Keyword Spotting with Deep Neural Network on Edge Devices

Conference Paper · July 2022

DOI: 10.1109/ICEIEC54567.2022.9835061

2 authors, including:

Md Naim Miah
Purdue University Fort Wayne
3 PUBLICATIONS **19** CITATIONS

Keyword Spotting with Deep Neural Network on Edge Devices

Md Naim Miah
Dept. of Electrical and Computer Engineering
Purdue University Fort Wayne
Fort Wayne, IN, 46805
miahm01@pfw.edu

Guoping Wang
Dept. of Electrical and Computer Engineering
Purdue University Fort Wayne
Fort Wayne, IN, 46805
wang@pfw.edu

***Abstract*—Interconnected devices are becoming attractive solutions to integrate physical parameters and make them more accessible for further analysis. The exploding number of sensors are demanding huge bandwidth and computational capabilities in the cloud. Additionally, continuous transmission of information to the remote server would hamper privacy. Deep Neural Network (DNN) is proving to be very effective for cognitive tasks and can easily be implemented on edge devices. In this paper, a DNN model was trained and implemented for Keyword Spotting (KWS) on two types of edge devices: a bare-metal embedded device (microcontroller) and a single board computer (Jetson Nano™) based robotic car. The unnecessary components and noise from audio samples were removed and speech features were extracted using Mel-Frequency Cepstral Coefficient (MFCC). A Depthwise Separable Convolutional Neural Network (DS-CNN) based model was proposed and trained with about 721 thousand trainable parameters. After implementing the DNN on a bare-metal platform, the converted model took only 11.52 Kbyte of RAM and 169.63 Kbyte of flash memory. The model was also utilized to develop a voice-controlled robotic vehicle, Jetbot. Thus, this research demonstrates an efficient implementation of DNN on edge devices in two major platforms.**

***Index Terms*—KWS, MFCC, DNN, Bare-metal, Jetbot**

I. Introduction

Internet of Things (IoT) or network interconnected devices are growing faster with the advancements in wireless networking technologies. The number is expected to cross about 125 billion by 2030 [1]. It connects a massive number of sensors and devices in cloud data centers. However, it is gradually becoming a challenging task for the clouds to handle this big data. It is required to deliver a huge computation power, which is unquestionably a serious challenge. Additionally, the increasing demand for data traffic is also touching the global maximum limits. Especially, applications that need continuous monitoring such as KWS from speech data. This data might include personal information and it poses privacy concerns to transfer this data to the cloud. Edge computing has proven to be an effective way out of this problem. Instead of sending the data to the cloud, it performs computation by itself and thus overcomes the issues with bandwidth cost, privacy, and scalability [2], [3]. However, edge devices often suffer from limited computation and storage capability. It also needs to provide high accuracy outputs in real-time.

In this paper, we implemented an efficient DNN on edge device in two different platforms. A brief description of some previous works is provided in the next section. Section III discusses about pre-processing the audio dataset. Section IV describes the architecture of the DNN model. Edge device hardware platforms are described in Section V. Finally, the experimental results are presented in Section VI that evaluates the performance of the trained DNN model on edge devices.

II. Related Works

KWS on edge devices has already proven to be very useful to interact with electronic devices, for example “Google Home” and “Amazon Echo.” Only after detecting a keyword, such as ”Okay Google” or ”Alexa,” these devices typically go online or record speech data and send it to the cloud. KWS is also very popular to interact with automated vehicles. Because of the unpredictable nature of cellular networks, it is not possible to maintain the connection between the vehicle and the cloud servers all the time. As the edge KWS does not need any internet connectivity, it can interact with the vehicle without any problem. These devices also need to be robust and noise resistant to implement in the real world. Deep Neural Networks (DNNs) have shown very high accuracy in complex applications like these. As the KWS mainly deals with time-series data, Recurrent Neural Networks (RNN) provides a very good response for this type of application. But, this type of network needs high computation power and storage. The RNN neuron requires eight times more weight and complexity than that of a standard Convolutional Neural Network (CNN) [4]. As the speech signal emerges from a speaker’s mouth, nose, and cheeks, it is a one-dimensional function where air pressure varies with time. But feature extraction enables the speech signal to act as a single-channel image. Research is going on to implement CNN for KWS application and the efficiency is improving over time. These open up the possibility to successfully implement KWS on edge devices.

