Binarizing Convolutional Neural Networks For Efficient Keyword Spotting

Improving Energy Efficiency For Mobile Devices

Justine Winkler

17.06.2018

1 Introduction

In our day-to-day life, we are constantly interacting with intelligent devices. To simplify this interaction, an increasing number of such devices can nowadays be controlled via spoken instructions. For instance, most of the smartphones put on the market feature integrated virtual assistants.

Two popular examples of virtual assistants are Apple's Siri and Google Speech Recognition (GSR). Constantly monitoring noises in their environment enables them to react when needed using automatic speech recognition (ASR) [Këpuska and Klein, 2009]: They transcribe incoming speech into text and subsequently parse that text to extract relevant information, such as the user's commands. Commands are usually expressed using specific phrases or names of certain apps. For instance, the virtual assistants themselves become activated by simply uttering a so-called wake-up word [Hou et al., 2016]. Examples of such pre-defined commands are "Hey Siri" or "Okay Google" for Siri or GSR, respectively. A subfield of ASR tasks, the set of so-called keywords spotting (KWS) problems, deals with correctly recognizing and detecting such important words in a given audio input.

A large body of literature has used Hidden Markov Models to approach ASR problems in general. However, more recent attention has focused on the utilization of neural networks (NNs) instead. Advantages of using NNs are that they can learn and model higher-order features from their input. After training and thereby learning how to extract those relationships, NNs can use this ability to perform inference on new data samples. Further, NNs require their input to fulfill only few restrictions regarding its representation. However, training a NN has high computational costs, requiring a great amount of time even when using high-performance hardware. In a nutshell, NNs are a powerful, yet computationally expensive method when applied to ASR problems.

It is this property of NNs which makes it difficult to employ them on smartphones. Thus, to improve digital assistants' KWS performance, state-of-the-art methods should be modified to reduce their computational costs. Some recent approaches use Convolutional Neural Networks (CNNs) to perform KWS because of their ability to learn their own input features [LeCun et al., 1995, Sainath and Parada, 2015]. This means they require little preprocessing and no specific feature extractors up front. A recent development in optimizing NNs for low-powered applications is research into Binary Neural Networks (BNN). In Binary Neural Networks all neuronal activations and weights are restricted to +1 or -1 [Courbariaux et al., 2016]. This means the activations and weights can be represented as bits, saving previous memory. Furthermore, logic machines such as computers are generally ideally suited for bit-operations, meaning inference on BNNs is much quicker. The BNN model has already been successfully applied to full-scale ASR networks [Xiang et al., 2017]. However, we could not find approaches that applied binarization to networks for KWS in the literature.

The objectives of this research are to determine whether the task of KWS can be solved more efficiently by utilizing binarization. To answer this general question, we will binarize CNNs that have been successfully applied to KWS problems before. The application of those architectures on single keyword detection from audio files will then be assessed in terms of performance and computational costs, leading to two hypotheses. First, we expect to have a similar performance among CNNs' canonical form and their binarized version. Further, we expect lower computational cost when applying BNNs to a problem.

1.1 Related Work

1.1.1 Neural Networks

CNNs and Speech Audio Certain feature extractors for audio result in a set of feature values for a given input timeframe. By applying such feature extractors (e.g. log-mel and MFCC) with a sliding window to an audio fragment, one gets back a matrix with the features in a column vector changing over time. As this matrix is akin to an image, standard CNN (training) techniques can be applied with little modification [Golik et al., 2015].

1.1.2 Binary Neural Networks

In order to represent non-integer real numbers, computers use the floating point number representation format. The format allows for the representation of non-integer numbers, as well as shifting accuracy. This means the format can represent both small numbers very precisely and large numbers. This format is however either slower to calculate with than regular integers or requires power-hungry circuits. Furthermore, circuit logic for floating point calculations is complex compared to circuit logic for integer and bit-operations. It follows that from a hardware and power perspective, the floating point representation is an unfavourable medium to perform calculations with.

To counter this, the idea was created to constrain the activation and weight of a neuron to either +1 or -1, so that they can be represented using a single bit [Courbariaux et al., 2016]. A bit operation is power efficient, simple to implement in circuitry and also consumes less memory. Because of the binary nature of using a single bit to represent activations and weights, networks with this constraint are called binary neural networks (BNNs).

Application of BNNs in image processing shows promising results. In one example applied to the MINIST/CIFAR10 dataset, researchers were able to get a 600% increase in computational performance and only 1-5 higher percent classification error-rate [Courbariaux et al., 2016].

In other research, applied to ASR in general, the researchers were able to gain an 400%-700% increase in execution performance, with only an average 9.5% relative increase in word error rate (WER) compared to the non-binary DNN [Xiang et al., 2017]. As the drop in task performance is far less than the increase in computational performance, it might be possible to increase the network size in order to compensate for the task performance drop while maintaining a superior computational performance.

1.1.3 Binarization

Given a network N with $L \geq 2$ layers, define the weight matrix between N_i and N_{i+1} as $W_{i,i+1}$, the activation of L_i as a_i and the bias in layer L_{i+1} as b_{i+1}. We denote the binarized weight matrices and activations as $\hat{W}_{i,i+1}$ and $\hat{a}_i$, respectively.

1.1.4 Feed-forward pass

Algorithm 1 describes the feed-forward pass in binary neural networks. Special consideration has to be put into the input layer as the input is not necessarily binary. One has to make the choice between forcing the input data to be converted into a binary representation during pre-processing, or keeping $W_{1,2}$ non-binary.

Equations 1 and 2 show two versions of the `Binarize(·)` function. Equation 1 shows the deterministic version which is used during inference. Equation 2 is the stochastic version with a random variable p drawn from a zero-mean normal distribution with unit-variance. The stochastic version is more computationally expensive due to the requirement of random number generation. The stochasticity reduces overfitting which is why training uses the stochastic version instead of the deterministic version.

Backpropogation pass The backpropogation pass is less straight forward because the gradient of `Binarize(·)` is 0 for all input as it is a step function. We therefore require some other method of providing a gradient value that can be used with gradient descend optimization. A set of alternatives are *Straight Through Estimators (STEs)* [Bengio et al., 2013, Xiang et al., 2017]. Equation 3 defines the STE used in [Courbariaux et al., 2016].

