EfficientLEAF: A Faster LEarnable Audio Frontend of Questionable Use

Jan Schlüter and Gerald Gutenbrunner
Institute of Computational Perception
Johannes Kepler University Linz, Austria
jan.schlueter@jku.at, gerald.gutenbrunner@gmail.com

***Abstract*—In audio classification, differentiable auditory filterbanks with few parameters cover the middle ground between hard-coded spectrograms and raw audio. LEAF [1], a Gabor-based filterbank combined with Per-Channel Energy Normalization (PCEN), has shown promising results, but is computationally expensive. With inhomogeneous convolution kernel sizes and strides, and by replacing PCEN with better parallelizable operations, we can reach similar results more efficiently. In experiments on six audio classification tasks, our frontend matches the accuracy of LEAF at 3% of the cost, but both fail to consistently outperform a fixed mel filterbank. The quest for learnable audio frontends is not solved.**

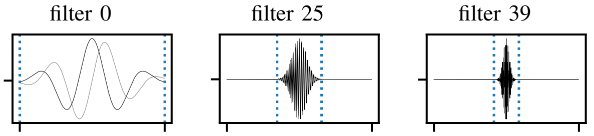
***Index Terms*—audio classification, CNNs, time-frequency representation, adaptive filterbanks**

I. Introduction

Deep-learning-based audio classification models typically operate on precomputed spectrograms – this holds for Convolutional Neural Networks (CNNs) [2], Recurrent Neural Networks [3], and Transformers [4]. This places the burden of choosing optimal spectrogram settings for a task on the practitioner, who may decide not to tune these at all, possibly resulting in suboptimal performance. Alternatively, models may be trained directly on raw audio samples. However, this gives the model much more free parameters, and only matches the performance of spectrogram-based models when given large quantities of training data [5], [6]. A solution in between these extremes is to apply a filterbank that is differentiable with respect to a small number of parameters, and learn these parameters along with the classifier.

A recent promising instance of the latter was proposed by Zeghidour et al. [1] and called LEarnable Audio Frontend (LEAF). In their experiments, it outperforms earlier proposals by other authors when evaluated over a range of tasks in different audio domains (speech, music, environmental audio). However, as it is based on two convolutions (a Gabor filterbank and temporal pooling) with large windows and small strides, and normalization by a sequentially computed exponential moving average (Per-Channel Energy Normalization, PCEN [7]), it is two orders of magnitude slower than typical spectrograms.

In this work, we propose two modifications of LEAF to improve computational performance by an order of magnitude without hindering trainability or hampering classification accuracy. Specifically, our modifications are:



LEAF	conv 401 // 1 pool 401 // 160	conv 401 // 1 pool 401 // 160	conv 401 // 1 pool 401 // 160
Ours	conv 401 // 40 pool 11 // 4	conv 123 // 4 pool 101 // 40	conv 69 // 1 pool 401 // 160

Fig. 1. The original LEAF implementation convolves the input with filters of 401 samples at stride 1, followed by squared modulus and temporal pooling of 401 samples at stride 160. We reduce filter lengths for higher bandwidths (dashed lines) and increase stride for lower center frequencies, adjusting pooling to approximate the original LEAF output.

– We adapt convolution window sizes and strides dynamically for subsets of filters, giving nearly the same results with less computation (Figure 1).
– We replace the sequentially computed normalization (PCEN) with a learnable logarithmic compression, temporal median subtraction and temporal batch normalization, all of which are parallelizable and thus faster to compute on Graphics Processing Units.

We evaluate our modifications against the original LEAF and fixed mel spectrograms on six tasks in three audio domains (speech, music, environmental audio). Contrary to Zeghidour et al. [1], we find that *none of the frontends has a clear advantage* over the others in terms of resulting accuracy.

The remainder of this paper is structured as follows: Section II discusses related work, followed by an introduction of LEAF and our modifications in Section III. In Section IV, we present experimental results. Section V concludes the paper.