The DNNs have successfully been implemented on edge devices for KWS and the accuracy is going higher. In [5], [6], the authors provided a comprehensive study for different neural networks, such as CNN, LSTM, RNN, etc. It provides a comparison of different networks considering accuracy, computational complexity, and memory footprint. They also

implemented the model on embedded hardware, 32-bit ARM microcontroller and achieved the best performance for a CNN-based network, Depthwise Separable Convolutional Neural Network (DSCNN). This network is a modified version of the MobileNet. In [7], the authors demonstrated MobileNet to be very efficient in classifying 2-D spatial data, such as image classification. It demonstrated a huge reduction in the computational requirement, which makes it a good choice for resource constraint devices. In [8], the authors proposed RNN based neural network, EdgeRNN, and implemented it in an edge device, Raspberry Pi. They used 1D CNN to process the frequency domain spatial information from the speech signal. Then RNN was used to process the spatial feature data. In order to reduce the power consumption and audio processing, the authors proposed the Sinc-Convolution approach to extract the features from raw audio samples in [9]. This layer is followed by a group of DSCNN to finally classify the keyword. Alongside KWS, sound recognition can also be very helpful to monitor the behavior or health of different animals or insects. In [10], the authors used ARM Cortex-M4 based microcontroller to monitor the health of bee families by analyzing sound with an ANN.

Microcontrollers have evolved just like other computing devices. Modern microcontrollers can provide enough computational power to implement Neural Networks on them. In [11], the author implemented a very simple fully connected multi-layer Neural Network on an 8-bit cost-effective microcontroller in 2008. So, the idea of implementing DNN is not new. However, the computational complexity of DNN is increasing rapidly. In order to meet this increasing demand, microcontrollers need to be more powerful in terms of computational complexity and storage. In [5], the authors demonstrated memory and number of operations needed for different types of Neural Network. Ideal models are expected to have small memory footprint and lower number of computations needed to obtain high accuracy.

III. Audio Processing

In any speech or voice recognition application, the first or basic step is to extract features from the audio samples. It extracts the necessary linguistic information while leaving the others that carry information such as emotion, background noise, etc. Mel Frequency Cepstral Coefficient (MFCC) is a very widely used feature extraction method for speech recognition applications [12]. MFCC provides a 2D feature matrix, similar to a grayscale image, which makes it a suitable choice to implement in a Neural Network based keyword spotting system. It was inspired by the human auditory system. Humans are good at recognizing small variations in pitch when the frequencies are low, nearly 1kHz. However, it is difficult for a human to notice changes for higher frequencies. Considering this fact, the speech features can be extracted like a human ear as proposed in [12], [13]. Equation 1 converts the linear frequency to Mel-scale and vice versa for Equation 2.

$$M(f) = 1125 \ln\left(1 + \frac{f}{700}\right) \tag{1}$$

$$M^{-1}(m) = 700\left(e^{\frac{m}{1125}} - 1\right) \tag{2}$$

Google Speech Command dataset version 2 [14] is used here to train the DNN model that comes audio samples of length one second or less. It provides a very good collection of voice command from number of speakers and different type of noisy environment. The KWS is performed on an input audio signal of 1 second with a sampling rate of 16kHz. The MFCC feature extraction method involves five major steps as follows:

Step 1: First, the input samples of size 16,000 are separated into small frames of 32ms or $16000 \times 0.032 = 512$ samples. An overlapping or stride of 256 points is considered between consecutive frames. If the last frame runs out of sample, it is padded with zeros. The number of output frames, T, can be found by Equation 3.

$$T = \frac{L - l}{s} + 1 \tag{3}$$

Here, L is the length of the input audio, l is the length of each frame and s is the stride, which gives total number of frame 62.

