A depthwise separable convolutional neural network for keyword spotting on an embedded system

Sørensen, Peter Mølgaard; Epp, Bastian; May, Tobias

Published in:
Eurasip Journal on Audio, Speech, and Music Processing

Link to article, DOI:
10.1186/s13636-020-00176-2

Publication date:
2020

Document Version
Publisher's PDF, also known as Version of record

Sørensen *et al. EURASIP Journal on Audio, Speech, and Music Processing* (2020) 2020:10
https://doi.org/10.1186/s13636-020-00176-2

EURASIP Journal on Audio, Speech, and Music Processing

RESEARCH Open Access

A depthwise separable convolutional neural network for keyword spotting on an embedded system

Peter Mølgaard Sørensen, Bastian Epp and Tobias May*

Abstract

A keyword spotting algorithm implemented on an embedded system using a depthwise separable convolutional neural network classifier is reported. The proposed system was derived from a high-complexity system with the goal to reduce complexity and to increase efficiency. In order to meet the requirements set by hardware resource constraints, a limited hyper-parameter grid search was performed, which showed that network complexity could be drastically reduced with little effect on classification accuracy. It was furthermore found that quantization of pre-trained networks using mixed and dynamic fixed point principles could reduce the memory footprint and computational requirements without lowering classification accuracy. Data augmentation techniques were used to increase network robustness in unseen acoustic conditions by mixing training data with realistic noise recordings. Finally, the system's ability to detect keywords in a continuous audio stream was successfully demonstrated.

Keywords: Keyword spotting, Speech recognition, Embedded software, Deep learning, Convolutional neural networks, Quantization

1 Introduction

During the last decade, deep learning algorithms have continuously improved performances in a wide range of applications, among others automatic speech recognition (ASR) [1]. Enabled by this, voice-controlled devices constitute a growing part of the market for consumer electronics. Artificial intelligence (AI) digital assistants utilize natural speech as the primary user interface and often require access to cloud computation for the demanding processing tasks. However, such cloud-based solutions are impractical for many devices and cause user concerns due to the requirement of continuous internet access and due to concerns regarding privacy when transmitting audio continuously to the cloud [2]. In contrast to these large-vocabulary ASR systems, devices with more limited functionality could be more efficiently controlled using only a few speech commands, without the need of cloud processing.

Keyword spotting (KWS) is the task of detecting keywords or phrases in an audio stream. The detection of a keyword can then trigger a specific action of the device. Wake-word detection is a specific implementation of a KWS system where only a single word or phrase is detected which can then be used to, for example, trigger a second, more complex recognition system. Early popular KWS systems have typically been based on hidden Markov models (HMMs) [3–5]. In recent years, however, neural network-based systems have dominated the area and improved the accuracies of these systems. Popular architectures include standard feedforward deep neural networks (DNNs) [6–8] and recurrent neural networks (RNNs) [9–12]. Strongly inspired by advancements in techniques used in computer vision (e.g., image classification and facial recognition), the convolutional neural network (CNN) [13] has recently gained popularity for KWS in small memory footprint applications [14]. The

*Correspondence: tobmay@dtu.dk
Center for Applied Hearing Research, Technical University of Denmark, Ørsteds Plads, Lyngby, Denmark

depthwise separable convolutional neural network (DS-CNN) [15, 16] was proposed as an efficient alternative to the standard CNN. The DS-CNN decomposes the standard 3-D convolution into 2-D convolutions followed by 1-D convolutions, which drastically reduces the number of required weights and computations. In a comparison of multiple neural network architectures for KWS on embedded platforms, the DS-CNN was found to be the best performing architecture [17].

For speech recognition and KWS, the most commonly used speech features are the mel-frequency cepstral coefficients (MFCCs) [17–20]. In recent years, there has, however, been a tendency to use mel-frequency spectral coefficients (MFSCs) directly with neural network-based speech recognition systems [6, 14, 21] instead of applying the discrete cosine transform (DCT) to obtain MFCCs. This is mainly because the strong correlations between adjacent time-frequency components of speech signals can be exploited efficiently by neural network architectures such as the CNN [22, 23]. An important property of MFSC features is that they attenuate the characteristics of the acoustic signal irrelevant to the spoken content, such as the intonation or accent [24].

One of the major challenges of supervised learning algorithms is the ability to generalize from training data to unseen observations [25]. Reducing the impact of speaker variability on the input features can make it easier for the network to generalize. Another way to improve the generalization is to ensure a high diversity of the training data, which can be realized by augmenting the training data. For audio data, augmentation techniques include filtering [26], time shifting and time warping [27], and adding background noise. However, the majority of KWS systems either have used artificial noises, such as white or pink noise, which are not relevant for real-life applications or have considered only a limited number of background noises [14, 17, 28].

Because of the limited complexity of KWS compared to large-vocabulary ASR, low-power embedded microprocessor systems are suitable targets for running real-time KWS without access to cloud computing [17]. Implementing neural networks on microprocessors presents two major challenges in terms of the limited resources of the platform: (1) memory capacity to store weights, activations, input/output, and the network structure itself is very limited for microprocessors; (2) computational power on microprocessors is limited. The number of computations per network inference is therefore limited by the real-time requirements of the KWS system. To meet these strict resource constraints, the size of the networks must be restricted in order to reduce the number of network parameters. Techniques like quantization can further be used to reduce the computational load and memory footprint. The training and inference of neural networks is typically done using floating-point precision for weights and layer outputs, but for implementation on mobile devices or embedded platforms, fixed point formats at low bit widths are often more efficient. Many microprocessors support single instruction, multiple data (SIMD) instructions, which perform arithmetic on multiple data points simultaneously, but typically only for 8/16 bit integers. Using low bit width representations will therefore increase the throughput and thus lower the execution time of network inference. Previous research has shown that, for image classification tasks, it is possible to quantize CNN weights and activations to 8-bit fixed point format with a minimum loss of accuracy [29, 30]. However, the impact of quantization on the performance of a DS-CNN-based KWS system has not yet been investigated.

This paper extends previous efforts [17] to implement a KWS system based on a DS-CNN by (a) identifying performance-critical elements in the system when scaling the network complexity, (b) augmenting training data with a wider variety of realistic noise recordings and by using a controlled range of signal-to-noise ratios (SNRs) that are realistic for practical KWS applications during both training and testing. Moreover, the ability of the KWS system to generalize to unseen acoustic conditions was tested by evaluating the system performance in both matched and mismatched background noise conditions, (c) evaluating the effect of quantizing individual network elements and (d) evaluating the small-footprint KWS system on a continuous audio stream rather than single inferences. Specifically, the paper reports the implementation of a 10-word KWS system based on a DS-CNN classifier on a low-power embedded microprocessor (ARM Cortex M4), motivated by the system in [17]. The KWS system described in the present study is targeted at real-time applications, which can be either always on or only active when triggered by an external system, e.g., a wake-word system. To quantify the network complexity where the performance decreases relative to the system in [17], we tested a wide range of system parameters between 2 layers and 10 filters per layer up to 9 layers and 300 filters per layer. The network was trained with keywords augmented with realistic background noises at a wide range of SNRs and the network's ability to generalize to unseen acoustic conditions was evaluated. With the goal to reduce the memory footprint of the system, it was investigated how quantization of weights and activations affected performance by gradually lowering the bit widths using principles of mixed and dynamic fixed point representations. In this process, single-inference performance was evaluated, motivated by the smaller parameter space and the close connection between the performance in single-inference testing and continuous audio presentation. In the final step, the performance of the suggested

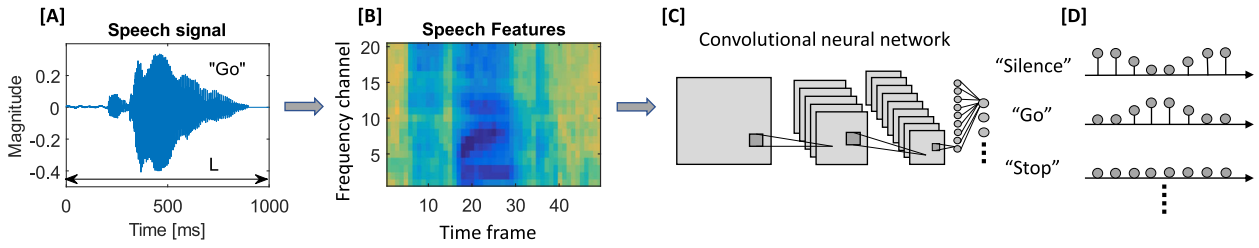

Fig. 1 Overview of the different stages in the KWS system. (A) A 1000-ms window of input signal. (B) Speech features are extracted from the input signal. (C) DS-CNN classifier generates probabilities of output classes. (D) Probabilities are combined in a posterior handling stage

KWS system was tested when detecting keywords in a continuous audio stream and compared to the reference system of high complexity.