II. Related Work

Existing attempts at implementing learnable filterbanks can be divided into categories based on two features of the filters: (1) The domain of operation, either time or frequency, and (2) the generation of coefficients, either given directly or produced by a parameterized function. We will discuss selected examples of each category.

Sainath et al. [8] learn the coefficients of frequency-domain filters initialized to a mel filterbank, constrained to their bandwidth at initialization, for speech recognition. Cakir et al. [9] remove this constraint, freely learning all coefficients

ISBN: 978-1-6654-6798-8

for sound event detection. In both cases, filters deviate from their initial triangular form, but stay close to a mel filterbank.

As examples of frequency-domain parametric filterbanks, Seki et al. [10] and Schlüter [11, p. 189] learn the center frequencies of Gaussian and triangular filters, respectively. Compared to directly learning coefficients, this reduces the number of learnable parameters and gives better interpretable filters, but is still based on a predefined, hand-tuned STFT.

Most freedom is attained by learning time-domain filter coefficients, as done e.g. by Palaz et al. [12], Tüske et al. [13], Sainath et al. [5] or Zeghidour et al. [14] for speech recognition. Time-domain filters are often initialized to a mel [14] or gammatone [5], [13] filterbank and followed by temporal pooling [5], [12], [14]. Early works failed to match performance of precomputed spectrograms [12], [13], which only changed with larger datasets [5] and models [14].

Parametric time-domain convolutions reduce learnable parameters and introduce inductive biases that may help training from limited datasets. Existing work includes Sinc [15], Sinc^2 [16], Wavelet [17], Gabor [16], [18], Gammatone [16], and Gammachirp [19] filters with learnable center frequencies and/or bandwidths for spectral decomposition, and average pooling [19], max pooling [15], [16] and smoothing windows [18] for temporal downsampling.

All these works have in common that they are evaluated on a single dataset, often in the speech domain, leaving open whether a demonstrated advantage over fixed filterbanks transfers to other datasets, tasks or domains. In contrast, LEAF [1] was evaluated on eight tasks, and shown to outperform TF-banks [14], SincNet [15] and fixed mel filterbanks.

III. Method

We will first describe LEAF [1], the starting point of our work, then detail our modifications of the filterbank and compression/normalization stages.

A. LEAF

LEAF applies a Gabor filterbank, squared modulus, temporal averaging and subsampling, and a compression/normalization in sequence to an audio signal to obtain a time-frequency representation consumed by a classifier. Stages are initialized to approximate a mel spectrogram, and optimized along with the classifier. We will briefly describe each stage in the following.

Filterbank: The first step is a convolution of the input signal with complex Gabor filters in the time domain. Given a center frequency ν, inverse bandwidth σ_c and odd filter size C, filter coefficients c_t are computed as:

$$c_t = e^{i2\pi\nu t}\frac{1}{\sqrt{2\pi}\sigma_c}e^{-\frac{t^2}{2\sigma_c^2}} \quad \text{for } t \in \{-\frac{C-1}{2},\dots,\frac{C-1}{2}\}$$

The coefficients are differentiable w.r.t. ν and σ_c. A filterbank of N filters is thus parametrized by a vector of N center frequencies and N inverse bandwidths.

Squared modulus: The N convolved signals are squared elementwise, resulting in real-valued sequences.

Averaging: Each sequence is convolved with a Gauss window. Given an inverse bandwidth σ_p and odd pooling size P, window coefficients p_t are computed as:

$$p_t = \frac{1}{\sqrt{2\pi}\sigma_p}e^{-\frac{t^2}{2\sigma_p^2}} \quad \text{for } t \in \{-\frac{P-1}{2},\dots,\frac{P-1}{2}\}$$

Formally, the N convolved sequences are then subsampled by keeping every Kth value (practically, a strided convolution is applied that only computes every Kth output). The averaging stage is parametrized by a vector of N inverse bandwidths, such that pooling can be tuned separately for each filter.

