Streaming keyword spotting on mobile devices

Oleg Rybakov[1], *Natasha Kononenko*[1], *Niranjan Subrahmanya*[2], *Mirkó Visontai*[2], *Stella Laurenzo*[1]

[1]Google Research [2]Google Speech

`{rybakov, natashaknk, sniranjan, mirkov, laurenzo}@google.com`

Abstract

In this work we explore the latency and accuracy of keyword spotting (KWS) models in streaming and non-streaming modes on mobile phones. NN model conversion from non-streaming mode (model receives the whole input sequence and then returns the classification result) to streaming mode (model receives portion of the input sequence and classifies it incrementally) may require manual model rewriting. We address this by designing a Tensorflow/Keras based library which allows automatic conversion of non-streaming models to streaming ones with minimum effort. With this library we benchmark multiple KWS models in both streaming and non-streaming modes on mobile phones and demonstrate different tradeoffs between latency and accuracy. We also explore novel KWS models with multi-head attention which reduce the classification error over the state-of-art by 10% on Google speech commands data sets V2. The streaming library with all experiments is open-sourced. [1]

Index Terms: speech recognition, keyword spotting, on-device inference

1. Introduction

Research and development of neural networks has many steps: data collection, model design and training, model latency or memory footprint optimization, model conversion to inference mode and execution of the model on different hardware, including mobile devices. In this work we are focused on the last three steps: model optimization, model conversion to inference mode and running it on mobile devices. A common method of model optimization is quantization [1, 2] allowing to reduce both model size and latency. It can be applied to many problems, including computer vision [1] and speech recognition [3]. Another optimization approach is the transformation of the NN model (which was used during training) into another NN streaming model which can be more efficient for inference/prediction mode [4]. In several applications, such as image classification, model representation in the training and inference modes are the same, while in others, such as sequence classification problems (for example, KWS), it can be different.

1.1. Model streaming example

Let's consider an example of convolutional NN (shown on Fig 1a) applied on KWS. A standard approach for model training is to use a non-streaming model representation, shown on Fig 1a. It receives the whole input sequence and then returns the classification result (on Fig 1 the whole input sequence has length 6 with single sample feature size 3).

In a KWS application we do not know when the keyword starts or ends, so we need to process every audio packet and

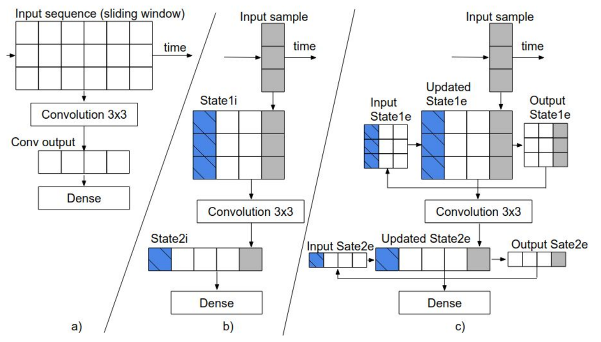

Figure 1: *Convolutional model: a) non-streaming b) streaming with internal state c) streaming with external state*

return the classification results in real time every 20ms (for example) - it is called streaming inference. It will not be efficient to use a non-streaming model representation (Fig 1a) in streaming inference mode because the same convolution will be recomputed on the input window multiple times. A standard approach to optimize latency in this case is to transform the non-streaming model (Fig 1 a) to a streaming one [4]. The resulting model will receive input samples with dimensions 3x1 incrementally process it (keeping the previously computed convolutions in a buffer to avoid unnecessary computations) and return the classification results in a streaming fashion, as shown on Fig 1b, c.

There are two options of implementing streaming inference: with internal state (Fig 1b) and with external state (Fig 1c). A model with internal state receives a new input sample with dimensions 3x1 (grey box 3x1 on Fig 1b), appends it to the ring buffer State1i, and at the same time removes the oldest sample (marked by blue with dashed lines on State1i) from State1i. This way, State1i always has a shape of 3x3. In the next step State1i is used by convolution 3x3 (on Fig 1b). Then conv output (1x1 grey box) is concatenated with buffer State2i. At the same time we remove the oldest sample (marked by blue with dashed lines on State2i) from State2i. In the end, State2i is fed into a dense layer. As a result, the convolution will be computed on the new input data only and all previous computations will be buffered, as shown on Fig 1 b. In this case states are internal variables of the model which have to be updated with every prediction. The same approach is shown on Fig 1c, with one difference: states State1e and State2e are inputs and outputs of the model. In this case model does not keep any internal states and developers will have to feed them into neural network as additional inputs and then in addition to classification results receive the updated states and feed them back on the next prediction cycle.

