Fused-Layer CNNs for Memory-Efficient Inference on Microcontrollers

Mark Deutel[1] **Frank Hannig**[1] **Christopher Mutschler**[2] **Jürgen Teich**[1]
[1]Friedrich-Alexander-Universität Erlangen-Nürnberg
[2]Fraunhofer IIS, Fraunhofer Institute for Integrated Circuits IIS
{mark.deutel, frank.hannig, juergen.teich}@fau.de
christopher.mutschler@iis.fraunhofer.de

Abstract

Convolutional Neural Networks (CNNs) have been established as the dominant approach to computer vision tasks. As a result, efficient inference of CNNs has become a major concern to enable the processing of image data close to where it is generated by camera sensors, most commonly microcontroller units (MCUs). However, major obstacles to deploying CNNs on MCUs are the strict memory and bandwidth constraints that make processing high-resolution images on many MCUs infeasible. In this work, we propose a method to fuse convolutional layers in quantized CNNs, which serves as an additional optimization of the memory requirements of CNNs during inference. By fusing memory-intensive convolutions in the early inverted residual blocks of MobileNetv2-like CNNs, we show that memory requirements during inference can be reduced by up to 54% at the cost of only about a 14% increase in latency and no change in accuracy. As an example, we show that this reduction enables the deployment of image processing pipelines on a Cortex-M7 MCU that supports image resolutions up to 320×320 pixels compared to the 128×128 pixels commonly used in related work.

1 Introduction

Layer fusion, as a means of simplifying the computational graph of convolutional neural networks (CNNs), is widely used to support the efficient deployment of CNNs on resource-constrained microcontroller units (MCUs). In particular, for large input applications such as vision tasks, layer fusion is in many cases even a mandatory optimization required to meet memory and latency constraints on the target device. In this work, we discuss the fusion of convolutional layers in *inverted residual blocks* of fully quantized MobileNetV2-like CNNs [14] as a means to overcome the RAM bottleneck found in early CNN layers. We show that using this optimization, the deployment of vision tasks with input resolutions up to 320×320 pixels becomes feasible within the memory constraints of common Cortex-M7 MCUs. Furthermore, we demonstrate the applicability of our approach to both generic MobileNetV2-like CNNs and MCUNet [8], a CNN specialized for MCU deployment. Finally, we present latency results measured on an IMXRT1862 and describe an algorithm that automatically finds the optimal number of inverted residual blocks to fuse to maximize memory savings while minimizing computational overhead for a CNN to demonstrate the effectiveness of our approach.

Our contribution is threefold. First, we propose a layer fusion method for inverted residual blocks for use on MCUs, which can yield memory savings of up to 54% for image classification tasks. Second, we evaluate our approach on two CNN architectures, a generic MobileNetV2-like CNN and MCUNet [8], and show that it enables the deployment of vision-based tasks with input resolutions of up to 320×320 pixels on Cortex-M7 MCUs that were not feasible in terms of memory requirements in the current state-of-the-art. Third, we analyze and discuss how layer fusion affects the underlying memory allocation scheme.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

The remainder of this paper is structured as follows. First, in Section 2, we give an overview of related work. Second, in Sections 3 and 4, we describe how to fuse the convolutions in inverted residual blocks and the impact it has on memory allocation. Third, in Sections 5 and 6, we present and discuss experimental results.

2 Related Work

Common layer fusion techniques implemented in many deep learning pipelines include folding Batchnorm layers into preceding convolutional or linear layers, and the fusion of ReLU activation functions with linear quantization layers required for fully quantized CNN inference [10]. The advantage of both techniques is that they simplify the computational complexity of CNNs. For example, Batchnorm folding explicitly computes the (relatively) expensive Batchnorm layer during deployment [10], thereby reducing the computational effort required to perform inference. The two previously discussed layer fusion methods have in common that while they reduce the computational complexity of a DNN, they have only little impact on the RAM usage since they do not alleviate the problem of big intermediate feature maps.

