Année Universitaire 2021 - 2022
SEATECH Ecole d'Ingénieur

Transformée en ondelette et classification de biosonars : Le régime amorti et son importance

OLIVER Loïc DOUTÉ Robin

Enseignant référent : GLOTIN Hervé, Directeur du Laboratoire Informatique & Systèmes de l'Université de Toulon. Responsable de la recherche en Intelligence Artificielle appliquée à la bioacoustique sous-marine.

Table des matières

1 Introduction

Le but de ce travail est de caractériser des signaux sonores de mammifères. Plus précisément, nous essayerons de déterminer la démarche optimale pour extraire le plus d'informations de ces-dits signaux. Nous en parlerons plus bas, mais voici notre principale interrogation :

Les informations du signal se trouvent-elles dans une zone réduite de l'enregistrement, ou au contraire sont-elles réparties à différents endroits et différentes fréquences ?

La réponse à cette question peut paraître évidente quand on regarde la représentation d'un signal sur la figure 1. Si l'on calcul le ratio signal / bruit, on obtiendra facilement une zone où ce rapport est maximum.

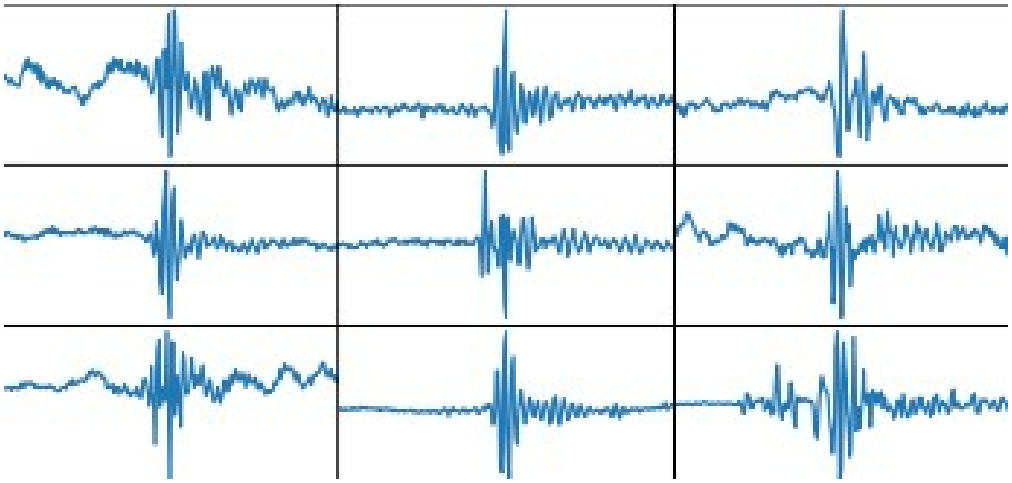


FIGURE 1 – Exemples d'enregistrements sonores - © *H.Glotin*

Voici donc le but de notre étude. Nous réaliserons différents tests, avec des paramètres différents, justifierons nos choix et expliquerons nos résultats.

2 Base de données mise à disposition

2.1 Les différentes espèces

Les données que nous allons étudier sont les données de la database DOCC10 [**FERRARI2020**]. Cette base de données contient 113 120 enregistrements de 10 espèces différentes, que voici :

— Le dauphin de Risso, ou *Grampus griseus*, noté GG dans DOCC10 ;
— Le globicéphale du Pacifique, ou *Globicephala macrorhynchus*, noté GM dans DOCC10 ;
— Le dauphin à flancs blancs, ou *Lagenorhynchus acutu*, noté LA dans DOCC10 ;
— La baleine à bec de Sowerby, ou *Lagenorhynchus acutus*, noté LA dans DOCC10 ;
— La baleine à bec de Gervais, ou *Mesoplodon europaeus*, noté ME dans DOCC10 ;
— Le grand cachalot, ou *Physeter macrocephalus*, noté PM dans DOCC10 ;
— Le dauphin tacheté de l'Atlantique, ou *Stenella sp*, noté SSP dans DOCC10 ;
— Le delphinidé type A,noté UDA dans DOCC10 ;
— Le delphinidé type B, noté UDB dans DOCC10 ;
— La baleine de Cuvier, ou *Ziphius cavirostris*, noté ZC dans DOCC10 ;

Ces 113 120 données sont équitablement réparties entre les 10 espèces (donc 11 312 enregistrements par espèce), et sont labelisées. Cela signifie qu'à chaque son est associé un des 10 mammifères. Cette base de données nous servira donc pour faire de l'apprentissage supervisé. Nous disposons aussi d'une base de données non labelisée de taille plus faible. L'objectif est donc d'identifier ces données non labelisées. Elles ont été acquises à des endroits et temps différents, cela permettra d'éprouver notre réseau de neurones.
De plus, on peut s'intéresser au format de cette base de données. Les labels sont stockés dans un fichier *Ytrain*, qui associe le numéro d'enregitrement de la donnée (commence à 20 960 et finit à 134080) et un label (0 pour UDA, etc). Le fichier *Xtrain* contient les données dans un fichier CVS. Chaque enregistrement contient 8192 valeurs, réparties sur 20ms.

FIGURE 2 – Melon de cachalot - © *David Salvatori*

2.2 Principe du biosonar

Le biosonar est un système de communication propre à plusieurs espèces de mammifères marins, dont les delphinidés. Il se trouve que chaque signal enregistré dans DOCC10 provient du biosonar d'un des mammifères présentés au dessus. Comme vous le voyez sur la figure 3, le clic émis provient d'air compressé dans le melon de l'animal, qui se relâche d'un coup et crée une véritable explosion. Cette onde sonore se propage donc à travers le crâne de l'animal, avant de se répandre dans son environnement proche.[**Vibrations**]

