OWSNet: Towards Real-time Offensive Words Spotting Network for Consumer IoT Devices

Bharath Sudharsan*, Sweta Malik*, Peter Corcoran‡, Pankesh Patel*†, John G. Breslin*, Muhammad Intizar Ali§
*Confirm SFI Centre for Smart Manufacturing, Data Science Institute, NUI Galway, Ireland
{bharath.sudharsan, sweta.malik, pankesh.patel, john.breslin}@insight-centre.org
‡School of Engineering and Informatics, NUI Galway, Ireland, peter.corcoran@nuigalway.ie
†Artificial Intelligence Institute, University of South Carolina, Columbia USA, ppankesh@mailbox.sc.edu
§School of Electronic Engineering, Dublin City University, Ireland, ali.intizar@dcu.ie

***Abstract*— Every modern household owns at least a dozen of IoT devices like smart speakers, video doorbells, smartwatches, where most of them are equipped with a Keyword spotting (KWS) system-based digital voice assistant like Alexa. The state-of-the-art KWS systems require a large number of operations, higher computation, memory resources to show top performance. In this paper, in contrast to existing resource-demanding KWS systems, we propose a light-weight temporal convolution based KWS system named *OWSNet*, that can comfortably execute on a variety of IoT devices around us and can accurately spot multiple keywords in real-time without disturbing the device's routine functionalities.**

When *OWSNet* is deployed on consumer IoT devices placed in the workplace, home, etc., in addition to spotting wake/trigger words like 'Hey Siri', 'Alexa', it can also accurately spot offensive words in real-time. If regular wake words are spotted, it activates the voice assistant; else if offensive words are spotted, it starts to capture and stream audio data to speech analytics APIs for autonomous threat and insecurities detection in the scene. The evaluation results show that the *OWSNet* is faster than state-of-the-art models as it produced $\approx$ *1-74 times* faster inference on Raspberry Pi 4 and $\approx$ *1-12 times* faster inference on NVIDIA Jetson Nano. In this paper, to optimize IoT use-case models like *OWSNet*, we present a generic multi-component ML model optimization sequence that can reduce the memory and computation demands of a wide range of ML models thus enabling their execution on low resource, cost, power IoT devices.

***Index Terms*—IoT Devices, Edge Intelligence, Keyword Spotting, Temporal Convolutions, Model Optimization.**

I. Introduction

Keyword spotting (KWS) in voice-based digital assistants is the first and most essential step where pre-defined keywords are detected from a stream of live audio data captured by a single microphone or an array of microphones. A high-performance low latency KWS system/model is essential to achieve a *true hands-free interaction* with IoT applications or AI assistants that control thermostats, projectors, lights, shades, doors. However, there are multiple challenges when implementing such a fast and accurate keywords spotting model on IoT devices that usually have limited hardware resources [1]. For example, the basic Alexa smart speaker models roughly have Quad-core, 64-bit ARM Cortex-A35, 8GB Flash, 2GB RAM, which is sufficient to run the top KWS models. But due to the high computational demands of top KWS models, the devices face the issue of delayed detection of keywords (due to higher model execution time), which impacts the responsiveness of the onboard Alexa Voice Service (AVS) application, resulting in degraded human-Alexa interaction quality.

We define *Offensive language/words as a part of hate speech spoken to express hatred on individuals or a targeted group of people to make them feel humiliated or insulted. In extreme cases, offensive language is used during threatening or when inciting violence*. Numerous studies have designed central models that can autonomously detect multilingual hate content on social media platforms like Twitter, Tumbler, Facebook. But currently, to the best of our knowledge, there is no system existing in any of the real human environments such as offices, residence, factory floors to monitor speech data and detect offensive words targeted to attack people/co-workers based on their attributes like race, gender, ethnicity, sexual orientation. Hence, in order to maintain a healthy verbal environment and avoid harmful incidents, there is a strong need for a distributed intelligent system in human environments to police the usage of language.

