2015 IEEE 15th International Conference on Data Mining Workshops

Scattering decomposition for massive signal classification : from theory to fast algorithm and implementation with validation on international bioacoustic benchmark

Randall Balestriero
Pierre et Marie Curie University
Paris, France
Email: randallbalestriero@gmail.com

Hervé Glotin
UTLN, IUF, CNRS
Toulon, France
Email: h.glotin@gmail.com

***Abstract*—With the computational power available today, machine learning is becoming a very active field finding its applications in our everyday life. One of its biggest challenge is the classification task involving data representation (the preprocessing part in a machine learning algorithm). In fact, classification of linearly separable data can be easily done. The aim of the preprocessing part is to obtain well represented data by mapping raw data into a "feature space" where simple classifiers can be used efficiently. For example, almost everything around audio/bioacoustic uses MFCC features until now. We present here a toolbox giving the basic tools for audio representation using the C++ programming language by providing an implementation of the Scattering Network [2] which brings a new and powerful solution for these tasks. We focused our implementation to massive dataset and servers applications. The toolkit of reference in scattering analysis is SCATNET from Mallat et al. [1]. This tool is an attempt to have some of the scatnet features more tractable for Big Data challenges. Furthermore, the use of this toolbox is not limited to machine learning preprocessing. It can also be used for more advanced biological analysis such as animal communication behaviours analysis or any biological study related to signal analysis. This implementation gives out of the box executables that can be used by simple commands without a graphical interface and is thus suited for server applications. As we will review in the next part, we will need to perform data manipulation on huge dataset. It becomes important to have fast and efficient implementations in order to deal with this new "Big Data" era.**

I. Fast Scattering

A. Introduction

The Scattering Network aims to find a better time-invariant data representation after numerous transformations of an input x. It's been developed by Stéphane MALLAT and his team and the only available implementation is in Matlab which provides high-level programming but is not optimal in term of execution time. In fact, this algorithm just started to be applied in concrete challenges and already finds its limitations in the required time to complete the transformation on massive dataset. We will review its core ideas and the algorithmic improvements that can be done. Finally our implementation will be done in C++ for final uses. We will focus our analysis on the $1D$ case.

The transform is obtained through series of linear and non linear operations on x. The linear operations are done through the convolutions while the non linearities come from the use of the absolute value on these convolutions. The use of the latter allows fast convergence by the contractive property. The first convolutions $x \star \psi_{\lambda_1}$ are decomposing the signal into a wavelet basis. A parallel can be made with the FFT and the complex sine decomposition. We then take the absolute value and define $U_1 := |x \star \psi_{\lambda_1}|$ since we are interested in the energy response of the filter. In addition, the application of the low-pass filter (father wavelet) ϕ brings time invariance (of length depending on the ϕ support/bandwidth) for the first scattering coefficients $S_1 := x \star \phi$.

The structure itself of the network can be compared to a Convolutionnal Neural Network where the filters are given and fully determined by the meta parameters and not learned during a training phase. This is an important difference in term of computation time allowing good representation without training. We have to keep in mind that filters generation is also complex and time consuming.

The mapped data into the feature space can be used for simple data analysis or data learning but it finds its best use in classification. In fact, this feature space is much more suited for the use of linear classifier. Note that in this implementation we won't look at the reconstruction problems since our main goal is not to use the Scattering Network for compression or reconstruction. Let's look at the general picture of the scattering network and analyse it briefly.

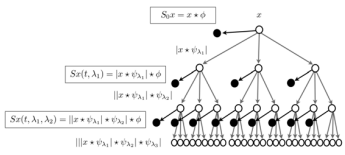


Fig. 1. Scattering Network Architecture[4]

In this case, the scattering network is made of 3 layers.

[1] http://www.di.ens.fr/data/software/scatnet/

978-1-4673-8493-3/15 $31.00 © 2015 IEEE
DOI 10.1109/ICDMW.2015.127

Each layer has a specific low-pass filter ($\phi_i, i = 1, 2, 3$) and high-pass filter banks ($\psi_{\lambda_1}, \psi_{\lambda_2}, \psi_{\lambda_3}$). In our specific case of $1D$ signals, there is only one ϕ per layer (only one orientation). We can thus associate a wavelet dictionary for each level i by $\{\phi_i, \psi_{\lambda_i}\}$. Given an input signal x of size N we perform the low-pass decomposition S_1 and a serie of high-decomposition $x \star \psi_{\lambda_1}$. Note that each convolution is then sub-sampled in the time dimension leading to different output sizes for each vector composing the resulting projection. The sub-sampling only reduces coefficient redundancy and depends on the filters parameters.
Finally on these last convolutions is applied the absolute value operation leading to the high-decomposition $U_1(t, \lambda_1)$ of the first scattering layer (which is a scalogram).
For the second layer, each one of the previous high-decomposition result (row of the scalogram) is treated as an input signal and the same algorithm is performed again. Details about this will be given in the scattering layer sectionII-C but we can already note that the meta parameters are specific to each scattering layer. Finally let's review what the meta parameters are about :

- T determines the ϕ support/bandwidth, this is basically the time invariant window. Taking a great T leads to huge time invariance but at the cost of variance (and information) loss with respect to the time dimension. We have the relation $T = 2^J$
- Q determines the quality factor (the number of filters per octave)
- J determines the number of octave to go through in the high-pass decomposition.
- PE (Periodization Extent) is the constant used in the filter periodization (1 by default)

B. Filter Bank

Filters are created given the meta-parameters, and the signal length. Using this, the coefficients σ (bandwidth) and ξ (frequency center) are deduced.

