Efficient Continual Learning for Keyword Spotting

Jian Vora, Anwesh Mohanty, Sai Gaurav

Department of Electrical Engineering
Indian Institute of Technology, Bombay, India
{170100026, 170070009, 170070008}@iitb.ac.in

Abstract

Significant research has been done in the field of deep learning to add new classes to an existing set of classes in a neural model, and achieve similar performance on the new classes compared to the older classes. We try to extend this idea into the field of speech recognition by considering two separate problem statements. Firstly, we choose the task of joint keyword spotting and speaker identification with the feature of online enrollment at test time, which was proposed in Interspeech'21. Next, we consider the task of continual batch learning for keyword spotting networks, where at each time step we only have access to the current speech data but none of the older data for training. In this setting, we propose a time-efficient solution to ensure that the speech model performance on older training datasets doesn't deteriorate as newer data keeps on coming.
Index Terms: keyword spotting, speaker identification, transfer learning, domain adaptation, continual learning

1. Problem Statement

1.1. Joint Keyword Spotting (KWS) and Speaker Identification (SpkID)

The problem can be considered as a binary classification task where at test time, if an utterance matches with an already enrolled keyword *and* is said by the same speaker then output 1 else output a 0. In addition to this, we might get new users saying different keywords at test-time. Hence these new users need to be enrolled to the system in an online manner.
We have thought about a few approaches to this problem which are mentioned below:

1. Start off with the simplest baseline of having a fixed number of speakers and train independent KWS and verification systems on the dataset provided in the INTERSPEECH AutoKWS challenge. The final system is then simply product of the outputs of the individual systems.
2. Try out ideas like weight sharing of LSTMs for both the tasks and formulate a joint multi-task kind of loss function (if time permits). The multiple tasks which we want sharing between are indeed keyword spotting and speaker verification [1].
3. We explore ideas from domain adaptation which shall help us use the already trained model and adapt it to a new setting and hope to get a good common feature extractor. We borrow ideas from domain adaption in image classification by adding a new regularizer on the model weights which helps in transfering across various domains.
4. If time permits, we might look into speech representations and methods involving wav2vec models to achieve better results in this task.

1.2. Continual Batch Keyword Spotting

The problem statement is as follows, initially we have a fixed set of data with fixed set of labels at t=0. We train a model on this data and achieve the required results. At t=t_1, a new dataset is introduced with a different set of models. But now we don't have access to the previous data, we only have access to the trained model parameters. In this setting, we need our model to perform well on the current data as well the old data. At t=t_2, another new dataset with new labels is introduced and this goes on. Naturally the model is bound to "forget" about the older data given it keeps on seeing new data, and performance on it will deteriorate. Since we do not access to the older data it is not possible to join both the old and new datasets and train the model on them together. Moreover, combining the datasets will anyway lead to longer training times. Hence we need an efficient solution to remedy this problem.
For this task, we have looked into several domain adaptation techniques used in image recognition/computer vision problems and feel that the theory behind those methods will hold in this setup also. We have currently explored the use of the Elastic Weight Consolidation (EWC), Learning Without Forgetting (LWF) and Continuous Replay Adaptation (CRA) in our project and obtained significantly better results compared to the baseline models. The results can be seen in Section3.
Two experiments were carried out using the above methods. In the first one, the dataset is divided into class batches (D_A, D_B, D_C) and corresponding model (M_i) is trained sequentially with the use of above domain adaptation for higher datasets and feature extractor of model M_{i-1}. During test phase, this model is used to test on different batches using their respective softmax layers (S_i). The accuracies are compared across different methods and varying the hyper-parameter in the loss function. In the second experiment, we do away with our previous assumption that we know to which batch a sample belongs to at test time. So the model at any time instant can receive samples from classes that are not part of the current classes that it is training on. For this setup, we increase the number of classes at any step by 1, to account for the "unknown" class. For samples being classified as "unknown" in S_i, it is passed to classify via S_{i-1}, until it finds the batch where it is classified not as "unknown".
In the subsequent sections, we outline our progress on each of the above tasks, describe the dataset and the experiments carried out.

