



RÉPUBLIQUE DU BÉNIN
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ D'ABOMEY-CALAVI

INSTITUT DE FORMATION ET DE
RECHERCHE EN INFORMATIQUE



BP 526 Cotonou Tel : +229 21 14 19 88
<http://www.ifri-uac.bj> Courriel : contact@ifri.uac.bj

MÉMOIRE

pour l'obtention du

Diplôme de Master en Informatique

Option : Systèmes d'Information et Réseaux Informatiques

Présenté par :

Auspicia Honorine Arènikè DJIMA

Système d'autoguidage pour les malvoyants

Sous la supervision :

Dr. Ing John AOGA, Enseignant à IFRI

Membres du jury :

Pr Eugène C. EZIN

Professeur Titulaire du CAMES

IFRI,IMSP/UAC

Président

Dr. Msc. Ratheil HOUNDJI

Professeur Assistant du CAMES

IFRI/UAC

Examinateur

Dr. Msc. John AOGA

Enseignant Chercheur

IFRI/UAC

Rapporteur

Sommaire

Dédicace	ii
Remerciements	iii
Résumé	iv
Abstract	v
Table des figures	vi
Liste des tableaux	viii
Liste des acronymes	ix
Introduction	1
1 Revue de littérature	3
2 Matériel et méthodes	21
3 Résultats et discussions	37
Conclusion	48
Bibliographie	49
Annexe	53
Table des matières	72

Dédicace

À MES CHERS PARENTS

Remerciements

Je tiens à remercier ici :

- l'Eternel Dieu tout puissant de m'avoir donné le courage, la force et la patience d'achever ce modeste travail;
- mes très chers parents pour les sacrifices que vous avez consentis pour mon instruction et mon bien-être. Que ce modeste travail soit l'exaucement de vos vœux tant formulés, le fruit de vos innombrables sacrifices;
- le Professeur Eugène C. Ezin, Directeur de l'Institut de Formation et de Recherche en Informatique (IFRI);
- Dr Ing John AOGA, encadreur de ce mémoire, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion et ses encouragements lors de la réalisation de ce mémoire;
- chacun des professeurs m'ayant encadrée le long de ma formation à l'IFRI, pour m'avoir donné les enseignements nécessaires pouvant me permettre de mieux me lancer dans le monde professionnel et de la recherche;
- à Johannès HOUNSINOU, Ingénieur en Génie électrique et Responsable du laboratoire d'Intelligence Artificielle de la bibliothèque Bénin Excellence et à Roméo LONTCHEDJI pour avoir été mon encadreur durant tout mon séjour au laboratoire;
- à mon beau-frère ABOU Aziz et ma très chère grande sœur Hermione DJIMA sans qui je n'aurais pu commencer ce Master, merci d'avoir cru en moi;
- à mon bien-aimé Placide N. HOUNHOUI, merci pour les conseils, encouragements et soutien constants et indéfectibles dont tu as fait montre tout le long de ce travail et de cette formation;
- à tous mes camarades étudiants de l'IFRI, avec qui j'ai passé deux années de partage dans une ambiance conviviale.

Résumé

Le déplacement constitue un véritable obstacle pour les personnes malvoyantes, impactant significativement leur capacité à se déplacer librement et à maintenir leur autonomie. Malgré les progrès technologiques considérables, les systèmes de navigation actuels souffrent de limitations en termes de détection en temps réel et d'interaction avec l'environnement, ce qui réduit leur efficacité dans des environnements complexes et en constante évolution. Une approche intégrée de la vision par ordinateur pourrait être une solution à ce problème. L'objectif de notre projet est donc de concevoir un système intelligent d'autoguidage pour malvoyants. Pour atteindre cet objectif, notre stratégie est basée sur trois aspects principaux : la conception d'un modèle de détection d'objets et de leur distance par rapport à la caméra, permettant de prioriser les obstacles proches et d'améliorer la sécurité des utilisateurs ; la conception d'un prototype expérimental composé d'une caméra, et d'un haut-parleur, le tout contrôlé par un système intelligent embarqué sur une Raspberry Pi. Ce système intelligent est constitué d'un modèle pré-entraîné MobileNetV2 SSD FPN, et d'un module de synthèse vocale basé sur gTTS (Google Text-to-Speech), permettant ainsi au système de fournir des instructions audio claires et concises aux utilisateurs. Enfin, l'ensemble du système est testé. Les résultats obtenus nous montrent une précision de 95,46% sur l'ensemble d'entraînement, 96,20% sur celui de validation et 98,75% sur celui de test. Ce travail vise à contribuer à l'évolution des technologies d'assistance pour les personnes malvoyantes en proposant une solution innovante et efficace.

Mots clés : malvoyants, autoguidage, vision par ordinateur, Raspberry Pi, MobileNetV2 SSD FPN, gTTS (Google Text-to-Speech).

Abstract

The movement poses a real challenge for the visually impaired, significantly affecting their mobility and independence. Despite remarkable technological advancements, existing navigation systems lack real-time object detection and interaction capabilities, thus limiting their effectiveness in complex and ever-changing environments. An integrated approach of computer vision could offer a solution to this issue. Therefore, the aim of our project is to design an intelligent guidance system for visually impaired individuals. To achieve this goal, our strategy is based on three main aspects: the development of an object detection model and its distance from the camera to prioritize nearby obstacles and enhance user safety; the creation of an experimental prototype comprising a camera, a speaker, and a microphone, all controlled by an embedded intelligent system on a Raspberry Pi. This intelligent system consists of a pre-trained MobileNetV2 SSD FPN model and a gTTS (Google Text-to-Speech) based speech synthesis module, enabling the system to provide clear and concise audio instructions to users. Finally, the entire system is tested, with results showing a precision of 95.46% on the training set, 96.20% on the validation set, and 98.75% on the test set. This work aims to contribute to the evolution of assistive technologies for visually impaired individuals by proposing an innovative and effective solution.

Key words: visually impaired, auto-guidance, computer vision, Raspberry Pi, MobileNetV2 SSD FPN, gTTS (Google Text-to-Speech).

Table des figures

1.1 Réseau de neurones artificiels / convolutifs / impulsionnels	4
1.2 L'intelligence artificielle et ses sous-domaines	4
1.3 Architecture d'un réseau de neurones artificiel	5
1.4 Les deux étapes de convolution séparable utilisées dans MobileNet V1	7
1.5 Les deux blocs de MobileNet V2	8
1.6 Schema fonctionnel d'un systeme TTS	9
1.7 Module NLP d'un système de synthèse vocale	10
1.8 Phonétisation basée sur un dictionnaire (à gauche) ou basée sur des règles (à droite) .	11
1.9 Différents types d'informations fournies par l'intonation (les lignes indiquent les mouvements de hauteur; l'accent solide)	12
1.10 Design du système proposé [20]	14
1.11 Schéma fonctionnel de base de la conception	15
1.12 Détection des mouvements ascendants et descendants dans les escaliers	16
1.13 Détection de flaques d'eau/boue	16
1.14 Architecture globale du système	17
1.15 Schéma du système proposé	18
2.1 L'architecture de TensorFlow Lite	23
2.2 Schéma du système proposé	25
2.3 Organigramme de Fonctionnement du Système	26
2.4 Architecture du SSD-MobileNet-v2	30
2.5 Topologie complète d'un modèle EfficientDet	31
2.6 Architecture du SSD-Mobilenet-v2-fpnlite	33
3.1 TensorBoard d'entraînement du modèle SSD-Mobilenet-v2-fpnlite-320 après collecte supplémentaire	41
3.2 TensorBoard d'entraînement du modèle SSD-Mobilenet-v2 après collecte supplémentaire	43
3.3 TensorBoard d'entraînement du modèle EfficientDet-D0 après collecte supplémentaire	44
3.4 Détection d'une personne à 1,27 m	45
3.5 Détection d'une Moto à 4,39 m avant la collecte supplémentaire	45
3.6 Détection d'une Moto à 5,68 m après collecte supplémentaire	45
3.7 Détection d'une Voiture à 2,678 m avant la collecte supplémentaire	46
3.8 Détection d'une Voiture à 3,03 m après collecte supplémentaire	46
3.9 Prototype du système vue de face	47
3.10 Prototype du système vu d'en haut	47
3.11 Image d'une Raspberry avec ses caractéristiques	53

3.12 Image d'un ESP32 devkit v1	54
3.13 Image d'un jumper male-femelle	54
3.14 Image d'une caméra Raspberry Pi Rev v1.3	55
3.15 Installer les Dépendances pour la Détection d'Objets avec TensorFlow	56
3.16 Installer les Dépendances pour la Détection d'Objets avec TensorFlow	56
3.17 Téléchargez l'ensemble de données d'image et préparez à la formation	57
3.18 Téléchargez l'ensemble de données d'image et préparez à la formation	57
3.19 Téléchargez l'ensemble de données d'image et préparez à la formation	57
3.20 Téléchargez l'ensemble de données d'image et préparez à la formation	58
3.21 Téléchargez l'ensemble de données d'image et préparez à la formation	58
3.22 Configuration de l'entraînement du modèle	59
3.23 Configuration de l'entraînement du modèle	59
3.24 Configuration de l'entraînement du modèle	60
3.25 Configuration de l'entraînement du modèle	60
3.26 Entraîner un modèle	61
3.27 Convertir le modèle en TensorFlow Lite	61
3.28 Testez le modèle TensorFlow Lite et calculez mAP	62
3.29 Testez le modèle TensorFlow Lite et calculez mAP	63
3.30 Testez le modèle TensorFlow Lite et calculez mAP	64
3.31 Testez le modèle TensorFlow Lite et calculez mAP	64
3.32 Deploiement du modèle sur la Raspberry Pi	65
3.33 Deploiement du modèle sur la Raspberry Pi	65
3.34 Deploiement du modèle sur la Raspberry Pi	66
3.35 Deploiement du modèle sur la Raspberry Pi	66
3.36 Deploiement du modèle sur la Raspberry Pi	67
3.37 Deploiement du modèle sur la Raspberry Pi	68
3.38 Définition du socket pour écoute des connexions entrantes sur le Raspberry Pi	68
3.39 Connexion de l'ESP32 à l'IoT Raspberry Pi	69
3.40 Implémentation du module TTS	69
3.41 Script shell de récupération des flux vidéo et lancement de la détection dès le démarrage du Raspberry Pi	70
3.42 Ajout du fichier dans crontab	70

Liste des tableaux

2.1	Liste des matériaux utilisés	21
2.2	Liste des matériaux utilisés (Suite)	22
2.3	Critère de sélection des types d'architecture CNN [39], [40]	29
2.4	Critères de sélection des détecteurs d'objets [42]	30
2.5	Définition des paramètres d'entraînement des modèles	34
2.6	Matrice de confusion d'un problème de classification binaire	34
2.7	Matrice de confusion multi-classe pour 3 classes	34
3.1	Liste des classes d'objets de la dataset	38
3.2	Liste des classes d'objets de la dataset (Suite)	39
3.3	Précision et perte du modèle SSD-Mobilenet-v2-fpnlite-320 avant collecte supplémentaire	40
3.4	Précision et perte du modèle SSD-Mobilenet-v2-fpnlite-320 après collecte supplémentaire	40
3.5	Précision et perte du modèle SSD-Mobilenet-v2 avant collecte supplémentaire	42
3.6	Précision et perte du modèle SSD-Mobilenet-v2 après collecte supplémentaire	42
3.7	Précision et perte du modèle EfficientDet-D0 avant collecte supplémentaire	43
3.8	Précision et perte du modèle EfficientDet-D0 après collecte supplémentaire	43

Liste des acronymes

API :

Application Programming Interface

CNN :

Convolutional Neural Network

COCO :

Common Objects in Context

CPU :

Central Processing Unit

CUDA :

Compute Unified Device Architecture

DMIPS :

Dhrystone Millions of Instructions Per Second

DMLA :

Dégénérescence Maculaire Liée à l'Age

DSP :

Digital Signal Processing

FC :

Fully Connected

FPN :

Feature Pyramid Network

GPS :

Global Positioning System

GPU :

Graphics Processing Unit

GSM :

Global System for Mobile Communications

gTTS :

Google Text-to-Speech

gTTS :

Google Text-to-Speech

IA :

Artificial Intelligence

IoT :

Internet of Things

IT :

Information Technology

MACs :

Multiplications-Accumulations

NLP :

Natural Language Processing

OCR :

Optical Character Recognition

OpenCV :

Open Source Computer Vision Library

RCNN :

Region-Based Convolutional Neural Network

ReLU :

Rectified Linear Unit

RVB :

Rouge-Vert-Bleu

SDRAM :

Static Random Access Memory

SMS :

Short Message Service

SPPNet :

Spatial Pyramid Pooling Networks

SVM :

Support Vector Machines

TTS :

Text-to-Speech

USB :

Universal Serial Bus

VGGNet :

Visual Geometry Group Network

XML :

eXtensible Markup Language

YOLO :

You Only Look Once

Introduction Générale

1. Problématique

Malgré les avancées technologiques récentes, l'aide à la mobilité des déficients visuels demeure limitée. Les solutions existantes actuellement manquent de possibilité de détection d'objets en temps réel et d'interaction, sont limitées en termes de précision et de rapidité de détection des obstacles. Cela compromet la sécurité de ces personnes lors de leurs déplacements, limitant ainsi leur efficacité dans un environnement complexe et en constante évolution. Dès lors, il apparaît nécessaire de mettre à leur disposition un système d'autoguidage adapté, capable de détecter les obstacles en temps réel et de leur fournir des instructions précises pour se déplacer de façon sécurisée. Un tel système contribuerait à renforcer leur autonomie et les aiderait à mieux s'intégrer socialement.

2. Contexte

La routine quotidienne, souvent considérée comme insignifiante par la plupart, représente en réalité un défi de survie pour certaines personnes, en particulier les malvoyants. Dans une société qui a malheureusement tendance à mettre de côté ces individus, la réalisation de tâches simples devient un véritable combat. Parmi les difficultés spécifiques rencontrées par ces derniers, on relève celles liées aux déplacements, accès à l'information et aux services. En effet, la très grande majorité des déficients visuels vivent en domicile ordinaire ; 16% des aveugles et malvoyants profonds, 12% des malvoyants moyens et moins de 3% des malvoyants légers vivent en institution [1]. Ceux qui ont la possibilité de rester chez eux disposent d'une personne pouvant les aider dans leur mobilité et leurs tâches quotidiennes. Cette situation engendre une perte d'autonomie et un risque accru d'exclusion sociale. Cependant, malgré les avancées technologiques récentes, l'aide à la mobilité des déficients visuels demeure toujours un challenge.

3. Objectifs

L'objectif principal de cette étude est de développer un système d'assistance destiné aux malvoyants. Ce système sera capable d'explorer l'environnement en temps réel en récupérant des flux vidéo, de les analyser, et de communiquer des informations pertinentes à l'utilisateur.

Les objectifs spécifiques de ce travail sont les suivants :

- mettre en place un mécanisme de collecte et de traitement des données : collecter des images sur terrain et sur internet afin de former un dataset convenable ;
- concevoir un modèle de détection d'objets et de leur distance par rapport à la caméra : implémenter un modèle de détection à longue portée, opérant en temps réel, permettant la détection d'objets dans l'environnement d'un individu malvoyant ;
- concevoir un prototype expérimental composé d'une caméra, d'un haut-parleur et d'un microphone, le tout contrôlé par un système intelligent embarqué sur une Raspberry Pi ;
- tester l'ensemble du système : mesurer l'efficacité du système en termes de rapidité des réponses, rapidité d'exécution en utilisant des mesures appropriées.

5. Organisation du mémoire

Ce mémoire s'articule autour de trois (3) chapitres :

- **Revue de littérature** : C'est le premier chapitre de ce document. Il part des différentes définitions, l'explication de certains mots ou groupes de mots nécessaires à la compréhension du thème de notre étude. Ensuite, nous faisons la présentation de quelques publications et travaux réalisés portant sur la même thématique. Enfin, nous portons un regard critique sur l'existant ;
- **Matériel et méthodes** : C'est le deuxième chapitre de notre document. Nous y présentons en détail l'architecture que nous comptons mettre en place au terme de cette étude. Tout en passant par la présentation des matériels et méthodes utilisés pour sa réalisation ;
- **Résultats et discussions** : Il s'agit du troisième et dernier chapitre de notre mémoire. Dans ce chapitre, nous parlons des difficultés rencontrées dans la réalisation de cette étude. Ensuite, nous présentons les résultats obtenus. Ces résultats sont suivis des perspectives qui pourront être explorées pour de futures études.

Revue de littérature

Introduction

L'état de l'art constitue une étape utile et importante dans une étude. Elle vise à présenter les travaux antérieurs pertinents, les avancées récentes, les lacunes dans la recherche actuelle et à fournir une base pour le développement ultérieur de nouvelles idées ou de nouvelles recherches. Nous commençons donc ce chapitre en définissant quelques thèmes autour desquels tourne notre étude. Nous présentons ensuite une vue d'ensemble du système d'auto-guidage pour les malvoyants. Nous finissons par l'étude et la critique de l'existant.

1.1 Vue d'ensemble sur la détection d'objets

1.1.1 Apprentissage profond

1.1.1.1 Généralités sur les réseaux de neurones artificiels

Un réseau neuronal est un modèle computationnel inspiré par le fonctionnement du cerveau humain, utilisé en intelligence artificielle pour analyser les données. Comme le montre la Figure 1.1 il se compose de plusieurs couches de neurones interconnectés, où chaque neurone traite et transmet des informations à travers des connexions pondérées. Ces réseaux sont généralement composés de multiples couches, telles que les couches d'entrée, cachées et de sortie. Les données sont propagées à travers le réseau, avec chaque couche recevant les sorties de la couche précédente, jusqu'à ce que les résultats finaux soient produits par la couche de sortie¹ [2], [3].

¹<https://eldorhaan.wordpress.com/2019/04/06/reseau-de-neurones-artificiels-quest-ce-que-c-est-et-a-quoi-ca-sert/>

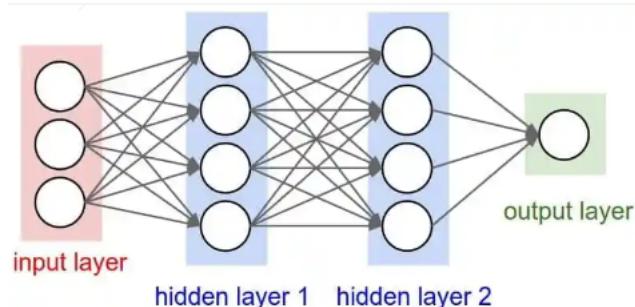


FIGURE 1.1 : Réseau de neurones artificiels / convolutifs / impulsionnels

1.1.1.2 Apprentissage profond : Deep learning

Le deep learning ou apprentissage profond est un sous-domaine du machine learning ou appren-tissage machine, sous-domaine de l'intelligence artificielle² (voir Figure 1.2), consistant à créer des systèmes capables d'apprendre, prévoir et décider en toute autonomie. Cette forme d'intelligence artificielle fonctionne avec des algorithmes capables d'imiter le cerveau humain grâce à un large réseau de neurones artificiels [4], [5], [6].

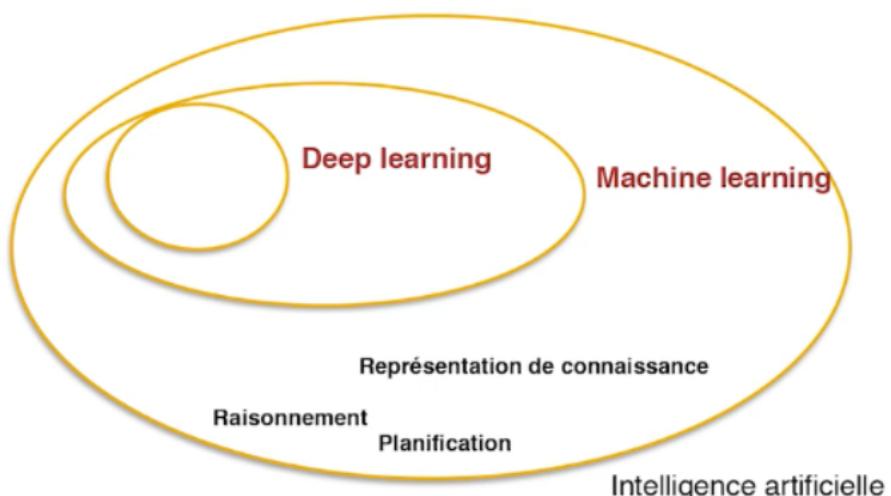


FIGURE 1.2 : L'intelligence artificielle et ses sous-domaines

1.1.1.3 Classification d'images par les réseaux de neurones convolutifs

Les réseaux de neurones convolutionnels, désignés par l'acronyme CNN (voir Figure 1.3), sont aujourd'hui les réseaux de neurones les plus performants pour classer des images. Ils se composent de deux parties distinctes.

D'une part, la partie convulsive agit comme un extracteur de caractéristiques des images en passant l'image à travers une succession de filtres ou noyaux de convolution. Ces filtres génèrent de nouvelles images appelées cartes de convolutions et certains réduisent la résolution de l'image par une opération de maximum local. Les cartes de convolutions sont ensuite mises à plat et concaténées en un vecteur de caractéristiques appelé code CNN³.

²<https://culturesciencesphysique.ens-lyon.fr/IA-apprentissage-Rousseau.xml>

³<https://www.scirp.org/journal/paperinformation.aspx?paperid=115011>

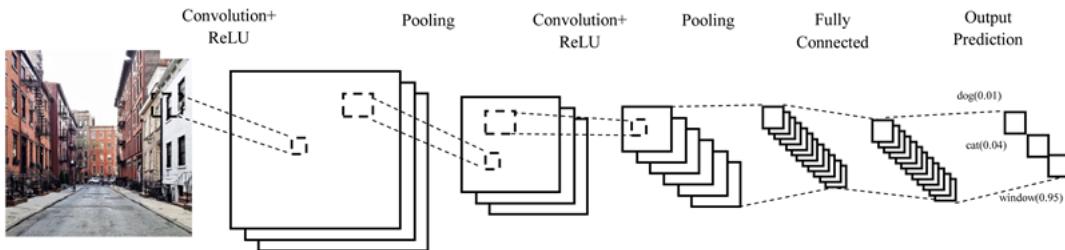


FIGURE 1.3 : Architecture d'un réseau de neurones artificiel

D'autre part, ce code CNN en sortie de la partie convulsive est ensuite utilisé comme entrée dans une deuxième partie constituée de couches entièrement connectées également appelées perceptron multicouche. Cette partie a pour rôle de combiner les caractéristiques extraites par le code CNN pour classer l'image. La sortie de cette partie est une dernière couche comportant un neurone par catégorie, produisant des valeurs numériques généralement normalisées entre 0 et 1 pour obtenir une distribution de probabilités sur les catégories.

Il existe plusieurs architectures de CNN, telles que LeNet, AlexNet, VGGNet, GoogLeNet (Inception), ResNet, EfficientNet et notamment MobileNet [7] [8], chacune adaptée à des besoins spécifiques en termes de précision, de vitesse ou de contraintes matérielles. Une architecture CNN est formée par un empilement de couches de traitement indépendantes comprenant la couche de convolution (CONV), la couche de pooling (POOL), la couche de correction ReLU, la couche entièrement connectée (Fully-Connected FC) et la couche de perte (LOSS).