In the above example, model representation in training and inference models can be different, so developers have to reimplement the NN model in streaming mode (for model latency optimization). In this work we would like to automate it, so that

[1]Preprint. Submitted to INTERSPEECH.

developers could write a NN model once, train it and then automatically convert it to streaming mode. For this purpose we designed a Keras streaming wrapper layer, described in section 2.

1.2. About speech frontend and modeling pipeline

KWS NN models use a speech feature extractor based on MFCC [6]. Our implementation of speech feature extractor supports both FFT and DFT. If DFT is selected, then it will increase model size, because of DFT weights. We implemented all the components of the speech feature extractor in Keras layers, so that they will be part of the Keras NN model and will be automatically converted to TFLite [7]. Since the speech feature extractor is a part of the model it will be easier to deploy it on a mobile device. The overall modeling pipeline (with given data) will have several steps:

- Design a model using Keras layers and Stream wrapper.
- Train the model.
- Automatically convert the trained non-streaming Keras model to Keras streaming one.
- Convert streaming Keras model to a TFLite module [7]. Quantize it if needed to optimize model size and performance.
- Benchmark TFLite module: measure accuracy and latency on CPU of mobile phone Pixel4 [8].

The main contributions of this paper are:

- We implemented several popular KWS models in Keras and designed a streaming Keras layer wrapper for automatic model conversion to streaming inference with internal/external states.
- We improved the classification error of the state of the art KWS model by 10% on datasets V2 [9].
- We trained KWS NN models on Google speech commands dataset [10] and compared their accuracy with streaming and non-streaming latency on a Pixel 4 phone.
- The code, pretrained models and experimental results are open-sourced and available at [11].

2. Model streaming

We designed a streaming Keras layer wrapper which allows automatic conversion of Keras models to streaming inference (with internal or external states, Fig 1b, c). With this approach developer does not need to manually rewrite the model for streaming mode. This allows us to reduce the time to model deployment. Below is an example of a functional Keras model:

```
output = tf.keras.layers.Conv2D(...)(input)
output = tf.keras.layers.Flatten(...)(output)
output = tf.keras.layers.Dense(...)(output)
```

By wrapping layers (which have to be streamed and require a buffer as shown in Fig 1b, c) with the Stream wrapper we get the model (in this example Dense layer does not keep any states in time, so it is streamable by default):

```
output = Stream(cell=tf.keras.layers.Conv2D(...))(input)
output = Stream(cell=tf.keras.layers.Flatten(...))(output)
output = tf.keras.layers.Dense(...)(output)
```

Now we can train this model (with no impact on training time) and convert it to streaming inference mode automatically. Stream wrapper creates states and manages in inference mode. In addition, the Stream wrapper needs to know the effective time filter size which is different for different layers, that is why inside of the Stream wrapper we have different logic for extracting the effective time filter size for a particular layer.

We designed the Stream wrapper with several requirements. First, it shouldn't impact the original model training or default non-streaming inference and will be used only for streaming inference. In training mode it uses the cell as it is, so the training time will be the same as without the Stream wrapper. Next, we would like to support cells with internal and external states as it is shown in Fig 1b (with internal state) and Fig 1c (with external state). So that we can use different inference engines and compare them with each other.

RNN layers also require states during streaming inference, so we build streaming-aware RNN layers with streaming function. It behaves as standard RNN during training, but after model conversion to streaming inference it will only call the RNN cell (with internal or external state defined by the user).

Automatic conversion to streaming inference mode has several steps: 1) set input layer feature size equal one frame; 2) traverse Keras NN representation and insert ring buffer for layers which have to be streamed or call streaming function in streaming-aware layers such as RNN. This approach is not specific to KWS models and can be used in other applications.