The filters are directly computed in the Fourier domain to speed up the decomposition algorithm, indeed we only have to compute the FFT of the input followed by inplace multiplications and the IFFT to perform the decomposition algorithm. Here ξ corresponds to the central frequency and so it is the global maximum position. It can be seen as a position parameter while σ is a scale parameter. In practice,from a mother wavelet, we derive all the wavelets (filters) with the following formula (l being the scale/dilation and x' being the translation) :

$$\psi_{l,x'} = l^{-1/2}\psi(\frac{x - x'}{l}), l \in \mathbb{R}^+, x' \in \mathbb{R} \quad (1)$$

a) Filters: In this implementation, high-pass filters are Morlet wavelets while low-pass filters are Gabor filters. The Morlet wavelet is a plane wave (sinus) modulated with a Gaussian :

$$\psi(x; \omega_0, \sigma_0) = \exp^{i\omega_0 x} \exp^{\frac{-x^2}{2\sigma_0^2}} \quad (2)$$

Or in the Fourier domain :

$$\hat{\psi}(\omega; \omega_0, \sigma_0) = H(\omega) exp^{-\frac{(\omega-\omega_0)^2}{2\sigma_0^2}} \quad (3)$$

Note that Morlet filters are actually another name for Gabor kernels. The difference between the Gabor function (non-zero-mean function) and the Gabor kernel (zero-mean function) is that the Gabor kernel satisfies the admissibility condition for wavelets (integral equals to 0), thus being suited for multi-resolution analysis. The admissibility condition ensures that the inverse transform and Parseval formula are applicable.

C. Scattering Layer

b) Decomposition Implementation: The core of the algorithm lies in this decomposition. Firstly, the convolutions defined in the section I-A are redundant and so a sub-sampling is applied so that following operations are applied with less computation time. This implies a reduced output length and faster overall decomposition. Note that the sub-sampling will actually be done in the Fourier domain. In fact, we will periodize the result in the frequency domain in order to sub-sample in the time domain to obtain the desired output size.

II. Scattering Network Fast Implementation

A. Meta-Parameters

In order to respect the scattering network architecture, this toolbox uses a specific structure : MetaParam using default parameters and a TtoJ method. Each layer has a set of parameters which fully define the associated filter bank. Let's now see the details of each implementation level and emphasize the implementation architecture used.

B. Filter Bank Creation

Filters are created through the constructor of the Filter1D class. Given meta-parameters and a support size, the constructor will initialize all the wanted variables and compute the actual filters. Note that the Filter1D class has two children : the MorletFilter1D and GaborFilter1D. These two specializations have their own filter generation algorithm. This also means that if one wants to implement a new filter, the only thing to do is to create another class of the name of this filter, inherit from the Filter1D class and implement the coefficients generation method.
Note that the constructor can be used in two different ways :

- Giving support size, meta parameters, and the position of the filter in this configuration
- Giving a support size, σ and ξ.

Let's now look at the high-pass filter bank generated for a scattering layer :

c) Sparse Coefficients: In addition, we know that the filters have finite and small support relative to our domain (by definition of wavelets), thus we will compute them only on their support. In addition of reducing computation time and memory usage for the filter generation, the decomposition algorithm (convolutions/inplace multiplications) will be faster.

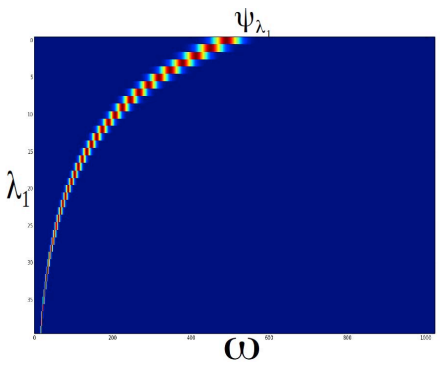


Fig. 2. Filters with support of 1024 bins (in Fourier domain), 8 wavelets per octave through 5 octaves, with constant normalization

We thus define

$$supp(f, \epsilon) = [a, b] := I \iff \forall x \notin I, |f(x)| < \epsilon \quad (4)$$

We will now compute the analytic formula to find the support of the Morlet filters given the mean and standard deviation :

$$supp(\psi(.; \hat{\omega_0}, \sigma)) = [\max\left(\omega_0 - \sigma\sqrt{-\log(\epsilon)2}, 0\right), \omega_0 + \sigma\sqrt{-\log(\epsilon)2}] \quad (5)$$

Proof:

$$\begin{aligned}
\hat{\psi}(\omega; \omega_0, \sigma) := 1_{\{\omega>=0\}} e^{\frac{-(\omega-\omega_0)^2}{2\sigma^2}} &> \epsilon \\
-\frac{(\omega-\omega_0)^2}{2\sigma^2} &> \log(\epsilon), \omega \geq 0 \\
-(\omega-\omega_0)^2 &> \log(\epsilon)2\sigma^2 \\
-\omega^2 + 2\omega\omega_0 - \omega_0^2 &> \log(\epsilon)2\sigma^2 \\
-\omega^2 + 2\omega\omega_0 - (\omega_0^2 + \log(\epsilon)2\sigma^2) &> 0 \\
-\omega^2 + 2\omega\omega_0 - (\omega_0^2 + \log(\epsilon)2\sigma^2) &= 0 \\
\implies \Delta = 4\omega_0^2 - 4(\omega_0^2 + \log(\epsilon)2\sigma^2) \\
\implies \Delta = -4\log(\epsilon)2\sigma^2 \\
\implies \omega_{a,b} = \frac{-2\omega_0 \pm \sqrt{\Delta}}{-2} \\
\implies a = \omega_0 - \sigma\sqrt{-\log(\epsilon)2} \\
\implies b = \omega_0 + \sigma\sqrt{-\log(\epsilon)2}
\end{aligned}$$