$$Binarize(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{otherwise} \end{cases} \tag{1}$$

$$Binarize(x) = \begin{cases} +1 & \text{if } x - p > 0 \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

$$\begin{aligned} \text{forward pass} \quad & y = \texttt{Binarize(x)} \\ \text{backward pass} \quad & \frac{\partial loss}{\partial x} = \frac{\partial loss}{\partial y} \times \mathbb{1}_{|x| \leq 1} \end{aligned} \tag{3}$$

```
input  : Input activation $a_1$, weight matrices $W_{1,2}, ..., W_{L-1,L}$, and biases $b_1, ..., b_L$
output: Activations $a_2, ..., a_L$

1. Input layer (Assuming non-binarized input)
$a_2 = W_{1,2} \times a_1 + b_2$
$a_2 = BatchNorm(a_2)$
$a_2 = HardTanh(a_2)$
$\hat{a}_2 = Binarize(a_2)$

2. Hidden layers
for $i \leftarrow 3$ to $L-1$ do
    $\hat{W}_{i-1,i} = Binarize(W_{i-1,i})$
    $a_i = \hat{W}_{i-1,i} \times \hat{a}_{i-1} + b_i$
    $a_i = BatchNorm(a_i)$
    $a_i = HardTanh(a_i)$
    $\hat{a}_i = Binarize(a_i)$
end

3. Output layer
$\hat{W}_{L-1,L} = Binarize(W_{L-1,L})$
$a_L = \hat{W}_{L-1,L} \times \hat{a}_{L-1} + b_L$
$a_L = BatchNorm(a_L)$
$\hat{a}_L = Softmax(a_L)$
```

Algorithm 1: Feed-forward pass of BNN [Courbariaux et al., 2016]

2 Methodology

2.1 Dataset

2.1.1 Google Speech Command Dataset

To train the networks, we required a set of words to spot and a large set of *filler* words. For the target words we used the Google AIY Speech Command dataset [Warden, 2017]. This dataset comprises of 30 different words including words such as 'yes', 'no', 'on', 'off', 'up', 'down' and all digits. The utterances were recorded by volunteers around the world without quality restrictions. The data was captured in a variety of formats and was converted to 16-bit mono-channel WAVE files of 1 second long resampled at 16kHz.

Furthermore, the GSC dataset contains 6 different types of background noise: someone doing dishes, exercising on a bicycle, a running tap, a man making noises, white noise and pink noise. All background noise files are 1 minute or longer. We will use these to augment our data. In Section 2.1.4 we go into more detail on our data augmentation.

2.1.2 TIMIT

Another dataset that is of interest is TIMIT [Garofolo et al., 1993]. TIMIT has a much more diverse vocabulary, but with less sample recordings per word. This would make it ideal to use as a source for filler words. In contrast to the Speech Command dataset, the TIMIT dataset was recorded at a single location; a Texas Instruments laboratory. The recordings are stored as 16kHz mono-channel WAVE files.

2.1.3 Combining Datasets

Using a diverse set of words as for the filler dataset, will result in a model with a more accurately distinctive representation between target and filler words. This means the model will give less false positives, or in the KWS literature referred to as *false alarms*. In order to increase the size of the filler dataset, we attempted to augment the Google AIY Speech Command dataset with the TIMIT dataset. Recordings of words that are also in the Speech Command dataset would be added to their respective classes in that dataset and the remainder of the words would be used to represent the filler class. However, when combining datasets there was the inherent possibility that networks trained on the resultant combination would fit on characteristics common within a dataset. Since TIMIT was recorded in a controlled environment, it would be possible that trained models would create a sort of 'is-timit' feature. Considering that Convolutional NNs are especially good at forming their own features, this scenario was quite feasible. Furthermore, because the proposed method of combining the datasets partially results in a TIMIT vs Speech Command task, it was important to determine whether a model will actually be able to pick up on these dataset characteristics.

To make sure the trained networks would not become sensitive to these characteristics, a separate network was trained, tasked to distinguish the datasets. The network was be based on the `cnn-trad-fpool3` network from [Sainath and Parada, 2015], but with only two outputs representing the source dataset. If the network would perform at or close to chance level, the differences in acoustic properties are so small that neural networks of this size trained on KWS would not be able to find the corresponding pattern. This meant that the networks trained for keyword spotting will be unlikely to pick up on acoustic characteristics of the datasets. In turn this would mean that the datasets could be combined without concern.

If on the contrary, the network *was* able to accurately distinguish between the datasets there is a distinct possibility that KWS models will form one or more features representing the source dataset. Consequently, the KWS system could determine an activation hit partially as an 'is-timit' decision. This is of course undesirable. If this was to be the case, the datasets could not be combined.

2.1.4 Data Augmentation

In order to help the model generalize over the training input, we augmented the data to add additional variance. First, we shifted the word onset in the sound fragment slightly (0-300 ms), so the network would not fit on onset timing, but on the word itself. Furthermore, because the performance test used a single long audio input, segmented into seconds, the specific word-onset within the segment will vary.

Second, we added varying degrees of different types of background noise. We randomly selected a background noise audio clip and extract a one second segment at random from this clip. This was then be combined with an input sample with a random loudness between 5dB and -10dB relative to the sample.

2.1.5 Pre-processing

We did a limited amount of pre-processing, as we want the CNN to be able to develop feature maps on its own.

We used the Mel-frequency cepstral coefficients (MFCCs) of an audio segment as input for the network. Using a sliding window, we extracted 40 coefficients a total of 32 times. This results in a matrix of 40x32 (coefficients vs time) as input to the CNN.

2.2 Models

Our baseline was the `cnn-trad-fpool3` model of [Sainath and Parada, 2015] (see Table 1 for structure). We furthermore also implemented some of the improved models from their work. This gave us further validation that results from our binary equivalents for they could be compared to existing results. The network architectures are listed in Tables 1, 2 and 3.

2.3 Experiments

2.3.1 Implementation & Training

All networks were implemented both in their canonical form as well as in binary form using keras [Chollet et al., 2015]. For the binary versions of models, we used the binary DNN and binary CNN layers from [Ding, 2017]. Training and evaluation was performed on a machine with an *AMD Threadripper 1950X* CPU and an *nVidia Geforce GTX 1080 Ti* GPU.

We used a ratio of 80:5:15 for the train, validation and test sets respectively. We used three different key words, namely "Marvin", "Happy" and "Cat". The other words were considered distractors. All networks were trained for 40.000 epochs with 1000 samples per epoch using Stochastic Gradient Descent [Zhang, 2004]. A decreasing learning rate was used in the form of Equation 4, which computes the learning rate for the n^{th} epoch (lr_n). We set the starting learning rate (lr_0) to 0.0001.

For each of the models, we report the accuracy, the confidence, and the standard deviation of the confidence, for each of the target words, the distractors and average over all input.

$$lr_n = lr_0 \frac{1}{1 + n\frac{lr_0}{N}} \tag{4}$$

The implementation used for these experiments can be found at [Bokkers et al., 2018].

2.3.2 Evaluation

For the evaluation of the models, the test split of the Google Speech Command Dataset was used. All the elements of the test set where padded to be one second long. A random onset shift of the audio, ranging between 0-300 ms was added. Then, normalization was applied to the sample and the respective MFCC representation was calculated.

Each of these samples was fed into the model, which generated a prediction. Using the output vector, the choice and confidence were calculated. The confidence was calculated using equation 5.

$$\text{confidence} = \frac{\max o}{\sum o} \tag{5}$$

Given that a binarized version of a network is seven times as fast [Courbariaux et al., 2016], and that the run time is mostly dependent on the amount of arithmetic operations, which scales approximately linearly with the number of parameters [Courbariaux et al., 2016], it follows that any binary network with less parameters than seven times that of a non-binary model will perform as efficient or better. Following this reasoning, we only compared the task performance of the different models.

