CIGAL
C++ sIGnal scAttering & waveLet fast representations

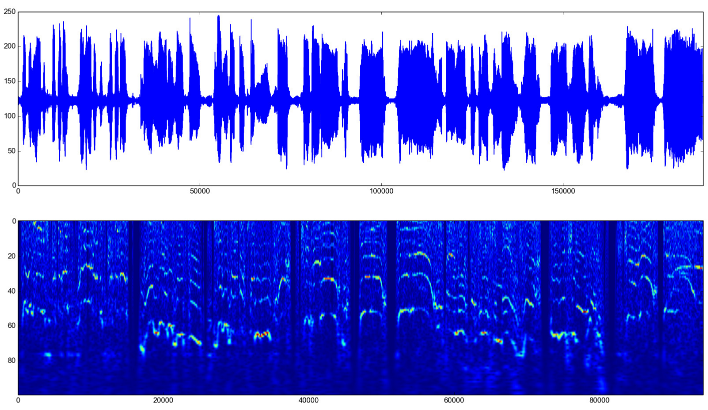
Randall BALESTRIERO
randallbalestriero@gmail.com

Department of Mathematics, Pierre et Marie Curie University Paris 6

Hervé GLOTIN

Aix Marseille Universit, ENSAM, Marseille
Universit de Toulon, CNRS, LSIS UMR, La Garde
Institut Universitaire de France (IUF), Paris

SABIOD.ORG
Scaled Acoustic Biodiversity
Mastodons Big Data



Contents

1 Introduction

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, classify linearly separable data is 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, everything around audio processing uses MFCC until now. This toolbox gives the basic tools for audio representation using the C++ programming language by providing an implementation of the Scattering Network [4] which brings a new and powerful solution for these tasks. The tool-kit of reference in scattering analysis is the SCATNET from Mallat et al. [1]. This tool is an attempt to have some of the scatnet features more tractable in large dataset. 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. One motivation for this work is the collaboration between DI ENS and the university of Toulon through the SABIOB Scaled Acoustic project.[15] [14]. This toolbox gives out of the box executables that can be used by simple bash commands. Examples are given in the README file. Finally, for each presented algorithm, a graph is provided in order to summarize how the computation is done in this toolbox.

2 Audio File io

2.1 File Structure

The WAV file is an instance of a Resource Interchange File Format (RIFF) defined by IBM and Microsoft. The header part of this file is made of complementary chunks describing the architecture of the wav allowing easy information storing. Let's see how these chunks are organized in a WAV file :

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

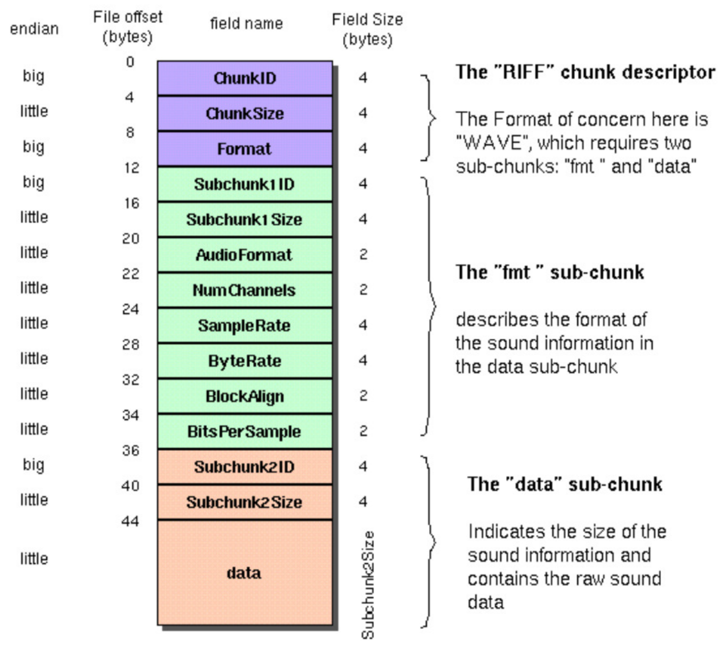


Figure 1: The Canonical WAV file format `https://ccrma.stanford.edu/courses/422/projects/WaveFormat/`

The file is made of three main chunks each having a specific role that we will describe here.

- ChunkID identifies the type of the first chunk with four characters : "RIFF".
- ChunkSize is the size of the file left from this point. It will be 36 (sum of the other chunks sizes) plus Subchunk2Size and this can be easily seen by summing the different sizes on the right of the header representation.
- Format will be four characters : "WAVE" (this allows us to check if we are really reading a wav file during the process).

- SubChunk1ID identifies the second chunk. It is a four characters name : "fmt " and starts the data description block.
- SubChunk1Size is simply the size of this block which is 16.
- AudioFormat, also called Format tag, is the option indicating the data compression used. It is almost always equal to 1 which stands for : no compression is used.
- NumChannels is the number of channels (1 for mono and 2 for stereo).
- SampleRate is simply the number of samples per second, the frequency.

- ByteRate is the average number of bytes per second, this can be found with the following formula : SampleRate $*$ NumChannels $* \frac{\text{BitsPerSample}}{8}$.
- BlockAlign won't be necessary for us. It can be computed with the formula : $NumChannels * \frac{\text{BitsPerSample}}{8}$.
- BitsPerSample can either be 8 or 16 but in general the later is used.

- SubChunk2ID identifies the last chunk block, it is made of the four characters : "data".
- Subchunk2Size is the size of the file left which is just the size of the data.
- Data is the block containing the values of the signal in the standard pulse-code modulation representation.

2.2 Implementation

The use of the built-in class WAV is simple, the only thing to provide is the name of the wav file. This can be done during the instantiation of the class or at any other time. Let's look at an example.

```
WAV<double> Signal("mysignal.wav");
WAV<double> Signal2;
"mysignal.wav">>Signal2;
"mysignal2.wav">>Signal;
```

The template parameter can be ignored leading to the default value : float. One instance of the class can be used for different wav files which can be useful. This WAV variable allows easy interactions and can provide informations about the loaded file :

```
Signal._Size;
Signal._SampleRate;
Signal_NumberOfChannel;
Signal[i];//return the ith value of the loaded signal
```

Finally, to export the loaded file two options are available. Firstly, it is possible to export it into a txt file which only export the signal data disregarding all the other informations. This loss can be avoided using a special method which exports the data back into a wav file, with the following syntax :

```
Signal>>"newsignal.txt";//to .txt
Signal.PrintWav("newsignal.wav");//to .wav
```

With this implementation, WAV can be seen as a special type. Note that no normalization is used. In fact, only the user can define the normalizing constant he is interested in (max, L^2-norm,...) and so has to apply it after the import. Finally, for an external use of this toolbox, one should not need to use this class since it is just here as a input/output convenience for the other algorithm we will now describe.

3 Fourier Transform

3.1 Definitions

A sinusoidal wave is characterised by three parameters: amplitude, frequency and phase.

- The amplitude is the amount the function varies, positively or negatively, from zero in the y direction.
- The frequency is how many complete cycles there are of the wave in unit distance on the x axis (which often measures time).
- The phase is relevant when comparing two waves of the same frequency. It is how much (measured in degrees or radians) one is shifted relative to the other on the x axis.

This terminology comes from sound engineering where higher frequency sounds have higher pitch and waves of greater amplitude are louder. As an alternative of specifying the frequency, the number of cycles in unit distance, we can instead specify the wavelength : the length of one cycle. The higher the frequency, the shorter the wavelength. The lower the frequency the longer the wavelength.
The Nyquist frequency is the maximum frequency that can be detected for a given sampling rate and it is half of it. This is because in order to measure a wave one needs at least two sample points to identify it (trough and peak).
We will abbreviate the continuous Fourier transform with CFT and the discrete Fourier transform with DFT.

