DÉPARTEMENT DES SCIENCES ET DE L'INGÉNIERIE

PROJET PLDAC

Machine Learning embarqué

Auteurs:
Lucie CHEN 28610408
Karima SADYKOVA 28610100

Mai 2023

Sommaire

1 Remerciements **1**

2 Introduction **2**
2.1 Objectif du projet 2
2.2 Motivation 2

3 Prise en main **3**
3.1 Étude preliminaire 3
3.2 Réseau de neurones et Pytorch 3
3.3 Matériel utilisé 4

4 Benchmark **5**
4.1 Temps d'inférence CPU/GPU 5
4.1.1 Étapes à réaliser 5
4.1.2 Implémentation 5
4.1.3 Résultats 6
4.2 Temps d'inférence avec EdgeTPU 8
4.2.1 Implémentation 8
4.2.2 Résultats 9

5 Difficultés rencontrées **12**
5.1 GPU 12
5.2 Coral USB Accelerator 12
5.3 Jetson Nano 12

6 Conclusion **13**

7 References **14**

1 Remerciements

Nous souhaitons exprimer notre gratitude envers nos professeurs, Nicolas Baskiotis et Olivier Schwander, pour leur soutien et leur encadrement tout au long de ce projet. Leurs conseils avisés et leur expertise dans les domaines des systèmes embarqués et de l'apprentissage automatique ont été d'une grande valeur pour notre travail.

Ce projet nous a offert une expérience précieuse pour nous familiariser avec les systèmes embarqués et les concepts de l'apprentissage automatique. Grâce à leur guidage, nous avons pu acquérir des connaissances pratiques et approfondir notre compréhension de ces domaines importants. Nous tenons également à remercier tous ceux qui ont contribué, directement ou indirectement, à la réalisation de ce projet et à notre parcours d'apprentissage.

2 Introduction

2.1 Objectif du projet

L'objectif principal de ce projet est d'appliquer nos connaissances en apprentissage automatique afin d'évaluer les possibilités et performances de différents dispositifs, telles que le Coral USB Accelerator de Google et la carte Jetson Nano de NVIDIA, par rapport à l'utilisation du CPU et du GPU d'un ordinateur. Nous souhaitons analyser et comparer les résultats obtenus pour déterminer quelle solution offre la meilleur performance en termes de temps de calcul.

En plus de cette comparaison, si le temps nous le permet, nous avons également pour ambition de développer une chaîne de traitement pour la conduite autonome d'un petit robot-voiture. Ce projet nous permettra ainsi d'explorer les possibilités offertes par les différentes plateformes matérielles et de mettre en pratique nos compétences en programmation, en utilisant principalement Python et des bibliothèques complémentaires telles que pytorch et tensorflowLite.

Au cours de ce projet, nous aborderons des sujets tels que la détection et classification des objets dans des images, l'optimisation des performances des réseaux de neurones et l'apprentissage par transfert. Nous espérons ainsi acquérir une meilleure compréhension des systèmes embarqués et des défis liés à l'intégration de l'apprentissage automatique dans des environnements à ressources limitées.

(a) Jetson Nano (b) Coral

Figure 1: Matériel utilisé

2.2 Motivation

L'avancement de l'apprentissage automatique est sans précédent dans l'histoire, chaque jour nous arrivons à créer des nouveaux pics. Mais est-ce qu'on peut élargir l'utilisation, comme l'exemple de l'ordinateur portable, à l'échelle d'une personne.

Par ce projet, on veut découvrir les possibilités d'une analyse et traitement de vidéo à temps réel. Pour mieux visualiser l'objectif, on peux se poser une question : est-ce qu'on peut appliquer efficacement de l'apprentissage automatique à une petite carte portable, qui pourra aider une personne malvoyante à classifier et détecter les objets à temps réel?

3 Prise en main

L'objectif de cette section est de poser les bases nécessaires pour la mise en œuvre du projet et de nous assurer que nous sommes en mesure d'utiliser efficacement les technologies choisies. Nous décrirons les étapes de configuration et de préparation des plateformes pour la suite de notre projet.

3.1 Étude preliminaire

Dans cette section, nous avons réalisé une étude préliminaire afin de choisir les outils appropriés pour notre projet.

