U.Porto Journal of Engineering, 9:5 (2023) 65-76
ISSN 2183-6493
DOI: 10.24840/2183-6493_009-005_002156

Received: 18 April, 2023
Accepted: 26 June, 2023
Published: 24 November, 2023

Optimisation Techniques for Compact CNN on Embedded Systems for Gesture Recognition

João Carlos N. Bittencourt[1,2], Walber Conceição de Jesus Rocha[3]
[1]Faculty of Engineering, the University of Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal (joaocarlos@ufrb.edu.br) ORCID 0000-0002-4540-512X
[2]Centre for Exact and Technological Sciences, Federal University of Recôncavo da Bahia, Rua Ruy Barbosa 720, Centro - Cruz das Almas, Brazil
[3]Centre for Exact and Technological Sciences, Federal University of Recôncavo da Bahia, Rua Ruy Barbosa 720, Centro - Cruz das Almas, Brazil (walber.jesus@aluno.ufrb.edu.br)

Abstract
Embedded applications are increasingly prevalent in various domains, from consumer electronics to industrial automation and smart cities. With the advances in integrated circuit manufacturing technologies, low-power chips can now execute complex algorithms, including machine learning models. However, the computational constraints of embedded devices require compact and efficient neural network models, as well as software frameworks and optimisation techniques tailored to their hardware resources. This study investigates the implementation of Convolutional Neural Network (CNN) models for gesture recognition on an STM32F4 microcontroller, by exploring the impact of freezing layers, fine-tuning and pruning techniques on pre-trained CNN models. The results demonstrate that fine-tuning and freezing layers improve accuracy by up to 18 %. Finally, this study demonstrates that pruning reduced the model size by 90 % with a 30 % accuracy impact, when compared to the uncompressed model, enabling it to perform gesture recognition on small devices. These findings are significant for developing software and optimisation techniques for embedded systems, particularly in the context of the Internet of Things.

Author Keywords. Embedded Systems, TinyML, Internet of Things, Convolutional Neural Networks, MobileNet, EfficientNet, Gesture Recognition.

Type: Research Article
Open Access Peer Reviewed CC BY

1. Introduction

Embedded systems contribute significantly in many fields, including industry, military, automotive, healthcare, robotics, and consumer devices to rapidly process collected data, often critical for these applications (Jazdi 2014; Shi et al. 2011). Unlike general-purpose computers, embedded systems prioritise a combination of hardware and software that can compute predetermined functions to meet strict requirements, such as memory as operation frequency, and power. However, due to limitations in storage and computational resources, processing data in such devices is not always feasible, and it must be transmitted to a server for further processing. Such a requirement can lead to undesirable delays, compromising the real-time requirements of critical applications. In recent years, embedded platforms have advanced significantly in processing capability, integration interfaces, low power, and low latency (Branco, Ferreira, and Cabral 2019). These improvements have allowed modern devices to perform complex algorithms that were once only possible on high-capacity computing units, including multimedia processing, computer vision, and machine learning

(Berthelier et al. 2020). However, challenges in providing specific solutions for embedded platforms remain since current techniques rely on memory and processing capacity, often leading to high energy consumption. Therefore, there is a need to develop more efficient methods to optimise the performance of embedded platforms, especially for applications with limited resources. Convolutional Neural Networks (CNN) have become fundamental for several applications, including image classification, similarity grouping, and gesture recognition (Aggarwal 2018). Gesture recognition has emerged as an effective tool for sign language recognition, video monitoring, human-robot collaboration, virtual gaming, and home automation (Liu and Wang 2018; Mahmud et al. 2022). Hand gestures also play a critical role in the healthcare and education industries to assist people with special needs (Ramani et al. 2022). The main challenge in developing a robust hand gesture recognition system is considering both spatial and temporal information, especially in the sign language analysis (Bastos, Angelo, and Loula 2015; Al-Hammadi et al. 2020; Ramani et al. 2022).

To enable gesture recognition on modern embedded platforms, it is crucial to explore lightweight models, machine learning frameworks, and optimisation techniques that can satisfy the computational restrictions. Currently, methods based on visual information require operations over large amounts of data, making it difficult to fit them into an embedded platform. By leveraging lightweight models, such as MobileNet (A. G. Howard et al. 2017) and EfficientNet (Tan and Le 2019), and optimisation techniques (Wu et al. 2020), gesture recognition can be performed by a microcontroller, thereby optimising the computing time, reducing implementation costs and failures. This approach improves the efficiency of gesture recognition and enables remote applications without cloud servers.