Interpretation of the CFT Using the Euler's formula, we can see the Fourier Transform as a decomposition of a signal into complex sinus by the use of convolutions.

$$e^{ix} = \cos(x) + i\sin(x)$$

$$\begin{aligned}
\hat{f}(\xi) &= \int_{-\infty}^{\infty} f(x)e^{-2\pi i x\xi}dx \\
&= \int_{-\infty}^{\infty} f(x)(\cos(-2\pi x\xi) + i\sin(-2\pi x\xi))dx \\
&= \int_{-\infty}^{\infty} f(x)\cos(-2\pi x\xi) + f(x)i\sin(-2\pi x\xi)dx \\
&= \int_{-\infty}^{\infty} f(x)\cos(-2\pi x\xi)dx + \int_{-\infty}^{\infty} f(x)i\sin(-2\pi x\xi)dx
\end{aligned}$$

3.2 Fast Fourier Transform

We will now denote x_k as the k^{th} value on the signal in the time space and X_k the k^{th} value of the signal in the frequency domain, N will denote the length of the signal. A length of N means the indices range from 0 to $N-1$.
The fast Fourier transform (FFT) is a instance of DFT which is able to perform the DFT in $O(N\log(N))$ complexity.
The DFT formula using the Twiddle Factor notation :

$$\forall k \in \mathbb{Z}, X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi i k n}{N}}$$

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{kn}$$

As we can see, we need to perform N operations for each $X_k, k \in \{0, 1, ..., N-1\}$ thus we are in $O(N^2)$ complexity.
Note that it is possible to use the scaling factor :$1/\sqrt{N}$ in order to have an unitary operator (Parseval's theorem) which implies that the sum (or integral) of the square of the function is equal to the sum (or integral) of the square of its transform which is not needed in this toolbox thus not used.
In order to go from N^2 operations to $N \log(N)$ operations, three main concepts have to be defined :

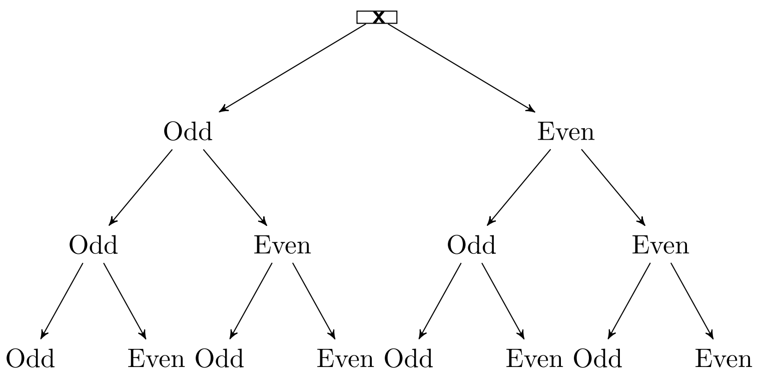
- The Danielson-Lanczos Lemma
- The Twiddle Factor properties
- The Butterfly Scheme

3.2.1 Danielson-Lanczos Lemma

This theorem is the foundation of the FFT by allowing a divide and conquer strategy. In fact, we have the following relations :

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi kn}{N}} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{\frac{-i2\pi 2kn}{N}} + x_{2n+1} e^{\frac{-i2\pi(2k+1)n}{N}} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{\frac{-i2\pi kn}{N/2}} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{\frac{-i2\pi 2kn - i2\pi n}{N}} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{\frac{-i2\pi kn}{N/2}} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{\frac{-i2\pi kn}{N/2}} \end{aligned}$$

For every X_i we can now divide the N sums into two different summation group (Even and Odd). Note that for the special case $N = 2$ the sums are removed and n is replaced by 0 which means that we are left with a simple linear combination of the input signal and the Twiddle Factor. If we apply this recursively we obtain the following architecture :



And now for any given input size we are able to break it done into a linear combination of the input signal with twiddle factors. For example, if $N = 4$ we have after full decomposition :

$$X_k = x_0 + W_2^k x_2 + W_4^k x_2 + W_4^k W_2^k x_3$$

And for $N = 8$:

$$X_k = x_0 + W_2^k x_4 + W_4^k x_2 + W_4^k W_2^k x_6 + W_8^k x_1 + W_8^k W_2^k x_5 + W_8^k W_4^k x_3 + W_8^k W_4^k W_2^k x_7$$

This puts a constraint though, the signal length has to be a power of 2. The number of decomposition is thus $\log_2(N)$. If the signal size is not a power of 2 it is necessary to use zero padding (add as may 0 as necessary at the end of the input). Padding with 0 in time domain leads to an interpolation of the FFT. Middle zero padding the FFT (in the frequency domain) interpolates the IFFT (time domain). Periodizing in the frequency domain implies sub-sampling in the time domain (this will be useful for the Scattering Network).
One last thing to notice here is the order of the input values in the decomposition. Because of the nature of this decomposition (even/odd) we end up with the x terms being rearranged in a specific order : the bit-reversal order. This can be found by taking the symmetric of the binary position of the input value as seen in this little example for $N = 8$:

$$\begin{aligned}
0 &: 000 \to 000 : 0\\
1 &: 001 \to 100 : 4\\
2 &: 010 \to 010 : 2\\
3 &: 011 \to 110 : 6\\
4 &: 100 \to 001 : 1\\
5 &: 101 \to 101 : 5\\
6 &: 110 \to 011 : 3\\
7 &: 111 \to 111 : 7
\end{aligned}$$

3.2.2 Twiddle Factor Properties

Complexity has already been broken down but we can still optimize the implementation by exploiting the Twiddle Factor properties using roots of unity properties. In fact we have :

$$W_N^k = e^{\frac{-i2\pi k}{N}} = \cos(2\pi k/N) - i\sin(2\pi k/N)$$

Thus for $N = 2$:

$$\begin{aligned}
W_2^0 &= W_2^2 = W_2^4 = ...\\
W_2^1 &= W_2^3 = W_2^5 = ...
\end{aligned}$$

And for $N = 4$

$$\begin{aligned} W_4^0 &= W_4^4 = W_4^8 = ... \\ W_4^1 &= W_4^5 = W_4^9 = ... \\ W_4^2 &= W_4^6 = W_4^{10} = ... \\ W_4^3 &= W_4^7 = W_4^{11} = ... \end{aligned}$$

And so on using trigonometric properties, with functions here being $N\pi$-periodic. So using this will allow us to perform less Twiddle Factor computation at each level.

3.2.3 Butterfly Scheme

Finally, the last brick is the butterfly scheme that can be seen in the next diagram3.3 allowing an in-place FFT which is memory friendly.

3.3 Implementation

Firstly, our implementation is made of three nested loops, the main one which will go through the $\log(N)$ levels of decomposition. The second one will go through the blocks inside a specific level (the last level as 1 block whereas the first level as $N/2$ blocks). Finally the last loop will go inside a block (a block on the first decomposition level will have size 2 while the block in the last decomposition level will be of size N). For each level (i), only 2^i Twiddle factors are computed in the main loop where i is the decomposition level from 0 to $\log(N) - 1$. A simple temporary variable is used in order to perform the swapping operations. Here is an instance of this implementation for $N = 8$ and a human friendly output explaining the performed steps.

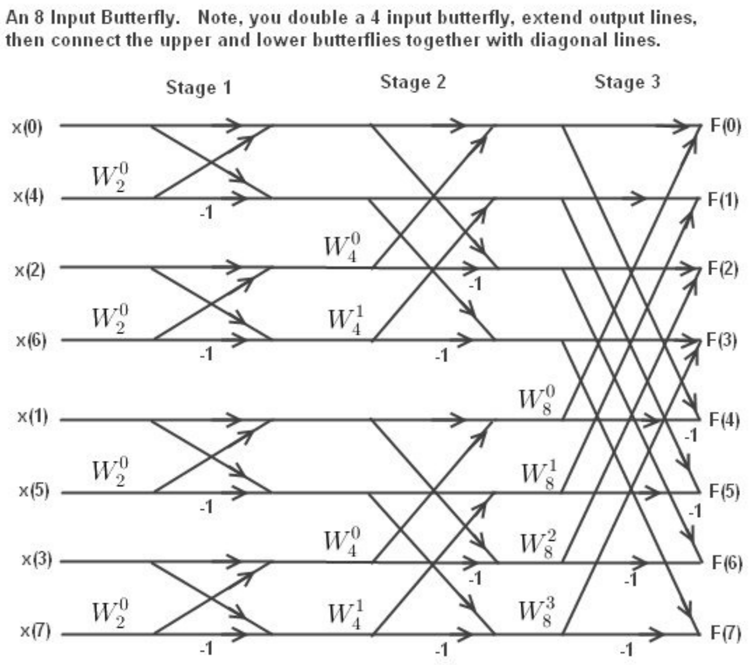

An 8 Input Butterfly. Note, you double a 4 input butterfly, extend output lines, then connect the upper and lower butterflies together with diagonal lines.
Stage 1
Stage 2
Stage 3
x(0)
x(4)
x(2)
x(6)
x(1)
x(5)
x(3)
x(7)
W_2^0
W_4^0
W_4^1
W_8^0
W_8^1
W_8^2
W_8^3
-1
F(0)
F(1)
F(2)
F(3)
F(4)
F(5)
F(6)
F(7)

