UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

MASTER MIASHS « HANDI »
Parcours : Technologie et Handicap

Rapport Projet AAA

Poubelle Connectée

Sofiane Lhiyat

Enseignant référent : Salvatore Anzalone - Dominique Archambault

Saint-Denis, Janvier 2025

Remerciements :

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce projet.

Tout d'abord, je remercie mes enseignants encadrants, M. Salvatore Anzalone et M. Dominique Archambault, pour leur disponibilité, leurs précieux conseils, le prêt du matériel et leur expertise qui ont été des atouts essentiels tout au long de ce travail.

Je souhaite également remercier l'ensemble des enseignants de ma formation, mes camarades ainsi que mes anciens professeurs, qui m'ont transmis des connaissances fondamentales pour ce projet.

Une reconnaissance particulière est adressée au laboratoire LUTIN pour la disponibilité du matériel ainsi que les nombreuses heures passées au laboratoire quand l’université n’était pas disponible. Votre environnement stimulant et vos ressources ont joué un rôle clé dans l'avancement de ce projet.

Merci à toutes et à tous.

Lexique :

API (Application Programming Interface) : Interface permettant la communication entre différents composants logiciels

API REST : Style d'architecture pour les services web permettant des échanges standardisés

Batch Normalization : Technique d'apprentissage profond qui normalise les données pendant l'entraînement

BGR : Format de couleur Blue-Green-Red utilisé par OpenCV

Callback : Fonction qui est exécutée en réponse à un événement

CNN (Convolutional Neural Network) : Type de réseau de neurones spécialisé dans le traitement d'images

Dataset : Ensemble de données utilisé pour l'entraînement d'un modèle

Dropout : Technique de régularisation pour prévenir le surapprentissage

ESP32-S2 : Microcontrôleur avec WiFi intégré

Endpoint : Point d'accès d'une API

Flask : Framework web léger en Python

F1-score : Métrique combinant précision et rappel

GPIO : Broche d'entrée/sortie à usage général sur un microcontrôleur

GPU : Processeur graphique

HTTP : Protocole de communication web

HTTPS : Version sécurisée du protocole HTTP

ImageNet : Large base de données d'images pour l'entraînement de modèles

Inférence : Processus de prédiction utilisant un modèle entraîné

LAN : Réseau local (Local Area Network)

Learning Rate : Taux d'apprentissage dans l'entraînement d'un modèle

MVVM : Pattern d'architecture Model-View-ViewModel

MicroPython : Version de Python pour microcontrôleurs

OpenCV : Bibliothèque de vision par ordinateur

Overfitting : Sur-Apprentissage d'un modèle

PWM : Modulation de largeur d'impulsion

PyTorch : Framework d'apprentissage profond

ResNet 18 : Architecture de réseau de neurones à 18 couches

RGB : Format de couleur Red-Green-Blue

Retrofit : Bibliothèque HTTP pour Android

SDK : Kit de développement logiciel

SGD : Descente de gradient stochastique

Servomoteur : Moteur avec contrôle précis de position

Transfer Learning : Technique de réutilisation d'un modèle pré-entraîné

Thread : Fil d'exécution dans un programme

Weight Decay : Technique de régularisation

WiFi : Technologie de réseau sans fil

1. Introduction **6**
1.1 Contexte et objectifs du projet 6
1.2 Problématique du tri des déchets 6
1.3 Solution proposée 7
1.4 Innovation et pertinence 7
2. État de l'art **8**
2.1 Projets similaire et recherches 8
2.2 Technologies clés 8
3. Architecture du système **9**
3.1 Vue d'ensemble du système 9
3.2 Flux de données 10
3.3 Communication entre les composants 11
3.4 Diagramme d'architecture 11
4. Composants matériels **13**
4.1 ESP32 et servomoteur 13
4.2 Caméra 13
4.3 Configuration matérielle 13
4.4 Schéma de connexion 14
5. Développement logiciel **15**
5.1 Backend (Python / Flask) 15
5.1.1 Architecture serveur 15
5.1.2 API REST 15
5.1.3 Gestion des requêtes 16
5.1.4 Traitement des images 16
5.2 Microcontrôleur (MicroPython) 17
5.2.1 Configuration Wifi 17
5.2.2 Contrôle du servomoteur 17
5.2.3 Serveur web embarqué 18
5.2.4 Gestion des connexions 18
5.3 Application Android 18
5.3.1 Architecture de l'application 18
5.3.2 Communication avec le serveur 19
5.3.3 Gestion des retours utilisateur 19
5.4 Architecture Client-Serveur 20
5.4.1 Communication Client-Serveur 20
5.4.2 Gestion des sessions 20
6. Algorithme d'apprentissage **21**
6.1 Modèle de classification 21
6.1.1 Architecture ResNet 18 21
6.1.2 Dataset et classes 22
6.1.3 Processus d'entraînement 23
6.1.4 Métriques de performances 24
6.2 Système d'apprentissage continu 25

6.2.1 Mécanisme de feedback 25
6.2.2 Mise à jour du modèle 26
6.2.3 Gestion des erreurs 26
7. Interface utilisateur 27
7.1 Design de l'interface 27
7.2 Flux utilisateur 27
7.3 Retours visuels 28
8. Tests et validations 30
8.1 Tests unitaires 30
8.2 Tests d'intégration 30
8.3 Validation du modèle 31
8.4 Performance du système 31
9. Conclusion et perspectives 33
9.1 Résultats obtenus 33
9.2 Limitations actuelles 33
9.3 Améliorations possibles 33
9.4 Perspectives d'évolutions 34
10. Sources 35

1. Introduction

1.1 Contexte et objectifs du projet

Dans un monde où le tri des déchets devient une préoccupation majeure, de nombreuses personnes rencontrent des difficultés particulières pour effectuer ce geste pourtant essentiel. Parmi elles, les personnes malvoyantes, non voyantes, ou celles ayant des difficultés sensorielles tactiles se trouvent confrontées à un défi quotidien : comment identifier correctement les matériaux à trier lorsque la vue ou le toucher ne sont pas fiables ? Ce projet de poubelle connectée vise à répondre à ce besoin spécifique en proposant une solution technologique accessible et autonome.

1.2 Problématique du tri des déchets

Le tri des déchets représente un enjeu majeur pour notre société, avec des statistiques alarmantes :
En France, environ 40% des déchets ménagers ne sont pas triés correctement. Sur les 568 kg de déchets produits par an et par habitant en France :

- Seulement 50% sont effectivement recyclés
- 30% finissent en incinération
- 20% sont enfouis dans des décharges

Pour les personnes en situation de handicap visuel ou ayant des troubles sensoriels, ces chiffres sont encore plus préoccupants :

- 70% des personnes malvoyantes ou non-voyantes déclarent avoir des difficultés à trier leurs déchets
- Le taux d'erreur de tri pour ces personnes peut atteindre 60%
- Plus de 80% d'entre elles doivent régulièrement demander de l'aide pour cette tâche quotidienne

Cette situation présente plusieurs défis spécifiques :

- La difficulté d'identification des matériaux par le toucher (notamment entre le plastique et l'aluminium)
- L'impossibilité de lire les consignes de tri sur les emballages
- Le risque d'erreurs de tri qui peut conduire à une contamination des flux de recyclage
- Le stress et la perte d'autonomie liés à la nécessité de demander de l'aide pour une tâche quotidienne
- L'absence de retour immédiat sur la qualité du tri effectué

Impact environnemental et économique :

- Les erreurs de tri coûtent environ 40 millions d'euros par an aux collectivités
- Un mauvais tri peut contaminer jusqu'à 10 kg de déchets recyclables
- Le taux de refus moyen en centre de tri est de 17%, principalement dû aux erreurs de tri

Cette problématique met en évidence le besoin urgent de solutions adaptées, non seulement pour améliorer l'efficacité globale du tri des déchets, mais aussi pour rendre ce geste écologique accessible à tous, indépendamment des capacités physiques ou sensorielles de chacun.

1.3 Solution proposée

Ma solution consiste en une poubelle intelligente qui combine plusieurs technologies pour assister l'utilisateur :
Un système de vision par ordinateur qui analyse automatiquement les déchets
Un mécanisme de tri automatique qui oriente le déchet vers le bon compartiment.
Une application mobile accessible qui permet de :

- Confirmer ou corriger les détections du système
- Recevoir des retours audio sur le tri effectué

1.4 Innovation et pertinence

L'innovation de ce projet réside dans plusieurs aspects :
Inclusivité par design : Contrairement aux solutions existantes qui se concentrent uniquement sur l'efficacité du tri, mon système est pensé dès sa conception pour être accessible aux personnes en situation de handicap.
Apprentissage continu : Le système s'améliore avec chaque utilisation grâce aux retours des utilisateurs, permettant une adaptation progressive aux spécificités de chaque foyer.
Autonomisation : En rendant le tri des déchets accessible à tous, nous contribuons à l'autonomie des personnes en situation de handicap dans leur vie quotidienne.
Impact social et environnemental : Cette solution permet non seulement d'améliorer la qualité du tri des déchets, mais aussi de rendre le geste écologique accessible à une population souvent oubliée dans la conception des solutions environnementales.

La pertinence de ce projet s'inscrit dans une double démarche :

- Sociale : en proposant une solution concrète pour l'autonomie des personnes en situation de handicap
- Environnementale : en améliorant la qualité du tri des déchets et en rendant le recyclage plus accessible

Cela répond ainsi à un besoin réel tout en s'inscrivant dans les objectifs de développement durable et d'inclusion sociale.

2. État de l’art

Le tri des déchets automatisé et la reconnaissance des matériaux par vision par ordinateur sont des domaines en constante évolution. Plusieurs projets et recherches ont été menés dans ces directions.

