**UNIVERSITE DE NGAOUNDERE**
FACULTE DES SCIENCES
Département de Mathématiques et Informatique

**THE UNIVERSITY OF NGAOUNDERE**
FACULTY OF SCIENCE
Department of Mathematics and Computer Science

# Mémoire de Master Recherche

Présenté en vue de l'obtention du diplôme de Master
en Ingénierie Informatique

Option : Systèmes et Logiciels en Environnements Distribués

## THÈME

# Prédiction du temps de cuisson de haricots basée sur TinyML en utilisant des données de temps de cuisson et des images de haricots

**année académique : 2024-2025**

**Rédigé par : MBAIGOLMEM DANG-DANG Thiery**
(Licencié en Ingénierie Informatique)
Matricule : **20A179FS**

Sous l'encadrement de :

**Pr. Dr.-Ing. JEAN LOUIS KEDIENG EBONGUE FENDJI**
**Maître de conférence**

**Pr. Dr. Ing. BOUKAR OUSMAN**
**Maître de conférence**

**Septembre 2025**

# Prédiction du temps de cuisson de haricots basée sur TinyML en utilisant des données de temps de cuisson et des images de haricots

MBAIGOLMEM DANG-DANG THIERY

11 septembre 2025

**DEDICACE**

Je dédie ce travail à ma famille avec tous mes sentiments de respect, d'amour, de gratitude et de reconnaissance.

# REMERCIEMENTS

La production de ce mémoire a nécessité beaucoup d'énergies et du temps. J'aimerais remercier ici tous ceux qui de près ou de loin ont contribué et ont donné de leur temps pour que mon étude soit menée à bien. Tout d'abord je rends grâce au Seigneur Dieu Tout Puissant qui m'accorde tous les jours non seulement le souffle de vie, mais l'intelligence et la sagesse pour accomplir des œuvres au quotidien.
Nos remerciements vont aussi précisément à l'endroit du :

— Recteur de l'Université de Ngaoundéré, **Pr MAMOUDOU Abdoulmoumini** , qui a rendu possible notre étude en Master 2 ;
— Doyen de la Faculté des Sciences de l'Université de Ngaoundéré, **Pr DJONKA Dieudonné**, pour avoir organisé nos études tout en préservant la discipline ;
— Chef de département de Mathématiques et Informatique, **Pr.Dr.Ing. DAYANG Paul**, pour l'accompagnement et l'orientation donnée à notre travail ;
— Directeur de ce mémoire, **Pr Dr Ing. FENDJI JEAN LOUIS KEDIENG EBONGUE**, qui nous a accompagnés et encadrés pendant tout le temps de rédaction et a su nous guider à travers des recommandations pour la réalisation de cette étude ;

Ensuite, nous exprimons notre gratitude à tous les enseignants du département de mathématiques et informatique, pour les différents cours dispensés depuis le cycle licence, pour les orientations nécessaires à notre travail et pour la disponibilité dont ils ont fait preuve lorsque nous nous sommes référés à eux.

Enfin, nos gratitudes vont à l'endroit des parents, collègues et camarades, pour les différents moyens et aides nécessaires qu'ils nous ont été apportés. Je citerai :

— Mes parents, pour tout ce que vous faites dans ma vie ;
— **Ousmane H., A. LeBon, Ibrahim A., Layibe P.** , mes camardes de promotion, pour la collaboration et la discussion tout au long de ce travail. Vous êtes les meilleurs ! ;
— Mes frères et sœurs pour leurs convivialités et encouragements.

# Abstract

The optimization of food processing is crucial to improve product quality, reduce costs, and minimize waste. Among these processes, bean cooking time is decisive, as it directly affects texture, nutritional value, and consumer acceptance. Traditional estimation methods, although reliable, are time-consuming, destructive, and unsuitable for online control. This thesis introduces an innovative approach based on *Tiny Machine Learning* (TinyML) to predict bean cooking time from images. A hybrid pipeline combining classification and regression was designed, leveraging lightweight neural architectures (custom CNNs, MobileNetV2, EfficientNet-B0, NASNetMobile) optimized and quantized for embedded deployment. Results highlight that the custom CNN achieves the best balance between accuracy and efficiency. Integrated into an Android application, the system demonstrates the feasibility of fast and reliable on-device prediction. Beyond technical contributions, this work illustrates the potential of TinyML in the agri-food sector and points to new perspectives for sustainable quality control.

**Keywords :** TinyML, computer vision, lightweight neural networks, beans, cooking time.

# Résumé

L'optimisation des procédés de transformation agroalimentaire est un enjeu majeur pour améliorer la qualité, réduire les coûts et limiter le gaspillage. Dans ce contexte, la cuisson des haricots occupe une place centrale, leur temps de cuisson influençant directement la texture, la valeur nutritionnelle et l'acceptabilité du produit. Les méthodes traditionnelles d'estimation, bien que précises, sont chronophages, destructives et peu adaptées à un contrôle en ligne. Ce mémoire explore l'apport du Tiny Machine Learning (TinyML) comme solution innovante pour prédire, à partir d'images, le temps de cuisson des haricots de manière non destructive et adaptée aux environnements contraints. La méthodologie adoptée repose sur un pipeline hybride combinant classification et régression. Après la constitution de jeux de données spécifiques, plusieurs architectures de réseaux de neurones légers (CNN personnalisés, MobileNetV2, EfficientNet-B0, NASNetMobile) ont été évaluées et optimisées par quantification pour un déploiement embarqué. Les résultats obtenus montrent que le modèle CNN conçu sur mesure atteint un compromis satisfaisant entre précision, latence et consommation mémoire. Intégré dans une application Android, le système démontre la faisabilité d'une prédiction embarquée fiable et rapide du temps de cuisson. Au-delà de la contribution technique, ce travail illustre le potentiel du TinyML dans l'agroalimentaire et ouvre des perspectives pour le développement d'outils intelligents, accessibles et économes en énergie, favorisant un contrôle qualité amélioré et durable.

**Mots clés :** TinyML, vision par ordinateur, réseaux de neurones légers, haricots, temps de cuisson.

# Table des matières

# Table des figures

# Liste des tableaux

# Liste des abréviations

| **Abréviation** | **Définition** |
|---|---|
| CNN | Convolutional Neural Network (réseau de neurones convolutifs) |
| TBNet | *Tiny Bean (cooking time) Network*, architecture proposée pour la prédiction du temps de cuisson des haricots à partir d'images. Le suffixe **2** (TBNet2) correspond à une couche finale de 256 filtres, tandis que le suffixe **5** (TBNet5) correspond à 512 filtres. |
| MAE | Mean Absolute Error (erreur absolue moyenne), mesure de l'écart moyen entre les valeurs prédites et observées. |
| RMSE | Root Mean Square Error (racine de l'erreur quadratique moyenne), mesure de l'écart quadratique entre prédictions et observations. |
| $R^2$ | Coefficient de détermination, mesure de la proportion de variance expliquée par le modèle. |
| MAPE | Mean Absolute Percentage Error (erreur absolue moyenne en pourcentage). |
| MaxErr | Maximum Error (erreur maximale observée). |
| TinyML | Tiny Machine Learning, déploiement de modèles d'apprentissage automatique sur des dispositifs embarqués à faibles ressources. |
| XAI | Explainable Artificial Intelligence (intelligence artificielle explicable), ensemble de méthodes visant à interpréter les décisions des modèles. |

# 1 Introduction générale

## 1.1 Contexte et justification

L'optimisation des procédés de transformation agroalimentaire constitue un enjeu stratégique à l'échelle mondiale. Elle conditionne à la fois la qualité et la sécurité des produits, la réduction des coûts de production et la lutte contre le gaspillage alimentaire, dans un contexte marqué par la croissance démographique et les pressions sur les ressources naturelles. Parmi ces procédés, la cuisson des légumineuses — et particulièrement celle des haricots — occupe une place centrale. Consommés dans de nombreuses régions du monde, les haricots représentent une source essentielle de protéines, de fibres et de micronutriments, contribuant à la sécurité nutritionnelle et à la souveraineté alimentaire de plusieurs pays [1]. Le temps de cuisson détermine de manière décisive la texture, la saveur et la valeur nutritionnelle des produits, tout en influençant leur acceptabilité par les consommateurs [2].

Traditionnellement, ce temps est estimé à partir de méthodes empiriques ou destructives, telles que les tests de cuisson ou la mesure de l'absorption d'eau. Bien que fiables, ces techniques présentent d'importants inconvénients : elles sont chronophages, coûteuses et difficilement applicables à un contrôle en ligne. Elles génèrent par ailleurs une dépense énergétique non négligeable, problématique dans un contexte où la maîtrise des consommations énergétiques est devenue prioritaire. Ces limites sont particulièrement contraignantes pour les petites et moyennes unités de production agroalimentaire, souvent dépourvues d'infrastructures lourdes ou d'équipements sophistiqués.

L'émergence de l'intelligence artificielle (IA) et des technologies embarquées ouvre de nouvelles perspectives pour surmonter ces obstacles. Le *Tiny Machine Learning* (TinyML) constitue en particulier une innovation de rupture : il permet de déployer des modèles d'apprentissage automatique directement sur des dispositifs embarqués à faible consommation énergétique et à ressources limitées. Grâce à des architectures légères telles que *MobileNetV2* ou des réseaux de neurones convolutionnels compacts, il devient possible de traiter des données visuelles localement, sur microcontrôleurs ou smartphones, sans recourir à une infrastructure informatique lourde ni à une connexion internet permanente [3]. Cette approche rend l'IA accessible dans des environnements contraints, et s'avère particulièrement pertinente pour les zones rurales et les petites unités de transformation, qui constituent une part importante de la chaîne de valeur agroalimentaire.

Dans ce contexte, l'intégration de la vision par ordinateur et du TinyML dans l'agri-

culture et l'agroalimentaire a déjà montré son efficacité pour l'évaluation de la maturité des fruits, la détection précoce de maladies ou le suivi de la qualité de denrées périssables [4]. L'application de ces avancées à l'estimation du temps de cuisson des haricots représente un enjeu à la fois scientifique et socio-économique : elle permettrait non seulement d'améliorer la maîtrise des procédés industriels, mais aussi de renforcer l'autonomie technologique des producteurs, de réduire les coûts énergétiques et de proposer aux consommateurs des produits de qualité constante. Ce travail s'inscrit ainsi dans une dynamique d'innovation technologique au service du développement durable et de la sécurité alimentaire.

## 1.2 Problématique

Malgré les progrès réalisés, la prédiction précise et non destructive du temps de cuisson des légumineuses reste un défi technique et scientifique. Les méthodes conventionnelles souffrent de plusieurs limites majeures :

— **Temps et coût** : leur mise en œuvre est longue et inadaptée à un contrôle en ligne ou à une production de grande échelle.

— **Manque de portabilité** : les dispositifs existants nécessitent des équipements encombrants et onéreux, peu accessibles aux petites structures.

— **Contraintes matérielles et énergétiques** : l'absence de solutions légères freine l'adoption de technologies avancées dans les zones rurales et au sein des petites unités de production.

Dès lors, la question centrale qui guide ce mémoire est la suivante :

> *Comment concevoir et déployer un système de prédiction du temps de cuisson des haricots, fondé sur l'analyse d'images, qui soit à la fois précis, portable et compatible avec les contraintes matérielles et énergétiques du TinyML ?*

## 1.3 Objectifs et contributions

### 1.3.1 Objectif général

Concevoir, développer et valider un modèle TinyML capable de prédire le temps de cuisson des haricots à partir d'images, de façon non destructive, fiable et en quasi temps réel.

### 1.3.2 Objectifs spécifiques

1. Constituer et prétraiter un jeu de données d'images de haricots annotées avec leur temps de cuisson.

2. Concevoir, entraîner et optimiser des modèles légers adaptés au déploiement embarqué (CNN compacts, MobileNetV2, EfficientNet-lite, etc.).
3. Évaluer les performances des modèles en termes de précision, de consommation mémoire et de temps d'inférence.
4. Intégrer et tester le modèle final sur une plateforme embarquée (smartphone).

### 1.3.3 Contributions attendues

— Une méthodologie reproductible et généralisable pour la prédiction visuelle du temps de cuisson des légumineuses.

— Un modèle optimisé, validé expérimentalement et conforme aux contraintes du TinyML.

— Une démonstration fonctionnelle sur dispositif embarqué, attestant de la faisabilité et de l'impact potentiel de l'approche dans des contextes réels.

## 1.4 Organisation du mémoire

La structure du mémoire a été pensée pour guider progressivement le lecteur, du cadre conceptuel aux contributions pratiques et scientifiques. Elle se décline comme suit :

— **Abstract** : une présentation synthétique du sujet, de la méthodologie, des résultats principaux et des apports du travail.

— **Chapitre 1 — Introduction générale** : mise en contexte scientifique et socio-économique, formulation de la problématique, présentation des objectifs et contributions attendues.

— **Chapitre 2 — Revue de la littérature** : analyse critique des travaux existants liés à la prédiction du temps de cuisson des aliments, au TinyML et aux applications de la vision par ordinateur dans l'agroalimentaire.

— **Chapitre 3 — Méthodologie et conception** : description de la démarche méthodologique, du protocole expérimental et de la conception du système proposé.

— **Chapitre 4 — Implémentation et déploiement** : présentation des choix technologiques, de l'implémentation logicielle et du déploiement sur plateformes embarquées.

— **Chapitre 5 — Résultats et discussion** : analyse et interprétation des résultats, confrontation avec l'état de l'art et discussion des limites.

— **Chapitre 6 — Conclusion et perspectives** : synthèse des contributions majeures et identification de pistes de recherche futures.

# 2 Revue de la littérature

## 2.1 Introduction

La revue de littérature a pour objectif de positionner le présent travail dans le paysage scientifique actuel en identifiant les contributions majeures, les lacunes existantes et les perspectives d'amélioration. Dans le cadre de ce mémoire, l'accent est mis sur deux axes complémentaires : (i) les recherches liées à la prédiction du temps de cuisson des légumineuses et aux travaux connexes dans le domaine de l'alimentation, et (ii) l'émergence du *Tiny Machine Learning* (TinyML) comme solution pour le traitement embarqué de données sous des contraintes strictes de mémoire et d'énergie. L'articulation de ces deux domaines ouvre la voie à des applications innovantes dans le secteur agroalimentaire, en particulier pour la cuisson et la qualité des aliments.

## 2.2 Intelligence artificielle et secteur agroalimentaire

L'intelligence artificielle (IA) a profondément transformé le secteur agroalimentaire en introduisant des outils pour l'automatisation, la classification et l'optimisation des procédés [5]. La vision par ordinateur, notamment, est devenue un outil incontournable pour analyser l'apparence des fruits et légumes, estimer leur maturité, évaluer leur qualité ou encore contrôler certains paramètres de cuisson [6].
Les applications sont multiples : détection de maladies végétales, suivi de la croissance, reconnaissance automatique des produits sur les chaînes de production ou encore estimation de la fermeté des denrées après cuisson. Les récents progrès en IA embarquée démontrent que la combinaison de modèles légers et de microcontrôleurs permet une surveillance en temps réel, directement sur le terrain [7, 8]. Ainsi, l'agroalimentaire bénéficie à la fois des avancées de la vision par ordinateur et des développements de l'IA embarquée.

## 2.3 Prédiction du temps de cuisson des légumineuses et travaux connexes

La cuisson des légumineuses est un processus complexe, influencé par la variété, la taille, la teneur en eau et les conditions de stockage [9, 10]. Traditionnellement, la prédiction du temps de cuisson repose sur des méthodes expérimentales basées sur des mesures physico-chimiques (dureté, humidité, structure cellulaire) ou sur des techniques spectroscopiques telles que le proche infrarouge. Ces approches, bien que précises, restent coûteuses, lentes et difficilement transposables en conditions réelles. Des travaux plus récents utilisent la vision par ordinateur et les réseaux de neurones pour corréler les caractéristiques visuelles (forme, couleur, texture de surface) à la tendreté et au temps de cuisson des haricots [11]. Ces approches présentent l'avantage

d'être non destructives et automatisables, constituant un atout considérable pour une intégration dans des systèmes embarqués.
Les recherches connexes sur d'autres aliments confirment la faisabilité de cette approche. Par exemple, [12] ont étudié la texture du riz après cuisson, tandis que [13] se sont intéressés à la fermeté des pâtes. De même, des travaux sur les fruits et légumes montrent que l'IA peut prédire la maturité ou la tendreté à partir d'images [14]. Ces études suggèrent que l'apparence visuelle contient des informations suffisamment discriminantes pour estimer la texture et le degré de cuisson, renforçant ainsi la pertinence de l'application aux légumineuses.
Néanmoins, malgré ces avancées, les études spécifiques aux haricots restent rares et fragmentaires. Les méthodes existantes sont soit limitées par la taille des échantillons, soit inadaptées au déploiement sur des dispositifs contraints. Cette lacune souligne la nécessité de développer des modèles optimisés pour le TinyML, capables de prédire en temps réel le temps de cuisson à partir d'images.

