École Nationale Supérieure d'Ingénieurs du Mans
Le Mans Université

Projet 5A : TinyML pour l'IA embarquée

Rapport de projet

Réalisé par :
NAIT HAMMOU Mehdi
BEN BRAHIM Abdelkarim

Encadré par :
Pr. HASSAN Kais

Jury :
Pr. HASSAN Kais
Mme. COUINEAUX Audrey

Année Universitaire : 2021/2022

Résumé

Tiny Machine Learning (TinyML) ou Machine learning pour les systèmes embarqués est devenu un sujet de recherche qui prend de plus en plus d'importance, car il cible l'implémentation d'algorithmes d'intelligence artificielle (IA) sur des appareils à faible puissance de calcul. Depuis longtemps, l'usage de l'IA demandait beaucoup de ressources de calcul. Les recherches dans le TinyML ont abouti à des méthodes d'optimisation d'algorithmes de l'IA pour ces appareils. Ce qui permet de réduire le besoin d'envoyer les données des capteurs par le réseau pour les traitements et la récupération des résultats. Ce qui permet d'avoir des systèmes moins sensibles aux problèmes de latence sur le réseau, ainsi qu'aux risques de sécurité liés à l'approche classique.

Le but du présent projet étant d'aboutir à une preuve de concept (PoC) de l'usage de cette technologie, ce qui va être effectué en proposant le développement et le déploiement d'une application de reconnaissance d'activité humaine à l'aide des données d'un accéléromètre et gyroscope montés sur le bras de l'utilisateur. Le présent document représente les résultats de la première partie du projet. Dans cette partie, le but est d'étudier et de comparer plusieurs architectures de réseaux de neurones, avant de les optimiser pour les adapter aux spécificités des cartes cibles (Arduino Nano BLE 33 Sense STLKT01V1 SensorTile).

Mots clés : TinyML, Systèmes embarqués, Intelligence Artificielle, Optimisation.

Abstract

Tiny Machine learning or TinyML has recently become one of the fast evolving topics in research, as it aims to run Artificial intelligence (AI) algorithms on low power embedded systems. Until recently, using AI required significant amounts of computing power, TinyML research resulted in novel methods of optimization of AI algorithms for low computing power devices. This allows these devices to reduce their need to stream the sensor's data through the network to run computations and retrieve results, resulting in systems that are less prone to latency issues and security threats.

The aim of the following project is to provide a proof of concept (PoC) of the use of this technology, through the implementation and deployment of a human activity classification algorithm using accelerometer and gyroscope data. The present document presents the first part of the project, where the aim was to perform a benchmark on multiple neural network architectures, before trying optimization techniques to adapt the recognition models to the specificity of the target devices (Arduino Nano BLE 33 Sense & STLKT01V1 SensorTile).

Keywords : TinyML, Embedded systems ,Artificial Intelligence, Optimization.

Table des matières

Résumé **1**

Abstract **1**

Introduction **4**

1 Contexte Technologique **5**
1.1 Intelligence artificielle 5
1.2 Edge Computing 6
1.3 Tiny Machine Learning 7

2 Présentation du projet **11**
2.1 Problématique 11
2.2 Objectifs du projet 12
2.3 Environnement hardware 12
2.4 Logiciels & frameworks utilisés 12
2.5 PlateformIO 13
2.6 Google Colab 13
2.7 Google TinyMotion Trainer 14

3 Etapes de développement **15**
3.1 Choix des architectures des réseaux de neurones 15
3.1.1 Réseau convolutif à une dimension 16
3.1.2 Réseau LSTM 17
3.2 Collecte des données 18
3.3 Entraînnement des modèles et analyse des résultats 20
3.4 Optimisation du modèle 25
3.5 Déploiement sur la carte Aduino BLE 33 26

Conclusion **28**

Bibliographie **29**

Annexe A : Gestion de projet **30**

Table des figures

1.1 Processus de développement d'un model IA 6
1.2 Architecture réseau Edge/Cloud 7
1.3 Présentation des deux sticks Intel NCS2 et Google Coral 8
1.4 Cycle de développement des algorithmes TinyML 8
1.5 Elagage des réseaux de neurones 8
1.6 Algorithme d'élagage des réseaux de neurones 9
1.7 Quantification des poids des réseaux de neurones 9
1.8 Knowledge distillation : modèle étudiant et enseignant 10

2.1 Exemple des gestes ciblés par la reconnaissance 11
2.2 Arduino Nano BLE 33 Sense 12
2.3 Logo du framework Tensorflow 13
2.4 Logo de l'IDE PlatformIO 13
2.5 Logo de l'environnement Colab 14
2.6 Interface Google TinyMotion trainer 14

3.1 Shématisation des étapes de réalisation du projet 15
3.2 représentation d'une opération de convolution 1D 16
3.3 représentation d'une opération de convolution 2D 16
3.4 Architecture détaillée d'un réseaux convolutif à une dimension 17
3.5 Exemple d'une cellule LSTM 17
3.6 Signal gyroscope et accéléromètre du signe d'accélération 18
3.7 Signal gyroscope et accéléromètre du signe d'arrêt 18
3.8 Signal gyroscope et accéléromètre du signe de ralentissement 18
3.9 Interface tinyMotion Trainer 19
3.10 Visualisation de données des signaux d'accélération 19
3.11 Visualisation de données des signaux de ralentissement 20
3.12 Visualisation de données des signaux d'arrêt 20
3.13 Architecture du réseaux de neurones dense utilisée 20
3.14 Architecture du réseaux de neurones convolutif 21
3.15 courbe de précision du réseau dense utilisé 21
3.16 courbe de précision du réseau dense utilisé 22
3.17 courbes des résultats de la cross-validation sur le réseau dense 22
3.18 courbes des résultats de la cross-validation sur le réseau convolutif 23
3.19 Matrice de confusion du réseau dense 23
3.20 matrice de confusion du réseau convolutif 24
3.21 Précision du modèle avant et après le grid search 25
3.22 Résultat de l'opération d'optimisation 25
3.23 Taille du fichier .h généré 26
3.24 Schématisation du fonctionnement de l'application 26
3.25 Visualisation du résultat d'exécution du modèle 26

3.26 Visualisation de la sortie sur le modèle du carrefour 27
3.27 Tableau Trello utilisé pour la gestion du projet 31
3.28 Diagramme de Gantt du projet 32

Introduction

L'essor des technologies d'intelligence artificielle a permis de proposer des applications adressant des besoins très spécifiques inatteignables par les paradigmes de la programmation classique. Ainsi, nous avons vu ces technologies s'intégrer de plus en plus à nos vies de tous les jours. Maisons connectées, outils d'accessibilité sur les téléphones mobiles, bracelets et montres connectés, sont quelques applications parmi d'autres qui proposent des fonctionnement adaptés à l'utilisateur en fonction de ses habitudes. Ces habitudes prennent une dimension plus concrète sous forme de flux de séries temporelles. Ces données sont étudiées par les algorithmes d'apprentissage pour proposer l'expérience la plus adaptée et la plus personnalisée en fonction de chaque utilisateur.
Dans ce projet, on se propose d'étudier une de ces applications, à travers une application destinée à la reconnaissance des gestes effectués par les utilisateurs. Dans un premier temps, nous allons situer le contexte technologique, avant de décrire les méthodes et les types de données utilisés pour reconnaître et recenser ces gestes. Ensuite, nous présenterons les différentes étapes et choix technologiques qu'on était ammenés à effectuer. Finalement, on discutera les résultats de l'étude ainsi que les différents problèmes rencontrés, en proposant des approches à adopter au cours de la deuxième partie de la réalisation du projet.

Chapitre 1

Contexte Technologique

1.1 Intelligence artificielle

La démocratisation des algorithmes d'intelligence artificielle constitue l'un des événements qui ont le plus marqué l'évolution technologique. Ce phénomène est principalement dû à l'avancée technologique au niveau hardware et à la baisse des prix des composants de calcul tels que les processeurs graphiques. Cette baisse peut être tracée aux usages des GPUs dans le domaine du divertissement, et du gaming en particulier. Ainsi, ces composants offrants des capacités de calcul parallèles étaient accessibles, ce qui a permis une évolution très rapide du domaine de l'IA qui a vu le jour dans les années 1950. Les limitations rencontrées à l'époque en termes de tailles de données et faiblesse des composants de calculs présentaient ce qu'on appelle des "bottleneck" qui ralentissaient considérablement la recherche dans ce domaine.

Le principe de fonctionnement des algorithmes d'intelligence artificielle n'étant plus secret à personne, il n'est pas aussi simple qu'il le semble. Il s'agit de concevoir des algorithmes permettant d'ajuster leurs sorties en fonction des entrées reçues. La conception de ces algorithmes est principalement basée sur des modèles statistiques combinés à une multitude de méthodes d'optimisation. Ces algorithmes ou "modèles" sont conçus suivant un enchaînement précis présenté sur la figure 3.24.

La figure 3.24 permet d'identifier un processus de conception en deux étapes : une étape d'entraînement et une étape de test. l'étape d'entraînement permet au modèle d'ajuster l'ensemble de ses paramètres pour adapter sa prédiction à la valeur réelle du résultat souhaité. Le processus d'ajustement est effectué différemment selon les modèles, les types d'apprentissage (supervisé, non supervisé, clustering,...) ainsi que la tâches auquelle le modèle est destinée (classification, régression,...). La deuxième étape étant l'évaluation du modèle après son entraînement par un jeu de données qui n'a pas été utilisé pendant la phase précédente, afin de conclure sur la précision du modèle.

Ces algorithmes nécessitent des quantités considérables de données. Ce qui requiert une puissance de calcul importante pour l'entraînement, d'où l'apparition et l'évolution des technologies de calcul dans le cloud, car ces plateformes, hébergés dans des data centers, offrent des services permettant aux entreprises ainsi qu'aux particuliers d'avoir à leurs disposition, des machine "virtuelles" leurs permettant d'avoir des puissance de calcul énorme, tout en se passant par les coûts d'achats et de maintenance de ces machines. Les services mentionnés précédemment sont proposés avec un mode de facturation selon l'usage (en heures ou jours de calcul), avec la possibilité de réduire et d'augmenter la puissance de calcul en fonction du besoin.