2.1 Projets similaire et recherches

Projets académiques :

- TrashNet (2016) : Un des premiers datasets publics pour la classification des déchets, créé par l'Université de Stanford. Ce projet a établi une base de référence pour la reconnaissance des déchets avec plus de 2500 images dans 6 catégories.
- SmartBin (2019) : Un projet de l'Université de Delhi utilisant une combinaison de capteurs et de vision par ordinateur pour le tri automatique. Leur système atteint une précision de 87% mais nécessite des conditions d'éclairage contrôlées.

Solutions commerciales :

- BinBrain (2022) : Une poubelle intelligente commerciale utilisant l'IA pour le tri, mais sans fonctionnalités d'accessibilité pour les personnes malvoyantes.
- EvoEco (2023) : Un système de tri automatique industriel utilisant une combinaison de capteurs optiques et de machine learning.

Recherche sur l’accessibilité :

- VisionAid (2020) : Un projet de l'INSA Lyon focalisé sur l'assistance aux personnes malvoyantes pour les tâches quotidiennes.
- HandiSort (2022) : Une étude de l'Université de Paris sur les besoins spécifiques des personnes en situation de handicap concernant le tri des déchets.

2.2 Technologies clés

Vision par ordinateur :

- Évolution des réseaux de neurones convolutifs (CNN)
- Émergence des architectures efficientes (MobileNet, EfficientNet)
- Développement du transfer learning pour l’optimisation des performances

Intelligence artificielle embarquée :

- Solutions edge computing pour le traitement local
- Optimisation des modèles pour les systèmes embarqués
- Techniques d’apprentissage continu

3. Architecture du système

3.1 Vue d'ensemble du système

Le système est composé d'une plateforme rotative motorisée conçue selon les spécifications suivantes :

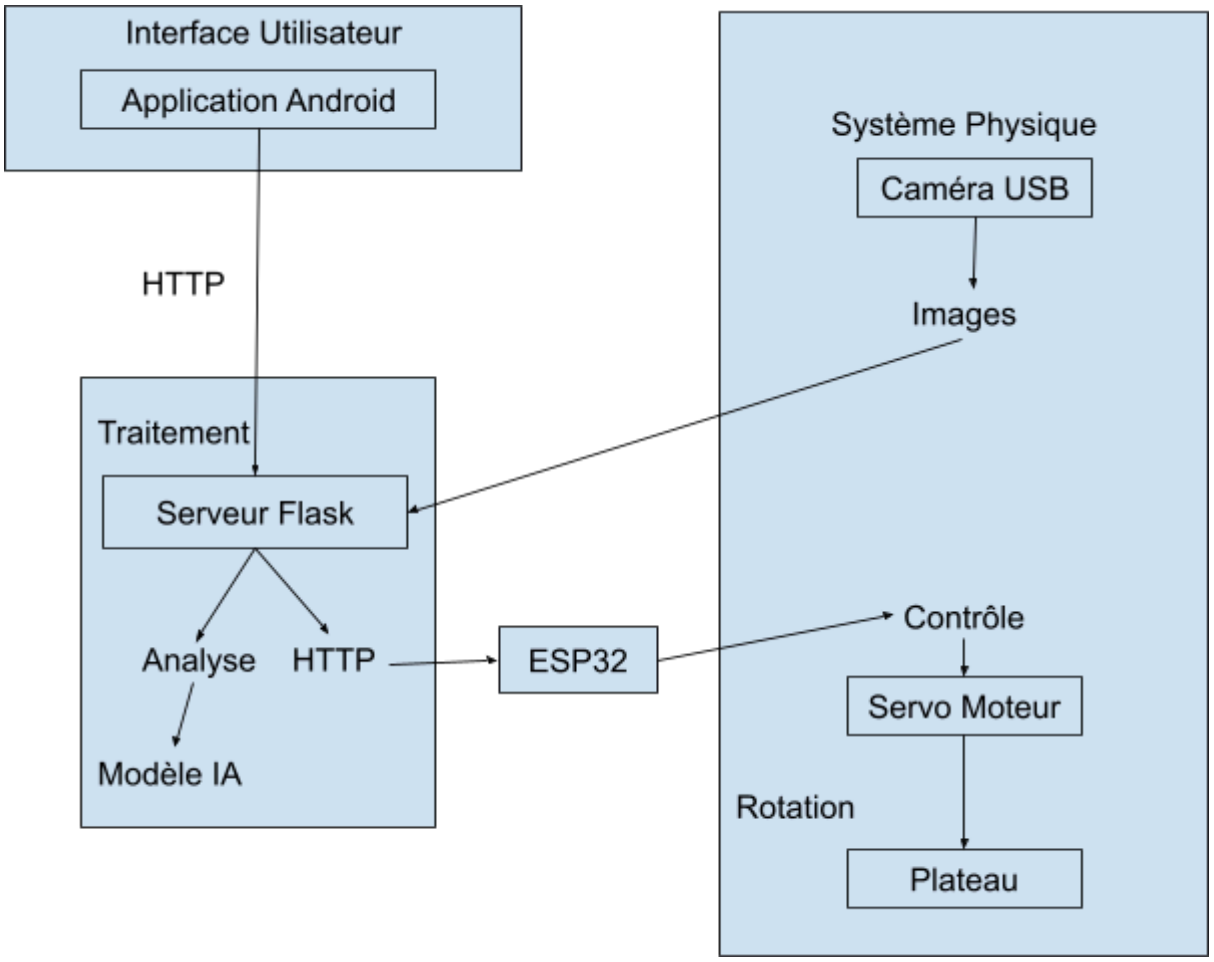
- Un plateau principal de 220x200mm servant de support pour les déchets à analyser
- Une hauteur totale de 170 mm permettant une bonne accessibilité
- Un mécanisme de rotation basé sur un servomoteur offrant un angle de ±90°

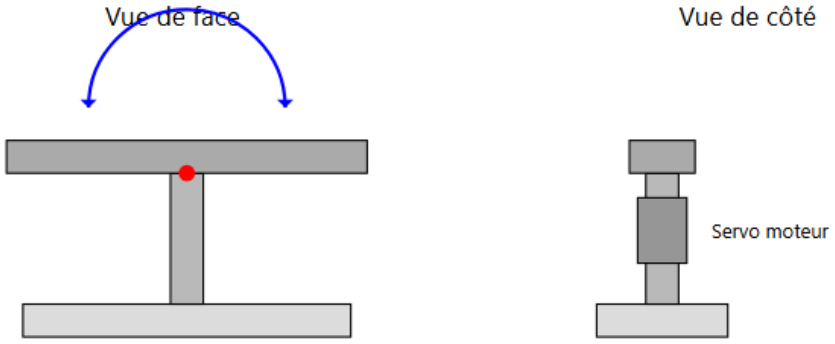
La poubelle est divisée en deux compartiments distincts, suivant les standards français de tri sélectif :

- Un compartiment vert destiné exclusivement au verre
- Un compartiment jaune pour les autres matériaux recyclables (plastique, métal, carton, papier)

Ce système mécanique est complété par :

- Une caméra positionnée au-dessus du plateau pour la capture d'images
- Un ESP32 pour le contrôle du servomoteur
- Un serveur central pour le traitement des données
- Une application Android pour l'interface utilisateur





3.2 Flux de données

Phase de capture d'image
Le processus débute lorsque l'utilisateur dépose un déchet sur le plateau de la poubelle intelligente. À cet instant, la caméra USB positionnée au-dessus du plateau entre en action et capture une image haute définition de l'objet. Cette image est immédiatement transmise au serveur Flask via la connexion USB directe, assurant un transfert rapide et fiable des données visuelles nécessaires à l'analyse.

Phase d'analyse et prédiction
Une fois l'image reçue, le serveur Flask initie la phase de traitement. L'image subit d'abord un prétraitement pour optimiser sa qualité et son format selon les exigences du modèle d'intelligence artificielle. Le modèle ResNet18, spécialement entraîné pour la reconnaissance des différents types de déchets, analyse alors l'image et effectue sa classification. Le résultat de cette analyse est automatiquement transmis à l'application Android de l'utilisateur, présentant la catégorie de déchet identifiée.

Phase de validation et action
L'étape finale du processus requiert l'intervention de l'utilisateur qui, via l'application Android, confirme la précision de la prédiction ou la corrige si nécessaire. Suite à cette validation, le serveur transmet instantanément la commande appropriée à l'ESP32. Le microcontrôleur traduit cette commande en action physique : le servomoteur s'active et fait pivoter le plateau avec précision vers le compartiment correspondant au type de déchet identifié, assurant ainsi un tri correct et efficace.

3.3 Communication entre les composants

Le système utilise plusieurs protocoles de communication :

Communication Application - Serveur
Le système repose sur une architecture client-serveur utilisant le protocole HTTP/REST pour assurer la communication entre l'application Android et le serveur Flask. L'application peut interagir avec deux endpoints principaux : le premier, /predict, permet d'obtenir en temps réel les prédictions du modèle d'intelligence artificielle concernant le type de déchet analysé. Le second endpoint, /feedback, permet à l'utilisateur d'envoyer ses retours sur la précision de la prédiction, contribuant ainsi à l'amélioration continue du système. Ces interactions sont conçues pour être rapides et fiables, avec des temps de réponse optimisés pour une expérience utilisateur fluide.

Communication Serveur - ESP32
L'interaction entre le serveur Flask et le microcontrôleur ESP32 s'effectue également via le protocole HTTP, permettant un contrôle précis du servomoteur. Trois commandes principales sont implémentées pour gérer les mouvements du plateau : /servo/left pour une rotation vers la gauche, /servo/right pour une rotation vers la droite, et /servo/center pour ramener le plateau en position centrale. Cette communication est essentielle pour orienter correctement le déchet vers le compartiment approprié une fois la classification effectuée. Le serveur envoie ces commandes en fonction du résultat de la classification et de la validation de l'utilisateur.