Nous avons d'abord approfondi nos connaissances sur le fonctionnement des réseaux de neurones, en particulier des CNN, étant donné que notre projet porte sur la traitement d'images et vidéos comme la classification et détection d'objets. Nous avons abordé les notiosn de perceptron, de l'utilisation de fonctions non linéaires et d'activation pour les réseaux de neurones. Nous avons également évoqué les problèmes d'invariance et l'importance des caractéristiques automatiques guidées par l'apprentissage.

En ce qui concerne l'image, nous avons discuté de l'utilisation d'histogrammes de patches pour détecter la présence de motifs dans une image. Nous avons également abordé les filtres, les convolutions, les opérations différentiables et l'apprentissage des filtres par rétropropagation. Nous avons mentionné l'utilisation de techniques telles que le pooling et la fonction ReLu (Rectified Linear Unit) pour résumer l'information et augmenter la réactivité des réseaux.

Pour la mise en œuvre, nous avons opté pour l'utilisation de PyTorch en raison de sa robustesse et de sa facilité de construction et d'expérimentation de différentes architectures de réseaux neuronaux. Nous avons donc consulté la documentation de PyTorch pour comprendre comment procéder.

L'étape de recherche a été complétée par le travail collaborée avec les bibliothécaires qui nous ont appris à faire des recherches d'information fiables et pertinentes. Ce travail a donnée fin à un carnet de bord.

Pour mieux s'organiser nous avons créé des canaux de communication (Mattermost, GitHub/GitLab, Overleaf), et nous avons étudié le tutoriel de PyTorch sur les CNN. Nous avons également décidé d'orienter notre projet vers des solutions embarquées et la détection d'objets. De plus, nous avons pris en main le matériel nécessaire, notamment la carte Jetson nano et l'accélerateur Coral, en suivant des tutoriels appropriés.

Cette étude préliminaire nous a permis d'établir une base solide pour la suite de notre projet. Nous avons acquis des connaissances essentielles sur les concepts de base de l'apprentissage automatique et nous sommes maintenant prêts à passer à l'étape suivante : la conception et l'implémentation de notre système de détection d'objets embarqué.

3.2 Réseau de neurones et Pytorch

Dans cette section, nous aborderons l'étape de recherche et de prise en main des différentes technologies et plateformes nécessaires à notre projet. Nous commencerons

par étudier en détail la Jetson Nano de NVIDIA et l'accélérateur USB Coral de Google. Nous examinerons également les différentes méthodes de calcul et d'inférence disponibles, telles que l'utilisation du CPU et du GPU de l'ordinateur.

Nous avons également consacré du temps à l'apprentissage de PyTorch en suivant le tutoriel "60 minutes blitz" disponible sur le site officiel de PyTorch. Nous nous sommes également penchés sur la détection d'objets avec la bibliothèque Torchvision.

De plus, nous avons réussi à charger des modèles pré-entraînés avec PyTorch, à les entraîner et à effectuer des prédictions sur les images du tutoriel à l'aide de nos ordinateurs. Cela nous a permis de nous familiariser avec les différentes étapes du processus d'apprentissage et d'inférence des modèles.

Une des tâches à venir consiste à réaliser un benchmark pour évaluer les performances des différentes architectures de modèles existants. Nous prévoyons d'utiliser des modèles tels que VGG et AlexNet disponibles dans la bibliothèque PyTorch. Nous effectuerons des tests pour mesurer le temps d'exécution, la capacité mémoire et d'autres aspects pertinents sur différents systèmes, notamment les GPU classiques, la carte Jetson et la carte Coral avec Edge TPU. L'objectif est de comprendre les limites et les possibilités des systèmes embarqués en termes de traitement d'images.

Ces expérimentations nous permettront d'obtenir des informations précieuses pour orienter nos prochaines étapes, notamment en ce qui concerne l'apprentissage et l'application des modèles sur des systèmes embarqués.

3.3 Matériel utilisé

Nous avons effectué nos implémentations et expérimentations sur les dispositifs suivants :

- Macbook Pro, 4 coeurs 2 GHz Intel, 16GB RAM
- Ordinateurs PPTI avec GPU RTX 2080
- Coral USB Accelerator de Google
- Carte Jetson Nano de NVIDI avec Ubuntu 18.04

4 Benchmark

Notre objectif est de comparer les temps d'inférence sur différents systèmes : l'ordinateur utilisant le CPU, l'odinateur utilisant le GPU, l'ordinateur avec l'accélerateur Coral (EdgeTPU), la carte Jetson.

