وزارة التعليم العالي و البحث العلمي

BADJI MOKHTAR- ANNABA UNIVERSITY

UNIVERSITE BADJI MOKHTAR ANNABA

جامعة باجي مختار-عنابة

Année : 2021

Faculté : Sciences de l'Ingéniorat
Département : Electronique

MEMOIRE
Présenté en vue de l'obtention du diplôme de : MASTER

Intitulé :

Classification et détection d'objet par les méthodes de Transfert Learning et TinyML

Domaine : Sciences et Technologie

Filière : Télécommunication

Spécialité : Système et Réseaux de télécommunication

Présenté par :
RAVELOMANANTSOA Vonimanitra Sarobidy
YAHYAOUI Abdenour

DEVANT LE JURY

Présidente : A. BOULMAIZ	MCB	UBM Annaba
Superviseur : NASRI Seif Allah El Mesloul	MCB	UBM Annaba
Co-Superviseur : Djamel Eddine Benrachou	MCB	QUT Australie
Examinateur : D. MESSADEG	Prof	UBM Annaba

Abstract

Object detection and image classification technologies stand as important research areas in the image processing and analysis field. Object detection and classification techniques are used in various sensitive applications such as medical diagnoses, human-machine interaction, smart agriculture, surveillance, and security. The emergence of artificial intelligence has increased the efficiency of object detection and classification systems. Our research project consists of deploying cutting-edges Artificial Intelligence techniques in object detection and classification area. Two machine learning techniques, TinyML and transfer learning, were used and explored throughout this research project. TinyML which is a form of reduced ML dedicated for embedded system, has been used in our dissertation to classify tomato disease based on their leaves. Transfer learning techniques has been also applied for the same application. Due to the current COVID-19 situation, we have considered a second application which detects if the person is wearing a mask or not using machine learning technology. Final results showed a considerable accuracy in both applications.

Key words:

Object detection, image classification, Machine learning, TinyML, Transfer learning.

Résumé

Les technologies de détection d'objet et de classification d'image constituent des domaines de recherche importants dans l'analyse et le traitement d'image. Elles sont utilisées dans diverses applications sensibles telles que les diagnostics médicaux, l'interaction homme-machine, les industries agricoles, la surveillance et la sécurité. L'émergence de l'intelligence artificielle a augmenté l'efficacité des systèmes de détection et de classification d'image. Notre projet de recherche consiste à déployer des techniques d'Intelligence Artificielle de pointe dans le domaine de détection d'objets et de classification d'image. Deux techniques d'apprentissage automatique qui sont notamment TinyML et l'apprentissage par transfert ont été utilisés et explorés tout au long de ce projet de recherche. TinyML, est une forme de ML réduite dédiée au système embarqué, on l'a utilisé dans notre thèse pour classer les maladies de la tomate en fonction de leurs feuilles. Des techniques d'apprentissage par transfert ont également été appliquées pour la même application. En raison de la situation actuelle du COVID-19, nous avons mis en œuvre une deuxième application qui permet de détecter le port de masque sur les personnes. Les résultats finaux ont montré une précision considérable dans les deux applications.

Mots Clés :

Détection d'objet, classification d'image, apprentissage automatique, TinyML, apprentissage par transfert

ملخص

تقف تقنيات الكشف عن الأجسام وتصنيف الصور كمجالات بحثية مهمة في مجال معالجة الصور وتحليلها. وتستخدم هذه التقنيات في مختلف التطبيقات الحساسة مثل التشخيص الطبي، والتفاعل بين الإنسان والآلة، الزراعة الذكية، المراقبة والأمن. أدى ظهور الذكاء الاصطناعي إلى زيادة كفاءة أنظمة الكشف عن الأجسام وتصنيفها. يتكون مشروعنا البحثي من نشر أحدث تقنيات الذكاء الإصطناعي في مجال الكشف عن الأجسام وتصنيفها، حيث تم استخدام تقنيتين للتعلم الآلي، TinyMLوالتعليم المتنقل، واستكشافهما في جميع أنحاء هذا المشروع البحثي .TinyML وهو شكل من أشكال ML مخفض مخصص للنظام المضمن، تم استخدامه في أطروحتنا لتصنيف أمراض الطماطم بناءً على أوراقها. كما تم تطبيق تقنيات التعلم المتنقل لنفس التطبيق. بسبب الوضع الحالي لـ COVID-19، لقد درسنا تطبيقًا ثانيًا يكتشف ما إذا كان الشخص يرتدي قناعًا أو لا، يستخدم تكنولوجية التعلم الآلي. أظهرت النتائج النهائية دقة كبيرة في كلا التطبيقين.

الكلمات الرئيسية:

الكشف عن الأشياء، تصنيف الصور، التعلم الآلي، TinyML، التعليم المتنقل.

REMERCIEMMENT

En préambule à ce mémoire nous remercions DIEU, LE TOUT PUISSANT qui nous a aidé et donné la persévérance et la patience durant ces longues années d'étude.

Nous tenons à remercier sincèrement nos Encadreurs Seif Allah Elmesloul Nasri et Djamel Eddine Benrachou, qui se sont toujours montrés à l'écoute et très disponible tout au long de la réalisation de ce mémoire, ainsi pour l'inspiration, l'aide et le temps qu'ils ont bien voulu nous consacrer et sans ces contributions ce mémoire n'aurait jamais vu le jour.

Nous tenons à remercier également tous les membres du jury, leurs précieux conseils et remarques ont contribué à l'amélioration de la qualité scientifique de ce manuscrit.

Nous souhaitons adresser nos vifs remerciements les plus sincères aux personnes qui nous ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire ainsi qu'à la réussite de ce formidable semestre universitaire. Ces remerciements vont tout d'abord au corps des enseignants et administratifs de la Faculté des Sciences de la Technologie de l'Université Badji Mokhtar Annaba, et plus précisément le département d'électronique pour la richesse et la qualité de leur enseignement et qui déploie de grands efforts pour assurer à leurs étudiants une formation actualisée et de qualité.

On n'oublie pas nos familles pour leurs contributions, leurs soutiens et leurs patiences. Enfin, nous adressons nos plus sincères remerciements à tous nos proches et amis, qui nous ont toujours encouragés au cours de la réalisation de ce mémoire.

Merci à toutes et à tous.

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS

CNN :	Convolutional Neural Networks / réseau de neurones convolution
COCO :	Common Objects in Context
DSP :	Digital Signal Processor/ Processeur de digital numérique
Flops (Tflops) :	floating-point operations per second (Teraflops), le nombre d'opération en virgule flottante par seconde
FFT :	Fast Fourier Transform / Transformation de Fourier rapide
FPS :	Frame Peer Second, nombre d'image par seconde
GPU :	Graphique Processing Unit / Processeur graphique
Hz (GHz) :	Hertz (GigaHertz) unité de la fréquence
IA :	Intelligence artificielle
ILSVRC :	ImageNet Large Scale Visual Recognition Challenge
MCU :	MicroController Unit / Microcontrôleur
MFCCs :	Mel-Frequency cepstral coefficients
ML :	Machine Learning /Apprentissage automatique
Ms :	millisecond, unité de temps
N.B :	Nota Bene
Octet (Go/Mo/Ko) :	Unité de mesure de la quantité e données informatique (1octet= 8bit) (Giga-octet/Méga-octet/KiloOctet)

RAM :	Random Access Memory/ mémoire vive
ROM :	Read Only Mémoire/ mémoire morte
SDK :	Software Developpement Kit
RoI :	Region of interest / Région d'intérêt
SSD :	Single shot detector
SVM :	Support Vector Machine /Machine à vecteur de support ou séparateur à vaste marge
R-CNN :	Region based Convolutional Neural Networks / Région basé sur réseau de neurones convolution
TinyML :	Tiny Machine Learning / Apprentissage automatique des machines de petite taille
TPU :	Tensor Processing Unit
W/mW :	Watt (Milliwatt), unité internatonale de puissance ou de flux énergétique
YOLO:	You Look Only Once

SOMMAIRE

Introduction générale

Le développement technologique de l'Intelligence Artificiel (IA) a connu des évolutions grâce au support de Machine Learning. Actuellement, il ouvre un large domaine d'application en apportant son développement dans plusieurs systèmes d'expertises restreints à des domaines spécifiques comme l'analyse en chimie, traitement des données, le diagnostic médical, les systèmes experts de l'industrie et le domaine militaire [1]. Cette évolution est liée à l'arrivée du Big Data, aujourd'hui cinq quintillions d'octets de données sont produits chaque jour par les appareils mais seulement 1% sont utilisées. L'exploitation de cette grande variété de donnée dans la technologie de Machine Learning a besoin de matériel de forte puissance [3]. En dépit de ces matériels, les contraintes de puissance entrent en jeu ce qui rendent les appareils à nos jours indépendants des sources énergétique, d'où le manque de l'ergonomie, flexibilité avec les prix élevés des composants [1] [7]. Récemment une nouvelle technologie appelée TinyML contourne cette difficulté et se concentre sur les caractéristiques clés afin d'adapter l'apprentissage automatique sur des appareils de puissance limitée qui nécessitent qu'une faible alimentation énergétique [4]. L'objectif de TinyML est de faire de ces produits un véritable moyen nécessaire pour transformer les données brutes des capteurs en informations exploitables localement, sur l'appareil lui-même, car les coûts énergétiques du flux de transmission sont avérés trop élevés pour être misent en pratique. Il en résulte un produit suffisamment petit pour s'intégrer dans n'importe quel environnement et capable de fonctionner pendant une durée utile sans aucune intervention humaine [1].

La détection d'objet et la classification d'image sont deux applications distinctes de l'apprentissage automatique utilisées dans le domaine de vision par ordinateur. Pour mettre en œuvre ces applications, on doit créer un modèle de réseau de neurones mais l'entrainement de bout-à-bout pour la création de ce dernier peut prendre des jours, voire même des semaines à s'entraîner sur de grands ensembles de données. Un moyen de raccourcir ce processus est la technique de transfert Learning. Notre projet de fin d'étude s'inspire du concept de TinyML en l'appliquant sur la classification et la détection d'objet. Afin de réaliser ces deux applications, on utilisera un ensemble de donnée pour ré-entraîner un réseau de neurones par la technique de transfert Learning puis le convertir afin de l'adapter à des systèmes embarqués de puissance limitée.

Ce projet est divisé en trois parties. Dans le premier chapitre nous allons donner quelques généralités sur l'apprentissage automatique en évoquant la classification et la détection d'objet et puis l'émergence de TinyML. Pour le deuxième chapitre nous allons présenter la méthodologie du travail à partir de la collecte des données jusqu'à l'extraction et

évaluation du modèle. Et en dernier chapitre, nous allons entamer le processus de réalisation de la classification et la détection d'objet. Après la présentation générale du projet, nous terminerons ce manuscrit par une conclusion générale.

Chapitre I
Machine Learning
et TinyML

1 Introduction

La grande majorité des processeurs dans le monde sont en fait des unités de microcontrôleur (MCU), qui sont largement utilisés pour effectuer des tâches de contrôle simples dans les applications allant des automobiles aux appareils médicaux et aux équipements bureautiques. Cependant, ces matériels pauvres en ressources sont limités considérablement par la complexité des modèles de Machines Learning mais qui peuvent être déployés de façon affinée grâce à TinyML [7]. Les progrès récents du matériel Machine Learning à très faible consommation d'énergie (TinyML) promettent de débloquer une nouvelle classe d'application intelligente [4]. Dans ce premier chapitre nous allons voir les applications de machine Learning et puis l'émergence de TinyML.

2 Machine Learning

2.1 Définition de L'IA, Machine Learning et le Deep Learning

La Machine Learning et le Deep Learning sont apparentés. Ce sont deux systèmes d'apprentissage basés sur la technologie de l'intelligence artificielle (IA) mais construits sur différentes couches d'abstractions. L'intelligence artificiel comme son nom l'indique est définie comme un programme qui présente des capacités cognitives similaires à celles d'un être humain. L'un des principales souches de l'intelligence artificielle est de faire en sorte que les ordinateurs pensent comme des humains et résolvent des problèmes quotidiens comme nous le faisons. Les Machines Learning ou l'apprentissage automatique est un sous domaine de l'IA [8]. Il peut avoir une définition légèrement différente, en fonction du contexte. On peut donner une définition idéale « L'apprentissage automatique est la science qui consiste à amener les ordinateurs à apprendre et à agir comme le font les humains, et à améliorer leur apprentissage au fil du temps de manière autonome, en leur fournissant des données et des informations sous forme d'observations et d'interactions dans le monde réel. » [6] En effet les programmations traditionnelles fixent une ou plusieurs règles pour obtenir des réponses de sorties à partir d'une entrée mais avec l'apprentissage automatique, le système en déduira une règle générale pour toutes sortes de données [1].

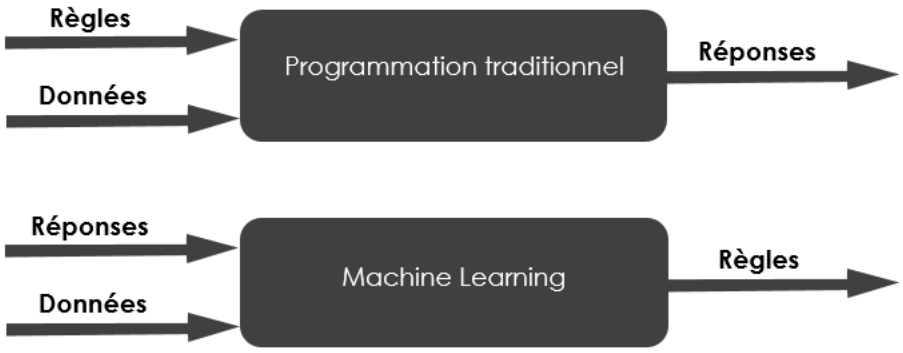


Figure 1: La différence entre les programmes traditionnel et les machine Learning [14]

D'un autre point de vue, l'apprentissage automatique (ML) est défini comme l'étude du programme informatique qui tire la partie d'algorithme pour apprendre par inférence et modélise sans être explicitement programmé [5]. Il existe 3 type de machine Learning :

- Apprentissage supervisé
- Apprentissage non supervisé
- Apprentissage par renforcement

Le Deep Learning ou l'apprentissage en profondeur est un sous-ensemble de l'apprentissage automatique qui dispose des réseaux capables d'apprendre sans surveillance à partir des données non structurées ou non étiquetées. Le Deep Learning est une version plus spécialisée de l'apprentissage automatique caractérisé par sa capacité à effectuer un apprentissage non supervisé. Les algorithmes d'apprentissage en profondeur qui imitent le fonctionnement du cerveau humain sont connus sous le nom de réseau de neurones [4] [8].

2.2 La vision par ordinateur

La « Computer Vision » est une branche de l'Intelligence Artificielle qui a pour objectif de permettre aux ordinateurs d'interpréter des données visuelles (photos ou vidéo) afin d'en extraire des informations. La Computer Vision est constituée de plusieurs branches, dont les deux principales sont la reconnaissance d'image et la Machine Vision. Ces tâches sont des algorithmes dits "d'apprentissage supervisé" : ils sont "entraînés" à partir d'exemples d'images annotées par les concepts qu'il faudrait pouvoir reconnaître, pour ensuite être capables de prédire les zones d'intérêt dans les images que l'algorithme n'a encore jamais rencontrés [13].

2.2.1 La classification d'image

La classification d'image consiste à reconnaître à quelle catégorie appartient une image parmi un ensemble de catégories prédéterminées. Toutes les données auront obligatoirement une seule classe. La classification d'images est une tâche fondamentale qui tente de comprendre une image entière dans son ensemble. Le but est de classer l'image en lui attribuant une étiquette spécifique. Typiquement, la classification d'images fait référence à des images dans lesquelles un seul objet apparaît et est analysé. En revanche, la détection d'objets implique à la fois des tâches de classification et de localisation, et est utilisée pour analyser des cas plus réalistes dans lesquels plusieurs objets peuvent exister dans une image [13].

2.2.2 La détection d'objet

La détection est une technique de reconnaissance d'image utilisée afin de localiser un ou plusieurs objets d'intérêt dans une image ou une vidéo. Elle permet de délimiter les objets reconnus par un rectangle sur l'image (appelé bounding box en anglais, une "boîte englobante"). On distingue trois tâches spécifiques pour la détection d'objet [11].

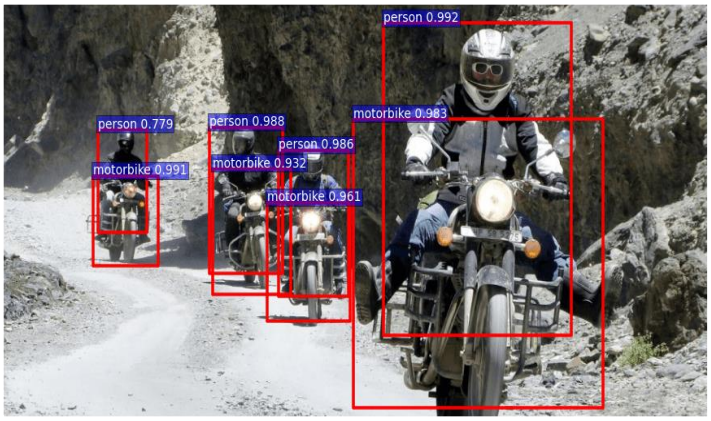


Figure 2: Un exemple de reconnaissance d'objet dans une image [11]

- Classification d'image : prédiction du type ou la classe d'un objet dans une image.
- Localisation d'objets : localisez la présence d'objets dans une image et indiquez leur emplacement avec un cadre de sélection.
- Détection d'objets : localisez la présence d'objets avec un cadre de délimitation et des types ou classes d'objets localisés dans une image.

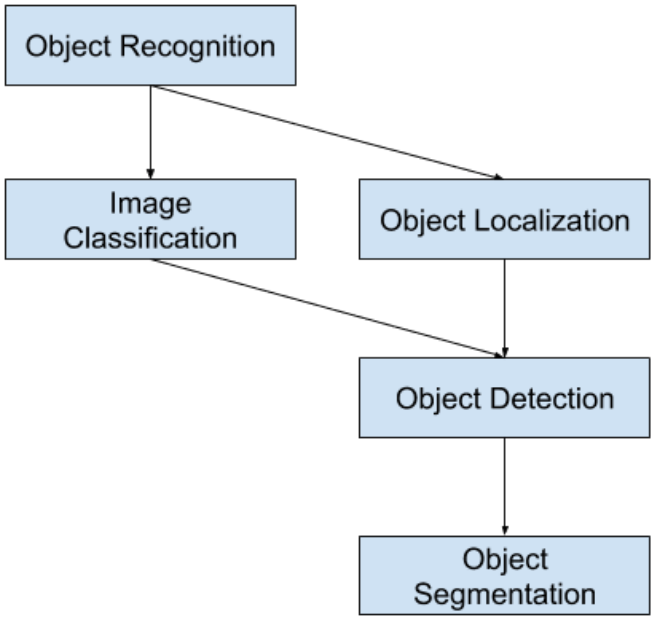


Figure 3: Présentation des tâches de vision par ordinateur sur la reconnaissance d'objets [11]

2.3 Les réseaux de neurones et le transfert Learning

2.3.1 Les réseaux de neurones

Les réseaux de neurones sont un ensemble d'algorithmes qui sont modélisés de manière approximative sur le cerveau humain. Ils interprètent les données sensorielles à travers une sorte de perception de la machine, d'étiquetage ou de regroupement des intrants bruts. Les modèles qu'ils reconnaissent sont numériques, contenus dans des vecteurs, dans lesquels toutes les données du monde réel, qu'il s'agisse d'images, de sons, de textes ou de séries temporelles [8]. Les réseaux de neurones sont caractérisés de plusieurs nœuds qui sont liés avec les nœuds des couches précédentes et suivantes. Ces nœuds ont des fonctions précises par couche pour extraire des informations. Un réseau de neurones profond est composé d'une couche d'entrée, une couche de sortie à l'extrémité et la troisième couche intermédiaire appelée couche cachée. Le réseau de neurones convolutifs (CNN) est une approche d'apprentissage en profondeur largement utilisée pour résoudre des problèmes vision par ordinateur complexe. Il surmonte les limites des approches traditionnelles d'apprentissage automatique [9] [10].