2 System

2.1 KWS system

The proposed DS-CNN-based KWS system consisted of three major building blocks as shown in Fig. 1. First, MFSC features were extracted based on short time blocks of the raw input signal stream (pre-processing stage). These MFSC features were then fed to the DS-CNN-based classifier, which generated probabilities for each of the output classes in individual time blocks. Finally, a posterior handling stage combined probabilities across time blocks to improve the confidence of the detection. Each of the three building blocks is described in detail in the following subsections.

2.2 Feature extraction

The MFSC extraction consisted of three major steps, as shown in Fig. 2. The input signal was sampled at a rate of 16 kHz and processed by the feature extraction stage in blocks of 1000 ms. For each block, the short-time discrete Fourier transform (STFT) was computed by using a Hann window of 40-ms duration with 50 % overlap, giving a total of 49 frames. Each frame was zero-padded to a length of 1024 samples before computing a 1024-point discrete Fourier transform (DFT). Afterwards, a filterbank with 20 triangular bandpass filters with a constant Q-factor spaced equidistantly on the mel-scale between 20 and 4000 Hz [31] was applied. The mel-frequency band energies were then logarithmically compressed, producing the MFSC features, resulting in a 2-D feature matrix of size 20×49 for each inference. The number of log-mel features was derived from initial investigations on a few selected network configurations where it was found that 20 log-mel features proved most efficient in terms of performance vs the resources used.

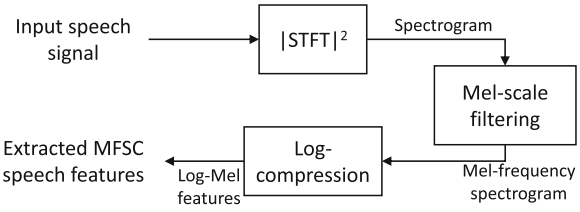

Fig. 2 Stages of MFSC feature extraction used as input to the DS-CNN classifier. A spectrogram was computed based on the audio signal. From the spectrogram, a mel-frequency spectrogram with log-compression was derived to produce MFSC features

2.3 DS-CNN classifier

The classifier had one output class for each of the keywords it should detect. It furthermore had an output class for *unknown* speech signals and one for signals containing *silence*. The input to the network was a 2-dimensional feature map consisting of the extracted MFSC features. Each convolutional layer of the network then applied a number of filters, N_{filters}, to detect local time-frequency patterns across input channels. The output of each network inference was a probability vector, containing the probability for each output class. The general architecture of the DS-CNN classifier is shown in Fig. 3. The first layer of the network was in all cases a standard convolutional layer. Following the convolutional layer was a batch-normalization layer with a rectified linear unit (ReLU) [32] activation function.

Batch normalization [33] was employed to accelerate training and to reduce the risk of overfitting through regularization. By equalizing the distributions of activations, higher learning rates can be used because the magnitude of the gradients of each layer is more similar, which results in faster model convergence. Because the activations of a single audio file are not normalized by the mean and variance of each audio file, but instead by the mean and variance of the mini-batch [32] in which it appears, a regularization effect is created by the random selection of audio files in the mini-batch. The batch-normalization layer was followed by a number of depthwise separable convolutions (DS-convs) [16], which each consisted of a depthwise convolution (DW-conv) and pointwise convolution (PW-conv) as illustrated in Fig. 4, both followed by a

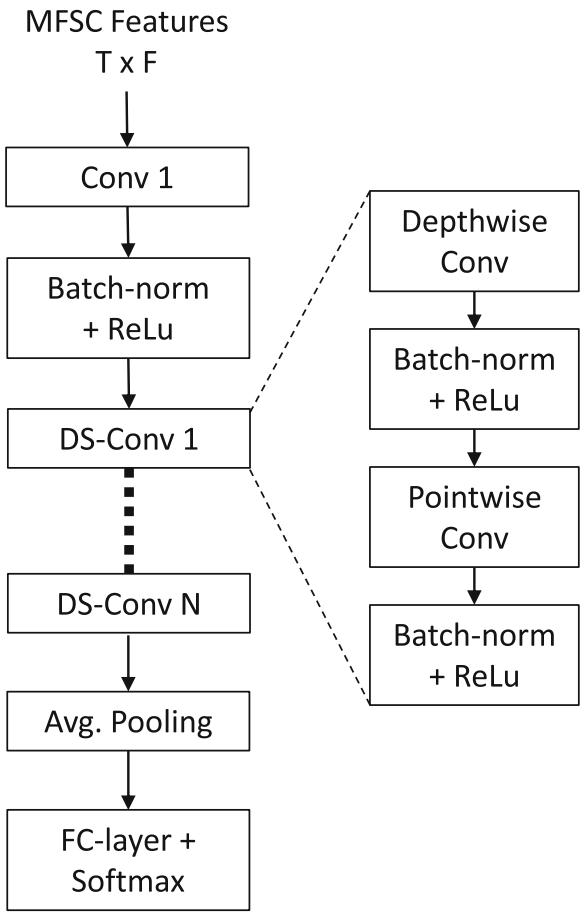

Fig. 3 General architecture of the DS-CNN. The input feature map is passed onto the first convolution layer, followed by batch normalization and ReLU activations. The following DS-convolution layers 1-N each consist of a depthwise convolution, followed by batch-normalization and ReLU activation, passed on to a pointwise convolution and another batch normalization and ReLU activation. At the end of the convolutional layers, the output undergoes average pooling and a fully connected (FC) layer with softmax activations

batch-normalization layer with ReLU activation. An average pooling layer then reduced the number of activations by applying an averaging window to the entire time-frequency feature map of each input channel. Finally, a fully connected (FC) layer with softmax [32] activations generated the probabilities for each output class.

2.4 Posterior handling

The classifier was run 4 times per second, resulting in a 250-ms shift and an overlap of 75 %. As the selected keywords were quite short in duration, they typically appeared in full length in multiple input blocks. In order to increase the confidence of the classification an integration period, $T_{\text{integrate}}$, was introduced, in which the predicted output probabilities of each output class were averaged. The system then detected a keyword if any of these averaged probabilities exceeded a predetermined detection threshold. To avoid that the same word would trigger multiple detections by the system, a refractory period, $T_{\text{refractory}}$, was introduced. When the system detected a keyword, it would be suppressed from detecting the same keyword during the refractory period. For this paper, an integration period of $T_{\text{integrate}} = 750$ ms and a refractory period of $T_{\text{refractory}} = 1000$ ms were used.

