



**UNIVERSITÉ DE  
NGAOUNDÉRE**  
FACULTÉ DES SCIENCES  
Département de  
Mathématiques  
et Informatique



**UNIVERSITY OF  
NGAOUNDÉRE**  
FACULTY OF SCIENCE  
Department of Mathematics  
and Computer Science

**ÉCOLE DOCTORALE : SCIENCES, TECHNOLOGIE ET  
INGÉNIERIE (STI)**

UFD : Mathématiques, Informatique, Ingénierie et Applications (M2IAP)

**MÉMOIRE**

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**

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

Sous la direction de :

**Pr. Dr.-Ing. JEAN LOUIS  
KEDIENG EBONGUE FENDJI**

Maître de Conférence

**Pr. Dr. Ing. BOUKAR  
OUSMAN**

Maître de Conférence

**Année académique : 2024–2025**

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

17 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 camarades 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)
FPU	Floating Point Unit (une unité du CPU qui effectue des calculs sur des nombres à virgule flottante)
MAE	Mean Absolute Error (erreur absolue moyenne), mesure de l'écart moyen entre les valeurs prédites et observées.
MAPE	Mean Absolute Percentage Error (erreur absolue moyenne en pourcentage)
MaxErr	Maximum Error (erreur maximale observée)
NASNetMobile	Neural Architecture Search Network Mobile
$R^2$	Coefficient de détermination, mesure de la proportion de variance expliquée par le modèle.
RMSE	Root Mean Square Error (racine de l'erreur quadratique moyenne), mesure de l'écart quadratique entre prédictions et observations
TBNet	<i>Tiny Bean (cooking time) Network</i> , architecture proposée pour la prédiction du temps de cuisson des haricots à partir d'images. Le suffixe <b>2</b> (TBNet2) correspond à une couche finale de 256 filtres, tandis que le suffixe <b>5</b> (TBNet5) correspond à 512 filtres
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

Les projections démographiques annoncent une croissance continue de la population mondiale, qui devrait atteindre près de 9,7 milliards d’habitants à l’horizon 2050 [1]. Cette évolution impose des défis sans précédent en matière de sécurité alimentaire, de gestion durable des ressources et de résilience des systèmes de production. Dans ce contexte, l’optimisation des procédés de transformation agroalimentaire apparaît comme un enjeu stratégique majeur. Elle conditionne non seulement la qualité et la sécurité des produits, mais aussi la maîtrise des coûts de production et la réduction des pertes post-récolte, dans un monde où la pression sur les ressources naturelles ne cesse de croître.

Parmi ces procédés, la cuisson des légumineuses — et plus particulièrement des haricots — occupe une place essentielle. Représentant une source majeure de protéines, de fibres et de micronutriments, les haricots jouent un rôle central dans l’équilibre nutritionnel et la souveraineté alimentaire de nombreuses régions du globe [2, 3]. Leur accessibilité économique en fait une denrée privilégiée pour répondre aux besoins nutritionnels des populations croissantes, notamment dans les pays en développement. Toutefois, le temps de cuisson des haricots constitue un facteur critique : il influence directement la texture, la saveur et la valeur nutritionnelle, tout en déterminant leur acceptabilité par les consommateurs.

Les méthodes conventionnelles d’estimation du temps de cuisson, basées sur des approches empiriques ou destructives (tests de cuisson, mesures d’absorption d’eau), bien que fiables, se révèlent chronophages, coûteuses et difficilement adaptables à une production industrielle en flux tendu. Elles entraînent également une consommation énergétique significative, peu compatible avec les impératifs actuels de durabilité. Ces contraintes affectent particulièrement les petites et moyennes unités de transformation, souvent dépourvues d’équipements sophistiqués et confrontées à des marges de manœuvre limitées.

L’émergence de l’intelligence artificielle (IA) et des technologies embarquées ouvre de nouvelles perspectives pour dépasser ces limitations. Le *Tiny Machine Learning* (TinyML), en particulier, permet de déployer des modèles légers directement sur des dispositifs à faibles ressources, tels que les microcontrôleurs ou les smartphones, sans nécessiter d’infrastructure lourde ni de connexion internet permanente [4]. Grâce à des architectures compactes comme *MobileNetV2* ou des réseaux convolutionnels sur mesure, il devient possible de traiter des données visuelles localement, avec un

coût énergétique réduit. Cette approche rend l’IA accessible dans des environnements contraints, notamment en zones rurales, et s’avère particulièrement pertinente pour des chaînes de valeur agroalimentaires où la flexibilité et l’efficacité sont essentielles [5].

Les travaux récents en vision par ordinateur appliquée à l’agriculture illustrent déjà ce potentiel : détection précoce de maladies, évaluation de la maturité des fruits ou suivi de la qualité de denrées périssables. Appliquer ces avancées à la prédiction du temps de cuisson des haricots s’inscrit donc dans une dynamique à la fois scientifique et socio-économique. Il s’agit non seulement d’améliorer la maîtrise des procédés industriels, mais aussi de contribuer à la sécurité alimentaire mondiale face aux défis démographiques à venir, en proposant des solutions accessibles, économes en énergie et adaptées aux réalités des producteurs.

## 1.2 Problématique

Malgré les avancées technologiques, la prédiction précise et non destructive du temps de cuisson des légumineuses demeure un défi. Les approches traditionnelles présentent plusieurs limites :

- **Temps et coût** : des méthodes chronophages, peu adaptées à un contrôle en temps réel ou à grande échelle.
- **Manque de portabilité** : des dispositifs lourds et onéreux, difficilement accessibles aux petites unités.
- **Contraintes matérielles et énergétiques** : peu de solutions compatibles avec les ressources limitées des environnements ruraux ou des micro-entreprises.

La question centrale à laquelle ce travail tente de répondre est donc 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

Développer et valider un modèle TinyML capable de prédire le temps de cuisson des haricots à partir d’images, de manière 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 par leur temps de cuisson.

2. Concevoir, entraîner et optimiser des modèles légers adaptés aux contraintes du TinyML (CNN compacts, MobileNetV2, EfficientNet-lite, etc.).
3. Évaluer les performances des modèles selon des critères 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 en contexte réel.

## 1.4 Organisation du mémoire

Le mémoire est structuré de manière à guider le lecteur du cadre conceptuel aux résultats pratiques :

- **Abstract** : synthèse du sujet, de la méthodologie et des résultats principaux.
- **Chapitre 1 — Introduction générale** : mise en contexte, problématique, objectifs et contributions.
- **Chapitre 2 — Revue de la littérature** : analyse critique des travaux antérieurs sur la prédiction du temps de cuisson, le TinyML et la vision par ordinateur appliquée à l'agroalimentaire.
- **Chapitre 3 — Méthodologie et conception** : description de la démarche expérimentale et des choix de conception.
- **Chapitre 4 — Implémentation et déploiement** : présentation des choix technologiques, de l'implémentation et du déploiement embarqué.
- **Chapitre 5 — Résultats et discussion** : analyse des performances obtenues et comparaison avec l'état de l'art.
- **Chapitre 6 — Conclusion et perspectives** : synthèse des contributions et identification de pistes 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 [6]. 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 [7].

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 [8, 9]. 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 [10, 11]. 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 [12]. 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, [13] ont étudié la texture du riz après cuisson, tandis que [14] 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 [15]. 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 *corrélacion croisée* (opération généralement utilisée dans les bibliothèques de deep learning) est définie par :

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

La *convolution* au sens strict correspond au noyau retourné :  $K'(u, v) = K(m-1-u, n-1-v)$  et  $S = I \star K'$ . 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 :

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

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 [16], 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 :

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

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

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

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

### 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 Normalization, ReLU, 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 [17, 18]. Elle transforme un vecteur de scores réels (appelés *logits*) en une distribution de probabilité sur les classes.

Soit un vecteur de sorties  $z = (z_1, z_2, \dots, z_K) \in \mathbb{R}^K$ , où  $K$  est le nombre de classes. La fonction softmax est définie par :

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



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 one-hot). Cette combinaison est particulièrement efficace pour l’optimisation de réseaux de neurones profonds [17].

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 :

$$\text{Dor701} \mapsto 0, \quad \text{TY339612} \mapsto 1, \quad \dots$$

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 [19]. 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) [20]. 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)  $\sim 32\text{--}1024\text{ kB}$  de SRAM, (ii)  $\sim 128\text{ kB}$  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 [8, 9].
- **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*  $z$  :

$$q = \text{clip}\left(\left\lfloor \frac{x}{s} + z \right\rfloor, q_{\min}, q_{\max}\right), \quad \hat{x} = s(q - z). \quad (2.6)$$

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  $\times 4$  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  $\times 2$  ; 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 [21].