Figure 2: Full FFT computation with $N = 8$ [1]

```
Level : 0
 W[0]=W(0,2)
  Block : 0
     signal[1]*=W[0]
     tmp=signal[0]
     signal[0]+=signal[1]
     signal[1]=tmp-signal[1]

  Block : 1
     signal[3]*=W[0]
     tmp=signal[2]
     signal[2]+=signal[3]
     signal[3]=tmp-signal[3]

  Block : 2
     signal[5]*=W[0]
     tmp=signal[4]
     signal[4]+=signal[5]
     signal[5]=tmp-signal[5]

  Block : 3
     signal[7]*=W[0]
     tmp=signal[6]
     signal[6]+=signal[7]
     signal[7]=tmp-signal[7]
```

```
Level : 1
 W[0]=W(0,4),W[1]=W(1,4)
  Block : 0
     signal[2]*=W[0]
     tmp=signal[0]
     signal[0]+=signal[2]
     signal[2]=tmp-signal[2]
     signal[3]*=W[1]
     tmp=signal[1]
     signal[1]+=signal[3]
     signal[3]=tmp-signal[3]

  Block : 1
     signal[6]*=W[0]
     tmp=signal[4]
     signal[4]+=signal[6]
     signal[6]=tmp-signal[6]
     signal[7]*=W[1]
     tmp=signal[5]
     signal[5]+=signal[7]
     signal[7]=tmp-signal[7]
```

```
Level : 2
 W[0]=W(0,8),W[1]=W(1,8)
 W[2]=W(2,8),W[3]=W(3,8)
  Block : 0
     signal[4]*=W[0]
     tmp=signal[ 0 ]
     signal[0]+=signal[4]
     signal[4]=tmp-signal[4]
     signal[5]*=W[1]
     tmp=signal[1]
     signal[1]+=signal[5]
     signal[5]=tmp-signal[5]
     signal[6]*=W[2]
     tmp=signal[2]
     signal[2]+=signal[6]
     signal[6]=tmp-signal[6]
     signal[7]*=W[3]
     tmp=signal[3]
     signal[3]+=signal[7]
     signal[7]=tmp-signal[7]
```

The Twiddle Factors are computed at the start of each main loop computing the needed values which are then reused throughout the blocks, meaning that for the first level only one value is computed and then reused all along the blocks. Here is an example of the use :

```
WAV<> wav("signal.wav");//load a wav into float type array
 fft<> signalfft(signal.ptr(),signal._Size);//default padding option=1
 signalfft.ComputeFFT();
 signalfft.ComputeIFFT();//get back to the original signal
 signalfft[2];//access the second coefficient
 signalfft>>"signalfft.txt";//export it
wav<<"processed.wav";//load a new wav
 signalfft.ComputeFFT(wav.ptr(),wav._Size);//perform a new FFT
```

Note that the parameters of the fft class are by default float and float, the first one stands for the type of the input signal and the latter for the coefficients type (complex<float>). Finally the padding option which by default is 1 can be set to 0 if the user is sure that the given signal is already a power of 2, this force to skip the padding part resulting in faster computation. Also the coefficients are stored as complex type even after having performed an IFFT meaning that one needs to use a typecast to retrieve the original float type signal for example.

3.3.1 Inverse Fourier Transform

In order to simplify the algorithm we sill use the following formula :

$$IFFT(x) = \frac{1}{N} conj(FFT(conj(x)))$$

3.3.2 Graph

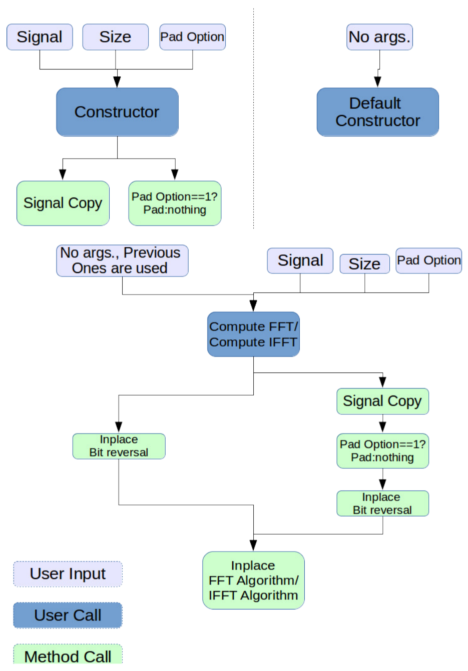


Figure 3: FFT Summary Diagram

4 Spectrogram

Each X_k is a complex number that encodes how strongly the oscillation at this frequency is represented in the data but by doing an FFT we loose the time component. A useful tool is the spectrogram allowing to retrieve part of the time information. The main idea is to perform multiples FFT on a signal each one being located enough in time so the frequency information gained by the

FFT can also be linked to a more or less specific time position in the signal. Note however that precision in both time and frequency is impossible to get but depending on the needs one can choose which one to enhance by modifying the size of the considered window. Larger window gives better frequency resolution but lesser time precision and vice-versa. It is easy to picture the fact that smaller windows are better for the high frequency part allowing good time precision while for low frequency a larger window has to be used for being able to capture it. This problem is lessen in wavelet decomposition and thus the scattering network since this window size is not constant anymore.

4.1 Algorithm

Conceptually a spectrogram is computed with the following scheme :

- splitting the signal into overlapping (or not) parts of equal length defined by the user.
- applying to each of these chunk a windowing function (typically hanning or hamming) in order to remove artefacts by periodizing the function so the limit points (start and end of the chunk) are equal. This part is called apodization
- computing the FFT on each of these chunks
- for each computed FFT, taking the absolute value of the coefficients will give the columns of the spectrogram.

The windowing is needed since the FFT computation presumes that the input data repeats over and over. This is important when the initial and final values of the data are not the same because the discontinuity causes artefacts in the spectrum computed by the FFT.
In addition, in this toolbox, only the first half of the FFT coefficient are put into the spectrogram thus avoiding symmetrical redundancy. This is due to the fact that our input signal is real and so the second half of the FFT coefficients is simply the complex conjugate of the first half, since in the spectrogram we display the absolute value of the coefficients, we get symmetry about the middle point.
Most window functions afford more influence to the data at the center of the window than tohe t data at the edges, which represents a loss of information. To mitigate that loss, it is common to use overlapping in time (usually 50%).

4.2 Implementation

It is important to note that the spectrogram (2D-matrix) is stored by column and not by line for faster computation. In fact, during the spectrogram calculation we need to access this matrix column-wise. The operator [] returns the column while the operator () takes two arguments and return the corresponding value in a normal way. Let's look at an example :

```
spectrogram<> b("signal1.wav",256,0.25);//default window function : hamming
WAV<> wav("signal2.wav");//load another wav
b.Perform(wav.ptr(),wav._Size);//compute spectrogram given these new entries
    //and default parameters with the already declared spectrogram variable
b>>"lifespectro.txt";//write the matrix into a txt file
b[1][0];//second column, first element
b(0,1);//first line second element same result as above
```

The template parameter defines the coefficients type. The default value is float. Also note that no transformation is performed after the absolute value is computed, which means that if one want to apply a logarithmic function (most common one) this has to be done after computation.
The apodization can be done using one of the available windowing function :

- hamming
- hanning
- triangular
- hann poisson

but can also be used with a specific user defined function passed as last argument when calling the Perfom method.

4.2.1 Graph

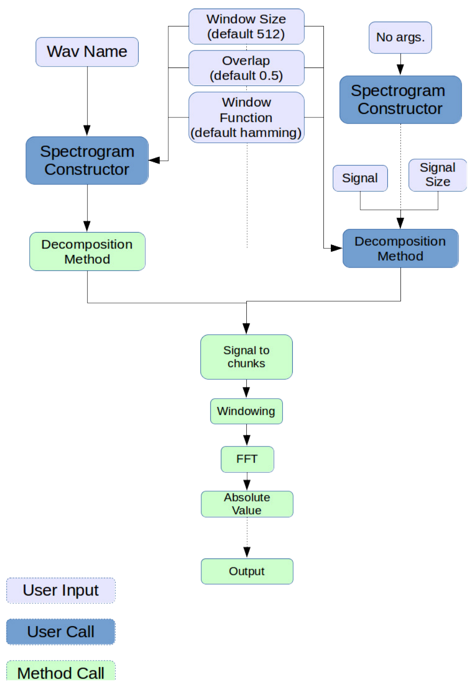


Figure 4: Spectrogram Summary Diagram

4.3 Examples

Let's look at some spectrogram examples. Note that a logarithmic function has been applied to the computed values (improving coefficient representation for us). The signals are from a bird of the BIRDLIFE CLEF Challenge 2014 [2]. and a Inia dolphin.

