Projet avancé en systèmes embarqués - Intelligence Artificielle : TinyML sur carte Raspberry

Yaël ALARY - Antoine CURALLUCCI - Benjamin CRESCENCE

Janvier 2022

Sommaire

1 Introduction

L'Intelligence Artificielle (ou IA) est une technologie de plus en plus croissante depuis plusieurs années. Utilisée dans énormément de domaines, en particulier récemment dans le cadre du développement de véhicules autonomes, elle permet d'apporter des solutions à certains problèmes que l'on ne sait pas comment résoudre. Ces problèmes sont souvent des problèmes de reconnaissance d'images, en temps réel ou non. En utilisant de nombreuses ressources matérielles, il est possible d'obtenir des résultats très précis et fiables.

Cette caractéristique est également le principal défaut de l'IA : pour obtenir des résultats les plus satisfaisants possibles, il est important d'être en possession de ressources conséquentes, que ce soit en terme d'acquisition de données ou de leur traitement. Dans le domaine des systèmes embarqués, cela se caractérise principalement par la façon dont on veut embarquer un modèle développé avec de l'IA sur un système, souvent très limité en terme de mémoire notamment pour les plus petits d'entre eux.

L'objectif de ce projet est de rendre accessible l'utilisation de l'IA sur un système embarqué, depuis son développement jusqu'à son utilisation, en passant par son implémentation sur le système. Nous allons nous intéresser aux différentes façons possibles d'aboutir à un modèle créé avec de l'Intelligence Artificielle, en particulier pour l'embarqué, et en particulier comment réduire l'utilisation des ressources tout en assurant un résultat satisfaisant.

2 Présentation du projet

Notre projet a pour but d'exploiter une caméra embarquée sur une carte RaspberryPi. Dans un contexte de création de modèle d'IA, il était possible d'exploiter plusieurs aspects, notamment la reconnaissance d'image et la classification d'objets. Nous avons donc choisi de travailler sur trois objectifs différents, ou plutôt trois façons différentes d'aborder des concepts similaires.

Pour ce faire, nous avons utilisé du matériel décrit ci-dessous :

2.1 Présentation du matériel hardware

— une carte Raspberry Pi 3B ;
— une caméra Raspberry Pi v2.1 8MP 1080p ;
— une carte Arduino Nano 33 BLE.

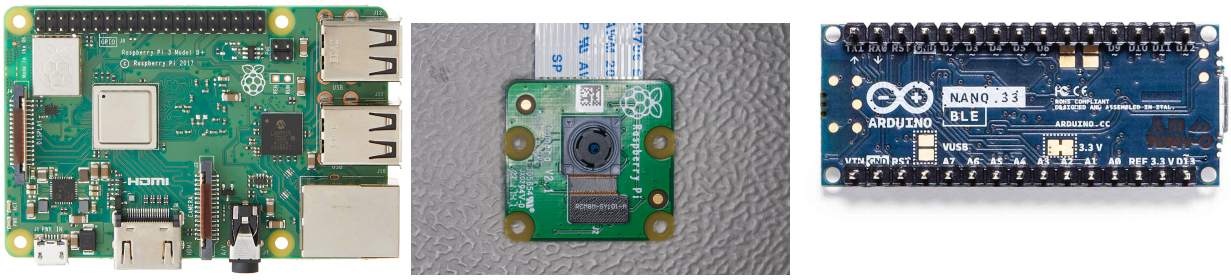


FIGURE 1 – Carte Raspberry / Caméra Raspberry / Carte Arduino

La carte Raspberry ainsi que la caméra constituent la partie principale du projet. La carte Arduino quant à elle nous a permis de réaliser un projet prototype que nous détaillerons par la suite.

2.2 Présentation de la partie logicielle

Nous avons utilisé plusieurs langages accompagnés de différentes bibliothèques :
— Python, langage adapté à la conception de modèles d'IA de par sa simplicité de prise en main et surtout ses bibliothèques qui simplifient beaucoup la création de réseaux de neurones. Nous nous en sommes également servis pour réaliser des programmes annexes ;
— langage C et C++ à travers l'environnement Arduino, afin de déployer les modèles sur la carte Arduino présentée ci-dessus.

Afin de développer nos réseaux de neurones, nous nous sommes servis des bibliothèques TensorFlow(Lite) ainsi que Keras (les versions utilisées seront spécifiées lors du détail de chaque projet). Ces bibliothèques contiennent notamment des fonctions permettant de réaliser un réseau de plusieurs couches en quelques lignes de code, et simplifient également son entraînement ainsi que son déploiement. Concernant TinyML, il s'agit d'un ensemble de méthodes, calculs, idées, qui permettent de déployer un modèle développé à partir d'un algorithme de Machine Learning (ML) sur système embarqué, d'où son appellation.

3 Elaboration du réseau de neurones pour la reconnaissance de lettres

Dans cette partie, nous avons voulu concevoir notre propre réseau de neurones convolutifs (CNN) pour effectuer de la reconnaissance de lettres en temps réel. Nous avons donc lu le livre "Intelligence Artificielle vulgarisée" (1) pour obtenir des méthodes et conseils sur l'élaboration de notre propre réseau de neurones. Globalement, pour effectuer de la reconnaissance de lettres en temps réel, il faut :

— Générer une architecture de CNN ;
— Entraîner le CNN ;
— Envoyer de nouvelles données au CNN.

3.1 Architecture de notre CNN

Notre réseau de neurones prendra en entrée des images de 28x28 pixels issues de la caméra de la Raspebrry PI 3B. Comme un réseau de neurones classique prend en entrée des nombres, il faut transformer ces images en nombres. Pour ce faire, nous utilisons des produits de convolutions et nous mettons à plat les matrices obtenues par ces différentes convolutions pour obtenir un vecteur que l'on pourra présenter en entrée à notre réseau de neurones.

Ainsi, l'architecture de notre CNN est dans une première étape constituée de 32 filtres convolutifs de taille 3x3 avec une fonction d'activation ReLU, et d'un étage de pooling pour réduire la taille des données obtenues pour passer de matrices 3x3 à des matrices 2x2. Finalement, une fois toute ces matrices obtenues, elles sont mises à plat pour ne former qu'un vecteur qui sera placé en entrée de notre réseau de neurones.

Concernant la sortie du réseau de neurones, elle est constituée de 26 neurones (qui correspondent aux 26 lettres de l'alphabet) associées à une fonction d'activation Softmax.

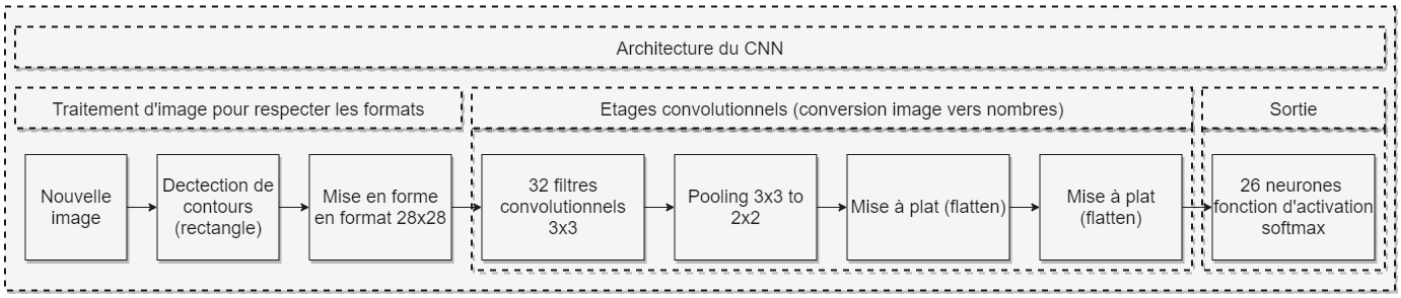


FIGURE 2 – Schéma synoptique de l'architecture mise en place

3.2 Entraînement de notre CNN

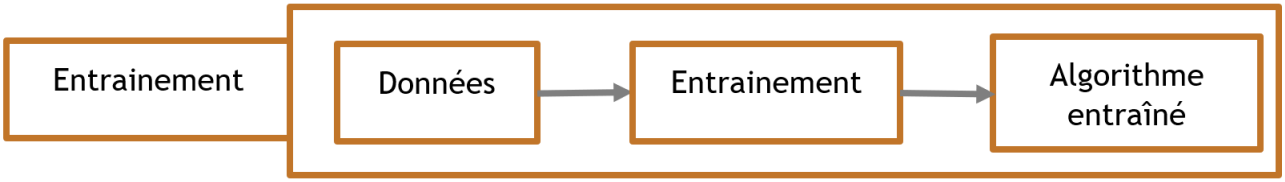


FIGURE 3 – Schéma synoptique du fonctionnement de notre algorithme d'entraînement

Pour effectuer l'entraînement de notre réseau de neurones, nous avons utilisé la banque de données des lettres étiquetées de MNIST. Dans celle-ci, il y a 124 800 lettres étiquetées : 75% de ces images servent à la phase d'apprentissage (93 600 images) et 25% sont utilisées pour la phase de validation de l'algorithme (31 200 images). On réduit ainsi la fonction de coût de la descente de gradients, ce qui permet d'augmenter la précision de notre algorithme de manière itérative.

```
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
732/732 [==============================] - 26s 32ms/step - loss: 1.0184 - accuracy: 0.7035 -
val_loss: 0.6924 - val_accuracy: 0.7948
Epoch 2/10
732/732 [==============================] - 24s 32ms/step - loss: 0.6261 - accuracy: 0.8137 -
val_loss: 0.5240 - val_accuracy: 0.8444
Epoch 3/10
732/732 [==============================] - 23s 32ms/step - loss: 0.5336 - accuracy: 0.8401 -
val_loss: 0.4758 - val_accuracy: 0.8592
Epoch 4/10
732/732 [==============================] - 23s 31ms/step - loss: 0.4867 - accuracy: 0.8538 -
val_loss: 0.4405 - val_accuracy: 0.8709
Epoch 5/10
732/732 [==============================] - 22s 31ms/step - loss: 0.4583 - accuracy: 0.8616 -
val_loss: 0.4256 - val_accuracy: 0.8746
Epoch 6/10
732/732 [==============================] - 22s 31ms/step - loss: 0.4380 - accuracy: 0.8683 -
val_loss: 0.4153 - val_accuracy: 0.8778
Epoch 7/10
732/732 [==============================] - 23s 31ms/step - loss: 0.4221 - accuracy: 0.8736 -
val_loss: 0.4062 - val_accuracy: 0.8810
Epoch 8/10
732/732 [==============================] - 23s 32ms/step - loss: 0.4079 - accuracy: 0.8774 -
val_loss: 0.3975 - val_accuracy: 0.8835
Epoch 9/10
732/732 [==============================] - 23s 32ms/step - loss: 0.4002 - accuracy: 0.8794 -
val_loss: 0.3914 - val_accuracy: 0.8863
Epoch 10/10
732/732 [==============================] - 23s 32ms/step - loss: 0.3922 - accuracy: 0.8826 -
```

FIGURE 4 – Entraînements successifs du modèle

Sur cette figure, on s'aperçoit que notre réseau de neurones a réussi à extraire des caractéristiques à partir de la base de données pour reconnaître les différentes lettres. En effet, on s'aperçoit qu'au fur et à mesure des itérations que la précision (accuracy) de notre réseau de neurones augmente : 70.35% dès la première itération et 88.26% à la 10ème itération.

