Implementation of Deep Learning models for Keyword Spotting on the edge

Dissertation
in partial fulfillment of the requirements for the degree of
Master of Technology
in
Electronic Systems

by
Shyama P
(183079031)

Under the guidance of

Prof. Rajbabu Velmurugan

Department of Electrical Engineering
Indian Institute of Technology Bombay
2020-21

Dissertation Approval

This Dissertation entitled **Implementation of Deep Learning models for Keyword Spotting on the edge** by **Shyama P (183079031)** is approved for the degree of **Master of Technology**.

Examiners:

.....

.....

.....

.....

Supervisor:

.....

Chairperson:

.....

Date: .....

Place: .....

Declaration

I declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date:

Shyama P
183079031

Abstract

Developments in Deep Learning (DL) have made considerable performance improvements in vision and speech-based tasks over traditional approaches. KeyWord Spotting (KWS) is an essential component in enabling speech-based user interactions on smart devices and neural networks have recently been a popular choice for KWS design. The KWS application has a very limited power budget due to its always-on nature and demands a real-time response for a good user experience. These requirements make microcontroller-powered edge devices a good choice for deploying KWS systems. However, the compute/memory/power-intensive nature of neural networks makes it challenging to implement them on low-power, memory/performance constrained microcontroller devices.

This project explores the various approaches for implementing DL-based KWS systems on microcontrollers and implements an end-to-end system on an ARM Cortex M7 microcontroller platform. The performance of the system is evaluated and compared with existing works.

Acknowledgements

I would like to express my sincere gratitude towards my guide, **Prof. Rajbabu Velmurugan**, for allowing me to work on this project under his supervision. His continuous guidance and support have helped me throughout this project. I would also like to thank my friends and family for their constant support and encouragement.

Contents

List of Figures

List of Tables

Abbreviations

CNN	**C**onvolutional **N**eaural **N**etwork
CRNN	**C**onvolutional **R**ecurrent **N**eural **N**etwork
DFSDM	**D**igital **F**ilter for **S**igma-**D**elta **M**odulator
DFT	**D**iscrete **F**ourier **T**ransform
DL	**D**eep **L**earning
DNN	**D**eep **N**eural **N**etwork
DS-CNN	**D**epthwise **S**eparable **C**onvolutional **N**eural **N**etwork
FC	**F**ully **C**onnected
FFT	**F**ast **F**ourier **T**ransform
FPU	**F**loating **P**oint **U**nit
KWS	**K**ey**W**ord **S**potting
LFBE	**L**og-mel **F**ilter **B**ank **E**nergy
MCU	**M**icro**C**ontroller **U**nit
MEMS	**M**icro-**E**lectro-**M**echanical **S**ystems
MFCC	**M**el **F**requency **C**epstral **C**oefficient
NN	**N**eural **N**etwork
PCEN	**P**er **C**hannel **E**nergy **N**ormalised
PCM	**P**ulse **C**ode **M**odulation
PDM	**P**ulse **D**ensity **M**odulation
ReLU	**Re**ctified **L**inear **U**nit
ResNet	**Res**idual **Net**work
VAD	**V**oice **A**ctivity **D**etector

Chapter 1

Introduction

1.1 Motivation

Deep Learning (DL) on Edge refers to data processing using DL algorithms near the network's edge, i.e., closer to the data source, instead of in a centralized facility. Traditionally, the edge nodes generate data and send it to remote servers for processing, and the remote servers transmit the processed information back. This approach increases the network overhead, especially in scenarios where the generated data volume is large, like a video surveillance system, or when the number of devices sending data is large, like home automation systems. It also introduces additional latency in the edge device's response, which can be critical in scenarios such as obstacle detection in a self-driving car. The extensive use of DL algorithms in various applications involving computer vision and audio processing makes it necessary to implement them on the edge devices to leverage their full potential.

Some of the advantages of implementing DL on edge are as follows.

- Volume of data transferred to cloud decreases (especially in scenarios where the volume of data transferred is large like video surveillance systems), i.e. lower network overhead
- Lower latency improves real-time performance (network latency is absent)

- Lower power consumption (no network transfer)
- Enhanced security and privacy (all data is processed on edge node and not send to remote servers)
- Decentralization - make systems more robust by providing service during network failure/ cyber attack.

KeyWord Spotting (KWS) is an application that needs implementing deep learning on edge. KWS refers to identifying a set of predefined commands (usually a small set) from a stream of input audio. Such a system can complement an automatic speech recognition (ASR) system by acting as a trigger (e.g., detect "Alexa", "Ok Google") or can be the speech-based user interface for smart devices (e.g., detect commands like "ON" and "OFF" for a smart lamp). Due to the always-on nature of the KWS system, it has an extremely low power budget; yet demands real-time performance and high accuracy for a good user experience. Besides, the transmission of user audio continuously to the cloud can cause serious privacy concerns. These requirements make microcontroller-powered edge devices a good choice for deploying KWS systems.

The major challenges of implementing DL on microcontrollers are listed below.

- Limited memory footprint: Typical microcontroller systems have a maximum memory of a few hundred kBs of on-chip SRAM and up to a few MBs of on-chip flash [1]. The flash memory stores the program binary, which contains the code and neural network model, including input/output buffers, weights, and activations. While execution, the program binary is loaded into the SRAM, and the SRAM is used as the main data memory. Hence, the SRAM size limits the size of the memory that can be used during inference.
- Limited computational resources: Low-power microcontrollers operate upto a few hundred MHz system clock frequency [1] and have limited computing resources. The real-time response requirement further limits the total number of operations per neural network inference.

- Latency: Real-time response requirement puts a tight constraint on the neural network execution time.
- Low power consumption: As smart devices are often battery operated and always-on, a limited power budget is available.

1.2 Research objectives

The objective of this project was to explore the various approaches for the implementation of deep learning systems on microcontroller platforms, identify a suitable task and platform and implement an end-to-end deep learning system on it.

1.3 Contributions

The project explores various approaches for the implementation of deep learning systems on microcontroller platforms. We implement an end-to-end KeyWord Spotting system using the Deep Residual Network-based neural network architecture [2], on the STM32F769I-DISC1 discovery board based on the ARM Cortex M7 processor and compares the results with existing works [1].

1.4 Organization of the report

Chapter 2 provides an overview of the existing neural network-based keyword spotting architectures in literature, which were considered to implement the end-to-end system. Chapter 3 describes the various approaches explored for implementing deep learning on microcontrollers. In Chapter 4, a detailed overview of the end-to-end KWS system is included. Finally, Chapters 5 and 6 discuss the results obtained, conclusion, and future scope.

Chapter 2

Literature review

2.1 Deep Learning (DL) based KeyWord Spotting (KWS) architectures

Recent advances in deep learning for speech-based tasks [3] and the availability of a public dataset specifically for keyword spotting [4] have attracted multiple works in the area of neural network-based keyword spotting. This section delves into some of the relevant works. These works were chosen based on their ease of implementation on a microcontroller (memory footprint, computational requirements, etc.), and results. The works can be summarised based on the three stages of the DL based KWS architecture - feature extractor, neural network-based classifier, and posterior probability handling module, as shown in Figure 2.1.

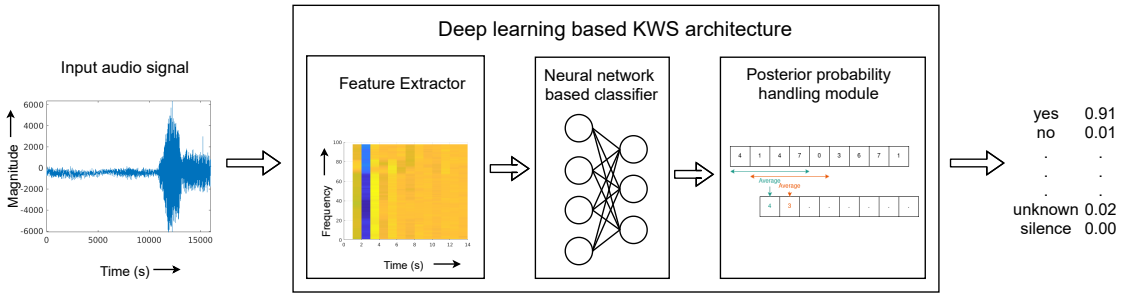


FIGURE 2.1: Stages in a deep learning based keyword spotting system

Feature extractor stage: The feature extractor stage converts the input audio signal into a 2D spectrogram enabling dimensionality compression of the neural network input. Some works, like [5] exclude the feature extractor stage and feed the audio signal directly to the NN, but the architecture considered here includes a feature extractor.

Figure 2.2 depicts the computation of overlapping time frames from input audio signal for generating speech features. In order to compute the speech features, the input audio signal (length L) is first framed into short overlapping frames (length l), with stride s. A total of T overlapping time frames are obtained by doing so, where

$$T = \frac{(L - l)}{s} + 1$$

F speech features are extracted from each of the T frames, generating a total of $T \times F$ features. The popular feature extractors used in literature are Mel Frequency Cepstral Coefficients (MFCC), Log-mel FilterBank Energies (LFBE), and Per Channel Energy Normalised (PCEN) [6][7][8][1][2].

