Efficient Keyword Spotting through Hardware-Aware Conditional Execution of Deep Neural Networks

J.S.P Giraldo *, Chris O'Connor †, Marian Verhelst *
* ESAT-MICAS KU Leuven † Skyworks Inc.

***Abstract*—Keyword spotting is a task that requires ultra-low power due to its always-on operation. State-of-the-art approaches achieve this by drastically pruning model size, yet often at the expense of accuracy. This work tackles this fundamental conflict between operating efficiency and accuracy in three ways: 1.) Exploiting dynamic neural network cascades for keyword spotting using an end-to-end hardware-aware training; 2.) Deriving the optimal number of stages and stage dimensions in function of the input class distributions; 3.) Using the low-latency response of the first stage for speculative execution of the later stages, training the dynamic cascade through a hardware-aware cost function. Results show the framework can generate cascade models optimized in function of the class distribution (background noise, target keywords and other-speech), reducing computational cost by 87% for always-on operation while maintaining the baseline accuracy of the most complex model of the cascade. On top of this, the hardware-aware speculative execution provides an additional 2x energy savings over the non-speculative case.**

***Index Terms*—Keyword Spotting, Deep Learning, Hardware-efficient Inference.**

I. Introduction

Over the last years, the field of machine learning has been revolutionized due to the development of deep learning algorithms such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. They have been applied successfully to different AI tasks including image recognition, speech processing, natural language processing, etc. achieving close to human level accuracy. Due to the big amount of computations and large memory bandwidth that is associated with these kernels, their execution, however, has mainly been restricted to cloud servers. The rise of mobile devices has recently encouraged the use of embedded deep learning for improving human-machine interfaces [1]. One of the examples of such technologies is keyword spotting or voice command recognition, which allows a total hands-free experience for the user by controlling several device functions through speech. For instance, commercial applications such as Siri, Amazon Echo or Cortana are usually controlled by one or a few 'wake-up' words, activating a complex speech processing environment. Since voice command recognition often acts as an always-on front-end layer, its power consumption is a crucial parameter.

Several deep learning algorithms have been applied for keyword spotting, such as Deep Neural Networks (DNNs) [2], CNNs [3] or Convolutional Recurrent Neural Networks (CRNNs) [4]. In general, higher accuracy for such models is correlated with a higher number of computations, imposing a trade-off when selecting a classifier for an embedded device. For example, working on the same dataset and keyword spotting task, advanced algorithms like CRNNs and Depthwise Convolutional Neural Networks (DS-CNNs) achieve 11% better accuracy in comparison with conventional DNNs but at an expense of about 30x more computations [5]. The problem is augmented due to the diminishing returns from increasing model size, which means achieving accuracy improvements is increasingly more costly in terms of computations [6].

To overcome this bottleneck, this work proposes a cascaded dynamic classifier for keyword spotting trained end-to-end, providing state-of-the-art accuracy while minimizing the average number of computations per inference. The cost of computation as well as the input class distribution (percentage of background noise, keywords and other-speech) are incorporated in the optimization and training of the cascade. Cheap and efficient first stages process most part of the input speech whereas the most complex stages are executed conditionally depending on the output of the front-end models. Furthermore, speculative execution of the most complex stage is proposed, in order to start its computation in advance. This allows to relax the latency constraints, and as such reduce the operating frequency and consequently power consumption in embedded devices.

Although previous approaches have already explored the use of cascaded execution towards improved energy efficiency for keyword spotting [7] [8], this work is the first one to incorporate a complete hardware-aware and data-distribution aware end-to-end training for such structure. Therefore, the main contributions of this work can be summarized as follows:

- An end-to-end training framework of a cascade classifier for keyword spotting that incorporates both the model's hardware cost and the input class distribution knowledge for model optimization.
- An exhaustive hyperparameter search towards Pareto-optimal models (in terms of accuracy and computational cost) adapted for different input class statistics.
- The introduction of speculative execution for the last cascade stage, again optimized through an end-to-end hardware-aware training for further energy reduction.

The paper is structured as follows: Section II introduces

some background knowledge on the application of deep learning for keyword spotting. Section III introduces the envisioned cascade classifiers for keyword spotting, with their improved training. Section IV augments the proposed architecture with speculative execution. Results from training the hardware-aware cascades, as well as from the speculatively executed networks, are shown in Section V. Finally, Section VI concludes this work.

II. Related Work

A. Neural Networks for Keyword Spotting

Keyword Spotting is traditionally carried out through a pipeline composed of a feature extraction module, followed by a machine learning classifier. Mel Frequency Cepstral Coefficients (MFCCs) are commonly used as features. These values are extracted from windows of audio samples, typically 25-32 ms windows with a 10-16 ms overlap between subsequent windows [9]. After feature extraction, either a feed-forward or a Recurrent Neural Network (RNN) model is deployed. The advantage of RNNs rely on their ability to work with only one feature vector per time step, reducing the amount of information to save between inferences [10]. Feedforward models, on the other hand, show opportunities for increased accuracy, yet they need to be provided with all the feature vectors of a given time window (e.g. 1 second of context information) [2]. This typically results in more computationally complex models, as well as larger memory requirements. Due to restrictions in terms of energy budget, the networks that can be deployed on commercial embedded systems are however severely constrained in terms of complexity, limiting the achievable application accuracy.