Step 2: A Discrete Fourier Transform is taken for each of the small frames. Equation 4 is used to calculate the Fourier Transform that converts the time domain signal in frequency domain.

$$S_i(k) = \sum_{n=1}^{N} s_i(n) e^{-j2\pi kn/N}, \quad 1 \le k \le K \tag{4}$$

where, $s_i(n)$ represents the time domain small signal frame, $S_i(k)$ is the signal in frequency domain, and i is the frame number. Now, the power spectrum, $P_i(k)$, is obtained by Equation 5.

$$P_i(k) = \frac{1}{N} |S_i(k)|^2 \tag{5}$$

Step 3: A Mel-spaced filter-bank is calculated in this step. It is a set of triangular filters and applied to the periodogram estimate as obtained from step 2. Each of these filters is a vector of 257 elements. Each vector has certain elements with non-zero values only within the frequency band of interest. Except that, mostly it is populated with zeros. For instance, taking a filter-bank containing 16 filters provides a filter-bank as shown in Figure 1. It filters the signal from 0Hz to 8kHz with higher resolution for lower frequencies.

Step 4: Now it needs to implement this filter-bank to calculate energies in each band. The band energy is computed by multiplying the periodogram estimate with the filter-bank. It indicates the amount of energy in a specific filter. In order to scale down the numbers, logarithm is taken for each of the 16 energies. It provides a total of 16 log filter-bank energies.

Step 5: A Discrete Cosine Transform (DCT) is taken of these 16 log filter-bank energies. It provides 16 cepstral

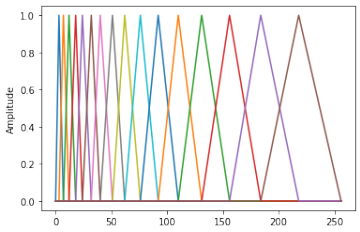


Fig. 1: Mel-spaced filter-bank of 16 filter

coefficients. But the higher order term is not usually significant as those are mostly zero. Only first 13 is kept discarding rest of the coefficients.

This illustration results a Mel Frequency Cepstral Coefficients of size 62×13. An spectrogram of all features from each frame is shown in Figure 2.

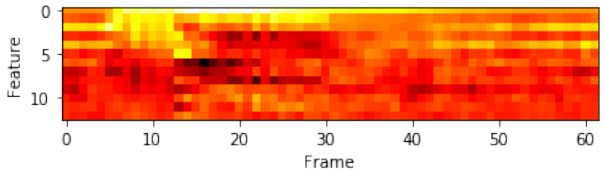


Fig. 2: Mel Frequency Cepstral Coefficients for keyword "go"

IV. DNN Model

The core layers of this DNN model is entirely based on DSCNNs, which is a comparatively newer version of separable convolution. Laurent Sifre developed the depthwise separable convolution and described detailed experimental results in his Ph.D. thesis, Section 6.2 [15]. This structure factorizes the 2D spatial convolution into depthwise convolution and pointwise convolution. The MFCC features have only 1 channel and the depthwise convolution applies a single filter on it. Then a 1×1 2D convolution is used to combine the outputs of the depthwise convolution. It reduces the model size, as well as the computational complexity of the network.

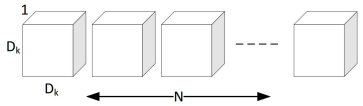
For single channel spatial data, a standard 2D convolution layer takes an input of $D_x \times D_y \times 1$ feature map F and gives $D_m \times D_n \times N$ feature map G, where D_x and D_y are width and height of the input feature map, D_m and D_n are the width and height of the output feature map, and N is the output depth.

