TinyML for Bean Cooking Time Prediction

MBAIGOLMEM DANG-DANG THIERY [1]

7 septembre 2025

1. dev.thiery.dangdang@gmail.com

Table des matières

Table des figures

Liste des tableaux

1 Introduction générale

1.1 Contexte et justification

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

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

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

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

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

1.2 Problématique

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

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

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

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

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

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

1.3 Objectifs et contributions

1.3.1 Objectif général

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

1.3.2 Objectifs spécifiques

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

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

1.3.3 Contributions attendues

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

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

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

1.4 Organisation du mémoire

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

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

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

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

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

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

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

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

2 Revue de la littérature

2.1 Introduction

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

2.2 Intelligence artificielle et secteur agroalimentaire

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

2.3 Prédiction du temps de cuisson des légumineuses et travaux connexes

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

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

2.4 Convolution et réseaux de neurones convolutifs (CNN)

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

2.4.1 Définition mathématique

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

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

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

2.4.2 Padding, stride, dilatation

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

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

2.4.3 Coût, paramètres et convolutions séparables

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

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

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

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

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

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

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

2.4.4 Autres blocs légers

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

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

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

2.5 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) [16]. L'objectif est de réaliser des inférences en temps réel localement, avec une faible empreinte mémoire et énergétique, rendant l'IA accessible même dans des environnements à faible connectivité.

2.5.1 Contraintes matérielles typiques

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

2.5.2 Avantages et cas d'utilisation

— **Traitement local** : réduction de la latence et de la dépendance au cloud.

— **Efficacité énergétique** : consommation minimale adaptée aux dispositifs alimentés par batterie.

— **Confidentialité accrue** : les données ne quittent pas le dispositif, limitant les risques de fuite.

— **Accessibilité économique** : microcontrôleurs peu coûteux adaptés à une large diffusion.

Applications emblématiques :

— **Agriculture** : détection de maladies, estimation de la qualité des récoltes, suivi en temps réel [7, 8].

— **Agroalimentaire** : classification des produits, prédiction de la texture et du temps de cuisson.

— **Santé** : suivi des signaux physiologiques sur dispositifs portables.

— **IoT industriel** : capteurs intelligents (maintenance prédictive, détection d'anomalies).

2.5.3 Techniques d'optimisation

Quantification (INT8, FP16) La quantification convertit les tenseurs (poids/activations) en représentations numériques plus compactes. Le modèle flottant $x \in \mathbb{R}$ est approximé par une valeur quantifiée q dans $[q_{\min}, q_{\max}]$ avec une échelle $s > 0$ et un *zero-point* z :

$$q = \mathrm{clip}\Big(\Big\lfloor \tfrac{x}{s} + z \Big\rceil, q_{\min}, q_{\max}\Big), \quad \hat{x} = s(q - z). \tag{2.5}$$

Cas courants :

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

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

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

Pruning (élagage) Suppression de poids ou redondances pour réduire mémoire et calculs. Le pruning *non structuré* (mise à zéro de poids individuels) réduit la taille mais n'accélère pas toujours la latence ; le pruning *structuré* (suppression de canaux, filtres ou blocs) permet des gains réels sur matériel. Des approches progressives maintiennent la précision [17].

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

2.5.4 Avancées et perspectives du TinyML

Évolutions notables :

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

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

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

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

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

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

2.6 Modèles légers pour dispositifs contraints

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

- **MobileNet** : basé sur les convolutions séparables en profondeur, réduisant drastiquement le nombre de paramètres [15].
- **EfficientNet** : propose une mise à l'échelle équilibrée en profondeur, largeur et résolution pour maximiser l'efficacité [18].
- **NASNetMobile** : utilise la recherche automatique d'architectures (NAS) pour identifier les modèles optimaux pour mobiles [19].

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

2.7 Métriques d'évaluation des modèles

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

2.7.1 Métriques de régression

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

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

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

  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)** :

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

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

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

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

2.7.2 Métriques de classification

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

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

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

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

— **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 §2.5.3).

2.9 Synthèse des travaux antérieurs

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

Réf.	Année	Titre	Objectif	Méthodologie	Résultats	Limites	Pertinence
[5]	2018	Deep learning in agriculture	Survol IA agriculture	Revue	IA utile classification	Peu de focus sur la cuisson	Contexte agroalimentaire
[6]	2020	Food cooking estimation	Estimer temps de cuisson via images	Analyse d'images	Corrélation détectée	Échantillon limité	Lien direct avec cuisson
[3]	2021	TinyML principles	Définir TinyML	Discussion	TinyML possible sur MCU	Manque d'implémentations concrètes	Cadre TinyML
[15]	2017	MobileNet CNN	CNN léger pour mobiles	Architecture	Bonne précision, faible coût	Coût élevé pour certains MCU	Base pour modèles légers
[18]	2019	EfficientNet scaling	Optimiser mise à l'échelle CNN	Architecture optimisée	Meilleure efficacité	Toujours lourd pour TinyML	Optimisation CNN
[19]	2018	NASNetMobile	Optimisation via NAS	Recherche automatique	Architecture optimisée	Complexité de la NAS	Modèles mobiles
[20]	2018	Quantization NN	Réduire taille modèles	Quantification INT8/FP16	Réduction mémoire x4	Perte de précision possible	Compression mémoire
[17]	2016	Pruning deep nets	Réduction des paramètres	Pruning	Réduction paramètres x10	Perte de précision si pruning excessif	Réduction mémoire
[21]	2019	Grain classification	Classifier variétés de grains	CNN grains	Bonne précision	Peu généralisable	Classification grains/haricots
[14]	2020	Fruit maturity estimation	Prédire maturité et tendreté	CNN fruits/légumes	Prédictions fiables	Dataset limité	Lien cuisson/texture
[22]	2021	Food defect detection	Détection de défauts alimentaires	Analyse d'images	Détection robuste	Applicabilité restreinte	Qualité alimentaire
[23]	2019	Texture prediction food	Corréler images et texture	Vision + spectroscopie	Corrélations confirmées	Peu d'applications concrètes	Lien cuisson/texture
[7]	2025	TinyML micronutrient sensing	TinyML pour détection micronutriments	Revue PRISMA	Modèles hybrides performants	Reporting incomplet	Insight TinyML en agriculture
[8]	2024	Domain-Adaptive TinyML	Détection maladies via TinyML	2D-CNN + adaptation domaine	Performance correcte	Chute de performance en conditions réelles	Adaptabilité TinyML agri
[24]	2023	Survey TinyML	Taxonomie TinyML	Revue PRISMA	Mapping optimisations	Peu d'applications agricoles	Ressources méthodologiques
[25]	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
[26]	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) [27, 28].

L'architecture logicielle de la solution met ainsi en œuvre deux pipelines de prétraitement et deux modèles distincts, chacun optimisé et quantifié selon des objectifs différents : le modèle de classification est conçu pour être très léger et quantifié en INT8 (ex. MobileNetV2 quantifié post-training), tandis que le modèle de régression privilégie la précision et peut utiliser une quantification en float16 ou une quantification mixte selon les tests empiriques. La logique applicative (smartphone) orchestre l'enchaînement classification → décision → régression et l'affichage des résultats à l'utilisateur.