State-of-the-art deep learning models have been successfully applied to embedded keyword spotting in recent years. One of the first breakthroughs [2] demonstrated significant performance improvements compared to Hidden Markov Models (HMM) by using DNNs. Working on these foundations, [3] leveraged a CNN to further improve accuracy. The use of hybrid neural networks such as CRNNs for keyword spotting has subsequently shown impressive results in terms of model efficiency and accuracy [4]. CRNNs take advantage of the spatial pattern recognition from feed-forward networks and sequential learning from RNNs, obtaining state-of-the-art accuracy for voice command recognition [5].

B. Cascade Classifiers

The traditional use of single stage keyword spotting, where one static deep learning model is applied to every incoming time window, presents some drawbacks that limit its energy efficiency. First of all, since the graph that defines such networks is fixed after training, easy-to-classify signals (like segments of background noise) are handled in the same way as potentially difficult keyword segments. This wastes resources for easy-to-classify inputs, which could also be served by a simple and cheaper network. Secondly, for many user scenarios, the input class distribution is clearly unbalanced. For instance, in the case of always-on operation, the amount of background noise segments could exceed the duration of actual speech with one or more orders of magnitude. Likewise, the frequency of the target commands compared to other natural speech could be significantly lower. These two observations make it obvious that the average number of computations can be considerably decreased if the system would operate in a data-driven way, in which the most common and simple inputs would be handled by models with reduced complexity, whereas the most difficult cases are still processed by a more complex model.

Some approaches have explored the use of such cascade execution of neural networks for different AI tasks, mainly focused on image recognition. [11] shows a feed-forward network for image classification capable of generating output predictions in early layers without having to execute the whole model. Likewise, [12] presents a methodology for training control edges and inference nodes jointly through a Reinforcement Learning (RL) framework. Despite the recent work developed in this area, the topic has been scarcely explored for voice command recognition. While some keyword spotting systems use voice activity detection as a wake-up stage [13], these two stages are trained separately, losing opportunities for joint co-optimization. Alternatively, [7] implemented a keyword spotting system using a cascade composed of a DNN and a SVM working on hand-crafted features. [8] shows the implementation of a cascade classifier using a small and a big DNN whose activation is regulated by tuning the output threshold of the first stage. Yet, all these approaches rely on the ensemble of individually pre-trained models, which demands a high amount of engineering to empirically tune the activation thresholds between stages. This work will overcome this through a jointly trained end-to-end classification pipeline for keyword spotting, taking both hardware- and data-dependencies into account.

III. Hardware and Input-aware Cascade Classification for Keyword Spotting

A. High-level operation

This section presents a cascade classifier for keyword spotting, optimized for computational efficiency and accuracy. The functional objective of this system is to detect a set of keywords discriminating them from background noise and other natural speech. In a cascade (Figure 1), each classifier stage carries out two basic operations: classify the current

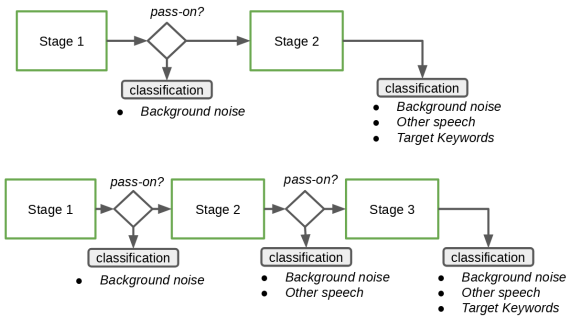

Fig. 1. Two and three stage embedded keyword spotting system

speech segment as one of the possible labels, or pass-on the instance to be processed by the next stage in the cascade. The passing-on of examples allows to rely on further stages for the inference of more difficult input samples, avoiding possible mistakes. As depicted in figure 1, two frameworks are explored: two stage and three stage cascade networks. In the first case, a first stage classifying background noise is coupled with an additional classifier in charge of classifying all the target keywords, other-speech, or previously missed background noise samples. As remarked before, stage 1 can output a background noise label for the input or it can pass-on the example to the next stage for further processing. For the second framework, the first stage, again in charge of detecting noise or passing-on, is stacked with an intermediate model that classifies other-speech and noise, before going to the final, most complex, stage. The final stage, as in the first framework, targets all the possible labels.