A single channel 2D convolution kernel K have the size of $D_k \times D_k \times 1 \times N$ where D_k is the spatial kernel size assumed to be squared, and N is the number of output channels. The output feature map can be computed by Equation 6.

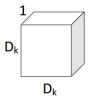
$$G_{k,l,n} = \sum_{i,j,1} K_{i,j,1,m} \cdot F_{k+i-1,l+j-1,1} \tag{6}$$

The computational cost depends on the number of output channels, input and output feature map size. It can be calculated by Equation 7.

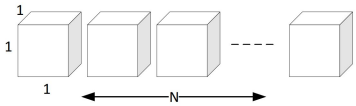
$$C_{conv2D} = D_x \cdot D_y \cdot N \cdot D_k \cdot D_k \tag{7}$$



(a) **Standard Convolution**



(b) **Depthwise Convolution**



(c) **Pointwise Convolution**

Fig. 3: Filter shape in standard convolution and depthwise separable convolution

On the other hand, the depthwise separable convolutions consist of two layers: depthwise and pointwise convolutions. For the depthwise convolution, only one filter is applied. Pointwise convolution is just like a standard 2D convolution, except the kernel size is kept 1×1. The output feature map of depthwise convolution with one filter can be calculated from Equation 8.

$$\hat{G}_{k,l,1} = \sum_{i,j} \hat{K}_{i,j,1} \cdot F_{k+i-1,l+j-1,1} \tag{8}$$

where, $\hat{K}$ is the kernel size of the depthwise convolution $D_k \times D_k \times 1$ that is applied to produce a feature map $\hat{G}$.

The computational cost of the depthwise convolution depends only on input and output feature map size, which can be calculated as $D_k \cdot D_k \cdot D_x \cdot D_y$.

This convolution only filters the inputs along the channel. In order to create a new feature map, all of the outputs need to be combined. A pointwise convolution does that task. It computes a linear combination from the outputs of the depthwise convolution and creates new features. Combined, the computational cost of this block can be found by Equation 9.

$$C_{conv\_DS} = D_k \cdot D_k \cdot D_x \cdot D_y + N \cdot D_x \cdot D_y \tag{9}$$

By taking the ratio of computation cost for depthwise separable convolution and standard 2D convolution, we get the reduction fraction of the computation as given by Equation 10.

$$\frac{C_{conv\_DS}}{C_{conv2D}} = \frac{1}{N} + \frac{1}{D_k^2} \tag{10}$$

The original MobileNet model [7] was developed for 3 channel image classification of size 224×224. However, the feature matrix returns a single channel spatial output of shape 62×13. In order to implement the MobileNet structure for this KWS application, the filter size is reduced and few layers

were discarded to avoid unnecessary computation. This model consists of 13 DS-CNN layers of different filter shape stacked on top of each other. Filter shape of depthwise convolution varies from $3\times3\times32$ to $3\times3\times512$. For pointwise convolution filter shape varies from $1\times1\times32\times32$ to $1\times1\times512\times512$. A batch normalization and rectified linear unit layer (ReLU) are added after each of these layers. Finally, this network yields 731,111 parameters among which 721,127 are trainable.

V. Hardware Platforms

The DNN model was implemented on two different platforms: 32-bit ARM Microcontroller (STM32F769NI) and (b) Robotic vehicle powered by Jetson Nano.

A. STM32F769NI Discovery Board

The STM32F769NI Discovery development board was developed by STMicroelectronics. It is based on 32-bit ARM Cortex M-7 core and share applications with the STM32F7 series microcontrollers as stated in [16]. The Cortex M series processors help to create cost-sensitive and power-constrained solutions with microcontroller. The Cortex-M7 based processor is the highest-performance member of the family as it was designed for mixed-signal devices and provides very high energy efficiency. The DSP capability makes it suitable for speech processing that needs to perform a tedious mathematical operations, like DFT or DCT. With the built-in floating-point unit, this processor can reduce power consumption and extend battery life by ten fold acceleration of single-precision, floating-point operations. Figure 4 shows the image of the development board used in this research.