■

We require $\epsilon \in [0, 1]$ thus the quantity $\Delta \in \mathbb{R}^+$ is positive since $(\log(\epsilon) < 0)$. This condition is natural since we want to disregard negligible coefficients only.
In addition, one can notice that reducing the effective support (removing small (< 1) but non-zero coefficients) doesn't mean loosing information. In fact, as long as the whole frequency spectrum is covered by the filter bank, there is no reconstruction error. A sufficient condition to guarantee reconstruction is given by :

$$supp(\psi_{i+1}) \cap supp(\psi_i) \neq \emptyset, \forall i$$

The difference will actually be in the representation which will become less redundant from scale to scale in the case of almost non overlapping filters.

Let's look at the computation time of the filters when applying the formula only on the effective support of the filters.

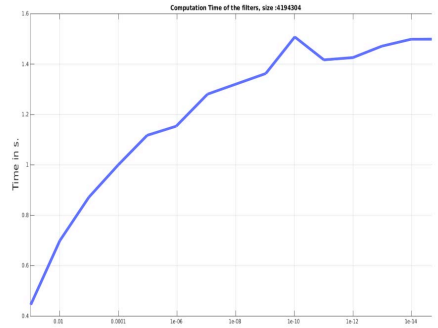


Fig. 3. Computation Time of the filters with different threshold values with 8 filters per octave on 6 octave

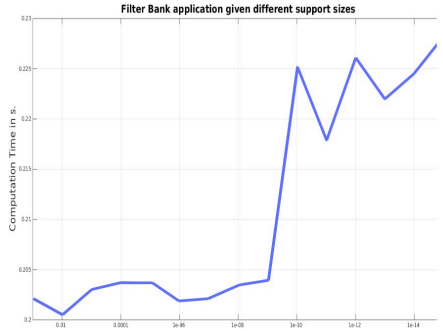


Fig. 4. Computation Time of the decomposition with different threshold values with 8 filters per octave on 6 octave

One can now see (Fig.5) the generated filter bank with finite support.

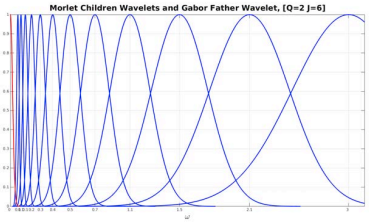


Fig. 5. Morlet Wavelet with the ϕ filter in red

C. Layer Implementation

The role of this class is to be the link between the input x, the meta parameters, and the filter banks by performing the decomposition process. Firstly, this class takes a $2D$ input (the input signal x has to be transformed for the first layer). This allows an easy link between layers by directly setting the input of the next one as the output of the previous one.

Given the input, private variables are computed determining the structure of the class by computing variables that will be passed to the next layer such as the size of the output (given the input size and the number of ψ filters : $Q \times J$). After this, the filters are generated using the Filter1D class and the sparse approach. The decomposition can now be performed.
Note that the decomposition is stored as a $2D$ matrix for every layer. Normally, layer i has a dimension of $i+1$ which is not true in term of memory management. In fact, in this toolbox, the graph structure of the scattering network (I-A) has been kept through $2D$ matrices. For example for the second layer if we have $|\lambda_1|$ filters for the first layer and $|\lambda_2|$ filters for the second layer, the U_2 matrix will be :

$$U_2 := \begin{pmatrix} ||x \star \psi_{\lambda_1=1}| \star \psi_{\lambda_2=1}| \\ ||x \star \psi_{\lambda_1=2}| \star \psi_{\lambda_2=1}| \\ \cdots \\ ||x \star \psi_{\lambda_1=1}| \star \psi_{\lambda_2=2}| \\ ||x \star \psi_{\lambda_1=2}| \star \psi_{\lambda_2=2}| \\ \cdots \end{pmatrix}$$

D. Optimizations and Computation Time

1) Filter Bank Optimizations: Since we want to use this implementation on massive dataset, it is mandatory to optimize the implementation.

The best way to improve the required time to perform the decomposition is around the filters (allocation, sparse creation, application). In fact, FFT/IFFT and the periodization algorithms already use efficient and optimal algorithms.

By only keeping the support of ψ we can improve the creation process but more importantly when we apply the operation $\hat{x}. \times \hat{\psi}_{\lambda_1}$ we only need to perform it on the support and set all the other coefficients to 0. Doing this is really efficient by the natural sparsity of the filters as shown before (2).
Now that we only save part of the coefficients, the dimension is reduced and it is even possible to optimize again the filters creation. We know that filters are completely determined by the size of the input N and the meta parameters Q and J. So in other word, it is possible to create filter bank dictionaries and save them in .txt files thus when we need to decompose a signal, we only need to load them and not explicitly compute them every time. This is particularly practical since the input size should be be a power of 2 to be the most efficient (true $O(n\log(n))$ complexity). Thus, the number of different combination of parameters is tractable.
By doing so, when we treat for example thousands of files, we will only load a few filter banks prior to the decomposition. Now, if we use the implementation in a loop over several signals, it is even possible to first load the filters and then perform all the decomposition without having to load filters every time.
Note however that this optimization is not always doable. In fact, the use of this toolbox on other platform/servers or just because the user doesn't want IO operations for example, requires to actually compute the filters every time. In this case, the generating algorithm has been optimized using standard C++ optimization techniques such as loop-unrolling, inplace methods,...