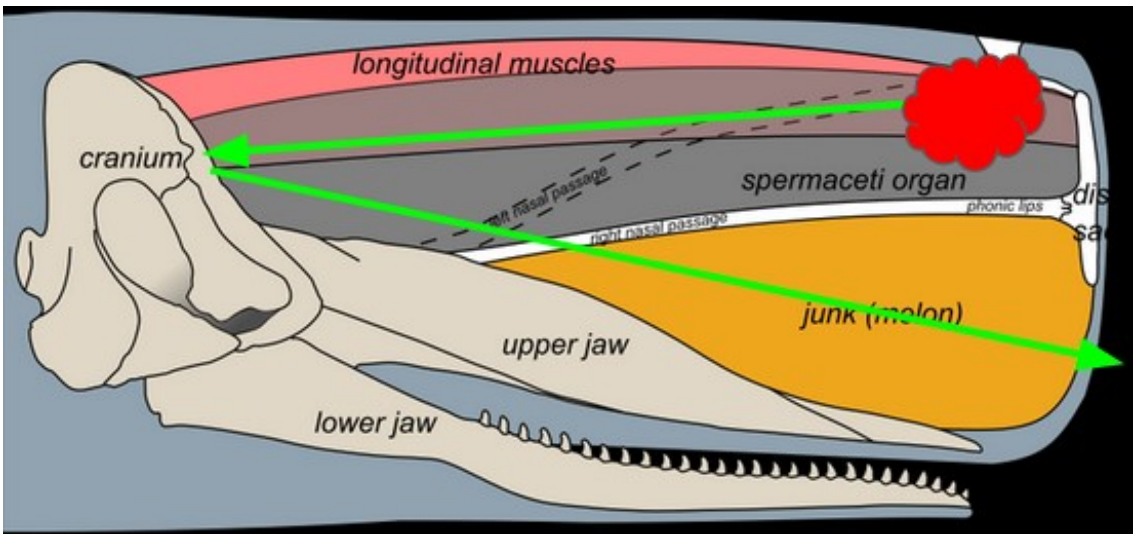


FIGURE 3 – Dessin du melon et explosion - d'après ©*TheLostCetocean*

Maintenant qu'on comprend le principe physique du biosonar, on peut s'intéresser précisément aux données enregistrées par nos hydrophones. Les 113 120 enregistrements sont répartis en 10 classes, dont on peut voir la représentation sur la figure 4. On précise d'abord que le pic du signal est centré dans la bande de 8192 samples. Avant le milieu de l'enregistrement, les oscillations à petite amplitude représentent le bruit de la mer. Puis un pic apparaît, avant d'être petit à petit atténué. Ce pic correspond à l'explosion initiale dans le melon de l'animal, et le régime amorti correspond au signal perdant de son intensité. A la fin du signal, on voit apparaître de nouveau le bruit de la mer.

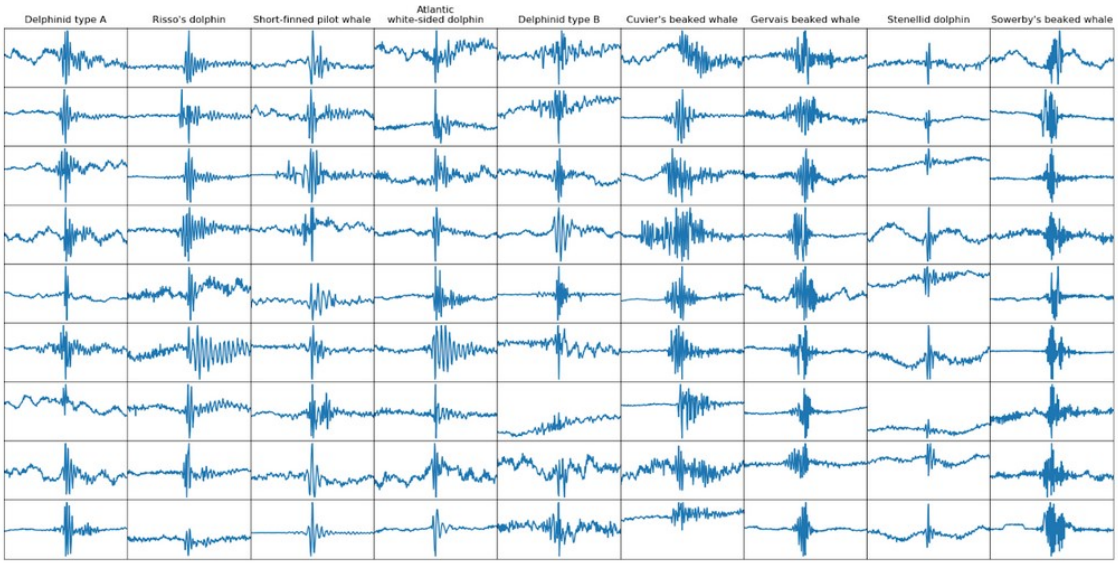


FIGURE 4 – Exemple de données de DOCC10

Le but de notre étude sera donc de déterminer si le régime amorti contient de l'information sur l'espèce.
En effet, il serait logique de penser que le pic est le seul endroit de l'enregistrement intéressant à analyser, dû à son rapport élevé signal / bruit. Ce que nous allons donc faire, c'est découper les 8192 samples en fenêtres plus petites, afin que chaque élément caractéristique du signal se trouve dans une seule fenêtre. Nous appliquerons ensuite une transformée par ondelette de Morlet, puis une classification par MLP. Nous expliquerons tout ça dans les prochaines parties.

FIGURE 5 – Impulsion et amortissement

En regardant la figure 5, on peut découper les signaux de DOCC10 en plusieurs phases. La première est le bruit de la mer, ambiant et non-interrompu. Vient ensuite l'impulsion, ici encadrée en rouge. Cette impulsion correspond à l'explosion dans le crâne du cachalot. La partie encadrée en vert est l'amortissement. Ce sont des ondes sonores ralenties ayant perdues plus d'énergie que le pic.

3 Théorie et rappels

3.1 IA et MLP

L'intelligence artificielle [**LeCun**] a pris, au cours des dernières décennies, une importance capitale dans le monde technologique en particulier, et dans notre vie quotidienne en général. Son application va de superviser des ruches d'abeilles à piloter une voiture sans l'intervention d'un homme, elle va de créer de la musique ou des peintures à surveiller les sites d'enfouissement de déchets nucléaires.

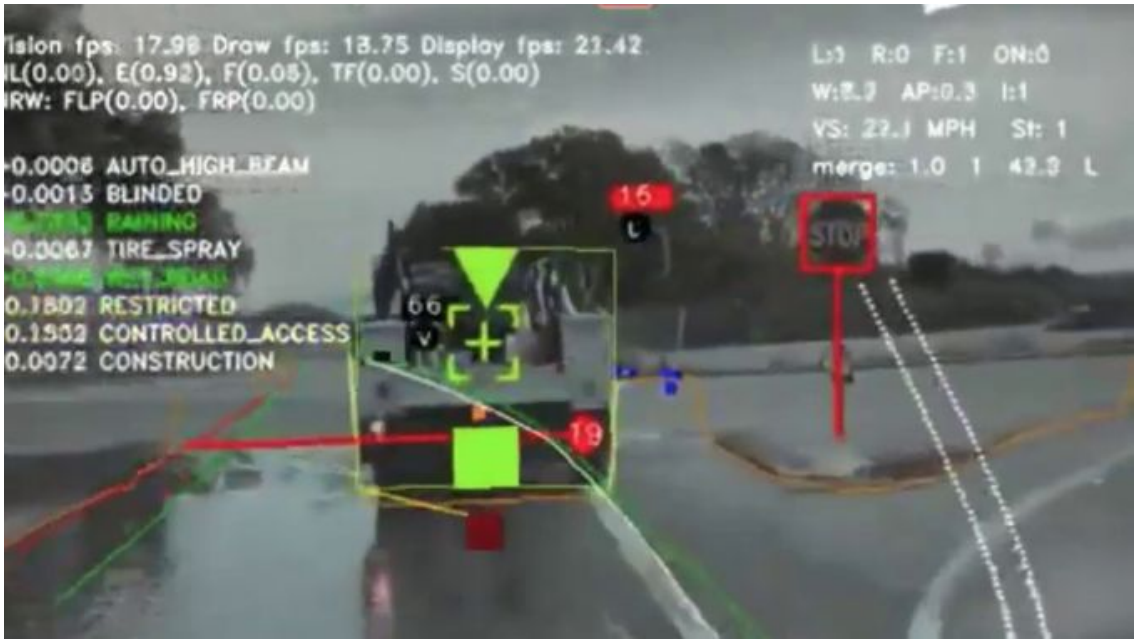
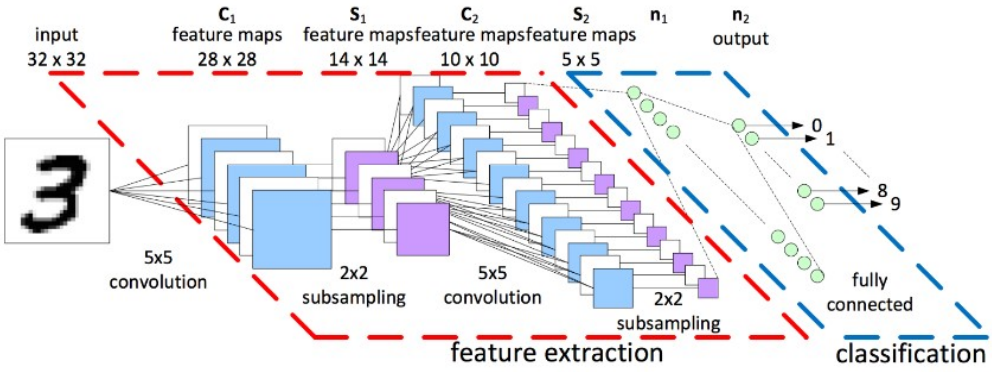