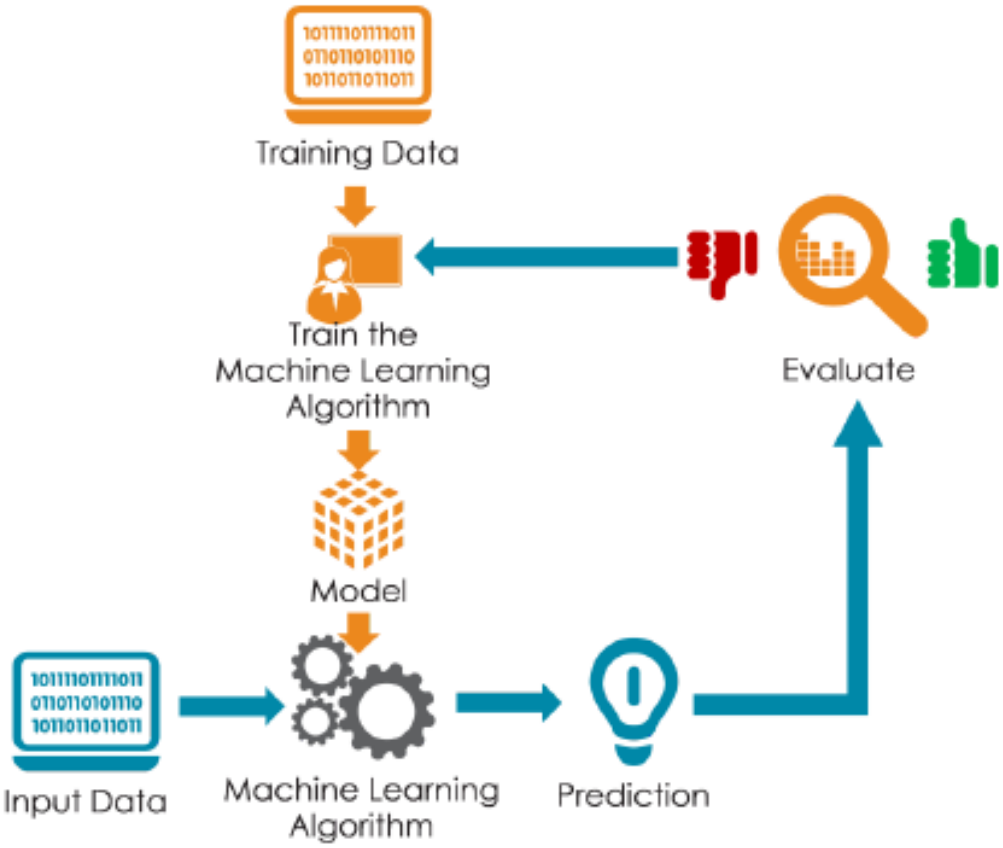

FIGURE 1.1 – Processus de développement d'un model IA

L'usage des services cloud représente un grand saut technologique. Néanmoins, pour des applications nécessitant le transfert de données massifs de leurs sources aux datacenters recevoir les résultats de calculs plusieurs contraintes font surface. Les problématiques majeures sont liées principalement à la sécurité des données, puisque les données transitent sur le réseau pour arriver à un datacenter, où ils seront stockés puis traités. Ce qui pose un risque majeur de sécurité dans le cas des intrusions ou hacks sur des datacenters, ce qui met en danger les données sensibles des utilisateurs et qui peut les mettre en danger dans d'autres. L'exemple le plus illustratif étant celui du fameux hack de l'une des voitures de Tesla en 2020.

D'autres problématiques liées à la disponibilité des services sont abordées lors des problèmes sur les datacenters hébergeant leurs services, d'autres liés à aux problèmes de latence dans des cas d'applications temps-réels.

1.2 Edge Computing

Pour répondre à ces problématiques, l'edge computing a vu le jour. C'est un paradigme de calcul distribué ustilisé dans le cadre du cloud computing consistant à traiter les données à la périphérie du réseau, près de la source des données. Cette technique est apparue pour résoudre les faiblesses du cloud comme le problème de latence, permet de diminuer les coûts et d'apporter de la puissance des objets connectés. Prenons l'exemple d'une caméra de surveillance qui enverrait en temps réel les images qu'elle collecte, ces images sont ensuite traitées par un réseau de neurones permettant d'effectuer une reconnaissance faciale. Le principal coût du dispositif est lié à la quantité de données à envoyer à travers le réseau et traiter via le cloud.

Le principe, comme montré sur la figure 1.2, consiste à traiter les données le plus proche de leur source le plus possible via l'edge computing.Les données ne pouvant pas être traitées via l'edge devront l'être par le fog computing. Les traitements ne pouvant être effectués ni par l'edge ni par le fog seront traités sur le cloud. Ce principe a pour but de limiter les données transitant sur le réseau pour améliorer la vitesse des traitements.

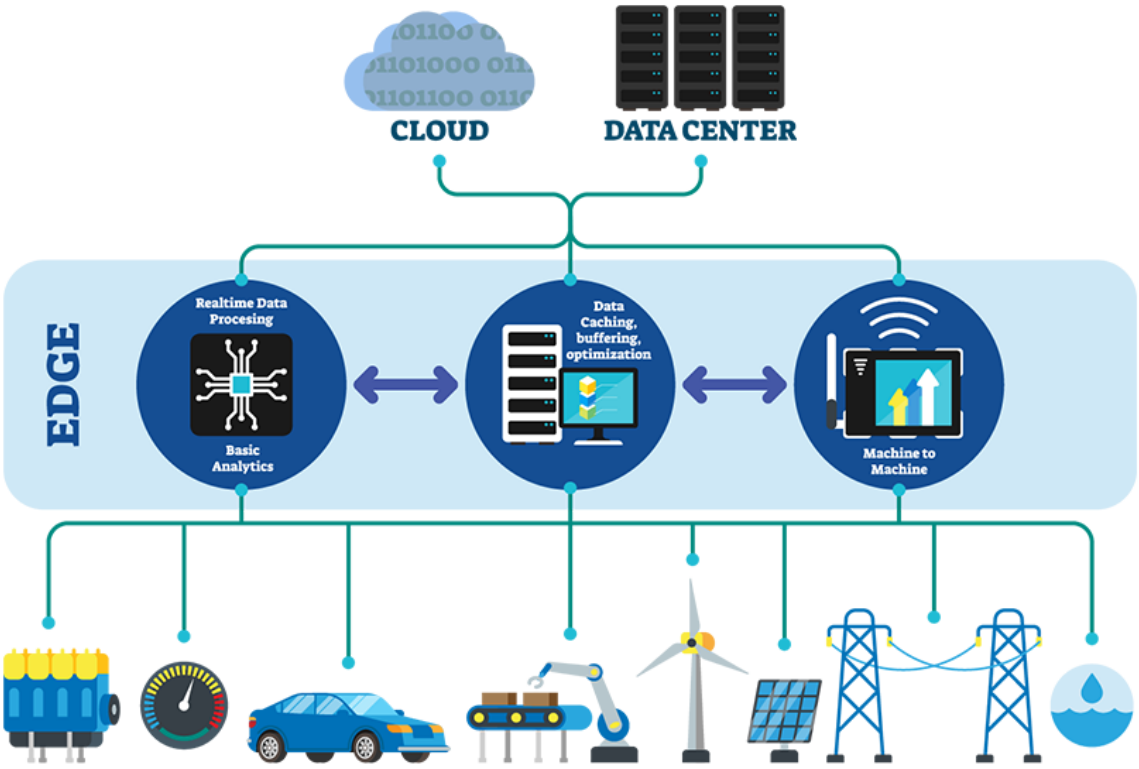

FIGURE 1.2 – Architecture réseau Edge/Cloud

1.3 Tiny Machine Learning

L'apparition de la discipline du TinyML vient pour répondre au besoin du Edge computing, principalement en proposant des applications pouvant être implémentées sur des environnements limités en ressources. Ainsi, plusieurs recherches ont été menées sur ce sujet, qui ont aboutit à des méthodes d'optimisation des modèles pour être exécutés sur des microcontrôleurs ou sur des accélérateurs d'inférence. Les géants de la technologie se sont vite tournés vers ce domaine à forte croissance. Ainsi nous avons vu le lancement de plusieurs gammes d'accélérateurs d'inférence ainsi que les frameworks de développement pour chaque plateforme.

La figure 1.3 montre deux accélérateurs de la marque Intel (Neural Compute Stick) fonctionnant sous OpenVino et celui de Google (Google Coral) utilisant Tensorflow lite ou Tensorflow Micro.

Le développement des modèles spécifiques à ces environnements hardware suit un cheminement bien particulier, car il doit prendre en compte les spécificités de chaque composant hardware, ce workflow est présenté dans la figure 1.4.

FIGURE 1.3 – Présentation des deux sticks Intel NCS2 et Google Coral

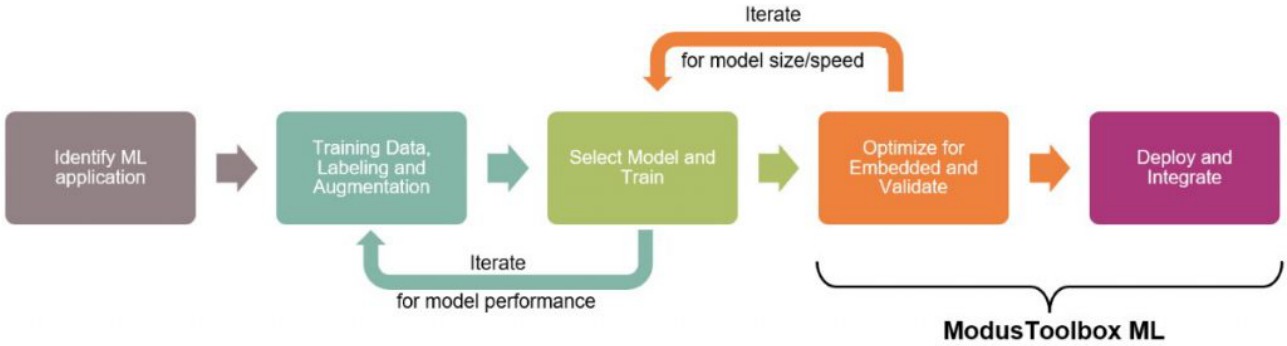

FIGURE 1.4 – Cycle de développement des algorithmes TinyML

On constate qu'au plus des étapes d'entraînement connus du modèle, s'ajoute une étape d'optimisation qui est spécifique à la plateforme ciblée.Généralement les frameworks de développement tels que Tensorflow ou PyTorch proposent des méthodes qui permettent de proposer une version plus optimisée. Les méthodes d'optimisation les plus courantes sont :

— **l'élagage des réseaux de neurones (Pruning)**, qui est un terme emprunté de l'agriculture signifiant découpage des branches non nécessaires au développement de l'arbre. Cette analogie est utilisée également dans le domaine de l'IA et consiste à désactiver ou d'enlever des neurones et les connexions qui contribuent le moins à la prédiction, cette opération est expliquée sur la figure 1.5.

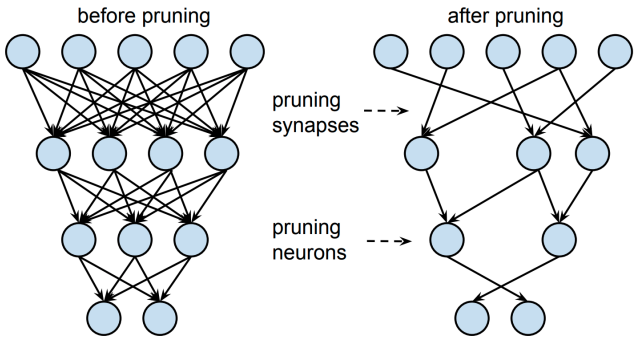

FIGURE 1.5 – Elagage des réseaux de neurones

Le choix des neurones ou des synapses à désactiver se fait à l'aide de méthodes statistiques permettant de définir la contribution du noeud au calcul. L'idée est relativement

simple, si un neurone est rarement activé avec une grande valeur, c'est qu'il est rarement utilisé par le réseau. Cette opération d'optimisation est un procédé itératif, comme le montre la figure 1.6.

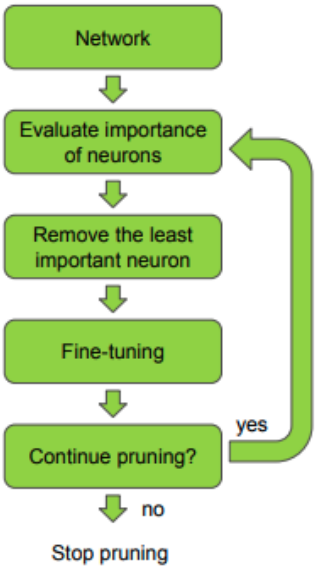

FIGURE 1.6 – Algorithme d'élagage des réseaux de neurones

— **Quantification des poids** ou Weight quantization, qui désigne la conversion des représentations en mémoire des poids et des activations des réseaux de neurones en des représentations plus légères en calcul et en mémoire, cette opération est montrée sur la figure 1.7. Le but est de faire la conversion en entiers INT8 ou INT16 des poids, sans avaoir une perte considérable de performance, ce qui n'est pas toujours le cas, ce qui oblige des fois à rester sur des représentations en virgule flottante. Par exemple : passer de Float64 à Float32 ou Float16.

