Differentiable Time-Frequency Scattering On GPU

John Muradeli, Cyrus Vahidi, Changhong Wang, Han Han, Vincent Lostanlen, Mathieu Lagrange, George Fazekas

DIFFERENTIABLE TIME–FREQUENCY SCATTERING ON GPU

John Muradeli[*], *Cyrus Vahidi*[#], *Changhong Wang*[+], *Han Han*[+], *Vincent Lostanlen*[+], *Mathieu Lagrange*[+], *George Fazekas*[#]

[+]LS2N, CNRS, Nantes Université
École Centrale Nantes, France
firstname.lastname@ls2n.fr

[#]Center for Digital Music
Queen Mary University of London, UK
f.lastname@qmul.ac.uk

[*]john.muradeli@gmail.com

ABSTRACT

Joint time–frequency scattering (JTFS) is a convolutional operator in the time–frequency domain which extracts spectrotemporal modulations at various rates and scales. It offers an idealized model of spectrotemporal receptive fields (STRF) in the primary auditory cortex, and thus may serve as a biological plausible surrogate for human perceptual judgments at the scale of isolated audio events. Yet, prior implementations of JTFS and STRF have remained outside of the standard toolkit of perceptual similarity measures and evaluation methods for audio generation. We trace this issue down to three limitations: differentiability, speed, and flexibility. In this paper, we present an implementation of time–frequency scattering in Python. Unlike prior implementations, ours accommodates NumPy, PyTorch, and TensorFlow as backends and is thus portable on both CPU and GPU. We demonstrate the usefulness of JTFS via three applications: unsupervised manifold learning of spectrotemporal modulations, supervised classification of musical instruments, and texture resynthesis of bioacoustic sounds.

1. INTRODUCTION

Human listening plays a central role in the development and evaluation of digital audio effects (DAFx) [1]. Yet, listening tests are costly and time-consuming as they typically rely on expert participants. For this reason, recent publications have proposed to mimic the behavioral response of the average listener by means of a computational surrogate [2, 3]. In particular, experimental findings in auditory neurophysiology suggest that our primary cortex responds selectively to spectrotemporal modulations at various rates and scales [4]. Each of these responses may be simulated by an idealized model known as spectrotemporal receptive field (STRF). Hence, the space of STRF coefficients appears as a natural candidate for comparing two sounds out of context; and indeed, studies in music psychology have confirmed that Euclidean distances in STRF space approximate timbre dissimilarity judgments between isolated musical notes [5].

However, despite its potential for developing perceptually informed audio synthesis, the STRF has received limited adoption within the DAFx community as well as music information retrieval (MIR) and machine learning for signal processing (MLSP). Indeed, we notice three shortcomings in the NSL Auditory–Cortical Toolbox [6], which we consider to be the reference implementation of STRF [1]. First, it lacks scalability: the code is written in MATLAB, does not accommodate parallel computing, and is not portable onto GPU hardware. Second, it lacks flexibility: the toolbox assumes that the audio input has a sample rate of 16 kHz and subsamples all subbands in the constant-Q filterbank to 125 Hz, even though the critical sample rate should depend on center frequency so as to minimize memory usage while avoiding aliasing artifacts. Thirdly, it lacks differentiability: although the authors do provide a modified Griffin-Lim algorithm for reconstructing an audio signal from its STRF coefficients, MATLAB's NSL and related toolboxes do not perform reverse-mode automatic differentiation, unlike PyTorch or TensorFlow. The same three issues are found again in the toolbox "strf-like-model" of [3], which is a Python port of the NSL Auditory–Cortical Toolbox using NumPy as its backend [2].

In this paper, we present a Python implementation of "Joint Time-Frequency Scattering" (JTFS) [7], that is, a fast and numerically accurate discretization of STRF. While there have been implementations of JTFS in MATLAB since 2014[3] [4], ours is the first to support GPU computing and automatic differentiation. To accomplish this, we have extended Kymatio[5], a library for wavelet-based processing in Python released in 2019 [8], with code that now constitutes a permanent branch called dafx2022-jtfs[6], in WaveSpin, a library currently under development. We show the potential of the implementation to different research topics with three examples: unsupervised manifold learning of spectrotemporal modulations, supervised classification of musical instruments, and texture resynthesis of bioacoustic sounds. Beyond the demonstrated use cases, differentiable implementations of scattering have potential to enable parametric scattering filterbanks [9] and audio synthesis loss functions. Our supervised classifier is the first instance of "hybrid representation learning" [10] which interfaces JTFS with a 2-D deep convolutional network (convnet). Furthermore, we outline an activation function for scattering-based neural networks, under the name of mean-based logarithm (μ–log). We demonstrate state-of-the-art musical instrument classification results in the setting of limited annotated training data. For the sake of reproducibility, we provide open-source code, with experiments reproduced in a repository named JTFS-GPU[7], alongside supplementary material[8].

Cyrus Vahidi is a researcher at the UKRI CDT in AI and Music, supported jointly by the UKRI (grant number EP/S022694/1) and Music Tribe. This work was conducted while at LS2N, CNRS. Changhong Wang is supported by an Atlanstic2020 project on Trainable Acoustic Sensors (TrAcS).

[1] http://nsl.isr.umd.edu/downloads.html
[2] https://github.com/EtienneTho/strf-like-model
[3] https://www.di.ens.fr/data/software/scatnet
[4] https://github.com/lostanlen/scattering.m
[5] https://www.kymat.io/
[6] Time-frequency scattering: https://github.com/OverLordGoldDragon/wavespin/tree/dafx2022-jtfs
[7] Experiments repository: https://github.com/cyrusvahidi/jtfs-gpu
[8] Companion website: https://cyrusvahidi.github.io/jtfs-gpu