3 Methods

3.1 Dataset

The *Speech Commands* dataset [34] was used for training and evaluation of the networks. The dataset consisted of 65000 single-speaker, single-word recordings of 30 different words. A total of 1881 speakers contributed to the dataset, ensuring a high speaker diversity. The following 10 words were used as keywords: {"Yes," "No," "Up," "Down," "Left," "Right," "On," "Off," "Go," "Stop"}. The remaining 20 words of the dataset were used to train the category "unknown." The dataset was split into "training," "validation," and "test" sets with the ratio 80 : 10 : 10, while restricting recordings of the same speaker to only

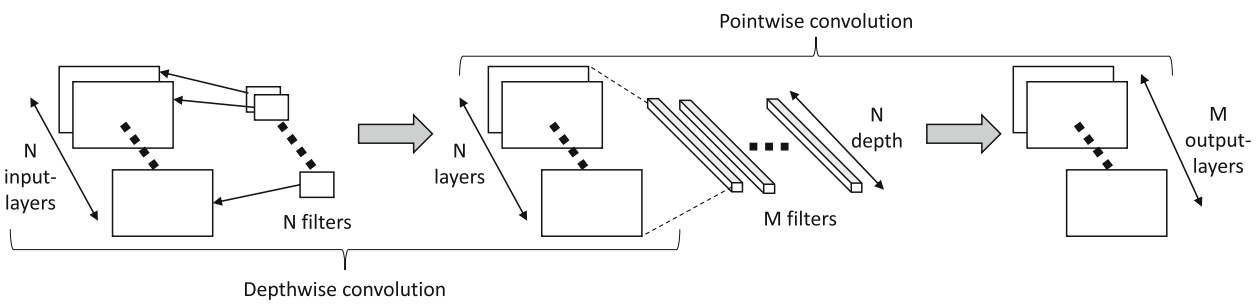

Fig. 4 Overview of a single depthwise separable convolutional layer consisting of a depthwise convolution followed by a pointwise convolution. (1) The depthwise convolution separately applies a 2-dimensional filter to each of the channels in the input, extracting time-frequency patterns. (2) The pointwise convolution then applies a number of 1-dimensional filters to the output of the depthwise convolution across all channels

appear in one of the three sets. For training, 10 % of the presented audio files were labeled *silence*, i.e., containing no spoken word; 10 % were *unknown* words; and the remaining contained keywords.

3.2 Data augmentation

For training, validation, and testing, the speech files were mixed with 13 diverse background noises at a wide range of SNRs. The background noise signals were real-world recordings, some containing speech, obtained from two publicly available databases, the TUT database [35] and the DEMAND database [36]. The noise signals were split into two sets, reflecting matched and mismatched conditions (see Table 1). The networks were either trained on the clean speech files or trained on speech files mixed with noise signals from noise set 1 with uniformly distributed A-weighted SNRs in the range between 0 and 15 dB. To add background noise to the speech files, the filtering and noise adding tool (FaNT) [37] was used. Noise set 2 was then used to evaluate the network performance in acoustic conditions that were not included in the training. Separate recordings of each noise type were used for training and evaluation.

3.3 Resource estimation

To compare the resources used by different network configurations, the following definitions were used to estimate number of operations, memory, and execution time.

3.3.1 Operations

The number of operations are per inference of the network, defined as the total number of multiplications and additions in the convolutional layers of the DS-CNN.

3.3.2 Memory

The memory reported is the total memory required to store the network weights/biases and layer activations, assuming 8-bit variables. As the activations of one layer are only used as input for the next layer, the memory for the activations can be reused. The total memory allocated for activations is then equal to the maximum of the required memory for inputs and outputs of a single layer.

3.3.3 Execution time

The execution times reported in this paper are estimations based on measured execution times of multiple different-sized networks. The actual network inference execution time of implemented DS-CNNs on the Cortex M4 was measured using the Cortex M4's on-chip timers, with the processor running at a clock frequency of 180 MHz. In this study, only two hyper-parameters were altered: the number of DS-conv layers, N_{layers}, and the number of filters applied per layer, N_{filters}. The number of layers was varied between 2 and 9, and the number of filters per layer was varied between 10 and 300. Convolutional layers after layer 7 had the same parameters as seen in the last layers in Table 2 in terms of filter size and strides.

Table 1 Matched and mismatched noise sets

Noise set 1 (matched)	Noise set 2 (mismatched)
Exercise bike, running tap, bus, cafe, car, city center, grocery store, metro station, office, park, miaowing, train, tram	Dish washing, beach, forest path, home, library, residential area, sports field, river, living room, office meeting, office hallway, public cafeteria, traffic

The specific recordings were taken from [35] and [36]

3.4 Quantization methods

The fixed point format represents floating-point numbers as N-bit 2's complement signed integers, where the B_I leftmost bits (including the sign-bit) represent the integer part, and the remaining B_F rightmost bits represent the fractional part. The following two main concepts were applied when quantizing a pre-trained neural network effectively [29].

3.4.1 Mixed fixed point precision

The fully connected and convolutional layers of a DS-CNN consist of a long series of multiply-and-accumulate (MAC) operations, where network weights multiplied with layer activations are accumulated to give the output. Using different bit widths for different parts of the network, i.e., mixed precision, has been shown to be an effective approach when quantizing CNNs [38], as the precision required to avoid performance degradation may vary in different parts of the network.

3.4.2 Dynamic fixed point

The weights and activations of different CNN layers will have different dynamic ranges. The fixed point format requires that the range of the values to represent is known beforehand, as this determines B_I and B_F. To ensure a high utilization of the fixed point range, *dynamic fixed point* [39] can be used, which assigns the weights/activations into groups of constant B_I.

For faster inference, the batch-norm operations were fused into the weights of the preceding convolutional layer

Table 2 Default hyper-parameters for the test network investigated in experiments

Layer Op.	N_{filters}	Filter dim. ($W_t \times W_f$)	Stride_t	Stride_f
Conv.	76	10 × 4	2	1
DS-conv.	76	3 × 3	2	2
DS-conv.	76	3 × 3	1	1
DS-conv.	76	3 × 3	1	1
DS-conv.	76	3 × 3	1	1
DS-conv.	76	3 × 3	1	1
DS-conv.	76	3 × 3	1	1

and quantized after this fusion. B_I and B_F were determined by splitting the network variables for each layer into groups of weights, biases, and activations, and estimating the dynamic range of each group. The dynamic ranges of groups with weights and biases were fixed after training, while the ranges of activations were estimated by running inference on a large number of representative audio files from the dataset and generating statistical parameters for the activations of each layer. B_I and B_F were then chosen such that saturation is avoided. The optimal bit widths were determined by dividing the variables in the network into separate categories based on the operation, while the layer activations were kept as one category. The effects on performance were then examined when reducing the bit width of a single category while keeping the rest of the network at floating-point precision. The precision of the weights and activations in the network was varied in experiment 3 between 32-bit floating-point precision and low bit width fixed point formats ranging from 8 to 2 bit.

3.5 Training

All networks were trained with Google's TensorFlow machine learning framework [40] using an Adam optimizer to minimize the cross-entropy loss. The networks were trained in 30,000 iterations with a batch size of 100. Similar to [17], an initial learning rate of 0.0005 was used; after 10,000 iterations, it was reduced to 0.0001; and for the remaining 10,000 iterations, it was reduced to 0.00002. During training, audio files were randomly shifted in time up to 100 ms to reduce the risk of overfitting.

3.6 Evaluation

To evaluate the DS-CNN classifier performance in the presence of background noise, test sets with different SNRs between −5 and 30 dB were used. Separate test sets were created for noise signals from noise set 1 and noise set 2. The system was tested by presenting single inferences (single-inference testing) to evaluate the performance of the network in isolation. In addition, the system was tested by presenting a continuous audio stream (continuous-stream testing) to approximate a more realistic application environment.

