Capteurs/actionneurs & IA embarquée

Projet Biodiversité CERN

Aurélia CHABANIER, Sami EL KATEB

UNIVERSITÉ CÔTE D'AZUR

MAI 2023

Table des matières

1 Introduction **2**

2 Matériels et Méthodes **3**
2.1 Flux de travail 3
2.2 Jeu de données 5
2.3 Description du CNN 6
2.4 Architecture du traitement de données par le capteur 6

3 Résultats **8**

4 Discussion **11**

1 — Introduction

Le domaine du CERN est un terrain riche en matière de biodiversité et présente plusieurs espèces rares [1]. Dans le cadre d'une politique de la sensibilisation à l'environnement l'Organisation a mis en place plusieurs mesures visant à favoriser la biodiversité sur ses terrains. Une des mesures mise en œuvre implique des relevés des diverses espèces qui composent la faune et la flore du domaine du CERN [2]

Dans ce projet, notre objectif sera de proposer un moyen de faciliter le relevé des différentes espèces d'oiseaux présent sur le domaine du CERN. Pour cela, nous utiliserons un microcontrolleur STM32 NUCLEO-L476RG ainsi qu'une carte d'acquisition audio personnalisée [1] founis par l'Université Côte d'Azur.

Notre objectif sera de pouvoir différencier 10 espèces d'oiseau d'intérêt potentiellement présentes sur le domaine du CERN. Pour tester nos résultats, nous vérifierons que les espèces soient correctement reconnues par la carte. Nous effectuerons ces tests en lançant sur haut-parleurs différents sons d'oiseaux. Les sons utilisés pour réaliser ces tests en temps réel seront différents des sons présents dans le jeu de données d'entrainement et de test.

[1] Cette carte personnalisée contient un convertisseur analogique-numérique TLV320ADC3101 et un microphone AOM-5024L-HD-R

2 — Matériels et Méthodes

2.1 Flux de travail

Pour réaliser le jeu de données nécessaire à l'entrainement de notre réseau de neurone, nous avons utilisé le site Xeno-Canto [3]. Celui-ci recense un grand nombre de chants d'oiseau de différentes espèces enregistrés par des utilisateurs. Cependant, les chants d'oiseaux enregistrés possèdent des caractéristiques différentes (durée d'enregistrement, format d'enregistrement, fréquence d'échantilonnage, bruit de fond). Notre premier défi a donc été de créer un jeu de données permettant d'obtenir des résultats corrects. Pour cela, nous nous sommes fixé pour objectif d'obtenir une précision supérieure à 80% sur les données de test avec un réseau de neurone de type M5 [4]. Cela nous permet entre autre de valider notre jeu de données. Nous avons donc suivi une approche itérative cf : Figure 2.1

À chaque itération, nous avons essayé d'obtenir une meilleure précision qu'à l'itération précédente. Notre première approche a consisté à récupérer le son de 10 espèces d'oiseaux depuis Xeno-Canto puis à les convertir en .wav avant de tester le réseau de neurone M5 dessus.

Pour ce faire nous avons développé un script de récupération de données (scrapping) en Python nous permettant de sélectionner les espèces que nous voulions télécharger depuis le site. Lors de la récupération des sons, nous avons choisi de laisser un délai de 1 seconde entre chaque téléchargement pour ne pas surcharger le site. Nous avons également utilisé un script de conversion des fichiers en wav et de resampling en 16 000 Hz. Les scripts utilisés sont disponibles sur notre repository GitHub [5].

Cette approche ne nous a malheureusement pas permis d'obtenir des résultats satisfaisants. En effet, le réseau de neurone M5 ne nous permettait d'obtenir qu'une précision de 20%.

Nous avons donc choisis de découper les sons de notre jeu de données en échantillons de 1 seconde. Ce traitement nous a permis d'obtenir une amélioration de 20% de la précision du réseau de neurone M5 qui est passée de 20% à 40%.

Cependant, nous pouvons trouver plusieurs défauts à cette approche. La principale est que sur les enregistrements de plusieurs minutes, les chants d'oiseau ne sont pas constant et il existe des périodes de silence. Nous en déduisons qu'une des raisons de la mauvaise performance de notre réseau de neurone et que certains échantillons ne contiennent pas de chants d'oiseaux.

