人工知能学会第二種研究会資料
AIチャレンジ研究会
SIG-Challenge-058-06

# Speech-Classification in Edge-Computing for IoT Applications

Haris Gulzar*, Muhammad Shakeel*, Kenji Nishida*, Katsutoshi Itoyama*, Kazuhiro Nakadai* †
*Dept. of Systems and Control Engineering, Tokyo Institute of Technology, Tokyo, Japan
Email: {gulzar, shakeel, nishida, itoyama}@ra.sc.e.titech.ac.jp,
†Honda Research Institute Japan Co., Ltd., Saitama, Japan
Email: nakadai@jp.honda-ri.com

***Abstract*—Speech based interface for interacting with smart devices has recently gained traction due to significant improvement in Machine Learning (ML) based algorithms for speech recognition and classification. Edge computing has come into play to make cloud-computing scalable because number of low power AI-enabled Internet of Things (IoT) devices is increasing rapidly. Convolutional Neural Network (CNN) has proved its performance in image recognition and speech classification alike. In this paper, we have proposed an edge computing solution using System-on-Chip based device from perspective of speech-enabled IoT applications. Speech commands classification task is performed to demonstrate the acceleration of CNN network on SoC edge computing device. As IoT and edge devices have limited computational resources, ideally a smaller model with similar performance as state-of-the-art models is required to be deployed. We have taken a data-centric approach to elevate the performance of CNN for audio classification task with a very light CNN model. The comparison of proposed approach has shown that our CNN model achieved similar performance as large models with 6X smaller number of parameters and 14X smaller number of Floating-Point Operations (FLOPs). We have also implemented the acceleration for our CNN model on FPGA part of SoC processor to reduce latency in real time implementation. Our implementation demonstrated more than 6X acceleration factor as compared to base implementation which is also higher than other proposed approaches for CNN acceleration.**

***Index Terms*—Edge Computing, Machine Learning, Internet of Things, Hardware Accelerator, Speech Classification**

## I. Introduction

Efficiency of various human machine interfaces like face expression, gestures and speech recognition has been improving with advancement and innovation in Machine Learning (ML) algorithms [1]. Many State-of-the-Art (SOTA) ML models have been proposed to improve the performance of speech recognition as it is becoming de-facto interface in Human Robot Interaction (HRI), which has already found its application in assistive robots for various indoor scenarios [2] and coworking robots in industrial environments [3]. A Convolutional Neural network (CNN) based ML approach has also been proposed to control the drones using different voice commands [4]. Another ML based approach is introduced to maneuver the airplane in a simulated setup which gives hint in further expansion of speech-based applications in diverse environments [5]. Currently high performing ML models are significantly large in terms of memory and computations and they are usually deployed on large cloud platforms with abundant resources available for high computations. Recently, the focus has been shifted to deploy ML models on low power IoT devices which are increasing in demand and in numbers in several daily life applications [6]. These low power IoT devices due to their small footprints have increased in number and with speech-enabled functionality they are particularly useful for handicapped people who have some mobility restrictions [7].

This work is supported by JST, CREST Grant No. JPMJCR19K1, Japan.

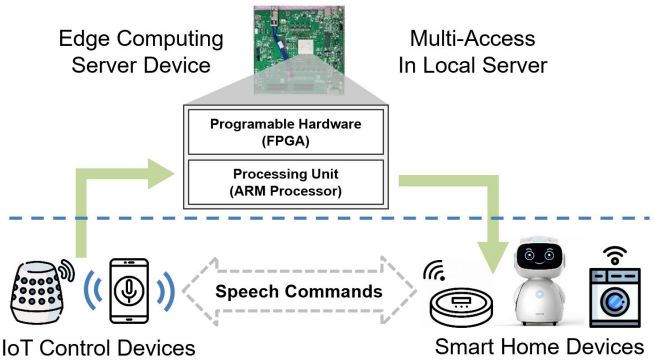


Fig. 1. Speech Interface for IoT devices enabled by Edge Computing.

