Classification de cris d'insectes

Karima Ghamnia[1,2], Cassandra Mussard[1,3] et Mickaël Song [1,4]

[1] 5ModIA, ENSEEIHT-INSA Toulouse
[2] Airbus [3] CNES [4] DGAC

Introduction

Les insectes terrestres qui resprésentent des composants essentiels de notre écosystème connaissent actuellement un déclin alarmant de près de 10% par décennie. Ce phénomène nécessite une compréhension approfondie et des méthodes de gestion efficaces. La recherche entomologique traditionnelle est limitée pour l'étude non invasive à haute fréquence spatio-temporelle des espèces diversifiées. C'est la raison pour laquelle nous allons envisager des méthodes d'IA en utilisant des enregistrements audio de type soundscape pour identifier les différentes espèces d'insectes.

Données

Nous disposons d'un jeu de données comprenant des enregistrements de 32 espèces d'insectes émettant des sons, totalisant 335 fichiers pour une durée totale de 57 minutes.

Ces données sont séparées en trois groupes : un dataset d'entraînement (sur lequel le modèle va apprendre la classification), un dataset de validation (qui permet d'évaluer les performances du modèle pendant l'entraînement) et un dataset de test (qui permet de voir une fois le modèle entraîné si il est capable de généraliser sur un dataset qu'il n'a pas vu pendant l'entraînement).

Découpage des audios

En analysant les enregistrements, nous avons remarqué que les audios n'avaient pas la même durée. Le problème est que si nous envisageons d'étudier les audios avec des CNN, nous allons avoir besoin en sortie d'un Multi-Layer Perceptron pour faire la classification. Ce MLP va nécessiter des features de taille fixe. Nous avons donc besoin d'avoir en entrée du réseau de neurones, des audios de même taille.

Pour remédier à ce problème, nous avons découper chaque audio en segment de durée 5 secondes. Enfin, si le dernier segment ne dure pas 5 secondes, nous le complétons par des 0.

Nous avons choisi de ne pas prendre des segments de 1 seconde car il n'y a pas assez d'informations à extraire pour faire la classification. Nous avons aussi essayé de prendre des audios de 8 secondes ou 10 secondes mais cela n'a pas amélioré notre classification.

Enfin, nous avons aussi essayé de compléter le dernier segment par le début de l'audio mais cela n'a pas changé le résultat de la classification. Nous sommes donc restés avec des segments de 5 secondes et le dernier segment a été rempli avec des 0.

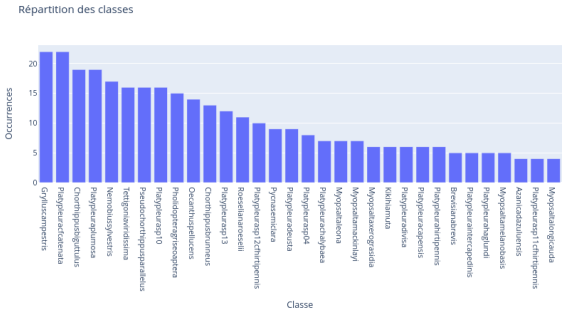

Figure 1: *Histogramme de la répartion des classes*

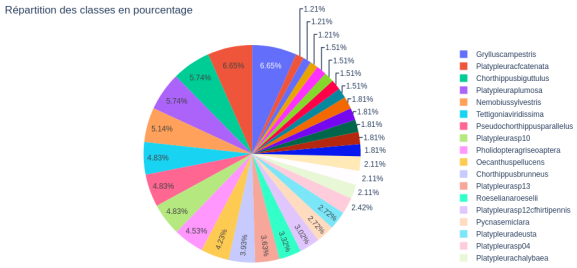

Figure 2: *Diagramme circulaire de la répartion des classes*

Occurence de chaque classe

Occurence des audios Comme nous pouvons le remarquer sur les figures (2) et (1), les classes sont très déséquilibrées. La classe la plus représentée est la classe Grylluscampestris avec plus de 20 audios au total. La classe la moins représentée est la classe Myopsaltalongicauda avec seulement 4 audios.

