A Depthwise Separable Convolutional Neural Network for Keyword Spotting on an Embedded System

Peter Mølgaard Sørensen

***Abstract*—This study investigated the feasibility of implementing keyword spotting algorithms on embedded systems using a depthwise separable convolutional neural network classifier. The proposed system consisted of 3 stages: (1) a feature extraction stage, (2) a classifier and (3) a posterior handling stage. Data augmentation techniques were used to increase network robustness in unseen acoustic conditions by mixing training data with realistic noise recordings. 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 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 test accuracy. The systems ability to detect keywords in an audio stream was tested, and it was seen that the systems performance could be fitted to match the application by altering the detection threshold.**

***Index Terms*—Keyword spotting, speech recognition, embedded software, deep learning, convolutional neural networks, quantization**

I. Introduction

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

Keyword spotting (KWS) is the task of detecting a small set of predefined words (keywords) in an audio stream. The detection of a keyword can then trigger a specific action of the device. Early popular KWS-systems have typically been based on hidden Markov models (HMMs) [2]–[4]. 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 network (DNN) [5] [6] and recurrent neural network (RNN) [7] [8]. Strongly inspired by advances in techniques used in computer vision (image classification, facial recognition...), the convolutional neural network (CNN) [9] has recently gained popularity for KWS in small memory footprint applications [10]. The depthwise separable convolutional neural network (DS-CNN) [11] 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 weights and computations. [12] compared multiple neural network architectures for KWS on embedded platforms and found the DS-CNN to be the best performing architecture.

For speech recognition and KWS, the most commonly used speech features are the mel-frequency cepstral coefficients (MFCCs) [13]–[15]. Many of the stages of the MFCC feature extraction seeks to mimic the effective signal processing performed by the auditory periphery, as this has been seen to generate relevant features for characterizing spoken content [16]. The last stage of the MFCC-extraction is applying the discrete cosine transform (DCT) to the mel-frequency spectral coefficients (MFSC). A major motivation for applying the DCT has been to reduce the input dimensionality in order to limit the computations for the model. This was an important factor in earlier HMM- or DNN-based models when processing power was more expensive. In recent years there have, however, been a tendency to use MFSC instead of MFCC with neural-network based speech recognition systems [5] [10]. 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 [17].
An important property of these features should be to attenuate the irregularities of the acoustic signal irrelevant to the spoken content, such as the intonation or accent. For supervised learning algorithms, one of the major challenges is the network's ability to generalize from training data to unseen observations [18], and reducing the impact of speaker variability on the input features can make it easier for the network to generalize. Another method for improving the generalization is to ensure a high diversity of the training data.

Data augmentation is a common method used in a variety of deep learning applications in order to increase the amount of data, reduce risk of overfitting and increase network robustness towards unseen acoustic conditions. For audio

data, augmentation techniques include filtering [19], time shifting and time warping [20] and adding background noise signals. Common for many of these KWS systems, where data is mixed with background noise, is that the diversity of the background noises is quite low, using only a few different noise types [12] [10], or single noise types [21] including artificial noise such as white noise and pink noise, which is not relevant in real-life applications.

Because of the limited scope of KWS compared to large-vocabulary ASR, low-power embedded microprocessor systems are suitable targets for running offline real-time KWS [12]. Implementing neural networks on microprocessors presents two major challenges in terms of the limited resources of the platform. There are strict network memory footprint constraints, as memory capacity is very limited for microprocessors The memory footprint includes weights, activations, input/output and the network structure itself. Computational constraints must also be considered, as computational power on microprocessors is limited. The number of computations per network inference are 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, while techniques like quantization can further 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 with strict memory and processing constraints it is desirable to use fixed-point formats at low bit-widths. Many microprocessors supports single instruction, multiple data (SIMD) instructions, which performs 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 [22] [23].

The purpose of this study was to asses the feasibility of implementing a KWS-system based on a DS-CNN classifier on a low-power embedded microprocessor (ARM Cortex M4).
In this context, the effects on the performance of different methods for training and implementing neural networks were investigated. The effects of training the network on data augmented with realistic background noises on the networks ability to generalize to unseen acoustic conditions were evaluated. Furthermore, it was investigated how classification accuracy was affected by varying the complexity of the network, and by the quantization of weights and activations to low bitwidths.
Finally, the KWS-systems performance when detecting keywords in a continuous audio stream was tested.

II. System

A. KWS system overview

The proposed DS-CNN-based KWS system is shown in Fig. 1 and consists of 3 major building blocks. First, MFSC features were extracted from a short time frame of the raw input signal in a pre-processing stage. These speech features were then fed to the DS-CNN based classifier, which generated probabilities for each of the output classes in the current time frame. Finally, a posterior-handling stage combined probabilities across frames to improve the confidence of the detection.

B. Feature extraction

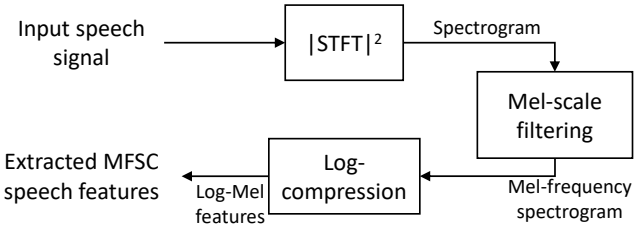


Fig. 2. Stages of MFSC feature extraction used as input to the DS-CNN clasifier.

The MFSC extraction consists of the following steps, as shown in Fig. 2:
1) The spectrogram of the input speech signal is obtained using the short-time discrete Fourier transform (STFT).
2) A filterbank of triangular band-pass-filters with a constant Q-factor spaced equidistantly on the Mel-scale [24] are applied.
3) The Mel-frequency band energies are then logarithmically compressed, to resemble perceived loudness. These features constitute the MFSC.

The duration of the input signal to the feature extraction stage was $1000\,\mathrm{ms}$, sampled at $16\,\mathrm{kHz}$. The spectrogram was computed using overlapping frames of $40\,\mathrm{ms}$ duration with a $20\,\mathrm{ms}$ frame shift resulting in a total of, $T = \frac{1000\,\mathrm{ms} - 40\,\mathrm{ms}}{20\,\mathrm{ms}} + 1 = 49$, time frames. Each frame was cut with a Hann-window, and zero-padded to a length of 1024 frames.
The Mel-filterbank consisted of 20 band-pass filters in the frequency range $20 - 4000\,\mathrm{Hz}$.

For each inference of the classifier, the size of the extracted input features were then, $T \times F = 49 \times 20$.[1]

C. DS-CNN Classifier

The task of the DS-CNN classifier was to predict which of the output classes the input signal belonged to. The classifier had an 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

[1]See appendix A, B, C for additional investigations on input features

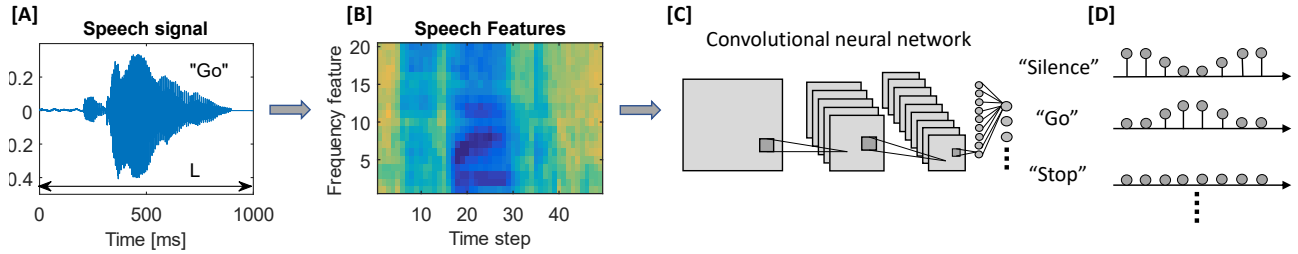


Fig. 1. Overview of stages in KWS system. [A] 1000 ms window of input signal. [B] Speech features are extracted from the input signal. [B] DS-CNN classifier generates probabilities of output classes. [C] Probabilities are combined in a posterior handling stage.

to the network was a 2-dimensional feature map of size $T \times F$, representing the extracted speech T-F features. Each convolutional layer of the network then applied a number of filters, $N_{filters}$, which can 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 that the input signal belonged to it.

