Releasing a toolkit and comparing the performance of language embeddings across various spoken language identification datasets

Matias Lindgren[1], *Tommi Jauhiainen*[2], *Mikko Kurimo*[1]

[1]Department of Signal Processing and Acoustics, Aalto University, Finland
[2]Department of Digital Humanities, University of Helsinki, Finland

matias.lindgren@iki.fi, tommi.jauhiainen@helsinki.fi, mikko.kurimo@aalto.fi

Abstract

In this paper, we propose a software toolkit for easier end-to-end training of deep learning based spoken language identification models across several speech datasets. We apply our toolkit to implement three baseline models, one speaker recognition model, and three x-vector architecture variations, which are trained on three datasets previously used in spoken language identification experiments. All models are trained separately on each dataset (closed task) and on a combination of all datasets (open task), after which we compare if the open task training yields better language embeddings. We begin by training all models end-to-end as discriminative classifiers of spectral features, labeled by language. Then, we extract language embedding vectors from the trained end-to-end models, train separate Gaussian Naive Bayes classifiers on the vectors, and compare which model provides best language embeddings for the back-end classifier. Our experiments show that the open task condition leads to improved language identification performance on only one of the datasets. In addition, we discovered that increasing x-vector model robustness with random frequency channel dropout significantly reduces its end-to-end classification performance on the test set, while not affecting back-end classification performance of its embeddings. Finally, we note that two baseline models consistently outperformed all other models.

Index Terms: spoken language identification, deep learning, x-vector, language embedding, TensorFlow

1. Introduction

Spoken Language Identification (SLI)[1] is the task of identifying the language of a spoken utterance. For a thorough introduction to SLI, see [1]. Automating the comparison of several different SLI models can be challenging if each model uses its own data pipeline, making it difficult to ensure that a particular comparison is not affected by unknown variability of the underlying implementations. Although several approaches to deep learning based end-to-end SLI have been proposed, there is no easy, unified way to train and compare several SLI models. We seek to remedy this situation by proposing an easy to use toolkit, built on the popular TensorFlow deep learning framework [2]. We use the toolkit to implement one baseline x-vector model, three variations of it, and three additional SLI architectures. These architectures are then trained on three SLI datasets discussed in Section 3.

X-vector SLI X-vector based SLI has in the past two years shown to be a viable alternative to i-vector based SLI [3, 4, 5, 6, 7, 8, 9], although x-vectors were originally proposed for speaker recognition [10]. Some extensions that have been proposed to the x-vector architecture include 2-dimensional (2D) convolutional neural network (CNN) feature extractor front-ends, attention mechanisms and long short-term memory (LSTM) layers [6], as well as a larger time-delayed deep neural network (TDNN) structure with residual, skip connections [11]. In the fourth oriental language recognition (AP19-OLR) challenge, an x-vector based SLI model was given as the baseline [12]. In addition to supporting end-to-end SLI, the x-vector architecture can also be used to learn a fixed-length language embedding representation for variable length utterances [4]. An alternative way of discovering embedding spaces is to explicitly map the embedded vectors onto a hypersphere by L_2-normalization, where the angular distance of embedding vectors imply class similarity. This approach has outperformed i-vector based systems both in SLI [13] and speaker recognition [14].

Contributions of this paper We publish a new, end-to-end SLI toolkit for running multiple SLI experiments on multiple datasets, implement seven existing SLI architectures on our toolkit, and run experiments on three SLI datasets. We implement the SphereSpeaker speaker recognition architecture [14] on our toolkit and apply it to SLI for the first time. We release our toolkit online as free open source software[2]. In addition, we publish[3] the configuration files, dataset metadata, and scripts for all experiments discussed in this paper to improve reproducibility of our results.