## 2.4 Convolution et réseaux de neurones convolutifs (CNN)

La convolution est une opération fondamentale des réseaux de neurones convolutifs (CNN) permettant d'extraire automatiquement des caractéristiques pertinentes d'une image.

### 2.4.1 Définition mathématique

Pour une image $I \in \mathbb{R}^{H\times W}$ et un noyau (filtre) $K \in \mathbb{R}^{m\times n}$, la *corrélation croisée* (opération généralement utilisée dans les bibliothèques de deep learning) est définie par :

$$S(i,j) = (I \star K)(i,j) = \sum_{u=0}^{m-1}\sum_{v=0}^{n-1} I(i+u, j+v)\, K(u,v). \tag{2.1}$$

La *convolution* au sens strict correspond au noyau retourné : $K'(u,v) = K(m{-}1{-}u,\, n{-}1{-}v)$ et $S = I * K'$. En pratique, cette distinction n'affecte pas l'apprentissage des filtres.
Pour des tenseurs couleur $I \in \mathbb{R}^{H\times W\times C_{in}}$ et $C_{out}$ filtres, chaque filtre $K^{(c)} \in \mathbb{R}^{k\times k\times C_{in}}$ produit une carte de caractéristiques ; les sorties sont ensuite empilées pour obtenir une sortie de dimension $\mathbb{R}^{H'\times W'\times C_{out}}$.

### 2.4.2 Padding, stride, dilatation

- **Padding** ($p$) : ajout de bordures pour contrôler la taille de sortie. Avec $k$ impair et $p = \frac{k-1}{2}$, on conserve $H' = H$ et $W' = W$.
- **Stride** ($s$) : pas de déplacement du filtre. $H' = \left\lfloor \frac{H-k+2p}{s} + 1 \right\rfloor$ et idem pour $W'$.

— **Dilatation** ($d$) : espacement des éléments du filtre (réception plus large) avec un noyau effectif $k_{\text{eff}} = k + (k-1)(d-1)$.

### 2.4.3 Coût, paramètres et convolutions séparables

Le nombre de paramètres d'une convolution standard $k \times k$ est :

$$\text{Params}_{\text{std}} = k^2 \cdot C_{in} \cdot C_{out}. \tag{2.2}$$

Le nombre d'opérations de type MACs est approximativement $H' \times W' \times \text{Params}_{\text{std}}$. Les **convolutions séparables en profondeur** (*depthwise separable*), utilisées notamment par MobileNet [15], factorisent une convolution standard en : (i) une convolution *depthwise* $k \times k$ appliquée canal par canal, puis (ii) une convolution *pointwise* $1 \times 1$ pour mélanger les canaux. Le nombre de paramètres devient :

$$\text{Params}_{\text{dw-sep}} = k^2 C_{in} + C_{in} C_{out} \quad \text{au lieu de } k^2 C_{in} C_{out}. \tag{2.3}$$

Le rapport de réduction théorique en paramètres et opérations est donc :

$$\frac{\text{Params}_{\text{dw-sep}}}{\text{Params}_{\text{std}}} \approx \frac{1}{C_{out}} + \frac{1}{k^2}, \tag{2.4}$$

ce qui explique les gains substantiels pour des noyaux $3 \times 3$ et de grands $C_{out}$.

### 2.4.4 Autres blocs légers

— **Bottlenecks résiduels** (Inverted Residuals, SE, etc.) : améliorent le flux d'information et la précision à coût quasi constant.

— **Global Average Pooling** : remplace des couches entièrement connectées coûteuses par une moyenne spatiale.

— **Normalisations/activations** : Batch/Group Normalization, ReLU/ReLU6/SiLU — souvent adaptées pour la stabilité sur matériel contraint.

### 2.4.5 Fonction d'activation *Softmax*

La fonction *softmax* est une activation essentielle en sortie des CNN lorsqu'il s'agit de problèmes de classification multi-classes [16, 17]. Elle transforme un vecteur de scores réels (appelés *logits*) en une distribution de probabilité sur les classes.
Soit un vecteur de sorties $z = (z_1, z_2, \ldots, z_K) \in \mathbb{R}^K$, où $K$ est le nombre de classes. La fonction softmax est définie par :

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \quad \text{pour } i = 1, \ldots, K. \tag{2.5}$$

Cette transformation garantit deux propriétés fondamentales :

— **Positivité** : $\text{softmax}(z_i) > 0$ pour tout $i$.

— **Normalisation** : $\sum_{i=1}^{K} \text{softmax}(z_i) = 1$.

Ainsi, chaque sortie peut être interprétée comme une probabilité associée à la classe correspondante. Dans l'entraînement, la fonction softmax est souvent couplée à la fonction de perte *entropie croisée*, qui mesure la divergence entre la distribution prédite et la distribution réelle (étiquettes une-hot). Cette combinaison est particulièrement efficace pour l'optimisation de réseaux de neurones profonds [16].
Enfin, bien que très répandue, la softmax présente une sensibilité aux valeurs extrêmes des logits (phénomène de saturation). Des alternatives comme la *sigmoid multi-label* ou la *temperature scaling* peuvent être utilisées selon les besoins.

### 2.4.6 Encodage des classes : utilisation du Label Encoding

Dans ce travail, les classes textuelles représentant les différentes variétés de haricots (par exemple : `Dor701`, `TY339612`, etc.) ont été converties en entiers grâce à la méthode **Label Encoding**. Cette transformation consiste à associer un entier unique à chaque classe :

$$\texttt{Dor701} \mapsto 0, \quad \texttt{TY339612} \mapsto 1, \quad \ldots$$

Le *Label Encoding* présente plusieurs avantages :

— Il est **simple et efficace**, nécessitant peu de mémoire et facilitant le prétraitement.

— Il est parfaitement adapté aux modèles de **deep learning** où les étiquettes sont ensuite converties implicitement en vecteurs lors de la phase d'entraînement, notamment lorsqu'elles sont couplées à une couche de sortie avec *softmax*.

— Dans notre approche, il s'intègre naturellement au pipeline basé sur **MobileNetV2** comme extracteur de caractéristiques, car les sorties du réseau correspondent directement à des indices entiers de classes.

Toutefois, il convient de noter que le *Label Encoding* introduit artificiellement une notion d'ordre entre les classes (par exemple, $0 < 1 < 2$), ce qui peut poser problème pour certains modèles linéaires sensibles aux relations ordinales [18]. Dans notre cas, ce biais est compensé par l'utilisation d'un réseau de neurones convolutif, pour lequel les indices sont simplement traités comme des catégories distinctes, sans relation hiérarchique implicite.
En résumé, le choix du *Label Encoding* dans ce projet permet de garantir une représentation compacte des étiquettes, compatible avec la couche de sortie *softmax* et la

fonction de perte par entropie croisée, assurant ainsi un apprentissage efficace de la classification des variétés de haricots.

## 2.5 TinyML : principes, avantages, cas d'usage et limites

Le *Tiny Machine Learning* (TinyML) désigne l'exécution de modèles d'apprentissage automatique directement sur des dispositifs embarqués à ressources limitées (microcontrôleurs, capteurs intelligents) [19]. L'objectif est de réaliser des inférences en temps réel localement, avec une faible empreinte mémoire et énergétique, rendant l'IA accessible même dans des environnements à faible connectivité.

### 2.5.1 Contraintes matérielles typiques

Les microcontrôleurs ciblés disposent souvent de : (i) ~32–1024 kB de SRAM, (ii) ~128 kB à quelques Mo de Flash, (iii) fréquences de l'ordre de dizaines à quelques centaines de MHz, (iv) parfois aucune FPU ou seulement une FPU simple précision. Ces contraintes guident la conception des modèles (taille, latence, consommation).

### 2.5.2 Avantages et cas d'utilisation

- **Traitement local** : réduction de la latence et de la dépendance au cloud.
- **Efficacité énergétique** : consommation minimale adaptée aux dispositifs alimentés par batterie.
- **Confidentialité accrue** : les données ne quittent pas le dispositif, limitant les risques de fuite.
- **Accessibilité économique** : microcontrôleurs peu coûteux adaptés à une large diffusion.

Applications emblématiques :

- **Agriculture** : détection de maladies, estimation de la qualité des récoltes, suivi en temps réel [7, 8].
- **Agroalimentaire** : classification des produits, prédiction de la texture et du temps de cuisson.
- **Santé** : suivi des signaux physiologiques sur dispositifs portables.
- **IoT industriel** : capteurs intelligents (maintenance prédictive, détection d'anomalies).

### 2.5.3 Techniques d'optimisation

**Quantification (INT8, FP16)** La quantification convertit les tenseurs (poids/activations) en représentations numériques plus compactes. Le modèle flottant $x \in \mathbb{R}$

est approximé par une valeur quantifiée $q$ dans $[q_{\min}, q_{\max}]$ avec une échelle $s > 0$ et un *zero-point* $z$ :

$$q = \mathrm{clip}\Big(\Big\lfloor \tfrac{x}{s} + z \Big\rceil, q_{\min}, q_{\max}\Big), \quad \hat{x} = s(q - z). \tag{2.6}$$

Cas courants :

— **INT8 (entier 8 bits)** : $q_{\min} = -128$, $q_{\max} = 127$ (quantification symétrique avec $z = 0$ ou quantification affine asymétrique). Gain mémoire d'environ ×4 par rapport au FP32 ; accélérations substantielles si l'ISA/MAC INT8 est disponible (CMSIS-NN, etc.).

— **FP16 (demi-précision)** : stockage en 16 bits flottants. Gain mémoire d'environ ×2 ; accélération dépendante du support FPU/ISA. Souvent utilisé pour les couches sensibles (première et dernière).

**Granularité.** Quantification *per-tensor* (simple, moins précise) versus *per-channel* (préférée pour les couches convolutionnelles). **Stratégies.** *Post-Training Quantization* (PTQ) avec calibration (quelques centaines d'échantillons représentatifs) ; *Quantization-Aware Training* (QAT) qui simule la quantification pendant l'entraînement pour limiter la perte de précision.

**Pruning (élagage)** Suppression de poids ou redondances pour réduire mémoire et calculs. Le pruning *non structuré* (mise à zéro de poids individuels) réduit la taille mais n'accélère pas toujours la latence ; le pruning *structuré* (suppression de canaux, filtres ou blocs) permet des gains réels sur matériel. Des approches progressives maintiennent la précision [20].

**Distillation de connaissances** Un modèle *student* plus compact apprend d'un modèle *teacher* plus large en utilisant des cibles douces (température $T$) et une combinaison de pertes (par ex. entropie croisée et divergence KL sur logits). Cette technique préserve souvent la performance tout en allégeant l'architecture.

### 2.5.4 Avancées et perspectives du TinyML

Évolutions notables :

— **Quantization-Aware Training (QAT)** et **mixed precision** : INT8 majoritaire, FP16/FP32 pour couches sensibles.

— **NAS contraint** (TinyNAS, Mobile-friendly NAS) : recherche d'architectures sous contraintes de latence, RAM et Flash.

— **Tiny Transfer Learning** : réutilisation de modèles pré-entraînés allégés et adaptation avec peu d'échantillons.

— **Kernels optimisés et compilateurs** : CMSIS-NN, TFLite Micro (TFLM), microTVM ; *operator fusion* et planification mémoire avancée.

— **Mesures sur cible** : prise en compte des métriques embarquées (latence réelle, pic RAM, Flash) dès les cycles d'itération.

Ces avancées permettent d'envisager des cas d'usage complexes (vision embarquée, audio, capteurs multimodaux) tout en maintenant un compromis adapté entre précision, latence et énergie sur MCU.

### 2.6 Modèles légers pour dispositifs contraints

La conception de modèles adaptés au TinyML repose sur des architectures légères et optimisées :

— **MobileNet** : basé sur les convolutions séparables en profondeur, réduisant drastiquement le nombre de paramètres [15].

— **EfficientNet** : propose une mise à l'échelle équilibrée en profondeur, largeur et résolution pour maximiser l'efficacité [21].

— **NASNetMobile** : utilise la recherche automatique d'architectures (NAS) pour identifier les modèles optimaux pour mobiles [22].

Ces architectures, combinées à la quantification et au pruning, permettent d'obtenir des modèles performants et économes en ressources, adaptés aux capteurs IoT agricoles ou culinaires [7].

### 2.7 Métriques d'évaluation des modèles

L'évaluation des modèles de prédiction repose sur plusieurs métriques standardisées, adaptées aux problèmes de régression ou de classification. Nous introduisons formellement les métriques utilisées dans ce mémoire et discutons leurs propriétés et limites.

#### 2.7.1 Métriques de régression

Soit un jeu d'observations $\{(y_i, \hat{y}_i)\}_{i=1}^{n}$.

— **Erreur quadratique moyenne (MSE)** :

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Sensible aux grandes erreurs, elle pénalise fortement les écarts importants dans les temps de cuisson.

— **Racine de l'erreur quadratique moyenne (RMSE)** :

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}.$$

Interprétable dans l'unité de $y$ (minutes de cuisson), facilitant la communication des résultats.

— **Erreur absolue moyenne (MAE)** :

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|.$$

Robuste aux valeurs aberrantes, elle mesure l'écart moyen absolu.

— **Erreur absolue moyenne en pourcentage (MAPE)** :

$$\mathrm{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|.$$

Exprime l'erreur en pourcentage. *Limites :* non définie si $y_i = 0$ ; instable si $|y_i|$ est très faible. Une alternative symétrisée (SMAPE) peut être utilisée :

$$\mathrm{SMAPE} = \frac{100}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2}.$$

— **Coefficient de détermination ($R^2$)** :

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i.$$

Mesure la proportion de variance expliquée. Variante *ajustée* :

$$R^2_{\mathrm{adj}} = 1 - (1 - R^2)\frac{n-1}{n-p-1},$$

où $p$ est le nombre de prédicteurs *effectifs*. Utile pour comparer des modèles de complexité différente.

### 2.7.2 Métriques de classification

Pour une matrice de confusion multiclasses $\mathbf{C}$ avec $C$ classes, et en notant $TP_c$, $FP_c$, $FN_c$, $TN_c$ les valeurs par classe $c$ :

— **Précision (Precision)** par classe : $\mathrm{Prec}_c = \frac{TP_c}{TP_c+FP_c}$ ; **Rappel (Recall)** : $\mathrm{Rec}_c = \frac{TP_c}{TP_c+FN_c}$ ; **F1** : $\mathrm{F1}_c = 2\frac{\mathrm{Prec}_c \times \mathrm{Rec}_c}{\mathrm{Prec}_c + \mathrm{Rec}_c}$.

— **Agrégations** : *macro* (moyenne non pondérée des classes), *micro* (calcul sur l'ensemble des instances), *pondérée* (pondération par support de chaque classe).

$$\mathrm{F1}_{\mathrm{macro}} = \frac{1}{C}\sum_{c=1}^{C}\mathrm{F1}_c, \quad \mathrm{Precision}_{\mathrm{micro}} = \frac{\sum_c TP_c}{\sum_c (TP_c + FP_c)}.$$

— **Exactitude (Accuracy)** : Acc = $\frac{\sum_c TP_c}{\sum_c (TP_c + FP_c + FN_c)}$ ; sensible au déséquilibre des classes.

— **Balanced Accuracy** : moyenne des rappels par classe, utile en cas de classes déséquilibrées.

— **ROC–AUC / PR–AUC** (binaire ou one-vs-rest multi-classes) : évaluent la capacité de classement ; PR–AUC est informatif en cas de forte rareté d'une classe.

Dans ce mémoire, ces métriques serviront à la classification éventuelle des variétés ou états de cuisson, ainsi qu'à la régression du temps de cuisson, facilitant une comparaison cohérente.

### 2.8 Bonnes pratiques d'évaluation embarquée

Outre les métriques prédictives, l'évaluation TinyML doit *impérativement* considérer :

— **Latence d'inférence sur cible** (en ms) et **throughput** (inférences/seconde) ;

— **Pic de RAM** (mémoire vive) et **empreinte Flash** (taille du binaire modèle + runtime) ;

— **Consommation énergétique** (en mJ par inférence), lorsque mesurable ;

— **Robustesse** aux variations d'illumination, angle de prise de vue et au bruit de capteurs, conditions typiques du domaine d'application réel.

Ces critères guideront les arbitrages entre précision et contraintes matérielles (voir §2.5.3).

