UCLouvain

epl

École polytechnique de Louvain

Application mobile hors-ligne pour la mesure automatique du diamètre des grumes

Auteurs: **Benuel Burla, Guillaume Nzakimuena**
Promoteur: **Pierre Schaus**
Lecteurs: **Augustin Crespin, Laurent De Plaen, Mathias de Schietere de Lophem**
Année académique 2025–2026
Master [60] en sciences informatiques

Résumé

Ce mémoire présente la conception et l'évaluation d'une application mobile hors-ligne destinée à mesurer automatiquement le diamètre de grumes à partir d'une photographie terrain. Le travail s'inscrit dans le contexte du projet *Wood Tracking System* (dWTS), développé pour améliorer la traçabilité du bois au Guyana. L'objectif est de proposer un prototype capable d'assister la mesure dendrométrique à partir d'un smartphone, sans dépendre d'une connexion réseau.

La méthode repose sur la segmentation d'instance de deux éléments : la section visible de la grume et un tag physique de référence. Plusieurs architectures YOLO ont été comparées afin d'identifier un compromis entre précision de segmentation, stabilité face aux conditions de prise de vue et coût d'inférence. Les expériences portent notamment sur la résolution d'entrée, le choix de l'architecture, l'effet des augmentations, l'évaluation finale sur un *test set* réel et la robustesse à des variations de cadrage.

Les résultats montrent que YOLO11n constitue le meilleur compromis pour l'intégration mobile. Il obtient la meilleure performance globale sur le *test set* tout en conservant une taille et une latence compatibles avec une exécution locale. Le pipeline complet de mesure atteint une erreur absolue moyenne de 0.81 cm sur le diamètre moyen, avec une erreur relative moyenne de 1.97 %.

Ces résultats sont encourageants compte tenu de la taille limitée du *dataset*. Ils montrent la faisabilité d'une mesure automatique hors-ligne, tout en soulignant la nécessité d'une validation terrain plus large avant un déploiement opérationnel.

Table des matières

1 Introduction

1.1 Contexte

Dans le cadre du projet *Wood Tracking System* (dWTS), déployé au Guyana avec le soutien de l'Agence Française de Développement, PEPPS développe un système numérique de traçabilité du bois, depuis l'abattage jusqu'à l'exportation.

Sur le terrain, les agents du *Forest Monitoring Division* contrôlent les grumes abattues à l'aide d'un tag physique unique fixé sur la section du bois. Le diamètre de la grume fait partie des informations importantes à collecter, car il intervient dans l'estimation du volume et dans le suivi commercial du bois.

Aujourd'hui, cette mesure reste principalement manuelle. Elle peut être lente, dépendre de l'opérateur et être sensible aux erreurs de lecture ou de retranscription. Le contexte forestier impose également des contraintes fortes : faible connectivité, luminosité variable, qualité d'image parfois moyenne, occlusions et diversité des formes de grumes.

1.2 Objectifs

L'objectif de ce mémoire est de concevoir et d'évaluer un prototype d'application mobile hors-ligne capable d'estimer automatiquement le diamètre apparent d'une grume à partir d'une photographie.

Le système doit permettre de capturer une image d'une grume portant un tag, d'identifier automatiquement ce tag comme référence dimensionnelle, de détecter la section visible de la grume, puis de calculer une estimation du diamètre par traitement local de l'image. L'application doit également afficher le résultat afin de permettre sa validation ou sa correction par l'utilisateur.

Ce travail cherche donc à valider l'ensemble du pipeline, depuis la prise de photo jusqu'à la mesure finale en centimètres, tout en respectant les contraintes d'un usage hors connexion sur smartphone Android.

1.3 Code et reproductibilité

Le projet est séparé en deux dépôts GitHub :

— application mobile hors-ligne : https://github.com/GuillaumeZaki/TFE_Grumes_App_Frontend.git

— modèles YOLO, entraînements et scripts d'analyse : https://github.com/Benuel/TFE_Grumes_YOLO

Cette séparation distingue la partie applicative, destinée à l'intégration dans l'écosystème dWTS, de la partie expérimentale utilisée pour entraîner, comparer et analyser les modèles.

2 État de l'art

L'automatisation de la mesure du diamètre des grumes se situe à l'intersection de la dendrométrie forestière, de la vision par ordinateur et de l'apprentissage profond. L'objectif n'est pas seulement de détecter la présence d'une grume dans une image, mais d'extraire une information géométrique fiable permettant d'estimer une dimension réelle.

Cette section présente d'abord les fondements dendrométriques de la mesure des grumes et les limites des méthodes manuelles. Elle introduit ensuite les solutions numériques existantes, depuis les systèmes 3D jusqu'aux approches par image. Enfin, elle regroupe les méthodes d'intelligence artificielle utilisées en vision par ordinateur, en mettant l'accent sur la segmentation d'instance, les architectures YOLO, le fine-tuning et la data augmentation.

2.1 Mesure forestière et contraintes dendrométriques

La mesure du diamètre constitue une opération fondamentale en dendrométrie forestière. Elle intervient directement dans l'estimation du volume des bois ronds, notamment à travers des formules classiques de cubage telles que celles de Huber, Smalian ou Newton [1]. Dans le cas des grumes abattues, le diamètre mesuré à l'extrémité de la pièce de bois permet d'alimenter les calculs de volume, de valeur commerciale et de traçabilité.

Cependant, une grume réelle s'écarte rarement d'une géométrie cylindrique idéale. Les sections transversales peuvent présenter des ovalisations, des excentricités, des fissures, des déformations mécaniques, des irrégularités de coupe ou encore des variations d'épaisseur d'écorce. Rondeux rappelle également que l'écorce constitue une composante spécifique à prendre en compte dans les mesures forestières, car les pratiques commerciales peuvent exiger des mesures sous écorce [1].

Dans ce contexte, la mesure manuelle du diamètre présente plusieurs limites :

— elle dépend fortement du positionnement de l'outil et de l'opérateur ;
— elle est sensible aux conventions de mesure, notamment pour les formes non circulaires ;
— elle devient plus difficile dans le cas de grumes fissurées, excentriques ou partiellement masquées ;
— elle prend du temps lorsqu'un grand nombre de grumes doit être contrôlé.

Ces limites motivent le développement de méthodes automatisées capables d'améliorer simultanément la rapidité, la reproductibilité et la traçabilité des mesures.

2.2 Numérisation de la mesure des grumes

Plusieurs approches technologiques ont été étudiées pour automatiser la mesure des grumes. Les méthodes laser, les scanners personnels et les systèmes de reconstruction 3D permettent d'obtenir des représentations géométriques détaillées des pièces de bois. De Miguel-Díez et al. montrent par exemple qu'un scanner laser personnel combiné à une technologie SLAM peut fournir des estimations volumétriques plus précises que plusieurs formules classiques de cubage [2].

Niță et Borz étudient également une solution mobile basée sur smartphone pour collecter des informations géométriques et estimer le volume de grumes. Leur travail

montre l'intérêt des plateformes mobiles pour la mesure forestière, notamment en raison de leur accessibilité, de leur mobilité et de leur coût réduit par rapport à des systèmes spécialisés [3].

Ces méthodes confirment l'intérêt de la numérisation des mesures forestières, mais elles ne répondent pas toujours aux contraintes du projet dWTS. Les systèmes fondés sur des scanners ou des reconstructions 3D peuvent nécessiter du matériel spécifique, une acquisition plus longue, un post-traitement plus lourd ou une expertise technique plus importante. Dans le cadre d'une application destinée à des agents de terrain, une solution fondée sur une simple image smartphone est plus facile à déployer massivement.

La revue comparative de Sandvik et al. montre d'ailleurs que les approches par caméra sont de plus en plus présentes dans les travaux sur le *scaling* et le *grading* des grumes, tandis que les modèles d'apprentissage profond sont devenus particulièrement pertinents pour les tâches de segmentation et de mesure en contexte forestier [4].

2.3 Vision par ordinateur classique : intérêt et limites

Avant l'usage massif de l'apprentissage profond, la détection des grumes reposait principalement sur des méthodes déterministes de traitement d'image. Les techniques les plus courantes incluent :

— la détection de contours ;
— la transformée de Hough circulaire ;
— le seuillage couleur ou intensité ;
— les opérations morphologiques ;
— les méthodes de type watershed ou graph-cut.

Ces approches peuvent fonctionner lorsque les conditions d'acquisition sont contrôlées : éclairage stable, contraste élevé, grumes bien séparées et formes proches du cercle. Elles deviennent cependant fragiles en conditions réelles. Les images forestières présentent souvent des ombres fortes, des textures complexes, des fissures, des traces de coupe, des variations de couleur, de l'herbe, de la boue, des occlusions et des grumes partiellement superposées.

Le rapport TreeTrace met en évidence les difficultés liées à l'analyse automatique d'images d'extrémités de bois ronds, notamment lorsque les textures, les marques de coupe et les conditions lumineuses perturbent les contours [5]. Zheng et al. soulignent également que les méthodes classiques deviennent particulièrement limitées dans le cas de piles denses ou de grumes partiellement occultées : les algorithmes basés sur la circularité ou les contours peuvent alors produire des erreurs de segmentation, des faux positifs ou des omissions [6].

Ces limites rendent nécessaire l'utilisation de modèles capables d'apprendre automatiquement des représentations visuelles robustes, plutôt que de dépendre exclusivement de règles géométriques ou colorimétriques définies manuellement.

2.4 Apprentissage automatique et vision profonde

Les limites des méthodes déterministes ont progressivement conduit au développement d'approches fondées sur l'apprentissage automatique, puis sur l'apprentissage profond. Dans ce cadre, le modèle n'est plus uniquement programmé à partir

de règles visuelles fixes : il apprend directement, à partir d'exemples annotés, les caractéristiques utiles à la tâche.

Dans le contexte de ce mémoire, l'apprentissage profond est pertinent pour trois raisons principales. Premièrement, les grumes présentent une forte variabilité visuelle : formes irrégulières, textures différentes selon les essences, fissures, écorce, ombres et traces de peinture. Deuxièmement, le tag de référence est un petit objet dont la détection doit rester fiable malgré les variations de distance et d'éclairage. Troisièmement, le modèle doit rester suffisamment léger pour être ensuite intégré dans une application mobile hors-ligne.

2.4.1 Réseaux convolutifs et extraction automatique de caractéristiques

Les réseaux de neurones convolutifs, ou *Convolutional Neural Networks* (CNN), constituent la base de nombreux modèles modernes de vision par ordinateur. Contrairement aux méthodes classiques, un CNN apprend automatiquement des caractéristiques visuelles à partir des données. Les premières couches détectent généralement des motifs simples, comme des bords, des textures ou des contrastes locaux, tandis que les couches plus profondes apprennent des structures plus abstraites et spécifiques à la tâche [7].

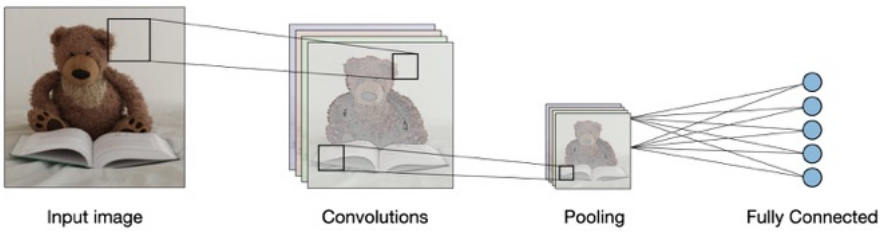


FIGURE 1 – Architecture générale d'un réseau convolutif : extraction progressive de caractéristiques visuelles à partir de l'image d'entrée [8].

Dans le contexte des grumes, cette capacité d'apprentissage est particulièrement importante. Les objets à détecter ne présentent pas une apparence stable : les essences de bois peuvent varier, les textures internes diffèrent d'une grume à l'autre, les coupes peuvent être propres ou abîmées, et les conditions lumineuses dépendent fortement du terrain.

Carratù et al. montrent que les CNN peuvent être appliqués à des images de bois pour automatiser des tâches auparavant réalisées par inspection visuelle humaine, comme l'évaluation de la qualité dans des piles de bois. Leur approche obtient des résultats prometteurs avec un F1-score de 0.89 pour la meilleure architecture testée [9]. Mazzochin et al. explorent également la segmentation et le comptage automatique de grumes à partir d'images, ce qui confirme l'intérêt croissant des modèles d'apprentissage profond pour les opérations forestières [10], [11].

Ces travaux ne résolvent pas exactement le même problème que ce mémoire, mais ils confirment que les modèles profonds sont capables d'apprendre des indices visuels utiles dans des scènes de bois complexes.

2.4.2 Détection d'objet et segmentation d'instance

Une première manière d'automatiser la mesure consiste à détecter la grume par une boîte englobante. La détection d'objet répond à la question : « quel objet est présent et où se trouve-t-il ? ». Elle fournit généralement une classe et une boîte rectangulaire autour de l'objet.

Cette représentation est suffisante pour des tâches de localisation ou de comptage, mais elle reste limitée pour une application métrologique. Une boîte englobante contient nécessairement du fond, de l'écorce, des zones extérieures au contour réel et parfois des éléments voisins. Elle ne décrit donc pas précisément la géométrie de la section transversale.

La segmentation d'instance fournit au contraire un masque pixel par pixel pour chaque objet détecté. Elle permet de distinguer plusieurs grumes dans une même image et d'extraire un contour exploitable pour des calculs géométriques.

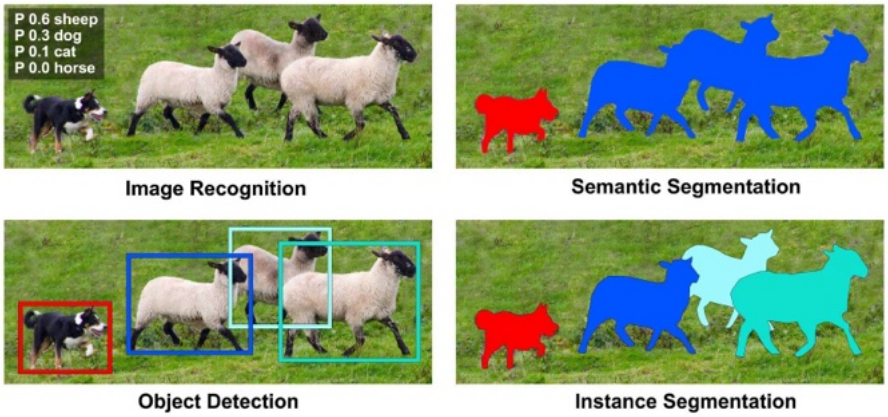


FIGURE 2 – Comparaison entre reconnaissance d'image, détection d'objets, segmentation sémantique et segmentation d'instance. Pour une mesure de diamètre, la segmentation d'instance fournit une information géométrique plus précise qu'une boîte englobante.

Li et al. proposent une méthode de segmentation d'instance rapide pour les faces de grumes dans des piles de bois. Leur méthode combine prédiction de masques, boîtes rectangulaires et apprentissage métrique afin de mieux séparer les zones de chevauchement entre grumes. Les auteurs rapportent une précision d'extraction de masque de 91.2% et une vitesse de 50.2 FPS, ce qui montre l'intérêt de la segmentation d'instance pour automatiser la mesure du diamètre [12].

Zheng et al. proposent également une méthode fondée sur YOLACT pour la segmentation de bois transporté par camion. Leur approche améliore la précision de segmentation et réduit les erreurs de faux positifs et d'omissions dans des piles denses, en combinant une architecture de segmentation rapide avec des mécanismes d'attention et des améliorations de régression des boîtes [6].

Ces travaux confirment que la segmentation d'instance est plus adaptée qu'une simple détection d'objet lorsque l'objectif est d'extraire une information géométrique fiable.

2.4.3 Mesure automatique du diamètre par vision

Plusieurs travaux récents traitent plus directement de la mesure automatique du diamètre des grumes à partir d'images.

Lu et al. proposent une architecture combinant YOLOv3 pour la détection des faces de grumes et DeepLabv3+ pour la segmentation. Leur système utilise également un AprilTag comme référence dimensionnelle et comme aide à l'estimation de la position de la caméra, afin de réduire les effets de perspective. Les auteurs rapportent une mAP de 97.28% pour la détection, un mIoU de 92.22% pour la segmentation, une précision moyenne de mesure de 96.26%, ainsi qu'une erreur de 0.73% sur le diamètre individuel dans leur test forestier [13].

Carratù et al. proposent de leur côté une méthode de mesure du diamètre fondée sur la vision et l'apprentissage profond, montrant l'intérêt d'une référence de taille connue pour convertir des mesures en pixels vers des dimensions réelles [14]. Ces travaux sont proches de la logique retenue dans ce mémoire : détecter ou segmenter la grume, détecter une référence physique, puis convertir la mesure image en mesure métrique.

Cependant, le contexte dWTS présente des contraintes spécifiques. L'application doit fonctionner hors-ligne, sur smartphone, avec un tag physique déjà utilisé dans le processus de traçabilité. Le problème n'est donc pas uniquement de mesurer un diamètre dans une image, mais de concevoir un pipeline complet compatible avec un usage terrain, une PWA et une inférence locale.

2.4.4 Architectures modernes de détection et intérêt de YOLO

L'évolution de la détection d'objet a suivi une transition importante : des méthodes classiques basées sur des descripteurs faits à la main vers des méthodes profondes, puis vers des détecteurs de plus en plus rapides. Zou et al. distinguent notamment deux grandes périodes : la période des détecteurs traditionnels avant 2014, puis celle des détecteurs profonds après l'arrivée des CNN modernes [15].

Les détecteurs profonds peuvent être divisés en deux grandes familles :

— les détecteurs en deux étapes, comme Faster R-CNN, qui génèrent d'abord des régions candidates avant de les classifier ;
— les détecteurs en une étape, ou *single-stage*, qui prédisent directement les objets en une seule passe réseau.

Les détecteurs en deux étapes sont souvent précis, mais plus complexes et plus coûteux. Les détecteurs en une étape sont généralement mieux adaptés aux applications temps réel et embarquées. YOLO, pour *You Only Look Once*, appartient à cette seconde famille. Ali et Zhang rappellent que YOLO s'est imposé comme une famille de modèles majeure grâce à son compromis entre vitesse et précision, ainsi qu'à son usage dans de nombreux domaines applicatifs [16].