Il est clair que ce déséquilibre va avoir un impact considérable sur la classification. Pour pallier à ce déséquilibre nous avons utilisés deux méthodes (augmentation de données et ajout de poids dans la loss) dont nous allons discuter dans la suite.

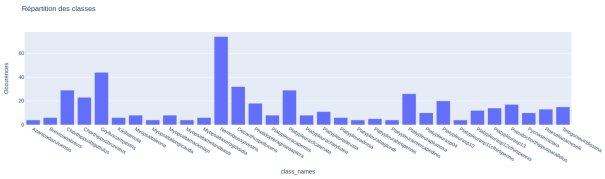

Figure 3: *Histogramme de la répartion des classes selon le découpage des audios*

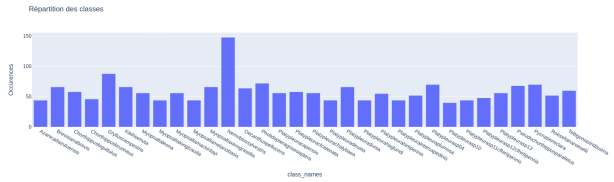

Figure 4: *Histogramme de la répartion des classes après augmentations de données*

Occurence des audios découpés Les graphiques ci-dessus représente les occurences des audios des classes. Selon la longueur des audios, une autre répartition des classes est donnée figure 3 car nous coupons les audios chaque 5 secondes : par exemple l'audio la plus longue, faisant 2 minutes 47 (Nemobiussylvestris_Take20.wav), nous donne 33 audios de 5 secondes pour la classe Nemobiussylvestris.

Dès lors, nous avons un jeu de données encore plus déséquilibrés avec des classes très sur-représentés, notamment la classe Nemobiussylvestris avec 74 données.

Augmentation de données Étant donné que le jeu de données contient très peu d'enregistrements en comparaison avec le nombre d'espèces que nous devons classer, nous avons réalisé une augmentation de données pour aider au mieux notre modèle à généraliser la classification. Nous avons appliqué les transformations suivantes qui nous paraissaient raisonables pour des enregistrements d'insectes :

- Ajout de bruit : Cette transformation consiste à bruiter l'audio (bruit blanc) en utilisant la fonction "AddGaussianNoise".
- Modification de la durée temporelle : Cette transformation réalisée avec la fonction "TimeStretch" consiste à ralentir ou accélérer l'enregistrement sans altérer sa fréquence.
- Modification du pitch : Cette transformation effectue un changement de hauteur (ou de pitch) du signal audio sans modifier sa durée temporelle. Elle est effectuée avec la fonction "PitchShift".
- Masque de fréquence : Cette transformation est utilisée pour masquer certaines fréquences dans le spectre audio avec la fonction "SpecFrequencyMask". L'utilisation de cette transformation est inspirée de [1], où cela est montré que cela apporte des amélioration à la performance du modèle.
- Masque de temps : Cette transformation consiste à masquer des plages de temps dans l'enregistrement avec la fonction "TimeMask".
- Inversion de temps : Cette transformation consiste à inverser le signal temporel de l'enregistrement avec la fonction "Reverse" et donc le début du signal devient la fin.
- Déplacement de temps : Cette transformation consiste à déplacer le signal temporel de l'enregistrement avec la fonction "Shift" pour simuler des variations de phase ou de positionnement dans le temps.

Les fonctions mentionnées ci-dessus proviennent de la librairie "Audiomentations".

Ces augmentations vont donc permettre d'augmenter le nombre de données pour notre entraînement mais également d'équilibrer notre jeu de données d'entraînement qui, comme souligné dans la partie , est très déséquilibré.

Pour cela, à partir de la figure 3, nous avons regroupé les classes en fonction de leur fréquence et leur avons appliqué un nombre différent de transformations choisies aléatoirement, sans remise parmi la liste ci-dessus : 10 transformations pour les classes du 1er quartile (sous-représentées), puis 6 (resp. 3) transformations pour les classes situées entre le 1er quartile et la médiane (resp. la médiane et le 3ème quartile). Enfin, une seule transformation est appliquée aux classes appartenant au 3ème quartile. Nous avons donc 10 fois plus de données des classes sous-représentées et 2 fois plus de données des classes déjà bien représentées. Nous obtenons ce nouveau histogramme de la répartition des classes figure 4.