(a) Weights after pruning

-1.01	1.00	0.00	0.88
0.00	0.17	0.00	-0.02
0.56	0.00	0.38	0.00
0.00	-0.49	-0.95	0.00

(b) Weights in quantization levels

-1.00	1.00	0.00	1.00
0.00	0.50	0.00	-0.50
0.50	0.00	0.50	0.00
0.00	-0.50	-1.00	0.00

(c) Weights stored in hardware

-2	2	0	2
0	1	0	-1
1	0	1	0
0	-1	-2	0

FIGURE 1.7 – Quantification des poids des réseaux de neurones

— **Knowledge distillation** désigne le process d'usage d'un deuxième modèle appelé modèle étudiant ou (Student model) pour prédire le modèle original entraîné. Ce process a été utilisé avec succès pour proposer des versions légères des modèles couramment utilisés tels que BERT. L'opération de distillation est montrée sur la figure 1.8.

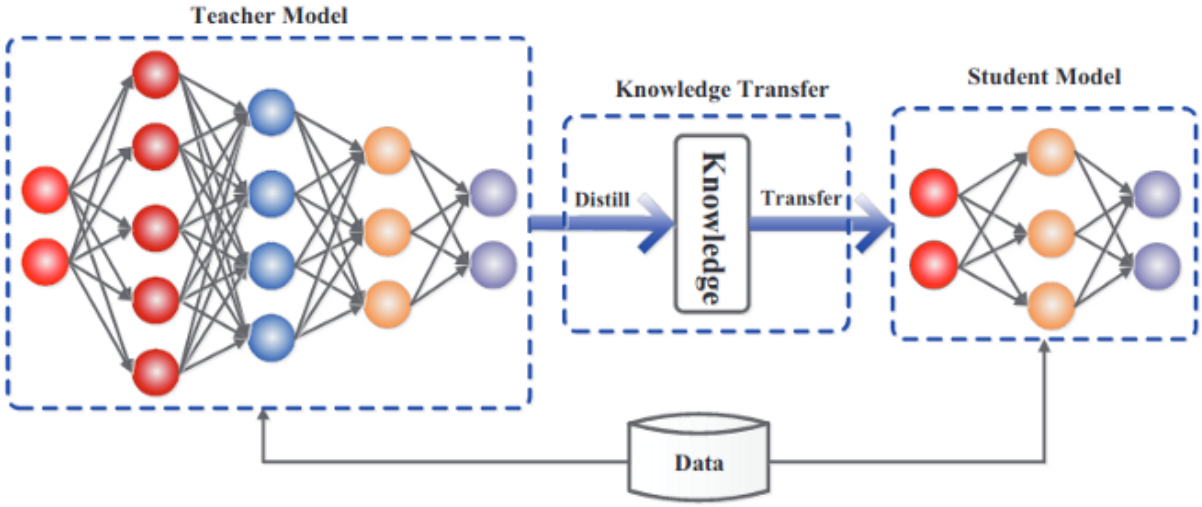

FIGURE 1.8 – Knowledge distillation : modèle étudiant et enseignant
Source : Knowledge Distillation : A Survey by Jianping Gou et al.

L'usage de ces techniques d'optimisation est une étape importante du présent projet, mais cet usage demeure marqué d'une difficulté liée à la contrainte temporelle du projet. Nous devons donc se contenter de comprendre le fonctionnement de ces trois techniques, et d'utiliser les implémentations qui sont déjà fournis par les frameworks d'apprentissage qui seront utilisés.

Chapitre 2

Présentation du projet

2.1 Problématique

La gestion du trafic de véhicules en ville est un problème de plus en plus pressant, certaines villes ont adopté des moyens de gestion intelligents tels que des feux dont la durée est calculée à partir des flux de véhicules. Cependant, la présence des policiers est des fois indispensable dans les cas d'accidents ou d'embouteillages afin de fluidifier la circulation. Or, cette présence peut elle-même causer des problèmes liés au manque de visibilité des policiers pour les conducteurs. Ainsi, une confusion peut s'installer auprès des conducteurs lorsqu'ils s'aperçoivent que le feu est vert, alors que tous les véhicules sont à l'arrêt, certains conducteurs se mettent à klaxonner, d'autres essaieront même des manoeuvres qui peuvent mettre en danger la vie des piétons et des conducteurs à côté.

Ce qui explique le besoin de synchroniser les feux de signalisation avec les mouvements du policier. Cette synchronisation se base sur une reconnaissance de gestuelle de ce dernier qui doit être effectuée en temps réel, et doit avoir une précision élevée pour ne pas être une autre cause d'accidents. L'usage d'une reconnaissance à base d'une caméra n'est pas adapté à ce contexte, malgré le fait qu'il donnera plus de précision. L'usage des accéléromètres est un choix optimal pour cette problématique, car il donne plus de liberté de mouvement au policier, et lui permet d'utiliser cette fonction dans des carrefours non équipés de dispositifs de surveillance.

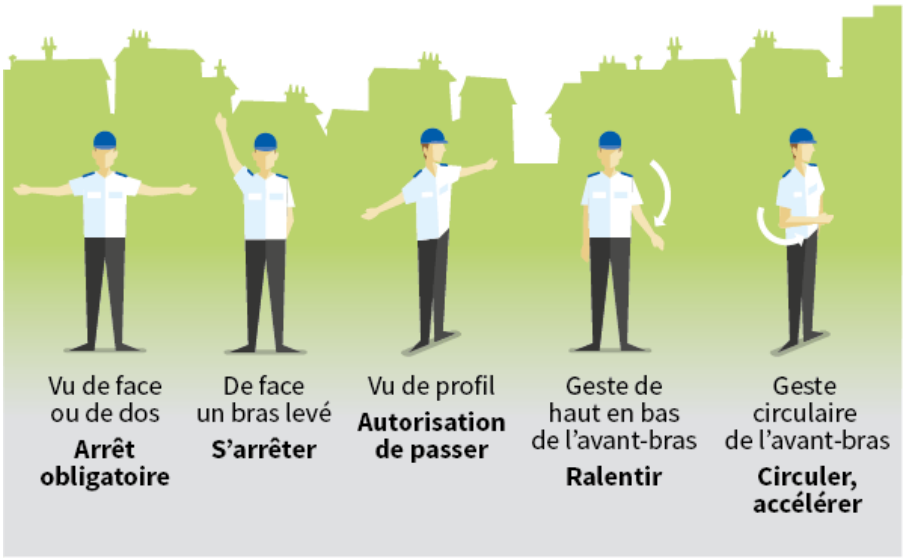

FIGURE 2.1 – Exemple des gestes ciblés par la reconnaissance

2.2 Objectifs du projet

L'objectif principal du projet est de présenter une preuve de concept d'une application TinyML ayant comme objectif la reconnaissance de la gestuelle d'un policier de route. Cette reconnaissance servira pour contrôler les feux de signalisation dans un carrefour miniaturisé. D'autres objectifs sont également envisagés, à savoir la rédaction d'une documentation exhaustive qui servira de guide regroupant les différentes étapes et bonnes pratiques à suivre par les étudiants qui seront amenés à travailler sur des sujets similaires.

Pour atteindre ces objectifs, il serait judicieux de fixer des jalons représentant les grandes étapes de développement. Ces étapes serviront également comme références pour quantifier l'avancement du projet et ainsi prendre des mesures en cas de dérive ou retard. Les jalons ainsi que les dates prévisonnelles de réalisations sont expliqués avec plus de détails dans l'Annexe A : gestion de projet.

2.3 Environnement hardware

Dans cette partie, on se propose de discuter les choix effectués pour les cartes électroniques utilisés dans le cadre de ce projet. Pour ce projet, nous avons choisit la carte Arduino Nano BLE 33 Sense (figure 2.2) comme carte cible dans un premier temps. Car cette carte est dotée d'un microcontrôlleur nRF52840 développé par Nordic Semiconductor, basé sur une architecture 32-bit ARM Cortex-M4 doté d'une unité de calcul à virgule flottante, et ayant une fréquence de 64 MHz. Ce microcontrôlleur est également doté d'une mémoire flash de 1MB et d'une RAM de 256KB. Le plus grand avantage avec les cartes de type arduino étant la simplicité d'usage, ce choix nous permettera de se concentrer dans un premier temps sur l'aspect IA du projet au niveau des performances des réseaux de neurones et les niveaux d'optimisation possibles, avant de porter le modèle sur la carte SensorTile commercialisée par ST Microelectronics.

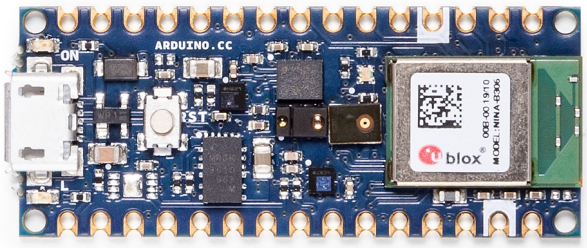

FIGURE 2.2 – Arduino Nano BLE 33 Sense

2.4 Logiciels & frameworks utilisés

Pour la partie intelligence artificielle, nous avons choisi d'utiliser le framework Tensorflow 2.7.0. Il s'agit de la version la plus récente de Tensorflow (figure 2.3, qui est un framework open-source dédié pour le domaine d'apprentissage automatique, développé par Google qui a lancé sa première version en 2015.

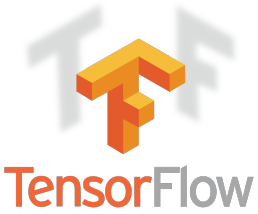

FIGURE 2.3 – Logo du framework Tensorflow

L'avantage de ce framework par rapport à ces plus grands concurrents (Pytorch, développé par Facebook) est son intégration de la version Tensorflow Lite, regroupant des méthodes d'optimisation pour l'inférence des modèles sur des appareils à faible puissance de calcul. Et malgré que Pytorch présente des avantages de stabilité et de vitesse par rapport à Tensorflow, et propose une version Pytorch Lightening (équivalent de TF lite) depuis 2019, Tensorflow lite reste en tête des statistiques d'utilisation vu sa stabilité par rapport à PyTorch Lightening.

2.5 PlateformIO

PlaterformIO (figure 2.4) est un IDE qui permet de programmer de nombreuses cartes à microcontrôleurs.On peut simplement installer le coeur du système platformIOCore, qui fournit des outils en ligne de commande pour simplifier les transferts et autres opérations vers les microcontôleurs. Nous avons choisi cet outil car il permet plus de liberté par rapport à l'IDE fournit par Arduino. Pour la deuxième carte, nous seront ammenés à travailler avec les IDE fournit par ST Microelectronics, car ils offrent une grande liberté au niveau de la configuration du hardware.

FIGURE 2.4 – Logo de l'IDE PlatformIO

2.6 Google Colab

Google Colab (figure 2.5) est un environnement de développement cloud développé par Google et dont le but est de fournir un accès gratuit (et payant pour une version plus puissante) à des machines puissantes destinés à entraîner des modèles IA directement dans le cloud en utilisant seulement un navigateur. Le choix de cet outil est justifié par le fait qu'un entraînement local sur nos machines va prendre énormément de temps. Il en résulte un perte monumentale de temps vu que l'optimisation des performance (hyperparameter tuning) demande plusieurs essais avec des paramètres différentes. Ce choix est également possible puisque le projet n'est pas confidentiel.