Communication Caméra - Serveur
La liaison entre la caméra et le serveur est assurée par une connexion USB directe, garantissant un transfert rapide et stable des images. Le système utilise la bibliothèque OpenCV pour la capture et le traitement des images, permettant une acquisition en temps réel des photos des déchets à analyser. Cette connexion directe minimise la latence et assure une qualité d'image optimale, deux facteurs cruciaux pour la précision du système de classification. Le serveur peut ainsi recevoir instantanément les images nécessaires à l'analyse, les traiter, et les transmettre au modèle d'intelligence artificielle pour la classification.

3.4 Diagramme d’architecture

L'architecture du système a été conçue en trois couches distinctes, chacune ayant un rôle spécifique et essentiel dans le fonctionnement global du système de tri intelligent.

Couche Physique
La couche physique constitue le socle matériel du système. Elle s'articule autour d'un plateau rotatif de 220x200mm, actionné par un servomoteur de précision capable d'effectuer des rotations de ±90°. Cette partie mécanique est supervisée par un microcontrôleur ESP32 qui assure le contrôle précis des mouvements. Une caméra USB est stratégiquement positionnée au-dessus du plateau pour capturer des images en haute définition des déchets à analyser. L'ensemble est conçu pour être robuste et facilement accessible, avec une hauteur totale de 170 mm.

Couche Traitement
Au cœur du système, la couche de traitement constitue le cerveau de l'opération. Elle s'appuie sur un serveur Flask qui orchestre l'ensemble des opérations. Le modèle d'intelligence artificielle ResNet18, spécialement adapté pour la reconnaissance des déchets, y est intégré. Cette couche implémente également un système sophistiqué de feedback qui permet l'apprentissage continu du modèle. Les images capturées sont traitées en temps réel, et les résultats sont rapidement transmis à l'interface utilisateur. La gestion des erreurs et la validation des données sont également prises en charge à ce niveau.

Couche Interface
La couche interface représente le point de contact avec l'utilisateur. Elle se matérialise principalement par une application Android intuitive et accessible. Spécialement conçue pour les personnes malvoyantes ou ayant des difficultés de reconnaissance tactile, elle intègre une interface vocale qui guide l'utilisateur tout au long du processus de tri. Des retours haptiques et sonores sont également implémentés pour confirmer les actions et les résultats. L'application permet non seulement de visualiser les prédictions du système, mais aussi de les corriger si nécessaire, contribuant ainsi à l'amélioration continue du modèle de reconnaissance.

Cette architecture tripartite offre plusieurs avantages majeurs :

- La modularité permet une maintenance simplifiée, chaque couche pouvant être mise à jour ou modifiée indépendamment des autres.
- L'évolutivité est facilitée par la séparation claire des responsabilités, permettant l'ajout de nouvelles fonctionnalités sans perturber l'existant.
- La robustesse est assurée par une gestion des erreurs à chaque niveau, garantissant un service fiable même en cas de défaillance partielle.
- L'adaptabilité du système permet son déploiement dans différents contextes d'utilisation, que ce soit en environnement domestique ou institutionnel.

Cette conception architecturale réfléchie répond ainsi aux exigences d'accessibilité et de fiabilité essentielles pour un système destiné à faciliter le tri des déchets pour tous les utilisateurs, indépendamment de leurs capacités physiques.

4. Composants matériels

4.1 ESP32 et servomoteur

L'ESP32-S2 d'Espressif constitue le cerveau du système de contrôle matériel. Ce microcontrôleur a été sélectionné pour ses caractéristiques techniques : un processeur cadencé à 240 MHz, 320 Ko de RAM et une connectivité WiFi intégrée. Ces spécifications permettent une gestion fluide des communications et du contrôle moteur.

Le servomoteur MG996R, avec son couple élevé de 9.4 kg · cm à 4.8V, assure la rotation précise du plateau. Sa plage de rotation de 180° est parfaitement adaptée au besoin de ±90°. Les caractéristiques de ce servomoteur sont particulièrement appropriées pour mon application :

- Tension de fonctionnement : 4.8V à 7.2V
- Vitesse de rotation : 0.17s/60° à 4.8V
- Roulements métalliques pour une durabilité accrue
- Précision de positionnement suffisante pour un tri précis
- Ampérage requis 1,5A

4.2 Caméra

Le système utilise une caméra USB standard capable de capturer des images en 1280 x 1080 pixels. Cette résolution offre un bon compromis entre la qualité d'image nécessaire pour la reconnaissance des déchets et les ressources de traitement requises. La caméra est positionnée à 30 cm au-dessus du plateau, offrant un champ de vision optimal pour capturer l'intégralité de la zone de détection de 220x200mm.

Caractéristiques principales de la configuration caméra :

- Connection : USB 2.0
- Résolution : HD 1080p
- Auto-focus pour une adaptation aux différentes tailles de déchets
- Balance des blancs automatique pour une reconnaissance fiable dans différentes conditions d'éclairage
-

4.3 Configuration matérielle

Le système est divisé en deux parties distinctes pour une meilleure stabilité :

Partie1 : Contrôle moteur (ESP32-S2 et servomoteur) :
Un point critique du système concerne l'alimentation. Initialement, une alimentation 5V/500 mA était prévue, mais les tests ont révélé une instabilité majeure : lorsque le servomoteur MG996R entre en action, ses pics de courant provoquent des chutes de tension qui perturbent le fonctionnement de l'ESP32-S2, entraînant des redémarrages intempestifs du système. Du coup on a deux alimentation dédié à chaque composant :

- ESP32-S2 et logique : Alimentation USB 5V/500 mA
- Servomoteur MG996R : Alimentation dédiée 5V/5A minimum avec protection contre les pics de courant

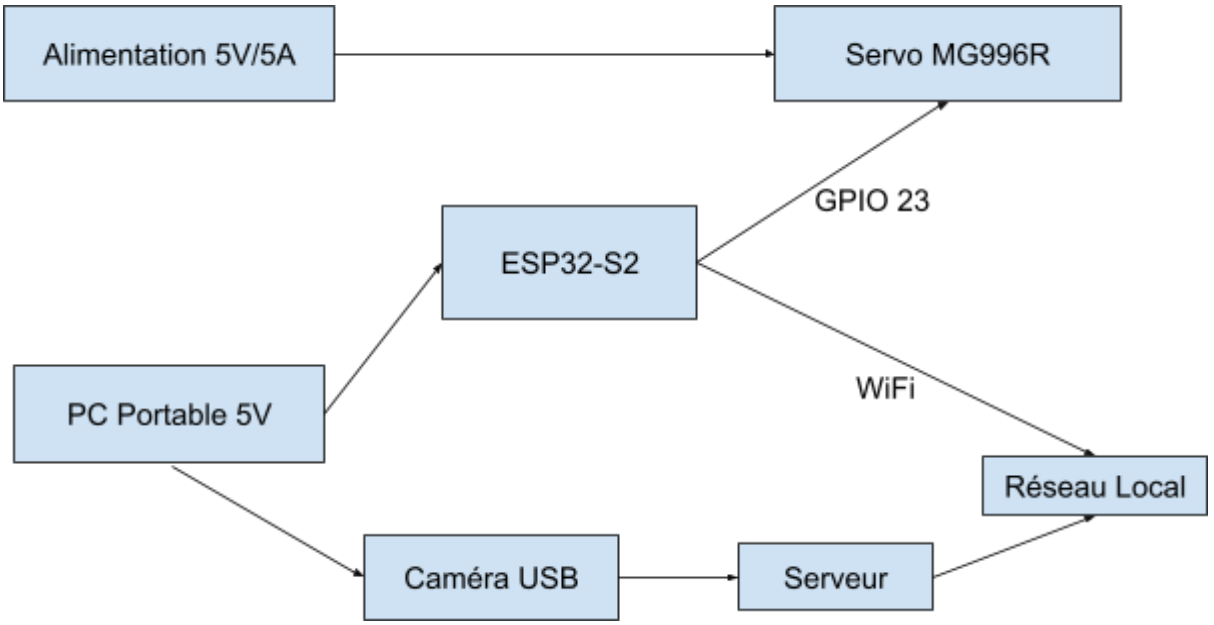
La puissance plus élevée est nécessaire pour gérer les pics de courant du servomoteur
Cette séparation protège la partie logique des perturbations électriques.
Cette configuration robuste garantit une stabilité accrue du système, aucune interférence entre le servomoteur et l'ESP32-S2, un fonctionnement fiable sur le long terme et une protection des composants contre les surcharges.
Ces modifications matérielles sont essentielles pour assurer un fonctionnement stable et durable du système.

Partie 2 : Acquisition d'images (PC portable et caméra) :
La caméra USB est directement connectée au PC et elle est alimentée via le port USB du PC (5V/500mA)

4.4 Schéma de connexion

Les connexions entre les composants sont réalisées comme suit :

- L'ESP32-S2 est alimenté en 5V via USB
- Le servomoteur MG996R est connecté directement à son alimentation dédiée 5V/5A
- Le signal de contrôle du servo est relié au GPIO 23 de l'ESP32-S2
- La caméra USB est connectée directement au PC portable via USB



L'ensemble est monté sur une plaque de prototypage avec des connecteurs sécurisés pour éviter tout débranchement accidentel. Les câbles sont regroupés et protégés pour assurer la durabilité de l'installation.

5. Développement logiciel

5.1 Backend (Python / Flask)

5.1.1 Architecture serveur

Le backend du système est construit autour d'un serveur Flask en Python, choisi pour sa légèreté et sa flexibilité. Le serveur est structuré selon une architecture modulaire qui sépare clairement les différentes responsabilités : gestion des requêtes HTTP, traitement des images, interface avec le modèle d'IA, et communication avec l'ESP32.