Despite the potential to provide high-quality solutions with satisfactory accuracy, there is still a lack of performance analysis in the literature regarding such methods in embedded devices. Therefore, one must assess how to effectively define optimisation parameters and compress the models to fit into memory-constrained units.

This work analyses the impact of fine-tuning, freezing layers and pruning optimisation techniques in pre-trained models of MobileNets (Sandler et al. 2018) and EfficientNet (Tan and Le 2019) applied for gesture recognition. These models are considered lightweight and suitable for real-time image classification processing on devices with limited resources. By doing so, we provide an accuracy of about 70 % while reducing the model size by 90 % compared to a base model. The main contributions of this paper are as follows:

- Our experimental results analyse several lightweight CNN models and evaluate freezing layers, fine-tuning and pruning optimisation in terms of accuracy. The aim is to give a further direction for the future development of gesture recognition in embedded systems.
- We provide an evaluation of the proposed models and optimisations on a microcontroller unit with constrained resources, evaluating the memory requirements. The rationale is that smaller models can better fit into low-footprint devices.

The following sections of this paper are organised as follows. Section 2 presents the specifications of the CNN architectures for gesture recognition, the approach used to analyse fine-tuning, freezing layers, and pruning optimisation techniques parameters. Section 3 details the hardware components involved in the experiments. Section 4 discusses the experimental results. Finally, Section 3 presents the conclusions and envisions future research directions.

2. Methodology

2.1. Convolutional Neural Network Architectures

This study analysed five neural network architectures based on their simplicity and suitability for application in platforms with strict computational and storage constraints. Table 1 presents the relevant characteristics of the studied architectures without considering optimisations. In compact CNN architectures, one needs to be concerned with the total number of parameters and network depth, as these characteristics directly impact storage requirements, often a strict constraint in embedded systems.

CNN Model	**Size (MB)**	**Parameter (M)**	**Deep (Layers)**
MobileNet	16	4.3	55
MobileNetV2	14	3.5	105
MobileNetV3	14	2.5	105
EfficientNet	29	5.3	132

Table 1: Characteristics of the compact CNN models used in this work.

MobileNet networks are compact architectures that use depthwise convolutions to replace a complete convolutional operator with a factored structure. The convolution consists of two layers: a depthwise convolution that uses a single convolutional filter per input channel, and a 1 x 1 convolution, known as pointwise convolution, which constructs new features by calculating linear combinations of input channels (A. G. Howard et al. 2017). The layer organisation of MobileNet networks consists of separable convolutions, except for the input layer, which performs a complete convolution to reduce processing requirements and model file size. Each convolutional layer follows a normalisation layer (batch normalisation) and the application of Rectified Linear Unit (ReLU) non-linear activation function. The final layer uses softmax to present the inference results through a set of classes.

MobileNetV2 uses an inverted residual structure, where the input and output of the residual block are convolutional layers. Unlike its predecessor, which uses an expanded representation at the input layer, MobileNetV2 uses depthwise convolutions to filter the layers while preserving the model's representation capacity (Sandler et al. 2018). In contrast, MobileNetV3 builds upon the MobileNetV2 architecture and modifies its structure using optimisations proposed by the Neural Architecture Search (NAS), hardware recognition, and the NetAdapt algorithm (A. Howard et al. 2019). NAS technique determines optimised neural network structures in each block, while NetAdapt provides the best organisation in each layer by adjusting the number of filters in the model. MobileNetV3 leverages this structure to find a faster and more accurate architecture depending on its specifications.

The EfficientNetB0 architecture applies a uniform scaling method to the input layer width, depth, and resolution, along with a set of coefficients (Tan and Le 2019). It consists of 18 convolutional layers, using kernel sizes of 3 x 3 or 5 x 5. These convolution filters reduce the resolution of the hidden layers, which compresses the feature map's size while expanding the layer's width and increasing accuracy. Following the same approach as MobileNet, the final layer of EfficientNetB0 uses a softmax function to present the inference results through a set of classes.

2.2. Training Dataset

Several datasets for hand gesture recognition have been made available in the literature (Kapitanov, Makhlyarchuk, and Kvanchiani 2022; Pi et al. 2019; Maqueda et al. 2015). This work relies on Maqueda et al. (2015) dataset, which consists of high-resolution images and is used to validate hand gesture recognition systems for human-computer interaction. The dataset contains images captured in realistic scenes with non-uniform backgrounds and other moving objects. It is divided into five gesture classes that mimic the functions of a computer mouse, as presented in Figure 1, and comprises 2,647 RGB-encoded images. Although the gestures are related to the computer mouse, it is possible to map them to gesture control in robotic platforms, industrial machinery, and other human-interaction applications.