To reduce the RAM overhead of DNN inference with layer fusion, Lin et al. [8] propose in-place depth-wise convolutions as part of their MCUNet ecosystem. The authors observed that since depth-wise convolutions do not perform filtering across channels, once a channel is computed, its input activation can be used to store the output activation of another channel, allowing for an in-place computation of the two convolution which as a results reduces the RAM overhead.

In contrast, we propose a layer fusion approach based on the observation that not only depthwise-separable convolutions, but any number of convolutions of any type can be combined and executed together as a single operation by fusing and then rearranging their loop nests. This significantly reduces the total memory required for intermediate feature maps by allowing tiling across spatial dimensions. We use this method to fuse a complete inverted residual block consisting of three convolutions. Compared to [8], which only combines the two convolutional layers of a depthwise-separable convolution, thereby compressing only one intermediate feature map, our approach compresses two feature maps instead. As a result, our approach can achieve a significantly higher compression rate, see Section 5.2.

Except for the fusion of dethwise-separable convolutions [8], the use of layer fusion and tiling for convolutional layers has been discussed mainly as a means to speed up inference on dedicated hardware and dataflow accelerators [6, 1, 12, 9, 3]. Although this reduction in inference time cannot be achieved on MCUs using layer fusion and tiling, we show in this work that the memory savings alone resulting from the approach are still highly interesting because memory is the main bottleneck for DNN inference on MCUs. As a result, using the approach enables processing of high-resolution input CNNs that previously could not be deployed on MCUs due to memory limitations.

3 Fused-Layer Inverted Residual Blocks

We propose the fusion of the convolutions in inverted residual blocks [14]. An inverted residual block consists of three convolutional layers: first, a 1×1 convolution to expand the number of channels; second, a 3×3 depth-wise convolution; and third, a 1×1 convolution to reduce the number of channels so that input and output can be (optionally) added. Compared to regular residual blocks as proposed by the ResNet architecture [5], which have a "wide-narrow-wide" structure with their number of channels, inverted residual blocks follow a "narrow-wide-narrow" approach. This makes them an optimal candidate for layer fusion because combining the three successive convolutions into a single operation can reduce the "wide" and therefore very costly intermediate feature maps. This leaves the inverted residual block with only the "narrow" less costly input and output feature maps.

We apply our proposed layer fusion approach to fully quantized CNNs [10]. As part of the quantization process, we fold Batchnorm layers as well as ReLU layers into their preceding convolutions. This leaves three fully quantized convolutions for each inverted residual block, which we then fuse in a single operation. To fuse the three convolutions, we use a tiling approach over the two spatial dimensions of the intermediate feature maps as described by [1] for FPGAs. The idea is that instead of computing entire feature maps layer by layer, the output feature map is instead divided into smaller tiles that can then be computed separately. This results in a vertical "tile-by-tile" computation method

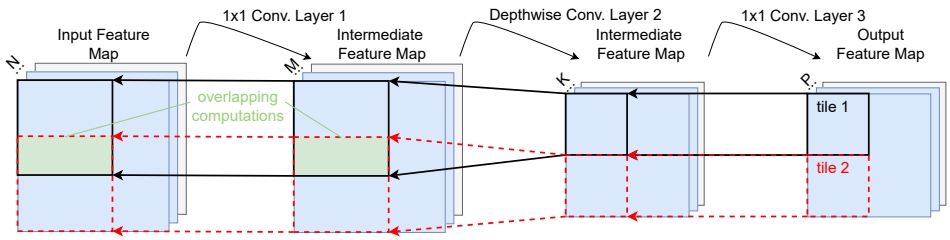


Figure 1: Schematic of an inverted residual block fused using our tiling approach. Each tile in the output feature map can be computed separately, allowing for a significant reduction of the intermediate feature maps to only the areas within the black boxes.

instead of the commonly used horizontal, "layer-by-layer" approach, see Fig. 1. Therefore, only portions of the intermediate outputs of the convolutional layers in the inverted residual blocks need to be computed and held in memory at the same time. Due to this fact and because each tile can be computed separately and sequentially, the same memory for intermediate results can be reused for all tiles, thereby significantly reducing the overall RAM requirements of the inverted residual block.