2.3.2 La modélisation par Transfert Learning

Les modèles de réseaux de neurones à convolution sont utilisés pour notre projet car elles sont efficaces pour aider une machine à comprendre le contenu d'une image. La modélisation par la technique bout-à-bout « end-to-end » peuvent prendre des jours, voire des semaines, à s'entraîner sur de très grands ensembles de données. Un moyen de raccourcir ce processus est transfert Learning. Il se base sur l'utilisation des modèles pré-entraînés comme point de départ, le Transfer Learning permet de développer rapidement des modèles performants et résoudre efficacement des problèmes complexes en « Computer Vision ». Il repose sur la réutilisation des connaissances acquises à partir d'un modèle pré-entrainé (sources) pour la modélisation d'un autre (cible).

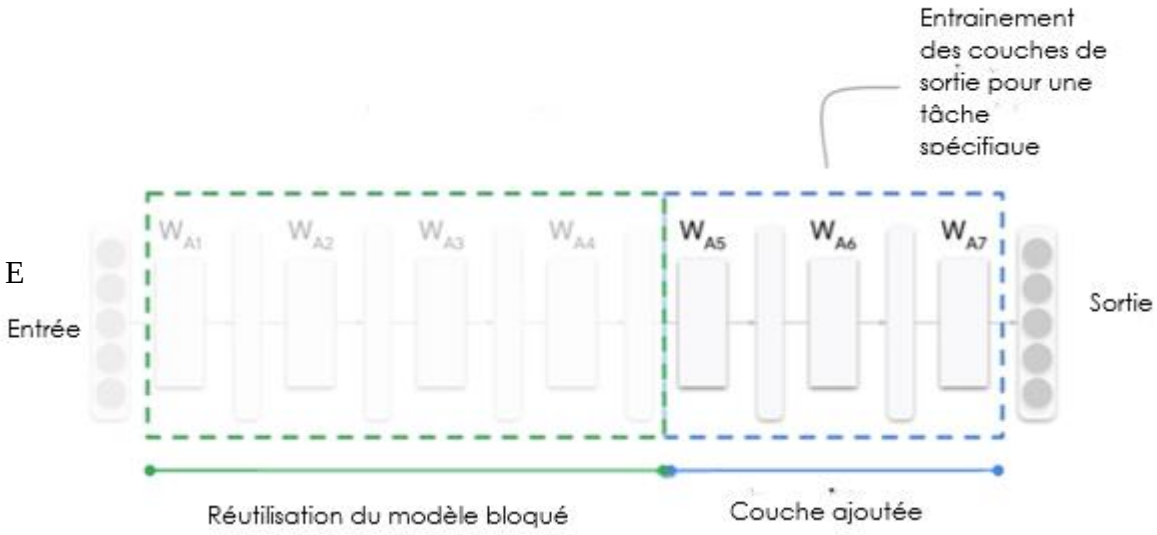


Figure 4: Exemple simplifié de l'entrainement par transfert Learning [14]

L'utilisation de cette technique est valable s'il y a une corrélation entre l'ensemble des données utilisées et des données d'origine afin de spécialiser le réseau de neurones à créer.

Le tableau suivant nous montre les règles générales pour l'utilisation du transfert Learning :

Tableau 1: Règle générale pour l'utilisation de la technique transfert Learning [14]

	Le nouvel ensemble de données est similaire à l'ensemble de données d'origine	Le nouvel ensemble de données est (moins) similaire à l'ensemble de données d'origine
Le nouvel ensemble de données est petit	Meilleur scénario pour l'apprentissage par transfert	Pas le meilleur scénario pour l'apprentissage par transfert
Le nouvel ensemble de données est volumineux	L'apprentissage par transfert fonctionnera	La formation à partir de zéro peut donner une meilleure précision

2.4 Limites des algorithmes de l'apprentissage automatique

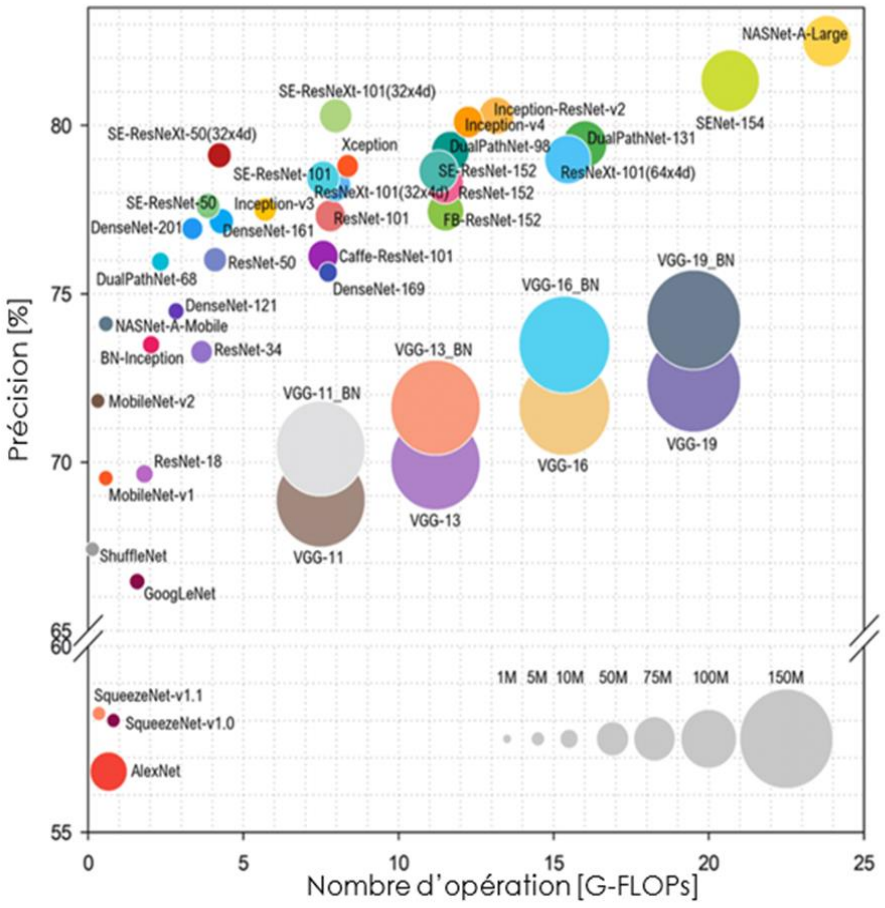


Figure 5: Variation des paramètres des modèles de classifications [14]

Toutes les architectures des réseaux de neurones n'utilisent pas leurs paramètres avec le même niveau d'efficacité et ils sont capables d'avoir une meilleure performance avec des inférences en temps réel sur un GPU puissant avec une empreinte mémoire minimale d'environ 0,6 Go. De ce fait, la tendance du déploiement de ces architectures se limite au niveau des grandes machines. Ils présentent ainsi des barrières pour apporter l'inférence ML aux appareils à très faible consommation [4]. Jusqu'à présent, la ML s'est principalement concentré sur l'inférence qui a conduit à de nombreuses avancées dans des modèles d'apprentissage automatique telles que l'exploitation de l'élagage, la Clustering, et quantification. Mais ces dernières années, cette contrainte est contournable grâce à l'émergence de la technique TinyML. L'objectif est d'apporter l'inférence ML aux appareils à très faible consommation permettant une plus grande réactivité et confidentialité tout en évitant le coût énergétique associé à la transmission [1] [4].

3 TinyML

3.1 Définition et raison de TinyML

Nous vivons actuellement dans un monde entouré de modèles d'apprentissage automatique. Les tâches quotidiennes comme le défilement sur les médias sociaux avec la réception de suggestion de pubs qui font illusions d'une coïncidence, prendre une photo, vérifier la météo, dépendent tous de modèles d'apprentissage automatique. Nous savons tous que la formation de ces modèles est informatiquement chère. Mais la plupart du temps, l'exécution de l'inférence sur ces modèles est également coûteuse en calcul. La vitesse à laquelle nous utilisons les services d'apprentissage automatique, nous avons besoin des systèmes informatiques suffisamment rapides pour les gérer. Ainsi, la plupart de ces modèles fonctionnent sur d'énormes centres de données avec des grappes de processeurs et de GPU (des TPU dans certains cas) [1]. TinyML est un domaine d'étude qui combine l'apprentissage automatique (ML) et les systèmes embarqués (SE) basés sur des petits appareils de puissance limitée tels que les microcontrôleurs. Il permet une inférence de modèle à faible latence, faible consommation et faible bande passante au niveau des périphériques. Alors qu'un processeur grand public standard consomme entre 65 watts et 85 watts et qu'un GPU grand public standard consomme entre 200W et 500W. Un microcontrôleur typique consomme de l'énergie de l'ordre de milliwatts ou microwatts ce qui est environ à l'ordre de mille fois moins de consommation d'énergie. Cette faible consommation d'énergie permet aux appareils TinyML de fonctionner sans alimentation sur batteries pendant des semaines, des mois et dans certains cas voire même des années, tout en exécutant des applications ML au périphérique [1].

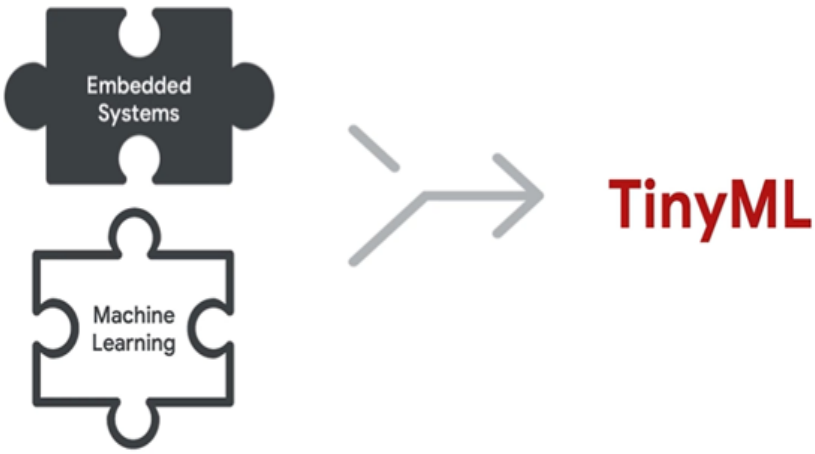


Figure 6: Schémas d'un système TinyML [1]

3.2 Challenge et les atouts de la technique TinyML

3.2.1 Les défis de TinyML

Libérer le potentiel du machine Learning dans les appareils embarqués nécessite à surmonter des défis cruciaux. Quand les ingénieurs ont déployé des réseaux de neurones à ces systèmes, ils ont construit des cadres uniques qui nécessitent une optimisation manuelle pour chaque plate-forme matérielle. Ces cadres ont eu tendance à être étroitement ciblés, manquant de fonctionnalités pour prendre en charge plusieurs applications et manquant flexibilité sur une large gamme de matériel. Le développement dans ses systèmes a donc été pénible, nécessitant une optimisation en finesse des modèles à exécuter sur un appareil spécifique. Et la rectification de ces modèles pour les déployer et fonctionner sur un autre appareil nécessitait un effort manuel et d'optimisation répétée. Dans un autre point de vue, l'effet de cette situation est le rythme de travail lent et le coût élevé de formation et de déploiement du modèle sur des matériels embarqués empêche les développeurs de justifier facilement l'investissement requis pour créer des nouvelles fonctionnalités. Nous pouvons distiller les défis généraux confrontés dans les listes suivantes [1] [4] :

- Incapacité de déployer facilement et inflexibilité des modèles sur plusieurs architectures matérielles embarquées.
- Manque d'optimisations qui tirent parti du sous-jacent matériel sans nécessiter de développeurs de Framework pour faire des efforts spécifiques à la plateforme.
- Manque d'outils de productivité reliant la formation pipeline vers les plates-formes et les outils de déploiement.
- Infrastructure incomplète pour la compression, la quantification, l'appel et l'exécution du modèle.
- Manque de tests dans les applications dans le monde réel.

Pour résoudre ces problèmes, nous présentons TensorFlow Lite. En effet, il est facile de faire fonctionner les applications TinyML sur plusieurs architectures car il permet aux fournisseurs de matériel d'optimiser les noyaux pour leurs appareils. Il offre les avantages suivants :

- L'approche se base sur la mobilité, la flexibilité, et la facilité pour l'adaptation aux fonctionnalités des nouvelles applications.

- Minimiser l'utilisation des dépendances externes et les exigences de la bibliothèque indépendante du matériel.
- Un cadre d'architecture du modèle ouvert à un large écosystème d'apprentissage automatique grâce à TensorFlow Lite qui possède une infrastructure de conversion et d'optimisation du modèle.

3.2.2 Les Framework pour surmonter les contraintes

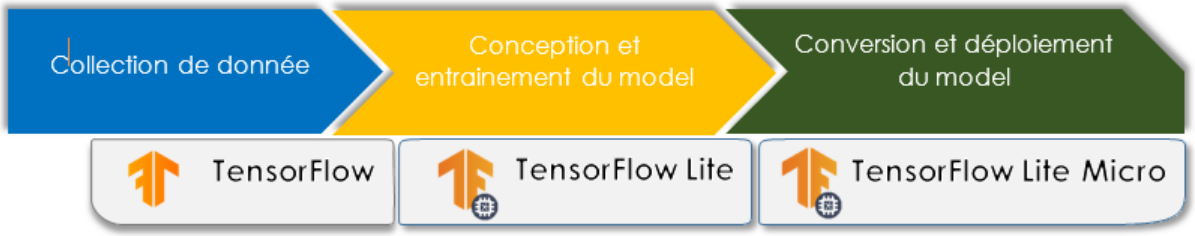


Figure 7: Schémas simplifié du flux de travail [14]

Dans le flux du travail pour la modélisation et le déploiement que nous verrons dans le prochain chapitre, nous allons utiliser à la fois TensorFlow et TensorFlow Lite. TensorFlow est une plate-forme Open Source de bout en bout dédiée à la machine Learning. Elle propose un écosystème complet et flexible avec une bibliothèque et ressources communautaires permettant aux chercheurs d'avancer dans le domaine du machine Learning, et aux développeurs de créer et de déployer facilement des applications qui exploitent cette technologie. Cette bibliothèque permet notamment d'entraîner et d'exécuter des réseaux de neurones pour la classification, la reconnaissance d'image et les réseaux de neurones récurrents. En ordre général, TensorFlow a été conçu à l'origine pour les gros systèmes informatiques et c'est la raison pour laquelle TensorFlow Lite a été développé pour former des modèles spécifiques dans le contexte de TinyML sur les appareils de faible puissance. TensorFlow Lite passe par une série d'étapes pour adopter ce grand modèle et le convertir en un format qui est beaucoup plus petit, beaucoup plus compact et bien plus adapté aux appareils terminaux. Le tableau suivant montre quelques différences majeures entre les trois Framework supplémentaire [15] :

Tableau 2 :Différence entre les framework de TensorFlow[14]

	TensorFlow	TensorFlow Lite	TensorFlow Lite Micro
Entrainement	Oui	No	No
Nombre d'opération	~1400	~130	~50
Support et outils de quantification native	No	Oui	Oui
Taille de base binaire	3MB+	100KB	~10KB
mémoire de base	~5MB	300KB	20KB
architectures optimisées	X86, TPUs, GPUs	Arm Cortex A, X86	Arm Cortex M, DSPs, MCUs
Besoin de système d'exploitation	No	Oui	Oui

3.2.3 Les avantages de TinyML

En utilisant le Framework TensorFlow, On va pouvoir exécuter le modèle sur les appareils, ce qui réduit la latence de la sortie et la bande passante car les données n'ont pas besoin d'être envoyées au serveur en permanence pour exécuter l'inférence. Les données seront traitées et stockées sur les périphériques alors on aura plus de sécurité et de confidentialité. Comme nous l'avons vu précédemment, les microcontrôleurs consomment très peu d'énergie. Cela leur permet de fonctionner sans être alimentés en permanence. On peut alors en déduire les atouts de la technique TinyML sur les points suivants [1] :

- Faible latence
- Bande passante faible

- Faible consommation énergétique
- Confidentialité de donnée.

3.3 Innovation et l'avenir de TinyML

Aujourd'hui, il y a plus de 250 milliards d'appareils embarqués actifs dans le monde, avec une croissance annuelle attendue de 20 % [16]. Sur ces appareils, environ 3 milliards actuellement en production sont capables de prendre en charge TensorFlow Lite. Rendre TinyML plus accessible aux développeurs sera crucial pour permettre la prolifération massive de l'apprentissage automatique afin de transformer les données gaspillées en informations exploitables et de créer de nouvelles applications dans de nombreux secteurs. TinyML sera bientôt partout, alimentant la prochaine génération d'appareils embarqués intelligents. Ces appareils seront dans nos maisons et dans des endroits très éloignés, permettant une surveillance à distance à la fois pour l'industrie et l'écologie. Aujourd'hui, dans ces paramètres de surveillance à distance, 99% des données brutes des capteurs sont non utilisées, ce qui constitue une mine de données pour l'apprentissage automatique [1]. Dans les prochaines sections, nous examinons quelques domaines d'application émergents qui ont un grand potentiel pour TinyML.

3.3.1 Maintenance prédictive industrielle

Dans les industries 4.0, TinyML est déjà utilisé pour fournir une détection plus intelligente qui permet une surveillance avancée améliorant la productivité et la sécurité. Les machines sont sujettes aux pannes. En utilisant TinyML sur des appareils de faible puissance, il est possible de surveiller la machine et de prévoir les défauts à l'avance en permanence. Cette maintenance prédictive peut conduire à des économies de coûts significatives. Ping Services, une start-up australienne a introduit un appareil IoT qui surveille de manière autonome les éoliennes en se fixant magnétiquement à l'extérieur de l'éolienne et en analysant des données détaillées à la périphérie. Cet appareil peut alerter les autorités des problèmes potentiels avant même qu'ils ne surviennent [17].

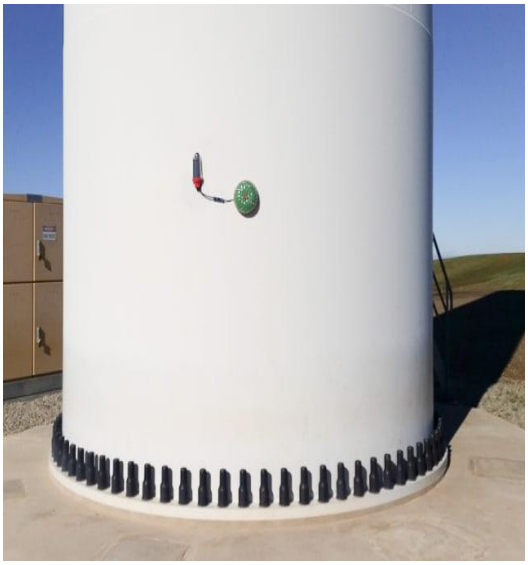


Figure 8: Moniteur Ping monté magnétiquement sur une tour de turbine [17].