1.1.1.4 Apprentissage par transfert avec les réseaux de neurones convolutifs

Le Transfer Learning (ou apprentissage par transfert) permet de faire du Deep Learning sans nécessiter des ressources de calcul intensives. Le principe est d'utiliser les connaissances acquises par un réseau de neurones lors de la résolution d'un problème afin d'en résoudre un autre plus ou moins similaire. Il existe plusieurs stratégies de transfer learning dont :

- **Stratégie 1 - Fine-tuning total** : La dernière couche fully-connected du réseau pré-entraîné est remplacée par un classifieur adapté au nouveau problème (SVM, régression logistique...) et initialisée aléatoirement. Ensuite, toutes les couches sont entraînées sur les nouvelles images. Cette stratégie est utilisée lorsque la nouvelle collection d'images est grande, permettant d'entraîner tout le réseau sans risque d'overfitting (**Voir Annexe 9 pour plus de détails**).
- **Stratégie 2 - Extraction des features** : Cette stratégie utilise les features du réseau pré-entraîné pour représenter les images du nouveau problème. La dernière couche fully-connected est retirée, et les autres paramètres sont fixés. Ensuite, un classifieur est entraîné sur ces représentations pour résoudre le nouveau problème. Elle est adaptée aux petits ensembles de données similaires aux données d'entraînement, minimisant ainsi le risque de surapprentissage.
- **Stratégie 3 - fine-tuning partiel** : Cette approche combine les stratégies 1 et 2 en remplaçant la dernière couche fully-connected par un nouveau classifieur initialisé aléatoirement tout en fixant les paramètres de certaines couches du réseau pré-entraîné. Les couches non-fixées, généralement les plus hautes du réseau, sont ensuite entraînées sur les nouvelles images, adaptées à un petit ensemble de données très différent de celui de l'entraînement initial. Cette stratégie offre un compromis efficace entre l'overfitting et la représentation des features.

1.1.1.5 Classification des algorithmes de détection d'objets

Ces dernières années, avec le développement de l'apprentissage profond, les algorithmes de détection d'objets ont réalisé de grandes avancées [9]. Les méthodes de détection d'objets populaires actuelles peuvent être divisées en deux catégories que sont : [10], [11].

1. Les détecteurs à deux étages (two-stage object detection algorithms) : La détection d'objets se fait en deux étapes distinctes, la création de zones d'intérêt et la classification finale avec la régression de la boîte englobante des objets. RCNN, Fast RCNN, SPPNet, Faster RCNN, etc., sont quelques-uns des détecteurs à deux étages [12]
2. Les détecteurs à un étage (one-stage object detection algorithms) : La détection d'objet est traitée comme un problème de régression simple qui prend des entrées et apprend les probabilités de classe ainsi que les coordonnées de la boîte englobante. YOLO, YOLO v2, SSD, RetinaNet, etc., sont quelques-uns des détecteurs à un étage [13]

1.1.2 Architecture CNN MobileNets

Les séries de réseaux MobileNets ont été développées pour répondre à un besoin crucial dans le domaine de la vision par ordinateur : l'optimisation des architectures de réseaux de neurones convolutifs (CNN) pour une utilisation efficace sur des appareils mobiles et d'autres plates-formes à ressources limitées [14], [15]. Les réseaux de neurones convolutionnels simples souffrent de deux inconvénients majeurs : une grande quantité de paramètres, conduisant au surajustement dans de nombreux cas, et une consommation élevée en calculs les rendant inadaptés aux applications mobiles. Dans ce contexte, Google a introduit les séries de réseaux MobileNets dont MobileNet-V1 (depth-wise separable convolution) et MobileNet-V2 (inverted residual blocks) visant à surmonter ces défis en proposant des architectures optimisées pour une utilisation mobile.

1.1.2.1 MobileNet V1 (depth-wise separable convolution)

L'architecture MobileNet V1 à la Figure 1.4 utilise un réseau de neurones convolutifs profonds basée sur des convolutions séparables en profondeur et en largeur, remplaçant les convolutions traditionnelles par des convolutions séparables divisées en deux étapes : une spatiale suivie d'une en profondeur. Cette structure en couches est composée de plusieurs couches de convolution, de normalisation et d'activation ReLU, avec pour objectif de réduire la dimensionnalité des données et le nombre de paramètres grâce à des convolutions séparables et d'une factorisation en profondeur pour une charge computationnelle réduite. Cette conception allégée rend MobileNet V1 adapté aux appareils mobiles et embarqués, offrant une meilleure efficacité sans compromettre les performances⁴ [14].

⁴<https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69>

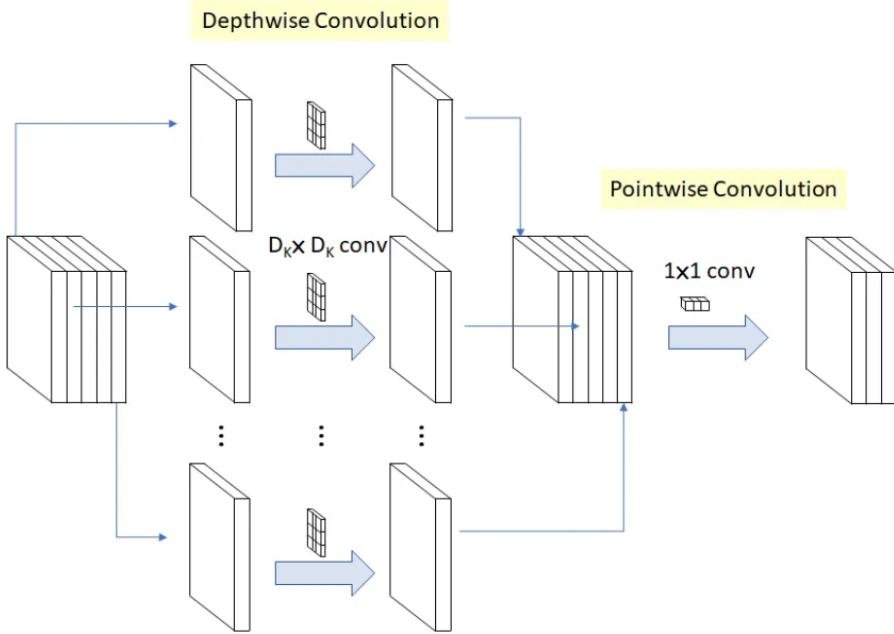


FIGURE 1.4 : Les deux étapes de convolution séparable utilisées dans MobileNet V1

1.1.2.2 MobileNet V2 (inverted residual blocks)

MobileNet V2 (voir Figure 1.5) se distingue par plusieurs améliorations par rapport à MobileNet V1, notamment l'introduction de blocs résiduels, de connexions résiduelles et d'une activation linéaire a posteriori dans chaque bloc. Il intègre également des blocs résiduels inversés avec des facteurs d'échelle linéaires dans les couches intermédiaires pour une meilleure représentation des caractéristiques ainsi qu'une fusion d'informations provenant de différentes résolutions spatiales grâce à des convolutions inverses. En outre, MobileNet V2 utilise une activation linéaire après certaines convolutions pour introduire la non-linéarité via les connexions résiduelles⁵. Globalement, MobileNet V2 offre une précision accrue et une efficacité améliorée par rapport à MobileNet V1 même si cela peut entraîner une légère augmentation du coût computationnel [16].

⁵<https://arxiv.labs.arxiv.org/html/1801.04381>

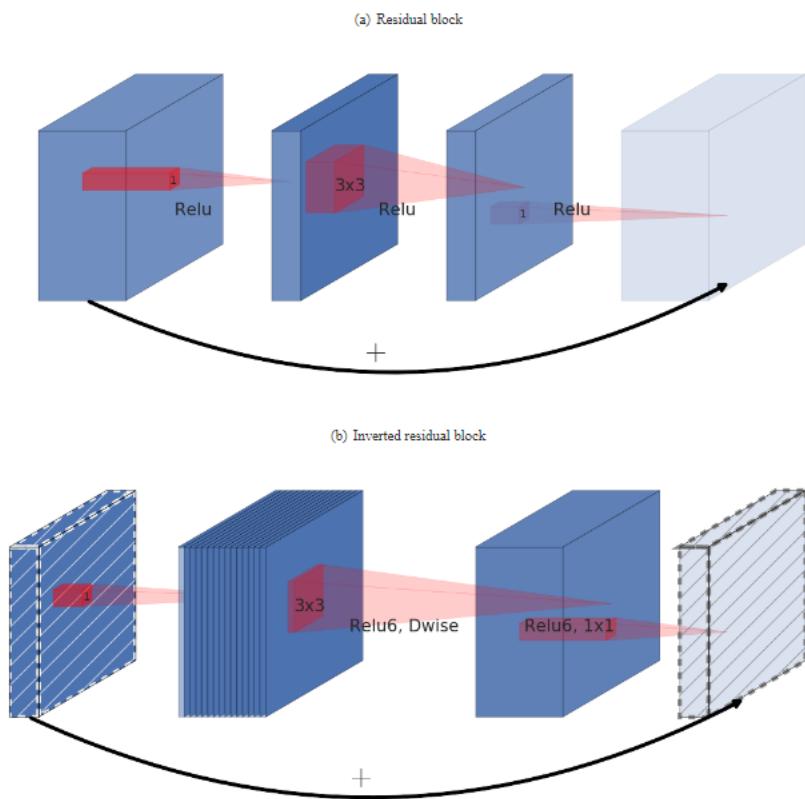


FIGURE 1.5 : Les deux blocs de MobileNet V2

1.2 Vue d'ensemble sur la synthèse vocale

Un synthétiseur de texte-parole (TTS) est un système informatique capable de lire à haute voix n’importe quel texte, qu’il soit introduit directement dans l’ordinateur par un opérateur ou numérisé et soumis à un système de reconnaissance optique de caractères (OCR).

Les applications potentielles des systèmes TTS de haute qualité sont en effet nombreuses. Voici quelques exemples : services de télécommunications, aide aux personnes handicapées, surveillance vocale.

Lorsqu’une machine lit du texte, elle ne suit pas exactement le même processus que les humains. Les sons vocaux sont régis par des équations de la mécanique des fluides et leur production implique des changements dynamiques dans la pression des poumons, la tension des cordes vocales ainsi que la configuration des voies vocales et nasales [17]. Cependant, les machines utilisent des modèles mathématiques et linguistiques pour effectuer cette tâche.

Un synthétiseur de texte en paroles typique comprend deux principaux composants (voir Figure 1.6) : un module de traitement du langage naturel (NLP) et un module de traitement du signal numérique (DSP). Le NLP produit une transcription phonétique du texte avec l’intonation et le rythme appropriés. Ensuite, le DSP transforme ces informations en parole.

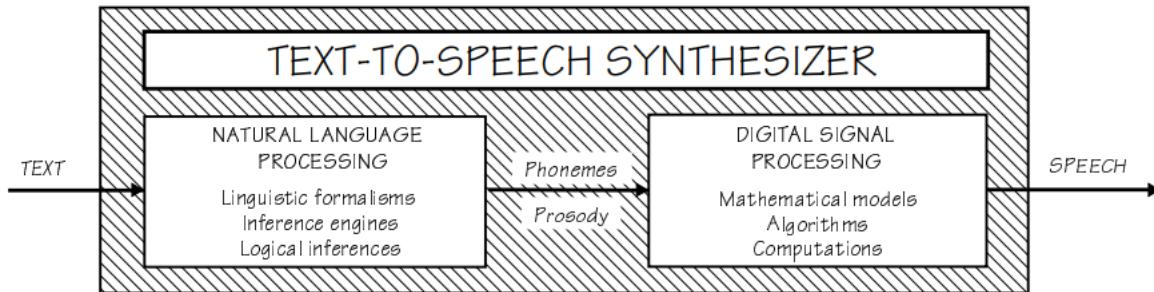


FIGURE 1.6 : Schema fonctionnel d'un système TTS

1.2.1 Traitement du Langage Naturel

Le Traitement du Langage Naturel pour Natural Language Processing (NLP) en anglais, est une branche de l'intelligence artificielle qui s'attache à donner la capacité aux machines de comprendre, générer ou traduire le langage humain tel qu'il est écrit et/ou parlé.

La Figure 1.7 présente le squelette d'un module NLP général à des fins de TTS. On remarque immédiatement que, en plus des blocs attendus de transformation lettre-son et de génération de prosodie, il comprend un analyseur morpho-syntaxique soulignant ainsi la nécessité d'un certain traitement syntaxique dans un système de synthèse vocale de haute qualité. En effet, être capable de réduire une phrase donnée à quelque chose comme la séquence de ses parties du discours et de la décrire ensuite sous forme d'arbre syntaxique qui révèle sa structure interne, est nécessaire pour au moins deux raisons :

Une transcription phonétique précise ne peut être réalisée que si la catégorie grammaticale de certains mots est disponible ainsi que si la relation de dépendance entre les mots successifs est connue. La prosodie naturelle dépend fortement de la syntaxe. Elle a évidemment beaucoup à voir avec la sémantique et la pragmatique. Mais étant donné que très peu de données sont actuellement disponibles sur les aspects génératifs de cette dépendance, les systèmes TTS se concentrent simplement sur la syntaxe. Pourtant, peu d'entre eux sont effectivement dotés de capacités de désambiguïsation et de structuration complètes [17].

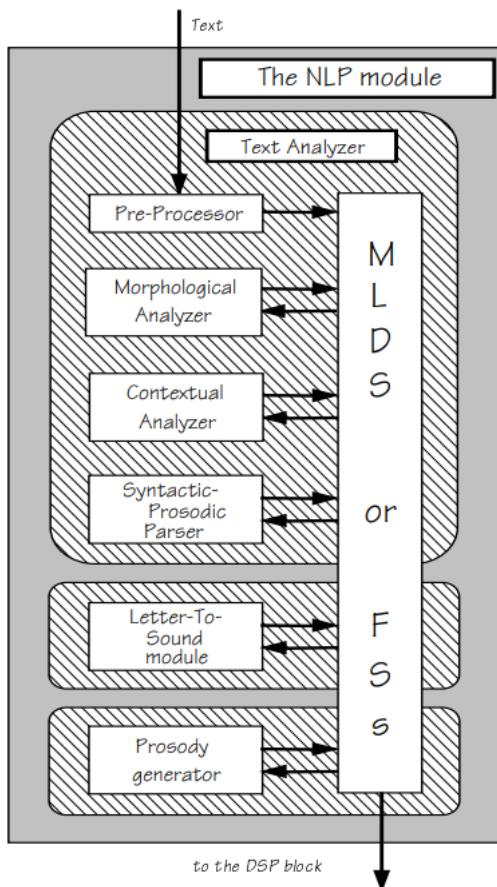


FIGURE 1.7 : Module NLP d'un système de synthèse vocale

1.2.1.1 Composant d'analyse du contexte

Le composant d'analyse du contexte dans un système de traitement du langage naturel (NLP) examine les mots dans leur contexte afin de réduire la liste des catégories grammaticales possibles à un nombre restreint d'hypothèses hautement probables. Pour cela, il utilise différentes approches, telles que les n-grams, les perceptrons multicouches ou les grammaires locales. Ces méthodes permettent de décrire les dépendances syntaxiques locales et d'inférer la structure grammaticale du texte. En fin de compte, l'analyseur syntaxique-prosodique cherche à trouver la structure textuelle qui correspond le mieux à l'organisation prosodique attendue du texte. En résumé, ce composant analyse le contexte linguistique pour comprendre le sens du texte et générer une structure syntaxique appropriée [17].

1.2.1.2 Phonétisation automatique

La phonétisation automatique consiste à déterminer la transcription phonétique d'un texte écrit à l'aide d'un système informatique. Bien que cela puisse sembler simple, plusieurs défis doivent être surmontés :

- Les variations morphologiques ne sont pas prises en compte par les dictionnaires de prononciation, nécessitant une composante supplémentaire appelée morphophonologie.
- Certains mots ont plusieurs transcriptions phonétiques en raison de leur contexte, créant des ambiguïtés de prononciation.

- Les dictionnaires de prononciation fournissent des transcriptions phonémiques plutôt que phonétiques, ne prenant pas en compte les variations réelles de prononciation.
- La prononciation des mots dans une phrase diffère souvent de celle lorsqu'ils sont isolés, influencée par des facteurs syntaxiques et prosodiques.
- De nombreux mots ne sont pas répertoriés dans les dictionnaires de prononciation, nécessitant une déduction basée sur des mots similaires.

Pour relever ces défis, les systèmes de phonétisation automatique peuvent être basés sur des dictionnaires ou des règles. Les approches basées sur des dictionnaires stockent des connaissances phonologiques dans un lexique tandis que les approches basées sur des règles utilisent des ensembles de règles pour déterminer la prononciation (voir Figure 1.8). Un compromis entre précision et couverture doit être trouvé. Ce qui dépend souvent de la langue et de la disponibilité des ressources linguistiques [17].

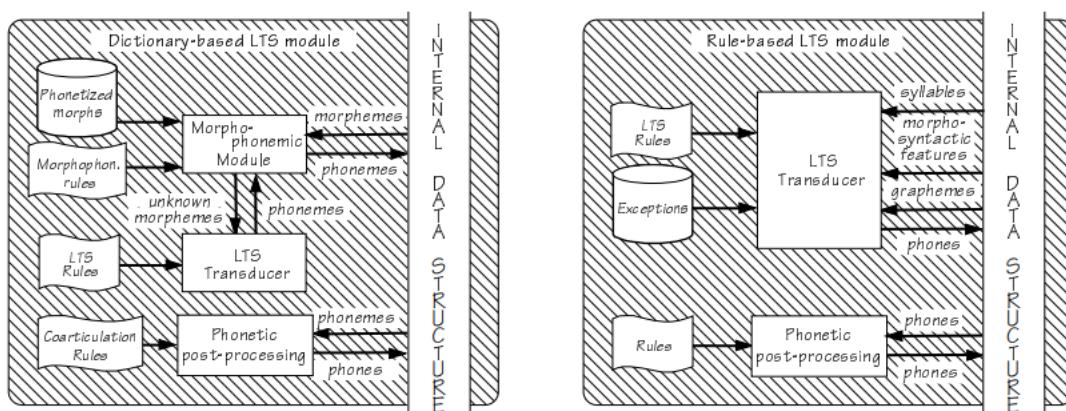


FIGURE 1.8 : Phonétisation basée sur un dictionnaire (à gauche) ou basée sur des règles (à droite)

1.2.1.3 Génération de la prosodie

La génération de la prosodie concerne les propriétés du signal vocal liées à des variations audibles de hauteur, de volume et de longueur des syllabes. Ces caractéristiques ont des fonctions spécifiques dans la communication verbale notamment en mettant en évidence certains éléments dans un discours, comme des mots ou des groupes de mots. Par exemple, des changements de hauteur peuvent mettre en évidence une syllabe, ce qui peut souligner l'importance d'un mot ou d'un groupe syntaxique dans le sens global de l'énoncé (voir Figure 1.9).

En plus de cet effet de mise en évidence, la prosodie permet également de segmenter la parole en groupes de syllabes, voire de regrouper ces syllabes et mots en unités plus larges. Ces caractéristiques prosodiques indiquent des relations entre ces groupes. Ce qui peut être hiérarchique mais pas nécessairement identique à la structure syntaxique de l'énoncé.

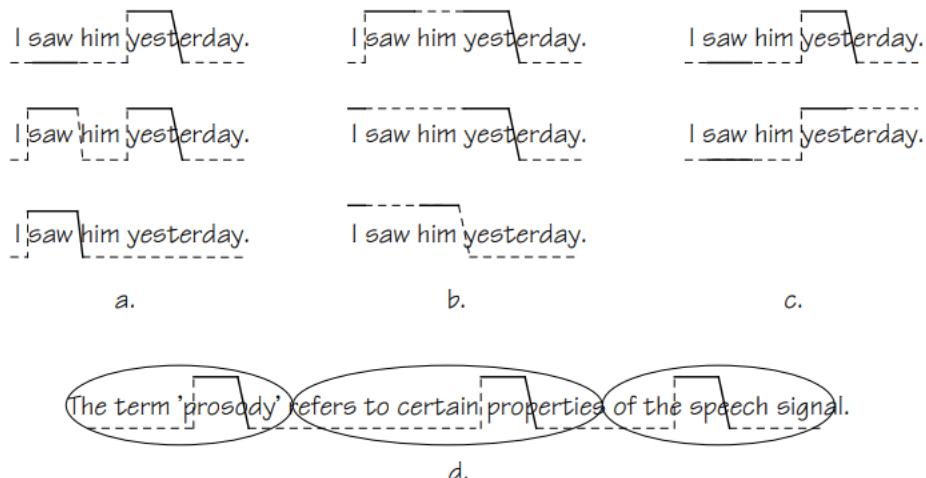


FIGURE 1.9 : Différents types d'informations fournies par l'intonation (les lignes indiquent les mouvements de hauteur ; l'accent solide)

1.2.2 Traitement du Signal Numérique

Le composant DSP (Traitement du Signal Numérique) dans la synthèse de la parole peut être comparé au contrôle dynamique des muscles articulatoires et de la fréquence vibratoire des cordes vocales dans le domaine informatique. Son objectif est d'ajuster le signal de sortie pour qu'il corresponde aux exigences du signal d'entrée. Pour ce faire, le DSP doit tenir compte des contraintes articulatoires car les transitions phonétiques sont plus importantes que les états stables pour la compréhension de la parole.

Il existe deux approches principales pour réaliser cela :

1. Explicitement, en utilisant une série de règles qui décrivent comment les phonèmes influencent les uns et les autres.
2. Implicitement, en stockant des exemples de transitions phonétiques et de coarticulations dans une base de données de segments de parole, puis en les utilisant directement comme unités acoustiques finales.

Ces deux approches ont donné lieu à deux principales classes de systèmes de synthèse de la parole : la synthèse par règle et la synthèse par concaténation. Ces classes diffèrent dans leurs méthodes et leurs objectifs, mais toutes deux visent à produire un rendu vocal naturel et compréhensible [17].

1.2.3 Google Text-to-Speech

Google Text-to-Speech, souvent abrégé en gTTS, est une implémentation spécifique de la synthèse vocale qui tire partie des ressources et des capacités de traitement du langage naturel de Google. gTTS est un système développé par Google qui automatise la conversion de texte écrit en discours vocal de manière naturelle. Son fonctionnement repose sur l'utilisation du traitement du langage naturel (NLP) pour analyser le texte et générer une transcription phonétique précise tout en intégrant la génération de la prosodie pour ajouter l'intonation et le rythme nécessaires à une lecture vocale fluide. Cette technologie utilise des concepts similaires à ceux abordés dans les sections précédentes, exploitant les avancées en NLP et la génération de la prosodie [18].

1.3 État de l'art de l'autoguidage et Étude critique de l'existant

1.3.1 Définition du déplacement

Le déplacement, concept fondamental, englobe bien plus que le simple mouvement physique. Il s'agit de la capacité à se déplacer d'un lieu à un autre de manière efficace et sécurisée, représentant ainsi une facette essentielle de la vie quotidienne. Cette compétence transcende les limites de la capacité visuelle et touche toutes les personnes impliquant également une navigation mentale à travers divers environnements. Elle inclut la compréhension de l'espace, la prise de décision rapide et la capacité à s'orienter dans des contextes variés.

1.3.2 Importance du système d'autoguidage

La perte de vision constitue un défi significatif pour le processus de déplacement augmentant les risques potentiels pour les personnes malvoyantes. L'importance du système d'autoguidage réside dans sa capacité à fournir des informations spatiales, des repères et des alertes, créant ainsi un environnement propice à un déplacement autonome et sécurisé. Pour les personnes malvoyantes, ce système devient une bouée de sauvetage améliorant leur indépendance et leur qualité de vie en facilitant des déplacements plus sûrs et plus efficaces [19].

1.3.3 Les maladies de la déficience visuelle : Causes, Symptômes et Aides

La cécité définit une déficience visuelle ou un état pathologique associé à une perte de la vision. Selon les situations et selon les causes, la perte de la vision peut être progressive ou brutale. La cécité est considérée comme un handicap dans la mesure où elle affecte la vie quotidienne des personnes qui en souffrent. Elle occasionne des difficultés, entre autres, dans toutes les activités qui mettent en jeu la vision centrale et dans les déplacements [19].

