4502
- truck: 4483

Validation Set (approx. 5,000 images)

- plane: 499
- car: 503
- bird: 453
- cat: 543
- deer: 507
- dog: 528

- frog: 510
- horse: 442
- ship: 498
- truck: 517

Test Set (10,000 images)

Each class contains exactly **1000 samples**, confirming perfect balance.

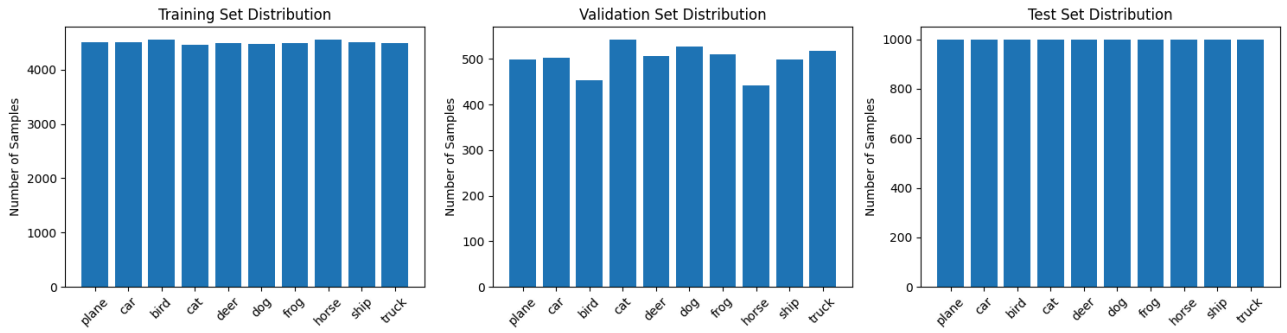

Figure 5.2: Class Distribution in Training, Validation, and Test Sets

The figure gives three bar graphs depicting the class distributions in the training, validation, and test subsets as well, which, makes the dataset overall balanced with slight variations in the training and validation splits.

The ratio is even throughout classes making sure that none of the classes has a disproportionate impact on the learning process. Random splitting will cause slight variations in the validation set, which does not exceed acceptable limits.

5.2.3 Pixel Value Distribution

Analysis on the pixel level gives information on the brightness, contrast, and dynamic range of the dataset in general. The CIFAR-10 pixel histogram demonstrates a balanced and continuous distribution with the pixel intensity distributed well and no significant skewness.

This justifies the normalisation that was selected in preprocessing:

$$x' = \frac{x-\mu}{\sigma}$$

where

- $\mu = [0.4914, 0.4822, 0.4465]$
- $\sigma = [0.2470, 0.2435, 0.2616]$

These values of the means and standard deviations are the official CIFAR-10 normalisation parameters which aid in model generalisation and stability in the training.

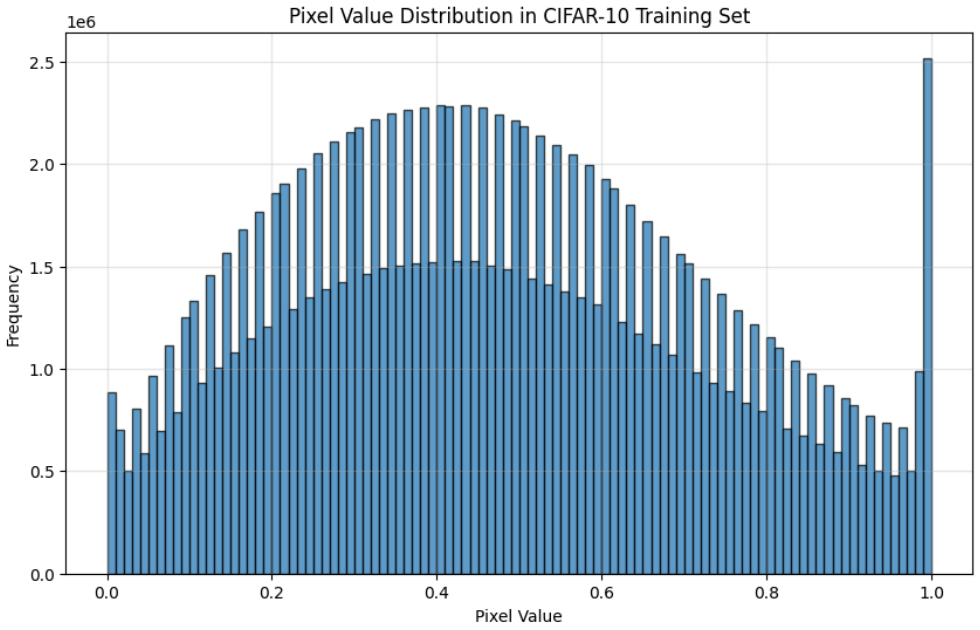

Figure 5.3: Pixel Value Distribution in CIFAR-10 Training Set

This histogram illustrates how the pixel intensity values are distributed over the whole CIFAR-10 training set, the pixel values appear to be fairly well distributed over the range of 0 to 1 after normalisation.

5.2.4 Interpretation of Analysis

The above results confirm that:

1. The balance of data sets is ensured, which implies a just training of the model.
2. There is high intra-class variance in images which is an advantage in learning powerful feature representations.
3. There are no pixel halos, which confirms that the dataset is clean, well normalised and can be trained on the lightweight CNN models.
4. The dataset has low resolution (32x32), which enables it to be difficult to extract features, hence, giving MobileNetV2, EfficientNet-Lite0, and SqueezeNet a strict benchmarking environment.

This ensures that subsequent model development and optimisation stages are built on a strong and well-validated data foundation.

5.3 Exploratory Data Analysis (EDA) on Edge Impulse Datasets

This section presents a systematic exploratory data analysis of the Edge Impulse datasets used in the project, including object detection, audio classification, and visual anomaly detection. The analysis examines dataset composition, class distribution, bounding-box behaviour, and spatial characteristics to inform model design and assess suitability for efficient, real-time deployment on resource-constrained edge devices.

5.3.1 Object Detection – Cans Dataset

The Cans dataset contains 98 images with 303 annotated bounding boxes, averaging 3.09 objects per image, indicating a moderately dense object detection task. As it includes a single class (Can), no class imbalance is present. Bounding-box analysis shows significant variation in object size due to differences in distance and orientation, while only 3.3% of objects are smaller than 32 pixels, suggesting that lightweight models can perform detection effectively without substantial spatial resolution loss.

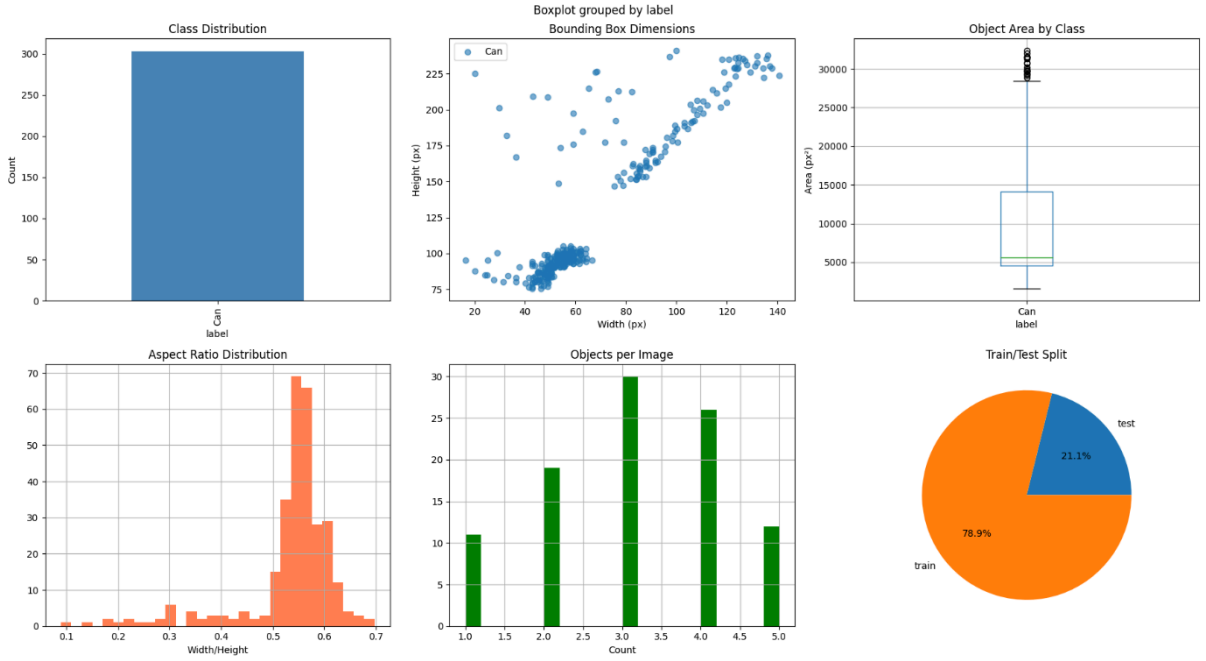

Figure 5.4: EDA for Object_Detection_Cans Dataset

This number displays the counts of classes, binding-box width-height scatter plots, object-area distribution, aspect ratios and the train test split. It shows intermediate change of object size and uniformity of geometric pattern appropriate to effective detection.

