République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de 8 Mai 1945-Guelma-
Faculté des Mathématiques, d'Informatique et des Sciences de la matière
Département d'Informatique

Mémoire de fin d'études Master

Filière : Informatique

Option :

Sciences et Technologies de l'Information et de la Communication

Thème

Développement d'un système intelligent de reconnaissance des insectes nuisibles dans les cultures agricoles

Encadré par :
Dr. BOUGHIDA ADIL

Présenté par :
*F*erdi Manar

Année Universitaire 2024/2025

Remerciements

Tout d'abord, je remercie **Allah**, Le Tout-Puissant, pour m'avoir accordé la force, la patience et la persévérance tout au long de mon parcours académique. C'est à Lui que revient le mérite en premier lieu, et sans Sa volonté, rien n'aurait été possible.

Je souhaite aussi remercier **moi-même**, pour avoir persévéré malgré les difficultés, pour avoir cru en mes capacités, et pour n'avoir jamais abandonné même dans les moments les plus éprouvants.

Je tiens ensuite à exprimer mes sincères remerciements à mon encadrant, **Monsieur Adel Bouguida**, pour sa disponibilité, ses conseils avisés, et son accompagnement tout au long de ce travail. Sa bienveillance et son expertise ont grandement contribué à la réalisation de ce projet.

Je tiens également à exprimer ma profonde gratitude à ma famille : **ma mère**, **mon père** et **ma tante Fatiha**, pour leur soutien inconditionnel, leur amour et leurs prières constantes qui m'ont porté dans les moments difficiles.

Je remercie particulièrement mon frère **Fakhr El Islam** pour sa présence constante à mes côtés au fil des années, pour sa patience et sa compréhension face à mes nombreuses plaintes.

Un remerciement spécial à mon amie **DJihane**, que je considère comme une sœur, pour son soutien moral, ses encouragements et sa présence précieuse tout au long de cette dernière année décisive de mon parcours universitaire.

Je remercie également ma cousine **Basma**, pour son affection, son soutien, et sa présence qui m'a été précieuse tout au long de ce parcours.

Je remercie chaleureusement ma cousine **Lamis**, pour son affection, son soutien, et tous les moments partagés qui m'ont aidé à garder le moral.

Enfin, je tiens à remercier du fond du cœur **chaque personne qui a été à mes côtés** de près ou de loin, qui m'a soutenu, encouragé, écouté ou inspiré durant ce parcours. Votre présence, même discrète, a eu un impact.

Dédicace

Je dédie ce travail

À Allah, Le Tout-Puissant, Le Très Miséricordieux,
pour la force, la patience et la guidance qu'Il m'a accordées.

À moi-même,
pour avoir persévéré malgré les défis, cru en mes rêves et poursuivi ce chemin avec foi.

À ma tante Fatiha,
que je considère comme une mère avant d'être une tante,
pour son amour inconditionnel et sa présence rassurante.

À mes très chers parents,
pour leurs sacrifices immenses, leur soutien et leurs prières constantes.

À ma soeur Wala et à mon frère Moataz Bellah,
pour leur amour fraternel et leur lumière dans ma vie.
Je prie Dieu de leur accorder une guérison complète et une santé parfaite.

À mes petits Serine et Djaser,
que Dieu les bénisse, les protège et illumine leur chemin.

À mes proches :
Le professeur ***Mohamed Nemmamcha*** *et la professeure* ***Nouar Rachida****,*
modèles d'excellence qui ont marqué mon parcours,
et j'ai l'immense fierté qu'ils fassent partie de ma famille.
Je souhaite un jour suivre leurs traces et leur ressembler.

À mon frère Fakhr El Islam,
et ***à mon amie DJihane, soeur de coeur,***
pour leur fidélité et leur précieux soutien.

À mes chères amies : Hala, Yasmine, Rana, Zahra et Nada,
rencontrées à l'université, pour les beaux souvenirs partagés.

À ma cousine Basma, à ma cousine Lamis, et à ma cousine Darine, *pour leur présence affectueuse et leur soutien indéfectible.*

À la famille Nemmamcha, la famille de ma mère,
pour leur amour, leurs encouragements et leur foi en moi.

Résumé

Ce mémoire de master s'intéresse à la problématique de la reconnaissance automatique des insectes nuisibles, un enjeu crucial pour l'agriculture de précision et la sécurité alimentaire. Face à la complexité et la lenteur de l'identification manuelle, l'objectif principal de ce travail est de développer et de valider un système intelligent, modulaire et léger, capable de détecter et d'identifier les insectes en temps réel pour un déploiement sur des terminaux mobiles.

Notre approche repose sur un pipeline en deux étapes : la détection d'objets avec le modèle **YOLOv12n** (une version de YOLOv12), suivie de la classification des espèces avec **FastViT-SA12** (une variante de la famille FastViT). Le système a été entraîné et évalué sur le jeu de données complexe **IP102**, qui compte 102 classes et présente des défis majeurs comme le déséquilibre des données et la grande variabilité visuelle des insectes. Les expérimentations ont démontré l'efficacité de notre approche : le module de détection a atteint un excellent score **mAP@50 de 95,8%** sur l'ensemble de test. Le module de classification a, quant à lui, obtenu une précision **Top-1 de 73,77%** et **Top-5 de 90,82%**, des performances très compétitives qui rivalisent avec l'état de l'art.

En conclusion, ces résultats valident la pertinence de notre système modulaire comme une solution pragmatique et performante. L'équilibre atteint entre la précision et l'efficacité, notamment une faible latence sur mobile (130 ms), confirme son adéquation pour une application concrète en agriculture de précision.

Mots clés : *Deep Learning, Détection d'objets, Classification d'images, YOLOv12, FastViT, IP102 Dataset, insectes nuisibles, agriculture*

Abstract

This master's thesis addresses the problem of automatic recognition of harmful insects, a crucial challenge for precision agriculture and food security. Faced with the complexity and slowness of manual identification, the main objective of this work is to develop and validate an intelligent, modular, and lightweight system capable of detecting and identifying insects in real-time for deployment on mobile devices.

Our approach is based on a two-stage pipeline : object detection with the **YOLOv12n** model (a version of YOLOv12), followed by species classification with **FastViT-SA12** (a variant from the FastViT family). The system was trained and evaluated on the complex **IP102** dataset, which includes 102 classes and presents major challenges such as data imbalance and high visual variability among insects. The experiments have demonstrated the effectiveness of our approach : the detection module achieved an excellent **mAP@50 score of 95.8%** on the test set. The classification module, in turn, obtained a **Top-1 accuracy of 73.77%** and a **Top-5 accuracy of 90.82%**, results that are highly competitive and rival the state of the art.

In conclusion, these results validate the relevance of our modular system as a pragmatic and high-performing solution. The balance achieved between accuracy and efficiency, notably a low mobile latency (130 ms), confirms its suitability for practical application in precision agriculture.

Keywords : *Deep Learning, Object Detection, Image Classification, YOLOv12, FastViT, IP102 Dataset, Pest Insects, Agriculture.*

Table des matières

Table des figures

Liste des tableaux

Abréviations et Acronymes

<AFFM> Adaptive Feature Fusion Module

<ANN> Artificial Neural Network

<BIFPN-S> Bidirectional Feature Pyramid Network - Small

<CNN> Convolutional Neural Network

<COCO> Common Objects in Context

<CSA> Canal/Spatial Attention

<DeiT> Data-efficient Image Transformer

<DETR> DEtection TRansformer

<ECA> Efficient Channel Attention

<EFLM> Effective Feature Location Module

<FLOPs> Floating Point Operations per Second

<FSL> Few-Shot Learning

<GAM> Global Attention Mechanism

<GLCM> Gray-Level Co-occurrence Matrix

<GPU> Graphics Processing Unit

<HSV> Hue, Saturation, Value (Teinte, Saturation, Valeur)

<ILSVRC> ImageNet Large Scale Visual Recognition Challenge

<IoU> Intersection over Union

<IPM> Integrated Pest Management (Lutte Intégrée contre les Ravageurs)

<IR> Imbalance Ratio

<KNN> K-Nearest Neighbors

<M-CBAM> Modified Convolutional Block Attention Module

<MAE> Mean Absolute Error

<mAP> Mean Average Precision

<MLP> Multi-Layer Perceptron

<MPDIoU> Mask-based Parallel Distance Intersection over Union

<NMS> Non-Maximum Suppression

<PCSA> Parallel Channel and Spatial Attention

<PSSM> Pyramid Scene Scoring Module

<ReLU> Rectified Linear Unit

<RF> Random Forest

<RPN> Region Proposal Network

<SGD> Stochastic Gradient Descent

<SSD> Single Shot MultiBox Detector

<SVM> Support Vector Machine

<ViT> Vision Transformer

<YOLO> You Only Look Once

Introduction Générale

Contexte

Les insectes nuisibles représentent une menace majeure pour l'agriculture mondiale, responsable de pertes de rendement considérables qui compromettent la sécurité alimentaire. Traditionnellement, l'identification de ces ravageurs repose sur une expertise humaine, une tâche complexe, lente et difficile à déployer à grande échelle [1]. Dans un monde où l'agriculture de précision devient une nécessité, l'intelligence artificielle, et plus particulièrement la vision par ordinateur, offre une opportunité unique pour développer des outils d'aide à la décision rapides, fiables et accessibles.

Problématique et Motivation

La motivation de ce travail est de concevoir et de développer un système intelligent capable d'identifier automatiquement les insectes nuisibles, directement sur le terrain. L'objectif est de fournir une solution pratique et performante, embarquée dans une application mobile, pour aider les agriculteurs et les techniciens agricoles à protéger les cultures de manière plus efficace et réactive [2].

Le développement d'un tel système se heurte à plusieurs défis techniques majeurs, principalement liés à la nature des données. Le jeu de données de référence que nous utilisons, **IP102** [3], illustre parfaitement ces complexités :

- **La diversité biologique :** Les insectes apparaissent sous différentes formes au cours de leur cycle de vie (œuf, larve, adulte), ce qui rend leur identification difficile [4].
- **Le déséquilibre des données :** Certaines espèces sont très représentées tandis que d'autres sont rares, ce qui risque de biaiser l'apprentissage du modèle.
- **La variabilité visuelle :** Dans les images, les insectes peuvent être de très petite taille, partiellement cachés ou dans des conditions d'éclairage variées, compliquant leur détection [5, 6].

Face à cette problématique, nous proposons une approche modulaire en deux phases successives pour garantir à la fois robustesse et efficacité :

1. **Une phase de détection**, qui utilise le modèle **YOLOv12n** [7] pour localiser avec rapidité et précision les insectes dans l'image. Ce choix est motivé par sa légèreté et ses performances élevées, même pour les petits objets [8].

2. **Une phase de reconnaissance**, qui s'appuie sur le modèle **FastViT-SA12** [9] pour identifier l'espèce de chaque insecte détecté. Cette architecture a été choisie pour son excellent équilibre entre précision et vitesse d'exécution sur les appareils mobiles [10].

Cette architecture **Détection** $\rightarrow$ **Classification** permet d'optimiser chaque tâche indépendamment et de construire un système complet et performant.

Structure du mémoire

Ce mémoire est organisé en trois chapitres afin de présenter notre démarche de manière claire et structurée :

- Le **premier chapitre** est consacré à l'état de l'art. Il retrace l'évolution des méthodes de reconnaissance d'insectes, des approches classiques aux techniques modernes de Deep Learning comme les CNN et les Vision Transformers.
- Le **deuxième chapitre** décrit en détail la conception de notre système. Il présente les caractéristiques du dataset IP102, justifie les choix architecturaux de YOLOv12n et FastViT-SA12, et détaille les stratégies de préparation des données et d'entraînement.
- Le **troisième chapitre** présente la mise en œuvre pratique et les résultats de nos expérimentations. Il analyse les performances de chaque module à l'aide de différentes métriques et illustrations, et compare nos résultats à ceux de l'état de l'art.

Enfin, une **conclusion générale** synthétise les contributions de ce travail et propose des pistes d'amélioration pour des développements futurs.

Chapitre 1

Évolution des Méthodes de Reconnaissance d'Insectes Nuisibles

1 Introduction

Les insectes nuisibles représentent une menace majeure pour l'agriculture, entraînant des pertes de rendement considérables. La reconnaissance manuelle de ces ravageurs, tâche complexe et chronophage, repose sur une expertise difficile à généraliser. L'intelligence artificielle, et notamment le Deep Learning, offre aujourd'hui des solutions automatisées prometteuses pour relever ce défi. Ce chapitre retrace l'évolution des méthodes de reconnaissance, en partant des approches conventionnelles (analyse morphologique, Machine Learning classique) pour aboutir à la révolution du Deep Learning. Nous y détaillons les architectures fondamentales, des Réseaux de Neurones Convolutifs (CNN) aux Vision Transformers (ViT), ainsi que les modèles de référence pour la classification et la détection d'objets. Une analyse des bases de données spécifiques au domaine et de leurs défis intrinsèques précède une revue de l'état de l'art, qui met en lumière les compromis actuels entre précision et applicabilité sur le terrain.

2 Les Méthodes Conventionnelles : L'Ère pré-Deep Learning

Les approches conventionnelles de reconnaissance des insectes nuisibles, qui prédominent dans l'ère pré-Deep Learning, reposent sur l'expertise humaine, l'analyse d'images manuelle et l'apprentissage automatique classique. Bien qu'elles aient été fondamentales, ces méthodes ont montré des limites intrinsèques qui ont motivé la recherche de solutions plus robustes. Leurs principales faiblesses peuvent être résumées comme suit :

- **Précision Limitée :** La performance était plafonnée par la subjectivité de l'expert et par la difficulté de concevoir manuellement des caractéristiques (texture, forme...) assez robustes pour distinguer des espèces similaires [10].
- **Faible Automatisation :** La forte dépendance à l'intervention humaine, notamment pour l'extraction manuelle des caractéristiques, freinait le passage à l'échelle pour traiter

de grands volumes de données [11].

- **Manque d'Adaptabilité :** Basés sur des caractéristiques fixes, ces modèles ne pouvaient généraliser efficacement face aux conditions réelles et variables (luminosité, angles, occultations...), contrairement aux approches modernes qui s'adaptent aux données [12].

2.1 Approches Basées sur l'Expertise Humaine

Pour identifier les insectes, les spécialistes se sont longtemps reposés sur leur expertise, considérée comme la pierre angulaire de la taxonomie [1]. Cette expertise combine deux approches principales : une méthode d'analyse détaillée, la **morphologie comparative**, et une reconnaissance plus globale basée sur l'intuition, le **traitement holistique**. Bien que ces méthodes traditionnelles soient essentielles, elles ont montré leurs limites, notamment en termes de **subjectivité**, de **lenteur** et de **difficulté de standardisation** [1], ce qui a poussé la recherche vers des solutions automatisées.

2.1.1 Morphologie comparative

Il s'agit de la méthode de base en taxonomie, qui consiste à identifier un insecte en examinant attentivement ses caractéristiques physiques, comme la forme de ses ailes ou de ses pattes [1]. Cette technique est très informative mais présente des inconvénients majeurs : elle demande un temps considérable et une expertise très pointue. De plus, sa fiabilité peut parfois être compromise par des similitudes dues à l'évolution (homoplasies), ce qui complique la classification [1].

FIGURE 1.1 – Exemple de morphologie comparative : distinction entre le papillon Monarque et le Vice-roi. La ligne noire distincte sur l'aile postérieure du Vice-roi (flèche) est un critère de différenciation clé utilisé par les experts pour les identifier (source de figure : [13])

2.1.2 Traitement holistique

Cette approche correspond à l'intuition de l'expert. Plutôt que d'analyser chaque détail, le spécialiste reconnaît l'insecte dans son ensemble, un peu comme on reconnaît un visage. Des recherches ont montré que les humains, tout comme les abeilles ou les guêpes, utilisent ce mécanisme cognitif pour identifier des stimuli visuels complexes [14]. Cette capacité, basée sur l'expérience, permet à un entomologiste aguerri d'identifier une espèce quasi-instantanément [1]. Le problème principal est que cette compétence est personnelle et subjective. Elle est donc difficile à expliquer, à formaliser et à reproduire dans un système informatique, ce qui lui vaut d'être souvent critiquée par les défenseurs de méthodes plus "objectives" [1].

2.2 Techniques d'analyse d'images classiques

Avant le Deep Learning, l'identification des insectes reposait sur des méthodes d'analyse d'images en plusieurs étapes. Le processus consistait généralement d'abord à isoler l'insecte de son arrière-plan (segmentation), puis à extraire des caractéristiques visuelles précises (texture, forme, couleur) pour les classifier, le tout sans apprentissage automatique direct à partir des pixels bruts [15].

Les méthodes traditionnelles de segmentation d'images visent à isoler l'insecte du fond. Le seuillage (global ou adaptatif) est une approche courante : par exemple, Kumar et al. (2017) utilisent un seuillage adaptatif pour séparer et compter les insectes nuisibles dans des images de cultures [16]. Un exemple de cette méthode est illustré dans la figure 1.3. On emploie aussi la détection de contours (edges) pour repérer précisément les insectes : Kasinathan et al. (2021) décrivent ainsi une chaîne de traitement où le premier plan (foreground) est extrait puis les contours de l'insecte sont identifiés pour le segmenter [17]. Les opérations de morphologie mathématique complètent souvent la segmentation : dilatations/érosions, ouvertures/fermetures ou même l'algorithme des watersheds servent à éliminer les bruits et à consolider la forme de l'insecte avant classification [18]. Ces techniques de prétraitement (filtrage, seuillage, contours, morphologie) préparent ainsi l'image pour la détection ou le comptage des ravageurs.

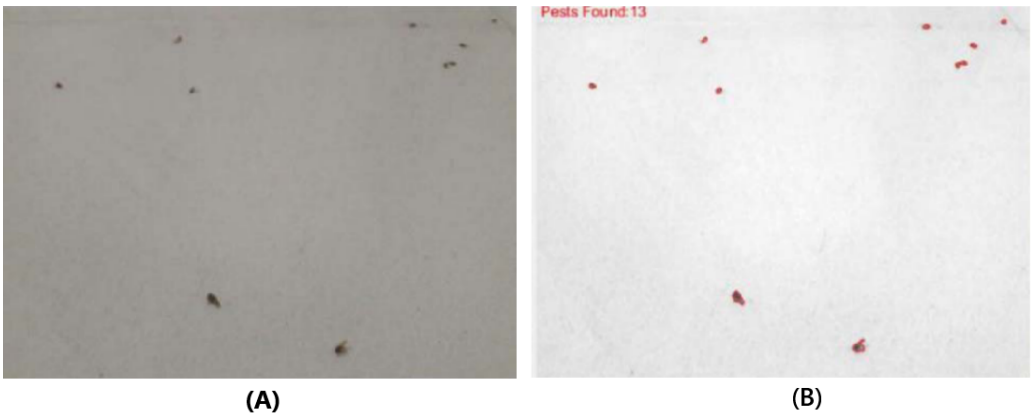


FIGURE 1.2 – Illustration de l'effet d'un seuillage adaptatif sur des images d'insectes nuisibles : image originale (A) et résultat du seuillage (B) selon la méthode proposée par Kumar et al. [16]. Ce prétraitement permet d'isoler efficacement les insectes du fond afin de faciliter leur comptage.

Extraction de caractéristiques visuelles

Une fois l'insecte segmenté, on extrait des descripteurs visuels pour le reconnaître. Les descripteurs de texture sont très utilisés : la matrice de cooccurrence (en anglais : *gray-level co-occurrence matrices* GLCM) [19] permet de quantifier des motifs de surface (entropie, contraste, corrélation, homogénéité...) capturant la texture du corps ou des ailes. Par exemple, Ebadi Manaa (2016) présente une méthode exploitant la GLCM dans quatre directions (0°, 90°, 180°, 270°) pour extraire des caractéristiques de texture et distinguer les insectes nuisibles [20]. D'autres travaux soulignent l'intérêt de ces statistiques de texture pour classer différentes espèces.

Outre la texture, la couleur peut différencier les insectes : on utilise alors des histogrammes de teinte et de saturation (HSV) ou des composantes R/G/B pour capturer les couleurs dominantes du corps ou des ailes. Enfin, la forme globale et les motifs veinés des ailes sont exploitables. Par exemple, Ling et al. (2023) binarisent les images d'ailes pour ne garder que le réseau de nervures, puis extraient des moments invariants de Krawtchouk [21] pour caractériser la forme complexe des ailes. Ils obtiennent ainsi une précision d'identification élevée (98,6 %) à l'échelle taxonomique (famille d'insectes) grâce à ces descripteurs de forme [22]. En résumé, la reconnaissance pré-DL combine classiquement des descripteurs de texture (GLCM), de couleur et de forme (contours, invariants géométriques) pour distinguer les espèces d'insectes nuisibles.

2.3 Techniques basé sur le Machine Learning classique

Les techniques basées sur l'apprentissage automatique classique reposent sur l'extraction manuelle de caractéristiques, telles que la texture, la forme et la couleur, pour classer les insectes nuisibles.

SVM (Support Vector Machine)

Plusieurs travaux ont utilisé des SVM pour classer des insectes nuisibles à partir d'images. Par exemple, Santhanambika et al. (2024) ont extrait des caractéristiques morphologiques d'insectes stockés (ravageurs de grains) et entraîné un SVM (ainsi que des KNN, CNN, Bayes naïf). Ils rapportent 81% d'exactitude pour le SVM (KNN 76%, CNN 98%) [23]. Doan (2023) a combiné des modèles EfficientNet pré-entraînés pour extraire des descripteurs d'images d'insectes, puis un SVM spécial (*Power Mean SVM*) pour la classification. Sur plusieurs *datasets* d'images de ravageurs (Xie24, D0, IP102), son approche atteint 99% de précision sur Xie24 et D0 (et 72,3% sur IP102) [24]. D'autres exemples illustrent la diversité des usages. Fuchida et al. (2017) ont conçu un système de vision pour distinguer les moustiques d'autres insectes (abeilles et mouches) à partir de mesures de proportions corporelles. Leur SVM obtient par exemple 85,2% d'exactitude pour identifier les moustiques (et 97,6% pour les autres insectes) [25]. La figure présente des exemples de classifications correctes et incorrectes réalisées par ce système.

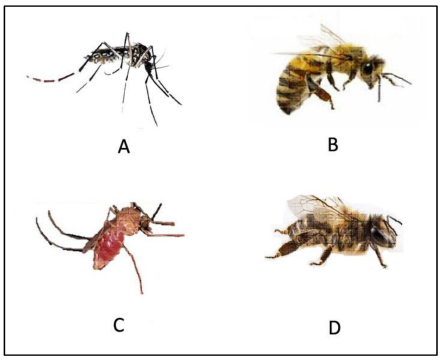


FIGURE 1.3 – Résultats d'exemples de classification. Un moustique correctement classé (A) et un *bug* correctement classé (B) ; (C) et (D) montrent respectivement un *bug* et un moustique incorrectement classés [25].

Tous ces travaux utilisent des images d'insectes (ou d'environnements de culture) comme données d'entrée, montrant la pertinence du SVM pour la classification d'insectes nuisibles dans le domaine agricole.

KNN (K-Nearest Neighbors)

Le KNN est également employé pour la reconnaissance d'insectes, souvent comme base de comparaison. Santhanambika et al. (2024) rapportent une exactitude de 76% pour un KNN sur leur *dataset* de ravageurs de céréales [23]. Dans un cas de classification multi-classes (huit ravageurs de la tomate), Praharsha et al. (2024) ont testé un KNN (k=5) sur 4263 images : ils n'obtiennent qu'environ 31% d'exactitude (précision 28%, rappel 27%, F1-score 27%) [26]. Dans l'ensemble, le KNN donne souvent des performances modestes sur des *datasets* complexes d'images d'insectes, mais il reste simple à mettre en œuvre pour des données où la distance (par exemple Euclidienne) entre caractéristiques de texture ou de forme est significative.

Autres techniques

Les arbres de décision seuls sont moins souvent mis en avant pour la reconnaissance d'images complexes. Dans l'étude de Praharsha et al. (2024) sur la tomate, un arbre de décision standard n'atteint qu'environ 31% de précision [26]. Cela s'explique par la difficulté d'un arbre simple à généraliser sur des données visuelles haute-dimensionnelles.

Le *Random Forest* est un classifieur ensembliste couramment utilisé. Par exemple, Praharsha et al. notent que leur forêt aléatoire atteint 46% d'exactitude sur le *dataset* multi-classes de ravageurs de tomate (précision 60%, rappel 38%, F1 42%), supérieure au SVM ou KNN dans ce cadre précis. Dans un autre domaine, Lin et al. (2022) ont construit des caractéristiques morphologiques (longueur et largeur du corps, etc.) d'instars larvaires de la légionnaire d'automne (Spodoptera frugiperda). En combinant ces mesures avec un Random Forest optimisé (sélection de caractéristique avec l'algorithme de Boruta [1] et utilisation de *Grid Search* [2]), ils obtiennent 92,22% d'exactitude pour classer les stades larvaires [29]. Ce résultat élevé illustre la puissance des RF sur des *datasets* riches en features physiques d'insectes.

3 La Révolution du Deep Learning en Vision par Ordinateur

Alors que les méthodes conventionnelles se heurtaient aux limites de leurs caractéristiques extraites manuellement, le Deep Learning a introduit un changement de paradigme. En permettant aux modèles d'apprendre automatiquement des représentations pertinentes directement à

1. Boruta est un algorithme de sélection de variables conçu pour identifier toutes les caractéristiques pertinentes d'un *dataset* pour une tâche de classification donnée, plutôt que de se limiter à l'ensemble optimal minimal [27].
2. **Grid Search :** est une technique d'optimisation des hyperparamètres utilisée pour trouver la meilleure combinaison d'hyperparamètres pour un modèle [28].

partir des données brutes, cette approche a déclenché une véritable révolution en vision par ordinateur, améliorant considérablement les performances pour des tâches comme la classification d'images, y compris la classification et la reconnaissance des insectes nuisibles [30].

Le *Deep Learning* est une branche du *Machine Learning* utilisant des réseaux de neurones profonds, c'est-à-dire avec de nombreuses couches cachées. Sa différence fondamentale avec les approches classiques réside dans sa capacité d'apprentissage de bout en bout (end-to-end). Au lieu de nécessiter une étape d'ingénierie manuelle pour extraire des caractéristiques (feature engineering), un modèle de *Deep Learning* apprend à la fois les caractéristiques pertinentes et la tâche de classification simultanément. Cette automatisation de l'extraction de caractéristiques est la clé de sa puissance et de sa supériorité en performance [31].

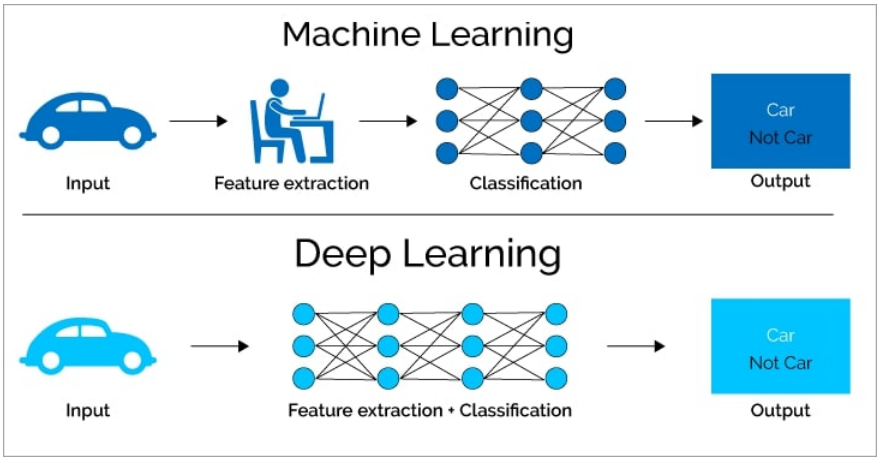


FIGURE 1.4 – Comparaison schématique entre l'apprentissage automatique classique, qui requiert une extraction de caractéristiques manuelle, et l'apprentissage profond (Deep Learning), qui intègre cette étape dans le modèle [32].

3.1 Convolutional Neural Network (CNN) : La Pierre Angulaire

Pour la vision par ordinateur, les Réseaux de Neurones Convolutifs (en anglais, *Convolutional Neural Network, CNN*) sont l'architecture de *Deep Learning* la plus fondamentale et la plus utilisée. Ils sont spécialement conçus pour traiter des données sous forme de grille, comme les pixels d'une image, en s'inspirant du cortex visuel humain [31].

Leur architecture type suit un flux logique pour transformer une image en une prédiction. D'abord, une série de **couches de convolution (Convolution Layers)** agissent comme des détecteurs de motifs. Les premières couches apprennent à reconnaître des traits simples (bords, couleurs, textures), tandis que les couches plus profondes combinent ces informations pour identifier des motifs plus complexes (des parties d'insectes, par exemple). Entre ces couches, des opérations de **pooling** réduisent la dimensionnalité des données pour ne conserver que les informations les plus essentielles, rendant le modèle plus efficace. Enfin, après cette phase d'extraction de caractéristiques, des **couches entièrement connectées (*Fully Connected Layers*)** analysent le résumé visuel de haut niveau et effectuent la classification finale, souvent via une couche **Softmax** qui convertit les scores en probabilités pour chaque classe.

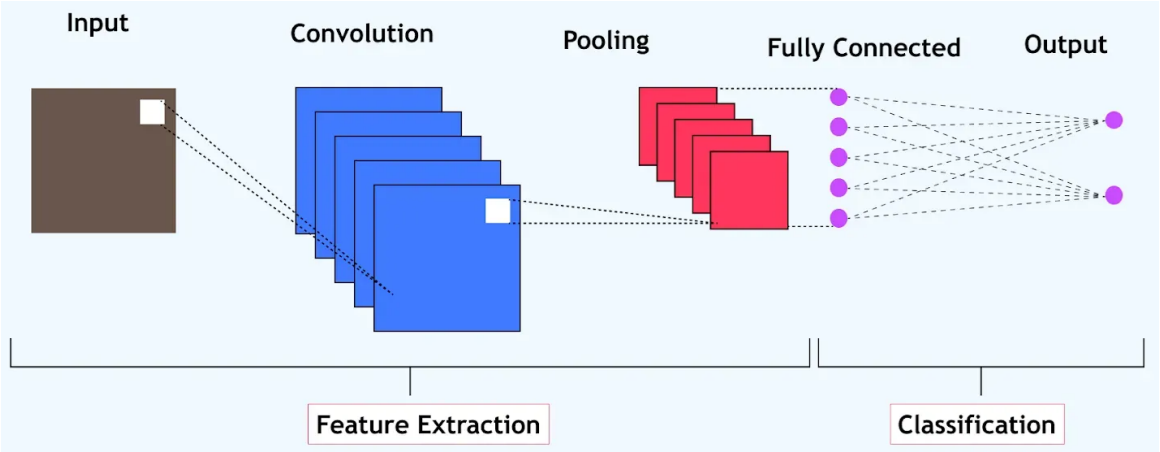


FIGURE 1.5 – Flux d'information dans une architecture CNN typique, de l'extraction de caractéristiques à la classification finale [33].

Pour que cet apprentissage ait lieu, le réseau s'appuie sur une boucle d'optimisation guidée par deux éléments clés :

- **Fonction de coût (Loss Function)** : La *loss function* évalue l'écart entre la sortie du modèle et la vérité terrain. Pour les tâches de classification multi-classes, la *cross-entropy loss* est la plus couramment utilisée [34]. Elle est définie comme suit :

$$\mathcal{L}_{\text{cross-entropy}} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

  où C est le nombre de classes, y_i la probabilité vraie (souvent `1` ou `0`), et $\hat{y}_i$ la probabilité prédite pour la classe i. Cette fonction pénalise fortement les mauvaises prédictions et pousse le modèle à affecter une probabilité élevée à la classe correcte [34].
- **L'algorithme d'optimisation (Optimizer) :** C'est le moteur qui ajuste les poids du réseau pour minimiser cette erreur. Des algorithmes modernes comme **Adam** [35] ou **SGD** (Stochastic Gradient Descent) [36] sont la norme. Ils calculent le gradient de l'erreur et mettent à jour les paramètres du modèle de manière itérative pour converger vers une solution optimale.

3.2 Vision Transformer (ViT) : vers une nouvelle ère du *Computer Vison*

L'architecture *Transformer* a été introduite en 2017 par Vaswani et al. pour le traitement du langage naturel [37]. Elle repose entièrement sur un mécanisme d'*attention* qui permet de pondérer dynamiquement les différentes parties de la séquence d'entrée en fonction de leur pertinence. Concrètement, chaque vecteur de la séquence calcule une somme pondérée des autres vecteurs, les poids dépendant d'une mesure de similarité entre éléments. Des travaux ultérieurs ont adapté cette idée à la vision par ordinateur. En particulier, Dosovitskiy et al. ont proposé en 2020 un modèle *Vision Transformer* (ViT) capable de traiter une image sous forme de séquence de patchs [38]. Ils démontrent qu'un *Transformer* purement appliqué aux patchs d'une image peut atteindre des performances de pointe sur la classification d'images, surtout

après un pré-entraînement sur de très larges *dataset*, tout en nécessitant moins de ressources de calcul qu'un réseau convolutif équivalent.

3.2.1 Fonctionnement du *Vision Transformer* (ViT)

La figure 1.6 illustre le processus général d'un ViT : l'image d'entrée est découpée en patchs, chaque patch est linéairement projeté en vecteur (*patch embedding*), auquel on ajoute un vecteur de position appris. Un *token* spécial «[class]» est concaténé en tête de séquence pour agréger l'information globale. Toute cette séquence est ensuite traitée par plusieurs blocs *Transformer*, et la sortie du *token* [class] sert à la classification finale [38].

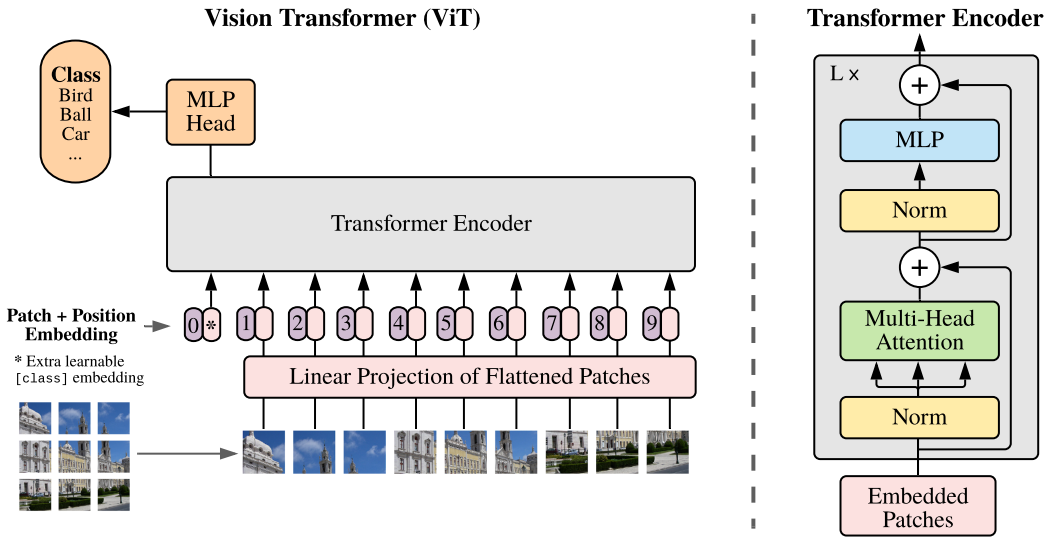


FIGURE 1.6 – Schéma simplifié de l'architecture d'un Vision Transformer (d'après Dosovitskiy et al. 2020) [38].

1. **Découpage en *patchs* et *patch embedding*** : l'image de taille $H \times W$ avec C canaux est découpée en $m = \left(\frac{H}{p}\right)\left(\frac{W}{p}\right)$ patchs carrés de côté p. Chaque patch est aplati en un vecteur de dimension $C \times p^2$ puis projeté par une couche linéaire dans un espace de dimension fixe D (embedding). On traite ainsi chaque patch comme un "mot" dans une séquence [39].
2. **Encodage positionnel et jeton `[CLS]`** : pour conserver l'information spatiale, on ajoute à chaque vecteur de patch un vecteur d'encodage de position (learnable). De même, on insère un vecteur spécial `[class]` (token *[CLS]*) en début de séquence, également projeté et enrichi par l'encodage positionnel. Ce jeton servira à collecter l'information globale de l'image.
3. **Blocs *Transformer* (multi-head *attention* + *MLP*)** : la séquence $\{\texttt{[CLS]}, patch_1, \ldots, patch_m\}$ passe ensuite par n blocs *Transformer*. Chaque bloc comporte une couche de *multi-head self-attention* qui calcule, pour chaque tête, des pondérations entre tous les patchs et le token `[CLS]` (captant ainsi les dépendances globales entre patchs) [40]. Chaque tête produit une représentation, et les sorties sont concaténées puis projetées. Un *perceptron multi-couches (MLP)* à deux couches (avec activation GELU) suit chaque attention, appliqué individuellement à chaque vecteur [40].