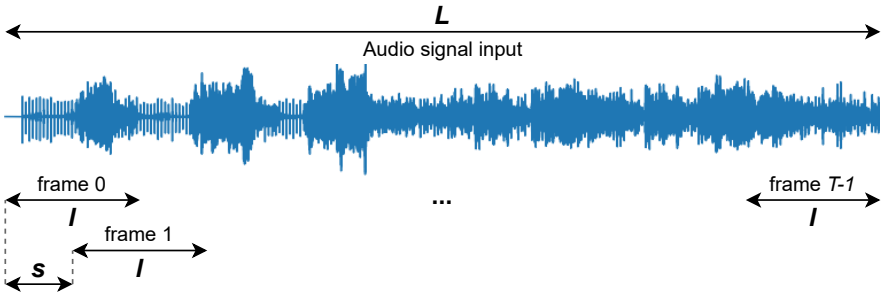


FIGURE 2.2: Computation of overlapping time frames from input audio signal for generating speech features.

Neural Network (NN) based classifier stage: The neural network-based classifier stage uses the extracted speech feature matrix to compute probabilities for the output classes (keywords). The stage uses different NN architectures like DNN, CNN, and ResNet for classification.

Posterior probability handling module: The posterior probability handling

module estimates the output class (keyword) probabilities over a period of time, improving the overall prediction on a continuous audio signal. It usually uses an averaging filter or a separate neural network [6][7][9].

2.1.1 Deep Neural Network (DNN) based KWS [6]

The DNN based KWS model using fully connected layers and ReLu activations was introduced in [6]. The network was used for a three class classification of incoming audio signals. The three classes used were: okay, google, "filler" (representing any class other than "okay" and "google"). The feature extractor, NN based classifier, and posterior probability handling stages used in [6] are described next.

Feature extractor stage: The input audio is first fed to a Voice Activity Detector (VAD). Only those audio frames with recognized vocal activity are analyzed further. The feature extractor computes 40 dimension Log-Mel Filter Bank Energy (LFBE) over a 25 ms audio frame. 25 ms overlapping audio frames are chosen with a 10 ms stride from a continuous audio stream to obtain the DNN input. The current audio frame is concatenated with 30 previous frames and 10 future frames to form input to the NN stage, as shown in Figure 2.3.

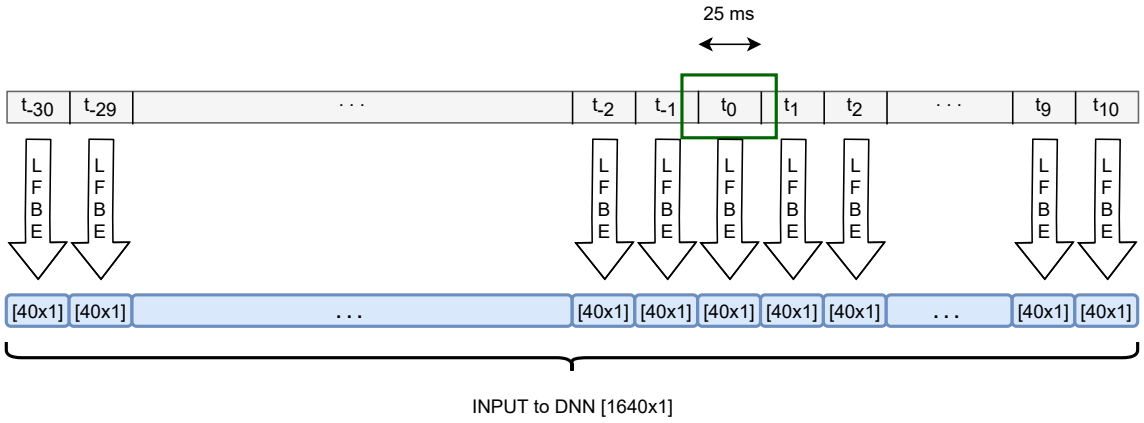


FIGURE 2.3: DNN input computation. Here t_0 represents the current time window, t_{-1} to t_{-30} past time windows and t_1 to t_{10} future time windows. Note that the time windows are overlapping, i.e. if t_0 represents time from 0 ms to 25 ms, t_1 represents time from 10 ms to 35 ms, etc.

Total input time frame considered per output $= 10 \times 30 + 25 + 10 \times 10 = 425$ ms

NN input vector size $= 40 \times (30 + 1 + 10) = [1640 \times 1]$

NN based classifier stage: A DNN based classifier is used with 3 fully connected layers (FC) of 128 neurons each and ReLU activations (Figure 2.4). The final output layer uses softmax activation to generate output probabilities.

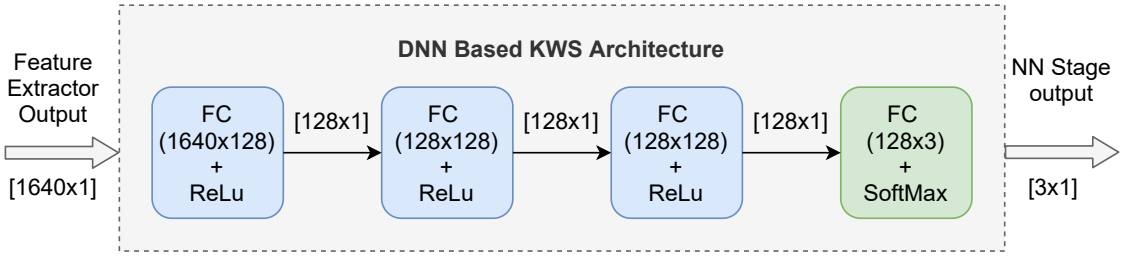


FIGURE 2.4: Model architecture of DNN [6]. FC represents a Fully Connected layer. The number of neurons in the FC layer is adjusted to tune the model size/ accuracy.

Posterior handling module: As given in Figure 2.5, the posterior handling module has 2 parts, posterior smoothing and confidence computation.

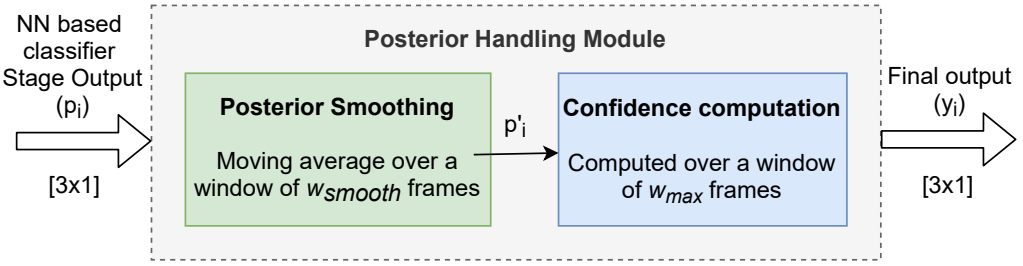


FIGURE 2.5: Posterior handling module for DNN/CNN based KWS architecture

Posterior smoothing is computed as a moving average of DNN stage outputs over a window of w_{smooth} frames, and is described by Equation 2.1. This is done inorder to reduce noise in the DNN stage output.

$$p'_i = \frac{1}{w_{smooth}} \sum_{k=i-w_{smooth}+1}^{i} p_k \tag{2.1}$$

where p_i is the output of the DNN stage for i^{th} frame (current frame), p'_i is the output of posterior smoothing and w_{smooth} is the window over which the smoothing is performed.

The confidence score (final output of the system) is computed on the smoothed posterior within a sliding window of size w_{max}, as given in Equation 2.2.

$$y_{ij} = \sqrt[n-1]{\prod_{m=1}^{n-1} max(p'_{kj}); i - w_{max} + 1 \leq k \leq i} \tag{2.2}$$

where $p'ij$ is the output of posterior smoothing for the i^{th} frame and j^{th} label, max() computes the maximum value over the range $i - w_{max} + 1 \leq k \leq i$, and n is a constant.

w_{smooth} is chosen to be 30 and w_{max} is chosen to be 100 in the original work [6] based on experimental results.

2.1.2 Convolutional Neural Network (CNN) based KWS [7]

The fundamental disadvantage of DNN based approach for KWS is that it neglects the local temporal and spectral correlation of input speech data. The convolutional neural network (CNN) based KWS described in [7] exploits these correlations and demonstrated a higher accuracy than the DNNs. The network was evaluated for a 15 class classification of incoming audio signals. The 15 classes used were: “answer call”, “decline call”, “email guests”, “fast forward”, “next playlist”, “next song”, “next track”, “pause music”, “pause this”, “play music”, “set clock”, “set time”, “start timer”, “take note” and “filler” (represents any class other than the keywords). The feature extractor, NN based classifier, and posterior probability handling stage used in [7] are described next.

Feature extractor stage: The input audio is first fed to a VAD. Only those audio frames with recognized vocal activity are analyzed further. The feature extractor computes 40 dimension LFBE over a 25 ms audio frame. 25 ms overlapping audio

frames are chosen with 10 ms stride from a continuous audio stream to obtain the CNN input. The current audio frame is concatenated with 23 previous frames and 8 future frames to form input to the NN stage, as shown in Figure 2.6.