FIGURE 6 – Reconnaissance d'objets par une voiture - *©Tesla*

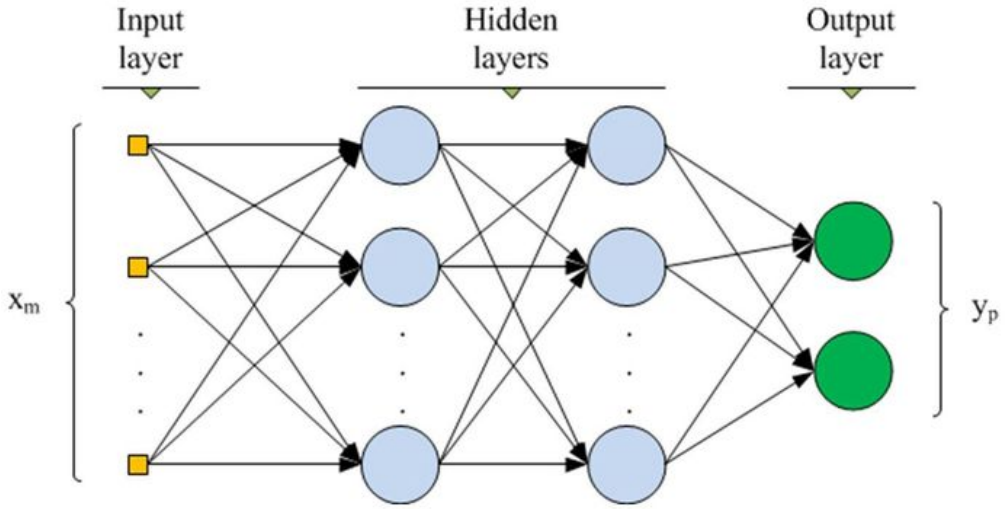
Dans notre étude, nous n'allons pas piloter une réaction de fusion ou envoyer une voiture dans l'espace, mais nous allons faire de la reconnaissance de mammifères marins. Ces différents animaux, comme le cachalot ou le dauphin commun, émettent des signaux sonores pour se localiser et pour communiquer. Le but de notre projet sera alors de récupérer ces clics (les signaux de localisation), et d'utiliser un réseau de neurones qui identifiera l'espèce émettrice du son.

Un réseau de neurones est une structure informatique reproduisant le cerveau humain. Le but de ce réseau est de traiter une grande quantité d'information en entrée, et en déduire une information en sortie. Dans le cas d'un humain, cela peut simplement être l'information à envoyer à son bras pour soulever un pot de yaourt. Dans le cas informatique, ce serait plutôt reconnaître une image, en associant le mot "poussin" à une matrices de pixels représentant un poussin.
Voici rapidement la structure d'un réseau de neurones :

Le neurone se base sur le fonctionnement d'un neurone humain : différentes variables x_1, x_2 , x_n sont envoyées en entrée d'un neurone. A ces dernières sont associées des poids w_1, w_2, w_n. Nous pouvons aussi retrouver une constante en entrée b ou bias. Ce neurone est aussi composé d'une fonction d'activation (il en existe plusieurs, ne rentrons pas dans les détails) prenant en paramètre toutes ces entrées. Le neurone peut donc être passant ou bloquant en fonction de cette fonction et de ces entrées.



(a) Exemple de CNN



(b) Exemple de classifier

FIGURE 7 – Différentes structures d'apprentissage - © *V.GIES*

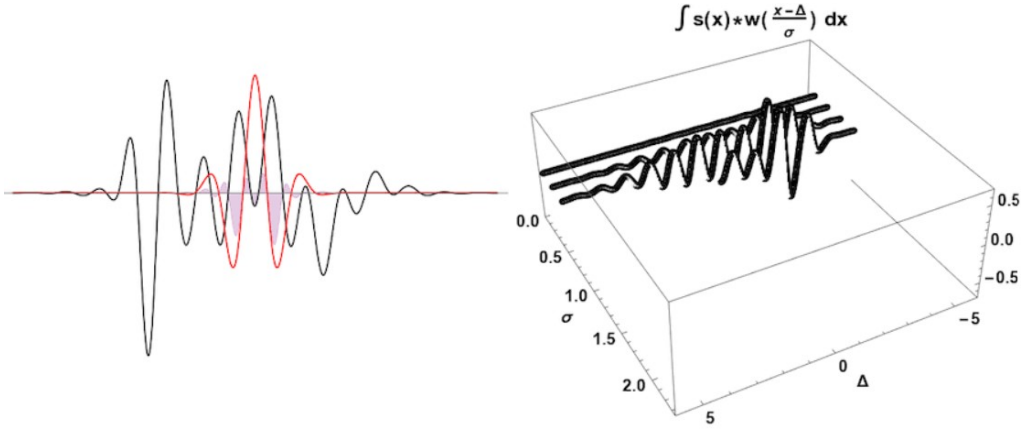
Voici à quoi pourrait ressembler un réseau de reconnaissance de chiffre. On constate que la majorité du réseau ne sert pas à classifier à proprement parler, mais à extraire des information, ou *features*, qui permettent de différentier chaque chiffre des autres. Dans le cas présenté, le but est de réduire une entrée de dimension 32x32x1 à une une sortie de dimension 1. Chaque couche de neurones a donc une rôle :

- Certaines effectuent des opération de convolution, en "déplaçant" une fenêtre sur l'entrée pour en ressortir une sortie de dimension plus faible, et représentant une caractéristique particulière de l'image ;
- Certaines récupèrent uniquement la valeur maximale dans une zone de pixels ;
- D'autres vont se charger d'exécuter un algorithme de K-means, c'est à dire une algorithme de classification.