B. Training

The optimization objective for the cascaded classifier is to find the neural network topology and its weights that maximize accuracy as well as minimize the average number of computations per inference. As can be deduced from the structures depicted in figure 1, these kind of networks are not directly differentiable through standard back-propagation, due to the varying execution paths that any of the examples can take. More specifically, the inputs that are handled by the second or third models are a subgroup of the ones received by the first stage. To overcome the difficulties from non-differentiable paths during training, a reinforcement learning approach, inspired by [12], is deployed when training the full model end-to-end.

As shown in figure 2, the first stages (Stage 1 for the 2-stage network and Stages 1 and 2 for the 3-stage network) are handled as RL agents. They can carry out two possible actions: classify the current speech segment as one of the $n-1$ possible labels of that stage, or pass the input features to the next stage for further processing. For this purpose, a neural network generates n Q-values that represent the predicted reward of taking a specific action (e.g, classify instance as class 0, pass-on, etc). The network that generates the Q-values is trained based on a reward given by the learning process, as will be explained later. In the forward pass, the action with the highest Q-value is selected. If the decision is to classify the instance as one of the possible labels, the inference is finished. Otherwise, the next stage is activated.

During training, a reward is given to each one of the decisions of a stage. This reward reflects the dual goal of obtaining accurate inferences, while minimizing the number of computations. For this purpose, the reward r given for any decision a at stage i over the current speech segment s (i.e the group of feature vectors that compose a segment) can be described as:

$$r(s,a) = \begin{cases} (\lambda)(P(s,a)) + (1-\lambda)(\frac{1}{\beta C(s,a)}), & \text{if } P(s,a) = 1 \\ 0, & \text{if } P(s,a) = 0 \end{cases}$$

Here, the value $P(s,a)$ is the expected accuracy which is a binary value for the current speech segment s (0 if it is an incorrect inference and 1 otherwise), $C(s,a)$ accounts for the expected total number of MAC (multiply-accumulate) operations carried out for the inference pass, β is a scaling factor for each class reflecting the unbalanced input data distribution of the real-life scenario (see further), whereas λ is a hyper-parameter, which allows to steer the accuracy/computation trade-off. As can be observed, if λ increases, the reward encourages accurate predictions; inversely it penalizes computationally expensive decisions.

In the referred reward function, the number of expected MAC operations for the given speech segment s is used as a metric for estimating the cost of an inference. For instance, if one speech segment is passed through three stages, the total cost associated with this inference would be the total number of MAC operations carried out by all 3 stages. On the other hand, if a segment is classified by the first stage, the computational cost is equal to the inference cost of just the model of the first stage. To incorporate the expected class distribution of the target real-life scenario into the reward function (while still obtaining a balanced accuracy across all classes) the number of samples per class is equally distributed in the training set; yet, for each inference of a segment s, the term $C(s,a)$ is scaled with the factor β, which reflects the expected statistical presence of the class in the targeted input data. For instance, if for an specific application the distribution between target keywords, other-speech and background noise is expected to be 1:5:10 respectively, the factor β for background noise segments would be 10 and for keyword segments would be 1. In contrast, the variable $P(s,a)$ is not scaled according to the input data distribution in order to maintain high accuracy for every possible class irrespective of the distribution (otherwise under-represented classes could suffer from much worse accuracy in comparison with the most common classes).

As such, the input- and hardware-aware training of a stage is reduced to a regression task, whose objective is to minimize the difference between the Q-function obtained when running the sample through the network, and the reward actually obtained for every possible action, resulting in the following loss function $L(s,a)$:

$$L(s,a) = (Q(s,a) - r(s,a))^2 \tag{2}$$

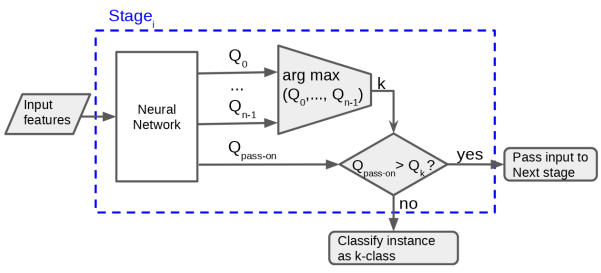

Fig. 2. Implementation of a cascade stage as a reinforcement learning agent

With s being the current speech segment and a the action. As can be observed, for each decision stage a neural network acts as a regression model generating Q-values that aim to fit the empirical reward r given by the training environment.

Since the last stage of the cascades only carries out classifications, standard back-propagation with cross-entropy loss is used. In summary, all the cascaded stages are jointly trained minimizing the correspondent loss for each model, taking hardware-cost and input-distribution aspects into account. After training, an optimal policy is found for every new input processed, by selecting the action in each stage that maximizes the expected reward. The results of the proposed approach are presented in section V.