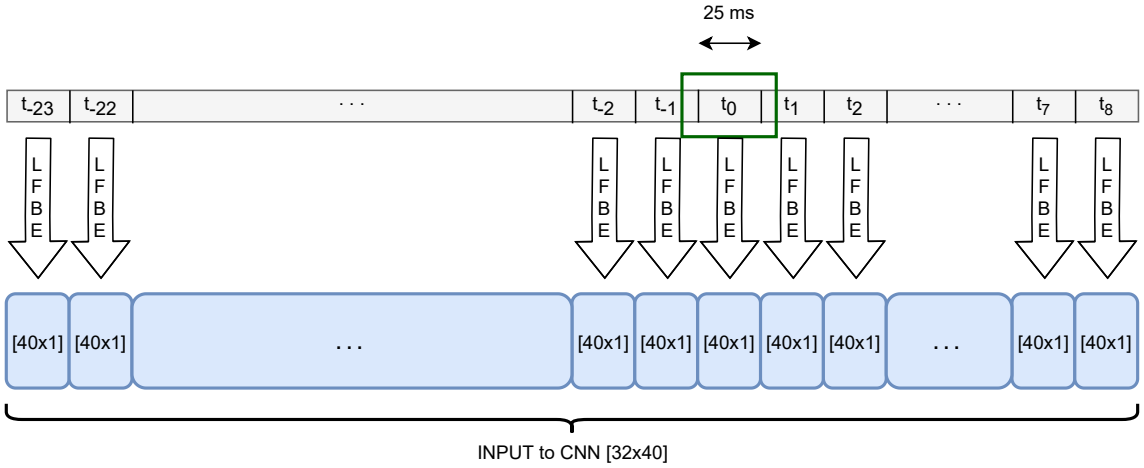


FIGURE 2.6: CNN input computation. Here t_0 represents the current time window, t_{-1} to t_{-23} past time windows and t_1 to t_8 future time windows. Note that the time windows are overlapping, i.e. if t_0 represents time from 0 ms to 25 ms, t_1 represents time from 10 ms to 35 ms, etc.

Total input time frame considered per output $= 10 \times 23 + 25 + 10 \times 8 = 335$ ms

NN input vector size $= (23 + 1 + 8) \times 40 = [32 \times 40]$

NN based classifier stage: The generic CNN Architecture used is given in Figure 2.7. Multiple permutations of the CNN architecture are explored in the work [7].

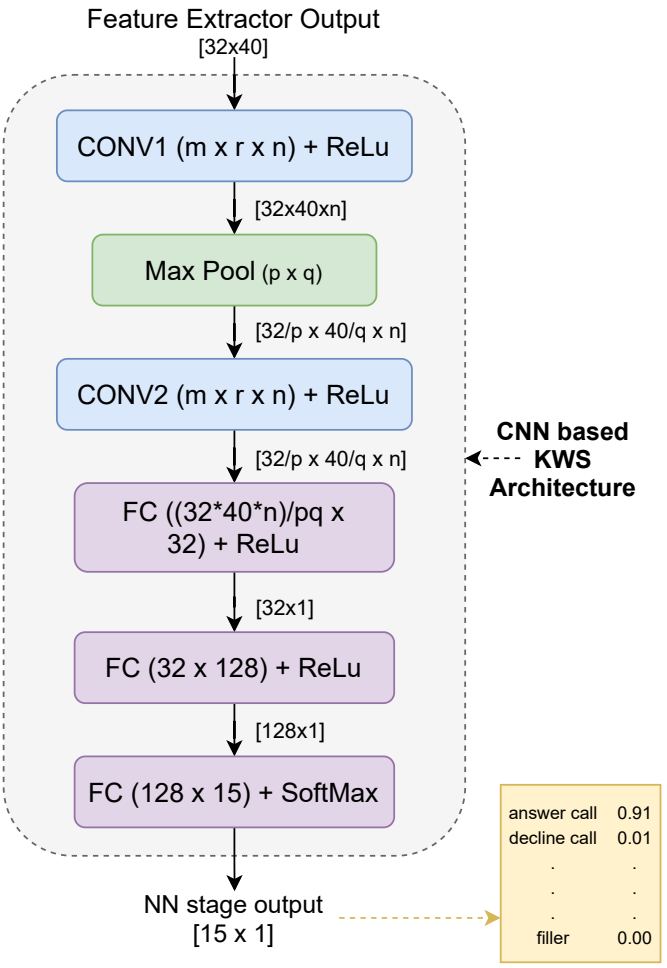


FIGURE 2.7: Model architecture of CNN [7]. CONV represents Convolutional layer and FC represents Fully Connected layer. The values of m,n,r,p,q are adjusted to obtain multiple permutations of the CNN architecture.

Posterior handling module: The posterior handling module used in the CNN based KWS architecture is same as the module used in the DNN based KWS architecture described in Section 2.1.1

2.1.3 Convolutional Recurrent NN (CRNN) based KWS [8]

When modelling time-varying signals (such as speech), CNNs have the disadvantage of ignoring long-term temporal dependencies. In [8], convolutional recurrent neural network based KWS is examined, combining the strengths of CNNs and Recurrent Neural Networks (RNNs). The network was used to identify the occurrence of the

keyword "TalkType" from the input audio signal. The feature extractor, and NN based classifier used in [8] are described next. The work does not mention the use of a posterior probability handling module.

Feature extractor stage: The feature extractor computes 40 dimension Per Channel Energy Normalised (PCEN) mel spectrogram over a 10 ms audio frame, on 1.5 sec audio for single input to the NN stage, as shown in Figure 2.8. A 100 ms stride is used between frames.

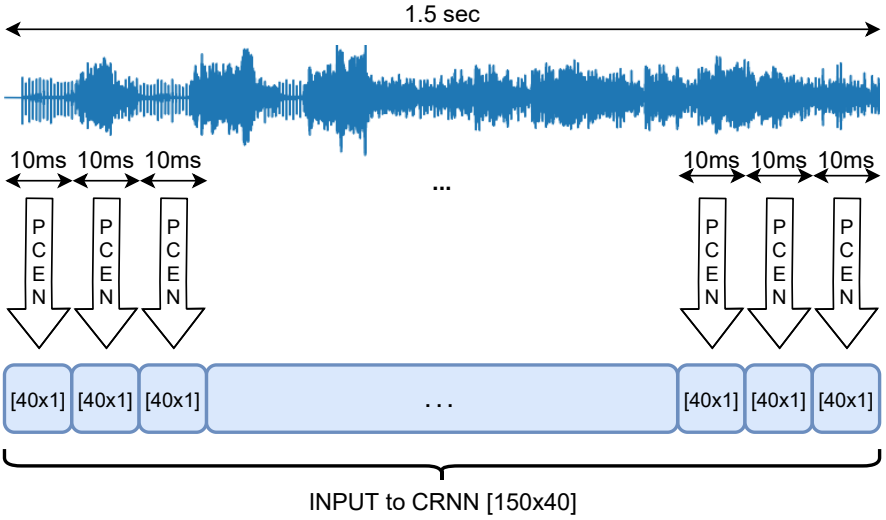


FIGURE 2.8: CRNN input computation

Total input frame size considered per output = 1.5 sec

NN input vector size = $[150 \times 40]$

NN based classifier stage

The generic CRNN Architecture used is given in Figure 2.9.

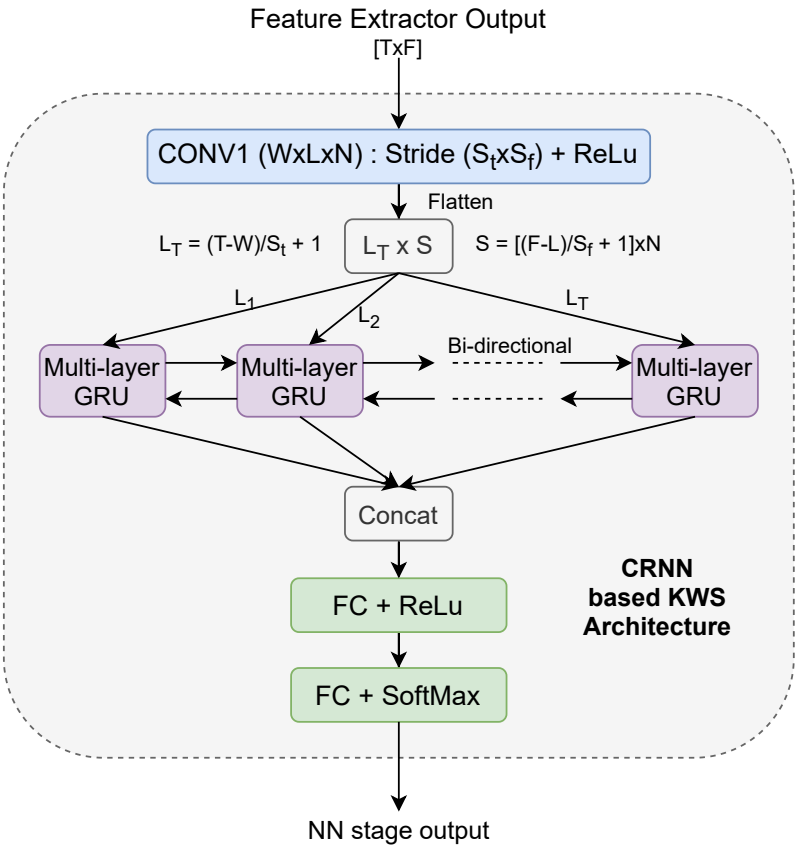


FIGURE 2.9: Model architecture of CRNN [8].