2. End-to-end deep learning SLI toolkit

Several frameworks and toolkits supporting end-to-end, automatic speech recognition (ASR) have been proposed [15, 16, 17, 18]. Applying ASR methods to SLI, e.g. by training language classifiers on phoneme embeddings extracted from a phoneme recognizer, has shown to work very well [19, 20, 21, 22]. While end-to-end SLI performed directly on labeled speech features is usually outperformed by models that utilize phoneme level information, it is sometimes possible to reach good performance also with end-to-end models [6, 23]. Our toolkit focuses only on the latter, simpler task of end-to-end SLI from spectral or cepstral features. This allows the toolkit to remain more lightweight compared to the larger frameworks that focus on ASR tasks such as sequence annotation based on connectionist temporal classification [24]. However, one trade-off is that there is no support for training multilingual bottleneck features. These must be acquired using other methods if one wishes to use them as input features. Nevertheless, our toolkit provides an easy starting point for training SLI models on an several speech datasets, therefore making the comparison of different methods significantly easier. Similar to librosa [25] providing an easy Python programming language interface for

[1]SLI is also known as Spoken Language Recognition (SLR).

[2]https://github.com/matiaslindgren/lidbox

[3]https://github.com/matiaslindgren/interspeech-2020-lidbox

audio analysis, we hope our toolkit can provide an easy way to get started with SLI experiments.

The core of our toolkit has been implemented with TensorFlow 2, which supports signal processing and model training on the GPU. Simple signal processing techniques based on the open source implementations of librosa [25] and Kaldi [15] have been implemented into our toolkit to support high performance, parallel feature extraction. Note that all feature extraction is performed with TensorFlow, librosa or Kaldi is not required. We apply dataset iterators[4] from the TensorFlow data module to construct parallelized data pipelines that support datasets with unbounded amounts of samples. All data processing steps from reading the acoustic data from disk to training a SLI model on spectral features is done in batches, allowing the user to control memory usage regardless of dataset size. Intermediate pipeline state can be cached to disk into a single binary file. This allows the user to perform all high latency operations, such as random disk access of several utterances, in a single pre-processing pass. The toolkit also supports extraction of language embedding vectors from trained end-to-end SLI models and a simple back-end training module for the language vectors. Lastly, we claim that the toolkit could also be used for end-to-end classification of speech signals beyond SLI, since the toolkit does not make any assumptions on what the provided signal labels encode.

3. Datasets

In this paper, we use three different datasets for training and testing. We did not have access to the NIST LRE datasets.

AP19-OLR Oriental Language Recognition challenge 2019 (AP19-OLR), contains speech in 10 languages mainly spoken in Asia and one out-of-set (OOS) mixture of European languages [12]. The dataset includes 261 hours of training data and 5 hours of test data. For testing, we use the AP19-OLR short-utterance task (AP19-OLR task 1), where all test utterances have a duration of exactly 1 second. For validation, we use the AP19-OLR short-utterance validation set, which contains 6 hours of exactly 1 second long utterances. The OOS mixture does not have any samples in the test or validation set.

MGB-3 3rd Multi-Genre Broadcast challenge (MGB-3), contains speech in 5 regional Arabic dialects [26]. The dataset includes 53 hours of training data and 10 hours of test data. We follow the approach of [27], who augmented the training set with randomly chosen validation set utterances. In this approach, we choose uniformly at random 90% validation set utterances separately for each of the 5 classes, create 4 new copies of each utterance, and include this 5-fold augmented validation set into the training set. This brings the amount of training data to 99 hours. The remaining 10% is used as a held-out validation set. The test set contains test utterances of varying length, with the median duration at 15 seconds.

DoSL Dataset of Slavic Languages (DoSL), contains speech in 11 Slavic languages [19]. The dataset includes 220 hours of training data and 8 hours of test data, where test utterances are almost uniformly distributed between 5 and 6 seconds. DoSL does not provide a validation set so we created our own held-out set from the training set. We choose uniformly at random 500 utterances for each of the 11 languages from the training set and remove these utterances from the training set. This results in 5500 validation set utterances (8 hours), with a median duration of 4.8 seconds.

[4] https://www.tensorflow.org/versions/r2.2/api_docs/python/tf/data/Dataset

Closed and open tasks All seven models described in Section 4 are first trained in a "closed task" manner, separately on every training set of every dataset, using the validation set of each dataset to monitor training progress. When the best weights for each model has been discovered, measured against the validation set, we evaluate the models as end-to-end language predictors on each of the test sets. Then, we use the same, trained models as feature extractors to generate fixed-length language vectors from the training and test sets of every dataset. The language vector training sets are then used for training a back-end classifier, which is evaluated as a language predictor on language vectors extracted from each test set, using each end-to-end model.