In this paper, we present *OWSNet*, a resource-friendly temporal convolutions based KWS system that can comfortably execute on a wide variety of microphone-based IoT devices existing around us. The *OWSNet*, without disturbing their routine functionalities, can spot multiple keywords (both offensive words and voice assistant wake words) in real-time with high accuracy. If regular voice assistant wake words are spotted, it activates the onboard AVS application; else if offensive words are spotted, *OWSNet* will start capturing audio data from the scene and stream it to APIs that perform autonomous threat detection and alerting by analyzing the transmitted speech data. The *OWSNet* guarantees the user privacy since it only utilizes that data that IoT devices already have access to. i.e., devices already have access to audio data for spotting wake words like 'Hey Siri', 'Alexa', in order to wake up/trigger voice assistants. The main contributions of this paper can be summarized as follows:

- We propose a resource-friendly multiple KWS network named *OWSNet* that can comfortably execute on mid-sized small CPU-based IoT devices. The temporal convolutions in our designed network reduce the computational strain during execution while showing superior inference (speedups) performance than the state-of-the-art KWS models.
- The proposed network can learn using various voice datasets like the standard Google Speech Commands

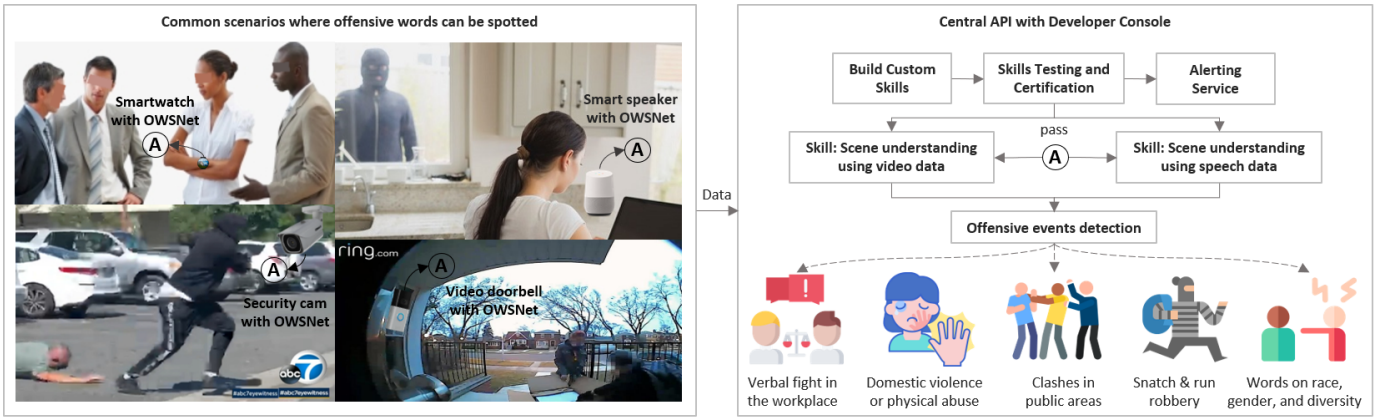

Fig. 1. The resource-friendly *OWSNet* is executable on a broad-spectrum of IoT devices and can accurately spot multiple keywords in real-time. If an utterance of offensive words is spotted, it starts streaming audio/video data to analytics APIs for autonomous threat and insecurities detection in the scene.

dataset and also hate speech audio datasets that contain voice recordings of multiple offensive words.

- We provide a multi-component ML model optimization sequence to optimize various CNNs for making it executable on resource-constrained hardware. We apply our sequence on standard CNNs, also on *OWSNet* and discuss how our sequence enables the execution of a wide range of ML models on tiny embedded systems like IoT devices.

Outline. The rest of the paper is organized as follows. In Section II, we present the complete *OWSNet* with its evaluation results. Then in Section III, we propose an end-to-end multi-component ML model optimization sequence and use it to optimize the *OWSNet*. In Section V, we summarize by providing greater context for future research.

II. Offensive Words Spotting

Although offensive words are not used in most cultured environments, yet there are ever-increasing reports and social media posts about the use of hate speech. It is evident from the recent events [2] that even people from dignifiable positions utter offensive words to humiliate minor community people or targetted individuals. Invariable of the environment (work, home, social gatherings), there are many examples where the utterance of just one offensive word had ended up in disputes between people. Also, in many scenarios, offensive language is spoken during threatening or when inciting violence. Hence, there is a strong need for a distributed intelligent system to exist around people to police the language and raise timely alerts to avoid harmful incidents.

