RAPPORT DE PROJET

Projet d'électronique n°[5] : [NEURAL SPEECH]

Auteurs :		*Enseignant :*
Angel	VELASCO	M. HAMADOUCHE
Jérémy	POPULAIRE	
Grégoire	MARCHAL	
Michael	ADDA	

Dans le cadre du module de calcul embarqué en 1ère année du cycle d'ingénieur à l'ECE, nous menons un projet ayant pour objectif la conception d'un système de reconnaissance vocale embarquée basé sur l'intelligence artificielle. Ce projet, intitulé Neural Speech, vise à exploiter l'ensemble des compétences acquises en traitement du signal et réseaux de neurones pour réaliser une application concrète.

Le projet repose sur l'utilisation de la carte Arduino Due, capable de numériser un signal audio provenant d'un microphone MAX9814. Le signal est ensuite filtré, caractérisé à l'aide de techniques comme l'extraction des coefficients MFCC, puis traité par un réseau de neurones embarqué afin d'identifier les mots prononcés.

Le cas d'usage que nous avons choisi nous permet de mettre en œuvre une chaîne complète d'acquisition, de traitement et de classification sur une cible contraignante en ressources. Il s'agit donc d'un projet à la fois technique et formateur, qui nous pousse à optimiser chaque étape pour respecter les limites de notre système embarqué.

Ce projet nous permet également de travailler en sous-groupes sur différentes briques fonctionnelles (acquisition, traitement du signal), et ainsi de simuler une approche de développement en équipe proche du monde industriel. Enfin, il constitue une transition précieuse entre l'apprentissage académique et l'application concrète des notions vues en cours.

Nous attestons que ce travail est original, qu'il est le fruit d'un travail commun au binôme et qu'il a été rédigé de manière autonome.

Paris, le 18/05/2025

Table des matières

I. Objectif

Ce document a pour but de présenter en détail le projet Neural Speech, afin de permettre au lecteur de comprendre à la fois les objectifs visés et les étapes clés de sa mise en œuvre. Il offre une vue d'ensemble structurée du travail accompli tout au long du projet, depuis les premières réflexions jusqu'à la réalisation concrète.

Le lecteur y trouvera dans un premier temps une introduction au projet avec le contexte dans lequel il s'inscrit, la problématique traitée, ainsi que la composition et l'organisation de notre équipe. Le document présentera ensuite les choix techniques effectués, l'architecture globale du système (tant sur les plans matériel que logiciel), ainsi que le fonctionnement détaillé des différents modules développés.

Les phases de test et de validation seront également décrites, avec les résultats obtenus et leur interprétation. Pour conclure, ce rapport propose un bilan critique sur les objectifs atteints, les points d'amélioration possibles, et une évaluation du travail collectif mené. Des ressources complémentaires et annexes seront également disponibles en fin de document.

II. Glossaire

Quels sont les termes et les acronymes qu'il est nécessaire de définir pour le lecteur ?

A. Termes

Terme	Définition
carte Arduino Due	Carte microcontrôleur 32 bits ARM Cortex-M3 avec 96 Ko de SRAM
down-sampling	Réduction de la résolution ou du nombre d'échantillons d'un signal ou d'une image

B. Acronymes

Acronyme	Signification	Explication
MFCC	Mel Frequency Cepstral Coefficients	Ce sont des coefficients qui représentent le spectre sonore selon l'échelle mél, largement utilisés en traitement audio pour extraire des caractéristiques pertinentes du signal
ADC	Convertisseur analogique-numérique	Composant électronique qui transforme un signal analogique continu en signal numérique discret
FFT	Transformée de Fourier	Algorithme pour calculer la transformée de Fourier discrète, utilisé pour analyser les fréquences d'un signal numérique

III. L'équipe

A. Présentation de l'équipe

Qui sont les membres qui composent l'équipe ?

Notre équipe est composée de quatre étudiants qui sont Angel Velasco, Jérémy Populaire, Michael Adda et Grégoire Marchal, tous en première année du cycle ingénieur à l'ECE.

Quelles sont leurs compétences et qualités ?

Nous avons tous suivi une formation en prépa intégrée à l'ECE, ce qui nous a permis d'acquérir des bases solides en électronique, en mesure avec capteurs, en systèmes bouclés et en FPGA. Cette année, nous avons commencé à nous familiariser avec les principes du calcul embarqué, ce qui nous a donné les outils nécessaires pour aborder ce projet avec sérieux.

Nous sommes investis dans notre travail, curieux d'approfondir nos compétences, et conscients de l'importance de collaborer efficacement en équipe. Ce projet est pour nous l'occasion de passer de la théorie à la pratique, de tester nos acquis, mais aussi de progresser en découvrant de nouveaux aspects du développement embarqué.

B. Organisation de l'équipe

Comment est organisée l'équipe ? Comment est réparti le travail ?

Pour organiser notre travail, nous avons d'abord décidé de répartir les tâches selon les compétences et intérêts de chacun. Angel et Jérémy se sont principalement concentrés sur la partie acquisition du signal audio, c'est-à-dire la numérisation et le conditionnement du signal à proprement parler. De leur côté, Grégoire et Michael ont pris en charge le développement du réseau de neurones, son entraînement ainsi que l'intégration des modèles dans le système embarqué.

Malgré cette répartition, nous avons toujours veillé à maintenir une bonne communication et à nous entraider régulièrement. Bien évidemment, lorsque l'une des équipes à terminer sa sous-partie, elle est venu prêter main-forte à l'autre pour avancer plus rapidement et éviter les blocages. Cela nous a permis de garder une cohésion de groupe et de progresser ensemble.

Nous essayons aussi de nous retrouver au moins une fois par semaine à l'école, parfois avant ou après les cours, pour travailler sur le projet en groupe. Ces sessions communes sont l'occasion de faire le point sur l'avancement, de partager nos difficultés et de trouver des solutions ensemble. Cette organisation nous aide à rester efficaces et à garder un bon rythme de travail tout au long du semestre.

C. Diagramme de Gantt

Comment est utilisé le temps alloué au projet ?

La figure 1 ci-dessous présente un diagramme de GANTT qui retrace la manière dont nous avons réparti notre temps sur les différentes étapes du projet au fil des six semaines. Ce diagramme met en évidence une organisation structurée et progressive, pensée pour avancer de façon cohérente sur chaque module sans négliger aucune partie du projet.

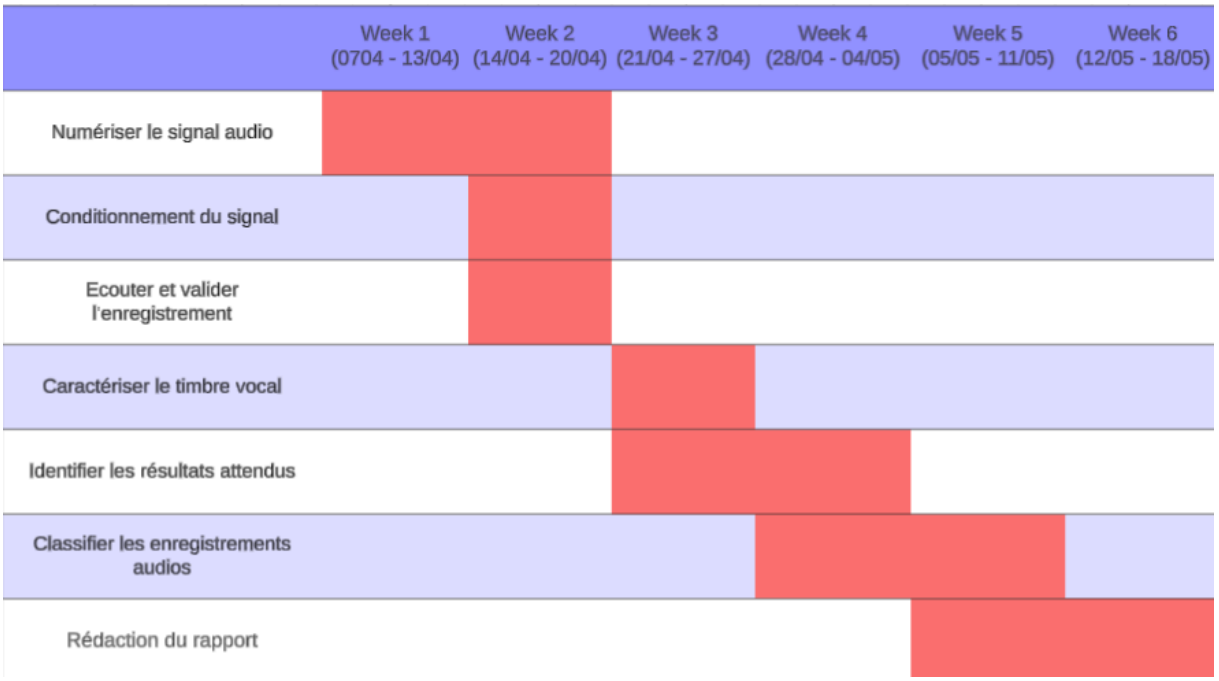


Figure 1 : Diagramme de GANTT

IV. Contexte et problématique

A. Contexte

Quel est le contexte économique et ou sociétal du projet ?

Comment est née l'invention / la technologie du projet, comment a-t-elle évolué ?