IV. Speculative Execution

A. Motivation

The last stages of a cascade classifier are typically only activated at the end of an evaluation window, when the first stage has been executed on the complete buffer of feature vectors (Fig. 3.a). While such scheme is computationally efficient, this has serious latency implications. Assume, for instance, the length of each speech evaluation window is 1 second, and the system has a 100 ms latency constraint. This would imply only 100 ms is available to execute the most complex models of the latter stages. In contrast, a speculative execution could already start earlier, i.e. activating the most complex stages after a speculative assumption of the presence of the keyword/key-phrase (Figure 3.b). In the example presented in figure 3, where the voice command is "turn on TV", the presence of "turn" in the middle of the phrase could be an indication to activate the most complex stages in advance. While this would come at a computational cost due to false wake-ups, this would at the same time bring a benefit of a relaxed latency constraint. In our example, the computation time of the latter stages would be extended to 600 ms (500 ms + 100 ms). This would allow to drastically reduce the minimal operating frequency of the final processing stages, and consequently lower the power consumption through frequency/voltage scaling.

Since the presence of false positives are prone to jeopardy the potential energy savings (i.e activating the later stages too often) speculative execution of latter stages must be carefully tuned dependent on the incoming speech segment as well as the input class distribution. This again calls for a hardware and data-aware end-to-end training strategy.

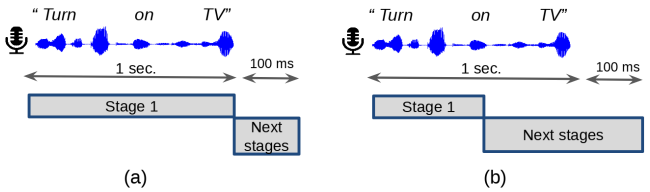

Fig. 3. Activation of the cascaded stages in function of time for (a) Standard cascade execution and (b) Speculative execution

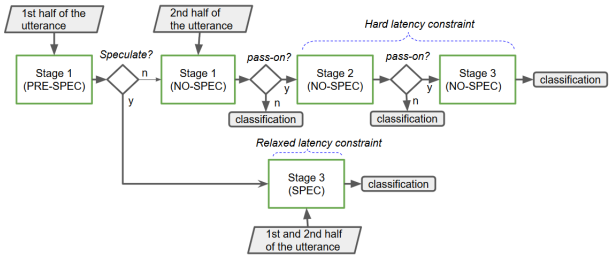

Fig. 4. Network Graph for speculative execution

B. Cascade classifier using speculation

Having in mind the presented benefits of speculation, a modified cascade network that allows to speculatively turn on the most complex stage is proposed (Figure 4). The 3-stage cascade network depicted in Figure 1 is enhanced with a speculative path that can directly activate the third stage. This decision is carried out in the middle of the execution of Stage 1. After half of the segment has been processed in real-time by Stage 1 (*Stage 1 PRE-SPEC*), two possible actions are evaluated: speculate or do not speculate. In case the Q-value for the speculate action is higher than the one for 'no speculation', this path is taken. In this case, the most complex stage starts computing the buffered half of the utterance, followed by executing in real-time the rest of the incoming feature vectors with a relaxed latency constraint (*Stage 3 SPEC*). Otherwise, in case the Q-value for 'no speculation' is higher, the non-speculative path is taken, completing the execution of the first stage (*Stage 1 NO-SPEC*) for the total duration of the utterance. After this, the Q-values for the actions *background noise* and *pass-on* of stage 1 are compared. In case the pass-on decision is chosen, the next models (*Stage 2 NO-SPEC* and potentially *Stage 3 NO-SPEC*) are executed with now a hard latency constraint, similar to the standard cascade execution.

C. Training

In order to take the speculation concept into account during training, the additional execution path and its associated reward were included. Following the framework developed in section III.b the Q-values of the two new actions for Stage 1 (PRE-SPEC) are added (*speculate* and *no speculate*). Additionally, since the relaxed latency constraint reduces the energy cost of computation in the speculative path compared to the non-speculative one, a computational cost factor α is introduced in the reward function: The cost of 1 MAC operation using the non-speculative path is weighted as the cost of α MAC operations for the speculative path. This factor α is an empirical value, dependent on the hardware platform as well as the frequency/voltage operating point.

Using the modified reward function, the whole classifier can be jointly trained through reinforcement learning, as presented in section III. Here, the speculative decision is jointly optimized for the current input class distribution, differential cost of speculation and accuracy/computation trade-off point.

V. Results

A. Dataset and experimental settings

The Google Speech Command Dataset (GSCD) [14] was used for the analysis of the proposed innovations. This database is composed of 65,000 examples of 1 second long audio using 16 ksamples/sec. Each one of the audio files corresponds to one of 30 possible commands. Likewise, samples from real-life environments, such as outdoor noise and silence are used as background noise instances. A 12-class problem was selected as benchmark, in line with keyword spotting state-of-the-art, such as [5] [9]. The objective is to classify 10 keywords: *yes,no,up,down,left,right,on,off,stop,go* plus the *other-speech* label (containing voice commands from the rest of the dataset) and the *background noise* label (containing environmental noise). The data is split into training, validation and testing using a distribution 80:10:10 respectively.