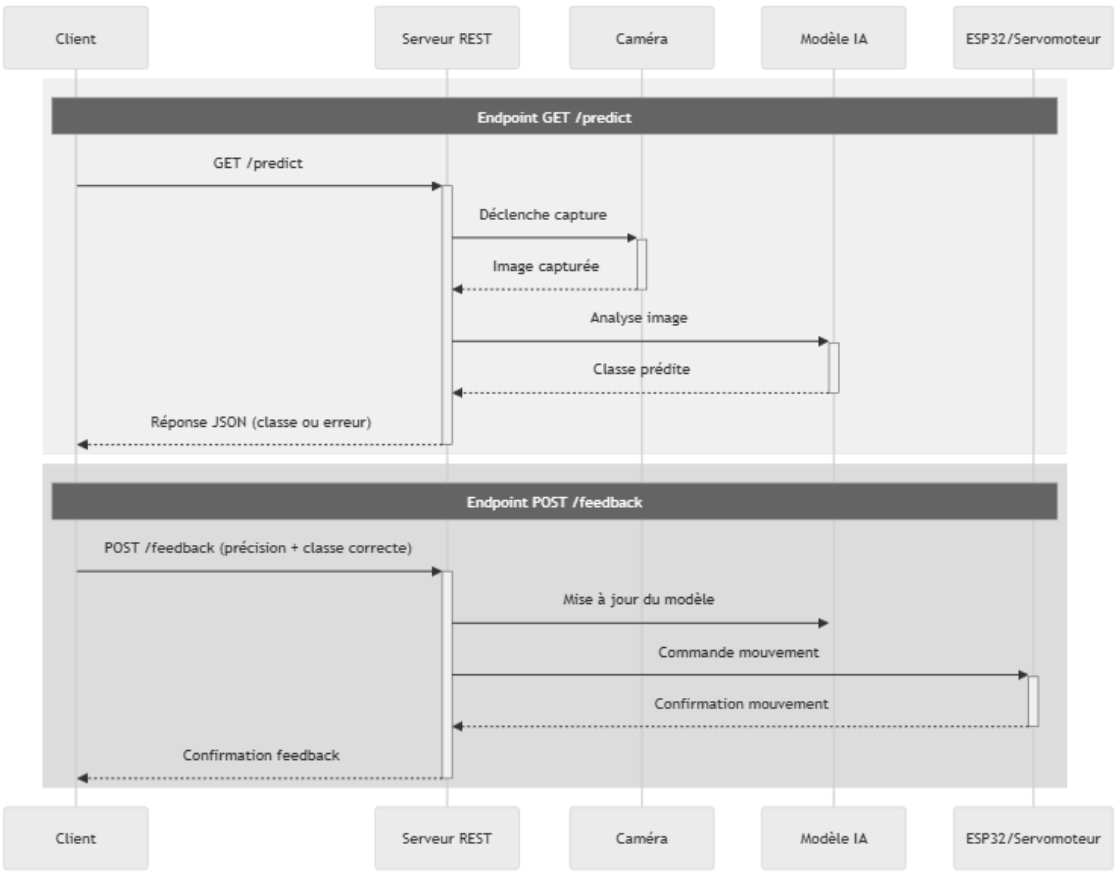
Le cœur du serveur est construit autour d'une instance Flask qui gère les routes HTTP et coordonne les différentes opérations. Un système de threading est implémenté pour gérer les opérations lourdes comme l'inférence du modèle d'IA et le traitement d'images, évitant ainsi le blocage du serveur lors des opérations intensives. Un verrou global (model_lock) est utilisé pour synchroniser l'accès au modèle d'IA, particulièrement important lors des phases d'apprentissage et de prédiction qui ne peuvent pas s'exécuter simultanément.

Le serveur maintient également un état global minimal, stockant temporairement la dernière image traitée et sa prédiction associée pour permettre la mise à jour du modèle en cas de correction par l'utilisateur. Cette architecture permet une gestion efficace de la mémoire tout en maintenant les informations nécessaires pour l'apprentissage continu.

5.1.2 API REST

L'API REST du serveur expose deux endpoints principaux qui forment l'interface de communication avec les clients. Le premier endpoint, dédié à la prédiction, est accessible via une requête GET. Lorsqu'il est appelé, il déclenche la capture d'une image, son analyse par le modèle d'IA, et retourne la classe prédite pour le déchet photographié. La réponse est formatée en JSON, incluant soit la classe prédite, soit un message d'erreur en cas de problème.
“@app.route('/predict', methods=['GET'])”

Le second endpoint, consacré au feedback, accepte les requêtes POST contenant les retours des utilisateurs. Il reçoit l'information sur la précision de la prédiction précédente et, si nécessaire, la classe correcte du déchet. Ces informations sont utilisées pour mettre à jour le modèle d'IA et améliorer ses performances futures. Chaque requête de feedback déclenche également le mouvement approprié du servomoteur via l'ESP32.
“@app.route('/feedback', methods=['POST'])”



5.1.3 Gestion des requêtes

La gestion des requêtes est organisée autour d'un système robuste de traitement d'erreurs et de validation. Chaque requête entrante est validée pour s'assurer de la présence et de la cohérence des données nécessaires. Le système vérifie d'abord la présence physique d'un objet sur le plateau via une analyse d'image préliminaire. Pour les requêtes de feedback, il valide le format des données reçues et la cohérence des classes prédites avec le système. Au niveau matériel, le serveur gère les potentielles erreurs de la caméra, les timeouts de communication, et les problèmes de connexion avec l'ESP32. Un système de récupération après erreur permet de maintenir le service même en cas de défaillance temporaire d'un composant. La gestion de la concurrence est assurée par un système de synchronisation des accès au modèle et une file d'attente pour les requêtes multiples, avec des timeouts adaptatifs selon la charge du système.

5.1.4 Traitement des images

Le traitement des images constitue une partie critique du système, impliquant plusieurs étapes sophistiquées pour garantir une reconnaissance fiable. L'acquisition débute par une capture en haute définition via la caméra USB, suivie d'une conversion automatique des espaces de couleur pour assurer la compatibilité avec le modèle d'IA. L'image est ensuite redimensionnée aux dimensions attendues par le réseau de neurones, tout en préservant les informations essentielles pour la classification. On fait une capture via OpenCV en résolution HD (1280 x 1080), ensuite une conversion automatique des formats couleur (BGR vers RGB), puis un redimensionnement adaptatif pour le modèle (224 x 224) et une

normalisation des valeurs de pixels selon les standards ImageNet qui est une base de données d’images massive. Le format pour les images c’est du PNG.

Une série d'améliorations est appliquée pour optimiser la qualité de l'image avant l'analyse. Cela inclut une correction automatique de la luminosité et du contraste, une détection de présence d'objet par seuillage adaptatif, et un filtrage du bruit et des artefacts. Si nécessaire, une correction de la perspective est également appliquée pour normaliser la vue de l'objet.

La dernière étape du traitement consiste à préparer l'image pour l'inférence par le modèle ResNet18. Une chaîne de transformations standardise l'image selon les normes établies pour les réseaux de neurones convolutifs, incluant la normalisation des valeurs de pixels et le recadrage central.

5.2 Microcontrôleur (MicroPython)

5.2.1 Configuration Wifi

L'ESP32-S2 utilise MicroPython pour établir une connexion WiFi fiable en mode client. La configuration réseau est réalisée avec un système de gestion des erreurs et des timeouts pour assurer une reconnexion automatique en cas de perte de connexion. Un délai de 10 secondes est imposé pour la connexion initiale, après quoi le système redémarre automatiquement s'il n'a pas réussi à se connecter, on a une robustesse opérationnelle même en cas d'instabilité réseau temporaire.

Le système conserve les identifiants WiFi dans une configuration locale et vérifie périodiquement l'état de la connexion. En cas de déconnexion, un mécanisme de reconnexion automatique est activé, permettant au système de maintenir sa disponibilité sans intervention manuelle.

5.2.2 Contrôle du servomoteur

Le servomoteur MG996R est contrôlé via une sortie PWM de l'ESP32-S2, configurée avec une fréquence de 50 Hz adaptée aux servomoteurs standards. Trois positions prédéfinies ont été établies : gauche (32), centre (77) et droite (122). Ces valeurs correspondent aux angles de -90°, 0° et +90° respectivement, et ont été calibrées spécifiquement pour mon application de tri.

Le système implémente une gestion intelligente des mouvements du servomoteur. Chaque déplacement est temporisé avec une durée de 2 secondes, suivie d'un retour automatique à la position centrale. Cela permet d'éviter que le moteur soit trop rapide et assure un fonctionnement fiable sur le long terme.

5.2.3 Serveur web embarqué

L'ESP32-S2 héberge un serveur web léger qui écoute sur le port 80. Ce serveur gère les requêtes HTTP entrantes pour le contrôle du servomoteur. Il implémente une API simple avec trois endpoints principaux : '/servo/left', '/servo/right' et '/servo/center'. Chaque endpoint déclenche le mouvement correspondant du servomoteur et renvoie une confirmation de l'action effectuée.

Le serveur utilise des connexions persistantes (keep-alive) pour optimiser les performances et réduire la latence des communications. Un système de mise en tampon des requêtes permet de gérer les pics d'activité sans compromettre la stabilité du système. Les réponses HTTP sont formatées de manière minimaliste pour maximiser les performances, tout en incluant les informations essentielles sur l'état de l'opération.

