DEX: Data Channel Extension for Efficient CNN Inference on Tiny AI Accelerators

Taesik Gong[12]* **Fahim Kawsar**[13] **Chulhong Min**[1]
[1]Nokia Bell Labs [2]UNIST [3]University of Glasgow
taesik.gong@unist.ac.kr
{fahim.kawsar, chulhong.min}@nokia-bell-labs.com

Abstract

Tiny machine learning (TinyML) aims to run ML models on small devices and is increasingly favored for its enhanced privacy, reduced latency, and low cost. Recently, the advent of tiny AI accelerators has revolutionized the TinyML field by significantly enhancing hardware processing power. These accelerators, equipped with multiple parallel processors and dedicated per-processor memory instances, offer substantial performance improvements over traditional microcontroller units (MCUs). However, their limited data memory often necessitates downsampling input images, resulting in accuracy degradation. To address this challenge, we propose Data channel EXtension (*DEX*), a novel approach for efficient CNN execution on tiny AI accelerators. DEX incorporates additional spatial information from original images into input images through patch-wise even sampling and channel-wise stacking, effectively extending data across input channels. By leveraging underutilized processors and data memory for channel extension, DEX facilitates parallel execution without increasing inference latency. Our evaluation with four models and four datasets on tiny AI accelerators demonstrates that this simple idea improves accuracy on average by 3.5%p while keeping the inference latency the same on the AI accelerator. The source code is available at https://github.com/Nokia-Bell-Labs/data-channel-extension.

1 Introduction

Tiny machine learning (TinyML) is an active research field focused on developing and deploying machine learning models on extremely resource-constrained devices, such as microcontroller units (MCUs) and small IoT sensors. Compared to cloud-based AI, TinyML on devices offers benefits in privacy preservation, low latency, and low cost. While research efforts in TinyML, such as model compression techniques [15, 17, 25, 27, 31, 32], have successfully reduced the size of AI models to fit into memory-constrained MCUs, the fundamental limitation in the processing capability of MCUs leads to long inference latency. This limitation hinders the widespread adoption of on-device AI, especially for real-time applications.

Recently, the advent of *tiny AI accelerators* like the Analog Devices MAX78000 [34] and Google Coral Micro [8] has revolutionized the TinyML field by dramatically boosting the model inference speed and leading a new phase of on-device AI. For instance, the MAX78000 AI accelerator [34] achieves 170× faster inference latency compared to an MCU processor (MAX32650 [33]).

To enable such acceleration, these tiny AI accelerators introduce several hardware optimization techniques. They often feature multiple convolutional processors (e.g., 64 processors in MAX78000 [34])

*This work was done entirely while the author was affiliated with Nokia Bell Labs.

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

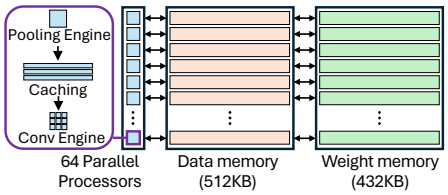


Figure 1: The architecture of a tiny AI accelerator (MAX78000 [34]).

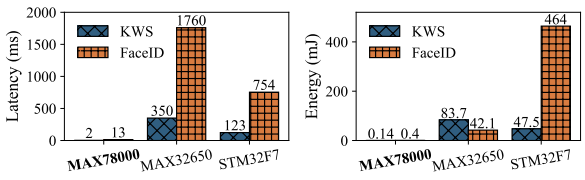


Figure 2: Comparison between an AI accelerator (MAX78000) and MCUs (MAX32650 and STM32F7).

and parallelize per-channel CNN operations across these processors. For further optimization, the memory architecture allows each processor to have a dedicated memory instance, i.e., *per-processor memory instance*. This design enables simultaneous memory access to multiple channels from different processors. While these hardware-level optimizations bring significant performance improvements, we found that they also have several constraints at the expense of the optimizations. First, the per-processor memory architecture highly restricts the supported input image size because the data memory each processor can use for its input/output channels is limited to the capacity of its dedicated memory instance, which is a fraction of the total data memory divided by the number of processors. Consequently, most vision models for these accelerators are designed to support very small images, such as 32×32 pixels. Given that images captured by cameras are often generated with higher resolutions, downsampling is inevitable, leading to accuracy degradation due to information loss from the original image. Second, we found that processors and data memory are underutilized for the input layer due to the per-processor memory architecture; since input images typically have a low number of channels (e.g., RGB three channels), only a limited number of processors tied to memory instances are utilized while the remaining processors remain idle. For instance, on the MAX78000, 61 of 64 processors and per-processor memory instances remain unused in the first layer.

In this work, we propose a novel approach, Data channel EXtension (*DEX*), to overcome these constraints while still benefiting from the acceleration power of tiny AI accelerators. The core idea is to boost accuracy by extending the data channels to incorporate additional image information into unused data memory instances and processors, instead of simple downsampling. Owing to the parallel processing and memory access capabilities of tiny AI accelerators, our method can achieve this accuracy improvement without compromising inference latency. Specifically, DEX involves two procedures: (1) pair-wise even sampling, where pixels from the original image are evenly sampled, and (2) channel-wise stacking, which arranges these samples across multiple channels.

To measure the impact of DEX on accuracy and resource utilization, we conducted experiments on the MAX78000 [34] and MAX78002 [37] tiny AI accelerator platforms. DEX was evaluated on four models, SimpleNet [16], WideNet [16], EfficientNetV2 [48], and MobileNetV2 [45], using four vision datasets: ImageNette [18], Caltech101 [11], Caltech256 [14], and Food101 [2]. Our results show that DEX improves average accuracy by 3.5%p compared to the original model with downsampling and 3.6%p compared to the existing coordinate augmentation approach (CoordConv [29]), without increasing inference latency. Additionally, DEX maximizes data memory and processor utilization, demonstrating its effectiveness in enhancing model performance on resource-constrained devices. In summary, DEX can significantly enhance the performance of neural networks on tiny AI accelerators, leading to more efficient and effective deployment of AI on resource-constrained devices.

2 Preliminary: tiny AI accelerators

The advent of tiny AI accelerators marks a pivotal shift towards on-device AI, greatly enhancing privacy and reducing latency. While a number of tiny-scale AI accelerators have emerged recently, such as Analog Devices MAX78000/MAX78002 [34, 37], Google Coral Micro [8], and GreenWaves GAP-8/GAP-9 [12], only a few are commercially available with access and control over their operations. In this paper, we focus on the MAX78000 [34] and MAX78002 [37] as our primary platforms since they are the most widely used tiny AI accelerator research platforms [1, 6, 13, 39, 40, 43] owing to the disclosed hardware details and open-source tools, enabling in-depth analysis and modification of their operations.

Architecture of tiny AI accelerators. The distinctive characteristic of tiny AI accelerators compared to conventional microcontroller units (MCUs) is *parallel processors* that parallelize per-channel CNN operations across these processors. Figure 1 depicts an abstracted architecture of the MAX78000; MAX78002 has a similar architecture to MAX78000 with increased memory (1.3 MB data and 2 MB weight memory). Further details are in Appendix A.1. It has 64 parallel convolutional processors, each capable of performing specific operations independently. To maximize performance, each processor has a dedicated memory instance, i.e., *per-processor memory instance* that optimizes data transfer with parallel access. For each CNN layer, operations on individual channels are assigned to separate convolutional processors and executed simultaneously, significantly reducing latency typically associated with convolutional algorithms. Each processor has a pooling engine, an input cache, and a convolution engine that can handle 3 by 3 kernels. The CNN accelerator includes 512 KB of data memory and 432 KB of weight storage memory. Within the 512 KB of data memory, an 8 KB per-processor memory instance is allocated to each of the 64 processors. Figure 3 shows the utilization of the processors (Pr_i) for executing CNNs with varying sizes of the input channels. Each processor communicates with a dedicated memory instance for each data channel. For example, given a three-channel image, three parallel processors are utilized in the first layer.

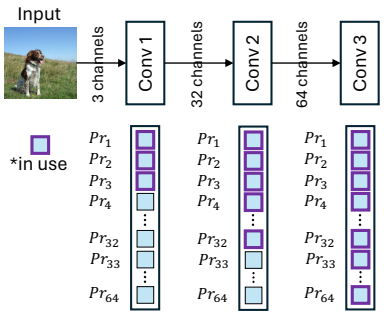


Figure 3: Processor utilization with varying input channels on the AI accelerator.

