ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Integrierte Systeme
Integrated Systems Laboratory

DEPARTMENT OF
INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Spring Semester 2022

Feature Extraction for Speech Recognition

Semester Project

Niels Escarfail
nescarfail@student.ethz.ch

18 July 2022

Advisors: Cristian Cioflan, cioflanc@iis.ee.ethz.ch
Lukas Cavigelli, lukas.cavigelli@huawei.com

Professor: Prof. Dr. L. Benini, lbenini@iis.ee.ethz.ch

Abstract

Most modern solutions to Automated Speech Recognition (ASR) problems now revolve around Deep Neural Networks (DNNs). Particularly, Convolutional Neural Networks (CNNs) have shown excellent proficiency in solving ASR tasks in a resource-preserving manner.

In any machine learning problem, a vital step constitutes feature extraction. Feature extraction's purpose is to reduce the signal size for subsequent processing all-the-while preserving and putting forward the most relevant information.

In the specific ASR problem of Keyword Spotting (KWS), for which the applications require low-power usage and high accuracy, CNNs are now prevalent. Furthermore, with these resource constraints, it has become increasingly important to optimise the entirety of the KWS pipeline, from feature extraction to subsequent model processing; however, little research exists on the subject.

This project aims to evaluate feature extraction methods for speech recognition and, in particular, for the task of Keyword Spotting.

Evaluating feature extraction methods for subsequent model analysis is not a trivial task. Indeed, each neural network architecture might be more or less suited to a feature extraction method. Moreover, a particular model not achieving high accuracy with a specific input does not mean the input is not correct or does not hold information.

To overcome the challenge of finding suitable neural network architecture configurations for each feature extraction configuration and attempt to reduce this model bias, we use a Neural Architecture Search alternative, the Once-For-All (OFA) [1] framework. Our implementation consists of a novel macro-model architecture comprised of Residual Depthwise Separable Convolutions and Inverted Residual Bottlenecks. Our OFAKWS network has an elastic neural architecture enabling training through progressive shrinking and, therefore, the training of all the networks in a configured neural architecture search space.

After training the OFA network and, therefore, the entire neural architecture space for different feature extraction methods and parameters, we evaluate the feature extraction methods by randomly sampling and evaluating sub-network architectures. Our findings show the popular Log Mel Spectrogram (LMS) and Mel Frequency Cepstrum Coefficients (MFCC) features achieve the best results. Finally, we evaluate the impact of parametrising the feature extraction methods on the resulting sub-network accuracy. Doing so gives us us insight into the properties of each feature extraction parameter, for instance, the number of Mel filterbanks used to generate a Mel Spectrogram and how we might want to modify it depending on our deployment constraints. Specifically, we bring insight into the impact of the hop length, window length, number of Mel filterbanks, and MFCC bins used when generating MFCCs or intermediary spectrograms in the MFCC pipeline.

Our approach achieves 95.37% and 94.44% accuracy for the networks using LMS and MFCC features respectively. More interestingly, our framework allows us to specialise these features for the low power setting. We find that in our case, in the lowest-memory context (less than 55 KB of memory), the MFCC feature extraction method achieves better results than the LMS. However, if we intend to function in the higher-memory regime (more than 55KB of memory), then the LMS is desirable.

This report is the first work approaching feature extraction evaluation using the OFA framework. The resulting framework allows us to find the best-suitable feature extraction method and its parameters, given the desired number of operations, accuracy, memory, or any other constraint. It can also be used for the efficient development of neural network architectures within a constrained environment. This method can also be applied to any task and domain, unrestricted to ASR.

Acknowledgments

I would like to express my most resounding gratitude and thanks to my advisors, Cristian Cioflan and Lukas Cavigelli, for their help and guidance throughout the project, no matter the challenges, from solving incomprehensible bugs to unlocking new ideas to advance forward. I would also like to thank Prof. Dr. L. Benini and the Integrated Systems Laboratory for hosting this project.

Furthermore, the base for this work is the Once For All framework. I am thankful to the authors, Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang and Song Han, for sharing their work and enabling this solution.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

FEATURE EXTRACTION FOR SPEECH RECOGNITION

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
ESCARFAIL	NIELS

With my signature I confirm that

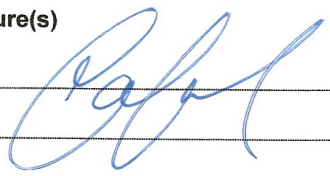
- I have committed none of the forms of plagiarism described in the ' ' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Zürich, 17th of July 2022

Signature(s)



For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Contents

Chapter 1

Introduction

Speech is presumably the most intuitive and natural form of human communication and interaction. Humans favour speech interfaces over haptic interfaces like touchscreens or keyboards when interacting with machines, not forgetting the improved usability which follows. Human-computer interaction consequently relies more and more on speech recognition to enable further applications and a more satisfactory user experience.

Speech Recognition, also known as ASR, designates the subfield of computational technologies enabling the recognition of spoken language into text by a computer. This technology enables natural language understanding (NLU) approaches to later process the word sequences created by ASR algorithms and enable various applications. Speech recognition technology nowadays is found in a wide variety of systems, from voice-activated assistants on a smartphone to voice-instructed industrial computers, and has significantly improved our working and living environment.

Specific devices and applications, for instance, "Wake words" in smart devices, narrow the speech recognition task to keyword spotting. Keyword Spotting (KWS) is a specific problem in speech recognition in which one has to identify words belonging to a finite predefined set, in utterances.

KWS applications range from wake words to emergency word detection; they typically have an always-on nature and require high accuracy for a satisfactory user experience. Due to this wide variety in potential applications and deployment platforms ranging from micro-controllers to industrial computers, KWS deployment pipelines must be tailored to specific constraints (low-power, memory and storage requirements) while achieving satisfactory accuracy.

In the past, a solution to platform-specific resource constraints was to off-load the compute workload to cloud services. However, that implies data-transmission energy costs, leading to short battery lifetimes. Furthermore, this makes the KWS pipeline dependent on the communication infrastructure and subject to high latency. Finally, one

always has to worry about privacy issues when communicating information. For all these reasons, it is necessary to be able to control and evaluate each step of the KWS pipeline, from feature extraction to neural network processing.

Feature extraction is the first step in the keyword spotting pipeline. Its purpose is to modify the speech waveform to a parametric representation while reducing the signal size for subsequent processing and analysis and retaining or putting forward relevant information. It is a prominent part of any machine learning solution, and the choice of feature extraction method determines the optimal neural network model architecture one might use to solve a problem.

However, little to no literature exists on processes to find the best feature extraction methods and configurations for a given task, and most of the existing research in machine learning has to go through a trial and error phase in order to correctly choose the suitable feature extraction method and configurations for a specific task and model. Comparing feature extraction methods is difficult as the resulting performance will depend highly on the neural network architecture. Trying to train a vast network which works on multiple feature extraction methods renders memory evaluation infeasible, even if it achieves high accuracy.

State-of-the-art methods [2] [3] for low-power KWS traditionally use Mel Frequencies Cepstrum Coefficients (MFCC) or its intermediary step, the Log Mel Spectrogram, as a feature extraction method. However, there is a missing point in the literature comparing the trade-offs between the two feature extraction methods, and therefore feature extraction is often left as another hyper-parameter to optimise, considered secondary to the neural network architecture.

To further advance state-of-the-art work in keyword spotting, it is necessary to understand the trade-offs of feature extraction methods and their configurations. Furthermore, our approach also allows tackling this task by simultaneously optimising both feature extraction methods and neural network architectures, as it often shows to achieve state-of-the-art or better results in multiple problems.

This work proposes a solution to the problem of comparison and evaluation of feature extraction methods via Neural Architecture Search (NAS) using the Once-For-All (OFA) framework in the task of keyword spotting. We analyse the optimal feature extraction parameters for KWS using this framework. Notably, this method applies to more complex problems unrestricted to the field of speech recognition.

Using Once-For-All allows us to lower the bias and surpass the difficulties related to neural networks input specificity when comparing feature extraction methods. In the specific task of keyword spotting, information and methods highlighted in this study will allow further optimisation and specificity when developing and deploying KWS modules.

Chapter 2 offers an overview of the KWS pipeline and some commonly used feature extraction methods for keyword spotting. We then outline state-of-the-art work in KWS

in Chapter 3, focusing on small footprint KWS. Chapter 4 goes through the methods used to perform feature extraction evaluation and its implementation using Once-For-All. Finally, Chapter 5 visualises and summarises our evaluation setup and findings. We summarise our work in Chapter 6 - Conclusion and Future Work.

Chapter 2

Background

Section 2.2 first introduces state-of-the-art feature extraction methods for speech processing, in particular for the task of keyword spotting. We present the Spectrogram, the Mel-Spectrogram and the Mel-Frequency Cepstrum Coefficients.
We later introduce in Section 2.3 known proficient neural network architectures in Keyword Spotting. We present Depthwise Separable Convolutions and Inverted Residual Bottlenecks. Finally, we introduce the concept of Neural Architecture Search, as it is a chosen methodology in this work.

2.1. Keyword Spotting pipeline

In order to efficiently perform keyword spotting using neural networks, most keyword spotting pipelines involve two steps. The first step is feature extraction, Section 2.2 shows examples of various transformations for audio feature extraction. Secondly after extracting relevant features and, most of the time, reducing the size of the input, we pass these features to a neural network which then predicts the keywords. Fig. 2.1 shows an overview of the KWS pipeline which we will consider in this project.

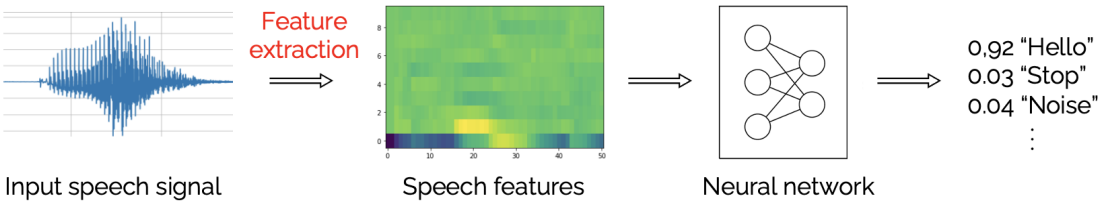


Figure 2.1.: The Keyword Spotting pipeline

Therefore, feature extraction is a vital constituent of the KWS pipeline, however there are many different ways to transform speech signals for feature extraction, the ones relevant to our context are described in Section 2.2 below.

2.2. Feature Extraction methods for Speech Recognition

Feature extraction in our context consists of modifying the speech waveform to a parametric representation while reducing the signal size for subsequent processing and analysis. High-quality features are necessary to obtain satisfactory classification accuracy and latency with any subsequent model.

There are multiple methods to extract features from a speech signal. In the context of Keyword Spotting, the most commonly used feature extraction methods are the Mel-Frequency Cepstrum Coefficients (MFCC) and its intermediate steps, shown in Fig. 2.2 and further described in the subsections below.

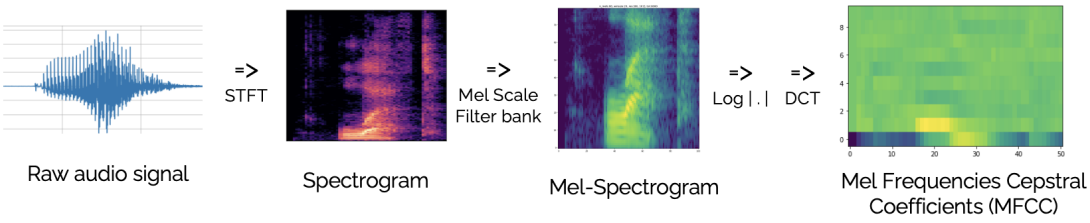


Figure 2.2.: MFCC computation steps

2.2.1. Spectrogram

A spectrogram is a visual representation of the frequency spectrum of a signal as it varies with time, corresponding to a time-frequency representation of the signal. A spectrogram, therefore, displays changes in the frequencies in a signal over time. The spectrogram's third dimension, with variable colour, represents the amplitude. Spectrograms are frequently used for speech recognition and a variety of other applications such as seismology.

One way of computing the spectrogram of a signal is using the Fourier transform. Indeed, you can obtain the spectrogram of a signal by computing the squared magnitude of the signal's short-time Fourier transform (STFT) (Eq. (2.1)). For a given window width w and hop length h, representing the distance between windows, we usually have $h \leq \frac{w}{2}$ to obtain overlapping windows. [1]

[1]The logic behind these overlapping window is further discussed in Section 5.3.1 later in this report.

$$Spectrogram(w,h) = |STFT(w,h)|^2 \tag{2.1}$$

Namely, time-domain data is broken up into (usually overlapping) windows. Taking the magnitude of each window's Fast Fourier Transform (FFT) gives the frequency spectrum's magnitude for each window. Each window thus corresponds to a vertical line in the image, a measurement of magnitude versus frequency for a specific moment in time, the window's middle point. Concatenating these spectrum form the Spectrogram.

2.2.2. Mel Spectrogram

The Mel scale

The Mel Scale is a logarithmic transformation of the frequency scale. This transformation is constructed such that sounds of equal distance in human pitch are also of equal distance to each other on the Mel Scale.

The Mel Spectrogram

As its name indicates, the Mel Spectrogram is just a Spectrogram with a Mel Scale as its y-axis. The Mel Spectrogram and the Log Mel Spectrogram have proven proficiency in speech recognition and in the task of Keyword Spotting, and work using such features has achieved state-of-the-art [4][2], with the Log Mel Spectrogram being more popular as it replicates both human perception of loudness (thanks to the logarithm) and pitch (thanks to the Mel scale).

2.2.3. Mel-Frequency Cepstrum Coefficients

The original concept of the Mel frequency Cepstrum Coefficients (MFCC) transformation is to model or replicate the human hearing system, intending to model the ear's working principle [5].

The MFCC of an audio signal is obtained by computing the Discrete Cosine Transform (DCT) of the logarithm of the Mel Spectrogram of the original signal. The MFCC transformation can be particularly beneficial as it drastically reduces the input size relative to the Mel-Spectrogram while attaining state-of-the-art results in "simple" speech recognition tasks such as Keyword Spotting.

To summarize the MFCC transformation intermediate steps [6] which we will be dealing with throughout this report, let us describe the MFCC computation steps described in Fig. 2.2:

1. Divide the audio signal into equally-sized, usually overlapping short frames.
2. For each frame, compute the short-time Fourier transform (STFT) of the input.
3. Square the coefficients in order to obtain a power spectrum, the Spectrogram Section 2.2.1.
4. Convert the resulting Spectrogram to the Mel-scale, by projecting the spectrum on triangular or Mel filters. In the end, the Mel Spectrogram Section 2.2.2 we obtain has the same number of features as the number of Mel filters. This converts the Spectrogram to match the human pitch perception.
5. Take the logarithm of the newly obtained Mel Spectrogram, effectively now taking into account the human perception of loudness by effectively converting to the decibel scale.
6. Compute the Discrete Cosine Transform (DCT) of the previously obtained Log Mel Spectrogram to obtain the Mel Frequency Cepstrum Coefficients (MFCC), for which we keep an arbitrary number of features, the number of MFCC bins.

2.3. Convolutional Neural Networks in Keyword Spotting

A Convolutional Neural Network (CNN) is a type of Artificial Neural Network most commonly applied to visual imagery analysis, as convolutional layers allow them to capture spatio-temporal information of the input. A convolutional layer uses filters or kernels that performs convolution operations at it is scanning the input with respect to its dimensions.

Convolutions have three main hyper-parameters:

1. the kernel or filter size K;
2. the stride S, denoting the number of pixels by which the window moves after each operation;
3. the padding P, representing the process of adding P zeros to each sides of the input boundary.

Importantly, a kernel of size (K $\times$ K) applied to an input containing C channels is a (K $\times$ K $\times$ C) volume that performs convolutions on an input of size (I $\times$ I $\times$ C) and produces an output of size (O $\times$ O $\times$ I), where $O = \frac{I-K+P\_start+P\_end}{S} + 1$ The resulting output is called a feature map.