Une fois le réseau de neurones fonctionnel, on le sauvegarde dans notre algorithme sous le nom "*modele_cas_pratique.h5*".

3.3 Utilisation de notre CNN

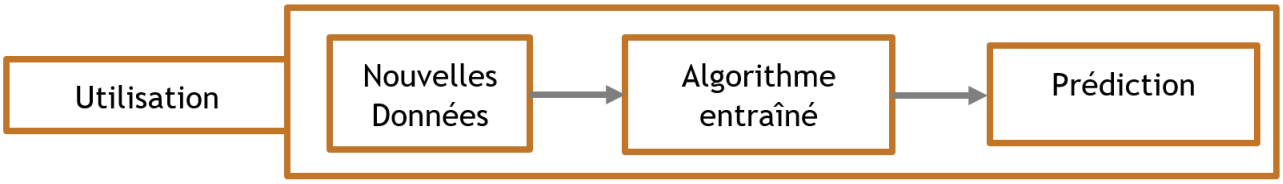


FIGURE 5 – Schéma synoptique du fonctionnement de notre algorithme lors de l'utilisation

Pour utiliser notre CNN et vérifier son bon fonctionnement, nous avons utilisé la caméra de la carte Rasberry PI 3B pour acquérir de nouvelles images. Cependant, notre réseau de neurones ne sait que reconnaître des lettres dans un format 28x28 pixels. Ainsi, il va falloir extraire, dans la vidéo, la partie où est située la lettre puis la redimensionner au format 28x28.

Pour effectuer cette tâche, nous avons principalement utilisé la bibliothèque OpenCV. Cette bibliothèque permet assez facilement d'utiliser des algorithmes de détection de contours et de reconnaissance de formes. Ainsi, pour réaliser quelque chose de simple et d'efficace, nous avons décidé qu'autour d'une lettre nous allons dessiner un rectangle, ce qui est facile à reconnaître avec un algorithme de détection de contours. Une fois ce rectangle détecte, on extrait son contenu et on le formate au format 28x28. Donc, si l'algorithme arrive à extraire le rectangle au bon endroit, la lettre de la vidéo filmée par la caméra de la Raspberry PI 3B sera envoyée au réseau de neurones et nous obtiendrons donc en sortie la prédiction sur cette lettre.

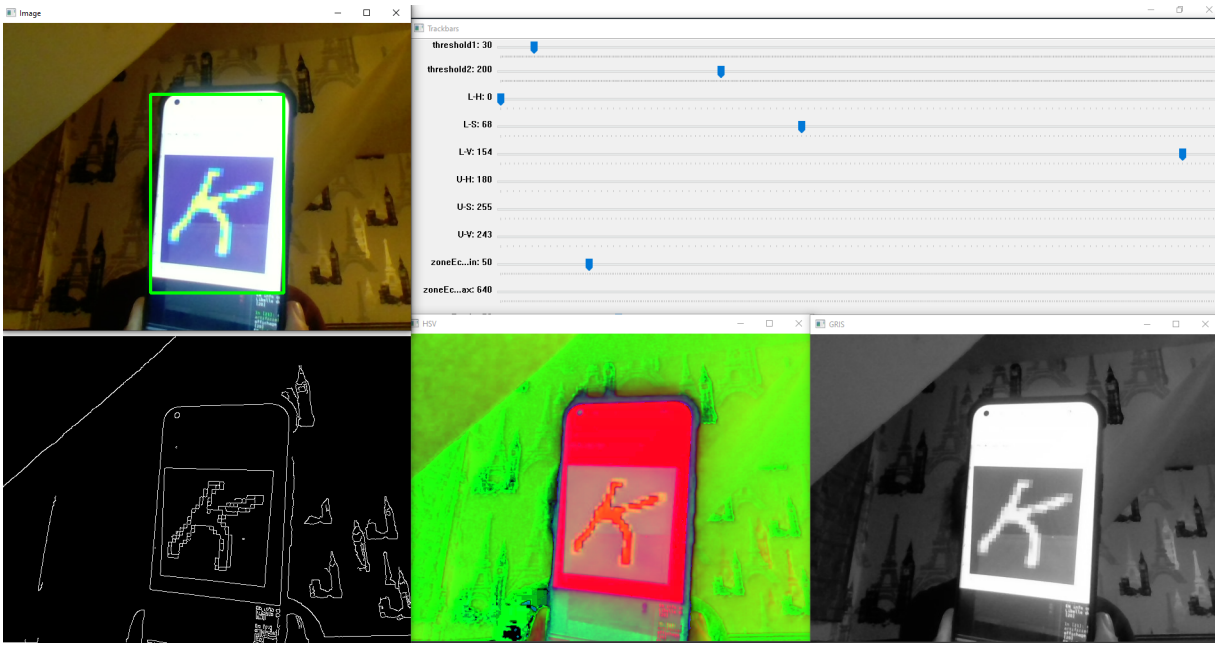


FIGURE 6 – Recherche des paramètres optimaux des algorithmes de détection de contours

Finalement, pour obtenir un résultat esthétique, nous avons décidé de faire apparaître en vert le rectangle détecté et d'afficher sur ce contour vert la prédiction de notre réseau de neurones. Puis nous utilisons la fonctionnalité "engine" de la bibliothèque *pyttsx3* pour que notre programme dise "Je lis la lettre : *Prediction(Lettre_*28×28*)*".

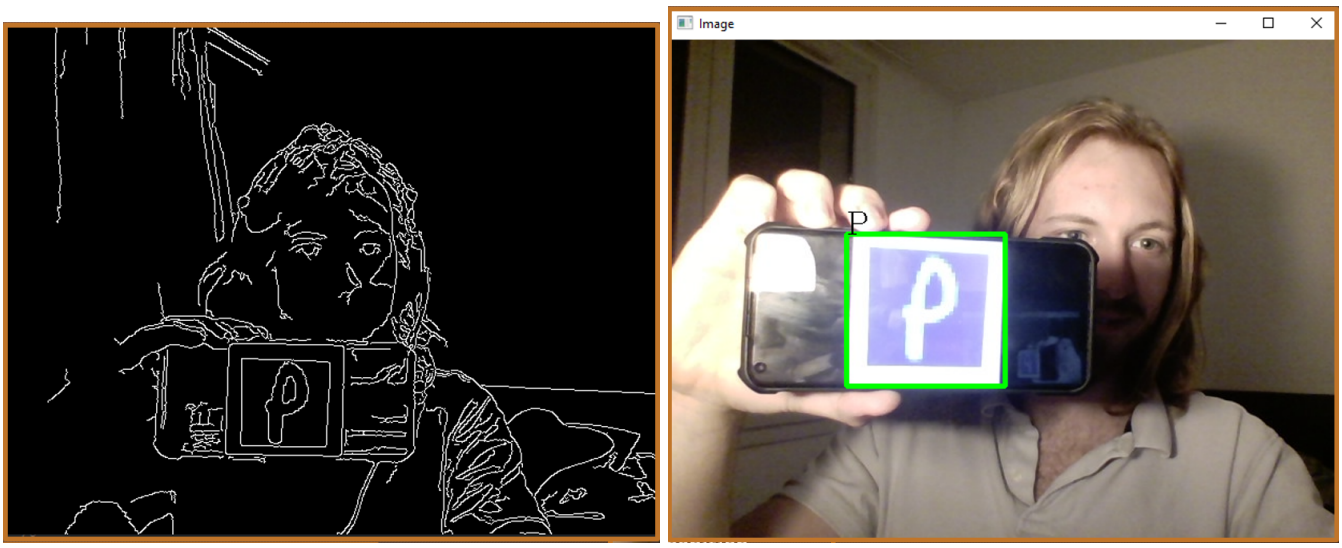


FIGURE 7 – Détection et reconnaissance en temps réel des lettres

On s'est donc servis de l'IA pour réaliser un modèle de reconnaissance de lettres. Cette approche aura permis de nous familiariser avec la bibliothèque TensorFlow en particulier, partie importante du projet à prendre en compte lorsque l'on voudra déployer ce modèle sur un système embarqué.

Vous trouverez dans le dossier "*Projet Projet IA/TinyML/Projet Reconnaissance Lettres Python*" les codes python qui ont permis d'implémenter ce système ainsi qu'une fiche tutoriel qui explique le fonctionnement du code et les versions requises des différentes librairies.

4 Création du modèle TinyML

Dans cette partie, nous nous sommes intéressés à la mise en place d'un modèle développé suivant les idées de TinyML. L'objectif est d'adapter le modèle de reconnaissance de lettres détaillé précédemment afin de le déployer sur un système embarqué.

La première étape de ce projet a consisté à mettre en application le modèle explicité dans le livre "TinyML : Machine Learning With TensorFlow Lite on Arduino and Ultra Low Power Microcontrollers" (2). Cet exemple est un cas d'école qui permet de prendre en main les différentes étapes qui permettent d'aboutir à un modèle déployé sur carte Arduino. Afin de comprendre comment créer notre propre modèle, il était important d'apprendre ces étapes, d'où l'intérêt de comprendre cet exemple (en plus de se familiariser avec les outils) donc de le refaire.