2) Computation Time: Using all the available optimization without the use of the filter bank IO (thus generating the filters every time) this toolbox needs about 20% of the signal duration (using one CPU of 2.5 GHz, in the case of a frequency sampling of 96 kHz) to perform the loading of the wav, scattering decomposition and the saving of the scattering coefficients into a .txt files). This required time obviously varies with the meta-parameters, the ones used here are generic (also used for the classification challenge presented later). Note that using the filter bank IO will lead to better performance on huge dataset by not recomputing the filters for every signal. The Scatnet implementation available in Matlab requires almost 400% of the signal time to perform the decomposition algorithm. This can be further improved by using standard meta-programming techniques in C++ and better optimization schemes.
Note that this toolbox is now used as the reference implementation for the scattering decomposition on ENS and UTLN servers for massive dataset decomposition (such as bioacoustic challenges).

III. Benchmark and Bioacoustic Classification

A. Challenge Description

The task is focused on bird identification based on different types of audio records over 999 species from South America centered on Brazil. Additional information includes contextual meta-data (author, date, locality name, comment, quality rates). The main originality of this dataset is that it was built through a citizen sciences initiative conducted by Xeno-canto, an international social network of amateur and expert ornithologists. This makes the task closer to the conditions of a real-world application:

- audio records of the same species are coming from distinct birds living in distinct areas
- audio records by different users that might not used the same combination of microphones and portable recorders
- audio records are taken at different periods in the year and different hours of a day involving different background noise (other bird species, insect chirping, etc)

The data is built from the outstanding Xeno-canto collaborative database (http://www.xeno-canto.org/) involving at the time of writing more than 192k audio records covering 9120 bird species observed all around the world thanks to the active work of more than 2020 contributors.

The subset of Xeno-canto data that will be used for the task will contain 33203 audio recordings belonging to the 999 bird species having the more recordings in the union of Brazil, Colombia, Venezuela, Guyana, Suriname and French Guiana Xeno-canto recordings. The amount of 999 classes will clearly go one step further previous benchmarks (80 species max) and foster brave new techniques. On the other side, the task will remain feasible with current approaches in terms of the number of records per species and the required hardware to process that data. Detailed statistics are the following:

- minimally 14 recordings per species (maximum $>$ 200)

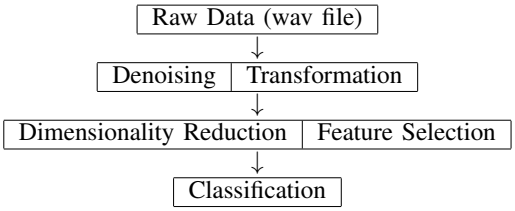
- minimally 10 different recordists, maximally > 40, per species.

Note that this year's challenge is two times bigger (in term of different classes and dataset size) than last year's challenge. The best submission last year reached a MAP of 35%.

This Challenge is organized by Pr. Glotin, Pr. Joly and Xeno Canto since 2014 following Challenges launched by Pr. Glotin and al. for NIPS4B 2013, ICML4B 2013 which were under the massive dataset project MI CNRS MASTODONS in the Scaled Acoustic Biodiversity project. SABIOD.ORG is organized by Pr. Glotin (UTLN) now with the collaboration of LifeClef (Clef Challenge).

B. Problem Approach

The goal of any classification task is to find a way to learn (not necessarily in a supervised way) the underlying structures of the data in order to then generalize it for classification of unseen examples. In our case, we start with a set (the training set) made of raw signals (vectors) and corresponding labels (scalars): $(\mathbf{x}, \mathbf{y})$.
Our goal is then to find a classification process f from the input space to the set of labels $\mathbf{X} \rightarrow \mathbf{Y}$ such that when given a new unseen example x, $f(x)$ predicts its most likely belonging class. This process is usually a multi-step algorithm, first mapping data into new feature spaces where we can then more easily apply a classification algorithm. A simple approach is to first find a new (more informative) representation of the signals into this new feature space. By doing so we wish to gain discriminative information while reducing the non informative variance (through denoising for example). In the end we want to have mapped the data into a new space where examples from the same class will be similarly represented and data from different classes will be easily separable.



a) Input: The input in this task is the raw wav file recorded with a sampling frequency of 48 kHz. Recording devices are not necessarily the same and signals structures are different (background noises, geographic locality, recorder quality ,...). The lengths of the signals are also really unequal (from 0.1 second to 45 min) and the ratio of the bird presence over the signal length is inconsistent from signal to signal.

b) Preprocessing: The key part of our approach concerns the preprocessing part and more specifically the scattering network architecture and the impact of its coefficients. In addition, denoising will be part of it so our whole preprocessing part will be made of the scattering network.

c) Feature Selection: By the use of the scattering network, the features (scattering coefficients) will be aggregated over the time dimension using different simple statistics.This way, we bring global time invariance and have features of same length whatever the input signal length was. Note that it is now possible to analyse each vector by a "vertical" component with the Joint-Scattering that has just been developed by the ENS team.

