République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

---

Université A.Mira de Béjaïa
Faculté des Sciences Exactes
Département d'Informatique

Mémoire de Fin de Cycle

En vue de l'obtention du diplôme de :

Master Recherche en Informatique

Option : **Intelligence Artificielle**

Thème:

Détection et Classification des Champignons Comestibles et Toxiques à l'aide de Réseaux de Neurones Convolutionnels

---

Réalisé par : Mr.Mayen CHERFI

Encadrant : Mr.Mohamed KHAMMARI

Co-encadrante : Mme.Cilia AZNI

Soutenue le 17/09/2025 devant le jury composé de :

Présidente : Mme.Djamila BOULAHROUZ
Examinateur : Mr.Fatah BOUCHEBBAH
Examinateur : Mr.Achour ACHROUFENE
Examinateur : Mme.Soraya ALOUI

Année universitaire : 2024 – 2025

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui m'ont soutenues et accompagnées tout au long de la réalisation de ce mémoire.

Je remercie tout particulièrement mes encadrants, **Mme Cilia AZNI** et **M. Mohammed KHAMMARI**, pour leurs précieux conseils, qui m'ont guidés dans la bonne direction tout au long de ce travail.

Je souhaite également remercier les membres du jury pour le temps qu'ils consacreront à l'évaluation de ce mémoire et pour leurs remarques constructives.

Mes remerciements vont aussi à mes collègues et amis pour leur soutien moral et leurs encouragements tout au long de ce parcours.

Enfin, je remercie ma famille pour sa patience, sa compréhension et son soutien indéfectible, sans lesquels ce travail n'aurait pas été possible.

CHERFI Mayen

Résumé

Les champignons occupent une place importante dans les écosystèmes et l'alimentation humaine, mais leur ressemblance visuelle rend difficile la distinction entre espèces comestibles, toxiques et mortelles. Cette difficulté entraîne chaque année de nombreuses intoxications, soulignant la nécessité d'outils d'identification fiables. Les approches traditionnelles, reposant sur l'expertise humaine, demeurent limitées face à la complexité morphologique et à la variabilité des conditions de prise de vue. Dans ce travail, nous proposons une méthode hybride pour l'identification automatique des champignons, combinant à la fois des caractéristiques profondes issues de modèles CNN pré-entraînés tels qu'InceptionV3 et des descripteurs classiques (LBP, GLCM, histogramme HSV, moments colorimétriques). La segmentation des images est assurée par un modèle U-Net basé sur MobileNetV3 enrichi de modules de Coordinate Attention, afin d'isoler précisément le champignon de son arrière-plan. Les représentations ainsi extraites sont ensuite fusionnées et exploitées par un classificateur SVM, permettant de tirer parti de la complémentarité entre descripteurs traditionnels et caractéristiques profondes, et d'améliorer la robustesse de la classification face à la variabilité visuelle.

Mots-clés : champignons, classification, deep learning, CNN, machine learning

Table des matières

Introduction générale

Les champignons jouent un rôle essentiel dans les écosystèmes et la nutrition humaine, mais certaines espèces représentent un danger important pour la santé. En effet, chaque année, de nombreuses intoxications graves sont causées par la consommation accidentelle de champignons toxiques, souvent confondus avec des espèces comestibles en raison de leur forte ressemblance visuelle. L'identification fiable des champignons, en particulier en milieu naturel, est donc un enjeu crucial tant pour la sécurité alimentaire que pour la mycologie.

Les méthodes traditionnelles d'identification reposent sur l'expertise humaine et l'observation de critères morphologiques, une tâche qui peut s'avérer longue, complexe et sujette à des erreurs. C'est dans ce contexte que les avancées technologiques en vision par ordinateur, et plus précisément en apprentissage profond (deep learning), offrent des solutions prometteuses. En particulier, les réseaux de neurones convolutifs (CNN) ont montré leur efficacité dans l'analyse d'images complexes, notamment pour des tâches de classification et de détection d'objets.

Cependant, l'application de ces méthodes à la reconnaissance automatique des champignons soulève plusieurs défis. Les images capturées en conditions réelles présentent souvent des arrière-plans complexes, une grande variabilité d'apparence (éclairage, angle de vue, stade de croissance) ainsi qu'une diversité inter- et intra-espèces marquée.

Dans ce contexte, ce mémoire a pour objectif de développer un modèle performant pour la classification automatique des champignons à partir d'images, en combinant des techniques avancées de segmentation et de classification. L'enjeu principal est de permettre une identification fiable des champignons comestibles, toxiques et mortels, même dans des conditions visuelles complexes.

Notre approche repose sur une méthodologie hybride articulée en trois étapes complémentaires. Dans un premier temps, la segmentation du champignon est réalisée à l'aide d'un modèle U-Net, dont l'encodeur est basé sur MobileNetV3 et enrichi par des modules de *Coordinate Attention*, afin d'isoler précisément l'objet d'intérêt de son arrière-plan. Ensuite, un vecteur de caractéristiques est construit en combinant des descripteurs classiques (LBP, GLCM, histogramme HSV, moments colorimétriques) et des descripteurs profonds extraits à partir de modèles CNN pré-entraînés tels qu'InceptionV3. Enfin, la classification est assurée par un SVM, permettant d'exploiter efficacement la complémentarité entre ces représentations et d'améliorer la robustesse du système face à la variabilité visuelle des

champignons (comestibles, toxiques et mortels).

Nous avons élaboré un plan de travail structuré qui couvre l'ensemble du projet, depuis les fondements théoriques jusqu'à la mise en œuvre technique. Ce mémoire est organisé en quatre chapitres, décrits comme suit :

- **Chapitre 1 : Définitions et concepts de base**
  Ce chapitre présente une brève introduction à la mycologie et aux enjeux de la classification des champignons. Il aborde ensuite les bases des réseaux de neurones convolutifs (CNN) et du transfert d'apprentissage, en mentionnant quelques modèles entraînés sur ImageNet.

- **Chapitre 2 : État de l'art**
  Ce chapitre passe en revue les travaux récents sur la classification des champignons et les architectures CNN légères et performantes, en mettant l'accent sur les approches adaptées aux images naturelles.

- **Chapitre 3 : Approche proposé**
  Ce chapitre décrit en détail l'architecture proposée,ensuite on entame l'implémentation de notre système passant de la création du dataset jusqu'à l'évaluation et analyse des performances.

- **Chapitre 4 : Conclusion**
  Ce chapitre conclut le mémoire, discute les apports et des limites de notre approche et propose des axes d'amélioration pour des recherches futures.

1. Définitions et concepts de base

Ce chapitre introduit les fondements théoriques nécessaires à la compréhension de ce travail. Il débute par une présentation de la mycologie, discipline centrale du contexte applicatif de cette étude. Nous exposons ensuite les enjeux liés à l'identification des champignons, tant sur le plan de la santé publique que de la biodiversité. Enfin, les concepts clés de l'apprentissage profond sont introduits, avec un accent particulier sur les réseaux de neurones convolutifs (CNN), qui constituent la base de notre approche.

1.1 Introduction à la mycologie et aux enjeux de la classification des champignons

La mycologie, branche de la biologie consacrée à l'étude des champignons, s'intéresse à des organismes dont la diversité morphologique, écologique et biochimique est considérable. Les champignons visibles à l'œil nu, souvent appelés champignons supérieurs ou à fructifications développées, regroupent une large gamme d'espèces qui apparaissent principalement en milieu naturel, dans les forêts, les prairies ou les zones humides. Ces organismes jouent un rôle écologique fondamental dans les écosystèmes terrestres : ils interviennent dans la décomposition de la matière organique (saprotrophes), vivent en symbiose avec les plantes (mycorhizes), ou peuvent adopter un comportement parasite vis-à-vis d'autres organismes [34].

La biodiversité fongique est immense, bien que largement sous-estimée. Si environ 120 000 espèces de champignons ont été décrites à ce jour, les estimations récentes suggèrent qu'il pourrait en exister plusieurs millions dans le monde [10]. Cette richesse s'explique par leur adaptabilité, leur rôle fonctionnel varié et leur répartition géographique étendue. Néanmoins, la courte durée de vie des fructifications, la variabilité inter et intra-espèce, ainsi que la dépendance aux conditions environnementales rendent leur étude et leur identification particulièrement complexes.

D'un point de vue pratique, les champignons peuvent être classés selon leur comestibilité :

- **Les champignons comestibles**, très appréciés pour leur goût et leur valeur nutritionnelle, incluent des espèces comme les cèpes (*Boletus edulis*), les girolles (*Cantharellus cibarius*) ou les morilles (*Morchella esculenta*).

(a) Boletus edulis.[14]

(b) Morchella esculenta.[14]

(c) Cantharellus cibarius.[14]

Figure 1.1

- **Les champignons non comestibles** sont considérés comme sans danger pour la santé mais sont exclus de la consommation à cause de leur amertume, de leur texture ou de leur mauvaise odeur. Certaines espèces de russules ou de tricholomes en font partie.

- **Les champignons toxiques** peuvent provoquer des symptômes digestifs, neurologiques ou physiologiques. Par exemple, *Amanita muscaria* (amanite tue-mouches) provoque des hallucinations et troubles nerveux.

Figure 1.2: amanita muscaria.[14]

- **Les champignons mortels**, enfin, contiennent des toxines létales. *Amanita phalloides* (amanite phalloïde), responsable de la majorité des décès liés à la consommation de champignons, illustre bien les dangers encourus en cas d'erreur d'identification [39, 4].

Figure 1.3: amanita phalloids.[14]

L'identification des espèces de champignons repose traditionnellement sur l'observation de critères morphologiques visibles à l'œil nu, comme la forme et la couleur du chapeau, la structure des lamelles, du pied, la présence éventuelle d'un anneau ou d'une volve. Elle peut également nécessiter des observations microscopiques portant sur la forme des spores, la structure des hyphes ou la présence de cystides spécifiques [20]. Toutefois, cette méthode reste complexe et sujette à erreur. La grande variabilité liée à l'âge du champignon, aux conditions de croissance, ainsi que les ressemblances morphologiques entre certaines espèces, comme c'est le cas chez plusieurs *Amanita*, compliquent considérablement le processus d'identification [22].

Ces difficultés ont des conséquences concrètes. Elles augmentent les risques d'intoxications accidentelles chez les cueilleurs amateurs, mais elles freinent aussi les efforts scientifiques d'inventaire, de suivi écologique et de conservation de la biodiversité fongique. Pour répondre à ces limites, il devient nécessaire de développer des outils d'aide à la reconnaissance, capables d'assister les non-spécialistes, notamment dans des contextes où l'expertise humaine est absente ou insuffisante.

Dans ce cadre, l'intelligence artificielle offre aujourd'hui des perspectives intéressantes. En particulier, les avancées récentes en vision par ordinateur permettent d'automatiser l'analyse visuelle des champignons, en s'appuyant sur des photographies prises en conditions réelles. Ces approches ne visent pas à remplacer les mycologues, mais à fournir une assistance précieuse dans des situations pratiques : prévention des intoxications, appui à la cueillette responsable, ou encore éducation à la reconnaissance des espèces.

L'**apprentissage profond** (*deep learning*) et, en particulier, les **réseaux de neurones convolutifs** (CNN), sont largement utilisés pour la classification et la détection d'objets visuels [43, 44, 18, 2]. Ces modèles apprennent automatiquement les caractéristiques distinctives des espèces, y compris dans des images prises sous différents angles, avec des variations d'éclairage ou des arrière-plans complexes.

Dans la section suivante, nous présentons les fondements de l'apprentissage profond et des réseaux de neurones convolutifs, avant de passer en revue les travaux existants portant sur l'identification automatique des champignons.

1.2 Apprentissage profond (Deep Learning)

L'apprentissage profond, ou *deep learning*, est une sous-branche de l'apprentissage automatique (*machine learning*) qui repose sur l'utilisation de réseaux de neurones artificiels comportant plusieurs couches successives, appelés réseaux de neurones profonds (*deep neural networks*) [21]. Contrairement aux méthodes classiques d'apprentissage supervisé qui nécessitent une extraction manuelle des caractéristiques, le deep learning permet au modèle d'apprendre directement des représentations à partir des données brutes [21].

Ces architectures hiérarchiques sont capables de modéliser des relations complexes en

extrayant automatiquement des caractéristiques de plus en plus abstraites au fur et à mesure que l'information traverse les différentes couches du réseau. Par exemple, dans le cas d'une image, les premières couches peuvent détecter des motifs simples (bords, textures), tandis que les couches intermédiaires repèrent des formes plus complexes (objets, structures), et les couches finales apprennent à reconnaître des entités spécifiques (visages, lettres, espèces de champignons, etc.)[21].

Le succès de l'apprentissage profond repose en grande partie sur trois éléments clés [6] :

- la disponibilité de grandes quantités de données annotées,
- des architectures de réseaux de plus en plus performantes,
- la puissance de calcul des processeurs graphiques (GPU), permettant l'entraînement efficace de ces modèles à grande échelle.

L'apprentissage profond s'est imposé comme la méthode de référence pour de nombreuses tâches liées à la vision par ordinateur, telles que [6]:

- la classification d'images (par exemple, reconnaître l'espèce d'un champignon à partir d'une photo),
- la détection d'objets (localiser et identifier plusieurs champignons dans une même image),
- la segmentation d'images (distinguer précisément les contours d'un champignon par rapport à l'arrière-plan).

Parmi les différentes architectures de réseaux de neurones utilisés en vision par ordinateur, les **réseaux de neurones convolutifs** (CNN, pour *Convolutional Neural Networks*) occupent une place centrale. Conçus pour traiter des données visuelles, ces réseaux exploitent des opérations de convolution pour extraire automatiquement les motifs spatiaux présents dans les images. Grâce à leur capacité à capturer efficacement les structures locales, les CNN ont révolutionné le traitement d'images et constituent la base de la majorité des modèles modernes en vision par ordinateur[7].

Dans la section suivante, nous détaillons le fonctionnement des réseaux convolutifs, leur architecture, ainsi que leur application dans le domaine de la reconnaissance visuelle des champignons.

1.3 Réseaux de Neurones Convolutifs (CNN)

Les réseaux de neurones convolutifs (*Convolutional Neural Networks*, CNN) sont des architectures spécifiques du deep learning conçues pour traiter les données ayant une structure en grille, comme les images. Les CNN exploitent la corrélation spatiale des pixels

à travers des opérations de convolution, ce qui permet d'extraire automatiquement les caractéristiques locales importantes. [30]

1.3.1 Composants d'un CNN

Un CNN se compose généralement de plusieurs types de couches :

- **Couches convolutives (Convolutional layers)** : Ces couches réalisent une opération appelée convolution qui consiste à appliquer un ensemble de filtres (ou noyaux) sur l'image d'entrée. Chaque filtre est une petite matrice de poids qui glisse sur l'image, calcule un produit scalaire entre ses poids et la région locale de l'image, et génère ainsi une valeur unique. En répétant cette opération sur toute l'image, on obtient une carte de caractéristiques (feature map) qui met en évidence la présence d'un motif particulier (comme un bord, une texture ou une forme). L'apprentissage permet d'ajuster les poids des filtres pour détecter les caractéristiques les plus pertinentes. Plusieurs filtres sont utilisés simultanément pour extraire diverses caractéristiques locales, ce qui enrichit la représentation de l'image.[30]

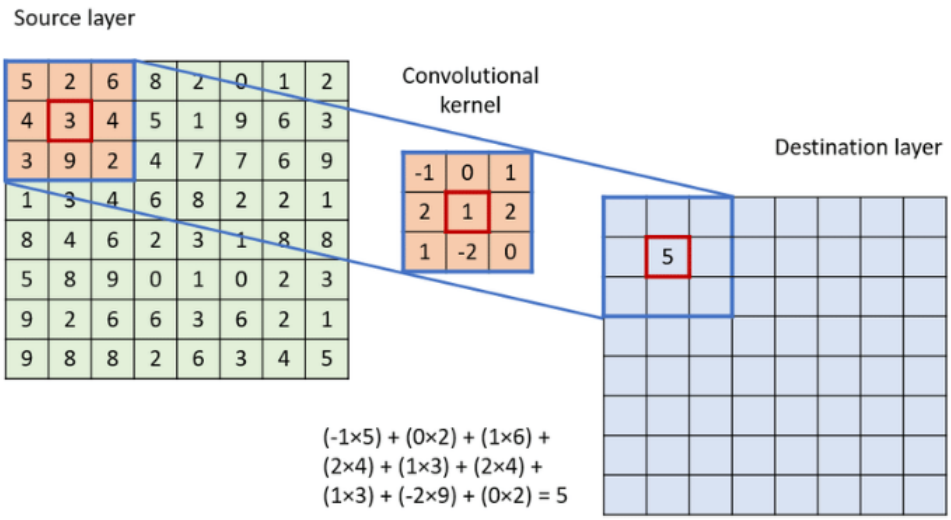


Figure 1.4: Exemple d'une convolution[28]

- **Couches de pooling (Pooling layers)** : Ces couches ont pour rôle de réduire la taille spatiale des cartes de caractéristiques produites par les couches convolutives, ce qui diminue la complexité computationnelle et la sensibilité du modèle aux translations et déformations locales. La méthode la plus répandue est le *max pooling*, qui divise la carte en sous-régions (par exemple, 2x2 pixels) et conserve uniquement la valeur maximale de chaque région. Cela permet de garder les informations les plus saillantes tout en réduisant la dimension des données. D'autres méthodes, comme le *average pooling*, calculent la moyenne locale mais sont moins utilisées pour la détection de motifs précis.[30]

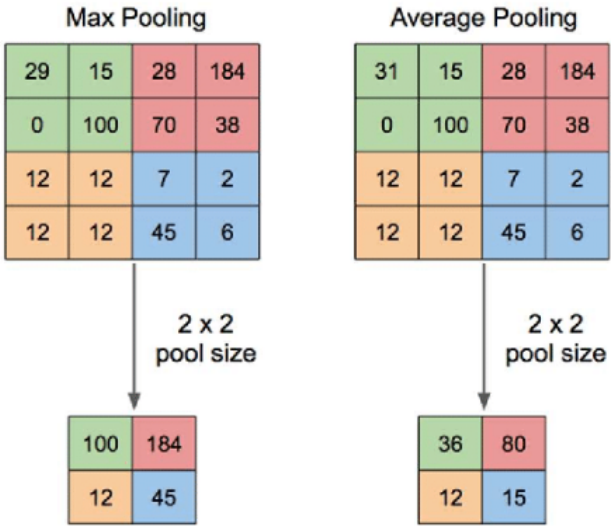


Figure 1.5: max et average pooling[42]

- **Couches de Flatten et Global Average Pooling (GAP)** : Avant de passer aux couches entièrement connectées, il est courant de transformer les cartes de caractéristiques issues des couches convolutives en un vecteur unidimensionnel. La couche **Flatten** effectue cette conversion en concaténant toutes les valeurs spatiales des cartes de caractéristiques en un long vecteur. Cette représentation vectorielle peut ensuite être utilisée comme entrée d'un perceptron entièrement connecté pour la classification.