- **Causes :** Il existe plusieurs causes à la cécité qui peut être qualifiée de cécité monosulaire, cécité corticale, cécité attentionnelle, cécité totale, etc. D'une manière générale, on retrouve :
 - La cécité prénatale : il s'agit d'une déficience visuelle diagnostiquée chez l'enfant avant la naissance ;
 - La cécité des personnes âgées : l'âge est un facteur important de déficience visuelle, avec une cécité qui augmente de façon importante passé l'âge de 60 ans. De nombreuses maladies liées au vieillissement sont ainsi très souvent à l'origine de la cécité ;
 - la DMLA (dégénérescence maculaire liée à l'âge) : La macula (petite zone de la rétine) se détériore avec l'âge, provoquant une perte progressive de la vision centrale ;
 - Le glaucome : à angle fermé ou à angle ouvert, le glaucome désigne une altération du nerf optique. Non pris en charge, le glaucome peut conduire à la cécité ;
 - La cataracte : le cristallin perd de sa transparence, ce qui entraîne une perte de vision avec la sensation de brouillard ou de voile devant les yeux ;
 - La rétinopathie diabétique : il s'agit d'une forme de complication du diabète, avec un niveau de sucre dans l'organisme trop important, qui endommage la rétine.

- On recense encore d'autres causes de la cécité, plus rares, comme la rétinite pigmentaire, la neuropathie optique héréditaire de Leber, certaines tumeurs, ou encore des complications liées à une chirurgie oculaire.
- **Symptômes** : Les symptômes de la cécité varient selon les individus, selon le stade de déficience visuelle et selon la cause de la perte de la vision. Voici quelques symptômes :
 - une vision floue;
 - des difficultés à voir de nuit (ou avec un faible éclairage);
 - des difficultés de lecture;
 - une perte de la vision latérale;
 - une sensibilité accrue à la lumière;
 - la déformation des lignes droites.

1.3.4 Revue de littérature sur les systèmes d'autoguidage

1. **Sanika H. Shinde, Mousami V. Munot et al** dans leur article, [20] ont conçu "Smart Stick" (voir Figure 1.10) qui est un dispositif innovant conçu pour améliorer la mobilité des personnes aveugles et malvoyantes. Face aux défis rencontrés par ces individus, tels que les obstacles au sol et les défis liés à la sécurité, ce dispositif propose une solution multifonctionnelle. Il intègre des capteurs infrarouges et à ultrasons pour détecter les obstacles, offrant une plage de détection allant de 10 cm à 3 mètres. Ces capteurs transmettent des informations à travers des modèles vibratoires, permettant à l'utilisateur de percevoir la proximité des obstacles par des vibrations. Une fonctionnalité supplémentaire alerte l'utilisateur des surfaces mouillées grâce à un buzzer, réduisant ainsi les risques de glissade. Pour assurer la sécurité, le dispositif utilise également une technologie de suivi de localisation, permettant aux proches de l'utilisateur d'être informés de sa position actuelle en cas de besoin. En combinant ces fonctionnalités, le "Smart Stick" vise à offrir une solution abordable et efficace pour améliorer la mobilité et la sécurité des personnes aveugles et malvoyantes⁶.

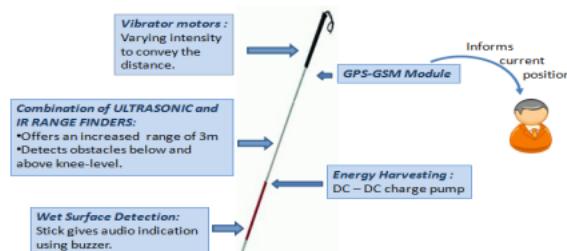


FIGURE 1.10 : Design du système proposé [20]

2. **De Alwis D et Samarakrama YC** [21] dans leur article, proposent une canne de marche intelligente (voir Figure 1.12 et 1.13) basée sur la technologie des ultrasons. La canne est équipée de plusieurs capteurs à ultrasons stratégiquement positionnés pour détecter une variété d'obstacles. Ces capteurs sont sensibles non seulement aux obstacles à hauteur d'Homme, tels que les piétons ou les objets suspendus, mais aussi aux obstacles au niveau du sol, comme les

⁶<https://www.ijitee.org/portfolio-item/J99570881019/>

flaques d'eau ou les débris. Le système est si sophistiqué qu'il peut même distinguer différents types d'obstacles, permettant ainsi à l'utilisateur de comprendre le danger spécifique qui se présente devant lui.

Outre la détection d'obstacles, la canne intègre également un capteur d'eau. Cette fonctionnalité est particulièrement utile pour alerter l'utilisateur en cas de flaques d'eau ou de zones humides, minimisant ainsi le risque de glissades ou de chutes.

Un autre aspect innovant de cette canne est sa capacité à fournir des retours haptiques à l'utilisateur. Plutôt que de se fier uniquement à des signaux sonores, la canne utilise des vibrations tactiles pour informer l'utilisateur des dangers. Ces vibrations sont conçues pour être intuitives et permettent à l'utilisateur de réagir rapidement sans avoir à détourner son attention de son environnement⁷.

L'article détaille également les tests pratiques réalisés pour évaluer l'efficacité de la canne dans différents environnements et conditions. Ces tests ont permis de valider la précision et la fiabilité de la canne, confirmant qu'elle est non seulement abordable mais aussi hautement fonctionnelle.

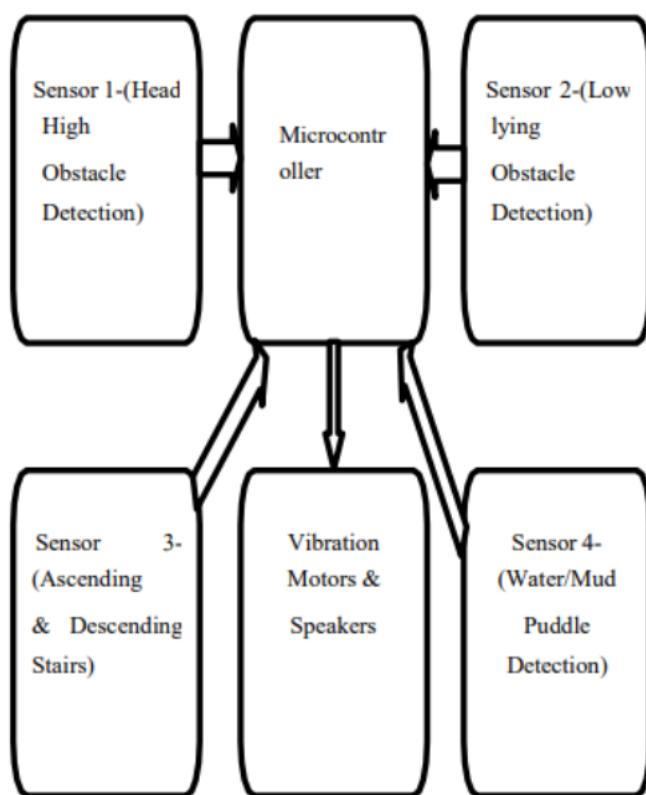


FIGURE 1.11 : Schéma fonctionnel de base de la conception

⁷<https://ijms.sljol.info/article/10.4038/ijms.v3i2.14/>

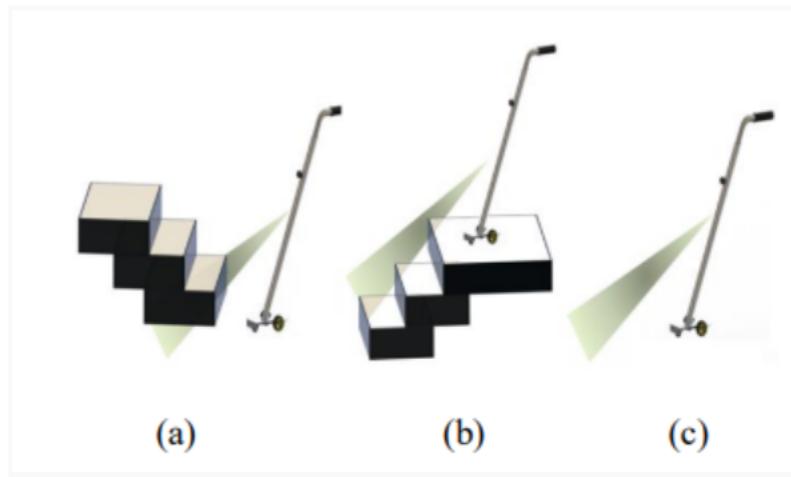


FIGURE 1.12 : Détection des mouvements ascendants et descendants dans les escaliers

- (a) Détection d'escaliers montants
- (b) Détection de descente d'escaliers
- (c) Terrain plat



FIGURE 1.13 : Détection de flaques d'eau/boue

3. Dirigé par Dr. Shivakumara Swamy de l'Institut de technologie Acharya, l'article [22] propose une ceinture intelligente dédiée aux personnes aveugles. Conçue avec des capteurs ultrasoniques, une caméra, un microprocesseur, un dispositif audio Bluetooth et des batteries rechargeables, cette ceinture grâce à ses capteurs à ultrasons analyse l'environnement immédiat de l'utilisateur pour détecter d'éventuels obstacles. Les données recueillies sont ensuite acheminées vers un modèle pré-entraîné YOLO qui analyse et classe les objets détectés. Enfin, pour informer l'utilisateur des obstacles détectés, la ceinture via le dispositif audio Bluetooth émet des alertes en temps réel, garantissant ainsi une navigation autonome et sécurisée pour les personnes malvoyantes (voir Figure 1.14).

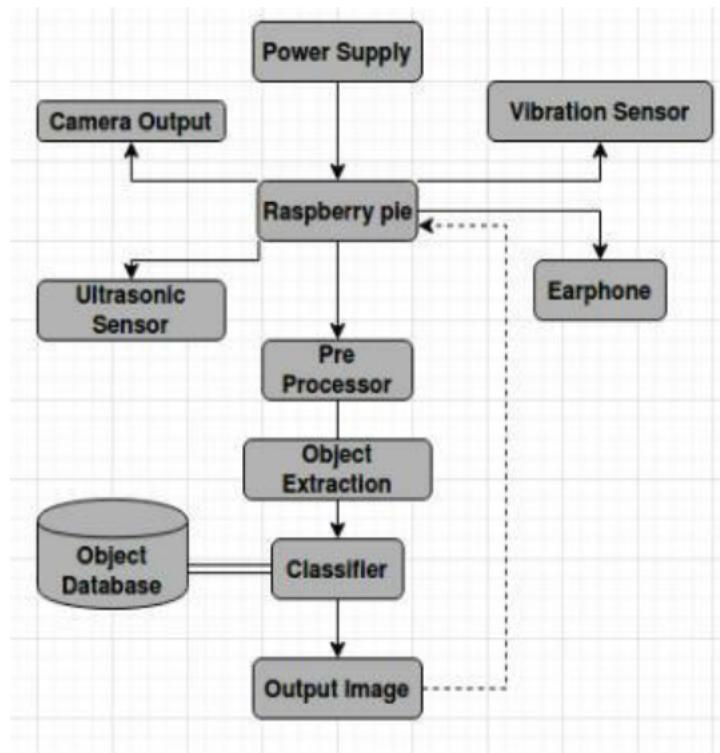


FIGURE 1.14 : Architecture globale du système

4. Cet article [23] de **Ali Jasim Ramadhan ORCHID** propose une montre intelligente portable qui surveille la trajectoire devant l'utilisateur jusqu'à une distance de 4 mètres. Lorsqu'un objet est détecté, le système émet des alarmes sonores et vibrantes. Si l'utilisateur trébuche ou a besoin d'aide, il peut activer une alarme vocale. L'appareil alerte également la famille et les soignants de la localisation de l'utilisateur via SMS. Le système est alimenté par une batterie à haute capacité soutenue par un panneau solaire. La conception du système comprend quatre parties principales : le contrôleur, les capteurs, les alarmes et l'alimentation électrique. Le contrôleur est basé sur un microcontrôleur ATmega328. Les capteurs utilisés comprennent un capteur ultrasonique pour la détection d'obstacles, un accéléromètre pour détecter les chutes de l'utilisateur et un module de reconnaissance vocale pour activer l'alarme vocale. Les alarmes comprennent des signaux sonores, des vibrations et une alerte GPS/GSM pour la localisation. L'alimentation est assurée par une batterie lithium-polymère rechargeable et un panneau solaire⁸ (voir Figure 1.15).

⁸<https://www.mdpi.com/1424-8220/18/3/843>

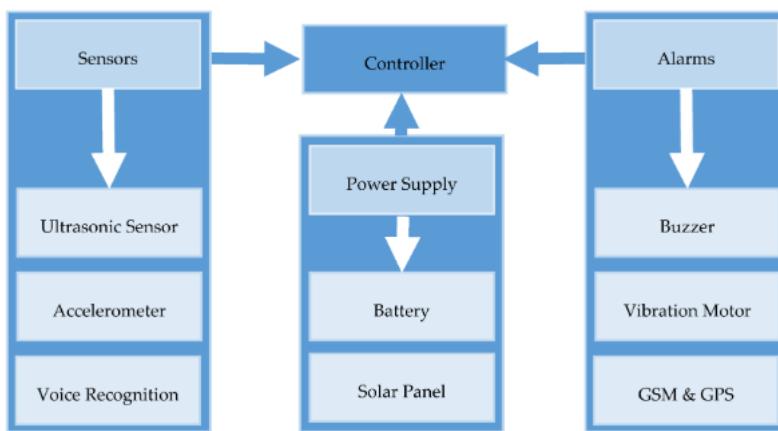


FIGURE 1.15 : Schéma du système proposé

1.3.5 Présentation de quelques applications intelligentes pour l'aide à la mobilité des personnes malvoyantes

- **Smartphones et Applications d'Assistance** : Les smartphones sont devenus des outils inestimables pour les personnes malvoyantes. Les systèmes d'exploitation mobiles, tels qu'iOS et Android, intègrent des fonctionnalités d'accessibilité puissantes qui transforment ces appareils en dispositifs d'assistance personnelle. Parmi les applications les plus utilisées, on trouve [24] :
 - **VoiceOver (iOS) et TalkBack (Android)** : Ce sont des lecteurs d'écran intégrés qui permettent aux malvoyants de naviguer sur leur smartphone en transformant le contenu de l'écran en discours vocal. Ils fonctionnent en tandem avec des gestes tactiles spécifiques, ce qui permet aux utilisateurs de lire des messages, d'explorer des applications et même de surfer sur le web de manière autonome.
 - **Seeing AI (iOS) et Lookout (Android)** : Ces applications utilisent la caméra du smartphone pour fournir des informations visuelles aux utilisateurs malvoyants. Seeing AI peut reconnaître des objets, des personnes, des documents, des couleurs et même des émotions sur les visages. Lookout, quant à lui, offre une description audio de l'environnement en temps réel, permettant aux utilisateurs de détecter les obstacles et d'explorer leur environnement avec plus de confiance.
 - **Google Maps** : Cette application de navigation GPS est largement utilisée par les malvoyants pour planifier des itinéraires, obtenir des indications vocales en temps réel et explorer des lieux à proximité. De plus, Google Maps propose une fonctionnalité "lieux accessibles" qui fournit des informations sur l'accessibilité des établissements, ce qui est particulièrement utile pour les déplacements.
 - Les systèmes de reconnaissance vocale, tels que **Siri (iOS)** et **Assistant Google**, offrent aux malvoyants un moyen pratique de contrôler leur smartphone par commande vocale. Ces assistants virtuels peuvent effectuer un large éventail de tâches, notamment appeler des contacts, envoyer des messages dictés, planifier des rappels et bien plus encore. Cette technologie est particulièrement bénéfique pour ceux qui trouvent difficile de naviguer dans les menus complexes des smartphones.

- **Intelligence Artificielle et Vision par Ordinateur dans l'assistance aux malvoyants :** Les avancées significatives dans le domaine de l'intelligence artificielle (IA) et de la vision par ordinateur ont ouvert de nouvelles perspectives passionnantes pour l'assistance aux malvoyants. Ces technologies exploitent des algorithmes sophistiqués et l'apprentissage automatique pour améliorer la qualité de vie et l'indépendance des personnes ayant une déficience visuelle [24] :
 - **Reconnaissance d'Objets et d'Environnements :** L'IA et la vision par ordinateur permettent aux dispositifs de reconnaissance d'objets de décrire précisément l'environnement d'une personne malvoyante. Les caméras intégrées aux lunettes ou aux cannes électroniques capturent des images en temps réel, et les algorithmes de traitement d'image analysent ces images pour identifier les objets, les obstacles et les caractéristiques de l'environnement.
 - **Seeing AI et Lookout :** Ces applications utilisent l'IA pour reconnaître divers objets, textes, couleurs et visages en temps réel grâce à la caméra d'un smartphone. Elles transmettent ensuite ces informations à l'utilisateur par synthèse vocale. Par exemple, elles peuvent aider à lire un panneau indicateur à une intersection ou à identifier la couleur d'un vêtement.
 - **Lunettes Intelligentes pour la Navigation :** Les lunettes intelligentes équipées de caméras et d'IA sont un autre type de système d'assistance autoguidée. Voici comment elles fonctionnent :

Caméras intégrées : Les lunettes intelligentes captent en continu des images de l'environnement de l'utilisateur. Les images sont ensuite analysées par des algorithmes IA pour détecter les objets, les obstacles et les caractéristiques de l'environnement.

Feedback Auditif ou Tactile : Les informations sont transmises à l'utilisateur sous forme de commentaires auditifs ou tactiles. Par exemple, les lunettes peuvent avertir l'utilisateur de la présence d'une porte ou d'un escalier, ou lui fournir des instructions pour contourner un obstacle.

Navigation en Intérieur et en Extérieur : Ces systèmes sont conçus pour fonctionner à la fois en intérieur et en extérieur, offrant une plus grande polyvalence aux utilisateurs malvoyants.

1.3.6 Critique de l'existant (limitations des technologies actuelles)

Après avoir examiné l'état actuel, il est essentiel de critiquer objectivement ce dernier. Cette démarche vise à identifier les lacunes observées durant cette analyse initiale afin de concevoir des solutions qui les prennent en compte. Parmi les différentes solutions envisagées, certaines lacunes ont été identifiées, notamment :

- Manque de possibilités de détection d'objets en temps réel et d'interaction : les systèmes existants ne sont pas réactifs pour détecter les obstacles en temps opportun. Ce qui pourrait compromettre la sécurité des utilisateurs. De plus, à cela s'ajoute le manque d'interac-

tions avec ces systèmes. Ce qui pourrait limiter leur utilité dans divers environnements et situations.

- Portée de détection restreinte : Bien que des dispositifs comme le "Smart Stick" et la canne basée sur les ultrasons soient innovants, ils présentent une portée de détection limitée. Cette portée restreinte signifie que les obstacles situés à une certaine distance ne sont pas détectés, ce qui pourrait exposer l'utilisateur à des risques.
- Fiabilité des retours haptiques : Bien que les retours haptiques soient une approche innovante pour informer les utilisateurs des dangers, comme mentionné dans la canne intelligente, leur efficacité et leur fiabilité peuvent être compromises dans des environnements bruyants ou encombrés.
- Intégration limitée avec les systèmes de navigation : La plupart des solutions actuelles se concentrent sur la détection d'obstacles mais n'intègrent pas des fonctionnalités de navigation comme Google Maps. Cette intégration manquante limite la capacité des malvoyants à suivre des itinéraires en extérieur.
- Dépendance à la puissance de calcul : L'utilisation de l'intelligence artificielle et de la vision par ordinateur pour détecter en temps réel les obstacles et fournir des informations de navigation précises nécessite une puissance de calcul significative. Ce que les smartphones ne peuvent pas toujours offrir. Cela entraîne des retards dans la détection des obstacles et une précision limitée. Ce qui peut mettre en danger la sécurité des utilisateurs.

Conclusion

Beaucoup de recherches ont été menées pour relever les défis que rencontrent les malvoyants dans leurs déplacements. Le Machine Learning et notamment le Deep Learning viennent apporter une grande aide pour une identification plus fiable et précise mais aussi pour rendre la tâche plus facile aux chercheurs. Dans le chapitre suivant, nous verrons les matériaux nécessaires pour mettre en place notre propre système d'auto-guidage pour les malvoyants basée sur le Machine Learning et l'IoT. Nous détaillerons également les méthodes que nous allons utiliser dans chaque module du système.

Chapitre 2

Matériel et méthodes

Introduction

Dans ce chapitre, nous faisons la description du matériel utilisé tout au long de nos travaux ainsi que les outils de travail. Nous détaillons ensuite la manière dont nous avons préparé notre environnement de travail, de même que les étapes de l'implémentation.[\[25\]](#)

2.1 Matériels

Pour notre implémentation et nos tests, nous avons travaillé sur :

TABLE 2.1 : Liste des matériaux utilisés

Outils	Caractéristiques
Google Colaboratory	processeur graphique Tesla K80 GPU
Raspberry Pi 3 (référence à l'annexe 3.11)	<ul style="list-style-type: none">Système d'exploitation : Raspbian;Type d'architecture du système d'exploitation : 32 bits;Puce (SoC) Broadcom : BCM2711;Mémoire vive : 8 Go;Stockage : microSD;Port USB : 4;Port Ethernet : 1000 Mbps;Port caméra : 1

TABLE 2.2 : Liste des matériaux utilisés (Suite)

Outils	Caractéristiques
ESP32 devkit v1 (référence à l'annexe 3.12)	<ul style="list-style-type: none"> • 240 MHz dual-core Tensilica LX6 microcontroller avec 600 DMIPS; • 520 KB SRAM intégré; • Émetteur-récepteur Wi-Fi 802.11 b/g/n HT40 Wi-Fi transceiver intégré, baseband, stack et LwIP; • Bluetooth dual mode intégré (classic et BLE); • 16 MB flash, memory-mapped to the CPU code space; • 2.3V to 3.6V operating voltage; • Temperature de fonctionnement de -40°C à +125°C; • Onboard PCB antenna / IPEX connector for external antenna
Jumper (référence à l'annexe 3.13)	<ul style="list-style-type: none"> • Modèles : Male-Male, Male- Femelle,Femelle-Femelle; • Lot Un lot de 40 jumper ou connecter; • Longueur : 20 cm; • Multi couleur; • poids : 0,030 kg
caméra Raspberry Pi Rev v1.3 (5MP, 1080p) (référence à l'annexe 3.14)	<ul style="list-style-type: none"> • Module de caméra : Omnidision 5647 5MP; • Résolution d'image fixe : 2592 x 1944; • Vidéo : prend en charge l'enregistrement 1080p à 30fps, 720p à 60fps et 640 x 480p; • Interface série de caméra MIPI : à 15 broches; • Taille : 20x25x9mm ; • Poids : 3g

2.1.1 Tensorflow

TensorFlow est un framework de programmation pour le calcul numérique qui a été rendu Open Source par Google en Novembre 2015. Depuis son release, TensorFlow n'a cessé de gagner en popularité, pour devenir très rapidement l'un des frameworks les plus utilisés pour le Deep Learning et donc les réseaux de neurones. Son nom est notamment inspiré du fait que les opérations courantes sur des réseaux de neurones sont principalement faites via des tables de données multidimensionnelles, appelées Tenseurs (Tensor). Un Tensor à deux dimensions est l'équivalent d'une matrice. Aujourd'hui, les principaux produits de Google sont basés sur TensorFlow : Gmail, Google Photos, Reconnaissance de voix.