4.1 Création du modèle

L'exemple consiste à utiliser un réseau de neurones afin d'approcher une courbe sinusoïdale.

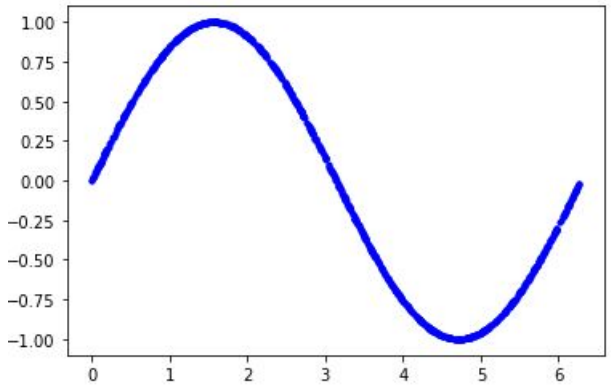


FIGURE 8 – Courbe sinusoïdale à approcher

L'objectif est donc de concevoir un réseau de neurones suffisamment performant afin d'approcher les valeurs que prend la fonction sinus (ici entre 0 et 2π), sans perdre de vue que le modèle doit rester déployable sur notre carte Arduino. Une contrainte d'occupation mémoire (ainsi que rapidité de calcul dans un second temps) est donc à considérer.

Le code utilisé est celui qui se retrouve dans le livre. Nous avons d'abord tenté d'installer les bibliothèques nécessaires en utilisant Anaconda Spyder. Cependant, il s'est avéré que nous avons rencontré de nombreux problèmes lors de la configuration de l'environnement de travail, en particulier une absence de la librairie TensorFlow, indispensable à notre projet. Afin de pallier ce problème, nous avons décidé d'effectuer le développement sur l'outil "Google Colaboratory", un outil de Google qui permet d'exécuter du code Python sur GPU en ligne. Il est ainsi beaucoup plus simple d'obtenir un environnement de travail adapté.

Nous avons alors étudié le code fourni par les auteurs du livre pas à pas. Or la version de TensorFlow utilisée lors de la rédaction du livre est la version "2.0.0-beta0". Nous n'avons pas réussi à installer cette version sur Google Colaboratory. Nous avons alors consulté le site internet des auteurs qui est mentionné au début de l'ouvrage, afin de vérifier si des errata ont été relevés (3). En effet, plusieurs errata ont été trouvés par des lecteurs, aussi un fichier source à jour comprenant tous les codes de l'exemple est mis à disposition sur GitHub (4). Nous avons donc suivi cette nouvelle version du code source.

Après avoir téléchargé les bibliothèques nécessaires au développement du modèle, nous pouvons plonger au coeur du développement.

La première étape consiste à visualiser notre objectif : approcher la courbe sinusoïdale comme indiqué précédemment. Afin que cela soit rendu possible, et comme pour tout modèle d'IA, il faut posséder suffisamment de données. Dans nos modèles précédents il s'agissait d'images de claviers, souris, téléphones portables ou lettres de l'alphabet, ici il est question

de points sur la courbe. Cependant, si les points que nous voulons utiliser pour l'entraînement du modèle sont situés directement sur la courbe, notre exercice n'a pas de sens, et n'est pas réaliste non plus : cela reviendrait à utiliser le résultat pour prédire ce même résultat. Il faut donc étendre la courbe de façon aléatoire, afin d'en extraire des points qui sont situés à proximité, mais pas dessus. Cela constitue nos différents groupes de données.

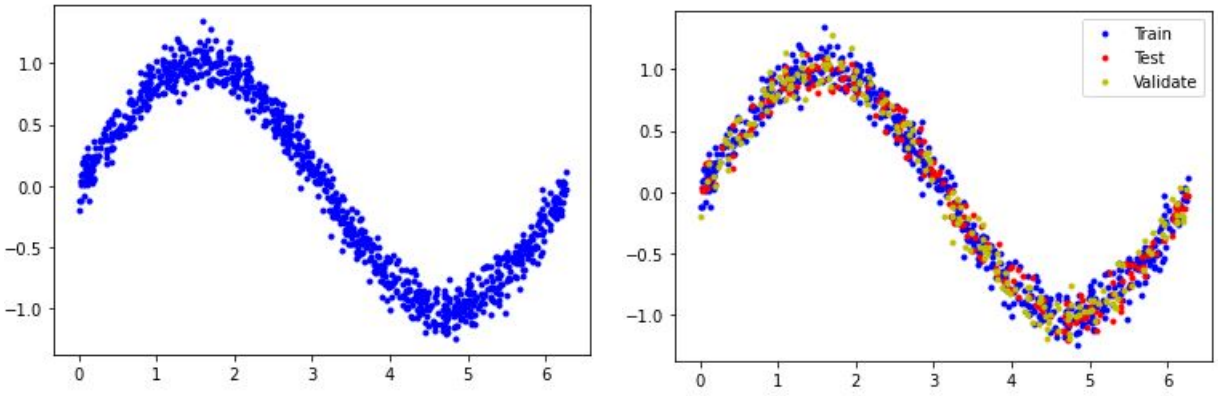


FIGURE 9 – Répartition des données étendues

Les données sont séparées en trois groupes différents :
— - "Train" : données d'entraînement. Elles permettent d'entraîner le modèle à approcher la courbe sinusoïdale ;
— - "Validation" : données de validation. Utilisées en parallèle des données d'entraînement, elles permettent de vérifier que notre modèle s'entraîne correctement ;
— - "Test" : données de test. A la fin de l'entraînement, elles permettent d'effectuer un test sur notre modèle et donc de constater si le résultat obtenu est conforme à ce que l'on attendait.

Maintenant que nous avons nos différents groupes de données, l'étape suivante est de sélectionner le type de réseau le plus adapté à notre objectif. Pour ce faire, on crée un premier réseau de neurones très simple de trois couches auquel on applique notre apprentissage.

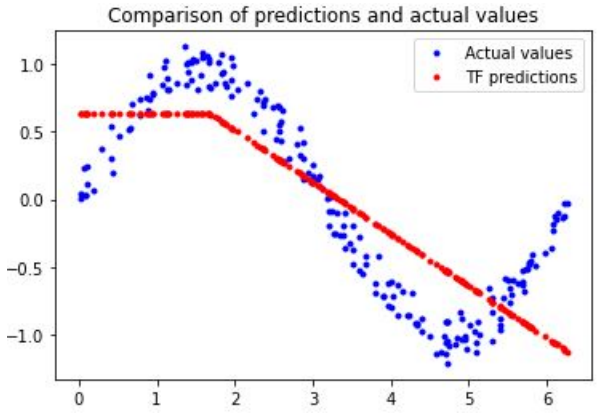


FIGURE 10 – Résultats liés au réseau de neurones très simple

On constate que le résultat en rouge ne suit pas du tout les valeurs en bleu qui correspondent aux données du sinus, et ce malgré le nombre important d'epochs, c'est-à-dire le nombre d'améliorations du réseau à la suite de chacun des entraînements successifs. Ceci est dû au fait que notre réseau est bien trop simpliste au vu de l'objectif que l'on souhaite atteindre.

Nous avons donc répété l'opération, cette fois avec un réseau plus complexe.

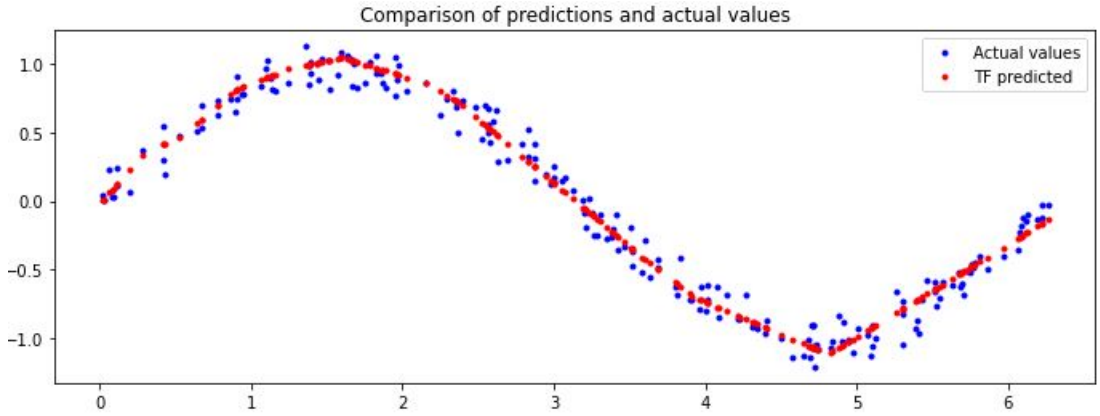


FIGURE 11 – Résultats liés au réseau de neurones complexe

Nous voyons cette fois que notre prédiction suit bien plus correctement les données en bleu. Le modèle que nous avons est satisfaisant, c'est celui-ci que nous allons retenir.

4.2 Conversion du modèle

Nous avons à présent un modèle construit autour de la bibliothèque TensorFlow fonctionnel. Il faut cependant pouvoir utiliser notre modèle sur notre carte Arduino. On procède donc à des optimisations pour réduire l'occupation mémoire de notre modèle. On utilise pour cela la bibliothèque liée à TensorFlow, TensorFlowLite.

Après avoir converti notre modèle pour qu'il soit exploitable par la bibliothèque, la première optimisation que l'on fait est une optimisation simple qui permet, en contrepartie d'une légère perte de précision sur le résultat, de grandement réduire la taille du modèle utilisé. Cette première optimisation est intéressante, mais celle que nous avons utilisée par la suite l'est bien plus : elle s'appelle "Quantization". Cette optimisation permet de transformer notre modèle de sorte que les calculs qui utilisaient précédemment des flottants sur 32 bits utilisent maintenant des entiers sur 8 bits. Le réseau est donc modifié de sorte que les opérations soient réalisées sans altérer le résultat. En plus de réduire drastiquement l'espace mémoire occupé, on accélère également les calculs, un CPU étant bien plus efficace pour faire des opérations sur des entiers que des flottants. On trace les résultats des différents modèles et on constate qu'ils sont toujours précis.