4.1 Temps d'inférence CPU/GPU

4.1.1 Étapes à réaliser

Nous souhaitons construire un benchmark afin d'évaluer les performances de différents pré-modèles de réseau de neurones.

Voici ce que nous avons réalisé :

- Nous avons suivi le tutoriel de PyTorch sur le finetuning des modèles torchvision.
- Nous avons utilisé la base de données CIFAR10.
- Nous avons entrainé et inférée sur des pré-modèles existants tels qu'AlexNet, VGG, et d'autres afin de les comparer.
- Nous avons testé différentes tailles de batch (16, 32, 64, etc.) pour voir comment cela impacte les performances. Nous faisons l'hypothèse que plus la taille de batch est grande, plus le temps d'inférence est lent et qu'au-delà d'une certaine taille de batch, l'exécution deviendrais impossible.
- Pour récapituler nos résultats, nous avons compilé un tableau comparant les vitesses d'exécution des modèles pour différentes tailles de batch.

4.1.2 Implémentation

Nous expliquerons ici, l'écriture du script qui nous a permis d'effectuer le benchmark sur plusieurs modèles, tels que VGG et AlexNet, en utilisant différentes tailles de batch.

Lors de l'implémentation, nous constatons que nous n'avons pas besoin de la partie apprentissage car cela n'influence pas sur le temps d'inférence.

Ainsi, nous avons chargé le modèle, puis bouclé sur une liste de batch-size. Pour chaque batch-size, nous lançons le modèle pour classifier 10000 images afin d'obtenir le temps moyen d'inférence pour une image.

Ensuite, nous avons refait les calculs pour tous les modèles sur une seule même image.

Ces tests nous permettront d'évaluer la différence des performances obtenues en utilisant différentes configurations matérielles. Ainsi, nous pourrons déterminer quel système offre les meilleures performances en termes de temps d'inférence pour nos modèles.

4.1.3 Résultats

Sur CPU

Les résultats des tests sur temps d'inférence, sur le CPU, sur 10000 images, en secondes, sont présentés dans le tableau suivant :

Modèle / Batch_size	16	32	64	128	256	512	1024
ResNet	20.511	13.938	11.404	9.238	8.460	8.884	8.618
AlexNet	121.701	114.122	112.790	115.851	108.952	119.650	112.001
SqueezeNet1_0	10.092	6.910	6.410	6.376	6.031	6.495	8.346
Googlenet	25.272	19.831	15.686	14.344	13.240	13.721	15.418
VGG 16	87.752	71.171	62.073	55.516	52.374	53.873	53.361
VGG 19	90.430	72.951	64.688	60.633	58.506	59.719	61.928

Table 1: Temps d'inférence en fonction du batch_size sur CPU

Nous avons pu représenter aussi en format image :