**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 [16].
- **EfficientNet** : propose une mise à l’échelle équilibrée en profondeur, largeur et résolution pour maximiser l’efficacité [22].
- **NASNetMobile** : utilise la recherche automatique d’architectures (NAS) pour identifier les modèles optimaux pour mobiles [23].

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 [8].

## 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)** :

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

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

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

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

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

- **Erreur absolue moyenne (MAE) :**

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

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

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

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

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 :

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

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

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

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

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

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 multiclassées  $\mathbf{C}$  avec  $C$  classes, et en notant  $TP_c$ ,  $FP_c$ ,  $FN_c$ ,  $TN_c$  les valeurs par classe  $c$  :

- **Précision (Precision)** par classe :  $\text{Prec}_c = \frac{TP_c}{TP_c + FP_c}$  ;
- **Rappel (Recall)** :  $\text{Rec}_c = \frac{TP_c}{TP_c + FN_c}$  ;

- **F1** :  $F1_c = 2 \frac{\text{Prec}_c \times \text{Rec}_c}{\text{Prec}_c + \text{Rec}_c}$ .
- **Agrégations** : *macro* (moyenne non pondérée des classes), *micro* (calcul sur l'ensemble des instances), *pondérée* (pondération par support de chaque classe).

$$F1_{\text{macro}} = \frac{1}{C} \sum_{c=1}^C F1_c, \quad \text{Precision}_{\text{micro}} = \frac{\sum_c TP_c}{\sum_c (TP_c + FP_c)}.$$

- **Exactitude (Accuracy)** :  $\text{Acc} = \frac{\sum_c TP_c}{\sum_c (TP_c + FP_c + FN_c)}$  ; 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 §??).

## 2.9 Synthèse des travaux antérieurs

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

Réf.	Année	Titre	Objectif	Méthodologie	Résultats	Limites	Pertinence
[6]	2018	Deep learning in agriculture	Survol IA agriculture	Revue	IA utile classification	Peu de focus sur la cuisson	Contexte agroalimentaire
[7]	2020	Food cooking estimation	Estimer temps de cuisson via images	Analyse d'images	Corrélation détectée	Échantillon limité	Lien direct avec cuisson
[4]	2021	TinyML principes	Définir TinyML	Discussion	TinyML possible sur MCU	Manque d'implémentations concrètes	Cadre TinyML
[16]	2017	MobileNet CNN	CNN léger pour mobiles	Architecture	Bonne précision, faible coût	Coût élevé pour certains MCU	Base pour modèles légers
[22]	2019	EfficientNet scaling	Optimiser mise à l'échelle CNN	Architecture optimisée	Meilleure efficacité	Toujours lourd pour TinyML	Optimisation CNN
[23]	2018	NASNetMobile	Optimisation via NAS	Recherche automatique	Architecture optimisée	Complexité de la NAS	Modèles mobiles
[24]	2018	Quantization NN	Réduire taille modèles	Quantification INT8/FP16	Réduction mémoire x4	Perte de précision possible	Compression mémoire
[21]	2016	Pruning deep nets	Réduction des paramètres	Pruning	Réduction paramètres x10	Perte de précision si pruning excessif	Réduction mémoire
[25]	2019	Grain classification	Classifier variétés de grains	CNN grains	Bonne précision	Peu généralisable	Classification grains/haricots
[15]	2020	Fruit maturity estimation	Prédire maturité et tendreté	CNN fruits/légumes	Prédictions fiables	Dataset limité	Lien cuisson/texteure
[26]	2021	Food defect detection	Détection de défauts alimentaires	Analyse d'images	Détection robuste	Applicabilité restreinte	Qualité alimentaire
[27]	2019	Texture prediction food	Corréler images et texture	Vision + spectroscopie	Corrélations confirmées	Peu d'applications concrètes	Lien cuisson/texteure
[8]	2025	TinyML micronutrient sensing	TinyML pour détection micronutriments	Revue PRISMA	Modèles hybrides performants	Reporting incomplet	Insight TinyML en agriculture
[9]	2024	Domain-Adaptive TinyML	Détection maladies via TinyML	2D-CNN + adaptation domaine	Performance correcte	Chute de performance en conditions réelles	Adaptabilité TinyML agri
[28]	2023	Survey TinyML	Taxonomie TinyML	Revue PRISMA	Mapping optimisations	Peu d'applications agricoles	Ressources méthodologiques
[29]	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
[30]	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) [31, 32]. 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  $\rightarrow$  décision  $\rightarrow$  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

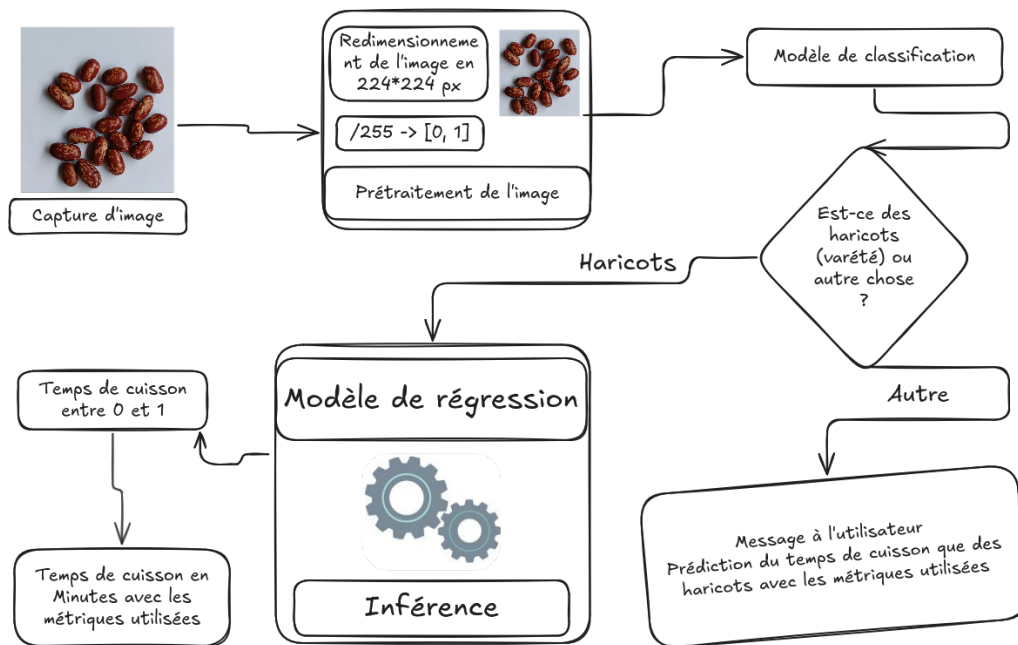


FIGURE 3.1 – Workflow fonctionnel de l'application Android : capture ou import d'image, prétraitement, classification -> inférence et affichage du temps de cuisson ( $T_c$ ) si haricots.

#### 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)* [33]. 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  $224 \times 224$  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-varietés de haricots du fait de similarités visuelles ; distribution hétérogène entre classes spécifiques et la classe "autre".

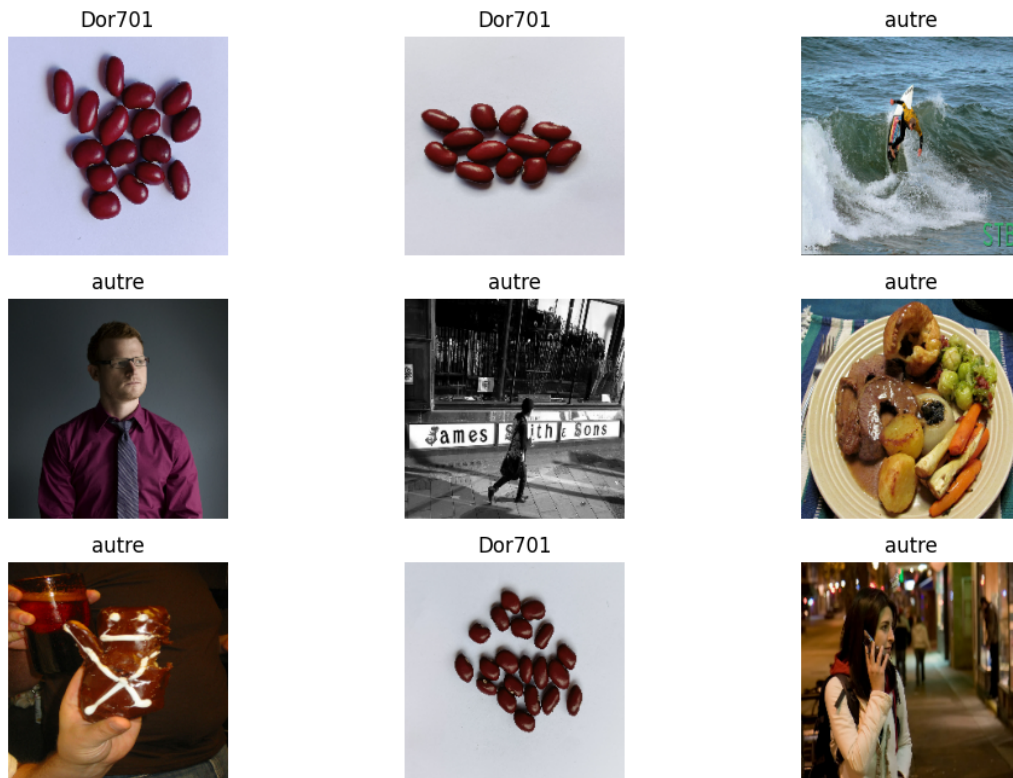


FIGURE 3.2 – Dataset de Classification [Extrait]