As IoT devices have become ubiquitous in most environments, every person is close to at least one device such as a smartphone, smartwatch, smart speaker, etc. Hence, we aim to deploy our resource-friendly *OWSNet* on IoT devices (rather than other platform devices) in order to provide them the ability to detect various offensive words. After detection, using the IoT device in the scene, we capture and stream the audio and/or video data to APIs for performing autonomous detection of threats and insecurities. For example, as shown in Fig. 1, after hate words detection by the IoT devices equipped with *OWSNet*, the data stream is received by the API. These APIs that perform the advanced analytics should contain models or certified skills that can understand and detect offensive events using the streamed data. For example, in [3], they have deployed a custom skill on the central Amazon developer console and interfaced it with their Alexa smart speaker prototype. Whenever their custom skill spots the command *Alexa, ask Friday what she sees*, the skill activates the smart speaker camera, executes an ML model on the device, and returns the names of the objects in the device's field of view. In Fig. 1, we show a few example scenarios where offensive words can be spotted and lead to undesired incidents. In the following, we brief how *OWSNet* can improve the safety of people in such scenarios.

Smart speakers with OWSNet. In today's modern households, multiple devices are distributedly located across the house, and devices with AI assistants are usually placed within 5 meters distance from the people to capture speech commands with greater audio detail. When household IoT devices are equipped with *OWSNet*, during the occurrence of undesirable events such as domestic violence or physical abuse, at least one device in the home would spot the offensive words, then stream the speech data to API skills for offensive events detection and alerting.

Smartwatches with OWSNet. Usually in casual meetings and outdoor gatherings there would be less presence of IoT devices. In such cases, smartwatches with *OWSNet*, can spot the utterance of offensive words on race, gender, diversity, etc., then stream speech data to API for detection of offensive events by performing advanced speech analytics.

Security cameras with OWSNet. Wide landscape infrastructures such as parking spaces, factory assembly lines, etc., contain blind spots since the few installed cameras cannot cover the entire area due to their limited field of view. When camera-based devices such as video doorbells, IP cameras, etc., are equipped with *OWSNet*, they can detect offensive words. Then, based on the estimated Direction of Arrival (DOA) of the detected words, the camera can steer or change its focus point facing the incident. Such DOA-based camera steering enables the provision of visual data after detection of hate

words from the scene, thus enabling API skills to identify offensive events with greater accuracy.

There are various other scenarios where the alerts raised by *OWSNet* equipped devices can save people who are about to fall prey for theft, racial discrimination, get caught in public clashes, etc. Since the devices with *OWSNet* start to stream audio and/or video data to APIs, the users can enable the option of storing the conversation or visual scene that can later be used as evidence to prove the offense caused to the victim. Also, such autonomous detection, analytics, and recording are critical in emergencies where the victim does not get the chance to record the audio or video when attacked. For example, in Fig. 1, in the scenario where a burglar is trying to enter the house or when the man is getting robbed in the parking spot, the victims might not instantly think and act to start the recording/streaming. In such cases, the offender would have had used offensive words, which *OWSNet* model can spot and stream the data from the offense scene to its interlinked API.

A. Offensive Words Spotting Network (OWSNet) Design

Here we describe and compare 2D convolutions with temporal convolutions, then present the *OWSNet* architecture.

1) 2D vs Temporal Convolutions: In a typical 2D convolution based network design, for the given input $\mathbf{X} \in \mathbb{R}^{w \times h \times c}$ and network weights $\mathbf{W} \in \mathbb{R}^{k_w \times k_h \times c \times c'}$, the standard 2D convolution outputs $\mathbf{Y} \in \mathbb{R}^{w \times h \times c'}$. Most such CNN-based KWS models are fed with a 2D Mel-Frequency Cepstral Coefficients (MFCC) as the input using which the raw audio is transformed into a time-frequency representation $\mathbf{I} \in \mathbb{R}^{t \times f}$ where, t is the time and f is the extracted feature from frequency domain. Most existing studies [4,5] use input tensor $\mathbf{X} \in \mathbb{R}^{w \times h \times c}$ where $c = 1$, $t = w$, $f = h$ and since thier model design is based on 2D convolution, their input tensor becomes $\mathbf{X}_{2\text{ d}} \in \mathbb{R}^{t \times f \times 1}$.

Since we intend to design a model that is fast executing (for real-time KWS results) and also accurate, we reshape the input from $\mathbf{X}_{2\text{ d}}$ into $\mathbf{X}_{1\text{ d}}$. Next, instead of using an intensity or grayscale image as an input (a widely used method to interpret audio data), we consider the MFCC of each frame as time-series data. We next consider $\mathbf{I}$ as a 1D sequence with its data features represented as f. In this step, instead of converting $\mathbf{I}$ to $\mathbf{X}_{2\text{ d}}$, we set $h = 1$ and $c = f$ which results in producing $\mathbf{X}_{1\text{ d}} \in \mathbb{R}^{t \times 1 \times f}$ and feed it to the temporal convolution.