3.3.2 Domaine sanitaire

Le projet Solar Scare Mosquito déploie des petites plates-formes robotiques intelligentes pour l'Internet des objets (IoT) afin de freiner la propagation des épidémies transmises par les moustiques telles que le paludisme, la dengue, le virus Zika et le chikungunya... Il détecte les conditions de reproduction des moustiques et agite l'eau pour empêcher leurs reproductions. Il fonctionne à l'énergie solaire et peut donc fonctionner indéfiniment [18].

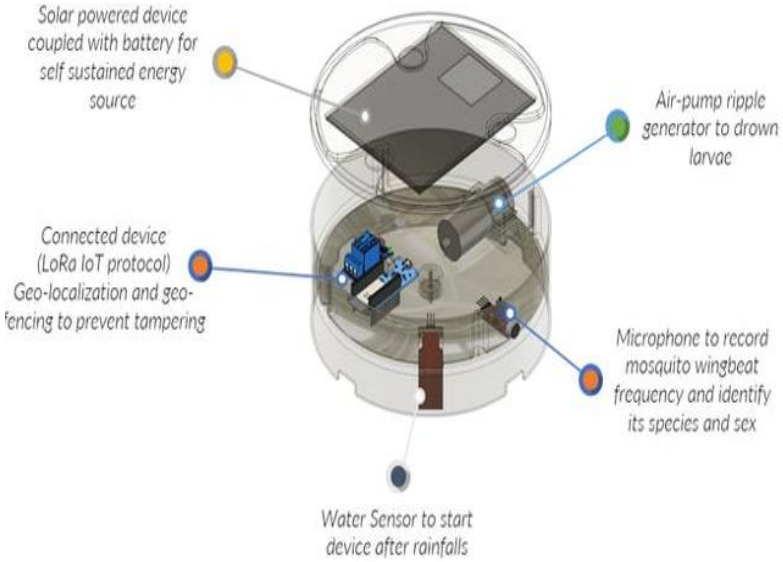


Figure 9: Solar Scare Mosquito [18]

3.3.3 Conservation de la faune

Le projet ElephantEdge a été créé récemment suite à une crise croissante en Afrique sur la diminution de la population d'éléphants qui peut être anéantie dans 10 ans. Ils utilisent des colliers intelligents pour localisation afin de prévenir les gardes par rapport à une zone de risques ou contre les braconniers. Des chercheurs du Laboratoire de bioacoustique appliquée de l'Université polytechnique de Catalogne ont conçu un système intelligent de capteurs acoustiques et thermiques utilisant des modèles d'apprentissage automatique personnalisés fonctionnant à l'énergie solaire comme système d'alerte précoce (voir l'image ci-dessous - ensemble tout-en-un avec la source d'énergie permet la proximité du chemin de fer sans infrastructure supplémentaire, par exemple les lignes électriques).

Figure 10: Test d'un capteur acoustique à proximité d'une voie ferrée [19]

4 Conclusion

Dans ce chapitre, on a vu quelques généralités sur l'intelligence artificielle en se focalisant dans les domaines de « vision computer » issue de l'apprentissage automatique (Machine Learning). La formation d'un réseau de neurones de bout-à-bout nécessite énormément de temps mais avec la technique de transfert de connaissance (transfert Learning) on peut réduire ce processus. Les architectures de l'apprentissage automatique touchent sa limite car la performance des réseaux de neurones demande une alimentation de machine puissante et ne sont pas compatibles aux systèmes embarqués ayant des ressources limitées.

Alors l'émergence récemment d'une nouvelle technique appelée TinyML a contourné cette contrainte en adaptant les algorithmes de ML dans les matérielles de faible puissance. Cette exploitation a connu une réussite grâce au Framework TensorFlow qui permet l'adaptation de ses réseaux de neurones sur les machines de puissance faible pour éviter les compromis entre les différentes parties intégrantes (machine Learning algorithmes, matériel, logiciel) des systèmes embarqués à ressources limitées. Cette technique TinyML est utilisé actuellement dans différents domaines et offre encore un large éventail d'applications.

Chapitre II

Méthodologie du travail

1 Introduction

Dans cette section, nous allons développer l'apprentissage d'un réseau de neurones en explorant le processus de travail utilisé dans le prochain chapitre. La formation des réseaux de neurones met en jeu plusieurs domaines d'étude. En commençant par l'ingénierie des données vue qu'on doit les traiter, manipuler afin de les adapter à la structure de notre model. Ensuite l'ingénierie de model qui va permettre de perfectionner notre réseau de neurones en réduisant notre modèle pour augmenter la vitesse de calcul et d'économiser les mémoires de notre système. Ensuite on pourra adopter notre réseau de neurones sur la machine de faible puissance [14]. Le flux du travail se décompose en trois étapes principales :

- Étape 1 : collecte et traitement des données
- Étape 2 : conception et entrainement du modèle
- Étape 3 : conversion et évaluation.

2 Données

Toutes les tâches de l'apprentissage automatique commencent par des ensembles de données. La croissance des médias sociaux a inauguré une ère de partage de données en ligne, dans laquelle la majorité des interactions quotidiennes sont menées entre des intermédiaires technologiques. Bien que cela offre d'énormes avantages en termes de productivité, de communication et d'accessibilité, cela présente également des défis importants en matière de confidentialité. L'ingénierie des données est une composante essentielle de l'apprentissage automatique et consiste à définir les exigences, à enregistrer les données, à les traiter et à améliorer l'ensemble des données. La qualité et la quantité des données collectées doivent inclure suffisamment des caractéristiques saillantes pour atteindre l'objectif car on n'aura peut-être pas assez d'informations pour apprendre le contenu de l'image avec peu de données, et trop d'échantillonnages mal collectés pourront sur-spécialiser notre réseau et pourront être très lent à apprendre. Nous utiliserons des ensembles de données existantes comme exemples pour fonder les principes, en particulier autour de l'identification des besoins des ensembles de données. On sépare ensuite nos données en trois parties :

- Notre premier sous-ensemble (rectangle jaune) est constitué des données avec lesquelles nous allons nous entraîner.
- Notre deuxième sous-ensemble (rectangle rouge) est constitué des données que nous utiliserons pour la rectification de notre réseau lors de la « Back Propagation ».
- Notre troisième et dernier sous-ensemble (rectangle bleu) est utilisé une fois qu'on a fini l'entrainement pour tester et valider la capacité de notre réseau.

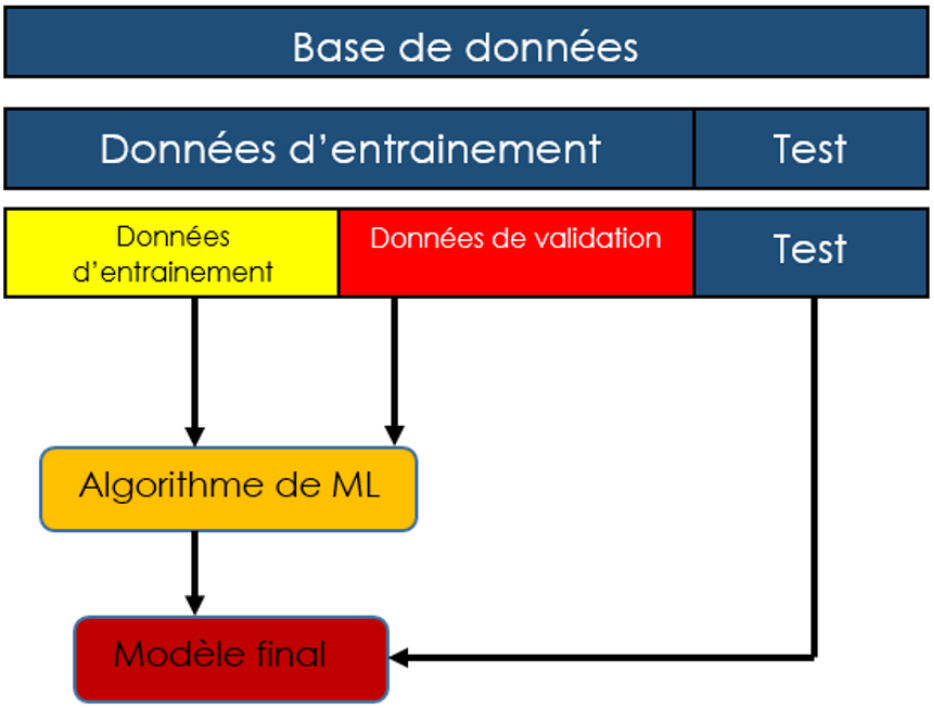


Figure 11 : Classification des données [14]

2.1 Description des données pour la classification d'image

On a trouvé plus de 50 000 images dans la base de données PlantVillage. Ils sont basés sur plusieurs plantes comme les pommes, les oranges, les tomates etc. On utilisera pour une plante spécifique afin que notre application se concentre sur une tâche pointue. On a exploité alors la base de données des tomates car ce dernier est plus complet par rapport aux autres plantes [20]. On a caractérisé notre base de données suivant 10 classes selon la maladie présente dans les feuilles de tomate :

- Tomato Bacterial spot
- Tomato Early blight
- Tomato healthy
- Tomato Late blight

- Tomato Leaf Mold
- Tomato Septoria leaf spot
- Tomato Spider mites Two-spotted spider mite
- Tomato Target Spot
- Tomato mosaic virus
- Tomato Yellow Leaf Curl Virus

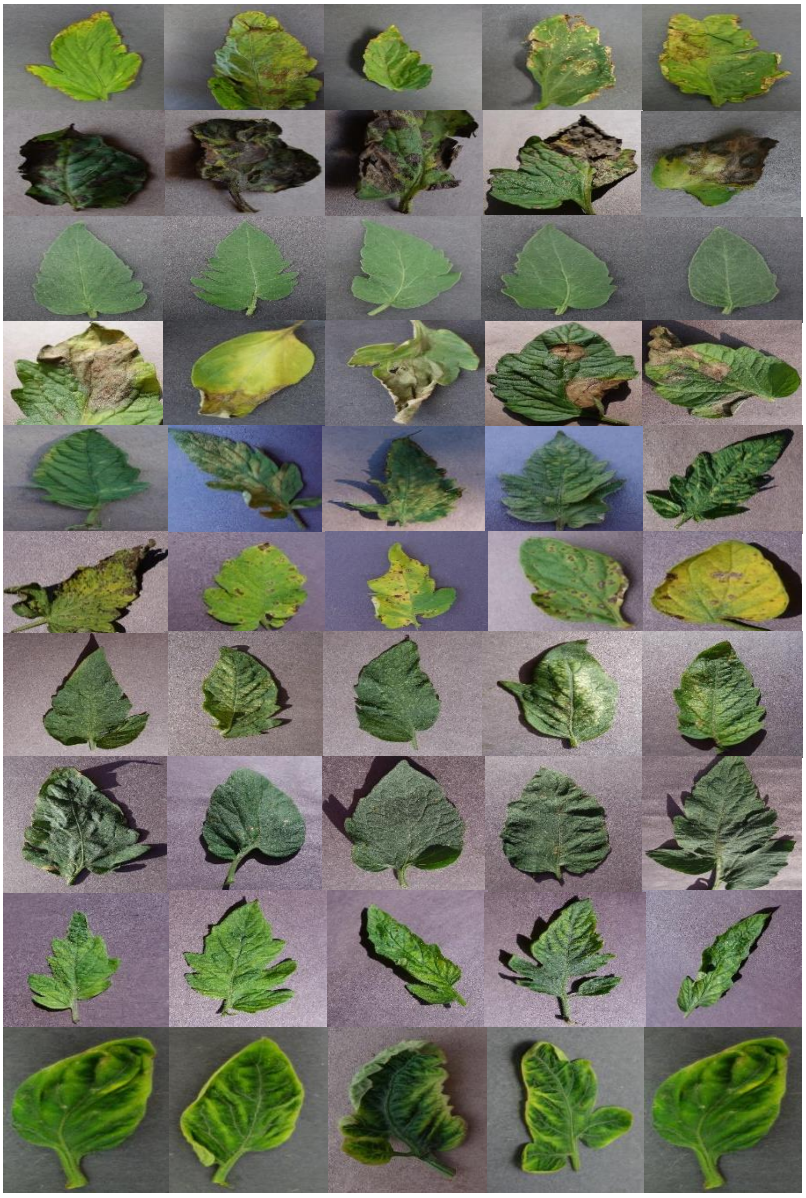


Figure 12: Exemples des différents phénotypes des maladies des feuilles de tomates [20]

2.2 Description des données pour la détection d'objet

On a utilisé plus de 1500 images dans notre l'application de détection de masque. Notre base de données est caractérisée de personnes en trois classes :

- Ce qui ne porte pas de masque
- Ce qui ne le porte pas correctement
- Ce qui le porte correctement

.

Figure 13: Un extrait de la base données sur les l'application de détection de masque

3 Modélisation

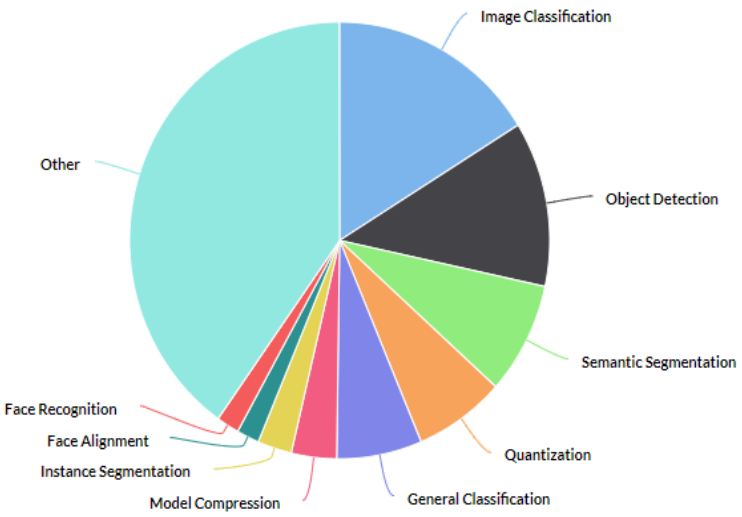
Une fois les données prêtes on pourra commencer à créer notre modèle. On dispose des offres d'accès pratiques à de nombreux modèles les plus performants sur les tâches de classification et détection d'image. Ce sont des modèles d'apprentissage en profondeur mis à disposition parallèlement à des pondérations pré-entraînées sur le jeu de données ImageNet,

COCO etc [21]. Certains modèles de détection comme le modèle Faster-RCNN donnent plus de précision mais plus lente et demande une alimentation de machine puissante pour une meilleure inférence ce qui n'est pas conforme dans le contexte TinyML. De ce fait notre choix se base sur les architectures de « Single Shot Detector » [23] permettant une détection plus rapide en dépit de sa précision avec l'extraction du modèle « MobileNet ». Ces améliorations permettent au SSD de correspondre à la précision du Faster R-CNN en utilisant des images de résolution inférieure, ce qui augmente encore la vitesse. Selon la comparaison suivante, il atteint la vitesse de traitement en temps réel et bat même la précision du Faster R-CNN [23].

Tableau 3: Caractéristique de quelque extrait de modèle [23]

Méthode	mAP	FPS	Résolution d'entrée
Faster R-CNN (VGG16)	73.2	7	1000 x 600
YOLO (VGG16)	66.4	21	448 x 448
SSD300	74.3	46	300 x 300
SSD512	76.8	19	512 x 512

MobileNetV2 est une architecture de réseau de neurones convolutifs qui se tourne dans le déploiement des machines de faible modèle comme des microcontrôleurs ou appareils mobiles [22]. MobilenetV2 est utilisé d'autant plus dans la classification d'image, détection d'objet et aussi pour d'autres tâches présentées dans la figure suivante.



Tâche	Papiers	Partager
Classification des images	37	16,16 %
Détection d'objets	28	12,23 %
Segmentation sémantique	19	8,30 %
Quantification	16	6,99%
Classification générale	15	6,55%
Compression du modèle	8	3,49%
Segmentation des instances	6	2,62 %
Alignement du visage	4	1,75%
Reconnaissance de visage	4	1,75%

Figure 14 : Les différentes taches utilisées pour le modèle MobilenetV2 [25]

4 Conversion et évaluation et du modèle

4.1 Conversion

TensorFlow est le Framework exploité dans les étapes précédentes. Elle est nécessaire dans sa capacité de former un modèle de machine Learning appelé ProtoBuffer. Ce dernier se présente sous forme d'un fichier «.Pb», c'est une représentation complète d'un format prêt à être employé mais demande une puissance d'exploitation pour le déploiement. Avec le Framework TensorFlow Lite, on passe par une série d'étapes pour adopter ce grand modèle et le convertir en un format qui est plus réduit, plus compact et adapté aux appareils terminaux

pour l'inférence sur des petits systèmes comme les téléphones, systèmes embarqués et microcontrôleurs.

4.2 Métrique d'évaluation

Dans l'évaluation, on va se focaliser sur les métriques de précision, la fonction de perte au niveau des sorties des modèles. La précision est le rapport entre les prédictions correctes et les prédictions totales. Ce dernier montre un aperçu de la fiabilité d'un réseau. Mais même avec un pourcentage assez élevé, on ne peut pas toujours se fier à ce paramètre car un réseau a peut-être subi un surentrainement ou des hypers paramètres non prises en compte. De ce fait on peut avoir quatre scénarios possibles correspondant au tableau suivant.

Tableau 4: Scénarios de prédiction en fonction des vraies valeurs [14]

		Vraie Valeur	
		1	0
Prédiction	1	Vraie Positive	Fausse Négative
	0	Fausse Positive	Vraie Négative

Dans le cas de la prédiction on peut manœuvrer sous deux éventualités :

$$Précision = \frac{VP}{VP + FP}$$

$$Précision\ total = \frac{VP + VN}{VP + FN + FP + VN}$$

VP : Vraie Positive

VN : Vraie Négative

FP : Fausse Positive

FN : Fausse Négative

Dans la notion de fonction de perte on pourra minimiser notre perte étant donné que la direction du gradient va pouvoir atteindre le minimum avec petit pas du taux d'apprentissage et plus le pas est petit plus il on se rapproche à la perte minimale.

