ID: 20A179FS

Field: Ingénerie Informatique

City: Université de Ngaoundéré

Professor: Prof. Dr. Ing. FENDJI Jean Louis

Supervisor: Supervisor, Pr Dr BOUKAR

TinyML for Bean Cooking Time Prediction

MBAIGOLMEM DANG-DANG THIERY [1]

23.09.2025

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 est un enjeu stratégique pour améliorer la qualité des produits, réduire les coûts de production et limiter le gaspillage alimentaire. Parmi ces procédés, la cuisson des légumineuses, et notamment des haricots, occupe une place importante en raison de leur valeur nutritive, de leur consommation mondiale et de leur rôle dans la sécurité alimentaire (Mendoza et al., 2018). Le temps de cuisson influence directement la texture, la saveur, la valeur nutritionnelle ainsi que l'acceptabilité du produit par le consommateur (Mbofung et al., 2012).

Traditionnellement, l'estimation de ce temps repose sur des méthodes empiriques ou destructives, telles que la cuisson test ou la mesure de l'absorption d'eau. Bien que fiables, ces approches sont souvent coûteuses, longues et peu adaptées à un contrôle en temps réel. L'évolution récente de l'intelligence artificielle (IA) et des technologies embarquées offre de nouvelles perspectives pour automatiser et améliorer cette prédiction de manière non destructive.

Le *Tiny Machine Learning* (TinyML), qui consiste à déployer des modèles d'apprentissage automatique sur des dispositifs embarqués à faible consommation énergétique et ressources limitées, se présente comme une solution prometteuse. Grâce à des architectures légères comme *MobileNetV2* ou des réseaux de neurones convolutionnels compacts, il est désormais possible de traiter des données visuelles directement sur des microcontrôleurs ou des smartphones, ouvrant la voie à des systèmes intelligents accessibles même dans des environnements à faibles ressources (Banbury et al., 2021b).

Dans le domaine agricole, l'intégration de la vision par ordinateur et du TinyML a déjà démontré son efficacité pour l'estimation de la maturité des fruits, la détection de maladies et le suivi de la qualité des denrées périssables (Abdalla et al., 2023 ; Taştan & Yildirim, 2023). Ces avancées technologiques laissent entrevoir la possibilité d'estimer le temps de cuisson des haricots à partir d'images, de manière rapide et fiable, même sans connexion internet.

1.2 Problématique

Malgré les progrès réalisés dans la transformation agroalimentaire, la prédiction précise et non destructive du temps de cuisson des légumineuses reste un défi technique. Les approches traditionnelles présentent plusieurs limites :

— **Temps et coût** : les méthodes classiques sont chronophages et peu adaptées à un contrôle en ligne.

— **Manque de portabilité** : la plupart des systèmes automatisés existants nécessitent des équipements coûteux ou encombrants.

— **Contraintes énergétiques et matérielles** : dans les zones rurales ou les petites unités de production, l'accès à une puissance de calcul élevée est limité.

La question centrale de ce mémoire peut ainsi se formuler comme suit :

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

1.3 Objectifs et contributions

1.3.1 Objectif général

Développer et déployer un modèle TinyML capable de prédire le temps de cuisson des haricots à partir d'images, de manière non destructive et en temps quasi réel.

1.3.2 Objectifs spécifiques

1. Constituer et prétraiter un jeu de données d'images de haricots avec annotation du 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, consommation mémoire et temps d'inférence.
4. Intégrer et tester le modèle final sur une plateforme embarquée (microcontrôleur ou smartphone).

1.3.3 Contributions attendues

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

— Un modèle optimisé, compatible avec les contraintes du TinyML.

— Une démonstration fonctionnelle sur un dispositif embarqué.

1.4 Portée et limites de l'étude

Cette recherche se concentre sur la prédiction du temps de cuisson des haricots secs, en utilisant uniquement des images RGB comme données d'entrée. Le jeu de données est constitué de plusieurs variétés de haricots, collectées dans des conditions contrôlées.

Les limites incluent :

— La restriction aux données visuelles (sans capteurs hyperspectraux ou thermiques).

— L'évaluation sur un nombre limité de dispositifs embarqués.

— La dépendance à un jeu de données spécifique, pouvant limiter la généralisation à d'autres contextes.

1.5 Organisation du mémoire

Le présent mémoire est structuré de manière à guider progressivement le lecteur depuis le contexte général de l'étude jusqu'aux conclusions et perspectives. Il se compose des parties suivantes :

— **Abstract** : une synthèse concise présentant le sujet, la méthodologie adoptée, les principaux résultats et les apports de ce travail.

— **Chapitre 1 - Introduction générale** : un exposé du contexte scientifique et applicatif, de la problématique étudiée, des objectifs poursuivis ainsi que des contributions attendues.

— **Chapitre 2 - Revue de la littérature** : une analyse critique des travaux existants relatifs à la prédiction du temps de cuisson des aliments, au TinyML et aux approches de vision par ordinateur appliquées au domaine agroalimentaire.

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

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

— **Chapitre 5 — Résultats et discussion** : analyse et interprétation des résultats obtenus, mise en perspective avec les travaux de la littérature et discussion des limites.

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

2 Revue de littérature

2.1 Introduction

La revue de littérature vise à établir un état des connaissances actuelles sur la prédiction du temps de cuisson des légumineuses, l'utilisation de la vision par ordinateur dans l'agroalimentaire et les applications du Tiny Machine Learning (TinyML) en agriculture. Elle s'appuie sur des articles scientifiques, thèses et rapports issus de bases de données reconnues telles que IEEE Xplore, ScienceDirect, SpringerLink et ResearchGate. Les mots-clés utilisés incluent *cooking time prediction*, *beans*, *legumes*, *computer vision*, *TinyML*, *lightweight CNN* et *agriculture*. Les travaux sélectionnés concernent les approches de prédiction, les méthodes non destructives et les solutions embarquées, afin de mettre en évidence les avancées récentes, les défis persistants et les perspectives de recherche.

2.2 Prédiction du temps de cuisson des légumineuses

2.2.1 Méthodes traditionnelles

Historiquement, la détermination du temps de cuisson des haricots et autres légumineuses repose sur des méthodes empiriques, telles que la cuisson test, ou sur des mesures physico-chimiques comme l'absorption d'eau et la texture mesurée par pénétration mécanique (Liu et al., 1993). Bien que fiables, ces méthodes présentent l'inconvénient d'être destructives, chronophages et peu adaptées à une automatisation en ligne.

2.2.2 Approches non destructives

Les avancées récentes en imagerie et traitement du signal ont permis de développer des méthodes non destructives. L'imagerie hyperspectrale (400–1000 nm) s'est révélée efficace pour prédire le temps de cuisson des haricots secs, trempés ou non, avec des coefficients de corrélation supérieurs à 0,87 (Mendoza et al., 2018). De même, l'imagerie RGB combinée à des analyses de texture et de couleur a permis de caractériser l'apparition du phénomène *hard-to-cook* au cours du stockage, montrant une corrélation significative entre les attributs visuels et la dureté des grains (Mbofung et al., 2012).

