Extending AIfES with Depthwise Convolution

Implementation and Evaluation of Depthwise Convolution on Microcontrollers

Zico Krassenburg[1]

Supervisor(s): Marco Zuñiga Zamalloa[1]**, Hao Liu**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Zico Krassenburg
Final project course: CSE3000 Research Project
Thesis committee: Marco Zuñiga Zamalloa, Hao Liu, Jana Weber

Abstract

Artificial Intelligence is increasingly being used in everyday devices. However, most AI systems are designed to run on powerful computers or cloud servers rather than on small, low power devices such as microcontrollers. Running AI directly on these devices can reduce energy consumption and enable systems to operate without an internet connection.

AIfES (Artificial Intelligence for Embedded Systems) is a machine learning framework that allows neural networks to be trained directly on microcontrollers. However, it currently lacks support for depthwise convolution, an important operation used in efficient neural network architectures such as MobileNet. As a result, many modern computer vision models cannot be trained within the framework.

This project extends AIfES with support for depthwise convolution and integrates the new operator into the existing training pipeline. The implementation was validated using a combination of manually verified test cases, comparisons with TensorFlow, and image classification experiments on embedded hardware.

The results show that the new operator functions correctly during both inference and training. Models containing the implemented layer successfully learned classification tasks and achieved behavior similar to equivalent TensorFlow models. By adding support for depthwise convolution, this work expands the range of neural network architectures that can be trained directly on microcontrollers and contributes to making on-device AI more practical and flexible.

1 Introduction

Artificial Intelligence has become a fundamental component of modern computing systems. However, most AI still rely heavily on resource rich environments such as servers and GPUs. This raises an important question: can AI be efficiently deployed on less resource rich devices such as microcontrollers, which are commonly found in wearables, sensors, and embedded systems? Using AI on these devices introduces limitations in memory, computation, and energy consumption.

Recent advances in TinyML (Tiny Machine Learning), a field focused on deploying machine learning models on resource-constrained embedded devices such as microcontrollers, have demonstrated that lightweight inference can be performed on microcontrollers. For example, frameworks such as TensorFlow Lite for Microcontrollers enable the deployment of compact neural networks on embedded hardware [4]. Similarly, AIfES provides support for both inference and limited training directly on microcontrollers, offering a promising direction towards AI usage on microcontrollers [1]. In addition, Neural Architecture Search (NAS) has been widely used to automatically design efficient neural networks that meet strict hardware constraints, significantly improving performance efficiency tradeoffs [3]. Previous work has also explored energy aware AI design, including methods for estimating the energy consumption of neural network inference to guide model selection [7]. Furthermore, event driven sensing approaches have been proposed to reduce energy usage by activating computation only when relevant input is detected [2].

Despite these advances, most of the existing research focuses mainly on optimizing inference efficiency or designing compact architectures. **Training directly on embedded devices remains relatively underexplored.** Although AIfES enables microcontroller training, it currently lacks support for important operators such as depthwise convolution, which are essential for efficient deep learning models like MobileNet [5]. These models are both designed for computer vision tasks like image classification and object detection. This limitation restricts the applicability of on-device learning and prevents the deployment of more expressive models. Consequently, an important gap remains in enabling efficient, flexible, and fully functional training pipelines directly on microcontrollers.

This paper focuses on addressing this gap through the following research question: How can the AIfES framework be extended to support DepthConv operators, enabling on-device training of computer vision models on microcontrollers? This question targets a critical limitation in current TinyML systems and represents a necessary step toward achieving fully autonomous, adaptive AI pipelines on constrained hardware. By enabling depthwise convolution operations, commonly used in efficient architectures, we aim to expand the range of trainable models that can run entirely on-device.

The main contributions of this work are three-fold, First, we analyze the architectural and computational constraints of the AIfES framework and identify the requirements for integrating DepthConv operators. Second, we design and implement an extension that enables depthwise convolution within the training pipeline on microcontrollers. Third, we evaluate the proposed solution by demonstrating successful on-device training, analyzing convergence behavior.

The remainder of this paper is structured as follows. Section 2 contains the background and methodology used to answer the research question. Section 3 describes the system design and implementation of the proposed DepthConv extension. Section 4 presents the experimental setup, including hardware configuration and evaluation methodology. Section 5 reflects on the ethical aspects of our research. Section 6 discusses the results, focusing on training performance.