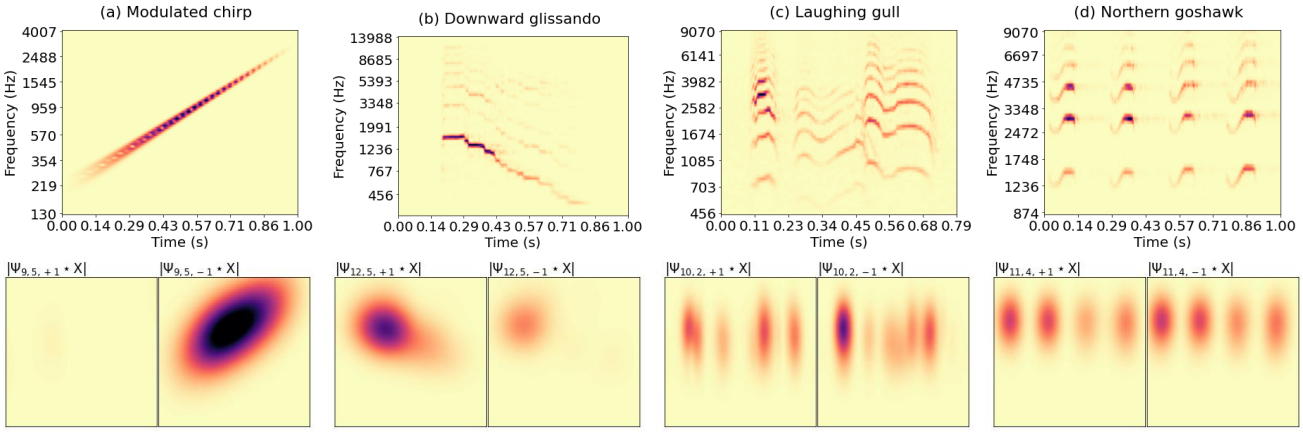

Figure 1: Scalograms (top) and scale–rate visualizations (bottom) from the joint time–frequency scattering transforms of exemplary audio signals: (a) amplitude-modulated chirp signal; (b) downward glissando musical instrument playing technique; (c) and (d) are the sounds of two types of birds, the laughing gull and the Northern goshawk. Each scale–rate visualization shows the response of a second-order 2-D wavelet ψ of temporal rate α, frequential scale β and orientation $\theta = \pm 1$ when convolved with the scalogram X. See Section 2 for details on joint time–frequency scattering.

2. TIME–FREQUENCY SCATTERING

Proposed in [7], the *joint time–frequency scattering* (JTFS) transform captures the spectrotemporal modulations of a signal at various rates and scales. This is achieved by decomposing the signal with joint wavelet convolutions, nonlinearities, and pooling operations.

Let $\boldsymbol{\psi}(t)$ and $\boldsymbol{\psi}(\lambda)$ denote the basis function ("mother wavelet") for the decomposition along the time, t, and the log-frequency axis, λ, respectively. Both wavelets are complex analytic, of which the Fourier transform is null for negative frequencies, i.e. $\hat{\boldsymbol{\psi}}(\omega) = 0$ for $\omega < 0$. Our implementation uses the Morlet wavelet, i.e. a complex sinusoid modulated by a Gaussian envelope, due to its quasi-optimality in terms of Heisenberg time–frequency uncertainty. $\boldsymbol{\psi}_\lambda(t)$ is the temporal wavelet filterbank dilated from $\boldsymbol{\psi}(t)$. Convolving a waveform $\boldsymbol{x}(t)$ with each wavelet in $\boldsymbol{\psi}_\lambda(t)$ and applying pointwise complex modulus yields the scalogram $\mathbf{X}(t,\lambda) = \left|\boldsymbol{x} * \boldsymbol{\psi}_\lambda\right|(t)$, which is a two-dimensional (2-D) time–frequency image, as shown by the examples in Fig. 1 (top).

To extract the spectrotemporal modulations of an STRF centered at (t,λ), we decompose the scalogram with a joint time–frequency wavelet $\boldsymbol{\Psi}_{\alpha,\beta,\theta}(t,\lambda)$. α and β are the temporal rate and frequential scale, respectively. $\theta = \pm 1$ is the orientation of the STRF, with $\theta = -1$ denoting a positive slope while $\theta = +1$ a negative one. $\boldsymbol{\psi}_\alpha(t)$ and $\boldsymbol{\psi}_\beta(\lambda)$ are the temporal and the frequential wavelet filterbank dilated from their mother wavelets, $\boldsymbol{\psi}(t)$ and $\boldsymbol{\psi}(\lambda)$, respectively:

$$\begin{aligned} \boldsymbol{\psi}_\alpha(t) &= 2^\alpha \boldsymbol{\psi}(2^\alpha t) \quad \text{and} \\ \boldsymbol{\psi}_{\beta,\theta}(\lambda) &= 2^\beta \boldsymbol{\psi}(\theta 2^\beta \lambda). \end{aligned} \tag{1}$$

The joint time–frequency wavelet $\boldsymbol{\Psi}_{\alpha,\beta,\theta}(t,\lambda)$ is the outer product between the temporal wavelet $\boldsymbol{\psi}_\alpha(t)$ and the frequential wavelet $\boldsymbol{\psi}_{\beta,\theta}(\lambda)$:

$$\boldsymbol{\Psi}_{\alpha,\beta,\theta}(t,\lambda) = \boldsymbol{\psi}_\alpha(t)\boldsymbol{\psi}_{\beta,\theta}(\lambda). \tag{2}$$

We then convolve the scalogram with the 2-D joint wavelet filterbank $\boldsymbol{\Psi}_{\alpha,\beta,\theta}(t,\lambda)$, apply a complex modulus, and average it by a 2-D lowpass filter $\boldsymbol{\Phi}_{T,F}(t,\lambda)$. Following [7], we define the joint time–frequency scattering of $\mathbf{X}(t,\lambda)$ as:

$$\mathbf{S}_2^{\mathrm{JTFS}}\boldsymbol{x}(t,\lambda,\alpha,\beta,\theta) = \left(\left|\mathbf{X} \overset{t,\lambda}{*} \boldsymbol{\Psi}_{\alpha,\beta,\theta}\right| \overset{t,\lambda}{*} \boldsymbol{\Phi}_{T,F}\right)(t,\lambda). \tag{3}$$

The symbol $\overset{t,\lambda}{*}$ denotes a 2-D convolution over both the time variable t and the log-frequency variable λ. Hence, for each location around (t,λ) in the time–frequency domain, we obtain a 3-D tensor indexed by (α,β,θ), capturing spectrotemporal modulation information. $\mathbf{S}_2^{\mathrm{JTFS}}\boldsymbol{x}$ in Eq. (3) has invariance properties to time-shifts, time-warps, and frequency transpositions, for a receptive field restricted by the time scale T and frequency interval F. In certain cases, we may omit frequential averaging in order to preserve equivariance to frequency transposition. This results in a variant of Eq. (3):

$$\mathbf{S}_2^{\mathrm{JTFS}}\boldsymbol{x}(t,\lambda,\alpha,\beta,\theta) = \left(\left|\mathbf{X} \overset{t,\lambda}{*} \boldsymbol{\Psi}_{\alpha,\beta,\theta}\right| \overset{t}{*} \boldsymbol{\Phi}_{T}\right)(t,\lambda). \tag{4}$$

Fig. 1 shows the scalogram (top) and the scale–rate visualizations of JTFS (bottom) for four audio examples. (a) is a synthesized amplitude-modulated chirp signal of constant chirp rate, see Section 3.2 for details. (b) is a glissando playing technique performed on the Chinese bamboo flute *(dizi)*; (c) and (d) are the vocalisations of two species of birds: the laughing gull and the Northern goshawk *(Accipiter gentilis)*. The first two examples are isolated events with upward and downward frequency change, respectively, while the remaining two are real-world acoustic events with temporal variations in chirp rate and directionality.

