TinyM^{2}Net-V2: A Compact Low Power Software Hardware Architecture for <u>M</u>ulti<u>m</u>odal Deep Neural Networks

HASIB-AL RASHID, University of Maryland, Baltimore County, USA
UTTEJA KALLAKURI, University of Maryland, Baltimore County, USA
TINOOSH MOHSENIN, University of Maryland, Baltimore County, USA

With the evaluation of Artificial Intelligence (AI), there has been a resurgence of interest in how to use AI algorithms on low-power embedded systems to broaden potential use cases of the Internet of Things (IoT). To mimic multimodal human perception, multimodal deep neural networks (M-DNN) have recently become very popular with the classification task due to their impressive performance for computer vision and audio processing tasks. This paper presents *TinyM^{2}Net-V2* - a compact low-power software hardware architecture for <u>m</u>ulti<u>m</u>odal deep neural networks for resource-constrained tiny devices. In order to compress the models to implement on tiny devices, cyclicly sparsification and hybrid quantization (4-bits weights and 8-bits activations) methods are used. Although model compression techniques are an active research area, we are the first to demonstrate their efficacy for multimodal deep neural networks, using cyclicly sparsification and hybrid quantization of weights/activations. *TinyM^{2}Net-V2* shows that even a tiny multimodal deep neural network model can improve the classification accuracy more than that of any unimodal counterparts. Parameterized M-DNN model architecture was designed to be evaluated in two different case-studies: vehicle detection from multimodal images and audios and COVID-19 detection from multimodal audio recordings. The most compressed *TinyM^{2}Net-V2* achieves 92.5% COVID-19 detection accuracy (6.8% improvement from the unimodal full precision model) and 90.6% vehicle classification accuracy (7.7% improvement from the unimodal full precision model). A parameterized and flexible FPGA hardware accelerator was designed as well for *TinyM^{2}Net-V2* models. To the best of our knowledge, this is the first work accelerating multimodal deep neural network models on low power Artix-7 FPGA hardware. We achieved energy efficiency of 9.04 GOP/s/W and 15.38 GOP/s/W for case-study 1 and case-study 2 respectively which is comparable to the state-of-the-art results. Finally, we compared our tiny FPGA hardware implementation results with off-the-shelf resource-constrained devices and showed our implementation is faster and consumed less power compared to the off-the-shelf resource-constrained devices.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Computer systems organization** → *Reconfigurable computing*; *Real-time system architecture*.

Additional Key Words and Phrases: tinyML, Multimodal Deep Neural Networks, FPGA, Model Compression.

1 INTRODUCTION

The era of Machine Learning (ML) and Deep Learning (DL) has had a tremendous impact on how we live today. This ease of use is achievable because AI devices can now handle computationally complex activities, reducing or even eliminating the need for human intervention. As a result, today ML-based systems can be seen in almost every industry including healthcare diagnostics, security, autonomous vehicles, robotics, image analytics, knowledge reasoning, navigation, and many more. Edge Machine Learning (edgeML) and Tiny Machine

Authors' addresses: Hasib-Al Rashid, University of Maryland, Baltimore County, USA, hrashid1@umbc.edu; Utteja Kallakuri, University of Maryland, Baltimore County, USA, ukalla1@umbc.edu; Tinoosh Mohsenin, University of Maryland, Baltimore County, Catonsville, MD, 21250, USA, tinoosh@umbc.edu.

1539-9087/2023/5-ART $15.00
https://doi.org/10.1145/3595633

Learning (tinyML) are two contemporary approaches that aim to make machine learning more accessible. These are the fields that fall at the confluence of machine learning and embedded Internet of Things (IoT) devices. A new generation of neural networks has emerged as a result of the exponential growth of resource-constrained microcontroller (MCU), microprocessor (MPU) and field programable gate array (FPGA)-powered devices. This new generation of neural networks is more concerned with model efficiency than model accuracy, and its size is also significantly reduced. We are able to perform direct data analytics close to the sensor by running ML and DL models on very tiny edge devices, which significantly expands the range of applications for artificial intelligence (AI). These AI-based systems are now performing with accuracy that is nearly on par with that of humans for certain kinds of work.

However, the human perceptual system is multimodal. Through their use of their senses, humans have the innate cognitive capacity to relate and absorb information coming from a variety of sensory modalities at the same time at a single event. In a multimodal method, we take in information and approach problems. The items are visible to us at the same time that we hear the sound of it. In order for machine learning algorithms to successfully imitate human behavior, it is imperative that they incorporate multiple modalities of input. This, in turn, gives rise to a new machine learning technique, Multimodal Deep Neural Networks (M-DNN). Modes are essentially different routes through which information can be transmitted. These data come from a variety of sources, and they are semantically associated with one another. Additionally, they may supply complimentary information to one another, thereby reflecting patterns that are not obvious when working with particular modalities on their own. Contemporary IoT and wearable devices, such as activity trackers, environmental sensors, image sensors, and audio sensors, are able to regularly generate massive amounts of data. In order to create results that are more accurate, modern ML techniques are becoming increasingly reliant on data coming from a wide variety of sources. M-DNN systems bring together those disparate, disconnected data from a variety of sensors, which enables them to contribute to the production of more accurate and robust predictions. There are two primary benefits associated with M-DNN systems. To begin, having several sensors monitor the same data at the same time can result in more accurate predictions. This is because identifying changes in the data may need the presence of both modalities. Second, the integration of a large number of sensors makes it possible to collect supplementary data or patterns that may be missed by individual modalities.

Due to the increasing number of model parameters and computations, M-DNN models are difficult to be implemented on resource limited edgeML and tinyML applications. For instance, the most cutting-edge ARM Cortex-M7 CPU-based MCU available today only has 320KB of on-chip SRAM and 1MB of Flash memory available to it. Even with more powerful hardware like the Raspberry Pi 4, which has an L2 cache of 1MB, this demonstrates that there is a large gap between the capacity that is required and that which is available in the hardware for DL models to be implemented. Additionally, a significant amount of energy is consumed whenever large model parameters are moved between memory hierarchies (such as from DRAM/Flash to SRAM or from SRAM to a register). Reports indicate that in order to run a DL model on an FPGA, a power consumption of between 3.5W and 9W is necessary, with over 80% of the power consumption being attributable to the transmission of model parameters [48]. This is a substantial increase from what is required by tinyML. The complexity of implementing efficient M-DNN inference while maintaining a low peak memory consumption is further increased by these new challenges. In this paper, we address the aforementioned issue and implement M-DNN models on different resource-constrained hardware. To implement energy-efficient M-DNN models on tiny processing hardware, we use all the benefits that state-of-the-art compression techniques have to offer. We proposed a compact low power software hardware architecture *TinyM^{2}Net-V2* which is re-configurable in terms of input data modality and data shapes, number of layers, filter sizes etc. hyper-parameters for the sake of application requirements. The high-level overview of the proposed *TinyM^{2}Net-V2* is presented in the figure 1. We evaluated *TinyM^{2}Net-V2* with two different multimodal case-studies: audio processing with multimodal audios and vehicle classification with multimodal images and audios. *TinyM^{2}Net-V2* is then implemented on low power tiny Artix-7 FPGA, off-the-shelf

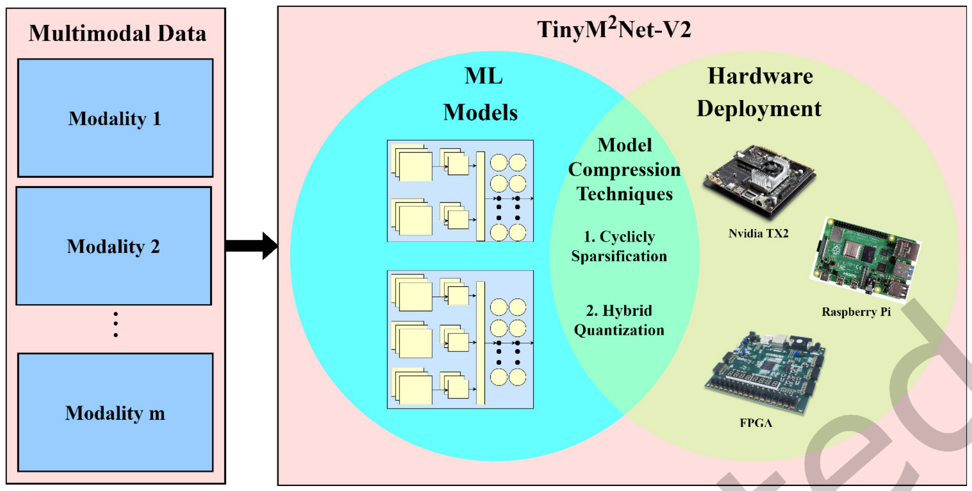

Fig. 1. The high-level overview of the proposed TinyM^{2}Net-V2. TinyM^{2}Net-V2 can take any number of modalities of data, designs ML models for the required task, compress the models with state-of-the-art compression techniques like cyclic sparsification and hybrid quantization and then deploy them onto tiny resource-constrained hardware.

NVIDIA Jetson TX2 board and commodity tiny MPU, Raspberry Pi 4 to measure real-time performance on different resource-constrained hardware. The main contributions of this paper are as follows:

- We propose *TinyM^{2}Net-V2*, an end-to-end flexible software-hardware co-designed architecture for M-DNN. *TinyM^{2}Net-V2* introduces multimodal data (images and audios) to be adapted in tinyML models to improve the application specific accuracies.
- We compress the M-DNN models with cyclic sparsely connected (CSC) architecture and hybrid quantization to reduce memory consumption and computational complexity for tinyML hardware implementation. Although model compression techniques are an active research area, we are the first to demonstrate their efficacy for M-DNNs, using cyclicly sparsification and hybrid quantization of weights/activations.
- We design a parameterized FPGA hardware accelerator introducing multimodality, cyclic sparsity and hybrid precision quantization for energy-efficient deployment on tinyML hardware. To the best of our knowledge, *TinyM^{2}Net-V2* is the first work accelerating M-DNN models on FPGA hardware.
- We evaluate the proposed *TinyM^{2}Net-V2* for two different case-studies. *Case-study 1* includes Covid-19 detection from multimodal cough, speech and breathing audio recordings. *Case-study 2* includes vehicle classification using multimodal images and audios. Our combined flow achieves state-of-the-art results when comparing energy efficiency of different FPGA hardware accelerators and also implementing on variety of edge/tinyML devices.

The rest of the paper is organized as follows: Section 2 presents some related works along with the theoretical backgrounds. Section 3 provides description of the proposed TinyM^{2}Net-V2. Section 4 provides detailed analysis and results for TinyM^{2}Net-V2 evaluation. Section 5 presents the future direction of this research. Section 6 concludes the paper.