  Une alternative à Flatten est la **Global Average Pooling (GAP)**. Au lieu de concaténer toutes les valeurs, GAP calcule la moyenne de chaque carte de caractéristiques, produisant un vecteur dont la dimension est égale au nombre de cartes. Cette approche réduit fortement le nombre de paramètres et limite le risque de surapprentissage, tout en conservant l'information sémantique globale présente dans chaque carte.

- **Couches entièrement connectées (Fully connected layers)** : Ces couches sont situées en fin de réseau et servent à interpréter les caractéristiques extraites par les couches précédentes pour effectuer la tâche finale, souvent la classification. Chaque neurone de ces couches est connecté à tous les neurones de la couche précédente, ce qui permet de combiner les différentes caractéristiques détectées pour produire une prédiction globale. Elles fonctionnent comme un classificateur non linéaire qui associe des motifs appris à des classes spécifiques.[30]

- **Fonctions d'activation** : Les fonctions d'activation introduisent la non-linéarité dans les réseaux de neurones, ce qui est indispensable pour apprendre des relations complexes entre les données. Sans ces fonctions, le réseau serait simplement une composition linéaire et incapable de modéliser des structures complexes.

  Parmi les fonctions couramment utilisées :

– *ReLU (Rectified Linear Unit)[30]* est populaire pour sa simplicité et son efficacité. Elle transmet uniquement les valeurs positives :

$$\text{ReLU}(x) = \max(0, x) \tag{1.1}$$

– *Hardswish[13]* est une variante plus récente du Swish, conçue pour une meilleure efficacité sur des dispositifs mobiles, tout en conservant les avantages des activations lissées. Sa définition est :

$$\text{Hardswish}(x) = x \cdot \frac{\text{ReLU6}(x+3)}{6} \tag{1.2}$$

où

$$\text{ReLU6}(x) = \min(\max(0, x), 6) \tag{1.3}$$

Ces fonctions sont particulièrement adaptées aux architectures modernes optimisées pour les plateformes à ressources limitées comme les smartphones [30].

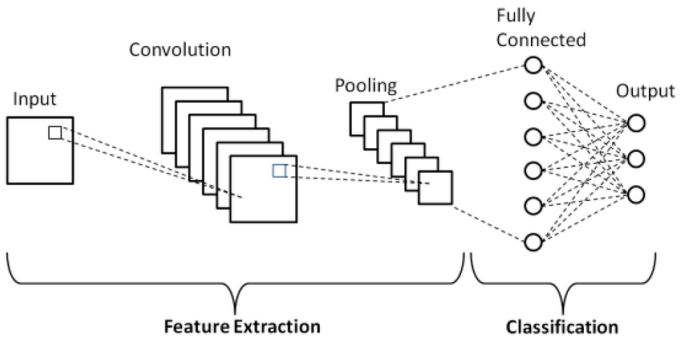


Figure 1.6: Architecture basic d'un CNN[27].

Après avoir présenté la structure des CNN, nous décrivons maintenant les principales techniques permettant d'optimiser leur performance.

1.4 Techniques d'optimisation

L'entraînement des réseaux de neurones profonds, comme les CNN, nécessite des techniques spécifiques pour améliorer la généralisation, éviter le surapprentissage (*overfitting*) et accélérer la convergence. Cette section présente quelques-unes des méthodes couramment utilisées.

1.4.1 Régularisation

La régularisation vise à limiter la capacité d'un modèle à trop bien s'adapter aux données d'entraînement, afin d'améliorer ses performances sur des données inconnues. Voici les principales approches :

- **Régularisation L1 et L2** : Ces techniques ajoutent un terme pénalisant à la fonction de perte pour contrôler la magnitude des poids du réseau. La régularisation L2, aussi appelée *weight decay*, est la plus utilisée et favorise des poids petits et distribués [6].

- **Dropout** : Proposé par Srivastava et al., le *dropout* consiste à désactiver aléatoirement un certain pourcentage de neurones pendant l'entraînement. Cela empêche le réseau de devenir trop dépendant de certains neurones et améliore la robustesse du modèle [36].

- **Early stopping** : Cette méthode surveille la performance sur un jeu de validation et arrête l'entraînement lorsque celle-ci cesse de s'améliorer, afin d'éviter l'overfitting [29].

1.4.2 Normalisation

Batch Normalization (BN) est une technique introduite par Ioffe et Szegedy qui permet d'accélérer l'entraînement et de stabiliser l'optimisation. Elle normalise les activations d'une couche en soustrayant la moyenne et en divisant par l'écart type, tout en apprenant des paramètres de redimensionnement [15].

1.4.3 Fonctions de perte

La fonction de perte guide l'apprentissage du modèle. Pour une tâche de classification binaire, la **binary cross-entropy**[6] est généralement utilisée. Elle mesure l'écart entre les sorties du modèle (probabilités) et les vraies étiquettes, et s'exprime par :

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

où y_i est l'étiquette réelle et $\hat{y}_i$ la prédiction du modèle.

1.4.4 Optimisation

Les algorithmes d'optimisation ajustent les poids du réseau à chaque itération pour minimiser la fonction de perte.

- **Stochastic Gradient Descent (SGD)** : Méthode de base qui utilise un sous-ensemble aléatoire des données (mini-batch) pour calculer le gradient et mettre à jour les poids[6].

- **Adam** : Introduit par Kingma et Ba, Adam est un optimiseur adaptatif combinant les avantages du *momentum* et de la normalisation du gradient. Il est populaire pour sa rapidité de convergence et sa stabilité [19].

1.4.5 Data Augmentation

L'**augmentation de données** consiste à générer artificiellement de nouvelles images à partir de celles existantes pour enrichir l'ensemble d'apprentissage. Cela inclut :

- Des transformations géométriques : rotations, translations, zooms, inversions horizontales, etc.
- Des variations photométriques : ajustements de la luminosité, du contraste ou du bruit.

Cette technique permet d'augmenter la robustesse du modèle et de limiter le surapprentissage, surtout quand le jeu de données est restreint [32].

1.5 Apprentissage par transfert (Transfer Learning)

L'apprentissage par transfert (*Transfer Learning*) est une technique puissante en apprentissage automatique qui consiste à utiliser un modèle préalablement entraîné sur une grande quantité de données pour résoudre une nouvelle tâche similaire ou connexe. Plutôt que d'entraîner un modèle à partir de zéro, cette approche permet de tirer parti des connaissances déjà acquises par le modèle pré-entraîné, notamment des représentations pertinentes extraites des données.[26]

Cette méthode est particulièrement utile lorsque les données disponibles pour la nouvelle tâche sont limitées, ce qui est souvent le cas dans de nombreux domaines, notamment en médecine ou en sciences naturelles. En effet, entraîner un modèle profond nécessite généralement un grand volume de données annotées, coûteuses à collecter et à labelliser.[26]

Le transfert peut se faire de différentes manières :

- **Fine-tuning (affinage)** : On réutilise un modèle pré-entraîné et on continue son entraînement sur la nouvelle base de données avec un taux d'apprentissage faible. Certaines couches peuvent être gelées (non modifiées) tandis que d'autres sont ajustées pour mieux s'adapter à la nouvelle tâche.[26]
- **Extraction de caractéristiques** : Le modèle pré-entraîné est utilisé comme un extracteur de caractéristiques fixes. Seules les dernières couches (typiquement des couches entièrement connectées) sont entraînées sur la nouvelle tâche, tandis que les couches convolutives restent inchangées.[26]

Les modèles pré-entraînés utilisés proviennent souvent de bases de données très larges et diversifiées, comme *ImageNet*, qui contient plus d'un million d'images réparties en mille classes. Ces modèles ont ainsi appris à détecter des caractéristiques visuelles fondamentales (bords, textures, formes, etc.) qui sont souvent transférables à d'autres domaines.[26]

En résumé, l'apprentissage par transfert permet de :

- Réduire les besoins en données annotées pour la nouvelle tâche.
- Accélérer la phase d'entraînement.
- Améliorer les performances sur des tâches connexes.

Cette méthode est devenue incontournable pour de nombreux projets de classification d'images, notamment lorsque les ressources sont limitées ou que le domaine d'application est spécialisé, comme dans la classification automatique des champignons étudiée dans ce mémoire.

1.6 Exemples de modèles pré-entraînés

Dans le cadre du transfert learning, plusieurs architectures convolutives ont été développées et entraînées sur de grands ensembles de données, notamment ImageNet [31]. Ces modèles apprennent des représentations visuelles riches et peuvent être adaptés à d'autres tâches avec peu de données supplémentaires. Nous présentons ci-dessous quatre modèles couramment utilisés dans ce contexte.

1.6.1 VGG16

VGG16 [33] est une architecture de réseau convolutif profond développée par le Visual Geometry Group (VGG) de l'Université d'Oxford. Elle s'est imposée comme l'une des architectures de référence en vision par ordinateur, notamment grâce à sa simplicité conceptuelle et son efficacité démontrée lors du concours ILSVRC 2014.

L'architecture VGG16 repose sur deux principes majeurs :

- **L'utilisation systématique de convolutions** 3×3 : contrairement à des modèles plus complexes qui varient la taille des filtres, VGG16 emploie uniquement de petites convolutions 3×3, empilées pour capturer des représentations hiérarchiques de plus en plus abstraites. Cette approche permet d'augmenter la profondeur du réseau tout en limitant le nombre de paramètres par couche.
- **Une grande profondeur contrôlée** : le modèle est constitué de 16 couches pondérées (13 convolutions et 3 couches entièrement connectées), organisées en blocs séparés par des opérations de *max-pooling*. Cette profondeur favorise l'apprentissage de caractéristiques visuelles riches et discriminantes.

Le schéma global suit une progression régulière : des convolutions appliquées sur des cartes de petites résolutions sont suivies par un sous-échantillonnage (*max-pooling*), jusqu'à atteindre des représentations compactes qui alimentent les couches entièrement connectées. La dernière couche est une *softmax* à 1000 neurones, correspondant aux classes du jeu de données ImageNet.

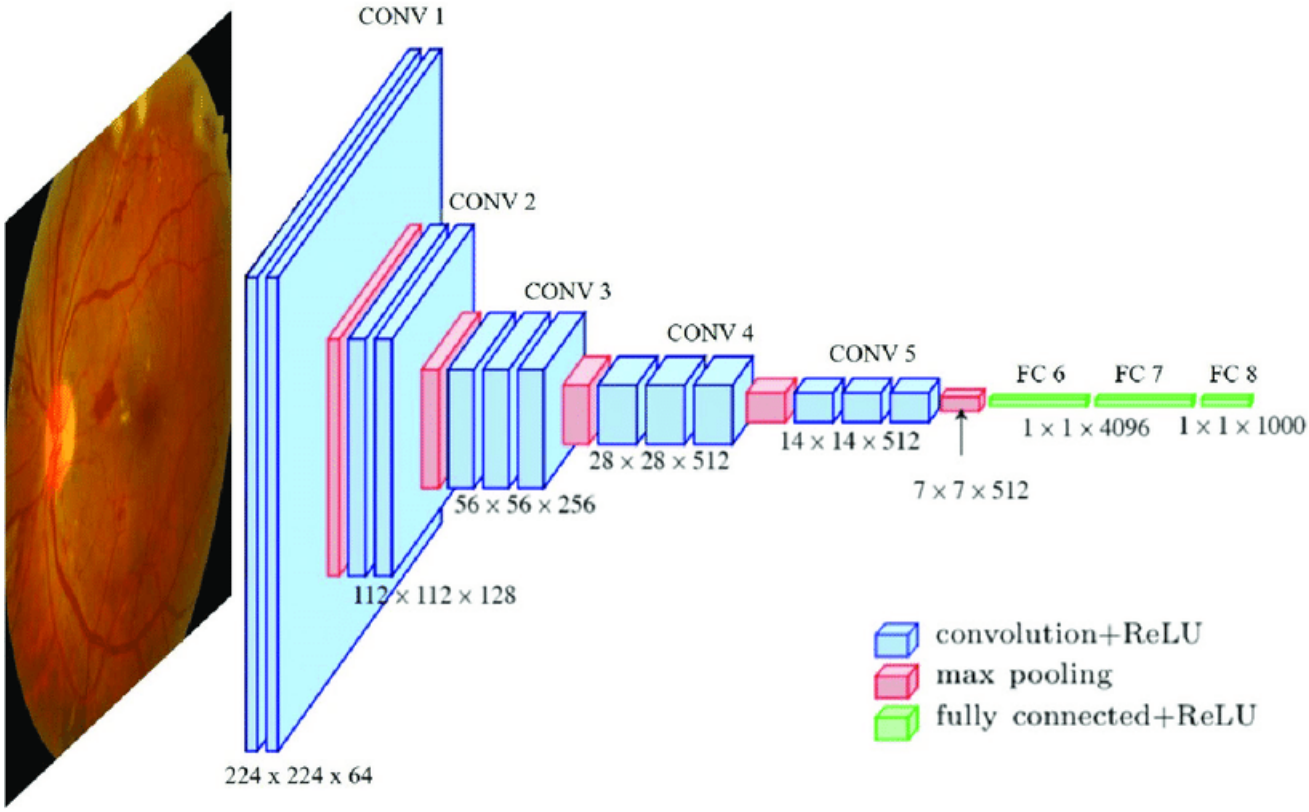


Figure 1.7: architecture VGG 16 .

1.6.2 ResNet50

ResNet50 [11] est une architecture de réseaux de neurones convolutifs introduite pour répondre au problème de la dégradation des performances observé lors de l'entraînement de réseaux très profonds. L'innovation principale repose sur l'introduction des **connexions résiduelles**, qui permettent d'entraîner efficacement des modèles comptant plusieurs dizaines, voire centaines de couches.

Principes clés de son architecture :

- **Blocs résiduels (Residual Blocks)**:chaque bloc apprend une fonction résiduelle de la forme :

$$y = F(x, \{W_i\}) + x$$

où x est l'entrée, $F(x, \{W_i\})$ représente les transformations convolutives, et la connexion directe (*skip connection*) permet de réinjecter l'information d'entrée sans altération. Cela facilite l'entraînement de réseaux très profonds en réduisant le risque de disparition du gradient.

- **Bottleneck blocks**:ResNet50 utilise des blocs optimisés comportant trois convolutions successives :
  - Convolution 1×1 pour réduire la dimensionnalité,
  - Convolution 3×3 pour l'extraction de caractéristiques spatiales,
  - Convolution 1×1 pour restaurer la dimension initiale.

Cette conception permet de diminuer le nombre de paramètres et les coûts de calcul.

- **Profondeur de 50 couches** : composée de 49 couches convolutives et d'une couche entièrement connectée, l'architecture suit la structure globale :
  - Convolution initiale + max pooling,
  - 4 étapes principales constituées de plusieurs blocs résiduels bottleneck,
  - Global Average Pooling,
  - Couche entièrement connectée avec softmax.
- **Avantage principal** : grâce aux connexions résiduelles, ResNet50 surmonte les difficultés liées à l'augmentation de profondeur et atteint de très hautes performances sur ImageNet, devenant un standard pour le transfert d'apprentissage.

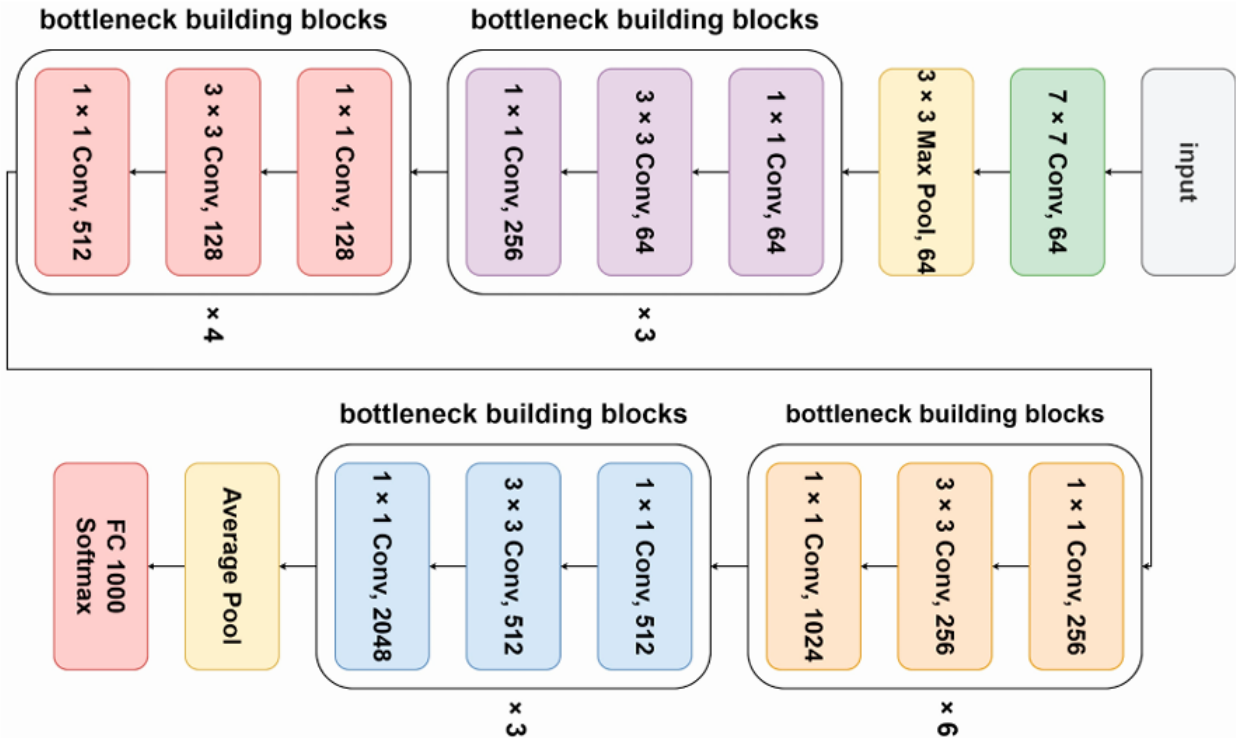


Figure 1.8: architecture Resnet50 [18].

1.6.3 InceptionV3

InceptionV3 [38] fait partie de la famille GoogLeNet. Son architecture repose sur des **modules Inception** combinant plusieurs convolutions (1×1, 3×3, 5×5) et du pooling en parallèle.

Les innovations principales sont :

- Factorisation des convolutions (ex : 5×5 en deux 3×3).
- Réduction du nombre de paramètres tout en améliorant les performances.

- Ajout de *auxiliary classifiers* pour régulariser l'apprentissage.

Ce modèle, entraîné sur ImageNet, est souvent utilisé pour des tâches nécessitant une grande précision et une extraction multi-échelle des caractéristiques.

1.6.4 Conclusion

Les architectures pré-entraînées telles que VGG16, ResNet50 et InceptionV3 représentent des approches complémentaires pour le transfert learning, chacune apportant des atouts spécifiques en termes de profondeur, de capacité de représentation et d'efficacité.

VGG16, bien que relativement coûteuse en calcul et en mémoire, demeure une architecture de référence par sa simplicité et son efficacité, offrant des performances solides dans de nombreux contextes. ResNet50 introduit les connexions résiduelles, qui permettent d'entraîner des réseaux beaucoup plus profonds tout en limitant le problème de dégradation des performances, ce qui en fait un modèle robuste et performant. InceptionV3, de son côté, exploite des stratégies avancées de factorisation et de régularisation afin de concilier profondeur et efficacité computationnelle, et se distingue par sa précision élevée sur des tâches complexes de reconnaissance visuelle.