3.6.1 Single-inference testing

For single-inference testing, the system was tested without the posterior handling stage. For each inference, the maximum output probability was selected as the detected output class and compared to the label of the input signal. When testing, 10 % of the samples were *silence*, 10 % were *unknown* words, and the remaining contained keywords. Each test set consisted of 3081 audio files, and the reported test accuracy specified the ratio of correctly labeled audio files to the total amount of audio files in the test. To compare different network configurations, the accuracy was averaged across the range 0 − 20 dB SNR, as this reflects SNRs in realistic conditions [41].

3.6.2 Continuous audio stream testing

Test signals with a duration of 1000 s were created for each SNR and noise set, with words from the dataset appearing approximately every 3 s. Seventy percent of the words in the test signal were keywords. The test signals were constructed with a high ratio of keywords to reflect the use case in which the KWS system is not run in an always-on state but instead triggered externally by, e.g., a wake-word detector. A hit was counted if the system detected the keyword within 750 ms after occurrence, and the hit rate (also called true positive rate (TPR)) then corresponds to the number of hits relative to the total number of keywords in the test signal. The false alarm rate (also called false positive rate (FPR)) reported is the total number of incorrect keyword detections relative to the duration of the test signal, here reported as false alarms per hour.

3.7 Network test configuration

Unless stated otherwise, the parameters summarized in Table 2 are used. The network had 7 convolutional layers with 76 filters for each layer.

As a baseline for comparison, a high-complexity network was introduced. The baseline network had 8 convolutional layers with 300 filters for each layer with hyper-parameters as summarized in Table 3. The baseline network was trained using the noise-augmented dataset and evaluated using floating-point precision weights and activations.

3.8 Platform description

Table 4 shows the key specifications for the FRDM K66F development platform used for verification of the designed systems. The deployed network used 8-bit weights and activations, but performed feature extraction using 32-bit floating-point precision. The network was implemented using the CMSIS-NN library [42] which

Table 3 Default hyper-parameters for the baseline network

Layer Op.	N_{filters}	Filter dim. ($W_t \times W_f$)	Stride_t	Stride_f
Conv.	300	10 × 4	2	1
DS-conv.	300	3 × 3	2	2
DS-conv.	300	3 × 3	1	1
DS-conv.	300	3 × 3	1	1
DS-conv.	300	3 × 3	1	1
DS-conv.	300	3 × 3	1	1
DS-conv.	300	3 × 3	1	1
DS-conv.	300	3 × 3	1	1

Table 4 FRDM K66F key specs

Processor	Arm Cortex M4 32-bit core, with floating-point unit
Architecture	Armv7E-M Harvard
DSP extensions	Single cycle 16/32-bit MAC Single cycle dual 16-bit MAC
Max. CPU Frequency	180 MHz
SRAM	256 KB
Flash	2 MB

features neural network operations optimized for Cortex-M processors.

4 Results

In experiment 1, the effect of training the network on noise-augmented speech on single-inference accuracy was investigated. The influence of network complexity was assessed in experiment 2 by systematically varying the number of convolutional layers and the number of filters per layer. Experiment 3 investigated the effects on performance when quantizing network weights and activations for fixed point implementation. Finally, the best performing network was tested on a continuous audio stream and the impact of the detection threshold on hit rate and false positive rate was evaluated.

4.1 Experiment 1: Data augmentation

Figure 5 shows the single-inference accuracies when using noise-augmented speech files for training. For SNRs below 20 dB, the network trained on noisy data had a higher test accuracy than the network trained on clean data, while the accuracy was slightly lower for SNRs higher than 20 dB. In the range between −5 and 5 dB SNR, the average accuracy for the network trained on noisy data was increased by 11.1 % and 8.6% for the matched and mismatched noise sets respectively relative to the training on clean data. Under clean test conditions, it was found that the classification accuracy of the network trained on noisy data was 4 % lower than the network trained on clean data. For both networks, there was a difference between the accuracy on the matched and mismatched test. The average difference in accuracy in the range from −5 to 20 dB SNR was 3.3 % and 4.4 % for the network trained on clean and noisy data, respectively. The high-complexity baseline network performed on average 3 % better than the test network trained on noisy data.

4.2 Experiment 2: Network complexity

Figure 6 shows a selection of the most feasible networks for different numbers of layers and numbers of filters per layer. For each trained network, the table specifies single-inference average accuracy in the range 0 − 20 dB SNR for both test sets (accuracy in parentheses for the mismatched test). Moreover, the number of operations per inference, the memory required by the model for weights/activations, and the estimated execution time per inference on the Cortex M4 are specified. For networks with more than 5 layers, no significant improvement (<1 %) was obtained when increasing the number of filters beyond 125. Networks with less than 5 layers gained larger improvements from using more than 125 filters, though none of those networks reached the accuracies obtained with networks with more layers.

Figure 7 shows the accuracies of all the layer/filter combinations of the hyper-parameter search as a function of the operations. For the complex network structures, the deviation of the accuracies was very small, while for

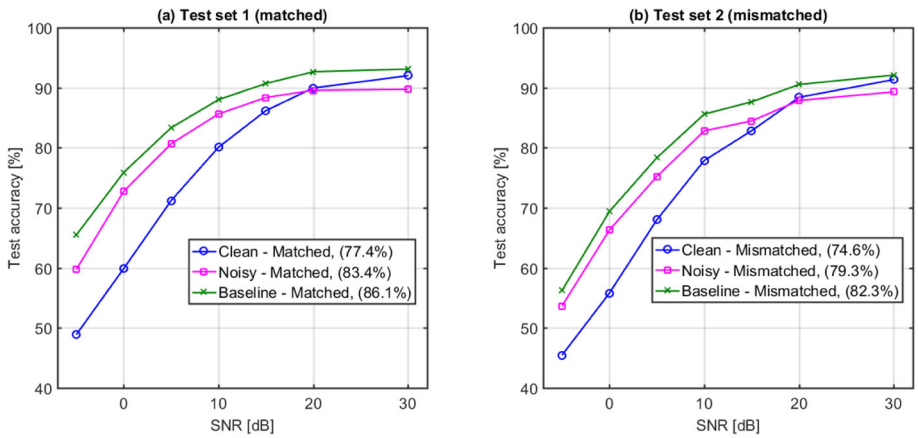

Fig. 5 Single-inference test accuracy of networks trained on clean and noisy speech. Networks were tested in both noise set 1 (matched) and noise set 2 (mismatched). The number in parentheses specifies the average accuracy in the range from 0 to 20 dB SNR. The performance of the baseline model trained on noisy data is shown for comparison

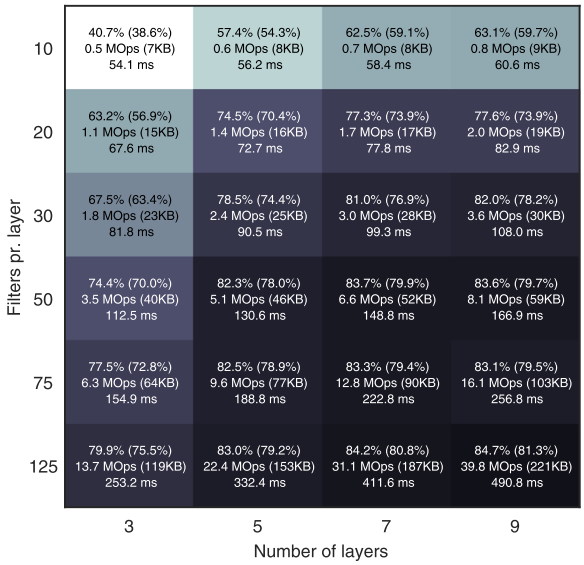

Fig. 6 Selection of hyper-parameter grid search. The first line in each box indicates the average accuracy in the range of 0 − 20 dB SNR for the matched and mismatched (in brackets) test. The number in the second line shows the number of operations and the required memory (in brackets). The third line indicates the execution time. The color indicates the accuracy (lighter color lower accuracy)

networks using few operations, there was a large difference in accuracy depending on the specific combination of layers and filters. For networks ranging between 5 and 200 million operations, the difference in classification accuracy between the best performing models was less than 2.5 %. Depending on the configuration of the network, it is therefore possible to drastically reduce the number of operations while maintaining a high classification accuracy.