## 2.9 Synthèse des travaux antérieurs

TABLE 2.1 – Synthèse des travaux pertinents pour la revue de littérature

| Réf. | Année | Titre | Objectif | Méthodologie | Résultats | Limites | Pertinence |
|---|---|---|---|---|---|---|---|
| [5] | 2018 | Deep learning in agriculture | Survol IA agriculture | Revue | IA utile classification | Peu de focus sur la cuisson | Contexte agroalimentaire |
| [6] | 2020 | Food cooking estimation | Estimer temps de cuisson via images | Analyse d'images | Corrélation détectée | Échantillon limité | Lien direct avec cuisson |
| [3] | 2021 | TinyML principles | Définir TinyML | Discussion | TinyML possible sur MCU | Manque d'implémentations concrètes | Cadre TinyML |
| [15] | 2017 | MobileNet CNN | CNN léger pour mobiles | Architecture | Bonne précision, faible coût | Coût élevé pour certains MCU | Base pour modèles légers |
| [21] | 2019 | EfficientNet scaling | Optimiser mise à l'échelle CNN | Architecture optimisée | Meilleure efficacité | Toujours lourd pour TinyML | Optimisation CNN |
| [22] | 2018 | NASNetMobile | Optimisation via NAS | Recherche automatique | Architecture optimisée | Complexité de la NAS | Modèles mobiles |
| [23] | 2018 | Quantization NN | Réduire taille modèles | Quantification INT8/FP16 | Réduction mémoire x4 | Perte de précision possible | Compression mémoire |
| [20] | 2016 | Pruning deep nets | Réduction des paramètres | Pruning | Réduction paramètres x10 | Perte de précision si pruning excessif | Réduction mémoire |
| [24] | 2019 | Grain classification | Classifier variétés de grains | CNN grains | Bonne précision | Peu généralisable | Classification grains/haricots |
| [14] | 2020 | Fruit maturity estimation | Prédire maturité et tendreté | CNN fruits/légumes | Prédictions fiables | Dataset limité | Lien cuisson/texture |
| [25] | 2021 | Food defect detection | Détection de défauts alimentaires | Analyse d'images | Détection robuste | Applicabilité restreinte | Qualité alimentaire |
| [26] | 2019 | Texture prediction food | Corréler images et texture | Vision + spectroscopie | Corrélations confirmées | Peu d'applications concrètes | Lien cuisson/texture |
| [7] | 2025 | TinyML micronutrient sensing | TinyML pour détection micronutriments | Revue PRISMA | Modèles hybrides performants | Reporting incomplet | Insight TinyML en agriculture |
| [8] | 2024 | Domain-Adaptive TinyML | Détection maladies via TinyML | 2D-CNN + adaptation domaine | Performance correcte | Chute de performance en conditions réelles | Adaptabilité TinyML agri |
| [27] | 2023 | Survey TinyML | Taxonomie TinyML | Revue PRISMA | Mapping optimisations | Peu d'applications agricoles | Ressources méthodologiques |
| [28] | 2025 | TinyML on-device inference | Avantages de l'inférence embarquée | Revue appli | Gains en latence et vie privée | Peu d'exemples en agro | Justification inférence locale |
| [29] | 2025 | From TinyML to TinyDL | Optimisation TinyDL | Comparatif architectures | Framework global | Peu d'applications cuisson | Inspiration pour l'optimisation modèles |

## 2.10 Conclusion

La littérature montre que l'IA et la vision par ordinateur offrent un potentiel considérable pour prédire le temps de cuisson et évaluer la qualité des aliments. Toutefois, les approches existantes présentent des limites : échantillons restreints, complexité des modèles et manque d'adaptation aux dispositifs contraints. Par ailleurs, la majorité des travaux portent sur d'autres aliments que les légumineuses, laissant un champ de recherche peu exploré.

Le TinyML apparaît comme une réponse pertinente à ces limitations. Grâce à ses techniques d'optimisation (quantification, pruning, distillation) et à ses architectures légères (MobileNet, NAS contraint), il permet d'envisager des systèmes intelligents, embarqués et autonomes, adaptés à des contextes réels. Le présent mémoire s'inscrit dans cette perspective en proposant une approche originale : exploiter le TinyML pour prédire le temps de cuisson des haricots à partir d'images, en évaluant conjointement la précision de la prédiction et les contraintes embarquées (latence, RAM, Flash, énergie).

# 3 Méthodologie et Conception du Système

## 3.1 Introduction

Ce chapitre présente la méthodologie complète et la conception du système proposé pour estimer le *temps de cuisson* ($T_c$, en minutes) des haricots à partir d'images. Le système adopté est hybride et séquentiel : il repose sur deux modèles indépendants et spécialisés — un modèle de **classification** chargé d'identifier si l'image contient un haricot (et, le cas échéant, d'identifier sa variété parmi huit classes) et un modèle de **régression** chargé d'estimer le temps de cuisson lorsque l'image a été validée comme étant un haricot. Ce découpage vise à améliorer la robustesse opérationnelle (éviter des estimations sur des images hors-domaine), à optimiser l'utilisation des ressources embarquées (ne solliciter le modèle de régression que lorsque nécessaire) et à faciliter l'intégration TinyML sur plateformes contraintes (smartphone et microcontrôleur) [30, 31]. L'architecture logicielle de la solution met ainsi en œuvre deux pipelines de prétraitement et deux modèles distincts, chacun optimisé et quantifié selon des objectifs différents : le modèle de classification est conçu pour être très léger et quantifié en INT8 (ex. MobileNetV2 quantifié post-training), tandis que le modèle de régression privilégie la précision et peut utiliser une quantification en float16 ou une quantification mixte selon les tests empiriques. La logique applicative (smartphone) orchestre l'enchaînement classification → décision → régression et l'affichage des résultats à l'utilisateur. Dans la suite du chapitre, nous décrivons les jeux de données (séparés pour classification et régression), l'analyse exploratoire, le protocole de prétraitement distinct pour chaque tâche, les architectures testées, le workflow d'entraînement et d'optimisation TinyML, ainsi qu'une comparaison chiffrée et une discussion critique des choix adoptés.

## 3.2 Vue d'ensemble du pipeline proposé

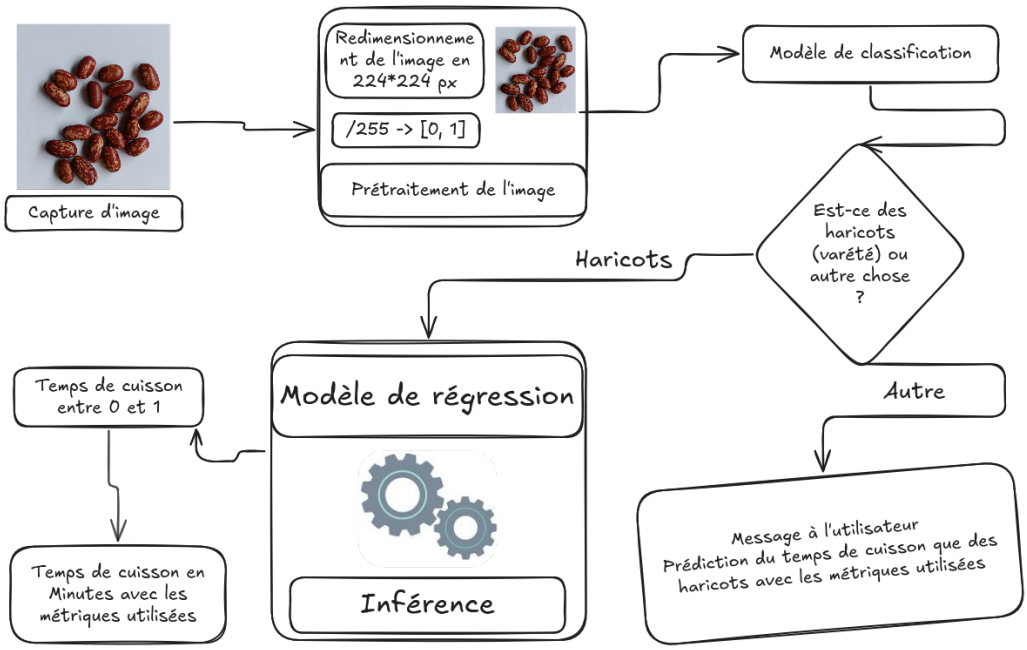



FIGURE 3.1 – Workflow fonctionnel de l'application Android : capture ou import d'image, prétraitement, classification -> inférence et affichage du temps de cuisson ($T_c$) si haricots.

### 3.2.1 Dataset de classification ($\mathcal{D}_{\text{clf}}$)

— **Composition** : 9 classes — huit variétés de haricots, chacune subdivisée en 7 sous-variétés (soit $8 \times 7 = 56$ sous-classes), et une classe supplémentaire `"autre"`. Chaque sous-variété contient 200 images, soit $56 \times 200 = 11\,200$ images pour les haricots. La classe `"autre"` regroupe 1 000 images issues d'objets divers (chien, chat, voiture, personne, etc.), extraites du dataset libre *Common Objects in Context (COCO)* [32]. Ces 1 000 images constituent un échantillon du split validation original du dataset COCO, qui en contient 5 000.

— **Taille totale** : 12 200 images (11 200 images de haricots + 1 000 images de la classe `"autre"`).

— **Splits** :

  — Entraînement/validation : 190 images par sous-variété (soit $190 \times 56 = 10\,640$) + 800 images de la classe `"autre"`, réparties selon une proportion 80/20.

  — Test : 10 images par sous-variété (soit $10 \times 56 = 560$) + 200 images de la classe `"autre"`.

— **Format** : images RGB redimensionnées en 224×224 pixels pour l'entraînement ; métadonnées associées : label de classe, label de sous-varieté, et identifiant d'image.

— **Normalisation de classes** : Les différentes classes de notre jeu de données ont été converties en valeurs numériques à l'aide de la méthode Label Encoding, afin de pouvoir être exploitées par le modèle.

— **Augmentation** : aucune stratégie d'augmentation n'a été appliquée sur ce jeu de données.

— **Problèmes identifiés** : variation de résolution et d'éclairage entre les sources d'images ; risque de confusion inter-sous-variétés de haricots du fait de similarités visuelles ; distribution hétérogène entre classes spécifiques et la classe `"autre"`.

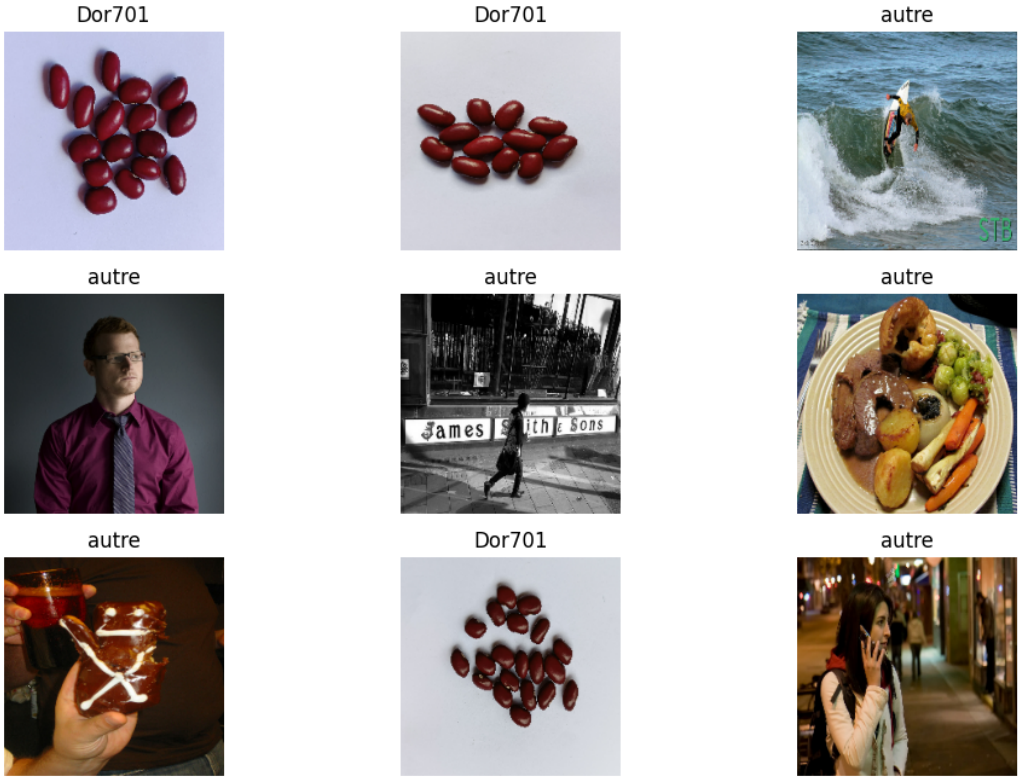



FIGURE 3.2 – Dataset de Classification [Extrait]

### 3.2.2 Dataset de régression ($\mathcal{D}_{\text{reg}}$)

Le second jeu de données est dédié à la tâche de régression visant à prédire le temps de cuisson ($T_c$, exprimé en minutes) des haricots.

— **Composition** : il comprend **8** variétés de haricots (identiques à celles présentes dans $\mathcal{D}_{\text{clf}}$, à l'exception de la classe `"autre"`). Le dataset totalise **11 200** images couleur, réparties uniformément avec **1 400** images par variété 3.3.

— **Annotation** : chaque image est associée à une mesure expérimentale du temps de cuisson $T_c$, déterminée selon le critère culinaire *fork-tender* [33]. Une incertitude expérimentale estimée à ±1 minute (variabilité opérateur et conditions de cuisson) est attachée à chaque échantillon.

— **Splits** : les données ont été séparées selon un ratio 80/10/10 en ensembles d'entraînement, validation et test, avec une seed fixe de **42** pour assurer la reproductibilité. Afin de respecter la distribution des temps de cuisson par variété, une stratification par quantiles de $T_c$ a été appliquée, comme recommandé dans des contextes de régression hétérogène [34].

— **Augmentation** : pour accroître la robustesse et réduire le risque de sur-apprentissage, seules les images d'entraînement ont subi des transformations de data augmentation (flips horizontaux et verticaux, mise à l'échelle, zoom aléatoire). Un facteur global de multiplication ×5 a ainsi été obtenu.

— **Prétraitement** : toutes les images ont été redimensionnées en 224×224 pixels. Contrairement aux pratiques standards de normalisation des intensités dans $[0, 1]$ [35], les images ont été conservées dans l'espace d'origine $[0, 255]$ afin de réduire la taille des fichiers sauvegardés (au format HDF5). La normalisation est réalisée dynamiquement au moment du chargement en mémoire, ce qui optimise le compromis entre efficacité de stockage et préparation des données pour l'entraînement.

— **Normalisation des labels** : les temps de cuisson bruts variaient initialement dans l'intervalle $[51, 410]$ minutes. Pour stabiliser l'entraînement et accélérer la convergence des modèles, une normalisation Min–Max a été appliquée :

$$T_c^{\text{norm}} = \frac{T_c - T_{\min}}{T_{\max} - T_{\min}}, \quad T_{\min} = 51,\ T_{\max} = 410.$$

Ainsi, les labels sont projetés dans $[0, 1]$. Ce choix est cohérent avec la littérature, la mise à l'échelle linéaire facilitant l'optimisation dans les réseaux de neurones et réduisant les effets liés à la différence d'échelle entre variables [17].

— **Observation** : la distribution des temps de cuisson est multimodale et fortement hétérogène entre variétés, certaines ayant une cuisson moyenne relativement courte (proches de 60 minutes), d'autres beaucoup plus longue (supérieure à 300 minutes). Ces caractéristiques influencent le biais de prédiction et rendent

la tâche particulièrement complexe.