On utilise par ailleurs des normalisations de couche (Layer Norm) et des connexions résiduelles pour stabiliser l'apprentissage.

4. **Tête de classification (*MLP Head*)** : après les blocs d'attention, on extrait la représentation finale du jeton `[CLS]`. Celle-ci est passée à travers une petite couche *MLP* (souvent une couche cachée + une couche linéaire) qui prédit la distribution de probabilité sur les classes [40]. C'est cette sortie qui donne la prédiction finale du modèle.

La figure 1.7 présente des exemples *d'attention maps* apprises par un Vision Transformer (modèle ViT-S/16). Pour chaque exemple, on visualise à gauche l'image d'entrée et à droite en couleurs chaudes une des *attentions maps* montrant comment un patch cible (cerclé) «attire» son attention sur d'autres régions de l'image. La capacité d'attention globale permet au modèle de relier facilement des objets distants dans l'image, illustrant l'un des avantages clés du Transformer en vision [40].

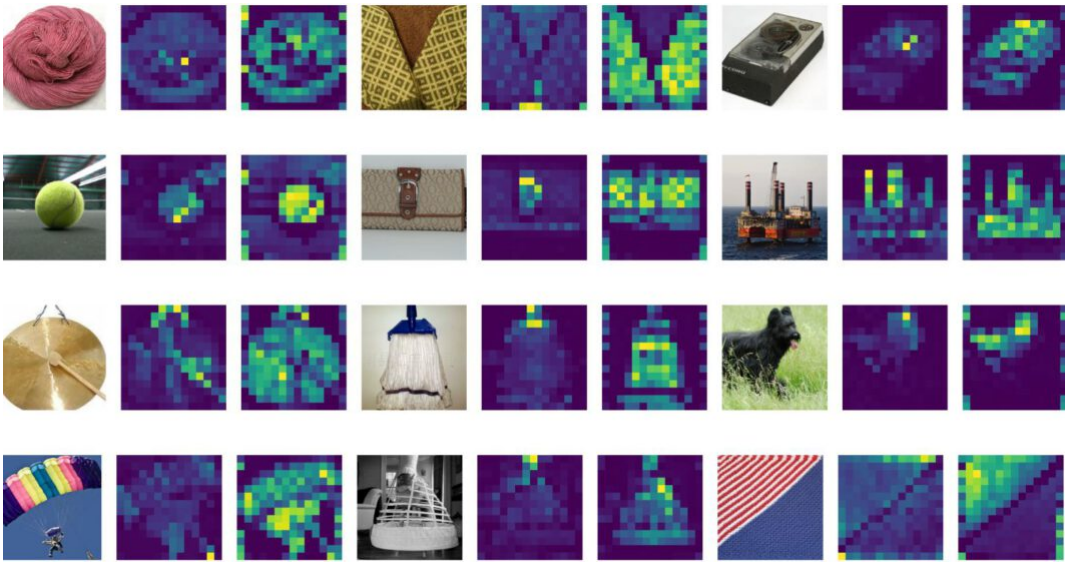


FIGURE 1.7 – Exemples d'*attention maps* apprises par un Vision Transformer (modèle ViT-S/16) [41].

3.2.2 CNN vs ViT : forces, faiblesses et cas d'usage

CNN (Convolutional Neural Network) : ce type de réseau utilise des filtres convolutifs pour extraire des caractéristiques locales (bords, textures, etc.), avec des opérations de pooling pour réduire la dimension spatiale. Les CNN sont hautement efficaces et robustes, notamment sur des *datasets* de taille modérée [42]. Leur inductive bias local leur permet d'apprendre rapidement des hiérarchies spatiales, ce qui est particulièrement utile quand les données annotées sont peu abondantes [42]. Toutefois, cette focalisation sur les caractéristiques locales peut limiter leur capacité à capter des relations globales complexes dans l'image. De plus, les CNN sont moins flexibles pour modéliser directement des dépendances longues (loin dans l'image), et peuvent être sensibles aux translations ou rotations non vues lors de l'entraînement [42].

ViT (Vision Transformer) : le ViT répartit l'image en patchs fixes (tokens) qu'il traite comme une séquence, exactement comme un modèle Transformer traite une phrase en langage. L'attention globale permet de modéliser directement toutes les interactions entre patchs, offrant

une compréhension du contexte global de l'image [42]. Quand le volume de données d'entraînement est très grand, les ViT surpassent les CNN en précision et en capacité de généralisation [42]. . En revanche, leur inductive bias faible fait que les ViT ont besoin de beaucoup de données ou d'astuces (data augmentation, régularisation) pour converger correctement sur de petits *datasets* [40, 42]. Les ViT sont aussi plus coûteux en calcul (auto-attention quadratique) et en paramètres pour des résolutions élevées [42].

Le tableau suivant illustre la comparaison entre les *CNN* et les *ViT* selon plusieurs critères essentiels tels que la structure, les performances, la robustesse et les exigences en données.

Critère	**CNN**	**ViT**
Structure	Basé sur des convolutions locales et du *pooling*	Basé sur le mécanisme d'*attention* globale (auto-attention)
Inductive Bias	Fort (hiérarchies locales, invariance translationnelle)	Faible, requiert plus de données pour généraliser
Capacité de généralisation	Bonne sur petits jeux de données	Excellente sur grands jeux de données, limitée sur petits jeux sans pré-entraînement
Modélisation du contexte global	Limitée (nécessite des couches profondes pour l'agrégation spatiale)	Naturellement présente via l'attention globale
Efficacité en calcul	Moins coûteux, efficace sur GPU/CPU standards	Plus gourmand en mémoire et en calcul (attention quadratique)
Robustesse au bruit et flou	Bonne tolérance locale (filtres fixes)	Sensible sans régularisation, mais flexible avec *fine-tuning*

TABLE 1.1 – Comparaison entre CNN et Vision Transformer (ViT)

Face aux compromis de chaque approche, une solution puissante a émergé : les **architectures hybrides** qui combinent un CNN et un Transformer. L'idée est de tirer le meilleur des deux mondes. Dans ces modèles, un CNN est souvent utilisé comme un "***backbone***" (une ossature) pour extraire efficacement les caractéristiques locales et spatiales de l'image. Les *feature maps* de haut niveau produites par le CNN, qui sont déjà riches en informations, sont ensuite découpées en une séquence de "*tokens*" et passées à un encodeur *Transformer*. Ce dernier se charge alors de modéliser les relations et le contexte global entre ces caractéristiques. Cette synergie permet de combiner l'efficacité et le biais inductif robuste des CNN avec la capacité des Transformers à comprendre le contexte global, menant souvent à des modèles très performants,

même sans nécessiter des *datasets* aussi massifs que les ViT purs [43].

3.3 Architectures de Référence pour la Reconnaissance d'Images

3.3.1 Les Pionniers : AlexNet et VGG

La révolution CNN débute avec **AlexNet** (Krizhevsky et al., 2012) [44] : un réseau à 8 couches (5 convolutives suivies de 3 couches entièrement connectées) utilisant des ReLU et du Dropout, entraîné sur 2 GPU. Il contient environ 60 millions de paramètres et a obtenu 84.7% de précision Top-5 sur ImageNet (15.3% d'erreur) surpassant largement ses prédécesseurs [44]. Cette architecture a introduit l'usage massif du GPU, du *data augmentation*, des fonctions d'activation non-saturantes et du *Dropout* pour combattre l'*overfitting*. Elle a lancé l'ère des réseaux profonds, mais reste très volumineuse et peu robuste (sensibilité au bruit) en raison de son nombre de paramètres et de sa profondeur.

Quelques années plus tard, **VGG** (Simonyan & Zisserman, 2014) [45] proposa de multiplier la profondeur du réseau en empilant de petits filtres 3×3. Les modèles VGG-16/19 (16–19 couches) utilisent exclusivement des convolutions 3×3 et du max-pooling [45]. Ils ont atteint de très bonnes performances sur ImageNet (victoire en classification et localisation ILSVRC2014 [3]) [45]. Cette approche a confirmé que la profondeur améliore la précision (top-1 71–72%), mais au prix d'un nombre de paramètres énorme (138 millions pour VGG-16) et d'une complexité de calcul très élevée. VGG reste donc un réseau plus performant qu'AlexNet, mais plus lourd et gourmand en ressources [46].

3.3.2 Optimisation de la profondeur : ResNet et DenseNet

ResNet (He et al., 2016) introduit la connexion résiduelle (« skip-connections ») pour faciliter l'apprentissage de très grands réseaux [47]. Un bloc résiduel apprend la différence entre l'entrée et la sortie de la couche au lieu de la transformer directement. Grâce à ce principe, ResNet permet de construire des réseaux profonds (50, 101 ou 152 couches) sans dégradation de la précision. Par exemple, ResNet-50 avec ses 25M paramètres atteint 76.0% de précision Top-1 sur ImageNet [48], et ResNet-152 (60M paramètres) 77.8% [48], tout en obtenant 3.57% d'erreur Top-5 dans une soumission ILSVRC2015 [47]. Les résidus améliorent le flux de gradient et rendent l'entraînement plus stable. L'inconvénient est que ResNet reste relativement coûteux en calcul (par ex. 4.1 GFLOPs [4] par image pour ResNet-50 [48]) et ne corrige pas entièrement la lourdeur des réseaux profonds.

DenseNet (Huang et al., 2017) va plus loin en connectant toutes les couches entre elles [50]. Chaque couche reçoit en entrée les cartes de caractéristiques de toutes les couches précédentes (connexion en « entonnoir »). Cette stratégie réduit encore le phénomène de gradient évanescent, encourage la réutilisation des caractéristiques et diminue le nombre total de paramètres [50].

3. ILSVRC : ImageNet Large Scale Visual Recognition Challenge, édition 2014. Il s'agit d'une compétition annuelle d'évaluation des algorithmes de vision par ordinateur sur des tâches de classification, détection et localisation d'objets à grande échelle à partir du *dataset* ImageNet.

4. FLOPs : Floating Point Operations per Second. Il s'agit d'une unité mesurant le nombre de milliards d'opérations en virgule flottante qu'un modèle peut effectuer par seconde, utilisée pour estimer la complexité computationnelle.

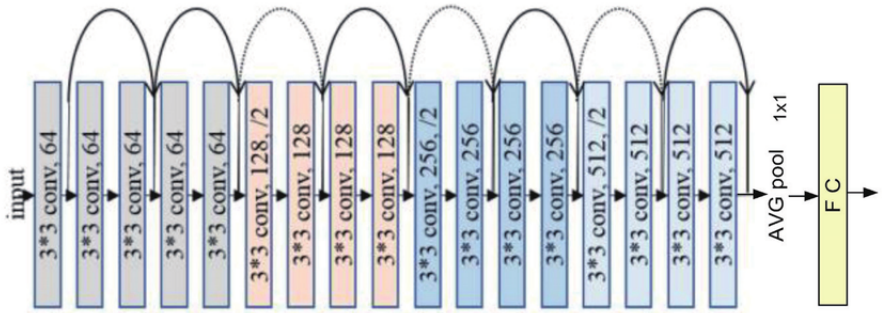


FIGURE 1.8 – Architecture simplifiée de ResNet18 : succession de blocs résiduels avec convolutions 3×3, suivie d'un pooling global et d'une couche entièrement connectée pour la classification [49].

DenseNet-169 (14M paramètres) atteint 76.2% de précision Top-1, et DenseNet-264 (34M) 77.9% [48]. Grâce à leurs connexions denses, ces réseaux sont souvent un peu plus compacts que ResNet pour une précision équivalente, mais restent complexes à déployer sur des appareils contraints. En résumé, ResNet et DenseNet ont révolutionné la profondeur des CNN : ResNet grâce aux blocs résiduels et DenseNet grâce aux connexions denses, permettant de repousser les performances sur ImageNet [50].

3.3.3 L'ère de l'efficacité : InceptionV3, RegNetY

Inception V3 (Szegedy et al., 2016) est issu de la famille Inception (GoogleNet). Il combine des modules « inception » contenant des convolutions parallèles de différentes tailles et des facteurs de réduction de dimension [51]. Inception V3 compte environ 24M paramètres et atteint 78.8% de précision Top-1 sur ImageNet [52]. Il a l'avantage d'intégrer de nombreuses optimisations (facteurs $1 \times n$ puis $n \times 1$, global pooling, normalisations), ce qui en fait un bon compromis efficacité/précision sur serveurs, mais son déploiement sur mobile reste plus coûteux que MobileNet ou EfficientNet.

Enfin, **RegNetY** (Radosavovic et al., 2020) explore un espace de design simple et régulier [53]. Les auteurs définissent des familles de réseaux « quantifiées » où la profondeur et la largeur croissent linéairement. Les modèles RegNet sont ainsi très scalables et optimisés. Sous des settings équivalents, RegNet surpasse souvent EfficientNet en performances tout en étant jusqu'à 5× plus rapide sur GPU. Par exemple, une version RegNetY-3.2GFLOPs (34M paramètres) atteint environ 82.0% Top-1 sur ImageNet [53]. RegNet offre un bon compromis rapidité/précision pour des architectures CNN modernes grâce à sa structure régulière.

3.3.4 Le changement de paradigme : Vision Transformers (ViT, DeiT)

Les Vision Transformers (ViT) [54], déjà présentés dans la section 3.2, représentent un tournant majeur en abandonnant les convolutions au profit d'une attention globale appliquée à des patchs d'image. Ils occupent désormais une place centrale dans les architectures modernes.

DeiT (Data-efficient Image Transformer) [55] est une adaptation du ViT visant à permettre un entraînement efficace sur des *datasets* de taille modérée, sans pré-entraînement sur des corpus

massifs. Pour ce faire, DeiT introduit une technique de distillation par token (distillation token) qui permet à un modèle étudiant Transformer d'apprendre d'un modèle enseignant CNN. Avec environ 22M de paramètres, DeiT-Tiny peut atteindre 72.2% de précision Top-1 sur ImageNet, tandis que DeiT-Base (86M) atteint environ 81.8% [55]. Cette famille de modèles est compétitive avec ResNet tout en conservant les avantages des Transformers, notamment leur capacité à capturer des dépendances globales.

Enfin, **Swin Transformer** (Shifted Window Transformer) [56] est une architecture hiérarchique qui applique l'attention sur des fenêtres locales avec un mécanisme de décalage entre couches. Cette approche permet une complexité réduite (linéaire par rapport à la taille de l'image) tout en préservant les capacités de modélisation globale des Transformers. Le Swin-B atteint 84.5% de Top-1 sur ImageNet avec 88M de paramètres [56] et surpasse les CNN dans plusieurs benchmarks de vision (segmentation, détection, etc.).

3.3.5 Architectures optimisées pour les dispositifs mobiles (MobileNet, EfficientNet, FastViT)

L'essor des applications embarquées et mobiles a favorisé le développement de modèles légers, capables d'offrir un bon compromis entre performance et efficacité. Ces architectures visent à réduire la latence, la complexité computationnelle (FLOPs) et le nombre de paramètres, tout en maintenant une précision compétitive sur des *datasets* standards comme ImageNet.

MobileNet (Howard et al., 2017)[57] repose sur des convolutions séparables en profondeur (*depthwise separable convolutions*) pour alléger le réseau. Chaque couche est factorisée en une convolution depthwise (indépendante pour chaque canal) suivie d'une convolution 1×1. MobileNetV1 (4M paramètres) atteint environ 70% de précision Top-1, et sa version améliorée MobileNetV2 (Sandler et al., 2018)[58] approche 72%. Ces réseaux proposent un excellent compromis entre taille et précision.

EfficientNet (Tan et Le, 2019)[59] introduit une méthode de *compound scaling* qui augmente simultanément la profondeur, la largeur et la résolution des images. Par exemple, EfficientNet-B0 (5.3M paramètres) atteint 77.1% Top-1, surpassant ResNet-50 (26M), tandis que EfficientNet-B7 (66M) atteint 84.3%. Cette gamme d'architectures est hautement optimisée pour l'efficacité calculatoire.

MobileViT (Mehta et Rastegari, 2022)[60] combine les avantages des CNN et des Transformers pour les appareils mobiles. Il insère des blocs d'attention dans un réseau léger tout en conservant les inductive biases convolutifs. MobileViT-XXX (6M paramètres) atteint 78.4% Top-1, surpassant MobileNetV3 et DeiT de taille équivalente.

FastViT (Vasu et al., 2023)[9] est un Transformer hybride conçu pour la latence réduite. Il intègre un opérateur de mélange de tokens (RepMixer) et une sur-paramétrisation structurelle, permettant une accélération significative de l'inférence. Par exemple, FastViT est $4.9\times$ plus rapide qu'EfficientNet et $1.9\times$ plus rapide que ConvNeXt pour une précision équivalente, tout en étant plus robuste aux corruptions (bruit, flou).

EdgeNeXt (2023) [61] est une architecture hybride CNN/Transformer conçue pour les environnements edge. Elle introduit les modules *split transpose attention* pour étendre le champ

réceptif sans surcharge computationnelle. La version *EdgeNeXt-small* (5.6M paramètres) atteint 79.4% de précision Top-1 sur ImageNet, avec des FLOPs inférieurs à ceux de MobileViT [61].

EfficientFormer (2022) [62] est un Transformer optimisé pour la latence mobile. Grâce à des blocs dimensionnellement compatibles et à une compression (*slimming*) centrée sur la latence, la version L1 (12.3M paramètres) atteint 80.5% Top-1 avec 1.3G FLOPs, et une inférence de 1.6ms sur iPhone 12 [62].

MobileOne (2023) [63] est une CNN re-paramétrable développée par Apple pour des inférences ultra-rapides. La version S4 (14.8M paramètres, 2.98G FLOPs) atteint 79.4% Top-1 avec seulement 1.86ms sur iPhone 12, soit 38× plus rapide que MobileFormer pour une précision équivalente [63].

GhostNet (Han et al., 2020)[64] propose des *Ghost modules* pour produire davantage de cartes de caractéristiques à moindre coût. GhostNet-1.0× (5.2M paramètres) atteint 73.9% Top-1 avec seulement 0.14G FLOPs, et surpasse MobileNetV3 en termes de rapidité et de précision [64].

ShuffleNet V2 (Ma et al., 2018)[65] est un CNN compact qui utilise le mélange de canaux (*channel shuffle*) et des groupements convolutifs pour réduire les coûts. Sa version 1.0× (2.3M paramètres) atteint 69.4% Top-1 avec seulement 0.15G FLOPs, offrant une bonne efficacité sur CPU et GPU malgré une précision modeste [65].

En résumé, ces modèles « mobiles » jouent un rôle fondamental dans le déploiement de solutions de vision par ordinateur en milieux réels contraints, notamment pour la reconnaissance des insectes nuisibles [66, 67]. Leur faible encombrement, leur efficacité énergétique et leur rapidité d'inférence les rendent particulièrement adaptés à une utilisation sur des dispositifs embarqués tels que les smartphones agricoles, les drones de surveillance des cultures ou les capteurs autonomes placés dans les champs. Grâce à ces architectures optimisées, il devient possible de détecter et identifier les insectes en temps réel directement sur le terrain, sans dépendre d'une infrastructure cloud coûteuse ou d'une connexion réseau stable [67].

3.3.6 Tableau de synthèse comparatif

Le tableau 1.2 présente une comparaison synthétique des principales architectures utilisées en reconnaissance d'images, en mettant l'accent sur trois critères clés : la précision (**Acc.**), la robustesse (**Rob.**) et le nombre de paramètres (**Par. (M)**). D'autres aspects comme la complexité computationnelle (FLOPs), l'usage de l'attention, et la compatibilité mobile sont également indiqués pour mieux évaluer leur applicabilité sur le terrain.

Modèle (Année)	**Par. (M)**	**Acc.**	**Rob.**	**FLOPs**	**Attention**	**Mobile**	**Avantages / Inconvénients**
AlexNet (2012) [44]	60	57%	Faible	1.4G	Non	Non	Pionnier CNN ; peu robuste aux corruptions.
VGG-16 (2014) [45]	138	71.5%	Faible	15G	Non	Non	Profond mais très lourd et coûteux.

Suite à la page suivante

Table 1.2 – Suite de la page précédente

Modèle (Année)	**Par. (M)**	**Acc.**	**Rob.**	**FLOPs**	**Attention**	**Mobile**	**Avantages / Inconvénients**
ResNet-50 (2016) [47]	26	76.0%	Modérée	4.1G	Non	Non	Connexions résiduelles efficaces mais réseau encore lourd.
DenseNet-169 (2017) [50]	14	76.2%	Modérée	3.5G	Non	Non	Connexions denses compactes mais complexes à déployer.
MobileNetV2 (2018) [58]	3.4	71.8%	Faible / Modérée	0.3G	Non	Oui	Ultra léger ; précision limitée.
EfficientNet-B0 (2019) [59]	5.3	77.1%	Modérée	0.39G	Non	Oui	Très bon compromis précision / complexité.
EfficientNet-B7 (2019) [59]	66	84.3%	Bonne	37G	Non	Partiel	Très précis mais lourd pour mobile.
Inception-V3 (2016) [51]	24	78.8%	Modérée	5.7G	Non	Non	Optimisé pour serveurs, pas pour mobile.
RegNetY-3.2G (2020) [53]	34	82.0%	Bonne	–	Non	Non	Robuste et rapide, structure régulière.
ViT-B/16 (2021) [54]	86	77–79%	Faible	55G	Oui	Non	Performant après pré-entraînement massif ; sensible au bruit.
DeiT-Base (2021) [55]	86	81.8%	Modérée	17.6G	Oui	Non	Transformer entraîné efficacement sans grands *datasets*.
Swin-B (2021) [56]	88	84.5%	Bonne	15.4G	Oui	Non	Hiérarchique ; très performant pour tâches de vision générale.
MobileViT (2022) [60]	6	78.4%	Bonne	0.8G	Oui	Oui	Hybride efficace pour mobile.
FastViT (2023) [9]	7–11	80–82%	Très bonne	1–2G	Oui	Oui	Ultra rapide et robuste ; faible latence.
EdgeNeXt (2023) [61]	5.6	79.4%	Bonne	–	Oui	Oui	Split transpose attention ; FLOPs réduits vs MobileViT.

Suite à la page suivante

Table 1.2 – Suite de la page précédente

Modèle (Année)	Par. (M)	Acc.	Rob.	FLOPs	Attention	Mobile	Avantages / Inconvénients
EfficientFormer-L1 (2022) [62]	12.3	80.5%	Bonne	1.3G	Oui	Oui	Slim attention ; inférence rapide (1.6ms sur iPhone).
MobileOne-S4 (2023) [63]	14.8	79.4%	Bonne	2.98G	Non	Oui	CNN reparamétrable ; <2ms sur iPhone ; très rapide.
GhostNet-1.0 (2020) [64]	5.2	73.9%	Bonne	0.14G	Non	Oui	Ghost modules très efficaces ; rapide et compact.
ShuffleNetV2 (2018) [65]	2.3	69.4%	Modérée	0.15G	Non	Oui	Très compact ; rapide sur CPU/GPU ; précision modeste.

TABLE 1.2 – Architectures de référence pour la reconnaissance d'images : comparaison des performances (**Acc.**), de la robustesse (**Rob.**) et des paramètres (**Par. (M)**).

On constate que les modèles historiques comme AlexNet ou VGG, bien qu'innovants à leur époque, souffrent de lourdeur et d'une faible robustesse. Les architectures plus récentes, telles que EfficientNet, Swin-B ou FastViT, offrent un meilleur compromis entre précision, efficacité computationnelle et robustesse. L'introduction des mécanismes d'attention, notamment avec les Transformers (DeiT, Swin-B) ou les modèles hybrides (MobileViT, FastViT), a permis d'améliorer significativement la modélisation des dépendances globales dans l'image, renforçant ainsi la robustesse aux variations et au bruit. Enfin, les modèles légers comme MobileNetV2, GhostNet ou ShuffleNet V2 restent des candidats de choix pour les systèmes embarqués, avec un bon rapport taille/performance.

3.4 Architectures de Référence pour la Détection d'Objets

Le Deep Learning a révolutionné la détection d'objets en remplaçant les méthodes traditionnelles par des modèles avancés capables d'extraire des caractéristiques complexes. Grâce aux architectures CNN et aux modèles basés sur l'attention, la détection est devenue plus rapide, plus précise et efficace, même en temps réel et dans des environnements complexes. Afin de mieux structurer l'analyse, les modèles de détection d'objets ont été classés en deux grandes catégories : les approches en deux étapes et celles en une seule étape.

3.4.1 Approches en deux étapes

Les méthodes en deux étapes séparent la détection d'objets en deux phases distinctes. Dans un premier temps, un *Region Proposal Network*[5](RPN) génère des régions candidates sus-

5. Region Proposal Network (RPN) est un réseau entièrement convolutionnel qui prédit simultanément les délimitations des objets et leur score de présence à chaque position. Le RPN est entraîné de bout en bout afin de générer des propositions de régions de haute qualité [68].

ceptibles de contenir des objets. Dans un second temps, un réseau de classification affine ces propositions (*fast R-CNN*). Par exemple, le détecteur Faster R-CNN introduit un RPN qui partage les caractéristiques convolutionnelles avec le réseau de détection, rendant la génération de propositions quasi gratuite en calcul [68]. Cette approche permet d'atteindre une très haute précision, car chaque proposition est traitée en détail, mais elle souffre d'une latence élevée (typiquement quelques images par seconde) et d'une complexité importante (dizaines de millions de paramètres). Faster R-CNN (avec VGG-16) atteint par exemple 5 FPS sur GPU tout en maintenant un état de l'art en précision [68].

3.4.2 Approches en une étape

Les méthodes en une seule passe prédisent directement les boîtes englobantes (*bounding box*) et leurs classes avec un seul réseau, sans étape de proposition explicite. Par exemple, **SSD** (Single Shot MultiBox Detector) associe à chaque position de la carte de caractéristiques un ensemble de boîtes par défaut de différentes tailles et proportions. Le réseau prédit en une seule évaluation la présence de chaque classe dans chaque boîte par défaut ainsi que les ajustements de boîte correspondants [69]. SSD élimine l'étape de proposition d'objets, simplifiant l'architecture. Il offre de bonnes performances tout en étant très rapide : SSD300 (300×300) atteint 72.1% de mAP sur VOC 2007 à 58 FPS (Titan X) [69].

De même, **RetinaNet** est un détecteur dense utilisant une *focal loss*[6] pour corriger le déséquilibre extrême entre background et objets lors de l'entraînement [70]. RetinaNet obtient une précision rivalisant avec les détecteurs deux-étapes tout en maintenant une vitesse comparable aux autres modèles monolithiques. En particulier, RetinaNet avec focal loss égalise la vitesse des meilleurs one-stage tout en surpassant la précision des meilleurs two-stage [70].

La série des détecteurs **YOLO** (You Only Look Once) conçoit la détection comme un problème de régression direct sur l'image entière. Un unique réseau convolutif prédit en une passe les boîtes et classes pour l'image complète [71]. Ce design unifié permet une optimisation de bout en bout et atteint des débits très élevés (par ex. 45 FPS pour le YOLO original [71]) au prix de quelques imprécisions sur de petits objets. Les évolutions successives de YOLO (v2, v3, v4, etc.) ont introduit des ancres, des backbones améliorés et des techniques d'entraînement avancées pour augmenter la précision tout en conservant la rapidité. Les versions récentes comme **YOLOv12** [7] intègrent des mécanismes attentionnels modernes (Area Attention, FlashAttention, R-ELAN) afin d'augmenter à la fois la précision et la rapidité. Par exemple, YOLOv12-N atteint 40.6 % de mAP[7] sur COCO *dataset*[8] avec une latence de seulement 1.64 ms sur GPU T4, surpassant YOLOv10-N et YOLOv11-N en précision tout en conservant des vitesses similaires. Grâce à ces optimisations, YOLOv12 établit un nouveau standard en détection temps

6. ***Focal Loss*** **:** est une fonction de perte proposée par Lin et al. (2017) dans le cadre du modèle RetinaNet, pour résoudre le déséquilibre entre les classes dans les tâches de détection d'objets, notamment dans les détecteurs denses où il y a beaucoup plus d'exemples négatifs (fond) que d'objets à détecter [70]

7. La métrique *mAP* (*mean Average Precision*) mesure la précision moyenne d'un détecteur sur plusieurs classes et seuils d'IoU (*Intersection over Union*). Plus elle est élevée, meilleure est la détection.

8. COCO (Common Objects in Context) est un *dataset* de référence en vision par ordinateur, contenant plus de 200 000 images annotées pour la détection, la segmentation et la reconnaissance d'objets. Site officiel : `https://cocodataset.org/`.

réel, offrant un compromis performant entre précision, latence et efficacité computationnelle.

Enfin, le détecteur **DETR** (Detection Transformer) [72] a introduit les *transformers* dans la détection d'objets. Contrairement aux méthodes classiques, il ne repose pas sur des ancres ni sur un NMS, mais sur une approche dite de *set prediction*, où le modèle apprend à associer un ensemble fixe de prédictions aux objets présents dans l'image. Avec un backbone ResNet-50, DETR atteint une précision proche de Faster R-CNN (environ 42 % de mAP sur COCO) tout en étant plus simple à concevoir. Bien qu'il nécessite un entraînement plus long et qu'il soit moins efficace sur les petits objets, DETR a ouvert la voie à de nouveaux détecteurs basés sur l'attention globale [72].

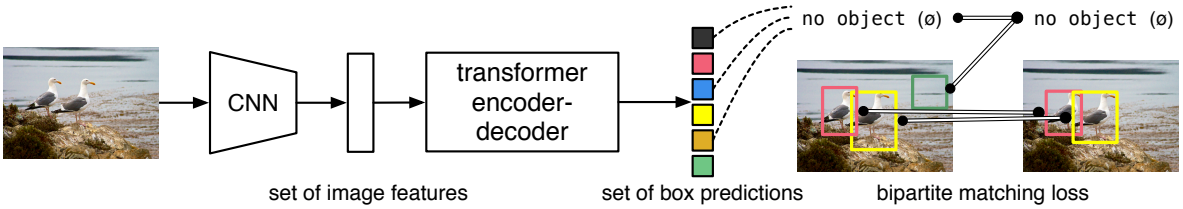


FIGURE 1.9 – DETR prédit directement, en parallèle, les objets présents dans l'image à l'aide d'un CNN suivi d'un transformer. Chaque prédiction correspond à une boîte ou à l'absence d'objet, sans étapes intermédiaires comme les ancres ou la suppression des doublons [72].

3.4.3 Tableau de synthèse comparatif

Le tableau 1.3 présente une synthèse comparative des principales architectures de détection d'objets utilisées dans la littérature récente. Les modèles y sont classés selon plusieurs critères : la précision moyenne (mAP), la robustesse face aux objets de petite taille, au flou et au bruit, la vitesse d'inférence (en images par seconde ou en millisecondes), ainsi que la taille du modèle (nombre de paramètres). Cette vue d'ensemble permet de situer les performances relatives des approches en deux étapes (comme Faster R-CNN) et en une seule étape (comme SSD, YOLO, DETR), tout en tenant compte de leur adéquation aux contraintes de temps réel et aux ressources disponibles.

Modèle (Année)	**Par. (M)**	**mAP**	**Robustesse**	**Vitesse**	**Avantages / Inconvénients**
Faster R-CNN (2015) [68]	∼41	∼42%	Très robuste aux petits objets et au bruit (via RPN)	Faible (∼5 FPS)	+ Précision élevée ; – Pipeline en 2 étapes, latence importante.
SSD (2016) [69]	∼26	72.1% (VOC07)	Moins bon sur petits objets ; tolère moyennement le flou	Très élevée (∼58 FPS)	+ Très rapide ; + Architecture simple ; – Moins précis sur objets petits ; – dépend des ancres.

Suite à la page suivante

Table 1.3 – Suite de la page précédente

Modèle (Année)	**Par. (M)**	**mAP**	**Robustesse**	**Vitesse**	**Avantages / Inconvénients**
RetinaNet (2017) [70]	∼36	∼39%	Bonne gestion multi-échelle et du déséquilibre via focal loss	Moyenne (∼10–20 FPS)	+ Bonne précision ; + Gestion du déséquilibre ; – Plus lent que SSD ; – dépend des ancres.
YOLOv8 (2023) [73]	3.2–60	37.3–53.9%	Bonne stabilité globale, multi-échelle, robuste	Très élevée (v8n ∼1000 FPS sur A100)	+ Temps réel ; + Pipeline simple ; – Moins précis sur très petits objets.
YOLOv12 (2025) [7]	6.5–60	40.6–55.2%	Similaire à YOLOv8, mais meilleure gestion des occlusions	Élevée (v12n ∼610 FPS T4)	+ Équilibre précision/vitesse ; + Attention efficace ; – Entraînement plus lourd.
DETR (2020) [72]	∼41	∼42%	Très bon contexte global ; moins bon sur objets petits	Moyenne (∼25–30 FPS)	+ Sans ancres, sans NMS ; – Convergence lente ; – sensible aux petits objets.

TABLE 1.3 – Comparaison des architectures modernes de détection d'objets sur COCO : précision (mAP), robustesse, vitesse et complexité.

On constate que les modèles en deux étapes, comme Faster R-CNN, offrent généralement une très bonne précision et une robustesse élevée, mais au prix d'une latence importante et d'une complexité accrue. À l'inverse, les modèles en une seule étape, tels que SSD ou YOLOv8, privilégient la vitesse d'exécution avec des compromis sur la précision, notamment pour les petits objets. Les versions récentes comme YOLOv12 introduisent des mécanismes attentionnels permettant de combler partiellement cet écart. Enfin, DETR inaugure une nouvelle classe de détecteurs basés sur l'attention globale et l'apprentissage de correspondances directes, alliant simplicité conceptuelle et bonnes performances, malgré une convergence lente. Ces résultats soulignent l'importance de choisir l'architecture en fonction des exigences spécifiques de l'application visée (précision, temps réel, ressources).

4 *Datasets* de detection et reconaissance des insectes nuisiles

4.1 *Datasets* spécifiques aux insectes nuisibles

La reconnaissance automatique des insectes nuisibles est un enjeu majeur pour l'agriculture, car elle permet de mieux cibler les actions de lutte et de limiter les pertes de récolte. Plusieurs datasets ont été développés pour entraîner et évaluer des modèles d'intelligence artificielle dans ce domaine. Le tableau suivant compare les principaux datasets dédiés à la classification et/ou à la détection des insectes nuisibles, en mettant en évidence leur taille, le type d'annotation, l'équilibre des classes et la qualité des étiquetages.

Nom (Année)	Nb Images	Nb Classes	Type d'annotation	Équilibre des classes	Bruit / Erreurs d'annotation
IP102 (2019) [3]	75 222	102	Classification + Détection	Non (IR≈80) [3]	Complexité visuelle, mais vérifié par 6/8 experts
IP-FSL (2022) [74]	6 817	142 (97+45)	Classification	Oui, équilibré par conception [74]	Non signalé ; sélection stricte sur apparence
Mango Pests (2023) [75]	9 212	10	Détection	Probablement non (pas indiqué)	Annotations manuelles sans contrôle expert ; bruit possible
Dangerous Farm Insects (2023) [76]	1 528	15	Classification	Supposé équilibré (pas de données précises)	Pas de validation par experts ; qualité raisonnable supposée
Forestry Pest Dataset (2022) [77]	7 163	31	Détection	Non [77]	Vérification manuelle par 3 experts + relecture
CPAF (2020) [78]	73 635 (4 909 originales)	19	Classification	Relativement équilibré après augmentation [78]	Non signalé, images nettoyées et augmentées
D0 / Xie (2018) [79]	4 508	40	Classification	Plutôt équilibré (78 images/-classe)	Téléchargement web, pas de validation multiple
Mango Indonésie (2020) [80]	510	16	Classification	Oui (∼32 images/classe) [80]	Non signalé ; étiquetage terrain manuel
Pest24 (2020) [81]	25 378	24	Boîtes englobantes (bounding boxes, multi-target)	Déséquilibre marqué entre les classes	présence de bruit d'annotation dû à la petite taille et à la densité des objets.

TABLE 1.4 – Comparaison des bases de données pour la reconnaissance et la détection des insectes nuisibles : type d'annotation, équilibre des classes, qualité des annotations.

Les caractéristiques clés de ces *datasets* (annotation, équilibre et qualité des labels) révèlent des compromis entre échelle et fiabilité. Nous analysons maintenant chaque dataset en détail :

1. **IP102 (2019)** — Jeu de données de grande envergure pour la classification et la détection des insectes nuisibles, contenant environ 75000 images réparties sur 102 catégories [3]. Il comprend des annotations de boîtes englobantes pour environ 19000 images et présente une distribution de classes fortement déséquilibrée (IR ≈ 80). Les annotations validées par des experts minimisent les erreurs d'étiquetage, bien que les images contiennent du bruit de fond provenant de sources web. Largement utilisé malgré les défis posés par le

déséquilibre des classes [3].

2. **D0/Xie2 (2018)** — Jeu de données de taille moyenne pour la classification, avec 4508 images réparties sur 40 classes de ravageurs des cultures [82, 83]. Les étiquettes sont supposées correctes bien qu'aucune vérification par annotateurs multiples ne soit mentionnée. Sert de pont entre CPAF et des jeux plus petits comme *Mango - Indonesia*.