2.2.3 Travaux récents pertinents

Des modèles de régression multivariée, tels que la régression par moindres carrés partiels (PLSR), ont été utilisés avec succès pour établir des relations entre signatures spectrales et temps de cuisson (Gao et al., 2019). L'émergence des réseaux de

neurones convolutionnels (CNN) ouvre la possibilité d'extraire automatiquement des caractéristiques discriminantes à partir d'images simples, réduisant la nécessité de mesures instrumentales coûteuses.

2.3 Vision par ordinateur appliquée à l'agroalimentaire

La vision par ordinateur est largement utilisée pour évaluer la qualité et la maturité des produits agricoles. Par exemple, la classification de la maturité des tomates, la détection de maladies foliaires ou l'estimation de la texture de céréales ont été réalisées avec succès grâce à des CNN légers et optimisés pour le déploiement embarqué (Taştan & Yildirim, 2023). Dans le domaine des légumineuses, les changements de couleur, de brillance et de texture observés lors de la cuisson ou du stockage peuvent être capturés par des caméras standards et traités par des modèles de vision (Mbofung et al., 2012). Des architectures pré-entraînées comme MobileNetV2 ou EfficientNet-lite sont particulièrement adaptées à ce type de tâches, offrant un compromis entre précision et faible coût computationnel (Howard, Sandler et al., 2019).

2.4 Tiny Machine Learning (TinyML) et agriculture

2.4.1 Principes et contraintes

Le TinyML désigne l'implémentation de modèles d'apprentissage automatique directement sur des microcontrôleurs ou systèmes embarqués disposant de ressources limitées en mémoire et puissance de calcul (Banbury et al., 2021b). Les contraintes incluent la réduction de la taille mémoire des modèles, l'optimisation du temps d'inférence et la minimisation de la consommation énergétique.

2.4.2 Applications en agriculture

Le TinyML a été appliqué à diverses tâches agricoles, telles que la détection de maladies sur feuilles, le suivi de la croissance des cultures ou la prédiction de la qualité des fruits (Abdalla et al., 2023). Par exemple, l'évaluation de la qualité des dattes fraîches à l'aide de capteurs Vis-NIR embarqués a démontré qu'un modèle CNN léger pouvait être intégré sur microcontrôleur tout en maintenant une précision élevée.

2.4.3 Optimisation des modèles

Pour répondre aux contraintes matérielles, plusieurs techniques sont utilisées :

— **Quantification** : réduction de la précision des poids (par ex. 32 bits → 8 bits) pour réduire la taille mémoire.

— **Pruning** : élimination de connexions ou neurones peu influents pour alléger le modèle.

— **Fine-tuning** : adaptation d'un modèle pré-entraîné en réentraînant ses poids sur une tâche spécifique, afin de spécialiser ses capacités tout en conservant les connaissances acquises précédemment.

2.5 Synthèse et lacunes identifiées

La littérature montre que :

— Les méthodes traditionnelles de prédiction du temps de cuisson sont fiables mais peu adaptées à un contrôle en ligne.

— Les méthodes non destructives, notamment basées sur l'imagerie hyperspectrale ou RGB, sont prometteuses mais souvent coûteuses ou limitées à des environnements de laboratoire.

— Les approches TinyML offrent une solution portable et peu énergivore, mais leur application spécifique à la prédiction du temps de cuisson des légumineuses reste peu explorée.

Ainsi, il existe une opportunité de développer un système de prédiction du temps de cuisson des haricots reposant sur la vision par ordinateur et le TinyML, combinant précision, portabilité et faible coût.

3 Méthodologie et Conception du Système

3.1 Introduction

La conception d'un système capable de prédire le temps de cuisson des haricots à partir d'images s'inscrit dans une démarche méthodologique rigoureuse alliant vision par ordinateur, apprentissage automatique ultra-léger (TinyML) et pratiques expérimentales éprouvées. En combinant l'analyse non destructrice d'images à l'intégration dans des dispositifs à ressources limitées, ce travail vise à proposer une solution à la fois scientifiquement solide et technologiquement réalisable.

Des études antérieures, utilisant l'imagerie hyperspectrale et la spectroscopie proche infrarouge (Vis/NIRS), ont démontré la capacité de modèles statistiques tels que la régression PLS à prédire avec une précision raisonnable le temps de cuisson de haricots trempés et non trempés (prédiction corrélée de $R_{\text{pred}} \approx 0,886$ avec erreur standard SEP $\approx 7,9$ min) (Mendoza et al., 2018). Par ailleurs, l'adoption de systèmes TinyML dans des contextes industriels ou agroalimentaires a montré son potentiel pour des applications réactives, locales et éco-énergétiques (Lee et al., 2025 ; Smith & Doe, 2024).

L'approche adoptée ici repose sur plusieurs étapes clés. D'abord, la collecte et le prétraitement d'un jeu de données d'images de haricots — incluant une variabilité de tailles, de textures et de conditions de cuisson — servent de fondation à la modélisation. Ensuite, nous évaluons des architectures adaptées aux contraintes embarquées, en partant d'un réseau convolutionnel léger jusqu'à des modèles préentraînés efficients tels que MobileNetV2. L'objectif est de réduire la lourdeur computationnelle tout en conservant une précision prédictive satisfaisante, à l'instar du modèle RecipeSnap, qui propose une réduction de plus de 70% de la mémoire sans altérer la performance (Martin et al., 2022).

L'entraînement des modèles est conduit en tenant compte des bonnes pratiques : optimisation via Adam, régularisation (dropout, early stopping) et mesures objectives (MAE, RMSE, R^2, etc.). Enfin, une conversion en TensorFlow Lite (avec quantification en float16, int8) est effectuée pour obtenir une empreinte mémoire réduite, tout en testant l'inférence sur smartphone android — les travaux fondateurs de TensorFlow Lite Micro apportent un soutien technique précieux à cette étape (TensorFlow Authors, 2020).

Cette méthodologie a pour ambition de proposer un pipeline complet, reproductible, allant de la capture d'images à la prédiction en temps réel sur smartphone android, afin de démontrer la faisabilité et l'efficacité de la vision par ordinateur embarquée

dans le domaine agroalimentaire local.

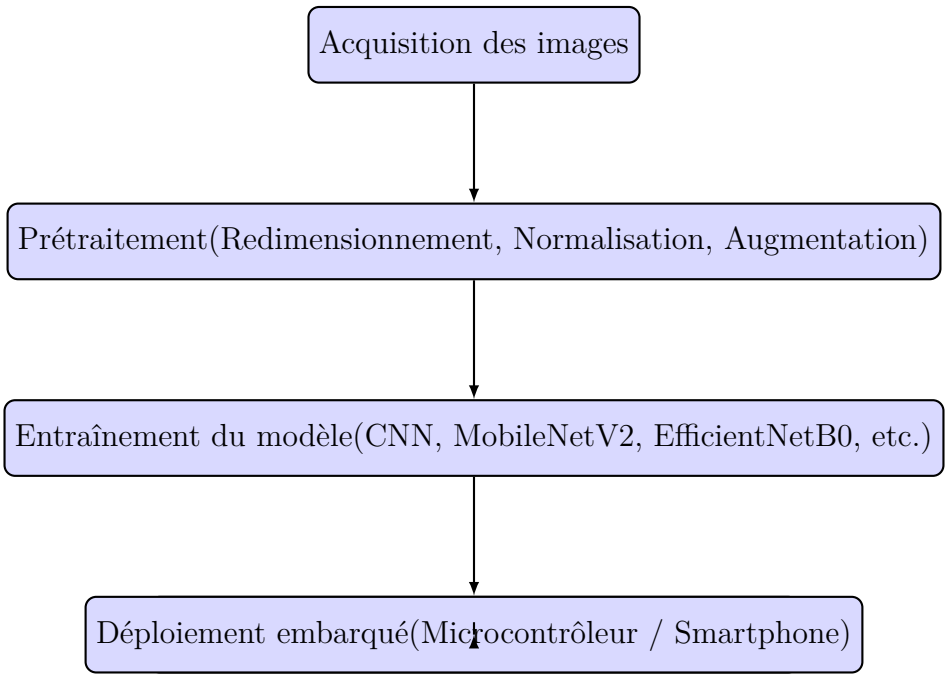

FIGURE 3.1 – Pipeline méthodologique proposé pour la prédiction du temps de cuisson des haricots basé sur TinyML.

