Détection de chants d'oiseaux

Steven François, Alix Nehr

1 Jeu de données

Les données sur lesquelles nous travaillons nous sommes fournies en trois dossiers contenant des **pistes** audios à extension wav ainsi que les labels indiquant la présence ou non d'un oiseau pour deux de ces dossiers, ce qui nous mène donc à une **classification binaire** de fichiers audios.

Ces pistes audios sont des enregistrements courts, jusqu'à 15 secondes, qui se montrent variés et souvent peu explicites. En effet, on y retrouve parfois des voix humaines claires mais aussi et souvent uniquement des bruits d'environnement ouvert avec du bruit blanc ce qui est difficile à interpréter, les enveloppes sont très épaisses.

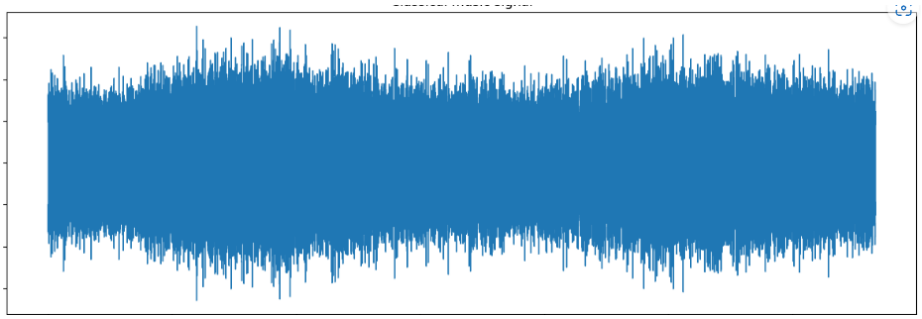

Figure 1: Enveloppe typique d'une des pistes audios

Dans un premier temps, nous écoutons quelques uns de ces fichiers en regardant les labels associés pour voir si la détection peut être a priori intuitive, tout du moins pour un humain, néanmoins trop de subtilités sont présentes pour se faire.

Nous comprenons avec ce premier point de vue qu'il nous faudra un modèle assez complexe pour réussir à identifier les différents sons : il ne s'agit pas tant de déceler des mélodies ou des patterns mais peut-être plutôt des fréquences précises, souvent dissimulées au milieu d'un tout assez fourni.

2 Pré-traitement

Ces données sont a priori très complexes de par leur format et leur dimension temporelle, nous concevons donc qu'il faut les transformer sous une autre forme afin de pouvoir nous ramener à des modèles de deep learning plus familiers.
Un format simple de chiffres ou de textes est trop limité pour capturer les différentes dimensions en jeu, nous nous dirigeons alors plutôt vers un format d'**images**.
À cet effet, nous nous penchons vers les modèles de **réseaux convolutionnels**, appropriés pour ces données transformées.

Pour passer d'audio à image, nous avons regardé trois possibilités :

- **Spectrogramme classique** : il s'agit d'une transformée de Fourier directe, néanmoins cela ne proposait pas de résultats convaincants car cette méthode n'est pas assez discriminante, les basses fréquences étant autant représentées que les hautes, nous avons besoin d'une transformation plus fine et complexe.

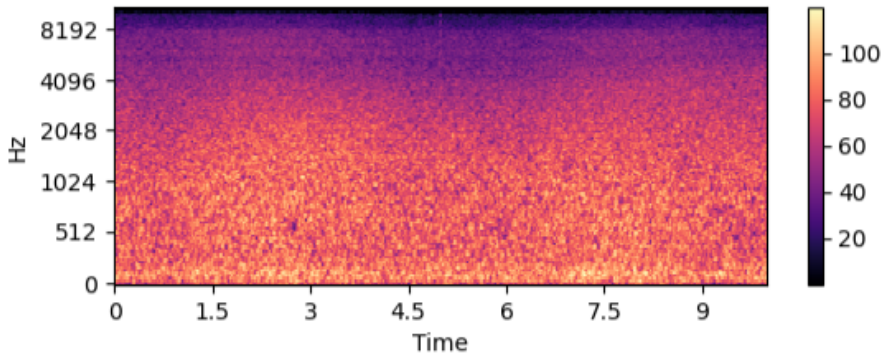

Figure 2: Spectrogramme typique d'une des pistes audios

- **Mel-spectrogramme** : il s'agit aussi d'une transformée de Fourier mais on y applique en plus des filtres linéaires type passe-bas ou passe-haut pour filtrer des fréquences parasites. Nous avons obtenu de meilleurs résultats car la transformation est plus discriminante mais cela gérait mal les bruits blancs donc nous étions alors mitigés quant à la qualité à cause des bruits de fond présents sur les pistes audios.