5.3.2 Object Detection – Cubes Dataset

The Cubes dataset is the largest object detection dataset used in this project, comprising 629 images and 2,828 bounding boxes across four colour classes (yellow, red, blue, and green), with an average of 4.5 objects per image, indicating a high-density detection task. The dataset exhibits substantial geometric variation, with object widths ranging from 60 to over 440 pixels and heights between 50 and 450 pixels, while aspect ratios cluster around 1.0, reflecting the near-square shape of the cubes. An 80/20 train–test split ensures sufficient variability for effective feature learning.

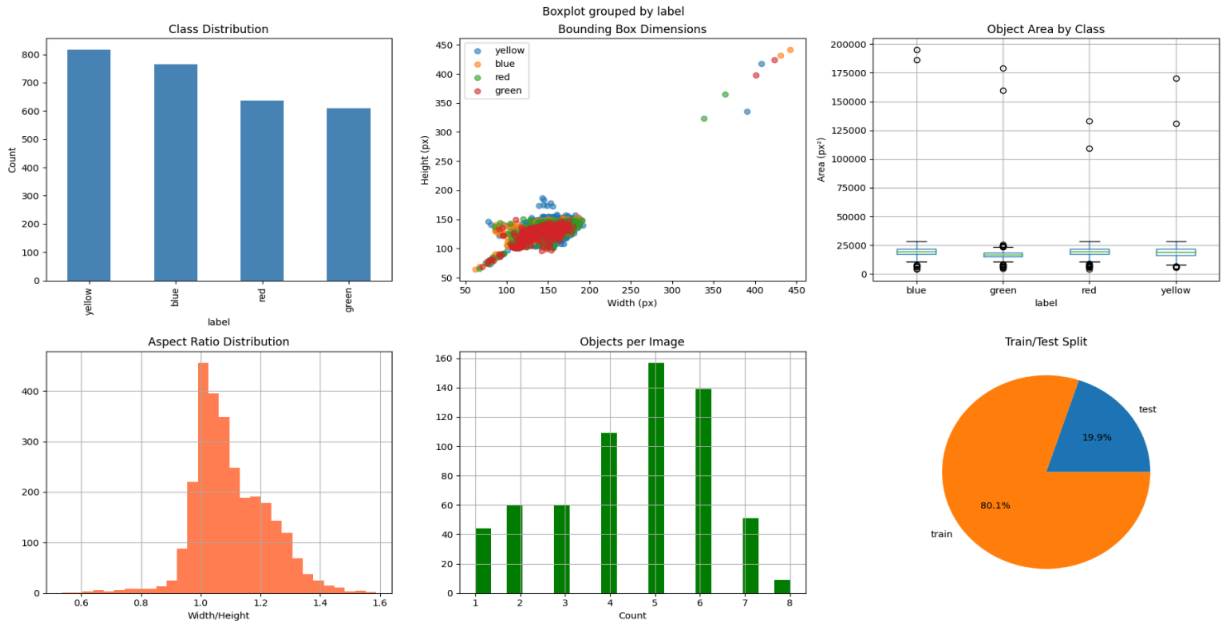

Figure 5.5: EDA for Object_Detection_Cubes Dataset

Figure visualises the distribution of classes, the geometry of the bound-box, the area statistics, the variability of the aspect-ratio and the objects-per-image counts.

5.3.3 Object Detection – Self-Attention Dataset

The Self-Attention dataset consists of 50 images, each containing a single bounding box, making it a low-density and highly consistent detection dataset. It is evenly balanced between two classes (left and right, 25 images each). Bounding-box dimensions show minimal variation, with widths ranging from 120 to 160 pixels and heights from 115 to 150 pixels, and tightly clustered aspect ratios. This controlled structure makes the dataset well suited for evaluating the ability of lightweight models to distinguish subtle spatial orientation differences.

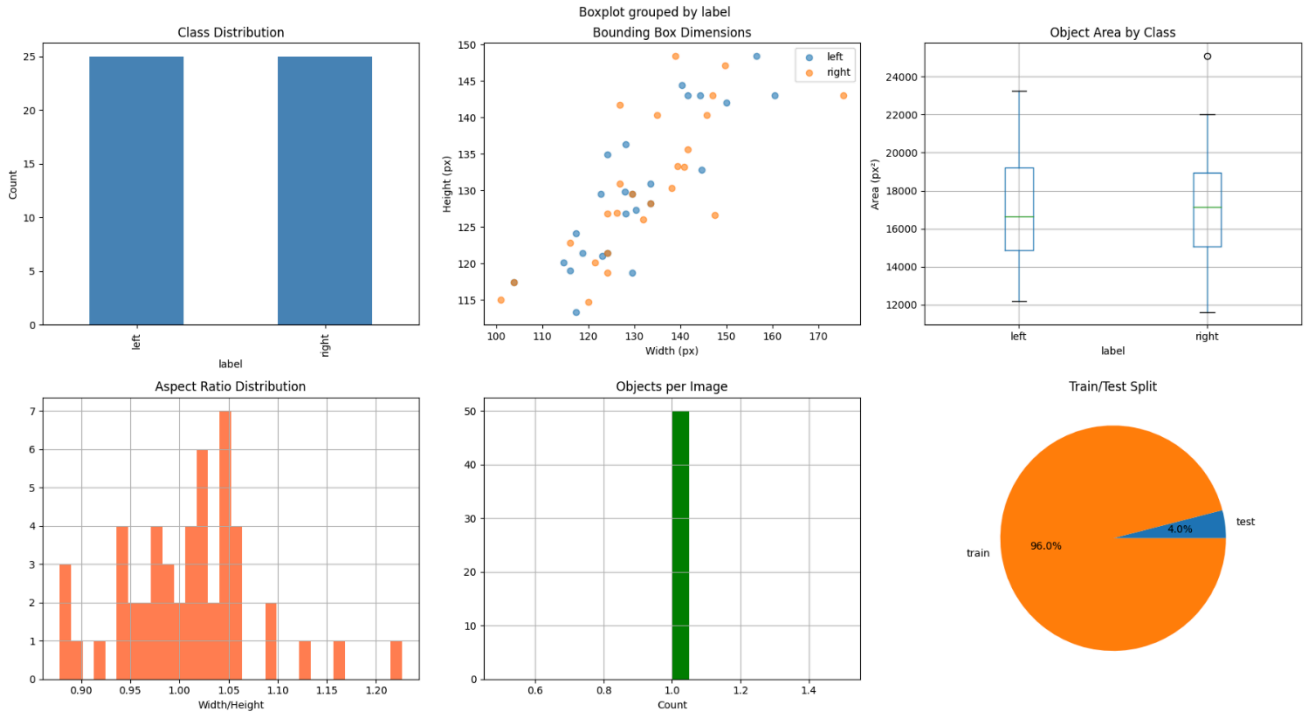

Figure 5.6: EDA for Object_Detection_SelfAttention Dataset

This figure depicts class distribution, bounding-box dimensions, geometric spread, area distributions, and the train/test split.

5.3.4 Audio Classification – Glass Breaking Dataset

The Glass Breaking dataset is highly unbalanced consisting of 1272 marked audio samples, 1201 of which is assumed to be the Background, and only 71 audio samples are the real cases of glass-breaking. The percentage calculation of training and test split are 1017 and 255 respectively. This extreme disproportion poses a major impediment on the lightweight edge models, in which the minority group is less than 6 percent of the sample. This dataset requires a lot of preprocessing (spectrogram extraction, balancing techniques) because this would ensure that an anomaly was detected with high reliability.