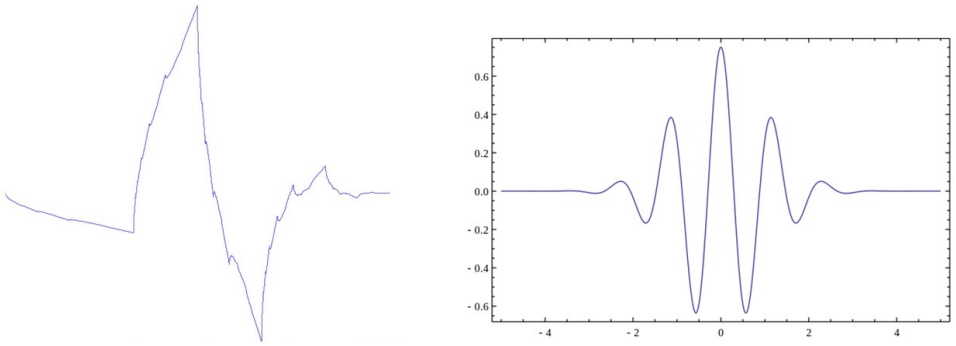
On voit donc que le choix des features est une étape importante. Nous l'aborderons plus tard dans notre rapport, mais que nous soyons dans le cas d'un classifier simple ou d'un réseau profond de neurones, ce choix des features est déterminant pour nos résultat, bien plus que le nombre ou la largeur de couches du réseau.

3.2 La transformée en ondelette

Cette transformée est une méthode utilisée dans plusieurs domaines : l'IA, la médecine, la compression de données, la résolution d'équation et même dans l'étude des glissements de terrains ! [**Mallat**]



(a) Convolution en ondelette



(b) Ondelette de Daubechies - Morlet

FIGURE 8 – La transformée en ondelette continue (CWT) - *©Jacopo Bertolotti*

Le principe est de déplacer une ondelette, créée à partir d'une porteuse et d'une enveloppe, sur notre signal d'origine. L'aire commune sous la courbe minimum entre le signal et l'ondelette est conservée. Puis on élargit l'ondelette en modifiant son facteur d'échelle et on réitère l'opération. On obtient alors une représentation qui dépend à la fois du temps (grâce à la translation sur l'axe) mais aussi en fréquence (grâce à l'évolution du facteur d'échelle).
Cette transformée est une alternative à la transformée de Fourrier dans le domaine du traitement de signal. Dans notre cas, nous avons une base de données de relevés d'hydrophones. En traçant l'évolution de ces valeurs en fonction du temps, nous pourrions visualiser l'évolution temporelle de l'énergie du son, mais la dimension fréquentielle serait alors inaccessible. En appliquant une transformée de Fourrier, nous pourrions obtenir des informations sur les fréquences fondamentale et harmoniques, ainsi que les amplitudes de l'énergie à ces fréquences. Mais dans le cas, la dimension temporelle est perdue.
La solution est alors de trouver une méthode qui rend visible les informations concernant à la fois la fréquence, mais aussi le temps. C'est pourquoi nous avons choisi la transformée en ondelette.

4 Mise en application

4.1 Composition base DOCC10

La base de donnée DOCC10 comporte 113 120 enregistrements décomposé en 10 classes de 11 312 échantillons. Chaque enregistrement est un vecteur de 8192 éléments. Le signal émis est centré sur l'enregistrement. Un enregistrement est un vecteur 1D échantillonné à 200kHz. En lui appliquant une transformation en ondelettes avec un support de Morlet de 32 canaux, nous obtenons 64 scalogrammes de dimension 32 par 128.

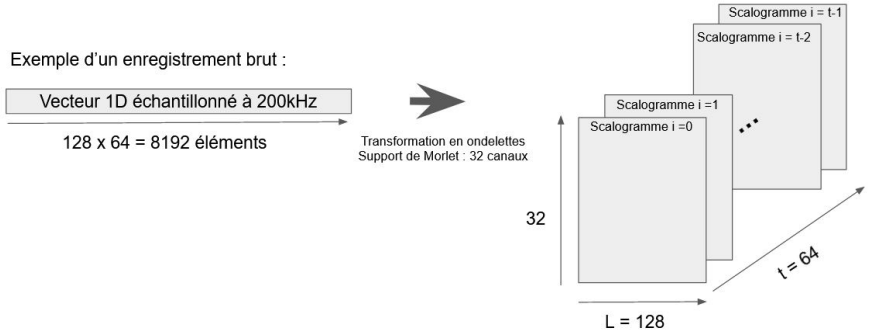


FIGURE 9 – Composition de la base de donnée

4.2 Ondelettes et scalogrammes

Comme expliqué plutôt, nous allons appliquer une transformée par ondelette à nos différents enregistrements. Avant cette étape, nous divisons la fenêtre de 8192 samples en 64 fenêtres de 128 samples. Le but est ici d'isoler l'impulsion et les amortissements. En effet, on ne souhaite pas moyenner les caractéristiques de chacun des phénomènes.
On convolue donc nos 64 fenêtres par l'ondelette de Morlet (*tous les codes se trouvent en annexe*). On obtient alors des figures appelées scalogrammes. Ce sont des représentations temps-quéfrence de nos signaux. Plus la couleur est claire, plus l'énergie du signal est élevée. On peut aussi tracer l'énergie du signal en fonction du temps, en appliquant un passe-haut pour supprimer le bruit de la mer. On obtient alors la figure 10.

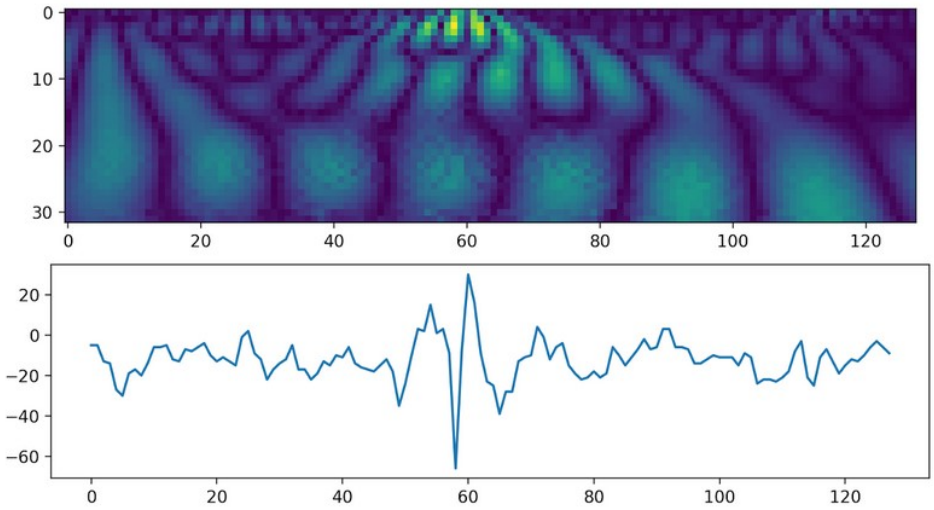


FIGURE 10 – Scalogramme et énergie associée

Le scalogramme a une forme particulière : on constate des zones d'énergie qui s'étirent vers le haut de la figure. Cela est du au passage temps-fréquence de la transformée en ondelette. Nous ne rentrerons pas dans les détails, mais ce passage est différent de celui de la transformée de Fourrier.
Toutefois, un problème apparaît au moment de la convolution : les effets de bord. En effet, ces derniers apparaissent aux extrémités du spectres, et plus particulièrement dans les angles supérieurs gauche et droit. Ces zones sont remplies de fausses données, il faudrait dans l'idéal ne pas les considérer dans la suite de l'étude.