L'architecture générale d'un modèle YOLO peut être décrite par trois blocs :

— un **backbone**, chargé d'extraire les caractéristiques visuelles ;
— un **neck**, chargé de fusionner les informations à plusieurs échelles ;
— une **head**, chargée de produire les prédictions finales.

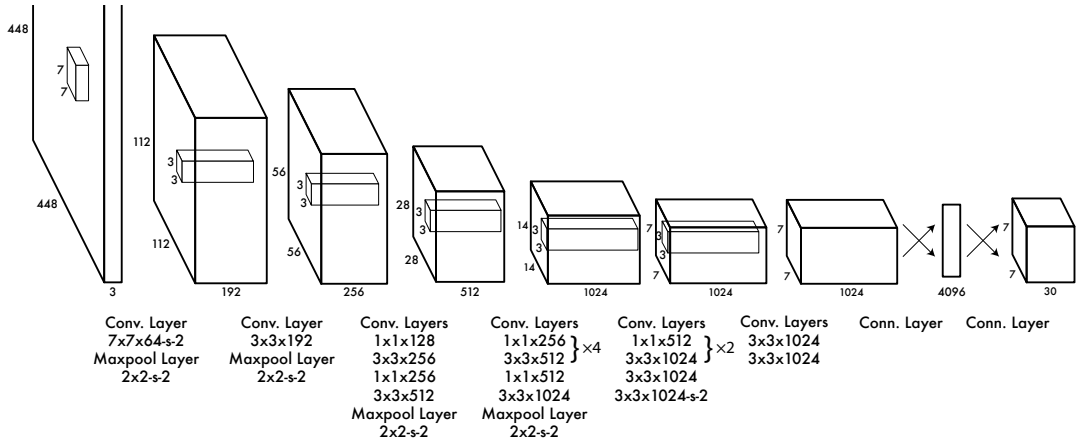


FIGURE 3 – Architecture générale d'un modèle YOLO : extraction de caractéristiques, fusion multi-échelle et prédictions finales.

Dans ce mémoire, trois générations récentes de YOLO sont retenues : YOLOv8, YOLO11 et YOLO26. YOLOv8 constitue une base moderne largement utilisée dans l'écosystème Ultralytics, avec une tête découplée et une logique *anchor-free*. YOLO11 poursuit cette évolution avec des modules orientés vers l'efficacité et une meilleure allocation des tâches d'apprentissage. YOLO26 est inclus en raison de ses promesses récentes en matière de simplification du post-traitement, notamment via une inférence annoncée comme native sans NMS (non-maximum suppression), une meilleure prise en compte des petits objets et une orientation plus marquée vers le déploiement [17]. Cependant, d'autres modèle de YOLO existent.

Cette comparaison permet donc d'évaluer non seulement la précision brute, mais aussi le compromis entre performance, taille de modèle et intégration mobile.

2.4.5 Transfer learning, petits datasets et data augmentation

L'entraînement complet d'un modèle profond nécessite généralement une grande quantité de données annotées. Or, dans un projet spécialisé comme la mesure de grumes taguées, les images réelles disponibles sont limitées, coûteuses à collecter et longues à annoter.

Le *transfer learning* constitue une solution courante à ce problème. Il consiste à partir d'un modèle pré-entraîné sur un grand dataset généraliste, puis à l'adapter à une tâche spécifique par fine-tuning. Le transfert d'apprentissage et le fine-tuning sont particulièrement utiles lorsque l'accès à de grands volumes de données annotées est difficile [18], notamment pour limiter l'impact d'un faible volume de données lors de l'entraînement d'un modèle de mesure de diamètre [13].

La data augmentation complète cette stratégie. Elle consiste à générer des variantes des images d'entraînement en appliquant des transformations qui conservent le sens de l'annotation. Mumuni et Mumuni distinguent notamment les transformations géométriques, comme la rotation, le changement d'échelle ou la perspective, et les transformations photométriques, comme la variation de luminosité, de contraste, de saturation ou de couleur [18].

Dans le cas des grumes, ces augmentations sont directement liées aux conditions terrain :

— les variations photométriques simulent les changements d'éclairage, d'ombre et de couleur ;

— les transformations géométriques simulent les variations de cadrage, de distance et d'orientation ;
— les transformations projectives simulent les prises de vue légèrement inclinées ;
— les données synthétiques peuvent enrichir des cas rares, comme certaines formes atypiques ou occlusions.

Cependant, toutes les augmentations ne sont pas nécessairement utiles. Une transformation trop forte peut produire des images peu réalistes et dégrader l'apprentissage. C'est pourquoi ce mémoire retient une approche expérimentale contrôlée : les familles d'augmentations sont comparées séparément afin d'identifier celles qui améliorent réellement la segmentation des grumes et du tag.

2.5 Vers une mesure automatique des grumes par segmentation d'instance

L'état de l'art montre une évolution claire des méthodes de mesure et d'analyse des grumes. Les approches traditionnelles restent importantes pour définir les grandeurs dendrométriques, mais elles sont lentes et sensibles à l'opérateur. Les solutions 3D, comme les scanners laser ou SLAM, offrent une précision élevée, mais leur déploiement massif reste plus complexe et souvent coûteux. Les méthodes classiques de traitement d'image sont intéressantes en environnement contrôlé, mais deviennent fragiles face aux ombres, aux textures du bois, aux occlusions et aux piles denses.

Les travaux récents convergent vers l'usage de modèles d'apprentissage profond, en particulier pour la segmentation et l'analyse de scènes complexes. Pour une mesure automatique du diamètre, la segmentation d'instance apparaît plus adaptée qu'une simple détection par boîte englobante, car elle fournit un contour exploitable géométriquement. Les architectures YOLO sont particulièrement pertinentes pour ce projet, car elles offrent un compromis entre précision, vitesse, simplicité d'export et compatibilité avec un déploiement embarqué.

Le positionnement de ce mémoire est donc le suivant : utiliser un modèle YOLO de segmentation d'instance, adapté par fine-tuning à un dataset spécifique de grumes et de tags, renforcé par data augmentation, puis intégré dans un pipeline de mesure géométrique compatible avec une exécution locale sur smartphone.

2.6 Inférence embarquée et déploiement côté client

Dans le contexte de la métrologie forestière, la contrainte réside dans l'environnement d'exécution. Les zones d'exploitations forestières sont généralement dépourvues de réseaux et par conséquent prendre une photo et l'envoyer vers les serveurs pour effectuer un traitement dans le cloud n'est donc pas envisageable. C'est pourquoi le traitement de l'image par le modèle doit se faire en local, sur le smartphone de l'utilisateur.

Les progrès techniques du *deep learning* ont considérablement favorisé l'utilisation de l'intelligence artificiel à tel point que plusieurs librairies et frameworks natifs comme TensorFlow [19] ont vu le jour. Historiquement le déploiement de modèles de machine learning nécessitait le développement d'application natives (Android et iOS) via des librairies compatibles pour chaque plateforme et la publication de ces applications sur l'appstore de chaque plateforme, qui est un processus complexe et qui prend du temps.

Avec l'émergence des PWA (Progressive Web App) et les progrès des technologies web comme HTML5, CSS3 et JavaScript, il est désormais possible de déployer des applications de deep learning directement sur le web, sur plusieurs plateformes, avec une expérience proche de celle des applications natives et moins dépendante du matériel utilisé. [20].

2.6.1 Du Serveur vers le Client (Edge)

L'exécution du machine learning côté client répond à un besoin d'autonomie, de faible latence et de déploiement multiplate-forme. Contrairement aux approches serveur, plus adaptées à l'entraînement de modèles lourds, elle évite les échanges réseau et permet une utilisation hors ligne, particulièrement pertinente pour une PWA de détection de grumes en environnement isolé [21].

2.6.2 Faisabilité de la détection d'objets dans le navigateur

L'exécution de modèles de machine learning dans le navigateur a longtemps été considérée comme difficile, notamment en raison des ressources de calcul nécessaires aux tâches de détection d'objets. Cependant, les progrès récents du machine learning côté client montrent qu'il est désormais possible d'exécuter ce type de modèle directement dans un environnement web, y compris pour des usages proches du temps réel [22]. Cette évolution s'appuie sur des frameworks adaptés au navigateur, comme TensorFlow.js et ONNX Runtime Web, ainsi que sur les capacités graphiques modernes offertes par WebGL et WebGPU [19], [23], [24]. Dans ce contexte, l'intégration d'un modèle de détection tel que YOLO dans une application web devient techniquement envisageable [25].

2.6.3 Le défi des performances et la nécessité d'optimisation

Bien que l'exécution de modèles de machine learning dans le navigateur soit techniquement faisable, les performances restent une contrainte importante. Les environnements web présentent généralement un écart de performance par rapport aux environnements natifs, notamment en raison d'un accès plus limité aux ressources matérielles, d'une gestion mémoire dépendante du navigateur et d'un besoin accru d'optimisation des modèles [20].

Ces contraintes montrent que l'intégration d'un modèle de détection de grumes dans une PWA ne peut pas se limiter à un simple portage de code. Elle nécessite une optimisation adaptée au contexte mobile, en exploitant les mécanismes d'accélération disponibles sur smartphone [21]. Dans cette logique, l'utilisation de moteurs d'exécution web comme WebAssembly ou WebGL apparaît pertinente pour améliorer les performances d'inférence côté client [23], [26].

2.6.4 De l'entraînement à l'exécution : l'interopérabilité du format ONNX

PyTorch est largement utilisé pour concevoir et entraîner des modèles de vision par ordinateur, notamment les architectures YOLO [27]. Cependant, son exécution directe dans un navigateur mobile reste peu adaptée, en raison des contraintes de compatibilité, de mémoire et de ressources.

ONNX répond à cette limite en servant de format intermédiaire entre le framework d'entraînement et l'environnement d'exécution final. Il permet d'exporter un

modèle entraîné sous PyTorch vers un format plus portable, indépendant du framework d'origine [28]. Cette conversion facilite ensuite l'exécution du modèle dans une PWA via ONNX Runtime Web.

Au-delà de la compatibilité, ONNX améliore aussi les performances d'inférence et permet d'exploiter des moteurs d'exécution optimisés, capables de tirer parti de l'accélération matérielle disponible sur smartphone [29], [30]. Pour ce projet, l'approche retenue consiste donc à entraîner le modèle YOLO sous PyTorch, puis à l'exporter en ONNX afin de l'exécuter efficacement dans le navigateur mobile.

2.6.5 Comparaison des API d'accélération du navigateur : WASM, WebGL et WebGPU

Une fois que le modèle est converti en ONNX, il faut pouvoir l'exécuter dans le navigateur. Mais le problème c'est que les navigateurs web n'ont pas été conçus initialement pour faire tourner des intelligences artificielles complexes. Pour pouvoir exécuter un modèle YOLO converti en ONNX dans le navigateur, il faut utiliser des technologies d'accélération matérielle (ou APIs) capables de communiquer avec le matériel du smartphone (CPU et GPU). Les trois technologies standards sont WebAssembly (WASM), WebGL et WebGPU.

- WASM (WebAssembly) : permet d'exécuter du code proche du natif directement dans le navigateur, principalement via le CPU. Il est adapté aux calculs intensifs et aux modèles légers, mais reste moins efficace pour les tâches de vision par ordinateur nécessitant une forte parallélisation graphique. [31]
- WebGL : exploite le GPU du navigateur, mais a été conçu à l'origine pour le rendu 2D/3D et non pour le machine learning. Son utilisation pour l'inférence impose donc des contournements techniques, ce qui limite son efficacité pour les applications très gourmandes en calcul [29], [32].
- WebGPU : est une API plus récente, pensée pour le calcul généraliste sur GPU. Elle offre un accès plus direct aux capacités graphiques de l'appareil et constitue donc une solution plus adaptée pour accélérer l'inférence de modèles d'IA dans le navigateur [29].

2.6.6 L'architecture d'ONNX Runtime Web et les Execution Providers

Pour faire le lien entre le fichier du modèle .onnx et les technologies d'accélération matérielles du navigateur que nous avons présentées précédemment il est nécessaire d'utiliser un moteur d'exécution. Ici, ONNX Runtime est utilisé pour déployer les modèles de machine learning, sous le format d'un fichier .onnx, sur les accélérateurs matériels du système.

Pour communiquer avec ces différentes APIs (Wams, WebGL, WebGPU), ONNX Runtime Web utilise des couches d'abstraction qu'on appelle Exécution Providers (EP). L'Execution Provider n'est pas l'API mais un module qui se trouve à l'intérieur de ONNX Runtime et qui sert à convertir les opérations du modèle vers l'API correspondante. Cela signifie qu'il existe un Execution Provider spécifique pour WASM, un pour WebGL et un autre pour WebGPU.

Lors du déploiement de la PWA sur le terrain, les smartphones des différents utilisateurs sont très variés. C'est pour cela qu'il est intéressant de combiner ces dif-

férentes technologies pour permettre une inférence solide dans le navigateur comme le souligne l'article [31]. L'idée est de déclarer un ordre de priorité pour les Execution Providers en tentant d'abord l'EP WebGPU, puis WebGL et enfin l'EP WASM. Cet ordre permet de garantir que l'application s'exécutera sur n'importe quel smartphone en évitant les crashs de l'application sur le terrain.

3 Dataset et stratégie d'annotation

Comme montré dans la section précédente, la mesure automatique du diamètre d'une grume nécessite une segmentation précise plutôt qu'une simple détection par boîte englobante. Le masque de la classe `grume` constitue en effet la base du calcul géométrique ultérieur : il permet d'isoler la section utile du bois, d'en extraire un contour exploitable, puis de calculer une boîte englobante ajustée à la grume.

Cette exigence est particulièrement importante dans les images de terrain, où plusieurs grumes peuvent apparaître simultanément et où les ombres, fissures, craquements, irrégularités d'écorce ou objets partiellement visibles peuvent fausser une détection classique. Le *dataset* a donc été construit selon une logique orientée terrain, afin de représenter des conditions réalistes d'acquisition tout en fournissant au modèle une diversité suffisante de formes, de textures, de conditions lumineuses et de situations d'occlusion.

3.1 Organisation générale du *dataset*

Le *dataset* final contient 109 images, réparties en trois sous-ensembles :
— **77 images dans le *train set*** ;
— **16 images dans le *validation set*** ;
— **16 images dans le *test set***.

Cette répartition correspond à un compromis entre deux contraintes. D'une part, le nombre d'images disponibles reste limité, ce qui impose de conserver une majorité des données pour l'apprentissage. D'autre part, il est indispensable de maintenir un *validation set* et un *test set* distincts afin d'évaluer la capacité de généralisation du modèle sur des images non utilisées pendant l'entraînement. Cette logique suit la séparation classique entre *train set*, *validation set* et *test set*, où le *test set* doit rester strictement réservé à l'évaluation finale [33].

Le choix d'un split proche de 70/15/15 permet ainsi de conserver suffisamment d'images dans le *train set* tout en gardant deux ensembles de contrôle exploitables. Le *test set* a volontairement été limité à des images réelles individuelles et mesurées, car il correspond au scénario d'utilisation principal de l'application : un agent photographie une grume de face afin d'en estimer automatiquement le diamètre.

3.1.1 Convention de nommage

Chaque image du *dataset* suit une convention de nommage structurée afin de conserver les informations essentielles sur son origine, sa typologie et son statut métrologique. La structure générale est la suivante :

`ID_SOURCE_TYPE_FORME_MESURE_D1_D2`

Les champs utilisés dans cette convention sont décrits dans le tableau 1.

TABLE 1 – Description des champs utilisés dans la convention de nommage des images

Champ	Code	Signification
SOURCE	R	Image réelle
	S	Image synthétique générée par un modèle génératif
TYPE	I	Grume individuelle occupant la majorité de l'image
	P	Pile de plusieurs grumes
FORME	C	Section circulaire
	S	Section carrée ou proche d'une forme rectangulaire
	A	Forme atypique, ni circulaire ni carrée
MESURE	M	Diamètre mesuré physiquement sur le terrain
	U	Diamètre non mesuré
D1, D2	–	Mesures de diamètre disponibles, lorsqu'elles existent

Le champ MESURE indique si le diamètre a été mesuré physiquement par les forestiers et transmis par l'ingénieur de PEPPS au Guyana. Les champs D1 et D2 permettent, lorsque l'information est disponible, de conserver directement les valeurs de référence associées à l'image.

Cette convention facilite la traçabilité des images et permet d'analyser automatiquement la distribution du *dataset* selon l'origine, le type de scène, la forme de la grume et la disponibilité des mesures.

3.1.2 Distribution finale

Le tableau 2 résume la distribution finale du *dataset*.

TABLE 2 – Rapport de distribution finale du *dataset* de grumes

Attribut	***Train set***	***Validation set***	***Test set***
Source (R / S)	46.8% / 53.2%	100% / 0%	100% / 0%
Type (I / P)	53.2% / 46.8%	62.5% / 37.5%	100% / 0%
Forme (C / S / A)	71.4% / 6.5% / 22.1%	75% / 0% / 25%	68.8% / 6.2% / 25%
Mesure (M / U)	22.1% / 77.9%	50% / 50%	100% / 0%
Total Images	**77**	**16**	**16**

La répartition montre que le *train set* combine des images réelles et synthétiques dans des proportions proches. Cette stratégie permet d'exposer le modèle à une plus grande diversité de situations visuelles tout en conservant une base réelle significative.

Le *validation set* et le *test set* sont exclusivement composés d'images réelles afin d'éviter que l'évaluation ne soit influencée par des textures ou détails anatomiques générés artificiellement.

La distribution des formes montre que les sections circulaires constituent la majorité des cas, tandis que les formes atypiques sont conservées afin d'exposer le modèle à des contours irréguliers. Les sections carrées restent très rares : seules deux photographies réelles étaient disponibles. Une image a donc été placée dans le *train*

set et l'autre dans le *test set*, afin de conserver cette géométrie rare à la fois dans l'apprentissage et dans l'évaluation finale.