2.1.2 TensorFlow Lite

Lorsque l'on utilise TensorFlow pour implémenter et entraîner un algorithme de machine learning, on se retrouve généralement avec un modèle qui occupe beaucoup d'espace de stockage et nécessite un GPU pour exécuter l'inférence. Sur la plupart des appareils mobiles d'énorme espace disque et des GPU ne sont pas utilisables, TensorFlow Lite est donc une solution permettant d'exécuter des modèles d'apprentissage automatique sur des appareils mobiles, les Raspberry PI, etc ... [26]. TensorFlow Lite est une adaptation du framework de machine learning TensorFlow de Google pour les mobiles et les systèmes embarqués¹. Il est un outil qui permet d'amener le machine learning dans les applications mobiles moyennant un binaire de faible taille, au chargement rapide.

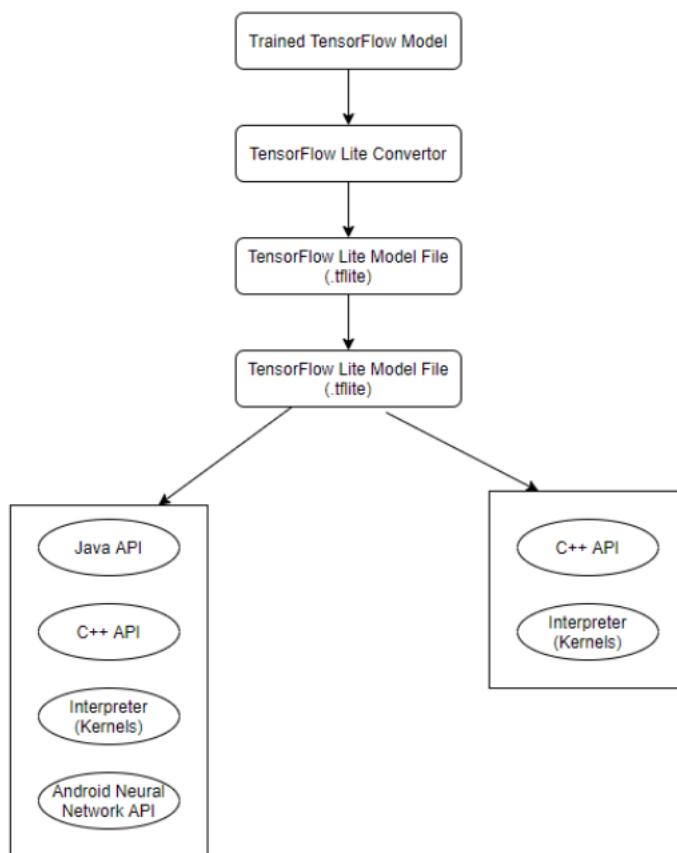


FIGURE 2.1 : L'architecture de TensorFlow Lite

2.1.3 OpenCV

OpenCV (pour Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. La société de robotique Willow Garage et la société ItSeez se sont succédé au support de cette bibliothèque. Depuis 2016 et le rachat de ItSeez par Intel, le support est de nouveau assuré par Intel. La bibliothèque OpenCV met à disposition de nombreuses fonctionnalités très diversifiées permettant de créer des programmes en partant des données brutes pour aller jusqu'à la création d'interfaces graphiques basiques. Elle propose la plupart des opérations classiques en traitement bas niveau des images telles que :

¹<https://www.geeksforgeeks.org/introduction-to-tensorflow-lite/>

- lecture, écriture et affichage d'une image;
- calcul de l'histogramme des niveaux de gris ou d'histogrammes couleurs;
- lissage, filtrage;
- seuillage, segmentation d'image.

2.1.4 Langage de programmation Python

Python est un langage de programmation de haut niveau interprété (il n'y a pas d'étape de compilation) et orienté objet avec une sémantique dynamique. Il est très sollicité par une large communauté de développeurs et de programmeurs. Python est un langage simple, facile à apprendre et permet une bonne réduction du coût de la maintenance des codes. Les bibliothèques (packages) python encouragent la modularité et la réutilisabilité des codes. Nous utilisons la version 3 de Python.

2.1.5 RealVNC

RealVNC offre une solution permettant d'accéder à distance à un appareil depuis n'importe quel endroit dans le monde, en fournissant une visualisation en temps réel du bureau et la possibilité de prendre le contrôle comme si l'utilisateur était physiquement devant l'appareil distant. Cette technologie est précieuse pour diverses applications, que ce soit pour faciliter le travail à distance, gérer des systèmes cruciaux depuis un lieu éloigné, ou assurer un support informatique à une organisation. Dans le cadre de cette étude ou nous utilisons une Raspberry Pi, RealVNC a été employé pour accéder au bureau de la Raspberry Pi à distance, offrant ainsi une connectivité pratique et sécurisée. Reconnu par les équipes informatiques de grandes entreprises dans le monde entier, RealVNC se distingue par sa facilité d'utilisation, sa sécurité renforcée et son coût abordable pour l'accès à distance.

2.1.6 Thonny

Thonny est un logiciel libre et gratuit permettant l'écriture et l'exécution de programmes écrits en Python. Il a été créé à destination des élèves et étudiants découvrant la programmation par des professeurs de l'institut de Sciences Informatiques de l'université de Tartu, en Estonie. Il est téléchargeable gratuitement, en version Windows, Mac ou Linux, et contient une version autonome de Python, qui ne perturbe en rien l'ordinateur sur lequel on installe le logiciel.

2.1.7 LabelImg

LabelImg est un outil d'annotation d'images graphiques open source développé à l'origine par TzuTa Lin et maintenu par la communauté des développeurs dans Label Studio. Il est écrit en Python et utilise l'interface graphique Qt. L'outil permet aux utilisateurs d'annoter des images en dessinant des cadres de délimitation autour des objets d'intérêt dans les images. Ces annotations peuvent être exportées dans plusieurs formats, y compris PASCAL VOC, YOLO et CreateML, sous forme de fichiers XML.

L'interface utilisateur de LabelImg est conviviale et offre des raccourcis clavier pour une navigation rapide et l'annotation de plusieurs images. Il est adapté aux systèmes basés sur Linux et fournit également une application autonome pour Windows. LabelImg est largement utilisé dans le domaine de l'apprentissage automatique pour annoter des données en vue de l'entraînement de modèles de détection d'objets.

2.2 Méthodes

La réussite de tout projet de développement de système d'information doit reposer sur une méthodologie de travail rigoureuse et bien définie garantissant sa cohérence, son efficacité et sa fiabilité. Cette section détaille notre approche méthodologique pour la conception de notre système.

2.2.1 Schéma du système proposé

L'ensemble de notre système comme le montre la figure ci-dessous (voir Figure 2.2) est composé d'un modèle pré-entraîné, d'un module de synthèse vocale, d'une caméra et d'un haut parleur :

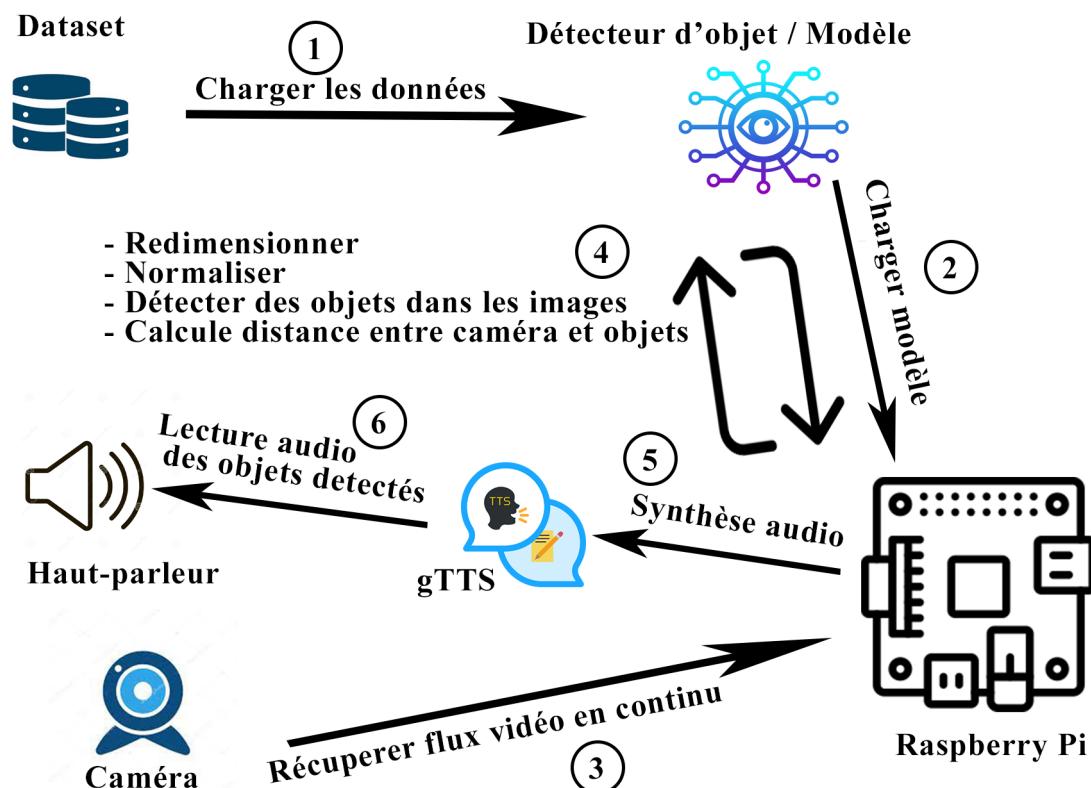


FIGURE 2.2 : Schéma du système proposé

2.2.2 Organigramme de fonctionnement du système

Ci-dessous (voir Figure 2.3), la description du processus illustré dans l'organigramme du système :

- **Allumage du système** : L'utilisateur démarre le système en appuyant sur l'interrupteur de la Raspberry Pi;
- **Chargement du modèle de détection** : Une fois le système allumé, la Raspberry Pi lance le chargement du modèle de détection;
- **Récupération des flux vidéo** : Dès que le modèle est chargé, les flux vidéo sont récupérés en continu à partir de la caméra;
- **Envoi des images capturées dans la vidéo** : Comme une vidéo est une succession d'images, les images dans la vidéo sont capturées et envoyées au modèle;
- **Envoi de la synthèse vocale des objets détectés** : Pour permettre à l'utilisateur aveugle d'avoir une perception de son environnement et de prioriser les objets les plus proches, seules les classes des objets situés au plus 100 cm sont converties en audio;
- **Lecture et diffusion de la sortie sous forme audio** : L'audio généré est lu et diffusé à l'utilisateur aveugle via un haut-parleur.

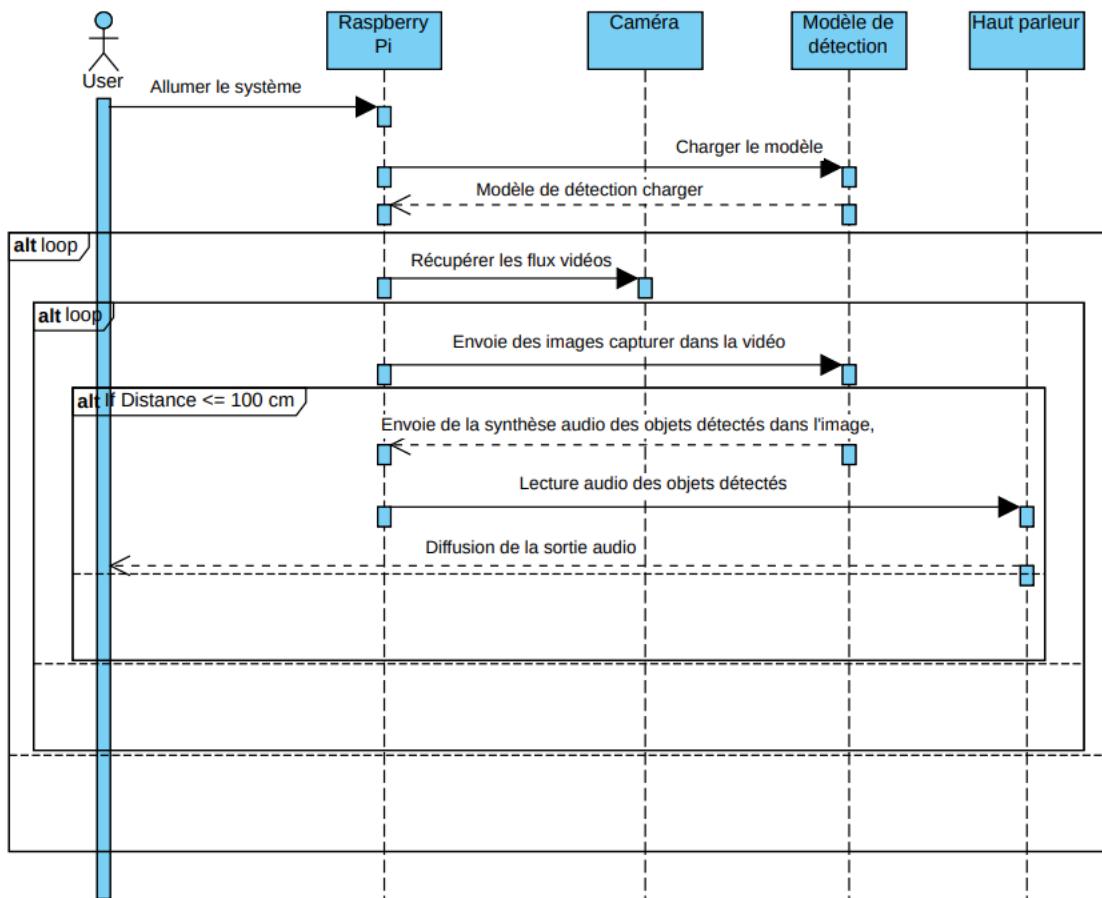


FIGURE 2.3 : Organigramme de Fonctionnement du Système

Maintenant que nous avons établi l'importance de la méthodologie et fourni un aperçu du système, explorons la méthodologie spécifique que nous avons adoptée pour ce travail.

2.2.3 Méthodologie de travail

2.2.3.1 Collecte de données

La collecte de données revêt une importance capitale dans l'élaboration de notre système d'autoguidage pour les personnes malvoyantes. Afin d'améliorer la capacité de généralisation du modèle, nous avons procédé à une collecte de données supplémentaires pour certaines classes d'objets, notamment les classes "moto" et "voiture" (qui, dans le dataset initial, étaient principalement des grosses motos de type rallye) sur le terrain à l'aide d'une caméra, capturant ainsi les images nous-mêmes.

2.2.3.2 Préparer ces données

Après avoir réussi à collecter les données nécessaires pour alimenter notre modèle de détection d'objets, la prochaine étape cruciale consiste à préparer ces données. Ce processus de prétraitement vise à garantir que notre modèle apprend efficacement à détecter les objets, en tenant compte de diverses variations telles que la rotation, le zoom, l'éclairage et autres. Les étapes impliquées dans la préparation de ces données comprennent notamment

- **Normalisation des images :** Lorsque nous abordons l'apprentissage automatique profond pour l'analyse d'images, la technique de la normalisation des pixels est souvent utilisée pour accélérer l'apprentissage du modèle. La normalisation d'une image consiste en la division de chacune des valeurs de pixel de celle-ci par la valeur maximale que peut prendre un pixel (255 pour une image en 8 bits, 4095 pour une image en 12 bits, 65 535 pour une image en 16 bits). Cette étape implique également l'ajustement des niveaux de luminosité et de contraste de chaque image pour garantir une cohérence dans l'intensité des pixels. En normalisant les données de chaque canal/tenseur pour que la moyenne soit nulle et l'écart type soit un, nous assurons que les informations du canal peuvent être mélangées et mises à jour pendant la descente de gradient (rétro-propagation) en utilisant le même taux d'apprentissage. De plus, la normalisation des images permet de réduire les variations d'éclairage entre les différentes images du jeu de données, ce qui facilite l'apprentissage du modèle en lui fournissant des données plus uniformes. Cette uniformité de données aide à limiter les gradients non nuls moins fréquents pendant l'entraînement, permettant ainsi aux neurones de notre réseau d'apprendre plus rapidement et améliorant ainsi les performances globales du modèle [27], [16].
- **Redimensionnement des images :** Notre objectif ici est de redimensionner toutes les images du jeu de données pour les adapter à la taille d'entrée requise par le modèle de détection d'objets. Contrairement à ce que l'on pourrait penser, il n'est pas toujours nécessaire de conserver les dimensions d'origine des images. En effet, lors de la formation de modèles de vision, il est courant de redimensionner les images à une dimension inférieure ((224 x 224), (299 x 299), etc.) pour permettre un apprentissage par mini-lots et également pour respecter les limitations de calcul. Pour cette étape, nous utilisons généralement des méthodes de redimensionnement d'image telles que l'interpolation bilinéaire. Il est important de noter que les images redimensionnées ne perdent pas beaucoup de leur caractère perceptuel pour l'œil humain. Ainsi, ce processus garantit que toutes les images ont les

mêmes dimensions, ce qui est nécessaire pour que le modèle puisse les traiter de manière uniforme lors de l'apprentissage et de l'inférence [28], [29].

- **Augmentation des données** : L'augmentation des données est une technique essentielle pour enrichir le jeu de données et améliorer la capacité du modèle à généraliser. Elle ne se limite pas à la création de données synthétiques, mais comprend également l'application de modifications mineures aux données existantes, telles que la rotation, le zoom, le décalage horizontal et vertical, et le retournement horizontal [30]. Ces transformations augmentent la variabilité des données d'entraînement, permettant au modèle de mieux généraliser et de mieux détecter les objets dans des conditions variables.

Pour notre projet, nous avons utilisé l'augmentation des données pour enrichir notre jeu de données existant en créant de nombreuses variantes des données d'origine. Cette approche a considérablement amélioré la performance de notre modèle en fournissant un ensemble de données plus diversifié et plus complet pour l'entraînement [31].

Les avantages de l'augmentation des données sont multiples. Elle améliore les performances du modèle en fournissant un jeu de données plus important pour l'entraînement, réduit la dépendance vis-à-vis des données en permettant l'utilisation de jeux de données plus petits, contribue à réduire le surajustement en fournissant un ensemble de données plus diversifié, et améliore la confidentialité des données en permettant la création de données synthétiques tout en préservant les propriétés statistiques et les pondérations des données originales [32], [33].

- **Etiquetage des objets dans les images** : L'annotation des images, ou étiquetage, est une étape essentielle dans les tâches de Deep Learning telles que la vision par ordinateur et l'apprentissage. Cette technique consiste à catégoriser une image en lui attribuant des annotations textuelles, généralement à l'aide d'outils logiciels spécialisés. L'objectif est de spécifier les caractéristiques des données que le modèle de Deep Learning doit identifier.

Pour notre projet, nous avons utilisé l'outil LabelImg pour annoter les objets dans les images, garantissant ainsi l'exactitude des annotations et la qualité des données d'entraînement. LabelImg nous a permis d'ajouter des métadonnées à notre ensemble de données, définissant ainsi les fonctionnalités que notre modèle devait reconnaître. Cette approche nous a permis de préparer efficacement nos données pour l'entraînement du modèle en assurant une compréhension précise des objets présents dans les images.

Il est crucial de choisir le bon logiciel d'annotation d'images pour garantir une annotation précise et efficace. LabelImg s'est avéré être un choix adapté pour notre projet, offrant les fonctionnalités nécessaires pour annoter les objets dans nos images collectées. Ce qui a contribué à la réussite de notre modèle de détection d'objets [34], [35].

2.2.3.3 Sélection de modèles

(a) Critères de sélection des types d'architecture CNN

Le choix de l'architecture CNN adaptée revêt une importance capitale dans le développement de notre système d'autoguidage pour les malvoyants, en particulier en vue de son déploiement sur des dispositifs embarqués tels que la Raspberry Pi. Pour garantir des performances optimales sur ce type de plateforme, plusieurs critères ont été soigneusement pris en compte lors de la sélection des modèles [16], [36] :

- La **taille du modèle** fait référence à la quantité de mémoire requise pour stocker le modèle CNN sur le dispositif embarqué, en particulier la Raspberry Pi. Une taille réduite est cruciale pour s'adapter aux contraintes matérielles de cet appareil [31];
- La **précision (COCO 2017)** indique la performance du modèle sur l'ensemble de données COCO 2017, une référence courante dans le domaine de la détection d'objets. Une précision élevée est souhaitable pour garantir des résultats fiables dans notre application;
- L'**efficacité informatique** se réfère à la capacité du modèle à utiliser efficacement les ressources matérielles disponibles, telles que la mémoire et la puissance de calcul, sur la Raspberry Pi. Un modèle efficace permet d'optimiser les performances tout en minimisant l'utilisation des ressources [37];
- Le **nombre de paramètres** correspond au nombre total de poids du modèle. Ce qui peut avoir un impact direct sur sa taille et sa complexité. Un nombre de paramètres plus faible est préférable pour réduire la taille du modèle et améliorer son efficacité [38];
- Le **scaling en largeur, profondeur et résolution** fait référence à la capacité du modèle à être adapté à différentes tailles d'entrée ou à être modifié pour atteindre différentes précisions. Cette flexibilité est importante pour s'adapter aux besoins spécifiques de notre application sur la Raspberry Pi;
- Les **MACs** (Multiplications-Accumulations) représentent le nombre total d'opérations de multiplication et d'accumulation effectuées par le modèle lors de l'inférence. Des valeurs plus basses de MACs indiquent une meilleure efficacité du modèle en termes de calcul [31].

TABLE 2.3 : Critère de sélection des types d'architecture CNN [39], [40]

Critère	MobileNet V2	EfficientNet	VGG	ResNet
Taille du modèle (MB)	14	5,3	535	110
Précision (COCO 2017)	0,695	0,756	0,49	0,54
Efficacité informatique (GFLOPs)	0,300	8,2	25	3,8
Nombre de paramètres (Millions)	3,4	5,3	138	25
MACs (Milliards)	1,44	6,6	30,7	3,9
Scalabilité en largeur	Limitée (0.25-2.0)	Élevée	Pas Scalable	Limitée
Scalabilité en profondeur	Limitée (1-1.4)	Élevée	Pas Scalable	Limitée
Scalabilité en résolution	Limitée (224-224)	Élevée (jusqu'à 672-672)	Pas Scalable	Limitée

Notre objectif est de choisir des architectures CNN qui offrent un équilibre entre perfor-

mances et efficacité pour répondre aux contraintes matérielles des appareils embarqués tout en assurant des résultats de qualité pour notre système.

(b) **Critères de sélection des détecteurs d'objets** Dans le cadre de notre système d'autoguidage pour les malvoyants, la sélection d'un détecteur d'objet adapté revêt une importance cruciale, notamment en raison de l'aspect temps réel de notre application. Nous récupérons des flux vidéo en temps réel. Ce qui nécessite des détecteurs d'objet capables de traiter les images rapidement et avec précision. Ainsi, plusieurs critères ont été pris en compte lors de la sélection des détecteurs, notamment : [10], [41] :

- La **détection en temps réel** se réfère à la capacité du détecteur à traiter les images en temps réel. Ce qui est essentiel pour notre application d'autoguidage sur la Raspberry Pi. Un détecteur capable de détecter en temps réel permet une interaction fluide avec l'utilisateur ;
- La **précision** indique la capacité du détecteur à détecter avec précision les objets dans les images. Une précision élevée est cruciale pour garantir des résultats fiables dans notre application d'autoguidage ;
- La **capacité de calcul** se réfère à la quantité de ressources matérielles nécessaires pour exécuter le détecteur efficacement. Un détecteur efficace en termes de calcul permet de maximiser les performances tout en minimisant l'utilisation des ressources matérielles sur la Raspberry Pi.