In addition, we train all models in an "open task" manner, where each model is first trained on the union of all three training sets, using the union of all three validation sets to monitor training progress. After discovering the best weights, we extract language vectors in a closed task manner, separately for each dataset and train the back-end classifiers. By doing this, we compare if end-to-end SLI models as language vector extractors benefit from training on a larger amount of data, while still extracting language vectors from a smaller amount of data.

Note that the union of all three training sets does not include the OOS mixture from AP19-OLR (label "unknown") since we could not confirm whether this mixture contains languages from MGB-3 or DoSL. In this case, including it would present multiple labels for the same language, which could have a detrimental effect for model performance in the open task.

4. End-to-end experiments

Models We use our toolkit to implement three different baseline models for the three datasets described in Section 3, one speaker recognition model, and three different x-vector variations. All models are based on existing architectures. The choice of baselines for MGB-3 and DoSL were motivated by the success previously reported for these two datasets. The baseline architecture for AP19-OLR was defined as the competition baseline by [12] and we choose the same baseline. We enumerate the model architectures used in this paper as follows:

1. Regular x-vector [4], but TDNN layers replaced with temporal convolution layers as in [27]. See Table 2 for our configuration. This is our baseline model for AP19-OLR. In addition, this is the baseline x-vector architecture for creating the variations, i.e. models 5, 6, and 7.
2. 1D CNNs with average pooling over the time dimension and three FC layers [27]. This is our baseline model for MGB-3.
3. Two bi-directional gated recurrent units (BGRU) and three FC layers [19]. This is our baseline model for DoSL.
4. SphereSpeaker architecture, that has recently been successful for speaker recognition [14], now applied to SLI.
5. Model 1 with increased robustness by applying channel dropout on input during training, using probability 0.5 [28]. See Figure 1 for an example on channel dropout applied on FBANK input.
6. Model 1 extended with additional layers before the statistics pooling layer as in [11, Table 3]. Initially, we used FC layers to extend the model but noticed that this caused unstable training progress, leading to non-finite weight values. Therefore, we use only temporal convolution layers before the statistics pooling layer, as in Model 1.

Model	10^6 params	D	mean min/epoch
1	4.5	512	5.1
2	9.0	1500	6.2
3	8.6	1024	30.9
4	5.1	1000	33.0
5	4.5	512	5.0
6	6.4	512	6.7
7	4.6	512	13.4

Table 1: *Amount of parameters in millions and the language embedding dimension D, representing the number of features, for each model. We also measured the average amount of minutes per epoch required to train each model architecture using a Tesla V100-PCIE-32GB on the open task of all three datasets.*

	Layer	Output shape
0	Input $\mathbf{X}$	198×40
1	Conv1D $512 \times 5 \times 1$	198×512
2	Conv1D $512 \times 3 \times 2$	99×512
3	Conv1D $512 \times 3 \times 3$	33×512
4	Conv1D $512 \times 1 \times 1$	33×512
5	Conv1D $1500 \times 1 \times 1$	33×1500
6	Reduce mean and stddev.	3000
7	FC ReLU 512	512
8	FC ReLU 512	512
9	FC log-softmax N	N

Table 2: *Implementation of model 1. Notation for convolution layers is "filters×kernel width×stride". All layers except 0, 6, and 9 are ReLU activated and batch normalized. Layers 1–5 are batch normalized over the time axis to avoid diluting information over different frequency channels. X-vectors are extracted as outputs of layer 7 before ReLU activation and batch normalization.*

7. Model 1 with a small 2D CNN front-end for gathering frequency information [6], see Table 3 for our 2D CNN configuration.

Model input and output All acoustic data consists of single-channel waveform signals of varying lengths at 16 kHz sample rate. Audio files with invalid headers are dropped and channels of multi-channel signals are merged by averaging. We assume the ratio of invalid files is negligible compared to all files. The list of training set audio files is shuffled before reading, separately for all three training sets. We note that energy-based voice activity detection (VAD) is used in 4 out of 7 cases in the reference experiments [4, 27, 6, 11], partially used in 2 out of 7 cases [19, 14], and not used in [28]. Therefore, we decided to use a simple energy-based VAD in all our experiments. Our VAD approach is based on comparing the root-mean-square (RMS) values of non-overlapping 10 ms windows to the mean RMS over all 10 ms windows within each signal. We require non-speech segments to contain at least 100 ms of continuous non-speech decisions before they are dropped.