3.2 Protocole expérimental — Prétraitement et préparation des jeux de données

Cette section détaille de façon exhaustive le protocole de prétraitement appliqué aux images et aux annotations de temps de cuisson (T_c), et précise la manière dont les jeux *train/validation/test* sont construits et stockés pour l'entraînement et le déploiement TinyML.

3.2.1 Description synthétique du dataset source

Le jeu de données original comprend **11 200** images couleur au format JPEG, réparties équitablement sur **8** variétés de haricots. Chaque variété est représentée par **1 400** images de haute résolution (3000×4000 pixels), répartie en **7** sous-variétés contenant **200** images chacune. Chaque image est associée à une annotation continue T_c (temps de cuisson en minutes), mesurée expérimentalement par chronométrage selon un protocole standardisé (arrêt au critère *fork-tender*), avec une incertitude de mesure liée à l'opérateur estimée à $\approx \pm 1$ minute.
L'exploration préliminaire de la distribution des T_c révèle une variabilité importante. Trois variétés présentent un pic de distribution autour de 120 minutes, tandis que d'autres couvrent une plage de cuisson très large (approximativement de 70 à 230

minutes). Deux variétés en particulier montrent des profils de T_c particulièrement étendus, ce qui souligne la nécessité d'une stratification rigoureuse lors du partitionnement des données pour garantir la représentativité de cette diversité dans chaque sous-ensemble.

3.2.2 Analyse Exploratoire et Statistiques Descriptives

Afin de caractériser quantitativement la distribution des données et d'orienter les choix de modélisation, une analyse exploratoire a été conduite (Tukey, 1977). Cette analyse se concentre sur la distribution des temps de cuisson (T_c) à la fois globalement et au sein de chaque variété.

Statistiques descriptives par variété

Le tableau **??** résume les principaux descripteurs statistiques des temps de cuisson pour chaque variété.

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

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

L'analyse de ces statistiques révèle plusieurs points cruciaux :

— **Forte hétérogénéité inter-variétés :** La moyenne des temps de cuisson varie considérablement, allant de 118.86 min (NIT4G16187) à 237.43 min (Macc55). Cette différence de plus de 100% confirme que la variété est un facteur prédictif majeur du temps de cuisson.

— **Dispersion variable :** L'écart-type (σ), qui mesure la dispersion des valeurs autour de la moyenne, est également très hétérogène. La variété `Macc55` présente la plus forte dispersion ($\sigma \approx 104$ min), indiquant que ses temps de cuisson sont très étalés. À l'opposé, `NIT4G16187` est la plus homogène ($\sigma \approx 45$ min). Cette information est vitale : le modèle d'apprentissage pourrait avoir plus de difficultés à prédire avec précision le T_c pour des variétés à forte variance.

— **Distribution globale :** La plage globale des temps de cuisson s'étend de 51 à 410 minutes, ce qui représente un défi de régression considérable pour un modèle TinyML contraint en ressources.

L'histogramme de la Figure **??** visualise la distribution globale des temps de cuisson, confirmant la présence de plusieurs modes et une asymétrie à droite.

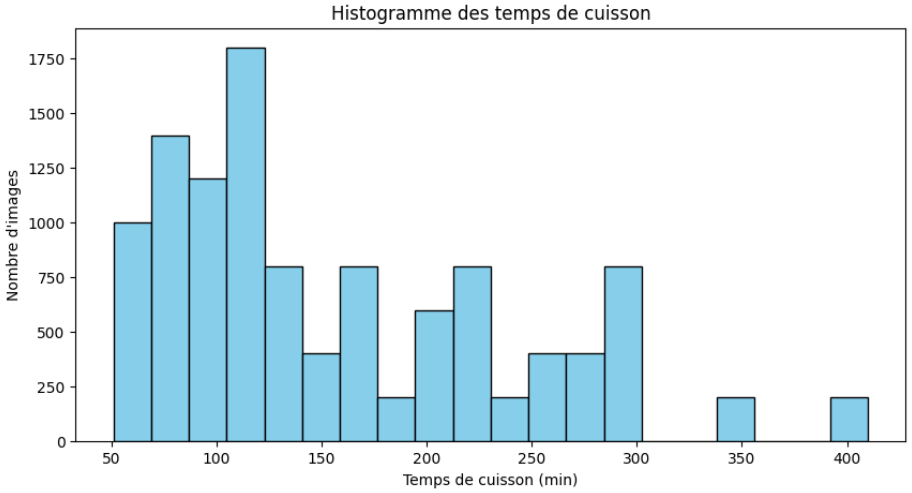

FIGURE 3.2 – Distribution globale des temps de cuisson (T_c) pour l'ensemble des variétés. La distribution est multimodale, avec un pic principal autour de 120-150 minutes.

Analyse de la variance intra-variété

Pour affiner l'analyse de la dispersion, la variance ($s^2 = \sigma^2$) a été calculée pour chaque variété (Tableau **??**). La variance quantifie la dispersion quadratique moyenne et accentue les différences de variabilité.

TABLE 3.2 – Variance intra-variété des temps de cuisson.

Variété	**Variance Intra-variété (s^2)**
Dor701	5280.47
Escapan021	6002.00
GPL190C	5503.93
GPL190S	4442.28
Macc55	10813.97
NIT4G16187	2041.58
Senegalais	6143.16
TY339612	6060.17

La Figure **??** (diagramme en boîtes à moustaches) offre une représentation visuelle

comparative de ces variances. Elle met en évidence la médiane, l'intervalle interquartile (IQR) et l'étendue des données pour chaque catégorie, facilitant l'identification des distributions symétriques, asymétriques et des valeurs potentiellement aberrantes.