Performance gain over MCUs. A recent benchmark study [35] demonstrates the remarkable performance gain of the MAX78000 in terms of latency and energy consumption. Figure 2 shows that the MAX78000 significantly outperforms widely-used MCUs (MAX32650 with a Cortex-M4 at 120 MHz [33], and a high-performance MCU, the STM32F7 with a Cortex-M7 at 216 MHz [47]) for face detection (FaceID) and keyword spotting (KWS). For KWS, latency is drastically reduced to only 2.0 ms, compared to 350 ms for the MAX32650 and 123 ms for the STM32F7. Accordingly, energy efficiency of the MAX78000 is also significant; it consumes only 0.40 mJ for FaceID, dramatically less than the 42.1 mJ and 464 mJ required by the MAX32650 and STM32F7, respectively.

3 DEX: Data channel extension for efficient CNN inference on AI accelerators

3.1 Constraints of per-processor memory instances in tiny AI accelerators for images

As mentioned in §2, tiny AI accelerators leverage per-processor memory instances for faster data transfer with parallel access. However, we disclose that this causes several constraints at the expense of rapid data access: (1) low image resolution and (2) underutilized processors and data memory.

Low image resolution due to limited per-processor memory size. MAX78000 [34] has 512 KB data memory which is divided into 64 segments of 8 KB memory instances per processor, each storing the data of each input channel. This memory architecture highly restricts the supported input resolution. For instance, an input image with a shape $3 \times 224 \times 224$ (channel, height, and weight), which is a typical size of ImageNet [9], does not fit the MAX78000 even with Q7 format (one byte for each value), as memory limit for each channel is 8 KB ($224 \times 224 \sim 50$ KB > 8 KB). Thus, the current practice on tiny AI accelerators is to shrink the resolution of input images by downsampling and accordingly, to design small models to process lower-resolution images, e.g., $3 \times 32 \times 32$. However, with this, it loses most of the information of the original image, which might lead to sub-optimal performance.

Underutilized processors and data memory for the input layer. Although per-processor memory instances allow simultaneous memory access from different processors, it also brings inefficiency in data memory and processor utilization, especially in the input layer. Specifically, given an input image I with the number of channels C_I, height H_I, and width W_I (e.g., $3 \times 224 \times 224$) as shown in Figure 4(a), Figure 4(b) illustrates the downsampled image with the number of channels C_I, height H_O, and width W_O (e.g., $3 \times 32 \times 32$), and its data memory usage in the AI accelerator.

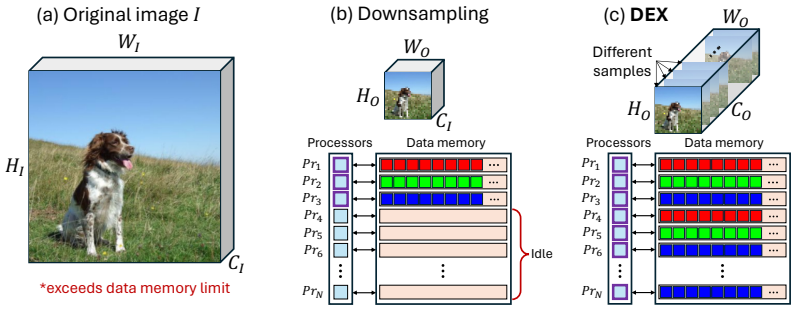


Figure 4: Comparison among different input data. (a) an original image that exceeds the data memory limit of the AI accelerator, (b) a downsampled image that fits the data memory but does not fully utilize parallel processors and data memory, and (c) a DEX-generated image that incorporates more information from original image by extending data across channels with full utilization of parallel processors and data memory instances.

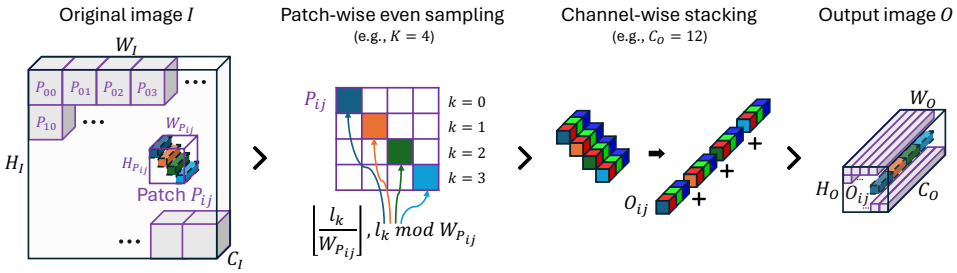


Figure 5: Overview of DEX. DEX divides the original image I into multiple patches. DEX then evenly samples pixels from each patch P_{ij} and constructs an output pixel O_{ij} by stacking samples across channels.

With three RGB channels, channel data are separately stored for each data memory instances for parallel execution. As there is N processors and corresponding data memory instances, it leaves the remaining $N - 3$ processors and data memory instances idle. This provides an opportunity to utilize these idle data memory instances and parallel processors, which we detail in the following section.

3.2 DEX procedure

As aforementioned, we note two key observations: (1) the input image needs to be downsampled due to the limited memory of tiny AI accelerators, which means most of the pixel information cannot be utilized, and (2) there exist idle data memory instances and processors that could process up to N channels in parallel. Although several recent studies have found efficient model architectures on tiny AI accelerators [39, 43], existing studies lack considerations on this inefficiency for input image processing in CNNs (further discussion on related work is in Appendix 5).

Based on our observations, we propose Data channel EXtension (*DEX*) for efficient CNN execution on tiny AI accelerators. The key intuition behind DEX is that we can utilize the remaining data memory to incorporate additional information from the original image into neural networks by extending the input data across channels. By utilizing this additional memory and processors, we can incorporate extra sample information for feature learning without sacrificing latency. Figure 4(c) shows the input data reshaped via DEX, where each channel contains different pixel information from the original image. With DEX extending data across channels (from C_I to C_O), it can fully utilize the data memory and associated parallel processors. Figure 5 shows an overview of the procedure of DEX. Given an input image I with a number of channels C_I, height H_I, and width W_I, DEX generates an output image O with an extended number of channels C_O, height H_O, and width W_O (e.g., $64 \times 32 \times 32$) via patch-wise even sampling and channel-wise stacking.

Patch-wise even sampling. The purpose of patch-wise even sampling is to select samples evenly spaced across the original image while keeping the spatial relationship among pixels. We first define

a *patch* from the original image in which a corresponding output pixel is generated. We denote i-th row and j-th column of patch P_{ij} in I as:

$$P_{ij} = I\left[\left\lfloor i \cdot \frac{H_I}{H_O} \right\rfloor : \left\lfloor (i+1) \cdot \frac{H_I}{H_O} \right\rfloor, \left\lfloor j \cdot \frac{W_I}{W_O} \right\rfloor : \left\lfloor (j+1) \cdot \frac{W_I}{W_O} \right\rfloor\right], \tag{1}$$

where $[:,:]$ refers to a 2-D array slicing operation, specifying the selection of rows and columns sequentially. The number of patches is determined by the resolution of the output image, i.e., $H_O \times W_O$. For each patch P_{ij}, we generate the corresponding output data O_{ij}. This ensures that the spatial relationships among pixels in the input image are preserved in the output, maintaining spatial consistency throughout the process.

The next step is to sample pixels within the patch considering the memory budget. Specifically, we define $K = \lceil \frac{C_O}{C_I} \rceil$ as the number of samples to be selected in each patch. Given the height and width of patch $H_{P_{ij}} = \left\lfloor (i+1) \cdot \frac{H_I}{H_O} \right\rfloor - \left\lfloor i \cdot \frac{H_I}{H_O} \right\rfloor$ and $W_{P_{ij}} = \left\lfloor (i+1) \cdot \frac{W_I}{W_O} \right\rfloor - \left\lfloor i \cdot \frac{W_I}{W_O} \right\rfloor$, the i-th row and j-th column of output O_{ij} can be represented by:

$$O_{ij} = \left\{ P_{ij}\left[\left\lfloor \frac{l_k}{W_{P_{ij}}} \right\rfloor,\ l_k \mod W_{P_{ij}}\right] \mid l_k = k \cdot \left\lfloor \frac{H_{P_{ij}} \cdot W_{P_{ij}} - 1}{K-1} \right\rfloor,\ \text{for } k = 0, 1, \ldots, K-1 \right\}, \tag{2}$$

which means a collection of evenly distributed samples within each patch to encourage diverse information while minimizing the use of localized pixel information. With patch-wise even sampling, selected samples are evenly distributed both across patches and within each patch.