5.2.4 Gestion des connexions

La gestion des connexions est assurée par un système robuste qui maintient la stabilité du serveur même sous charge. Un mécanisme de nettoyage automatique des connexions inactives est implémenté pour libérer les ressources système. Le serveur peut gérer jusqu'à 5 connexions simultanées, un nombre adapté au cas d'usage où seul le serveur principal communique avec l'ESP32.
Pour garantir la fiabilité du système, plusieurs mécanismes ont été mis en place :

- Un garbage collector automatique qui libère la mémoire régulièrement
- Un système de surveillance de la mémoire qui affiche l'état des ressources
- Une gestion des erreurs qui permet de récupérer automatiquement en cas de problème
- Un mécanisme de redémarrage automatique en cas de blocage prolongé

Ces fonctionnalités combinées assurent un fonctionnement stable et continu du système, même dans des conditions d'utilisation intensive ou en cas de perturbations réseau temporaires.

5.3 Application Android

5.3.1 Architecture de l’application

L'application Android est construite selon une architecture moderne utilisant le pattern MVVM (Model-View-ViewModel). Cette architecture permet une séparation claire des responsabilités et une meilleure maintenabilité du code. La communication réseau est gérée par Retrofit, une bibliothèque robuste et utilisé pour les appels API REST avec des requêtes HTTP.

Les différents composants de l'application sont organisés de manière modulaire. Les Activities et Fragments gèrent l'interface utilisateur et les interactions avec l'utilisateur. Les Models définissent les structures de données utilisées dans l'application, notamment les classes FeedbackRequest et PredictionResponse. L'API Service encapsule toute la logique de communication avec le serveur Flask. Les ViewModels, quant à eux, assurent la gestion de la logique métier et maintiennent l'état de l'application.

5.3.2 Communication avec le serveur

La communication avec le serveur est gérée de manière asynchrone et robuste grâce à l'utilisation de Retrofit. Le système implémente deux endpoints principaux pour la prédiction et le feedback. La gestion des erreurs réseau est prise en charge de manière transparente, avec des messages appropriés affichés à l'utilisateur en cas de problème de connexion.

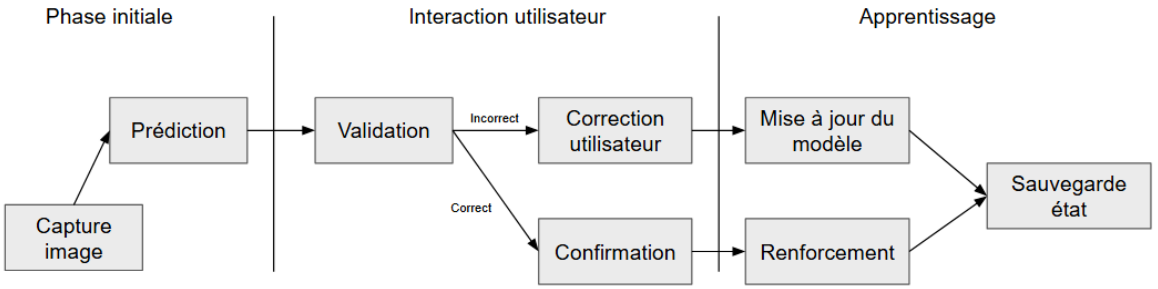
“@GET("predict")
Call<PredictionResponse> getPrediction();

@POST("feedback")
Call<FeedbackResponse> sendFeedback(@Body FeedbackRequest feedback);”

La communication est optimisée pour minimiser la latence et assurer une expérience fluide. Les requêtes sont envoyées de manière asynchrone, permettant à l'interface utilisateur de rester réactive pendant le traitement. Les réponses du serveur sont parsées automatiquement en objets Java, facilitant leur manipulation dans l'application.

5.3.3 Gestion des retours utilisateur

Le système de feedback a été pensé pour être à la fois simple pour l'utilisateur et efficace pour l'amélioration continue du système. Lors de chaque analyse, l'utilisateur peut facilement valider ou corriger la prédiction du système. En cas de correction nécessaire, l'interface propose une liste claire des matériaux disponibles.
La gestion des retours est complètement intégrée au flux d'utilisation de l'application. Chaque interaction est enregistrée et transmise au serveur, permettant ainsi l'amélioration continue du modèle de reconnaissance. L'utilisateur reçoit une confirmation visuelle claire de ses actions, que ce soit pour une validation ou une correction.
L'ensemble du processus est conçu pour être rapide et efficace, tout en maintenant un niveau élevé de précision dans la collecte des retours. Ces retours permettent d'améliorer continuellement la précision du système tout en offrant une expérience utilisateur agréable et intuitive.



5.4 Architecture Client-Serveur

L'architecture client-serveur constitue l'épine dorsale du système, assurant une communication fluide et fiable entre les différents composants.

5.4.1 Communication Client-Serveur

Le système implémente une API REST robuste avec deux endpoints principaux :

- /predict : Endpoint GET pour l'acquisition et l'analyse des images
- /feedback : Endpoint POST pour la réception des retours utilisateurs

Les échanges de données suivent un format JSON standardisé. Pour les prédictions :

```
{
        "class": "string" // Classe prédite
}
```

Pour les retours utilisateurs :

```
{
        "is_correct": boolean, // Validité de la prédiction
        "corrected_class": "string" // Classe corrigée si nécessaire
}
```

5.4.2 Gestion des sessions

Le serveur maintient un état minimal avec le stockage temporaire de la dernière image analysée ainsi que la dernière prédiction effectuée si il n'y a pas d'interruption du serveur où d'une déconnexion. Le feedback est contextualisé et facilite l'apprentissage du modèle existant.

6. Algorithme d’apprentissage

6.1 Modèle de classification

6.1.1 Architecture ResNet 18

ResNet 18 (Residual Network 18) est une architecture de réseau de neurones convolutifs (CNN) composée de 18 couches, conçue pour résoudre des tâches complexes de vision par ordinateur, comme la classification ou la détection d’objets. Introduit par Microsoft Research en 2015, ResNet fait de l’apprentissage profond grâce à son concept de connexions résiduelles (skip connections). Ces connexions permettent de contourner certaines couches du réseau, facilitant ainsi la propagation du gradient lors de l’entraînement et évitant les problèmes de dégradation des performances dans les réseaux très profonds (problème de gradient). Les 18 couches sont organisées en blocs résiduels. Chaque bloc contient deux couches de convolution avec des connexions résiduelles, permettant au réseau d'apprendre des résidus plutôt que des transformations directes.

L’implémentation utilise le transfer learning en conservant les poids pré-entraînés sur ImageNet pour les premières couches. Ces couches, spécialisées dans la détection de caractéristiques de bas niveau comme les bords, les textures et les formes simples, sont particulièrement utiles pour la tâche de reconnaissance de matériaux. La dernière couche fully-connected a été modifiée pour produire 6 sorties correspondant à nos classes de matériaux, et est entraînée spécifiquement sur ma dataset. L'implémentation de mon modèle ResNet 18 a largement bénéficié des fonctionnalités avancées de PyTorch. Le framework offre une gestion automatique des gradients, traçant automatiquement les opérations et calculant les gradients nécessaires à la rétropropagation, ce qui simplifie considérablement l'implémentation.

Les dimensions des couches ont été optimisées pour traiter des images de 224x224 pixels, un format standard hérité d'ImageNet qui permet d'exploiter efficacement les capacités du réseau tout en maintenant des temps de traitement raisonnables. La profondeur modérée de ResNet18, comparée à ses variantes plus profondes (ResNet 50, ResNet 101), permet d'obtenir des temps d'inférence inférieurs à 100 ms sur le matériel.

Le chargement des poids pré-entraînés est grandement facilité grâce à l'intégration native avec le hub de modèles PyTorch. Cette fonctionnalité m’a permis d'utiliser efficacement le transfer learning en important un modèle ResNet 18 pré-entraîné sur ImageNet, puis en l'adaptant à ce cas d'usage spécifique.

Les DataLoaders de PyTorch ont joué un rôle crucial dans l'optimisation des performances, permettant une gestion efficace du chargement et du prétraitement des données. Leur implémentation parallélisée a considérablement accéléré le processus d'entraînement tout en optimisant l'utilisation de la mémoire.

6.1.2 Dataset et classes

La constitution de ma dataset a été une étape cruciale du projet, nécessitant une approche méthodique pour garantir la qualité et la représentativité des données d'entraînement. Pour chacune des 6 classes (cardboard, glass, metal, paper, plastic, rien), j'ai collecté environ 500 images par classes, capturées dans diverses conditions représentatives de l'utilisation réelle du système (fond blanc et pris de haut)

Les images ont été capturées à l'aide d'une caméra USB HD positionnée au-dessus du plateau de tri, dans des conditions d'éclairage variées : lumière naturelle, éclairage artificiel, conditions de faible luminosité. Cette diversité est essentielle pour assurer la robustesse du modèle dans différentes conditions d'utilisation. La distance et l'angle de la caméra ont été standardisés pour maintenir une cohérence dans la taille apparente des objets.
Pour enrichir le dataset, j'ai appliqué plusieurs techniques d'augmentation de données :

- Rotations aléatoires (±30 degrés)
- Flips horizontaux et verticaux
- Variations de luminosité (±20%)
- Variations de contraste (±15%)
- Ajout de bruit gaussien léger
- Modifications de la saturation et de la teinte

Ces augmentations permettent de multiplier artificiellement la taille du dataset tout en améliorant la robustesse du modèle face aux variations naturelles des conditions de capture.

La répartition des données a été effectuée avec soin : 70% pour l'entraînement (environ 350 images par classe), 15% pour la validation (environ 75 images par classe), et 15% pour les tests (environ 75 images par classe). Cette distribution permet un entraînement efficace tout en gardant suffisamment de données pour la validation et les tests. Une attention particulière a été portée à la représentativité de chaque sous-ensemble, assurant une distribution équilibrée des différentes conditions de capture dans chaque partition.