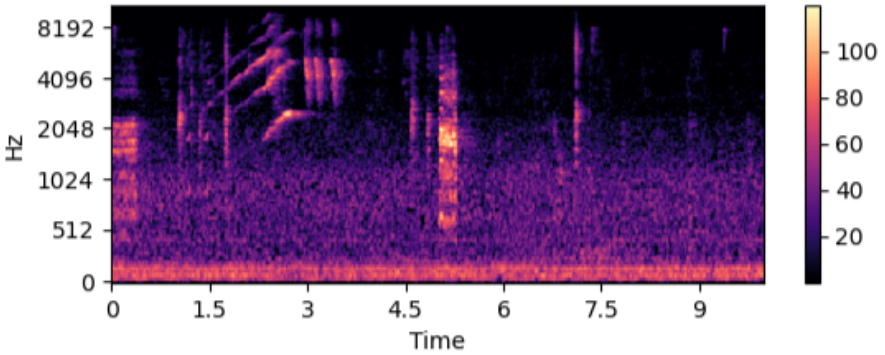

Figure 3: Mel-spectrogramme (passe-bas) typique d'une des pistes audios

- **PCEN** : il s'agit encore et toujours d'une transformée de Fourier mais cette fois-ci les filtres sont non-linéaires. Cette transformation est plus complexe à appréhender et à gérer mais nous avons fini par obtenir des résultats biens meilleurs avec une bonne discrimination des fréquences et un bon effacement de bruit blanc.

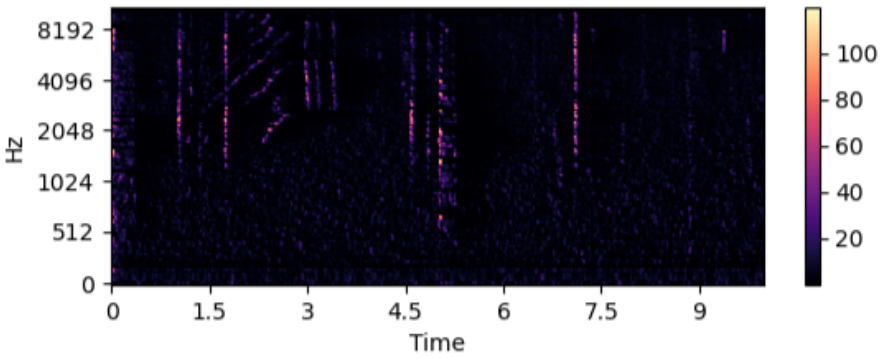

Figure 4: PCEN typique d'une des pistes audios

Ces transformations nous sont fournies par la librairie **librosa**, on peut en modifier une sélection assez vaste de paramètres (par exemple pour *librosa.pcen*) :

- *sr* : la fréquence d'échantillonnage du signal audio → **valeur par défaut donnée** par *librosa.load*
- *hop_length* : la taille du saut entre chaque frame → $\frac{\mathbf{sr}}{\mathbf{taille\ totale\ du\ son}} * \mathbf{1.1}$ (x1.1 pour éviter les effets de bord)
- *gain* : le facteur de gain → **0.98** (choix empirique)
- *bias* : le biais → **1** (valeur conseillée par librosa pour environnement ouvert + bruit blanc industriel)
- *power* : la puissance → **2** (idem)

Comme nous obtenons alors des images, nous avons tout normalisé sous le même format (128x431) dans trois tableaux correspondant aux trois dossiers.

Nous nous sommes ensuite intéressés à de la **data augmentation** qui se révèle toujours utile pour limiter l'overfitting. Il était un peu perturbant de retourner des images qui sont des spectrogrammes de fichiers audios donc nous ne sommes pas allés très loin mais avons fait quelques transformations basiques pour obtenir quelques milliers de fichiers supplémentaires.

3 Modèle

Maintenant que nous avons des images prétraitées, nous pouvons construire un modèle de réseau convolutionnel. À cet effet, nous préférons chercher sur Internet les modèles typiques pour la classification d'images plutôt que de construire à la main les couches.
Nous nous arrêtons alors sur le modèle **VGG-16**, déjà implémenté sur PyTorch. L'idée est ensuite de jouer sur les paramètres du modèle afin d'obtenir les résultats les plus convaincants possibles, néanmoins le nombre très important d'images injectées dans le réseau le rend très lent, même avec un data loader, et nous ne voulions pas limiter le nombre d'epochs de manière absurde. Dès lors, nous avons dû limiter le nombre d'essais pour chaque paramètre afin de pouvoir régler chacun d'entre eux.

3.1 Choix des hyperparamètres

- *Loss function* : **Binary cross-entropy** (choix usuel pour un modèle de classification binaire)
- *Metrics* : **Binary accuracy** (idem)
- *Optimizer* : **SGD (avec momentum à 0.9)** (après quelques recherches sur Internet nous nous sommes intéressés à Adam et SGD (avec momentum à 0.9). Nous avons ensuite fait un tirage à 20 epochs et learning rate de 0.001 pour chacun afin de comparer.)

