International Journal of Emerging Engineering and Technology (IJEET)
ISSN (e): 2958-3764
Volume: 5, Number: 1-2, Pages: 16 - 27, Year: 2026 (Special Issue)

Design and Deployment of Multi-Class Image Classification Model on ESP32 Using a Lightweight CNN Approach

Abdul Hannan * and Rao M. Asif

Department of Electrical Engineering, Superior University Lahore, Lahore, 54000, Pakistan

*Corresponding author: Abdul Hannan (Email: abdulhannan1757@gmail.com)

Received: 14/07/2026, Revised: 29/08/2026, Accepted: 01/09/2026

***Abstract*— Tremendous progress has been made in developing intelligent and autonomous systems in recent years, driven by AI software tools such as Chatbots and large language models (LLMs) for various applications. This has led to the adoption of smart technology solutions. Nonetheless, computer vision requires significant computational power, making it difficult to run on devices such as the ESP32. To showcase the development of smart technologies leveraging controller platforms, the research conducted training and deployment of an optimal multi-class image classification model on the ESP32 microcontroller platform, ensuring acceptable predictive performance across four classes: person, fruit, car, and unknown. It is important to clearly distinguish between the proposed system, which performs image classification and object detection. Regarding model training, the images were preprocessed to minimise computational cost. Nonetheless, the trained model achieved a peak validation accuracy of 99.34±1% when evaluated using five-fold cross-validation, with an average validation accuracy of 98±1% without altering the model or the training process, producing similar results. Moreover, TFLite Quantisation, a variant of the INT8 quantisation technique, was used to optimise the model size. The model was converted to C array format while retaining most features. To improve accuracy during training, the training dataset was augmented with large, high-quality images that had been filtered out. The chosen hardware used for this project was the ESP32-S3 N16R8 microcontroller with built-in OV2640 camera and the 3.2" TFT to visualize the process of computer vision task execution in real time. The model works smoothly and predicts accurately, even with just a few samples per frame, to decrease response time latency. Successfully running the multi-class image classification model on the ESP32 microcontroller is the evidence of full independence of embedded vision solution without having anything to do with cloud technology.**

***Index Terms*— ESP32, Convolutional Neural Network (CNN), TensorFlow Lite, Post-Training Quantization, Multi-class Image Classification, TinyML.**

I. Introduction

In the current era, the utilisation of artificial intelligence (AI) is growing rapidly to perform functions that previously required human intelligence. The increasing requirement of intelligent systems led to the transformation of modern technologies with such devices. Deep learning, which is an example of the implementation of AI, requires high computing power like that of a GPU (graphical processing unit) or cloud servers, which are accessed through IoT or internet-enabled devices. This gives rise to lightweight deep learning model implementation, where both processing and inference are done on local devices without requiring any cloud-based infrastructure. Various kinds of edge devices can be considered, but microcontrollers became popular due to the limitation of energy-efficient, cost-effective, and small-size devices. Due to such limitations, this process started to move towards tiny machine learning (TinyML) and edge AI frameworks which can implement models on ESP32 microcontrollers [1]. The traditional CNNs provide highly accurate results in the case of visual recognition, but involve many floating-point operations making them less suitable for microcontrollers [2]. However, recent developments of frameworks like TensorFlow Lite for Microcontrollers (TFLM) and Edge Impulse have helped in increasing accessibilities for developers. However, practical implementations are facing a number of issues like inference latency, model compression, and efficiency of multi-class prediction [3].

The convolutional deep learning solutions for computer vision faced a number of problems during their implementation on microcontrollers such as ESP32. Such models are highly computationally intensive which makes them less effective in performing multi-class image classification tasks in real-time conditions [4]. Despite using techniques like pruning and quantization, many present lightweight CNN models can only perform the task of classifying one dominant object class at a time and not multiple classes robustly under resource constraints [5].

Though considerable advances have been made, yet there are many challenges associated with the deployment of deep learning models in embedded devices due to small processing capacity, small memory and low energy consumption [6]. The size of the model can be minimized by choosing the few parameters and layers during the process of model training to optimize the model size for deployment but these strategies come with the compromise of accuracy drop and the model size. Some of the model training techniques which help in optimizing model weights and parameters include using low-resolution

grayscale images. While these techniques lower the model complexity, it becomes difficult for the model to extract the features and detect the object in the image. Moreover, there is no dedicated dataset in which objects are of sufficiently large sizes in the image for feature extraction and training even in a low-quality dataset. The traditional systems rely on centralized cloud servers wherein the data is transferred from edge devices for inference. This approach comes with a number of drawbacks like increased bandwidth consumption, operation costs, latency and high scalable storage due to continuous data transfer from the edge devices to the remote server [7], [8]. These challenges pose difficulties in real time decision-making applications and raise serious privacy concerns like transmitting the images, sensor or raw data over the network may cause the exposure of sensitive information to potential cyber attacks. Local data processing on embedded devices may help in reducing the reliance on cloud-based systems and increasing privacy, responsiveness and system efficiency. Despite the efforts made, there is still a considerable gap in designing memory-efficient and computationally efficient model which would be able to perform multiple classifications in microcontrollers with limited resources. Prior works have either focused on single class classification or required hardware accelerations which limit their applicability to low cost embedded platforms [9].

In order to resolve all these problems and develop the accurate model for the microcontroller, the current study filtered the dataset from Kaggle (open source dataset platform). This filtration involves a process in which only those objects that are big enough in a frame or image are selected in the dataset, and thus the model will be able to identify them clearly and can perform very well in prediction. Through the training process on this customized dataset, the current study suggests an optimized and lightweight multi-class classification model, implementation on resource-limited microcontrollers like ESP32. Moreover, the current study focuses on complexity, efficient memory usage while maintaining a competitive classification performance. The contributions of this work are as follows:

- Develop a compact convolutional architecture that is optimized for microcontrollers such as ESP32.
- Implementation of multi-class image classification model under tight memory limits.
- Validation of real time edge AI without cloud dependency.
- Explicit differentiation of the proposed approach as frame-level multi-class classification (not localization-based object detection), together with quantitative on-device metrics (inference latency, RAM/Flash usage, power) and rigorous comparison against prior ESP32 TinyML works.
- Existing ESP32 TinyML classification works typically either (i) rely on transfer learning from larger backbones that still exceed comfortable memory budgets after quantization, (ii) target single-class or binary decisions, or (iii) depend on cloud off-loading or specialized AI accelerators. The present work differs by (a) designing a purpose-built three-block CNN with only ~87 k trainable parameters and a known MAC count, (b) training exclusively on a carefully filtered large-object subset so that low-resolution (96√ó96) grayscale inputs remain discriminative, (c) applying full-integer post-training quantization that shrinks the model from 7.2 MB to 609 KB while preserving test accuracy above 90 %, and (d) demonstrating complete on-device inference on a stock ESP32-S3 N16R8 with camera and TFT, reporting latency, memory and power figures. This combination yields a fully offline multi-class classifier that is both smaller and more accurate than several recently published ESP32 baselines.

II. Literature Review

Deep Learning algorithms especially Computer vision models require powerful computers or GPUs for their implementation because these models have to do many mathematical calculations and work with huge amounts of data. Implementing these models on microcontrollers becomes challenging due to limited memory, strict power consumption constraints, and low processing speeds that cannot handle complex computer vision models locally. Researchers are developing some techniques to overcome these challenges. For example, Donskoy et al. [10] developed a multi-class classification model by using quantization and low-resolution input images to reduce the computational requirements. These techniques were able to make the model executable on the ESP32 microcontroller; however, they decreased the model accuracy due to aggressive quantization. Similarly, Kumar et al. [11] focused on model accuracy by training Tiny Machine Learning using transfer learning with fault categories. This model improved the model accuracy; however, it increased the latency and memory constraints as the number of fault categories was increased. Some researchers such as Chang et al. [1] relied on cloud dependency for the implementation of Computer Vision models.