The model topology used for the first stages (Stage 1 in the case of the 2-stage network and Stage 1 and 2 for the 3-stage case) is one layer of LSTM units with a varying number of units and a layer of output neurons (one for each possible action). This decision is determined by the competitive accuracy and reduced model size of LSTMs for voice command recognition [5]. Furthermore, the frame-wise response from the LSTM stages will be exploited for implementing speculative execution.

For the last stage, a CRNN with architecture based on [5] was used. This is composed of one CNN layer with 100 filters of size 10x4, one bidirectional GRU layer with 200 units in total, one Fully Connected (FC) layer of 188 neurons and finally 12 output neurons for each class. The size of this model is 500 kB and 14 MOPs per inference. By training this architecture as a stand-alone classifier, it achieves 94.6% average accuracy on the 12-classes problem. Due to its bi-directional operation, this model is not suitable for speculative execution. For the experiments implementing speculation, a CRNN with unidirectional GRUs was used instead (2-layer GRU with 140 units in each layer, with for the rest an identical CRNN topology). The model-size and number of operations hence remains the same as the bidirectional case. Using this architecture as a stand-alone classifier, a test accuracy of 94.4% is obtained, or a marginal reduction from the CRNN using bidirectional RNNs.

The Keras framework was used as development tool, applying mean squared error for the regression models and cross-entropy for the CRNNs. An adaptive learning rate was applied with a batch-size of 128 samples, starting with a learning rate of 0.001. For feature extraction, 10 MFCCs are extracted per frame following the results obtained from [9], with a 32ms frame length and 16ms of stride. This accounts for 600 (60x10) values for 1 second of audio.

B. Standard cascade classifier

A first set of experiments will assess the benefits of the cascaded classifiers, trained using hardware- and data-awareness. This is done for a wide variety of networks through a hyper-parameter search in order to find the optimal settings for the cascade network. The targeted parameters include:

- Number of cascade stages (Between 2 and 3)
- Number of LSTM nodes in stage 1 (8,16,32,64 or 128)
- Number of LSTM nodes in the intermediate stage (8,16,32,64 or 128)
- The factor λ tuning the accuracy/computation trade-off (swept between 0 and 1 with intervals of 0.1)

To demonstrate the impact of data-awareness, three representative input class distributions were analyzed: *always-on-sensor*, *voice-assistant* and *push-to-talk*. The first setting corresponds to an always-on scenario where the system must handle long intervals of background noise. In the second distribution, corresponding to a system guarded by a voice-activity detector, the presence of background noise intervals is reduced. Finally, the push-to-talk distribution refers to an application where the user manually pushes a button when he want to say something to the device. As such, keywords are much more common and the silence intervals are reduced. The following values indicate the used proportions of classes for each one of the distributions:

- *Always-on-sensor*: Background noise 90%, other-speech 9%, Target keywords 1%.
- *Voice-assistant*: Background noise 50%, other-speech 45%, Target keywords 5%.
- *Push-to-talk*: Background noise 33.3%, other-speech 33.3%, Target keywords 33.33%.

Considering the targeted hyper-parameters and the distributions presented, the number of possible exploration combinations is 225. Each of these possibilities was trained with the hardware-aware training framework for a total of 40 epochs, which was the average number of iterations for convergence. The results for each one of the data distributions are shown in figures 5, 6 and 7. In these graphs, the average accuracy and normalized number of MAC operations (ratio of the average number of computations per inference to the total number of computations in the last stage CRNN model) are shown for every trained model. Since the addressed problem is unbalanced in terms of classes, the average accuracy across classes is used instead of the overall accuracy. Each point of the graphs corresponds to a different combination of hyper-parameters.

In figure 5, showing the results for the distribution *always-on-sensor*, it is observed that 87% of the total number of MAC operations can be saved, while maintaining the baseline CRNN accuracy of 94.6%. Specifically for this distribution, the two and three stage networks obtain similar results. The explanation for these findings can be found in the big amount of background noise examples (90% of the total time), which increases the importance of the first stage. As noted by the dashed line, a group of optimal models can be obtained according to Pareto optimality. Changing the trade-off between accuracy and computation through λ, it is possible to obtain the different optimal models along this Pareto curve. Figure 8

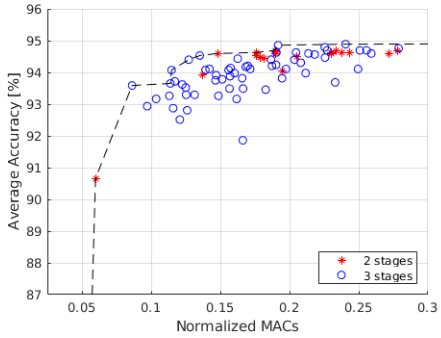

Fig. 5. Accuracy vs Normalized number of computations for cascade classifier applied to distribution *always-on-sensor* (90% noise, 9% other-speech, 1% keywords.)