Aujourd'hui, la reconnaissance vocale est de plus en plus présente dans notre quotidien. On la retrouve dans les assistants vocaux, les systèmes domotiques, les applications de traduction instantanée ou encore dans les outils d'accessibilité pour les personnes en situation de handicap. Cette technologie s'inscrit dans un contexte sociétal où l'interaction homme-machine tend à devenir plus intuitive, plus naturelle, et où la voix devient un moyen de commande aussi courant que le tactile ou le clavier. Sur le plan économique, de nombreux secteurs investissent massivement dans l'intelligence artificielle embarquée, que ce soit pour améliorer l'expérience utilisateur, automatiser des tâches ou développer des objets connectés intelligents. Dans ce contexte, les systèmes embarqués capables de traiter la voix localement, sans dépendre d'un serveur distant, prennent de plus en plus d'importance.

L'idée de base de la reconnaissance vocale remonte aux premiers systèmes capables d'identifier quelques mots isolés dans les années 1950. Ces systèmes étaient très limités, coûteux et peu fiables. C'est au fil des décennies, avec l'augmentation des puissances de calcul, l'amélioration des algorithmes de traitement du signal et surtout l'arrivée des réseaux de neurones, que la reconnaissance vocale a réellement évolué. Aujourd'hui, grâce à des techniques comme l'extraction des coefficients MFCC ou l'utilisation de modèles d'apprentissage automatique, on peut reconnaître des mots ou des phrases en temps réel, même sur des systèmes embarqués avec peu de ressources.

Ce projet s'inscrit donc dans cette évolution technologique. Il propose d'explorer une version simplifiée mais concrète de ce que font aujourd'hui des géants comme Google ou Amazon, en se basant

uniquement sur une carte Arduino Due et un micro externe. En travaillant sur cette problématique, nous avons l'opportunité de mieux comprendre les enjeux techniques de la reconnaissance vocale embarquée, tout en nous confrontant aux contraintes réelles du matériel. C'est aussi l'occasion de réfléchir à la manière dont ces technologies peuvent être utilisées demain, dans des contextes variés, qu'ils soient domestiques, industriels ou même médicaux.

B. Problématique

À quelle problématique répond le projet ?

Le projet Neural Speech s'intéresse à une problématique de plus en plus présente aujourd'hui, celle de rendre possible la reconnaissance vocale sur un système embarqué simple, sans passer par des serveurs externes ou des équipements lourds. Dans un contexte où la commande vocale se généralise, que ce soit à travers les assistants personnels ou les objets connectés, l'enjeu est de faire fonctionner ce type de technologie localement et de manière totalement autonome.

L'objectif est de traiter et d'interpréter un signal vocal directement sur une carte Arduino Due, qui reste une plateforme relativement limitée en ressources. Cela impose plusieurs contraintes. Il faut d'abord réussir à capter le son, le filtrer, l'analyser, en extraire les bonnes informations, et tout cela dans un temps réduit sans surcharger la mémoire ni le processeur. Ce projet nous pousse donc à concevoir des solutions simples, efficaces et adaptées aux capacités réelles de la carte.

Avec cette approche, l'idée est de permettre à un utilisateur d'activer une fonction, de donner un ordre ou d'interagir avec une machine simplement à la voix.

C. Spécifications techniques

Quelles sont les spécifications techniques du projet ?

NB : Certains projets d'électronique à l'ECE n'en ont pas.

Le projet Neural Speech repose sur des spécifications techniques essentielles pour garantir un traitement vocal embarqué fiable et efficace. La carte Arduino Due joue un rôle central dans l'architecture du système. Grâce à son microcontrôleur ARM Cortex-M3 de 32 bits cadencé à 84 MHz, elle offre une puissance de calcul suffisante pour assurer à la fois l'acquisition du signal audio, son traitement en temps réel et l'exécution du réseau de neurones embarqué. Elle dispose de 96 Ko de mémoire RAM, ce qui permet de stocker temporairement les données audio et les coefficients extraits, tout en conservant une bonne réactivité du système.

Le microphone MAX9814 complète cette configuration en assurant la capture du signal vocal avec une grande sensibilité. Ce module intègre un amplificateur automatique qui ajuste le gain en fonction de l'intensité du son, ce qui permet d'obtenir un signal exploitable même lorsque la voix de l'utilisateur varie en volume. Il est particulièrement adapté aux environnements où le niveau sonore peut être instable, ce qui en fait un choix pertinent pour ce type d'application.

Ces deux composants, choisis pour leur complémentarité, permettent au système de fonctionner de manière autonome et fiable, tout en respectant les contraintes liées à l'environnement embarqué.

V. Conception

A. Architecture fonctionnelle

Quelle est l'architecture fonctionnelle du projet ?

NB : Les fonctionnalités sont des verbes à l'infinitif suivi de compléments.
À ce stade, aucun choix technique n'est fait.

Sur la figure 2, nous pouvons observer le schéma de l'architecture fonctionnelle de notre projet, qui regroupe les différentes étapes du traitement vocal embarqué, depuis la capture du signal audio jusqu'à la reconnaissance et l'exécution d'une commande.

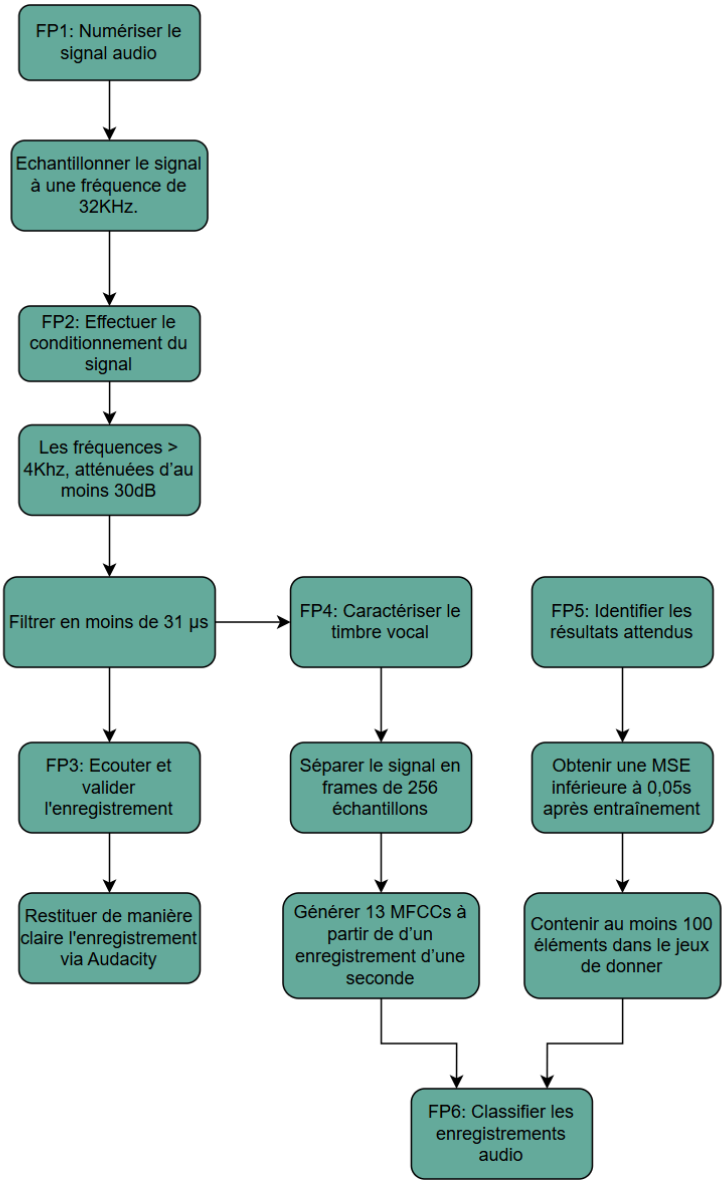


Figure 2 : Schéma de l'architecture fonctionnelle du projet

B. Architecture matérielle

Quel matériel est utilisé et pourquoi ?
Comment les différentes briques techniques sont connectées entre elles ?

NB : Cela peut-être une schématique de circuit électronique.

Comme nous pouvons l'observer sur la Figure 3 ci-dessous, nous avons utilisé pour la réalisation de notre système de reconnaissance vocale embarquée les composants suivants :

- Une carte Arduino DUE, pour piloter l'ensemble du système, effectuer l'acquisition et le traitement du signal, et exécuter le réseau de neurones embarqué.
- Microphone MAX9814, pour capter le signal vocal et l'amplifier avant son traitement.
- Deux leds (une rouge et une bleue), pour indiquer visuellement le mot reconnu par le système.
- Un bouton poussoire, pour lancer manuellement l'enregistrement vocal à analyser.
- Un écran OLED, pour afficher une animation lors d'une demande vocale.