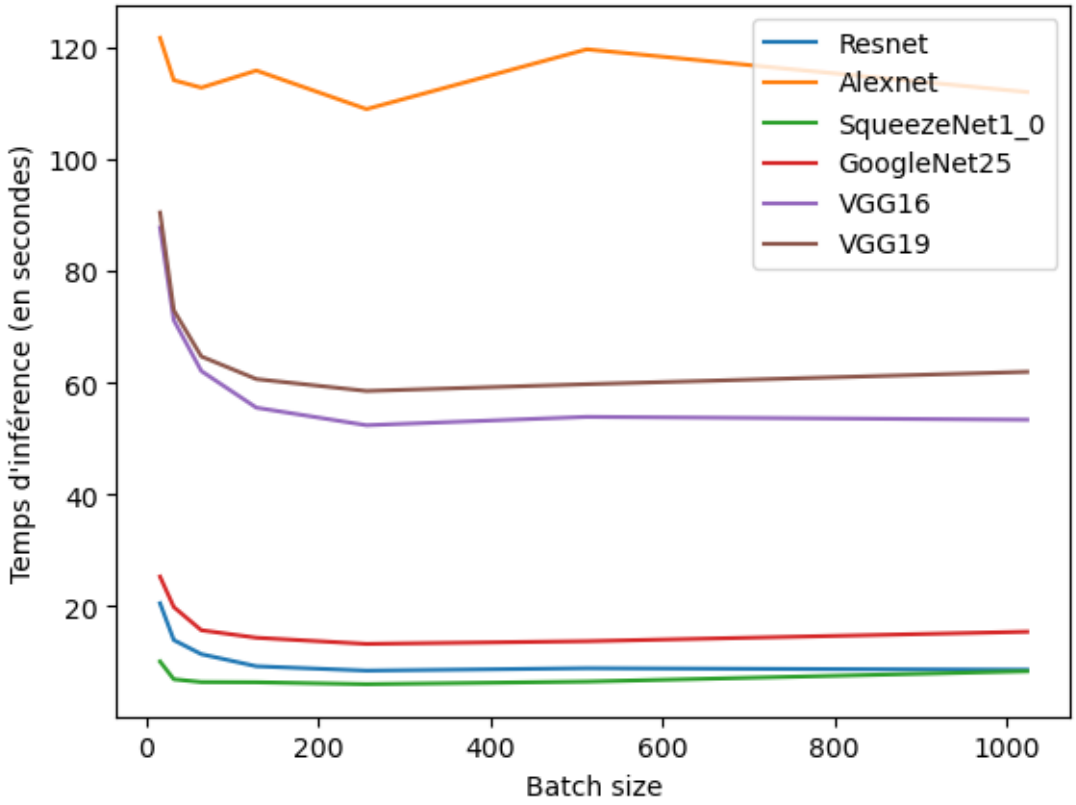

Figure 2: Temps d'inférence en fonction du batch_size sur CPU

Les modèles qui donnent les meilleurs résultats sont SqueezeNet1_0, Resnet et GoogleNet25. Ensuite on a VGG et en dernier AlexNet.

Nous constatons sur ces résultats que notre hypothèse faite précédemment sur le batch-size était faux. Le temps d'inférence ne dépend pas du batch-size, on ne

vois qu'une descente entre 16 et 32. Et sachant que la carte coral ne prend qu'une image pour calculer le temps d'inférence, nous refaisons les tests sur une image.

Nous travaillons aussi sur plusieurs modèles et nous régardons la différence de temps d'inférence en fonction de la taille des modèles, ci-dessous les tables qui contiennent le temps d'inférence d'une seule image sur CPU, en ms:

ResNet	18	34	50	101	152
	115.368	120.668	122.374	127.171	133.668

Table 2: Temps d'inférence sur CPU - Resnet

AlexNet
132.637

Table 3: Temps d'inférence sur CPU - AlexNet

SqueezeNet	1_0	1_1
	116.346	126.462

Table 4: Temps d'inférence sur CPU - SqueezeNet

MobileNet	v2	v3_Small	v3_Large
	128.578	129.864	166.465

Table 5: Temps d'inférence sur CPU - MobileNet

VGG	11	13	16	19
	136.048	182.250	168.671	171.614

Table 6: Temps d'inférence sur CPU - VGG

On voir que pour chaque modèle, plus la taille du modèle est grand, plus l'inférence prend du temps. Ainsi, on préfère d'utiliser des modèles de taille petits.

En comparant le temps d'inférence minimale de chaque modèle, on a que ResNet et SqueezeNet sont les plus rapides, puis suivent MobileNet, AlexNet et VGG. Cette comparaison nous donne une idée initiale des performances relatives des modèles que nous avons testés.

Sur GPU

Les mêmes calculs sont effectuées mais sur GPU, et on obtient les tables qui suivent :

ResNet	18	34	50	101	152
	4.166	5.994	7.634	10.201	14.463

Table 7: Temps d'inférence sur GPU - Resnet

AlexNet
2.718

Table 8: Temps d'inférence sur GPU - AlexNet

SqueezeNet	1_0	1_1
	3.253	3.327

Table 9: Temps d'inférence sur GPU - SqueezeNet

MobileNet	v2	v3_Small	v3_Large
	67.142	65.874	5.236

Table 10: Temps d'inférence sur GPU - MobileNet

VGG	11	13	16	19
	3.393	3.492	3.948	4.472

Table 11: Temps d'inférence sur GPU - VGG

Les résultats montrent que le GPU est significativement plus rapide que le CPU pour l'inférence. Cela indique que l'utilisation du GPU peut accélérer considérablement le temps d'inférence des modèles.