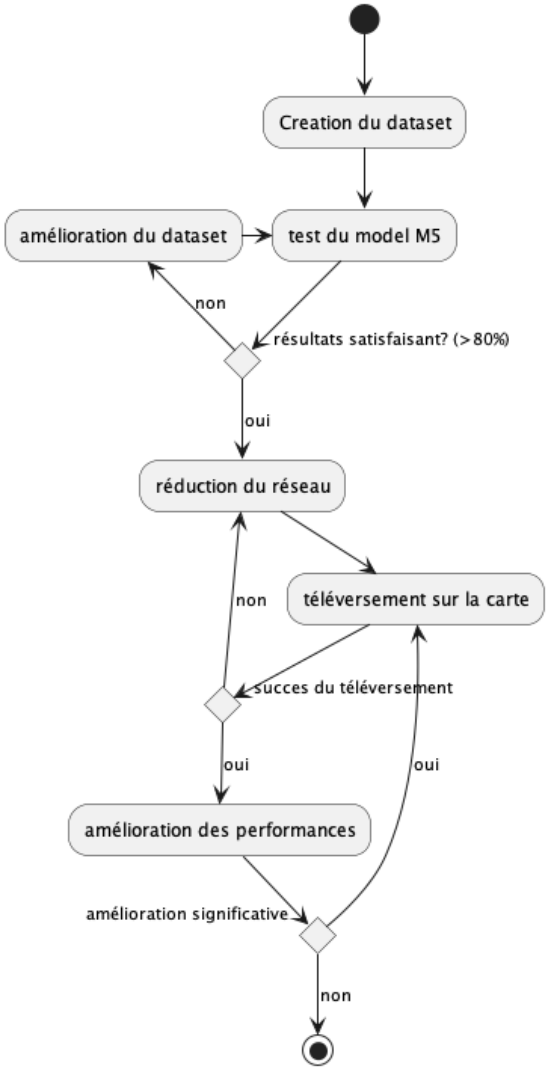


FIGURE 2.1 : Workflow Iteratif

Nous avons réfléchi à plusieurs façons de résoudre ce problème :

- Contrôler manuellement la présence ou non de chant d'oiseau
- Filtrer les sons automatiquement et supprimer les silences
- Entrainer un algorithme non supervisé pour permettre de détecter les échantillons potentiellement mal labélisés

Pour des raisons de temps et d'efficacité, nous avons choisi la seconde approche. Pour cela, nous avons utilisé le script "Silence Remove" proposé dans l'article *Audio Processing and Remove Silence using Python* [6]

Cette approche nous a finalement permis d'obtenir un taux de précision adéquat de 82% sur les données de tests. Nous avons donc choisi d'utiliser cette solution pour la création de notre jeu de données.

Par la suite, il a été nécessaire de réduire la taille du réseau de neurone ainsi que la quantité de données le traversant. Nous avons donc réduit le réseau de neurone M5 de manière itérative, en essayant de garder une précision aussi élevée que possible. Ceci a permis de téléverser sur la carte STM32 pour tester notre modèle. Ce qui finalise notre flux de travail cf : Figure 2.2.