While CNNs initially have become popular for their performance and efficiency in solving image recognition tasks, they have achieved impressive performance in automated speech recognition in the later years, and we particularly describe below some known efficient architectures or blocks for the task of Keyword Spotting.

We will choose an example to illustrate a Convolution which we will later use to explain the Depthwise Separable Convolutions. For a RGB input image of 12 $\times$ 12 pixels, giving us a total input shape of 12 $\times$ 12 $\times$ 3. We can perform a 5 $\times$ 5 convolution with no padding and a stride of 1. Considering the height and width of the image, we will end up with an image with 12 $\times$ 12 - 5 $\times$ 5 = 8 $\times$ 8 dimensions. Performing the 5$\times$5 convolution, we undergo 25 multiplications for each channel and we end up with an output shape of $12 - 5 + 1 = 8$. Noticeable a normal convolution undergoes 5$\times$5$\times$3=75 convolutions every time the kernel moves. Therefore, a 12 $\times$ 12 $\times$ 3 image will become a 8 $\times$ 8 $\times$ 1 image after going through a 5x5x3 kernel. If we want to increase the number of channels in our output image, we can simply create N kernels in order to create N 8 $\times$ 8 $\times$ 1 images, then stacking them in order to obtain a 8 $\times$ 8 $\times$ N output.

2.3.1. Depthwise Separable Convolutional Neural Networks

Depthwise separable convolutions factorise a convolution into two steps. First, we perform a depthwise convolution, which is essentially a spatial feature learning stage. Secondly, we perform a pointwise convolution, a channel-combination stage. The depthwise separable convolutions therefore not only deals with the spatial dimension but also with the depth dimension, the number of channels. For instance, images may be constituted of 3 RGB channels; as the input goes through convolutions, this number of channels can also increase.

Depthwise Convolution

The first stage is the depthwise convolution, in which we give the input image a convolution while not changing the depth (number of channels), therefore for instance, on a 3 channel input we can do so using 3 kernels of shape (5$\times$5$\times$1). As each (5$\times$5$\times$1) kernel will iterate over a single channel of the image, gathering the scalar products of every 25 pixel region and therefore outputting an (8$\times$8$\times$1) image. We stack the 3 obtained images in order to obtain a (8$\times$8$\times$3) image.

Remembering the previous example for the regular convolution, we can see the next step's role is to increase the number of channels of the output in order to match a regular convolution.

Pointwise Convolution

The second stage is the pointwise convolution, the channel-combination stage.

After performing the depthwise convolution, we end up with a 8$\times$8$\times$3 image. As its name indicates, the pointwise convolution uses a 1x1 kernel, which therefore iterates through every single point. The depth of this kernel is equal to the number of channels

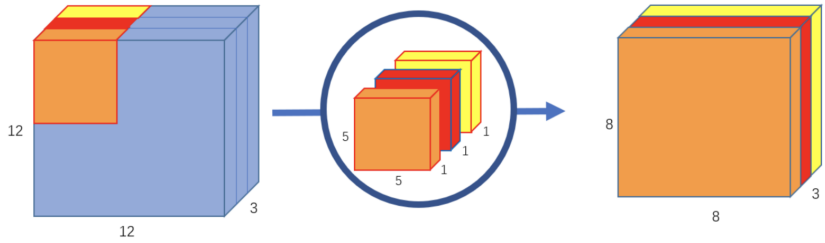


Figure 2.3.: Depthwise convolution example, figure reprinted from [7]

of the input images, therefore 3 in our case. Passing the 1×1×3 kernel through our 8×8×3 image yields a 8×8×1 images. Therefore, creating N 1×1×3 kernels, we can concatenate the outputs to form a final image of shape 8×8×N.

Existing work [8] often mentions Depthwise Separable Convolutional Neural Networks (DS-CNN) as providing the best classification accuracy trade-off for memory footprint and computational resources.

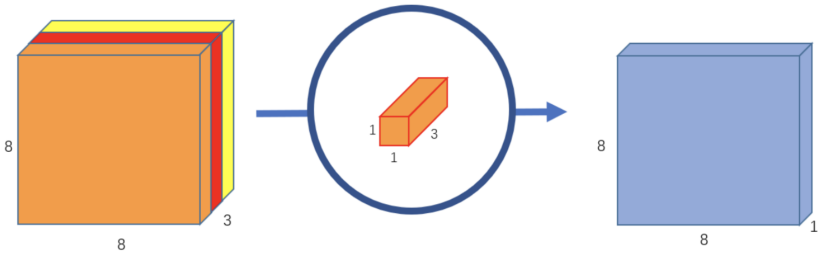


Figure 2.4.: Pointwise convolution example, figure reprinted from [7]

2.3.2. Inverted Residual Bottlenecks

Introduced and popularized by the MobileNetV2 architecture [9], inverted residual blocks are inverted bottleneck layers which usese its first convolution to expand expand instead of reducing features. Therefore, instead of going from wide to narrow, then from narrow back to wide, as in normal bottleneck blocks, we perform the opposite narrow to wide, then wide to narrow transformations.

Going further with inverted residual blocks, we can perform them using depthwise convolutions are they are fit for the task and require less computations. These inverted residual bottlenecks blocks are now also named MBConvs after the popular MobileNet [9] architecture with which they were introduced.

Residual connections in general have been shown to further help neural networks gain accuracy with respect to resources in many tasks, and it is also the case for speech recognition

2.4. Neural Architecture Search

Neural Architecture Search (NAS) is a set of techniques aimed at automatically discovering model architectures of artificial neural networks. Several NAS methods succeed in obtaining architectures that meet or exceed the performance of manually created models. It is relevant to mention that NAS processes can infer various neural network types, ranging from CNNs to autoencoders. When performing NAS, one has to specify a search space representing the set of candidate architectures and the operations that compose them. NAS has already been a solution to finding efficient neural network architectures in terms of power usage and accuracy, with [10] referencing architectures for speech recognition for instance.

Chapter 3

Related Work

This section first introduces state-of-the-art neural network methods for Keyword Spotting. We then present Once-for-All, a Neural Architecture Search framework which serves as a foundation for this work.

3.1. Keyword Spotting

The rapid development of mobile and other voice-interacting devices has made speech-related technologies increasingly researched and popular. Running on mobile devices requires low computational power and a small memory footprint (typically less than 128KB of memory, but can often go lower to less than 30KB of memory for instance).

Keyword Spotting model architecture

Therefore, to obtain a KWS system able to run on constrained devices efficiently, Convolutional Neural Networks (CNN) have become the best option. Spectral speech representations have substantial time and frequency correlations. CNNs can extract an image's local spatial features and are presumably better suited to model them. Secondly, it has also been shown distinct speaking styles cause a frequency shift, and CNNs can overcome this via translational invariance, meaning they work very well detecting a shape independently of where it is in an image. They also offer other advantages such as easily adjustable model sizes and are complemented with many advances given their popularity. Finally, they have shown proficiency in many speech processing tasks, one of which is KWS.

Hello Edge [8] compares neural network architectures for KWS applications on microcontrollers with constraints, minding accuracy and memory/compute requirements.

They notably show that the Depthwise Separable Convolutional Neural Network (DS-CNN) architecture achieves an accuracy of 95.4%, which is about 10% higher than a DNN model with a similar number of parameters. Other works have then further improved these results by using temporal convolutions [11, 12], self-attention [3, 13], or binary neural networks [14, 15].

Another addition to the neural network architecture which has proven efficient in Keyword Spotting is residual learning. Deep Residual Learning for Small-Footprint Keyword Spotting [16] first showed in 2018 a proficient architecture involving a succession of residual blocks, made up of a series of batch normalisation and 3x3 convolutions. Finally, Broadcasted Residual Learning for Efficient Keyword Spotting [2] achieve state-of-the-art 98% accuracy with a small amount of parameters and less MACs than other architectures.

While all these architectures shine in their own regards, our solution to evaluate feature extraction methods will involve Neural Architecture Search, and we therefore opt to leverage the simplicity and efficiency of the DS-CNN architecture, while also taking benefit of residual learning with residual connections.

Keyword Spotting Feature Extraction Methods

State-of-the-art results for keyword spotting have been achieved using MFCC features [11], Log-Mel-Spectrogram features [2], and Mel-Spectrogram features [4, 3]. To a large extent, KWS systems' energy usage also comes from feature extraction, therefore, significant work is done nowadays to not only optimise the model architecture, but also feature extraction methods. This comes in various forms, with some research trying to remove completely feature extraction with end-to-end neural models [17]. Recently, work using other spectrogram alternatives with binarised features for the use with binary neural networks have also shown promising accuracy to memory efficiency [15]. Notably, recent work has even gone further and achieved state-of-the-art accuracy and energy efficiency by using analog binary feature extraction with binary neural networks [14](replacing the digital pre-processing with an analog front-end). With more and more promising feature extraction methods for KWS, it is relevant to find an efficient way to evaluate them.

3.2. Neural Architecture Search with Once-For-All

Once-For-All (OFA) is a Neural Architecture Search (NAS) solution which enables the training and deployment of multiple neural network architecture configurations simultaneously. Therefore, we select only part of the once-for-all network for inference. The resulting network flexibly supports different depths, widths, kernel sizes and resolutions. However, the neural architecture search space is too large to be computationally

tractable, for instance, taking a macro architecture of four macro blocks which all have 10 resolutions, 3 kernel possibilities, 4 depths, and 2 different widths, we roughly have more than $10 \times (3^4 \times 4 \times 2^4)^4 > 10^{15}$ possible sub-networks. Therefore, training the network requires a novel method called progressive shrinking.

3.2.1. Progressive Shrinking

Progressive shrinking is the training algorithm which constitutes the heart of once-for-all. The progressive shrinking consists of training the largest neural network with maximum depth, width (channels), and kernel size, for every resolutions, and then progressively fine-tuning the once-for-all network to support smaller sub-networks that share the weights of larger sub-networks.

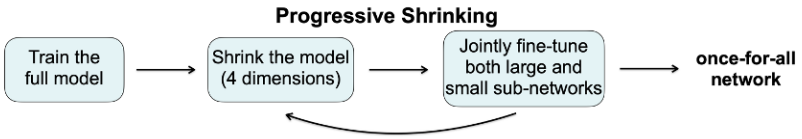


Figure 3.1.: OFA network training: Progressive Shrinking, figure by the original OFA paper [1]

3.2.2. Once-For-All for efficient development with constraints

Once-For-All therefore allows us to solve two problems at once. First we do not have to worry about whether our given model is fit for a particular feature extraction method configuration, as we perform neural architecture search. Furthermore, evaluating Once-For-All sub-networks gives an efficient overview of the chosen input, as it gives detailed overview of the impact of feature extraction methods on any evaluation metrics such as the accuracy or the number of operations by the model.

Chapter 4

Methods & Implementation

The main challenge in feature extraction evaluation for the later treatment by neural networks is that each neural network architecture is more or less suited for a given (transformed) input.

This section documents the adopted methodology to compare feature extraction methods for KWS. Specifically, we describe how OFA is used in our context to overcome the challenge of finding suitable neural network architectures for different feature extraction configurations in Section 4.1.

We notably describe the neural architecture search space in Section 4.2, namely, our OFA network macro architecture and its varying architecture parameters. We follow by describing the feature extraction methods and parameters search space analysed with our framework. Finally, we present the steps used to train the model following the progressive shrinking approach in Section 4.3.4. Giving us in the end a feature extraction specialised sub-network evaluation dataset, which allows us to analyse various feature extraction methods. The final section Section 4.4 goes through the PyTorch implementation of the OFAKWS framework.

4.1. Leveraging Once-for-all to enable feature extraction evaluation

As our goal is to evaluate feature extraction methods for keyword spotting, the framework on which to perform neural architecture search needs to be capable of receiving and functioning with inputs of various forms. Furthermore, in this specific task, we wanted to be able to view similar network configuration performances with different feature extraction methods.

OFA's elastic resolution allows us to maintain the same macro-architecture throughout the different runs corresponding to different feature extraction types. Therefore, our approach is to replace the flexible resolution from the original Progressive Shrinking [1] algorithm to transition from elastic image sizes to elastic feature extraction configurations.

Another benefit is OFA's simultaneous optimization of the whole sub-network architecture space, allowing us to train substantially more network configurations than other NAS methods while using fewer resources. To add to the choice of OFA as a framework for this evaluation, its capability to efficiently extract a given sub-network under specialized constraints makes it even more attractive for future research and on-device evaluation.

Therefore, using an OFA network allows us to train various model architectures for multiple feature extraction parameters simultaneously. Once we have trained the OFA network, we can then extract and evaluate sub-networks to understand the ability of each feature extraction method to convey relevant information to resulting models while reducing the size of the input.

4.2. Once-for-All for Neural Architecture Search

In order to circumvent the challenge of adapting a neural network architecture for each feature extraction configuration, we perform neural architecture search with once-for-all.

4.2.1. OFAKWSNet macro architecture

As the array of KWS neural network architectures is wide, ranging from Transformers to CNN variants, we decide to focus on promising low-power and low-memory CNN architectures for our research. Researching existing literature for small-footprint KWS shows the promises of Depthwise Separable Convolutional Neural Networks (DS-CNN) with work such as [8]; furthermore, analysing the NAS-ASR benchmark [10] referencing 8,242 ASR models shows the prevalence of residual learning, with multiple other writings [2], [16] to cite a few, utilising residual connections to achieve high accuracy in constrained environments. For these reasons, we begin by selecting a macro-architecture for the once-for-all network which revolves around depthwise separable convolutions and residual connections, as we have in mind achieving high accuracy while requiring low computational resources.

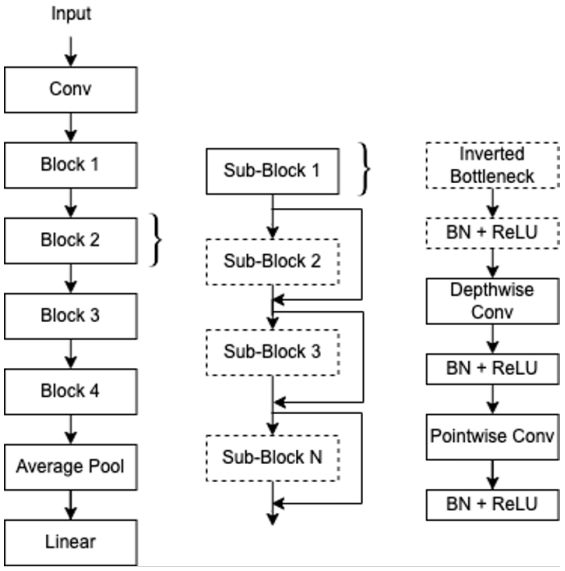


Figure 4.1.: Macro, Block, and Sub-block architectures for the OFAKWSNet

OFAKWSNet complete architecture

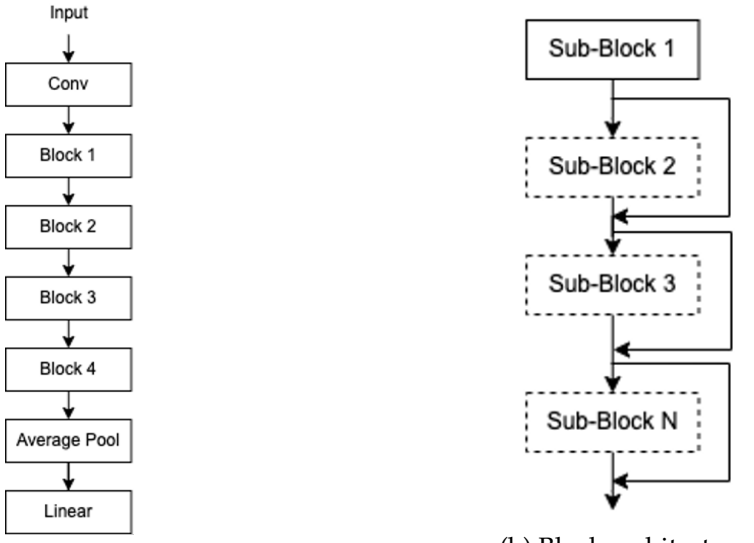
In the larger picture, the OFAKWSNet can be described by its fixed macro architecture. Fig. 4.1 shows the Macro architecture, Block architecture and Sub-block architecture of the OFAKWS network. Dotted lines represent elastic features in the macro-architecture which can be non-present when selecting a sub-network from the NAS space, this is explained more in depth below.