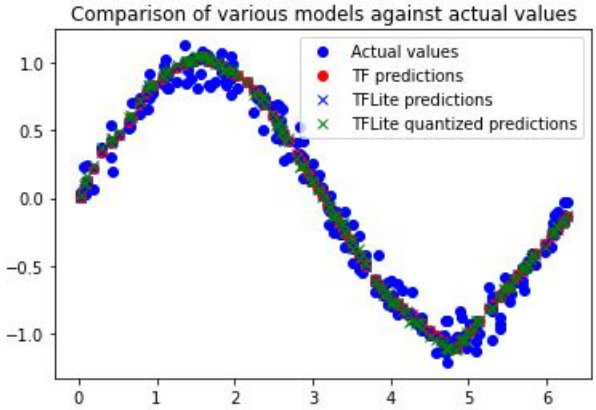


FIGURE 12 – Résultats liés aux différents réseaux optimisés

4.3 Déploiement du modèle

Nous avons un modèle optimisé. Pour pouvoir le déployer sur notre carte Arduino, il faut être en mesure de l'exploiter dans un format reconnaissable par le système. On le convertit donc en une séquence de "char" facilement exécutable par un microcontrôleur grâce à la commande "xxd" dans notre script Python. On obtient alors une séquence de 2408 caractères.

```
unsigned char g_model[] = {
  0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
  0x14, 0x00, 0x20, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x10, 0x00,
  0x14, 0x00, 0x00, 0x00, 0x18, 0x00, 0x1c, 0x00, 0x14, 0x00, 0x00, 0x00,
  0x03, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00,
  0x28, 0x01, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00, 0xd0, 0x00, 0x00, 0x00,
  0x48, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x98, 0x03, 0x00, 0x00,
  0x01, 0x00, 0x00, 0x00, 0x30, 0x01, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x00,
  0x10, 0x09, 0x00, 0x00, 0x0c, 0x09, 0x00, 0x00, 0xdc, 0x07, 0x00, 0x00,
```

FIGURE 13 – Extrait de la séquence de données obtenue

On a par la suite utilisé l'IDE d'Arduino. On a branché la carte Arduino sur le port USB du PC afin d'y envoyer la portion de code fournie également par les auteurs du livre. On y a uniquement modifié la séquence du modèle que l'on a développé. Afin de constater que notre modèle est satisfaisant et correctement implémenté, on contrôle l'éclairage de la LED en suivant les valeurs du sinus : lorsque le sinus atteint 0, la LED est éteinte, et plus on s'éloigne de 0 pour atteindre les valeurs extrêmes, plus la LED est allumée.

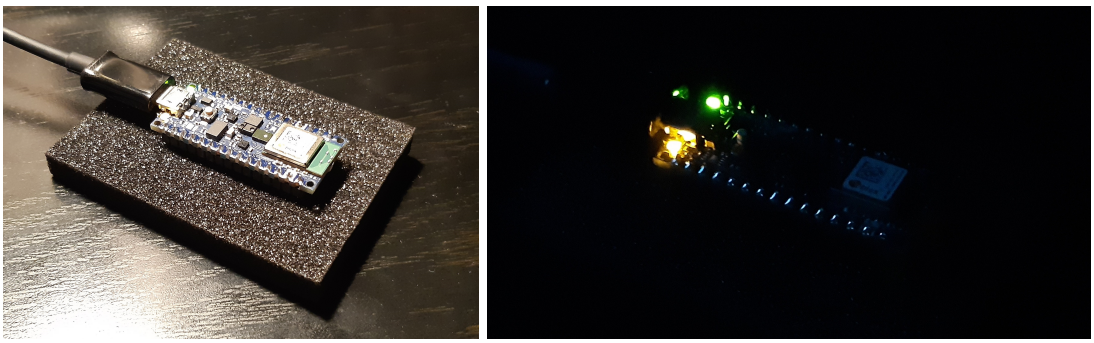


FIGURE 14 – Allumage de la LED sur la carte Arduino Nano BLE 33

On a donc réussi à répondre à une problématique, ici approcher les valeurs d'une fonction sur un intervalle donné, en utilisant l'Intelligence Artificielle, et ce modèle a été implémenté avec succès sur un système embarqué.

Maintenant que l'on connaît les différentes étapes à suivre pour développer notre modèle et le déployer, l'objectif est d'adapter notre modèle de reconnaissance de lettres pour l'intégrer à notre système embarqué.

4.4 Problèmes rencontrés

Nous avons souhaité suivre la même démarche que l'exemple du livre. Pour commencer, il a fallu adapter notre modèle pour approcher l'exemple du livre, pour par la suite et si on a un modèle concluant, prendre davantage de libertés et expérimenter les différentes optimisations.

Or c'est à partir de cette étape que nous avons rencontré des difficultés. En effet, nous avons majoritairement travaillé séparément sur les 3 projets. En ce sens et pour faciliter notre travail, chacun de nous a travaillé avec les versions de Python ou de TensorFlow qui était le plus adéquat. Lors de la fusion de notre travail, nous avons donc été confrontés à des problèmes de compatibilité entre les différentes versions des outils utilisés. En particulier, certaines fonctions qui permettaient de créer le réseau de neurones du projet de reconnaissance de lettres n'étaient pas disponibles dans la version

2.4.0 de TensorFlow utilisée pour le projet Arduino. Nous avons donc été bloqués à cette étape, mais cette piste aurait été creusée plus profondément si nous avions disposé de plus de temps.

Finalement, nous savons maintenant comment développer un projet d'IA pour l'embarqué. Il faut cependant être vigilant quant à la version des outils utilisés afin d'éviter tout problème de compatibilité.

5 Edge Impulse

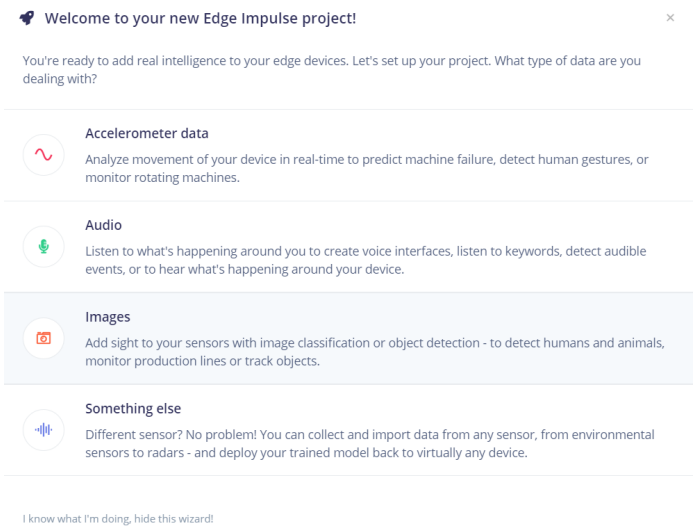
Étant donné que l'IA se démocratise de plus en plus, certains secteurs se tournent sur le développement d'outils permettant aux particuliers de créer leur propre modèle de Machine Learning. On peut citer par exemple Google Colab que nous avons utilisé précédemment, qui alloue à ses utilisateurs des ressources matérielles pour de l'apprentissage. On peut aussi citer Jupyter Notebook qui est une excellente alternative à Google Colab. Mais l'utilisation de ce genre d'outils nécessite des connaissances approfondies dans le domaine de l'IA, surtout en programmation, ce qui fait que ceux-ci sont souvent hors de portée des débutants et des curieux. Mais il existe un outil permettant de rendre l'IA accessible à une grande majorité : Edge Impulse.

En effet, Edge Impulse est un outil en ligne permettant de développer son propre modèle de réseaux de neurones, et de le déployer sur une grande variété de systèmes embarqués, tels que des cartes Raspberry Pi, des modèles avancés de carte Arduino ou encore des cartes Nvidia Jetson, et bien d'autres modèles connus dans le monde de l'IOT (*Internet Of Things* = Internet des objets).

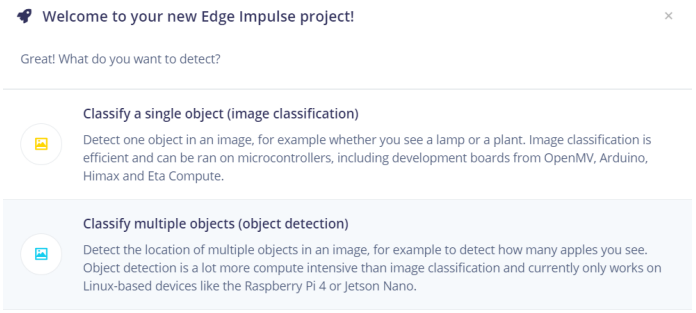
5.1 Premiers pas et création d'un modèle

Un projet Edge Impule se base sur les mêmes étapes de développement d'un modèle d'IA classique. En effet, ces différents étapes consistent à collecter des données, choisir le type de modèle voulu, entraîner le modèle avec la banque de données, et enfin déployer le modèle sur un système. Afin de nous initier au fonctionnement de Edge Impulse, nous avons décidé de créer un modèle de détection d'objet, capable de localiser sur une image ou sur une vidéo en direct des souris d'ordinateur, des claviers et des smartphones. Etant donné que nous travaillons dans une salle de projets, il sera plus facile de prendre en photo ce genre d'équipement.

1. **Création d'un projet**
   Lors de la création d'un projet Edge Impulse, un assistant permet de sélectionner le type de données à traiter et l'objectif du modèle d'IA souhaité pour ce projet. C'est une étape facultative car ces paramètres pourront être configurés plus tard.