Current version of Stream wrapper does not support striding nor pooling more than 1 in the time dimension, but it can be implemented in the future.

3. Model architectures

The standard end to end model architecture applied for KWS [12, 14] consists of a speech feature extractor(optional) and a neural network based classifier. An input audio signal of length 1sec is framed into overlapping frames of length 40ms with an overlap of 20ms as in [5]. Each frame is fed into a speech feature extractor which is based on MFCC [6]. The extracted features are classified by a neural network which produces the probabilities of the output classes. We use cross entropy loss function with Adam optimizer for model training. In this section we overview the neural network architectures evaluated in this work, We implemented popular models from [5], [27], [28], in our library for benchmarking streaming and non streaming inference on mobile phone.

3.1. Deep Neural Network (DNN)

The DNN model applies fully-connected layers with rectified linear unit (ReLU) activation function on every input speech feature. Then the outputs are stacked over 49 frames and processed by a pooling layer followed by another sequence of fully-connected layers with ReLU. We observed that this architecture with a pooling layer gives higher accuracy than the standard DNN model published in [5], as shown in Table 1. We use a similar number of model parameters with DNN model in [5].

3.2. Convolutional Neural Network (CNN)

CNN [15] is a popular model which is used in many applications including KWS [14,15,22]. As in [5] it is composed of sequence of 2D colvolutions with ReLU non linearities followed by fully connected layers in the end. Below we benchmarked several variations of CNN: one with striding equal 2 (CNN+strd on Table 1 and Table 2) and another with no striding (CNN on Table 2, it can be automatically converted to streaming mode).

3.3. Recurrent Neural Networks: LSTM, GRU

RNNs [16] are successfully applied in KWS problems [17, 18]. It receives speech features as a sequence and processes it sequentially with updating the internal states of the model. This approach is well suited for streaming inference mode. We explore two versions of RNN: LSTM [19] (in Table 1 called LSTM) and a GRU-based model [20] and compare them with baseline [5] on Table 1.

3.4. Convolutional Recurrent Neural Network (CRNN)

CRNN [17] combines properties of both CNN, which captures short term dependencies, and RNN, which uses longer range context. It applies a set of 2D convolutions on speech features followed by a GRU layer with fully connected layers. We compare CRNN model in our library with baseline [5] on Table 1.

3.5. Depthwise Separable Convolutional Neural Network (DSCNN)

DSCNN [21] models are well applied on KWS [5]. This model processes speech features by using a sequence of 2D convolutional and 2D depthwise layers followed by batch normalization with average pooling (as in [5]) and finished by applying fully connected layers. Below we benchmarked several variations of DSCNN: one with striding equal 2 (it is similar to DSCNN in [5] and shown as DSCNN+strd on Table 1 and Table 2) and another with no striding (DSCNN on Table 2, it can be automatically converted to streaming mode).

3.6. Multihead attention RNN (MHAtt-RNN)

The development of the Attention mechanism [24,25] improved the accuracy on multiple tasks including KWS [27]. In [27] the authors build a model Att-RNN which takes a mel-scale spectrogram and convolves it with a set of 2D convolutions. Then two bidirectional LSTM [16] layers are used to capture two-way long term dependencies in the audio data. The feature in the center of the bidirectional LSTM's output sequence is projected using a dense layer and is used as a query vector for the attention mechanism. Finally, the weighted (by attention score) average of the bidirectional LSTM output is processed by a set of fully connected layers for classification [27]. We extended this approach with multi-head attention (4 heads) and replaced LSTM with GRU (and call this version MHAtt-RNN). It allows us to reduce classification error by 10% in comparison to the state of the art (shown in Table 1). Both Att-RNN and MHAtt-RNN models are using bidirectional RNN, so they cannot be converted to streaming mode and have to receive the whole sequence before producing classification results.

3.7. Singular value decomposition filter (SVDF)

We implemented a simplified version of [28], so that it does not require aligned annotation of audio data for training. This model is composed of several SVDF and bottleneck layers with one softmax layer in the end. The SVDF block is a sequence of one dimensional convolution and one dimensional depthwise convolution layers [28].