Channel-wise stacking. Channel-wise stacking arranges sampled data across multiple channels and keeps this procedure for all pixels to maintain data integrity. Channel-wise stacking is beneficial as it maintains consistency within each channel, preserving the spatial and contextual relationships of the sampled data. Specifically, after patch-wise even sampling, the samples are stacked across the channel axis in ascending order of the index k, and this is repeated for each O_{ij}. Note that $l_k = 0$ when $K = 1$, and this is identical to traditional downsampling. If $K > \frac{C_O}{C_I}$, it fills up the target channel with P's data until the limit and discards the remaining channels. For instance, when using RGB channels ($C_I = 3$) and if $C_O = 64$ and $K = 22$, it takes only the red channel for $i = 21$ and discards the remaining green and blue channels that exceed the channel limit of 64. Algorithm 1 provides the pseudo-code that describes the procedure of DEX's channel extension algorithm.

Algorithm 1 DEX Channel Extension Algorithm

Input: Source image I in a shape (C_I, H_I, W_I)
Output: Reshaped image O in a shape (C_O, H_O, W_O)

$O \leftarrow \text{zeros}(C_O, H_O, W_O)$
for $i \leftarrow 0$ **to** $H_O - 1$ **do**
  for $j \leftarrow 0$ **to** $W_O - 1$ **do**
    $\text{start\_row}, \text{end\_row} \leftarrow \text{floor}(i \cdot \frac{H_I}{H_O}), \text{floor}((i+1) \cdot \frac{H_I}{H_O})$
    $\text{start\_col}, \text{end\_col} \leftarrow \text{floor}(j \cdot \frac{W_I}{W_O}), \text{floor}((j+1) \cdot \frac{W_I}{W_O})$
    $P_{ij} \leftarrow I[:, \text{start\_row} : \text{end\_row}, \text{start\_col} : \text{end\_col}]$ ▷ Patch P_{ij} of I
    $K \leftarrow \text{ceil}(\frac{C_O}{C_I})$ ▷ Number of samples to be selected in P_{ij}
    for $k \leftarrow 0$ **to** $K - 1$ **do** ▷ Get channels of O_{ij} from P_{ij}
      $H_{P_{ij}} \leftarrow \text{end\_row} - \text{start\_row}$
      $W_{P_{ij}} \leftarrow \text{end\_col} - \text{start\_col}$
      $l_k = k \cdot \text{floor}(\frac{H_{P_{ij}} \cdot W_{P_{ij}} - 1}{K-1})$
      $O[k \cdot C_I : (k+1) \cdot C_I,\ i,\ j] = P_{ij}[:, \text{floor}(\frac{l_k}{W_{P_{ij}}}),\ l_k \mod W_{P_{ij}}]$
return O

3.3 Further analysis on DEX

Understanding how DEX leads to performance improvement. DEX's ability to incorporate additional pixel information from the original image can improve the accuracy of CNNs. The extended channels provide further samples of adjacent areas in the original image, significantly broadening the receptive fields of features in the initial CNN layer. This expansion allows the model to detect more complex and subtle features early in the processing pipeline, which is critical for the nuanced understanding and interpretation of visual data. Specifically, Figure 6 visualizes how the first CNN layer operates with DEX, where $L^1_{kernel\_size}$ and $L^1_{c\_out}$ refer to the kernel size and the output channel size of the first layer, respectively. It illustrates the application of the convolution operation across each enhanced channel (C_O as opposed to C_I), where distinct kernel weights are applied to each channel. This ensures that the additional information is integrated into the output feature maps, thereby enriching the model's feature extraction capabilities. The convolutional layer processes the increased channel input, which is reflected in weight sums that construct output channels.

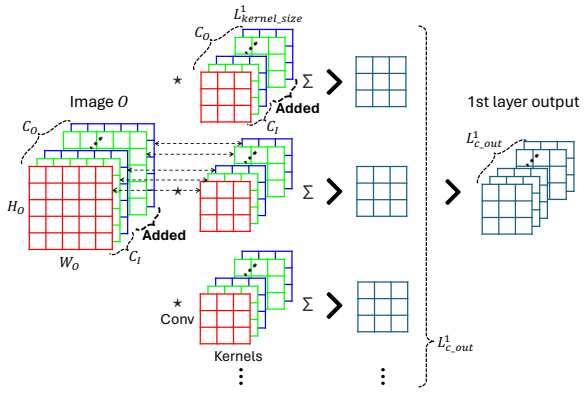


Figure 6: The initial CNN layer's operation with DEX.

Impact of channel extension on the number of parameters. Given the first CNN layer's kernel size $L^1_{kernel\_size}$ and the first layer's channel output size $L^1_{c\_out}$, the number of parameters required for the input layer can be calculated as $O_C \cdot L^1_{kernel\_size} \cdot L^1_{c\_out}$. If O_C is 3, it is the same as the traditional downsampling without our channel extension. Note that this channel extension does not incur additional inference latency on the AI accelerator. We found that the channel extension increases $\sim 3\%$ of the total parameters as we show in our experiment §4.2. The rest of the layers remain the same. In addition to its simplicity, we have several reasons to change the first layer only, which we discuss further in §6.

Utilization of the original image information. With traditional downsampling, the utilization of the original input is $\frac{H_O \cdot W_O}{H_I \cdot W_I}$, but with DEX, this is extended to $\frac{C_O}{C_I} \cdot \frac{H_O \cdot W_O}{H_I \cdot W_I}$. For instance, given a $3 \times 256 \times 256$ input image, a downsampled image $3 \times 32 \times 32$ utilizes only 1.6% of the original information, while with DEX and an output channel size $C_O = 64$, it can utilize 33.3% of the original information. DEX can accommodate all the information when $C_O = C_I \cdot \frac{H_I \cdot W_I}{H_O \cdot W_O}$

Maximum number of output channels. Increasing the number of output channels allows DEX to accommodate the original image information. The number of output channels denoted as O_C, that can be extended without increasing latency on AI accelerators is limited by the number of data memory instances D_N, i.e., $O_C < D_N$. For example, the MAX78000 has 64 data memory instances, allowing it to support up to $O_C = 64$ output channels without affecting inference latency.

4 Evaluation

4.1 Experimental settings

Here we explain experimental settings. Further details are in Appendix A.

On-device testbed. We evaluated DEX on the off-the-shelf MAX78000 feather board [36] and MAX78002 Evaluation Kit [38], which are a development platform for the MAX78000 [34] and MAX78002 [37], respectively, as shown in Figure 9. In this paper, we select these accelerators because they provide open-source tools for thorough analysis and modification of their internal processes, making them the most widely used tiny AI accelerator research platforms [1, 6, 13, 39, 40, 43].

Model training and deployment. In our experiment, we use four models officially supported in the Analog Devices MAX78000/78002 Training framework [20]: SimpleNet [16], WideNet [16],

Table 1: Average classification accuracy (%) and corresponding standard deviations over three runs for each dataset and method. Bold type indicates those of the highest classification accuracy.

Dataset	Method	SimpleNet	WideNet	EfficientNetV2	MobileNetV2	AVG (%)
ImageNette	Downsampling	57.8 ± 1.2	61.8 ± 0.2	51.3 ± 0.5	62.0 ± 0.7	58.2
	CoordConv	58.0 ± 1.1	61.7 ± 0.2	51.9 ± 0.1	61.6 ± 0.3	58.3
	CoordConv (r)	55.4 ± 1.5	61.4 ± 0.2	51.7 ± 1.0	61.2 ± 1.1	57.4
	DEX (ours)	**61.4 ± 0.6**	**65.6 ± 0.6**	**56.8 ± 0.5**	**64.4 ± 0.6**	**62.0**
Caltech101	Downsampling	54.6 ± 2.1	55.8 ± 1.2	38.6 ± 0.9	51.4 ± 1.6	50.1
	CoordConv	53.8 ± 1.6	56.5 ± 0.1	38.7 ± 0.2	49.8 ± 0.5	49.7
	CoordConv (r)	52.7 ± 0.5	56.0 ± 1.7	38.2 ± 1.0	49.7 ± 1.2	49.1
	DEX (ours)	**56.9 ± 1.3**	**61.1 ± 1.4**	**45.9 ± 1.9**	**53.3 ± 1.7**	**54.3**
Caltech256	Downsampling	19.8 ± 0.6	20.8 ± 0.5	14.7 ± 0.4	22.4 ± 1.0	19.4
	CoordConv	19.8 ± 0.5	21.3 ± 0.8	14.8 ± 0.8	22.7 ± 0.8	19.6
	CoordConv (r)	20.0 ± 1.6	20.9 ± 0.6	14.5 ± 0.3	22.7 ± 0.4	19.5
	DEX (ours)	**22.8 ± 0.5**	**22.9 ± 0.9**	**18.3 ± 0.9**	**26.3 ± 0.5**	**22.6**
Food101	Downsampling	16.0 ± 0.4	17.7 ± 0.7	12.1 ± 0.2	22.4 ± 0.6	17.1
	CoordConv	16.1 ± 0.8	17.7 ± 0.3	12.0 ± 0.1	21.7 ± 0.3	16.9
	CoordConv (r)	16.3 ± 0.4	17.3 ± 0.6	12.0 ± 0.6	20.9 ± 0.3	16.6
	DEX (ours)	**18.4 ± 0.4**	**20.9 ± 0.4**	**16.4 ± 0.1**	**23.3 ± 1.1**	**19.8**