La fonction de perte est issue d'une fonction d'erreur. Il existe différentes fonctions d'erreur tel que : l'erreur quadratique moyenne, l'erreur absolue moyenne ou la perte Huber etc. Dans notre classification on a utilisé la fonction de perte croisée

$$Perte = \sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

- n : nombre de valeurs scalaires dans la sortie du modèle
- i : itération
- y_i : la valeur cible correspondante
- $\hat{y}_i$: la valeur scalaire de sortie du modèle

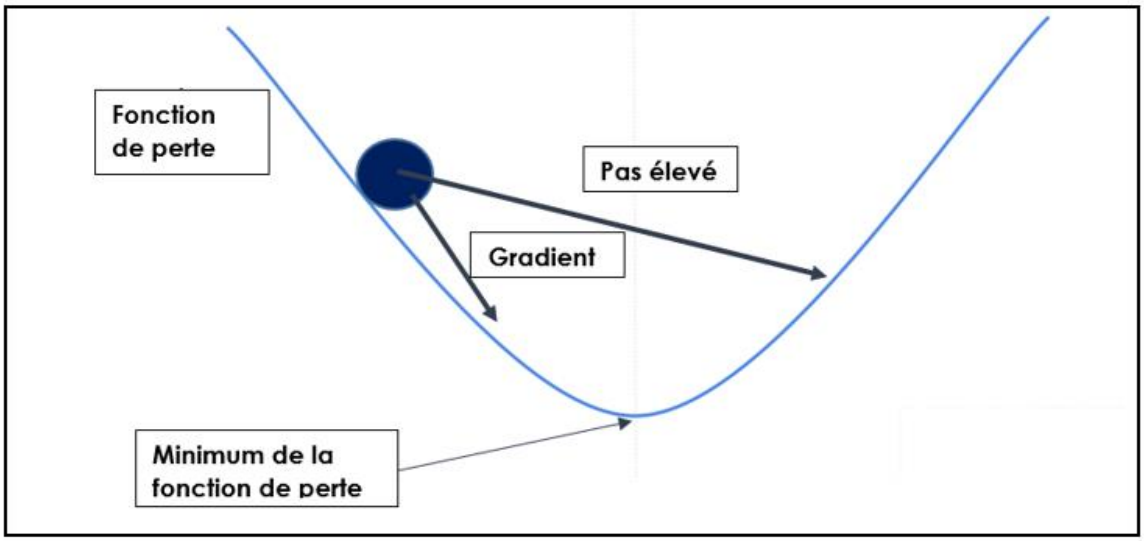


Figure 15: Exemple de fonction de perte [14]

Dans la notion de fonction de perte on pourra minimiser notre perte étant donné que la direction du gradient va pourvoir atteindre le minimum avec petit pas du taux d'apprentissage et plus le pas est petit plus il on se rapproche à la perte minimale. Une fois dans le déploiement on va pour voir mesurer l'inférence de notre modèle en termes de latence et les caractéristiques des matériels minimaux sur l'appareil déploiement.

5 Description des matériels utilisés dans notre processus

5.1 Plateforme Edge-Impulse et Google Colab (classification d'image)

Edge Impulse est une plate-forme de développement, où les utilisateurs peuvent contribuer et étendre à la fois les algorithmes et la prise en charge des appareils cibles. Le logiciel de l'appareil, y compris les SDK, les clients et le code généré qui est fourni en open

source sous une licence Apache 2.0. Cela signifie qu'aucun GPU ou TPU n'est nécessaire car tout l'apprentissage machine et la formation sur le réseau neuronal est effectuée au préalable dans le Cloud avec des méthodes avancées. Edge Impulse génère un modèle entraîné qui se déploie dans l'appareil et lui permet de classer les images (ou le son, le mouvement…) dans le bord sans aucune exigence matérielle particulière.

D'un autre côté, on a utilisé Google Colab qui est un environnement en ligne créer par Google permettant aux utilisateurs d'accéder aux matériels de puissance capables de mettre en œuvre des grands réseaux d'envergure. Grace à cette interface on a eu avoir l'accès à une carte graphique TeslaK80 qui est un GPU ayant les caractéristiques permettant de faciliter la formation :

- 4992 cœurs CUDA avec conception bi-GPU
- 2,91 TFlops de performances en double précision avec NVIDIA GPU Boost
- 8,73 TFlops de performances en simple précision avec NVIDIA GPU Boost
- 24 Go de mémoire GDDR5
- 480 Go/s de bande passante globale
- Protection ECC pour fiabilité renforcée
- Optimisation serveur garantissant un rendement maximal au Data Center

5.2 Machine personnelle (détection d'objets)

Pour mettre en expérience l'évaluation de la durée de l'entrainement et l'application de notre réseau de neurones, on a entrainé le modèle sur notre machine personnelle et exploité l'inférence du modèle de sortie (ProtoBuffer) sur la machine. Le tableau suivant montre les caractéristiques de la machine utilisée :

Tableau 5: Caractéristique de la machine personnelle

Processeur	Intel® Core™ i7-6700HQ
Carte graphique	NVIDIA GeForce GTX 960M 4Gb
Taille de Stockage	256G SSD/ 1TB HDD
Taille de la RAM	8Gb

6 Conclusion

Dans ce chapitre on a exploré le processus de travail à travers les différentes plateformes notamment Edge Impulse et Google Colab pour la classification d'image et sur la machine personnelle pour la détection d'objet. La collecte des données est la base de tout le travail car toutes les terminologies de notre travail dépendent de la qualité et la quantité des informations issues de nos données. C'est une tâche complexe car d'un côté elle demande énormément de temps pour réussir à inclure suffisamment de caractéristiques saillantes et d'un autre côté des notions de confidentialité complique la collecte. Alors pour fonder les principes on a utilisé des bases de données disponibles. Ensuite pour éviter de reprendre tout l'entrainement de zéros, la technique de Transfert Learning permet de simplifier l'entrainement en se focalisant sur les couches de sortie d'un réseau de neurones prédéfini pour modifier les tâches de ce dernier à une autre tâche similaire personnalisé. Enfin on va convertir notre propre modèle grâce au Framework TensorflowLite afin de les adapter aux structures des systèmes embarqués de puissance limité et puis les évaluer suivant les métriques et la capacité requise pour l'exploitation.

Chapitre III
Classification et détection d’objets d’intérêts

1 Introduction

Il est important que la production alimentaire mondial augmente, particulièrement dans les pays en développement, ou cette dernière doit atteindre d'ici 2050 un gain de 70%. Et cela en conséquent a l'augmentation exponentiel des populations dans ces régions du monde [20]. À l'heure actuelle, des maladies réduisent le rendement potentiel de 40 % en moyenne et de nombreux agriculteurs subissent des pertes de rendement pouvant atteindre 100 % à cause de l'indétermination de ces maladies. D'un autre coté sur le cadre de santé, la maladie à coronavirus raisonne actuellement une forte oppression sur la population mondiale avec un nombre plus de 170 millions de cas enregistré dans le monde. Le port des masques est l'un des premières barrières efficaces car il est conçu pour filtrer et réduire jusqu'à 99% le risque de propagation des gouttelettes buccale ou nasale. Généralement, la plupart des gens ne portent pas de maques suite à un oubli involontaire ou un manque d'habitude. Récemment, des méthodes de la robotique et l'apprentissage machine proposent une solution pour la détection et la reconnaissance visuellement distinguable ; ou un robot muni d'une caméra utilise les méthodes de vision par ordinateur pour procéder à la reconnaissance et la catégorisation à partir de leur apparence (couleur, forme, etc.).

Dans ce chapitre, nous proposons en premier lieu une solution permettant la classification des feuilles de tomate pour aider les agriculteurs à remédier et éviter la propagation des maladies. En deuxième lieu une solution de surveillance en intégrant des algorithmes capables de faire la détection d'objet (masque) dans une caméra afin de réduire la propagation du virus.

2 Classification des feuilles de tomate

2.1 Entrainement à partir de l'interface Edge-Impulse :

Pour entraîner un modèle d'apprentissage automatique avec Edge Impulse, on crée un compte Edge Impulse, puis on démarre un nouveau projet. L'image ci-dessous nous montre la plateforme du travail.

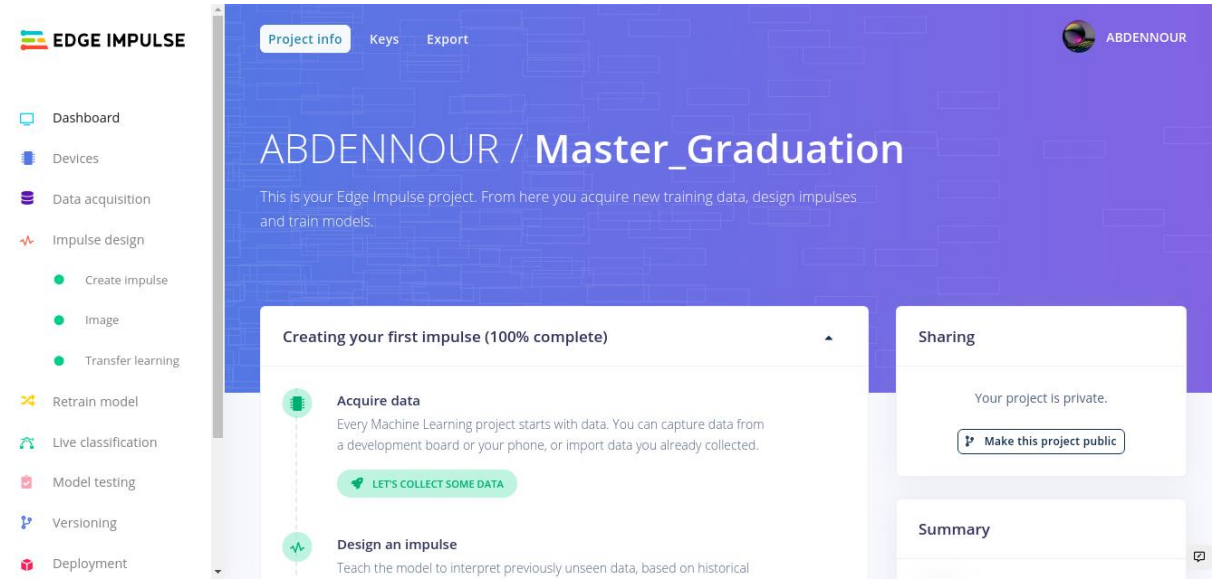


Figure 16: Plateforme Edge Impulse

N.B : On peut collecter les données en connectant un appareil mobile ou une carte Raspberry Pi ou n'importe quelle autre machine avec notre compte Edge Impulse(en cliquant sur Devices)

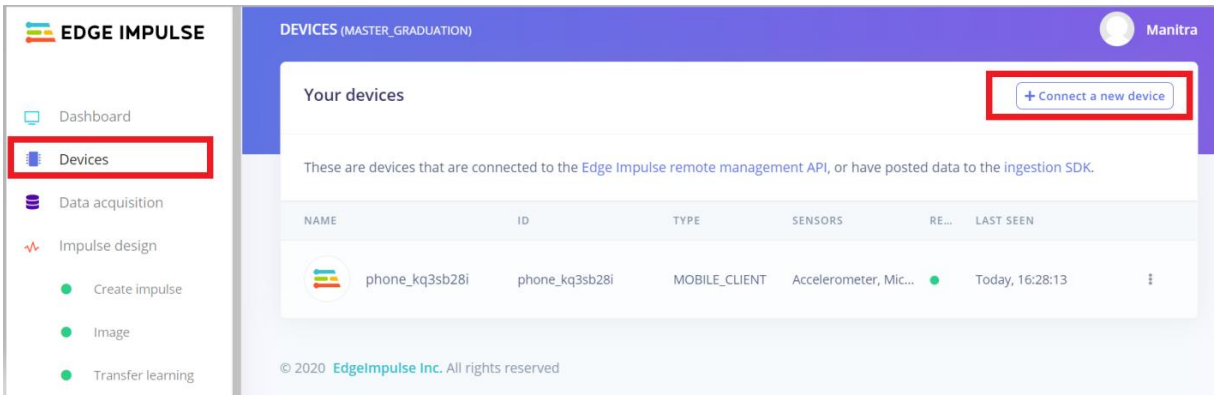
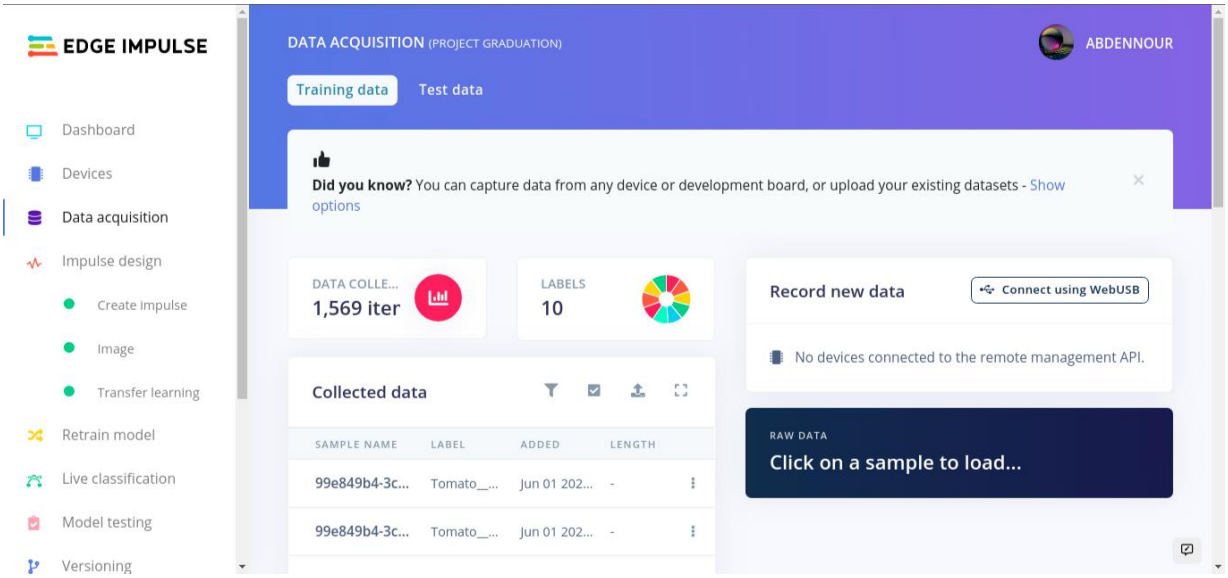


Figure 17: Liaison de l'appareil sur Edge Impulse

2.1.1 Les données

Comme mentionné précédemment, nous utilisons Edge Impulse Studio pour entraîner notre modèle de classification d'images. Pour cela, nous avons importé un ensemble de données

des feuilles de tomate dans plusieurs états de santé, contenant les échantillons d'objets que nous aimerions classer, afin qu'on puisse faire la distinction entre elles.



Nous avons collecté 1569 images classées sur 10 étiquetages pour notre modèle d'entrainement.

N.B : Les données de test doivent représenter au moins 20 % des données d'entraînement pour qu'on puisse avoir une bonne classification.

2.1.2 Formation du modèle

Maintenant, sur la section « Créer une impulsion », en cliquant sur « Ajouter un bloc de traitement » puis sur le bouton « Ajouter » à côté du bloc « Image » pour ajouter un bloc de traitement qui normalisera les données d'image et réduira la profondeur de couleur. Après cela, en cliquant sur le bloc « Apprentissage par transfert (images) » pour récupérer un modèle pré-entraîné destiné à la classification d'images, sur lequel nous effectuerons un apprentissage par transfert pour but de régler notre tâche de reconnaissance sur les feuilles de tomate. Ensuite sur « Enregistrer l'impulsion ».

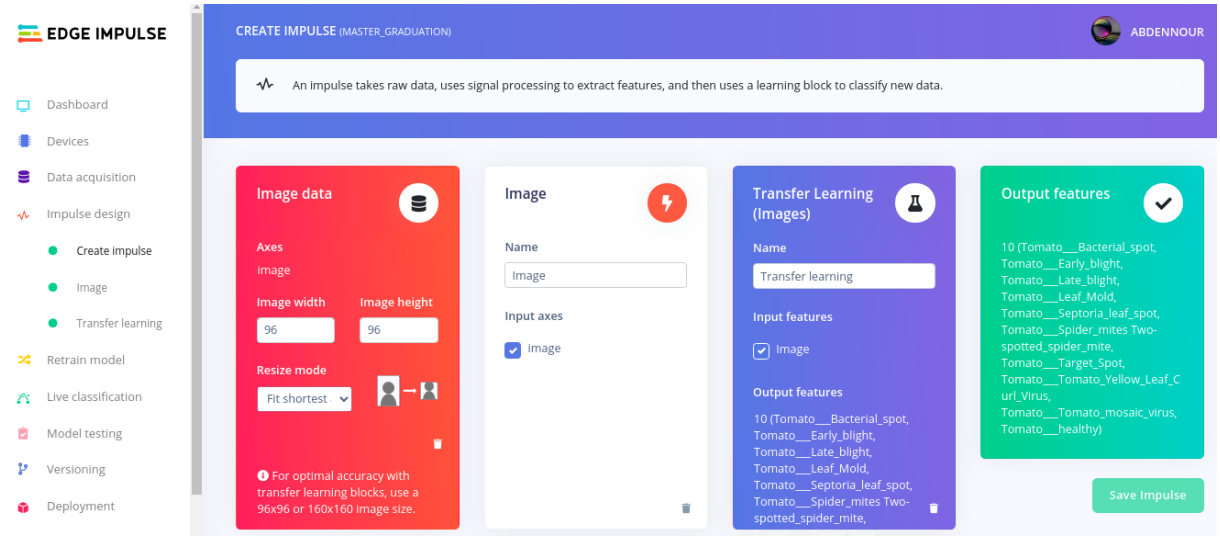


Figure 18 : Pré-entrainement du modèle

N.B : la taille des images impulsées de 96x96 pixels suffira pour un modèle d'entrainement de classification d'images.

2.1.3 Extraction des caractéristiques

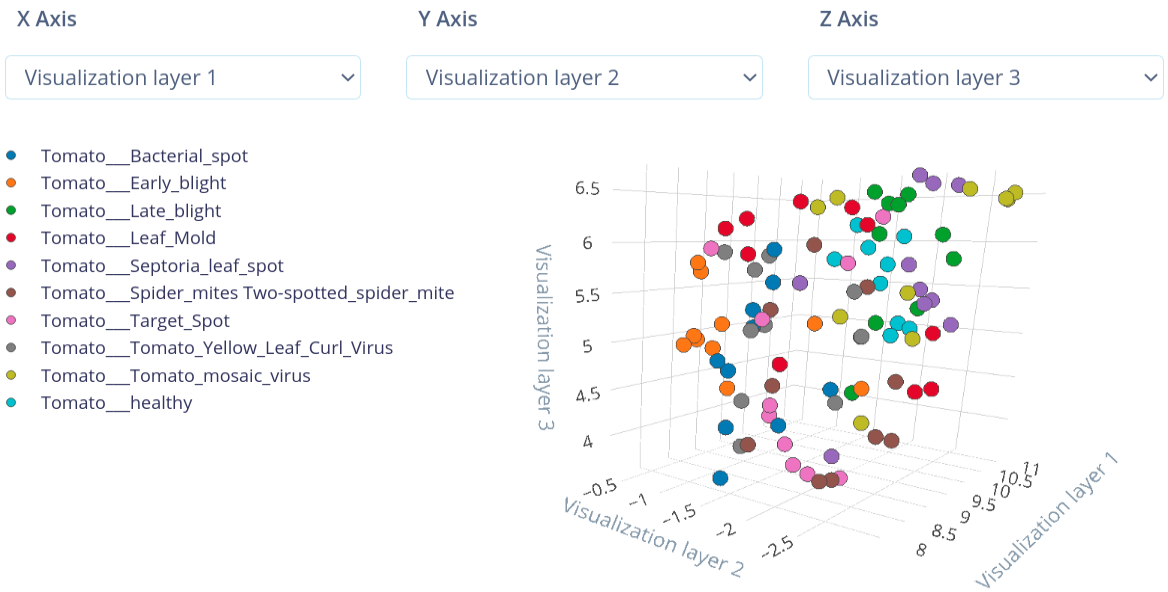


Figure 19 : Extraction des échantillons 3D

Avec le modèle 3D généré à partir des données d'entraînement téléchargées sur le stockage, on a pu voir à quel point les objets classés sont différents (figure ci-dessus). Après avoir importé et traité les données. On va commencer à former le réseau de neurones pour reconnaître les modèles des données.