Figure 3 shows the general architecture of the DS-CNN

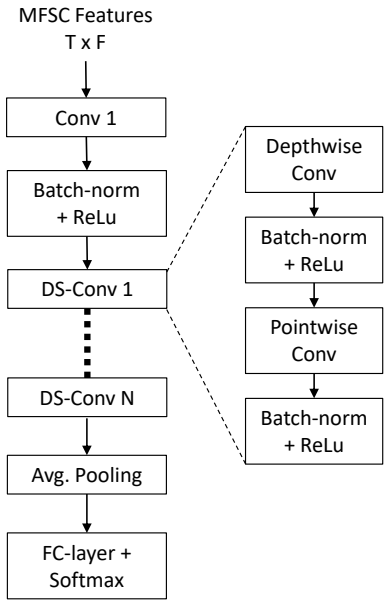


Fig. 3. General architecture of the DS-CNN.

models explored in this paper. 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) [25] activation function.

Batch normalization [26] is a training mechanism that seeks to reduce the internal covariate shift during training by normalizing the layer activations. The two main benefits of this is that it accelerates training, and adds regularization to reduce overfitting. By equalizing the distributions of activations, higher learning rates can be used because the magnitude of the gradients of each layer are more similar, which results in faster training. Activations are not normalized by the mean and variance of each sample, but instead by the mean and variance of the mini-batch [25] in which it appears. Because the activations of a single input sample are affected by the randomly selected samples in the mini-batch, a regularization effect is created.

The batch-normalization layers was followed by a number of depthwise separable convolutions (DS-convs), which each consisted of a depthwise convolution (DW-conv) and pointwise convolution (PW-conv), both followed by a batch-normalization layer with ReLU activation. An average pooling layer then reduced the number of activations significantly by applying an averaging window to the entire time-frequency feature map of each input channel. Finally, a fully connected (FC) layer with softmax [25] activations generated the probabilities for each output class. The softmax activations ensured that the sum of the output probabilities was always 1.

D. Posterior handling

The posterior handling of the KWS-system functioned as the decision stage of the system, as it decided when a keyword was detected and what keyword it was, based on predefined rules.

The classifier ran 4 times per second, meaning a 250 ms shift between the 1000 ms input frames giving an input frame overlap of 75%. As the selected keywords were quite short in duration, they typically appeared in full length in multiple input frames. So to increase the confidence of the decision-basis, the predicted output probabilities for each output class were averaged over 2 frames. The system then detected a keyword if any of these averaged probabilities exceeded a predetermined detection threshold. [2]

III. Methods

A. Dataset

The *Speech Commands* dataset [27] was used for training and evaluation of the networks. The dataset consisted of approximately 65000 single-speaker, single-word recordings of people saying 30 different words with high speaker variability The following 10 words were used as keywords: {'Yes', 'No',

[2]See appendix D for investigations of posterior handling parameters

'Up', 'Down', 'Left', 'Right', 'On', 'Off', 'Go', 'Stop}. The remaining 20 words of the dataset was 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 appear in one of the three sets.

B. Data augmentation (multi-conditional training)

For training, validation and testing, the speech samples were mixed with background noise. The background noise signals were real-world recordings, some including speech. Table I shows the noise signals used as background noise for the speech samples. The noise signals were split into two sets, each consisting of 13 diverse noise environments.
The networks evaluated in the present study were either trained on the clean speech samples, or trained on samples mixed with noise signals from noise set 1 with uniformly distributed A-weighted SNRs in the range 0-15 dB. To add background noise to the speech samples, Filtering and noise adding tool (FaNT) [28] was used. Noise set 2 was then used to evaluate the networks performance in acoustic conditions that were not included in the training.
Separate recordings of each noise type were used for training and evaluation.

TABLE I
NOISE SETS. SPECIFIC RECORDINGS ARE FROM [29] AND [30]

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

C. 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.

1) Operations: The number of operations reported in this paper is per inference of the network, and is defined as the total number of multiplications and additions in the convolutional layers of the DS-CNN.

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 is 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 maximum of the required memory for inputs and outputs of a single layer.

3) Execution time: The execution times reported in this paper are estimations based on measured execution times of two different-sized networks. The actual network inference execution time of implemented DS-CNNs on the Cortex M4 were measured using the Cortex M4's on-chip timers, with the processor running at a 180 MHz clock frequency.

This paper investigated how network complexity affects the classification performance by altering network hyper-parameters. This study was limited to altering two hyper-parameters: 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 in different steps. Convolutional layers after layer 7 had the same parameters as seen in the last layers in table II in terms of filter size and strides.

D. Quantization Methods

When quantizing the network weights and activations, the fixed-point format was used, which 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.
As explored in [22] the following two main concepts were applied when quantizing a pre-trained neural network effectively.

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, ie. mixed precision, has been shown to be an effective approach when quantizing CNNs [31], as the precision required to avoid performance-degradation may vary in different parts of the network.

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* [32] 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. The weights were then 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 samples 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 bitwidths 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 bitwidth of a single category while keeping the rest of the network at floating point precision.

E. Training

All networks were trained with Google's TensorFlow machine learning framework [33] using an Adam optimizer to minimize the cross-entropy loss. The networks were trained in 30000 iterations with a batch size of 100. An initial learning rate of 0.0005 was used, after 10000 iterations it was reduced to 0.0001, and for the remaining 10000 iterations was reduced to 0.00002. During training, input samples were randomly shifted in time up to 100 ms.

F. Evaluation

To evaluate the DS-CNN classifiers performance in the presence of background noise, test sets with different signal-to-noise ratios (SNRs) were created. FaNT [28] was used to create these test sets by adding noise signals to the test samples at fixed A-weighted SNRs between -5 dB and 30 dB SNR. 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.

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 presented samples were labelled *silence*, ie. containing no spoken word, 10% were *unknown*-words and the remaining contained keywords.
Each test set consisted of 3081 samples, and the reported test accuracy specified the ratio of correctly labelled samples to the total amount of samples in the test.
To easily compare different network configurations the average accuracy in the range $0-20\,\mathrm{dB}$ SNR was used, as this reflects realistic conditions [34].

2) Continuous audio stream testing: Test signals with a duration of $1000s$ were created for each SNR and noise set, with words from the dataset appearing approximately every 3 seconds. 70% of the words in the test signal were keywords. A hit was counted if the system detected the keyword within 750ms after occurrence. The false-positive-rate (FPR) reported is the total number of incorrect keyword detections relative to the total number of times the system makes a decision (4000 in total).

G. Network test configuration

Unless otherwise stated, the parameters summarized in table II were used. The network had 7 convolutional layers with 76 filters for each layer.