FIGURE 2.5 – Logo de l'environnement Colab

2.7 Google TinyMotion Trainer

Il s'agit d'une application faisant partie de la collection Google Experiments, qui est une collection d'applications liées à l'IA, et développées par les équipes de Google ou par des développeurs tiers en utilisant des outils fournits par Google. Cette plateforme permet de simplifier le processus de développement d'applications IA pour les microcontrôlleurs via une application web (figure 2.6). Cette application propose la simplification complète du processus de développement de bout en bout ; l'acquisition des données via une communication bleutooth avec la carte ciblée, l'entraînement du modèle et le packaging de l'application sous forme d'un firmware à flasher directement sur la carte ciblée. Pour notre projet, nous utiliserons cette plateforme uniquement pour la génération de la dataset, qui va être exploitée par la suite sur Google Colab pour entraîner le modèle de reconnaissance.

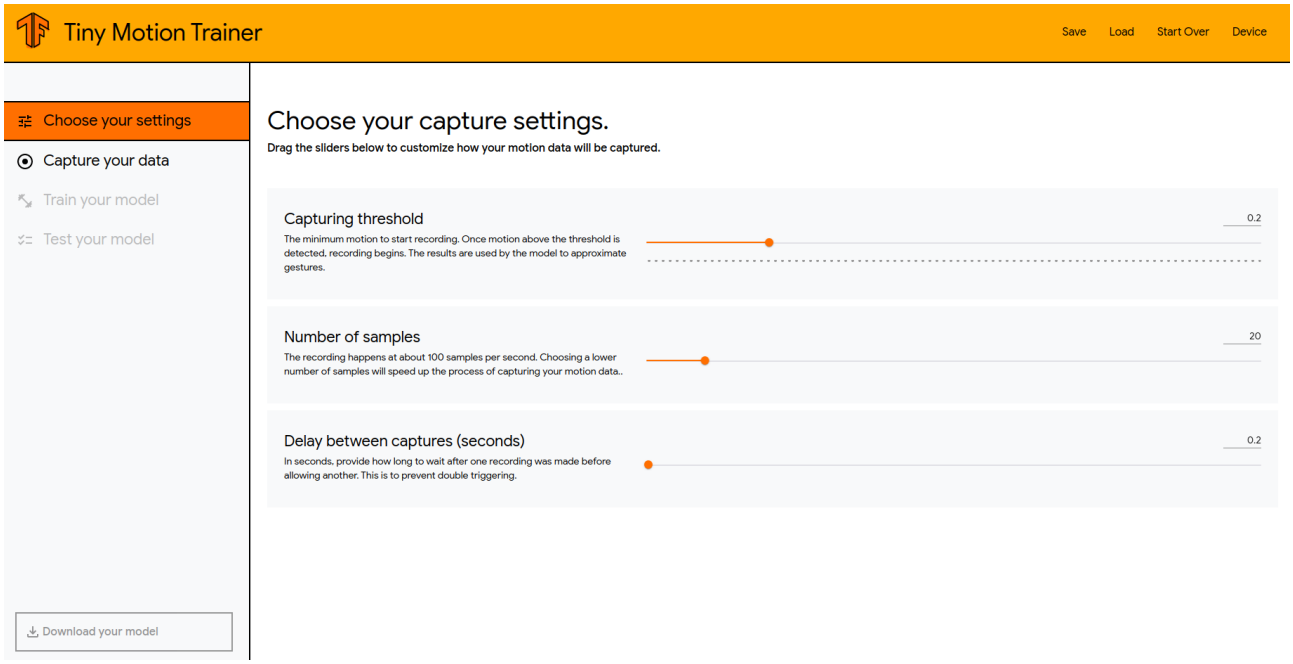

FIGURE 2.6 – Interface Google TinyMotion trainer

Chapitre 3

Etapes de développement

Une fois le choix des outils effectué, il convient de détailler les étapes de développement du projet. la figure 3.1 présente l'ensemble des étapes de développement. Dans les sections suivantes, nous détaillerons les grandes partie réalisés jusqu'à cette date.

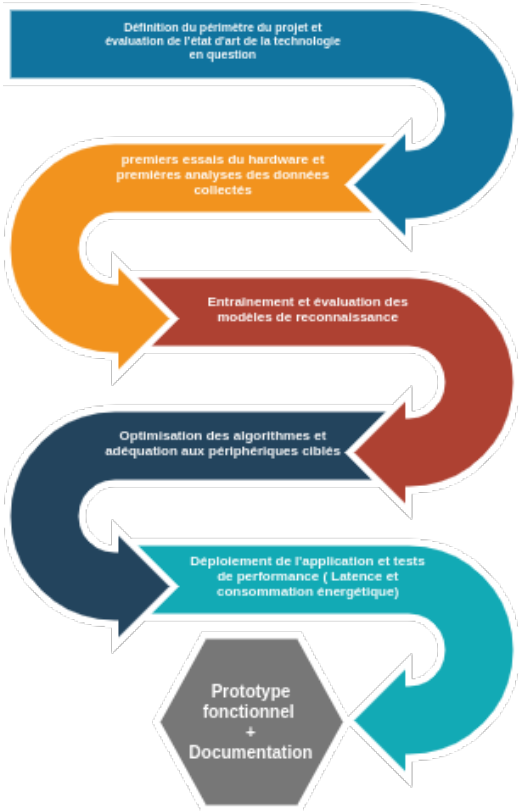

FIGURE 3.1 – Shématisation des étapes de réalisation du projet

3.1 Choix des architectures des réseaux de neurones

Le présent projet est un projet de reconaissance de signaux échantillonnés. Il est donc évident que l'architecture du réseau de neurones doit prendre en compte l'ordre des séquences, ce qui renvoie à l'utilisation de modèles récurrents ou convolutifs. Pour cette raison, nous avons décidé de comparer un réseau LSTM (Long Short Term Memory), un réseau convolutif à convolution unidimentionnelle (1D CNN) ainsi qu'un réseau de neurones profond standard, qui servira de baseline.

3.1.1 Réseau convolutif à une dimension

Il s'agit d'un type de réseaux de neurones à base d'opérations de convolution à une dimension (figure 3.2), ce type a eu un énorme succès dans le domaine de la vision par ordinateur. La différence entre cette architecture et celle utilisé computer vision (figure 3.3) est la représentation des filtres utilisés pour l'opération de convolution (matricielle pour les réseaux convolutifs classiques et en vecteur à une ligne pour les réseaux convolutif à une dimension).

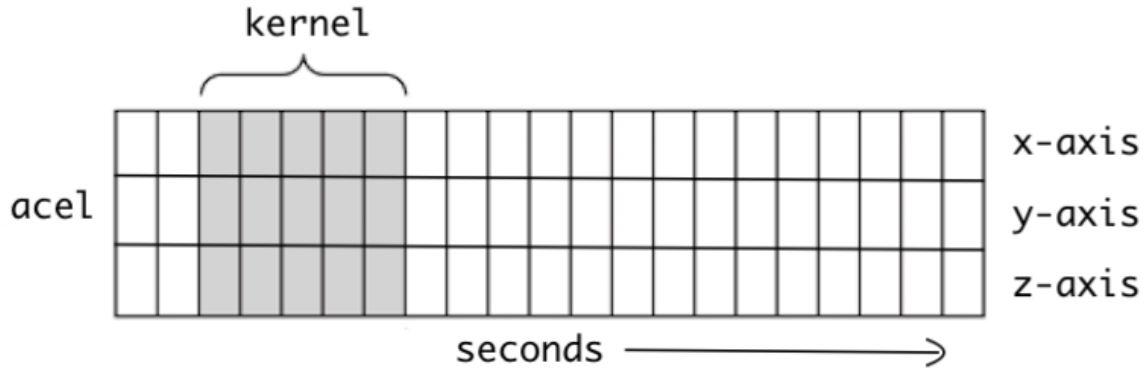

FIGURE 3.2 – représentation d'une opération de convolution 1D

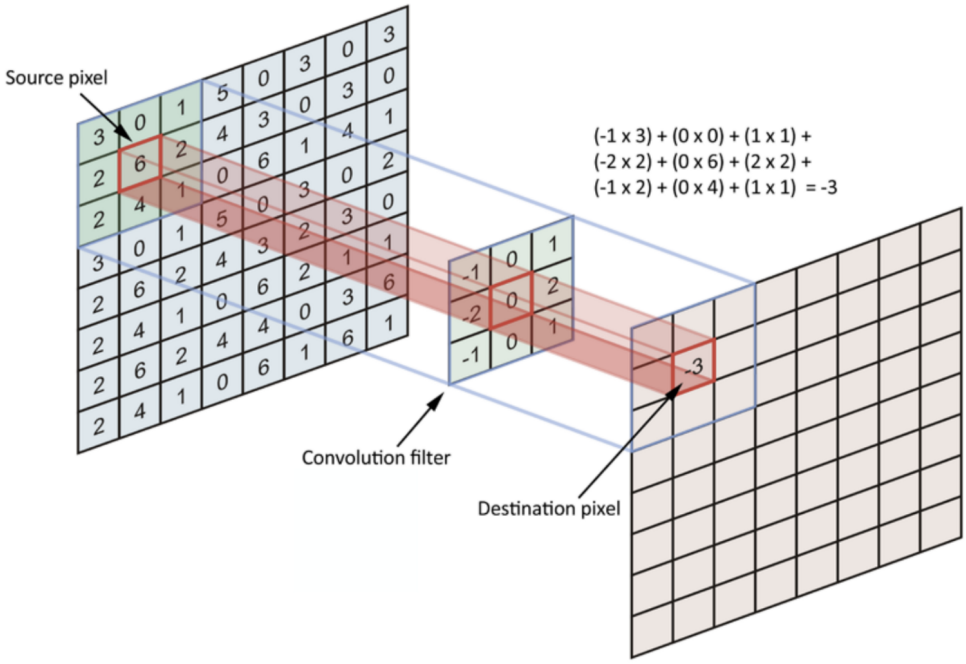

FIGURE 3.3 – représentation d'une opération de convolution 2D

Ce type de réseaux est composé de deux parties, la première étant le bloc de convolution, dans lequel on effectue plusieurs convolutions successives dans le but d'extraire des caractéristiques du signal, ces caractéristiques sont par la suites passés au deuxième bloc, qui est un bloc de réseaux de neurones denses, ce qui permet de calculer la prédiction. Ces opérations sont illustrés sur la figure 3.4.

