UNIVERSITÉ DE TOULON

UE73
TP 1
Construction du dataset

Enseignant : Chareyre Adam

Master 1 Informatique | Université de Toulon | 27 novembre 2025

Résumé

L'objectif de cette séance est de préparer et construire, en Python, un jeu de données complet et exploitable pour un futur modèle de détection basé sur une architecture de type *Transformer Encoder*.

Pour cela, vous allez :

— générer des spectrogrammes (linéaires ou mel) à partir de fichiers audio ;
— lire et exploiter un fichier d'annotations `.csv` ;
— générer automatiquement des exemples négatifs ;
— intégrer des techniques simples d'augmentation de données ;
— agréger et mélanger l'ensemble des exemples afin de produire les fichiers finaux (`X` et `Y`) qui constitueront votre dataset.

Chapitre 1

Questions

1.1 Question 1 – Construction d'un spectrogramme

Pour la création du dataset, la première étape consiste à charger un fichier audio de 5 minutes et à en calculer un spectrogramme (linéaire ou mel) à l'aide du module Python `librosa`.

Le module `librosa` (comme la plupart des modules Python) dispose d'une documentation détaillée, incluant des exemples de code pour la génération de spectrogrammes. Vous êtes encouragés à la consulter.

1. Écrivez une fonction `load_signal` qui charge le fichier audio `ValabFr-20221128_133055.735.flac` et renvoie :
   — le signal audio sous forme de tableau NumPy ;
   — la fréquence d'échantillonnage.
2. Écrivez ensuite une fonction `get_spectro` qui, à partir du signal et de la fréquence d'échantillonnage, renvoie la matrice correspondant au spectrogramme calculé (linéaire ou mel, à votre choix).
3. Affichez le spectrogramme obtenu en utilisant le module `matplotlib`. Pour cela, écrivez la fonction `show_spectro` qui prend en entrée la matrice du spectrogramme (et éventuellement l'axe temps/fréquence) et en produit une visualisation.

La visualisation du spectrogramme vous permettra de choisir des paramètres de construction cohérents (taille de fenêtre, recouvrement, bande de fréquences, etc.) afin de faciliter la convergence du modèle de détection ultérieur.

1.2 † Question 2 – Centrage du spectrogramme

Pour l'instant, votre programme génère un spectrogramme pour la totalité du signal (les 5 minutes).

Cependant, pour une tâche de *détection de pulses*, il n'est pas pertinent de travailler sur l'intégralité du signal en une fois (présence de plusieurs pulses, bruit, fausses détections, etc.). Nous allons plutôt travailler sur des *fenêtres temporelles* du signal.

On sait qu'un pulse de rorqual commun :
— dure environ 1 seconde ;
— se situe approximativement autour de 125 Hz.

1. Proposez une taille de fenêtre définie :
   — par une durée (en secondes) ;
   — par une bande de fréquences (en Hz),
   de manière à obtenir un spectrogramme mieux adapté à la détection de ces pulses.
2. Adaptez vos fonctions `load_signal` et `get_spectro` pour qu'elles appliquent cette fenêtre :
   — ne gardez qu'une portion temporelle de la durée choisie ;
   — ne conservez, dans le spectrogramme, que la bande de fréquences pertinente.

1.3 Question 3 – Le fichier `detections.csv`

Jusqu'ici, vous centriez votre fenêtre *à la main*. Vous disposez maintenant d'un fichier `detections.csv` qui indique, pour chaque fichier audio, les instants où se situent les pulses de rorquals (en secondes).

L'objectif est d'exploiter ce fichier pour placer **automatiquement** le premier pulse du fichier audio `ValabFr-20221128_133055.735.flac` au centre de votre fenêtre de spectrogramme.

1. Chargez le fichier `detections.csv` à l'aide de la bibliothèque Python `pandas`.
2. Identifiez, dans ce fichier, la première annotation correspondant au fichier `ValabFr-20221128_133055.73`
3. Modifiez votre code pour que le premier pulse soit automatiquement centré sur dans la fenêtre temporelle, et générez le spectrogramme correspondant.

1.4 † Question 4 – Décalage temporel aléatoire

Vous avez maintenant un pulse centré dans une fenêtre temporelle.

On souhaite à présent écrire une fonction `shift_signal` qui réalise la même opération que précédemment (extraction d'une fenêtre contenant le pulse), mais cette fois-ci, le pulse ne doit plus être au centre de la fenêtre : il doit se trouver à une position *aléatoire* dans la fenêtre.

1. Programmez la fonction `shift_signal` qui, à partir de l'instant du pulse, de sa durée, de la durée de la fenêtre et du signal total, génère sélectionne une partie du signal qui contiendra à une position aléatoire le pulse correspondant (en s'assurant que la totalité du pulse reste contenue dans la fenêtre).
2. Utilisez cette fonction pour générer le spectrogramme où le pulse est décentré.
3. Répondez brièvement : selon vous, est-il utile de réaliser ce décentrage aléatoire du pulse ? Quels avantages cela peut-il apporter lors de l'entraînement d'un modèle de détection ?

1.5 † Question 5 – Agrégation des pulses (positifs)

Vous êtes maintenant en mesure de générer un spectrogramme pour un pulse donné.

1. Écrivez la fonction `get_spectros_pos_one_file` qui :
   — prend en entrée le nom d'un fichier audio (par exemple `ValabFr-20221128_133055.735.flac`) et les annotations correspondantes dans `detections.csv` ;
   — renvoie, sous la forme d'un tableau NumPy (aux dimensions `(N, F, T)` : N pulses, F fréquences, T pas de temps), la matrice de spectrogramme pour chaque annotation positive de ce fichier.