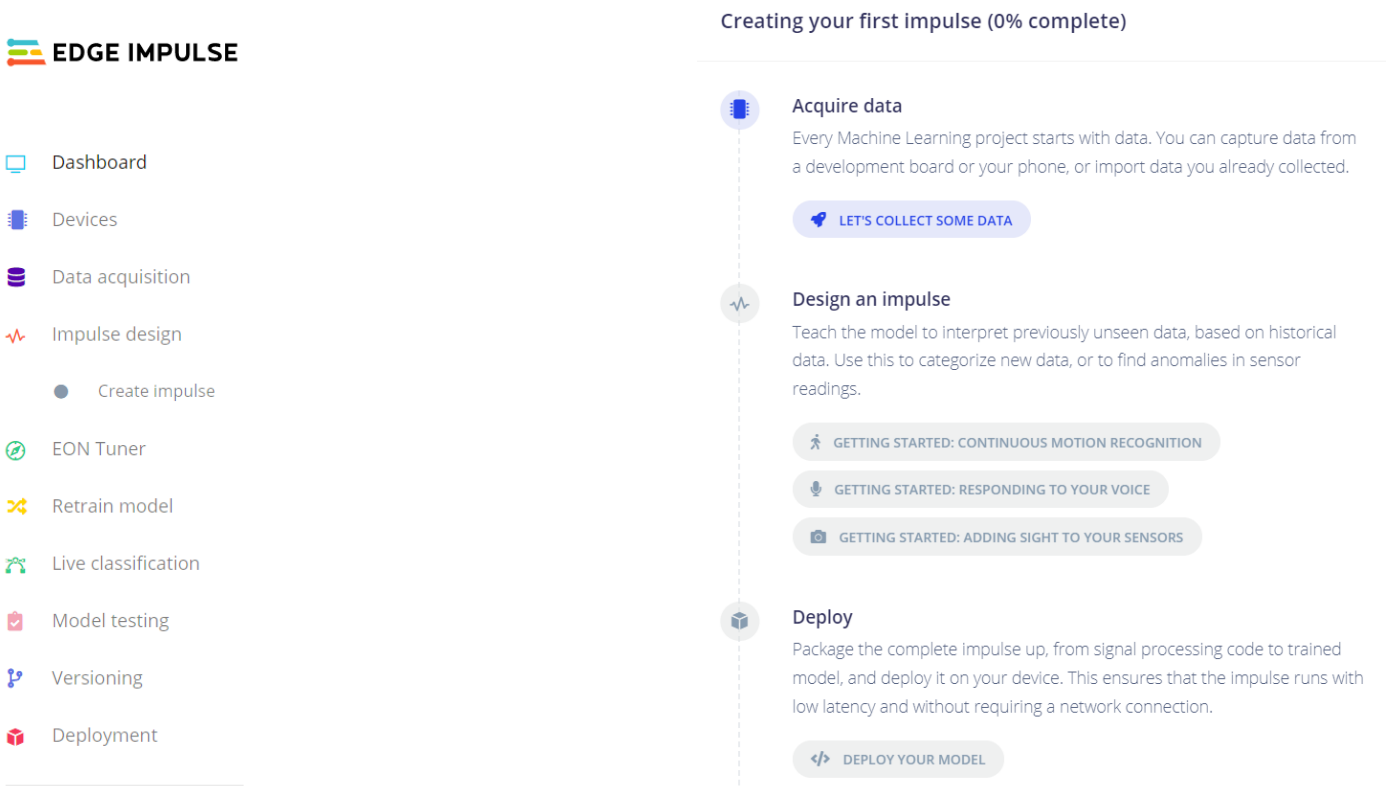
(a) Choix du type de données à traiter



(b) Choix du modèle d'IA

FIGURE 15 – Assistant de création de projet

Dans ce projet, nous choisirons donc le type de donnée "Images" pour le "détection d'objet". Une fois le projet créé, nous nous retrouvons sur un tableau de bord résumant les différentes étapes de développement du modèle, ainsi qu'un menu permettant de naviguer entre ces différents étapes.

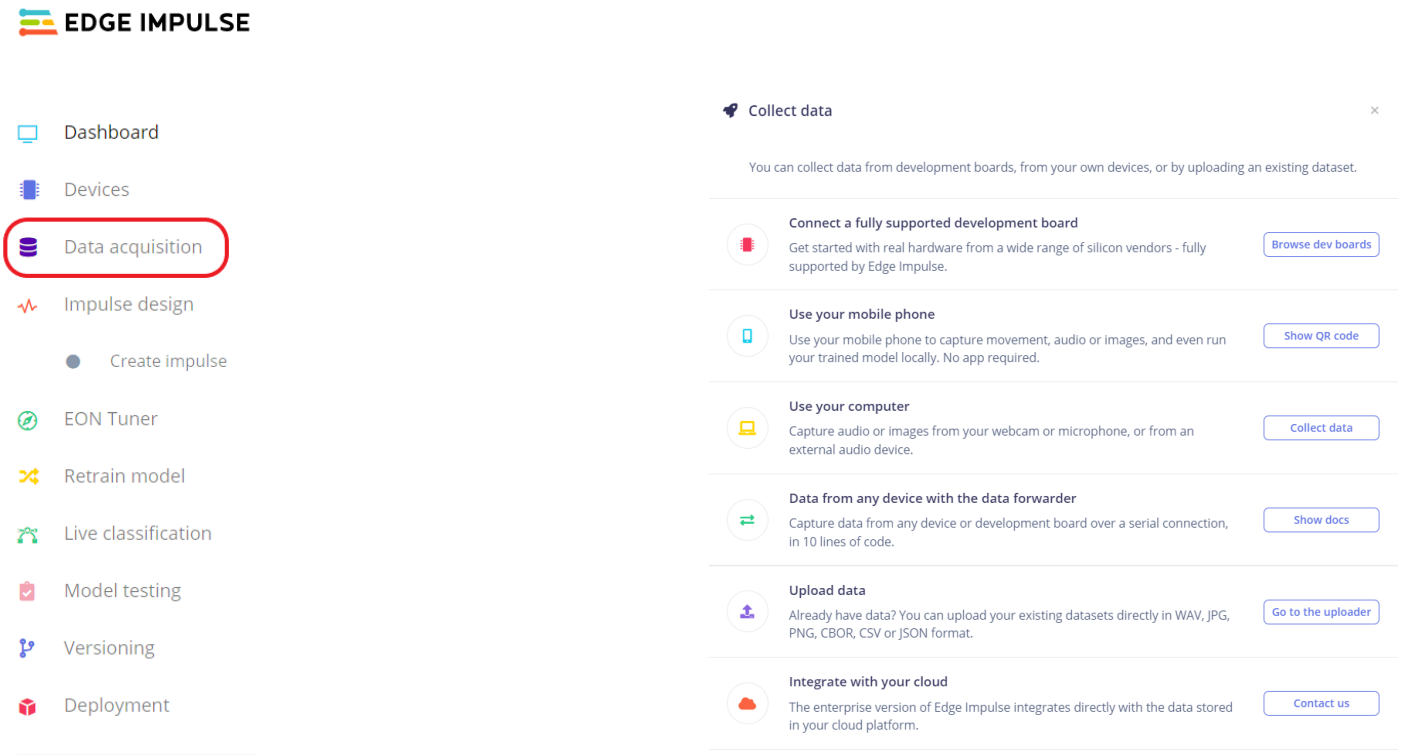


(a) Menu du projet

(b) Validation des différentes étapes de développement

FIGURE 16 – Tableau de bord du projet

2. **Collecte d'images**
   Une fois le projet créé, nous pouvons commencer à collecter des images de clavier, de souris et de smartphone. Pour cela, un menu est dédié pour la collecte de données.

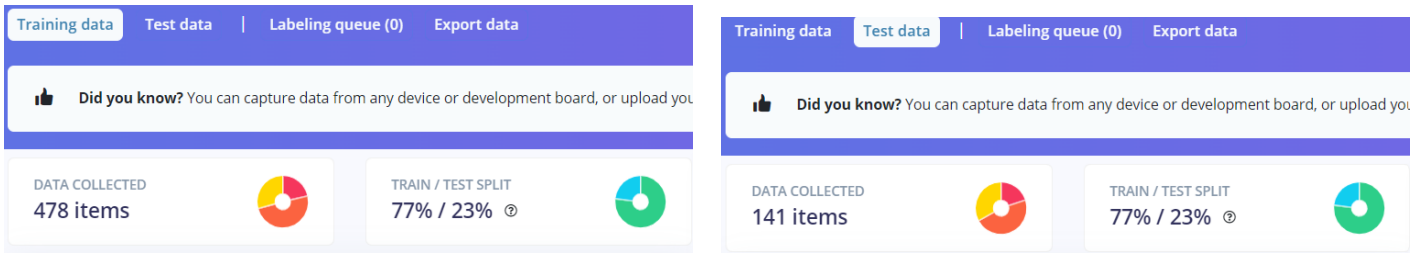


(a) Menu du projet

(b) Menu de collecte de données

FIGURE 17 – Collecte de données

Comme nous pouvons voir sur la figure précédente, la collecte de données est possible avec différents appareils. En effet, nous pouvons utiliser nos smartphones pour la collecte de photos et d'enregistrements audio (à l'aide d'un QR code généré par Edge Impulse). Nous pouvons aussi uploader directement des fichiers audio, images ou autre types de données à partir de nos ordinateurs, mais nous pouvons aussi utiliser la webcam et/ou le microphone de l'ordinateur pour capturer en live les données. Nous pouvons aussi récupérer des données (images, son, données de capteurs...) directement depuis un système embarqué type RaspberryPi, Arduino... Pour cela, Edge Impulse nous redirige vers son site dédié à l'acquisition de données sur support embarqué (pour la biblio : https ://docs.edgeimpulse.com/docs/cli-data-forwarder). Nous remarquerons que par défaut, les données se divisent en $\approx 80\%$ dans le "training set", utilisé pour l'entraînement, et $\approx 20\%$ dans le "testing set", utilisé ici pour les tests. Dans notre cas, nous avons majoritairement pris des photos de claviers, smartphones et souris disponibles dans la salle de TP. Une poignée d'images libres de droit a aussi été utilisé pour compléter la base de données d'images.



(a) "Training set"

(b) "Testing set"

FIGURE 18 – Répartition des données

3. **Annotation des images**
   Pour rappel, la problématique de notre projet ici est la localisation d'objet sur une image. Il est donc nécessaire d'annoter toute la banque d'images afin que le modèle puisse reconnaître les différents labels, a savoir : les claviers, les smartphones et les souris d'ordinateur.
   Pour cela, toujours dans l'onglet "Data acquisition", nous utilisons l'outil "labeling queue" permettant de localiser les différents objets. A chaque image, il est demandé de sélectionner, à l'aide de la souris, la ou les zones où se trouve(nt) les différents objets.