[2] `http://www.imageclef.org/lifeclef/2015`

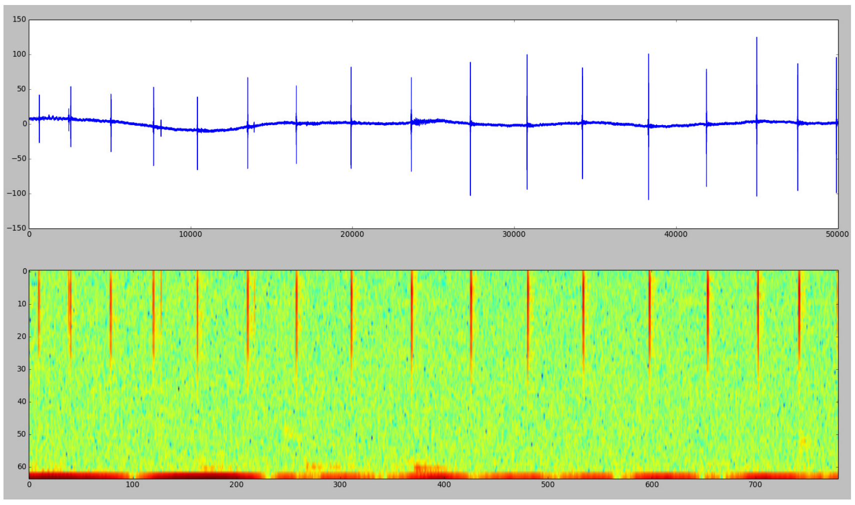


Figure 5: Inia Dolphin Slow Clicks : Spectrogram 128 50%

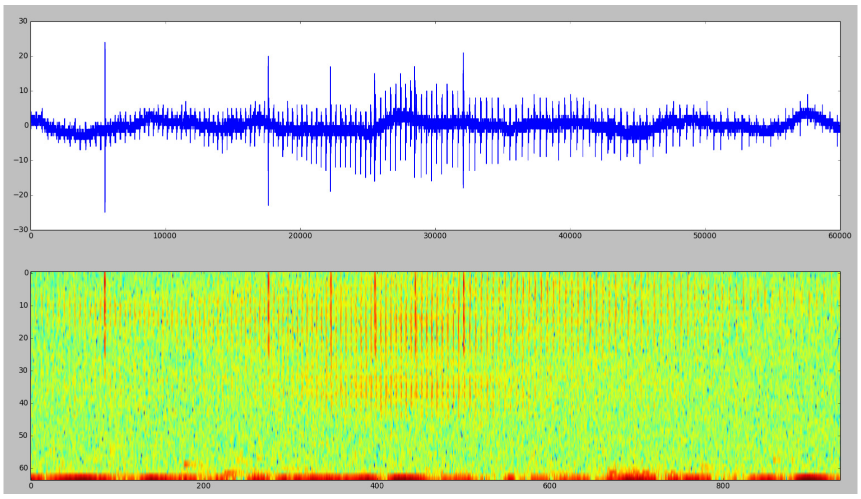


Figure 6: Inia Dolphin Fast Clicks : Spectrogram 128 50%

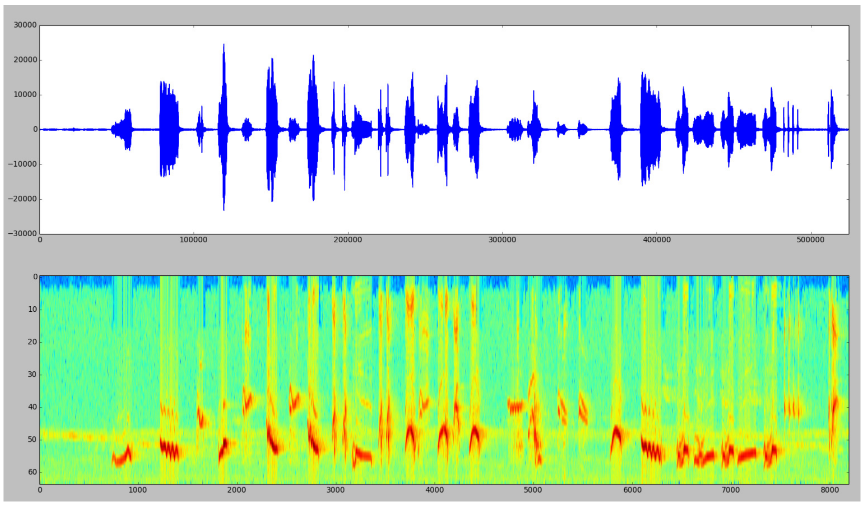


Figure 7: Bird : Spectrogram 512, 50%

5 Scattering Network

The Scattering Network aims to find a better data representation after numerous transformations of a raw input. It's been developed by Stéphane MALLAT and its team and his team in matlab, which is not the fastest implementation. In fact, this algorithm just started to be applied in concrete challenges and problems. We will review its core ideas and the implementation architecture I chose.

5.1 Introduction

The basic idea is to perform series of linear and non linear operations. The linear operations are done through the convolutions while the non linear ones are the use of the absolute value on these convolutions. The use of the latter allows fast convergence by the contractive property. The convolutions are basically decomposing the signal into a wavelet basis. A parallel can be made with the FFT and the complex sine decomposition. The structure itself of the network can be compared to a Convolutionnal Neural Network where the filters are computed and fully determined by the meta parameters while in a CNN they are learned during training. This is a huge 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,reconstruction,... Let's look at the general picture of the scattering network and analyse it briefly.

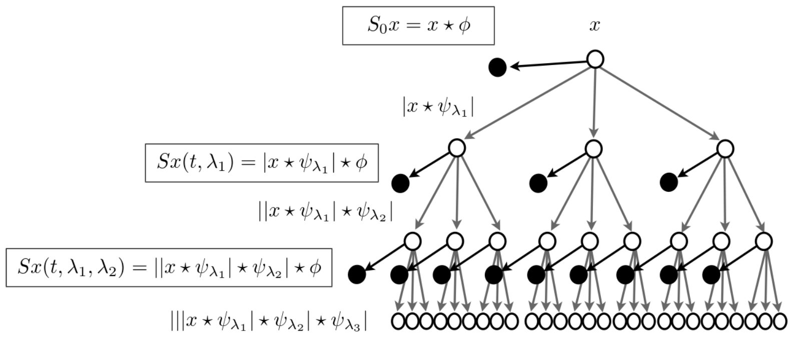

$S_0x = x \star \phi$
x
$|x \star \psi_{\lambda_1}|$
$Sx(t, \lambda_1) = |x \star \psi_{\lambda_1}| \star \phi$
$||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}|$
$Sx(t, \lambda_1, \lambda_2) = ||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}| \star \phi$
$|||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}| \star \psi_{\lambda_3}|$

Figure 8: Scattering Summary.[6]

In this case the scattering network is made of 3 layers. Each layer has low-pass filters (ϕ) and high-pass filters (ψ). In our specific case of 1D signals, there is only one ϕ per layer. Given an input signal x of size N we perform a low-pass decomposition (S_0x) by performing the convolution $x \star \phi$ and a high-decomposition leading to a output size of $2N/T \times \text{NumberOfPsisFilters}$ by performing NumberOfPsisFilters convolutions $x \star \psi_{i,\lambda 1}$ where $\psi_{i,\lambda 1}$ is the i^{th} filter of the ψ-filter bank generated by the meta parameters $\lambda 1$. Finally on this high-decomposition is applied the absolute value operation.
Then for the second layer, each one of the previous high-decomposition is treated as an input signal and the same algorithm is performed. Details about this will be given in the scattering layer section5.3 but we can already note that the meta parameters are specific to a scattering layer Finally let's review what the meta parameters are about :

- T determines the time resolution by changing the size of the filter. Small T is suited for important time precision for high frequency signals.
- Q determines the quality factor (the number of filters per octave)
- J determines the number of octave to go through.
- PE (Periodization Extent) constant used in the filter periodization (1 by default)

In order to respect this architecture, this toolbox uses a specific structure : MetaParam using default parameters and a TtoJ method :

```
MetaParam L1param(500);
//L1param._T=500, L1param._Q=1, L1param._J=8, L1param._PE=1
L1param=MetaParam(500,2);
//L1param._T=500, L1param._Q=2, L1param._J=6, L1param._PE=1
L1param=MetaParam(500,2,4);
//L1param._T=500, L1param._Q=2, L1param._J=4, L1param._PE=1
L1param=MetaParam(500,4,4,4);
//L1param._T=500, L1param._Q=4, L1param._J=4, L1param._PE=4
```

Let's now see the details of each implementation level and emphasize the implementation architecture used.