1.3. Dataset Description

We use the dataset provided by the competition organizers for the first task of Joint KWS and SpkID. All data are recorded by near-field mobile phones, (located in front of the speakers at around 0.2m distance). Each sample is recorded in single channel, 16-bit streams at a 16kHz sampling rate [2]. There

are a total of 100 speakers each having their own different keywords. There are a total of 10 utterances per speaker which are positive (saying the assigned keyword) and 30 other utterances where the speaker id is the same however the utterance need not match with the keyword. These 2 basically serve as natural splits for training keyword spotting and speaker identification respectively. Each speaker may potentially speak in a different language and hence we need to handle these multi-lingual variations as well in any sort of feature extractor we build. Apart from these we have 5 other speakers which are to be enrolled at test time.
The other dataset which we have used for the continual batch KWS task is the trimmed version of Speech Commands dataset [3] (same as used in Assignment 2) with 105,829 one second long utterances and 35 different classes. We introduce different splits in the classes to perform the continual learning tasks.

2. Joint Keyword Spotting and Speaker Identification tasks

In this section, we outline our progress in the first point defined in the problem statement, namely training a keyword spotting system followed a speaker identification block. We report the architechture for each of these, the training methods and the final results for both of these.

2.1. Keyword Identification using TC-ResNet

Due to dearth of labelled data as stated above, we try to experiment with transfer learning for the keyword identification task. The output is clearly a 100 length softmax output as each speaker utters a different keyword. We use the architecture of Temporal Convolution for Real-time Keyword Spotting (TC-ResNet) [4] and the weights from the pretrained model on Speech Commands dataset .

The input to this model is the time-freqeucny MFCC of the raw audio. TC-ResNet using temporal convolutions for the MFCC instead on convential 2D convolutions. They employ temporal convolutions to increase the effective receptive field and follow the original ResNet implementation for other layers by adopting strided convolutions and excluding dilated convolutions. The final softmax layer was replaced and changed by training on the above dataset. The numbers reported are the multiclass classification accuracies (think of it as the Hamming Loss). The final resuts were as follows:

Train Accuracy: 98.12
Validation Accuracy: 85.64

2.2. Speaker Identification using SincNet

For speaker recognition, we use the architecture of SincNet [5] which is like a basic CNN based classifier however the first layer consists of sinc filters (bandpass filters in frequency domain). This bank of filters has varying cutoff frequencies which helps in focussing different parts of the input audio signal. Besides, the sinc functions reduce the number of parameters on the SincNet first layer because each sinc function of any size only have two parameters to learn against L from the conventional convolutional filter, where L is the size of the filter. As a result, the sinc functions enables the network to converge faster. This was trained end-to-end using the training data as more utterances were available for speaker identification task. We intially also tried using transfer learning in this as well with pretrained weights from the TIMIT dataset. Using transfer learning, the results were not that great as mentioned below which motivated us to train end-to-end:

Train Accuracy: 74.32
Test Accuracy: 57.83

For end-to-end training we explore two different loss functions - the first one being the conventional softmax over a 100 length vector and the second was a modified additive margin softmax (AM-Softmax) [6]. The additive margin works as a better class separator than the traditional decision boundary from softmax. Furthermore, it also forces the samples from the same class to become closer to each other thus improving results for tasks such as classification. The results were:

Train Accuracy SincNet: 86.53, **AM-SincNet:** 99.74
Validation Accuracy SincNet: 72.74, **AM-SincNet:** 86.17

2.3. Combined Results

We now describe the final results after combining the keyword spotting and speaker identification systems. For this, we simply multiply the softmax outputs (after thresholding to 0/1) of the above two models and if the final vector is all 0, we predict 0 and if an index is 1 (there can be only one such), then the device is activated as both the keyword and speaker are verified.