TABLE 2.4 : Critères de sélection des détecteurs d'objets [42]

Critère	SSD	SSD FPN	EfficientDet-D0	YOLO	Faster RCNN
Détection en temps réel	Oui	Oui	Variable	Oui	Non
Précision (mAP)	0,7	0,76	0,83	0,46	0,73
Capacité de calcul (GFLOPs)	23,2	41,1	Variable	220	195

Notre objectif est de choisir des détecteurs d'objet qui garantissent des performances fiables tout en respectant les contraintes de temps et de calcul de notre système d'autoguidage.

(c) **Architecture des modèles selectionnés**

- **Architecture SSD-Mobilenet-v2**

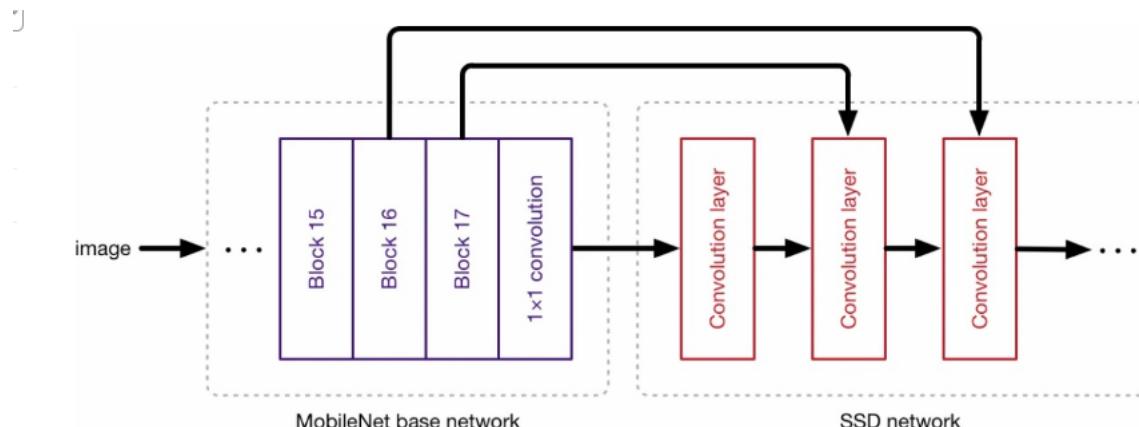


FIGURE 2.4 : Architecture du SSD-MobileNet-v2

L'architecture du modèle SSD MobileNetV2 (voir Figure 2.4) est composée de deux parties principales : le réseau de base MobileNetV2 et la couche SSD (Single Shot Multibox Detector)². Voici une explication détaillée de chaque partie :

– **Réseau de base (MobileNetV2) :**

- * MobileNetV2 est un réseau de neurones convolutif léger conçu pour la vision mobile. Il utilise des convolutions séparables en profondeur pour réduire le nombre de paramètres tout en conservant la capacité de représentation.
- * Les couches principales de MobileNetV2 sont construites sur des filtres séparables en profondeur à l'exception de la première couche qui est une convolution complète.
- * Il agit comme un extracteur de fonctionnalités pour l'image détectée.

– **Couche SSD :**

- * La couche SSD est la partie responsable de la détection d'objets. Elle produit une collection de boîtes englobantes de taille fixe et attribue des scores à la présence d'instances de classes d'objets dans ces boîtes.
- * Le SSD est conçu pour être indépendant du réseau de base. Ce qui signifie qu'il peut fonctionner avec différents réseaux de base, tels que VGG, YOLO, ou dans ce cas, MobileNetV2.
- * Le réseau SSD est basé sur une approche de détection d'objet à prise unique. Ce qui signifie qu'il prend une seule passe pour détecter plusieurs objets dans une image.

– **Architecture MobileNetSSDv2 :**

- * La première partie du modèle est constituée du réseau de base MobileNetV2, qui sert d'extracteur de caractéristiques pour l'image.
- * La deuxième partie est la couche SSD qui utilise les caractéristiques extraites pour détecter et classer les objets présents dans l'image.
- * L'utilisation de MobileNetV2 comme réseau de base contribue à la légèreté et à l'efficacité du modèle, le rendant adapté aux applications embarquées et à faible consommation d'énergie.

• **Architecture EfficientDet-D0**

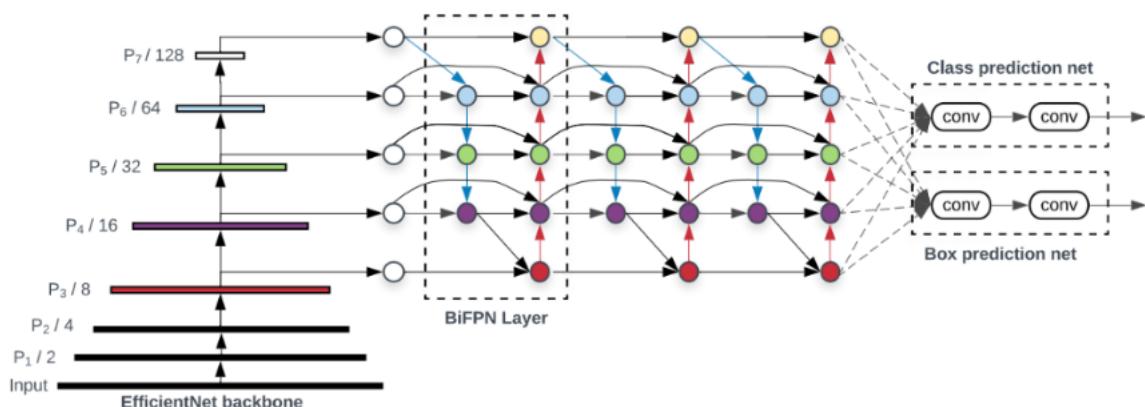


FIGURE 2.5 : Topologie complète d'un modèle EfficientDet

²<https://roboflow.com/model/mobilenet-ssd-v2>

L'architecture du modèle EfficientDet (voir Figure 2.5) est conçue pour être à la fois précise et efficace en termes de ressources. Ses composants clés et son fonctionnement sont les suivantes :

– **Backbone - EfficientNet :**

- * Le modèle EfficientDet utilise les modèles EfficientNet comme backbone. EfficientNet est une famille de modèles d'apprentissage profond qui optimise la profondeur, la largeur et la résolution pour obtenir un équilibre entre la précision du modèle et son efficacité en termes de ressources.
- * Les variantes d'EfficientNet de B0 à B6 sont utilisées pour la conception du backbone, permettant de réutiliser les coefficients de mise à l'échelle en largeur et en profondeur déjà définis pour ces modèles.

– **Bi-directional Feature Pyramid Network (BiFPN) :** est une nouvelle architecture de réseau bidirectionnel de pyramide de fonctionnalités. Il est conçu pour permettre une fusion rapide et facile des caractéristiques multiscales. Cette fusion bidirectionnelle aide à améliorer la représentation des objets à différentes échelles dans l'image.

– **Méthode de mise à l'échelle composée (Compound Scaling) :**

- * EfficientDet utilise une méthode de mise à l'échelle composée qui utilise un coefficient composé unique pour ajuster uniformément la résolution, la profondeur et la largeur de tous les composants du réseau, y compris le backbone, le BiFPN et les réseaux de prédiction de classes/boîtes.
- * Cette méthode de mise à l'échelle uniforme permet de maintenir un équilibre entre différentes dimensions du réseau, assurant ainsi une efficacité globale.

– **Efficacité du modèle :**

- * Les modèles EfficientDet ont été conçus pour être jusqu'à 9 fois plus petits et utiliser jusqu'à 42 fois moins d'opérations à virgule flottante (FLOPs) que les détecteurs précédents tout en maintenant une précision de pointe.
- * Ils fonctionnent également de 2 à 4 fois plus rapidement sur GPU et de 5 à 11 fois plus rapidement sur CPU par rapport à d'autres détecteurs.
- **Performances spécifiques (EfficientDet-D7) :** La variante EfficientDet-D7 atteint une précision de 55,1 AP sur le jeu de données COCO test-dev avec seulement 77 millions de paramètres et 410 milliards de FLOPs, démontrant une efficacité exceptionnelle.

EfficientDet combine un backbone EfficientNet, un BiFPN pour une fusion rapide des caractéristiques, et une méthode de mise à l'échelle composée pour ajuster uniformément toutes les dimensions du réseau. Ces éléments clés travaillent ensemble pour fournir des modèles de détection d'objets à la fois précis et adaptés à diverses contraintes de ressources.

- Architecture SSD-Mobilenet-v2-fpnlite

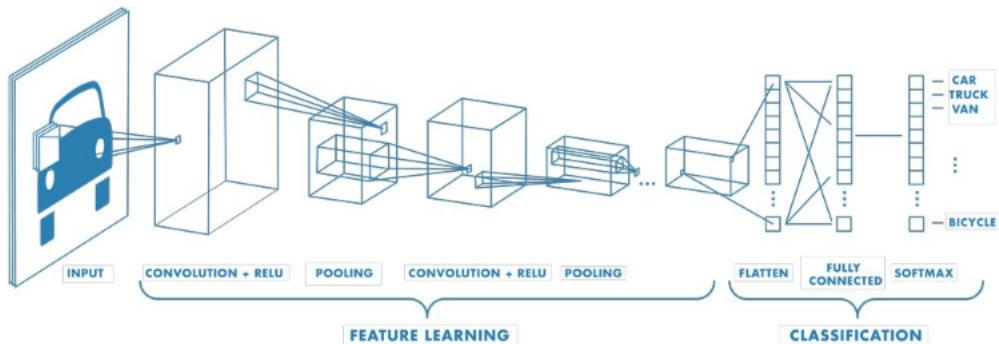


FIGURE 2.6 : Architecture du SSD-Mobilenet-v2-fpnlite

Ce modèle (voir Figure 2.6) utilise MobileNetV2 comme backbone et intègre une version "Lite" de Feature Pyramid Network (FPN) pour optimiser la fusion des caractéristiques à différentes échelles spatiales. Avec une résolution d'entrée de 320x320 pixels, comparable au **SSD MobileNetV2**, l'ajout de FPN Lite renforce la capacité du modèle à détecter efficacement des objets de diverses tailles. Cette amélioration dans la gestion des caractéristiques à différentes échelles contribue à une détection plus robuste et précise d'objets, renforçant ainsi les performances du modèle dans des scénarios variés de détection d'objets³.

2.2.3.4 Fine-turning

Le fine-turning d'un modèle est une étape cruciale dans le processus d'entraînement des réseaux de neurones, en particulier dans le domaine de l'apprentissage par transfert. Il permet d'adapter les modèles pré-entraînés à des tâches spécifiques telles que la détection d'objets ou la segmentation sémantique. En utilisant les connaissances générales acquises par ces modèles pré-entraînés, le fine-turning réduit le besoin de collecter de grandes quantités de données et accélère l'entraînement économisant ainsi du temps et des ressources. De plus, il améliore les performances en adaptant les caractéristiques du modèle à la tâche spécifique favorisant ainsi une meilleure généralisation et prévenant le sur-ajustement. En permettant la réutilisation efficace des modèles existants, le fine-turning favorise également le partage des connaissances et accélère le progrès dans le domaine de l'apprentissage automatique.

Afin d'adapter nos choix de modèles (SSD-Mobilenet-v2, EfficientDet-D0) à notre problème, nous avons appliqué la stratégie 1 de transfert d'apprentissage : le Fine-turning total. En conséquence, les couches supérieures de ce modèle ont été finement ajustées en ré-entraînant ces dernières avec notre propre ensemble de données d'entraînement annotées. Cela a impliqué le ré-entraînement des couches responsables de l'extraction des caractéristiques à partir des images.

En ce qui concerne les paramètres d'entraînement, le tableau ci-dessous définit pour chaque modèle comment ils ont été définis :

³<https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/mobilenetv2-ssd-fpn>

TABLE 2.5 : Définition des paramètres d'entraînement des modèles

Options	SSD-Mobilenet-v2	EfficientDet-D0
La taille du lot	16	4
Nombre d'étapes d'entraînement	20 000	20 000
Optimiseur	Momentum avec une décroissance cosinus pour le taux d'apprentissage	Momentum avec une décroissance cosinus pour le taux d'apprentissage
Augmentation de données	Retournement horizontal et le redrage aléatoire	Retournement horizontal et la mise à l'échelle aléatoire

2.2.3.5 Mesures de performances des modèles pour les problèmes de classification multi-classes

Après l'ingénierie des fonctionnalités et la modélisation, évaluer l'efficacité du modèle est crucial. L'utilisation de données de test avec différentes métriques, comme la matrice de confusion (voir tableau 2.6), permet d'évaluer les performances du modèle et d'interpréter ses résultats.

TABLE 2.6 : Matrice de confusion d'un problème de classification binaire

		Valeur Actuelle	
Valeur Prédite	Positive		Négative
	Positive	TP (Vrai Positif)	FP (Faux Positif)
Négative	FN (Faux Négatif)	TN (Vrai Négatif)	

Dans une classification multi-classe comme celle étudiée ici, la matrice de confusion comprend k lignes et k colonnes, où chaque ligne représente une classe réelle et chaque colonne représente une classe prédite (voir tableau 2.7).

TABLE 2.7 : Matrice de confusion multi-classe pour 3 classes

		Valeur Actuelle		
Valeur Prédite	Class A		Class B	Class C
	Class A	TP A	FN B, FP A	FN C, FP A
Class B	FP B, FN A	TP B	FN B, FP B	
Class C	FP C, FP A	FP C, FP B	TP C	

Une fois que la matrice de confusion a été établie, elle peut être utilisée pour des mesures plus approfondies afin d'obtenir une meilleure évaluation de la qualité du modèle. Parmi les mesures de classification, on trouve :

- **Précision (Precision)** : La précision mesure la proportion de vrais positifs parmi toutes les instances classées comme positives par le modèle. C'est un indicateur de l'exactitude des prédictions positives.

$$\text{Precision} = \frac{\text{Vrais Positifs}}{\text{Vrais Positifs} + \text{Faux Positifs}} \quad (2.1)$$

- **Rappel (Recall)** : Le rappel mesure la proportion de vrais positifs parmi toutes les instances réellement positives. Cela évalue la capacité du modèle à détecter toutes les occurrences de la classe.

$$\text{Recall} = \frac{\text{Vrais Positifs}}{\text{Vrais Positifs} + \text{Faux Négatifs}} \quad (2.2)$$

- **F-mesure (F1 Score)** : La F-mesure est la moyenne harmonique de la précision et du rappel. Elle fournit un équilibre entre ces deux métriques, en particulier lorsque les classes sont déséquilibrées.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (2.3)$$

- **Exactitude (Accuracy)** : L'exactitude mesure la proportion totale de prédictions correctes parmi toutes les prédictions.

$$\text{Accuracy} = \frac{\text{Vrais Positifs} + \text{Vrais Négatifs}}{\text{Total des prédictions}} \quad (2.4)$$

2.2.3.6 Intégration de la synthèse vocale (TTS)

Le module Text-to-Speech (TTS) revêt une importance capitale dans notre système, facilitant une interaction fluide avec les utilisateurs. Pour notre implémentation, nous avons opté pour gTTS en raison de ses performances fiables, de sa convivialité et de sa parfaite compatibilité avec notre architecture existante.

gTTS offre une solution efficace pour convertir du texte en parole avec une qualité sonore satisfaisante, répondant ainsi aux exigences de notre système d'autoguidage pour les malvoyants [18].

Pour garantir la sécurité des utilisateurs malvoyants, nous avons mis en place un système qui calcule la distance focale [43] entre la caméra et les objets détectés. Cette mesure nous permet de limiter les notifications vocales aux objets situés à plus de 100 cm de l'utilisateur. Cette approche vise à garantir que seuls les objets proches, potentiellement pertinents pour l'utilisateur, déclenchent une réponse vocale. Cela contribue à rendre l'interaction avec le système plus intuitive et sécurisée, en fournissant des informations précises et pertinentes en temps opportun.

(Voir Annexe 5 pour plus de détails sur le code)

2.2.3.7 Déploiement du modèle sur un Raspberry Pi

Pour support de déploiement de notre modèle, nous utilisons un Raspberry Pi.

(Voir Annexe 4 pour plus de détails sur le code de déploiement)

Afin de récupérer les flux vidéo et lancer la détection dès le démarrage du Raspberry Pi nous écrivons un fichier shell (lancement.sh) que nous avons ajouté dans le crontab.

(Voir Annexe 7 et 8 pour plus de détails sur le code)

Conclusion

Dans ce chapitre, nous avons présenté les différents langages de programmation, les bibliothèques, et décrit les caractéristiques des différents matériaux utilisés pour l'implémentation. Nous avons également décrit la méthodologie de travail suivie pour la mise en place de notre système.

Chapitre 3

Résultats et discussions

Introduction

Les chapitres précédents nous ont permis de faire un état de l'art, de présenter les outils et méthodes dont nous avons fait usage pour l'élaboration de notre solution. Dans ce chapitre, nous présentons les résultats obtenus en se basant sur les précédents chapitres de ce document. Après avoir présenté le dataset utilisé, nous allons, dans un premier temps présenter les résultats obtenus puis, enfin, présenter les résultats de tests sur la Raspberry.

3.1 Le dataset utilisé

Le modèle initial a été entraîné sur le dataset COCO 2017, qui contient environ **122 929** images. Afin d'améliorer la capacité de généralisation du modèle pour certaines classes d'objets, notamment les classes "voiture" et "moto" (qui, dans le dataset initial, étaient principalement des grosses motos de type rallye alors qu'en Afrique, les motos les plus courantes sont de petites motos de types Djènannan, Badjaj et Math), nous avons complété ce dataset par une collecte de données supplémentaires sur le terrain. Nous avons pris un total de **1 338** images de ces classes (voiture et moto) avec une caméra, que nous avons ensuite labellisées à l'aide du logiciel LabelImg. Notre jeu de données final comprend donc **124 267** images, réparties équitablement en **79** classes, comme répertorié dans le tableau ci-dessous :

TABLE 3.1 : Liste des classes d'objets de la dataset

Numero	Classes
1	person
2	bicycle
3	car
4	motorcycle
5	airplane
6	bus
7	train
8	truck
9	boat
10	traffic light
11	fire hydrant
12	stop sign
13	parking meter
14	bench
15	bird
16	cat
17	dog
18	horse
19	sheep
20	cow
21	elephant
22	bear
23	zebra
24	giraffe
25	backpack
26	umbrella
27	handbag
28	tie
29	suitcase
30	frisbee
31	skis
32	snowboard
33	sports ball
34	baseball bat
35	baseball glove
36	skateboard
37	surfboard
38	tennis racket
39	bottle
40	wine glass

TABLE 3.2 : Liste des classes d'objets de la dataset (Suite)

Numero	Classes
41	cup
42	fork
43	knife
44	spoon
45	bowl
46	banana
47	apple
48	sandwich
49	orange
50	broccoli
51	carrot
52	hot dog
53	pizza
54	donut
55	cake
56	chair
57	couch
58	potted plant
59	bed
60	dining table
61	toilet
62	tv
63	laptop
64	mouse
65	remote
66	keyboard
67	cell phone
68	microwave
69	oven
70	toaster
71	sink
72	refrigerator
73	book
74	clock
75	vase
76	scissors
77	teddy bear
78	hair drier
79	toothbrush

Nous avons divisé notre dataset en train, validation et test dans les proportions respectives de 80%, 10% et 10%.

3.2 Résultats des entraînements faits sur le jeu de données avec le modèle SSD-Mobilenet-v2-fpnlite-320

- Avant la collecte supplémentaire

Avant la collecte supplémentaire le modèle SSD-Mobilenet-v2-fpnlite-320 entraîné avec les données d'entraînement pendant **20 000** itérations (étapes / epochs) avec un lot de **16** images pour chacune, nous avons obtenu une précision de **86,88%** contre une perte de **13,12%**.

TABLE 3.3 : Précision et perte du modèle SSD-Mobilenet-v2-fpnlite-320 avant collecte supplémentaire

Perte (Loss)	Précision (Accuracy)
13,12%	86,88%

- Après la collecte supplémentaire

Après la collecte supplémentaire le modèle SSD-Mobilenet-v2-fpnlite-320 entraîné avec les données d'entraînement pendant **20 000** itérations (étapes / epochs) avec un lot de **16** images pour chacune, nous avons obtenu une précision de **95,46%** contre une perte de **4,53%**.

TABLE 3.4 : Précision et perte du modèle SSD-Mobilenet-v2-fpnlite-320 après collecte supplémentaire

Perte (Loss)	Précision (Accuracy)
4,53%	95,46%

Au cours des 20 000 itérations de notre entraînement, nous constatons une diminution significative de la perte jusqu'à atteindre 0,1% (voir Figure 3.1).

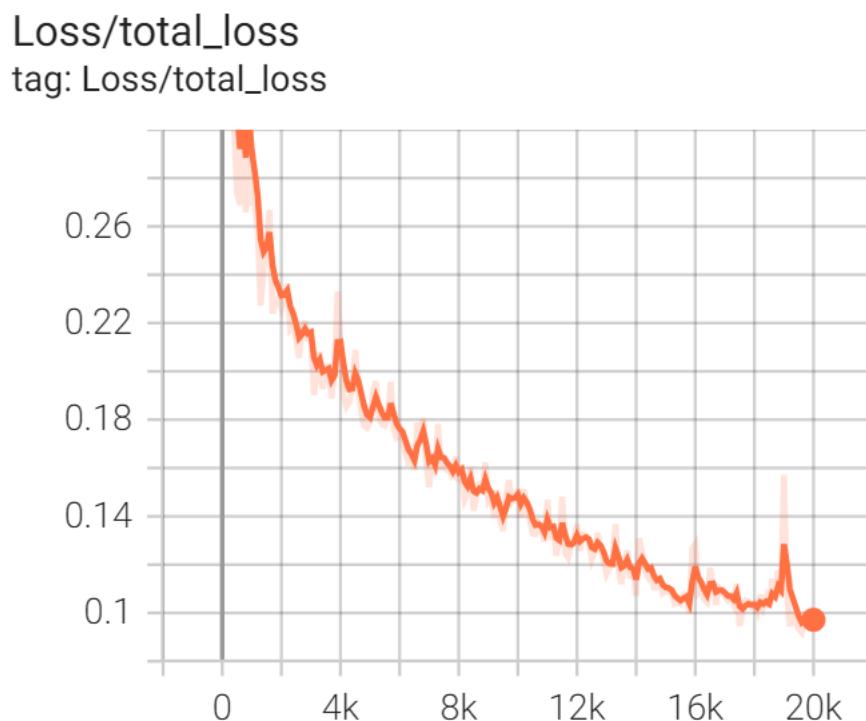


FIGURE 3.1 : TensorBoard d’entraînement du modèle SSD-Mobilenet-v2-fpnlite-320 après collecte supplémentaire

3.3 Résultats des entraînements faits sur le jeu de données avec le modèle SSD-Mobilenet-v2

- Avant la collecte supplémentaire

Avant la collecte supplémentaire le modèle SSD-Mobilenet-v2 entraîné avec les données d'entraînement pendant **20 000** itérations (étapes / epochs) avec un lot de **16** images pour chacune, nous avons obtenu une précision de **74,33%** contre une perte de **25,67%**.

TABLE 3.5 : Précision et perte du modèle SSD-Mobilenet-v2 avant collecte supplémentaire

Perte (Loss)	Précision (Accuracy)
25,67%	74,33%

- Après la collecte supplémentaire

Après la collecte supplémentaire le modèle SSD-Mobilenet-v2 entraîné avec les données d'entraînement pendant **20 000** itérations (étapes / epochs) avec un lot de **16** images pour chacune, nous avons obtenu une précision de **86,28%** contre une perte de **13,71%**.

TABLE 3.6 : Précision et perte du modèle SSD-Mobilenet-v2 après collecte supplémentaire

Perte (Loss)	Précision (Accuracy)
13,71%	86,28%

Au cours des 20 000 itérations de notre entraînement, nous constatons une diminution significative de la perte jusqu'à atteindre 0,18% (voir Figure 3.2).