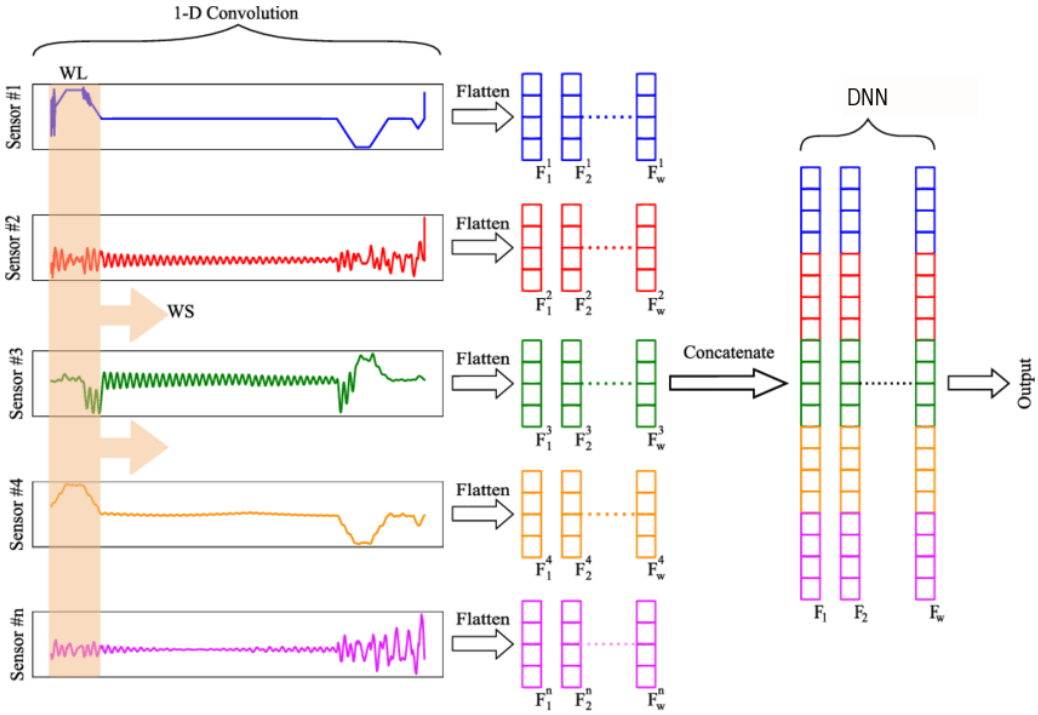

FIGURE 3.4 – Architecture détaillée d'un réseaux convolutif à une dimension

3.1.2 Réseau LSTM

Les réseaux LSTM ou Long Short term memory sont une catégorie de réseaux de neurones récurrents. Utilisés principalement pour des tâches de traitement de parole ou de traitement de texte, cette architecture est conçue pour tenir compte de l'aspect séquentiel des données qu'elle reçoit en entrée. Dans une phrase ou un signal, la position de l'échantillon ou le mot ainsi que son contexte (les données qui le précèdent) représentent des informations pertinantes qui sont perdues si on utilise un réseau de neurones standard, ce qui rend leurs performance très peu fiable dans les cas d'utilisation mentionnés.

Cette architecture s'organise sous forme de blocs, représentés sur la figure 3.5, chaque bloc est composée de trois portes (ou gates), une porte d'oubli (forget gate) qui a pour fonction de filtrer les informations reçus par la cellule précédente (ou à la date précédente t-1) pour récupérer les caractéristiques pertinantes à la tâche. La deuxième porte est la porte d'entrée (Input gate), chargée d'identifier les informations d'input à la date t. La combinaison de ces deux vecteurs d'informations est alors passée à travers une fonction d'activation qui va modifier la sortie de la cellule (New hidden state + cell state).

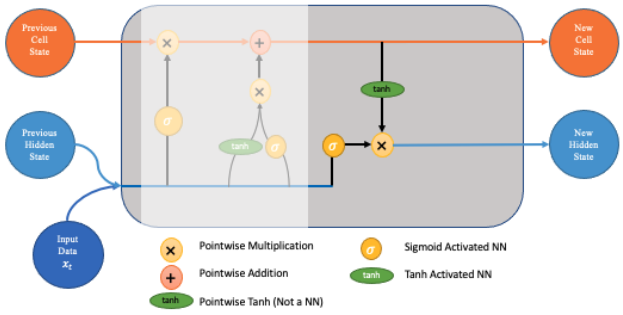

FIGURE 3.5 – Exemple d'une cellule LSTM

3.2 Collecte des données

La phase de collecte des données étant une étape déterminante de tout projet AI, car cette partie requiert une attention particulière à la qualité de données. Une dataset non homogène ou contenant un nombre important d'outils peut conduire à un modèle très peu performant. D'où le temps énorme consacré par les data scientists pour la collecte et le nettoyage des données récupérées (data collection data cleaning). Pour notre cas, vu que notre cible est la reconnaissance de trois mouvements, à savoir le signe d'accélération, d'arrêt et de ralentissement, on a jugé nécessaire de visualiser les signaux résultants de ce mouvement dans le but de les comprendre, et voir si une optimisation est possible à travers une réduction de dimensions du signal. Les signaux capturés sont présentés dans les figures 3.7, 3.8 et 3.6.

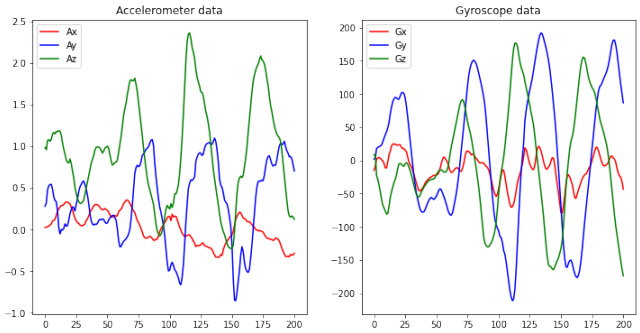

FIGURE 3.6 – Signal gyroscope et accéléromètre du signe d'accélération

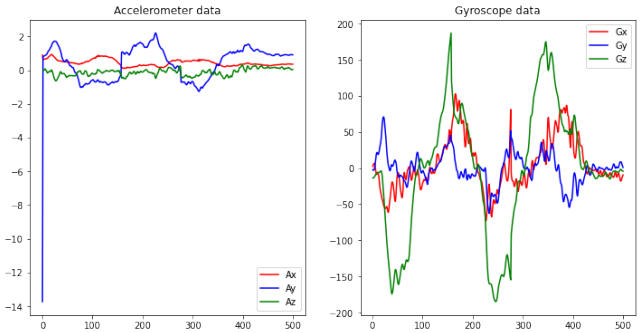

FIGURE 3.7 – Signal gyroscope et accéléromètre du signe d'arrêt

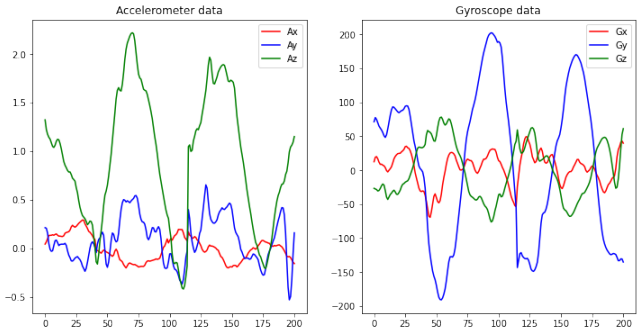

FIGURE 3.8 – Signal gyroscope et accéléromètre du signe de ralentissement

Une première idée étant de diminuer la dimension des signaux en calculant le module des trois signaux du gyroscope ainsi que ceux de l'accéléromètre. Cette méthode a été écartée par la suite car elle entraîne la perte d'information sur les axes concernés par le mouvement.

La visualisation de ces signes nous permet d'identifier des patterns à l'oeil nue, ce qui laisse supposer que les réseaux de neurones n'auront relativement pas de difficultés à reconnaître ces signaux. Dans la section suivante, on va évaluer cette hypothèse.

Une fois l'étape de visualisation effectuée, nous avons utilisé la plateforme TinyMotion Trainer pour collecter les signaux pour l'apprentissage. La figure 3.9 montre l'interface de collecte de données sur la plateforme.

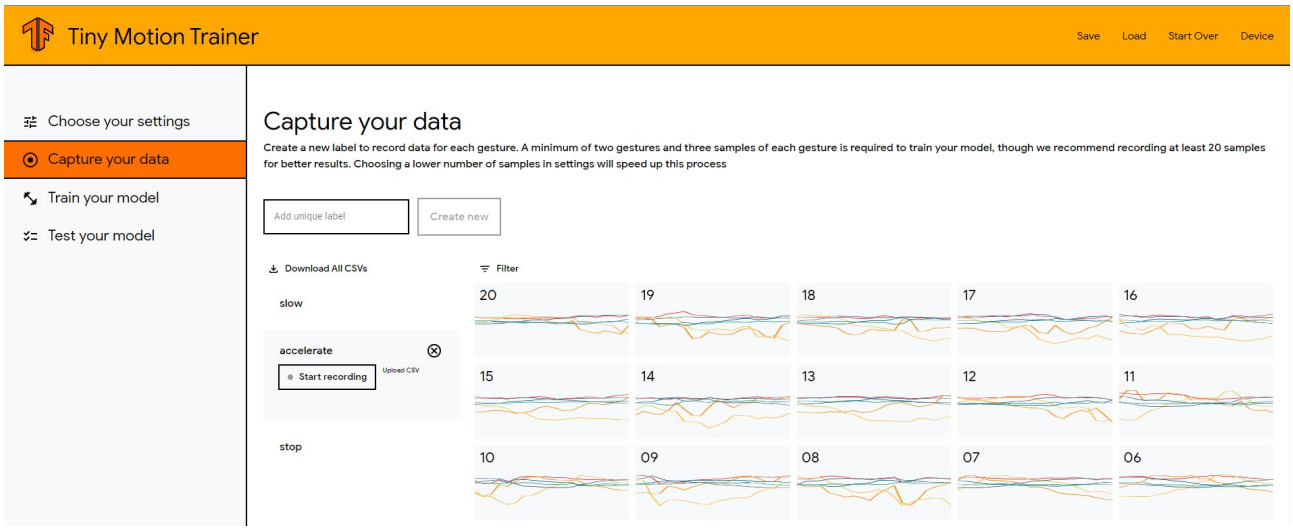

FIGURE 3.9 – Interface tinyMotion Trainer

En définissant les paramètres du signal désiré (nombre d'échantillons par signal, délai entre l'enregistrement de deux signaux, ...) la plateforme génère un fichier .Csv contenant les différents signaux d'un seul mouvement. Pour notre cas, nous avons décidé d'opter pour un signal de 50 échantillons, qui correspond à une durée de 0.42 secondes, vu que la fréquence d'échantillonnage du capteur est de 119Hz.

La dataset résultante est composée de 1500 signal par mouvement. Une dataset beaucoup plus importante que celle utilisée pendant la première partie du projet. Les figures suivantes représentent les signaux d'accéléromètre d'une personne parmi celles qui ont été sollicités pour la collecte de la nouvelle dataset. La visualisation des trois signaux représentant les trois gestes nous conduit à la même hypothèse formulée précédamment.