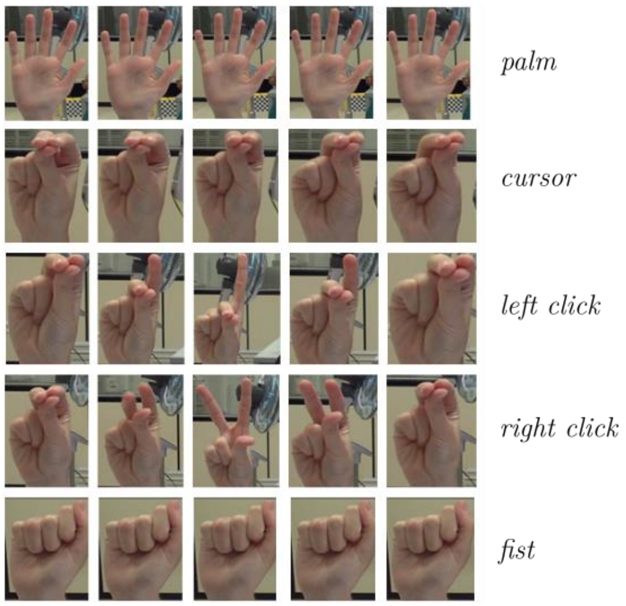


Figure 1: This set encompasses five distinct hand gestures. To aid the training of machine learning models, the gestures have several short video sequences available. Extracted from Maqueda et al. (2015).

To evaluate the performance of the gesture recognition model 70 % of the samples were used for training and 30 % for validation and testing. The images in the dataset are grouped based on the number of frames needed to represent a gesture accurately. We selected sequences of gestures performed by five people to train the model. For testing, the setup sequences have gestures performed by six people from the same dataset. These test sequences contained non-uniform backgrounds and moving objects, which made them more realistic and challenging for the model to recognise.

2.3. Training Setup

Convolutional Neural Networks (CNNs) are powerful tools for image recognition and other computer vision tasks. However, training a CNN from scratch can be challenging due to the large amount of labelled data and the significant computational resources required. To address this challenge, researchers have proposed neural network training strategies to optimise the performance and accuracy of the model (Bengio 2009). One common approach is to use a pre-trained CNN and fine-tune it for a specific task. Such a method involves updating the weights of some or all layers of the CNN using a smaller dataset and a lower learning rate. By doing this, the CNN can adapt to the new task while preserving the general features learned from the original dataset. This approach is highly effective when the dataset is large enough to achieve accurate results (Goodfellow, Bengio, and Courville 2016). However, fine-tuning a pre-trained CNN can still be time-consuming and requires significant computational resources. Therefore, researchers are exploring new methods to improve the efficiency of this process, such as transfer learning and knowledge distillation.

The training process consists of five steps: (i) importing the base model, (ii) refactoring the architecture, (iii) freezing layers, (iv) training, and (v) fine-tuning. The first step imports the base model and initialises its parameters with the images from the dataset. This process also removed the upper layers to enable feature extraction during training. Replacing the layers that do not correspond to the desired classes is performed in the refactoring step. This results in a final set of fully connected layers, replaced by two new layers. The first fully connected layer has 128 neurons. These neurons gather the data extracted by the previous layers and prepare it for the following softmax layer, which has five neurons. The last activation layer determines the probability of the input belonging to one of the pre-trained classes. Freezing a layer means that its weights remain unchanged during the training because it does not need any modification in the data contained. The model learns the parameters related to new classes while the frozen layers retain their original values. The proposed freezing approach depicted in Figure 2 is performed from two different scenarios, freezing 10 % and 20 % of the total layers according to the model.

During the training, the model learns to generalise the images based on their probability of belonging to the new classes. Before training, configuration parameters were specified, such as the type of optimisation, loss function, learning rate, and evaluation metric, as presented in Table 2. We trained the model for 50 epochs.

Parameter	**Base Training**	**Fine-tuning**
Size of the input image	(64, 64, 3)	n.a.
Optimisation function	Adam	RMSprop
Evaluation metric	Categorical CrossEntropy	n.a.
Learning rate	1-5	1-6