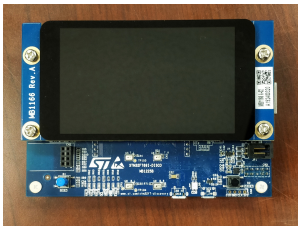


Fig. 4: STM32F769NI Discovery development board

STM32F769 Discovery board comes with four on-board microphones, which makes this device an excellent choice for voice recognition [16]. Neural Network requires relatively higher amount of flash memory and RAM, to store the model parameters or weights, compared to other embedded applications. This board provides 2 Mbytes of flash memory and 532 Kbytes of RAM, which can provide enough storage or memory requirement to classify several voice commands. Additionally, it includes 128 Mbit SDRAM, making this board more attractive for AI applications.

B. Robotic Vehicle with Jetson Nano

Voice command recognition was also performed on a robot car, commercially known as JetBot AI Robot Car [17]. Figure 5 shows the picture of the robotic vehicle. This robot is based on a single board computer device, Jetson Nano™, developed by NVIDIA®. It comes with a 128-core NVIDIA Maxwell™ GPU, which allows the tensors to perform computations in parallel. It has Quad-Core Arm Cortex-A57 MPCore processor that is built with Armv8-A architecture enabled with DSP functionality. It can perform 472 GFLOPs that is sufficient enough to run small DNN models. It has a 4 GB, 64-bit LPDDR4 RAM with a bus speed of 25.6 GB/s. It also supports external memory up to 128 GB in micro SD card. It consumes 5 Watts of power in regular mode and 10 Watts in boost mode.

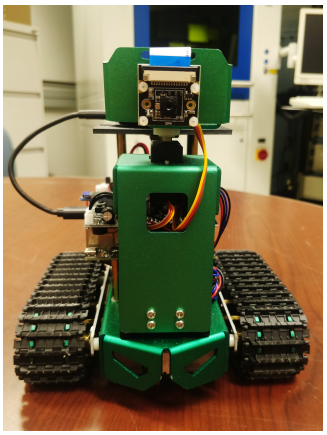


Fig. 5: JetBot AI Robot Car

However, JetBot does not have any microphone to receive the audio signals. An USB microphone array, ReSpeaker Mic Array v2.0 module [18], is used to receive the signal. The DSP processor, XMOS XVF-3000, enables to implement 4-mic mono echo cancellation algorithm to extract voice data in a challenging acoustic environment. It has an omnidirectional sensitivity of -26 dBFS, acoustic overload point of 120 dBSPL, and SNR of 63 dB.

VI. Experimental Results

In this section, we discuss the experimental results from speech signal processing to hardware implementation. As the audio processing is performed in Python and C language, a comparison is made between the outputs. Then, the accuracy of the trained model is discussed and validated the implementation in both hardware platforms.

A. MFCC Output

The speech features are extracted using MFCC. In the training dataset, the data rate was 16,000 samples per second. Although, most of the audio files were 1 second long, some did not match with the length. Shorter files are padded to 16,000 and longer are truncated in length. The samples were divided into multiple frames of length 32ms (512 samples) with a stride of 16ms (256 samples). It provided a feature-bank of 62 feature sets, each containing 13 features. During training and JetBot implementation, the audio processing is performed using Python language. While it is done in C language for bare-metal implementation using CMSIS-DSP library [19]. Figure 6 shows the plot of features extracted from

one frame for a speech command "go" using both Python 3 and C languages. It shows that the extracted features using both languages are almost identical. Although there is a small difference between these outputs, it is acceptable to implement them in this KWS application.