The latency of a convolutional layer at index $i \in \{1,2,3\}$ of an inverted residual block is assumed to be proportional to the number of MAC operations to be executed and can be computed by $L_c^i = H_o^i \cdot W_o^i \cdot F^i \cdot K_h^i \cdot K_w^i \cdot C^i$, where H_o^i and W_o^i are the height and width of the output feature map, F^i and C^i are the number of filters and channels of the convolution, and K_h^i and K_w^i are the height and width of the kernel. Correspondingly, the latency of a depthwise convolutional layer at index i can be described as $L_{dc}^i = H_o^i \cdot W_o^i \cdot F^i \cdot K_h^i \cdot K_w^i$. For an unfused inverted residual block, this results in a latency as shown in Eq. (1), where each convolution is computed in a "layer-by-layer" fashion.

$$\begin{aligned} L_{\text{unfused}} &= L_c^1 + L_{dc}^2 + L_c^3 = H_o^1 \cdot W_o^1 \cdot F^1 \cdot K_h^1 \cdot K_w^1 \cdot C^1 \\ &\quad + H_o^2 \cdot W_o^2 \cdot F^2 \cdot K_h^2 \cdot K_w^2 + H_o^3 \cdot W_o^3 \cdot F^3 \cdot K_h^3 \cdot K_w^3 \cdot C^3 \end{aligned} \tag{1}$$

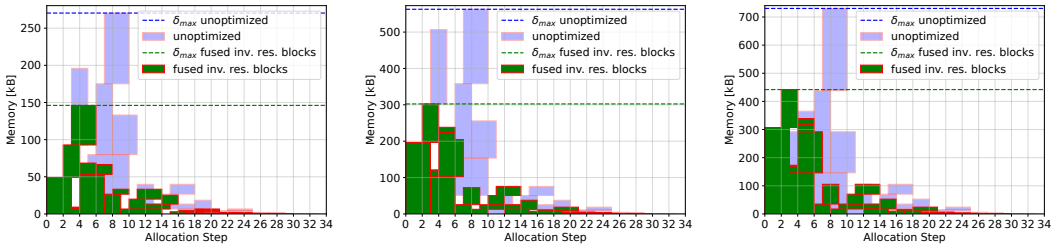
By fusing and then rearranging the two spatial dimensions of the three convolutions, the latency of the fused inverted residual block can be formulated as shown in Eq. (2) where $T_H, T_W \in \mathbb{N}$ are the number of horizontal and vertical tiles.

$$\begin{aligned} L_{\text{fused}} =& T_H \cdot T_W \cdot (H_{to}^1 \cdot W_{to}^1 \cdot F^1 \cdot K_h^1 \cdot K_w^1 \cdot C^1 \\ &+ H_{to}^2 \cdot W_{to}^2 \cdot F^2 \cdot K_h^2 \cdot K_w^2 + H_{to}^3 \cdot W_{to}^3 \cdot F^3 \cdot K_h^3 \cdot K_w^3 \cdot C^3) \end{aligned} \tag{2}$$

For each tile, its width H_{to}^i and height W_{to}^i are calculated based on the strides s_h^i and s_w^i and the kernel height K_h^i and width K_w^i of each layer i. We show this for H_{to}^i in Eq. (3), where the equations for W_{to}^i can be calculated analogously.

$$\begin{aligned} H_{to}^3 &= H_o^3 / T_H \\ H_{to}^2 &= (H_{to}^3 \cdot s_h^2) + (K_h^2 - s_h^2) \\ H_{to}^1 &= (H_{to}^2 \cdot s_h^1) + (K_h^1 - s_h^1) \end{aligned} \tag{3}$$