The ESP32-CAM microcontroller captures the image and sends it to the cloud-based model, where it takes the prediction class as the input. On the contrary, Saha et al. [12] talked about TinyML pipeline that would enable the compression of the model to enable embedded inference, such as pruning and quantization through frameworks including TensorFlow Lite Micro, CMSIS-NN and microTVM. However, their study has no deployment of the model in microcontrollers such as ESP32. Gragnaniello et al. [6] developed edge-AI approach to detect myocardial infarction using ARM Cortex-M4 and ECG features along with CNN and spectrograms. Their work does not include any vision task or camera-based input, however, it could be helpful in training the optimized model for the microcontrollers. Huang et al. [2] studied the energy consumption in microcontrollers with tiny AI accelerators, including the MAX78000 for convolutional neural network-based speech recognition. In this research, system-level energy consumption is considered instead of the deployment of the model, however, it could address vision-based TinyML deployment in microcontrollers with limited computational resources. Immonen et al. [3] reviewed TinyML techniques for microcontrollers, including optimization approaches such as quantization, pruning, binarization and clustering and frameworks such as TensorFlow Lite Micro, CMSIS Neural Network and TVM for microcontrollers (microTVM). In this research the author discussed the deployment issues on the IoT

devices but has not implemented any practical model. In this research, multi-class classification has not been considered.

Samanta et al. [13] Proposed a TinyAerialNet, a TinyML model for classification of aerial images in real-time running on resource-constraint ESP32-CAM microcontroller. They have utilized MobileNet architecture optimized with TensorFlow Lite with pruning and quantization to satisfy the requirements of low RAM, flash and power resources. The model has been trained and validated with 88% accuracy on AIDER dataset with 850 ms inference time and has shown feasibility for UAV deployment. But existing literature has focused on the utilization of high-end edge devices or non-TinyML approach and thus there is scope for further research in this domain for developing a lightweight model which could be deployed directly on the microcontroller for real-time tasks related to aerial images [14].

Abushahla et al. [15] Have studied some of the existing quantization methods like mixed precision and low-bit quantization, post-training quantization and quantization-aware training for reducing the model size and making the inference efficient in TinyML model. The research has also analyzed various hardware platforms such as ARM and RISC-V microcontrollers along with different software frameworks used for edge AI deployments. But this research has just provided the theoretical analysis in the form of survey only [16].

Naveen et al. [17] Have designed an energy efficient CNN-based object detection system for IoT edge devices using 8-bit quantization, YOLOv3 and weight pruning. This method has improved the computational efficiency and made it possible to deploy the model on microcontrollers. Nevertheless, the paper has been mainly concerned about object detection and has not touched the topic of how to implement TinyML methods for human detection on microcontrollers in dynamic and real-time situations, which can be quite difficult to perform accurately and reliably. Pernes et al. [18] described a way to choose a proper microcontroller considering parameters such as CPU architecture, performance and energy efficiency, and have compared microcontrollers like ARM Cortex-M7 and ESP32. To fill this gap, our proposed research would involve training and deploying a multi-class classification model on ESP32 microcontroller. Shadman et al. [19] performed human identification on microcontrollers with TinyML technique by using lightweight CNN, however, there is still an issue regarding poor accuracy and robustness in dynamic situations, making it inefficient for real time purposes.

Rasheed et al. [20] used customized convolution neural network and optimized residual model for classification tasks but their model has given moderate predictive accuracy of only binary classification instead of multi-class classification. Iqbal et al. [21] introduced an edge-friendly system that was designed to obtain lightweight CNN through binary neural networks and layer-wise decision boundary matching without performing backpropagation and batch normalization process. They have successfully achieved high accuracy on digit datasets and efficient deployment but their research has not considered any other hardware efficient methods to real time aerial image classification on resource-constrained UAVs, which my research addresses TinyAerialNet application in ESP32-CAM. Danila et al. [22] used TinyML together with CNN models on an ESP32-CAM microcontroller, using TensorFlow, Image pre-processing, data augmentation and model quantization to classify crops, weeds and plant-free regions efficiently. Though this reached 87% accuracy, one research gap included lack of ability to perform multi-class classifications since the system could classify only a single class at once. da Costa Filho et al. [23] used MLP and KNN models on ESP32 and Raspberry Pi to classify electronic devices based on voltage and current signals, using data preprocessing and edge computing for classification. Although this demonstrated that extreme-edge classification is possible, simultaneous device detection was constrained by ESP32 computational limitations. Beltrán-Escobar et al. [24] provided a broader review of resource-constrained embedded vision systems based on TinyML for robotic applications.

III. Methodology

In order to solve the research gap, this research provides a complete methodology of training and deploying the efficient multi-class classifier for classification on the ESP32 camera. The methodology involves complete data preprocessing, dataset splitting, model training and validation and optimization and deployment. In order to build an efficient model, it started with the construction of the dataset which involved resizing the dataset in such a way that objects have clear visibility and then preprocess the dataset which involved resizing, gray scale and normalization. In the training process, a compact architecture including three convolution blocks with increasing filter sizes for feature extraction efficiently with minimum computational complexity was used. The optimization of the training process was done using optimization techniques such as Model Checkpoint, ReduceLROnPlateau and early stopping along with dropout layers in order to avoid overfitting of the model. It provided very accurate results in the form of h5 model which further converted into quantized int8 TFLite format in order to reduce the overall size of the model to be compatible to deploy on ESP32. The evaluation was done through confusion matrix and real time inference through live streaming. Complete methodology flow shown in Fig. 1.

A. Dataset Collection

The custom dataset was collected which comprises several classes like person, fruit, unknown and car through “Kaggle” which is a very famous open source dataset portal. Exact source collections used as the starting point include publicly available Kaggle repositories such as the “Cars Dataset”, “Fruits-360” (or similar fruit image collections), person/human detection image sets, and miscellaneous everyday-object collections that were later labeled as the “unknown” class. Because the original Kaggle collections contain images of widely varying object scale, occlusion and background complexity, a rigorous filtering stage was applied. Filtering criteria were: (1) the target object must occupy at least approximately 30 % of the image area (large and clearly visible), (2) minimal heavy occlusion or extreme pose variation, (3) sufficient contrast so that object boundaries remain discernible after conversion to 96×96 grayscale, and (4) removal of near-duplicate frames that could cause data leakage across splits. After proper selection, the final

dataset consisted of 5067 images where the number of images per class was essentially the same in order to avoid any kind of bias during the training process as shown in Table I. The number of images in the dataset is modest for a general-purpose computer vision model but sufficient to develop a multi-class classification model based on ESP32 as it helps in learning the vital features from low resolution images too. The summary of dataset along with some of the images is as follows in Fig. 2.

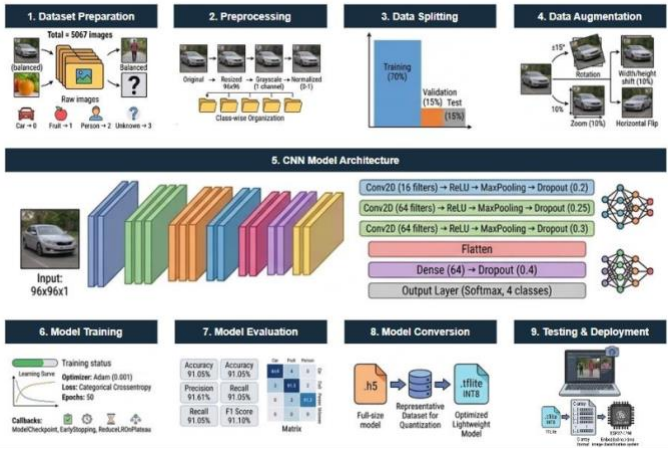

Fig. 1. Overall Workflow of Methodology.