In Fig. 8, a selection of the best performing networks is shown as a function of required memory and operations per inference. The label for each network specifies the parameter configuration $[N_{\text{layers}}, N_{\text{filters}}]$ and the average accuracy in 0 − 20 dB SNR for noise set 1 and 2. The figure illustrates the achievable performance given the platform resources and shows that high accuracy was reached with relatively simple networks. From this investigation, it was found that the best performing network

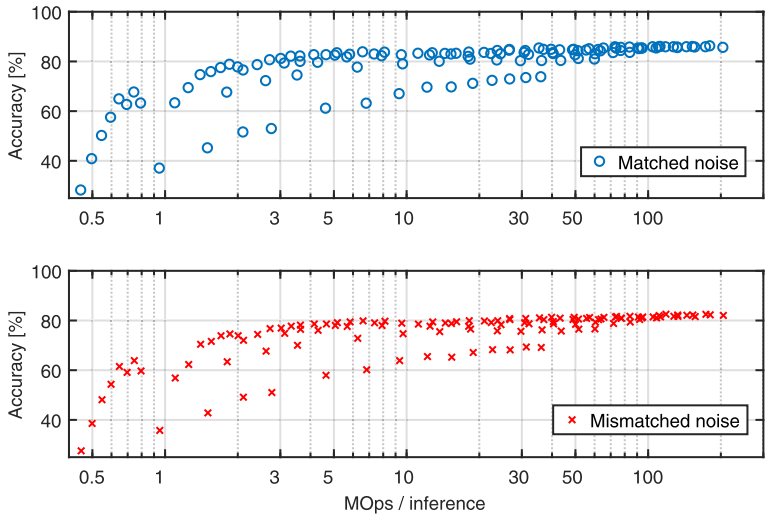

Fig. 7 Average accuracy in the range between 0 and 20 dB SNR of all trained networks in hyper-parameter search as a function of the number of operations per inference

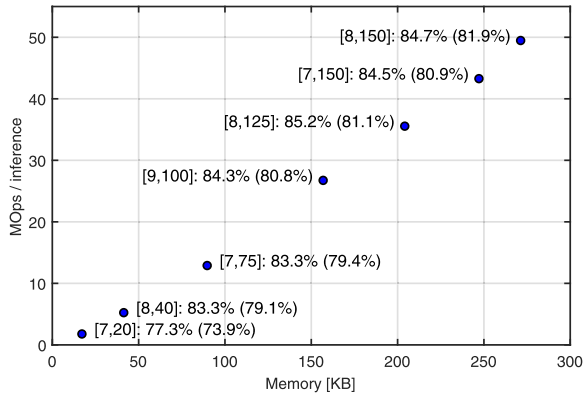

Fig. 8 Best performing networks of grid search for different memory and computational requirements

fitting the resource requirements of the platform consisted of 7 DS-CNN layers with 76 filters per layer, as described in Section 3.7.

4.3 Experiment 3: Quantization

Table 5 shows the single-inference test results of the quantized networks, where each part of the network specified in Section 3.7 was quantized separately, while the remainder of the network was kept at floating-point precision.

All of the weights and activations could be quantized to 8-bit using dynamic fixed point representation with no loss of classification accuracy, and the bit widths of the weights could be further reduced to 4 bits with only small reductions in accuracy. In contrast, reducing the bit width of activations to less than 8 bits significantly reduced classification accuracy. While the classification accuracy was substantially reduced when using only 2 bits for regular convolution parameters and FC parameters, the performance completely broke down when quantizing pointwise and depthwise convolution parameters and layer activations with 2 bits. The average test accuracy in the range of $0-20$ dB SNR of the network with all weights and activations quantized to 8 bits was 83.2 % for test set 1 (matched) and 79.2 % for test set 2 (mismatched), which was the same performance as using floating-point precision.

Table 5 Single-inference accuracies of networks with individually quantized network parts

Floating point accuracy 83.2 % (79.3 %)			
Bit width	8-bit	4-bit	2-bit
Convolution weights	83.2 % (79.1 %)	82.3 % (78.8 %)	46.8 % (44.6 %)
DW-Convolution weigths	83.3 % (79.3 %)	80.0 % (76.7 %)	12.0 % (12.1 %)
PW-Convolution weights	83.4 % (79.2 %)	78.9 % (75.4 %)	11.2 % (10.8 %)
Fully connected weights	83.2 % (79.3 %)	82.9 % (79.1 %)	70.1 % (63.9 %)
Layer activations	83.2 % (79.2 %)	53.6 % (52.0 %)	8.6 % (8.7 %)

Using 8-bit fixed point numbers instead of 32-bit floating point reduced the required memory by a factor of 4, from 366 to 92 KB, with 48 KB reserved for activations and 44 KB for storing weights. Utilizing mixed fixed point precision and quantizing activations to 8 bits and weights to 4 bits would reduce the required memory to 70 KB.

4.4 Experiment 4: Continuous audio stream

Figure 9 shows the hit rate and false positive rate obtained by the KWS system on the continuous audio signals. The system was tested using different detection thresholds, which affected the system's inclination towards detecting a keyword. It was found that the difference in hit rates was constant as a function of SNR when the detection threshold was altered, while the difference in false positive rates increased towards low SNRs. For both test sets, the hit rate and false positive rate saturated at SNRs higher than 15 dB. Figure 10 shows the corresponding DET curve obtained for the test network and baseline network.

4.5 FRDM K66F implementation

Table 6 shows the distribution of execution time over network layers for a single inference for the implementation on the FRDM K66F development board. The total execution time of the network inference was 227.4 ms , which leaves sufficient time for feature extraction and audio input handling, assuming 4 inferences per second.

5 Discussion

Experiment 1 showed that adding noise to the training material increased the classifier robustness in low SNR conditions. The increase in accuracy, compared to the same network trained on clean speech files, was most significant for the matched noise test, where the test data featured the same noise types as the training material. For the mismatched test, the increase in accuracy was slightly smaller. A larger difference in performance between clean and noisy training was expected, but as explained in [43], the dataset used was inherently noisy and featured invalid audio files, which could diminish the effect of adding more noise. For both test sets, the network trained on clean data performed better under high SNRs, i.e., SNR $>$ 20 dB. From the perspective of this paper however, the performance in high SNRs was of less interest as the focus was on real-world application. If the performance in clean conditions is also of concern, [28] demonstrated that the performance decrease in clean conditions could be reduced by including clean audio files in the noisy training. As was also found in [28], it was observed that the noisy training enabled the network to adapt to the noise signals and improve the generalization ability, by forcing it to detect patterns more unique to the keywords. Even though the two noise sets consisted of different noise environment recordings, many of the basic noise types, such