5.2 Filter Bank implementation

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 (*gamma*)
- Giving a support size, a σ and a ξ.

The first one is more practical for the ψ generation since the size and the meta parameters are fixed for a layer, we just have to loop through γ (the filter number in the filter bank). On the other hand, the second constructor is simpler for the ϕ filter generation, in fact, since only one low-pass filter is made per layer, we just have to compute ξ and σ for this filter.
Here is an example with arbitrary coefficients :

```
Filter1D* BankFilter=new Filter1D[5];
BankFilter[0]=GaborFilter1D(500,0,1,2);//500 points , xi=0,sigma=1,PE=2
for(int i=1;i<5;++i)
    BankFilter[i]=MorletFilter1D(500,2+0.5*i,0.2*i);//500 points ,xi=f(i),
                                                    //sigma=g(i),PE=1 (default)
ofstream file("filters.txt");
for(int i=0;i<5;++i){
    file<<BankFilter[i];// use of the overloaded operator
    file<<"\n";
}
delete[] BankFilter;
file.close();
```

Giving the following result :

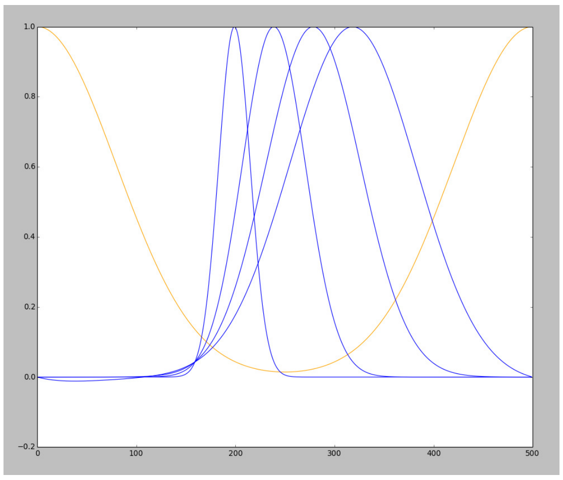


Figure 9: Filters generation example, orange : Gabor filter, blue : Morlet wavelets.

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 to perform the decomposition algorithm now. Here ξ corresponds to the central frequency and so to the global maximum position. It can be seen as a position parameter while σ is a scale parameter. In practice, in order to generate the filters we always take the mother coefficients that are transformed through a scale coefficient following exponential change. We have then as mother coefficients :

$$\Xi = \frac{\pi}{2} * (2^{-1/Q} + 1)$$

$$\Sigma = \sqrt{3} * (1 - 2^{-1/Q})$$

The scaling factor for the filter i is :$\lambda_i = 2^{-i/Q}$ which leads to the following coefficients for any given filter i for a specific layer having the same meta parameters :

$$\xi_i = \Xi * \lambda_i$$

$$\sigma_i = \Sigma * \lambda_i$$

Filters In this implementation, high-pass filters are Morlet wavelets while low-pass filters are Gabor filters. 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.

Filter Periodization In order to increase resolution of the filters, we can compute them on a bigger interval than the one we are interested in and then periodize them in the Fourier domain :

$$f(x) = \sum_{n \in \mathcal{Z}} f(x + 2\pi n)$$

In practice nothing assures the convergence for any function f but our filters are generated through Gaussian functions which assure convergence. In practice, we use $n \in \{-PE, -PE+1, ..., PE, PE+1\}$ with $x \in \{x \in \mathbb{R}, i = 0, ..., T-1 : x = i * 2\pi/T\}$ which is similar to $x \in \{0, 2\pi/T, 2*2\pi/T, ..., (T-1)2\pi/T\}$. With this definition x covers $[0, 2\pi[$ with T points linearly separated by a distance of $1/T$. In all the examples presented here a periodization extent of 1 is used.
The n coefficients affect the range on which the wavelet is evaluated which grows with bigger n :

$$[-2\pi * PE, 2\pi * (1 + PE) - 1/T]$$

It is then shrunk into the desired support size by the periodization process.

5.2.1 Graph

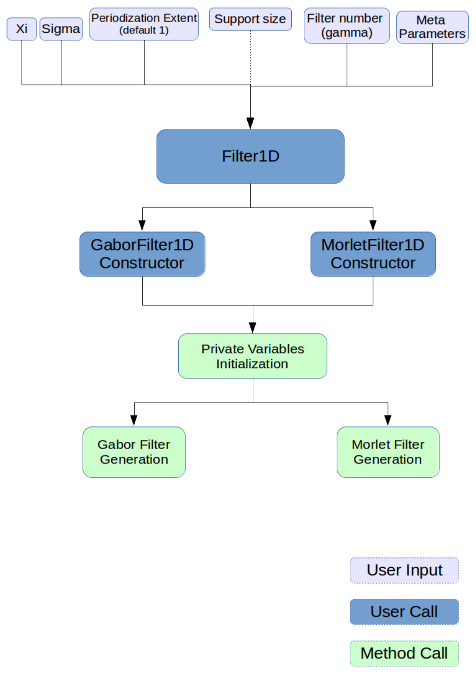


Figure 10: Filter1D Summary Diagram

5.3 Layer Implementation

The role of this class is to be the link between the raw input, the meta parameters, and the bank filters by performing the decomposition process. Firstly, this class takes a $2D$ input (the input signal 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 :$Q * J$). Then when all the ψ filters are available a Littlewood-Paley normalization is performed (due to the logarithmic spaced filters). After this, the filters are generated using the Filter1D class. 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 this toolbox. In fact, in this toolbox, the graph structure of the scattering network5.1 has been kept through $2D$ matrices. For example for the second layer if we have k filters for L1 and l filters for L2 and with $\psi_{i,j}$ being the h^{th} filter of the i^{th} layer, the

high-decomposition matrix of L2 will be :

$$\begin{pmatrix} ||x \star \psi_{1,1}| \star \psi_{2,1}| \\ \dots \\ ||x \star \psi_{1,1}| \star \psi_{2,l}| \\ ||x \star \psi_{1,2}| \star \psi_{2,1}| \\ \dots \\ ||x \star \psi_{1,2}| \star \psi_{2,l}| \\ \dots \\ ||x \star \psi_{1,k}| \star \psi_{2,1}| \\ \dots \\ ||x \star \psi_{1,k}| \star \psi_{2,l}| \end{pmatrix}$$

If one wants to select a specific $\psi_{2,i}$ decomposition it can be done by selecting the lines $i, i + l, i + 2l, \ldots$ and thus only analysing one path of the scattering network.