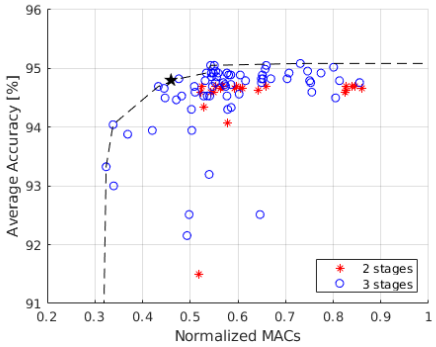

Fig. 6. Accuracy vs Normalized number of computations for cascade classifier applied to distribution *voice-assistant* (50% noise, 45% other-speech, 5% keywords.)

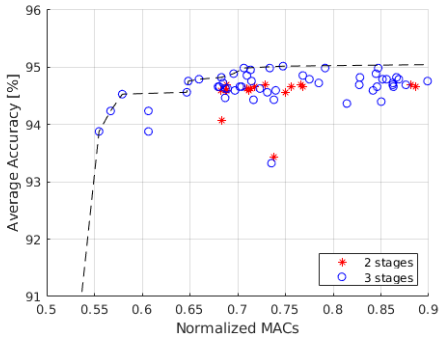

Fig. 7. Accuracy vs Normalized number of computations for cascade classifier applied to distribution *push-to-talk* (33% noise, 33% other-speech, 33% keywords.)

shows some of the Pareto-optimal models for this distribution, along with their hyper-parameters and rate of activation for the last stage. In general, it is possible to observe how increasing the activation rate for the CRNN improves accuracy but at a cost in computation overhead.

The distributions *voice-assistant* and *push-to-talk* (Figure 6 and 7, respectively) exhibit a much larger difference between the results obtained for the two and three stage networks. Due

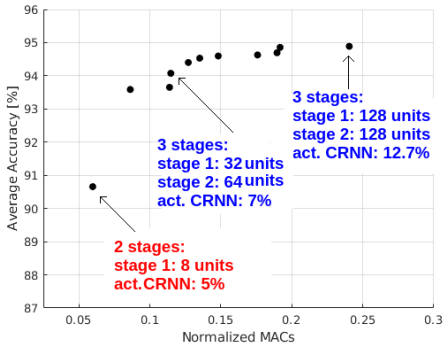

Fig. 8. Pareto optimal models for distribution *always-on-sensor* (90% noise, 9% other-speech, 1% keywords)

TABLE I
TPR, FPR AND PASS-ON RATE FOR CASCADE CLASSIFIER USING DISTRIBUTION VOICE-ASSISTANT

	Stage 1	Stage 2	Stage 3	Total
TPR noise	93.75%	18.75%	100%	100%
FNR noise	0%	0%	0%	0%
POR noise	6.25%	81.25%		
TPR other-speech		28.9%	92.3%	94.5%
FNR other-speech	0%	0%	7.7%	5.5%
POR other-speech	100%	71.1%		
TPR keywords			94.25%	94.2%
FNR keywords	0.16%	0.32%	5.75%	5.8%
POR keywords	99.84%	99.68%		

to the increasing amount of other-speech and target keywords, the topology of the second stage becomes more important. For these distributions it is possible to obtain 70% and 55% computational savings while attaining a competitive accuracy. For the *push-to-talk* distribution, the complexity reduction is less, given the large amount of target keywords, which demands the execution of the last stage more frequently.

In order to illustrate the dynamic execution of the proposed approach, one of the Pareto-optimal classifiers trained for the distribution *voice-assistant* was analyzed (marked with a star symbol in figure 6). This model is composed of 16 and 32 LSTM neurons for the first and second stage respectively, attaining an accuracy of 94.6% correspondent to the baseline CRNN accuracy. Yet, it uses only 40% of the computations needed by a stand-alone CRNN. The model size of 500 kB for the CRNN is moreover increased by only 7kB due to the additional LSTM stages. The values of pass-on rate (POR), true positive rate (TPR) and false negative rate (FNR) for each class and stage are presented in Table I according to the following definitions:

$$TPR = \frac{TP}{P} \quad (3) \quad FNR = \frac{FN}{P} \quad (4) \quad POR = \frac{PO}{P} \quad (5)$$

Where P refers to the total number of class instances that enter a stage, TP the number of class instances classified correctly, FN the number of class instances incorrectly classified and PO the number of class instances that are passed-on to the next stage. Of course, the sum of TPR, FNR and

POR must be equal to 100% for each stage and class. For the target keywords the TPR, FNR and POR for the 10 commands are averaged. As shown, the first stage is able to classifies 93.75% of noise examples correctly, whereas only 6.25% are passed to the next stage. As wanted, the first stage pass-on-rate of *other-speech* and keyword classes are close to 100%. The second stage is able to classify 18% of the total noise examples passed to it, whereas the rest is sent to the third stage. For the *other-speech* class, TPR of 28.9% is obtained in the second stage. The relatively low TPR for this stage is explained by the complexity of classifying speech not contained in the keywords set, which makes the classifier passing a significant portion of speech examples to the last stage to avoid false positives. The FNR for stages 1 and 2 is remarkably low (<0.5%) which helps for attaining high recall up to the last stage. The FNR for stage 3 is higher due to the high number of possible targets (12 targets) that this model possess. Finally, the total TPR and FNR for the whole classifier are shown. Overall, it is observed how the first and second stage effectively reduce the activation of the most complex stage given the unbalanced input data distribution, while maintaining high accuracy for each one of the classes.