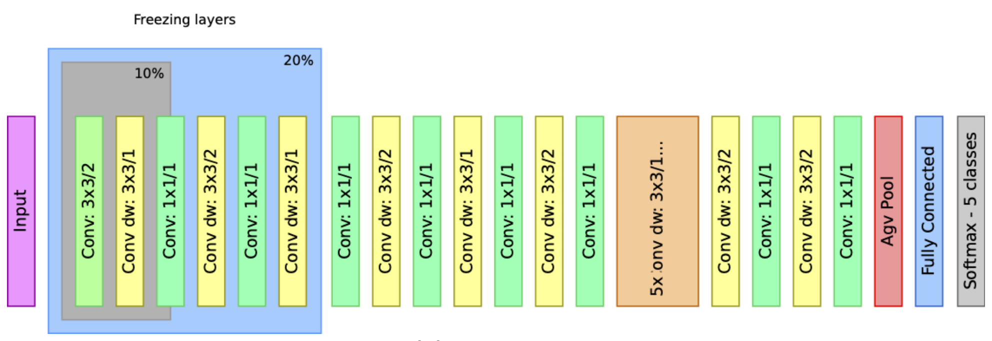
Table 2: Parameters applied in the training setup.

Fine-tuning involves repeating the training process with a low learning rate to adjust the model representations and make them more relevant to gesture recognition. A model compression technique was applied during fine-tuning to reduce computational costs for embedding the model in a microcontroller platform. The method relies on the network sparsity control proposed by Fernandes and Kung (2021), which applies the hyperparameter α variation to all layers. Besides reducing the number of hyperparameters, this technique also reduces the model file size, which is essential for embedding in low-capacity flash memories.

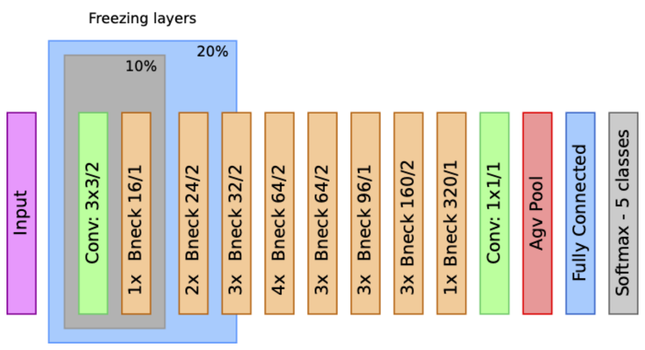
The training for each architecture involved iterating four times, with the α parameter decreasing by 0.25 in the range of 1 to 0.25 at each iteration. This process resulted in 32 models validated according to the methodologies described in the following section. By applying these fine-tuning and model compression techniques, we were able to optimise the performance and efficiency of the CNN models for gesture recognition on a resource-constrained device.

2.4. Test and Validation

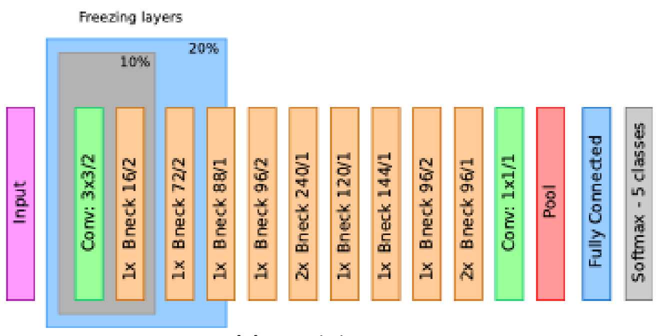
The performance evaluation used the confusion matrix validation technique (Cavalin and Oliveira 2018). This method verifies the capability of classifying gestures and identifies differences between expected and obtained results. The confusion matrix allows for the analysis of CNN performance based on the actual values and those predicted by the classifier. To this end, prediction results are either “true positives”, “true negatives”, “false positives”, or “false negatives”.



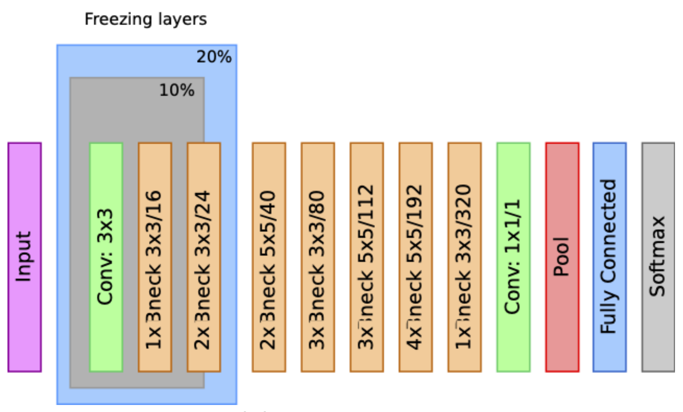
(a) MobileNet



(b) MobileNetV2



(c) MobileNetV3



(d) EfficientNetB0

Figure 2: Architectures with fine-tuning and freezing layers. The grey boxes indicate 10 % of the frozen layers, and the blue boxes indicate 20 % frozen layers.