d) Prediction: Finally the classification will be made using a two-layer perceptron (with a multinomial logistic regression for class prediction as the last layer). This classifier will only be optimized through the number of hidden nodes for the hidden layer.
Note that other approaches are possible, this could be made with SVM, random forests,...

e) Sub-training set: This is obviously not possible to test algorithms on the whole training set. The common approach is to create a sub-training set where tests are made. This puts a condition : this set has to be representative and has to include all the difficulties and most of the special cases of the task. This has been done for example for image classification where algorithms are trained on CIFAR10 to then be generalized to the CIFAR100 challenge. We will use a sub-training set of 45 classes each one having 15 examples.

f) Comments: The use of an Artificial neural Network means that lots of fine-tuning can be done (L1/L2 norm penalty,learning rate,batch size,...) but our focus here remains on the preprocessing part and how can we find a good mapping for a bioacoustic dataset.

IV. Classification Algorithms using the Scattering Network

A. Second Scattering Coefficients

1) Architecture: This first model is based on the second scattering coefficients. The time dimension will be suppressed by aggregating the information using the mean, max and standard deviation of the coefficients.
The scheme is as follow :

$$\begin{array}{c} \text{Raw Signal Input } x(t) \in \mathbb{R}^N \\ \text{normalization } \rightarrow x(t) \in [-1,1], 0 \leq t \leq N \\ \downarrow \\ |x \star \psi_{\lambda_1}| := U_1(t, \lambda_1) \\ \downarrow \\ U_1 \star \phi_2 := S_2(t, \lambda_1) \end{array} \tag{6}$$

Where we have :

$$U_1(t, \lambda_1) = \begin{pmatrix} |x \star \psi_{\lambda_1=1}| \\ |x \star \psi_{\lambda_1=2}| \\ \cdots \end{pmatrix} \in \mathbb{R}^{|\lambda_1| \times N_1}$$

$$S_2(t, \lambda_1) = \begin{pmatrix} |x \star \psi_{\lambda_1=1}| \star \phi_2 \\ |x \star \psi_{\lambda_1=2}| \star \phi_2 \\ \cdots \end{pmatrix} \in \mathbb{R}^{|\lambda_1| \times N_2}$$

Then we aggregate the time component using basic descriptive statistics (mean, standard deviation, max) leading to the actual feature vector which is a concatenation of the following basic statistics :

$$\left(\overline{S_2}\right) := \left(\frac{\sum_{t=1}^{N_2} S_2(t, \lambda_1)}{N_2} \right)_{\lambda_1=1}^{|\lambda_1|} \in \mathbb{R}^{|\lambda_1|}$$

$$(\sigma(S_2)) := \left(\sqrt{\frac{\sum_{t=1}^{N_2} \left(S_2(t,\lambda_1) - \overline{S_2(t,\lambda_1)} \right)^2}{N_2}} \right)_{\lambda_1=1}^{|\lambda_1|} \in \mathbb{R}^{|\lambda_1|}$$

$$\left(\max_t(S_2) \right) := \left(\max_t(S_2(t,\lambda_1)) \right)_{\lambda_1=1}^{|\lambda_1|} \in \mathbb{R}^{|\lambda_1|}$$

This choice comes from the fact that the time component contains informations about the interesting (and most likely discriminative) behaviours of the bird songs. However, the location in time of these features is not at all required to classify a signal. This is why time-invariance is such an important property.
The degrees of freedom of the model are (T_2, Q_1, J_1, h_1) with h_1 being the number of hidden neurons in the classifier. The parameters Q_1 and J_1 are chosen a priori knowing the communication behaviours of birds ($Q_1 = 8$, $J_1 = 5$). Thus, the parameters T_2 and h_1 will have to be selected by optimizing the classification accuracy. A common method to select a coefficient is cross-validation. We basically compute the accuracy of the trained model for each value and then keep the best one. Obviously, the values to be tested can be a priori reduced or influenced using some probabilistic methods for example.

2) Cross-validation : T: The parameter T_2 has to be cross-validated since it is not easy a priori to determine the amount of time invariance we want to introduce. In fact, having a large T_2 can reduce the variance but this can hide some discriminative informations (the structure of fast bird calls for example) and so can make two different birds have the same scattering representation. On the contrary, having a too small T_2 will leave us with a huge variance which will be mainly made of noise (thus non informative). Having the chance to be in a supervised learning task, applying cross-validation on this parameter is the best way to maximize our prediction accuracy.

3) Cross-validation : Features: Not only the representation has to be tuned but the features computed from it have to be chosen too. At a first shot we might say that the most simple (but informative) statistics are the mean, maximum and standard deviation. More complicated features could be computed from the scattering coefficients such as the Discrete Cosine Transform, allowing time invariance while keeping informations on the structure of the scattering coefficients for example.

Even with these 3 simple statistics, we can see (Fig. 6, 7, 8) that the features can be correlated. Let's look at a plot of each statistics for the first 25 classes. Each column is a feature vector made of 80 coefficients (the statistics for the 80 lines of the scattering). Since we have 15 examples by class we end up with a matrix feature of size 80×15 for each class.

We can now see in Fig. 6, Fig. 7 and Fig. 8 the different statistics for all the sub-training set.

The standard deviation features provide the best information, we will then keep this configuration meaning that with only 80 coefficients we are able to reach the best accuracy for the given architecture.