Finally, Section 7 concludes the paper and outlines directions for future work.

2 Background and Methodology

2.1 AIfES Framework

AIfES (Artificial Intelligence for Embedded Systems) is a lightweight machine learning framework designed specifically for embedded devices and microcontrollers [1]. Unlike many TinyML frameworks that primarily focus on inference, AIfES also supports on-device training, enabling neural networks to adapt directly on constrained hardware without relying on external servers or cloud infrastructure.

The framework provides implementations for several neural network layers and operators, including dense layers and standard convolution layers. These components are optimized for devices with limited computational resources, memory capacity, and energy availability. Because microcontrollers operate under strict hardware constraints, implementing training functionality requires careful consideration of memory usage and computational efficiency.

Although AIfES already supports standard convolution operations, it currently lacks support for Depthwise Convolution (DepthConv) operators. As a result, many efficient lightweight computer vision architectures cannot currently be trained directly within the framework.

2.2 Depthwise Convolution

Depthwise convolution is a computationally efficient alternative to standard convolution and is widely used in lightweight neural network architectures such as MobileNet [5]. In a standard convolution layer, each output channel is generated by applying a convolution filter across all input channels simultaneously. Consequently, each filter performs both spatial feature extraction and channel mixing in a single operation.

A standard convolution can be expressed as

$$G_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} F_{k+i-1,l+j-1,m} \tag{1}$$

where F denotes the input feature map, K the convolution kernel, and G the resulting output feature map. The indices k and l represent spatial positions, while m and n denote the input and output channels respectively.

Depthwise convolution decomposes this operation by applying a separate spatial filter to each input channel independently. Instead of combining information across channels during the convolution itself, each channel is processed separately according to

$$\hat{G}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} F_{k+i-1,l+j-1,m} \tag{2}$$

where $\hat{K}$ represents the depthwise convolution kernel associated with channel m.

Because depthwise convolution performs only spatial filtering, a subsequent 1 x 1 pointwise convolution is typically used to combine information between channels. Together, these operations form a depthwise separable convolution. Figure 1 illustrates how a standard convolution filter can be decomposed into a depthwise convolution followed by a pointwise convolution.

A major motivation for using depthwise separable convolutions is the reduction in computational complexity. According to Howard et al.[5], the computational cost for a standard convolution with kernel size $D_K \times D_K$, M input channels, N output channels, and a feature map size of $D_F \times D_F$ is

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F. \tag{3}$$

For a depthwise separable convolution, the depthwise stage requires

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \tag{4}$$

operations, while the pointwise stage requires

$$M \cdot N \cdot D_F \cdot D_F. \tag{5}$$

The total computational cost therefore becomes

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F. \tag{6}$$

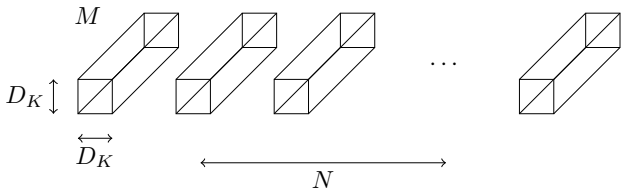
Compared to a standard convolution, the relative computational cost is

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2} \tag{7}$$

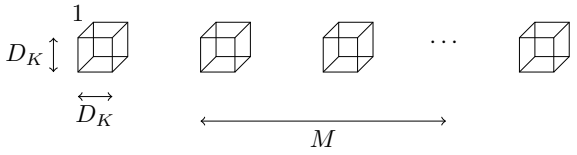
of that required by a standard convolution. This reduction in arithmetic operations and trainable parameters makes depthwise separable convolutions particularly attractive for TinyML applications running on resource-constrained microcontrollers.

The reduction in computational complexity is achieved at the cost of a small reduction in representational capacity, since channel interactions are deferred to the pointwise convolution stage. Nevertheless, MobileNet and related architectures have shown that depthwise separable convolutions can maintain competitive accuracy while significantly reducing resource requirements.

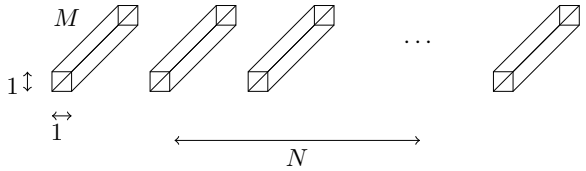
Despite its simple mathematical formulation, implementing depthwise convolution efficiently on embedded devices is non-trivial. Existing convolution implementations are typically designed around filters that operate across all channels



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters (Pointwise Convolution)

Figure 1: Comparison of standard convolution, depthwise convolution, and pointwise convolution filters. Adapted from Howard et al. [5].

simultaneously. Supporting depthwise convolution therefore requires dedicated handling of channel-wise kernels, tensor indexing, memory management, and gradient computation. These challenges motivate the implementation presented in Section 3.