We first start by describing the macro architecture located on the left of Fig. 4.1, then will move on to the block architecture in the middle, and finally will describe our sub-block architecture.

Macro & Block Architecture

The macro architecture Fig. 4.2a consists of a modulable input stem (which we set as a Convolutional Layer in our case), followed by a succession of four macro blocks. After which there is an average pooling layer and a final linear layer, the classifier.

The blocks in this macro architecture are themselves constituted of one to four residual sub-blocks Fig. 4.2b. We denominate this by the *elastic depth* in our neural architecture search space.



(a) Macro architecture

(b) Block architecture

Figure 4.2.: OFAKWSNet macro and block architectures

Sub-block Architecture

Finally, let us look more in depth into the residual sub-blocks composing the blocks in Fig. 4.3.They are made up of a succession of an inverted bottleneck layer, known for their usage in the recognised MobileNet architecture, and depthwise separable convolutions, namely a depthwise convolution followed by a pointwise convolution. Between each of these layer we perform Batch Normalization and ReLU activation.

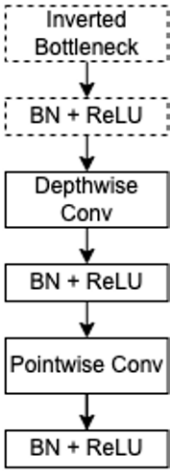


Figure 4.3.: Sub-block architecture

We can denote two more elastic architecture features in these sub-blocks. First, the kernel size of each layer in this architecture can be modified, we denominate this as the *elastic kernel size* in our neural architecture search space. Secondly, we denote an elastic expand ratio in the initial inverted bottleneck layer, which determines the factor with which the incoming channels will be expanded. We denominate this by *elastic expand ratio* in our neural architecture search space. Note that when the expand ratio is set to one, this layer is removed. We now have to select a neural architecture search space in order to run the OFAKWS network.

4.2.2. Neural architecture search space

As we introduced in the macro architecture description, the neural architecture search space setting is based on our choice of the three elastic parameters of the once-for-all network:

1. The elastic depth;
2. The elastic kernel size;
3. The elastic expand ratio.

Elastic feature OFAKWSNet	Search space
Input resolution	cf Table 4.6
Kernel size	{3,5,7}
Block depth	{1,2,3,4}
Expand ratio	{1,2,3}

Table 4.1.: Neural architecture search space for the OFA network

As our intent is to focus on network architectures deployable on micro-controllers, we choose a relatively small search space, described in Table 4.1. Furthermore, we fix the number of channels of each blocks to 64, matching the popular DS-CNN [8] architecture. Note that our framework however also supports the possibility of an elastic width.

In the end, as we train these networks on more than 100 different feature extraction configurations (taking feature extraction methods and their various parameters); we roughly obtain more than $100 \times (3^4 \times 4 \times 3^4)^4 > 10^{19}$ different sub-networks. Following the functioning of the original once-for-all implementation, the training procedure will then train all network configurations within this search space simultaneously. The next Section 4.3 will treat with why and how we use once-for-all in order to evaluate and compare feature extraction methods for keyword spotting.

4.3. Once-for-All for feature extraction evaluation: Feature extraction methods

As mentioned earlier, the first problem to overcome is to find suitable neural network architectures for any given input, or feature extracted input. In order to overcome this, we over-view in Section 4.2 the model architecture search space and once-for-all as a solution in order to find the best sub-network configuration for a given input. This section describes the feature extraction types and parameters used in this project to train and evaluate the once-for-all network.

The feature extraction methods evaluated in this project are the MFCC and its intermediate steps, from the raw input to the Log Mel Spectrogram. In order for the once-for-all training process to function, we need to vary the feature extraction parameters for each run in order to simulate resolution change in the resulting spectrogram.

Particularly, we focus our attention on the Log Mel Spectrogram (Section 4.3.1) and the MFCC feature extraction methods (Section 4.3.2), as they have proven proficiency in KWS.

4.3.1. (Log) Mel Spectrogram parameters impact on the resulting Spectrogram

We seek to modify the Mel Spectrogram or the Log Mel Spectrogram generation through two of its parameters.

1. The number of Mel bins

The number of Mel bins, also known as Mel filters or Mel filterbanks, used to generate the Mel Spectrogram controls the resolution of the resulting spectrogram on the frequency axis. Intuitively, the higher the number of Mel bins, the higher the resolution on the frequency axis. Fig. 4.4 shows the increasing resolution of the resulting spectrogram in its y-frequency axis. We also provide in Table 4.2 the increase in the number of generated features as we augment the number of Mels used to generate the Spectrogram.

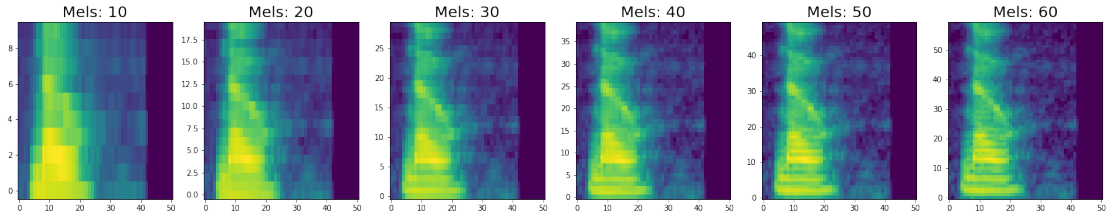


Figure 4.4.: Impact of the number of Mel filter-banks on the resulting Mel Spectrogram, for fixed window length 40ms

Mels	Resolution	Features
10	[10, 51]	510
20	[20, 51]	1020
30	[30, 51]	1530
40	[40, 51]	2040
50	[50, 51]	2550
60	[60, 51]	3060

Table 4.2.: Mel Spectrogram features in function of the number of Mel bins

2. The window length

The window length controls the resolution of the resulting spectrogram on the time axis. Intuitively, the bigger the window length, the smaller the resolution on the time axis. The impact the window length has on the Mel Spectrogram resolution is visualised in Fig. 4.5 and described in Table 4.3.

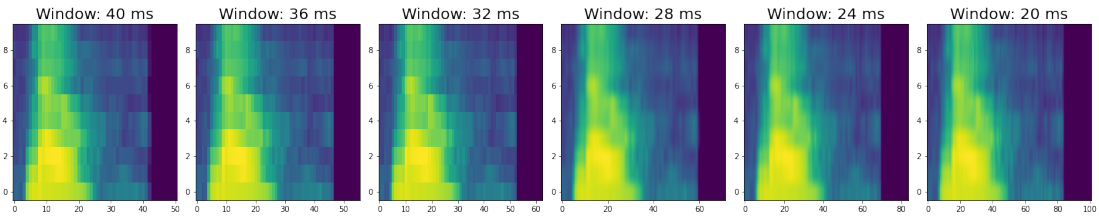


Figure 4.5.: Window length (in ms) impact on the resulting Mel Spectrogram, for fixed 10 Mel filter-banks

Win	Resolution	Features
40	[10, 51]	510
36	[10, 56]	560
32	[10, 63]	630
28	[10, 72]	720
24	[10, 84]	840
20	[10, 101]	1010

Table 4.3.: Mel Spectrogram features in function of the number of Mel filter-banks

4.3.2. MFCC parameters impact on the resulting MFCC Spectrogram

We seek to modify the MFCC generation through three of its parameters. The number of MFCC bins, the number of Mel filter-banks, and the window length used.

1. The number of MFCC bins

The number of MFCC bins conserved controls the cutoff point for the number of Mel Frequency Cepstrum Coefficients retained. Intuitively, this number defines a "crop" to apply on the MFCC spectrogram. The impact of the number of MFCC bins on the resulting MFCC is visualised in Fig. 4.6 and its resulting resolution is described in Table 4.4.

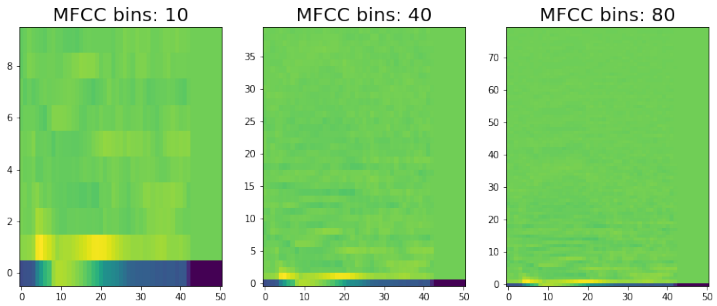


Figure 4.6.: MFCC bins impact on the resulting MFCC features, for fixed 30ms window length and 80 Mel filter-banks.

MFCC bins	Resolution	Features
10	[10, 51]	510
40	[40, 51]	2040
80	[80, 51]	4080

Table 4.4.: MFCC features in function of the number of MFCC bins, for fixed 40ms window length

Even though at first glance it may seem there is not much more information present on the MFCC when taking more MFCC bins, it is simply because MFCCs orders coefficients such that the higher their order, the more levels of spectral details they represent. Therefore depending on the complexity of our task and our model, we might want to vary this parameter.

2. The number of Mel filterbanks

The number of Mel filter-banks used to generate the MFCC controls the resolution of the resulting spectrogram on the frequency axis. This corresponds to the number of Mel bins in the previously seen Log Mel Spectrogram, as it is an intermediate step in the MFCC generation. Intuitively, the higher the number of Mel bins, the higher the resolution on the frequency axis of the pre-processed LMS. It does not affect the resulting resolution of the output, however, we have an evident constraint imposed by the number of Mel filter-banks used; the number of Mel filter-banks must be greater or equal to the number of MFCC bins. We adjust the feature extraction search space accordingly in our experiments.

3. The window length

The window length controls the resolution of the resulting spectrogram on the time axis. Intuitively, the bigger the window length, the smaller the output resolution on the time axis. The impact of the window length on the resulting MFCC is visualised in Fig. 4.7 and its resulting resolution is described in Table 4.5.

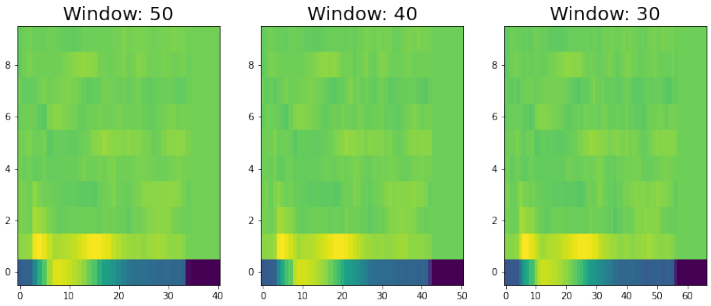


Figure 4.7.: Window length (in ms) impact on the resulting MFCC features, for fixed 10 MFCC bins

Win len (ms)	Resolution	Features
50	[10, 41]	410
40	[10, 51]	510
30	[10, 67]	670

Table 4.5.: MFCC features in function of the window length (in ms), using 10 MFCC bins

4.3.3. Feature extraction methods & parameters search space summary

The summary of analysed parameters for each feature extraction methods can be found in Table 4.6 below.

Feature extraction method	Varied parameters
Raw input	
Log Spectrogram	1. Number of FFTs 2. Hop length
(Log) Mel Spectrogram	1. Number of Mel bins 2. Window length
MFCC	1. Number of MFCC bins 2. Window length 3. Number of Mel filters

Table 4.6.: Feature extraction parameters search space

The exhaustive list of each feature extraction methods parameters which we train and evaluate the OFA network on can be found in Appendix C.

4.3.4. OFA model training with Progressive Shrinking

The progressive shrinking algorithm is modified to accommodate elastic feature extraction methods instead of the original implementation's flexible resolution.

Therefore, our training process will start by training the largest network of the neural architecture search space, for every feature extraction parameters, and will progressively incorporate smaller and smaller sub-networks in the neural architecture search space used for training the OFA network.

To train any network on all resolutions during training, we perform the following:

1. For each training batch, we start by sampling a feature extraction configuration which we then apply to the batch in order to have the batch dataset.
2. For each iteration in our chosen dynamic batch size (from 1 to 4, increasing as we go through the training steps, from kernel incorporation to expand incorporation), we then randomly sample and assign a sub-network configuration to the OFA network. Therefore, only weights associated with this particular sub-network are activated during this process.

3. We then update the weights of the OFA network using the combined loss of the network with the loss of the largest, the initial network, which we call the teacher network.

This procedure allows us to simultaneously train all sub-network architecture configurations in the neural architecture search space for all feature extraction methods in our feature extraction search space.

We describe the changes made to the code to accommodate for feature extraction variability in Section 4.4.

4.3.5. Comparing feature extraction methods by sampling OFA sub-networks

As the once-for-all network is trained for all resolutions/feature extraction configurations, we can infer important information on feature extraction configuration and their impact on model inference by evaluating the resulting sub-networks. Namely, we choose to randomly sample sub-networks configurations and evaluate them on each feature extraction configuration in the parameters search space defined in Table 4.6. Sampling and evaluating as such allows us to gather any measurable evaluation metric such as the accuracy, number of parameters, number of floating point operations (FLOPs), memory or latency.

For instance, by visualising each sub-network's accuracy with respect to their FLOPs, we will be able to determine for each FLOPs regime-levels, which is the best performing sub-network architecture and feature extraction parameters pair, and therefore which is the best performing feature extraction method for the given regime.

These observations will be made and discussed in the following Chapter 5.

Model training with Progressive Shrinking

Therefore when training the once-for-all network, we train simultaneously all sub-networks in the neural architecture search space for all feature extraction configurations within our feature extraction parameters search space.

For each training batch, we randomly sample and apply feature extraction parameters from the predefined search space. This is how progressive shrinking incorporates elastic resolution and therefore how the network is able to learn various feature extraction resolutions at once.

4.4. Python implementation

The code for this project is available at:
https://github.com/NielsEscarfail/FeatureExtractionKWS [18].

This project was implemented using the Pytorch 1.10.2 framework.

The Once-For-All framework [1] serves as a base for this project. The elastic and non-elastic network layers and operations, progressive shrinking scripts and utility functions are reused from this framework. We also follow their code structure and modify the data-loading procedure, run configuration, and run management in order to implement elastic feature extraction. Furthermore, we implement a new OFA architecture, implying the customization of the progressive shrinking functioning and sub-network configurations sampling.

Our implementation enables to train an OFA network from scratch, something not supported by the original implementation of the training scripts. Finally, we create a sub-network evaluation script, enabling us to sample and evaluate sub-networks for each given feature extraction parameters, to create the data-set used to analyse feature extraction methods.

4.4.1. Main contributions and differences with the original OFA framework

In order to enable the OFA training process to support different feature extraction configurations, we have to modify OFA's elastic resolution. Furthermore, implementing a new OFAKWSNet architecture requires many adjustments, all of which are described below.

Implementing elastic feature extraction

To modify OFA to support the efficient loading of various feature extraction methods, we need to enable the passing of feature extraction parameters configurations to the data loading process used to train and evaluate the OFA network.

Therefore, we introduce the KWSRunConfig and our modified RunManager enabling the passing of feature extraction parameters configurations to the data loading section, which is the DataProvider described below.

We provide a KWSDataProvider which handles:

1. Initialization of feature extraction transformations within our feature extraction search space.
2. Data loading from the dataset and data-augmentation.

3. Transformation of the raw audio data with the chosen feature extraction method and parameters, which are either random (for training) or arbitrary (for evaluation during training).