The confusion matrix is structured according to the classification frequency for each input, interpreting as true positive the case where the input data is positive, and the classifier also considers it a positive (da Fona Costa and Cesar 2018). Equation 1 describes the method for calculating accuracy, defined as the quotient between the total number of correct inferences and the inferences performed.

$$\text{accuracy} = \frac{\text{correct inferences}}{\text{inferences}} \tag{1}$$

Equation 2 presents the method for extracting the precision based on the proportion of identifications, measured by the quotient between “true positive” results and the sum of false positives and false negatives.

$$\text{precision} = \frac{\sum true\ positive}{\sum false\ positive\ +\ \sum false\ negative} \tag{2}$$

The obtained accuracy, precision metrics, and their final size are crucial in deciding whether to implement them on the hardware platform. The confusion matrix technique provided valuable insights into the performance, enabling the determination of the correct and incorrect classifications made by the models. Finally, considering the performance metrics and the computational requirements, one can select suitable models for implementation on the hardware platform.

3. Hardware Platform

The hardware platform used in this study is the STM32F4-Discovery development board, which features the STM32F407G microcontroller. This microcontroller has a clock speed of 168 MHz, 1 MB of flash memory, and 192 KB of RAM. Its architecture includes a floating-point unit, favouring the efficient implementation of CNN models. The test platform connected to a desktop computer relies on an FTDI serial module that uses the RS232 USART transmitter/receiver protocol. Thus, this prototype transmits the images and captures inference data via the serial channel.

Current ARM microprocessors offer a vast software and library ecosystem, facilitating firmware development for final products. Moreover, a new set of tools and libraries are devoted to converting machine learning models into compatible C++ firmware. We leverage the STM32 X-CUBE-AI expansion package to transform the models, which offers a comprehensive set of tools and libraries, allowing developers to create machine learning applications efficiently. The library is compatible with Keras and TensorFlow Lite libraries and supports quantisation techniques. Additionally, the tool enables the storage of weights in an external flash memory and activation buffers in external RAM, enabling the implementation of complex networks in low-capacity microcontrollers.

The models initially developed with TensorFlow in Python are converted to TensorFlow Lite (David et al. 2021) models and then transformed into compatible firmware. The design of the machine learning algorithm in STM32 X-CUBE-AI comprises two stages: (i) analysis; and (ii) validation. The analysis stage optimises the implementation by mapping the architecture, weights, and parameters. Such a procedure involves checking hardware restrictions after quantisation, which produces the C++ firmware and memory reports. The quantisation process involves applying it to values in floating point, which results in 4- or 8-bit precision, thus, reducing the model size, CPU latency, and energy consumption. The validation stage tests the classifier with input samples. The results are compared to those obtained with the reference model before conversion to ensure the implementation is accurate and reliable.

4. Experimental Results

Experimentation was conducted to evaluate not only the accuracy of the different networks but also their deployability on embedded devices with limited resource capabilities. In this regard, we accompanied the accuracy analysis with an evaluation of memory constraints based on the optimisation parameters. Our primary objective was to determine the feasibility of implementing the models on resource-constrained devices while maintaining acceptable levels of accuracy.

4.1. Accuracy Analysis

To assess the accuracy of the CNN models after fine-tuning, we compared them against the base model (without fine-tuning) and with fine-tuning by freezing 10 % and 20 % of the layers. Table 3 summarises the results obtained from the training and test/validation sets, allowing us to evaluate which architectures perform the best.

CNN Model	**Accuracy (%)**							
	Base model		**Fine-tuning**		**10 % freezing layers**		**20 % freezing layers**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
MobileNet	73.4	78.6	90.2	93.8	93.4	94.0	94.7	94.8
MobileNetV2	71.2	73.0	88.3	90.3	91.5	91.7	91.0	92.1
MobileNetV3	58.8	53.9	78.5	79.2	81.2	81.0	81.8	81.3
EfficientNetB0	70.4	63.9	83.2	83.8	92.2	91.8	92.5	91.5

Table 3: Accuracy comparison after fine-tuning and with layer freezing at 10 % and 20 % proportion.

Fine-tuning significantly impacts the accuracy of selected models. Such a solution increases up to 18 % in MobileNet accuracy compared to the base models, with no frozen layers. EfficientNet had an increase in accuracy of 23 % compared to its base model. MobileNet presented a 94 % and 95 % accuracy for the training and testing set in both freezing scenarios after fine-tuning. Among the MobileNet architectures, the MobileNetV3 presented the lowest accuracy rate after freezing, at about 81 % for the same training setup. The training and validation of the EfficientNetB0 model indicate an accuracy rate of 92 %, similar to MobileNetV2. Compared to the base model, optimisation based on fine-tuning and freezing layers by 20 % resulted in a 31 % accuracy improvement. The results demonstrate that freezing layers have a positive impact on accuracy. Thus, the following analysis concentrates only on those models with the freezing layers to evaluate their performance on a microcontroller unit.