3 Design and Implementation of the DepthConv2D Layer

The main contribution of this work is the implementation of a Depthwise Convolution (DepthConv2D) layer for the AIfES framework. Since AIfES already contains support for standard convolution layers, the existing Conv2D implementation was used as a starting point for the design of the new operator. The implementation was developed in C, the programming language used throughout AIfES.

The goal of the implementation was to enable both inference and training of models containing depthwise convolution layers while maintaining compatibility with the existing AIfES architecture.

3.1 Forward Propagation Implementation

To support depthwise convolution within AIfES, a new trainable layer was implemented. The implementation follows the mathematical formulation presented in Equation 2 and computes the convolution independently for each input channel. To make this work several implementation considerations were required.

AIfES stores tensors using contiguous memory buffers and supports configurable stride, padding, dilation, and kernel sizes. Consequently, the implementation could not simply iterate over image pixels. Instead, the correct memory locations had to be computed for every input channel, kernel position, and output position while maintaining compatibility with the tensor representation used throughout the framework. Another important consideration was memory usage. Creating temporary tensors for intermediate computations would increase memory consumption significantly and limit the size of trainable networks. To avoid this overhead, the implementation performs all computations directly on the input and output tensors managed by AIfES.

Unlike a standard convolution layer, where each output feature map is computed from all input channels simultaneously, the DepthConv2D layer applies a separate kernel to each channel and stores the resulting feature maps directly in the output tensor. Consequently, no accumulation across channels is required during the convolution operation.

Algorithm 1 summarizes the forward propagation procedure. The algorithm iterates over all input channels and output positions. For each output element, a weighted sum is computed by multiplying the corresponding input values with the kernel weights and accumulating the results. The computed value is then stored in the output tensor.

Algorithm 1 DepthConv2D Forward Propagation

```
for each input channel c do
    for each output position (x, y) do
        sum ← 0
        for each kernel element (i, j) do
            sum ← sum + input[c][x + i][y + j] ·
            kernel[c][i][j];
        end for
        output[c][x][y] ← sum;
    end for
end for
```

3.2 Backpropagation Implementation

To enable on-device training, gradient computation was implemented for both the trainable kernel parameters and the input tensor. During backpropagation, gradients must be propagated through the depthwise convolution layer so that the optimizer can update the kernel weights and earlier layers can continue receiving gradient information.

Algorithm 2 summarizes the implemented backpropagation procedure. Similar to the forward pass, all computations are performed independently for each channel. For every output gradient value, contributions are accumulated to both the kernel gradients and the input gradients.

In Algorithm 2, X denotes the input tensor, K denotes the depthwise convolution kernel, and δ denotes the gradient re-

ceived from the next layer. The variables g_K and g_X represent the gradients of the loss function with respect to the kernel weights and input tensor respectively.

Algorithm 2 DepthConv2D Backpropagation

for each input channel c **do**
  for each output gradient position (x, y) **do**
    for each kernel element (i, j) **do**
      $g_K[c][i][j] \leftarrow g_K[c][i][j]$
      $+X[c][x+i][y+j] \cdot \delta[c][x][y]$
      $g_X[c][x+i][y+j] \leftarrow g_X[c][x+i][y+j]$
      $+K[c][i][j] \cdot \delta[c][x][y]$
    end for
  end for
end for

The first update accumulates the gradient of each kernel weight by multiplying the corresponding input value with the output gradient. The second update propagates gradients back to the input tensor by multiplying the kernel weight with the output gradient. Since each channel is processed independently, no cross-channel gradient accumulation is required. This simplifies the implementation compared to standard convolution and naturally aligns with the structure of the depthwise convolution operation.

The resulting gradients are stored using the existing AIfES memory management infrastructure. Consequently, no modifications to the framework's optimization algorithms were required, allowing the new layer to be trained using existing optimizers such as SGD and Adam.