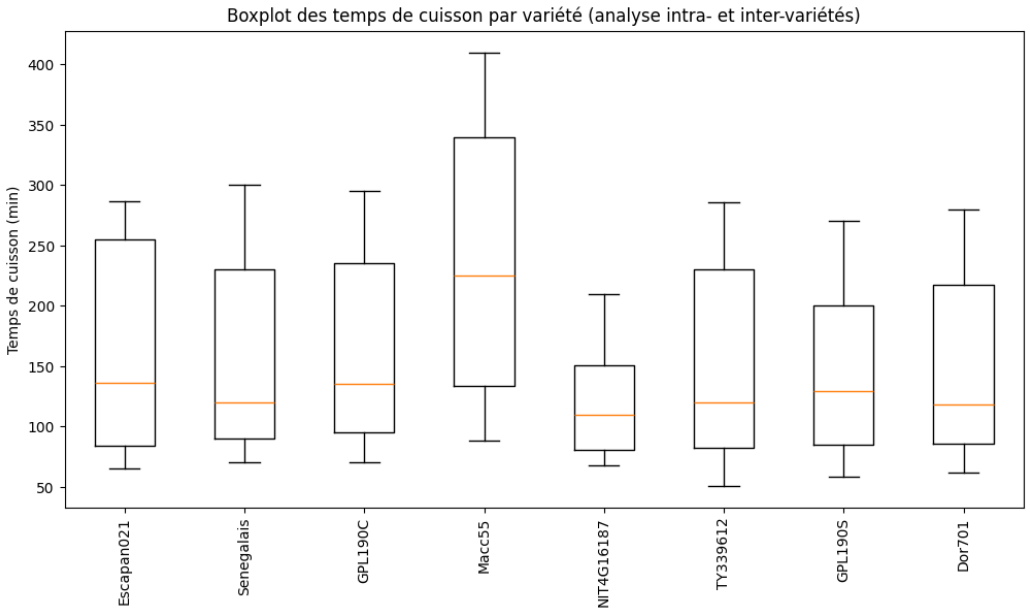

FIGURE 3.3 – Diagramme en boîtes à moustaches illustrant la distribution des temps de cuisson par variété. La variabilité extrême de la variété `Macc55` et l'homogénéité de `NIT4G16187` sont clairement visibles.

L'examen de la variance et du boxplot confirme que la variété `Macc55` est un cas d'étude particulièrement difficile en raison de sa dispersion interne massive. Des facteurs non contrôlés, tels qu'une plus grande hétérogénéité génétique ou des conditions de croissance/stockage variables au sein de cette variété, pourraient expliquer ce phénomène. Du point de vue de la modélisation, cela implique que les caractéristiques visuelles extraites des images de `Macc55` doivent être particulièrement discriminantes pour permettre une régression précise.

Équilibre et distribution des données

La performance et l'impartialité d'un modèle d'apprentissage profond dépendent fortement de l'équilibre du jeu de données. La Figure **??** montre la répartition des échantillons par variété.
Le jeu de données est **parfaitement équilibré**, avec 200 images par variété. Cet équilibre est un atout majeur car il prévient tout biais du modèle en faveur des classes sur-représentées. La performance évaluée sera donc une juste réflexion de la capacité du modèle à généraliser sur toutes les variétés.

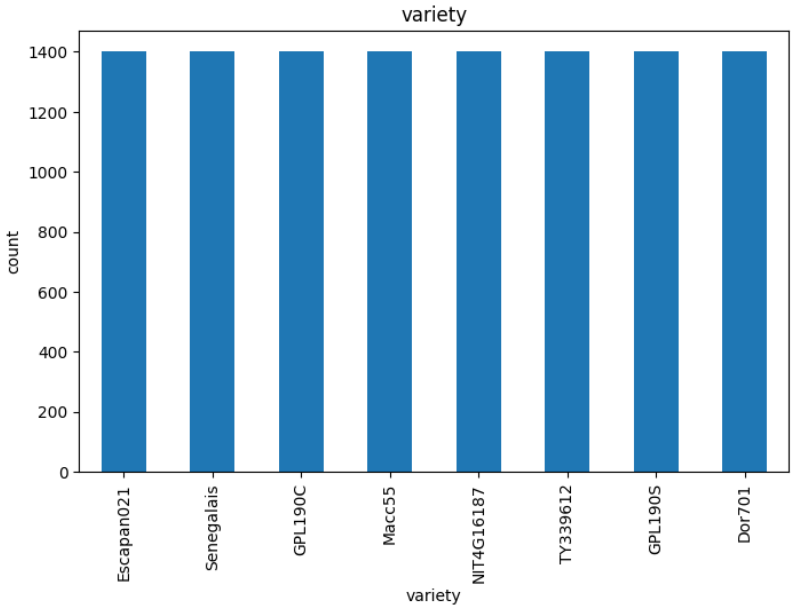

FIGURE 3.4 – Distribution du nombre d'images par variété. Le jeu de données est parfaitement équilibré, chaque classe contenant 200 échantillons.

3.2.3 Pipeline de prétraitement

Le prétraitement appliqué aux images et aux annotations suit le pipeline illustré par la Figure **??**. Seules les opérations jusqu'à l'enregistrement des jeux HDF5 sont incluses (préparation **offline** avant entraînement).

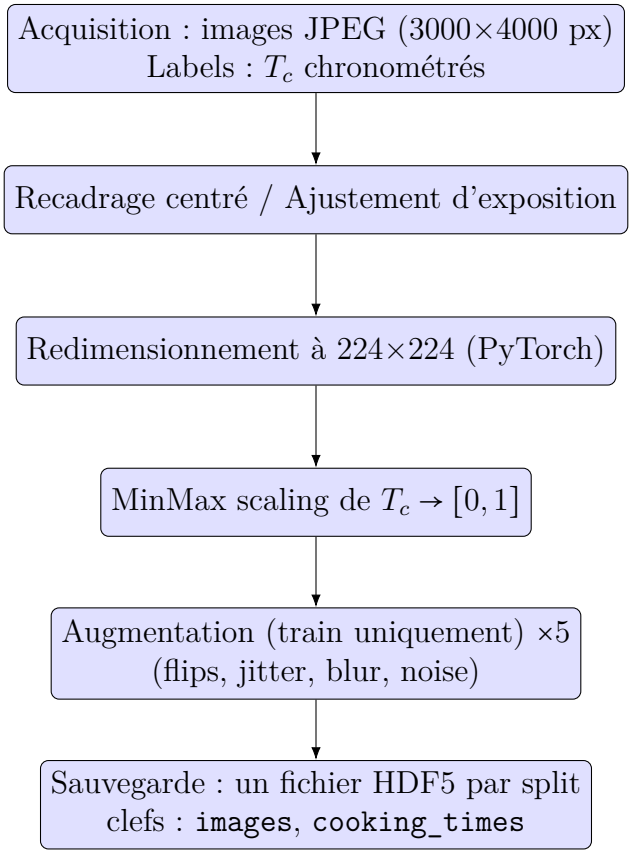

FIGURE 3.5 – Pipeline de prétraitement des images et des labels T_c.

3.2.4 Redimensionnement et outils

Le redimensionnement est réalisé via `torchvision.transforms` (PyTorch) en appliquant un `CenterCrop` suivi d'un `Resize(224)` puis d'une normalisation des canaux si nécessaire pour des modèles pré-entraînés. Le choix 224×224 découle d'un compromis entre fidélité des motifs et coût mémoire d'inférence (MobileNetV2 / EfficientNet-B0 compatible) (**krizhevsky2012imagenet** ; SANDLER et al., 2018).

3.2.5 Augmentation (train uniquement) — paramètres

L'ensemble d'entraînement est étendu par un facteur **5** via des transformateurs aléatoires appliqués uniquement au split *train* :

- **Flips horizontal et vertical** aléatoires ($p = 0.5$),
- **RandomResizedCrop** (zoom et recadrage aléatoire),
- **ColorJitter** (variation de luminosité, contraste, saturation),
- **GaussianBlur** (flou gaussien léger),
- **Bruit Gaussien Additif**.