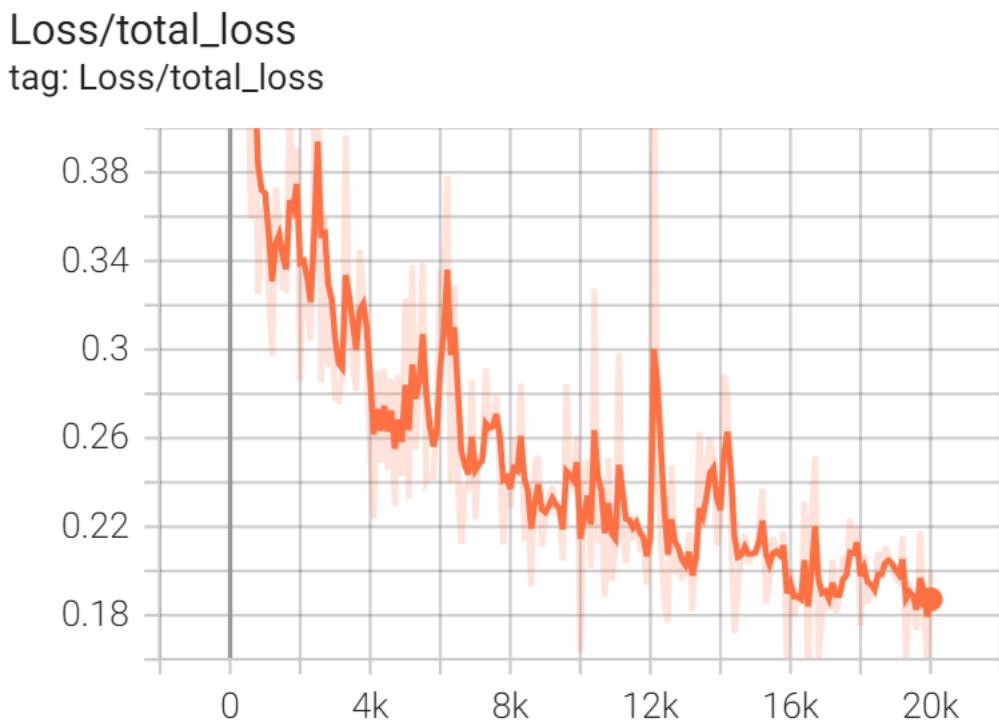


FIGURE 3.2 : TensorBoard d’entraînement du modèle SSD-Mobilenet-v2 après collecte supplémentaire

3.4 Résultats des entraînements faits sur le jeu de données avec le modèle EfficientDet-D0

- Avant la collecte supplémentaire

Avant la collecte supplémentaire le modèle EfficientDet-D0 entraîné avec les données d’entraînement pendant 20 000 itérations (étapes / epochs) avec un lot de 16 images pour chacune, nous avons obtenu une précision de 65,98% contre une perte de 34,02%.

TABLE 3.7 : Précision et perte du modèle EfficientDet-D0 avant collecte supplémentaire

Perte (Loss)	Précision (Accuracy)
34,02%	65,98%

- Après la collecte supplémentaire

Après la collecte supplémentaire le modèle EfficientDet-D0 entraîné avec les données d’entraînement pendant 20 000 itérations (étapes / epochs) avec un lot de 16 images pour chacune, nous avons obtenu une précision de 79,19% contre une perte de 20,80%.

TABLE 3.8 : Précision et perte du modèle EfficientDet-D0 après collecte supplémentaire

Perte (Loss)	Précision (Accuracy)
20,80%	79,19%

Au cours des 20 000 itérations de notre entraînement, nous constatons une diminution significative de la perte jusqu'à atteindre 0,2% (voir Figure 3.3).

Loss/total_loss
tag: Loss/total_loss

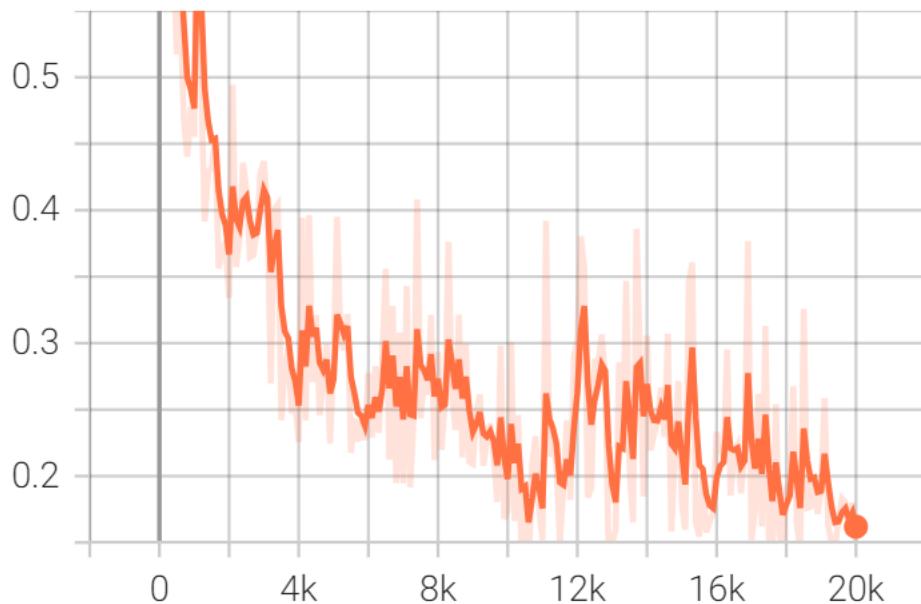


FIGURE 3.3 : TensorBoard d'entraînement du modèle EfficientDet-D0 après collecte supplémentaire

3.5 Test sur le Raspberry Pi avec le modèle SSD-Mobilenet-v2-fpnlite-320

Après obtention de notre modèle, nous avons procédé à une conversion puis à son intégration sur une Raspberry Pi 3. Les figures 3.4, 3.7, 3.8 montrent quelques tests qui ont été effectués :

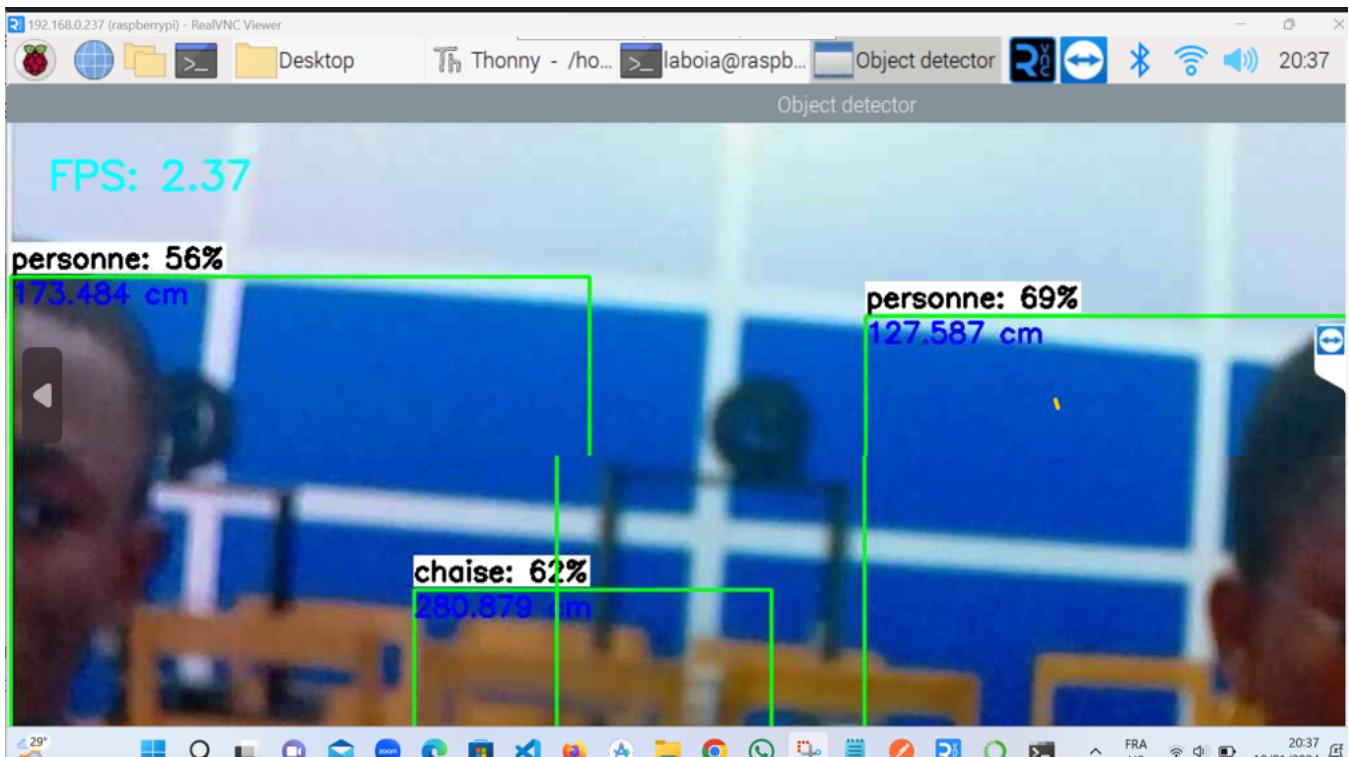


FIGURE 3.4 : Détection d'une personne à 1,27 m



FIGURE 3.5 : Détection d'une Moto à 4,39 m avant la collecte supplémentaire



FIGURE 3.6 : Détection d'une Moto à 5,68 m après collecte supplémentaire



FIGURE 3.7 : Détection d'une Voiture à 2,678 m avant la collecte supplémentaire

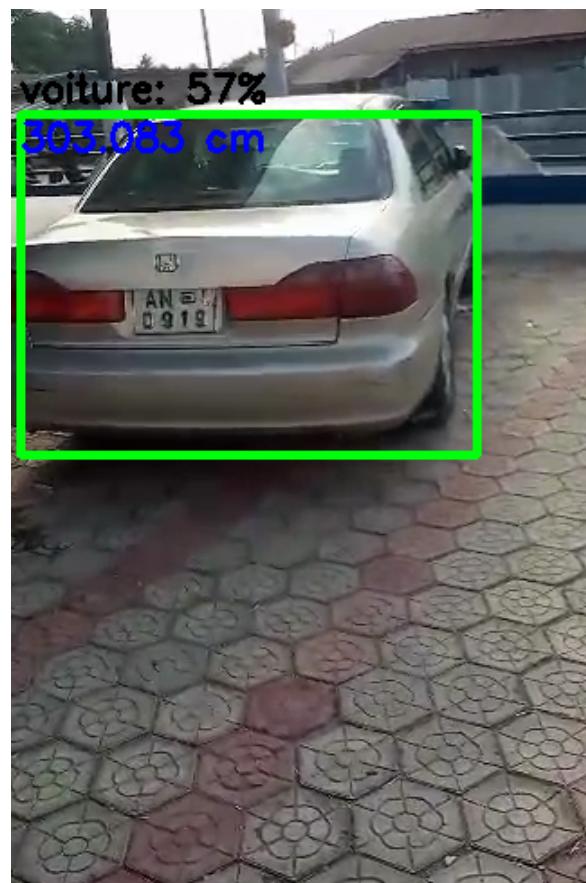


FIGURE 3.8 : Détection d'une Voiture à 3,03 m après collecte supplémentaire

Une fois la collecte de données supplémentaire nous remarquons que le modèle améliore la delimitation par une globante des objets détectés ce qui par conséquent améliore la probabilité de détection et la valeur de la distance.

3.6 Prototype expérimentale du système d'autoguidage



FIGURE 3.9 : Prototype du système vue de face



FIGURE 3.10 : Prototype du système vu d'en haut

Le prototype expérimental (voir Figure 3.9 et Figure 3.10) est un chapeau muni d'une caméra ayant pour support un PAL (un plastique biosourcé et biodégradable) située à l'avant, capturant les flux vidéo. À l'intérieur, il comprend une Raspberry Pi embarquée d'un système intelligent comprenant un module de détection d'objets et de synthèse vocale. Le système est également équipé d'un port audio pour la sortie audio des écouteurs, d'une batterie pour alimenter la Raspberry Pi, et d'un PAL servant de support pour la Raspberry Pi. De plus, il est équipé d'un interrupteur relié à la Raspberry Pi via un câble jumper, agissant comme un interrupteur marche/arrêt pour la Raspberry Pi.

Conclusion

Les données sont essentielles pour garantir un système performant et efficace. En effet, la qualité des données peut souvent avoir un impact plus significatif que la sophistication de l'algorithme utilisé. Actuellement, notre modèle de détection d'obstacles a atteint une précision de **95,46%** sur l'ensemble d'entraînement, **96,20%** sur l'ensemble de validation, et **98,75%** sur l'ensemble de test.

Conclusion Générale et Perspectives

Le déplacement autonome représente un enjeu majeur pour les personnes malvoyantes, leur offrant la liberté et l'indépendance nécessaires pour mener une vie quotidienne épanouissante. Cependant, ce défi est souvent entravé par les obstacles environnementaux auxquels elles sont confrontées. Dans ce contexte, l'accès à un système d'autoguidage efficace revêt une importance cruciale, permettant à ces individus de naviguer en toute sécurité et avec assurance dans leur environnement. Bien que les progrès technologiques récents aient apporté des améliorations significatives, les solutions actuelles présentent encore des limitations, telles que le manque de possibilité de détection d'objets en temps réel et d'interaction, une portée de détection restreinte et une fiabilité des retours haptiques compromise. De plus, l'intégration limitée à la simple détection d'obstacles limite la capacité des utilisateurs à naviguer efficacement. Dans le cadre de notre travail, nous avons exploré plusieurs approches basées sur l'apprentissage profond pour la détection et la classification en temps réel d'objets dans un flux vidéo. En utilisant des techniques avancées telles que l'augmentation de données et l'apprentissage par transfert, nous avons développé un modèle robuste. SSD-MobileNet-v2-fpnlite, avec la stratégie de fine-tuning a atteint une précision 95,46% sur l'ensemble d'entraînement, 96,20% sur l'ensemble de validation, et 98,75% sur l'ensemble de test.

En plus de la détection des objets, notre système intègre également un calcul de la distance entre la caméra et les objets détectés, ce qui permet de prioriser les obstacles proches et d'améliorer la sécurité des utilisateurs. Ce modèle a été intégré avec succès sur une Raspberry Pi, offrant ainsi une solution compacte et portable. Grâce à un module de synthèse vocale intégré, notre système fournit des instructions audios claires et concises aux utilisateurs.

Pour garantir la pertinence continue et l'évolution de notre système, plusieurs pistes de développement sont envisagées :

- Intégrer un modèle de traitement du langage naturel permettrait une interaction plus intuitive avec le système ;
- Intégrer un système de navigation, offrant aux personnes malvoyantes la possibilité de suivre des itinéraires en extérieur de manière autonome

Bibliographie

- [1] "Les personnes ayant un handicap visuel - Les apports de l'enquête Handicaps - Incapacités - Dépendance", 2005.
- [2] "Réseau de neurones artificiels : qu'est-ce que c'est et à quoi ça sert?", RESEAU ACTU. (6 avr. 2019), adresse : <https://eldorhaan.wordpress.com/2019/04/06/reseau-de-neurones-artificiels-quest-ce-que-cest-et-a-quoi-ca-sert/> (visité le 23/01/2024).
- [3] "Artificial neural network tutorial - javatpoint", www.javatpoint.com. (), adresse : <https://www.javatpoint.com/artificial-neural-network> (visité le 02/03/2024).
- [4] "A free online introduction to artificial intelligence for non-experts", Elements of AI. (), adresse : <https://course.elementsofai.com/> (visité le 23/01/2024).
- [5] H. WELTER. "Deep learning : définition, fonctionnement et applications". (), adresse : <https://blog.hubspot.fr/marketing/deep-learning> (visité le 02/03/2024).
- [6] C. DELUZARCHE. "Définition | Deep Learning - Apprentissage profond | Futura Tech", Futura. (), adresse : <https://www.futura-sciences.com/tech/definitions/intelligence-artificielle-deep-learning-17262/> (visité le 23/01/2024).
- [7] A. G. HOWARD, M. ZHU, B. CHEN et al., *MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications*, 16 avr. 2017. arXiv : 1704.04861 [cs]. adresse : <http://arxiv.org/abs/1704.04861> (visité le 02/03/2024).
- [8] M. SANDLER, A. HOWARD, M. ZHU, A. ZHMOGINOV et L.-C. CHEN, "MobileNetV2 : inverted residuals and linear bottlenecks", in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT : IEEE, juin 2018, p. 4510-4520, ISBN : 978-1-5386-6420-9. DOI : 10.1109/CVPR.2018.00474. adresse : <https://ieeexplore.ieee.org/document/8578572/> (visité le 02/03/2024).
- [9] J. REN et Y. WANG, "Overview of object detection algorithms using convolutional neural networks", *Journal of Computer and Communications*, t. 10, n° 1, p. 115-132, 12 jan. 2022, Number : 1 Publisher : Scientific Research Publishing. DOI : 10.4236/jcc.2022.101006. adresse : <https://www.scirp.org/journal/paperinformation.aspx?paperid=115011> (visité le 02/03/2024).

-
- [10] “Détection d’objets à l’aide de YOLO et Mobilenet SSD - Plato Data Intelligence”. Section : Big Data. (22 sept. 2022), adresse : <https://zephyrnet.com/fr/d%C3%A9tection-d%27objets-%C3%A0-1%27aide-de-yolo-et-mobilenet-ssd/> (visité le 02/03/2024).
- [11] W. LIU, D. ANGUELOV, D. ERHAN et al., “SSD : Single Shot MultiBox Detector”, in t. 9905, 2016, p. 21-37. DOI : [10.1109/ICCV.2015.73](https://doi.org/10.1109/ICCV.2015.73). arXiv : [1512.02325 \[cs\]](https://arxiv.org/abs/1512.02325). adresse : [http://arxiv.org/abs/1512.02325](https://arxiv.org/abs/1512.02325) (visité le 02/03/2024).
- [12] J. COHEN, C. F. CRISPIM-JUNIOR, J.-M. CHIAPPA et L. TOUGNE, “MobileNet SSD : étude d’un détecteur d’objets embarquable entraîné sans images réelles”, in ORASIS 2021, Saint Ferréol, France : Centre National de la Recherche Scientifique [CNRS], sept. 2021. adresse : <https://hal.science/hal-03531390> (visité le 02/03/2024).
- [13] R. J. WANG, X. LI et C. X. LING, “Pelee : A Real-Time Object Detection System on Mobile Devices”, in *Advances in Neural Information Processing Systems*, t. 31, Curran Associates, Inc., 2018. adresse : https://proceedings.neurips.cc/paper_files/paper/2018/hash/9908279ebbf1f9b250ba689db6a0222b-Abstract.html (visité le 02/03/2024).
- [14] “Depth wise separable convolutional neural networks”, GeeksforGeeks. Section : Machine Learning. (18 sept. 2018), adresse : <https://www.geeksforgeeks.org/depth-wise-separable-convolutional-neural-networks/> (visité le 03/03/2024).
- [15] K. BAPAT. “Introduction to MobileNet v1 using Depth Wise Separable Convolution”. (), adresse : <https://krutikabapat.github.io/MobileNet-And-Depth-Wise-Separable-Convolutions/> (visité le 03/03/2024).
- [16] “MobileNetV2 SSD FPN”. (), adresse : <https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/mobilenetv2-ssd-fpn> (visité le 23/01/2024).
- [17] T. DUTOIT, “High-quality text-to-speech synthesis : an overview.”,
- [18] gTTS : gTTS (*Google Text-to-Speech*), a Python library and CLI tool to interface with Google Translate text-to-speech API, version 2.5.1.
- [19] “Cécité : définition, causes et traitements”, Elsan. (), adresse : <https://www.elsan-care.fr/pathologie-et-traitement/maladies-des-yeux/cecite-symptomes-causes> (visité le 08/03/2024).
- [20] R&D DEVELOPMENT ENGINEER, DASSAULT SYSTEMES R&D, PUNE, INDIA., S. SHINDE, M. MUNOT et al., “Intelligent companion for blind : smart stick”, *International Journal of Innovative Technology and Exploring Engineering*, t. 8, n° 10, p. 4251-4256, 30 août 2019, ISSN : 22783075. DOI : [10.35940/ijitee.J9957.0881019](https://doi.org/10.35940/ijitee.J9957.0881019). adresse : <https://www.ijitee.org/portfolio-item/J99570881019/> (visité le 23/01/2024).
- [21] D. DE ALWIS et Y. SAMARAWICKRAMA, “Low cost ultrasonic based wide detection range smart walking stick for visually impaired”, *International Journal of Multidisciplinary Studies*, t. 3, n° 2, p. 123, 28 jan. 2017, ISSN : 2362-079X. DOI : [10.4038/ijms.v3i2.14](https://doi.org/10.4038/ijms.v3i2.14). adresse : <https://ijms.sjol.info/article/10.4038/ijms.v3i2.14/> (visité le 23/01/2024).
- [22] D. S. SWAMY et A. ROY, “Design and development of smart belt for blind”, t. 7, n° 6, 2020.

-
- [23] "Sensors | Free Full-Text | Wearable Smart System for Visually Impaired People". (), adresse : <https://www.mdpi.com/1424-8220/18/3/843> (visité le 23/01/2024).
 - [24] CAROLE. "12 applications pour les personnes aveugles ou malvoyantes", Le Webzine OKEENEA. (20 oct. 2020), adresse : <https://webzine.okeenea.com/applications-personnes-aveugles-malvoyantes/> (visité le 06/06/2024).
 - [25] "Clustering : Definition, avantages et fonctionne". (), adresse : <https://www.50a.fr/0/clustering> (visité le 23/01/2024).
 - [26] "Introduction to TensorFlow lite", GeeksforGeeks. Section : Python. (12 mai 2021), adresse : <https://www.geeksforgeeks.org/introduction-to-tensorflow-lite/> (visité le 23/01/2024).
 - [27] "Why and how to normalize data for computer vision (with PyTorch)". Section : Deep Learning. (11 juin 2021), adresse : <https://inside-machinelearning.com/en/why-and-how-to-normalize-data-object-detection-on-image-in-pytorch-part-1/> (visité le 05/03/2024).
 - [28] K. TEAM. "Keras documentation : learning to resize in computer vision". (), adresse : https://keras.io/examples/vision/learnable_resizer/ (visité le 05/03/2024).
 - [29] S.-H. TSANG. "Review — learning to resize images for computer vision tasks (image classification & image quality... ", Medium. (15 jan. 2022), adresse : <https://sh-tsang.medium.com/review-learning-to-resize-images-for-computer-vision-tasks-image-classification-image-quality-c05cbe284bc6> (visité le 05/03/2024).
 - [30] J. ROBERT. "Data augmentation : Qu'est-ce que c'est ? À quoi ça sert?", Formation Data Science | DataScientest.com. (1^{er} sept. 2023), adresse : <https://datascientest.com/data-augmentation-tout-savoir> (visité le 06/03/2024).
 - [31] "Qu'est-ce que l'apprentissage non supervisé ? | Linedata". (), adresse : <https://fr.linedata.com/quest-ce-que-lapprentissage-non-supervise> (visité le 23/01/2024).
 - [32] P. HALLAJ. "Data augmentation : benefits and disadvantages", Medium. (20 sept. 2023), adresse : <https://medium.com/@pouyahallaj/data-augmentation-benefits-and-disadvantages-38d8201aead> (visité le 06/03/2024).
 - [33] "A complete guide to data augmentation". (), adresse : <https://www.datacamp.com/tutorial/complete-guide-data-augmentation> (visité le 06/03/2024).
 - [34] "Étiqueter les objets pour le Deep Learning—ArcGIS Pro | Documentation". (), adresse : <https://pro.arcgis.com/fr/pro-app/3.1/help/analysis/image-analyst/label-objects-for-deep-learning.htm> (visité le 06/03/2024).
 - [35] G. BOESCH. "LabelImg for Image Annotation", viso.ai. (11 fév. 2022), adresse : <https://viso.ai/computer-vision/labelimg-for-image-annotation/> (visité le 06/03/2024).
 - [36] N. MALINGAN. "EfficientNet", Scaler Topics. (19 avr. 2023), adresse : <https://www.scaler.com/topics/deep-learning/efficientNet/> (visité le 07/03/2024).