Train Accuracy: 88.56
Validation Accuracy: 59.34

We observe a big drop in the validation accuracy. For this, per speaker we had set aside 12 utterances of the same - 8 of them should be predicted as 0 as the keyword is not present and 4 of them should output 1. Thus we had a total of 1200 validation utterances for which this accuracy is calculated. We had expected the final accuracy to be around the product of individual accuracies which comes out to be around 70. We noticed that both the KWS and verification models were not matching even though they performed well on independent tasks. Out of the 400 to be allowed, 147 of them were not allowed by the KWS system itself to proceed further. In general, the system predicted a lot of zeros as compared to ones (false positive rate was less) and hence the final accuracy took a hit. We also realised that the original problem statement needed us to enroll new users at test time which we had not handled and hence switched gears to the next problem statement on continual batch keyword spotting.

3. Continual Batch Keyword Spotting

In this section, we first outline the basic problem of considering batch-wise data and outline the idea of domain adaptation which is conventionally used for image classification. We consider two similar problem statements in which we have different domains, each domain consists of a different set of words, and we need to make a feature extractor that can make accurate predictions on each domain. The domains arrive continually and we don't have access to the previous domains at any point of time. In the first experiment, we assume that during test time we know from which domain a test sample is chosen from. In the second experiment we let go of this ideal assumption and introduce an "unknown" class to account for out-of-vocabulary words at each stage. We have formally shown the final framework of the model in a later section. First we describe about the domain adaptation techniques we use to ensure that our model doesn't forget about the previous domains.

3.1. Domain Adaptation Basics

In the problem statements defined above, we assume that we don't have access to the previous domains. In such cases of domain shift, the model would ideally forget the important characteristics of the older domains. Domain adaptive methods mitigate the harmful effects of domain shift by learning transformations that find common invariant features between different domains and hence ensure that the performance on the older domains doesn't deteriorate. In an unsupervised setup as ours, the problem of domain shift is even much severe. Hence we adopt three popular techniques used in domain adaptation in image datasets, with the hope that the basic foundations of those methods can be applied to our speech task and prevent forgetting as much as possible when we move from one dataset to the next.

These methods are included as penalties with the loss function during training on any domain and constraint the model to learn features which are embedded in all the domains we have encountered till now. We briefly discuss about the techniques below.

3.1.1. Elastic Weight Consolidation (EWC)

EWC [7, 8] tries to capture the information of past tasks into the posterior distribution $p(\theta|\mathcal{D}_{old})$. Since the true posterior is intractable, it is approximated as a gaussian distribution with mean given by the parameters θ^*_{old} (optimal weights for the old task) and the diagonal precision given by the diagonal of the Fisher Information matrix F. F is equivalent to the second derivative of the loss function near a minimum and can easily be calculated. Given this approximation, the final loss function in EWC can be written as

$$\tilde{L}(\theta) = L_{new}(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{old,i})^2 \qquad (1)$$

λ is a hyperparameter. The summation i is over all parameters with $_i$ being a single parameter, and F_i is the diagonal element of corresponding Fisher information matrix of the parameter when training on task A. $L_{new}(\theta)$ is the loss for the current task only. When moving to a third task, task C, EWC will try to keep the network parameters close to the learned parameters of both task A and B.

3.1.2. Learning Without Forgetting (LWF)

Given a neural network with shared parameters θ_s and task specific parameters θ_{old}, LWF [9] tries to learn parameters θ_{new} that work well on both the old and new tasks by using only samples and labels from the new task. For this, we introduce an extra term in our loss function which measures how much the features generated by θ_{new} and θ_{old} differ in the feature space. If our feature extractor works well on both the old and new tasks, the features generated on the current task by both the feature extractors shouldn't be far apart. So for this, the penalty applied is a L2 penalty on the features. The final loss function using LWF can be written as

$$\tilde{L}(\theta) = L(\theta_{new}) + \lambda \sum_{X_i} (\theta_{new}(X_i) - \theta_{old}(X_i))^2 \qquad (2)$$

where λ is the hyperparameter, $L(\theta_{new})$ is the loss function only on the new task and X_i belongs to the set of samples in the new task.