2.3.3 Experiments, First Round

In the first round of experiments, we trained and evaluated the architectures below. We trained and tested both the binary and non-binary versions. The corresponding results can be found in Figure 2a-2c are discussed in Section 3.1.

type	n	m	r	x	y	params
conv	20	8	64	1	3	10,304
conv	10	4	64	1	1	163,904
lin	-	-	32	-	-	65,600
dense	-	-	128	-	-	4,224
dense	-	-	4	-	-	516
Total	-	-	-	-	-	244,548

Table 1: `cnn-trad-fpool3` $s, v = 1$

type	n	m	r	x	y	params
conv	21	8	94	2	3	32,786
conv	6	4	94	1	1	903,458
lin	-	-	32	-	-	193,472
dense	-	-	128	-	-	4,224
dense	-	-	4	-	-	516
Total	-	-	-	-	-	1,134,456

Table 2: `cnn-tpool2` $s, v = 1$

type	n	m	r	x	y	params
conv	15	8	94	3	3	11,374
conv	6	4	94	1	1	212,158
lin	-	-	32	-	-	103,616
dense	-	-	128	-	-	4,224
dense	-	-	4	-	-	516
Total	-	-	-	-	-	331,888

Table 3: `cnn-tpool2` $s, v = 1$

2.3.4 Experiments Round Two

As described in Section 3.1, the performance of the binarized models was not on par or close to the original models. We therefore modified the binary version `cnn-trad-fpool3` to have a larger number of parameters while staying below 1,711,836 (7 times the amount of parameters the original network). The modified architectures are described in Tables 4-7.

bcnn-trad-fpool3_more-filters
: has 256 filters instead of 64 in the second convolution layer. The full structure is listed in Table 4.

bcnn-trad-fpool3_last-bin-layer
: has only the output layer set to binary. We argued that the earlier layers might not be able to capture enough information when binarized. Since the output of the second to last layer encodes *deep features*, a final binary layer should be able to perform better given that the deep features are still real-numbered.

bcnn-trad-fpool3_last-bin-layer_more-neuron
: has in increased number of neurons in the dense layer. Given that binary layers can only have +1 or -1 in as weights, more neurons would give rise to a larger variety of inputs for the binary output layer to combine.

bcnn-trad-fpool3_last-bin-layer_minus-conv
: is an small modification of the `bccn-trad-fpool3_last-bin-layer_more-neurons` architecture. By replacing the linear layer by a convolutional layer, we get four times as many outputs from the convolutional layer, which might present more information to the dense layer.

type	n	m	r	x	y	params
conv	20	8	64	1	3	10,304
bconv	10	4	256	1	1	655,360
lin	-	-	32	-	-	491,520
bdense	-	-	256	-	-	65,536
bdense	-	-	4	-	-	1024
Total	-	-	-	-	-	1,223,680

Table 4: `bcnn-trad-fpool3_more-filters` $s, v = 1$

type	n	m	r	x	y	params
conv	20	8	64	2	3	10,240
conv	10	4	64	1	1	163,840
lin	-	-	32	-	-	65,536
dense	-	-	128	-	-	4,096
bdense	-	-	4	-	-	512
Total	-	-	-	-	-	244,244

Table 5: `bcnn-trad-fpool3_last-bin-layer` $s, v = 1$

type	n	m	r	x	y	params
conv	20	8	64	1	3	10,240
conv	10	4	64	1	1	163,840
conv	30	1	32	-	-	61,440
dense	-	-	512	-	-	65,536
bdense	-	-	4	-	-	2,048
Total	-	-	-	-	-	303,104

Table 6: `bcnn-trad-fpool3_last-bin-_minus-conv` $s, v = 1$

type	n	m	r	x	y	params
conv	20	8	64	2	3	10,240
conv	10	4	64	1	1	163,840
lin	-	-	32	-	-	65,536
dense	-	-	512	-	-	16,384
bdense	-	-	4	-	-	2,048
Total	-	-	-	-	-	258,048

Table 7: `bcnn-trad-fpool3_last-bin-layer_more-neurons` $s, v = 1$

2.4 Within-class variation

To further inspect classes, we decided to inspect the within-class variation. For each audiosample in a class, the MFCC values were computed, respectively. Then, the resulting MFCC matrix was converted to a row vector and subsequently all MFCC row vectors were concatenated. The average and standard deviation values of these vectors were computed in the vertical direction. The contents of the resulting vector are the average and standard deviation per MFCC matrix-cell. From those we computed the mean average, the standard deviation of the average, the mean standard deviation and the standard deviation of the standard deviation.