In Fig. 1, we visualize JTFS coefficients before averaging, i.e. $|\mathbf{X} * \boldsymbol{\Psi}_{\alpha,\beta,\theta}|$, with one scale–rate combination for each of the audio examples. All visualizations cover the complete time t and log-frequency λ axes, and both directions of θ. For instance, Fig. 1 (a) bottom displays the JTFS obtained by convolving $\mathbf{X}(t,\lambda)$ with $\boldsymbol{\Psi}_{9,5,-1}$ and $\boldsymbol{\Psi}_{9,5,1}$ respectively, taking complex modulus and lowpass filtering. 9 and 5 denote the temporal rate and frequential scale indices, respectively. The scale–rate visualizations of (b), (c), and (d) are obtained in the same way. As can be seen in

the first two isolated examples (a) and (b), the direction of the spectrotemporal patterns are clearly captured: the JTFS energy concentrates on $\theta = -1$ for upward frequency changes in (a), while for the downward frequency variations in (b), $\theta = +1$ dominates the spectrotemporal modulations. The main directions of frequency modulations of each isolated event in examples (c) and (d) are also captured. We explore these two examples further in Section 5 in a task of audio texture resynthesis via JTFS coefficients.

3. SIMILARITY RETRIEVAL

3.1. Motivation

In this section, we compare the abilities of audio representations to serve as a similarity measure between audio signals with real-world factors of variability. We design a sound synthesizer to generate a dataset of *amplitude modulated chirp* (AM/FM) signals, that is controlled by three parameters: carrier frequency (f_c, in Hz), amplitude modulation frequency (f_m, in Hz) and chirp rate (γ, in octaves/second). Such amplitude and frequency modulations are typically found within musical instrument playing techniques [11]. We visualize similarity of the synthesized signals on a manifold embedding and assess recovery of the synthesizer parameters under several audio representations: Mel-frequency cepstral coefficients (MFCCs), time scattering (Scattering1D), time–frequency scattering (JTFS), spectrotemporal receptive fields (STRFs) and OpenL3 embeddings. MFCCs result from computing a log-mel spectrogram (logmelspec) followed by a discrete cosine transform (DCT). The cortically-inspired spectrotemporal receptive field (STRF) transformation serves as a representation of spectrotemporal modulations [3]. OpenL3 embedding is a deep feature representation that results from training L3-Net for audiovisual correspondence [12].

3.2. Synthetic Dataset of Modulated Chirps

We define a generator $\boldsymbol{g}$ of exponential "chirps" with three factors of variability: a carrier frequency f_c, an amplitude modulation (AM) frequency f_m, and a frequency modulation (FM) rate γ. Denoting by $\boldsymbol{\theta}$ the triplet (f_c, f_m, γ), we have for every $\boldsymbol{\theta}$:

$$\boldsymbol{g_\theta} : t \longmapsto \phi_w(\gamma t) \sin(2\pi f_m t) \sin\left(\frac{2\pi f_c}{\gamma \log 2} 2^{\gamma t}\right), \qquad (5)$$

where ϕ_w is a Gaussian window of characteristic width equal to w. The AM/FM signal $\boldsymbol{g_\theta}$ has an instantaneous frequency of $f_c 2^{\gamma t}$ and an essential duration of w/γ. Thus, it covers a bandwidth w, independently from $\boldsymbol{\theta}$. We set $w = 2$ octaves in the following.

We highlight a physical correspondence of the synthesizer parameters to the wavelet variables in Section 2. f_c corresponds to the log-frequency λ of first-order scattering wavelets. Amplitude modulation frequency f_m can be adequately described by second-order temporal wavelet rate α. The chirp rate γ accounts for a relationship between frequential wavelets of scale β and rate α, i.e. $\beta = \frac{\alpha}{\gamma}$.

We apply Eq. (5) for 16 values of f_c, f_m, and γ, arranged in a geometric progression; hence yielding a dataset of $16^3 = 4096$ audio signals in total. We vary f_c between 512 Hz to 1024 Hz; f_m, between and 4 Hz to 32 Hz; and γ, between 0.5 and 4 octaves/second respectively. Fig. 2 illustrates the constant-Q transform of two chirp signals of 512 Hz fundamental frequency, with chirp rates $\gamma = 0.5$ and $\gamma = 4$ octaves per second and equal bandwidth 2 octaves[9].

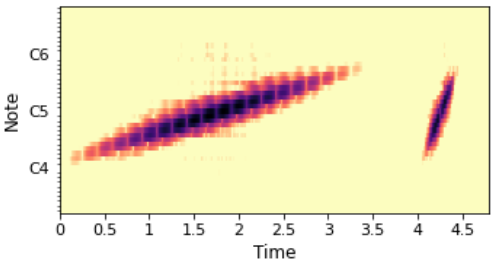

Figure 2: constant-Q transform of two AM/FM signals with chirp rate $\gamma = 0.5$ (left) and $\gamma = 4$ (right) octaves/second, respectively. Both cases have a carrier frequency f_c of 512 Hz, a modulation frequency f_m of 3 Hz, and a bandwidth equal to two octaves.

3.3. Manifold Learning and Visualization

To visualize similarity relationships between the AM/FM signals, we apply the Isomap algorithm for unsupervised dimensionality reduction [13]. Isomap assembles a geodesic distance matrix by using neighborhood relationships from high-dimensional Euclidean distances. We first compute the MFCCs, Scattering1D and JTFS coefficients, STRFs and OpenL3 embeddings over the dataset of AM/FM signals. Under the Isomap algorithm, we consider each representation separately. To compute the nearest neighbor graph, we consider the 40 nearest neighbors for each transformed data point. We select three components for the manifold visualization. The audio dataset described in Section 3.2 characterizes three independent degrees of freedom, therefore we postulate that Isomap will reveal whether the coordinates of an audio representation reflect similarities within the AM/FM signals. Musical instrument playing techniques are a class of signals that vary significantly in amplitude and frequency modulation content. A recent publication showed that distances on the K-nearest-neighbor graph of JTFS reflected human similarity judgements between playing techniques [2].