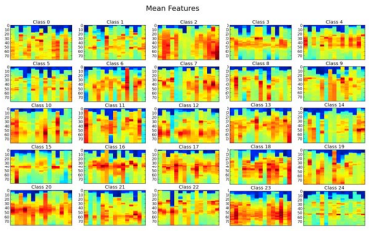


Fig. 6. The 80 mean features for the first 25 classes.

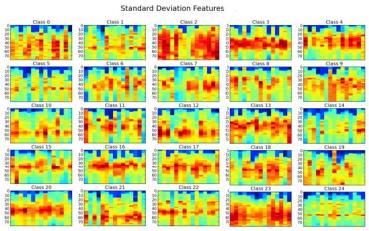


Fig. 7. The 80 standard deviation features for the first 25 classes.

4) Cross-validation : h_1: Finally, the classifier itself needs some fine-tuning. The main parameter is the number of hidden neurons for the hidden layer of our two-layer perceptron. We now have our (close to the) best configuration from the scattering network parameters to the classifier using 150 neurons. Note that the learning rate, the momentum, the L1/L2 norms conditions are left with the default "rule-of-thumb" values.

5) Results Summary and Discussions: After every step of the cross-validation process we end up with a classification accuracy of 32% for 45 classes and with a feature vector of size 80. It is clear that we lack information about the structure of the scattering coefficients. With the standard deviation we don't have access to information such as the repartition of the coefficients (rithmicity), the transition probabilities,... One way to capture more information without having to use more complicated statistics is to use a second scattering layer which will actually capture some of this missing information by its underlying (more complex) structure.

B. Third Scattering Coefficients

1) Architecture: The architecture now is an extension of the last one. We compute a first layer which is then used to compute a second scattering layer to then apply the features computation and the classification algorithm.

We have then U_1 and S_2 as previously defined plus :

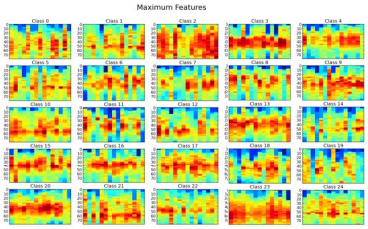


Fig. 8. The 80 maximum features for the first 25 classes.

$$U_2(t,\lambda_1,\lambda_2) = \begin{pmatrix} ||x \star \psi_{\lambda_1=1}| \star \psi_{\lambda_2=1}| \\ ||x \star \psi_{\lambda_1=2}| \star \psi_{\lambda_2=1}| \\ \dots \\ ||x \star \psi_{\lambda_1=1}| \star \psi_{\lambda_2=2}| \\ ||x \star \psi_{\lambda_1=2}| \star \psi_{\lambda_2=2}| \\ \dots \end{pmatrix} \in \mathbb{R}^{|\lambda_1|*|\lambda_2|\times N_3}$$

$$S_3(t,\lambda_1,\lambda_2) = \begin{pmatrix} U_2(t,1,1) \star \phi_3 \\ U_2(t,2,1) \star \phi_3 \\ \dots \\ U_2(t,1,2) \star \phi_3 \\ U_2(t,2,2) \star \phi_3 \\ \dots \end{pmatrix} \in \mathbb{R}^{|\lambda_1|*|\lambda_2|\times N_4}$$

We now compute the statistics on S_3 leading to a feature vector of greater dimension. We will also explore the case where we concatenate the feature of S_2 and S_3.
The degrees of freedom are now : Q_1, J_1,T_2, Q_2, J_2, T_3 and h_1 with h_1 being the number of hidden neurons in the classifier. The parameters T_2, Q_1, J_1 are chosen to be the optimized coefficients of the one-layer case. The parameters Q_2 and J_2 are chosen a priori $(2, 8)$ thus we are left with T_3 and h_1. In fact the number of hidden neurons has to be optimized again since the input features won't be of the same structure (dimension and underlying relations). Once again these parameters will be optimized through cross-validation.

2) Cross-validation : T_3: The first parameter to cross-validate is the T_3 coefficient which defines the ϕ_3 and (Q_2, J_2) for ψ_{λ_2} filter bank. A too small T_3 makes the discriminative information non relevant from the noise, while with higher T_3 we can have better contrast until we reach a point where once again the invariance implies the loss of discriminative information.

3) Cross-validation : features: It is interesting to note that the mean features always seem to be far behind, but also that the full vector has a dimension of 1280×3 and is still better than the mean alone thus the dimensionality reduction by a factor of 3 doesn't compensate the loss of information induced by not using either the max or the std coefficients.

Now we can also try to add the coefficients of the first scattering layer leading to a feature vector with basically $[std(S_2), max(S_3)]$. The dimension will be greater by 80 coefficients so we have a feature vector of size $80+80\times2\times8$.

We will then keep this version of the algorithm by using informations from the first layer and the second layer. This is natural since we need both to keep the information about the signal. For example when looking at reconstruction problems, the coefficients of all the layers have to be kept in order to have better reconstructions.

4) Results Summary and Discussions: By using two layers we are able to reach an accuracy of 33% for 45 classes using a feature vector of size $1280 + 80$ which is reasonable and after optimizing the number of hidden neurons (4000).
It is clear that the parameters Q_2 and J_2 are sub-optimal because we should try to optimize not only the combination that brings a rich representation but also the one that has the best compromise of information gain over dimensionality. However, the option to optimize simultaneously all the filter bank coefficients (T_3, Q_2, J_2) involves high level machine learning techniques and wasn't used here.
Finally, the accuracy gain (compared to the accuracy of the one layer architecture) doesn't justify the use of a second scattering layer. The reason for this small improvement is also the dimension of the feature vector compared to the information gain.
One way to mitigate this problem might be to use some sort of dimensionality reduction on this second scattering layer through a possible hard thresholding (we aim to eliminate the values not coding information about the birds for example).