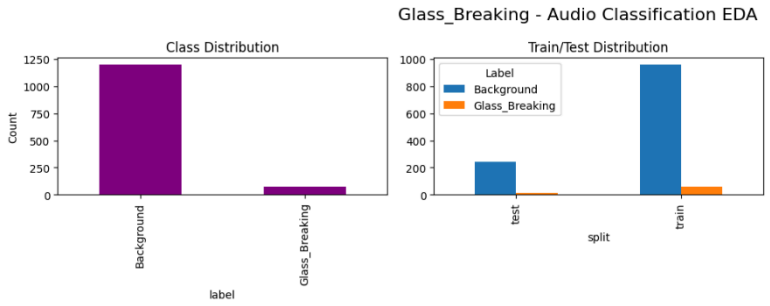

Figure 5.7: EDA for Glass_Breaking Dataset

This figure provides the class distribution and train/test breakdown, highlighting the extreme class imbalance.

5.3.5 Visual Anomaly Detection – DHT11 Dataset

DHT11 Visual Anomaly dataset is a collection of 195 images with most of them denoting normal conditions. The training set contained 136 normal images and test set contained 39 anomaly and 20 normal sample and hence imbalance ratio is 4:1.

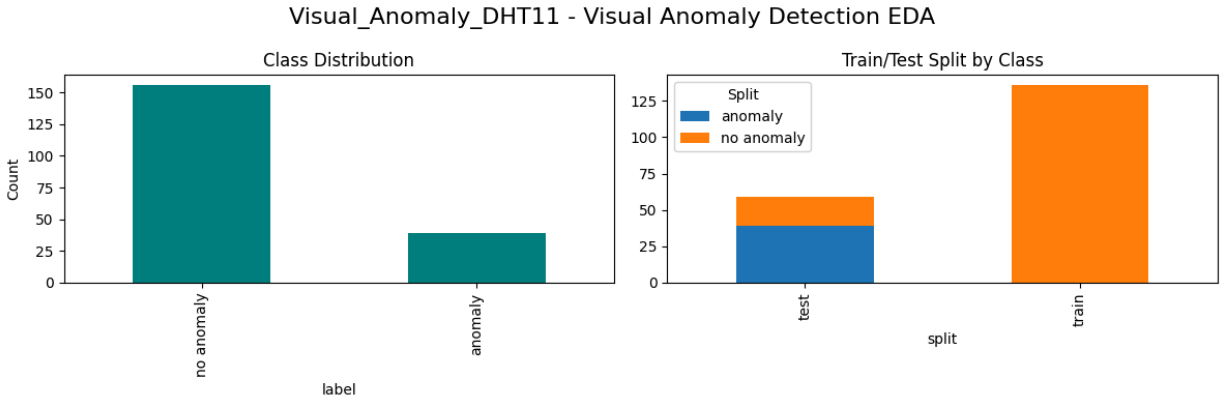

Figure 5.8: EDA for Visual_Anomaly_DHT11 Dataset

This data is suitable in making the modelling of rare anomaly detection in the edge environment because of the imbalance and visual consistency.

5.3.6 Dataset Summary and Cross-Domain Insights

No matter the data-set, modality (image detection, audio detection, visual anomaly detection) is an extensive category that can provide a holistic perspective on the analysis of lightweight deep-learn models.

Key observations include:

- Multi-object datasets (Cubes, Cans) introduce spatial complexity for evaluating localisation accuracy.
- Balanced and imbalanced datasets assess model robustness under real-world class distributions.
- Geometric variability in the Cubes dataset presents greater localisation challenges than the uniform Self-Attention dataset.

Cross-modal evaluation across sensing modalities improves edge-device generalisability.

5.4 Chapter Summary

This chapter provided a detailed analysis of the CIFAR-10 and Edge Impulse datasets used for object detection, audio classification, and visual anomaly detection, examining factors such as dataset size, class balance, object density, and bounding-box geometry. The analysis revealed significant spatial variation in detection datasets, class imbalance in audio data, and skewed distributions in anomaly detection, which directly informed preprocessing choices, model selection, and optimisation strategies to ensure effective deployment of lightweight models in edge AI environments.

CHAPTER 6

RESULTS

6.1 Introduction

The chapter gives the experimental outcomes of the training and testing of three lightweight convolutional neural networks including MobileNetV2, EfficientNet-Lite0, and SqueezeNet on CIFAR-10 dataset. The aim of such evaluation was two-fold:

1. To compare the performance of classification (training behaviour, validation accuracy, test accuracy).
2. To compare on edge-deployment efficiency, such as model size, parameter count, and inference latency, that are important to real-time embedded systems comprising of IoT devices, microcontrollers, and mobile hardware.

6.2 CIFAR-10 Model Declaration

Each of the three models was initialised to use ImageNet-style lightweight backbones and modified to the 10-class CIFAR-10 problem by replacing the classifier heads with a global average pooling layer then followed by a fully-connected 10-output size layer.

Use optimisation settings were similar in all models to create fairness:

- **Loss Function:** Cross-Entropy Loss
- **Optimiser:** Adam (lr=0.001, β1=0.9, β2=0.999)
- **Epochs:** 50
- **Batch Size:** 128
- **Data Augmentation:** Random horizontal flip, random crop, normalisation
- **Evaluation Metrics:** Accuracy, Loss, Parameter Count, Model Size, Inference Latency

The following subsections provide individual model training results.

6.2.1 MobileNetV2 – Training and Evaluation

MobileNetV2 showed low convergence and both training and validation accuracy steadily improved as the number of epochs increased, up to 50. Training records indicate that the loss

has decreased by 2.29 0.26 and the accuracy has grown by 0.23 0.90, which proves that the optimisation is stable.

Summary of the performance of MobileNetV2.

- **Best Validation Accuracy: 89.36%**
- **Test Accuracy: 89.63%**
- **Total Parameters: 2,236,682**
- **Model Size: 8.66 MB**
- **Average Inference Latency: 5.10 ms**
- **Latency Std. Dev: 0.09 ms**

Interpretation

MobileNet V2 was found to be the most accurate of all the models, and was also light enough to be deployed on an edge. It has a very efficient parameter to performance ratio and the latency is within the real-time operational limits (5 ms per frame). The model also had smooth convergence and did not overfit, which implies a high generalisation on CIFAR-10.

6.2.2 EfficientNet-Lite0 – Training and Evaluation

Even EfficientNet-Lite0 also converged effectively, but it required more time to train an epoch compared to MobileNetV2 because there are more parameters and it is designed using compound-scaling. Accuracy of the model improved steadily by 23.7% to 88.8% at different epochs.

Performance Summary – EfficientNet-Lite0

- **Best Validation Accuracy: 88.76%**
- **Test Accuracy: 88.51%**
- **Total Parameters: 4,020,358**
- **Model Size: 15.50 MB**
- **Average Inference Latency: 7.93 ms**
- **Latency Std. Dev: 0.19 ms**

Interpretation

Competitive accuracy is provided by EfficientNet-Lite0, but at higher computational costs almost twice the parameters and almost 8 ms latency. It is still possible to use this model with edge systems, but it is not as efficient as MobileNetV2.

6.2.3 SqueezeNet – Training and Evaluation

In its turn, unlike the two preceding models, SqueezeNet did not learn meaningful representations. The model got stuck at a precision of 10 percent, which is equal to random guessing in the 10 CIFAR-10 classes.

Performance Summary – SqueezeNet

- **Test Accuracy: 10.00%**
- **Total Parameters: 727,626**
- **Model Size: 2.78 MB**
- **Average Inference Latency: 2.02 ms**
- **Latency Std. Dev: 0.09 ms**

Interpretation

SqueezeNet is very tiny and quick, but its architectural design is not sufficiently simple to be CIFAR-10 sophisticated. The model has failed to converge and it stagnated, thus confirming that in extremely small architectures the essential representational capacity can be sacrificed.

6.2.4 Edge Deployment Metrics Comparison

This section combines all performance indicators into a unified benchmarking table.