2.1.4 Depthwise Separable CNN (DS-CNN) based KWS [1]

In the field of computer vision, depthwise separable convolution has been proposed as an efficient alternative to the usual 3D convolution operation [10] and has been utilised to construct compact network designs. [1] explores the use of DS-CNN for KWS and compares its performance against other approaches (DNN, CNN, RNN, CRNN) from the perspective of implementing on microcontrollers.

Depthwise separable convolutional layer first convolves each channel in the input feature map with a separate 2D filter and then uses pointwise convolutions (1×1) to combine the outputs in the depth dimension. By decomposing the standard 3D convolutions into 2D convolutions followed by 1D convolutions, depthwise separable convolutions are more efficient both in number of parameters and operations [10], making deeper and wider architecture possible on resource-constrained

microcontroller devices.

The DS-CNN based network was evaluated by classifying the incoming audio signal into one of the 12 classes namely - yes, no, up, down, left, right, on, off, stop, go and "silence" or "unknown", from the Google Speech Commands dataset [4]. The feature extractor, and NN based classifier used in [1] are described next. The work does not mention the use of a posterior probability handling module.

Feature extractor stage: The feature extractor computes 10 dimension MFCC over a 40 ms audio frame. 40 ms overlapping audio frames are chosen with 20 ms stride from a continuous audio stream inorder to obtain the DS-CNN input, as shown in Figure 2.2 with $L = 1$ sec, $l = 40$ ms and $s = 20$ ms. A total of 1 sec audio is considered to compute the input features for the NN stage.
Total input time frame considered per output = 1 sec
NN input vector size = $[49 \times 10]$

NN based classifier stage: The generic DS-CNN architecture used is given in Figure 2.10.

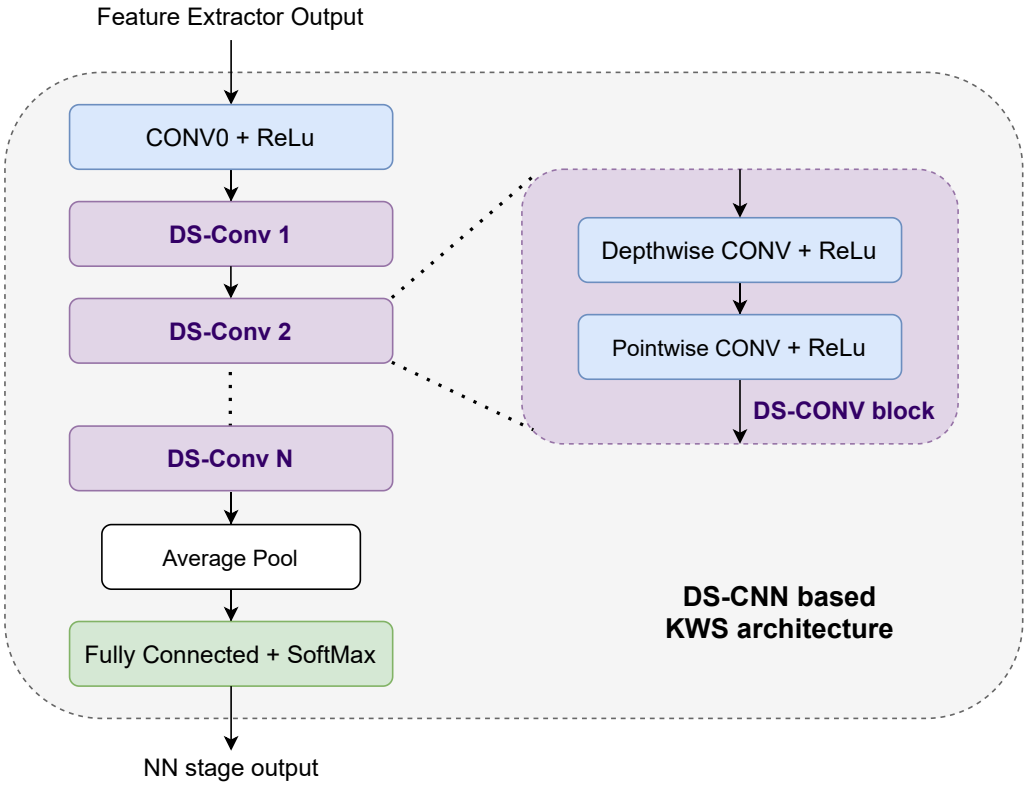


FIGURE 2.10: Model architecture of DS-CNN [1]. CONV represents Convolutional layer, DS-Conv represents Depthwise Separable Convolutional layer.

2.1.5 Deep Residual Network (ResNet) based KWS [2]

Deep residual networks (ResNets) [11] are a game-changing advancement in deep learning that has allowed researchers to train deeper networks effectively. They were first used in image recognition, where they helped to boost state-of-the-art performance significantly [11]. ResNets have since been used to identify speakers [12] and recognise speech automatically [13]. The application of deep residual learning techniques to the job of keyword detection is investigated in [2].

The network was evaluated by classifying the incoming audio signal into one of the 12 classes namely - yes, no, up, down, left, right, on, off, stop, go and "silence" or "unknown", from the Google Speech Commands dataset [4]. The feature extractor, and NN based classifier used in [2] are described next. The work does not

mention the use of a posterior probability handling module.

Feature extractor stage: The feature extractor computes 40 dimension MFCC over a 30 ms audio frame. 30 ms overlapping audio frames are chosen with 10 ms stride from a continuous audio stream inorder to obtain the ResNet input, as shown in Figure 2.2 with $L = 1$ sec, $l = 30$ ms, and $s = 10$ ms. A total of 1 sec audio is considered to compute the input features for the NN stage.

Total input time frame considered per output = 1 sec

NN input vector size = $[98 \times 40]$

NN based classifier stage: The generic ResNet architecture used is given in Figure 2.11. The project further uses one of the variants of this architecture referred to as res8-narrow [2]. The details of this model and reason for choosing this model as baseline is explained in Section 3.2.1.

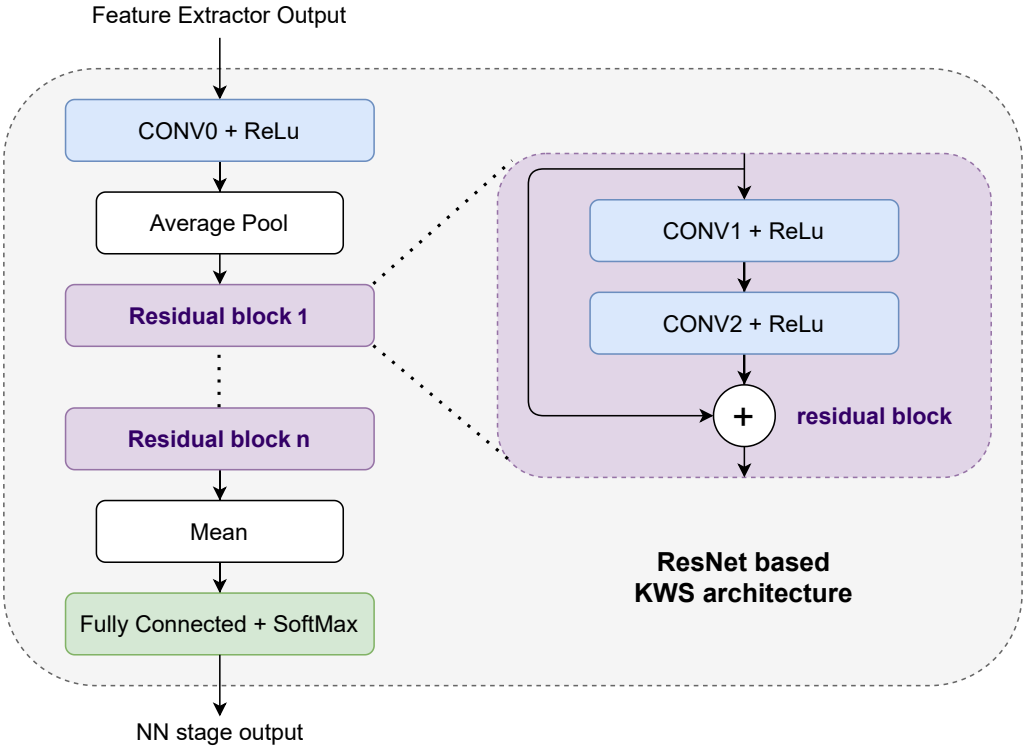


FIGURE 2.11: Model architecture of ResNet based KWS architecture [2]. CONV represents Convolutional layer.

2.2 Implementation of NN on microcontroller

The very low power budget of always-on tasks like keyword spotting makes microcontrollers an ideal choice for their implementation. Major challenges faced while implementing neural networks on micrcontrollers are their low memory footprint and limited computational resources. Approaches followed in literature to tackle these challenges are listed below.