3.1.3. Continuous Replay Adaptation (CRA)

CRA [10] tries to address the issue of forgetting previous domain information by randomly saving the scores for a few previously seen examples and using a "replay loss" (L_{replay}) to ensure that the response of current model to the same examples is the same as before. Hence at every stage of adaptation on a dataset, we produce a mini-dataset with a few selected observations from the specific dataset together with their predicted scores/labels. The final loss function using CRA can be written

$$\tilde{L}(\theta) = L(\theta_{new}) + \lambda L_{replay}(C(FE(X_p)), Y_p) \qquad (3)$$

where λ is the hyperparameter that controls how much to optimize for past domain efficiency, $L(\theta_{new})$ is the loss function only on the new task and X_p, Y_p are the random samples and their predicted scores saved from the previous domain/dataset. For our purpose, we have used cross entropy loss as our replay loss.

3.2. Experiments and Results

3.2.1. Experiment I

We perform experiments on the Speech Commands dataset [3] with 35 classes. We split the dataset into smaller datasets, with each dataset containing a subset of words; train a keyword classifier for one split and then once the other splits are presented, we tune the weights of the common feature extractor of both of them so as to ensure that the overall performance remains good. The feature extractor (FE) is a standard CNN model with residual connections to prevent vanishing gradient and the classifier (C) used is a linear layer with softmax activation to help in predicting the required classes.

The flow of the experiments can be understood with a small example. Assume the original dataset has been split into 2 datasets; dataset A with 15 classes and dataset B with 20 classes. The model (FE + C) is trained on dataset A first. Then it is presented with dataset B. A naive method can be to initialize the FE of the model of dataset B with parameters from A and simply train the classifier normally. But this will disregard the important features learnt by the model about dataset A and hence the FE will perform poorly if it is plugged back into A. Instead we use the aforementioned DA methods (EWC/LWF/CRA) to ensure that the model performs well on both the datasets.

The final results by setting the optimal value of hyperparameters are presented in Table 1 (2 splits only) and Table 2 (3 splits). For just two splits A (15 classes) and B (20 classes), the accuracy on A before adaptation was **86.78%**.

Method	A	B
Baseline	42.51	86.04
EWC ($\lambda = 15$)	63.2	80.48
LWF ($\lambda = 0.0005$)	44.5	86
CRA ($\lambda = 1$)	53.17	86.52

Table 1: *Domain Adaptation Results for 2 splits*

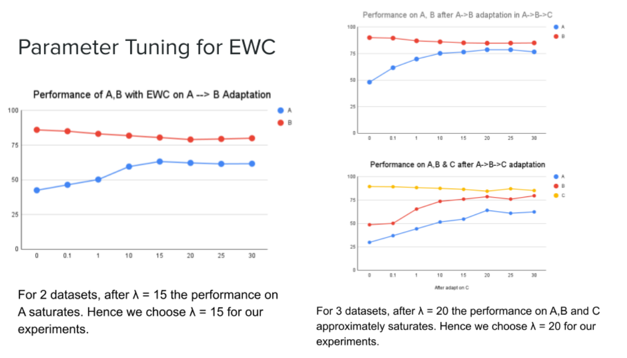
Next, we present the same results but for 3 splits A, B, and C with the number of classes being 12, 12, and 11 respectively.

The accuracy on A before adaptation was **89.26%** and the order of adaptation was A followed by B followed by C.