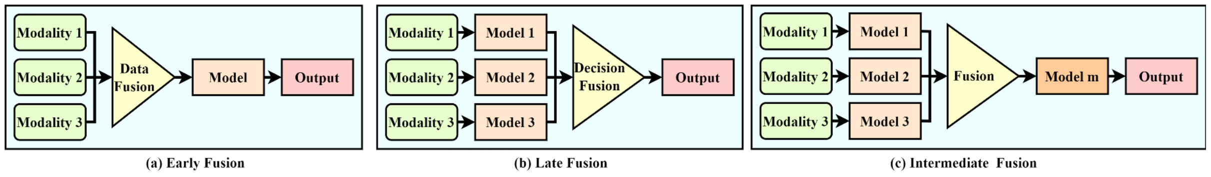

Fig. 2. Different types of data fusion techniques used by state-of-the-art multimodal models. We adopted hybrid/intermediate fusion technique for our proposed TinyM^{2}Net-V2.

2 RELATED WORKS AND THEORETICAL BACKGROUNDS

2.1 Multimodal Deep Neural Networks

The purpose of using multimodal data is that complementary information can be retrieved from each of the modalities evaluated for a given learning task, resulting in a better representation that can be utilized to achieve superior results than if only one modality is employed. The most important feature of the multimodal approach is the ability to represent data at various abstraction levels. The learned information can be integrated into two or more modalities for a certain hypothesis by employing an intermediate formulation. Different data fusion algorithms have been proposed in the literature to address this research question [4, 9]. The most popular and traditional data fusion algorithms are early fusion, late fusion, and hybrid/intermediate fusion algorithms. Figure 2 presents a high-level overview of the multimodal data fusion algorithms. A common practice for integrating data before analysis is called Early fusion (Fig. 2(a)). It uses two techniques. The first technique involves merging data while eliminating correlations. The second technique is to integrate the data at its latent sub Space, which has a lower dimension. On unprocessed raw data, early fusion is used and executed. To reduce complexity for modalities with variable sample rates, features are removed before fusion. It is challenging to sync data sources that are both discrete and continuous. Individual modality sources are used by Late fusion (Fig. 2(b)) throughout the decision-making process. The techniques of bagging and boosting are used to extract it from the ensemble classifiers. This technique can be applied when the modal data sources are uncorrelated in terms of sampling rate and data dimensionality. Many scientists prefer late fusion to early fusion even though there is no concrete evidence that late fusion is superior to early fusion [40]. Early and late fusions are combined to create Hybrid/Intermediate fusion (Fig. 2(c)). A joint representation of various modalities is learned through hybrid fusion. At the layer of representation that is shared by all, the fusion occurs. Throughout the training process, the loss is transmitted back to the feature extractor network. Using slow or gradual fusion, several modalities can be integrated [17, 40]. Intermediate fusion procedures are beneficial because they allow for a customized approach to fusing marginal representations, both in terms of depth and sequencing. It is arguable that this better depicts the genuine connections between the various modes of expression. It is possible then that more helpful latent components, both joint and marginal, may be identified. Since DL architectures make it simple to fuse marginal representations by linking them to a shared layer and ensuring that hierarchical representations match the real world, they are well-suited for intermediate fusion [40]. We adopted intermediate fusion technique for our TinyM^{2}Net-V2 based on this intuition.

Multimodal Deep Neural Networks (M-DNN) have recently gained much popularity in the domain of human activity recognition [7, 29], autonomous systems [20, 24] and most importantly, medical applications. Computer-aided diagnosis [23], tissue and organ segmentation [18], multimodal medical image retrieval [47], and multimodal medical image registration [39] are just some of the applications of multimodal deep learning in medical domain. Combining different medical imaging modalities such as MRI, CT scans, X-Rays etc are very popular nowadays in disease diagnosis. Recently, multimodal audio processing has gained popularity for detecting respiratory diseases [26, 35, 36].

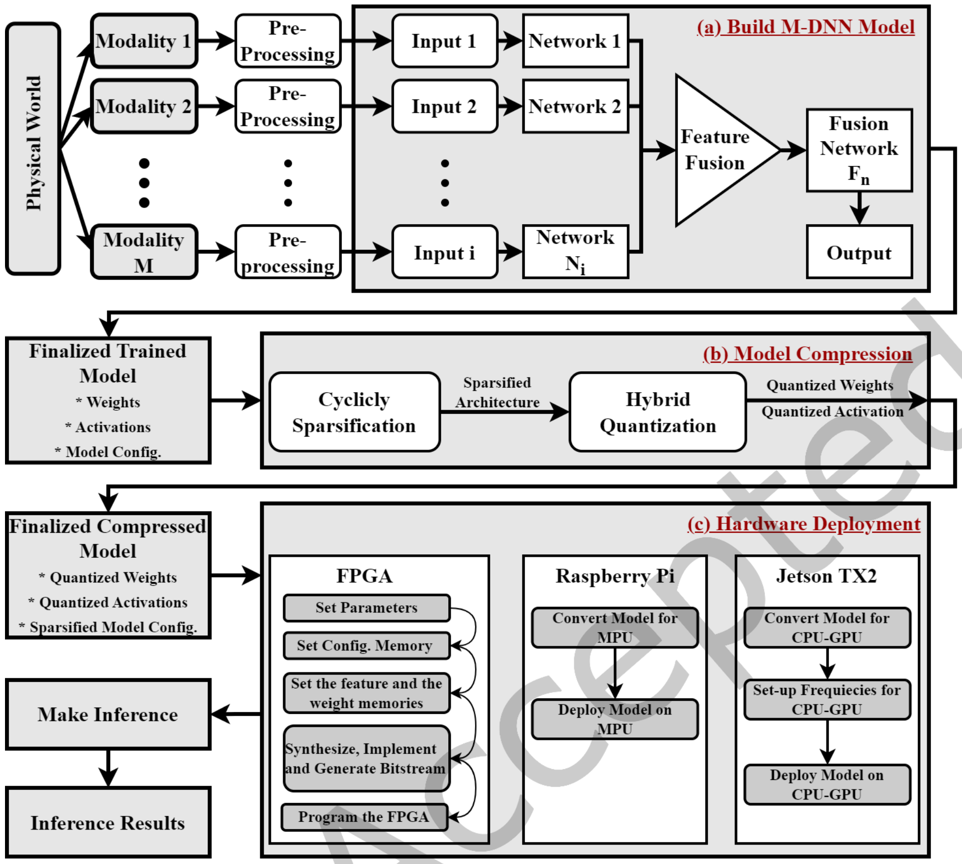

Fig. 3. The flow diagram of proposed TinyM^{2}Net-V2 software hardware architecture. We consider pre-processed multimodal inputs for our proposed TinyM^{2}Net-V2. Proposed TinyM^{2}Net-V2 is the sequential combination of the steps shown in the diagram.

2.2 TinyML Inference Hardware

A high-level review of the optimization strategies of deep neural networks (DNN) for tinyML on device inferences was presented by the authors in [25]. TinyML model optimization utilizes a variety of algorithms for parameter search, including Pruning, Sparsification and Quantization techniques. Element-wise pruning[28] and Structured Pruning [2, 19] compress the models so that they can be implemented on tinyML devices by reducing the weights of irrelevant elements. Researchers have decided to implement mixed precision quantization [13, 14, 45, 46, 49] and extremely low precision quantization [1, 21] in their DNN models in order to reduce the memory needs placed on them by these models. MCUNet [22], MicroNets [3], EtinyNet [48], TinyLSTMs [8] are networks that have been proposed for use in the deployment of DNN models on microcontroller units (MCU). Recently, ARIS [33], CoughNet-V2 [36], TinyM^{2}Net [35], OMAD [6], RhythmEdge [10] and authors in [30–32, 34] implemented various optimized unimodal and multimodal neural networks on different edge devices and tiny devices.

3 TinyM^{2}Net-V2 SOFTWARE HARDWARE ARCHITECTURE

3.1 Multimodal Deep Neural Network Model Architecture Design

The figure 3 shows how Multimodal Deep Neural Network (M-DNN) models are built taking different modalities of data from the physical world and pre-process them according to the need of the neural network formulation. The purpose of employing multimodal data is that complementary information can be recovered from each of the modalities evaluated for a given learning task, resulting in a better representation that can be used to achieve superior results than if only one modality is employed. We formulated the following multimodal learning problem, where different modalities of data are exploited in classification tasks. We adopted *Intermediate fusion* technique to fuse multiple modalities in this work due to its superiority over other types of fusion [40].

At first, let us consider that we have i number of input streams of different modalities: $X_1 = \{x_1^l, ..., x_T^l\}$, $X_2 = \{x_1^m, ..., x_T^m\}$, ... $X_i = \{x_1^n, ..., x_T^n\}$; where x_T^l, x_T^m, ... x_T^n refer to the $l-$, $m-$, ... $n-$ dimensional feature vectors of X_1, X_2, ... X_i modalities in time T, respectively. Then, by combining three modalities at time T, we consider the three unimodal output distributions at different levels of representation. We will then train our multimodal fusion network, F_n which will map X_1, X_2, ... X_i to the same categorical set of ground truth labels, $G = \{G^1, ..., G^T\}$. Then we will construct i different unimodal networks from X_1, X_2, ... X_i which will be denoted as N_1, N_2, ... N_i respectively. Here, $N_1 : X_1 \longrightarrow Y$, $N_2 : X_2 \longrightarrow Y$, ... $N_i : X_i \longrightarrow Y$, and $F_n = N_1 \oplus N_2 \oplus ... \oplus N_i$.

Y signifies the predicted class labels of the training samples generated by the output of the constructed network. $\oplus$ represents the fusion operation.

We designed the multimodal model empirically following the proposed multimodal model in [35]. We replaced the depthwise separable (DS) CNN layers with vanilla CNN layers to mitigate the accuracy degradation due to DS-CNN. Hyper-parameters (filter size, number of filters) for each unimodal network can be decided by adopting neural architecture search (NAS) algorithms. However, for our tinyML case-studies, unimodal networks are decided empirically as the search space is comparatively tractable due to limited model architectures. Then the unimodal networks with best accuracies are selected for each modality to be concatenated. Then after getting features from each of the unimodal network, they are fused and passed to the fusion network. Finally, the classification output is visualized as the probability distribution of the last fully connected layer using the Softmax activation function.