Devices with smaller footprints would have small space for computational resources, thus the computational burden has to be unloaded to nearby computational resource and this is where edge computing comes into play. In addition to that transferring personal speech data to remote cloud is vulnerable to cyber-attacks and pose a privacy concern. Continues integration with cloud computing would also require a seamless internet connection and any disturbance in network connection may lead to unwanted scenario in terms of high latency or network interruption. Edge computing solves these problems by bringing the computational resources from cloud close the application node. A device enabled with speech recognition having additional computational resource in the local server in form of network edge is an ideal scenario for speech-enabled IoT applications. Fig. 1 shows that how small devices can utilize the local edge server to utilize its computation power to run the speech enabled applications. A multi-access edge computing approach also allows to elevate the fault tolerance of the overall system by using multiple

devices, where one device can take over the role of other device in case of device failure.

Edge devices are still inferior to cloud devices in terms of available resources due to their smaller footprint; they are basically scaled sown version of cloud nodes. ML algorithms have to be tailored according to the resource constraints of the targeted device in such a manner that there is not a significant performance degradation while reducing their computational cost and memory. Different types of devices have been considered for Edge applications including CPU, GPU and FPGA [8]. FPGA is one of the best candidates in this case due to its flexibility to be a reprogrammable hardware which allows faster delivery of the final product to the market. System on Chip (SoC) based processor has further enhanced the FPGA ability and performance by fabricating the CPU and FPGA on the same chip. As illustrated in Fig.1 SoC has two parts; in our case FPGA acts as an accelerator and CPU hosts the Operating System to provide a platform for real time application. Our work has made following contributions while keeping in mind the requirements of speech application in low power IoT devices:

- It proposed the lightest CNN model in terms of parameters and computational cost while maintaining the SOTA performance.
- It shows how taking a data-centric approach can help in achieving high performance as large models with a significantly smaller model.
- It demonstrates the deployment of FPGA accelerator for CNN for speech classification on Edge devices.

## II. Related Work

A plethora of Recurrent Neural Network (RNN) and CNN based ML approaches have been proposed for speech commands classification, which are summarized in Table 1. An attention based RNN technique is also proposed for similar task [9-10], showing significant improvement in classification accuracy performance. But as seen from the Table 1, RNN models are large in terms of number of parameters which makes it harder to deploy them on small memory devices. CNN models on the other hands deliver good performance with comparatively less number of parameters. EdgeSpeechNet which is a CNN based approach [11] achieves similar performance as RNN based approaches with smaller number of parameters, but Floating-Point Operations (FLOPs) are very large. RES15 has also delivered a reasonable performance but number of parameters and FLOPs increase even further [14].

Depth wise Separable CNN (DS-CNN) approach [15] maintains good performance by exploiting depth wise convolution operations and achieves 95.4% accuracy for speech commands classification. CNN network can learn better from training data if the size of network is increased significantly but it leads to very high number of operations and parameters [16] making it impossible to be deployed on small devices. Type of network affects performance but efficiently utilizing useful information in the training data also leads to better performance with smaller network size.

In this paper, we have also shed some light on data-centric approach by demonstrating the high accuracy performance

TABLE I
Performance Metrics of Different Models on Google Speech-Commands Dataset V1

| Model Type | Model Name | Accuracy (%) | Params. (K) | FLOPS (M) |
|---|---|---|---|---|
| RNN | MHAtt-RNN [9] | 97.2 | 743 | - |
| | EdgeRNN [10] | 96.62 | 830 | 26.96 |
| | EdgeRNN-G [10] | 96.82 | 830 | 2.96 |
| CNN | EdgeSpeechNetA[11] | 96.8 | 107 | 343 |
| | EdgeSpeechNetD[11] | 95.8 | 80.3 | 24.5 |
| | RES15 [14] | 95.8 | 238 | 894 |
| | DS-CNN [15] | 95.4 | 161 | 56.9 |
| | CNN [16] | 96.19 | 1,488 | - |
| | **CNNv1 (Ours)** | 96.45 | 224 | **23.7** |
| | **CNNv1-L (Ours)** | 95.4 | **67.7** | **6.5** |
| | **CNNv2 (Ours)** | 95.11 | **93.2** | **3.7** |