We compute time–frequency scattering coefficients by means of our newly introduced implementation. We transform each of the 4096 signals synthesized via Eq. (5), setting $J = 14$ octaves, $Q = 8$ filters per octave and $T = 1000$ ms. We omit frequential averaging to preserve equivariance to pitch transposition (see Eq. (4)). We set $J = 14$ to enable analysis of slower modulations via the temporal filterbank, with center frequencies reaching approximately 0.1 Hz. We also compute time scattering (Scattering1D) coefficients using Kymatio [8], setting $Q = 1$ and $J = 14$ with global temporal averaging. Time scattering does not capture spectrotemporal patterns beyond a log-frequency interval $1/Q_f$, where Q_f is the quality factor (ratio of center frequency to bandwidth). Hence, by setting $Q = 1$, which results in $Q_f = 2.5$, we guarantee that the scalogram contains at least one amplitude modulation cycle, given a modulation frequency of at least 4 Hz and a chirp rate of at most 4 octaves per second.

Fig. 3(b) and (c) show three-dimensional (3-D) visualizations of the Isomap embeddings for time scattering ($Q = 1$) and time–frequency scattering ($Q = 8$), respectively. In the case of both transformations and the application of Isomap manifold learning,

[9] See companion website: https://cyrusvahidi.github.io/jtfs-gpu

the dataset of AM/FM signals is represented as a 3-D mesh where the principal components align independently with f_c, f_m and γ. Both transformations with their respective hyperparameters are capable of disentangling and linearizing fundamental frequency, tremolo rate and chirp rate, which describe spectrotemporal modulation patterns. Fig. 3(c) visualizes the embedding for time scattering when $Q = 8$. In this case, we observe that time scattering lies on a 2-D manifold that adequately describes f_c and γ, yet fails to account for similarity in f_m due to the aforementioned reasons. Despite time scattering successfully disentangling the 3 factors of variability when $Q = 1$ (Fig. 3(b)), other applications may demand a a greater quality factor in order to better localize in frequency.

As a comparison, we also compute Isomap embedding for the dataset's MFCCs (Fig. 3(a)), STRFs (Fig. 3(e)), and OpenL3 embeddings (Fig. 3(f)). We compute MFCCs using librosa V0.8 default parameters, yielding 20 coefficients [14]. STRFs are computed by means of the 'strf-toolkit' [3] using the default parameters and setting the input duration to 4 seconds. OpenL3 embeddings are extracted using the *music* model of the publicly available Python package[10], resulting in 6144 coefficients after globally averaging in time.

We observe that in the case of MFCCs, the Isomap embedding forms a curved 2-D manifold, whereas our dataset contains three factors of variability. Only the fundamental frequency f_c clearly aligns with one of the Cartesian coordinates. Meanwhile, similarities between amplitude modulation rates f_m and chirp rates γ are not represented faithfully. Therefore, neighboring points on the graph may have very dissimilar values of f_m and γ. This is also the case for STRFs and OpenL3, where proximity relationships in the Isomap embeddings do not reflect similarity in tremolo rate f_m. In our experiment, the STRF fails to retrieve similarity in amplitude modulation rates. To determine the cause of this outcome demands a more thorough investigation as this behavior is contrary to its theoretical specification. Fundamental frequency and chirp rate are disentangled onto independent components, yet high chirp rates are densely clustered for all carrier frequencies.

3.4. Regression from Nearest Neighbors

As a quantitative supplement to the visualizations from the previous section, we assess regression of the synthesizer's three parameters $\boldsymbol{\theta} = (f_c, f_m, \gamma)$ with K-nearest neighbors regression algorithm (K-NN). K-NN parameter regression relies on Euclidean distances between examples in their feature representations. Therefore, its regression error sheds light on the degree of topological alignment between feature space and parameter space. Such an alignment is essential in common audio recognition tasks, as the parameters are physical correspondents of audio similarity.

For each example, we start from an empty set of neighbors $\mathcal{N}_0 = \varnothing$. Then at each iteration, we select its closest neighbor by computing its pairwise Euclidean distance with all other examples. We stop after K iterations, resulting in a set of K nearest neighbors.

$$\mathcal{N}_{k+1}(\boldsymbol{\theta}_i) = \mathcal{N}_k(\boldsymbol{\theta}_i) \cup \left\{ \underset{\boldsymbol{\theta}_j \notin \mathcal{N}_k(\boldsymbol{\theta}_i)}{\arg\min} \; \|\mathbf{S}g(\boldsymbol{\theta}_j) - \mathbf{S}g(\boldsymbol{\theta}_i)\|_2 \right\} \quad (6)$$

We compute an estimate of the parameter $\widetilde{\boldsymbol{\theta}}_i$ as the average of its values at the K nearest neighbors. We define the error ratio as