EfficientNetV2 [48], and MobileNetV2 [45]. The supported models from the framework were trained via quantization-aware training with 8-bit integers in PyTorch [41]. We follow the official training configuration (details in Appendix A.2). The checkpoints are synthesized as embedded C codes for via the Analog Devices MAX78000/70002 Synthesis framework [19]. SimpleNet and WideNet are developed for MAX78000 while EfficientNetV2 and MobileNetV2 are for MAX78002 considering the size of the models. All models are originally designed to take $3 \times 32 \times 32$ inputs, and DEX increases the number of the channels in the first layer to 64.

Datasets. We evaluated on four common vision datasets: (1) ImageNette [18], a ten-class subset of ImageNet [9] with 9469/3925 train/test samples with the original image shape of $3 \times 350 \times 350$, (2) Caltech101 [11] with 101 objects classes having 6941/1736 train/test samples with the original image shape of $3 \times 300 \times 300$, (3) Caltech256 [14] with 256 objects classes having 23824/5956 train/test samples with the original image shape of $3 \times 300 \times 300$, and (4) Food101 [2] with 101 food categories with 75750/25250 train/test samples with the original image shape of $3 \times 512 \times 512$.

Baselines. For baselines, we compare with the Downsampling method which is a straightforward way to reduce the size of the input under memory-constrained devices. It downsamples the input image to $3 \times 32 \times 32$. In addition, we compare DEX with CoordConv [29] which pointed out the limitation of traditional CNNs that relied on RGB images for the coordinate transformation problem and introduced the augmentation of i and j coordinates, which improved object detection efficiency by using two extra channels. The authors of CoordConv also introduced the third channel for an r coordinate, where $r = \sqrt{(i - h/2)^2 + (j - w/2)^2}$, which they found effective in some experiments.

4.2 Result

Overall accuracy. Table 1 shows the overall accuracy for four different datasets with the baselines and DEX. As shown, extending data channels to utilize additional input information improves accuracy in DEX. Specifically, DEX achieved 3.5%p higher accuracy compared to downsampling and 3.6% higher accuracy compared to CoordConv across datasets. CoordConv shows lower accuracy compared with downsampling (0.1%p degradation on average), showing they are not very effective solutions. This finding aligns with previous results indicating that CoordConv is useful for specific tasks such as object detection, where coordinate information is important [29]. We found CoordConv (r) has a similar pattern to CoordConv. Overall, DEX's accuracy improvement shows the effectiveness of using extra information from the original image for feature learning.

Resource usage. Table 2 compares the resource usage of the baseline and DEX. First, we found that, although DEX extends the number of channels in the first CNN layer to 64, its impact on the

Table 2: Model size (Size), utilization of the original image information (InfoRatio), accelerator's processor utilization for the first layer (ProcUtil), and inference latency on the accelerator (Latency) for different models and methods averaged over three runs.

Model	Method	InputChan	Size (KB)	InfoRatio (×)	ProcUtil (%)	Latency (μs)
SimpleNet	Downsampling	3	162.6	1.0	4.7	2592 ± 1
	CoordConv	5	162.9	1.0	7.8	2592 ± 2
	CoordConv (r)	6	163.0	1.0	9.4	2592 ± 2
	DEX (ours)	64	171.2	21.3	100.0	2591 ± 1
WideNet	Downsampling	3	306.4	1.0	4.7	3820 ± 1
	CoordConv	5	306.9	1.0	7.8	3820 ± 0
	CoordConv (r)	6	307.1	1.0	9.4	3819 ± 1
	DEX (ours)	64	319.3	21.3	100.0	3818 ± 1
EfficientNetV2	Downsampling	3	742.4	1.0	4.7	11688 ± 2
	CoordConv	5	743.0	1.0	7.8	11685 ± 3
	CoordConv (r)	6	743.2	1.0	9.4	11689 ± 1
	DEX (ours)	64	759.6	21.3	100.0	11690 ± 2
MobileNetV2	Downsampling	3	1317.8	1.0	4.7	3553 ± 4
	CoordConv	5	1318.2	1.0	7.8	3554 ± 1
	CoordConv (r)	6	1318.4	1.0	9.4	3554 ± 2
	DEX (ours)	64	1330.7	21.3	100.0	3552 ± 3

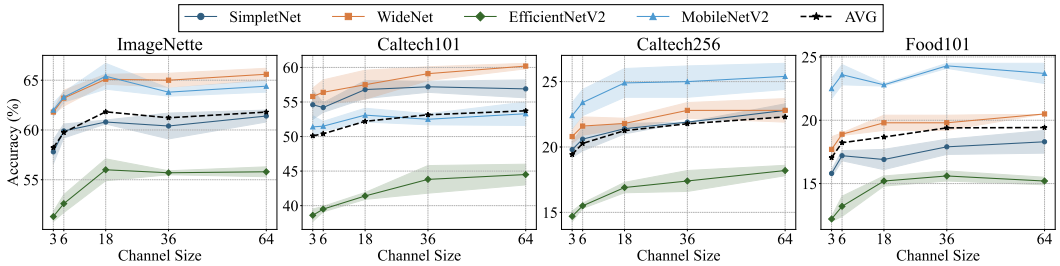


Figure 7: Accuracy of DEX varying the channel size. The shaded areas are standard deviations.

model size is negligible (an average increment of 3.2% compared to no channel extension). DEX utilizes 21.3× more image information compared to downsampling, which is the primary reason for the accuracy improvement. As expected, DEX does not increase on-device inference latency, even though it maximally utilizes the processors on the AI accelerators for information processing. This result is consistent across the four datasets, as all the models are designed to take the same input size in MAX78000 and MAX78002.

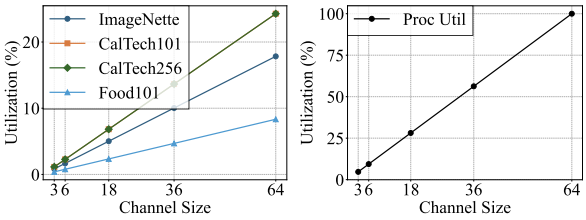
Accuracy according to the channel size. We varied the size of the channels from 3 (downsampling) to 6, 18, 36, and 64 with DEX to understand the impact of the channel size in terms of accuracy. Figure 7 shows the accuracy variation according to the channel size across the four datasets. As shown, it seems that a higher number of channels increases accuracy in general. This means that selecting the highest channel size supported in AI accelerators might be an effective strategy in practice, considering that it does not incur the latency increase. Still, there are some cases where the accuracy of the highest channel size (64) is not the best among them. This means there might be an optimal number of channels tailored to a specific dataset and model architecture, which might be found in the model development process.

Resource usage according to the channel size. We also measure resource usage varying the channel size. First, we measured the model size and inference latency as shown in Table 3. The model size increment is negligible and inference latency remains the same across different numbers of channels. The model size and inference latency are the same for the four datasets as all the models are designed to take the same input size in MAX78000 and MAX78002. Second, we measure the information utilization from the original image and processor utilization in the AI accelerators (Figure 8).

Table 3: Model size (Size) with relative increment (%) compared to the three channels and average inference latency on the accelerator (Latency) with standard deviations over three runs, varying the channel size.