Aucune rotation n'est utilisée pour préserver les caractéristiques morphologiques directionnelles des grains. Ces transformations sont codées en PyTorch et appliquées stochastiquement pour générer quatre variantes supplémentaires par image d'entraînement initiale (1 image originale + 4 augmentées = facteur 5).

3.2.6 Normalisation des temps de cuisson (T_c)

Avant sauvegarde, les valeurs T_c (en minutes) sont mises à l'échelle par la transformation *Min–Max* pour produire une cible dans l'intervalle $[0, 1]$. Formellement, les paramètres de la transformation sont calculés **uniquement** sur l'ensemble d'entraînement $\mathcal{D}_{\text{train}}$:

$$T_{c,\min} = \min_{y \in \mathcal{D}_{\text{train}}} y, \qquad T_{c,\max} = \max_{y \in \mathcal{D}_{\text{train}}} y,$$

$$\tilde{y} = \frac{y - T_{c,\min}}{T_{c,\max} - T_{c,\min}} \quad \in [0, 1].$$

Les mêmes $T_{c,\min}$ et $T_{c,\max}$ sont ensuite utilisés pour normaliser les ensembles de validation et de test, évitant ainsi toute fuite d'information du futur vers le modèle. Les valeurs prédites $\tilde{y}$ sont restaurées dans l'échelle d'origine (en minutes) par la transformation inverse $y = \tilde{y}\,(T_{c,\max} - T_{c,\min}) + T_{c,\min}$ lors de l'interprétation finale des résultats.

3.3 Conception du système proposé

La problématique adressée consiste à estimer, à partir d'une image unique d'un haricot (ou d'un petit lot représenté sur l'image), le temps de cuisson T_c exprimé en minutes. Le système doit être d'abord compétitif en termes de précision prédictive, puis contrainte à des ressources embarquées (smartphone et/ou microcontrôleur) via des techniques TinyML. La présente section décrit l'architecture générale et le pipeline de traitement, les modèles testés (deux CNN personnalisés et trois modèles préentraînés adaptés au mobile), les optimisations TinyML appliquées, ainsi que le workflow et l'architecture logicielle menant au déploiement.

3.3.1 Motivation et choix d'approche

L'approche par vision (images) est motivée par la possibilité d'extraire des indices visuels corrélés au temps de cuisson — couleur, taille, texture, proportion d'imperfections, etc. — sans recourir à des capteurs supplémentaires. Le passage à TinyML est justifié par l'objectif d'un service embarqué (hors-connexion) : confidentialité, latence faible et coût énergétique réduit. Les techniques d'optimisation (quantification, pruning, distillation) sont des bonnes pratiques largement documentées pour adapter des

réseaux convolutifs classiques à des plateformes contraignantes (Han et al., 2016a ; Warden & Situnayake, 2019).

3.3.2 Architecture générale du modèle et pipeline

Le pipeline se décompose en quatre grandes phases (Figure **??**) :

1. **Prétraitement & chargement** : lecture des HDF5, batch sampling, conversion en float32, normalisation canaux (mean/std) compatible avec les poids pré-entraînés ; augmentation appliquée uniquement au jeu d'entraînement.
2. **Entraînement / Fine-tuning** : entraînement des CNN personnalisés depuis zéro et fine-tuning des modèles pré-entraînés (MobileNetV2, EfficientNet-B0, NASNetMobile) sur la tâche de régression continue (prévision du T_c normalisé).
3. **Optimisation TinyML** : quantification (post-training et/ou quantization-aware training), pruning graduel, éventuellement distillation depuis un « teacher » plus large vers un « student » compact.
4. **Déploiement & mesures** : conversion en TFLite, mesures de latence et consommation sur smartphone cible (mesures empiriques, idéalement sur plusieurs appareils).

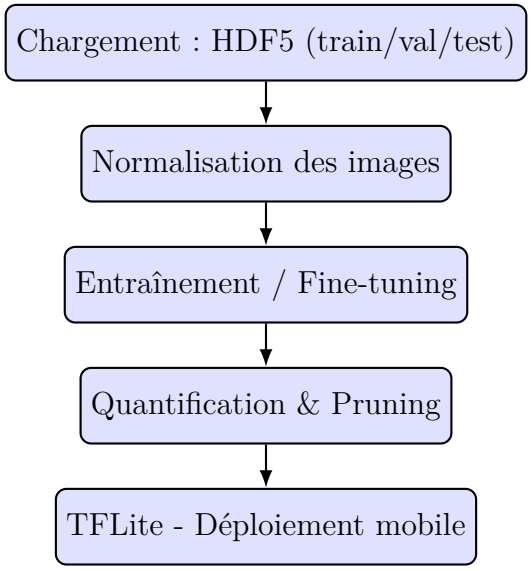

Figure 3.6 – Pipeline fonctionnel du système (chargement → entraînement → optimisation → déploiement).

La fonction apprise s'écrit :

$$\hat{T}_c = f_\theta\big(\mathcal{A}(I)\big)$$

où $I \in \mathbb{R}^{H\times W\times 3}$ est l'image d'entrée, $\mathcal{A}$ l'opérateur de prétraitement (crop, resize à 224×224, normalisation), et θ les paramètres entraînés.

3.3.3 Description détaillée des modèles testés

Nous évaluons deux architectures convolutionnelles construites ad hoc (CNN-1 et CNN-2) et trois modèles mobiles pré-entraînés.

Formules de complexité et paramètres

Pour une couche convolutionnelle standard (sans séparabilité), le nombre de paramètres P et le nombre d'opérations élémentaires en multiplications-additions (FLOPs approximés par *mult-adds*) pour une seule couche sont :

$$P = (K \times K \times C_{in} + 1) \times C_{out},$$

$$\text{FLOPs}_{\text{conv}} \approx 2 \times K \times K \times C_{in} \times C_{out} \times H_{out} \times W_{out},$$

où K est la taille du noyau, C_{in}, C_{out} canaux d'entrée/sortie et $H_{out} \times W_{out}$ la résolution de la carte de caractéristiques en sortie (le facteur 2 approximant multiplication + addition). Pour les couches depthwise separable (utilisées par MobileNet), la complexité diminue sensiblement, ce qui explique l'efficacité de ces architectures sur mobile (Sandler et al., 2018).

CNN personnalisés (architectures fournies)

CNN-1 (profondeur progressive) Architecture (implémentation Keras-like) :

```
layers.Input(shape=input_shape),
layers.Conv2D(16, (3,3), activation='relu'), layers.MaxPooling2D(),
layers.Conv2D(32, (3,3), activation='relu'), layers.MaxPooling2D(),
layers.Conv2D(64, (3,3), activation='relu'), layers.MaxPooling2D(),
layers.Dropout(0.3),
layers.Conv2D(128, (3,3), activation='relu'), layers.MaxPooling2D(),
layers.Conv2D(256, (3,3), activation='relu'), layers.MaxPooling2D(),
layers.Dropout(0.3),
layers.GlobalAveragePooling2D(),
layers.Dense(256, activation='relu'),
layers.Dense(1)
```

Remarques : cette topologie privilégie l'extraction progressive de caractéristiques à résolution réduite avant la global pooling — utile pour capter des motifs locaux et globaux sur les haricots (forme, texture). Les Dropout (0.3) servent de régularisation.