4.2. Model Compression

We initially trained the models with unchanged sparsity parameters to establish a baseline for comparison metrics such as file size and model accuracy. The proposed fine-tuning involved gradually reducing the α parameter from 1.0 to 0.25, with steps of 0.25, while repeating the training process. Tables 4 and 5 present the experimental results for model size and accuracy after fine-tuning with 10 % and 20 % of the layers frozen, respectively.

According to the results, MobileNet achieved the best performance among the tested architectures with a 0.75 compression rate, reducing the model size by around 46 % with only a 2.5 % decrease in accuracy. However, we were unable to obtain significant reductions in model size for MobileNetV3 and EfficientNetB0. Furthermore, when comparing the results of freezing layers by 10 % and 20 %, we observed a reduction in model size.

For a 0.5 compression rate, MobileNetV2 also demonstrated the best compression of the model size, with an accuracy above 82 %, while MobileNetV3 and EfficientNetB0 showed limited compression capacity and significant loss in model accuracy. The pruning optimisation reduced the model size by up to 90 %, but higher compression rates could impact the accuracy by up to 30 %. Therefore, high compression rates may not be suitable for critical or safety systems where low efficiency could pose a risk of physical, personal, or financial damage. However, systems with soft restrictions can take advantage of compression techniques.

CNN Model	**α**							
	1.00		**0.75**		**0.50**		**0.25**	
	Size (MB)	**Accuracy (%)**	**Size (MB)**	**Accuracy (%)**	**Size (MB)**	**Accuracy (%)**	**Size (MB)**	**Accuracy (%)**
MobileNet	26.1	94.0	15.0	91.7	6.9	82.1	2.1	67.4
MobileNetV2	18.6	91.7	11.6	90.3	6.3	86.3	3.9	77.2
MobileNetV3	12.8	81.0	12.8	80.3	8.7	76.1	5.6	71.1
EfficientNetB0	33.1	91.8	33.1	84.2	25.7	74.1	16.2	69.4

Table 4: Compression results after fine-tuning with 10 % of freezing layers.

CNN Model	**α**							
	1.00		**0.75**		**0.50**		**0.25**	
	Size (MB)	**Accuracy (%)**	**Size (MB)**	**Accuracy (%)**	**Size (MB)**	**Accuracy (%)**	**Size (MB)**	**Accuracy (%)**
MobileNet	26.0	94.8	14.9	91.1	6.9	80.6	2.0	62.0
MobileNetV2	18.5	92.1	11.6	90.6	6.2	82.2	3.9	78.0
MobileNetV3	12.8	81.3	12.8	79.7	8.7	74.2	5.6	68.4
EfficientNetB0	33.1	91.5	33.1	81.2	25.7	73.1	16.2	66.4

Table 5: Compression results after fine-tuning with 20 % of freezing layers.

MobileNet and MobileNetV2 presented compressed models that can be implemented in platforms embedded with strict storage and computational restrictions, making them a good option for applications with such constraints.

4.3. Hardware Validation

Implementing machine learning algorithms into embedded devices requires a specific workflow, which involves transforming the trained model using a converter. In our experiments, the STM32 X-CUBE-AI automatic conversion process did not convert models based on MobileNetV3 and EfficientNetB0 architectures. This is due to the absence of support for Lambda and Rescaling layers on the tool.

RAM and flash memory are critical hardware specifications when implementing CNN on microcontrollers due to the large memory requirements of CNN. The amount of RAM and flash memory available on the microcontroller will determine the size and complexity of the CNN that can be implemented. Table 6 summarises the implementation requirements for MobileNet on the STM32F407G platform by freezing 20 % of the layers with 0.25 compression rate. It indicates that the model is implemented without requiring external memory when the compression rate is 0.25. However, when combined with an 8-bit quantisation and a compression rate of 0.50, it requires an external flash memory to store the parameters. Thus, quantisation reduces the memory requirements for RAM and Flash, increasing the model efficiency.

α	**Quantisation**					
	W/o		**4-bit**		**8-bit**	
	RAM (KB)	**Flash (MB)**	**RAM (KB)**	**Flash (MB)**	**RAM (KB)**	**Flash (MB)**
1.00	332.85	12.25	332.85	12.23	332.85	12.23
0.75	264.73	6.95	264.73	6.94	264.73	6.94
0.50	196.60	3.15	193.60	3.14	189.89	3.14
0.25	128.48	0.86	127.21	0.86	126.72	0.86

Table 6: RAM and Flash memory usage by the model based on the MobileNet architecture, when applying freezing 20 % of the layers.