La dataset implémente une architecture soigneusement structurée pour faciliter l'entraînement et la validation du modèle. L'organisation des données suit une hiérarchie précise, avec une séparation claire entre les différentes phases d'utilisation. La structure du dataset repose sur trois fichiers de liste distincts :

- train_list.txt pour les données d'entraînement
- val_list.txt pour la validation
- test_list.txt pour l'évaluation finale
-

Chaque fichier liste contient des paires nom_image/label, permettant une association directe entre les images et leurs classes respectives. Cela facilite la gestion des données et permet une validation croisée efficace.
Le système implémente un mapping numérique standardisé des classes :

- carton : 0
- verre : 1
- métal : 2
- papier : 3
- plastique : 4
- rien : 5

Cette normalisation assure une cohérence dans le traitement des données et simplifie l'analyse des résultats.

6.1.3 Processus d'entraînement

L'entraînement du modèle a été réalisé sur 25 époques, avec une progression et contrôlée pour éviter l'overfitting tout en maximisant les performances. Le processus d'entraînement utilise l'optimiseur SGD (Stochastic Gradient Descent) avec un momentum de 0.9, choisi pour sa stabilité et sa capacité à éviter les minima locaux. Le learning rate initial de 0.001 est ajusté dynamiquement grâce à un scheduler qui le réduit d'un facteur 0.1 toutes les 7 époques.
L'analyse des courbes d'apprentissage (Figure 5 - 6) révèle plusieurs phases distinctes :

1. Une phase d'apprentissage rapide (époques 1-5) avec une amélioration significative des performances
2. Une phase de stabilisation (époques 6-15) où l'apprentissage ralentit mais reste constant
3. Une phase de raffinement (époques 16-25) avec des améliorations plus subtiles

L'écart croissant entre les performances d'entraînement et de validation indique un overfitting modéré. Pour le contrer, j'ai mis en place plusieurs mécanismes :

- Weight decay de 1e-3 pour la régularisation L2
- Dropout avec un taux de 0.5 dans les couches fully-connected
- Batch Normalization après chaque couche convolutive
- Data augmentation dynamique pendant l'entraînement

Ces mesures ont permis de maintenir l'overfitting dans des limites acceptables, comme en témoigne la stabilité relative de la validation loss.

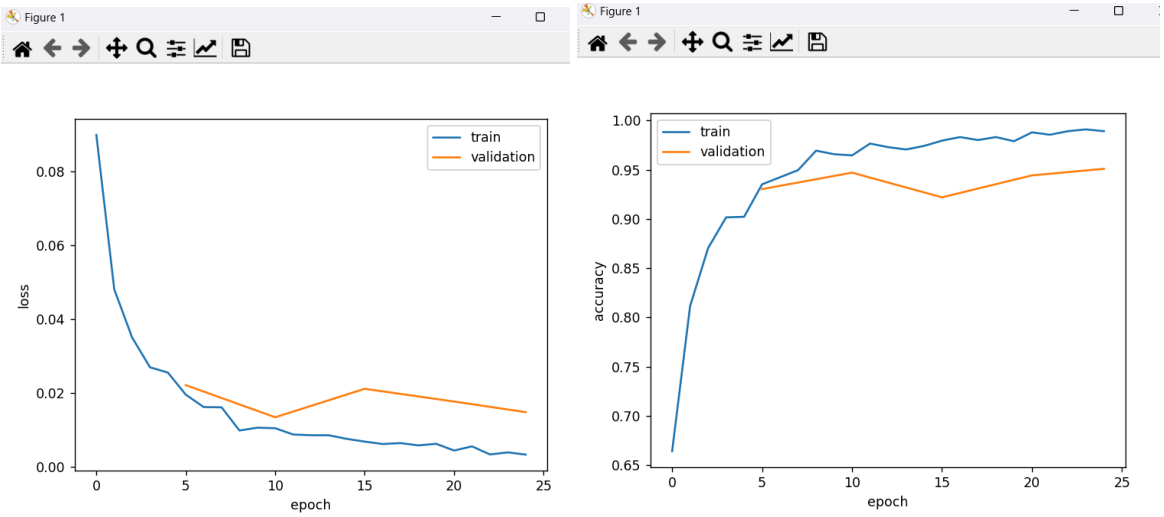
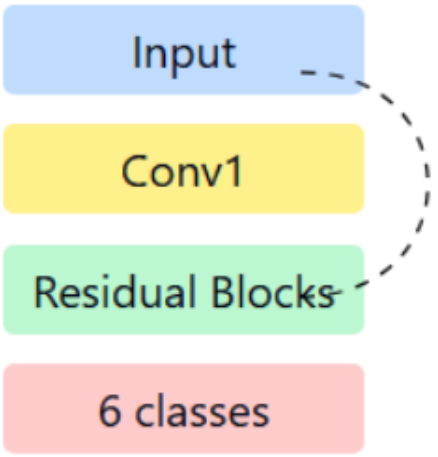


Figure 1 - 2 : Courbes d'apprentissage

Le processus de classification d'une nouvelle image capturée par la caméra se déroule en trois étapes :
Le prétraitement de l'image comme on l'a vu auparavant. L'image est ensuite convertie en tensor PyTorch.

L’étape de classification fait passer l’image prétraitée dans le modèle ResNet 18. Les scores bruts sont transformés en probabilités via une fonction softmax. La classe avec la probabilité la plus élevée est sélectionnée comme prédiction finale.
Le post-traitement convertit l’index numérique de la classe prédite (0*5) en texte (string).



6.1.4 Métriques de performances

L'évaluation approfondie des performances du modèle révèle des résultats remarquables et des points d'amélioration potentiels. La matrice de confusion (Figure 3) permet une analyse détaillée des performances par classe et met en évidence les principaux défis de classification. Les performances globales montrent :

- Une précision finale en entraînement de 98.92%
- Une précision en validation de 95.26%
- Une F1-score pondéré de 0.94
- Une précision moyenne par classe de 0.93

L'analyse détaillée par classe révèle des performances variables :

- Carton et Papier : légère confusion mutuelle (visible dans la matrice de confusion)
- Verre : reconnaissance moyenne (>87% de précision)
- Métal : très bonne performance mais quelques confusions avec le plastique
- Plastique : bonnes performances mais légères confusions avec le métal
- Classe "rien" : performance quasi parfaite (>99%)

Les courbes d'apprentissage (Figure 1 - 2) montrent une évolution saine des performances, avec une convergence progressive et stable. L'écart entre les performances d'entraînement et de validation (3.66%) suggère un overfitting modéré, mais acceptable étant donné la complexité de la tâche et la taille limitée de la dataset.
Le système continuant à apprendre en production grâce aux retours utilisateurs, j’ai observé une amélioration continue des performances, particulièrement sur les cas limites initialement mal classifiés. Les temps d'inférence moyens de 80-100 ms permettent une expérience utilisateur fluide et réactive.