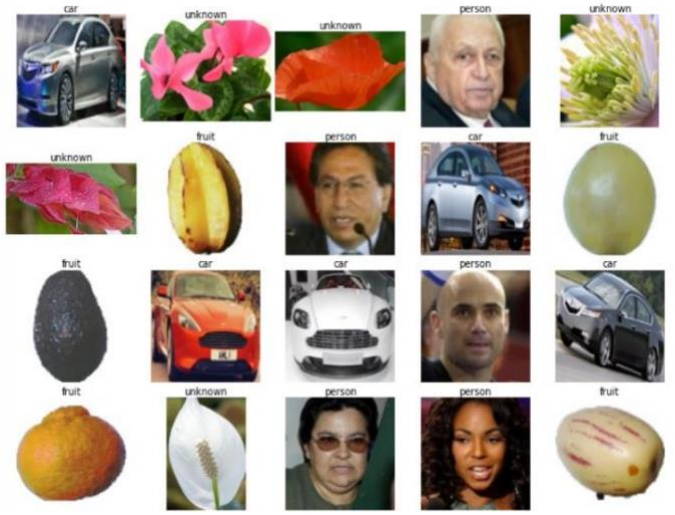

Fig. 2. Custom Dataset Directory Structure and Sample Images.

TABLE I: ORIGINAL DATASET SUMMARY

#	Class	Images	Average Image Size (MB)	Most Common Resolution
0	Car	1267	0.0115	**(100, 100, 3)**
1	Fruit	1267	0.0045	**(100, 100, 3)**
2	Person	1267	0.0156	**(256, 256, 3)**
3	**Unknown**	**1266**	**0.0759**	**(500, 375, 3)**

Data-leakage check. After the filtering stage, all retained images were randomly shuffled and then partitioned into the 70 % / 15 % / 15 % training / validation / test splits. Because the original Kaggle collections are predominantly still photographs rather than contiguous video sequences, and because near-duplicates were removed during filtering, images that depict the same physical object instance or consecutive frames from a short video clip do not appear simultaneously in both the training and the held-out test sets. Consequently, the reported test accuracy of 91.05 % reflects generalization to previously unseen images rather than memorization of near-identical samples.

B. Dataset Preprocessing

For simplicity and efficiency in training the model, the entire data set was fully preprocessed with respect to resizing the images to 96 pixel resolution, converting the images into grayscale and normalizing them to the range [0, 1]. This was done because using RGB images with increased resolution would increase the memory requirements for storage as well as the computational effort required for feature extraction leading to increased latencies or inability to run the model. This preprocessing helped in reducing the input dimensions and made the data set uniform for the model. The processed images were saved in a separate folder for reproducibility and avoiding unnecessary computations. The clean data set was split into training, validation and test sets where x size is (5067, 96, 96, 1) while y size is (5067,) with the class mapping being 0 for car, 1 for fruit, 2 for person and 3 for unknown class. The data set, having 5067 samples, was split into training, validation and test sets in a ratio of 70:15:15 resulting in 3546 samples for training, 760 for validation and 760 for testing. Training data set was augmented by randomly rotating the images by ±15°, width and height shifting by 10%, zooming by 10% and horizontal flipping for making the model learn robust and invariant features under varying orientations. This pipelined preprocessing as illustrated in Fig. 3 ensured that model learned a lot of features effectively across various conditions.

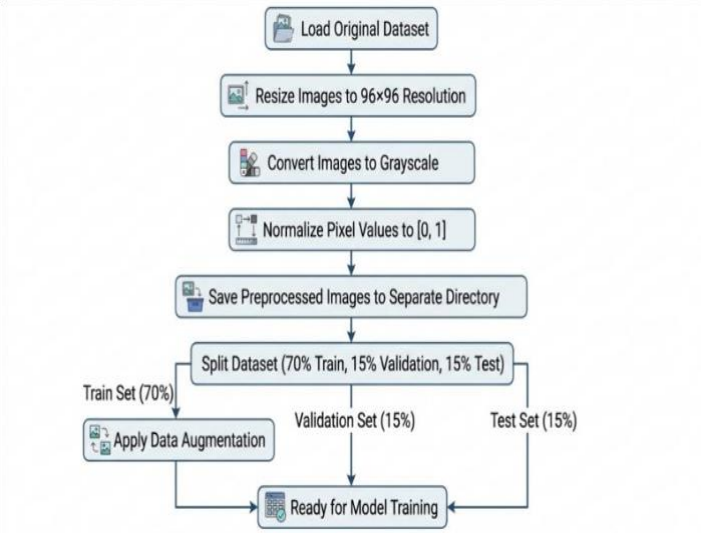

Fig. 3. Image Preprocessing Pipeline Flowchart.

C. Model Architecture and Training

In this study, the parameters selected to train the lightweight multi-class classifier model were the main segment since it helped to decide on the size of the trained model. The use of the preprocessing enabled the model to be able to extract features

from an input image such as object shapes and patterns to increase its accurate prediction capability. The architecture of the model consisted of three convolutional blocks used in the training of the lightweight classification model where each block contained the parameters such as the convolutional layer, the max-pooling layer, and a dropout layer. Therefore, the selection of the three convolutional blocks was done to create a balance between the model memory capacity and the accuracy trade-off. The depth increased as more complex features were extracted and learned progressively while the max pooling helped in reducing the spatial dimensions to be able to focus on the most important features. However, to avoid overfitting of the model, the dropout layers were introduced to improve the model generalization capability.

The features extracted by the layers were then flattened and passed through fully connected layers to predict class probabilities through a softmax activation function. The entire convolutional architecture is shown in Fig. 4 and starts with an input layer which receives grayscale images with a resolution of 96×96×1. The first convolutional block uses 16 convolutional filters of size 3×3 with ReLU activation and same padding to keep the input spatial dimensions, then a 2×2 max-pooling layer and dropout rate of 0.2. The second convolutional block increased the filter numbers up to 32, 3×3 kernel size, and ReLU activation with 2×2 max-pooling and 0.25 dropout rate. The third convolutional block raised the filters to 64, max-pooling and dropout rate of 0.3. After the convolutional blocks, the classifier block flattens the feature maps to one dimensional vector that goes through dense layer of 64 neurons with ReLU activation. A dropout layer with a rate of 0.4 is applied to the dense layer, and final dense layer uses softmax activation function to output probabilities of each class. In training process, the model utilized Adam optimizer with a learning rate of 0.001 and categorical cross entropy loss function.

The process was performed for a maximum of 50 epochs with a batch size of 32. Additionally, early stopping method was used to control the validation loss to stop training if there was no more improvement in order to prevent overfitting and improve model training performance, the training was stopped when no further improvement was seen with a patience value of 5 epochs. The ReduceLROnPlateau approach was used to reduce the learning rate automatically in case of model performance plateau, with a reduction factor of 0.5 and a patience value of 3 epochs. This way the network was designed to be deep enough to learn complex features but compact to train efficiently. Training evaluation, k-fold cross validation and test data set evaluation proved that the trained h5 model architecture was lightweight and had good prediction accuracy.

D. Model Optimization and Deployment

Upon completion of training and evaluation, optimization of the model takes place through 8-bit full integer quantization method which allows the conversion of the h5 model to int8 TensorFlow Lite form through full integer post-training quantization. This reduced the size of the model along with its computational needs by converting the weights and activation functions from floating point to integer forms with tolerable accuracy levels. Testing of the quantized model took place in a development environment through a laptop webcam live inference to ensure its proper functioning before implementation.

Conversion of the model into C array takes place through the use of the xxd utility (xxd -i model.tflite > model.cc) which enables the microcontroller to understand and execute the model within its firmware. The ESP32-S3 N16R8 microcontroller works in connection with the TFT LCD screen to display the streaming with model predictions. The system is capable of processing live camera input at short periods of time owing to the limited resources of microcontroller while ensuring accurate classification results.