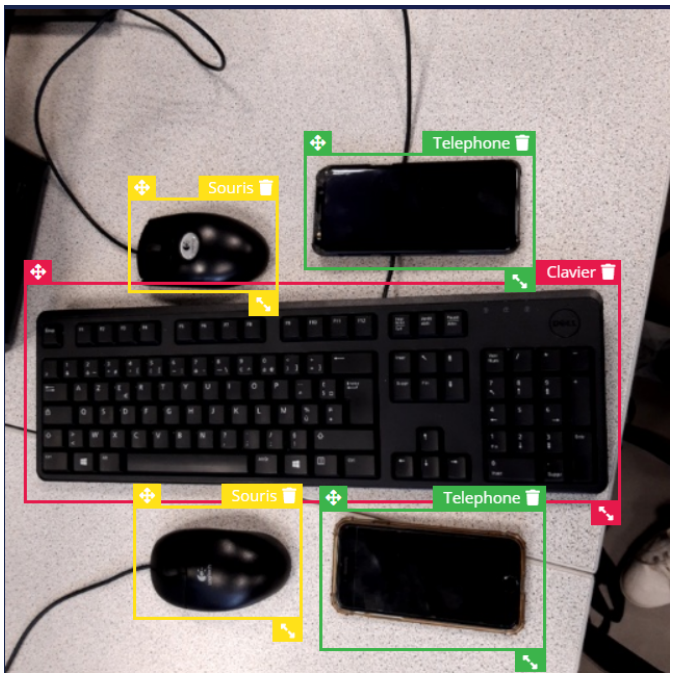
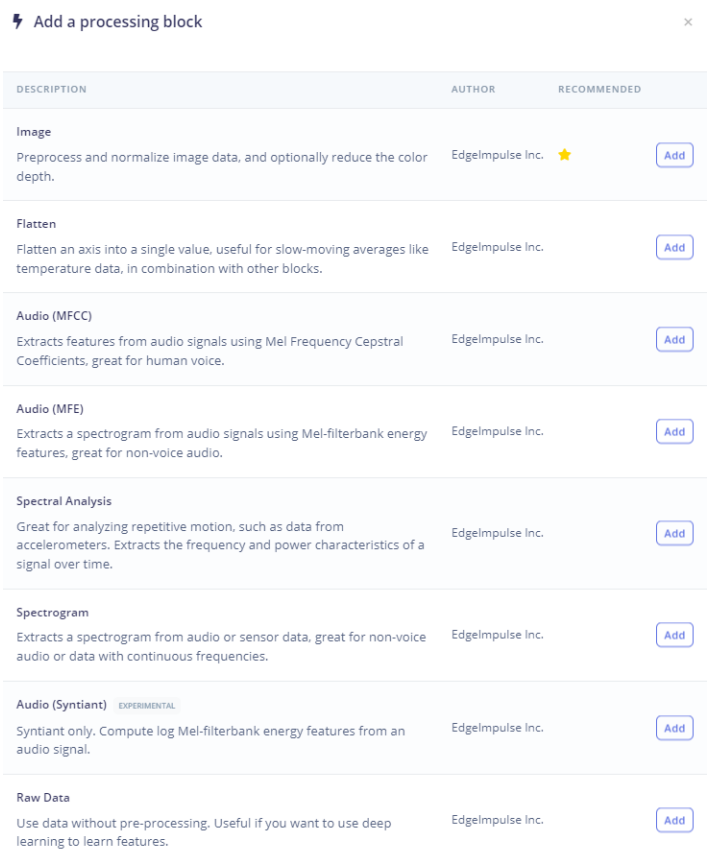


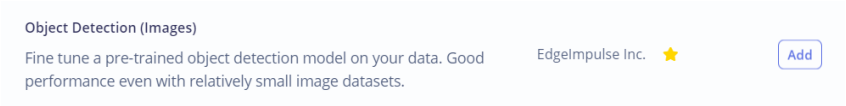
FIGURE 19 – Exemple d'annotation d'une image

   Il s'agit de l'étape la plus longue du processus de développement du au fait que la base de données est à annoter manuellement.

4. **Choix du modèle**
   Une fois la base de donnée annotée, nous pouvons designer le modèle de machine learning. Pour cela, dans l'onglet "Impulse Design", "Create Impulse", nous ajoutons un "processing block" permettant de sélectionner le type de données "image" à traiter. Nous ajoutons ensuite un "learning block" dédié à la détection d'objet.



(a) Choix du type de données à traiter



(b) Choix du modèle d'IA

FIGURE 20 – Paramétrage du design du modèle

Une fois le design paramétré, nous obtenons le récapitulatif suivant :

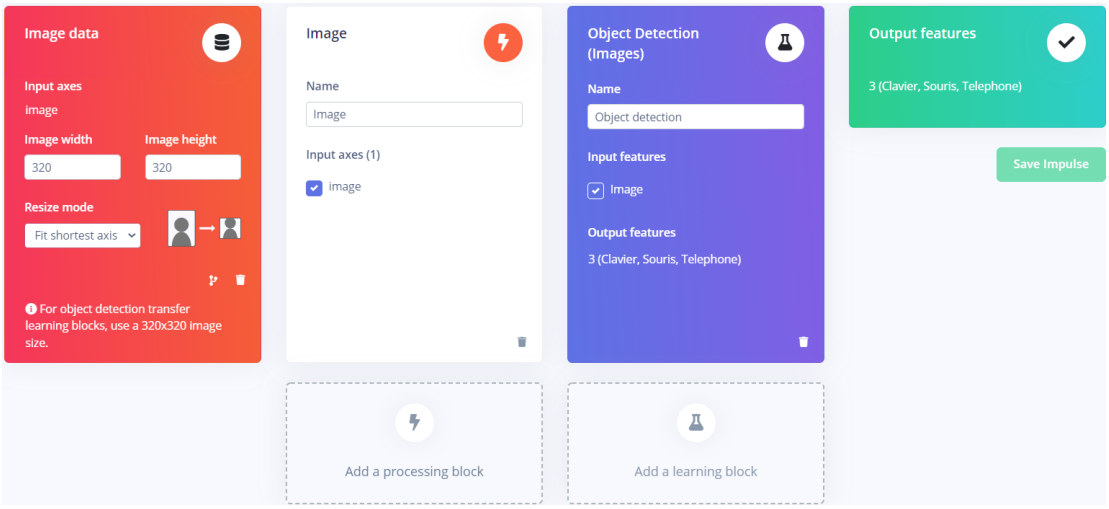


FIGURE 21 – Récapitulatif du design du modèle

Une fois ces paramètres sauvegardés, nous passons dans l'onglet suivant. Ici, l'objectif sera d'avoir une vision globale de la base de données d'images sur un graphique en 3D. Dans un premier temps, il sera proposé de choisir un type de traitement colorimétrique des images qui sera soit RGB (par défaut) ou soit nuance de gris. Nous garderons le traitement d'images par défaut. Ensuite, nous utilisons l'outil de génération de fonctionnalités "generate features". Cet outil permet de redimensionner toutes les images, appliquer le block de traitement d'images et visualiser en 3D le jeu de données complet en fonction de leurs labels. En effet, grâce à l'explorateur de fonctionnalité, les images sont regroupées par similarité. Plus les images sont similaires, plus nous verrons des zones et des régions se former sur le graphe.

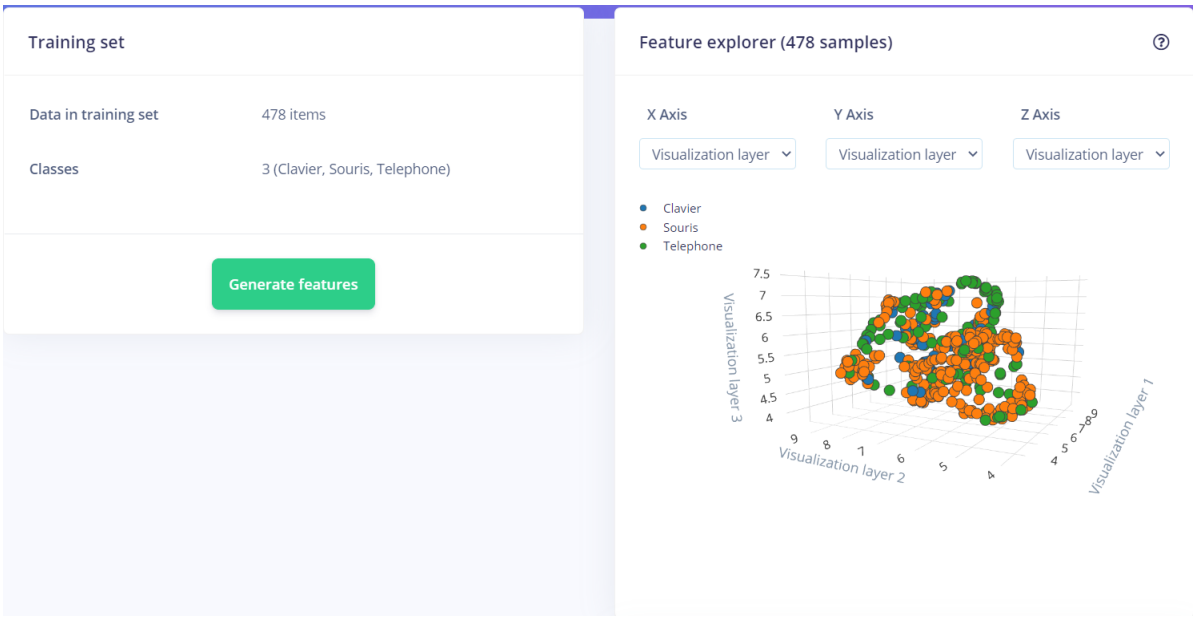


FIGURE 22 – Visualisation globale de la banque d'images

Un regroupement d'images de même label facilitera donc grandement l'apprentissage du modèle. Ici dans notre cas, nous pouvons voir que les images comportant un clavier forment une zone assez restreinte dans une partie du graphique, alors que les images de souris et de smartphones se confondent plus facilement. Il sera donc assez difficile d'avoir un modèle précis pour la détection de souris et de téléphone.