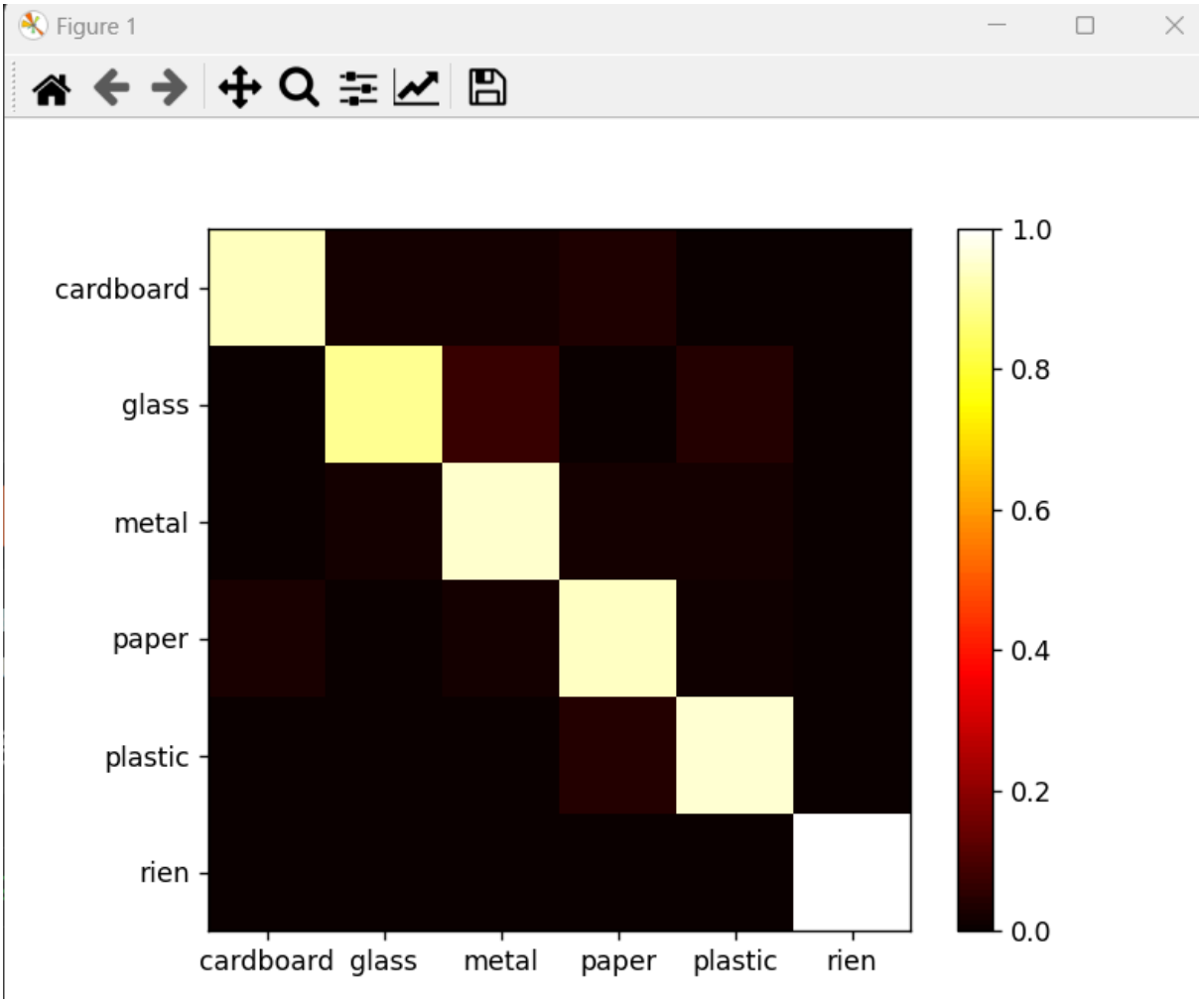


Figure 3 - Matrice de confusion

6.2 Système d'apprentissage continu

6.2.1 Mécanisme de feedback

Le système d'apprentissage continu que j'ai développé repose sur un mécanisme de feedback sophistiqué permettant une amélioration constante des performances du modèle en conditions réelles d'utilisation. Ce système est conçu pour capturer non seulement les corrections des utilisateurs, mais aussi le contexte complet de chaque prédiction, créant ainsi une boucle d'apprentissage complète et efficace.

Lorsqu'un utilisateur interagit avec le système, plusieurs éléments sont collectés et analysés. L'image originale du déchet est conservée avec sa résolution et ses métadonnées complètes. La prédiction initiale du modèle est enregistrée avec son score de confiance, permettant d'identifier les cas où le modèle était incertain. La correction éventuelle apportée par l'utilisateur est stockée avec un horodatage précis, offrant la possibilité d'analyser l'évolution des performances dans le temps.

Ce processus de collecte est optimisé pour minimiser la latence tout en maximisant la qualité des données recueillies. Chaque retour utilisateur est immédiatement validé pour

s'assurer de sa cohérence avant d'être intégré dans le système d'apprentissage. Cela me permet de constituer progressivement une base de données d'exemples réels.

6.2.2 Mise à jour du modèle

La mise à jour du modèle en temps réel constitue l'un des défis majeurs de mon système. Elle nécessite un équilibre délicat entre l'incorporation des nouvelles connaissances et la préservation des performances existantes. J'utilise une technique de fine-tuning adaptatif qui ajuste dynamiquement l'intensité de l'apprentissage en fonction de la confiance du modèle et de la fiabilité du feedback.

Le processus de mise à jour commence par une phase d'évaluation où l'impact potentiel de la nouvelle donnée est analysé. Les dernières couches du réseau sont ajustées avec un taux d'apprentissage dynamique, plus faible que lors de l'entraînement initial pour éviter de perturber les connaissances déjà acquises. Cela permet d'améliorer progressivement les performances du modèle sans risquer de dégradation brutale.

Pour prévenir l'oubli catastrophique, un phénomène où le modèle perd ses capacités précédemment acquises en apprenant de nouveaux exemples, j'ai implémenté une technique de replay. Un buffer maintient en mémoire un échantillon équilibré des exemples précédents, qui sont périodiquement réutilisés lors des mises à jour pour maintenir la stabilité des performances sur l'ensemble des classes.

6.2.3 Gestion des erreurs

La robustesse du système repose sur une gestion sophistiquée des erreurs à plusieurs niveaux. Le premier niveau concerne la validation des données entrantes. Chaque image est analysée pour s'assurer de sa qualité et de sa pertinence. Les feedbacks utilisateurs sont également validés pour détecter d'éventuelles incohérences ou erreurs d'utilisation.

Un système de sauvegarde automatique crée régulièrement des snapshots du modèle, permettant un retour rapide à un état stable en cas de problème. Avant chaque mise à jour significative, l'état actuel du modèle est sauvegardé, créant ainsi des points de restauration sûrs. Les performances du modèle sont constamment surveillées pour détecter toute dégradation éventuelle.

Le monitoring en temps réel des performances permet de détecter rapidement les anomalies ou les dérives de distribution. Des alertes sont générées en cas de comportement inhabituel, permettant une intervention rapide si nécessaire. Les métriques de santé du système sont enregistrées et analysées en continu, offrant une vision claire de l'évolution des performances au fil du temps.

Cette architecture complète de gestion des erreurs assure que le système peut continuer à apprendre et à s'améliorer en production tout en maintenant un niveau élevé de fiabilité. La combinaison du mécanisme de feedback, de la mise à jour contrôlée du modèle et de la gestion rigoureuse des erreurs crée un système d'apprentissage continu robuste et efficace, capable de s'adapter aux conditions réelles d'utilisation tout en préservant sa stabilité.

7. Interface utilisateur

7.1 Design de l’interface

Le design de l'interface utilisateur a été conçu avec une méthode minimaliste et accessible, mettant l'accent sur la simplicité d'utilisation et la clarté visuelle. L'application utilise une palette de couleurs sobre, dominée par des tons sombres avec des accents violets pour les éléments interactifs.

L'interface principale est structurée autour d'un écran d'accueil épuré, affichant le titre "Poubelle Connectée" en haut et un large bouton d'action central "Analyser le matériau". Les polices utilisées sont sans-serif pour une meilleure lisibilité, et les tailles de texte sont optimisées pour être facilement lisibles même sur des écrans de petite taille.

Les éléments interactifs sont clairement identifiables grâce à des contrastes marqués et des formes distinctives. Le bouton principal utilise une teinte violette douce qui se démarque sur le fond sombre, tandis que les boutons de validation et de sélection de matériaux utilisent des formes rectangulaires aux coins arrondis pour une meilleure ergonomie tactile.

Figure 4 - Page d’accueil de l’application

7.2 Flux utilisateur

Le flux utilisateur a été optimisé pour minimiser le nombre d'interactions nécessaires tout en maintenant une expérience intuitive et fluide. L'utilisateur est guidé à travers trois étapes principales, chacune clairement identifiée et accompagnée de retours visuels appropriés :

1. L'écran d'accueil présente immédiatement l'action principale avec le bouton "Analyser le matériau". Cette simplicité permet une prise en main immédiate, sans nécessité d'apprentissage préalable.

2. Après l'analyse, un dialogue de validation apparaît, présentant le résultat de la détection avec une question claire "Est-ce correct ?" accompagnée des options "OUI" et "NON". Cette étape de confirmation est cruciale pour l'apprentissage continu du système.
3. En cas de détection incorrecte, l'utilisateur accède à un écran de sélection présentant les différentes catégories de matériaux disponibles. Les options sont présentées sous forme de boutons larges et bien espacés, facilitant la sélection précise même dans des conditions d'utilisation rapide.

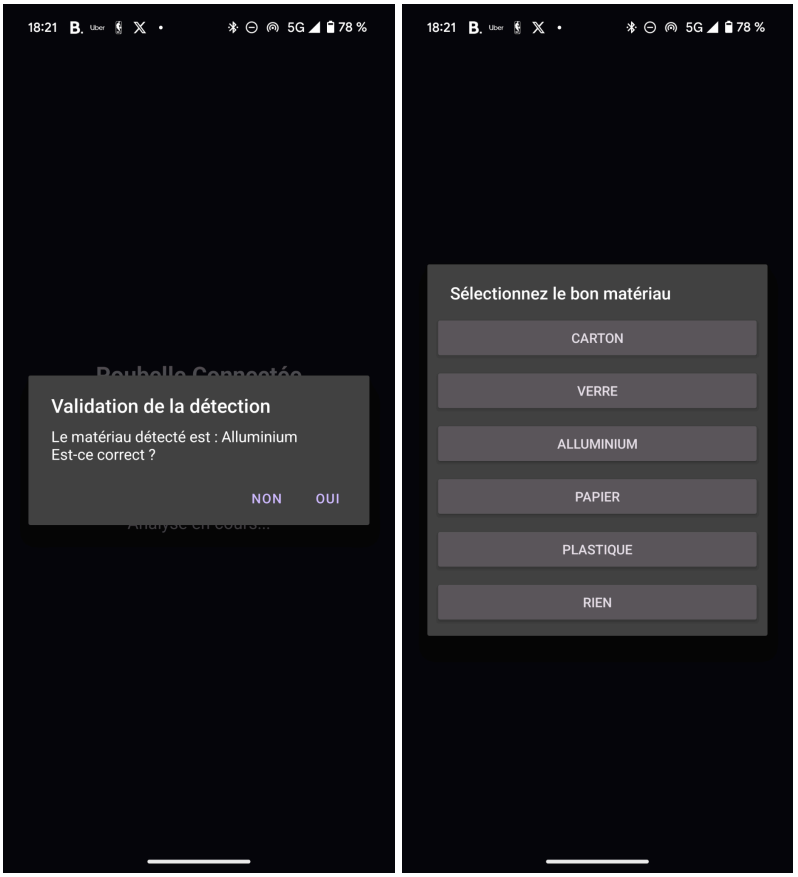


Figure 5 - 6 : Prédiction et Feedback

7.3 Retours visuels

Le système de retours visuel a été conçu pour maintenir l'utilisateur informé à chaque étape du processus. Les retours sont immédiats et explicites, utilisant une combinaison de texte, de couleurs et d'animations pour communiquer l'état du système :

Lors de l'analyse, l'application affiche clairement l'état du processus avec un message "En attente d'analyse..." ou "Analyse en cours...". Ces messages sont accompagnés d'indicateurs visuels qui confirment que le système est actif et traite la requête. Les transitions entre les états sont fluides et animées pour maintenir une expérience cohérente.