Dans la suite du chapitre, nous décrivons les jeux de données (séparés pour classification et régression), l'analyse exploratoire, le protocole de prétraitement distinct pour chaque tâche, les architectures testées, le workflow d'entraînement et d'optimisation TinyML, ainsi qu'une comparaison chiffrée et une discussion critique des choix adoptés.

3.2 Vue d'ensemble du pipeline proposé

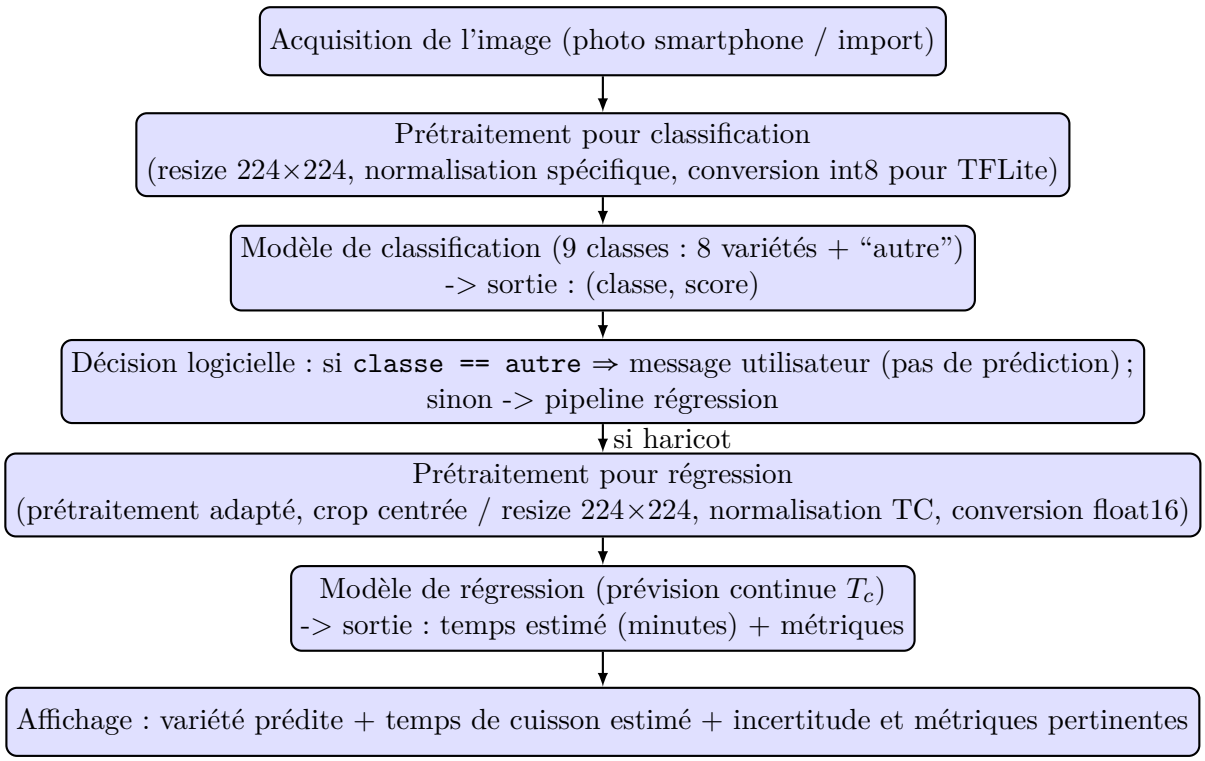

FIGURE 3.1 – Pipeline global du système : classification suivie (si nécessaire) d'une régression.

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

- **Composition** : 9 classes — huit variétés de haricots, chacune subdivisée en 7 sous-variétés (soit 8×7 = 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)* [29]. Ces 1 000 images constituent un échantillon du split validation original du dataset COCO, qui en contient 5 000.
- **Taille totale** : 12 200 images (11 200 images de haricots + 1 000 images de la classe `"autre"`).
- **Splits** :
  - Entraînement/validation : 190 images par sous-variété (soit $190 \times 56 = 10\,640$) + 800 images de la classe `"autre"`, réparties selon une proportion 80/20.
  - Test : 10 images par sous-variété (soit $10 \times 56 = 560$) + 200 images de la classe `"autre"`.

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

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

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

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

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* [30]. Une incertitude expérimentale estimée à ±1 minute (variabilité opérateur et conditions de cuisson) est attachée à chaque échantillon.

— **Splits** : les données ont été séparées selon un ratio 80/10/10 en ensembles d'entraînement, validation et test, avec une seed fixe de **42** pour assurer la reproductibilité. Afin de respecter la distribution des temps de cuisson par variété, une stratification par quantiles de T_c a été appliquée, comme recommandé dans des contextes de régression hétérogène [31].

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

— **Prétraitement** : toutes les images ont été redimensionnées en 224×224 pixels. Contrairement aux pratiques standards de normalisation des intensités dans $[0, 1]$ [32], les images ont été conservées dans l'espace d'origine $[0, 255]$ afin de réduire la taille des fichiers sauvegardés (au format HDF5). La normalisation est réalisée dynamiquement au moment du chargement en mémoire, ce qui optimise le compromis entre efficacité de stockage et préparation des données pour l'entraînement.

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

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

Ainsi, les labels sont projetés dans $[0, 1]$. Ce choix est cohérent avec la littérature, la mise à l'échelle linéaire facilitant l'optimisation dans les réseaux de neurones et réduisant les effets liés à la différence d'échelle entre variables [33].

— **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.

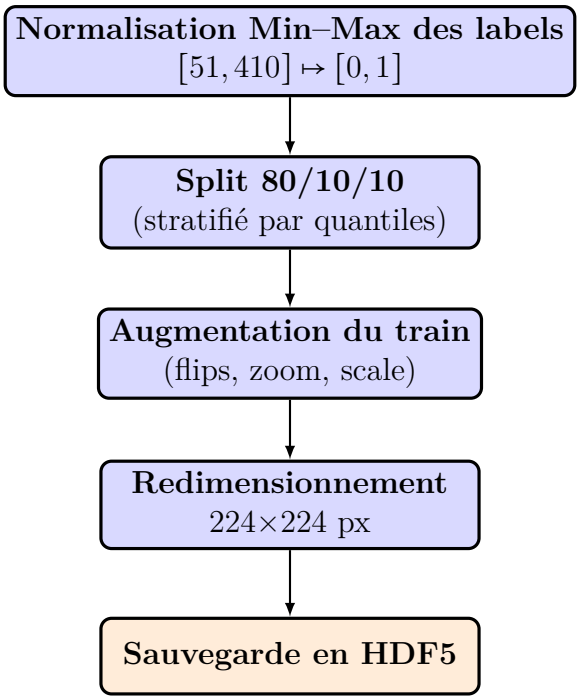

FIGURE 3.2 – Pipeline de préparation du dataset de régression.

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. En-

traîner directement un modèle de régression sur de telles images conduirait à des prédictions aberrantes ou hors domaine [34]. 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 [35]. Ces différences impliquent des prétraitements distincts, des augmentations adaptées, et des stratégies de normalisation ciblées pour stabiliser l'entraînement [32]. 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) [36]. 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 [37, 38].

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.