TABLE II
DEFAULT HYPER PARAMETERS FOR NETWORKS 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

H. Platform description

Table III shows the key specifications for the FRDM K66F development platform used for verification of the designed systems.

TABLE III
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

IV. RESULTS

Experiment 1: Data augmentation

In this experiment, the effects on single-inference accuracy of training the network on noise-augmented speech samples were investigated. Figure 4 shows the test accuracies.

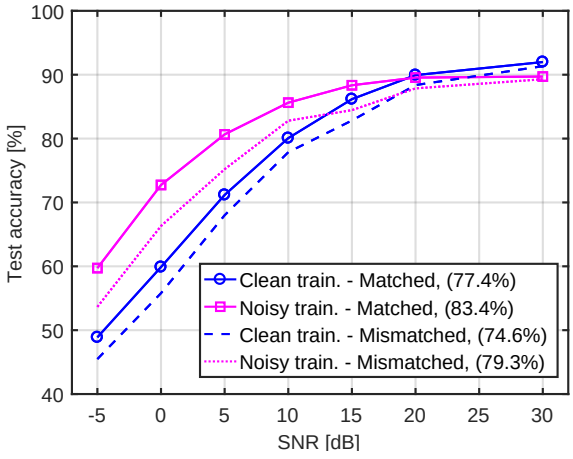


Fig. 4. Single-inference test accuracy of networks trained on clean and noisy speech. Networks were tested in both noise set 1 (matched) and noise set 2 (mismatched). The number in parentheses specifies the average accuracy in 0-20 dB SNR.

For SNRs below 20 dB the network trained on noisy data had a higher test accuracy than the network trained on clean data, while it performed slightly worse for SNRs higher than 20 dB. The performance difference was most notable in very noisy conditions. In the -5 to 5 dB SNR range, the

average accuracy for the network trained on noisy data was increased by 11.1% and 8.6% for matched and mismatched tests respectively. 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 gap between the accuracies on the matched and mismatched test. The average gap in accuracy in the range -5 to 20 dB SNR for the network trained on clean data was 3.3%, while for the network trained on noisy data this was 4.4%.

Experiment 2: Network complexity

This experiment investigated how network complexity affected the classification performance by systematically varying the number of convolutional layers and the number of filters per layer. Figure 5 shows a small selection of the most feasible networks. For each trained network, the table specifies single-inference average accuracy in 0-20 dB SNR for both test sets. The accuracy in parentheses is for the mismatched test. It also shows the number of operations needed per inference, the memory required by the model for weights/activations and the estimated execution time per inference on the Cortex M4. For networks with more than 5 layers, no significant improvements ($< 1\%$) were 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 featuring more layers.

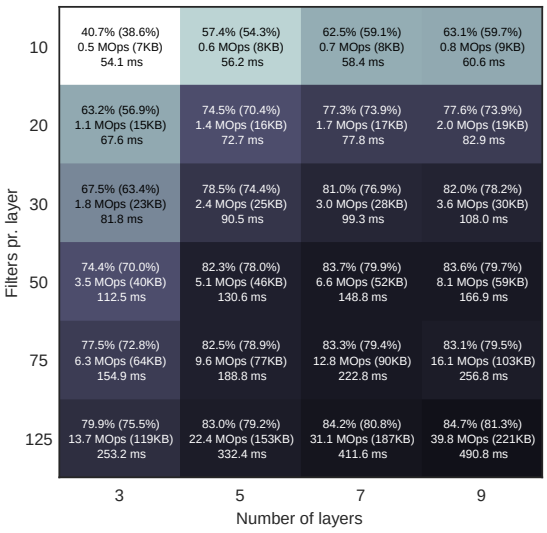


Fig. 5. Selection of hyper-parameter grid-search.[3]

Fig. 6 shows the accuracies of all the layers/filters combinations of the hyperparameter search as a function of the operations. It was seen that for large computational loads, the deviation of the accuracies was very small, while for networks using few operations there was a large difference in accuracy depending on the specific combination of layers and filters. For networks in the range 5-200 million operations, the difference in classification accuracy between the best performing models was less than 2.5%.

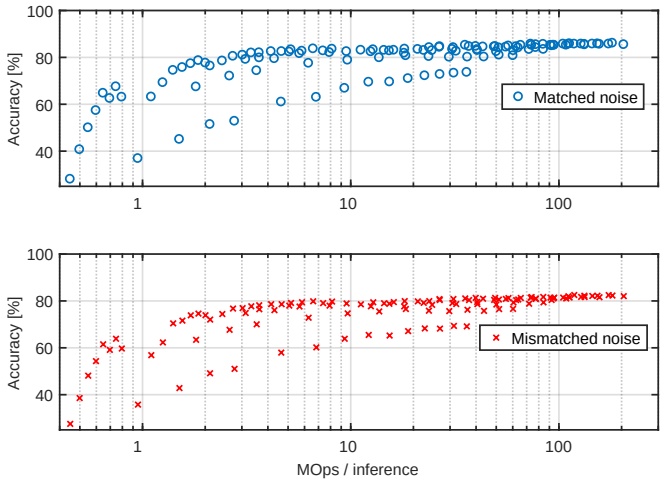


Fig. 6. Average accuracy in 0-20 dB SNR of all trained networks in hyperparameter search as a function of the number of operations per inference.

In Fig. 7 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 $[N_{layers}, N_{filters}]$ and the average accuracy in 0-20 dB SNR for noise set 1 and 2.

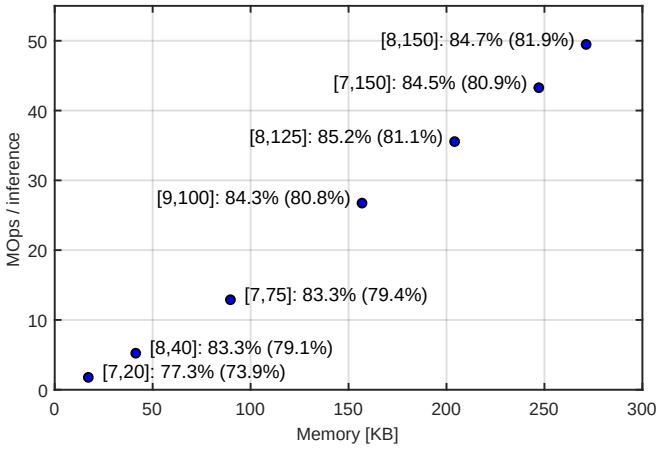


Fig. 7. Best performing networks of grid-search at different memory and computational requirements.

The best performing network of the hyperparameter search consisted of 8 layers with 300 filters per layer and had a test accuracy of 86.1% (82.3%), while the worst performing network consisted of 2 layers with 10 filters per layer and had a test accuracy of 28.1% (27.5%)

Experiment 3: Quantization

This experiment investigated the effects on performance of quantizing network weights and activations for fixed-point implementation. Table IV shows the single-inference

[3]See appendix E for full table

TABLE IV
SINGLE-INFERENCE ACCURACIES OF NETWORK WITH INDIVIDUALLY QUANTIZED NETWORK PARTS.