3.2 Origine et rôle des images

3.2.1 Images réelles

Les images réelles proviennent de photographies de terrain prises dans le cadre du projet dWTS au Guyana. Elles représentent des grumes individuelles ou des piles de grumes dans des conditions variées : grumes posées au sol, grumes chargées sur camion, scènes encombrées, présence d'herbe, différences d'éclairage, ombres portées et arrière-plans forestiers.

La figure 4 présente trois exemples représentatifs des images réelles utilisées : une grume individuelle photographiée de face, une pile de grumes chargée sur camion et une pile de grumes posée au sol. Ces situations correspondent aux principaux cas rencontrés dans les données de terrain.

(a) Grume individuelle. (b) Pile sur camion. (c) Pile au sol.

FIGURE 4 – Exemples d'images réelles issues du contexte terrain du projet dWTS.

Une attention particulière a été portée à la séparation des sous-ensembles. Lorsqu'une même pile de grumes apparaissait sous plusieurs angles ou à plusieurs distances, les images correspondantes ont été conservées dans le même sous-ensemble. Cette précaution vise à limiter les fuites de données entre le *train set*, le *validation set* et le *test set*, qui pourraient conduire à une surestimation artificielle des performances.

3.2.2 Rôle des images réelles dans le *test set*

Le *test set* contient uniquement des photographies réelles de grumes individuelles, toutes accompagnées de mesures physiques de référence. Ce choix est volontaire : l'objectif principal de l'application est de permettre à un agent de mesurer une grume à partir d'une photographie centrée de sa section transversale.

Le *test set* évalue donc la situation d'usage réelle, c'est-à-dire le cas où l'opérateur cherche à mesurer une grume donnée. Les scènes plus complexes, comme les piles de grumes, ne sont pas ignorées pour autant : elles sont utilisées pendant l'entraînement afin d'améliorer la robustesse visuelle du modèle pour généraliser. La robustesse est ensuite étudiée dans la méthodologie expérimentale, notamment à travers des tests de recul de cadrage simulé.

Cette organisation permet ainsi de concentrer l'évaluation finale sur le cas d'usage principal, tout en exploitant les scènes plus complexes dans le *train set* pour renforcer la robustesse visuelle du modèle.

3.2.3 Images synthétiques

Le volume d'images réelles étant limité, des images synthétiques ont été générées afin d'enrichir le *train set*. Elles ont été produites à partir d'images de référence, à l'aide du modèle génératif d'images Nano Banana 2 de Google. L'objectif était de créer de nouvelles scènes cohérentes avec le contexte forestier du projet, tout en introduisant des variations visuelles supplémentaires : changements d'éclairage, détails d'écorce différents, environnement forestier modifié, occlusions partielles, empilements de grumes ou formes plus atypiques.

Ces données synthétiques ont été utilisées pour renforcer l'apprentissage dans des situations peu fréquentes dans les données réelles, par exemple :

— pile de grumes sur un camion ;
— grumes partiellement obstruées par de l'herbe ;
— formes atypiques ;
— lueurs fortes ou effets de soleil ;
— grumes peintes ou marquées.

La figure 5 présente trois exemples de situations synthétiques générées : une scène de pile, une variation lumineuse forte et une forme de grume atypique.

(a) Pile synthétique. (b) Variation lumineuse. (c) Forme atypique.

FIGURE 5 – Exemples d'images synthétiques utilisées pour enrichir le *train set*.

Cependant, les images synthétiques n'ont pas été utilisées dans le *test set*. Ce choix est essentiel, car les textures internes du bois, les détails d'écorce et les caractéristiques anatomiques fines générées par IA ne peuvent pas être considérés comme des données biologiquement fiables. Elles sont utiles pour améliorer la robustesse visuelle du modèle de segmentation, mais elles ne constituent pas une base fiable pour des tâches plus fines, comme l'identification de l'essence du bois.

3.2.4 Limite volontaire : non-prédiction de l'essence

Un objectif initial du projet consistait à explorer la possibilité de prédire l'essence des grumes à partir des images. Cette piste n'a finalement pas été retenue dans le périmètre expérimental du mémoire.

L'identification automatique des essences de bois est une tâche de classification fine. Elle repose sur des différences visuelles parfois très faibles entre espèces et nécessite généralement des images de haute qualité, standardisées et suffisamment nombreuses. La littérature sur l'identification du bois par vision montre que cette tâche est sensible aux faibles variations interspécifiques et à la qualité du protocole d'acquisition [34].

Dans ce projet, plusieurs facteurs rendaient cette tâche peu réaliste :

— le nombre d'images réelles par essence était insuffisant ;
— les conditions d'acquisition n'étaient pas standardisées pour l'identification anatomique ;
— les images synthétiques ne garantissent pas la fidélité des détails propres à chaque essence ;
— l'objectif principal du système est métrologique, et non taxonomique.

La prédiction de l'essence est donc considérée comme une perspective future nécessitant un *dataset* dédié, plus large, équilibré par espèce et acquis selon un protocole photographique spécifique.

3.3 Stratégie d'annotation

Les annotations ont été réalisées manuellement à l'aide de CVAT (*Computer Vision Annotation Tool*), puis exportées au format YOLO Segmentation compatible avec le framework Ultralytics.

Deux classes ont été définies :

— `tag` : étiquette physique de référence ;
— `grume` : section transversale visible du bois.

Dans le fichier de configuration du *dataset*, les classes sont encodées de la manière suivante :

```
nc: 2
names:
  0: tag
  1: grume
```

Chaque image possède un fichier texte associé contenant une ligne par instance annotée :

```
<class_index> <x1> <y1> <x2> <y2> ... <xn> <yn>
```

Les coordonnées des points polygonaux sont normalisées dans l'intervalle $[0, 1]$, ce qui rend les annotations indépendantes de la résolution originale de l'image.

3.3.1 Annotation de la grume

L'annotation de la classe `grume` suit la section utile du bois plutôt que le contour externe complet de l'écorce. Cette décision est liée à l'objectif métrologique du projet : le diamètre recherché correspond au diamètre commercial utile, plus proche d'une mesure sous écorce que d'un contour externe incluant les irrégularités de l'écorce.

Dans les cas simples, le masque suit donc la limite interne de l'écorce, en épousant la forme générale de la section. Dans les cas plus complexes, notamment lorsque

la grume présente une fissure importante ou une découpe irrégulière, l'annotation conserve l'objet comme une seule grume. Même lorsqu'une fissure divise visuellement la section en deux zones, le masque doit représenter l'ensemble de la section de bois à mesurer.

Cette stratégie permet d'éviter que le modèle n'apprenne à séparer artificiellement une grume en plusieurs objets lorsque celle-ci présente des fentes, cassures ou dégradations internes.

Certaines images individuelles contiennent principalement une seule grume, mais d'autres grumes peuvent être visibles en arrière-plan ou sur les côtés. Lorsque ces objets étaient suffisamment discernables, ils ont également été annotés afin d'éviter que le modèle n'apprenne à les considérer comme du fond.

(a) Annotation suivant la section utile sous écorce.

(b) Grume fissurée annotée comme une seule instance.

FIGURE 6 – Exemples de choix d'annotation pour la classe `grume`.

3.3.2 Annotation du tag de référence

Le tag constitue la référence dimensionnelle utilisée pour convertir les distances en pixels vers des distances réelles. Sa largeur physique connue est utilisée dans l'étape de calibration géométrique. Pour cette raison, son annotation doit être particulièrement précise.

Le masque du tag suit la surface visible réelle de l'étiquette, y compris lorsque celle-ci est pliée, partiellement déformée ou fixée de manière imparfaite sur la grume. Les variations de couleur des tags ont également été conservées afin que le modèle n'associe pas la classe uniquement à une apparence chromatique particulière.

Certaines images présentent des tags superposés à des marques peintes ou à des zones de bois colorées, notamment des grumes peintes en vert. Ces cas ont été conservés afin d'apprendre au modèle à distinguer le tag comme objet de référence, indépendamment des autres marques visibles sur la section.

(a) Variation de couleur.

(b) Tag plié ou déformé.

(c) Tag sur grume marquée et colorée.

FIGURE 7 – Variabilité visuelle de la classe `tag`.

3.3.3 Gestion des ambiguïtés visuelles

Certaines images réelles présentent des situations où la section de la grume n'est pas entièrement visible. Les deux cas les plus importants concernent les occlusions par végétation et les piles denses, dans lesquelles certaines grumes sont partiellement cachées par d'autres.

Dans le cas de la végétation, les herbes ou feuilles peuvent masquer une partie du contour de la grume. Lorsque la section restait identifiable, le masque a été prolongé de manière cohérente avec la forme visible. En revanche, les zones totalement invisibles n'ont pas été inventées.

Dans les piles denses, certaines grumes ressortent davantage au premier plan, tandis que d'autres sont partiellement cachées à l'arrière. Dans ce cas, seules les parties visibles appartenant clairement à une grume ont été annotées. Cette règle permet de conserver une annotation fidèle à l'image tout en exposant le modèle à des cas réalistes d'occlusion et de chevauchement.

(a) Grumes partiellement masquées par la végétation.

(b) Pile dense avec grumes partiellement cachées.

FIGURE 8 – Exemples d'occlusions prises en compte lors de l'annotation du *dataset*.

3.4 Limites du *dataset*

Le *dataset* utilisé dans ce mémoire reste de taille réduite par rapport aux grands *datasets* généralement utilisés pour entraîner des modèles de vision profonde depuis zéro. Cette limite est assumée. Le projet ne vise pas à entraîner une architecture complète à partir de rien, mais à adapter par fine-tuning des modèles YOLO pré-entraînés.

Dans les contextes où les données annotées sont limitées, le transfert d'apprentissage et l'augmentation de données constituent des stratégies classiques pour améliorer la généralisation. La littérature sur les problèmes de données limitées en apprentissage profond souligne notamment l'intérêt d'utiliser des modèles pré-entraînés et des techniques d'augmentation lorsque le volume de données propres au domaine est restreint [35].

Le *dataset* constitue donc une base expérimentale réaliste pour évaluer la faisabilité du système, comparer plusieurs architectures et identifier un modèle candidat pour intégration mobile. En revanche, il ne doit pas être interprété comme une base suffisante pour couvrir l'ensemble des conditions forestières possibles au Guyana. Une validation terrain à plus grande échelle resterait nécessaire avant un déploiement opérationnel complet.

3.5 Synthèse

La stratégie de constitution du *dataset* repose sur trois principes :

1. préserver un *test set* réel, mesuré et non synthétique pour l'évaluation finale ;
2. enrichir le *train set* par des images synthétiques et des scènes complexes afin d'améliorer la robustesse visuelle ;
3. annoter les objets selon une logique métrologique, en privilégiant la section utile de la grume et la précision du tag de référence.

Cette organisation permet de relier directement le *dataset* à l'objectif final du projet : mesurer automatiquement le diamètre d'une grume à partir d'une photographie terrain, tout en conservant une certaine robustesse face aux conditions visuelles difficiles.

4 Méthodologie expérimentale

L'état de l'art a montré que la segmentation d'instance constitue l'approche la plus adaptée à une mesure automatique du diamètre de grumes. Contrairement à une simple détection par boîte englobante, elle fournit un masque exploitable géométriquement, ce qui est nécessaire pour estimer un diamètre à partir du contour visible de la section de bois. Par ailleurs, les architectures YOLO apparaissent particulièrement pertinentes dans le contexte du projet, car elles offrent un compromis favorable entre précision, rapidité d'inférence, segmentation native et compatibilité avec un déploiement embarqué.

La méthodologie expérimentale vise donc à identifier un modèle YOLO de segmentation d'instance capable de détecter simultanément la section transversale d'une grume et le tag physique de référence, tout en restant compatible avec une exécution locale sur smartphone. Le choix du modèle ne repose pas uniquement sur la précision maximale : il doit également tenir compte de la robustesse terrain et du coût d'inférence dans une application hors-ligne.

Trois familles récentes de modèles YOLO ont été retenues : YOLOv8, YOLO11 et YOLO26. YOLOv8 constitue une référence largement documentée et utilisée dans l'écosystème Ultralytics. YOLO11 représente une évolution plus récente, annoncée comme plus efficace à précision comparable. YOLO26 est inclus afin d'évaluer une architecture encore plus récente, censée améliorer le compromis entre performance et coût de post-traitement. Pour chaque famille, deux tailles sont comparées : une version *Nano*, plus légère et adaptée au mobile, et une version *Small*, plus capacitaire mais plus coûteuse.

La démarche expérimentale cherche à établir un compromis entre quatre contraintes :

— la précision de segmentation des masques ;
— la capacité à détecter le tag, indispensable à la conversion pixel-centimètre ;
— la robustesse aux variations de prise de vue et aux conditions terrain ;
— le coût computationnel, en vue d'une intégration dans une PWA hors-ligne.

La méthodologie est structurée en six expériences successives. Chaque expérience répond à une question précise et conditionne la suivante :

1. **Expérience 1 — Résolution d'entrée** : déterminer si une résolution de 640 px est suffisante ou si une résolution plus élevée est nécessaire pour préserver les petits objets, notamment le tag.
2. **Expérience 2 — Benchmark initial des architectures** : comparer les familles YOLO retenues afin d'établir un premier compromis entre précision et coût d'inférence.
3. **Expérience 3 — Ablation des augmentations** : mesurer l'effet de différentes familles d'augmentations sur la segmentation des grumes et des tags.
4. **Expérience 4 — Entraînement final** : entraîner chaque architecture avec sa meilleure configuration d'augmentation identifiée lors de l'ablation.
5. **Expérience 5 — Évaluation finale sur *test set*** : comparer les modèles finaux sur le *test set* réel, non synthétique et mesuré.
6. **Expérience 6 — Robustesse au recul de cadrage simulé** : vérifier la stabilité des modèles lorsque la grume occupe une surface plus réduite dans l'image, à la suite des erreurs observées lors de l'inspection manuelle des prédictions.

Cette organisation permet de séparer clairement les décisions expérimentales. La résolution est d'abord fixée, les architectures sont ensuite comparées, les augmentations sont étudiées de manière contrôlée, puis les modèles finaux sont évalués dans des conditions de plus en plus proches du scénario d'utilisation final.

Métrique principale d'évaluation

La métrique principale utilisée dans les expériences est la mAP50-95 masque. Elle évalue la qualité des masques de segmentation en comparant le masque prédit au masque annoté à l'aide de l'IoU (*Intersection over Union*) :

$$IoU = \frac{M_{\text{prédit}} \cap M_{\text{réel}}}{M_{\text{prédit}} \cup M_{\text{réel}}} \tag{1}$$

La mAP50 correspond à une évaluation avec un seuil d'IoU fixé à 0.50. Une prédiction est donc considérée correcte si son masque recouvre suffisamment le masque réel. La mAP50-95 est plus stricte : elle moyenne les performances sur plusieurs seuils d'IoU, de 0.50 à 0.95. Elle pénalise donc davantage les masques imprécis.

Dans ce mémoire, la mAP50-95 masque est privilégiée, car la précision du contour influence directement l'estimation géométrique du diamètre.

4.1 Expérience 1 : impact de la résolution d'entrée

La première expérience vise à déterminer la résolution d'entrée la plus adaptée au problème. Les modèles YOLO peuvent être entraînés avec différentes tailles d'image, mais une résolution trop faible peut entraîner une perte d'information sur les petits objets.

Dans ce projet, cette question est centrale, car le tag physique de référence occupe une surface réduite par rapport à la grume. Or, une mauvaise segmentation du tag compromet directement la conversion pixel-centimètre utilisée pour l'estimation du diamètre.

Deux résolutions ont donc été comparées :

— **640 px**, correspondant à une résolution couramment utilisée dans les entraînements YOLO ;

— **1024 px**, permettant de conserver davantage de détails fins.

L'expérience a été réalisée sur une architecture légère afin d'isoler l'effet de la résolution. Les performances sont comparées à l'aide de la mAP50-95 masque, mesurée globalement ainsi que séparément pour les classes `tag` et `grume`.

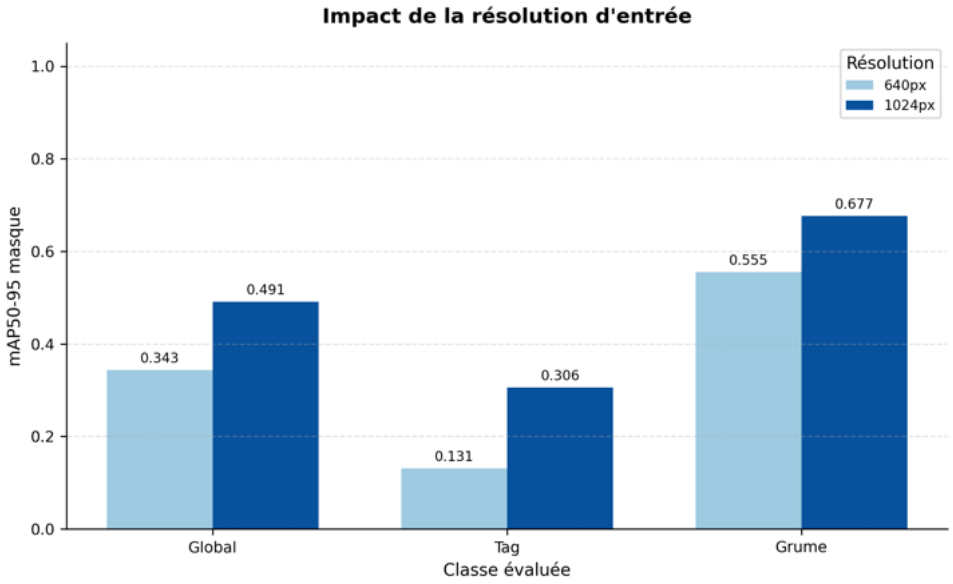


FIGURE 9 – Impact de la résolution d'entrée sur la segmentation des masques.

Les résultats montrent que le passage de 640 px à 1024 px améliore nettement la segmentation, en particulier pour la classe `tag`. Cette amélioration confirme que la résolution de 640 px ne préserve pas suffisamment les informations géométriques nécessaires à la détection fiable du tag de référence.

La résolution de 1024 px est donc retenue pour l'ensemble des expériences suivantes. Ce choix augmente le coût d'inférence, mais il est justifié par le rôle métrologique du tag dans le pipeline de mesure.