Méthodes

Nous avons décidé d'étudier ce problème uniquement avec des données 2D, c'est à dire que nous considérons chaque segment comme un spectrogramme.

Points communs de toutes les méthodes

- Données d'entrée : Chaque architecture va prendre en entrée un spectrogramme produit par la fonction stft de librosa. Les spectrogrammes sont alors de taille 1025x690. Nous avons essayé de changer la façon de construire le spectrogramme, en utilisant les méthodes MFCC ou le mel-spectrogramme mais cela n'a pas affecté les résultats.

- Loss : Pour toutes les méthodes étudiées nous avons utilisé la loss Cross Entropy car nous avons un problème multi-classe. De plus, comme nous sommes en présence d'un dataset déséquilibré nous avons ajouté des poids à cette loss. Nous avons attribué un poids plus élevé aux classes minoritaires et un poids plus faible aux classes majoritaires. Pour réaliser cela, nous avons créer une liste où pour chaque classe nous avons associé un poids qui est égal à $\frac{1}{nb\_occurences\_classe}$.
- Algorithme d'optimisation : Nous avons travaillé avec l'algorithme Adam, avec un pas de 10^{-4} et une régularisation L2 car nous avons beaucoup d'overfitting dû au peu de nombre de données et du déséquilibre des classes.
- Early Sotpping : Comme expliqué précédemment, nous avons beaucoup d'overfitting. Nous avons donc réalisé une méthode d'Early Stopping où nous arrêtons l'entraînement dès que la loss de validation commence à augmenter alors que la loss d'entraînement continue de descendre.

Architecture permettant d'avoir la meilleur accuracy

Nous avons testé plusieurs modèles basés sur des CNN et adaptés aux représentations type images (spectrogrammes dans notre cas), mais celui qui nous a permis d'avoir la meilleure accuracy de **86%** est un modèle "fine-tuned" sur Xception [2].

Dans la figure 5 nous pouvons voir l'architecture du modèle Xception qui se compose de blocs intermédiaires avec des connexions directes et débute/termine par des blocs d'entrée-sortie tout en utilisant des convolutions en profondeur séparée (depthwise separable convolutions) pour capturer des caractéristiques complexes, ce qui contribue à la réduction de nombre de paramètres du modèle et l'amélioration des performances. Pour constuire le modèle fine-tuné nous avons commencé par récupérer les poids de Xception pré-entrainé sur ImageNet [3]. Puis, nous l'avons adapté à nos données en rajoutant les couches suivantes :

- Une couche de convolution 2D avec 32 filtres de taille 3x3 sans biais, un batch de taille 1 et un stride de 2.
- Une couche linéaire qui resprésente notre classifieur avec 32 neurones (égal au nombre de classes).

Ensuite, nous avons entrainé tout le modèle (y compris les couches de Xception pré-entrainées) sur notre jeu de données qui contient en plus les données augmentées comme expliqué dans .

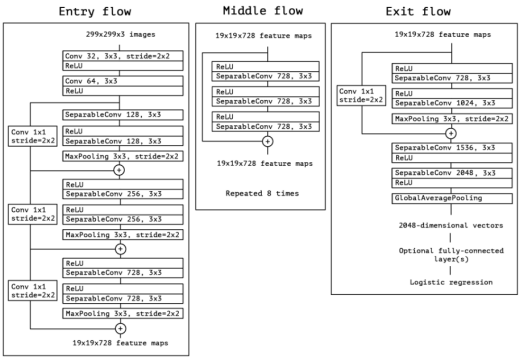

Figure 5: *Architecture de Xception*

Autres architectures étudiées

ResNet-18 Avant l'architecture Xception, nous avons testé une architecture ResNet-18 dont nous avons modifié la couche d'entrée et la couche de sortie afin de les adapter à nos données :

- Couche d'entrée : couche de convolution 2D avec 64 filtres de kernel 7x7 sans biais et de padding 3x3.
- Couche de sortie : couche linéaire qui resprésente notre classifieur avec 32 neurones (égal au nombre de classes).