Pour entraîner le réseau de neurones pour l'apprentissage par transfert, on a utilisé les paramètres ci-dessous :

- Nombre de cycles de formation jusqu'à 100
- Taux d'apprentissage :

[0,0001 / 0,0005 / 0,001 / 0,005 / 0,0075 / 0,01 / 0,05]

- Augmentation des données : désactivée
- Indice de confiance minimum : 0,85
- La Durée de l'entrainement nous a pris 1h30m en moyennes.
- Modèle du réseau de neurones utilisé :
  MobileNetV1 96x96 0.1 ; MobileNetV1 96x96 0.2 ; MobileNetV2 96x96 0.05 ; MobileNetV2 96x96 0.1 ; MobileNetV2 96x96 0.35.

Les réseaux de neurones sont des algorithmes conçus pour reconnaître des modèles tel que : MobilNetV2, dont on a entrainé notre modèle avec **100 époques.**

En cliquant sur 'Start training' le réseau de neurones commencera à calculer toutes les images et à s'entraîner pour générer le modèle d'apprentissage automatique. Le tableau ci-dessous nous montre la précision prédite pour chaque modèle du réseau de neurones en fonction des différentes valeurs du taux d'apprentissage :

Tableau 6: Expérimentation issue des formations de chaque modèle

Modèle\Taux d'apprentissage =>Précision	**0.0001**	**0.0005**	**0.001**	**0.005**	**0.0075**	**0.01**	**0.05**
MobileNetV1 96x96 0.1	23.2	31.2	28.0	32.8	29.3	31.5	8.6
MobileNetV1 96x96 0.2	52.5	64.6	59.2	18.5	35.0	34.4	8.6
MobilNetV2 96x96 0,05	63.7	62.4	67.2	55.4	51.3	46.8	8.6
MobileNetV2 96x96 0.1	62.7	62.7	64.3	8.6	8.6	8.6	8.6
MobileNetV2 96x96 0,35	79.3	69.5	61,8	40.1	38.9	35.4	8.6

Après l'entrainement de ces différents modèles, on a pu extraire les meilleures matrices de confusion pour chaque type du modèle (figure ci-dessous)

MobileNetV1 96x96 0.2

ACCURACY 64.6% LOSS 1,03

Confusion matrix (validation set)

	T...	T...	T...	T...	T...	T...	T...	T...	T...	T...
T...	50%	3.1%	6.3%	18.8%	21.9%	0%	0%	0%	0%	0%
T...	8.8%	38.2%	0%	5.9%	2.9%	0%	44.1%	0%	0%	0%
T...	3.7%	3.7%	66.7%	7.4%	14.8%	0%	3.7%	0%	0%	0%
T...	0%	0%	3.0%	45.5%	39.4%	0%	12.1%	0%	0%	0%
T...	5.9%	8.8%	2.9%	0%	55.9%	0%	26.5%	0%	0%	0%
T...	0%	0%	0%	0%	0%	55.6%	37.0%	0%	0%	7.4%
T...	3.7%	0%	0%	0%	0%	0%	92.6%	0%	3.7%	0%
T...	2.7%	5.4%	0%	2.7%	2.7%	10.8%	0%	70.3%	0%	5.4%
T...	0%	0%	0%	0%	0%	0%	16.7%	0%	83.3%	0%
T...	0%	0%	0%	0%	2.6%	5.1%	0%	0%	0%	92.3%
F1...	0.57	0.48	0.73	0.51	0.47	0.63	0.53	0.83	0.89	0.91

MobileNetV1 96x96 0.1

ACCURACY 31.5% LOSS 7,36

Confusion matrix (validation set)

	T...	T...	T...	T...	T...	T...	T...	T...	T...	T...
T...	6.3%	3.1%	0%	18.8%	0%	3.1%	6.3%	3.1%	56.3%	3.1%
T...	8.8%	5.9%	8.8%	8.8%	8.8%	5.9%	2.9%	8.8%	23.5%	17.6%
T...	7.4%	7.4%	11.1%	7.4%	0%	3.7%	0%	7.4%	29.6%	25.9%
T...	0%	0%	6.1%	9.1%	6.1%	0%	3.0%	6.1%	27.3%	42.4%
T...	0%	0%	14.7%	2.9%	29.4%	5.9%	2.9%	0%	2.9%	41.2%
T...	0%	3.7%	3.7%	0%	0%	22.2%	3.7%	0%	14.8%	51.9%
T...	0%	7.4%	0%	7.4%	0%	11.1%	3.7%	0%	44.4%	25.9%
T...	2.7%	0%	0%	5.4%	0%	0%	0%	32.4%	0%	59.5%
T...	0%	0%	0%	4.2%	0%	0%	0%	0%	87.5%	8.3%
T...	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%
F1...	0.10	0.10	0.15	0.11	0.41	0.29	0.06	0.42	0.40	0.47

MobileNetV2 96x96 0.1

ACCURACY 64.3% LOSS 1,68

Confusion matrix (validation set)

	T...	T...	T...	T...	T...	T...	T...	T...	T...	T...
T...	46.9%	12.5%	6.3%	3.1%	15.6%	0%	15.6%	0%	0%	0%
T...	11.8%	41.2%	2.9%	2.9%	2.9%	0%	38.2%	0%	0%	0%
T...	3.7%	7.4%	48.1%	3.7%	11.1%	0%	3.7%	3.7%	7.4%	11.1%
T...	0%	0%	0%	69.7%	15.2%	3.0%	6.1%	0%	6.1%	0%
T...	0%	0%	0%	2.9%	73.5%	0%	17.6%	0%	2.9%	2.9%
T...	0%	0%	0%	0%	7.4%	48.1%	40.7%	0%	0%	3.7%
T...	0%	3.7%	0%	0%	0%	14.8%	70.4%	0%	7.4%	3.7%
T...	5.4%	8.1%	2.7%	2.7%	5.4%	5.4%	2.7%	64.9%	0%	2.7%
T...	0%	0%	0%	0%	0%	0%	4.2%	0%	95.8%	0%
T...	0%	0%	0%	0%	0%	2.6%	10.3%	0%	2.6%	84.6%
F1...	0.56	0.48	0.59	0.75	0.65	0.54	0.42	0.77	0.84	0.84

MobileNetV2 96x96 0.05

ACCURACY 67.2% LOSS 1,88

Confusion matrix (validation set)

	T...	T...	T...	T...	T...	T...	T...	T...	T...	T...
T...	43.8%	3.1%	18.8%	9.4%	6.3%	3.1%	3.1%	12.5%	0%	0%
T...	8.8%	41.2%	5.9%	5.9%	5.9%	2.9%	11.8%	14.7%	0%	2.9%
T...	0%	11.1%	63.0%	7.4%	7.4%	0%	0%	7.4%	0%	3.7%
T...	0%	3.0%	9.1%	72.7%	9.1%	0%	0%	0%	0%	6.1%
T...	0%	2.9%	11.8%	8.8%	61.8%	0%	8.8%	2.9%	2.9%	0%
T...	0%	0%	0%	0%	3.7%	59.3%	3.7%	3.7%	7.4%	22.2%
T...	7.4%	3.7%	0%	0%	0%	22.2%	63.0%	0%	3.7%	0%
T...	8.1%	0%	0%	0%	2.7%	0%	0%	86.5%	0%	2.7%
T...	0%	0%	4.2%	0%	0%	0%	4.2%	0%	91.7%	0%
T...	0%	0%	0%	5.1%	2.6%	0%	2.6%	0%	2.6%	87.2%
F1...	0.52	0.51	0.57	0.70	0.63	0.63	0.62	0.78	0.86	0.81

Figure 20 : Matrices de confusion issue des expérimentations

Ces matrices nous montrent les pourcentages de précisons prédite sur la connaissance des feuilles de tomate entrainé pour chaque expérimentation des réseaux de neurones.

La matrice de confusion ci-dessous nous montre le pourcentage de meilleure précision dont on peut tirer profit sur cette partie avec le modèle du réseau de neurones **MobileNetV2 96x96 0,35** qui est le plus efficace pour notre mise en pratique de la classification d'image.

	T...	T...	T...	T...	T...	T...	T...	T...	T...	T...
T...	81.3%	6.3%	6.3%	0%	3.1%	0%	3.1%	0%	0%	0%
T...	5.9%	70.6%	5.9%	2.9%	5.9%	0%	8.8%	0%	0%	0%
T...	0%	7.4%	77.8%	7.4%	0%	0%	0%	7.4%	0%	0%
T...	0%	9.1%	3.0%	72.7%	12.1%	0%	0%	0%	0%	3.0%
T...	2.9%	0%	2.9%	0%	85.3%	0%	5.9%	0%	2.9%	0%
T...	3.7%	7.4%	3.7%	0%	0%	51.9%	18.5%	7.4%	0%	7.4%
T...	7.4%	3.7%	0%	0%	0%	7.4%	70.4%	0%	11.1%	0%
T...	0%	0%	0%	2.7%	0%	5.4%	0%	89.2%	2.7%	0%
T...	0%	0%	0%	0%	0%	4.2%	8.3%	0%	87.5%	0%
T...	0%	0%	0%	2.6%	0%	0%	0%	0%	0%	97.4%
F1...	0.81	0.71	0.76	0.77	0.83	0.61	0.64	0.89	0.84	0.95

Figure 21 : Matrice de confusion MobileNetV2 96x96 0,35

Une fois l'entraînement terminé, on voit que la précision est de 79,3 % et une perte qui n'est pas négligeable de 1,07 avec une matrice de confusion et les performances requises pour l'appareil de déploiement.

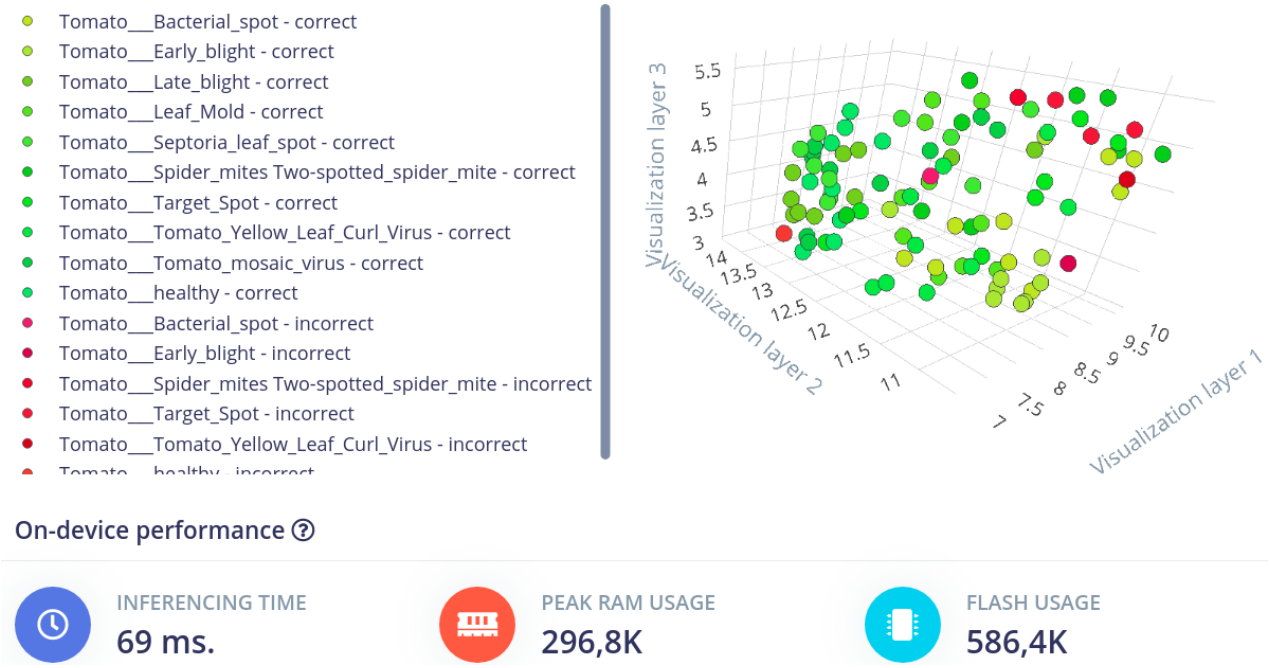


Figure 22: Explorateur de fonctionnalités (ensemble d'entraînement complet)

Avec un réseau de 16 neurones entraîné sur notre modèle on a pu traiter nos données en 69 ms du temps de traitements avec une consommation de 296,8K de mémoire vive et 586,4K du processeur.

2.1.4 Test et évaluation

Live Classification :

On prend une ou plusieurs photos de la partie test de l'acquisition d'image, pour qu'on puisse les classifier directement avec notre modèle entrainée (l'exemple de notre cas).

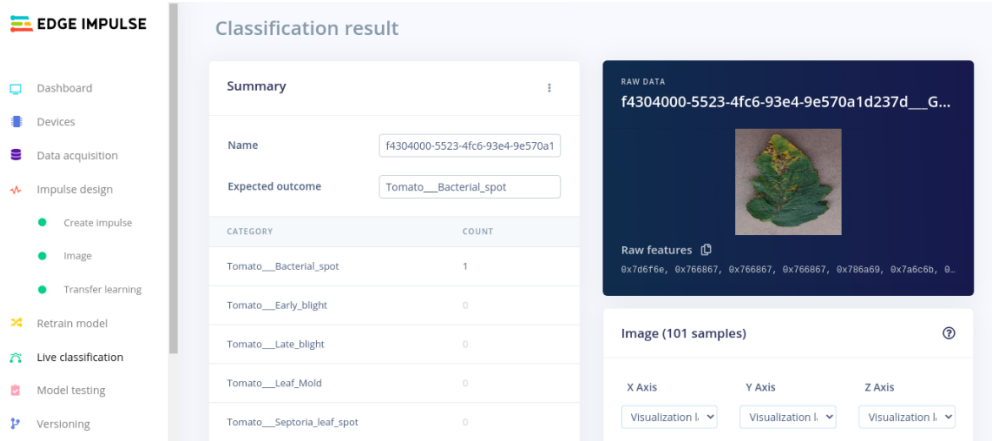
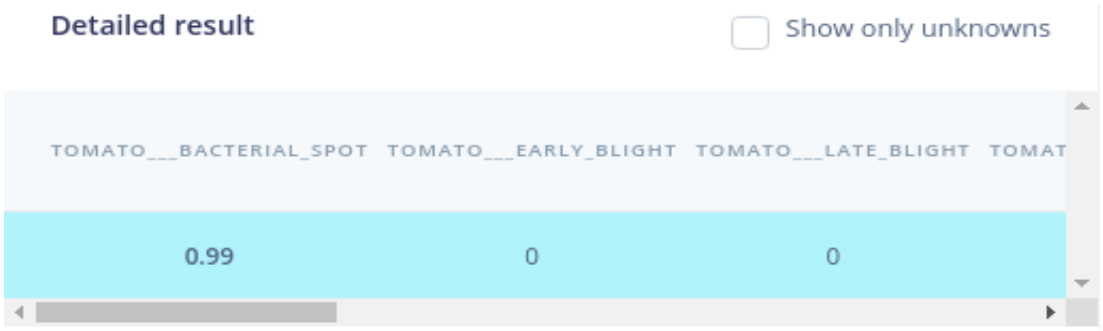


Figure 23: Résultat de la classification

On voit très bien que notre programme fonctionne parfaitement, il a pu classifier l'image selon la capture, la lecture et la classification des pixels, avec une précision de 99 % étant donné qu'on a pris l'image d'une feuille de plante Tomato___Bacterial_spot.



Test du modèle :

Pour pouvoir tester notre modèle, on a translaté minimum 20 % des données entrainé sur notre compte Edge Impulse vers les données test. Le modèle en sortie classifie toutes les images test en donnant un résultat de précision globale ; ce qui diffère sur la partie précédent (live classification) qui classifie une seule image.

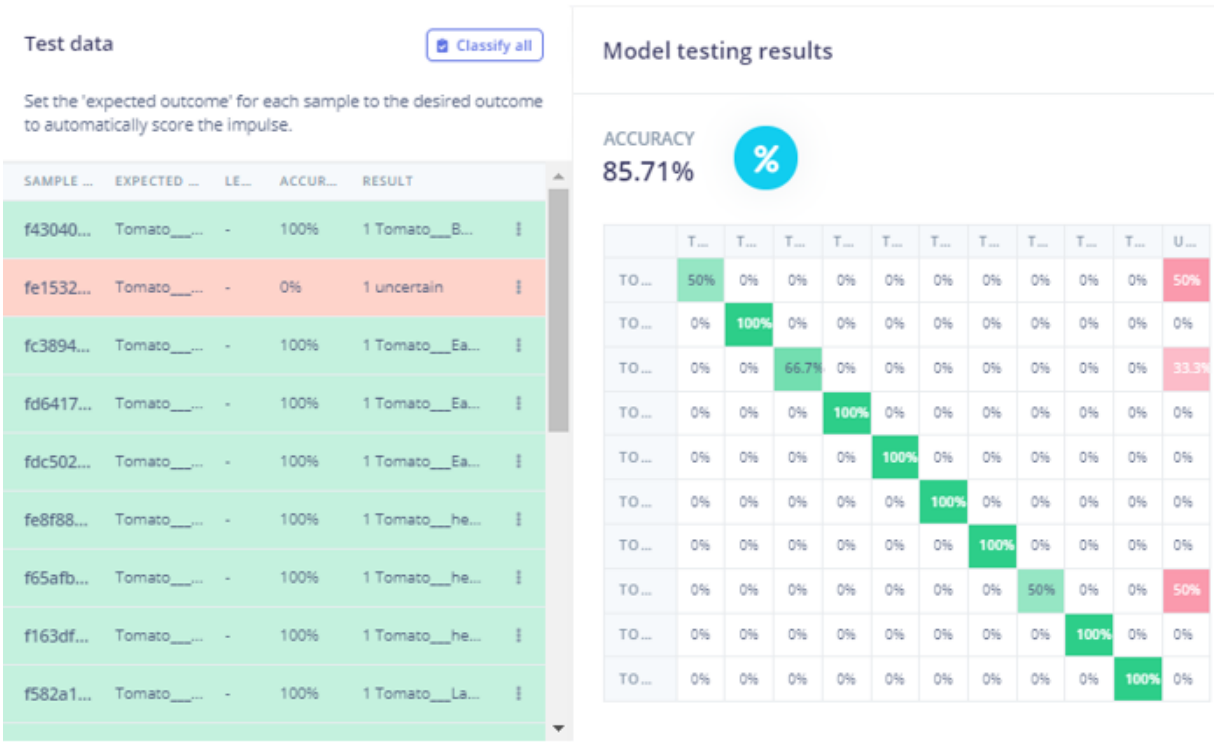


Figure 24: Test du modèle

On voit très bien que cette fois la précision est augmenté à 85,71 % Aussi les classes en couleur rouge nous montre les erreurs de classification avec une précision moins de 50 %.

2.1.5 Conversion et évaluation sur Edge import

L'interface Edge Impulse permet une optimisation et conversion simplifié lors de l'entrainement. Il permet de choisir entre les deux modèle Protobuffer (float32) et celle de FlattBuffer issue d'une quantification de 8 bits. L'utilisation des données réelles (float) nous donne des précisions mais exige par contres des performances élevées en termes de mémoire et latence de calcul. La conversion en INT8 ou UINT8 avec l'optimisation nous permet d'éliminer cette contrainte en adaptant à un IDE utilisé mais la précision diminuera en conséquence.