4.2 Expérience 2 : benchmark initial des architectures

Après avoir fixé la résolution d'entrée à 1024 px, une seconde expérience compare plusieurs architectures YOLO de segmentation d'instance. L'objectif n'est pas encore de sélectionner le modèle final, mais d'établir une première cartographie du compromis entre précision de segmentation et coût d'inférence.

Les architectures comparées sont :

— **YOLOv8n-seg** et **YOLOv8s-seg** ;
— **YOLO11n-seg** et **YOLO11s-seg** ;
— **YOLO26n-seg** et **YOLO26s-seg**.

Les suffixes `n` et `s` désignent respectivement les versions *Nano* et *Small*. Les modèles Nano sont plus légers et mieux adaptés à une exécution mobile, tandis que les modèles Small disposent d'une capacité supérieure, mais au prix d'une latence plus élevée.

La comparaison repose sur la mAP50-95 masque et sur la latence CPU mesurée localement. Cette seconde métrique est privilégiée dans l'analyse principale, car elle est plus directement liée à l'objectif d'intégration dans une application hors-ligne que le coût computationnel théorique exprimé en GFLOPs.

La latence CPU a été mesurée localement sur un MacBook Air équipé d'une puce Apple M2, avec une résolution d'entrée de 1024 px. Cette mesure ne prétend pas représenter directement la latence finale sur smartphone Android, mais elle fournit un indicateur comparatif homogène du coût d'inférence entre architectures.

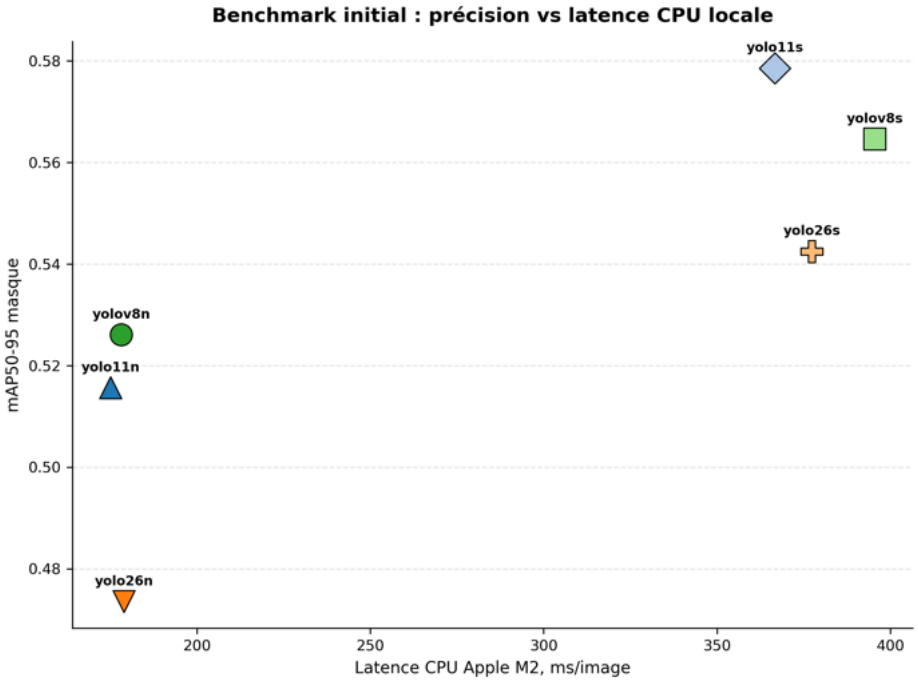


FIGURE 10 – Benchmark initial des architectures : compromis entre précision de segmentation et latence CPU mesurée localement sur Apple M2 avec une entrée de 1024 px.

Le benchmark met en évidence une séparation claire entre les modèles Nano et Small. Les architectures Small obtiennent les meilleures performances brutes, avec notamment YOLO11s en tête. En revanche, leur latence CPU est environ deux fois plus élevée que celle des modèles Nano.

À l'inverse, YOLOv8n et YOLO11n présentent une précision légèrement inférieure, mais une latence nettement plus faible. Ces modèles apparaissent donc comme des candidats plus réalistes pour un déploiement mobile. YOLO26n présente une latence comparable aux autres modèles Nano, mais une précision initiale plus faible.

Cette expérience ne désigne donc pas encore le modèle final. Elle permet plutôt d'identifier deux tendances : les modèles Small constituent une borne haute de précision, tandis que les modèles Nano représentent les candidats les plus pertinents pour l'intégration embarquée.

4.3 Expérience 3 : étude d'ablation des augmentations

La troisième expérience vise à mesurer l'effet des augmentations de données sur la segmentation des grumes et des tags. Cette étape est essentielle, car le *dataset* reste de taille limitée et présente une forte variabilité visuelle : différences d'éclairage, ombres, formes irrégulières, tags de couleurs différentes, occlusions, arrière-plans forestiers et scènes de piles.

L'objectif n'est pas d'effectuer une optimisation automatique globale des hyperparamètres, mais d'isoler l'effet de familles d'augmentations interprétables. Les autres paramètres d'entraînement sont donc maintenus constants afin que les différences observées soient principalement attribuables aux augmentations appliquées.

Chaque architecture a été entraînée pendant 60 époques selon cinq configurations :

— `baseline_no_aug` : entraînement sans augmentation spécifique ;
— `photometric_hsv` : variations de teinte, saturation et luminosité ;
— `geometric_scale_rotation` : changements d'échelle et rotations ;
— `projective_viewpoint` : transformations liées au point de vue, comme la perspective ou le cisaillement ;
— `full_field_aug` : combinaison des augmentations photométriques, géométriques et projectives.

Les augmentations photométriques visent à reproduire les variations d'éclairage rencontrées sur le terrain. Les transformations géométriques exposent le modèle à des changements d'échelle, d'orientation et de cadrage. Les transformations projectives simulent quant à elles des prises de vue légèrement inclinées, fréquentes lorsque l'opérateur photographie une grume sans être parfaitement aligné avec la section transversale.

La métrique principale reste la mAP50-95 masque, définie en début de chapitre.

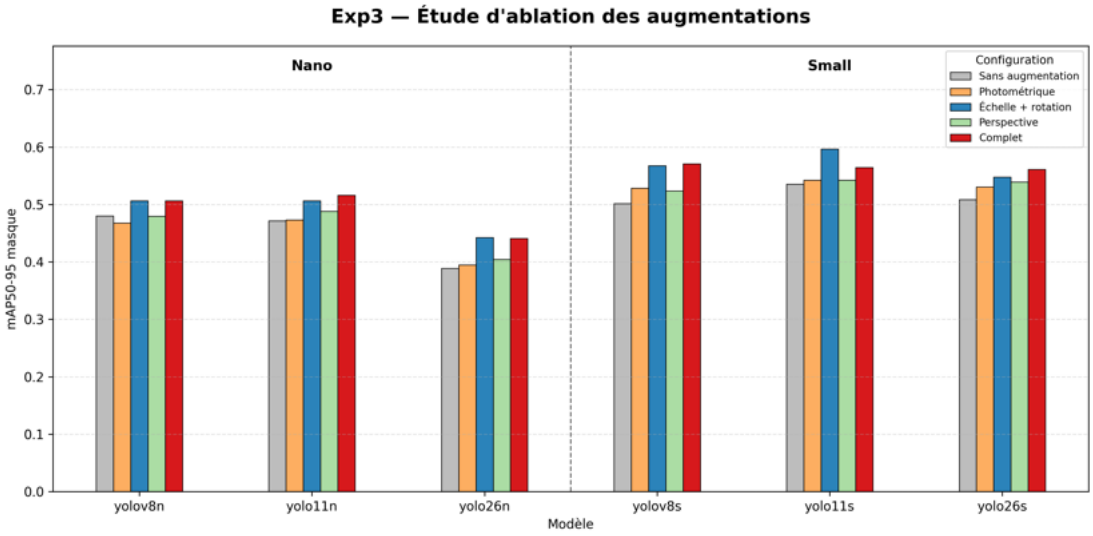


FIGURE 11 – Étude d'ablation des augmentations : comparaison des configurations pour chaque architecture. Les modèles Nano sont regroupés à gauche et les modèles Small à droite.

La figure 11 montre que les augmentations améliorent globalement les performances par rapport à la configuration sans augmentation. L'effet est visible pour l'ensemble des familles de modèles, mais son amplitude varie selon l'architecture.

Chez les modèles Nano, les meilleurs scores sont obtenus avec `full_field_aug` pour YOLOv8n et YOLO11n, avec respectivement 0.5065 et 0.5159 de mAP50-95 masque. Pour YOLO26n, la meilleure configuration est `geometric_scale_rotation`, avec 0.4422. Chez les modèles Small, YOLO11s atteint le meilleur résultat de l'expérience avec `geometric_scale_rotation`, soit 0.5968, tandis que YOLOv8s et YOLO26s obtiennent leurs meilleurs scores avec `full_field_aug`.

Ces résultats montrent que la meilleure stratégie d'augmentation n'est pas identique pour toutes les architectures. Les configurations complètes sont souvent avantageuses, mais elles ne dominent pas systématiquement. Cette observation justifie le choix de sélectionner une configuration propre à chaque modèle plutôt que d'imposer une stratégie unique.

TABLE 3 – Meilleure configuration d'augmentation retenue par architecture à l'issue de l'expérience 3.

Modèle	Configuration retenue	mAP50-95 masque
YOLOv8n	`full_field_aug`	0.5065
YOLO11n	`full_field_aug`	0.5159
YOLO26n	`geometric_scale_rotation`	0.4422
YOLOv8s	`full_field_aug`	0.5710
YOLO11s	`geometric_scale_rotation`	0.5968
YOLO26s	`full_field_aug`	0.5609

L'expérience 3 joue donc un rôle de sélection méthodologique. Elle ne constitue pas l'évaluation finale des modèles, puisque les entraînements sont limités à 60 époques et évalués sur le *validation set*. Elle permet en revanche d'identifier, pour chaque architecture, la configuration d'augmentation la plus prometteuse. Ces configurations sont ensuite utilisées dans l'expérience 4 pour réaliser les entraînements finaux.

4.4 Expérience 4 : entraînement final des modèles

L'expérience 4 consiste à réentraîner chaque architecture avec la meilleure configuration d'augmentation identifiée lors de l'expérience 3. Contrairement à l'ablation, qui servait principalement à comparer les stratégies d'augmentation sur 60 époques, cette étape vise à produire les modèles finaux qui seront ensuite évalués sur le *test set*.

Chaque modèle est entraîné pendant 100 époques, avec une résolution d'entrée de 1024 px. Pour chaque entraînement, le meilleur modèle est sélectionné à partir de la meilleure mAP50-95 masque obtenue sur le *validation set*.

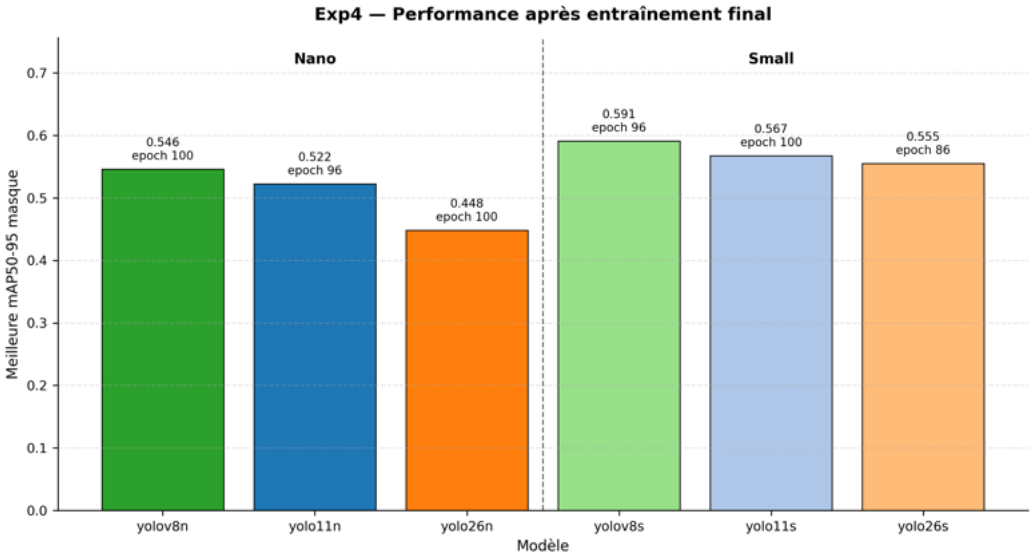


FIGURE 12 – Meilleure performance de validation obtenue après entraînement final pour chaque modèle.

La figure 12 montre que les modèles Small conservent globalement un avantage en

précision de validation par rapport aux versions Nano. YOLOv8s obtient le meilleur score de validation avec une mAP50-95 masque de 0.591, suivi de YOLO11s avec 0.567 et YOLO26s avec 0.555.

Parmi les modèles Nano, YOLOv8n obtient le meilleur résultat avec 0.546, devant YOLO11n à 0.522 et YOLO26n à 0.448. Ces résultats confirment que les modèles légers restent compétitifs, mais avec une précision généralement inférieure aux modèles Small.

TABLE 4 – Comparaison des performances Nano et Small après entraînement final.

Famille	**mAP Nano**	**mAP Small**	**Diff. absolue**	**Diff. relative**
YOLOv8	0.546	0.591	+0.045	+8.2%
YOLO11	0.522	0.567	+0.045	+8.6%
YOLO26	0.448	0.555	+0.107	+23.8%

Le tableau 4 montre que le passage de Nano à Small apporte un gain modéré pour les familles YOLOv8 et YOLO11, avec environ 8% d'amélioration relative. Le gain est beaucoup plus marqué pour YOLO26, où la version Small améliore la mAP50-95 masque de près de 24%. Cette différence suggère que la version Nano de YOLO26 dispose d'une capacité trop limitée pour ce *dataset*, tandis que les versions Nano de YOLOv8 et YOLO11 restent plus proches de leurs équivalents Small.

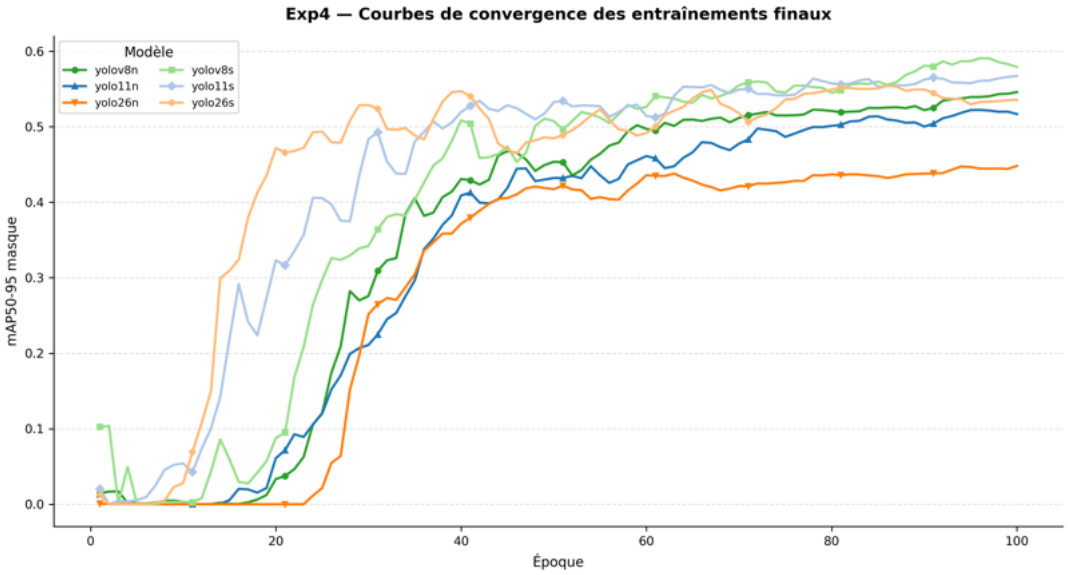


FIGURE 13 – Courbes de convergence des entraînements finaux sur le *validation set*.

Les courbes de convergence de la figure 13 indiquent que les modèles atteignent progressivement un plateau de performance en fin d'entraînement. Certains modèles atteignent leur meilleur score avant la dernière époque, notamment YOLOv8s à l'époque 96, YOLO11n à l'époque 96 et YOLO26s à l'époque 86. Cela justifie l'utilisation du meilleur checkpoint de validation plutôt que du dernier checkpoint d'entraînement.

Cette expérience produit donc les six modèles finaux à comparer sur le *test set*. Les résultats de validation donnent une première indication des performances attendues, mais ils ne suffisent pas à sélectionner définitivement le modèle final.

4.5 Expérience 5 : évaluation finale sur le *test set*

L'expérience 5 évalue les six modèles finaux sur le *test set*. Contrairement aux étapes précédentes, ce sous-ensemble contient uniquement des images réelles, individuelles et mesurées, comme défini dans la section consacrée au *dataset*. Il correspond donc au scénario principal d'utilisation de l'application : un agent photographie une grume de face afin d'en estimer automatiquement le diamètre.

Les modèles évalués sont les meilleurs checkpoints issus de l'expérience 4. L'évaluation est réalisée avec une résolution d'entrée de 1024 px. La latence indiquée dans les tableaux correspond à une inférence CPU mesurée localement sur un MacBook Air équipé d'une puce Apple M2. Elle sert uniquement d'indicateur comparatif entre modèles et ne doit pas être interprétée comme la latence finale sur smartphone Android.