In order to efficiently compute the transformations and not slow down the training process, the `KWSDataProvider` pre-loads and augments the data using the `PyTorch DataLoader` with four workers. Customizing the `collate_batch` function allows us to modify the feature extraction parameters for each batch during training, effectively allowing the network to learn multiple feature extraction input resolutions.

Implementing OFAKWSNet

The OFAKWSNet architecture is implemented to support four different elastic neural architecture parameters:

1. The elastic kernel size;
2. The elastic depth;
3. The elastic expand ratio;
4. The elastic width: denoting the number of channels between each sub-blocks in the each macro-block's architecture.

Note that for performance and simplicity reasons, as well as the fact that we wanted to ensure the DS-CNN-small architecture, which has proven proficiency in KWS, was contained as a subset of our neural architecture search space; we decided to not perform search on the elastic width in our runs and fix it to 64.

We implement OFAKWSNet using a similar structure as the provided OFA models in the original OFA implementation [1]. We re-use the provided regular and elastic neural network operations and layers in order to constitute the KWSNet and OFAKWSNet. Besides the fact that our architecture is new, our implementation of OFAKWSNet is notably able to vary the kernel size, depth, expand ratio, and width for a total OFA run; in contrast to the original MobileNetV3 and ResNet OFA implementations, which are only compatibly with kernel, depth, and either width or expand. This is implemented in the `set_active_subnet` and `sample_active_subnet` functions of the OFAKWSNet located in the `once_for_all/elastic_nn/networks/ofa_kws_net.py` file. Furthermore, to gather and set OFA network configurations are suited to our architecture.

Implementing training and evaluation

Finally, the training and evaluation of the OFAKWSNet is modified in order to enable our task.

First the training with progressive shrinking and validation of the OFAKWSNet is modified to support feature extraction setting and modification in `once_for_all/elastic_nn/training/progressive_shrinking.py`. The complete training run is managed in `train_ofa_net.py`, in which we enable to train an OFA network from scratch (from the largest network down to our entire search space), contrarily to the original implementation which only allows pre-trained models loading. As in the original OFA description, this script instantiate the OFAKWSNet model and the largest network to be used as a teacher network, and launches the RunManager to train the model.

The evaluation and results gathering script is located in `eval_ofa_net.py`, in which we use the modified method PerformanceDataset from `once_for_all/evaluation/perf_dataset.py`, allowing the consistent sampling and evaluation of sub-network configurations. It then saves the sampled sub-network evaluation results as a `.json` file.

We use this section to mention we implemented in the same manner as the original OFA network implementations an unused Architecture Encoder, located in the `once_for_all/evaluation/arch_encoder.py` file. This architecture encoder can be reused for further research in order to train an external neural network to predict network accuracy (or any other metrics) from the given sub-network architecture encoding. This would in consequence enable efficient evaluation and deployment on any devices, as was the original purpose of the Once-For-All [1] framework.

Code organisation

The code directory structure can be found in Appendix A.
The code organisation and general code references are described in Appendix B.

Chapter 5

Results

5.1. Evaluation setup

5.1.1. Dataset

KWS has considerably less publicly available data than other ASR tasks, and we, therefore, turn ourselves to the standard dataset used for KWS, meaning the Google Speech Commands Dataset. We train and evaluate our models using the Google-Speech-Commands-V0.02 dataset [19], containing utterances of 12 keywords "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", "Go" alongside a "silence" class (no words spoken) and "unknown words". The unknown words in the dataset allow us to distinguish unseen words; for these reasons, most speakers only said them once during the recording phase. These include "Bed", "Bird", "Cat", "Dog", "Happy", "House", "Marvin", "Sheila", "Tree", and "Wow".

We partition the data set into training, validation and testing sets, each containing 33200, 4000 and 4400 samples, respectively. We ensure the partitioning consistency because the audio filename hash determines this split; therefore, we do not have to worry about a repartition bias in our different runs.

5.1.2. Model training

Data augmentation

Before going through feature extraction, 80% of the raw audio samples are augmented by adding in a randomly picked background noise (white noise or real recorded noisy environments), also amplified by an $\mathcal{U}(0,1)$ factor. Finally, the audio samples are shifted by a $\mathcal{U}(-200,200)$ milliseconds factor before feature extraction.

Training setup

We use the architecture space described in Table 4.1. For training the full network, we use the standard SGD optimizer with Nesterov momentum 0.9 and weight decay 3e-5. The initial learning rate is 0.001 and we use the cosine schedule [20] for learning rate decay. Every run is done with batch size 512 on 4 GPUs, on a 128 GB, 20 cores computing cluster.

The largest network is trained for 140 epochs, for all feature extraction parameters selected as the feature extraction parameters search space. We then follow the Progressive Shrinking schedule to progressively incorporate and fine-tune smaller sub-networks. To be precise, Table 5.1 depicts the increase in the neural architecture search space and the number of epochs for each training phase. Note we further divide some training steps into phases as [1] has shown it improves the ending achieved accuracy and consistency of the resulting sub-networks.

Training phase	Kernels	Depth	Expand ratios	Epochs
Largest Network	7	4	3	140
Kernel	{3,5,7}	4	3	100
Depth 1	{3,5,7}	{3,4}	3	25
Depth 2	{3,5,7}	{2,3,4}	3	25
Depth 3	{3,5,7}	{1,2,3,4}	3	100
Expand 1	{3,5,7}	{1,2,3,4}	{2,3}	25
Expand 2	{3,5,7}	{1,2,3,4}	{1,2,3}	100

Table 5.1.: OFAKWSNet training phases

The training process duration varies heavily depending on the feature extraction method and thus input resolution. It ranges from 6 hours with the smallest MFCC resolution to more than 20 hours for the largest Spectrogram features.

Validation is performed every 2 epochs during training and we reload the best performing epoch of the previous phase when starting a phase to avoid over-fitting. In order to consistently measure the once-for-all network accuracy across validation steps, we select the "edge" sub-networks from the currently training neural architecture search space. For instance, referring to Table 5.1, training during the Depth 2 phase, we would first evaluate the smallest possible network in this phase, thus, with every kernel size set to 3, every block depth set to 2, every expand ratio set to 3, for every feature extraction transformation in our feature extraction parameters search space. We would then also evaluate the largest network and intermediary reference sub-networks (for instance with all kernel sizes set to 5).

At the end of training, we end up with a once-for-all network capable of extracting any given trained specialized sub-network configuration. The next section describes the sampling procedure used for evaluation.

5.1.3. Sampling procedure

We train multiple OFA networks. Different runs are required for each feature extraction method (MFCC, LMS, Mel Spectrogram, Log Spectrogram, Raw input), and for certain runs we are constrained to varying only few parameters at once. For instance, the MFCC bins are incompatible with each other, as the network does not comprehend differently "cropped" inputs and is specialised for resolution. Therefore, we end up with a different OFA trained network for each feature extraction methods. The next step now is to sample and evaluate sub-networks on various feature extraction methods in order to evaluate them.

Our sampling procedure consists in randomly sampling from the neural architecture search space neural architecture configurations in the form kernel sizes, depths, expand ratios. We then save these macro architecture configurations in order to evaluate them on each feature extraction parameters for comparison. In the end, we sample and evaluate more than 200 000 sub-networks in total for all feature extraction methods.

5.1.4. Evaluation metrics

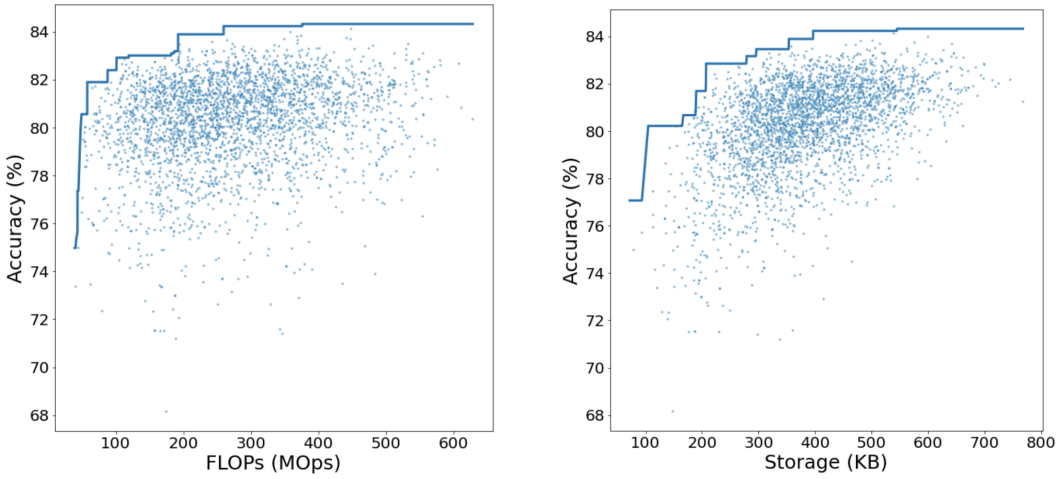
The metrics gathered when evaluating the OFA sub-networks for each feature extraction parameters can be found in Table 5.2. Note that we assume 16 bits of storage per parameters in our computations.

Metric	Unit
Accuracy	%
Floating Point Operations (FLOPs)	MOps
Storage requirements	KB
Memory requirements	KB

Table 5.2.: Metrics gathered during evaluation

5.2. Baseline: Raw audio input

We display here the OFA sub-networks evaluation results following the procedure described above. Fig. 5.1 displays the sub-networks Accuracy to Floating Point Operations (FLOPs) (in MOps) and accuracy to storage requirements (in KB, assuming 16bit parameters storage). In these and the following figures contained in this report, each dot represents a sub-network configuration, evaluated for the given feature extraction method.



(a) Accuracy / FLOPs [MOps]

(b) Accuracy / storage [KB]

Figure 5.1.: Accuracy to model requirements using the raw input

Noticeably, the highest sub-network accuracy achieved using no feature extraction is 84.32 %. We also point out the logarithmically distributed best accuracy/FLOPs and accuracy/storage curves. The subsequent evaluation and comparison of feature extraction methods will treat with attaining the best accuracy while respecting defined constraints (storage, operations, others).

5.3. Log Spectrogram features

Even though the Spectrogram, respectively the Log Spectrogram are less frequently used feature extraction methods in Keyword Spotting, they represent a well-known transformation and represent an intermediate step in the later evaluated methods. Our findings show networks using a Log Spectrogram transformed input achieve an accuracy of 92.35%.

This result is already interesting at it shows that even higher-resolution inputs can attain a high (>92%) accuracy, in opposition to the raw input. We will later view how

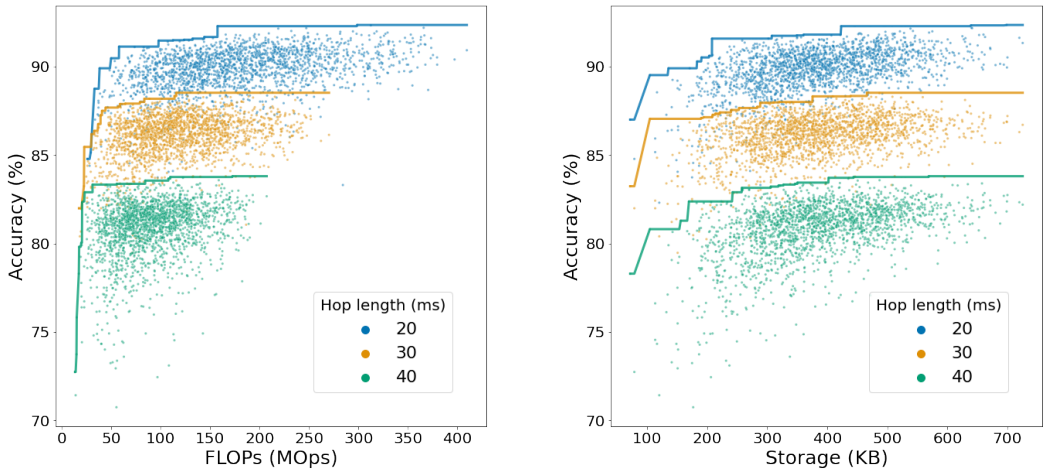
the later MFCC stages perform, but let us start by analysing the impact of a feature extraction parameter for the Spectrogram.

5.3.1. Performance impact of the hop length

We will analyse the impact of the hop length of the Spectrogram features in sampled sub-networks evaluations. Table 5.3 provides descriptive statistics of the sub-models accuracy when changing the hop length between consecutive FFTs used to generate the Log Spectrogram. First looking at the accuracy evolution with respect to the hop length used to generate the Log Spectrogram features, we notice the achievable accuracy of the network significantly is inversely correlated with the hop length; therefore the smaller the hop length, the higher the achievable accuracy of the sampled sub-networks.

Hop length	Top Acc [%]	Avg Acc [%]	Std
20	**92.35**	**89.84**	1.19
30	88.52	86.09	1.17
40	83.80	80.82	1.61

Table 5.3.: Accuracy Statistics for the Log Spectrogram feature extraction method depending on the hop length (in ms)



(a) Accuracy / FLOPs [MOps]

(b) Accuracy / Storage [KB]

Figure 5.2.: Accuracy to model requirements using the Log Spectrogram in function of the hop length (in ms)

Fig. 5.2 displays the accuracy to FLOPs and accuracy to Storage (in KB) for the sampled sub-networks using the Log Spectrogram feature extraction method, with varying hop length (the length of the non-intersecting portion of window length) of 20, 30, 40ms. The distribution of the sub-networks accuracy to FLOPs shows a large hop length induces a reduced achievable accuracy in the OFA sub-network. This is expected as the lesser the overlap (greater the hop length), the fewer dynamics are captured in the STFT, as all windows weight centered samples higher than samples at the edges. The information loss in our case proves to be substantial when evaluating sub-networks on spectrograms generated with different hop lengths. Therefore, a larger hop length induces less overlapping when computing the FFTs, which causes the spectrogram to capture fewer dynamics.

5.4. Log Mel-Spectrogram features

5.4.1. Performance impact of the number of Mel filterbanks

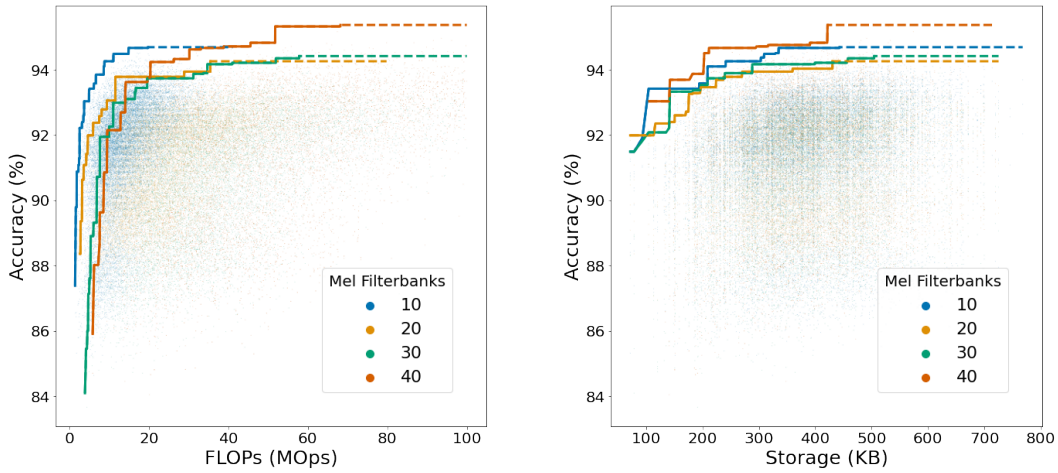
We will analyse the impact the number of Mel filterbanks, also called Mel bins or triangular filters, of the Log Mel Spectrogram features in sampled sub-networks evaluations. Table 5.4 provides descriptive statistics of the sub-models accuracy when changing the number of Mel filterbanks used to generate the Log Mel Spectrogram. We show the accuracy quartiles (Q1, Median, Q3) as well as the threshold indicating the top 5% of the sub-networks (Q.95). We first notice the best performing network with LMS feature extraction achieves 95.37% accuracy using 40 Mel bins. However, looking at the detailed statistics, we also observe than 10 Mel bins achieves a high accuracy in the end, while being even more consistent in its results; indeed, Table 5.4 10 Mel bins is even superior to 40 Mels in our evaluation in terms of average accuracy and quartiles. Finally, all top 5% of networks using 10 Mels achieve more than a 93.49% accuracy; this shows the higher accuracy relative to intermediate number of bins (20 and 30 in our case) is not simply due to a sampling outlier case.