C. Two Layers with U_2 aggregation

1) Architecture: The aim of this new approach is to combine the information gain of the second scattering layer while finding the best compromise between dimensionality and information. In order to do so a natural approach is to see that not all the ψ_{λ_2} filters are suited to encode the bird song behaviours. It is then natural to try to put some kind of thresholding at this level. Multiple approaches can be used. The one used here is to simply make a linear combination of the U_2 decomposition with respect to the λ_2 dimension. Another approach could be to directly find the best wavelets for the second layer through some sparse coding techniques.
Let's see the scheme :

$$\begin{aligned} U_2(t,\lambda_1,\lambda_2) \quad \rightarrow \quad & U_2'(t,\lambda_1,\lambda_2') \\ & \downarrow \\ & |U_2' \star \psi_{\lambda_3}| := U_3'(t,\lambda_1,\lambda_2',\lambda_3) \\ & \downarrow \\ & U_3' \star \phi_4 := S_4'(t,\lambda_1,\lambda_2',\lambda_3) \end{aligned} \tag{7}$$

Where we have, using the previous notations :

$$U_2'(t,\lambda_1,\lambda_2') = \sum_{i=1}^{|\lambda_2|} k_i * U_2(t,\lambda_1,i)$$

$$U_2' \in \mathbb{R}^{|\lambda_1|\times N}$$

Clearly the point is not only to reduce the dimension but also to introduce some thresholding (by weighting the contribution of the given ψ_{λ_2} with a β_i. The set of weighting coefficients can be optimized by multiple techniques. The most natural one could be to try to find some clustering of the data vectors and to take the centroid properties for the weighting. Since the aim of this section is to prove the usefulness of

this approach, only two sets of coefficients will be tested and compared. The first one is by taking all the coefficients equal to 1 thus there are no thresholding but aggregation of the λ_2 component. The second approach will be done using β_i of a pyramidal shape (attenuating the first and last ψ_{λ_2} filters contributions. Let's first look at some examples of what we are trying to threshold.

2) Examples: There are 2 different signals and each time is presented the U_2 for each ψ_{λ_2}. The second plot is then a linear combination of these by groups of 3. Note that these representations are made with the coefficients $Q_2 = 1, J_2 = 14$.

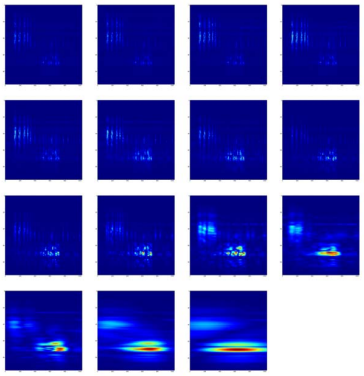


Fig. 9. Signal name : RN5821.wav, each subplot $i = 0, ..., 14$ represents $||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2=i}|$, the application of the i^{th} ψ filter of the second filter bank on U_1 which are then re-normalized independently.

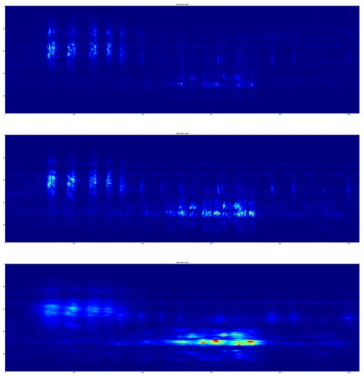


Fig. 10. Each subplot $j = 0, 1, 2$ is equal to $\sum_{i=0}^{4} U_{2,j*5+i}$ representing an element wise addition of the previous subplots (5 by 5 with $\beta_i = 1 \forall i$ so no thresholding of a particular $\psi_{\lambda_2=i}$)

In this last case we can see that a "denoising" technique could be efficient.
Note that before the linear combination all the layers have been normalized independently from each other so their values are between 0 and 1.

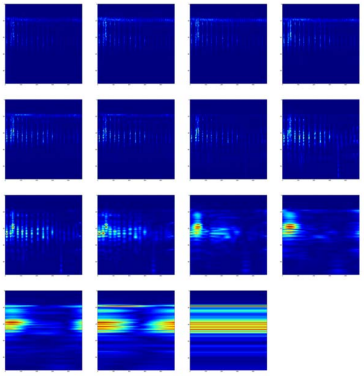


Fig. 11. Signal name : RN1017.wav, each subplot $i = 0, ..., 14$ represents $||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2=i}|$, the application of the i^{th} ψ filter of the second filter bank on U_1 which are then re-normalized independently.