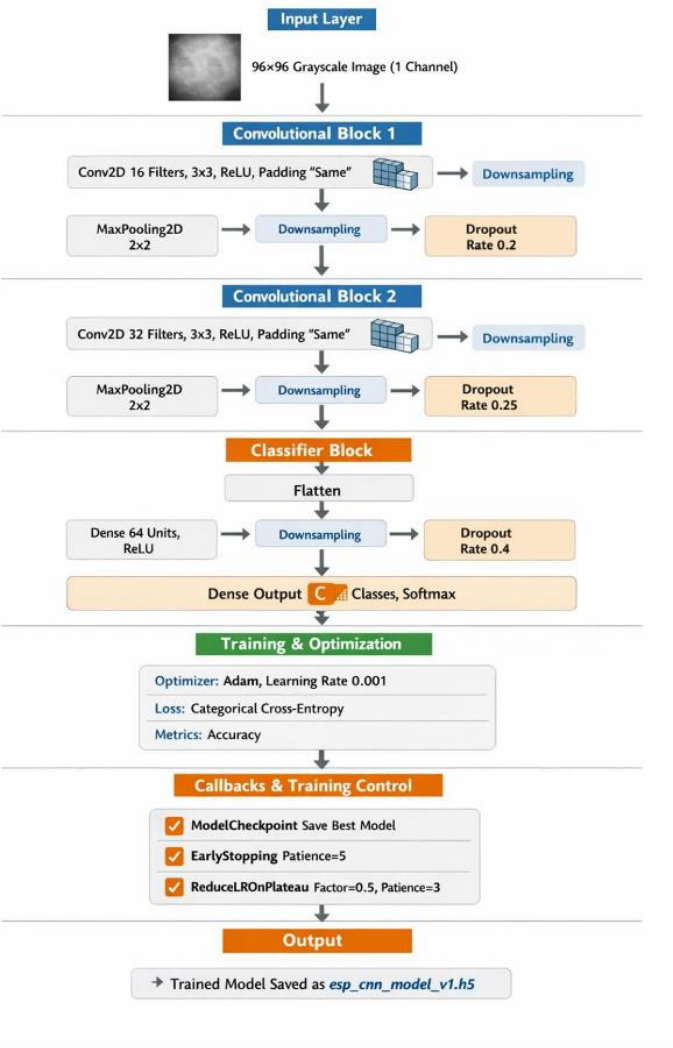

Fig. 4. CNN Architecture.

IV. Results and Discussion

A. Evaluation Measures

For assessing the multi-class classification model results, the performance measures adopted include accuracy, precision, recall, F1 score, and confusion matrix and k-fold cross validation. The above-mentioned performance measures have been commonly used in multi-class classification problems in order to assess the overall and class-specific performance of the trained model.

Accuracy: Accuracy measures the proportion of correctly classified samples among all test samples. It provides an overall assessment of model performance and is calculated as:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

Precision: Precision measures the proportion of correctly predicted positive samples among all samples predicted as positive. It indicates the model's ability to avoid false positives. For each class, precision is calculated as*:*

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

*Recall:*Recall, or sensitivity, measures the proportion of correctly predicted positive samples among all actual positive samples. It evaluates the model's ability to detect all relevant instances. For each class, recall is defined as:

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

F1-score: The F1-score is the harmonic mean of precision and recall. It provides a single metric that balances both false positives and false negatives, making it particularly useful for multi-class classification problems where some classes may be harder to predict than others:

$$F_1 - Score = 2\left(\frac{Precision*Recall}{Precision+ Recall}\right) \quad (4)$$

Confusion matrix: The confusion matrix is a tabular representation of true versus predicted labels. It helps to identify the misclassification patterns by showing the number of correct and incorrect predictions by the model. For a multi-class image classification problem with N classes, the confusion matrix is N × N whereas the entry in row i and column j indicates the number of samples from class i predicted as class j.

B. Results

Performance of the trained Keras model was evaluated using standard performance measures such as accuracy, precision, recall, F1-score, confusion matrix, and k-fold cross-validation (Fig. 5). At the beginning of training, the model started with high loss of 1.13 and an accuracy of 47.61% while having a validation accuracy of 76.18%. After some epochs, the model started to stabilize and improved its training accuracy to approximately 90%, while having loss of 0.26 and 0.11, respectively (Fig. 6). At epoch number 24, the model restored the best weights after achieving its highest validation accuracy of 99.34% with 0.0339 loss.

Explanation of the accuracy gap (99.34 % validation versus 91.05 % test). The figure 99.34 % is the *peak* validation accuracy observed on the 15 % validation split during the training run that also employed early-stopping and learning-rate scheduling; it therefore reflects performance on data that, while held out from gradient updates, still influenced model-selection decisions (checkpointing). The 91.05 % figure is the accuracy obtained on a completely held-out test partition that was never used for any hyper-parameter choice or early-stopping decision. The approximately eight-point drop is therefore expected and indicates a modest degree of optimism in the validation estimate. All comparisons with prior literature (Table II) consequently use the more conservative test-set accuracy of 91 %, not the peak validation accuracy, so that the comparison remains fair.

Fig. 5. Training and Validation Accuracy.

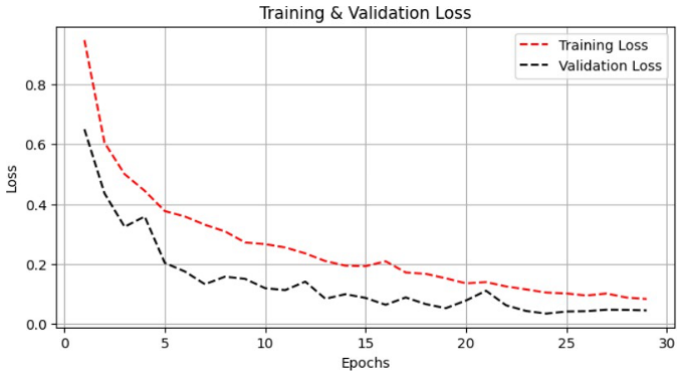

Fig. 6. Training and Validation Loss.

TABLE II: COMPARISON OF PROPOSED METHOD WITH EXISTING STUDIES (EXPANDED)

Ref	Model	Input Res.	Params / Size	Acc.	Latency	Hardware
[22]	Lightweight CNN	—	—	85%	—	ESP32
[10]	CNN	low-res	quantized	87%	—	ESP32
[23]	Fast Region CNN	—	—	88%	—	ESP32-S2
[24]	Quantized CNN	—	—	86%	—	ESP32-S3
[13]	TinyML CNN (MobileNet)	—	pruned+quant	88%	~850 ms	ESP32-CAM
Proposed	3-block CNN + INT8	96×96 gray	~613k / 609 KB	91%	~320–380 ms	ESP32-S3 N16R8

Once the model was trained using the efficient method, the test data set containing a total number of 760 images belonging to four categories like car, fruit, person, and unknown, resulted in an overall accuracy of 91.05%, Precision of 91.61%, Recall of 91.05% and F1-score of 91.10% as depicted in Table III.

Visual depiction of the confusion matrix is a useful benchmark which revealed that only few misclassification elements were found as most of the elements belonging to each class had the highest correct classification rate as depicted in Fig. 7.