	Model	Chan = 3	Chan = 6	Chan = 18	Chan = 36	Chan = 64
Size (KB)	SimpleNet	162.6	163.0 (+0.3%)	164.7 (+1.3%)	167.3 (+2.9%)	171.2 (+5.3%)
	WideNet	306.4	307.1 (+0.2%)	309.6 (+1.0%)	313.4 (+2.3%)	319.3 (+4.2%)
	EfficientNetV2	742.4	743.2 (+0.1%)	746.6 (+0.6%)	751.7 (+1.3%)	759.6 (+2.3%)
	MobileNetV2	1317.8	1318.4 (+0.0%)	1321.0 (+0.2%)	1324.8 (+0.5%)	1330.7 (+1.0%)
Latency (μs)	SimpleNet	2592 ± 1	2592 ± 2	2591 ± 1	2590 ± 1	2591 ± 1
	WideNet	3820 ± 1	3820 ± 2	3825 ± 1	3819 ± 3	3818 ± 1
	EfficientNetV2	11688 ± 2	11691 ± 2	11692 ± 3	11691 ± 0	11690 ± 2
	MobileNetV2	3553 ± 4	3553 ± 1	3552 ± 1	3554 ± 0	3552 ± 3

The utilization of the original image information depends on the size of the original data size, which grows linearly according to the channel size. We found a correlation between information utilization rate and accuracy improvement. For example, Caltech101 and Caltech256 had utilization rates of 24.3%, improving accuracy by 4.2%p and 3.2%p, respectively, while Food101 had an 8.3% utilization rate with a 2.7%p accuracy improvement. The processor utilization linearly increases until 100% with 64 channels size, which is the number of parallel processing units in the evaluated platforms.



(a) Information utilization. (b) Processor utilization.

Figure 8: Resource usage varying the channel size.

Comparison of alternative data extension strategies in DEX. To understand the effectiveness of our patch-wise even sampling and channel-wise stacking, we compared DEX with other possible data channel extension strategies. We compared with four strategies: repeating the same downsampled image across the channels (Repetition), generating slightly different images through rotation (Rotation), dividing the original image into multiple tiles and stacking those tiles across channels (Tile), patch-wise sequential sampling (Patch-wise seq.) that samples pixels sequentially within a patch, and patch-wise random sampling (Patch-wise rand.) that randomly samples within a patch. Further implementation details are in Appendix A.5. In this experiment, we used SimpleNet and evaluated it on ImageNette. Table 4 shows the results. Repetition does not improve accuracy over downsampling, indicating that merely increasing the number of kernels does not lead to performance gains. Rotation shows a slight decrease in accuracy compared to Repetition, which suggests that slight changes through rotation do not enhance performance. Interestingly, Tile shows low accuracy, demonstrating the importance of having a complete view of the original image in each channel, rather than focusing on specific regions. Both Patch-wise sequential and Patch-wise random samplings show lower accuracy than DEX's patch-wise even sampling, highlighting the importance of even sampling for better performance.

Table 4: Comparison of data extension strategies.

Method	InputChan	InfoRatio (×)	Accuracy
Downsampling	3	1.0	57.8 ± 1.2
Repetition	64	1.0	56.3 ± 0.8
Rotation	64	1.0	55.4 ± 0.7
Tile per channel	64	21.3	39.4 ± 0.7
Patch-wise seq.	64	21.3	61.0 ± 1.5
Patch-wise rand.	64	21.3	60.4 ± 1.0
DEX	64	21.3	**61.4 ± 0.6**

5 Related work

TinyML. Tiny Machine Learning (TinyML) is an emerging field that focuses on adapting machine learning techniques for highly resource-constrained devices, such as microcontroller units (MCUs). These devices often come with limited memory, typically hundreds of kilobytes of SRAM. Research in this area has mostly concentrated on reducing model size through various compression techniques, such as model pruning [15, 17, 25, 27, 31, 32], model quantization [7, 15, 42, 44, 49, 53, 54], and neural architecture search (NAS) [4, 5, 10, 24]. In addition, several studies have explored the

efficient utilization of memory resources (e.g., SRAM). Examples include optimizing on-device training processes [23, 28] and designing memory-efficient neural architectures [26, 52]. Unlike these approaches that primarily target MCUs, our research utilizes the distinctive architecture of tiny AI accelerators to enhance both memory efficiency and overall performance.

Tiny AI accelerators. Several studies have leveraged tiny AI accelerators for small-scale on-device AI applications. For instance, TinyissimoYOLO [39] offers a quantized, memory-efficient, and ultra-lightweight object detection network, showcasing its effectiveness on the MAX78000 platform. Additionally, KP2DTiny [43] introduces a quantized neural keypoint detector and descriptor specifically optimized for MAX78000 and Coral AI accelerators. Moreover, Synergy represents a multi-device collaborative inference platform across wearables equipped with tiny AI accelerators [13]. Another line of studies utilized tiny AI accelerators in battery-free or intermittent computing scenarios [1, 6]. Traditionally, hardware accelerators on low-power AI platforms were capable of only one-shot atomic executions of a neural network inference without intermediate result backups. A study proposed a toolchain to address this that allows neural networks to execute intermittently on the MAX78000 platform [6]. To the best of our knowledge, there has been no work that manipulates data and models to efficiently utilize computing resources considering the unique architecture of tiny AI accelerators.

Image channel extension in CNNs. Several studies have explored augmenting images with additional information to construct multi-channel inputs for Convolutional Neural Networks (CNNs). Liu et al. proposed a multi-modality image fusion approach, combining visible, mid-wave infrared, and motion images for enhanced object detection [30], while Wang et al. presented depth-aware CNN for image segmentation [50]. These approaches require extra sensing channels to acquire data, such as infrared cameras and depth cameras. Similarly, other research has incorporated location data to improve performance for segmentation [51] and object detection tasks [29]. For instance, CoordConv [29] pointed out the limitation of traditional CNNs that relied solely on RGB images for the coordinate transformation problem and introduced the augmentation of i and j coordinates, which improved object detection efficiency. However, these methodologies often necessitate additional sensor modalities or are tailored for specific applications such as object detection, which restricts their general use. Nevertheless, adapting findings from those studies within DEX could be an interesting future direction.

6 Discussion and conclusion

We introduced DEX, a novel method to enhance CNN efficiency on tiny AI accelerators by augmenting input data across unused memory. Evaluations on four image datasets and models showed that DEX improves accuracy without increasing inference latency. This method maximizes the processing and memory capabilities of tiny AI accelerators, making it a promising solution for efficient AI model execution on resource-constrained devices.

Limitations and potential societal impacts. We modified only the initial CNN layer due to simplicity, effectiveness, and memory constraints. The first layer, representing image data in three channels (RGB), has the most unused processors after initial data assignment. Extending channels at the first layer significantly increases data utilization with minimal impact on model size. This approach aligns with the design of weight memory in tiny AI accelerators, which maximizes model capacity by collective use across processors. We think DEX might be less effective in certain tasks where incorporating more pixel information is not beneficial. In those cases, alternative data extension strategies might be used instead of patch-wise even sampling to utilize the additional channel budget. While our focus was on small models supported by the MAX78000/MAX78002 platforms, evaluating larger models could be valuable, given rapid AI hardware advancements. Regarding societal impact, leveraging additional processors and memory to improve accuracy might increase carbon emissions [46], highlighting the need to balance accuracy improvements with environmental sustainability.

References

[1] Abu Bakar, Rishabh Goel, Jasper de Winkel, Jason Huang, Saad Ahmed, Bashima Islam, Przemysław Pawełczak, Kasım Sinan Yıldırım, and Josiah Hester. Protean: An energy-efficient and heterogeneous platform for adaptive and hardware-accelerated battery-free computing. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, pages 207–221, 2022.

[2] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.

[3] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

[5] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[6] Luca Caronti, Khakim Akhunov, Matteo Nardello, Kasım Sinan Yıldırım, and Davide Brunelli. Fine-grained hardware acceleration for efficient batteryless intermittent inference on the edge. *ACM Transactions on Embedded Computing Systems*, 22(5):1–19, 2023.

[7] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[8] Google Coral Micro. https://coral.ai/products/dev-board-micro/. Accessed: 20 May. 2024.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[10] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough. Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. *Advances in Neural Information Processing Systems*, 32, 2019.

[11] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Pattern Recognition Workshop*, 2004.

[12] Greenwaves Technology. https://greenwaves-technologies.com/low-power-processor/. Accessed: 20 May. 2024.

[13] Taesik Gong, Si Young Jang, Utku Günay Acer, Fahim Kawsar, and Chulhong Min. Collaborative inference via dynamic composition of tiny ai accelerators on mcus. *arXiv preprint arXiv:2401.08637*, 2023.

[14] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[15] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

[16] Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. *arXiv preprint arXiv:1608.06037*, 2016.

[17] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[18] Jeremy Howard. Imagenette. https://github.com/fastai/imagenette/. Accessed: 20 May. 2024.