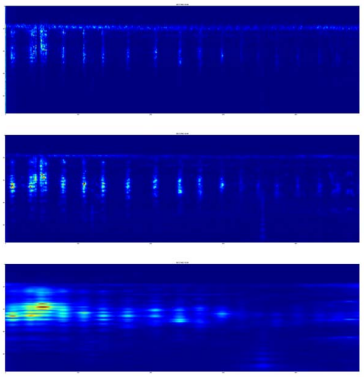


Fig. 12. Each subplot $j = 0, 1, 2$ is equal to $\sum_{i=0}^{4} U_{2,j*5+i}$ representing an element wise addition of the previous subplots (5 by 5 with $\beta_i = 1 \forall i$ so no thresholding of a particular $\psi_{\lambda_2=i}$)

3) Constant Weighting: In this case we are using $\beta_i = 1, \forall i$ if we are treating the normalized decompositions, thus we use $\beta_i = \frac{1}{max(|U_1 \star \psi_{\lambda_2=i}| \star \phi_3)}$ in order to have a "normalized version" of the decompositions. The thing here is not to threshold but to aggregate the information in order to reduce the feature vector length. it is import to note that the features computed (mean,max,std) are sensitive to this change.
Once again, we need to cross-validate the T_3 coefficient.

We can then reach an accuracy of 35% by simply aggregating information. Even though the improvement is not great (only 2%) we will now see that introducing active weighting can better improve this result.

4) Pyramidal Weighting: The coefficients β_i are now of the pyramidal form and then the coefficients are weighted by the normalization so : $\beta_i* = \frac{1}{max(|U_1 \star \psi_{\lambda_2=i}| \star \phi_3)}$ thus, we hardly threshold the extreme cases. This is from the fact that insects are usually present for the first filters and background noises

are encoded by the last filters.

It is clear that the weighting is playing an important role for the accuracy. We are able to reach 39% of accuracy with this coefficients and architecture which is yet another improvement from constant coefficients.

V. Conclusion

The presented challenge is a unique experience for testing algorithms around bioacoustic classification. This task contains all the main difficulties and problems encountered in a real life problem while involving a huge dataset.
We have seen that the scattering transform provides a wonderful new data representation but it is not yet easy to know how to extract features from its coefficients. We obviously didn't try every possible algorithm but the idea of thresholding/combination in the λ_2 dimension seems to be interesting. The key part is to be able to capture feature with a time-invariant algorithm in term of the position of the feature in a signal, but, we need to keep a "time" component to analyse frequencies evolution inside the features.

Given the massive dataset it has also been a unique experience to learn ways and methods to deal with computational limitations and the challenge deadline. more importantly, this has helped to deeply understand the scattering network and the pre-processing part in general when trying to apply machine learning algorithms for a given task.

Acknowledgment

I particularly want to thank Pr Stéphane MALLAT for accepting me into his team for months during this research internship and for guiding me through this task. I also want to thank all his team and especially Vincent LOSTANLEN with whom I learned a lot about machine learning in general and the right way to approach such problems. I thank Pr. Hervé GLOTIN and his team who allowed me to use their server and for all their advices and analysis on bioacoustic and signal processing in general. I also thank the SABIOD Project [2] MI CNRS MASTODONS for supporting me.

[2] http://sabiod.org

References

[1] A DFT and FFT tutorial. http://www.alwayslearn.com/dft%20and%20fft%20tutorial/DFTandFFT_BasicIdea.html, June 2014.

[2] DI ENS. Scatnet toolbox. http://www.di.ens.fr/data/software/scatnet/, 2011-2015.

[3] H Goeau, H Glotin, WP Vellinga, and A Rauber. Lifeclef bird identitfication task 2014. Clef working notes, 2014.

[4] Anden J. and Mallat S. Deep scattering spectrum. Deep Scattering Spectrum,Submitted to IEEE Transactions on Signal Processing, 2011.

[5] Alexis Joly, Herve Goeau, Herve Glotin, Concetto Spampinato, and Henning Muller. *Lifeclef 2014: multimedia life species identification challenges*. Information Access Evaluation. Multilinguality, Multimodality, and Interaction, Springer International Publishing, 2014.

[6] Stéphane Mallat. *A wavelet tour of signal processing*. Academic press, 1999.

[7] Stephane MALLAT. Group invariant scattering. Communications in Pure and Applied Mathematics, vol. 65, no. 10, pp. 1331-1398, 2012.

[8] Joo Martins. How to implement the fft algorithm. http://www.codeproject.com/Articles/9388/How-to-implement-the-FFT-algorithm, February 2005.

[9] Vlodymyr Myrny. A simple and efficient fft implementation in c++:part 1. http://www.drdobbs.com/cpp/a-simple-and-efficient-fft-implementatio/199500857, May 2007.

[10] BALESTRIERO Randall and GLOTIN Herve. Bachelor's research internship report. http://www.lsis.univ-tln.fr/~rbalestriero/intern_report_l3.pdf, December 2013.

[11] BALESTRIERO Randall and GLOTIN Herve. Cigal : C++ signal scattering and wavelet fast representations. https://www.academia.edu/10202482/CIGAL_C_sIGnal_scAttering_and_waveLet_fast_representations, December 2014.

[12] Craig Stuart Sapp. Wave pcm soundfile format. https://ccrma.stanford.edu/courses/422/projects/WaveFormat/, December 2011.

[13] Laurent Sifre and Stephane Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[14] Marie Trone, Hervé Glotin, Randall Balestriero, David E Bonnett, and Jerry Blakefield. Heterogeneity of amazon river dolphin high-frequency clicks: Current odontoceti bioacoustic terminology in need of standardization. *Proceedings of Meetings on Acoustics*, 22(1):070003, 2015.

[15] Wikipedia. Fast fourier transform. http://en.wikipedia.org/wiki/Fast_Fourier_transform, December 2014.

[16] Wikipedia. Wav. http://en.wikipedia.org/wiki/WAV, November 2014.