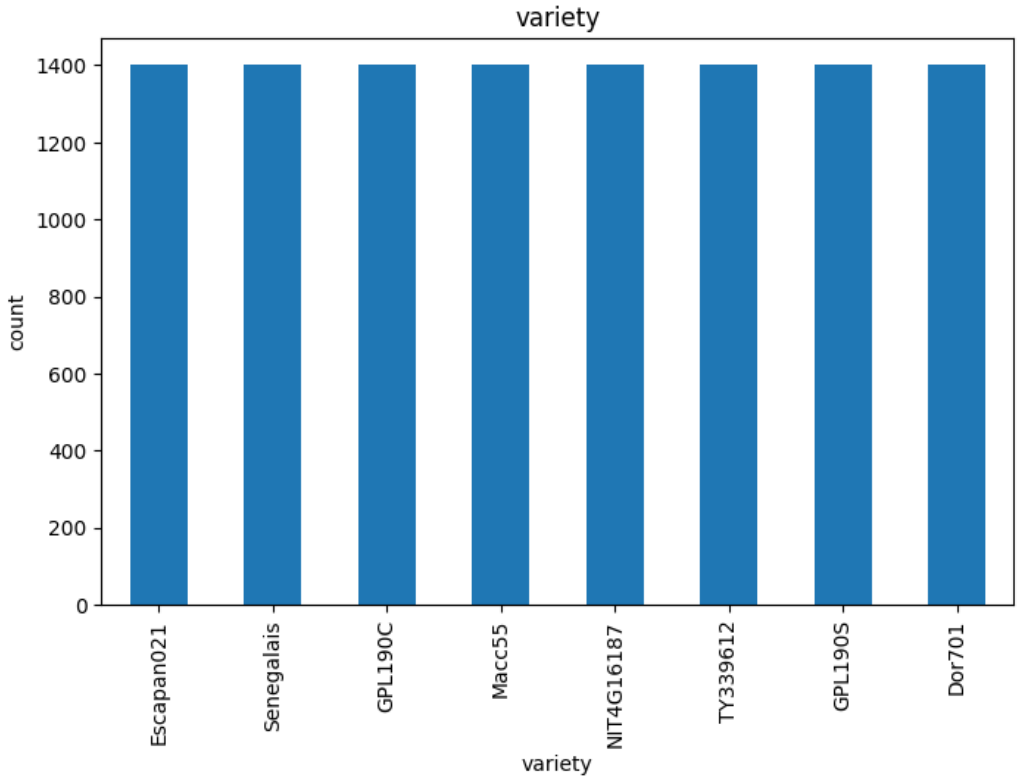



FIGURE 3.3 – Distribution du Dataset de régression par variété

### 3.2.3 Justification du découpage en deux jeux distincts

La séparation en deux jeux de données distincts ($\mathcal{D}_{\text{clf}}$ pour la classification et $\mathcal{D}_{\text{reg}}$ pour la régression) repose sur plusieurs considérations conceptuelles et pratiques :

1. **Sécurité et validité des prédictions** : lors de l'acquisition, des images hors-catégorie (objets non pertinents ou bruit visuel) peuvent être présentes. Entraîner directement un modèle de régression sur de telles images conduirait à des prédictions aberrantes ou hors domaine [36]. L'utilisation d'un modèle de classification préalable permet de filtrer les entrées et de garantir que la régression est appliquée uniquement sur des images pertinentes, améliorant ainsi la robustesse du système.
2. **Différences méthodologiques et d'optimisation** : les tâches de classification et de régression diffèrent par leurs objectifs et contraintes. La classification repose sur un objectif catégoriel (cross-entropy ou focal loss), tandis que la régression vise à prédire une variable continue ($T_c$) avec des fonctions de perte telles que MSE ou MAE [16]. Ces différences impliquent des prétraitements distincts, des augmentations adaptées, et des stratégies de normalisation ciblées

pour stabiliser l'entraînement [35]. Traiter les deux tâches séparément permet d'optimiser finement chaque pipeline sans compromis sur la performance.

3. **Contraintes liées aux systèmes embarqués et à la quantification** : dans un contexte TinyML, la politique de quantification et d'optimisation peut diverger selon la tâche. Les modèles de classification peuvent être fortement quantifiés en INT8 pour réduire la latence et la consommation énergétique, tandis que la régression, nécessitant souvent plus de précision, peut exploiter une quantification plus douce (float16 ou mixte) [37]. Séparer les pipelines permet donc d'adapter la compression et l'optimisation à chaque objectif.

Cette approche modulaire, consistant à découpler la classification et la régression, est largement adoptée dans la littérature sur les systèmes de vision embarquée et la détection d'objets complexes [38, 39].

### 3.3 Analyse exploratoire (EDA) et statistiques descriptives

Cette section résume les analyses réalisées sur $\mathcal{D}_{\text{reg}}$ et $\mathcal{D}_{\text{clf}}$, et met en évidence les implications pour la modélisation.

### 3.4 Analyse exploratoire (EDA) et statistiques descriptives

Cette section résume les analyses réalisées sur $\mathcal{D}_{\text{reg}}$ et $\mathcal{D}_{\text{clf}}$, et met en évidence les implications pour la modélisation.

#### 3.4.1 Synthèse statistique pour la régression

Le tableau 3.1 compile les principales statistiques par variété (moyenne, écart-type, min, max, effectif). Les résultats confirment la forte hétérogénéité inter-variétés : la moyenne de $T_c$ varie de $\approx 118.9$ min à $\approx 237.4$ min tandis que les variances s'étendent largement, notamment pour la variété `Macc55`.

TABLE 3.1 – Statistiques descriptives des temps de cuisson ($T_c$ en minutes) par variété (extrait).

| **Variété** | **Moyenne ($\mu$)** | **Écart-type ($\sigma$)** | **Min** | **Max** | **Effectif** |
|---|---|---|---|---|---|
| Dor701 | 145.86 | 72.67 | 62 | 280 | 1400 |
| Escapan021 | 159.00 | 77.47 | 65 | 287 | 1400 |
| GPL190C | 160.00 | 74.19 | 70 | 295 | 1400 |
| GPL190S | 144.57 | 66.65 | 58 | 270 | 1400 |
| Macc55 | 237.43 | 103.99 | 88 | 410 | 1400 |
| NIT4G16187 | 118.86 | 45.18 | 68 | 210 | 1400 |
| Senegalais | 155.71 | 78.38 | 70 | 300 | 1400 |
| TY339612 | 148.14 | 77.85 | 51 | 286 | 1400 |

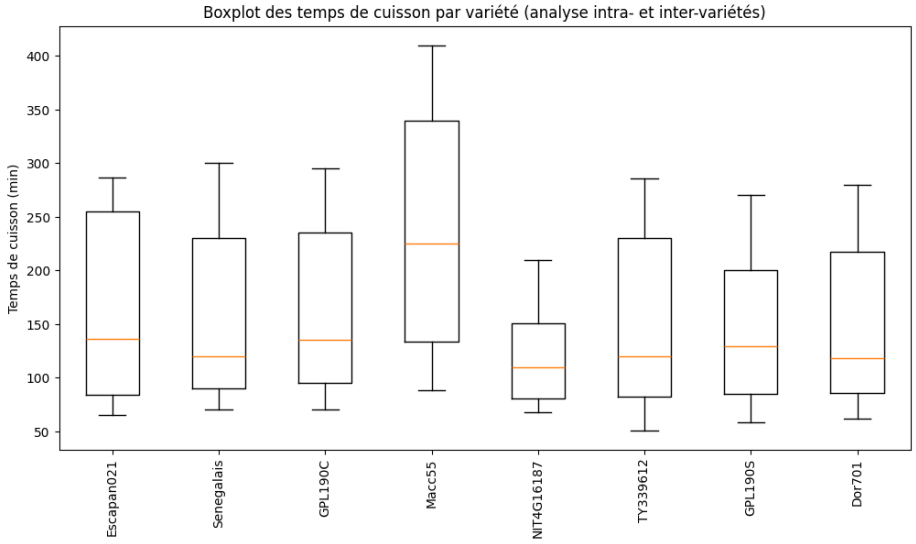



FIGURE 3.4 – Distribution des temps de cuisson par variété : le boxplot met en évidence les médianes, les quartiles et les valeurs extrêmes.

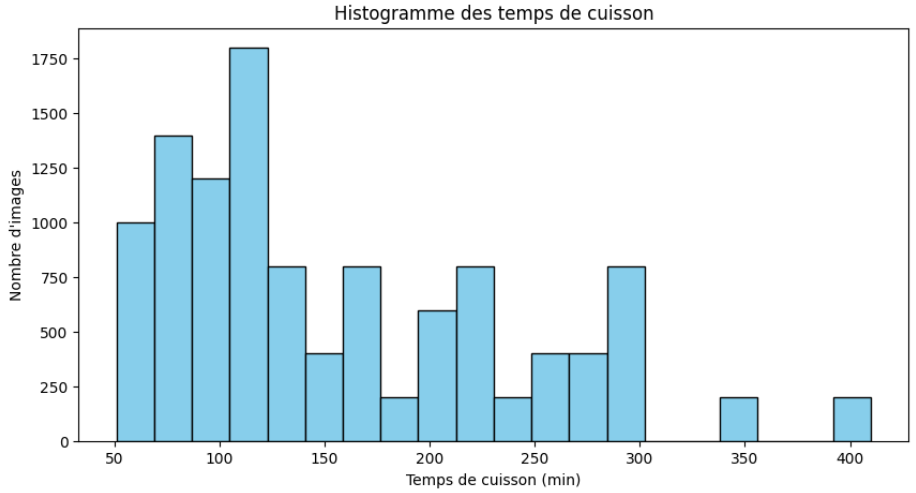



FIGURE 3.5 – Histogramme global des temps de cuisson ($T_c$) toutes variétés confondues. La distribution multimodale reflète l'hétérogénéité du dataset.

**Implications**

— Les variétés à forte variance (ex. `Macc55`) exigeront un modèle capable de capturer une large palette d'apparences visuelles ; l'erreur absolue moyenne (MAE) attendue sera plus élevée pour ces classes.

— Une normalisation Min–Max globale des cibles est possible, mais on doit considérer l'utilisation d'une normalisation par variété ou l'introduction d'un embedding de variété (si la variété est connue) pour améliorer la performance locale.

### 3.4.2 Analyse du dataset de classification

L'analyse exploratoire du dataset $\mathcal{D}_{\text{clf}}$ met en évidence plusieurs points importants :

- **Qualité des images** : toutes les images ont été vérifiées, et aucune image corrompue ou illisible n'a été détectée, garantissant la fiabilité des données pour l'entraînement.
- **Distribution des classes** : le dataset contient 9 classes — 8 variétés de haricots et une classe `autre` issue d'un échantillon du dataset COCO [32]. Chaque variété comprend 7 sous-variétés avec 190 images chacune, soit 1400 images par variété. La classe `autre` contient 1000 images diverses (chien, chat, voiture, personne, etc.), ce qui représente un choix volontaire pour refléter la proportion réaliste d'images hors-catégorie.
- **Variabilité visuelle** : des différences de luminosité, de contraste et de résolution entre sous-variétés ont été observées 3.2, justifiant l'usage futur d'augmentations photométriques pour améliorer la robustesse du modèle [40].

### 3.4.3 Stratégie adoptée pour la classe `autre`

Bien que la classe `autre` soit plus petite que les autres classes, cette répartition est **intentionnelle** afin de reproduire la proportion réaliste d'images hors-catégorie :

## 3.5 Architectures modèles et justification

Nous testons deux familles de modèles : (i) architectures conçues ad hoc (CNN-1 / CNN-2) et (ii) modèles mobiles pré-entraînés (MobileNetV2, EfficientNet-B0, NASNetMobile). Les configurations et motivations restent proches de celles décrites précédemment ; nous rappelons ici les décisions liées au couplage classification / régression.

### 3.5.1 Modèle de classification

- **Backbone recommandé** : MobileNetV2 (1.0, 224) fine-tuné sur $\mathcal{D}_{\text{clf}}$. Raisons : excellente performance/empreinte mémoire, blocs inverted-residual favorables à la quantification INT8 et historique d'utilisation mobile [41].
- **Sortie** : softmax 9 unités (8 variétés + `autre`).
- **Fonction de perte** : categorical cross-entropy avec éventuelle pondération de classes (class weights) pour corriger le déséquilibre.
- **Métriques** : accuracy globale, F1-score macro (pour tenir compte du déséquilibre), recall/precision par classe, matrice de confusion. (Les définitions formelles des métriques sont détaillées dans la revue de littérature.)
- **Optimisations** : PTQ int8 ; si dégradation trop forte, utiliser QAT.

### 3.5.2 Modèle de régression

— **Backbones testés** : CNN personnalisés (TBNet2 / TBNet5) et modèles pré-entraînés (EfficientNet-B0, MobileNetV2, NASNetMobile). Le modèle final retenu dans les expériences précédentes a été un CNN personnalisé entraîné *from scratch*, plus simple et mieux adapté au jeu de données disponible.

— **Sortie** : une seule unité pour la régression continue (valeur normalisée $\tilde{y}$).

— **Fonction de perte** : MSE (principale), complétée par MAE/MAPE comme métriques secondaires et critères d'arrêt ; le coefficient de détermination $R^2$ a également été utilisé pour mesurer la part de variance expliquée.

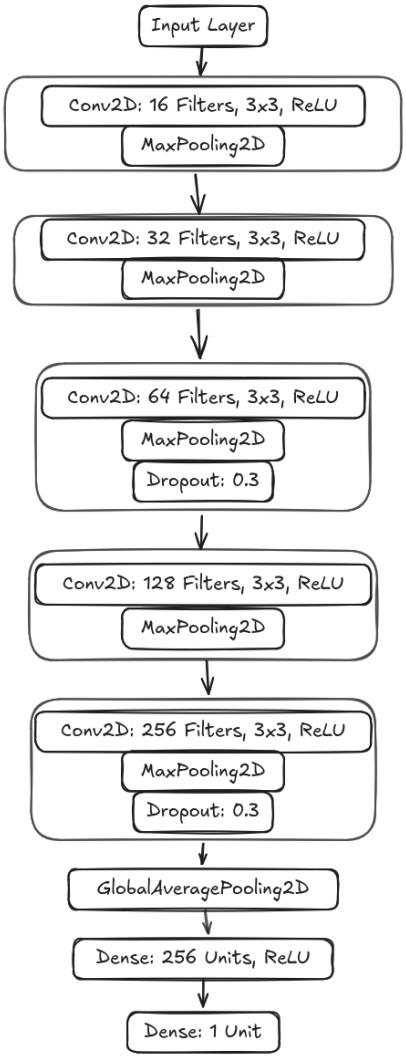
— **Optimisations** : quantification float16 pour préserver la précision numérique ; quantification post-entraînement (PTQ) int8 testée avec calibration pour évaluer son impact.

#### Architectures testées

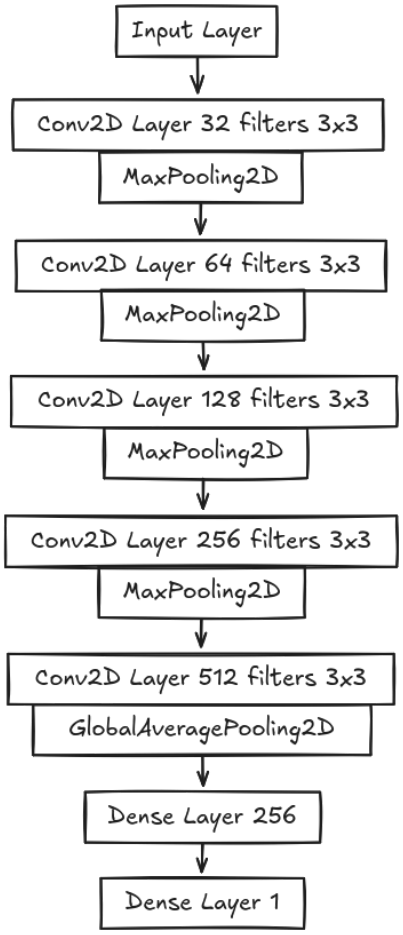
Plusieurs architectures ont été étudiées afin de comparer des modèles *from scratch* (TBNet2, TBNet5) et des modèles *pré-entraînés* (EfficientNet-B0, MobileNetV2, NASNetMobile). Le choix des backbones a été motivé par un compromis entre performance, capacité de généralisation et contraintes liées au déploiement sur microcontrôleurs (TinyML).

**TBNet2** TBNet2 est un CNN personnalisé conçu pour être léger tout en conservant une expressivité suffisante. Il repose sur une progression du nombre de filtres convolutionnels ($16 \rightarrow 256$), combinés à des couches de *max pooling* pour la réduction de dimension et à des couches de *dropout* pour limiter le surapprentissage. La couche de *Global Average Pooling* condense l'information avant une couche dense de 256 neurones et une sortie scalaire adaptée à la régression. Ce modèle est particulièrement adapté à un entraînement *from scratch* sur un volume de données limité, tout en offrant un bon compromis entre simplicité et performance [42].

**TBNet5** TBNet5 prolonge la logique de TBNet2 en augmentant la profondeur et la capacité d'extraction des caractéristiques. Il s'appuie sur des convolutions successives ($32 \rightarrow 512$ filtres), suivies d'un *Global Average Pooling* et d'une couche dense. La profondeur accrue permet de capturer des motifs visuels plus complexes, mais au prix d'un temps de calcul supérieur et d'un risque accru de surapprentissage [16].




(a) Architecture TBNet2 développée pour la régression du temps de cuisson.




(b) Architecture TBNet5 développée pour la régression du temps de cuisson.

FIGURE 3.6 – Architectures CNN personnalisées utilisées pour la tâche de régression.

**EfficientNet-B0** EfficientNet [43] est une famille de réseaux qui optimise simultanément la profondeur, la largeur et la résolution via une stratégie de *compound scaling*. La variante B0, plus compacte, constitue un excellent compromis entre précision et complexité, ce qui la rend adaptée aux environnements contraints. Dans ce travail, EfficientNet-B0 a été utilisé en transfert d'apprentissage, avec des poids pré-entraînés sur ImageNet adaptés à la tâche de régression.