[19] Analog Devices Inc. Ai8x synthesis repository. https://github.com/analogdevicesinc/ai8x-synthesis, 2024. Accessed: 20 May. 2024.

[20] Analog Devices Inc. Ai8x training repository. https://github.com/analogdevicesinc/ai8x-training, 2024. Accessed: 20 May. 2024.

[21] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[22] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

[23] Young D Kwon, Rui Li, Stylianos I Venieris, Jagmohan Chauhan, Nicholas D Lane, and Cecilia Mascolo. Tinytrain: Deep neural network training at the extreme edge. *arXiv preprint arXiv:2307.09988*, 2023.

[24] Edgar Liberis, Łukasz Dudziak, and Nicholas D Lane. μnas: Constrained neural architecture search for microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, pages 70–79, 2021.

[25] Edgar Liberis and Nicholas D Lane. Differentiable neural network pruning to enable smart applications on microcontrollers. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(4):1–19, 2023.

[26] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. *arXiv preprint arXiv:2110.15352*, 2021.

[27] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. *Advances in neural information processing systems*, 30, 2017.

[28] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256kb memory. *Advances in Neural Information Processing Systems*, 35:22941–22954, 2022.

[29] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *Advances in neural information processing systems*, 31, 2018.

[30] Shuo Liu and Zheng Liu. Multi-channel cnn-based object detection for enhanced situation awareness. *arXiv preprint arXiv:1712.00075*, 2017.

[31] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3296–3305, 2019.

[32] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

[33] Analog MAX32650. https://www.analog.com/en/products/max32650.html. Accessed: 20 May. 2024.

[34] Analog MAX78000. https://www.analog.com/en/products/max78000.html. Accessed: 20 May. 2024.

[35] Cutting the AI Power Cord: Technology to Enable True Edge Inference. https://cms.tinyml.org/wp-content/uploads/talks2020/tinyML_Talks_Kris_Ardis_and_Robert_Muchsel_-201027.pdf. Accessed: 20 May. 2024.

[36] Analog MAX78000FTHR. https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/max78000fthr.html. Accessed: 20 May. 2024.

[37] Analog MAX78002. https://www.analog.com/en/products/max78002.html. Accessed: 20 May. 2024.

[38] Analog MAX78002EVKIT. https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/max78002evkit.html. Accessed: 20 May. 2024.

[39] Julian Moosmann, Marco Giordano, Christian Vogt, and Michele Magno. Tinyissimoyolo: A quantized, low-memory footprint, tinyml object detection network for low power microcontrollers. In *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–5. IEEE, 2023.

[40] Arthur Moss, Hyunjong Lee, Lei Xun, Chulhong Min, Fahim Kawsar, and Alessandro Montanari. Ultra-low power dnn accelerators for iot: Resource characterization of the max78000. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, pages 934–940, 2022.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[42] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[43] Thomas Rüegg, Marco Giordano, and Michele Magno. Kp2dtiny: Quantized neural keypoint detection and description on the edge. In *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–5. IEEE, 2023.

[44] Manuele Rusci, Alessandro Capotondi, and Luca Benini. Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers. *Proceedings of Machine Learning and Systems*, 2:326–335, 2020.

[45] 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.

[46] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green ai. *Commun. ACM*, 63(12):54–63, nov 2020.

[47] STM32F7 Series. https://www.st.com/en/microcontrollers-microprocessors/stm32f7-series.html. Accessed: 20 May. 2024.

[48] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pages 10096–10106. PMLR, 2021.

[49] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8612–8620, 2019.

[50] Weiyue Wang and Ulrich Neumann. Depth-aware cnn for rgb-d segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 135–150, 2018.

[51] Zhenyi Wang and Olga Veksler. Location augmentation for cnn. *arXiv preprint arXiv:1807.07044*, 2018.

[52] Hong-Sheng Zheng, Yu-Yuan Liu, Chen-Fong Hsu, and Tsung Tai Yeh. Streamnet: Memory-efficient streaming tiny deep learning inference on the microcontroller. *Advances in Neural Information Processing Systems*, 36, 2024.

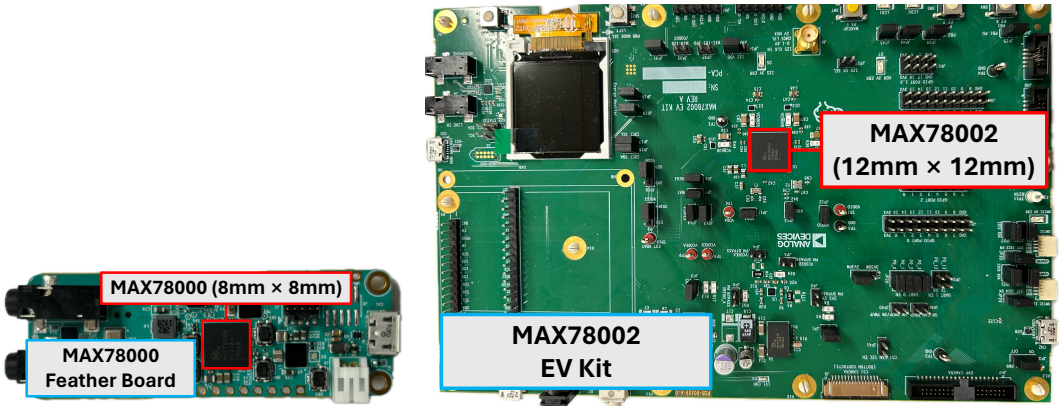
[53] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[54] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. In *International Conference on Learning Representations*, 2016.

A Experimental details

For all experiments conducted in the paper, we used three different random seeds (0, 1, 2) and reported the average accuracy with standard deviations.

A.1 Tiny AI accelerator platforms



(a) MAX78000 Feather Board [36]. (b) MAX78002 Evaluation (EV) Kit [38].

Figure 9: Two tiny AI accelerator development platforms used in our work. Note that although the development platform is bulky, the actual size of the accelerators is tiny (e.g., 8mm×8mm for MAX78000). All data storing and model inference is done only in the AI Accelerator part (MAX78000 and MAX78002).

Table 5: Comparison of MAX78000 and MAX78002.

Component	**MAX78000** [34]	**MAX78002** [37]
MCU Processor	Arm Cortex-M4 (100 MHz), RISC-V	Arm Cortex-M4 (120 MHz), RISC-V
Flash Memory	512 KB	2.5 MB
SRAM	128 KB	384 KB
CNN Processor	64 parallel CNN processors	64 parallel CNN processors
Data Memory	512 KB	1.3 MB
Weight Memory	432 KB	2 MB
Bias Memory	2 KB	8 KB

In this paper, we focus on the MAX78000 [34] and MAX78002 [37] as our primary platforms since they are the most popular research platforms [1, 6, 13, 39, 40, 43] owing to the disclosed hardware details and open-source tools, enabling in-depth analysis and modification of their operations. Figure 9 shows our testbed. Note that all operations required for the model inference are done under the AI accelerator part (highlighted with the red boxes), while the entire boards are bigger for development purposes. For on-device deployment and measurement, we used MAX78000 for SimpleNet and WideNet and MAX78002 [37] for EfficientNetV2 and MobileNetV2, following the memory requirement. For the sake of explanation, we assumed each processor is mapped to one memory instance in this paper, although MAX78000/MAX78002 group four data memory instances together to communicate with four processors in reality. Our experiments were conducted on the actual implementations. Table 5 compares MAX78000 and MAX78002.

A.2 Model training

We followed the official model training code for MAX78000 and MAX78002 platforms [20]. Here, we detail the hyperparameters used in the official training code. We trained models with NVIDIA A40 GPUs.

For all models, quantization-aware training is conducted with support for batch normalization after convolutional layers through batch normalization fusing [21]. This fusing operation integrates the effects of batch normalization directly into the parameters of the preceding convolutional layer by adjusting the weights and bias values. Consequently, after the fusing/folding process, the network no longer contains any batch normalization layers. Instead, the effects of batch normalization are reflected in the modified weights and biases of the preceding convolutional layers.

SimpleNet. SimpleNet [16] was trained for 300 epochs using the Adam optimizer [22] with an initial learning rate of 0.001 and a batch size of 32. A multi-step learning rate scheduler was used with milestones set at epochs 100, 150, and 200, and a multiplicative factor of learning rate decay value of 0.25. Quantization-aware training (QAT) was introduced starting at epoch 240. During QAT, a shift quantile of 0.985 was applied to manage activation ranges. The weight precision was primarily set to 2 bits. However, exceptions were made for certain layers: the 1st convolutional layer utilized 8-bit weights, while the 2nd, 11th, 12th, 13th, and 14th convolutional layers used 4-bit weights.