5. **Entraînement**

Nous pouvons maintenant passer à l'entraînement du modèle sur notre base de données. Pour cela, nous avons le contrôle de plusieurs hyperparamètres :

— Le nombre de cycles d'entraînement ;
— Le taux d'apprentissage ;
— Le pourcentage d'images d'entraînement utilisées pour la validation.

Il n'y a pas vraiment de règle sur le choix de ces paramètres mis à part le fait qu'il est important de les modifier à chaque nouvel entraînement afin d'obtenir un modèle plus précis après plusieurs tentatives. Généralement, nous commençons avec un nombre de cycle de 20 et un taux d'apprentissage de 0.01, ensuite nous les peaufinons à chaque entraînement.

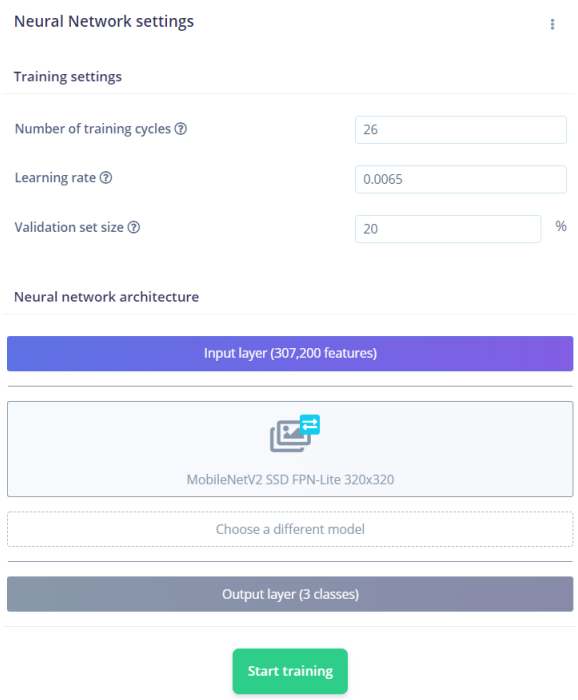


FIGURE 23 – Choix du modèle et des hyperparamètres

Pour l'apprentissage, nous utilisons le modèle pré-entraîné "MobileNetV2 SSD FPN-Lite 320x320". Il s'agit d'un modèle pré-entraîné à détecter jusqu'à 10 objets sur une image, et d'afficher en sortie la localisation des objets sous forme de boîte englobante. Le modèle a une taille d'environ 3,7 Mo et supporte en entrée des images couleurs de 320x320 pixels. Une fois l'entraînement lancé, une zone de la page indique l'état d'apprentissage du modèle, affichant entre autre les informations du numéro du cycle en cours, la précision atteinte à chaque cycle..., un peut à la manière d'un terminal. Et une fois l'apprentissage terminé, nous avons un récapitulatif indiquant la précision du modèle, la taille mémoire du modèle résultant et une approximation du temps d'inférence (temps de latence de traitement de chaque nouvelle image).

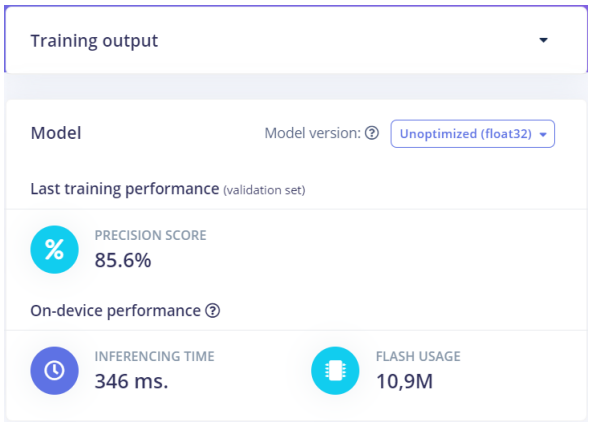
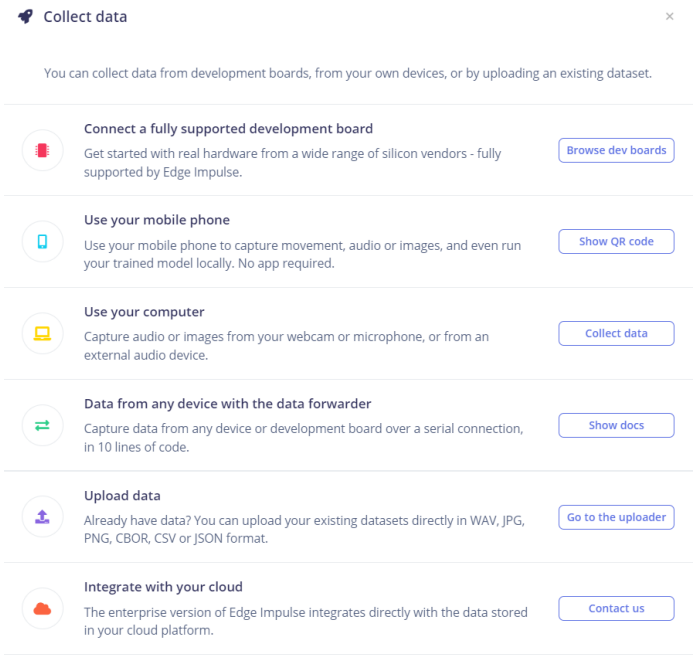


FIGURE 24 – Récapitulatif du modèle obtenu

6. **Déploiement**
   Nous avons maintenant un modèle près à être utilisé. Dans un premier temps, nous allons déployer ce modèle sur smartphone. Pour cela dans l'onglet "device", nous rajoutons notre smartphone à l'aide d'un QR code généré par Edge Impulse.



(a) Sélection de d'un smart-phone

(b) QR code pour l'appairage

FIGURE 25 – Appairage d'un smartphone au projet Edge Impulse

Après avoir flashé le QR code, nous nous retrouvons sur le menu permettant de soit collecter des données, soit utiliser le mode de classification d'images associé à notre modèle.

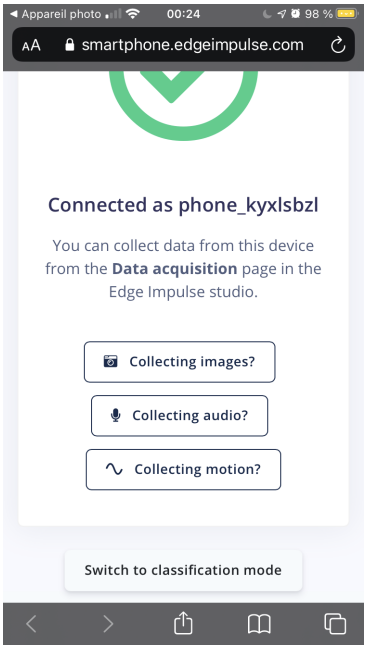


FIGURE 26 – Accueil du projet Edge Impulse sur smartphone

Une fois dans le mode classification, nous pouvons prendre en photo différents objets et vérifier l'efficacité du modèle de détection d'objets.

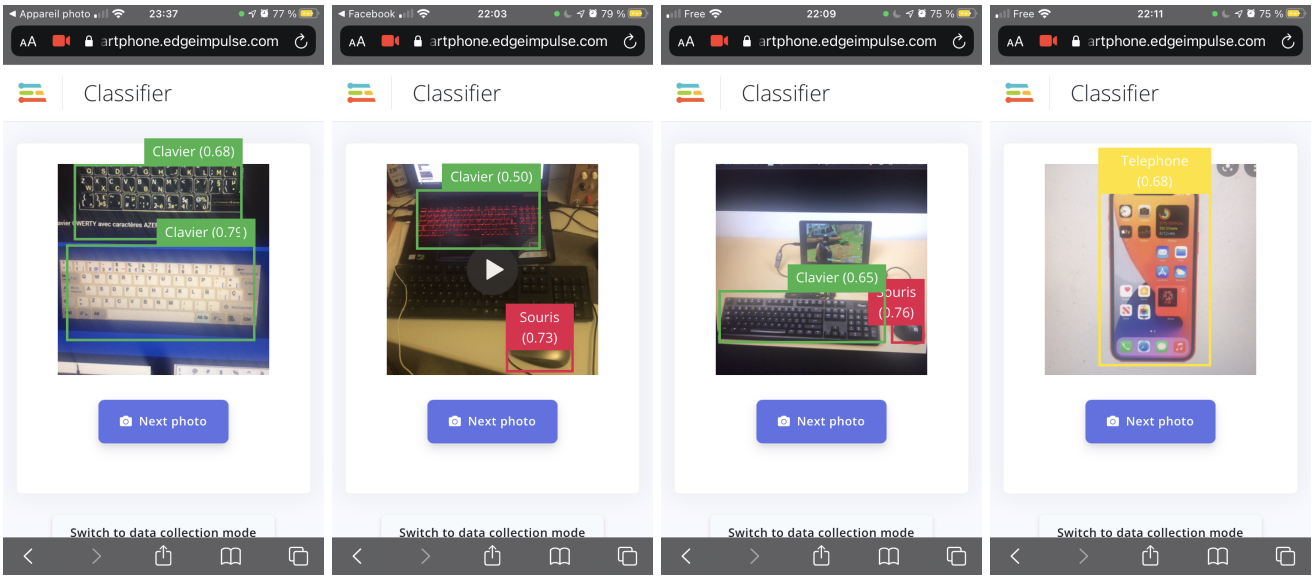


FIGURE 27 – Quelques tests du modèle Edge Impulse sur smartphone

Nous pouvons maintenant passer à la détection d'objets en live grâce à la Raspberry Pi et à sa caméra. Pour cela, quelques pré-requis sont nécessaires. En effet, il faut au préalable installer toutes les dépendances de Edge Impulse sur la carte Raspberry à l'aide des commandes suivantes :

```
sudo apt update

sudo apt upgrade

curl -sL https://deb.nodesource.com/setup_12.x | sudo bash -

sudo apt install -y gcc g++ make build-essential nodejs sox gstreamer1.0-tools
gstreamer1.0-plugins-good gstreamer1.0-plugins-base gstreamer1.0-plugins-base-apps

npm config set user root && sudo npm install edge-impulse-linux -g -unsafe-perm
```

Ensuite il faut activer la caméra de la Raspberry dans le menu `sudo raspi-config`. Après il faut appairer la Raspberry avec le projet Edge Impulse avec la commande `edge-impulse-linux` (le login et le mot de passe du compte Edge Impulse sera demandé). Une fois la Raspberry reconnue dans le tableau de bord Edge Impulse, nous pouvons déployer le modèle sur la carte avec la commande `edge-impulse-linux-runner`. Une fois le modèle compilé sur la carte, la Raspberry génère un serveur web permettant, grâce à son adresse ip, de voir la détection d'objet en direct sur n'importe quel navigateur web.