- Use of parameter efficient NN architectures MobileNet, SqueezeNet, etc.
- Compression and quantization of existing neural network models

The popular neural network frameworks such as PyTorch, and TensorFlow, used for implementing DL models, cannot be directly used for deploying the models to microcontrollers due to their large library size (Table 2.1) and lack of support on microcontrollers. Some libraries available for deploying NN models to ARM Cortex M microcontrollers are studied in the following sections.

Framework	**Library size**
Tensorflow/ PyTorch	100 - 300 MB
Tensorflow Lite/ Pytorch mobile	1 - 2 MB
Tensorflow Lite for Microcontrollers/ Edge Impulse / CMSIS-NN	$<$ 300 kB

Table 2.1: Comparison of library size of various DL frameworks

2.2.1 CMSIS-NN library [14]

As this project explores implementation of the Deep Residual Network based NN model (Section 2.1.5) on an ARM Cortex M microcontroller, the CMSIS-NN library from ARM for Cortex M was studied.

The CMSIS-NN library is a collection of efficient neural network kernels that maximizes the performance and minimizes the memory footprint of neural networks on Cortex-M processor cores. It supports 8b/16b/32b fixed point computations for various NN layers.

Steps followed to convert a neural network to CMSIS-NN are given below.

- Check if all layers in NN are supported
- Replace unsupported NN layers with an equivalent combination of CMSIS-NN and CMSIS-DSP functions
- Compute activation statistics inorder to fix a quantization scheme and word size for the NN weights
- Compute the Q-formats (fixed point format) for each NN layer
- Compute input and output Q-format for each layer based upon the quantization scheme and the layer constraints
- Rewrite the forward propagation for the NN using CMSIS-NN library functions

2.2.2 TensorFlow Lite for Microcontrollers (TFLM) framework [15]

TensorFlow Lite for Microcontrollers (TFLM) is a machine learning framework designed to run on microcontrollers and other devices with limited memory. It doesn't need an operating system, standard `C` or `C++` libraries, or dynamic memory allocation and is free and open source. It is written in `C++11` and supports a wide range of 32b microcontrollers. Also, it has been extensively tested with a variety of processors based on the ARM Cortex-M Series architecture.
The workflow involved in deploying a Tensorflow based DL model to a microcontroller using TFLM is as follows [15].

- **Train a model:**
  - Generate a small TensorFlow model that can fits the target device and contains supported operations
  - Convert to a TensorFlow Lite model using the TensorFlow Lite converter

  - Convert to a `C` byte array using standard tools to store it in a read-only program memory on device

- **Run inference** on device using the TFLM `C++` library and process the results

The limitations of TFLM are given below.

- Support for a limited subset of TensorFlow operations
- Support for a limited set of devices
- Low-level `C++` API requiring manual memory management
- On device training is not supported

2.2.3 Edge Impulse platform[16]

Edge Impulse is a developer platform that helps in the end-to-end development and deployment of neural networks to microcontrollers. It supports the training of neural networks using the Keras API and 8b quantization of the NN model and generates `C++` code with the source libraries that can be used on the microcontroller. The code generation framework of Edge Impulse uses TFLM and CMSIS-NN in the backend to get optimized performance on the microcontroller.

Chapter 3

DL based KWS on microcontroller

The workflow involved in the design, development and deployment of a deep learning model for keyword spotting on to a microcontroller board is shown in Figure 3.1.

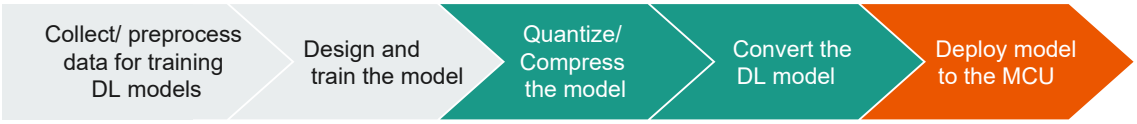


FIGURE 3.1: Workflow involved in the design, development and deployment of a DL model on a microcontroller

3.1 Data collection/ pre-processing

3.1.1 Dataset for training

The most popular open source dataset available/used for training NN based KWS models in literature is the Google Speech Commands dataset [4]. The dataset consists of 1,05,829 utterances of 35 words namely "Backward", "Bed", "Bird", "Cat", "Dog", "Down", "Eight", "Five", "Follow", "Forward", "Four", "Go", "Happy", "House", "Learn", "Left", "Marvin", "Nine", "No", "Off", "On", "One", "Right", "Seven", "Sheila", "Six", "Stop", "Three", "Tree", "Two", "Up", "Visual", "Wow", "Yes", "Zero". Each utterance is stored as a one-second long audio file in WAV format, with the sample data encoded as linear 16b single-channel PCM values, at a

16 kHz rate. The utterances are from around 2,618 different speakers. Ten words from these were chosen as a valid classes‘ (“Yes”, “No”, “Up”, “Down”, “Left”, “Right”, “On”, “Off”, “Stop”, and “Go”) for training the networks. The remaining words were assigned a class “unknown”. A 12^{th} class called “silence” was also introduced to identify “no voice activity”.

3.1.2 Data pre-processing

The raw audio waveform is not the best choice as input to the NN as the loudness can largely impact the samples, and the input dimension will also be high ([16000×1]). Hence MFCC of the audio signal is computed before feeding it to the NN. More details of the exact choice of MFCC parameters used is given in Section 4.4.

3.2 Design and training of the DL model

3.2.1 Choice of Deep Residual Network as baseline

The following NN based architectures were considered while choosing the baseline model for the project.

- CNN based NN for KWS (Section 2.1.2)
- DS-CNN based NN for KWS (Section 2.1.4)
- Deep Residual Network based NN for KWS (Section 2.1.5)

A comparison of the specific models considered for baseline, based on model accuracy, memory footprint and number of operations involved is given in Table 3.1. As the task needs to be extended to a larger set of keywords, it will be essential to train a deeper NN, without largely increasing the memory footprint. One of the factors that favored the choice of the ResNet based model is the better tunability of the model by controlling the depth (number of network layers) as well as width (number of features per layer) of the model, without largely impacting its memory

footprint. Also, ResNets have demonstrated better ability to efficiently train deeper networks [11].

The idea of depthwise separable convolutional layers originally introduced in [10] and explored for the task of KWS in [1] may help reduce the memory footprint and number of operations of the ResNet based models further. The effectiveness of depthwise separable convolutions in ResNet architecture is yet to be explored.

NN Model	**Accuracy**	**Memory footprint**	**Operations**
CNN	91.6 %	79.0 kB	5.0 M
DS-CNN	94.4 %	38.6 kB	5.4 M
res8-narrow*	90.1 %	79.6 kB	5.65 M
res8*	94.1 %	440 kB	30 M
res15-narrow*	94.0 %	170.4 kB	160 M

TABLE 3.1: Comparison of accuracy, memory footprint and number of multiplication operations for different models considered for baseline. Memory footprint refers to the total size of the NN weights. Results are taken from [1][2]. *Results are based on 32b floating point models as results for 8b quantized models are not available.

The specific model used for the further system implementation is res8-narrow, shown in Figure 3.2 [1].

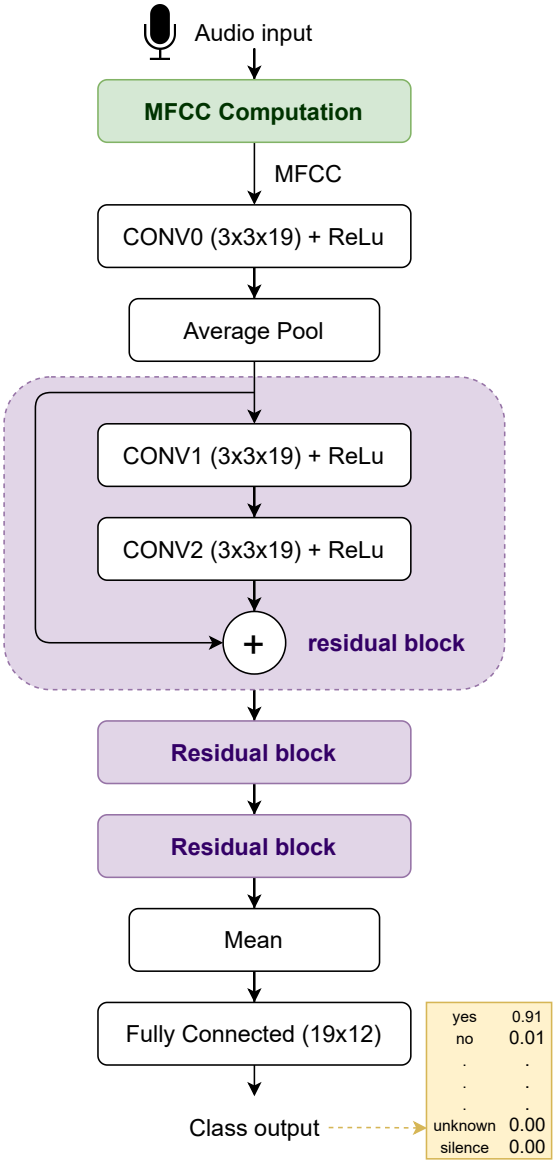


FIGURE 3.2: res8-narrow NN model

3.2.2 Model/training details

The task formulated for evaluating the end-to-end system is classification of the incoming audio signal into one of the 12 classes namely - yes, no, up, down, left, right, on, off, stop, go, "silence" or "unknown".