3.3 Integration into AIfES

After implementing forward propagation and backpropagation, the new DepthConv2D layer was integrated into the existing AIfES training pipeline. The layer follows the same interface conventions as other trainable AIfES layers and can therefore be combined with existing network components without modification. Once compiled into a model, the layer participates in forward propagation, gradient computation and parameter updates in the same manner as the existing convolution layers. As a result, models containing depthwise convolution layers can be constructed, trained, and evaluated using the same workflow as other AIfES models.

4 Evaluation of the DepthConv2D Implementation

4.1 Experimental Setup

The goal of this evaluation was to determine whether the implemented DepthConv2D layer could be successfully integrated into the AIfES training pipeline and used to train convolutional neural networks on embedded hardware.

All experiments were performed using the AIfES framework on an Arduino Nano 33 BLE microcontroller. The implementation was evaluated using a combination of synthetic and real world image classification tasks. In addition, the behaviour of the implemented layer was compared against equivalent TensorFlow models during development to verify the correctness of the implementation.

To evaluate the practical applicability of the implemented layer, a MobileNet inspired architecture was constructed [5]. MobileNet was chosen because depthwise separable convolutions are a fundamental component of the architecture and represent one of the primary motivations for implementing DepthConv2D support within AIfES.

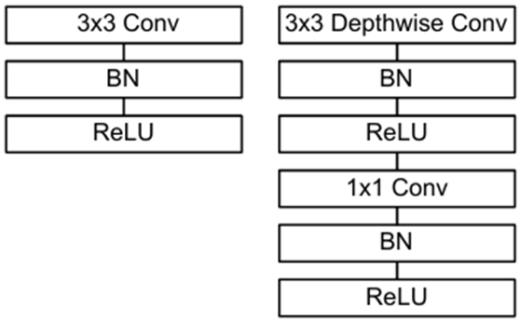


Figure 2: Comparison between a standard convolution block and a depthwise separable convolution block. The latter consists of a depthwise convolution followed by a pointwise convolution.

Figure 2 illustrates the difference between a standard convolution block and the depthwise separable convolution block used throughout the experiments.

This architecture follows the core building block used by MobileNetV1, where a standard convolution is replaced by a depthwise convolution followed by a pointwise convolution. Batch normalization and ReLU activation functions were included after each convolution operation to improve training stability and mirror the original MobileNet design. **By successfully training this architecture within AIfES, the implementation demonstrates that the framework can now support an important class of lightweight computer vision models that were previously unavailable.**

4.2 Training Results

The first evaluation used a custom synthetic dataset designed specifically for validating the interaction between depthwise and pointwise convolutions. The dataset consisted of RGB inspired images containing 3 channels each containing a simple geometric pattern in the form of vertical, horizontal, and diagonal lines. Example samples from both classes are shown in Figure 3.

The dataset was constructed such that the same visual features appeared in both classes but were assigned to different image channels. Consequently, classification could not be performed solely by detecting the presence of a feature. Instead, the network was required to learn relationships between features across channels. This property makes the dataset particularly suitable for evaluating depthwise separable convolutions, since the depthwise stage extracts channel specific features while the pointwise stage combines information across channels.

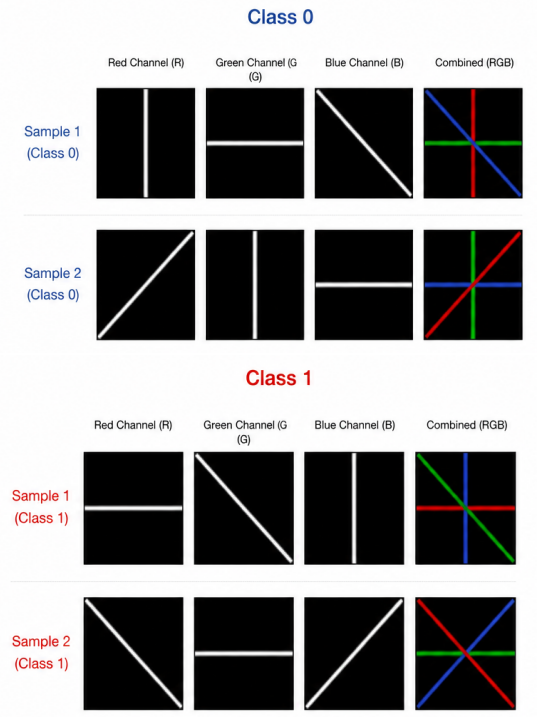


Figure 3: Example images from the synthetic dataset. The two classes contain identical geometric features arranged in different RGB channels. Classification therefore depends on learning cross-channel relationships rather than detecting individual features.