Method	$A \rightarrow B$		$A \rightarrow B \rightarrow C$		
	A	B	A	B	C
Baseline	48.1	90.1	29.82	48.67	89.45
EWC ($\lambda = 20$)	78.65	84.79	64.05	78.62	84.48
LWF ($\lambda = 0.0005$)	62.56	91.68	37.42	62.77	91.14
CRA ($\lambda = 1$)	61.34	82.01	49.29	62.35	88.36

Table 2: *Domain Adaptation Results for 3 splits*

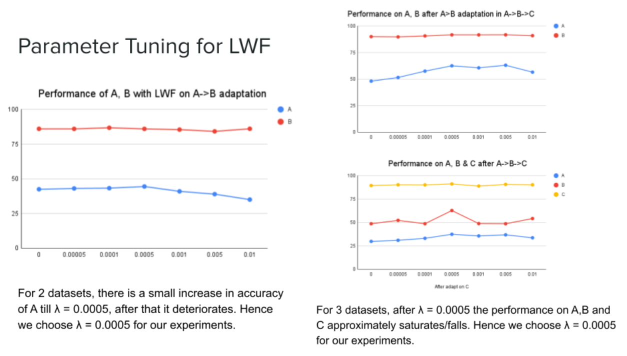
Next, we show some plots which were used to tune the hyperparameters and get the optimal ones whose results were shown in Table 1 and Table 2 above. The one on the left is for the 2-split case while the ones on the right are for the 3-split case.



For 2 datasets, after λ = 15 the performance on A saturates. Hence we choose λ = 15 for our experiments.

For 3 datasets, after λ = 20 the performance on A,B and C approximately saturates. Hence we choose λ = 20 for our experiments.

Figure 1: *Parameter tuning on EWC*



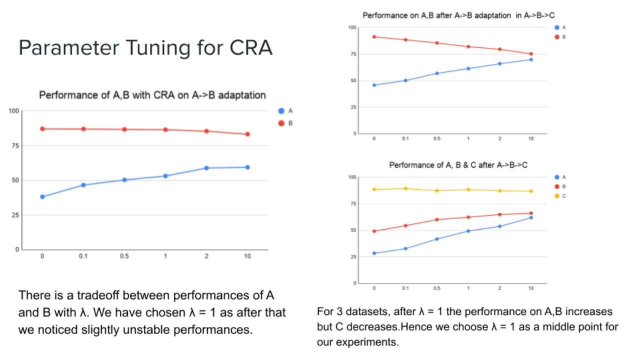
For 2 datasets, there is a small increase in accuracy of A till λ = 0.0005, after that it deteriorates. Hence we choose λ = 0.0005 for our experiments.

For 3 datasets, after λ = 0.0005 the performance on A,B and C approximately saturates/falls. Hence we choose λ = 0.0005 for our experiments.

Figure 2: *Parameter tuning on LWF*

Some observations from the above results are presented -

1. Applying EWC during adaptation leads to significantly better results compared to the baseline. The current model is able to perform well on the current dataset as well as the older datasets.
2. LWF doesn't work very well in this setup as can be seen from the results and parameter tuning graphs. There is only a marginal increase in performance at most times.
3. CRA works quite well but the results are still inferior compared to EWC. The performance on older domains increases with λ but the model becomes more unstable as λ is kept on increasing.



There is a tradeoff between performances of A and B with λ. We have chosen λ = 1 as after that we noticed slightly unstable performances.

For 3 datasets, after λ = 1 the performance on A,B increases but C decreases.Hence we choose λ = 1 as a middle point for our experiments.

Figure 3: *Parameter tuning on CRA*

3.2.2. *Experiment II*

In this experiment, we do away with our previous assumption that we know to which dataset a sample belongs to at test time. So the model at any time instant can receive samples from classes that are not part of the current classes that it is training on. For this setup, we increase the number of classes at any step by 1, to account for the "unknown" class. Consider an example in which dataset A has 12 classes. We train a model with 13 classes and use samples from other classes (not in the 12) and give them "unknown" label. Next, the previously defined DA methods are used to train the model on dataset B. During test time now, a sample first goes through the feature extractor and softmax layer of B and checks if it is unknown or not. If it is unknown, then we check it through the softmax layer of A to see if belongs to dataset A. This can be better understood with the help of Fig 4.