5.3.1 Graph

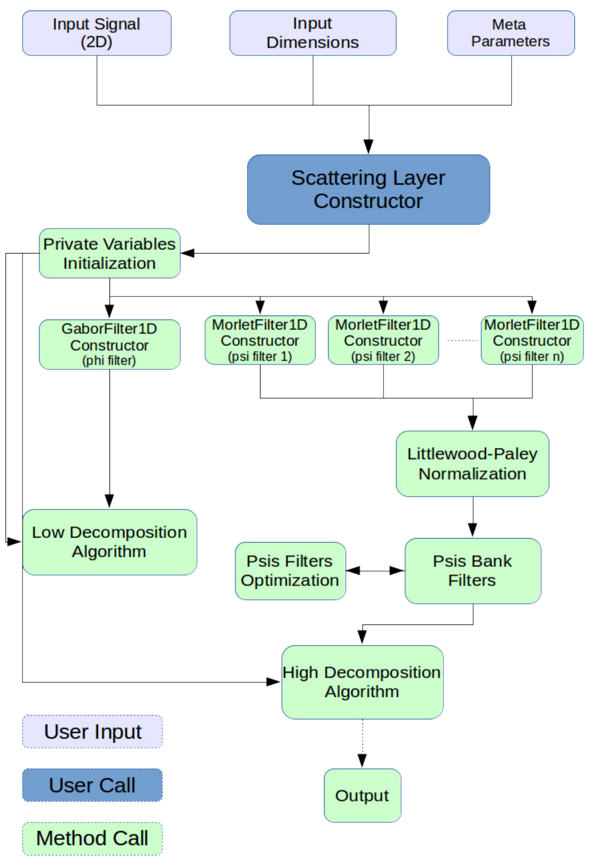

Input Signal (2D)
Input Dimensions
Meta Parameters
Scattering Layer Constructor
Private Variables Initialization
GaborFilter1D Constructor (phi filter)
MorletFilter1D Constructor (psi filter 1)
MorletFilter1D Constructor (psi filter 2)
MorletFilter1D Constructor (psi filter n)
Littlewood-Paley Normalization
Low Decomposition Algorithm
Psis Filters Optimization
Psis Bank Filters
High Decomposition Algorithm
Output
User Input
User Call
Method Call

Figure 11: Scattering Layer Summary Diagram

5.4 Decomposition Implementation

The core of the algorithm lies in this decomposition. Firstly, the convolution defined in the section 5.1 is redundant and so is only performed on every $T/2$ spaced points. This implies a reduced output length and faster computation. Thus, it is necessary to perform a periodization before computing the IFFT (allowing a time sub-sampling). The output length must then be InputSize$*2/T$. Doing this for each psis filter gives the output of the layer. Here is a simple scheme to emphasize the algorithm :

```
Data: Input,inputN,inputM,Meta Parameters
Result: Output,outputN,outputM
NumberOfPsis=J*Q;
outputN=inputN*NumberOfPsis;
outputM=inputM$*2/T$;
BankPsis creation;
Phi;
for $i = 0 \rightarrow inputN$ do
    inputFFT=FFT(input[i]);
    LowDecomposition[i]=IFFT(periodize(inputFFT.*Phi));
    for $j = 0 \rightarrow NumberOfPsis$ do
        HighDecomposition[i*NumberOfPsis+j]=IFFT(periodize(inputFFT.*BankPsis[j]));
    end
end
```

Algorithm 1: Decomposition Algorithm

With ”periodize” being the function that will periodize the result in order to sub-sample in the time domain to obtain the desired output size. In the algorithm, after a layer has performed the decompositions, filters are freed in order to reduce memory consumption. In fact, filters of computed layers wont be reused and it would be a waste to keep them.

5.5 Scattering Network Implementation

Finally here is how to perform the Scattering Network on a signal and to save the outputs :

```
MetaParam* opt=new MetaParam[3];
opt[0]=MetaParam(8,30,4,1);
opt[1]=MetaParam(64,1,1,1);
opt[2]=MetaParam(1024,1,1,1);
ScatteringNetwork decomposition(”signal.wav”,opt,3);

ofstream file;
file.open(”layer1.txt”);
file<<decomposition[0];
file.close();
file.open(”layer2.txt”);
file<<decomposition[1];
file.close();
file.open(”layer3.txt”);
file<<decomposition[2];
file.close();
delete[] opt;
```

In fact the operator [] is overloaded to return the specific layer which itself uses its overloaded operator to export the coefficients.

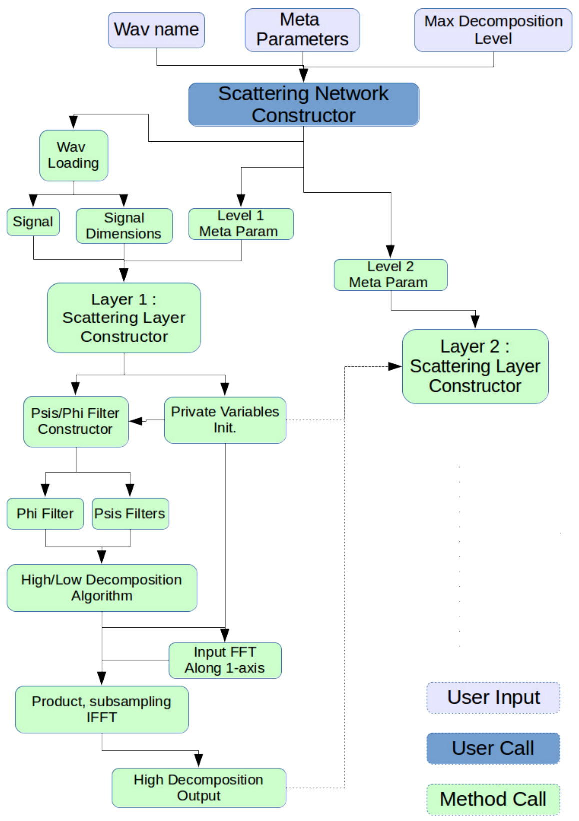

Wav name
Meta Parameters
Max Decomposition Level
Scattering Network Constructor
Wav Loading
Signal
Signal Dimensions
Level 1 Meta Param
Level 2 Meta Param
Layer 1 : Scattering Layer Constructor
Layer 2 : Scattering Layer Constructor
Psis/Phi Filter Constructor
Private Variables Init.
Phi Filter
Psis Filters
High/Low Decomposition Algorithm
Input FFT Along 1-axis
Product, subsampling IFFT
High Decomposition Output
User Input
User Call
Method Call

Figure 12: Scattering Network Summary Diagram

5.6 Examples

In the examples below we did not apply any operation (nor logarithm of re-normalization nor else). The sub-plots are from top to bottom the signal, and the L1, L2, L3 from the scattering. Ordinates are the j index.

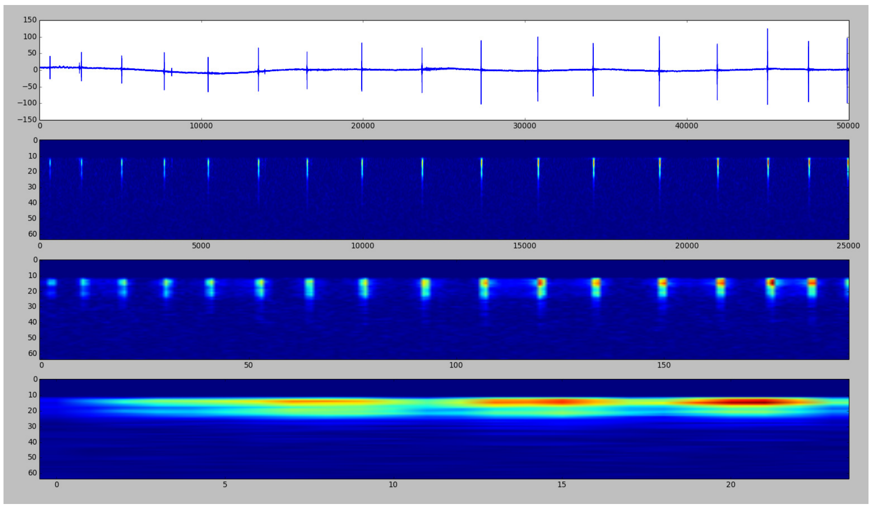


Figure 13: Signal, L1, L2, L3 of Inia Perou (slow clicks) T1:4 Q1:32 J1:2 T2:256 Q2:1 J2:1 T3:16 Q3:1 J3:1

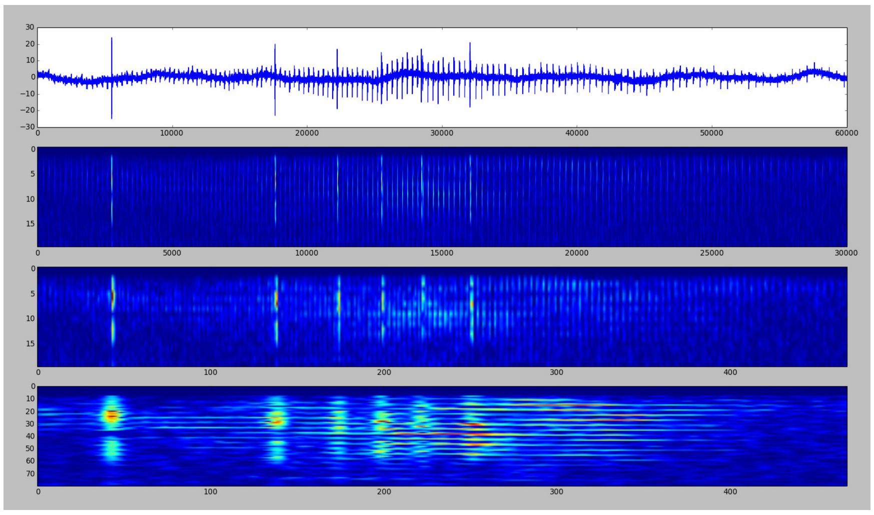


Figure 14: Signal, L1, L2, L3, Inia Perou (fast clicks) T1:4 Q1:20 J1:1 T2:128 Q2:1 J2:1 T3:2 Q3:4 J3:1

Some of the discontinuities seen on the y axis of L3 are from the way the results are stored and are due to the 2D representation of a 3D matrix. In fact here the L3 output is not of dimension 4 since L2 doesn't add a dimension because it is made of only one high-pass filter. Using only one high-pass filter for L3 would mean that the output of L3 is again of dimension 2.