-
- [37] A. SARKAR. "EfficientNetV2 — faster, smaller, and higher accuracy than Vision Transformers", Medium. (8 oct. 2022), adresse : <https://towardsdatascience.com/efficientnetv2-faster-smaller-and-higher-accuracy-than-vision-transformers-98e23587bf04> (visité le 07/03/2024).
 - [38] S.-H. TSANG. "Review : MobileNetV2 — light weight model (image classification)", Medium. (1^{er} août 2019), adresse : <https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c> (visité le 07/03/2024).
 - [39] "Image Classification With MobileNet", Built In. (), adresse : <https://builtin.com/machine-learning/mobilenet> (visité le 12/06/2024).
 - [40] "[U+FFFD] [U+FFFD] [U+FFFD] [U+FFFD] VGG vs ResNet vs Inception vs MobileNet [U+FFFD] [U+FFFD] [U+FFFD] [U+FFFD] | Kaggle". (), adresse : <https://www.kaggle.com/discussions/getting-started/a> (visité le 12/06/2024).
 - [41] J. HUI. "Object detection : speed and accuracy comparison (faster r-CNN, r-FCN, SSD, FPN, RetinaNet and...)", Medium. (26 mars 2019), adresse : <https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359> (visité le 07/03/2024).
 - [42] "YOLOv8 vs SSD : Choosing the Right Object Detection Model", Keylabs : latest news and updates. (22 déc. 2023), adresse : <https://keylabs.ai/blog/yolov8-vs-ssd-choosing-the-right-object-detection-model/> (visité le 12/06/2024).
 - [43] "Realtime Distance Estimation Using OpenCV - Python - GeeksforGeeks". (), adresse : <https://www.geeksforgeeks.org/realtime-distance-estimation-using-opencv-python/> (visité le 02/02/2024).

Annexe

Annexe 1 : Matériel



FIGURE 3.11 : Image d'une Raspberry avec ses caractéristiques

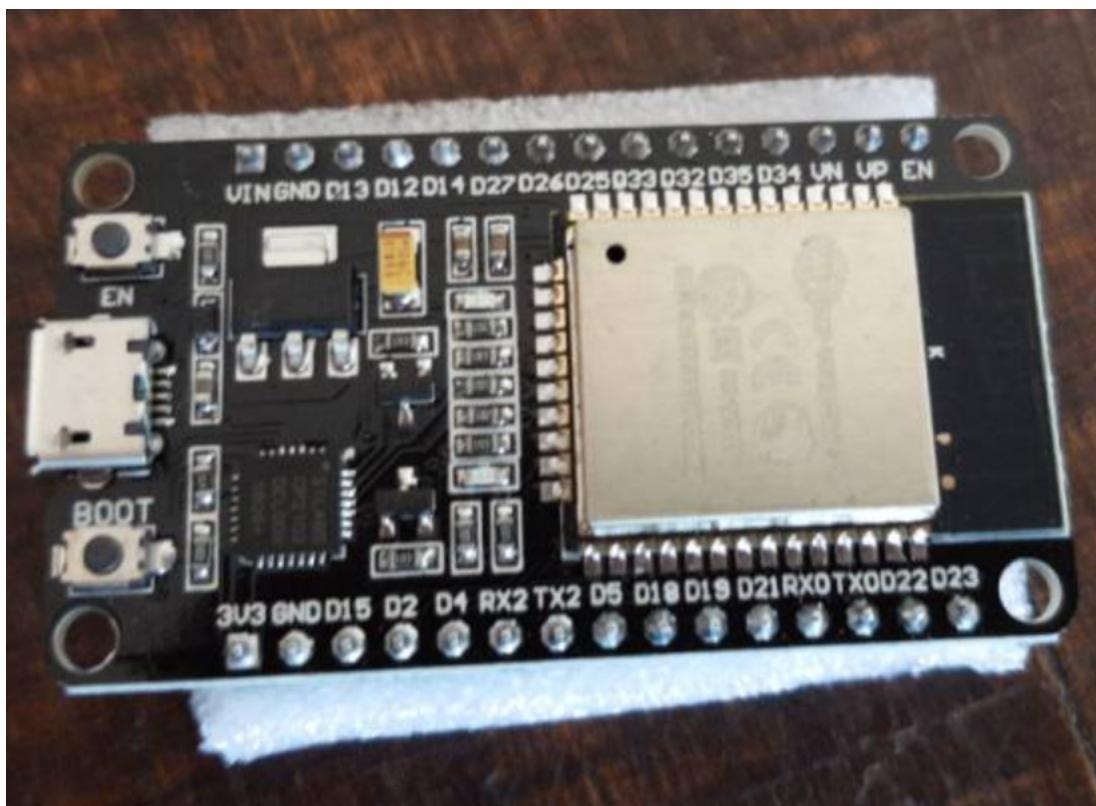


FIGURE 3.12 : Image d'un ESP32 devkit v1



FIGURE 3.13 : Image d'un jumper male-femelle

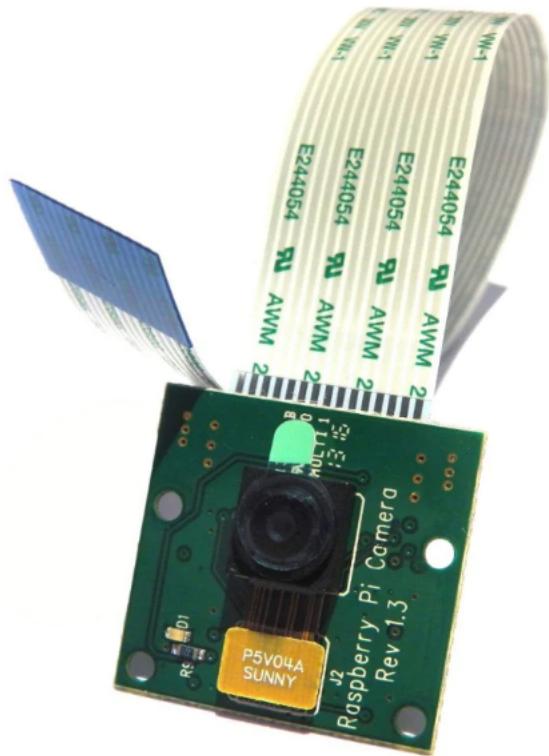


FIGURE 3.14 : Image d'une caméra Raspberry Pi Rev v1.3

Annexe 2 : Mise en place du modèle de détection d'objet

Installer les dépendances pour la détection d'objets avec TensorFlow

```
# Clone the tensorflow models repository from GitHub
!pip uninstall Cython -y # Temporary fix for "No module named 'object_detection'" error
!git clone --depth 1 https://github.com/tensorflow/models

# Copy setup files into models/research folder
%%bash
cd models/research/
protoc object_detection/protos/*.proto --python_out=.
#cp object_detection/packages/tf2/setup.py .

# Modify setup.py file to install the tf-models-official repository targeted at TF v2.8.0
import re
with open('/content/models/research/object_detection/packages/tf2/setup.py') as f:
    s = f.read()

with open('/content/models/research/setup.py', 'w') as f:
    # Set fine_tune_checkpoint path
    s = re.sub('tf-models-official>=2.5.1',
               'tf-models-official==2.8.0', s)
    f.write(s)
```

FIGURE 3.15 : Installer les Dépendances pour la Détection d'Objets avec TensorFlow

```
# Install the Object Detection API (NOTE:
#This block takes about 10 minutes to finish executing)

# Need to do a temporary fix with PyYAML because Colab isn't able to install PyYAML v5.4.1
!pip install pyyaml==5.3
!pip install /content/models/research/

# Need to downgrade to TF v2.8.0 due to Colab compatibility bug with TF v2.10 (as of 10/03/22)
!pip install tensorflow==2.8.0

# Install CUDA version 11.0 (to maintain compatibility with TF v2.8.0)
!pip install tensorflow_io==0.23.1
!wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/cuda-ubuntu1804.pin
!mv cuda-ubuntu1804.pin /etc/apt/preferences.d/cuda-repository-pin-600
!wget http://developer.download.nvidia.com/compute/cuda/11.0.2/local_installers/cuda-repo-ubuntu1804-11-0-local_11.0.2-450.51.05-1_amd64.deb
!dpkg -i cuda-repo-ubuntu1804-11-0-local_11.0.2-450.51.05-1_amd64.deb
!apt-key add /var/cuda-repo-ubuntu1804-11-0-local/7fa2af80.pub
!apt-get update && sudo apt-get install cuda-toolkit-11-0
!export LD_LIBRARY_PATH=/usr/local/cuda-11.0/lib64:$LD_LIBRARY_PATH

# Run Model Bulider Test file, just to verify everything's working properly
!python /content/models/research/object_detection/builders/model_builder_tf2_test.py
```

FIGURE 3.16 : Installer les Dépendances pour la Détection d'Objets avec TensorFlow

```

# Upload images
from google.colab import drive
drive.mount('/content/gdrive')
!cp /content/gdrive/MyDrive/path/to/images.zip /content

# Split images into train, validation, and test folders
!mkdir /content/images
!unzip -q images.zip -d /content/images/all
!mkdir /content/images/train; mkdir /content/images/validation; mkdir /content/images/test

!wget https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/master/util_scripts/train_val_test_split.py
!python train_val_test_split.py

```

FIGURE 3.17 : Téléchargez l'ensemble de données d'image et préparez à la formation

```

# Create Labelmap and TFRecords
### This creates a "labelmap.txt" file with a list of classes the object detection model will detect.
%%bash
cat <<EOF >> /content/labelmap.txt
person
bicycle
car
motorcycle
airplane
bus
train
truck
boat
traffic light
fire hydrant
stop sign
parking meter
bench
bird
cat
dog
horse

```

FIGURE 3.18 : Téléchargez l'ensemble de données d'image et préparez à la formation

```

horse
sheep
cow
elephant
bear
zebra
giraffe
backpack
umbrella
handbag
tie
suitcase
frisbee
skis
snowboard
sports ball
kite
baseball bat
baseball glove
skateboard
surfboard
tennis racket
bottle
wine glass

```

FIGURE 3.19 : Téléchargez l'ensemble de données d'image et préparez à la formation

```
cup
fork
knife
spoon
bowl
banana
apple
sandwich
orange
broccoli
carrot
hot dog
pizza
donut
cake
chair
couch
potted plant
bed
dining table
toilet
tv
laptop
mouse
remote
keyboard
cell phone
microwave
oven
toaster
sink
refrigerator
book
clock
vase
scissors
teddy bear
hair drier
toothbrush

EOF
```

FIGURE 3.20 : Téléchargez l'ensemble de données d'image et préparez à la formation

Téléchargez l'ensemble de données d'image et préparez à la formation

```
# Download data conversion scripts
! wget https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/master/util_scripts/create_csv.py
! wget https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/master/util_scripts/create_tfrecord.py

# Create CSV data files and TFRecord files
!python3 create_csv.py
!python3 create_tfrecord.py --csv_input=images/train_labels.csv --labelmap=labelmap.txt --image_dir=images/train --output_path=train.tfrecord
!python3 create_tfrecord.py --csv_input=images/validation_labels.csv --labelmap=labelmap.txt --image_dir=images/validation --output_path=val.tfrecord

train_record_fname = '/content/train.tfrecord'
val_record_fname = '/content/val.tfrecord'
label_map_pbtxt_fname = '/content/labelmap.pbtxt'
```

FIGURE 3.21 : Téléchargez l'ensemble de données d'image et préparez à la formation

Configuration de l'entraînement du modèle

```
# Change the chosen_model variable to deploy different models available in the TF2 object detection zoo
chosen_model = 'ssd-mobilenet-v2-fpnlite-320'

MODELS_CONFIG = {
    'ssd-mobilenet-v2': {
        'model_name': 'ssd_mobilenet_v2_320x320_coco17_tpu-8',
        'base_pipeline_file': 'ssd_mobilenet_v2_320x320_coco17_tpu-8.config',
        'pretrained_checkpoint': 'ssd_mobilenet_v2_320x320_coco17_tpu-8.tar.gz',
    },
    'efficientdet-d0': {
        'model_name': 'efficientdet_d0_coco17_tpu-32',
        'base_pipeline_file': 'ssd_efficientdet_d0_512x512_coco17_tpu-8.config',
        'pretrained_checkpoint': 'efficientdet_d0_coco17_tpu-32.tar.gz',
    },
    'ssd-mobilenet-v2-fpnlite-320': {
        'model_name': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8',
        'base_pipeline_file': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config',
        'pretrained_checkpoint': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz',
    },
    # The centernet model isn't working as of 9/10/22
    #'centernet-mobilenet-v2': {
    #    'model_name': 'centernet_mobilenetv2fpn_512x512_coco17_od',
    #    'base_pipeline_file': 'pipeline.config',
    #    'pretrained_checkpoint': 'centernet_mobilenetv2fpn_512x512_coco17_od.tar.gz',
    #}
}
```

FIGURE 3.22 : Configuration de l'entraînement du modèle

```
model_name = MODELS_CONFIG[chosen_model]['model_name']
pretrained_checkpoint = MODELS_CONFIG[chosen_model]['pretrained_checkpoint']
base_pipeline_file = MODELS_CONFIG[chosen_model]['base_pipeline_file']

# Create "mymodel" folder for holding pre-trained weights and configuration files
%mkdir /content/models/mymodel/
%cd /content/models/mymodel/

# Download pre-trained model weights
import tarfile
download_tar = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/' + pretrained_checkpoint
!wget {download_tar}
tar = tarfile.open(pretrained_checkpoint)
tar.extractall()
tar.close()

# Download training configuration file for model
download_config = 'https://raw.githubusercontent.com/tensorflow/models/master/research/object_detection/configs/tf2/' + base_pipeline_file
!wget {download_config}

# Set training parameters for the model
num_steps = 40000

if chosen_model == 'efficientdet-d0':
    batch_size = 4
else:
    batch_size = 16

# Set file locations and get number of classes for config file
pipeline_fname = '/content/models/mymodel/' + base_pipeline_file
fine_tune_checkpoint = '/content/models/mymodel/' + model_name + '/checkpoint/ckpt-0'

def get_num_classes(pbtxt_fname):
    from object_detection.utils import label_map_util
    label_map = label_map_util.load_labelmap(pbtxt_fname)
    label_map = label_map_util.load_labelmap(label_map_pbtxt_fname)
    categories = label_map_util.convert_label_map_to_categories(
        label_map, max_num_classes=90, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)
    return len(category_index.keys())
num_classes = get_num_classes(label_map_pbtxt_fname)
print('Total classes:', num_classes)
```

FIGURE 3.23 : Configuration de l'entraînement du modèle

```

# Create custom configuration file by writing the dataset, model checkpoint, and training parameters into the base pipeline file
import re

%cd /content/models/mymodel
print('writing custom configuration file')

with open(pipeline_fname) as f:
    s = f.read()
with open('pipeline_file.config', 'w') as f:

    # Set fine_tune_checkpoint path
    s = re.sub('fine_tune_checkpoint: ".*?"',
               'fine_tune_checkpoint: "{}".format(fine_tune_checkpoint), s)

    # Set tfrecord files for train and test datasets
    s = re.sub(
        'input_path: "(.*?)(PATH_TO_BE_CONFIGURED/train)(.*?)"', 'input_path: "{}".format(train_record_fname), s')
    s = re.sub(
        'input_path: "(.*?)(PATH_TO_BE_CONFIGURED/val)(.*?)"', 'input_path: "{}".format(val_record_fname), s')

    # Set label_map_path
    s = re.sub(
        'label_map_path: ".*?"', 'label_map_path: "{}".format(label_map_pbtxt_fname), s')

    # Set batch_size
    s = re.sub('batch_size: [0-9]+',
               'batch_size: {}'.format(batch_size), s)

    # Set training steps, num_steps
    s = re.sub('num_steps: [0-9]+',
               'num_steps: {}'.format(num_steps), s)

    # Set number of classes num_classes
    num_classes=4
    s = re.sub('num_classes: [0-9]+',
               'num_classes: {}'.format(num_classes), s)

    # Change fine-tune checkpoint type from "classification" to "detection"
    s = re.sub(
        'fine_tune_checkpoint_type: "classification"', 'fine_tune_checkpoint_type: "{}".format("detection"), s')

    # If using ssd-mobilenet-v2, reduce learning rate (because it's too high in the default config file)
    if chosen_model == 'ssd-mobilenet-v2':
        s = re.sub('learning_rate_base: .8',
                   'learning_rate_base: .08', s)

```

FIGURE 3.24 : Configuration de l’entraînement du modèl

```

s = re.sub('warmup_learning_rate: 0.13333',
           'warmup_learning_rate: .026666', s)

# If using efficientdet-d0, use fixed_shape_resizer instead of keep_aspect_ratio_resizer (because it isn't supported by TFLite)
if chosen_model == 'efficientdet-d0':
    s = re.sub('keep_aspect_ratio_resizer', 'fixed_shape_resizer', s)
    s = re.sub('pad_to_max_dimension: true', '', s)
    s = re.sub('min_dimension', 'height', s)
    s = re.sub('max_dimension', 'width', s)

f.write(s)

# (Optional) Display the custom configuration file's contents
!cat /content/models/mymodel/pipeline_file.config

# Set the path to the custom config file and the directory to store training checkpoints in
pipeline_file = '/content/models/mymodel/pipeline_file.config'
model_dir = '/content/training/'

```

FIGURE 3.25 : Configuration de l’entraînement du modèl

Entraîner un modèle

▼ Entrainer un modèle

```
[ ] %load_ext tensorboard  
%tensorboard --logdir '/content/training/train'  
  
# Run training!  
!python /content/models/research/object_detection/model_main_tf2.py \  
--pipeline_config_path={pipeline_file} \  
--model_dir={model_dir} \  
--alsologtosterr \  
--num_train_steps={num_steps} \  
--sample_1_of_n_eval_examples=1
```

FIGURE 3.26 : Entrainer un modèle

Convertir le modèle en TensorFlow Lite

```
# Make a directory to store the trained TFLite model  
!mkdir /content/custom_model_lite  
output_directory = '/content/custom_model_lite'  
  
# Path to training directory (the conversion script automatically chooses the highest checkpoint file)  
last_model_path = '/content/training'  
  
!python /content/models/research/object_detection/export_tflite_graph_tf2.py \  
--trained_checkpoint_dir {last_model_path} \  
--output_directory {output_directory} \  
--pipeline_config_path {pipeline_file}  
  
# Convert exported graph file into TFLite model file  
import tensorflow as tf  
  
converter = tf.lite.TFLiteConverter.from_saved_model('/content/custom_model_lite/saved_model')  
tflite_model = converter.convert()  
  
with open('/content/custom_model_lite/detect.tflite', 'wb') as f:  
    f.write(tflite_model)
```

FIGURE 3.27 : Convertir le modèle en TensorFlow Lite

Testez le modèle TensorFlow Lite et calculez mAP

```
# Inference test images

# Script to run custom TFLite model on test images to detect objects
# Source: https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/TFLite\_detection\_image.py

# Import packages
import os
import cv2
import numpy as np
import sys
import glob
import random
import importlib.util
from tensorflow.lite.python.interpreter import Interpreter

import matplotlib
import matplotlib.pyplot as plt

%matplotlib inline

### Define function for inferencing with TFLite model and displaying results

def tflite_detect_images(modelpath, imgpath, lblpath, min_conf=0.5, num_test_images=10, savepath='/content/results', txt_only=False):

    # Grab filenames of all images in test folder
    images = glob.glob(imgpath + '*.jpg') + glob.glob(imgpath + '*.JPG') + glob.glob(imgpath + '*.png') + glob.glob(imgpath + '*.bmp')

    # Load the label map into memory
    with open(lblpath, 'r') as f:
        labels = [line.strip() for line in f.readlines()]

    # Load the Tensorflow Lite model into memory
    interpreter = Interpreter(model_path=modelpath)
    interpreter.allocate_tensors()

    # Get model details
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    height = input_details[0]['shape'][1]
    width = input_details[0]['shape'][2]

    float_input = (input_details[0]['dtype'] == np.float32)

    input_mean = 127.5
    input_std = 127.5

    # Randomly select test images
    images_to_test = random.sample(images, num_test_images)
```

FIGURE 3.28 : Testez le modèle TensorFlow Lite et calculez mAP

```

# Loop over every image and perform detection
for image_path in images_to_test:

    # Load image and resize to expected shape [1xHxWx3]
    image = cv2.imread(image_path)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    imH, imW, _ = image.shape
    image_resized = cv2.resize(image_rgb, (width, height))
    input_data = np.expand_dims(image_resized, axis=0)

    # Normalize pixel values if using a floating model (i.e. if model is non-quantized)
    if float_input:
        input_data = (np.float32(input_data) - input_mean) / input_std

    # Perform the actual detection by running the model with the image as input
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()

    # Retrieve detection results
    boxes = interpreter.get_tensor(output_details[1]['index'])[0] # Bounding box coordinates of detected objects
    classes = interpreter.get_tensor(output_details[3]['index'])[0] # Class index of detected objects
    scores = interpreter.get_tensor(output_details[0]['index'])[0] # Confidence of detected objects

    detections = []

    # Loop over all detections and draw detection box if confidence is above minimum threshold
    for i in range(len(scores)):
        if ((scores[i] > min_conf) and (scores[i] <= 1.0)):

            # Get bounding box coordinates and draw box
            # Interpreter can return coordinates that are outside of image dimensions, need to force them to be within image using max() and min()
            ymin = int(max(1,(boxes[i][0] * imH)))
            xmin = int(max(1,(boxes[i][1] * imW)))
            ymax = int(min(imH,(boxes[i][2] * imH)))
            xmax = int(min(imW,(boxes[i][3] * imW)))

            cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

            # Draw label
            object_name = labels[int(classes[i])] # Look up object name from "labels" array using class index
            label = '%s: %s%%' % (object_name, int(scores[i]*100)) # Example: 'person: 72%'
            labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Get font size
            label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label too close to top of window
            cv2.rectangle(image, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw white box to put label text in
            cv2.putText(image, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text

            detections.append([object_name, scores[i], xmin, ymin, xmax, ymax])