Mel bins	Top Acc [%]	Avg Acc [%]	Std	Q1 [%]	Median [%]	Q3 [%]	Q.95 [%]
10	**94.69**	**91.52**	1.62	**90.81**	**91.86**	**92.70**	93.49
20	94.26	91.09	1.36	89.97	91.31	92.24	92.97
30	94.42	90.67	1.81	89.36	91.02	92.15	93.08
40	**95.37**	91.26	1.92	89.84	91.67	92.79	**93.76**

Table 5.4.: Accuracy Statistics for the Log Mel Spectrogram feature extraction method depending on the number of Mel filterbanks used

Accuracy to FLOPs analysis

A first observation to be made observing Fig. 5.3 is that the highest accuracy is reached with the highest number of Mel filterbanks. However, analysing more specifically the lower-regime networks by visualising only those with only less than 30M FLOPs, we can visualize the Log Mel Spectrogram number of Mel filters which are the most rewarding (highest accuracy with regards to FLOPs).



(a) Accuracy / FLOPs [MOps]

(b) Accuracy / Storage [KB]

Figure 5.3.: Accuracy to model requirements using the Log Mel Spectrogram in function of the number of Mel filter-banks used

In our case, Fig. 5.4 shows the low-regime (<30MOps) favours a low number of Mel filterbanks (10) ; however the highest accuracy is reached with the highest number of filterbanks, therefore we make the two following conclusions. First of all, the lower regimes favours fewer Mel-filterbanks. Secondly, in order to achieve a higher achievable accuracy, one has to increase the number of Mel-filterbanks. Finally, using a relatively large number of Mel filterbanks is also advised to solve more complex tasks than KWS.

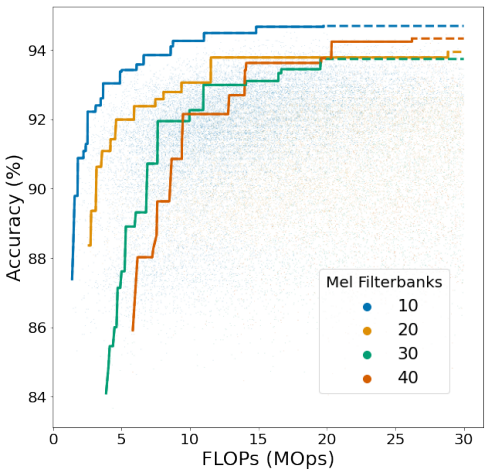


Figure 5.4.: Accuracy / FLOPs [MOps] in function of the number of Mels for sampled sub-nets with less than 30 MOps

Accuracy to memory analysis

Now looking at the memory impact of the number of Mel filterbanks, we show in Fig. 5.5 the importance of the number of Mel filterbanks in determining sub-networks' memory constraints. Indeed, a large majority of, and the best performing, networks using under 100 KB memory use 10 Mel filters to compute the LMS. The evaluation framework thus enables us to select feature extraction parameters depending on the constraints we may have for a deployment platform.

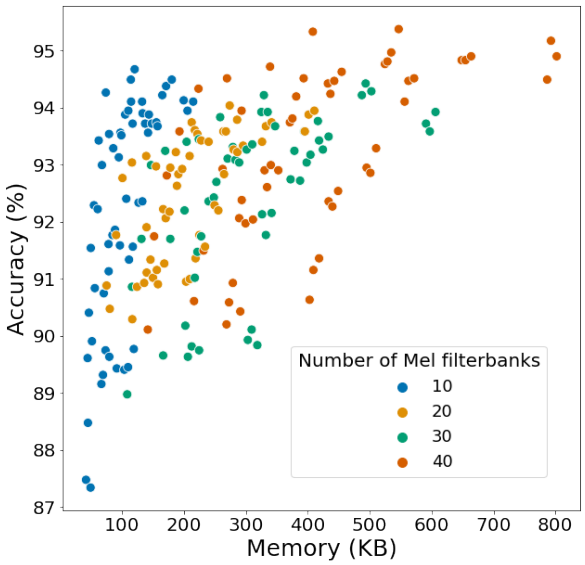


Figure 5.5.: Accuracy / memory [KB] in function of the number of Mels for sampled sub-nets using LMS

5.4.2. Performance impact of the window length

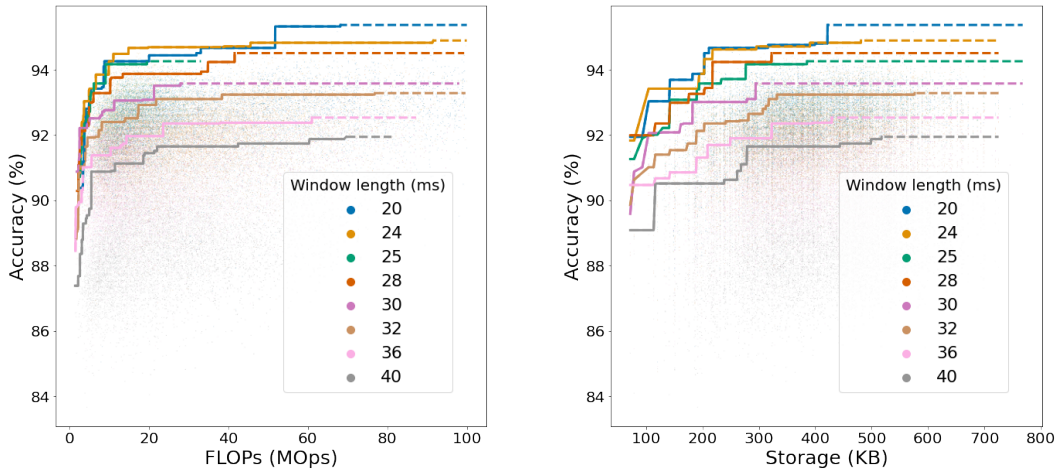
We will analyse the impact of the window length of the Log Mel Spectrogram features in sampled sub-networks evaluations. Table 5.5 provides descriptive statistics of the sub-models accuracy when changing the window length used to generate the Log Mel Spectrogram.

Window length (ms)	**20**	**24**	**25**	28	30	32	36	40
Top Acc [%]	**95.37**	94.90	94.26	94.51	93.58	93.28	92.54	91.95
Avg Acc [%]	92.47	**92.49**	92.48	91.96	91.56	90.89	89.81	88.48
Q1 [%]	92.01	92.01	**92.04**	91.54	91.15	90.45	89.31	87.75
Q.95 [%]	93.72	**93.81**	93.56	91.13	92.58	92.11	91.13	90.18

Table 5.5.: Accuracy Statistics for the Log Mel Spectrogram feature extraction method depending on the window length (in ms)

As we can see in Table 5.5, the smaller the window length when generating the Log Mel Spectrogram, the higher the ending network accuracy. We also provide the first quartile "Q1" as well as the threshold indicating the accuracy of the top 5% of networks "Q.95" in order to show this relationship holds both for the average and the edge (worse and best) cases. The most accurate sampled sub-network in the end uses a window length of 20ms, achieving 95.37% accuracy. Interestingly, we can start to notice that a window length of 24ms or 25ms perform almost as well as a 20ms window length everywhere except for their best achieved accuracy.

Accuracy to FLOPs analysis



(a) Accuracy / FLOPs [MOps]

(b) Accuracy / storage [KB]

Figure 5.6.: Accuracy to model requirements using the Log Mel Spectrogram in function of the window length (in ms)

Fig. 5.6 visualises the evaluated sub-networks accuracy depending on their window length, in milliseconds. A first observation we can make is that larger window lengths tend to restrict the samples sub-networks in terms of achievable accuracy.

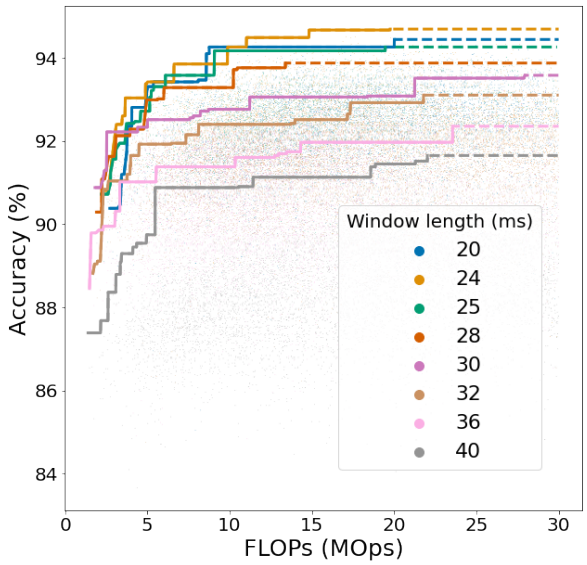


Figure 5.7.: Accuracy / FLOPs [MOps] in function of the window length (ms) for sampled sub-nets with less than 30 MOps

Looking more in-depth into the sub-networks functioning with less than 30 MOps, Fig. 5.7 shows that in the lower regimes (less than 10MOps) for the Log Mel Spectrogram, networks using a bigger window length achieve the same accuracy than sub-nets using a smaller window length.

If the intent is to consider low-power usage solely, one would prefer a more significant window length as it would imply less computational effort and memory usage as the resulting input resolution is smaller.

However, as we go into the intermediary 10-35MOps regime, the best performing networks use a window length of 24ms, and later for more than 35MOps, the best window length in terms of accuracy is 20ms. As the difference between respective window sizes and accuracy is small, it is hard to conclude with complete certainty there is an optimal window size value depending on the regime for the low and intermediary regimes.

Accuracy to memory analysis

Fig. 5.8 depicts the evolution of the LMS sub-networks' accuracy depending on their memory requirements (in KB). This figure shows how as we decrease the window length, the network's achievable accuracy increase, but so does their memory. OFAKWS allows us to define the accuracy to memory trade-off depending on the window size. Therefore, we can use it to clearly define the optimal window length configuration depending on any given memory requirements. For the LMS, the best window length and its respective accuracy depending on memory is explicitated in Table 5.6.

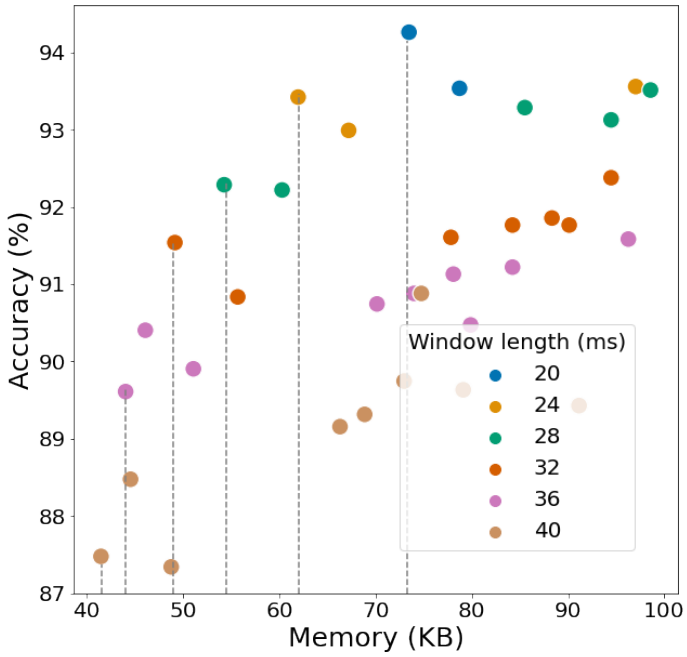
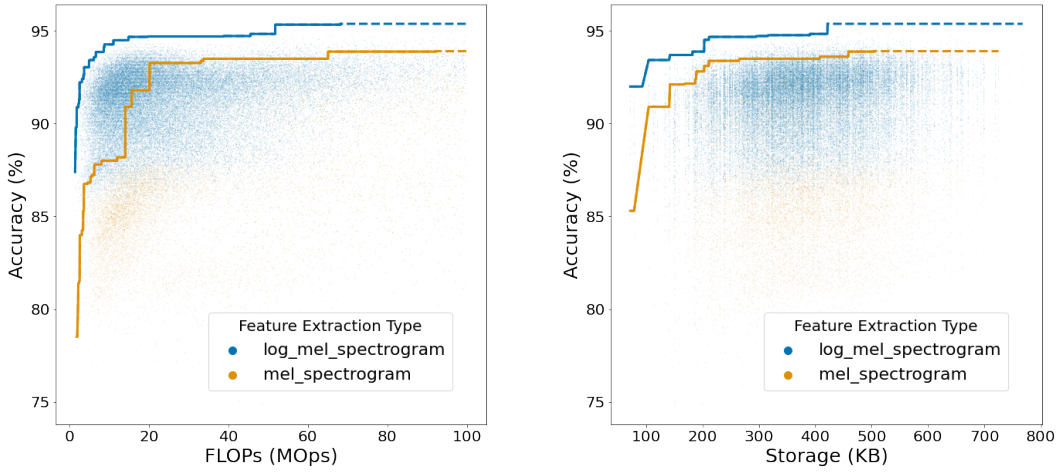


Figure 5.8.: Accuracy / memory in function of the window length [ms] for sampled sub-nets using the LMS

Memory requirements [KB]	≤ 44	[44 - 48]	[48 - 54]	[54 - 62]	[62 - 73]	≥ 73
Attained accuracy [%]	87.52	89.75	91.62	92.36	93.50	94.21
Best window length [ms]	40	36	32	28	24	20

Table 5.6.: Best LMS window lengths (in ms) depending on the sub-network's memory (in KB)

5.4.3. Log-Mel-Spectrogram vs Mel-Spectrogram features



(a) Accuracy / FLOPs [MOps]

(b) Accuracy / storage [KB]

Figure 5.9.: Accuracy to model requirements using Log- and standard Mel Spectrogram features

Fig. 5.9 explicitly shows the accuracy and performance gained by using the Log Mel Spectrogram versus the Mel Spectrogram. The difference between both feature extraction methods is substantial, we can deduce the decibel scale is substantially easier to learn and differentiate by our sampled sub-networks. Indeed, the raw Mel Spectrogram is displaying power as its third dimension, taking the log gives us a decibel scale. Intuitively, it makes sense that the feature extraction method using the scale in which humans hear is able to differentiate better than the one using the power scale.

5.5. MFCC features

This section describes the performance impact of the MFCC parameters on the resulting evaluated sub-networks.

5.5.1. Performance impact of the number of MFCC bins

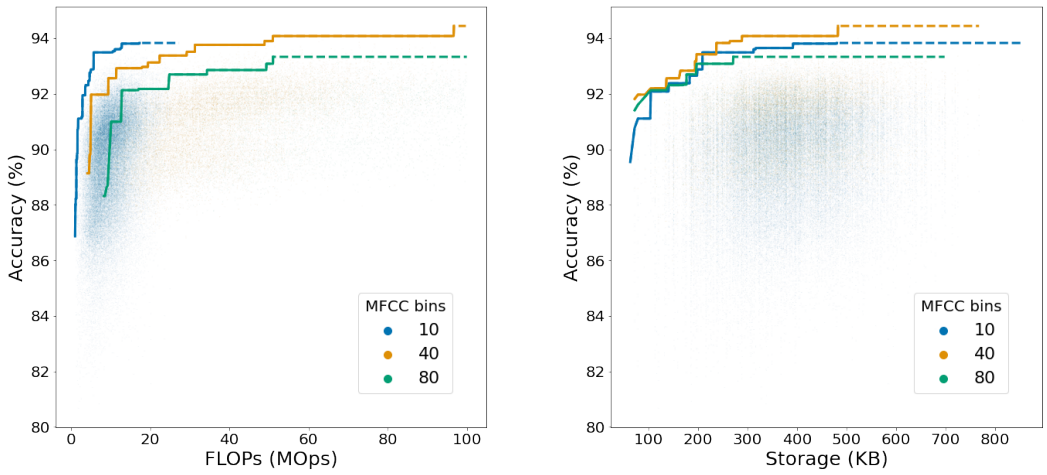
First analysing the sub-networks accuracy with respect to the number of MFCC bins used in Table 5.7, we can see that in our context, all the networks achieved a relatively high accuracy in the end. One interesting observation to be made is the networks using more MFCC bins are able to achieve a high accuracy more consistently, as can be seen

by the 95th percentile and other relevant statistics of the 80 MFCC bins row in Table 5.7. This can be explained as the number of Mel filter-banks used to generate such MFCCs is 80 or more, and as we have seen before and we will continue to see in the next section, increasing the number of Mel-filterbanks increases the data granularity and the achievable accuracy of sub-networks, however, that implies larger and more complex networks are required in order to better grasp this granularity.