When applying this method during the KWS network design, the resultant model will be smaller in size. When the 2D convolutions with $\mathbf{W}_{2\text{ d}} \in \mathbb{R}^{3 \times 3 \times 1 \times c}$ and temporal convolutions with weights $\mathbf{W_{1d}} \in \mathbb{R}^{3 \times 1 \times f \times c'}$ have the same parameters count, the temporal convolutions can execute faster on IoT devices since it has a smaller number of computations than the traditional 2D convolutions. Similarly, the output $\mathbf{Y}_{1\text{ d}} \in \mathbb{R}^{t \times 1 \times c'}$ from the first layer of a temporal convolutions-based network is smaller than $\mathbf{Y}_{2\text{ d}} \in \mathbb{R}^{t \times f \times c}$ which is the output of a 2D convolutional layer. This size reduction also reduces the model's computational complexity and memory footprint, resulting in faster model execution (i.e., spots keywords in real-time).

TABLE I
COMPARING THE PERFORMANCE OF TOP MODELS WITH *OWSNet*. THE BEST RESULT IN EACH COLUMN IS IN BOLD.

Network	FLOPs	Total Params	Inference Time (ms)		Accuracy (%)
			D1	D2	
CNN-stride4 [6]	**1.5M**	148K	1.9	1.1	84.6
CNN-fpool3 [6]	76.1M	524K	39.1	2.7	90.7
DS-CNN-S [5]	5.4M	24K	2.3	1.3	94.4
DS-CNN-M [5]	19.8M	140K	9.2	0.5	**94.9**
Res8-Narrow [4]	143.2M	**20K**	57.6	4.4	90.1
Res15-Narrow [4]	348.7M	43K	112.4	9.7	94.0
OWSNet (ours)	3.0M	66K	**1.5**	**0.8**	93.7

2) OWSNet Model Architecture: We adopt the ResNet architecture and make the following modifications: (i) we use $m \times 1$ kernels instead of the 3×3 kernels. (ii) Use $m = 3$ for the first network layer and $m = 9$ for the remaining layers. (iii) We remove the bias from all existing fully connected and convolution layers. (iv) All the existing Batch Normalization (BN) layers have trainable parameters for scaling and shifting. (v) We directly use the identity shortcuts whenever the dimensions of input and output are of matching dimensions. If not matching, similar to [4], we use an extra *conv-BN-ReLU* layer for dimensions matching, but in a setting with temporal convolutions instead of their conventional 3×3 kernel. (vi) For other network layers, we follow the original ResNet implementation but exclude dilated convolutions and adopt strided convolutions.

B. Experimentation: Training OWSNet

We train *OWSNet* using the Google Speech Commands dataset, which contains 64,727 one-second-long recorded and labeled audio files. Out of the existing 30 categories of recordings, we train the network to distinguish 10 classes that are: *yes, no, up, down, on, off, stop, go, silence, and unknown*. We split the dataset and use 80% for training, 10% for validation, and 10% for testing. For data augmentation and preprocessing, we apply random shift and inject noise into the training data. We generate background noise by cropping random background noises from the selected dataset, then sample a random co-efficient from the uniform distribution U(0, 0.1) and multiply it with the background noise. The audio file from the dataset is decoded to a float tensor and shifted by s seconds (sampled from U (-0.1, 0.1)) with zero padding and blended with the background noise.

We next follow the procedure from [4] and decomposed the audio files into a sequence of frames with the window of 30 ms and the stride of 10 ms for feature extraction. For each frame, we use 40 MFCC features and stack the features over the time axis. We now perform the training in TensorFLow, with a 0.001 weight decay and a 0.5 probability dropout to avoid overfitting. We use SGD on a mini-batch of 100 samples, and *OWSNet* was trained from scratch for 27k iterations.