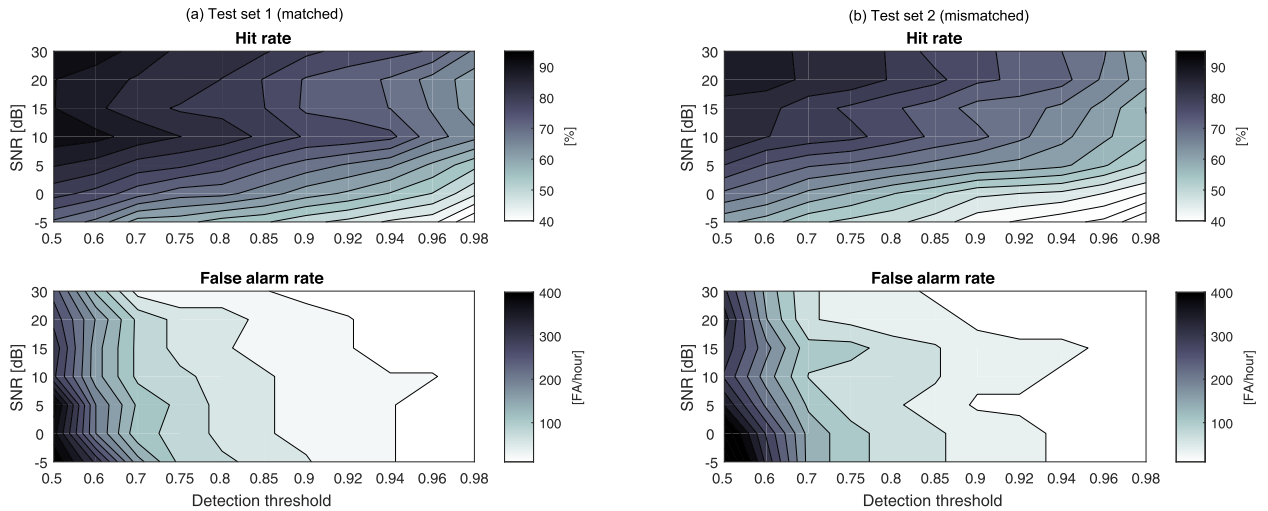

Fig. 9 Continuous audio stream test results. Hit rate and false alarm rates are shown as a function of SNR and detection threshold in **a** matched noise conditions (noise set 1) and **b** mismatched noise conditions (noise set 2). The trade-off between the hit rate and the false alarm rate can be controlled by selection of the detection threshold

as speech, motor noise, or running water, were present in both noise sets. This would explain why, that even though the network was only trained on data mixed with noise set 1 (matched), it also performed better on test set 2 (mismatched) than the network trained on clean data.

The main result of experiment 2 was that the classification accuracy as a function of network complexity reached a saturation point. Increasing the number of layers or the number of filters per layer beyond this point only resulted in negligible accuracy gains, $< 2\,\%$. This was explicitly shown in Fig. 5 for the single-inference classification accuracy and Fig. 10 for continuous audio streaming, where the high-complexity baseline network was directly compared with the smaller network chosen for the implementation. It was also found that, given a fixed computational and memory constraint, higher accuracies were achieved by networks with many layers and few filters than by networks with few layers and many

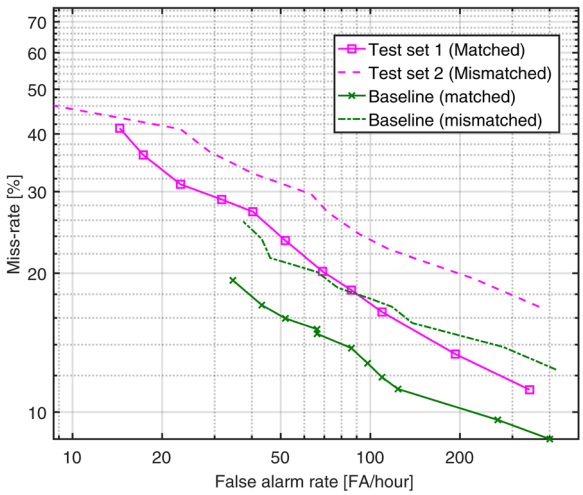

Fig. 10 Continuous audio stream test results showing the mean hit rate and false alarm rate in the range 0 — 20 dB SNR. Results are shown for the test network and the baseline network in matched and mismatched noise conditions. The test network was quantized to 8-bit activations and weights while the baseline network used floating-point precision

Table 6 Layer distribution of execution time and operations for classifier inference

Layer	Execution time [ms]	Millions of operations
Conv 1	99.0 (43.5%)	3.04 (23.2%)
DW-conv 1	8.3 (3.7%)	0.17 (1.4%)
PW-conv 1	13.0 (5.7%)	1.50 (11.5%)
DW-conv 2	8.3 (3.7%)	0.17 (1.4%)
PW-conv 2	13.0 (5.7%)	1.50 (11.5%)
DW-conv 3	8.3 (3.7%)	0.17 (1.4%)
PW-conv 3	13.0 (5.7%)	1.50 (11.5%)
DW-conv 4	8.3 (3.7%)	0.17 (1.4%)
PW-conv 4	13.0 (5.7%)	1.50 (11.5%)
DW-conv 5	8.3 (3.7%)	0.17 (1.4%)
PW-conv 5	13.0 (5.7%)	1.50 (11.5%)
DW-conv 6	8.3 (3.7%)	0.17 (1.4%)
PW-conv 6	13.0 (5.7%)	1.50 (11.5%)
Avg-pool	0.5 (0.2%)	0.01 (0.1%)
FC	0.1 (0.04%)	0.001 (< 0.1%)
Total	227.4	13.12

filters. In a convolutional layer, the number of filters determines how many different patterns can be detected in the input features. The first layer detects characteristic patterns of the input speech features, and each subsequent convolutional layer will detect patterns in the patterns detected by the previous layer, adding another level of abstraction. One interpretation of the grid-search results could therefore be, that if the network has sufficient levels of abstraction (layers), then the number of distinct patterns needed at each abstraction level to characterize the spoken content (number of filters) can be quite low. As the classifier should run 4 times per second, feasible network configurations were limited to inference execution times below 250 ms , which ruled out the majority of the configurations tested. In terms of the resource constraints set by the platform, execution time was the limiting factor for these networks and not the memory required for weights and activations. This was not unexpected as the DS-CNN, contrary to network architectures such as the DNN, reuses the same weights (filters) for computing multiple neurons. The DS-CNN therefore needs fewer weights relative to the number of computations it must perform, making this approach especially suitable for platforms with very limited memory capacity.

The results from experiment 3 showed that weights and activations of a network trained using floating-point precision could be quantized to low bit widths without affecting the classification accuracy. Quantizing all numbers in the network to 8 bit resulted in the same classification accuracy as using floating-point precision. It was also found that the weights of the network could all be quantized to 4 bit with no substantial loss of accuracy, which can significantly reduce the memory footprint and possibly reduce the processing time spent on fetching data from memory. These results showed that mixed fixed point precision leads to the most memory-efficient network, because the different network components (weights and activations) are robust to different reductions in bit width. For many deep CNN classifiers [29, 30, 38], it was reported that networks are very robust to the reduced resolution caused by quantization. The reason for this robustness could be that the networks are designed and trained to ignore the background noise and the deviations of the speech samples. The quantization errors are then simply another noise source for the network, which it can handle up to a certain magnitude. Gysel et al. [29] found that small accuracy decreases of CNN classifiers, caused by fixed point quantization of weights and activations, could be compensated for by partially retraining the networks using these fixed point weights and activations. A natural next step for the KWS system proposed in this paper would therefore also be to fine tune the quantized networks. Because the network variables were categorized, the quantization effects on the overall performance could be evaluated individually for each category. Results showed that different bit widths were required for the different categories, in order to maintain the classification accuracy achieved using floating-point numbers. It is however suspected that, because some of the categories span multiple network layers, a bottleneck effect could occur. For example, if the activations of a single layer require high precision, i.e., large bit width, but the other layers' activations required fewer bits, this would be masked in the experiment because they were all in the same category. It is therefore expected that using different bit widths for each of the layers in each of the categories would potentially result in a lower memory footprint. In this paper, the fixed point representations had symmetric, zero-centered ranges. However, all of the convolutional layers use ReLU activations functions, so the activations effectively only utilize half of the available range as values below zero are cutoff. By shifting the range, such that zero becomes the minimum value, the total range can be halved, i.e., B_I is decreased by one. This in turn frees a bit, which could be used to increase B_F by one, thereby increasing the resolution, or it could be used to reduce the total bit width by one.