Compression/Normalization: Finally, Per-Channel Energy Normalization (PCEN [7]) is applied to each sequence. Given ϵ, α, δ, r and s, and denoting the input sequence as x_t, the output sequence y_t is:

$$y_t = \left(\frac{x_t}{(\epsilon + m_t)^\alpha} + \delta\right)^r - \delta^r,$$

where m_t is computed using a simple infinite impulse response (IIR) filter:

$$m_0 = x_0, \quad m_t = (1-s)m_{t-1} + s\,x_t$$

This process is applied separately to each of the N sequences, using a separate set of learnable parameters for each (except for ϵ, which is fixed). The result is a division of each frequency band by its long-term past magnitude (sequence m_t), and a nonlinear compression by raising to the power of r. Wang et al. [7] learned the logarithm of α, δ, r; Zeghidour et al. [1] instead learn the inverse of r and enforce $\alpha \leq 1$, $r \leq 1$.

B. EfficientLEAF

We are now ready to discuss our changes to the LEAF filterbank, pooling and normalization/compression stages.

Filterbank: LEAF initializes filters to a mel scale, with roughly logarithmically increasing center frequencies and bandwidths. With increasing bandwidth, filter energy concentrates in fewer coefficients (see Figure 1). We can thus save computations by truncating the filter. Specifically, we compute a filter size $\hat{C} = b\,\sigma_c$ and round up to the next odd integer, where σ_c is the inverse bandwidth and b can be tuned to trade accuracy for computation. Khan et al. [17] proposed to do so for a complete filterbank, here we compute C for each filter separately. With decreasing center frequency, filter responses are smoother over time, and change less from sample to sample. We can thus save computations by increasing the convolution stride. Specifically, we compute $\hat{L} = d\,\pi/\nu$ and round down to the next divisor of the pooling stride P, where ν is the center frequency and d can be tuned to trade accuracy for computation. Since convolution implementations profit from applying multiple same-sized filters at once, we group adjacent filters and pick the largest filter size and smallest stride per group. The number of groups g, ideally a divisor of N, becomes another hyperparameter.

Pooling: Both the pooling stride P and the pooling scale σ_p need to be divided by the convolution stride L to match results of the original LEAF. Figure 1 gives the resulting window sizes and strides for matching the default settings of LEAF, with $b = 4.75$ chosen to reproduce a maximal window size of 401 at initialization, $g = 4$, and $d = 1$ chosen conservatively.

Normalization/Compression: The sequential computation of the exponential moving average in PCEN is not suited well for massively parallel hardware. We replicate some of its effects by different means. As a first step, we compute $y_t = \log(1 + 10^a\, x_t)$, where a is a separate learnable parameter for each frequency band. This results in a nonlinear compression similar to exponentiation by r. PCEN's division by an exponential moving average levels out different impulse responses of recording devices, or static noise floors. As we applied an (approximate) logarithm, we require subtraction instead of division. To avoid the exponential moving average, we subtract the median over the sequence instead (separately per frequency band). As this improves performance for some tasks only, reducing it for others, we keep the original sequence as a second input channel. Finally, we normalize the sequence with batch normalization over time, using separate learnable parameters per frequency band and channel.

IV. Experiments and Results

We can now empirically compare EfficientLEAF to LEAF, and to a fixed mel filterbank. We will first introduce the datasets used, then describe training and model settings, and finally present results for three experiments: Our main comparison, a hyperparameter optimization of EfficientLEAF, and an extension to longer input sequences.

A. Datasets

For our experiments, we employ five datasets with six tasks:
– *SpeechCommands*: one-second recordings of 35 spoken commands; 84843 training, 9981 validation, 11005 test
– *VoxForge*: variable-length recordings in 6 languages; 128594 training, 44119 validation, 30136 test
– *Crema-D*: variable-length recordings displaying 6 emotions; 5144 training, 738 validation, 1555 test
– *NSynth*: 4-second recordings of 11 instruments in 128 pitches; 289205 training, 12678 validation, 4096 test
– *BirdCLEF 2021*: variable-length recordings of 397 bird species; 40836 training, 5637 validation, 16401 test
If no split was published along with the data, we use the one from tensorflow_datasets[1]. Unfortunately, Zeghidour et al. [1] did not publish their splits, and we could not reproduce any.