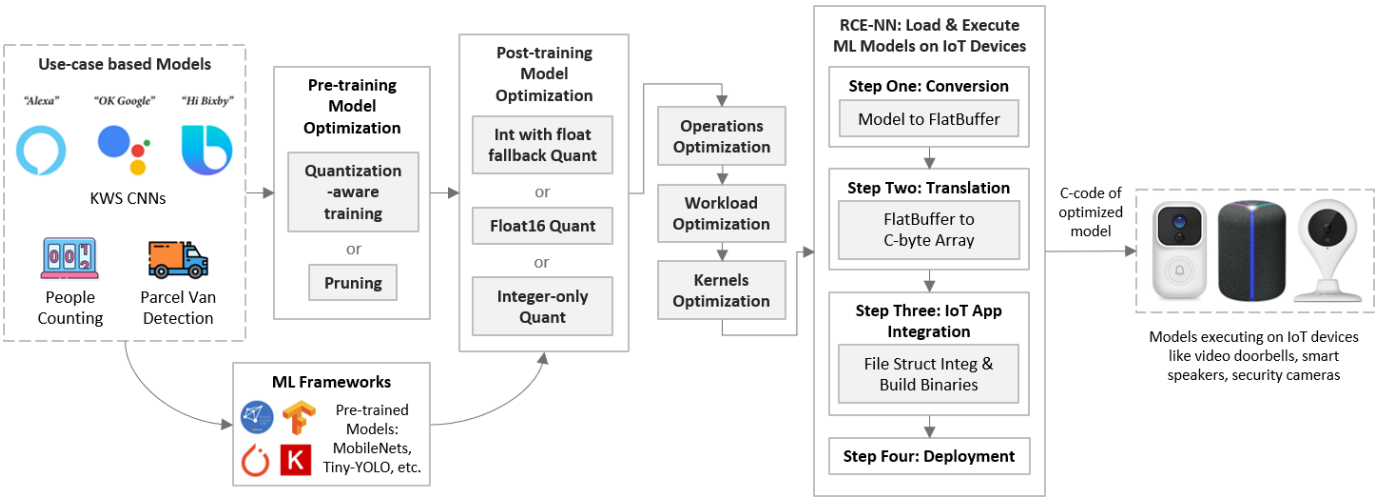

Fig. 2. Multi-component model optimization sequence: sequence to follow for optimizing any ML model to enable its execution on tiny MCUs and small CPUs based IoT devices. We apply this optimization sequence on the designed KWS network to enable its execution on consumer IoT devices.

C. OWSNet Evaluation and Results Comparison

We trained *OWSNet* 6 times, then average and report the model performance in this section (we use accuracy metric for evaluation). To ensure that *OWSNet* is friendly enough to execute on real-world resource-constrained devices, we quantify and report the *OWSNet*'s computation performance in terms of FLOPs and the number of all parameters (instead of reporting only trainable parameters). We also measure and report the inference time consumed by the trained *OWSNet* during execution on small CPUs and edge GPUs based devices, where device one (D1) is a Raspberry Pi 4 CPU, and device 2 (D2) is an NVIDIA Jetson Nano. For statistical validation, we measure the inference time for 50 runs and report the average.

In Table I, we report the FLOPs, total parameters, inference time on selected devices, and accuracy of *OWSNet* next to the results from the papers of top KWS models. The CNN-stride4 and CNN-fpool3 represents the cnn-one-fstride4 and cnn-trad-fpool3 models from [6] respectively. The results of the next DS-CNN-S and DS-CNN-M that contains depthwise convolutions are imported from [5]. The next, Res8-Narrow (contains 8 layers) and Res15-Narrow (contains 15 layers) imported from [4] employs a residual architecture to spot keywords. In these networks, the suffix Narrow indicates that it contains a reduced number of channels. In the remainder, based on Table I, we analyze and compare the results.

DS-CNN-M showed the top accuracy of 94.9% (1.2% higher than *OWSNet*), CNN-stride4 has the lowest FLOPs (1.5M FLOPs lower than *OWSNet*), and Res8-Narrow contains the least number of parameters (46K lesser than *OWSNet*). Although other networks show better results, the *OWSNet* is the fastest as it produces 1.26-74.9 times faster inference on D1 and 1.37-12.12 times faster inference on D2. We were able to achieve such inference speedups due to the incorporation of temporal convolutions into the network design.

III. Multi-component Model Optimizer Design

We propose an end-to-end multi-component optimization sequence to enable the execution of high memory and computation demanding ML models on the low resource, cost, power IoT devices[1]. Fig. 2, shows how to optimize ML models in multiple aspects to obtain resource-friendly models that can readily be deployed on devices like smart speakers, security cams, video doorbells, etc. The presented generic optimization sequence is applicable for *OWSNet* and also for ML models trained to solve problems in use-cases such as anomaly detection, predictive maintenance, machine vision.