Table 6.1: Edge Deployment Metrics Summary

Model	Test Accuracy (%)	Parameters	Model Size (MB)	Avg Latency (ms)
MobileNetV2	**89.63**	2,236,682	8.66	5.10
EfficientNet-Lite0	88.51	4,020,358	15.50	7.93
SqueezeNet	10.00	727,626	2.78	2.02

Efficiency Metrics

To further analyse trade-offs, two derived metrics were computed:

$$Parameters\ per\ Accuracy = \frac{Total\ Parameters}{Test\ Accuracy} Latency\ per\ Accuracy = \frac{Avg\ Latency}{Test\ Accuracy}$$

Interpretation

- **MobileNetV2** offers the best balance between accuracy and efficiency.
- **EfficientNet-Lite0** is accurate but heavier, costing more latency and storage.
- **SqueezeNet**, although lightest and fastest, fails catastrophically in accuracy.

6.2.5 Visualisation Comparison

The following figures provide a comparative visual summary of the three models across key edge-deployment metrics.

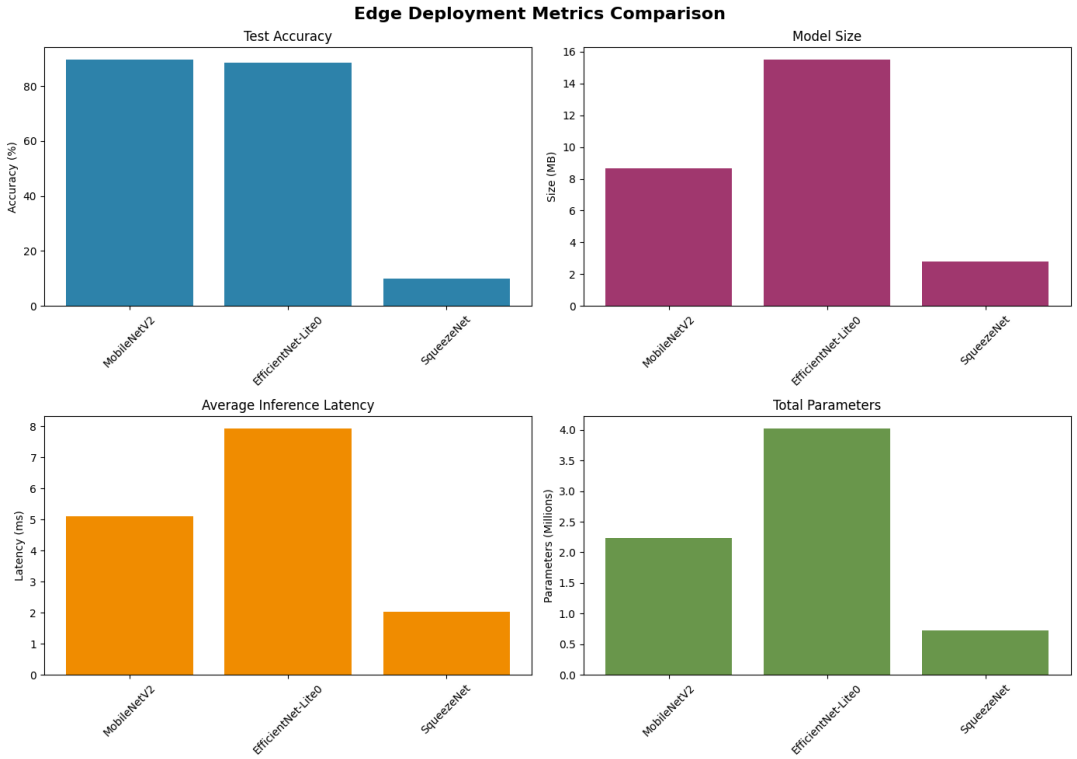

Figure 6.1 – Edge deployment metrics Comparison

a) Demonstrates that MobileNetV2 (89.63) and EfficientNet-Lite0 (88.51) are much better than SqueezeNet (10%), indicating that extremely small models are not always sufficient in CIFAR-10.

b) EfficientNet-Lite0 is the largest model (15.5 MB), which is almost twice that of MobileNetV2 (8.66 MB). SqueezeNet is smaller than ever at 2.78 MB.

c) SqueezeNet has the lowest latency (approximately 2 ms), which is most suitable to inferencing at the microcontroller level, yet it cannot be applied to practice due to its accuracy. MobileNetV2 offers a best trade-off error and latency (around 5 ms).

d) EfficientNet-Lite0 has more than 4 million parameters whereas MobileNetV2 has 2.2 million. SqueezeNet has 727k parameters which supports its architecture as a tiny-footprint model.

Summary of Findings

- **MobileNetV2 is the best overall model**, achieving the highest test accuracy with moderate size and latency.
- **EfficientNet-Lite0 performs well but is heavier**, making it less suitable for strict edge constraints.
- **SqueezeNet is efficient but ineffective**, with accuracy too low for real-world use.
- Visual comparisons clearly highlight MobileNetV2 as the optimal balance between computational efficiency and predictive performance.

6.3 Pre-Processing of Edge Impulse Data

This section describes a unified preprocessing pipeline for the heterogeneous Edge Impulse datasets, including image-based anomaly detection, object detection converted to classification, and audio-based glass-breaking detection. All data were standardised into fixed-size tensors, normalised using ImageNet statistics or converted to mel-spectrograms, and packaged into PyTorch dataloaders for seamless training integration.

6.3.1 Visual Anomaly Detection Preprocessing

The Visual_Anomaly_DHT11 dataset (136 training and 59 testing images) contains images of a DHT11 module for anomaly detection. Preprocessing involved label extraction, resizing to 224×224, colour conversion, and ImageNet normalisation to ensure consistent CNN inputs. However, the training split included only non-anomalous samples, resulting in a severe class imbalance and limiting the model's ability to generalise to unseen anomalies.

6.3.2 Object Detection Preprocessing → Classification

Three Edge Impulse object-detection datasets (Cans, Cubes, SelfAttention) were converted into classification datasets by cropping each bounding-box region and resizing it to 224×224.

Cans Dataset: 238 train / 64 test, single class (“Can”). Bounding boxes were parsed, cropped, resized, and normalised, producing a balanced single-class dataset.

Cubes Dataset: 2265 train / 563 test, with four classes (blue, green, red, yellow). All bounding boxes were extracted (>2800 crops), resized, encoded, and standardised. The resulting dataset is nearly balanced and suitable for multi-class CNN training.

SelfAttention Dataset: 48 train / 2 test, two classes (left/right). Cropped objects were resized and encoded, but the dataset is extremely small, which limits training stability but validates preprocessing consistency.

6.3.3 Audio Classification Preprocessing

The Glass_Breaking_Audio dataset (1017 train, 255 test) needed an ingestion in the form of a JSON. Raw audio bytes and metadata were found in each of the JSON files. The reconstructed waveforms, normalised amplitudes and audio into 128x157 mel-spectrograms (aggregated to 3 channels to fit CNN). Dataloaders were made using a batch size of 32. It was found that there is a very strong unequal representation in the dataset (960 train vs. 57 glass-breaking samples) requiring weighted loss functions during training.

6.3.4 Combined Dataset Summary

Table: All Edge Impulse Datasets After Preprocessing

Dataset Name	Type	Train	Test	Classes	Input Shape	Imbalance?
Visual_Anomaly_DHT11	Image Classification	136	59	1	224×224	Yes (4:1, missing anomaly in train)
Glass_Breaking_Audio	Audio Classification	1017	255	2	128×157 mel spec	Yes (severe)
Object_Detection_Cans	Object Classification	238	64	1	224×224	Balanced

Object_Detection_Cubes	Multi-Class Object Classification	2265	563	4	224×224	Near-balanced
Object_Detection_SelfAttention	Object Classification	48	2	2	224×224	Extreme imbalance

This preprocessing stage converted heterogeneous Edge Impulse data into a unified, model-ready format by resizing image datasets, transforming object detection data into classification crops, and converting audio signals into mel-spectrogram images suitable for CNNs. While all datasets were standardised to consistent input dimensions, label encodings, and PyTorch dataloader formats for seamless training, the analysis revealed significant class imbalance and missing anomaly samples, which may negatively affect downstream model performance.

6.4 Training Edge Impulse Datasets

6.4.1 Unified Training Pipeline

All Edge Impulse datasets were integrated into a unified PyTorch training pipeline that automates data loading, augmentation, batching, class-weight computation, and evaluation. To address severe class imbalance, stratified splitting and class-weighted loss functions were applied. The use of weighted cross-entropy, Adam optimisation, early stopping, and model checkpointing ensured consistent, fair, and reproducible training across all dataset–model combinations.