### 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 ??.
- **Annotation** : chaque image est associée à une mesure expérimentale du temps de cuisson  $T_c$ , déterminée selon le critère culinaire *fork-tender* [34]. Une incertitude expérimentale estimée à  $\pm 1$  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 [35].
- **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  $\times 5$  a ainsi été obtenu.
- **Prétraitement** : toutes les images ont été redimensionnées en  $224 \times 224$  pixels. Contrairement aux pratiques standards de normalisation des intensités dans  $[0, 1]$  [36], 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 :

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

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 [18].

- **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.

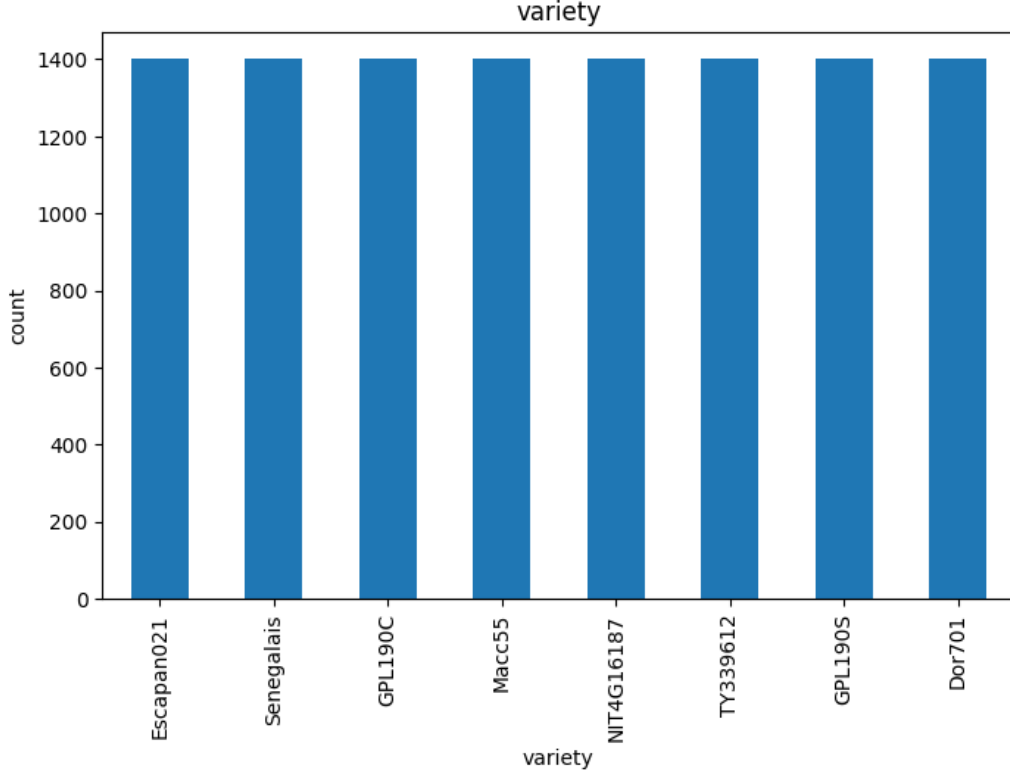


FIGURE 3.3 – Distribution du Dataset de régression par variété

### 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 [37]. 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 [17]. Ces différences impliquent des prétraitements distincts, des augmentations adaptées, et des stratégies de normalisation ciblées

pour stabiliser l’entraînement [36]. 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) [38]. 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 [39, 40].

### 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 ?? 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

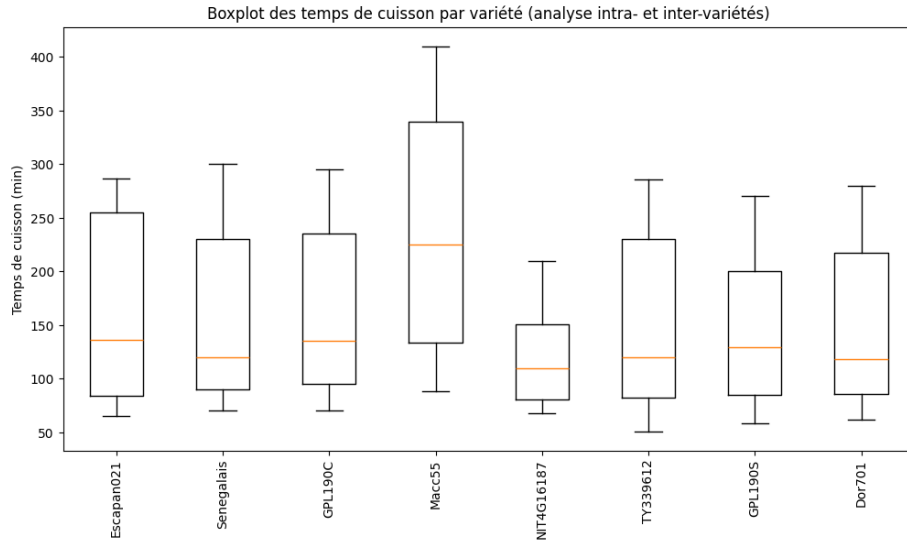


FIGURE 3.4 – Distribution des temps de cuisson par variété : le boxplot met en évidence les médianes, les quartiles et les valeurs extrêmes.

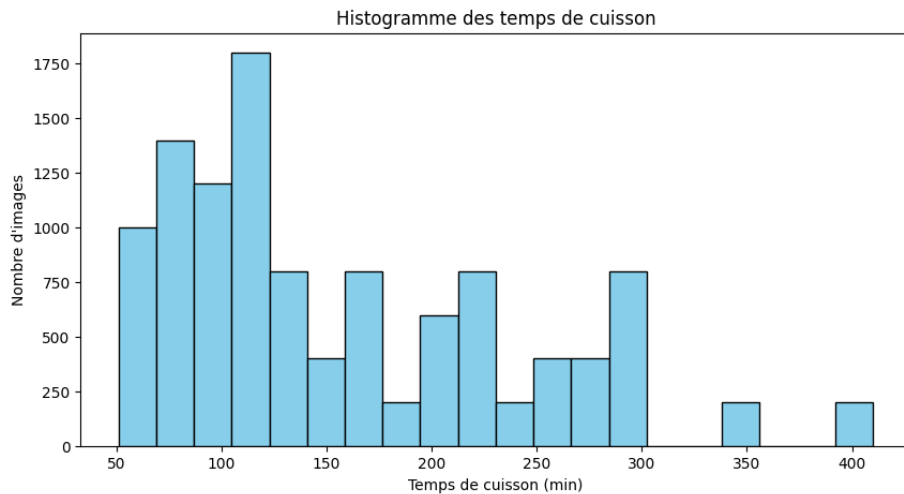


FIGURE 3.5 – Histogramme global des temps de cuisson ( $T_c$ ) toutes variétés confondues. La distribution multimodale reflète l'hétérogénéité du dataset.

## 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 [33]. 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 ??, justifiant l’usage futur d’augmentations photométriques pour améliorer la robustesse du modèle [41].