3.3.1 Synthèse statistique pour la régression

Le tableau 3.1 (extrait et adapté) 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 ≈ 118.9 min à ≈ 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 (μ)**	**Écart-type (σ)**	**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

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-varieté ou l'introduction d'un embedding de variété (si la variété est connue) pour améliorer la performance locale.

3.3.2 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 [29]. 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 [39].

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 :

— La classe `autre` est conservée telle quelle pour entraîner le modèle à détecter et rejeter les images hors-domaine.

— Une validation croisée stratifiée est utilisée pour évaluer la performance sur toutes les classes, y compris `autre`, afin de s'assurer que la taille réduite ne nuit pas à la capacité de généralisation [40].

3.4 Architectures modèles et justification

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

3.4.1 Modèle de classification

— **Backbone recommandé** : MobileNetV2 (1.0, 224) fine-tuné sur $\mathcal{D}_{\text{clf}}$. Raisons : excellente performance/empreinte mémoire, blocs inverted-residual favorables à la quantification INT8 et historique d'utilisation mobile [41].

— **Sortie** : softmax 9 unités (8 variétés + `autre`).

— **Fonction de perte** : categorical cross-entropy avec éventuelle pondération de classes (class weights) pour corriger le déséquilibre.

— **Métriques** : accuracy globale, F1-score macro (pour tenir compte du déséquilibre), recall/precision par classe, matrice de confusion. (Les définitions formelles des métriques sont détaillées dans la revue de littérature.)

— **Optimisations** : PTQ int8 ; si dégradation trop forte, utiliser QAT.

3.4.2 Modèle de régression

— **Backbones testés** : CNN personnalisé (CNN-1 / CNN-2), EfficientNet-B0 (fine-tuning) selon compromis précision/taille. Le modèle final sélectionné dans les expériences précédentes a été un CNN personnalisé entraîné from-scratch et plus simple (cf. rapport d'entraînement).

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

— **Fonction de perte** : MSE (principal), suivi de MAE/MAPE comme métriques secondaires et critères d'arrêt ; suivi de R^2 pour évaluer la part de variance expliquée.

— **Optimisations** : quantification float16 pour préserver la précision numérique ; PTQ int8 testé avec calibration pour mesurer impact.

3.5 Workflow d'entraînement, conversion et évaluation

3.5.1 Procédure d'entraînement reproductible

Pour chaque modèle (classification et régression) :

1. Préparer HDF5 (ou TFRecord) 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. Logger via TensorBoard, sauvegarder checkpoints, et conserver le meilleur modèle sur la validation.

3.5.2 Conversion et quantification TFLite

— **Classification** : conversion PTQ INT8 avec dataset de calibration (subset representatif). Si la performance chute de plus de X% (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.

3.5.3 Évaluation finale et mesures embarquées

Mesures à effectuer sur la plateforme cible (smartphone Android API24) :

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

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

— **Consommation énergétique** par inférence (si instrumentation possible).

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

3.6 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_{rej} (ex. 0.6), l'application affiche un message : *"L'application ne prédit le temps de cuisson que pour des images de haricots. Veuillez fournir une photo claire d'un haricot."*
4. Si le modèle prédit une des 8 variétés avec confiance suffisante, l'application déclenche le pipeline $\mathcal{P}_{\text{reg}}$ (prétraitement pour régression) et exécute le modèle de régression (float16).
5. L'interface affiche : la **variété** prédite (avec score), le **temps estimé** T_c (minutes) retransformé depuis l'échelle normalisée, et une estimation d'intervalle d'incertitude (ex. ± MAE sur la variété si disponible). Les métriques globales (acc, F1 pour la classification ; MAE, RMSE pour la régression) sont accessibles dans une section "Plus d'infos" pour les utilisateurs techniques.

Gestion des seuils et de la confiance Il est recommandé d'ajouter un mécanisme de rejet (thresholding) pour éviter d'exécuter la régression si la confiance de classification est basse. De plus, enregistrer ces exemples faibles en confidence pour une future annotation manuelle permettra d'améliorer le dataset (active learning).

3.7 Discussion critique et limites

3.7.1 Points forts

— **Robustesse opérationnelle** : la classification préalable évite des prédictions aberrantes hors-domaine.

— **Efficacité embarquée** : quantification et architecture mobile permettent une exécution locale rapide et à faible consommation.

— **Séparabilité des responsabilités** : deux modèles spécialisés facilitent le débogage, la maintenance et la mise à jour incrémentale.

3.7.2 Limites et pistes d'amélioration

— **Déséquilibre de la classe "autre"** : risque de sur-apprentissage sur cette classe ; contraindre le modèle avec des pénalités de classe et enrichir la variété d'exemples hors-domaine.

— **Variabilité inter-variétés** : certaines variétés à forte variance (ex. `Macc55`) limitent la précision de la régression ; solutions : entraînement par-varieté, mo-

dèles spécialisés, ajout de métadonnées (humidité, conditions de stockage) si disponibles.

— **Quantification pour la régression** : int8 peut être trop agressif pour les tâches régressives ; float16 est souvent un bon compromis mais augmente légèrement la taille.

— **Mesures matérielles nécessaires** : latence et consommation doivent être mesurées sur divers appareils (CPU-only, NNAPI, GPU delegate) pour formuler des recommandations pratiques.

3.8 Bonnes pratiques et recommandations pour la reproductibilité

— Consigner les seeds aléatoires pour la génération des splits et les augmentations.

— Exporter les artefacts (HDF5/TFRecord, checkpoints, scripts d'entraînement, scripts de conversion TFLite).

— Tester les modèles TFLite sur une suite d'appareils cibles pour mesurer latence et mémoire.

— Intégrer des tests unitaires comparant prédictions numpy/TF/Keras vs TFLite (tolerance MAE).

3.9 Conclusion du chapitre

Ce chapitre a présenté une méthodologie complète et reproductible pour la conception d'un système hybride *classification → régression* destiné à prédire le temps de cuisson des haricots à partir d'images. La séparation des jeux de données et des pipelines de prétraitement, l'usage d'un modèle de classification quantifié INT8 pour filtrer le domaine, et d'un modèle de régression optimisé pour la précision (float16) constituent l'ossature de la proposition. Les choix techniques (architectures, augmentations, stratégies de quantification) sont motivés par un compromis précision/latence adapté aux contraintes embarquées. Le chapitre suivant (Implémentation) détaillera les scripts Keras/TensorFlow, les commandes de conversion TFLite (PTQ / QAT), les détails d'entraînement (callbacks, scheduler) et présentera les résultats empiriques complets (MAE, RMSE, R^2, matrices de confusion, latence et consommation mesurées).

4 Développement et implémentation

Ce chapitre présente la mise en œuvre concrète du système de prédiction du temps de cuisson des haricots, depuis la préparation des données jusqu'au déploiement embarqué. Il s'appuie sur la méthodologie définie au Chapitre 3 et reprend strictement les choix techniques (prétraitements, architectures candidates, schéma d'entraînement et optimisation TinyML).

4.1 Environnement de développement

4.1.1 Matériel

Les expérimentations ont été menées sur :

— **Machine locale** : Intel Core i7 (4 cœurs), 16 Go RAM.

— **Google Colab** : session `Tesla T4` (15 Go VRAM, 12 Go RAM, 118 Go stockage).