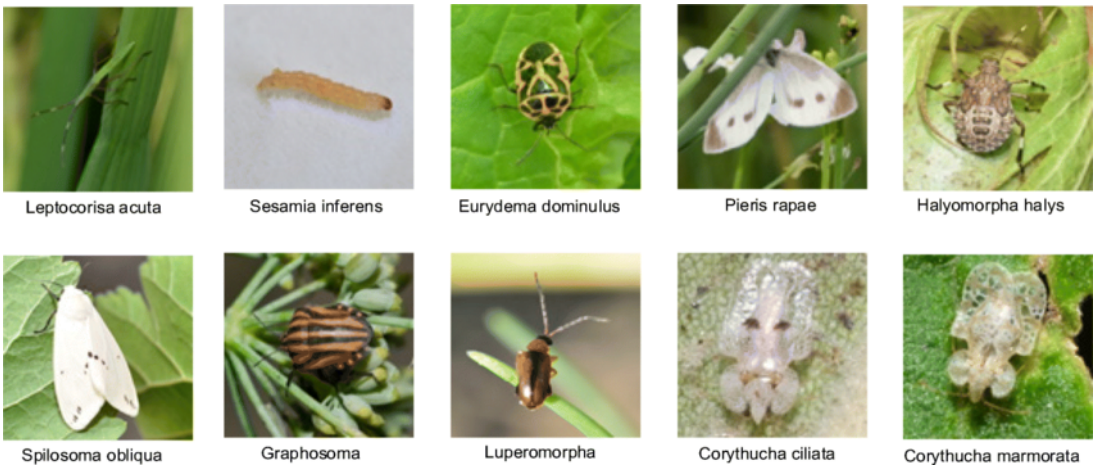


FIGURE 1.10 – Échantillons de 10 insectes différents du *dataset* D0 [83].

3. **IP-FSL (2022)** — Variante du jeu de données IP102 dédiée à l'apprentissage par peu d'exemples (few-shot learning) [74], contenant 6 817 images réparties en 97 classes adultes et 45 juvéniles. Imposition d'un maximum de 50 images par classe pour équilibrer les données. Conçu uniquement pour la classification (sans annotations de détection). Fournit un sous-ensemble plus propre pour la reconnaissance multi-étapes des insectes [74].
4. **Mango Pests (2023)** — Jeu de données Kaggle (non évalué par des pairs) avec annotations au format YOLO pour 10 classes de nuisibles spécifiques à la mangue [75]. Taille modeste (quelques milliers d'images) avec des erreurs d'étiquetage potentielles en raison d'une documentation limitée. Complémentaire au jeu *Mango - Indonesia* ci-dessous.
5. **Dangerous Farm Insects (2023)** — Autre jeu Kaggle pour la classification de 15 types d'insectes nuisibles agricoles [76]. Taille présumée moyenne avec un équilibre des classes, mais absence de métadonnées publiques sur la qualité des annotations.
6. **Forestry Pest Dataset (2022)** — Jeu de données axé sur la détection, contenant 7163 images de 31 espèces d'insectes forestiers [77]. Validation multi-experte (3 annotateurs + volontaires) garantissant la fiabilité des boîtes englobantes. Fort déséquilibre entre les classes, reflétant la distribution réelle dans la nature.
7. **CPAF (2020)** — Jeu de données de classification de 20 nuisibles courants, initialement constitué de 4909 images, augmenté à 73635 [78]. Utilisation de métriques d'exactitude équilibrée pour traiter le déséquilibre résiduel. Plus grand que *D0/Xie2*, mais plus petit que IP102.
8. **Mango - Indonesia (2020)** — Petit jeu de données équilibré de 510 images (16 classes) pour les nuisibles de la mangue [80]. Augmenté à 62047 images pour l'entraînement. Données collectées sur le terrain avec étiquetage manuel. Complémentaire à *Mango Pests*, mais avec une documentation plus rigoureuse.

9. **Pest24 - China (2020)** — Jeu de données à grande échelle contenant 25378 images de terrain annotées avec des boîtes englobantes pour 24 catégories de nuisibles [81]. Conçu pour la détection multi-cibles de très petits objets via l'apprentissage profond. Les images ont été collectées par pièges automatiques dans des champs agricoles. Fortement déséquilibré et difficile en raison de la densité des nuisibles, de leur forte similarité visuelle et du bruit dans les annotations (ex. : boîtes manquantes ou superposées).

4.2 Synthèse comparative des *datasets*

L'analyse systématique des huit *datasets* étudiés permet de dégager quatre enseignements majeurs pour la reconnaissance automatique des insectes nuisibles.

Diversité et exhaustivité Les *datasets* présentent un compromis clair entre couverture taxonomique et profondeur d'annotation. Avec 102 classes et 75 222 images, IP102 [3] constitue la référence la plus exhaustive, alors que les collections spécialisées comme Mango Indonésie [80] (16 classes) offrent une granularité accrue pour des applications ciblées. Notons que les approches d'augmentation artificielle (CPAF [78], Mango Indonésie) permettent de pallier la limitation des petits corpus initiaux.

Rigueur d'annotation La qualité des étiquetages varie significativement :

- Standards élevés : IP102 (validation par 6/8 experts) et Forestry Pest Dataset (3 experts + relecture) [77]
- Documentation limitée : *datasets* Kaggle (Mango Pests [75]) sans protocole de validation explicite

Déséquilibre des classes Seuls IP-FSL [74] et Mango Indonésie atteignent un équilibre intentionnel. Pour IP102, il y a une distribution long-tail [9] extrême (avec IR=80).

Adéquation méthodologique Le choix optimal dépend de la tâche :

- Détection : IP102 et Forestry Pest Dataset (boîtes validées)
- Few-shot learning : IP-FSL (architecture dédiée)
- Classification légère : D0/Xie2 [82] (taille modérée)

Cette analyse comparative souligne l'absence de solution universelle. La sélection doit s'opérer selon un triangle d'exigences : exhaustivité taxonomique, qualité d'annotation, et représentativité écologique - où chaque *dataset* occupe une position distincte.

9. Distribution long-tail : Déséquilibre extrême où quelques classes dominent (ex : 80% des échantillons) tandis que la majorité des classes ont très peu d'exemples (ex : 20% restants répartis sur de nombreuses classes)

5 Travaux connexes basées sur les techniques de *Deep Learning*

Afin de mieux situer l'état de l'art dans le domaine de l'identification automatisée des insectes nuisibles, cette section passe en revue les contributions récentes, en les organisant selon la nature de la tâche traitée. En effet, les travaux existants peuvent être classés en deux catégories : ceux qui se concentrent sur la **classification** (identification d'un insecte dans une image préalablement cadrée), et ceux qui visent la **détection** (localisation de l'insecte dans une scène).

5.1 Méthodes de détection d'insectes nuisibles

Kundur et Mallikarjuna (2022) proposent une détection par **Faster R-CNN** avec backbone EfficientNet (versions B4 et B7) sur IP102. Cinq, 10 ou 15 classes d'insectes sont détectées sur de nouvelles images de blé. Le meilleur modèle (EfficientNet-B7) atteint 99,0% d'accuracy en détection sur 5 classes, 96,0% sur 10 classes et 93,0% sur 15 classes [84]. Il s'agit d'un modèle en deux étapes (RPN + classification) très robuste et lent. Le nombre de paramètres est élevé (66M pour EfficientNet-B7) et le calcul intensif, mais le GPU permettant, ils notent un temps de prédiction réduit. Mobile non applicable.

Albattah et al. (2022) – Les auteurs utilisent une **variante de CornerNet** pour la localisation sans ancres (one-stage anchor-free). Leur "Custom CornerNet" emploie un backbone DenseNet-100 (7,08M paramètres) [85]. Sur IP102 (102 classes), la classification obtenue via ce détecteur est de 68,74% [85], meilleure que des CNN traditionnels étudiés. Ils ne reportent pas le mAP précis, mais leur modèle améliore le recall et la précision sur IP102. La complexité est modérée (7M de paramètres) comparé aux grosses architectures, et l'approche est en temps quasi réel possible (one-stage) [85]. Cependant, ce n'est pas spécifiquement un modèle "mobile".

Liu et al. (2019) – Propose un modèle R-CNN complet nommé **PestNet** pour la détection multi-classe. Trois composantes clés sont ajoutées à Faster R-CNN : un module attention canal/spatial (CSA), un RPN pour proposer des régions, et des cartes de score (PSSM) sans couches FC traditionnelles. Sur leur grand *dataset* MPD2018 (80k images, 16 classes), PestNet atteint 75,46% de mAP [86], surpassant plusieurs méthodes existantes. La structure est en deux étapes (Region Proposal + classification), avec ajout d'attention, donc assez coûteuse en calcul. Le nombre de paramètres n'est pas donné explicitement, mais un RPN + CSA + PSSM rend le modèle lourd [86].

Li et al. (2024) – Ils ont créé **LLA-RCNN** (Lightweight Locality-Aware R-CNN) pour deux *datasets* de ravageurs (fond laboratoire "FP6" et fond naturel "RP5"). Le backbone est MobileNetV3 (très léger), et des blocs d'attention coordonnée sont ajoutés. Sur FP6, LLA-RCNN atteint une précision de détection seulement 0,5% inférieure au meilleur YOLOv8s, tout en utilisant 42% de paramètres en moins (que YOLOv8s) [87]. De plus, LLA-RCNN prédit en 31% de temps en moins par image comparé à YOLOv5s. En somme, c'est un Faster R-CNN très allégé et optimisé pour la rapidité : le modèle est fortement réduit en taille (quelques millions de paramètres) et relativement rapide, ce qui le rend proche d'une solution mobile friendly [87].

Huang et al. (2022) – Proposent un **YOLOX amélioré** pour la détection d'insectes forestiers (utilisant IP102 comme *dataset* de référence). Ils ajoutent de la fusion multi-échelle et d'autres mécanismes. Selon l'article, leur algorithme atteint les « meilleurs résultats » sur IP102 en détection, sans donner de métriques détaillées [88]. On comprend qu'ils surpassent le YOLOX standard. Le modèle reste un one-stage (YOLOX) relativement rapide, avec une complexité modérée (typiquement 30–50M de paramètres selon la taille du modèle YOLOX) [88].

Zhang et al. (2023) – **"C3M-YOLO"** est une version améliorée de YOLOv5 intégrant un module C3M à convolutions déformables et un mécanisme d'attention global (GAM). Sur IP102, ils constatent un gain de +2,4% en précision (P) de détection et +1,7% en mAP0.75 par rapport au YOLOv5 de base [89]. Ils rapportent également que leur mAP0.5 et mAP0.75 excèdent celles d'un modèle YOLOX comparable de 5,1% et 6,2% respectivement [89]. Le modèle reste de la famille YOLOv5 (quelques millions de paramètres), légèrement plus lourd que la version originelle à cause des modules additionnels, mais conserve une vitesse proche du réel [89].

Liu et al. (2024) – Proposent un **YOLOv8 très light pour le riz**. Ils intègrent un module d'attention M-CBAM, une nouvelle fonction de perte (MPDIoU) et des couches Ghost pour alléger le réseau. Le résultat est très convaincant : AP=95,8% et F1=94,6% sur leur *dataset* de test, avec seulement 2,15 millions de paramètres [90]. Autrement dit, un très bon équilibre précision-vitesse : le modèle est ultra léger (2,15M seulement) et atteint une précision quasi parfaite pour la détection des insectes sur riz. Il est clairement conçu pour le mobile/embedded (complexité très faible) [90].

Tang et al. (2021) introduisent **Pest-YOLO**, un détecteur à une étape basé sur YOLOv4 enrichi de pooling spatial et de focal loss. Sur le *dataset* Pest24 [91] (24 classes), Pest-YOLO atteint 69,6% de mAP (et 77,7% de rappel) avec 46FPS en inférence [92]. Les mêmes auteurs améliorent Pest-YOLO (**Improved Pest-YOLO**) en y ajoutant un mécanisme d'attention canal efficace (ECA) et un encodeur Transformer dans la tête de détection ; ils annoncent un mAP 73,4% sur des images de ravageurs [93].

Qi et al. (2023) ont modifié le modèle **YOLOv5m**. Pour l'alléger, ils remplacent son "backbone" par l'architecture efficace GhostNet, et pour améliorer la précision, ils ajoutent un module d'attention (ECA) et une couche de détection supplémentaire dédiée aux cibles de petite taille. Testée sur le *dataset* Pest24, leur version améliore la précision moyenne (mAP) qui passe de 71,5% à 74,1%, tout en doublant quasiment la vitesse pour atteindre 104 images par seconde, ce qui la rend particulièrement adaptée aux applications en temps réel [94].

Pour améliorer la détection des ravageurs en serre, **Dai et al. (2023)** modifient l'architecture de **YOLOv5m**. Ils intègrent une structure de fusion de caractéristiques plus efficace (BIFPN-S) et incorporent un bloc Swin-Transformer dans la tête de détection pour mieux analyser le contexte de l'image grâce à l'auto-attention. Testé sur leur propre *dataset* collecté en laboratoire et en serre (comprenant cinq types de ravageurs), le modèle atteint une précision moyenne (mAP) très élevée de 96,4% [95].

Visant le déploiement sur des appareils mobiles et embarqués, **Zheng et al. (2024)** se concentrent sur l'optimisation du détecteur d'objets YOLO. Ils proposent **Rice-YOLO**, une

version allégée de YOLOv8-n, en y intégrant des modules plus efficaces (comme C2f-Faster) et en simplifiant son architecture pour réduire le coût de calcul. Cette approche leur permet d'atteindre un mAP@0.5 de 78,1% (et un *F1-score* de 74,3%) sur le *dataset* IP102, tout en conservant une taille de modèle très faible et une vitesse d'exécution adaptée au temps réel [96]. D'autres travaux récents suivent cette même tendance, en modifiant YOLOv8 avec des modules CNN légers (**GLU-YOLOv8**) [97] ou des backbones Transformer (**DAMI-YOLOv8**) [98]pour optimiser le compromis entre précision et vitesse.

Zhang et al. (2025) proposent une méthode nommée **Pest-RTDetr** basé sur le modèle RT-DETR [99]. Il intègre un mécanisme d'attention déformable et une fusion multi-échelle pour améliorer la précision sur des cibles denses et de tailles variées (voir figure 1.11). L'approche est évaluée sur IP102, et atteint une *mAP* de 68,4% [4]. Il faut noter que le dernier papier fait aussi la tâche de reconnaissance des insectes nuisibles.

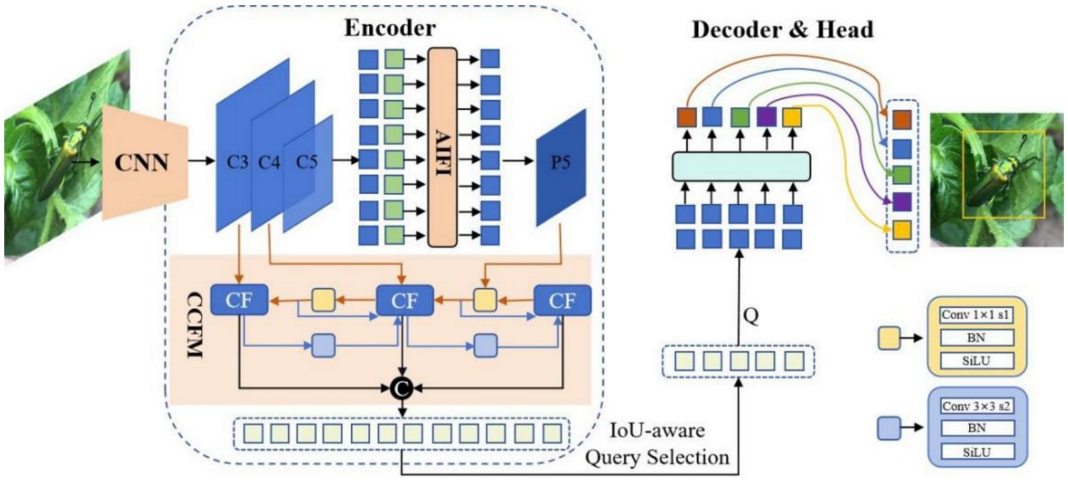


FIGURE 1.11 – Architecture du réseau Pest-RTDetr. Le modèle Pest-RTDetr utilise un CNN pour extraire des caractéristiques, fusionnées par le module CCFM. Les requêtes sont sélectionnées via un mécanisme IoU-aware, puis traitées par le décodeur pour détecter les insectes [4].

5.2 Approches de classification d'images d'insectes nuisibles

Kasinathan et al. (2020) – Les auteurs ont évalué **plusieurs méthodes classiques** (ANN, SVM, KNN, Naïve Bayes, Random Forest) et un CNN sur les bases de données Wang et Xie (respectivement 9 et 24 classes d'insectes). Le CNN a obtenu les meilleurs résultats, avec un taux de reconnaissance maximal de 91,5% [100]. Ils n'indiquent pas directement le nombre de paramètres du modèle, mais l'approche CNN reste d'une complexité modérée par rapport aux méthodes traditionnelles.

Anwar et Masood (2023) – Les auteurs ont conçu **un ensemble de transfert learning** avec VGG16, VGG19 et ResNet50, réunis par vote majoritaire, pour classifier les insectes sur IP102. Avec cette approche "pipeline parallèle", ils ont obtenu 82,5% d'exactitude globale sur IP102 [101]. Ce résultat dépasse plusieurs modèles antérieurs sur la même base. La complexité est élevée car chaque VGG pèse environ 138-143M de paramètres et ResNet50 25M, soit plusieurs centaines de millions au total pour l'ensemble complet. Le modèle n'est pas pensé pour du mobile [101].

Li et al. (2022) – Ce travail utilise le modèle **ResNeXt-50 (32×4d)** en transfert learning avec data augmentation (random crop, CutMix...) sur IP102. Leur meilleure configuration atteint 86,95% de précision de classification après *fine-tuning* [102]. Ils soulignent que le transfert learning réduit drastiquement le temps de convergence. La complexité du modèle est celle de ResNeXt-50, soit environ 25–30M de paramètres, un réseau relativement lourd (non mobile) [102].

Qasim et al. (2025) – L'article présente deux architectures hybrides appelées **PestNet-EF** (fusion précoce) et **PestNet-LF** (fusion tardive). Ces modèles combinent caractéristiques issues de réseaux profonds (convnets) et de descripteurs experts sélectionnés adaptativement (CFS, RFE). En validant sur IP102 (15 classes de ravageurs de riz, maïs, blé), PestNet-EF a obtenu 96% de précision de classification, tandis que PestNet-LF (avec vote majoritaire) tourne autour de 92–94% [103]. Les modèles ont une architecture complexe (plusieurs couches CNN fusionnées, modules d'attention, etc.). Le nombre exact de paramètres n'est pas donné, mais il s'agit d'ensembles de réseaux, donc plutôt massif et non optimisé pour le temps réel [103].

Peng et Wang (2022) – Ils proposent **un réseau hybride mêlant CNN et Transformer** pour la classification de ravageurs. Sur deux petits *datasets* de référence, leur modèle a atteint 99,472% (dataset D0) et 97,935% (dataset « Li ») de précision [104]. Cela illustre une grande puissance de classification (proche de 100%), mais sur des collections assez restreintes (peu d'images). La complexité est très élevée : leur architecture combine des blocs CNN (Inception-V3, ResNet-50) et des Transformers, ce qui implique plusieurs dizaines de millions de paramètres (ResNet50 25M, ViT selon la configuration). Ce n'est clairement pas un modèle mobile [104].

Les auteurs **An et al. (2023)** fusionnent des caractéristiques extraites par un CNN (ResNet152) et deux Transformers (ViT et Swin-T), via un **mécanisme d'attention multi-vue**s. Le classifieur final est un SVM sur les zones importantes (Grad-CAM). Sur IP102 (102 classes), leur approche atteint 65,6% d'exactitude [105], surpassant nettement les CNN classiques (par exemple 61,9% pour FusionSum). La complexité est énorme : ResNet152 (60M paramètres) plus deux gros Transformers, fusionnés, et un SVM. Pas de déploiement mobile envisagé [105].

Ayan et al. (2020) proposent une méthode d'**ensemble** pour améliorer la classification des ravageurs. Leur approche combine les prédictions de deux CNN pré-entraînés (probablement des architectures comme VGG ou ResNet) en leur assignant des poids différents. La nouveauté réside dans l'utilisation d'un algorithme génétique pour rechercher automatiquement la combinaison de poids la plus performante. Sur le grand *dataset* IP102, cet ensemble pondéré et optimisé atteint une précision de 67,13%, surpassant ainsi les scores que chaque modèle obtiendrait individuellement [106].

Feng et al. (2022) présentent le réseau **Ms-ALN.** Basée sur une architecture ResNet-50, leur méthode fusionne les caractéristiques extraites à différentes échelles pour conserver les détails des cibles de petite taille. Un mécanisme d'attention est ensuite appliqué sur ces informations fusionnées pour forcer le modèle à se concentrer sur le ravageur et à ignorer le bruit de fond. Sur le *dataset* public IP102, cette approche atteint 74,61% de précision, surpassant ainsi les performances du ResNet-50 standard [107].

Pour mieux gérer les grandes variations de taille et de posture des ravageurs, **Xu et al.**

(2022) proposent **l'architecture MSCCN**. Ce modèle combine une convolution multi-échelle pour capturer les insectes à différentes échelles, et un *Capsule Network* pour mieux préserver leurs informations spatiales (comme l'orientation), ce qui est souvent perdu dans les CNN classiques. Sur le *dataset* public IP102, leur méthode atteint 92,4% de précision, se montrant plus performante que des modèles de référence tels qu'AlexNet, VGG16 ou GoogLeNet.

Zhao et al. (2022) ont intégré un **module d'attention parallèle (PCSA) à un réseau ResNet-50**. En traitant simultanément les informations d'attention spatiale et de canal, le modèle parvient à mieux extraire les caractéristiques du ravageur tout en supprimant le bruit de fond complexe. Sur un *dataset* de 10 classes de ravageurs, cette approche atteint une précision de 98,17%, une amélioration notable par rapport au modèle de base, au prix d'une très légère augmentation de la taille et du temps de calcul du modèle [108].

Chen et al. (2023) présentent une méthode de **fusion multi-images** pour améliorer la robustesse de la classification (voir Figure 1.12). Leur approche se base sur l'analyse de cinq clichés d'un même insecte, en appliquant une localisation de caractéristiques et un filtrage adaptatif pour consolider les informations les plus fiables. Cette technique leur permet d'atteindre 96,1% de précision, bien qu'elle implique une contrainte forte sur la capture des données [109].

Nanni et al. (2021) – Une approche **ensembliste regroupant cinq architectures CNN différentes** (ResNet50, GoogLeNet, ShuffleNet, MobileNetv2, DenseNet201) a été proposée. Chaque CNN a été entraîné avec des méthodes de data augmentation et deux variantes Adam novatrices. Les meilleurs résultats en classification ont été atteints avec l'ensemble augmentations/Adam, donnant 95,52% de précision sur le dataset Deng (SMALL) et 73,46% sur IP102 [110]. La complexité est très élevée car on combine cinq réseaux profonds pré-entraînés de grande taille, ce qui implique plusieurs dizaines de millions de paramètres (par ex. DenseNet201). Ce modèle est assez lourd et non adapté au mobile.

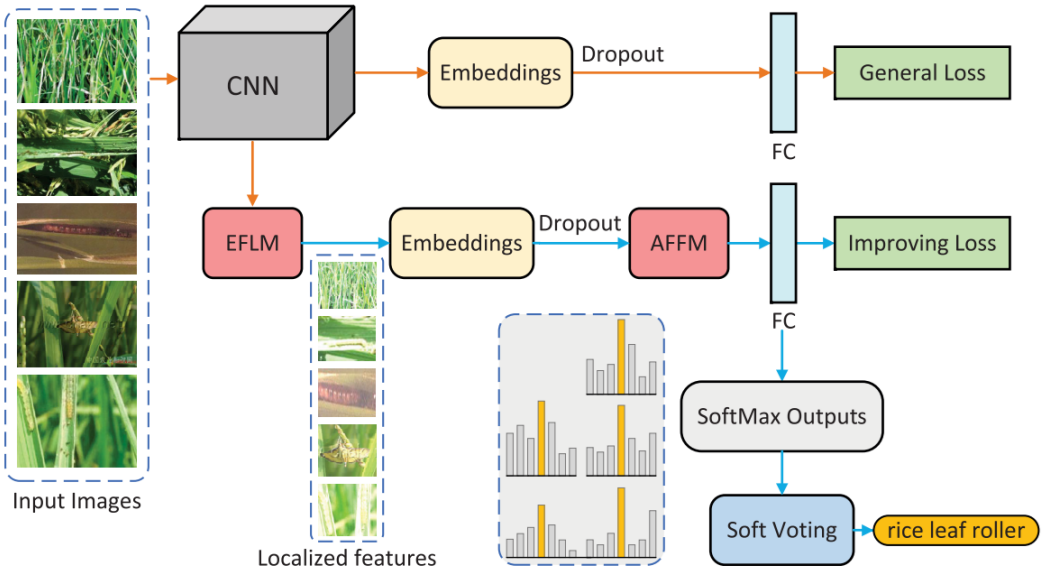


FIGURE 1.12 – Méthode de classification avec localisation et vote soft. Le modèle extrait des caractéristiques via un CNN, localise les régions d'intérêt avec EFLM, filtre et fusionne les informations utiles avec AFFM, puis applique un vote soft pour améliorer la précision finale de classification des ravageurs [109].

5.3 Comparaison des travaux et synthèse

Ce tableau comparative (tableau 1.5 présente de manière structurée les principales approches de classification et de détection des insectes nuisibles évoquées précédemment. Chaque modèle est comparé selon la nature de la tâche (classification ou détection), le type d'architecture utilisé, les performances obtenues (accuracy ou mAP) sur des bases de données spécifiques (comme IP102, Deng ou Pest24), la complexité du modèle (en termes de paramètres ou d'architecture), ainsi que sa capacité ou non à être déployé sur des dispositifs mobiles. Cette analyse met en évidence les compromis actuels entre précision, légèreté et adaptation au temps réel, en particulier pour les applications mobiles.

Modèle (Année)	**Tâche**	**Type d'approche**	***Accuracy* (Dataset)**	**Complexité**	**Mobile**
Faster R-CNN (2022) [84]	Détection	R-CNN (EffNet-B7)	99,0 % (5 cl.), 93,0 % (15 cl.) (IP102)	Très élevée (~66M + RPN)	Non
Custom CornerNet (2022) [85]	Détection	One-stage (DenseNet100)	68,74 % (IP102)	Modérée (~7M)	Non
PestNet (2019) [86]	Détection	Faster R-CNN + attention + PSSM	mAP 75,46 % (MPD2018)	Élevée (multi-modules)	Non
LLA-RCNN (2024) [87]	Détection	MobileNetV3 + Attention coordonnée	~0,5 % en moins que YOLOv8s	Faible (~4M, optimisé)	Oui
YOLOX amélioré (2022) [88]	Détection	YOLOX + fusion multi-échelle	—	Moyenne (~30–50M)	Non
C3M-YOLO (2023) [89]	Détection	YOLOv5 + C3M + GAM	+2,4 % précision (IP102)	Moyenne (YOLOv5++)	Non
YOLOv8 léger (2024) [90]	Détection	YOLOv8 + Ghost + M-CBAM	AP 95,8 % (riz)	Très faible (2,15M)	Oui
Pest-YOLO (2021) [92]	Détection	YOLOv4 + pooling spatial + focal loss	mAP 69,6 % (Pest24)	Moyenne (YOLOv4)	Oui
Improved Pest-YOLO (2023) [93]	Détection	YOLOv4 + ECA + Transformer	mAP 73,4 %	Élevée	Non
YOLOv5m GhostNet (2023) [94]	Détection	YOLOv5m + GhostNet + ECA	mAP 74,1 % (Pest24)	Moyenne (~5M)	Oui
YOLOv5m + Swin (2023) [95]	Détection	YOLOv5m + BIFPN + Swin	mAP 96,4 % (greenhouse)	Élevée	Non

Suite à la page suivante

Table 1.5 – suite de la page précédente

Modèle (Année)	**Tâche**	**Type d'approche**	***Accuracy* (Dataset)**	**Complexité**	**Mobile**
Rice-YOLO (2024) [96]	Détection	YOLOv8n allégé + C2f	mAP@0.5 78,1 %, F1=74,3 % (IP102)	Faible (mobile-ready)	Oui
GLU-YOLOv8 (2024) [97]	Détection	YOLOv8 + CNN légers	—	Modérée	Oui
DAMI-YOLOv8 (2024) [98]	Détection	YOLOv8 + backbone Transformer	—	Élevée	Non
Pest-RTDETR (2025) [4]	Détection	RT-DETR + attention déformable	mAP 68,4 % (IP102)	Élevée	Non
CNN (2020) [100]	Classification	CNN standard	91,5 % (Wang/Xie)	Modérée	Non
TL Ensemble (2023) [101]	Classification	VGG16 + VGG19 + ResNet50	82,5 % (IP102)	Très élevée	Non
ResNeXt-50 (2022) [102]	Classification	CNN transf. learning	86,95 % (IP102)	Moyenne (~30M)	Non
PestNet-EF (2025) [103]	Classification	CNN + features + vote	96 % (IP102)	Élevée	Non
CNN + Transformer (2022) [104]	Classification	InceptionV3 + ResNet50 + ViT	99,47 % (D0), 97,94 % (Li)	Très élevée	Non
Multi-view + SVM (2023) [105]	Classification	ResNet152 + ViT + Swin + SVM	65,6 % (IP102)	Très élevée	Non
Genetic ensemble (2020) [106]	Classification	CNN (GA pondéré)	67,13 % (IP102)	Élevée	Non
Ms-ALN (2022) [107]	Classification	ResNet50 + attention multi-échelle	74,61 % (IP102)	Élevée	Non
MSCCN (2022) [111]	Classification	CNN multi-échelle + Capsule	92,4 % (IP102)	Élevée	Non
PCSA-ResNet (2022) [108]	Classification	ResNet50 + attention parallèle	98,17 % (10 classes)	Moyenne	Oui
Fusion multi-images (2023) [109]	Classification	CNN + AFFM + soft voting	96,1 %	Élevée	Non
Ensemble CNN (2021) [110]	Classification	5 CNN (ResNet, DenseNet...)	95,52 % (Deng), 73,46 % (IP102)	Très élevée	Non

Suite à la page suivante

Table 1.5 – suite de la page précédente

Modèle (Année)	Tâche	Type d'approche	*Accuracy* (Dataset)	Complexité	Mobile

TABLE 1.5 – Comparaison des modèles de classification et de détection d'insectes nuisibles selon la précision, la complexité, et l'orientation mobile.

5.3.1 Synthèse des travaux connexes

L'analyse des travaux récents sur l'identification des insectes nuisibles révèle deux tendances principales qui définissent l'état de l'art actuel.

Tendance 1 : La quête de la performance maximale Une part importante de la recherche vise l'obtention de la plus haute précision possible, souvent au détriment de l'efficacité computationnelle. Dans le domaine de la détection, des modèles en deux étapes comme Faster R-CNN démontrent une robustesse et une précision remarquables, atteignant jusqu'à 99,0% de précision sur des sous-ensembles de classes. De même, pour la classification, les approches les plus performantes reposent sur des ensembles de plusieurs réseaux de neurones profonds (VGG, ResNet50) , des architectures hybrides complexes mêlant CNN et Transformers , ou des fusions multi-images. Ces méthodes atteignent des scores de précision très élevés, dépassant parfois 96%. Cependant, leur coût est une complexité très élevée, avec des dizaines, voire des centaines de millions de paramètres, les rendant inapplicables pour un déploiement sur des appareils mobiles.

Tendance 2 : L'émergence des modèles légers pour le déploiement mobile En opposition à la première tendance, une part croissante de la recherche se concentre sur le développement de modèles légers et rapides, spécifiquement conçus pour les applications mobiles et en temps réel. Cette approche est particulièrement visible dans le domaine de la détection, avec une prolifération de variantes optimisées de la famille YOLO. Des stratégies comme l'utilisation de backbones efficaces (par exemple MobileNetV3 ou GhostNet) , l'intégration de modules d'attention légers (M-CBAM, ECA) et la simplification architecturale permettent de concevoir des modèles avec très peu de paramètres (parfois seulement 2,15 millions) tout en conservant une excellente précision et une vitesse très élevée. Des modèles comme Rice-YOLO ou LLA-RCNN illustrent parfaitement ce compromis réussi entre légèreté et performance, rendant la détection sur le terrain réalisable.

Observations et lacunes identifiées En conclusion, l'état de l'art est caractérisé par un compromis entre des modèles de haute précision mais lourds et non-mobiles, et des modèles de détection légers et rapides dont la précision peut être en retrait sur des tâches complexes. Des stratégies transversales comme la fusion de caractéristiques , l'utilisation de mécanismes d'attention et le transfert d'apprentissage sont couramment employées pour améliorer les performances dans les deux cas. Le défi majeur reste de combiner la haute performance des approches

complexes avec l'efficacité des architectures légères pour créer des solutions de bout en bout, de la détection à la classification, qui soient à la fois précises et déployables en conditions réelles.

6 Conclusion

Ce chapitre a tracé l'évolution des méthodes de reconnaissance des insectes nuisibles, depuis les approches conventionnelles jusqu'à la révolution apportée par le Deep Learning. Nous avons d'abord exploré les techniques traditionnelles basées sur l'expertise humaine et l'analyse d'images classique, en soulignant leurs limites en termes de précision, d'automatisation et d'adaptabilité. Ces faiblesses ont ouvert la voie à des solutions plus robustes et automatisées.

L'avènement du Deep Learning a constitué un changement de paradigme, en permettant l'apprentissage automatique de caractéristiques directement à partir des données brutes. Nous avons détaillé le fonctionnement des architectures fondamentales que sont les Réseaux de Neurones Convolutifs (CNN) et les Vision Transformers (ViT) , ainsi que l'émergence de modèles de référence, qu'ils soient pionniers (AlexNet, VGG), profonds (ResNet, DenseNet) ou optimisés pour l'efficacité et le déploiement mobile (EfficientNet, FastViT). Une analyse similaire a été menée pour les architectures de détection d'objets, distinguant les approches en deux étapes (Faster R-CNN) des modèles en une étape conçus pour le temps réel (famille YOLO , DETR). L'étude des bases de données disponibles, comme IP102 , a également souligné l'importance des données de qualité, tout en relevant les défis liés au déséquilibre des classes et à la rigueur des annotations.

La synthèse des travaux récents a confirmé l'existence d'un compromis majeur dans le domaine : un arbitrage constant entre la recherche de la plus haute précision, souvent atteinte avec des modèles très complexes et lourds , et la nécessité de développer des solutions légères et rapides adaptées aux contraintes du terrain. Il en ressort un besoin clair pour des systèmes intelligents qui intègrent à la fois une détection et une reconnaissance précises, tout en étant suffisamment efficaces pour être déployés sur des dispositifs mobiles.

Dans le chapitre suivant, nous présenterons la conception, l'architecture générale et les différentes étapes de développement de notre système intelligent de reconnaissance des insectes nuisibles, pensé pour répondre à ce défi.

Chapitre 2

Conception de l'approche proposée

1 Introduction

Dans ce chapitre, nous proposons une approche complète pour la détection et l'identification d'insectes en temps réel. Notre système repose sur une architecture modulaire conçue en deux phases distinctes : une phase de **Détection** pour localiser les insectes dans une image, suivie d'une phase de **Reconnaissance** pour identifier leur espèce. Pour ce faire, nous nous appuyons sur le modèle **YOLOv12** , choisi pour son excellent compromis entre vitesse et précision de détection, et sur le modèle **FastViT-SA12**, reconnu pour son efficacité sur les dispositifs à ressources limitées. L'ensemble du pipeline, qui suit le schéma **Détection** → **Classification**, a été développé et évalué sur le Dataset de référence **IP102**. Cette approche modulaire se justifie par la possibilité d'optimiser chaque composant indépendamment, garantissant ainsi une robustesse et une performance accrues pour le système global. Nous détaillerons dans les sections suivantes la conception de chaque module, les stratégies de préparation des données et les méthodologies d'entraînement spécifiques mises en oeuvre.

2 Dataset utilisé

Dans notre travail, nous avons utilisé l'ensemble de données **IP102** [3]. *Le Dataset* IP102 contient **75 222 images** réparties en **102 classes** d'insectes nuisibles. Il est utilisé principalement pour les tâches de *classification*. Pour les besoins de la *détection*, une sous-partie de **18 983 images** a été annotée avec des boîtes englobantes par des experts.Nous disposons donc de deux *Datasets* distincts :

1. Un ensemble de **classification** (75 222 images) avec une étiquette de classe par image ;
2. Un sous-ensemble de **détection** (18 983 images) avec des annotations de localisation (bounding boxes).