### 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, NAS-NetMobile). 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 [42].
- **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

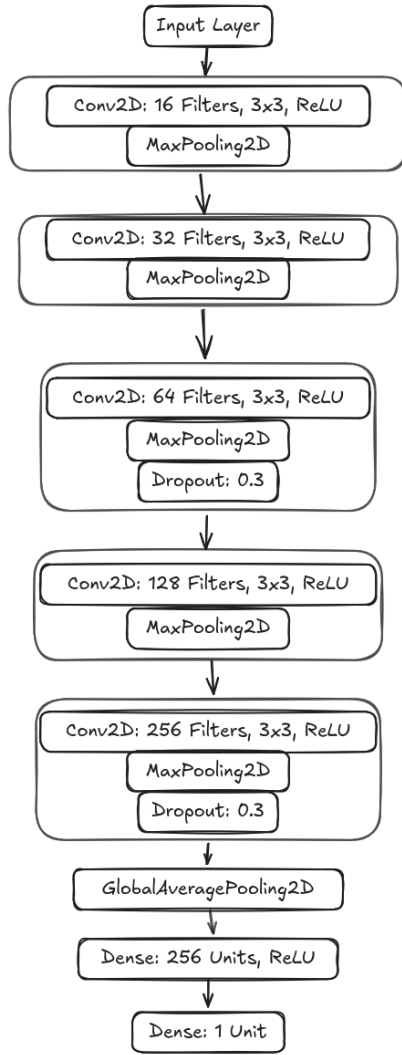
- **Backbones testés** : CNN personnalisés (TBN2 / TBN5) et modèles pré-entraînés (EfficientNet-B0, MobileNetV2, NASNetMobile). Le modèle final retenu dans les expériences précédentes a été un CNN personnalisé entraîné *from scratch*, plus simple et mieux adapté au jeu de données disponible.
- **Sortie** : 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

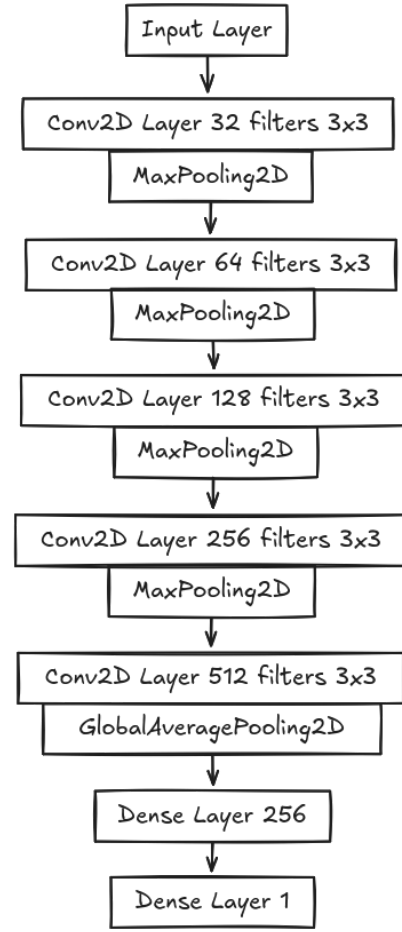
Plusieurs architectures ont été étudiées afin de comparer des modèles *from scratch* (TBN2, TBN5) et des modèles *pré-entraînés* (EfficientNet-B0, MobileNetV2, NASNetMobile). Le choix des backbones a été motivé par un compromis entre performance, capacité de généralisation et contraintes liées au déploiement sur microcontrôleurs (TinyML).

**TBN2** TBN2 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 [43].

**TBN5** TBN5 prolonge la logique de TBN2 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 [17].



(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 [44] 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.

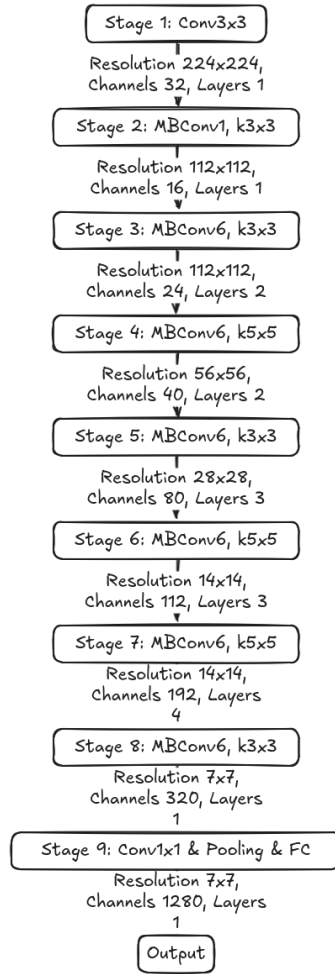


FIGURE 3.7 – Schéma simplifié de l'architecture EfficientNet-B0 (source : [44]).

**MobileNetV2** MobileNetV2 [42] 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.

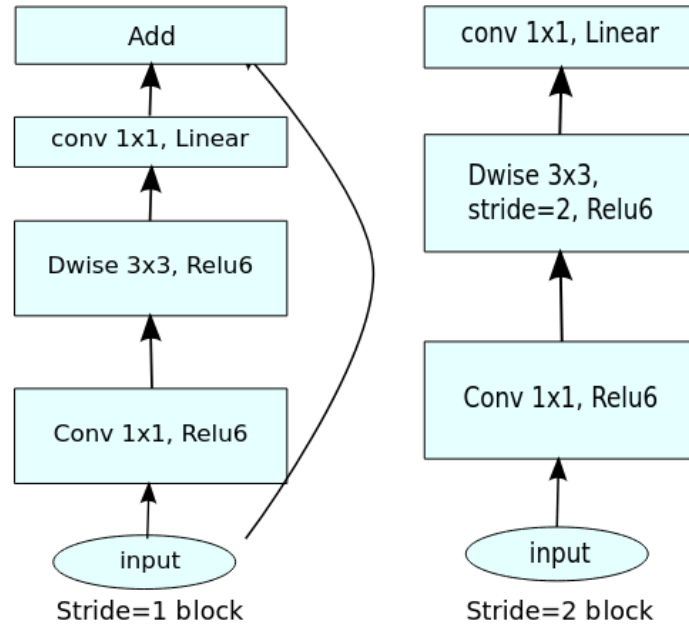


FIGURE 3.8 – Architecture MobileNetV2 utilisée comme extracteur de caractéristiques.

**NASNetMobile** NASNet [23] 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.

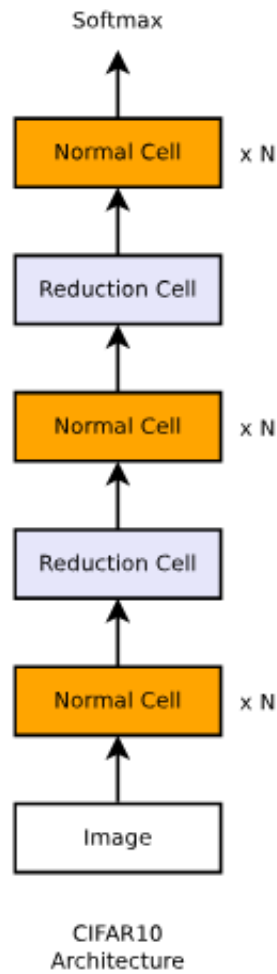


FIGURE 3.9 – Schéma de l’architecture NASNetMobile (source : [23]).

### 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.  $\pm$  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 confiance 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 (tolérance 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 ?? 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 ?? 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’entraînement 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.

---

**Algorithm 1** Création des dossiers pour les données d'entraînement et de test

---

Définir `base_path` et chemins `train_path`, `test_path` Créer dossiers `train_path` et `test_path` s'ils n'existent pas

---

---

**Algorithm 2** Téléchargement et extraction du dataset COCO val2017

---

Vérifier et télécharger `val2017.zip` si nécessaire Extraire le contenu si dossier absent  
Afficher message de disponibilité du dataset

---

---

**Algorithm 3** Sélection et copie d'un sous-ensemble d'images dans les dossiers

---

Lister images `.jpg`, prendre subset de 1000 images Copier 800 images dans `train_path`, 200 dans `test_path` Afficher confirmation

---

---

**Algorithm 4** Chargement des chemins d'images et des labels de variétés

---

Collecter chemins et labels depuis dossiers de classes Indexer labels en tableau numérique Afficher résumé des données chargées

---

---

**Algorithm 5** Création d'un dataset TensorFlow optimisé

---

Définir fonction `load_image` pour lecture, redimensionnement, normalisation Créer dataset `tf.data` avec mise en batch, mélange et préchargement

---

---

**Algorithm 6** Chargement automatique des images avec Keras

---

Utiliser `image_dataset_from_directory` sur `train_dir` Paramétrer taille images, batch size et mélange Afficher classes détectées

---

---

**Algorithm 7** Comptage et visualisation de la distribution des images par classe

---

Définir fonction de comptage des images par classe Calculer et afficher la distribution (graphique histogramme)

---

---

**Algorithm 8** Validation des images : intégrité et caractéristiques

---

Lister tailles images valides, détecter images corrompues Tracer histogrammes de dimensions et distributions colorimétriques Calculer statistiques pixel (moyenne et écart-type) sur un batch

---

**Description :** Ce processus couvre la préparation méthodique d'un dataset d'images pour classification, incluant gestion des dossiers, récupération des données, constitution des étiquettes, construction optimisée de pipelines TensorFlow, ainsi que analyses exploratoires de la distribution et qualité des images. Ces étapes assurent un cadre robuste et reproductible pour l'entraînement des modèles de machine learning. 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.