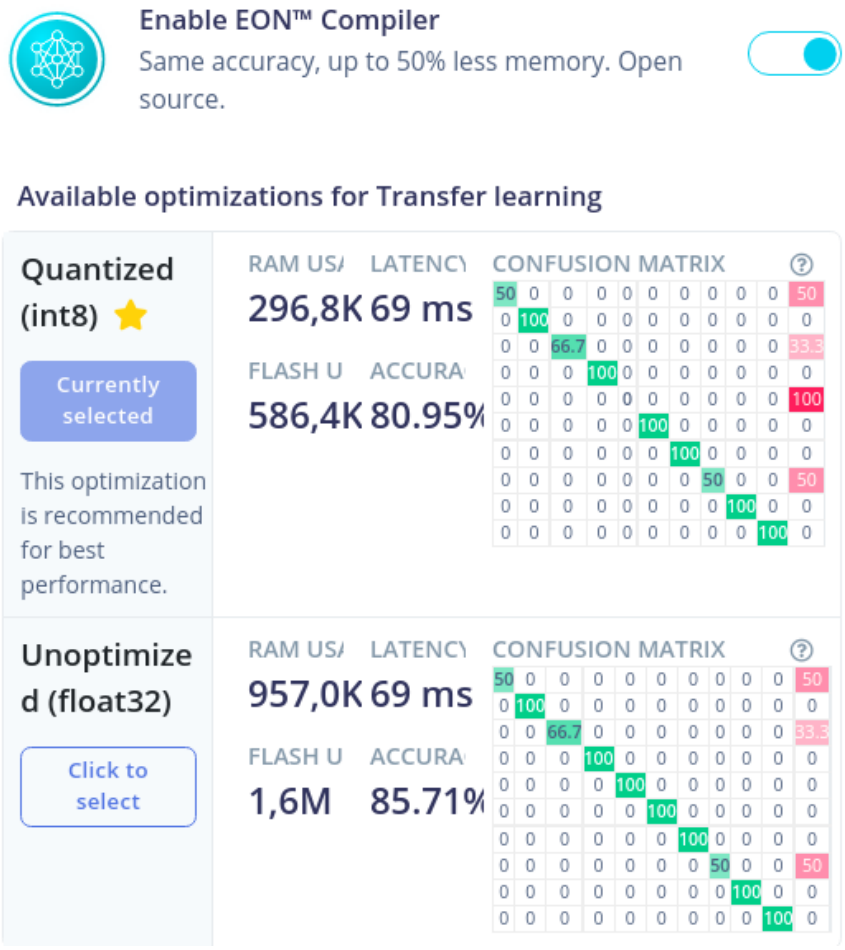


Figure 25 : choix de modèle entre la version optimisée et non optimisée

La précision du modèle non optimisé est de 85,71 % nécessitant une taille de 1,6 Mo et une mémoire vive de 957,0 Ko pour une latence de 69 ms. Avec l'optimisation la précision est

réduite à 80,95 % mais par contre on a une diminution considérable des configurations requises adapté au contexte TinyML avec une capacité de mémoire de 586,4 Ko et une mémoire vive de 296.8 Ko permettant une latence de 69ms.

2.2 Entrainement à partir de l'interface Google Colab

2.2.1 Les données

Avec Google Colab, on a importé les bases de données dans Drive afin de permettre l'exploitation de ces derniers dans l'interface Google Colab. Les données ont passé des prétraitements afin de l'adapter à notre réseau. Les images sont amincies à une taille de 256*256. Les pixels des images qui varient de 0 à 255 alors les diviser par 255 sera approprié au neurone. On a fait quelque rotation et effet miroir de quelque donnée d'entrainement afin d'éviter des positions uniformes.

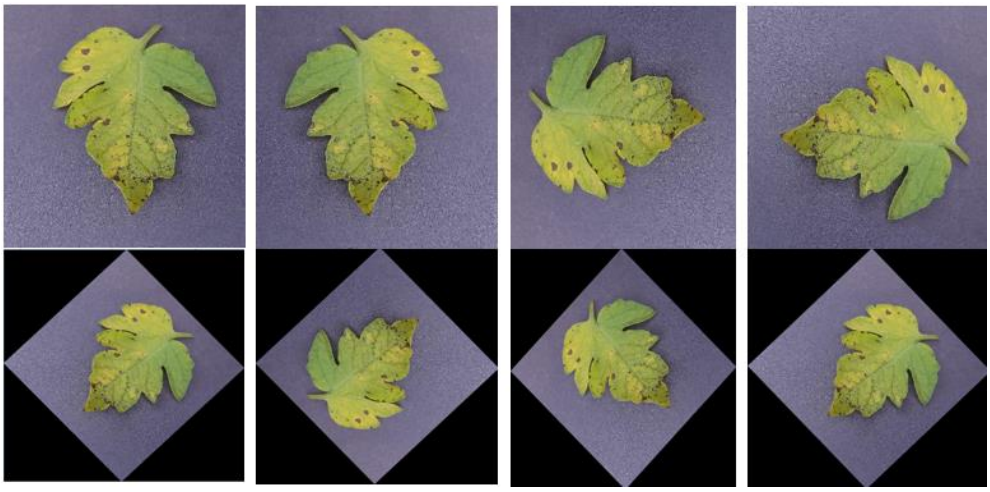


Figure 26: Extrait de présentation de l'augmentation de donnée

2.2.2 Formation du modèle

Pour notre choix de modèle pour la classification. Keras est une bibliothèque « open-source » utilisée dans l'IA. Elle offre un accès pratique à de nombreux modèles les plus performants sur les tâches de reconnaissance d'image ImageNet. Les applications Keras sont des modèles d'apprentissage en profondeur mis à disposition parallèlement à des pondérations pré-entraînées. MobileNetV2 est le choix raisonnable pour notre application. Elle a une taille

réduite (16 Mo) par rapport aux autres choix et de plus avec une précision qui est assez tolérable [24]. Pour entraîner notre réseau de neurones pour l'apprentissage par transfert, on a utilisé les paramètres ci-dessous :

- Nombre d'époque : 10
- Taux d'apprentissage à 0,003

On a utilisé TensorFlow-GPU qui permet d'utiliser la carte graphique pour fournir une puissance de traitement supplémentaire pendant l'entraînement. D'après notre expérience, l'utilisation de TensorFlow-GPU au lieu de TensorFlow classique (qui utilise le processeur) réduit environ 3 fois le temps de formation (de 17598 secondes ou 4.8 heures d'entrainement du notre modèle au lieu de 12h). La figure suivante montre le graphe de notre formation.

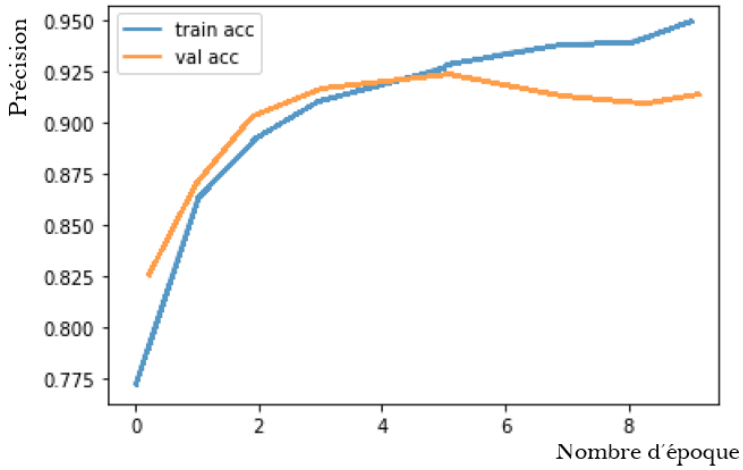


Figure 27 : Graphe de la précision lors de l'entrainement

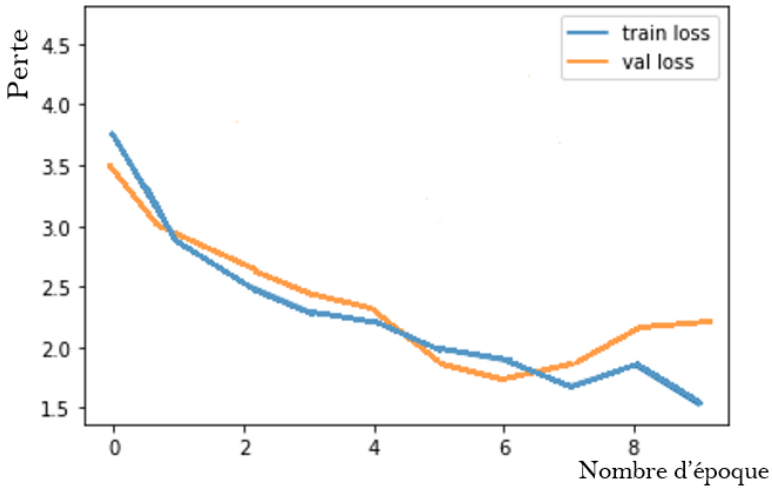


Figure 28 : Graphe de la fonction de perte lors de l'entrainement

Le premier graphe nous montre l'évolution de la précision de l'entrainement et de la validation durant la formation du modèle. D'après les suites d'expériences qu'on a effectuées, lors de la moitié de la précision, la validation commence à atteindre son apogée. Tout de même pour le graphe de la perte dans la deuxième figure, la fourche des deux métriques a commencé à donner un écart de divergence alors on a arrêté notre entrainement pour éviter le surapprentissage.

2.2.3 Conversion et évaluation sur Google Colab

Dans Google Colab, on a tout d'abord une sortie de modèle de type ProtoBuffer. L'entrainement de ce modèle a pu atteindre une précision de 96,7% mais la taille a été encore d'un côté assez élevé. Après la conversion par TensorFlow lite on a pu réduire la taille de 30% avec une taille de 7Mo mais en dépit de cela la précision à diminuer de 85%. Pour le déploiement de ce modèle FlattBuffer converti peut être implémenté sur un système embarqué comme la Raspberry Pi 4 mais avec le manque de temps et de disponibilité des équipements on n'a pas pu mis en pratique cette partie.

3 Détection d'Objet

3.1 Les données

Après avoir collecté les données, on les a partitionnées sous trois groupements comme suit :

- Les données d'entrainement à 75%
- Les données de validation à 20%
- Les données d'entrainement à 5%.

Pour les caractéristiques des images dans la détection d'objet, on peut avoir plusieurs objets sur une même image avec des positions variées. On a utilisé un outil d'étiquetage ou d'annotation d'image pour rendre lisibles un objet spécifique (masque) dans une image à la machine.

Figure 29: Extrait d’Etiquetage d’une image pour la détection d'objet.

Cette manipulation va créer pour chaque image étiquetée un fichier ‘.xml ‘ qui est un langage de structuration de données contenant les informations comme les coordonnées de l’objet à détecter sur l’image et le nom l’image liée au fichier d’extension ‘.xml ‘.