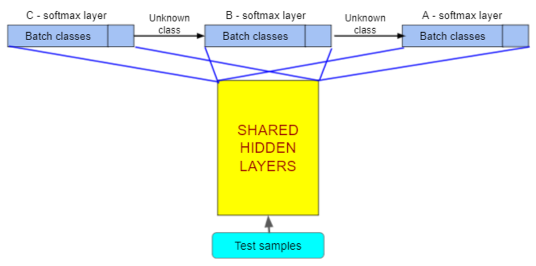


Figure 4: *Model used in combined testing*

Just like previous experiment, 3 batches of dataset (A, B, C), 10 classes each, were assumed. After training B with adaptation on A, the above model is tested with samples from both A and B (referred to as AB in Table 3) and on individual batches too. This process is again repeated after adaptation on the next batch C, testing using samples from all batches (referred to as ABC in Table 3) and on the above described different domain adaptation methods.
The following observations can be made from the above results-

1. With the combined testing approach used, the accuracies on test dataset becomes lower as more batches are adapted, with recently adapted batch having the highest accuracy.
2. The accuracies on individual batch samples are also lower compared to Expt.1 due to increasing possibility

Test dataset	EWC	LWF	CRA
	A $\rightarrow B$		
AB	69.67	61.24	71.34
B	82.29	86.11	84.37
A	58.06	47.51	57.18
	A $\rightarrow B \rightarrow C$		
ABC	53.43	41.28	56.38
C	79.99	65.10	68.18
B	40.82	38.17	45.86
A	37.43	34.36	36.27

Table 3: *Domain Adaptation for 3-way split*

of misclassification. For example, sample belonging to class of lower batch, can be correctly classified only if it is classified as the unknown labels in the higher batches.

3. On average, the accuracy metrics for EWC and CRA loss seems to perform better than LWF for same adaptation scenarios and test data.

4. Conclusions and Future Work

From the two different types of experiments carried out for continual learning, it is clear that the assumption made during test time of experiment 1 is a very crucial one i.e. if we know to which domain a test sample belongs to, then the model performance will be much better as we won't have to worry about misclassifications happening due to the unknown class. Moreover, the DA methods employed in the project are certainly able to reduce the amount of forgetting happening due to domain shift by a considerable amount; it might still not be enough especially when more domains are introduced. Based on the experiments, we can also conclude that EWC in general performs better than CRA and LWF (LWF performs the worst), which is expected as EWC imposes a much stronger constraint on the model compared to the other 2 methods.

One possible line of future work could be to do away with the extra unknown class label and introducing a discriminator which when given a sample gives us the dataset/domain to which the sample belongs to. Once we have that, we can just perform inference on the model for that particular dataset and get the exact label for the sample to be tested.

5. References

[1] S. Sigtia, E. Marchi, S. Kajarekar, D. Naik, and J. Bridle, "Multi-task learning for speaker verification and voice trigger detection," *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2020. [Online]. Available: http://dx.doi.org/10.1109/ICASSP40776.2020.9054760

[2] J. Wang, Y. He, C. Zhao, Q. Shao, W.-W. Tu, T. Ko, H. yi Lee, and L. Xie, "Auto-kws 2021 challenge: Task, datasets, and baselines," 2021.

[3] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018.

[4] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, and S. Ha, "Temporal convolution for real-time keyword spotting on mobile devices," 2019.

[5] M. Ravanelli and Y. Bengio, "Speaker recognition from raw waveform with sincnet," 2019.

[6] J. A. Chagas Nunes, D. Macedo, and C. Zanchettin, "Additive margin sincnet for speaker recognition," *2019 International Joint Conference on Neural Networks (IJCNN)*, Jul 2019. [Online]. Available: http://dx.doi.org/10.1109/IJCNN.2019.8852112

[7] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," 2017.

[8] A. Madasu and V. A. Rao, "Sequential domain adaptation through elastic weight consolidation for sentiment analysis," 2020.

[9] Z. Li and D. Hoiem, "Learning without forgetting," 2017.

[10] A. Bobu, E. Tzeng, J. Hoffman, and T. Darrell, "Adapting to continuously shifting domains," 2018. [Online]. Available: https://openreview.net/forum?id=BJsBjPJvf