Experiment 4 tested the KWS system performance on a continuous audio stream. As found in most signal detection tasks, lowering the decision criterion, i.e., the detection threshold, increases the hit rate but also the FPR, which means there is a trade-off. The detection threshold should match the intended application of the system. For always-on systems, it is crucial to keep the number of false alarms as low as possible, while for externally activated systems where the KWS is only active for a short time window in which a keyword is expected, a higher hit rate is more desirable. One method for lowering the FPR and increasing the true negative rate could be to increase the ratio of negative to positive samples in the training, i.e., use more "unknown" and "silence" samples. This has been shown as an effective method in other machine learning detection tasks [44, 45]. Another approach for lowering the FPR could be to create a loss function for the optimizer during training, which penalizes errors that cause false alarms more than errors that cause misses. There were significant discrepancies between the estimated number of operations and actual execution time of the different layers of the implemented network (see Table 6). The convolutional functions in the software library used for the implementation [42] all use highly optimized matrix multiplications (i.e. general matrix-matrix multiplication, GEMM) to compute the convolution. However, in order to compute 2D convolutions using matrix multiplications, it is necessary to first rearrange the input data and weights during run time. It was argued that, despite this time consuming and memory

expanding data reordering, using matrix multiplications is still the most efficient implementation of convolutional layers [46, 47]. The discrepancies between operations and execution time could be explained by the fact that the reordering of data was not accounted for in the operation parameter and that the different layers required different degrees of reordering. For the pointwise-convolutions and fully connected layer, the activations were stored in memory in an order such that no reordering was required to do the matrix multiplication, whereas this was not possible for the standard convolution or depthwise convolution. The number of arithmetic operations for running network inference should therefore not solely be used to asses the feasibility of implementing neural networks on embedded processors, as done in [17], as this parameter does not directly reflect the execution time. Instead, this estimate should also include the additional work for data reordering required by some network layers. Based on the results presented in this paper, there are several possible actions to take to improve performance or optimize implementation of the proposed KWS system. Increasing the size of the dataset and removing corrupt recordings, or augmenting training data with more varied background noises, such as music, could increase network accuracy and generalization. Reducing the number of weights of a trained network using techniques such as pruning [48] could be used to further reduce memory footprint and execution time.

Python training scripts and FRDM K66F deployment source code as well as a quantitative comparison of performance for using MFCC vs MFSC features for a subset of networks are available on Github [49].

6 Conclusion

In this paper, methods for training and implementing a DS-CNN-based KWS system for low-resource embedded platforms were presented and evaluated. Experimental results showed that augmenting training data with realistic noise recordings increased the classification accuracy in both matched and mismatched noise conditions. By performing a limited hyper-parameter grid search, it was found that network accuracy saturated when increasing the number of layers and filters in the DS-CNN and that feasible networks for implementation on the ARM Cortex M4 processor were in this saturated region. It was also shown that using dynamic fixed point representations allowed network weights and activations to be quantized to 8-bit precision with no loss in accuracy. By quantizing different network components individually, it was found that layer activations were most sensitive to further quantization, while weights could be quantized to 4 bits with only small decreases in accuracy. The ability of the KWS system to detect keywords in a continuous audio stream was tested, and it was seen how altering the detection threshold affected the hit rate and false alarm rate. Finally, the system was verified by the implementation on the Cortex M4, where it was found that the number of arithmetic operations per inference are not directly related to execution time. Ultimately, this paper shows that the number of layers and the number of filters per layers provide a useful parameter when scaling system complexity. In addition, it was shown that a 8-bit quantization provides a significant reduction in memory footprint and processing time and does not result in a loss of accuracy.

Abbreviations

AI: Artificial intelligence; ASR: Automatic speech recognition; CNN: Convolutional neural network; DCT: Discrete cosine transform; DFT: Discrete Fourier transform; DNN: Deep neural network; DS-CNN: Depthwise separable convolutional neural network; DS-conv: Depthwise separable convolution; DW-conv: Depthwise convolution; FaNT: Filtering and noise adding tool; FC: Fully connected; FPR: False positive rate; HMM: Hidden Markov model; KWS: Keyword spotting; MAC: Multiply and accumulate; MFCC: Mel-frequency cepstral coefficients; MFSC: Mel-frequency spectral coefficients; PW-conv: Pointwise convolution; ReLU: Rectified linear unit; RNN: Recurrent neural network; SIMD: Singe instruction, multiple data; SNR: Signal-to-noise ratio; STFT: Short-time discrete Fourier transform; TPR: True positive rate

Acknowledgements

This research was supported by the Centre for Applied Hearing Research (CAHR). We would like to thank Andreas Krebs for the helpful input regarding the implementation on the hardware target and fruitful discussions.

Authors' contributions

PMS carried out the numerical experiments. PMS, BE, and TM participated in the design of the study and drafted and revised the manuscript. The authors read and approved the final manuscript.

Funding

This research was supported by the Centre for Applied Hearing Research (CAHR).

Availability of data and materials

The data that support the findings of this study are available from [34].

Competing interests

The authors declare that they have no competing interests.

Received: 8 July 2019 Accepted: 10 May 2020
Published online: 25 June 2020

References

1. G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012). https://doi.org/10.1109/MSP.2012.2205597
2. New Electronic Friends. https://pages.arm.com/machine-learning-voice-recognition-report.html. Accessed 30 May 2018
3. R. C. Rose, D. B. Paul, in *International Conference on Acoustics, Speech, and Signal Processing*. A hidden Markov model based keyword recognition system, (1990), pp. 129–1321. https://doi.org/10.1109/ICASSP.1990.115555
4. J. R. Rohlicek, W. Russell, S. Roukos, H. Gish, in *International Conference on Acoustics, Speech, and Signal Processing,*. Continuous hidden Markov modeling for speaker-independent word spotting, (1989), pp. 627–6301. https://doi.org/10.1109/ICASSP.1989.266505