3.8. Temporal Convolution ResNet (TC-ResNet)

The TC-ResNet model [22] is composed of sequence of residual blocks which use one dimensional convolution. In this paper we use neural network topology called TC-ResNet14 [22]. To improve accuracy of this model we increase number of parameters from 305K to 365K and use SpecAugment [26]. Baseline TC-ResNet accuracy with its improvement on data sets V1 and V2 are shown on Table 1.

Table 1: *Baseline models accuracy on data V1* and V2* with paper references and our models accuracy on data V1 and V2*

Models	V1*[%]	V1[%]	V2*[%]	V2[%]
DNN	86.7 [5]	91.2		90.6
CNN+strd	92.7 [5]	95.4		95.6
SVDF		96.3		96.9
DSCNN+strd	95.4 [5]	97.0		97.1
GRU	94.7 [5]	96.6		97.2
LSTM	94.8 [5]	96.9		97.5
CRNN	95.0 [5]	97.0		97.5
Att-RNN	95.6 [27]		96.9 [27]	
TC-ResNet	96.6 [22]	97.1		97.4
Embed+head			97.7 [13]	
MHAtt-RNN		97.2		98.0

4. Experimental results

4.1. Datasets and accuracy metrics

On Table 1 we compare the accuracy of published models with our implementation on a Google dataset V1 [29] and V2 [9].

We use the standard data set up from TensorFlow speech commands example code, proposed at [10]. The NN is trained on twelve labels: ten words "yes", "no", "up", "down", "left", "right", "on", "off", "stop", and "go" with additional two words: "silence" and "unknown". The "unknown" category contains remaining 20 keywords from the dataset. As in [5, 10] we use an algorithm from [30] for splitting the data into training, validation and testing set with ratio 80:10:10. The length of the one training speech sample is 1 sec, and the sampling rate is 16kHz.

After applying the standard data set up (described above) on data sets V1 [29] we have 22246, 3093, 3081 samples for training validation and testing respectively. With data set V2 [9] we have 36923, 4445, 4890 samples for training validation and testing respectively.

The training data is augmented with:

- time shift in range -100ms...100ms (as in [30], [5]);
- signal resampling with resampling factor in range 0.85...1.15;
- background noise (as in [30], [5]);
- frequency/time masking, based on SpecAugment [26] (except time warping).

For side-by-side comparison purposes we use classification accuracy metric as in [5]. It is calculated by running the model on the testing data, and comparing the classification result against the expected label.

4.2. Comparison with baseline

We implemented popular KWS approaches DNN [5], CNN [5], LSTM [5], GRU [5], CRNN [5], DSCNN [5], TC-ResNet [22], described above, using our library for benchmarking streaming and non streaming models. We improved their accuracy on datasets V1 and V2, as shown on Table 1, by applying SpecAugment [26](except time warping) with hyper-parameters optimization of both neural net and speech feature extractor parameters. After model is trained on datasets V1/V2 we convert it to TFLite format (to be able to run it on a mobile phone), then run inference with TFLite and report its accuracy on Table 1 (columns V1, V2). The baseline accuracy with reference to paper is shown on Table 1 (columns V1*, V2*).

One of the best KWS models is Embed+head [13]. It achieves the state of the art accuracy by using additional data sets from YouTube to train embedding layer. We introduced MHAtt-RNN model. It reduces classification error on datasets V2 by 10% in comparison to Embed+head [13], as shown on Table 1. The cost of this improvement is MHAtt-RNN model has two times more parameters than Embed+head [13]. Another recently published promising approach is Matchbox [23], but authors use different training testing data set up, so we could not compare it side by side.

In addition we implemented and benchmarked SVDF model [28] on both data sets V1 and V2 shown at Table 1 and demonstrated that it has good properties for streaming at Table 2.

4.3. Streaming and non-streaming latency with accuracy

In the real production environment we do not know neither the beginning nor ending of the speech command produced by the user. Also we need to provide real time responses for a good user experience. As a result, the speech command detector is running in streaming mode by classifying every 20 milliseconds (for example) of the input audio stream.