4.3 Région d'intérêt et extraction des traits

Afin de faire une bonne classification, nous devons choisir des paramètres qui nous semblent judicieux et discriminants entre espèces pour les détecter. Nous devons faire un compromis sur la taille de ces informations afin de sélectionner le critère qui est le plus important.
Nous avons fait le choix de découper l'image en 6 régions d'intérêts (ROI) et de calculer sur chacune d'elles les moments d'ordres 1, 2 et 4. A ces informations, on y ajoute la quéfrence de l'énergie maximale du scalogramme qui nous semblait être un éléments important.

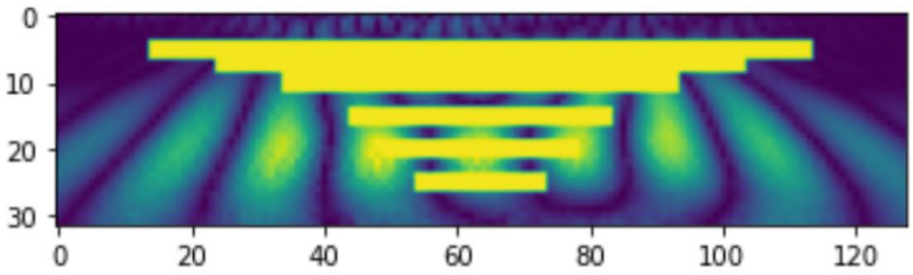


FIGURE 11 – Régions d'intérêts du scalogramme

Comme dit précédemment, nous connaissons la date à partir de laquelle le signal débute. Nous pouvons donc à partir de ce scalogramme correspondant, récupérer plus ou moins d'images consécutives pour vérifier le caractère discriminant de l'amortissement du signal émis par le mammifère. Nous récupérerons des informations du premier jusqu'au cinquième scalogramme. Sur chaque scalogramme, nous récupérons 3 moments sur chaque régions d'intérêts et on ajoute une quéfrence. Nous extrayons donc 19 traits au maximum sur chaque scalogramme.

4.4 La réduction de dimensions

4.4.1 La malédiction des dimensions

Nous avions parlé plus haut de dimensions discriminantes, cela implique donc des dimensions non ou peu discriminantes. La question qu'on peut alors se poser est la suivante :
Pourquoi ne pouvons-nous pas conserver les dimensions peu discriminantes ?
En effet, l'interrogation est pertinente, mais la réponse est contre-intuitive : il ne faut pas garder toutes les dimensions. Ce problème se nomme *la malédiction des dimensions*. Elle stipule que la précision d'un classifieur augmentera quand on rajoute des dimensions, puis diminuera passée une certaines valeur. Pour être plus précis, la norme entre les différents éléments d'un espace tend vers 0 quand le nombre de dimensions tend vers ∞ . Cette norme cessera alors de porter une information quand le nombre de dimensions augmente. La figure 11 représente ce phénomène :

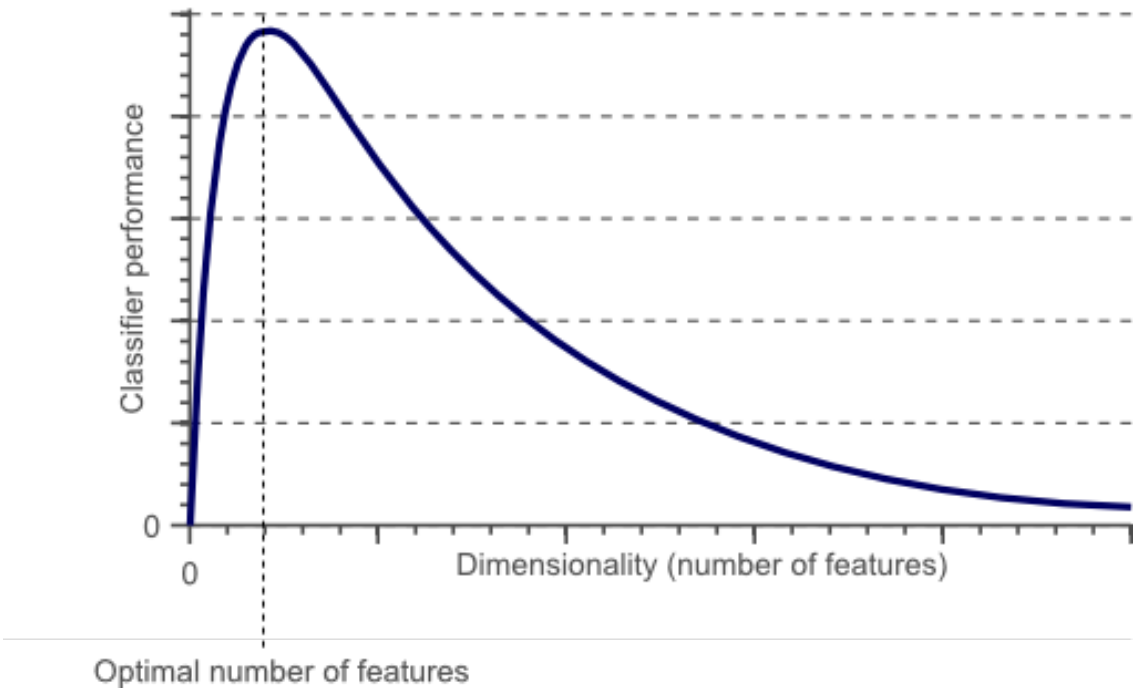


FIGURE 12 – La malédiction des dimensions

4.4.2 Analyse en composante principale

L'analyse en composantes principales (ACP) permet de transformer des variables corrélées en variables décorrélées baptisée "composantes principales". Plus précisément, cette méthode vise à réduire le nombre de variables appliquées à des individus, pour simplifier les observations tout en conservant un maximum d'informations. Seules une, deux ou trois variables dites "composantes principales" sont conservées en général. Lors du premier passage dans cette fonction, nous ne savons pas combien de dimensions sont à garder, nous les gardons toutes et nous affichons la somme cumulée des coefficients triés de chaque composantes.