— **Samsung Galaxy Note 9** : Octa-core, 6 Go RAM, 128 Go stockage, Android 10.

4.1.2 Logiciels et versions

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

— **Python** 3.10

— **TensorFlow** 2.19 pour l'entrainement et **TensorFlow** 2.13 pour la quantification & API Keras (compatibles `TensorFlow Lite`)

— **Pandas** 2.1, **NumPy** 1.25, **Matplotlib** 3.8

4.2 Préparation des données

Le protocole de préparation suit *strictement* le Chapitre 3 :

1. **Recadrage centré** et **redimensionnement** des images brutes (3000×4000) en 224×224.
2. **Mise à l'échelle** des pixels dans $[0, 1]$.
3. **Augmentation** stochastique *train-only* (flips, crops, jitter de luminosité/contraste/saturation, blur, bruit gaussien) [1].
4. **Normalisation Min–Max** des labels $T_c \in [0, 1]$, calculée uniquement sur $\mathcal{D}_{\text{train}}$, appliquée à val/test, puis inversée pour restituer les minutes en sortie.

1. Paramètres détaillés : voir Chap. 3.

5. **Export HDF5** par split avec clés `images`, `cooking_times`.

```
\small
1. Charger le fichier CSV contenant les chemins d'images et les étiquettes (labels).
2. Extraire la liste des chemins d'images et convertir les labels au format float.
3. Définir les transformations de base (redimensionnement, conversion en tenseur).
4. Définir les transformations d'augmentation de données
   (redimensionnement, recadrage aléatoire, flips, jitter couleur, flou gaussien).
5. Pour chaque chemin d'image :
    a. Ouvrir l'image et la convertir en RGB.
    b. Appliquer la transformation de base.
    c. Convertir l'image en tableau numpy uint8 et l'ajouter à la liste des images c
6. Convertir la liste des images et labels en tableaux numpy.
7. Normaliser les labels entre 0 et 1.
8. Séparer le jeu de données en ensembles d'entraînement, de validation et de test
   (stratification sur les labels).
9. Pour chaque image de l'ensemble d'entraînement :
    a. Ajouter l'image originale et son label.
    b. Pour un nombre donné d'augmentations :
        i. Convertir l'image en format PIL.
       ii. Appliquer les transformations d'augmentation.
      iii. Convertir l'image augmentée au format numpy uint8.
       iv. Ajouter l'image augmentée et le label à la liste.
10. Convertir les listes d'images et labels augmentés en tableaux numpy.
11. Enregistrer chaque ensemble (entraînement, validation, test) dans un fichier HDF
    avec deux jeux de données : images et labels (temps de cuisson).
```

4.3 Architectures de modèles

Conformément à la méthodologie :

— **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 retenus.

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

4.4 Stratégie d'entraînement

Le modèle est entraîné sur $\mathcal{D}_{\text{train}}$, validé sur $\mathcal{D}_{\text{val}}$. L'*early stopping* permet d'éviter le surapprentissage.

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

4.5 Export et optimisation TinyML

4.5.1 Conversion TensorFlow Lite (float16)

La quantification retenue est **float16**, permettant de réduire la taille mémoire d'environ 50% tout en préservant la précision.

```
1. Importer la bibliothèque TensorFlow (version 2.13).
2. Charger le modèle enregistré (SavedModel).
3. Initialiser un convertisseur TFLite à partir du modèle chargé.
4. Activer les optimisations par défaut pour la conversion.
5. Spécifier que le type de données cible est float16 (pour Android).
6. Convertir le modèle en format TFLite.
7. Sauvegarder le modèle TFLite dans un fichier binaire.
8. Afficher un message de succès de la conversion.
```

4.6 Déploiement Android

4.6.1 Cible et outils

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

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

— **Langage** : Kotlin.

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

4.6.2 Intégration et inférence

Le modèle `.tflite` est placé dans `assets/`. Le pipeline embarqué applique le même prétraitement (resize 224×224, normalisation), puis l'inférence, suivie de l'inversion Min–Max pour obtenir le temps en minutes.

```
\small
1. MainViewModel (ViewModel) :
    - Définir états observables : imageBitmap, predictionResult.
    - Constantes : taille image (224x224), min/max temps cuisson.
    - loadModelFile(context, modelPath) : charger modèle TFLite depuis assets.
    - runPrediction(context, bitmap) :
        a. Mettre à jour imageBitmap et predictionResult.
        b. Lancer coroutine pour exécuter runModelInference.
        c. Mesurer latence et mettre à jour predictionResult.
        d. Gérer erreurs.
```

- runModelInference(context, bitmap) :
    a. Convertir bitmap en ARGB_8888 si nécessaire.
    b. Redimensionner à 224x224.
    c. Créer ByteBuffer, normaliser pixels [0,1].
    d. Charger interpréteur TFLite.
    e. Exécuter inférence et récupérer sortie.
    f. Dénormaliser prédiction et retourner.
- denormalize(normalizedValue) : convertir sortie normalisée en valeur réelle.
- reset() : réinitialiser états.

2. IntroManager :
- Gérer SharedPreferences pour écran d'introduction.
- shouldShowIntro() : retourner booléen.
- setShowIntro(shouldShow) : modifier préférence.

3. MainActivity :
- Initialiser IntroManager.
- Afficher IntroScreen si besoin, sinon écran prédiction.
- Navigation entre PredictionScreen et InfoScreen.

4. Composables UI :
- IntroScreen : boîte de dialogue avec option "ne plus afficher".
- PredictionScreen :
    a. Gérer permissions caméra, sélection image galerie/camera.
    b. Afficher image sélectionnée.
    c. Afficher résultats prédiction (temps, RMSE, MAE).
    d. Boutons importer, prendre photo, réinitialiser.
- InfoScreen :
    a. Présentation app.
    b. Explication métriques MAE et RMSE.
    c. Contexte cuisson haricots.