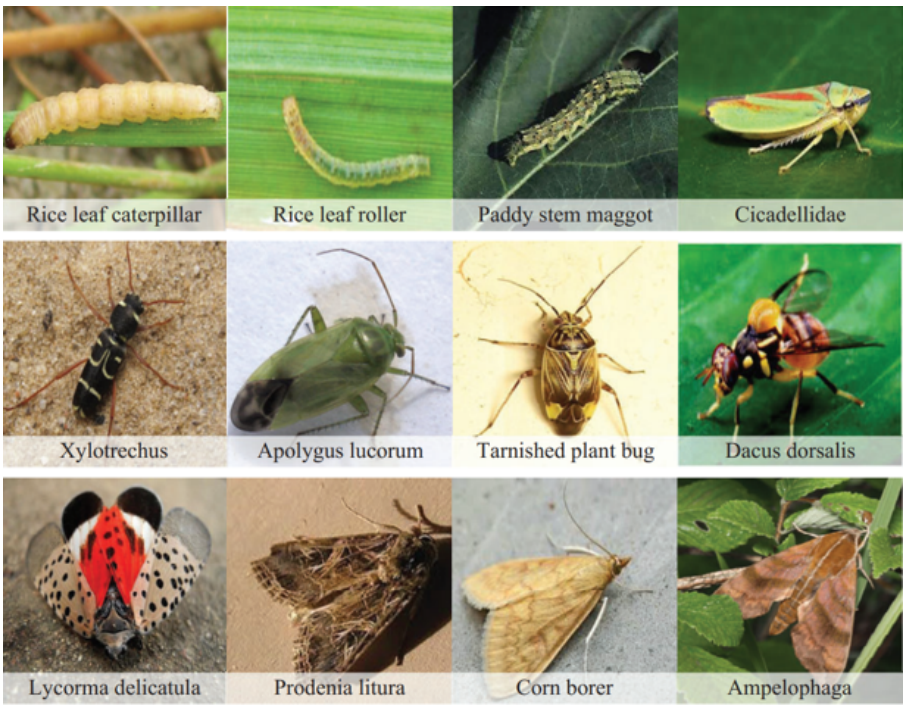


FIGURE 2.1 – Exemples d'images de l'ensemble de données [3].

2.1 Collection des données

Les images ont été collectées sur Internet, en particulier à partir de **sites spécialisés en agriculture et en entomologie**, ainsi que via des **moteurs de recherche d'images** tels que **Google**, **Flickr** et **Bing**.

Pour chaque espèce d'insecte, les chercheurs ont utilisé son **nom anglais** ainsi que ses **synonymes** comme mots-clés de recherche. Jusqu'à **2 000 images** ont été récupérées par terme.

Les images ont ensuite été filtrées afin de ne conserver que celles jugées pertinentes, capturées **dans des conditions réelles sur le terrain**, et illustrant clairement les insectes ciblés.

Au total, plus de **300 000 images candidates** ont été rassemblées lors de cette phase initiale. Après un processus de filtrage manuel réalisé par des experts, environ **75 000 images** ont été finalement retenues[3].

2.2 Annotation des données

L'annotation des données a été réalisée par des **experts agricoles**, spécialisés dans les principales cultures affectées par les insectes nuisibles. Chaque expert était chargé d'étiqueter les images liées à une culture spécifique, selon une taxonomie hiérarchique définie à l'avance [3].

Pour faciliter le processus, un **système de questions/réponses** a été mis en place, permettant aux experts d'assigner à chaque image une **catégorie d'insecte** parmi 102 classes possibles. Afin de garantir la fiabilité des annotations, plusieurs experts ont validé chaque image selon un **protocole collaboratif**[3].

Concernant la **détection des insectes**, un **sous-ensemble d'images a été sélectionné aléatoirement** pour être annoté avec des **bounding boxes** [3]. Ces annotations, réalisées

par les mêmes experts, suivent le format standard Pascal VOC[112] et permettent de localiser précisément les insectes dans l'image. Cette étape est essentielle pour entraîner des modèles de détection comme YOLO.

Voici un **exemple d'annotation** d'une image dans ce format : Le fichier XML ci-dessous décrit une image nommée *IP001000016.jpg*, de taille 199×163 pixels. L'insecte détecté appartient à la classe "1", et sa position est indiquée par un rectangle avec les coordonnées xmin = 10, ymin = 54, xmax = 189 et ymax = 123.

```
<annotation>
    <folder>IP103_final_new1</folder>
    <filename>IP001000016.jpg</filename>
    <path>/home/ubuntu2/Desktop/IP103_final_new1/IP001000016.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>199</width>
        <height>163</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>1</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>10</xmin>
            <ymin>54</ymin>
            <xmax>189</xmax>
            <ymax>123</ymax>
        </bndbox>
    </object>
</annotation>
```

2.3 Répartition du dataset

Le dataset **IP102** contient **75 222 images** réparties en **102 classes** d'insectes nuisibles. Certaines classes sont très peu représentées, avec parfois seulement **71 images**, ce qui rend la répartition importante pour assurer une évaluation fiable. Selon les auteurs [3], cette partition garantit une distribution équilibrée des données, permettant au modèle d'apprendre efficacement à partir de l'ensemble de formation, et d'évaluer les performances sur l'ensemble de test inédit.

Pour les expériences de **classification**, les images ont été réparties selon un ratio approximatif de **60 % pour l'entraînement**, **10 % pour la validation**, et **30 % pour les tests**. Cela donne :

- **45 095 images** pour l'entraînement
- **7 508 images** pour la validation
- **22 619 images** pour les tests

Pour la tâche de **détection**, un **sous-ensemble de 18 983 images annotées avec des boîtes englobantes** a été utilisé. Ce sous-ensemble est divisé comme suit :

- **15 178 images (80%)** du dataset ont été utilisées pour l'entraînement et la validation.
- **3 798 images (20%)** ont été réservées pour les tests.

2.4 Classes et structure du dataset

Le **dataset IP102** contient **102 classes d'insectes nuisibles**, couvrant les espèces les plus courantes qui affectent les cultures agricoles. Pour organiser ces classes de manière cohérente, une **structure hiérarchique** a été mise en place.

Chaque classe (espèce d'insecte) est regroupée selon la **culture principale** qu'elle affecte. On distingue ainsi **8 cultures** (par exemple : riz, maïs, blé, agrumes, etc.), elles-mêmes regroupées en deux **super-classes** :

- **FC (Field Crops)** – cultures de plein champ
- **EC (Economic Crops)** – cultures à haute valeur économique

Le tableau ci-dessous présente la liste des super-classes, des classes associées, ainsi que le **nombre d'images** disponibles pour chaque classe (données à extraire du **dataset IP102**) :

Super-classe		Nombre de classe	Classe (espèce)	Nombre d'images	Total
FC	Rice	14	Rice leaf roller, Yellow rice borer, ...	8417	40660
	Corn	13	Grub, Mole cricket, Corn borer, ...	14015	
	Wheat	9	Green bug, Wheat sawfly, ...	3418	
	Beet	8	Beet fly, Beet weevil, ...	4420	
	Alfalfa	13	Alfalfa weevil, Flax budworm, ...	10390	
EC	Vitis	16	Xylotrechus, Aphidoidea, ...	17551	34562
	Citrus	19	Icerya purchasi, Aphis citricola, ...	7273	
	Mango	10	Dasineura sp., Cicadellidae, ...	9738	
Totale classe		102	Total image IP102	75222	

TABLE 2.1 – Répartition du nombre de classes et d'images par super-classe dans le dataset IP102[3].

Pour l'ensemble des données de détection, il est utile de connaître combien d'objets annotés (bounding boxes) sont présents dans chaque image. La figure ci-dessous montre la distribution

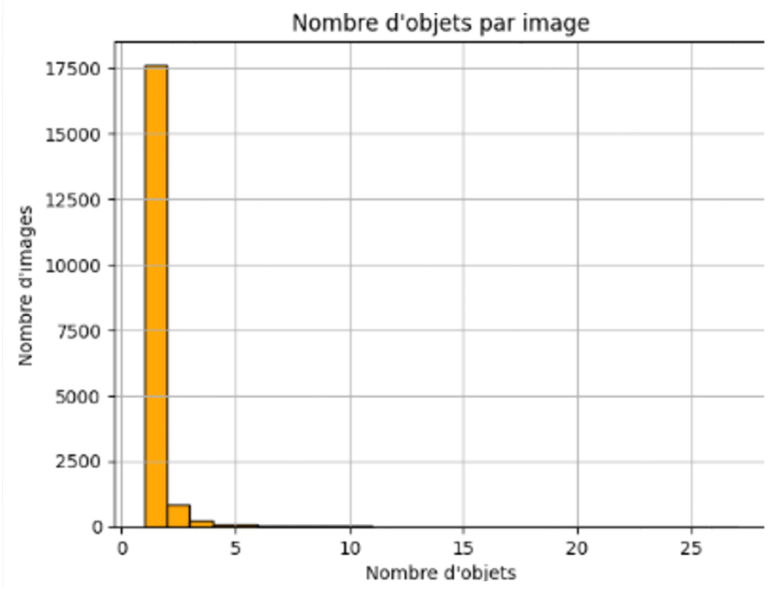


FIGURE 2.2 – Nombre de bounding boxes par image.

du nombre d'objets par image. On observe que la majorité des images contient **un seul objet**, tandis que très peu en contiennent plusieurs. Ce type de distribution indique que la plupart des échantillons sont **simples à traiter**, ce qui est généralement favorable à l'entraînement d'un modèle de détection comme YOLOv12. Toutefois, quelques images présentent plus de *5* objets, ce qui peut introduire de la **variabilité dans la complexité visuelle**.

2.5 Exploration des défis liés aux données d'IP102

Afin d'évaluer la pertinence du *dataset* IP102 pour la tâche de détection d'insectes nuisibles, **une analyse visuelle des données a été réalisée**. Cette analyse a permis de mieux comprendre la **distribution** des échantillons entre les classes, **d'extraire les tailles des insectes** présents dans les images et de mettre en évidence une difficulté supplémentaire liée à la **diversité biologique** des espèces.

1er challenge (Diversité biologique) : Le *dataset* **IP102** présente une grande **diversité biologique**, ce qui constitue à la fois une richesse et un défi pour les modèles de reconnaissance. En effet, les insectes peuvent apparaître sous différentes formes correspondant aux **stades de leur cycle de vie** : œuf (a_1), larve (a_2), chrysalide (a_3) et adulte (a_4) [4]. **La Figure ci-dessous** illustre ces quatre étapes.

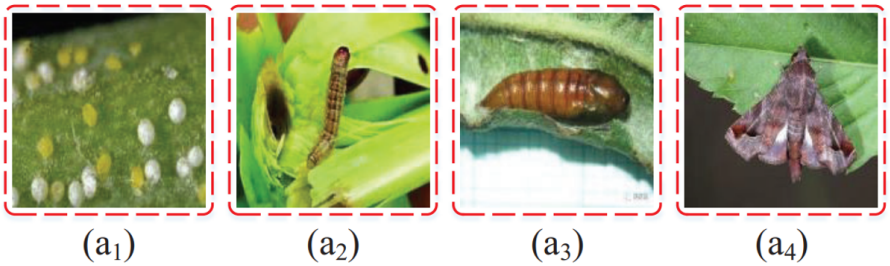


FIGURE 2.3 – Exemples d'un insecte à différents stades – (a_1) œuf, (a_2) larve, (a_3) chrysalide, (a_4) adulte [4].

Reconnaître une même espèce à travers toutes ces formes est difficile, car leur **apparence change considérablement** selon la phase. Cela complique la tâche des modèles de classification, qui doivent apprendre à regrouper des images très différentes sous une même classe.

2ème challenge (Imbalanced dataset) : Un autre défi majeur du *dataset* est le **déséquilibre entre les classes (imbalanced dataset)**. Certaines classes contiennent plusieurs milliers d'images, tandis que d'autres en ont à peine quelques dizaines. Ce déséquilibre peut **biaiser l'apprentissage du modèle**, qui risque de mieux performer sur les classes surreprésentées [4]. **La figure(4)** illustre la distribution des classes du dataset IP102.

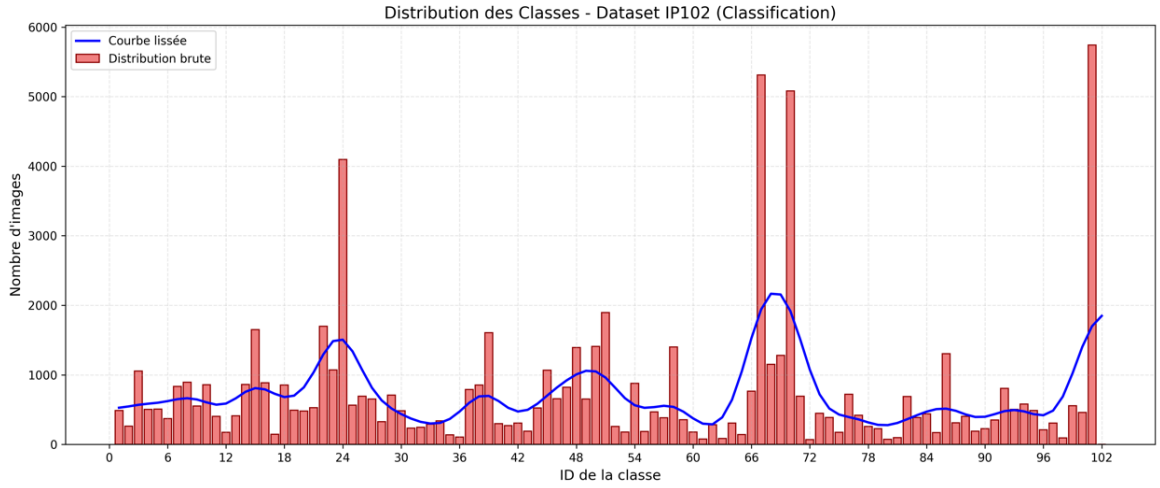


FIGURE 2.4 – Distribution des classes du Dataset IP102.

Wu, Xiaoping, et al. [3] ont montré que plusieurs classes présentent un**Imbalance Ratio (IR) supérieur à 9**, ce qui souligne la gravité de ce déséquilibre. Il faut noter que **l'IR (Imbalance Ratio)** correspond au **rapport entre le nombre d'échantillons de la classe majoritaire et celui de la classe minoritaire**, et qu'un IR élevé indique un fort déséquilibre dans la distribution des données [4]. Donc avant d'entraîner le modèle, il est important de prendre en compte cette inégalité, notamment en appliquant une **data augmentation ciblée** afin d'augmenter artificiellement le nombre d'images dans les classes minoritaires.

3ème challenge (variabilité des tailles des insectes) : En plus de la diversité biologique et du déséquilibre entre les classes, un autre défi majeur réside dans la **variabilité des tailles des insectes** dans les images. Ces objets peuvent apparaître très petits, partiellement visibles ou dans des échelles très différentes selon l'espèce ou son stade de développement. Cette **variabilité multi-échelle** représente une difficulté importante pour les modèles de détection, qui doivent être capables d'identifier efficacement des objets de tailles très variées.

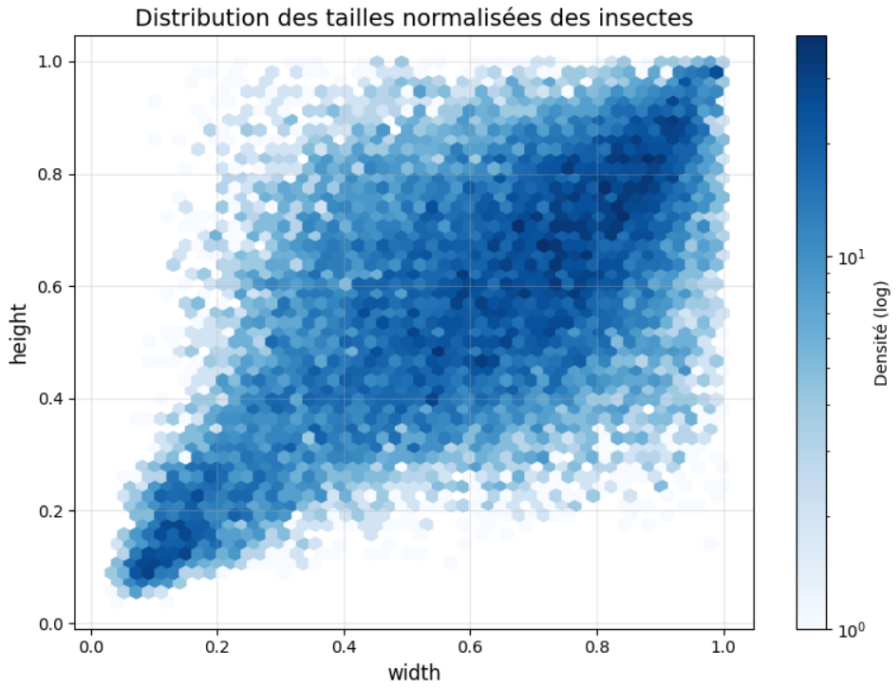


FIGURE 2.5 – Distribution des tailles normalisées des insectes.

parfois peu contrastés par rapport au fond. La **figure** 5 montre cette diversité à travers la répartition des hauteurs et largeurs normalisées des insectes dans l'ensemble IP102. Cette distribution confirme que le dataset reflète des **conditions réalistes** de terrain, mais aussi qu'il exige des modèles robustes capables de gérer ces différences d'échelle.

3 Vue d'ensemble de l'architecture modulaire du système

Le système proposé est fondé sur une architecture modulaire comportant deux phases principales : la détection et la reconnaissance.

- **Phase de détection** : cette première étape repose sur l'utilisation d'une sous-partie annotée du *dataset* **IP102**, convertie au format **YOLO**, afin d'entraîner un modèle de détection **YOLOv12**. Ce modèle est reconnu pour son excellent compromis entre *vitesse d'inférence* et *précision de détection*, ce qui le rend particulièrement adapté aux applications en temps réel.
- **Phase de reconnaissance** : utilisation du modèle **FastViT-SA12**, choisi pour son équilibre entre légèreté, rapidité et précision, appliqué sur des images prétraitées issues de la partie classification du *dataset* IP102.
- **Pipeline global** :
  - Les régions détectées par YOLOv12 sont d'abord extraites.
  - Ces régions sont ensuite transmises à FastViT-SA12 pour l'identification des espèces.

  Cette organisation en deux étapes permet d'optimiser chaque module individuellement tout en améliorant les performances globales du système.

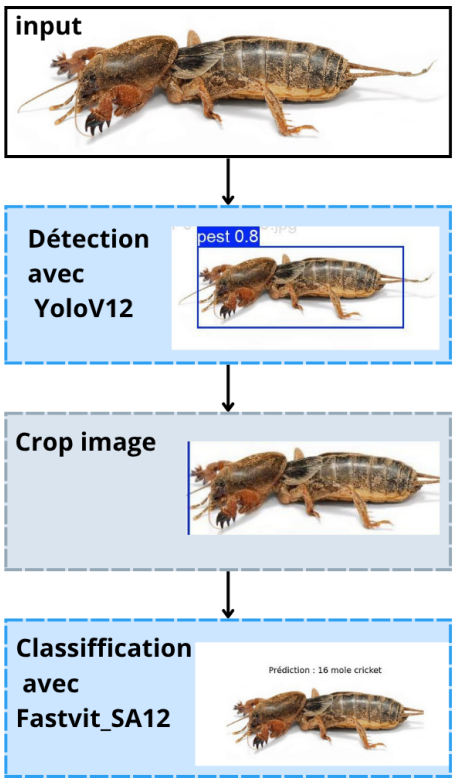


FIGURE 2.6 – Schéma global de l'architecture modulaire pour la détection et la reconnaissance des insectes nuisibles.

4 Partie 1 : Module de détection d'insectes avec YOLOv12

Avant de pouvoir reconnaître les espèces d'insectes nuisibles, il est d'abord nécessaire de les localiser dans l'image, c'est-à-dire déterminer précisément leur position. Cette phase de détection constitue une étape cruciale, car elle permet au système de se concentrer uniquement sur les zones pertinentes de l'image avant d'appliquer une classification fine.

Dans cette partie, nous présentons le module de détection basé sur le modèle YOLOv12, en détaillant les étapes de préparation des données, de data augmentation, ainsi que le fine-tuning du modèle pré-entraîné.

La Figure 2.7 illustre l'ensemble du pipeline, depuis l'annotation du dataset IP102 jusqu'à la détection finale d'un insecte dans une image test. Ce schéma montre clairement :

- la conversion des annotations vers le format YOLO,
- l'application de techniques de data augmentation sur le trainset uniquement,
- l'entraînement supervisé du modèle YOLOv12 (initialisé à partir d'un modèle pré-entraîné),
- l'utilisation du modèle entraîné pour détecter les insectes dans de nouvelles images.

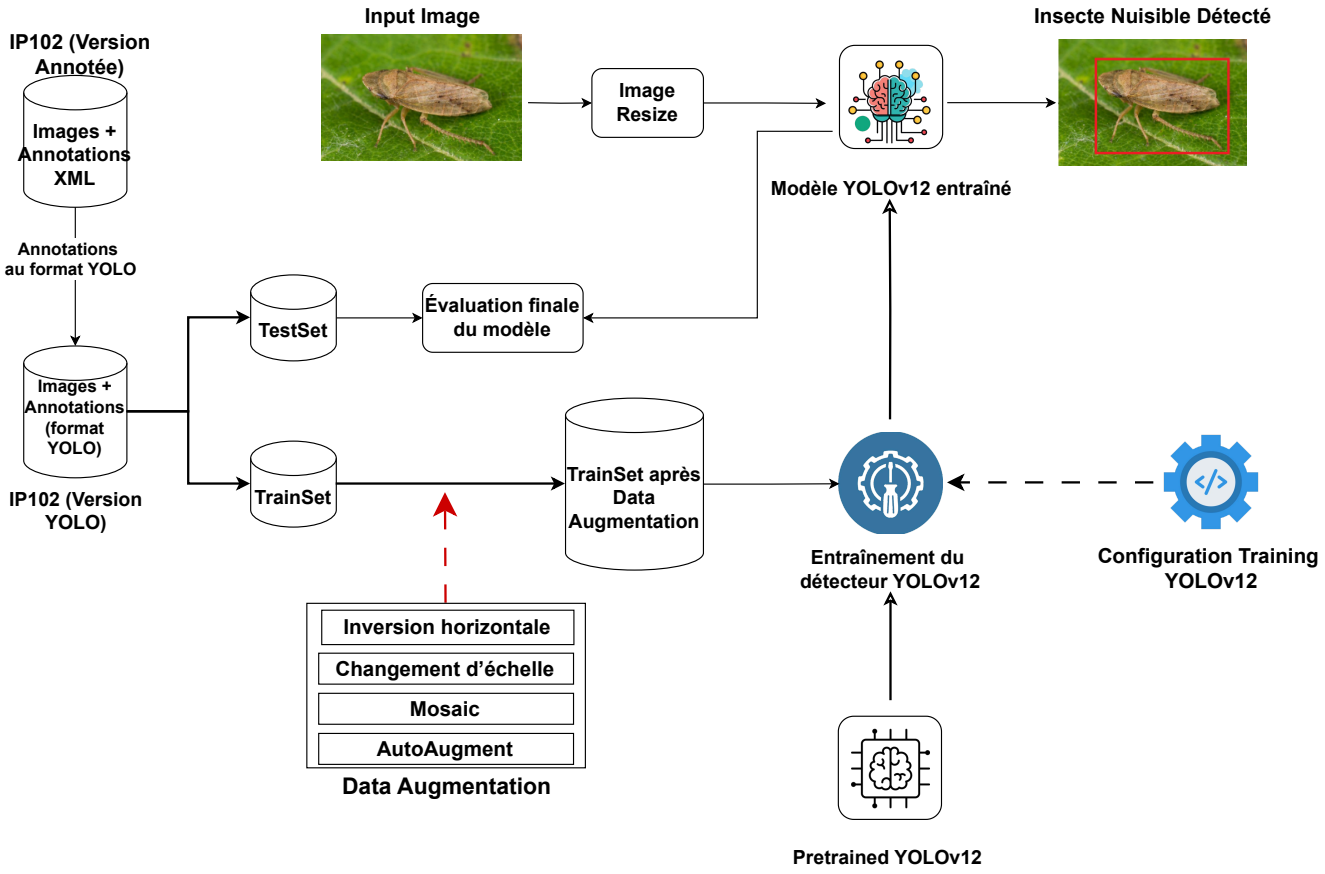


FIGURE 2.7 – Schéma proposé de détection des insectes nuisibles avec YOLOv12

4.1 Pertinence de YOLOv12 pour la détection d'insectes nuisibles

La détection des insectes nuisibles présente plusieurs défis, en particulier en temps réel dans les applications agricoles qui sont utilisées dans des environnements complexes pleins de détails (par exemple, la végétation, les ombres et les tissus naturels) qui peuvent affecter la précision du modèle. En outre, les insectes sont de petits organismes souvent cachés sous les feuilles. Aussi, la variabilité des espèces en termes de forme, de couleur et de stade de développement exige que le modèle soit très précis dans sa généralisation.

YOLOv12 est un choix approprié pour ces contraintes, car il est plus précis que les versions précédentes (YOLOv5/YOLOv8). Ceci est dû à l'incorporation de **mécanismes d'attention** avancés tels que Area Attention et FlashAttention. Ces mécanismes permettent au modèle de se concentrer sur **les régions pertinentes des images, même lorsque les objets sont petits et ne sont pas entièrement visibles**. Il intègre également une optimisation multi-échelle améliorée via des FPN (*Feature Pyramid Network*)[1] optimisés [5], ce qui **accroît sa capacité à détecter des objets de petite taille tels que des insectes**. La caractéristique la plus importante qui fait de YOLOv12 le meilleur choix est sa **vitesse de décection** [7], qui

1. FPN (Feature Pyramid Network) est un composant architectural qui améliore la détection d'objets à différentes échelles. Il combine des caractéristiques à basse résolution et riches en sémantique (provenant des couches profondes) avec des caractéristiques à haute résolution mais pauvres en sémantique (issues des couches superficielles).

est la chose la plus importante dans les applications en temps réel.

Une comparaison directe avec les versions précédentes montre que YOLOv12 offre le meilleur équilibre entre précision et rapidité [5]. Par exemple, dans les résultats officiels de YOLOv12 (voir figure 2.8), le modèle surpasse YOLOv8 et YOLOv11 en termes de précision moyenne (mAP), tout en conservant une vitesse d'inférence égale ou supérieure. Grâce à ces caractéristiques, YOLOv12 est un choix approprié pour une détection précise, rapide et fiable des insectes nuisibles dans des environnements naturels complexes.

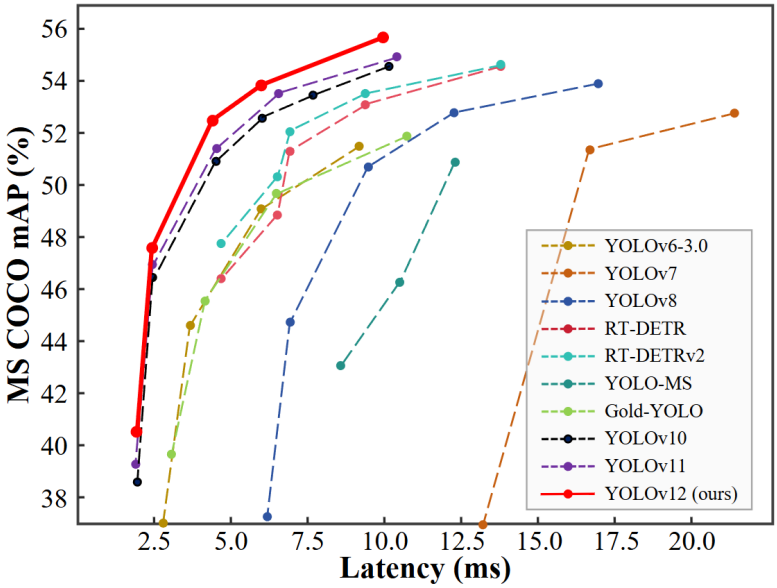


FIGURE 2.8 – Comparaison de YOLOv12 avec d'autres méthodes populaires en termes de compromis latence/précision [7].

4.2 Présentation de YOLOv12

YOLO (You Only Look Once) est l'un des modèles de détection d'objets les plus utilisés dans l'industrie, développé par Joseph Redmon et Ali Farhadi à l'Université de Washington. Lancé en 2015, YOLO a gagné en popularité grâce à sa rapidité et à sa précision (surtout en temps réel)[7]. YOLO compte plusieurs versions, de la version 2 au dernier modèle, la version 12, apparue en février 2025. YOLOv8 est le plus récent à l'heure actuelle, mais YOLOv5 reste le plus utilisé en pratique[113].

Architecture YOLOv12

YOLOv12 est basé sur une architecture centrée sur les mécanismes d'attention, en particulier **le self-attention optimisé**, implémenté à travers des modules tels que **FlashAttention** et **Area Attention**, ce qui représente une avancée majeure par rapport aux versions précédentes. Cette approche s'éloigne des méthodes traditionnelles basées sur les réseaux neuronaux convolutifs (CNNs), comme dans les versions précédentes de YOLO. Le modèle se caractérise par le maintien de la vitesse d'inférence en temps réel, une propriété essentielle pour de nombreuses applications pratiques. En apportant des améliorations systématiques aux mécanismes d'attention et à la conception globale du réseau, YOLOv12 atteint une grande précision dans la détection des objets tout en conservant sa rapidité selon [7].

La figure (2.9) illustre l'architecture générale de YOLOv12, composée de trois parties principales : le **Backbone** pour extraire les caractéristiques de l'image, le **Neck** pour combiner les informations à différentes échelles, et le **Head** qui prédit les *boxes* (les zones rectangulaires autour des objets détectés) et les classes des objets détectés (le type d'objet, par exemple insecte A, insecte B).

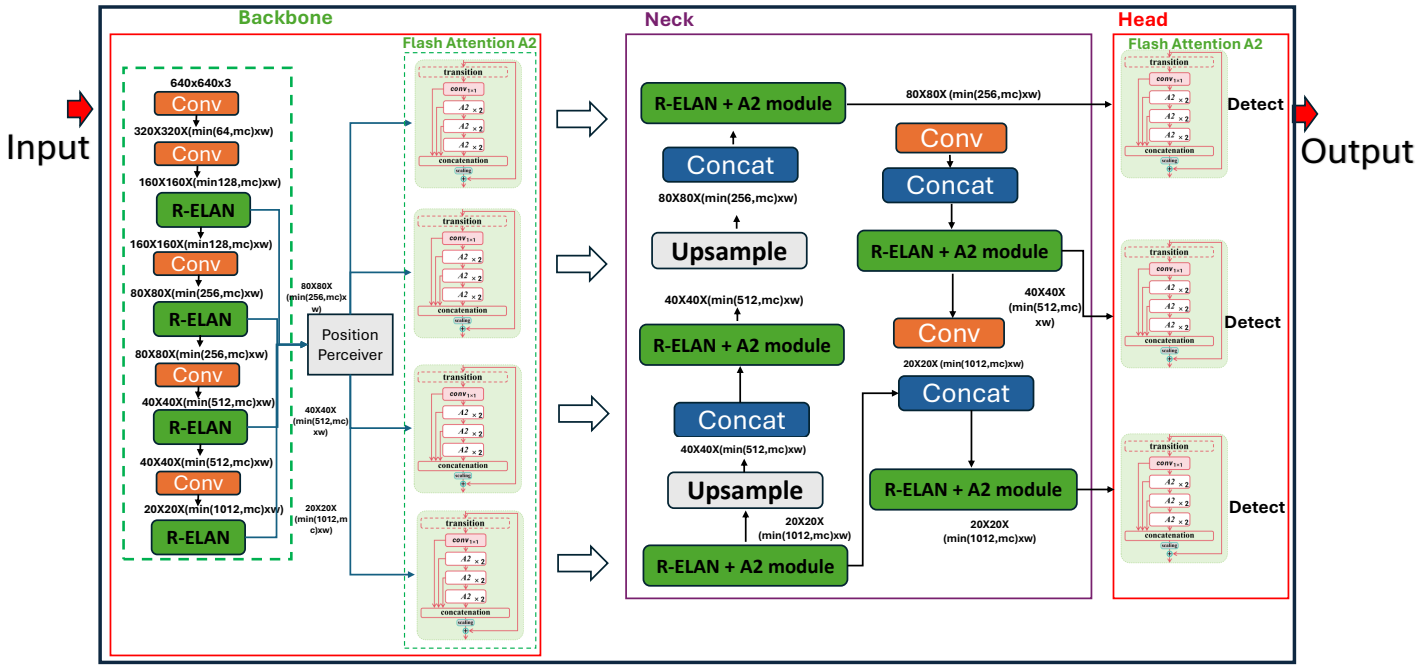


FIGURE 2.9 – Architecture du modèle de détection YOLOv12 [114].

Le **Backbone** est chargé d'extraire les caractéristiques de l'image. Il utilise une série de couches convolutives et des blocs appelés **R-ELAN (Residual Efficient Layer Aggregation Networks)** [7] pour capter des informations visuelles à différents niveaux de profondeur. Cette partie produit plusieurs cartes de caractéristiques, chacune représentant un niveau de détail différent [7]. Il faut noter que les **cartes de caractéristiques** (feature maps en anglais) sont des **représentations intermédiaires** générées par un réseau de neurones à partir de l'image d'entrée.

Le **Neck** c'est la **partie intermédiaire** de l'architecture qui sert à combiner ces cartes de caractéristiques. Il utilise notamment des opérations de concaténation et d'upsampling (augmentation de la résolution) pour fusionner les informations provenant de différentes résolutions. Un module d'attention rapide appelé **FlashAttention** [115] est utilisé ici pour permettre au modèle de mieux se concentrer sur les zones importantes, tout en maintenant une vitesse d'exécution élevée [7].

La **Head** prend ces cartes combinées et effectue la prédiction finale : elle génère des **boxes** autour des objets détectés et assigne une **classe** à chaque objet. Ce processus est réalisé à plusieurs échelles (petite, moyenne, grande) afin de pouvoir détecter des objets de tailles variées [7].

4.3 Préparation des données pour la détection

4.3.1 Transformation des annotations IP102 au format YOLO

Dans le but d'entraîner le modèle YOLOv12 à détecter la **présence d'insectes nuisibles**, les annotations d'origine du dataset IP102, fournies au format **XML (Pascal VOC)**, ont été converties vers le **format texte utilisé par YOLO**. Cette étape est essentielle pour adapter les données à la structure attendue par le détecteur.

Chaque fichier XML contient une ou plusieurs *bounding boxes* définies par les coordonnées de l'insecte dans l'image. Ces coordonnées $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$ ont été transformées en format YOLO sous la forme :

```
class_id   x_center   y_center   width   height
```

Où toutes les valeurs sont normalisées (comprises entre 0 et 1), et représentent respectivement :

- Le centre de la boîte en abscisse et en ordonnée,
- Sa largeur et sa hauteur, toutes relatives à la taille de l'image.

Dans le cadre de ce travail, toutes les annotations ont été affectées à une seule et même classe générique (`class_id = 0`), car l'objectif est uniquement de détecter la présence d'un insecte nuisible, indépendamment de son espèce. Ce choix de simplification permet de se concentrer sur la tâche de localisation, tandis que l'identification fine des espèces est traitée ultérieurement dans un module de classification dédié.

4.3.2 Exemple de conversion vers le format YOLO

Considérons l'extrait suivant d'un fichier XML (voir section 2.2). Cette boîte englobante correspond à un objet allant de la position $(10, 54)$ à $(189, 123)$ sur une image de taille 199×163 pixels.

Le format YOLO exige que toutes les valeurs (sauf `class_id`) soient **normalisées** entre 0 et 1 par rapport à la taille de l'image :

$$x_{\text{center}} = \frac{x_{\min} + x_{\max}}{2 \times \text{width}} = \frac{10 + 189}{2 \times 199} \approx 0.500$$

$$y_{\text{center}} = \frac{y_{\min} + y_{\max}}{2 \times \text{height}} = \frac{54 + 123}{2 \times 163} \approx 0.545$$

$$\text{width} = \frac{x_{\max} - x_{\min}}{\text{width}} = \frac{189 - 10}{199} \approx 0.899$$

$$\text{height} = \frac{y_{\max} - y_{\min}}{\text{height}} = \frac{123 - 54}{163} \approx 0.423$$

Cela signifie qu'un **objet de classe 0** est centré en $(0.500, 0.545)$, avec une largeur relative de **0.899** et une hauteur de **0.423**. Le format YOLOv12 pour cette image va être donc :

```
0   0.500   0.545   0.899   0.423
```

4.3.3 Stratégies d'augmentation des données

L'analyse préliminaire du *dataset* **IP102** a mis en évidence trois défis majeurs pour la tâche de détection d'insectes nuisibles :

1. Une **diversité biologique marquée**, notamment liée aux différents stades de développement des insectes (œuf, larve, adulte), impliquant une variabilité de formes et d'apparences.
2. Une **variabilité significative du nombre d'objets annotés par image (voir figure 2)**, certaines scènes présentant une densité élevée d'insectes.
3. Une **grande diversité des tailles des insectes** présents dans les images, allant d'individus très petits à d'autres beaucoup plus grands, comme l'illustre la Figure 5 représentant la distribution normalisée des hauteurs et largeurs des objets annotés.