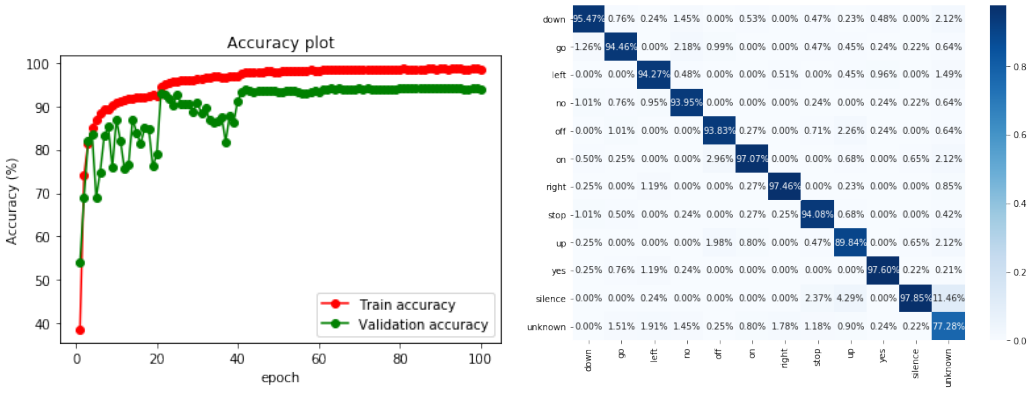
NN Architecture: The architecture chosen was the res8-narrow architecture from [2] shown in Figure 3.2. The original 32b floating point model was converted to 8b fixed point during implementation on microcontroller.

Dataset: The dataset used for training the NN was the Google Speech Commands Dataset v2.0 [4] (More details in Section 3.1.1).

Training: The original NN was implemented and trained using the PyTorch framework[17][18]. Inorder to facilitate the use of Tensorflow Lite for Microcontrollers library and Edge Impulse platform, the model was retrained using Keras API in Tensorflow.

3.2.3 Training results

The accuracy plot and confusion matrix obtained on retraining the res8-narrow model using Tensorflow is shown in Figure 3.3. The final test accuracy obtained was 93.59%.



(a) Validation accuracy plot for res8-narrow model during training

(b) Confusion matrix of the trained model, evaluated on test dataset

FIGURE 3.3: Training results of the res8-narrow NN model

3.3 Quantization/ compression of DL model

The trained 32b floating point model is quantized to 8b fixed point before implementing on the microcontroller. This helps in reducing the memory footprint of the NN model by 4 times and improving the inference time (fixed point operations are faster than floating point operations). The model size of the res8-narrow model

before and after quantization is shown in Table 3.2. The quantization of the model is achieved using the TFLM and Edge Impulse frameworks.

Model	**Floating point model size**	**8b quantized model size**
res8-narrow	80 kB	20 kB

TABLE 3.2: res8-narrow model memory footprint before and after 8b quantization

Quantization can be performed during training (quantization-aware training) or after training on the trained NN model (post-training quantization). Post-training quantization is performed in this project with full integer quantization, i.e., both the neural network weights and activations are quantized to 8b integer format. This helps in reducing the latency (as integer operations are performed instead of floating-point operations) and peak memory usage (as activations are also 8b integers compared to 32b floating point). Post-training quantization does not cause a significant reduction in accuracy of the neural network model, as seen in Table 3.3. The steps involved in post-training quantization of a trained 32b floating point model is given in Algorithm 1. While estimating the fixed point format for activations in Algorithm 1, any outliers in the representative dataset can increase the minimum-maximum range. This can impact the accuracy of the quantized model. Hence the "quantize activations" step needs to be revisited to obtain best quantized model accuracy.

NN Model	**NN input dimension**	**32b floating point model accuracy (%)**	**8b fixed point model accuracy (%)**
DNN [6]	25×10	78.2	77.0
DS-CNN [1]	49×10	93.4	92.4
res8-narrow [2]	49×10	89.0	87.0

TABLE 3.3: Comparison of reduction in accuracy for various NN models when quantized from 32b floating point to 8b fixed point (post-training full integer quantization) using TFLM

Algorithm 1: Post-training full integer quantization steps

input **:** 32b floating point trained NN model (model)

quantize weights **:**

while *weight in model* **do**

min = minimum(weight), max = maximum(weight)

integer bits = ceil(log2(maximum(absolute(min),absolute(max))))

fraction bits = 7-integer bits

weight = round(weight*(2**fraction bits))

end

quantize activations:

initialize max, min for each layer in model layers

while *sample in representative dataset* **do**

run inference on 32b floating point model

while *layer in model layers* **do**

max temp = maximum(layer activation)

min temp = minimum(layer activation)

if *max temp > max[layer]* **then**

max[layer] = max temp

end

if *min temp < min[layer]* **then**

min[layer] = min temp

end

end

end

while *layer in model layers* **do**

integer bits[layer] =

ceil(log2(maximum(absolute(min[layer]),absolute(max[layer]))))

fraction bits[layer] = 7-integer bits[layer]

end

output **:** 8b quantized fixed point model

3.4 Conversion and deployment of DL model

Two approaches were followed to convert and deploy the model to the ARM Cortex M7 board.

1. Using Tensorflow Lite for Microcontrollers framework
2. Using Edge Impulse platform

A comparison of the results obtained (memory footprint/latency) using either frameworks is consolidated in Table 3.4. The input size for the network was fixed at 98×40 for this experiment i.e. feature extractor stage computes MFCC with window size 30 ms, stride 10 ms and feature size 40.

Inference library	**Accuracy (%)**	**Latency**	**Memory footprint***	**Peak RAM usage****
TFLM	93.59	2575 ms	280 kB	91 kB
Edge Impulse with TFLM	93.9	7660 ms	301 kB	69 kB
Edge Impulse with CMSIS-NN	93.9	960 ms	320 kB	69 kB

TABLE 3.4: Comparison of results using Tensorflow Lite for Microcontrollers (TFLM) and Edge Impulse framework. Tested on STM32F769I-DISC1 board running at 216 MHz. *Memory footprint refers to the total memory occupied by the code - including NN weights, stored audio sample and code library. **Peak RAM usage refers to the maximum RAM used by the code at any time - includes NN weights and activations.

Chapter 4

End-to-end system

In this project we implemented an end-to-end deep learning based keyword spotting system on an ARM Cortex M7 based microcontroller development board. The details of the microcontroller board used in the project are given in Section 4.1. A detailed overview of the system is provided in the sections that follow.

4.1 Microcontroller platform

The microcontroller platform used for implementing the end-to-end deep learning based keyword spotting system is the STM32F769I-DISC1 discovery board from ST Microelectronics [19] (Figure. 4.1). The key features of the board are given below.

- STM32F769NIH6 microcontroller with Cortex M7 Core
- Maximum operating frequency of 216 MHz
- 2 MB Flash memory
- 512 kB RAM
- Floating Point Unit (FPU) is available

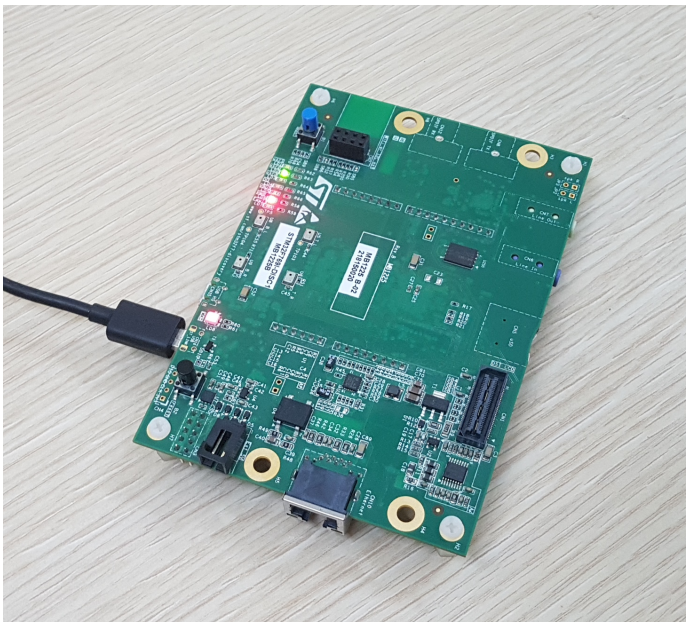


FIGURE 4.1: The STM32F769I-DISC1 discovery board from ST Microelectronics used for implementing the end-to-end deep learning based keyword spotting system.

The Cortex-M7 processor available on this board, is the highest performance member of the energy-efficient Cortex-M processor family from ARM. With reasonably high RAM (512 kB) and flash (2 MB), it is suitable for experimenting with different neural network architectures. Additionally the four digital MEMS microphones available on this board, makes it suitable for running the system on real-time audio input. The STMCubeIde from STMicroelectronics is used throughout the project for development purposes.