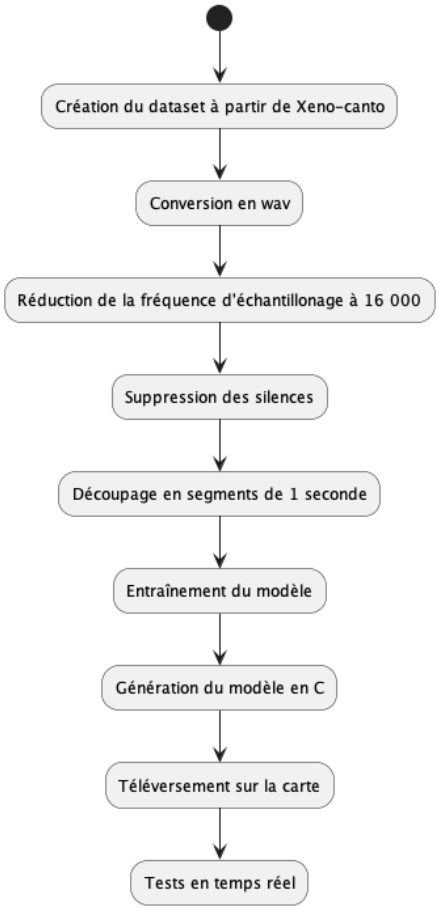


FIGURE 2.2 : Workflow Final

2.2 Jeu de données

Notre jeu de données final contient un total de 33 105 échantillons de 1 seconde répartis de manière équitable sur 10 classes. Celui-ci est disponible sur notre Google Drive[7]

Jeu de données : Espèces d'oiseaux			
Nom Français	Nom Anglais	Espèce	Nombre d'échantillons
Alouette des champs	Eurasian skylark	Alauda arvensis	3 381
Bruant jaune	Yellowhammer	Emberiza citrinella	3 278
Bruant zizi	Cirl bunting	Emberiza cirlus	3 305
Coucou gris	Common cuckoo	Cuculus canorus	3 323
Effraie des clochers	Barn owl	Tyto alba	3 254
Faucon crécerelle	Common kestrel	Falco tinnunculus	3 311
Fauvette des jardins	Garden warbler	Sylvia borin	3 398
Gobemouche gris	Spotted flycatcher	Muscicapa striata	3 319
Grèbe castagneux	Little grebe	Tachybaptus ruficollis	3 167
Hirondelle de fenêtre	Common house martin	Delichon urbicum	3 369

2.3 Description du CNN

Pour pouvoir déployer le CNN sur la carte STM32 NUCLEO-L476RG, il a été nécessaire de minimiser les ressources utilisées par notre réseau de neurone. Pour cela les paramètres à prendre en compte sont la taille et la quantité de données utilisée par le réseau (RAM et ROM) ainsi que le nombre de paramètres du réseau (temps de calcul).

Pour réduire la taille et le nombre de paramètres de notre réseau, nous avons diminué le nombre de filtres de nos Conv1D. Nous avons ensuite utilisé un MaxPool en entrée du réseau pour réduire la quantité de données initiales. Ceci a permis de ne pas dépasser la RAM disponible sur la carte et de réduire le nombre de paramètres à 12 234.

Input : 16000x1
Maxpool : 10x1 (output : 1600 × n)
conv1d [80/4, 32]
Maxpool : 4x1 (output : 100 × n)
conv1d [3, 32]
Maxpool : 4x1 (output : 25 × n)
conv1d [3, 32]
Maxpool : 4x1 (output : 6 × n)
conv1d [3, 32]
Maxpool : 3x1 (output : 1 × n)

2.4 Architecture du traitement de données par le capteur

Le microcontroller sur lequel nous avons déployé notre CCN est un STM32 NUCLEO-L476RG. Celui-ci est équipé d'une carte d'acquisition audio personnalisée contenant un convertisseur analogique-numérique TLV320ADC3101, un microphone AOM-5024L-HD-R ainsi qu'une prise d'entrée stéréo de 3,5.

Le microphone utilisé est analogique et nécessite donc l'utilisation d'un ADC.

Les ADC (Analogue-to-Digital Converter) permettent de convertir un signal analogique (signal continu) en signal digital (signal discret) qui peut ensuite être lu et traité par un microcontroller [8]. Dans notre cas l'ADC permet d'obtenir un signal numérique à partir du signal anologique fourni par le microphone.

Sur la carte Nucleo, l'ADC est connecté via la communication serial I2S. Les données numériques récupérées depuis l'ADC sont ensuite stockées automatiquement par le DMA.