Dans le cas du GPU, AlexNet devient le modèle le plus vite, suivi de SqueezeNet, VGG et ResNet. On constate que MobileNet est très lent pour v2 et v3_3Small mais très efficace pour v3_small.

4.2 Temps d'inférence avec EdgeTPU

4.2.1 Implémentation

Nous avons choisi d'explorer le dispositif fourni pour notre projet, l'Accélérateur Coral USB, qui utilise EdgeTPU en combinaison avec TensorFlow pour effectuer la classification et la détection d'objets dans des images provenant du COCO Dataset.

Bien qu'il nous ait fallu du temps pour comprendre le processus et les configurations nécessaires, nous avons constaté des améliorations significatives en utilisant l'Accélérateur Coral avec EdgeTPU par rapport à l'utilisation d'un CPU ou GPU conventionnel.

Dans cette section, nous regardons le temps d'inférence pour la classification et nous nous concentrons ensuite sur le processus de détection d'objets dans les images, qui est utile pour le traitement des vidéos en continu.

Nous avons utilisé le seul modèle disponible pour le Coral : MobileNet v2 SSD pour effectuer les inférences. L'accelerator nous permet de classifier qu'une seule image à la fois, ainsi, afin de comparer avec les résultats précédents, nous avons mesuré le temps d'inférence pour la classification d'une seule image pour tous les dispositifs. Ensuite pour évaluer les performances d'autres procédures, nous avons regardé le temps d'inférence pour la détection d'objets pour une image et pour des batchs de différentes tailles, tout en comparant les résultats obtenus avec et sans l'utilisation de l'accélérateur USB.

4.2.2 Résultats

Classification d'une image

Afin de comparer les performances entre CPU/GPU, nous avons mesuré le temps d'inférence nécessaire pour classifier une image à l'aide du modèle MobileNet v2.

Modèle / Analyse d'une image	EdgeTPU
MobileNet v2	9.01 ms

Table 12: Temps d'inférence sur EdgeTPU pour la classification d'une image

Nous tirons de ce résultat que le USB accelerator Coral, améliore efficacement en termes de temps d'inférence comparée aux résultats de CPU (128.578 ms) et GPU (67.142 ms).

Détection d'objets

Ensuite, nous nous sommes davantage concentrés sur la détection d'objets, car cela nous permettrait de mieux comprendre le traitement d'une chaîne vidéo pour la détection d'objets.

Modèle / Analyse d'une image	EdgeTPU	CPU
MobileNet v2	80.42 ms	138.57 ms

Table 13: Temps d'inférence sur EdgeTPU pour la détection d'objets dans une image

On peut remarquer que le temps d'inférence pour la détection d'objets est plus long, car la tâche de détection d'objets est plus complexe.

Nous avons répété l'expérimentation pour différents nombres d'images, et on obtient le résultat ci-dessous :

Modèle / Nb d'images analysées	16	32	64	128
MobileNet v2 SSD (CPU + USB Accélérateur)	0.93 s	1.40 s	2.99 s	6.29 s
MobileNet v2 SSD (CPU)	2.01 s	4.19 s	8.34 s	16.6 s

Table 14: Temps d'inférence sur EdgeTPU pour la détection d'objets sur plusieurs images

Ces chiffres représentent le temps d'inférence nécessaire pour effectuer la détection d'objets du modèle utilisé en fonction du nombre d'images analysées simultanément.

On voit bien que l'accélérateur est efficace et améliore le temps d'exécution sur CPU. Plus le nombre d'images traités est grande, plus la différence est importante.

Pour s'assurer que le temps d'inférence ne dépend pas de **batch_size** nous avons réalisé des calculs de temps d'inférence sur 1000 exemples provenant du dataset COCO en variant la taille du batch.