	Layer	Output shape
0	Input $\mathbf{X}$	198×40
1	Add channel dimension	$198 \times 40 \times 1$
2	Conv2D $256 \times (1,5) \times (1,1)$	$198 \times 36 \times 256$
3	Conv2D $128 \times (1,3) \times (1,2)$	$198 \times 17 \times 128$
4	Conv2D $64 \times (1,3) \times (1,3)$	$198 \times 5 \times 64$
5	Conv2D $32 \times (1,3) \times (1,3)$	$198 \times 1 \times 32$
6	Flatten channels	198×32

Table 3: *2D CNN front-end of an x-vector model [6], which is prepended to model 1 to produce model 7. Notation for convolution layers is "filters×kernel size×strides". I.e. we use unit width and stride over time steps but larger height and stride over frequency channels. Outputs of layers 2–5 are ReLU activated and batch normalized.*

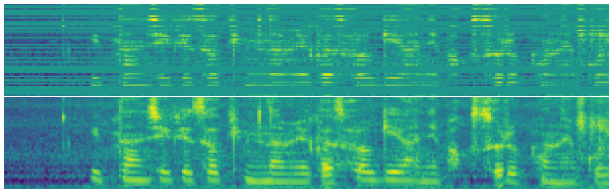

Figure 1: *Log-scale Mel-spectrogram of a randomly chosen 3 second training utterance from AP19-OLR with (top) and without (bottom) channel dropout with probability 0.5.*

After VAD, every signal that is shorter than 2 seconds is repeatedly appended to itself until its duration is at least 2 seconds. Then, all signals are divided into utterances of exactly 2 seconds, with 0.5 second overlap. A similar repeating method was used by [29], who suggested that repeating the feature sequence extracted from a short utterance is an effective way of providing more information about the utterance to the language classifier. However, we choose to repeat the short utterances already in the time domain, because using fixed length samples allows our toolkit to store all acoustic data more effectively into a single file, containing batches of 2 second utterances. In addition, using a fixed utterance length provides more comparable SLI results between different model architectures. We store all 2 second utterances into 9 files, each containing the training, validation, and test sets for all three datasets. From these 2 second utterances, log-scale Mel-spectra $\mathbf{X} \in \mathbb{R}^{198\times40}$ (FBANK) is extracted with a 512-point FFT from 25 ms windows using 10 ms offset, warped into 40 Mel-frequency bins. Finally, each channel is centered to zero mean within each $\mathbf{X}$. The same procedure is applied also to the validation and test data.

Each model outputs non-positive log-softmax language scores $\mathbf{y} \in \mathbb{R}^N$ where N is the amount of languages in the target set. Final language scores for a variable length test utterance are produced by averaging over the outputs on all its 2 second chunks. In case a model fails to produce predictions for a test utterance, smallest possible (worst case) log-softmax language scores are generated for all language classes for that test utterance.

Training and testing All seven models are trained using TensorFlow version 2.1 with the Adam optimizer [30] using learning rate 0.0001. All other optimizer parameters are left to their default values. Training samples $\mathbf{X}$ are shuffled within a buffer containing $2 \cdot 10^4$ samples, from which training batches containing 64 samples are produced. One exception is model 3, for which we apply an additional preparation step before shuffling, where each FBANK $\mathbf{X}$ is divided into non-overlapping chunks $\mathbf{X}' \in \mathbb{R}^{30\times40}$. We chose a time context of 30 time steps based on the results by [19]. In addition, the shuffle buffer size for training model 3 is $\lfloor 198/30 \rfloor \cdot 2 \cdot 10^4 = 1.2 \cdot 10^5$, to make sure the amount of information is approximately same as in the shuffle buffers of all other models. Otherwise model 3 is trained exactly as all other models. For all models, early stopping is applied with the condition that multi-class cross-entropy loss [31, Eq. 4.108] has not improved within 20 epochs from its lowest value, measured on the validation set. After early stopping, model weights are reset to the best weights, chosen from the