Floating point accuracy: 83.2% (79.3%)			
Bitwidth	8-bit	4-bit	2-bit
Conv weights	83.2% (79.1%)	82.3% (78.8%)	46.8% (44.6%)
DW-Conv weigths	83.3% (79.3%)	80.0% (76.7%)	12.0% (12.1%)
PW-Conv weights	83.4% (79.2%)	78.9% (75.4%)	11.2% (10.8%)
FC 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%)

test results of the quantized networks, where each part of the network was quantized isolated at varying bit-widths, while the rest of the network was kept at floating-point precision. The results showed that all of the weights and activations could be quantized to 8-bit using dynamic-fixed point representation with no loss of classification accuracy, and that the bit-widths of the weights could be further reduced to 4 bit with only small accuracy drops. Whereas reducing the bit width of activations to less than 8 bits significantly reduced classification accuracy. Quantizing to just 2 bits greatly reduced accuracy for regular convolution parameters and FC-parameters, and completely broke down classification for pointwise- and depthwise-convolution parameters and layer activations. The average test accuracy in 0-20 dB SNR of the network with all weights and activations quantized to 8 bit 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.

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

Experiment 4: Continuous audio stream

In this experiment the KWS-system's ability to detect keywords in an audio stream was tested. Figure 8 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 it was seen that the hit-rate and false positive rate saturated at SNRs higher than 15 dB.

FRDM K66F implementation

The KWS-system was implemented on the Cortex M4 based FRDM K66F development board. 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 [35] which features neural network operations optimized for Cortex-M processors.

[4]See appendix F for level offset test

TABLE V
LAYER DISTRIBUTION OF EXECUTION TIME AND OPERATIONS FOR CLASSIFIER INFERENCE.

Layer	Execution time [ms]	Operations$\times 10^{-6}$
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

Table V shows the distribution of execution time over network layers for a single inference. 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.

V. DISCUSSION

Experiment 1 showed that adding noise to the training material increased the classifiers robustness in low SNR conditions. The increase in accuracy, compared to the same network trained on clean speech samples, 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. For both test sets, the network trained on clean data performed better under clean conditions, i.e. $\mathrm{SNR} > 20\,\mathrm{dB}$. From the perspective of this paper however, the performance in clean conditions was of less interest as the focus was on real-world application. If the performance in clean conditions is also of concern, [21] demonstrated that the performance decrease in clean conditions could be reduced by including clean samples in the noisy training.

As was also found in [21], 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 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 discovery 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

(a) Test set 1 (matched) (b) Test set 2 (mismatched)

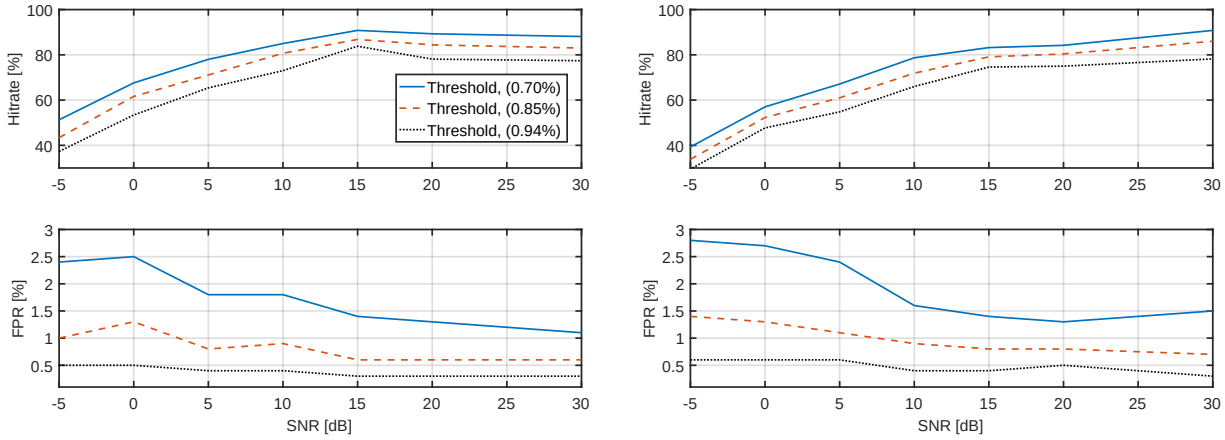


Fig. 8. Continuous audio stream test results. (a) test set 1 results. (b) test set 2 results

number of filters per layer beyond this point only resulted in insignificant accuracy gains, $< 2\,\%$. It was also seen that, given a fixed computational and memory constraint, higher accuracies were achieved for networks with many layers and few filters than for networks with few layers and many 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 $250\,\mathrm{ms}$, 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 how 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 seen that the weights of the network could all be quantized down 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. For many deep CNN-classifiers [22], [23], [31] it has also been seen 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.

[22] found that small accuracy decreases of CNN-classifiers, caused by fixed-point quantization of weights and activations, could be compensated 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. 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 requires high precision, ie. 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 have had symmetric, zero-centered ranges, mainly for convenience during implementation. However, all of the convolutional layers uses ReLU activations functions, so the activations effectively only utilize half of the available range as values below zero are cut off. By shifting the range, such that zero becomes the minimum value, the total range can be halved, ie. 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-systems performance on a continuous audio stream. As seen in most signal detection tasks, lowering the decision criterion ie. 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, ie. use more 'unknown' and 'silence' samples. This has been shown as an effective method in other machine learning detection tasks [36] [37].
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.

As seen in Tab. V, there were significant discrepancies between the estimated number of operations and actual execution time of the different layers of the implemented network. The convolutional functions in the software-library used for the implementation [35], all use highly optimized matrix multiplications (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. [38] and [39] argue that, despite this time consuming and memory expanding data-reordering, using matrix multiplications is still the most efficient implementation of convolutional layers.
The discrepancies between operations and execution time could be explained by the fact that the reordering of data was not accounted for in the operations 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 [12], 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 [40], could be used to further reduce memory footprint and execution time.

VI. 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 hyperparameter 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. Weights could be quantized to 4 bits with only small accuracy decreases. 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 hitrate and FPR. Finally, the system was verified by implementation on the Cortex M4, where it was found that the number of arithmetic operations per inference are not directly related to execution time.

Python training scripts and FRDM K66F deployment source code are available on Github [41].

References

[1] "New electronic friends," https://pages.arm.com/machine-learning-voice-recognition-report.html, Accessed: 2018-05-30.

[2] R. C. Rose and D. B. Paul, "A hidden markov model based keyword recognition system," in *International Conference on Acoustics, Speech, and Signal Processing*, Apr 1990, pp. 129–132 vol.1.

[3] J. R. Rohlicek, W. Russell, S. Roukos, and H. Gish, "Continuous hidden markov modeling for speaker-independent word spotting," in *International Conference on Acoustics, Speech, and Signal Processing,*, May 1989, pp. 627–630 vol.1.

[4] J.G. Wilpon, L.G. Miller, and P. Modi, "Improvements and applications for key word recognition using hidden Markov modeling techniques," in *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*. 1991, pp. 309–312, IEEE.

[5] Guoguo Chen, Carolina Parada, and Georg Heigold, "Small-footprint keyword spotting using deep neural networks," 2014, Proc. ICASSP.

[6] Kaixiang Shen, Meng Cai, Wei-Qiang Zhang, Yao Tian, and Jia Liu, "Investigation of DNN-Based Keyword Spotting in Low Resource Environments," *International Journal of Future Computer and Communication*, vol. 5, no. 2, pp. 125–129, 2016.

[7] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber, "An application of recurrent neural networks to discriminative keyword spotting," in *Artificial Neural Networks – ICANN 2007*, Joaquim Marques de Sá, Luís A. Alexandre, Włodzisław Duch, and Danilo Mandic, Eds., Berlin, Heidelberg, 2007, pp. 220–229, Springer Berlin Heidelberg.