C. Speculative Execution

Assuming a maximum application latency of 100 ms, and given the duration of the audio segments in the GSCD (around 1 second), speculative execution was allowed to be initiated in the middle of the utterance, allowing to relax the timing constraints to about 600 ms. This 6x relaxation of the latency constraint is translated in a 6x reduction in operating frequency for the speculative path. Taking into account information from previous heterogeneous processors [15], the factor α (referred in section IV) was set to 4 which means the energy cost of a MAC operation carried out through the non-speculative path is weighted as 4x the energy cost of a MAC using the speculative-path (Figure 4).

To evaluate the effectiveness of the technique, a cascade classifier was trained supporting speculative execution for different accuracy/computation trade-offs. One of the optimal networks obtained previously for data distribution *voice-assistant* (shown with a star in figure 6) was used as a starting point. The model contains 16 and 32 LSTM units in the first and second stage respecivelly as well as an unidirectional CRNN. Figure 9 shows the accuracy vs. normalized energy per inference (normalized to the energy cost of the CRNN standalone) for speculative and non-speculative execution using this topology. The factor λ was swept in order to find different Pareto trade-off points. As depicted in the graph, it is possible to again achieve 94.5% accuracy, with an additional 2x energy savings compared to a cascade classifier without speculation.

One of the Pareto-optimal models found through this parameter search was selected for further analysis (marked with a star symbol in figure 9). This model achieves an average accuracy of 94.5%, using only 17% of the expected energy associated with the CRNN used. The pass-on rates for its speculative and non-speculative paths are shown in table II.

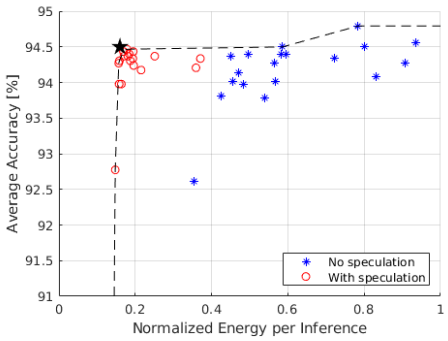

Fig. 9. Accuracy vs Normalized number of computations for Speculative Execution using a network composed of (16 LSTM units - 32 LSTM units - CRNN) for distribution *voice-assistant*.

TABLE II
SPECULATIVE (SPEC) AND NO SPECULATIVE (NO-SPEC) PASS-ON-RATE (POR)

	SPEC POR	NO-SPEC POR
Noise	8.98 %	91.02%
Keywords	92.80%	7.2%
Other-speech	92.18%	7.82%

For each class the table shows the percentage of examples that take the speculative path and the no-speculative path. As shown, the speculative path is actively used by the target keywords and *other-speech*, reaching a POR of 92.8% and 92.18% respectively. This is explained by the relatively cheap cost of the MAC operations following the speculative path, which makes the optimizer scheduling more examples to the most complex classifier. Only 8.98% of the noise examples are sent to the speculative path. Also this is expected and optimal, as these samples are very unlikely to propagate to stage 2 or 3, and hence have a lower cost in the non-speculative path than in the speculative path. The TPR, FNR and POR for the stages after the speculation decision are shown in table III using the nomenclature of figure 4. The referred metrics are shown also for Stage 3 (for the speculative and non-speculative path together) and for the total network. Stage 1 through the non-speculative path has a POR of 6% for background noise, classifying 94% of the noise examples in this phase. Stage 2 through the non-speculative path is able to discriminate around 14.28% of the *other-speech* examples. As observed, the real-time detection of speech in the first part of the utterance, allows to spot in advance potential keyword segments, which in turn relaxes the latency constraint and consequently the average energy per inference.