Ainsi, le choix d'un modèle pré-entraîné dépend fortement des contraintes de l'application cible : ressources disponibles, rapidité d'inférence et niveau de précision attendu.

2. État de l'art

Introduction

L'identification automatique des champignons comestibles, toxiques et mortels constitue un enjeu majeur, à la fois pour la sécurité alimentaire et pour la préservation de la biodiversité. Ces dernières années, les progrès en vision par ordinateur et en apprentissage automatique ont permis le développement de systèmes capables de reconnaître efficacement différentes espèces fongiques.

Dans ce chapitre, nous présentons certaines des techniques utilisées dans les travaux de classification des champignons. Nous commençons par les méthodes classiques d'apprentissage automatique, basées sur l'extraction manuelle de caractéristiques et des algorithmes traditionnels de classification. Nous abordons ensuite les approches plus récentes de deep learning, qui permettent un apprentissage automatique des représentations visuelles à partir des images brutes.

2.1 Techniques classiques

La classification automatique des champignons repose historiquement sur l'utilisation de **techniques classiques de vision par ordinateur et d'apprentissage supervisé**. Ces approches consistent à extraire manuellement des **descripteurs caractéristiques** à partir des images ou de mesures morphologiques (couleur, texture, forme, dimensions, etc.), puis à exploiter ces caractéristiques comme vecteurs d'entrée pour des modèles de **machine learning** classiques. Parmi les classifieurs les plus couramment utilisés figurent le **Support Vector Machine (SVM)**, le **K-Nearest Neighbors (KNN)**, les **arbres de décision** et le **Random Forest**. Ces techniques offrent une bonne capacité de discrimination lorsqu'elles sont combinées à des descripteurs pertinents. Nous présentons ci-après deux études récentes qui ont appliqué ces méthodes pour la classification des champignons.

2.1.1 Mushroom Classification Using Feature-Based Machine Learning Approach [24]

Maurya et Singh [24] ont proposé une approche de classification des champignons basée sur des **descripteurs de texture** extraits manuellement, tels que le **Local Binary Pattern (LBP)** et la **matrice de cooccurrence de niveaux de gris (GLCM)**. Ces descripteurs sont utilisés comme caractéristiques pour plusieurs modèles classiques de machine learning, notamment le **SVM**, le **KNN**, l'arbre de décision, la régression logistique et le Random Forest. Les résultats montrent que le **SVM** obtient la meilleure précision avec 76,6 %, démontrant l'efficacité de la combinaison des descripteurs de texture et des modèles classiques pour distinguer les champignons comestibles des toxiques.

2.1.2 Mushroom Classification Using Machine Learning Techniques [25]

Ottom et al [25] ont développé une méthode de classification des champignons en deux classes (poisonneux et non poisonneux) en combinant plusieurs types de **descripteurs** extraits à partir d'images et de mesures morphologiques. Les caractéristiques utilisées incluent les **Eigen features**, les **dimensions réelles** (diamètre du chapeau, hauteur et diamètre de la tige), les **dimensions virtuelles** estimées à partir des images, ainsi que des **descripteurs paramétriques et histogrammes**. Ces données ont ensuite été exploitées par différents modèles de machine learning, tels que le **KNN**, le **SVM**, les **arbres de décision** et les **réseaux de neurones**. Les résultats expérimentaux montrent que le **KNN** obtient la meilleure performance avec une précision de 94,4 % lorsque les Eigen features sont combinées aux dimensions réelles, confirmant l'importance des descripteurs morphologiques et visuels pour la classification fiable des champignons.

2.2 Techniques utilisant l'apprentisage profond

Avec l'essor du deep learning, la classification automatique des champignons s'est enrichie de méthodes capables d'apprendre directement des caractéristiques visuelles à partir des images, sans nécessiter d'extraction manuelle. Ces approches reposent principalement sur des **réseaux de neurones convolutifs (CNN)**, qui permettent de capturer des informations complexes de couleur, de texture et de forme. Contrairement aux méthodes classiques, les CNN peuvent être utilisés tels quels via le **transfert d'apprentissage**, entraînés à partir de zéro sur des architectures optimisées, ou combinés à des techniques avancées de prétraitement comme la segmentation d'image. Nous présentons ci-après plusieurs études illustrant ces différentes approches appliquées à la classification et à l'évaluation de champignons.

2.2.1 Using deep convolutional neural networks to classify poisonous and edible mushrooms found in china[43]

Zhang et al[43] proposent un système automatique pour distinguer les champignons comestibles des champignons toxiques à partir de 450 images (250 toxiques, 200 comestibles). Les images ont été prétraitées (redimensionnement, recadrage, augmentation et normalisation) et utilisées pour entraîner des modèles CNN préentraînés (**AlexNet**, **VGG16**, **DenseNet121**, **ResNet50**) adaptés à la classification binaire.

L'entraînement s'est fait avec **SGD**, une **binary cross-entropy** et un **early stopping** sur 500 époques. **ResNet50** a obtenu les meilleures performances avec une **précision de 0,756** et un **F1-score de 0,766**.

Malgré la taille limitée du jeu de données, l'étude montre que les CNN pré-entraînés peuvent fournir un système d'aide à la classification efficace. Les auteurs recommandent d'augmenter le volume de données et de tester des architectures plus récentes pour améliorer la généralisation.

2.2.2 Deep learning based approach for classification of mushrooms. [2]

Demirel et al [2] proposent une approche fondée sur le modèle *MobileNetV2-GAP-flatten-fc*, une version légère et performante de MobileNetV2, adaptée aux environnements à ressources limitées. Le jeu de données comprend 5470 images réparties en quatre classes de champignons, prétraitées par redimensionnement et augmentation pour améliorer la généralisation.

Le modèle, entraîné de zéro avec l'optimiseur Adam sur 100 époques, combine *Global Average Pooling*, aplatissement et couche entièrement connectée pour la classification finale. Il atteint une précision moyenne de 0,977, un rappel de 0,979 et un F1-score de 0,978, surpassant plusieurs CNN pré-entraînés tout en restant peu coûteux en calcul.

L'étude montre l'efficacité de cette approche pour la classification automatique de champignons, mais sa généralisation reste limitée par le faible nombre d'espèces et l'absence de tests en conditions réelles. Les auteurs recommandent d'étendre le jeu de données pour améliorer la robustesse.

2.2.3 A new deep learning model for the classification of poisonous and edible mushrooms based on improved alexnet convolutional neural network [18]

Ketwongsa et al [18] proposent un modèle basé sur une version améliorée d'AlexNet intégrant un module Inception pour optimiser l'extraction des caractéristiques et réduire le temps d'apprentissage. Le réseau combine trois couches convolutionnelles, un module Inception, puis trois couches entièrement connectées.

L'entraînement a été effectué sur 2000 images générées à partir de 623 originales par augmentation de données. Le modèle atteint une précision de 0,985 pour la classification et de 0,955 pour la détection, tout en étant plus rapide que des architectures plus profondes comme ResNet-50 ou GoogLeNet.

Cette approche illustre un compromis efficace entre précision et coût computationnel. L'étude souligne cependant la nécessité de tester le modèle sur des jeux de données plus diversifiés pour améliorer sa généralisation.

2.2.4 Using deep learning for prediction of edible and poisonous mushrooms [23]

Mahanta et al [23] proposent une méthode combinant segmentation d'image et deep learning pour classifier les champignons comestibles et toxiques. Le jeu de données Danish Fungi 2018a été filtré via U2Net pour supprimer les arrière-plans, puis des régions d'intérêt ont été extraites avec OpenCV.

Quatorze espèces ont été retenues (7 comestibles, 7 toxiques), et les images ont été augmentées pour obtenir 192 images par espèce. La classification repose sur InceptionV3 pré-entraîné sur ImageNet, avec une couche dense de 128 neurones et une sortie sigmoïde. L'entraînement a utilisé SGD, avec 16 images par espèce réservées au test. Le modèle atteint une précision moyenne de 0,625 pour les comestibles et de 0,854 pour les toxiques.

Cette étude montre l'efficacité de l'approche combinant segmentation et deep learning, tout en soulignant que la précision reste limitée pour les espèces comestibles à cause du jeu de données restreint.

2.2.5 An improved mobilenetv3 mushroom quality classification model using images with complex backgrounds[44]

Zhu et al [44] présentent un modèle amélioré de MobileNetV3-Large pour classifier la qualité visuelle des champignons *shiitake* en trois niveaux. Le jeu de données initial (275 images) a été segmenté pour supprimer l'arrière-plan et augmenté à 10991 images pour améliorer la généralisation.

Le modèle intègre des améliorations telles que le remplacement du module SE par des couches entièrement connectées, un ajustement du facteur de compression des canaux et l'usage de `PolyFocalLoss` pour gérer le déséquilibre des classes. Il atteint une précision de 0,9991 pour seulement 11,9Mo, avec des visualisations Grad-CAM confirmant la pertinence des zones analysées.

Malgré ces résultats, le modèle reste limité à une seule variété de champignons et n'a pas été testé sur des jeux de données externes, ce qui restreint sa généralisation et sa robustesse en conditions réelles.

En résumé, les approches classiques basées sur des descripteurs manuels et les techniques récentes de deep learning montrent des performances intéressantes mais avec des limites propres à chaque méthode. Les premières dépendent fortement du choix et de la qualité des descripteurs, tandis que les secondes nécessitent de larges jeux de données diversifiés pour assurer une bonne généralisation. Afin de mieux comparer ces travaux, nous présentons ci-après un tableau récapitulatif des principales études analysées.

Synthèse des articles

Table 2.1: Résumé des travaux récents sur la classification de champignons

Référence	**Taille du dataset**	**Nbr de classes**	**Prétraitement /Augmentation**	**Technique /Modèle**	**Résultats**
Maurya et Singh (2024) [24]	Non précisé	2	LBP, GLCM	SVM, KNN, arbre de décision, régression logistique, Random Forest	SVM : précision 0.766
Ottom et al. (2019) [25]	120 images	2	Eigen features, dimensions réelles/virtuelles, histogrammes	KNN, SVM, arbres de décision, réseaux de neurones	KNN : précision 0.944
Zhang et al. (2022) [43]	450	2	Redimensionnement, recadrage, rotation, retournement, normalisation	AlexNet, VGG16, DenseNet121, ResNet50 (pré-entraînés)	ResNet50 : précision 0.756, F1-score 0.766
Demirel et al. (2023) [2]	5470	4	Redimensionnement, rotation, zoom, translation	MobileNetV2-GAP-flatten-fc	Précision 0.977, Recall 0.979, F1-score 0.978
Ketwongsa et al. (2022) [18]	2000	2	Augmentation classique	AlexNet modifié avec module Inception	Précision classification 0.985, détection 0.955
Mahanta et al. (2025) [23]	2688	2	Segmentation U2Net, érosion, dilatation, rotation, variation de luminosité	InceptionV3 pré-entraîné	Précision comestible 0.625, toxique 0.854
Zhu et al. (2023) [44]	10,991	3	Segmentation, rotation, translation, flip, variations de luminosité	MobileNetV3-Large avec SE amélioré	Précision 0.9991

2.2.6 Discussion:

L'analyse du tableau met en évidence deux tendances principales dans les travaux récents sur la classification des champignons. D'une part, les approches dites **classiques**, représentées par Maurya et Singh (2024) et Ottom et al. (2019), reposent sur l'extraction manuelle de descripteurs (LBP, GLCM, Eigen features, dimensions morphologiques) combinés à des classifieurs tels que le SVM ou le KNN. Ces méthodes offrent des résultats intéressants (jusqu'à 94,4 % de précision avec le KNN), mais restent limitées en termes de capacité de généralisation et fortement dépendantes du choix des descripteurs.

D'autre part, les méthodes basées sur le **deep learning** exploitent des réseaux convolutifs capables d'apprendre automatiquement des représentations discriminantes à partir des images. Certaines études se concentrent sur l'adaptation ou l'amélioration d'architectures existantes, comme l'AlexNet modifié de Ketwongsa et al. (2022) ou le MobileNetV3 amélioré de Zhu et al. (2023), tandis que d'autres évaluent des modèles pré-entraînés tels que ResNet50 ou InceptionV3 (Zhang et al., 2022 ; Mahanta et al., 2025). Les meilleures performances sont obtenues par les approches améliorées (jusqu'à 99,91 % pour Zhu et al., 2023), mais souvent sur des jeux de données restreints ou spécifiques à une seule espèce, ce qui limite leur applicabilité générale.

Par ailleurs, l'intégration d'une étape de **segmentation** en prétraitement (Mahanta et al., 2025 ; Zhu et al., 2023) apparaît comme une stratégie prometteuse pour isoler les champignons des arrière-plans complexes et renforcer la pertinence des caractéristiques extraites.

Enfin, il est important de noter que la majorité des travaux existants se contentent d'entraîner les modèles sur des jeux de données de taille réduite, souvent centrés sur un nombre limité d'espèces de champignons. Cette restriction empêche une réelle évaluation de la capacité de généralisation des approches proposées et limite leur applicabilité en conditions variées. La constitution de bases de données plus volumineuses et couvrant une plus grande diversité d'espèces apparaît donc comme une étape essentielle pour progresser vers des systèmes de classification automatique plus robustes et utilisables dans des contextes réels.

Ces constats motivent la démarche présentée dans le chapitre suivant, où nous proposons une méthodologie combinant segmentation, extraction de caractéristiques et classification, afin de développer un système plus robuste et performant pour la classification automatique des champignons.

3. Approche proposée

Dans ce chapitre, nous présentons l'approche méthodologique développée pour traiter la problématique de classification des champignons comestibles et toxiques à partir d'images capturées en milieu naturel. L'objectif est d'isoler le champignon de son environnement grâce à une étape de segmentation, puis de le classifier selon sa catégorie.

Ce chapitre s'articule autour de plusieurs sections : nous présentons dans un premier temps l'architecture globale du modèle, organisée en trois étapes principales, à savoir la segmentation, l'extraction de caractéristiques et la classification. Ensuite, nous décrivons le processus de constitution du dataset ainsi que les étapes de prétraitement appliquées aux images. Enfin, nous détaillons l'implémentation, la stratégie d'entraînement et les critères d'évaluation permettant de mesurer la performance du système proposé.

3.1 Modèle proposée

3.1.1 Vue d'ensemble de l'approche

L'approche proposée vise à réaliser une classification automatique des champignons en trois classes : **comestible**, **toxique** et **mortelle**. Elle repose sur une architecture hybride combinant à la fois des méthodes de *segmentation par réseaux de neurones convolutionnels (CNN)* et des *descripteurs de caractéristiques hétérogènes* (profonds et manuellement extraits).

L'architecture se décompose en trois étapes principales :

1. **Segmentation de l'objet d'intérêt (le champignon)** à l'aide d'un modèle U-Net avec MobileNetV3 comme *backbone*.

2. **Extraction et réduction de caractéristiques** à partir de l'image segmentée, en exploitant à la fois des méthodes d'apprentissage profond et des descripteurs classiques de texture et de couleur.

3. **Fusion des vecteurs de caractéristiques et classification** par un classifieur SVM.

Étape 1 : Segmentation du champignon

Afin de supprimer l'influence des arrière-plans complexes et non pertinents, nous avons choisi d'intégrer une étape de *prétraitement par segmentation*. Pour ce faire, nous avons adopté une architecture **U-Net** dont l'encodeur est basé sur **MobileNetV3**, ce qui limite le risque de surapprentissage et assure de bonnes performances même lorsque la taille de la base d'apprentissage reste modérée. De plus, afin d'améliorer la capacité de localisation et de mieux préserver les contours du champignon, nous avons intégré un mécanisme d'**attention par coordonnées (Coordinate Attention)** dans la branche de décodage du U-Net.

- L'entrée est l'image brute du champignon.
- La sortie est un masque binaire dans lequel le champignon est préservé, tandis que l'arrière-plan est mis en noir.
- L'image segmentée sert alors de base pour l'extraction de caractéristiques.

Étape 2 : Extraction et réduction des caractéristiques

À partir de l'image segmentée, plusieurs types de caractéristiques complémentaires ont été extraites, afin de capturer différentes informations (forme, texture, couleur, apparence globale). :

- **Caractéristiques profondes** : obtenues à l'aide du modèle **InceptionV3** appliqué sur l'image segmentée.Avant l'extraction, une étape de **prétraitement par égalisation d'histogramme** a été réalisée sur le canal **Y** de l'espace **YCbCr**, afin d'améliorer le contraste sans altérer les composantes de chrominance. Les activations issues de la couche *Global Average Pooling (GAP)* sont ensuite utilisées comme vecteur de caractéristiques.
- **Caractéristiques de texture et de couleur** : Avant l'extraction, une étape de **prétraitement par filtrage bilatéral** a été appliquée sur l'ensemble des images, afin de **réduire le bruit** tout en **préservant les contours et les détails visuels**. Ce choix permet d'améliorer la qualité des données d'entrée sans altérer les informations discriminantes nécessaires à l'extraction des caractéristiques.
  - **LBP rotation-invariant (LBP ror)** pour capturer les motifs de texture locaux robustes aux rotations.
  - **GLCM (Gray Level Co-occurrence Matrix)** pour décrire la distribution spatiale des intensités de gris.
  - **Color moments** (moyenne, écart-type, asymétrie) calculés dans l'espace RGB.
  - **Histogramme HSV** : distribution de la *teinte*, de la *saturation* et de la *luminosité*.

Étant donné que chaque méthode génère des vecteurs de dimensions différentes, une étape de **réduction de dimensionnalité par Analyse en Composantes Principales (PCA)** a été appliquée **séparément sur chaque vecteur**. Cette étape permet de conserver l'essentiel de l'information tout en réduisant la redondance et le coût computationnel.

Étape 3 : Fusion et classification

Après réduction, les différents vecteurs de caractéristiques sont **concaténés** pour former un vecteur global représentant chaque image segmentée. Ce vecteur est ensuite introduit dans un **classifieur SVM (Support Vector Machine)**, choisi pour sa capacité à traiter efficacement des données de haute dimension et à généraliser même avec des bases de données de taille modérée.

Le classifieur SVM produit en sortie une prédiction appartenant à l'une des trois classes : **comestible**, **toxique**, **mortelle**.