3.2 Model Compression for Tiny Machine Learning Hardware Implementation

Pruning and quantization are two methods commonly used to reduce the size of large neural network models for implementation on edge hardware, which often have limited computational resources and memory. Pruning techniques remove unimportant or redundant parameters from the model, thereby reducing its size. However, this also introduces irregularity into the DNN architecture, as some weights are zeroed out while others are kept. To handle these non-zero weights, an additional indexing memory is required. Quantization, on the other hand, reduces the precision of the weights and activations in the model. It converts the parameters from floating-point numbers to integer values, which requires less memory and computation. Combining pruning and quantization together can provide a significant reduction in the size of the model while still maintaining its accuracy to a certain extent. In order to overcome the irregularity introduced by pruning and reduce the need for indexing memory, structural sparsity is introduced, which eliminates this indexing through an appropriate architecture design. The following sections describe special cases for structural sparsity and quantization together which are used in our TinyM^{2}Net-V2 to perform model compression

3.2.1 Cyclicly Sparsification. Cyclic Sparsely Connected Convolutional Neural Networks (CSC-CNNs) [12] is a variation of the conventional Convolutional Neural Networks (CNNs) that employ sparse connectivity patterns between neurons. The sparse connectivity in CSC-CNNs creates a loop-like structure, which allows for an

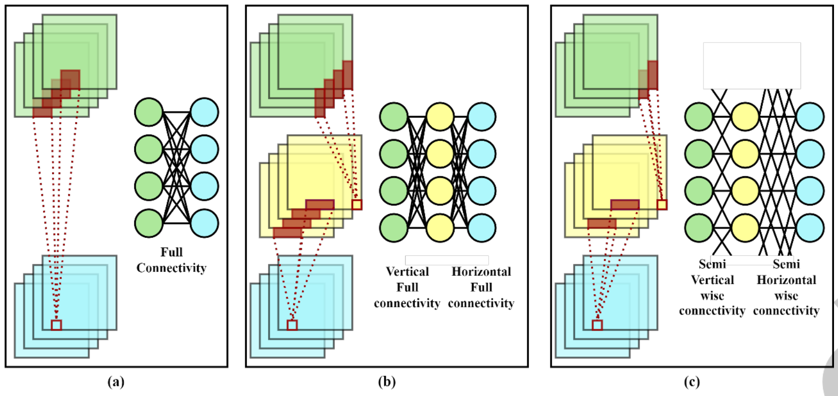
Full Connectivity
Vertical Full connectivity
Horizontal Full connectivity
Semi Vertical wise connectivity
Semi Horizontal wise connectivity
(a)
(b)
(c)

Fig. 4. Key idea in compact schemes of CSC architecture illustrated with a typical multi-layer architecture and an equivalent structured graphs. (a) Baseline CNN with fully connectivity (b) Low-Rank Expansion [16, 42, 43] (c) Cyclic Sparsely Connected (CSC) CNN.

efficient flow of information. This type of architecture reduces the number of parameters, resulting in a decrease in overfitting and improved performance on resource-constrained devices.

In comparison, GoogleNet and Inception-V3 [16, 42, 43] implementations utilize a low-rank decomposition of filters, which involves splitting them into two separable layers and applying them in sequence. This approach reduces the number of parameters and computational complexity, but still maintains full connectivity between neurons as shown in figure 4 (b). On the other hand, CSC-CNNs are a specialized form of low-rank decomposition that employs sparse connectivity patterns in the convolutional layers to achieve an even further reduction in the number of parameters and computational complexity, improving efficiency.

It was proposed by the authors in [11, 12] that cyclic sparsely connected layers shown in figure 4, which are cascades of a few layers, have memory and computation of the order of $O(NlogN)$ and their structure of connection can be adjusted to the point where they can be used in place of both CNN and Dense layers. The cyclic sparse connection occurs channel-to-channel for CNN layers. As a replacement for CNN and dense layers, we used the cascaded two-layer CSC-II architecture from [12].

We use the CSC-II architecture from [11, 12] to compress our model. CSC hyper-parameters like as connection (C), fan in/out (F), and dilation (D) can be fine-tuned using equations mentioned in [11]:

$$F^L = NC \tag{1}$$

$$\begin{cases} C\sum_{i=0}^{N-1} x^i & = p_0(x)\cdot p_1(x) \quad \bmod \left(x^N - 1\right) \\ p_0(x) & = \sum_{i=0}^{F-1} x^{D_0.i} \quad , D_0 = 1 \\ p_1(x) & = \sum_{i=0}^{F-1} x^{D_1.i} \quad , D_1 = \frac{F}{C} \end{cases} \tag{2}$$

Here, N = number of nodes and $p_T(x) = C\sum_{i=0}^{N-1} x^i$ is a generator polynomial. The end goal for these hyperparameter tuning was to maintain the higher accuracy depending on the case-studies. Equations 10 and 12 from [12] are the base to replace the calculations of the original dense and CNN layers with CSC connected dense and CNN layers. To have better performance, we have included CSC in all layers except the first CNN layer that processes the input data and the very last dense layer that is the classification layer.

3.2.2 Hybrid Quantization. Quantization is the process of converting real-valued numbers in a neural network to lower precision representations, such as fixed-point numbers, in order to reduce the memory and computation requirements for deployment on hardware with limited resources. Now-a-days uniform 8 bits quantization achieves full precision level of accuracy [49]. We can further reduce memory requirements changing the bit precision of both the weights and activation to the different lower bits which is termed as hybrid quantization.

However, quantizing the activations of a neural network is more challenging compared to quantizing the weights, as the activations have a wider range of values and can dynamically fluctuate with the input to the network. Activations are the output of each neuron in a neural network and are used as input for the next layer of neurons. They are computed by multiplying the inputs with the weights and passing the result through a non-linear activation function. Because of the multiply-accumulate operations, the range of activations can be much wider compared to the weights, making it difficult to choose an appropriate quantization scheme. Moreover, the activations can change rapidly with changes in the input, making it difficult to determine the right quantization range ahead of time. To maintain a good level of accuracy in the quantized model, it is important to preserve a higher level of precision in the activations compared to the weights. This can help avoid the need to retrain or recalibrate the model, which can be time-consuming and computationally expensive. Therefore we used higher level of precision for activations and used lower bits of weights.

We used Qkeras [5] to incorporate *quantize-aware training* scheme. This scheme uses the quantization into the training process. The model is trained with quantization operations included, so that the quantization error is reduced and the model accuracy is improved. Authors in [15] describes that for each layer, quantization is parameterized by the number of quantization levels and clamping range, and is performed by applying point-wise the quantization function q defined as follows:

$$\begin{aligned}\text{clamp}(r; a, b) &:= \min(\max(x, a), b)\\ s(a, b, n) &:= \frac{b-a}{n-1}\\ q(r; a, b, n) &:= \left\lfloor \frac{\text{clamp}(r; a, b) - a}{s(a, b, n)} \right\rceil s(a, b, n) + a,\end{aligned} \tag{3}$$

where r is a real-valued number to be quantized, $[a; b]$ is the quantization range, n is the number of quantization levels, and $\lfloor . \rceil$ denotes rounding to the nearest integer.

3.3 TinyML Hardware Deployment

Three different hardware choices - tiny FPGA, Raspberry Pi 4 and NVIDIA Jetson TX2 board are used in proposed TinyM^{2}Net-V2 hardware deployment as mentioned in the Fig. 3(c). We would advocate for tiny FPGA implementation TinyM^{2}Net-V2 hardware deployment because FPGA provides better flexibility, parallelism, energy efficiency and reduced latency compared to off-the-shelf hardware deployment boards. However, the FPGA implementation has implementation complexity and lack of libraries to directly convert the models from its software counterpart, which is very user-friendly when off-the-shelf devices are used for hardware deployment.

The tiny FPGA design is described in Verilog and can be programmed onto the target device after assigning a set of hardware specific parameters. Fig. 3(c) shows the ingredients needed to program and deploy the tiny FPGA as the target inference device. These parameters are the Number of Processing Elements (# PEs), the sizes of the input feature map memory (which is large enough to hold the largest feature map generated for all the modalities), the size of the weight memories (one in each PE) and the output memory size. Finally, the parameter M which indicates the number of modalities is also assigned. The next step is to generate the binary representation files (or the mem files) for the weights and the input feature maps. Generate the network configuration file to load the configuration memory. The design is now synthesized, implemented and programmed onto the target tiny FPGA using the Xilinx Vivado tool. For Raspberry Pi 4, the compressed models are converted using available libraries and then deployed onto it. For Jetson TX2 board, after converting the compressed models with available libraries, we can set-up the working frequencies for the CPU-GPU configuration. Then we can deploy the models on the Jetson TX2 board which is mentioned in Fig. 3(c). Following sections describes the deployment procedure in details.

3.3.1 FPGA Hardware Architecture Design and Deployment. The hardware for the tiny FPGA is designed such that it is independent of a DRAM. It integrates the processing components along with a sufficient on-chip SRAM memory to store the DCNN model and generated intermediate feature maps needed for one inference[50]. For a well designed hardware with the aforementioned consideration, with #PE number of Processing Elements (or processing components), #MAC number of Multiply-Accumulators, running at a frequency of freq MHz, the peak-performance can approach,

$$2 \times \#PE \times freq \times \#MAC \tag{4}$$

Fig. 5 shows the hardware architecture for the CSC-CNN processor. The hardware architecture aims to implement the equation. 12 from [12] for the CSC-CNN layers. With little modification to the parameters of the CSC layers, the hardware architecture can be programmed to implement CSC-Dense layers. Additional DCNN layers, such as standard convolution and depth-wise separable convolutions, can also be implemented by modifying the parameters. The hardware architecture shown in the Fig. 5 has been designed in Verilog HDL and implemented on the Xilinx Artix-7 FPGA. The hardware architecture is designed to compute DCNNs with or without CSC layers and have a programmable number of Processing Engines (PEs), MACs per PE, input modalities along with allowing a variable precision to the datawidth that is equal to the model quantization.

The main components of the hardware shown in Fig. 5 includes the PE array, a dedicated CSC router and a partitioned feature map memory to store the intermediary input feature maps. Each PE is also equipped with another SRAM (the output memory) that buffers the generated layerwise outputs back to the input feature map memory once the current layer is processed for all the modalities. The PEs are further provided with a programmable number of MAC units that compute the MACs between the filters and the corresponding feature values. The MAC modules support hybrid-precision by extending the lower precision data value to match the higher precision data value by extending zeros on the most significant side. This ensures that the inference accuracy of the accelerator is comparable to the testing accuracy of the software.

The address generator unit generates the addresses to the weight and the feature map memories in accordance to the parameters N_i, F and M that are the number of input nodes, number of non-zero filters and the modality respectively. The inclusion of M in the address generator as shown in Fig. 5 ensures that all the different modalities of the current layer are processed before moving to the next layer. Hence, the computations of the DCNN are parallelized within each modality while serially processing different modalities in a layerwise fashion. The weight memory within each PE is designed so that it can hold sufficient data for the PE to process the convolution according to the output channel tiling scheme. The weight memory, moreover, is designed to have a variable data-width size to enable the hybrid precision operations. Similarly, the partitioned output memory is designed such that it can store the generated results from the MAC modules. At the end of each patch of feature map, the parallel data from the MAC modules is stored in a single row in the output memory.