Pour faire face à ces défis et rendre le modèle **YOLOv12** plus robuste et capable de mieux généraliser, plusieurs techniques d'augmentation de données ont été utilisées pendant l'entraînement. Ces transformations transformations ont pour but d'augmenter la diversité des exemples, de réduire le risque de surapprentissage et d'améliorer les performances du modèle sur de nouvelles images.

Les transformations spécifiques mises en oeuvre sont les suivantes :

- **Inversion horizontale aléatoire (environ 50 % des cas)** : Cette technique simule différentes orientations des insectes dans leur environnement. Elle permet d'éviter que le modèle n'apprenne une configuration spatiale spécifique, le rendant ainsi plus invariant à l'orientation des objets.

FIGURE 2.10 – une figure d'un insecte avant et après l'application de l'inversion horizontale.

- **Changement d'échelle aléatoire (environ 50 % de variation)** : Cette transformation ajuste dynamiquement la taille des objets dans les images. Elle est cruciale pour que le modèle apprenne à détecter efficacement des insectes de tailles très variées, répondant directement au défi de la diversité des tailles identifié dans le *dataset* IP102.
- **Mosaic (appliquée dans 100 % des cas en début d'entraînement)** : Cette technique consiste à fusionner aléatoirement quatre images d'entraînement (et leurs annotations respectives) en une seule. Elle génère ainsi des scènes complexes contenant des objets à de

multiples échelles et des densités variables, renforçant directement la capacité du modèle à gérer la variabilité du nombre d'objets et la diversité de leurs tailles, tout en l'exposant à des contextes d'arrière-plan plus riches.

- **AutoAugment (sous sa variante RandAugment)** : Une série de politiques d'augmentation optimales est appliquée de manière aléatoire. RandAugment sélectionne et applique dynamiquement diverses transformations (telles que des modifications de luminosité, de contraste, de netteté, de couleur, ainsi que des déformations géométriques mineures) pour augmenter considérablement la diversité visuelle du jeu de données. Ceci est essentiel pour s'adapter à la variabilité d'apparence des insectes due à leurs stades de développement, aux conditions d'éclairage changeantes ou à la diversité des arrière-plans.

FIGURE 2.11 – un insecte avant et après l'application de Mosaic.

Le tableau suivant résume les paramètres spécifiques appliqués pour chaque technique d'augmentation mentionnée précédemment :

Paramètre d'augmentation	**Valeur**
Inversion horizontale (flip left-right fliplr)	0.5
Scale augmentation	0.5
Mosaic augmentation	1.0
RandAugment	Activé

TABLE 2.2 – Paramètres des techniques d'augmentation utilisées.

5 Configuration, Entraînement et Optimisation du Modèle YOLOv12n

Cette section présente de manière détaillée le cadre expérimental mis en place pour l'entraînement du modèle YOLOv12n sur les données de détection du *dataset* IP102. Elle est structurée en sous-parties couvrant la configuration matérielle, la structure du dataset, les hyperparamètres d'apprentissage, les fonctions de perte, ainsi que les mécanismes d'évaluation et d'arrêt anticipé. L'objectif est de garantir la reproductibilité des résultats présentés dans le chapitre 3.

Le modèle utilisé est **YOLOv12n**, une version *nano* du modèle YOLOv12, conçue pour les déploiements en environnement à ressources limitées. Bien que plus compacte que les versions `s`, `m` ou `l`, YOLOv12n maintient un bon compromis entre légèreté et précision, ce qui le rend idéal pour des scénarios en temps réel dans le domaine agricole [7]. Le modèle a été initialement pré-entraîné sur `COCO128`, puis réentraîné sur les données de détection IP102.

5.1 Configuration Matérielle

L'entraînement a été réalisé sur une plateforme **Google Colab Pro+**, bénéficiant des ressources matérielles suivantes :

- **GPU** : NVIDIA A100 (40 Go VRAM), architecture Ampere
- **RAM** : 83,5 Go de mémoire vive
- **CPU** : 2 cœurs
- **Stockage** : 235,7 Go d'espace SSD

5.2 Structure et Prétraitement du Dataset

Pour la tâche de **détection**, un **sous-ensemble de 18,983 images annotées avec des boîtes englobantes** issues du dataset IP102 a été utilisé. Ces images couvrent **102 classes d'insectes nuisibles**, conformément à la taxonomie hiérarchique définie dans le jeu de données original.

Les images ont été redimensionnées à une taille standard de **640×640 pixels**, assurant une compatibilité optimale avec les contraintes d'entrée du modèle YOLOv12n.

La répartition du corpus s'est faite en deux étapes :

- **Train/Validation** : **15,178 images** (soit 80% du total) ont été sélectionnées pour les phases d'apprentissage et d'ajustement des hyperparamètres. Ce lot a ensuite été divisé comme suit :
  - **Entraînement** : **13,660 images** (90% de la partition Train/Val)
  - **Validation** : **1,518 images** (10% de la partition Train/Val)
- **Test** : **3 798 images** (20%) ont été conservées pour l'évaluation finale des performances du modèle.

5.3 Fonction de perte (Loss Function)

La fonction de perte utilisée par YOLOv12 est la même que YOLOv8 [8]. Elle est une combinaison pondérée de plusieurs composantes, adaptées à la tâche de détection d'objets. Elle peut être exprimée de manière générale comme suit :

$$\mathcal{L}(\theta) = \frac{\lambda_{box}}{N_{pos}}\mathcal{L}_{box}(\theta) + \frac{\lambda_{cls}}{N_{pos}}\mathcal{L}_{cls}(\theta) + \frac{\lambda_{dfl}}{N_{pos}}\mathcal{L}_{dfl}(\theta) \tag{2.1}$$

où :

- $\mathcal{L}_{box}$ (*box loss*) représente la perte de localisation des boîtes. C'est une perte généralement basée sur la CIoU [2],
- $\mathcal{L}_{cls}$ est la perte de classification multi-label (cross-entropy),
- $\mathcal{L}_{dfl}$ est la *Distribution Focal Loss* (DFL), utilisée pour améliorer la précision des bords de boîte,
- N_{pos} est le nombre total de prédictions positives,
- λ_{box}, λ_{cls}, λ_{dfl} sont des hyperparamètres de pondération. Dans notre configuration, **ils ont été fixés respectivement à 7.5, 0 et 1.5**, traduisant une priorité donnée à la précision de localisation des objets, tandis que la classification inter-classes est désactivée (**absence de tâche de classification**), et que la Distribution Focal Loss contribue modérément à l'affinage des prédictions.

Cette fonction permet de concilier précision de localisation, classification robuste et généralisation efficace du modèle.

5.4 Paramètres d'Entraînement et d'Optimisation

L'entraînement a été effectué pendant **100 époques**, en utilisant une configuration optimisée pour les contraintes matérielles tout en garantissant une bonne convergence :

- **Taille de lot (*batch size*)** : fixée à **128**, elle exploite efficacement les 40 Go de mémoire GPU disponible, tout en assurant une stabilité dans la mise à jour des gradients.
- **Algorithme d'optimisation (*Optimizer*)** : Stochastic Gradient Descent (SGD) avec décroissance de poids (*weight decay*), largement utilisé pour les tâches de détection d'objets en raison de sa stabilité et de sa capacité à mieux généraliser sur des jeux de données complexes. Formellement, la mise à jour des poids w_t à l'étape t est donnée par :

$$w_t = w_{t-1} - \eta \cdot (\nabla \mathcal{L}(w_{t-1}) + \lambda \cdot w_{t-1}) \tag{2.2}$$

  où η est le taux d'apprentissage, $\nabla \mathcal{L}(w_{t-1})$ le gradient de la fonction de perte, et λ le coefficient de décroissance de poids (*Weight Decay*).
- **Taux d'apprentissage (*Learning Rate*)** : **0.01**, valeur classique pour les architectures YOLO, avec une phase de **warm-up** progressive sur les **3 premières époques** afin de stabiliser les gradients.
- **Décroissance de poids (*Weight Decay*)** : **0.0005**, ajoutée pour limiter le surapprentissage en contrôlant la croissance des poids du réseau.

2. La **Complete Intersection over Union (CIoU)** [6] est une métrique qui intègre trois composantes : l'IoU, la distance entre les centres des boîtes et la différence de ratio d'aspect. Elle offre une convergence plus rapide et des prédictions de boîtes plus précises que l'IoU

6 Partie 2 : Module de reconnaissance des insectes avec FastViT

Cette seconde partie est consacrée au module de reconnaissance, qui constitue le coeur de notre système d'identification d'espèces. Son objectif est de classifier avec précision l'espèce d'un insecte à partir d'une image, qui aurait été préalablement localisée par le module de détection. Pour y parvenir, nous avons opté pour le modèle FastViT-SA12 , une architecture de vision hybride reconnue pour son excellent compromis entre précision, rapidité et légèreté, ce qui la rend idéale pour une application en temps réel. Ce module est spécifiquement conçu et entraîné sur le dataset IP102 pour surmonter ses défis inhérents, tels que le déséquilibre important entre les classes et la grande diversité visuelle des insectes à différents stades de leur vie.

Le schéma de la Figure 2.12 illustre l'ensemble du pipeline que nous avons mis en place pour l'entraînement et l'évaluation de ce module. Le processus débute avec le dataset IP102, qui est divisé en trois ensembles distincts : entraînement (TrainSet), validation (ValidationSet) et test (TestSet). La phase d'entraînement est un processus itératif : pour chaque lot de données, les images du TrainSet subissent un prétraitement standard (redimensionnement, normalisation) suivi d'une série d'augmentations de données dynamiques, incluant des techniques avancées comme Mixup et Cutmix, pour enrichir la diversité des exemples. Le modèle FastViT-SA12, initialisé avec des poids pré-entraînés, est ensuite mis à jour sur ces données. Parallèlement, une phase de validation est menée à la fin de chaque époque : un modèle lissé, obtenu par Moyenne Mobile Exponentielle (EMA), est évalué sur le ValidationSet pour sauvegarder la meilleure version du modèle (Best Model). Une fois l'entraînement terminé, ce meilleur modèle est utilisé pour une évaluation finale sur le TestSet, garantissant une mesure impartiale de la performance.

Dans ce qui suit, nous allons détailler chaque composant de ce module de reconnaissance. Nous commencerons par justifier le choix de l'architecture **FastViT** et de sa variante spécifique, le **FastViT-SA12**. Ensuite, nous présenterons en détail son **architecture interne**, puis nous aborderons les stratégies de **prétraitement** et d'**augmentation des données** mises en œuvre pour adapter le modèle aux spécificités du dataset IP102. Enfin, nous conclurons avec la description de la **méthodologie d'entraînement**, incluant les hyperparamètres, la fonction de perte et les techniques d'optimisation utilisées.

6.1 Introduction et motivation du choix de FastViT

Dans cette section, nous abordons la tâche de classification des espèces d'insectes nuisibles à partir des images du dataset IP102. Le modèle retenu pour cette étape est **FastViT-SA12**, une variante efficace de la famille **FastViT** [9], récemment proposée par Apple.

FastViT est une architecture hybride de vision introduite par Apple, qui combine les avantages des réseaux convolutifs (CNN) et des Transformers. Elle se distingue par un excellent compromis entre précision et rapidité, en particulier sur des dispositifs embarqués.

L'étape de détection permet déjà de localiser efficacement les insectes sur les images. La classification intervient ensuite pour identifier précisément l'espèce présente dans chaque région détectée. Ce processus soulève plusieurs défis importants :

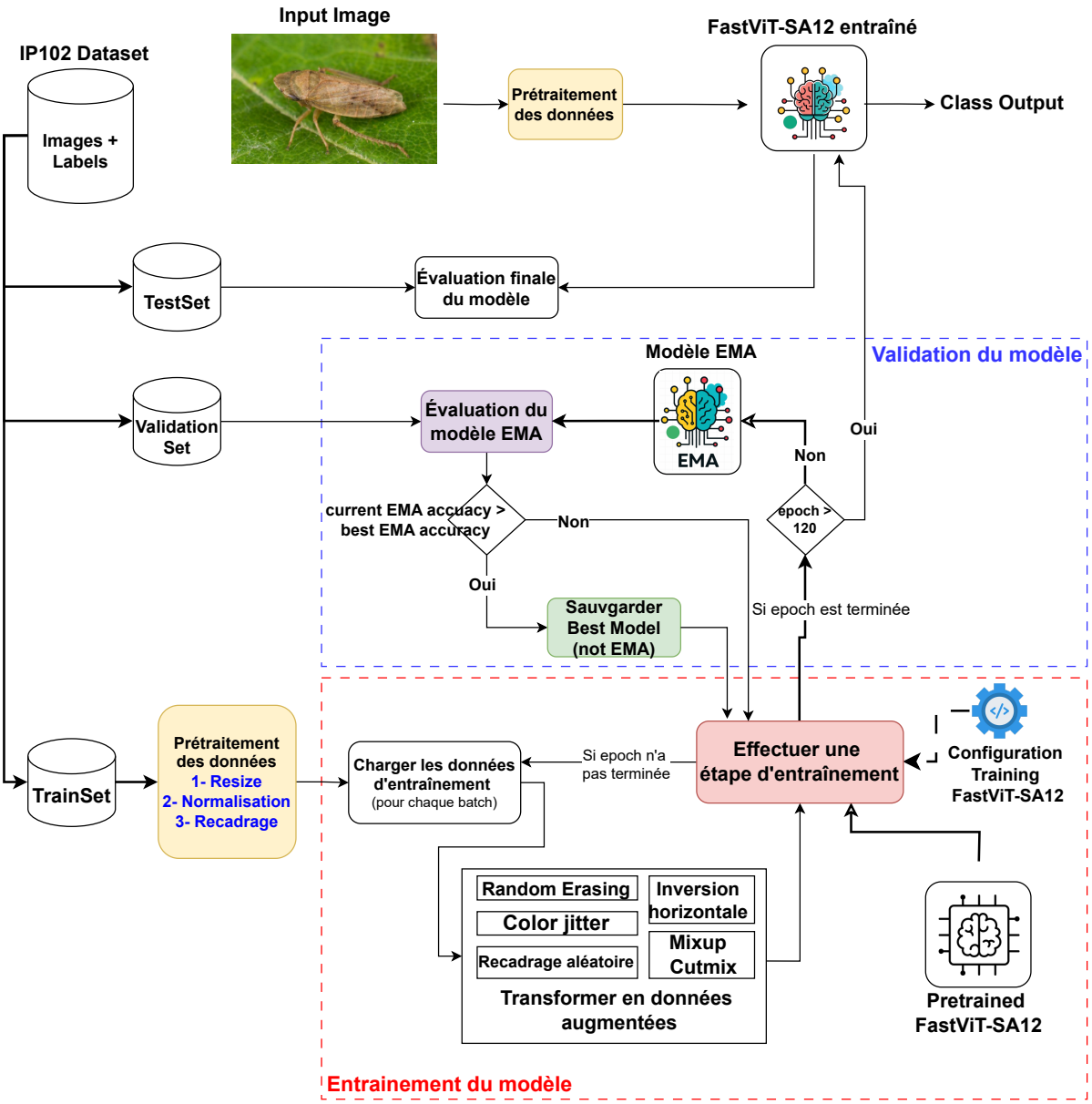


FIGURE 2.12 – Schéma proposé de reconnaissance des insectes nuisibles avec FastViT-SA12.

- une **grande diversité biologique** au sein du dataset IP102 (stades de vie variés, apparences multiples) ;
- un **déséquilibre marqué entre les classes**, certaines espèces étant très peu représentées ;
- la nécessité d'un **traitement en temps réel**, indispensable pour des déploiements embarqués (sur mobile ou drone).

Face à ces contraintes, le choix du modèle **FastViT-SA12** s'est imposé grâce à ses qualités techniques remarquables :

- un **temps d'inférence extrêmement court**, parfaitement adapté aux contraintes de l'inférence embarquée ;
- une **précision Top-1 élevée**, équivalente voire supérieure à celle de modèles plus lourds comme `ConvNeXt` ou `DeiT` [9] ;

- une **latence inférieure à 3 ms** sur GPU et mobile pour la version SA12, tout en maintenant une précision supérieure à 81,9 % sur `ImageNet` [9].

Ces performances sont clairement visibles sur la Figure 2.13, où **FastViT** se distingue comme la meilleure option en termes de compromis précision/latence. En rouge, **FastViT** domine nettement en termes de latence mobile, tout en maintenant une précision élevée. En comparaison, les modèles comme `ConvNeXt`, `EfficientNet`, ou `DeiT` affichent des temps d'inférence plus élevés pour des performances similaires.

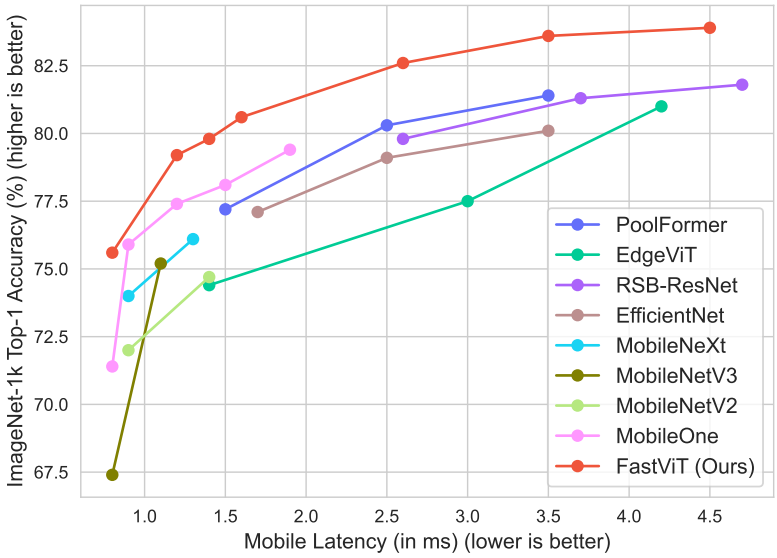


FIGURE 2.13 – Courbes d'accuracy en fonction de la latence mobile pour les méthodes récentes. Les modèles sont évalués sur un iPhone 12 Pro selon [9]. Seuls ceux atteignant une accuracy Top-1 supérieure à 79 % sont représentés. **FastViT** présente le meilleur compromis accuracy/latence dans un contexte mobile.

6.2 Choix du version de fastViT

Le modèle FastViT est une architecture de vision hybride qui se décline en plusieurs variantes (tableau 2.3), conçues pour offrir un large éventail de compromis entre performance et efficacité. La nomenclature de ces modèles permet de comprendre leur structure : les versions préfixées par « **T** » **(Tiny)** utilisent un ratio d'expansion MLP (Multi-Layer Perceptron) réduit , tandis que les préfixes « **S** » et « **M** » désignent respectivement des modèles avec des dimensions d'enchâssement (embedding) plus petites ou plus grandes. Le préfixe « **SA** » (Self-Attention) est particulièrement important, car il indique que le modèle intègre des couches d'auto-attention dans ses derniers étages pour améliorer la précision. Enfin, le nombre qui suit (par exemple, 12 dans SA12) représente le nombre total de blocs FastViT dans le réseau.

La famille des modèles **FastViT** se décline en plusieurs versions, différenciées par leur taille (nombre de paramètres), leur rapidité (latence sur GPU et mobile) et leur précision. Les versions les plus légères, telles que *FastViT-T8* et *T12*, offrent une très faible latence au prix d'une précision plus modeste. Les variantes intermédiaires comme *FastViT-S12* et *SA12* atteignent un bon compromis entre efficacité et performance. Enfin, les modèles plus volumineux tels que

SA36 ou *MA36* visent une précision maximale, mais au détriment de la légèreté et de la vitesse d'exécution.

Dans la tâche de classification des insectes nuisibles, nous avons choisi **FastViT-SA12**. Cette sélection a été basée sur plusieurs critères techniques et contextuels. Comme le montre le tableau 2.3, le modèle FastViT-SA12 présente un équilibre optimal entre la précision, la complexité et l'efficacité. Le dispositif présente une latence minimale (1,6 milliseconde) tout en maintenant un niveau élevé de performance, ce qui en fait un candidat optimal pour les systèmes légers ou les applications mobiles en temps réel. En termes de complexité comparable, ce modèle est plus efficace que beaucoup d'autres modèles, en particulier en ce qui concerne la vitesse d'exécution.

Modèle	Taille Image	Param (M)	Latence GPU (ms)	Latence Mobile (ms)	Top-1 Acc. (%)
FastViT-T8	256	3.6	1.7	0.8	76.7
FastViT-T12	256	6.8	2.1	1.2	80.3
FastViT-S12	256	8.8	2.2	1.4	80.5
FastViT-SA12	**256**	**20.3**	**2.5**	**1.6**	**81.9**
FastViT-SA24	256	26.0	2.6	2.6	83.4
FastViT-SA36	384	52.2	5.2	4.5	84.6
FastViT-MA36	256	42.7	6.7	4.5	84.6

TABLE 2.3 – Comparaison des modèles FastViT en termes de taille d'image, nombre de paramètres, latence (GPU et mobile) et précision Top-1 [9].

Ce choix est validé par un test expérimental (voir section5.4 du chapitre 3)

6.3 Architecture FastViT

La figure 2.14 illustre la structure globale de FastViT, composée de plusieurs blocs successifs :

Le Stem [3] constitue la première étape du réseau. Il applique une série de convolutions légères pour transformer l'image d'entrée en une représentation compacte [4]. Ce bloc utilise une structure enrichie pendant l'entraînement, qui est ensuite simplifiée pour l'inférence, permettant ainsi de conserver de bonnes performances sans alourdir le modèle final [9].

Les Stages 1 à 3 partagent la même structure basée sur des blocs appelés **RepMixer**. Contrairement aux Transformers classiques, ces blocs n'utilisent pas de *skip connections* (connexions résiduelles permettant de transmettre directement l'information d'une couche à une autre [47]), ce qui permet de réduire le coût mémoire et d'accélérer l'inférence. Chaque bloc combine une normalisation (BatchNorm) avec une convolution *depthwise* (convolution appliquée séparément sur chaque canal, réduisant fortement le nombre de calculs), le tout étant reparamétré en une seule opération pendant l'inférence [9].

3. Le terme *stem* désigne les couches initiales du réseau, responsables du traitement des données d'entrée, souvent des images.

4. Une représentation compacte est un ensemble de cartes de caractéristiques (feature maps) de plus petite taille que l'image d'origine, mais qui conservent les informations visuelles essentielles (textures, contours, couleurs, etc.) nécessaires à la suite du traitement.

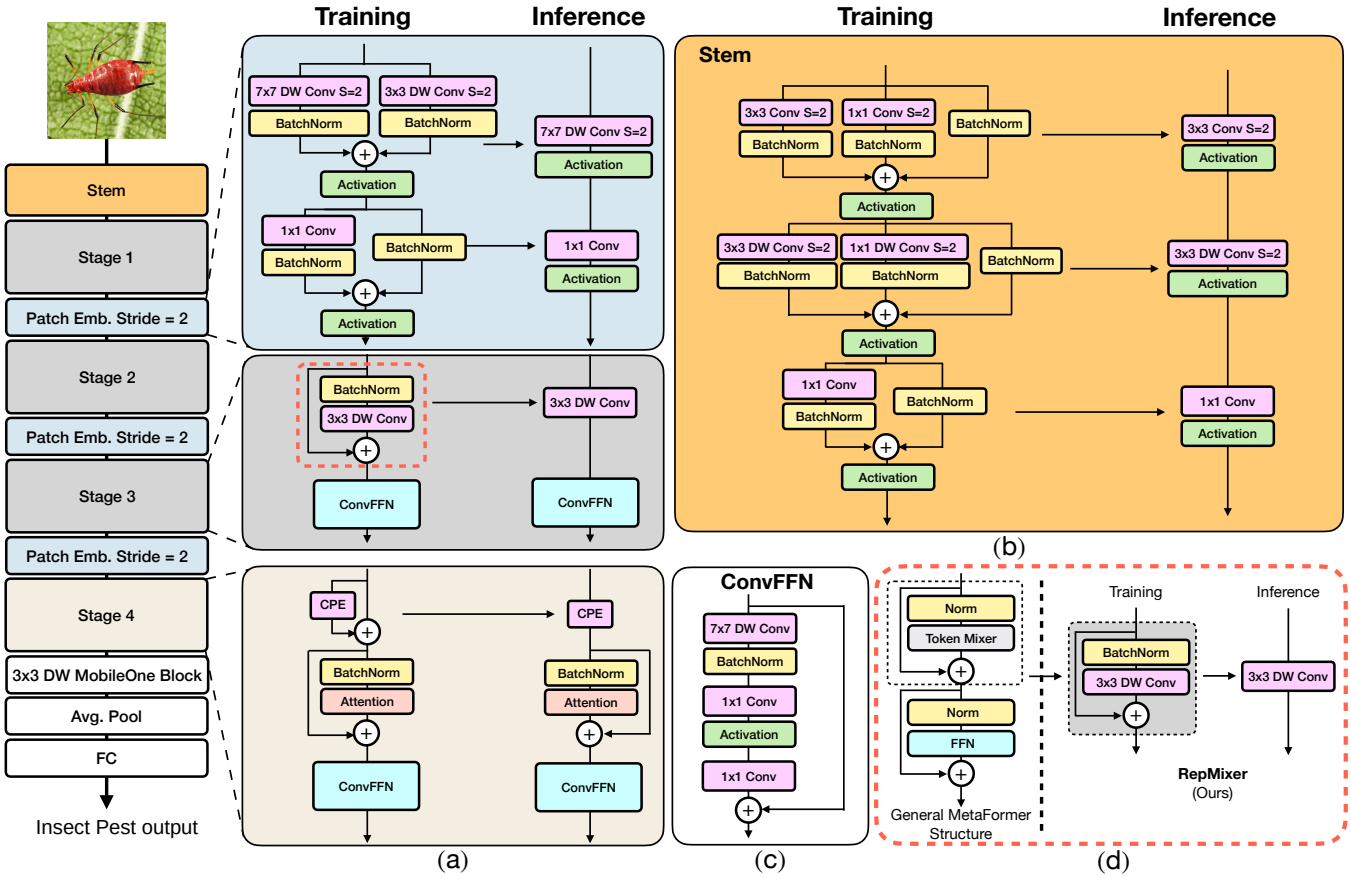


FIGURE 2.14 – Aperçu simplifié de l'architecture de FastViT. (a) Les stages 1, 2 et 3 ont une structure identique et utilisent RepMixer, tandis que le stage 4 utilise des couches de self-attention. (b) Structure du stem convolutionnel. (c) Bloc FFN basé sur des convolutions. (d) Bloc RepMixer avec reparamétrisation de connexion résiduelle pour l'inférence [9].

Le Stage 4 introduit des mécanismes d'**attention locale** pour améliorer l'*accuracy* globale, tout en restant plus léger que les couches de self-attention complètes. Il intègre également des encodages positionnels conditionnels (*Conditional Positional Encoding*, CPE[5]) [116] générés par des convolutions.

Les blocs FFN (Feed Forward Networks) de FastViT adoptent une variante inspirée de `ConvNeXt` [117], où des **convolutions à large noyau** (par exemple 7×7) sont utilisées à la place de couches linéaires. Cela améliore le champ de vision global du modèle et sa **robustesse face aux perturbations**.

Enfin, la tête de classification (*head*) applique un `average pooling` suivi d'un classifieur fully-connected pour produire la prédiction finale [9].

5. Le rôle du CPE est d'ajouter une information de position aux représentations spatiales, tout en l'adaptant dynamiquement au contenu local de l'image, ce qui permet au modèle de mieux capturer la structure spatiale sans recourir à des encodeurs de position explicites [116].

6.4 Prétraitement et augmentation des données

6.4.1 Prétraitement des images

Avant l'entraînement du modèle de classification FastViT-SA12, chaque image du dataset **IP102** est soumise à un ensemble de transformations standardisées :

- **Redimensionnement** à la taille $(3, 256, 256)$, afin d'uniformiser les dimensions des entrées du modèle.
- **Normalisation** des canaux RGB à l'aide des statistiques standards d'ImageNet (moyennes : $(0.485, 0.456, 0.406)$, écarts-types : $(0.229, 0.224, 0.225)$) pour accélérer la convergence du réseau.
- **Recadrage central** avec un facteur `crop_pct = 0.9`, ce qui permet de recentrer les objets pertinents (insectes) tout en éliminant les zones périphériques moins informatives.

6.4.2 Stratégies d'augmentation des données

L'augmentation de données joue un rôle crucial dans le traitement du dataset **IP102**, qui présente plusieurs défis majeurs. Ce jeu de données souffre notamment d'une **distribution en longue traîne** (*long-tail distribution*), où certaines espèces d'insectes sont largement représentées (classes dites *head*), tandis que d'autres ne disposent que de très peu d'exemples (classes *tail*). Une telle répartition déséquilibrée compromet l'apprentissage équilibré du modèle et le rend moins performant sur les classes rares. À cela s'ajoutent deux autres difficultés structurelles :

1. Une **forte diversité biologique**, avec plus de 100 espèces présentant des formes et textures parfois très similaires ;
2. Des insectes souvent **petits**, **partiellement visibles**, voire situés en **arrière-plan**, ce qui complique leur identification.

Pour renforcer la capacité de généralisation du classifieur **FastViT-SA12** face à ces contraintes, plusieurs techniques d'augmentation des données ont été mises en œuvre au cours de l'entraînement.

1. Mixup [118] (appliqué dans 50 % des cas avec un coefficient de mélange de 0,2)
Mixup consiste à combiner linéairement deux images ainsi que leurs étiquettes respectives, selon un coefficient de mélange aléatoire. Cette technique produit des images synthétiques intermédiaires, associées à des étiquettes interpolées entre les deux classes d'origine [118].

- Elle améliore la **robustesse du modèle** face aux variations intra-classes, en particulier pour les espèces avec peu d'exemples.
- Elle agit comme une forme de **régularisation**, en lissant les frontières de décision entre classes visuellement proches.
- Contrairement aux augmentations classiques (flips, crops), Mixup agit à la fois sur le niveau pixel et le label, renforçant l'effet d'enrichissement des données.

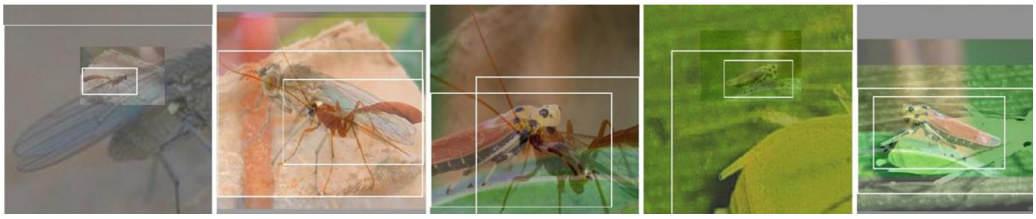


FIGURE 2.15 – Exemples d'images traitées à l'aide de l'algorithme Mixup

Par exemple, en combinant une image contenant une *mouche des fruits* (classe A) et une autre avec une *chenille du chou* (classe B), avec un ratio de 0,6 pour A et 0,4 pour B, on obtient une image floue composée des deux espèces, et une étiquette pondérée à 60 % pour la classe A et 40 % pour la classe B.

2. Cutmix [119] (appliqué à chaque image pendant l'entraînement) Cutmix remplace une zone rectangulaire d'une image par un patch provenant d'une autre image du dataset. L'étiquette finale est calculée proportionnellement à la surface occupée par chaque classe. Cette méthode permet de simuler des insectes partiellement visibles, améliore la capacité du modèle à apprendre à partir de contextes incomplets, et renforce la présence visuelle des classes rares, contribuant ainsi à réduire le déséquilibre du dataset [119].

Par exemple, si une image contient un *ver de maïs* (classe A) et qu'on y insère un patch d'une image contenant une *punaise verte* (classe B) couvrant 30 % de la surface, alors la nouvelle image aura une étiquette composée à 70 % de la classe A et à 30 % de la classe B.

3. Effacement aléatoire (Random Erasing) Cette méthode consiste à masquer une petite région rectangulaire placée de manière aléatoire sur l'image d'entrée. Elle est appliquée dans environ **un quart des cas (25%)** durant l'entraînement. Son objectif est de forcer le modèle à s'adapter à l'absence d'informations localisées, en s'appuyant davantage sur les indices contextuels environnants. Cette propriété est particulièrement utile pour traiter les cas où les insectes sont partiellement occultés ou déformés.

4. Inversion horizontale aléatoire (environ 50% des cas) Dans environ une image sur deux, l'image est retournée horizontalement. Cette transformation augmente la diversité spatiale des exemples d'entraînement sans altérer l'identité visuelle ou biologique des insectes. Elle constitue une méthode simple mais efficace pour enrichir le *dataset*, notamment dans le cas d'espèces dont l'orientation n'est pas déterminante.

5. Jitter de couleur Le jitter de couleur consiste à **modifier aléatoirement la luminosité, le contraste, la saturation et la teinte de l'image**, a**vec une intensité modérée fixée à 0,4** [120]. Cette transformation permet de simuler la **variabilité des conditions d'éclairage** typiques des environnements agricoles, influencées par la météo, l'heure de la journée ou le type de capteur utilisé. Le dataset **IP102** étant composé d'images prises dans des conditions très hétérogènes (pleine lumière, ombre partielle, feuillages, sols clairs ou sombres), cette variabilité visuelle ne reflète pas directement les différences entre espèces. Le jitter de couleur permet

donc au modèle de se concentrer sur les caractéristiques visuelles pertinentes, **améliorant sa robustesse et sa capacité de généralisation** face aux changements d'apparence non liés à l'insecte lui-même.

FIGURE 2.16 – Exemples de transformations visuelles générées par l'algorithme de Jitter de couleur (variation de luminosité, contraste, saturation et teinte) [120].

6. Recadrage aléatoire avec variation d'échelle et de ratio (Random Resized Crop) Cette opération sélectionne un cadrage aléatoire dans l'image, représentant **entre** 70 % et 100 % de sa surface totale, et avec un ratio largeur/hauteur compris **entre 0,75 et 1,33.** Le cadrage est ensuite redimensionné à la taille standard d'entrée du modèle. Ce procédé permet d'exposer le réseau à des représentations variées d'un même objet, renforçant sa capacité à reconnaître des insectes de tailles et de proportions diverses.

Le tableau suivant présente un résumé des principales techniques d'augmentation appliquées pendant l'entraînement du classifieur :

Technique	Fréquence	Paramètres principaux
Mixup [118]	50 %	$\alpha = 0{,}2$
Cutmix [119]	100 %	
Effacement aléatoire	25 %	
Flip horizontal	50 %	
Jitter couleur [120]	100 %	Facteur = 0,4
Crop aléatoire	100 %	Échelle $[0{,}7 - 1{,}0]$; Ratio $[0{,}75 - 1{,}33]$

TABLE 2.4 – Résumé synthétique des techniques d'augmentation utilisées dans le processus d'entraînement du modèle.

6.5 Entraînement et Optimisation du Modèle FastViT-SA12

Cette section détaille la méthodologie d'entraînement et d'optimisation pour le classifieur **FastViT-SA12**. Afin de tirer parti du transfert d'apprentissage (*transfer learning*), notre approche s'appuie sur un modèle **pré-entraîné sur le vaste jeu de données ImageNet-1k**. Ce modèle a ensuite été spécialisé pour notre tâche via un processus de ré-entraînement fin (*fine-tuning*) sur le dataset IP102. L'objectif est d'adapter ce modèle puissant pour surmonter

les défis spécifiques de notre problématique, notamment le déséquilibre des classes et la grande diversité intra-classe.

La configuration matérielle utilisée pour cette phase est **identique à celle employée pour l'entraînement du modèle YOLOv12** : Colab Pro+ avec GPU A100 VRAM 40Go, RAM 83.5Go et Stockage 235.7Go (voir la section 5.1), et la répartition des données de classification suit la division présentée à la section 2.3. Les sous-sections suivantes justifieront en détail les choix des hyperparamètres, les techniques de régularisation et la stratégie d'optimisation retenus.

6.5.1 Stratégie d'Optimisation

L'entraînement a été mené sur **120 époques**. Le choix des paramètres d'optimisation est crucial pour garantir une convergence stable vers un minimum optimal de la fonction de perte, tout en évitant le surapprentissage.

- **Optimiseur : AdamW.** L'optimiseur **AdamW** [121] a été retenu. Contrairement à l'optimiseur Adam classique, AdamW découple la décroissance de poids (*weight decay*) de la mise à jour des gradients. Cette particularité en fait une méthode de régularisation plus efficace et stable, particulièrement adaptée aux modèles complexes comme les Transformers, prévenant ainsi le surapprentissage.