```

FIGURE 3.29 : Testez le modèle TensorFlow Lite et calculez mAP

```

# All the results have been drawn on the image, now display the image
if txt_only == False: # "text_only" controls whether we want to display the image results or just save them in .txt files
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(12,16))
    plt.imshow(image)
    plt.show()

# Save detection results in .txt files (for calculating mAP)
elif txt_only == True:

    # Get filenames and paths
    image_fn = os.path.basename(image_path)
    base_fn, ext = os.path.splitext(image_fn)
    txt_result_fn = base_fn +'.txt'
    txt_savepath = os.path.join(savepath, txt_result_fn)

    # Write results to text file
    # (Using format defined by https://github.com/cartucho/mAP, which will make it easy to calculate mAP)
    with open(txt_savepath,'w') as f:
        for detection in detections:
            f.write('%s %.4f %d %d %d %d\n' % (detection[0], detection[1], detection[2], detection[3], detection[4], detection[5]))

return

# set up variables for running user's model
PATH_TO_IMAGES='/content/images/test' # Path to test images folder
PATH_TO_MODEL='/content/custom_model_lite/detect.tflite' # Path to .tflite model file
PATH_TO_LABELS='/content/labelmap.txt' # Path to labelmap.txt file
min_conf_threshold=0.5 # Confidence threshold (try changing this to 0.01 if you don't see any detection results)
images_to_test = 10 # Number of images to run detection on

# Run inferencing function!
tfLite_detect_images(PATH_TO_MODEL, PATH_TO_IMAGES, PATH_TO_LABELS, min_conf_threshold, images_to_test)

#Calculate mAP
%%bash
git clone https://github.com/cartucho/mAP /content/mAP
cd /content/mAP
rm input/detection-results/*
rm input/ground-truth/*
rm input/images-optional/*
wget https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/master/util\_scripts/calculate\_map\_cartucho.py

!cp /content/images/test/* /content/mAP/input/images-optional # Copy images and xml files
!mv /content/mAP/input/images-optional/*.xml /content/mAP/input/ground-truth/ # Move xml files to the appropriate folder
!python /content/mAP/scripts/extra/convert_gt_xml.py

```

FIGURE 3.30 : Testez le modèle TensorFlow Lite et calculez mAP

```

# Set up variables for running inference, this time to get detection results saved as .txt files
PATH_TO_IMAGES='/content/images/test' # Path to test images folder
PATH_TO_MODEL='/content/custom_model_lite/detect.tflite' # Path to .tflite model file
PATH_TO_LABELS='/content/labelmap.txt' # Path to labelmap.txt file
PATH_TO_RESULTS='/content/mAP/input/detection-results' # Folder to save detection results in
min_conf_threshold=0.1 # Confidence threshold

# Use all the images in the test folder
image_list = glob.glob(PATH_TO_IMAGES + '/*.jpg') + glob.glob(PATH_TO_IMAGES + '/*.JPG') + glob.glob(PATH_TO_IMAGES + '/*.png') + glob.glob(PATH_TO_IMAGES + '/*.bmp')
images_to_test = min(500, len(image_list)) # If there are more than 500 images in the folder, just use 500

# Tell function to just save results and not display images
txt_only = True

# Run inferencing function!
print('Starting inference on %d images...' % images_to_test)
tfLite_detect_images(PATH_TO_MODEL, PATH_TO_IMAGES, PATH_TO_LABELS, min_conf_threshold, images_to_test, PATH_TO_RESULTS, txt_only)
print('Finished inferencing!')

%cd /content/mAP
!python calculate_map_cartucho.py --labels=/content/labelmap.txt

```

FIGURE 3.31 : Testez le modèle TensorFlow Lite et calculez mAP

Annexe 3 :Déploiement du modèle sur un Raspberry Pi

```
16     # Import packages
17     import os
18     import argparse
19     import cv2
20     import numpy as np
21     import sys
22     import time
23     from threading import Thread
24     #import pygame
25     import importlib.util
26
27     #gtts pour la traduction
28     from gtts import gTTS
29
30     # Audio
31     import io
32     from pydub import AudioSegment
33     from pydub.playback import play
34
```

FIGURE 3.32 : Deploiement du modèle sur la Raspberry Pi

```
41
42 class VideoStream:
43     """Camera object that controls video streaming from the Picamera"""
44     def __init__(self,resolution=(640,480),framerate=30):
45         # Initialize the PiCamera and the camera image stream
46         self.stream = cv2.VideoCapture(0)
47         ret = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
48         ret = self.stream.set(3,resolution[0])
49         ret = self.stream.set(4,resolution[1])
50
51         # Read first frame from the stream
52         (self.grabbed, self.frame) = self.stream.read()
53
54     # Variable to control when the camera is stopped
55     self.stopped = False
56
57     def start(self):
58         # Start the thread that reads frames from the video stream
59         Thread(target=self.update,args=()).start()
60         return self
61
62     def update(self):
63         # Keep looping indefinitely until the thread is stopped
64         while True:
65             # If the camera is stopped, stop the thread
66             if self.stopped:
67                 # Close camera resources
68                 self.stream.release()
69                 return
70
71             # Otherwise, grab the next frame from the stream
72             (self.grabbed, self.frame) = self.stream.read()
73
74     def read(self):
75         # Return the most recent frame
76         return self.frame
77
78     def stop(self):
79         # Indicate that the camera and thread should be stopped
80         self.stopped = True
81
```

FIGURE 3.33 : Deploiement du modèle sur la Raspberry Pi

```

82
83     def synthesize_audio(text, filename, distance):
84         #text_to_speak = ' '.join(text)
85         distance = round(distance, 1)
86         text = 'un' + text + ' ' + ' ' + str(distance) + ' centimetre'
87         tts = gTTS(text=text, lang='fr')
88
89         # Enregistrer le texte parlé au format MP3
90         bytes_io = io.BytesIO()
91         tts.write_to_fp(bytes_io)
92
93         # Revenir au début du fichier
94         bytes_io.seek(0)
95
96         # Charger le fichier MP3 et le convertir en format WAV
97         audio_mp3 = AudioSegment.from_file(bytes_io, format="mp3")
98         audio_wav = audio_mp3.set_frame_rate(24000).set_channels(1)
99
100        # Sauvegarder le fichier WAV
101        audio_wav.export(filename, format="wav")
102
103    def play_audio(response_audio_file):
104        # Charger et jouer le fichier audio avec pydub
105        audio = AudioSegment.from_wav(response_audio_file)
106        play(audio)
107
108
109
110    measured_distance=180
111    reference_real_width=50
112    reference_width_image=852
113
114    # focal length finder function
115    focal_length = (reference_width_image* measured_distance)/ reference_real_width
116
117    #Calcule de la disatance entre l'objet et la camera
118    def estimate_distance(object_width_image, reference_width_image, reference_real_width, focal_length):
119        return (reference_real_width * focal_length) / (2 * object_width_image)
120
121

```

FIGURE 3.34 : Deploiement du modèle sur la Raspberry Pi

```

131
132    # Define and parse input arguments
133    parser = argparse.ArgumentParser()
134    parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
135                        required=True)
136    parser.add_argument('--graph', help='Name of the .tflite file, if different than detect.tflite',
137                        default='detect.tflite')
138    parser.add_argument('--labels', help='Name of the labelmap file, if different than labelmap.txt',
139                        default='labelmap.txt')
140    parser.add_argument('--threshold', help='Minimum confidence threshold for displaying detected objects',
141                        default=0.5)
142    parser.add_argument('--resolution', help='Desired webcam resolution in WxH. If the webcam does not support the resolution entered, errors may occur.',
143                        default='1280x720')
144    parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up detection',
145                        action='store_true')
146
147    args = parser.parse_args()
148
149    MODEL_NAME = args.modeldir
150    GRAPH_NAME = args.graph
151    LABELMAP_NAME = args.labels
152    min_conf_threshold = float(args.threshold)
153    resW, resH = args.resolution.split('x')
154    imW, imH = int(resW), int(resH)
155    use_TPU = args.edgetpu
156
157    # Import TensorFlow libraries
158    # If tflite_runtime is installed, import interpreter from tflite_runtime, else import from regular tensorflow
159    # If using Coral Edge TPU, import the load_delegate library
160    pkg = importlib.util.find_spec('tflite_runtime')
161    if pkg:
162        from tflite_runtime.interpreter import Interpreter
163        if use_TPU:
164            from tflite_runtime.interpreter import load_delegate
165        else:
166            from tensorflow.lite.python.interpreter import Interpreter
167            if use_TPU:
168                from tensorflow.lite.python.interpreter import load_delegate
169
170    # If using Edge TPU, assign filename for Edge TPU model
171    if use_TPU:
172        # If user has specified the name of the .tflite file, use that name, otherwise use default 'edgetpu.tflite'
173        if (GRAPH_NAME == 'detect.tflite'):
174            GRAPH_NAME = 'edgetpu.tflite'
175
176    # Get path to current working directory
177    CWD_PATH = os.getcwd()

```

FIGURE 3.35 : Deploiement du modèle sur la Raspberry Pi

```

170
171 # Path to .tflite file, which contains the model that is used for object detection
172 PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)
173
174 # Path to label map file
175 PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)
176
177
178 # Load the label map
179 with open(PATH_TO_LABELS, 'r') as f:
180     labels = [line.strip() for line in f.readlines()]
181
182 with open('/home/labioia/tflite1/Sample_TFLite_model/labelmap_Fr.txt', 'r') as f:
183     labels_fr = [line.strip() for line in f.readlines()]
184
185
186 # Have to do a weird fix for label map if using the COCO "starter model" from
187 # https://www.tensorflow.org/lite/models/object_detection/overview
188 # First label is '???', which has to be removed.
189 if labels[0] == '???:':
190     del(labels[0])
191
192 if labels_fr[0] == '???:':
193     del(labels_fr[0])
194
195 # Load the Tensorflow Lite model.
196 # If using Edge TPU, use special load_delegate argument
197 if use_TPU:
198     interpreter = Interpreter(model_path=PATH_TO_CKPT,
199                               experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
200     print(PATH_TO_CKPT)
201 else:
202     interpreter = Interpreter(model_path=PATH_TO_CKPT)
203
204 interpreter.allocate_tensors()
205
206 # Get model details
207 input_details = interpreter.get_input_details()
208 output_details = interpreter.get_output_details()
209 height = input_details[0]['shape'][1]
210 width = input_details[0]['shape'][2]
211
212 floating_model = (input_details[0]['dtype'] == np.float32)
213
214 input_mean = 127.5
215 input_std = 127.5
216
217 # Check output layer name to determine if this model was created with TF2 or TF1,
218 # because outputs are ordered differently for TF2 and TF1 models
219 outname = output_details[0]['name']
220

```

FIGURE 3.36 : Deploiement du modèle sur la Raspberry Pi

```

221 if ('StatefulPartitionedCall' in outname): # This is a TF2 model
222 | boxes_idx, classes_idx, scores_idx = 1, 3, 0
223 else: # This is a TF1 model
224 | boxes_idx, classes_idx, scores_idx = 0, 1, 2
225
226 # Initialize frame rate calculation
227 frame_rate_calc = 1
228 freq = cv2.getTickFrequency()
229
230 # Initialize video stream
231 videotream = VideoStream(resolution=(imW,imH),framerate=30).start()
232 time.sleep(1)
233
234 #for frame1 in camera.capture_continuous(rawCapture, format="bgr",use_video_port=True):
235 while True:
236
237     # Start timer (for calculating frame rate)
238     t1 = cv2.getTickCount()
239
240     # Grab frame from video stream
241     frame1 = videotream.read()
242     frame1 = cv2.rotate(frame1, cv2.ROTATE_180)
243
244     # Acquire frame and resize to expected shape [1xHxWx3]
245     frame = frame1.copy()
246     frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
247     frame_resized = cv2.resize(frame_rgb, (width, height))
248     input_data = np.expand_dims(frame_resized, axis=0)
249
250     # Normalize pixel values if using a floating model (i.e. if model is non-quantized)
251     if floating_model:
252         input_data = (np.float32(input_data) - input_mean) / input_std
253
254     # Perform the actual detection by running the model with the image as input
255     interpreter.set_tensor(input_details[0]['index'],input_data)
256     interpreter.invoke()
257
258     # Retrieve detection results
259     boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0] # Bounding box coordinates of detected objects
260     classes = interpreter.get_tensor(output_details[classes_idx]['index'])[0] # Class index of detected objects
261     scores = interpreter.get_tensor(output_details[scores_idx]['index'])[0] # Confidence of detected objects
262
263

```

FIGURE 3.37 : Deploiement du modèle sur la Raspberry Pi

```

import socket

socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket.bind(('0.0.0.0', 5000))

#tentative de connexion
socket.listen(5)
print('En attente de connexion...')

while True:
    conn, addr = socket.accept()
    print('Client connecté :', addr)

    while True:
        # Reception de données du client
        data = conn.recv(1024)

        if not data:
            print('connexion interrompue')
            break
        print(data.decode())

```

FIGURE 3.38 : Définition du socket pour écoute des connexions entrantes sur le Raspberry Pi

```

#connexion de l'esp au wifi
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
if not wlan.isconnected():
    print('connecting to network...')
    wlan.connect('LABO_IA', 'Lab_Ia2o23 ')
    while not wlan.isconnected():
        pass
#afficher l'adresse ip de l'esp sur le reseau
print('network config:', wlan.ifconfig())

# Connexion au raspberry
host = '192.168.0.237'
port = 5000
socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket.connect((host, port))
print('Connecté au raspberry')

message = str(225.856) + ' ' + str(888.655)
socket.send(message.encode())
print('Message envoyé :', message)

```

FIGURE 3.39 : Connexion de l'ESP32 à la Raspberry Pi

Annexe 4 : Méthodologie du module TTS

```

✓ def synthesize_audio(text, filename, distance):
    #text_to_speak = ' '.join(text)
    distance = round(distance, 1)
    text = 'un' + text + ' ' + 'à' + ' ' + str(distance) + 'centimetre'
    tts = gTTS(text=text, lang='fr')

    # Enregistrer le texte parlé au format MP3
    bytes_io = io.BytesIO()
    tts.write_to_fp(bytes_io)

    # Revenir au début du fichier
    bytes_io.seek(0)

    # Charger le fichier MP3 et le convertir en format WAV
    audio_mp3 = AudioSegment.from_file(bytes_io, format="mp3")
    audio_wav = audio_mp3.set_frame_rate(24000).set_channels(1)

    # Sauvegarder le fichier WAV
    audio_wav.export(filename, format="wav")

✓ def play_audio(response_audio_file):
    # Charger et jouer le fichier audio avec pydub
    audio = AudioSegment.from_wav(response_audio_file)
    play(audio)

```

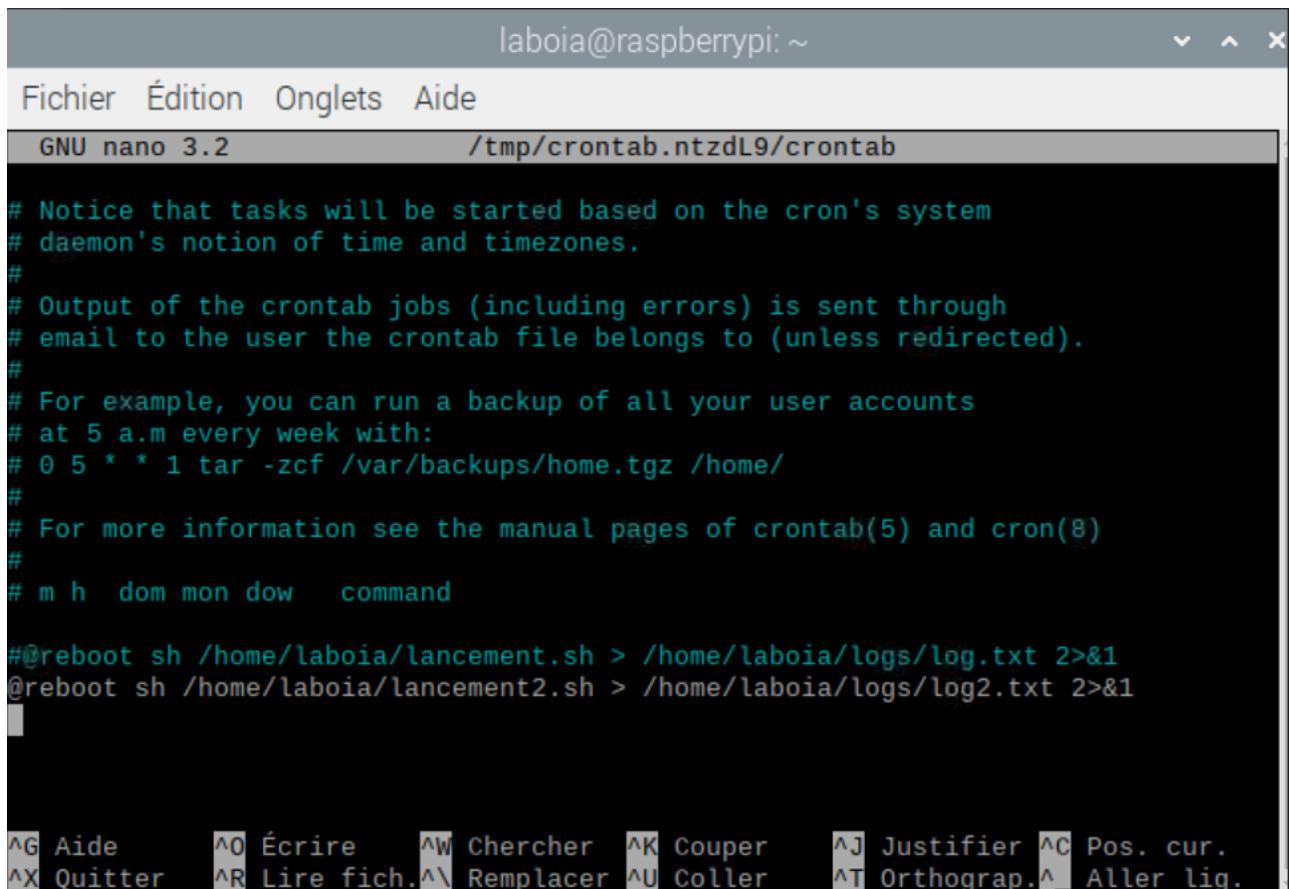
FIGURE 3.40 : Implémentation du module TTS

Annexe 5 : Script shell de lancement

```
1 #!/bin/bash
2
3 sleep 60
4
5 cd /home/labolia/tflite1
6 . tflite1_env/bin/activate
7 python3 TFLite_detection_webcam1.py --modeldir=Sample_TFLite_model
8
```

FIGURE 3.41 : Script shell de récupération des flux vidéo et lancement de la détection dès le démarrage du Raspberry Pi

Annexe 6 : Ajout du fichier dans crontab



The screenshot shows a terminal window titled "labolia@raspberrypi: ~". The menu bar includes "Fichier", "Édition", "Onglets", and "Aide". The title bar also displays the path "/tmp/crontab.ntzdL9/crontab". The main area of the terminal shows a crontab file with the following content:

```
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
#@reboot sh /home/labolia/lancement.sh > /home/labolia/logs/log.txt 2>&1
@reboot sh /home/labolia/lancement2.sh > /home/labolia/logs/log2.txt 2>&1
```

At the bottom of the terminal window, there is a status bar with keyboard shortcuts:

^G Aide **^O Écrire** **^W Chercher** **^K Couper** **^J Justifier** **^C Pos. cur.**
^X Quitter **^R Lire fich.** **^V Remplacer** **^U Coller** **^T Orthograp.** **^_ Aller lig.**

FIGURE 3.42 : Ajout du fichier dans crontab

Annexe 7 : Fine-tuning total

Dans le cadre du Fine-tuning total, la dernière couche fully-connected (ou la couche de classification) du modèle pré-entraîné est remplacée par une nouvelle couche adaptée au nouveau problème que nous voulons résoudre. Cette nouvelle couche de classification peut être un classifieur tel qu'un SVM (Machine à Vecteurs de Support), une régression logistique, ou même une nouvelle couche fully-connected spécifiquement conçue pour le nouveau problème.

Après avoir remplacé la dernière couche, nous initialisons généralement les poids de cette

couche de manière aléatoire. Cela signifie que les poids sont initialement réglés de manière aléatoire et ne contiennent aucune information spécifique sur le nouveau problème.

Ensuite, nous entraînons tout le réseau, y compris les nouvelles couches ajoutées, sur le nouveau jeu de données. L'objectif est de faire en sorte que le réseau s'adapte aux caractéristiques spécifiques du nouveau jeu de données tout en préservant les connaissances générales apprises lors de l'entraînement initial sur le grand ensemble de données.

Cette stratégie est particulièrement efficace lorsque le nouveau jeu de données est suffisamment grand pour permettre l'entraînement de toutes les couches du réseau sans risque de surapprentissage (overfitting). En effet, en entraînant tout le réseau sur les nouvelles données, nous pouvons tirer parti des connaissances préalables acquises par le modèle pré-entraîné tout en adaptant les caractéristiques spécifiques du nouveau problème. Cela permet souvent d'obtenir des performances élevées avec moins de temps et de ressources de calcul par rapport à l'entraînement complet d'un réseau à partir de zéro.

Table des matières

Dédicace	ii
Remerciements	iii
Résumé	iv
	iv
Abstract	v
	v
Table des figures	vi
Liste des tableaux	viii
Liste des acronymes	ix
Introduction	1
1 Revue de littérature	3
Introduction	3
1.1 Vue d'ensemble sur la détection d'objets	3
1.1.1 Apprentissage profond	3
1.1.1.1 Généralités sur les réseaux de neurones artificiels	3
1.1.1.2 Apprentissage profond : Deep learning	4
1.1.1.3 Classification d'images par les réseaux de neurones convolutifs	4
1.1.1.4 Apprentissage par transfert avec les réseaux de neurones convolutifs	5
1.1.1.5 Classification des algorithmes de détection d'objets	6
1.1.2 Architecture CNN MobileNets	6
1.1.2.1 MobileNet V1 (depth-wise separable convolution)	6
1.1.2.2 MobileNet V2 (inverted residual blocks)	7
1.2 Vue d'ensemble sur la synthèse vocale	8
1.2.1 Traitement du Langage Naturel	9
1.2.1.1 Composant d'analyse du contexte	10
1.2.1.2 Phonétisation automatique	10
1.2.1.3 Génération de la prosodie	11
1.2.2 Traitement du Signal Numérique	12
1.2.3 Google Text-to-Speech	12
1.3 État de l'art de l'autoguidage et Étude critique de l'existant	13

1.3.1	Définition du déplacement	13
1.3.2	Importance du système d'autoguidage	13
1.3.3	Les maladies de la déficience visuelle : Causes, Symptômes et Aides	13
1.3.4	Revue de littérature sur les systèmes d'autoguidage	14
1.3.5	Présentation de quelques applications intelligentes pour l'aide à la mobilité des personnes malvoyantes	18
1.3.6	Critique de l'existant (limitations des technologies actuelles)	19
	Conclusion	20
2	Matériel et méthodes	21
	Introduction	21
2.1	Matériels	21
2.1.1	Tensorflow	22
2.1.2	TensorFlow Lite	23
2.1.3	OpenCV	23
2.1.4	Langage de programmation Python	24
2.1.5	RealVNC	24
2.1.6	Thonny	24
2.1.7	LabelImg	24
2.2	Méthodes	25
2.2.1	Schéma du système proposé	25
2.2.2	Organigramme de fonctionnement du système	26
2.2.3	Méthodologie de travail	27
2.2.3.1	Collecte de données	27
2.2.3.2	Préparer ces données	27
2.2.3.3	Sélection de modèles	28
2.2.3.4	Fine-turning	33
2.2.3.5	Mesures de performances des modèles pour les problèmes de classification multi-classes	34
2.2.3.6	Intégration de la synthèse vocale (TTS)	35
2.2.3.7	Déploiement du modèle sur un Raspberry Pi	35
	Conclusion	35
3	Résultats et discussions	37
	Introduction	37
3.1	Le dataset utilisé	37
3.2	Résultats des entraînements faits sur le jeu de données avec le modèle SSD-Mobilenet-v2-fpnlite-320	40
3.3	Résultats des entraînements faits sur le jeu de données avec le modèle SSD-Mobilenet-v2	42
3.4	Résultats des entraînements faits sur le jeu de données avec le modèle EfficientDet-D0	43
3.5	Test sur le Raspberry Pi avec le modèle SSD-Mobilenet-v2-fpnlite-320	44
3.6	Prototype expérimentale du système d'autoguidage	47
	Conclusion	47

Conclusion	48
Bibliographie	49
Annexe	53
Table des matières	72