B. Settings

We set up LEAF to match [1]: An input sample rate of 16 kHz, 40 filters initialized with a mel scale from 60 Hz (lower bound of first filter) to 7800 Hz (upper bound of last filter), a convolution and pooling window size of 401 samples, and a pooling stride of 160 samples. Pooling scales σ_p are initialized to 0.4. PCEN is initialized with $\alpha = 0.96$, $s = 0.04$, $\delta = 2$, $r = 0.5$ and has $\epsilon = 10^{-12}$. For EfficientLEAF, we set $b = 4.75$, $d = 1$, $g = 4$, $a = 5$ as a close match to LEAF, but we perform a parameter search in our second experiment.

For classification, we follow [1] and add an EfficientNet-B0 [20] backbone with global max pooling instead of global average pooling, and a single linear classification layer.

To train the model, Zeghidour et al. [1] used ADAM with mini-batches of 256 randomly chosen one-second excerpts, and ran 1 million updates at a constant learning rate of 10^{-4}. This amounts to thousands of epochs depending on the dataset, and a constant learning rate seems suboptimal. Instead, we start with an initial learning rate of 10^{-3}, reduce it by a factor of ten when the validation loss does not improve for ten consecutive epochs, and stop training when the learning rate falls below 10^{-5}. This improves results for all frontends.

At test time, we compute predictions for non-overlapping one-second excerpts and average logits per recording, following [1] except that final incomplete excerpts are dropped, not padded (which skews results as no padding occurs in training).

C. Model Comparison

In our first set of experiments, we compare a set of models on the six tasks. Starting with the original LEAF, we first replace the filterbank and pooling with our grouped version, then replace PCEN with our combination of log compression, median filtering and temporal batch normalization ("L-M-TBN"). Finally, we replace the filterbank and pooling with a fixed STFT-based mel filterbank (also using a window size of 401 and stride 160) and hold log compression fixed.

Table I lists the results (ignore the second to last column for now). Focusing on throughput (forward + backprop), we see that the grouped Gabor filterbank at its conservative settings is 3x as fast, and replacing PCEN gives another 5% (this will be more pronounced for longer input sequences). Fixed mel spectrograms are 100x faster and could even be precomputed. In terms of accuracy, there seems to be a consistent decline when replacing PCEN for VoxForge. However, results for VoxForge are either extremely sensitive to the split, or models are overfitting: On the validation set, accuracies behave inversely, improving from 74.2% for LEAF to 79.8% for a fixed mel filterbank. The poor performance of PCEN-based frontends on Crema-D, the smallest dataset, warrants investigation.

D. Hyperparameter Optimization

EfficientLEAF has three parameters affecting its efficiency and accuracy: The convolution window size factor b, convolution stride factor d, and number of groups g. We perform a grid search with $b \in \{2, 4.75, 6\}$, $d \in \{1, 2, 3, 8, 16\}$ and $g \in \{2, 4, 8, 10\}$, doing 3 training runs on SpeechCommands each. For space constraints, we can only summarize results. For almost all settings, $g = 8$ is the fastest. $b = 2$ slightly deteriorates results, $b = 6$ is only marginally slower than $b = 4.75$. d scales computational speed almost linearly, without affecting results on this task in the range of considered values. This is in line with Dörfler et al. [18], who use a stride of 21 for a sample rate of 22050 Hz. In Table I, the previous to last column shows results with $g = 8$, $b = 6$ and $d = 16$, which match the more conservative settings of $g = 4$, $b = 4.75$ and $d = 1$ at much better efficiency.