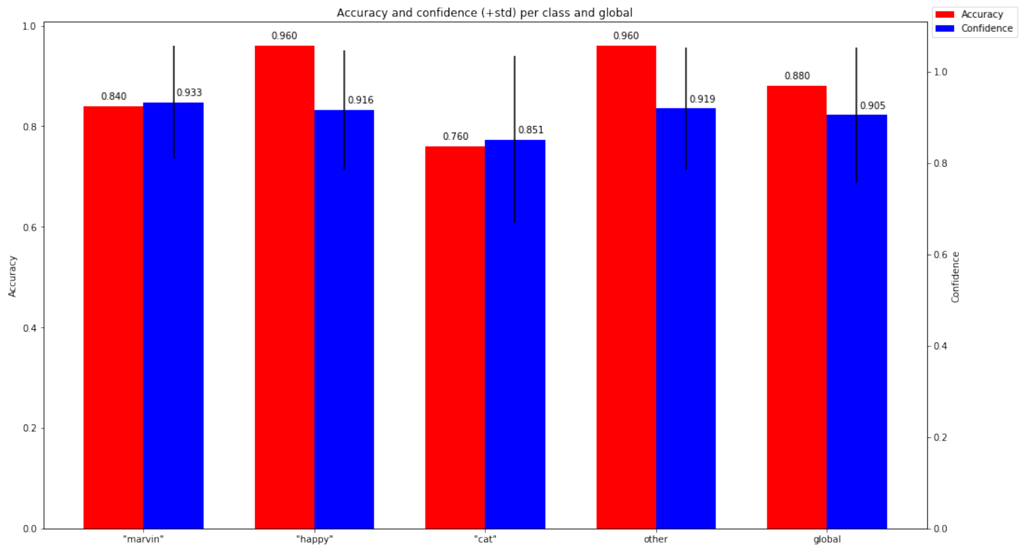
3 Results

3.1 Round One

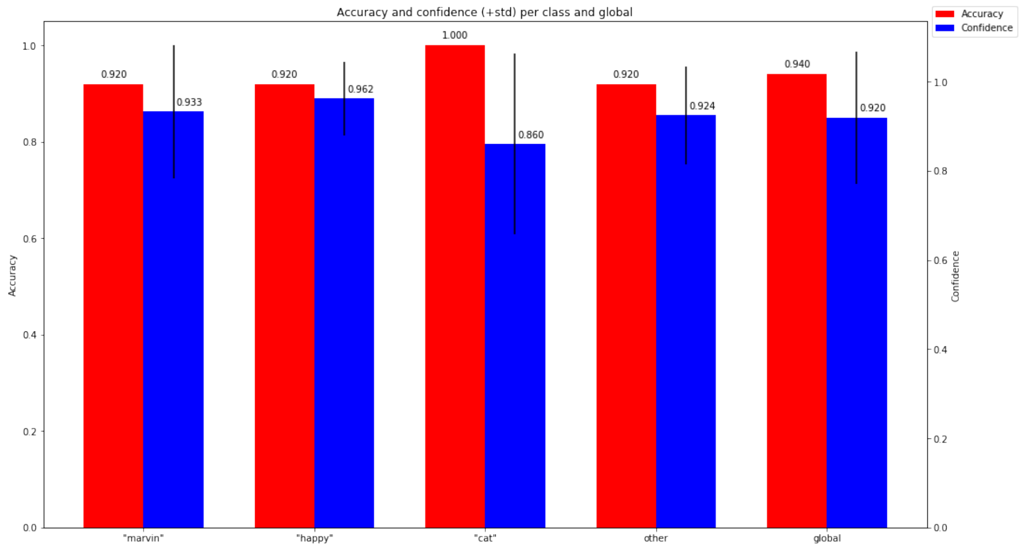
In Figure 1 we see the performance of the non-binary networks. Average performance ranges from 0.880 to 0.940 with confidences ranging from 0.905 to 0.941. Most of all, we notice that all models perform decent at the distinction between target words and distractor words. Confidences are also high, confirming the strong ability to distinct between classes for all models.

The binary versions of these models perform less well as showed in Figure 2. The first thing that stands out is that the confidence is 1.000 for all classes in all three models. Upon retrospection in Section 4 this is because of a poor choice of confidence measure. Secondly, all models seem to be able to classify *marvin* best, followed by *happy* and *cat*. All models perform bad at the distractor category. It is also noteworthy that `bcnn-tpool3` performs proportionally better at the *happy* class than other models. Finally, it seems odd that all networks have the same order of classes they score on.

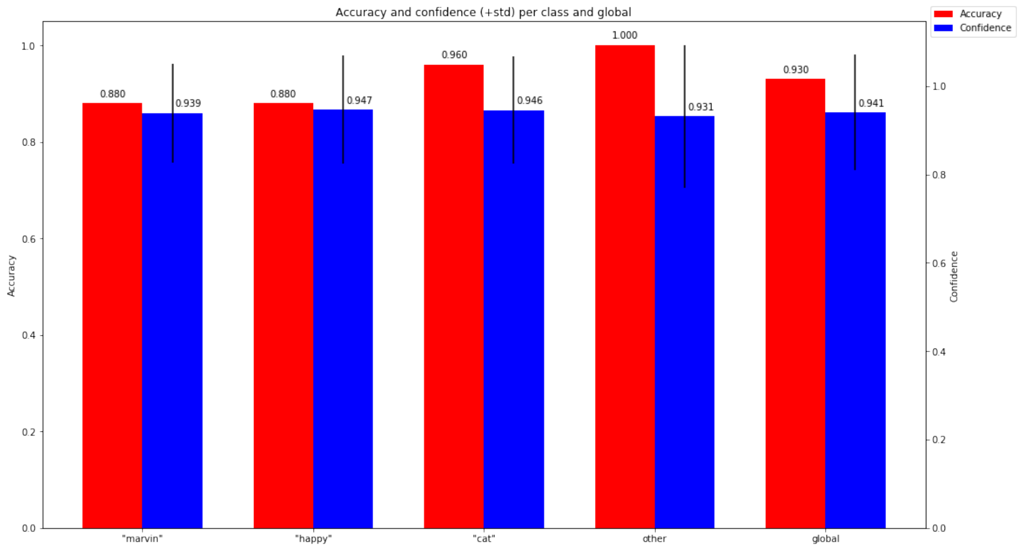
Because of the lacking performance of these models, we decided to do another round of experiments, this time with updated models. See Section 2.3.4.