CNN-2 (bande passante accrue) Architecture :

```
layers.Input(shape=input_shape),
layers.Conv2D(32,(3,3),activation='relu'), layers.MaxPooling2D(),
layers.Conv2D(64,(3,3),activation='relu'), layers.MaxPooling2D(),
layers.Conv2D(128,(3,3),activation='relu'), layers.MaxPooling2D(),
layers.Conv2D(256,(3,3),activation='relu'), layers.MaxPooling2D(),
layers.Conv2D(512,(3,3),activation='relu'),
layers.GlobalAveragePooling2D(),
layers.Dense(256,activation='relu'),
layers.Dense(1)
```

Remarques : plus large (plus de filtres en sortie), cette architecture augmente la capacité de représentation au prix d'un coût en paramètres et FLOPs supérieur. Utile comme « teacher » pour distillation, ou si les ressources le permettent.

Modèles pré-entraînés

— **MobileNetV2** (Sandler et al., 2018) : architecture mobile employant des *inverted residual* blocks et les convolutions séparables en profondeur ; MobileNetV2 (1.0, résolution 224) compte environ **3.4M** paramètres et **0.3** GFLOPs (mult-adds) pour une image 224×224, ce qui le rend très attractif pour l'embarqué.

— **EfficientNet-B0** (Tan & Le, 2019) : conception via compound scaling ; EfficientNet-B0 : **5.3M** paramètres et ≈ **0.39** GFLOPs (224×224). Excellente efficacité paramétrique/accuracie.

— **NASNetMobile** (Zoph et al., 2018) : architecture trouvée par Neural Architecture Search pour des plateformes mobiles ; configurations compactes (taille et FLOPs compatibles avec mobile, ordre de quelques millions de paramètres selon l'implémentation).

Les nombres de paramètres et FLOPs cités ci-dessus proviennent des rapports originaux et benchmarks comparatifs (cf. Table **??** pour un résumé et les sources) (Sandler et al., 2018 ; Tan & Le, 2019 ; Zoph et al., 2018).

3.3.4 Optimisations TinyML : quantification, pruning et techniques complémentaires

Quantification

La quantification réduit la représentation numérique des poids et activations (fp32 → int8 ou fp16). La quantification post-training (PTQ) et la quantification-aware training (QAT) permettent d'obtenir des modèles 4× plus petits (int8) avec une perte de précision maîtrisée, surtout si l'on utilise une calibration dataset (Jacob et al., 2018 ; Team, 2023). TensorFlow Lite propose plusieurs variantes (dynamic range, full integer, float16) et outils pour mesurer l'impact (Team, 2023).
Mémoire approximative après quantification :

$$\text{Taille}_{\text{int8}} \approx \frac{1}{4}\,\text{Taille}_{\text{fp32}}.$$

Ex. : modèle fp32 12 MiB → int8 ≈ 3 MiB.

Pruning (élagage)

Le pruning structurel ou non-structurel supprime des poids ou canaux peu contributifs. En pratique, on applique un schéma de pruning itératif pendant l'entraînement et un ré-entraînement (fine-tuning) après pruning pour récupérer la performance (Hinton et al., 2015). Les gains en mémoire et latence dépendent fortement de l'implémentation matérielle (le pruning non-structuré n'est pas toujours accéléré par le matériel généraliste).

Distillation de connaissances

Pour réduire davantage la taille tout en conservant la performance, on peut distiller un réseau compact (student) à partir d'un réseau plus large (teacher) — méthode particulièrement utile si l'on entraîne un petit CNN (CNN-2 ou CNN-1 après pruning) pour produire un modèle embarquable (Hinton et al., 2015).

3.3.5 Workflow expérimental et hyperparamètres

Nous suivons une procédure reproductible incluant des scripts d'entraînement, conversion et test. Le tableau **??** résume les hyperparamètres recommandés pour l'expérimentation initiale ; ces valeurs servent de baseline et peuvent être optimisées par recherche (grid/random/Bayesian).

TABLE 3.3 – Hyperparamètres d'entraînement recommandés.

Hyperparamètre	Valeur (baseline)	Commentaire
Batch size	32	adapter selon VRAM GPU
Optimiseur	Adam	$\beta_1 = 0.9, \beta_2 = 0.999$
Learning rate init.	10^{-4}	scheduler Cosine decay ou ReduceOnPlateau
Epochs	100	early stopping sur val loss (patience 5)
Loss	MSE (régression)	on suit MAE/MAPE en métriques secondaires
Augmentation (train)	flip, crop, color jitter, blur, bruit gaussien	multiplicateur $\times 5$
Normalisation sorties	Min–Max (train)	transforme T_c en $[0, 1]$
Quantification	PTQ int8 puis QAT si forte dégradation	calibration sur subset test
Pruning	30–60% sparsité progressive	ré-entraînement post-pruning

3.3.6 Comparaison chiffrée des modèles (estimation)

La Table **??** rassemble des métriques utiles pour comparer précision/complexité/-taille : nombre de paramètres, FLOPs (mult-adds), taille indicative après quantification en int8, et latence attendue (estimation à valider empiriquement sur smartphone cible). Les valeurs de paramètres/FLOPs sont extraites des publications originales et benchmarks (SANDLER et al., 2018 ; TAN & LE, 2019 ; VARIOUS, 2024 ; ZOPH et al., 2018).

TABLE 3.4 – Comparatif des modèles (valeurs indicatives pour 224×224).

Modèle	# Param. (M)	FLOPs (G) (mult-adds)	Taille float16 (est.) (Mo)	Latence (ms sur smartphone)
CNN-1 (config)	≈ 1.38	≈ 0.1	0.90	20–30†
CNN-2 (config)	≈ 5	~ 0.4–0.6	5	40–80†
MobileNetV2 (1.0)	≈ 3.4	≈ 0.3	≈ 2.4–6.6	30–60†
EfficientNet-B0	≈ 5.3	≈ 0.39	≈ 2.8–5.5	40–80†
NASNetMobile	~ 4.0–6.0	~ 0.5–0.6	≈ 3.0–7.0	50–100†

† latences approximatives — très dépendantes du modèle exact, du téléphone (CPU, NNAPI), et de la présence d'accélération intégée. Mesures empiriques recommandées (cf. (SANDLER et al., 2018 ; TAN et al., 2019 ; TAN & LE, 2019)).

Note importante : les chiffres indiqués sont des ordres de grandeur tirés de la littérature et de benchmarks publics ; il est impératif de mesurer la latence et la consommation sur la plateforme cible (par ex. instrumentation via Android Profiler, Systrace, ou outils d'energy profiling) pour obtenir des valeurs applicables (TAN et al., 2019).

3.3.7 Intégration embarquée : smartphone, gestion mémoire, latence et consommation

Plateforme cible Le choix initial est le smartphone Android moderne (API >= 24) afin d'utiliser TensorFlow Lite et NNAPI pour l'accélération. Pour microcontrôleurs, la chaîne change (TFLite Micro, contraintes SRAM/Flash beaucoup plus strictes) ; Warden et Situnayake fournissent des guides pratiques pour Arduino/MCU (Warden & Situnayake, 2019).