Schéma récapitulatif de l'approche

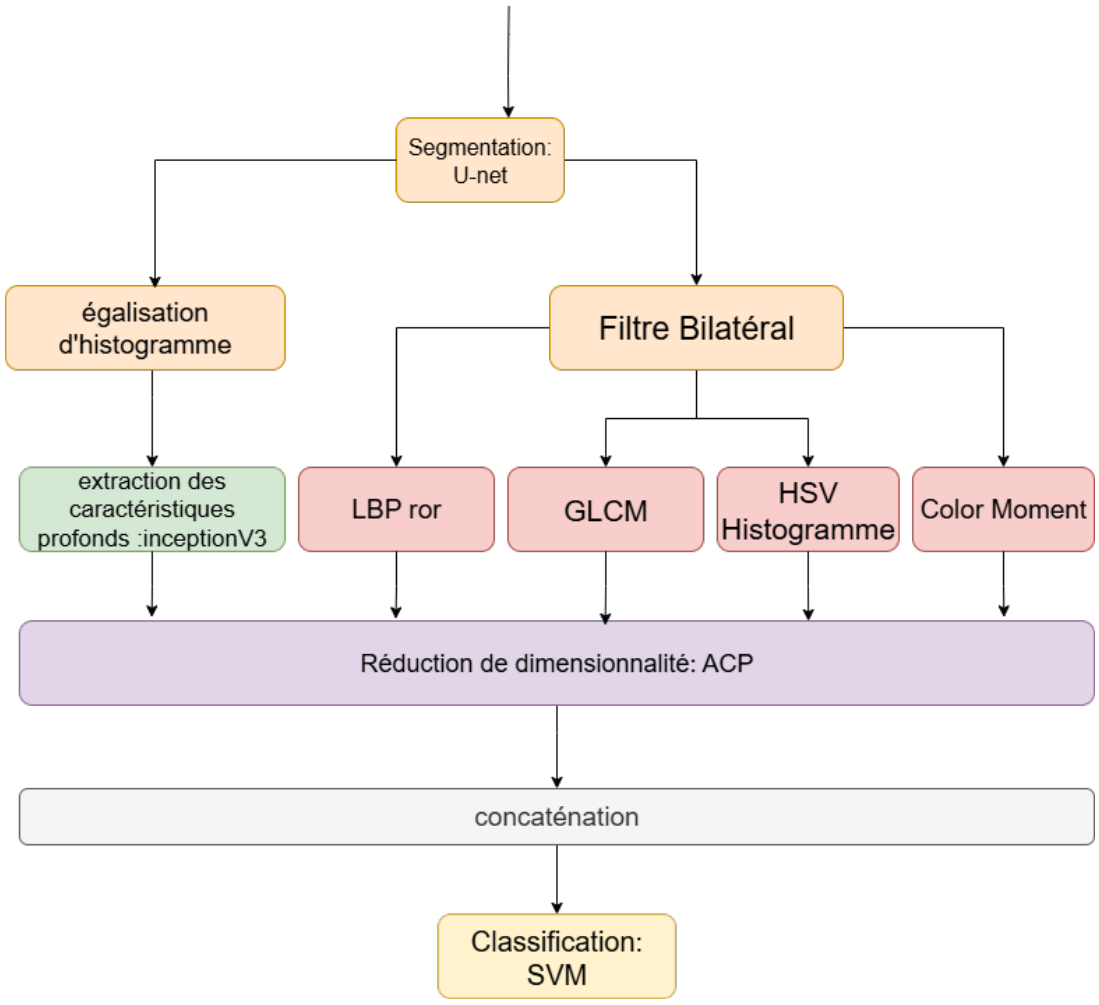


Figure 3.1: Schéma global de l'approche proposée.

3.1.2 Description détaillée des composants de l'architecture

Dans cette section, nous décrivons en détail chacun des modules constituant l'approche proposée.

3.1.2.1 Modèle de Segmentation :

Le module de segmentation repose sur une architecture **U-Net**, dont l'encodeur est basé sur **MobileNetV3-Large**[13]. Afin d'améliorer la capacité de localisation tout en conservant une complexité réduite, nous avons intégré le mécanisme d'**attention par coordonnées (Coordinate Attention)** dans la branche de décodage.

1.1 MobileNetV3-Large: MobileNetV3-Large[13] est une architecture optimisée pour les systèmes à ressources limitées, tout en maintenant de bonnes performances sur des tâches de classification et de détection. Elle repose sur plusieurs **composants clés** qui travaillent ensemble pour assurer une efficacité maximale :

- **Inverted Residuals avec Linear Bottlenecks et Depthwise Separable Convolutions** : pour réduire la complexité tout en conservant la capacité d'extraction de caractéristiques.
- **Squeeze-and-Excitation (SE) Blocks** : pour appliquer une attention adaptative sur les canaux.
- **Fonction d'activation h-swish** : pour améliorer la non-linéarité avec un faible coût computationnel.
- **Recherche d'Architecture Automatique (NAS)** : pour optimiser automatiquement la combinaison des composants.

Ces éléments sont organisés en blocs au sein du réseau afin de maximiser l'efficacité du traitement, tout en préservant la précision sur des jeux de données visuels complexes.

Composants détaillés :

Inverted Residuals avec Linear Bottleneck : Le cœur de l'architecture MobileNetV3 repose sur des blocs appelés **Inverted Residuals** avec **Linear Bottleneck**. Chaque bloc suit une structure en trois étapes :

1. **Expansion** : une convolution 1×1 augmente la dimension des canaux d'entrée.
2. **Depthwise Separable Convolution** : l'opération spatiale est appliquée indépendamment sur chaque canal via une *depthwise convolution*, puis recombinée avec une *pointwise convolution* (1×1). Ce mécanisme réduit considérablement le nombre de paramètres et le coût de calcul, tout en maintenant de bonnes performances.

3. **Projection** : une seconde convolution 1×1 réduit la dimension pour obtenir la sortie du bloc.

Lorsque les dimensions d'entrée et de sortie sont compatibles, une connexion **résiduelle** est ajoutée pour faciliter la rétropropagation et la stabilité du réseau.

Squeeze-and-Excitation (SE) Blocks : Certains blocs de MobileNetV3 intègrent des modules d'attention appelés **Squeeze-and-Excitation (SE)**. Leur rôle est de recalibrer dynamiquement l'importance de chaque canal, selon le contexte de l'image :

- **Squeeze** : réduction de chaque carte de caractéristiques à un scalaire via une moyenne globale(Global Average Pooling).
- **Excitation** : passage du vecteur obtenu dans deux couches fully connected avec des activations ReLU puis Sigmoïde.
- **Recalibration** : multiplication élément par élément (canal par canal) entre les cartes de caractéristiques et le vecteur de pondération.

Fonction d'activation h-swish : MobileNetV3 adopte une combinaison de deux fonctions d'activation non linéaires :

- **ReLU** : utilisée dans les couches peu profondes pour sa simplicité et rapidité.
- **h-swish (hard-swish)** : utilisée dans les couches plus profondes. Elle est définie comme :

$$\text{h-swish}(x) = x \cdot \frac{\text{ReLU6}(x+3)}{6} \quad \text{où} \quad \text{ReLU6}(x) = \min(\max(0, x), 6)$$

H-swish est plus performante que ReLU tout en étant plus rapide à calculer que swish. Elle permet d'améliorer la précision du modèle avec un coût computationnel minimal.

Neural Architecture Search (NAS) : MobileNetV3 a été en grande partie conçue à l'aide d'un processus automatique appelé **Neural Architecture Search (NAS)**. Ce procédé explore et sélectionne les meilleures configurations d'architecture en fonction de critères comme la précision, le coût de calcul ou la taille du modèle. NAS a permis d'optimiser :

- Le type de blocs à utiliser selon les couches (taille des noyaux, présence de SE, fonction d'activation),
- Le nombre de canaux et les paramètres de stride (sous-échantillonnage),
- La profondeur du réseau.

Synthèse de l'architecture MobileNetV3 est une succession de blocs bottleneck, avec certains intégrant des modules SE et des activations h-swish. La structure typique d'un bloc est :

1. Convolution 1×1 (expansion)
2. Convolution depthwise (3×3 ou 5×5)
3. (optionnel) SE block
4. Convolution 1×1 (projection)
5. (optionnel) connexion résiduelle
6. Activation (ReLU ou h-swish)

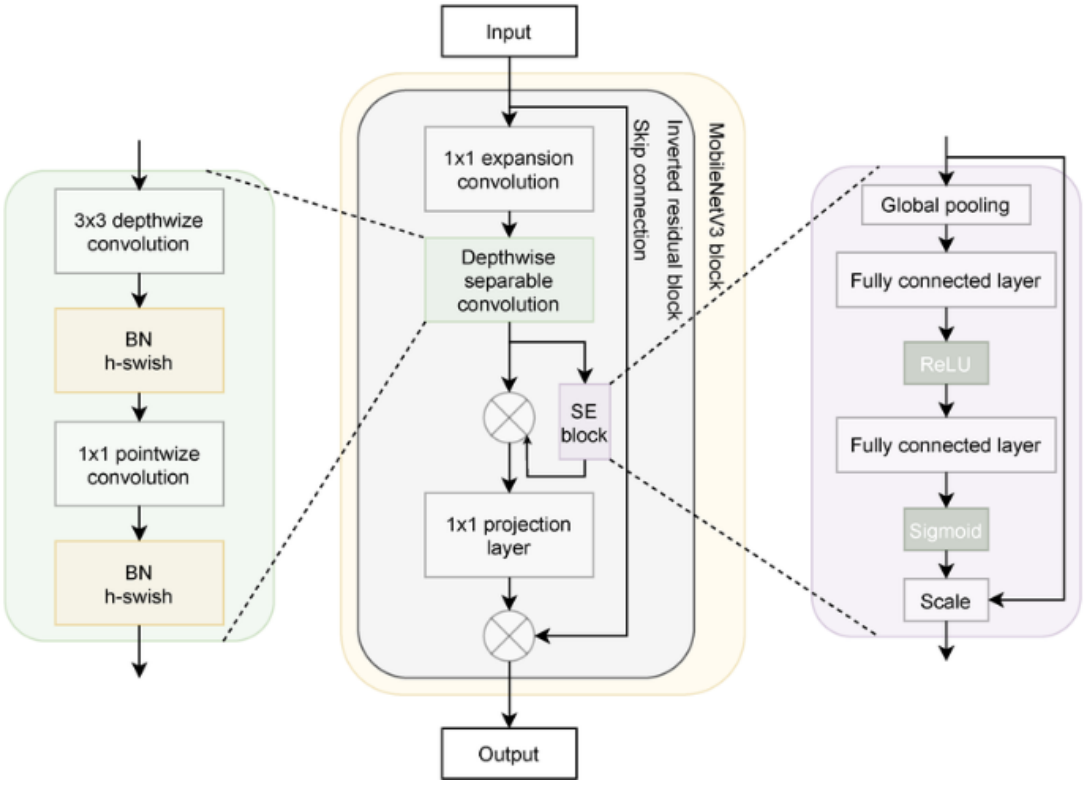


Figure 3.2: Bloc Bottleneck[3]

Ces blocs sont ensuite empilés et organisés en plusieurs **étages hiérarchiques**, avec des paramètres modulés automatiquement par la méthode *Neural Architecture Search (NAS)*. Les variations incluent :

- **La taille des noyaux** de convolution (souvent 3×3 ou 5×5),
- **Les strides** pour moduler la résolution spatiale,
- **L'activation (ReLU ou h-swish)** selon la position dans le réseau,
- **L'usage sélectif des blocs SE** pour renforcer certaines couches critiques.

La structure globale de MobileNetV3-Large peut être schématisée ainsi :

- Une première couche de **convolution classique** pour extraire les premières caractéristiques visuelles (bords, textures),
- Une série de **blocs bottleneck** empilés, chacun optimisé selon les principes ci-dessus,
- Une couche de **global average pooling** pour compacter l'information spatiale,
- Une ou plusieurs couches **fully connected (dense)** en sortie, selon la tâche (classification ou détection).

Le tableau suivant présente la séquence complète des blocs utilisés dans la version Large de MobileNetV3, en incluant la tête finale du réseau ainsi que la couche de pooling global. Il permet de mieux comprendre comment les composants théoriques précédemment décrits s'organisent dans une architecture concrète.

Table 3.1: Configuration des blocs de MobileNetV3-Large (avec tête finale)[13]

Bloc	**Entrée**	**Noyau**	**Canaux de sortie**	**SE**	**Activation**	**Stride**
Conv/BN/HS	224×224×3	3×3	16	Non	Hardswish	2
Bloc 1	112×112×16	3×3	16	Non	ReLU	1
Bloc 2	112×112×16	3×3	24	Non	ReLU	2
Bloc 3	56×56×24	3×3	24	Non	ReLU	1
Bloc 4	56×56×24	5×5	40	Oui	ReLU	2
Bloc 5	28×28×40	5×5	40	Oui	ReLU	1
Bloc 6	28×28×40	5×5	40	Oui	ReLU	1
Bloc 7	28×28×40	3×3	80	Non	Hardswish	2
Bloc 8	14×14×80	3×3	80	Non	Hardswish	1
Bloc 9	14×14×80	3×3	80	Non	Hardswish	1
Bloc 10	14×14×80	3×3	80	Non	Hardswish	1
Bloc 11	14×14×80	3×3	112	Oui	Hardswish	1
Bloc 12	14×14×112	3×3	112	Oui	Hardswish	1
Bloc 13	14×14×112	5×5	160	Oui	Hardswish	2
Bloc 14	7×7×160	5×5	160	Oui	Hardswish	1
Bloc 15	7×7×160	5×5	160	Oui	Hardswish	1
Tête finale	7×7×160	1×1	1280	-	Hardswish	1
pooling	7×7×1280	AvgPool	1280	-	-	-

1.2 Module Coordinate Attention : Le module *Coordinate Attention (CA)*, proposé par Hou et al. [12], a été conçu pour intégrer à la fois des informations de localisation spatiale et des dépendances entre canaux dans les réseaux convolutifs. Contrairement aux mécanismes d'attention classiques tels que le *Squeeze-and-Excitation* (SE), qui se limitent à un recalibrage global des canaux, le module CA introduit une attention directionnelle. Cette approche permet de préserver les coordonnées spatiales tout en capturant des relations contextuelles à longue portée, offrant ainsi à chaque position de l'image un contexte plus informatif et localisé.

Le mécanisme repose sur deux étapes principales :

- **Encodage directionnel** : au lieu d'utiliser un *Global Average Pooling* standard (comme dans SE), CA effectue une agrégation moyenne séparée selon les dimensions spatiales H (hauteur) et W (largeur). Cela produit deux vecteurs distincts : un pour la hauteur (*feature vertical*) et un pour la largeur (*feature horizontal*). Cela permet de conserver les informations de localisation le long d'une direction tout en effectuant une agrégation dans l'autre.

- **Fusion et réaffectation d'attention** : les deux vecteurs directionnels sont concaténés, puis passés à une couche de convolution 1×1 suivie d'une non-linéarité (*h-swish*) et d'une normalisation. Ensuite, deux branches convolutives 1×1 sont utilisées pour générer des cartes d'attention distinctes pour chaque direction. Ces cartes sont ensuite appliquées au tenseur d'entrée pour recalibrer dynamiquement les caractéristiques spatiales.

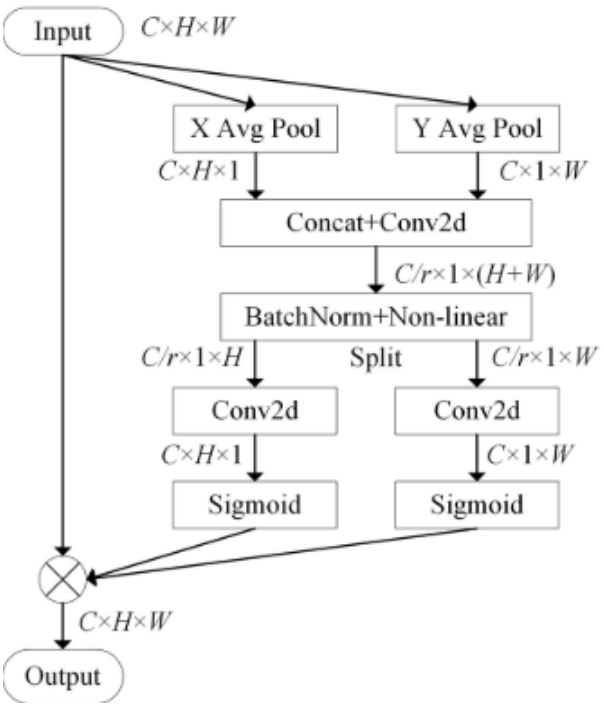


Figure 3.3: Module CA .[12]

En intégrant ce module , le réseau apprend à se concentrer non seulement sur les canaux les plus pertinents, mais aussi à localiser les objets d'intérêt dans l'image avec plus de précision. Dans le contexte de la segmentation de champignons, où les contours peuvent être flous, partiellement occultés ou confondus avec l'arrière-plan naturel (sol, feuilles, bois), le module Coordinate Attention permet d'affiner la délimitation spatiale des objets en combinant efficacement l'information positionnelle et contextuelle. Il contribue ainsi à une meilleure extraction des masques et à une amélioration de la précision de segmentation.

1.3 Intégration dans l'architecture de segmentation : Le backbone de notre modèle repose sur l'architecture *MobileNetV3-Large*, pré-entraînée sur ImageNet. Afin d'exploiter ses représentations hiérarchiques, nous extrayons des cartes de caractéristiques à plusieurs profondeurs.
Ces cartes intermédiaires sont intégrées sous forme de *skip connections* dans le décodeur, ce qui permet de combiner des informations à la fois locales (basse fréquence) et globales (haute fréquence) pour améliorer la précision spatiale.
Concrètement, les couches suivantes sont sélectionnées comme points de saut (*skip connections*) depuis MobileNetV3-Large. Elles correspondent aux sorties de certains blocs *bottleneck* spécifiques :

- `conv` : sortie initiale (après la première convolution), contenant des caractéristiques bas niveau (bords, textures simples).
- `expanded_conv_2_add` : sortie du **2ème bloc bottleneck**, correspondant aux premiers motifs et formes localisées.
- `expanded_conv_4_add` : sortie du **4ème bloc bottleneck**, représentant des structures intermédiaires.
- `expanded_conv_7_add` : sortie du **7ème bloc bottleneck**, capturant des contextes plus larges et complexes.

Ces cartes sont injectées dans le décodeur après redimensionnement spatial pour correspondre aux dimensions courantes. De plus, la sortie `expanded_conv_12_project_bn` (issue du **12ème bloc bottleneck**) est utilisée comme point de départ du décodeur.

Le Décodeur : La décodeur suit une structure inspirée de l'architecture **U-Net**. il est composé d'une succession de **quatre blocs** `Decoder`, chacun comprenant :

- un sur-échantillonnage par interpolation bilinéaire ;
- une concaténation avec la *skip connection* correspondante ;
- deux convolutions séparables en profondeur ;
- un module *Coordinate Attention* pour améliorer la localisation spatiale.

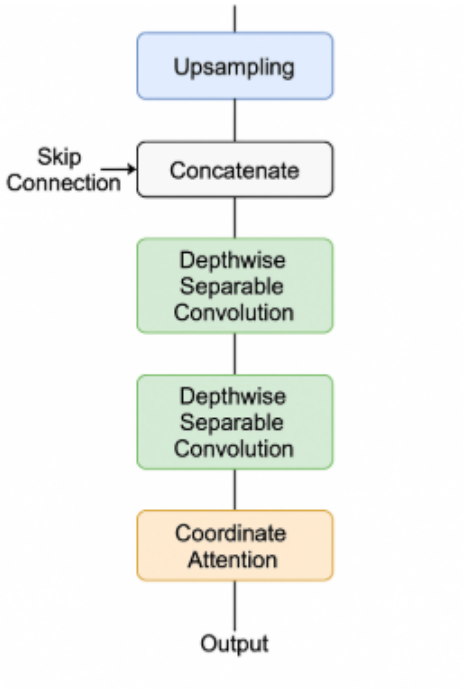


Figure 3.4: Bloc `Decoder`.

