Open-Vocabulary Keyword Spotting With Audio And Text Embeddings

Niccolò Sacchi[1], *Alexandre Nanchen*[2], *Martin Jaggi*[1], *Milos Cernak*[3]

[1]École Polytechnique Fédérale de Lausanne [2]Idiap Research Institute [3]Logitech Europe S.A.

niccolo.sacchi@epfl.ch, alexandre.nanchen@idiap.ch, martin.jaggi@epfl.ch, milos.cernak@ieee.org

Abstract

Keyword Spotting (KWS) systems allow detecting a set of spoken (pre-defined) keywords. Open-vocabulary KWS systems search for the keywords in the set of word hypotheses generated by an automatic speech recognition (ASR) system which is computationally expensive and, therefore, often implemented as a cloud-based service. Besides, KWS systems could use also word classification algorithms that do not allow easily changing the set of words to be recognized, as the classes have to be defined a priori, even before training the system. In this paper, we propose the implementation of an open-vocabulary ASR-free KWS system based on speech and text encoders that allow matching the computed embeddings in order to spot whether a keyword has been uttered. This approach would allow choosing the set of keywords a posteriori while requiring low computational power. The experiments, performed on two different datasets, show that our method is competitive with other state of the art KWS systems while allowing for a flexibility of configuration and being computationally efficient.

Index Terms: Speech recognition, keyword spotting, open-vocabulary, ASR-free, audio&text embeddings.

1. Introduction

Keyword Spotting (KWS) systems are in high demand nowadays as they enable a simple voice user interface to consumer electronics devices. There are two kinds of KWS systems: (i) *closed-vocabulary*, e.g. [1, 2], characterized by a small footprint model and trained on repetitions of the desired keywords such as in the popular Google command and control dataset [3], and (ii) *open-vocabulary*, e.g. [4, 5] that parse word or phoneme lattices to spot the keywords and usually require an Automatic Speech Recognition (ASR) system, and, thus, have a bigger memory and computation footprint. A configuration of the open-vocabulary KWS system is usually text-based; when audio input is required for configuration, keyword spotting then becomes a spoken query system. With the advent of end-to-end modelling the distinction between ASR and ASR-free KWS approaches is diminished, as there are also ASR-free approaches that still require cloud computing [6].

Our aim is to explore KWS architectures with the following features: (i) *versatile* (open vocabulary), i.e. the set of keywords is relatively small but not fixed, words may be added to and removed from it without retraining any model, (ii) *user-friendly*, i.e. new keywords could be added both in the form of text, e.g. by typing the keyword, and in the form of audio, e.g. by recording the keyword a few times, (iii) *robust* to the detection of words that have not been used during training, (iv) *computationally-efficient*, i.e. the architecture is thought to run on small devices, e.g. speakers, headphones, cameras.

We have hypothesized that these goals can be achieved with a system based on audio and text encoders which extract similar *embeddings* for the same word (Figure 1). In this way, whether a *keyword* is provided in the form of audio or text we would be able to compute its embedding and compare it with the embedding obtained from the input audio. A distance metric can be used to detect whether the two embeddings are close enough, i.e. whether they correspond to the same word. Therefore, just by computing the embeddings representing the keywords we can start detecting them from the input audio. In addition, there is no limitation on which keyword should be used, i.e. this approach allows having a completely *flexible* set of keywords to be detected which can be chosen by the user by both typing and/or recording them.

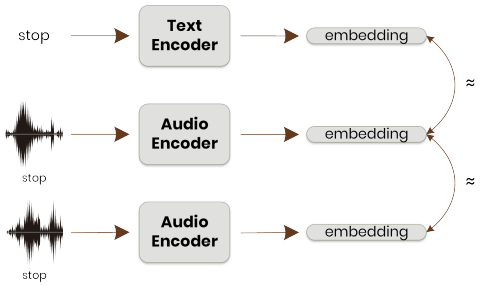

Figure 1: *Audio and text encoders extract a similar representation for inputs representing the same word.*

Moreover, since the purpose of such encoders is in extracting meaningful information from the input that might be noisy, they are not only restricted to solve a classification task (what is the label of this sample?) but they could also be employed in verification (does the two samples have the same label?), clustering (find common samples among all samples) and information retrieval (fetch the K closest samples).