4.2 System overview

The system implemented in this project can be summarized as shown in Figure 4.2. The different stages of the system are explained in the following sections.

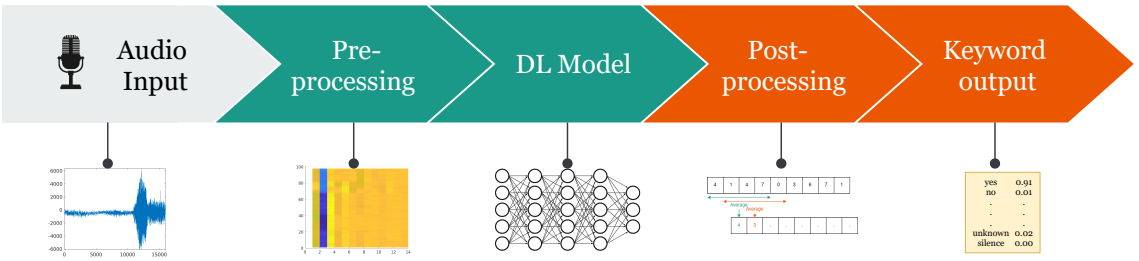


FIGURE 4.2: DL based KWS pipeline

4.3 Audio input

The input audio provided to the system was 16b Pulse Code Modulated (PCM) single-channel signal at 16 kHz frequency. This was chosen based on the specification of audio data used for training the various neural network models, from Google Speech Commands dataset [4]. The audio input for the system was provided either as audio samples stored in flash memory or as live audio input from the onboard digital MEMS microphone.

4.4 Data pre-processing

The data pre-processing stage computes the MFCCs from the input audio signal read from flash memory/microphone. For computation of MFCC features, an audio signal of one second duration was converted into shorter overlapping frames with window size (l) and stride (s) (Figure 2.2), and an N feature MFCC was computed for each frame.

Inorder to implement MFCC on the microcontroller, two approaches were explored.

1. Using CMSIS-DSP and math functions: CMSIS-DSP [20] is a DSP library provided by ARM with over 60 Functions optimized for the SIMD instruction set available in Cortex-M4/M7/M33/M35P processors.

2. Using KISS FFT library: KISS (Keep It Simple Stupid) FFT [21] is a small and simple FFT library without platform-specific optimizations.

Table 4.1 summarizes the time taken for computation of MFCC using the above approaches.

Library used	**Latency (ms)**
CMSIS-DSP	27
KISS FFT	182

TABLE 4.1: Comparison of time taken for computation of MFCC on the STM32F769I-DISC1 board running at 216 MHz. MFCC feature size was set to 10, window length to 40 ms, window stride to 20 ms and MFCC was evaluated for stored audio samples of duration 1 sec.

The steps involved in the computation of MFCC coefficients is described in Algorithm 2 and a split-up of the time taken for the various steps of MFCC computation (L=1 sec, window size = 40 ms, stride = 20 ms, number of coefficients = 13) using CMSIS-DSP library on an STM32F769NI DISC1 discovery board running at 216MHz is given in Table 4.2. As observed from the table, the computation of MFCC coefficients for a single time frame of 40 ms takes around 0.58 ms. In an audio sample of 1 sec duration, for the configuration with window length = 40 ms and stride = 20 ms, 49 such frames will be present. So, the computation of MFCC over a 1 sec audio sample, takes around 0.58 ms $\times$ 49 = 28.42 ms, as observed in Table 4.1. Also, from Table 4.2, it can be seen that the FFT computation step takes up a significant share of the computation time. Hence by decreasing the number of FFT points, the latency in MFCC computation can be further reduced. For the audio sample used, the 16 kHz sampling rate restricts the number of FFT points to 1024. By reducing the audio sampling rate to 8 kHz, which is suitable for speech signals, the number of FFT points can be reduced to 512. This approach is explored in Section 5.3.

Algorithm 2: Time taken for steps in computation of MFCC using CMSIS-DSP lilbrary

input : Audio sample

Frame the audio sample into shorter frames of window size l and stride s (Figure 2.2)

while *frame in frames* **do**

- Apply a Hanning window to the frame
- Compute N point FFT of windowed frame
- Convert FFT power spectrum
- Apply mel filterbanks on power spectrum
- Take log of mel filterbanks
- Compute DCT of log mel filterbanks

end

output: MFCC coefficients

Step	**Duration (ms)**
Computation of 512 point FFT	0.35
Conversion of FFT to power spectrum	0.09
Computation of log mel filterbanks from power spectrum	0.09
Computation of DCT of log mel filterbanks	0.05
Total time for one frame of length 40 ms	0.58

TABLE 4.2: Time taken for different steps in the computation of MFCC on an STM32F769I-DISC1 board running at 216 MHz using the CMSIS-DSP library. MFCC feature size was set to 13, window length to 40 ms, window stride to 20 ms.

Different configurations of MFCC, namely window size, stride, and number of MFCC features were explored and its impact on accuracy and latency of the system was analysed. The results are summarised in Table 4.3. Based on these experiments and the performance of the NN on the microcontroller board the final network was designed with the following specification for feature extractor stage.

- MFCC features: 13
- Window size: 40 ms
- Stride: 20 ms

MFCC Configuration				Accuracy (%) 32b floating point model	Accuracy (%) 8b fixed point model	Total system latency on MCU (in ms)
Window size (ms)	Stride (ms)	MFCC features	NN input size			
40	40	10	25×10	87.4	83.8	107
40	20	10	49×10	89.0	87.0	200
40	20	13	49×13	86.2	83.3	273
30	10	40	98×40	93.5	87.0	960

TABLE 4.3: Comparison of impact of MFCC configuration on system performance. Tested on STM32F769I-DISC1 board running at 216 MHz.

4.5 DL model implementation

The deep learning model used for the neural network stage is the res8-narrow model [2], shown in Figure 3.2. Inorder to implement the res8-narrow model on the Cortex M7 microcontroller board, the following approaches were explored as described in Section 3.4.

- Using CMSIS-NN library
- Using Tensorflow Lite for Microcontrollers framework
- Using Edge Impulse platform

The final end-to-end system implementation uses the Edge Impulse framework with CMSIS-NN library due to the better inference time (system latency).

4.6 Post-processing

As the system was processing the audio input sequentially, post processing unit is excluded.

Chapter 5

Results

5.1 Results on stored audio samples

Table 5.1 consolidates the results obtained by evaluating the NN models (DNN (Figure 5.2), DS-CNN (Figure 5.3), res8-narrow (Figure 3.2)) on audio samples stored in flash memory. All the results are based on running the corresponding NN models on the STM32F769I-DISC1 (Section 4.1) development board at 216 MHz system clock frequency.

Test dataset details: The test dataset for the Google Speech Commands dataset [4] has a total of around 6000 audio samples of 1 sec duration, and each file occupies 32 kB on the microcontroller which makes the test set size approximately 200 MB. The flash memory of the microcontroller is limited to 2 MB, which makes it difficult to run the code on the entire test dataset. Considering these limitations a small subset of the test dataset was used for obtaining an estimate of the accuracy of various models on the microcontroller board. The subset included 50 randomly chosen samples for each keyword from the test dataset.

NN model	**Floating point model accuracy** (%)	**Fixed point (8b) model accuracy** (%)	**Accuracy observed (saved audio)** (%)	**Latency per inference (in ms)**	**Memory footprint (in kB)**	**Peak RAM usage (in kB)**
DNN (Input size: 25x10)	-	84.6	80.2	15	380	159
DS-CNN (Input size: 49×10)	-	94.4	84	100	329	159
res8-narrow (Input size: 49×10)	89.0	87.0	72.5	200	319	69
res8-narrow (Input size: 49×13)	86.2	83.3	82.0	273	320	74

TABLE 5.1: Comparison of the results obtained by evaluating the NN models on stored audio samples on the STM32F769I-DISC1 board running at 216 MHz system clock frequency.

5.2 Results on live audio samples

Table 5.2 consolidates the results obtained by testing the end-to-end system on live audio input by running the res8-narrow NN model on the STM32F769I-DISC1 (Section 4.1) development board at 216 MHz system clock frequency. For testing, 100 random samples per class from the Google Speech Commands test dataset [4] was played aloud, and the inference was analysed. A keyword-wise breakup of the accuracy is provided in Table 5.3.

Parameter	**Value**
Accuracy	69.9 %
Latency for inference	91 ms
Latency for feature extraction	182 ms
Memory footprint	320 kB
Peak RAM usage	74 kB

TABLE 5.2: Results obtained on live audio input on the res8-narrow based end-to-end KWS system implemented on the STM32F769I-DISC1 board running at 216 MHz system clock frequency.

Keyword	**Accuracy (in %)**
yes	79.00
no	69.00
right	70.00
left	70.00
go	43.00
stop	88.00
up	56.00
down	56.00
on	84.00
off	63.00
unknown	61.00
silence	90.00

TABLE 5.3: Keyword wise accuracy obtained on live audio input on the res8-narrow based end-to-end KWS system implemented on the STM32F769I-DISC1 board running at 216 MHz system clock frequency.