Although it presents lower values of accuracy, this model was selected in the experiment since the goal is to use a lightweight architecture regarding the device limitations. These models were successfully converted and implemented on the hardware platform, demonstrating the effectiveness of the proposed workflow. The Table 6 provides information on RAM and Flash memory usage for both models and their classification accuracy on the test dataset.

On the other hand, the experimental results for MobileNetV2 show that it cannot be implemented on the STM32F407G, regardless of the compression and quantisation employed. By performing a compression of 0.25, the model requires 278 KB of RAM, making it unfeasible on this platform. However, despite this limitation, implementing the MobileNetV2 machine learning model on embedded platforms is feasible, as it has low-capacity requirements.

We validate the MobileNet model by inferring the validation dataset images on the STM32F4-Discovery platform through a software interface provided by the STM-CUBE-AI tool. According to the experiments, the *cursor*, *fist*, and *palm* gestures had the highest “true positive” classifications. However, the *left-click* and *right-click* gestures had higher classification errors, indicating that more training samples are needed to improve their generalisation in such a compact model. Despite a slight decrease in overall accuracy, the proposed methodology has demonstrated its ability to facilitate the implementation of machine-learning models on lightweight microcontrollers.

The MobileNet implementation with optimisation techniques on an embedded platform provides a promising solution for performing computer vision tasks on devices with low computation capacity. The experimental results confirm the accuracy reported in Table 5 due to the application of the pruning method. This solution is particularly relevant in IoT systems which require efficient and lightweight models. Possible applications include industry, robotics, and accessibility devices for people with disabilities. Overall, the MobileNet with optimisation techniques offers a practical solution for executing CNNs on embedded platforms, potentially paving the way for further advancements in this field.

5. Conclusions

The present study investigated compact CNN architectures, specifically MobileNet and EfficientNet networks. These models have lightweight structures with fewer parameters and a favourable model size for use in embedded systems. The study also included an analysis of optimisation techniques for an embedded platform focusing on gesture recognition. Experimental results demonstrate that fine-tuning with layer freezing improves accuracy, leading to an 18 % increase compared to the base model. Pruning also compresses the model by 90 %, enabling their implementation in embedded platforms. However, we verify that compression rates can negatively impact accuracy, reducing it by about 30 %.

This article contributes to low-cost embedded systems used in machine learning applications for IoT and smart cities. While the work focused on lightweight models, there is a need for research on other machine learning paradigms to explore the implementation of new neural networks on embedded systems platforms. It is also desirable to investigate optimisation methods on validated architectures through layer remodelling to increase network accuracy. Although the results demonstrated the efficiency of library optimisations for CNN synthesis, other frameworks may present a potential for similar applications. Future studies should explore these frameworks to identify further optimisation.

References

Aggarwal, Charu C. 2018. Neural Networks and Deep Learning: A Textbook. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-94463-0.

Al-Hammadi, Muneer, Ghulam Muhammad, Wadood Abdul, Mansour Alsulaiman, Mohamed A. Bencherif, and Mohamed Amine Mekhtiche. 2020. "Hand Gesture Recognition for Sign Language Using 3DCNN." IEEE Access 8: 79491–509. https://doi.org/10.1109/ACCESS.2020.2990434.

Bastos, Igor L.O., Michele F. Angelo, and Angelo C. Loula. 2015. "Recognition of Static Gestures Applied to Brazilian Sign Language (Libras)." In 2015 28th SIBGRAPI Conference on Graphics, Patterns and Images, 305–12. Salvador, Bahia, Brazil: IEEE. https://doi.org/10.1109/SIBGRAPI.2015.26.

Bengio, Y. 2009. "Learning Deep Architectures for AI." Foundations and Trends® in Machine Learning 2 (1): 1–127. https://doi.org/10.1561/2200000006.

Berthelier, Anthony, Thierry Chateau, Stefan Duffner, Christophe Garcia, and Christophe Blanc. 2021. "Deep Model Compression and Architecture Optimization for Embedded Systems: A Survey." Journal of Signal Processing Systems 93 (8): 863–78. https://doi.org/10.1007/s11265-020-01596-1.

Branco, Sérgio, André G. Ferreira, and Jorge Cabral. 2019. "Machine Learning in Resource-Scarce Embedded Systems, FPGAs, and End-Devices: A Survey." Electronics 8 (11): 1289. https://doi.org/10.3390/electronics8111289.