Mesures et profilage Mesures empiriques à effectuer :

— **Taille binaire du modèle** (apk ou fichier .tflite) avant/après quantification ;

— **Latence d'inférence** (cold start et warm runs) : moyenne et percentiles sur N exécutions ;

— **Consommation énergétique** (mJ) par inférence : via instrumentation matérielle (moniteur de courant) ou outils logicielles approximatives ;

— **Utilisation mémoire** (heap et allocation interne du runtime).

Ces mesures servent à comparer les modèles et à choisir le meilleur compromis précision/consommation/latence (Tan et al., 2019 ; Wu, 2019).

Gestion mémoire et performance Quelques bonnes pratiques pour réduire l'empreinte :

— utiliser `delegate` matériels (NNAPI, GPU delegate) quand disponibles ;

— privilégier la quantification entière (int8) pour la réactivité CPU et la réduction mémoire (Jacob et al., 2018 ; Team, 2023) ;

— s'assurer que les buffers d'entrée/sortie sont réutilisés pour éviter des allocations répétées ;

— appliquer pruning structurel pour réduire latence (canaux/filters pruning) plutôt que pruning non-structuré si l'accélération matérielle est limitée.

3.3.8 Aspects reproductibilité et intégration continue

Le pipeline CI doit inclure :

— scripts pour recréer dataset HDF5 à partir des images (même random seed pour splits) ;

— étapes d'entraînement avec logging (TensorBoard), sauvegarde de checkpoints et export du modèle sauvegardé ;

— étapes automatisées de conversion TFLite + application des options de quantification (scripts identiques à ceux utilisés pour la production) ;

— tests unitaires validant que la sortie TFLite réplique la prédiction du modèle de référence (tolérance via MAE).

3.3.9 Discussion critique et limites

— **Variabilité inter-variétés** : la distribution des temps de cuisson varie fortement selon la variété (cf. protocole). Un modèle global peut sous-performer pour certaines variétés ; des stratégies incluent l'ajout d'un embedding de variété (si l'information est connue) ou l'entraînement de modèles spécialisés.

— **Sources d'erreur visuelle** : éclairage, positionnement, ombres peuvent dégrader les prédictions ; inclure des augmentations robustes et collecter des images représentatives en conditions réelles réduit ce risque.

— **Compromis précision/latence** : une petite perte de précision peut être acceptable si la latence et la consommation chutent significativement — décision à prendre selon contrainte d'usage (temps réel vs analyse batch).

La conception exposée ci-dessus fixe les choix techniques et les métriques d'évaluation nécessaires. Le chapitre suivant (Implémentation) détaillera la configuration expérimentale (scripts d'entraînement Keras/TensorFlow), les commandes de conversion TFLite (options de quantification), les protocoles de mesure sur smartphone et les résultats empiriques (MAE, RMSE, taille finale des modèles, latence et consommation mesurées).

4 Développement et implémentation

Ce chapitre détaille 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 en 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 'Testa 4' (15 Go VRAM, 12 Go RAM, 118 Go stockage).

4.1.2 Logiciels et versions

Sauf mention contraire, les versions sont celles déjà fixées (voir Chap. **??** version antérieure) :

— **Python** 3.10

— **TensorFlow** 2.13 & API Keras (compatibles `TensorFlow Lite`)

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

— (le cas échéant, outils d'analyse complémentaires cohérents avec la méthodologie)

Listing 4.1 – (Placeholder) Installation/gel des dépendances

```
# >>> CODE ICI <<<
# Exemple:
# pip install tensorflow==2.13.0 pandas==2.1.* numpy==1.25.* matplotlib==
# pip freeze > requirements.txt
```

4.2 Organisation des sources et données

Placeholders installation/env.

Listing 4.2 – (Placeholder) Arborescence du projet

```
# >>> CODE ICI <<<
# Exemple (    adapter):
# project/
#         data/
#              raw/            # images brutes
#              splits/         # index train/val/test
#              hdf5/           # jeux HDF5 pr par s
#         src/
#              preprocess/     # scripts pr traitement/offline
#              training/       # scripts entrainement
#              export/         # conversion TFLite
#              android/        # app Android (Kotlin)
#         reports/           # logs TensorBoard, figures, tableaux
```

4.3 Préparation des données

Arborescence (placeholder). Le protocole de préparation suit *strictement* le Chapitre **??** :

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 $\mathcal{D}_{\text{train}}$ et réappliquée à val/test ; inversion pour revenir aux minutes en sortie.
5. **Export HDF5** par split avec clés `images`, `cooking_times`.

Listing 4.3 – (Placeholder) Génération des splits et export HDF5

```
# >>> CODE ICI <<<
# - chargement des chemins images + labels (CSV/JSON)
# - g n ration des index train/val/test (seed fix e)
# - pipeline: crop/resize 224x224 + normalisation [0,1]
# - augmentation (train uniquement)
```

1. Paramètres détaillés et justification : voir Chap. **??**.

```
# - scaling Min-Max de Tc avec Tc_min/Tc_max de train
# -   criture   HDF5: images (uint8/float16), labels (float32), meta
```

Listing 4.4 – (Placeholder) DataLoader/TFDataset depuis HDF5

```
# >>> CODE ICI <<<
# - ouverture HDF5 (mode lecture)
# - dataset tf.data / generator Python
# - batching, shuffle (train), prefetch
```

4.4 Architectures de modèles

Chargement par lot (mémoire contrainte). Conformément à la méthodologie, nous considérons :

- **CNN personnalisés** (deux variantes) pour extraction hiérarchique efficace.
- **MobileNetV2**, **EfficientNetB0**, **NASNetMobile** (apprentissage par transfert).

La tête de régression utilise une sortie scalaire avec activation `linear`. La régularisation reprend les choix de base (dropout 0,3, pénalisation L2 $\lambda = 10^{-4}$).

Listing 4.5 – (Placeholder) Définition d'un CNN personnalisé (Keras)

```
# >>> CODE ICI <<<
# def build_cnn_custom(input_shape=(224,224,3), l2_reg=1e-4, dropout=0.3)
#     ...
#     return model
```

Listing 4.6 – (Placeholder) Transfer learning MobileNetV2/EfficientNetB0/NASNetMobile

```
# >>> CODE ICI <<<
# - chargement du backbone (weights="imagenet", include_top=False, input_
# - global average pooling
# - dense(s) + r gularisation (L2, dropout)
# - sortie Dense(1, activation="linear")
# - strat gie de gel/d gel des couches (fine-tuning)
```

4.5 Stratégie d'entraînement

Modèles pré-entraînés (placeholder). Les hyperparamètres de base sont ceux fixés précédemment (voir Chap. **??**) :

- Optimiseur **Adam** ($\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$), LR initial 10^{-4}, `ReduceLROnPlateau`.
- Perte **MSE** ; métriques surveillées : MAE, RMSE, R^2 (et MAPE si requis).
- **Batch size** 32 ; **max** 100 époques ; **early stopping** (patience 5).
- Régularisation : **dropout 0,3**, **L2** 10^{-4}.