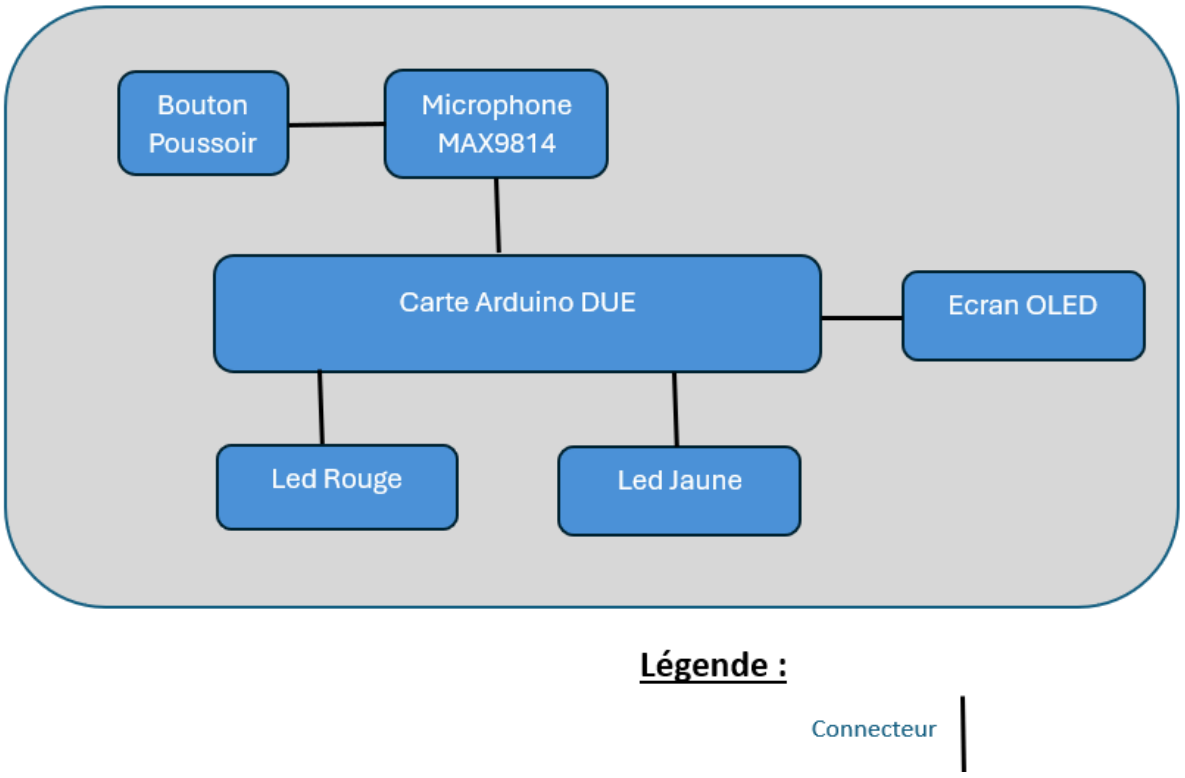


Figure 3 : Schéma de l'architecture matérielle du projet

C. Architecture logicielle

Comment fonctionne le programme ?

NB : Présenter un algorigramme de votre code si vous en avez un.

Sur la figure 4, nous pouvons retrouver le diagramme de fonctionnement logiciel de notre projet Neural Speech, qui détaille l'enchaînement des différentes étapes de l'acquisition du signal vocal, passant par son traitement jusqu'à la classification du mot prononcé.

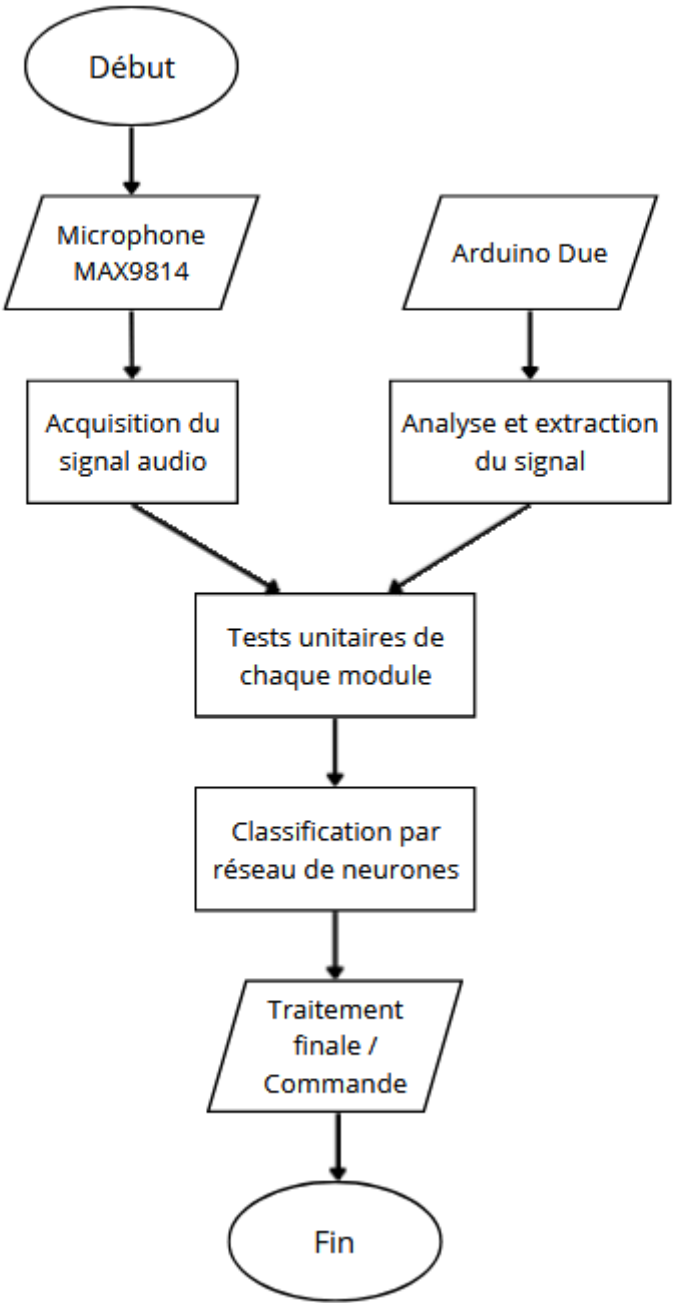


Figure 4 : Diagramme de fonctionnement de notre projet Neural Speech

VI. Développement

L'idée est de présenter ici comment ont été développés les différents blocs du projet. Cela peut rassembler des calculs théoriques, des choix techniques, etc. et surtout bien expliquer le concept clef derrière sa fabrication.

Le lecteur doit être capable de comprendre les enjeux techniques et de développer le module en question à l'aide de ces sous-sections.

A. FP1 : Numériser le signal audio

L'objectif de ce module est de numériser le signal audio provenant du microphone MAX9814 à l'aide de l'ADC de la carte Arduino Due, à une fréquence d'échantillonnage précise de 32 kHz. Cette fréquence est essentielle pour garantir une fidélité suffisante dans la capture des signaux vocaux et permettre, par la suite, l'application de filtres numériques efficaces.

Pour assurer un échantillonnage régulier à 32 kHz, nous avons configuré l'ADC de l'Arduino Due en mode interruption sur timer. Ce mode permet d'appeler périodiquement une fonction de lecture ADC sans recourir à des boucles actives bloquantes. L'utilisation d'un timer matériel garantit une fréquence d'échantillonnage stable, ce qui est indispensable pour éviter la distorsion temporelle du signal audio.

La formule permettant de calculer la période d'interruption est :

$$T_{échantillonnage} = \frac{1}{f_{échantillonnage}} = \frac{1}{32000} = 31,25\ \mu s$$

Nous avons donc programmé le Timer Counter (TC) de l'Arduino Due pour générer une interruption toutes les 31.25 µs. Lors de chaque interruption, une conversion ADC est déclenchée et la valeur obtenue est stockée dans un buffer circulaire pour traitement ultérieur.

Afin de valider que la fréquence d'échantillonnage est bien respectée, nous avons mis en place un test de vérification basé sur la reconversion du signal numérisé à l'aide du DAC de la carte. En injectant ce signal analogique dans un oscilloscope, nous avons pu mesurer visuellement le taux de variation du signal en fonction du temps. Nous vous présenterons cela lors de l'étape tests et validation de ce module.

Nous avons également vérifié que la condition de Nyquist était respectée. En effet, avec une fréquence d'échantillonnage de 32 kHz, le théorème de Nyquist impose que seules les composantes fréquentielles inférieures à 16 kHz soient correctement reconstruites :

$$f_{max} < \frac{f_{échantillonnage}}{2} = 16\ kHz$$

Or, la voix humaine est principalement contenue dans la bande 300 Hz – 3400 Hz, ce qui est largement inférieur à cette limite, garantissant la validité de notre choix de fréquence.

B. FP2 : Conditionnement du signal

Maintenant, nous avons pour objectif de préparer le signal audio numérisé pour une analyse efficace par le réseau de neurones embarqué. Cela inclut l'atténuation des fréquences supérieures à 4 kHz d'au moins 30 dB et la réduction de la fréquence d'échantillonnage de 32 kHz à 8 kHz. Ce processus est indispensable pour éviter le repliement spectral lors du down-sampling, tout en assurant la clarté du signal vocal.

Pour atteindre cet objectif, deux types de filtres numériques sont envisageables. Ce sont les filtres RII (Réponse Infinie à l'Impulsion) et les filtres RIF (Réponse Infinie Finie). Le filtre RII présente l'avantage d'être plus léger en calcul, mais il est potentiellement instable et sa phase n'est pas linéaire. À l'inverse, le filtre RIF est toujours stable, garantit une phase linéaire (donc pas de distorsion temporelle des composantes fréquentielles), mais nécessite un plus grand nombre de coefficients.

Au vu des avantages offerts par la stabilité et la linéarité de phase, nous avons opté pour un filtre RIF. La conception de ce filtre a été réalisée sur Python à l'aide de la bibliothèque SciPy. Nous avons utilisé la méthode de la fenêtre de Hamming pour obtenir un filtre passe-bas ayant une fréquence de coupure de 4 kHz. Le code de conception est le suivant :

```
from scipy.signal import firwin, freqz

fir_coefficients = firwin(num_taps=61, cutoff=4000, fs=32000, window="hamming")
```

Une fois les coefficients générés, ils ont été intégrés dans le microcontrôleur Arduino Due pour une exécution en temps réel. Pour ce faire, nous avons implémenté un buffer circulaire de taille 61 pour stocker les échantillons ADC et effectuer la convolution avec les coefficients du filtre. Cette opération est réalisée à chaque interruption déclenchée par le timer programmé à 32 kHz.