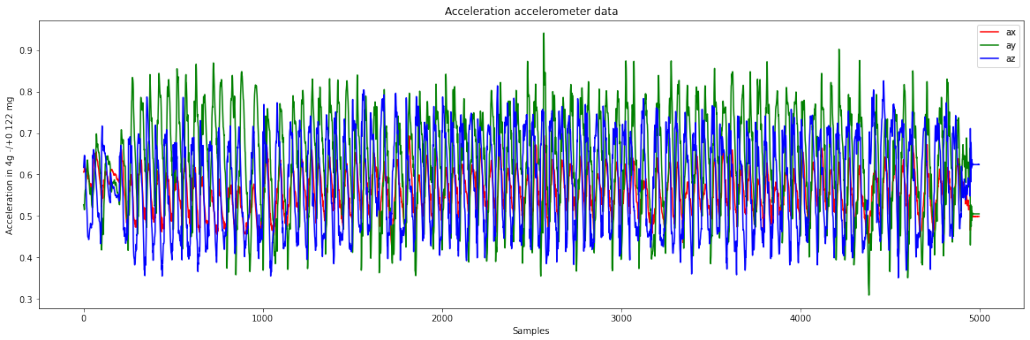

FIGURE 3.10 – Visualisation de données des signaux d'accélération

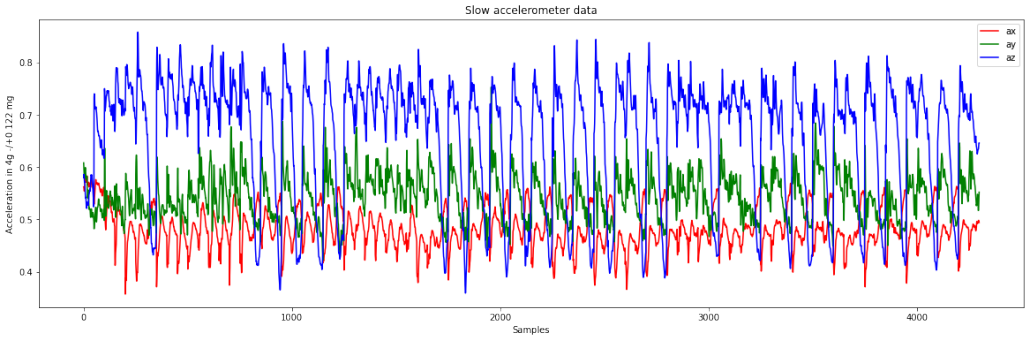

FIGURE 3.11 – Visualisation de données des signaux de ralentissement

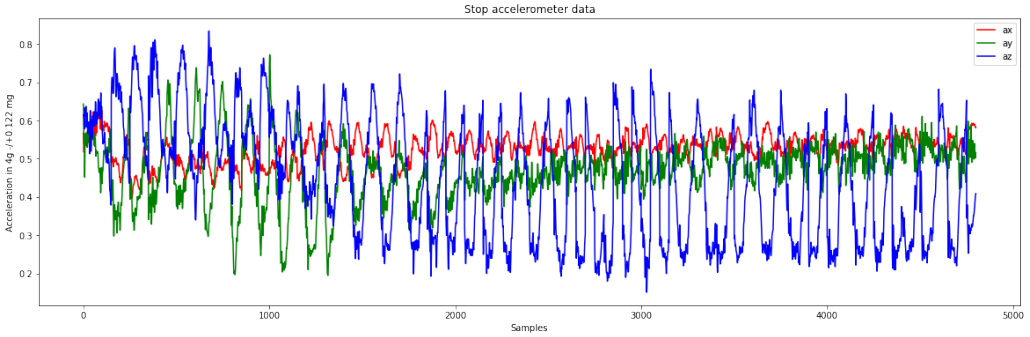

FIGURE 3.12 – Visualisation de données des signaux d'arrêt

3.3 Entraînnement des modèles et analyse des résultats

Une fois la collecte des données effectuée, l'étape qui a suivi était celle de l'intanciation des modèles et les premiers entaînements. Nous avons pu entraîner et optimiser deux sur trois des modèles précédamment mentionnés, à savoir le 1D CNN et le réseau Dense standard. Pour le réseau LSTM, il est toujours en phase d'optimisation de performances. les figures 3.13 et 3.14 montrent l'architecture DNN et CNN utilisés.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 flatten_38 (Flatten)         (1, 300)                  0

 dense_120 (Dense)            (1, 50)                   15050

 dense_121 (Dense)            (1, 15)                   765

 dense_122 (Dense)            (1, 3)                    48

=================================================================
Total params: 15,863
Trainable params: 15,863
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 3.13 – Architecture du réseaux de neurones dense utilisée

Layer (type)	Output Shape	Param #
conv1d_4 (Conv1D)	(None, 48, 64)	1216
conv1d_5 (Conv1D)	(None, 46, 64)	12352
max_pooling1d_2 (MaxPooling 1D)	(None, 23, 64)	0
flatten_2 (Flatten)	(None, 1472)	0
dense_4 (Dense)	(None, 100)	147300
dense_5 (Dense)	(None, 3)	303

Total params: 161,171
Trainable params: 161,171
Non-trainable params: 0

FIGURE 3.14 – Architecture du réseaux de neurones convolutif

Le résultat obtenus après les phases d'entraînement et d'optimisation ont permis de vérifier l'hypothèse mentionnée dans la section précédente, car même avec un réseau de neurones dense standard, on a réussi à obtenir une précison de 92% (figure 3.15), contre une précision de 98% pour le réseau convolutif (figure 3.16).

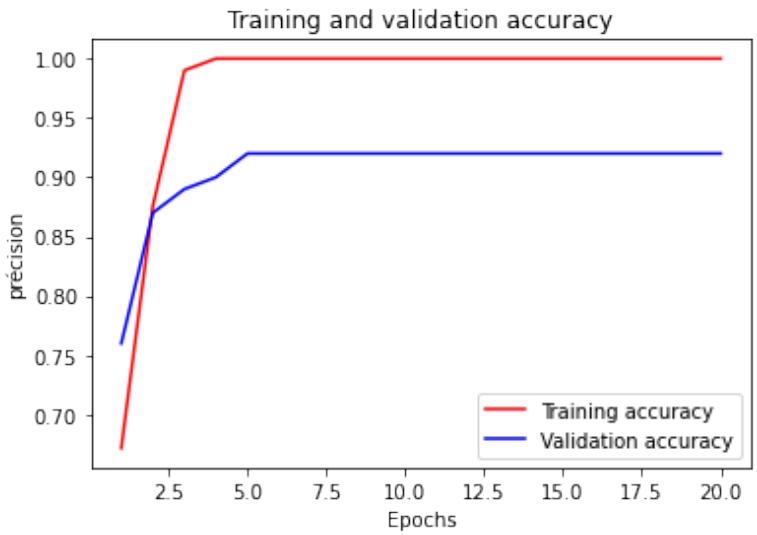

FIGURE 3.15 – courbe de précision du réseau dense utilisé

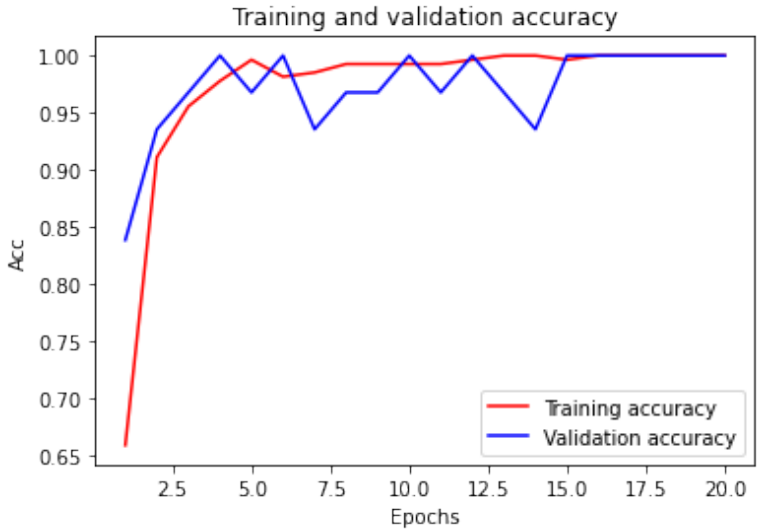

FIGURE 3.16 – courbe de précision du réseau dense utilisé

Les résultats étant meilleurs que ceux attendus pour un premier essai, nous avons effectué une vérification par validation croisée (Cross-validation), ainsi qu'une évaluation des performances des modèles sur des signaux nouveaux à l'aide d'une matrice de confusion. les résultats de la validation croisée sont présenté sur les figures 3.17 et 3.18.

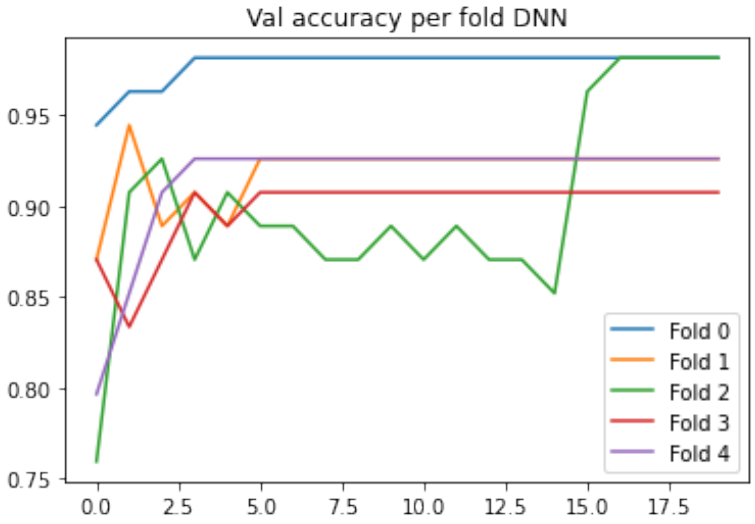

FIGURE 3.17 – courbes des résultats de la cross-validation sur le réseau dense

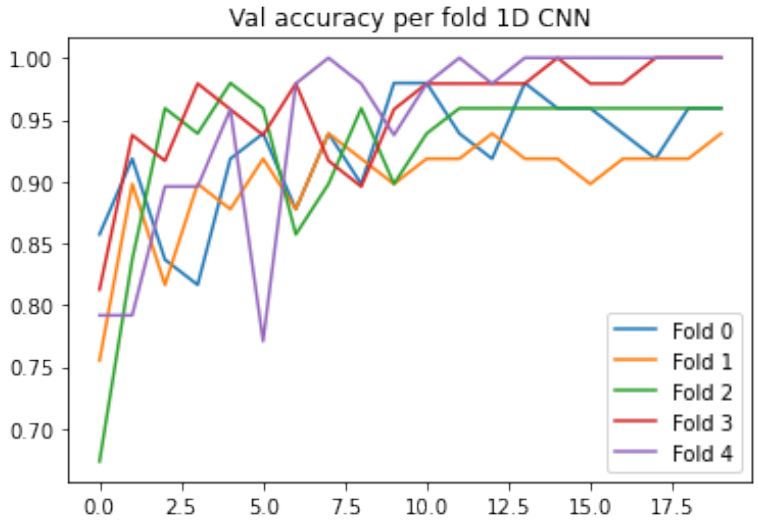

FIGURE 3.18 – courbes des résultats de la cross-validation sur le réseau convolutif

L'analyse des résultats de la validation croisée nous conduit à confirmer les performances du modèles obtenus précédemment, car on arrive à des précisions similaires en ayant des ensembles d'entraînement et de validation différents à chaque étape ou fold. Ce qui prouve que les deux modèles ont retenus l'essentiel des signaux d'entraînement et sont capable de généraliser sur de nouveaux signaux.

La vérification par matrice de confusion permet de visualiser le résultat de classification sur une nouvelle dataset, utilisée uniquement pour le test. Les deux matrices dbeie confusion sont présentés sur la figure 3.19 et 3.20.