[10] https://openl3.readthedocs.io

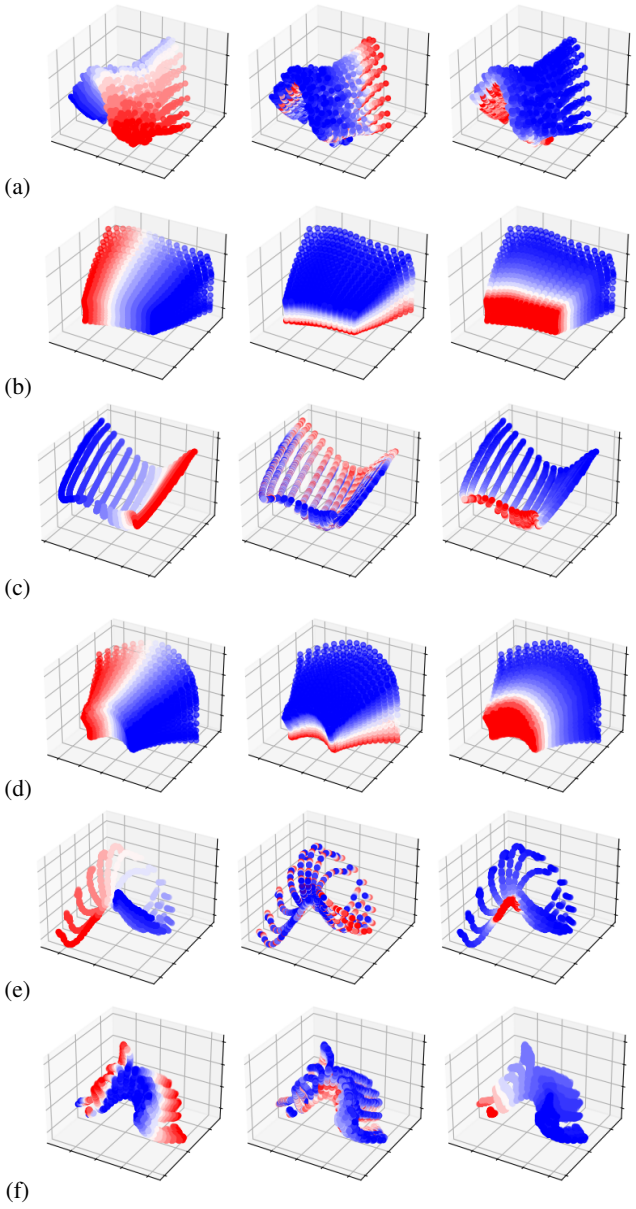

Figure 3: Isomap embeddings of a synthetic dataset of amplitude modulated sinusoidal chirps (see Section 3.2) represented by: (a) MFCCs, (b) time scattering ($Q = 1$) and (c) ($Q = 8$), (d) time–frequency scattering, (e) STRFs and (f) OpenL3 embeddings. The configurations for these features are outlined in Section 3.3. The embedding is fully unsupervised, using acoustic features alone. The colour range of the markers indicates an increasing value from blue to red via white, corresponding to the signals' carrier frequency f_c (left), tremolo rate f_m (center) and chirp rate γ (right).

$\widetilde{\boldsymbol{\theta}}_i / \boldsymbol{\theta}_i$, where:

$$\widetilde{\boldsymbol{\theta}}_i = \frac{1}{K} \sum_{\theta_j \in \mathcal{N}_K(\theta_i)} \boldsymbol{\theta}_j. \quad (7)$$

We use the same $K = 40$ nearest neighbor graph computed by the Isomap algorithm in the previous section. We regress each exam-

ple's parameters for each of the audio representations and plot their error ratios in Fig. 4. All feature representations are capable of regressing carrier frequency f_c with error ratios close to 1. However, larger performance discrepancies can be observed in modulation frequency and chirp rate. Aligned with our observations in Section 3.3, time scattering and JTFS excel at linearizing modulation frequency in the Euclidean space, with error ratios within range of 0.75 to 1.5. Meanwhile, all features except MFCCs extract chirp rate within error ratios between 0.75 to 1.25.

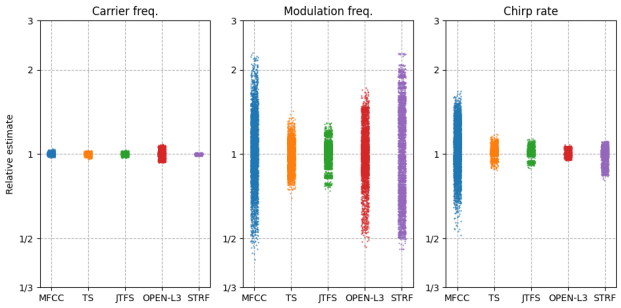

Figure 4: Error ratios that result from K-nearest neighbors (K-NN) regression of the AM/FM signal dataset's (Section 3.2) three parameters: carrier frequency (f_c), tremolo modulation frequency (f_m) and chirp rate (γ). We performed K-NN regression via the nearest neighbor graphs ($K = 40$) that result from MFCCs, time scattering coefficients, time–frequency scattering coefficients, OpenL3 embeddings and STRFs. We refer the reader to Section 3 for details on the feature extraction hyperparameters.

4. TIME–FREQUENCY SCATTERING 2-D: CONVOLUTIONAL CLASSIFIER

Recent publications have demonstrated time–frequency scattering as a state-of-the-art feature extractor for music classification tasks, including detection of the type of instruments played [7] and detection of playing techniques [15] on solo performances. This is achieved by learning a shallow linear layer over time–frequency scattering coefficients. Time–frequency scattering is yet to be explored as a frontend feature extractor to a deep convolutional neural network (convnet) classifier. Exploiting the 3-D structure of time–frequency scattering ($\lambda_2 = (\alpha, \beta, \theta), \lambda, t$), where the response of joint second-order wavelet filters across time and frequency compose the channels, may enhance contrast in spectrotemporal variations across the time–frequency image. In this section, we seek to compare audio representations as a frontend to a convnet in a task of supervised classification of musical instrument solos.

Eq. (8) describes the output of the first layer of 2-D convolution between the time–frequency scattering image $\mathbf{S}\boldsymbol{x}$ around (λ_2, λ, t) and kernel $\boldsymbol{w}$, where the multiindex variable λ_2 represents the tuple (α, β, θ).

$$Y(\lambda_3, \lambda, t) \sum_{\lambda_2, \delta, \tau} \mathbf{S}\boldsymbol{x}(\lambda_2, \lambda + \delta - 1, t + \tau - 1)\boldsymbol{w}(\lambda_3, \delta, \tau) \quad (8)$$

4.1. Dataset

We perform supervised classification of isolated musical instruments from the Medley-solos-DB dataset [16]. The task of musical instrument recognition has previously been benchmarked with convolutional networks in [17] and joint time–frequency scattering in [7]. Every example in the dataset consists of a fixed 2^{16} discrete-time samples at a sampling rate of 44.1 kHz, corresponding to approximately 3 seconds of audio. Each clip includes the presence of one musical instrument from a highly imbalanced taxonomy of 8 classes: tenor saxophone, trumpet, flute, clarinet, female singer, distorted electric guitar, violin and piano, of 123, 149, 155, 251, 318, 404, 2040 and 2401 training samples, respectively. The training, validation and test subsets consist of a total of 5841, 3494 and 12236 samples, respectively.

4.2. Convolutional Network Feature Design

We parametrize time–frequency scattering such that it yields a 3-D output that is 44×32 along log-frequency and time. To achieve this, we set $J = 13$ octaves for the first and second order temporal wavelet filterbanks. We set $Q = 16$ filters per octave at first order and $Q_2 = 1$ at second order. We perform frequential averaging with the lowpass filter $\boldsymbol{\phi}_F$ over a quarter of an octave with $F = 4$. We set the support of the temporal lowpass filter $\boldsymbol{\phi}_T$ to $T = 2^{11}$, which is applied to an input of $N = 2^{16}$ discrete time samples. The frequential wavelet filterbank has its own set of parameters. Q_{fr} and J_{fr} control the number of wavelets per octave and number of octaves and maximal scattering scale, respectively. We set $Q_{fr} = 1$ and $J_{fr} = 6$.