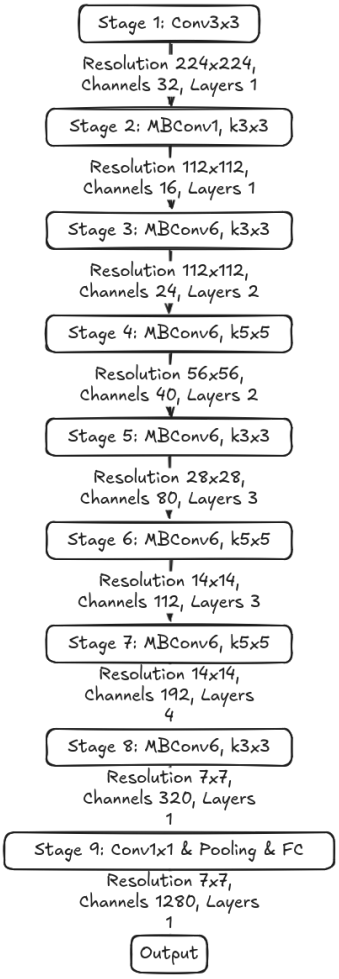



FIGURE 3.7 – Schéma simplifié de l'architecture EfficientNet-B0 (source : [43]).

**MobileNetV2** MobileNetV2 [41] est une architecture optimisée pour les environnements embarqués. Elle repose sur l'utilisation de blocs *inverted residuals* et de convolutions en profondeur séparables (*depthwise separable convolutions*), qui réduisent significativement le nombre de paramètres. Ce modèle est particulièrement adapté aux contraintes de TinyML, offrant un compromis solide entre rapidité d'inférence et précision.

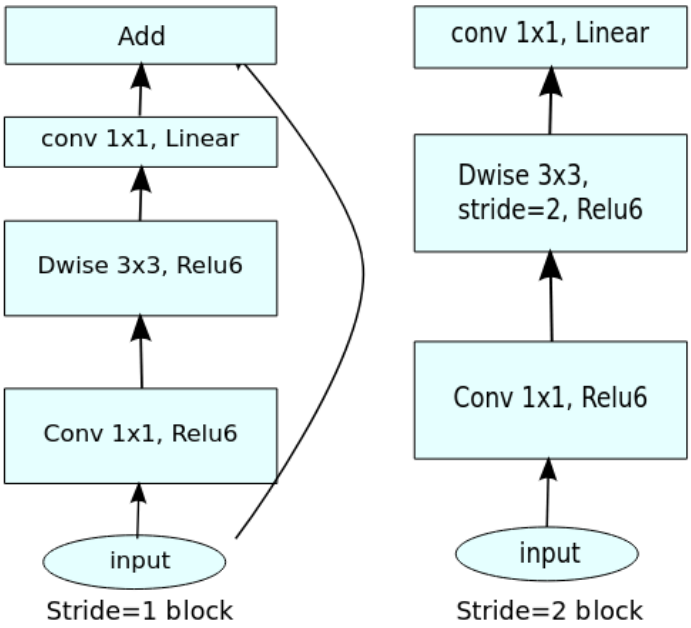



FIGURE 3.8 – Architecture MobileNetV2 utilisée comme extracteur de caractéristiques.

**NASNetMobile** NASNet [22] a été obtenu par recherche automatique d'architecture (*Neural Architecture Search*). La variante Mobile a été spécifiquement optimisée pour des ressources limitées. Elle repose sur la répétition de cellules convolutives normalisées, offrant une bonne expressivité avec un nombre réduit de paramètres. Dans ce travail, NASNetMobile a été testé en *fine-tuning* pour évaluer son compromis entre capacité de représentation et portabilité.

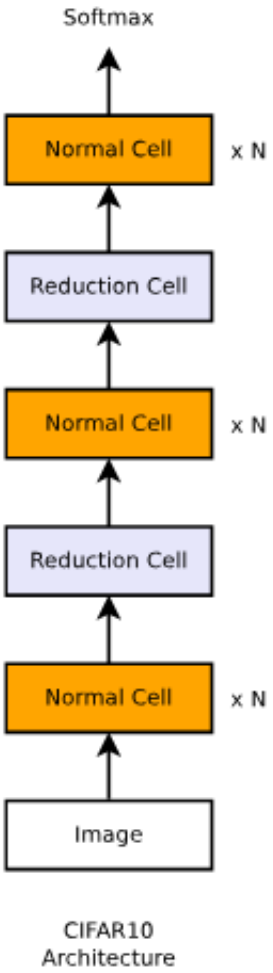



FIGURE 3.9 – Schéma de l'architecture NASNetMobile (source : [22]).

## 3.6 Workflow d'entraînement, conversion et évaluation

### 3.6.1 Procédure d'entraînement reproductible

Pour chaque modèle (classification et régression) :

1. Préparer HDF5 par split avec seed fixe (reproductibilité).
2. Appliquer les augmentations et normalisations correspondantes selon le pipeline.
3. Entraîner avec Adam (baseline) : lr = $10^{-4}$, batch size = 32 (ajuster selon VRAM), scheduler (Cosine ou ReduceOnPlateau), early stopping (patience 5 sur validation loss). Epochs baseline = 100.
4. Sauvegarder les checkpoints, et conserver le meilleur modèle sur la validation.

### 3.6.2 Conversion et quantification TFLite

— **Classification** : conversion PTQ INT8 avec dataset de calibration (subset representatif). Si la performance chute de plus de 25% (seuil à définir empiriquement), exécuter QAT quelques epochs puis reconvertir.

— **Régression** : conversion float16 par défaut ; tester PTQ int8 en gardant un set de test calibrant pour estimer la dégradation MAE/RMSE.

### 3.6.3 Évaluation finale et mesures embarquées

Mesures à effectuer sur la plateforme cible (smartphone Android API26) :

— **Taille binaire** (.tflite) avant/après quantification.

— **Latence** d'inférence (cold start, warm runs), percentiles 50/90/99 sur N exécutions.

— **Concordance** entre sortie du modèle float32 de référence et sortie TFLite (tolerance MAE).

## 3.7 Intégration dans l'application mobile et UX

Au niveau applicatif (smartphone Android), le fonctionnement est le suivant :

1. L'utilisateur capture ou charge une image.
2. L'image subit le pipeline $\mathcal{P}_{\text{clf}}$ puis passe au modèle de classification quantifié INT8.
3. Si le modèle prédit la classe `autre` avec confiance supérieure à un seuil $s_{\text{rej}}$ (ex. 0.25), l'application affiche un message : *"L'application ne prédit le temps de cuisson que pour des images de haricots. Veuillez fournir une photo claire d'un haricot."*
4. Si le modèle prédit une des 8 variétés avec confiance suffisante, l'application déclenche le pipeline $\mathcal{P}_{\text{reg}}$ (prétraitement pour régression) et exécute le modèle de régression (float16).
5. L'interface affiche : la **variété** prédite (avec score), le **temps estimé** $T_c$ (minutes) retransformé depuis l'échelle normalisée, et une estimation d'intervalle d'incertitude (ex. ± MAE sur la variété si disponible). Les métriques globales (acc, F1 pour la classification ; MAE, RMSE pour la régression) sont accessibles dans une section "Plus d'infos" pour les utilisateurs techniques.

**Gestion des seuils et de la confiance** Il est recommandé d'ajouter un mécanisme de rejet (thresholding) pour éviter d'exécuter la régression si la confiance de

classification est basse. De plus, enregistrer ces exemples faibles en confidence pour une future annotation manuelle permettra d'améliorer le dataset (active learning).

### 3.8 Bonnes pratiques et recommandations pour la reproductibilité

- Consigner les seeds aléatoires pour la génération des splits et les augmentations.
- Exporter les artefacts (HDF5/TFRecord, checkpoints, scripts d'entraînement, scripts de conversion TFLite).
- Tester les modèles TFLite sur une suite d'appareils cibles pour mesurer latence et mémoire.
- Intégrer des tests unitaires comparant prédictions numpy/TF/Keras vs TFLite (tolerance MAE).

### 3.9 Conclusion du chapitre

Ce chapitre a présenté une méthodologie complète et reproductible pour la conception d'un système hybride *classification → régression* destiné à prédire le temps de cuisson des haricots à partir d'images. La séparation des jeux de données et des pipelines de prétraitement, l'usage d'un modèle de classification quantifié INT8 pour filtrer le domaine, et d'un modèle de régression optimisé pour la précision (float16) constituent l'ossature de la proposition. Les choix techniques (architectures, augmentations, stratégies de quantification) sont motivés par un compromis précision/latence adapté aux contraintes embarquées. Le chapitre suivant 4 détaillera les scripts Keras/TensorFlow, les commandes de conversion TFLite (PTQ / QAT) et les détails d'entraînement.

# 4 Développement et implémentation

Ce chapitre présente la mise en œuvre concrète du système de prédiction du temps de cuisson des haricots, depuis la préparation des données jusqu'au déploiement embarqué. Il s'appuie sur la méthodologie définie au Chapitre 3 et reprend strictement les choix techniques (prétraitements, architectures candidates, schéma d'entraînement et optimisation TinyML).
L'approche adoptée est **hiérarchique** : une première étape de **classification** permet de distinguer les images de haricots des autres objets, suivie d'une **régression** sur les images de haricots afin de prédire le temps de cuisson.

## 4.1 Environnement de développement

### 4.1.1 Matériel

Les expérimentations ont été menées sur :

- **Machine locale** : Intel Core i7 (4 cœurs), 16 Go RAM.
- **Google Colab** : session `Tesla T4` (15 Go VRAM, 12 Go RAM, 118 Go stockage).
- **Samsung Galaxy Note 9** : Octa-core, 6 Go RAM, 128 Go stockage, Android 10.

### 4.1.2 Logiciels et versions

Sauf mention contraire, les versions utilisées sont celles définies en méthodologie :

- **Python** 3.10
- **TensorFlow** 2.19 pour l'entrainement et **TensorFlow** 2.13 pour la quantification & API Keras (compatibles `TensorFlow Lite`)
- **Pandas** 2.1, **NumPy** 1.25, **Matplotlib** 3.8

## 4.2 Préparation des données

La préparation des données est subdivisée en deux pipelines distincts, correspondant aux **deux modèles** :

1. **Prétraitement pour la classification** : consiste à créer un dataset équilibré avec classes "haricots" et "autres objets", à normaliser et à préparer les images pour l'entraînement du modèle de classification.
2. **Prétraitement pour la régression** : consiste à préparer uniquement les images de haricots, à normaliser les pixels et les labels (temps de cuisson), à appliquer les augmentations, puis à créer les splits train/validation/test.

### 4.2.1 Prétraitement des données pour la classification

Le pipeline de classification consiste à préparer les images pour l'entraînement d'un modèle de type MobileNetV2 afin de déterminer si une image contient des haricots ou autre chose. Chaque étape est décrite ci-dessous avec le pseudo-code correspondant.

```
Bloc 1 : Création des dossiers pour données d'entraînement et de test
Définir base_path pour le dataset
Définir train_path = base_path + "/Trainset/autre"
Définir test_path = base_path + "/Testset/autre"
Créer dossier train_path s'il n'existe pas
Créer dossier test_path s'il n'existe pas
```

Après ce bloc, on s'assure que les dossiers nécessaires pour stocker les images d'entraînement et de test existent. Cette étape est cruciale pour éviter toute erreur lors de la copie ou du chargement des images, et pour structurer correctement le dataset.

```
# Bloc 2 : Téléchargement et extraction du dataset COCO val2017
Si fichier val2017.zip n'existe pas:
    Télécharger val2017.zip depuis URL_COCO
Si dossier val2017 n'existe pas:
    Extraire contenu de val2017.zip dans /content/
Afficher message "COCO val2017 prêt"
```

Ce bloc permet de récupérer un sous-ensemble standard d'images pour la classe "autre". Le dataset COCO val2017 est utilisé comme source d'images non-haricots afin de créer un dataset équilibré pour la classification.

```
# Bloc 3 : Sélection d'un sous-ensemble d'images et copie dans train et test
Lister toutes les images .jpg dans val2017
Prendre un sous-ensemble des 1000 premières images
Copier les 800 premières dans dossier train_path
Copier 200 images suivantes dans dossier test_path
Afficher message "Classe 'autre' créée dans ton dataset"
```

Ici, on sélectionne un nombre limité d'images afin de contrôler le volume de données et garantir un entraînement rapide. Les images sont séparées en ensembles d'entraînement et de test pour évaluer la généralisation du modèle.

```
# Bloc 4 : Chargement des chemins d'images et labels des variétés
Initialiser listes image_paths et labels_variete
Pour chaque dossier de classe dans base_path/Trainset:
    Si le dossier est un répertoire:
        Extraire nom de la variété depuis nom du dossier
        Pour chaque fichier image dans ce dossier:
            Ajouter chemin à image_paths
            Ajouter label variété à labels_variete
Afficher nombre d'images chargées et un exemple
Créer dictionnaire variete_to_idx pour indexer les variétés
Convertir labels_variete en tableau numpy y_variete d'indices
Afficher les variétés détectées et la forme de y_variete
```

Ce bloc construit la liste complète des chemins d'images et leurs labels associés. La conversion en indices numériques est nécessaire pour l'entraînement supervisé d'un modèle de classification.

```
# Bloc 5 : Création d'un dataset TensorFlow à partir des chemins et labels
Définir fonction load_image(path, variete):
    Lire et décoder jpg
    Redimensionner à 224x224 et normaliser pixels
    Retourner image et label variete sous forme dictionnaire

Créer dataset tf.data à partir de (image_paths, y_variete)
Mapper dataset avec load_image
Mélanger dataset, batcher par 32, précharger données de façon optimisée
```

Cette étape permet de créer un pipeline de données efficace pour TensorFlow, incluant le prétraitement (redimensionnement, normalisation) et l'optimisation de l'entraînement via batch et prefetch.

```
# Bloc 6 : Chargement automatique du dataset depuis un dossier avec Keras
Charger dataset depuis train_dir avec Keras image_dataset_from_directory
Taille images redimensionnées à 224x224
Taille batch 32, mélange aléatoire activé
Afficher noms des classes détectées dans train_dir
```

Ce bloc illustre l'utilisation de la fonction Keras 'image_dataset_from_directory', qui simplifie le chargement des images classées par dossier et automatise le prétraitement de base.

```
# Bloc 7 : Comptage des images par classe dans un dossier donné
Définir fonction count_images_per_class(dataset_path):
    Initialiser dictionnaire counts
    Pour chaque classe dans dataset_path:
        Si c'est un dossier:
            Compter fichiers images (.jpg, .jpeg, .png) dans dossier
            Mettre dans counts[classe]
    Retourner counts

Calculer train_counts avec count_images_per_class sur train_dir
Afficher train_counts
```

Le comptage des images par classe permet de détecter un éventuel déséquilibre et de prendre des mesures si nécessaire (sous-échantillonnage ou sur-échantillonnage).

```
# Bloc 8 : Visualisation de la distribution des images par classe (bar chart)
Créer figure matplotlib
Tracer histogramme barres avec counts par classes (train_counts)
Rotation des labels en abscisse
Titre "Distribution des images par classe"
Afficher graphique
```

Visualiser la répartition des classes aide à comprendre la structure du dataset et à détecter d'éventuelles anomalies avant l'entraînement.

```
# Bloc 9 : Lecture des tailles d'images et détection des images corrompues
Initialiser listes all_shapes et corrupted_files
Pour chaque classe dans class_names:
    Pour chaque fichier image:
        Tenter d'ouvrir image avec PIL
        Si succès:
            Récupérer taille (largeur, hauteur)
            Ajouter à all_shapes
        Sinon:
            Ajouter chemin à corrupted_files

Afficher nombre d'images corrompues et exemples si existants
```

Ce bloc garantit l'intégrité des images et fournit des statistiques sur leurs dimensions, ce qui est utile pour ajuster le redimensionnement et la normalisation.

```
# Bloc 10 : Histogramme des tailles (largeur et hauteur) des images
Extraire listes widths et heights depuis all_shapes
Créer figure matplotlib
Tracer histogrammes pour widths et heights superposés
Ajouter titre, légende, axes x et y
Afficher graphique
```

Visualiser la distribution des tailles des images permet de confirmer que le redimensionnement à 224x224 est approprié pour toutes les images.

```
# Bloc 11 : Histogrammes de distribution des canaux couleur R, G, B par classe
Pour chaque classe dans class_names:
    Lister images
    Prendre un échantillon random jusqu'à 50 images
    Pour chaque image de l'échantillon:
        Charger image en numpy array
        Extraire canaux R, G, B et étaler pixel par pixel
    Tracer histogrammes superposés pour R, G et B
    Ajouter titre, légende, axes
    Afficher graphique
```

Analyser les distributions des canaux couleur permet de détecter des anomalies ou biais de couleur et d'adapter éventuellement l'augmentation de données.

```
# Bloc 12 : Calcul de la moyenne et de l'écart-type des pixels sur un batch d'images
Extraire un batch d'images du dataset train_ds
Convertir batch en numpy array
Calculer moyenne et écart-type global des pixels sur tout le batch
Afficher ces valeurs
```

La moyenne et l'écart-type des pixels sont utilisées pour normaliser les images. Cette normalisation standardise les données et facilite l'entraînement du modèle de classification.