La sortie finale est une **carte de segmentation binaire** de même taille que l'image d'entrée, obtenue via un suréchantillonnage, une convolution 1×1 suivie d'une activation *sigmoïde*.
Cette carte de segmentation est ensuite utilisée comme *masque binaire* afin de reconstruire une image dans laquelle seul le champignon est conservé, tandis que l'arrière-plan est remplacé par du noir.

Afin de limiter le coût computationnel et réduire le risque de surapprentissage, seules les **75 dernières couches du backbone** (à partir du $10^{\text{ème}}$ bloc) sont rendues entraînables. Les premières couches restent gelées afin de préserver les connaissances préapprises sur ImageNet.

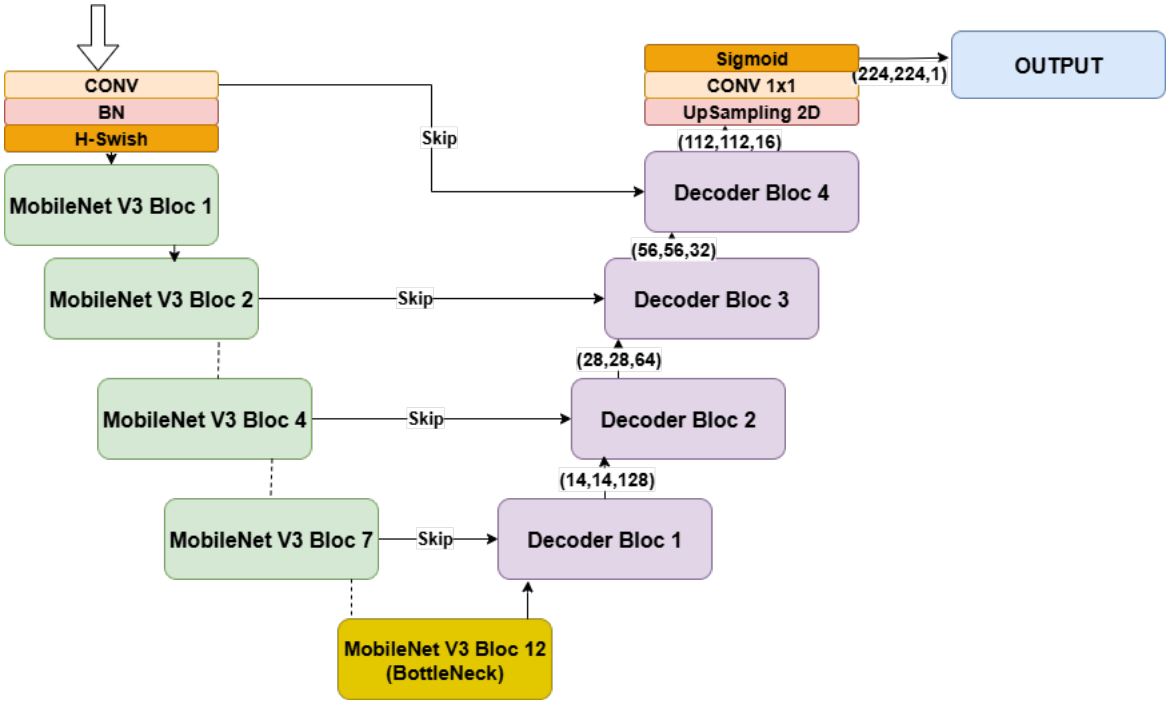


Figure 3.5: schéma du modèle de segmentation.

3.1.3 Extraction de caractéristique profondes :

3.1. Égalisation d'histogramme sur le canal Y de YCbCr

Afin d'améliorer le contraste des images avant l'extraction de caractéristiques profondes, nous avons appliqué une **égalisation d'histogramme** sur la composante de luminance Y de l'espace colorimétrique **YCbCr**. Cette opération agit uniquement sur l'intensité lumineuse des pixels tout en préservant l'information colorimétrique contenue dans Cb et Cr.

Conversion RGB → YCbCr Soit une image en espace RGB, les canaux Y, Cb et Cr sont calculés selon la norme ITU-R BT.601, largement utilisée en traitement vidéo :

$$Y = 0.299R + 0.587G + 0.114B$$

$$Cb = -0.168736R - 0.331264G + 0.5B + 128$$

$$Cr = 0.5R - 0.418688G - 0.081312B + 128$$

où Y représente la luminance (composante de gris), Cb la chrominance bleue et Cr la chrominance rouge.

Égalisation de l'histogramme de Y L'égalisation d'histogramme a pour but de redistribuer les niveaux de gris afin d'obtenir une meilleure répartition des intensités.

1. Calculer l'histogramme $H(Y)$ du canal Y, qui représente le nombre de pixels pour chaque intensité (0 à 255).

2. Construire la fonction de répartition cumulative (CDF) :

$$CDF(k) = \sum_{i=0}^{k} H(i)$$

3. Normaliser la CDF afin d'étaler les niveaux de gris :

$$Y' = \frac{CDF(Y) - CDF_{\min}}{N - CDF_{\min}} \times (L - 1)$$

où N est le nombre total de pixels et $L = 256$ le nombre de niveaux d'intensité.

4. Remplacer Y par Y' afin d'obtenir une image dont la luminance est corrigée et mieux contrastée.

Reconstruction de l'image RGB Après égalisation, les canaux Y', Cb et Cr sont recombinés, puis reconvertis en espace RGB à l'aide des équations inverses :

$$R = Y' + 1.402(Cr - 128)$$

$$G = Y' - 0.344136(Cb - 128) - 0.714136(Cr - 128)$$

$$B = Y' + 1.772(Cb - 128)$$

Ainsi, le contraste global de l'image est renforcé, mais les couleurs d'origine sont préservées.

—

3.2. InceptionV3 :

Pour l'extraction des caractéristiques profondes, nous avons adopté le modèle **InceptionV3** [38], pré-entraîné sur la base de données ImageNet. Cette architecture est conçue pour capturer des informations visuelles à plusieurs échelles tout en limitant le nombre de paramètres.

L'architecture détaillée d'InceptionV3 peut être décrite comme suit :

- **Bloc initial (stem) :** constitué d'une succession de convolutions 3×3, suivies de normalisation par lot (Batch Normalization), de fonctions d'activation ReLU et de couches de sous-échantillonnage (max pooling et convolutions stridées). Ce bloc prépare les représentations de bas niveau en extrayant des motifs simples comme les contours et textures.

- **Modules Inception-A :** trois modules successifs qui appliquent en parallèle plusieurs branches :

  – convolution 1×1 pour la réduction de dimension,

  - convolution 3×3 et factorisations de 5×5 pour capturer des motifs à différentes échelles,
  - pooling (max ou average pooling) pour résumer l'information locale.

  Les sorties de ces branches sont concaténées, produisant une carte multi-échelle.

- **Module de réduction A :** utilisé pour réduire la taille spatiale des cartes ($35 \times 35 \rightarrow 17 \times 17$) tout en augmentant la profondeur (jusqu'à 768 canaux). Il comporte trois branches principales :
  - **Branche 1 :** une convolution 3×3 avec stride 2,
  - **Branche 2 :** une convolution 1×1, suivie d'une convolution 3×3, puis d'une autre convolution 3×3 avec stride 2,
  - **Branche 3 :** un max pooling 3×3 avec stride 2.

  Les sorties sont concaténées pour former une représentation compacte mais plus profonde.

- **Modules Inception-B :** quatre modules consécutifs qui introduisent la **factorisation des grandes convolutions** : une convolution 7×7 est remplacée par deux convolutions asymétriques (1×7 puis 7×1). Cette approche réduit considérablement le nombre de paramètres tout en gardant un champ réceptif large.

- **Module de réduction B :** applique une nouvelle réduction spatiale ($17 \times 17 \rightarrow 8\times8$), tout en augmentant encore la profondeur (jusqu'à 1280 canaux). Il comporte également plusieurs branches :
  - **Branche 1 :** une convolution 1×1, suivie d'une convolution 3×3 avec stride 2,
  - **Branche 2 :** une convolution 1×1, suivie de convolutions factorisées (1×7 et 7×1), puis d'une convolution 3×3 avec stride 2,
  - **Branche 3 :** un max pooling 3×3 avec stride 2.

  Cette combinaison conserve la richesse de l'information tout en réduisant la dimension spatiale.

- **Modules Inception-C :** deux modules finaux appliqués sur des cartes de taille 8×8, produites après le module de réduction B. Leur objectif est de raffiner les représentations de haut niveau en utilisant des convolutions factorisées. Chaque module comporte plusieurs branches en parallèle :
  - **Branche 1 :** une convolution 1×1, qui sert de projection linéaire et réduit la dimension.

- **Branche 2 :** une convolution 1×1, suivie de deux convolutions asymétriques (1×3 puis 3×1), permettant de simuler une convolution 3×3 à moindre coût.
- **Branche 3 :** une convolution 1×1, suivie d'une convolution factorisée en deux branches parallèles :
  - une convolution 1×3,
  - une convolution 3×1.

  Les sorties sont concaténées, ce qui enrichit la diversité des motifs détectés.
- **Branche 4 :** un average pooling 3×3, suivi d'une convolution 1×1 pour maintenir la richesse de l'information contextuelle.

Enfin, les sorties des quatre branches sont concaténées le long de l'axe des canaux, produisant une carte de caractéristiques compacte ($8 \times 8 \times 2048$). C'est cette représentation qui sera résumée par la couche *Global Average Pooling (GAP)* en un vecteur $1 \times 1 \times 2048$.

- **Classifieurs auxiliaires (optionnels) :** placés après certains modules intermédiaires, ils facilitent l'optimisation du réseau et atténuent le problème des gradients évanescents lors de l'entraînement.
- **Bloc final :** à la sortie des Inception-C, une couche de *Global Average Pooling (GAP)* est appliquée afin de résumer chaque carte de caractéristiques par sa moyenne globale. Le résultat est un vecteur de dimension 2048, constituant une représentation compacte et discriminante de l'image.

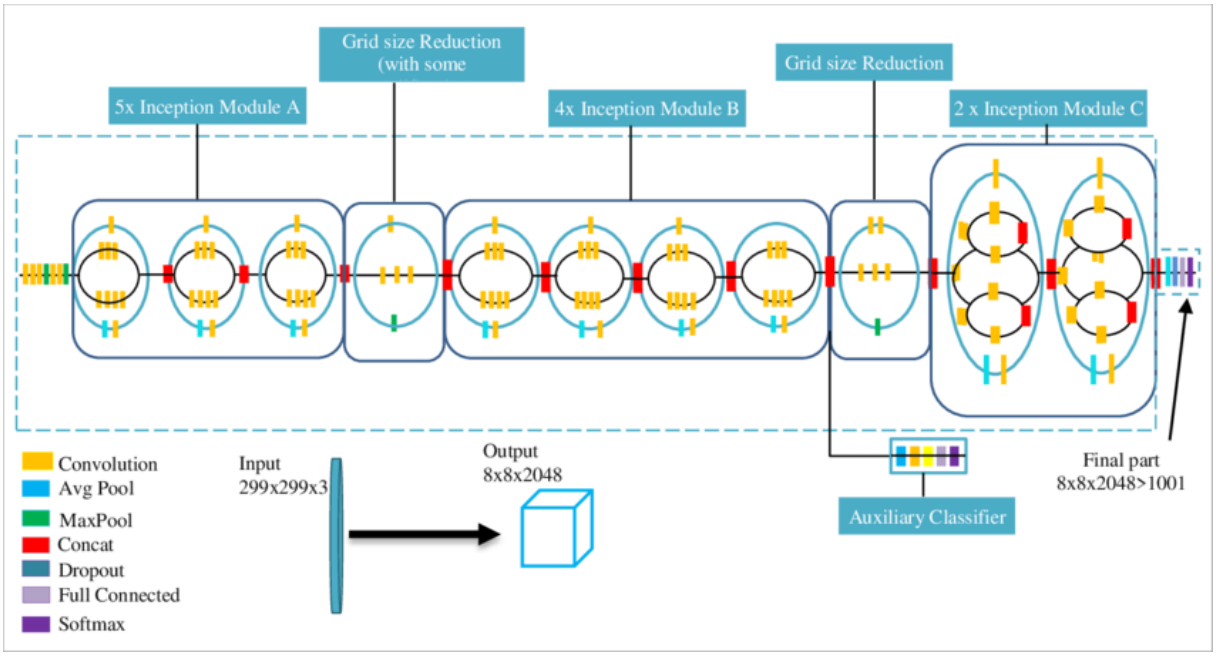


Figure 3.6: Architecture du modèle Inception V3.(avec la couche GAP)

3.1.4 Extraction de caractéristiques manuelles:

4.1. Filtrage bilatéral

Le **filtrage bilatéral**[40] est une technique de lissage non linéaire introduite par Tomasi et Manduchi (1998), qui permet de réduire le bruit tout en préservant les contours de l'image. Contrairement aux filtres de lissage classiques (moyenne, gaussien), il prend en compte à la fois la proximité spatiale des pixels et la similarité de leurs intensités.

Le filtre bilatéral est défini mathématiquement comme suit :

$$I^{\mathrm{bf}}(x) = \frac{1}{W(x)} \sum_{y \in \Omega} I(y) \cdot \exp\left(-\frac{\|x-y\|^2}{2\sigma_s^2}\right) \cdot \exp\left(-\frac{|I(x)-I(y)|^2}{2\sigma_r^2}\right)$$

où :

- $I(x)$ représente l'intensité du pixel central x.
- $I(y)$ représente l'intensité d'un pixel voisin y.
- σ_s contrôle l'influence spatiale (distance entre x et y).
- σ_r contrôle la sensibilité aux différences d'intensité (contraste).
- $W(x)$ est un facteur de normalisation défini par :

$$W(x) = \sum_{y \in \Omega} \exp\left(-\frac{\|x-y\|^2}{2\sigma_s^2}\right) \cdot \exp\left(-\frac{|I(x)-I(y)|^2}{2\sigma_r^2}\right)$$

Ainsi, deux pixels voisins y n'auront une contribution significative à la valeur filtrée de x que s'ils sont proches spatialement ($\|x-y\|$ faible) et similaires en intensité ($|I(x)-I(y)|$ faible).

Avantages pour notre cas d'étude :

- Réduction efficace du bruit sans perte excessive de détails.
- Préservation des contours des champignons.
- Amélioration de la qualité visuelle des images en évitant le sur-lissage.

4.2. LBP-ror (Local Binary Pattern – rotation invariant)

Le descripteur **LBP**[35] est une méthode classique d'analyse de texture. Il repose sur le codage des motifs locaux en comparant l'intensité du pixel central avec celles de ses voisins.

- Pour un voisinage défini par un rayon R et P voisins régulièrement espacés, chaque pixel est associé à un code binaire :

$$LBP_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(I_p - I_c) \cdot 2^p$$

où I_c est l'intensité du pixel central, I_p celle du voisin p, et

$$s(x) = \begin{cases} 1 & \text{si } x \geq 0, \\ 0 & \text{sinon.} \end{cases}$$

- Dans la version **rotation invariant (LBP-ror)**, les codes binaires sont normalisés par rotation circulaire, de sorte qu'un même motif local génère un identifiant unique quelle que soit son orientation.

Exemple : Considérons un pixel central entouré de ses 8 voisins (fenêtre 3×3) :

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & \mathbf{1} & 3 \\ 2 & 5 & 0 \end{bmatrix}$$

– On compare chaque voisin avec le pixel central,on obtient le code binaire :

$$\text{LBP} = 11101000$$

Pour obtenir le code rotation-invariant, on effectue une rotation circulaire et on choisit la représentation minimale :

$$11101000 \ \rightarrow \ 00011101$$

Ainsi, ce motif sera représenté de manière identique quelle que soit son orientation dans l'image.

Ainsi, le descripteur **LBP-ror** génère un **histogramme de motifs de texture invariants à la rotation**, dont la dimension est fixée par le nombre de motifs distincts possibles. Dans notre configuration ($P = 8$, $R = 1$), l'histogramme obtenu est de taille **36**, c'est-à-dire que chaque image segmentée est représentée par un vecteur de caractéristiques à **36 composantes**.

4.3. GLCM (Gray Level Co-occurrence Matrix)

La **matrice de cooccurrence des niveaux de gris** (GLCM)[9] est une méthode statistique qui décrit comment les niveaux de gris d'une image sont distribués en fonction des

relations spatiales entre les pixels. Elle capture des informations de **texture** en mesurant la fréquence à laquelle une paire de niveaux de gris (i, j) apparaît, selon une orientation θ et une distance d données.

Construction de la GLCM :

- Chaque élément $P(i, j)$ de la matrice indique la probabilité qu'un pixel de niveau i soit voisin d'un pixel de niveau j, avec un décalage fixé (d, θ).
- Plusieurs GLCM peuvent être calculées pour différentes orientations (par exemple $0°, 45°, 90°, 135°$) et distances (souvent $d = 1$ pixel).
- Une GLCM est généralement normalisée afin de représenter des probabilités.

Caractéristiques extraites : À partir de la matrice de cooccurrence normalisée $P(i, j)$ (où $i, j \in [0, L-1]$ et L est le nombre de niveaux de gris), on peut calculer plusieurs mesures statistiques décrivant la texture :

- **Contraste (Contrast)** : mesure l'intensité locale et la variation des niveaux de gris.

$$Contrast = \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} (i-j)^2 \, P(i, j)$$

- **Corrélation (Correlation)** : évalue la dépendance linéaire entre les niveaux de gris des pixels voisins.

$$Correlation = \frac{\sum_{i=0}^{L-1} \sum_{j=0}^{L-1} (i-\mu_i)(j-\mu_j) \, P(i, j)}{\sigma_i \sigma_j}$$

  où μ_i, μ_j et σ_i, σ_j sont les moyennes et écarts-types des distributions marginales.

- **Énergie (Energy / Angular Second Moment)** : mesure l'uniformité de la texture (plus elle est élevée, plus la texture est régulière).

$$Energy = \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} P(i, j)^2$$

- **Homogénéité (Homogeneity / Inverse Difference Moment)** : mesure la proximité des valeurs par rapport à la diagonale principale de la GLCM (textures douces).

$$Homogeneity = \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} \frac{P(i, j)}{1 + |i-j|}$$

- **Dissimilarité (Dissimilarity)** : mesure la différence moyenne entre les paires de niveaux de gris.

$$Dissimilarity = \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} |i-j| \, P(i,j)$$

Ainsi, la **GLCM** constitue un descripteur puissant pour quantifier la texture des images. Dans notre configuration expérimentale, les matrices de cooccurrence sont calculées pour plusieurs paramètres : des distances $d = 1, 2, 3$ pixels et des orientations $\theta = 0°, 45°, 90°, 135°$. Pour chaque combinaison (d, θ), cinq propriétés statistiques classiques sont extraites : *contraste*, *dissimilarité*, *homogénéité*, *énergie* et *corrélation*. Ainsi, chaque image est représentée par un vecteur de **60 caractéristiques** $(5 \times 3 \times 4)$, ce qui offre une description texturale riche et complémentaire aux approches basées sur les filtres locaux comme le **LBP**.