A. Model Optimization Components

As shown in Fig. 2, we initiate the optimization sequence by providing methods to perform quantization-aware training or pruning. Next, the model should undergo any one of the post-training quantization schemes to obtain a quantized version of the actual model. Then we apply the operations optimization techniques on the models to keep the operational cost low without impacting the model architecture. Next, we reduce the computational workloads of CNNs, by realizing our workload optimization components. Finally, to maximize the performance while minimizing the memory footprint of CNNs, we utilize lighter kernels. After optimization, the users need to follow the steps from the RCE-NN pipeline [7] to convert and execute the optimized models on resource-constrained devices.

B. Optimizing standard CNNs

Before optimizing the *OWSNet*, we implement each of the optimizer components on standard CNNs and present the experimental results in our repository. For the experiments, we use the standard MNIST Fashion and MNIST Digits datasets to train a basic CNN whose architecture consists of a reshape layer, Convolution 2D layer, MaxPooling2D layer, Flatten layer, and finally a Dense layer. In the repository, we first import both the datasets via the *tf.keras.dataset.name* function with its default train and test sets. We then apply all suitable optimizers before, during, and after the training of CNNs and show the memory conservation and inference speedups achieved after realizing each optimizer component. In the following, we brief each optimization component.

[1]The implementation of the optimizer components are freely available at https://github.com/bharathsudharsan/CNN_on_MCU

Pruning. We implemented the magnitude-based weight pruning, where the model's weights are gradually zeroed out during the training process in order to achieve model sparsity. Thus obtained sparse models are easier to compress, and the zeroes can be skipped during inference, resulting in latency improvements. With a minimal accuracy loss, we experienced up to 5x compression as a result of training CNNs with pruning.

Quantization-aware training (QAT). When quantizing a CNN, the parameters and computations go from a higher to lower precision, resulting in improved execution efficiency at a cost of information loss. This loss is because the model weights can only take a small set of values, thus losing the minute differences between them. To reduce the loss and maintain the model accuracy, we implement the QAT scheme from [8]. Here, the quantization error is introduced as noise during the model training and as part of the overall loss, which the optimization algorithm in use tries to minimize.

Post-training Quantization. Here, we quantize the models by reducing the precision of their weights to save memory and simplify calculations often without much impact on accuracy. Recently in [7], we quantized WiFi and BLE RSS (Received Signal Strength) predicting CNNs and executed them on multiple MCU-based boards with almost no loss on their Mean Absolute Error (performance).

Operations Optimization. When designing a model aimed to execute on IoT devices, only a limited subset of operations can be used to keep the operational cost low. We notice that more than 90% arithmetic operations are used by convolutional (CONV) layers, so we perform depthwise separation of the 2-D CONVs to reduce parameters and operations count, thus enabling the fitting of larger networks on tiny IoT devices.

Workload and Kernels Optimization. The complexity and size of the model impact the workload. Models with dense architectures lead to increased processor workload and result in a higher duty cycle, resulting in IoT devices spending more time working (elevates power consumption and heat output) and less time idle. To reduce workloads without much engineering/implementation efforts, we recommend users to follow the list of methods from [7]. When further reduction is required, we recommend offloading the bulk of CNN workloads (convolutional layers) to nearby accelerators like co-MCUs/processors. Then, the exported plain C code of the ML model to be executed needs to be stored in a shared memory location (EEPROM) that can be accessed via common load/store ports. Also, the parallel offloading approach needs to be practiced during the programming phase since it leads to internal data reuse and improvement of inference performance.

Since the quantized model weights are stored and re-used during inference, we found that reordering the matrix weights can reduce the pointer accesses. We recommend users inherit any weight interleaving method to implement the weights reordering tasks. Then, we recommend replacing the default ReLU with its optimized version from [9].

RCE-NN Pipeline. Here, in four steps, we outline how to deploy and execute any ML model on the MCUs and small CPUs of IoT devices. As shown in Fig. 2, the Model conversion is Step 1, where the given model is converted into a `FlatBuffer` file containing its direct data structures such as the information arrays of graphs, subgraphs, lists of tensors, operators, etc. Next, in Step 2 which is Model translation, since MCUs lack native file-system support, we cannot load and run models directly on such devices, so here we convert the model's FlatBuffer file into a *c-byte array* (the model in a char array) using the *xxd* UNIX command. Step 3 is Model integration, where the *c-byte array* of the *OWSNet* model is fused with the main IoT program/application, for which the devices' executable binaries are built. Finally, in the Model deployment (Step 4), the generated binaries are flashed onto the program memory of the IoT devices.