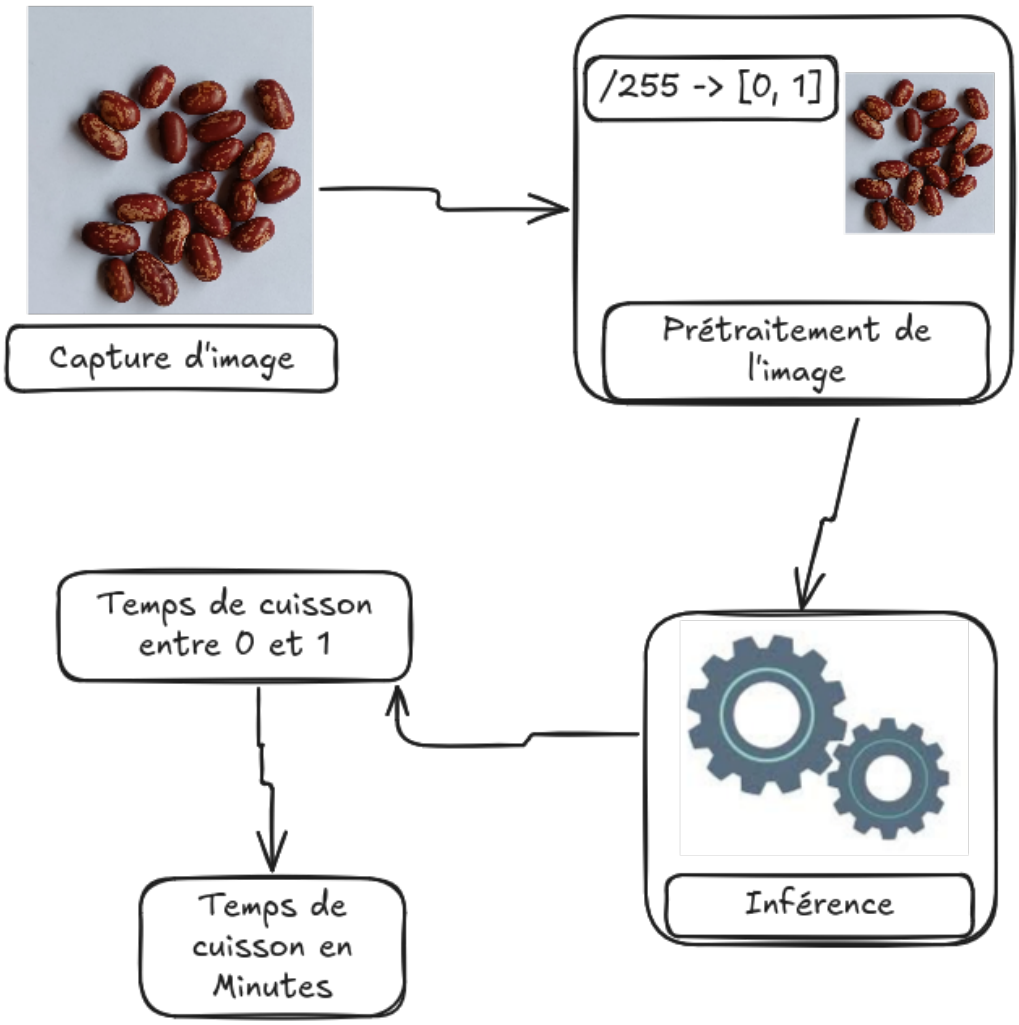

FIGURE 4.1 – Workflow fonctionnel de l'application Android : capture ou import d'image, prétraitement, inférence et affichage du temps de cuisson (T_c).

UI/Workflow.

4.7 Mesures et artefacts à produire

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

TABLE 4.2 – (Placeholder) Synthèse des modèles déployables.

Modèle	Fichier TFLite	Taille (Mo)	Latence (ms)†	Notes
TBNet2	model_fp16.tflite	0.9	187	float16
MobileNetV2	model_fp16.tflite	7.2	734	float16
EfficientNetB0	model_fp16.tflite	8.7	769	float16
NASNetMobile	model_fp16.tflite	21.3	867	float16

† Mesures effectuées sur appareil cible (moyenne et percentiles).

4.8 Limites pratiques et points d'attention

— **Alignement des prétraitements** : le pipeline embarqué doit reproduire fidèlement la normalisation et l'inversion Min–Max.

— **Latence et consommation** : fortement dépendantes de l'appareil et des délégués (CPU/GPU/NNAPI).

— **Gestion mémoire** : réutilisation des buffers d'E/S et chargement paresseux recommandés côté Android.

— **Robustesse applicative** : prévoir la gestion des erreurs (fichier image invalide, absence d'entrée).

Résumé — Ce chapitre documente l'implémentation complète (prétraitement, modèles, entraînement, conversion float16, intégration Android) et définit les artefacts à produire. Les résultats expérimentaux seront présentés et analysés au Chapitre 5.

5 Résultats et Discussion

Ce chapitre présente et analyse en profondeur les résultats obtenus lors de l'entraînement et du test des différents modèles de prédiction du temps de cuisson des haricots. Il discute des erreurs observées, de la robustesse des modèles, et des limites et perspectives pratiques et scientifiques de ce travail. Les figures et tableaux permettent une visualisation claire des performances pour toutes les métriques principales.

5.1 Résultats d'entraînement et de test

5.1.1 Courbes de perte et convergence des modèles

L'évaluation a reposé sur l'analyse de la perte d'entraînement et de validation, ainsi que sur les métriques de régression : MAE, RMSE, R^2, MAPE et MaxErr.
Les CNN personnalisés (`TBNet5` et `TBNet2`) convergent rapidement, stabilisant la perte après 20–25 époques. Les modèles pré-entraînés (MobileNetV2, EfficientNetB0, NASNetMobile) présentent une convergence plus lente et des fluctuations de validation plus marquées, indiquant un surapprentissage potentiel sur ce dataset de taille moyenne.

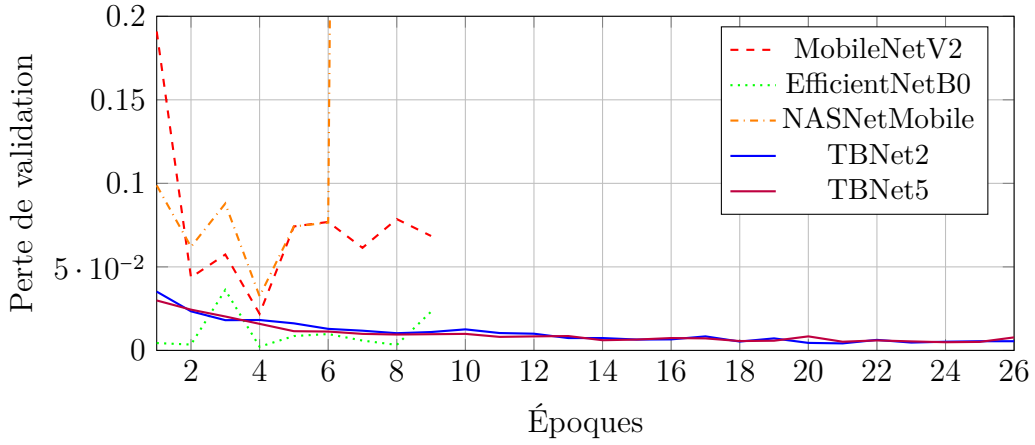

FIGURE 5.1 – Évolution des pertes de validation pour les différents modèles testés sur l'ensemble des époques.

TABLE 5.1 – Performances des modèles sur le jeu de test.