In Eq. (3), we make the simplified assumption that H_o^3 is divisible by T_H without a remainder. Of course, if this is not the case, a remainder $H_{rto}^3 = H_o^3 - H_{to}^3$ has to be considered additionally. Furthermore, it can be seen that for both H_{to}^2 and H_{to}^3, whenever $s_h^i < K_h^i$ and $s_h^i > 1$, then $T_H \cdot H_{to}^i > H_o^i$. This is because such a convolutional layer requires the tiles to overlap at their boundaries in order to be computed correctly, see Fig. 1 for reference. As a result, the latency L_{fused} may be higher than L_{unfused}. The only feasible configuration of a convolutional layer where $T_H \cdot H_{to}^i = H_o^i$, i.e., where tiling comes at no additional cost in latency, is for $s_h^i = 1$ and $K_h^i = 1$, which is commonly referred to as a 1×1 convolution. In inverted residual blocks, such 1×1 convolutions are used as the first and the last convolution to inflate and then compress the intermediate feature maps across channels. However, the convolutional layer in between the two 1×1 convolutions must be a depthwise convolution with $K_h^2 > 1$ to allow for dependencies in the spatial dimensions, thereby making the use of $s_h^2 = 1$ and $K_h^2 = 1$ infeasible. As a result, while our proposed approach is mathematically equivalent to computing the inverted residual block "layer-by-layer", its use comes at the cost of increased latency due to the need to recompute intermediate results of the second depthwise convolution for multiple tiles.



(a) CIFAR10, 128×128 pixels (b) GTSRB, 256×256 pixels (c) Flowers, 320×320 pixels

Figure 2: The memory allocation schemes for the MobileNetV2-like CNN and the three tested vision datasets. Each tensor is represented by a box, where the height of the box is the size of the tensor in kB and the width of the box is the lifetime of the tensor. The green boxes show the allocation scheme found when using layer fusion (tiling rate 4×4) compared to the light blue boxes showing the scheme when not using layer fusion.

4 Memory Allocation Scheme

Like other inference engines for MCUs, such as tflite-micro [4] or TinyEngine [8], we use an offline memory scheduling approach that simulates memory allocation for a complete inference ahead of time. This results in each tensor being assigned a fixed size and offset in a memory array, often also called a *tensor arena*, allowing for a malloc-free implementation on the MCU. Based on the observations that (a) the same allocation scheme can be used for each inference and (b) intermediate tensors during CNN inference have different lifetimes, usually shorter than the inference itself and in many cases not overlapping, the required size σ_{max} of the tensor arena can be minimized by maximizing the number of tensors that can safely reuse memory since they do not overlap. In the following, we analyze how our layer fusion approach helps the memory scheduler to find a better solution by reducing the size σ_{max} of the tensor arena for all three vision tasks evaluated.

We visualize our memory allocation scheme in Fig. 2 for a MobileNetV2-like CNN and for three vision tasks with different input image resolutions which we also consider in our evaluation, see Section 5. Each box in the three plots represents a tensor. The green boxes show the memory allocation scheme when fusing inverted residual blocks, while the blue boxes show the allocation scheme for the same CNN but without layer fusion. The y-axis of the plots denotes memory (RAM), while the x-axis denotes allocation steps, i.e., the progression of "time" in the simulation. Thus, the height of each box is the size of the corresponding tensor in kilobytes, while its width is the lifetime of the tensor. The logical allocation step counter increases each time a layer (or operand) of the CNN is executed. This means that in each allocation step, either new tensors can be allocated or existing tensors that are no longer needed can be freed.