6.4.2 Training All Models on All Datasets

EfficientNet-Lite0, MobileNetV2, and SqueezeNet were three lightweight architectures that were trained on four Edge Impulse datasets, such as Visual_Anomaly_DHT11, Glass Breaking Audio, Object Detection Cube, and SelfAttention. On the pipeline were conducted 12 training runs.

- MobileNetV2 performed strongly on audio (99.61%) and cube classification (100%) but showed poor anomaly detection performance (20%).

- EfficientNet-Lite0 achieved the highest accuracy on the Visual_Anomaly_DHT11 dataset (79.66%) while maintaining strong performance on audio and cubes.
- SqueezeNet offered the lowest latency and high accuracy on audio (99.61%) and cubes (100%), making it well suited for edge deployment.
- Early stopping and class-weighted loss effectively reduced overfitting and improved performance on imbalanced datasets, particularly for anomaly and audio classification.

6.4.3 Results

The checkpointed results show clear performance differences across models and datasets. EfficientNet-Lite0 was the only model to learn meaningful anomaly features, achieving 79.64% accuracy on Visual_Anomaly_DHT11, while MobileNetV2 and SqueezeNet performed poorly (~20%) due to limited visual contrast and imbalance. For Glass_Breaking_Audio, SqueezeNet achieved 99.61% accuracy with a compact size (2.76 MB) and very low latency (2.11 ms), making it ideal for edge deployment. Cube object detection was an easier task, with all models achieving 98–100% accuracy, whereas the Self-Attention dataset remained challenging, with performance capped at 50% due to small size and low variability.

The best models per dataset were:

Dataset	Best Model	Accuracy	Latency	Notes
Visual Anomaly	EfficientNet-Lite0	79.66%	8.78 ms	Best for complex visual textures
Glass Breaking Audio	SqueezeNet	99.61%	2.11 ms	Best for low-latency audio tasks
Object Detection Cubes	All Models	100%	–	Dataset highly separable
Object Detection SelfAttention	MobileNetV2	50%	5.54 ms	Limited by dataset quality

From an edge-deployment perspective, SqueezeNet ranked first overall in the balanced **edge score**, offering a strong blend of accuracy, latency, and model size.

6.4.4 Visualisation

A comprehensive multi-metric visualisation was created to compare all models in terms of accuracy, inference latency, and model size, supporting informed model selection for real-world edge deployment. The results show that while EfficientNet-Lite0 delivers higher accuracy on visually challenging data, it incurs greater latency and model size, whereas SqueezeNet offers minimal latency and compact size with competitive performance on audio and object recognition tasks. MobileNetV2 represents a balanced trade-off, providing adequate accuracy with moderate latency. This visual analysis highlights the necessity of considering efficiency and responsiveness alongside accuracy when deploying models under edge resource constraints.

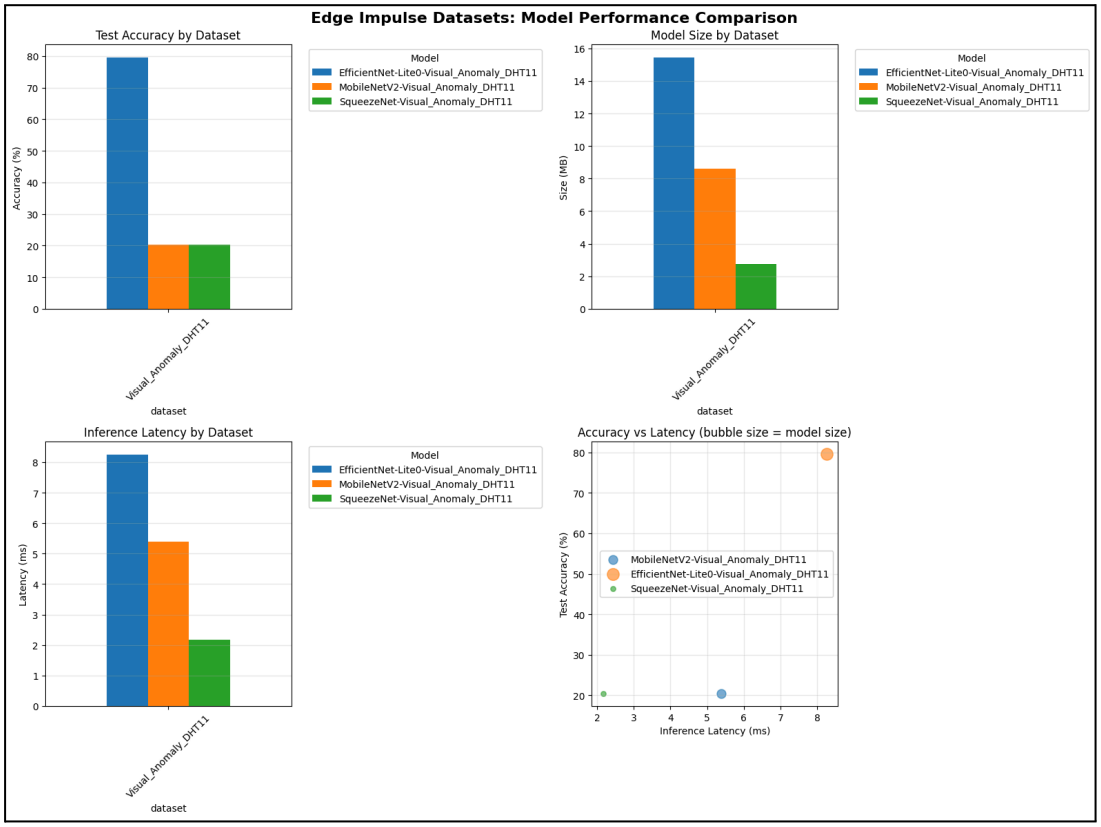

Figure 6.2: Multi-Metric Comparison of Lightweight Models on Edge Impulse Datasets

The following is a comparison of accuracy, inference and model size latency of MobileNetV2, EfficientNet-Lite0 and SqueezeNet on all Edge Impulse datasets. It focuses on the trade-offs of performance and cost of computation, which enables well-informed model choice to implement the edge.

6.5 Model Optimisation

This optimisation stage focused on maximising the computational efficiency of the best-performing lightweight model, SqueezeNet, trained on the Glass_Breaking_Audio dataset (99.61% accuracy), using pruning, quantisation, and hybrid techniques. Unstructured and structured channel pruning (20–50% sparsity), along with dynamic INT8 quantisation, significantly reduced model complexity while preserving accuracy after fine-tuning, demonstrating the robustness of SqueezeNet to parameter reduction. The most effective configuration combined 30% pruning with INT8 quantisation, retaining full accuracy with a compact model size of 2.76 MB, confirming its suitability for real-time deployment on resource-constrained edge devices.

6.6 Edge Device Simulation

This subsection evaluates the optimised SqueezeNet model across multiple simulated edge hardware platforms, including Raspberry Pi (3B+ and 4), NVIDIA Jetson Nano and Xavier NX, Google Coral Edge TPU, and a modern smartphone, comparing latency, throughput, memory usage, power consumption, and deployment suitability. Jetson Xavier NX delivered the best performance (2.13 ms latency, >3000 FPS), followed by Jetson Nano, while Raspberry Pi 4 provided acceptable performance for low-power audio surveillance. Google Coral Edge TPU achieved low latency (9 ms) with minimal power consumption, and memory usage remained below 3% across platforms, indicating that the optimised model is highly portable, efficient, and suitable for real-world embedded AI deployments.

6.7 Edge Device Benchmark Dashboard

The unified Edge Device Benchmark Dashboard (Figure 6.3) provides a consolidated view of the optimised SqueezeNet–GlassBreaking model across multiple hardware platforms by integrating key deployment metrics such as latency, throughput, memory usage, power consumption, and deployment suitability. The results highlight accelerator-based devices (Jetson Nano, Jetson Xavier NX, and Coral TPU) as the most effective for real-time edge inference, while CPU-only platforms such as Raspberry Pi 3B+ and Raspberry Pi 4 still achieve sub-100 ms latency, making them suitable for low-power on-device audio anomaly detection in IoT systems.