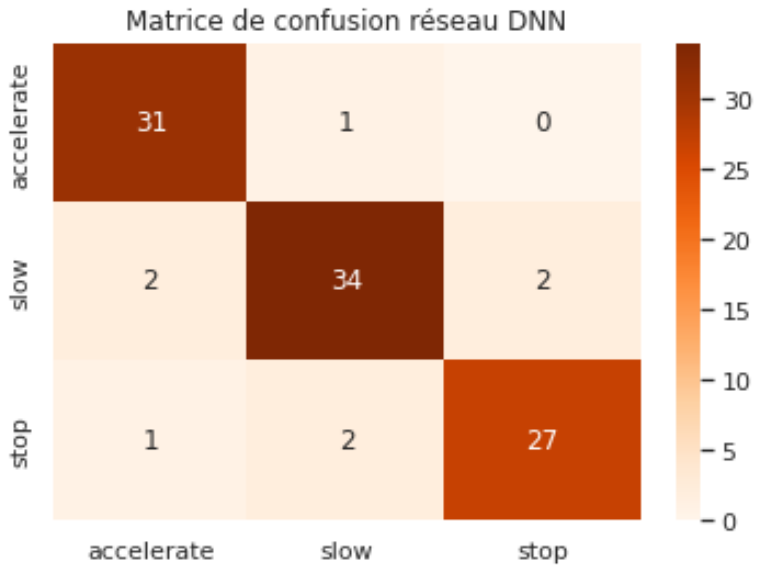

FIGURE 3.19 – Matrice de confusion du réseau dense

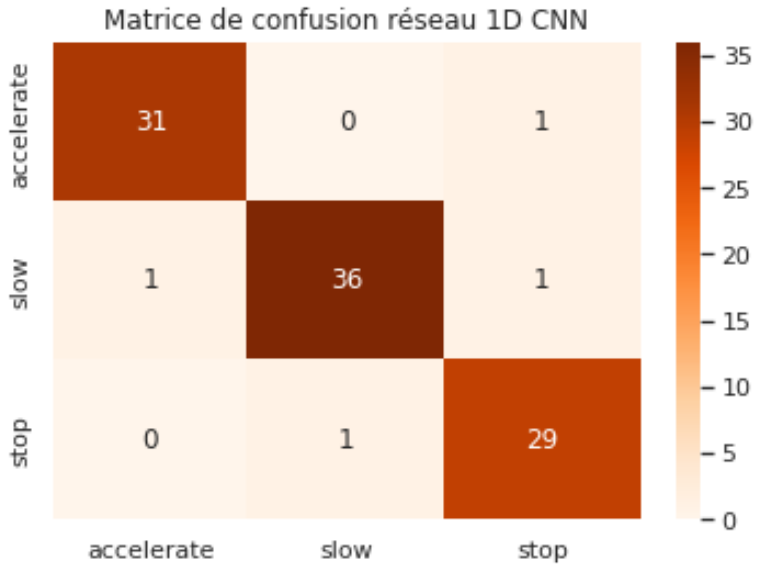

FIGURE 3.20 – matrice de confusion du réseau convolutif

Pour qu'un modèle soit performant, il doit être capable de reconnaître chaque mouvement correctement, ce qui implique que toutes les valeurs de sa matrice de confusion doivent être nulles en dehors de celles de la diagonale. Ce qui est le cas pour nos deux modèles avec un léger avantage pour le réseau convolutif.

Le problème avec ces trois modèles c'est qu'il donnent des résultats aux alentours de 30% lorsqu'ils sont testés avec des données récupérés directement depuis la carte arduino (pour simuler un traitement sur la carte cible). Ce qui remet en question la méthode de préparation de données utilisée initialement.

Initialement, l'ensemble de nos données était composé de signaux collectés par cinq personnes, ces données ont été divisés en trois parties, à savoir un ensemble d'entraînement, un ensemble de validation et un ensemble de test selon un ratio de 75% pour l'entraînement, 15% pour la validation et 10% pour le test. Cette approche a donné de faux bons résultats ; car on n'a pas pris en compte le fait que les mouvements sont effectués de différentes manières selon chaque personne. Ainsi, si notre ensemble d'entraînement et de test contiennent des signaux des mêmes personnes, le résultat est faussé puisque le modèle a tout simplement "appris" comment cette personne a fait ce mouvement, et ne sera pas capable de généraliser sur les mouvements d'autres personnes.

La solution était donc d'ajouter des signaux provenant de d'autres personnes, entraîner le modèle sur des signaux d'une sous partie de l'ensemble des participants, valider sur une sous-partie différente et tester sur une autre.
Il s'est avéré que nos modèles n'arrivent pas à dépasser les 33%, qui correspondent à la précision qu'aura un modèle qui fait un choix aléatoire à chaque fois. Pour optimiser la précision, nous avons utilisé une technique de recherche des hyper-paramètres optimales nommée "grid search", qui consiste à parcourir un ensemble de paramètres avec des valeurs définies en faisant un entraînement pour chaque combinaison, et retourne la combinaison donnant le meilleur résultat. En utilisant cette méthode nous aboutissons à une précision de 80% en agissant sur l'algorithme d'optimisation d'erreur et le taux d'apprentissage. L'inconvénient de cette méthode c'est qu'elle demande beaucoup de temps puisqu'elle effectue un entraînement par combinaison. La figure 3.21 montre l'évolution des précision avant et après l'usage du grid search.

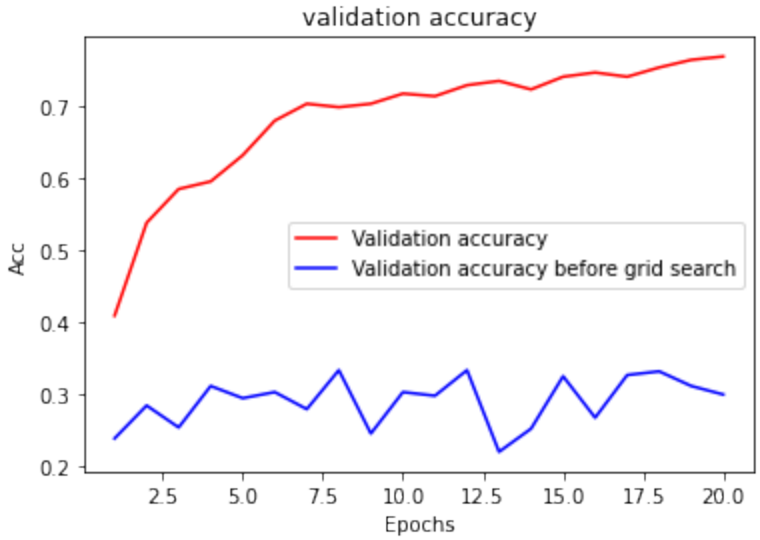

FIGURE 3.21 – Précision du modèle avant et après le grid search

3.4 Optimisation du modèle

Pour que le modèle soit adapté à la carte cible, il doit être optimisé, ce qui signifie la réduction de son empreinte mémoire ainsi que sa taille. Pour ce faire, on utilise la bibliothèque Model Optimization qui fait partie de l'environnement Tensorflow. Dans notre cas, nous allons utiliser la technique de Pruning, expliquée précédemment dans la partie 1.3 ainsi que la méthode de quantification des poids qui est faite avec Tflite.
Cette bibliothèque utilise l'algorithme de décroissance polynomiale (Polynomial decay) pour réduire la taille du réseau de neurones en augmentant le nombre des zéros sur l'ensemble des paramètres à chaque phase d'entraînement. La mise à zéro d'un paramètre revient à supprimer le nœud ou la connexion qui y est associée.
En choisissant un taux de sparsity adéquat, on peut avoir un équilibre entre la minimisation de la taille du modèle et la non dégradation de la précision du modèle. la figure 3.23 montre le gain en taille du modèle tout en gardant la même précision, on constate un gain de plus de 60% en termes de taille entre le modèle initial et le modèle optimisé.

```
print("Size of gzipped baseline Keras model: %.2f bytes" % (get_gzipped_model_size(keras_file)))
print("Size of gzipped pruned Keras model: %.2f bytes" % (get_gzipped_model_size(pruned_keras_file)))
print("Size of gzipped pruned TFlite model: %.2f bytes" % (get_gzipped_model_size(pruned_tflite_file)))
```

```
Size of gzipped baseline Keras model: 61148.00 bytes
Size of gzipped pruned Keras model: 21098.00 bytes
Size of gzipped pruned TFlite model: 19437.00 bytes
```

FIGURE 3.22 – Résultat de l'opération d'optimisation

Après la quantification, on arrive à une taille du modèle de 6MB, soit 10% de la taille initiale du modèle. Une fois les deux étapes d'optimisation terminées, il est nécessaire d'exporter le modèle sous forme d'un header (fichier .h) qui va être exploitable sur l'arduino. La figure **??** montre la taille du fichier .h généré.

```
INFO:tensorflow:Assets written to: /tmp/tmp8f26o3p6/assets
INFO:tensorflow:Assets written to: /tmp/tmp8f26o3p6/assets
WARNING:absl:Buffer deduplication procedure will be skipped when flatbuffer library is not properly loaded
Saved quantized and pruned TFLite model to: /tmp/tmpr2oyhhlr.tflite
Size of gzipped baseline Keras model: 61148.00 bytes
Size of gzipped pruned and quantized TFlite model: 6632.00 bytes
```

FIGURE 3.23 – Taille du fichier .h généré

3.5 Déploiement sur la carte Aduino BLE 33

Comme expliqué sur la section problèmatique, le modèle est déployé sur une carte arduino Nano qui servira de contrôle d'un carrefour miniature composé de quatre feux de signalisation. Une modélisation simplifiée de l'application est présentée sur le schéma 3.24 suivant :

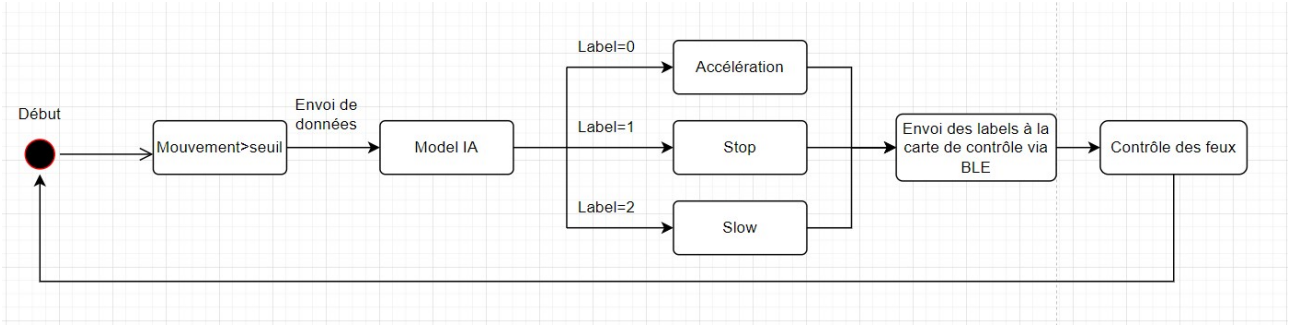

FIGURE 3.24 – Schématisation du fonctionnement de l'application

L'application récupère en continue les valeurs d'accélération, et dès que cette dernière dépasse un certain seuil, fixé à 1.5*G, avec G = 9.8 m/s², la carte récupère un signal de 50 échantillons. C'est ce signal qui va être utilisé pour prédire le mouvement après l'avoir standardisé (réduire la plage des valeurs à [-1,1]). La valeur de sortie du modèle servira pour déterminer le mouvement correspondant, qui va être envoyé par la suite par bluetooth à la carte qui contrôle le modèle miniature du carrefour. Les figures 3.25 et 3.26 montrent respectivement la sortie du modèle sur la carte principale, et le résultat sur le modèle miniature.

```
Accelerometer sample rate = 119.00 Hz
Gyroscope sample rate = 119.00 Hz

Accelerate: 0.311034
Slow: 0.656198
Stop: 0.032768

Accelerate: 0.590038
Slow: 0.405686
Stop: 0.004276

Accelerate: 0.135629
Slow: 0.858874
Stop: 0.005496
```

FIGURE 3.25 – Visualisation du résultat d'exécution du modèle

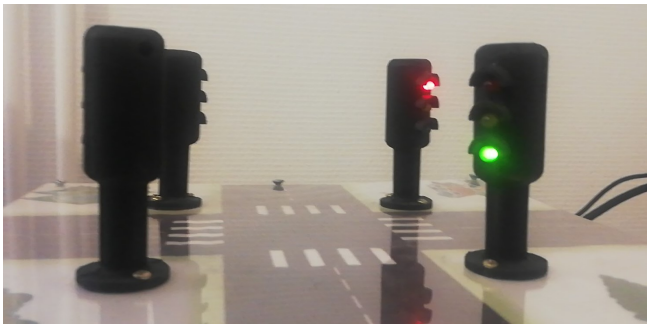

FIGURE 3.26 – Visualisation de la sortie sur le modèle du carrefour

Le constat effectué sur les résultats du déploiement étant que la précision théorique atteinte pendant l'entraînement n'a pas été atteinte, notamment car le modèle prédit dans la plus part des cas que le mouvement effectué est un mouvement d'accélération. Ce qui peut être expliqué par une perte de précision lors du processus de quantification des poids, ou à une différence de la distribution des données.