The accelerator also houses a dedicated CSC based high-bandwidth router to fetch the data from the feature map memory to the PE array and load the data from the output memory to the input feature map memory. At the end of the computation of each layer for all the modalities, the data from the output memory is written to the feature map memory. This feature map data acts as the input for the next layers. For the sake of simplicity, the hardware architecture does not show the state-machine,max-pooling, batch-normalization, and the nonlinear activation functions.

To perform the convolution computations (i.e. repeated MAC operations), there are at least three different parallel processing schemes namely, output channel tiling, input channel tiling and image patch tiling. In this work, the PE arrays adhere to the output channel tiling scheme. Each PE processes the 2D convolution of one input feature map channel that can be accessed through the router with the corresponding filter set that is available in the weight memory. Additionally, within each PE; depending on the number of MACs-per-PE, follow

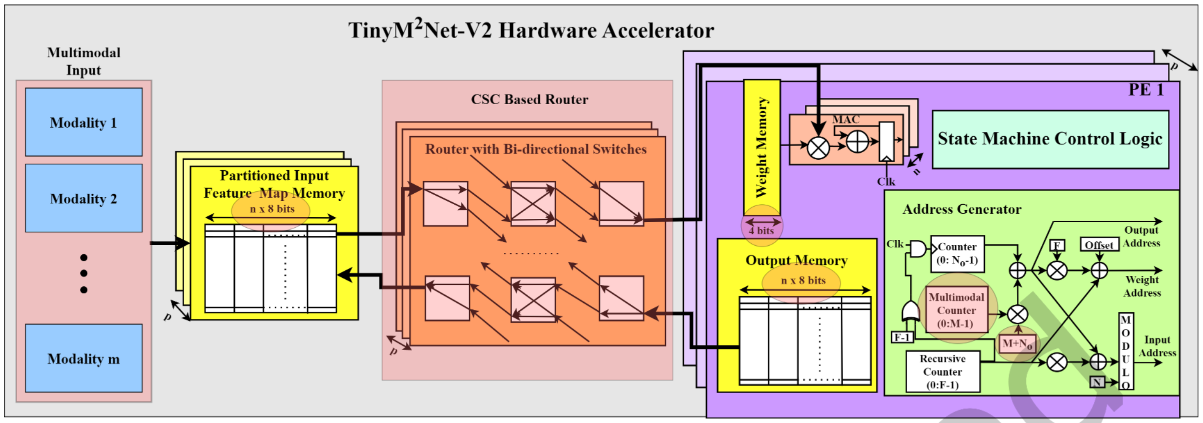

Fig. 5. Proposed TinyM^{2}Net-V2 tiny FPGA hardware accelerator architecture. Three key contributions towards the hardware architecture design are marked transparent red: 1. Multimodal input and multimodal counter, 2. Cyclic Sparsely Connection (CSC) based router, 3. Hybrid quantized feature map, weight and output memories. Complete hardware architecture is implemented on Artix-7 FPGA chip.

the image patch tiling scheme where the 2D channel is subdivided into multiple rows that can be worked on by the parallel multipliers.

3.3.2 Off-The-Shelf Device Deployment. The off-the-shelf device are single-board computer systems, that have great performance despite having a relatively straightforward architecture. There is a wide variety of single-board computer systems available, each of which contains a CPU and/or a GPU in addition to providing the opportunity to design hardware and software. A few examples of these are the Jetson Nano / TX2, Raspberry Pi, BeagleBoard, and Asus Tinker Board. ATMs, medical diagnostics, precision agriculture, intelligent home systems, robotic systems, and many other types of applications all make use of single-board computers for the tinyML implementation [41].

Raspberry Pi 4 is a single credit-card size board that can be used as an tinyML inference hardware. It has quad-core ARM Cortex A72 64-bit CPU running at 1.5GHz. NVIDIA Jetson TX2 is a mid-level board of the NVIDIA Jetson ecosystem, which is a small, powerful single board computer allowing parallel operations of multimple neural networksfor different applications. It has quad-core ARM Cortex-A57 CPU and dual-core NVIDIA Denver 2 CPU, both running at 2GHz. It also houses NVIDIA Pascal 256 CUDA cores which can be run at 1300MHz. We implemented compressed TinyM^{2}Net-V2 models on Raspberry Pi 4 and NVIDIA Jetson TX2 board. Tensorflow Lite is used as the library to convert the compressed model for both the off-the-shelf device deployments.

4 TinyM^{2}Net-V2 EVALUATION RESULTS AND ANALYSIS

4.1 Evaluation Case-Study 1: Vehicle Classification from Multimodal Images and Audios

4.1.1 Dataset. We used a subset of the dataset provided with the paper [44]. This large audio-visual events (AVE) dataset encompasses 28 different audio-visual events captured in 4143 10-second youtube videos. We have selected a small subset of 4 classes from this large dataset to make compatible with our resource constrained tiny hardware implementation. We selected 4 distinguished vehicles' AVE data which makes our target application to be vehicle classification from multimodal images and audios. After subsetting, we have 609 video recordings from where we extracted the images and corresponding audios of Airplane, Motorcycle, Car, and Truck. Sampling rate of the image extraction was one frame per 2 seconds. We have collected the images in .jpg format. We extracted in total 3045 images for all the 4 classes. On the contrary, sampling frequency was 22050Hz for the audio recordings. Length of our audios was 1 sec. We have takens 2 sec windows from the audios and converted

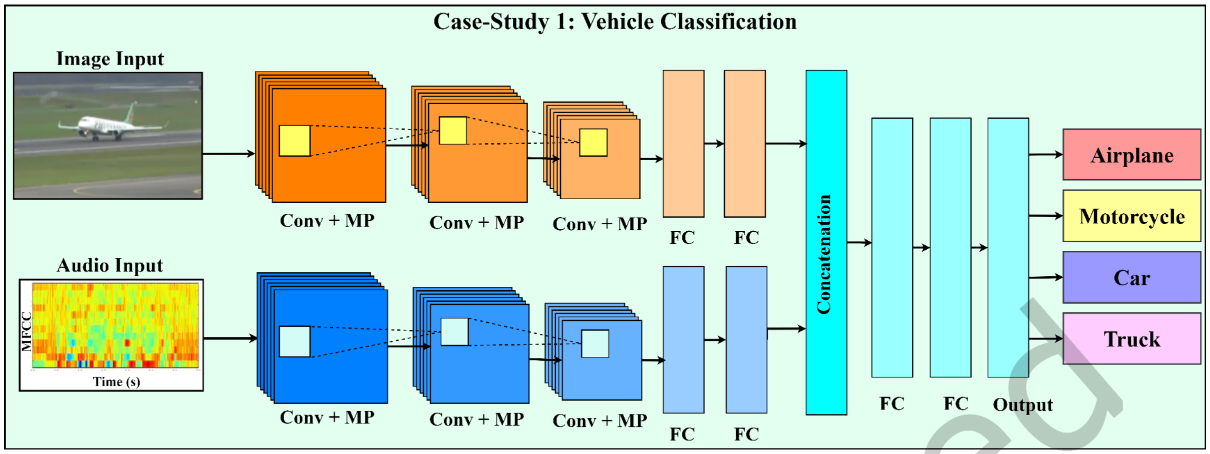

Fig. 6. The model architecture of the proposed *TinyM^{2}Net-V2* for Case-study 1. Here, Conv = 2 dimensional CNN, MP = Max Pooling, and FC = Fully Connected Layer.

into Mel-frequency cepstral coefficients (MFCC) spectrograms. We extracted in total 3045 MFCC spectrograms for 4 classe swhere rows correspond to features from MFCC and columns correspond to time (window). The total number of extracted images and corresponding audio MFCC spectrograms were as follows: Airplane - 920, Motorcycle - 495, Car - 940, Truck- 690. We have used the 70% of the data for training, 10% for the validation and 20% for testing during our experiments.

4.1.2 Experimental Setup, Results and Analysis. Proposed *TinyM^{2}Net-V2* received both the images and the audio MFCCs that corresponded to them as inputs. Following this, *TinyM^{2}Net-V2* will process two distinct modalities using its parallel CNN layers, will extract features, will fuse those features, and will finally categorize the data using a multiclass classification. Fig. 6 depicts the model architecture used for the case-study 1. After training the model with its baseline configuration, we created CSC configurations for the CNN layers and the fully connected layers, leaving the very 1st CNN layers and the output fully connected layer as it is, which is the common practice to address the accuracy degradation. The we re-trained the model again setting bit precision for weights as 4-bits and activations as 8-bits. The detailed baseline network architecture and the corresponding CSC architecture are mentioned in Table 1. Our model was trained for 200 epochs using categorical cross-entropy loss and the Adam optimizer. Accuracy was the performance evaluation metrics for our multimodal model. We also measured accuracy for the unimodal networks by passing the output of the fully connected layers to the Softmax layer.

Figure 7(a) presents the evaluation results of proposed TinyM^{2}Net-V2 for vehicle classification task. It is evident from the figure that when only image or audio is used for the vehicle classification, the classification accuracy were much lower. When both image and audio modality were used together as multimodal input, the classification accuracy for baseline CNN model increased to 92.8%, which is 7.7× improvement from unimodal model. Then after compressing the multimodal baseline model with CSC layers and reducing the data bit-width using hybrid quantization, the accuracy degrades to 90.6% which is 2.2% less. This 2.2% performance degradation is still allowable to be implemented on resource constrained edge hardware, considering the hybrid quantization and model compression due to CSC compressed layers. Table 1 presents the model compression ratio for baseline model and CSC compressed model in terms of number of parameters and number of computations required. In figure 7(b), the FPGA memory requirement for baseline model is 36.2Mbits. The FPGA memory requirements

Table 1. Baseline TinyM^{2}Net-V2 for Case-study 1 and its compressed model using CSC-II architecture in detail. Input shape is denoted by $W_x \times H_x \times N_i$, Filter shape is denoted by $W_f \times H_f \times N_i \times F$. D denotes dilation; I, and A denote Image, and Audio modalities. M denotes multimodality. Number of parameters reduces 5.5× and number of computation reduced 1.8× due to CSC compression.