4.4. Color Moments

Les moments de couleur[37] constituent un descripteur statistique simple mais efficace pour représenter la distribution des intensités dans une image. Ils reposent sur l'hypothèse que la majorité de l'information relative à la couleur d'une image est contenue dans les premiers moments de cette distribution.

Pour chaque canal de couleur R, G, B on calcule trois moments :

- **La moyenne** : traduit la valeur moyenne d'intensité du canal,

$$\mu = \frac{1}{N} \sum_{i=1}^{N} p_i$$

où p_i désigne l'intensité du pixel i et N le nombre total de pixels.

- **L'écart-type** : mesure la dispersion des intensités par rapport à la moyenne,

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (p_i - \mu)^2}$$

- **L'asymétrie (skewness)** : caractérise la symétrie de la distribution,

$$s = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{p_i - \mu}{\sigma} \right)^3$$

Ainsi, chaque canal est représenté par un vecteur de trois composantes (μ, σ, s). En considérant les trois canaux de l'image, on obtient un descripteur global de dimension 9, sous la forme :

$$\left[\mu_1, \sigma_1, s_1, \mu_2, \sigma_2, s_2, \mu_3, \sigma_3, s_3\right]$$

Ce descripteur compact permet de capturer efficacement la répartition des couleurs, et se révèle particulièrement complémentaire aux descripteurs de texture comme LBP et GLCM.

4.5. Histogramme HSV

L'histogramme de couleurs est un descripteur simple et robuste qui représente la répartition des pixels en fonction de leurs valeurs de couleur. Contrairement à l'espace *RGB*, l'espace **HSV (Hue, Saturation, Value)** sépare la composante chromatique (teinte) des informations d'intensité, ce qui rend l'analyse plus proche de la perception humaine des couleurs.[5]

- **Teinte (H)** : correspond à l'angle de la roue des couleurs (0°–360°), représentant la couleur dominante (rouge, vert, bleu, etc.),
- **Saturation (S)** : indique le degré de pureté ou d'intensité de la couleur (0 = gris, 1 = couleur vive),
- **Valeur (V)** : représente la luminosité ou brillance de la couleur (0 = noir, 1 = maximum de lumière).

Conversion de RGB vers HSV Soit un pixel de l'image représenté en espace RGB normalisé ($R, G, B \in [0, 1]$). On définit :

$$C_{\max} = \max(R, G, B), \quad C_{\min} = \min(R, G, B), \quad \Delta = C_{\max} - C_{\min}.$$

La conversion se fait ainsi :

$$H = \begin{cases} 0, & \Delta = 0 \\ 60^\circ \times \left(\frac{G-B}{\Delta} \bmod 6\right), & C_{\max} = R \\ 60^\circ \times \left(\frac{B-R}{\Delta} + 2\right), & C_{\max} = G \\ 60^\circ \times \left(\frac{R-G}{\Delta} + 4\right), & C_{\max} = B \end{cases}$$

$$S = \begin{cases} 0, & C_{\max} = 0 \\ \frac{\Delta}{C_{\max}}, & C_{\max} \neq 0 \end{cases}$$

$$V = C_{\max}.$$

Ainsi, $H \in [0, 360]$, $S \in [0, 1]$, $V \in [0, 1]$.
(OpenCV utilise une implémentation discrétisée : $H \in [0, 179]$, $S \in [0, 255]$, $V \in [0, 255]$).

Construction de l'histogramme HSV L'histogramme est obtenu en suivant les étapes suivantes :

1. Conversion de l'image RGB en HSV selon les formules ci-dessus,
2. Quantification de chaque canal (H, S, V) en un nombre fixe de classes (*bins*),
3. Comptage du nombre de pixels appartenant à chaque intervalle,
4. Normalisation de l'histogramme pour éliminer l'effet de la taille de l'image.

Dans notre implémentation, chacun des trois canaux est discrétisé en 32 intervalles, ce qui conduit à un vecteur de caractéristiques final de dimension $32 \times 3 = 96$.

3.1.5 Réduction de dimension : ACP

L'**Analyse en Composantes Principales (ACP)**[16] est une méthode statistique utilisée pour réduire la dimensionnalité d'un jeu de données tout en conservant l'essentiel de l'information. Elle permet de transformer un ensemble de variables initialement corrélées en un nouvel espace de variables non corrélées, appelées *composantes principales*.

Principe Étant donné un ensemble de vecteurs de caractéristiques $\mathbf{x}_i \in R^d$, l'ACP consiste à :

1. Centrer les données en soustrayant la moyenne de chaque dimension,
2. Calculer la matrice de covariance C :

$$C = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T$$

   où μ est le vecteur moyen des données,
3. Extraire les valeurs propres et vecteurs propres de C,
4. Ordonner les vecteurs propres selon les valeurs propres décroissantes,
5. Projeter les données sur les k premiers vecteurs propres afin d'obtenir une nouvelle représentation de dimension réduite.

Application dans ce travail L'Analyse en Composantes Principales (ACP) a été appliquée de manière indépendante à chaque ensemble de caractéristiques. Plus précisément, elle a été utilisée sur les descripteurs manuels, à savoir le *LBP-ror*, le *GLCM*, les moments colorimétriques et les histogrammes HSV, ainsi que sur les représentations profondes issues du modèle InceptionV3. Dans ce dernier cas, chaque image est initialement représentée par un vecteur de dimension 2048, extrait de la couche de *Global Average Pooling (GAP)*. Pour l'ensemble de ces descripteurs, l'ACP a permis de réduire la dimensionnalité tout en préservant 95% de la variance, contribuant ainsi à limiter la redondance et à atténuer le risque de surapprentissage.

3.1.6 Modèle de Classification : Support Vector Machine (SVM)

Le **Support Vector Machine (SVM)**[1] est un algorithme d'apprentissage supervisé visant à séparer deux classes à l'aide d'un hyperplan optimal.

Principe du SVM linéaire Étant donné un ensemble d'apprentissage constitué de n exemples (x_i, y_i) avec $x_i \in R^d$ et $y_i \in \{-1, +1\}$, l'objectif du SVM est de trouver un hyperplan défini par un vecteur de poids w et un biais b :

$$f(x) = \text{sign}(w^T x + b)$$

Un hyperplan sépare correctement les données si :

$$y_i(w^T x_i + b) \geq 1 \quad \forall i$$

Le SVM cherche à maximiser la marge $\frac{2}{\|w\|}$ entre les deux classes. Cela conduit au problème d'optimisation suivant :

$$\min_{w,b} \ \frac{1}{2}\|w\|^2$$

sous contraintes :

$$y_i(w^T x_i + b) \geq 1 \quad \forall i$$

Dans le cas où les données ne sont pas parfaitement séparables, des variables d'assouplissement $\xi_i \geq 0$ sont introduites, et le problème devient :

$$\min_{w,b,\xi} \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

sous contraintes :

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i$$

où $C > 0$ est le paramètre de régularisation qui contrôle le compromis entre marge maximale et minimisation des erreurs.

Extension aux SVM non linéaires : le noyau Lorsque les données ne sont pas linéairement séparables dans l'espace d'origine, le SVM utilise une fonction noyau $K(x_i, x_j)$ permettant de projeter implicitement les données dans un espace de dimension plus élevée :

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b\right)$$

où les α_i sont des multiplicateurs de Lagrange associés aux vecteurs de support.

Le noyau RBF :Le noyau **Radial Basis Function (RBF)** est défini par :

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

- Si γ est grand, le noyau devient très localisé : un seul vecteur de support influence fortement la décision, ce qui peut mener à du *surapprentissage*.
- Si γ est petit, l'influence est plus large, et la frontière de décision est plus lisse mais peut manquer de précision.

Résumé des paramètres clés

- C **(régularisation)** : contrôle l'équilibre entre marge large et erreurs de classification.
- γ **(noyau RBF)** : détermine l'étendue de l'influence d'un point d'entraînement.

Choix dans notre expérience: Après avoir appliqué l'ACP indépendamment à chaque type de descripteur et représentations profondes extraites d'InceptionV3, nous avons procédé à la **concaténation** de l'ensemble des caractéristiques réduites. Le vecteur final ainsi obtenu a été utilisé comme entrée du classifieur SVM.

Dans notre travail, nous avons utilisé un SVM avec noyau RBF, paramétré comme suit :

$$C = 10, \quad \gamma = \textit{scale}$$

Le choix de $C = 10$ favorise une classification stricte des données, en pénalisant davantage les erreurs. L'option $\gamma = \textit{scale}$ permet une estimation automatique :

$$\gamma = \frac{1}{d \cdot \mathrm{Var}(X)}$$

où d est la dimension des caractéristiques concaténées et $\mathrm{Var}(X)$ la variance des données, assurant un compromis équilibré entre flexibilité et capacité de généralisation.

3.1.7 Conclusion du système proposé:

En résumé, le système que nous proposons suit une chaîne de traitement structurée : **segmentation** des images pour isoler le champignon, suivie d'une **préparation des images** : **égalisation d'histogramme** appliquée sur la composante Y de l'espace YCbCr pour les images destinées à **InceptionV3**, et **filtrage bilatéral** appliqué aux images utilisées pour les descripteurs classiques afin de réduire le bruit tout en préservant les contours. Ensuite, nous réalisons l'**extraction de caractéristiques** (LBP-ror, GLCM, histogrammes HSV, moments colorimétriques et représentations profondes d'InceptionV3), la **réduction de**

dimension par ACP appliquée séparément à chaque type de descripteur, puis la **concaténation** des vecteurs réduits. Enfin, un **classifieur SVM avec noyau RBF** est utilisé pour la tâche de classification.

Cette méthodologie, qui combine la complémentarité des descripteurs classiques et la puissance des représentations profondes, constitue la base du système que nous allons maintenant mettre en œuvre dans la phase d'**implémentation**.

3.2 Implémentation et évaluation

Dans cette section, nous décrivons les étapes pratiques de mise en œuvre de notre approche. Nous commençons par la construction du dataset à partir d'images collectées, suivie du prétraitement des données. Ensuite, nous présentons l'implémentation du modèle sur la plateforme Kaggle, depuis l'entraînement jusqu'à la prédiction. Enfin, nous évaluons les performances du modèle à l'aide de métriques standards afin de valider son efficacité.

3.2.1 Technologies et outils utilisé:

Kaggle:
Kaggle est une plateforme en ligne développée par Google, dédiée aux projets de science des données, de machine learning et d'intelligence artificielle. Elle fournit aux utilisateurs un environnement cloud gratuit pour exécuter des notebooks Jupyter, avec des ressources matérielles considérables. Chaque session de notebook Kaggle permet d'accéder à :

- **29 Go de RAM** pour le traitement des données en mémoire,
- **57 Go de stockage disque**, dont **20 Go de disque local temporaire** pour les fichiers d'exécution,
- **Accélérateurs matériels** optionnels, tels que des GPU NVIDIA Tesla T4 ou P100, et des TPU (Tensor Processing Units),
- **Accès direct à des datasets publics** et à une API pour importer des jeux de données personnels ou en provenance de la communauté.

Kaggle est donc une plateforme idéale pour entraîner des modèles de deep learning sans disposer localement d'un matériel puissant. Dans notre projet, nous avons utilisé Kaggle à la fois pour collecter des images de champignons depuis des datasets publics, et pour exécuter des scripts de traitement, d'augmentation de données et d'entraînement du modèle.

Albumentations:
Albumentations est une bibliothèque open-source spécialisée dans l'augmentation d'images pour les tâches de vision par ordinateur. Elle offre une grande variété de transformations rapides et flexibles (rotations, flips, bruit, changements de luminosité, etc.), tout en maintenant la compatibilité avec les annotations (masques, bounding boxes). Nous l'avons utilisée pour équilibrer notre dataset en générant artificiellement de nouvelles images à partir des images originales.

rembg:
`rembg` est une bibliothèque Python conçue pour la suppression automatique de l'arrière-plan dans les images. Elle repose sur le modèle U^2-Net, un réseau convolutif profond spécialisé dans la détection d'objets avec précision. En l'utilisant, nous avons pu générer automatiquement des masques binaires pour la tâche de segmentation des champignons, sans avoir recours à une annotation manuelle.

OpenCV:
`OpenCV` (Open Source Computer Vision Library) est une bibliothèque open source spécialisée dans le traitement d'images et la vision par ordinateur. Elle propose un large éventail de fonctions permettant la manipulation d'images, l'extraction de caractéristiques, la détection d'objets ainsi que diverses transformations géométriques et colorimétriques. Grâce à son efficacité et sa compatibilité avec `NumPy`, elle est largement adoptée dans les applications de vision artificielle. Dans notre travail, `OpenCV` a été utilisé pour le prétraitement des images et l'extraction de certains descripteurs (par exemple, les histogrammes de couleurs).

TensorFlow et Keras :
TensorFlow est une bibliothèque open source développée par Google pour le calcul numérique et l'apprentissage automatique. Elle fournit une infrastructure complète pour la création, l'entraînement et le déploiement de modèles de deep learning à grande échelle. TensorFlow est optimisé pour l'exécution sur des environnements hétérogènes (CPU, GPU, TPU) et offre une compatibilité avec de nombreux langages, dont Python, C++ et JavaScript.

Keras, initialement développé comme une interface indépendante, est aujourd'hui intégré nativement à TensorFlow (sous le module `tf.keras`). Il s'agit d'une API de haut niveau conçue pour faciliter la création rapide de modèles de réseaux de neurones. Elle permet de définir des architectures complexes à l'aide d'une syntaxe simple et modulaire, tout en bénéficiant de la puissance et de l'optimisation bas-niveau offertes par TensorFlow.

Dans notre projet, nous avons utilisé `tf.keras` pour implémenter l'ensemble du pipeline d'apprentissage profond : conception du modèle multitâche, définition des fonctions de perte personnalisées, compilation du modèle, entraînement avec suivi des métriques, et exportation pour l'inférence. Cette combinaison de Keras et TensorFlow a permis une

flexibilité maximale dans l'expérimentation tout en garantissant des performances élevées lors de l'exécution sur GPU.

scikit-learn:
`scikit-learn` (souvent abrégé en `sklearn`) est une bibliothèque Python largement utilisée pour l'apprentissage automatique. Elle fournit une vaste collection d'outils pour la classification, la régression, la réduction de dimensionnalité, le clustering et la validation de modèles. Grâce à son interface simple et cohérente, elle permet de mettre en œuvre et de comparer facilement différents algorithmes d'apprentissage supervisé et non supervisé. Dans notre travail, `scikit-learn` a été employée pour la mise en place des modèles de classification et pour l'évaluation de leurs performances.

scikit-image:
`scikit-image` (souvent abrégé en `skimage`) est une bibliothèque Python dédiée au traitement et à l'analyse d'images. Elle propose un large éventail de fonctions pour la transformation, la segmentation, le filtrage, l'extraction de caractéristiques et la mesure de propriétés géométriques. Sa conception repose sur `NumPy` et `SciPy`, ce qui garantit une intégration fluide avec d'autres outils de l'écosystème scientifique Python. Dans notre travail, `scikit-image` a été utilisée pour l'extraction de descripteurs utiles à la classification.

pandas:
`pandas` est une bibliothèque Python dédiée à la manipulation et à l'analyse de données. Elle fournit des structures de données performantes telles que les `DataFrame` et les `Series`, qui facilitent la gestion, le nettoyage et la transformation de grands ensembles de données. Son intégration fluide avec d'autres bibliothèques scientifiques (comme `NumPy` ou `scikit-learn`) en fait un outil incontournable pour le prétraitement et l'organisation des données. Dans notre étude, `pandas` a été utilisé pour structurer les caractéristiques extraites et préparer les jeux de données en vue de l'apprentissage automatique.

3.2.2 Création du dataset

L'un des principaux défis rencontrés dans ce travail est l'absence de jeux de données publics à la fois riches, équilibrés et adaptés à la double tâche de classification et de segmentation de champignons. La plupart des datasets existants se concentrent uniquement sur la classification d'espèces sans fournir de masques de segmentation, ce qui limite considérablement les approches basées sur l'apprentissage multitâche ou la localisation d'objet.

Pour répondre à ce manque, nous avons entrepris la création d'un dataset personnalisé, structuré autour de trois étapes principales : la collecte des images, la génération

automatique des masques de segmentation, et l'augmentation de données pour équilibrer le jeu de données.

Récolte d'images

La démarche a été précédée par une étude bibliographique[17, 8] portant sur les espèces de champignons les plus courantes en Algérie[17, 8], sélectionnées en fonction de leur intérêt culinaire et toxicologique. Douze espèces ont ainsi été retenues, réparties en trois catégories :

- **Espèces toxiques** : *Amanita muscaria*, *Amanita citrina*, *Agaricus xanthodermus*
- **Espèces mortelles** : *Amanita phalloides*, *Amanita verna*, *Galerina marginata*
- **Espèces comestibles** : *Pleurotus ostreatus*, *Morchella esculenta*, *Lactarius deliciosus*, *Cantharellus cibarius*, *Boletus edulis*, *Agaricus bisporus*

Les images utilisées dans notre dataset ont été récoltées à partir de plusieurs jeux de données disponibles sur la plateforme Kaggle, en extrayant celles contenues dans les fichiers correspondant aux douze espèces ciblées. Cette collecte a mis en évidence un déséquilibre important entre le nombre d'images disponibles pour chaque espèce.

Figure 3.7: Exemples d'images pour chacune des douze espèces de champignons sélectionnées.

Génération automatique des masques de segmentation

Concernant la génération des masques de segmentation, les images brutes ont été automatiquement traitées à l'aide de la bibliothèque `rembg`.

Dans notre pipeline, chaque image a été traitée individuellement afin d'en extraire un masque binaire correspondant à la région occupée par le champignon. Le masque résultant est une image binaire où les pixels du champignon sont blancs (valeur 255) et le fond est noir (valeur 0).

Afin d'assurer une correspondance exacte entre chaque image et son masque, ces derniers ont été enregistrés sous le même nom de fichier, dans des dossiers séparés. Le résultat est un dataset structuré, dans lequel chaque paire `(image, masque)` est parfaitement alignée et exploitable par le modèle.

Le dataset est organisé hiérarchiquement : dans chaque catégorie principale (`edible`, `toxic`, `deadly`), on trouve plusieurs sous-dossiers correspondant aux espèces. La même structure est utilisée à la fois pour les images originales et pour les masques binaires générés automatiquement.

Table 3.2: Structure du dataset de champignons

Dossier principal	**Sous-dossier**	**Contenu**
`mushroom_dataset/images/`	`edible/`	Contient un sous-dossier pour chaque espèce comestible : `Pleurotus_ostreatus/`, `Morchella_esculenta/`, `Lactarius_deliciosus/`, etc. Chaque dossier d'espèce contient les images originales.
	`toxic/`	Contient les espèces : `Amanita_muscaria/`, `Amanita_citrina/`, etc.
	`deadly/`	Contient les espèces : `Amanita_phalloides/`, `Galerina_marginata/`, etc.
`mushroom_dataset/masks/`	`edible/`	Même structure que dans `images/`, mais avec les masques binaires correspondant aux images comestibles.
	`toxic/`	Masques pour les espèces toxiques, organisés par espèce.
	`deadly/`	Masques pour les espèces mortelles, organisés par espèce.