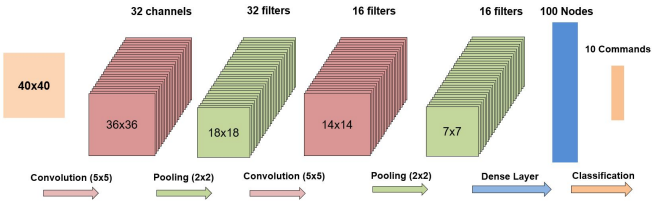


Fig. 2. CNN architecture of CNNv2.

with a significantly smaller model having small number of parameters and FLOPs. In second part of the paper, we have demonstrated the deployment of accelerator for CNN which achieved better acceleration factor as compared to proposed approaches so far. The following sections explain about the dataset used for this task, CNN network architecture, introduction of hardware and network deployment.

## III. Dataset & Input Features

The dataset being used in this work is named as Speech Commands, developed by Google [23]. There are ten most common speech commands in first version of the dataset e.g., “Yes”, “No”, “Up”, “Down”, “Left”, “Right”, “On”, “Off”, “Stop” and “Go”. For developing speech-based applications, this data is a very reasonable starting point. The complete dataset is divided into two parts as 80% training data and 20% as validation data. Increasing the amount of training examples using data augmentation improves the overall performance of ML model [13]. In our work we have used random time shifting of audio signal as our data augmentation technique.

Speech commands audio files are sampled at 16kHz frequency. First of all we have taken Frequency Cepstral Coefficients (MFCC) of the signal which has 13 frequencies in one time bin equivalent to 25ms. MFFCs provide rich information of speech signal in lower dimension by extracting most of the phoneme information as speech signal is basically the continuous sequence of phonemes. In speech signal amplitude of consecutive phonemes vary significantly so we take the difference of amplitude of power spectrum for consecutive phonemes.

These features are known as delta features, which have same effect as derivative of the signal. After taking 13 delta features of the signal, we take another 13 double delta features

by taking the difference once again [24]. These features inherently make the features robust to white noise because it is suppressed by taking difference of signal. Lastly, the final feature of the signal is spectrogram power for each frame, making the final shape of features matrix to be 40x40. The CNN architecture, its accelerator design and hardware introduction are explained in the following sections.

TABLE II
OUR CNN PERFORMANCE WITH DIFFERENT INPUT FEATURES

| Features Type | Features Dimension | Accuracy (%) |
|---|---|---|
| MFCC | 13x40 | 94.11 |
| MFCC, Delta | (13+13)x40 | 94.89 |
| MFCC, Delta, Double-Delta, Delta-Energy | (13+13+13+1)x40 | 95.11 |

## IV. OUR CNN ARCHITECTURE AND ITS COMPARISON WITH STATE OF THE ART

CNN along with its many variations has proved very high performance for image recognition task due to its ability to recognize local patterns. CNN can also perform equally well for sound classification task, given the speech data is first converted into a two-dimensional matrix like an image. In image recognition task, optimizing the network architecture for the given task is mostly the only way to improve performance, but in speech classification, we have another task to focus on, that is to efficiently extract the information from audio data. After efficient data extraction our CNN model has achieved good performance even though it is smaller in size.

TABLE III
CNN HYPERPARAMETERS COUNT

| Layer | Parameters | CNNv1 | CNNv2 | CNNv1-L |
|---|---|---|---|---|
| Conv.1 | Channels | 64 | 32 | 32 |
| | Kernel | 5x5 | 5x5 | 5x5 |
| Conv.2 | Channels | 128 | 16 | 64 |
| | Kernel | 3x3 | 5x5 | 3x3 |
| Conv.3 | Channels | 64 | - | 32 |
| | Kernel | 3x3 | - | 3x3 |
| Dense | Nodes | 128 | 100 | 100 |
| Total No. of Parameters | | 224,576 | 93,158 | 67,702 |

The model being used in this work is CNN with convolutional layers followed by a fully connected layer and classification layer. The input shape of spectrogram is a symmetric 2-dimensional matrix; hence first layer of the network is 2-dimensional convolution layer. First convolutional layer for both CNNv1 and CNNv2 has same size of symmetric kernel i.e. 5x5. The stride length of 1 for convolutional layer and 2 for pooling layers is used throughout the network. Hyperparameters like number of output channels, number of convolutional layers, kernel-size for following layers and number of nodes in fully-connected layer are varied in three versions of CNN models. Rectified Linear Unit (ReLU) activation function is used after each pooling layer. Activation function is relatively simpler to model in High Level Synthesis (HLS) as it only requires to change negative values to zero. Three variations of CNN model have been used in our experiment with different hyperparameters. The detail of hyperparameters for each