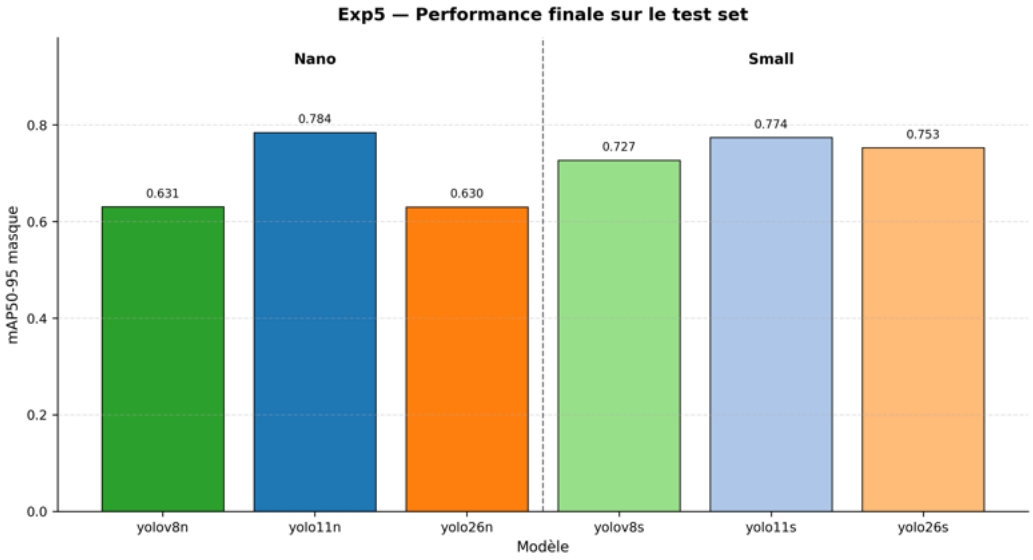


FIGURE 14 – Performance finale des modèles sur le *test set* réel.

La figure 14 montre que YOLO11n obtient la meilleure performance globale sur le *test set*, avec une mAP50-95 masque de 0.784. Ce résultat inverse partiellement la tendance observée en validation : alors que les modèles Small dominaient davantage en Exp4, YOLO11n reprend l'avantage lorsqu'il est évalué sur les images réelles du scénario cible.

YOLO11s reste très proche avec une mAP de 0.774, suivi de YOLO26s avec 0.753 et YOLOv8s avec 0.727. Les modèles YOLOv8n et YOLO26n sont plus en retrait, avec des scores proches de 0.63. À ce stade, YOLO11n apparaît donc comme le candidat le plus intéressant, car il combine une architecture légère avec la meilleure performance globale sur les données de test.

TABLE 5 – Écart entre la meilleure performance de validation Exp4 et la performance sur le *test set* Exp5.

Modèle	**Exp4 val.**	**Exp5 test**	**Écart abs.**	**Écart rel.**
YOLOv8n	0.546	0.631	+0.085	+15.5%
YOLO11n	0.522	0.784	+0.262	+50.2%
YOLO26n	0.448	0.630	+0.182	+40.6%
YOLOv8s	0.591	0.727	+0.136	+23.0%
YOLO11s	0.567	0.774	+0.207	+36.4%
YOLO26s	0.555	0.753	+0.198	+35.7%

Le tableau 5 compare les performances obtenues en validation lors de l'expérience 4 avec celles obtenues sur le *test set*. Cet écart doit être interprété avec prudence, car les deux évaluations ne portent pas sur le même sous-ensemble. Il ne s'agit donc pas d'un gain d'entraînement, mais d'une différence de performance entre deux contextes d'évaluation.

Tous les modèles obtiennent un score plus élevé sur le *test set* que sur le *validation set*. Cette hausse s'explique probablement par la nature du *test set* : il est composé uniquement de grumes individuelles, réelles et mesurées, ce qui correspond davantage au scénario d'utilisation final. À l'inverse, le *validation set* contient davantage de variété visuelle, notamment des piles ou des situations plus complexes.

TABLE 6 – Résumé des performances finales sur le *test set*. La latence est mesurée localement sur CPU MacBook Air Apple M2.

Modèle	**mAP globale**	**mAP Tag**	**mAP Grume**	**Précision**	**Rappel**	**Latence M2 CPU (ms)**	**Taille (MB)**
YOLOv8n	0.631	0.870	0.392	0.717	0.751	155.2	6.6
YOLO11n	0.784	0.896	0.672	0.866	0.886	148.3	5.8
YOLO26n	0.630	0.813	0.448	0.632	0.732	150.4	6.3
YOLOv8s	0.727	0.872	0.583	0.874	0.812	365.8	22.8
YOLO11s	0.774	0.902	0.646	0.876	0.867	312.5	19.7
YOLO26s	0.753	0.864	0.642	0.815	0.815	323.2	22.3

Le tableau 6 confirme que YOLO11n obtient le meilleur score global tout en conservant une latence et une taille de modèle faibles. Il atteint 0.784 de mAP50-95 masque, avec une latence CPU mesurée à 148.3 ms/image sur MacBook Air M2 et une taille de modèle de 5.8 MB. En comparaison, YOLO11s atteint un score très proche, mais avec une latence plus élevée et une taille de modèle nettement plus importante.

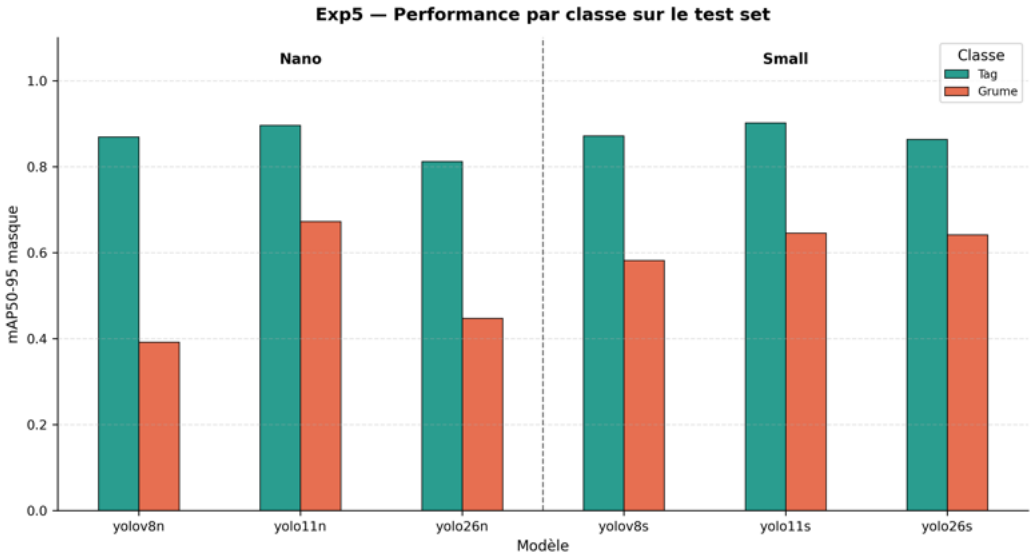


FIGURE 15 – Performance par classe sur le *test set*.

La figure 15 met en évidence une différence nette entre les deux classes. Les scores obtenus sur la classe `tag` sont élevés pour tous les modèles, généralement entre 0.81 et 0.90. À l'inverse, les scores obtenus sur la classe `grume` sont plus faibles et plus dispersés.

Cette différence est cohérente avec la nature des objets. Le tag possède une forme rectangulaire relativement stable et une apparence plus régulière. La grume, en revanche, présente une variabilité beaucoup plus forte : formes circulaires, carrées ou atypiques, textures différentes selon les essences, fissures, écorce irrégulière, traces de peinture, ombres et dégradations de surface.

YOLO11n obtient le meilleur score sur la classe `grume`, avec 0.672. Ce point est déterminant pour le projet, car la qualité du masque de la grume influence directement l'estimation géométrique du diamètre. Le tag est indispensable pour calculer le facteur de conversion pixel-centimètre, mais la mesure finale dépend également de la précision du contour de la section de bois.

En résumé, l'expérience 5 place YOLO11n en tête sur le *test set* réel. Ce résultat en fait le candidat principal avant l'analyse finale de robustesse.

4.6 Expérience 6 : robustesse au recul de cadrage simulé

L'expérience 6 vise à tester la robustesse des modèles face à une variation de cadrage, en simulant un éloignement progressif entre l'appareil photo et la grume. Cette expérience a été motivée par l'inspection manuelle des prédictions obtenues après l'expérience 5. Sur certaines images du *test set*, plusieurs modèles définissaient mal la boîte englobante ou le masque lorsque la grume principale occupait presque toute l'image. À l'inverse, des grumes plus éloignées ou visibles en arrière-plan étaient parfois mieux détectées.

Cette observation suggère que certains modèles peuvent être plus à l'aise lorsque la grume occupe une surface plus réduite dans l'image. Cela peut s'expliquer par la présence, dans le *train set*, de piles de grumes ou de scènes où les sections de bois apparaissent à plus petite échelle. L'expérience 6 simule donc un recul de cadrage

progressif en appliquant plusieurs facteurs de réduction d'échelle aux images du *test set*.

Les facteurs testés sont $x1.0$, $x1.5$, $x2.0$, $x2.5$ et $x3.0$. Le facteur $x1.0$ correspond à l'image originale, tandis que le facteur $x3.0$ représente une grume trois fois plus petite au centre de l'image. Les labels polygonaux sont remappés automatiquement afin que l'évaluation reste cohérente avec les nouvelles positions des objets.

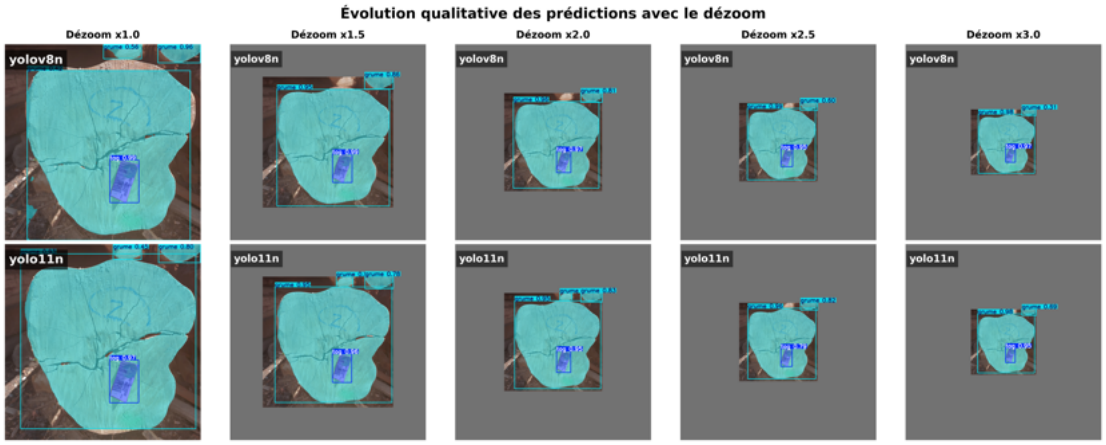


FIGURE 16 – Illustration qualitative de l'effet du recul de cadrage simulé sur les prédictions de YOLOv8n et YOLO11n.

La figure 16 illustre l'intuition initiale de cette expérience. Sur l'image originale, YOLOv8n détecte bien la grume principale, mais sa boîte englobante est moins bien ajustée à la section de bois. Le masque couvre largement la grume, mais l'encadrement inclut davantage de contexte. À mesure que le recul de cadrage simulé augmente, YOLOv8n conserve une détection de la grume principale, mais les grumes secondaires ou floues en arrière-plan sont moins clairement prises en compte.

YOLO11n présente un comportement plus stable. Le masque et la boîte englobante restent mieux ajustés à la grume principale lorsque l'objet devient plus petit dans l'image. De plus, YOLO11n continue à détecter certaines grumes d'arrière-plan floues lorsque le recul de cadrage simulé augmente, ce qui indique une meilleure capacité à conserver des indices visuels sur des objets de plus petite taille. Cette observation qualitative est cohérente avec les résultats quantitatifs présentés ci-dessous.

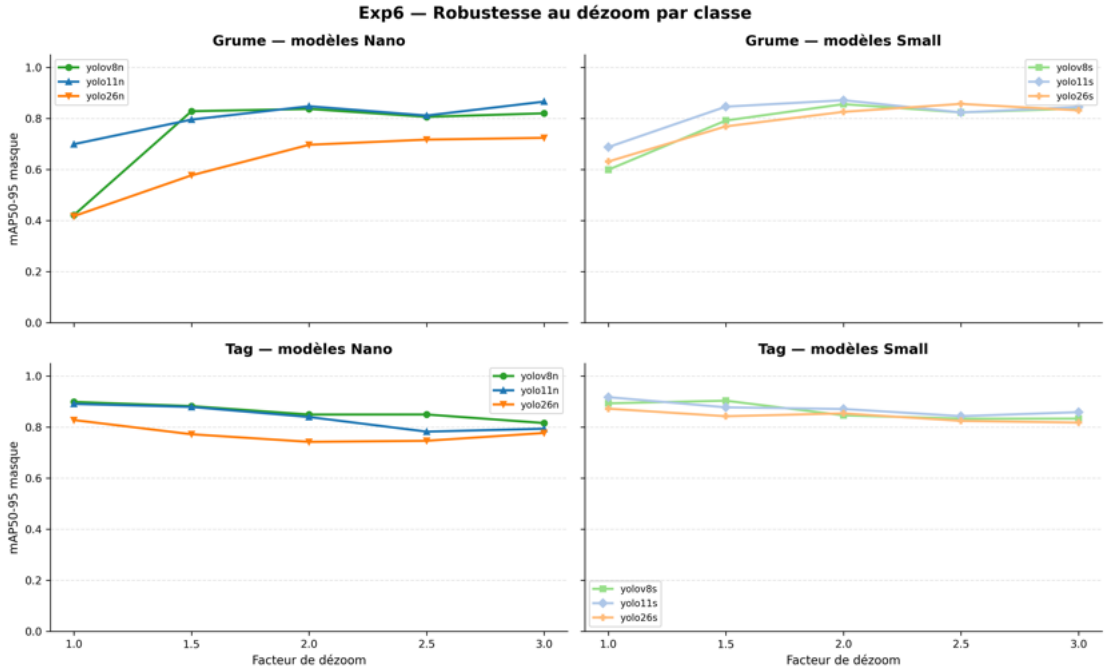


FIGURE 17 – Évolution de la mAP50-95 masque en fonction du facteur de recul de cadrage simulé, séparément pour les classes `grume` et `tag`.

La figure 17 compare l'évolution des performances pour les classes `grume` et `tag`. Les modèles Nano sont présentés séparément des modèles Small afin de faciliter la lecture.

Pour la classe `grume`, les performances augmentent globalement lorsque le facteur de recul de cadrage simulé passe de $x1.0$ à $x2.0$ ou $x3.0$. Ce résultat confirme l'hypothèse formulée à partir de l'analyse visuelle : lorsque la grume occupe toute l'image, certains modèles définissent moins bien son contour. En revanche, lorsque la section devient plus petite et davantage contextualisée dans l'image, les modèles retrouvent une géométrie plus proche de celle observée dans les scènes d'entraînement.

Parmi les modèles Nano, YOLO11n est le plus robuste. Sa mAP50-95 sur la classe `grume` passe de 0.700 à $x1.0$ à 0.866 à $x3.0$. Il dépasse ainsi YOLOv8n et YOLO26n sur les facteurs de recul de cadrage simulé élevés. YOLOv8n progresse fortement entre $x1.0$ et $x1.5$, mais reste légèrement moins stable que YOLO11n sur l'ensemble de la courbe. YOLO26n part plus bas et reste globalement le plus faible des trois modèles Nano.

Les modèles Small présentent également de bonnes performances, mais leur avantage n'est pas suffisant pour remettre en cause l'intérêt de YOLO11n. YOLO11s obtient la meilleure moyenne sur la classe `grume`, mais YOLO11n reste très proche tout en étant beaucoup plus léger. Ce point est important dans le contexte d'une intégration mobile hors-ligne.

Pour la classe `tag`, les courbes sont plus stables et les écarts entre modèles sont moins marqués. Les scores restent élevés pour la majorité des modèles. Cette stabilité s'explique par la forme plus régulière du tag. En revanche, une légère baisse peut apparaître lorsque le recul de cadrage simulé augmente, car le tag devient un objet de plus en plus petit dans l'image.

TABLE 7 – Robustesse moyenne au recul de cadrage simulé, selon la mAP50-95 masque.

Modèle	mAP glob.	mAP Tag	mAP Grume
YOLOv8n	0.801	0.859	0.743
YOLO11n	0.821	0.837	0.804
YOLO26n	0.700	0.773	0.627
YOLOv8s	0.822	0.862	0.782
YOLO11s	0.844	0.874	0.815
YOLO26s	0.813	0.842	0.783

Le tableau 7 résume les performances moyennes sur l'ensemble des facteurs de recul de cadrage simulé. YOLO11s obtient la meilleure moyenne globale, mais YOLO11n est le meilleur modèle Nano et reste très proche des modèles Small. Sur la classe `grume`, qui est la plus importante pour la mesure du diamètre, YOLO11n atteint une moyenne de 0.804, contre 0.743 pour YOLOv8n et 0.627 pour YOLO26n. Cela confirme que YOLO11n est le modèle léger le plus robuste face aux variations de cadrage.

4.6.1 Analyse par morphologie de grume

Une analyse complémentaire a été réalisée sur les modèles Nano afin d'étudier leur comportement selon la morphologie des grumes. Les images du *test set* sont regroupées selon la convention définie dans la section *dataset* : `C` pour circulaire, `S` pour carrée et `A` pour atypique.

Cette analyse est limitée aux modèles Nano, car ce sont les candidats les plus réalistes pour une intégration mobile. Elle permet de vérifier si un modèle performant en moyenne présente malgré tout une faiblesse importante sur un sous-type de grume.

TABLE 8 – Analyse morphologique des modèles Nano sur la classe `grume`, selon la mAP50-95 masque.

Modèle	Forme	Images	mAP moy.	mAP x1	mAP x3
YOLOv8n	Circulaire	11	0.720	0.376	0.807
YOLOv8n	Atypique	4	0.829	0.492	0.928
YOLOv8n	Carrée	1	0.995	0.995	0.995
YOLO11n	Circulaire	11	0.810	0.735	0.862
YOLO11n	Atypique	4	0.815	0.597	0.887
YOLO11n	Carrée	1	0.995	0.995	0.995
YOLO26n	Circulaire	11	0.636	0.502	0.718
YOLO26n	Atypique	4	0.659	0.381	0.752
YOLO26n	Carrée	1	0.796	0.000	0.995

Le tableau 8 confirme l'intérêt de YOLO11n. Sur les grumes circulaires, qui constituent la majorité du *test set*, YOLO11n obtient la meilleure mAP moyenne sur la classe `grume` avec 0.810. Il est également plus stable que YOLOv8n entre $x1.0$ et $x3.0$, ce qui indique une meilleure robustesse à la variation de cadrage.