(a) `cnn-trad-fpool3`

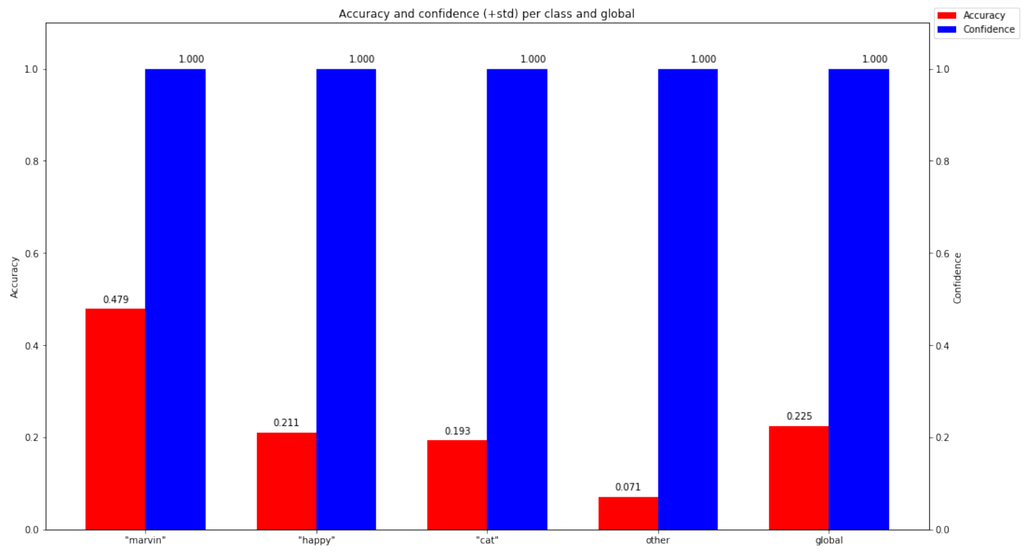


(b) `cnn-tpool3`

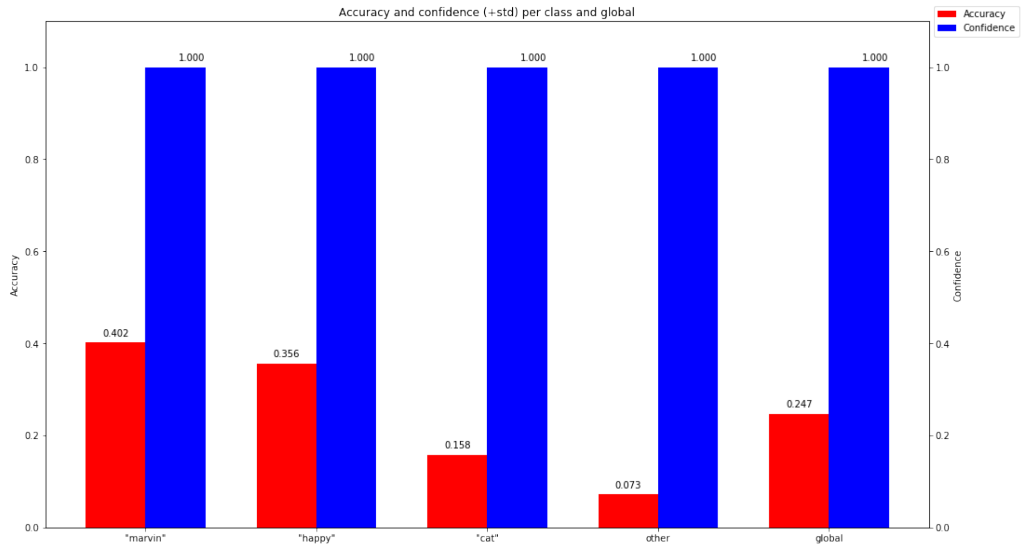


(c) `cnn-tpool2`

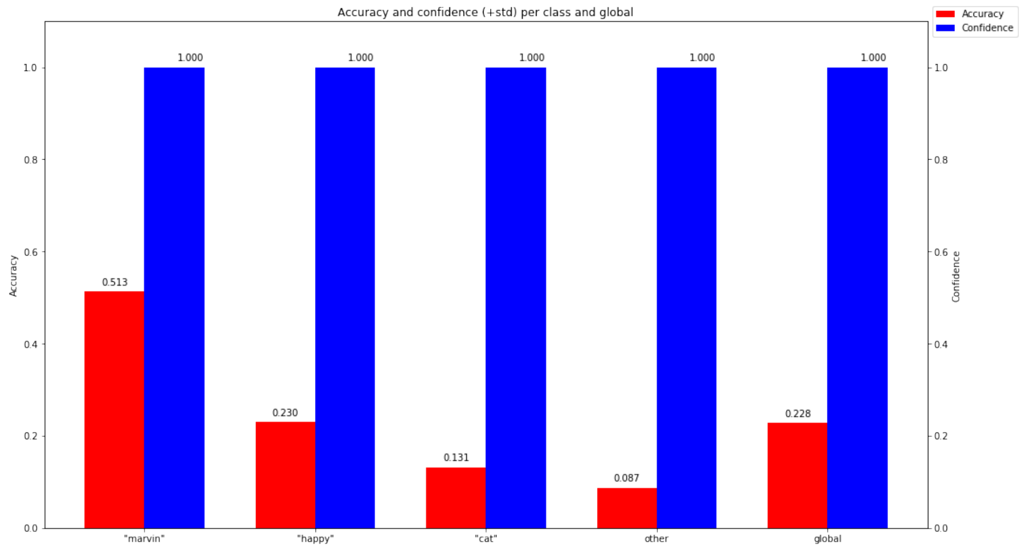
Figure 1: Accuracy and confidence (and standard deviation) per class and averaged



(a) `bcnn-trad-fpool3`



(b) `bcnn-tpool3`



(c) `bcnn-tpool2`

Figure 2: Accuracy and confidence per class and averaged over all classes.

3.2 Round Two

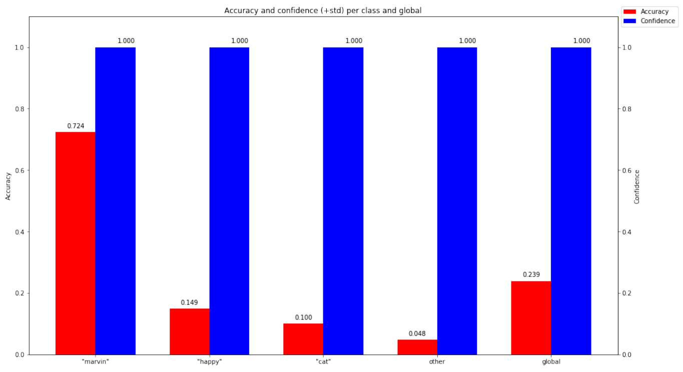
Figure 3 shows individual architectures' performance in accuracy and confidence depending on keywords. A comparison of all architectures' accuracy can be found in Figure 4. Overall, the confidence remained constant (at 1.00) across all architectures and classes. As for accuracy, there appeared to be only little variation in global values across all architectures. Another common feature among architectures was that they all reached their highest accuracy for the keyword *Marvin.* For almost all architectures, individual keywords' accuracy can be sorted in descending order from *Marvin* to *happy* to *cat* to the distractor class. Consequently, *cat* scores are the lowest for most of the architectures. The only exception is the one with more filters, where distractor scores are higher than those for *cat.*

Figure 4 describes the accuracy of the keywords for all of the binary models. For each keyword class, there is a different model that performs best. For *Marvin*, `bcnn-trad-fpool3_last-bin-layer` (Figure 3a), scores highest with an accuracy of 0.724. "Happy" is best predicted by `bcnn-tpool3` (Figure 2b) with an accuracy of 0.356. For the keyword "Cat", `bcnn-trad-fpool3` (Figure 2a) performs best with an accuracy of 0.193. `bcnn-trad-fpool3_more-filters` (Figure 3d) predicts the other category best, with an accuracy of 0.120.

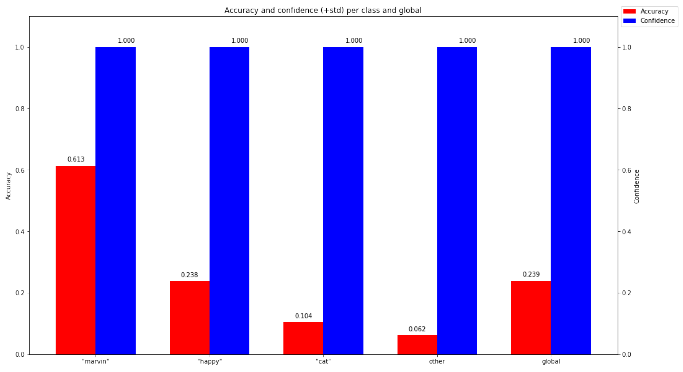
Notably, each of these models specializes on this keyword. None of the mentioned models perform equally well on the other keywords, or the other class. Also, thought there is variations in performance on the specific classes, the overall performance of the models is very comparable.

The confusion matrices (Figures 5 & 6) of all of the BCCN variations show a trend in a columned fashion. All the models predict the *Marvin* Class most often, resulting in a darker column on the left hand side. Also, in all of these results, except the confusion matrix of `bccn-trad-fpool3_more-filters`, the other class is predicted the least often.