We trained all the models on datasets V2 and converted DNN, CNN no stride, CRNN, DSCNN no stride, SVDF to a streaming inference mode and benchmarked them on datasets V2. The models DSCNN with stride, CNN with stride and MHAtt-RNN are not streamable with our library. To emulate the streaming environment during accuracy evaluation we did not reset the RNN model states between testing sequences. We observed up to 2x accuracy reduction on such models (also baseline RNN models in Table 1 have the same issue). We addressed it by re-training RNN models GRU, CRNN) with stateful argument=True [31]. The last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch (the model will learn how to process cell states on its own). To distinguish such RNN models on Table 2 we append (S). We observed accuracy reduction of statefully trained models GRU (S) CRNN (S), shown on Table 2 in comparison to their non statefully trained versions GRU, CRNN shown Table 1 (column V2).

The latency and accuracy of non-streaming and streaming models with the number of parameters are presented in Table 2. We use a Pixel 4 mobile phone [8] and TFlite benchmarking tools [32] to measure the models latency. Non-streaming latency is the processing time of the whole 1 sec speech sequence by the non-streaming model representation. Streaming latency is the processing time of one audio frame by the streaming model (it receives 20ms of audio and returns classification result). Processing time includes both feature extraction and neural network classification (end to end).

In Table 2 we observe that the most effective and accurate streaming models are SVDF, CRNN and GRU. Layers with striding/pooling are not streamable in our library now, but it can be implemented in the future. With support of striding/pooling in streaming mode, models such as TC-ResNet, Embed+head and Matchbox can be more preferable. The most accurate non-streaming model is MHAtt-RNN. It is based on bidirectional LSTM, so non streamable by default.

Table 2 shows that the average latency ratio of non-streaming to streaming convolutional models (CNN, SVDF, DSCNN) is around 10x. The same ratio for RNN models (GRU(S), CRNN(S)) is around 20x. We explain this difference by the fact that non-streaming RNN models still have to be executed sequentially frame by frame over all frames belonging to 1 second of input audio, whereas non-streaming convolutions can be computed over the whole sequence in one batch. In an ideal case this ratio has to be around 50x (there are 50 frames in one second). As a result, there are opportunities for latency speed-up of streaming models: reduce memory allocations in the ring buffers and enable support of internal state in the inference engine. At the same time the latency of non-streaming models also can be optimized further. Detailed profiling of non-streaming models showed that the speech feature extraction latency is around 3.7ms. It can be reduced by using FFT and model quantization (after enabling both we observe almost 2x latency reduction). As expected non-streaming models with striding are several times faster than the same model with no striding: CNN and DSCNN with striding are two times faster than the same models without striding.

Table 2: *Accuracy on data V2 with latency and model size*

Models	**accuracy, [%]**	**non stream latency, [ms]**	**stream latency, [ms]**	**model size, [K]**
DNN	90.6	4	0.6	447
CNN+strd	95.6	6	N/I	529
CNN	96.0	15	1.5	606
GRU (S)	96.3	11	0.5	593
CRNN (S)	96.5	9	0.5	467
SVDF	96.9	5	0.6	354
DSCNN	96.9	19	1.6	490
DSCNN+strd	97.0	9	N/I	485
TC-ResNet	97.4	5	N/I	365
MHAtt-RNN	98.0	10	N/A	743

5. Conclusion

We built a library allowing end-to-end model conversion to streaming inference on mobile phones using Keras and TFLite. The converted model encapsulates speech feature extraction. It simplifies model deployment on mobile devices. We reduced classification error by 10% relative, in comparison to the state of the art models on datasets V2. It was achieved by extending the Att-RNN [27] model with multi-head attention and applying SpecAugment [26]. For benchmarking purpose we implemented several popular models using our streaming library and measured streaming and non streaming latency on a Pixel 4 mobile phone and demonstrated different tradeoffs between accuracy and latency. All code with experimentation results are open-sourced and available at [11].

6. Acknowledgements

The authors would like to thank Pete Warden, Karolis Misiunas, Yukun Zhu, Ben Vanik, Robert Suderman, Rohit Prabhavalkar, Kevin Kilgour and BDI team for valuable discussions and suggestions.