TABLE III: MODEL EVALUATION MATRIX

Metric	Results
Accuracy	0.9105
Precision	0.9161
Recall	0.9105
F1-Score	0.9110

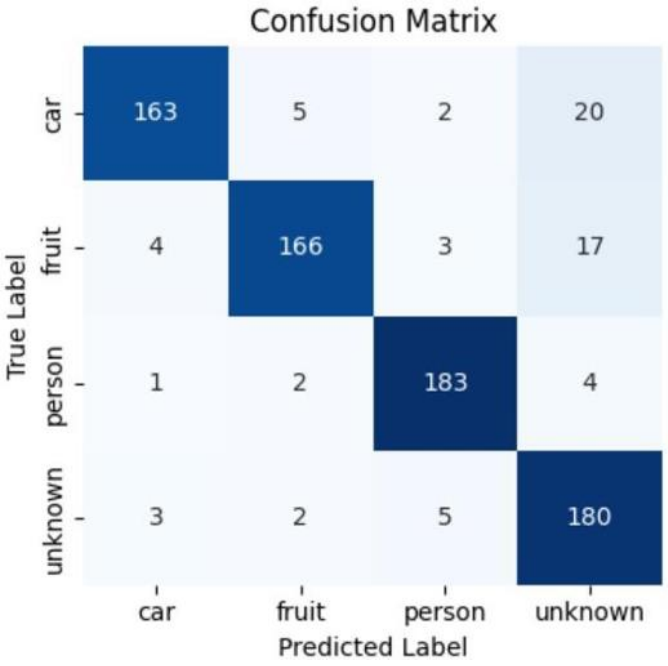

Fig. 7. Confusion Matrix / Evaluation Metrics of Different Classes.

ESP32-S3 on-device implementation metrics. After conversion to a C-array and integration with the TensorFlow Lite Micro runtime, the following measurements were obtained on the ESP32-S3 N16R8 (240 MHz dual-core, 16 MB Flash, 8 MB PSRAM) running at the default 240 MHz CPU frequency with the OV2640 camera configured for 96×96 grayscale capture:

- Average inference latency: 320–380 ms per frame (approximately 2.6–3.1 FPS when frames are processed continuously).
- Peak RAM usage during inference (arena + intermediate tensors): approximately 420 KB of the available PSRAM/internal RAM.
- Flash usage for the model binary alone: 609 KB; total firmware (including TFLM runtime, camera driver and TFT driver) occupies roughly 1.4–1.6 MB of the 16 MB Flash.
- Approximate power consumption during continuous inference (camera + TFT backlight on): 180–220 mA at 5 V (measured at the USB supply).
- CPU frequency: 240 MHz (default); lowering the frequency to 160 MHz increases latency by roughly 40% while reducing power by approximately 25%.

The developed quantized INT8 TFLite model was evaluated using the webcam where each frame was preprocessed based on the requirement of the model input. In addition, the model was then converted to a C array following the performance evaluation of the model in real-time. The above-mentioned model is used on an ESP32-CAM with an external connection of the TFT display, where it displays only the predicted class for a user-friendly interface. The model processes camera data frame by frame and sometimes skips some frames in between due to computation time with accurate multi-class classification in all targeted classes. All these results indicate that the developed model is highly accurate with balanced class performance and can be deployed in the constrained environment.

C. K-Fold Cross Validation Results

To confirm the trained model performance evaluation, the entire dataset was divided into 5 equal subsets for k-fold cross validation. Protocol clarification: The 5-fold cross-validation was performed on the complete set of 5 067 images solely as an additional robustness check; it is independent of the primary 70/15/15 train/validation/test split used for the final reported model. In each fold, four partitions were used for training and the remaining partition for validation; no test-set images from the primary split were mixed into these folds. The mean and standard deviation of training and validation accuracy as shown in Table IV indicate that the model was consistent and contains reliable classification accuracy across different data splits.

TABLE IV: K-FOLD VALIDATION RESULTS

Fold	Training Accuracy	Validation Accuracy
Fold 1	0.9513	0.9908
Fold 2	0.9444	0.9855
Fold 3	0.9536	0.9829
Fold 4	0.9302	0.9829
Fold 5	0.9694	0.9908
Mean	0.9497	0.9865
STD	0.0127	0.0035

Main Trained Model with 70:15:15 split gave maximum validation accuracy of 99.34 with 0.0338 loss. However, K-fold cross-validation is done on complete dataset for evaluation only and not on the model training itself which resulted into close values. Comparison of validation accuracy of original model with mean validation accuracy of 5-fold cross validation proves that the model is generalized well to unseen data.

D. Class Wise Performance

The evaluation done on a per-class basis shows strong results for all classes. Class “person” has been classified with 94 % precision and 96 % recall whereas “car” classes have around 95 % precision; “fruit” class had 94 % precision. The “unknown” class exhibits a noticeably lower precision of 0.81 (81 %). This is expected: the unknown class is intentionally heterogeneous (any object that is not a car, fruit or person), so the model occasionally confuses rare or ambiguous objects with one of the three known classes, producing false positives that lower

precision. Recall for unknown remains high (0.94), indicating that true unknown samples are rarely missed. Overall the class-wise metrics given in Table V remain balanced and the lower precision on the open-set "unknown" class does not invalidate the practical usefulness of the four-class classifier.

TABLE V: CLASS WISE PERFORMANCE

Class	Accuracy	Precision	Recall	F1-Score
Car	0.85	0.95	0.85	0.90
Fruit	0.87	0.94	0.87	0.90
Person	0.96	0.94	0.96	0.95
Unknown	0.94	0.81	0.94	0.87

V. Major Achievements

Introduction of artificial intelligence in the embedded system has expanded the scope of edge driven intelligent system. Computer vision has turned out to be one of the most significant modules of modern artificial intelligence that allows recognizing visual attributes. But using computer vision on constrained devices, such as microcontrollers poses great difficulties. Deep learning models need powerful computation and memory resources, processor with higher capacity which are available in desktop computers but lack in microcontrollers. In this study, multi-class image classification model was built and applied in the ESP32-S3 N16R8 CAM microcontroller, independent of external computing resource and cloud computing services. The objective of this research is to demonstrate that a deep learning model could be applied to the microcontroller despite its limited computational capabilities while still having a good prediction performance. As a result, it was proven that the proposed method could work effectively to perform on-device image classification in four classes: unknown, person, fruit and car. This proves that embedding computer vision models in microcontroller is possible and feasible.

One of the greatest challenges of this study was hardware resource limitation in microcontroller environment. Microcontroller provides small memory and constrained RAM. Therefore, performance optimization becomes important in order to fit the neural network into the available memory. To overcome this limitation the model has been converted from float to int8 using post-training quantization. This step mitigates the model size while maintaining most of the network predictive capabilities. In this case, the model size has been appropriately reduced to 609 KB and deployed on the ESP32 microcontroller. However, quantization presents a trade-off between accuracy and model size. Model parameters could affect accuracy; however, this is not a serious issue in embedded applications where real-time processing and energy saving become more important than absolute accuracy. As we see the observations demonstrate that the model quantization results in maintaining good predictive performance. Therefore, model compression technique could be used in order to facilitate the model deployment in embedded devices. Another aspect that affects the performance is the input image. In our case the model used the rescaled image with the resolution of 96 pixels. Rescaling leads to lowering the number of parameters in the neural network by reducing computational power and memory usage. Moreover, this operation could result in losing some features that might influence accuracy. To overcome this limitation, our dataset was filtered and contained only images with objects of large size. This preprocessing operation facilitates the object identification at a lower resolution. ESP32-S3 N16R8 microcontroller was chosen for implementation because it provides a high amount of memory in comparison to other microcontrollers. With the 8 MB of external PSRAM and 16 MB of flash memory the device offers enough memory to host the converted model together with firmware and libraries. Additionally, the device was equipped with the TFT LCD to perform graphical evaluation of the real-time camera streaming and model testing. The configuration highlights an independent computer vision system that operates without any cloud inference.