masque380.xml

```xml
<annotation>
    <folder>train</folder>
    <filename>masque380.jpg</filename>
    <path>C:\Tensorflow\workspace\training_demo\images\train\masque380.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>301</width>
        <height>400</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>with_mask</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>47</xmin>
            <ymin>210</ymin>
            <xmax>157</xmax>
            <ymax>359</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 30 : Exemple de contenu d'un fichier.xml

Ensuite ses derniers sont utilisés pour générer un autre type de fichier '.record'. Ce fichier est le type définitif utilisé pour le pipeline de l'entrainement.

3.2 Entrainement dans la machine personnelle (méthodologie)

On a utilisé le Framework TensorFlow-GPU pour fournir une puissance de traitement supplémentaire pendant l'entraînement. La durée de chaque entrainement qu'on a procéder avant de l'arrêter pour atteindre une perte entre la fourche de 0.2 à 0.01 est d'environ 15h en moyenne. TensorFlow enregistre une sortie similaire à celle-ci toutes les 100 étapes du processus, Cette sortie est utilisée dans TensorFlowBoard qui fournit des graphes de progression de la formation. Un des graphes importants est celui de perte globale du classificateur au fil du temps.

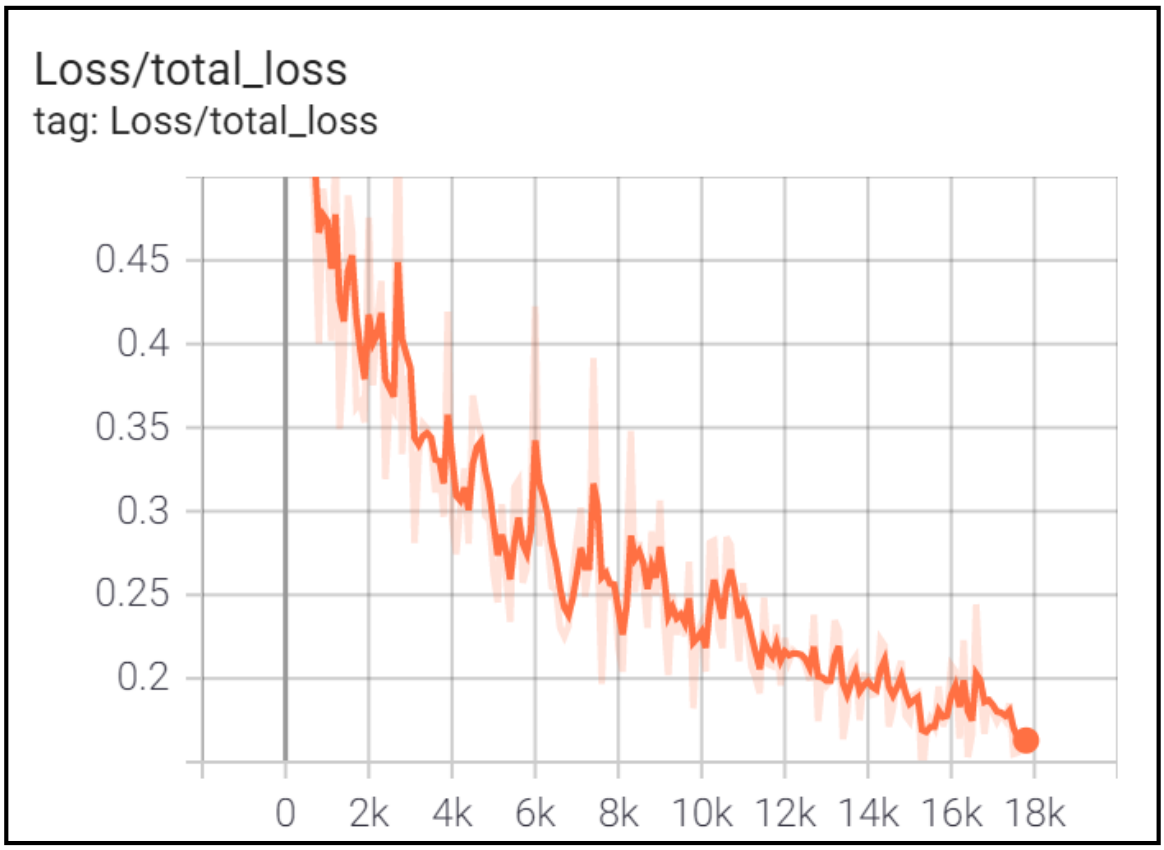


Figure 31: Graphe de la perte totale dans TensorFlow Board

3.3 Test et évaluation

Après l'entrainement, on a eu à la sortie un modèle ProtoBuffer de taille 10 834 Ko. Le modèle a pu atteindre une précision de 98% avec une perte totale inférieure à 0.2%. Comme on l'a décrit, on peut déployer ce modèle sur des appareils ayant une ressource performante. Alors avec notre machine personnelle on a pu exploiter le modèle.

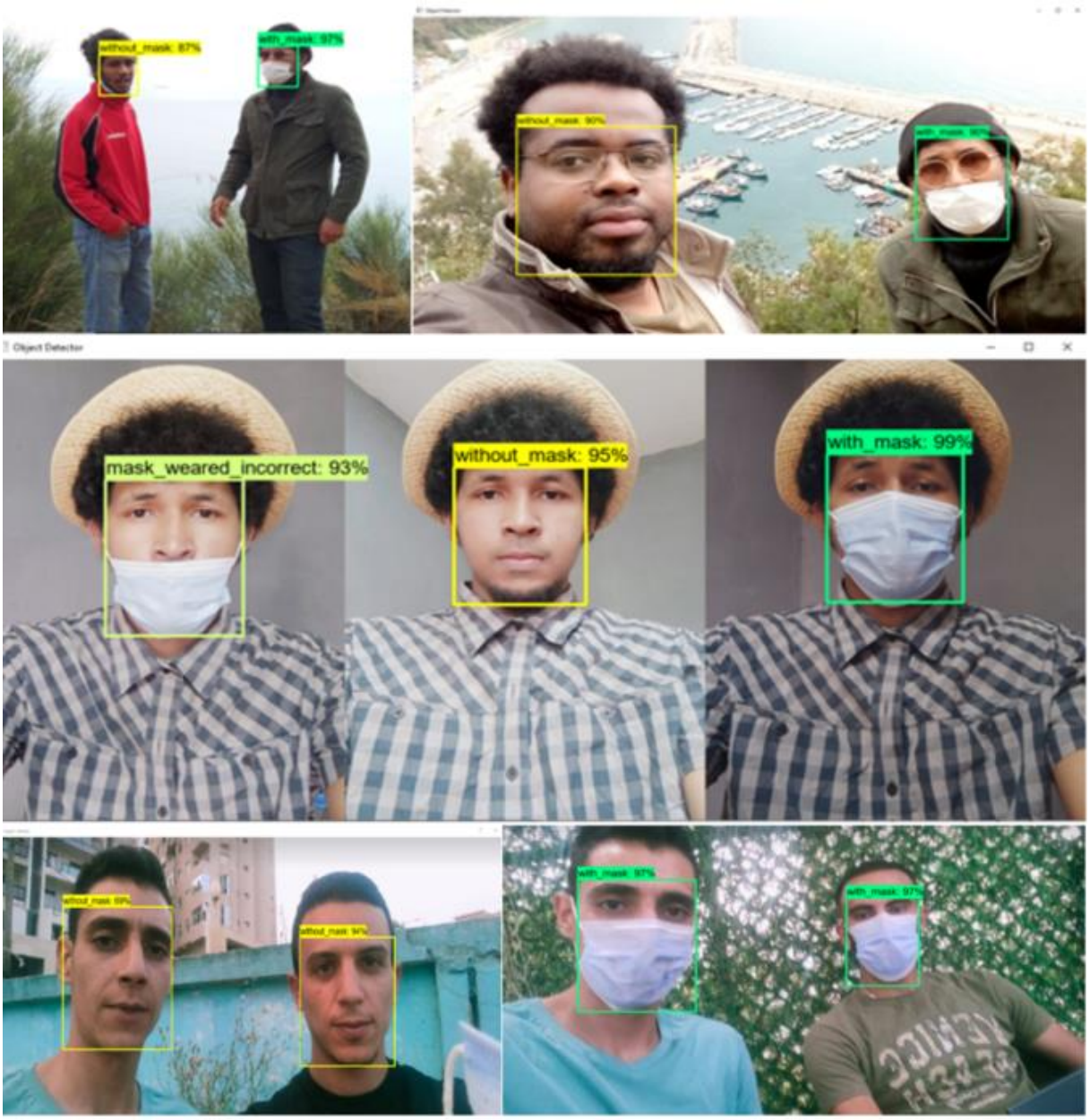


Figure 32: exemple de detection d'objet sur des images

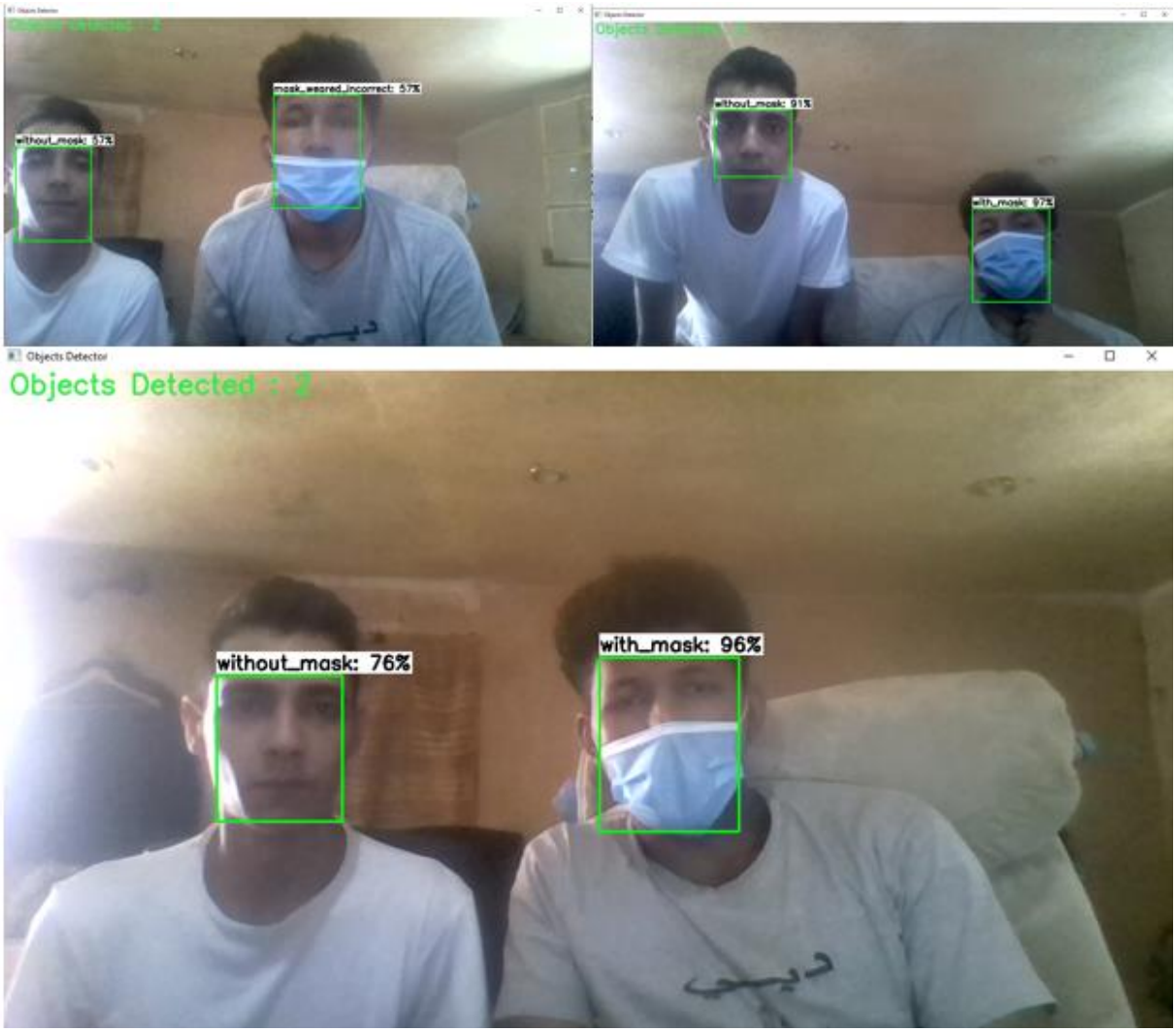


Figure 33 : Exemple de live détection

On a varié les fonds des images et la teinte de la peau des personnes figurantes afin de mettre en évidence la performance de notre réseau de neurones pour la détection d'image. La deuxième figure nous montre la détection d'objet en live et malgré la qualité d'image issue de la caméra on a pu encore avoir une précision assez élevée.

3.4 Conversion du modèle

Après la conversion par TensorFlow lite on a pu réduire la taille de 10 % avec une taille de 9Mo mais pour le déploiement de ce modèle FlattBuffer converti qui peut être implémenté sur un système embarqué mais avec le manque de temps et de disponibilité des équipements on n'a pas pu mis en pratique cette partie.

4 Conclusion

Dans ce dernier chapitre, on a mis en œuvre un réseau de neurones pour la classification des images et la détection des objets. Pour la classification des images, on a utilisé une base de données de feuille de tomate en les entrainant sur le modèle gelé de MobileNetV2. L'entrainement pour la classification d'image a été exploré sur deux plateformes qui sont Edge Impulse et Google Colab. Pour la détection d'image, on a utilisé une base de données pour la détection des masques en les entrainant sur le modèle SSD_MobileNet_V2. Pour la détection on a entrainé le modèle sur notre machine personnelle afin de faciliter l'évaluation.

Conclusion Générale

Ce projet de fin d'étude nous a donné l'opportunité de découvrir des nouvelles extensions et connaissances de la nouvelle technologie de pointe TinyML. Nous avons exploré une méthodologie de travail en commençant par la collecte des données jusqu'au déploiement. Ce projet nous a orientés vers les solutions des limites des algorithmes de l'apprentissage automatique pour une meilleure exploitation et élargissement de zone d'application de l'intelligence artificiel dans les systèmes de faible puissance.

Le but de ce projet était sur la réalisation de la classification et la détection d'objet par les méthodes de transfert Learning et de TinyML. La classification d'image consistait à reconnaître à quelle catégorie appartient une image parmi un ensemble de catégories prédéterminées. La détection d'objet était assez méticuleuse car elle englobe trois concepts fondamentaux notamment la classification, la localisation et puis la détection.

Pour la réalisation de notre propre modèle nous avons tout d'abord commencé par la collecte de donnée. Cette étape de collecte a été mise en échelle car la terminologie du travail se base sur cette étape. La collecte des données demande beaucoup de temps car d'une part on doit faire appel à différents domaines pour éviter les mauvais étiquetages des données, d'une autre part des notions de confidentialité nous limitent sur les données à exploitées. Alors la solution que nous avons prise pour fonder ce principe est d'utiliser des données libres existantes. Pour l'entrainement de notre modèle, on a utilisé la technique de Transfert Learning qui nous à permit une simplification afin de ne pas reprendre un entrainement complet des données. Dans le cadre de TinyML on a choisi l'architecture SSD et le modèle de convolution MobileNetV2 parmi les modèles disponibles. Une fois le modèle ProtoBuffer conçu, on l'a converti en FlattBuffer grâce au Framework TensorFlow Lite pour optimiser et adapter notre réseau de neurones sur un domaine d'application adapté au TinyML.

Nous avons constaté que l'application de l'apprentissage automatique est un domaine de recherche exploitable et applicable au service de quotidienne en utilisant les techniques de Transfert Learning et TinyML. Toutes fois, l'optimisation de leurs caractéristiques reste à améliorer pour démocratiser cette nouvelle technologie.

Bibliographie

[1] Warden, P., & Situnayake, D. (2019). Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers. O'Reilly Media.

[2] Carpenter, R., & Freeman, J. (2005). Computing machinery and the individual: the personal turing test. Computing, Accessed September, 22, 2009.

[3] Valentin Blanchot, le 20 août 2018 ‘’Histoire de l’intelligence artificiel » en ligne <https://siecledigital.fr/2018/08/20/histoire-intelligence-artificielle> Mis à jour le 18 décembre 2020, consulté le 15 Avril 2021.

[4] Banbury, C. R., Reddi, V. J., Lam, M., Fu, W., Fazel, A., Holleman, J & Yadav, P. (2020). Benchmarking TinyML systems: Challenges and direction. arXiv preprint arXiv:2003.04821.

[5] Anirudh VK, le 13 Décembre 2019, “What Is Machine Learning: Definition, Types, Applications and Examples”. En ligne : <https://www.toolbox.com/tech/artificial-intelligence/tech-101/what-is-machine-Learning-definition-types-applications-and-examples/>, consulté le 15 Avril 2021.

[6] Daniel Faggella, mise à jour le 15 février 2020, “What is machine Learning”. En ligne :<https://emerj.com/ai-glossary-terms/what-is-machine-Learning/>, consulté le 16 avril 2021.

[7] Fedorov, I., Adams, R. P., Mattina, M., & Whatmough, P. N. (2019). Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. arXiv preprint arXiv:1905.12107.

[8] Anirudh VK, le 13 décembre 2019, “What Is Deep Learning: Definition, Framework, and Neural Networks”. En ligne : <https://www.toolbox.com/tech/artificial-intelligence/tech-101/what-is-deep-Learning-definition-framework-and-neural-networks/>, consulté le 20 Avril 2021

[9] Indolia, S., Goswami, A. K., Mishra, S. P., & Asopa, P. (2018). Conceptual understanding of convolutional neural network-a deep learning approach. Procedia computer science, 132, 679-688.

[10] A.I. Wiki, Pathmind “neural-network”.En ligne :< https://wiki.pathmind.com/neural-network>, consulté le 22 Avril 2021

[11] Jason Brownlee, le 22 Mai 2019, “A Gentle Introduction to Object Recognition with Deep Learning”. En ligne : <https://machineLearningmastery.com/object-recognition-with-deep-Learning/>, consulté le 24 Avril 2021

[12] Rahaman, H., Johnston, M., & Champion, E. (2021). Audio-augmented arboreality: wildflowers and language. Digital Creativity, 32(1), 22-37.

[13] Eldrogi, N., Larroque, B., Bolliet, V., & Luthon, F. Vision par ordinateur pour suivi automatique de civelles en bassin Computer vision for automatic detection and tracking of glass-eels.

[14] Pr Vijay Janapa Reddi de l’université d’Harvard. Formation en ligne: “Application of TinyML” “Fundamentals of TinyML” “Deploying TinyML”.

[15] TensorFlow .En ligne :<https://www.TensorFlow.org/>, consulté le 20 Juin 2021.

[16] Fredrik Dahlqvist, Mark Patel, Alexander Rajko, and Jonathan Shulman, le 22 Juillet 2019,” Growing opportunities in the Internet of Things” .En ligne :<https://www.mckinsey.com/industries/private-equity-and-principal-investors/our-insights/growing-opportunities-in-the-internet-of-things>, consulté le 10 Juin 2021.

[17] Tom Lombardo, le 01 Septembre 2020,” IoT Device Detects Wind Turbine Faults in the Field” .En ligne :< https://www.engineering.com/story/iot-device-detects-wind-turbine-faults-in-the-field>, consulté le 10 Juin 2021.

[18] Pranav, Hackaday,” Solar Scare Mosquito 2.0” .En ligne :< https://hackaday.io/project/174575-solar-scare-mosquito-20>

[19] Anna Solanna, le 17 Aout 2020,” Elephants vs trains: This is how AI helps ensure they don't collide” .En ligne <https://www.zdnet.com/article/elephants-vs-trains-this-is-how-ai-helps-ensure-they-dont-collide/>, consulté le 12 Juin 2021.

[20] Hughes, D., & Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics. arXiv preprint arXiv:1511.08060.

[21] Coco common objects in Context. En ligne< https://cocodataset.org/#home>, consulté le 12 Juin 2021.

[22] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).

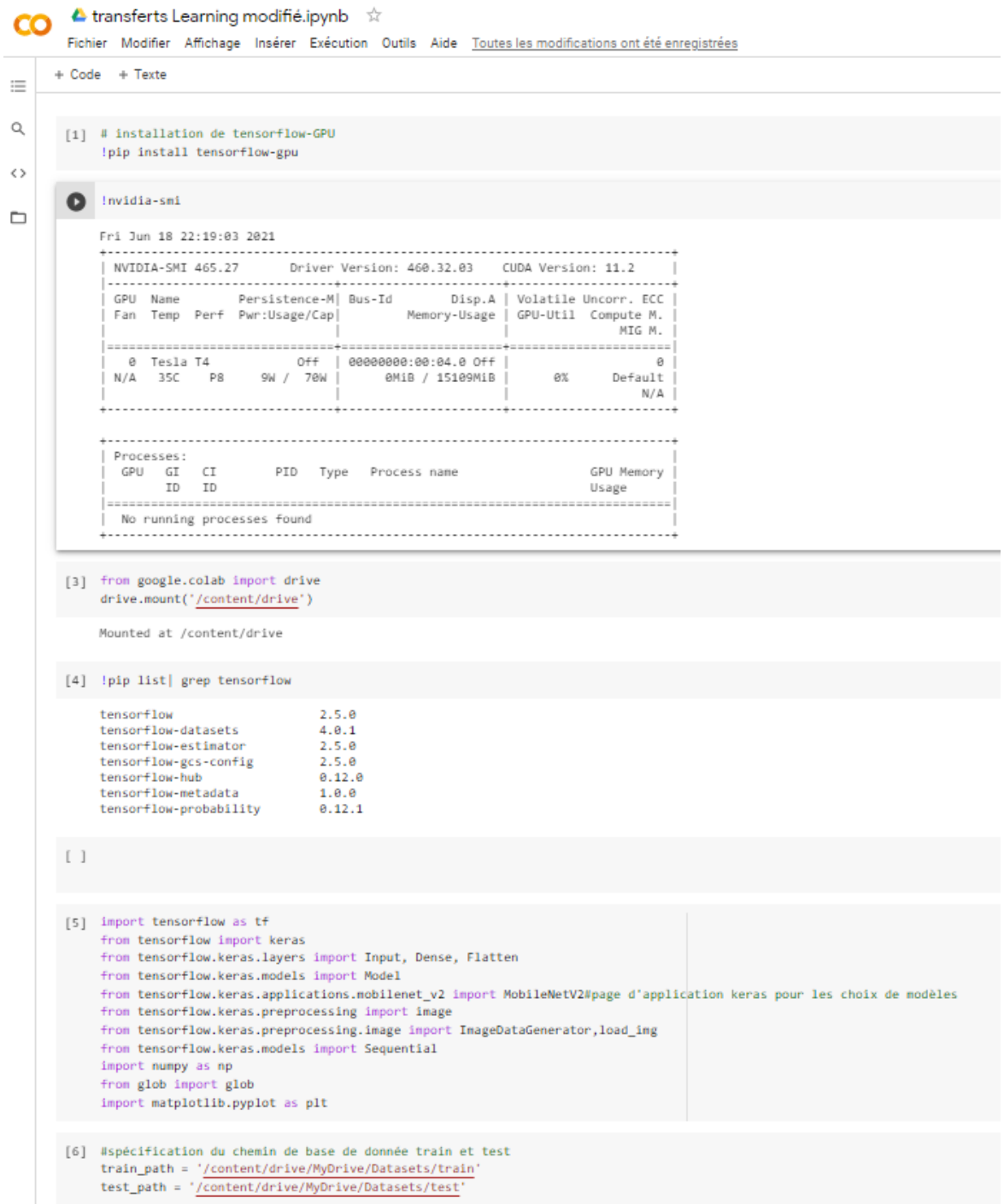
[23] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

[24] Keras application. En ligne < https://keras.io/api/applications/ >, consulté le 12 Juin 2021.

[25] Sandler et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. En ligne < https://paperswithcode.com/method/mobilenetv2

ANNEXES

Programmation dans l'interface Google Colab



```
#Spécification de la taille des images d'entrée
IMAGE_SIZE = [224, 224]
```

+ Code + Texte

```
[8] #préparation du modèle de base nommé mobileNet

    mobileNet = MobileNetV2(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9412608/9406464 [==============================] - 0s 0us/step
```

```
[ ]
```

```
[9] # utilisation des nom et le nombre de dossiers pour la classe
    folders = glob('/content/drive/MyDrive/Datasets/train/*')
    folders
```

```
['/content/drive/MyDrive/Datasets/train/Tomato___Bacterial_spot',
 '/content/drive/MyDrive/Datasets/train/Tomato___Early_blight',
 '/content/drive/MyDrive/Datasets/train/Tomato___healthy',
 '/content/drive/MyDrive/Datasets/train/Tomato___Late_blight',
 '/content/drive/MyDrive/Datasets/train/Tomato___Leaf_Mold',
 '/content/drive/MyDrive/Datasets/train/Tomato___Septoria_leaf_spot',
 '/content/drive/MyDrive/Datasets/train/Tomato___Spider_mites Two-spotted_spider_mite',
 '/content/drive/MyDrive/Datasets/train/Tomato___Target_Spot',
 '/content/drive/MyDrive/Datasets/train/Tomato___Tomato_mosaic_virus',
 '/content/drive/MyDrive/Datasets/train/Tomato___Tomato_Yellow_Leaf_Curl_Virus']
```

```
# bloqué le modèle de base
for layer in mobileNet.layers:
    layer.trainable = False

#ajout des couches de sorties
x = Flatten()(mobileNet.output)
prediction = Dense(len(folders), activation='softmax')(x)

#création du modèle
model = Model(inputs=mobileNet.input, outputs=prediction)
model.summary()
```

block_14_depthwise_BN (BatchNor	(None, 7, 7, 960)	3840	block_14_depthwise[0][0]
block_14_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_14_depthwise_BN[0][0]
block_14_project (Conv2D)	(None, 7, 7, 160)	153600	block_14_depthwise_relu[0][0]
block_14_project_BN (BatchNorma	(None, 7, 7, 160)	640	block_14_project[0][0]
block_14_add (Add)	(None, 7, 7, 160)	0	block_13_project_BN[0][0] block_14_project_BN[0][0]
block_15_expand (Conv2D)	(None, 7, 7, 960)	153600	block_14_add[0][0]
block_15_expand_BN (BatchNormal	(None, 7, 7, 960)	3840	block_15_expand[0][0]
block_15_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_15_expand_BN[0][0]
block_15_depthwise (DepthwiseCo	(None, 7, 7, 960)	8640	block_15_expand_relu[0][0]
block_15_depthwise_BN (BatchNor	(None, 7, 7, 960)	3840	block_15_depthwise[0][0]
block_15_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_15_depthwise_BN[0][0]
block_15_project (Conv2D)	(None, 7, 7, 160)	153600	block_15_depthwise_relu[0][0]
block_15_project_BN (BatchNorma	(None, 7, 7, 160)	640	block_15_project[0][0]
block_15_add (Add)	(None, 7, 7, 160)	0	block_14_add[0][0] block_15_project_BN[0][0]
block_16_expand (Conv2D)	(None, 7, 7, 960)	153600	block_15_add[0][0]
block_16_expand_BN (BatchNormal	(None, 7, 7, 960)	3840	block_16_expand[0][0]
block_16_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_16_expand_BN[0][0]
block_16_depthwise (DepthwiseCo	(None, 7, 7, 960)	8640	block_16_expand_relu[0][0]
block_16_depthwise_BN (BatchNor	(None, 7, 7, 960)	3840	block_16_depthwise[0][0]
block_16_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_16_depthwise_BN[0][0]
block_16_project (Conv2D)	(None, 7, 7, 320)	307200	block_16_depthwise_relu[0][0]
block_16_project_BN (BatchNorma	(None, 7, 7, 320)	1280	block_16_project[0][0]
Conv_1 (Conv2D)	(None, 7, 7, 1280)	409600	block_16_project_BN[0][0]
Conv_1_bn (BatchNormalization)	(None, 7, 7, 1280)	5120	Conv_1[0][0]
out_relu (ReLU)	(None, 7, 7, 1280)	0	Conv_1_bn[0][0]
flatten (Flatten)	(None, 62720)	0	out_relu[0][0]
dense (Dense)	(None, 10)	627210	flatten[0][0]