[8] K.P. Li, J.A. Naylor, and M.L. Rossen, "A whole word recurrent neural network for keyword spotting," in *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*. 1992, pp. 81–84, IEEE.

[9] Yann LeCun and Yoshua Bengio, "The handbook of brain theory and neural networks," chapter Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. MIT Press, Cambridge, MA, USA, 1998.

[10] Tara N. Sainath and Carolina Parada, "Convolutional neural networks for small-footprint keyword spotting," in *INTERSPEECH*, 2015.

[11] Francois Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 7 2017, pp. 1800–1807, IEEE.

[12] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra, "Hello edge: Keyword spotting on microcontrollers," *CoRR*, vol. abs/1711.07128, 2017.

[13] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, Aug 1980.

[14] Ittichaichareon Chadawan, Suksri Siwat, and Yingthawornsuk Thaweesak, "Speech Recognition using MFCC," Pattaya (Thailand), 2012, International Conference on Computer Graphics, Simulation and Modeling (ICGSM'2012).

[15] Bhadragiri Jagan Mohan and Ramesh Babu N., "Speech recognition using MFCC and DTW," in *2014 International Conference on Advances in Electrical Engineering (ICAEE)*. 1 2014, pp. 1–4, IEEE.

[16] Manish P Kesarkar and Preeti Rao, "Feature Extraction for Speech Recogniton," Tech. Rep., M.Tech. Credit Seminar Report, Electronic Systems Group, EE. Dept, IIT Bombay,, 2003.

[17] Abdel-Rahman Mohamed, *Deep Neural Network acoustic models for ASR*, Ph.D. thesis, University of Toronto, 2014.

[18] Xiong Xiao, Jinyu Li, Eng Siong Chng, Haizhou Li, and Chin-Hui Lee, "A Study on the Generalization Capability of Acoustic Models for Robust Speech Recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 6, pp. 1158–1169, 8 2010.

[19] Ilyes Rebai, Yessine BenAyed, Walid Mahdi, and Jean-Pierre Lorré, "Improving speech recognition using data augmentation and acoustic model fusion," *Procedia Computer Science*, vol. 112, pp. 316–322, 1 2017.

[20] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur, "Audio augmentation for speech recognition," in *INTERSPEECH*, 2015.

[21] Shi Yin, Chao Liu, Zhiyong Zhang, Yiye Lin, Dong Wang, Javier Tejedor, Thomas Fang Zheng, and Yinguo Li, "Noisy training for deep neural networks in speech recognition," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2015, no. 1, pp. 2, 12 2015.

[22] Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi, "Hardware-oriented Approximation of Convolutional Neural Networks," 2016, pp. 1–8, arXiv:1604.03168.

[23] Darryl Dexu Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy, "Fixed point quantization of deep convolutional networks," *CoRR*, vol. abs/1511.06393, 2015.

[24] Douglas O'Shaughnessy, *Speech communication : human and machine*, Addison-Wesley Pub. Co, 1987.

[25] Michael A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015.

[26] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[27] Pete Warden, "Speech commands: A public dataset for single-word speech recognition.," *Dataset available from* $http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz$, 2017.

[28] H.-Guenter Hirsch, "FaNT -Filtering and Noise Adding Tool," Tech. Rep., Hochschule Niederrhein, 2005.

[29] Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen, "TUT database for acoustic scene classification and sound event detection," in *2016 24th European Signal Processing Conference (EUSIPCO)*. 8 2016, pp. 1128–1132, IEEE.

[30] Joachim Thiemann, Nobutaka Ito, and Emmanuel Vincent, "DEMAND: a collection of multi-channel recordings of acoustic noise in diverse environments," June 2013, Supported by Inria under the Associate Team Program VERSAMUS.

[31] Naveen Mellempudi, Abhisek Kundu, Dipankar Das, Dheevatsa Mudigere, and Bharat Kaul, "Mixed low-precision deep learning inference using dynamic fixed point," *CoRR*, vol. abs/1701.08978, 2017.

[32] D. Williamson, "Dynamically scaled fixed point arithmetic," in *[1991] IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference Proceedings*. pp. 315–318, IEEE.

[33] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Software available from tensorflow.org.

[34] Karolina Smeds, Florian Wolters, and Martin Rung, "Estimation of signal-to-noise ratios in realistic sound scenarios.," *Journal of the American Academy of Audiology*, vol. 26 2, pp. 183–96, 2015.

[35] Liangzhen Lai, Naveen Suda, and Vikas Chandra, "CMSIS-NN: efficient neural network kernels for arm cortex-m cpus," *CoRR*, vol. abs/1801.06601, 2018.

[36] Zhanzhan Cheng, Kai Huang, Yang Wang, Hui Liu, Jihong Guan, and Shuigeng Zhou, "Selecting high-quality negative samples for effectively predicting protein-rna interactions," *BMC Systems Biology*, vol. 11, no. 2, pp. 9, Mar 2017.

[37] Rafa Kurczab, Sabina Smusz, and Andrzej J Bojarski, "The influence of negative training set size on machine learning-based virtual screening.," *Journal of cheminformatics*, vol. 6, pp. 32, 2014.

[38] Pete 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: 2018-05-19.

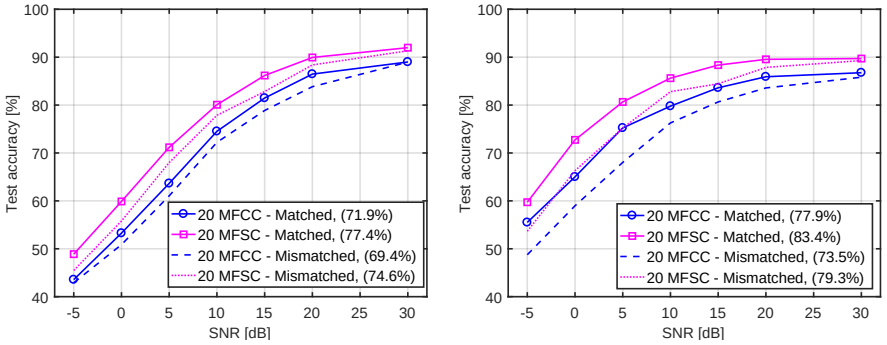
[39] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer, "cudnn: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014.

[40] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, vol. abs/1611.06440, 2016.

[41] Peter M. Sørensen, "A depthwise separable convolutional neural network for keyword spotting on embedded systems," https://github.com/PeterMS123/KWS-DS-CNN-for-embedded, 2018.