La réduction de la fréquence d'échantillonnage (down-sampling) est obtenue en conservant un échantillon sur quatre, soit une fréquence finale de 8 kHz, adaptée à la bande passante de la voix humaine (300 Hz – 3400 Hz).

Chaque calcul de filtrage doit être réalisé en moins de 31 µs (correspondant à la période d'échantillonnage à 32 kHz). Pour ce faire, nous avons vérifié que le microcontrôleur exécutait correctement les 61 multiplications/accumulations dans ce laps de temps, en mesurant les durées avec la fonction micros().

Enfin, le signal filtré est transmis sous forme binaire via le port série, afin de faciliter son analyse lors du FP3 avec Audacity.

C. FP3 : Écouter et valider l'enregistrement

Notre but dans ce module, est de vérifier que le signal audio capturé et filtré est correctement interprétable en pratique. Plus précisément, nous devons être capables de restituer un enregistrement clair du mot « Électronique » en utilisant le logiciel Audacity. Cette étape permet de valider le bon fonctionnement des modules précédents (FP1 pour la numérisation, et FP2 pour le conditionnement du signal).

Pour atteindre cet objectif, nous avons mis en œuvre un mécanisme de transfert des données audio filtrées depuis l'Arduino Due vers un PC en utilisant la liaison série. Le port série a été configuré à une vitesse de 460800 bauds, afin d'assurer une transmission rapide et stable des échantillons tout en limitant le risque de saturation du buffer.

La chaîne d'acquisition établie dans les modules précédents (FP1 et FP2) a été intégralement réutilisée. Elle repose sur une fréquence d'échantillonnage initiale de 32 kHz, combinée à un filtrage passe-bas numérique réalisé en interruption à l'aide d'une convolution avec 45 coefficients. Ce filtrage est suivi d'un processus de décimation avec un facteur de réduction de 4, ramenant la fréquence effective à 8 kHz, ce qui correspond à la bande utile pour l'analyse de la parole humaine.

Lorsque le buffer de sortie atteint sa capacité maximale (ici fixée ici à 100 échantillons), les données sont immédiatement transmises au PC sous forme binaire. Chaque échantillon est envoyé en deux octets successifs, en commençant par l'octet de poids faible (LSB), puis l'octet de poids fort (MSB), garantissant une reconstruction correcte lors de l'importation dans un logiciel de traitement audio comme Audacity.

Pour valider le bon fonctionnement du module, nous avons commencé par lancer la capture audio directement depuis l'Arduino Due. Une fois l'enregistrement effectué, les données binaires ont été transmises via la liaison série. Nous avons utilisé un terminal série pour réceptionner ces données et les enregistrer dans un fichier brut au format .bin. Ce fichier a ensuite été importé dans le logiciel Audacity pour écouter l'enregistrement. Grâce à cette procédure, nous avons réussi à écouter le mot enregistré et d'en analyser la qualité sonore. Nous vous présenterons dans la partie tests et validation de ce module, un lien pour écouter l'enregistrement du mot “électronique” grâce à notre microphone MAX9814.

D. FP4 : Caractériser le timbre vocal

Dans ce module, l'objectif est de caractériser le signal vocal numérisé et filtré à l'aide de l'algorithme MFCC, en générant 13 coefficients représentatifs pour chaque frame de 256 échantillons. Cette transformation permet de condenser les informations utiles du signal vocal en vue de la classification par un réseau de neurones.

Pour cela, nous avons utilisé la librairie arduinoMFCC, adaptée pour fonctionner sur la carte Arduino Due en temps réel. Cette librairie intègre toutes les étapes de calcul nécessaires à l'extraction des MFCCs.

Avant d'implémenter l'algorithme complet, nous avons d'abord configuré le système pour traiter les frames issues du module précédent (FP2). Chaque frame contient 256 échantillons avec un recouvrement de 50 %, soit une superposition de 128 échantillons entre deux frames consécutives. Ce recouvrement garantit la continuité temporelle du signal analysé.

Pour chaque frame de 256 échantillons, le signal audio subit une série d'étapes de traitement destinées à en extraire les caractéristiques spectrales les plus pertinentes. Tout d'abord, les échantillons, initialement codés en entiers, sont convertis en nombres flottants afin de permettre les calculs de transformation fréquentielle avec une meilleure précision. Une fenêtre de Hamming est ensuite appliquée sur la frame pour réduire les discontinuités aux extrémités et ainsi limiter les effets de bord dans le domaine fréquentiel. Le signal fenêtré est alors transformé par une transformée de Fourier rapide qui permet d'obtenir une représentation fréquentielle du signal. Cette représentation est ensuite traitée à l'aide d'un banc de filtres de Mel, dont la distribution est inspirée de la perception humaine des fréquences, afin de produire un spectre perceptuel plus pertinent. Enfin, une transformée en cosinus discrète (DCT) est appliquée sur le logarithme du spectre de Mel, ce qui permet de compacter l'information en un ensemble réduit de 13 coefficients appelés MFCCs, qui résument efficacement le contenu spectral de la frame.

E. FP5 : Identifier les résultats attendus

L'objectif de ce module est de concevoir un réseau de neurones capable de distinguer des mots prononcés à partir des caractéristiques extraites du signal vocal, à savoir les MFCCs. Le réseau doit être entraîné sur ordinateur, puis ses poids exportés pour une implémentation embarquée sur Arduino Due.

Le principe repose sur la capacité d'un réseau de neurones à apprendre à discriminer différentes classes à partir de motifs cachés dans les données d'entrée. Dans notre cas, les entrées sont des matrices de MFCCs extraits à partir d'un enregistrement vocal d'une seconde, soit environ 61 frames, chacune contenant 13 coefficients.

Pour ce faire, nous avons donc constitué un jeu de données composé de 100 enregistrements vocaux, répartis équitablement entre les deux mots que nous avons décidés de cibler, qui sont « Lampe » et « Rideau ». Chaque enregistrement a été transformé en une matrice de coefficients MFCC de dimension [61 × 13], représentant l'évolution temporelle du timbre vocal. Afin de rendre ces données compatibles avec les couches convolutionnelles de Keras, elles ont été reformées sous le format [61, 13, 1], en ajoutant une dimension de profondeur. Un jeu de test indépendant, composé de 10 échantillons également équilibrés entre les deux classes, a été utilisé pour évaluer la performance du modèle.

Pour la classification, nous avons conçu un réseau de neurones convolutionnel léger, spécifiquement adapté aux contraintes de calcul d'un microcontrôleur. L'entrée du modèle correspond donc à l'image MFCC de dimension (100, 61, 13) que nous avons créé. Le réseau commence par une couche convolutionnelle avec 8 filtres de taille 3×3 et une activation ReLU, suivie d'une opération de

sous-échantillonnage via MaxPooling2D. Une seconde couche convolutionnelle, cette fois avec 16 filtres et également suivie d'un MaxPooling2D, extrait des caractéristiques plus complexes. La sortie est ensuite aplatie, puis transmise à deux couches denses successives de 32 et 8 neurones, toutes deux activées par la fonction ReLU. Enfin, une couche de sortie à un seul neurone avec activation sigmoid permet de produire une probabilité, utilisée pour la classification binaire entre les deux mots cibles.

Nous avons entraîné notre réseau de neurones à l'aide de la fonction de perte MSE et l'optimiseur Adam, pendant 200 époques avec une patience d'arrêt anticipé de 10. Et nous avons également vérifié la robustesse du modèle échantillon par échantillon, grâce au fait que chaque prédiction donne une probabilité comprise entre 0 et 1. Pour ce faire, nous avons fixé un seuil à 0.5, qui délimite la prédiction entre nos 2 mots, puisqu'une prédiction supérieure est interprétée comme « Rideau », inférieure comme « Lampe ».

Enfin, pour finir, nous avons extrait les poids du modèle sous forme de tableaux C à l'aide du même script Python, pour une intégration lors du module 6 sur la carte Arduino Due.

F. FP6 : Classifier les enregistrements audios

L'objectif de ce module est de mettre en œuvre l'exécution du réseau de neurones embarqué sur la carte Arduino Due, afin de classifier un mot prononcé en temps réel à partir de ses coefficients MFCC. En d'autres termes, il s'agit ici de faire fonctionner la chaîne complète de reconnaissance vocale, depuis la capture vocale jusqu'à l'exécution de la commande d'éléments vocale (LEDs et animations écran OLED).

À la suite de l'apprentissage du modèle de classification sur ordinateur (FP5), les poids et biais du réseau de neurones ont été extraits sous forme de tableaux statiques et intégrés directement dans le programme Arduino sous forme de tableaux PROGMEM. Nous avons utilisé une architecture dense simple (MLP), suffisante pour distinguer deux mots (« Lampe » et « Rideau ») tout en respectant les contraintes de mémoire du microcontrôleur.

Le réseau implémenté possède :

- Une entrée de dimension 144 (correspondant à 61 frames de 13 MFCCs aplatis)
- Une couche cachée de 8 neurones avec activation ReLU
- Une sortie unique, avec une activation sigmoïde pour effectuer une classification binaire

L'évaluation repose sur l'utilisation de la bibliothèque NeuralNetwork.h, qui permet d'exécuter efficacement un MLP en exploitant ses poids et biais. Le processus commence par l'appui sur un bouton poussoir, déclenchant un enregistrement audio d'une seconde via le microphone MAX9814. Une fois le signal acquis, l'Arduino Due applique directement l'algorithme MFCC à l'aide de la fonction computeNormalizedMFCCs, permettant d'extraire les caractéristiques essentielles du signal vocal sous forme de coefficients MFCC.