The network was trained for 250 epochs using the Adam optimizer with a learning rate of 0.01. The training dataset consisted of 8 images 4 of each class, and the same holds for the testing dataset. Due to the intentionally simple nature of the dataset, the goal of this experiment was not to evaluate classification performance, but rather to verify that the implemented forward and backward propagation routines enable successful learning.

Figure 4 shows the training loss over time. The loss decreased from an initial value of approximately 0.24 to nearly zero after 40 epochs, indicating successful convergence of the optimization process. This demonstrates that gradients are computed correctly and that the implemented DepthConv2D layer can be trained using the existing AIfES optimization infrastructure.

The trained network correctly classified all training and test samples, achieving 100% accuracy on both datasets. These results provide initial evidence that the implemented layer functions correctly during both inference and training and can successfully learn feature relationships across multiple image channels when combined with a pointwise convolution layer.

4.3 Comparison with TensorFlow

To validate the correctness of the implemented DepthConv2D layer, an equivalent network architecture was implemented and trained in TensorFlow[4]. TensorFlow was selected as a

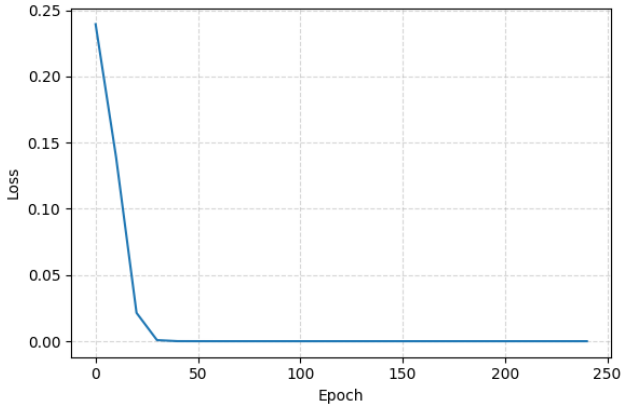


Figure 4: Training loss over epochs for the implemented DepthConv2D layer.

reference because it provides a mature and widely used implementation of depthwise convolution, making it suitable for verifying both the functional correctness and training behavior of the AIfES implementation.

Both models were trained on the same synthetic classification task described previously. Identical training and test datasets were used for both frameworks to ensure a fair comparison.

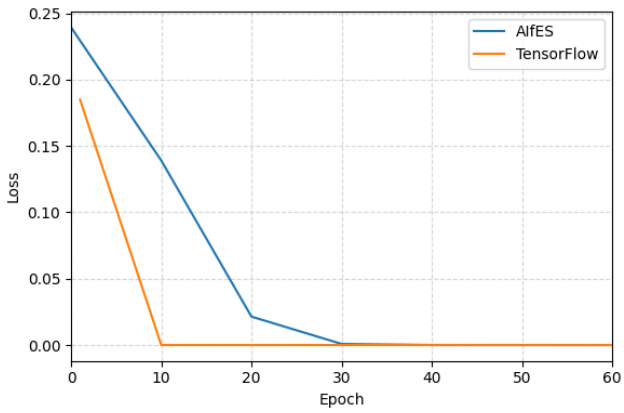


Figure 5: Training loss comparison between the TensorFlow reference implementation and the proposed AIfES DepthConv2D implementation during the first 60 training epochs. Both models converge successfully, although TensorFlow reaches near-zero loss more rapidly.

Figure 5 compares the training loss of the TensorFlow and AIfES implementations during the first 30 training epochs. Both models exhibit stable convergence and successfully minimize the loss. The TensorFlow implementation converges more rapidly, reaching near zero loss after approximately 10 epochs, whereas the AIfES implementation requires approximately 30 to 40 epochs to achieve a comparable loss value. Despite this difference in convergence speed, both implementations ultimately converge to near zero

loss.

The final prediction outputs further confirm the correctness of the implementation. The AIfES model correctly classified all training samples and achieved the expected predictions on the test dataset. Similarly, the TensorFlow model achieved perfect classification on the training set and produced equivalent predictions for the test samples. Minor numerical differences in the prediction values were seen but were also expected due to differences in parameter initialization between the two frameworks.