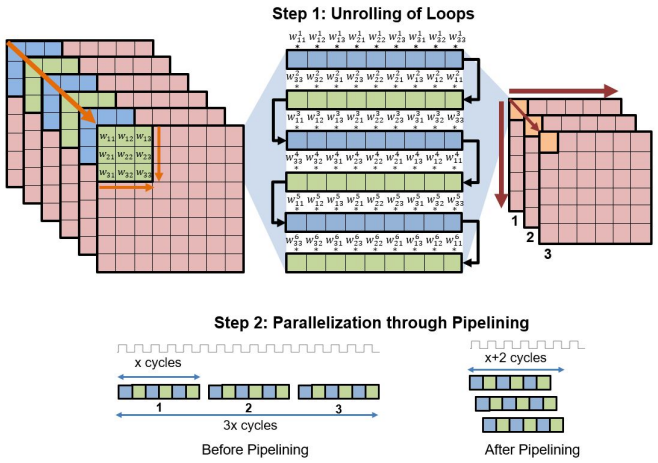


Fig. 3. Parallelization by loop unrolling followed by pipelining.

version of the model are listed in Table 3. Largest model is decided according to the constraints of the edge device which we have use in this experiment. For network training Adam optimizer is used with cross entropy loss. Training is performed for 300 epochs where validation loss and accuracy are stabilized.

CNNv1 which is the largest model in our case achieves SOTA performance with 6X smaller number of parameters than another CNN approach for a similar task [16]. In a similar way, our largest model has achieved better performance than RES15 []14] with enormously low number of operations e.g. 40X. We performed another experiment by reducing the size of CNNv1 up to 3X while maintaining performance up to 95%, and lighter version of the model is named as CNNv1-L. This model achieves similar performance as DS-CNN [15] with 2X smaller number of parameters and 9X smaller number of FLOPs. This model also has 9X smaller FLOPs than EdgeSpeechNetD [11] while achieving almost similar accuracy. We also tried to investigate the effect on accuracy by changing the number of convolutional layers and used a network with two convolutional layers, named as CNNv2. This model has large number of parameters than CNNv1-L still has lower accuracy. This clearly shows the supremacy of convolutional layers, which delivers better performance by doing more operations with less parameters. This also implies that deeper CNNs has better performance than shallow CNN networks.

Even our largest model CNNv1 has 7X smaller number of parameters than CNN [14] but still achieves higher performance. Similar, CNNv1 has 4X smaller size than EdgeSpeechNetD [I8] but still achives 0.65% better accuracy.

## V. HARDWARE INTRODUCTION

The device used in this research is a custom-built Multiaccess Edge Computing (MEC) device named as M-KUBOS [18] which acts as edge server in our case. It has Xilinx Zynq Ultrascale+ XCZU19 System on Chip (SoC) processor on board. SoC on M-KUBOS has FPGA as programmable hardware and ARM processor as processing unit fabricated

on the same chip. Moreover, SoC has on-chip memory for storing weights of the neural network logic which cuts short the reading and writing time of data into a separate memory while computations take place. SoC delivers high end performance by leveraging the synergistic usage of processing unit and programable hardware which communicate to each other through Advanced eXtensible Interface (AXI) as illustrated in Fig. 4. In addition to the on-chip memory, M-KUBOS has additional Ultra-RAM on board for storing large data which can be accessed by ARM processor and FPGA through Direct Memory Access (DMA). M-KUBOS has Linux operating system with Python production for Zynq devices (PYNQ) functionality [22]. PYNQ framework in Linux enables easy development of applications based on Zynq devices by flexibly using developed hardware components in Python program. M-KUBOS acts as shh server and can be seamlessly accessed remotely.