Looking at the unoptimized blue allocation schemes in Fig. 2, it can be seen that there is a significant imbalance in memory distribution between the first and last layers of the CNNs, since especially the early layers produce feature maps of significant size compared to later layers. Therefore, the main memory bottleneck of most CNNs is found in the first layers. Since layer fusion increases latency, it is sufficient to apply it only to the early, memory-intensive layers, while leaving later layers unchanged. This minimizes the latency increase while maximizing RAM savings.

The process of finding the optimal number of inverted residual blocks in a CNN to fuse, given a number of tiles $T_H, T_W \in \mathbb{N}$, can be automated. Since layer fusion increases latency, the goal is to find a tuple $\mathscr{L}$ of fused inverted residual blocks that minimizes the RAM requirement σ_{max} from the n-tuple $(r_0, r_1, \ldots, r_{n-1})$ of all fusible inverted residual blocks. We describe a simple algorithm that can be used offline as part of the memory allocation, see Algorithm 1. The algorithm first assumes that $\mathscr{L} = (r_0, r_1, \ldots, r_{n-1})$ to query σ_{best} as the minimal RAM requirements that can be achieved by layer fusion. Then, each element r of the reversed tuple $(r_{n-1}, \ldots, r_1, r_0)$ is iteratively removed from $\mathscr{L}$ and the resulting RAM requirements σ_{max} are determined. If $\sigma_{max} \leq \sigma_{best}$, then r can be safely removed from $\mathscr{L}$ without increasing the RAM requirements.

Comparing the optimized green allocation schemes with the unoptimized blue ones for the MobileNetV2-like CNNs in Fig. 2, it can be seen that for all three examples, the optimization

Algorithm 1: Pseudocode example for finding the optimal tuple $\mathscr{L}$ of inverted residual blocks to fuse from the n-tuple $(r_0, r_1, \ldots, r_{n-1})$ of all possible fusible inverted residual blocks of a given CNN and a given tiling width and height $T_H, T_W \in \mathbb{N}$.

$\mathscr{L} \leftarrow (r_0, r_1, \ldots, r_{n-1})$;
$\sigma_{\text{best}} \leftarrow$ DETERMINEMEMORYSCHEDULE$(\mathscr{L}, T_H, T_W)$;
foreach $r \in (r_{n-1}, \ldots, r_1, r_0)$ **do**
    $\sigma_{\text{max}} \leftarrow$ DETERMINEMEMORYSCHEDULE$(\mathscr{L}$.REMOVE$(r), T_H, T_W)$;
    if $\sigma_{max} \leq \sigma_{best}$ **then**
        $\mathscr{L} \leftarrow \mathscr{L}$.REMOVE$(r)$;

problem of minimizing the size σ_{max} of the tensor arena has become easier since the layer fusion resulted in initial tensors being either removed or replaced by smaller ones. Therefore, the memory scheduler could find a better solution, i.e., a smaller σ_{max}.

5 Evaluation

We test the effectiveness of our fused inverted residual blocks on three vision tasks, i.e., CIFAR10 [7], GTSRB [15], and flowers [11], and with three image resolutions, 128×128, 256×256, and 320×320 pixels, see Fig. 3. We trained all three datasets with (a) a generic MobileNetV2-like CNN architecture, consisting of an initial convolutional layer with Batchnorm and ReLU, followed by seven inverted residual blocks and a linear classification head, resulting in about 340k trainable parameters, and (b) MCUNet [8], a CNN architecture commonly used for MCU deployment. Using these two architectures, we achieved a classification accuracy of about 88% on the CIFAR10 task, about 98% on the GTSRB task, and about 84% on the flowers task after 200 epochs of training using baseline models pre-trained on ImageNet [13]. For all three tasks, these results either match or improve the accuracies reported in the respective papers or in benchmarks such as TinyML perf [2]. Furthermore, in our experiments, we applied layer fusion only to those inverted residual blocks needed to overcome the RAM bottleneck which we determined using Algorithm 1. For the MobileNetV2-like CNNs, we fused the first two blocks, and for MCUNet, the first three blocks, leaving all other blocks unchanged.