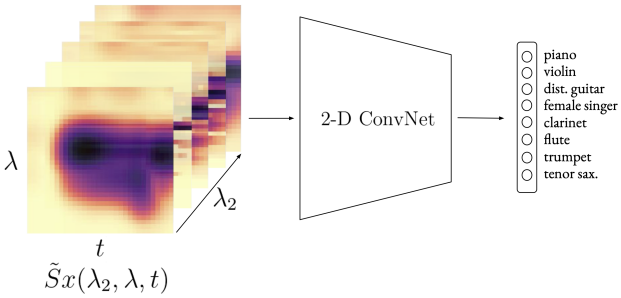

Figure 5: 2-D convnet architecture with a 3-D time–frequency scattering input tensor. The multiindex variable $\lambda_2 = (\alpha, \beta, \theta)$ represents the response of a second-order scattering wavelet at temporal rate α, frequential scale β and spin θ. The 2-D convnet is an EfficientNetB0 architecture with a classification head over 8 classes (see Section 4 for details).

Our implementation supports outputs in various number of dimensions. In the case of 2-D output, the variables correspond to scattering path and time, and first-order and second-order coefficients are concatenated. In contrast, under the "out_3D" mode, we return 2-D and 3-D tensors, for first-order and second-order tensors respectively. Since $\boldsymbol{\phi}_F$ is not applied to first-order coefficients, we learn a convolution filter on the first order output and average pool across log-frequency. We set the convolution filter's kernel size to $(16, 1)$, whose support covers an octave. We apply average pooling to the convolution layer's output, with a kernel size of $(4, 1)$ corresponding to the same support of $\boldsymbol{\phi}_F$ of a quarter of an octave. We concatenate the output of this layer with the 3-D second-order coefficients, resulting in a tensor of size $(195, 45, 32)$ whose indices correspond to log-frequency and log-quefrency unrolled, log-frequency and time. We apply a 2-D batch normalization

	channels	**kernel size**
Conv	195	$(7, 7)$
Batch Norm, ReLU	195	
Max Pool		$(2, 2)$
Conv	390	$(5, 5)$
Batch Norm	390	
SqueezeExcite, ReLU	390	
Conv	780	$(3, 3)$
Batch Norm	780	
SqueezeExcite, ReLU	780	
Global Average Pool		
Linear, ReLU, Dropout 0.5	64	
Linear, Softmax	8	

Table 1: Table outlining the 2-D convnet architecture that process the time–frequency scattering tensor. The channels column indicates the number of channels or hidden units output after a layer.

layer to this tensor, which serves as the input into a 2-D convnet. Table 1 outlines our 2-D convnet that accepts the pre-processed time–frequency scattering tensor as input. We use a *squeeze-and-excitation* layers [18] with a reduction factor of 16.

4.3. Adaptive Logarithmic Compression

Prior to input to the convnet, we apply a transformation to each time–frequency scattering path, that seeks to match a decibel-like perception of loudness. The transformation in Eq. (10), μ–log, consists of mean-based renormalization and a pointwise logarithm. We compute μ across the training set. When ε is a non-learnable constant, we refer to this transformation as μ–log. Previous publications have shown that for music sounds, μ–log transforms each λ_2 such that its histogram of magnitudes is closer to Gaussian [19]. We set ε to the same predefined constant 0.1 per path, which was chosen based on our observations of the skewness of the magnitude histograms. To standardize the input features to the convnet backend, we compute the mean and standard deviation per λ_2 across all $\widetilde{\mathbf{S}}\boldsymbol{x}$ in the training set.

$$\mu(\lambda_2) = \frac{1}{N}\sum_{n=1}^{N}\iint \mathbf{S}\boldsymbol{x_n}(\lambda_2, \lambda, t)\,\mathrm{d}t\,\mathrm{d}\lambda \tag{9}$$

$$\widetilde{\mathbf{S}}\boldsymbol{x}(\lambda_2, \lambda, t) = \log\left(1 + \frac{Sx(\lambda_2, \lambda, t)}{\varepsilon\mu(\lambda_2)}\right) \tag{10}$$

4.4. Baselines

As a performance comparison to our time–frequency scattering hybrid convnet, we train a 2-D convnet on top of the Constant-Q Transform (CQT) and a 1-D convnet on top of time scattering.

We compute the CQT using nnAudio [20] with a hop size of 256 samples, 96 frequency bins, 12 octaves and a minimum centre frequency of 32.7 Hz, and subsequently convert the amplitudes to the decibel scale. We standardize the CQT per bin using the means and standard deviations from the training set. We perform average pooling over frequency and time with a kernel of size $(3, 8)$, yielding a $(32, 32)$ time–frequency image for input into a 2-D convnet. To perform the classification, we use the same 2-D convnet that is outlined in Table 1, however the successive convolution blocks have 64, 128 and 256 channels respectively, the third convolution layer uses max pooling and all max pooling is performed with a kernel size of 2.

Additionally, we extract time scattering (Scattering1D) coefficients. Time scattering is computed similarly by applying only a 1-D temporal wavelet filterbank to the first-order scalogram. To match the setting of JTFS, we set $Q = 16$ filters per octave, $J = 13$ octaves and a temporal lowpass filter support of $T = 2^{11}$. We concatenate first and second order coefficients, yielding a 1423-dimensional vector for each of the 32 time frames. Note that we do not average the time scattering coefficients along first-order log-frequency λ. We also apply the pathwise transformation μ–log across time scattering paths. Time scattering is structured as a vector of scattering paths for each time frame (p, t) where the path multiindex p encompasses both scattering orders. Hence, the integral in Eq. (9) is over the time variable alone. As the convnet classifier, we implement the same convnet as for CQT and JTFS, but with 1-D convolution, batch normalization and pooling operations.

4.5. Training Setup

We train the JTFS based models for 30 epochs and CQT and Scattering1D models for 20 epochs. We use an epoch size of 8192 and batch size of 32, by means of the AdamW optimizer with an initial learning rate of 10^{-3} and weight decay coefficient of 0.1. To set the learning rate of each parameter in the network, we use a cosine annealing schedule with a minimum learning rate of 10^{-11}, and apply warmup by starting at schedule's lowest point. In order to compensate for class imbalance in the Medley-solos-DB dataset, we use a weighted cross entropy (WCE) loss as per Eq. (11), where $N = \{N_1, ..., N_8\}$ is the set of training example supports per class.

$$\mathrm{WCE} = -\sum_{i}^{B}\frac{\max(N)}{N_{k=y_i}}y_i\log(f(x_i)) \tag{11}$$

4.6. Results

In Table 2, we report the classwise and macro-averaged accuracy on the test set. For each run, we use an early stopping procedure that checkpoints the model at each epoch. We select the best checkpoint out the final ten epochs that achieves the higher validation accuracy. We report the mean test set accuracy over three randomly seeded runs.