High Level Synthesis (HLS) tool by Vivado is used to design the CNN Logic in C language. Neural network logic can be synthesized and exported as Intellectual Property (IP) core to be flexibly used in hardware integration on Vivado Design Suite. Vivado Design Suite provides immense amount of customization for SoC based boards like M-KUBOS, where available hardware resources can be flexibly put together using Function Block Diagram (FBD) based programming [17]. Finally integrated hardware design is synthesized and bitstream file is burned into FPGA through Python-based application program in Linux operating system. Weights from trained ML model are also transferred to on-chip memory and processing system uses FPGA as an accelerator for inference as shown in Fig. 3.

## VI. Accelerator Implementation

A convolutional layer in neural network has six loops to compute activations for next layer from input channels. As described in Algorithm 1, two inner most loops compute output for one kernel operation. The loop on top of kernel operation iterates over each channel of input layer. Looping over these kernel operations for every input channel followed by a summation returns a value corresponding to one output channel as shown in Fig. 3. These kernel operations in convolutional network provides enormous amount of opportunity for parallelization. As each channel in output layer has different weights for all input channels, computations corresponding to output values for each output channel can be performed concurrently.

Convolutional operations in CNN network can be parallelized by first unrolling the loops and then performing pipelining. For example, in Fig. 3, three inner most loops are unrolled into a long vector which is responsible for outputting one value in an output channel. If the output channels are three, conventional approach would compute output activation corresponding to each channel in series manner. The simplest operation in CNN takes 3 clock cycles i.e., reading, multiplying or adding and writing. If computing a single value in one output channel takes x number of clock cycles, then three output values corresponding to each output channel would take 3x cycles. However, these independent operations can be scheduled to be executed in parallel with just one cycle delay for scheduling them as illustrated in Fig. 3.

**Algorithm 1:** Convolutional Function

```
   Input: input, weight, bias
   Output: activation output
1  Convolution(input, weight, bias, output)
2  for w ∈ {1, ..., output_channel_width} do
3      for h ∈ {1, ..., output_channel_height} do
4          #pragma HLS PIPELINE
5          for n ∈ {1, ..., output_channels} do
6              #pragma HLS UNROLL
7              for i ∈ {1, ..., input_channels} do
8                  for j ∈ {1, ..., kernel_width} do
9                      for k ∈ {1, ..., kernel_height} do
10                         output_whn+ = w_ijk * input_ijk
11                         +bias_ijk
12             Above three loops unrolled into a vector.
13         Above loop is pipelined for parallel processing.
14 end Convolution
```

Vivado HLS tool allows to flexibly select the loops to be unrolled and parallelized. Algorithm 1 shows that we have unrolled three inner most loops of convolution function. The loop on top of three unrolled loops is pipelined and will execute its operation in parallel. Similarly, more number of loops can be unrolled and pipelining operation can be used for even higher loops. As we can on unrolling the loops, the size of one operations will increase i.e. multiplication and as we do pipelining size of parallel operations will increase i.e. number of operations. Hence total resource utilization will increase after optimization. Theoretically all loops in convolution layer can be parallelized, given that there is no limitation on available resources. However, finding the optimal balance between amount of parallelization and meeting the resource constraints is an important task [17]. Given the availability of computational resources in M-KUBOS, three to four loops are unrolled followed by pipelining in different convolutional layers of CNN network.

## VII. Experimental Setup & Results

Firstly, optimized CNN network i.e. CNNv is trained on Google Speech-Commands training dataset in Pytorch framework, and weights are obtained to transfer to chip memory before execution on M-KUBOS. Then HLS logic is synthesized on Vivado HLS and a hardware bitstream file of neural network logic is transferred to M-KUBOS device for programming the FPGA hardware. PYNQ programming framework is used for making and executing application program on peocessing system part of SoC device. After programming FPGA hardware and transferring network weights to on-chip memory Python-based application program sequentially takes the inputs and computes them by utilizing the on-chip FPGA accelerator. End-to-end latency is measured by executing base

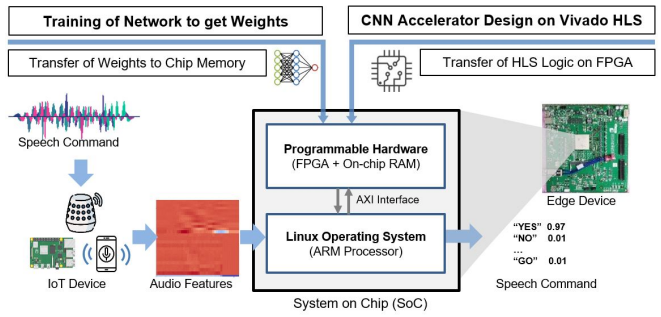


Fig. 4. Application framework for sound classification on edge device.