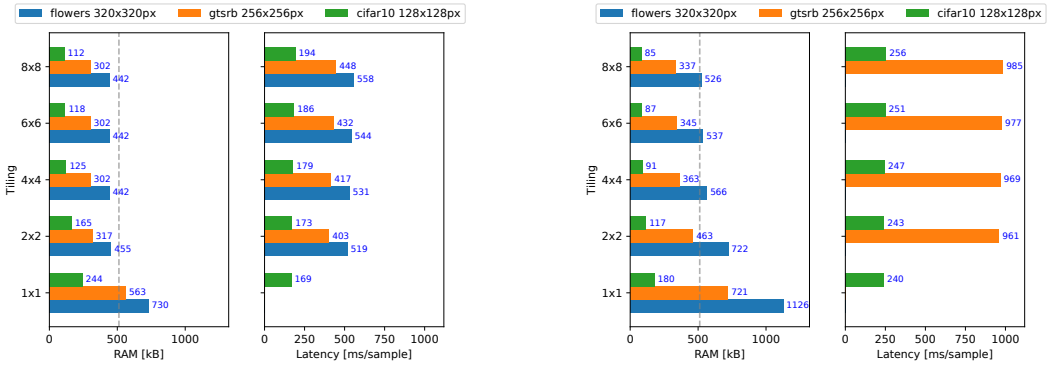
5.1 Evaluation on Vision Tasks

We evaluate the memory utilization for different tiling rates for the generic MobileNetV2-like architecture, see the left plot in Fig. 3a. A tiling rate of 1×1 corresponds to unfused inverted residual blocks and thus denotes the baseline of no layer fusion. It can be seen that for all three image resolutions, even a small tiling rate of 2×2 or 4×4 significantly reduces the memory consumption of the CNN. A small tiling rate, i.e., large tiles, is beneficial because it minimizes the area of tile overlap and thus reduces the increase in latency, since fewer intermediate results need to be recomputed.

Fig. 3 (right) shows the impact of our layer-fusion approach on latency based on the applied tiling rate on the IMXRT1862 Cortex-M7 MCU. Our results show that the rate at which latency increases for different tiling rates is relatively slow. Combined with the observation that fusing only the initial layers with a small tiling rate is usually sufficient, our layer fusion is also feasible from a throughput perspective. For example, for the CIFAR10 task, selecting a tiling rate of 8×8 can reduce RAM requirements by 50% at the cost of a 14.7% increase in latency compared to not using layer fusion. More specifically, for CIFAR10 this means a reduction in processed frames per second (FPS) from 5.9 to 5.1, i.e., a reduction of less than 1 FPS, which is barely noticeable for most vision applications.

5.2 Benchmarking using MCUNet

To verify the flexibility of our layer-fusing approach on other MobileNetV2-based CNN architectures, we tested our method on MCUNet [8] using the version pre-trained on ImageNet [13] provided by the authors online, see Fig. 3b. To this end, we first fine-tuned the architecture for our three vision tasks, before applying our proposed layer fusion technique during deployment. Similar to the experiments discussed in the previous Section, in Fig. 3b, we provide memory results for increasing tiling in the left plot and latency results in the right plot. For MCUNet, we fused the first three inverted residual blocks, which allowed us to overcome the memory bottleneck in the initial layers for all three datasets.



(a) MobileNetV2-like CNN

(b) MCUNet

Figure 3: Memory consumption and latency measured on the IMXRT1862 for the flowers, gtsrb, and CIFAR10 datasets and for different tiling rates for (a) a MobileNetV2-like CNN and (b) MCUNet. A tiling rate of 1×1 is equivalent to no tiling at all and thus represents the baseline. The deviation in latency we observed was always less than 1ms for all measurements. Missing latency results are due to the CNN exceeding the 512 kB memory limit of the IMXRT1862, see the gray dashed lines.