MFCC bins	Top Acc	Avg Acc	Std	Q1	Median	Q3	Q.95
10	93.83	90.02	1.45	89.11	90.38	91.08	91.9
40	94.44	89.93	1.82	88.88	90.13	91.33	92.38
80	93.33	91.03	1.08	90.22	91.29	91.88	92.47

Table 5.7.: Accuracy Statistics for the MFCC feature extraction method depending on the number of MFCC bins

Evaluating the sampled MFCC sub-networks and grouping them in terms of the number of MFCC bins used in the MFCC feature extraction, we can see in Fig. 5.10a that for the lower regimes (less than 35 MOps), the best performing sub-networks use 10 MFCC bins, however in the higher regimes, the best performing MFCC parameters use 40 MFCC bins.



(a) Accuracy / FLOPs [MOps]

(b) Accuracy / Storage [KB]

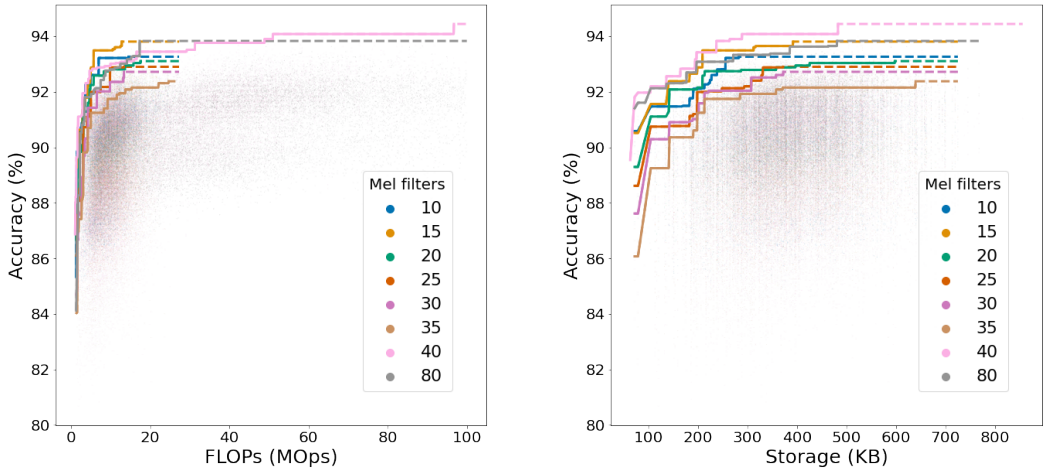
Figure 5.10.: Accuracy to model requirements using MFCC features in function of the number of MFCC bins

Therefore, the higher the number of MFCC bins, the higher the achievable accuracy if we disregard any constraints. Furthermore, using more MFCC bins might be desirable

for more complex ASR tasks than KWS. However it is clearly visible in Fig. 5.10 that the best performing configurations in the lower regime uses 10 MFCC bins.

5.5.2. Performance impact of the number of Mel filterbanks

Evaluating the sampled MFCC sub-networks and grouping them in terms of the number of Mel filterbanks used in the MFCC feature extraction, we can see in Fig. 5.11, with no regards to memory or storage constraints for now, that the higher the number of Mel filters used when generating the MFCC, the higher the achievable accuracy of the sampled OFA subnets. Intuitively, a higher amount of Mel filters lead to a higher data granularity and therefore the information is better conserved for the neural network to analyse.



(a) Accuracy / FLOPs [MOps]

(b) Accuracy / storage [KB]

Figure 5.11.: Accuracy to model requirements using MFCC features in function of the number of Mel filters

Now let us look at these results more in detail for the low-regime as we are also interested in low-power and low-memory micro-controller usages. Fig. 5.12 shows the accuracy / FLOPs trade-off in a lower-power context. For 0 to 5 MOps, the highest performing network uses 10 Mel filter-banks, however, from 10 to 20 MOps, it is 15 Mel filter-banks. Finally, the best performing OFA subnets for higher FLOPs counts were using 40 Mel filter-banks.

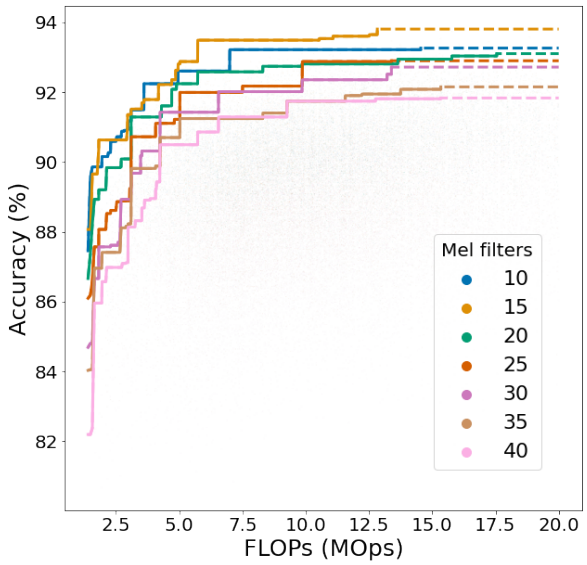


Figure 5.12.: MFCC accuracy / FLOPs [MOps] for varying number of Mel filters

Noticeably, looking at the statistics in table Table 5.8, we can notice that 15 Mel filter-banks seem to function well in both settings, achieving better accuracy than other low-resolutions and proving better consistency, with an only 0.63% accuracy difference in comparison with the highest scoring 40 Mel filter-banks.

Mel filters	Top Acc [%]	Avg Acc [%]	Q.95 [%]
10	93.26	89.97	91.81
15	**93.81**	**90.71**	**92.54**
20	93.1	89.84	91.90
25	92.9	89.15	91.42
30	92.72	88.63	91.13
35	92.38	88.08	90.56
40	**94.44**	90.05	92.31
80	93.83	90.73	92.45

Table 5.8.: Sub-nets accuracy statistics using MFCC features depending on the number of Mel Filter-banks

5.5.3. Performance impact of the window length

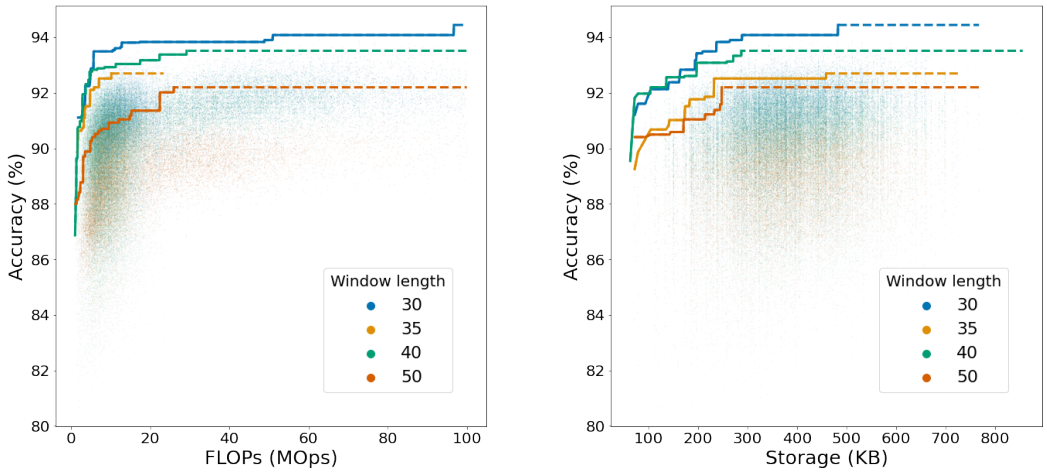
Table 5.9 shows the larger the window length used when generating the MFCC, the higher the attainable accuracy will be for the resulting network. That is, if we do not account for resources constraints.

Window length [ms]	Top Acc [%]	Avg Acc [%]
30	94.44	90.86
40	93.51	90.21
50	92.2	88.91

Table 5.9.: Sub-nets accuracy statistics using MFCC features depending on the window length (in ms)

Accuracy to FLOPs analysis

In this case, Fig. 5.13a shows the accuracy to FLOPs trade-off in the lower regime is similar even with different window lengths. This implies that in a keyword spotting pipeline deployment on a resources-constrained device, one would want to first consider a window length of 40ms to save compute power and reduce the input size as much as possible, as it there is no observed draw-down in accuracy for this regime.



(a) Accuracy / FLOPs [MOps]

(b) Accuracy / storage [KB]

Figure 5.13.: Accuracy to model requirements using MFCC features in function of the window length (in ms)

Accuracy to memory analysis

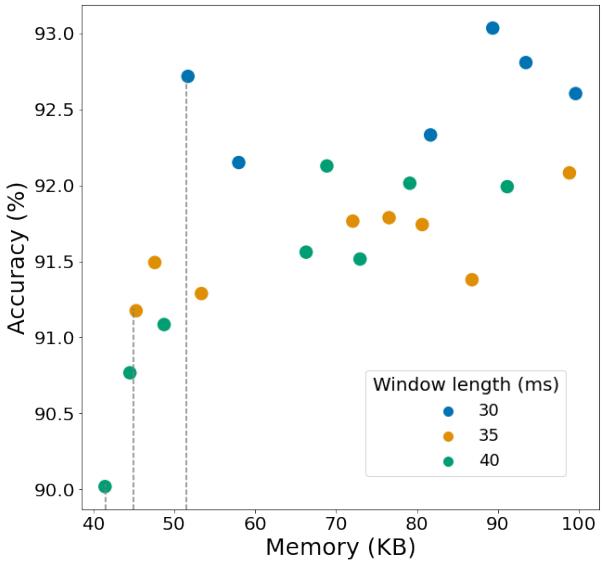


Figure 5.14.: Accuracy / memory [KB] in function of the window length (ms) for sampled sub-nets using MFCCs

Fig. 5.14 depicts the evolution of the MFCC sub-networks' accuracy depending on their memory requirements (in KB). In this case once again, the figure shows the trade-off between window length and memory usage. In our case, the sampled sub-networks indicate the optimal window length are 40 ms for the lower regime ($\leq$ 44 KB), then it is a 35ms window to achieve a memory between 44KB and 52KB. Finally, the smallest tested window length (30 ms) shows achieve the highest accuracy once we surpass a memory of 52 KB.

5.6. Feature extraction methods comparison and summary

This section presents a summary and comparison of all feature extraction types, taking into account every run's feature extraction parameters search spaces. The complete list of evaluated feature extraction parameters, their best performing networks accuracy and FLOPs can be found in Appendix C.

Table 5.10 shows the Log Mel Spectrogram is the best performing network in terms of accuracy. However, when looking at the low-operations regime, we find the MFCC to be the most prominent feature extraction method. What is striking is the Log Mel

Spectrogram's consistency in achieved accuracy over all the evaluated sub networks, as we can see it dominates the popular MFCC feature extraction methods in terms of accuracy in both mean and extreme cases. Indeed, looking at its 3 quartiles (Q1, median, Q3) and its Q.95 statistics; we see the accuracy distribution of the LMS spectrogram is more concentrated on the higher-accuracy side than the MFCC.

F.E. type	Top Acc [%]	Mean Acc [%]	Q1 [%]	Median [%]	Q3 [%]	Q.95 [%]
MFCC	94.44	90.04	89.02	90.31	91.29	92.29
LMS	**95.37**	**91.21**	**90.11**	**91.58**	**92.49**	**93.42**
Mel Spectr.	93.90	86.97	84.62	86.57	90.20	92.38
Log Spectr.	92.35	85.58	81.85	86.23	89.25	90.93
Raw input	84.32	80.27	79.37	80.63	81.60	82.65

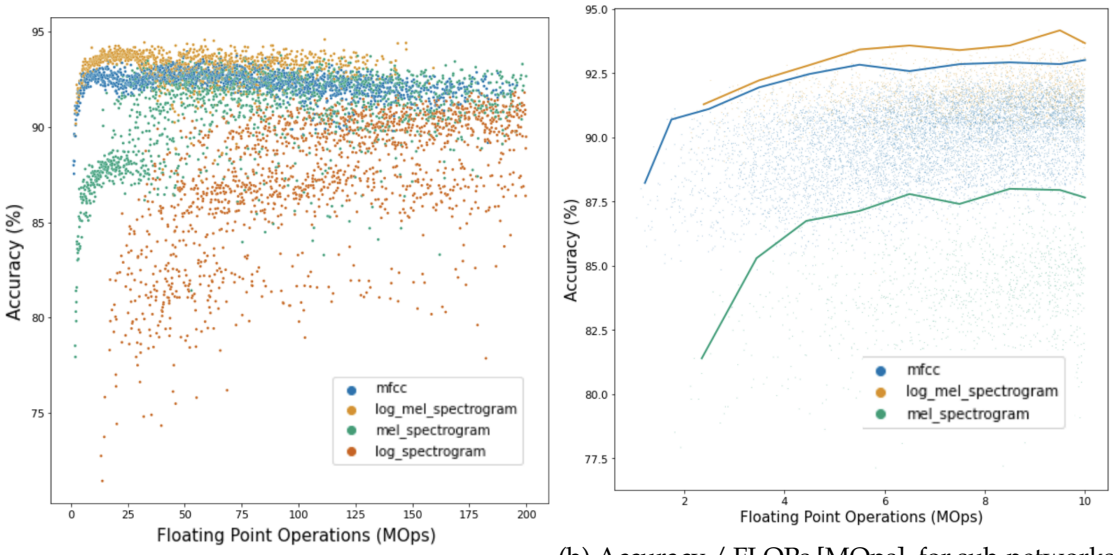
Table 5.10.: Sub-nets accuracy statistics depending feature extraction (F.E.) type

This first table did not account for any constraints in terms of performance (FLOPs, memory), therefore let us now look at how the networks number of floating point operations are distributed. Interestingly, Table 5.11 shows the networks using the LMS did not perform a significantly larger amount of FLOPs than MFCC networks.

F.E. type	Accuracy [%]			FLOPs [MOps]		
	Max	Mean	Min	Max	Mean	Min
MFCC	94.44	90.04	80.67	211.82	21.52	1.06
Log Mel Spectrogram	95.37	91.21	83.67	159.27	28.08	1.39
Mel Spectrogram	93.9	86.97	74.82	318.44	55.67	1.82
Log Spectrogram	92.35	85.58	70.74	409.72	128.94	13.22
Raw input	84.32	80.27	68.15	628.99	272.29	39.15

Table 5.11.: Sub-nets statistics depending feature extraction (F.E.) type

Fig. 5.16 shows the best-performing network architectures for each feature extraction methods. Note that in order to obtain this visualisation, we only keep the best performing networks using the same amount of FLOPs when evaluating.