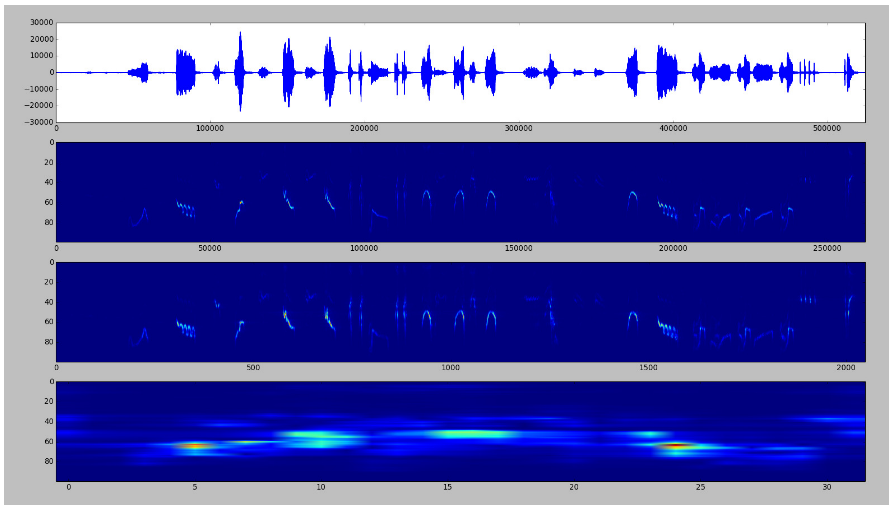


Figure 15: Signal, L1, L2, L3, Bird (BIRDLIFE CLEF Challenge) T1:4 Q1:25 J1:4 T2:256 Q2:1 J2:1 T3:128 Q3:1 J3:1

6 Data representation

In order to appreciate the decompositions done either with spectrograms or the scattering network, this toolbox provides a imagesc like utility. The presentation of it is as follow : description of the import method, optimization of the rendering algorithm, representation of the data through OpenGL. Note that a special executable file is done and can be used on every .txt file containing a $2D$ matrix by calling it. More descriptions are in the annexA.

6.1 Import data

The Data is imported by reading a .txt file. This method doesn't need to know the size of the matrix then is not the most efficient one. When reading the file, the first line is used to learn the size of the matrix resulting for the other lines in a faster loading method. In fact,the first line uses the vector push back method that expands the size of the vector by 2 times its actual size when it's full meaning that for N points only $\log(N)$ expansions are made if we started from a vector of length 1. The following lines are directly loaded into a vector of the right size.

6.2 Optimization

Rendering is done using openGL. The matrix is decomposed into squares with nodes being representation of the matrix points. Using squares instead of triangles is more efficient in this particular case since all the points lie in a $2D$ plane. The values of the points are described through the colors. Another optimization used is through arrays (either vertex arrays, color arrays,...) by enabling client states. In fact passing directly the vertex, color and points arrays reduces the number of function calls and improve performances.

6.3 Colormap

To render good spectrograms or scalograms it is necessary to have a good colormap which is simply a $\mathbb{R} \to \mathbb{R}^3$ function mapping the value of a point to a RGB color. The one used here is the same as Matlab or Python.

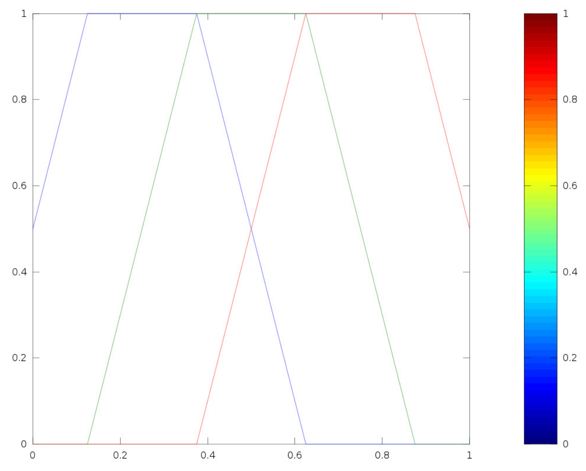


Figure 16: Colormap used in the toolbox

This colormap need normalized data as input but offers very pleasant rendering colors.

6.4 Implementation

Given a matrix $N \times M$, squares are computed using or not the original aspect ratio. If the aspect ratio is kept then a matrix of size $N \times 2N$ for example will be displayed into $[0,1] \times [0,0.5]$ and this is true for every possible ratio. However this kind of displaying can become very unpleasant for extrem ratios (few rows and millions of columns for example). That is why it is possible to disregard the original ratio drawing the matrix into the unitary square by adding a simple argument when calling the function as follow:

```
./imshow myspectro.txt//keeping original ratio
./imshow 0 myspectro.txt//squared ratio
```

The 0 option specifies to not keep the original aspect ratio meaning that now the image will be displayed as a square. During the data visualization it is possible to zoom in or out using + and -. Moves are possible through the mouse or the a,w,s,d keys.

7 Validation on a large scale bioacoustic data set

In order to test the whole tool on a real data set, we used the Bird Challenge 2014 [3] contening nearly 100 hours of recording distributed on 14K files of bird songs from the Amazonian forest.

7.1 Recall of previous attempts

Previous attempts to run scattering decomposition on this dataset were difficult. We tried first[2] to run scatnet on each file, over time windows of 300 ms. The main difficulties were then the duration of the processing, and the cuts between the windows that was difficult to manage for a second stage of classification over the whole signal. In fact, the scatnet signal length limit is of 2^{15} bins.

7.2 Results with CIGAL

CIGAL allowed on 1 CPU 2600 Mhz of the UTLN server to run all the files in 72 hours for layer 0,1 and 2. Note that once layer 0 done, computation time for the other layers decreases significantly. We then get continuous decomposition as illustrated below for several files using Morlet wavelets.

7.3 Perspective on BIRD2015

We give in fig 17 to 20 the spectrogram versus the scattering of some tropical bird species from the Bird lifeclef challenge 2014 [ref]. They demonstrate the clarity of the features. We currently use these features to train Convolutional Neural Net for bird species identification.

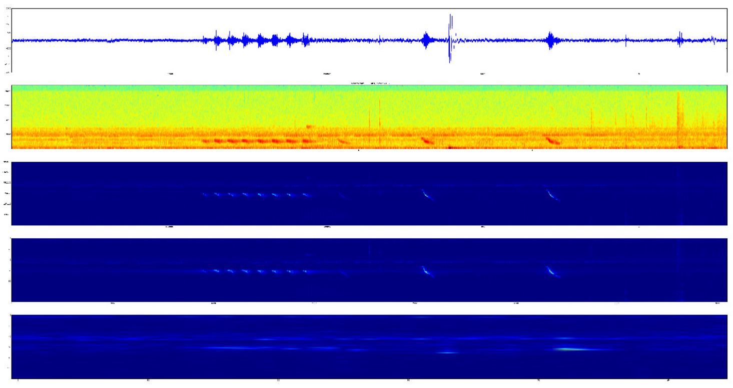


Figure 17: XC73908.wav with T1:4 Q1:20 J1: 5, T2:256 Q2:1 J2:1, T3:128 Q3:1 J3:1, Spectrogram : 128 bins and 50% overlap