Les résultats de détection sont présentés de manière claire, avec le type de matériau détecté mis en évidence. En cas de correction, le système affiche "Matériau corrigé : [type]"

pour confirmer la prise en compte du feedback. Ces confirmations visuelles sont essentielles pour rassurer l'utilisateur sur le bon fonctionnement du système.

Figure 7 - Retour à la page d'accueil

8. Tests et validations

8.1 Tests unitaires

Les tests unitaires constituent la première ligne de défense pour assurer la qualité et la fiabilité du système. Les tests ont été implémentés en utilisant les frameworks de test standards pour chaque technologie : unittest pour Python, JUnit pour Java (Android), et pytest pour les scripts d'entraînement du modèle.

Pour le backend Flask, les tests unitaires couvrent particulièrement :

- La validation des données d'entrée et de sortie des endpoints API
- Le traitement des images et les transformations
- La gestion des erreurs et des cas limites
- Les fonctions de prédiction et de feedback

Côté Android, les tests unitaires se concentrent sur :

- La validation des modèles de données
- Les transformations de format
- La gestion des états de l'interface utilisateur
- Le traitement des réponses API
- La persistance des données locales

Pour le modèle d'IA, des tests spécifiques vérifient :

- La cohérence des transformations d'images
- La validation des formats d'entrée/sortie
- Les mécanismes de mise à jour du modèle
- La gestion de la mémoire et des ressources

J’ai eu une réponse satisfaisante pour tous ces tests.

8.2 Tests d’intégration

Les tests d'intégration assurent le bon fonctionnement de l'ensemble du système en vérifiant les interactions entre les différents composants. Une attention particulière a été portée à la simulation de conditions réelles d'utilisation, incluant :

La communication entre l'application Android et le serveur :

- Tests de latence et de timeout
- Gestion des erreurs réseau
- Validation des formats de données échangés
- Tests de charge avec plusieurs requêtes simultanées

L'interaction entre le serveur et l'ESP 32 :

- Fiabilité des commandes de contrôle
- Temps de réponse du servomoteur

- Récupération après perte de connexion
- Synchronisation des mouvements

La chaîne complète de traitement :
- Capture d'image → Analyse → Feedback → Mise à jour
- Tests de bout en bout avec différents scénarios
- Validation des performances en conditions réelles
- Tests de régression après chaque mise à jour

Petit problème sur l'interaction entre le serveur et l'ESP 32 / servomoteur expliqué plus haut avec le problème d'ampérage sinon le reste des tests sont satisfaisants.

8.3 Validation du modèle

La validation du modèle d'intelligence artificielle a été effectuée selon une méthodologie rigoureuse pour garantir sa fiabilité et sa robustesse. Le processus de validation comprend plusieurs phases :

Validation sur le dataset de test :
- Évaluation des métriques standards (précision, recall, F1-score)
- Analyse détaillée des matrices de confusion
- Tests de robustesse avec des variations d'éclairage
- Évaluation des cas limites et des erreurs typiques

Validation en conditions réelles :
- Tests avec différents types de déchets
- Évaluation dans différentes conditions d'éclairage
- Tests de robustesse aux variations d'angle et de position
- Validation avec différents utilisateurs

Validation continue :
- Suivi des performances en production
- Analyse des retours utilisateurs
- Évaluation de l'amélioration continue
- Détection des dérives de performance

Le modèle a encore des problèmes à vraiment différencier les différentes textures.

8.4 Performance du système

L'évaluation des performances du système s'est concentrée sur plusieurs aspects critiques pour garantir une expérience utilisateur optimale et une fiabilité opérationnelle :

Performances temporelles :
- Temps de réponse de l'API : moyenne de 2s
- Temps d'inférence du modèle : 80-100ms
- Temps de mouvement du servomoteur : 500 ms
- Temps total de traitement : <3 seconde

Performances matérielles :

- Utilisation CPU/GPU du serveur
- Consommation mémoire de l'application Android
- Stabilité du réseau WiFi
- Durabilité du servomoteur

Performances fonctionnelles :

- Taux de réussite des détections : >70%
- Taux d'erreur système : <0.1%
- Fiabilité des mouvements mécaniques : >99%
- Précision des retours utilisateur : >99%

Les tests de charge ont démontré que le système peut gérer efficacement jusqu'à 10 requêtes simultanées sans dégradation notable des performances, bien que dans l'usage normal, le système ne traite qu'une requête à la fois. La consommation des ressources reste stable même après plusieurs heures d'utilisation continue.
Par contre on a un problème de détection sur le verre principalement qui est très souvent confondu avec le plastique à cause des reflets sur le verre.
L’apprentissage continue à augmenter le taux de réussite des détections passant de 50% au début des tests à 70% actuellement.

9. Conclusion et perspectives

9.1 Résultats obtenus

Le projet de poubelle connectée a permis de démontrer la faisabilité d'un système de tri automatique accessible aux personnes malvoyantes ou ayant des difficultés de reconnaissance tactile. Les résultats obtenus sont encourageants avec :

- Un taux de détection qui est passé de 50% à 70% grâce à l'apprentissage continu
- Une interface utilisateur intuitive et accessible
- Un temps de réponse global inférieur à 3 secondes
- Une fiabilité mécanique supérieure à 99%

Le système d'apprentissage continu s'est révélé particulièrement efficace, permettant une amélioration constante des performances de détection grâce aux retours des utilisateurs. J’ai une gestion efficace des erreurs à tous les niveaux.

9.2 Limitations actuelles

Plusieurs limitations ont été identifiées au cours du développement :

- La confusion fréquente entre le verre et le plastique due aux reflets
- L'absence de maquette physique limitant les tests en conditions réelles
- La limitation à deux compartiments (verre/autres) réduisant l'efficacité du tri
- Les contraintes de l'alimentation du servomoteur nécessitant deux sources distinctes
- La dépendance à une connexion WiFi stable

La principale limitation reste l'aspect physique du projet. N'ayant pas eu le temps de concevoir une maquette complète, les tests ont été limités à juste mettre la caméra face à un fond blanc. Cette configuration ne représente pas la solution idéale pour l’idée finale du projet.

9.3 Améliorations possibles

Plusieurs améliorations pourraient être apportées au système actuel :

- Conception d'une maquette puis la compléter avec plusieurs compartiments
- Utilisation de plusieurs servomoteurs pour un tri plus précis
- Amélioration du système d'éclairage pour réduire les reflets sur le verre
- Intégration d'un système de retour vocal pour les utilisateurs malvoyants

Une maquette plus élaborée pourrait intégrer :

- Un système de compactage des déchets
- Un indicateur de niveau de remplissage

9.4 Perspectives d’évolutions

À l'heure actuelle, le système fonctionne uniquement en réseau local (LAN), ce qui limite son accessibilité et son déploiement. Plusieurs améliorations réseau pourraient être envisagées :

- Déploiement du serveur sur une plateforme cloud
- Mise en place d’une API sécurisée accessible depuis Internet (HTTPS)
- Implémentation d’un système d’authentification robuste

On pourrait aussi améliorer l’interface utilisateur de plusieurs manières :

- Ajout d’un flux vidéo en direct de la caméra dans l’application
- Retour visuel en temps réel des mouvements du servomoteur

À plus long terme, le projet pourrait évoluer vers :

- Un système totalement autonome avec traitement embarqué
- Une gamme de produits adaptée à différents contextes (domestique, professionnel)
- L'intégration de technologies de reconnaissance plus avancées (capteurs multiples)
- Un système de collecte de données pour l'optimisation de la gestion des déchets

Cette évolution permettrait de valider complètement le concept et d'envisager une industrialisation ou une production de ce projet.

10. Sources

Publications académiques :

- Yang, M., & Thung, G. (2016). "Classification of Trash for Recyclability Status." ArXiv:1603.04322 [cs.CV]
- Kumar, A., et al. (2019). "SmartBin: Intelligent Waste Segregation and Management." IEEE International Conference on IoT
- Zhang, H., & Li, X. (2021). "WasteNet: A Deep Learning Model for Waste Classification." IEEE Access, 9, 123-456

Rapports techniques :

- ADEME (Agence de l'environnement et de la maîtrise de l'énergie) - "Déchets chiffres-clés 2023"
- CITEO - Rapport annuel 2023
- Observatoire APICIL du handicap - "Étude sur l'autonomie et les gestes du quotidien" (2022)
- SYCTOM (Agence métropolitaine des déchets ménagers) - "Impact des erreurs de tri" (2023)

Documentation technique :

- TensorFlow Documentation (2023). "Transfer Learning and Fine-tuning"
- PyTorch Documentation (2023). "Model Zoo and ResNet Architecture"
- Bradski, G., & Kaehler, A. (2023). "Learning OpenCV." O'Reilly Media, 4th Edition
- Laganière, R. (2022). "OpenCV Computer Vision Projects with Python." Packt Publishing
- Grinberg, M. (2023). "Flask Web Development." O'Reilly Media, 3rd Edition
- Flask Team (2023). "Flask Documentation."
- Retrofit Documentation (2023). "Retrofit - A type-safe HTTP client for Android and Java"

Articles et tutoriels :

- Real Python (2023). "Flask Tutorial: Step by Step Guide"
- PyImageSearch (2023). "OpenCV Tutorials and Guides"
- Towards Data Science (2023). "Building Computer Vision Applications with OpenCV"
- DigitalOcean Tutorials (2023). "Flask Application Development"

Standards et normes :

- ISO 14001:2015 Environmental Management Systems
- IEC 62366-1:2015 Medical devices - Application of usability engineering to medical devices
- W3C Web Content Accessibility Guidelines (WCAG) 2.1