C. Optimizing the OWSNet

Here, similar to how we optimized a basic CNN in the previous section, we take the trained *OWSNet* model from Section II and apply the optimization sequence from Fig. 2. But since we already had trained the network using the Google Speech Commands dataset, we skip the Pre-training optimization and directly quantize the model. The original *OWSNet* model was 981 kB in size, after Post-training model optimization; the *Int with float fallback quantized* version is 96.4 kB, the *Float 16 quantized* version is 169.4 kB in size, the *Integer-only quantized* version is 97.2 kB in size. As shown in the RCE-NN pipeline in Fig. 2, any of the quantized `.tflite` *OWSNet* model need to be converted into a *c-byte array* with the `.cc` format. After the conversion, it is common for the `.cc` model to be 5-7 times larger in size than its `.tflite` version. This model size expansion cannot be addressed since most MCU-based tiny IoT devices do not have native filesystem support. Hence as mentioned, the only way is to convert, include the quantized model as a c-array, then compile it along with the IoT application and flash it together on the device. In the case of better-resourced devices such as Alexa smart speaker, which roughly has Quad-core, 64-bit ARM Cortex-A35 8GB Flash, 2GB RAM, or a video doorbell which roughly has ARM Cortex-A9 CPU, 2GB RAM, 2GB Flash, the TensorFlow Light framework can be highly altered to remove the parts that are not needed to run the user's ML model, then installed on the device and perform inference directly using the quantized tflite use-case model.

IV. Related Work

Since we designed *OWSNet* KWS system, and also a multi-component ML model optimization sequence, the review consists of the two following subsections:

A. Keyword Spotting Networks

After the release of TensorFlow Lite and Micro frameworks, many studies have adopted CNNs for KWS use-cases. In [6], low parameters network was designed for KWS. In [5], CNN architecture exploration and evaluation for running KWS on resource-constrained devices was performed. Article [4], explored the popular ResNet architecture and varied the depth

and width of the model to improve its compactness while also outperforming other KWS networks. In all such studies, their network contains a 2D convolutional layer that is more expensive to run in a limited RAM budget. To provide a more resource-friendly solution, study [10] have used 1D kernels in the network designed for their music feature extraction use-case. Similarly, study [11] presented a rare sound event detection network, which is a combination of 1D convolutional neural network and recurrent neural network. We take inspiration from these studies and apply 1D convolution to design a network for the offensive words spotting use-case. But unlike them, we apply the convolutions on the temporal axis of the time–frequency signal representation rather than focusing on the frequency axis of the input audio signals.

B. ML Model Optimization Techniques

In the category of algorithms for model optimization, there is a set of articles proposing compression techniques to reduce the size of the model's weights using quantization and pruning. Condensa [12], a system for users to compose simple operators to build complex model compression strategies. Two new compression methods jointly leverage weight quantization and the distillation of larger networks was proposed in [13]. Authors in [14] have implemented a tree-based algorithm for efficient prediction in milliseconds even on slow MCUs. Similarly, ProtoNN [15], k-NN inspired algorithm with several orders lower storage and prediction complexity addresses the problem of real-time and accurate prediction on resource-scarce devices. In both [12,13] and other similar articles proposing compressing [16,17] and optimization [9,18] methods, the models are trained in high resource setups, then a multi-stage MCU-aware optimization (tailored) is performed before deployment. In [19]–[22], SRAM optimized porting of Decision Trees (DT) and Random Forest (RF) is performed, and the ported models are efficiently executed on MCU boards. The popular sklearn-porter, m2cgen, emlearn libraries can be used to generate optimized C code, using which use-case based ML models like the 'Adaptive Strategy' SVR [23], 'Edge2Guard' [24], 'Covid-away' [25] can be ported and deployed on a range of IoT devices.

V. Conclusion

In this paper, we presented *OWSNet*, a resource-friendly temporal convolutions-based network design that can perform real-time spotting of multiple keywords on consumer IoT devices. Although a few models have lesser FLOPs, lesser total parameters, and slightly higher accuracy, our *OWSNet* is faster than them and other state-of-the-art models as it produced $\approx$ 1-74 times faster inference on Raspberry Pi 4 and $\approx$ 1-12 times faster inference on NVIDIA Jetson Nano. We also presented a generic multi-component ML model optimization sequence that can optimize any high memory and computation demanding ML model and enable its execution on the low resource, cost, and power IoT devices. We applied a few optimization components on *OWSNet*, which reduced the model size by 10 times with almost no loss of accuracy.