Ces coefficients sont ensuite transférés vers le réseau de neurones embarqué, qui les classe et identifie le mot prononcé. En fonction du mot détecté, l'Arduino commande l'activation des LEDs et l'animation sur l'écran OLED. La LED jaune s'allume si le mot est "Lampe" et la LED rouge si le mot est "Rideaux". Par ailleurs, l'affichage animé de la maison réagit à la commande vocale, activant l'animation du soleil ou des rideaux, selon le mot reconnu.

Ce processus garantit un traitement en temps réel optimisé, permettant une reconnaissance vocale embarquée efficace et intégrée à une interface interactive. Le système est ensuite prêt à recommencer, en attente d'un nouvel appui sur le bouton poussoir.

VII. Tests et validation

Une section au moins aussi importante que celle sur le développement.

Il est question ici de montrer les performances techniques du système et de valider le développement module par module puis au global (intégration) en accord avec la partie IV.

Chaque résultat (bien souvent des courbes) doit être décrit comme suit :

- Ce qui a été fait ;
- Ce que l'on est censé obtenir et critère de réussite du test ;
- Ce que l'on obtient ;
- Conclusion : validation ou non du bon fonctionnement du module.

A. FP1 : Numériser le signal audio

Dans ce premier module, l'objectif est de valider que le système est capable de numériser un signal audio provenant du microphone MAX9814 à une fréquence d'échantillonnage fixe et précise de 32 kHz. Cette fréquence est essentielle pour garantir une qualité suffisante du signal vocal numérisé, permettant une analyse et un traitement efficaces par la suite.

Pour assurer un échantillonnage à 32 kHz, l'ADC de la carte Arduino Due a été configurée en mode interruption sur Timer, permettant de déclencher automatiquement une conversion à intervalle régulier, sans intervention active du programme principal.

La période théorique d'échantillonnage est donnée par :

$$T = \frac{1}{f} = \frac{1}{32000} = 31,25\ \mu s$$

Nous avons programmé un Timer Counter pour générer une interruption toutes les 31,25 µs. Lors de chaque interruption, une lecture est effectuée via l'ADC, puis la valeur est stockée dans un buffer circulaire.

Afin de valider expérimentalement que la fréquence d'échantillonnage était correctement respectée, nous avons converti les données en signal analogique via le DAC de l'Arduino, puis visualiser ce signal sur un oscilloscope.

Comme on peut observer sur la figure 5, nous avons mesuré grâce à l'oscilloscope la période du signal (notée *Prd[1]*), et avons trouvé qu'il est de 30,48 µs. Cela correspond à une fréquence mesurée:

$$f_{mesurée} = \frac{1}{T} = \frac{1}{30.48\times10^{-6}} \approx 32.81kHz$$

Cette valeur est parfaitement cohérente avec l'affichage de l'oscilloscope (Freq[1] = 32.81 kHz) et très proche de la valeur attendue de 32 kHz.

Nous avons également vérifié la condition de Nyquist. Avec une fréquence d'échantillonnage de 32.81 kHz, la fréquence maximale théorique du signal pouvant être reconstruite sans aliasing est :

$$f_{max} = \frac{f_s}{2} = \frac{32.81}{2} = 16.405kHz$$

Ce seuil reste largement supérieur à la bande utile de la voix humaine (≤ 8 kHz), assurant ainsi une captation fidèle du signal vocal sans repliement fréquentiel.

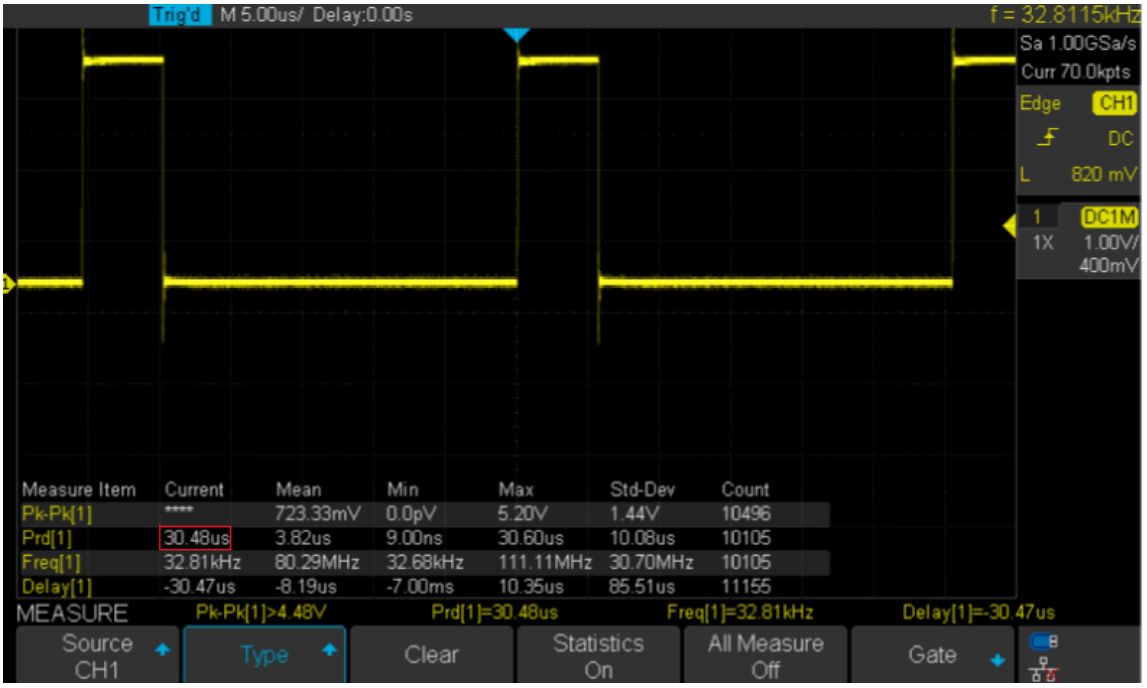


Figure 5 : Signal de sortie du DAC visualisé à l'oscilloscope

Pour conclure, le test réalisé montre que la fréquence d'échantillonnage effective est très proche de la valeur cible, avec une erreur relative inférieure à 3%. La condition de Nyquist est respectée, garantissant une acquisition correcte du signal vocal sans artefact de repliement.

Ce module FP1 est donc validé, avec une acquisition fiable, stable et conforme aux exigences spécifiées pour le traitement ultérieur.

B. FP2 : Conditionnement du signal

Ce module a pour objectif de préparer le signal audio numérisé pour une analyse optimale par le réseau de neurones. Plus précisément, il s'agit d'atténuer les fréquences supérieures à 4 kHz d'au moins 30 dB et de procéder à une réduction de la fréquence d'échantillonnage de 32 kHz à 8 kHz (down-sampling). Ces étapes permettent de limiter le repliement spectral tout en assurant la conservation des informations vocales essentielles.

Pour respecter l'exigence d'atténuation au-delà de 4 kHz, nous avons conçu un filtre passe-bas de type RIF à l'aide de Python, en utilisant la méthode de la fenêtre de Hamming via la bibliothèque SciPy.

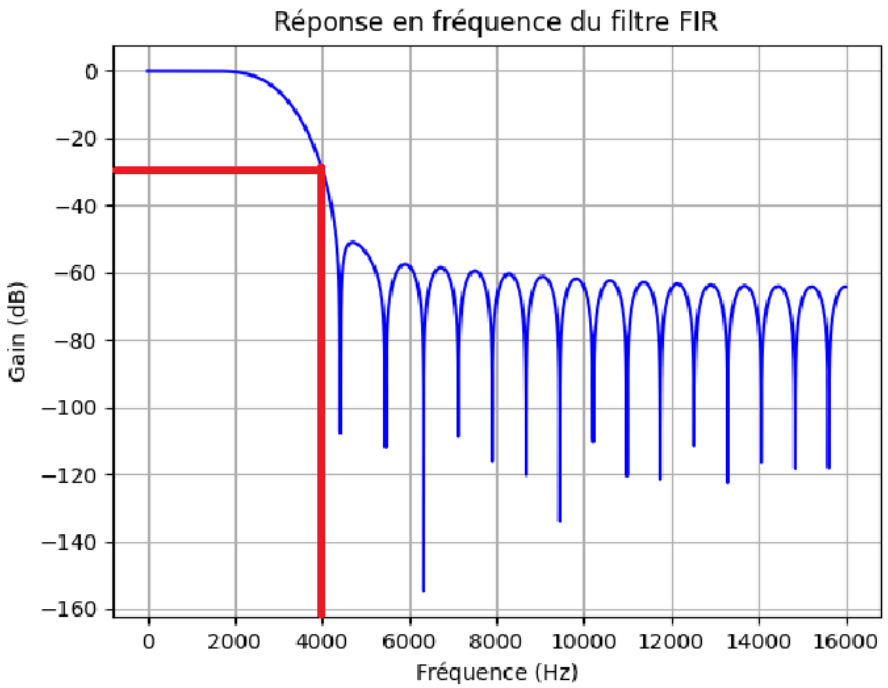
Les coefficients générés ont été intégrés dans le code sur la carte Arduino Due. L'implémentation repose sur un buffer circulaire de 61 valeurs, mis à jour à chaque échantillon ADC (32 kHz). Le filtrage temps réel s'effectue par convolution entre le buffer et les coefficients.

Après filtrage, un échantillon sur quatre est conservé, ce qui réduit la fréquence à 8 kHz tout en évitant les artefacts liés au repliement spectral.