Cavalin, Paulo, and Luiz Oliveira. 2019. "Confusion Matrix-Based Building of Hierarchical Classification." In Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, edited by Ruben Vera-Rodriguez, Julian Fierrez, and Aythami Morales, 11401:271–78. Lecture Notes in Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-13469-3_32.

Costa, Luciano Da Fona, and Roberto Marcond Cesar, Jr. 2018. Shape Classification and Analysis: Theory and Practice, Second Edition. 0 ed. CRC Press. https://doi.org/10.1201/9781315222325.

David, Robert, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, et al. 2020. "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems." https://doi.org/10.48550/ARXIV.2010.08678.

Fernandes, Marcelo A. C., and H. T. Kung. 2021. "A Novel Training Strategy for Deep Learning Model Compression Applied to Viral Classifications." In 2021 International Joint Conference on Neural Networks (IJCNN), 1–9. Shenzhen, China: IEEE. https://doi.org/10.1109/IJCNN52387.2021.9534430.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: The MIT Press.

Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." https://doi.org/10.48550/ARXIV.1704.04861.

Howard, Andrew, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, et al. 2019. "Searching for MobileNetV3." In 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 1314–24. Seoul, Korea (South): IEEE. https://doi.org/10.1109/ICCV.2019.00140.

Jazdi, N. 2014. “Cyber Physical Systems in the Context of Industry 4.0.” In 2014 IEEE International Conference on Automation, Quality and Testing, Robotics, 1–4. Cluj-Napoca, Romania: IEEE. https://doi.org/10.1109/AQTR.2014.6857843.

Kapitanov, Alexander, Andrew Makhlyarchuk, and Karina Kvanchiani. 2022. “HaGRID - HAnd Gesture Recognition Image Dataset.” https://doi.org/10.48550/ARXIV.2206.08219.

Lakshmi Ramani, B., T. Sri Lakshmi, N. Sri Durga, Shaik Sana, T. Sravya, and N. Jishitha. 2023. “Recognition of Hand Gesture-Based Sign Language Using Transfer Learning.” In Computer Communication, Networking and IoT, edited by Suresh Chandra Satapathy, Jerry Chun-Wei Lin, Lai Khin Wee, Vikrant Bhateja, and T. M. Rajesh, 459:95–103. Lecture Notes in Networks and Systems. Singapore: Springer Nature Singapore. https://doi.org/10.1007/978-981-19-1976-3_12.

Liu, Hongyi, and Lihui Wang. 2018. “Gesture Recognition for Human-Robot Collaboration: A Review.” International Journal of Industrial Ergonomics 68 (November): 355–67. https://doi.org/10.1016/j.ergon.2017.02.004.

Mahmud, Jubayer Al, Bandhan Chandra Das, Jungpil Shin, Khan Md. Hasib, Rifat Sadik, and M. F. Mridha. 2022. “3D Gesture Recognition and Adaptation for Human–Robot Interaction.” IEEE Access 10: 116485–513. https://doi.org/10.1109/ACCESS.2022.3218679.

Maqueda, Ana I., Carlos R. del-Blanco, Fernando Jaureguizar, and Narciso García. 2015. “Human–Computer Interaction Based on Visual Hand-Gesture Recognition Using Volumetric Spatiograms of Local Binary Patterns.” Computer Vision and Image Understanding 141 (December): 126–37. https://doi.org/10.1016/j.cviu.2015.07.009.

Pi, Lanyue, Kai Liu, Yun Tie, and Lin Qi. 2019. “A New Comprehensive Database for Hand Gesture Recognition.” In Genetic and Evolutionary Computing, edited by Jeng-Shyang Pan, Jerry Chun-Wei Lin, Bixia Sui, and Shih-Pang Tseng, 834:253–61. Advances in Intelligent Systems and Computing. Singapore: Springer Singapore. https://doi.org/10.1007/978-981-13-5841-8_27.

Sandler, Mark, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 4510–20. Salt Lake City, UT: IEEE. https://doi.org/10.1109/CVPR.2018.00474.

Shi, Jianhua, Jiafu Wan, Hehua Yan, and Hui Suo. 2011. “A Survey of Cyber-Physical Systems.” In 2011 International Conference on Wireless Communications and Signal Processing (WCSP), 1–6. Nanjing, China: IEEE. https://doi.org/10.1109/WCSP.2011.6096958.

Tan, Mingxing, and Quoc V. Le. 2019. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.” https://doi.org/10.48550/ARXIV.1905.11946.

Wu, Hao, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. 2020. “Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation.” https://doi.org/10.48550/ARXIV.2004.09602.