TinyM^{2}Net-V2	Base Model			CSC II Compressed		
Layer	Input Shape	Filter Shape	Dial.	Input Shape	Filter Shape	Dial.
CNN_I 1	224×224×3	3×3×3×64	1	-	-	-
CNN_I 2	112×112×64	3×3×64×64	1	112×112×64	3×1×64×16	1
				112×112×64	1×3×64×16	4
CNN_I 3	56×56×64	3×3×64×32	1	56×56×64	3×1×64×8	1
				56×56×64	1×3×64×8	4
FC_I 1	1×1×64	1×1×64×64	1	1×1×64	1×1×64×8	1
				1×1×64	1×1×64×8	4
FC_I 2	1×1×64	1×1×64×64	1	1×1×64	1×1×64×8	1
				1 × 1 × 64	1 × 1 × 64 × 8	8
CNN_A 1	997 × 13 × 1	3 × 3 × 1 × 64	1	-	-	-
CNN_A 2	498 × 6 × 64	3 × 3 × 64 × 64	1	498 × 6 × 64	3 × 1 × 64 × 16	1
				498 × 6 × 64	1 × 3 × 64 × 16	4
CNN_A 3	249 × 3 × 64	3 × 3 × 64 × 32	1	249 × 3 × 64	3 × 1 × 64 × 32	1
				249 × 3 × 64	1 × 3 × 64 × 32	4
FC_A 1	1 × 1 × 64	1 × 1 × 64 × 64	1	1 × 1 × 64	1 × 1 × 64 × 8	1
				1 × 1 × 64	1 × 1 × 64 × 8	4
FC_A 2	1 × 1 × 64	1 × 1 × 64 × 64	1	1 × 1 × 64	1 × 1 × 64 × 8	1
				1 × 1 × 64	1 × 1 × 64 × 8	8
Concatenation						
FC_M 1	1 × 1 × 64	1 × 1 × 64 × 64	1	1 × 1 × 128	1 × 1 × 128 × 16	1
				1 × 1 × 128	1 × 1 × 128 × 16	4
FC_M 2	1 × 1 × 64	1 × 1 × 64 × 64	1	1 × 1 × 64	1 × 1 × 64 × 8	1
				1 × 1 × 64	1 × 1 × 64 × 8	8
FC_M 4	1 × 1 × 64	1 × 1 × 64 × 4	1	-	-	-
Parameters	137728			24832 (**5.54×**)		
Computations	738 M			395 M (**1.86×**)		
Accuracy	92.8%			90.6%		

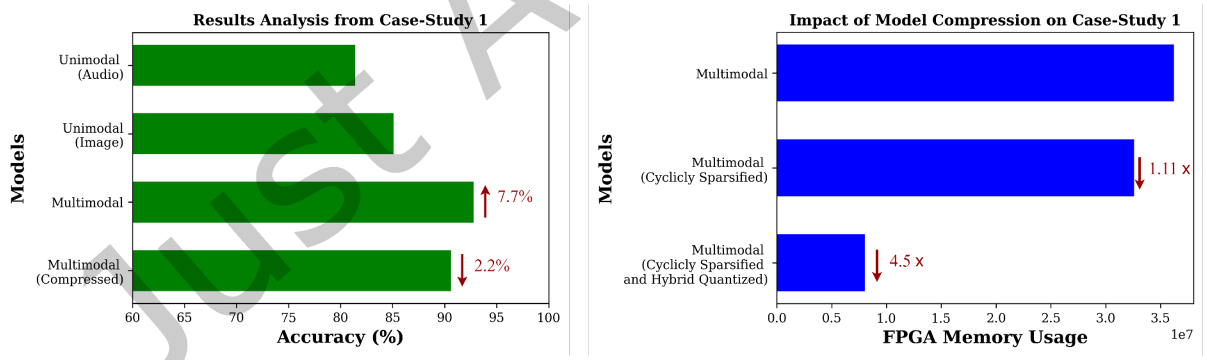

Fig. 7. (a) TinyM^{2}Net-V2 classification results for Case-Study 1 in terms of both unimodal and multimodal settings. The multimodal settings improved accuracy 7.7% compared to unimodal (audio) settings. Model compression techniques reduce 2.2% accuracy of multimodal setting. (b) Impact of model compression on TinyM^{2}Net-V2 for Case-Study 1. Using only CSC reduces 1.11× FPGA memory usage from baseline model. Using both CSC and hybrid quantization reduces 4.5× FPGA memory usage baseline model.

refers to largest feature map size for the particular model. With CSC compression and hybrid precision, the memory requirement is compressed by 4.5× to 8 Mbits.

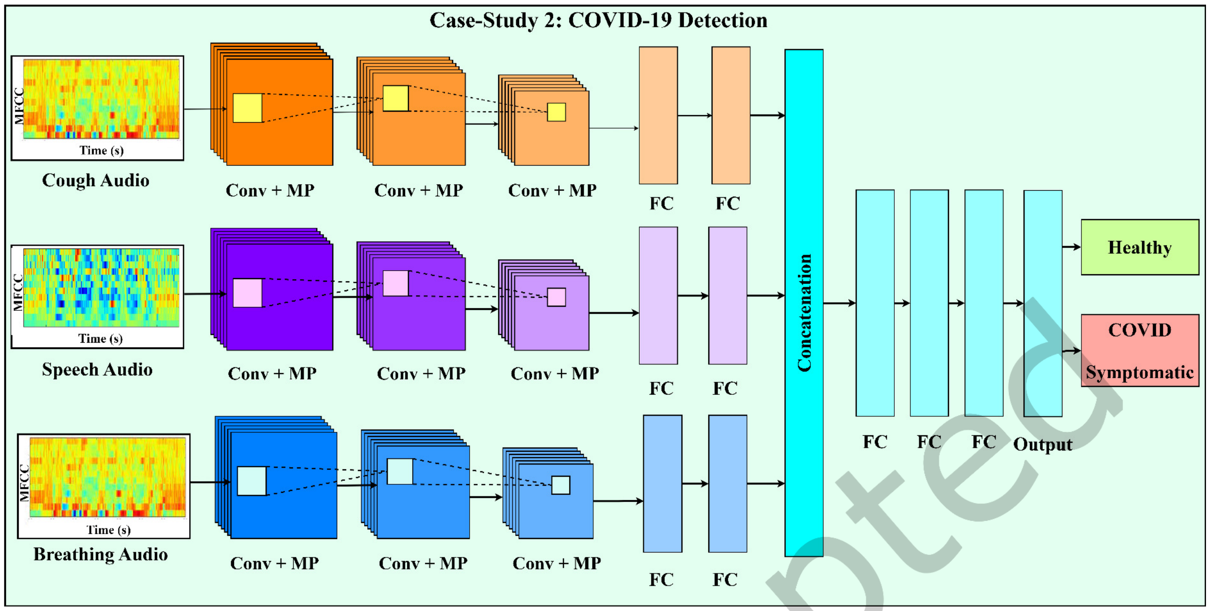

Fig. 8. The model architecture of the proposed *TinyM^{2}Net-V2* for Case-study 2. Here, Conv = 2 dimensional CNN, MP = Max Pooling, and FC = Fully Connected Layer.

4.2 Evaluation Case-Study 2: COVID-19 Detection from Multimodal Audios

4.2.1 Dataset. The second DiCOVA challenge dataset [38] was used in this case study. This dataset is a subset of the larger dataset, Coswara [37]. There are 929 participants in this dataset, each contributing one cough audio, one breathing audio, and one speech audio, among whom 172 participants were COVID positive. Each participant provided three different audio recordings, cough, speech, and breathing audios, along with their self-reported COVID-19 test result. To construct the two-class classification task, the original COVID-19 test results were classified as either positive (designated as 'P') or negative (identified as 'N'). As a last step, we generated 6000 random samples by dividing the audios into two-second chunks and balanced the samples in terms of their binary classes. Then we converted them into MFCC spectrogram and passed them to TinyM^{2}Net-V2. We selected the sampling frequency for all recordings as 22.5 KHz. We have used 70% data for training, 10% for validation and 20% for evaluation.

4.2.2 Experimental Setup, Results and Analysis. Proposed *TinyM^{2}Net-V2* received all the audio MFCCs as inputs. Following this, *TinyM^{2}Net-V2* will process three distinct modalities using its parallel CNN layers, will extract features, will fuse those features, and will finally categorize the data using a binary classification. Fig. 8 depicts the model architecture used for the case-study 2. After training the model with its baseline configuration, we created CSC configurations for the CNN layers and the fully connected layers, leaving the very 1st CNN layers and the output fully connected layer as it is, which is the common practice to address the accuracy degradation and to have better generalized performance. The we re-trained the model again setting bit precision for weights as 4-bits and activations as 8-bits. The detailed baseline network architecture and the corresponding CSC architecture are mentioned in table 2. Our model was trained for 200 epochs using categorical cross-entropy loss and the

Table 2. Baseline of TinyM^{2}Net-V2 for Case-study 1 and its compressed model using CSC-II architecture in detail. Input shape is denoted by $W_x \times H_x \times N_i$, Filter shape is denoted by $W_f \times H_f \times N_i \times F$. D denotes dilation; C, B, S denoting Cough, Breathing, Speech modalities. M denotes multimodality. Number of parameters reduces 5× and number of computation reduced 2.7× due to CSC compression.

TinyM^{2}Net-V2	Base Model			CSC II Compressed		
Layers	Input Shape	Filter Shape	D	Input Shape	Filter Shape	D
CNN_C 1	212×20×1	3×3×1×16	1	-	-	-
CNN_C 2	106×10×16	3×3×16×32	1	106×10×16	3×1×16×8	1
				106×10×16	1×3×16×8	2
CNN_C 3	53×5×32	3×3×32×64	1	53×5×32	3×1×32×8	1
				53×5×32	1×3×32×8	4
FC_C 1	1×1×64	1×1×64×64	1	1×1×64	1×1×64×16	1
				1×1×64	1×1×64×16	4
FC_C 2	1×1×64	1×1×64×32	1	1×1×32	1×1×32×8	1
				1×1×32	1×1×32×8	4
CNN_B 1	249×20×1	3×3×1×16	1	-	-	-
CNN_B 2	124×10×16	3×3×16×32	1	124×10×16	3×1×16×8	1
				124×10×16	1×3×16×8	2
CNN_B 3	62×5×32	3×3×32×64	1	62×5×32	3×1×32×8	1
				62×5×32	1×3×32×8	4
FC_B 1	1×1×64	1×1×64×64	1	1×1×64	1×1×64×16	1
				1×1×64	1×1×64×16	4
FC_B 2	1×1×64	1×1×64×32	1	1×1×32	1×1×32×8	1
				1×1×32	1×1×32×8	4
CNN_S 1	203×13×1	3×3×1×16	1	-	-	-
CNN_S 2	101×6×16	3×3×16×32	1	101×6×16	3×1×16×8	1
				101×6×16	1×3×16×8	4
CNN_S 3	50×3×32	3×3×32×64	1	50×3×32	3×1×32×8	1
				50×3×32	1×3×32×8	2
FC_S 1	1×1×64	1×1×64×64	1	1×1×64	1×1×64×16	1
				1×1×64	1×1×64×16	4
FC_S 2	1×1×64	1×1×64×32	1	1×1×32	1×1×32×8	1
				1×1×32	1×1×32×8	4
Concatenation						
FC_M 1	1×1×96	1×1×96×256	1	1×1×96	1×1×96×24	1
				1×1×96	1×1×96×24	4
FC_M 2	1×1×256	1×1×256×128	1	1×1×128	1×1×128×16	1
				1x1×128	1×1×128×16	8
FC_M 3	1×1×128	1×1×128×64	1	1×1×64	1×1×64×16	1
				1×1×64	1×1×64×16	4
FC_M 4	1×1×64	1×1×64×2		1×1×64	1×1×64×2	1
Parameters	153648			30512 (**5.03×**)		
Computations	28.5 M			10.5 M (**2.7**×)		
Accuracy	94.1%			92.5%		

Adam optimizer. Accuracy was the performance evaluation metrics for our multimodal model. We also measured accuracy for the unimodal networks by passing the output of the fully connected layers to the Softmax layer.