---

**Algorithm 9** Préparation et augmentation du jeu de données d'images

---

éeFichier CSV avec chemins d'images et labels Jeux de données entraînement, validation et test augmentés et sauvegardés  
 Charger CSV des chemins d'images et labels Extraire chemins d'images et convertir labels en float Définir transformations de base (redimensionnement, conversion tenseur) Définir transformations d'augmentation (recadrage, flips, jitter, flou)  
**foreach** *chemin d'image* **do**  
 Ouvrir image, convertir en RGB Appliquer transformation de base Convertir en tableau numpy uint8 Ajouter à liste images de base  
 Convertir listes images et labels en tableaux numpy Normaliser labels (mise à l'échelle Min-Max) Séparer données en ensembles entraînement, validation, test avec stratification  
**foreach** *image dans ensemble entraînement* **do**  
 Ajouter image originale et label **for** *nombre d'augmentations* **do**  
 Convertir image en PIL Appliquer transformations d'augmentation Convertir en numpy uint8 Ajouter image augmentée et label  
 Convertir listes augmentées en tableaux numpy Enregistrer ensembles (entraînement, validation, test) dans fichiers HDF5 avec images et labels

---

**Description :** Cet algorithme présente la préparation complète des données d'images pour un projet d'apprentissage profond incluant le chargement des chemins et labels, les transformations et augmentations d'images, la normalisation et la séparation du dataset, ainsi que la sauvegarde finale au format optimisé pour l'entraînement. Cette procédure garantit une base de données enrichie et stable pour des modèles robustes.

### 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 <b>ReduceLROnPlateau</b>
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**

---

**Algorithm 10** Entraînement et sauvegarde du modèle de classification

---

Modèle compilé, ensembles **train\_ds** et **val\_ds**, nombre d’époques **EPOCHS** Modèle entraîné sauvegardé, historique d’entraînement

Entraîner le modèle sur **train\_ds** et **val\_ds** pour **EPOCHS** époques Sauvegarder le modèle entraîné Extraire historique d’entraînement (accuracy, loss) Tracer et afficher courbes d’accuracy et loss **return** *historique*

---

---

**Algorithm 11** Fonction d'entraînement pour modèles de régression avec early stopping

---

Chemin jeu de données, nombre d'époques *EPOCHS*, taille batch, fonction *model\_builder*, patience Historique d'entraînement  
Initialiser meilleure validation MAE et compteur patience Construire modèle via *model\_builder* Initialiser dictionnaire historique Charger jeu de données d'entraînement et calculer steps par époque  
**for** *epoch*  $\leftarrow$  1 **to** *EPOCHS* **do**  
Afficher numéro d'époque Créer générateur de batches Entraîner modèle sur une époque complète Récupérer loss et MAE d'entraînement Évaluer sur validation : *val\_loss* et *val\_mae* Enregistrer métriques dans historique Afficher résumé résultats  
**if** *val\_mae* s'améliore **then**  
Sauvegarder modèle Réinitialiser compteur patience  
**else**  
Incrémenter compteur patience **if** *compteur atteint patience maximale* **then**  
Arrêter entraînement prématurément  
Libérer mémoire générateur  
Afficher fin d'entraînement Fermer loader de données **return** *historique*

---

**Description :** Ces algorithmes détaillent les procédures d'entraînement supervisé pour les modèles de classification et régression. Le suivi des métriques pendant l'entraînement, ainsi que l'utilisation d'un mécanisme d'early stopping pour la régression, permettent d'améliorer la généralisation en évitant le surapprentissage. La sauvegarde périodique du modèle assure la conservation des meilleurs poids selon la performance de validation.

## Export et optimisation TinyML

### Quantification complète INT8 pour la classification

---

**Algorithm 12** Conversion du modèle classification MobileNetV2 en TFLite quantifié INT8

---

Modèle TensorFlow SavedModel, chemin de sauvegarde TFLite Modèle TFLite quantifié INT8 sauvegardé  
Définir chemin de sauvegarde TFLite Charger modèle TensorFlow SavedModel Initialiser convertisseur TFLite Activer optimisations par défaut Définir dataset représentatif (exemple : 100 images du jeu de test) Configurer quantification complète INT8 :  
- Inputs : uint8  
- Outputs : uint8  
- Calibration avec dataset représentatif Convertir modèle en TFLite Sauvegarder modèle TFLite quantifié Afficher message de succès de la conversion

---

**Description :** Cette conversion INT8 optimise la taille mémoire et la vitesse d'exécution sur dispositifs embarqués tout en conservant une bonne précision. La calibration avec un dataset représentatif est cruciale pour assurer une quantification efficace et fidèle du modèle.

## Conversion TensorFlow Lite pour la régression (float16)

---

### Algorithm 13 Conversion du modèle régression en TFLite float16

---

Modèle TensorFlow SavedModel, chemin de sauvegarde TFLite Modèle TFLite float16 sauvegardé

Importer TensorFlow version 2.13 Charger modèle SavedModel de régression Initialiser convertisseur TFLite Activer optimisations par défaut Définir type de données cible : float16 (réduction mémoire d'environ 50%) Convertir modèle en format TFLite Sauvegarder fichier binaire TFLite Afficher message de succès

---

**Description :** Cette conversion en float16 permet de diminuer significativement la taille du modèle de régression tout en préservant une précision élevée, ce qui est particulièrement adapté pour TinyML sur dispositifs embarqués à ressources limitées.

## 4.5 Déploiement Android

### 4.5.1 Cible et outils

Le déploiement a été effectué sur un **Samsung Galaxy Note 9** (Android 10, 6 Go RAM, 128 Go stockage) via une application développée avec :

- **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.

### 4.5.2 Architecture logicielle de l'application

L'application mobile repose sur une architecture modulaire :

- **Interface utilisateur (UI)** : importation des photos de haricots et affichage des résultats.
- **Module de classification** : MobileNetV2 quantifié INT8, détermine si l'image contient des haricots.
- **Module de régression** : CNN personnalisé (TFLite float16), estime le temps de cuisson.

- **Gestionnaire d’inférence** : orchestre les prétraitements, les appels aux modèles et la dénormalisation.
- **Module résultats** : restitution de la variété détectée, du temps de cuisson estimé, de la latence et des métriques (MAE, RMSE, accuracy).

### 4.5.3 Intégration et inférence

---

**Algorithm 14** Pipeline de prédiction du temps de cuisson des haricots

---

Image de haricot sous forme de bitmap Temps de cuisson estimé et variété détectée

**Fonction** `RunPrediction(context, bitmap)` :

```

Mettre à jour l'image affichée avec bitmap Afficher "Analyse en cours..." Convertir
bitmap en format ARGB_8888 si nécessaire // Étape 1 : Classification
1(classePrédite, score) ← runClassification(context, bitmap) if classePrédite =
"autre" then
2Afficher message d'erreur indiquant image non reconnue comme haricot Afficher
métriques classification (Précision, Recall, F1-score)
3else
// Étape 2 : Régression
4Démarrer chronomètre temps ← runRegression(context, bitmap) Arrêter chrono-
mètre Afficher variété détectée : classePrédite Afficher temps de cuisson estimé :
temps minutes Afficher latence de prédiction Afficher métriques régression (RMSE,
MAE) et classification
5return

```

**Fonction** `runClassification(context, bitmap)` :

```

Redimensionner bitmap en 224 × 224 Convertir pixels en ByteBuffer UINT8 (sans
normalisation) Charger modèle TFLite quantifié int8 "bean_classifier.tflite" Exé-
cuter inférence pour obtenir sortie UINT8 Trouver indice de la classe ayant le score
maximum return (classe correspondante, indice)

```

**Fonction** `runRegression(context, bitmap)` :

```

Redimensionner bitmap en 224 × 224 Convertir pixels en tableau float normalisé
en [0,1] Charger modèle TFLite float16 "bean_regressor.tflite" Exécuter inférence
pour obtenir sortie float32 Dénormaliser la sortie pour obtenir temps cuisson :

```

$$\text{temps} = \text{valeurNormalise} \times (410 - 51) + 51$$

```

return temps

```

**Fonction** `Réinitialiser` :

```

Mettre à null imageBitmap et predictionResult

```

---

### 4.6 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

## 4.7 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 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 structurales 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 ?? 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
<b>Accuracy</b>	0.97			760
<b>Macro Avg.</b>	0.97	0.97	0.97	760
<b>Weighted Avg.</b>	0.97	0.97	0.97	760

La Figure ?? 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.