Time–frequency scattering has previously seen performance of 78% accuracy on the Medley-Solos-DB dataset when used with a shallow linear classifier [7]. Earlier spiral convolutional network architectures achieved a highest average accuracy of 74% [17]. Our results show that the addition of a 2-D convnet backend exceeds the performance of a shallow linear classifier.

JTFS exceeds the average accuracy of CQT and Scattering1D by roughly 23 and 7 percentage points respectively, while attaining the highest accuracy across the majority of musical instrument classes. We found that the unsupervised logarithmic transformation, μ–log, and careful selection of its tunable parameter c were essential factors for improving performance.

JTFS achieves state-of-the-art musical instrument classification performance in the regime of limited annotated data. As a reference, we compute accuracy metrics on the test set using a YAMNet classifier that was pretrained on the very large AudioSet dataset. This achieves 93% accuracy with no additional trained layers. Yet we emphasise a distinction between these tasks; one is trained in

	tenor sax.	trumpet	flute	clarinet	female singer	dist. guitar	violin	piano	avg
CQT	5.7 ± 3.8	**80.9** ± 3.5	40.2 ± 5.3	**85.3** ± 5.1	84.4 ± 0.7	87.8 ± 1.8	64.2 ± 12.8	98.5 ± 1.4	68.4 ± 1.8
Scattering1D	54.8 ± 9.8	70.9 ± 9.7	43.9 ± 8.9	60.1 ± 7.9	93.7 ± 2.4	96.1 ± 1.1	74.1 ± 4.3	98.7 ± 0.2	74.0 ± 2.4
JTFS	**71.5** ± 5.9	77.8 ± 8.7	**57.0** ± 4.3	59.5 ± 1.3	**96.3** ± 0.7	**96.4** ± 0.0	**93.0** ± 5.6	**99.8** ± 0.1	**81.4** ± 0.6

Table 2: Test set classwise and macro average accuracy for 2-D convnet architectures trained for musical instrument classification on Medley-solos-DB. Classes are in ascending order (left to right) of number of training set examples. We report the results of CQT, time scattering and time–frequency scattering frontends for a convnet classifier. See Section 4 for details.

the regime of limited annotated data, while the other has access to millions of annotated examples. The newly introduced implementation has enabled a previously unexplored interaction with learned 2-dimensional deep convolutional networks for supervised classification. We expect this to enable further research interfacing JTFS and convnets for audio analysis and synthesis.

5. TEXTURE SYNTHESIS

Under the context of an audio classification task, feature representations of waveforms benefit from invariance to time-shifting, pitch-shifting and small spectrotemporal deformations. Yet, the introduction of invariance induces inevitably a loss of information. Texture resynthesis from time-shift invariant feature is an effective way to examine what information is preserved and what is lost. Echoing work from [21], [22], and [7], the following section demonstrates the procedure of texture resynthesis, illustrates the texture preservation qualities of time-shift invariant JTFS, and reports the speed improvement of texture resynthesis with GPU-enabled JTFS-GPU over MATLAB toolbox `scattering.m`.

Starting from an audio signal $\boldsymbol{x}(t)$, we first calculate its scattering coefficients $\mathbf{S}\boldsymbol{x}$. To reconstruct the signal, we initialize a trial signal $\boldsymbol{y}(t)$ with random noise, and use backpropagation to update $\boldsymbol{y}$ at each iteration, such that the normalized error $E(\boldsymbol{y_n}) = \|\mathbf{S}\boldsymbol{x} - \mathbf{S}\boldsymbol{y_n}\|/\|\mathbf{S}\boldsymbol{x}\|$ is progressively reduced,

$$\boldsymbol{y}_{n+1}(t) = \boldsymbol{y}_n(t) + \mu \boldsymbol{\nabla} E(\boldsymbol{y_n}). \tag{12}$$

While the loss function $E(\boldsymbol{y_n})$ is nonconvex and may have local minimizers, our goal is not to mathematically invert the forward scattering operations, but to approximate a sonification of scattering coefficients $\mathbf{S}\boldsymbol{x}$. The gradient $\boldsymbol{\nabla} E(\boldsymbol{y_n})$ is computed via reverse-ordered Hermitian adjoints of the forward scattering operations, detailed in [22]. We adopt the PyTorch backend for gradient computation, as well as a bold driver heuristic to update adaptively the learning rate μ. The bold driver heuristic increases the learning rate by a constant factor if the loss decreases and vice versa [23]. The reconstruction error decreases progressively: it reaches a normalized loss of about 10% after 20 iterations and about 3% after 100 iterations.

In order to illustrate the information that is preserved and lost in time-shift invariant JTFS, we experiment on the two bird chirps in Section 2 (see Fig. 1 (c) and (d)) and compare their resynthesis results with those of second-order time scattering coefficients. The temporal support of time-shift invariance T is chosen to be 370 ms, which is of the order of three bird calls in example (f) and thus relevant in a recognition task. We fix the scattering scale $J = 12$ and filters per octave $Q = 12$ in all our experiments. As shown in Fig. 6, both JTFS and time scattering preserve well the spacing along frequency axis and the amplitude modulation trend in each frequency bands. Meanwhile, both lose the precise temporal location of discrete bird call events because of temporal averaging. However, a clear distinction between them can be observed in terms of the frequency band alignment in time. Unlike time scattering, JTFS manages to recover synchronicity over frequency subbands.

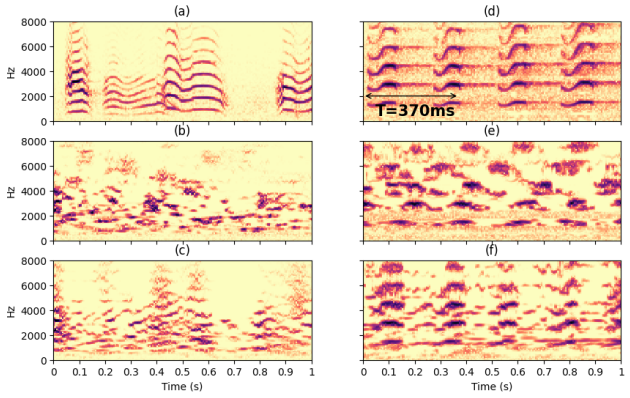

Figure 6: Reconstructed birdcalls from time-shift invariant scattering coefficients. (a) and (d) are scalograms of the original audio of two distinct birdcalls. (c) and (f) are the corresponding reconstructed scalograms from joint time–frequency scattering coefficients. (b) and (e) from second-order time scattering coefficients. All of the coefficients are computed with $J = 12$, $Q = 12$, enforcing time-shift invariance with a temporal lowpass filter of support $T = 2^{13}$ samples (370 ms).