Model	AP19-OLR	MGB-3	DoSL	Avg
1	**0.125**	0.260	**0.019**	0.135
2	0.126	**0.236**	0.024	**0.129**
3	0.128	0.263	0.037	0.143
4	0.146	0.238	0.024	0.136
5	0.222	0.334	0.166	0.241
6	**0.125**	0.285	0.026	0.145
7	0.139	0.325	0.030	0.165
Avg	0.144	0.277	0.047	

Table 4: *C_{avg} closed task, end-to-end.*

epoch when validation loss was its lowest value.

After training, language scores $\mathbf{y}$ are predicted for each test set utterance $\mathbf{X}$. For model 3, language scores are first predicted for all $\mathbf{X}'$, which are then averaged to produce language scores for the original 2 second utterance chunk $\mathbf{X}$, from which all $\mathbf{X}'$ were partitioned from. For simplicity, we place equal weight on each chunk. Then, we use our toolkit to compute the average detection cost (C_{avg}) as defined in NIST LRE2017 [32, Eq. 6], with parameters $C_{Miss} = C_{FA} = 1$ and $P_{Target} = 0.5$, on the predicted language scores.

Results From Table 4, we compare the C_{avg} results of models 1, 2, and 3 to the respective baseline results of AP19-OLR (0.126) [12], MGB-3 (0.218) [27], and DoSL (0.013) [19], and note that our results are within 0.1, 1.8, and 2.4 percentage points. We note that model 1 outperforms other models on AP19-OLR and DoSL, while model 2 is best on its reference dataset MGB-3 and the best overall model on average. Also, model 5 clearly produces worse results compared to all other models.

5. Back-end classifiers

We choose Gaussian Naive Bayes[5] (GNB) for back-end classification because we found it to be stable and fast to train. While it is currently the only back-end classifier supported by our toolkit, we believe new classifiers are relatively easy to add, especially if they conform to the scikit-learn classifier interface[6]. The GNB models are trained on fixed-length language vectors $\mathbf{x} \in \mathbb{R}^D$ (see Table 1), which are extracted from end-to-end models trained on the closed and open tasks.

Language vector extraction For all models (except 4), $\mathbf{x}$ is extracted from a fully-connected (FC) layer, without activations and batch normalization. For models 1, 5, 6, and 7, $\mathbf{x}$ is an x-vector, i.e. the outputs of the first FC layer after the statistics pooling layer [4]. For model 2, we choose $\mathbf{x}$ as the output of the first FC layer after the average pooling layer. For model 3, we choose $\mathbf{x}$ as the output of the first FC layer after the second BGRU layer. For model 4, $\mathbf{x}$ is the L_2-normalized output of the SphereSpeaker embedding layer [14].

Classification For all three training and test sets, we feed $\mathbf{X}$ to the seven different, trained end-to-end models and collect new training and test sets of language vectors $\mathbf{x}$. All D features of each $\mathbf{x}$ are scaled to zero mean and unit variance using statistics computed separately on each training set. Dimensionality is reduced to $N-1$ by probabilistic linear discriminant analysis[7] [33] and L_2-normalization is applied on the reduced vectors. Then, GNB is fitted on the reduced $\mathbf{x}' \in \mathbb{R}^{N-1}$ training set vectors. After training, we predict log-likelihoods on the language vectors extracted from the test set. Finally, we compute C_{avg} values from these log-likelihoods using the same approach as with the log-softmax scores and report the minimum C_{avg} value as the final result. We have included this back-end training pipeline into our toolkit.