Le réseau était pré-entraîné sur ImageNet et nous l'avons fine-tuné avec notre jeu de données d'entraînement après augmentations de données. Nous obtenons une accuracy de **72%** sur le dataset de test avec 10 epochs. Ce réseau a l'avantage d'être plus rapide au niveau de l'entraînement par rapport aux autres architectures que nous avons testé, tout en ayant de bons résultats.

Transformer Toujours dans le but de vouloir améliorer notre classification, nous avons testé une architecture de transformers pour les images avec une capacité bien plus élevée et qui permettrait peut-être de mieux capter les informations de nos données.
Comme notre dataset est petit et déséquilibré, nous avons réalisé du fine-tuning.
L'architecture utilisée provient de ([4]), nous pouvons la voir dans la figure (6). Ils ont proposé un transformer type ViT où chaque spectrogramme va être découpé en patch de taille 16x16.

Nous avons modifié la dernière couche pour obtenir en sortie une prédiction pour chacune des 32 classes. Ensuite, les données d'entrées ont été

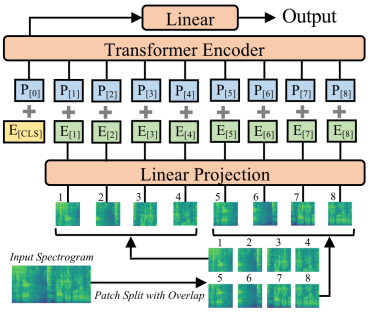

Figure 6: *Architecture d'encodeur de AST*

modifiées car cette architecture nécessite d'avoir des mel-spectrogrammes. Ce transformer a été entraîné sur ImageNet puis sur AudioSet (un dataset de clips audio extraits de vidéos provenant de YouTube). Nous atteignons seulement **63%** d'accuracy sur le dataset de test avec 4 epochs. Ce faible résultat est sûrement dû au fait que notre dataset est trop petit et déséquilibré.

Résultats

Pour évaluer nos résultats nous avons utiliser 2 méthodes.

La première consiste à calculer l'accuracy (pourcentage de bonne prédiction du modèle). Cela permet d'avoir un premier aperçu de la qualité du modèle choisi. Le calcul réalisé est le suivant :

$$Accuracy = \frac{Nombredeprdictionscorrectes}{Nombretotaldeprdictions} \times 100$$

.

La deuxième méthode consiste à calculer la f-mesure. La formule est la suivante :

$$F_1 = \frac{2 \times Prcision \times Rappel}{Prcision + Rappel}$$

Avec la précision qui mesure le ratio entre les vrais positifs et les instances positives prédites par le réseau:

$$Prcision = \frac{VP}{VP + FP}$$

Et le rappel mesure si le modèle est capable de prédire toutes les instances positives réelles parmi l'ensemble des instances réellement positives:

$$Rappel = \frac{VP}{VP + FN}$$

Résultats sur Xception

Comme expliqué dans la section précédente nous avons réussi à obtenir une accuracy d'environ 86% sur le dataset test sur 6 epochs.

Figure 7: *Accuracy sur le train et la validation*

```
F1-score pour la classe 0 : 0.6666666666666666
F1-score pour la classe 1 : 0.0
F1-score pour la classe 2 : 1.0
F1-score pour la classe 3 : 1.0
F1-score pour la classe 4 : 1.0
F1-score pour la classe 5 : 0.5
F1-score pour la classe 6 : 1.0
F1-score pour la classe 7 : 1.0
F1-score pour la classe 8 : 1.0
F1-score pour la classe 9 : 1.0
F1-score pour la classe 10 : 0.6666666666666666
F1-score pour la classe 11 : 0.6666666666666666
F1-score pour la classe 12 : 1.0
F1-score pour la classe 13 : 1.0
F1-score pour la classe 14 : 1.0
F1-score pour la classe 15 : 0.75
F1-score pour la classe 16 : 1.0
F1-score pour la classe 17 : 0.6666666666666666
F1-score pour la classe 18 : 0.6666666666666666
F1-score pour la classe 19 : 0.6666666666666666
F1-score pour la classe 20 : 1.0
F1-score pour la classe 21 : 0.3333333333333333
F1-score pour la classe 22 : 0.4
F1-score pour la classe 23 : 1.0
F1-score pour la classe 24 : 0.6666666666666666
F1-score pour la classe 25 : 0.0
F1-score pour la classe 26 : 0.6666666666666666
F1-score pour la classe 27 : 0.6
F1-score pour la classe 28 : 1.0
F1-score pour la classe 29 : 1.0
F1-score pour la classe 31 : 1.0
F1-score moyen : 0.8027027027027027
```