  La règle de mise à jour pour un poids θ est la suivante :

  $$\theta_t \leftarrow \theta_{t-1} - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_{t-1} \right)$$

  Où chaque symbole représente :

  - θ_t : Le poids mis à jour à l'étape t.
  - θ_{t-1} : Le poids à l'étape précédente.
  - η : Le taux d'apprentissage (*learning rate*).
  - $\hat{m}_t$: L'estimation corrigée de la moyenne mobile des gradients.
  - $\hat{v}_t$: L'estimation corrigée de la moyenne mobile des carrés des gradients.
  - λ : Le coefficient de décroissance de poids (*Weight Decay*).
  - ϵ : Une petite constante pour la stabilité numérique.
- **Taux d'apprentissage et Ordonnancement.** Nous avons utilisé un taux d'apprentissage initial de 5×10^{-4}, une valeur de départ éprouvée pour le *fine-tuning* de modèles de vision pré-entraînés. Pour réguler ce taux, une stratégie de **décroissance cosinusoïdale (*Cosine Annealing*)** a été mise en place. Cette méthode diminue progressivement le taux d'apprentissage en suivant une courbe sinusoïdale, ce qui permet au modèle d'explorer l'espace des solutions de manière plus large au début, puis d'affiner sa convergence vers un minimum local de manière plus stable en fin d'entraînement. **Une phase d'échauffement (*warm-up*) de 3 époques** a été intégrée pour stabiliser l'entraînement initial.
- **Taille de lot (Batch Size).** La taille de lot a été fixée à **512**. Cette grande taille, permise par les ressources matérielles disponibles (GPU A100), offre deux avantages majeurs : elle

fournit une estimation plus stable du gradient à chaque itération et accélère le temps de calcul global pour chaque époque.

6.5.2 Gestion du Déséquilibre des Données : Le `WeightedRandomSampler`

Le dataset IP102 souffre d'un **fort déséquilibre** : certaines classes d'insectes contiennent des milliers d'images, tandis que d'autres, les classes minoritaires, n'en ont que quelques dizaines. Un entraînement naïf biaiserait le modèle en faveur des classes majoritaires. Pour contrer ce problème, un **`WeightedRandomSampler` (échantillonneur aléatoire pondéré)** a été utilisé.

Pour résumer, le `WeightedRandomSampler` est une technique qui corrige le déséquilibre des classes dans un *dataset*. Il fonctionne en **attribuant un poids plus élevé aux images des classes rares et un poids plus faible à celles des classes fréquentes** [122].

Lors de la création des lots d'entraînement, cet échantillonneur pioche les images en fonction de ces poids, assurant ainsi que le modèle soit exposé de manière équilibrée à toutes les classes. Cela force le modèle à accorder une attention égale aux classes minoritaires et majoritaires, réduisant ainsi le biais d'apprentissage [122].

La formule pour calculer le poids w d'une image appartenant à une classe spécifique c est simplement l'inverse du nombre d'images dans cette classe :

$$w_{\text{image}} = \frac{1}{N_c}$$

Où :

- w_{image} est le poids assigné à l'image.
- N_c est le nombre total d'images dans la classe c à laquelle l'image appartient.

6.5.3 Fonction de Perte Adaptée à l'Augmentation : `SoftTargetCrossEntropy`

Les techniques d'augmentation de données retenues, **Mixup** et **Cutmix**, créent des images synthétiques en mélangeant plusieurs exemples. **Le label associé à une image augmentée n'est donc plus une classe unique (label "hard"), mais une distribution de probabilités entre plusieurs classes (label "soft")**. La fonction de perte **`SoftTargetCrossEntropy`** est spécifiquement conçue pour ce scénario. Contrairement à la cross-entropie standard, qui attend une seule bonne réponse (ex : `[0, 0, 1, 0]`), elle compare la prédiction du modèle à un vecteur de probabilités cibles.

La fonction `SoftTargetCrossEntropy` utilise ce $\text{Label}_{\text{mix}}$ comme vérité terrain via la formule :

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \cdot \log(p_i)$$

où y_i est la probabilité cible (soft) pour la classe i et p_i est la probabilité prédite par le modèle.

6.5.4 Stabilisation de l'Évaluation : `Exponential Moving Average (EMA)`

Une moyenne mobile exponentielle (*Exponential Moving Average*, **EMA**) des poids du modèle est maintenue durant l'entraînement [123]. Elle est définie par la relation suivante :

$$\theta_t^{\text{EMA}} = \alpha \cdot \theta_{t-1}^{\text{EMA}} + (1 - \alpha) \cdot \theta_t$$

où :

- θ_t^{EMA} : poids EMA à l'étape t,
- θ_t : poids actuels du modèle,
- $\alpha \in [0, 1)$: facteur de décroissance contrôlant le lissage (**fixé par** 0.9995).

Contrairement aux poids du modèle principal, ceux de l'EMA ne participent **ni à la rétropropagation ni à l'optimisation**. Ils servent uniquement à l'**évaluation sur l'ensemble de validation**, offrant ainsi des performances plus stables et une meilleure généralisation [123].

La valeur EMA évolue lentement, atténuant les fluctuations rapides des poids du modèle principal.

6.5.5 Déroulement de l'Entraînement

Le processus d'entraînement pour chaque époque peut être résumé comme une séquence d'opérations rigoureusement orchestrées :

1. **Création des lots :** Le `WeightedRandomSampler` assemble un lot de 512 images, en suréchantillonnant les classes rares.
2. **Augmentation des données :** Ce lot est traité par le pipeline d'augmentation (`Cutmix`, `Mixup`, *jitter de couleur*, etc.), produisant des images synthétiques et leurs labels "soft".
3. **Passe avant (Forward Pass) :** Le lot est transmis au modèle **FastViT-SA12** pour générer les prédictions. L'entraînement est réalisé en précision mixte (AMP [6]) pour accélérer les calculs.
4. **Calcul de la perte :** La fonction `SoftTargetCrossEntropy` calcule l'erreur en comparant les prédictions du modèle aux labels "soft".
5. **Rétropropagation et Optimisation :** La perte est rétropropagée. L'optimiseur **AdamW** met à jour les poids du modèle.
6. **Mise à jour de l'EMA :** Simultanément, les poids du modèle EMA sont mis à jour.
7. **Validation :** À la fin de chaque époque, la version EMA du modèle est utilisée pour l'évaluation sur l'ensemble de validation, et le meilleur point de contrôle est sauvegardé.

6.5.5.1 Résumé des Hyperparamètres Le tableau suivant synthétise la configuration finale utilisée pour l'entraînement du classifieur.

6. **AMP (Automatic Mixed Precision)** est une technique d'entraînement qui accélère l'apprentissage profond et réduit l'utilisation de la mémoire en utilisant intelligemment des opérations en virgule flottante 16 bits et 32 bits [124].

Hyperparamètre	**Valeur**	**Justification / Rôle**
Nombre d'époques	120	Durée suffisante pour assurer la convergence sur un jeu de données complexe.
Optimiseur	AdamW	Stabilité et régularisation efficace pour les architectures de type Transformer.
Taux d'apprentissage	5×10^{-4} (initial)	Point de départ standard et efficace pour le *fine-tuning*.
Scheduler	Cosine	Raffine progressivement le taux d'apprentissage pour améliorer la convergence.
Warm-up epochs	3	Stabilise les débuts de l'entraînement en évitant des mises à jour trop brusques.
Décroissance des poids (Weight Decay)	0.05	Régularisation forte pour prévenir le surapprentissage.
Taille de batch	512	Optimise l'utilisation du GPU et stabilise les estimations de gradients.
Dropout / Drop path	0.3 / 0.1	Régularisation pour éviter la co-adaptation des neurones.
Coefficient EMA	0.9995	Coefficient de lissage pour les poids du modèle d'évaluation.
Fonction de perte	*Soft Target CrossEntropy*	Adaptée aux labels "soft" générés par Mixup et Cutmix.
Précision mixte (AMP)	Activée	Accélère l'entraînement sur les GPU compatibles.

TABLE 2.5 – Configuration d'entraînement et justification des hyperparamètres pour le modèle FastViT-SA12.

7 Conclusion

Ce chapitre a posé les fondations conceptuelles de notre système de détection et de reconnaissance d'insectes nuisibles. Nous y avons présenté une architecture modulaire en deux phases, **Détection** et **Reconnaissance** , une approche qui permet d'optimiser chaque tâche de manière indépendante pour une performance globale accrue. Les choix stratégiques des modèles, à savoir **YOLOv12** pour sa rapidité et sa précision en détection et **FastViT-SA12** pour son efficacité en classification , ont été justifiés au regard des défis posés par le dataset **IP102**.

Après avoir défini ce cadre architectural et méthodologique, le chapitre suivant se concentrera sur sa concrétisation. Nous y décrirons en détail la **mise en œuvre technique** de notre solution, avant de présenter et d'analyser les **résultats expérimentaux** obtenus lors des tests

de performance de notre système de détection et de reconnaissance.

Chapitre 3

Mise en oeuvre et résultats de l'approche proposée

1 Introduction

Après avoir posé les fondations conceptuelles de notre approche dans le chapitre précédent, ce chapitre est consacré à sa mise en œuvre pratique et à l'évaluation de ses performances. Nous commencerons par présenter l'environnement de développement, en détaillant la configuration matérielle et logicielle mobilisée pour ce projet. Ensuite, nous aborderons la phase d'expérimentation en analysant les résultats de nos deux modules principaux : d'abord la détection d'insectes nuisibles, puis leur reconnaissance. Pour chaque étape, une analyse des métriques et des résultats visuels nous permettra d'évaluer l'efficacité de notre système, d'identifier ses points forts et de discuter de ses limites.

2 Présentation de l'environnement et des outils utilisés

Cette section décrit les différents outils matériels et logiciels mobilisés au cours de la réalisation du projet. Elle présente d'abord les configurations matérielles utilisées pour l'entraînement et le test, puis les plateformes de développement adoptées, le langage de programmation choisi, ainsi que les bibliothèques principales exploitées. Ces outils ont été sélectionnés pour leur accessibilité, leur efficacité et leur adéquation avec les besoins spécifiques du projet.

2.1 Environements et Matériel utilisé

Les expérimentations ont été menées à la fois sur des plateformes locales et en environnement cloud, selon les ressources nécessaires pour chaque étape :

- **Pour l'entraînement :** nous avons utilisé la plateforme **Google Colab Pro+** [1], bénéficiant d'un **GPU NVIDIA A100** doté de **40 Go de VRAM**, ainsi que **83 Go de**

1. Colab Pro+ est un service d'abonnement premium proposé par Google, offrant un accès prioritaire à des ressources matérielles haut de gamme, notamment des GPU comme la NVIDIA A100, une mémoire étendue, et des temps d'exécution prolongés.

mémoire vive, comme décrit dans le chapitre 2. Cette configuration a permis un entraînement rapide et efficace, notamment pour les modèles lourds comme `YOLOv12n` et `FastViT-SA12`.

- **Pour les tests sur PC :** L'implémentation a été également exécutée localement sur un ordinateur personnel disposant des caractéristiques suivantes :
  - **Processeur (CPU)** : Intel Core i5-6300U cadencé à 2.40 GHz
  - **Mémoire vive (RAM)** : 8 Go
  - **Système d'exploitation** : Windows 10 Professionnel (64 bits)

  Pour ces expérimentations locales, nous avons utilisé un environnement Python basé sur **Jupyter Notebook**, exécuté sous la distribution **Anaconda**[2], facilitant ainsi la gestion des dépendances et l'exécution interactive du code.
- **Pour les tests mobiles :** le système a été testé sur un smartphone **Realme 11 4G**, dans le but d'évaluer la **latence d'inférence mobile** du modèle `FastViT-SA12`. Pour permettre ce test, le modèle a été préalablement **converti au format ONNX**, puis intégré dans une application Android développée avec **Android Studio**[3]. Cette étape visait à valider la légèreté et la rapidité du modèle dans un contexte de déploiement embarqué.

2.2 Langage de programmation et Bibliothèques

Le langage **Python** a été utilisé comme base pour l'ensemble des expérimentations. Grâce à sa simplicité et à sa richesse en bibliothèques scientifiques et en IA, il a permis de structurer efficacement les scripts d'entraînement, de prétraitement et d'évaluation des modèles [125].

Nous avons ainsi présenté les **principales bibliothèques** utilisées en complément de Python pour la mise en œuvre de notre approche :

1. **PyTorch :** Le framework PyTorch a été utilisé pour la définition et l'entraînement des modèles de détection (YOLOv12n) et de classification (FastViT-SA12). Il a permis une gestion dynamique des graphes computationnels et un contrôle fin des processus d'optimisation et d'augmentation [126].
2. **torchvision :** elle fait partie de PyTorch. Torchvision a été utilisé pour les transformations d'images standards, notamment les redimensionnements, recadrages et normalisations, ainsi que pour l'utilisation de datasets prêts à l'emploi durant les phases de test [127].
3. **timm :** La bibliothèque timm (Py**T**orch **Im**age **M**odels) a été utilisée pour charger des variantes de modèles pré-entraînés, notamment FastViT-SA12. Elle facilite l'intégration rapide de fameux modèles avec une interface cohérente pour l'entraînement et l'évaluation [128].

2. Anaconda est une distribution Python open source utilisée pour la science des données et le machine learning. `https://www.anaconda.com/`

3. Android Studio est l'environnement de développement officiel pour la création d'applications Android. `https://developer.android.com/studio`

4. **Matplotlib :** Cette bibliothèque a été utilisée pour la visualisation des courbes d'apprentissage, des distributions de classes, ainsi que des bounding boxes sur les images. Elle a facilité l'interprétation visuelle des résultats [129].
5. **Tkinter :** Tkinter a servi à la création d'une interface graphique simple permettant de charger une image et d'afficher les résultats de détection et de reconnaissance en temps réel, facilitant ainsi les démonstrations du système [130].
6. **PIL :** La bibliothèque PIL (via Pillow) a été utilisée pour le traitement des images, notamment le redimensionnement, la conversion de formats et l'annotation visuelle des prédictions durant les phases de test [131].

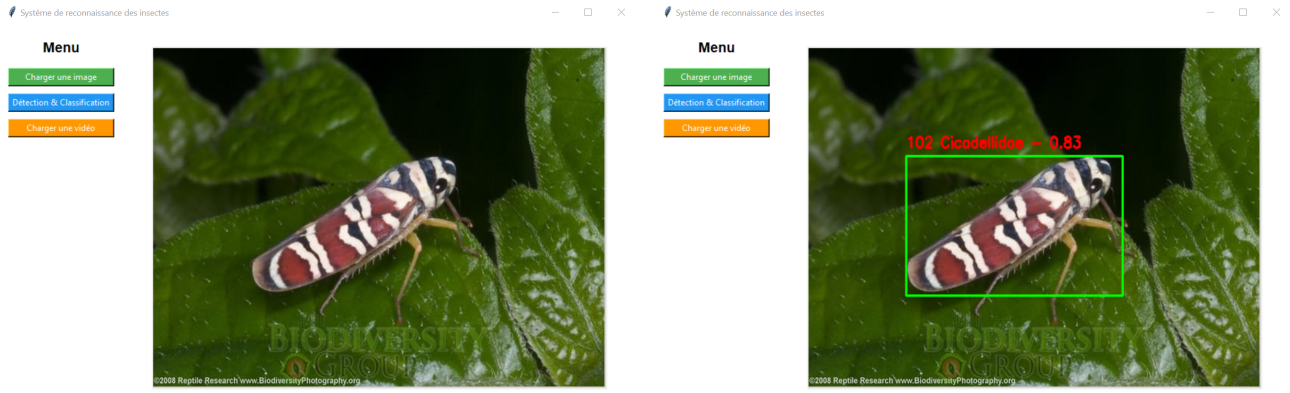
3 Interface graphique et démonstration du système

Pour démontrer l'efficacité de notre approche et fournir un outil interactif, nous avons développé une interface graphique de bureau. Conçue avec la bibliothèque `Tkinter` de Python, l'interface se veut simple et accessible, même pour un utilisateur non spécialiste. Son agencement est organisé en deux zones principales : un **panneau de contrôle latéral** contenant les boutons pour charger un média et lancer l'analyse, et une **fenêtre de visualisation centrale** où l'image et les résultats sont affichés.

Le processus d'utilisation est direct. L'utilisateur commence par charger une image ou une vidéo. Une fois le média affiché dans la zone de visualisation (Figure 3.1a), il peut lancer le traitement. Le système exécute alors le pipeline complet en arrière-plan :

1. Le modèle de détection **YOLOv12n** est appliqué pour localiser les insectes dans l'image.
2. Chaque région d'insecte détectée est ensuite recadrée et transmise individuellement au modèle de classification **FastViT-SA12**.
3. Enfin, les résultats sont affichés en temps réel sur l'image d'origine. Chaque détection est entourée d'une boîte englobante et annotée avec le nom de l'espèce prédite et son score de confiance, comme l'illustre la figure 3.1b.

Cette application fonctionnelle valide la capacité de notre système à intégrer les deux modèles pour fournir une analyse complète, de la détection à la classification, de manière simple et visuelle.



(a) Interface après le chargement d'une image, avant l'analyse.

(b) Résultat après analyse : l'insecte est détecté et classifié.

FIGURE 3.1 – Démonstration de l'interface graphique du système avant et après le traitement.

4 Résultats du modèle de détection proposé

Cette section présente les résultats expérimentaux de notre module de détection basé sur le modèle YOLOv12n. Nous commencerons par définir les métriques d'évaluation utilisées, avant d'analyser en détail les performances du modèle durant sa phase d'apprentissage et sur l'ensemble de test. Enfin, nous le comparerons aux approches de l'état de l'art pour situer notre contribution.

4.1 Métriques utilisées pour la détection

Nous décrivons les principales métriques utilisées pour évaluer les performances du modèle de détection d'insectes.

1. Précision (Precision) La précision mesure la proportion des prédictions positives correctes parmi toutes les prédictions positives émises par le modèle :

$$\text{Précision} = \frac{TP}{TP + FP}[132].$$

Elle indique **la fiabilité des détections effectuées**. Une boîte de détection est considérée comme un vrai positif (TP) si son score d'IoU (Intersection over Union, c'est-à-dire le rapport entre l'aire d'intersection et l'aire d'union des boîtes prédite et réelle) dépasse un certain seuil (par exemple 0,5), sinon elle est comptée comme un faux positif (FP).

2. Rappel (Recall) Le rappel évalue la **capacité du modèle à détecter toutes les instances pertinentes** :

$$\text{Rappel} = \frac{TP}{TP + FN}[132].$$

Il reflète la propension du modèle à ne pas omettre d'objets présents.

3. F1-score Le *F1-score* est une mesure harmonique qui combine la précision (*precision*) et le rappel (*recall*). Il est particulièrement utile lorsque les classes sont déséquilibrées. Sa formule est la suivante :

$$\text{F1} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3.1}$$

Plus le F1-score est proche de 1, meilleure est la performance du modèle en termes d'équilibre entre précision et rappel.

4. Précision Moyenne (AP) et Moyenne des Précisions Moyennes (mAP) L'*Average Precision* (AP) correspond à l'aire sous la courbe *précision–rappel* pour une classe donnée i. Mathématiquement, elle est définie comme :

$$AP_i = \int_0^1 P_i(R_i), dR_i \tag{3.2}$$

Où : R_i est le taux de rappel (recall) pour la classe i, et $P_i(R_i)$ est la précision (precision) obtenue pour un certain niveau de rappel R_i.

La **mAP** (*mean Average Precision*) est la moyenne des AP calculées pour toutes les classes du dataset :

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{3.3}$$

Cette métrique reflète globalement la capacité du modèle à détecter correctement les objets sur l'ensemble des classes.

Dans le cadre de la détection, une boîte prédite est considérée comme correcte (vrai positif) si elle atteint un certain niveau de chevauchement avec la boîte réelle, mesuré par l'IoU (*Intersection over Union*). L'IoU est défini comme le ratio entre l'aire de l'intersection et celle de l'union des deux boîtes :

$$\text{IoU} = \frac{\text{aire}(\text{prédiction} \cap \text{vérité terrain})}{\text{aire}(\text{prédiction} \cup \text{vérité terrain})} \tag{3.4}$$

La précision moyenne dépend donc du seuil d'IoU utilisé pour décider si une prédiction est correcte :

- **AP50** désigne la moyenne de précision avec un seuil d'IoU fixé à 0,5, ce qui correspond à des conditions de chevauchement relativement tolérantes ;
- **AP50–95** représente la moyenne des AP calculées pour des seuils d'IoU allant de 0,50 à 0,95 par pas de 0,05, fournissant une évaluation plus stricte et complète.

Ainsi, dans les tâches de détection multi-classes, un **mAP élevé** est un indicateur fiable de bonnes performances du modèle, en prenant en compte à la fois la précision des localisations et la qualité des classifications.

4.2 Évaluation des performances du modèle YOLOv12n durant l'apprentissage

Cette section présente et analyse les performances du modèle YOLOv12n lors de sa phase d'entraînement sur le jeu de données IP102. L'évaluation s'appuie sur les métriques de détection

clés ainsi que sur l'évolution des fonctions de perte, afin de valider la convergence et la robustesse du modèle.

4.2.1 Synthèse des résultats

L'entraînement a été mené sur 60 époques. Le tableau 3.1 résume les performances finales obtenues sur l'ensemble de validation.

Métrique	**Valeur Finale (sur *valSet*)**
Precision	0.924
Recall	0.895
F1-Score	0.910
mAP@50	0.943
mAP@50-95	0.575
Durée totale d'entraînement	~70 minutes

TABLE 3.1 – Principaux résultats de l'entraînement du modèle YOLOv12n.

Les résultats finaux témoignent de la grande efficacité du modèle. Avec une précision de 92,4% et un rappel de 89,5%, le modèle est capable d'identifier correctement la grande majorité des insectes présents dans les images. La performance la plus notable est le score **mAP@50 de 94,3%**, qui indique une excellente capacité à localiser les objets. Le score **mAP@50-95 de 57,5%** est également très encourageant, car il confirme que le modèle maintient une bonne précision de localisation même avec des critères de superposition (IoU) très stricts.

4.2.2 Analyse des courbes d'apprentissage

L'analyse des courbes d'évolution des métriques et de la perte tout au long des 60 époques permet de mieux comprendre le comportement du modèle.

4.2.2.1 Convergence du modèle La figure 3.2 illustre l'évolution de la perte de localisation (*box loss*) pour les ensembles d'entraînement et de validation. On observe une diminution rapide et continue pour les deux courbes durant les premières époques, suivie d'une stabilisation progressive. Ce comportement est idéal et témoigne d'une bonne convergence. De plus, l'écart entre les deux courbes reste constant et faible, ce qui indique que le modèle généralise bien et ne souffre pas de surapprentissage.

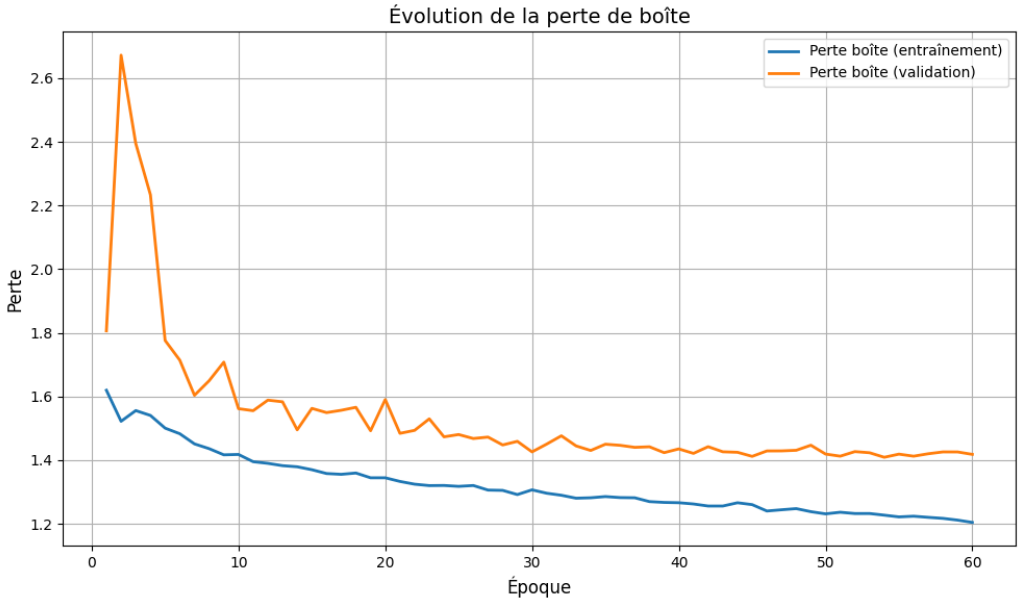
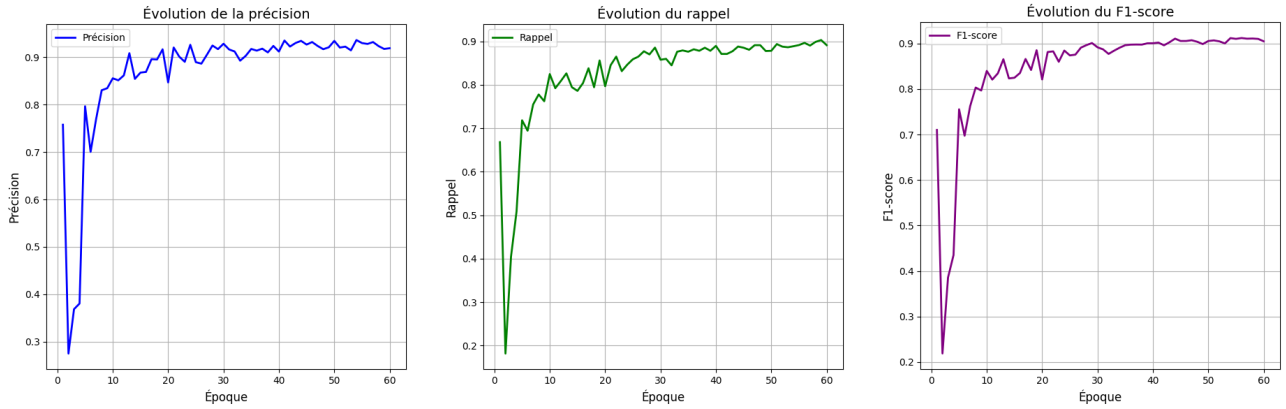


FIGURE 3.2 – Évolution de la perte de localisation (*box loss*) sur les ensembles d'entraînement et de validation.

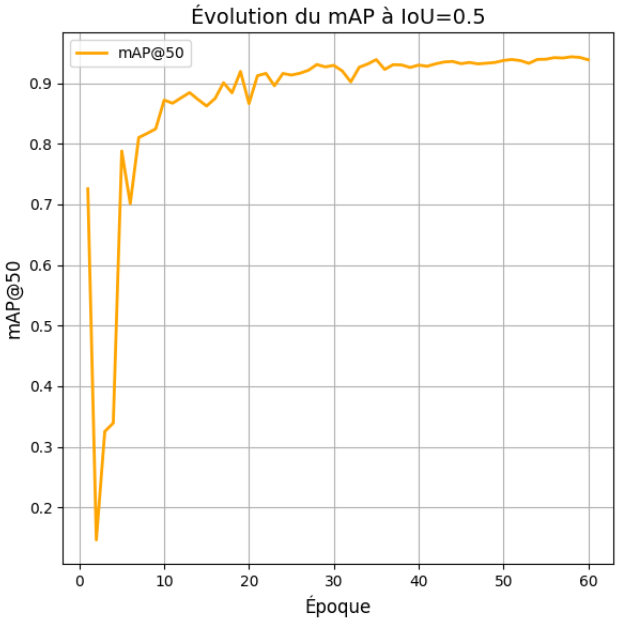
4.2.2.2 Performance de la détection La figure 3.3 montre l'évolution des métriques de performance principales. La précision, le rappel et le F1-score augmentent tous rapidement et se stabilisent à des valeurs élevées (autour de 0,9), confirmant la robustesse et la fiabilité du modèle après la phase de convergence.



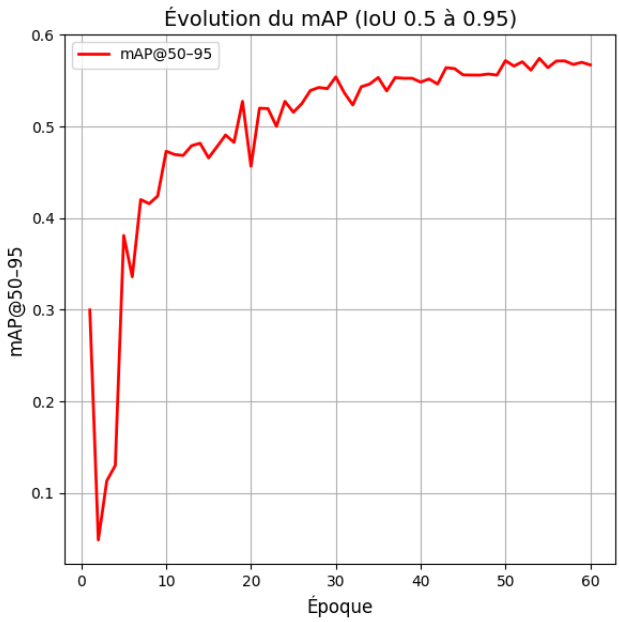
(a) Évolution de la précision (b) Évolution du rappel (c) Évolution du F1-score

FIGURE 3.3 – Évolution des métriques de détection : précision, rappel et F1-score au fil des époques.

4.2.2.3 Précision de la localisation (mAP) La figure 3.4 compare les deux métriques de mAP. La courbe du **mAP@50** (à gauche) atteint rapidement un plateau très élevé (0.94), ce qui signifie que le modèle apprend très vite à détecter la présence des insectes. La courbe du **mAP@50-95** (à droite) montre une progression plus lente mais continue, indiquant que le modèle affine progressivement la précision des boîtes englobantes pour satisfaire des critères de superposition plus exigeants. L'atteinte d'un score de 0.575 est une performance solide pour cette métrique difficile.



(a) Évolution du mAP à IoU=0.5 (mAP@50)



(b) Évolution du mAP moyen (mAP@50-95)

FIGURE 3.4 – Comparaison de l'évolution des métriques mAP à différents seuils d'IoU.

4.3 Évaluation finale des performances du modèle YOLOv12n

Après la phase d'entraînement, le meilleur modèle sauvegardé est évalué sur l'ensemble de test, qui n'a jamais été utilisé auparavant. Cette étape est cruciale pour mesurer la capacité de généralisation du modèle sur des données inconnues. Les performances sont analysées à travers plusieurs métriques et visualisations clés.

4.3.1 Synthèse des performances sur l'ensemble de test

Le tableau 3.2 résume les performances quantitatives obtenues. Le modèle atteint une précision de 93,9% et un rappel de 91,5%, ce qui confirme sa robustesse et sa fiabilité sur des données nouvelles. Ces scores élevés indiquent que le modèle produit peu de fausses détections tout en identifiant la majorité des insectes cibles.

Précision	Rappel	F1-score	mAP@50	mAP@50-95
0.939	0.915	0.927	0.958	0.607

TABLE 3.2 – Métriques de performance du modèle YOLOv5n sur l'ensemble de test.

4.3.2 Analyse de la courbe Précision-Rappel (PR)

La courbe Précision-Rappel (Figure 3.5) illustre le compromis entre la précision et le rappel pour tous les seuils de confiance. La courbe se maintient à un niveau de précision très élevé (proche de 1.0) sur une large plage de rappel, avant de chuter rapidement. Une courbe avec une grande aire sous-jacente comme celle-ci est le signe d'un modèle très performant. C'est

cette excellente performance qui explique directement le score **mAP@50 de 0.958**, car cette métrique est une approximation de l'aire sous la courbe PR.

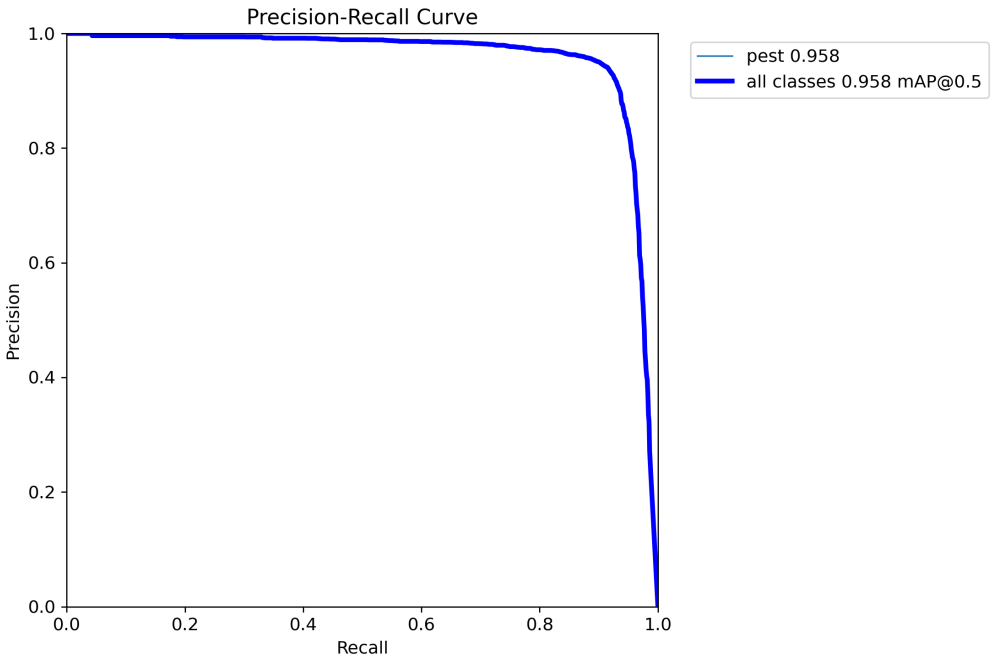
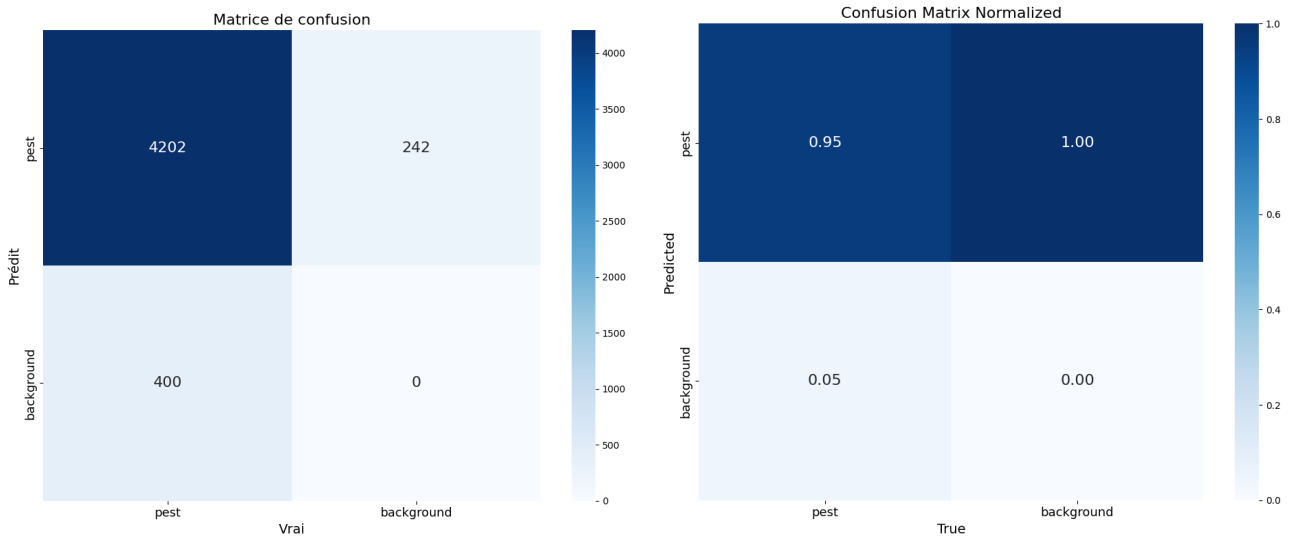


FIGURE 3.5 – Courbe Précision-Rappel du modèle sur l'ensemble de test. L'aire sous la courbe élevée correspond à un mAP de 0.958.

4.3.3 Analyse des matrices de confusion

Les matrices de confusion (Figure 3.6) permettent d'analyser en détail les erreurs de classification du modèle.



(a) Matrice de confusion (valeurs brutes) (b) Matrice de confusion (normalisée)

FIGURE 3.6 – Matrices de confusion du modèle sur l'ensemble de test du modèle de détection.

La matrice brute (à gauche) nous apprend que :

- **Vrais Positifs (TP) :** 4202 insectes ont été correctement détectés.

- **Faux Négatifs (FN) :** 400 insectes ont été manqués par le modèle (classés à tort comme "background").
- **Faux Positifs (FP) :** 242 détections étaient en réalité des zones d'arrière-plan.

La matrice normalisée (à droite), qui est calculée par ligne (par prédiction), confirme ces observations. La valeur de 0.95 dans la case en haut à gauche signifie que **lorsque le modèle prédit "pest", il a raison dans 95% des cas** (c'est la précision). Cependant, le modèle montre une tendance à la sur-détection : il préfère faire une erreur en classant un fond comme un insecte (242 fois) plutôt que de manquer un vrai insecte. Cela explique pourquoi le modèle est très bon pour trouver les insectes (rappel élevé) mais génère un nombre non négligeable de fausses alertes.