4.4 Comparison with Standard Convolution

To evaluate the practical benefits of the implemented DepthConv2D layer, the MobileNet inspired architecture was compared with an equivalent architecture using a standard convolution layer. The standard convolution architecture replaced the depthwise convolution and pointwise convolution layers with a single conventional convolution layer while maintaining a similar overall network structure. These are shown in Figure 2. Both networks were trained on the synthetic dataset described in Section 4.1 using identical training procedures and hyperparameters.

Figure 6 compares the training loss of both architectures. The loss curves show that both models successfully learned the classification task and converged to near zero loss. Furthermore, both architectures achieved 100% classification accuracy on the training and test datasets. These results indicate that replacing the standard convolution with a depthwise separable convolution does not negatively affect the network's ability to learn the task.

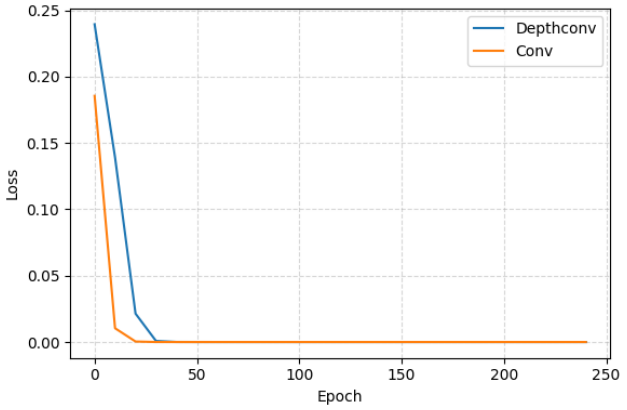


Figure 6: Training loss comparison between the standard convolution architecture and the depthwise separable convolution architecture. Both models successfully converge to near-zero loss and achieve identical classification performance on the synthetic dataset.

While both architectures achieved identical predictive performance, differences were observed in their memory requirements. Table 1 summarizes the measured parameter and training memory usage of both implementations. The depthwise separable convolution architecture required 2300 bytes of parameter memory compared to 2948 bytes for the standard convolution architecture. This corresponds to a reduction of 648 bytes, or approximately 22%.

Table 1: Comparison between standard convolution and depthwise separable convolution.

Architecture	Parameters	Training Memory
Conv2D	2948 B	50108 B
DepthConv + PW Conv	2300 B	60572 B

In contrast, the depthwise separable convolution architecture required more training memory, increasing from 50108 bytes to 60572 bytes. This represents an increase of 10464 bytes, or approximately 21%. The training memory includes intermediate activations, gradients, and optimizer state, whereas the parameter memory consists only of trainable weights and biases.

The observed increase in training memory is primarily a consequence of the small network and dataset used in this evaluation. Under these conditions, the memory required for intermediate training data dominates the overall memory footprint. In larger networks, the reduction in trainable parameters achieved by depthwise separable convolutions becomes increasingly significant, as parameter savings accumulate across multiple layers and contribute to lower memory consumption and computational complexity.

Overall, the results demonstrate that the implemented DepthConv2D layer achieves the same classification performance as a standard convolution layer while reducing parameter memory by approximately 22%. These findings are consistent with the theoretical advantages discussed in Section 2.2 and confirm that the implementation enables more parameter efficient neural network architectures within AIfES.

4.5 CIFAR-10 Case Study

To demonstrate that the implemented DepthConv2D layer can be applied to real world image data, an additional experiment was performed using a subset of the CIFAR-10 dataset[6]. The experiment focused on two classes, namely airplanes and automobiles, and used RGB images with a resolution of 32×32 pixels.

Due to the limited memory available on the Arduino Nano 33 BLE of 256 kB RAM, only a very small subset of the dataset could be used. Even after reducing the network architecture, memory limitations restricted the experiment to only 2 training and 2 test images while using the depthwise separable convolution pipeline. As a result, the dataset was too small to provide a meaningful assessment of classification accuracy or performance.

Due to these limitations, the network would overfit on the selected CIFAR-10 samples as it had such few training samples to learn from. While the experiment does not provide sufficient evidence to evaluate the effectiveness of the resulting model, it does show that the proposed implementation can support end-to-end training on real world image datasets. Future work could try the same experiment but then using a

more powerful microcontroller which allows for the use of a bigger subset or even the whole CIFAR-10 dataset.

5 Responsible Research

This research focuses on extending a machine learning framework rather than developing a new application that directly affects end users. Consequently, the ethical risks associated with the work are relatively limited. Nevertheless, several aspects should be considered.