[1] https://tensorflow.org/datasets, accessed June 12, 2022

TABLE I
THROUGHPUT OF AUDIO FRONTEND IN EXAMPLES PER SECOND (ON ONE-SECOND EXCERPTS) AND ACCURACY ON SIX TASKS (MEAN ± STD. DEV. OVER THREE RUNS), FOR FIVE COMBINATIONS OF FILTERBANK AND COMPRESSION/NORMALIZATION. (*: PARAMETERS FIXED, NOT LEARNED)

Filterbank Compression	Gabor PCEN	Gabor 4G PCEN	Gabor 4G L-M-TBN	Gabor 8G-opt L-M-TBN	STFT-Mel* L-M-TBN*
Throughput	250	742	776	9251	85367
SpeechCommands	95.1 ± 0.3	95.1 ± 0.1	**95.3** ± 0.2	95.2 ± 0.1	95.1 ± 0.2
VoxForge	**91.5** ± 0.4	91.4 ± 0.9	86.5 ± 0.9	86.6 ± 1.0	85.6 ± 0.6
Crema-D	50.2 ± 2.3	50.0 ± 2.6	58.0 ± 2.8	**60.2** ± 0.8	58.8 ± 3.2
NSynth Instr.	69.2 ± 0.2	68.3 ± 1.2	70.4 ± 0.5	71.7 ± 0.6	**72.1** ± 0.7
NSynth Pitch	92.2 ± 0.1	92.1 ± 0.1	**92.7** ± 0.2	92.4 ± 0.1	91.9 ± 0.3
BirdCLEF 2021	42.3 ± 0.7	42.3 ± 0.8	42.0 ± 0.1	**42.9** ± 0.1	39.9 ± 1.9

TABLE II
THROUGHPUT AND ACCURACY FOR THE FIRST, THIRD AND FOURTH MODEL FROM TABLE I ON BIRDCLEF 2021, TRAINED ON 8- OR 16-SECOND EXCERPTS.

length (s) batchsize	8 32	16 16
#1 thrpt.	27	12
#1 acc.	71.9 ± 0.4	69.6 ± 0.4
#3 thrpt.	95	48
#3 acc.	71.4 ± 0.9	66.0 ± 2.4
#4 thrpt.	1053	516
#4 acc.	72.2 ± 0.3	69.4 ± 0.3

E. Longer Input Sequences

Following [1], all results discussed so far were obtained by training and evaluating on one-second audio excerpts. This recipe is not applicable to every audio classification task. For example, for weakly-labeled recordings, not every excerpt will be discriminative, as is the case for the BirdCLEF 2021 data. In this setting, it will be necessary to train on longer excerpts.

Table II shows results for training and evaluating the original LEAF, the default EfficientLEAF and optimized EfficientLEAF on either 8-second or 16-second excerpts, for BirdCLEF 2021. Two observations are important: (1) longer excerpts indeed perform dramatically better, and (2) while EfficientLEAF throughput scales inversely linearly with input length, LEAF is put at a larger disadvantage, increasing the gap in throughput. This is due to PCEN: As it has to process each item sequentially, a batch of 32 8-second excerpts allows fewer parallel computations than a batch of 256 1-second excerpts, stalling the GTX 1080 Ti used for testing.

V. DISCUSSION

We have demonstrated that LEAF [1] can be modified to improve computational efficiency, especially for long input sequences, without impacting accuracy on downstream tasks. We also found that LEAF may not be needed: Our deviation from Zeghidour et al. [1] in training and inference (Sec. IV-B) and compression (Sec. III-B) improved results, but also narrowed the advantage of LEAF over a fixed mel filterbank. Whether and why LEAF is beneficial will require further scrutinization and experiments, and maybe our implementation (github.com/CPJKU/EfficientLEAF) can speed up this process.

Regarding EfficientLEAF, an interesting feature has not been explored yet: Since convolution window sizes are chosen dynamically, it could learn to analyze lower frequencies than would be permitted by a predefined window size, or be initialized to cover a much wider range of frequencies than affordable with a fixed window.

Finally, during experimentation, we observed that learned center frequencies and bandwidths do not deviate much from their initial values (in line with [1, A.3]). As in [11, 189–190], we tried increasing the frontend learning rate. This indeed allows some frontend parameters to converge, but reduces classification performance, asking for a better solution.