Le DMA (Direct Memory Access) est une méthode de transfert de données où les périphériques accèdent directement à la mémoire sans passer par le CPU. Le CPU n'est alors responsable que de l'initiation du transfert de données et reçoit une interruption lorsque le transfert est terminé. Cela permet aux périphériques d'I/O d'accéder à la mémoire, sans monopoliser le CPU. [9]

3 — Résultats

Performance théorique : Suite à l'apprentissage du réseau de neurone, nous avons obtenu une précision de 59.0% sur les données d'entrainement, de 58.9% sur les données de tests avant quantization et 58% après. La matrice de confusion nous permet de voir les chants ont été globalement bien appris avec une majorité de vrais positifs pour la plupart des classes cf : Figure 3.2. Les classes présentant le plus de confusion sont les classes 0, 3, 4 et 5 avec un maximum de confusion pour la classe 3. Ces classes représentent les oiseaux suivants :

- 0 : Alauda Arvensis
- 3 : Emberiza Cirlus
- 4 : Emberiza Citrinella
- 5 : Falco Tinnunculus

```
[[476  12  78  41  44  27  35 150  66  31]
 [  3 715  74  18  10  46  58  13  38  19]
 [  6  36 588  26   9  75 195  24  37  27]
 [  9   9 109 458  41  80  84  51 105  32]
 [ 18  15  55  34 388 173  87  98  64  47]
 [  1  20  54  13  30 642  67  57  71  63]
 [  3  31  97  22  29 162 451  37  70  66]
 [ 23   3  35  11  45 111  20 787  30   5]
 [  5  14  42  37   9 141  88  16 567  38]
 [  4  15  32  26   8  32  71   0  29 803]]
```

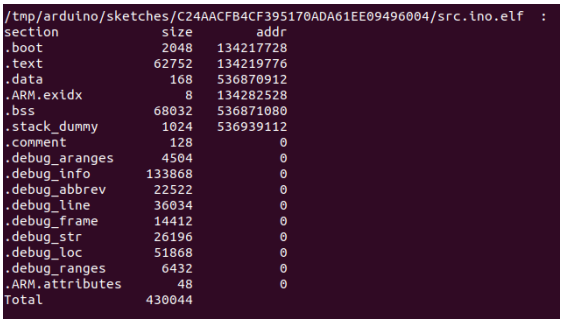
FIGURE 3.1 : Matrice de confusion

Performance expérimentale : Pour tester nos résultats nous avons déployé le réseau de neurone sur la carte Nucleo à l'aide de l'IDE Arduino. Pour cela nous avons utilisé le même projet Arduino que dans le Lab 5, lors de la réalisation du CNN à partir du jeu de donnée Google Speech Commons. Nous avons modifié le programme pour qu'il n'écrive dans la sortie Serial que lorsque la valeur prédite maximum est supérieure à 8000. Cela permet le n'afficher que les résultats ou ceux-ci sont plus en faveur d'une classe que d'une

autre. Nous avons ensuite testé de modèle en jouant sur haut-parleur des sons n'étant pas présents dans notre jeu de données.

Nous avons pu remarquer que lorsque qu'un chant d'oiseau est joué et qu'un résultat est donné, celui-ci est le bon dans la majorité des cas. Cependant, nous recevons parfois des résultats en présence de bruits forts, alors qu'aucun bruit d'oiseau n'est joué. Cela peut être dû à la présence dans notre jeu de données d'échantillons de bruits de fond, où aucun oiseau ne peut être entendu.

Mémoire : Pour évaluer la mémoire utilisée par la carte, nous avons utilisé Arduino en mode verbose pour pouvoir lire la ROM puis nous avons utilisé la dernière commande exécutée pour accéder au fichier elf et estimer la RAM.



```
/tmp/arduino/sketches/C24AACFB4CF395170ADA61EE09496004/src.ino.elf  :
section              size        addr
.boot                2048   134217728
.text               62752   134219776
.data                 168   536870912
.ARM.exidx              8   134282528
.bss                68032   536871080
.stack_dummy         1024   536939112
.comment              128           0
.debug_aranges       4504           0
.debug_info        133868           0
.debug_abbrev       22522           0
.debug_line         36034           0
.debug_frame        14412           0
.debug_str          26196           0
.debug_loc          51868           0
.debug_ranges        6432           0
.ARM.attributes        48           0
Total              430044
```

FIGURE 3.2 : Contenu du fichier elf

ROM	64 968 octets
RAM	68 032 octets

Latence : Nous n'avons pas pu tester la latence de la carte à l'aide du script d'évaluation python. Nous considérons donc que celle-ci est de 60 ms.