Modèle / Batch size	8	16	32	64	128
MobileNet v2 SSD (EdgeTPU)	59.76 ms	42.94 ms	42.15 ms	43.16 ms	41.66 ms
MobileNet v2 SSD (CPU)	133.43 ms	124.8 ms	123.87 ms	124.72 ms	123.2 ms

Table 15: Temps d'inférence sur EdgeTPU pour la détection d'objets en fonction de batch size

Les résultats sous représentation graphique :

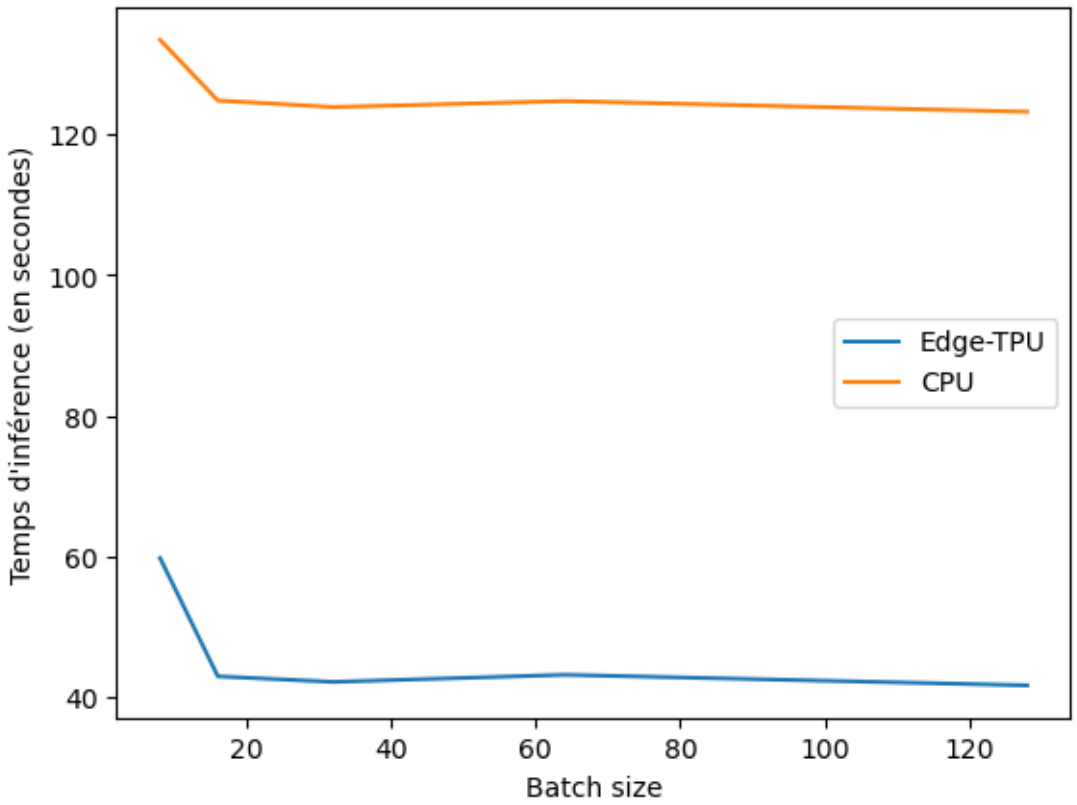

Figure 3: Temps d'inférence en fonction du batch_size sur EdgeTPU et CPU

On voit bien que les résultats coïncident avec ce qu'on avait trouvé précédemment : le temps d'inférence ne dépend pas du batch_size. Sur ce graphe, on peut aussi voir que l'EdgeTPU est plus rapide que le CPU.

Les résultats que nous avons obtenus démontrent l'avantage de l'EdgeTPU pour accélérer le processus d'inférence. L'utilisation de l'accélérateur USB réduit consid-

érablement le temps nécessaire pour effectuer les inférences par rapport à l'utilisation du CPU seul. Cela confirme que l'utilisation de la carte Coral avec l'EdgeTPU est une option extrêmement bénéfique pour les tâches de détection d'objets dans des ensembles de données tels que le COCO Dataset que nous avons utilisé.

5 Difficultés rencontrées

5.1 GPU

Les ordinateurs Macbook comprennent un petit GPU, mais qui n'est pas suffisant pour montrer les différences entre CPU et GPU. Nous avons eu recours aux ordinateurs proposées par le service PPTI qui contiennent un GPU RTX 2080.

Mais le problème de ces dispositifs est qu'ils ont des restrictions. D'abord au niveau des librairies mis en disposition, nous n'avons pas pytorch et puis nous n'avons droit qu'à 2GB de cache.