Conclusion

Ce projet présent était et est toujours une opportunité intéressante de découvrir TinyML. Ce qui est un grand avantage compte tenu de l'utilisation plus importante de ces technologies dans l'industrie ainsi que dans la recherche. Travailler sur de telles applications nous permet d'acquérir les connaissances théoriques et pratiques qui nous permettraient de travailler en tant qu'ingénieurs dans des domaines qui nécessitent cette technologie.

Le processus de développement de ce projet a été marqué par de multiples difficultés qui nous ont ralentis et nous ont coûté beaucoup de temps, comme l'arrivée tardive des capteurs et des cartes de développement. En un mot, le projet est n'a pas pu être réalisé dans sa globalité, notamment suite à des retards de livraisons du matériel, ainsi que des erreurs lors du processus d'entraînement au niveau de la collecte et la préparation des données.

Bibliographie

[1] towardsdatascience.com/tiny-machine-learning-the-next-ai-revolution-495c26463868

[2] towardsdatascience.com/pruning-neural-networks-1bb3ab5791f9

[3] www.ornikar.com/code/cours/circulation/intersections

[4] en.wikipedia.org/wiki/Long$_s hort - term_m emory$

[5] fr.wikipedia.org/wiki/R%C3%A9seau$_n euronal_c onvolutif$

[6] Choudhury, Tanzeem Al. (2008). The Mobile Sensing Platform : An Embedded Activity Recognition System. Pervasive Computing, IEEE. 7. 32-41. 10.1109/MPRV.2008.39.

[7] Kai KunzePaul LukowiczHolger JunkerGerhard Tröster, Where am I : Recognizing On-body Positions of Wearable Sensors

[8] https ://www.tensorflow.org/api$_d ocs/python/tf/lite/TFLiteConverter$

[9] https ://www.tensorflow.org/model$_o ptimization/api_d ocs/python/tfmot/sparsity?hl = fr$

Annexe A : Gestion de projet

Pour ce qui est de la gestion de projet, nous avons décidé d'utiliser une méthode Scrum car elle était la plus adaptée aux fréquences des réunions, et échanges requises dans le cadre du projet.

L'adoption de cette méthodologie requiert un ensemble d'outils pour être mise en œuvre, ces outils peuvent être regroupés en deux catégories. La première catégorie étant celles de communication dont le but est d'assurer une communication continue entre les membres d'équipe ainsi que notre encadrant. Le deuxième type étant des outils de gestion de projet, utilisés pour faciliter les tâches de planification et de gestion de projet.

Pour les outils utilisés, nous avons opté pour :

— **Discord** : Plateforme de collaboration d'équipe en ligne offrant une multitude de fonctionnalités particulièrement adaptées aux besoins de l'équipe et à la situation actuelle. Cette plateforme permet d'effectuer des réunions en ligne, d'effectuer des partages d'écrans et de fichiers, de centraliser les documents et le travail effectué et de les éditer collaborativement en temps réel par le biais de l'éditeur de l'application. De ce fait, cet outil répond à la grande majorité des besoins immédiats de l'équipe pour avoir un confort de travail et une collaboration correcte.

— **Trello** : Il s'agit d'un outil de gestion de projet inspiré de la méthode Kanban. Il propose une méthode de planification des tâches à l'aide de tableaux de tâches. Il permet également d'affecter et de gérer l'attribution des tâches aux membres du groupe de projet.

— **GitHub** : Il s'agit d'un outil de gestion de versions distribué, utilisé dans le cadre du projet pour gérer l'historique des versions de développement et de les partager d'une manière instantanée.

— **Diagramme de Gantt** : Le but de ce diagramme est de représenter et estimer la réalisation des différentes tâches par les membres de l'équipe. Ce diagramme a donc pour but de faire une prévision temporelle de l'évolution détaillée du projet, offrant de nouvelles perspectives de collaboration et organisation.

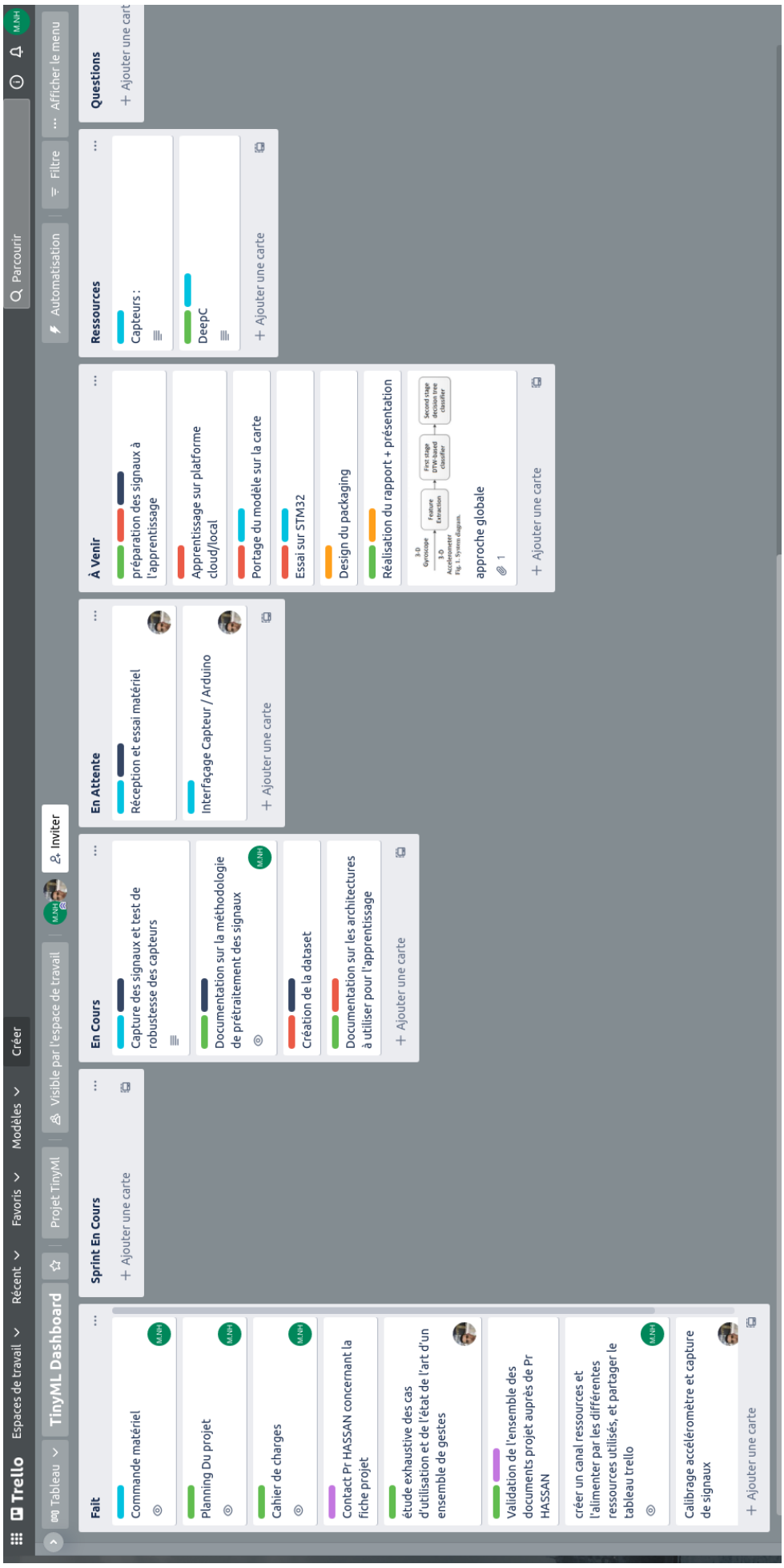

FIGURE 3.27 – Tableau Trello utilisé pour la gestion du projet

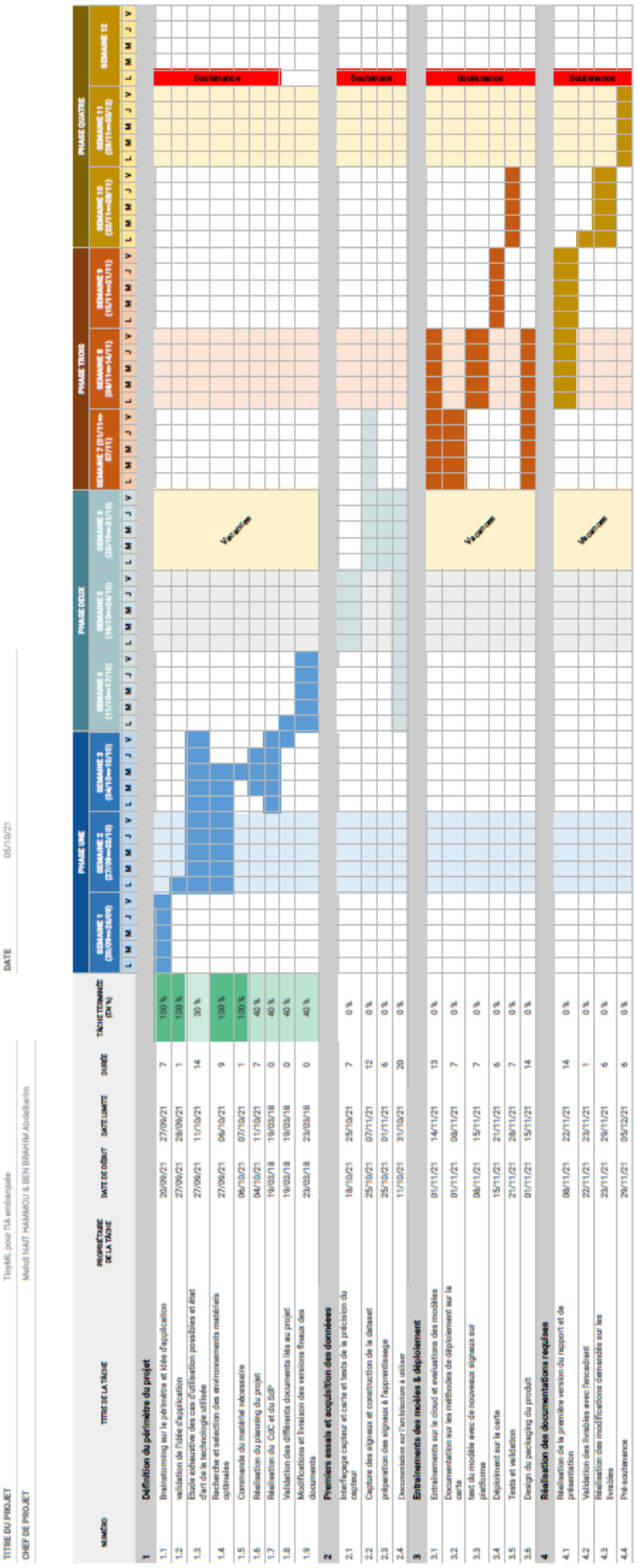

FIGURE 3.28 – Diagramme de Gantt du projet