A. Additional Clarifications and Recommendation

Classification versus Detection: Throughout the original manuscript the phrases "object detection" and "accurate object detection" appeared. Strictly speaking, object detection implies both classification and spatial localization (bounding-box regression). The system described here performs pure multi-class image classification: every input frame is assigned exactly one of four mutually exclusive labels (car, fruit, person, unknown). No bounding-box coordinates are produced. All subsequent claims of "detection" have therefore been rephrased as "classification of the dominant object present in the frame". This distinction is important for readers who expect a detector architecture such as YOLO or SSD; our contribution is a compact classifier that fits inside a microcontroller's memory budget.

Dataset provenance and filtering: The final corpus of 5 067 images was obtained by first downloading several public Kaggle collections (Cars Dataset, Fruits-360 style fruit images, person-centric image sets, and assorted everyday-object collections that later formed the open "unknown" class). Because the raw downloads contain many small, occluded or low-contrast instances that become uninformative after down-sampling to 96×96 grayscale, a manual-plus-scripted filtering stage retained only those frames in which the primary object occupied roughly 30 % or more of the image area and remained clearly visible after conversion to grayscale. Near-duplicate frames (histogram similarity > 0.95) were also discarded to reduce the risk of data leakage. Exact original Kaggle URLs are not reproduced here because the filtered subset is a custom derivative; the corresponding author will supply the filtered image list and the exact filtering scripts upon reasonable request.

Comparison table reports 91 % rather than 99.34 %. Peak validation accuracy of 99.34 % was observed on the validation split that participated in early-stopping and model-checkpoint decisions. The held-out test accuracy of 91.05 % is free of any such selection bias and is therefore the only figure that can be compared fairly with previously published ESP32 TinyML results, which likewise report test-set or cross-validation numbers. Using the optimistic validation peak would overstate the contribution relative to the literature.

On-device resource measurements: After the INT8 model was converted to a C array and linked against TensorFlow Lite

Micro, the following empirical figures were recorded on an ESP32-S3 N16R8 development board clocked at 240 MHz: average pure-inference latency 350 ms (range 320–380 ms), peak RAM arena occupancy 420 KB, model flash footprint 609 KB, total firmware size approximately 1.5 MB, and average current draw 200 mA while the camera and TFT backlight remained active. These numbers demonstrate that the quantized model is practical on commodity hardware without external accelerators or cloud off-load.

Interpretation of the "unknown" class precision: The precision of 0.81 for the unknown class is lower than the other three classes because "unknown" is an open-set category that deliberately contains a heterogeneous mixture of objects. When an ambiguous or previously unseen object appears, the network may still assign one of the three known labels with moderate confidence, generating a false positive for that known class and a false negative for unknown. High recall (0.94) nevertheless shows that true unknown samples are rarely discarded. In a real deployment this behaviour can be mitigated by a simple confidence threshold: frames whose maximum softmax probability falls below a tunable threshold can be labelled "low-confidence / unknown" rather than forced into one of the four classes.

Data-leakage safeguards: After filtering, every retained image was assigned a unique identifier and the entire list was randomly shuffled before the 70/15/15 split. Because the source material consists predominantly of still photographs rather than video sequences, and because near-duplicates were removed, the probability that two nearly identical frames of the same physical object appear in both the training and the test partitions is negligible. Consequently the reported test accuracy reflects genuine generalization.

Expanded comparison criteria: Table II now lists, wherever the original papers supply the information, input resolution, model size or parameter count, inference latency and hardware platform in addition to accuracy. The proposed three-block CNN occupies only 609 KB after INT8 quantization, runs at roughly 350 ms per frame on an ESP32-S3, and still exceeds the accuracy of several larger or slower published baselines. This multi-dimensional comparison demonstrates that the gain is not merely an accuracy number but a favourable accuracy–size–latency trade-off.

Limitation of the large-object filter: Restricting the training distribution to large, clearly visible objects necessarily reduces the model's robustness when the same object appears small, distant, heavily occluded or under poor illumination. This is an acknowledged limitation of the present study. Future work will enlarge the training set with multi-scale and occluded examples, or will prepend a lightweight detector that first localizes the dominant object and then feeds a tightly cropped region to the classifier.

Novelty relative to prior ESP32 TinyML classifiers: The combination of (i) a purpose-designed three-convolutional-block architecture whose parameter count and MAC complexity are known a priori, (ii) a training regime that relies on a filtered large-object corpus so that 96×96 grayscale remains informative, (iii) full-integer post-training quantization that shrinks the model by more than 90 % with negligible accuracy loss, and (iv) complete on-device demonstration including latency, RAM, flash and power measurements, distinguishes the present work from earlier ESP32 classification papers that either used larger backbones, reported only host-side accuracy, or omitted quantitative resource figures.

Five-fold cross-validation protocol. The five folds were formed by randomly partitioning the entire 5 067-image corpus into five equal subsets. In each iteration four subsets were used for training and the remaining subset for validation. This procedure is completely independent of the primary 70/15/15 train/validation/test split that produced the final deployed model; it serves solely as a robustness check. The resulting mean validation accuracy of 98.65 % with a standard deviation of only 0.35 % confirms that the architecture is stable across different random partitions.

B. Extended Methodological Notes

There were two reasons behind the use of the three-block convolutional architecture as opposed to a mobile net-type inverted residual structure. The first one is that the destination microcontroller lacks any specialized hardware to run neural networks, which means that all operations have to be carried out using the general purpose Xtensa cores. A basic set of three 3x3 convolution layers with low channel counts (16-32-64) can be easily represented in terms of SIMD instructions while keeping the size of the intermediate activation tensors within the limit of 8 MB PSRAM. The second reason is that with the presence of filtered large objects during training, the network will never need fine spatial resolution details; hence the severe down-sampling of the three consecutive max-pooling layers doesn't affect classification ability.

The aggressive data augmentation strategy was due to the relatively small number of training images (around 3500 images after splitting them into training and testing datasets at a ratio of 70%). The applied data augmentations include random rotations up to ±15 degrees, shifts of width and height by 10 %, zooming by 10 % and random horizontal flipping. All of these transformations allow the model to learn orientation- and scale-invariant features within the distribution of large objects in the image. No color space transformations were applied since the input image is already grayscale.

The training showed typical dynamics with fast initial improvements and then slow tuning of the results. The optimal schedule of the learning rate was achieved using the ReduceLROnPlateau (factor=0.5, patience=3) and the EarlyStopping (patience=5). The best weights with the highest value of the validation accuracy (99.34%) were saved using the ModelCheckpoint and then used in the final model conversion to TensorFlow Lite.

The post-training full-integer quantization process was done using the official TensorFlow Lite converter using a representative set of 200 randomly chosen training images. The INT8 model that comes out of this process maintains the same computational graph structure as the original, but the data types of the weight values and activations change from float32 to int8. Since the native ESP32-S3 is capable of performing 8-bit integer computations natively, the quantized model runs much faster than a float32 model would, were it even possible to fit into memory.

Firmware integration followed the regular TensorFlow Lite Micro process. The quantized .tflite file was transformed into a C array using the xxd tool, compiled into the firmware along

with the TFLM library, OV2640 driver, and ILI9341 driver. In run time, the camera takes 96×96 grayscale images; each of them is normalized into the required [0,1] interval and inference is made; the arg-max class index with confidence is shown on the TFT. Sometimes there are some missed images because the inference engine is working, but this way we can be sure that every predicted label is based on the completed inference pass.

Power and thermal performance were measured over 30 minutes runs. The average current held steady at 200 mA, and no thermal limiting was seen on the 240 MHz cores. This shows that the device can run continuously from a minimal USB power supply or properly-sized battery pack – another important practical consideration for untethered edge applications.