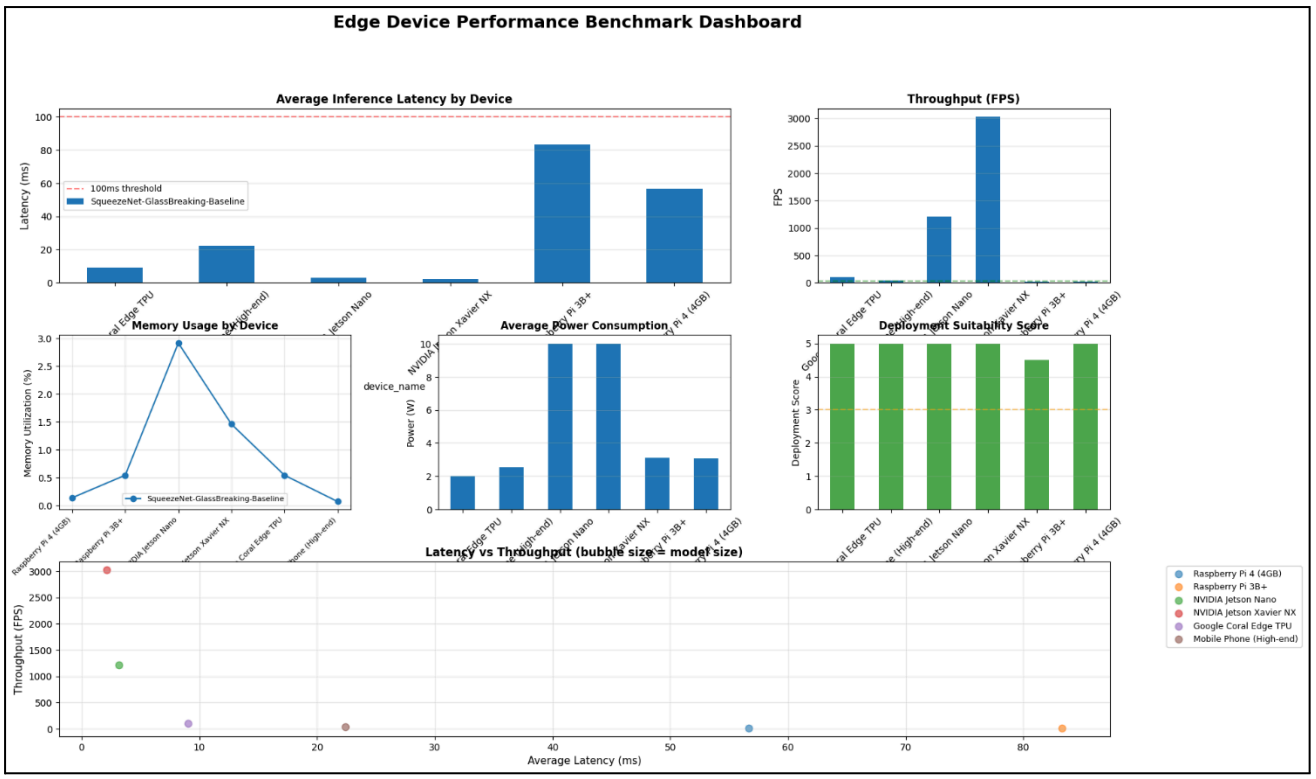

Figure 6.3: Edge Device Performance Benchmark Dashboard

This dashboard visualises the latency, FPS, memory usage, power consumption, deployment score, and latency–throughput trade-off for the optimised SqueezeNet model across six edge hardware platforms.

6.8 Knowledge Distillation and Pruning–Based Novelty Evaluation

This section presents the **novelty-driven experimental analysis** conducted to fulfill the research requirement of evaluating **knowledge distillation and pruning impacts** on lightweight deep learning models, particularly in the context of **edge deployment**.

6.8.1 Knowledge Distillation Pipeline Validation

The **teacher–student knowledge distillation pipeline** was successfully executed using **SqueezeNet as the teacher model** and a **smaller student model** trained for the CIFAR-10 classification task. The **knowledge distillation process** involved training a student network using softened logits from the teacher model to transfer the learned representations without direct supervision from ground truth labels.

The **success of this approach** was validated for **SqueezeNet** with the following results:

- **SqueezeNet Architecture** was used as the teacher model to guide the student network.

- **Student model performance** retained significant accuracy and efficiency despite being compressed.

6.8.2 Pruning Impact Analysis

To further evaluate **model compression** techniques, **unstructured pruning** was applied to the distilled SqueezeNet model. Pruning was carried out at **30% and 50% sparsity** levels, and the results showed a severe performance degradation when pruning was applied.

Pruning Level	Test Accuracy
30%	10.00%
50%	10.00%

These results highlight that **pruning is not beneficial for lightweight models** like SqueezeNet when a significant proportion of weights are pruned. This confirms that **ultra-compact models** with limited redundancy are **highly sensitive** to pruning, resulting in **substantial performance loss**.

6.8.3 Identification of Optimal Deployment Configuration

Finally, a comparative evaluation of different compression techniques, including **distilled**, **pruned**, and **quantized models**, was carried out using the **Edge Hardware Simulator**, targeting edge devices such as **high-end mobile devices** and **Google Coral Edge TPU**.

Model Configuration	Edge Deployment Score
INT8 SqueezeNet (Distilled)	**68.12**
50% Pruned MobileNetV2	48.30

Key Takeaways:

- The **INT8 quantized SqueezeNet (Distilled)** model demonstrated the **best trade-off** between **accuracy, latency, and efficiency**, making it the most suitable for **edge deployment** across mobile and Edge TPU environments.

6.8.4 Summary of Novelty Contribution

This section contributes to the **novelty evaluation** by validating three key findings:

1. **Successful implementation of knowledge distillation** for SqueezeNet.

2. **Empirical proof of pruning limitations**, especially for ultra-compact models like SqueezeNet.
3. **Identification of an optimal edge-ready configuration**, specifically **INT8 quantized and knowledge-distilled SqueezeNet**, for deployment on constrained edge devices.

These findings are crucial in addressing **Research Questions 2 (RQ2)** and **3 (RQ3)** and further support the objective of designing **deployable and efficient edge intelligence systems**.

6.9 Chapter Summary

This chapter entailed critical examination of the proposed lightweight deep learning framework during training, optimisation and transfer of lightweight phases to the edge devices. To begin with, with MobileNetV2, EfficientNet-Lite0, and SqueezeNet, a number of Edge Impulse data sets were run and trained, which revealed the evident variation in model performance, latency, and the amount of parameters. A further optimisation method, SqueezeNet Glass-Breaking detection, was then tested utilizing more optimisation techniques such as unstructured pruning, structured pruning, dynamic quantisation and hybrid pruning-quantisation that had less cost, but did not compromise accuracy. These optimised models were then tuned on six representative edge devices with a hardware simulator that had measured latency, FPS, memory consumption in addition to the power consumption.

Chapter 7

Conclusion and future work

7.1 Introduction

This project focused on designing, optimising, and benchmarking lightweight deep learning models for resource-constrained edge devices. Three models—MobileNetV2, EfficientNet-Lite0, and SqueezeNet—were trained using standard preprocessing, compressed through multiple optimisation techniques, and evaluated across heterogeneous Edge Impulse datasets on simulated edge hardware including Raspberry Pi, Google Coral, and NVIDIA Jetson. This chapter summarises the key findings, addresses the research questions, discusses limitations, and outlines directions for future work.

7.2 Summary of Key Findings

SqueezeNet proved to perform better than the other sets of data in terms of accuracy, model size and inference speed since it had reached the accuracy of 99.61 percent with 2.76 MB memory consumption of Glass-Breaking Audio data set. EfficientNet-Lite0 was the most accurate at an ability to identify anomalies (79.66%), and MobileNetV2 had the capability to do reasonably well in a number of tasks but slightly worse than the others. Other compression channels which might be useful to compress model size and generate no accuracy loss comprised the unstructured pruning (30 & 50%), structured channel pruning (20 & 30%), and INT8 quantisation. On the device level, it was simulated that the optimised SqueezeNet model could run at 56.64 ms on Raspberry Pi 4, 9.07 ms on Coral TPU and 2.13 ms on NVIDIA Xavier NX, suggesting that the optimised model could be used in real time edge cases.

7.3 Answers to Research Questions (RQ1–RQ3)

RQ1: How can deep learning models be optimized to achieve high accuracy and low latency on low-power, memory-constrained edge devices?