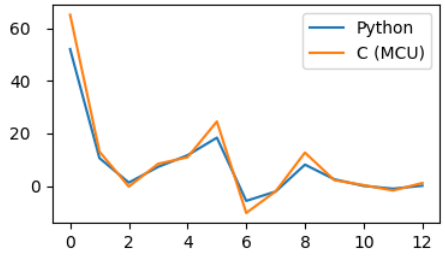


Fig. 6: Single feature from feature-bank of keyword "go"

B. Training Output

After finishing the training, the training accuracy of the model was about 93 percent and the validation accuracy was about 90 percent. A confusion matrix was generated as shown in Figure 7 by performing prediction on the test data-set. The horizontal axis indicates the predicted outputs and the vertical axis indicates the actual outputs. It also shows a heat-map for all labels, where the darker means less accurate. The keyword *stop* and *left* were predicted with higher accuracy and the *unknown* keyword was less accurate.

$$Accuracy = \frac{FP + FN}{TP + TN} \tag{11}$$

The test accuracy was calculated by Equation 11 as the ratio of false prediction (False Positive, FP and False Negative, FN) to true prediction (True Positive, TP and True Negative, TN). Using Equation 11, the test accuracy was obtained 91%.

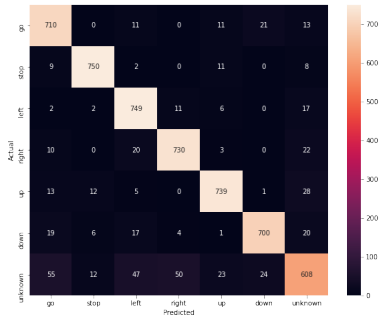


Fig. 7: Confusion Matrix on test data-set

C. Bare-metal implementation

The trained model must be converted into C equivalent code before it can be implemented in a microcontroller. It is done with XCUBE-AI tool provided by STMicroelectronics. It also includes a template, which makes implementation much easier. After converting the trained DNN model for C equivalent, it occupied only 11.52 Kbyte (2.16%) RAM and 169.63 Kbyte (8.48%) Flash of the test device. It also requires a memory complexity of 287,673 MACC. The microcontroller was programmed to provide a serial output for the predicted keywords along with the prediction score at a baud rate of 115200 bps. The predictions were received by connecting a computer with ST-LINK debugger USB port of STM32F769 microcontroller board. It took about 7ms to finish each prediction that includes all of the audio processing steps and running the trained DNN model. A summary of the bare-metal implementation is provided in Table I.

TABLE I: Summary of bare-metal implementation

Name	Value
Flash Usage	169.63 kBytes (8.48%)
RAM Usage	11.52 kBytes (2.16%)
Complexity	287673 MACC
Execution Time	6.941 ms

D. Jetbot Implementation

The trained DNN was implemented on Jetbot and the prediction was used to drive the car. This device was accessed remotely from a linux terminal and an Python script was executed to run the robot car. It received a one second speech signal at a rate of 16,000 samples per second and extracted MFCC features as described in Section III. Using TensorFlow, the trained model was used to perform prediction on extracted features. The robot car was moved using traction motors base on the predicted output. At the end of the execution, the infinite loop was closed using keyboard interruption. This exit triggered several process to terminate the connection with microphone array and motor driver. The execution of the DNN model took about 15ms, which was measured using internal timer of the Jetson Nano.

VII. Conclusion

In this paper, a small footprint Deep Neural Network-based model is proposed, trained, and implemented in two different types of edge devices. The first type of edge device includes a 32-bit ARM Cortex-M7 bare-metal microcontroller, STM32F769NI. It is designed to capture speech signals, execute a trained DNN model for KWS and transmit the prediction output through a UART terminal. The second device is Jetbot, a self-driving robot car that responds to voice commands. It also performs prediction with the KWS model, and, additionally, able to drives the robot based on prediction. The MFCC method is used to extract speech features, which results in 2D spatial data. These features are extracted using only 16 filters. For faster calculation speed, the model is implemented in C language on the bare-metal device, using a DSP library. It is implemented in Jetbot in Python 3 using the popular NumPy library. The output of MFCC is nearly identical in these two different implementations. The model is trained using the Google Speech Commands dataset version 2. It achieves a final accuracy of around 91%. The accuracy would improve even more as the filter dimension was increased while extracting the features. It would, however, raise the cost of computation. Given the network's reduced computational