[5] https://scikit-learn.org/0.23/modules/generated/sklearn.naive_bayes.GaussianNB.html

[6] https://scikit-learn.org/stable/supervised_learning.html

[7] https://github.com/RaviSoji/plda

Model	AP19-OLR	MGB-3	DoSL	Avg
1	**0.135**	0.218	0.030	**0.128**
2	0.153	**0.211**	**0.028**	0.130
3	0.147	0.248	0.056	0.151
4	0.167	0.213	0.032	0.137
5	0.136	0.248	0.041	0.142
6	0.143	0.243	0.029	0.138
7	0.149	0.255	0.040	0.148
Avg	0.147	0.234	0.036	

Table 5: *C_{avg} closed task, GNB on embeddings.*

Model	AP19-OLR	MGB-3	DoSL	Avg
1	0.162	0.193	0.030	**0.128**
2	0.173	**0.181**	**0.028**	**0.128**
3	0.169	0.226	0.064	0.153
4	0.202	0.193	0.035	0.143
5	**0.150**	0.200	0.041	0.130
6	0.173	0.209	0.038	0.140
7	0.167	0.202	0.041	0.137
Avg	0.171	0.201	0.040	

Table 6: *C_{avg} open task, GNB on embeddings.*

Results From Table 5 we see that on MGB-3, all models except 5 are better as language vector extractors than end-to-end classifiers, but not on AP19-OLR or DoSL. We also see that frequency channel dropout (model 5) significantly weakens end-to-end SLI performance (Table 4), without affecting language embedding quality (Table 5). This is in contrast with the common assumption that speaker embedding quality is proportional to the end-to-end classification performance of the speaker recognition model used for extracting the embeddings [34]. Regarding the open task training condition, we note by comparing Tables 6 and 5 that increasing the dataset size improves the quality of language embeddings on MGB-3 even further, but at the same time reduces the results on both AP19-OLR and DoSL.

6. Conclusions

We proposed a new toolkit for easier end-to-end SLI and applied the toolkit for comparing SLI performance of different deep learning models both end-to-end and using back-end classifiers on language embedding vectors. We noticed that language embeddings on the Arabic dialect dataset MGB-3 are easier to classify with GNB when we allow an open task approach where language embedding model is trained on all available, three training sets. However, this was not beneficial for the two other datasets. In addition, we noticed that poor end-to-end SLI performance of frequency channel dropout [28] did not imply poor back-end classification performance.

7. Acknowledgements

This work was supported by EU's Horizon 2020 research and innovation programme via the project MeMAD (GA 780069). Computational resources were provided by Aalto Science-IT.

8. References

[1] H. Li, B. Ma, and K. A. Lee, "Spoken Language Recognition: From Fundamentals to Practice," *Proceedings of the IEEE*, vol. 101, no. 5, pp. 1136–1159, 5 2013.

[2] Martín Abadi et al. (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from https://tensorflow.org.

[3] A. Mccree, D. Snyder, G. Sell, and D. Garcia-Romero, "Language recognition for telephone and video speech: The jhu hltcoe submission for nist lre17." in *Odyssey*, 2018, pp. 68–73.

[4] D. Snyder, D. Garcia-Romero, A. McCree, G. Sell, D. Povey, and S. Khudanpur, "Spoken Language Recognition using X-vectors," in *Proc. Odyssey 2018 The Speaker and Language Recognition Workshop*, 6 2018, pp. 105–111.

[5] A. Hanani and R. Naser, "Spoken arabic dialect recognition using x-vectors," *Natural Language Engineering*, pp. 1–10, 5 2020.

[6] X. Miao, I. McLoughlin, and Y. Yan, "A New Time-Frequency Attention Mechanism for TDNN and CNN-LSTM-TDNN, with Application to Language Identification," in *Proc. Interspeech 2019*, 9 2019, pp. 4080–4084.

[7] H. Wu, W. Cai, M. Li, J. Gao, S. Zhang, Z. Lyu, and S. Huang, "Dku-tencent submission to oriental language recognition ap18-olr challenge," in *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 2019, pp. 1646–1651.

[8] W. Cai, J. Chen, J. Zhang, and M. Li, "On-the-fly data loader and utterance-level aggregation for speaker and language recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 1038–1051, 2020.

[9] P. Jain, K. Gurugubelli, and A. K. Vuppala, "Study on the effect of emotional speech on language identification," in *2020 National Conference on Communications (NCC)*. IEEE, 2020, pp. 1–6.

[10] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-Vectors: Robust DNN Embeddings for Speaker Recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4 2018, pp. 5329–5333.