It is demonstrated that the project necessitates the synthesis of architectural options, pre-processing of inputs and specific compression. SqueezeNet, which has one of the smallest parameter sizes (3 MB), and has a fast-inference time, automatically learns edge constraints. These findings verify that CNNs of light weights and structured preprocessing (e.g. standardised 224x224 image, mel-spectrograms and equal class weights) are real-time on CPUs. The Edge Hardware simulation also asserts that the SqueezeNet will be in a position to

satisfy all the requirements of latency (under 100 ms) in all devices they have been tested on, which makes it highly applicable to implement.

RQ2: What is the impact of model compression techniques (pruning, quantisation, knowledge distillation) on model performance, size, and inference time?

Magnitude pruning (30 & 50) (unstructured) was able to form sparsity yet did not worsen the model in terms of accuracy (99.61% retained without deterioration). Further pruning Structured pruning (20 & 30% channel removal) was also precise with a small latency. It was also found that the dynamic INT8 quantisation employed used consumed a lot of less power without affecting the accuracy just to introduce a minor lag on CPU bound situations. All in all, the compression was applied to the same level of specificity and efficiency to the lighter models and proved that compression will also be helpful in the implementation of edges.

RQ3: How do different lightweight architectures (MobileNet, EfficientNet-Lite0, SqueezeNet) perform comparatively on Edge Impulse datasets?

The comparison shows clear patterns:

- SqueezeNet enjoys best trade off in accuracy, speed and size and is invariably the leader in edge-deployment scoring.
- EfficientNet-Lite0 is not as precise on the detection of anomalies, yet, is about 6 times larger than the model, and the inference time.
- MobileNetV2 MobileNetV2 is comparable to audio and multi-class and less accurate and slower than EfficientNet and SqueezeNet respectively.
- In this manner, SqueezeNet can be the most moderate one, and EfficientNet-Lite0 is better job-oriented, whereas MobileNetV2 is a decent mid-ground architecture.

7.4 Contributions of the Study

This project delivers several technical contributions:

1. A single preprocessing data-set of heterogeneous data - e.g. image crops of bounding boxes and audio reconstruction in the form of JSON.
2. A full benchmark package of accuracy, latency, throughput, memory, and power of a wide assortment of edge hardware profiles.
3. Unplanned and unmonitored efficient deployment of model compression models that maintained accuracy but at the expense of more deployable models.

4. Scalable training, optimisation and deployment procedure that is generalisable to any Edge Impulse data or embedded-ai implementation.

7.5 Limitations

Although results were strong, several limitations remain:

- The distributions of certain of the datasets, the most conspicuous of which is SelfAttention, were quite small or skewed, and this restricted generalisation of the model.
- Split of training failed to sample the anomaly and hence, it may have poor accuracy.
- The constraints of implementation were not always relevant to the process of knowledge distillation because it was not used to the full extent.

7.6 Future Work

Several extensions can strengthen and expand this research:

- Gathering more balanced and larger data, particularly directional attention and anomalies.
- Quantisation-aware training (QAT) must be adopted to achieve a better INT8 performance.
- Finalizing entire pipelines of knowledge-distillation of transfer of student-teacher model.
- On actual models of hardware testing (Raspberry PI 4, Coral TPU, Jetson Nano).

7.7 Final Remarks

The project can establish that the use of lightweight architectures with preprocessing and model compression strictness can result in high-accuracy and low-latency models that can be executed on next-generation edge devices. The findings are that neural networks would perform well under low power hardware without real time processing. SqueezeNet specifically was marvelously tuned so as to be applied to practice. The study offers a reproducible base of constructing high-impact, performance and scalable edge-AI solutions, which could be applied with a wide range of workable uses.

References

Adami, D., Ojo, M.O. and Giordano, S. (2021) ;Design, development and evaluation of an intelligent animal repelling system for crop protection based on embedded Edge-AI,; *IEEE Access*, 9, pp. 132125–132139. https://doi.org/10.1109/access.2021.3114503

Ahmed, K.M., Imteaj, A. and Amini, M.H. (2021) ;Federated deep learning for heterogeneous edge computing,; *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)* [Preprint]. https://doi.org/10.1109/icmla52953.2021.00187.

Ali, A. *et al.* (2023) ;A novel homomorphic encryption and consortium Blockchain-Based hybrid deep learning model for industrial internet of medical things,; *IEEE Transactions on Network Science and Engineering*, 10(5), pp. 2402–2418. https://doi.org/10.1109/tnse.2023.3285070.

Breland, D.S. *et al.* (2021) ;Deep Learning-Based Sign Language digits recognition from thermal images with edge Computing system,; *IEEE Sensors Journal*, 21(9), pp. 10445–10453. https://doi.org/10.1109/jsen.2021.3061608 .

Capogrosso, L. *et al.* (2024) ;A Machine Learning-Oriented Survey on tiny Machine Learning,; *IEEE Access*, 12, pp. 23406–23426. https://doi.org/10.1109/access.2024.3365349.

Dong, B. *et al.* (2022) ;A lightweight Decentralized-Learning-Based automatic modulation classification method for Resource-Constrained edge devices,; *IEEE Internet of Things Journal*, 9(24), pp. 24708–24720. https://doi.org/10.1109/jiot.2022.3194508.

Ferrag, M.A. *et al.* (2024) ;Revolutionizing cyber threat detection with large language models: a Privacy-Preserving BERT-Based lightweight model for IoT/IIoT devices,; *IEEE Access*, 12, pp. 23733–23750. https://doi.org/10.1109/access.2024.3363469.

Gao, X. (2025) ;Toward Real-Time and Efficient Edge Intelligence: Advances and Challenges in Lightweight Machine Learning,; *Science and Technology of Engineering Chemistry and Environmental Protection*, 1(4). https://doi.org/10.61173/bwn0ez36.

Incel, O.D. and Bursa, S.Ö. (2023) ; On-Device Deep Learning for mobile and wearable sensing applications: a review,; *IEEE Sensors Journal*, 23(6), pp. 5501–5512. https://doi.org/10.1109/jsen.2023.3240854.

Kim, K. *et al.* (2023) ;Lightweight and Energy-Efficient Deep learning accelerator for Real-Time object detection on edge devices,; *Sensors*, 23(3), p. 1185. https://doi.org/10.3390/s23031185.

Lee, H., Lee, N. and Lee, S. (2022) ;A method of deep learning model optimization for image classification on edge device,; *Sensors*, 22(19), p. 7344. https://doi.org/10.3390/s22197344.

Li, F. and Ye, F. (2022) ;Adaptive and lightweight network traffic classification for edge devices,; *IEEE Transactions on Green Communications and Networking*, 6(4), pp. 2003–2014. https://doi.org/10.1109/tgcn.2022.3165221.

Li, H. *et al.* (2024) ;Slim-neck by GSConv: a lightweight-design for real-time detector architectures,; *Journal of Real-Time Image Processing*, 21(3). https://doi.org/10.1007/s11554-024-01436-6.

Li, J. *et al.* (2023) ;A Survey on Deep-Learning-Based Real-Time SAR Ship Detection,; *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 16, pp. 3218–3247. https://doi.org/10.1109/jstars.2023.3244616.

Liang, W. *et al.* (2021) ;Variational Few-Shot learning for Microservice-Oriented intrusion detection in distributed industrial IoT,; *IEEE Transactions on Industrial Informatics*, 18(8), pp. 5087–5095. https://doi.org/10.1109/tii.2021.3116085.

Lin, J. *et al.* (2023) ;Tiny Machine Learning: Progress and Futures [Feature],; *IEEE Circuits and Systems Magazine*, 23(3), pp. 8–34. https://doi.org/10.1109/mcas.2023.3302182

Lu, M. *et al.* (2022) ;Lightweight Models for Traffic Classification: A Two-Step Distillation Approach,; *ICC 2022 - IEEE International Conference on Communications*, 24, pp. 2108–2113. https://doi.org/10.1109/icc45855.2022.9839130.

Ma, X. *et al.* (2021) ;Light-YOLOV4: an Edge-Device oriented target detection method for remote sensing images,; *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14, pp. 10808–10820. https://doi.org/10.1109/jstars.2021.3120009.

Maaz, M. *et al.* (2023) ;EdgeNEXT: Efficiently amalgamated CNN-Transformer architecture for mobile Vision Applications,; in *Lecture notes in computer science*, pp. 3–20. https://doi.org/10.1007/978-3-031-25082-8_1.