[3] SABIOD.ORG

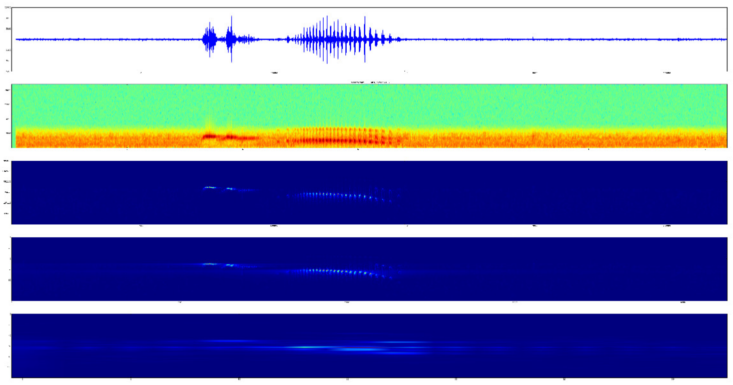


Figure 18: XC83327.wav with T1:4 Q1:20 J1: 5, T2:256 Q2:1 J2:1, T3:128 Q3:1 J3:1, Spectrogram : 128 bins and 50% overlap

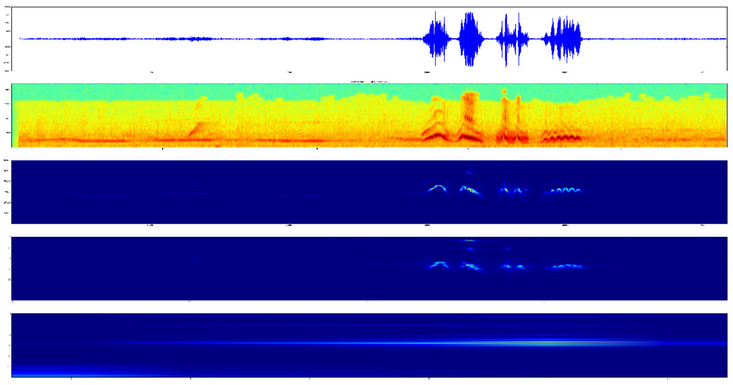


Figure 19: XC81545.wav with T1:4 Q1:20 J1: 5, T2:256 Q2:1 J2:1, T3:128 Q3:1 J3:1, Spectrogram : 128 bins and 50% overlap

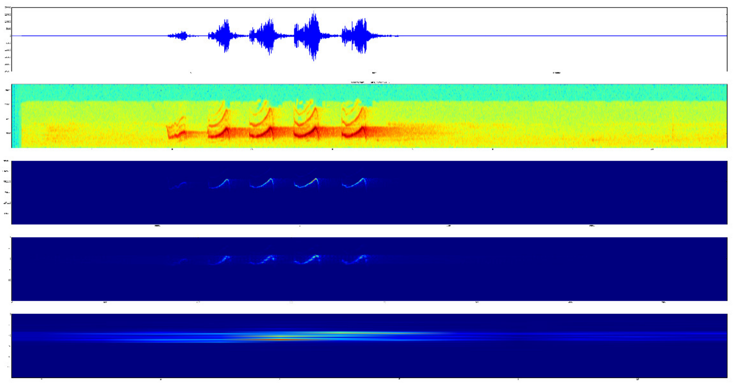


Figure 20: XC80923.wav with T1:4 Q1:20 J1: 5, T2:256 Q2:1 J2:1, T3:128 Q3:1 J3:1, Spectrogram : 128 bins and 50% overlap

8 Future work

Concerning computation time for the Scattering Network, here is a plot for each signal length (power of 2 from 2^8 to 2^{16}) with two layers. The asymptotic complexity is $O(n\log(n))$ for the scattering layer although here with the chosen coefficients we can't yet see the curve.

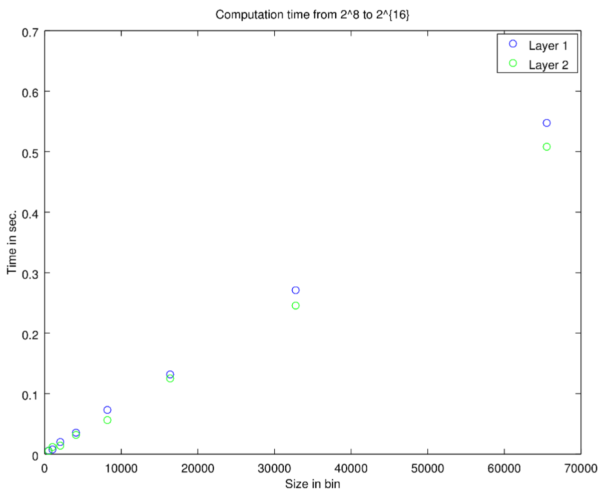


Figure 21: First Layer Coefficients : T=2,Q=8,J=8. Second Layer Coefficients :T=16,Q=1,J=1

With this toolbox, it will be possible to perform deep analysis on the massive

signal datasets taking advantage of the speed benefits of this implementation. More and more challenges are available such as the BIRDLIFE CLEF 2014 challenge[4] offering a huge dataset.
This toolbox shall open new investigations into large scale databases, with complex and not yet well known sources, including multipath propagation. It may be useful for bioacoustic tropical forest databases or cetacean real time survey.
The FFT3D algorithm will be added to this toolbox having the advantage of containing phse information which seems to be an interesting information for good representation.
Extension to GPU processing is one of the possible future work which could fully exploit the architecture of the presented algorithms.

Acknowledgements

We thank SABIOD MI MASTODONS CNRS which supported some analysis of bioacoustic corpus which conducted to scale these signal decompositions. We thank Marie Trone for her continuous interest on our signal processing, and for sharing the INIA database. We also thank Vincent LOSTANLEN and Stephane MALLAT for their contributions.

A README

Description of the folders :

- txt is where the results of the executable programs should be stored
- source contains all the algorithms and utilities
- exec contains code that use algorithms from source in order to make a usable command line executable for
  - scattering
  - spectrogram
  - imshow
- wav is a folder where the signals are stored, in this version some samples are already present for testing

The Makefile is used without any arguments, it will create all the executables (3) which can then be used as follow

SPECTROGRAM:
default arguments
./spectrogram wav/slow.wav //this will generate the file
./txt/spectrogram.txt which can be displayed as follow
specify the window size
./spectrogram wav/slow.wav 256
specify the window size plus the overlapping

[4] `http://sabiod.univ-tln.fr/public_data/BIRD_CHALLENGE/`

./spectrogram wav/slow.wav 256 0.2

IMSHOW:
keeping original ratio
./imshow txt/spectrogram.txt
this utility acts like imagesc in matlab and can be used on any .txt storing a plain matrix
square ratio
./imshow 0 txt/spectrogram.txt

SCATTERING:
arguments are in order : wav file, T1, Q1, J1, T2, Q2, J2,...
the number of layers is determined by the number of args given. Doing the following command will save the layers in txt files into the txt directory
./scattering 8 20 4 8 1 1 32 1 1 wav/processed.wav

DISPLAYING THE SCATTERING
In order to display the wav plus all the scattering layers, a simple python file is used for convenience and is used by simply calling it and passing the wav on which the scattering has been done
python exec/Plots.py wav/processed.wav

B Licence GPL

to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

   A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
   (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
   (c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
   (d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

   (a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
   (b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written

offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

(c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

(d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

(e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

   "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

   When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

   Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   (a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
   (b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

References

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

[2] Randall Balestriero. Scattering for bioacoustics. Internship research report, Univ. Toulon, 2013-14 supervised by H. Glotin.

[3] Randall Balestriero and Herve Glotin. Humpback whale song representation. NIPS4B 2013.

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

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

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

[7] 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.

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

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

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

[11] 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.

[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] Trone, Balestriero, and Glotin. Gabor scalogram reveals formants in high-frequency dolphin clicks. Proc. of Neural Information Processing Scaled for Bioacoustics: from Neurons to Big Data, 2013, Ed. Glotin H., LeCun Y., Artieres T., Mallat S., Tchernichovski O., Halkias X., joint to NIPS Conf., http://sabiod.org/publications.html, ISSN 979-10-90821-04-0, 2013.

[15] Marie Trone, Herve Glotin, Randall Balestriero, and Bonnett E David. All clicks are not created equally: Variations in high-frequency acoustic signal parameters of the amazon river dolphin (inia geoffrensis). in The Journal of the Acoustical Society of America, Volume 136, Issue 4, short letter, Oct 2014, long paper in preparation., 2014.

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

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