In the future, we plan to build a hate speech audio dataset that contains voice recordings of multiple offensive words, then train *OWSNet* using this dataset and benchmark the offensive words spotting performance. We then plan to deploy and execute the resultant model on the microphone-array based Alexa smart speaker prototypes [3,26] and perform real-world evaluations.

Acknowledgement

This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/16/RC/3918 (Confirm) and also by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289_P2 (Insight), with both grants co-funded by the European Regional Development Fund.

References

[1] B. Sudharsan, J. G. Breslin, and M. I. Ali, "Edge2train: A framework to train machine learning models (svms) on resource-constrained iot edge devices," in *10th International Conference on the Internet of Things*, ser. IoT '20, 2020.

[2] B. O'Brien, "White police officer cleared of charges in wisconsin shooting of black man," *Reuters*, 01 2021.

[3] B. Sudharsan, S. P. Kumar, and R. Dhakshinamurthy, "Ai vision: Smart speaker design and implementation with object detection custom skill and advanced voice interaction capability," in *ICoAC*, 2019.

[4] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," in *IEEE ICASSP*, 2018.

[5] Y. Zhang, V. Chandra *et al.*, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint*, 2017.

[6] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *INTERSPEECH*, 2015.

[7] B. Sudharsan, J. G. Breslin, and M. I. Ali, "RCE-NN: a five-stage pipeline to execute neural networks (cnns) on resource-constrained iot edge devices," in *International Conference on Internet of Things*, 2020.

[8] B. Jacob, D. Kalenichenko *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference."

[9] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," arXiv Preprint, 2018.

[10] K. Choi, G. Fazekas, M. Sandler, and K. Cho, "Convolutional recurrent neural networks for music classification," in *IEEE ICASSP*, 2017.

[11] H. Lim, J. Park, and Y. Han, "Rare sound event detection using 1d convolutional recurrent neural networks," in *DCASE Workshop*, 2017.

[12] V. Joseph, S. Muralidharan, A. Garg, M. Garland, and G. Gopalakrishnan, "A programmable approach to model compression."

[13] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *arXiv Preprint*, 2018.

[14] A. Kumar, S. Goyal, and M. Varma, "Resource-efficient machine learning in 2 KB RAM for the internet of things," in *ICML*, 2017.

[15] C. Gupta, P. Jain *et al.*, "Protonn: Compressed and accurate knn for resource-scarce devices," in *ICML*, 2017.

[16] W. J, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint*, 2015.

[17] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," *arXiv preprint*, 2016.

[18] S. Bhattacharya, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *SenSys*, 2016.

[19] B. Sudharsan, P. Patel, J. G. Breslin, and M. I. Ali, "Ultra-fast machine learning classifier execution on iot devices without sram consumption," in *IEEE PerCom Workshops*, 2021.

[20] B. Sudharsan, P. Patel, J. G. Breslin, and M. I. Ali, "Sram optimized porting and execution of machine learning classifiers on mcu-based iot devices: demo abstract," in *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems (ICCPS)*, 2021.

[21] B. Sudharsan, M. I. Ali *et al.*, "Demo abstract: Porting and execution of anomalies detection models on embedded systems in iot," *IoTDI*, 2021.

[22] B. Sudharsan and P. Patel, "Machine learning meets internet of things: From theory to practice," *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2021.

[23] B. Sudharsan, J. G. Breslin, and M. I. Ali, "Adaptive strategy to improve the quality of communication for iot edge devices," in *World Forum on Internet of Things (WF-IoT)*, 2020.

[24] B. Sudharsan, D. Sundaram, P. Patel, J. Breslin, and M. Intizar Ali, "Edge2guard: Botnet attacks detecting offline models for resource-constrained iot devices," *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2021.

[25] B. Sudharsan, D. Sundaram, J. G. Breslin, and M. I. Ali, "Avoid touching your face: A hand-to-face 3d motion dataset (covid-away) and trained models for smartwatches," in *10th International Conference on the Internet of Things Companion*, ser. IoT '20 Companion, 2020.

[26] B. Sudharsan, P. Corcoran, and M. I. Ali, "Smart speaker design and implementation with biometric authentication and advanced voice interaction capability," in *Proceedings for the 27th AIAI AICS*, 2019.