5. J. G. Wilpon, L. G. Miller, P. Modi, in *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*. Improvements and applications for key word recognition using hidden Markov modeling techniques, (1991), pp. 309–312. https://doi.org/10.1109/ICASSP.1991.150338. http://ieeexplore.ieee.org/document/150338/
6. G. Chen, C. Parada, G. Heigold. Small-footprint keyword spotting using deep neural networks, (2014). https://doi.org/10.1109/icassp.2014.6854370
7. K. Shen, M. Cai, W.-Q. Zhang, Y. Tian, J. Liu, Investigation of DNN-based keyword spotting in low resource environments. Int. J. Future Comput. Commun. **5**(2), 125–129 (2016). https://doi.org/10.18178/ijfcc.2016.5.2.458
8. G. Tucker, M. Wu, M. Sun, S. Panchapagesan, G. Fu, S. Vitaladevuni. Model compression applied to small-footprint keyword spotting, (2016), pp. 1878–1882. https://doi.org/10.21437/Interspeech.2016-1393
9. S. Fernández, A. Graves, J. Schmidhuber, in *Artificial Neural Networks – ICANN 2007*, ed. by J. M. de Sá, L. A. Alexandre, W. Duch, and D. Mandic. An application of recurrent neural networks to discriminative keyword spotting (Springer, Berlin, Heidelberg, 2007), pp. 220–229
10. K. P. Li, J. A. Naylor, M. L. Rossen, in *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 2*. A whole word recurrent neural network for keyword spotting, (1992), pp. 81–842. https://doi.org/10.1109/ICASSP.1992.226115
11. M. Sun, A. Raju, G. Tucker, S. Panchapagesan, G. Fu, A. Mandal, S. Matsoukas, N. Strom, S. Vitaladevuni, Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting. CoRR. **abs/1705.02411** (2017). 1705.02411
12. S. Ö,.. Arik, M. Kliegl, R. Child, J. Hestness, A. Gibiansky, C. Fougner, R. Prenger, A. Coates, Convolutional recurrent neural networks for small-footprint keyword spotting. CoRR. **abs/1703.05390** (2017). 1703.05390
13. Y. LeCun, Y. Bengio, in *Chap. Convolutional Networks for Images, Speech, and Time Series*. The Handbook of Brain Theory and Neural Networks (Press, MIT, Cambridge, MA, USA, 1998), pp. 255–258. http://dl.acm.org/citation.cfm?id=303568.303704
14. T. N. Sainath, C. Parada, in *INTERSPEECH*. Convolutional neural networks for small-footprint keyword spotting, (2015)
15. F. Chollet, Xception: deep learning with depthwise separable convolutions. CoRR. **abs/1610.02357** (2016). 1610.02357
16. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: efficient convolutional neural networks for mobile vision applications. CoRR. **abs/1704.04861** (2017). 1704.04861
17. Y. Zhang, N. Suda, L. Lai, V. Chandra, Hello edge: keyword spotting on microcontrollers. CoRR. **abs/1711.07128** (2017). 1711.07128
18. S. Davis, P. Mermelstein, Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE Trans. Acoust. Speech Signal Process. **28**(4), 357–366 (1980). https://doi.org/10.1109/TASSP.1980.1163420
19. I. Chadawan, S. Siwat, Y. Thaweesak, in *International Conference on Computer Graphics, Simulation and Modeling (ICGSM'2012)*. Speech recognition using MFCC, (Pattaya (Thailand), 2012)
20. Bhadragiri Jagan Mohan, Ramesh Babu N., in *2014 International Conference on Advances in Electrical Engineering (ICAEE)*. Speech recognition using MFCC and DTW, (2014), pp. 1–4. https://doi.org/10.1109/ICAEE.2014.6838564
21. O. Abdel-Hamid, A. Mohamed, H. Jiang, L. Deng, G. Penn, D. Yu, Convolutional neural networks for speech recognition. IEEE/ACM Trans. Audio Speech Lang. Process. **22**(10), 1533–1545 (2014). https://doi.org/10.1109/TASLP.2014.2339736
22. A.-R. Mohamed, *Deep Neural Network acoustic models for ASR. PhD thesis*. (University of Toronto, 2014). https://tspace.library.utoronto.ca/bitstream/1807/44123/1/Mohamed_Abdel-rahman_201406_PhD_thesis.pdf
23. S. Watanabe, M. Delcroix, F. Metze, J. R. Hershey, in *Springer International Publishing*. New era for robust speech recognition, (2017), p. 205. https://doi.org/10.1007/978-3-319-64680-0
24. J. W. Picone, Signal modeling techniques in speech recognition. Proc. IEEE. **81**, 1215–1247 (1993). https://doi.org/10.1109/5.237532
25. X. Xiao, J. Li, Chng. E.S., H. Li, C.-H. Lee, A study on the generalization capability of acoustic models for robust speech recognition. IEEE Trans. Audio Speech Lang. Process. **18**(6), 1158–1169 (2010). https://doi.org/10.1109/TASL.2009.2031236
26. I. Rebai, Y. BenAyed, W. Mahdi, J.-P. Lorré, Improving speech recognition using data augmentation and acoustic model fusion. Procedia Comput. Sci. **112**, 316–322 (2017). https://doi.org/10.1016/J.PROCS.2017.08.003
27. T. Ko, V. Peddinti, D. Povey, S. Khudanpur, in *INTERSPEECH*. Audio augmentation for speech recognition, (2015)
28. S. Yin, C. Liu, Z. Zhang, Y. Lin, D. Wang, J. Tejedor, T. F. Zheng, Y. Li, Noisy training for deep neural networks in speech recognition. EURASIP J. Audio Speech Music Process. **2015**(1), 2 (2015). https://doi.org/10.1186/s13636-014-0047-0
29. P. Gysel, M. Motamedi, S. Ghiasi, Hardware-oriented approximation of convolutional neural networks. CoRR. **abs/1604.03168** (2016). 1604.03168
30. D. D. Lin, S. S. Talathi, V. S. Annapureddy, Fixed point quantization of deep convolutional networks. CoRR. **abs/1511.06393** (2015). 1511.06393
31. D. O'Shaughnessy, *Speech Communication: Human and Machine*, (1987), p. 150
32. M. A. Nielsen, *Neural Networks and Deep Learning*, (2015). http://neuralnetworksanddeeplearning.com/. Accessed 26 May 2020
33. S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift. CoRR. **abs/1502.03167** (2015). 1502.03167
34. P. Warden, Speech commands: a public dataset for single-word speech recognition (2017). Dataset available from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz
35. A. Mesaros, T. Heittola, T. Virtanen, in *2016 24th European Signal Processing Conference (EUSIPCO)*. TUT database for acoustic scene classification and sound event detection, (2016), pp. 1128–1132. https://doi.org/10.1109/EUSIPCO.2016.7760424
36. J. Thiemann, N. Ito, E. Vincent, DEMAND: a collection of multi-channel recordings of acoustic noise in diverse environments. Supported by Inria under the Associate Team Program VERSAMUS (2013). https://doi.org/10.5281/zenodo.1227121
37. H.-G. Hirsch, FaNT -filtering and noise adding tool. Technical report. Hochschule Niederrhein (2005). http://dnt.kr.hs-niederrhein.de/download/fant_manual.pdf. Accessed 26 May 2020
38. N. Mellempudi, A. Kundu, D. Das, D. Mudigere, B. Kaul, Mixed low-precision deep learning inference using dynamic fixed point. CoRR. **abs/1701.08978** (2017). 1701.08978
39. D. Williamson, in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference Proceedings*. Dynamically scaled fixed point arithmetic (IEEE, 1991), pp. 315–318. https://doi.org/10.1109/PACRIM.1991.160742. http://ieeexplore.ieee.org/document/160742/
40. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: large-scale machine learning on heterogeneous systems. Software available from tensorflow.org (2015). https://www.tensorflow.org/. Accessed 26 May 2020
41. K. Smeds, F. Wolters, M. Rung, Estimation of signal-to-noise ratios in realistic sound scenarios. J. Am. Acad. Audiol. **26 2**, 183–96 (2015)
42. L. Lai, N. Suda, V. Chandra, CMSIS-NN: efficient neural network kernels for arm cortex-M CPUS. CoRR. **abs/1801.06601** (2018). 1801.06601
43. P. Warden, Speech commands: a dataset for limited-vocabulary speech recognition. CoRR. **abs/1804.03209** (2018). http://arxiv.org/abs/1804.03209
44. Z. Cheng, K. Huang, Y. Wang, H. Liu, J. Guan, S. Zhou, Selecting high-quality negative samples for effectively predicting protein-RNA interactions. BMC Syst. Biol. **11**(2), 9 (2017). https://doi.org/10.1186/s12918-017-0390-8
45. R. Kurczab, S. Smusz, A. J. Bojarski, The influence of negative training set size on machine learning-based virtual screening,. J Cheminformatics. **6**, 32 (2014). https://doi.org/10.1186/1758-2946-6-32
46. P. Warden, Why GEMM is at the heart of deep learning. https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/. Accessed 19 May 2018

47. S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, E. Shelhamer, cudnn: efficient primitives for deep learning. CoRR. **abs/1410.0759** (2014). 1410.0759
48. P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient transfer learning. CoRR. **abs/1611.06440** (2016). 1611.06440
49. P. M. Sørensen, A depthwise separable convolutional neural network for keyword spotting on embedded systems. GitHub (2018). https://github.com/PeterMS123/KWS-DS-CNN-for-embedded

Publisher's Note