Sur les grumes atypiques, YOLOv8n obtient une moyenne légèrement supérieure. Cependant, YOLO11n reste très proche et présente un comportement plus équilibré entre les formes circulaires et atypiques. YOLO26n est systématiquement inférieur sur les deux groupes les plus fiables.

Les résultats associés aux grumes carrées doivent être interprétés avec prudence, car cette catégorie ne contient qu'une seule image. Les scores élevés ne permettent donc pas de conclure sur la généralisation du modèle à ce type morphologique. Cette catégorie est conservée à titre indicatif, mais elle relève davantage de l'observation qualitative que d'une analyse statistique robuste.

En synthèse, l'expérience 6 confirme le choix de YOLO11n comme meilleur candidat pour l'application mobile. Il ne s'agit pas toujours du modèle le plus performant dans chaque sous-cas, mais il présente le meilleur compromis entre performance globale sur le *test set*, robustesse au recul de cadrage simulé, qualité de segmentation de la grume et légèreté du modèle. Cette conclusion justifie son utilisation comme modèle principal pour l'intégration dans la PWA hors-ligne.

4.7 Interprétation des résultats de YOLO11n

Afin de compléter l'analyse quantitative des expériences précédentes, les sorties automatiquement générées lors de l'entraînement final de YOLO11n ont également été examinées. Ces visualisations permettent d'interpréter plus finement le comportement du modèle retenu, au-delà de la seule valeur de mAP50-95.

Les courbes présentées reposent principalement sur les notions de précision et de rappel. Dans le cadre d'une tâche de détection ou de segmentation, une prédiction peut être classée comme vraie positive, fausse positive ou fausse négative :

- **TP** (*True Positive*) : objet correctement détecté et correctement associé à sa classe ;
- **FP** (*False Positive*) : objet prédit par le modèle alors qu'il ne correspond pas à une annotation réelle correcte ;
- **FN** (*False Negative*) : objet réel présent dans l'image, mais non détecté par le modèle.

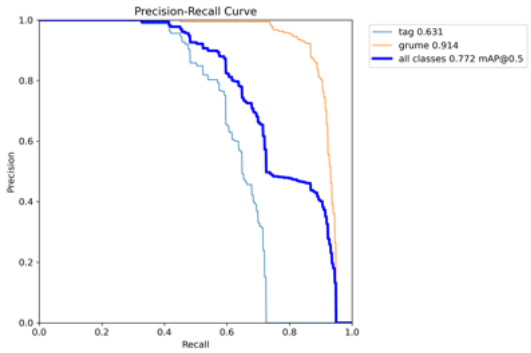
La précision mesure la proportion de prédictions correctes parmi toutes les prédictions produites par le modèle :

$$\text{Précision} = \frac{TP}{TP + FP} \tag{2}$$

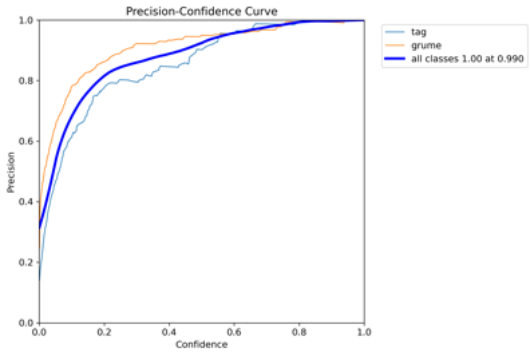
Le rappel mesure la capacité du modèle à retrouver les objets réellement présents dans l'image :

$$\text{Rappel} = \frac{TP}{TP + FN} \tag{3}$$

Une précision élevée signifie que le modèle produit peu de fausses détections, tandis qu'un rappel élevé indique qu'il oublie peu d'objets réels. Ces deux métriques sont complémentaires : augmenter le seuil de confiance tend généralement à améliorer la précision, mais peut réduire le rappel, car seules les prédictions les plus sûres sont conservées.



(a) Courbe précision-rappel.



(b) Courbes précision-confiance.

FIGURE 18 – Courbes de performance lors de l'entraînement final de YOLO11n.

La figure 18 montre que les courbes associées à la classe `grume` restent globalement plus élevées et plus régulières que celles associées à la classe `tag`. Cette différence est cohérente avec la nature des objets : la grume occupe généralement une surface plus importante dans l'image, tandis que le tag est plus petit, parfois éloigné ou partiellement moins lisible.

La courbe précision-rappel indique que le modèle conserve une bonne précision sur une large partie de la plage de rappel, surtout pour la classe `grume`. La courbe précision-confiance montre quant à elle que la précision augmente lorsque le seuil de confiance devient plus strict. Ce comportement est intéressant pour l'application, car le seuil pourra être ajusté selon le compromis recherché entre nombre de détections et fiabilité des prédictions.

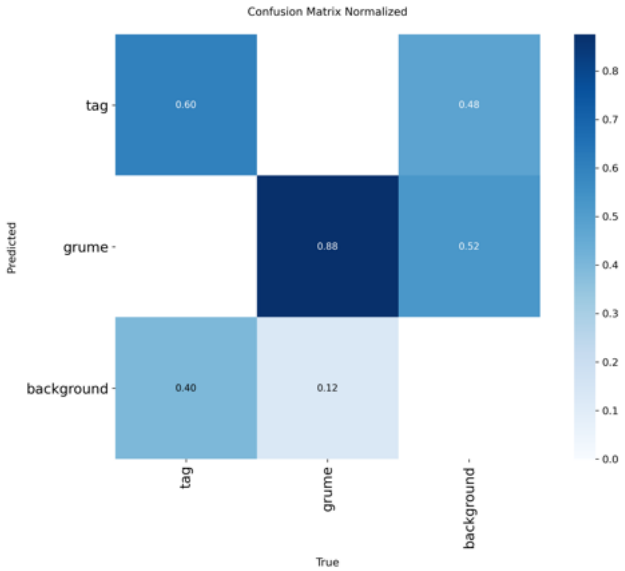


FIGURE 19 – Matrice de confusion normalisée du modèle YOLO11n.

La matrice de confusion normalisée de la figure 19 résume les principales erreurs du modèle. La diagonale correspond aux prédictions correctes, tandis que les autres cases indiquent les confusions avec une autre classe ou avec le fond.

Elle montre que la classe `grume` est mieux reconnue que la classe `tag`. Les erreurs

restantes concernent surtout les tags très petits et certaines zones de fond confondues avec des objets, ce qui reste cohérent avec la difficulté du *validation set*.

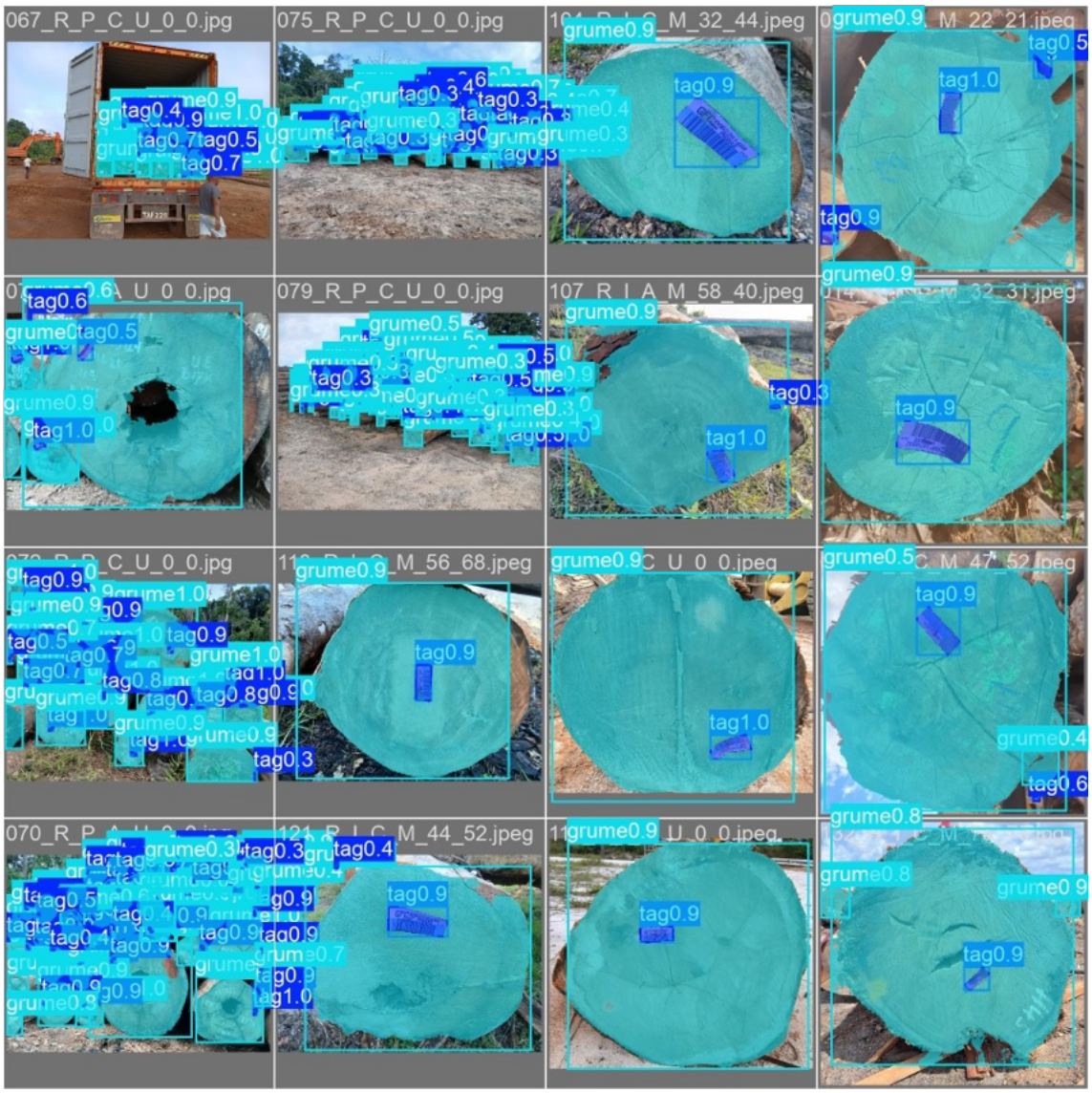


FIGURE 20 – Exemples de prédictions de YOLO11n sur un lot du *validation set*.

La figure 20 illustre quelques prédictions sur le *validation set*. Les grumes individuelles semblent globalement bien localisées, tandis que les scènes de piles restent plus difficiles lorsque les objets sont petits ou partiellement masqués.

Ces exemples montrent que le *validation set* constitue un contexte exigeant, avec des piles, des tags de faible taille et des arrière-plans encombrés. Cela peut expliquer une partie de l'écart observé avec le *test set*, plus proche du scénario principal d'utilisation.

Cette analyse complémentaire soutient le choix de YOLO11n, tout en rappelant que les tags très petits et les scènes de piles complexes restent des cas limites à valider plus largement sur le terrain.

5 Pipeline ONNX

5.1 Architecture logicielle et gestion du cache hors-ligne

Pour réaliser la PWA (Progressive Web App), nous nous sommes appuyés sur une architecture robuste et sur des technologies modernes nous permettant de répondre aux fortes contraintes d'un environnement forestier dépourvu de connexion réseau. Nous avons donc choisi de développer la PWA avec la librairie React.js combinée à TypeScript. Ce choix de technologies nous permet de construire une interface utilisateur (UI) fluide tout en garantissant une robustesse de code grâce au typage statique fourni par TypeScript. L'application devant pouvoir s'exécuter hors ligne, nous avons utilisé l'outil de construction front-end Vite, dont le plugin très performant vite-plugin-pwa facilite le développement d'une PWA en créant le Service Worker et la stratégie de mise en cache nécessaire au mode hors-ligne.

5.1.1 Structuration et outils de compilation

Nous avons structuré la partie frontend en ayant deux objectifs en tête : garder une structure et un code clairs tout en respectant les standards React et séparer la partie liée aux calculs lourds pour que l'application reste fluide :

```
grume-app-vite/
├── public/ ..... Fichiers statiques et assets globaux
│   └── model/ ..... Modèles ONNX
└── src/ ..... Code source de l'application
    ├── components/  Composants d'interface réutilisables (WebcamComponent,
    │   etc.)
    ├── context/ ..... Gestion de l'état global (PredictionProvider)
    ├── pages/  ... Vues applicatives principales (CameraTab, ResultPrediction)
    ├── utils/ ..... Bibliothèque algorithmique géométrique (functions.ts)
    └── workers/ ..... Threads d'arrière-plan (yolo.worker.ts)
```

Modèle d'inférence : Le dossier public/model contient les modèles de détection convertis au format .onnx. Ce dossier public/model peut donc contenir plusieurs modèles, mais l'application n'en charge et n'en utilise qu'un seul. Nous avons donc sélectionné notre modèle offrant les meilleures performances en termes de détection et de précision.

Le Web Worker (.worker.ts) L'environnement JavaScript est single-thread. C'est-à-dire que le fil d'exécution principal qu'on appelle Main Thread gère le cycle de vie des composants React, les événements utilisateurs et le rendu de l'interface utilisateur. Notre modèle de machine learning YOLO a besoin d'une certaine puissance de calcul qui fait que si notre modèle était exécuté sur le thread principal, cela provoquerait un blocage total de l'interface utilisateur.

Pour résoudre ce problème, nous utilisons l'API des Web Workers [36] dans notre projet. Un fichier de type Web Worker se termine avec l'extension '.worker.' indiquant à l'outil de construction (Vite) qu'il doit créer un script indépendant qui sera

instancié par la suite par l'application et exécuté dans un thread en arrière-plan, en parallèle de l'exécution du thread principal.

Le fichier yolo.worker.ts gère donc l'exécution asynchrone des tâches d'inférence. Il reçoit l'image envoyée par le thread principal (depuis le composant Webcam-Component) via un système de messagerie 'workerRef.current ?.postMessage(type : "PREDICT", payload : bitmap , [bitmap]) ;'.

Ensuite le 'worker' prend en charge le prétraitement, l'exécution de la session ONNX Runtime Web et le post-traitement avant de renvoyer les coordonnées des 'bounding boxes' à dessiner sur le canvas. Cette parallélisation de thread permet à la UI de garder une bonne fluidité sans la bloquer.

Le fichier functions.ts contient l'ensemble des fonctions géométriques (conversion métrique via le tag, calcul des diamètres, ...) qui seront détaillées par la suite.

5.1.2 Stratégie de mise en cache via Service Workers

La contrainte principale de ce projet est l'utilisation de la web app en mode hors-ligne. Pour y parvenir, c'est le Service Worker qui va s'occuper de la gestion du cache. La génération de ce Service Worker est entièrement automatisée par le plugin vite-plugin-pwa [37], qui s'appuie sur la librairie Workbox [38] développée par Google.

Gestion du cache via le Service Worker : L'approche utilisée est une stratégie de precaching. Lors du premier chargement de l'application via une connexion réseau, l'application va télécharger et stocker localement dans le cache toutes les ressources nécessaires comme les fichiers HTML, CSS, JS et les icônes. La liste complète des fichiers à garder en mémoire est inscrite dans le fichier sw.js. Cela garantit à l'application un fonctionnement hors ligne, une fois l'opérateur sur le terrain.

Nous avons dû apporter une modification dans la gestion du cache pour pouvoir intégrer les modèles de détection YOLO dans l'application côté client :

- Nous avons ajusté la configuration du Workbox pour forcer le processus de precaching à intégrer les fichiers .onnx et .wasm en ajoutant ces extensions dans la liste globalPatterns.
- Nous avons dû modifier la valeur du champ maximumFileSizeToCacheInBytes car par défaut, Workbox refuse de mettre en cache des fichiers de plus de 2 Mo pour protéger l'espace de stockage de l'appareil. Cette valeur a donc été augmentée en raison de la taille volumineuse de nos modèles au format .onnx :

```
workbox: {
  globPatterns:
     ['**/*.{js,css,html,ico,png,svg,wasm,onnx}'],
  maximumFileSizeToCacheInBytes: 100000000
}
```

Les ajustements que nous avons apportés garantissent que les modèles et les fichiers d'exécution sont téléchargés lors de l'installation de l'application.

5.2 Pipeline ONNX : exportation, intégration et prétraitement

Pour exécuter un modèle d'IA dans un navigateur web, il est nécessaire qu'il soit léger, rapide et compatible. C'est tout l'intérêt du pipeline ONNX que nous avons mis en place. Il permet de transformer un modèle entraîné avec PyTorch sous un format compatible avec JavaScript.

5.2.1 Exportation du modèle et choix de la précision (FP32)

Nous avons dans un premier temps exporté le modèle *PyTorch* vers le format ONNX grâce à un script Python. Pour que ce modèle soit compatible avec les contraintes d'une PWA devant s'exécuter sur smartphone, une optimisation a été appliquée dans le script d'exportation :

- `simplify=True` : réduit la complexité structurelle du graphe ONNX en fusionnant les opérations redondantes, ce qui permet d'accélérer l'inférence.

- **Utilisation de la précision FP32** : nous avons maintenu le choix de la précision float32 (FP32) pour garantir des performances optimales et respecter notre délai de livraison du projet. La raison principale pour laquelle nous avons choisi de garder la précision en float32 est que l'environnement WebAssembly ne gère pas nativement le float16 (FP16), ce qui aurait forcé le moteur d'inférence ONNX Runtime Web à faire des conversions (casting) entre float16 et float32 et aurait entraîné une dégradation de la latence. Ensuite, une transition tardive vers le float16 nous aurait obligés à apporter des modifications dans le code. Donc conserver la précision float32 était notre choix le plus judicieux :

```
IMGSZ = 640
OPSET = 17
DYNAMIC = False
SIMPLIFY = True

exported = model.export(
    format="onnx",
    imgsz=IMGSZ,
    opset=OPSET,
    dynamic=DYNAMIC,
    simplify=SIMPLIFY,
)
```

5.2.2 Intégration et initialisation (ONNX Runtime Web)

Le modèle d'inférence est chargé côté client au sein du Web Worker (le fichier yolo.worker.ts). Comme expliqué dans la section de l'état de l'art, c'est ici que la

création et l'exécution de la session d'inférence ainsi que l'ordre des *execution providers* sont définis :

```
executionProviders: ["webgpu","webgl", "wasm"]
session = await ort.InferenceSession.create(MODEL_PATH, {
   executionProviders});
```

Cet ordre tente d'abord d'utiliser la carte graphique du smartphone de l'utilisateur via l'*execution provider* `"webgpu"`. Si le matériel ne supporte pas cette technologie, alors ONNX Runtime Web tente d'utiliser `"webgl"`. Et si ce dernier n'est lui aussi pas disponible, alors il se rabat sur le processeur via `"wasm"` (WebAssembly).