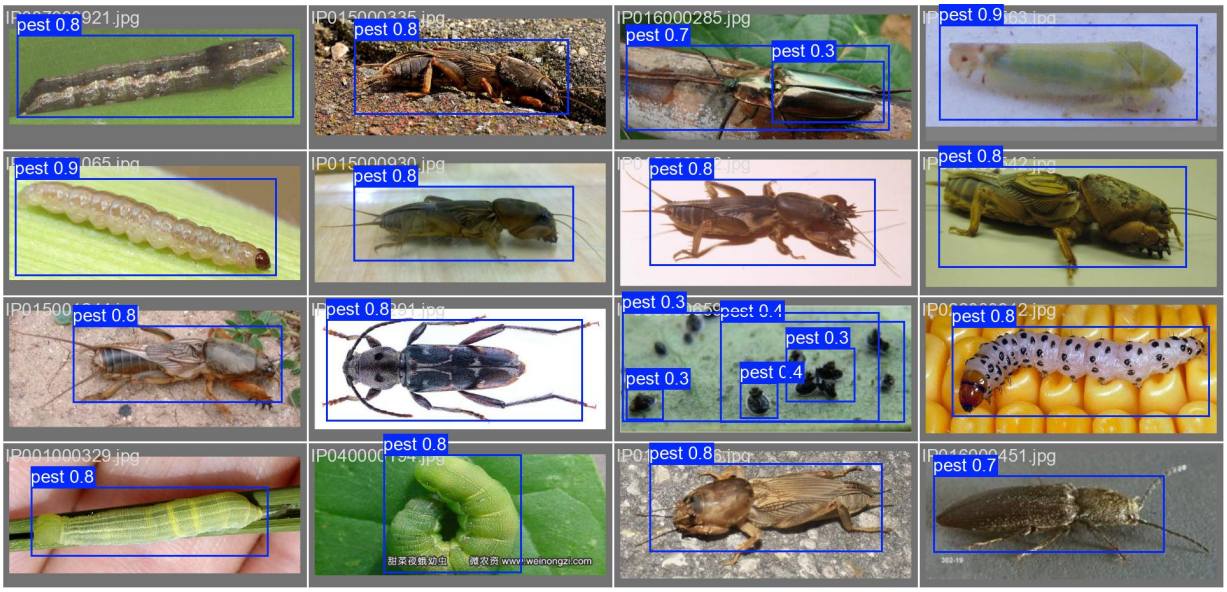


FIGURE 3.7 – Exemples de détection d'insectes sur les images de test.

4.4 Comparaison des performances sur IP102

Plusieurs travaux récents ont évalué des détecteurs d'objets sur le jeu de données IP102 (102 espèces d'insectes ravageurs) en reportant le mAP@50. Le tableau 3.3 résume cinq méthodes phares (2018–2025) et leurs performances, avec pour chacune le modèle utilisé, la référence bibliographique, l'année et le mAP@50 obtenu sur IP102.

Modèle	Référence	Année	mAP@50 (%)
C3M-YOLO [89]	Zhang et al., *Agronomy*	2023	57.2
CLT-YOLOX [89]	Zhang et al., *Agronomy*	2023	57.7
AEC-YOLOv8n [133]	Luo et al., *Agronomy*	2024	67.1
Maize-YOLO [134]	Yang et al., *Insects*	2023	76.3*
RDW-YOLO [135]	Song et al., *Insects*	2025	71.3
YOLOv12n (proposé)	**Ce travail**	**2025**	**95.80%**

TABLE 3.3 – Comparaison des performances de détection sur IP102 entre YOLOv12n et d'autres modèles récents. *Résultat obtenu sur un sous-ensemble (maïs) d'IP102.

4.4.1 Analyse comparative

En comparant ces résultats à notre modèle **YOLOv12n**, on constate que ce dernier **surpasse** les méthodes de l'état de l'art en atteignant un **score mAP@50 de 95,80 %** sur le dataset IP102, soit une avance significative par rapport aux meilleurs modèles existants. De plus, YOLOv12n se démarque par sa **vitesse d'inférence élevée** et sa **légèreté**, grâce à sa version « nano » spécifiquement conçue pour les environnements à ressources limitées. Il intègre une architecture centrée sur l'attention optimisée, notamment avec les modules **FlashAttention** et **Area Attention**, tout en conservant des performances en temps réel. Cette conception permet une **meilleure détection des petits objets** [136], un atout crucial pour l'identification précise d'insectes souvent discrets et visuellement complexes dans des environnements naturels variés.

Ainsi, contrairement à certaines méthodes spécialisées (p.ex. Maize-YOLO) qui obtiennent de bons résultats dans des contextes restreints, notre modèle **YOLOv12n** se distingue par sa **supériorité globale**, en atteignant un **mAP@50 de 95,80 %** sur l'ensemble IP102, soit un écart significatif par rapport aux meilleurs résultats précédemment rapportés. Son architecture attentionnelle optimisée lui permet non seulement d'atteindre une précision exceptionnelle, mais également de rester **plus compact et rapide** – un avantage déterminant pour les applications embarquées ou sur le terrain. En résumé, **YOLOv12n combine efficacité, rapidité et robustesse**, en particulier pour la détection des objets de petite taille, ce qui en fait une solution de pointe pour la détection automatisée des insectes dans des environnements agricoles complexes et variés.

5 Résultats du modèle de classification

Cette section est consacrée à l'évaluation complète des résultats de notre module de classification, basé sur le modèle **FastViT-SA12**. Nous y détaillerons les métriques utilisées, analyserons la phase d'apprentissage pour valider la convergence du modèle, puis nous évaluerons ses performances finales sur l'ensemble de test. Enfin, nous justifierons le choix de la variante SA12

et nous la comparerons aux travaux de l'état de l'art pour contextualiser notre contribution.

5.1 Métriques utilisées pour la classification

Pour évaluer la performance de notre module de classification (FastViT-SA12), nous utilisons un ensemble de métriques standards en reconnaissance d'images.

1. **Précision, Rappel et F1-Score** : ces métriques sont également fondamentales pour évaluer la classification. Leurs définitions sont identiques à celles présentées pour la détection (voir section 4.1). Dans ce contexte, un Vrai Positif (TP) correspond à une image correctement classée dans sa catégorie, un Faux Positif (FP) à une image classée dans une catégorie incorrecte, et un Faux Négatif (FN) à une image qui n'a pas été reconnue comme appartenant à sa véritable catégorie.
2. **Top-1 Accuracy** : c'est est la métrique la plus courante pour la classification. Elle mesure simplement la proportion d'images pour lesquelles la prédiction principale du modèle (la classe avec le score de confiance le plus élevé) est correcte.

$$\text{Top-1 Accuracy} = \frac{\text{Nombre de prédictions Top-1 correctes}}{\text{Nombre total d'images}}$$

   Une Top-1 Accuracy élevée indique que le modèle est très souvent correct dans sa première proposition.
3. **Top-5 Accuracy** : c'est la métrique plus souple. Elle mesure la proportion d'images pour lesquelles la véritable classe se trouve parmi les cinq classes les plus probables prédites par le modèle.

$$\text{Top-5 Accuracy} = \frac{\text{Nombre d'images où la bonne classe est dans le Top-5}}{\text{Nombre total d'images}}$$

   Cette métrique est particulièrement utile pour les jeux de données avec un grand nombre de classes (comme IP102 avec 102 classes), où la distinction entre certaines espèces peut être subtile. **Elle indique si le modèle a une "bonne idée" de la réponse, même si sa première prédiction n'est pas parfaite**.

5.2 Performances de FastViT-SA12 en apprentissage

L'évaluation des performances durant la phase d'apprentissage est essentielle pour valider la convergence du modèle et sa capacité à généraliser. Cette section analyse les métriques clés obtenues lors de l'entraînement du modèle FastViT-SA12 sur 110 époques.

5.2.1 Synthèse des résultats d'entraînement

Le tableau 3.4 synthétise les performances optimales atteintes au cours de l'entraînement. Il est important de noter que ces performances sont celles du modèle **EMA (Exponential Moving Average)**, qui est utilisé pour l'évaluation sur l'ensemble de validation.

Métrique	Valeur
Meilleure Époque (basée sur EMA)	65 / 110
Accuracy Top-1 (EMA)	**75,28 %**
Accuracy Top-5 (EMA)	**93,31 %**
Perte d'entraînement (finale)	~1.20
Perte de validation (finale)	~1.15
Durée totale d'entraînement	~181 minutes

TABLE 3.4 – Principaux résultats de l'entraînement du modèle FastViT-SA12.

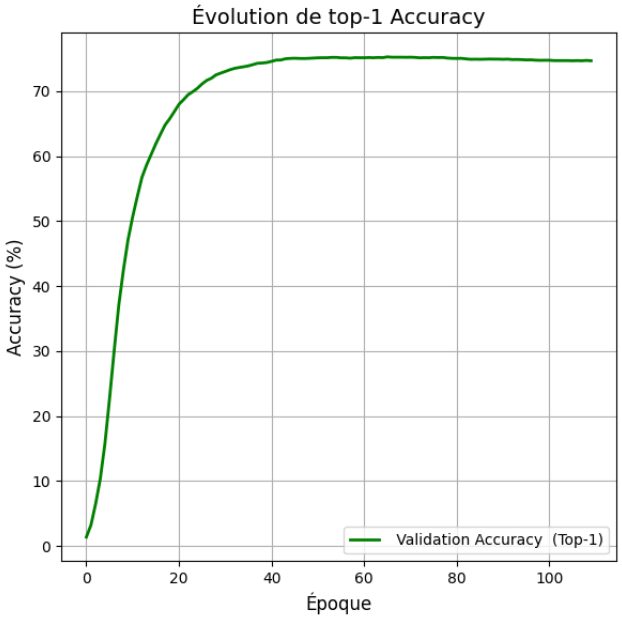
Justification de l'évaluation avec EMA L'entraînement peut faire osciller les poids du modèle principal à chaque mise à jour. Le modèle EMA, en étant une moyenne lissée de ces poids sur plusieurs itérations, représente une version plus "stable" et plus générale du modèle. Évaluer cette version lissée sur le jeu de validation donne une estimation plus fiable de la performance réelle du modèle sur des données nouvelles. C'est pourquoi le meilleur point de contrôle est déterminé en se basant sur les performances du modèle EMA, qui sont souvent un meilleur indicateur de la capacité de généralisation du modèle.

Analyse des performances Les résultats du modèle EMA sont très positifs. Le modèle atteint une excellente précision **Top-5 de 93,31%**, ce qui signifie que pour la quasi-totalité des images de validation, la bonne classe se trouve dans les cinq premières prédictions. La précision **Top-1 de 75,28%** confirme que le modèle est capable d'identifier correctement la bonne espèce dans la majorité des cas.

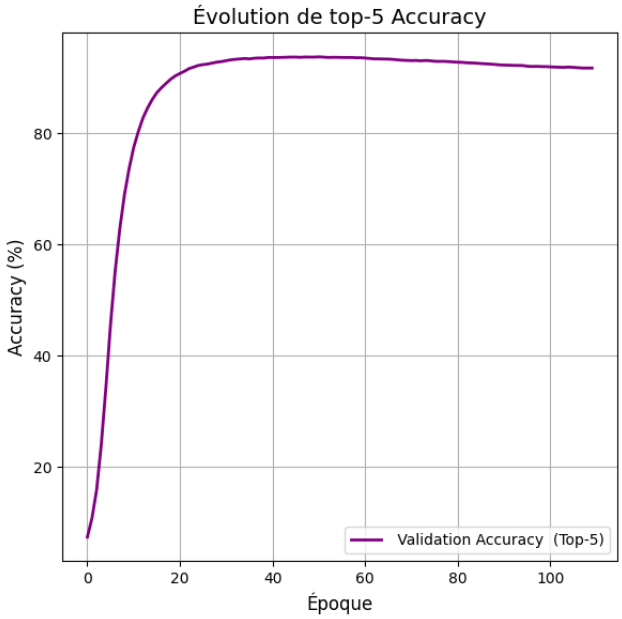
5.2.2 Analyse des courbes d'apprentissage

Les figures ci-dessous illustrent la dynamique de l'apprentissage du modèle EMA au fil des époques.

Évolution des précisions (Top-1 et Top-5) La figure 3.8 montre l'évolution des précisions Top-1 et Top-5 du modèle EMA sur l'ensemble de validation. Les deux courbes suivent une trajectoire idéale : une croissance très rapide durant les 40 premières époques, suivie d'une phase de stabilisation. Cette stabilisation indique que le modèle a atteint sa capacité d'apprentissage maximale avec les données et la configuration fournies. L'écart important entre la Top-1 et la Top-5 est typique des problèmes de classification fine et confirme que, même lorsque le modèle hésite, il a une compréhension très juste des classes possibles.



(a) Évolution de la précision Top-1 (EMA)



(b) Évolution de la précision Top-5 (EMA)

FIGURE 3.8 – Évolution des précisions Top-1 et Top-5 du modèle EMA sur l'ensemble de validation.

Analyse de la convergence et du surapprentissage La figure 3.9 présente les courbes de perte pour l'entraînement (en bleu) et la validation (en rouge). On observe une diminution rapide et synchronisée des deux pertes, ce qui témoigne d'une convergence réussie. Fait très important, la courbe de perte de validation se stabilise à un niveau inférieur à celle de l'entraînement. Ce phénomène est un signe très positif : il indique que le modèle **généralise extrêmement bien** et qu'il n'y a **aucun signe de surapprentissage** (*overfitting*). Les stratégies de régularisation et d'augmentation de données ont donc été particulièrement efficaces.

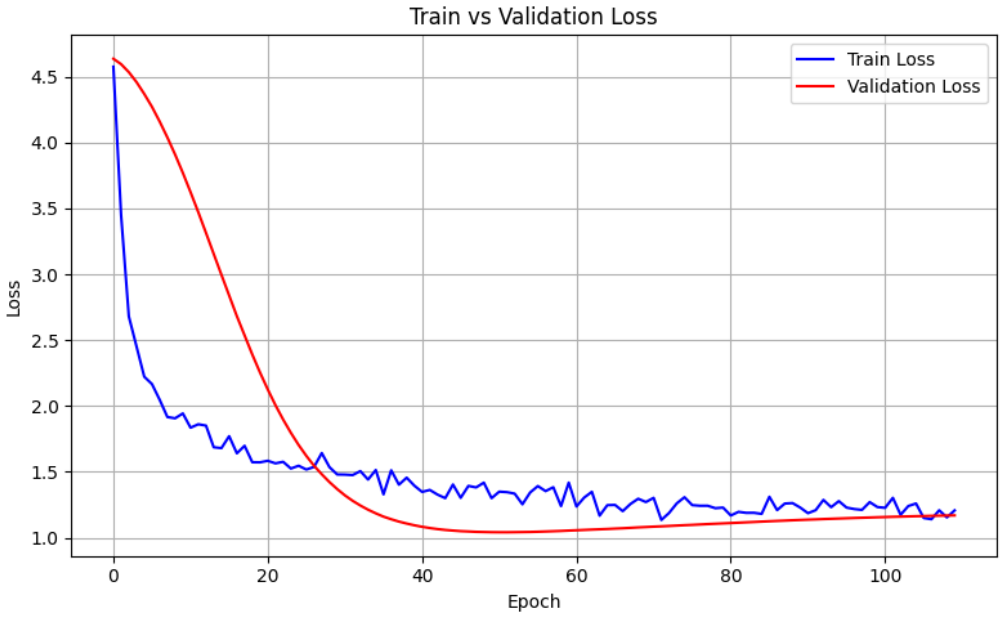


FIGURE 3.9 – Courbes de perte (*loss*) durant l'entraînement et la validation.

5.3 Analyse des performances du modèle FastViT-SA12

Cette section présente une analyse détaillée des performances du classifieur FastViT-SA12 sur l'ensemble de test du jeu de données IP102. L'objectif est de quantifier la performance globale du modèle, d'identifier ses points forts, mais aussi d'analyser ses faiblesses pour comprendre les axes d'amélioration possibles.

5.3.1 Performance globale et exemples de prédictions

Sur l'ensemble de test, notre modèle **FastViT-SA12** a obtenu des résultats très prometteurs dans le cadre de la classification fine des insectes sur le dataset IP102. Il atteint une **Top-1 Accuracy de 73,77%**, ce qui signifie que le modèle identifie correctement l'espèce d'insecte dans près de trois cas sur quatre. De plus, la **Top-5 Accuracy s'élève à 90,82%**, indiquant que la bonne classe se trouve parmi les cinq premières prédictions du modèle dans plus de neuf cas sur dix, ce qui est particulièrement pertinent pour des classes visuellement proches.

L'analyse est également enrichie par une **précision moyenne (Precision)** de **73,63%**, un **rappel moyen (Recall)** de **73,77%**, et un **F1-score global** de **73,52%**. Le F1-score étant la moyenne harmonique entre la précision et le rappel, sa proximité avec l'accuracy confirme la bonne stabilité du modèle et sa capacité à bien généraliser, même en présence d'un grand nombre de classes (102) et de déséquilibres dans les données.

Enfin, le modèle montre une bonne efficacité en déploiement mobile, avec une **latence d'inférence de 130 ms** sur un smartphone Android (Realme 11), ce qui le rend adapté aux applications en temps réel.

Métrique	**Valeur**
Accuracy (Top-1)	73,77%
Accuracy (Top-5)	90,82%
Précision moyenne	73,63%
Rappel moyen	73,77%
F1-score global	73,52%
Latence mobile	130 ms

La figure 3.10 montre des exemples de prédictions du modèle sur des images de test. On y observe des cas de classification parfaite, où le modèle identifie sans erreur des espèces comme le "rice leaf roller" ou le "mole cricket". On note également une erreur de classification où un "wireworm" est confondu avec un "corn borer", illustrant les défis liés à la similarité entre certaines espèces.

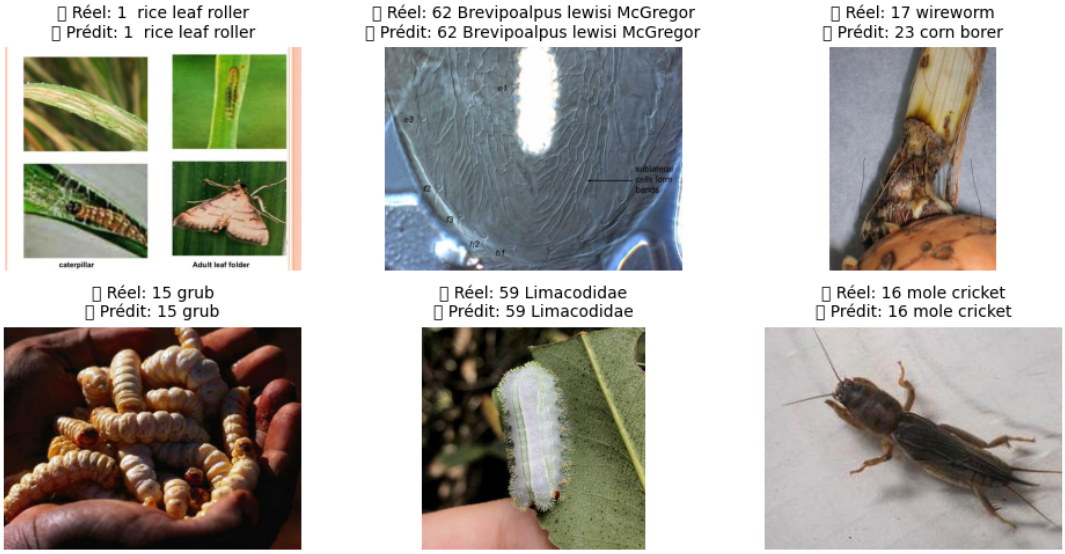


FIGURE 3.10 – Exemples de prédictions sur des images de test, avec les classes réelles et prédites.

5.3.2 Analyse des classes les mieux reconnues

Pour comprendre où le modèle excelle, nous avons analysé les 20 classes pour lesquelles la précision est la plus élevée (supérieure à 70%). La matrice de confusion de la figure 3.11 illustre ces excellentes performances. La diagonale fortement marquée indique que les prédictions sont très majoritairement correctes. Des classes comme *16 mole cricket* (479 prédictions correctes), *59 Limacodidae* (390), ou *68 Lycorna delicatula* (1437) sont reconnues avec une très grande fiabilité et très peu de confusion. Cela démontre que le modèle est particulièrement robuste et performant sur les classes visuellement distinctes et bien représentées dans le jeu de données.

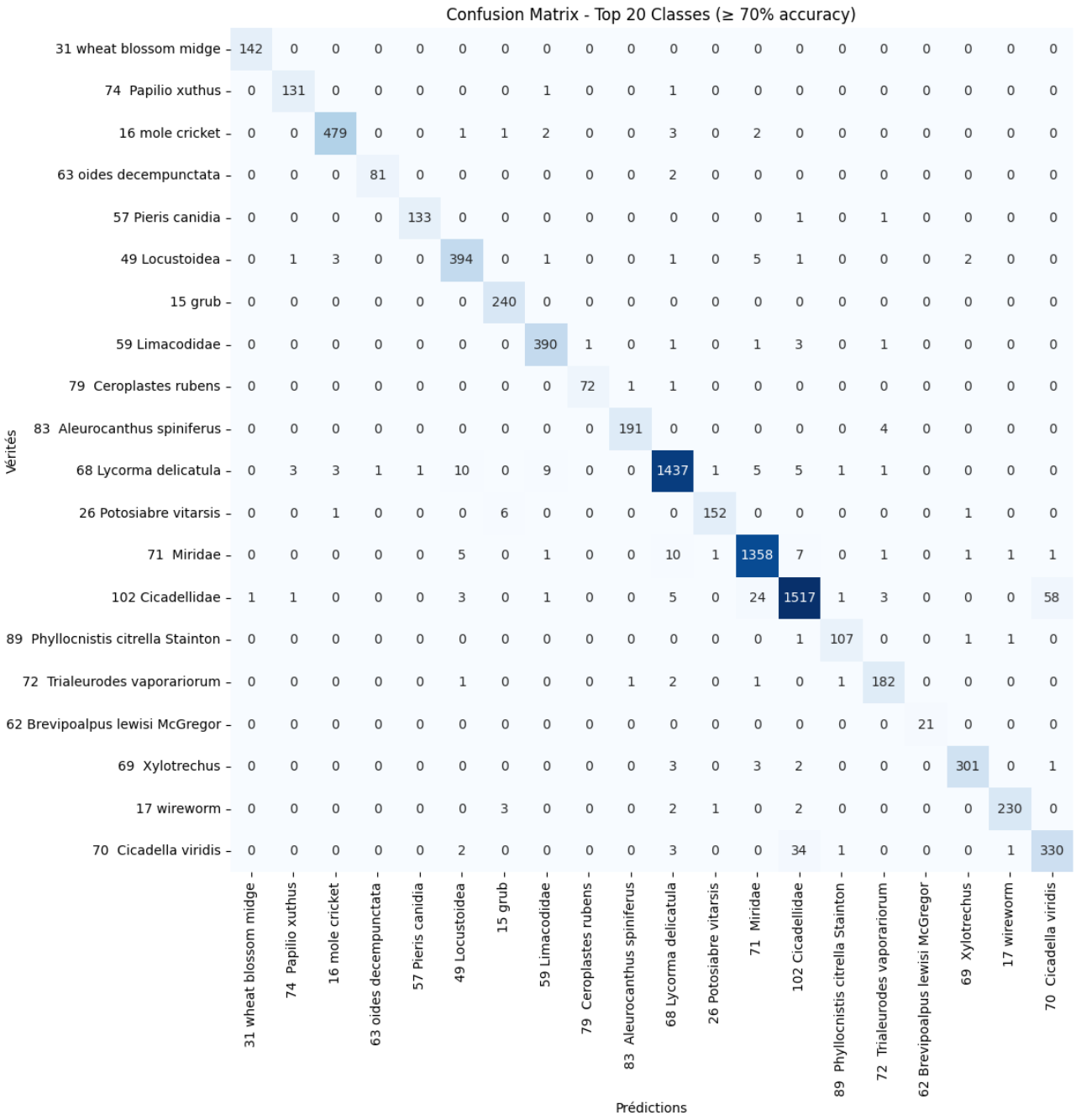
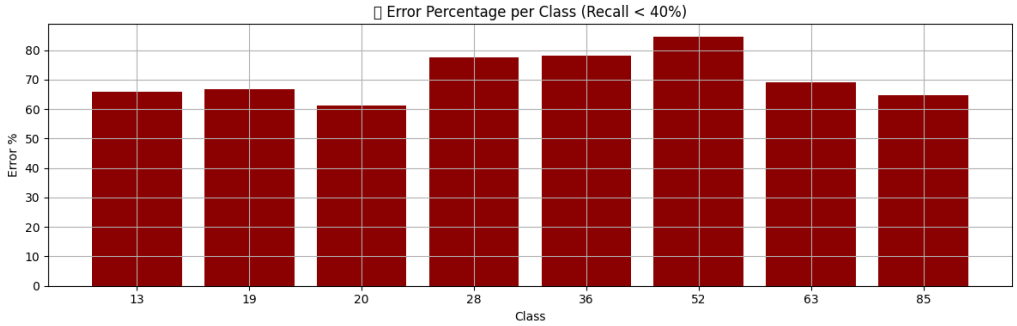


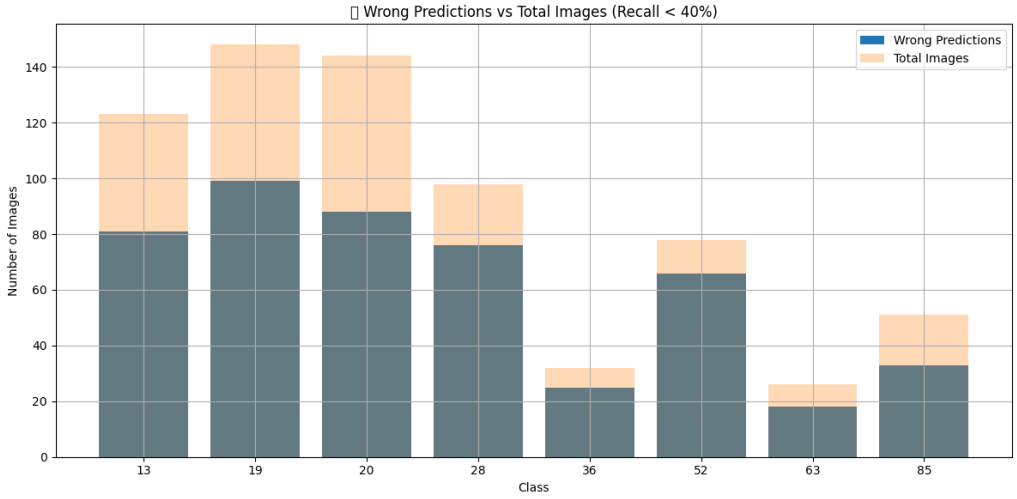
FIGURE 3.11 – Matrice de confusion pour les 20 classes les mieux classifiées (accuracy > 70%).

5.3.3 Analyse des classes les moins performantes

À l'inverse, il est crucial d'identifier et de comprendre les faiblesses du modèle. La figure 3.12 analyse les classes pour lesquelles le rappel est inférieur à 40%. Le graphique de gauche montre que pour ces classes, le taux d'erreur est systématiquement supérieur à 60%, atteignant même plus de 80% pour la classe 52. Le graphique de droite confirme que ce n'est pas seulement un pourcentage : pour la plupart de ces classes, le nombre d'erreurs est bien supérieur au nombre de prédictions correctes.



(a) Pourcentage d'erreurs pour chaque classe.



(b) Nombre d'erreurs vs total d'images.

FIGURE 3.12 – Analyse des erreurs pour les classes avec un rappel inférieur à 40%.

Ces faibles performances s'expliquent principalement par deux facteurs : un **faible nombre d'échantillons** pour ces classes dans le jeu de données (déséquilibre) et une **forte similarité visuelle** avec d'autres espèces.

5.3.4 Étude de cas : analyse des confusions

Pour illustrer concrètement ces difficultés, les figures 3.13 et 3.14 analysent les erreurs en détail.

La matrice de confusion des classes les moins performantes (Figure 3.13) révèle où se produisent les erreurs. On observe par exemple de fortes confusions entre les classes **20 (large cutworm)** et **21 (yellow cutworm)**, qui sont deux types de vers-gris très similaires. De même, la classe **29 (green bug)** est souvent confondue avec la classe 22.

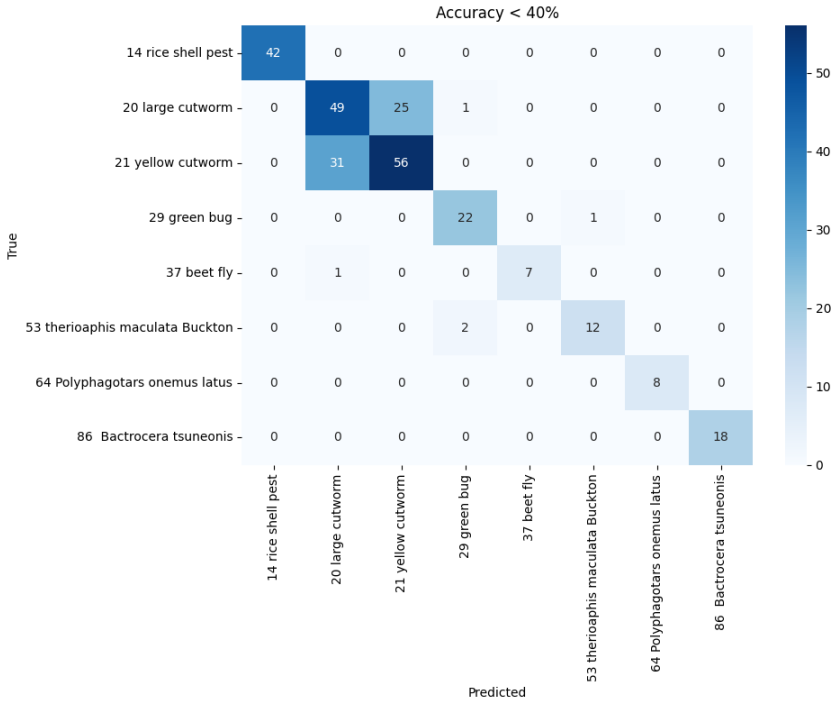
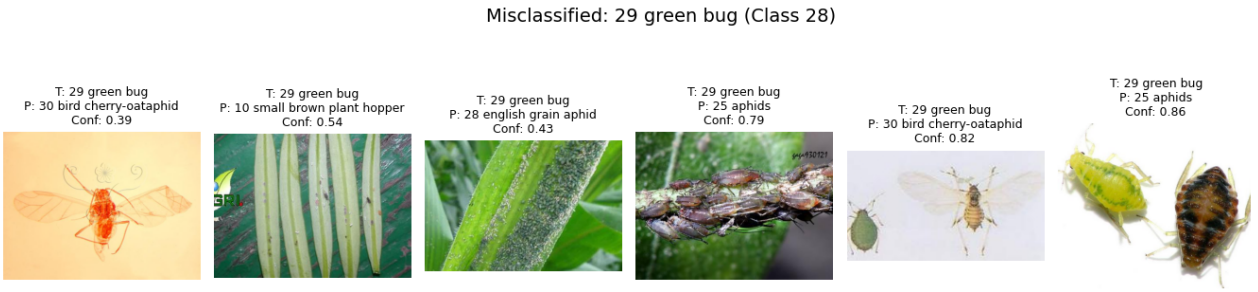
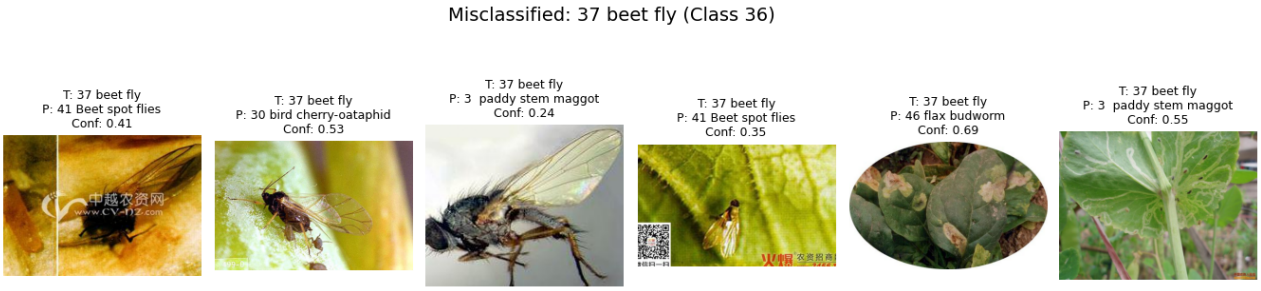


FIGURE 3.13 – Matrice de confusion des classes avec une accuracy inférieure à 40%.

Les exemples visuels de la figure 3.14 confirment ces hypothèses. Pour la classe **28 (green bug)**, le modèle la confond avec d'autres pucerons ou de petits insectes verts. Pour la classe **36 (beet fly)**, les images sont bruitées et les insectes difficiles à distinguer, ce qui explique les prédictions erronées. Cette analyse qualitative met en évidence que les erreurs du modèle sont souvent dues à des ambiguïtés visuelles réelles.



(a) Exemples d'erreurs pour la classe 28 (green bug).



(b) Exemples d'erreurs pour la classe 36 (beet fly).

FIGURE 3.14 – Analyse qualitative des erreurs de classification pour deux classes difficiles.

L'évaluation du modèle FastViT-SA12 sur le jeu de test IP102 montre une performance

globale solide et encourageante. Le modèle excelle sur les classes bien définies et distinctes. Ses faiblesses se situent principalement sur un sous-ensemble de classes rares ou visuellement ambiguës, où les confusions sont fréquentes.

5.4 Justification du choix du modèle FastViT-SA12

Afin de sélectionner la variante de FastViT la plus adaptée à notre application, nous avons mené une analyse comparative. Plusieurs modèles de la famille FastViT ont été évalués sur notre jeu de test issu d'IP102, en mesurant à la fois la précision de classification (Top-1 Accuracy) et la vitesse d'inférence (latence) sur un appareil mobile (Realme 11) pour simuler des conditions d'utilisation réelles.

Le tableau 3.5 et la figure 3.15 synthétisent les résultats de cette expérimentation.

Modèle	**Accuracy Top-1 (%)**	**Latence moyenne (ms)**
FastViT-T8	69.5	65.0
FastViT-T12	72.3	97.5
FastViT-S12	72.9	113.8
FastViT-SA12	**73.77**	**130.0**
FastViT-SA24	73.8	211.3
FastViT-SA36	74.3	284.4
FastViT-MA36	74.5	373.8

TABLE 3.5 – Comparaison des modèles FastViT en termes de précision et de latence sur le jeu de test IP102.

Comme l'illustre la figure 3.15, les résultats confirment le compromis classique entre performance et rapidité. Les modèles légers comme 'FastViT-T8' sont très rapides mais offrent une précision limitée. À l'inverse, les modèles plus lourds comme 'FastViT-MA36' atteignent la meilleure précision, mais au prix d'une latence très élevée qui les rend inadaptés à une application en temps réel.

Dans ce contexte, le modèle **FastViT-SA12** émerge comme le choix optimal. Il se positionne parfaitement dans la "zone idéale" (en vert sur la figure), où l'équilibre entre la précision et la latence est le meilleur. Avec une **précision de 73,77%**, il surpasse nettement les modèles plus légers, tout en conservant une latence moyenne de **130 ms**. Ce temps de traitement est plus de deux fois plus rapide que celui des modèles plus performants, pour une différence de précision marginale, ce qui en fait le candidat idéal.

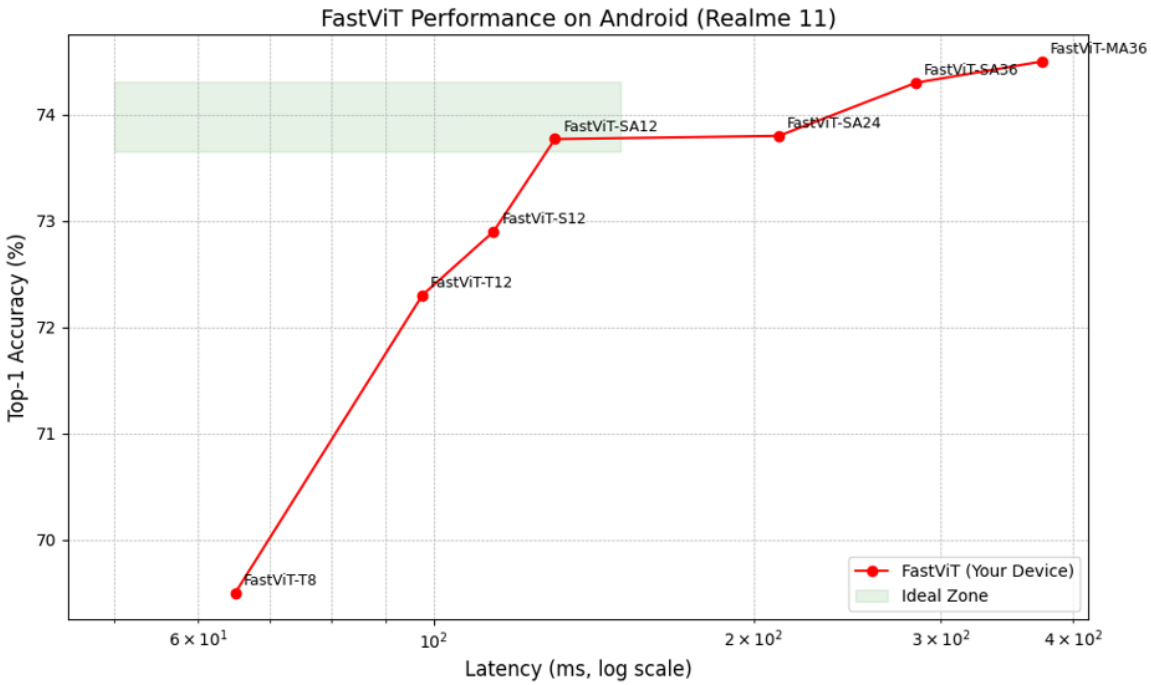


FIGURE 3.15 – Compromis entre la précision Top-1 et la latence des modèles FastViT sur un appareil mobile.