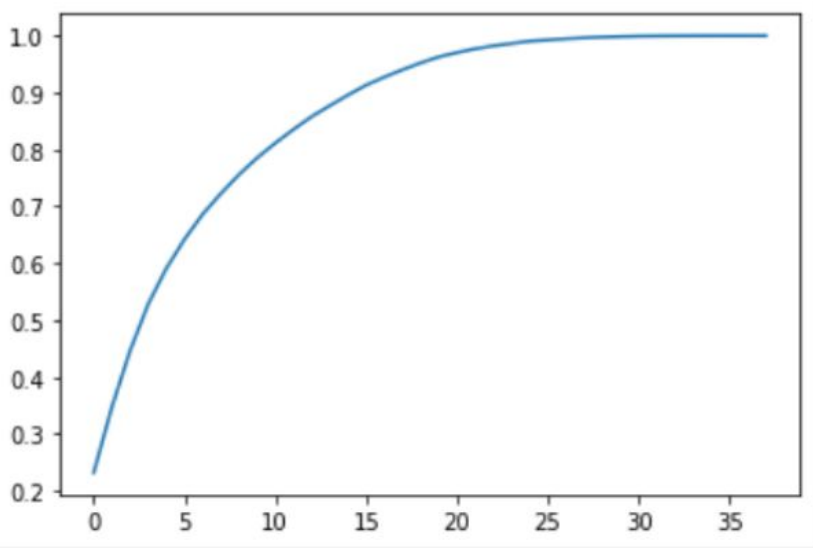


FIGURE 13 – Pourcentage cumulé de la variance expliquée pour chaque composantes d'entrées

Ce graphique nous sert à déterminer le nombre de composantes principales par un calcul de dérivée seconde. En effet, la fonction est fortement croissante au début puis tend vers 1 rapidement. La dérivée seconde permet de connaître le nombre de dimension nécessaires pour avoir un maximum d'information tout en étant précis.

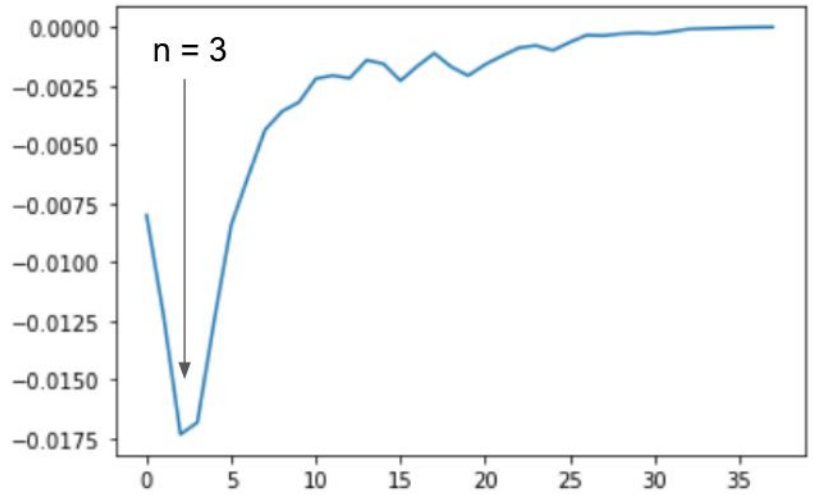


FIGURE 14 – Calcul du nombre de dimensions principales (technique du coude)

Dans notre cas, nous observons le minimum de la fonction pour n égale à 2 ce qui correspond à n+1 composantes principales.

4.5 Apprentissage par le MLP

L'objectif de notre projet n'était pas d'avoir un réseau performant, l'objectif était de rechercher et d'extraire des traits qui étaient discriminants afin de classifier nos espèces. Nous avons donc utilisé un simple réseau de neurones du type perceptron multicouche (multilayer perceptron MLP). C'est un réseau neuronal artificiel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement.

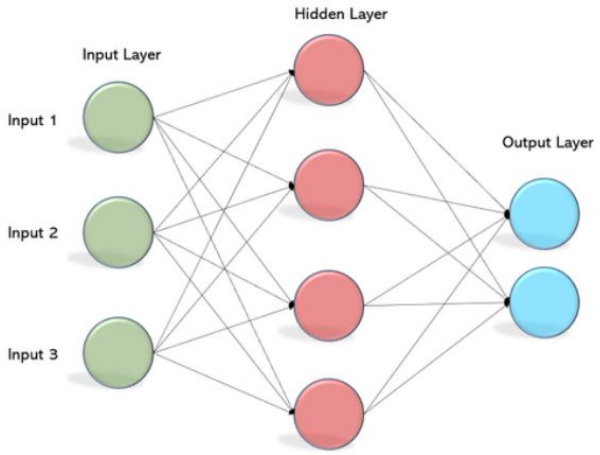


FIGURE 15 – Réseau de neurone artificiel - *©Awhan Mohanty*

Sa mise en place était plutôt simple et avons eu rapidement des résultats. Nous avons déclaré la forme du réseau avec les couches cachées, le nombre d'itération maximum que le réseau devait effectuer pour l'apprentissage. Il existe un paramètre par défaut qui permet de stopper l'entraînement avant ce nombre d'itérations si l'a précision convergeait. L'entraînement se stoppait en moyenne autour des 400 itérations.
Le paramètre *relu* permet de fixer la fonction d'activation. Cette fonction est très couramment utilisée. Acronyme de Rectified Linear Unit (unité linéaire rectifiée), elle permet tout simplement de remplacer les résultats négatifs par zéro.
Nous fixons ensuite le solver qui a pour but d'optimiser la solution. Il va comparer les prédictions aux valeurs attendues, cela donne la fonction d'erreur. Un algorithme dit *optimizer* joue le rôle de corriger les poids du perceptrons pour réduire cette erreur. Le but de l'optimizer n'est pas d'avoir une erreur nulle pour un exemple mais d'avoir l'erreur la plus faible sur tous les exemples. Nous avons fais le choix d'utiliser *Adam*. Il s'agit de l'optimizer le plus utilisé en raison de son efficacité et de sa stabilité.

```
classifier = MLPClassifier(
    hidden_layer_sizes=(128, 128, 90, 10),
    max_iter=5000,
    activation = 'relu',
    solver='adam',
    random_state=1,
    verbose=True)
```

FIGURE 16 – Déclaration du classifier

5 Résultats et analyse

Nous avons extraits différents traits dans chaque scalogramme et avons essayé de visualiser l'effet discriminant de l'amortissement du signal. Autrement dit, nous prenons plusieurs fenêtres à la suite du pic d'énergie. En fonction du nombre de scalogrammes, nous regardons l'évolution de la précision. Voici un tableau récapitulatif des résultats de nos expériences.