WideNet. WideNet [16] was trained for 300 epochs using the Adam optimizer [22] with an initial learning rate of 0.001 and a batch size of 100. A multi-step learning rate scheduler was used with milestones set at epochs 100, 150, and 200, and a multiplicative factor of learning rate decay value of 0.25. Quantization-aware training (QAT) was introduced starting at epoch 240. During QAT, a shift quantile of 0.985 was applied to manage activation ranges. The weight precision was primarily set to 2 bits. However, exceptions were made for certain layers: the 1st convolutional layer utilized 8-bit weights, while the 2nd, 11th, 12th, 13th, and 14th convolutional layers used 4-bit weights.

EfficientNetV2. EfficientNetV2 [48] was trained for 300 epochs using the Adam optimizer [22] with an initial learning rate of 0.001 and a batch size of 100. A multi-step learning rate scheduler was used with milestones set at epochs 50, 100, 150, 200, and 250 and a multiplicative factor of learning rate decay value of 0.5. Quantization-aware training (QAT) was introduced starting at epoch 210. During QAT, a shift quantile of 0.995 was applied to manage activation ranges. The weight precision was primarily set to 8 bits.

MobileNetV2. MobileNetV2 [45] was trained for 300 epochs using the stochastic gradient descent optimizer (SGD) [3] with an initial learning rate of 0.1 and a batch size of 128. A multi-step learning rate scheduler was used with milestones set at epochs 100, 150, 175, and 250 and a multiplicative factor of learning rate decay value of 0.235. Quantization-aware training (QAT) was introduced starting at epoch 200. During QAT, a shift quantile of 1.0 was applied to manage activation ranges. The weight precision was primarily set to 8 bits.

A.3 Datasets

ImageNette. Imagenette [18] is a smaller, more manageable subset of ImageNet [9], containing 10 classes. These classes include tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, and parachute. ImageNette has 9469/3925 train/test samples with the original image shape of $3 \times 350 \times 350$. All images were normalized with the ImageNet mean (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225), and then converted to Q7 format (one byte per data) to support on-device inference with the tiny AI accelerator platforms (MAX78000 and MAX78002).

Caltech101. Caltech101 [11] is a dataset composed of images representing objects from 101 different categories, in addition to a background clutter category. Each image features a single object and is labeled accordingly. The number of images per category ranges from approximately 40 to 800, resulting in a total of around 8677 images. Caltech101 has 6941/1736 train/test samples and the original image shape of $3 \times 300 \times 300$. All images were normalized with the ImageNet mean (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225), and then converted to Q7 format (one byte per data) to support on-device inference with the tiny AI accelerator platforms (MAX78000 and MAX78002).

Caltech256. Caltech256 [14] built upon its previous version, Caltech101, offering enhancements such as larger category sizes, additional and more extensive clutter categories, and increased overall difficulty. The dataset contains 29780 images across 256 classes after removing the clutter class. Caltech256 has 23824/5956 train/test samples with the original image shape of $3 \times 300 \times 300$. All images were normalized with the ImageNet mean (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225), and then converted to Q7 format (one byte per data) to support on-device inference with the tiny AI accelerator platforms (MAX78000 and MAX78002).

Food101. Food101 [2] includes 101 food categories, each with 750 images for training and 250 images for testing, which is a total of 101000 images. The original images were rescaled to have a maximum side length of 512 pixels. This dataset has 75750/25250 train/test samples and the original image with the shape of $3 \times 512 \times 512$. All images were normalized with the ImageNet mean (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225), and then converted to Q7 format (one byte per data) to support on-device inference with the tiny AI accelerator platforms (MAX78000 and MAX78002).

A.4 Baseline details

Downsampling. Downsampling is a straightforward method that collects samples evenly distributed across the original image. This approach is equivalent to the case when the number of channels is equal to three in DEX.

CoordConv. CoordConv [29] pointed out the limitation of traditional CNNs that relied solely on RGB images for the coordinate transformation problem and introduced the augmentation of i and j coordinates, which improved object detection efficiency. We referred to the Pytorch implementation of CoordConv[2] for implementing this baseline.

CoordConv (with r). The authors of CoordConv also introduced the third channel for an r coordinate, where $r = \sqrt{(i - h/2)^2 + (j - w/2)^2}$, which they found effective in some experiments. Similar to CoordConv, we referred to the Pytorch implementation of CoordConv for implementing this baseline.

A.5 Alternative data channel extension methods' details

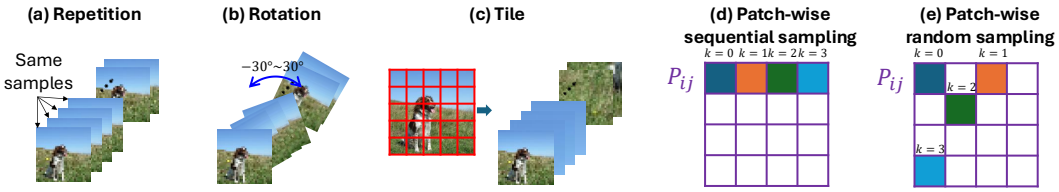


Figure 10: Visulaization of four alternative data extension methods.

Repetition. Repetition (Figure 10(a)) repeats the same downsampled image across the channels until it reaches the maximum possible number of input channels, which is the same as the number of data memory instances (64).

Rotation. Rotation (Figure 10(b)) generates slightly different images through rotating images from the downsampled image. It makes rotated images until it reaches the maximum possible number of input channels. The angle of rotation ranges from -30 to 30 degrees. For instance, given a downsampled three-channel image input and target channel size of 64, it generates rotated images with an angle linearly spaced between -30 to 30 degrees.

Tile. Tile (Figure 10(c)) divides the original image into multiple tiles and stacks those tiles across channels. Specifically, given the number of images take $K = \lceil \frac{C_O}{C_I} \rceil$, it finds the nearest square

[2] https://github.com/walsvid/CoordConv

number S that is higher than or equal to K, (e.g., $S = 5^2 > 22$ when $K = 22$). The original image is then divided into equal-sized patches. Each patch is subsequently downsampled to the target size. The downsampled patches are collected and concatenated along the channel dimension, forming a new image with the desired number of channels. If the total number of patches exceeds the target number of channels, the excess patches are discarded.

Patch-wise sequential sampling. Patch-wise sequential sampling (Figure 10(d)) is similar to DEX but it involves sequential sampling within a patch instead of even sampling. Specifically, it samples the first K samples for each patch and follows the same channel-wise stacking procedure in DEX.

Patch-wise random sampling. Patch-wise random sampling (Figure 10(e)) is similar to DEX but it involves random sampling within a patch instead of even sampling. Specifically, it samples randomly-selected K samples for each patch and follows the same channel-wise stacking procedure in DEX.

B Additional Experiments

B.1 Overhead of the channel expansion on devices

Channel expansion latency. The latency of the channel expansion process depends on the processor's computational capability. During our evaluation, we pre-processed data on a powerful server, and thus data processing was negligible. We additionally conducted data processing on the ultra-low-power MCU processor on the board (Arm Cortex-M4) to understand the data processing overhead on less-capable devices. We measured the overhead of applying DEX to expand channels from a $3 \times 224 \times 224$ image (a typical size for ImageNet) to $64 \times 32 \times 32$ (the highest channel expansion used in our accelerators) on the MAX78002's Arm Cortex-M4 (120MHz).

This process took 2.2 ms on the Arm Cortex-M4. In terms of memory, this addition took the SRAM memory of 62KB ($64 \times 32 \times 32$ Bytes - $3 \times 32 \times 32$ Bytes) on the processor. However, since DEX extends data to a size that the data memory in the AI accelerator can accommodate, this additional memory will not be an issue from the AI accelerator's perspective.

Impact on end-to-end inference performance. Note that the MCU processor and the AI accelerator are independent processing components that run in parallel. This means that if the inference latency on the accelerator is higher than the data processing latency, data can be pre-processed for the next inference during the current inference (and thus data processing latency can be hidden). For inference, the inference latency of EfficientNet (11.7ms) is higher than the data processing latency of 2.2ms, and thus the inference throughput remains the same under continuous inference.

However, this depends on the scenario. The end-to-end impact of data processing latency depends on the processor's computational capability, the dimension of the data, and the size of channel expansion. For instance, in scenarios where data processing is done and transferred in more capable machines (e.g., cloud servers, smartphones, etc.) than the MCU processor on the tiny AI accelerator, the impact of data processing can be even more negligible.