[11] J. Villalba, N. Chen, D. Snyder, D. Garcia-Romero, A. McCree, G. Sell, J. Borgstrom, F. Richardson, S. Shon, F. Grondin *et al.*, "The JHU-MIT system description for NIST SRE18," *Johns Hopkins University, Baltimore, MD, Tech. Rep*, 2018.

[12] Z. Tang, D. Wang, and L. Song, "AP19-OLR Challenge: Three Tasks and Their Baselines," in *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 11 2019, pp. 1917–1921.

[13] G. Gelly and J. Gauvain, "Spoken Language Identification Using LSTM-Based Angular Proximity," in *Proc. Interspeech 2017*, 8 2017, pp. 2566–2570.

[14] T. Kaseva, A. Rouhe, and M. Kurimo, "Spherediar: An effective speaker diarization system for meeting data," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 12 2019, pp. 373–380.

[15] 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," *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, 2011, iEEE Catalog No.: CFP11SRW-USB.

[16] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 12 2015, pp. 167–174.

[17] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N. Enrique Yalta Soplin, J. Heymann, M. Wiesner, N. Chen, A. Renduchintala, and T. Ochiai, "ESPnet: End-to-end speech processing toolkit," in *Proc. Interspeech 2018*, 9 2018, pp. 2207–2211.

[18] M. Ravanelli, T. Parcollet, and Y. Bengio, "The pytorch-kaldi speech recognition toolkit," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6465–6469.

[19] L. Mateju, P. Cerva, J. Zdansky, and R. Safarik, "Using Deep Neural Networks for Identification of Slavic Languages from Acoustic Signal," in *Proc. Interspeech 2018*, 9 2018, pp. 1803–1807.

[20] Z. Tang, D. Wang, Y. Chen, L. Li, and A. Abel, "Phonetic Temporal Neural Model for Language Identification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 1, pp. 134–144, 1 2018.

[21] Z. Ren, G. Yang, and S. Xu, "Two-Stage Training for Chinese Dialect Recognition," in *Proc. Interspeech 2019*, 9 2019, pp. 4050–4054.

[22] M. Zhao, R. Li, S. Yan, Z. Li, H. Lu, S. Xia, Q. Hong, and L. Li, "Phone-aware multi-task learning and length expanding for short-duration language recognition," in *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 2019, pp. 433–437.

[23] L. Wan, P. Sridhar, Y. Yu, Q. Wang, and I. L. Moreno, "Tuple-max Loss for Language Identification," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5 2019, pp. 5976–5980.

[24] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. ACM, 2006, pp. 369–376.

[25] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8, 2015.

[26] A. Ali, N. Dehak, P. Cardinal, S. Khurana, S. H. Yella, J. Glass, P. Bell, and S. Renals, "Automatic Dialect Detection in Arabic Broadcast Speech," in *Proc. Interspeech 2016*, 2016, pp. 2934–2938.

[27] S. Shon, A. Ali, and J. Glass, "Convolutional Neural Network and Language Embeddings for End-to-End Dialect Recognition," in *Proc. Odyssey 2018 The Speaker and Language Recognition Workshop*, 6 2018, pp. 98–104.

[28] G. Kovács, L. Tóth, D. V. Compernolle, and S. Ganapathy, "Increasing the robustness of CNN acoustic models using autoregressive moving average spectrogram features and channel dropout," *Pattern Recognition Letters*, vol. 100, pp. 44 – 50, 2017.

[29] Z. Ma, H. Yu, W. Chen, and J. Guo, "Short Utterance Based Speech Language Identification in Intelligent Vehicles With Time-Scale Modifications and Deep Bottleneck Features," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 121–128, 1 2019.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015*, 5 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[31] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[32] S. O. Sadjadi, T. Kheyrkhah, A. Tong, C. Greenberg, D. Reynolds, E. Singer, L. Mason, and J. Hernandez-Cordero, "The 2017 nist language recognition evaluation," in *Proc. Odyssey 2018 The Speaker and Language Recognition Workshop*, 2018, pp. 82–89.

[33] S. Ioffe, "Probabilistic linear discriminant analysis," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 531–542.

[34] S. Wang, Y. Qian, and K. Yu, "What Does the Speaker Embedding Encode?" in *Proc. Interspeech 2017*, 2017, pp. 1497–1501.