Figure 9(a) presents the evaluation results of proposed TinyM^{2}Net-V2. It is evident from the figure that when only single input modalities were used, for example, only coughing audio or only speech audio, the binary classification accuracies were much lower. When all available input modalities were used together as multimodal inputs, the binary classification accuracy for the CNN baseline model increased. The baseline models' accuracy for case-study 2 is 94.1%. Then after compressing the model with CSC layers and reducing the data bit width to hybrid bit precision settings, the accuracy degrades to 92.5% which is 1.6% less compared to full precision models. Multimodal models are memory intensive due to their simultaneous storing and processing of different modalities. CSC layers and hybrid quantization help here to reduce the to reduces the computation and the memory cost from

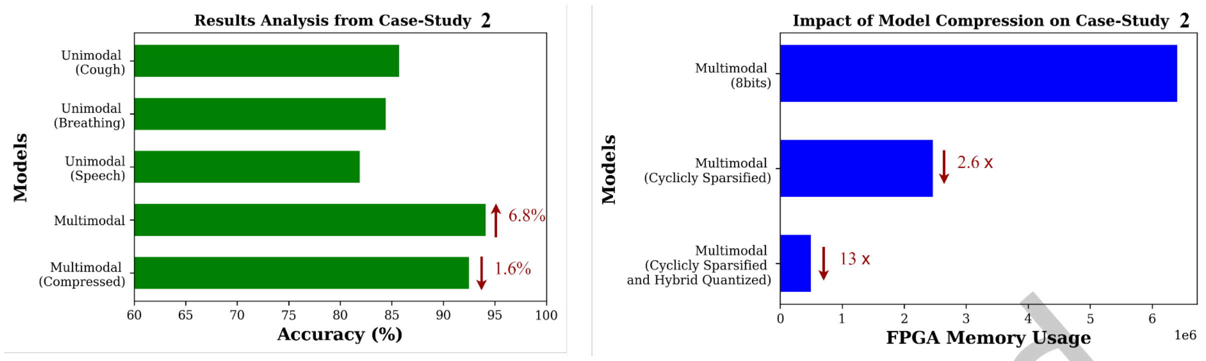

Fig. 9. (a) TinyM^{2}Net-V2 classification results for Case-Study 2 in terms of both unimodal and multimodal settings. The multimodal setting improved 6.8% accuracy compared to unimodal (speech) classification setting. Model compression techniques reduce 1.6% accuracy of the multimodal setting. (b) Impact of model compression on TinyM^{2}Net-V2 for Case-Study 2. Using only CSC reduces 2.6× FPGA memory usage from baseline model. Using both CSC and hybrid quantization reduces 13× FPGA memory usage baseline model.

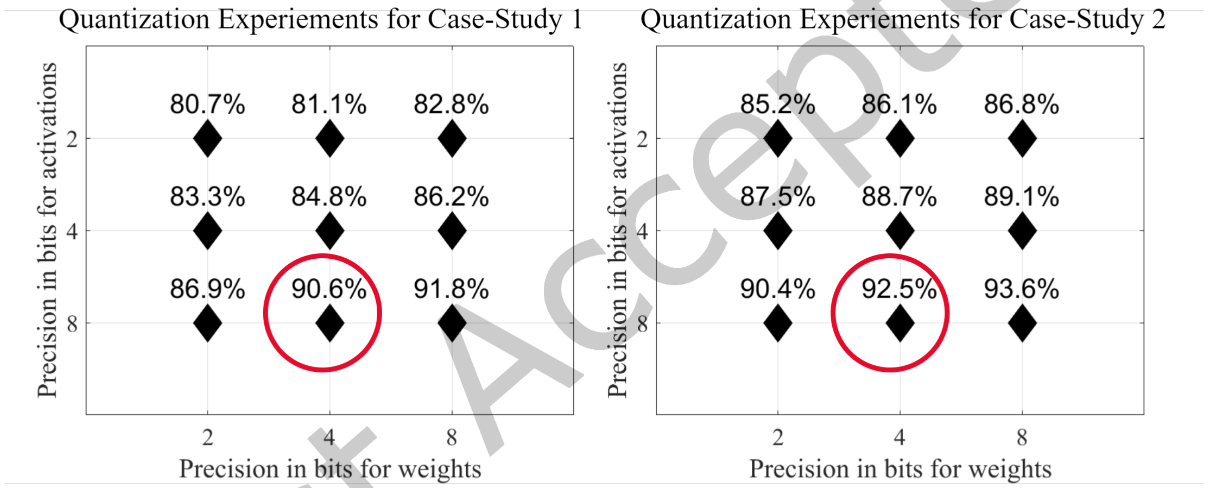

Fig. 10. Empirical Study for Hybrid Quantization of Case-Study 1 and Case-Study 2.

$O(N^2)$ to $O(NlogN)$. This 1.6% performance degradation due to CSC layers and quantization is still allowable, considering the energy-efficient tinyML deployment for resource-constrained tiny POC hardware. In figure 9(b), the FPGA memory requirement for baseline model is 6.4 Mbits. With CSC compression and hybrid precision, the memory requirement is compressed by 13× to 0.4 Mbits.

To select the bit precision for our hybrid precision models we experimented with both the case-studies and selected bit precision settings empirically. Figure 10 presents our experimental results for different bit precision settings. We can see, 8 bits of activation and 4 bits of weights gives us optimal amount of accuracy for both the case-studies. This is the reason we have selected 4 bits of weights and 8 bits of activation throughout our hybrid quantized model.

4.3 FPGA Implementation and Results

Given a variety of parameters to tweak in the accelerator to obtain the most energy-efficient hardware configuration, we use an Energy Efficiency vs Number of PE curve. It allows us to select a configuration that provides

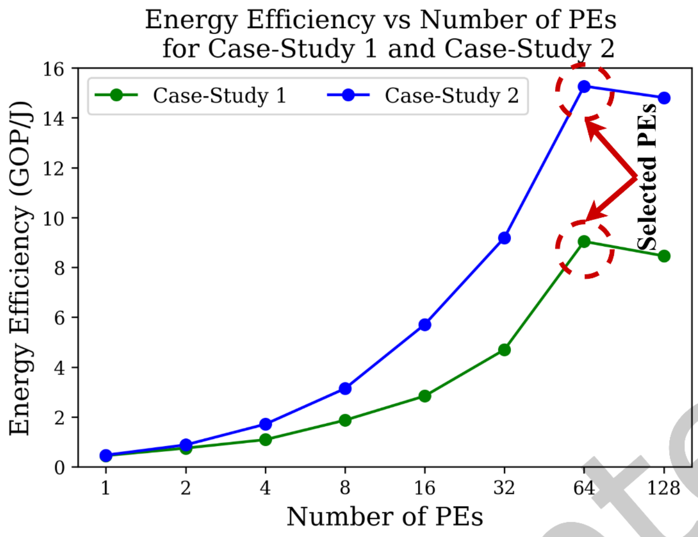

Fig. 11. TinyM^{2}Net-V2 Energy Efficiency cs Number of PEs for the case study 1 and case study 2. Highlighted in red is the most energy efficient configuration.

us the highest performance for the least hardware utilization. Figure 11, summarizes the results for the Energy Efficiency vs Number of PEs for Case-Study 1 and Case-Study 2 for the TinyM^{2}Net-V2 hardware architecture. For both the case studies we increased the number of PEs and measured the energy efficiency. Plotting a curve for this reveals a ridge point until where the energy efficiency increases consistently with each new PE added. Beyond this point further increasing in PEs does not contribute to increase in parallaization as there are PEs that are being under utilized. For the case studies here, we obtain the highest Energy Efficiency for the 64PE configuration. The enrgy efficiency for Case Study 1 is 7.62 GOP/s/W and the enrgy efficiency for Case Study 2 is 15.38 GOP/s/W.

The compressed model along with its corresponding parameters is loaded into the configuration memory of the accelerator. The design was then implemented on the Xilinx Artix-7 FPGA at a clock frequency of 100MHz using the Xilinx Vivado 2018.3 design suite. Various measurements such as power, fpga utilization, latency etc. are also obtained from the Vivado suite using appropriate switching activity files obtained using well-written test-benches. The hardware for the compressed network is configured to have a total of 64 PEs and 1 MAC per PE. The accelerator has also been configured to work with 4 bit quantized weights and 8 bit quantized features.

Our target FPGA platform is the Xilinx Artix-7 FPGA (xc7a200t) to demonstrate the feasibility of achieving milliwatts of power for running tiny multimodal models. It has 134.6K LUTs, 1.6MB of on-chip block RAM (BRAM) and 740 DSP slices. To implement baseline models for our case-studies, 32.6MB of on-chip block RAM (BRAM) is required (for case-study 1). We highly compress the baseline models using model compression techniques mentioned in the previous sections, so that we could fit the models into low power Artix-7 FPGAs. Table 3 shows the utilization for the proposed TinyM^{2}Net-V2 for both the case studies. Our design requires 96.7 % and 25 % of the available BRAM to store the weights, input and the output feature maps for case-study 1 and 2 respectively. It also requires 18 % and 30 % of the available LUTs for case-study 1 and 2 respectively.

Minimizing power consumption on an Artix-7 FPGA can be achieved through a combination of design techniques and device-specific features. Some techniques employed in this work to minimize power consumption on the Artix-7 FPGA to mW range are but not limited to the following:

Table 3. Resource utilization data of TinyM^{2}Net-V2 implemented on Artix -7 FPGA (XC7A200t) at 100 MHz with 64 PE configuration.

Resources	LUTs	BRAMs	DSPs
Available	134.6K	365	740
Case-Study 1 Utilization	24K	353	71
Case-Study 2 Utilization	40K	90	71

Table 4. Implementation results and comparisons of the proposed TinyM^{2}Net-V2 with its baseline hardware designs. The baseline results are obtained for uniform 8 bits quantized models at a clock frequency of 100 MHz as full precision models do not fit in target Artix-7 FPGA board.