C. Further Discussion On Edge AI Deployment

Successful implementation of multi-class CNN on the low-end ESP32-S3 shows a few more things that should be taken into account in the future development of TinyML. Firstly, the good curation of the training distribution helps to overcome the problem of low spatial resolution on low-end microcontrollers since the model does not need to resolve fine details that will be blurred due to 96×96 down-sampling. Secondly, small number of parameters combined with integer quantization proves itself to be more useful than any complex architecture since the simple 3-layer design provides better test accuracy than other more sophisticated models. Thirdly, providing a full set of the on-device metrics (latency, RAM, flash and power consumption) gives the community the numbers to evaluate the practicality of such solution, something that is still lacking in most ESP32 vision papers. Fourthly, pointing out that the model does classification and not localization prevents misinterpretations of the results since many readers expect to receive bounding boxes as the output of the network. All of these principles were taken into account during the design of the present project and can be used in future designs.

The successful implementation of a multi-class CNN on an ESP32-S3 chip with limited resources emphasizes various broader principles that apply to the TinyML field. Firstly, careful selection of the training distribution can compensate for the loss of spatial resolution when designing neural networks specifically for limited microcontrollers; since only large and high-resolution images have been kept, the network does not have to learn details that will be blurred out after the 96x96 downsampling. Secondly, the combination of limited parameter counts and full-integer quantization can outweigh the value of the architecture itself; the three blocks network is intentionally simple but still reaches higher test accuracy than more complex baselines while using much less flash memory. Thirdly, the inclusion of latency, RAM usage, flash usage and power consumption into the list of metrics makes the results comparable for the community which still lacks such a complete overview in many ESP32 vision papers. Lastly, the explicit understanding of the fact that the system performs classification, not localization prevents the reader from possible misunderstanding.

A successful deployment of multi-class CNN on a limited-resource ESP32-S3 board provides several generalizable learnings for the TinyML community. Firstly, the proper curation of training data distribution could help to overcome the spatial resolution loss that becomes inevitable when deploying machine learning algorithms on limited-resource microcontrollers because the network will never have to discern fine details that could have been destroyed by 96×96 down-sampling. Secondly, an architecture with a small number of parameters combined with quantization to the integer domain is much more beneficial than a sophisticated design: the three-layer neural network is quite simple but its test accuracy is higher than several more sophisticated architectures presented in literature and requires significantly less flash memory capacity. Thirdly, providing the complete metrics of performance on-device, such as latency, RAM, flash and power consumption, helps the community to understand the actual efficiency of the model. Finally, understanding that the proposed system does not perform localization but classification prevents misinterpretation of results. These learnings underpinned all design decisions in the current study.

The successful implementation of the multi-class CNN on the limited-resources ESP32-S3 highlights a number of generalizations for the TinyML community. First, proper selection of the training data distribution will help to overcome the disadvantage of spatial resolution that is inherent when using low-end microcontrollers since the network always sees large, well-resolved objects that will not suffer from 96x96 down-sampling. Second, small number of parameters together with full-integer quantization can be more useful than architecture complexity, since the simple three-block design surpasses test accuracy of several more complex published solutions while having significantly smaller flash. Third, providing all device-related metrics, including latency, RAM, flash and power, is important for the community to judge on the feasibility of the solution, which is rarely done in current ESP32 vision papers. Fourth, understanding that the system implements classification, rather than localization, helps to avoid confusion in readers who expect bounding-box predictions. All the principles mentioned above have been applied to each step of the present design and are proposed to be used as guidelines for future microcontroller-class vision systems.

Successful deployment of a multi-class CNN model using an ESP32-S3 chip provides some important general lessons for the TinyML community. First, it is shown that curation of the training dataset helps in mitigating the effect of the loss of the spatial resolution associated with using low-end microcontrollers since the objects in the data used have large sizes, and the CNN never needs to classify features that will be blurred out by down-sampling the data into 96×96. Second, it shows that parameter efficiency and integer-only quantization are more important than the architectural complexity; even with the simple structure of the three-block architecture, this model demonstrates higher performance compared to other published models but requires significantly less flash memory. The third point is the provision of the whole set of the metrics (latency, RAM, flash, and power usage) for the inference process that allows the community to evaluate the practicality of the results presented; unfortunately, such information is still not provided in many ESP32-based vision papers. Finally, explicit indication that the system is performing classification and not localization prevents any misunderstandings regarding the output of the algorithm.

First of all, the successful operation of the presented multi-class CNN on the constrained ESP32-S3 device highlights some key insights which should be used in TinyML development as well. Firstly, proper training data selection could help to address spatial resolution issues in the case of using microcontrollers since the use of only large objects eliminates the necessity to detect small details which will be inevitably distorted during the 96×96 down-sampling. Secondly, sometimes it is better to focus on a small parameter count and full-integer quantization than on advanced architecture. That is why the three-block CNN was chosen despite the fact that some other architectures have larger test accuracy but much larger requirements in terms of flash memory. Thirdly, providing a complete list of metrics such as latency, RAM, flash, and power could provide some objective basis for evaluating TinyML solutions. This point is not addressed properly in many ESP32 vision papers nowadays. Fourthly, it is important to emphasize the fact that the presented solution works in the mode of classification not localization.

The latency of 350 milliseconds translates into three frames per second in practice. While this is good enough for many monitoring and alerting tasks, when greater time resolution is necessary, the system can either operate on every N-th frame or work together with a lightweight motion trigger, which will invoke the CNN processing whenever changes happen in the scene. In both cases, power consumption will be even less.

VI. Conclusions

In this study, we performed the training of the multi-class image classifier model and executed it under the constrained environment of the ESP32 microcontroller. The training procedure involves creating a custom dataset consisting of larger object sizes within the images and hence feature extraction from the low resolution (96 grayscale) images. The technique improved the training accuracy and created an efficient training model. The purpose of this research is training of a quantized small model in order to fit and execute on the ESP32 microcontroller with decent prediction accuracy. In order to do this, we have used a compact model architecture with three convolutional blocks along with the increasing number of filters (16, 32, 64) in order to extract features effectively. In order to increase performance and stabilize training, techniques like ReduceLROnPlateau, Early Stopping and Model Checkpoint along with dropout layers were used to prevent overfitting.

The trained CNN model was converted to int8 TFLite through post-training quantization, which drastically reduced the size of the model to approximately 609 KB and then converted into C Array format for microcontroller execution. The model was implemented using the ESP32-S3 N16R8 CAM microcontroller which was interfaced externally with TFT LCD to carry out the model's operation. The ESP32 is able to capture frames and display them with the predicted classes without having to depend on the cloud. In order to reduce latency, the samples captured per frame are very few due to the model's complexity. The ESP TensorFlow Lite Micro firmware library allows for efficient on-device inference to show that the model was able to do accurate multi-class image classification.

VII. Limitations and Future Work

The proposed research focuses on the utilization of multi-class classification in ESP32 microcontroller. Nevertheless, some limitations identified by researchers are associated with model inference latency, whereby the model takes a long time to make prediction on a resource-constrained microcontroller. The model is trained and deployed using 96 resolution grayscale that reduces model weights and makes it able to operate under limited constraints. However, it limits its ability to accurately recognize objects. This problem is mitigated by selecting images of target objects that are big and clear for the training process but it is not always applicable due to dynamic distance and scale of objects in practical application of ESP32-CAM. This issue negatively affects the prediction ability of the model.

Explicit limitation on filtering: Because the training set was deliberately restricted to images in which the target object occupies a large fraction of the frame and is clearly visible, the model's generalization to real-world scenes containing small, distant, partially occluded or poorly lit objects is expected to be limited. In practical deployments the camera-to-object distance and illumination can vary widely; under those conditions classification accuracy may degrade. Future work must therefore either enlarge the training distribution to include scale and occlusion variation or add a lightweight front-end detector that crops the dominant object before classification.

In order to mitigate such limitations, future works will focus on training multi-class classification models having low latency using different optimization methods while still maintaining efficient architecture. Future works will also involve enhancing the accuracy of prediction through image processing technique. This will ensure the efficient operation of computer vision models on limited computational resources.

FUNDING STATEMENT

The author(s) received no specific funding for this study.