APPENDIX A
MFCC VS MFSC

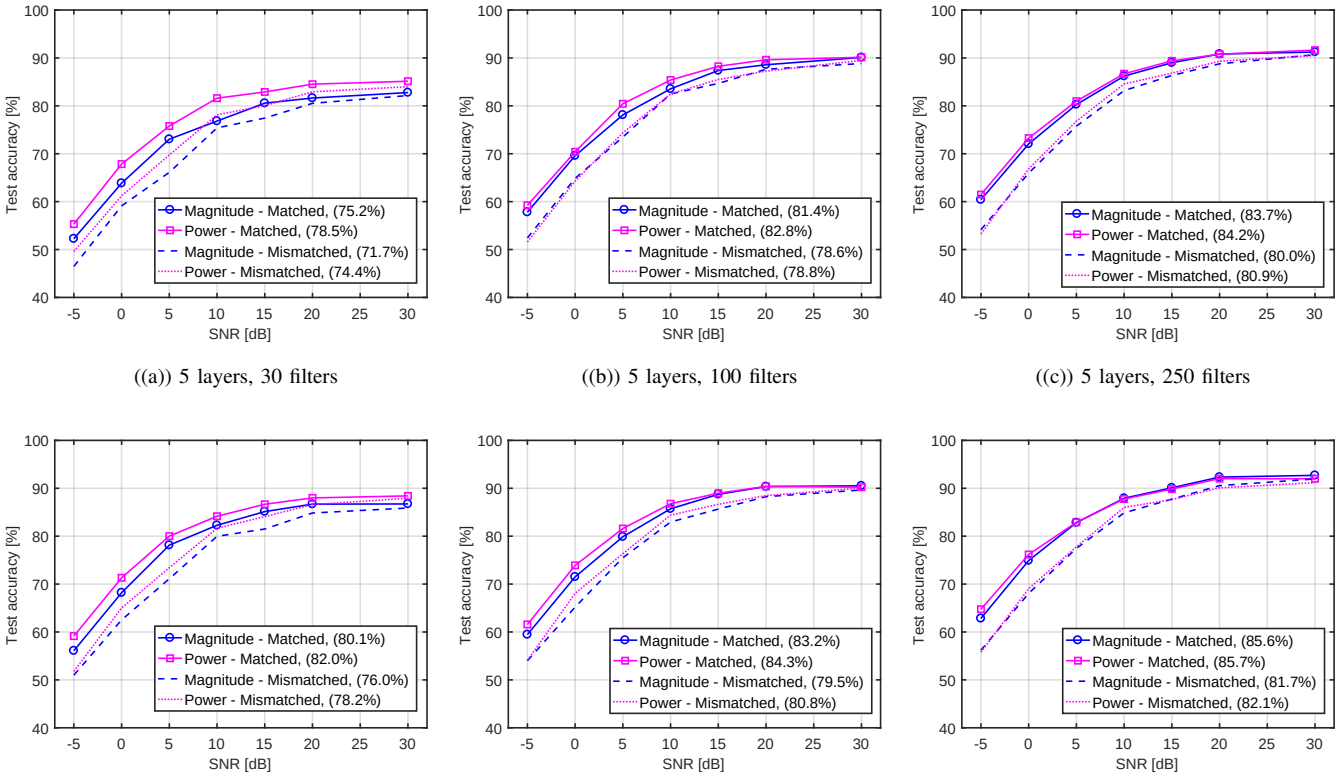
Figure 9 shows the performance improvement achieved using 20 MFSC-features instead of 20 MFCC for the DS-CNN classifier. Using the same number of input coefficients, it was seen that accuracy increased across all SNRs in the test for both clean and noisy trained networks. For the MFCC-extraction, the mel-filterbank consisted of 40 filters in the range 20-4000 Hz, while for the MFSC, the number of filters in the mel-filterbank was equal to the number of features. The same frequency range was used for both approaches.



((a)) Clean training ((b)) Noisy training

Fig. 9. MFCC vs MFSC input features. Figure shows classification accuracies of networks trained on either clean data or data mixed with background noise. The network consists of 7 convolutional layers with 76 filters per layer.

APPENDIX B
POWER VS MAGNITUDE SPECTROGRAM



((a)) 5 layers, 30 filters ((b)) 5 layers, 100 filters ((c)) 5 layers, 250 filters

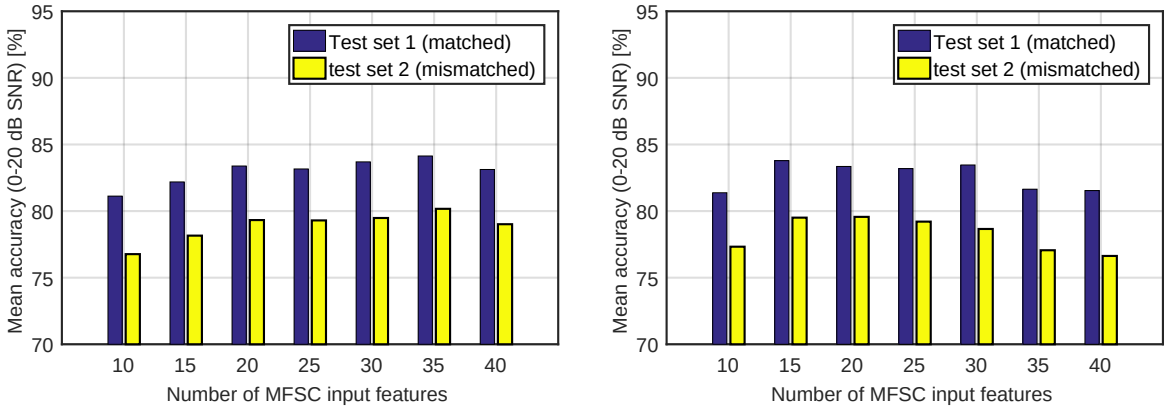
((d)) 9 layers, 30 filters ((e)) 9 layers, 100 filters ((f)) 9 layers, 250 filters

Fig. 10. Accuracies of various-sized networks trained and tested using either the magnitude or power of the spectrogram in the feature extraction stage. Networks were trained on noisy data

Figure 10 shows the effects on classification accuracy of using either the magnitude or the power of the spectrogram in the MFSC feature extraction. This was tested for a variety of network sizes. For all network configurations it was seen that the accuracy was higher when using the power spectrogram, than when the magnitude was used. Interestingly, the difference in performance was largest for the smallest networks.

APPENDIX C
NUMBER OF MFSC FEATURES

The impact of varying the number of MFSCs per spectrogram time frame on the classification accuracy was investigated for two networks of different size. Fig. 11.
The number of features directly impact the size of the input an should therefore be kept low to decrease the computational load. For the the two networks, 20 MFSCs was seen to perform well and was therefore selected for the feature extraction of the KWS-system.



((a)) 7 layers, 76 filters

((b)) 5 layers, 172 filters

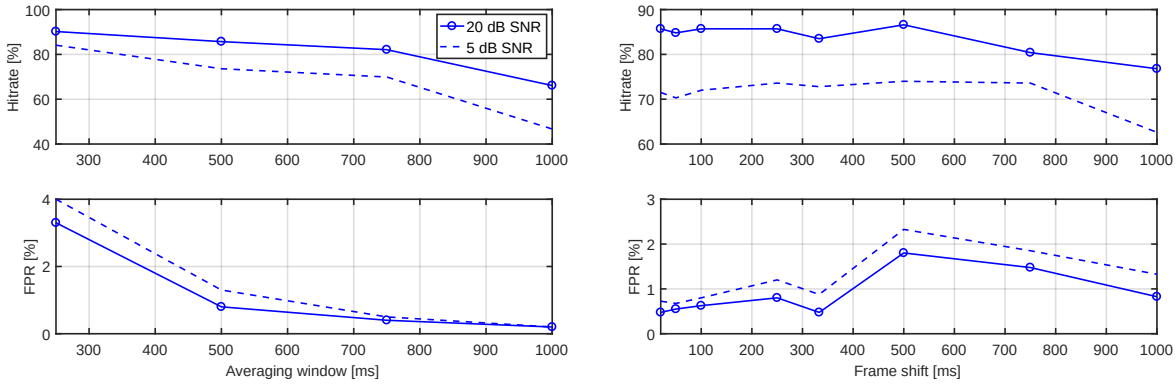
Fig. 11. Classification accuracy as a function of the number of MFSC input features, for two networks of different size. Network were trained on noisy data.