	Case-Study 1		Case-Study 2	
Model Compression	Uncompressed (Uniform 8 bits)	Compressed (CSC + 8/4 bits)	Uncompressed (Uniform 8 bits)	Compressed (CSC + 8/4 bits)
No. of P.E.	64			
Freq. (MHz)	100			
Power (W)	1.28	0.94	0.65	0.52
Latency (ms)	175.9	49.5	3.65	1.3
Energy (mJ)	225.16	46.53	2.37	0.67
Performance (GOP/s)	4.2	8.5	7.8	8
Energy Efficiency (GOP/s/W)	3.28	9.04	12.01	15.38

- Optimal design for the hardware architecture
- Efficiently using of the available hardware resources - for instance, the amount of power block RAM consumes is directly proportional to the amount of time it is enabled. To save power, the block RAM enable can be driven Low on clock cycles when the block RAM is not used in the design.
- Design level decisions such as, minimizing the asynchronous control signals to minimize the use of routing resources.
- Pipelining the design to minimize the size of the combinatorial cones.
- Sharing resources such as the BRAMs. Once the feature map values have been served the same set of locations can hold new values. Such designs minimizes the amount of BRAMs consumed.
- Block RAM (BRAM) rich designs are excellent candidates for power savings as the Vivado tool uses a variety of optimization techniques to generate 'enables' and in-turn save power.

Static power consumption for Artix-7 board is 140 mW. The table 5 shows that our FPGA implementation consumes total power of 0.94 W for case-study 1 (static 140 mW and dynamic 808 mW) and 0.52 W for case-study 2 (static power = 140 mW and dynamic power = 384 mW), which are less than tinyML power consumption limit of 1W. Figure 12 shows the total dynamic power breakdown per FPGA resources of TinyM^{2}Net-V2 for both the case-studies. With the highest utilization of the available BRAMs, our design consumes 77 % and 46 % of the total dynamic power for the BRAMs utilization in case study 1 and 2 respectively. It also consumes 4 % and 10 % of the total dynamic power for the Logic in case study 1 and 2 respectively. Table 4 describes a comparison with the baseline models of the same applications with our adopted model compression techniques to show the improvement using our intended model compression techniques. As per the expectation, our proposed model achieves a reduction in power consumption and improved energy efficiency from its baseline counterpart

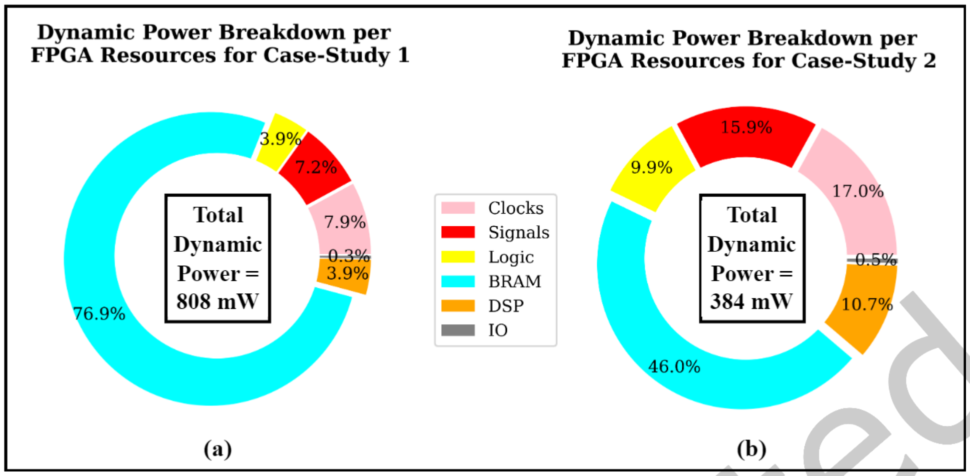

Fig. 12. Dynamic power breakdown per FPGA resources of TinyM^{2}Net-V2 for case-studies 1 and 2. Both of the implementations consume 140 mW of static power on Artix-7 board.

4.4 Comparison with State-of-the-Art Implementations

The table 5 presents the FPGA implementation results for both of our case-studies and also comparison with the recent state-of-the-art FPGA accelerators [27, 51] while keeping the application category the same. At 100 MHz frequency, we achieved 49.5 ms latency for vehicle classification and 1.3 ms latency for COVID-19 detection. Our implementation has 9.04 GOP/s/W energy efficiency for case-study 1 and 15.38 GOP/s/W energy efficiency for case-study 2.

Both [51] and our case-study 1 are performing image classification task in high level. Although [51] is faster with than ours in terms of latency and computations, we achieve better power consumption than that of [51] (1.01 W vs 0.94 W) due to our highly compressed models. Our implementation has 5.47× better energy-efficiency when compared to the [51] implementation.

Both [27] and our case-study 2 are performing multimodal audio classification task in high level. Although [27] is smaller than ours in terms of model size (40 KB vs 62 KB), our computations are highly reduced due to CSC architecture. [27] model has 6 Giga Operations (GOP), whereas our case-study 2 model has only 0.01 GOP. This results in 3.13× improved energy-efficiency when our implementation is compared to the [27] implementation. Our hardware implementation is much faster compared to [27] which is very important for real-time audio classification systems.

4.5 Comparison with Off-The-Shelf Device Implementations

To establish a point of reference, we also deployed our proposed TinyM^{2}Net-V2 models on resource-constrained commodity edge hardware, NVIDIA Jetson TX2 and Raspberry Pi 4 and measured hardware merits such as latency and power consumption. All platforms were configured to perform at their peak performance. The implementation result is summarized in Table 6. From Table 6, it can obviously be said that our tiny FPGA implementation outperforms any other edge hardware by a comparable margin. Our FPGA implementation is 25 × faster from the Raspberry Pi 4 and 3.23 × from the NVIDIA CPU-GPU configuration for case-study 1. Our FPGA implementation is 753 × faster from the Raspberry Pi 4 and 76.9 × from the NVIDIA CPU-GPU configuration for case-study 2. Our tiny FPGA power consumption is below 1W which is the power consumption limit for state-of-the-art tinyML devices.

Table 5. Implementation results and comparisons of the proposed TinyM²Net-V2 with state-of-the-art CNN hardware designs. The results of our work are obtained for hybrid (4bits weights and 8bits activations) quantized CSC compressed multimodal neural network at a clock frequency of 100 MHz.

Architecture	[51]	[27]	**This Work**	
FPGA platform	Zync XC7Z020	Artix 7	Artix 7	Artix 7
Application Type	Image Classification	Audio Classification	Image Classification	Audio Classification
Model Size (KB)	-	40	932	62
Computations (GOP)	0.02	6	0.42	0.01
Frequency (MHz)	100	80	100	
Latency (ms)	16	5000	49.5	1.3
Power (W)	1.01	0.24	0.94	0.52
Energy (mJ)	33.7	1.2	46.53	0.67
Performance (GOP/s)	1.67	1.2	8.5	8
Energy Efficiency (GOP/s/W)	1.65	4.9	9.04	15.38

Table 6. TinyM²Net-V2 hardware implementation results on the various commercial off-the-shelf edge devices.

Edge Devices	NVIDIA TX2 (with GPU)		Raspberry Pi 4		Artix-7 FPGA	
Case-Study	1	2	1	2	1	2
Frequency (MHz)	CPU: 2035 GPU: 1300		1500		100	
Latency (ms)	160	100	1240	980	49.5	1.3
Power (mW)	9856	8876	1567	994	948	524

5 FUTURE WORKS

TinyM²Net-V2 is a compact and low-power software and hardware architecture for multi-modal deep neural networks that can run on resource-constrained tiny devices. The field of tiny machine learning, which deals with deep learning models for ultra-low power devices, has primarily focused on a train-and-deploy approach. This results in static models that cannot be adjusted based on new data without cloud-based retraining. To make multi-modal tiny machine learning devices more prevalent, there is a growing need for adaptive methods that can update the model parameters in response to changes in the environment, sensors, and input data. However, the limited computational resources on these devices pose a significant challenge to implementing adaptive online learning methods, which typically require substantial computational and memory resources. One of the future direction of this research would be implementation of adaptive online learning method for resource-constrained edge devices.

Hardware-agnostic model compression refers to the process of reducing the size of a deep learning model in a manner that is not specific to a particular hardware platform. This approach is aimed at making the compressed model more suitable for deployment on resource-constrained devices. However, this type of model compression can often result in a deterioration of both model accuracy and performance. This is because the compression technique used is not specifically tailored to the hardware platform, and as a result, the compressed model may not perform optimally on the target hardware. The compression process may also lead to a loss of information and a reduction in accuracy, which can negatively impact the performance of the model. Thus, hardware-agnostic model compression may not be the best approach for achieving high accuracy and performance on resource-constrained devices. To address this research gap, hardware aware model compression techniques would be addressed in the future of this research. Hardware-aware quantization and pruning would be implemented on tiny machine learning hardware systems to achieve better performance. Impact of choice of quantization such as linear, biModel, trained quantization, ternary connect etc. would be interesting study for this research in the future. In practical applications like autonomous driving cars, hardware constraints often prove to be a significant obstacle in fully utilizing multimodal neural network models. Additionally, designing a manual architecture that takes into account hardware limitations and diversity is a challenging task. Hence, hardware-aware multimodal architecture search is an area of future research that holds promise in this field. We would address these research gaps to make more robust TinyM^{2}Net in future.

6 CONCLUSION

In this paper, we introduce TinyM^{2}Net-V2, a compact and low-power software and hardware architecture for multi-modal deep neural networks that can run on resource-constrained tiny devices. We are the first to show the effectiveness of model compression strategies for multimodal deep neural networks employing cyclically sparsification and hybrid quantization of weights/activations. We evaluated our TinyM^{2}Net-V2 with two distinct case studies: COVID-19 detection from multimodal audio recordings and vehicle detection from multimodal images and audios. The most compressed TinyM^{2}Net-V2 achieves 92.5% accuracy in COVID-19 detection (6.8% improvement over the unimodal full precision model) and 90.6% accuracy in vehicle classification (7.7% improvement over the unimodal full precision model). For TinyM^{2}Net-V2 models, a parameterized and adaptable FPGA hardware accelerator was also created. This is the first effort, as far as we are aware, at accelerating multimodal deep neural network models on the low-power Artix-7 FPGA devices. For vehicle classification task and COVID-19 detection task, respectively, we obtained energy efficiencies of 9.04 GOP/s/W and 15.38 GOP/s/W, which are comparable to results from leading-edge research. Finally, we compared our tiny FPGA hardware implementation results with off-the-shelf resource-constrained devices and showed our implementation is faster and consumed less power compared to the off-the-shelf resource-constrained devices.