To compare the computational speed, we perform texture resynthesis on an audio segment of size $N = 2^{16}$ samples, i.e., around three seconds; both with JTFS-GPU and `scattering.m`. We record the time elapsed during each iteration of backpropagation. GPU computing accelerates the resynthesis procedure by close to ten times relative to `scattering.m`, with an average of 720 milliseconds per iteration.

6. CONCLUSION

Deriving auditory representations that act as proxies for perceptual similarity is an essential step for the enhancement of generative audio models and digital audio effects. In this paper, we have highlighted the need for a scalable computational model of spectrotemporal receptive fields (STRF) of the auditory cortex. We have provided *scale–rate* visualizations of time–frequency scattering, analogous to those used in auditory perception research. By

means of practical examples, we have introduced a differentiable implementation of time-frequency scattering that is compatible with modern deep learning frameworks. Through manifold embedding visualizations and parameter recovery, we showed that time–frequency scattering can adequately serve as a representation of similarity for AM/FM signals. By using our implementation's 3-D time–frequency scattering output as a frontend feature extractor for a 2-D convolutional neural networks classifier, we have exceeded previous state-of-the art benchmarks on the task of supervised classification of musical instrument solos with limited annotations. Finally, we have demonstrated resynthesis of a variety of texture signals via their scattering coefficients, benefiting from a $10\times$ speedup over previous benchmarks.

7. ACKNOWLEDGMENT

We thank the DAFx 2022 organizing committee for their help.

8. REFERENCES

[1] Daniel Pressnitzer and Stephen McAdams, "Acoustics, psychoacoustics and spectral music," *Contemporary Music Review*, vol. 19, no. 2, pp. 33–59, 2000.

[2] Vincent Lostanlen, Christian El-Hajj, Mathias Rossignol, Grégoire Lafay, Joakim Andén, and Mathieu Lagrange, "Time–frequency scattering accurately models auditory similarities between instrumental playing techniques," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2021, no. 1, pp. 1–21, 2021.

[3] Etienne Thoret, Baptiste Caramiaux, Philippe Depalle, and Stephen Mcadams, "Learning metrics on spectrotemporal modulations reveals the perception of musical instrument timbre," *Nature Human Behaviour*, vol. 5, no. 3, pp. 369–377, 2021.

[4] Didier A Depireux, Jonathan Z Simon, David J Klein, and Shihab A Shamma, "Spectro-temporal response field characterization with dynamic ripples in ferret primary auditory cortex," *Journal of neurophysiology*, 2001.

[5] Kailash Patil, Daniel Pressnitzer, Shihab Shamma, and Mounya Elhilali, "Music in our ears: The biological bases of musical timbre perception," *PLoS computational biology*, vol. 8, no. 11, pp. e1002759, 2012.

[6] Taishih Chi, Powen Ru, and Shihab A Shamma, "Multiresolution spectrotemporal analysis of complex sounds," *The Journal of the Acoustical Society of America*, vol. 118, no. 2, pp. 887–906, 2005.

[7] Joakim Andén, Vincent Lostanlen, and Stéphane Mallat, "Joint time–frequency scattering," *IEEE Transactions on Signal Processing*, vol. 67, no. 14, pp. 3704–3718, 2019.

[8] Mathieu Andreux, Tomás Angles, Georgios Exarchakis, Roberto Leonarduzzi, Gaspar Rochette, Louis Thiry, John Zarka, Stéphane Mallat, Joakim Andén, Eugene Belilovsky, et al., "Kymatio: Scattering transforms in Python.," *Journal of Machine Learning Research*, vol. 21, no. 60, pp. 1–6, 2020.

[9] Shanel Gauthier, Benjamin Thérien, Laurent Alsène-Racicot, Muawiz Chaudhary, Irina Rish, Eugene Belilovsky, Michael Eickenberg, and Guy Wolf, "Parametric scattering networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5749–5758.

[10] Edouard Oyallon, Sergey Zagoruyko, Gabriel Huang, Nikos Komodakis, Simon Lacoste-Julien, Matthew Blaschko, and Eugene Belilovsky, "Scattering networks for hybrid representation learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 9, pp. 2208–2221, 2018.

[11] Changhong Wang, Emmanouil Benetos, Shuge Wang, and Elisabetta Versace, "Joint scattering for automatic chick call recognition," *arXiv preprint arXiv:2110.03965*, 2021.

[12] Jason Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello, "Look, listen, and learn more: Design choices for deep audio embeddings," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 3852–3856.

[13] Joshua B Tenenbaum, Vin de Silva, and John C Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[14] Brian McFee, V Lostanlen, A Metsai, M McVicar, S Balke, C Thomé, C Raffel, F Zalkow, A Malek, K Lee, et al., "librosa/librosa: 0.8. 0," *Version 0.8. 0, Zenodo, doi*, vol. 10, 2020.

[15] Changhong Wang, Vincent Lostanlen, Emmanouil Benetos, and Elaine Chew, "Playing technique recognition by joint time–frequency scattering," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 881–885.

[16] Vincent Lostanlen, Rachel M. Bittner, and Slim Essid, "Medley-solos-db: a cross-collection dataset of solo musical phrases," Aug. 2018.

[17] Vincent Lostanlen and Carmine-Emanuele Cella, "Deep convolutional networks on the pitch spiral for musical instrument recognition," 2016.

[18] Jie Hu, Li Shen, and Gang Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[19] Vincent Lostanlen, Joakim Andén, and Mathieu Lagrange, "Extended playing techniques: the next milestone in musical instrument recognition," in *Proceedings of the 5th International Conference on Digital Libraries for Musicology*, 2018, pp. 1–10.

[20] Kin Wai Cheuk, Hans Anderson, Kat Agres, and Dorien Herremans, "nnAudio: An on-the-fly GPU audio to spectrogram conversion toolbox using 1d convolutional neural networks," *IEEE Access*, vol. 8, pp. 161981–162003, 2020.

[21] Joakim Anden, Vincent Lostanlen, and Stephane Mallat, "Joint time-frequency scattering for audio classification," *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, Sep 2015.

[22] Vincent Lostanlen and Florian Hecker, "The shape of remixxxes to come: Audio texture synthesis with time-frequency scattering," in *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 06 2019.

[23] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2013, pp. 1139–1147.