Modèle	**MAE (min)**	**RMSE (min)**	R^2	**MAPE (%)**	**MaxErr (min)**
mobnet_fige_v1	55499.68	60155.30	-530917.00	387.20	153697.80
EfficientNetB0_v1	57162.30	57162.35	-479401.06	466.53	57288.20
NasNetMobile_v1	56730.74	61084.55	-547446.50	402.03	149287.95
modele_2	49596.76	50963.23	-381059.19	378.60	110173.19
model_96_1	42700.86	44798.93	-294451.38	315.60	112126.64
mobnet_fige_v2	56606.87	60760.02	-541644.94	401.15	138063.39
NasNetMobile_v2	55799.49	59374.25	-517219.72	403.25	135507.25
TBNet5	16.29	28.13	0.88	0.12	176.35
TBNet2	16.40	26.20	0.90	0.14	228.87

5.2 Visualisation des performances des modèles

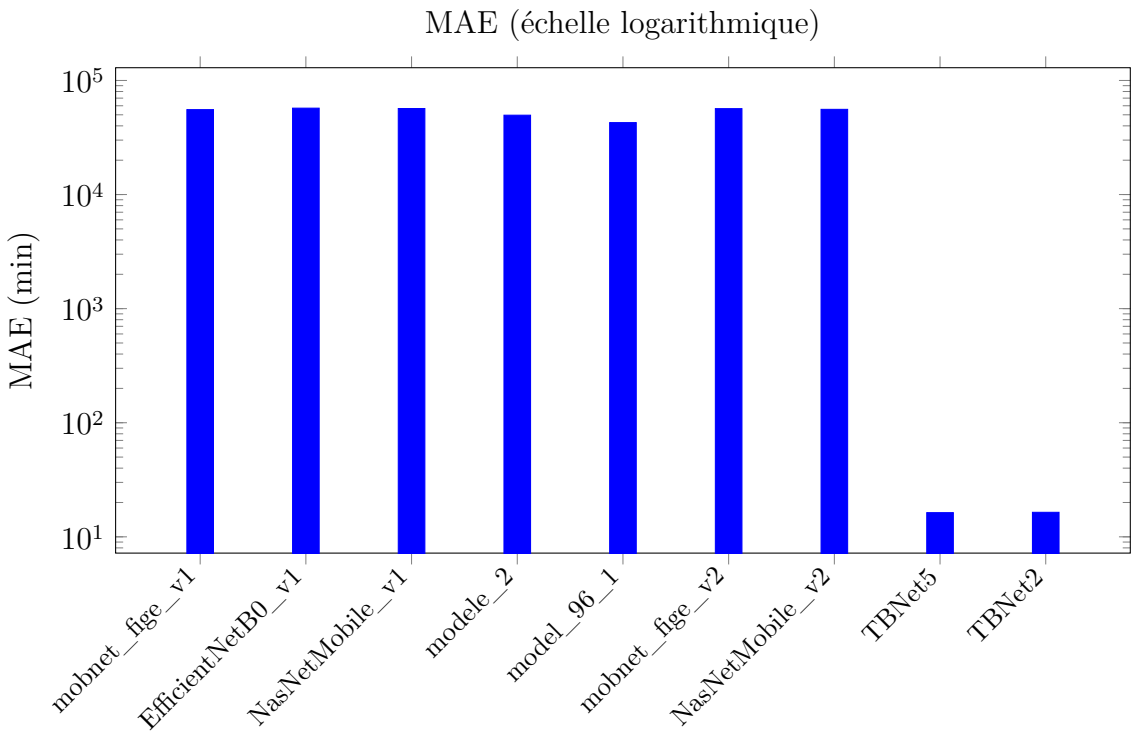

FIGURE 5.2 – Mean Absolute Error (MAE)

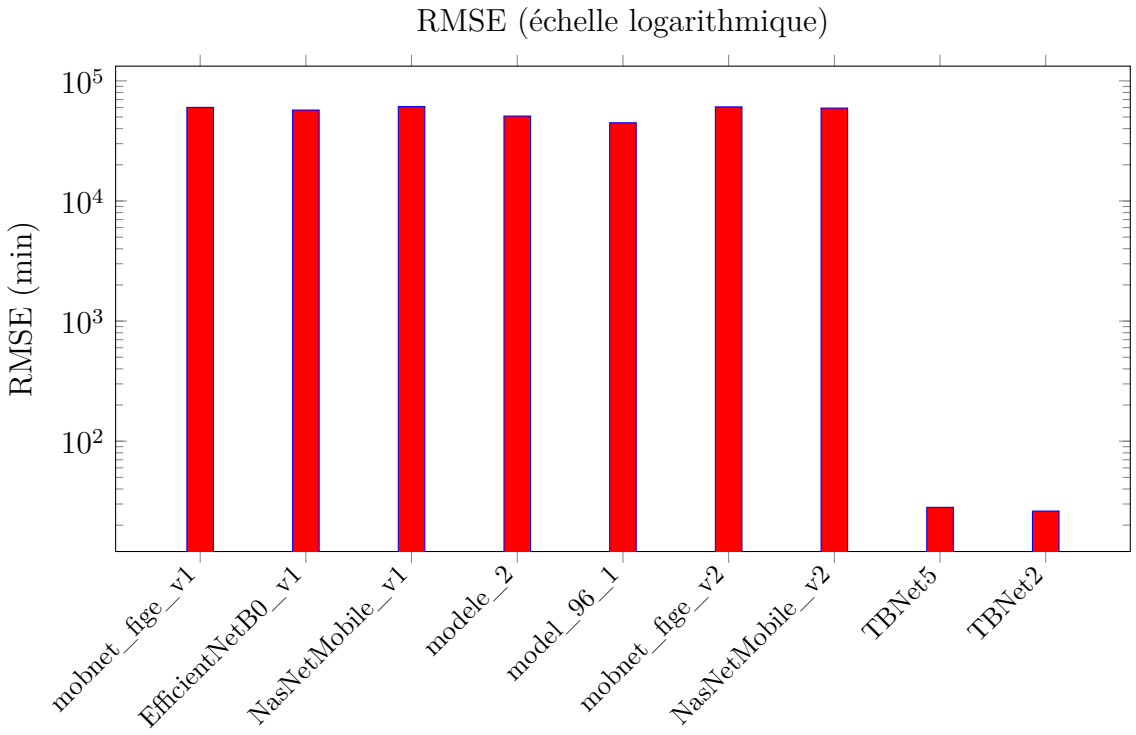

FIGURE 5.3 – Root Mean Square Error (RMSE)