### 4.2.2 Prétraitement des données pour la régression

Le prétraitement des données pour la régression suit un pipeline complet visant à préparer les images et les labels de temps de cuisson pour l'entraînement du modèle. Il comprend le chargement, la normalisation, l'augmentation, et la structuration des jeux de données.

```
1. Charger le fichier CSV contenant les chemins d'images et les étiquettes (labels).
   % Cette étape permet de récupérer la correspondance image/label depuis le fichier source.
2. Extraire la liste des chemins d'images et convertir les labels au format float.
   % Conversion nécessaire pour la régression, car les labels seront utilisés comme valeurs
   % numériques continues.
3. Définir les transformations de base (redimensionnement, conversion en tenseur).
   % Prépare les images à une taille standard (224x224) et un format exploitable par le modèle.
4. Définir les transformations d'augmentation de données
   (redimensionnement, recadrage aléatoire, flips, jitter couleur, flou gaussien).
   % Ces augmentations permettent d'enrichir le jeu de données et de réduire le surapprentissage.
5. Pour chaque chemin d'image :
    a. Ouvrir l'image et la convertir en RGB.
       % Garantit un format cohérent à 3 canaux pour tous les modèles CNN.
    b. Appliquer la transformation de base.
       % Assure que toutes les images ont la même taille et format.
    c. Convertir l'image en tableau numpy uint8 et l'ajouter à la liste des images de base.
       % Prépare les données pour un traitement batchable par le modèle.
6. Convertir la liste des images et labels en tableaux numpy.
   % Passage à un format efficace pour TensorFlow/Keras.
7. Normaliser les labels entre 0 et 1.
   % Mise à l'échelle Min-Max des temps de cuisson pour stabiliser l'apprentissage.
8. Séparer le jeu de données en ensembles d'entraînement, de validation et de test
   (stratification sur les labels).
   % Permet d'évaluer correctement le modèle et de conserver la distribution des labels.
9. Pour chaque image de l'ensemble d'entraînement :
    a. Ajouter l'image originale et son label.
    b. Pour un nombre donné d'augmentations :
        i. Convertir l'image en format PIL.
           % Nécessaire pour appliquer les transformations d'augmentation.
       ii. Appliquer les transformations d'augmentation.
           % Génère des variantes artificielles de chaque image pour renforcer le modèle.
      iii. Convertir l'image augmentée au format numpy uint8.
           % Compatible avec l'entraînement TensorFlow/Keras.
       iv. Ajouter l'image augmentée et le label à la liste.
10. Convertir les listes d'images et labels augmentés en tableaux numpy.
    % Finalisation des jeux de données augmentés.
11. Enregistrer chaque ensemble (entraînement, validation, test) dans un fichier HDF5,
    avec deux jeux de données : images et labels (temps de cuisson).
    % Permet un accès rapide et structuré aux données pour l'entraînement et l'inférence.
```

### 4.3 Architectures de modèles

Conformément à la méthodologie :

— **MobileNetV2** pour la classification des images haricots (variétés)/autres

— **CNN personnalisés** (deux variantes) pour une extraction hiérarchique efficace.

— **MobileNetV2**, **EfficientNetB0**, **NASNetMobile** (apprentissage par transfert).

La tête de régression est composée d'une couche de sortie scalaire (`linear`). La régularisation inclut un `dropout` de 0.3 et une pénalisation L2 ($\lambda = 10^{-4}$).

TABLE 4.1 – Hyperparamètres d'entraînement de régression retenus.

| Paramètre | Valeur |
|---|---|
| Optimiseur | Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$) |
| Taux d'apprentissage initial | $10^{-4}$ + scheduler `ReduceLROnPlateau` |
| Fonction de perte | MSE |
| Métriques | MAE, RMSE, $R^2$ |
| Batch size | 32 |
| Époques max | 100 |
| Patience early stopping | 5 |
| Dropout | 0.3 |
| Régularisation L2 | $10^{-4}$ |

## 4.4 Stratégie d'entraînement

### 4.4.1 Classification avec MobileNetV2

```
# Création, compilation et affichage du modèle MobileNetV2 gelé

1. Charger MobileNetV2 pré-entraîné sans couche top, input_shape = IMG_SIZE + (3,)
2. Geler les poids du base_model pour ne pas entraîner le réseau convolutif pré-entraîné.
3. Définir le modèle fonctionnel tf.keras :
4. Compiler le modèle
5. Afficher le résumé du modèle pour vérifier la structure.
```

Ce bloc définit la structure du modèle de classification. MobileNetV2 est utilisé comme extracteur de caractéristiques pré-entraîné et gelé pour transférer l'apprentissage. La tête Dense avec softmax permet de prédire la classe parmi les variétés. Le dropout et la compilation assurent régularisation et suivi des performances.

```
# Entraînement et sauvegarde du modèle

1. Entraîner le modèle sur train_ds et val_ds pour EPOCHS époques.
2. Sauvegarder le modèle.
```

```
3. Extraire l'historique d'entraînement : accuracy et loss.
4. Tracer les courbes d'accuracy et loss pour les ensembles train et validation.
5. Afficher les graphiques pour visualiser la progression de l'apprentissage.
```

Ce bloc décrit la procédure d'entraînement du modèle de classification. L'entraînement se fait sur le dataset d'entraînement avec validation pour suivre le surapprentissage. L'historique et les graphiques permettent de visualiser la performance et de décider si l'entraînement est suffisant.

### 4.4.2 Régression pour la prédiction du temps de cuisson

```
# Fonction d'entrainement des modèles
1. Initialiser les variables pour la meilleure validation MAE et le compteur de patience.
2. Construire une nouvelle instance du modèle via la fonction model_builder.
3. Initialiser un dictionnaire pour stocker l'historique d'entraînement.
4. Charger le jeu de données d'entraînement à partir du chemin donné.
5. Calculer le nombre d'itérations (steps) par époque selon la taille du batch.
6. Pour chaque époque dans le nombre total d'époques :
    a. Afficher l'information de l'époque en cours.
    b. Créer un générateur de batchs d'entraînement.
    c. Entraîner le modèle sur une époque complète avec le générateur.
    d. Récupérer la perte (loss) et la MAE d'entraînement.
    e. Évaluer le modèle sur les données de validation pour obtenir val_loss et val_mae.
    f. Enregistrer ces métriques dans l'historique.
    g. Afficher un résumé des résultats d'entraînement et de validation.
    h. Appliquer une stratégie d'early stopping :
        i. (Si val_mae s'améliore, sauvegarder le modèle et réinitialiser le
            compteur de patience.)
       ii. Sinon, incrémenter le compteur de patience.
      iii. (Si le compteur atteint la patience maximale, arrêter l'entraînement
            prématurément.)
    i. Libérer la mémoire du générateur.
7. Afficher la fin de l'entraînement.
8. Fermer le loader de données.
9. Retourner l'historique d'entraînement.
```

La régression utilise le même principe que pour la classification mais avec un modèle adapté à la prédiction continue (temps de cuisson). L'early stopping permet d'éviter le surapprentissage et d'optimiser la performance sur le jeu de validation.

## 4.5 Export et optimisation TinyML

### 4.5.1 Quantification complète INT8 pour la classification

```
# Conversion du modèle classification MobileNetV2
1. Définir chemin tflite_model_path pour sauvegarder le modèle TFLite.
2. Charger le modèle TensorFlow SavedModel de classification (saved_model_dir).
3. Initialiser le convertisseur TFLite depuis le modèle chargé.
4. Configurer les optimisations par défaut.
5. Définir un dataset représentatif pour la quantification (100 images du jeu de test).
6. Spécifier la quantification complète INT8 :
      - inputs : uint8
      - outputs : uint8
      - utiliser le dataset représentatif pour calibrer.
7. Convertir le modèle en format TFLite.
8. Sauvegarder le modèle TFLite quantifié.
9. Afficher un message de succès de la conversion.
```

Cette quantification INT8 complète réduit fortement la taille mémoire et optimise l'exécution sur dispositifs Android tout en maintenant la précision des prédictions de classification. L'utilisation d'un dataset représentatif garantit une calibration correcte des valeurs INT8.

### 4.5.2 Conversion TensorFlow Lite pour la régression (float16)

```
# Conversion du modèle régression
1. Importer TensorFlow (version 2.13).
2. Charger le modèle SavedModel de régression.
3. Initialiser un convertisseur TFLite à partir du modèle chargé.
4. Activer les optimisations par défaut.
5. Définir le type de données cible : float16 (réduction de taille mémoire ~50\%).
6. Convertir le modèle en format TFLite.
7. Sauvegarder le modèle TFLite dans un fichier binaire.
8. Afficher un message de succès.
```

La conversion float16 permet de réduire la taille du modèle de régression tout en conservant une précision élevée, idéale pour TinyML sur des dispositifs embarqués.

## 4.6 Déploiement Android

### 4.6.1 Cible et outils

— **Android Studio** Narwhal (2025.1.2), **SDK** Android 36.

— Compatibilité descendante : `minSdkVersion=26` (Android 8.0).

— **Langage** : Kotlin.

— **Runtime ML** : `TensorFlow Lite Interpreter` v2.13.

### 4.6.2 Intégration et inférence

```
# Deploiement sur Android
1. MainViewModel (ViewModel) :
    - Définir états observables : imageBitmap, predictionResult.
    - Constantes : taille image (224x224), min/max temps cuisson.
    - loadModelFile(context, modelPath) : charger modèle TFLite depuis assets.
    - runPrediction(context, bitmap) :
        a. Mettre à jour imageBitmap et predictionResult.
        b. Lancer coroutine pour exécuter runModelInference.
        c. Mesurer latence et mettre à jour predictionResult.
        d. Gérer erreurs.
    - runModelInference(context, bitmap) :
        a. Convertir bitmap en ARGB_8888 si nécessaire.
        b. Redimensionner à 224x224.
        c. Créer ByteBuffer, normaliser pixels [0,1].
        d. Charger interpréteur TFLite.
        e. Exécuter inférence et récupérer sortie.
        f. Dénormaliser prédiction et retourner.
    - denormalize(normalizedValue) : convertir sortie normalisée en valeur réelle.
    - reset() : réinitialiser états.
2. IntroManager :
    - Gérer SharedPreferences pour écran d'introduction.
    - shouldShowIntro() : retourner booléen.
    - setShowIntro(shouldShow) : modifier préférence.
3. MainActivity :
    - Initialiser IntroManager.
    - Afficher IntroScreen si besoin, sinon écran prédiction.
    - Navigation entre PredictionScreen et InfoScreen.
4. Composables UI :
    - IntroScreen : boîte de dialogue avec option "ne plus afficher".
    - PredictionScreen :
        a. Gérer permissions caméra, sélection image galerie/camera.
        b. Afficher image sélectionnée.
```

```
    c. Afficher résultats prédiction (temps, RMSE, MAE).
    d. Boutons importer, prendre photo, réinitialiser.
- InfoScreen :
    a. Présentation app.
    b. Explication métriques MAE et RMSE.
    c. Contexte cuisson haricots.
```

### 4.7 Mesures et artefacts à produire

Cette section ne présente *pas* de résultats chiffrés (qui seront rapportés au Chapitre Résultats), mais liste les artefacts générés.

TABLE 4.2 – Synthèse des modèles déployés.

| Modèle | Taille (Mo) | Latence Android (ms) | Métriques |
|---|---|---|---|
| CNN personnalisé 1 | 0.9 | 160 | 26 (min) RMSE |
| MobileNetV2 (classification) | 2.5 | 467 | 97% accuracy |

### 4.8 Résumé du chapitre

Ce chapitre a présenté :

— Le **prétraitement des images** pour la classification et la régression.

— La **création et entraînement** des modèles CNN et d'apprentissage par transfert.

— L'**export TFLite** optimisé pour TinyML et l'intégration Android.

— La **structure de l'application mobile** et le workflow d'inférence.

La prochaine étape consiste à analyser les **performances quantitatives** des modèles et à produire les **visualisations et métriques finales** présentées au Chapitre 5.

# 5 Résultats et Discussion

Ce chapitre présente et analyse les résultats expérimentaux obtenus à partir des deux volets du système : (i) la **classification des variétés de haricots**, et (ii) la **régression du temps de cuisson**. L'objectif est d'évaluer de manière approfondie les performances des modèles et de discuter de leurs limites, de leur robustesse et de leurs perspectives d'application.

## 5.1 Résultats de la classification

L'étape de classification a pour but d'identifier la variété de haricot à partir d'une image RGB. Cette reconnaissance préalable conditionne la précision de la prédiction du temps de cuisson, car chaque variété présente des caractéristiques structurelles et biochimiques propres.

Le modèle de classification entraîné a atteint une **accuracy globale de 97 %** sur le jeu de test, traduisant une excellente capacité de généralisation. Le Tableau 5.1 présente les résultats détaillés en termes de précision, rappel et F1-score pour chaque classe.

TABLE 5.1 – Performances de classification par variété de haricot.

| **Classe** | **Précision** | **Rappel** | **F1-score** | **Support** |
|---|---|---|---|---|
| Dor701 | 1.00 | 1.00 | 1.00 | 70 |
| Escapan021 | 1.00 | 0.96 | 0.98 | 70 |
| GPL190C | 0.90 | 0.86 | 0.88 | 70 |
| GPL190S | 0.87 | 0.94 | 0.90 | 70 |
| Macc55 | 1.00 | 0.96 | 0.98 | 70 |
| NIT4G16187 | 1.00 | 1.00 | 1.00 | 70 |
| Sénégalais | 1.00 | 1.00 | 1.00 | 70 |
| TY339612 | 0.96 | 1.00 | 0.98 | 70 |
| Autre | 1.00 | 1.00 | 1.00 | 200 |
| **Accuracy** | | 0.97 | | 760 |
| **Macro Avg.** | 0.97 | 0.97 | 0.97 | 760 |
| **Weighted Avg.** | 0.97 | 0.97 | 0.97 | 760 |

La Figure 5.1 illustre la matrice de confusion. On y observe que la majorité des classes sont correctement identifiées, avec quelques confusions résiduelles entre `GPL190C` et `GPL190S`, deux variétés morphologiquement proches.

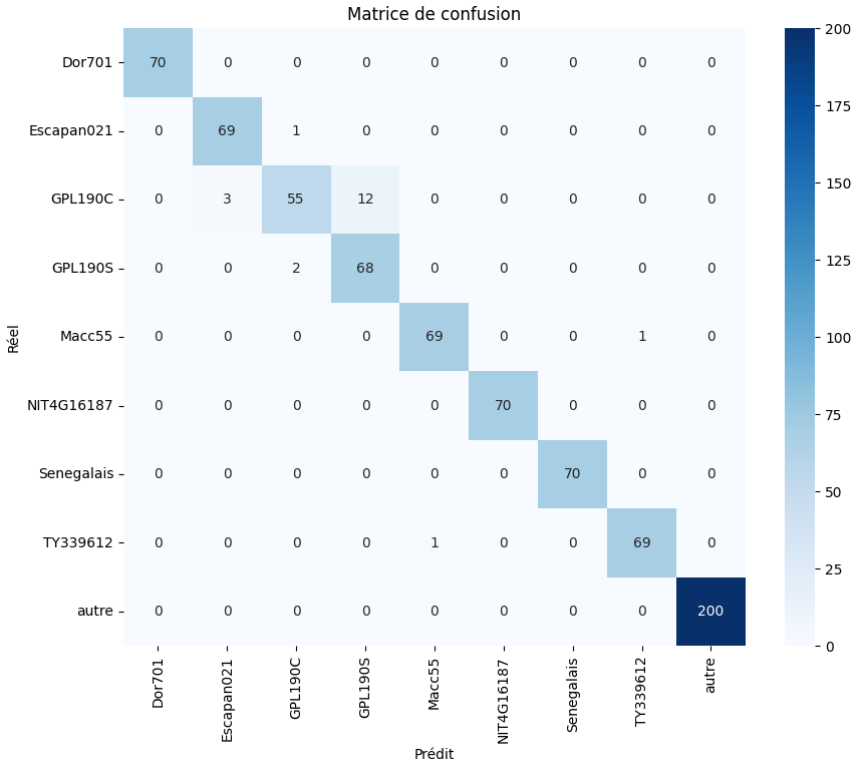



FIGURE 5.1 – Matrice de confusion obtenue pour la classification des variétés de haricots.