Un prototype fonctionnel a été développé et déployé sous forme d'application mobile (Figure 3.16) afin d'évaluer concrètement les performances du modèle en situation réelle. Ce prototype a permis de **mesurer la latence complète du processus de classification**. Les tests réalisés confirment que le modèle fonctionne de manière fluide sur un dispositif embarqué, avec des temps de réponse compatibles avec une utilisation interactive en temps réel.

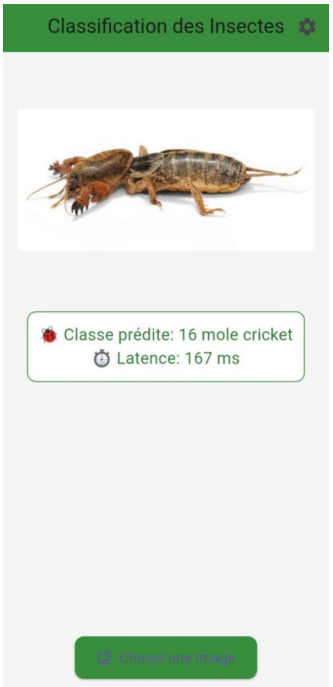


FIGURE 3.16 – Exemple d'interface mobile pour la classification d'insectes, illustrant une prédiction du modèle FastViT-SA12. Elle est utilisée pour calculer la latence.

5.5 Comparaison aux modèles de l'état de l'art

Afin de situer objectivement les performances de notre modèle **FastViT-SA12**, nous le comparons à plusieurs approches de la littérature évaluées sur le jeu de données IP102. Ce jeu de données, riche de 102 classes d'insectes, présente des défis importants liés à la grande variabilité intra-classe, aux similarités inter-espèces et à la distribution déséquilibrée des échantillons.

Le tableau 3.6 présente une synthèse de ces comparaisons.

Modèle	**Référence (Auteurs, Année)**	**Acc. Top-1**	**F1-score**
ResNet-50	Wu et al., 2019	49.5 %	40.6 %
FR-ResNet (Feature Reuse)	Ren et al., 2019	55.2 %	–
DMF-ResNet (Multi-branch)	Liu et al., 2020	59.1 %	–
EfficientNet + SVM	Doan et al., 2021	71.84 %	–
CRN (Conv. Rebalancing Net)	Yang et al., 2021	70.4 %	–
STN+ResNet50 (avec attention)	Yang et al., 2021	73.3 %	–
SGDL-Net	Luo et al., 2021	72.7 %	60.6 %
MS-ALN (Multiscale Att. Net)	Feng et al., 2022	72.91 %	–
Ensemble CNN (GAEnsemble)	Nanni et al., 2022	74.11 %	–
PCNet (mobile model)	Zheng et al., 2023	73.7 %	–
WBEnsemble (Vec+MatEnsemble)	Qian et al., 2024	**77.39 %**	–
FastViT-SA12 (Notre approche)	**Ours, 2025**	**73.77 %**	**73.51 %**

TABLE 3.6 – Comparaison des performances de classification sur le jeu de données IP102.

5.5.1 Analyse des résultats

Notre modèle **FastViT-SA12** atteint une **Top-1 Accuracy de 73,77%**, une performance qui le positionne très favorablement face aux approches de l'état de l'art. Il surpasse de nombreux modèles spécialisés comme MS-ALN (72,91%) et PCNet (73,7%), et se montre très compétitif face à des approches d'ensemble complexes comme GAEnsemble (74,11%).

Bien que le modèle le plus performant du tableau soit WBEnsemble (77,39%), une méthode d'ensemble lourde, notre approche se distingue sur deux critères fondamentaux pour une application pratique : **l'équilibre des performances** et **l'efficacité architecturale**.

Premièrement, notre modèle obtient un **F1-score de 73,51%**, une valeur exceptionnellement élevée qui démontre un excellent équilibre entre la précision et le rappel. Cette métrique, souvent plus révélatrice sur des jeux de données déséquilibrés comme IP102, est rarement rapportée par les autres travaux, ce qui rend la performance de notre modèle d'autant plus significative.

Deuxièmement, et c'est là son avantage majeur, **FastViT-SA12 est une architecture unique, légère et conçue pour être rapide**, notamment sur des dispositifs mobiles. Contrairement aux approches d'ensemble (GAEnsemble, WBEnsemble) qui combinent plusieurs modèles lourds et sont difficiles à déployer, notre solution offre une précision de premier plan dans une architecture directement applicable sur le terrain.

Notre approche ne se contente pas d'être compétitive en termes de précision brute, mais elle offre une solution **efficace, équilibrée et pragmatique**, parfaitement adaptée aux contraintes des applications agricoles en conditions réelles.

6 Discussion Générale et Perspectives

L'évaluation individuelle des modules de détection et de classification a montré des performances solides. Cette section vise à synthétiser ces résultats, à discuter des forces et des limites du système global, et à tracer des perspectives pour les travaux futurs.

6.0.0.1 Synthèse de la performance du système modulaire

Notre approche en deux étapes, combinant la rapidité de **YOLOv12n** pour la détection et la précision de **FastViT-SA12** pour la classification, s'est avérée être une stratégie gagnante. Le détecteur, avec un mAP@50 de 95,8%, agit comme un filtre très efficace qui permet de ne soumettre au classifieur que des régions d'intérêt pertinentes. Cette synergie permet au système d'être à la fois rapide et précis, en évitant d'analyser l'intégralité de l'image avec le classifieur, plus coûteux en calculs. La performance finale de classification de 73,77% Top-1, obtenue sur des images déjà localisées, est donc le reflet d'une performance en conditions quasi-réelles.

Limites de l'approche et pistes d'amélioration Malgré ses succès, notre système présente des limites inhérentes à son architecture et à ses performances, qui ouvrent la voie à de futures améliorations :

- **Propagation des erreurs :** La performance globale est plafonnée par le maillon le plus faible. Un insecte manqué par YOLOv12n (un faux négatif en détection) ne sera jamais classifié. De même, une détection imprécise (une boîte englobante mal placée) peut nuire à la qualité de la classification.
- **Gestion des classes difficiles :** Malgré une performance globale très compétitive qui rivalise avec l'état de l'art , notre analyse détaillée révèle que le modèle peine encore sur certaines classes spécifiques. Ces difficultés sont souvent dues à une forte similarité visuelle entre espèces ou à leur sous-représentation dans le jeu de données. Le système actuel, qui traite toutes les classes de la même manière après la détection, ne dispose pas de mécanisme pour gérer ces cas particuliers.

- **Latence pour le traitement vidéo en temps réel :** Bien que FastViT-SA12 soit optimisé pour le mobile, sa latence mesurée à 130 ms pour la classification, à laquelle s'ajoute le temps de détection, peut constituer un frein pour des applications vidéo très fluides (par exemple, analyse sur un drone à 15-30 images par seconde). Une optimisation supplémentaire est donc souhaitable.

Ces limites tracent plusieurs perspectives d'amélioration claires :

1. **Amélioration des classes faibles :** Mettre en place des stratégies de *data augmentation* ciblées ou des techniques d'apprentissage comme le *few-shot learning* pour les classes les moins performantes.
2. **Optimisation de la latence :** Explorer des techniques de **quantification post-entraînement** (par exemple, la conversion des poids en 8 bits) ou de **distillation de connaissances** pour créer une version encore plus rapide du classifieur, sans sacrifier la précision de manière significative.
3. **Validation en conditions réelles étendues :** Tester la robustesse du système face à des variations non présentes dans le dataset, comme différentes conditions météorologiques, d'éclairage, ou sur d'autres espèces de cultures.

Implications pour l'agriculture de précision En conclusion, ce travail démontre qu'un système modulaire bien conçu peut fournir un outil puissant et pragmatique pour la surveillance automatisée des cultures. La combinaison d'un détecteur rapide et d'un classifieur léger et précis est particulièrement adaptée aux contraintes du monde agricole, où des solutions déployables sur des appareils mobiles ou des drones sont nécessaires, et où chaque milliseconde de latence économisée améliore l'interactivité et l'autonomie de l'appareil. Notre système constitue une base solide pour le développement futur d'outils d'aide à la décision pour les agriculteurs.

7 Conclusion

Ce chapitre a détaillé la mise en œuvre et l'évaluation de notre système intelligent pour la reconnaissance des insectes nuisibles. L'approche modulaire, combinant un module de détection **YOLOv12n** et un module de classification **FastViT-SA12**, a démontré une efficacité remarquable sur le jeu de données IP102.

Les résultats expérimentaux ont validé la performance de chaque composant. Le module de détection a atteint un excellent score **mAP@50 de 95,8%**, garantissant une localisation fiable des insectes. Le module de classification, quant à lui, a obtenu une **Top-1 Accuracy de 73,77%**, une performance très compétitive qui rivalise avec les approches de l'état de l'art. L'analyse a confirmé la robustesse du système, tout en identifiant des limites sur certaines classes visuellement similaires, qui constituent une piste claire pour de futurs travaux.

En conclusion, ce système constitue une solution complète et pragmatique. Ses performances solides, alliées à l'efficacité de ses modèles en conditions réelles, en font un outil prometteur pour la surveillance agricole automatisée, répondant ainsi aux besoins de l'agriculture de précision.

Conclusion Générale

Dans ce projet de fin d'études, nous nous sommes intéressés au domaine de l'agriculture de précision et avons proposé une approche modulaire pour la reconnaissance intelligente des insectes nuisibles. Ce travail a permis de développer un système complet, de sa conception théorique à sa validation expérimentale. Nous clôturons ce mémoire par une synthèse de nos contributions, les perspectives de recherches futures, et un résumé des connaissances acquises durant ce projet.

Contributions

La contribution principale de ce travail réside dans la conception et la mise en œuvre d'un système de reconnaissance d'insectes de bout en bout, performant et adapté aux contraintes du monde réel. Nos apports peuvent être résumés en trois points :

- **Développement d'une architecture modulaire efficace :** Nous avons validé une approche en deux phases (**Détection** → **Classification**) qui s'est avérée très robuste. La synergie entre un détecteur rapide et un classifieur précis permet d'optimiser les performances globales.
- **Validation de modèles de pointe sur le dataset IP102 :** Nous avons démontré l'efficacité de modèles récents sur cette tâche complexe. Notre module de détection, basé sur **YOLOv12n**, a atteint un excellent **mAP@50 de 95,8%**. Le module de classification, avec **FastViT-SA12**, a obtenu une **Top-1 Accuracy de 73,77%**, une performance qui rivalise avec les meilleurs travaux de la littérature tout en utilisant une architecture légère et rapide.
- **Réalisation d'un prototype fonctionnel :** Au-delà des expériences, nous avons implémenté un démonstrateur avec une interface graphique, prouvant la faisabilité d'intégrer notre système dans une application concrète et validant sa performance en conditions de déploiement mobile (latence de 130 ms).

Limites et Perspectives

Malgré des résultats très encourageants, ce travail présente des limites qui ouvrent la voie à des perspectives de recherche intéressantes. La performance de notre système en pipeline est dépendante de chaque module, signifiant qu'une erreur de détection ne peut être rattrapée

par le classifieur. De plus, l'analyse a montré que les performances diminuent sur les classes d'insectes rares ou visuellement très similaires.

Pour les travaux futurs, plusieurs pistes peuvent être explorées :

- **Optimisation des performances :** Pour les classes les plus difficiles, des stratégies de *data augmentation* ciblées ou l'utilisation de techniques d'apprentissage par peu d'exemples (*few-shot learning*) pourraient être bénéfiques. De plus, la latence mobile pourrait être encore réduite via des techniques de quantification ou de distillation de modèles.
- **Extension des fonctionnalités :** Le système pourrait être enrichi en intégrant la reconnaissance de maladies végétales, la géolocalisation des infestations, ou encore des modules de recommandation de traitements pour en faire un outil d'aide à la décision encore plus complet.
- **Validation sur le terrain :** Une campagne de tests en conditions agricoles réelles et variées (météo, luminosité, types de cultures) serait une étape cruciale pour valider définitivement la robustesse de notre solution.

Acquis Personnels

La réalisation de ce projet a été une expérience d'apprentissage très riche sur plusieurs plans. Sur le plan **technique**, ce travail nous a permis de maîtriser l'ensemble du cycle de vie d'un projet de Deep Learning : de la recherche bibliographique à la mise en œuvre pratique. Nous avons acquis des compétences solides dans l'utilisation de frameworks comme **PyTorch**, dans la manipulation d'architectures complexes comme les **Transformers (ViT)** et les détecteurs **YOLO**, ainsi que dans les techniques de prétraitement et d'augmentation de données.

Sur le plan **méthodologique**, nous avons appris à structurer une démarche de recherche, à définir une problématique, à mener une analyse critique de l'existant, et à valider une approche de manière rigoureuse et scientifique. Enfin, ce projet a renforcé notre capacité à gérer un projet de A à Z, à surmonter les défis techniques et à communiquer nos résultats de manière claire et structurée.

Bibliographie

[1] S. Yu. Sinev. Coordinating the traditional and modern approaches in the systematics of insects. *Entomological Review*, 92(2) :154–161, 2012.

[2] Amal Al-Shahrani, Rana Alsaedi, Ameera Alfadli, Taif Alahmadi, Ohoud Alzubaidi, and Deema Alqthami. Automated recognition model for identifying harmful and harmless insects in crop management. In *Proceedings of the 2024 3rd International Conference on Creative Communication and Innovative Technology (ICCIT)*, pages 1–7, 2024.

[3] Xiangxin Wu, Cheng Zhan, Yu-Kun Lai, Ming-Ming Cheng, and Jian Yang. Ip102 : A large-scale benchmark dataset for insect pest recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8787–8796, 2019.

[4] L. Zhang, S. Sun, H. Zhao, Z. Li, and D. Li. Multi-species insect recognition method based on computer visions : Sustainable agricultural development. *Ecological Informatics*, 88 :103125, 2025.

[5] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[6] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss : Faster and better learning for bounding box regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 12993–13000, 2020.

[7] Yunjie Tian, Qixiang Ye, and David Doermann. Yolov12 : Attention-centric real-time object detectors. *arXiv preprint arXiv :2502.12524*, 2025.

[8] Dillon Reis, Jordan Kupec, Jacqueline Hong, and Ahmad Daoudi. Real-time flying object detection with yolov8. *arXiv preprint arXiv :2305.09972*, 2023.

[9] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. Fastvit : A fast hybrid vision transformer using structural reparameterization. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5785–5795, 2023.

[10] Basavaraj S Anami and Chetan V Sagarnal. A fusion of hand-crafted features and deep neural network for indoor scene classification. *Malaysian Journal of Computer Science*, 36(2) :193–207, 2023.

[11] Vinita Abhishek Gupta, MV Padmavati, Ravi R Saxena, Pawan Kumar Patnaik, and Raunak Kumar Tamrakar. A study on image processing techniques and deep learning techniques for insect identification. *Karbala International Journal of Modern Science*, 9(2) :16, 2023.

[12] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning : A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(7) :3523–3542, 2021.

[13] Mary Holland. Viceroy vs. monarch. `https://naturallycuriouswithmaryholland.wordpress.com/2018/08/18/viceroy-vs-monarch/`, 2018. Accessed : 2025-06-08.

[14] Aurore Avarguès-Weber, Déborah d'Amaro, Marisa Metzler, Verena Finke, David Baracchi, and Adrian G Dyer. Does holistic processing require a large brain ? insights from honeybees and wasps in fine visual recognition tasks. *Frontiers in Psychology*, 9 :1313, 2018.

[15] S. Yu. Sinev. Coordinating the traditional and modern approaches in the systematics of insects. *Entomological Review*, 92(2) :154–161, 2012.

[16] Yogesh Kumar, Ashwani Kumar Dubey, and Adityan Jothi. Pest detection using adaptive thresholding. In *Proc. of the 2017 International Conference on Computing, Communication and Automation (ICCCA)*, pages 42–47, Greater Noida, India, 2017. IEEE.

[17] Thenmozhi Kasinathan, Dakshayani Singaraju, and Srinivasulu Reddy Uyyala. Using modern machine learning techniques for crop pest classification and detection. *Information Processing in Agriculture*, 8 :446–457, October 2021. Published online : 8 October 2020, Open Access under CC BY-NC-ND license.

[18] Shilpa Itnal, Mathena Akhila, Syed Sha Noorulla Khadri, and Vanukuri Meher Sreemaiee. Pest detection using image processing. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 9(2) :1496–1499, 2019.

[19] Bino Sebastian V, A Unnikrishnan, and Kannan Balakrishnan. Gray level co-occurrence matrices : generalisation and some new features. *arXiv preprint arXiv :1205.4831*, 2012.

[20] Mehdi Ebadi Manaa. Detection of harmful insects based on gray-level co-occurrence matrix (glcm) in rural areas. Technical report, University of Babylon, 2016.

[21] P-T Yap, Raveendran Paramesran, and Seng-Huat Ong. Image analysis by krawtchouk moments. *IEEE Transactions on image processing*, 12(11) :1367–1377, 2003.

[22] Min Hao Ling, Tania Ivorra, Chong Chin Heo, April Hari Wardhana, Martin J. R. Hall, Siew Hwa Tan, Zulqarnain Mohamed, and Tsung Fei Khang. Machine learning analysis of wing venation patterns accurately identifies *sarcophagidae*, *calliphoridae* and *muscidae* fly species. *Medical and Veterinary Entomology*, 37(4) :767–781, 2023.

[23] MS Santhanambika, G Maheswari, N Valliammal, and G Sudhamathy. Leveraging machine learning and deep learning techniques for accurate classification of stored grain pests. *Indian Journal of Entomology*, pages 1–5, 2024.

[24] Thanh-Nghi Doan. Large-scale insect pest image classification. *Journal of Advances in Information Technology*, 14(2) :328–340, 2023. Manuscript received October 13, 2022 ; revised October 30, 2022 ; accepted December 19, 2022 ; published April 17, 2023.

[25] Masataka Fuchida, Thejus Pathmakumar, Rajesh Elara Mohan, Ning Tan, and Akio Nakamura. Vision-based perception and classification of mosquitoes using support vector machine. *Applied Sciences*, 7(1) :51, 2017.

[26] Chittathuru Himala Praharsha, Alwin Poulose, and Chetan Badgujar. Comprehensive investigation of machine learning and deep learning networks for identifying multispecies tomato insect images. *Sensors*, 24(23) :7858, 2024.

[27] Miron B Kursa and Witold R Rudnicki. Feature selection with the boruta package. *Journal of statistical software*, 36 :1–13, 2010.

[28] Jay I Myung, Daniel R Cavagnaro, and Mark A Pitt. A tutorial on adaptive design optimization. *Journal of mathematical psychology*, 57(3-4) :53–67, 2013.

[29] Jiajun Xu, Zelin Feng, Jian Tang, Shuhua Liu, Zhiping Ding, Jun Lyu, Qing Yao, and Baojun Yang. Improved random forest for the automatic identification of spodoptera frugiperda larval instar stages. *Agriculture*, 12(11) :1919, 2022.

[30] Wenyong Li, Tengfei Zheng, Zhankui Yang, Ming Li, Chuanheng Sun, and Xinting Yang. Classification and detection of insects from field images using deep learning for smart pest management : A systematic review. *Ecological Informatics*, 66 :101460, 2021.

[31] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553) :436–444, 2015.

[32] Aarti Bagdi. Data mining vs machine learning vs ai : Key differences. `https://www.softwaretestinghelp.com/data-mining-vs-machine-learning-vs-ai/`, 2023. Accessed : 2025-06-08.

[33] MK Gurucharan. Basic CNN Architecture : A Detailed Explanation of the 5 Layers in Convolutional Neural Networks. Blog post, upGrad, June 2025. Updated June 06 2025 ; 23min read.

[34] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-entropy loss functions : Theoretical analysis and applications. In *International conference on Machine learning*, pages 23803–23828. PMLR, 2023.

[35] Diederik P Kingma. Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*, 2014.

[36] Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 477–485, 2009.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[38] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words : Transformers for image recognition at scale. *arXiv preprint arXiv :2010.11929*, 2020.

[39] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Vision transformer. `https://d2l.ai/chapter_attention-mechanisms-and-transformers/vision-transformer.html`, 2021. Accessed : 2025-06-08.

[40] Viso.ai. Vision transformer (vit) : Overview, architecture, and applications. `https://viso.ai/deep-learning/vision-transformer-vit/`, 2023. Accessed : 2025-06-08.

[41] Xiangning Chen, Cho-Jui Hsieh, and Boqing Gong. When vision transformers outperform resnets without pre-training or strong data augmentations. *arXiv preprint arXiv :2106.01548*, 2021.

[42] GeeksforGeeks. Vision transformers vs. convolutional neural networks (cnns), October 2024. Publié approximativement 8 mois avant juin 2025.

[43] Asifullah Khan, Zunaira Rauf, Anabia Sohail, Abdul Rehman Khan, Hifsa Asif, Aqsa Asif, and Umair Farooq. A survey of the vision transformers and their cnn-transformer based variants. *Artificial Intelligence Review*, 56(Suppl 3) :2917–2970, 2023.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[46] G. Rohini. Everything you need to know about vgg16, September 2021. Consulté le 09 juin 2025.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[48] Nico Klingler. Efficientnet : Optimizing deep learning efficiency, March 2024. Consulté le 09 juin 2025.

[49] Hatim Master. Resnet – understanding the revolutionary neural network architecture for deep learning, September 2021. Consulté le 09 juin 2025.

[50] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[51] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[52] Mingxing Tan and Quoc V. Le. Efficientnet : Rethinking model scaling for convolutional neural networks, 2020.

[53] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10428–10436, 2020.

[54] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words : Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[55] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 2021.

[56] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer : Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[57] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets : Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv :1704.04861*, 2017.

[58] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2 : Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[59] Mingxing Tan and Quoc Le. Efficientnet : Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[60] Sachin Mehta and Mohammad Rastegari. Mobilevit : light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv :2110.02178*, 2021.

[61] Muhammad Maaz, Abdelrahman Shaker, Hisham Cholakkal, Salman Khan, Syed Waqas Zamir, Rao Muhammad Anwer, and Fahad Shahbaz Khan. Edgenext : efficiently amalgamated cnn-transformer architecture for mobile vision applications. In *European conference on computer vision*, pages 3–20. Springer, 2022.

[62] Yanyu Li, Geng Yuan, Yang Wen, Ju Hu, Georgios Evangelidis, Sergey Tulyakov, Yanzhi Wang, and Jian Ren. Efficientformer : Vision transformers at mobilenet speed. *Advances in Neural Information Processing Systems*, 35 :12934–12949, 2022.

[63] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. Mobileone : An improved one millisecond mobile backbone. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7907–7917, 2023.

[64] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet : More features from cheap operations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1580–1589, 2020.

[65] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2 : Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[66] SD Subburaj, Cowshik Eswaramoorthy, VISHNU GUNASEKARAN Latha, and RAKSHAN KAARTHI PALANISAMY Chinnasamy. Efficient pest detection through advanced machine learning technique. *Curr Agri Res*, 12(3), 2024.

[67] Edmond Maican, Adrian Iosif, and Sanda Maican. Precision corn pest detection : Two-step transfer learning for beetles (coleoptera) with mobilenet-ssd. *Agriculture*, 13(12) :2287, 2023.

[68] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn : Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[69] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd : Single shot multibox detector. In *European Conference on Computer Vision (ECCV)*, pages 21–37, 2016.

[70] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[71] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once : Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[72] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision (ECCV)*, 2020.

[73] Glenn Jocher, Ayush Chaurasia, and Jirka Qiu. Ultralytics yolov8. `https://github.com/ultralytics/ultralytics`, 2023. Accessed June 2025.

[74] Jacó C. Gomes and Díbio L. Borges. Insect pest image recognition : A few-shot machine learning approach including maturity stages classification. *Agronomy*, 12(8) :1733, 2022.

[75] Muhammad Arslan Ajmal. Mango pests dataset, 2022. Accessed : 2025-03-21.

[76] Tarun Dalal. Dangerous insects dataset, 2022. Accessed : 2025-03-21.

[77] Bing Liu, Luyang Liu, Ran Zhuo, Weidong Chen, Rui Duan, and Guishen Wang. A dataset for forestry pest identification. *Frontiers in Plant Science*, 13 :857104, 2022.

[78] Jin Wang, Yane Li, Hailin Feng, Lijin Ren, Xiaochen Du, and Jian Wu. Common pests image recognition based on deep convolutional neural network. *Computers and Electronics in Agriculture*, 179 :105834, 2020.

[79] Hieu T. Ung, Huy Q. Ung, and Binh T. Nguyen. An efficient insect pest classification using multiple convolutional neural network based models. *arXiv preprint arXiv :2107.12189v1*, 2021.

[80] K Kusrini, S Suputa, A Setyanto, IMA Agastya, H Priantoro, K Chandramouli, and E Izquierdo. Dataset for pest classification in mango farms from indonesia. *Mendeley Data*, 2020.

[81] Qi-Jin Wang, Sheng-Yu Zhang, Shi-Feng Dong, Guang-Cai Zhang, Jin Yang, Rui Li, and Hong-Qiang Wang. Pest24 : A large-scale very small object data set of agricultural pests for multi-target detection. *Computers and Electronics in Agriculture*, 175 :105585, 2020.

[82] Dan Popescu, Alexandru Dinca, Loretta Ichim, and Nicoleta Angelescu. New trends in detection of harmful insects and pests in modern agriculture using artificial neural networks. a review. *Frontiers in Plant Science*, 14 :1268167, 2023.

[83] Zhongbin Su, Jiaqi Luo, Yue Wang, Qingming Kong, and Baisheng Dai. Comparative study of ensemble models of deep convolutional neural networks for crop pests classification. *Multimedia Tools and Applications*, 82(19) :29567–29586, 2023.

[84] Niranjan C Kundur and PB Mallikarjuna. Insect pest image detection and classification using deep learning. *International Journal of Advanced Computer Science and Applications*, 13(9), 2022.

[85] Waleed Albattah, Momina Masood, Ali Javed, Marriam Nawaz, and Saleh Albahli. Custom cornernet : a drone-based improved deep learning technique for large-scale multiclass pest localization and classification. *Complex & Intelligent Systems*, 9(2) :1299–1316, 2023.

[86] Liu Liu, Rujing Wang, Chengjun Xie, Po Yang, Fangyuan Wang, Sud Sudirman, and Wancai Liu. Pestnet : An end-to-end deep learning approach for large-scale multi-class pest detection and classification. *Ieee Access*, 7 :45301–45312, 2019.

[87] Kai-Run Li, Li-Jun Duan, Yang-Jun Deng, Jin-Ling Liu, Chen-Feng Long, and Xing-Hui Zhu. Pest detection based on lightweight locality-aware faster r-cnn. *Agronomy*, 14(10) :2303, 2024.

[88] Jiyu Huang, Yong Huang, Hongliang Huang, Weirong Zhu, Jun Zhang, and Xiaolong Zhou. An improved yolox algorithm for forest insect pest detection. *Computational Intelligence and Neuroscience*, 2022(1) :5787554, 2022.

[89] Lijuan Zhang, Cuixing Zhao, Yuncong Feng, and Dongming Li. Pests identification of ip102 by yolov5 embedded with the novel lightweight module. *Agronomy*, 13(6) :1583, 2023.

[90] Jianjun Yin, Pengfei Huang, Deqin Xiao, and Bin Zhang. A lightweight rice pest detection algorithm using improved attention mechanism and yolov8. *Agriculture*, 14(7) :1052, 2024.

[91] Qi-Jin Wang, Sheng-Yu Zhang, Shi-Feng Dong, Guang-Cai Zhang, Jin Yang, Rui Li, and Hong-Qiang Wang. Pest24 : A large-scale very small object data set of agricultural pests for multi-target detection. *Computers and electronics in agriculture*, 175 :105585, 2020.

[92] Zhe Tang, Zhengyun Chen, Fang Qi, Lingyan Zhang, and Shuhong Chen. Pest-yolo : Deep image mining and multi-feature fusion for real-time agriculture pest detection. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 1348–1353. IEEE, 2021.

[93] Zhe Tang, Jiajia Lu, Zhengyun Chen, Fang Qi, and Lingyan Zhang. Improved pest-yolo : Real-time pest detection based on efficient channel attention mechanism and transformer encoder. *Ecological Informatics*, 78 :102340, 2023.

[94] Fang Qi, Yuxiang Wang, Zhe Tang, and Shuhong Chen. Real-time and effective detection of agricultural pest using an improved yolov5 network. *Journal of Real-Time Image Processing*, 20(2) :33, 2023.

[95] Min Dai, Md Mehedi Hassan Dorjoy, Hong Miao, and Shanwen Zhang. A new pest detection method based on improved yolov5m. *Insects*, 14(1) :54, 2023.

[96] Yong Zheng, Weiheng Zheng, and Xia Du. A lightweight rice pest detection algorithm based on improved yolov8. *Scientific Reports*, 14(1) :1–18, 2024.

[97] Guangbo Yue, Yaqiu Liu, Tong Niu, Lina Liu, Limin An, Zhengyuan Wang, and Mingyu Duan. Glu-yolov8 : An improved pest and disease target detection algorithm based on yolov8. *Forests*, 15(9) :1486, 2024.

[98] Xiao Chen, Xinting Yang, Huan Hu, Tianjun Li, Zijie Zhou, and Wenyong Li. Dami-yolov8l : A multi-scale detection framework for real-time and multi-class insect pests monitoring. *Available at SSRN 4845843*, 2025.

[99] Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. Detrs beat yolos on real-time object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16965–16974, 2024.

[100] Thenmozhi Kasinathan, Dakshayani Singaraju, and Srinivasulu Reddy Uyyala. Insect classification and detection in field crops using modern machine learning techniques. *Information Processing in Agriculture*, 8(3) :446–457, 2021.

[101] Zeba Anwar and Sarfaraz Masood. Exploring deep ensemble model for insect and pest detection from images. *Procedia Computer Science*, 218 :2328–2337, 2023.

[102] Chen Li, Tong Zhen, and Zhihui Li. Image classification of pests with residual neural network based on transfer learning. *Applied Sciences*, 12(9) :4356, 2022.

[103] Muhammad Qasim, Syed M Adnan Shah, Qamas Gul Khan Safi, Danish Mahmood, Adeel Iqbal, Ali Nauman, and Sung Won Kim. An adaptive features fusion convolutional neural network for multi-class agriculture pest detection. *Computers, Materials and Continua*, 83(3) :4429–4445, 2025.

[104] Yingshu Peng and Yi Wang. Cnn and transformer framework for insect pest classification. *Ecological Informatics*, 72 :101846, 2022.

[105] Jingmin An, Yong Du, Peng Hong, Lei Zhang, and Xiaogang Weng. Insect recognition based on complementary features from multiple views. *Scientific Reports*, 13(1) :2966, 2023.

[106] Enes Ayan, Hasan Erbay, and Fatih Varçın. Crop pest classification with a genetic algorithm-based weighted ensemble of deep convolutional neural networks. *Computers and Electronics in Agriculture*, 179 :105809, 2020.

[107] Fuxiang Feng, Hanlin Dong, Youmei Zhang, Yu Zhang, and Bin Li. Ms-aln : Multiscale attention learning network for pest recognition. *IEEE Access*, 10 :40888–40898, 2022.

[108] Shengyi Zhao, Jizhan Liu, Zongchun Bai, Chunhua Hu, and Yujie Jin. Crop pest recognition in real agricultural environment using convolutional neural networks by a parallel attention mechanism. *Frontiers in Plant Science*, 13 :839572, 2022.

[109] Yanan Chen, Miao Chen, Minghui Guo, Jianji Wang, and Nanning Zheng. Pest recognition based on multi-image feature localization and adaptive filtering fusion. *Frontiers in Plant Science*, 14 :1282212, 2023.

[110] Loris Nanni, Alessandro Manfè, Gianluca Maguolo, Alessandra Lumini, and Sheryl Brahnam. High performing ensemble of convolutional neural networks for insect pest image detection. *arXiv preprint arXiv :2108.12539*, 2021. Affiliations : 1 University of Padova, via Gradenigo 6, Padova 35131, Italy ; 2 DISI, University of Bologna, via dell'Università 50, Cesena, Italy ; 3 Missouri State University, 901 S National Ave, Springfield, MO 65897, USA.

[111] Cong Xu, Changqing Yu, Shanwen Zhang, and Xuqi Wang. Multi-scale convolution-capsule network for crop insect pest recognition. *Electronics*, 11(10) :1630, 2022.

[112] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2) :303–338, 2010.

[113] Rafia Khanam and Mohammad Hussain. A review of yolov12 : Attention-based enhancements vs. previous versions. *arXiv preprint arXiv :2504.11995*, 2025.

[114] Ramesh Sapkota, Rohit H. Cheppally, Anuj Sharda, and Manoj Karkee. Rf-detr object detection vs yolov12 : A study of transformer-based and cnn-based architectures for single-class and multi-class greenfruit detection in complex orchard environments under label ambiguity. *arXiv preprint arXiv :2504.13099*, 2025.

[115] Tri Dao, Daniel Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention : Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359, 2022.

[116] Xiangxiang Chu, Zhi Tian, Bo Zhang, Xinlong Wang, and Chunhua Shen. Conditional positional encodings for vision transformers. *arXiv preprint arXiv :2102.10882*, 2021.

[117] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

[118] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup : Beyond empirical risk minimization. *arXiv preprint arXiv :1710.09412*, 2017.

[119] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix : Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[120] paperswithcode.com. Colorjitter. Data augmentation method changing brightness, contrast, saturation and hue, 2025. PyTorch torchvision implementation ; listed in Papers With Code—Image Data Augmentation methods.

[121] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv :1711.05101*, 2017.

[122] Pavlos S Efraimidis. Weighted random sampling over data streams. *Algorithms, Probability, Networks, and Games : Scientific Papers and Essays Dedicated to Paul G. Spirakis on the Occasion of His 60th Birthday*, pages 183–195, 2015.

[123] Frank Klinker. Exponential moving average versus moving exponential average. *Mathematische Semesterberichte*, 58 :97–107, 2011.

[124] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv :1710.03740*, 2017.

[125] Python Software Foundation. The python tutorial. `https://www.python.org/doc/`, 2025. Consulté le 4 juin 2025.

[126] PyTorch Team. PyTorch official website. `https://pytorch.org/`. Consulté le 4 juin 2025.

[127] Torchvision Contributors. Torchvision : Datasets, transforms and models specific to computer vision. `https://docs.pytorch.org/vision/stable/index.html`, 2024. Accessed : 2025-06-18.

[128] Ross Wightman. Pytorch image models (timm). `https://timm.fast.ai/`, 2019. Accessed : 2025-06-18.

[129] Sandro Tosi. *Matplotlib for Python Developers*. Packt Publishing, Birmingham - Mumbai, 2009.

[130] Harshavardhan Seetha, Vimal Tiwari, Kartik Reddy Anugu, D.S. Makka, and D.R. Karnati. A gui based application for pdf processing tools using python & customtkinter. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 11(1) :1613–1618, 2023.

[131] PIL Contributors. Python imaging library (pil) documentation. `https://pillow.readthedocs.io/en/stable/`, 2025. Accessed : June 18, 2025.

[132] V7 Labs. Mean average precision (map) explained : precision, recall, ap and map in object detection. `https://www.v7labs.com/blog/mean-average-precision`, 2023. Consulté le 6 juin 2025.

[133] Jinfan Wei, He Gong, Shijun Li, Minghui You, Hang Zhu, Lingyun Ni, Lan Luo, Mengchao Chen, Hongli Chao, Jinghuan Hu, et al. Improving the accuracy of agricultural pest identification : Application of aec-yolov8n to large-scale pest datasets. *Agronomy*, 14(8) :1640, 2024.

[134] Shuai Yang, Ziyao Xing, Hengbin Wang, Xinrui Dong, Xiang Gao, Zhe Liu, Xiaodong Zhang, Shaoming Li, and Yuanyuan Zhao. Maize-yolo : a new high-precision and real-time method for maize pest detection. *Insects*, 14(3) :278, 2023.

[135] Jiaxin Song, Ke Cheng, Fei Chen, and Xuecheng Hua. Rdw-yolo : A deep learning framework for scalable agricultural pest monitoring and control. *Insects*, 16(5) :545, 2025.

[136] Labellerr. Evaluation of yolov12 : Exploring the capabilities of the new object detector, 2024. Consulté en juin 2025.

.