First, enabling more efficient on-device training can improve privacy by reducing the need to transmit user data to external servers. Data can remain on the device while model updates are performed locally. This reduces the risk of unauthorized data collection and can improve user control over personal information.

Second, increased accessibility of embedded AI may also introduce risks. More capable microcontroller based systems could be deployed in surveillance, monitoring, or data collection applications without adequate transparency. The ethical implications therefore depend largely on how such technology is used in practice.

Reproducibility was an important consideration throughout this project. The implementation was developed as an extension of the open source AIfES framework and evaluated using clearly defined experiments. The correctness of the implementation was verified through manually computed test cases, comparison with TensorFlow, and end-to-end training experiments. All experiments were performed using publicly available software tools and standard hardware platforms, making it possible for other researchers to reproduce the results.

To further improve reproducibility, a public GitHub repository[1] was created containing the complete source code of the DepthConv2D implementation, the experimental setup, and the evaluation code used throughout this work. This allows other researchers and developers to build the modified AIfES framework, repeat the experiments presented in this paper, and verify the reported results independently. By making both the implementation and evaluation pipeline publicly available, this work aims to support future research on on-device learning for microcontrollers and facilitate further development of the AIfES framework.

6 Discussion

The results demonstrate that the implemented DepthConv2D layer produces correct outputs and gradients and integrates successfully into the existing AIfES training framework. Verification against manually computed examples and TensorFlow showed that both inference and training behavior match the expected results.

The experiments also highlight the trade-offs associated with embedded machine learning. While depthwise convolution reduces the number of trainable parameters compared to standard convolution, the memory required for training remains dominated by intermediate activations and gradient storage. Consequently, the benefits of depthwise convolution become more apparent in larger networks.

The CIFAR-10 experiment further illustrates the practical limitations of on-device training. Due to the memory constraints of the Arduino Nano 33 BLE, only a very small subset of the dataset could be used. Although this experiment demonstrated that the implementation functions correctly on real hardware, it does not provide a meaningful evaluation of model accuracy. Instead, its primary purpose was to validate the integration of the new layer within the AIfES training pipeline.

Overall, the results indicate that trainable depthwise convolution can be successfully supported within AIfES and provide a foundation for future lightweight neural network architectures on embedded devices.

7 Conclusions and Future Work

This work investigated the research question: How can the AIfES framework be extended to support trainable depthwise convolution layers for on-device learning on resource-constrained microcontrollers?

The results demonstrate that this goal can be achieved. A new trainable DepthConv2D layer was successfully implemented in C and integrated into the AIfES architecture, enabling both forward propagation and backpropagation within the existing training pipeline. The implementation supports inference and on-device training while remaining compatible with the framework's memory management and optimization infrastructure.

The correctness of the implementation was validated through manually computed test cases, comparison with TensorFlow, and end-to-end training experiments. These evaluations demonstrated that the layer produces correct outputs and gradients and can be trained using existing AIfES optimization algorithms without modification.

The main contribution of this work is the addition of trainable depthwise convolution support to AIfES. Prior to this work, MobileNet inspired architectures could not be trained within the framework. The implemented DepthConv2D layer expands the range of efficient neural network architectures available to embedded machine learning developers and provides a foundation for future work on lightweight on-device learning.

Several opportunities for future work remain. First, the implementation could be optimized further to reduce memory consumption and improve training performance on microcontrollers. Second, support for additional lightweight operators commonly used in modern neural networks could be added to the framework. Finally, future research could investigate the deployment and training of larger MobileNet style architectures on more capable embedded platforms to better evaluate the practical benefits of depthwise convolution in real world TinyML applications.

[1] https://github.com/ZicoKrassenburg/AIfES_depthconv

References

[1] Alexander Bauer et al. Aifes: A framework for on-device training of neural networks on microcontrollers. In *Embedded World Conference*, 2021.

[2] Mike Davies et al. A survey of event-driven and neuromorphic computing for low-power ai. *Proceedings of the IEEE*, 2021.

[3] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:1–21, 2019.

[4] Google. Tensorflow lite for microcontrollers. https://www.tensorflow.org/lite/microcontrollers, 2019.

[5] Andrew G. Howard et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *arXiv preprint arXiv:1704.04861*, 2017.

[6] Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[7] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. In *ACL*, 2019.