5.2.3 Prétraitement de l'image

Une fois l'image capturée et envoyée au worker, elle doit subir un prétraitement pour être convertie en une structure de données qu'on appelle `tenseur` (tableau multidimensionnel). Ce traitement est réalisé dans la fonction `processImageData`.

1. **Redimensionnement** : notre modèle YOLO reçoit une image carrée de taille 640x640 pixels. Pour éviter que l'image ne soit déformée lors du redimensionnement de celle-ci, on applique un `scale ratio` qui permet de préserver les proportions d'origine de l'image. Cette dernière est ensuite dessinée au centre d'un canvas noir ('canvas') créé au préalable avec des marges horizontales et verticales.
2. **Extraction et normalisation** : l'étape suivante est la normalisation qui consiste à extraire les pixels de notre image et obtenir la valeur des couleurs de chaque pixel (entre 0 et 255) pour ensuite les diviser par 255.0 et obtenir une valeur flottante de ces couleurs comprise entre 0.0 et 1.0.
3. **Tenseur** : finalement, on organise les valeurs pour passer d'un format Rouge, Vert, Bleu successifs à un format où tous les rouges sont les 640x640 premiers éléments du tableau, ensuite tous les verts et finalement tous les bleus. On crée donc un format à plat qui est requis par YOLO. Le résultat est ensuite enregistré dans un objet `ort.Tensor` de type float32 et de dimension [1, 3, size, size] où :
   - **1** représente le nombre d'images dans le batch ;
   - **3** représente les canaux de couleurs R, V, B ;
   - **640x640** représente la hauteur et largeur de l'image.

```
        const size = 640;
        const canvas = new OffscreenCanvas(size, size);
        const ctx = canvas.getContext("2d", {
           willReadFrequently: true })!;

        function preprocessImageData(image: ImageBitmap) {

        ctx.fillStyle = "black";
        ctx.fillRect(0, 0, size, size);

        const scale = Math.min(size / image.width, size /
           image.height);
        const newWidth = image.width * scale;
        const newHeight = image.height * scale;
        const padX = (size - newWidth) / 2;
        const padY = (size - newHeight) / 2;

        ctx.drawImage(image, padX, padY, newWidth, newHeight);

        const resizedData = ctx.getImageData(0, 0, size,
           size).data;
        const floatArray = new Float32Array(3 * size * size);

        for (let i = 0; i < size * size; i++) {
          floatArray[i] = resizedData[i * 4] / 255.0;
          floatArray[i + size * size] = resizedData[i * 4 +
             1] / 255.0;
          floatArray[i + 2 * size * size] = resizedData[i * 4
             + 2] / 255.0;
        }

        return { tensor: new ort.Tensor("float32",
           floatArray, [1, 3, size, size]), scale, padX, padY
           };
    }
```

Notre tenseur est envoyé à la fonction session.run() qui démarre l'inférence du modèle et retourne les données de détection :

```
const results = await session.run({ images: tensor });
```

5.3 Post-traitement des détections brutes

5.3.1 Filtrage par Intersection sur Union (IoU) et Suppression des Non-Maximums (NMS)

Une fois que la détection par IA est terminée, le tenseur contient de nombreuses bounding boxes. Puisque YOLO détecte souvent un même objet (tag ou grume) plusieurs fois, il y a un amas de bounding boxes qui se superposent. Pour éliminer ces doublons et ne garder que la plus précise, il faut faire un post-traitement :

1. Intersection sur Union (IoU) : la fonction calculateIoU détermine le degré de chevauchement entre deux bounding boxes. L'aire de leur intersection est ensuite calculée et divisée par l'aire de leur union. La valeur obtenue est un nombre compris entre 0 (= il n'y a aucun chevauchement et les deux bounding boxes sont parfaitement distinctes) et 1 (= signifie une superposition parfaite des deux bounding boxes) :

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

2. Suppression des Non-Maximums (NMS) : la fonction `applyNSM` s'appuie sur le résultat de la fonction `calculateIoU` pour filtrer les prédictions dupliquées et ne conserver que la plus précise. L'algorithme se déroule comme suit :
   (a) Il trie les détections par ordre décroissant en fonction de leur probabilité de certitude.
   (b) Il sélectionne la détection qui a la plus haute probabilité et la valide.
   (c) Il compare ensuite cette détection qu'il a validée avec les autres bounding boxes de la même classe (tag ou grume) qui sont inférieures dans la liste triée. Si l'IoU entre la bounding box validée et une autre bounding box dépasse un certain seuil (0.45 dans notre code, la valeur par défaut fournie par YOLO), la bounding box avec la probabilité inférieure est supprimée car elle est considérée comme un doublon.

Ce processus permet d'éliminer les faux positifs et les erreurs de comptage.

6 Inférence embarquée et calcul métrologique

6.1 Algorithme de post-inférence

Une fois que l'inférence a été réalisée, on récupère des données qu'il va falloir manipuler pour obtenir des mesures en cm que nous pourrons utiliser. Dans le domaine du machine learning et de la vision par ordinateur, c'est souvent la bibliothèque OpenCV [39] qui est utilisée pour faire du traitement géométrique sur image. Cependant, nous avons choisi de ne pas importer la librairie OpenCV.js dans notre Progressive Web App. L'intégration de cette librairie aurait ajouté un fichier très volumineux qui risquerait de saturer l'espace limité du cache du Service Worker et ralentirait le chargement hors ligne. Pour remédier à ce problème, nous avons fait le choix d'implémenter nous-mêmes ces algorithmes en TypeScript (dans le fichier `functions.ts`). Pour écrire ces algorithmes complexes de façon optimale nous nous sommes appuyés sur l'intelligence artificielle, `'Gemini 3.1 Pro'`, en tant qu'assistant de développement.

6.1.1 Conversion pixels/cm

Pour parvenir à mesurer le diamètre d'une grume, il faut établir une échelle sur la base d'un tag de référence apposé au préalable sur la grume. Les dimensions de ce tag sont connues à l'avance et sont fixes : 4.0 cm pour la largeur et 9.5 cm pour la hauteur.

Pour ne pas utiliser la largeur et la hauteur du tag qui pourraient être déformées, abîmées lors de la pose ou même avoir subi une distorsion liée à l'angle de vue ou à l'inclinaison, nous utilisons la diagonale que nous calculons via le théorème de Pythagore :

```
const TAG_WIDTH_CM = 4.0;
const TAG_HEIGHT_CM = 9.5;
const TAG_DIAGONAL_CM = Math.sqrt(TAG_WIDTH_CM *
   TAG_WIDTH_CM + TAG_HEIGHT_CM * TAG_HEIGHT_CM);
```

Une fois que le tag est détecté par le modèle, on calcule sa diagonale en pixels via la fonction mathématique `Math.hypot()`. Le ratio pixels par cm est ensuite déterminé en divisant la diagonale en pixels par la diagonale réelle en cm :

```
const diagonalPx = Math.hypot(tag.rect.width,
   tag.rect.height);
const pixelsPerCm = diagonalPx / TAG_DIAGONAL_CM;
```

6.1.2 Mesure du diamètre

Une fois que les objets ont été détectés par le modèle, nous obtenons des masques de segmentation avec des contours irréguliers. Pour déterminer avec précision les dimensions de ces objets (le tag et la grume), la méthode se divise en plusieurs étapes et adapte sa logique de calcul en fonction de l'objet analysé.

1. Enveloppe convexe (Convex Hull [40]) : L'algorithme extrait les pixels du masque et calcule l'enveloppe convexe via l'algorithme de la chaîne monotone (`Monotone Chain Algorithm`) dans notre fonction getConvexHull. Cette opération permet d'améliorer les contours irréguliers du masque de segmentation.
2. Calcul de la bounding box : Au départ, la recherche du rectangle minimal qui englobe l'objet (Minimum Bounding Box) était appliquée à la fois pour les tags et pour les grumes. Par la suite nos tests ont montré que pour des grumes aux formes ovales ou atypiques, la recherche du rectangle minimal donnait une orientation erronée de la bounding box. Cette méthode ignorait la plupart du temps l'axe le plus large du tronc ce qui donnait un diamètre maximal incorrect.

   Pour éviter cette situation, nous avons divisé notre algorithme `getMaskData` en deux pour traiter l'objet analysé en fonction de sa classe :

   - Pour le Tag (Surface minimale) : Le tag ayant une forme rectangulaire, nous appliquons la fonction `getTagBoundingBox`. Cette dernière s'inspire de la méthode des Rotating Calipers [41]. L'algorithme teste différentes inclinaisons de la grume sur une certaine plage, -90° à +90° tous les 2 degrés, dessine un rectangle autour de ces pixels et calcule ensuite la surface de ce rectangle. L'angle qui permet d'obtenir le rectangle le plus petit englobant le masque est le Minimum Bounding Box.

   - Pour les grumes (Diamètre maximal) : La fonction `getGrumeBoundingBox` commence par chercher les deux points les plus éloignés l'un de l'autre. Le segment reliant ces deux points est alors l'axe principal de la grume, le diamètre d1. Ensuite pour trouver le second diamètre (d2), l'algorithme mesure tout simplement la largeur maximale de la grume perpendiculairement à cet axe d1. Grâce à cette méthode, les mesures respectent la forme de la grume.
3. Une fois les dimensions de la bounding box trouvées, ces dimensions en pixels sont divisées par le ratio `pixelsPerCm` établi précédemment. Le diamètre final de la grume est finalement estimé en calculant la moyenne des deux diamètres : $D = (d1 + d2)/2$

6.1.3 Association Tag-Grume (Ray Casting)

Lors de l'inférence, plusieurs grumes et tags peuvent se trouver simultanément sur la même image. Il faut donc pouvoir attribuer le bon tag à la bonne grume pour pouvoir déterminer le bon ratio pixelsPerCm et calculer les bonnes mesures.

Pour cela, nous avons implémenté le test du `Point-in-Polygone` qui implémente l'algorithme du `Ray-Casting` [42]. Ce test détermine si un point (ici le tag) se trouve à l'intérieur d'un polygone (l'enveloppe convexe de la grume). Et si oui, il associe le tag à la bonne grume.

L'algorithme du `Ray-Casting` trace une droite horizontale vers la droite jusqu'à l'infini et compte le nombre de fois où la droite horizontale coupe un côté du polygone. Si le nombre d'intersections avec le polygone est impair, cela signifie que le

point (le centre du tag) se trouve dans le polygone. Si c'est pair, cela signifie que le point se trouve à l'extérieur du polygone :

```
function pointPolygonTest(pt: {x:number,y:number}, poly:
   {x:number,y:number}[]): number {
  let inside = false;

  // Ray-Casting algorithm
  for (let i = 0, j = poly.length - 1; i < poly.length; j =
     i++) {
      const xi = poly[i].x, yi = poly[i].y;
      const xj = poly[j].x, yj = poly[j].y;
      const intersect = ((yi > pt.y) !== (yj > pt.y)) &&
         (pt.x < (xj - xi) * (pt.y - yi) / (yj - yi) + xi);
      if (intersect) inside = !inside;
  }

  // Compute the minimum distance to the segment
  let minD = Infinity;
  for (let i = 0, j = poly.length - 1; i < poly.length; j =
     i++) {
      minD = Math.min(minD, pointToSegmentDist(pt, poly[j],
         poly[i]));
  }
  return inside ? minD : -minD;
}
```

Finalement, pour éviter d'éventuelles erreurs liées au mauvais positionnement physique du tag sur la grume ou à un masque de segmentation incomplet, la distance entre le centre du tag et le segment de l'enveloppe convexe est calculée avec une marge de tolérance. L'association entre le tag et la grume est donc validée, même si le tag se trouve jusqu'à 10 pixels à l'extérieur du masque de la grume :

```
      if (pointPolygonTest(tag.rect.center, grume.hull) >=
         -10) {
        if (grume.area > maxArea) {
            maxArea = grume.area;
            matchedGrume = grume;
        }
      }
```

6.2 Validation Métrique

Malgré la précision de notre modèle d'inférence, les conditions sur le terrain peuvent varier (changement de luminosité, ombres, écorces abîmées, grumes en partie cachées). Pour permettre de corriger les erreurs liées à ces conditions imprévisibles, l'application met à disposition un outil dans une interface dédiée permettant à l'opérateur d'appliquer des corrections manuelles :

- Ajustement géométrique : Si la bounding box d'un tag ou d'une grume est légèrement décalée, trop grande/petite ou même mal orientée à cause d'une mauvaise détection, l'opérateur a la possibilité de la redimensionner, la déplacer et de modifier son orientation.

- Recalcul dynamique : Toute modification effectuée sur les bounding boxes provoque une mise à jour des mesures des diamètres de la bounding box. Si la bounding box du tag est modifiée, le ratio pixelsPerCm est recalculé. De même que lorsque la bounding box d'une grume est modifiée, les mesures de ses diamètres sont mises à jour en temps réel.

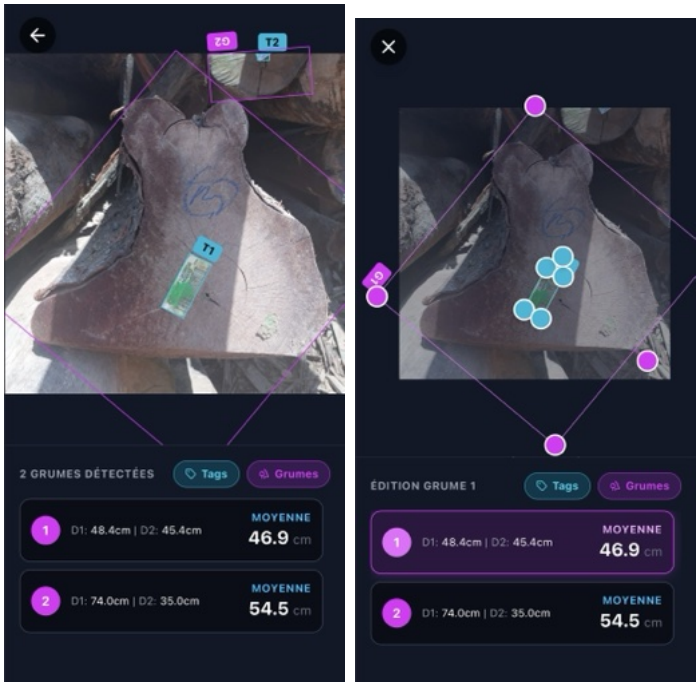


FIGURE 21 – Illustration de l'outil d'édition sur grume atypique

Dans le cas particulier d'une grume de forme carrée ou rectangulaire, l'algorithme a tendance à orienter la bounding box sur l'axe le plus long (la diagonale). Puisque cette orientation fausse les mesures de diamètres, il est nécessaire de réorienter la bounding box manuellement pour obtenir les bonnes mesures :

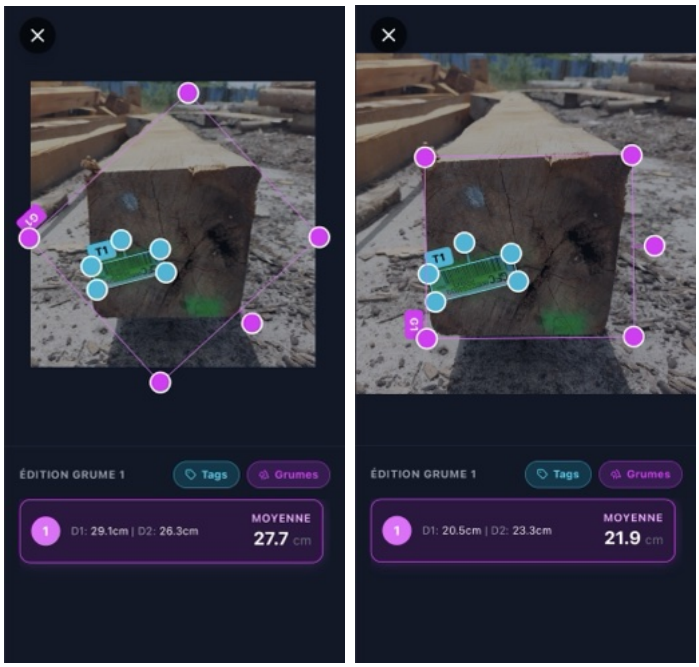


FIGURE 22 – Illustration de l'outil d'édition sur une grume carrée

7 Évaluation des mesures produites par l'application

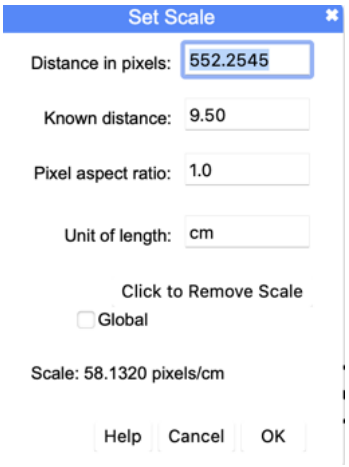
Lors de notre phase de développement et de test, nous avons constaté que les mesures de diamètre fournies avec le dataset d'images de grumes étaient incorrectes et ne reflétaient pas la réalité pour certaines images de grumes. Nous avons donc pris l'initiative de remesurer nous-mêmes ces grumes pour pouvoir effectuer une comparaison avec notre application.