Ces problèmes ont été résolus en suivant les instructions de configuration de l'environnement : accéder et utiliser le répertoire `/tempory`.

5.2 Coral USB Accelerator

Lors de notre projet, nous avons rencontré plusieurs difficultés que nous avons dû surmonter. Parmi celles-ci, l'une des principales difficultés a été la prise en main et l'implémentation sur Coral.

La première difficulté a été de comprendre le fonctionnement et la configuration de Coral. Nous avons dû prendre le temps de nous familiariser avec ces dispositifs et de comprendre comment les intégrer dans notre projet.

De plus, l'adaptation de notre code existant pour utiliser les fonctionnalités spécifiques des cartes Coral a également été un défi. Nous avons dû mettre en place les bonnes bibliothèques et les dépendances nécessaires, ainsi que comprendre comment optimiser notre code pour tirer pleinement parti de l'accélération matérielle offerte par ces dispositifs.

De plus, l'Accélérateur USB nécessite des modèles au format .tflite qui doivent être compatibles avec EdgeTPU et quantifiés, c'est-à-dire réduits en taille. Bien que nous ayons réussi à convertir avec succès les modèles TensorFlow en TensorFlow Lite, nous avons rencontré des problèmes d'incompatibilité de taille avec les images que nous testions, ce qui nous a empêché de les utiliser.

5.3 Jetson Nano

De même, l'implémentation sur les cartes Jetson a présenté des défis similaires. Nous avons dû nous familiariser avec les spécificités de la carte, telles que l'installation du système d'exploitations Ubuntu 18.03, besoin du matériel extra (Carte micro-SD, USB adaptateur pour Wi-fi).

Le problème principale qui nous a bloqué pour toute manipulation de la carte et l'impossibilité de télécharger l'environnement sur la carte.

Pour ce problème, après les recherches faites, serait de mettre à jour le système d'exploitation à Ubuntu 20.04.

6 Conclusion

Malgré le fait que nous n'ayons pas atteint tous nos objectifs techniques, ce projet nous a offert une expérience de projet avec des outils d'apprentissage profond comme pytorch, une meilleure compréhension des systèmes embarqués et du traitement hard-ware/soft-ware. Obtenir des résultats significatifs nécessite une réflexion approfondie et une patience à chaque étape du processus.

Nos expérimentations au cours de ce projet ont confirmé que le traitement des données est bien plus complexe que la simple construction du modèle. Nous avons rencontré de nombreux essais et erreurs, car la différence entre différents modèles peut être important, de même que les différents hyper-paramètres possibles. Cependant, ce projet nous a permis de mieux comprendre une application intéressante de l'apprentissage automatique, tout en mettant en évidence plusieurs problématiques importantes à résoudre.

Nous avons également réalisé que la réussite d'un projet d'apprentissage automatique ne se limite pas à l'obtention de résultats précis, mais comprend également des aspects tels que la gestion des ressources matérielles, la compatibilité des dispositifs et la prise en compte des contraintes de performances en temps réel. Nous avons acquis une expérience précieuse dans la gestion de ces défis et sommes convaincus que cela nous sera utile dans nos futurs projets.

En conclusion, ce projet nous a permis de repousser nos limites et de développer nos compétences dans le domaine de l'apprentissage automatique appliqué aux systèmes embarqués. Nous sommes fiers des réalisations que nous avons accomplies et sommes impatients d'appliquer ces connaissances dans de futurs projets passionnants.

7 References

1 Jetson Nano Developer Kit. NVIDIA Developer. Retrieved March 22, 2023 from https://developer.nvidia.com/embedded/jetson-nano-developer-kit

2 Getting Started — PyTorch 2.0 documentation. Retrieved March 22, 2023 from https://pytorch.org/docs/stable/dynamo/get-started.html

3 TorchVision Object Detection Finetuning Tutorial — PyTorch Tutorials 2.0.0+cu117 documentation. Retrieved March 22, 2023 from https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html

4 Welcome to PyTorch Tutorials — PyTorch Tutorials 2.0.0+cu117 documentation. Retrieved March 22, 2023 from https://pytorch.org/tutorials/

5 Get started with the Dev Board Mini. Coral. Retrieved March 22, 2023 from https://coral.ai/docs/dev-board-mini/get-started/