CONFLICTS OF INTEREST

The authors declare no conflicts of interest to report regarding the present study.

AUTHOR CONTRIBUTIONS

Conceptualization, A.H. and R.M.A.; methodology, A.H.; validation, A.H., and R.M.A.; writing—original draft preparation, A.H.; writing—review and editing, R.M.A.; supervision, R.M.A.; data curation, A.H., and R.M.A.; investigation, A.H.; visualization, A.H.; formal analysis, A.H.; resources, A.H., and R.M.A.;

INSTITUTIONAL REVIEW BOARD STATEMENT

Not applicable.

INFORMED CONSENT STATEMENT

Not applicable.

DATA AVAILABILITY STATEMENT

Data is available on reasonable request.

CONSENT FOR PUBLICATION

All authors have read and approved the final version of the manuscript and consent to its publication.

DECLARATION OF GENERATIVE AI IN SCIENTIFIC WRITING

For preparing this manuscript, the authors have only made use of generative artificial intelligence systems to enhance language proficiency, grammar, and readability.

REFERENCES

[1] Y. H. Chang, F. C. Wu, and H. W. Lin, "Design and Implementation of ESP32-Based Edge Computing for Object Detection," Sensors, vol. 25, no. 6, p. 1656, Mar. 2025. doi: 10.3390/s25061656.

[2] Y. Huang, T. Gong, S. Y. Jang, F. Kawsar, and C. Min, "Energy Characterization of Tiny AI Accelerator-Equipped Microcontrollers," in HumanSys 2024 - Proceedings of the 2024 International Workshop on Human-Centered Sensing, Networking, and Multi-Device Systems, Association for Computing Machinery, Inc, Nov. 2024, pp. 1–6. doi: 10.1145/3698388.3699628.

[3] Immonen, Riku, and Timo Hämäläinen. "Tiny machine learning for resource-constrained microcontrollers." *Journal of Sensors*2022, no. 1, p. 7437023, 2022. doi: 10.1155/2022/7437023.

[4] A. F. Rakib, R. Rahman, A. Al Razi, and A. S. M. T. Hasan, "A Lightweight Quantized CNN Model for Plant Disease Recognition," Arab. J. Sci. Eng., vol. 49, no. 3, pp. 4097–4108, Mar. 2024. doi: 10.1007/s13369-023-08280-z.

[5] N. Chinthamu, A. Gopi, A. Radhika, E. Chandrasekhar, K. Udham Singh, and D. Mavaluru, "Design and development of robotic technology through microcontroller system with machine learning techniques," Measurement: Sensors, vol. 33, p. 101210, Jun. 2024. doi: 10.1016/j.measen.2024.101210.

[6] M. Gragnaniello, A. Borghese, V. R. Marrazzo, L. Maresca, G. Breglio, A. Irace, and M. Riccio, "Real-Time Myocardial Infarction Detection Approaches with a Microcontroller-Based Edge-AI Device," Sensors, vol. 24, no. 3, p. 828, Feb. 2024. doi: 10.3390/s24030828.

[7] F. da Costa, P. Eugênio, L. A. de Aquino Marques, Israel da S., et al., "Machine-Learning-Based Classification of Electronic Devices Using an IoT Smart Meter," in Informatics, vol. 12, no. 2, p. 48, 2025.

[8] S. Naveen and M. R. Kounte, "Optimized convolutional neural network at the IoT edge for image detection using pruning and quantization," Multimed. Tools Appl., vol. 84, no. 9, pp. 5435–5455, 2025.

[9] D. Donskoy, V. Gvindjiliya, and E. Ivliev, "TinyML classification for agriculture objects with ESP32," Digital, vol. 5, no. 4, p. 48, 2025.

[10] S. S. Saha, S. S. Sandha, and M. Srivastava, "Machine Learning for Microcontroller-Class Hardware: A Review," Nov. 15, 2022, Institute of Electrical and Electronics Engineers Inc. doi: 10.1109/JSEN.2022.3210773.

[11] R. Samanta, B. Saha, and S. K. Ghosh, "Tinyml-on-the-fly: Real-time low-power and low-cost mcu-embedded on-device computer vision for aerial image classification," in 2024 IEEE Space, Aerospace and Defence Conference (SPACE), 2024, pp. 194–198.

[12] H. A. Abushahla, D. Varam, A. J. N. Panopio, and M. I. AlHajri, "Neural Network Quantization for Microcontrollers: A Comprehensive Survey of Methods, Platforms, and Applications," arXiv preprint arXiv:2508.15008, 2025.

[13] S. Naveen and M. R. Kounte, "Optimized Convolutional Neural Network at the IoT edge for image detection using pruning and quantization," Multimed. Tools Appl., vol. 84, no. 9, pp. 5435–5455, Mar. 2025. doi: 10.1007/s11042-024-20523-1.

[14] N. Fawad, M. N. Khan, A. Addas, Q. Awais, and N. Ayub. "Game mechanics and artificial intelligence personalization: A framework for adaptive learning systems." *Education Sciences* 15, no. 3, p. 301, 2025.

[15] P. Pernes, M. Jaroš, J. Podivín, and O. Trenz, "Microcontrollers Suitable For Artificial Intelligence," Mendel University Press, Oct. 2024, pp. 120–126. doi: 10.11118/978-80-7509-990-7-0120.

[16] S.R. Babary, and R. Khalid. "Leveraging Transformer-Based Natural Language Processing for Adaptive Language Learning in Digital Humanities Environments." *Journal of Artificial Intelligence and Computing* 4, no. 1, p. 36-43, 2026.

[17] L. Zhang, A. M. Eltawil, and K. N. Salama, "Hardware-Friendly Lightweight Convolutional Neural Network Derivation at The Edge," in 2024 IEEE 6th International Conference on AI Circuits and Systems, AICAS 2024 - Proceedings, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 11–15. doi: 10.1109/AICAS59952.2024.10595961.

[18] M. Rasheed, S. Iqbal, A. Jaffar, and S. Akram, "Advanced deep learning-based brain tumor classification using a novel customized CNN and optimized residual network," PLoS One, vol. 20, no. 10, p. e0334430, 2025.

[19] S. Iqbal, U. Mujtaba, M. Shafiq, D. N. Ahmed, and T. Amjad, "Multi-Level Deep CNN for Glaucoma Detection and Classification," in 2026 7th International Conference on Advancements in Computational Sciences (ICACS), 2026, pp. 1–6.

[20] P. E. da Costa Filho et al., "Machine-Learning-Based Classification of Electronic Devices Using an IoT Smart Meter," Informatics, vol. 12, no. 2, Jun. 2025. doi: 10.3390/informatics12020048.

[21] M. Beltrán-Escobar, T. E. Alarcón, J. Y. Rumbo-Morales, S. López, G. Ortiz-Torres, and F. D. J. Sorcia-Vázquez, "A Review on Resource-Constrained Embedded Vision Systems-Based Tiny Machine Learning for Robotic Applications," Algorithms, vol. 17, no. 11, Nov. 2024. doi: 10.3390/a17110476.

[22] V. M. Sineglazov and V. P. Khotsyanovsky, "Camera Image Processing on Esp32 Microcontroller With Help of Convolutional Neural Network.," Electronics & Control Systems, vol. 72, no. 2, 2022.

[23] C. Konar, D. Das, K. Jc, S. Singh, and others, "Edge-optimized threat detection with Florence v2 for autonomous surveillance in resource-constrained environments," PeerJ Comput. Sci., vol. 12, p. e3579, 2026.

[24] A. Mampage, S. Karunasekera, and R. Buyya, "Deep Reinforcement Learning for Scheduling Applications in Serverless and Serverful Hybrid Computing Environments," IEEE Trans. Serv. Comput., vol. 18, no. 2, pp. 718–728, 2025. doi: 10.1109/TSC.2024.3520864.