Pour cela, nous avons utilisé le logiciel d'analyse d'image en appliquant le protocole de mesure forestier suivant :

1. Nous commençons par mesurer la longueur du tag de référence (en pixels) sur l'image en renseignant la longueur connue du tag en centimètres. Le logiciel calcule automatiquement le ratio pixel/cm pour l'image.
2. Mesure du diamètre 1 : Nous traçons la droite passant par les deux points les plus éloignés de la grume en prenant soin de ne pas inclure l'écorce. Cette droite correspond au diamètre le plus grand de la grume (en pixels). Le logiciel effectue la conversion en cm grâce au ratio calculé précédemment.
3. Mesure du diamètre 2 : Nous traçons ensuite la plus grande droite strictement perpendiculaire à l'axe du premier diamètre d1. Pour s'assurer que cette seconde droite est bien perpendiculaire à l'axe du premier diamètre, nous vérifions qu'il y a bien une différence de 90 degrés entre les angles des droites. Là aussi le logiciel effectue la conversion en centimètres.

FIGURE 23 – Mesures initiales des diamètres de grume : d1=43cm, d2=38cm

(a) Mesure du tag en pixels sur l'image



(b) Calcul du ratio pixel/cm (longueur en cm connue)

FIGURE 24 – Étapes de mesure

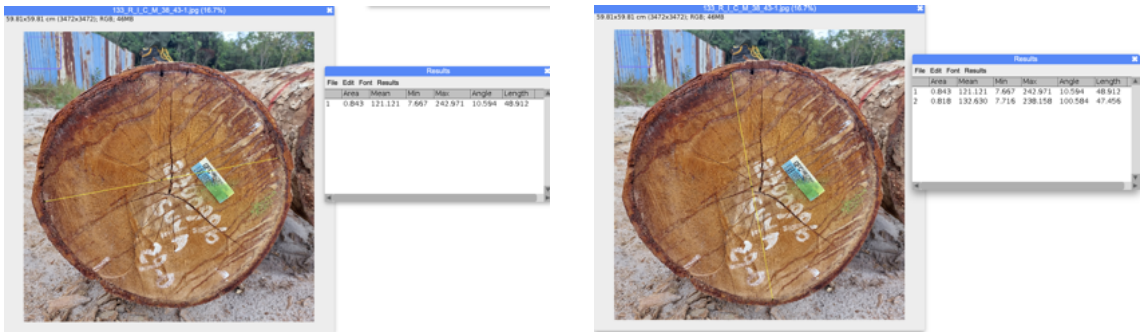


FIGURE 25 – Mesure des diamètres d_1 et d_2 de la grume

Après la sélection de YOLO11n comme modèle principal, le pipeline complet de mesure a été évalué sur les 16 images du *test set*. Cette évaluation porte sur la mesure finale produite par l'application, après segmentation de la grume, segmentation du tag, conversion pixel-centimètre et estimation géométrique du diamètre.

Pour chaque image, deux mesures manuelles de référence sont disponibles, notées D_1 et D_2. L'application produit également deux estimations, notées $\hat{D}_1$ et $\hat{D}_2$. Afin de résumer chaque grume par une seule valeur, le diamètre moyen réel D_m et le diamètre moyen prédit $\hat{D}_m$ sont calculés comme suit :

$$D_m = \frac{D_1 + D_2}{2} \tag{4}$$

$$\hat{D}_m = \frac{\hat{D}_1 + \hat{D}_2}{2} \tag{5}$$

L'erreur de mesure est définie comme la différence entre la valeur prédite et la valeur réelle :

$$e_i = \hat{D}_i - D_i \tag{6}$$

Une erreur positive indique une surestimation du diamètre, tandis qu'une erreur négative indique une sous-estimation. Les performances sont résumées à l'aide de quatre métriques : la MAE, la RMSE, le biais et la MAPE.

La MAE mesure l'erreur absolue moyenne :

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |\hat{D}_i - D_i| \tag{7}$$

La RMSE pénalise davantage les erreurs importantes :

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{D}_i - D_i)^2} \tag{8}$$

Le biais indique si le système tend globalement à surestimer ou à sous-estimer les mesures :

$$Biais = \frac{1}{n}\sum_{i=1}^{n}(\hat{D}_i - D_i) \tag{9}$$

Enfin, la MAPE exprime l'erreur absolue moyenne en pourcentage de la valeur réelle :

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{\hat{D}_i - D_i}{D_i}\right| \tag{10}$$

Le tableau 9 présente la synthèse des erreurs obtenues sur D_1, D_2 et sur le diamètre moyen D_m.

TABLE 9 – Synthèse des erreurs de mesure de l'application sur le *test set*.

Grandeur	**MAE (cm)**	**RMSE (cm)**	**Biais (cm)**	**MAPE (%)**	**Erreur max. (cm)**	**Erreur médiane (cm)**
D_1	0.94	1.23	-0.74	2.11	2.64	0.61
D_2	0.76	1.12	-0.37	2.02	2.71	0.30
D_m	0.81	1.06	-0.55	1.97	2.35	0.56

Les résultats sont encourageants. Sur le diamètre moyen D_m, l'erreur absolue moyenne est de 0.81 cm et la RMSE atteint 1.06 cm. L'erreur relative moyenne est de 1.97%, ce qui montre que les mesures produites par l'application restent proches des mesures physiques de référence.

Le biais moyen de -0.55 cm indique une légère tendance à sous-estimer les diamètres. Cette sous-estimation reste limitée et peut s'expliquer par la stratégie d'annotation, qui privilégie la section utile du bois plutôt que le contour externe complet de l'écorce, ainsi que par les imprécisions possibles dans la segmentation du tag ou du contour de la grume.

Les erreurs observées sur D_1 et D_2 restent du même ordre de grandeur. Cela indique que le système ne dépend pas uniquement d'un axe particulier de la section et que l'estimation géométrique reste globalement cohérente.

Ces résultats doivent toutefois être interprétés avec prudence. Le *dataset* d'entraînement reste réduit et le *test set* ne contient que 16 images. Dans ce contexte, obtenir une erreur moyenne inférieure au centimètre sur le diamètre moyen constitue néanmoins un résultat satisfaisant pour un prototype.

Enfin, PEPPS n'a pas communiqué de seuil d'erreur maximal acceptable pour l'usage terrain. L'objectif était donc de produire la meilleure estimation possible dans les contraintes du projet, plutôt que de valider l'application par rapport à une tolérance opérationnelle prédéfinie.

8 Conclusion

Ce mémoire avait pour objectif d'évaluer la faisabilité d'une application mobile hors-ligne capable de mesurer automatiquement le diamètre de grumes à partir d'une photographie terrain. Le travail s'inscrit dans le contexte du projet dWTS, où les agents doivent pouvoir collecter des mesures fiables en forêt, sans dépendre d'une connexion réseau.

Un *dataset* spécifique a été constitué, annoté et utilisé pour comparer plusieurs modèles YOLO de segmentation d'instance. Les expériences menées montrent que YOLO11n constitue le meilleur compromis pour le prototype développé, en combinant de bonnes performances de segmentation, une taille réduite et une latence compatible avec une intégration mobile.

Le pipeline complet, depuis la détection de la grume et du tag jusqu'au calcul géométrique du diamètre, fournit des résultats encourageants. Sur le *test set* réel, l'erreur absolue moyenne obtenue sur le diamètre moyen est de 0.81 cm, avec une erreur relative moyenne de 1.97 %. Ces résultats restent à interpréter avec prudence, car le nombre d'images mesurées demeure limité.

Ce travail ne constitue donc pas une solution opérationnelle définitive, mais il démontre la faisabilité technique d'une mesure automatique hors-ligne. Les perspectives principales concernent l'élargissement du *dataset*, la validation sur davantage de grumes mesurées, les tests directs sur smartphone Android et l'amélioration de la robustesse dans les cas difficiles, notamment les tags très petits, les occlusions et les piles de grumes.

Utilisation de l'intelligence artificielle

Des outils d'intelligence artificielle, notamment des modèles de langage, ont été utilisés au cours de ce travail comme aide ponctuelle à la rédaction, à la programmation et à la compréhension de certaines ressources scientifiques.

Ces outils ont principalement servi à corriger l'orthographe et la formulation de certains passages, à améliorer la clarté du texte, à reformuler des explications techniques et à accélérer la lecture exploratoire de certains articles scientifiques. Ils ont également été utilisés comme support lors du développement, notamment pour aider à structurer certains scripts, comprendre des erreurs de code, proposer des pistes de débogage ou identifier des fonctions utiles.

L'intelligence artificielle a donc été employée comme un outil d'assistance, au même titre qu'un correcteur linguistique, un moteur de recherche ou une aide au développement. Elle n'a pas remplacé le travail scientifique réalisé dans ce mémoire. Les choix méthodologiques, la constitution du *dataset*, les annotations, les expériences, l'interprétation des résultats et les décisions finales relèvent de notre propre analyse.

Les contenus générés ou suggérés par ces outils ont été relus, vérifiés et adaptés avant d'être intégrés au mémoire. L'utilisation de l'IA a ainsi contribué à améliorer l'efficacité du travail, sans se substituer à la démarche scientifique, technique et critique menée dans le cadre de ce projet.

Références

[1] J. Rondeux, *La mesure des arbres et des peuplements forestiers*. Les Presses Agronomiques de Gembloux, 2021.

[2] F. de Miguel-Díez et al., "Further Application of Using a Personal Laser Scanner and Simultaneous Localization and Mapping (SLAM) Technology to Estimate the Log's Volume and Its Comparison with Traditional Methods," *International Journal of Applied Earth Observation and Geoinformation*, t. 109, p. 102 779, 2022. doi : 10.1016/j.jag.2022.102779.

[3] M. D. Niţă et S. A. Borz, "Accuracy of a Smartphone-Based Freeware Solution and Two Shape Reconstruction Algorithms in Log Volume Measurements," *Computers and Electronics in Agriculture*, t. 205, p. 107 653, 2023. doi : 10.1016/j.compag.2023.107653.

[4] Y. J. Sandvik, C. M. Futsæther, K. H. Liland et O. Tomic, "A Comparative Literature Review of Machine Learning and Image Processing Techniques Used for Scaling and Grading of Wood Logs," *Forests*, t. 15, n° 7, p. 1243, 2024. doi : 10.3390/f15071243.

[5] F. Longuetaud et al., "Traçabilité et évaluation de la qualité des bois ronds à partir de l'analyse d'images d'extrémités," INRAE, rapp. tech., 2022, Projet ANR TreeTrace, HAL-04217998.

[6] J. Zheng, J. Zheng, S. Zhang et H. Yu, "Segmentation Method for Whole Vehicle Wood Detection Based on Improved YOLACT Instance Segmentation Model," *IEEE Access*, t. 11, p. 81 434-81 448, 2023. doi : 10.1109/ACCESS.2023.3300900.

[7] D. Bhatt et al., "CNN Variants for Computer Vision : History, Architecture, Application, Challenges and Future Scope," *Electronics*, t. 10, n° 20, p. 2470, 2021. doi : 10.3390/electronics10202470.

[8] A. Amidi et S. Amidi, *CS 230 – Convolutional Neural Networks Cheatsheet*, Consulté le 2026-04-12, Stanford University. adresse : https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks/.

[9] M. Carratù, V. Gallo, C. Liguori, A. Pietrosanto, M. O'Nils et J. Lundgren, "A CNN-Based Approach to Measure Wood Quality in Timber Bundle Images," in *Proceedings of the 2020 IEEE International Instrumentation and Measurement Technology Conference*, 2020.

[10] J. V. C. Mazzochin, E. M. F. Diniz, G. A. Oliveira et É. O. Rodrigues, "Mobile Software for Automated Segmentation, Counting, and Management of Wood Logs," *Revista Caderno Pedagógico*, t. 21, n° 8, p. 1-21, 2024. doi : 10.54033/cadpedv21n8-128.

[11] J. V. C. Mazzochin et al., "A Novel Approach for the Counting of Wood Logs Using cGANs and Image Processing Techniques," *Forests*, t. 16, n° 2, p. 237, 2025. doi : 10.3390/f16020237.

[12] H. Li, J. Liu et D. Wang, "A Fast Instance Segmentation Technique for Log End Faces Based on Metric Learning," *Forests*, t. 14, n° 4, p. 795, 2023. doi : 10.3390/f14040795.

[13] Z. Lu et al., "A Deep Learning Method for Log Diameter Measurement Using Wood Images Based on YOLOv3 and DeepLabv3+," *Forests*, t. 15, nº 5, p. 755, 2024. DOI : 10.3390/f15050755.

[14] M. Carratù, V. t. Gallo, C. Liguori, J. Lundgren, M. O'Nils et A. Pietrosanto, "An Innovative Method for Log Diameter Measurements Based on Deep Learning," in *Proceedings of the 2023 IEEE International Instrumentation and Measurement Technology Conference*, 2023, p. 1-6.

[15] Z. Zou, K. Chen, Z. Shi, Y. Guo et J. Ye, "Object Detection in 20 Years : A Survey," *Proceedings of the IEEE*, t. 111, nº 3, p. 257-276, 2023. DOI : 10.1109/JPROC.2023.3238524.

[16] M. L. Ali et Z. Zhang, "The YOLO Framework : A Comprehensive Review of Evolution, Applications, and Benchmarks in Object Detection," *Computers*, t. 13, nº 12, p. 336, 2024. DOI : 10.3390/computers13120336.

[17] R. Sapkota et M. Karkee, "Ultralytics YOLO Evolution : An Overview of YOLO26, YOLO11, YOLOv8, and YOLOv5 Object Detectors for Computer Vision and Pattern Recognition," *arXiv preprint arXiv :2510.09653*, 2026. arXiv : 2510.09653.

[18] A. Mumuni et F. Mumuni, "Data Augmentation : A Comprehensive Survey of Modern Approaches," *Array*, t. 16, p. 100 258, 2022. DOI : 10.1016/j.array.2022.100258.

[19] Google, *TensorFlow.js : Machine Learning for the Web and Beyond*, 2024. adresse : https://www.tensorflow.org/js.

[20] Y. Ma, D. Xiang, S. Zheng, D. Tian et X. Li, "Moving Deep Learning into Web Browser : How Far Can We Go ?" In *The World Wide Web Conference*, 2019, p. 1234-1244. DOI : 10.1145/3308558.3313639.

[21] I. Mysiuk et R. Shuvar, "Comparative Analysis of Client-Side and Server-Side Machine Learning Technologies," *Electronics and Information Technologies*, t. 18, 2024. adresse : https://publications.lnu.edu.ua/collections/index.php/electronics/article/view/4468/4911.

[22] X. Pournaras et D. A. Koutsomitropoulos, "Deep Learning on the Web : State-of-the-Art Object Detection Using Web-Based Client-Side Frameworks," in *2020 11th International Conference on Information, Intelligence, Systems and Applications*, 2020, p. 1-8. DOI : 10.1109/IISA50023.2020.9284358.

[23] Mozilla Developer Network, *WebGL API*, MDN Web Docs, 2024. adresse : https://developer.mozilla.org/docs/Web/API/WebGL_API.

[24] W3C WebGPU Working Group, *WebGPU*, 2024. adresse : https://webgpu.org/.

[25] Ultralytics, *Documentation officielle Ultralytics YOLO*, 2024. adresse : https://docs.ultralytics.com/fr/.

[26] WebAssembly Community Group, *WebAssembly*, 2024. adresse : https://webassembly.org/.

[27] S. Nandanwar, "Cross-Framework Validation of CNN Architectures : From PyTorch to ONNX," mém. de mast., Florida Institute of Technology, mai 2024.

[28] P. Jajal, E. Kocinare, Y.-H. Lu, W. Jiang, J. Woo et G. K. Thiruvathukal, "Interoperability in Deep Learning : A User Survey and Failure Analysis of ONNX Model Converters," *Proceedings of the ACM*, 2024.

[29] S. Sengupta, K. Jana, N. Wu et S. Chen, "From WebGL to WebGPU : A Reality Check of Browser-Based GPU Acceleration," in *Proceedings of the ACM Web Conference*, 2024.

[30] I. J. Ratul, Y. Zhou et K. Yang, "Accelerating Deep Learning Inference : A Comparative Analysis of Modern Acceleration Frameworks," *Electronics*, t. 14, n° 2977, 2025.

[31] K. I. D. Kyriakou, "Leveraging WebAssembly and WebGPU for Efficient Integration of AI Models into Web Applications," *SSRN*, 2024.

[32] S. Chickerur, S. Balannavar, P. Hongekar, A. Prerna et S. Jituri, "WebGL vs. WebGPU : A Performance Analysis for Web 3.0," *Procedia Computer Science*, t. 233, p. 919-928, 2024.

[33] Kili Technology, *Training, Validation and Test Sets : How To Split Machine Learning Data*, Consulté le 2026-05-27, 2023. adresse : https://kili-technology.com/blog/training-validation-and-test-sets-how-to-split-machine-learning-data.

[34] S. Liu et al., "How to Discriminate Wood of CITES-Listed Tree Species from Their Look-Alikes : Using an Attention Mechanism with the ResNet Model on an Enhanced Macroscopic Image Dataset," *Frontiers in Plant Science*, 2024. doi : 10.3389/fpls.2024.1368885.

[35] A. Safonova et al., "Ten Deep Learning Techniques to Address Small Data Problems with Remote Sensing," *International Journal of Applied Earth Observation and Geoinformation*, 2023. doi : 10.1016/j.jag.2023.103569.

[36] Mozilla Developer Network, *Using Web Workers*, MDN Web Docs, 2024. adresse : https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers.

[37] Vite PWA Contributors, *Vite Plugin PWA Guide*, Documentation officielle Vite Plugin PWA, 2024. adresse : https://vite-pwa-org.netlify.app/guide/.

[38] Google Chrome Developers, *Workbox*, Documentation officielle Workbox, 2024. adresse : https://developer.chrome.com/docs/workbox.

[39] OpenCV Team, *OpenCV*, Official OpenCV Website, 2024. adresse : https://opencv.org/.

[40] Wikipédia, *Calcul de l'enveloppe convexe*, Wikipédia, l'encyclopédie libre, 2026. adresse : https://fr.wikipedia.org/wiki/Calcul_de_l%27enveloppe_convexe.

[41] G. T. Toussaint, "Solving Geometric Problems with the Rotating Calipers," in *Proceedings of IEEE MELECON*, 1983, p. 1-8.

[42] Rosetta Code, *Ray-Casting Algorithm*, 2024. adresse : https://rosettacode.org/wiki/Ray-casting_algorithm.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**