D. Comparison with state of the art

A comparison of the proposed cascade networks with other state-of-the-art deep learning models for keyword spotting in terms of accuracy and computational complexity is presented in figure 10. Taking into consideration the optimized models found in [5] for the Google Speech Command Dataset, the average accuracies obtained for CNN, LSTM, GRU, CRNN and DS-CNN on the 12-classes problem are shown. In the

TABLE III
TPR, FPR AND PASS-ON RATE USING SPECULATIVE EXECUTION FOR DISTRIBUTION VOICE-ASSISTANT

	Stage 1 (NO-SPEC)	Stage 2 (NO-SPEC)	Stage 3 (NO-SPEC + SPEC)	Total
TPR noise	94%	0%	97.29%	99.6%
FNR noise	0%	0%	2.71%	0.4%
POR noise	6%	100%		
TPR other-speech		14.28%	89.76%	91.79%
FNR other-speech	0.54%	0%	10.24%	8.21%
POR other-speech	99.46%	85.72%		
TPR keywords			94.55%	94.37%
FNR keywords	2%	0%	5.45%	5.63%
POR keywords	98%	100%		

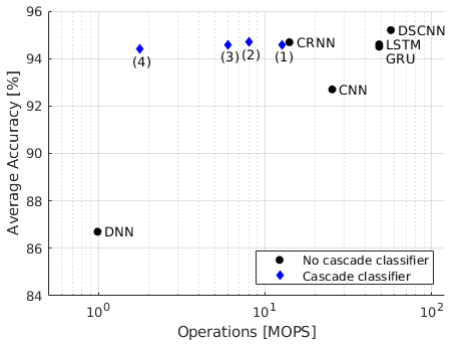

Fig. 10. Accuracy vs Number of operations per second under real-time operation for no-cascade and cascade classifiers using distributions: (1) Balanced (2) *push-to-talk* (3) *voice-assistant* (4) *always-on-sensor*

referred work [5], through a complete hyperparameter search, some efficient and accurate networks are obtained for each one of the algorithms mentioned. Since these standard deep learning models are not adaptable to the unbalanced distributions, the number of operations per inference is constant. Some of the Pareto-optimal cascade models found in this work for the three targeted distributions are also depicted. All the networks present in the graph have a similar average model size of 500kB using 8-bits weights. The comparison shows that the proposed hardware-aware cascade networks attain average accuracy close to CRNN because of the use of this model as its last stage. Yet, the proposed framework achieves this accuracy requiring only 2 MOPs for real-time operation under the most skewed distribution, compared to the 14 MOPs for the stand-alone SotA CRNN. Hence, using a cascade network, the average number of operations per inference automatically adapts to the input data statistics, decreasing significantly the computational overhead, while maintaining state-of-the-art accuracy.

VI. CONCLUSIONS

In this paper, a framework for hardware-aware and data distribution aware cascaded classification using hybrid neural networks is developed and applied to multi-keyword spotting. A hyper-parameter search combined with hardware-aware training using RL allows to find the optimal cascaded models for specific input data distributions, showing the computational savings that can be obtained with such structures. For an always-on scenario, the cascade classifier is able to save up to 87% of MAC operations through selective execution of neural networks, while maintaining the baseline accuracy of the most complex model used. Furthermore, speculative conditional execution is proposed as a way to exploit the low latency response from the RNN stages, allowing to execute later stages speculatively with a low operating voltage/frequency without sacrificing overall classification latency. This allows up to 2x additional energy savings. The presented framework brings always-on keyword spotting closer to the limits of embedded execution on resource constrained devices.

REFERENCES

[1] N. D. Lane and P. Georgiev, "Can deep learning revolutionize mobile sensing?" in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. ACM, 2015, pp. 117–122.

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

[3] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," 2015.

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

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

[6] A. L. Maas, A. Y. Hannun, C. T. Lengerich, P. Qi, D. Jurafsky, and A. Y. Ng, "Increasing deep neural network acoustic model size for large vocabulary continuous speech recognition," *arXiv preprint arXiv:1406.7806*, 2014.

[7] M. Sun, V. Nagaraja, B. Hoffmeister, and S. Vitaladevuni, "Model shrinking for embedded keyword spotting," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 369–374.

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

[9] M. Shahnawaz, E. Plebani, I. Guaneri, D. Pau, and M. Marcon, "Studying the effects of feature extraction settings on the accuracy and memory requirements of neural networks for keyword spotting," in *2018 IEEE 8th International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*. IEEE, 2018, pp. 1–6.

[10] M. Sun, A. Raju, G. Tucker, S. Panchapagesan, G. Fu, A. Mandal, S. Matsoukas, N. Strom, and S. Vitaladevuni, "Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting," in *2016 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2016, pp. 474–480.

[11] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[12] L. Liu and J. Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[13] G. Chen, C. Parada, and T. N. Sainath, "Query-by-example keyword spotting using long short-term memory networks," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 5236–5240.

[14] P. Warden, "Speech commands: A public dataset for single-word speech recognition," *Dataset available from http://download. tensorflow. org/data/speech_commands_v0*, vol. 1, 2017.

[15] H. T. Mair, G. Gammie, A. Wang, R. Lagerquist, C. Chung, S. Gururajarao, P. Kao, A. Rajagopalan, A. Saha, A. Jain *et al.*, "4.3 a 20nm 2.5 ghz ultra-low-power tri-cluster cpu subsystem with adaptive power allocation for optimal mobile soc performance," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2016, pp. 76–77.