APPENDIX D
POSTERIOR HANDLING PARAMETERS

To determine the frame shift of the input window and the number of frames to average posterior probabilities over in order to obtain a reasonable performance, the KWS-system was tested on a continuous audio stream with varying input frame shift and averaging window. Figure 12(a) shows the hitrate and false positive rate when the averaging window is varied, and an input frame shift of 250 ms is used. It was seen that not averaging caused many false alarms, while averaging over many frames significantly decreased the hitrate. A 500 ms averaging window was chosen for the KWS-system, as this resulted in a high hitrate with relatively few false alarms.
Figure 12(b) shows the effect of varying the input frame shift. It was found that a large frame shift could be used without decreasing the hitrate significantly, but that the FPR reached high values at frame shifts larger than approximately 300 ms. Based on this, an input frame shift of 250 ms was selected fro the KWS-system.

APPENDIX E
NETWORK COMPLEXITY - EXTENDED

Figure 13 shows the full grid search, where the number of convolutional layers and number of filters per layer were varied. Figure 14 shows an extension of the hyper parameter grid search, where network configurations with many layers and few filters per layer were investigated. None of the networks in the extended grid search did however surpass the accuracies relative to the execution time seen in the original grid search, Fig. 13



((a)) Averaging window ((b)) Input frame shift

Fig. 12. Hitrate and FPR as a function of the averaging window applied to posterior classifier probabilities, and the frame shift between classifier inference.

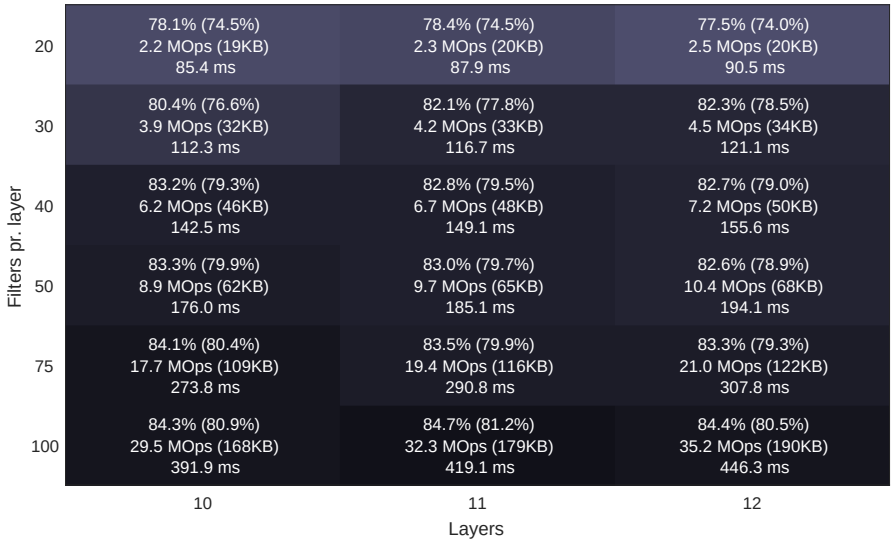


Fig. 14. Extension of grid search. 10-12 layers, 20-100 filters per layer

APPENDIX F
LEVEL OFFSET

The networks robustness to changes in input signal level was tested by multiplying the input time-domain signals by a gain-factor between -30 dB and 30 dB, and measuring the classification accuracy. The results are seen in Fig. 15.
The network was tested using both floating point precision and 8-bit fixed-point precision. In both cases it was seen that the accuracy dropped slightly when the test samples were attenuated, but deteriorated greatly for gains larger than 5 dB. The large performance decrease in high gains for both floating point and fixed point precision suggests that it was caused by the signal being clipped, which causes signal distortions.
In the range of -15 dB to 5 dB level offset, the performance was unaffected.