7. References

[1] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev, "Compressing deep convolutional networks using vector quantization," *CoRR*, vol. abs/1412.6115, 2014. [Online]. Available: http://arxiv.org/abs/1412.6115

[2] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 06 2018, pp. 2704–2713.

[3] Y. He, T. Sainath et al, "Streaming End-to-end Speech Recognition For Mobile Devices," *CoRR*, vol. abs/1811.06621, 2018. [Online]. Available: https://arxiv.org/abs/1811.06621

[4] A. Coucke, M. Chlieh, T. Gisselbrecht, D. Leroy, M. Poumeyrol, T. Lavril, "Efficient keyword spotting using dilated convolutions and gating," *ICASSP 2019*, [Online]. Available: https://arxiv.org/abs/1811.07684

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

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

[7] [Online]. Available: https://www.tensorflow.org/lite

[8] [Online]. Available: https://store.google.com/product/pixel_4

[9] "Speech commands dataset v2." [Online]. Available: https://storage.googleapis.com/download.tensorflow.org/data/speech_commands_v0.02.tar.gz

[10] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *CoRR*, vol. abs/1804.03209, 2018. [Online]. Available: http://arxiv.org/abs/1804.03209

[11] [Online]. Available: https://github.com/google-research/google-research/tree/master/kws_streaming

[12] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014, Florence, Italy, May 4-9, 2014*. IEEE, 2014, pp. 4087–4091. [Online]. Available: https://doi.org/10.1109/ICASSP.2014.6854370

[13] J. Lin, K. Kilgour, D. Roblek, and M. Sharifi, "Training Keyword Spotters with Limited and Synthesized Speech Data," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain* . IEEE, 2020, pp. 7474–7478. [Online]. Available: https://doi.org/10.1109/ICASSP40776.2020.9053193

[14] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*. ISCA, 2015, pp. 1478–1482. [Online]. Available: http://www.isca-speech.org/archive/interspeech_2015/i15_1478.html

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[17] S. Ö. 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," in *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*, [Online]. Available: https://arxiv.org/abs/1703.05390

[18] 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 2016, San Diego, CA, USA, December 13-16, 2016*. IEEE, 2016, pp. 474–480. [Online]. Available: https://doi.org/10.1109/SLT.2016.7846306

[19] F. A. Gers and J. Schmidhuber, "LSTM recurrent networks learn simple context-free and context-sensitive languages," *IEEE Trans. Neural Networks*, vol. 12, no. 6, pp. 1333–1340, 2001. [Online]. Available: https://doi.org/10.1109/72.963769

[20] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, [Online]. Available: https://arxiv.org/abs/1406.1078

[21] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[22] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, S. Ha, "Temporal Convolution for Real-time Keyword Spotting on Mobile Devices," *CoRR*, vol. abs/1704.04861, 2019. [Online]. Available: http://arxiv.org/abs/1904.03814

[23] S. Majumdar, B. Ginsburg, "MatchboxNet: 1D Time-Channel Separable Convolutional Neural Network Architecture for Speech Commands Recognition," *CoRR*, vol. arXiv:2004.08531v2, 2020. [Online]. Available: https://arxiv.org/abs/2004.08531v2

[24] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. [Online]. Available: http://arxiv.org/abs/1409.0473

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need

[26] D. Park, W. Chan, Y. Zhang, C. Chiu, B. Zoph, E. Cubuk and Q. Le, "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition," in *Interspeech 2019*, [Online]. Available: https://arxiv.org/pdf/1904.08779

[27] D. C. de Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, "A neural attention model for speech command recognition," *CoRR*, vol. abs/1808.08929, 2018. [Online]. Available: http://arxiv.org/abs/1808.08929

[28] R. Alvarez and H. Park, "End-to-end streaming keyword spotting," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019*. IEEE, 2019, pp. 6336–6340. [Online]. Available: https://doi.org/10.1109/ICASSP.2019.8683557

[29] "Speech commands dataset v1." [Online]. Available: http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz

[30] [Online]. Available: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/speech_commands/input_data.py#L61

[31] [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

[32] [Online]. Available: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/tools/benchmark