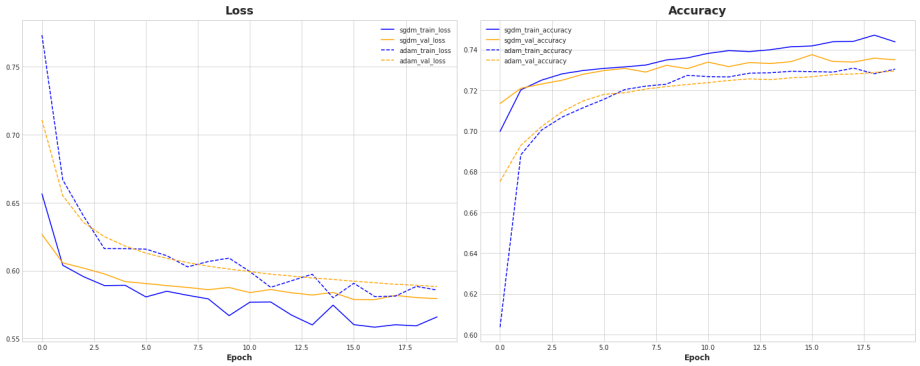

Figure 5: Loss et Accuracy pour chacun des deux optimizers retenus

- *Nombre d'epochs* : **20** (concession faite à cause du temps que chaque epoch prend, nous avons choisi ce nombre vu les courbes obtenues dans l'image précédente mais aurions aimé en prendre davantage.)
- *Batch size* : **4** (arbitraire)

3.2 Choix des paramètres du CNN

Nous avons préféré ne pas tant toucher à l'architecture qui est déjà relativement optimisée pour ce genre de projets pour nous concentrer sur des paramètres un peu plus spécifiques (dropout, batch normalization) et sur de l'augmantati

- *Taille des kernels* : **2** pour les layers MaxPool et **3** pour les layers convolutionnels
- *Stride* : **2** pour les layers MaxPool
- *Padding* : **1** pour les layers convolutionnels
- *Dropout* : **0.9 input et 0.5 hidden** (essais à 0, 0.5, 0.6 puis 0.4 pour les hidden layers puis une fois fixé à 0.5 nous avons placé à 0.9 pour l'input layer et cela a donné des résultats un peu meilleurs encore.)
- *Batch normalization* : **Oui** (choix intuitif)

4 Résultats

Figure 6: Loss pour le modèle final

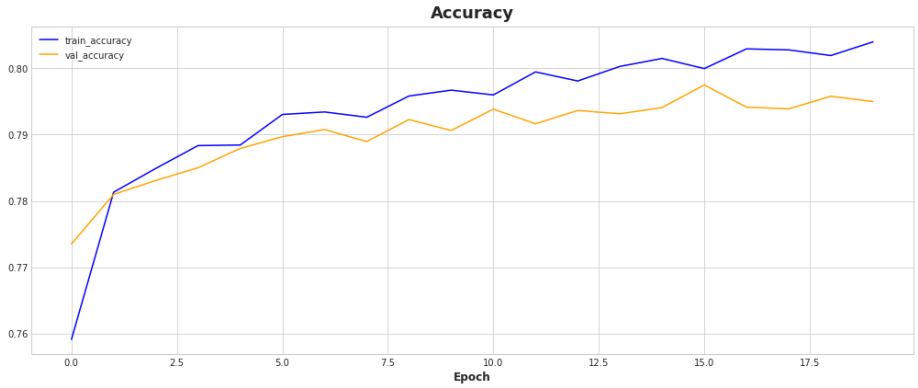

Figure 7: Accuracy pour le modèle final

Nous ne pouvons proposer une **matrice de confusion** étant donné que le jeu de test n'est pas labellisé en amont.

On voit une amélioration des résultats sans que l'on obtienne quelque chose de beaucoup plus convaincant niveau précision pour autant.
Le résultat final atteint les alentours des **0.795 d'accuracy** pour le jeu de validation ce qui n'est pas un score très appréciable.
Nous pensons tout d'abord qu'augmenter le nombre d'epochs aurait été profitable, notamment sur un jeu de données aussi grand. À part cela, nous estimons avoir un bon choix d'hyperparamètres.
L'amélioration pourrait plutôt se faire sur l'architecture du CNN, nous pensons

que le choix du VGG16 reste pertinent vu nos recherches internet en amont néanmoins il ne nous paraissait pas tant intéressant de modifier les paramètres comme la taille des kernels car ils sont intrinsèques au principe de ce modèle. Enfin, les paramètres type dropout et batch normalization ont pu être modifiés pour optimiser notre réseau.
Dès lors, l'amélioration devrait pouvoir se situer au niveau du pré-traitement. Nous pensons avoir exploré des pistes intéressantes de conversion des pistes audios en images jusqu'à aboutir au PCEN mais peut-être que certains paramètres de cette transformation auraient pu être encore plus optimaux, ou peut-être même qu'une autre transformation aurait été encore plus efficace. Nous pensons aussi que notre data augmentation a été suffisante vu que l'on semble correctement éviter l'overfitting mais peut-être que créer davantage de fichiers encore donnerait de meilleurs résultats finaux.