complexity, this efficiency is adequate for KWS applications. The trained model is converted into C equivalent code using XCUBE-AI tool from STMicroelectronics. It takes about 7ms for the microcontroller to execute a complete prediction for KWS. However, the JetBot takes a little longer time, about 15ms, for a single prediction. Even though Jetbot has much more processing power, the longer processing time could be due to software overhead. A noise from the caterpillar treads appears while implementing the model in Jetbot. It interferes with the speech frequency bands. As a result, the model performs better when the car's treads are stationary than when it is moving. A more noise-resistant model may be a good solution for improving performance. There is also room to improve the DNN model's accuracy while keeping the computation cost and memory footprint to a minimum.

REFERENCES

[1] A. Alnoman, S. K. Sharma, W. Ejaz, and A. Anpalagan, "Emerging edge computing technologies for distributed iot systems," *IEEE Network*, vol. 33, no. 6, pp. 140–147, 2019.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] B. A. Mudassar, J. H. Ko, and S. Mukhopadhyay, "Edge-cloud collaborative processing for intelligent internet of things: A case study on smart surveillance," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[4] M. H. Samavatian, A. Bacha, L. Zhou, and R. Teodorescu, "Rnnfast: An accelerator for recurrent neural networks using domain-wall memory," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 16, no. 4, pp. 1–27, 2020.

[5] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," 2018.

[6] L. Lai and N. Suda, "Enabling deep learning at the lot edge," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–6.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[8] 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," *IEEE Access*, vol. 8, pp. 81 468–81 478, 2020.

[9] S. Mittermaier, L. Kürzinger, B. Waschneck, and G. Rigoll, "Small-footprint keyword spotting on raw audio data with sinc-convolutions," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7454–7458.

[10] V. K. Abdrakhmanov, R. B. Salikhov, and K. V. Vazhdacv, "Development of a sound recognition system using stm32 microcontrollers for monitoring the state of biological objects," in *2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*, 2018, pp. 170–173.

[11] N. J. Cotton, B. M. Wilamowski, and G. Dundar, "A neural network implementation on an inexpensive eight bit microcontroller," in *2008 International Conference on Intelligent Engineering Systems*, 2008, pp. 109–114.

[12] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE transactions on acoustics, speech, and signal processing*, vol. 28, no. 4, pp. 357–366, 1980.

[13] W. Han, C.-F. Chan, C.-S. Choy, and K.-P. Pun, "An efficient mfcc extraction method in speech recognition," in *2006 IEEE international symposium on circuits and systems*. IEEE, 2006, pp. 4–pp.

[14] "Speech commands dataset version 2," [Online]. Available: http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz [Accessed: 30-May-2022].

[15] L. Sifre, "Rigid-motion scattering for image classification," Ph.D. dissertation, CMAP, Ecole Polytechnique, Palaiseau, France, 2014.

[16] "Discovery kit with stm32f769ni mcu," [Online]. Available: https://www.st.com/en/evaluation-tools/32f769idiscovery.html [Accessed: 30-May-2022].

[17] "Jetbot ai robot car," [Online]. Available: http://www.yahboom.net/study/JETBOT [Accessed: 30-May-2022].

[18] "Respeaker mic array v2.0," [Online]. Available: https://www.seeedstudio.com/ReSpeaker-Mic-Array-v2-0.html [Accessed: 30-May-2022].

[19] "Cmsis dsp software library," [Online]. Available: https://www.keil.com/pack/doc/CMSIS/DSP/html/index.html [Accessed: 30-May-2022].