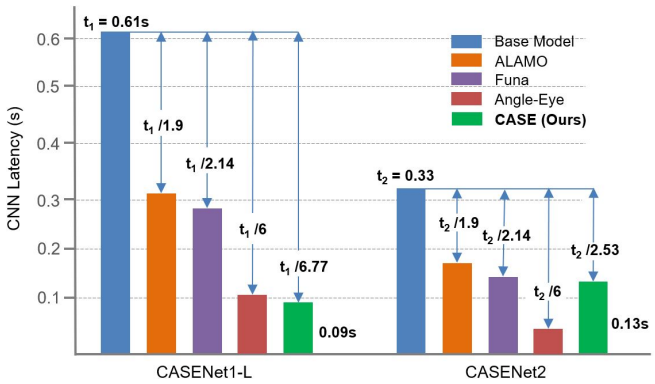


Fig. 5. Improvement in latnecy after parallelization in comparison to other methods.

implementation of our CNN logic followed by its optimized version for CNNv1-L and CNNv2. Improvement in latency after using optimized version is illustrated in Fig. 5. We used Python-based time library to measure the end-to-end execution latency for speech classification inference.

In addition to our approach, some other works are being proposed for CNN acceleration on image classification tasks, where CNN implementation remains the same but audio features input matrix is replaced by image matrix. Fune [19], a CNN accelerator tool achieves up to 2.14X speed-up of CNN by providing optimal parameters search for hardware synthesis. ALAMO [20] is another approach which also explores CNN acceleration for image recognition task by using an approach of loop unrolling followed by sequential computation, resulting in 1.9X speed-up factor. Another proposal of CNN acceleration, Angle-Eye [21] achieved 6X as their best speed-up factor by exploiting the potential of parallelization in CNN. Our approach in case of CNNv1-L has achieved an acceleration factor of 6.7X, which is higher than the previously mentioned approaches. An important observation in our experiment is that CNNv2 has lower latency without parallelization but still achieves lower acceleration factor than CNNv1. The reason is that 3 Convolutional Layers in CNNv1 provide more opportunity of parallelization than two convolutions in CNNv2. As convolution reduces size of input, so lesser number of convolutional layers in network means more parameters before fully connected layer which has relatively lower potential of parallelization.

## VIII. Conclusion & Future Work

This work is focused on implementing a CNN accelerator on SoC based edge device for classification of speech commands. The CNN proposed in this paper is a very light network which delivered SOTA accuracy performance while keeping the computational cost and memory significantly lower. We demonstrated that how keeping data centric approach helps to achieve high performance with small model by efficiently extracting useful information from training data. The comparison of our model shows that it delivers high performace same as large SOTA models. Second half of the paper focused on accelerator design for CNN network which is deployed on FPGA part of SoC processor. After using the accelerated design, the end-to-end latency of the speech coammand classification was reduced by 6X. The future work related to this on-going research is as follows:

- To extend the implementation to Automatic Speech Recognition (ASR) from speech commands classification.
- To make an integrated framework for sound sepration, source localization and speech recognition.
- To measure and improve the power consumption of FPGA based device and compare it with GPU and CPU.

## References

[1] R. Supreeth & Kamath Ajay & N. Srinidhi & Kumaraswamy Ramaswamy. (2021). Fully Responsive Image and Speech Detection Artificial Yankee (FRIDAY): Human Assistant. SN Computer Science. 2. 10.1007/s42979-021-00630-8.

[2] José Novoa, Rodrigo Mahu, Jorge Wuth, Juan Pablo Escudero, Josué Fredes, and Néstor Becerra Yoma. 2021. Automatic Speech Recognition for Indoor HRI Scenarios. J. Hum.-Robot Interact. 10, 2, Article 17 (May 2021), 30 pages.