Figure 8: *F-mesure sur chaque classe*

La courbe d'accuracy obtenue pendant l'entraînement sur le dataset d'entraînement et de validation est visible sur la figure 7.

Ensuite, en ce qui concerne la f-mesure nous avons obtenu un résultat de 0.80 en moyenne sur toutes les classes. Sur chaque classe voici ce que nous avons obtenu :

Nous pouvons reamrquer que le modèle réussit mieux à prédire les espèces de type Orthopera que celles de type Cicadidae. Nous constatons aussi que le modèle n'arrive pas à prédire correctement les classes 1 et 25. Pour la classe 25 une simple explication peut-être que les classes 25 et 26 sont très proches et le réseau a du mal à les différencier. En ce qui concerne la classe 1, elle a très peu d'occurences donc le réseau a aussi du mal à distinguer les éléments de cette classe. Visuellement, nous avons affiché la matrice de confusion 8 et nous pouvons en tirer les mêmes conclusions que pour la f-mesure.

En conclusion, le réseau Xception est capable de bien classer chaque segment dans sa classe dans sa globalité mais rencontre quelques difficultés sur certaines classes notamment à cause du fait que cer-

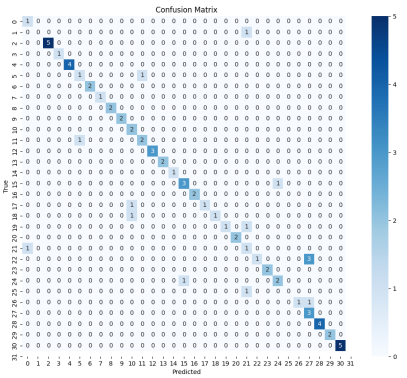

Figure 9: *Matrice de confusion*

taines classes sont similaires et que le dataset est assez petit.

Conclusion

Nous avons exploré plusieurs modèles avec différentes architectures, mais compte tenu des données disponibles, il est particulièrement difficile d'atteindre une précision supérieure à 80%. Il est important de noter que les chercheurs qui ont créé ce jeu de donnée en sélectionnant des sons à partir de "the Global Cicada Sound Collection hosted on Bioacoustica" ainsi que d'autres source, n'ont pas réussi à dépasser 78% d'accuracy même en utilisant "LEAF" [5], un réseau de neurones qui apprend la représentation spéctrale (spectrogramme) qui permet d'avoir la meilleure accuracy de classification possible. Nous pouvons donc conclure que nos résultats obtenus avec Xception, fine-tuné sur un jeu de données comprenant des augmentations, sont considérés comme satisfaisants en comparaison.

References

[1] Daniel S. et al. "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition". In: *Google Brain* (2019).

[2] Chollet F. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *Google* (2017).

[3] Deng J. et al. "ImageNet: A large-scale hierarchical image database". In: *IEEE Conference on Computer Vision and Pattern Recognition* (2009).

[4] Y. Gong, Y. Chung, and J. Glass. "AST: Audio Spectrogram Transformer". In: *MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA 02139, USA* (2021).

[5] M. Faiß and D. Stowell. "Adaptive Representations of Sound for Automatic Insect Recognition". In: ().

Remerciements

Nous remercions le Centre de Calcul du CNES pour l'utilisation de leurs ressources (utilisation d'une carte graphique Nvidia A100 80G).