FIGURE 28 – Quelques tests du modèle Edge Impulse déployé sur Raspberry Pi

5.2 Augmentation du jeu de données

Après plusieurs entraînements successifs, nous avons constaté qu'il était difficile d'obtenir un modèle avec une précision élevée (maximum 86%), et ce malgré les différentes modifications des hyperparamètres en vu d'améliorer cette valeur. Le principal problème ici est le nombre d'images de la base de données. En effet, pour qu'un réseau de neurones soit précis, il est souvent nécessaire de lui faire utiliser plusieurs centaines voire milliers d'échantillons de données différents. Dans notre cas, quelques centaines de photos différentes n'étaient pas suffisantes, et prendre davantage de photos manuellement est long et fastidieux.

Nous avons donc développé un programme en Python nous permettant de générer des nouvelles images pour la banque de données. Pour ce faire, on sélectionne une image, puis on lui applique une rotation aléatoire afin de créer une nouvelle image. On peut alors répéter l'opération plusieurs fois sur autant d'images qu'on le souhaite pour faire augmenter drastiquement le nombre de photos dans la base de données. Seul restera le processus d'annotation, puisqu'il faudra toujours indiquer manuellement à l'outil où se trouvent nos différents objets sur chaque image.

Grâce à cette méthode, nous avons réussi à passer d'une base de données de 619 images à une base contenant un total de 3031 images. L'étape d'annotation a été le processus le plus long et pénible à réaliser.

5.3 Importation de jeux de données

Ici nous avons créé notre propre base de données avec des images et des labels personnalisés. Cependant, dans un contexte un peu plus avancé, il est également possible de s'intéresser à des bases de données publiques et de les utiliser directement dans Edge Impulse. En effet, certaines plate-formes comme Roboflow (5) proposent de créer des bases de données personnelles de manière plus avancée, ou alors de récupérer publiquement des bases de données annotées. Toutefois cette méthode nécessite des connaissances supplémentaires. Edge Impulse gère la labellisation des images au format JSON, et si l'on désire modifier les labels de la base de données importée par exemple, il sera nécessaire de modifier le fichier répertoriant ces labels.

```
{
    "version": 1,
    "type": "bounding-box-labels",
    "boundingBoxes": {
        "mypicture.jpg": [{
            "label": "jan",
            "x": 119,
            "y": 64,
            "width": 206,
            "height": 291
        }, {
            "label": "sami",
            "x": 377,
            "y": 270,
            "width": 158,
            "height": 165
        }]
    }
}
```

FIGURE 29 – Format d'annotation supporté par Edge Impulse

Cela peut prendre du temps, mais cela reste bien moins chronophage que de modifier manuellement chaque label sur Edge Impulse comme précédemment. Egalement, de plus amples connaissances en langage JSON permettraient d'exploiter ces bases de données plus profondément. Néanmoins, une solution a été retenue. En effet, sur la plate-forme de Roboflow, il est possible de télécharger un jeu d'images où plusieurs formats de fichiers d'annotations correspondant à plusieurs modèles de machine learning sont disponibles. Le format ressemblant le plus au format compatible avec Edge Impulse est le format *CreateML JSON format* utilisé pour les modèles *CreateML* de chez Apple ou *Turi Create tools*.

```
[
    {
        "image": "853209948_jpg.rf.0058dfb92ec10a7a1cb9d56eaf61ab30.jpg",
        "annotations": [
            {
                "label": "8",
                "coordinates": {
                    "x": 198,
                    "y": 79,
                    "width": 32,
                    "height": 36
                }
            },
            {
                "label": "14",
                "coordinates": {
                    "x": 229.5,
                    "y": 75,
                    "width": 29,
                    "height": 36
                }
            },
            {
                "label": "3",
                "coordinates": {
                    "x": 277.5,
                    "y": 79.5,
                    "width": 29,
                    "height": 33
                }
            }
        ]
    },
    {
        "image": "390636413_jpg.rf.0206c4d3741708779132f7d4b354fb03.jpg",
        "annotations": [
            {
                "label": "13",
                "coordinates": {
                    "x": 176,
                    "y": 238,
                    "width": 18,
                    "height": 16
                }
            },
            {
                "label": "3",
                "coordinates": {
                    "x": 198,
                    "y": 249,
                    "width": 24,
                    "height": 24
                }
            },
            {
                "label": "2",
                "coordinates": {
                    "x": 213.5,
                    "y": 262,
                    "width": 25,
                    "height": 26
                }
            }
        ]
    },
```


(a) Format d'annotation pour CreateML


```
{
    "version":1,
    "type":"bounding-box-labels",
    "boundingBoxes": {
        "853209948_jpg.rf.0058dfb92ec10a7a1cb9d56eaf61ab30.jpg": [
            {
                "label": "8",
                "x": 198,
                "y": 79,
                "width": 32,
                "height": 36
            },
            {
                "label": "14",
                "x": 229.5,
                "y": 75,
                "width": 29,
                "height": 36
            },
            {
                "label": "3",
                "x": 277.5,
                "y": 79.5,
                "width": 29,
                "height": 33
            }
        ],
        "390636413_jpg.rf.0206c4d3741708779132f7d4b354fb03.jpg": [
            {
                "label": "13",
                "x": 176,
                "y": 238,
                "width": 18,
                "height": 16
            },
            {
                "label": "3",
                "x": 198,
                "y": 249,
                "width": 24,
                "height": 24
            },
            {
                "label": "2",
                "x": 213.5,
                "y": 262,
                "width": 25,
                "height": 26
            }
        ],
```


(b) Format d'annotation pour Edge Impulse

FIGURE 30 – Adaptation du format JSON CreateML vers JSON Edge Impulse

Il faut aussi prêter attention au nom du fichier d'annotation pour qu'il soit reconnu par Edge Impulse lors de l'upload de la base de données dans le projet. En effet, ce fichier doit s'appeler "bounding_boxes.labels".

5.4 Limitation des ressources matérielles

Dans ce projet nous avons été contraints aux restrictions matérielles d'Edge Impulse. En effet, si la base de données contient trop d'images, alors l'entraînement ne peut pas se lancer (dépassement de la mémoire ram utilisée pour l'entraînement). De même, trop de labels dans un projet font saturer le programme et il est alors impossible d'entraîner notre modèle. En effet, nous avons importé dans un projet une base d'images de carte "uno" comportant une quinzaine de labels. Suite à ça, nous avons découvert que le modèle "MobileNetV2" utilisé par Edge Impulse pouvait une base d'images qui

contenait au maximum 10 labels. Ce modèle étant conçu pour des systèmes embarqués, il est nécessaire que ce modèle ne consomme pas beaucoup de ressources. Les ressources matérielles sont donc une contrainte majeure dans ce projet, et il est difficile de résoudre ce problème car ici nous utilisons un outil externe, que nous n'avons pas développé nous-mêmes. Ceci est le plus gros désavantage de ce genre d'outils qui certes sont très pratiques en surface lorsque l'on désire s'initier à l'IA ou réaliser de simples projets, mais qui dans un contexte professionnel par exemple, ou tout simplement lorsque l'on désire réaliser des projets plus complexes, ne proposent pas des solutions intéressantes, du moins pour leur version gratuite. Également, Edge Impulse est un outil qui a été développé assez récemment. La communauté derrière ce projet est donc encore naissante. Il n'est donc pas rare d'être confronté à des problèmes encore jamais rencontrés jusqu'ici, et de devoir contacter directement les développeurs afin de trouver des solutions.

6 Conclusion

Ce projet nous aura permis d'explorer plusieurs aspects de l'IA pour l'embarqué. Depuis la génération d'un réseau custom, que ce soit en utilisant un outil conçu pour simplifier le développement, ou grâce à des bibliothèques Python, jusqu'à son déploiement sur plusieurs plateformes, Raspberry ou Arduino, l'ensemble des projets que nous avons menés nous aura fait progresser dans ce domaine en expansion. Nous aurons également pu nous confronter aux difficultés lors de l'installation des outils ou de leur compatibilité mutuelle, ainsi qu'aux contraintes imposées par un outil ou un support hardware. En ce sens et avec toutes les connaissances que nous avons pu mettre en pratique, les pistes d'amélioration auraient pu être explorées davantage avec plus de temps notamment, ainsi qu'avec une étude des outils et fonctions des bibliothèques Python plus approfondie mais qui peut s'avérer complexe.

Pour retrouver tous les fichiers sources, datasets et autres documents, se rendre sur ce lien : `https://drive.google.com/drive/folders/15w8YzNfpyiGRO0fGTEi5VPhLZ8Kcbw9e?usp=sharing`

Références

[1] Aurélien VANNIEUWENHUYZE, "Intelligence artificielle vulgarisée", eni. France 2019

[2] Pete WARDEN and Daniel SITUNAYAKE, "TinyML : MachineLearning With TensorFlow Lite on Arduino and Ultra Low Power Microcontrollers", O'REILLY, 2020

[3] Lien vers les errata du livre [2] ci-dessus : `https://www.oreilly.com/catalog/errata.csp?isbn=0636920254508`

[4] Lien vers le fichier source mentionné dans le livre [2] ci-dessus : `https://colab.research.google.com/github/tensorflow/tflite-micro/blob/main/tensorflow/lite/micro/examples/hello_world/train/train_hello_world_model.ipynb`

[5] Lien vers des datasets prêts-à-l'emploi : `https://app.roboflow.com/`