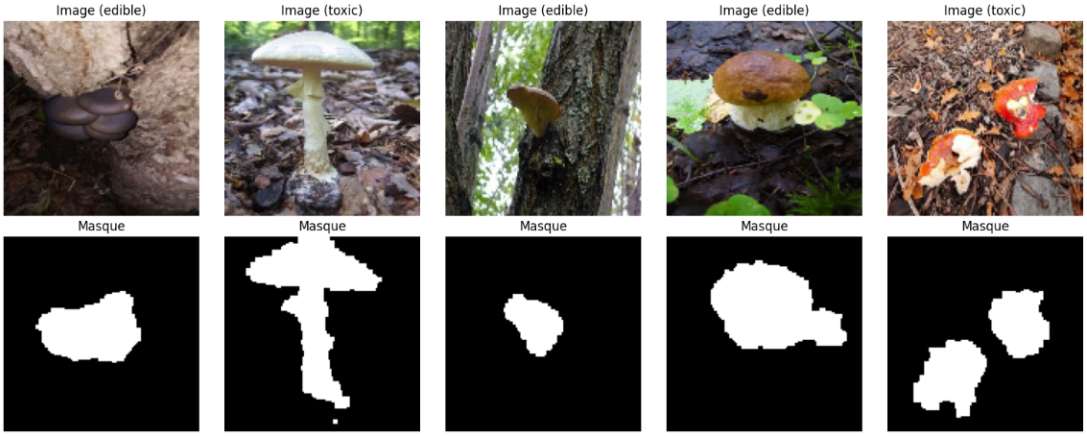


Figure 3.8: Extrait du dataset : images originales et leurs masques binaires associés.

Augmentation de données

Afin de compenser le déséquilibre observé entre les espèces lors de la collecte initiale, nous avons appliqué une stratégie d'augmentation de données (*data augmentation*) adaptée à chaque espèce. À l'aide de la bibliothèque **Albumentations**, nous avons utilisé plusieurs transformations d'image, notamment :

- la symétrie horizontale et verticale (*horizontal flip*, *vertical flip*),

- des rotations de 90°, 180° et 270°,
- des ajustements de luminosité et de contraste,
- un zoom de ±10%,
- l'ajout de bruit gaussien.
- la transformation élastique

Ces transformations ont permis d'enrichir la variabilité du dataset tout en maintenant un équilibre d'environ 800 images par espèce.

Table 3.3: Nombre d'images par espèce avant et après augmentation de données

Espèce	**Nombre initial**	**Nombre après augmentation**
Amanita phalloides	487	779
Amanita verna	111	800
Amanita muscaria	254	800
Amanita citrina	568	800
Galerina marginata	321	781
Agaricus xanthodermus	688	800
Pleurotus ostreatus	329	790
Morchella esculenta	138	800
Lactarius deliciosus	423	790
Cantharellus cibarius	387	800
Boletus edulis	719	800
Agaricus bisporus	233	788
Total	**4 658**	**9 528**

La construction de ce dataset personnalisé a permis de surmonter les limitations des jeux de données publics existants, en fournissant à la fois des images annotées par masques binaires et une répartition équilibrée entre les différentes classes. Grâce à la combinaison d'une collecte ciblée, d'un prétraitement automatisé via `rembg`, et d'une stratégie d'augmentation de données, nous avons obtenu un corpus de 9,528 images structuré, cohérent et adapté aux tâches conjointes de classification et de segmentation. Ce dataset constitue ainsi une base solide pour entraîner efficacement notre modèle.

3.2.3 Construction et Entraînement de notre système :

La réalisation de ce travail repose sur plusieurs étapes successives , ces étapes sont:

1. Entraînement du modèle de segmentation :

L'entraînement du modèle de segmentation constitue une étape fondamentale, car il permet d'extraire les régions pertinentes de l'image en distinguant le champignon de l'arrière-plan. La qualité de cette segmentation conditionne directement les performances de la classification qui suit.

1. **Chargement des données :** Dans cette étape, les images ainsi que leurs masques binaires correspondants ont été chargés à partir des répertoires du dataset. Chaque image a été redimensionnée à une taille fixe de 128×128 pixels, afin de réduire significativement la charge mémoire . Les images et leurs masques ont ensuite été normalisés dans l'intervalle $[0, 1]$. Par ailleurs, chaque image a été associée à une étiquette de classe correspondant à sa catégorie (*comestible : 0, toxique : 1, mortelle : 2*). Cette organisation permet de structurer les données sous forme de triplets (image, masque, étiquette), directement exploitables pour l'entraînement du modèle.

2. **Préparation du dataset TensorFlow :** Le pipeline d'entraînement a été mis en place à l'aide de l'API `tf.data` de TensorFlow, qui permet une gestion efficace et modulaire des données. Lors du chargement, les images et leurs masques sont automatiquement redimensionnés à une taille de 224×224 pixels, garantissant une homogénéité des dimensions en entrée du modèle tout en limitant l'empreinte mémoire. Le jeu de données est ensuite réparti selon un ratio de 80 % pour l'entraînement et 20 % pour la validation. Un *batch size* de 64 est utilisé, accompagné d'opérations de prétraitement, de mise en lot et de préchargement asynchrone. Cette configuration optimise l'utilisation du GPU et assure une scalabilité efficace durant l'entraînement.

3. **Construction du modèle :** La construction a été réalisé à l'aide de bibliothèque TensorFlow (Keras), Les étapes suivie sont les suivantes :

   (a) **Définition des modules personnalisés :**
   Le module *Coordinate Attention* ainsi que le bloc *Decoder* ont été implémentés en tant que couches personnalisées à l'aide de l'API `keras.layers`.

   (b) **Importation de MobileNetV3 :**
   Le backbone **MobileNetV3Large**, pré-entraîné sur *ImageNet*, a été importé via `tf.keras.applications` sans sa tête de classification. Seules les 75 dernières couches ont été rendues entraînables afin d'affiner les représentations. Par ailleurs, les sorties intermédiaires du backbone ont été extraites pour alimenter les Skip connexions nécessaires à la branche de segmentation.

   (c) **Construction de la branche de décodage :**
   La branche de décodage a été conçue à l'aide de quatre blocs *Decoder* empilés, suivis d'une opération d'*upsampling* finale. Une convolution 1×1, combinée à une fonction d'activation sigmoïde, permet ensuite de générer le masque binaire de segmentation.

   Enfin, le modèle complet a été assemblé à l'aide de l'API tf.keras.Model avec une sortie **Mask_output**

4. **Compilation et entraînement du modèle :**
   Pour la segmentation, nous avons adopté la **Dice Loss**, particulièrement efficace dans le cas de classes déséquilibrées. Elle est définie comme suit :

$$\mathcal{L}_{Dice} = 1 - \frac{2\sum_i p_i g_i + \varepsilon}{\sum_i p_i + \sum_i g_i + \varepsilon}$$

   où :

   - p_i représente la prédiction,
   - g_i correspond à la vérité terrain (*ground truth*),
   - $\varepsilon = 10^{-6}$ est un terme de lissage ajouté afin d'éviter la division par zéro.

   Cette formulation maximise le chevauchement entre la prédiction et la vérité terrain, ce qui la rend particulièrement adaptée aux tâches de segmentation présentant un déséquilibre de classes.

   Deux métriques principales ont été utilisées pour l'évaluation :

   (a) **Dice coefficient** :

$$Dice = \frac{2\sum_i p_i g_i + \varepsilon}{\sum_i p_i + \sum_i g_i + \varepsilon}$$

   (b) **Intersection over Union (IoU)** :

$$IoU = \frac{\sum_i p_i g_i}{\sum_i p_i + \sum_i g_i - \sum_i p_i g_i}$$

   L'optimisation a été effectuée à l'aide de l'algorithme **Adam**, avec un taux d'apprentissage initial fixé à 10^{-3}. Deux mécanismes de régularisation ont également été intégrés :

   - **ReduceLROnPlateau** : divise le taux d'apprentissage par 2 en cas de stagnation de la perte de validation (*val-loss*) pendant 6 époques consécutives ;
   - **EarlyStopping** : interrompt l'entraînement lorsqu'aucune amélioration n'est observée durant 10 époques consécutives.

   L'entraînement a été lancé sur 100 époques maximum à l'aide de la fonction **model.fit** en utilisant le jeux de données préparés avec tf.data (batch de taille 64).

2. Création des DataFrames :

Afin de préparer les données pour la phase de classification, nous avons mis en place un pipeline permettant de transformer nos images en un dataframe où chaque ligne représente les caractéristiques d'une image et sa classe. Nous avons suivi ces étapes :

1. **Segmentation des images** : Nous avons importé notre modèle de segmentation préalablement entraîné à l'aide de la fonction `load_model` de `keras.models`. Après

prédiction du masque binaire, celui-ci a été appliqué à l'image en effectuant une simple multiplication entre l'image et le masque, de manière à conserver uniquement la région du champignon et à supprimer l'arrière-plan.

2. **Extraction des caractéristiques** :

   - Pour les caractéristiques profondes, nous avons importé le modèle **InceptionV3** de `keras.applications`. Les images segmentées ont été prétraitées en appliquant une **égalisation d'histogramme** sur le canal Y de l'espace YCbCr à l'aide de `cv2.equalizeHist`, puis redimensionnées et normalisées avec la fonction `preprocess_input`. Les vecteurs de caractéristiques ont été extraits à partir de la couche *Global Average Pooling*.
   - Pour les descripteurs classiques, nous avons appliqué un **filtrage bilatéral** (`cv2.bilateralFilter`) afin de réduire le bruit tout en préservant les contours, puis nous avons exploité différents modules :
     - `skimage.feature` pour calculer les **LBP** (`local_binary_pattern`) et les **GLCM** (`greycomatrix`, `greycoprops`),
     - `cv2.calcHist` de la bibliothèque **OpenCV** pour générer les histogrammes de couleur dans l'espace HSV,
     - les fonctions `mean` et `std` de **NumPy**, ainsi que `skew` de **scipy.stats**, pour calculer respectivement la moyenne, l'écart-type et l'asymétrie des canaux de couleur (moments de couleur).

3. **Transformation en DataFrames** : Les vecteurs de caractéristiques extraits pour chaque descripteur ont été organisés sous forme de `DataFrames` à l'aide de la bibliothèque **pandas**, chaque ligne représentant une image et chaque colonne une caractéristique.

4. **Réduction de dimensionnalité** : Une **Analyse en Composantes Principales (ACP)** a ensuite été appliquée séparément à chaque `DataFrame`, en utilisant la classe `PCA` de `sklearn.decomposition`, afin de réduire la dimension et conserver uniquement les composantes principales les plus représentatives.

5. **Concaténation des DataFrames réduits** : Une fois l'ACP appliquée, les `DataFrames` obtenus pour chaque type de descripteur ont été concaténés afin de constituer une base d'entrée unique pour les modèles de classification. Enfin, une colonne supplémentaire indiquant la *classe* de chaque image (0:comestible, 1:toxique, 2:mortelle.) a été ajoutée au `DataFrame` final, de manière à préparer l'ensemble des données pour l'entraînement et l'évaluation des modèles de classification.

3. Modèle de classification :

Pour la phase de classification, nous avons entraîné un **Support Vector Machine (SVM)** avec noyau gaussien (*RBF kernel*), implémenté via la classe `SVC` de `sklearn.svm`, en fixant les paramètres $C = 10$ et $\gamma = \text{scale}$.

Afin d'évaluer les performances du modèle de manière plus robuste, nous avons adopté une validation croisée à 10 plis (`KFold` avec `n_splits` = 10). L'évaluation a été réalisée à l'aide de la fonction `cross_validate` de `sklearn.model_selection`, permettant d'extraire plusieurs métriques, notamment la précision (**Precision**), le rappel (**Recall**) et le score F_1.

Les formules utilisées pour ces métriques sont les suivantes :

- **Accuracy** : proportion de bonnes prédictions par rapport au nombre total d'échantillons :
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$
- **Précision (Precision)** : proportion de prédictions positives correctes parmi l'ensemble des prédictions positives :
$$\text{Precision} = \frac{TP}{TP + FP}$$
- **Rappel (Recall)** : proportion de prédictions positives correctes parmi l'ensemble des échantillons réellement positifs :
$$\text{Recall} = \frac{TP}{TP + FN}$$
- **Score F1** : moyenne harmonique entre la précision et le rappel :
$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

où TP désigne les vrais positifs, TN les vrais négatifs, FP les faux positifs et FN les faux négatifs.

3.2.4 Résultats et analyse du modèle :

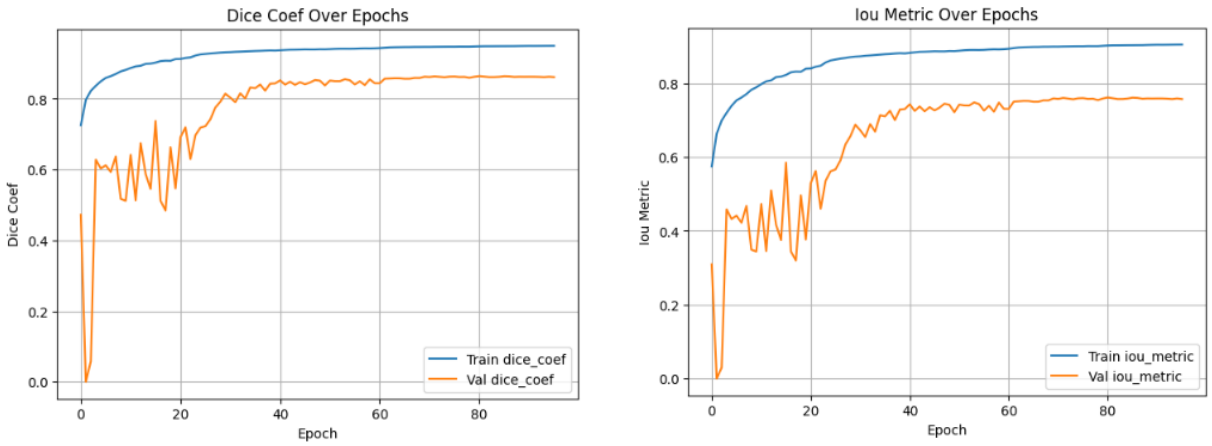
Performance du modèle de segmentation :

Les résultats de validation obtenus pour la segmentation sont les suivants : **Dice coeff = 0,8646** et **IoU = 0,7618**, avec une perte de **Dice loss = 0,1355** (contre Dice = 0,9117, IoU = 0,8378 et Dice loss = 0,0885 pendant l'entraînement).

Ces scores indiquent une très bonne capacité du modèle à délimiter correctement les champignons, même sur des images jamais vues auparavant. La légère baisse de perfor-

mance entre les phases d'entraînement et de validation reste raisonnable, et témoigne d'un modèle bien généralisé, sans surapprentissage excessif.

Ces résultats sont d'ailleurs supérieurs aux seuils de qualité fréquemment rencontrés dans des compétitions Kaggle sur des tâches de segmentation d'objets biologiques [41], où un *Dice* supérieur à 0,80 et un *IoU* au-delà de 0,60 sont généralement considérés comme d'excellentes performances. Cela montre que notre architecture basée sur U-Net avec Coordinate attention remplit efficacement sa fonction de localisation spatiale.



(a) Évolution du Dice coefficient

(b) Évolution de l'IoU

Figure 3.9: Courbes d'évolution des métriques de segmentation sur les ensembles d'entraînement et de validation.

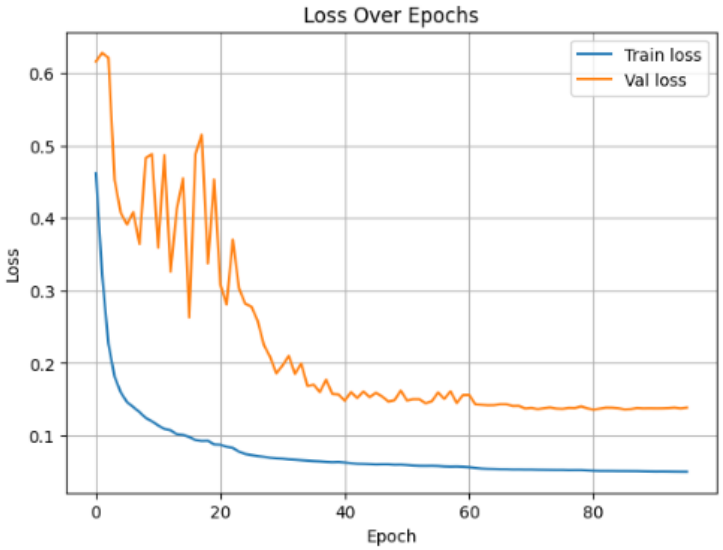


Figure 3.10: Évolution de la fonction de perte (loss) pendant l'entraînement et la validation.

Performance de la classification :

Le tableau suivant présente les résultats de la validation croisée à 10 plis pour le modèle SVM. Les métriques incluent l'*Accuracy*, la *Precision*, le *Recall* et le *F1-score* pour chaque fold, ainsi que les valeurs moyennes en bas du tableau.

Fold / Classe	Accuracy	Précision	Rappel	F1-score
1	0.8688	0.8580	0.8483	0.8525
2	0.8646	0.8630	0.8398	0.8499
3	0.8741	0.8744	0.8516	0.8617
4	0.8447	0.8364	0.8172	0.8252
5	0.8678	0.8674	0.8434	0.8538
6	0.8741	0.8728	0.8511	0.8599
7	0.8583	0.8592	0.8314	0.8431
8	0.8720	0.8645	0.8475	0.8550
9	0.8782	0.8747	0.8552	0.8638
10	0.8866	0.8871	0.8621	0.8730
Moyenne globale	0.8689	0.8657	0.8448	0.8538
Écart-type	0.0108	0.0128	0.0122	0.0123
Classe 0	–	0.8772	0.9415	0.9082
Classe 1	–	0.8504	0.7742	0.8102
Classe 2	–	0.8696	0.8186	0.8430

Table 3.4: Résultats de la validation croisée à 10 plis pour le modèle SVM, avec moyennes globales, écarts-types et métriques par classe.

Les résultats obtenus mettent en évidence une **exactitude globale de 86.89%**, confirmant la capacité du modèle à généraliser efficacement sur de nouvelles données. Plus en détail :

- La classe **comestible** présente le meilleur score global avec un **F1-score = 0.9082**, soutenu par un rappel particulièrement élevé (**Recall = 0.9415**) et une précision de **0.8772**, indiquant que la grande majorité des champignons comestibles ont été correctement identifiés.

- La classe **toxique** obtient un **F1-score = 0.8102**. La précision reste correcte (**0.8504**), mais le rappel plus faible (**0.7742**) suggère que certains champignons toxiques ont été mal classés.