The results obtained using the live audio stream are not on par with those obtained using the stored audio samples. A sample audio input captured using the MEMS digital microphone is given in Figure 5.1.

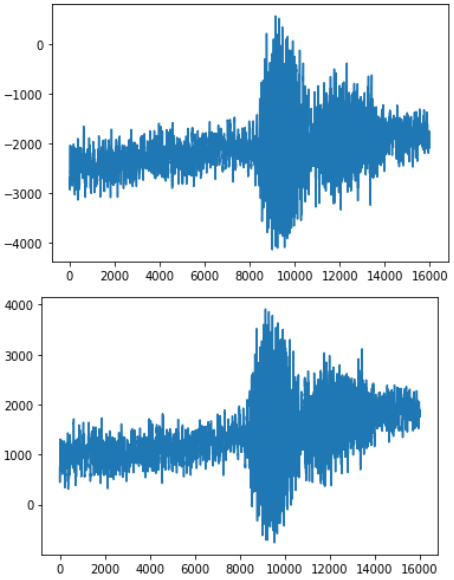


FIGURE 5.1: Audio input waveform captured by the top left (top graph) and top right (bottom graph) MEMS digital microphones on the STM32f769I-DISC1 discovery board

5.3 Results on reduced sampling rate

Inorder to reduce the latency of the feature extractor layer, audio samples sampled at 8 kHz frequency was considered instead of 16 kHz. To do so the res8-narrow model was retrained on a dataset with 8 kHz audio samples (generated from 16 kHz audio samples of Google speech commands dataset). The results obtained is shown in Table 5.4.

As observed from Table 4.2, a significant amount of computation time of MFCC comes from the FFT computation. For a 16 kHz input audio signal, a 40 ms frame would contain $40 \times 16 = 640$ samples, making the FFT length 1024. Whereas for an 8 kHz audio signal, a 40 ms frame would contain $40 \times 8 = 320$ samples, which requires an FFT length of 512 only. This can significantly reduce the time spent in feature extraction and is observed in Tables 5.2 and 5.4. While the 16 kHz audio samples of 1 sec duration required 182 ms for feature extraction, the 8 kHz audio samples required only 102 ms. As speech signals can be represented without much data loss using 8 kHz sampling rate, the overall system accuracy is not significantly

impacted. The library used for the implementation of MFCC feature extraction here is the KISS FFT library.

Parameter	Value
Accuracy (32b floating point model)	81.6 %
Accuracy (8b fixed point model)	79.8 %
Latency for inference	91 ms
Latency for feature extraction	102 ms
Memory footprint	259 kB
Peak RAM usage	43 kB

TABLE 5.4: Results obtained on training the res8-narrow model on 8 kHz audio samples and implementing it on the STM32F769I-DISC1 board running at 216 MHz system clock frequency. MFCC parameters used are window size = 40 ms, stride = 20 ms, number of coefficients = 13.

5.4 Models used for results comparison

The DNN and DS-CNN based neural networks [1] used for comparison of results, is shown in Figures 5.2 and 5.3 respectively.

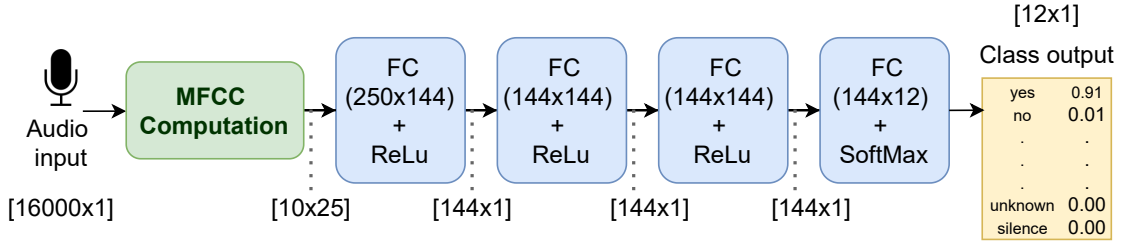


FIGURE 5.2: DNN model used for results comparison [1]

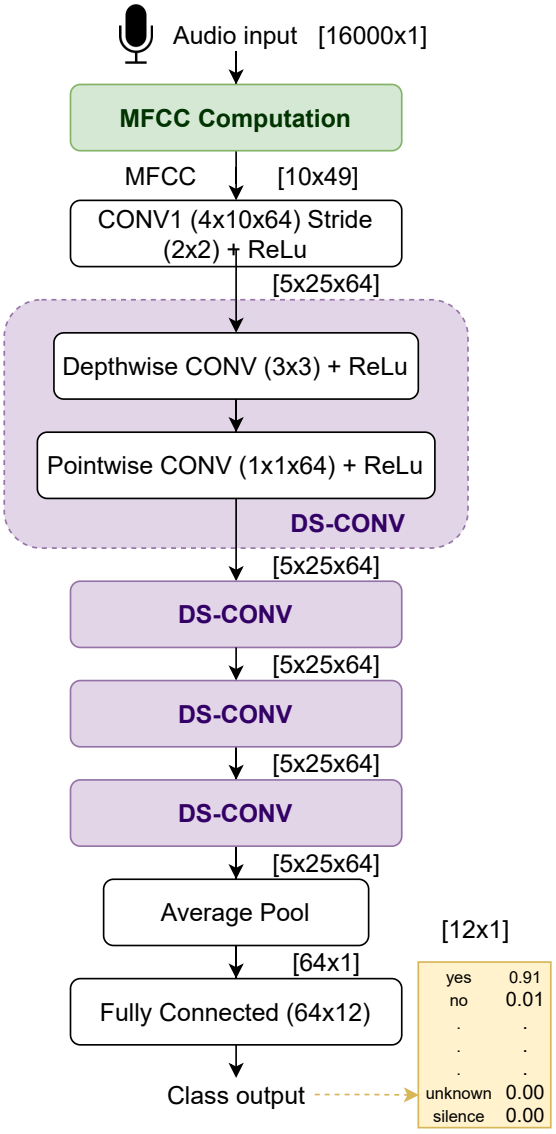


FIGURE 5.3: DS-CNN model used for results comparison [1]

Chapter 6

Conclusion and future scope

An end-to-end deep learning-based keyword spotting system was implemented on the STM32F769I-DISC1 development board. Though the system demonstrated decent accuracy while evaluating stored audio samples, the response to the live audio stream was not up to the mark. This could be due to the variation in the audio profile of the signal captured by the MEMS microphone on the development board as compared to the samples from the original dataset. Further analysis and modifications need to be done to make the system robust to such variations.

As a future scope, the area of neural architecture search based neural network design and co-design of the neural network and inference engine (for microcontroller) [22] can be explored, in order to extend the system to more keywords and make it more robust.

References

[1] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint arXiv:1711.07128*, 2017.

[2] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2018, pp. 5484–5488.

[3] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The microsoft 2017 conversational speech recognition system," in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, IEEE, 2018, pp. 5934–5938.

[4] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[5] T. Kim, J. Lee, and J. Nam, "Comparison and analysis of samplecnn architectures for audio classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 285–297, 2019.

[6] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2014, pp. 4087–4091.

[7] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[8] S. O. Arik, M. Kliegl, R. Child, J. Hestness, A. Gibiansky, C. Fougner, R. Prenger, and A. Coates, "Convolutional recurrent neural networks for small-footprint keyword spotting," *arXiv preprint arXiv:1703.05390*, 2017.

[9] S. Team, "Hey siri: An on-device dnn-powered voice trigger for apple's personal assistant," *Apple Machine Learning Journal*, vol. 1, no. 6, 2017.

[10] 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," *arXiv preprint arXiv:1704.04861*, 2017.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[12] I. McGraw, R. Prabhavalkar, R. Alvarez, M. G. Arenas, K. Rao, D. Rybach, O. Alsharif, H. Sak, A. Gruenstein, F. Beaufays, *et al.*, "Personalized speech recognition on mobile devices," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2016, pp. 5955–5959.

[13] C Teacher, H Kellett, and L Focht, "Experimental, limited vocabulary, speech recognizer," *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 3, pp. 127–130, 1967.

[14] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.

[15] *TensorFlow Lite for Microcontrollers*, `https://www.tensorflow.org/lite/microcontrollers`.

[16] *Edge Impulse*, `https://docs.edgeimpulse.com/docs`.

[17] *Pytorch code for training the Deep Residual Network for KWS*, `https://github.com/castorini/honk`.

[18] *Pytorch code for training the Deep Residual Network for KWS with additional noise*, `https://github.com/archiki/KeyWordSpotting-Models`.

[19] *STM32F769I-DISC1 Development Board specification document*, https://www.st.com/resource/en/user_manual/dm00276557-discovery-kit-with-stm32f769ni-mcu-stmicroelectronics.pdf.

[20] *CMSIS-DSP Library*, https://www.keil.com/pack/doc/CMSIS/DSP/html/index.html.

[21] *KISS FFT Library*, https://directory.fsf.org/wiki/Kiss_FFT.

[22] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," *arXiv preprint arXiv:2007.10319*, 2020.