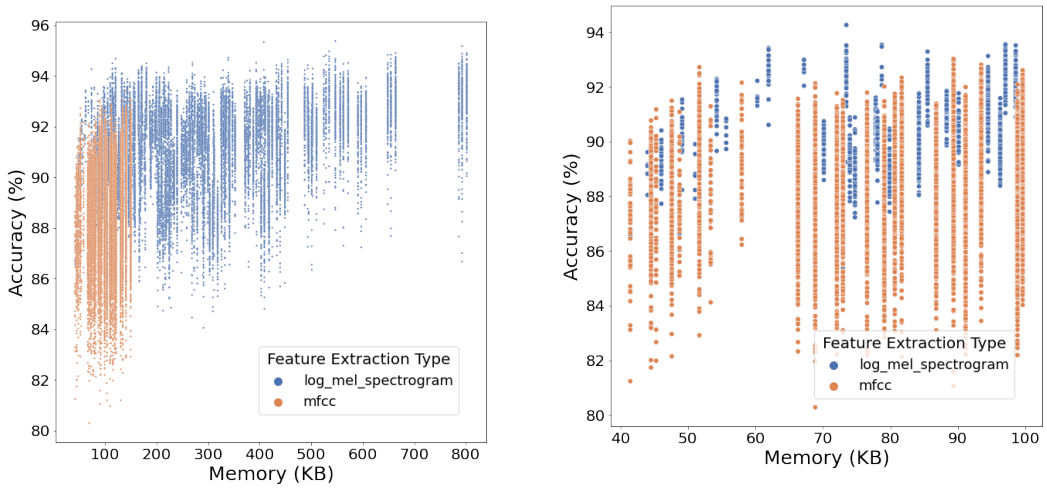


(a) Accuracy / FLOPs [MOps]

(b) Accuracy / FLOPs [MOps], for sub-networks with less than 10 MOps

Figure 5.15.: Accuracy to FLOPs for all feature extraction methods

Fig. 5.16a and Fig. 5.16b compare the resulting sub-networks' memory using MFCC (10 MFCC bins) and LMS feature extraction methods. In accordance to our previous findings, we can see that the networks using LMS feature extraction achieve a better accuracy for the same memory when operating in the higher-regime setting. Now looking at these values more precisely on Fig. 5.17 by keeping the best networks in terms of accuracy for each memory regime. We can observe a separation in which the best performing network using less than 55 KB of memory uses the MFCC, and over 55KB of memory the best accuracy is obtained using LMS feature extraction.



(a) Accuracy / memory [KB]

(b) Accuracy / memory [KB], for sub-networks with less than 100 KB memory usage

Figure 5.16.: Accuracy to memory for MFCC and LMS feature extractions

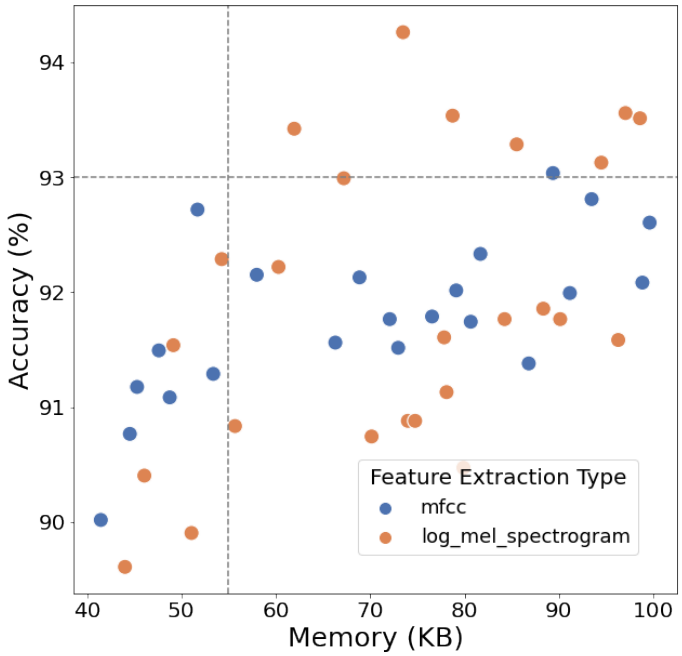


Figure 5.17.: Accuracy to memory for MFCC and LMS feature extraction

To summarise our findings, we empirically find the Log Mel Spectrogram feature extraction method to be the most prominent in terms of achievable accuracy. However, we notice that in the lower regime, which is prominent in keyword spotting applications (using less than 55KB of memory in our case, or less than 3MOps depending on which dimension we are looking at), the MFCC feature extraction method becomes desirable over the LMS. Therefore, in the lower regime, for instance when running a KWS pipeline on MCUs, using the MFCC feature extraction method is advised.

5.7. Limitations

Before discussing the shortcomings of our solution, we have to mention its many advantages. First of all, the network is consistent across runs, meaning we will get similar accuracy when training the same feature extraction parameters multiple times on different OFA runs, with different parameters search space. Secondly, using these clouds of evaluated sub-network configurations and therefore feature extraction methods, one can then specify its memory, storage, latency constraints, and filter out the corresponding sub-networks and feature extraction configurations in order to figure out the best

parameters to use for a given task and platform. It is also not restricted to feature extraction evaluation and can be reused for efficient deployment with constraints. Therefore, we can simultaneously optimise feature extraction and neural network architectures. OFAKWS thus enables us to train, evaluate and deploy neural network architectures for different KWS devices with various memory constraints in a single training, efficient way.
The following sections discuss the constraints and limitations encountered in our solution.

5.7.1. Neural Network Architecture

There are a few limitations regarding our choice of OFA macro architecture and architecture search space. First, the macro architecture in our context is quite simple, and the space quite narrow, and we could have arguable achieved better results, especially for higher resolution features by expanding this search space to more complex and large or deep architectures.

This choice of simple architecture was made as our initial goal was to compare feature extraction methods for a latter usage in a low-power environment. Furthermore, as OFA has never been used before for feature extraction evaluation, restraining the model architecture was a way to guarantee consistency and ensure a known neural network architecture, the DS-CNN was a subset of our OFA Network.

5.7.2. Feature extraction

There are limitations in the extent with which we can extend our feature extraction parameters search space. First of all, OFA does not support learning different image "crops" in the same run. For instance, in order for OFA to work, we can not train a single OFA network to predict MFCCs generated with 10 MFCC bins and 40 MFCC bins in the same run. When trying to in test runs, the OFA networks using the larger resolution (40 MFCC bins) would achieve a high accuracy but the network would not be able to evaluate smaller resolutions using 10 MFCC bins.

5.7.3. Sampling

Finally, the sampling procedure could be further enhanced in order to favour the sampling of smaller networks. Indeed, when evaluating the sub-networks memory, it turned out there were too few different data points. When randomly sampling a sub-net configuration in the neural architecture search space, there is a high probability that at least one of the randomly selected layers among all the layers in the OFA network is the largest possible in our search space (for instance kernel size 7) ; and therefore, the memory of the sampled networks is often similar.

In order to overcome this limitation, we could create a forced sampling which filters out the largest network configurations in order to specify the search for low-power evaluated sub-networks.

Moreover, building upon the `PerformanceDataset` described in Section 4.4 may allow one to remove the need for further sampling in order to obtain the expected accuracy given a sub-network and feature extraction configurations. To do so, we would need to build a dataset of network architectures and accuracy pairs using our provided KWS ArchitectureEncoder in `once_for_all/evaluation/arch_encoder.py`.

Chapter 6

Conclusion and Future Work

6.1. Conclusion

Our findings show the Log Mel Spectrogram and the MFCC features perform best among all feature extraction methods, backing their usage by many these past years, achieving an accuracy of 95.37% and 94.44% respectively on our framework. We also can show a separation between both of these popular feature extraction methods for KWS. Namely, we put forward the advantages offered by MFCC features in the lower regimes, achieving higher accuracy than the Log Mel Spectrogram features for the same memory consumption, or the same amount of FLOPs. Indeed, we are able to identify the memory constraints cut-off point defining that our networks perform better in the lower-regime (less than 55KB of memory) when using MFCC features, achieving 92.7% accuracy for 53KB memory usage. However, if we push this memory constraint, LMS networks also achieve a high accuracy for a relatively low memory, achieving 93,3% accuracy with 64KB memory and 94.3% accuracy with 75KB of memory.

Our evaluation solution using the once-for-all framework allowed us to gather relevant information on any feature extraction methods. Therefore, we can use it to determine how to parametrise a particular feature extraction method given imposed constraints

Doing so lets us notice each parameter's general impact on the resulting features. We showed that for the Spectrogram, a slight hop length and, therefore, a large window overlapping phenomenon, in general, is necessary for the sub-networks to accurately recover and comprehend information (as we lose information with a too slight or no overlapping). Interestingly, spectral leakage is often considered negligible when dealing with the Spectrogram (the magnitude of the STFT). However, our experiments clearly show a significant drop in accuracy the less we overlap successive Fast Fourier Transform windows.

Secondly, we also showed that the Decibel scale was preferential for the resulting networks to accurately classify sounds, as the Log Mel Spectrogram significantly outperformed the Mel Spectrogram in our evaluation runs, achieving an accuracy of 95.37% for the LMS in opposition to 92.36% for the Mel Spectrogram ; as well as showing less variance in its results.

To summarise the impact of the (Log) Mel Spectrogram and MFCC feature extraction parameters, the number of Mel filters used to generate any spectrogram greatly influences the achievable accuracy of the resulting sub-networks. Therefore, more Mel filters would imply a higher data granularity explaining the higher achievable accuracy. Conversely, "too few" Mel filters lead to a loss in accuracy, even for the same number of operations. However, there is a range of Mel filter counts (10 to 20 Mels) for which we do not suffer from a performance loss even though their resulting input resolution is smaller. In practice, we would use OFA to find these values for any given parameters to optimise our given accuracy to memory or storage. Furthermore, the window length used when generating the Spectrograms have a significant impact on our achievable accuracy, in the sense that a smaller window size leads to a higher data granularity and, therefore higher achievable accuracy in general. The converse is also true.

Regarding the MFCC features and the number of MFCC bins used, the KWS task can be realised with a few (10) MFCC bins. However, we can see that for more complex ASR tasks, using more MFCC bins might be desirable as it improves the sub-networks average accuracy when not accounting for constraints.

To wrap this up, we can say this framework enables us to evaluate feature extraction methods in an efficient way, attempting to overcome the neural architecture bias we face when working with feature extraction for neural network treatment. The usage of OFA, therefore, allows us to determine the constraints cutoff points at which a particular feature extraction parametrisation starts to be beneficial to use. Finally, we emphasise that this method is not restricted to Keyword Spotting and applies to any feature extraction and neural network pipeline.

6.2. Future Work

Even though our training and evaluation run aim not to find an optimal KWS network architecture for our task (but the better feature extraction method), we can implement some improvements and further analysis to obtain better-performing sub-networks.

First, improvements can be made in the choice of the macro architecture and the neural architecture search space to avoid fixed constraints.

- Allow more flexibility than our fixed four macro blocks in the OFA macro architecture. One could notably use this framework to perform NAS and feature extraction simultaneously on the most-promising KWS architectures, such as BCResNet [2] or MatchboxNet [11].
- Build a larger network, as indeed our implementation was restricted in terms of complexity, as well as using a more extensive neural architecture search space.
- Noticeably, our implementation has a fixed 64-channels width between macro blocks to allow comparison with the commonly-used DS-CNN architecture. However our code implementation still allows for elastic width, even if it is unused in the application part. Using it would allow us to search values for the number of channels between blocks.

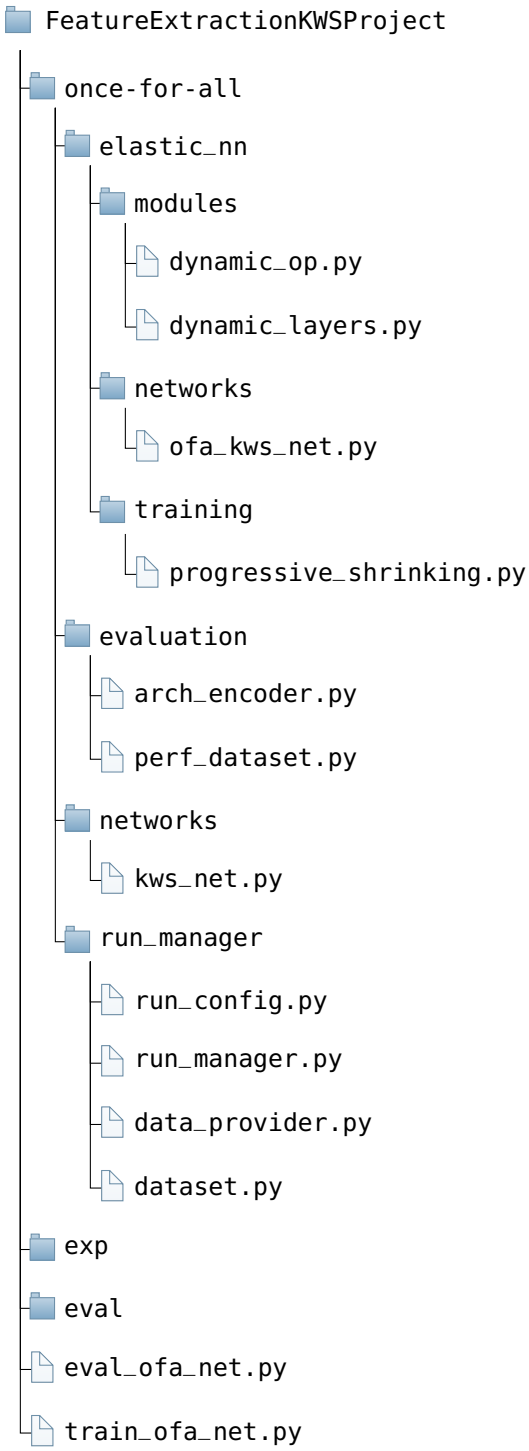
Finally, one could try to specialise this framework to focus on low-resources constrained KWS applications and, for this purpose, build a new evaluation tool which forces the sampling process to evaluate smaller sub-networks. Doing so would overcome the challenges we faced when evaluating the sub-networks memory (most of our sub-networks had similar memory constraints, as they were imposed by an inverse bottleneck layer, frequently present because of random sampling). In this regard, the architecture encoder provided in our implementation can be used to train a separate DNN to predict the KWS network's accuracy from their network configuration.

Furthermore, the resulting sub-networks architectures and their configurations can be further analysed (for instance, looking at the network kernel sizes in function of their ordering in a sub-network) in order to gather information about the functioning and shared properties of the best performing network architectures for given feature extraction configurations.

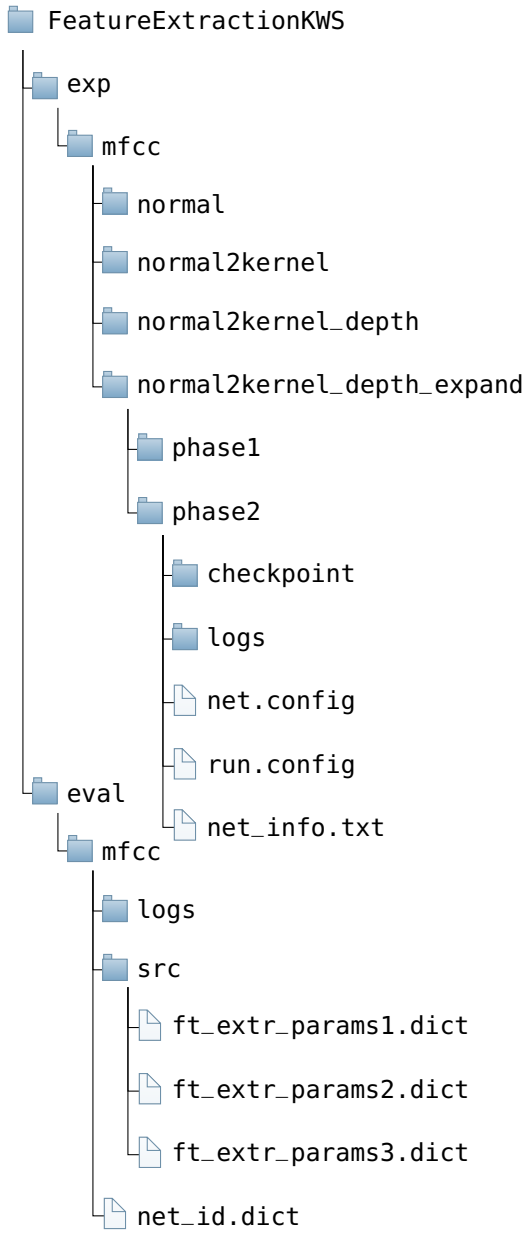
To conclude future work that could be beneficial for this topic ; it is essential to say that this feature extraction evaluation method can be applied to any other field or problem, also unrestricted to speech processing. Therefore, future work could replicate this usage of the once-for-all framework to evaluate feature extraction methods in other fields.