Par ailleurs, il est impératif que le temps de traitement d'un échantillon soit inférieur à 31 µs (correspondant à la période d'échantillonnage). Nous avons donc mesuré le temps réel de traitement via la fonction micros() dans la boucle d'interruption ADC.

On peut observer sur le graphique de la figure 6, que le gain atteint -30 dB à 4 kHz, comme attendu. Les fréquences comprises entre 4 kHz et 8 kHz présentent une atténuation progressive atteignant -60 dB, ce qui assure une forte atténuation des hautes fréquences.

Ces résultats confirment que le filtre RIF est bien conçu et remplit l'exigence d'atténuation nécessaire pour éviter le repliement spectral lors du down-sampling.



- *Figure 6: Graphique représentant le Gain du filtre RIF en fonction de la fréquence*

De plus, on peut observer sur la figure 7 que nous avons mesuré le temps de traitement total (mise à jour du buffer + application du filtre) sur 11 cycles. Le temps maximal mesuré est de 21 µs, ce qui est bien inférieur à la limite critique de 31 µs. Ce résultat indique que le filtrage est effectué en temps réel, avec une marge de sécurité de 10 µs, garantissant la stabilité du système à haute fréquence d'échantillonnage.

```
=== Données filtrées ===
Durée max de traitement : 21 µs
=== Données filtrées ===
Durée max de traitement : 21 µs
=== Données filtrées ===
Durée max de traitement : 21 µs
=== Données filtrées ===
Durée max de traitement : 21 µs
=== Données filtrées ===
Durée max de traitement : 21 µs
=== Données filtrées ===
Durée max de traitement : 21 µs
=== Données filtrées ===
Durée max de traitement : 21 µs
=== Données filtrées ===
Durée max de traitement : 21 µs
=== Données filtrées ===
Durée max de traitement : 21 µs
=== Données filtrées ===
Durée max de traitement : 21 µs
=== Données filtrées ===
Durée max de traitement : 21 µs
```

- *Figure 7 : Copie d'écran du temps de conversion affichée sur la console sur 11 cycles*

Notre filtre RIF implémenté respecte donc parfaitement les contraintes de gain et de fréquence de coupure. Son intégration sur Arduino Due s'effectue efficacement, avec un temps de traitement maîtrisé. La réduction d'échantillonnage à 8 kHz peut donc s'opérer sans aliasing, grâce à l'atténuation suffisante des hautes fréquences.

Le module FP2 est donc pleinement validé, tant sur le plan fonctionnel que temporel, et constitue une base solide pour l'analyse vocale par le réseau de neurones.

C. FP3 : Écouter et valider l'enregistrement

Dans ce module, l'objectif est de valider la qualité du signal audio capturé et conditionné par les modules précédents (FP1 et FP2), en s'assurant qu'il est audible, clair et exploitable. Plus précisément, il s'agit de vérifier que le système permet d'enregistrer et restituer de manière intelligible le mot « Électronique » à l'aide du microphone MAX9814, l'ADC de l'Arduino Due et le logiciel Audacity.

Pour ce faire, le port série de l'Arduino Due a été configuré à une vitesse élevée de 460800 bauds afin de limiter les risques de saturation lors du transfert. Lorsque le buffer atteint 100 échantillons, ceux-ci sont envoyés en binaire, octet par octet (LSB puis MSB), assurant une reconstruction correcte du signal.

Les données ont été réceptionnées sur PC via un terminal série, puis enregistrées dans un fichier brut (.bin). Un script Python a été utilisé pour convertir ce fichier en format .wav, ensuite importé dans Audacity pour une analyse visuelle et auditive.

Après enregistrement et conversion, le fichier audio obtenu a été importé dans Audacity. L'analyse de la forme d'onde ainsi que l'écoute directe permettent d'attester d'un enregistrement net, sans distorsion ni bruit de fond notable.

Comme on peut écouter en cliquant sur le lien de la figure 8, le mot « Électronique » est parfaitement identifiable, avec une qualité sonore qui confirme le bon fonctionnement de toute la chaîne d'acquisition.

electronique.wav

- *Figure 8 : Lien d'écoute du mot « Électronique » enregistré*

Ce test valide avec succès l'ensemble de la chaîne d'acquisition, depuis le microphone jusqu'au transfert des données audio, en passant par l'ADC, le filtrage numérique et le down-sampling. La qualité sonore obtenue dans Audacity confirme que les modules FP1 et FP2 fonctionnent correctement.

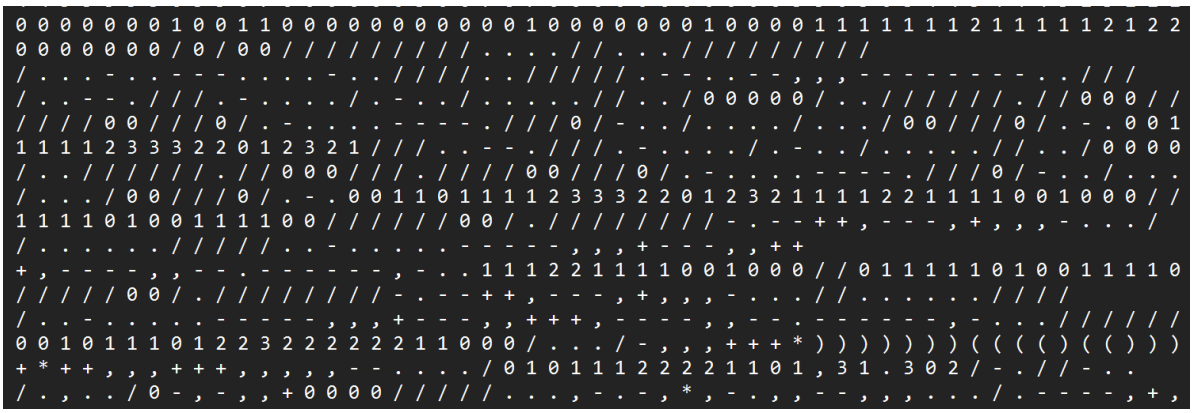
Le module FP3 est donc pleinement validé, garantissant que les signaux audio capturés sont fiables, exploitables, et prêts pour les prochaines étapes de caractérisation MFCC.

D. FP4 : Caractériser le timbre vocal

Ce module a pour objectif d'analyser et de caractériser le timbre vocal à partir d'un signal audio. Pour cela, nous devons séparer le signal audio en frames de 256 échantillons chacune. Un algorithme joue ce rôle, il permet également, grâce à ces échantillons, de générer des coefficients appelés MFCCs.

Pour valider la bonne séparation des échantillons, nous utilisons le logiciel PuTTY. Ce logiciel nous permet d'accéder à un port de notre ordinateur, dans lequel se trouvent les échantillons une fois l'algorithme lancé.

Lorsque l'algorithme est lancé, les frames audio découpées sont envoyées et s'affichent sous forme binaire dans PuTTY. Un extrait de cette sortie est visible ci-dessous.



- *Figure 9 : Aperçu des échantillons sous forme binaire dans la console PuTTY*

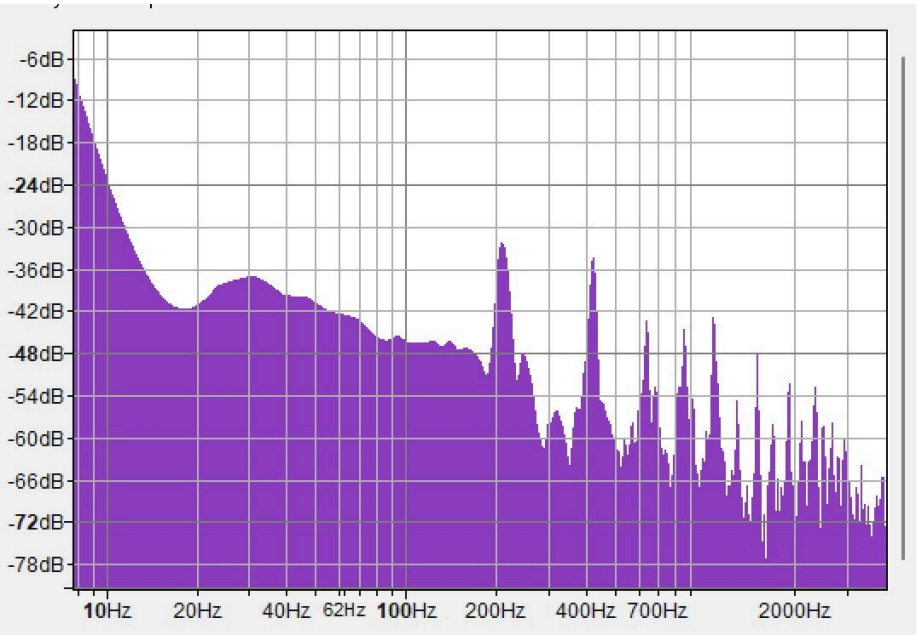
Ce qui est affiché par PuTTY correspond à notre signal découpé, pour autant, il est impossible de le comprendre sous cette forme. C'est pourquoi nous utilisons un code python qui va traduire cette forme binaire en nombres. Ci-dessous, nous pouvons voir la sortie réelle de notre algorithme MFCC.

```
7936
7424
7168
6912
6656
6400
6144
6144
5888
5888
6144
6400
6400
6400
6656
6912
7168
7168
7424
7680
7680
7680
7680
7936
7680
7680
7680
7680
7680
```

- *Figure 10 : Sortie de l'algorithme MFC sous forme de nombres*

Cette sortie correspond aux valeurs brutes de notre signal audio une fois reconstruites. Elle confirme le bon fonctionnement de l'étape de découpage en échantillons, ainsi que l'acquisition du signal via le microphone et l'envoi série vers l'ordinateur.