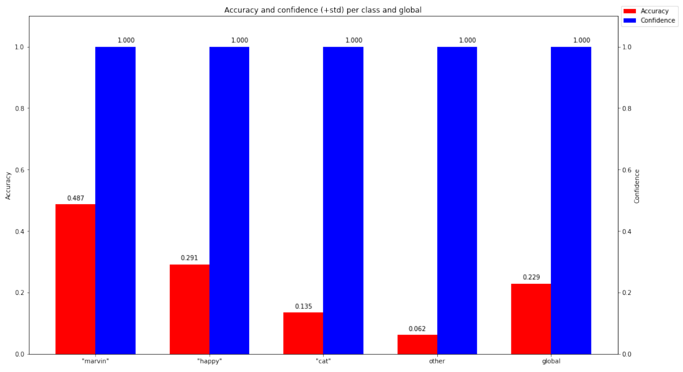
The results of the within-class variation inspection are shown in Figure 7. Notably, the standard deviation of both the average and standard deviation values for "marvin" is the lowest. The average of the standard deviations is the highest for the class 'other', whereas the class 'cat' has the highest standard deviation of all standard deviations.



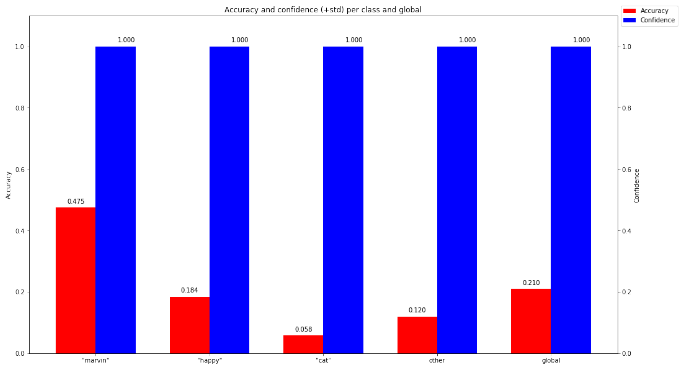
(a) `bcnn-trad-fpool3_last-bin-layer`



(b) `bcnn-trad-fpool3`
`_last-bin-layer_more-neurons`



(c) `bcnn-trad-fpool3`
`_last-bin-layer_minus-conv`



(d) `bcnn-trad-fpool3`
`_more-filters`

Figure 3: Accuracy and confidence per class and averaged over all classes..

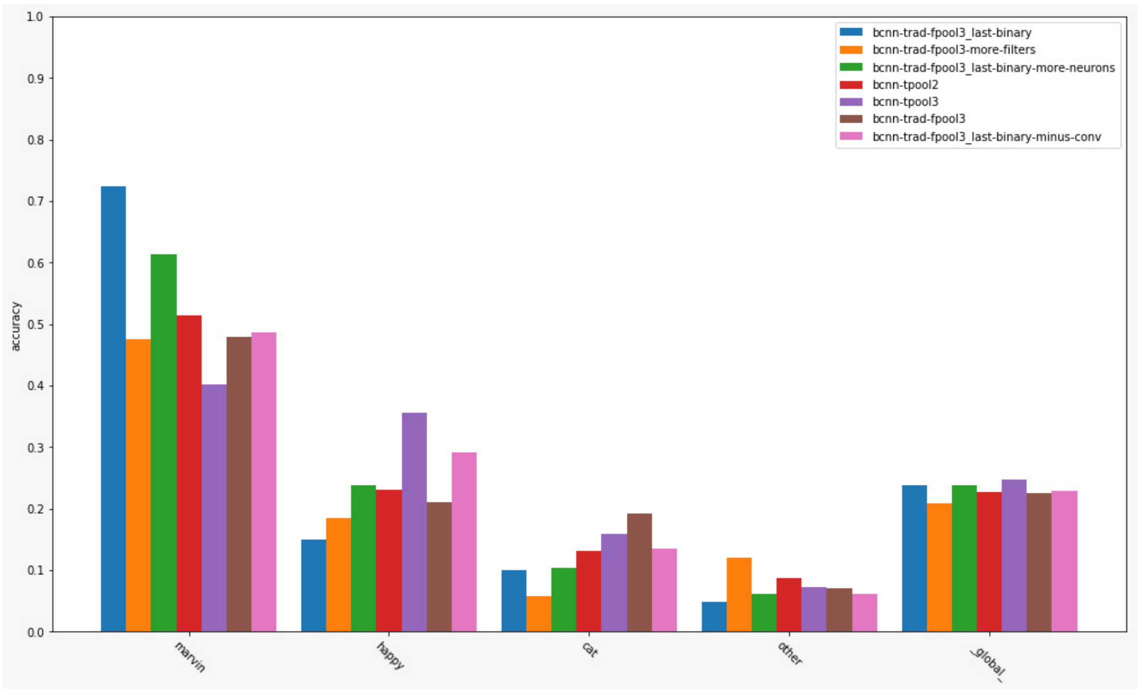
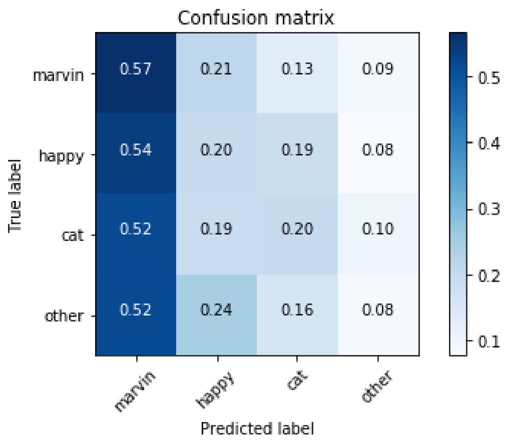
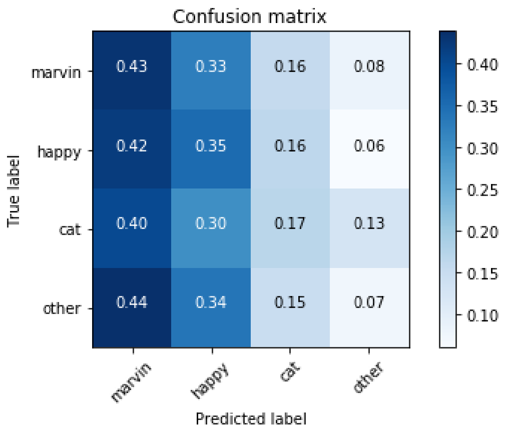