Appendix A

Project directory structure



Where `exp` and `eval`, contain the OFAKWSNet checkpoints and logs for the training and evaluation phases of the project respectively. They are structured as follows:



Appendix B

Code organisation

The code directory structure can be found in Appendix A.

B.1. OFA network

The code relating to the OFA model, as well as the network layers and operations is located in the files or directories described in Table B.1.

Content	File location
KWSNet	`once_for_all/networks/kws_net.py`
OFAKWSNet	`ofa/elastic_nn/networks/ofa_kws_net.py`
Utility functions and normal operations	`utils.py`
Dynamic operations and layers	`ofa/elastic_nn/modules`

Table B.1.: Location of each component related to the OFA model in the framework

B.2. Run management, training and evaluation

The code relating to the run handling, configurations, training and evaluation is located in the files or directories described in Table B.2.

Content	File location
Run Configuration	`ofa/run_manager/run_config.py`
Run Management	`ofa/run_manager/run_manager.py`
Dataset loading and data-augmentation	`ofa/run_manager/dataset.py`
Dataset and feature extraction handling	`ofa/run_manager/data_provider.py`
Training run	`train_ofa_net.py`
Evaluation run	`eval_ofa_net.py`

Table B.2.: Location of each component related to the run management in the framework

Appendix C

List of tested feature extraction parameters

This appendix describes a summary of all evaluated feature extraction methods, their best performing network accuracy in %, and their respective number of floating point operations (FLOPs) in MOps for these networks.

C. List of tested feature extraction parameters

Params id	MFCC bins	Mel filters	Window length [ms]	Top accuracy [%]	Top accuracy FLOPs
1	10	10	30	92.76	27.31
2	10	10	30	93.26	27.31
3	10	10	35	92.31	23.43
4	10	10	40	91.95	20.89
5	10	10	40	92.22	20.89
6	10	10	50	89.31	17.01
7	10	15	30	93.81	27.31
8	10	15	35	92.7	23.43
9	10	15	40	93.17	20.89
10	10	20	30	93.1	27.31
11	10	20	35	91.97	23.43
12	10	20	40	92.13	20.89
13	10	25	30	92.9	27.31
14	10	25	35	91.63	23.43
15	10	25	40	91.18	20.89
16	10	30	30	92.72	27.31
17	10	30	35	92.08	23.43
18	10	30	40	91.13	20.89
19	10	35	30	92.38	27.31
20	10	35	35	91.65	23.43
21	10	35	40	90.79	20.89
22	10	40	30	93.44	27.31
23	10	40	30	91.95	27.31
24	10	40	35	90.9	23.43
25	10	40	40	92.65	21.72
26	10	40	40	93.04	20.89
27	10	40	40	89.86	20.89
28	10	40	50	90.63	17.01
29	10	80	30	93.83	27.31
30	10	80	40	92.99	20.89
31	10	80	50	90.74	17.01
32	40	40	30	94.44	105.95
33	40	40	40	93.51	81.04
34	40	40	40	92.99	81.04
35	40	40	50	91.95	65.88
36	40	80	30	93.65	105.95
37	40	80	40	93.38	81.04
38	40	80	50	92.2	65.88
39	80	80	30	93.13	211.82
40	80	80	40	93.33	162.01
41	80	80	50	91.72	131.72

Table C.1.: Trained and evaluated feature extraction configurations for the MFCC feature extraction type

Params id	Mel filters	Window length [ms]	Top accuracy [%]	Top accuracy FLOPs
1	10	20	94.44	41.07
2	10	20	94.44	41.07
3	10	24	94.69	33.73
4	10	25	94.26	33.06
5	10	28	93.87	28.9
6	10	30	93.06	27.31
7	10	32	92.51	25.68
8	10	36	91.97	22.48
9	10	40	89.77	20.89
10	20	20	94.26	79.79
11	20	24	93.74	65.56
12	20	28	93.42	56.17
13	20	32	92.83	49.92
14	20	36	91.76	43.69
15	20	40	91.33	40.6
16	30	20	93.92	120.55
17	30	24	94.42	99.03
18	30	28	93.69	84.85
19	30	32	93.24	75.41
20	30	36	92.35	66.0
21	30	40	90.43	61.33
22	40	20	94.6	159.27
23	40	20	95.37	159.27
24	40	24	94.9	130.86
25	40	28	94.51	112.13
26	40	30	93.58	105.95
27	40	32	93.28	99.65
28	40	36	92.54	87.21
29	40	40	91.95	81.04
30	40	40	91.36	81.04

Table C.2.: Trained and evaluated feature extraction configurations for the Log Mel Spectrogram feature extraction type

Params id	Mel filters	Window length [ms]	Top accuracy [%]	Top accuracy FLOPs
1	10	20	88.77	41.07
2	10	25	88.73	33.06
3	10	30	87.23	27.31
4	40	20	93.9	159.27
5	40	30	89.66	105.95
6	40	40	84.35	81.04
7	80	20	93.74	318.44
8	80	30	91.9	211.82
9	80	40	88.43	162.01

Table C.3.: Trained and evaluated feature extraction configurations for the Mel Spectrogram feature extraction type

Params id	Number of FFTs	Hop length [ms]	Top accuracy [%]	Top accuracy FLOPs
1	400	20	92.35	409.72
2	400	30	88.52	270.24
3	400	40	83.8	207.17

Table C.4.: Trained and evaluated feature extraction configurations for the Log Spectrogram feature extraction type

Params id	Top accuracy [%]	Top accuracy FLOPs
1	84.32	628.99

Table C.5.: Trained and evaluated feature extraction configurations for raw input

Appendix D

Task Description

D.1. Short Description

The objective of keyword spotting (KWS) is to detect a set of predefined keywords within a stream of user utterances. For most of KWS pipelines, as well as any other audio-based task, the acoustic model and/or lingustic model is preceded by a feature extraction segment. Several approaches have been proposed to perform this step, the most notable one being the Mel Frequency Cepstral Coefficients (MFCC). As this technique represents only one of the possible approaches towards feature extraction, and since it has been showed that the performance of the system can be largely influenced by this choice, the aim of this project is to do an in-depth evaluation of feature extraction methods for KWS.

D.2. Introduction

Feature extraction represents the process of deriving essential, non-redundant information from a set of measured data. Through the selection and/or combination of input variables into features, the dimension of the given data can be largely reduced, thus also reducing the computational effort within the processing pipeline. Nonetheless, such techniques should not affect the integrity of the data; the extracted features should still accurately described the data set. Often used interchangeably, we differentiate between feature (pre)processing and feature extraction, defining the former as the techniques applied to alter the data in order to emphasize or remove certain characteristics. To better understand the concept of feature extraction, we will use the Mel-Frequency Cepstral Coefficients (MFCC) as an example. Extracted from an audio signal received as input by the system, the MFCCs are one of the current standards in KWS [2] [6]. They are a cepstral representation of the signal, but, compared to the simple cepstrum, the

MFCC use equally spaced frequency bands, based on the Mel scale. This leads to a closer representation of the signal to the actual response of the human auditory system. The derivation techniques are described in detail in [5]; we will enumerate the main steps of PyTorch's MFCC computation: Windowing - applying a window function (e.g., Hamming window) to each frame, mainly to counteract the infinite-data assumption made during the FFT computation; Fast Fourier Transform (FFT) - calculating the frequency spectrum; Triangular Filters - applying triangular filters on a Mel-scale to extract frequency bands; Logarithm - transforming the MelSpectrogram into LogMelSpectrogram to obtain additive behaviour; Discrete Cosine Transform (DCT) - calculating the Cepstral coefficients from the LogMel-Spectrogram.

Feature extraction is often used in Machine Learning settings, and especially in the con-text of Deep Neural Networks (DNN), due to the fact that reducing the feature dimensionality further reduces the computational load, as well as the storage requirements, of such a DNN. Therefore, by pre-determining the relevant features from the input data using human expertise, the model is alleviated from the tedious task of identifying and filtering redundant information. A deep learning pipeline for KWS integrating MFCC is presented in Figure 1.

In the recent years, it was shown that state-of-the-art results can be achieved not only by using the MFCCs, but also by employing the results of an intermediate step, such as the MelSpectrogram [9] or the LogMelSpectrogram [4]. Moreover, surveys [1] [7] [3] [8] on the topic present alternative strategies in performing feature extraction for audio-based ML tasks. Nevertheless, there are no such works, to the best of our knowledge, presenting a meta-analysis of the different extraction techniques and their impact in the scope of KWS. As Choi et al. [3] have shown, the choice of the feature extraction mechanism has a significant impact in the context of music tagging, so it is only natural to assume that a similar conclusion would be drawn when performing a comparative evaluation for a keyword spotting system.

D.3. Character

- 20% literature research
- 40% feature extraction implementation
- 40% evaluation

D.4. Prerequisites

Must be familiar with Python. Knowledge of deep learning basics, including some deep learning framework like PyTorch or TensorFlow from a course, project, or self-taught with some tutorials.

D.5. Project Goals

The main tasks of this project are:

1. **Task 1: Familiarize yourself with the project specifics (1-2 Weeks)**
   Read up on feature extraction methods mentioned in the reference materials. Learn about DNN training and PyTorch, how to visualize results with TensorBoard. Read up on DNN models aimed at time series (e.g. TCNs, RNNs, transformer networks) and the recent advances in KWS.
2. **Task 2: Implement and evaluate the baseline (2-3 Weeks)**
   Select a dataset and analyse the models which can represent baselines for our work. Particularly check for publicly available code. The supervisors will provide you with the MFCC implementation that will represent the starting point for the aforementioned analysis. If no code is available: design, implement, and train KWS models, considering the state-of-the-art architectures for time series. Compare the model against the selected baseline and figures in the paper.
3. **Task 3: Implement feature extraction techniques (4-5 Weeks)**
   Using the referenced work, implement the feature extraction methods against which MFCC will be compared. Optimize said methods with respect to computational effort and storage requirements. (Optional) Perform parameter tuning for the implemented techniques.
4. **Task 4: Evaluate feature extraction techniques (4-5 Weeks)**
   Evaluate and analyse the accuracy of the KWS pipeline with respect to the implemented feature extraction setting. Evaluate and analyse the accuracy of the aforementioned methods under similar operating regimes. Evaluate and analyse the compatibility between the processing techniques and the DNN architecture.
5. **Task 5 - Gather and Present Final Results (2-3 Weeks)**
   Gather final results. Prepare presentation (10 min. + 5 min. discussion). Write a final report. Include all major decisions taken during the design process and argue your choice. Include everything that deviates from the very standard case - show off everything that took time to figure out and all your ideas that have influenced the project.

D.6. Project Organization

D.6.1. Weekly Meetings

The student shall meet with the advisor(s) every week in order to discuss any issues/problems that may have persisted during the previous week and with a suggestion of next steps. These meetings are meant to provide a guaranteed time slot for mutual

exchange of information on how to proceed, clear out any questions from either side and to ensure the student's progress.

D.6.2. Report

Documentation is an important and often overlooked aspect of engineering. One final report has to be completed within this project. Any form of word processing software is allowed for writing the reports, nevertheless the use of LaTeX with Tgif (See: http://bourbon.usc.edu:8001/tgif/index.html and http://www.dz.ee.ethz.ch/en/information/how-to/drawing-schematics.html) or any other vector drawing software (for block diagrams) is strongly encouraged by the IIS staff.

Final Report

A digital copy of the report, the presentation, the developed software, build script/project files, drawings/illustrations, acquired data, etc. needs to be handed in at the end of the project. Note that this task description is part of your report and has to be attached to your final report.

Presentation

At the end of the project, the outcome of the thesis will be presented in a 15-minutes talk and 5 minutes of discussion in front of interested people of the Integrated Systems Laboratory. The presentation is open to the public, so you are welcome to invite interested friends. The exact date will be determined towards the end of the work.

List of Figures

List of Tables

Bibliography

[1] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," 2019. [Online]. Available: https://arxiv.org/abs/1908.09791

[2] B. Kim, S. Chang, J. Lee, and D. Sung, "Broadcasted residual learning for efficient keyword spotting," 2021. [Online]. Available: https://arxiv.org/abs/2106.04140

[3] A. Berg, M. O'Connor, and M. T. Cruz, "Keyword transformer: A self-attention model for keyword spotting," in *Interspeech 2021*. ISCA, aug 2021. [Online]. Available: https://doi.org/10.21437%2Finterspeech.2021-1286

[4] R. Vygon and N. Mikhaylovskiy, "Learning efficient representations for keyword spotting with triplet loss," in *Speech and Computer*. Springer International Publishing, 2021, pp. 773–785. [Online]. Available: https://doi.org/10.1007%2F978-3-030-87802-3_69

[5] S. Alim and N. K. Alang Md Rashid, *Some Commonly Used Speech Feature Extraction Algorithms*, 12 2018.

[6] "Mfcc implementation and tutorial: detailed mfcc derivation steps," https://www.kaggle.com/code/ilyamich/mfcc-implementation-and-tutorial/notebook, accessed: 2022-07-17.

[7] "Figures from: A basic introduction to separable convolutions," https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728, accessed: 2022-07-17.

[8] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," 2017. [Online]. Available: https://arxiv.org/abs/1711.07128

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019.

[10] A. Mehrotra, A. G. C. P. Ramos, S. Bhattacharya, Ł. Dudziak, R. Vipperla, T. Chau, M. S. Abdelfattah, S. Ishtiaq, and N. D. Lane, "{NAS}-bench-{asr}: Reproducible neural architecture search for speech recognition," in *International Conference on Learning Representations (ICLR)*, 2021.

[11] S. Majumdar and B. Ginsburg, "MatchboxNet: 1d time-channel separable convolutional neural network architecture for speech commands recognition," in *Interspeech 2020*. ISCA, oct 2020. [Online]. Available: https://doi.org/10.21437%2Finterspeech.2020-1058

[12] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, and S. Ha, "Temporal Convolution for Real-Time Keyword Spotting on Mobile Devices," in *Proc. Interspeech 2019*, 2019, pp. 3372–3376.

[13] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming keyword spotting on mobile devices," in *Interspeech 2020*. ISCA, oct 2020. [Online]. Available: https://doi.org/10.21437%2Finterspeech.2020-1003

[14] G. Cerutti, L. Cavigelli, R. Andri, M. Magno, E. Farella, and L. Benini, "Sub-mw keyword spotting on an mcu: Analog binary feature extraction and binary neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2201.03386

[15] H. Qin, X. Ma, Y. Ding, X. Li, Y. Zhang, Y. Tian, Z. Ma, J. Luo, and X. Liu, "Bifsmn: Binary neural network for keyword spotting," 2022. [Online]. Available: https://arxiv.org/abs/2202.06483

[16] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," 2017. [Online]. Available: http://arxiv.org/abs/1710.10361

[17] D. Peter, W. Roth, and F. Pernkopf, "End-to-end keyword spotting using neural architecture search and quantization," 2021. [Online]. Available: https://arxiv.org/abs/2104.06666

[18] E. Niels, "Feature extraction for speech recognition: Ofakws," https://github.com/NielsEscarfail/FeatureExtractionKWS, 2022.

[19] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018. [Online]. Available: https://arxiv.org/abs/1804.03209

[20] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," 2016. [Online]. Available: https://arxiv.org/abs/1608.03983

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," 2015.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[23] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," 2017. [Online]. Available: https://arxiv.org/abs/1709.01507