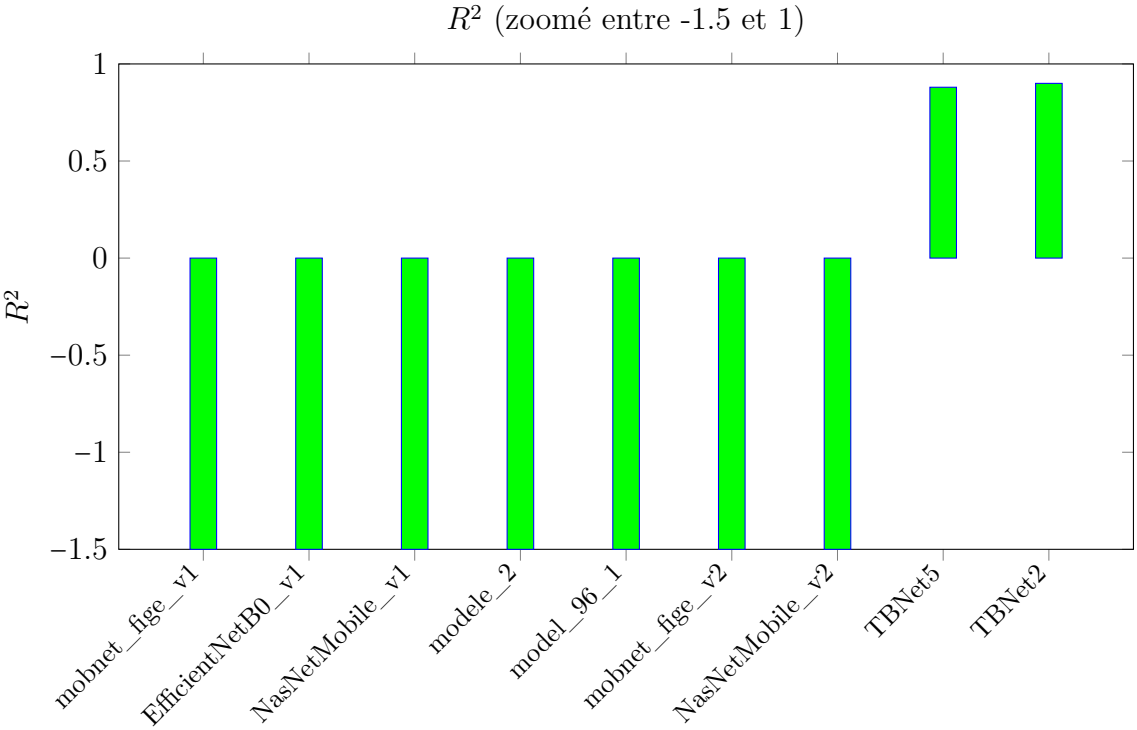

FIGURE 5.4 – Coefficient de détermination (R^2)

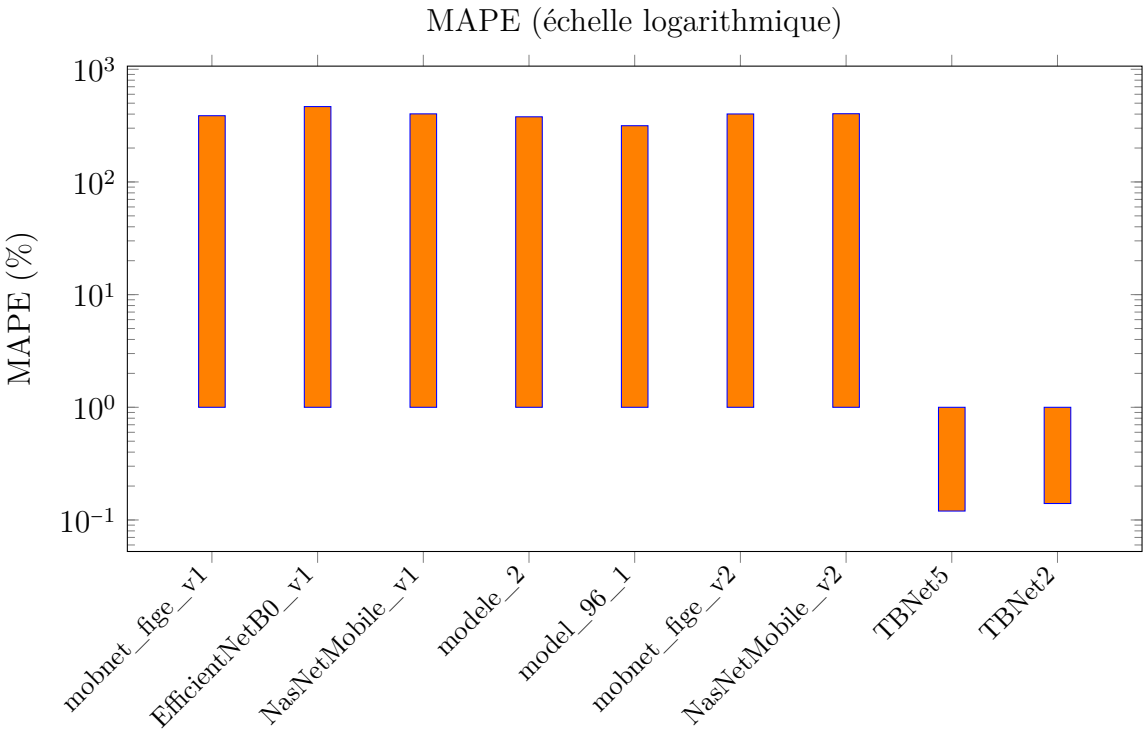

FIGURE 5.5 – Mean Absolute Percentage Error (MAPE)

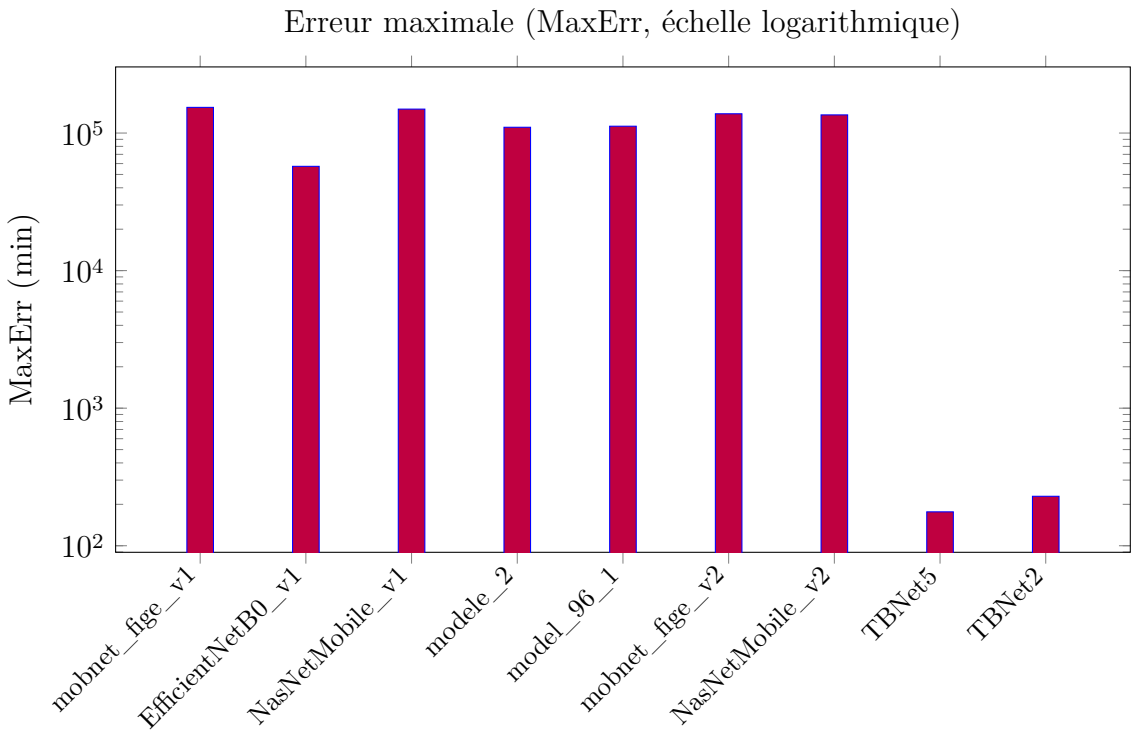

FIGURE 5.6 – Erreur maximale (MaxErr)

5.3 Analyse par variété

Les variétés sombres ou homogènes augmentent l'erreur, confirmant l'importance de l'augmentation de données. Le Tableau 5.2 montre un exemple représentatif.

TABLE 5.2 – Performances par variété pour le modèle `TBNet5`.