Figure 4: A comparison of accuracy between all the BCCN variations

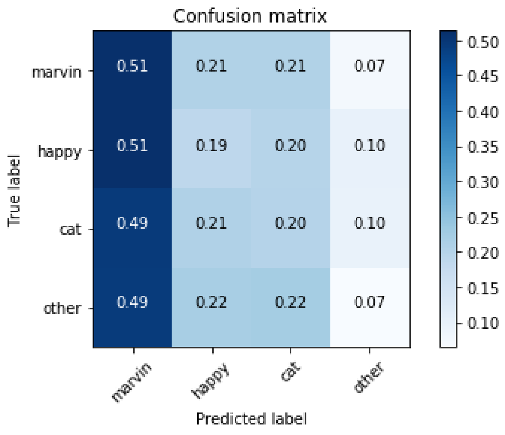
3.3 Confusion Matrices



(a) `bcnn-tpool2`



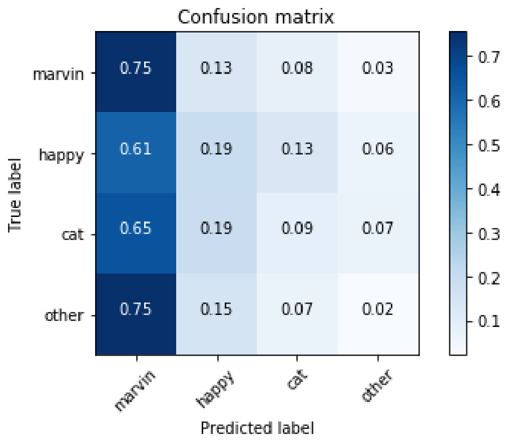
(b) `bcnn-tpool3`



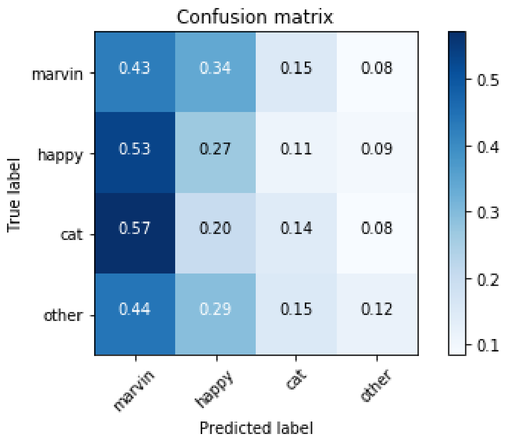
(c) `bcnn-trad-fpool3`

Figure 5: Confusion matrices of binary networks

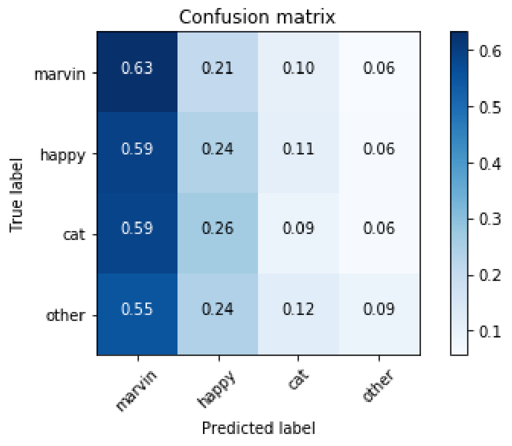
.



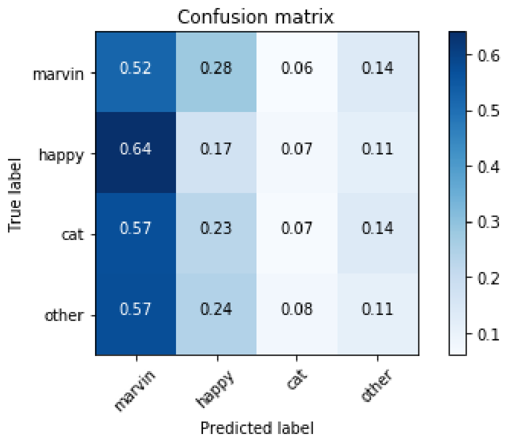
(a) `bcnn-trad-fpool3_last-bin-layer`



(b) `bcnn-trad-fpool3`
`_last-bin-layer_minus-conv`



(c) `bcnn-trad-fpool3`
`_last-bin-layer_more-neurons`



(d) `bcnn-trad-fpool3`
`_more-filters`

Figure 6: Confusion matrices of modified binary networks
.

3.4 Inter Speaker Variability

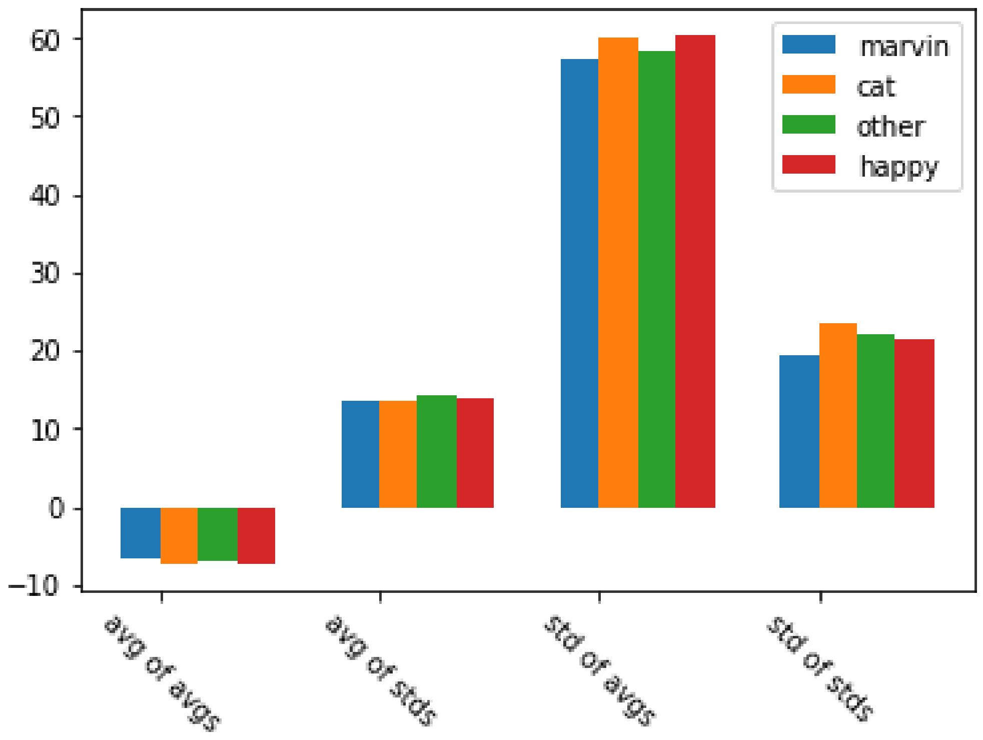


Figure 7: The within-word variation

4 Discussion

In the present study, we aimed at elucidating the effect of binarization of a CNN on its application on a keyword detection problem. More specifically, we hypothesized that the performance of a CNN should not notably differ between its canonical and its binarized form, assuming that the binarization would, however, lead to lower computational costs.