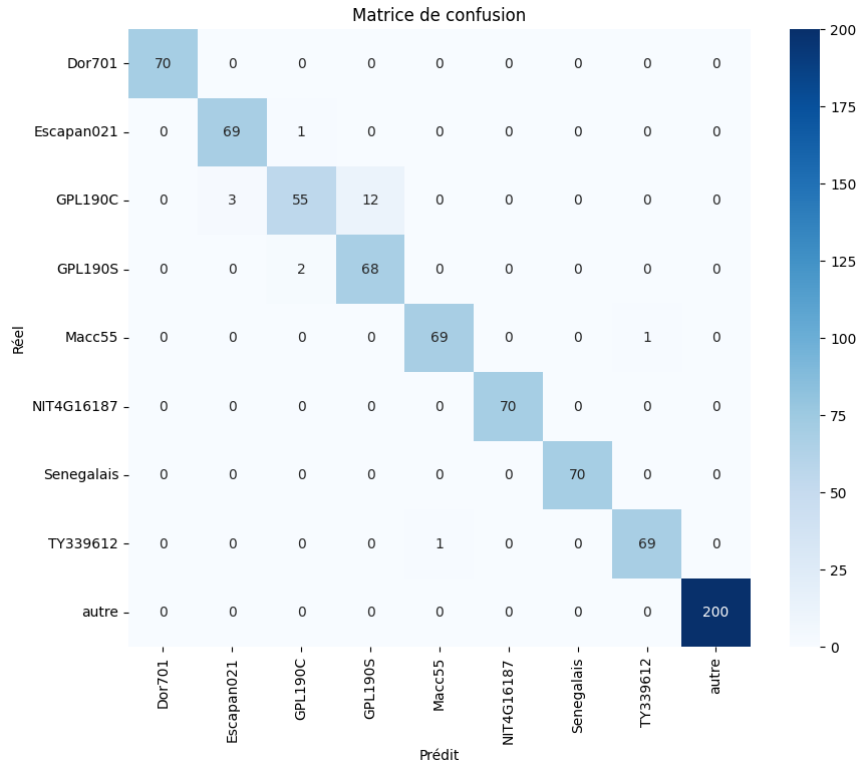


FIGURE 5.1 – Matrice de confusion obtenue pour la classification des variétés de haricots.

### 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, [45] rapportent une précision de 98 % pour la classification de cultures agricoles, tandis que [46] atteignent 96 % pour l'identification de maladies foliaires. Dans le cas spécifique des graines, [47] 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 TBN<sub>et</sub>5 et TBN<sub>et</sub>2 convergent rapidement, stabilisant la perte après 20–25 époques, contrairement aux modèles pré-entraînés qui présentent davantage de fluctuations. Toutefois, TBN<sub>et</sub>5 présente des signes d'*overfitting* [17] : 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 TBN<sub>et</sub>2, ce qui limite son adéquation aux environnements contraints.

À performances quasi équivalentes, TBN<sub>et</sub>2 constitue donc le meilleur compromis et a été retenu comme modèle final pour la suite des analyses.

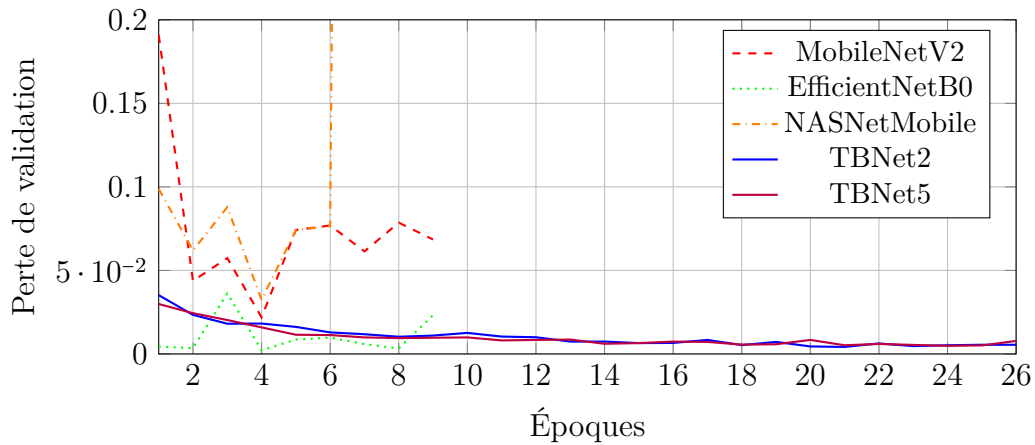


FIGURE 5.2 – Évolution des pertes de validation pour les différents modèles testés sur l'ensemble des époques.

### 5.2.2 Performances globales des modèles

Le Tableau ?? compare les performances obtenues. Si TBN<sub>et</sub>5 et TBN<sub>et</sub>2 affichent des résultats proches ( $R^2 = 0.88$  et  $R^2 = 0.90$  respectivement), la légèreté de TBN<sub>et</sub>2 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	RMSE	$R^2$	MAPE	MaxErr	Taille (Av.Q)
EfficientNetB0	57162.30	57162.35	-479401.06	466.53	57288.20	50
MobileNetV2	56606.87	60760.02	-541644.94	401.15	138063.39	31
NasNetMobile	55799.49	59374.25	-517219.72	403.25	135507.25	55
TBN <sub>et</sub> 5	16.29	28.13	0.88	0.12	176.35	5.6
TBN <sub>et</sub> 2	16.40	26.20	0.90	0.14	228.87	20.5

### Unité des métriques

- MAE, RMSE, MaxErr en minutes
- **Taille (Av.Q)** en Mo
- **MAPE** en pourcentage

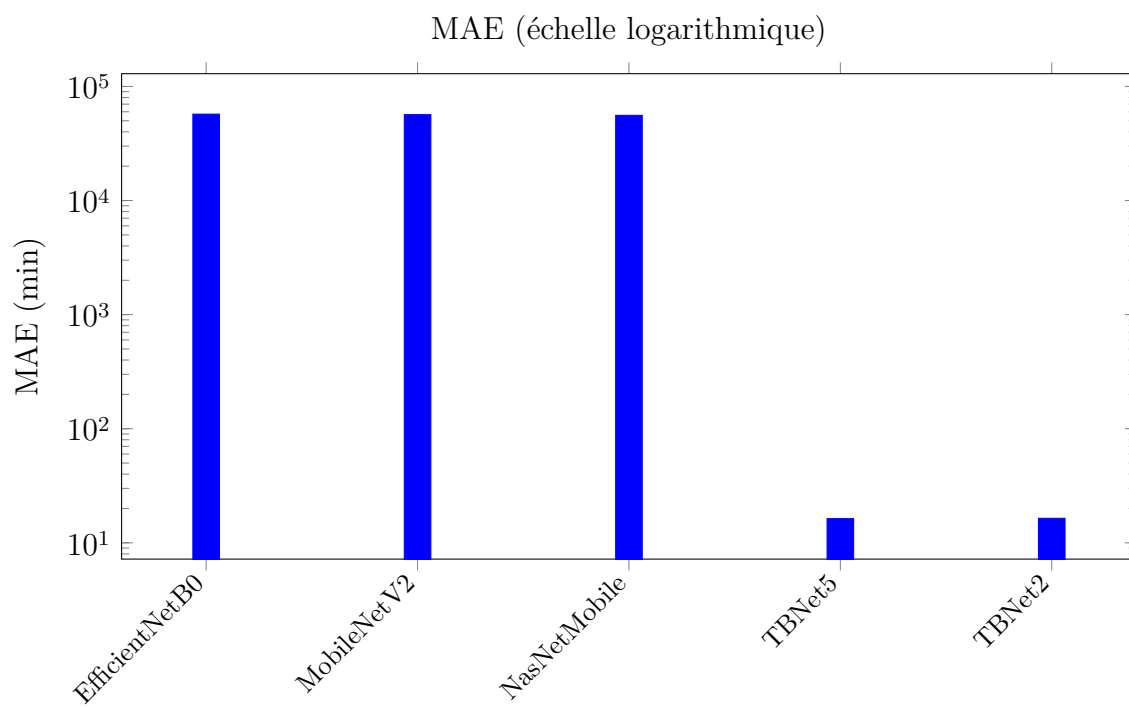


FIGURE 5.3 – Mean Absolute Error (MAE)