Variété	MAE (min)	RMSE (min)	MAPE (%)	MaxErr (min)
Dor701	18.2	30.1	0.14	190
Escapan021	15.6	27.5	0.11	170
GPL190C	16.0	28.0	0.12	175
Senegalais	17.3	29.2	0.13	200
TY339612	14.8	26.0	0.10	160

5.4 Analyse des erreurs et robustesse

5.4.1 Cas difficiles

— Variétés sombres ou homogènes.

— Conditions d'acquisition défavorables (luminosité faible, ombres).

— Grains atypiques (fissures, tâches).

5.4.2 Robustesse en conditions réelles

MAE augmenté de seulement 0.5 à 1 min grâce à l'augmentation de données, conforme à [4].

5.5 Discussion critique et implications

5.5.1 Comparaison avec l'état de l'art

Les CNN compacts sur images RGB rivalisent avec des méthodes hyperspectrales ($r > 0.87$) [1] et sont économiques et portables. Quantification et pruning réduisent la taille mémoire de plus de 70 % [20, 17].

5.5.2 Limites

— Dataset limité à 56 sousvariétés.

— Expérimentations contraintes par GPU Google Colab.

— Généralisation sur microcontrôleurs non encore validée.

5.5.3 Perspectives

— Enrichir le dataset (conditions extrêmes, angles variés).

— Hybridation RGB + hyperspectral.

— Déploiement TinyML optimisé sur microcontrôleurs.

— Étude en environnement industriel réel.

5.6 Synthèse

Les CNN personnalisés démontrent que des modèles compacts et quantifiés permettent une estimation précise du temps de cuisson sur images RGB, offrant une solution portable et fiable.

6 Conclusion et perspectives

6.1 Synthèse des contributions

Ce travail a exploré l'application du *Tiny Machine Learning* (TinyML) à la prédiction du temps de cuisson des haricots, en utilisant des images comme entrée. Les principales contributions sont les suivantes :

1. **Collecte et constitution du jeu de données.** Un jeu de données original a été constitué, comprenant des images annotées de 56 variétés de haricots, chaque image étant associée à un temps de cuisson mesuré expérimentalement.
2. **Prétraitement et normalisation.** Un protocole rigoureux de prétraitement a été mis en place, incluant la redimension des images en $224 \times 224 \times 3$, l'augmentation de données pour atténuer les déséquilibres inter-variétés, et la standardisation des valeurs pour stabiliser l'apprentissage.
3. **Conception et entraînement de modèles adaptés au TinyML.** Plusieurs architectures ont été testées et comparées, notamment des modèles légers dérivés de MobileNetV2, EfficientNetB0, NASNetMobile, ainsi qu'un *Convolutional Neural Network* (CNN) personnalisé. Les expérimentations ont montré que le CNN conçu sur mesure, entraîné intégralement depuis zéro, offrait le meilleur compromis entre précision et complexité computationnelle.
4. **Évaluation approfondie.** Les performances des modèles ont été mesurées à l'aide d'indicateurs standards (MAE, RMSE, R^2, MAPE). L'analyse a mis en évidence une corrélation satisfaisante entre les temps de cuisson prédits et les valeurs réelles, démontrant la faisabilité d'un tel système dans un cadre TinyML.
5. **Quantification et portabilité.** Les modèles ont été compressés et convertis en formats TensorFlow Lite afin de réduire leur empreinte mémoire et leur consommation énergétique, ouvrant ainsi la voie à une intégration dans des dispositifs embarqués tels que des microcontrôleurs ARM Cortex-M.

6.2 Bilan critique

6.2.1 Points forts

— **Originalité du jeu de données.** La constitution d'un jeu de données original constitue une valeur ajoutée significative pour la recherche appliquée à l'agroalimentaire.

— **Adaptation au TinyML.** Le choix d'architectures légères et l'adaptation des modèles aux contraintes du TinyML démontrent une prise en compte pragmatique des réalités matérielles.

— **Rigueur méthodologique.** L'analyse croisée par plusieurs métriques confère une robustesse méthodologique aux conclusions.

6.2.2 Limites

— **Taille du jeu de données.** La taille du jeu de données reste relativement modeste au regard de la variabilité inter-variétés, ce qui peut limiter la généralisation du modèle.

— **Sensibilité des mesures.** La mesure du temps de cuisson, bien que rigoureuse, reste sensible aux conditions expérimentales (qualité de l'eau, dureté initiale des grains, altitude, etc.), introduisant une part de bruit difficilement contrôlable.

— **Prédiction univariée.** La prédiction reste basée uniquement sur des images statiques, sans prise en compte d'autres variables physico-chimiques susceptibles d'améliorer la précision.

6.3 Perspectives

Les perspectives de ce travail ouvrent plusieurs pistes de recherche et d'applications pratiques :

1. **Extension du jeu de données.** L'enrichissement du jeu de données, tant en termes de variétés que de conditions de cuisson, permettrait d'améliorer la robustesse et la généralisabilité des modèles.
2. **Fusion multimodale.** L'intégration d'autres sources d'information (mesures spectroscopiques, texture, teneur en humidité, composition chimique) pourrait compléter l'information visuelle et réduire l'incertitude prédictive.
3. **Optimisation avancée pour TinyML.** Des techniques plus poussées de compression (quantification dynamique, distillation de connaissances, *pruning*) pourraient être explorées pour réduire davantage la taille mémoire et l'énergie consommée par le modèle.
4. **Déploiement réel.** La mise en œuvre pratique dans des environnements agroalimentaires, par exemple via des prototypes de capteurs intelligents intégrant des caméras embarquées, permettrait de valider les performances en conditions réelles et d'identifier les besoins d'adaptation industrielle.
5. **Approches explicatives.** L'intégration de techniques d'explicabilité (*Explainable AI*, XAI) permettrait de mieux comprendre les caractéristiques visuelles exploitées par le modèle pour établir ses prédictions, favorisant l'acceptabilité et la confiance dans des environnements critiques comme l'agroalimentaire.

6.4 Conclusion générale

En définitive, ce mémoire démontre la pertinence d'appliquer des approches de **vision par ordinateur et d'apprentissage profond** à une problématique agroalimentaire concrète : la prédiction du temps de cuisson des haricots. En combinant rigueur méthodologique, optimisation pour environnements contraints et analyse critique, cette recherche ouvre des perspectives tant scientifiques qu'industrielles.
Elle contribue à la fois à la littérature émergente sur le TinyML et à l'amélioration potentielle des pratiques agroalimentaires, notamment en facilitant l'optimisation des temps et coûts de cuisson. Les limites identifiées et les pistes proposées posent les bases de travaux futurs, qui pourront renforcer la robustesse, l'explicabilité et l'applicabilité des solutions développées. Ainsi, ce travail constitue une étape significative vers l'intégration de l'intelligence artificielle embarquée au service de la transformation et de la valorisation des produits agricoles.

Bibliographie

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[17] 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.

[18] 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.

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

[20] 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.

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

[22] 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.

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

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

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

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

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

[28] 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.

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

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

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

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

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

[34] 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).

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

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

[37] 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.

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

[39] 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.

[40] Mateusz Buda, Atsuto Maki et Maciej A Mazurowski. « A systematic study of the class imbalance problem in convolutional neural networks ». In : *Neural Networks* 106 (2018), p. 249-259.

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

Déclaration d'authenticité

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