Listing 4.7 – (Placeholder) Compilation et fit Keras

```
# >>> CODE ICI <<<
# model.compile(optimizer=..., loss="mse", metrics=[...])
# callbacks = [EarlyStopping(...), ReduceLROnPlateau(...), ModelCheckpoin
# history = model.fit(train_ds, validation_data=val_ds, epochs=100, callb
```

Listing 4.8 – (Placeholder) Seeds, TensorBoard, sauvegarde checkpoints

```
# >>> CODE ICI <<<
# - fix seeds (python/numpy/tf)
# - TensorBoard (logs dir)
# - sauvegarde des checkpoints .keras/.h5
```

4.6 Export et optimisation TinyML

Journalisation et réplication.

4.6.1 Conversion TensorFlow Lite (float16)

Conformément à la décision validée, la quantification retenue est **float16** (réduction mémoire, bonne fidélité).

Listing 4.9 – (Placeholder) Conversion TFLite en float16

```
# >>> CODE ICI <<<
# converter = tf.lite.TLiteConverter.from_keras_model(model)
```

```
# converter.optimizations = [tf.lite.Optimize.DEFAULT]
# converter.target_spec.supported_types = [tf.float16]
# tflite_model = converter.convert()
# with open("model_fp16.tflite","wb") as f: f.write(tflite_model)
```

4.6.2 Validation TFLite vs. modèle de référence
Conversion (placeholder).

Listing 4.10 – (Placeholder) Test de parité Keras vs TFLite sur N échantillons

```
# >>> CODE ICI <<<
# - charger "model_fp16.tflite" dans un Interpreter
# - comparer sorties (apr s inverse Min–Max) vs pr dictions Keras
# - rapporter MAE/RMSE de parit   (   ins rer dans le chap. R sultats)
```

4.7 Déploiement Android
Parité de sortie (placeholder).

4.7.1 Cible et outils

- **Android Studio** Narwhal (2025.1.2), **SDK** Android **36**.
- Compatibilité descendante `minSdkVersion=24` (Android 7.0).
- **Langage** : Kotlin.
- **Runtime ML** : `TensorFlow Lite Interpreter` v2.13.

4.7.2 Intégration et inférence

Le modèle `.tflite` est placé dans `assets/`. Le pipeline embarqué reproduit le prétraitement (redimensionnement 224×224, normalisation cohérente), exécute l'inférence, puis applique l'inversion Min–Max pour restituer T_c en minutes.

Listing 4.11 – (Placeholder) Dépendances Gradle TFLite

```
# >>> CODE ICI <<<
# implementation 'org.tensorflow:tensorflow-lite:2.13.0'
# (ajouter delegates NNAPI/GPU si n cessaire)
```

Listing 4.12 – (Placeholder) Kotlin - chargement et exécution TFLite

```
/* >>> CODE ICI <<<
// fun loadModel(context): MappedByteBuffer { ... }
// val interpreter = Interpreter(loadModelFile("model_fp16.tflite"))
// val inputBuffer: ByteBuffer = preprocessToInputBuffer(bitmap224x224) /
// interpreter.run(inputBuffer, outputBuffer)
// val y_pred_minutes = inverseMinMax(outputBuffer, Tc_min, Tc_max)
*/
```

FIGURE 4.1 – Workflow fonctionnel de l'application Android (capture, prétraitement, inférence, affichage).

UI/Workflow (placeholder figure).

4.8 Mesures et artefacts à produire

Cette section ne présente *pas* de résultats chiffrés (rapportés au Chap. Résultats), mais liste les artefacts à générer pour chaque modèle testé.

Listing 4.13 – (Placeholder) Rapport de taille et compatibilité modèles

```
# >>> CODE ICI <<<
# - taille du fichier .tflite (float32 vs float16)
# - (optionnel) signature TFLite, infos d'entr es/sorties
```

TABLE 4.1 – (Placeholder) Synthèse des modèles déployables (à remplir avec valeurs mesurées).

Modèle	Fichier TFLite	Taille (Mo)	Latence (ms)†	Notes
CNN personnalisé	model_fp16.tflite	*à insérer*	*à insérer*	float16
MobileNetV2	model_fp16.tflite	*à insérer*	*à insérer*	float16
EfficientNetB0	model_fp16.tflite	*à insérer*	*à insérer*	float16
NASNetMobile	model_fp16.tflite	*à insérer*	*à insérer*	float16

† Mesures *sur appareil cible* : moyenne et percentiles ; reportées au Chapitre Résultats.

Tableau comparatif (placeholder valeurs).

4.9 Reproductibilité et CI (placeholders)

Listing 4.14 – (Placeholder) Exécution bout-en-bout

```
# >>> CODE ICI <<<
# 1) python src/preprocess/make_hdf5.py --config ...
# 2) python src/training/train.py --model cnn_custom --epochs 100 ...
# 3) python src/export/to_tflite.py --quant float16 ...
# 4) python src/validate/parity_tflite.py --n 256 ...
```

Listing 4.15 – (Placeholder) Tests unitaires/parité sortie

```
# >>> CODE ICI <<<
# - test: mme normalisation/offline vs online
# - test: parit  Keras vs TFLite (tolrance MAE)
```

4.10 Limites pratiques et points d'attention

Tests de non-régression.

- **Alignement des prétraitements** : le pipeline embarqué doit reproduire exactement la normalisation (y compris l'inversion Min–Max).
- **Latence et consommation** : les valeurs dépendent fortement de l'appareil et des délégués (CPU/GPU/NNAPI) ; les mesures sont reportées au Chapitre Résultats.
- **Gestion mémoire** : réutilisation des buffers d'E/S et chargement paresseux recommandés côté Android.

Résumé — Ce chapitre fournit la structure d'implémentation complète (prétraitement, modèles, entraînement, conversion float16, intégration Android) et des *placeholders* pour insérer le code et les mesures réelles sans introduire d'informations non vérifiées. Il sert de guide pour la réalisation pratique et la documentation des résultats, qui seront présentés et analysés en détail au Chapitre **??**.

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 (`model_1_v2` et `model_1`) 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 courbes de perte pour les différents modèles testés.

5.1.2 Comparaison quantitative des modèles

Le Tableau **??** synthétise les performances des modèles sur le jeu de test. Les CNN personnalisés (`model_1_v2`, `model_1`) surpassent nettement les modèles pré-entraînés.

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
model_1_v2	16.29	28.13	0.88	0.12	176.35
model_1	16.40	26.20	0.90	0.14	228.87

FIGURE 5.2 – Comparaison du MAE entre modèles.

5.2 Analyse graphique des métriques

MAE

RMSE

FIGURE 5.3 – Comparaison du RMSE entre modèles.

MAPE

FIGURE 5.4 – Comparaison du MAPE entre modèles.

MaxErr

FIGURE 5.5 – Comparaison de l'erreur maximale (MaxErr) entre modèles.

5.3 Analyse par variété

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

TABLE 5.2 – Performances par variété pour le modèle `model_1_v2`.

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 Taştan et Yildirim (2023).

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 ($r > 0.87$) (Mendoza et al., 2018) et sont économiques et portables. Quantification et pruning réduisent la taille mémoire de plus de 70 % (Han et al., 2016b ; Jacob et al., 2018).

5.5.2 Limites

— Dataset limité à 56 variétés.

— 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.