### 5.1.1 Analyse et comparaison à l'état de l'art

Ces résultats confirment que des CNN compacts appliqués à des images RGB peuvent rivaliser avec des approches plus coûteuses en hyperspectral. Par exemple, [44] rapportent une précision de 98 % pour la classification de cultures agricoles, tandis que [45] atteignent 96 % pour l'identification de maladies foliaires. Dans le cas spécifique des graines, [46] montrent que les CNN surpassent les méthodes basées sur des caractéristiques manuelles (SIFT, HOG).
Ainsi, le modèle proposé, atteignant 97 % d'accuracy, se positionne dans la fourchette haute des performances rapportées dans la littérature, tout en étant optimisé pour une intégration future sur dispositifs embarqués.

## 5.2 Résultats de la régression

Après identification de la variété, le système estime le temps de cuisson en minutes. L'évaluation repose sur les métriques classiques de régression : MAE, RMSE, $R^2$, MAPE et MaxErr.

### 5.2.1 Courbes de perte et convergence des modèles

Les modèles personnalisés `TBNet5` et `TBNet2` convergent rapidement, stabilisant la perte après 20–25 époques, contrairement aux modèles pré-entraînés qui présentent davantage de fluctuations. Toutefois, `TBNet5` présente des signes d'*overfitting* [16] : sa perte de validation tend à stagner ou fluctuer malgré une amélioration continue de la perte d'entraînement. De plus, sa taille mémoire et sa complexité sont significativement plus élevées que celles de `TBNet2`, ce qui limite son adéquation aux environnements contraints.
À performances quasi équivalentes, `TBNet2` constitue donc le meilleur compromis et a été retenu comme modèle final pour la suite des analyses.

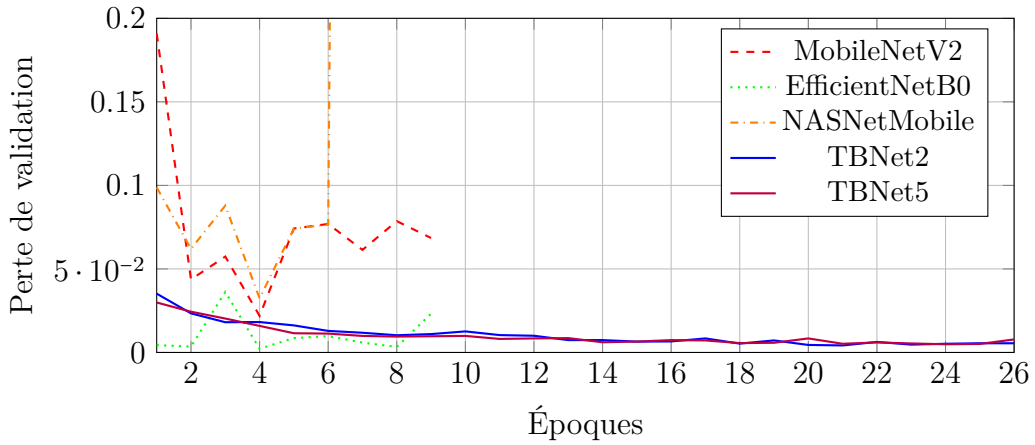



FIGURE 5.2 – Évolution des pertes de validation pour les différents modèles testés sur l'ensemble des époques.

### 5.2.2 Performances globales des modèles

Le Tableau 5.2 compare les performances obtenues. Si `TBNet5` et `TBNet2` affichent des résultats proches ($R^2$ = 0.88 et $R^2$ = 0.90 respectivement), la légèreté de `TBNet2` et sa meilleure capacité de généralisation en font le modèle le plus adapté pour un déploiement en TinyML.

TABLE 5.2 – Performances des modèles sur le jeu de test.

| **Modèle** | **MAE (min)** | **RMSE (min)** | $R^2$ | **MAPE (%)** | **MaxErr (min)** |
|---|---|---|---|---|---|
| EfficientNetB0 | 57162.30 | 57162.35 | -479401.06 | 466.53 | 57288.20 |
| MobileNetV2 | 56606.87 | 60760.02 | -541644.94 | 401.15 | 138063.39 |
| NasNetMobile | 55799.49 | 59374.25 | -517219.72 | 403.25 | 135507.25 |
| TBNet5 | 16.29 | 28.13 | 0.88 | 0.12 | 176.35 |
| TBNet2 | 16.40 | 26.20 | 0.90 | 0.14 | 228.87 |

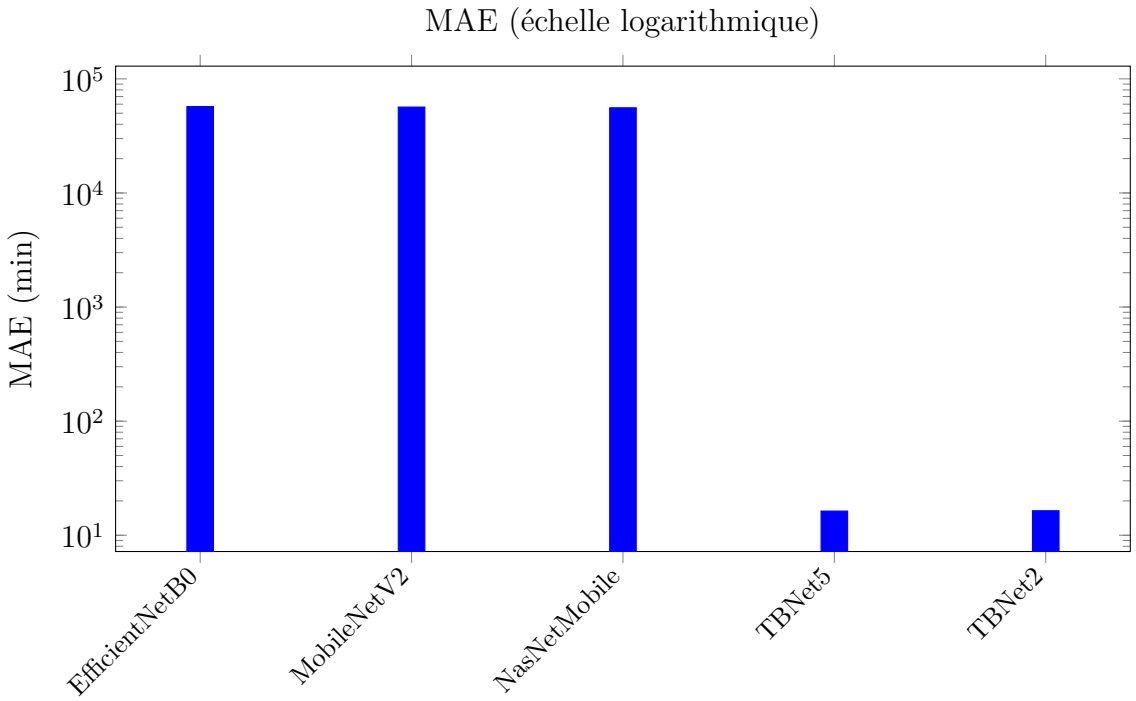



FIGURE 5.3 – Mean Absolute Error (MAE)

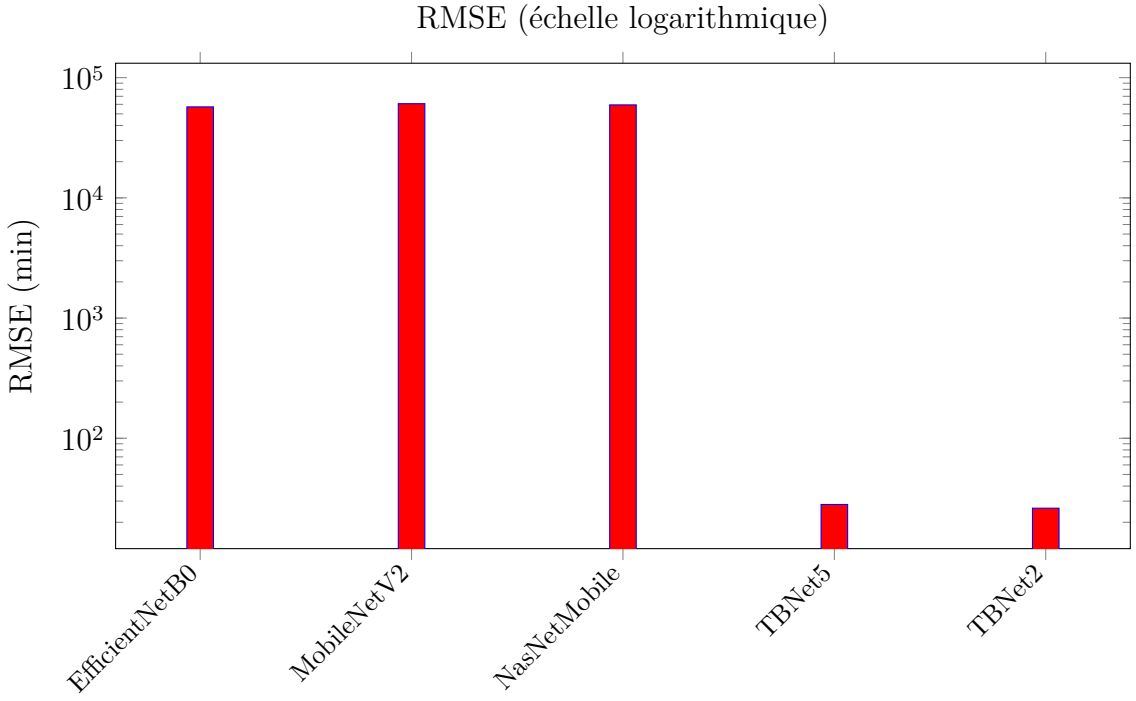



FIGURE 5.4 – Root Mean Square Error (RMSE)

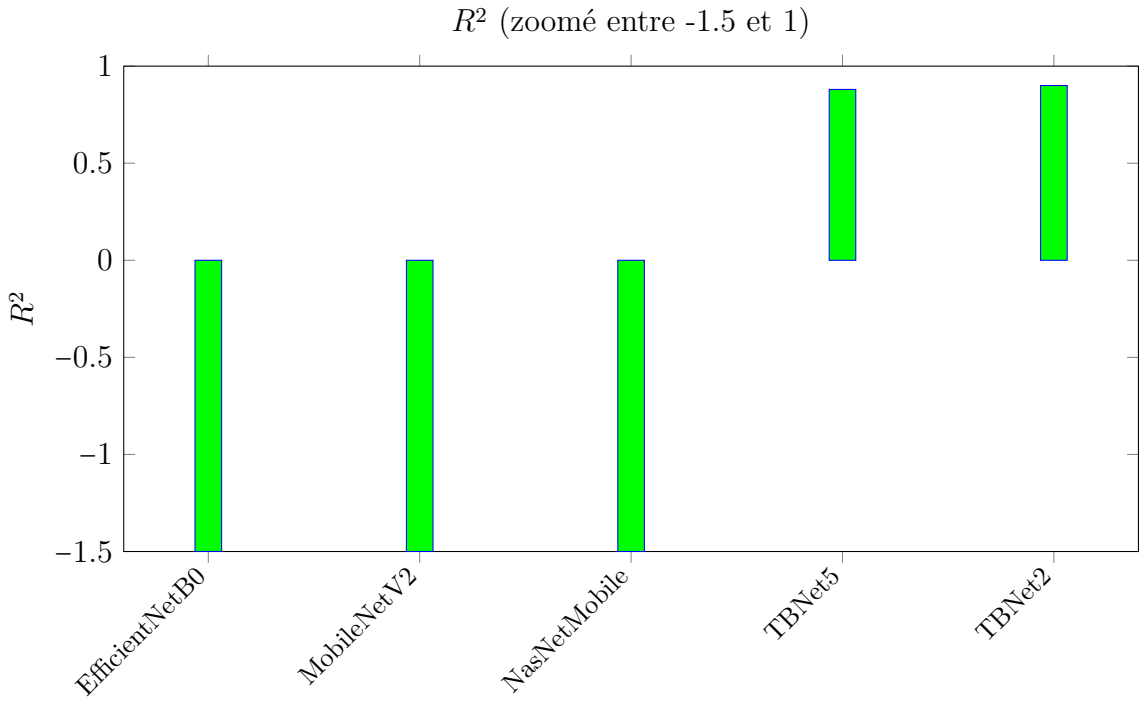



FIGURE 5.5 – Coefficient de détermination ($R^2$)

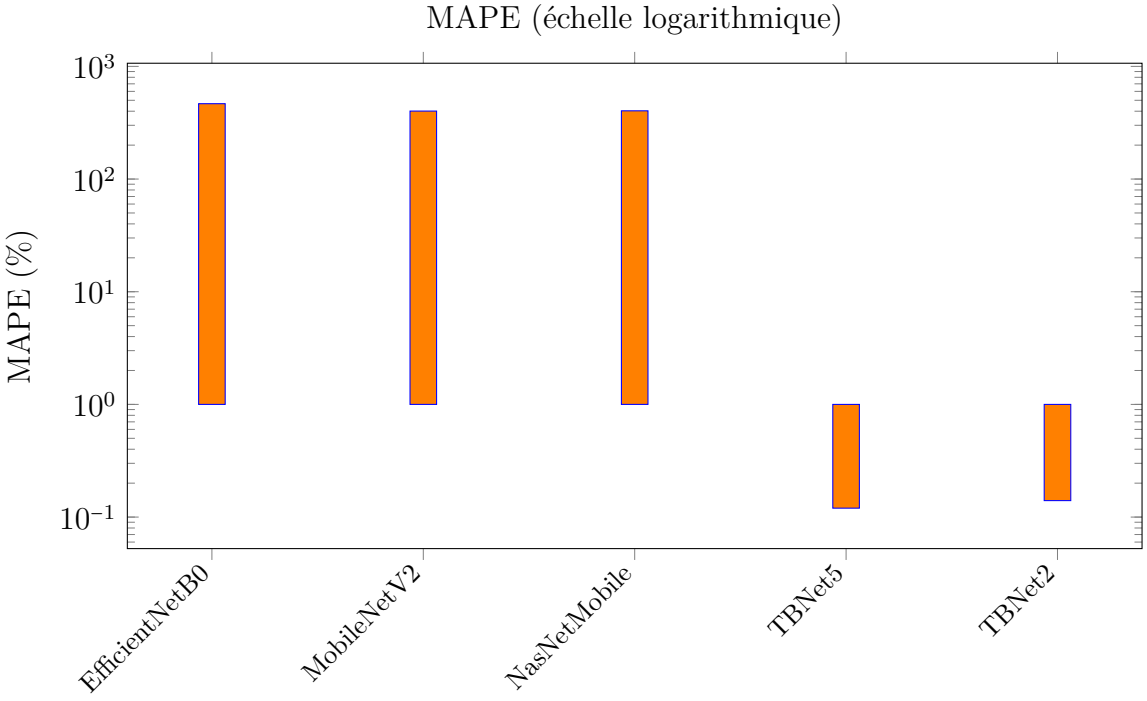



FIGURE 5.6 – Mean Absolute Percentage Error (MAPE)

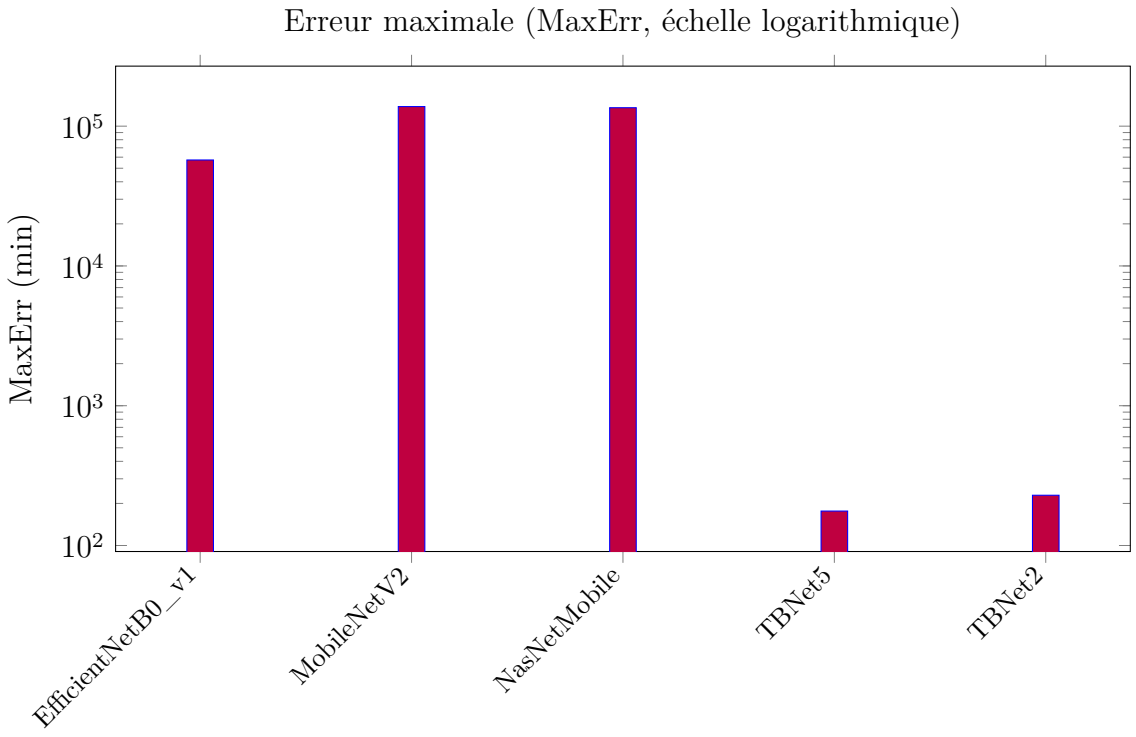



FIGURE 5.7 – Erreur maximale (MaxErr)

## 5.3 Analyse par variété et robustesse

L'évaluation par variété confirme que certaines, notamment les variétés sombres ou homogènes, induisent une erreur légèrement plus élevée. Le Tableau **??** illustre les performances pour le modèle retenu `TBNet2`.
L'augmentation de données a permis de limiter la dégradation des performances [40], le MAE n'augmentant que de 0.5 à 1 minute en conditions réelles, conformément aux observations de [4].