2. Triplet-loss Encoders

We devised two encoders. Firstly, we focused on an audio encoder (section 2.2), i.e. an encoder that maps the input audio in an embedding space where clusters of word embeddings are formed. Then, we devised an audio-text encoder (section 2.3) that allows also obtaining embeddings of keywords for which no audio samples are available but only their textual representation. Next section explains the training of the encoders.

2.1. Training With Triplets

To train an encoder that extracts a similar representation for audio samples representing the same word, we have decided to opt for the triplet loss [7]. The goal of this loss is to train an encoder that extract embeddings that allow easily discriminating the classes in the embedding space. This goal is achieved by selecting two samples with the same label, respectively called

anchor and *positive*, and a sample with a different label, called *negative*. Then, all the three samples are encoded with the same encoder and the triplet loss is applied to the three embeddings as follows:

$$L(a, p, n) = |d(a, p) - d(a, n) + margin|_+$$

where $|\cdot|_+$ is the hinge loss, *a, p, n* are respectively the anchor, positive and negative embeddings, *margin* specifies a threshold (if $d(a, n) - d(a, p) > margin$ then the loss is 0) and d must be a (sub-)differentiable distance metric used to compute the distance between two embeddings. In particular, after performing different tests, we have decided to use squared euclidean distance and a $margin$ of 1. Applying this loss on a batch of triplets simply results in averaging over them. However, training with this loss is known to be challenging and, in particular, selecting wisely the triplets is crucial to both boost the training of the encoder.

Motivated by [8], we have used the batch-hard technique which consists in the selection of worst triplets over the batch, thus leading to a greater loss and update the model parameters and, also, a more stable convergence. The core idea is to first form batches by randomly sampling P classes, e.g. P words, and K samples of each class, e.g. K audio samples for each a word, thus resulting in a batch of $P \times K$ samples. Each sample is encoded, i.e. a batch of $P \times K$ embeddings is computed. Then, each embedding is treated as the *anchor* sample exactly once and its *hardest* positive embedding, i.e. the farthest embedding with the same label, and its *hardest* negative embedding, i.e. the closest embedding with a different label are selected from the embeddings in the batch so to form the triplets. P and K have to be carefully chosen depending on the amount of noise in the dataset. In particular, with noisy datasets, it is suggested to use small values of P and K as the *batch-hard* technique is greatly sensitive to the presence of outliers and mislabeled samples. In our experiments we have found that with $P = 5$ and $K = 3$ the train loss was nicely getting lower and the encoder was learning to extract good embeddings.

2.2. Audio Encoder

We have trained an *audio2word-vector (a2wv)* encoder, to encode audio samples, each representing a single keyword. Such encoder is implemented as a GRU network with two unidirectional stacked layers, a hidden state (equal to the size of the embeddings) of 64 floating point numbers and a dropout of 0.3 between the two stacked layers. Figure 2 depicts the trained encoder.

The bottom plot shows one cluster (color) per word where each dot is an embedding obtained by encoding an audio sample of someone speaking the corresponding word. Moreover, *unseen* words are words that have not been used during training.

2.3. Audio-Text Encoder

Purpose of this encoder is to enable possibility to compute, during the configuration phase of the KWS system, reference keyword embeddings directly from their textual representation. This solution requires both to train a phone encoder and also a dictionary to map words to their sequence of phonemes, e.g. $hello \mapsto HH, AH, L, OW$. We now explain how phone embeddings are extracted from *audio* and from *text* and, then, how they are merged into the final word embedding.

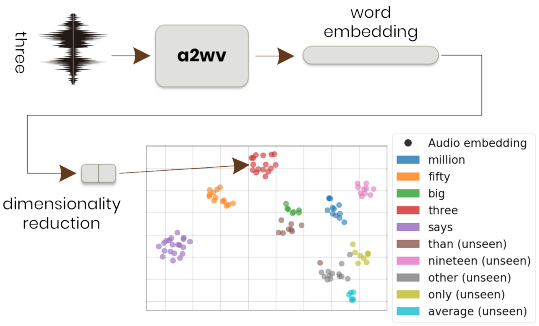

Figure 2: *Embeddings obtained by a2wv encoding, each representing one word. For purpose of visualization the dimensionality of the embeddings has been reduced with t-SNE [9] to* 2 *dimensions.*

2.3.1. Phone embeddings from audio

To obtain phone embeddings from audio we have implemented an audio encoder similarly as the one explained in section 2.2. However, there are two differences: (1) at the input, during training, we feed phones (with boundaries) instead of words and (2) since, clearly, no phone boundaries are provided during testing, this encoder continuously outputs phone embeddings while processing the audio. In particular, it is trained such that each output embedding actually represents the last pronounced phone, i.e. we expect the embedding of a phone to be computed and provided at the output as soon as that phone is processed. We call this model *audio2phone-vectors* (*a2pvs*) as it outputs many embeddings from the input audio.

Moreover, since a phone often occupies more than one input frame, *a2pvs* is designed so that it halves the dimension of the input, i.e. it outputs one embedding every two processed frames, therefore mitigating the problem of having more phone embeddings at output than spoken phone in the input audio. As we will explain in the next sections, this also helps having a more similar amount of phone embeddings when extracting them from text or from audio, thus helping in obtaining similar word embeddings. Figure 3 depicts how *a2pvs* is used to extract phoneme embeddings. Clearly, encoding phones is harder than encoding words, indeed here we can notice that the clusters are less discriminable than the clusters we obtained when encoding words.

2.3.2. Phone embeddings from text

To obtain phone embeddings from the text we have used the previously trained *a2pvs*. In particular, since there is a limited number of phones, we can use the phone encoder and a dataset of spoken phones to build a 1-to-1 mapping table, called *phone2phone-vector* (*p2pv*), that maps each phone to its average embedding, which is estimated by averaging over the embeddings obtained by encoding a few audio samples representing the phone. These average phone embeddings can be interpreted as the representations of the phones since they should be an estimate of the center of the corresponding phone clusters. While with *a2pvs* we can encode audio to obtain phone embeddings, for text we first use a dictionary to get the list of phones corresponding to a word and, then, *p2pv* to map each phone to its estimated average embedding. Note that, with this approach, we expect to obtain similar phone embeddings from the text and from audio representing the same phone.

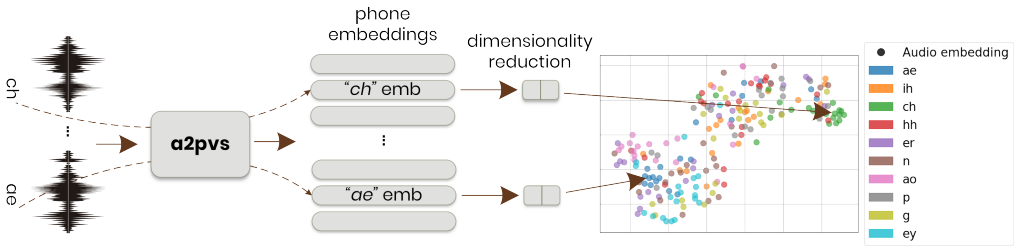

Figure 3: *a2pvs encodes the input audio sample to a sequence of embeddings so that each embedding represents the last pronounced phone. The plot on the right displays embeddings (dots) that have been obtained after a phone has been pronounced.*

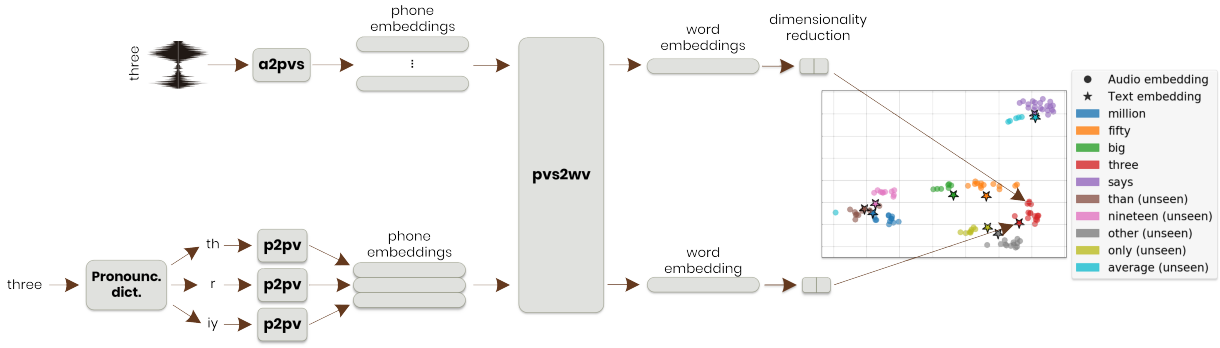

Figure 4: *Audio (dots) and text (stars) embeddings obtained with the audio-text encoder.*

2.3.3. Encoding phone embeddings

Finally we train another encoder, called *phone-vectors2word-vector* (*pvs2wv*), to merge all the phone embeddings, whether obtained from audio or text, into the final word embedding. *pvs2wv* is implemented as a GRU network to manage a variable number of phone embeddings in input. Figure 4 depicts the whole audio-text encoder and, in particular, how it encodes audio (top) and text (bottom). To encode audio, first, *a2pvs* encodes the audio sample to a sequence of phone embeddings, then, *pvs2wv* encodes the latter to the word embedding. To encode text, embeddings are obtained by means of a dictionary and of *p2pv* (c.f. section 2.3.2), then, the network *pvs2wv* is used to encode the embeddings to the text embedding of the word. On the plot on the right, each dot represents an embedding obtained from audio while each star represents an embedding obtained by encoding the textual representation of the corresponding word.

3. Keyword Spotting System

The KWS system we propose is made of two architectures: (i) an *encoder*, which is trained to compute similar embeddings from audio recordings and text representing the same word, (ii) a *classifier*, which implements a metric to compare two embeddings and assess if they represent the same word. After encoders' training, running such a KWS system consists of the two following phases (Figure 5):

1. *Configuration phase*. The user will provide either some recordings or the textual representation for each keyword to be detected. The encoder is then used to compute (at least) one reference embedding per keyword which are stored and used during the *detection phase*.
2. *Detection phase*. The classifier uses the implemented metric to compare each computed embedding with all the reference keyword embeddings and, consequently, maps each recording to the corresponding keyword, if any.

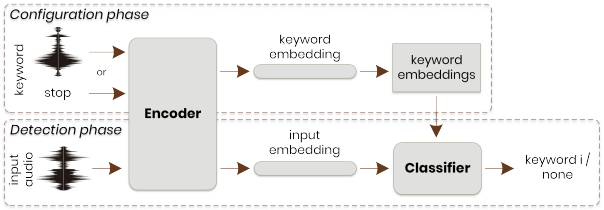

Figure 5: *The two phases of the KWS system: (1) configuration phase, the keywords to be detected are provided either by text or audio recording, (2) detection phase, the classifier maps each computed input embedding to a prediction.*

The approach we propose allows tackling also the cases in which the user pronounces or types in a word that has not been used during training, i.e. *unseen* words. In particular, the KWS system can be configured to detect such words too.

3.1. Classifier

Independently of how the system is configured, the prediction is obtained by selecting a keyword whose embedding is the closest to the input embedding and by using a *threshold*. The latter defines the minimum allowed distance to the closest keyword embedding. Precisely, the classification is defined as follows:

$$c(x) = \begin{cases} argmin_i\ d(e_i, f(x)) & if\ \ \exists j\ \ s.t.\ \ d(e_j, f(x)) < t \\ -1 & otherwise \end{cases}$$

where c represents the classification function, x input audio to be classified, d is the distance metric, f is the encoder, e_i the

embedding of the *i-th* keyword, t the threshold and -1 indicates that no keyword embedding is close enough, i.e. no keyword has been detected.

4. Experimental Settings and Evaluation

We first evaluate the proposed systems, configuring them by text and/or audio, on the evaluation `test_eval93` subset of Wall Street Journal (WSJ) [10] and on keyword sets containing only one keyword so to provide a proof of concept. Then, we evaluate and compare our solutions with a baseline and on a bigger keyword set. The baseline for keyword spotting is an ASR system based on Kaldi framework [11] with a lattice-free Maximum Mutual Information [12] model and a keyword grammar with a phone-loop garbage model.

Both our systems and the baseline are trained on the `train_si284` subset of WSJ data, preprocessed with 40 dimensional high resolution mel frequency cepstral coefficients. Both systems use the same input features and data splits. We used the `test_eval92` as a validation subset for our triplet loss encoders. In the following evaluation we will refer to KWS systems based on audio encoder and audio-text encoder as respectively KWS-AE and KWS-ATE.

4.1. Evaluation on WSJ

We have built two different sets of words, each containing 101 words, which have to be detected in the input audio. One set contains words that were used during the training of the encoders, i.e. *seen*, while the other contains words that were removed from the train set, i.e. *unseen*. The purpose is to assess how robust is our system to the detection of new words, even when spoken by speakers that were not in the train set. Then, we compute a DET curve for each each words in the sets and average them to obtain one DET curve for the *seen* keywords and one for the *unseen* ones. Figure 6 shows the DET curves (in a log-log plot for easier comparison) that have been obtained with KWS systems that use the *audio* and *audio-text* encoders. The latter has been configured both by text or by audio to evaluate respectively the scenario in which a user provides the word in the text format or records himself pronouncing it.

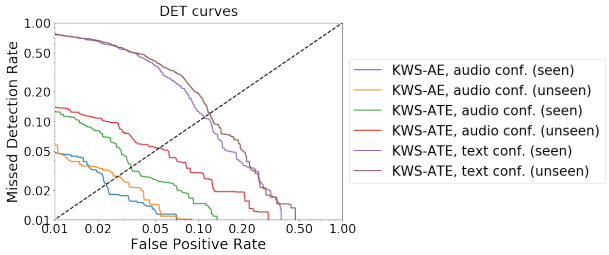

Figure 6: *Average seen and unseen DET curves obtained a KWS system employing either the audio or the audio-text encoder configured by text or audio.*

4.2. Evaluation on Logitech Command Dataset

Then, we evaluated the KWS system using a (proprietary) Logitech voice control dataset. We used a subset of 12 US English native speakers and 14 keywords (commands). We followed the evaluation protocol as described in [3] with 14 command labels, one additional special label for "Unknown Word", and another for "Silence" (no speech detected). The "Silence" category has one-second clips extracted randomly from the background noise audio files. Configuration of the KWS system has been done by three audio samples per keyword to compute an average reference embedding. These samples were removed from the evaluation set. The test is then done by providing examples for each of the 14 categories. The "Unknown Word" category contains commands randomly sampled from the Logitech dataset and that are not part of the keywords.

4.2.1. Results

Figure 7 shows the computed DET curves. We observe that our model *a2wv* is comparable with the Kaldi one. However, we point out that *a2wv* is also greatly simpler and lighter which would make it more feasible to be deployed on a small device. Moreover, differently than the Kaldi model, our model has not been specifically trained on any of the commands in the dataset.

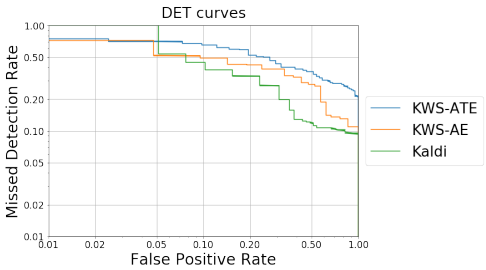

Figure 7: *DET curves computed with a2wv and Kaldi on the Logitech Dataset.*

5. Conclusions and Next Steps

We have proposed the architecture of an open-vocabulary KWS system based on joint encoding audio and text. Being this approach ASR-free, it is computationally more efficient than an ASR-based system, and no Language Model and no computationally expensive graph beam searches are required. In particular, the novelty of our work resides both in the idea of a KWS system configurable either by audio or text and also in the design of an *audio-text* encoder, which is capable of encoding *any* word, whose phone sequence is available, to the same embedding space in which also encoding of speech is mapped. This approach would allow obtaining a very light KWS system that the user/constructor can configure to detect any keyword of their choice by either typing in or pronouncing the commands they want to be detected.

Moreover, we trained a stream-oriented version of both the *audio* and the *audio-text* encoders. In particular, these architectures can be used to map the input audio stream to a stream of embeddings in such a way that each computed embedding represent the last *pronounced* word. However, a *classifier* to detect on-line in real-time keywords from this stream of embeddings has still to be designed.

Our future work is to explore all the potential of this approach. In particular, we propose: (1) design of a stream-oriented classifier that allows detecting keywords in real-time from a stream of word embeddings computed by the encoder, (2) training of the encoders on a more heterogeneous dataset so to improve the robustness to the environment and make the system more usable for real use-cases, (3) train bigger and more powerful networks by means of transfer learning which we observed greatly improving the training with triplet loss.

6. References

[1] 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*, 2015, pp. 1478–1482. [Online]. Available: http://www.isca-speech.org/archive/interspeech_2015/i15_1478.html

[2] 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

[3] 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

[4] I. Szöke, P. Schwarz, P. Matejka, L. Burget, M. Karafiát, M. Fapso, and J. Cernocký, "Comparison of keyword spotting approaches for informal continuous speech," in *INTERSPEECH 2005 - Eurospeech, 9th European Conference on Speech Communication and Technology, Lisbon, Portugal, September 4-8, 2005*, 2005, pp. 633–636. [Online]. Available: http://www.isca-speech.org/archive/interspeech_2005/i05_0633.html

[5] J. Trmal, M. Wiesner, V. Peddinti, X. Zhang, P. Ghahremani, Y. Wang, V. Manohar, H. Xu, D. Povey, and S. Khudanpur, "The kaldi openkws system: Improving low resource keyword search," in *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*, 2017, pp. 3597–3601. [Online]. Available: http://www.isca-speech.org/archive/Interspeech_2017/abstracts/0601.html

[6] J. Guo, K. Kumatani, M. Sun, M. Wu, A. Raju, N. Ström, and A. Mandal, "Time-delayed bottleneck highway networks using a dft feature for keyword spotting," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5489–5493.

[7] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *CoRR*, vol. abs/1503.03832, 2015. [Online]. Available: http://arxiv.org/abs/1503.03832

[8] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *CoRR*, vol. abs/1703.07737, 2017. [Online]. Available: http://arxiv.org/abs/1703.07737

[9] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: http://www.jmlr.org/papers/v9/vandermaaten08a.html

[10] D. B. Paul and J. M. Baker, "The design for the wall street journal-based csr corpus," in *Proceedings of the Workshop on Speech and Natural Language*, ser. HLT '91. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, pp. 357–362. [Online]. Available: https://doi.org/10.3115/1075527.1075614

[11] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The kaldi speech recognition toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011, iEEE Catalog No.: CFP11SRW-USB.

[12] D. Povey, V. Peddinti, D. Galvez, P. Ghahremani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur, "Purely sequence-trained neural networks for ASR based on lattice-free MMI," in *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, 2016, pp. 2751–2755. [Online]. Available: https://doi.org/10.21437/Interspeech.2016-595