- La classe **mortelle** atteint un **F1-score = 0.8430**, avec une bonne précision (**0.8696**) et un rappel de **0.8186**. Les prédictions sont globalement fiables, mais quelques cas mortels n'ont pas été détectés.

Les moyennes macro globales sont les suivantes :
Précision = 0.8657, **Rappel = 0.8448**, **F1-score = 0.8538**.
Ces valeurs confirment un niveau de performance homogène du modèle, avec une sensibilité particulièrement élevée pour la classe comestible.

Afin d'affiner l'analyse, la **matrice de confusion** est utilisée pour visualiser la répartition des prédictions correctes et erronées. Cette représentation graphique permet d'identifier précisément les types de confusions commises par le modèle entre les différentes classes.

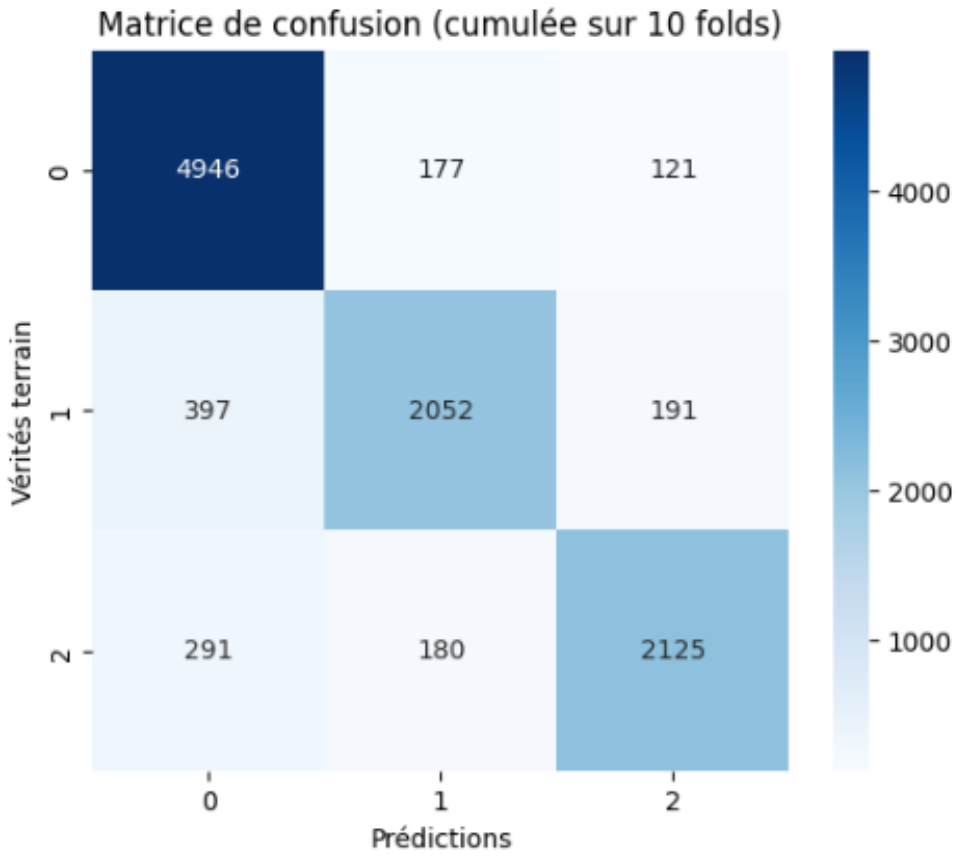


Figure 3.11: Matrice de confusion du modèle de classification

L'analyse de la matrice de confusion met en évidence plusieurs tendances importantes. On observe tout d'abord que la majorité des champignons *comestibles* sont correctement identifiés (**4946 sur 5244**), avec toutefois **177** des confusions vers la classe *toxique* et **121** vers la classe *mortelle*. La classe *toxique* présente des résultats satisfaisants avec **2052 bonnes prédictions sur 2640**, bien que **397** des spécimens aient été classés comme *comestibles* et **191** comme *mortels*. Enfin, la classe *mortelle* est globalement bien reconnue avec **2125 bonnes prédictions sur 2596**, mais **291** des cas sont confondus avec la classe *comestible* et **180** avec la classe *toxique*.

Ces résultats montrent que les confusions les plus critiques concernent les erreurs entre champignons dangereux (*toxiques* ou *mortels*) et la classe *comestible*. Ce phénomène peut s'expliquer par la forte ressemblance morphologique entre certaines espèces. Par exemple, l'*Agaricus bisporus* (comestible) présente de grandes similarités visuelles avec l'*Agaricus xanthodermus* (toxique), ce qui peut induire des confusions. De même, l'*Amanita verna* (mortelle) est proche, d'un point de vue visuel, de l'*Amanita citrina* (toxique).

Il est également important de noter que la classe *comestible* est plus représentée dans le jeu de données, ce qui contribue à ses meilleures performances globales par rapport aux deux autres classes. Par ailleurs, le jeu de données initial comportait un nombre limité d'images par espèce. Pour remédier à cette contrainte, une étape d'**augmentation de données** a été mise en œuvre, permettant d'équilibrer partiellement les classes et d'atteindre environ **800 images par espèce**. Néanmoins, cette stratégie ne peut pas reproduire fidèlement certains types de variabilité naturelle (par exemple, liées aux angles

de prise de vue ou aux stades de croissance), ce qui explique la persistance de certaines confusions.

Un élargissement du dataset par l'acquisition de nouvelles images réelles demeure donc essentiel pour améliorer la robustesse et la généralisation du modèle.

Validation du choix de l'architecture CNN

Dans le but d'évaluer l'impact du choix de l'architecture CNN utilisée comme extracteur de caractéristiques, nous avons testé plusieurs modèles CNN pré-entraînés : **VGG16**, **VGG19**, **ResNet50** et **InceptionV3**. Pour chaque configuration, les descripteurs classiques (LBP, GLCM, histogramme HSV et moments colorimétriques) ont été conservés inchangés, seul l'extracteur de caractéristiques profondes étant remplacé.

Les performances ont été évaluées à l'aide d'une **validation croisée en 10 folds**, et le tableau suivant présente les résultats moyens ainsi que leurs écarts-types (±) en termes de précision, rappel, F1-score et exactitude (*accuracy*).

Modèle	**Précision**	**Rappel**	**F1-score**	**Accuracy**
VGG16	0.8559 ± 0.0129	0.8351 ± 0.0129	0.8440 ± 0.0128	0.8606 ± 0.0122
VGG19	0.8547 ± 0.0154	0.8336 ± 0.0159	0.8425 ± 0.0157	0.8591 ± 0.0130
ResNet50	0.6947 ± 0.0141	0.6494 ± 0.0167	0.6637 ± 0.0162	0.7056 ± 0.0130
InceptionV3	0.8657 ± 0.0128	0.8448 ± 0.0122	0.8538 ± 0.0123	0.8689 ± 0.0108

Table 3.5: Comparaison des performances (moyenne ± écart-type) obtenues par validation croisée 10-fold des modèles CNN utilisés comme extracteurs de caractéristiques profondes

L'analyse met en évidence que les architectures **VGG16** et **VGG19** obtiennent des performances proches, avec des valeurs de précision, rappel et F1-score autour de 0.84–0.85. En revanche, le modèle **ResNet50** se montre nettement moins performant (*F1-score* = 0.66), ce qui suggère une moins bonne adaptation de cette architecture au contexte étudié. Enfin, **InceptionV3** atteint les meilleurs résultats avec une exactitude globale de **86.89%**, confirmant sa pertinence comme extracteur de caractéristiques profondes dans notre approche.

En résumé, l'évaluation comparative montre que, parmi les différents extracteurs de caractéristiques testés, **InceptionV3** se distingue par sa capacité à fournir des représentations plus discriminantes, ce qui en fait le choix le plus adapté pour notre système de classification.

Comparaison avec l'état de l'art

Pour évaluer la pertinence de notre approche, nous avons implémenté 3 méthodes issues de l'état de l'art et les avons réentraînées sur notre dataset, afin de disposer d'une base de comparaison équitable.

Table 3.6: Performances obtenues sur notre dataset avec différentes approches issues de l'état de l'art

Méthode	**Accuracy**	**Précision**	**Recall**	**F1-score**
Notre travail	0.869	0.866	0.845	0.854
Maurya et Singh [24] (LBP+GLCM+SVM)	0.48	0.52	0.48	0.49
Zhu et al [44] (MobileNetV3)	0.79	0.82	0.79	0.80
Demirel et al [2](mobileNetV2)	0.68	0.68	0.66	0.67

L'analyse du tableau 3.6 met en évidence plusieurs constats importants. Tout d'abord, notre approche surpasse nettement les méthodes issues de l'état de l'art, en obtenant une accuracy de 86,9 % et un F1-score de 85,4 %. Cette amélioration peut s'expliquer par l'utilisation conjointe d'un prétraitement adapté, d'une architecture optimisée et d'une stratégie d'entraînement robuste.

À l'inverse, la méthode classique de Maurya et Singh [24], reposant sur des descripteurs manuels (LBP et GLCM) combinés à un SVM, obtient les performances les plus faibles (accuracy de 48 %). Cela illustre les limites des approches basées uniquement sur des descripteurs de texture, qui peinent à capturer la variabilité visuelle des champignons.

La méthode proposée par Zhu et al. [44], intégrant une étape de segmentation suivie d'un MobileNetV3, atteint des résultats intermédiaires (accuracy de 79 %), ce qui confirme l'intérêt d'un pipeline basé sur le deep learning mais souligne également l'importance d'adapter le modèle au dataset utilisé.

Enfin, l'approche de Demirel et al. [2], reposant sur MobileNetV2 avec GAP, Flatten et une couche fully connected, obtient une précision de 68 %. Bien que supérieure à l'approche classique de Maurya et Singh, elle reste inférieure à notre méthode et à celle de Zhu et al., probablement en raison d'un nombre limité de couches denses et d'une architecture moins adaptée à la complexité du problème.

En résumé, ces résultats montrent que les méthodes classiques basées sur descripteurs manuels offrent des performances limitées, tandis que les approches deep learning permettent des améliorations substantielles. Toutefois, la conception de l'architecture et la qualité du prétraitement jouent un rôle déterminant pour atteindre des performances élevées.

3.2.5 Conclusion

Dans ce chapitre, nous avons détaillé la méthodologie proposée pour la détection et la classification des champignons. Celle-ci repose sur une étape de *segmentation* destinée à isoler l'objet d'intérêt, suivie d'une phase d'*extraction de caractéristiques* combinant descripteurs classiques (LBP, GLCM, histogramme HSV, moments colorimétriques) et descripteurs profonds issus de modèles CNN pré-entraînés. Plusieurs architectures ont été évaluées dans ce rôle, et les résultats comparatifs ont montré que **InceptionV3** offre les performances les plus satisfaisantes, confirmant sa pertinence dans le cadre de notre système.

Les expériences réalisées ont mis en évidence la robustesse du système proposé, capable de maintenir un niveau de performance homogène sur l'ensemble des classes. L'intégration conjointe de descripteurs classiques et profonds constitue un atout majeur, permettant de tirer parti à la fois d'informations locales et globales.

Enfin, la comparaison avec des travaux issus de l'état de l'art a montré que notre approche obtient de meilleurs résultats, confirmant ainsi la pertinence de la méthodologie développée. Néanmoins, quelques confusions subsistent principalement au niveau des espèces moins représentées dans le jeu de données, ce qui souligne l'importance d'un équilibre entre classes pour optimiser davantage la précision du modèle.

4. Conclusion

L'objectif principal de ce mémoire était de concevoir un système capable d'assurer à la fois la détection et la classification des champignons à partir d'images naturelles. Pour atteindre cet objectif, nous avons exploré les approches de vision par ordinateur reposant sur l'apprentissage profond, tout en intégrant des descripteurs classiques issus du traitement d'images. L'étude de l'état de l'art a mis en évidence l'émergence de travaux récents exploitant des modules d'attention pour améliorer la reconnaissance visuelle. Cependant, elle a également mis en évidence la rareté des approches capables d'exploiter conjointement la richesse des descripteurs classiques et la puissance des représentations profondes.

C'est à partir de cette observation que nous avons proposé une méthodologie hybride, articulée autour de trois étapes principales : (i) la segmentation des champignons par un modèle U-Net dont l'encodeur est basé sur *MobileNetV3*, et enrichi par un module de *Coordinate Attention* afin d'améliorer la précision de localisation et la capture des détails pertinents, (ii) l'extraction de caractéristiques à la fois classiques (LBP, GLCM, histogramme HSV, moments colorimétriques) et profondes (InceptionV3 et autres architectures CNN pré-entraînées), et (iii) la classification supervisée à l'aide d'un SVM, exploitant la complémentarité entre ces descripteurs.

Les résultats expérimentaux ont montré que l'intégration conjointe de descripteurs classiques et profonds, renforcée par l'étape de segmentation, améliore significativement la robustesse du modèle dans des environnements complexes ou visuellement ambigus. L'évaluation comparative des différents extracteurs de caractéristiques a confirmé la pertinence d'InceptionV3, qui a offert les meilleures performances parmi les architectures testées. De plus, notre approche a obtenu de meilleurs résultats que les trois méthodes de l'état de l'art avec lesquelles elle a été comparée, confirmant l'efficacité de cette méthodologie hybride pour l'identification automatique des champignons.

Ainsi, ce mémoire s'inscrit dans une logique d'intégration entre différentes approches de vision par ordinateur, en proposant une solution modulaire et efficace, reposant sur des composants éprouvés et facilement adaptables. Bien que le modèle ne soit pas strictement léger sur le plan computationnel, sa conception reste relativement simple à mettre en œuvre et à entraîner grâce à la réutilisation d'architectures pré-entraînées et de descripteurs classiques. Ce cadre ouvre également des perspectives intéressantes vers des modèles plus interprétables et directement exploitables dans des applications pratiques, telles que l'assistance à la reconnaissance sur le terrain ou la cueillette sécurisée de champignons.

Références

[1] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[2] Yağmur Demirel and Gözde Demirel. Deep learning based approach for classification of mushrooms. *Gazi University Journal of Science Part A: Engineering and Innovation*, 10(4):487–498, 2023.

[3] Mohamed Elsayed Abd Elaziz, Abdelghani Dahou, Naser Alsaleh, Ammar Elsheikh, Amal Saba, and Mahmoud Ahmadein. Boosting covid-19 image classification using mobilenetv3 and aquila optimizer algorithm. *Entropy*, 23, 10 2021.

[4] S. Giraud. Intoxications fongiques: surveillance et prévention. *Revue Médicale Mycologique*, 12:15–22, 2018.

[5] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, 3rd edition, 2008.

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

[7] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Linwei Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, and Jianbing Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[8] Yamna Hadj Hafsi. Inventaire, cartographie et identification des macromycètes dans la région de m'sila. Master's thesis, Université Mohamed Boudiaf, M'Sila, 2022. Mémoire de Master portant sur les espèces de champignons observées dans les régions semi-arides algériennes.

[9] Robert M Haralick, Karthik Shanmugam, and Its'hak Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(6):610–621, 1973.

[10] D. Hawksworth. The magnitude of fungal diversity: the 1.5 million species estimate revisited. *Mycological Research*, 105:1422–1432, 2001.

[11] 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 (CVPR)*, pages 770–778, 2016.

[12] Qibin Hou, Ming-Ming Zhou, and Jiashi Feng. Coordinate attention for efficient mobile network design. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13713–13722, 2021.

[13] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Vijay Vasudevan, Quoc V Le, and Hartwig Adam. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019.

[14] iNaturalist contributors. Images de champignons issues de la plateforme inaturalist. https://www.inaturalist.org, 2025.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[16] Ian T. Jolliffe. *Principal Component Analysis*. Springer, 2 edition, 2002.

[17] Abdelaziz Keddad and Zouaoui Bouznad. Catalogue des champignons d'algérie. *Académie d'Agriculture de France*, 2018. Ouvrage recensant les principales espèces fongiques présentes en Algérie, incluant des espèces comestibles et toxiques.

[18] Wacharaphol Ketwongsa, Sophon Boonlue, and Urachart Kokaew. A new deep learning model for the classification of poisonous and edible mushrooms based on improved alexnet convolutional neural network. *Applied Sciences*, 12(7):3409, 2022.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] B. Krieger. *Field Guide to Mushrooms of North America*. FalconGuides, 2010.

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

[22] D. et al. Lindner. Intraspecific variability of fungal morphological traits. *Fungal Biology*, 117:346–357, 2013.

[23] Hridoy Jyoti Mahanta and Hillul Chutia. Using deep learning for prediction of edible and poisonous mushrooms. 2025.

[24] S. Maurya and V. Singh. Mushroom classification using feature-based machine learning approach. *Indonesian Journal of Data and Science*, 5(3):199–210, 2024.

[25] Zeinab M. Ottom, Raja Al-Ashraf, Nidaa Alawad, and Raed Nahar. Classification of mushroom fungi using machine learning techniques. *International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE)*, 8(5):2153–2160, 2019.

[26] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[27] Van Phung and Eun Rhee. A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. *Applied Sciences*, 9:4500, 10 2019.

[28] Damian Podareanu, Valeriu Codreanu, Sandra Aigner, Caspar Leeuwen, and Volker Weinberg. Best practice guide - deep learning, 02 2019.

[29] Lutz Prechelt. Early stopping—but when? *Neural Networks: Tricks of the trade*, pages 55–69, 1998.

[30] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[31] Olga et al. Russakovsky. Imagenet large scale visual recognition challenge. Accessed 2025, 2015. https://image-net.org/challenges/LSVRC/.

[32] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[34] S. Smith and D. Read. *Mycorrhizal Symbiosis*. Academic Press, 2010.

[35] Ke-Chen Song, Yun-Hui Yan, Wen-Hui Chen, and Zhang Xu. Research and perspective on local binary pattern. *Acta Automatica Sinica*, 39(6):730–744, 2013.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[37] Michael Stricker and Marco Orengo. Similarity of color images. *Proceedings of SPIE 2420, Storage and Retrieval for Image and Video Databases III*, pages 381–392, 1995.

[38] 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.

[39] J. Taylor and T. Davidson. Poisonous fungi: a global overview. *Toxicon*, 45:345–350, 2005.

[40] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision (ICCV)*, pages 839–846. IEEE, 1998.

[41] Tianqi Wei, Zhi Chen, Xin Yu, Scott Chapman, Paul Melloy, and Zi Huang. PlantSeg: A Large-Scale In-the-Wild Dataset for Plant Disease Segmentation. arXiv preprint arXiv:2409.04038, 2024. Disponible sur arXiv.

[42] Muhamad Yani, S Irawan, and Casi Setianingsih. Application of transfer learning using convolutional neural network method for early detection of terry's nail. *Journal of Physics: Conference Series*, 1201:012052, 05 2019.

[43] Baiming Zhang, Ying Zhao, and Zhixiang Li. Using deep convolutional neural networks to classify poisonous and edible mushrooms found in china. *arXiv preprint arXiv:2210.10351*, 2022.

[44] Fengwu Zhu, Yan Sun, Yuqing Zhang, Weijian Zhang, and Ji Qi. An improved mobilenetv3 mushroom quality classification model using images with complex backgrounds. *Agronomy*, 13(12):2924, 2023.