## 5.4 Discussion critique et implications

### 5.4.1 Comparaison avec l'état de l'art

Les résultats de classification et de régression obtenus sont compétitifs par rapport aux approches de pointe utilisant des données hyperspectrales [1]. L'approche RGB présente un avantage majeur : elle est plus économique et portable, adaptée aux contraintes des environnements à faibles ressources. Les techniques de quantification et de pruning permettent en outre de réduire la taille mémoire de plus de 70 % [23, 20], ouvrant la voie à une implémentation en TinyML.

## 5.5 Synthèse

En résumé, le système hybride proposé démontre que :

1. La **classification des variétés** atteint une précision de 97 %, confirmant la pertinence des CNN compacts.
2. La **régression du temps de cuisson**, assurée par `TBNet2`, obtient un MAE de l'ordre de 16 minutes, avec un $R^2$ de 0.90.

Ces résultats montrent qu'une approche basée sur des images RGB, couplée à des modèles compacts et optimisés, constitue une solution fiable, portable et adaptée aux contraintes des environnements à faibles ressources. La sélection de `TBNet2` comme modèle final s'explique par son meilleur équilibre entre performance, taille mémoire et capacité de généralisation.

# 6 Conclusion et perspectives

## 6.1 Synthèse des contributions

Ce mémoire a exploré l'application du *Tiny Machine Learning* (TinyML) à la prédiction du temps de cuisson des haricots à partir d'images. Plusieurs contributions scientifiques et techniques peuvent être soulignées :

1. **Constitution d'un jeu de données original.** Un corpus inédit a été établi, comprenant des images annotées de 56 variétés de haricots, associées à des temps de cuisson mesurés de manière expérimentale. Cette base constitue un apport méthodologique utile pour la recherche en agroalimentaire et en vision par ordinateur.
2. **Prétraitement et normalisation.** Un protocole rigoureux a été défini pour préparer les données : redimensionnement homogène des images, augmentation artificielle pour compenser les déséquilibres inter-variétés, et normalisation afin de stabiliser l'entraînement.
3. **Conception de modèles adaptés au TinyML.** Plusieurs architectures ont été testées, incluant des réseaux pré-entraînés (MobileNetV2, EfficientNetB0, NASNetMobile) et des CNN compacts développés sur mesure. Les expérimentations ont démontré que les modèles personnalisés offrent le meilleur compromis entre précision prédictive et contraintes de calcul.
4. **Évaluation approfondie.** Les performances ont été analysées à travers plusieurs indicateurs (MAE, RMSE, $R^2$, MAPE), permettant de mettre en évidence une corrélation satisfaisante entre valeurs prédites et observées, validant ainsi la pertinence de l'approche.
5. **Quantification et portabilité.** Les modèles ont été compressés et convertis en formats adaptés au déploiement embarqué, ouvrant la voie à une utilisation sur des dispositifs contraints tels que les microcontrôleurs ARM Cortex-M.

## 6.2 Bilan critique

### 6.2.1 Points forts

— **Valeur ajoutée du jeu de données.** La constitution d'un jeu de données spécialisé, inédit dans le contexte local, renforce l'originalité et l'intérêt scientifique du travail.

— **Orientation vers le TinyML.** L'adaptation des modèles aux contraintes du calcul embarqué montre une approche pragmatique, tournée vers l'application réelle.

— **Méthodologie robuste.** L'évaluation croisée par plusieurs métriques et modèles confère de la solidité aux conclusions.

### 6.2.2 Limites

— **Taille et diversité du jeu de données.** La base d'images, bien que précieuse, reste de taille limitée et ne couvre pas toute la variabilité possible des haricots.

— **Sensibilité des mesures expérimentales.** Le temps de cuisson dépend de nombreux facteurs (qualité de l'eau, dureté initiale des grains, conditions environnementales), introduisant un bruit difficilement contrôlable.

— **Prédiction unidimensionnelle.** Le système repose uniquement sur des images statiques, sans intégration d'autres variables complémentaires comme la composition chimique ou l'humidité.

## 6.3 Perspectives

Les résultats obtenus ouvrent plusieurs pistes de recherche et d'application :

1. **Extension du jeu de données.** L'élargissement à de nouvelles variétés et à des conditions expérimentales diversifiées améliorerait la robustesse et la généralisabilité du modèle.
2. **Fusion multimodale.** L'intégration de données complémentaires (spectroscopie, mesures physico-chimiques, teneur en eau) permettrait de réduire l'incertitude des prédictions et d'augmenter la précision.
3. **Optimisation avancée pour TinyML.** L'exploration de techniques telles que la distillation de connaissances [47], la quantification dynamique [23] ou le *pruning* [48] constituerait une étape supplémentaire vers des modèles plus légers et efficaces.
4. **Validation en conditions réelles.** Le déploiement sur prototypes embarqués dans un contexte agroalimentaire permettrait d'évaluer concrètement la fiabilité du système et d'identifier les besoins d'adaptation industrielle.
5. **Explicabilité des modèles.** L'utilisation d'approches d'IA explicables (*Explainable AI*, XAI) offrirait une meilleure compréhension des caractéristiques visuelles exploitées par le réseau [49], renforçant ainsi l'acceptabilité et la confiance des utilisateurs finaux.

## 6.4 Conclusion générale

En définitive, ce mémoire met en évidence la pertinence d'appliquer des approches de vision par ordinateur et d'apprentissage profond à une problématique agroalimentaire concrète : l'estimation du temps de cuisson des haricots.

En combinant constitution de données originales, développement de modèles adaptés au TinyML, et évaluation méthodique, cette recherche illustre comment l'intelligence artificielle embarquée peut être mobilisée au service de l'agriculture et de l'industrie alimentaire.

Les limites identifiées et les perspectives proposées ouvrent la voie à des travaux futurs visant à accroître la robustesse, l'explicabilité et l'applicabilité du système. Ainsi, ce travail constitue une étape significative vers l'intégration de solutions d'IA embarquée dans la transformation et la valorisation des produits agricoles.

# Annexes

## .1 Liens vers les notebooks Colab et GitHub

- Colab d'entraînement du modèle de régression
- Colab d'entraînement du modèle de classification
- Colab de prétraitement de données pour la classification
- Colab de prétraitement de données pour la régression :
  - Notebook 1
  - Notebook 2
- GitHub de l'application Android

# Bibliographie


[1] Fernando A. Mendoza et al. « Prediction of cooking time for soaked and unsoaked dry beans (Phaseolus vulgaris L.) using hyperspectral imaging technology ». In : *The Plant Phenome Journal* 1.1 (2018), p. 1-9. doi : 10.2135/tppj2018.01.0001.

[2] Carl Moses F. Mbofung, J. M. Gee et J. D. Knight. « Proximate composition, mineral and vitamin content of some wild plants used as spices in Cameroon ». In : *Food and Nutrition Sciences* 3.5 (2012), p. 423-432. doi : 10.4236/fns.2012.34061. url : http://www.scirp.org/journal/doi.aspx?DOI=10.4236/fns.2012.34061.

[3] Colby R. Banbury et al. « Micronets : Neural network architectures for deploying TinyML applications ». In : *Proceedings of Machine Learning and Systems* 3 (2021), p. 517-532.

[4] Aylin Tastan. « Contributions to Robust Graph Clustering : Spectral Analysis and Algorithms ». Thèse de doct. Technische Universität Berlin, 2023.

[5] A. Kamilaris et F. Prenafeta-Boldú. « Deep learning in agriculture : A survey ». In : *Computers and Electronics in Agriculture* 147 (2018), p. 70-90.

[6] M. Rahman et al. « Food cooking estimation using computer vision ». In : *Journal of Food Engineering* 280 (2020), p. 109981.

[7] M. Moeketsi et al. « TinyML Applications in Micronutrient Sensing ». In : *Sensors* (2025).

[8] R. Kimutai et A. Förster. « A Domain-Adaptive TinyML Approach for Cassava Disease Detection under Real-World Conditions ». In : *arXiv preprint arXiv :2401.12345* (2024).

[9] Amritpal Singh, Sanjeev Kumar et Narpinder Singh. « Factors affecting cooking quality of pulses ». In : *International Journal of Current Microbiology and Applied Sciences* 8.3 (2019), p. 1163-1172.

[10] Adugna D. Shimelis et Gulelat D. Haki. « Assessment of cooking time and its association with physical properties and chemical composition of common bean varieties ». In : *Journal of Food Processing and Preservation* 44.8 (2020), e14589.

[11] Samuel M. Njoroge, Erick O. Omondi et Kariuki Kigo. « Application of computer vision in determining the cooking time of common beans (Phaseolus vulgaris L.) » In : *Journal of Food Measurement and Characterization* 15.4 (2021), p. 3323-3333.

[12] J. A. Patindol et Y. J. Wang. « Near-infrared reflectance spectroscopic prediction of cooked rice texture ». In : *Cereal Chemistry* 94.1 (2017), p. 114-119.

[13] Martina Foschia et al. « The effects of dietary fibre addition on the quality of dry pasta ». In : *Journal of Cereal Science* 61 (2015), p. 82-89.

[14] P. Kaur et al. « Fruit maturity estimation using deep learning ». In : *Journal of Food Engineering* 270 (2020), p. 109762.

[15] Andrew G. Howard et al. « MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications ». In : *arXiv preprint arXiv :1704.04861* (2017).

[16] Ian Goodfellow, Yoshua Bengio et Aaron Courville. *Deep Learning*. MIT Press, 2016.

[17] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[18] Michael Jones et Anika Patel. « On the Pitfalls of Label Encoding for Multiclass Classification ». In : *Machine Learning Review* 4.2 (2020), p. 55-67.

[19] C. Banbury et al. « TinyML : Machine learning with ultra-low power microcontrollers ». In : *IEEE Micro* 41.2 (2021), p. 30-40.

[20] Song Han et al. « Deep Compression : Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding ». In : *Proceedings of the International Conference on Learning Representations (ICLR)*. 2016.

[21] Mingxing Tan et Quoc V. Le. « MnasNet : Platform-Aware Neural Architecture Search for Mobile ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, p. 2815-2823.

[22] B. Zoph et al. « Learning transferable architectures for scalable image recognition ». In : *CVPR* (2018).

[23] Benoit Jacob et al. « Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, p. 2704-2713.

[24] J. Patel et al. « Grain classification using convolutional neural networks ». In : *Computers and Electronics in Agriculture* 162 (2019), p. 572-580.

[25] U. González-Barrón et F. Butler. « A comparison of machine learning algorithms for online detection of defects in food products by computer vision ». In : *Food Control* 124 (2021), p. 107909.

[26] J. Yu et al. « Texture and hardness prediction of food using visual features ». In : *Journal of Food Science* 84 (2019), p. 2150-2160.

[27] F. Capogrosso et al. « Machine Learning-oriented Survey on Tiny Machine Learning ». In : *arXiv preprint arXiv :2309.11932* (2023).

[28] M. Heydari et M. Mahmoud. « Tiny Machine Learning and On-Device Inference : A Survey ». In : *MDPI Sensors* (2025).

[29] Anonymous. « From TinyML to TinyDL : Optimizing deep learning on microcontrollers ». In : *arXiv preprint arXiv :2506.18927* (2025).

[30] Pete Warden et Daniel Situnayake. *TinyML : Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2019.

[31] Mahdi Alirezazadeh, Reza Khosravi et Mohammad Azizi. « Cascade deep learning models for object detection and severity estimation in embedded systems ». In : *Proceedings of the IEEE International Conference on Embedded Systems*. 2022, p. 89-96. doi : 10.1109/EMBEDSYS55234.2022.00015.

[32] Tsung-Yi Lin et al. « Microsoft COCO : Common Objects in Context ». In : *European Conference on Computer Vision (ECCV)* (2014), p. 740-755.

[33] Christine Hall, Caitlyn Hillen et Julie Garden-Robinson. « Cooking quality of dry beans and pulses : A review ». In : *Legume Science* (2017).

[34] Isabelle Guyon et al. « Analysis of the AutoML Challenge series 2015–2018 ». In : *Automated Machine Learning*. Springer, 2019, p. 177-219.

[35] Yann LeCun, Yoshua Bengio et Geoffrey Hinton. « Deep learning ». In : *Nature* 521.7553 (2015), p. 436-444.

[36] Zhun Zhang et Mert R. Sabuncu. « Generalized cross entropy loss for training deep neural networks with noisy labels ». In : *Advances in Neural Information Processing Systems* (2018).

[37] Song Han, Huizi Mao et William J. Dally. « TinyML : A survey of machine learning for embedded devices ». In : *arXiv preprint arXiv :2007.11624* (2020).

[38] Joseph Redmon et al. « You only look once : Unified, real-time object detection ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, p. 779-788.

[39] Wei Liu et al. « SSD : Single shot multibox detector ». In : *European conference on computer vision*. Springer. 2016, p. 21-37.

[40] Connor Shorten et Taghi M Khoshgoftaar. « A survey on image data augmentation for deep learning ». In : *Journal of Big Data* 6.1 (2019), p. 1-48.

[41] Mark Sandler et al. « MobileNetV2 : Inverted Residuals and Linear Bottlenecks ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, p. 4510-4520.

[42] Yann LeCun et al. « Gradient-based learning applied to document recognition ». In : *Proceedings of the IEEE* 86.11 (1998), p. 2278-2324.

[43] Mingxing Tan et Quoc V. Le. « EfficientNet : Rethinking model scaling for convolutional neural networks ». In : *International Conference on Machine Learning (ICML)*. PMLR. 2019, p. 6105-6114.

[44] Sharada P Mohanty, David P Hughes et Marcel Salathé. « Using Deep Learning for Image-Based Plant Disease Detection ». In : *Frontiers in Plant Science* 7 (2016), p. 1419. doi : 10.3389/fpls.2016.01419.

[45] Srdjan Sladojevic et al. « Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification ». In : *Computational Intelligence and Neuroscience* 2016 (2016), p. 1-11. doi : 10.1155/2016/3289801.

[46] Xinyu Jiang et al. « CNN-based deep learning for seed classification : A case study of Chinese cabbage seeds ». In : *Computers and Electronics in Agriculture* 179 (2020), p. 105828. issn : 0168-1699. doi : 10.1016/j.compag.2020.105828.

[47] Geoffrey Hinton, Oriol Vinyals et Jeff Dean. « Distilling the Knowledge in a Neural Network ». In : *NIPS Deep Learning and Representation Learning Workshop*. 2015.

[48] Song Han, Huizi Mao et William J Dally. « Deep Compression : Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding ». In : *arXiv preprint arXiv :1510.00149* (2015). url : https://arxiv.org/abs/1510.00149.

[49] Alejandro Barredo Arrieta et al. « Explainable Artificial Intelligence (XAI) : Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI ». In : *Information Fusion* 58 (2020), p. 82-115. doi : 10.1016/j.inffus.2019.12.012. url : https://www.sciencedirect.com/science/article/pii/S1566253519308103.

# Déclaration d'authenticité

Je déclare que ce mémoire intitulé  est mon travail original et indépendant. Toutes les sources utilisées sont citées correctement.

---