7 ACKNOWLEDGEMENT

This research was partly supported by the National Science Foundation CAREER Award under Grant No. 1652703. We also acknowledge the partial support of the University of Maryland, Baltimore, Institute for Clinical & Translational Research (ICTR) and the National Center for Advancing Translational Sciences (NCATS) Clinical Translational Science Award (CTSA) grant number UL1TR003098.

REFERENCES

[1] Hande Alemdar, Vincent Leroy, Adrien Prost-Boucle, and Frédéric Pétrot. 2017. Ternary neural networks for resource-efficient AI applications. In *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2547–2554.
[2] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13, 3 (2017), 1–18.
[3] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. 2021. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems* 3 (2021).
[4] George Barnum, Sabera Talukder, and Yisong Yue. 2020. On the Benefits of Early Fusion in Multimodal Representation Learning. *arXiv preprint arXiv:2011.07191* (2020).
[5] Claudionor N Coelho, Aki Kuusela, Shan Li, Hao Zhuang, Jennifer Ngadiuba, Thea Klaeboe Aarrestad, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, and Sioni Summers. 2021. Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nature Machine Intelligence* 3, 8 (2021), 675–686.
[6] Emon Dey and Nirmalya Roy. 2020. OMAD: On-device Mental Anomaly Detection for Substance and Non-Substance Users. In *2020 IEEE 20th International Conference on Bioinformatics and Bioengineering (BIBE)*. 466–471. https://doi.org/10.1109/BIBE50027.2020.00081
[7] Changxing Ding and Dacheng Tao. 2015. Robust face recognition via multimodal deep face representation. *IEEE Transactions on Multimedia* 17, 11 (2015), 2049–2058.

[8] Igor Fedorov, Marko Stamenovic, Carl Jensen, Li-Chia Yang, Ari Mandell, Yiming Gan, Matthew Mattina, and Paul N Whatmough. 2020. TinyLSTMs: Efficient neural speech enhancement for hearing aids. *arXiv preprint arXiv:2005.11138* (2020).

[9] Konrad Gadzicki, Razieh Khamsehashari, and Christoph Zetzsche. 2020. Early vs late fusion in multimodal convolutional neural networks. In *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*. IEEE, 1–6.

[10] Zahid Hasan, Emon Dey, Sreenivasan Ramasamy Ramamurthy, Nirmalya Roy, and Archan Misra. 2022. RhythmEdge: Enabling Contactless Heart Rate Estimation on the Edge. In *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*. 92–99. https://doi.org/10.1109/SMARTCOMP55677.2022.00028

[11] Morteza Hosseini, Mark Horton, et al. 2019. On the Complexity Reduction of Dense Layers from $O(N^2)$ to $O(NlogN)$ with Cyclic Sparsely Connected Layers. In *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM.

[12] Morteza Hosseini, Nitheesh Kumar Manjunath, Bharat Prakash, Arnab Mazumder, Vandana Chandrareddy, Houman Homayoun, and Tinoosh Mohsenin. 2021. Cyclic Sparsely Connected Architectures for Compact Deep Convolutional Neural Networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 10 (2021), 1757–1770.

[13] Morteza Hosseini and Tinoosh Mohsenin. 2021. QS-NAS: Optimally Quantized Scaled Architecture Search to Enable Efficient On-Device Micro-AI. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2021).

[14] Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. 2020. Improving post training neural quantization: Layer-wise calibration and integer programming. *arXiv preprint arXiv:2006.10518* (2020).

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

[16] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866* (2014).

[17] Gargi Joshi, Rahee Walambe, and Ketan Kotecha. 2021. A review on explainability in multimodal deep neural nets. *IEEE Access* 9 (2021), 59800–59821.

[18] Ryan Kiros, Karteek Popuri, Dana Cobzas, and Martin Jagersand. 2014. Stacked multiscale feature learning for domain independent medical image segmentation. In *International workshop on machine learning in medical imaging*. Springer, 25–32.

[19] Carl Lemaire, Andrew Achkar, and Pierre-Marc Jodoin. 2019. Structured pruning of neural networks with budget-aware regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9108–9116.

[20] Ian Lenz, Honglak Lee, and Ashutosh Saxena. 2015. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research* 34, 4-5 (2015), 705–724.

[21] Shuang Liang, Shouyi Yin, Leibo Liu, Wayne Luk, and Shaojun Wei. 2018. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* 275 (2018), 1072–1086.

[22] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319* (2020).

[23] Siqi Liu, Sidong Liu, Weidong Cai, Hangyu Che, Sonia Pujol, Ron Kikinis, Dagan Feng, Michael J Fulham, et al. 2014. Multimodal neuroimaging feature learning for multiclass diagnosis of Alzheimer's disease. *IEEE transactions on biomedical engineering* 62, 4 (2014), 1132–1140.

[24] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3431–3440.

[25] Arnab Neelim Mazumder, Jian Meng, Hasib-Al Rashid, Utteja Kallakuri, Xin Zhang, Jae-sun Seo, and Tinoosh Mohsenin. 2021. A Survey on the Optimization of Neural Network Accelerators for Micro-AI On-Device Inference. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2021).

[26] Arnab Neelim Mazumder, Haoran Ren, Hasib-Al Rashid, Morteza Hosseini, Vandana Chandrareddy, Houman Homayoun, and Tinoosh Mohsenin. 2021. Automatic Detection of Respiratory Symptoms Using a Low Power Multi-Input CNN Processor. *IEEE Design Test* (2021), 1–1. https://doi.org/10.1109/MDAT.2021.3079318

[27] Arnab Neelim Mazumder, Haoran Ren, Hasib-Al Rashid, Morteza Hosseini, Vandana Chandrareddy, Houman Homayoun, and Tinoosh Mohsenin. 2021. Automatic Detection of Respiratory Symptoms Using a Low Power Multi-Input CNN Processor. *IEEE Design & Test* (2021).

[28] Jian Meng, Shreyas Kolala Venkataramanaiah, Chuteng Zhou, Patrick Hansen, Paul Whatmough, and Jae-sun Seo. 2021. FixyFPGA: Efficient FPGA Accelerator for Deep Neural Networks with High Element-Wise Sparsity and without External Memory Access. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 9–16.

[29] Sankha S Mukherjee and Neil Martin Robertson. 2015. Deep head pose: Gaze-direction estimation in multimodal video. *IEEE Transactions on Multimedia* 17, 11 (2015), 2094–2107.

[30] Mozhgan Navardi, Prakhar Dixit, Tejaswini Manjunath, Nicholas R Waytowich, Tinoosh Mohsenin, and Tim Oates. 2022. Toward Real-World Implementation of Deep Reinforcement Learning for Vision-Based Autonomous Drone Navigation with Mission. *UMBC Student Collection* (2022).

[31] Mozhgan Navardi, Edward Humes, and Tinoosh Mohsenin. 2022. E2EdgeAI: Energy-Efficient Edge Computing for Deployment of Vision-Based DNNs on Autonomous Tiny Drones. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*. 504–509. https://doi.org/10.1109/SEC54971.2022.00077

[32] Mozhgan Navardi, Aidin Shiri, Edward Humes, Nicholas R Waytowich, and Tinoosh Mohsenin. 2022. An Optimization Framework for Efficient Vision-Based Autonomous Drone Navigation. In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 304–307.

[33] Pretom Roy Ovi et al. 2021. ARIS: A Real Time Edge Computed Accident Risk Inference System. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*. 47–54. https://doi.org/10.1109/SMARTCOMP52413.2021.00027

[34] Pretom Roy Ovi, Emon Dey, Nirmalya Roy, Aryya Gangopadhyay, and Robert F Erbacher. 2022. Towards developing a data security aware federated training framework in multi-modal contested environments. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications IV*, Vol. 12113. SPIE, 189–198.

[35] Hasib-Al Rashid, Pretom Roy Ovi, Aryya Busart, Carl Gangopadhyay, and Tinoosh Mohsenin. 2022. TinyM2Net: A Flexible System Algorithm Co-designed Multimodal Learning Framework for Tiny Devices. *ArXiv* (2022).

[36] Hasib-Al Rashid, Mohammad M Sajadi, and Tinoosh Mohsenin. 2022. CoughNet-V2: A Scalable Multimodal DNN Framework for Point-of-Care Edge Devices to Detect Symptomatic COVID-19 Cough. In *2022 IEEE Healthcare Innovations and Point of Care Technologies (HI-POCT)*. IEEE, 37–40.

[37] Neeraj Sharma et al. 2020. Coswara–A Database of Breathing, Cough, and Voice Sounds for COVID-19 Diagnosis. (2020).

[38] Neeraj Kumar Sharma, Srikanth Raj Chetupalli, Debarpan Bhattacharya, Debottam Dutta, Pravin Mote, and Sriram Ganapathy. 2021. The Second DiCOVA Challenge: Dataset and performance analysis for COVID-19 diagnosis using acoustics. *arXiv preprint arXiv:2110.01177* (2021).

[39] Martin Simonovsky, Benjamín Gutiérrez-Becker, Diana Mateus, Nassir Navab, and Nikos Komodakis. 2016. A deep metric for multimodal registration. In *International conference on medical image computing and computer-assisted intervention*. Springer, 10–18.

[40] Sören Richard Stahlschmidt, Benjamin Ulfenborg, and Jane Synnergren. 2022. Multimodal deep learning for biomedical data fusion: a review. *Briefings in Bioinformatics* 23, 2 (2022), bbab569.

[41] Ahmet Ali Süzen, Burhan Duman, and Betül Şen. 2020. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. IEEE, 1–5.

[42] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[43] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2818–2826.

[44] Yapeng Tian et al. 2018. Audio-visual event localization in unconstrained videos. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 247–263.

[45] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8612–8620.

[46] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. 2018. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090* (2018).

[47] Pengcheng Wu, Steven CH Hoi, Hao Xia, Peilin Zhao, Dayong Wang, and Chunyan Miao. 2013. Online multimodal deep similarity learning with application to image retrieval. In *Proceedings of the 21st ACM international conference on Multimedia*. 153–162.

[48] Kunran Xu, Yishi Li, Huawei Zhang, Rui Lai, and Lin Gu. 2022. EtinyNet: Extremely Tiny Network for TinyML. (2022).

[49] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. 2021. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*. PMLR, 11875–11886.

[50] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello edge: Keyword spotting on microcontrollers. *arXiv preprint arXiv:1711.07128* (2017).

[51] Guanwen Zhong, Akshat Dubey, Cheng Tan, and Tulika Mitra. 2019. Synergy: An hw/sw framework for high throughput cnns on embedded heterogeneous soc. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 2 (2019), 1–23.