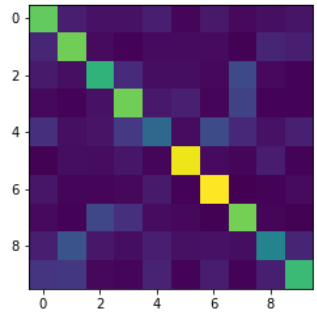
Features \ Nombre de fenêtres	1	2	3	4	5
Means	40.1	44.0	46.2	46.9	47.3
Means + Var	46,7	50.5	51.8	52.4	54.4 *
Means + Var+ Kurtosis	44.0	-	-	-	52.0
PCA(M+V+K)	21.6	-	-	-	22.4

FIGURE 17 – Tableau des accuracy pour chaque expérience

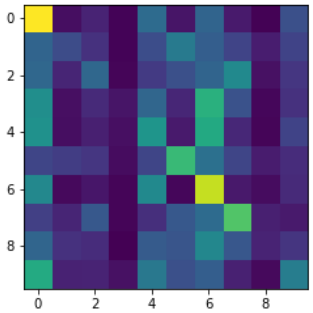
Dans ce tableau, nous observons en abscisse le nombre de fenêtres prises en compte et en ordonnée le type de trait.

Dans un premier temps, nous remarquons que la seconde ligne avec les informations *Moyennes + Variances* ont des scores plus élevés. On peut en déduire que rajouter des informations telles que le *Kurtosis* n'est pas judicieux puisque détériore le score.

On peut voir ci-dessous des matrices de confusion. L'abscisse est la classe prédite, et l'ordonnée est la classe réelle. Le but est donc d'avoir un maximum de valeurs sur la diagonale, et peu autre part. On voit donc que la matrice associée au test de gauche est plutôt concluant, alors que le test associée à celle de droite ne l'est pas du tout !



(a) Résultat concluant, Means+Var sur 5 fenêtres



(b) résultat non concluant, PCA sur 5 fenêtres

FIGURE 18 – Matrices de confusion

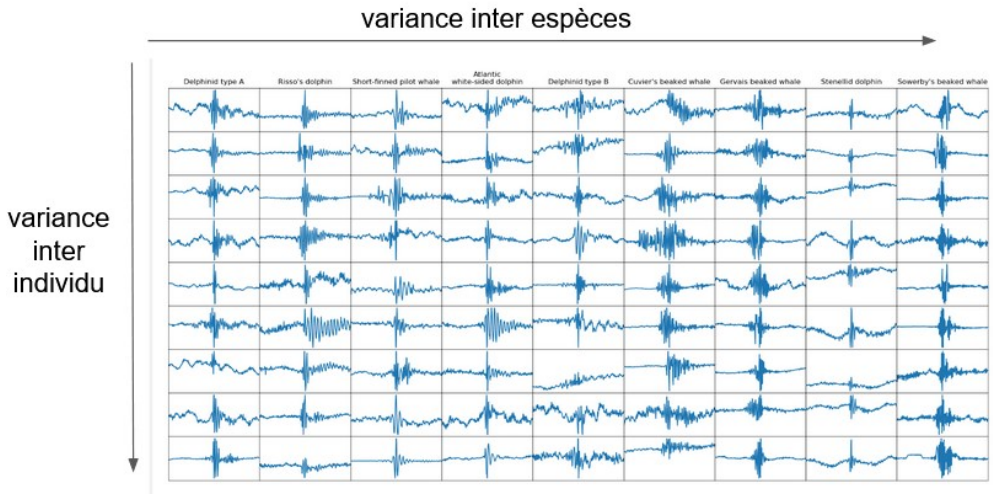


FIGURE 19 – Variance inter-espèce

Deuxièmement, on observe que la PCA n'est pas du tout efficace pour cette extraction de traits. En effet, on constate que la précision du modèle est deux voire trois fois plus faible que les autres résultats. Cela peut s'expliquer quand on regarde la figure 19. On constate que les signaux sont relativement différents au sein d'une même espèce. Or, la PCA est un algorithme non supervisée, c'est à dire un algorithme qui ne prend pas en compte les classes pendant la création des nouvelles dimensions. On peut donc supposer que la PCA prend plus en compte la variance inter-individu que la variance inter-espèce.

Cependant, ce qu'il ressort principalement de ce tableau c'est l'évolution des valeurs situées sur la même ligne. Peu importe le trait choisi, la valeur de précision augmente avec le nombre de fenêtre. C'est à dire que l'information située quelques fenêtres après le pic d'énergie permet de déterminer l'espèce. Nous en déduisons que l'amortissement du signal joue un rôle dans la classification de ces mammifères marins.

6 Conclusion

Cette étude nous a permis, sans chercher une performance absolue d'un classifieur, d'utiliser l'IA pour décomposer et mesure l'information engendré par un phénomène physique (ici l'émission de biosonar). A l'issue de cette étude, on constate plusieurs choses :

— **L'amortissement est effectivement déterminant dans la précision de la classification. On pourrait penser que cette zone du signal est inexploitable à cause du rapport signal / bruit trop faible, mais nos résultats ont montré que conserver les 4 fenêtres suivant l'impulsion améliorait la précision ;**

— Le *Multi Layer Perceptron* est efficace sur des scalogrammes. Les résultats montrent qu'un simple classifier suffit pour notre étude. Ca contrairement à un CNN, notre MLP ne fait pas de convolution, qui est effectuée pendant la transformée en ondelette ;

— Aller chercher les moments d'ordre 4 n'est peut être pas une solution. En regardant les composantes principales de l'ACP, on remarque que ce sont les variances, moments d'ordre 2, qui sont le plus déterminant. Cela dit, peut être que le moment d'ordre 10 est discriminant, mais cela parait peu probable ;

— *L'analyse en composantes principales* n'est pas pertinente et fonctionne même plutôt mal. On a vu que c'était sans doute dû à la variance inter-espèce et le caractère non supervisé de la méthode. Pour observer une amélioration, il faudrait utiliser *une analyse linéaire discriminante* (LDA) qui, elle, est supervisée et prend en compte les différentes classes. Toutefois, une réduction des dimensions paraît indiquée. La LDA pourrait donc nous éviter de subir la malédiction des dimensions ;

— Notre code est plutôt robuste et rapide. On traite en effet 113 120 enregistrements (4 Go), et après la transformée en ondelette, ce sont 350 Go de données qui apparaissent sur le disque dur !

— Enfin, on pourrait changer d'ondelette, et sélectionner par exemple l'ondelette de Daubechies, qui n'est pas symétrique. L'ondelette de Morlet n'est peut être pas la mieux adaptée pour traiter un signal qui n'est évidemment pas symétrique.

Références

[1] Maxence Ferrari, Hervé Glotin, Ricard Marxer, Mark Asch. DOCC10 : Open access dataset of marine mammal transient studies and end-to-end CNN classification. IJCNN, Jul 2020, Glasgow, United Kingdom.

[2] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." Nature 521.7553 (2015) : 436-444.