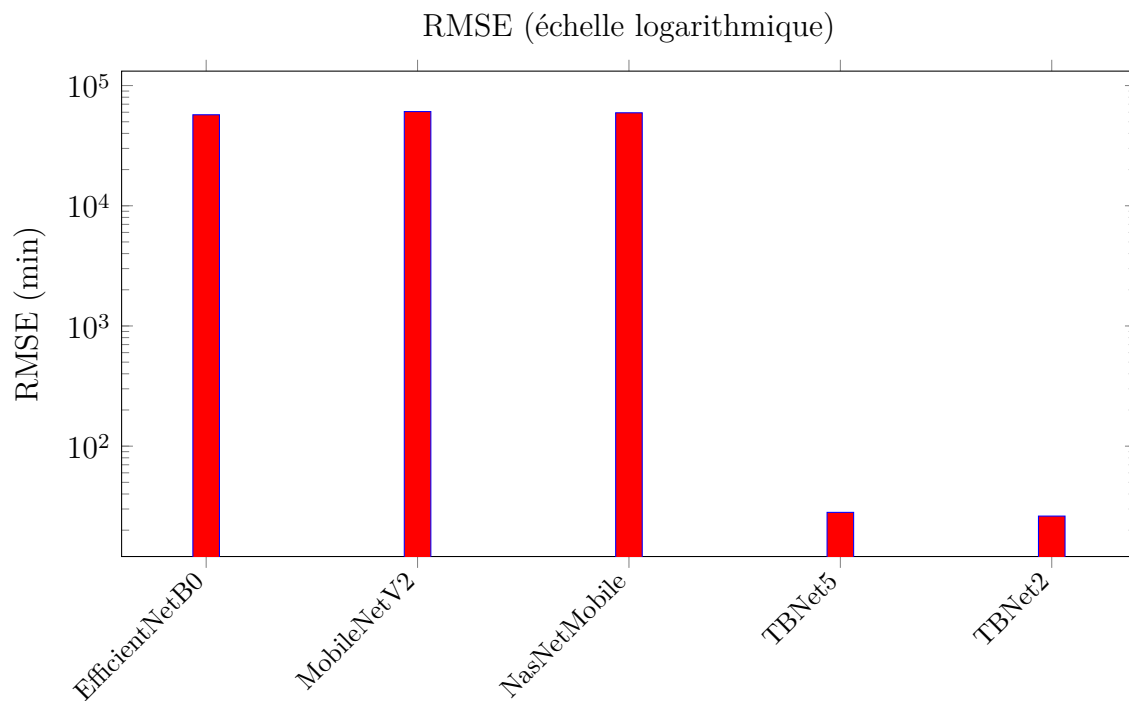


FIGURE 5.4 – Root Mean Square Error (RMSE)

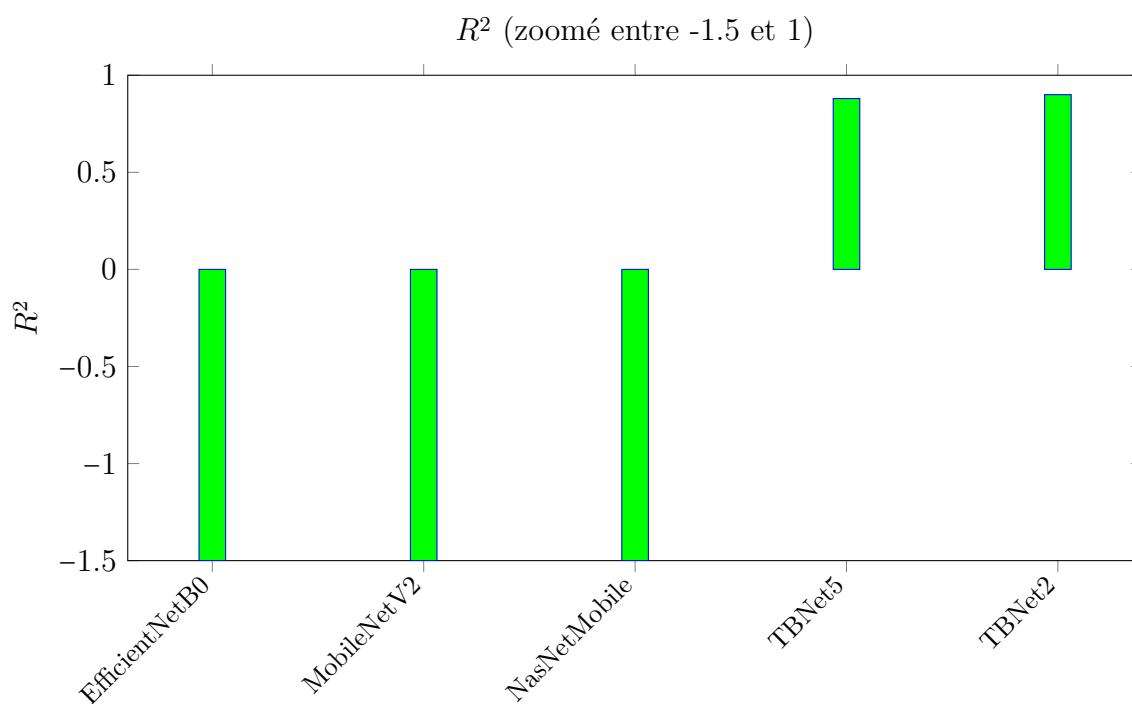


FIGURE 5.5 – Coefficient de détermination ( $R^2$ )

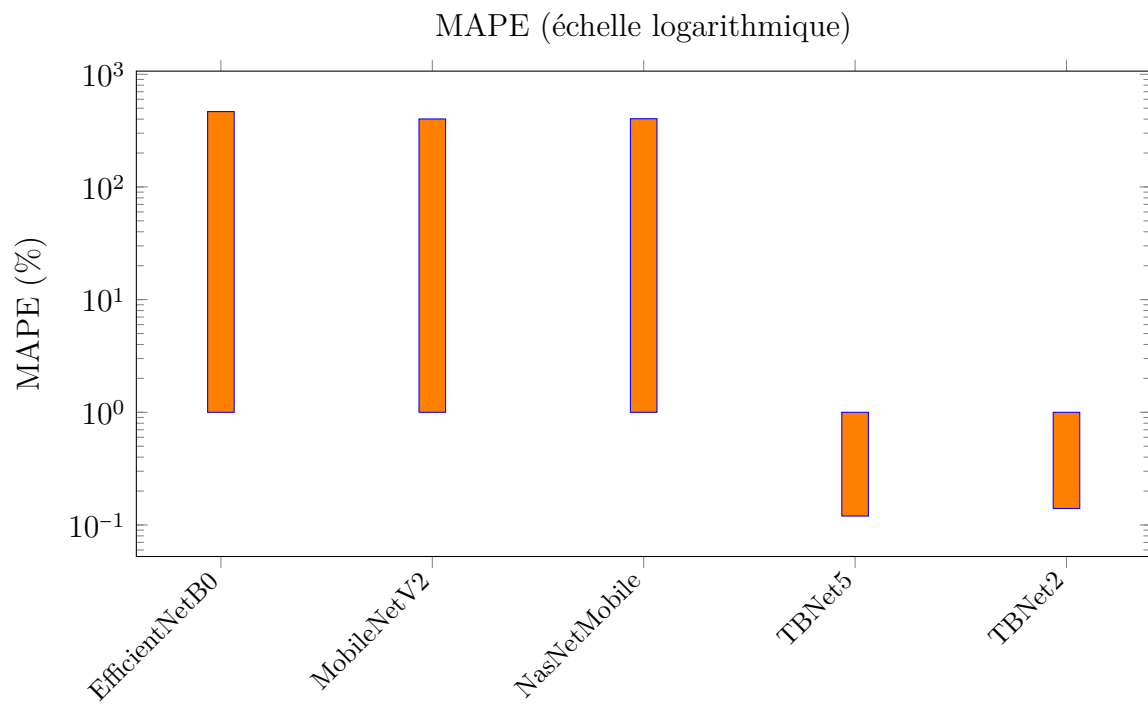


FIGURE 5.6 – Mean Absolute Percentage Error (MAPE)