Contrary to this expectation, the binarized versions of networks overall displayed a low performance. All our binary networks were outperformed by our baseline. This result appears especially surprising in view of previous literature suggesting good performance in NNs' binarized versions. There might be several factors leading to this outcome. At first one could expect that a possible explanation could lie in the binary layer implementation we utilized here. Since we employed an already existing library [Ding, 2017], there might be passages in that code which we misinterpreted or did not correctly detect as mistakes. However, since we manually checked that code several times, this explanation does not appear likely.
Apart from the actual implementation, it might be properties of our architectures that lead to a weak performance. For example, there might be an inherent difference between the DNN that had been successfully binarized before to solve an ASR task [Xiang et al., 2017] and the CNNs used by [Sainath and Parada, 2015], which we binarized here and further modified in the second round of experiments. However, it should be noted that binarizations of CNNs in general had been successful before. On the image classification datasets MNIST, CIFAR-10 and SVHN, the binarized ConvNet architecture of [Courbariaux et al., 2016] reached almost state-of-the-art performance. Also, the binarized version of the CNN AlexNet reached an accuracy that was similar to that of its canonical form on the ImageNet classification task [Rastegari et al., 2016]. Yet, regarding such promising results, one has to keep in mind that they have been obtained in the setting of an actual image classification tasks.
From a human point of view, visualizing audio input appears to complicate the understanding of that information. This leads to the question of whether the same might hold for machines, denoting that this procedure might (partly) explain our poor performance. If this was the case, another

architecture or another representation of input more suitable for the KWS task might improve the classification accuracy.
Overall, the MFCC pixel representation of the input contains a lot of information. Therefore, it seems reasonable to assume that the information across the layers of our networks is also high. Under the general assumption of binarization leading to information loss, one characteristic feature of an architecture with only the last layer being binary should be that it models the input information more properly than others. However, in our experiment, the corresponding networks still had a low performance.
In a nutshell, there are various levels on which our approach could have caused the unexpected performance appearing in our results. However, a single error source cannot be identified.

Apart from our initial hypothesis, one striking-out finding of the present study is that the networks seemed to specialize on one keyword. A possible cause for this outcome is that the binarization might lead to less information being represented by an individual network. This leads to the assumption that, aiming at a reduction of the loss, the network might in fact focus on a single feature of the whole problem.
Notably, the more filters network yielded the best performance on the 'other' class. As shown by the inter-speaker variability analysis, this class has the highest average standard deviation. This does not seem surprising, given that this is the class with the highest variance, since it contains multiple words. In the more filters archictecture, information is projected from the first layer to a larger binary layer. Consequently, the number of filters in that binary layer is higher than that of the first layer. In this way, a binary encoding of that projected information could be created by the binary layer. The network might therefore be able to best represent a broader variety of words.
When inspecting the results on single keywords in more detail, it becomes apparent that all networks had a high performance when detecting 'marvin'. A possible explanation for this phenomenon might be that the amount of variation as shown by the inter-speaker difference for this keyword is lower than for others. This could make it easier for a network to properly detect the corresponding keyword, especially given the assumption that BNNs have less information available to create proper fits to words than their canonical counterparts. Another important question is why the scores for "cat" were the lowest for almost all architectures. Since it is the shortest word containing only one syllable, the other words might have been learned better due to their length. Further, we did not find a clear explanation for why the keywords' order of accuracy was almost the same for all networks.

5 Conclusion

In this study, we were not able to show that the binarized version of a CNN was performing similarly to its canonical form when used to solve the same KWS task. More precisely, our results suggest that directly binarizing already existing architectures and applying them to such a problem does not necessarily lead to comparable performance. Furthermore, solely increasing the amount of parameters of respective architectures does not guarantee a rise in performance. Rather, it is crucial to pay attention to which set of parameters' size should be increased. Overall, since our approach had several properties that might have lead to this low performance, we also were not able to find a single cause of this outcome in our binarized networks.

The present study, however, did reveal a specialization on a single class of keywords in the networks. In the present set of experiments, we had four classes of keywords. Thus, if there was less information to encode overall, it can be expected that the performance would be better. Considering our results in this paper, it still remains unclear how the models we used would generalize if the number of classes was increased instead. In this case, there would be the same capacity available to store patterns that might be indicators for the keyword classes, but there would also be more classes to differentiate between at the same time. The models we used in our experiments had a high difference between their corresponding performances for individual keywords. Therefore, we also cannot make generalizations about other additional keywords.

Altogether, our results did not support our initial hypothesis about the performance of BNNs on KWS tasks. Nevertheless, they suggest several aspects of investigation for future research.

5.1 Future Research

The present study only binarized a limited amount of architectures. Thus, in order to reliably draw conclusions on binarizations' effect on KWS problems, it seems reasonable to binarize other existing architectures with a better performance. In addition, creating new architectures upon consideration of this study's finding on information projection which are then in turn binarized might provide valuable insights.
To improve performance, one could consider changing the preprocessing procedure. Instead of a MFCC representation of the input, another representation might actually suit binarized networks better. To find such an import format, one could experiment with both visualizations and other representations of auditory input and utilize representations that contain less information.
Since we could not make generalizations about additional keywords, future research should include bigger sets of classes. In these sets, the inter-speaker variability should be systematically varied. In addition, word length should also be changed to examine its effect on classification.
Also, it is necessary to find a suitable way of modelling confidence values in binary architectures. Last but not least, future research should in fact measure computational demands of NNs and their binarized versions when applying them to KWS problems.

References

[Bengio et al., 2013] Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

[Bokkers et al., 2018] Bokkers, L., Persad, B., Winkler, J., and van Deelen, B. (2018). Source code for this project. `https://github.com/Damacustas/ASR-2018`.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. `https://keras.io`.

[Courbariaux et al., 2016] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*.

[Ding, 2017] Ding, K. (2017). Keras binary layers. `https://github.com/DingKe/nn_playground/tree/master/binarynet`.

[Garofolo et al., 1993] Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., and Pallett, D. S. (1993). Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon technical report n*, 93.

[Golik et al., 2015] Golik, P., Tüske, Z., Schlüter, R., and Ney, H. (2015). Convolutional neural networks for acoustic modeling of raw time signal in lvcsr. In *Sixteenth Annual Conference of the International Speech Communication Association*.

[Hou et al., 2016] Hou, J., Xie, L., and Fu, Z. (2016). Investigating neural network based query-by-example keyword spotting approach for personalized wake-up word detection in mandarin chinese. In *Chinese Spoken Language Processing (ISCSLP), 2016 10th International Symposium on*, pages 1–5. IEEE.

[Këpuska and Klein, 2009] Këpuska, V. and Klein, T. (2009). A novel wake-up-word speech recognition system, wake-up-word recognition task, technology and evaluation. *Nonlinear Analysis: Theory, Methods & Applications*, 71(12):e2772–e2789.

[LeCun et al., 1995] LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

[Rastegari et al., 2016] Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer.

[Sainath and Parada, 2015] Sainath, T. N. and Parada, C. (2015). Convolutional Neural Networks for Small-footprint Keyword Spotting. *Proceedings INTERSPEECH*, pages 1478–1482.

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

[Xiang et al., 2017] Xiang, X., Qian, Y., and Yu, K. (2017). Binary Deep Neural Networks for Speech Recognition. *Interspeech 2017*, 1:533–537.

[Zhang, 2004] Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM.