00460211 - small footprint KWS in Hebrew

Raz Shemesh Tomer Ashkenazi

January 2025

1 Abstract

Keyword spotting (KWS) is a well-established problem, yet most novel models and datasets are predominantly designed for English. Developing a multilingual dataset for keywords presents significant challenges; hence, this project aims to adapt a model that is mostly trained on the Google Commands Dataset[1], to provide KWS in Hebrew. we were inspired by the offline-online work structure presented in [2], utilizing an offline-trained feature extractor and then deployed onto an embedded device to be utilized.

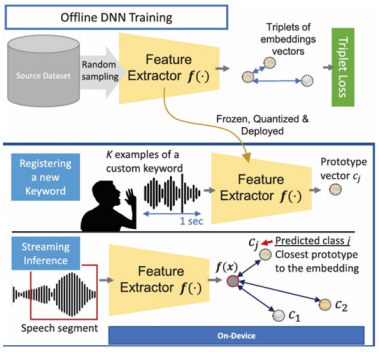

Figure 1: problem setup

Practilcly separating the process into two parts:
1. creating a robust generalized feature extractor.
2. customizing it on-device as a few-shot word classifier.

For the First part, we trained a DS-CNN(S) architecture on the Google command dataset with triplet loss (margin=1) using the same word with different utterances as a positive pair and a different word as a negative pair. To "tune" the model to Hebrew we've also used Weakly supervised auxiliary data from the ivrit.ai [3] dataset (inspired by 4]) using 1-sec snips of audio where speech is guaranteed, augmenting it slightly to create a positive pair and using different 1-sec snip as the negative pair; then using the combined regularized triplet loss. Weakly Supervised audio data is easier to generate and find in all languages, hence these kinds of adjustments could be implemented in every language. Regarding the second part, we've used data from ivrit.ai as the "unknown" label, creating unknown embeddings (we've used 50 utterances), those embeddings can be used on the device to customize the classifier, specifically given a sensitivity measure to the model (K in K-NN , minimal euclidien distance from a prototype to classify etc..) one could adjust said measure such that the classifier archives a specific FAR (false acceptance rate). Furthermore, we've also experimented with few-shot learning, that is using the few samples given not only to tune the classifier but also to adjust the Feature Extractor learnable parameters.

2 Method

2.1 Feature Extractor

The DS-CNN is a novel architecture in tiny-ML, we've used -"- overall XX params and YY Kb. The auxiliary data was augmented with random parameters for Pitch Volume and Pre-emphasis. we've trained the feature extractor for 20 epochs only on the Google commends dataset starting with a learning rate of 0.001 and decreasing to 0.0005 after 10 epochs, afterwards, we trained for 5 epochs with the auxiliary data. (with varying regulation parameter). the Triplet loss function allows as to separate the latent space as we intend, by maximizing the distance between negative pair and minimizing for positive pair. Given a word utternce W in the google dataset we denote W as the anchor , W^+ a positive pair i.e. the same word with a different utternece and W^- a negative pair i.e. a different word. in the auxliry data the positive pair is the anchor slightly augmented.

$$L_{Triplet} = max(d(W, W^+) - d(W, W^-) + margin, 0)$$

$$L_{total} = L_{Triplet\ \ GSC} + \lambda_{aux} L_{Triplet\ \ ivrit.ai}$$

In the Few-shot learning scenario, we've used a Prototypical Network [5] and implemented the pseudo-code provided in the article. in our case, V = Random Sample (choose 3 out of 4) Sk consists of one vector, and Qk contains the 4 that are left.

Algorithm 1 Training episode loss computation for prototypical networks. N is the number of examples in the training set, K is the number of classes in the training set, $N_C \leq K$ is the number of classes per episode, N_S is the number of support examples per class, N_Q is the number of query examples per class. RANDOMSAMPLE(S, N) denotes a set of N elements chosen uniformly at random from set S, without replacement.

Input: Training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where each $y_i \in \{1, \ldots, K\}$. $\mathcal{D}_k$ denotes the subset of $\mathcal{D}$ containing all elements $(\mathbf{x}_i, y_i)$ such that $y_i = k$.
Output: The loss J for a randomly generated training episode.

$V \leftarrow$ RANDOMSAMPLE$(\{1, \ldots, K\}, N_C)$ ▷ Select class indices for episode
for k in $\{1, \ldots, N_C\}$ **do**
  $S_k \leftarrow$ RANDOMSAMPLE$(\mathcal{D}_{V_k}, N_S)$ ▷ Select support examples
  $Q_k \leftarrow$ RANDOMSAMPLE$(\mathcal{D}_{V_k} \setminus S_k, N_Q)$ ▷ Select query examples
  $\mathbf{c}_k \leftarrow \frac{1}{N_C} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_{\boldsymbol{\phi}}(\mathbf{x}_i)$ ▷ Compute prototype from support examples
end for
$J \leftarrow 0$ ▷ Initialize loss
for k in $\{1, \ldots, N_C\}$ **do**
  for $(\mathbf{x}, y)$ in Q_k **do**
    $J \leftarrow J + \frac{1}{N_C N_Q} \left[d(f_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{c}_k)) + \log \sum_{k'} \exp(-d(f_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{c}_k)) \right]$ ▷ Update loss
  end for
end for

Figure 2: pseudo-code implemented

After 20 episodes are completed Wev'e used the 50 'unknown' samples buffer to calculate the current Feature extractor accuracy (using the classifier (2)) to be asserted to 'Reduce on plateau' scheduler. That was considered 1 Epoch , we've trained for 100 Epoches keeping the model who had the highest accuracy.

2.2 classifier

A few approaches were tested to generate the classifier, a zero-shot approach (1) - using the predictor suggested in [2], where the label vectors are the mean of embedding vectors of the few-shot recording available

$$y_{pred} = \begin{cases} label_{(argmin(d(label_i, y))} & if\ min_i(d(label_i, y)) \leq \gamma \\ 'unknowen' & else \end{cases}$$

Using γ = margin =1 from the triplet loss training.

The Second approach gets a list of reasonable gamma values, iterates over the,m and chooses the one that has the highest accuracy and below a dictated FAR (False Acceptance Rate); in a case where few gammas satisfies the condition we take the lowest one. if none of the gammas satisfy the condition we take gamma=1. we've used 5 percent as the FAR threshold, this customized classifier (2) then uses classifier (1) with "tuned" gamma.

2.3 Testing

We focused on a few specific aspects of the outcome classifier, Firstly the performance metrics, namely the accuracy i.e. the percentage of correctly labeled test utterances out of the total. and FAR (false-acceptance rate) i.e. the percentage of 'unknown' labeled as known.
Secondly, the variance of the received classifier; means the change in performance metrics when using different utterances to create the embedding vectors and different utterances as the unknown.
That means that in each testing iteration, we've sampled 5 utterances to create the embedding vector (or for the few-shot learning) and 50 'unknown' utterances for the customization. then we tested on different 'unknown' utterances and the 15 samples left in the labeled data.

3 Results

3.1 zero-shot classifier

although the highest accuracy belonged to the non-augmented model, the regulation parameter did seem to be correlated with the FAR and std of FAR decrease.

Lamda_aug	Avg Accuracy	Standard Deviation	Avg FAR	Standard Deviation
0	70.7%	4.2%	14.1%	4.4%
0.01	70%	3%	5.7%	2.7%
0.1	67.3%	4.1%	5.1%	3.1%
1	62.5%	2%	0.4%	0.8%

Figure 3: auxiliary regulation parameter vs performance metric

3.2 Zero shot optimize gamma classifier

Here it seems that using the auxiliary data only worsens the classifier

Lamda_aug	Avg Accuracy	Standard Deviation	Avg FAR	Standard Deviation
0	68.8%	4%	5%	4%
0.01	67.5%	4.1%	3.7%	4.3%
0.1	65.6%	4.5%	2.6%	2.6%
1	65.1%	4.3%	2.5%	2.9%

Figure 4: auxiliary regulation parameter vs performance metric aimed FAR is 5 percent

3.3 Prototypical net - few shot learning

The non-auxiliary model improved significantly

Lamda_aug	Avg Accuracy	Avg Standard Deviation	Avg FAR	Avg Standard Deviation
0	70.8%	1.6%	9.1%	1.6%
0.01	64.9%	1.2%	5.6%	1.2%

Figure 5: auxiliary regulation parameter vs performance metric aimed FAR is 5 percent

4 Summery

4.1 This Work

Multilingual Small Footprint KWS is a challenging problem, it looks like using Weakly supervised data (as it was used in this project) comes with tradeoff, while decreasing in accuracy the FAR was better for the auxiliary models trained.
Prototypical-net only improved the performance of the non-auxiliary model.

4.2 Further improvements

In retrospect using the auxiliary data to produce triplets like it did probably hasn't been the best way to utilize this data. perhaps a less ambitious approach of using only misclassification loss for the ivrit.ai data and triplet loss + misclassification loss on the GCS, and to implement some sort of an early exit (or at all some architectural difference between the embedding vector output and some on off-domain separation metric output), would've worked better.
personally, I believe the way to really produce Quality multilingual kws is by automating the dataset creation - to receive something similar to Google commands in other languages.

5 References

[1]Warden, Pete. "Speech commands: A dataset for limited-vocabulary speech recognition." arXiv preprint arXiv:1804.03209 (2018)

[2] M. Rusci and T. Tuytelaars, "On-Device Customization of Tiny Deep Learning Models for Keyword Spotting With Few Examples," in IEEE Micro, vol. 43, no. 6, pp. 50-57, Nov.-Dec. 2023

[3]Marmor, Yanir, Kinneret Misgav, and Yair Lifshitz. "ivrit. ai: A Comprehensive Dataset of Hebrew Speech for AI Research and Development." arXiv preprint arXiv:2307.08720 (2023)

[4] Yang, Seunghan, et al. "Improving Small Footprint Few-shot Keyword Spotting with Supervision on Auxiliary Data." arXiv preprint arXiv:2309.00647 (2023)

[5]Snell, Jake, Kevin Swersky, and Richard Zemel. "Prototypical networks for few-shot learning." Advances in neural information processing systems 30 (2017).