B.2 Power consumption

We measured the power consumption of the inference on MAX78000 by varying the size of the channel extension with a Monsoon Power Monitor. The result is shown in Table 6. As the number of channels increased, power consumption increased accordingly. This is because a higher number of channels uses more processors in the AI accelerator, leading to increased power consumption.

Table 6: The power consumption of inference measured by varying the size of the channel extension with a Monsoon Power Monitor. All numbers are in milliwatts (mW).

Model	Chan = 3	Chan = 6	Chan = 18	Chan = 36	Chan = 64
SimpleNet	53.82	53.85	58.21	61.42	68.9
WideNet	60.74	61.37	63.76	67.92	77.14

C Example images generated from DEX

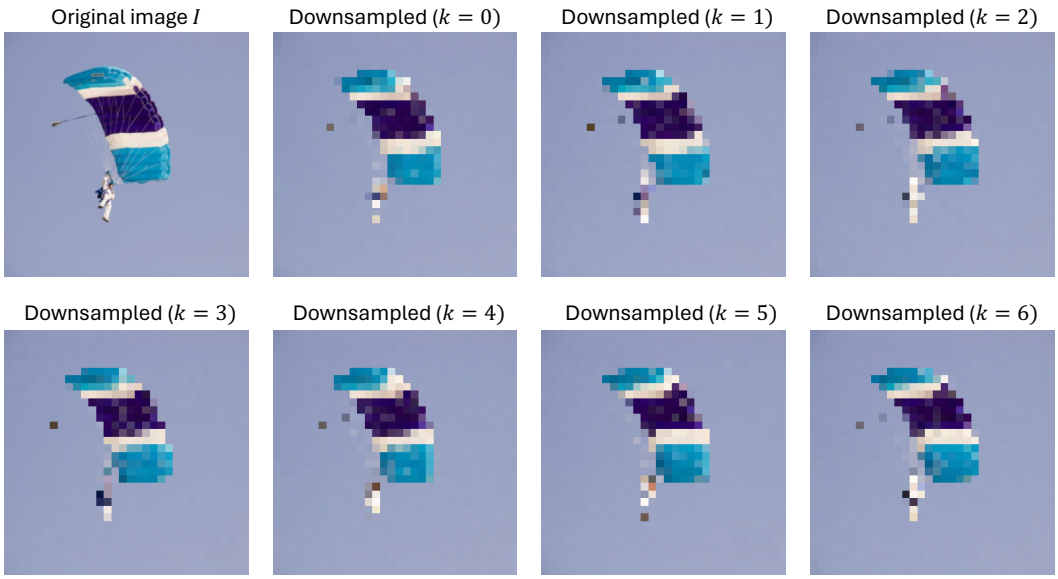


Figure 11: Examples images generated from an original $3 \times 350 \times 350$ image (ImageNette) to $3 \times 32 \times 32$ downsampled image via DEX. $k = 0$ to $k = 6$ cases are shown only. Each generated image contains different pixel information, which collectively enhances feature learning in CNNs.

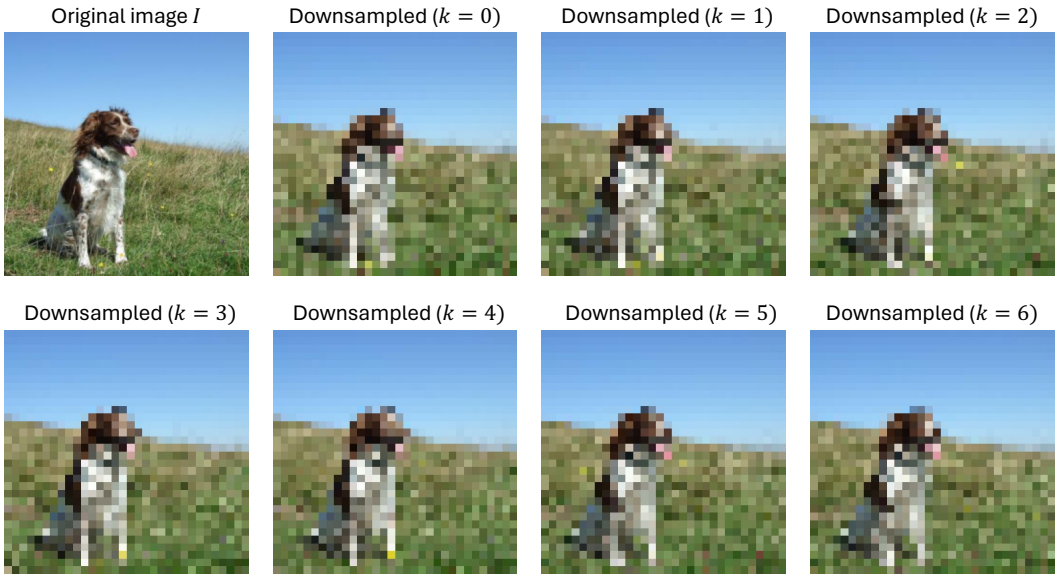


Figure 12: Examples images generated from an original $3 \times 350 \times 350$ image (ImageNette) to $3 \times 32 \times 32$ downsampled image via DEX. $k = 0$ to $k = 6$ cases are shown only. Each generated image contains different pixel information, which collectively enhances feature learning in CNNs.

D License of assets

Datasets. ImageNette dataset (Apache-2.0 license), Caltech101 (CC BY 4.0), Caltech256 (CC BY 4.0), and Food101 dataset (MIT license).

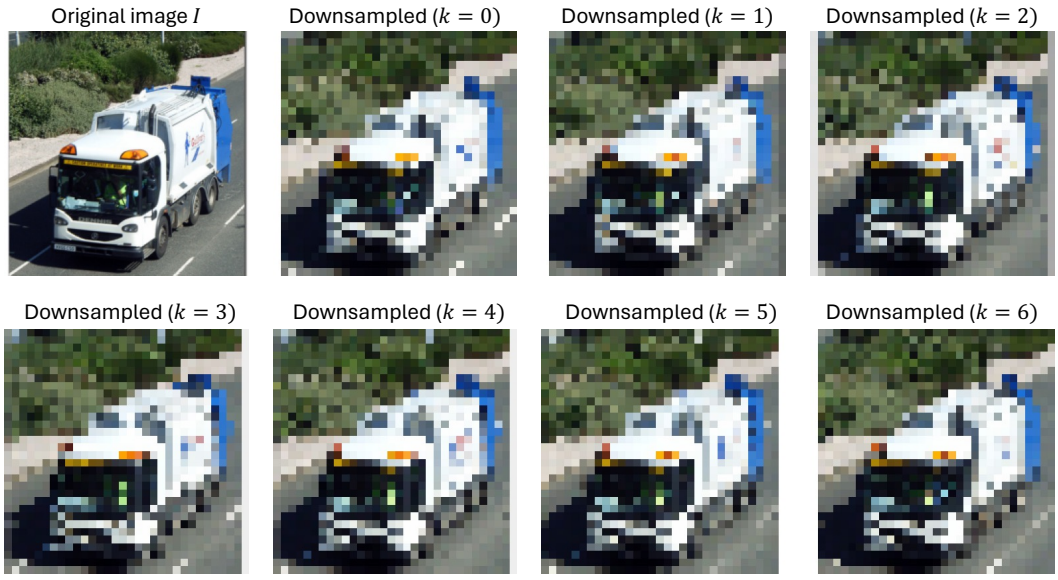


Figure 13: Examples images generated from an original $3 \times 350 \times 350$ image (ImageNette) to $3 \times 32 \times 32$ downsampled image via DEX. $k = 0$ to $k = 6$ cases are shown only. Each generated image contains different pixel information, which collectively enhances feature learning in CNNs.

Codes. AI8X-training for MAX78000 and MAX78002 (Apache-2.0 license), AI8X-synthesis for MAX78000 and MAX78002 (Apache-2.0 license), and PyTorch implementation of CoordConv (MIT license).

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and precede the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS paper checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: See §6.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: No theoretical result.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Experimental details are in §4 and Appendix A.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Yes, the source code is available at https://github.com/Nokia-Bell-Labs/data-channel-extension.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Experimental details are in §4 and Appendix A.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: See 4.2. We ran the experiments with three random seems (0,1,2) and reported the standard deviations.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See §2, §4, and Appendix A.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We follow the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: See §6.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: No such components.

    Guidelines:

    - The answer NA means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: See §4 and Appendix D.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: No new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: No crowdsourcing conducted.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: No IRB required.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.