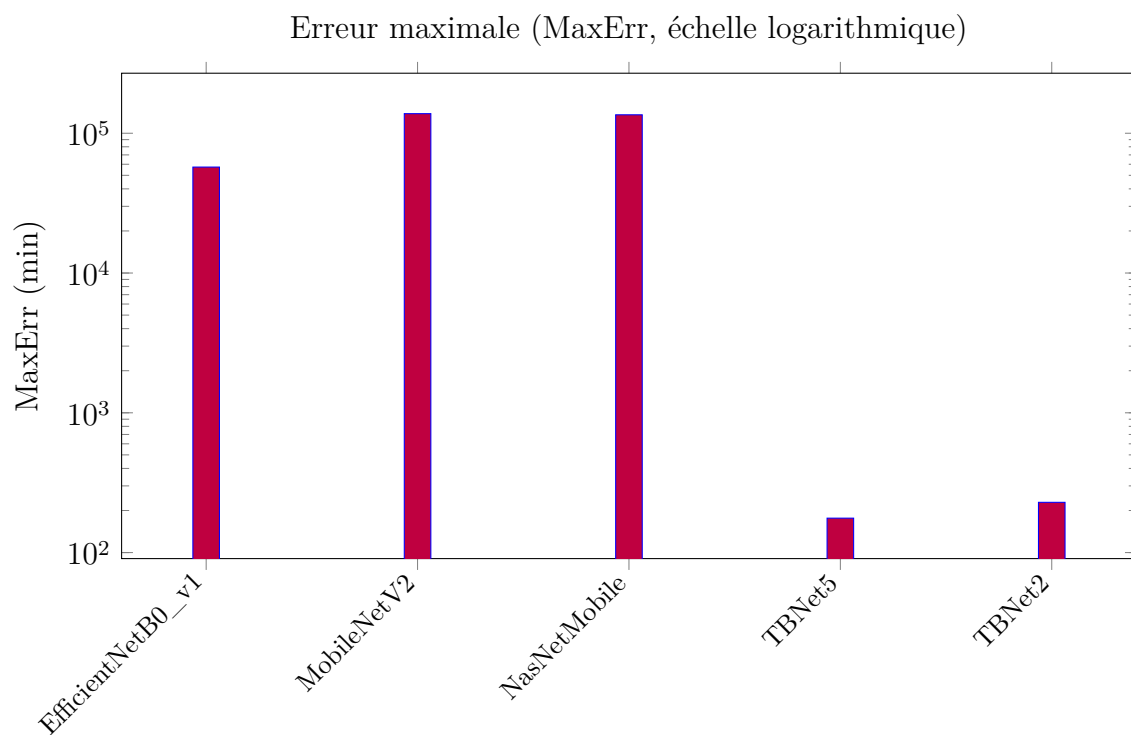


FIGURE 5.7 – Erreur maximale (MaxErr)

### 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 [41], le MAE n'augmentant que de 0.5 à 1 minute en conditions réelles, conformément aux observations de [5].

### 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 [2]. 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 % [24, 21], 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 8 variétés (56 sous-variétés au total) 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 [48], la quantification dynamique [24] ou le *pruning* [49] 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 [50], 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.

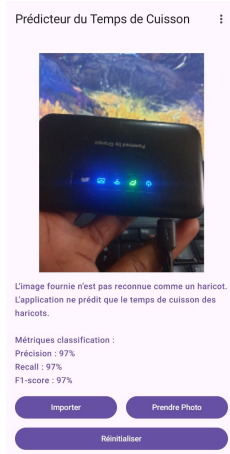
Les limites identifiées et les perspectives proposées ouvrent la voie à des travaux futurs visant à accroître la robustesse, l'explicabilité et l'applicabilité du système. Ainsi, ce travail constitue une étape significative vers l'intégration de solutions d'IA embarquée dans la transformation et la valorisation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

# Annexes

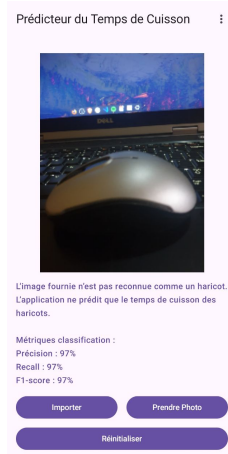
## .1 Liens vers les notebooks Colab et GitHub

- 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

Dans le but de valider l'intégration du modèle sur l'application Android, nous présentons ci-après un ensemble d'images de test capturées sur l'appareil, illustrant la performance et la cohérence des prédictions obtenues.



(a) Image 4



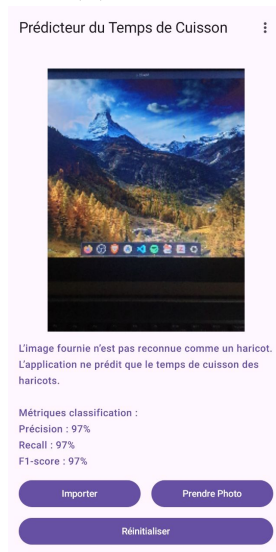
(b) Image 2



(c) Image 3



(d) Image 1



(e) Image 5



(f) Image 6



(g) Image 7



(h) Image 8



(i) Image 9

FIGURE 1 – Images de test de l'application Android montrant les résultats de prédiction.

## Bibliographie

- [1] Department of Economic UNITED NATIONS et Population Division SOCIAL AFFAIRS. *World Population Prospects 2022 : Summary of Results*. <https://population.un.org/wpp/>. Consulté le 16 septembre 2025. 2022.
- [2] 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.
- [3] 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>.
- [4] 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.
- [5] Aylin TASTAN. « Contributions to Robust Graph Clustering : Spectral Analysis and Algorithms ». Thèse de doct. Technische Universität Berlin, 2023.
- [6] A. KAMILARIS et F. PRENAFETA-BOLDÚ. « Deep learning in agriculture : A survey ». In : *Computers and Electronics in Agriculture* 147 (2018), p. 70-90.
- [7] M. RAHMAN et al. « Food cooking estimation using computer vision ». In : *Journal of Food Engineering* 280 (2020), p. 109981.
- [8] M. MOEKETSI et al. « TinyML Applications in Micronutrient Sensing ». In : *Sensors* (2025).
- [9] 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).
- [10] 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.
- [11] 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.

- [12] Samuel M. NJOROGÉ, 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.
- [13] 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.
- [14] 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.
- [15] P. KAUR et al. « Fruit maturity estimation using deep learning ». In : *Journal of Food Engineering* 270 (2020), p. 109762.
- [16] Andrew G. HOWARD et al. « MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications ». In : *arXiv preprint arXiv :1704.04861* (2017).
- [17] Ian GOODFELLOW, Yoshua BENGIO et Aaron COURVILLE. *Deep Learning*. MIT Press, 2016.
- [18] Christopher M. BISHOP. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [19] Michael JONES et Anika PATEL. « On the Pitfalls of Label Encoding for Multi-class Classification ». In : *Machine Learning Review* 4.2 (2020), p. 55-67.
- [20] C. BANBURY et al. « TinyML : Machine learning with ultra-low power micro-controllers ». In : *IEEE Micro* 41.2 (2021), p. 30-40.
- [21] 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.
- [22] 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.
- [23] B. ZOPH et al. « Learning transferable architectures for scalable image recognition ». In : *CVPR* (2018).
- [24] 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.
- [25] J. PATEL et al. « Grain classification using convolutional neural networks ». In : *Computers and Electronics in Agriculture* 162 (2019), p. 572-580.

- [26] 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.
- [27] J. YU et al. « Texture and hardness prediction of food using visual features ». In : *Journal of Food Science* 84 (2019), p. 2150-2160.
- [28] F. CAPOGROSSO et al. « Machine Learning-oriented Survey on Tiny Machine Learning ». In : *arXiv preprint arXiv :2309.11932* (2023).
- [29] M. HEYDARI et M. MAHMOUD. « Tiny Machine Learning and On-Device Inference : A Survey ». In : *MDPI Sensors* (2025).
- [30] ANONYMOUS. « From TinyML to TinyDL : Optimizing deep learning on microcontrollers ». In : *arXiv preprint arXiv :2506.18927* (2025).
- [31] Pete WARDEN et Daniel SITUNAYAKE. *TinyML : Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2019.
- [32] 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.
- [33] Tsung-Yi LIN et al. « Microsoft COCO : Common Objects in Context ». In : *European Conference on Computer Vision (ECCV)* (2014), p. 740-755.
- [34] Christine HALL, Caitlyn HILLEN et Julie GARDEN-ROBINSON. « Cooking quality of dry beans and pulses : A review ». In : *Legume Science* (2017).
- [35] Isabelle GUYON et al. « Analysis of the AutoML Challenge series 2015–2018 ». In : *Automated Machine Learning*. Springer, 2019, p. 177-219.
- [36] Yann LECUN, Yoshua BENGIO et Geoffrey HINTON. « Deep learning ». In : *Nature* 521.7553 (2015), p. 436-444.
- [37] 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).
- [38] Song HAN, Huizi MAO et William J. DALLY. « TinyML : A survey of machine learning for embedded devices ». In : *arXiv preprint arXiv :2007.11624* (2020).
- [39] 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.
- [40] Wei LIU et al. « SSD : Single shot multibox detector ». In : *European conference on computer vision*. Springer. 2016, p. 21-37.

- [41] 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.
- [42] 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.
- [43] Yann LECUN et al. « Gradient-based learning applied to document recognition ». In : *Proceedings of the IEEE* 86.11 (1998), p. 2278-2324.
- [44] 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.
- [45] 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.
- [46] 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.
- [47] 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.
- [48] Geoffrey HINTON, Oriol VINYALS et Jeff DEAN. « Distilling the Knowledge in a Neural Network ». In : *NIPS Deep Learning and Representation Learning Workshop*. 2015.
- [49] 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>.
- [50] 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>.