[3] Mustafa Can Bingol, Omur Aydogmus, “Performing predefined tasks using the human–robot interaction on speech recognition for an industrial robot”, Engineering Applications of Artificial Intelligence, Volume 95, 2020,103903, ISSN 0952-1976.

[4] C. M. J. Galangque and S. A. Guirnaldo, ”Speech Recognition Engine using ConvNet for the development of a Voice Command Controller for Fixed Wing Unmanned Aerial Vehicle (UAV),” 2019 12th International Conference on Information & Communication Technology and System (ICTS), 2019, pp. 93-97.

[5] M. Vukovic, M. Stolar and M. Lech, ”Cognitive Load Estimation From Speech Commands to Simulated Aircraft,” in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 29, pp. 1011-1022, 2021.

[6] C. Zonios and V. Tenentes, ”Energy Efficient Speech Command Recognition for Private Smart Home IoT Applications,” 2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST), 2021, pp. 1-4.

[7] Mayer, J.. “IoT Architecture for Home Automation by Speech Control Aimed to Assist People with Mobility Restrictions.” (2017).

[8] G. Dinelli, G. Meoni, E. Rapuano and L. Fanucci, ”Advantages and Limitations of Fully on-Chip CNN FPGA-Based Hardware Accelerator,” 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1-5.

[9] Rybakov, Oleg & Kononenko, Natasha & Subrahmanya, Niranjan & Visontai, Mirko & Laurenzo, Stella. (2020). Streaming keyword spotting on mobile devices. arxiv.org/abs/2005.06720.

[10] S. Yang, Z. Gong, K. Ye, Y. Wei, Z. Huang and Z. Huang, ”EdgeRNN: A Compact Speech Recognition Network With Spatio-Temporal Features for Edge Computing,” in IEEE Access, vol. 8, pp. 81468-81478, 2020.

[11] Lin, Zhong & Chung, Audrey & Wong, Alexander. (2018). EdgeSpeechNets: Highly Efficient Deep Neural Networks for Speech Recognition on the Edge. arxiv.org/abs/1810.08559.

[12] J. Xu, S. Li, J. Jiang and Y. Dou, "A Simplified Speaker Recognition System Based on FPGA Platform," in IEEE Access, vol. 8, pp. 1507-1516, 2020.
[13] A. N. Çayır and T. S. Navruz, "Effect of Dataset Size on Deep Learning in Voice Recognition," 2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), 2021, pp. 1-5.
[14] Tang, Raphael & Lin, Jimmy. (2017). Deep Residual Learning for Small-Footprint Keyword Spotting. arxiv.org/abs/1710.10361.
[15] Zhang, Yundong, Naveen Suda, Liangzhen Lai and V. Chandra. "Hello Edge: Keyword Spotting on Microcontrollers." ArXiv abs/1711.07128 (2017).
[16] A. Soliman, S. Mohamed and I. A. Abdelrahman, "Isolated Word Speech Recognition Using Convolutional Neural Network," 2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE), 2021.
[17] L. Stornaiuolo, M. Santambrogio and D. Sciuto, "On How to Efficiently Implement Deep Learning Algorithms on PYNQ Platform," 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2018, pp. 587-590.
[18] "FPGA computing platform M-KUBOS", https://www.paltek.co.jp/design/original/m-kubos/index.html
[19] Q. Xiao and Y. Liang, "Fune: An FPGA Tuning Framework for CNN Acceleration," in IEEE Design & Test, vol. 37, no. 1, pp. 46-55, Feb. 2020.
[20] Yufei Ma, Naveen Suda, Yu Cao, Sarma Vrudhula, Jae-sun Seo, "ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler Integration", Volume 62, 2018, Pages 14-23.
[21] K. Guo et al., "Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 1, pp. 35-47, Jan. 2018.
[22] Python Productivity for Zynq (PYNQ), http://www.pynq.io/
[23] Warden, Pete. "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition." ArXiv abs/1804.03209 (2018).
[24] Delta and Delta-delta audio features, Introduction to Speech Processing, https://wiki.aalto.fi/display/ITSP/Deltas+and+Delta-deltas