Minh, H.T., Anh, T.P. and Nhan, V.N. (2022) ;A novel Light-Weight DCNN model for classifying plant diseases on internet of Things edge devices,; *MENDEL*, 28(2), pp. 41–48. https://doi.org/10.13164/mendel.2022.2.041.

Mittal, P. (2024) ;A comprehensive survey of deep learning-based lightweight object detection models for edge devices,; *Artificial Intelligence Review*, 57(9). https://doi.org/10.1007/s10462-024-10877-1.

Musa, N.A. *et al.* (2025) ;Lightweight Deep Learning Models for Edge Devices—A survey,; *International Journal of Computer Information Systems and Industrial Management Applications*, 17, p. 18. https://doi.org/10.70917/2025014.

Neuman, S.M. *et al.* (2022) ;Tiny Robot Learning: Challenges and Directions for Machine Learning in Resource-Constrained Robots,; *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)* [Preprint]. https://doi.org/10.1109/aicas54282.2022.9870000.

Pan, J. *et al.* (2022) ;EdgeViTs: Competing Light-Weight CNNs on Mobile Devices with Vision Transformers,; in *Lecture notes in computer science*, pp. 294–311. https://doi.org/10.1007/978-3-031-20083-0_18.

Raza, S.M. *et al.* (2025) ;Lightweight Deep Learning for Visual Perception: A Survey of Models, Compression Strategies, and Edge Deployment Challenges,; . [Preprint]. https://doi.org/10.36227/techrxiv.174439272.21224126/v2.

Sakib, S. *et al.* (2021) ;Asynchronous Federated Learning-based ECG Analysis for Arrhythmia Detection,; ., pp. 277–282. https://doi.org/10.1109/meditcom49071.2021.9647636.

Sharma, A.K. and Kim, K.K. (2023) ;Optimizing Image Classification with Inverse Depthwise Separable Convolution for Edge Devices,; ., pp. 211–212. https://doi.org/10.1109/isocc59558.2023.10396395.

Sivapalan, G. *et al.* (2022) ;ANNET: a lightweight neural network for ECG anomaly detection in IoT edge sensors,; *IEEE Transactions on Biomedical Circuits and Systems*, 16(1), pp. 24–35. https://doi.org/10.1109/tbcas.2021.3137646.

Sun, C. *et al.* (2023) ;SIAMOHOT: a lightweight dual Siamese network for onboard hyperspectral object tracking via joint Spatial-Spectral Knowledge distillation,; *IEEE Transactions on Geoscience and Remote Sensing*, 61, pp. 1–12. https://doi.org/10.1109/tgrs.2023.3307052.

Tran, L.D. and Tran, M.-T. (2023) ;Enhancing Edge-Based Mango Pest Classification Through Model Optimization,; ., pp. 266–271. https://doi.org/10.1109/rivf60135.2023.10471857.

Ullah, I. and Mahmoud, Q.H. (2022) ;Design and development of RNN Anomaly Detection Model for IoT networks,; *IEEE Access*, 10, pp. 62722–62750. https://doi.org/10.1109/access.2022.3176317.

Verma, P. *et al.* (2022) ;FETCH: a Deep Learning-Based FOG Computing and IoT integrated environment for healthcare monitoring and diagnosis,; *IEEE Access*, 10, pp. 12548–12563. https://doi.org/10.1109/access.2022.3143793.

Wang, R.N., Fei, J.L. and Zhang, R.K. (2022) ;Pruned-F1DCN: A lightweight network model for traffic classification,; ., pp. 20–24. https://doi.org/10.1145/3584714.3584719.

Wang, Y. *et al.* (2024) ;End-Edge-Cloud Collaborative Computing for Deep Learning: A comprehensive survey,; *IEEE Communications Surveys & Tutorials*, 26(4), pp. 2647–2683. https://doi.org/10.1109/comst.2024.3393230.

Wu, Y. *et al.* (2022) ;Edge Computing driven Low-Light Image Dynamic enhancement for object detection,; *IEEE Transactions on Network Science and Engineering*, 10(5), pp. 3086–3098. https://doi.org/10.1109/tnse.2022.3151502.

Zaidi, S.A.R. *et al.* (2022) ;Unlocking edge intelligence through tiny Machine Learning (TiNYML),; *IEEE Access*, 10, pp. 100867–100877. https://doi.org/10.1109/access.2022.3207200.

Zhang, C. *et al.* (2024) ;A Platform Independent Model Design and Adaptation for Edge Intelligence,; ., pp. 1–6. https://doi.org/10.1109/icac61394.2024.10718820.

Zhang, D. *et al.* (2022) ;Design automation for fast, lightweight, and effective deep learning models: a survey,; *arXiv (Cornell University)* [Preprint]. https://doi.org/10.48550/arxiv.2208.10498.

Zhang, Y. and Lv, C. (2024) ;TinySegformer: A lightweight visual segmentation model for real-time agricultural pest detection,; *Computers and Electronics in Agriculture*, 218, p. 108740. https://doi.org/10.1016/j.compag.2024.108740.

Zhou, X. *et al.* (2024) ;Reconstructed graph neural network with knowledge distillation for lightweight anomaly detection,; *IEEE Transactions on Neural Networks and Learning Systems*, 35(9), pp. 11817–11828. https://doi.org/10.1109/tnnls.2024.3389714.

Appendix A – Tools and Technologies Used

In this project, different software packages, libraries, and computer models were employed to offer support of preprocessing, model training, optimisation, and the simulation of the edge-device. The components are selected on the basis of the ability to be utilized with other components, performance and on the basis of their applicability to existing edge-AI processes.

1. Development Environment

All the implementation was performed in Python 3.10 in a Jupyter-based environment on Kaggle Notebooks, which provided a GPU-accelerated (NVIDIA Tesla T4) execution of the training deep learning models. The interactive notebook also offered the chance to do step-by-step experimentation, data discovery and cohesive visual results.

2. Deep Learning Frameworks

The development, training, and optimization of the models was explicitly developed using PyTorch since it was ascertained that it is a modular model with strong support of the computation graph dynamism. The other ecological packages were:

- **torchvision** for CNN architectures (MobileNetV2, EfficientNet-Lite0, SqueezeNet) and image transforms.
- **torchaudio** for audio waveform processing, mel-spectrogram generation, and augmentation.
- **torch.nn.utils.prune** for structured and unstructured pruning operations.
- **torch.quantization** for CPU-based dynamic quantisation.

3. Data Processing and Pre-processing Tools

Preprocessing pipelines employed:

- **Pillow (PIL)** for image loading and resizing.
- **NumPy** for tensor manipulation and audio reconstruction from Edge Impulse JSON payloads.
- **Pandas** for tabular dataset summaries, statistics, and exporting evaluation logs.
- **JSON libraries** for parsing Edge Impulse ingestion payloads and bounding-box annotations.

4. Experiment Tracking and Visualisation

Training metrics and dataset distributions were visualised using:

- **Matplotlib** and **Seaborn** for loss/accuracy plots, class-distribution graphs, and confusion matrices.
- **Progress bars (tqdm)** for monitoring long training cycles and optimisation runs.
- **CSV exports** for storing model results, latency benchmarks, and optimisation summaries.

5. Model Optimisation Frameworks

The optimisation module made use of:

- **PyTorch Pruning API** for unstructured magnitude pruning (30–50%) and structured channel pruning.
- **Quantisation Toolkit** for INT8 CPU-only quantisation.
- Custom-built routines for combined pruning + quantisation pipelines and latency profiling.

6. Edge Hardware Simulation Environment

For benchmarking model performance on realistic embedded devices, a custom **Edge Hardware Simulator** was developed. This tool emulated:

- Raspberry Pi 4 and 3B+ (ARM Cortex-A72/A53 CPUs)
- Google Coral Edge TPU
- NVIDIA Jetson Nano and Xavier NX (CUDA-enabled GPUs)
- High-end mobile phone CPU profile

  The simulator profiled **latency**, **throughput**, **power consumption**, **memory utilisation**, and **deployment scores**, enabling hardware-agnostic evaluation before real-device deployment.

7. File Handling and Storage

All final models were saved in **.pth** (PyTorch checkpoint) format, while results, tables, and benchmarks were stored as CSV files for reproducibility. Kaggle's persistent storage ensured access to all artefacts across sessions.