   Vous devez, pour chaque pulse, utiliser la fonction qui réalise le décalage temporel aléatoire (`shift_signal`).
2. Écrivez ensuite la fonction `get_all_spectros_pos` qui :
   — parcourt l'ensemble des fichiers audio mentionnés dans `detections.csv` ;
   — collecte tous les spectrogrammes positifs générés ;
   — renvoie l'ensemble sous la forme d'un tableau NumPy, toujours de la forme `(N, F, T)`.

1.6 ‡ Question 6 – Génération automatique des annotations négatives

Vous savez désormais générer vos exemples positifs.

Dans certains jeux de données, des segments négatifs (ne contenant pas de pulse) sont explicitement annotés. Ce n'est pas le cas ici : vous devez les générer vous-même.

L'idée principale est la suivante : tout ce qui n'est pas annoté comme positif (i.e. qui ne figure pas dans `detections.csv`) est considéré comme négatif. Il s'agit donc de sélectionner des portions du signal à des positions temporelles aléatoires, en vérifiant qu'elles ne recouvrent aucun pulse positif.

1. Écrivez la fonction `generate_negative` qui :
   — choisit une fenêtre temporelle aléatoire dans le signal audio ;
   — vérifie que cette fenêtre ne contient aucun pulse annoté ;
   — renvoie le spectrogramme associé à cette fenêtre.

1.7 † Question 7 – Agrégation des non-pulses (négatifs)

Comme pour la question 5, vous allez maintenant agréger les exemples négatifs.

1. Écrivez la fonction `get_spectros_neg_one_file` qui, pour un fichier audio donné (par exemple `ValabFr-20221128_133055.735.flac`), renvoie sous la forme d'un tableau NumPy la matrice de spectrogrammes pour N négatifs générés automatiquement. Ici, N est un paramètre de la fonction.
2. Écrivez ensuite la fonction `get_all_spectros_neg` qui :
   — parcourt les M fichiers audio mentionnés dans `detections.csv` ;
   — génère, pour le fichier numéro i, un nombre N_i de négatifs égal au nombre de positifs dans ce fichier ;
   — renvoie l'ensemble des spectrogrammes négatifs sous la forme d'un tableau NumPy.
3. Répondez brièvement : quel problème nous rencontrons si nous ne génèrons pas autant de négatifs que de positifs ?

1.8 † Question 8 – Agréger les positifs et les négatifs

Vous disposez maintenant de l'ensemble des spectrogrammes positifs et négatifs.

1. Mélangez ces deux ensembles pour obtenir un nouvel ensemble de données, noté X, contenant à la fois des exemples positifs et négatifs. Écrivez la fonction `generate_dataset` qui :
   — prend en entrée l'ensemble des spectrogrammes positifs et l'ensemble des spectrogrammes négatifs ;
   — les concatène ;
   — applique une permutation aléatoire ;
   — sauvegarde le résultat dans un fichier `x.npy`.
2. Introduisez un vecteur d'étiquettes Y associé à X, par exemple :
   — $Y_i = 1$ pour un spectrogramme positif ;
   — $Y_i = 0$ pour un spectrogramme négatif.
   Modifiez la fonction `generate_dataset` pour qu'elle sauvegarde également ce vecteur dans un fichier `y.npy`.
3. Attention : Y doit subir **la même permutation aléatoire** que X, sans quoi vous perdriez la correspondance entre X_i et Y_i. Assurez-vous que cette cohérence est bien respectée dans votre code.

1.9 ‡ Question 9 – Data augmentation

En apprentissage automatique, la qualité et la quantité des données constituent un facteur essentiel de performance. Plus un modèle dispose d'exemples variés et représentatifs, meilleure sera sa capacité de généralisation.

Pour enrichir artificiellement un jeu de données limité, on peut appliquer différentes techniques de *data augmentation*. Nous en proposons ici deux adaptées à notre tâche :

— **Time shift augmentation** : à partir d'un même pulse, on génère K versions différentes en modifiant aléatoirement la position du pulse dans la fenêtre temporelle (ce qui revient à effectuer K décentrages différents). Avant cette question, votre pipeline équivalait à $K = 1$.
— **Brown noise augmentation** : on ajoute au signal original un bruit de type *brown noise* (bruit marron), simulant un bruit de fond de type "bruit de mer". Cette opération réduit volontairement le rapport signal-sur-bruit (SNR) afin d'entraîner un modèle plus robuste.

1. Modifiez votre code afin de permettre, via un paramètre de fonction (par exemple `K_shifts`), de générer automatiquement K versions *time-shiftées* d'un même pulse. Votre fonction devra donc :
   — répéter K fois l'opération de décalage temporel aléatoire ;
   — produire K spectrogrammes distincts pour chaque annotation positive.
   Vous pouvez directement modifier la fonction `get_spectros_pos_one_file`.
2. **Bonus** : ajoutez une seconde option de data augmentation permettant d'injecter du bruit marron au signal avant la génération du spectrogramme (par exemple via un paramètre `add_brown_noise=True/False`). Pour cela, vous pouvez vous inspirer du code disponible à l'adresse suivante : `https://github.com/Kettukaa/brown-noise`

Votre pipeline complet devra donc être capable, selon les paramètres fournis, de générer plusieurs spectrogrammes décalés temporellement, avec ou sans ajout de bruit marron.