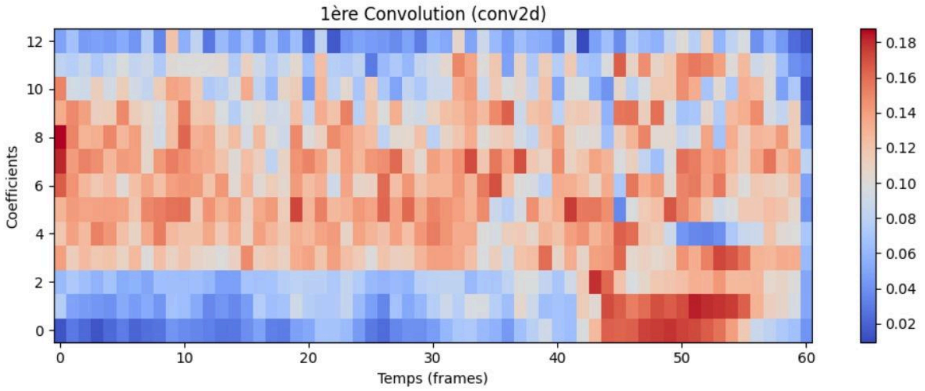
Il est également nécessaire de vérifier le signal audio. Pour se faire, nous regardons son spectre sur Audacity. Pour être en adéquation avec notre projet, nous allons tracer le spectre du mot "Lampe".



- *Figure 11 : Spectre de Fourier du signal « lampe » à 1 kHz obtenu avec Audacity*

Le spectre obtenu est conforme, on voit bien qu' il montre une présence marquée dans les basses et moyennes fréquences, ce qui est attendu pour un signal vocal. Cela nous permet de valider le contenu audio et de continuer son traitement.

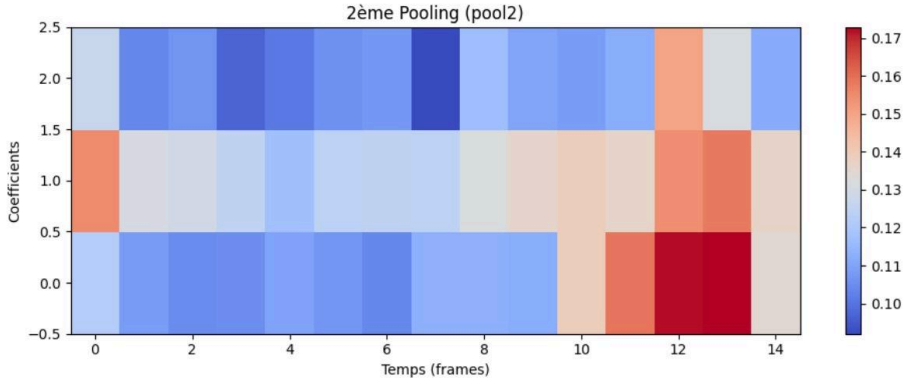
Une deuxième partie importante de ce module est la génération de 13 MFCCs pour un signal audio d'une seconde. Pour implémenter ceci, il faut filtrer le signal audio. Le signal doit parcourir différentes transformations. La première est une convolution qui a pour but d'extraire des motifs locaux dans les coefficients.



- *Figure 12 : Spectre du signal après la convolution*

Ce spectre montre la répartition des coefficients MFCC en fonction du temps. Même si cette répartition reste relativement homogène, on observe des débuts de structures dans les coefficients, ce qui indique que le traitement commence à être efficace.

Pour améliorer encore cette représentation, nous appliquons un deuxième pooling, qui permet de réduire la dimension des données tout en conservant les informations pertinentes.



- *Figure 13 : Spectre des coefficients après le deuxième Pooling*

En regardant ce spectre, nous observons une nette amélioration des coefficients par rapport à précédemment. La répartition est bien plus centrée sur un seul endroit, ce qui prouve la validité de notre filtre.

Cette étape FP4 nous a permis de valider le bon découpage du signal audio, la cohérence de son spectre fréquentiel, ainsi que la qualité de l'extraction des MFCCs à travers des traitements

convolutionnels et de pooling. Les spectres obtenus confirment la pertinence de notre chaîne de traitement, et les coefficients sont prêts à être utilisés pour la classification dans les étapes suivantes.

E. FP5 : Identifier les résultats attendus

Ce module a pour objectif de classifier automatiquement un objet sonore, ici “Lampe” et “Rideau”, à partir des MFCCs extraits d’un signal audio. Pour cela, nous avons recours à un réseau de neurones convolutif construit et entraîné avec TensorFlow.

Le réseau prend en entrée une structure de donnée de dimension (48, 13, 1) correspondant à une seconde de signal audio traitée sous forme de MFCCs. Une fois le modèle entraîné, il est capable de prédire une valeur entre 0 et 1, que l’on interprète comme une probabilité que l’objet entendu soit une "Lampe" (proche de 1) ou un "Rideau" (proche de 0).

L'entraînement du modèle a été réalisé à l’aide d’un jeu de données de 100 échantillons (50 Rideaux et 50 Lampes). L’évaluation a ensuite été faite sur 10 données tests équilibrées. La structure du réseau est constituée de deux couches de convolution avec pooling, suivies de couches denses.

Une fois le modèle entraîné, nous avons mesuré l'erreur quadratique moyenne (MSE) sur les données de test pour valider ses performances. La sortie du script Python ci-dessous montre les prédictions effectuées sur chaque échantillon de test, comparées aux valeurs attendues.

```
MSE test = 0.030003

--- Classification testdata détaillée ---
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step
idx= 0 pred=0.000508 → Rideau (attendu=0.0)
idx= 1 pred=0.000645 → Rideau (attendu=0.0)
idx= 2 pred=0.000165 → Rideau (attendu=0.0)
idx= 3 pred=0.001355 → Rideau (attendu=0.0)
idx= 4 pred=0.004000 → Rideau (attendu=0.0)
idx= 5 pred=0.999590 → Lampe (attendu=1.0)
idx= 6 pred=0.997327 → Lampe (attendu=1.0)
idx= 7 pred=0.997935 → Lampe (attendu=1.0)
idx= 8 pred=0.999448 → Lampe (attendu=1.0)
idx= 9 pred=0.998768 → Lampe (attendu=1.0)
```

- *Figure 14 : Capture d’écran des résultats de la classification avec les prédictions du modèle*

Dans cette figure, on observe que toutes les prédictions sont correctes. On peut observer que les cinq premières valeurs sont proches de 0, correspondant au mot "Rideau", tandis que les cinq suivantes sont proches de 1, correspondant au mot "Lampe". Le MSE obtenu est de 0.030003, ce qui est largement inférieur au seuil de 0.05 fixé. Cela valide la capacité du modèle à distinguer les deux classes de manière fiable.

Afin de rendre ce modèle utilisable dans un environnement embarqué, nous avons constitué un tableau contenant les MFCCs extraits des 100 enregistrements audio d'entraînement. Chaque entrée du tableau correspond à un signal vocal d'une seconde, transformé en une matrice de 61 frames de 13 coefficients MFCC, soit une structure à trois dimensions de taille [100][61][13].

Ce tableau training a permis de constituer une base d'entraînement du modèle. Nous avons choisis de prendre 61 frames au lieu de 256, afin d'avoir une durée de l'audio traitée plus courte (environ 1 seconde au lieu de 4) et pour rendre le réseau beaucoup plus adapté à un microcontrôleur comme l'Arduino Due, qui a peu de RAM et de puissance de calcul.

```
const float training[100][61][13] = {
{
{0.376697, 0.012885, 0.510338, 0.453832, 0.555932, 0.598886, 0.626651, 0.797105, 0.425664, 0.973252, 0.643184, 0.518570, 0.330356},
{0.185323, 0.029679, 0.431023, 0.440088, 0.522872, 0.769533, 0.702816, 0.705302, 0.607976, 0.499242, 0.600723, 0.489172, 0.608002},
{0.176695, 0.073817, 0.523996, 0.433511, 0.700103, 0.656673, 0.659913, 0.893581, 0.596963, 0.641280, 0.604151, 0.361071, 0.312416},
{0.093040, 0.000000, 0.200372, 0.314089, 0.506820, 0.669488, 0.849754, 0.966196, 0.605006, 0.425490, 0.721517, 0.495644, 0.375202},
{0.101247, 0.074050, 0.483739, 0.395795, 0.688589, 0.587210, 0.718751, 0.812215, 0.232542, 0.742119, 0.520371, 0.325688, 0.334507},
{0.149621, 0.072548, 0.374617, 0.483679, 0.539274, 0.617394, 0.548598, 0.695738, 0.218745, 0.394486, 0.754550, 0.344449, 0.204819},
{0.113570, 0.127910, 0.345669, 0.338908, 0.528545, 0.881127, 0.692299, 0.658566, 0.308535, 0.382076, 0.638534, 0.085457, 0.377648},
{0.086757, 0.173397, 0.399412, 0.458582, 0.537677, 0.644085, 0.754224, 0.967047, 0.461905, 0.445797, 0.735680, 0.396751, 0.126614},
{0.051494, 0.173293, 0.395320, 0.431167, 0.555028, 0.685888, 0.770052, 0.946895, 0.446595, 0.528227, 0.728742, 0.222120, 0.348914},
```

- *Figure 15 : Extrait du tableau training contenant les MFCCs des 100 échantillons audio*

Une fois l'entraînement réalisé avec ce jeu de données, nous avons extrait les paramètres appris par le réseau, à savoir les poids et biais de chaque couche. Ces données ont été exportées sous forme de tableaux (all_weights et all_biases) à l'aide d'un script Python.