Energie : La carte utilisée étant la même que celle de la RFThings-AI Dev Kit board, nous faisons supposons que la consommation en mode active est la même (62 mW) et que la consommation en mode veille est nulle. Nous supposons également que l'appareil collectera des données sur une durée de 2,56 secondes. Enfin, nous considérons que la carte utilisera deux piles de type AA Alcalines ayant une charge de 3900 mWh.

	Unité de mesure	Estimation
Consommation énergétique en mode actif	mW	62
Consommation énergétique moyenne par inférence	mW	$62 * \frac{0.06}{2.56} = 1.453$
Énergie de batterie	mW	7 800
Autonomie de l'appareil avec mode veille	h	$\frac{7800}{1.453} = 5368$
Autonomie de l'appareil sans mode veille	h	$\frac{7800}{62} = 126$

4 — Discussion

En conclusion, ce projet nous a permis de découvrir et de mettre en pratique les techniques d'IA sur microcontrôleurs. Au travers de ce projet, nous avons appris à construire un jeu de données, et à optimiser un réseau de neurones pour pouvoir l'utiliser dans un système à ressources restreintes.

Nos résultats de détection des différentes espèces d'oiseaux ont été satisfaisants. Cependant, l'amélioration de la qualité de notre jeu de données aurait pu permettre d'obtenir de meilleurs résultats.

Une fois le projet terminé, nous avons pu continuer à nettoyer notre jeu de données en utilisant un algorithme de forêts aléatoires pour marquer les données potentiellement incorrectement labellisées. Les résultats obtenus ont été prometteurs et ont permis d'améliorer légèrement nos résultats précédents tout en ayant un nombre inférieur d'échantillon. Cette approche couplée à une augmentation des données pourrait donner de bons résultats.

Il pourrait être intéressant d'utiliser ces techniques pour améliorer nos jeux de données et ainsi obtenir de meilleurs résultats lors des tests sur microcontrôleurs.

Bibliographie

[1] Cian O'Luanaigh. *The birds and the beams : Biodiversity at CERN*. 2015. URL : https://home.cern/news/news/cern/birds-and-beams-biodiversity-cern (visité le 22/04/2023).

[2] CERN. *Environmental awareness : exploring CERN's biodiversity*. 2022. URL : https://home.cern/news/news/cern/environmental-awareness-exploring-cerns-biodiversity (visité le 22/04/2023).

[3] Xeno Canto. *Bird Sound Database*. URL : https://xeno-canto.org (visité le 22/04/2023).

[4] Wei Dai et al. "Very deep convolutional neural networks for raw waveforms". In : mars 2017, p. 421-425. DOI : 10.1109/ICASSP.2017.7952190.

[5] Sami EL KATEB Aurelia CHABANIER. *Projet Biodiversité CERN*. URL : https://github.com/SamiElkateb/embedded_ai_project.

[6] Bala Murugan N G. *Audio Processing and Remove Silence using Python*. 2020. URL : https://ngbala6.medium.com/audio-processing-and-remove-silence-using-python-a7fe1552007a (visité le 22/04/2023).

[7] Sami EL KATEB Aurelia CHABANIER. *Dataset Projet Biodiversité CERN*. URL : https://drive.google.com/file/d/1zVLOBS-vfTlgu_EdVVaLhvU9dm-XX1qR/view?usp=sharing.

[8] Dogan Ibrahim. "Chapter 1 - Microcomputer Systems". In : *PIC32 Microcontrollers and the Digilent chipKIT*. Sous la dir. de Dogan Ibrahim. Oxford : Newnes, 2015, p. 1-14. ISBN : 978-0-08-099934-0. DOI : https://doi.org/10.1016/B978-0-08-099934-0.00001-6. URL : https://www.sciencedirect.com/science/article/pii/B9780080999340000016.

[9] Microcontrollerslab. *Direct Memory Access (DMA)*. URL : https://microcontrollerslab.com/dma-introduction-working-programming-mode-arbitration-advantages/.