Filters pr. layer	2 Layers	3 Layers	4 Layers	5 Layers	6 Layers	7 Layers	8 Layers	9 Layers
10	28.1% (27.5%) 0.4 MOps (7KB) 53.0 ms	40.7% (38.6%) 0.5 MOps (7KB) 54.1 ms	50.0% (48.1%) 0.5 MOps (7KB) 55.1 ms	57.4% (54.3%) 0.6 MOps (8KB) 56.2 ms	64.7% (61.5%) 0.6 MOps (8KB) 57.3 ms	62.5% (59.1%) 0.7 MOps (8KB) 58.4 ms	67.5% (63.9%) 0.7 MOps (8KB) 59.5 ms	63.1% (59.7%) 0.8 MOps (9KB) 60.6 ms
20	36.9% (35.8%) 1.0 MOps (14KB) 65.0 ms	63.2% (56.9%) 1.1 MOps (15KB) 67.6 ms	69.3% (62.3%) 1.3 MOps (16KB) 70.1 ms	74.5% (70.4%) 1.4 MOps (16KB) 72.7 ms	75.7% (71.6%) 1.6 MOps (17KB) 75.2 ms	77.3% (73.9%) 1.7 MOps (17KB) 77.8 ms	78.7% (74.6%) 1.9 MOps (18KB) 80.3 ms	77.6% (73.9%) 2.0 MOps (19KB) 82.9 ms
30	45.0% (42.8%) 1.5 MOps (22KB) 77.5 ms	67.5% (63.4%) 1.8 MOps (23KB) 81.8 ms	76.4% (72.0%) 2.1 MOps (24KB) 86.2 ms	78.5% (74.4%) 2.4 MOps (25KB) 90.5 ms	80.5% (76.7%) 2.7 MOps (27KB) 94.9 ms	81.0% (76.9%) 3.0 MOps (28KB) 99.3 ms	81.9% (77.7%) 3.3 MOps (29KB) 103.6 ms	82.0% (78.2%) 3.6 MOps (30KB) 108.0 ms
40	51.4% (49.1%) 2.1 MOps (29KB) 90.3 ms	72.1% (67.7%) 2.6 MOps (31KB) 96.8 ms	79.2% (74.8%) 3.1 MOps (33KB) 103.3 ms	79.8% (76.4%) 3.6 MOps (35KB) 109.9 ms	82.5% (78.6%) 4.1 MOps (38KB) 116.4 ms	82.5% (78.6%) 4.7 MOps (40KB) 122.9 ms	83.3% (79.1%) 5.2 MOps (42KB) 129.5 ms	81.6% (77.6%) 5.7 MOps (44KB) 136.0 ms
50	52.8% (51.0%) 2.8 MOps (37KB) 103.4 ms	74.4% (70.0%) 3.5 MOps (40KB) 112.5 ms	79.5% (76.1%) 4.3 MOps (43KB) 121.5 ms	82.3% (78.0%) 5.1 MOps (46KB) 130.6 ms	82.7% (79.5%) 5.8 MOps (49KB) 139.7 ms	83.7% (79.9%) 6.6 MOps (52KB) 148.8 ms	82.8% (79.1%) 7.4 MOps (56KB) 157.8 ms	83.6% (79.7%) 8.1 MOps (59KB) 166.9 ms
75	61.0% (57.9%) 4.6 MOps (58KB) 137.9 ms	77.5% (72.8%) 6.3 MOps (64KB) 154.9 ms	82.1% (78.0%) 7.9 MOps (71KB) 171.9 ms	82.5% (78.9%) 9.6 MOps (77KB) 188.8 ms	83.1% (78.6%) 11.2 MOps (83KB) 205.8 ms	83.3% (79.4%) 12.8 MOps (90KB) 222.8 ms	83.1% (79.0%) 14.5 MOps (96KB) 239.8 ms	83.1% (79.5%) 16.1 MOps (103KB) 256.8 ms
100	63.0% (60.2%) 6.8 MOps (79KB) 174.6 ms	78.8% (74.7%) 9.7 MOps (91KB) 201.7 ms	82.4% (77.7%) 12.5 MOps (102KB) 228.9 ms	82.8% (78.8%) 15.3 MOps (113KB) 256.1 ms	83.6% (79.9%) 18.2 MOps (124KB) 283.3 ms	83.4% (79.8%) 21.0 MOps (135KB) 310.4 ms	84.2% (79.9%) 23.8 MOps (146KB) 337.6 ms	84.3% (80.8%) 26.7 MOps (157KB) 364.8 ms
125	66.8% (63.9%) 9.4 MOps (102KB) 213.5 ms	79.9% (75.5%) 13.7 MOps (119KB) 253.2 ms	81.8% (77.7%) 18.1 MOps (136KB) 292.8 ms	83.0% (79.2%) 22.4 MOps (153KB) 332.4 ms	84.6% (80.4%) 26.8 MOps (170KB) 372.0 ms	84.2% (80.8%) 31.1 MOps (187KB) 411.6 ms	85.2% (81.1%) 35.5 MOps (204KB) 451.2 ms	84.7% (81.3%) 39.8 MOps (221KB) 490.8 ms
150	69.5% (65.5%) 12.2 MOps (127KB) 254.8 ms	80.7% (76.5%) 18.4 MOps (151KB) 309.1 ms	82.9% (78.3%) 24.6 MOps (175KB) 363.4 ms	83.4% (79.1%) 30.8 MOps (199KB) 417.7 ms	84.7% (80.4%) 37.0 MOps (223KB) 472.0 ms	84.5% (80.9%) 43.2 MOps (247KB) 526.2 ms	84.7% (81.3%) 49.4 MOps (272KB) 580.5 ms	84.4% (80.9%) 55.6 MOps (296KB) 634.8 ms
175	69.6% (65.2%) 15.4 MOps (152KB) 298.3 ms	80.4% (75.8%) 23.7 MOps (185KB) 369.5 ms	82.7% (78.7%) 32.1 MOps (217KB) 440.7 ms	83.3% (79.6%) 40.5 MOps (250KB) 512.0 ms	84.5% (80.4%) 48.9 MOps (282KB) 583.2 ms	84.9% (81.2%) 57.2 MOps (315KB) 654.5 ms	85.2% (81.3%) 65.6 MOps (347KB) 725.7 ms	85.2% (81.6%) 74.0 MOps (380KB) 796.9 ms
200	71.0% (67.1%) 18.9 MOps (179KB) 344.0 ms	80.1% (75.7%) 29.7 MOps (221KB) 434.4 ms	83.0% (78.6%) 40.6 MOps (263KB) 524.9 ms	84.1% (80.6%) 51.5 MOps (305KB) 615.3 ms	84.6% (80.4%) 62.3 MOps (348KB) 705.8 ms	85.7% (81.4%) 73.2 MOps (390KB) 796.2 ms	85.6% (81.7%) 84.1 MOps (432KB) 886.6 ms	85.2% (81.4%) 94.9 MOps (474KB) 977.1 ms
225	72.2% (68.3%) 22.7 MOps (207KB) 392.0 ms	80.2% (76.2%) 36.4 MOps (260KB) 503.9 ms	82.5% (78.5%) 50.1 MOps (313KB) 615.8 ms	84.1% (80.6%) 63.8 MOps (366KB) 727.7 ms	85.5% (81.0%) 77.4 MOps (419KB) 839.6 ms	85.2% (81.4%) 91.1 MOps (472KB) 951.5 ms	85.7% (81.4%) 104.8 MOps (525KB) 1063.4 ms	85.7% (82.5%) 118.5 MOps (578KB) 1175.3 ms
250	72.8% (68.2%) 26.8 MOps (236KB) 442.2 ms	80.2% (75.8%) 43.7 MOps (301KB) 577.9 ms	82.9% (79.3%) 60.5 MOps (367KB) 713.5 ms	84.2% (80.9%) 77.3 MOps (432KB) 849.1 ms	85.3% (81.1%) 94.2 MOps (497KB) 984.7 ms	85.7% (82.0%) 111.0 MOps (562KB) 1120.3 ms	85.7% (81.7%) 127.8 MOps (628KB) 1255.9 ms	85.7% (82.1%) 144.7 MOps (693KB) 1391.5 ms
275	73.3% (69.3%) 31.3 MOps (266KB) 494.8 ms	81.0% (76.5%) 51.6 MOps (345KB) 656.3 ms	83.4% (78.8%) 71.9 MOps (424KB) 817.9 ms	85.0% (80.4%) 92.2 MOps (502KB) 979.5 ms	85.9% (81.4%) 112.5 MOps (581KB) 1141.1 ms	85.4% (81.7%) 132.8 MOps (660KB) 1302.7 ms	85.8% (82.2%) 153.1 MOps (738KB) 1464.3 ms	85.7% (82.5%) 173.4 MOps (817KB) 1625.8 ms
300	73.6% (69.2%) 36.1 MOps (298KB) 549.5 ms	80.8% (76.6%) 60.2 MOps (392KB) 739.3 ms	83.4% (79.4%) 84.3 MOps (485KB) 929.1 ms	85.2% (81.1%) 108.4 MOps (578KB) 1119.0 ms	85.6% (82.2%) 132.5 MOps (671KB) 1308.8 ms	85.6% (81.6%) 156.6 MOps (765KB) 1498.6 ms	86.1% (82.3%) 180.7 MOps (858KB) 1688.4 ms	85.5% (82.0%) 204.8 MOps (951KB) 1878.2 ms

Fig. 13. Full hyper-parameter grid search table

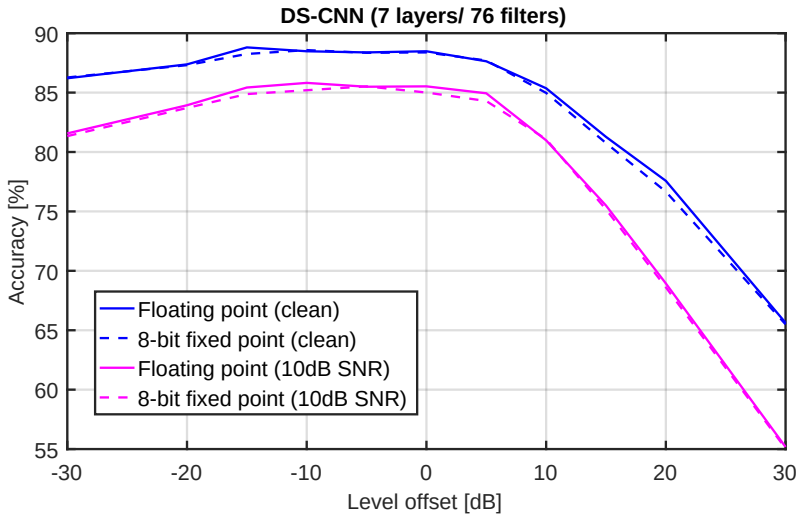


Fig. 15. Classification accuracy as a function of input signal level offset. Tested in clean conditions and at 10 dB SNR for networks using floating point precision and 8-bit fixed-point precision.