```
Total params: 2,885,194
Trainable params: 627,210
Non-trainable params: 2,257,984
```

```
[12] # compilation du modèle avec l'optimisation
     model.compile(
                    loss='categorical_crossentropy',
                    optimizer=keras.optimizers.Adam(learning_rate=1e-3, decay=1e-5),
                    metrics=['accuracy']
                  )
```

```
[13] # Générateur d'image pour la création d'image par augmentation
     from tensorflow.keras.preprocessing.image import ImageDataGenerator

     train_datagen = ImageDataGenerator(rescale = 1./255,
                                        shear_range = 0.2,
                                        zoom_range = 0.2,
                                        horizontal_flip = True)

     training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/Datasets/train',
                                                      target_size = (224, 224),
                                                      batch_size = 16,
                                                      class_mode = 'categorical')

     test_datagen = ImageDataGenerator(rescale = 1./255)

     test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/Datasets/test',
                                                 target_size = (224, 224),
                                                 batch_size = 16,
                                                 class_mode = 'categorical')
```

```
Found 18345 images belonging to 10 classes.
Found 4280 images belonging to 10 classes.
```

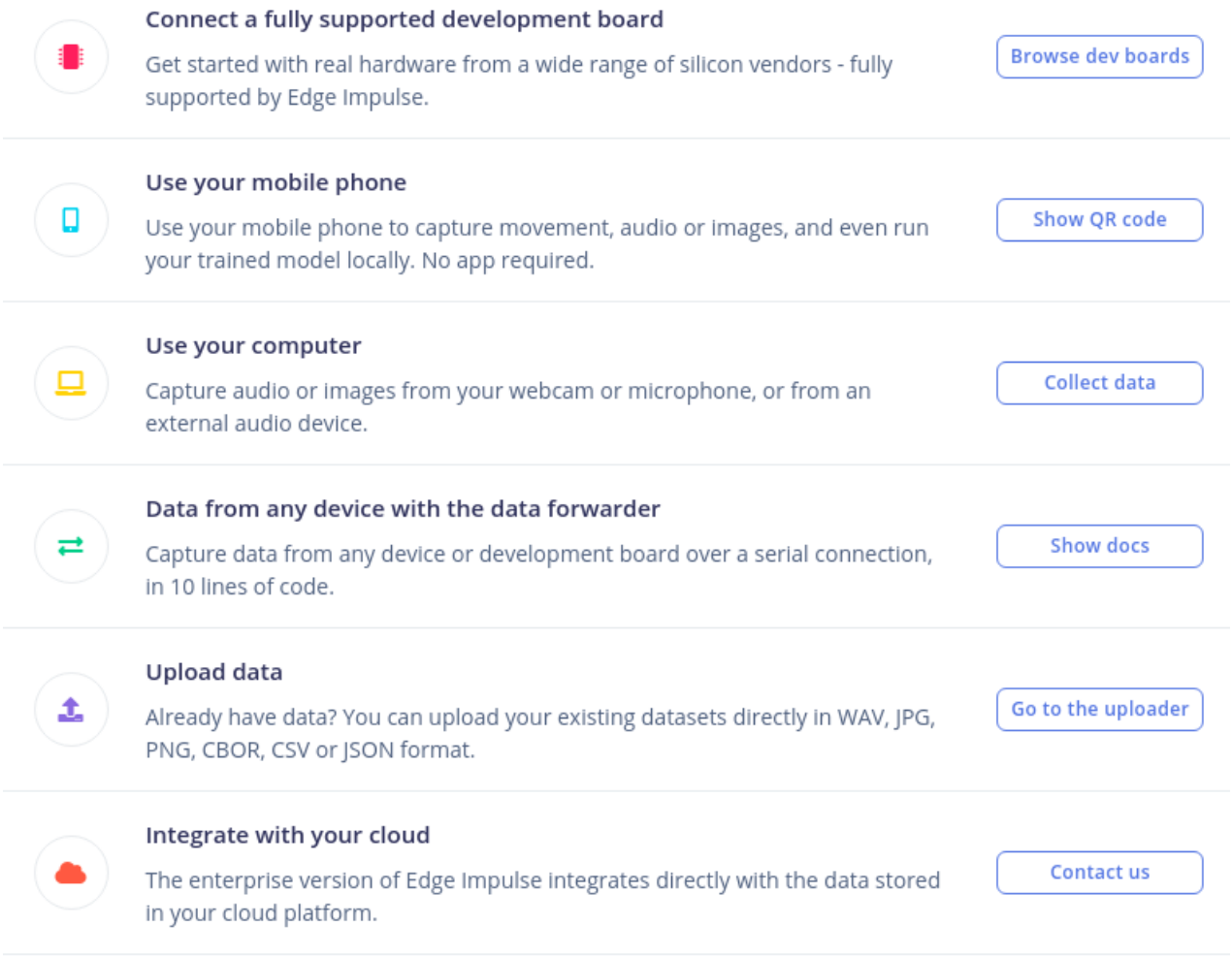
```
#entrainement du modèle
r = model.fit(
  training_set,
  validation_data=test_set,
  epochs=10,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

```
[15] # plot the loss
     import matplotlib.pyplot as plt
     plt.plot(r.history['loss'], label='train loss')
     plt.plot(r.history['val_loss'], label='val loss')
     plt.legend()
     plt.show()
     plt.savefig('LossVal_loss')

     # plot the accuracy
     plt.plot(r.history['accuracy'], label='train acc')
     plt.plot(r.history['val_accuracy'], label='val acc')
     plt.legend()
     plt.show()
     plt.savefig('AccVal_acc')
```

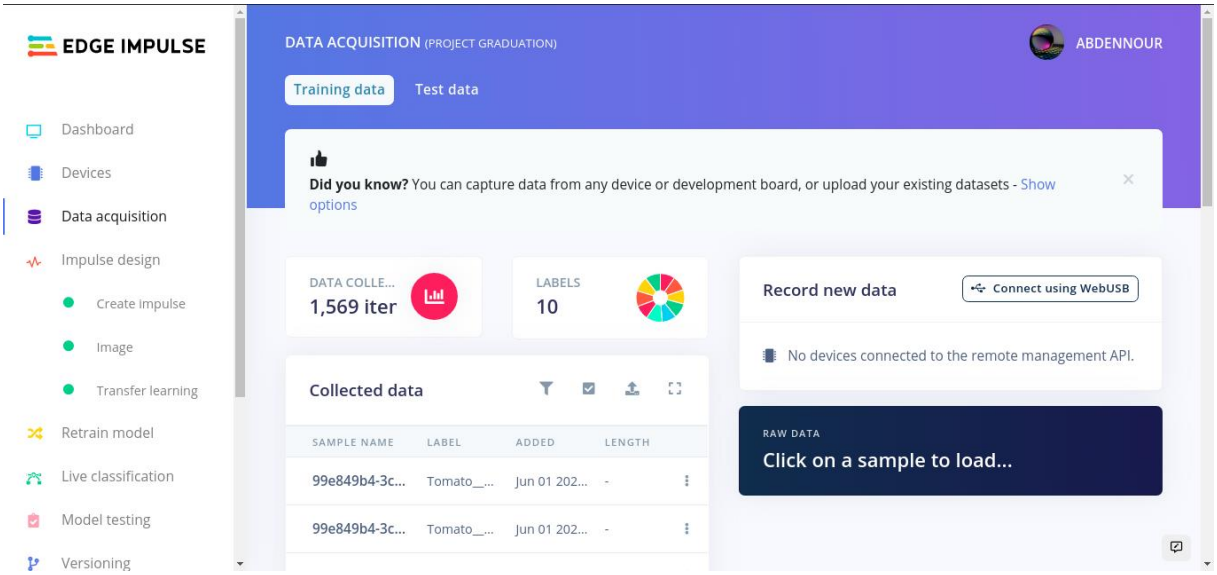
```
[16] # Sauvegarde du sortis de modèle de l'entrainement
     from tensorflow.keras.models import load_model
     model.save('model_mobilelnet.h5')

     #chargement du modèle Pb
     model1=load_model('/content/model_mobilelnet.h5')
     # Convertion du modèle
     converter = tf.lite.TFLiteConverter.from_saved_model(model1) #
     tflite_model = converter.convert()

     # auvegarde du modèle converti
     with open('model.tflite', 'wb') as f:
       f.write(tflite_model)
```

Edge Impulse :

Les Différentes possibilités pour la collecte de donnée sur Edge Impulse



Interface de l'acquisition des données

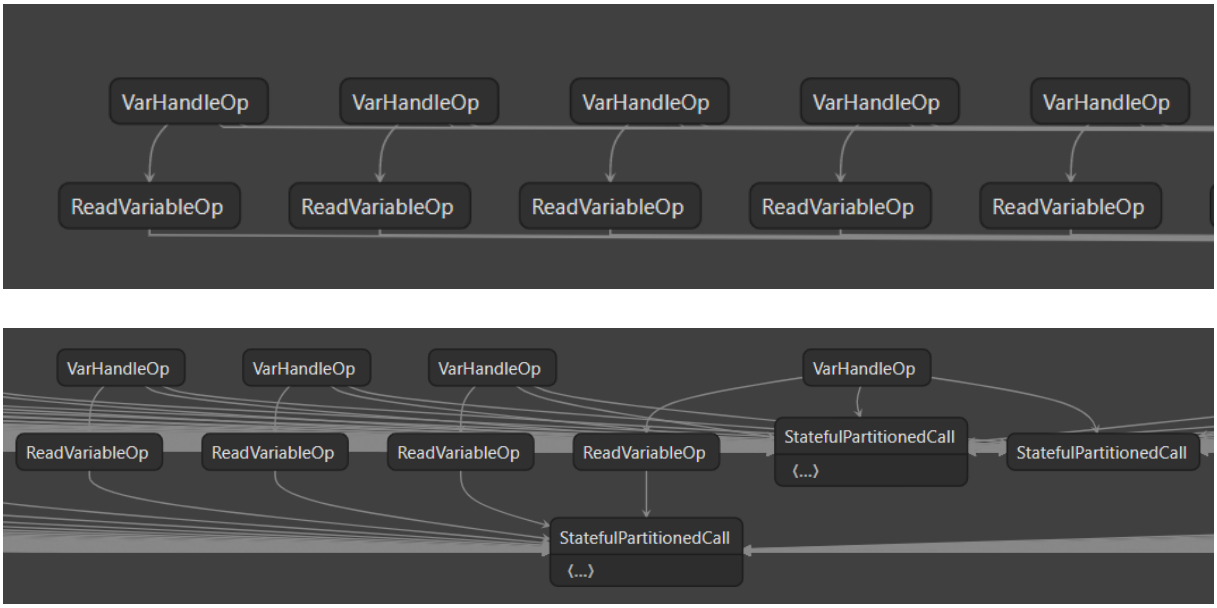


Programme pour l'entrainement de modèle dans l'Interface Edge Impulse :

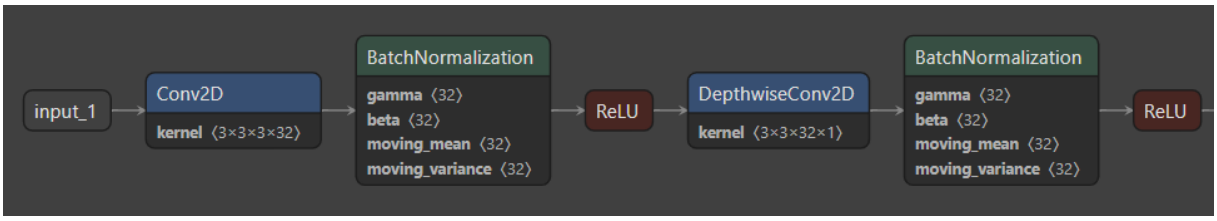
```
import math
from pathlib import Path
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer, Dropout, Conv1D, Flatten, Reshape, MaxPooling1D, BatchNormalization, Conv2D, GlobalMaxPooling2D, Lambda
from tensorflow.keras.optimizers import Adam, Adadelta
from tensorflow.keras.losses import categorical_crossentropy
sys.path.append('./resources/libraries')
import ei_tensorflow.training

WEIGHTS_PATH = './transfer-learning-weights/keras/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_0.35_96.h5'

INPUT_SHAPE = (256, 256, 3)

base_model = tf.keras.applications.MobileNetV2(
    input_shape = INPUT_SHAPE, alpha=0.35,
    weights = WEIGHTS_PATH
)

base_model.trainable = False

model = Sequential()
model.add(InputLayer(input_shape=INPUT_SHAPE, name='x_input'))
# Don't include the base model's top layers
last_layer_index = -3
model.add(Model(inputs=base_model.inputs, outputs=base_model.layers[last_layer_index].output))
model.add(Dense(16))
model.add(Dropout(0.1))
model.add(Flatten())
model.add(Dense(classes, activation='softmax'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

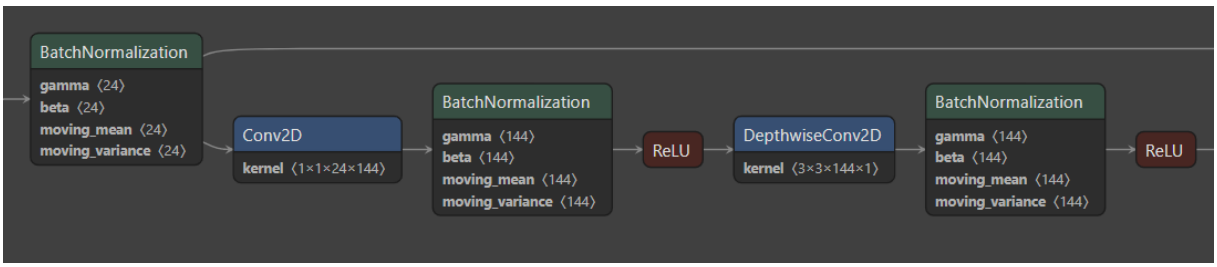
```
BATCH_SIZE = 32
train_dataset = train_dataset.batch(BATCH_SIZE, drop_remainder=False)
validation_dataset = validation_dataset.batch(BATCH_SIZE, drop_remainder=False)
callbacks.append(BatchLoggerCallback(BATCH_SIZE, train_sample_count))
model.fit(train_dataset, validation_data=validation_dataset, epochs=100, verbose=2, callbacks=callbacks)

print('')
print('Initial training done.', flush=True)

# How many epochs we will fine tune the model
FINE_TUNE_EPOCHS = 50
# What percentage of the base model's layers we will fine tune
FINE_TUNE_PERCENTAGE = 65

print('Fine-tuning best model for {} epochs...'.format(FINE_TUNE_EPOCHS), flush=True)
# Load best model from initial training
model = ei_tensorflow.training.load_best_model(BEST_MODEL_PATH)

# Determine which layer to begin fine tuning at
model_layer_count = len(model.layers)
fine_tune_from = math.ceil(model_layer_count * ((100 - FINE_TUNE_PERCENTAGE) / 100))

# Allow the entire base model to be trained
model.trainable = True
# Freeze all the layers before the 'fine_tune_from' layer
for layer in model.layers[:fine_tune_from]:
    layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.000045),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_dataset,
              epochs=FINE_TUNE_EPOCHS,
              verbose=2,
              validation_data=validation_dataset,
              callbacks=callbacks)
```

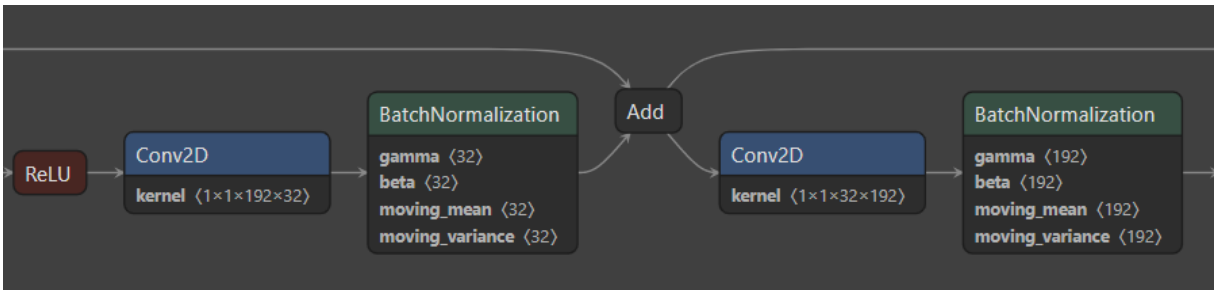
Extrait de couche dans les réseaux de neurones formé à partir de l'outil Neutron :

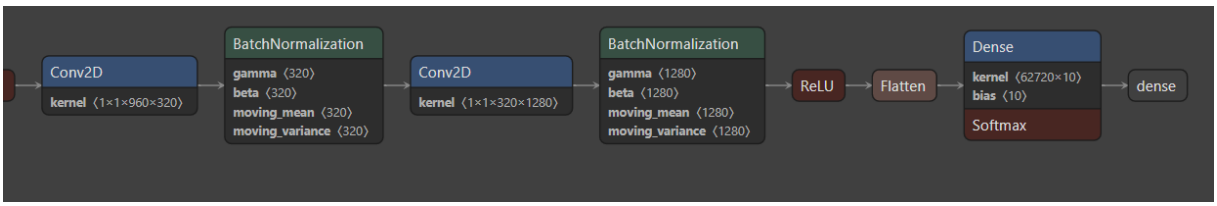



Const
Const
Const
Const
Const
Const
Const
NoOp
serving_default_input_tensor
Placeholder
ReadVariableOp


input_1
Conv2D
kernel ⟨3×3×3×32⟩
BatchNormalization
gamma ⟨32⟩
beta ⟨32⟩
moving_mean ⟨32⟩
moving_variance ⟨32⟩
ReLU
DepthwiseConv2D
kernel ⟨3×3×32×1⟩
BatchNormalization
gamma ⟨32⟩
beta ⟨32⟩
moving_mean ⟨32⟩
moving_variance ⟨32⟩
ReLU


BatchNormalization
gamma ⟨24⟩
beta ⟨24⟩
moving_mean ⟨24⟩
moving_variance ⟨24⟩
Conv2D
kernel ⟨1×1×24×144⟩
BatchNormalization
gamma ⟨144⟩
beta ⟨144⟩
moving_mean ⟨144⟩
moving_variance ⟨144⟩
ReLU
DepthwiseConv2D
kernel ⟨3×3×144×1⟩
BatchNormalization
gamma ⟨144⟩
beta ⟨144⟩
moving_mean ⟨144⟩
moving_variance ⟨144⟩
ReLU


ReLU
Conv2D
kernel ⟨1×1×192×32⟩
BatchNormalization
gamma ⟨32⟩
beta ⟨32⟩
moving_mean ⟨32⟩
moving_variance ⟨32⟩
Add
Conv2D
kernel ⟨1×1×32×192⟩
BatchNormalization
gamma ⟨192⟩
beta ⟨192⟩
moving_mean ⟨192⟩
moving_variance ⟨192⟩


Conv2D
kernel ⟨1×1×960×320⟩
BatchNormalization
gamma ⟨320⟩
beta ⟨320⟩
moving_mean ⟨320⟩
moving_variance ⟨320⟩
Conv2D
kernel ⟨1×1×320×1280⟩
BatchNormalization
gamma ⟨1280⟩
beta ⟨1280⟩
moving_mean ⟨1280⟩
moving_variance ⟨1280⟩
ReLU
Flatten
Dense
kernel ⟨62720×10⟩
bias ⟨10⟩
Softmax
dense