[3] Stéphane Mallat. "A wavelet tour of signal processing."1998

[4] F. Bosia1 , V. Dal Poggetto2 , A. S. Gliozzi1 , G. Greco2 , M. Lott1 , M. Miniaci3 , F. Ongaro2 , M. Onorato4 , S.F. Seyyedizadeh1,4 , M. Tortello1 , N. M. Pugno2*. "Optimized structures for vibration attenuation and sound control in Nature : a review." Université de Turin, 2020

7 Annexe

7.1 Décomposition en ondelette et calcul des features

```
import pywt
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import kurtosis
from scipy.io import wavfile
import scipy.io
from tqdm import tqdm
import pandas as pd

##### Paths
path = 'DOCC10_train/'
path_results = 'results_DOCC10/'  #R pertoire d'enregistrement des r sultats

### Lecture des labels
labels = pd.read_csv(path+'DOCC10_Ytrain_IDS.csv')

sons = np.load(path+'DOCC10_Xtrain.npy')

imaxE = np.load(path_results+'imax_Etrain.npy')
mean_imax = round(np.mean(imaxE))

### Param tre     modifier
# index du WAV     traiter
# Dans DOCC10_Xtrain.npy [20960 : 134079]
# Dans DOCC10_Xtest.npy [0 : 20959]
wav = 20960

#########################################################################################
# DECOUPE DES ROI (6 r gions), CALCUL MEAN, VAR, STD SUR 5 FENETRES
# 6ROI x 5FENETRES x 3MOMENTS = 90 features par enregistrement
# 90 features x 123120
Features = []
Nb_fenetres = 1 # Nb de fenetre que l'on prend apr s notre pic d' nergie
l1, l2, l3, l4, l5, l6 = 5, 7, 10, 15, 20, 25        # Indice ligne ROI
L1, L2, L3, L4, L5, L6 = 100, 80, 60, 40, 30, 20     # Longueur ligne ROI
c1 = int(64-L1/2)                                    # Indice colone ROI
c2 = int(64-L2/2)
c3 = int(64-L3/2)
c4 = int(64-L4/2)
c5 = int(64-L5/2)
c6 = int(64-L6/2)

for k in tqdm(range(sons.shape[0])):
    fs = 200000
    son = sons[k]
    L = 128 # sur 0.5 ms de signal
    scales = np.arange(4,36,1) # 32 canaux de scalogramme (support morlet )
    ROI1, ROI2, ROI3, ROI4, ROI5, ROI6 = [],[],[],[],[],[]
```

```
    # Extraction des labels
    label = labels['TARGET'][labels['ID']==wav].values[0]
    if label == 'GG':
        features = [0]
    elif label =='GMA':
        features = [1]
    elif label =='LA':
        features = [2]
    elif label =='MB':
        features = [3]
    elif label =='ME':
        features = [4]
    elif label =='PM':
        features = [5]
    elif label =='SSP':
        features = [6]
    elif label =='UDA':
        features = [7]
    elif label =='UDB':
        features = [8]
    else:
        features = [9]

    # Condition obligatoire sinon sera en dehors du vecteur
    if imaxE[k] > (64-Nb_fenetres):
        imaxE[k] = mean_imax

    for i in range(imaxE[k]*L, (imaxE[k]+Nb_fenetres)*L, L):
        # Parcours les Nb_fenetres fenetres suivantes le pic d' nergie
        coef, freq = pywt.cwt(np.hanning(L) * son[i:i+L], scales, 'morl')
        features.append(coef)

        # R cup ration de la quefrance pour l'amplitude max
        index_max_value = int(np.argmax(coef)/L)
        features.append(index_max_value)

        # CALCUL 1ERE ROI : MEAN, VAR, STD
        ROI1 = np.append(ROI1, coef[l1-1, c1:c1+L1])
        ROI1 = np.append(ROI1, coef[l1, c1:c1+L1])
        ROI1 = np.append(ROI1, coef[l1+1, c1:c1+L1])
        features.append(np.average(ROI1))
        features.append(np.var(ROI1))
        features.append(np.std(ROI1))
        features.append(kurtosis(ROI1))

    # ADD AU VECTEUR Features
    Features.append(features)
    wav += 1

np.save(path_results+'Features.npy', Features)
```

7.2 ACP

```
### Ouverture du Dataset
training_set = np.load(path_results + 'Features/' + 'Features_O.npy')
training_set = pd.DataFrame(training_set)

nb_elements_features = training_set.shape[1]

# Classifying the predictors and target variables as X and Y
X_train = training_set.iloc[:,1:nb_elements_features].values
Y_train = training_set.iloc[:,0].values

# Ici on s pare nos X et Y en 2 bases test et train
X_train, X_test, Y_train, Y_test = train_test_split(X_train, Y_train,
                                                    test_size=0.2,
                                                    random_state=0)
### Feature Scaling
sc = StandardScaler()                    # Instancie un objet StandardScaler
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)

##############################################################################################
# On importe la fonction ACP (en anglais PCA, principal component analysis)
#pca = PCA(n_components = nb_elements_features-1)
pca = PCA() # Par d faut, prend tous les vecteurs
pca.fit_transform(X_train)
pca.fit_transform(X_test)

# Recuperation coef puis afficher la cumsum pour d terminer le nb d'elements a garder
ratios = pca.explained_variance_ratio_
# Calcul cassure du coude (d riv e seconde)
derS = np.gradient(np.gradient(np.cumsum(ratios)))
nb_elements_principaux = np.argmin(derS) + 1

##############################################################################################
# Calcul nouvelle matrice de features avec elements principaux
pca = PCA(n_components = 2)#nb_elements_principaux)
pca.fit_transform(X_train)
pca.fit_transform(X_test)

components = pca.components_

# Matrice de passage de l'ancienne matrice    la nouvelle
mat_pass_train = pca.transform(X_train.data)
mat_pass_test = pca.transform(X_test.data)
np.save(path_results + 'Features_PCA_train_T.npy', mat_pass_train)
np.save(path_results + 'Features_PCA_test_T.npy', mat_pass_test)
np.save(path_results + 'Y_test_PCA_T.npy', Y_test)
np.save(path_results + 'Y_train_PCA_T.npy', Y_train)
```

7.3 MLP

```
#########################################################################################################
### Measuring the Accuracy
def accuracy(confusion_matrix):
    diagonal_sum = confusion_matrix.trace()
    sum_of_all_elements = confusion_matrix.sum()
    return diagonal_sum / sum_of_all_elements

### Building the MLPClassifier
#Initializing the MLPClassifier
classifier = MLPClassifier(hidden_layer_sizes = (128, 128, 90, 10),
                                                    max_iter=5000,
                                                    activation = 'relu',
                                                    solver='adam',
                                                    random_state=1,
                                                    verbose=True)

    # hidden_layer_sizes : This parameter allows us to set the number of layers
    # and the number of nodes
    # max_iter: It denotes the number of epochs
    # activation: The activation function for the hidden layers
    # solver: This parameter specifies the algorithm for weight optimization
    # across the nodes
    # random_state: The parameter allows to set a seed for reproducing the same results
    # verbose: gives progress (and sometimes an indication on the rate of convergence)

#Fitting the training data to the network
classifier.fit(X_train, Y_train)
classifier.save("my.h5")

#Predicting y for X_test
Y_pred = classifier.predict(X_test)

#Printing the accuracy
cm = confusion_matrix(Y_test, Y_pred)
print("Accuracy of MLPClassifier :", accuracy(cm))
loss_values = classifier.loss_curve_
```

biblio