As a result, like for the generic MobileNetV2-like architecture, our layer-fusing approach reduced RAM requirements to about 50%. Comparing the RAM compression factor achieved by our approach for MCUNet's most commonly used input resolution of 128×128 pixels, we achieved a $2.1\times$ reduction in memory requirements for the CIFAR10 task. This compression factor is significantly higher than the $1.6\times$ reduction reported in [8] for their in-place depth-wise convolution approach, demonstrating (a) the greater flexibility of our approach to work for different CNN architectures without having to modify them, and (b) the superior compression factor that can be achieved by fusing inverted residual blocks compared to simply optimizing depth-wise separable convolutions.

6 Discussion

While layer fusion of convolutional layers can result in significant memory savings, as we have shown in Section 5, the technique is limited because it requires explicit co-design between the architecture of the CNN and the runtime used for CNN execution on the MCU. This is because each combination of fused layers must be explicitly implemented by the runtime. Of course, depending on what layer combinations the runtime supports, this can significantly reduce the design space from which a CNN architecture can be constructed. For example, the layer fusion approach proposed by [8] is only designed for their own CNN architecture MCUNet and works only within their ecosystem.

To at least partially circumvent this "co-design problem", we proposed to fuse a well-defined block from MobileNetV2 [14], the inverted residual block. Especially for CNNs targeting MCUs, this architecture is most often chosen as a starting point to derive suitable architectures, and especially the inverted residual block is popular due to its memory-efficient design. Therefore, as we showed in our evaluation in Section 5, our approach does not only work for generic versions of MobilNetV2, see Section 5.1, but can also be effectively applied to other CNN architectures used on the edge, including one of the most popular architectures MCUNet, shown in Section 5.2.

7 Conclusion

In this work, we presented a layer-fusion approach for convolutional layers to reduce the memory requirements of inverted residual blocks in MobileNetV2-like CNNs. As a result, we were able to support high input image resolutions up to 320×320 RGB within the constraints of a Cortex-M7 MCU with no loss of accuracy and only at the cost of a slight increase in latency.

Acknowledgments. This work was supported by the Bavarian Ministry for Economic Affairs, Infrastructure, Transport and Technology through the Center for Analytics-Data-Applications (ADA-Center) within the framework of "BAYERN DIGITAL II".

References

[1] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer CNN accelerators. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[2] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. MLPerf tiny benchmark. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[3] Shail Dave, Aviral Shrivastava, Youngbin Kim, Sasikanth Avancha, and Kyoungwoo Lee. dMazeRunner: Optimizing convolutions on dataflow accelerators. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1544–1548. IEEE, 2020.

[4] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. Tensorflow Lite Micro: Embedded machine learning for TinyML systems. *Machine Learning and Systems*, 3:800–811, 2021.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[6] Christian Heidorn, Frank Hannig, and Jürgen Teich. Design space exploration for layer-parallel execution of convolutional neural networks on CGRAs. In *Proceedings of the 23th International Workshop on Software and Compilers for Embedded Systems*, pages 26–31, 2020.

[7] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[8] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. MCUNet: Tiny deep learning on IoT devices. *Advances in Neural Information Processing Systems*, 33:11711–11722, 2020.

[9] Linyan Mei, Pouya Houshmand, Vikram Jain, Juan Sebastian Piedrahita Giraldo, and Marian Verhelst. ZigZag: Enlarging joint architecture-mapping design space exploration for DNN accelerators. *IEEE Transactions on Computers*, 70(8):1160–1174, 2021.

[10] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

[11] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Proceedings of the Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008.

[12] Wajahat Qadeer, Rehan Hameed, Ofer Shacham, Preethi Venkatesan, Christos Kozyrakis, and Mark A Horowitz. Convolution engine: Balancing efficiency & flexibility in specialized computing. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pages 24–35, 2013.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252, 2015.

[14] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[15] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. Computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32: 323–332, 2012.