REFERENCES

[1] N. Zeghidour, O. Teboul, F. de Chaumont Quitry, and M. Tagliasacchi, “LEAF: a learnable frontend for audio classification,” *ICLR*, 2021.

[2] Q. Kong, Y. Cao, T. Iqbal, Y. Wang, W. Wang, and M. D. Plumbley, “Panns: Large-scale pretrained audio neural networks for audio pattern recognition,” *TASLP*, vol. 28, pp. 2880–2894, 2020.

[3] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *ICASSP*, 2013, pp. 6645–6649.

[4] Y. Gong, Y.-A. Chung, and J. Glass, “AST: Audio Spectrogram Transformer,” in *Interspeech*, 2021, pp. 571–575.

[5] T. N. Sainath, R. J. Weiss, A. W. Senior, K. W. Wilson, and O. Vinyals, “Learning the speech front-end with raw waveform cldnns,” in *Interspeech*, 2015, pp. 1–5.

[6] J. Pons, O. Nieto, M. Prockup, E. M. Schmidt, A. F. Ehmann, and X. Serra, “End-to-end learning for music audio tagging at scale,” in *ISMIR*, 2018, pp. 637–644.

[7] Y. Wang, P. Getreuer, T. Hughes, R. F. Lyon, and R. A. Saurous, “Trainable frontend for robust and far-field keyword spotting,” in *ICASSP*, Mar. 2017, pp. 5670–5674.

[8] T. N. Sainath, B. Kingsbury, A.-r. Mohamed, and B. Ramabhadran, “Learning filter banks within a deep neural network framework,” in *IEEE Workshop on Automatic Speech Recognition and Understanding*, 2013, pp. 297–302.

[9] E. Cakir, E. C. Ozan, and T. Virtanen, “Filterbank learning for deep neural network based polyphonic sound event detection,” in *IJCNN*, 2016, pp. 3399–3406.

[10] H. Seki, K. Yamamoto, and S. Nakagawa, “A deep neural network integrated with filterbank learning for speech recognition,” in *ICASSP*, 2017, pp. 5480–5484.

[11] J. Schlüter, “Deep learning for event detection, sequence labelling and similarity estimation in music signals,” Ph.D. dissertation, Johannes Kepler University Linz, Austria, Jul. 2017.

[12] D. Palaz, R. Collobert, and M. Magimai-Doss, “Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks,” in *Interspeech*, 2013, pp. 1766–1770.

[13] Z. Tüske, P. Golik, R. Schlüter, and H. Ney, “Acoustic modeling with deep neural networks using raw time signal for LVCSR,” in *Interspeech*, 2014, pp. 890–894.

[14] N. Zeghidour, N. Usunier, I. Kokkinos, T. Schaiz, G. Synnaeve, and E. Dupoux, “Learning filterbanks from raw speech for phone recognition,” in *ICASSP*, 2018, pp. 5509–5513.

[15] M. Ravanelli and Y. Bengio, “Speaker recognition from raw waveform with sincnet,” in *SLT*. IEEE, 2018, pp. 1021–1028.

[16] E. Loweimi, P. Bell, and S. Renals, “On learning interpretable cnns with parametric modulated kernel-based filters,” in *Interspeech*, 2019, pp. 3480–3484.

[17] H. Khan and B. Yener, “Learning filter widths of spectral decompositions with wavelets,” in *NeurIPS*, vol. 31, 2018, pp. 4601–4612.

[18] M. Dörfler, T. Grill, R. Bammer, and A. Flexer, “Basic filters for convolutional neural networks applied to music: Training or design?” *Neural Comput. Appl.*, vol. 32, no. 4, pp. 941–954, 2020.

[19] I. López-Espejo, Z. Tan, and J. Jensen, “Exploring filterbank learning for keyword spotting,” in *EUSIPCO*. IEEE, 2020, pp. 331–335.

[20] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in *ICML*, 2019, pp. 6105–6114.