Cette étape nous a permis de créer les tableaux all_weights et all_biases, qui seront ensuite intégrés dans l'Arduino Due, permettant ainsi à l'appareil d'exécuter la prédiction vocale localement, sans avoir besoin de recalculer les poids ni de réentraîner le modèle. Nous avons donc réussi à réaliser ce module FP5.

F. FP6 : Classifier les enregistrements audios

L'objectif de ce module est de démontrer que notre réseau de neurones est capable de distinguer correctement les deux mots "Lampe" et "Rideau", à partir d'un enregistrement vocal transformé en MFCC. Pour valider le fonctionnement, nous devons atteindre une précision d'au moins 95 % sur un jeu de test de 10 échantillons, soit un taux d'erreur inférieur à 5 %.

Après avoir entraîné le modèle sur ordinateur avec TensorFlow, nous avons exporté les poids et biais du réseau vers des tableaux all_weights[] et all_biases[]. Ces paramètres ont été intégrés dans le programme Arduino et permettent d'exécuter localement le réseau de neurones.

L'Arduino enregistre un échantillon audio d'une seconde, puis extrait les coefficients MFCC correspondants. Ceux-ci sont passés dans un réseau convolutionnel dans le fichier cnn.cpp qui applique successivement une couche de convolution, un pooling et un flatten. Les données aplaties sont ensuite transmises à un réseau dense pré-entraîné, qui retourne une probabilité indiquant s'il s'agit du mot "Lampe" ou "Rideau".

```
Sample 0 | Prédiction: 0.7985 | Attendu: 0.00 (Rideau) | MSE: 0.637610
Sample 1 | Prédiction: 0.8021 | Attendu: 0.00 (Rideau) | MSE: 0.643346
Sample 2 | Prédiction: 0.8095 | Attendu: 0.00 (Rideau) | MSE: 0.655236
Sample 3 | Prédiction: 0.8129 | Attendu: 0.00 (Rideau) | MSE: 0.660777
Sample 4 | Prédiction: 0.8053 | Attendu: 0.00 (Rideau) | MSE: 0.648527
Sample 5 | Prédiction: 0.8259 | Attendu: 1.00 (Lampe)  | MSE: 0.030299
Sample 6 | Prédiction: 0.8079 | Attendu: 1.00 (Lampe)  | MSE: 0.036912
Sample 7 | Prédiction: 0.8029 | Attendu: 1.00 (Lampe)  | MSE: 0.038843
Sample 8 | Prédiction: 0.8032 | Attendu: 1.00 (Lampe)  | MSE: 0.038723
Sample 9 | Prédiction: 0.7950 | Attendu: 1.00 (Lampe)  | MSE: 0.042032
========================
MSE finale sur testData[] : 0.343230
```

- *Figure 16 : Capture d'écran des résultats de la classification avec les prédictions du modèle sur l'Arduino*

Les résultats de prédiction du réseau embarqué sur Arduino ne sont pas cohérents avec les étiquettes attendues. Comme le montre la figure ci-dessus, tous les échantillons produisent une prédiction proche de 0.8, ce qui conduit à la reconnaissance systématique du mot "Lampe", y compris lorsque la commande réelle était "Rideau".

Cette incohérence se reflète dans l'erreur quadratique moyenne (MSE) finale, qui atteint environ 0.34, contre seulement 0.03 lors des tests sur ordinateur avec Python. Cela met en évidence un écart significatif entre le comportement du modèle en simulation et son exécution embarquée.

Bien que la version logicielle côté python ait démontré que le réseau pouvait atteindre une excellente précision, l'intégration sur Arduino n'a pas donné les résultats voulus. En conséquence, l'objectif de l'ET9 n'a pas pu être pleinement validé.

VIII. Bilan

A. État d'avancement

Où en est le projet ? A-t-on atteint les objectifs ?

Quels modules restent à finaliser (ou à perfectionner pour être en accord avec les spécifications techniques) ?

Le projet Neural Speech est globalement bien avancé et la plupart des objectifs techniques définis dans le cahier des charges ont été atteints. L'échantillonnage à 32 kHz a été correctement mis en œuvre via l'ADC de l'Arduino Due, avec un filtrage numérique efficace permettant un down-sampling à 8 kHz, tout en respectant le temps de traitement imposé de moins de 31 µs par échantillon. L'enregistrement audio est clair et lisible sur Audacity, validant la qualité du signal numérisé et filtré. Le traitement des données, de la séparation en frames à l'extraction des 13 coefficients MFCC par frame, a également été correctement exécuté, ce qui nous a permis d'entraîner un réseau de neurones convolutionnel avec succès. Sur Python, les performances obtenues sur les mots « rideau » et « lampe » sont satisfaisantes, avec une MSE inférieure à 0,05, conformément aux exigences.

Cependant, un blocage majeur persiste avec la classification sur la carte Arduino Due elle-même. Bien que l'entraînement sous TensorFlow fournisse des résultats fiables en simulation, l'implémentation des poids dans l'environnement embarqué via la librairie NeuralSpeech, comme vu en TP, ne permet pas d'obtenir une classification correcte des mots ciblés. Les performances chutent significativement sur Arduino, avec une MSE qui augmente et une détection erronée, ce qui nous empêche de déclencher les commandes vocales prévues. Ce problème, malgré une semaine complète d'investigations, reste non résolu à ce jour. Il concerne probablement un mauvais encodage ou formatage des poids du modèle, ou une incompatibilité entre la structure du réseau entraîné et celle interprétée sur l'Arduino.

Ainsi, bien que tous les modules fonctionnels aient été développés, il reste à finaliser la partie de classification embarquée. L'objectif à court terme est d'assurer une correspondance stricte entre le modèle entraîné en Python et sa réplique sur la carte, pour que la reconnaissance vocale soit pleinement opérationnelle et respecte les contraintes d'exécution sur l'Arduino Due.

B. Pertinence de la solution technique

Quelles sont les limites techniques de la solution développée ?

Quelles sont les possibilités d'évolution ou de poursuite ?

La solution actuelle, bien que opérationnelle sur de nombreux aspects, présente plusieurs limites techniques, principalement liées à l'implémentation embarquée du réseau de neurones. La principale faiblesse réside dans la perte de performance du modèle lors de son passage de l'environnement Python à l'environnement Arduino. Cela s'explique en partie par les ressources limitées de la carte Arduino Due, tant en mémoire qu'en puissance de calcul, qui rendent difficile la prise en charge de modèles de classification vocale complexes. De plus, l'absence d'outils de débogage détaillés sur Arduino complique l'analyse de l'erreur entre la version simulée et celle embarquée. L'optimisation du traitement des poids du réseau et leur quantification pour une architecture avec autant de contraintes reste donc une limite technique importante.

Pour faire évoluer le projet, plusieurs pistes d'amélioration peuvent être envisagées. Tout d'abord, l'intégration d'une carte embarquée plus performante, permettrait de gérer un réseau plus complexe tout en conservant des temps de réponse acceptables. Une autre voie possible serait d'optimiser l'architecture du CNN utilisé, en explorant par exemple des modèles compressés ou quantifiés comme ceux issus de TensorFlow Lite. Enfin, pour renforcer l'expérience utilisateur et rendre le système autonome, il serait intéressant d'ajouter une détection automatique de la voix sans bouton d'activation. Ce type de fonctionnalités permettrait de tendre vers un assistant vocal embarqué plus réactif, ergonomique et proche de solutions que l'on connaissait tous, comme l'assistant vocal Alexa ou encore Siri sur iphone.

C. Bilan sur le travail d'équipe

Qu'avez-vous appris individuellement ? Quelles compétences vont pouvoir être mises en avant lors de votre prochaine recherche de stage ?

Comment l'équipe aurait pu mieux s'organiser ? Proposer un plan d'action pour le prochain projet.

Ce projet nous a permis de renforcer nos compétences en traitement du signal, en particulier sur l'acquisition et le filtrage audio en temps réel, mais aussi en intelligence artificielle avec l'entraînement et l'implémentation de réseaux de neurones. Nous avons également développé une meilleure compréhension des contraintes du calcul embarqué, notamment en termes d'optimisation du code et de gestion de la mémoire.

Concernant le travail en équipe, notre organisation était plutôt efficace avec une répartition claire et anticipée des tâches dès le début du projet. Pour les projets à venir, il serait pertinent de définir un calendrier précis avec des points d'avancement hebdomadaires, et d'utiliser un outil collaboratif pour le suivi des tâches.

IX. Bibliographie

[1] Dr Schneider, Maxime, «La Toolbox,» [En ligne]. Available: https://boostcamp.omneseducation.com/pluginfile.php/3538704/mod_resource/content/1/RAPPORT.pdf. [Accès le 14 juin 2024].

[2] IEEE, «IEEE Editorial Style Manual,» 29 février 2024. [En ligne]. Available: https://journals.ieeeauthorcenter.ieee.org/your-role-in-article-production/ieee-editorial-style-manual/. [Accès le 14 juin 2024].

[3] Nicholas Renotte, YouTube, Tensorflow Tutorial for Python in 10 Minutes

Cette section doit contenir tous les documents utilisés et sites internet consultés pour développer le projet. La syntaxe à utiliser (le standard bibliographique IEEE) est détaillée dans le document « Comment rédiger un rapport » [1].

Lorsqu'ils sont référencés dans le corps de texte du rapport, un renvoi numéroté doit apparaître, conformément au standard bibliographique IEEE [2].

X. Annexes

Documents annexes, éventuels codes (**pas de code dans le rapport**).